Sistemi Operativi
Gestione della Memoria (parte 2)

Docente: Claudio E. Palazzi

cpalazzi@math.unipd.it

Crediti per queste slides al Prof. Tullio Vardanega

Memoria Virtuale – 1

- Una singola partizione o anche l'intera RAM sono presto divenute insufficienti per ospitare un intero processo
- La prima soluzione fu di suddividere il processo in parti chiamate *overlay*
  - Veniva caricata in RAM una parte alla volta
  - Non appena "consumata" le veniva sovrapposta la parte successiva
  - Suddivisione a cura del programmatore!

Memoria Virtuale – 2

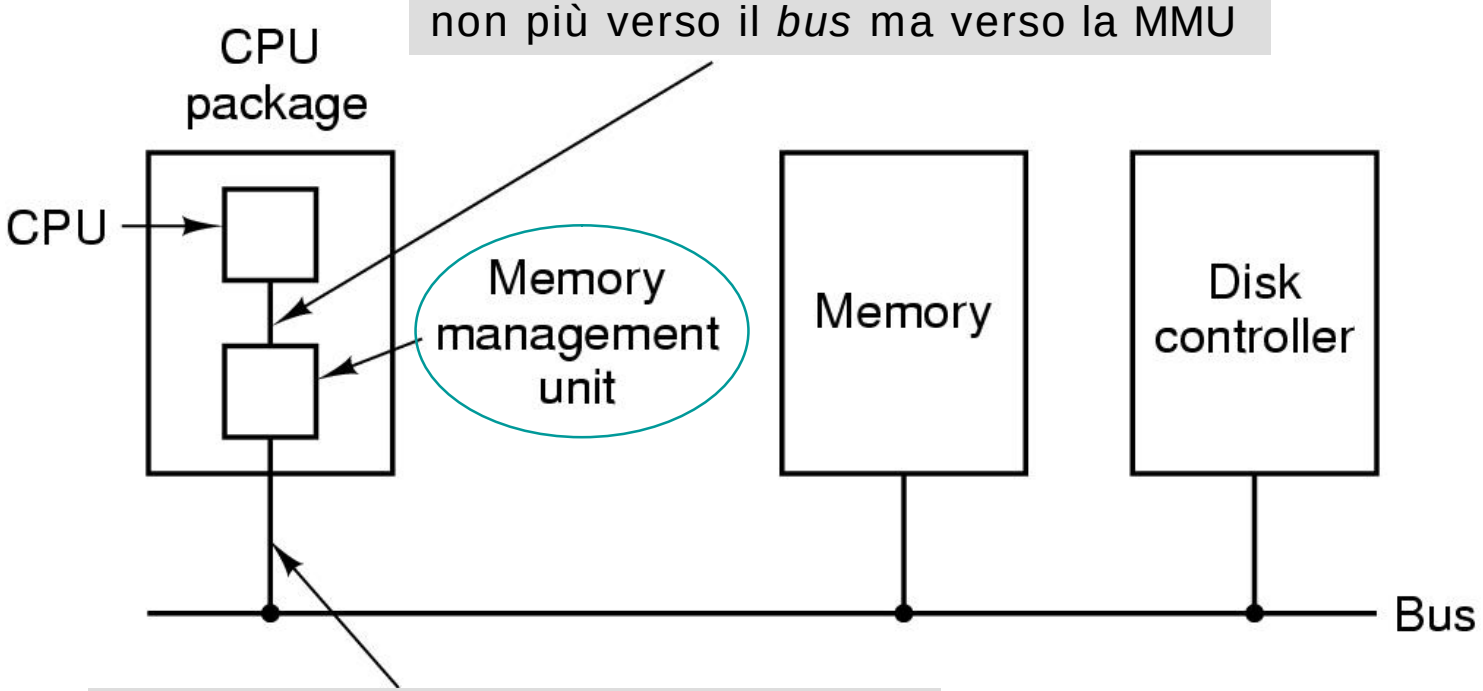
- L'idea di **memoria virtuale** nasce nel '61
- Il principio cardine è che un singolo processo può liberamente avere ampiezza **maggiore** della RAM disponibile
  - Basta caricarne in RAM solo la parte strettamente necessaria lasciando il resto su disco
  - **Senza** intervento del programmatore
- Ogni processo ha un suo proprio spazio di memoria virtuale
- Due tecniche alternative di gestione
  - **Paginazione**
  - **Segmentazione**

Memoria Virtuale – 3

- Gli indirizzi generati dal processo **non** denotano più direttamente una locazione in RAM
  - Ma vengono interpretati da un'unità detta **MMU** che li mappa verso indirizzi fisici reali
    - **Prima** di essere emessi sul *bus*
  - Il tipo di interpretazione a carico della MMU dipende dalla tecnica usata per la gestione della memoria virtuale

