SISTEMI DISTRIBUITI

DEFINIZIONE
Insieme di sistemi distinti per ***località***
che ***cooperano*** per ottenere risultati ***coordinati***

Esempio di problemi nei sistemi distribuiti

- Due eserciti e decisione di attacco
- in presenza di **generali bizantini**

- messaggio che deve arrivare a destinazione
  con i possibili problemi
- Protocollo di coordinamento tra due processi che devono decidere una azione comune
  (nessuna ipotesi sui guasti)

SISTEMI DISTRIBUITI

NON soluzioni ad-hoc
*MA **ingegnerizzazione (qualità di servizio)***

Necessità di
- teoria di soluzione
- metodologia di soluzione
- standard di soluzione
- verifica e accettazione della soluzione

conoscenza dei sistemi esistenti e delle soluzioni

SISTEMI DISTRIBUITI
(motivazioni tradizionali)

introducono la possibilità di
- accedere a **risorse remote**
- condividere localmente **risorse remote**

SISTEMI di GRANDI DIMENSIONI e con MOLTISSIME RISORSE

POSSIBILITÀ di
- **replicazione** delle risorse
- **bilanciare** uso delle risorse
- tollerare **fallimenti** di risorse
- ***garantire qualità*** *delle operazioni*

DINAMICITÀ del SISTEMA
aggiungere risorse al sistema aperto
*QUALITÀ dei SERVIZI (**QoS**)*
TRASPARENZA della ALLOCAZIONE

COMPLESSITÀ

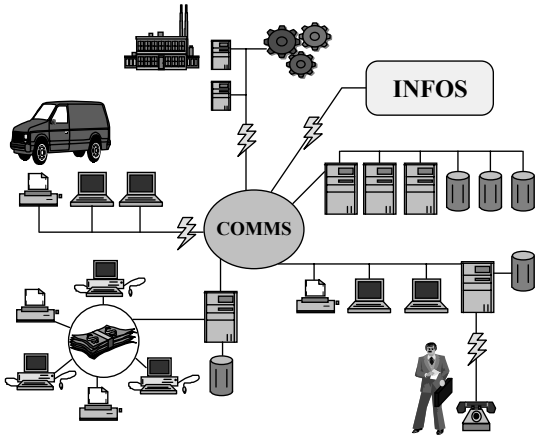
seri problemi teorici e pratici
- **TERMINAZIONE** DEI PROGRAMMI
- **COMPLESSITÀ** DEI PROGRAMMI
- **CORRETTEZZA**
- **EFFICIENZA**

Concorrenza: moltissime attività (processi) possono essere in esecuzione
Nessun tempo globale: nessuna sincronicità degli orologi di un sistema distribuito
Fallimenti indipendenti: molte cause di fallimento, crash di macchine e possibili problemi di rete.

Sistemi Distribuiti

ARCHITETTURE

(Local Area Network LAN - Wide Area Network WAN)



rete fissa composita +
terminali eterogenei e molto differenziati (rete mobile)

REQUISITI tecnologici ed economici

affidabilità

per tollerare guasti **dependability**

condivisione delle risorse

adeguamento alle ***richieste distribuite***
domande distribuite (prenotazioni aeree)
eterogeneità degli accessi

uniformità in crescita e ***scalabilità***
indipendenza dal numero dei nodi del sistema

apertura* del *sistema
capacità di **evoluzione** secondo necessità

ETERONEITÀ E APERTURA

i più comuni sistemi aperti sono i

Sistemi Eterogenei

costituiti da

processori e **interconnessioni**
per la rete fissa

terminali diversissimi
per la rete fissa e mobile

sistemi di supporto (sistemi operativi) diversi
per workstation, per personal,
per dispositivi limitati (LAPtop, PDA, telefonini)

applicazioni e servizi differenziati
forniti a clienti molto diversi

utenti con competenze diversissime
progettisti, esperti, medi, scarsi di conoscenze

proprietà di apertura va oltre

APERTURA di un SISTEMA distribuito

☺ capacità di **adattamento** alle **condizioni** di **stato** successive alla messa in opera senza blocchi

😐 necessità di **identificare** (monitor) e controllare (gestione) lo **stato** del sistema

non esiste un sistema aperto in assoluto, ma solo in relazione a una specificata esigenza

Se il **sistema è aperto** si devono potere cambiare alcune parti o strategie durante la esecuzione
(senza dovere fermare il servizio e ripartire da zero con una nuova esecuzione)

LOCALITÀ DISTINTE

PROBLEMI intrinseci alla distribuzione
creazione e gestione di risorse globali

complessità intrinseca
sistemi più complessi da specificare e risolvere
sicurezza *(security)*
più difficile garantire integrità e correttezza
sbilanciamenti nell'uso delle risorse
necessità di sfruttare in modo giusto le risorse

capacità di esecuzione **totale *limitata***
inferiore a quella di un mainframe equivalente

Legge di Grosh
migliore bilancio costo/performance
con un monoprocessore mainframe
(senza problemi di memoria e I/O)

ovviamente con i limiti alla potenza di calcolo
- velocità della luce
- costi elevati aumentando l'integrazione

NOTA - la **distribuzione delle risorse** è
sempre più
un vincolo di tutte le architetture disponibili
un assunto necessario per le risorse più comuni

AREE DI INTERESSE

nei **settori** applicativi
- previsioni meteorologiche
- dinamica molecolare
- modelli biologici
- evoluzione di sistemi spaziali
- ***sistemi globali (Internet, Web, ...)***
- ***sistemi con garanzie di qualità (Multimedia)***

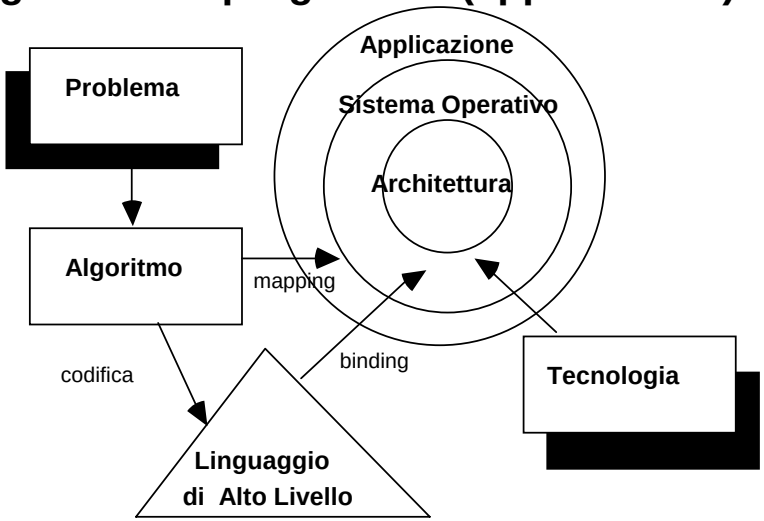
Soluzioni facili (o meno)
- processi indipendenti (con poca comunicazione)
per ottenere speed-up ed efficienza
- calcolo **scientifico ed ingegneristico**
molta computazione
- progetto automatico **VLSI**
ampio spazio delle soluzioni
- operazioni **database**
possibilità di concorrenza
- **i**ntelligenza **a**rtificiale
obiettivi a breve e lungo termine

- sistemi distribuiti ad amplissimo raggio
calcolo algoritmi NP completi
accesso ad informazioni globalmente distribuite

Necessità di applicazioni:
Controllo traffico aereo affidabile
Sistemi **multimediali in rete**
Sistemi **nomadici** e **mobili**

Servizi in sistemi **Web compatibili**

Progetto di un programma (applicazione)



Algoritmo ==> Risoluzione astratta
- codifica in un opportuno **linguaggio di alto livello**

SVILUPPO in uno o più linguaggi

MAPPING - *configurazione*
- decisioni *di allocazione per l'architettura scelta*

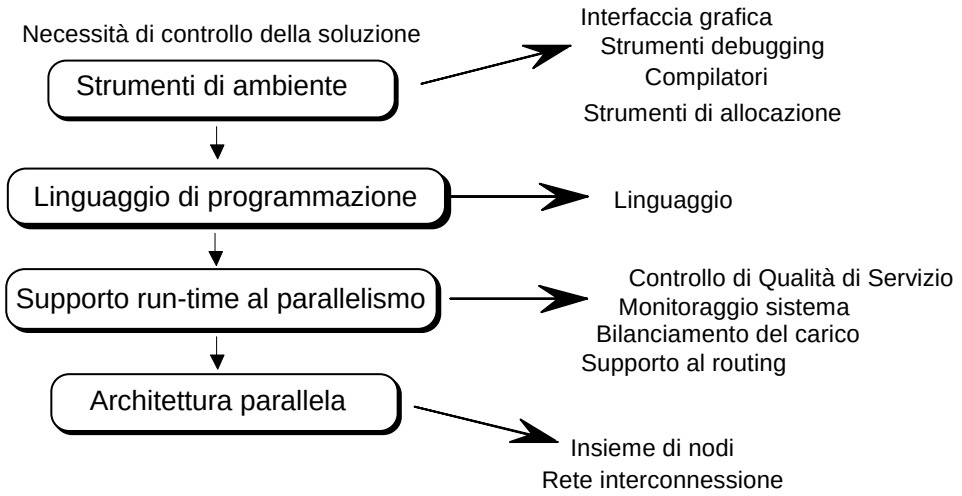
DEPLOYMENT
- decisioni *specifiche* per la *esecuzione sulla architettura*

BINDING
aggancio delle *entità di programma* sulle *risorse di sistema*

decisioni prima della esecuzoine
politiche prefissate e non variabili o adattabili
GESTIONE STATICA

il binding differito alla esecuzione ==> NECESSITÀ di
GESTIONE DINAMICA del **BINDING** e delle **RISORSE**

Ambiente di programmazione



Un ambiente di programmazione tende a fornire soluzione per una ampia fascia di utenti

astrazione
(nascondere complessità parallelismo)

trasparenza e **indipendenza** dalla architettura
(portabilità su nuove architetture)

corretta gestione delle risorse (statica e dinamica)

NON esistono **ancora ambienti di programmazione** soddisfacenti e completamente trasparenti

In ogni caso, **soluzione** ==>
uso diretto di **funzioni dinamiche**, con visibilità della architettura - tipo ***primitive di KERNEL***

problemi di portabilità

MODELLI COMPUTAZIONALI

modello Von Neumann
modello sequenziale con
una sola capacità di esecuzione

modelli di esecuzione
considerando la molteplicità dei flussi di dati e di esecuzione

Single **I**nstruction **M**ultiple **D**ata (**SIMD**)
Multiple **I**nstruction **M**ultiple **D**ata (**MIMD**)

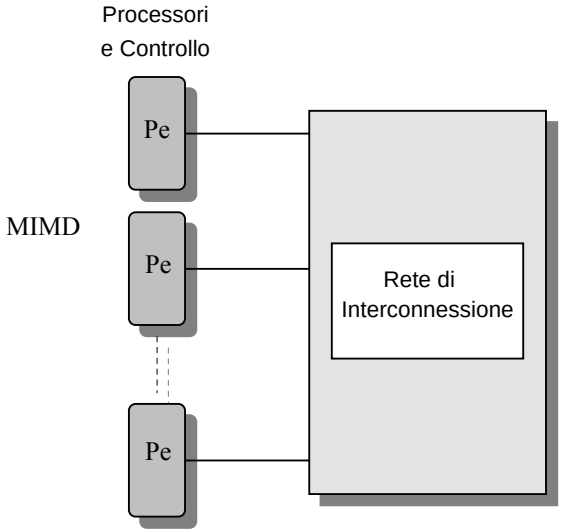
Flynn

data stream	unico flusso di dati	flussi multipli di dati
instruction stream		
unico flusso di istruzioni	SISD (Von Neumann)	SIMD (vettoriali)
flussi multipli di istruzioni	MISD (pipeline ?)	MIMD (macchine multiple)

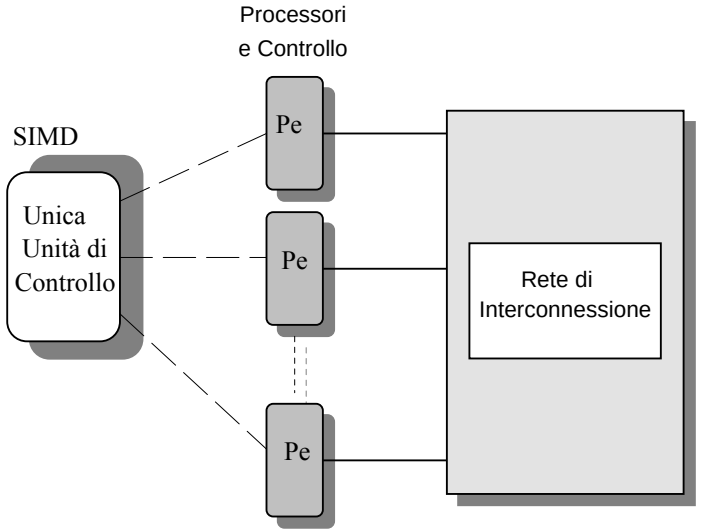
Altri modelli più complessi non sono stati poi largamente accettati

Possiamo distinguere **processori** e **interconnessione**

Tipica architettura MIMD



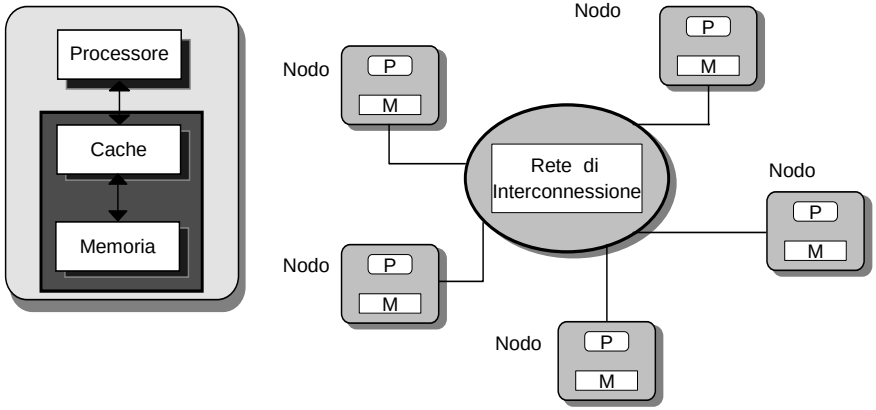
Tipica architettura SIMD



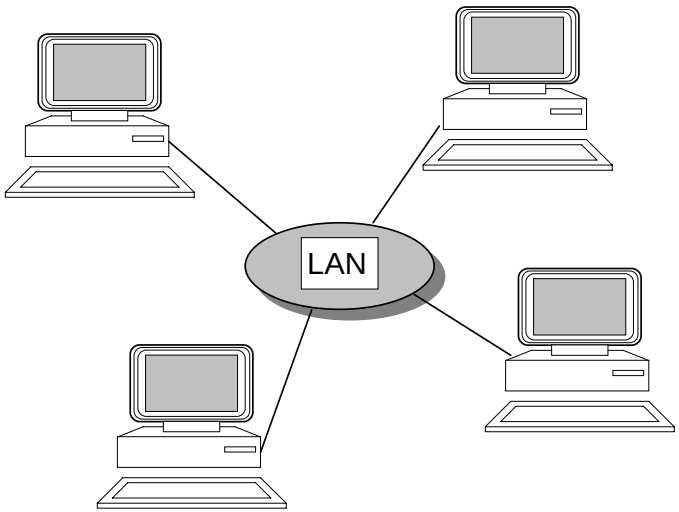
SISTEMI MULTICOMPUTER MIMD

Nodo processore collegato alla memoria *privata*
anche **organizzata a livelli**

Sistema distribuito **Non Uniform Resource Access**

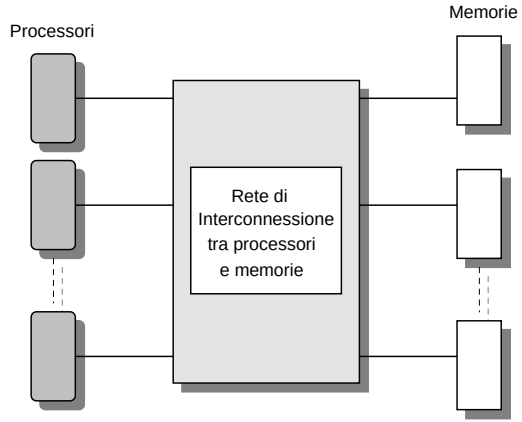


Reti di workstation
Calcolatori indipendenti connessi da una rete locale (LAN)

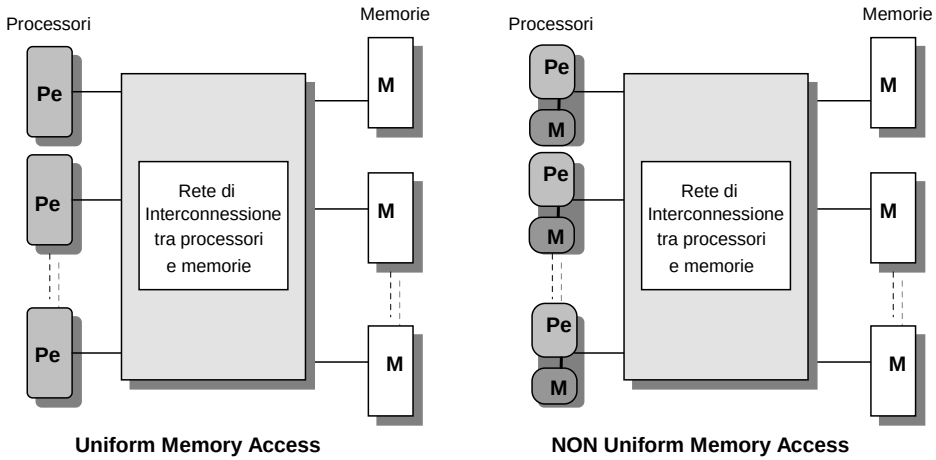


Rete di workstation

INTERCONNESSIONE



Topologia della rete di interconnessione tra processori e memorie con uso di cache



UMA **NUMA**

Idea di località nell'accesso alle risorse
Accesso differenziato alle risorse

ARCHITETTURA ASTRATTA

composta di risorse

processori (**P**rocessing **E**lement)
memoria input/output
canali

Sistema distribuito con memoria e canali

NON UNIFORM RESOURCE ACCESS

ARCHITETTURA ASTRATTA

decomposizione/comunicazione

ARCHITETTURA REALE

Processori - compromesso numero-potenza

No. di processori	Tipi di architetture
10	Mainframe (elevato *throughput*, non speed-up) Architetture a bus Reti di workstation
100	reti di interconnessione dirette od indirette
1000	molti processori (a 1 bit) su singoli chip

Memoria ed I/O

Dimensionamento della memoria ed I/O dipende da:

- la potenza del processore
- il rapporto elaborazione/comunicazione
- la frequenza e tipo di I/O richiesto

***1** Mbyte Memoria e I/O rate **1** Mbyte/sec per MIPS*

INTERCONNESSIONE in base a

Modelli GLOBALI/ RISTRETTI o LOCALI

GRADO di PARALLELISMO

MIMD vs. SIMD

- MIMD più flessibili
- SIMD possono essere emulate con macchine MIMD

Sistemi Paralleli

Sistemi paralleli

allo stato attuale della tecnologia possono essere composti da alcune decine di processori.

