MEMORIA VIRTUALE

↳ la memoria di un calcolatore è composta da celle contigue di **1 byte** (8 bit) a cui in genere si accede per gruppi di byte (=parole);

→ **dim. mem = n° byte**

sist. general purpose → nessuno strumento è specializzato per qualcosa

→ nei sist. gen. purp. la memoria è esterna e gestita dai bus tramite indirizzi riferiti alle varie celle (tendenz. in binario)

→ molti sistemi dispongono di quantità di memoria inferiore rispetto a quella 'teorica':

- S.O. e HW gestiscono la mem, mentre il programmatore 'finge' di averla a disposizione tutta: - **FISICA** → n° byte memoria effettivamente disponibile;
  - **VIRTUALE** → n° byte indirizzabili da una parola (gruppo di byte): N byte → 2^N indr

$$SPAZ_{VIRT} \geq SPAZ_{FIS}$$

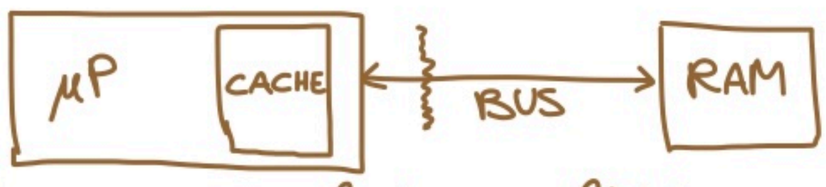
→ la mem. virtuale rappresenta quindi un'astrazione per il programmatore;

→ **indirizzi virtuali**: generati dal linker alla fine del processo di compilaz., iniziando da 0;

↳ se ho la memoria virtuale al caricamento di un programma evito la **rilocazione** degli indr in base alla posizione del codice;

genera indirizzi fisici, avviene anche in caso di mem. virtuale

→ necessaria traduzione: **indr logici → indr fisici**



logico → fisico

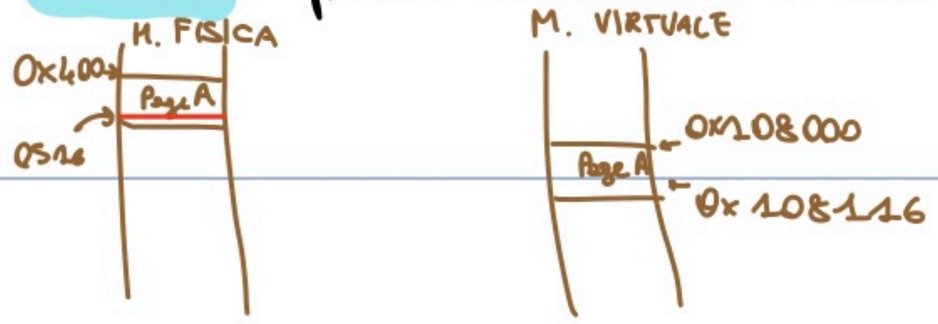
→ come passo da log. a fisico? **memory mapping**, svolto da MMU e S.O.

memory management unit

↳ **mem. mapping** e **rilocazione** consentono l'esistenza del concetto di mem. virt., permettono di allocare dati senza preoccuparsi della reale pos. in memoria e simulare di averla tutta a disposizione;

↳ questo è possibile perché nel caso di un prog. in esec., esso non risiede completamente nella mem. fisica ma parti del progr. vengono caric/scaric dinamicamente;

PAGINA → parte di memoria a dim fissa allocata a un prog.;



↳ una pagina possiede: indr virt (pag), indr dato virt, indr fisico (pag), indr dato fisico;

es. 0001 0000 1000 0001 0001 0110 (N° PAG | OFFSET)

1 0 8 | 1 1 6

0x0420 | 0x116

ovvero parto da sx, suddivido in gruppi con il 1° da 2 e gli altri da 4

per mem. virt

→ per fisica: 0000 0101 0001 0110

0 5 1 6

0x01 | 0x116

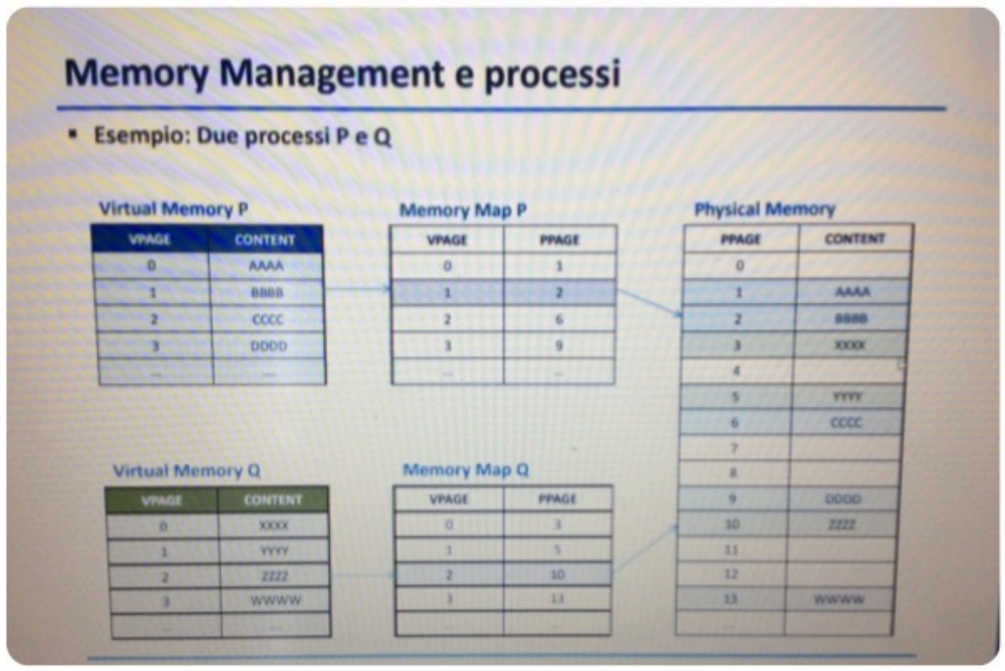
↳ cambia solo il **n° di pagina**, per cui cerco associazione tra **NPV → NPF**:

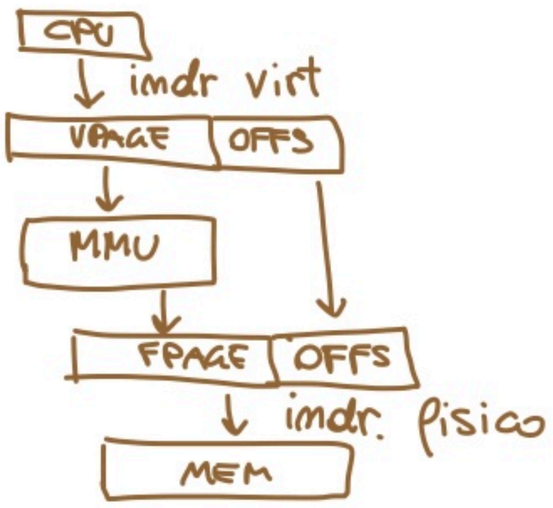
memory map:
- spezzo indr VPAGE e OFFS
- cerco VPAGE nella mappa
- prelevo corrispondente FPAGE
- aggiungo OFFS + FPAGE = indr

NPV	NPF
0x0420	0x01

0x108000 → 0x0400

non completa a carico di s.o., che altrim. richiederebbe istr assembly → ∃ parte HW: MMU (fa lavoro di mapping)
ad ogni esec. di processo si configura in base alla mem. map. del processo corrente

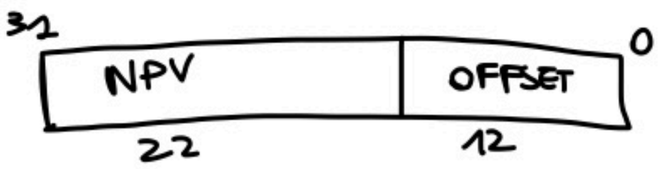




→ per evitare di contenere troppe righe di memoria la MMU carica solo una parte (la più usata) di tabelle delle pag. di ogni processo, in modo da usare una MMU per più processi;
↳ VPAGE non fa da unico indice in tabella, ma viene usato anche il PID del processo;
↳ risolve prob. frammentazione memoria;

CONVERSIONI DA RICORDARE → $K=10^3$ (=1000) in byte $K=2^{10}$ (=1024) quindi $G=10^9=(10^3)^3 \xrightarrow{\text{in Byte}} G=(2^{10})^3=2^{30}$

es. indr virt da 4GByte → 2^{32} indirizzi → 32 bit x indr
dim pag da 4KByte → 2^{12} → 12 bit
indr fisico da 4MByte → 2^{22} → 22 bit



$$NPF=\frac{2^{22}}{2^{12}}=2^{10}$$

es. Scomposizione per esadecimali (su 16 bit)

0000 1111 0001 0000 → 0x0F10
paginazione 4K → 12 bit NPV-OFFS: 0000 111100010000 → offset da 3 bit F10
↳ solo se paginazione multiplo di 4 NPV → 0

→ con questo metodo di rilocazione i vari processi possono condividere pagine; (indr fisico uguale mentre virtuale è diverso)
→ questo meccanismo rileva accessi a zone non consentite: viene generata una interrupt di violazione di memoria (da MMU o S.O.)
→ associo diversi bit di protezione che definiscono i diritti di accesso: es. lettura(R) scrittura(W) ...