Memoria Virtuale – 4

La CPU emette indirizzi virtuali (logici) non più verso il *bus* ma verso la MMU

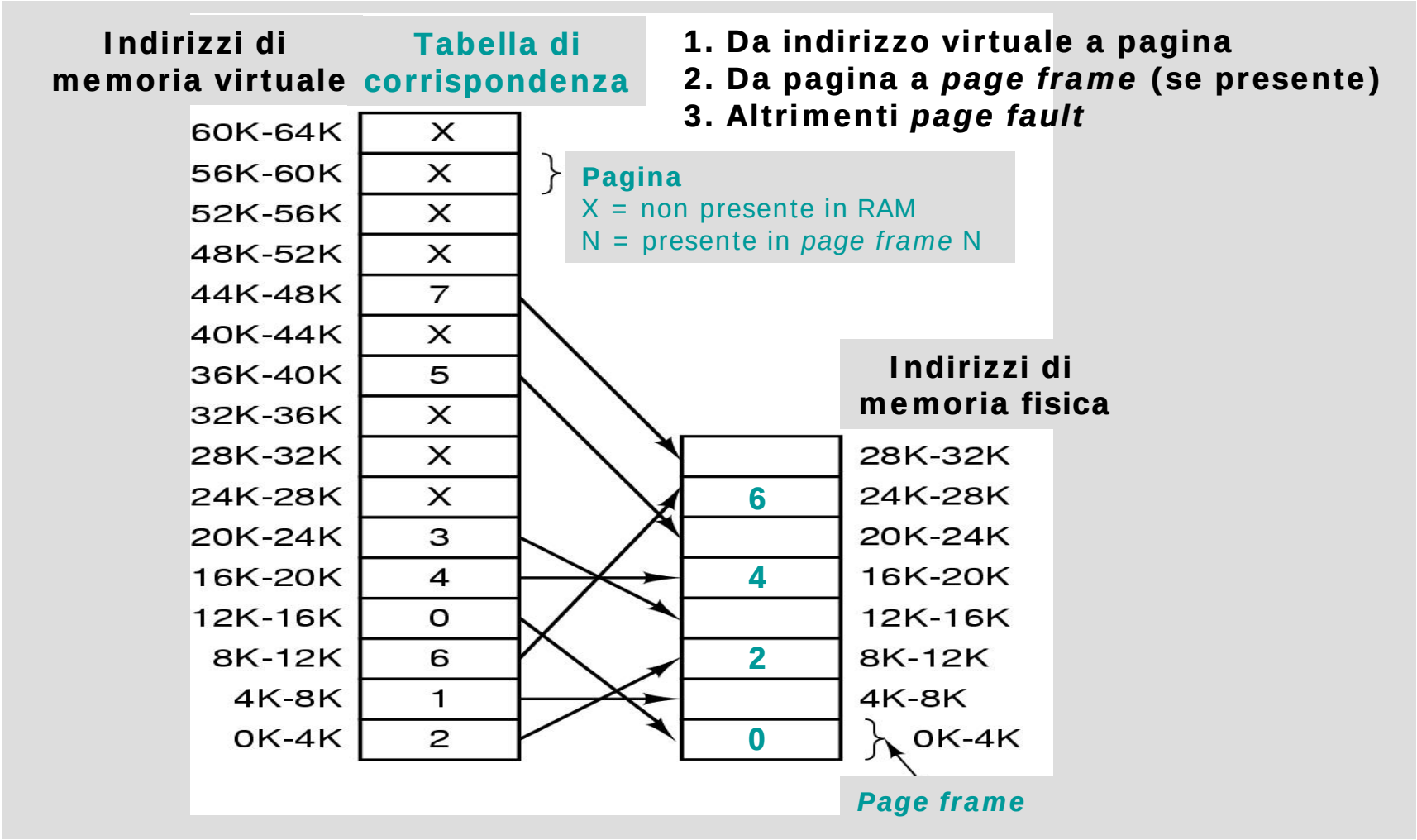


La MMU trasforma gli indirizzi virtuali in indirizzi fisici e li emette sul *bus*

Paginazione: premesse – 1

- La memoria virtuale è suddivisa in unità a dimensione fissa dette **pagine**
- La RAM è suddivisa in unità “cornici” ampie come le pagine (*page frame*)
- I trasferimento da e verso disco avvengono sempre in pagine
- Di ogni pagina occorre sapere se sia presente in RAM oppure no
  - *Bit* di presenza
  - Se una pagina è assente quando riferita si genera un evento *page fault* gestito dal S/O tramite **`trap`**