Sistemi massicciamente paralleli

caratterizzati da architetture che scalano fino a migliaia di processori (massimo circa 64000)

Sistemi globali

caratterizzati da numeri illimitati di processori (WEB)
milioni di nodi

Sistemi Paralleli

Moderatamente Paralleli decine di PE
numero di processori fino al massimo 30
CRAY, Convex, Sequent, Encore, Alliant FX, ...

Massicciamente Paralleli migliaia di PE
numero di processori da 30 a 64000
iPSC(/2 e /860), NCUBE, MEIKO, SuperNode, ...

Per i primi, anche **memoria condivisa**
Per i secondi, **no memoria condivisa** (solo a cluster)

In sistemi globali, solo comunicazione

ARCHITETTURA

MEMORIA CONDIVISA

con **bus unico**

proprietario o standard (Multibus, VMEbus)

senza **bus**

uso di reti con switch dinamico multistage

o memoria veramente condivisa

SENZA MEMORIA CONDIVISA

gerarchia di bus per scambio messaggi

con gerarchia un cluster di PE connessi con un bus proprietario o standard

reti di interconnessione tra PE con diverse topologie

con possibilità di variare anche la topologia, generalmente regolare

reti di interconnessione tra workstation

in particolare, con opportune gerarchie di interconnessione veloce

PROCESSORI PARALLELI

- CPU ***proprietarie*** progettate appositamente
  CRAY, Convex, IBM 3900, ...
- CPU ***progettate*** per il calcolo parallelo e distribuito, in genere con integrazione con la comunicazione
  T800, T9000, nCUBE, iWARP, ...
- CPU ***'off-the-shelf'***
  RISC, SPARC, ...

2 CATEGORIE PRINCIPALI

A) ARCHITETTURE con **memoria distribuita**
scalabili il giusto

B) SISTEMI ETEROGENEI basati
su reti diverse

Esempi:

Sistemi a Memoria Distribuita A)

Meiko CS ...

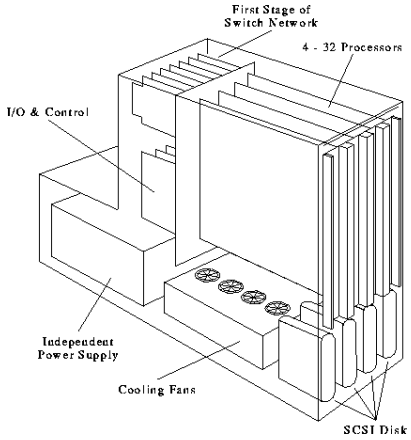
Sistemi Distribuiti di dimensione elevata B)

reti di workstation e multiprocessori

SISTEMI A MEM DISTRIBUITA: MEIKO A)

Meiko CS-1 Fino a 2048 nodi MIMD con processori T800
Topologia statica riconfigurabile (4 link per nodo)
Comunicazione con Message passing proprietario

Meiko CS-2 Struttura modulare, fino a centinaia di nodi MIMD con processori off-the-shelf SPARC
Topologia ad albero di switch

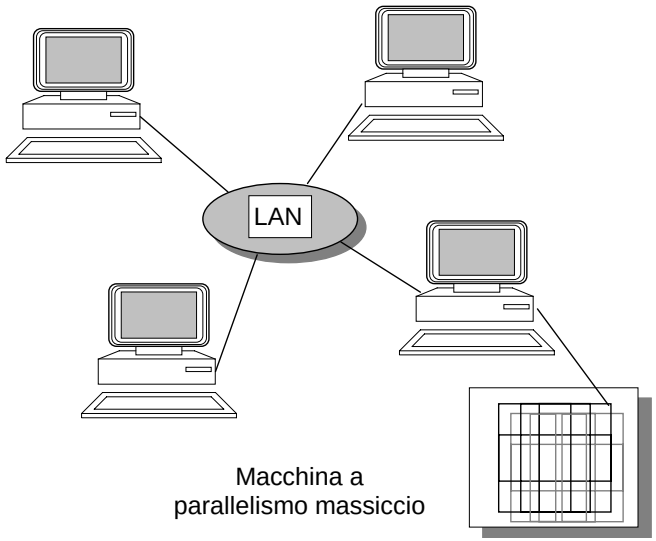


Introduzione di **cluster** di processori

SISTEMI DISTRIBUITI B)

Reti di workstation e multiprocessori
Sistemi eterogenei
- reti
- hardware e software parallelo

Mancanza di sistemi operativi e di strumenti di sviluppo



*Uso di architetture speciali (**MEIKO**) attraverso la rete*

Soluzioni con cluster
introducendo l'idea di località a livello di architettura

Località introdotta con vincoli,
- sia attraverso reti, sia attraverso bus,
- sia attraverso memoria comune

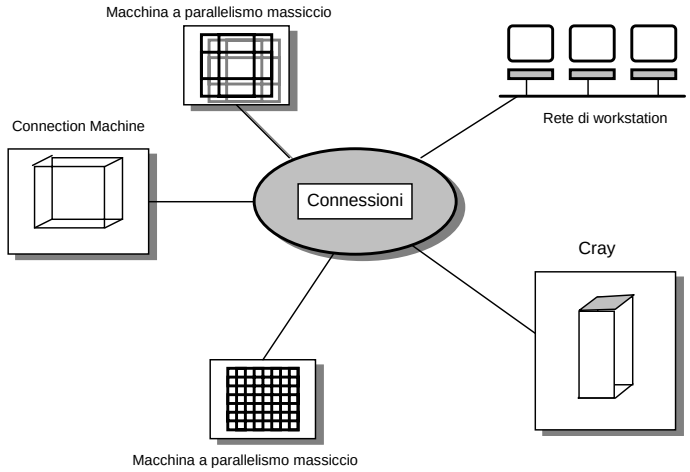
Sistemi Distribuiti Globali
interconnesione aperta di reti con componenti di calcolo eterogenei

COMPUTAZIONE ETEROGENEA
uso di sistemi esistenti
per produrre un **ambiente integrato**
modularità decomposizione e
parallelismo differenziato

Differenze a livello di
- architettura, dati, protocolli, sistemi operativi, risorse

e con necessità di **standard**

Ogni applicazione presenta più tipi di parallelismo
*le parti vengono così eseguite sui componenti più adatti su **esplicita richiesta** o in modo **implicito***



Sistemi globali interconnessi da Internet e WEB con la condivisione di capacità di esecuzione
Web computing

Stato dell'arte

esaminando i **sistemi operativi** ossia i **supporti**

UNIX prevede
- **processi** con spazio di **nomi locali**
- **file** system **condiviso** tra tutti i processi in esecuzione
- **comunicazione** punto a punto (socket)

UNIX rappresenta **il** modello di **conformità**
per caratteristiche di **accesso ai file**
per possibilità di **concorrenza**

Tutti i sistemi operativi **general-purpose** tendono a fornire soluzioni ispirate o analoghe. Si pensi a:

- evoluzioni che fanno ancora i conti con le proprietà di UNIX (vedi anche *Linux*)
- **Microsoft Windows NT** che introduce processi e controllo di accesso

rinforzato da **OPEN SOURCE** e **FREE SOFTWARE**

Limiti di UNIX

limiti dovuti ai **processi pesanti**

I sistemi operativi **evoluti** preservano la compatibilità ma forniscono migliori prestazione per la gestione delle risorse

➪ per i **processi** (leggeri) e per la **distribuzione** delle risorse (processi e dati)

Inoltre, ispirazione agli standard
- **Object-Oriented CORBA**
- **Linguaggio Java**

STATO DELL'ARTE

*Anziché usare **kernel monolitici** (vedi UNIX)*
*e che si accettano **in blocco** (o meno)*
e che pesano sulle performance

Uso di **microkernel**

***realizzazioni minimali** dei meccanismi del S.O. in un **middleware** (sistema interrupt driven e non processo)*
*le **politiche** sono specificate al disopra del microkernel in termini di **processi servitori***
a livello applicativo **utente**
processi applicativi e di sistema

☺ **apertura a nuove strategie** (generalità)
☹ **costi superiori delle strategie realizzate**
(rispetto a soluzione ad-hoc)

*Tipicamente, i microkernel contengono il **supporto per i processi** e per le **comunicazioni tra processi***
Le politiche sono realizzate al disopra in spazio utente
*ad esempio, il **file system**, il **sistema di nomi**, ecc.*

I microkernel non sono sviluppati
solo da ambienti tipo **UNIX**,

ma anche in ambienti tipo **Windows**
in cui l'accento è
sulle interfacce grafiche,
sulla interazione visuale, etc.

In ogni caso, anche gli **approcci proprietari** devono tenere e tengono in conto di questa evoluzione

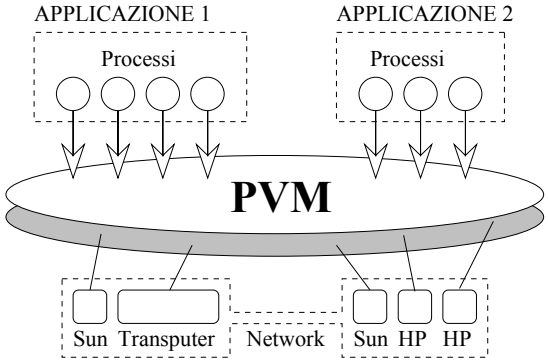
INTEROPERABILITÀ e RIUSO

livelli di ***omogeneizzazione della eterogeneità***

STRATI INDIPENDENTI DAL
SISTEMA OPERATIVO LOCALE ==>
interpreti problemi di efficienza

SUPPORTI UNIFICANTI PER LA
SOLA COMUNICAZIONE TRA PROCESSI
codice nativo + primitive di comunicazione

SISTEMI per la COMUNICAZIONE
PVM (*Parallel Virtual Machine)*



Gli approcci tipo PVM (anche altri: MPI, etc.) sono basati sulla standardizzazione dello scambio di messaggi tra nodi eterogenei

Si mantengono ambienti locali differenziati che sono resi omogenei dalle primitive unificate
(uso di linguaggi diversi e S.O. diversi)

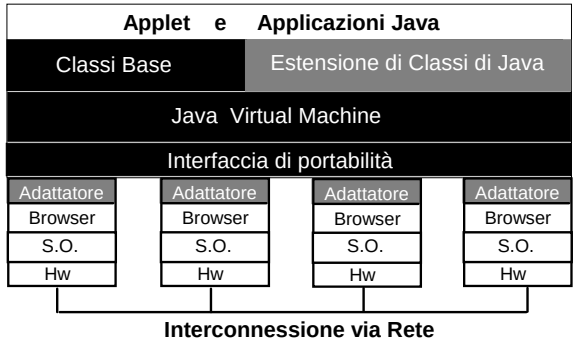
Approcci ad ambienti aperti

possibilità di avere un **ambiente aperto** per superare la **eterogeneità** delle diverse piattaforme anche durante la esecuzione **senza fare ripartire il sistema**

Ambienti standard (anche proprietari)
ANSA, DCE, OSF, etc.

Ambiente interpretati
linguaggi script
tipo *Shell, TCL/Tk, Perl, Python*
Java legato allo scenario **Web**

Java si basa su un interprete e una macchina virtuale



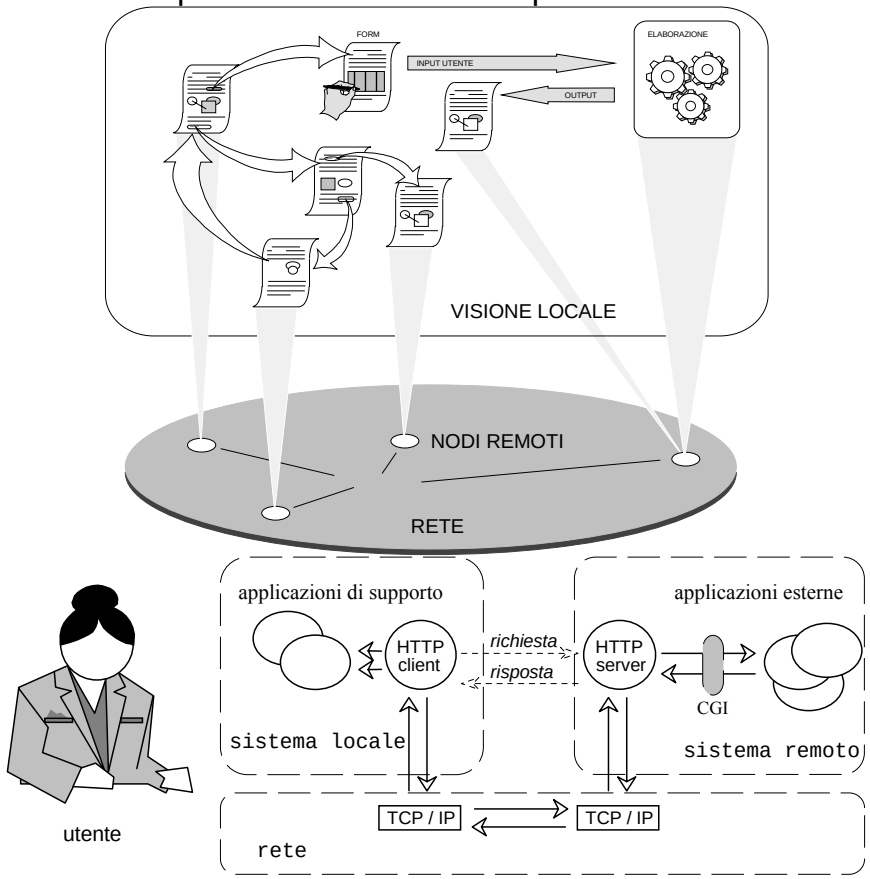
Java consente una facile integrazione con le informazioni Web attraverso applet e la facile integrabilità
Inoltre, comincia ad avere strumenti per il supporto:

- **distribuzione dinamica del codice**
- **sicurezza**
- **riflessione**
- **gestione e monitoraggio delle risorse** (??? in attesa)

Sistemi di rete Cliente/Servitore

Spesso i sistemi sono solo **sistemi di rete** in cui le potenzialità non sono sfruttate

Scenario Web
Informazioni presentate in modo trasparente



DINAMICITÀ
Se cominciamo a replicare i dati e memorizzare le informazioni su siti intermedi => **evoluzione**

Sistemi operativi di Rete vs. Sistemi operativi Distribuiti

Opposizione tra
Network Operating Systems (NOS) vs.
Distributed Operating Systems (DOS)

I **NOS** sono
indipendenti e **non trasparenti**
I DOS sono
coordinati e **trasparenti** e **aperti**

GESTIONE RISORSE TRASPARENTE

DOS
paralleli al loro interno
uso di risorse parallele interne
paralleli a livello di utente
con maggior omogeneità
capaci di gestire **nuove risorse** non note
da aggiungere alle statiche

*? La **trasparenza** consente di crescere e di modificare il sistema ?*
*? Si deve avere sia **trasparenza** a livello utente sia **visibilità** a livello di sistema*

In tempi recenti emerge la esigenza di PUNTI DI VISTA DIVERSI E MULTIPLI

allo stato dell'arte
i sistemi devono fornire anche visibilità
livelli diversi di trasparenza per diversi livelli di uso

Sistemi Operativi Distribuiti

ottimizzano l'uso delle **risorse distribuite**
e sono basati sulla
CONDIVISIONE per
- **scambio** di informazioni
- **ridistribuzione** del carico
- **replicazione** delle informazioni
- **parallelismo** nella computazione

Unica macchina virtuale con
Proprietà
- Controllo allocazione delle **risorse**
- **Comunicazione**
- Autorizzazione
- **Trasparenza** (qualche livello)
- Controllo dei servizi del sistema (**QoS**)
- Capacità evolutiva (**dinamicità**)

Apertura del sistema (OPEN system)
le risorse possono essere inserite durante la esecuzione del sistema e non sono note staticamente prima della esecuzione

Naturalmente,
non esistono **sistemi aperti** in assoluto,
ma che sono aperti a fronte di
alcune variazioni e di ***alcuni*** cambiamenti

SISTEMI OPERATIVI DISTRIBUITI

come insieme di **gestori di risorse**

Resource management

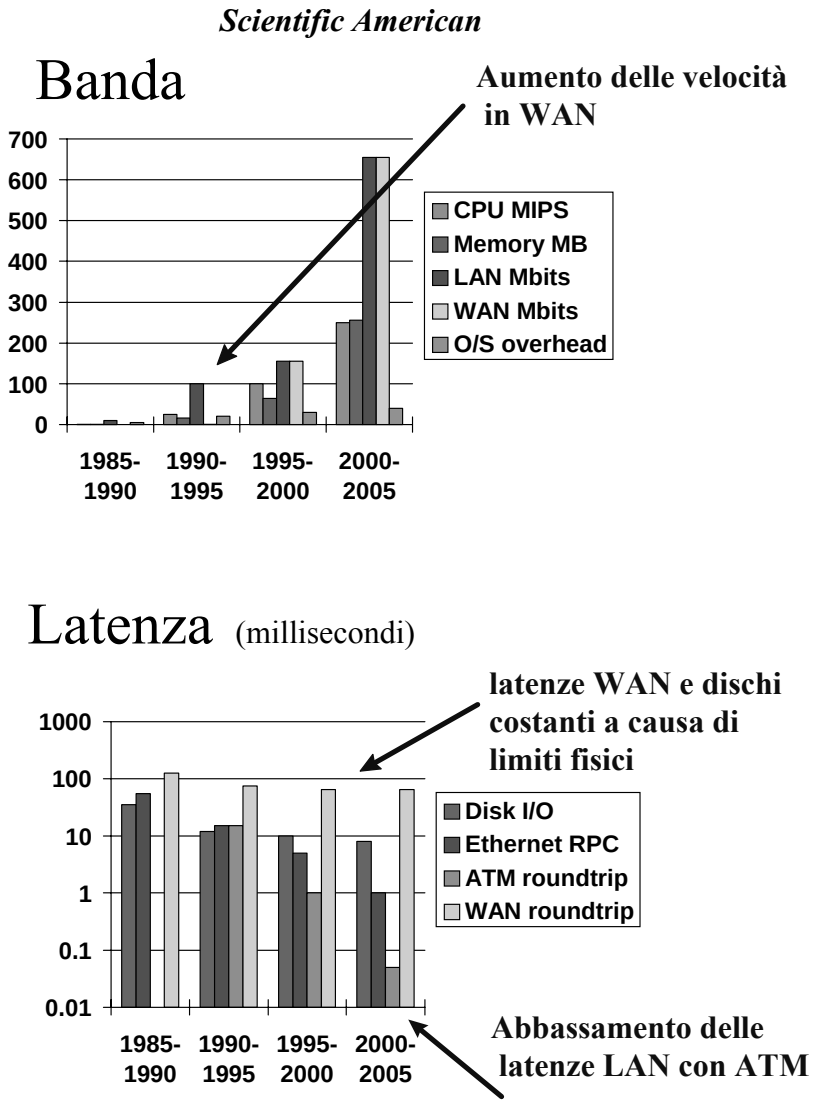
Processor management
Process management
Memory management
File management

FILONI

Replicazione
Gestione dei Nomi
Comunicazione e Sincronizzazione
Processi
Sicurezza
Allocazione e riallocazione delle risorse
Gestione dei Servizi applicativi e della qualità

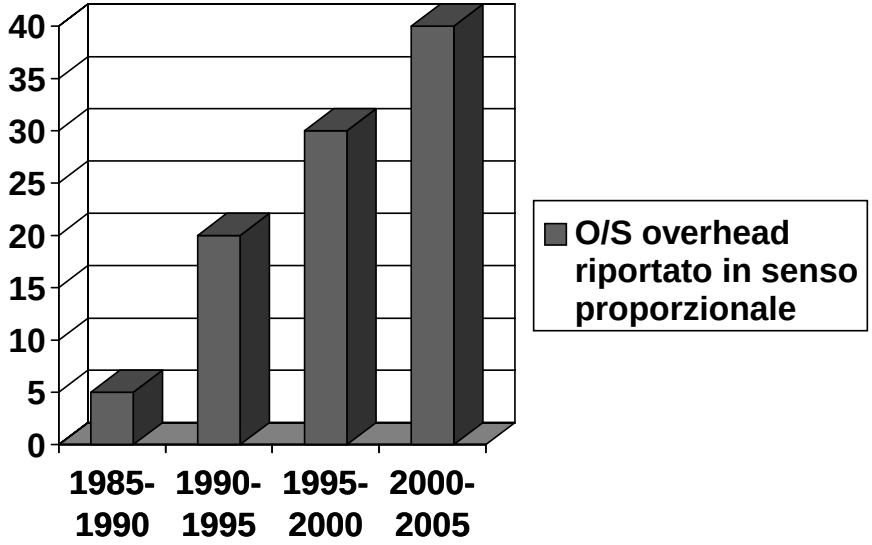
STANDARDIZZAZIONE
INDIPENDENZA dalla **ARCHITETTURA**
APERTURA (OPEN SYSTEM)
INFORMAZIONI SUL SISTEMA (MONITORAGGIO)

TREND delle prestazioni



TREND overhead

Overhead della parte di **sistema operativo**
cioè della gestione locale



Anche se ci possiamo aspettare dei **miglioramenti** in **prestazione** delle diverse parti di un sistema (comunicazione, dischi remoti, accessi veloci, etc.)