PAGE FAULT → quando una pagina non risiede in mem. fisica, non ha quindi un corrispondente in tabella delle pagine (→ strutt. dati dell'S.O. che richiede sforzo all'accesso)
↳ interrupt di segnalazione
↳ processo messo in attesa di risoluzioni;
↳ per sapere se è residente o meno utilizziamo il bit di validità introdotto in tab. di mem;

tab. associativa MMU → contiene solo una parte di dati;
tab delle pag → memorizzata in strutt dati nell'S.O.
} → TABLE MISS, quando viene richiesto accesso a PV non MMU table va aggiornata ← in tab. associativa; in modo da avere dim ≅ R(n° pag. in mem);

↳ per liberare la memoria fisica una parte di pagine viene salvata su disco
swap-out → bit di validità invalidato in modo da creare spazio in mem e associato a page fault; → controllo passa a S.O.
swap-in → quando cerco una pagina invalidata viene eseguita una ricarica in mem (RAM) e validato il bit in questione;

↳ nell' indirizzo virtuale sono presenti info sulla posizione su disco;

↳ per scegliere su che pagina eseguire swap-out guardo 2 bit:

1 → access bit, inizialmente =0 ad ogni accesso è posto pari a 1;

2 → dirty bit, inizialmente =0 viene posto a 1 se si vuole scrivere in modo da indicare se va aggiornata una pag. al momento di swap-out;

come decido quale pagina sostituire? 1 → random;

2 → LRU, least recent used, pag che probabilmente non appartiene più al workingset;

3 → FIFO, elimino la più vecchia; anche questa usa contatore

2 → usa access e dirty bit: • misura invecchiamento ponendo ↷ 1 all'accesso ↷ 0 posto da S.O. periodic.

→ SEMPLICE: scelgo pag con access bit=0;

→ AVANZATO: uso un contatore per pagine con meno accessi;

CARICAMENTO PAGINE:

WORKING SET ord K → insieme di pag utilizzate negli ultimi K accessi;

per un programma valgono 2 principi: ↷ TEMPORALE (LOC.), elevata prob. di accesso a indr vicino all'ultimo effettuato;

↷ SPAZIALE (LOC.), elevata prob. di accesso ad un indr con ultimo accesso recente;

↳ R stimato da workingset

↙ minimizza page fault aumentando R

↘ Minimizza n° pag. in memoria diminuendo R

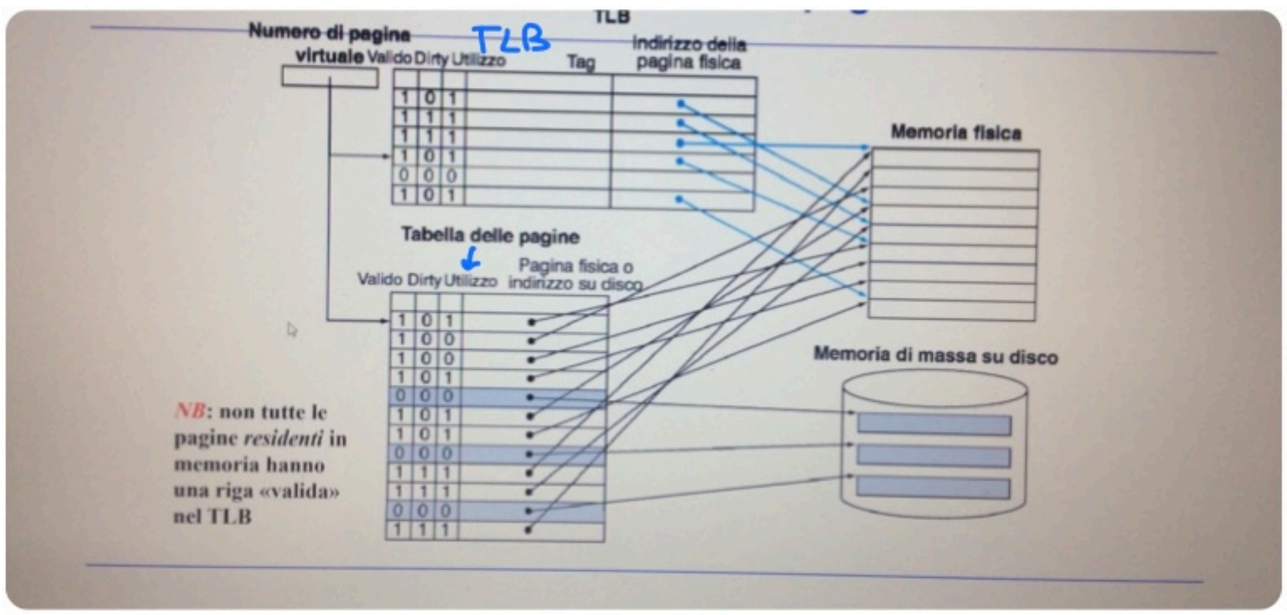
→ SOLUZ: mantengo in mem le K pagine più accedute in modo da stimare K e diminuire i page fault;

• DEMANDING PAGE → le pagine vengono caricate appena si tenta di accedervi per la prima volta (anche detto 'lazy loading'); a inizio proc. si verificano tutti page fault

→ TLB: (translation lookaside buffer), memoria (cache) tampone che MMU usa per velocizzare la traduzione degli indr virtuali.

ricerca → MISS, quando cerco nella tab e alloca nella mem (→ table miss) se non trova in TLB

NPV in TLB → HIT, trovo la pagina in 1 ciclo di clock



↳ in conclusione sarebbe necessario per ridurre la prob. di TLB miss avere pag in TLB pari a R (n° pag. residenti in mem invece che in dis