Paginazione: premesse – 2



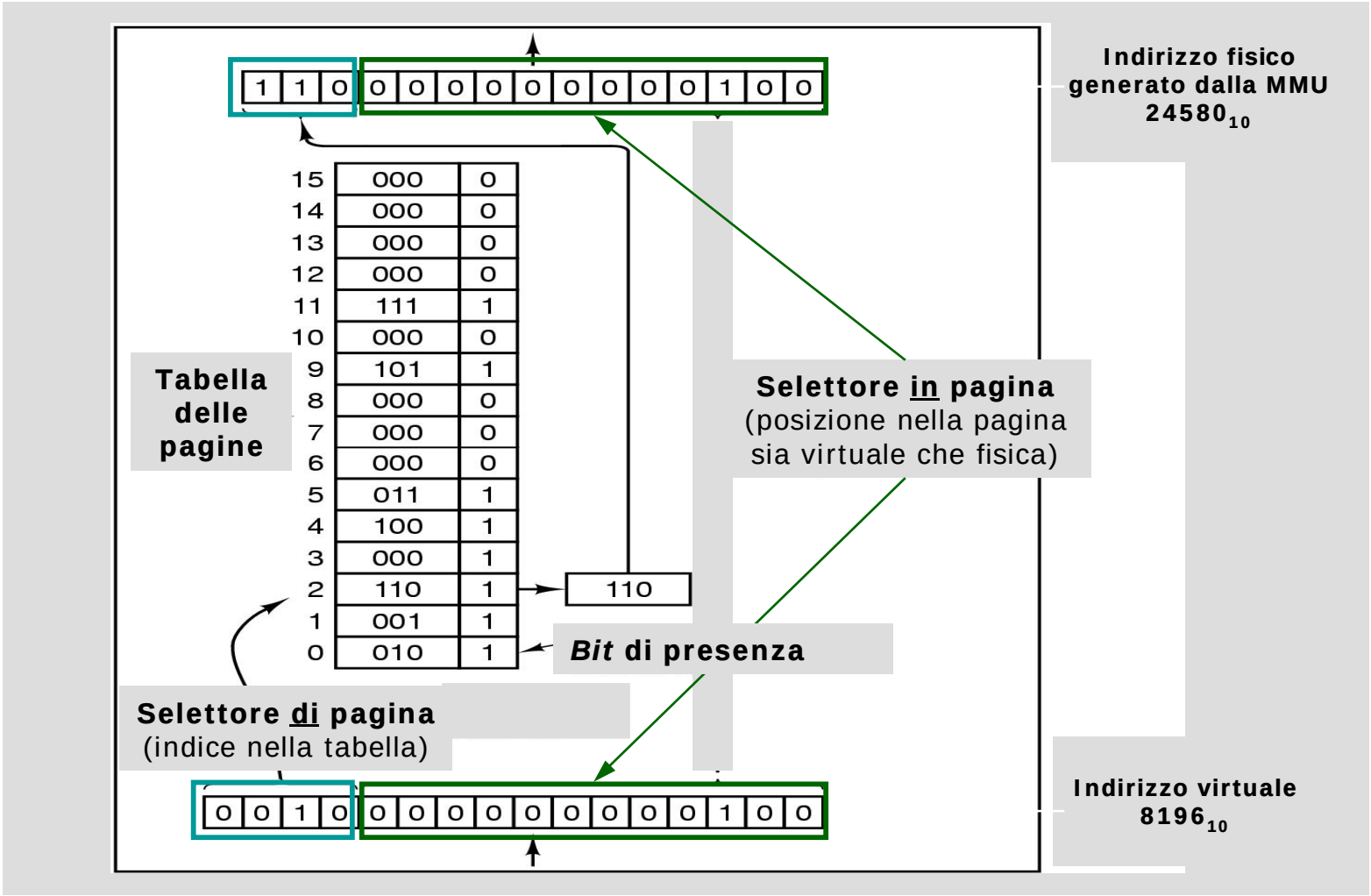
Paginazione: strutture – 1

- La traduzione da indirizzo virtuale a fisico avviene tramite una **tabella delle pagine**
  - Indicizzata per numero di pagina
    - Indirizzo $_{fisico} = \varphi$ (indirizzo $_{virtuale}$)
- La tabella può essere molto **grande**
  - Indirizzi virtuali da 32 *bit* e pagine da 4 KB → memoria virtuale da 4 GB = 1 M pagine!
- Ciascun processo ha la sua (grande) tabella delle pagine
  - Poiché ha il suo spazio di indirizzamento virtuale

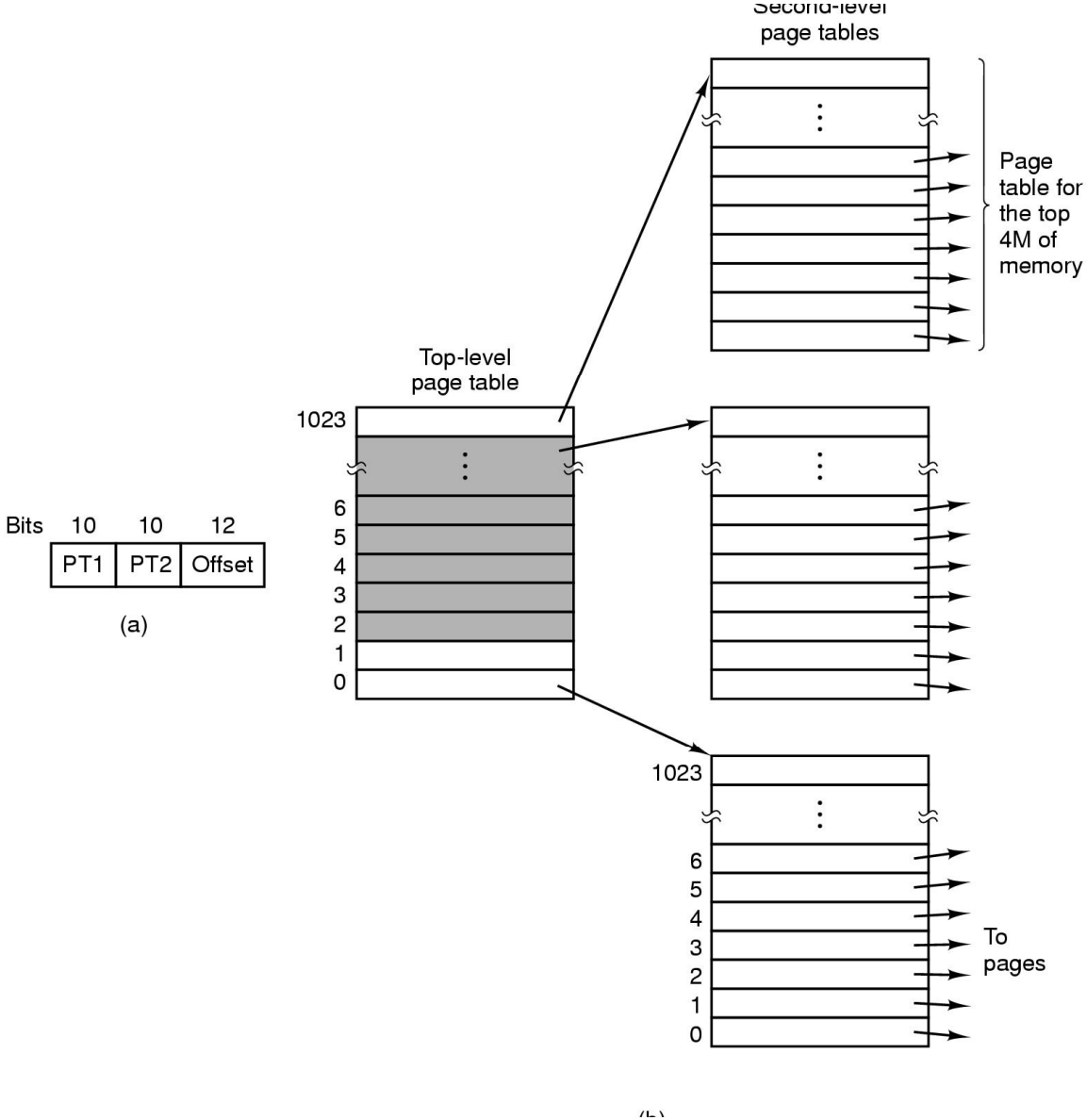
Paginazione: strutture – 1 bis

- La traduzione deve essere molto **veloce**
  - Ogni istruzione potrebbe fare riferimento più volte alla tabella delle pagine
    - Dunque se un'istruzione impiega ad es. 4 ns, allora il riferimento alla page table deve avvenire in circa 1 ns, viceversa sarà un bottleneck del sistema
  - Ogni indirizzo emesso dal processo (istruzione o operando) deve essere tradotto
    - Semplicemente (e concettualmente) potrebbe utilizzare un vettore di registri (uno per ogni pagina virtuale) caricato a ogni cambio di contesto (vedi figura nella slide seguente)
      - Lineare e non si rischia di dover accedere a memoria per scoprire il riferimento, ma costoso cambiare tutti i registri ad ogni context switch
    - Oppure come una struttura sempre residente in RAM
      - Un singolo registro punta all'inizio della page table
      - Difficile che sia usato come soluzione in modo puro

