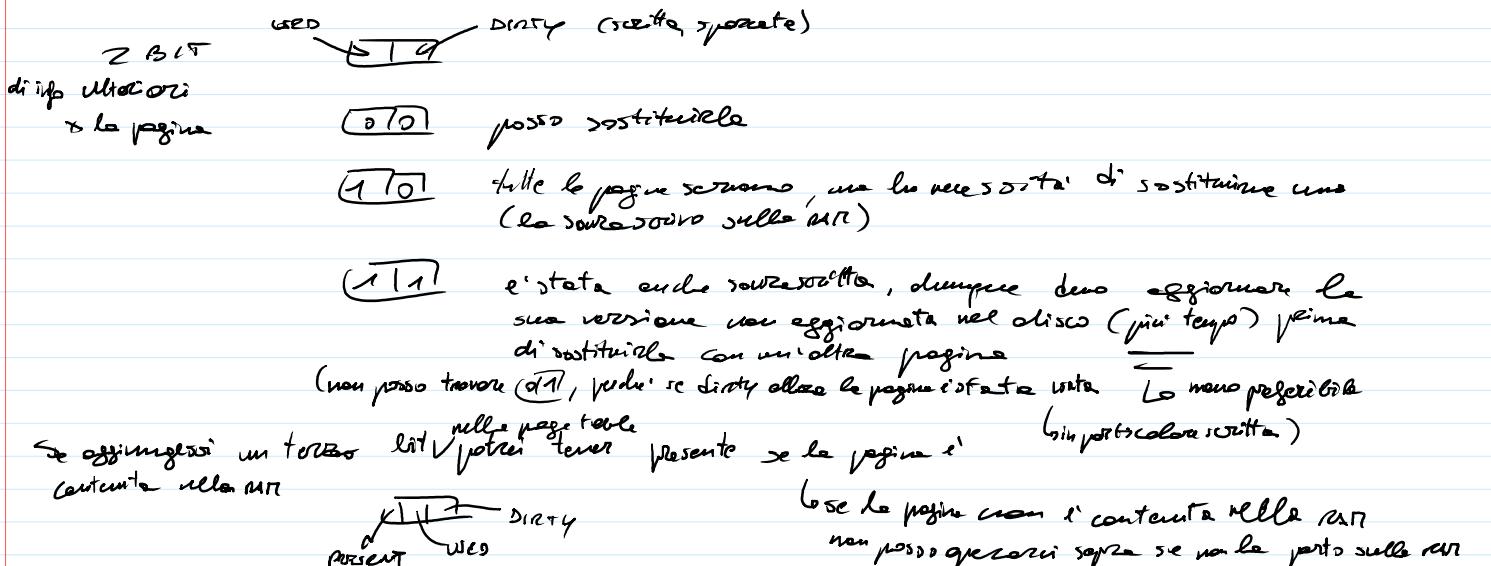


Lo acciavente e' fondamentale sapere se  
le pagine nelle RAM sono state già scritte o no,

se una pagina e' priva di aggiornamenti significa che non e' stata usata e puo  
sostituirsi con un'altra



N.B. Lo CPU non puo accedere direttamente all'Area di swap, ma puo' far indirizzamenti sulla RAM.

- implementazione delle  
l'immagazzinazione e richiesta (DEMAND PAGING), avendo portato un'elaborazione della RAM di pagina
- inizializza tutte le pagine fisiche a zero (non ha corrispondenze con le pag virtuali nell'area di swap)
- quando lo CPU invia un indirizzo la RAM vede che il bit di presenza per quella pagina e' zero (non e' presente la pagina), invia un trap (si interrompe il programma) e non c'e' ind. fisico corrispondente
- si invia il gestore delle trap che analizza l'interruzione e procede dicendo adre al sistema op. di rendere una qualunque pagina libera e associ l'ind. fisico all'ind. virtuale (bit di presenza = 1) e ritorno (esce) dalla trap.
- Lo CPU effettua la traduzione e viene indirizzato alla cella di memoria (pagina) scelta.