Il problema da risolvere è la **corretta gestione**

Notate non stiamo ipotizzando la **gestione ottima** ma una apertura del sistema per alcune proprietà

ad esempio, evitando alcuni **colli di bottiglia** attraverso un **monitoraggio del sistema**

CLIENTE/SERVITORE

tipico caso di invocazione **procedurale**:
il **cliente** aspetta il completamento del **servizio**
attuato dal **servitore (in modo sincrono)**
Se mettiamo in gioco **due processi**:
potremmo anche avere **schemi** diversi

Se il **cliente** ha bisogno di una informazione la richiede
=> in genere aspetta fino a quando non è disponibile
Questo modo carica il cliente della responsabilità dell'ottenere le informazioni

modello pull

Se il cliente non vuole bloccarsi, aspettando
richieste a risposta immediata
CICLO di richieste ad intervallo di ***polling***
che deve essere o esaudita subito
o restituire un insuccesso

ancora **modello pull**

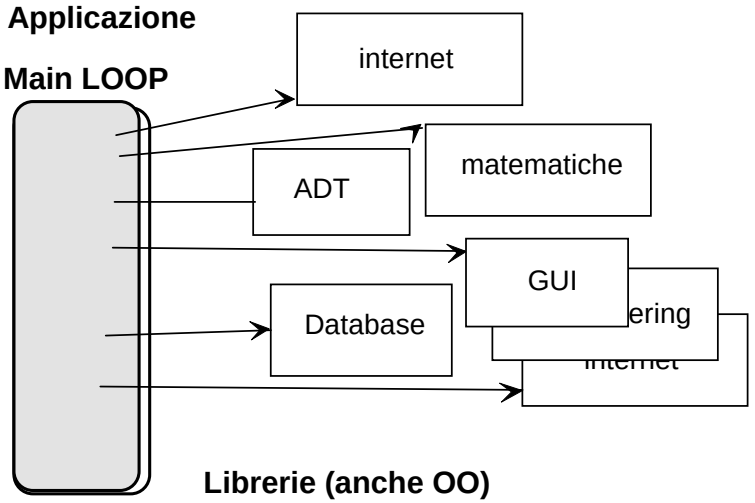
Si può risolvere in modo diverso, dividendo le responsabilità
il cliente segnala il proprio interesse, poi fa altro
è compito del server di inviare la informazione
se e quando disponibile
Questa interazione divide i compiti tra cliente e servitore

modello push (per il servitore)
che forza le informazioni rovesciando i ruoli

RIUSO dell'esistente
Framework, librerie di componenti
molte modalità diverse ed eterogeneità

TERMINOLOGIA

LIBRERIE
Un **insieme di librerie** costituisce un **set di funzioni** (interfaccia nota) che un livello può chiamare



Ogni applicazione si deve uniformare a questa struttura
In caso di più applicazioni, ogni applicazione deve replicare la stessa organizzazione

Le librerie sono ad aggancio **dinamico** (DLL)
Dynamic **L**inked **L**ibraries

DLL ***NON*** *sono SVILUPPATE e COLLEGATE insieme con i programmi utilizzatori*

la libreria è **sviluppata separatamente** e **collegata** solo al momento della prima **richiesta**

Principio di riuso: una **unica immagine** per usi **multipli** e per **processi anche diversi**

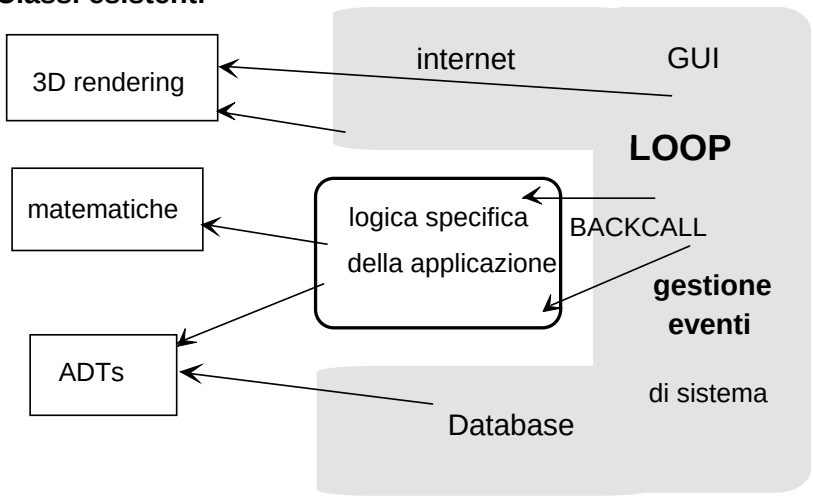
FRAMEWORK

Un **FRAMEWORK** costituisce un modello di esecuzione ed un ambiente di richiesta di funzionalità più integrato tende:

- a non replicare la struttura implicita
- a rovesciare il controllo (per eventi di sistema)
  *vedi **Windows** struttura a loop di attesa di eventi che vengono smistati ai richiedenti*

meccanismi di supporto della cornice

Classi esistenti



Funzioni e servizi

Sono possibili backcall o upcall (**push** del framework) dal Framework alle applicazioni

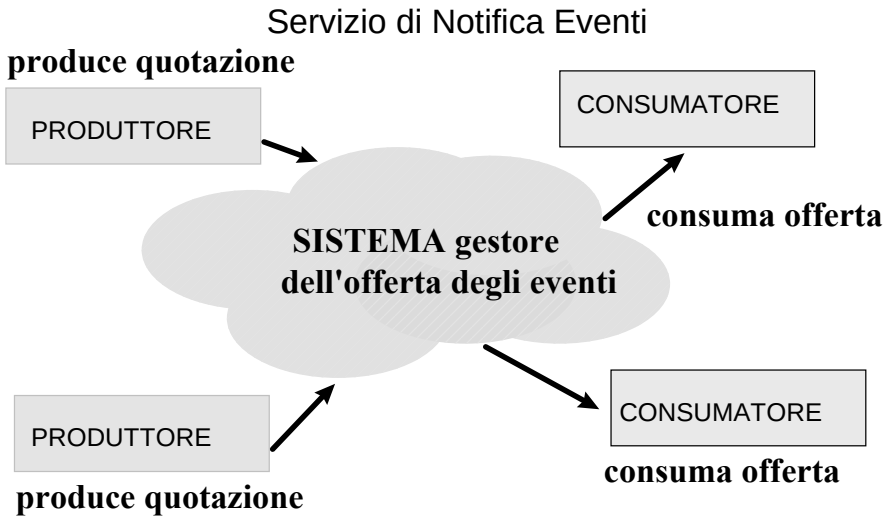
eventi asincroni

MODELLO ad EVENTI

in contrapposizione ad un **modello sincrono** di richiesta e di attesa della risposta

si gestisce la possibilità di inviare messaggi su necessità disaccoppiando gli interessati

- I consumatori eseguono le attività
- I produttori segnalano le necessità
- Il gestore degli eventi segnala l'occorrenza degli eventi significativi agli interessati



push dell'evento alle entità interessate e registrate
Vedi la **maggior parte dei sistemi a finestre**

CONCETTI di BASE

nei sistemi distribuiti
Esistono? Servono?

MODELLI di SERVIZIO
modelli statici/dinamici
modelli preventivi /reattivi

modello di esecuzione nel sistema
monoutente/multiutente
processore/processori
modello per esecuzione attiva
processi/oggetti replicazione
modello dei processi
processi pesanti/ processi leggeri

modello di allocazione / riallocazione
entità del sistema sulle risorse del sistema
modello di naming
conoscenza reciproca
modello di comunicazione

modelli di guasto
modelli di replicazione

modelli di monitoring e controllo qualità

?trasparenza?
necessità di standard

modello preventivi /reattivi

modelli ottimisti e pessimisti

Esempi

approcci per gestire il deadlock

- approcci che prevengono il deadlock
  - **avoidance** si evita a priori tramite introduzione di forti vincoli (sequenzializzazione completa)
  - **prevention** si evita la specifica situazione con algoritmi (accesso in ordine)

- approcci che fanno fronte all'occorrenza del deadlock
  - **recovery** se succede, interveniamo

il problema della interferenza/congestione

- **ring** previene la congestione con regole precise di controllo di accesso
- **CSMA/CD** tenta l'accesso senza controllo e deve tenere conto della interferenza possibile, con opportune strategie

I primi sono approcci **pessimisti/preventivi**
I secondi sono approcci **ottimisti/reattivi**

Naturalmente, in genere
i **sistemi preventivi** sono **statici** per l'aspetto considerato
i **sistemi reattivi** sono **dinamici**

modello di esecuzione nel sistema

monoutente/multiutente

in genere, l'uso di un sistema in modo dedicato è tipico delle fasi prototipali

l'uso di più utenti, consente di formare un migliore mix di entità eseguibili sul/sui sistemi per un migliore equilibrio

☹ Problemi di **configurazione** del sistema ed
interferenza tra le attività

processore/processori usati con trasparenza

modello workstation

si utilizzano le risorse locali preferenzialmente
poi si possono considerare le risorse di rete

Sistemi distribuiti in rete (tradizionali)

modello processor pool

si utilizzano le risorse in modo trasparente
a secondo del loro utilizzo e disponibilità

Sistemi distribuiti veri e propri

☹ maggiore overhead di coordinamento

modello di naming e sistemi di nome

necessità di **conoscenza reciproche** delle entità / servizi
in una relazione **cliente/servitore**
il cliente deve avere un riferimento al servitore

```
indirizzoServitore
nomeNodo.nomeServitore
nomeGestore.nomeServitore
nomeServizio
nomegestore.nomeServizio
```

Notiamo che i riferimenti sono distribuiti nel codice dei clienti, degli utilizzatori, delle librerie, ecc.
Si deve garantire la consistenza

NON TRASPARENZA vs. TRASPARENZA
dei servizi ai nodi
degli indirizzi all'interno del nodo

Come e si qualificano i nomi e
quando si risolvono i riferimenti?

BINDING STATICO VS. DINAMICO

statico: i riferimenti sono risolti prima della esecuzione
dinamico: i riferimenti sono risolti al bisogno

In caso di sistemi concentrati =>
Binding tipicamente statico
ma a causa delle necessità di riutilizzo
anche librerie dinamiche

NOMI nel DISTRIBUITO

CASO STATICO
invarianza dei nomi e della **allocazione** delle entità
si risolve il tutto **staticamente** e
non si necessita di un servizio di nomi

I nomi sono risolti prima della esecuzione e non è il caso di cambiare alcuna allocazione (altrimenti ☹)

E se le entità si **muovono**?
entità trasparenti e **NOMI invarianti**
INDIPENDENZA dalla ALLOCAZIONE
nomi non trasparenti dipendenti dalla locazione corrente
solo TRASPARENZA dalla ALLOCAZIONE
si devono riqualificare i nomi di tutti i possibili **clienti**

CASO DINAMICO
In caso dinamico, nasce la necessità di un servizio di nome (name server) che mantiene e risolve i nomi
e fornisce il servizio durante la esecuzione
coordinandosi con i gestori della allocazione

CASO DINAMICO entità non staticamente fissate
Uso di **tabelle di allocazione** ==>
controllate da opportuni **GESTORI dei NOMI**

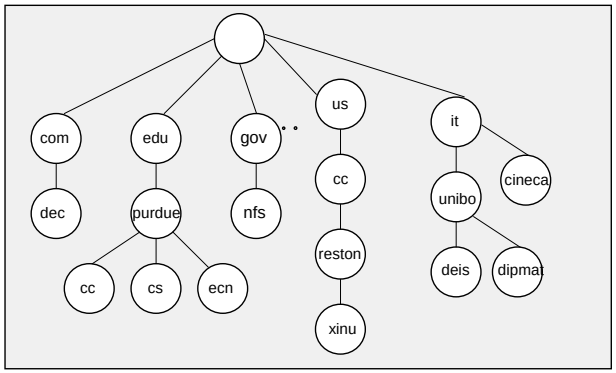
modello di naming in ***sistemi aperti***
possibilità di inserire nuove entità
compatibili con il sistema già esistente

Gestori di nomi
sono tipicamente responsabili
dei servizi di nomi e
della trasparenza dei nomi
e sono tipicamente più di uno

COORDINAMENTO di **gestori di nomi** distinti

partizionamento dei gestori
ciascuno responsabile di una sola partizione dei riferimenti - **località**
(in generale i riferimenti più richiesti)
replicazione dei gestori
ciascuno responsabile con altri di partizione dei riferimenti - **coordinamento**

Spesso organizzazioni a livelli
gestore generale che coordina gestori di più basso livello in molti livelli con località informazioni
(vedi CORBA o DNS)



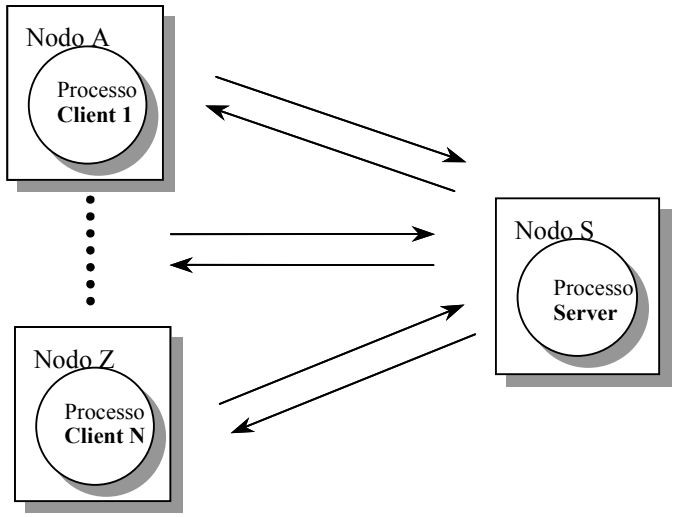
modello di comunicazione

schema di base **cliente servitore**

semantica per riferimento

Modello di comunicazione **asimmetrica**, **molti:1**
in cui i servizi sono forniti da ***un servitore***

- i **clienti** conoscono il servitore
- il **servitore** rappresenta la astrazione dei servizi e non conosce i possibili clienti



MODELLO astratto

interazione **sincrona (default)**
asincrona / non bloccante

Progetto dei componenti

progetto del server
progetto del client

Maggiore complessità di progetto dei Server rispetto al progetto dei Client

il Server può accedere alle risorse del sistema considerando i problemi di:

autenticazione utenti
autorizzazione all'accesso
integrità dei dati
privacy delle informazioni

e deve gestire richieste contemporanee da molti Client (server concorrenti)

schemi più complessi

clienti / servitori multipli

I servizi sono forniti da **più servitori coordinati** (molti:molti) con diverse possibilità

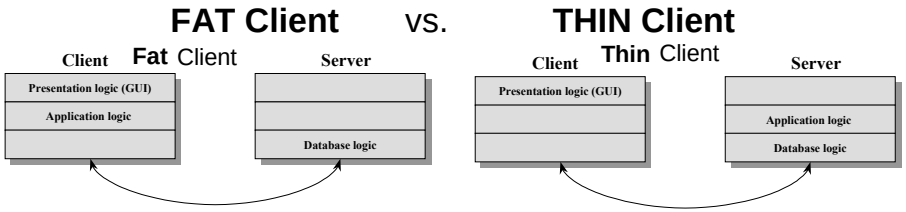
- 1 solo servizio per ogni richiesta
  (stampa di un file)
- 1 servizio da ogni possibile servitore
  (aggiorna copie di un file)

Necessità di coordinamento tra i servitori

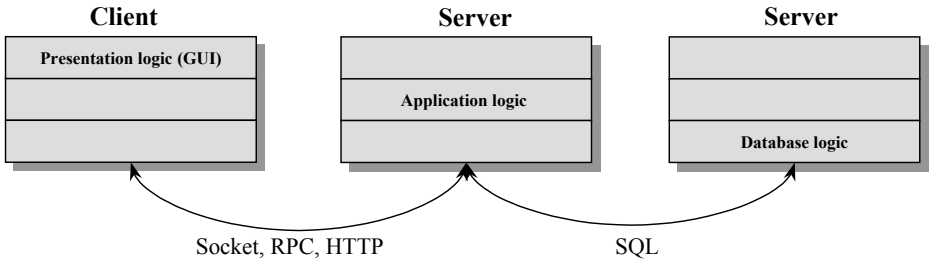
con Replicazione o Suddivisione

PROGETTO DI CLIENTI E SERVITORI

Distribuzione della **logica** applicativa tra **Client** e **Server**
Il server prevede un progetto complesso e il client più snello



SISTEMI MULTI TIER (3-tier)
sistema C/S decomposto in parti, per **distribuire il carico di lavoro** di un Server su diverse macchine



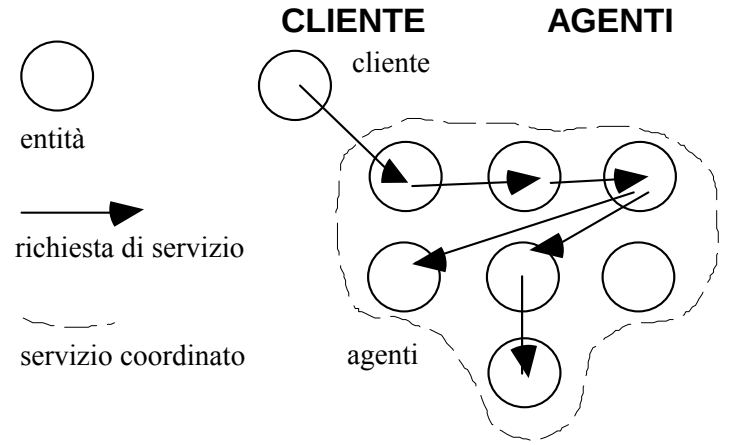
Si possono anche avere molti **nodi diversi** che si incaricano di svolgere funzioni molto diverse e sono separate nel sistema per ragioni diverse
di bilanciamento di carico
di limiti di esecuzione
di disponibilità e tolleranza ai guasti
di vicinanza geografica