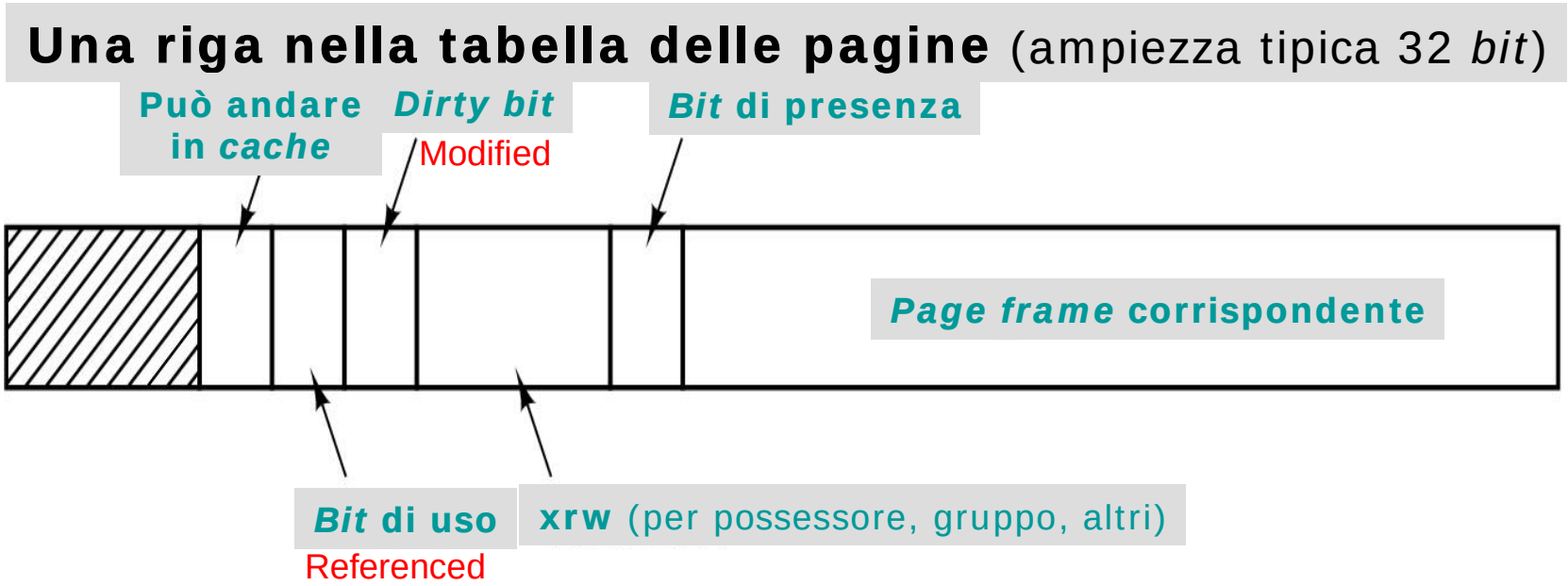
Paginazione: strutture – 2



Paginazione: strutture – 2 bis



Paginazione: strutture – 3



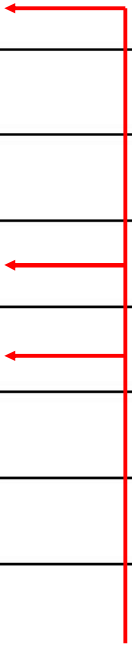
- L'indirizzo di disco ove la pagina si trova quando non è in RAM **non** è nella tabella!
  - La tabella delle pagine serve alla MMU (*hardware*)
  - Il caricamento della pagina da disco viene effettuato dal S/O (*software*) all'occorrenza di un page fault
  - L'informazione dell'uno **non serve** all'altro

Paginazione: strutture – 4

- La tabella delle pagine è così grande che non può risiedere su registri
  - Dunque deve stare in RAM
  - Riferirla per ogni indirizzo emesso (istruzioni e operandi) ha un impatto devastante sulle prestazioni
- Serve una struttura supplementare (HW) più agile che ne sia come una *cache*
  - **Piccola** memoria associativa che consente scansione parallela (*translation lookaside buffer*, TLB)
    - Solitamente interna alla MMU
    - Ricerca su tutte le righe simultaneamente
  - Basata sull'osservazione che un processo in genere usa più frequentemente **poche** pagine

Paginazione: strutture – 4 bis

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75



Es. ciclo

Paginazione: strutture – 5

- Ogni indirizzo emesso verso la MMU viene prima trattato con la TLB
  - Se la sua pagina è presente e l'accesso richiesto è permesso la traduzione avviene tramite TLB
    - **Senza** accedere alla tabelle delle pagine
  - Se non presente si ha l'equivalente di una *cache miss* e le informazioni richieste vengono caricate in TLB dalla tabella delle pagine
    - Rimpiazzando una cella in TLB e riflettendone il valore nella tabella delle pagine
      - Ma solo se cambiato!