Tale processo puo' essere ripetuto fino a che non spazio libero, quando termina  
mi posso basare sui bit used e dirty (preferendo quelle con meno operazioni da fare)

### Accanto CPU (meno usato di recente) → riferito a quelle pagine riportate

- i programmi solitamente consumano una caratteristica particolare: LOCALITA' NEL TEMPO
- durante i processi delle CPU puo' accadere che una pagina non venga usata per un po'  
(capitolo pagina)
- Dopo aver usato quella data pagina per la prima volta (detto dalla CTR... loc. nec TERRA)  
DATA STATISTICA
- e' probabile che nel futuro tare verrà scritturata, dunque mi conviene non sostituirla  
(andare a riportarla nuovamente dentro = spese di trap). [Le pagine consumano dei bit per  
individuarne la quantità di tempo passato]
- La CPU puo' poi a un'altra pagina senza tornare su questa per un po', quindi 7 working

Le CPU passa<sup>ra'</sup> a un'altra pagina senza tornare in questa per un po', quindi in caso di memoria piena utilizza la pagina meno usata di recente, cioè la pagina "abbandonata" appena usata.

possò far partire una TRAP di interruzione del bit di uso della pagina dopo un certo tempo (per poter "andare" scambiabili)

→ non necessariamente vero  
Se il bit ~~uso~~ relativo a una pagina non usata è recente

(dipende quando ha fatto l'incremento) → dopo quanto

mi serve sapere se aggiornare la pag. nel disco  
non orario

aggiornamento  
dell'algoritmo  
LRU o sostituzione

presumibilmente  
fuori del  
working set

offrirei cose verificabili su cosa ha  
fatto effettivamente lavorando  
(e che oggi tocca in RTT x ottimizzare i  
tempi)

possò per sbaglio sostituire  
pagine sul working set  
(dove si portava dentro)

il risultato  
PAGE non  
necessariamente  
è più efficiente  
(le perdite di  
tempo e' detta  
THRASHING)

possò uscire con uno SWAP OUT

ottengo le quantità di memoria RAM associate alle pagine di un programma

portando fuori tutte le pagine della RAM, copiandole nello swap di swap,

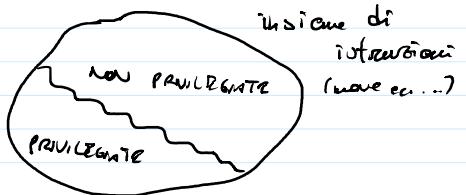
concludo: programmi (più veloci e prioritari), per poi riprenderne il programma iniziale  
(nel caso avessi troppi programmi caricati ne sospendo alcuni per poi tornarci)

Se il THRASHING continua e essenzialmente significa che il programma ha un working set notevolmente maggiore delle quantità di RAM (viste conseguenze eseguito, ma in maniera più lenta).

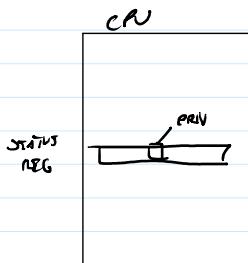
N.B.: Più il programma ha un working set elevato, più tempo ci vuole allo esecuzione di questo, perde bisogna caricare tutte le sue pagine dentro la RAM prima di poter operare su esse (operazione più dispendiosa di tempo)

L'azio è più lento e solo con l'esecuzione il programma è più veloce.  
del programma