SCHEMA a CLIENTI e AGENTI MULTIPLI

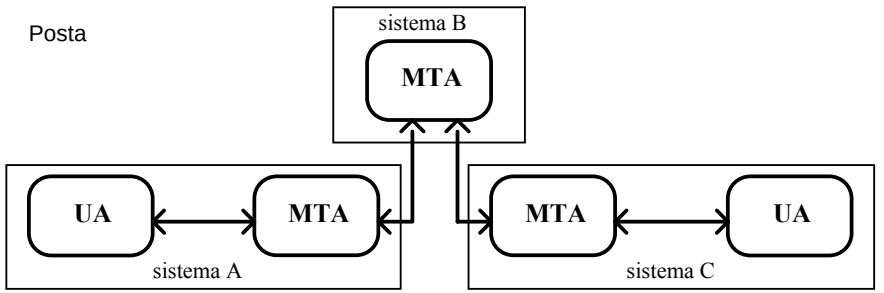
I servizi sono forniti dal ***coordinamento*** di più servitori che forniscono un servizio ***globale unico***
modello two tier

gli agenti forniscono il servizio e possono:
- partizionare le capacità di servizio
- replicare le funzionalità di servizio

in modo trasparente al cliente

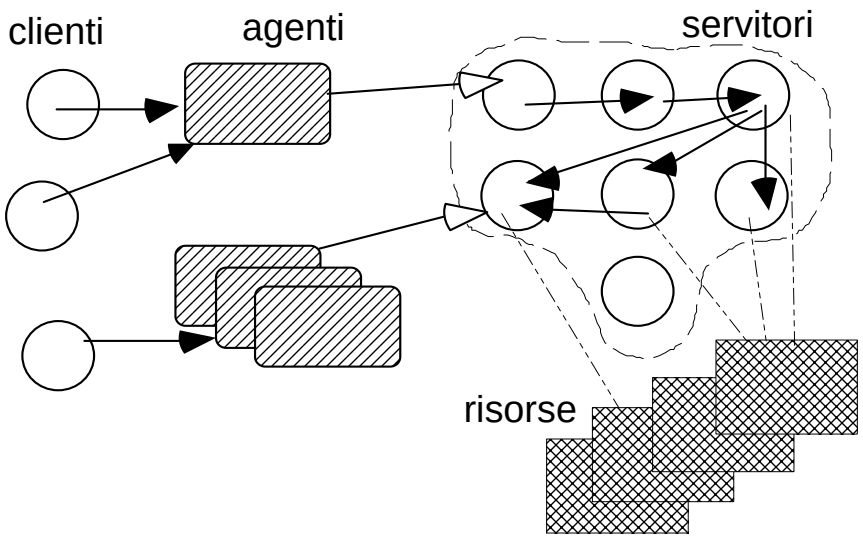


Mail



schemi con **clienti agenti/servitori multipli**

CLIENTE AGENTI / SERVITORI e RISORSE



AGENTI

anche paralleli e capaci di coordinarsi

SERVITORI

anche paralleli replicati e coordinati

PROTOCOLLI

di *coordinamento*, *sincronizzazione*, *rilevazione* e *tolleranza* ai *guasti*

Due livelli di **coordinamento** nell'accesso alle **risorse** da parte dei **clienti** ***modello 3 tier***

MODELLI a LIVELLI MULTIPLI
per la divisione dei compiti

confinando il lavoro sul livello meno congestionato

SCHEMI DI COMUNICAZIONE A MULTICAST/BROADCAST

STATO DEL SERVITORE

Due tipi di interazione tra **Client** e **Server**

stateful, cioè si mantiene lo **stato dell'interazione** e un messaggio e la operazione conseguente può dipendere da quelli precedenti

stateless, non si tiene traccia dello stato, ogni messaggio è completamente indipendente dagli altri

Lo **stato dell'interazione** usualmente memorizzato nel **Server** (che può essere *stateless* o *stateful*)

Un Server **stateful** garantisce efficienza *(dimensioni messaggi più contenute e migliore velocità di risposta del Server)*

Un Server **stateless** è più affidabile in presenza di malfunzionamenti *(soprattutto causati dalla rete)*

Il **Server stateful** deve poter identificare il Client
Per esempio, si consideri le differenze tra un file server stateless e stateful (NFS di SUN è stateless)

La scelta tra *server stateless* o *stateful* deriva soprattutto dalla logica dell'applicazione
Un'interazione stateless è conveniente **SE** il **protocollo** applicativo è progettato con **operazioni idempotenti**

Operazioni idempotenti producono sempre lo stesso risultato, per esempio, un Server fornisce sempre la stessa risposta a un messaggio M indipendentemente da altri messaggi (anche uguali ad M) ricevuti dal Server stesso

STATO nel modello cliente/servitore

Le realizzazioni più comuni non prevedono **stato della connessione** per il servitore
NESSUNO STATO nel SERVER (server stateless)

Concetto di **STATO della interazione**
STATO PERMANENTE della interazione
? mantenuto dal cliente o dal servitore
STATO SOFT nel SERVER
(mantenuto solo per un tempo predeterminato)

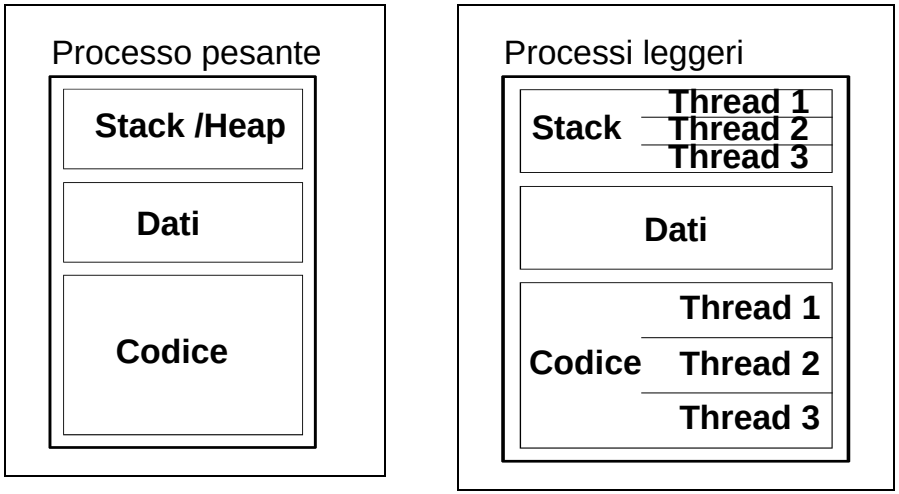
- ***Servitore che tiene traccia dei clienti***
  servitore con **stato**
- ***Servitore che tiene traccia della storia della interazione con ogni cliente***
  servitore con **stato partizionato**: una partizione per ogni sessione di interazione
- ***Servitore che conosce i clienti***
  uso di una lista per la autorizzazione
- ***Servitore che consente interazione tra i clienti***
  servitore con stato e interazione mutua tra i clienti attraverso i servizi
- ***Servitori coordinati con interazione tra i clienti***
  coordinamento dei servitori con stato
  interazione mutua tra clienti e servitori

- Giochi di simulazione distribuita (MOO)
- Prenotazione aerei
- Navigazione in Web
- mantenere **stato** su Web

e la consistenza delle copie replicate?

MODELLO DELLE ATTIVITÀ O DEI **PROCESSI**

Processi differenziati



confronto tra processi

processi pesanti/ processi leggeri

Processo
IMPLEMENTATIVAMENTE

SPAZIO di **indirizzamento**
SPAZIO di **esecuzione**
insieme di codice, dati (statici e dinamici),
parte di **supporto**, cioè di interazione con il **sistema operativo** (file system, shell, socket, etc.) e per la comunicazione

un'aggregazione di parecchi componenti
Processo pesante ad esempio in UNIX
cambiamento di contesto operazione molto pesante
overhead

PROCESSI granularità

SOLUZIONE per superare i limiti

Possibilità di creare **entità più leggere**
con limiti precisi di visibilità e barriere di condivisione

Processi leggeri

attività che condividono **visibilità** tra di loro e che sono caratterizzati da uno **stato limitato in dimensioni**

overhead limitato nel cambiamento di contesto

ad esempio in UNIX, i **thread** o
lightweight process

Quale **contenitore unico** si può considerare per la condivisione della visibilità dei thread?

Il processo pesante funziona anche da
contenitore di processi leggeri

In genere, un **processo pesante fornisce l'ambiente unico** che contiene **più processi leggeri o thread**
che offre una unica interfaccia di accesso al kernel

sessioni sul file system, socket di comunicazione come dati condivisi visibili ai thread

PROGETTO delle APPLICAZIONI

Tutti i sistemi vanno nel senso di offrire
granularità differenziate per ottenere
un servizio migliore e più adatto ai requisiti dell'utente

Si usano **processi pesanti** se è il caso e
processi **leggeri** se si vuole limitare l'overhead

In caso di **mobilità**, si deve considerare cosa muovere
e si è facilitati nel **muovere processi pesanti** e non leggeri

modello di esecuzione

Entità ed interazione

modello a **processi**

Processo

entità in esecuzione che possono eseguire e comunicare direttamente o indirettamente con altri processi attraverso
- **memoria condivisa**
- **scambio di messaggi**

Uso di dati **esterni** ai **processi** stessi
(**scarso confinamento**)

modello ad **oggetti** (e processi)

Oggetto

entità dotata di astrazione che
- racchiude risorse interne (astrazione) su cui definisce operazioni
- agisce su risorse interne alla richiesta di operazioni dall'esterno

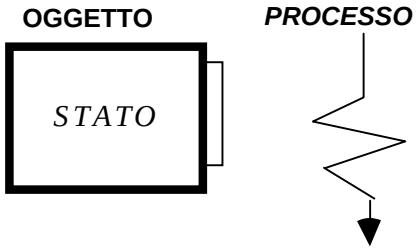
- **oggetti passivi**
  astrazioni su cui agiscono processi esterni

- **oggetti attivi**
  capaci di esecuzione e scheduling

OGGETTI E PARALLELISMO

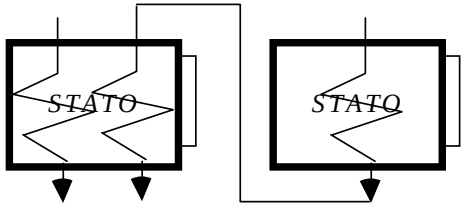
come inserire il ***parallelismo*** in un modello ad oggetti

a) affiancando il concetto
di PROCESSO
a quello di
OGGETTO



===> modelli ad oggetti PASSIVI

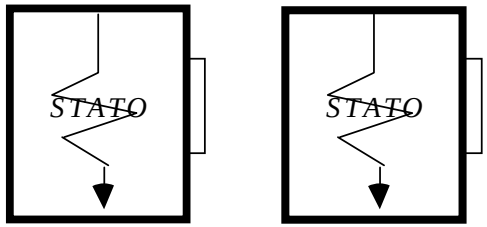
SVANTAGGI:
perdita di
uniformità di concetti
protezione



b) **integrando** il concetto di PROCESSO ed OGGETTO

===> modelli ad oggetti ATTIVI

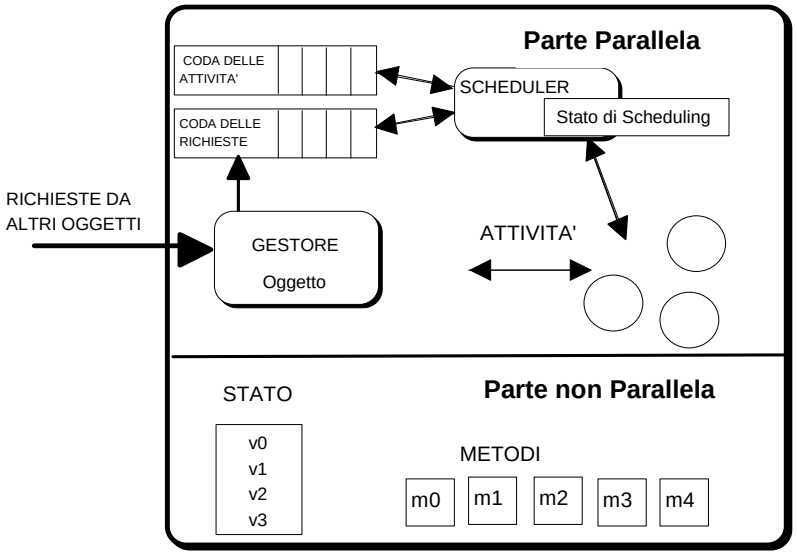
VANTAGGI:
uniformità di concetti
protezione
information hiding



Modello a maggior confinamento **(information hiding)**
Non esiste alcun processo che si muove sugli oggetti
Oggetti Attivi decidono independentemente e nascondono il comportamento interno

Oggetto attivi

Ogni oggetto racchiude al proprio interno la necessaria **capacità di concorrenza** e definisce la propria **gestione della concorrenza**
Ogni oggetto deve prevenire eventuali interferenze



dimensione interoggetto
dimensione intraoggetto

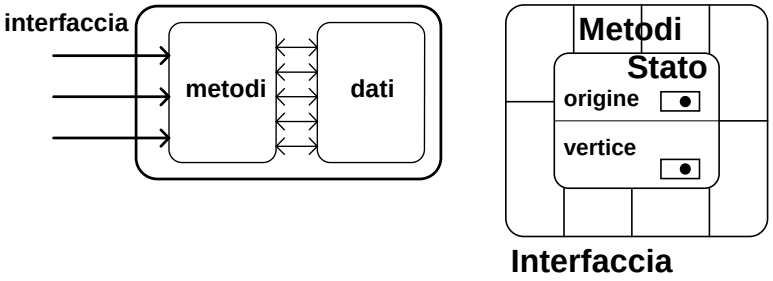
Problemi

Allocazione dei processi ed attività
Comunicazione
Dinamicità

È più facile riallocare un processo od un oggetto?
e i suoi riferimenti verso altri oggetti **(come cliente)**
e i riferimenti verso l'oggetto stesso **(come servitore)**

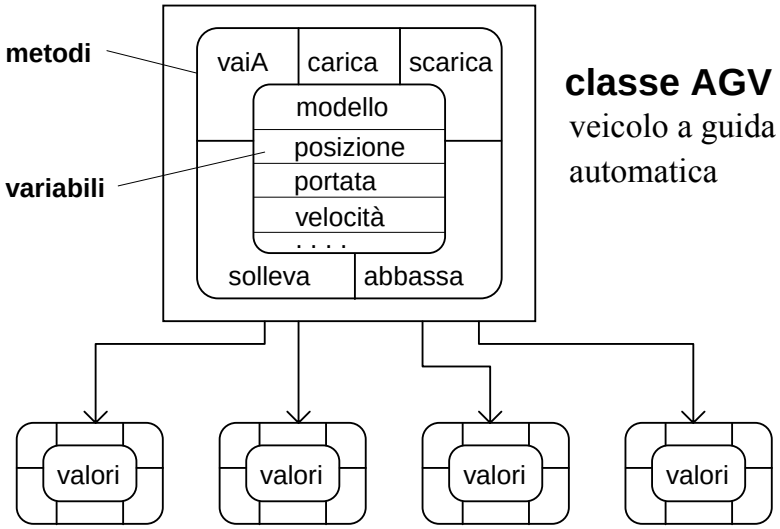
OGGETTI E CLASSI - RELAZIONI LOGICHE

OGGETTI COME ASTRAZIONE DELLO STATO



Interfaccia

Classi come descrittori di insiemi di oggetti, istanze create e descritte dalla stessa classe



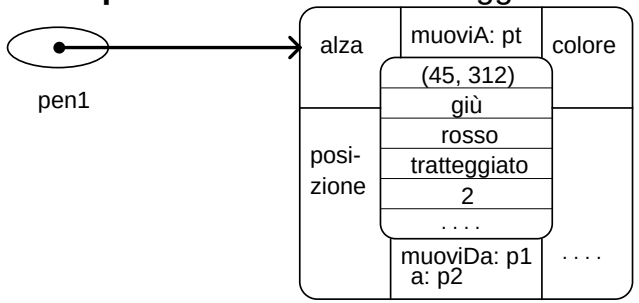
classe AGV
veicolo a guida automatica

istanze

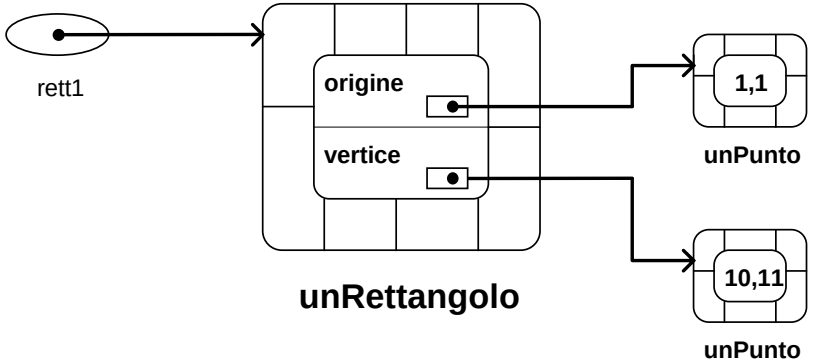
SEMANTICA PER RIFERIMENTO

lo stato di un oggetto è costituito di **variabili del linguaggio** visibili e manipolabili solo dentro gli oggetti
lo stato è composto di
- **valori primitivi**
- **riferimenti ad altri oggetti**

variabili come puntatori ad istanze di oggetti



Oggetti come contenitori di variabili che possono puntare ad istanze di oggetti



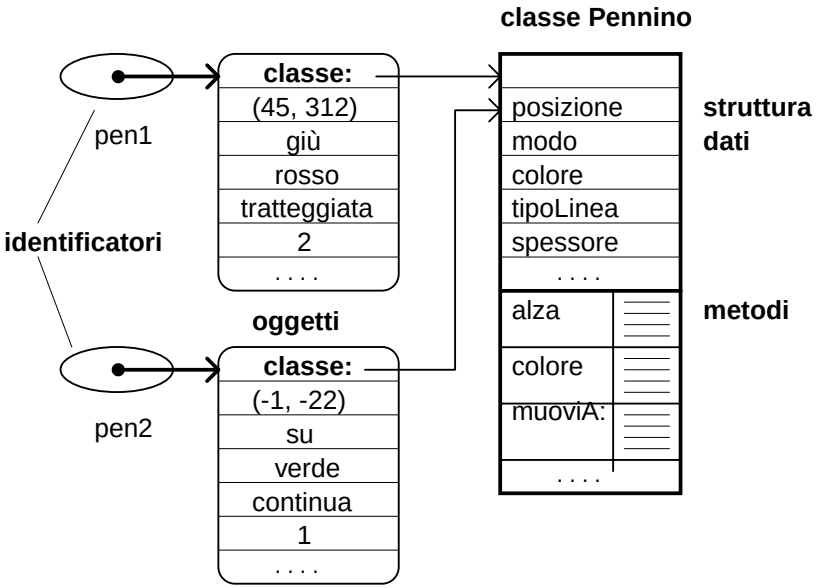
unRettangolo

Possiamo pensare ad una applicazione come una navigazione sulle informazioni legate dalle relazione tra oggetti
(Grafo di riferimenti tra oggetti)

IMPLEMENTAZIONE delle relazioni

LEGAME OGGETTO/CLASSE ISTANCE-OF IS-A

Oggetti che fanno riferimento alla loro classe
durante la esecuzione per ritrovare i metodi