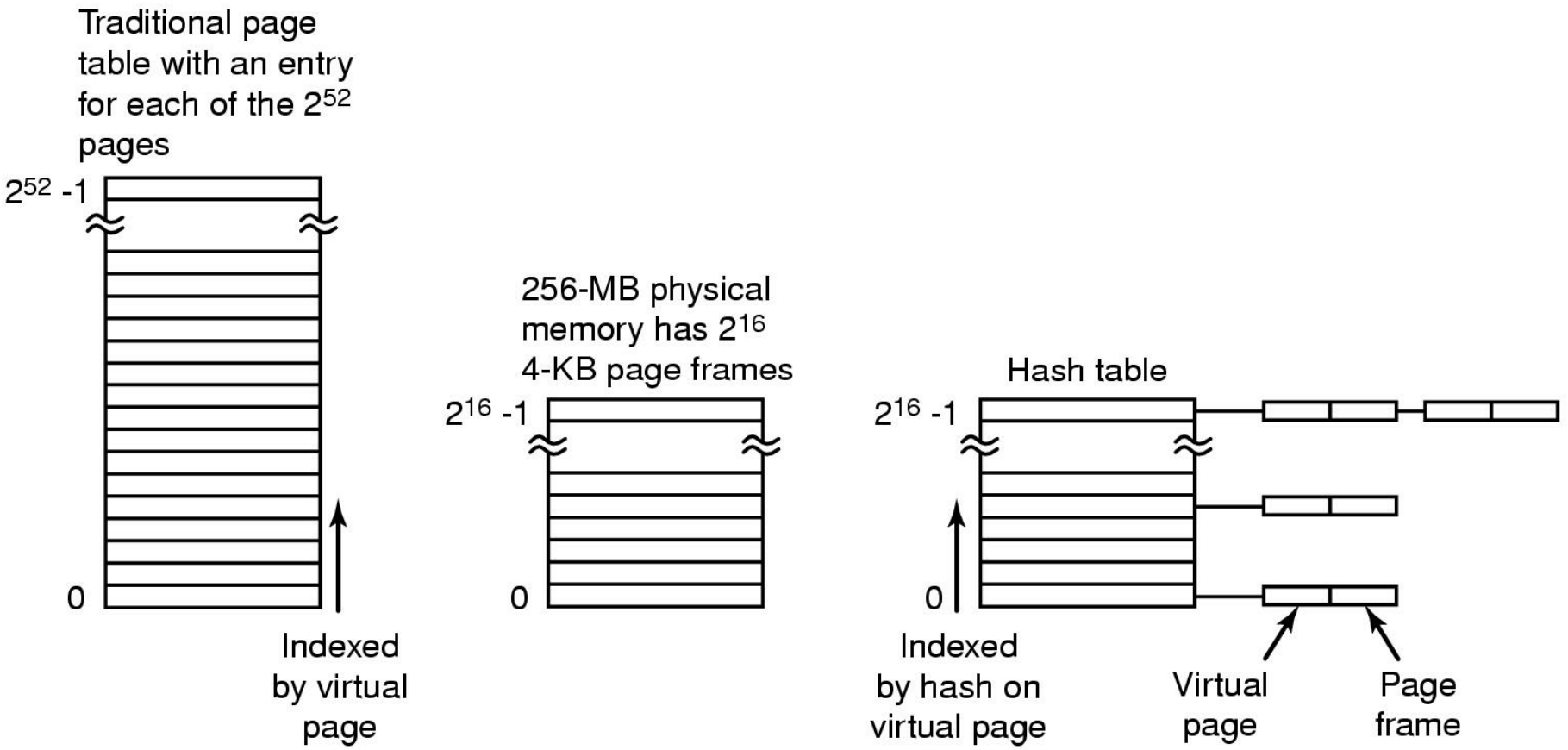
Paginazione: strutture – 6

- Oggi le TLB sono prevalentemente realizzate in *software* invece che in *hardware* nelle MMU
  - Le prestazioni sono accettabili
  - La MMU ne guadagna in semplicità e riduzione di spazio che viene dedicato ad altri usi ritenuti più vantaggiosi (*cache*)
- Con le architetture a 64 *bit* però le tabelle delle pagine assumono dimensioni **proibitive**
  - 64 *bit* → memoria virtuale da 16 EB
    - (1 E = 1 G × 1 G)
  - Pagine da 4 KB → 4 P pagine
    - (1 P = 1 M × 1 G)
  - 32 *bit* per pagina in tabella → ampiezza 16 PB!
- Serve un'altra soluzione

Paginazione: strutture – 7

- La soluzione adottata impiega una **tabella invertita**
  - Non più una riga per pagina ma per *page frame* in RAM
    - Considerevole risparmio di spazio
  - La traduzione da virtuale a fisico diventa però molto più complessa
    - Poiché la pagina potrebbe risiedere in **qualunque** *page frame* bisognerebbe scandire l'intera tabella per trovarla
      - Per ogni indirizzo emesso dal processo!
      - Grande dispendio di tempo
  - Ricerca velocizzata dall'uso di TLB
  - E anche realizzando la tabella invertita come una tabella *hash* indicizzata da f_{Hash} (indirizzo $_{virtuale}$)
    - I dati relativi alle pagine i cui indirizzi virtuali indicizzino una stessa riga di tabella vengono collegati in lista

Paginazione: strutture – 7 bis



Paginazione: rimpiazzo – 1

- Quando si produce un *page fault* il S/O deve **rimpiazzare** una pagina
  - Salvando su disco la pagina rimossa
    - Ma solo se modificata nell'uso
- Inopportuno rimpiazzare pagine in uso frequente
  - Altrimenti si paga prezzo doppio dovendole riportare troppo presto in RAM
- Problema del tutto analogo a quello della *cache*
  - Anche di quelle emulate a *software* per la gestione di informazioni logiche

Paginazione: rimpiazzo – 2

- Rimpiazzo ottimale (optimal replacement)
  - Rimpiazza la pagina in memoria che non sarà usata per maggior tempo
- La scelta perfetta **non** è realizzabile
  - Perché il S/O non ha modo di sapere quali pagine il processo accederà in futuro
    - Un po' come scegliere il processo più breve
- Le scelte realizzabili sono sempre e solo approssimazioni sotto-ottimali
  - Sulla base di osservazioni empiriche sull'uso recente delle pagine attualmente in RAM

Paginazione: rimpiazzo – 3

- **NRU** (*Not Recently Used*)
  - Per ogni *page frame* vengono aggiornati
    - *Bit* M (*modified*), inizializzato a 0 dal S/0
    - *Bit* R (*referenced*), posto a 0 **periodicamente** dal S/O per stimare la frequenza d'uso
  - Le pagine nei *page frame* sono classificate in
    - **Classe 0**: non riferita, non modificata
    - **Classe 1**: non riferita, modificata
    - **Classe 2**: riferita, non modificata
    - **Classe 3**: riferita, modificata
  - NRU sceglie una pagina **a caso** nella classe non vuota a indice più basso

Paginazione: rimpiazzo – 4

- **FIFO**
  - Rimuove la pagina di ingresso più antico in RAM
    - Basta una lista ordinata di *page frame*
      - Ogni inserimento viene marcato in coda e la rimozione avviene dalla testa
- ***Second chance***
  - Corregge FIFO rimpiazzando solo le pagine con *bit* R = 0
    - Altrimenti il *page frame* viene considerato come appena caricato, posto in fondo alla coda e R viene posto a 0
    - Degenera in FIFO quando tutti i *page frame* siano stati recentemente riferiti