**PRIVILEGIO:** stato di funzionamento della CPU  
definito all'interno del registro  
di stato della CPU.

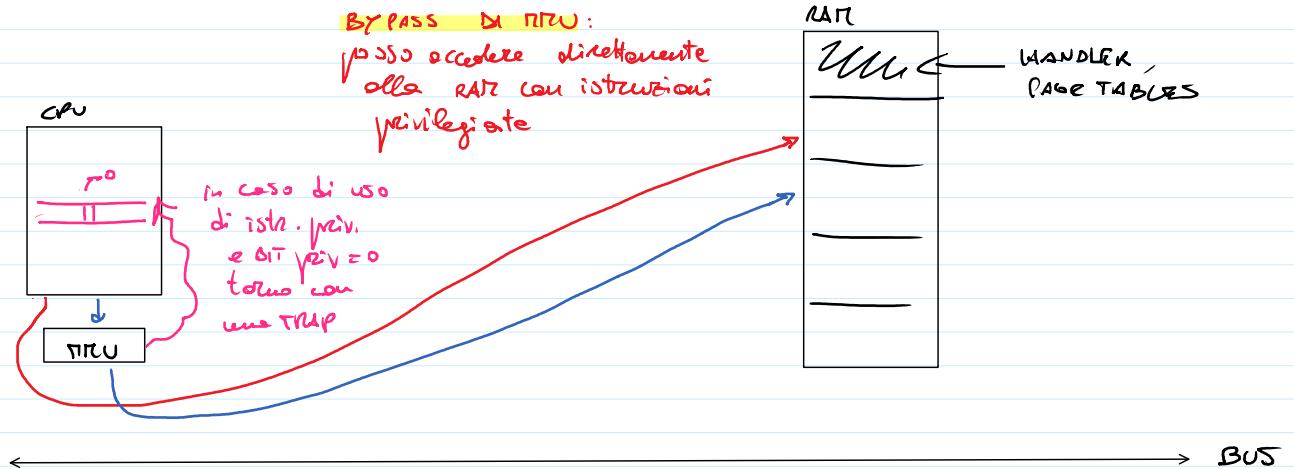
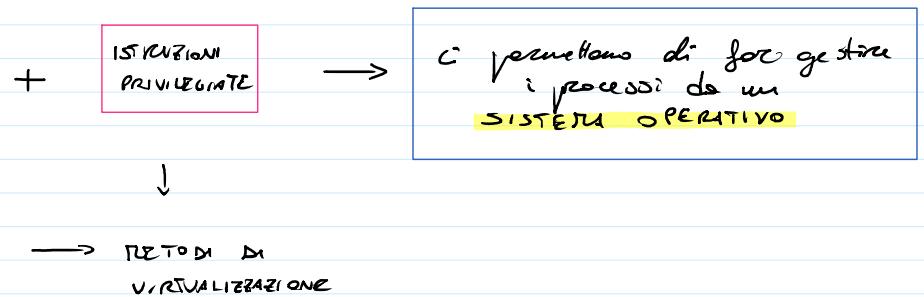


Per semplicità immaginiamo 1 bit di privilegio, { 0 NO PRIV  
1 PRIV



- Le istruzioni non priv. sono sempre eseguibili
  - Le istruzioni priv. sono eseguibili solo quando il bit di privilegio è 1
- Se si esegue un'istruzione priv. ma con bit di priv. nel SR = 0 si inizia una TRAP (situazione di errore)

- INTERRUPT/TRAP**  
→ HANDLER  
(funzioni di gestione INT/TRAP)
- RETORNA VIRTUALE**  
(a segmentazione +  
paginazione)



- FASE DI BOOTSTRAP (avvio del computer): prima che handler, page tables ecc... vengono caricati nella RAM, la MMU (che si basa su essi) non deve funzionare. Dunque opera un accesso diretto CPU - RAM che corrisponde a istruzioni di privilegio.
- Dopo che forza la CPU a passare per la MMU quando si esegue un programma. → Tale possibilità si ottiene grazie all'utilizzo del bit e di istruzioni di privilegio.

SECURITY KERNEL (nucleo di sicurezza)

## SECURITY KERNEL (nucleo di sicurezza)

- Comprende i gestori delle trap ecc.. (metodi di virtualizzazione visti sopra)
 

N.B. Si cerca di avere questo nucleo ESENTE da errori; un eventuale errore nel nucleo di sicurezza può comportare danni (anche permanenti) ai gestori handler, alle RRU o ad altri dispositivi.
- Vice versa garantisce la protezione dell'hardware del sistema, il nucleo gestisce i meccanismi di TRAP / INTERRUPT, memoria virtuale e il sistema del bit di privilegio:
 

Cambia il BIT 1 → 0 (come possono farlo tutte le istruzioni anche non privilegiate, ma vince il passaggio 0 → 1 solo tramite TRAP (e se stesso).  
/INTERRUPT
- Solitamente viene usato il modello **MICROKERNEL**, che ne gestisce in maniera minimale i processi descritti sopra, limitandosi alla gestione della virtualizzazione hardware (grandi handler, RRU, bit di privilegio).
- Una maggiore sicurezza la si ottiene con maggiore semplicità di realizzazione (= minore margine di errore).
 

N.B. Durante la fase di bootstrapping il sistema è privo di potenziali hardware (ma ha ancora caricato gestori trap, quindi non possono lanciare trap!).