LEGAME CLIENTE/SERVITORE USES

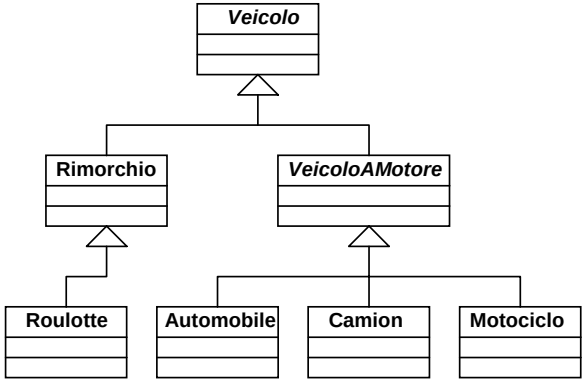
uso delle variabili per chiedere ad oggetti di eseguire metodi attraverso le variabili che puntano ad altri oggetti

pen2 alza
pen1 muoviA: punto1

Relazione di Ereditarietà

sempre tra classi

Classi che possono derivare da classi
precedentemente specificate



Anche forme di **ereditarietà multipla**
Una classe deriva da molteplici classi già specificate

Le istanze tengono conto (e hanno lo stato possono ricevere richieste per metodi) di tutte le superclassi

ereditarietà multipla più difficile

- da usare nel **progetto logico** delle soluzioni del problema
- da **realizzare** a livello implementativo

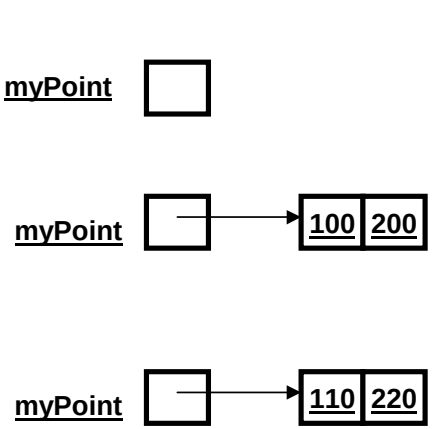
Presenza di tutte le entità sullo stesso nodo

LINGUAGGI PURI AD OGGETTI (JAVA)

SEMANTICA PER RIFERIMENTO

Uso solo di variabili che sono riferimenti ad altri oggetti
In più, sono anche **tipate**
(per consentire controlli statici, prima della esecuzione)

Java come esempio



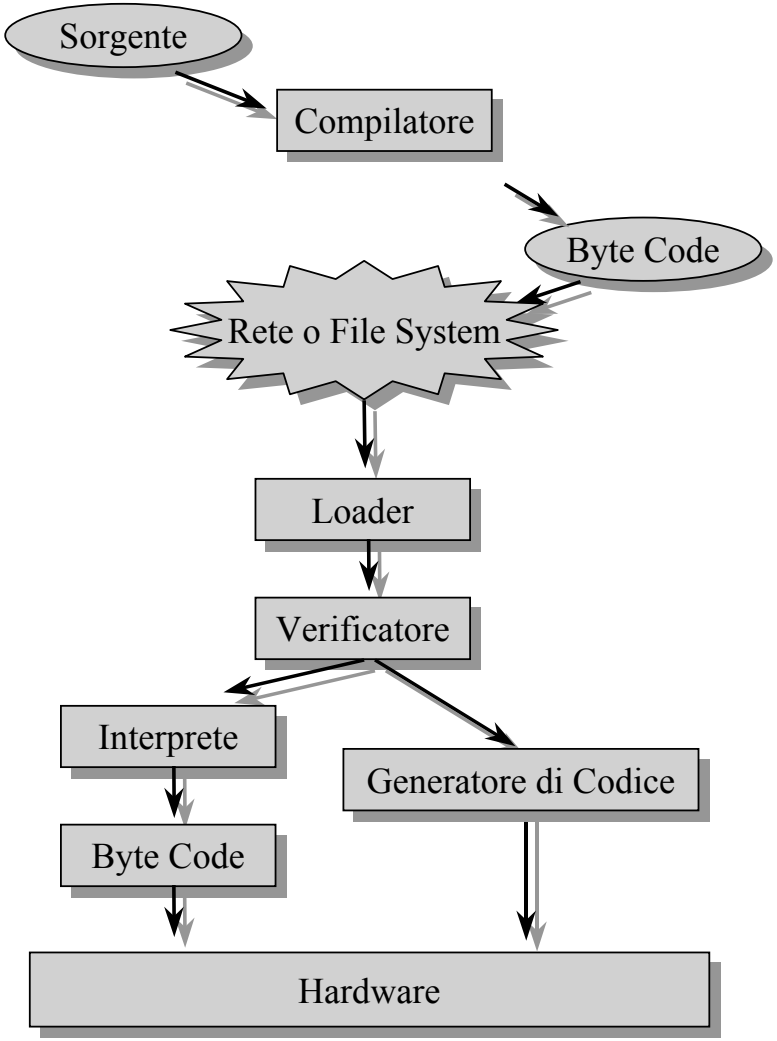

```
java.awt.Point myPoint;

Point myPoint    ;

myPoint = new Point(100,200);

/* dot notation */
 myPoint.x += 10;
 myPoint.y += 20;
```

Gli oggetti non sono innestati
e la richiesta di un metodo può avvenire solo avendo una variabile che contiene il riferimento
- ottenuta
- o inizialmente
- o durante la esecuzione

In genere i riferimenti sono locali, nella stessa località o (in Java) nella stessa macchina virtuale

e la distribuzione? aspettiamo un po'

Ambiente Java
dinamicità del caricamento



Caricamento dinamico dei componenti
(classi e oggetti?)

Java Virtual Machine

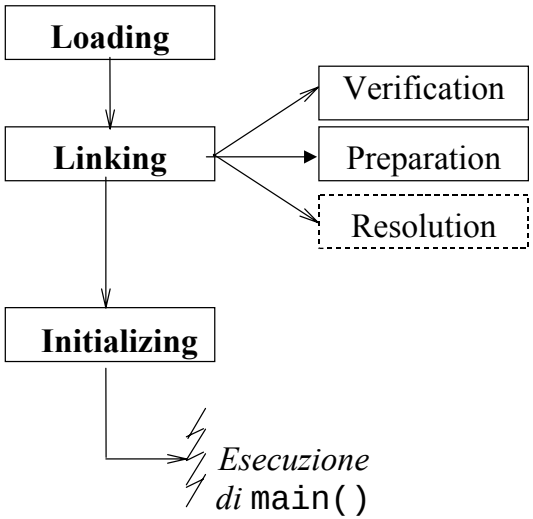
caricamento su necessità

lazy loading
ossia sfruttando la dinamicità

caricamento in memoria file *.class*

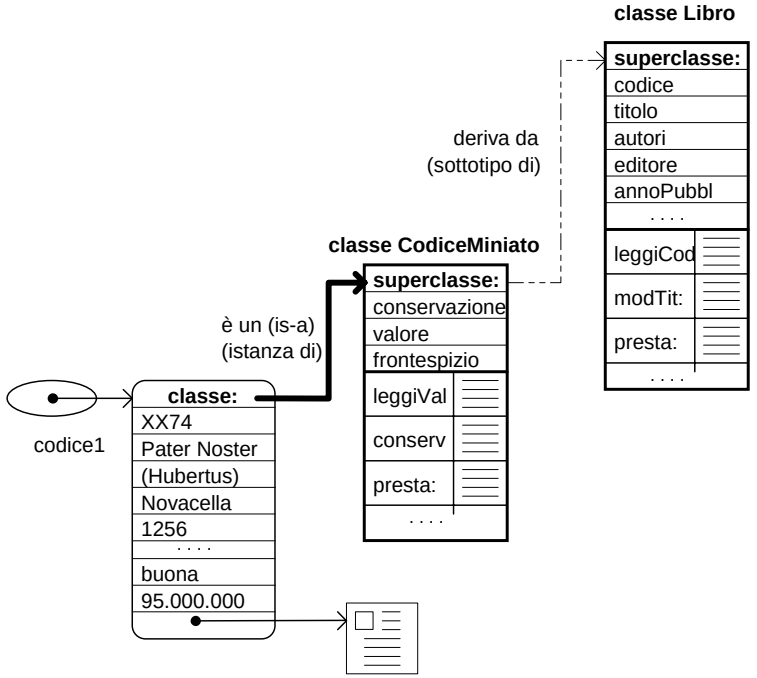
verifica byte-code
allocazione spazio in memoria
risoluzione di tutti i riferimenti a classi

inizializzazione delle variabili statiche allocate



Esempio:
int intero = 5; **Punto** unpunto;
preparation: memoria per la classe int o per la classe punto
se non presente
inizializing: il valore di int viene modificato in 5
o si attribuisce valore alle variabili di stato dell'oggetto istanza

Riferimenti tra oggetti dello stesso nodo



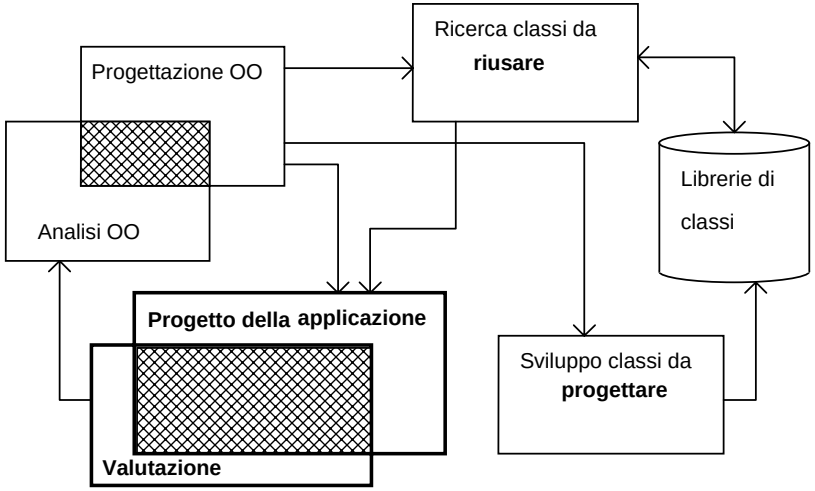
L'ereditarietà non cambia il comportamento istanze

Problemi nelle applicazioni distribuite
allocazione di tutte le classi e le relative istanze insieme
Sbilanciamento **carico dei nodi** e **problemi di allocazione**

soluzione ovvia:
Classi non modificabili e sono **copiate su ogni nodo**
Si spezza il grafo di connessione tra le istanze
istanze che devono riferire solo **oggetti sullo stesso nodo di esecuzione**, si possono mettere su qualunque nodo

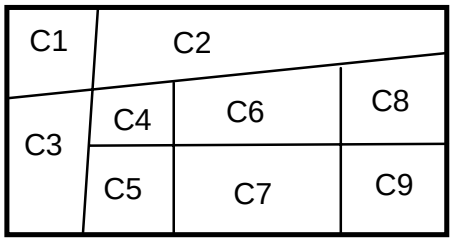
(realizzazione **monoapplicazione** e **monoutente**)

Sviluppo di una applicazione



In modo indipendente dalla configurazione e senza preoccuparci di come verrà mappato sulla architettura

Applicazione



Possibile suddivisione in componenti

Suddivisione fatta**:**
- in modo **automatico e trasparente alla applicazione**
- sulla base di aiuti forniti dal **progettista di applicazione**

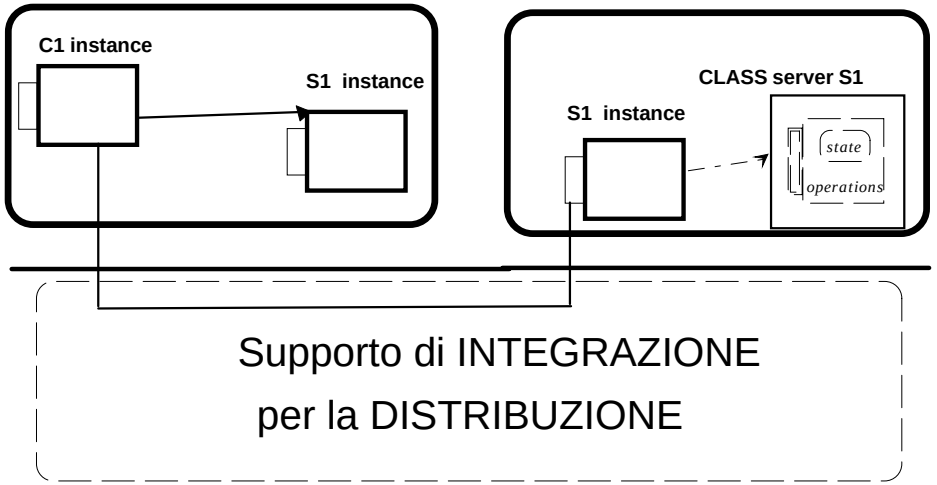
Allocazione dei componenti

*In genere i **sistemi più diffusi** (e anche Java) assumono che un programma (o una applicazione) faccia solo **riferimenti interni e locali***

Per ritrovare entità esterne si usano protocolli introdotti per i sistemi distribuiti con **esplicita visibilità di allocazione**

ad esempio
***TCP/IP** ed il suo sistema di nomi (e **socket**)*
*o altri tipo **CORBA***

*o ad-hoc come **PVM, ANSA, ADA***

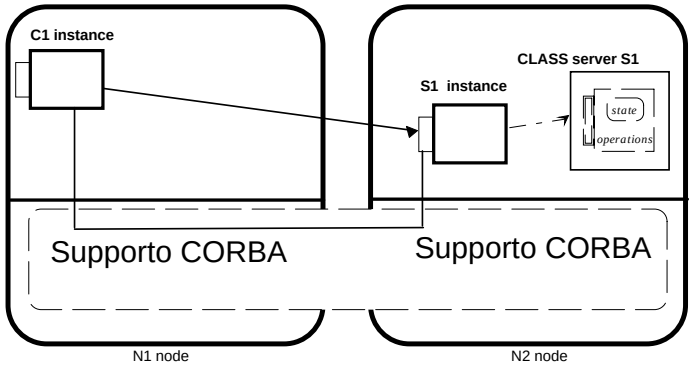


NON trasparenza alla allocazione
ma **riferimenti locali** e **riferimenti remoti**

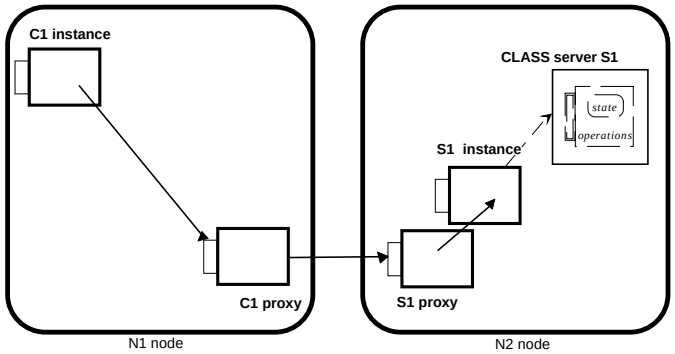
Riferimenti remoti

in **Java non sono possibili riferimenti remoti**

ma si possono costruire in molti modi
- uso di **CORBA**



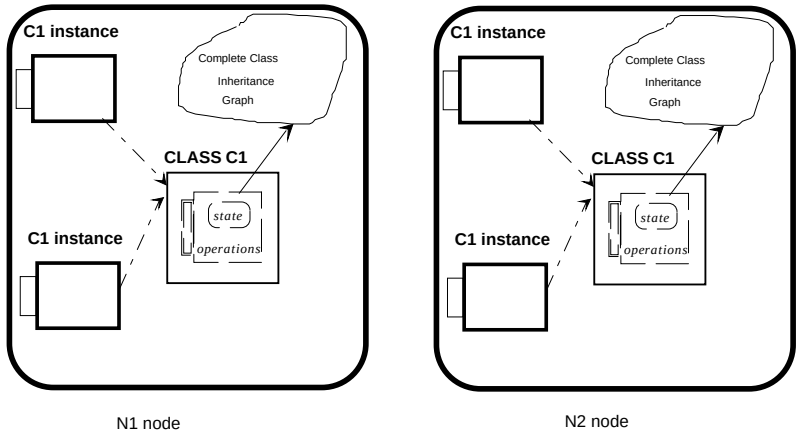
- uso di **Remote Method Invocation RMI**
  due **proxy**,
  uno dalla parte del **cliente,**
  uno dalla parte del **servitore**



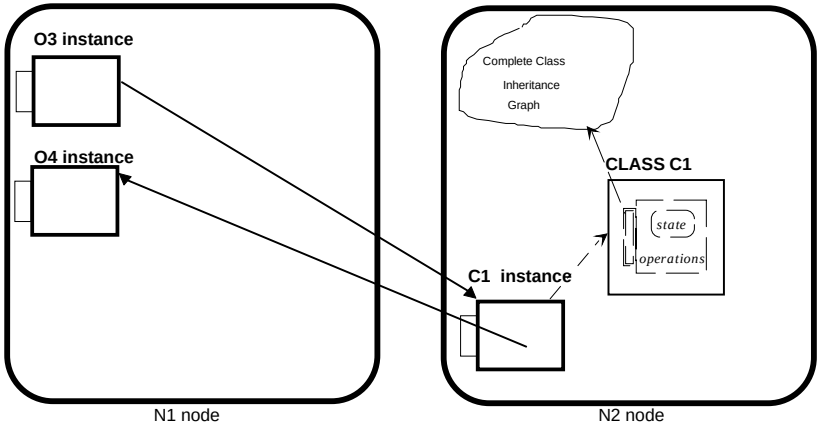
- interazione via API ad hoc (**socket**)
interazione a visibilità non **integrata** ma anche più efficiente

Distribuzione classi

Classi replicate ovunque
i **riferimenti remoti** ottenuti da un **livello aggiuntivo**



Nodi Indipendenti



O3 riferisce un oggetto di classe C1 che per rispondere deve riferire un oggetto O4
O3 e O4 corisiedono

SCENARIO FINALE

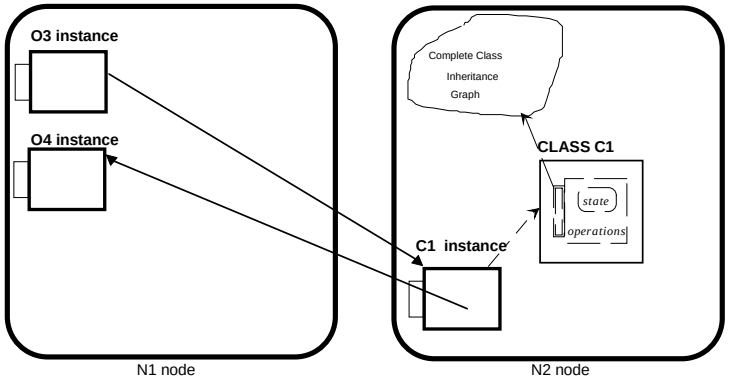
SISTEMI CON OPERAZIONI REMOTE

SISTEMI AD HOC
per il supporto alla distribuzione

gli oggetti (alcuni solamente) possono essere riferiti da remoto e visti dalle altre località che si sono create durante la esecuzione

Cliente su un nodo **Servitore su un altro**
(*abbiamo creato un livello di virtualizzazione che ci può rendere trasparente la allocazione concreta dei componenti*)

Qunid, si può eseguire in **locale o remoto**
decisioni statiche di allocazione per oggetti statici
riferimenti statici (risolti prima dell'esecuzione)



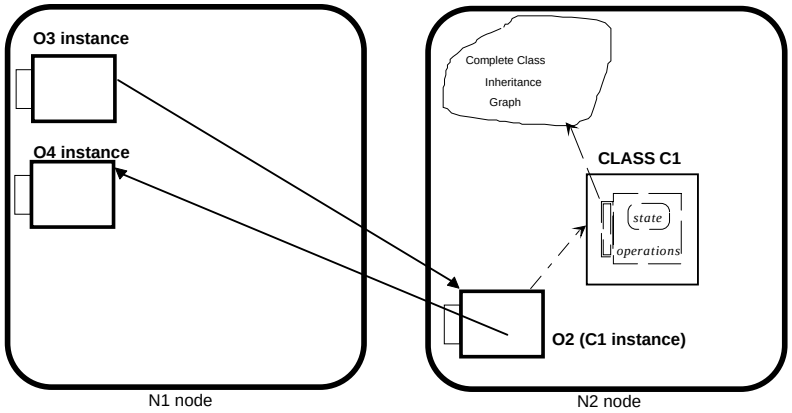
E le **esigenze** di **oggetti** da **riallocare dinamicamente?**
Oggetti che si conoscono dinamicamente
Oggetti che si muovono

Dinamicità di allocazione

movimenti anche espliciti di **oggetti** (con classi)
- e le **superclassi**
- tutte le **classi riferite** attraverso le **variabili istanza** dell'oggetto da muovere (si può ritardare)

O3 si sposta
COPIE di classi e scarto quando non sono più riferite (garbage collection) con migrazioni
- specificate dall'utente della applicazione
- decise dal sistema

E una migliore esecuzione?
La esecuzione richiesta da O3 sull'oggetto O2 di classe C1 trae vantaggio dalla presenza di O4 localmente all'oggetto istanza di C1



O2 operazione (12, 34, variabilecheriferisceO4)
O4 si muove da N1 a N2 per essere stato passato come parametro del metodo
movimento in modo definitivo o temporaneo

LEGAMI LOGICI tra OGGETTI

Le relazioni logiche tra entità possono servire per determinare **vincoli** nelle specifiche di **movimento**

sia relazioni **statiche** sia **dinamiche** da valutare solo durante la esecuzione

le statiche possono essere completamente determinate prima della esecuzione e comportare meno overhead

RELAZIONI

classi/ superclassi

classi/istanze

cliente/servitore

un oggetto con il suo stato riferisce istanze di certe classi

queste possono essere considerate solidali con l'oggetto stesso e muoversi insieme

movimento immediato (sempre)

movimento differito (su necessità)

invocazioni dinamiche

una invocazione di un metodo ha per parametri oggetti che possono essere spostati (con le classi) insieme con la richiesta

queste sono considerate solidali con l'oggetto stesso solo per la durata dalla invocazione e muoversi insieme

movimento immediato/differito

movimento temporaneo/definitivo

modelli di allocazione

le entità di una applicazione possono essere

o **statiche** o **dinamiche** o miste

Allocazione **statica**

data una specifica configurazione, le risorse sono decise prima della allocazione

Allocazione **dinamica**

la allocazione delle risorse sono decise durante la esecuzione ==> **sistemi dinamici**

In sistemi dinamici, è possibile la **riallocazione** (migrazione) le risorse possono muoversi sulla configurazione

- **APPROCCIO ESPLICITO**

===> l'utente deve specificare il mappaggio di ogni oggetto

- **APPROCCIO IMPLICITO (AUTOMATICO)**

il sistema che si occupa del mappaggio dell'applicazione sull'architettura ===> *ALLOCATION TRANSPARENCE*

APPROCCIO MISTO

il sistema adotta una ***politica di default*** (obiettivo **bilanciamento di carico computazionale)**

politica applicata sia inizialmente che dinamicamente mediante allocazione dei nuove risorse e migrazione di quelli già esistenti

se l'utente fornisce delle indicazioni, il sistema ne tiene conto nella politica di bilanciamento di carico per migliorare le prestazioni

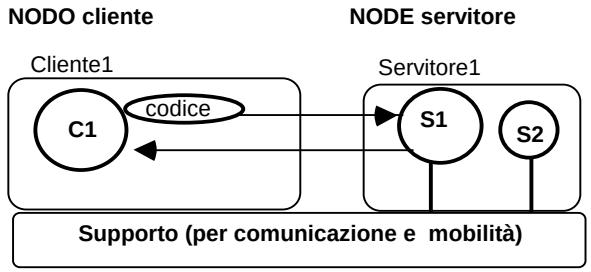
Nel caso di **agenti mobili**, allocazione **esplicita**

Verso modelli di mobilità di codice

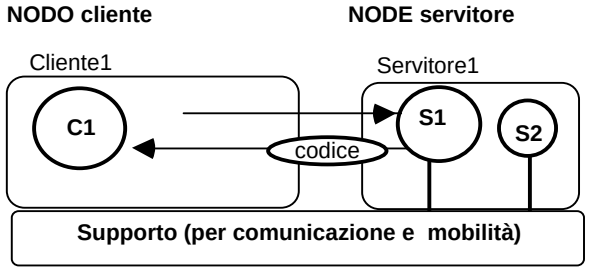
Oltre al modello **cliente servitore**, in cui si scambiano dati tra il cliente ed il servitore fornisce il servizio ed il risultato

1) **Remote Evaluation** (**REV**)
   il cliente manda al servitore anche il codice da eseguire nel servizio corrispondente
   (invio di pezzi di codice nuovo)

2) **Code On Demand** (**COD**)
   il cliente manda richiede dal servitore il codice da eseguire per un servizio che ottiene una gestione locale (**Applet** Java)



REMOTE EVALUATION



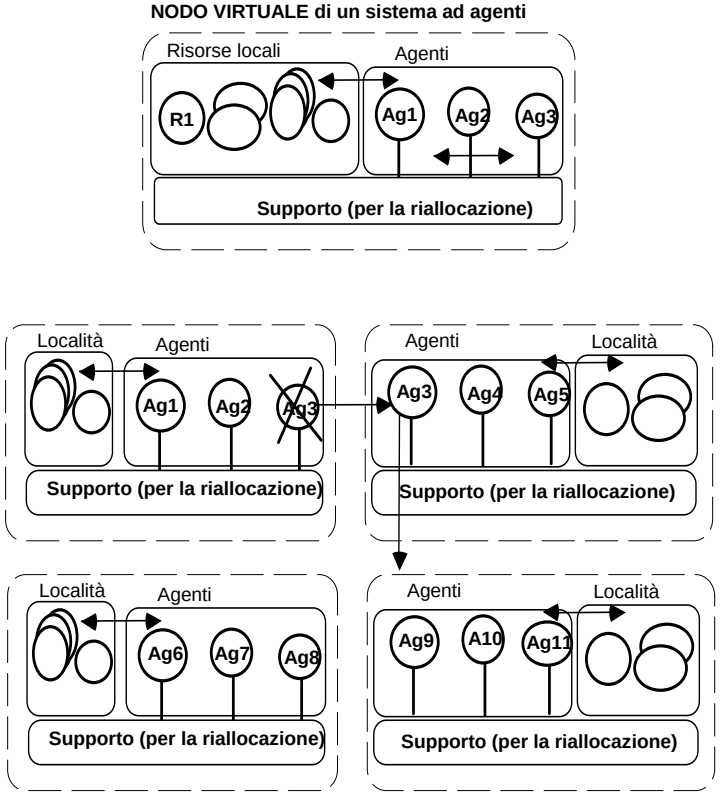
CODE ON DEMAND

Nel settore degli apparati di telecomunicazioni, si cominciano a considerare migrazioni di codice anche per router e bridge (**reti intelligenti** e **reti attive**)

SISTEMI ad AGENTI MOBILI

tecnologia in cui, una o più **attività** possono **muoversi, dopo** avere cominciato la esecuzione su un determinato nodo, cambiando la propria allocazione e mantenendo lo **stato** già **raggiunto**

INSIEME DI NODI E DI AGENTI MOBILI



Java ha problemi nel trasferire lo stato di esecuzione dei **thread** che **sono dipendenti** dalla **JVM locale**
   si pensi ai file aperti o alle variabili (oggetti) condivise

Modelli di mobilità

Mobilità debole
possibilità di eseguire delle attività su un nodo e di riprendere la esecuzione dall'inizio su un nodo diverso

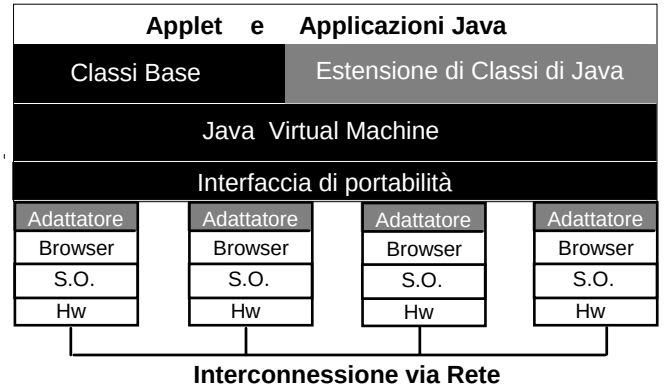
Mobilità forte
possibilità di eseguire delle attività su un nodo e di riprendere la esecuzione su un nodo diverso

MODELLI ad AGENTI MOBILI

I modelli ad **agenti** hanno rivevuto molto interesse negli ultimi anni

PROBLEMA
come trasferire il codice e farlo riprendere sul nuovo nodo, superando le eterogeneità

Soluzione ad **interprete** con un codice intermedio indipendente dalla architettura interpretato dalla macchina virtuale di ogni nodo



Sistemi nomadici e mobili
come mantenere i servizi in movimento

altri modelli di comunicazione

pipeline, farm, etc.
tuple

pattern di interconnessione concorrenti

per la decomposizione automatica

Pipeline

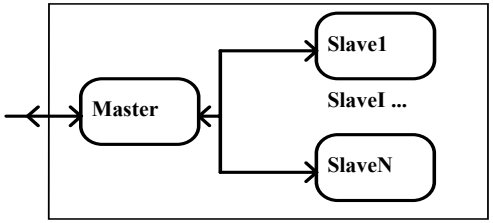
Insieme di entità che comunicano in modo ordinato:
uscita di uno stage è l'ingresso del successivo
Ogni stage lavora sui dati elaborati dal precedente

Possibilità di **concorrenza** nei servizi
di richieste diverse
Aumento del **throughput** (efficienza)

throughput numero di servizi per unità di tempo

Farm

il master è il gestore del lavoro degli slave che restituiscono l'eventuale risultato al master



molti modelli diversi di suddivisione del lavoro
con attese diverse degli slave da parte del master

Tuple

Astrazione della ***memoria condivisa*** + **comunicazione**

Lo spazio delle **tuple** è un insieme strutturato di relazioni, intese come attributi e valori

Operazioni di **scrittura e lettura** concorrenti

Accesso in base al **contenuto** degli attributi

Operazioni di In e Out

In inserisce una tupla nello spazio

Out estrae una tupla, se esiste

attende nel caso la tupla non sia ancora presente

==> una tupla con alcuni attributi non specificati

esempio

relazione messaggio (dachi, achi, corpo)

In: messaggio (P, Q, testo1)

Out: messaggio (?, Q, ?)

tuple meccanismo generale

per la *comunicazione* e *sincronizzazione*

Linda (Gelertner)

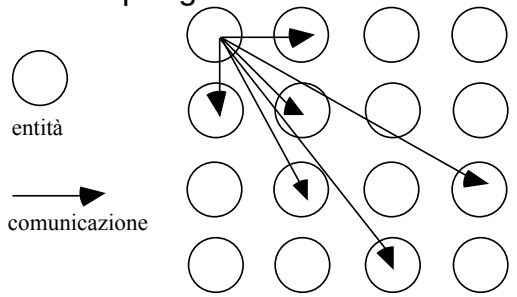
realizzazione della memoria condivisa a confronto con lo scambio di messaggi

MODELLO di interconnessione

località vs. globalità

modelli **globali** **(NON VINCOLATI)**

non impongono restrizioni alle interazioni

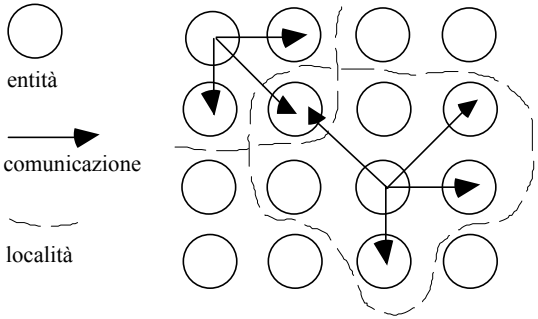


operazioni non scalabili

dipendenti dal diametro del sistema

modelli **locali** **(RISTRETTI, VINCOLATI)**
NON TRASPARENTI
ACCESSO NON UNIFORME

prevedono una limitazione alla interazione



operazioni (forse) scalabili

poco *dipendenti dal diametro del sistema*

verso la **località**,si definiscono i **domini**, i **vincoli**

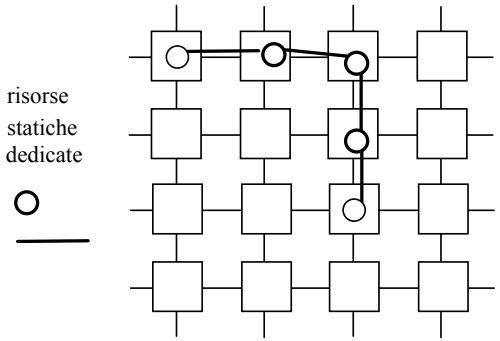
per migliorare la **scalabilità**

modello di interconnessione

a connessione/ senza connessione

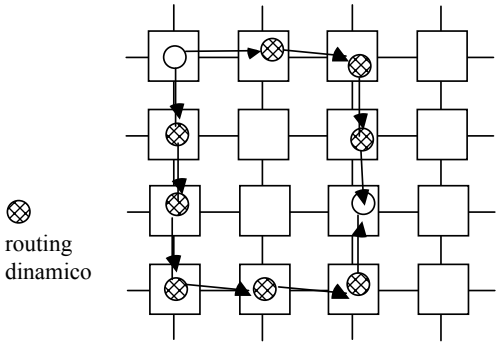
CONNESSIONE

le entità stabiliscono di preallocare le risorse per la comunicazione (statico)



SENZA CONNESSIONE

le entità comunicano utilizzando le risorse che sono disponibili al momento della azione (dinamico)



Alcuni modelli, detti a connessione (**TCP**), non impegnano risorse intermedie

Vedremo questi comportamenti meglio più avanti...

modelli di guasto

errore qualunque comportamento diverso da quello previsto dai requisiti (**fault**)

failure comportamento nel sistema che può causare errori

Programma che sbaglia e
aggiorna un database in modo sbagliato

fault ==> transienti, intermittenti, permanenti

Bohrbug ⇒ errori ripetibili, sicuri, e spesso facili da correggere

Eisenbug ⇒ errori poco ripetibili, difficili da riprodurre e difficili da correggere

Gli Eisenbug sono spesso legati a problemi transienti e quindi molto difficili da eliminare

Cliente.Servitore **identificazione** e **controllo** reciproci

Strategie di

Identificazione dell'errore
Recovery dell'errore

Numero di errori che si possono tollerare

(insieme o durante il protocollo di recovery)

Ipotesi sui guasti semplificano il trattamento

Un guasto alla volta (**Guasto Singolo**)

il tempo di identificazione e recovery deve essere inferiore (**TTR** Time to Repair) al tempo tra due guasti (**TBF** o Time Between Failure o **MTBF Mean TBF**)

Con **3** copie si tollera un guasto, due sono identificati

Generalizzazione

con **3t + 1** copie, si tollerano **t** guasti

Protocolli necessari

I protocolli impegnano le risorse
- **durata degli algoritmi**
- **realizzazione degli algoritmi (affidabilità)**

Il supporto (hw e sw) per il protocollo di recovery deve essere più affidabile del resto *(come si garantisce?)*

Principio di minima intrusione

limitare l'impegno di risorse (*overhead*) introdotto dai livelli di supporto di sistema

Definizioni

DEPENDABILITY FAULT TOLERANCE

possibilità di affidarsi al sistema nel completamento di quanto richiesto
sia in senso hardware, sia software, in generale
confidenza per ogni aspetto progettuale

RELIABILITY AFFIDABILITÀ

possibilità del sistema di fornire risposte corrette
(accento sulla *correttezza* delle risposte)
es. disco per salvare dati ==> tempo di risposta?

AVAILABILITY (continuità)

possibilità del sistema di fornire comunque risposte in un tempo limitato *accento sul tempo di risposta*
replicazione con copie attive e sempre disponibili

RECOVERABILITY

Consistenza, Sicurezza, Privacy, ...

Reliability

MTBF mean time between failures disponibilità della risorsa
MTTR mean time to repair indisponibilità

Availability **A = MTBF / (MTBF + MTTR)**

diversa per letture e scritture
se si hanno copie, la lettura può essere fornita anche se una o più copie sono non disponibili

Reliability probabilità di servizio disponibile in Δt
R (Δt) = reliable sul tempo Δt
R(0) = A, in modo generale come limite

Proprietà formali

Correttezza garanzia che non si verifichino **problemi**
Safety invarianti sempre rispettati

Liveness raggiungimento obiettivi con **successo**
Vitalità vedi terminazione

- Un sistema con **safety** e **liveness** maschera i fault
- Un sistema senza **safety** e **liveness** non garantisce niente per quel fault specifico (**nessuna tolleranza**)

  - Un sistema con **safety** e senza **liveness** opera **sempre in modo corretto**
  - Un sistema senza **safety** e con **liveness** opera fornendo un **risultato**, anche **non corretto**

Soluzioni per entrambe

Ridondanza in **spazio** o in **tempo**

ipotesi di fault nel **distribuito**

FAIL-STOP

un processore fallisce fermandosi (halt) e gli altri possono verificarne lo stato

FAIL-SAFE (CRASH o HALT)

un processore fallisce fermandosi (halt) e gli altri possono non conoscerne lo stato

FAILURE BIZANTINE

un processore fallisce esibendo un qualunque comportamento

SEND & RECEIVE OMISSION

un processore fallisce ricevendo/trasmettendo solo una parte dei messaggi che dovrebbe trattare

GENERAL OMISSION

un processore fallisce ricevendo/trasmettendo solo una parte dei messaggi che dovrebbe trattare o fermandosi

NETWORK FAILURE

l'interconnessione non garantisce comportamenti corretti

NETWORK PARTITION

la rete di interconnessione può partizionarsi, perdendo ogni collegamento tra le due parti

Replicazione come soluzione e
per la realizzazione di componenti

MEMORIA STABILE

uso di replicazione per ottenere certezza di non perdere informazioni (***uso di disco***)

Ipotesi di guasto limitative: *si considera di avere un sistema con una probabilità trascurabile di guasti multipli su banchi di memoria scorrelati*

In ogni caso, la probabilità di guasto durante un eventuale protocollo di recovery deve essere **minima**

Memoria con blocchi sicuri: ogni errore viene trasformato in **omissione** (*codice di controllo associato al blocco e blocco guasto*)
I blocchi sono associati in **due** copie distinte su dischi a *bassa probabilità di **errore congiunto***: le due copie hanno lo stesso contenuto di informazioni

Ogni **lettura** da un blocco **errato** viene trasformata in failure di omissione
La lettura procede da una delle copie e poi dall'altra.
Se uno dei blocchi fallisce, si tenta il recovery

Ogni **scrittura** procede su una delle copie e poi sull'altra

In caso di **recovery**, si attiva un protocollo che ripristina uno stato consistente:
se le copie sono uguali nessuna azione,
se una scorretta si copia il valore dell'altra,
se diverse e corrette si ripristina la consistenza

Costo elevato della realizzazione

Costi della replicazione di RISORSE

costi molto elevati per la implementazione in due dimensioni
spazio sia in **risorse** a disposizione
tempo sia in **tempo** di risposta
Spesso le ipotesi di guasto più ampie rendono
difficile il progetto dei protocolli e
inaccettabile il costo delle soluzioni

costi dipendenti da molti fattori diversi
costi di memoria
overhead di communication overhead
complessità di implementazione
cosa replicare, quante repliche, dove allocarle, etc.

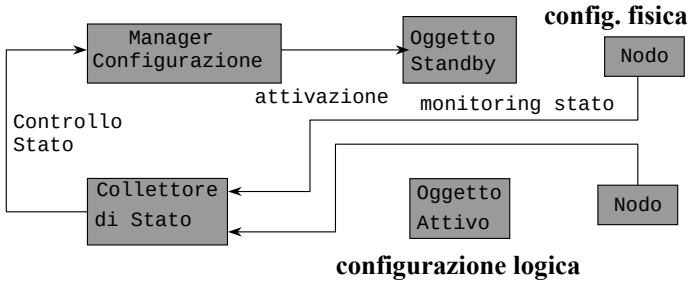
TREND

Sistemi special-purpose
TANDEM
replicazione in doppio di ogni unità di sistema (due processori, due bus, due dischi)
Obiettivo: sistema **fail-safe**
Un errore viene identificato attraverso la replicazione di CPU (doppio caldo)
Memoria su disco stabile con accesso attraverso uno di due bus a blocchi di memoria replicati
costo del sistema molto elevato

La presenza di copie **replicate** può prevedere
- di fare le azioni in **doppio** o
- di farle solo **una volta** e riportare le variazioni sulla copia (o sulle copie)

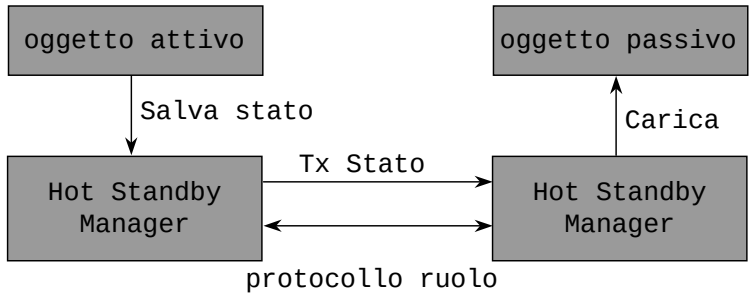
Sistemi Stand by
Cold stand by (copie fredde)

Ogni oggetto implementato da una **sola istanza**
solo al fallimento si rigenera una **nuova istanza**
☹ Nessuna informazione di stato viene mantenuta tra attivazioni diverse di istanze
☹ Periodo di riconfigurazione elevato



Hot stand by (copie calde)

Un oggetto consiste di almeno **due istanze**
Una istanza esegue ed è *attiva*, l'altra è *passiva*
la copia **attiva** (master) esegue e
quella **passivo** si mantiene aggiornato
Se fallisce la copia attiva, la passiva viene promossa
Se fallisce la copia passiva
si usano copie passive di stand by per fare una nuova copia passiva che ottiene lo stato dall'oggetto attivo



Sistemi general-purpose

replicazione su memoria di massa diventata comune

RAID

Redundant **A**rray of **I**nexpensive **D**isk
insieme di dischi a basso costo ma coordinati in azioni per ottenerne diversi standard di tolleranza ai guasti

costo limitato
spesso considerati in sistemi commerciali

RAID nati per striping, ossia per velocizzare l'accesso ai file memorizzati su più di un disco

Raid 0: sterpigni	elevata velocità I/O parallelo	nessuna ridondanza, peggior MTBF	applicazioni con I/O intensivo
Raid 1: mirroring	alta availability, buone perf. in lettura	alto costo **ridondanza massima**	alto numero letture (transaction)
Raid 3: striping **con** disco di parità dedicato	alta velocità tx per blocchi di grandi dimensioni	volumi molto grandi, un I/O alla volta	trasferimento grossi blocchi (immagini)
Raid 5: striping **senza** disco di parità dedicato **parità distribuita**	buona velocità molte letture di piccoli blocchi scritture di grossi blocchi	read before write per avere performance	I/O misto con buona velocità di trasferimento (LAN server)

Anche sistemi a **cluster di processori** equivalenti

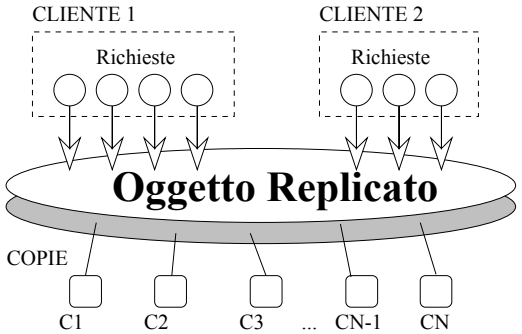
GESTIONE DELLE RISORSE

Possibilità di risorse nel distribuito

risorse partizionate
le risorse sono allocate in modo unico su nodi diversi

risorse replicate
copie multiple delle risorse su nodi
Uso della replicazione
dati replicati
processi replicati

Astrazione di **risorsa unica**

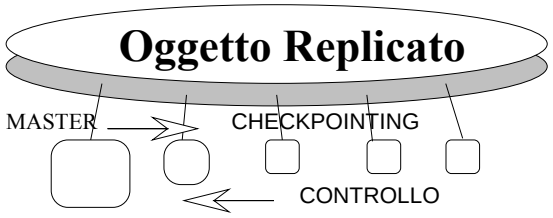


grado di replicazione
numero di copie della entità da replicare

maggiore il **numero** di **copie**
maggiore la **ridondanza**
migliore le proprietà di **affidabilità** e **disponibilità**
reliability & availability

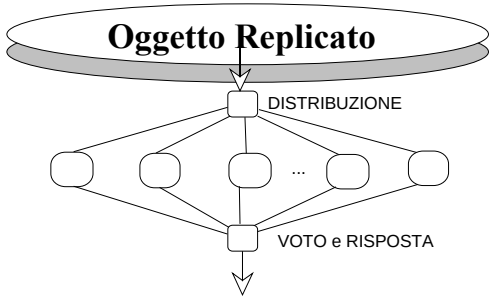
DUE TIPI PRINCIPALI DI REPLICAZIONE

Modello Passivo
Una sola copia esegue,
le altre sono solo di riserva



Il master fornisce i servizi e aggiorna gli slave
Gli slave controllano il master

Modello Attivo
Tutte le copie eseguono
con problemi di coordinamento



Nella **TMR** (Triple Modular Redundancy) **tre** copie
si ovvia ad un errore e se ne identificano fino a due

Le diverse copie possono anche eseguire algoritmi diversi

modelli di replicazione

Modelli
Master Slave (modello passivo)
Copie attive (modello attivo)

modello passivo

master/slave (primary/backup)
Un **solo processo** (attivo) esegue le azioni sui dati
La altre copie (passive) servono in caso di guasto
una sola copia corretta, le altre anche non aggiornate

Scollamento dello stato tra il **master** e gli **slave**
*In caso di **guasto**, si dovrebbe riprendere dall'**inizio***
Il master aggiorna lo stato degli slave (**checkpoint**)

La **politica** separa
le azioni necessarie per garantire una risposta sicura:
aggiornamento copia **primaria**
dalle azioni successive
aggiornamento copie **secondarie** (*checkpointing*)

STATO DELLE COPIE

Il cliente ottiene una risposta con **tempi inferiori**
gestione con protocolli all'interno dell'oggetto
checkpointing ==> *azione di aggiornamento*

checkpoint

azione di aggiornamento e di passaggio
del nuovo stato sulle copie slave

Azione *periodica* **time-driven**
Azione *scatenata da evento* **event-driven**

In caso la risorsa sia **sequenziale**,
lo stato è evidente e '**facile**' da identificare e instaurare

In caso di **risorsa parallela**, bisogna pensare a tutte le operazioni che possono essere in atto insieme
lo stato dovuto alle diverse **azioni contemporanee** è più difficile da identificare e instaurare

RECOVERY DEL GUASTO

Rilevazione del guasto: da parte di chi? quando?
Recovery del guasto

La copie secondarie devono identificare il guasto tramite una osservazione del master

- *uso di messaggi esistenti e assunzioni sugli intervalli di tempo*
- *messaggi ad-hoc per stimolare risposte dal master*

Può bastare
uno slave per il protocollo (*se guasto singolo*)
gerarchia di slave e protocolli molto più complessi
(*se guasto multiplo*)

La **risorsa**, vista dall'esterno, tollera, a secondo delle strategie, un numero diverso di errori ed è in grado di fornire operatività in caso di errori (**fault transparency**)

MODELLO ATTIVO

copie attive
Un processo per ogni copia di dato esegue la operazione sui dati privati
Comunicazione del cliente con i servitori in modo
esplicito od **implicito**

Se il controllo del cliente esplicito => manca astrazione

Controllo **implicito interno alla risorsa**
o esiste un **solo gestore** (schema **statico**)
schema a **farm** centralizzato
e poi da lì si dirigono le richieste verso gli altri

o esiste un **gestore** (schema **dinamico**)
per ogni richiesta un gestore diverso
da lì si dirigono le richieste verso gli altri

gestori decisi per **località** / **a rotazione**
necessità di evitare interferenza di **gestori**

In genere, nei modelli attivi
perfetta sincronia
tutte le copie devono agire d'accordo, soprattutto in caso di risposta o di azioni verso l'esterno (innestamento)
approssimazioni alla sincronia meno costose
quasi tutte lo copie devono essere in accordo,
ma le azioni possono procedere prima del completo accordo

Queste strategie meno sincrone sono meno costose
in tempo di risposta per il cliente

COORDINAMENTO TRA LE COPIE

Ogni azione richiede di aggiornare lo stato di tutti
azione di aggiornamento
prima di fornire il risultato

Se si lavora con gestori diversi è compito del gestore comandare le azioni degli altri
decisioni sulla durata dell'azione
*In caso di **guasto durante una operazione**, si deve dichiarare il guasto e potere continuare a fornire una qualche risposta*

ACCORDO prima di dare risposta
Tutte le copie devono avere fatto l'azione
Voting (non tutte la copie)
a ***maggioranza* (quorum)**
pesati
le copie non aggiornate sono in uno stato non utilizzabile immediatamente, ma solo dopo un reinserimento nel gruppo (recovery)

Rilevazione del guasto: da parte di chi? quando?
Rilevazione del reinserimento: da parte di chi? quando?
Necessità di monitoraggio e di **controllo esecuzione**

Semantica di gruppo
Si possono approssimare semantiche molto diverse *anche nei confronti delle azioni che si attuano nel gruppo e degli ordini di esecuzione delle azioni*

modello di servizio
qualità della connessione e servizio

QoS qualità del servizio

Prontezza di risposta
ritardo, tempo di risposta,
jitter (definito come variazione del **ritardo di consegna**)
Banda
bit o byte al secondo di sistema e di applicazione
Throughout
numero di operazioni al secondo
Affidabilità
percentuale di successi/insuccessi

molti aspetti legati alla qualità del servizio anche
non funzionali ma **dipendenti**
dalla applicazione specifica o
da fattori esterni

? *perché interesse?*
stream di informazioni audio e video
con cui cominciano a giocare fattori real-time
banda trasmissiva, ritardi ammissibili, variazioni
jitter *variazioni di ritardo ammissibile*

le entità stabiliscono di garantire una certa **banda** e **tempi di risposta** per i servizi ripetuti o di flusso

QoS

In caso di sistemi multimediali, o
flussi continui di informazioni video
Video on Demand (VoD)
erogazione di servizi video via una infrastruttura

anche **proprietà non funzionali**
- **dettagli dell'immagine**
- **accuratezza dell'immagine**
- **velocità di risposta**
- **sincronizzazione audio/video**

...

In genere, una QoS può essere garantita solo attraverso un accordo **negoziato** e **controllato**
osservando il sistema in esecuzione e
adeguando dinamicamente il servizio

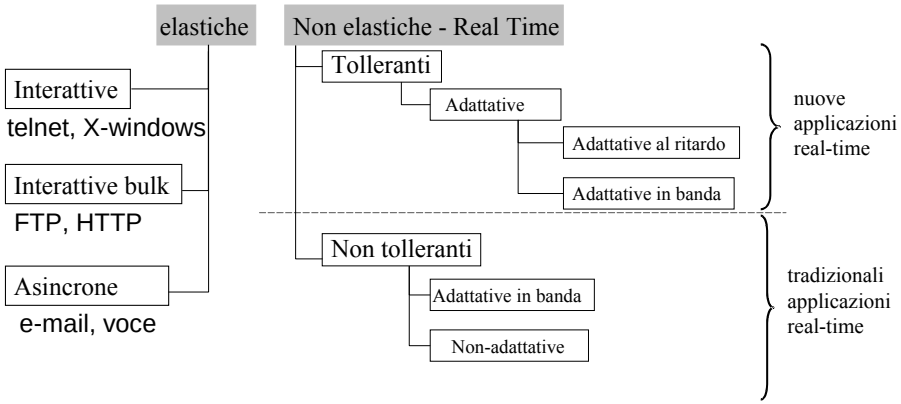
SENZA CONNESSIONE
le entità comunicano utilizzando le risorse che sono disponibili al momento della azione (dinamico)
come garantire il flusso
CON CONNESSIONE TCP/IP
le entità che comunicano con connessione TCP, non impegnano risorse e non possono dare garanzie
IN OSI
le entità OSI impegnano risorse e possono anche fornire garanzie, che devono essere rispettate da tutte le entità del percorso

Come garantire QoS in TCP/IP?

requisiti di qualità delle applicazioni

da garantire a livelli diversi (OSI **N/T**)

Applicazioni Elastiche e Non Elastiche



Le **elastiche** non introducono vincoli di qualità
seppure con requisiti diversi indipendenti da ritardi
pure lavorando meglio con ritardi bassi
Interattive meglio con ritardi inferiori a 200ms

Le **non elastiche** hanno necessità di garanzie
e presentano tipicamente **vincoli di tempo**
e sono poco tolleranti per essere **usabili**

Il **livelli di servizio** si possono adeguare ai requisiti
adattative al **ritardo** => audio scarta pacchetti
adattative alla **banda** => video che si adatta la qualità

Management QoS

La buona gestione si può ottenere solo con azioni
- sia di tipo statico,
- sia di tipo dinamico

Sia azioni **statiche** **PRIMA DELLA EROGAZIONE**
- **specifica** dei requisiti e variazioni
- **negoziazione**
- **controllo di ammissione**
- **riserva e impegno delle risorse** necessarie

Sia azioni **dinamiche** **DURANTE LA EROGAZIONE**
- **monitoring** delle proprietà e delle variazioni
- **controllo** del rispetto e **sincronizzazione**
- **variazione** delle risorse per mantenere QoS
- **rinegoziazione** delle **risorse** necessarie
- **adattamento** a nuove situazioni

sono necessari dei modelli precisi di gestione

modello di monitor e qualità

Problema del costo della strumentazione
Necessità di avere meccanismi di raccolta di dati dinamici e di politiche che non incidano troppo sulle risorse (usate anche dalle applicazioni)

Principio di minima intrusione

SERVIZI

best-effort adatto per servizi elastici
vedi servizi Internet
nessun throughput garantito, ritardi qualunque, non controllo duplicazioni o garanzie di ordine azioni

controlled load
simili a best-effort con basso carico
ma con **limiti superiori** al ritardo
servizi elastici e real-time tolleranti

guaranteed load
limite stretto al ritardo
ma non limiti al jitter (alle fluttuazioni)
servizi real-time non tolleranti

IP => best-effort
TCP => elastico con garanzie di ordinamento, unicità, controllo flusso
OSI => le qualità di servizio di N e T
Naturalmente, le **garanzie di qualità di servizio** hanno un **costo**

Transizione
da **infrastruttura a basso costo e basse prestazioni**
⇒ a **infrastruttura**
a costi differenziati e prestazioni corrispondenti

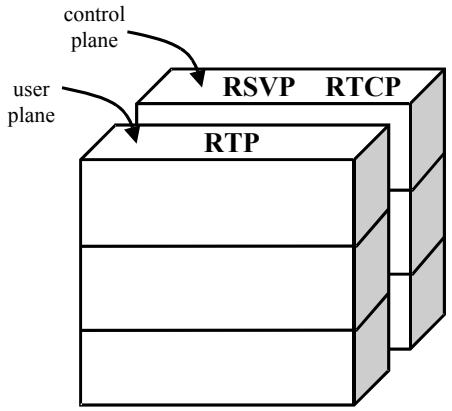
Servizi **Integrati** a livello di **singolo** flusso
Servizi **Differenziati** **aggregando flussi** per qualità
http://www.rfc-editor.org/

Operazione e Controllo delle operazioni

Necessità di accoppiare il piano operativo (o utente) con gli strumenti di controllo della operatività

Piano **User** per
protocolli utente

Piano di **Controllo**
per la gestione e
la negoziazione



EVOLUZIONE DEI PROTOCOLLI

Uso di **nuovi protocolli** per adeguare anche Internet per il controllo delle **operazioni** e **risorse** mantenendo la **caratteristica best-effort**

RSVP=> Resource Reservation Setup Protocol
(RFC 2205) protocollo per **riservare e garantire risorse** sui nodi intermedi

RTP => Real-Time Protocol
(RFC 1889) formato di messaggi generali in datagrammi UDP invio con **reliability** e **multicast**

RTCP=> Real-Time Control Protocol
segnalazione e controllo per mantenere QoS negoziato

Trasparenza (Network Transparency)

Accesso
omogeneità accesso alle risorse locali e remote

Allocazione
locazione delle risorse locali e remote trasparente

Nome
nome non dipende dal nodo di residenza

Esecuzione
capacità di eseguire usando risorse locali e remote

Performance
non degrado nell'uso dei servizi locali o remoti

Fault
capacità di fornire servizi in caso di guasto

Replicazione

VANTAGGI
Più facile sviluppo delle applicazioni e maggiore affidabilità
Sviluppo dinamico ed incrementale
Maggiore complessità

necessità di **STANDARD**

Eterogeneità come diversità di
Hardware
Rete di interconnessione
Sistema Operativo

TEORIA E MODELLI

Classificazioni diverse delle **architetture** e dei **modelli**
alla base del parallelismo

MODELLO
Astrazione della architettura
==>

Possibilità di **guidare nel**
progetto della soluzione

Associato ad una ***metodologia*** di sviluppo
Costo della possibile soluzione
Efficienza implementativa

Alcuni modelli standard di riferimento

Modello ISO/OSI
sono inizialmente specifiche astratte di standardizzazione
non *sono modelli operazionali*

Modello CORBA OMG, DCE OSF
sono proposte di standard di comitato / fatto

Modello COM, DCOM, SOAP
sono proposte dagli specifici produttori

Nuove proposte di standard

TINA-C

Telecommunications **I**nformation **N**eworking **A**rchitecture
Consortium 1993
Convergenza di aziende di diversi settori
provider, fornitori, utilizzatori di servizi

considerando mercati globali
- **telecomunicazioni**
- **internet**
- **multimedia**

esplosione di servizi mobile
integrando voce, dati & video su Internet,
Insieme di architetture
per modellare i diversi aspetti del progetto di una **architettura integrata**

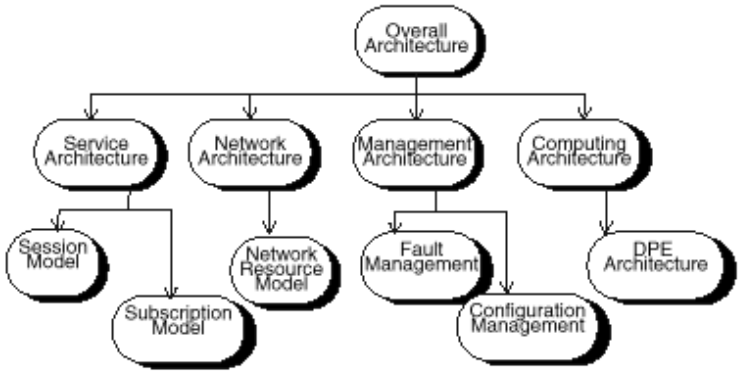


Figure 2-1. Decomposition of the TINA architecture

TINA 2000

Modelli di servizio e loro disponibilità
accordo e negoziazione

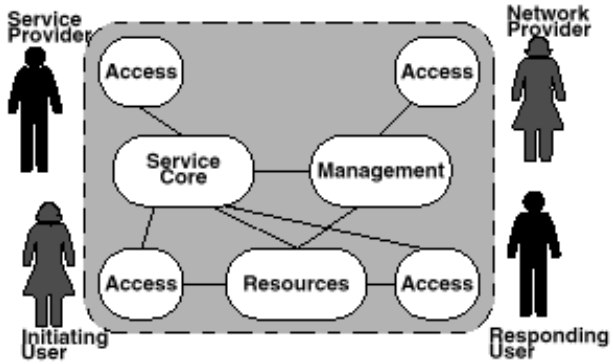


Figure 5-2. Generic Service Model

Architetture fondamentali separate ed interagenti:
Computing, **Service**, **Network Architecture**

Visione trasparente delle applicazioni e dei servizi

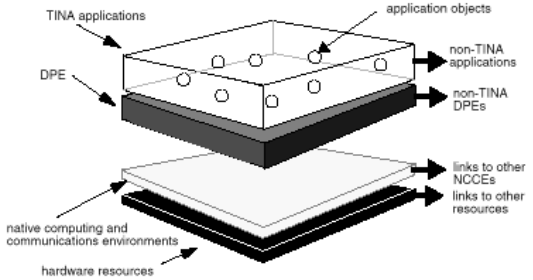


Figure 3-2. Basic structure of telecommunications software in a TINA environment

DPE Distributed Processing Environment
NCCE Native Computing Communication Environment

TINA implementazione

Una applicazione è basata su
- **Servizi**
- **Risorse**
- **Elementi**

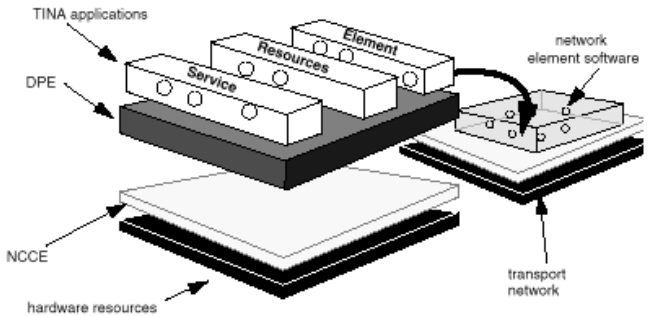


Figure 5-1. Layers and separations in TINA

Visione non trasparente
per il progetto e durante la implementazione

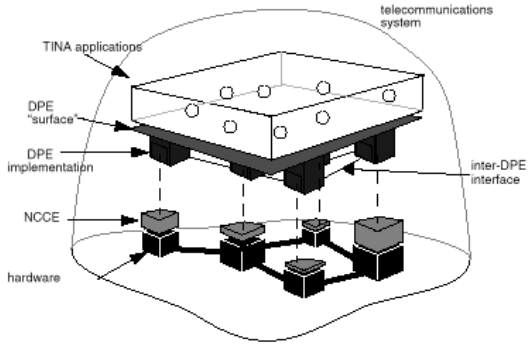


Figure 3-3. Underlying nodes of TINA systems

Modelli **RAM**

Una macchina ad accesso random
(**R**andom **A**ccess **M**achine)
è costituita da:
- **un** programma inalterabile composto di istruzioni in sequenza;
- **una** memoria composta di una sequenza di parole, ciascuna capace di contenere un intero;
- **un** solo accumulatore capace di operare;
- **un** nastro di input (read-only);
- **un** nastro di output (write-only);

Durante un passo la RAM esegue una istruzione
in sequenza, a parte i salti.

Istruzioni sono, ad esempio:
read, write, load, store, add, sub, mul, div, jump (acc), ..., halt

Modi di indirizzamento:
immediato, diretto, indiretto

La RAM è una macchina special-purpose per la risoluzione di un problema specifico

limiti:
- non c'è limite alla memoria di programma
- le istruzioni richiedono lo stesso tempo

(vedi SIMD)

Modelli PRAM

Una macchina parallela ad accesso random
(**P**arallel **R**andom **A**ccess **M**achine)
è costituita da una collezione di RAM.
Una PRAM si dimensione P è composta da:
- **P** programmi inalterabili fatto di istruzioni in sequenza;
- **una sola** memoria composta di una sequenza di parole capaci di contenere un intero;
- **P** accumulatori capace di operare;
- **un** nastro di input ed **uno** di output;

Durante un passo, la PRAM esegue P istruzioni, una per ogni programma
(modello MIMD o meglio MIMD sincrono)

DISTINZIONE semantica
come si eseguono le operazioni su una stessa cella di memoria (operazioni di lettura/scrittura)
operazioni ***simultanee*** (**C**oncorrenti)
operazioni ***sequenzializzate*** (**E**sclusive)

	lettura	scrittura
CREW	concorrente	esclusiva
EREW	esclusiva	esclusiva
CRCW	concorrente	concorrente
ERCW	esclusiva	concorrente

Concurrent **R**ead **E**xclusive **W**rite (**CREW** PRAM)
Exclusive **R**ead **E**xclusive **W**rite (**EREW** PRAM)
Concurrent **R**ead **C**oncurrent **W**rite (**CRCW** PRAM)

Exclusive **R**ead **C**oncurrent **W**rite (**ERCW** PRAM)
vedi MISD

Operazioni concorrenti: quali effetti?

Principio di serializzazione ==>
gli effetti come se una fosse fatta per prima, poi un'altra in successione, un'altra ancora, etc.

limiti:
- le istruzioni richiedono lo stesso tempo
- il tempo di operazione non dipende da P
  accesso a **memoria comune** in un tempo che non cresce con P
- il tempo di operazione di input/output trascurato
  accesso ad un unico nastro non cresce con P

I/O bottleneck

➪ **Uso di concorrenza per la gestione del sottosistema I/O**

Modelli message passing MP-RAM

Una macchina message passing ad accesso random
(**M**essage **P**arallel **R**andom **A**ccess **M**achine)
è costituita da una collezione di RAM ciascuna con una memoria privata e connesse da canali punto a punto.
Una MP-RAM si dimensione P è composta da:
- **P** programmi inalterabili fatto di istruzioni in sequenza;
- **P** memorie composta di una sequenza di parole;
- **P** accumulatori capaci di operare sulla propria memoria;
- **un** nastro di input ed **uno** di output;
- **un** grafo di interconnessione punto-a-punto.

Per esempio ad albero, a stella, etc.
Ogni nodo ha un certo *numero* di vicini ed un *grado di interconnessione*
Maggiore il grado di interconnessione, maggiore il costo della architettura, minore necessità di routing intermedio
Possibilità di comunicazione bidirezionale

Le istruzioni sono accresciute con:
send e *receive neighbor*

Durante un passo, la MP-RAM esegue P istruzioni, una per ogni programma

DISTINZIONE semantica
cosa succede se una receive arriva prima della send sul canale corrispondente?
semantica sincrona

limiti:
modello espressivo come la PRAM
modello meno "potente" della PRAM

MP-RAM introduce località
PRAM è ancora un modello globale

modello PRAM può emulare il modello MP-RAM
la memoria comune viene vista come un insieme di spazi per i diversi canali
Il canale come lo spazio per il dato ed un flag
send e receive come istruzioni che agiscono sul flag e poi sul valore da ricevere/inviare

vice versa
la realizzazione del **multicast/broadcast** *richiede passi intermedi e router intermedi*

Problema
Il modello PRAM è troppo *potente ed astratto* **(globale)**
Gli algoritmi sono più veloci sul modello che nei sistemi reali modellati

COMPLESSITÀ DEGLI ALGORITMI

dipendenza dalla dimensione del problema **N**

complessità in tempo CT(n) (abbreviato in **T(N)**)
complessità in spazio CS(n)

COMPLESSITÀ
T(1,N) soluzione sequenziale
T(P,N) soluzione parallela con P processori

SPEED-UP
Miglioramento dal sequenziale al parallelo

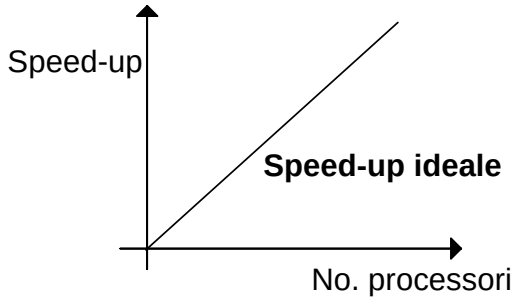
$$S(P,N) = \frac{T(1,N)}{T(P,N)} \qquad S_P(N) = \frac{T_1(N)}{T_P(N)}$$

EFFICIENZA
Uso delle risorse

$$E(P,N) = \frac{\text{Speed up}}{\text{Numero Processori}} = \frac{S_P(N)}{P}$$

$$E_P(N) = \frac{T_1(N)}{P\ T_P(N)}$$

$S_P(N)$ al massimo P ed $E_P(N)$ al massimo 1



ragioniamo sui valori medi

Correlazione tra N e P

possibilità di considerare

N indipendente da P
N dipendente da P

FATTORE DI CARICO (LOADING FACTOR)

$$L = \frac{N}{P}$$

dependent size (N funzione di P)
independent size
identity size (N == P)

OBIETTIVO

trovare la migliore approssimazione per l'algoritmo che ci interessa verificare

anche

Heavily Loaded Limit

$T_{HL}(N) = \inf_P T_P(N)$

il valore di ***P*** tale per cui si ottiene la minore complessità del problema (cioè T minimo)

in genere, per **N** molto **elevato**,
cioè ogni processore molto caricato

SPEED-UP

Massimo speed-up ottenibile
nel passaggio dal sequenziale al parallelo

LEGGE DI AMDHAL

Un programma è diviso in una parte *parallela* ed
in una parte *sequenziale*
lo speed-up è **limitato dalla parte sequenziale**

Se un programma è costituito di 100 operazioni e solo
80 possono andare in parallelo e
20 richiedono sequenzialità
Anche con 80 processori ==>
lo speed-up non può arrivare sopra a 5

Situazione ideale ==>
considerando la migliore allocazione possibile

$T_P(N) = T_{CompP} + T_{CommP}$

$T_{CompP} = T_{CompPar} + T_{CommSeq}$

Limite di Amdhal rapporto tra le due parti dell'algoritmo

Tenendo presente casi di speed-up anomalo
==> *dipendenti dall'algoritmo*

Problema di dimensione N con uso di P processori
per il caso di problema di *somma di N interi*

Complessità del sequenziale O(N)

Complessità del modello parallelo
identity size (N == P)

Con un numero di processori pari a P
Ogni nodo foglia contiene un valore
Interconnessioni ad albero

$N = P = 2^{H+1}-1$
$H = O(\log P) = O(\log N)$
$T_P(N) = O(H) = O(\log N)$

I valori fluiscono dalle foglie in su ed
ogni nodo li somma ad ogni passo

$$L = \frac{N}{P} = 1$$

$$\mathbf{S_P(N)} = \frac{T_1(N)}{T_P(N)} = \frac{O(N)}{O(\log N)} = O\left(\frac{N}{\log N}\right) = O\left(\frac{P}{\log P}\right)$$

$$\mathbf{E_P(N)} = \frac{T_1(N)}{P\ T_P(N)} = O\left(\frac{1}{\log N}\right) = O\left(\frac{1}{\log P}\right)$$

Efficienza tende a zero

Complessità del modello parallelo
independent size

Se possiamo dividere il problema parallelizzandolo con un certo fattore di carico
Una fase locale di lavoro ed una fase di scambio di informazioni per combinare i risultati parziali

$L = N/P$
$T(P,N) = O(N/P + \log P) = O(L + \log P)$

per lo speed-up

$$\mathbf{S_P(N)} = \frac{T_1(N)}{T_P(N)} = O\left(\frac{N}{N/P + \log P}\right) = O\left(\frac{P}{1 + P/N \log P}\right)$$

per l'efficienza

$$\mathbf{E_P(N)} = O\left(\frac{1}{1 + P/N \log P}\right)$$

Se $N \gg P$
allora lo speed-up può diventare anche P (efficienza 1)

Se $P \gg N$
allora **inefficienza**

Complessità del modello parallelo

heavily loaded limit

L cresce

$\mathbf{T_{P\ HL}(P,N)} = O(L + \log P) \Rightarrow O_{HL}(L)$

$$\mathbf{S_{P\ HL}(N)} = \frac{O(LP)}{O(L + \log P)} \Rightarrow O_{HL}(P)$$

$\mathbf{E_{P\ HL}(N)} = \Rightarrow O_{HL}(1)$

Cioè, intuitivamente

se carichiamo ogni nodo molto ==>
L molto elevato

Si può raggiungere anche uno speed-up ideale
ed una efficienza ideale

PROBLEMI

- schematizziamo a meno di fattori costanti
- a volte il caso peggiore può essere più significativo
  trascuriamo completamente
- trascuriamo
  movimento di dati *I/O* e il *mappaggio*
  sono necessarie comunicazioni

MAPPAGGIO

Assumiamo di avere mappato il problema nel modo migliore

Spesso non si possono fare allocazioni così facili
problemi dinamici nella comunicazione
dopo la allocazione

A volte si usa anche:
funzione di **Overhead Totale T_0**
cioè tenendo in conto
*le **risorse** ed il **tempo** speso in **comunicazione***

$\mathbf{T_1(N)}$ tempo sequenziale di esecuzione
$\mathbf{T_0(N)} = T_0(T_1, P) = P * T_P(N) - T_1(N)$

Se si lavora con efficienza massima, overhead nullo
$\mathbf{T_0(N) = 0} \Rightarrow P * T_P(N) = T_1(N)$

$$\mathbf{T_P(N)} = \frac{T_0(N) + T_1(N)}{P}$$

$$\mathbf{S_P(N)} = \frac{T_1(N)}{T_P(N)} = \frac{T_1(N)\ P}{T_0(N) + T_1(N)}$$

$$\mathbf{E_P(N)} = \frac{S}{P} = \frac{T_1(N)}{T_0(N) + T_1(N)} = \frac{1}{T_0(N)/T_1(N) + 1}$$

Si possono fare misure precise

Nel caso della somma di N numeri con P processori

Consideriamo unitario
il costo della somma e della comunicazione

$T_P(N) \approx N/P + 2 \log P$

$T_0(N) = P\, T_P(N) - T_1(N)$ nodi totali k^n

$T_0(N) = P\,(N/P + 2 \log P) - N$

$T_0(N) \approx 2\, P \log P$

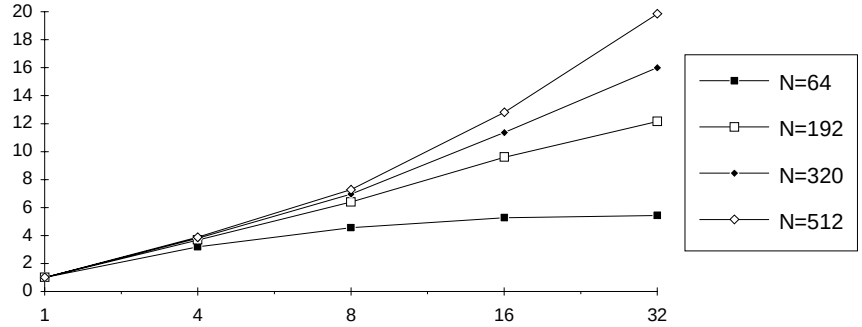
$$S_P(N) \approx \frac{N}{N/P + 2 \log P} = \frac{N\,P}{N + 2\,P \log P}$$

$$E_P(N) \approx \frac{N}{N + 2\,P \log P}$$

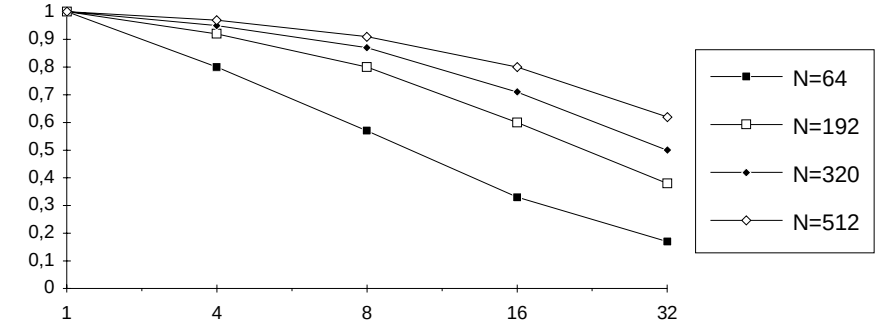
I due indicatori dipendono sia dal numero di processori
sia dalla dimensione del problema

PER IL CASO PRECEDENTE

SPEED-UP



EFFICIENZA



Gli indicatori possono anche rimanere costanti
(al variare di P)

ISOEFFICIENZA

$$E_P(N,P) = \frac{1}{T_0(N)/T_1(N) + 1}$$

$T_1(N)$ è il lavoro utile

Obiettivo ==> mantenere l'efficienza costante

$$T_0(N)/T_1(N) = \frac{1 - E}{E} \qquad T_0(N) = \frac{1 - E}{E} T_1(N)$$

$$\mathbf{T_0(N,P) = \frac{1 - E}{E} T_1(N,P) = K\ T_1(N, P)}$$

La costante K caratterizza il sistema
Nel caso precedente (1 nodo /1 valore) K non costante
Nel caso di albero, K vale 2 P log P

Funzione isoefficienza

Se teniamo costante N e variamo P, K determina se un sistema parallelo possa mantenere un'efficienza costante ==> cioè uno speed-up ideale

Se K è piccola ==> alta scalabilità
Se K è elevata ==> poca scalabilità
K può non essere costante ==>
sistemi non scalabili

Albero, K vale 2 P log P
sistema scarsamente scalabile

Valutazione di massima

Data una applicazione costituita

da Q processi
con infiniti processori a disposizione
come gestire la allocazione dei processori?

ARGOMENTATE SULLA QUANTITÀ DI RISORSE PROCESSORE DA IMPIEGARE

Come sono i processi?
Quale è la interazione
Quanto è necessario caricare ogni singola macchina
La applicazione basata su oggetti?
E le classi sono replicate?

Come è la vostra esperienza di utilizzatori di PC e workstation?
Cosa dice la legge di Grosh?
Che senso ha parlare di processi leggeri e pesanti?
Cosa ci guida per la efficienza?

se volessimo considerare l'esperienza dei sistemi concentrati di calcolo

heavily loaded limit come buona situazione
buona efficienza con processori molto carichi