Paginazione: rimpiazzo – 5

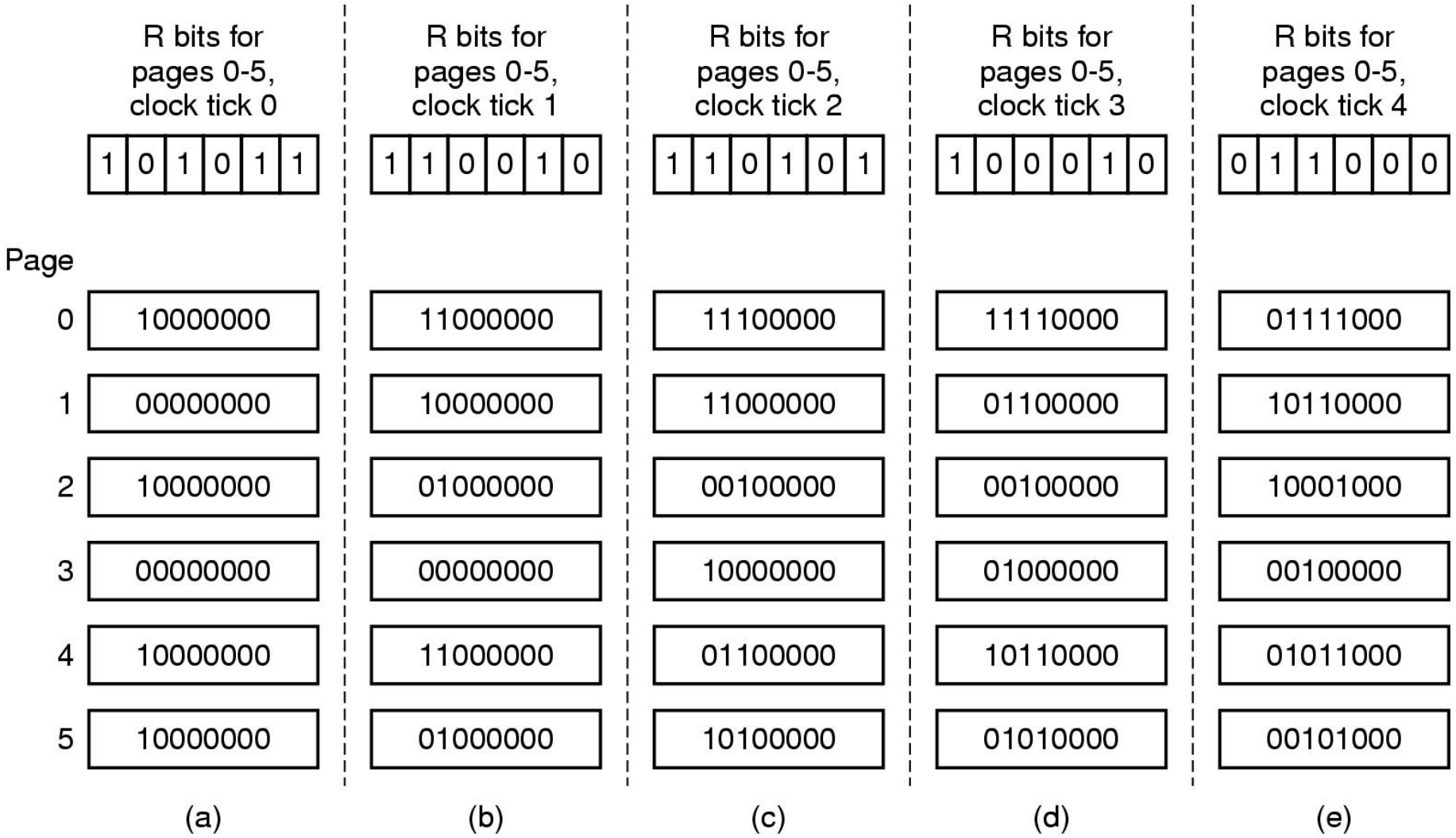
- **Orologio**
  - Come SC ma i *page frame* sono mantenuti in una lista **circolare**
    - L'indice di ricerca si muove come una lancetta
- **LRU** (*Least Recently Used*)
  - Approssima l'algoritmo ottimale
  - Richiede lista aggiornata ad ogni riferimento a memoria
  - Necessita di *hardware* dedicato
- **NFU** (*Not Frequently Used*)
  - Realizzabile a *software*
  - Per ogni *page frame* aggiorna periodicamente un "contatore" C che cresce di più se R = 1
  - PROBLEMA: non dimentica nulla!

Paginazione: rimpiazzo – 5 bis

- **Aging** (*not frequently used* modificato)
  - Realizzabile a *software*
  - Per ogni *page frame* aggiorna periodicamente un “contatore” C che cresce di più se R = 1
    - Non incrementa C con R ma gli inserisce R a sinistra
  - Approssima LRU con differenze importanti
    - Valuta solo periodicamente (a grana grossa)
    - Usando N *bit* per C perde memoria dopo N aggiornamenti

Paginazione: rimpiazzo – 5 ter

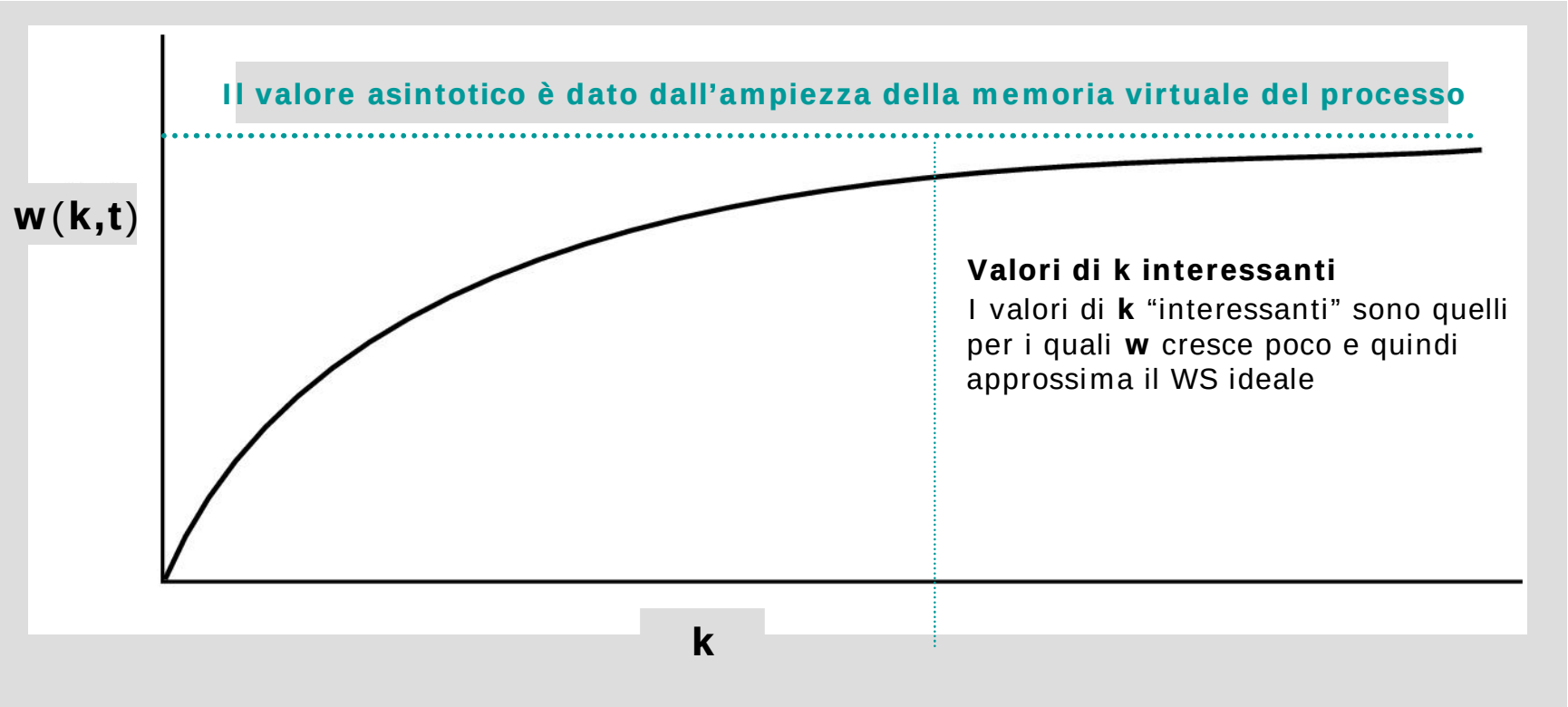
- **Aging** (*not frequently used* modificato)