## APPLICAZIONI

- In caso di errori (o di uso di applicazione sbagliata) si lancia una TRAP al security kernel

Dopo la fase di bootstrapping, il bit di privilegio viene portato a zero, rendendo impossibile l'esecuzione di ulteriori istruzioni privi. In caso di esecuzione di queste, lo CPU manda una trap e se stessa e, se necessario, riporta il bit a 1, altrimenti salta l'istruzione.  
gestione delle trap

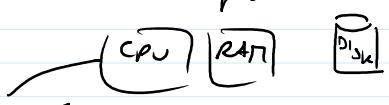
$$\text{PRIV} \begin{cases} 1 \rightarrow 0 & \text{può essere effettuata da un'istruzione più}\\ & \text{avvenuta prima.} \\ 0 \rightarrow 1 & \text{" attraverso una trap (o interrupt)} \end{cases}$$

Il nucleo di sicurezza software è il sistema operativo, che gestisce gli errori software  
[Teoria sulle istru. priv.- Vedi dimostrazioni POPEN e GOLDBERG]

## VIRTUAL MACHINE (v-m o m-v)

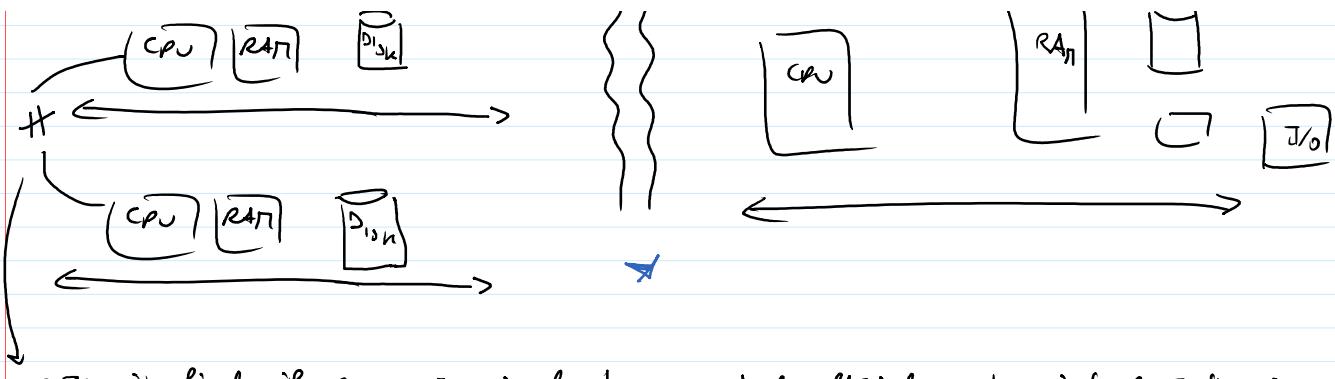
È un software che utilizza i sistemi di virtualizzazione visti sopra.

Simula uno o più sistemi completi



SIMULAZIONE =





SENZA vincoli che il processore sia lo stesso, né gli altri dispositivi virtualizzati siano gli stessi

- \* La "simulazione" è effettuata da un **IPERVISORE** (**Hypervisor**): che sostituisce lo stesso
  - = puo' sostituire un sistema operativo, non offrendo interezza, ma offrendo all'utente una più macchine virtuali utilizzabili come processori fisici
  - = oppure un ipervisore che mi permette di vedere altre m-v. Sulle m-v posso installare un sistema op. diverso da quello di altre m-v e diverso da quello dell'ipervisore (cioè del sistema host fisico)

### Come funziona?

L'idea di base è il nucleo (che puo' essere il microkernel o l'hypervisor) che utilizza i meccanismi HW per virtualizzare (CPU in modalità privilegio), caricando gli Handler ecc... all'inizio, per poi porfere il bit di priv. da 1 → 0.  
Dopo di che si gestiscono le trap/interrupt.

Si puo' programmare una interrupt con un timer (= dopo n tempo manda un'interruzione), in modo tale da:

- far intervenire l'HYPERVISOR
- sfruttare il TIME SHARING della CPU: le applicazioni vengono portate su un po' per volta (con velocità altissime, la commutazione fra applicazione è rapidissima), anche se il timer non raggiunge un certo tempo, la CPU si concentra su un programma, quando il timer raggiunge l'n tempo, scatta l'interrupt e la CPU passa a lavorare su un altro programma

↳ = CONTEXT SWITCH (si salvano dati del programma su cui si lavora per poi recuperarli in tempi successivi)

All'arrivo delle m-v il bit di privilegio risulta essere sotto controllo dell'ipervisore (o del kernel), viene levata una trap e viene gestita dal trap handler nell'ipervisore, che "simula" il passaggio del bit di priv. 0 → 1 per poter virtualizzare; il sistema operativo ospite crede di poter gestire quindi l'HW.

[ Con un sistema di trap, il sistema op. ospite vede un RICARICAMENTO della CPU, seppur il comportamento di questo rimane invariato. Questo perde le istruzioni privilegiate ]

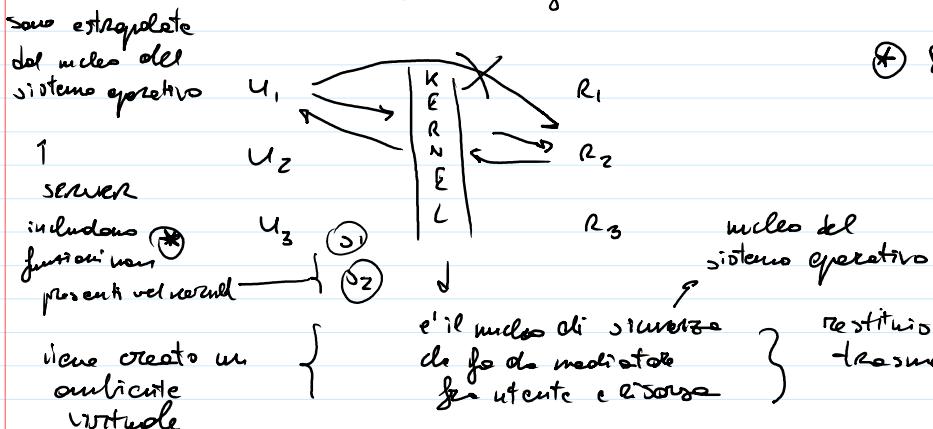
[ Con un sistema di trap, il sistema op. aspetta vedere un RICARICOZATO delle CPU, seppur il comportamento di questo rimane invariato. Questo poiché le istruzioni privilegiate (che necessitano del bit  $\text{priv} = 1$  per poter essere eseguite) subiscono una trap, l'ipervisor gestisce la trap e verifica che non è un errore e "rimuove" temporaneamente il passaggio  $0 \rightarrow 1$ . ]

↳ viene simulata una CPU rallentata, che gestisce i processi e invia trap (a se stesso)

N.B. { L'ipervisor fa da barricata (e mediatore) di sicurezza (fra ospite e host),  
1° PRINCIPIO l'utente è il sistema operativo, le risorse sono i componenti del sistema  
DI DENNING: fisico (host)  
RIVARZIONE Il tutto è trasparente, a meno delle velocità' rallentate delle istruzioni privilegiate (che passano prima per un processo di trap).

Principi di Dunning (principi di successo)

1) Separazione fra utenti e risorse (mediatione → virtualizzazione)  
" " componenti hardware  
applicazioni e dati su essi  
lanciate dagli utenti



④ gli errori in applicazioni fuori dal kernel  
non fanno danni globali  
(es. err. nel file system e' locale ai  
dati, ma nel kernel intera funzionalità)

restituire risultati, ed eventualmente  
trasmette la risposta all'utente

↳ ogni utente non puo' interfacciarsi con gli altri utenti (entendo così error), né modifichi al kernel.

## 2. ANELLO AEROLE



la debolezza delle catene è data dal singolo canto debole, non sotto dei punti più forti: bisogna focalizzarsi sul migliorare quel punto fragile.

### 3. ECONOMIA (O RI DONDEANZA) IN CONTROLLO

→ semplificare il più possibile le caratteristiche dell'oggetto (es. programma), in modo tale da poterlo controllare in tempi più brevi e trarre le dellezze.

→ aggiungo meccanismi inoperativi degli altri, es. verifica password:  
posso fare una verifica aggiuntiva, es. SRS legate alle conoscenze  
delle password, così aumentare il livello di sicurezza.

All'attuale debito ne possono aggiungere altri che rendono la  
città più solida  
e con più tollerabilità  
della debilità iniziale



4. **RM e MI PRIVILEGI**: mai oltre privilegi oltre il minimo indispensabile dell'utente, in caso d'errore, il privilegio in più non necessario si può ritirare contro l'integrità del sistema.

(es. rimuovere il privilegio di scrittura nelle forme del codice nella RAR, catturando problemi a quei segmenti)

ovvero meno istru. priv. e più van. priv.