Paginazione: *working set* – 1

- Studi accurati mostrano come i processi emettano la maggior parte dei loro riferimento entro un ristretto spazio locale
  - **Località dei riferimenti**
- ***Working set*** (WS) è l'insieme di pagine che un processo ha in uso a un dato istante
  - Se la memoria non basta ad accogliere il WS si crea il fenomeno di **thrashing**
  - Se il WS viene caricato prima dell'esecuzione si ha **prepaging** (evitando page fault)
  - **w**(**k, t**) è l'insieme di pagine che soddisfano i **k** riferimenti emessi al tempo **t**
    - Funzione monotonica crescente

Paginazione: *working set* – 2



Paginazione: *working set* – 3

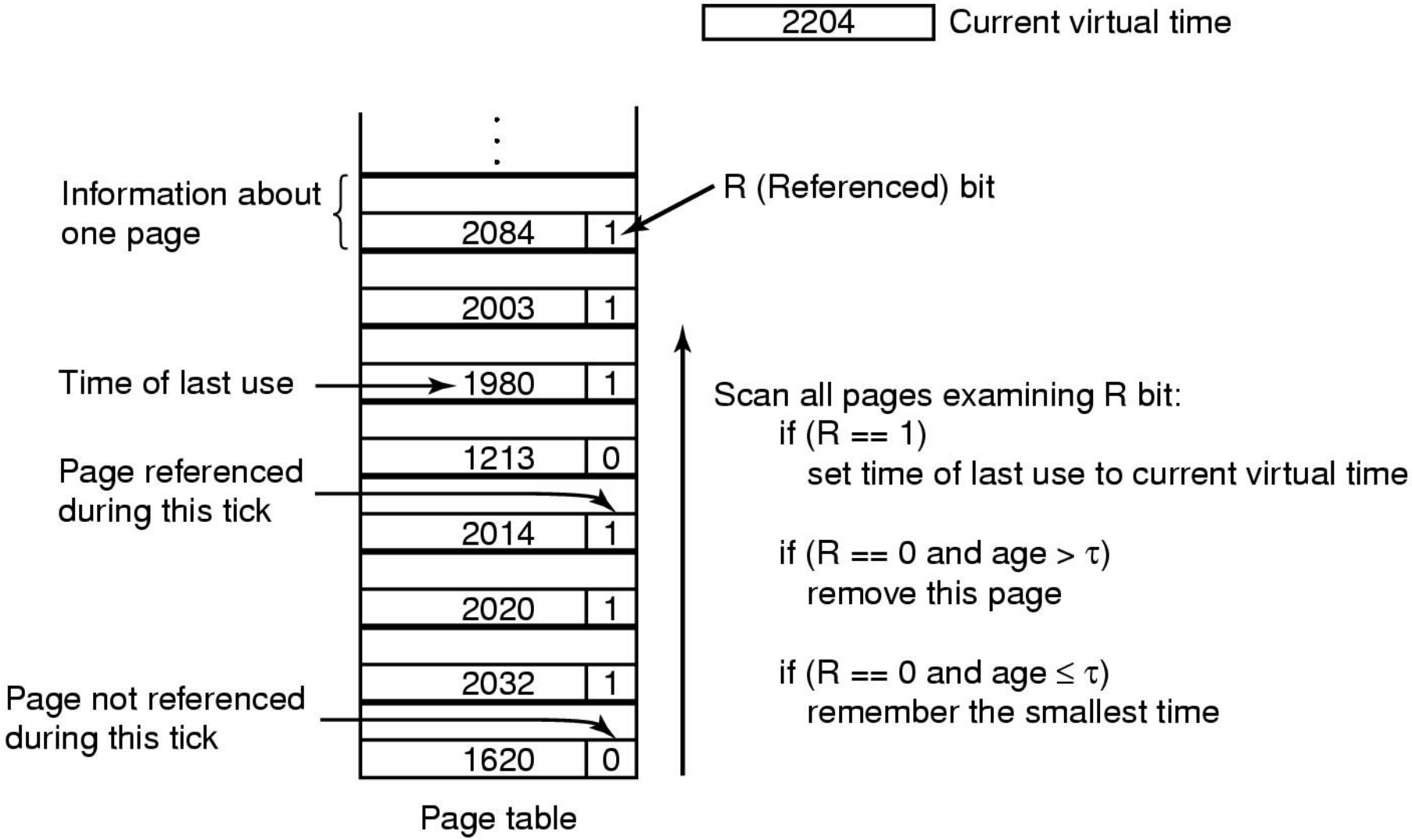
- Se si conoscesse il WS dei processi le pagine da rimpiazzare sarebbero quelle che **non** vi fossero comprese
- Conoscere precisamente il WS dei processi a tempo d'esecuzione è però **troppo costoso**
  - Quanto deve valere **k**?
  - Più facile fissare **t** come $(t, t + \Delta t)$
    - Considerando **t** come valore dell'effettivo tempo di esecuzione di quel processo (**tempo virtuale corrente**)
      - Non del tempo trascorso!
    - WS è fatto dalle pagine riferite dal processo nell'ultimo Δt

Paginazione: rimpiazzo - 6

- **WS approssimato**
  - Simile all'*Aging*
    - Ogni *page frame* in RAM ha un attributo temporale che indica se a un dato istante appare come riferita (R = 1)
      - Tale attributo prende il valore **t** del **tempo virtuale corrente** all'arrivo di un *page fault*
      - R e M sono posti a 1 dall'*hardware*
      - R è posto a 0 (se non in uso) da un controllo periodico e al *page fault*
    - Al *page fault* sono rimpiazzabili le pagine con R = 0 e valore di attributo **antecedente** all'intervallo (**t** − Δt, **t**)
      - Se non ci sono, si prende la più vecchia con R = 0
      - Se all'istante **t** tutti i *page frame* avessero R = 1 verrebbe rimpiazzata una pagina scelta a caso, con M = 0
    - Nel caso peggiore bisogna scandire l'**intera** RAM!

Paginazione: rimpiazzo – 6 bis

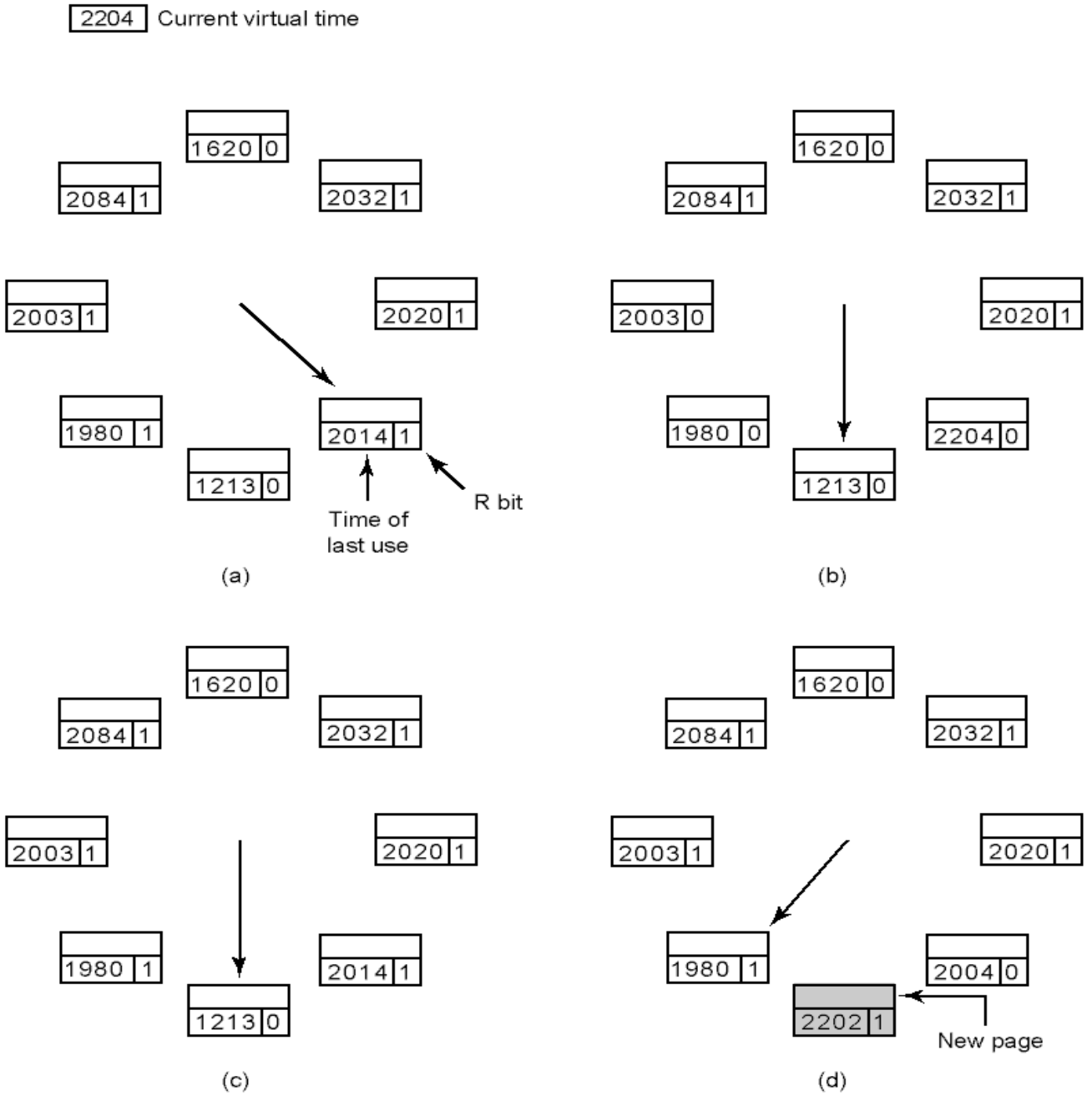
- **WS approssimato**



Paginazione: rimpiazzo – 7

- **WS approssimato con orologio**
  - *Page frame* organizzati in lista circolare
    - Come per l'orologio semplice
    - Ma con le informazioni del WS approssimato
  - Una "lancetta" indica il *page frame* corrente
    - Al *page fault* se R = 1 la lancetta avanza e R = 0
    - Se R = 0 si valuta l'attributo temporale
      - Se fuori da **w**(**k,t**) e con M = 0 allora rimpiazzo
      - Altrimenti il *page frame* va in una coda di trasferimento su disco e la lancetta avanza
        - » Alla ricerca di un *page frame* rimpiazzabile direttamente
        - » Quando **N** pagine in coda si trasferisce su disco
    - Se nessun *page frame* è rimpiazzabile allora si sceglie una pagina con M = 0 altrimenti quella cui punta la lancetta

- **WS approssimato con orologio**
  - Molto usato in pratica



Paginazione: rimpiazzo - 8

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm