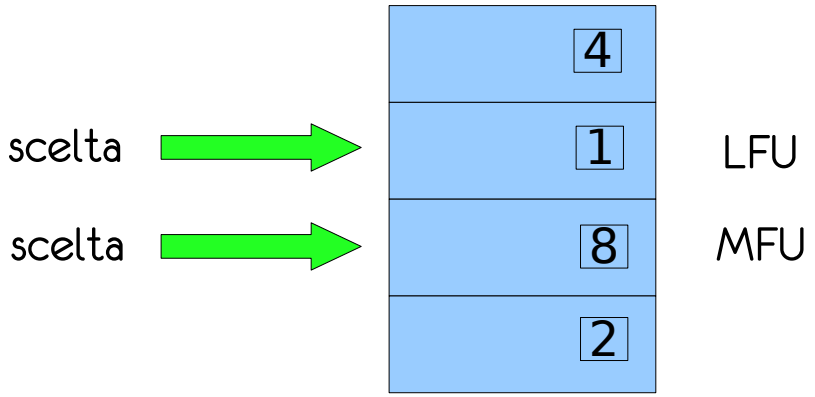
Sostituzione su conteggio

- Fra i tanti algoritmi per la sostituzione delle pagine che sono stati ideati, particolarmente rilevanti quelli basati sul conteggio del numero di riferimenti fatti a ciascuna pagina
- Appartengono a questa categoria:
  - l'algoritmo LFU (least frequently used): sostituisce la pagina con il minor numero di riferimenti
  - l'algoritmo MFU (most frequently used): sostituisce la pagina con il maggior numero di riferimenti

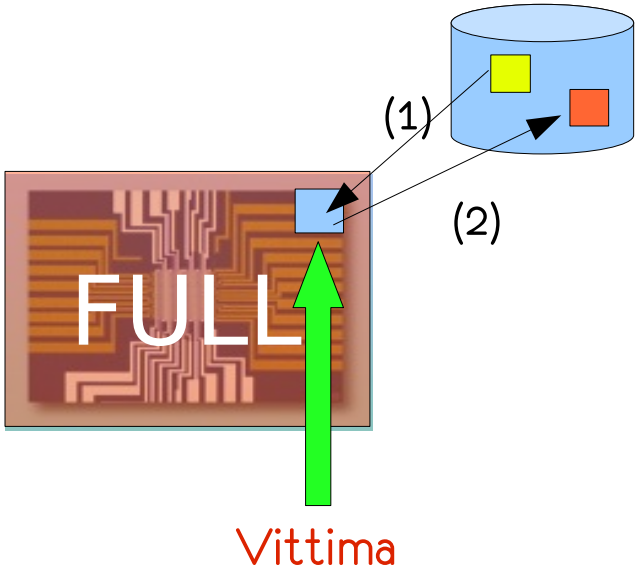


Commenti

- LFU si basa sull'idea che una pagina molto usata ha un conteggio alto, mentre una pagina che serve poco avrà un conteggio basso
- problema: le pagine sono solitamente usate per un certo periodo di tempo. Se non si ha un modo per ridurre nel tempo il conteggio degli accessi, non si riesce a distinguere fra una pagina che è stata molto usata ma ora non lo è più e una pagina che è attualmente molto usata
- MFU: si basa sul principio opposto che una pagina con un contatore basso è stata probabilmente appena caricata, quindi è utile

Pool of free frames

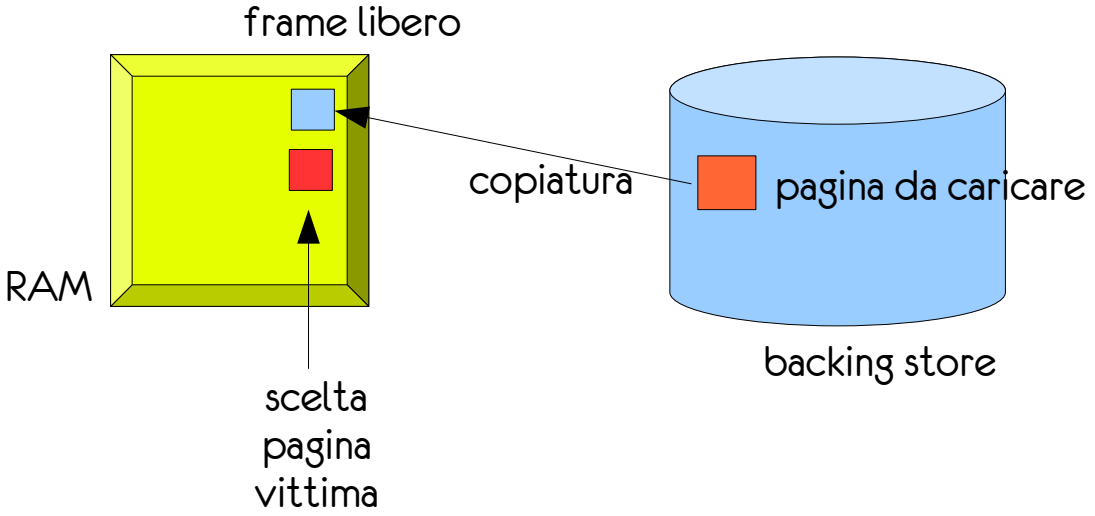
- Spesso gli algoritmi di sostituzione delle pagine sono affiancati da altre procedure finalizzate a incrementare le prestazioni del sistema
- Fra queste una tecnica che consente di iniziare la copiatura in RAM della pagina necessaria per proseguire (1) **prima** che la pagina vittima sia stata ricopiata in memoria secondaria (2)



Pool of free frames

- L'idea è associare a ogni processo un piccolo pool di frame liberi
- quando diventa necessario caricare una pagina nuova:
  - la si copia in un frame libero associato al processo
  - durante la copiatura, si sceglie una vittima e la si copia in backing store
  - in questo modo si libera un frame che viene aggiunto al pool di frame liberi del processo
- **NB**: la vittima non viene cancellata!
- fino a quando non viene sovrascritta, si tiene memoria della sua presenza in RAM, in questo modo non è necessario ricopiarla in caso di un successivo tentativo di accesso

Pool of free frames



La pagina vittima va ad arricchire il pool dei frame liberi assegnati al processo ma non viene cancellata o sovrascritta, al contrario rimane accessibile attraverso la tabella delle pagine

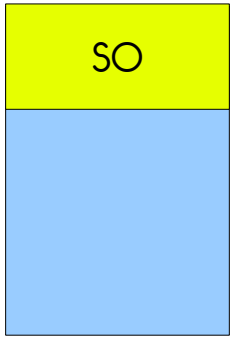
Allocazione dei frame

- per i processi utente
- per i processi kernel

Allocazione dei frame

- L'algoritmo di allocazione dei frame completa l'implementazione della memoria virtuale, iniziata con la definizione di un algoritmo di sostituzione delle pagine
- Questo algoritmo viene applicato quando si hanno a disposizione N frame liberi, occorre caricare uno o più processi e bisogna decidere quanti frame assegnare a ciascuno (come spartirli)
- Partiamo da uno scenario semplice, in un contesto di paginazione su richiesta pura:
  - 1 utente
  - RAM da 128K
  - pagine da 1K
  - un processo

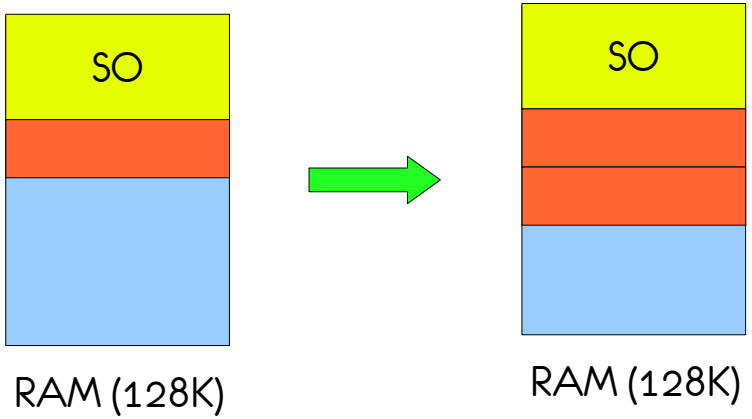
Allocazione dei frame



Una porzione deve comunque essere riservata al sistema Operativo

La porzione rimanente può essere allocata per il processo del singolo utente

Supponiamo che il SO occupi 28K e che si attui una gestione di paginazione a richiesta pura: il processo utente viene avviato senza caricare alcuna pagina, poi al primo page fault si effettua il primo caricamento



Quando ce ne sarà bisogno si caricherà la seconda pagina, ecc. ecc. ecc. fino a un massimo di 100K

La strategia di allocazione dei frame adottata consiste nel **non** assegnare inizialmente alcun frame libero ai processi

Allocazione dei frame

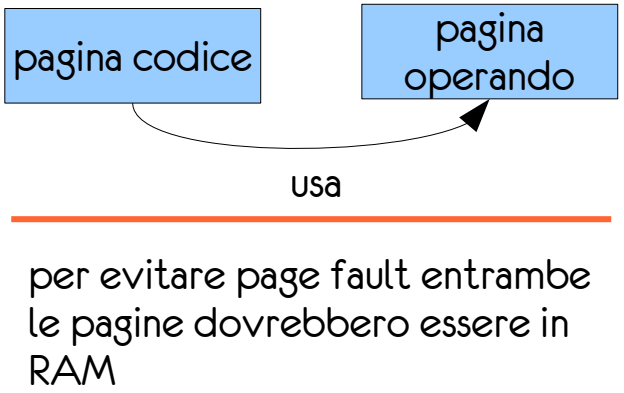
- Allocare inizialmente 0 frame per processo e poi 1 per volta quando necessario è l'unica scelta possibile? È la migliore?

Allocazione dei frame

- Allocare inizialmente 0 frame per processo e poi 1 per volta quando necessario è l'unica scelta possibile? È la migliore?
- Considerazione
  - La scelta operata è guidata dai page fault
  - Ogni page fault introduce grossi ritardi nell'esecuzione
- Scopo
  - page fault = ritardo, ritardo = inefficienza → noi desideriamo minimizzare i ritardi quindi cerchiamo di minimizzare la frequenza dei page fault
- Approccio
  - in generale il *#page fault* è inversamente proporzionale al *# di frame allocati* per ciascun processo
  - idea: cercare di mantenere in memoria un *numero minimo di frame* per processo tale da ridurre la probabilità che si generi un page fault

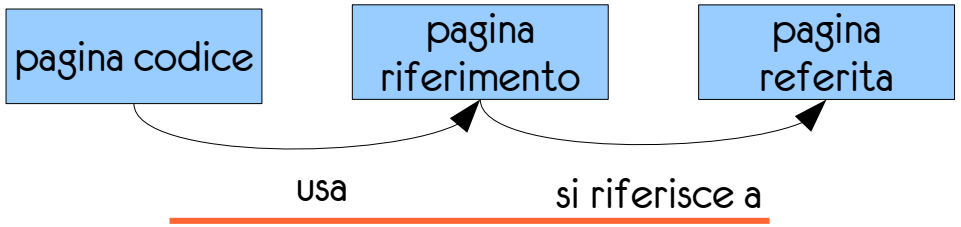
Numero minimo di frame

- Consideriamo un'istruzione con un solo operando
- È probabile che per caricarla ed eseguirla occorrano almeno due pagine:
  - una pagina di codice, contentente l'istruzione
  - una pagina di dati, contenente l'operando



Numero minimo di frame

- Se l'operando è un riferimento in memoria allora il numero di pagine sale a tre:
  - una pagina di codice, contentente l'istruzione
  - una pagina di dati, contenente l'operando
  - una pagina contentente il dato puntato dall'operando



per evitare page fault tutte e tre le pagine dovrebbero essere in RAM

Numero minimo di frame

- ...
- Inoltre:
  - un'istruzione può occupare uno spazio abbastanza grande da stare a cavallo di due pagine. Se ciò accade occorreranno almeno due pagine per il solo caricamento dell'istruzione
  - si possono avere più livelli di indirizzamento indiretto: nel caso peggiore tutta la memoria virtuale dovrebbe essere caricata in RAM per evitare page fault

Numero minimo di frame

- Il numero minimo di frame necessari al caricamento e all'esecuzione di un'istruzione dipende dall'architettura
- Supponiamo di avere M frame liberi ed N processi:
  - invece di applicare una paginazione su richiesta **pura** posso attuare una politica diversa e decidere di riservare ad ogni processo un certo numero di frame, caricando subito più pagine in RAM:
    - tornando ai nostri 100K di RAM liberi (frame da 1 K), se dobbiamo caricare 4 processi, possiamo pensare di assegnare a ciascuno 24 frame e di lasciarne 4 come pool di frame liberi (allocazione uniforme)
    - in alternativa: poiché i processi hanno dimensione diversa, alloco per ciascun processo un #frame proporzionale alla sua dimensione (allocazione proporzionale)

Allocazione proporzionale

- Indichiamo con VM^i la dimensione della memoria logica occupata dal generico processo P^i
- La quantità di memoria logica occupata da N processi sarà $V = \sum^{i \in [1,N]} VM^i$
- Indicando con M il numero di frame disponibili, il numero di frame allocati al processo P^i sarà:

$$m = \frac{VM^i}{V} \times M$$

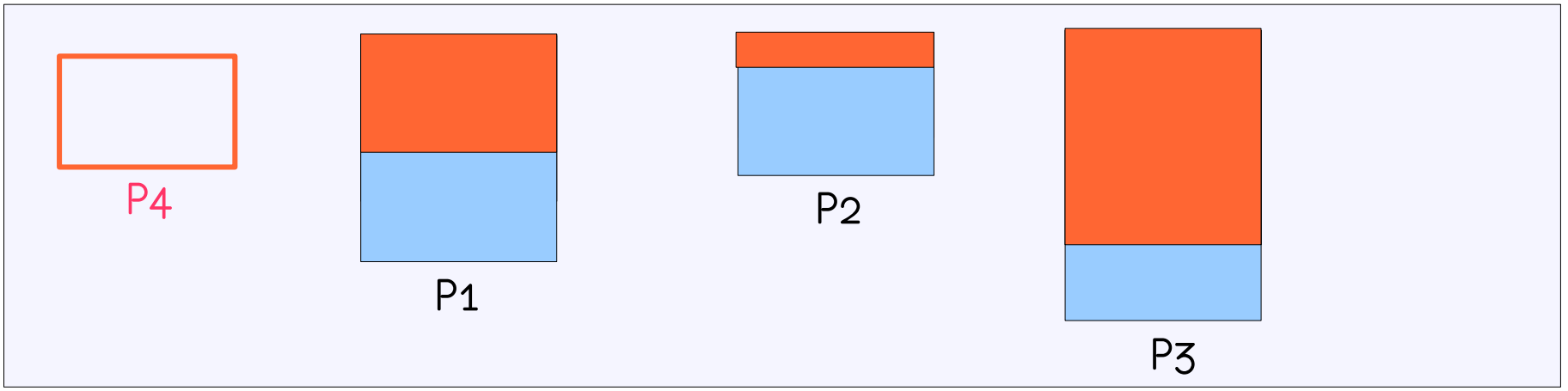
- Se abbiamo definito un num. minimo di frame necessari per caricare un'istruzione, potrebbe essere necessario incrementare tale valore di qualche unità per i processi più leggeri, decrementando di conseguenza i valori assegnati ai processi più pesanti

Dinamicità

- NB: il numero di processi in RAM è una funzione del tempo
- se il livello di multiprogrammazione cresce (num. processi in RAM aumenta) occorrerà redistribuire un certo numero di frame ancora liberi ai nuovi processi
- se il livello di multiprogrammazione diminuisce, sarà possibile assegnare ai processi in RAM un numero maggiore di frame

Riassumendo

In generale la RAM è suddivisa a priori fra i
processi secondo un certo criterio: ogni processo ha un tot di frame



In ogni istante una parte dei frame riservati x un processo sarà occupata
e una parte libera

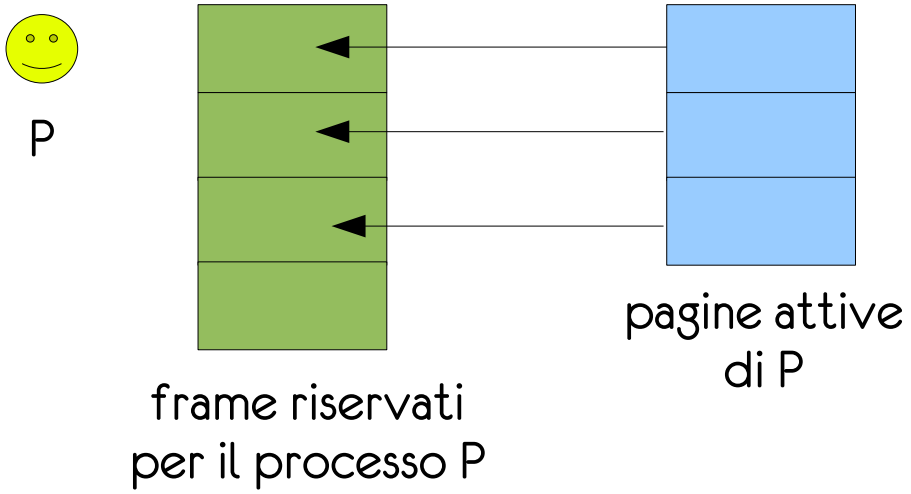
Di tanto in tanto l'ingresso di un nuovo processo causerà una riassegnazione dei frame rimasti liberi

Allocazione globale/locale

- Una questione che non abbiamo ancora considerato è la priorità dei processi in connessione all'allocazione dei frame
- Aspettativa: processi a priorità maggiore hanno a disposizione un maggior numero di frame
- Le strategie viste sono “eque” da questo punto di vista: i processi hanno tutti la stessa importanza
- Consideriamo il caso particolare in cui un processo ad alta priorità esaurisce il proprio pool di frame. Ci sono due possibilità:
  - si sacrifica una delle sue pagine, sostituendola con quella di interesse (allocazione locale delle pagine: uso solo le pagine riservate per il processo)
  - si sacrifica un altro processo, che ha un frame libero, sottraendoglielo (allocazione globale delle pagine, posso usare qualsiasi frame libero)

Thrashing

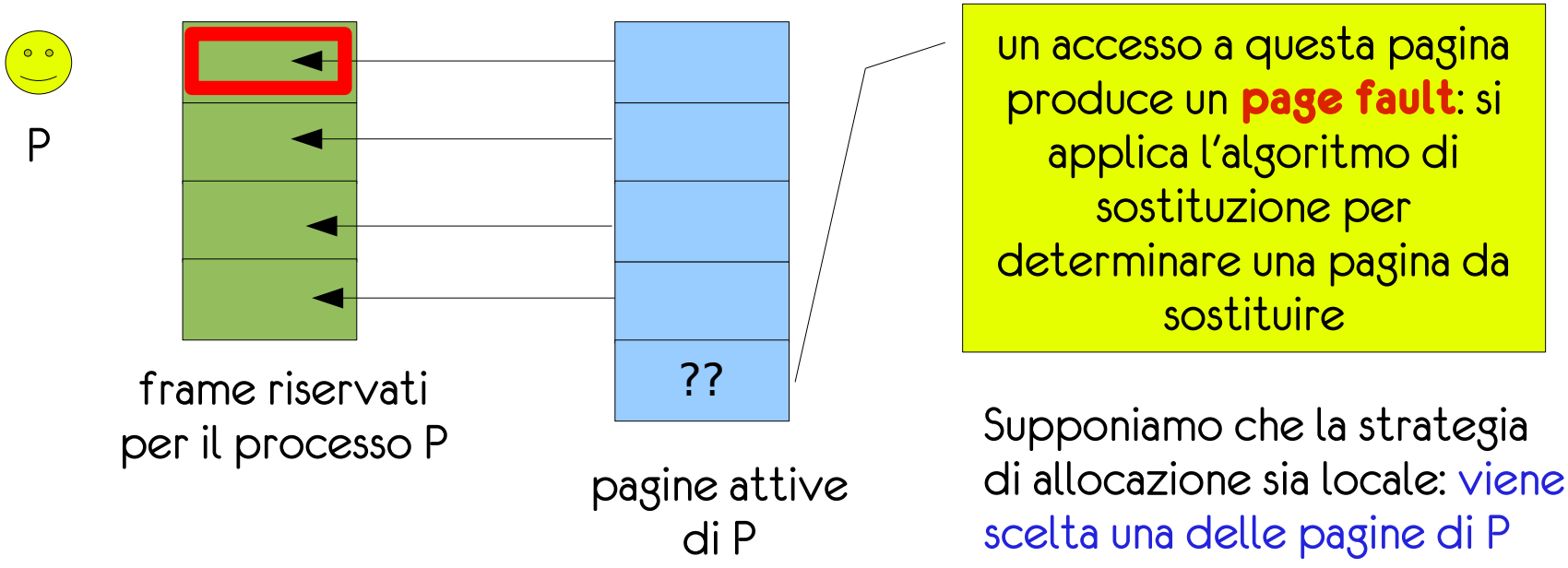
- un altro aspetto da discutere è un fenomeno noto come thrashing
- il thrashing è un fenomeno degenerativo che si può verificare nella gestione della memoria virtuale
- ogni processo ha un numero minimo di frame riservati
- l'esecuzione di ciascun processo si appoggia in ogni istante a un certo numero di pagine “attive”



Il numero di pagine attive cambia nel tempo: supponiamo che cresca: può succedere che a un certo punto vi siano più pagine attive che frame a disposizione

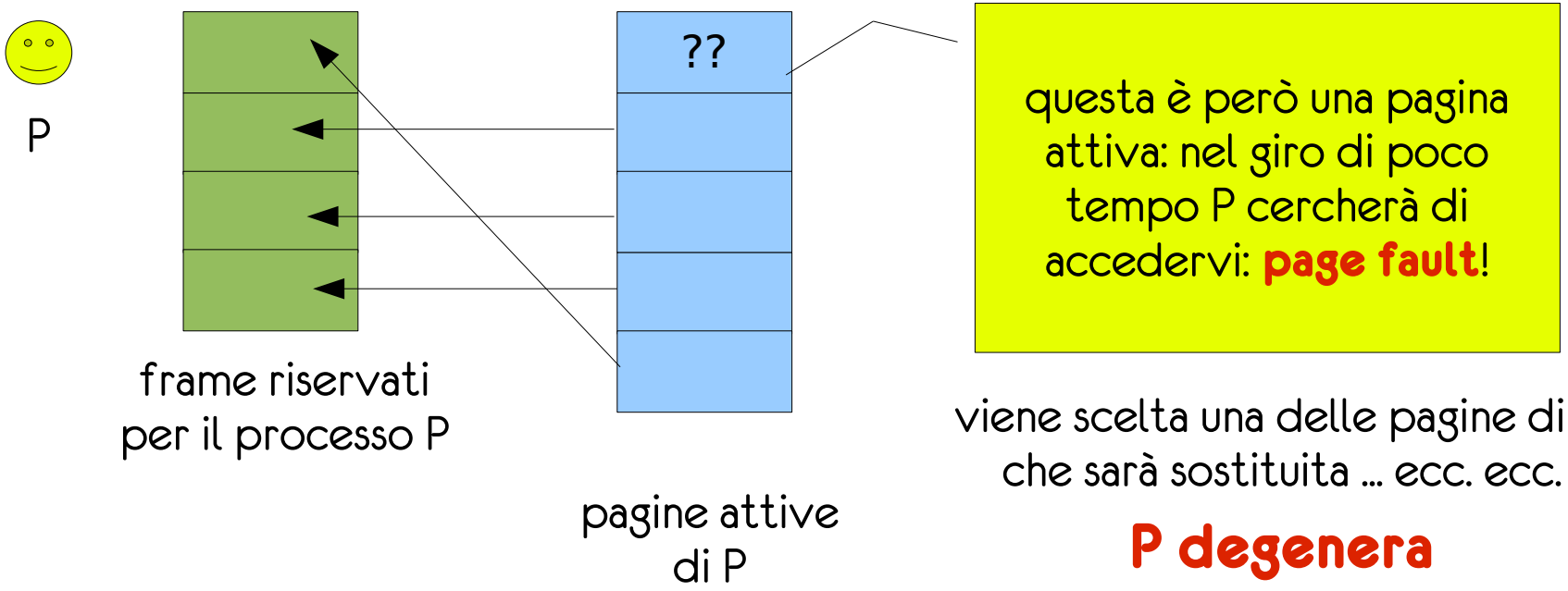
Thrashing

- un altro aspetto da discutere è un fenomeno noto come thrashing
- il thrashing è un fenomeno degenerativo che si può verificare nella gestione della memoria virtuale
- ogni processo ha un numero minimo di frame riservati
- l'esecuzione di ciascun processo si appoggia in ogni istante a un certo numero di pagine “attive”



Thrashing

- un altro aspetto da discutere è un fenomeno noto come thrashing
- il thrashing è un fenomeno degenerativo che si può verificare nella gestione della memoria virtuale
- ogni processo ha un numero minimo di frame riservati
- l'esecuzione di ciascun processo si appoggia in ogni istante a un certo numero di pagine “attive”

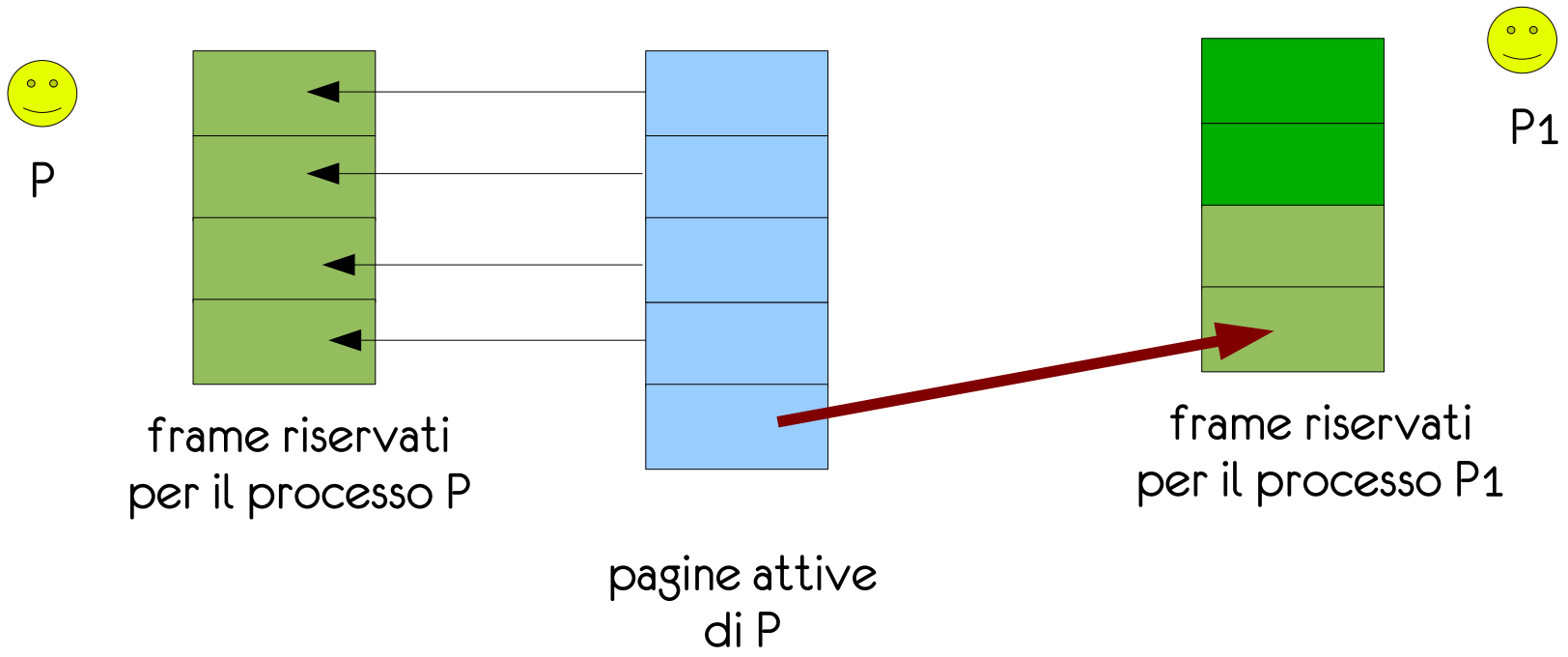


Thrashing

- Quando il numero di pagine attive è maggiore del numero minimo di frame riservato per il processo ed occorre caricare una nuova pagina (page fault), il normale processo di sostituzione sceglierà come vittima una pagina attiva (per forza di cose)
- a questo punto si innesta un meccanismo perverso: per proseguire il processo si produce un page fault, che genera una sostituzione che rimuove una pagina utile quasi immediatamente, si produce un nuovo page fault, che sostituisce una pagina attiva, e così via
- in breve si spende più tempo nel sostituire pagine che nell'eseguire il processo: si parla di thrashing

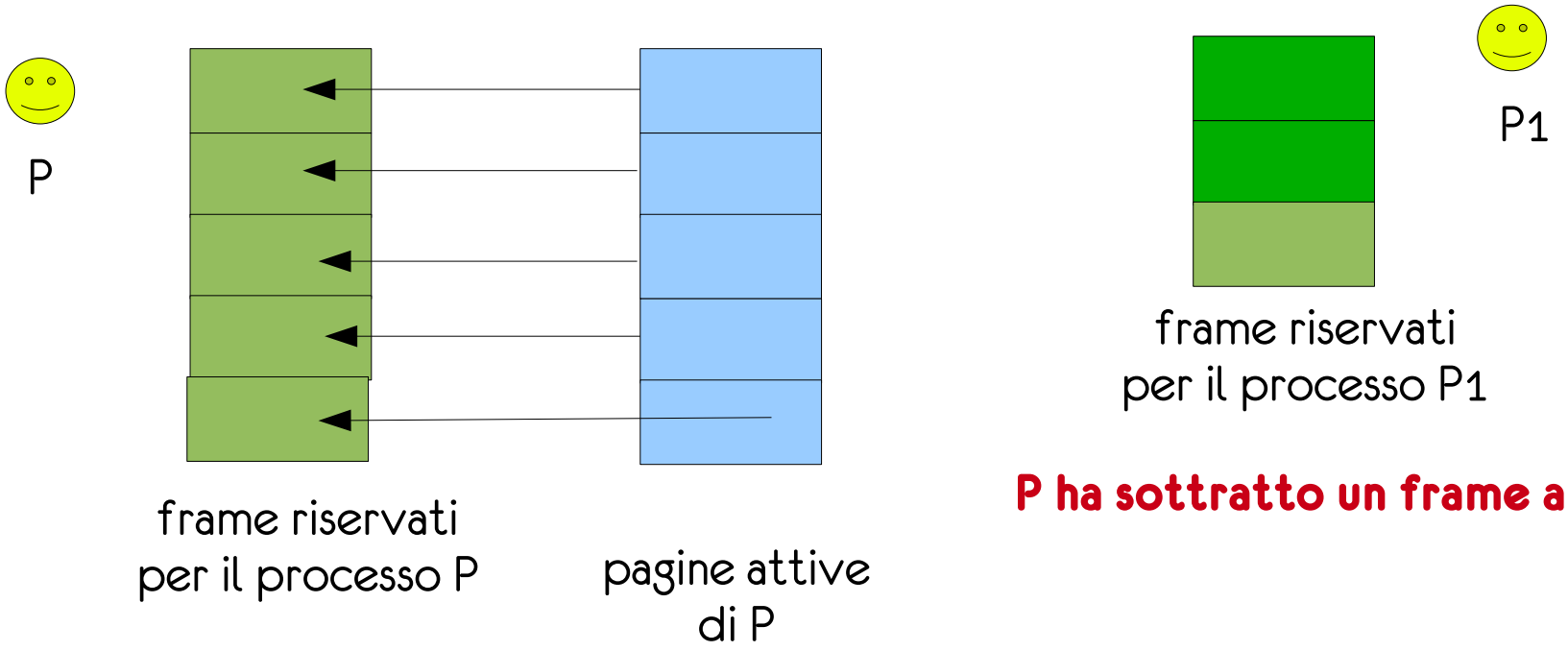
Thrashing

- Possibili alternative?
- e se il meccanismo di allocazione fosse globale?
- in questo caso potrebbe essere scelto un frame di un'altro processo



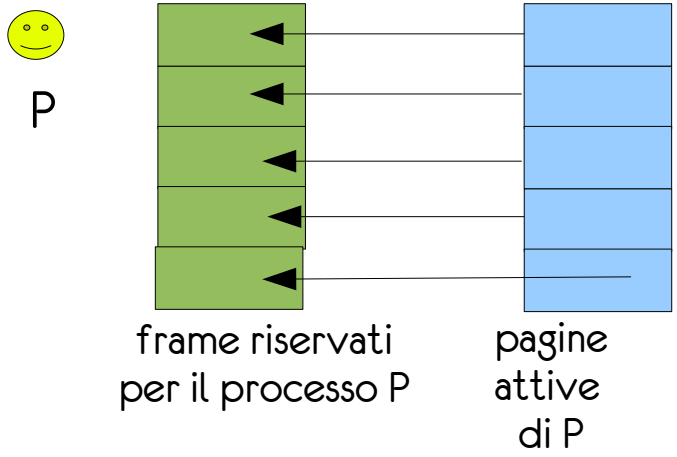
Thrashing

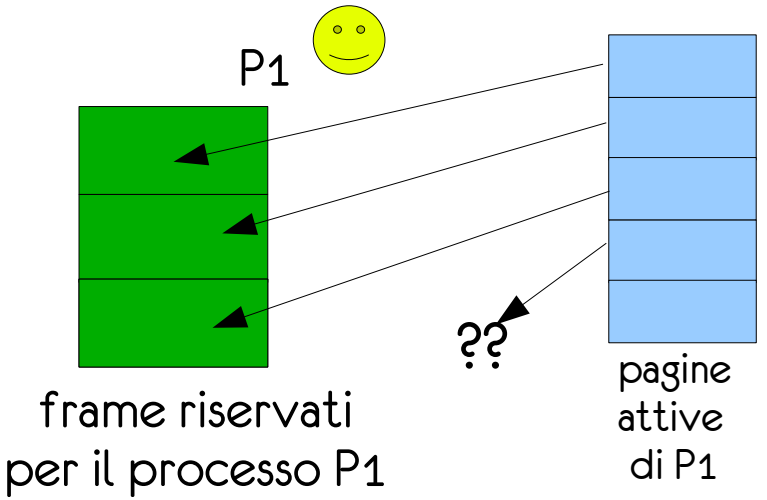
- e se il meccanismo di allocazione fosse globale?
- in questo caso potrebbe essere scelto un frame di un'altro processo



Thrashing

- e se il meccanismo di allocazione fosse globale?
- in questo caso potrebbe essere scelto un frame di un'altro processo

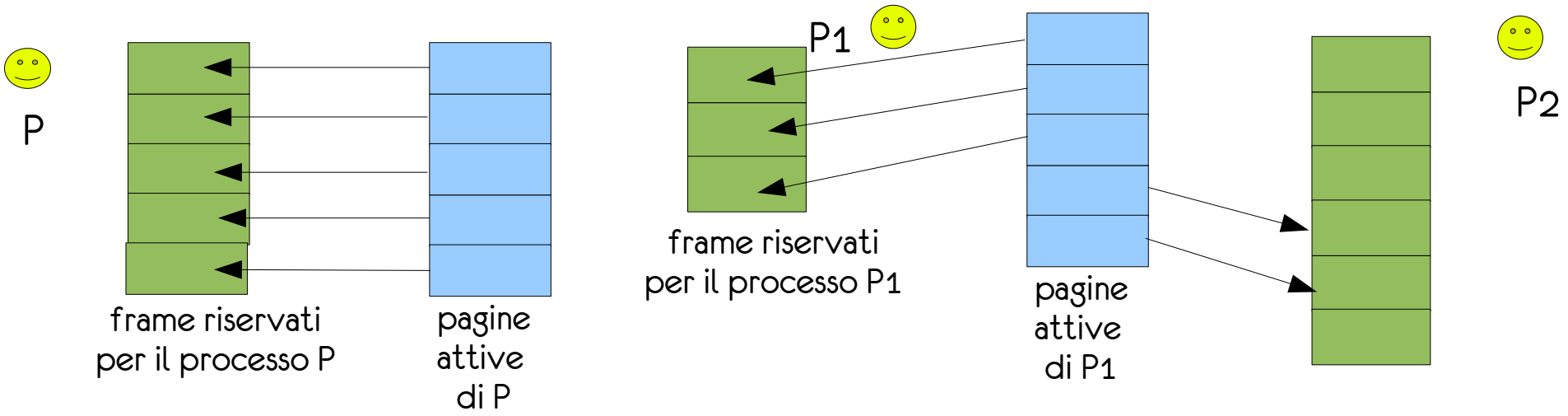




A questo punto se il numero di pagine attive per P1 cresce, P1 riempirà i propri frame residui ...

Thrashing

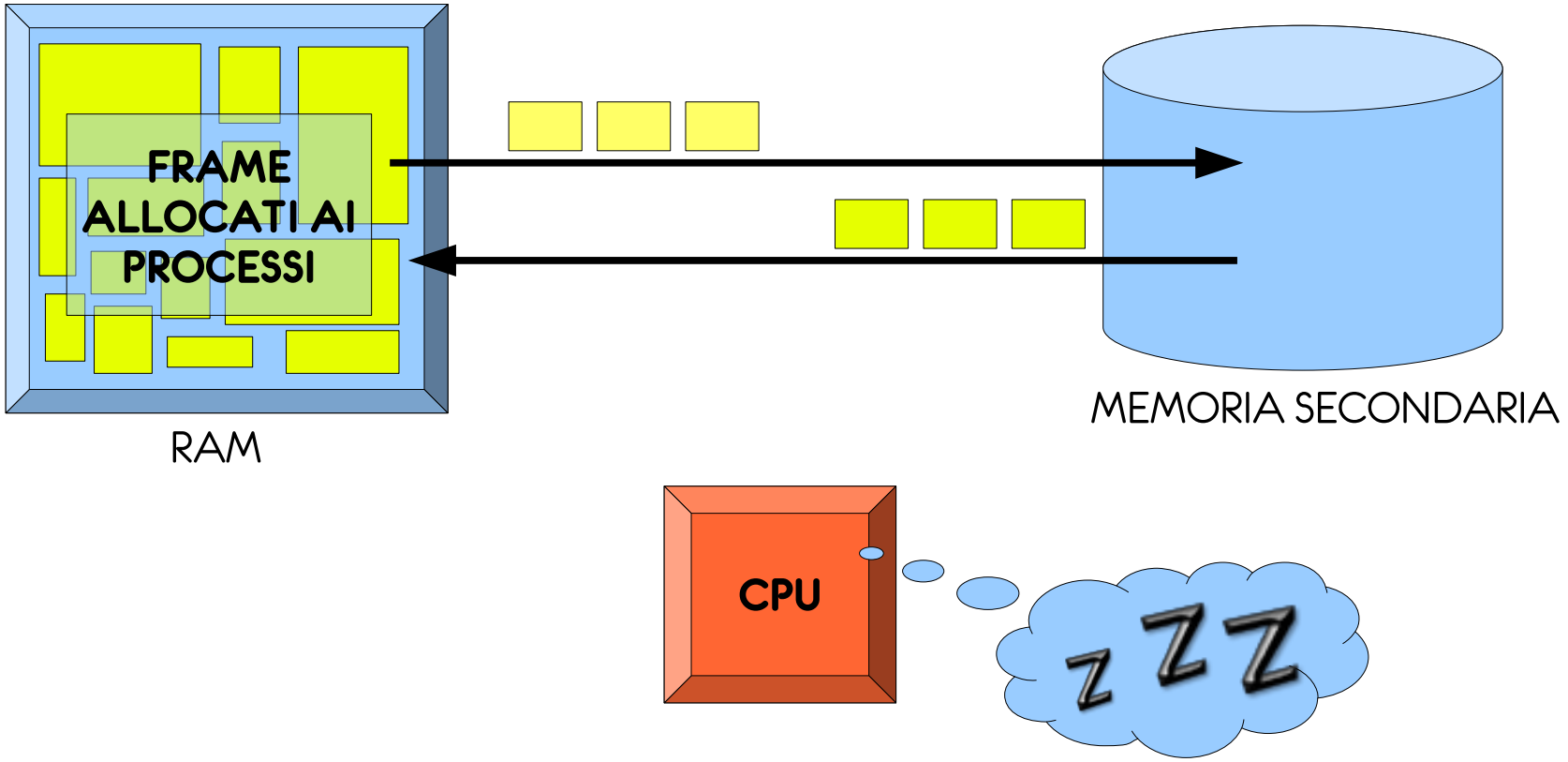
- e se il meccanismo di allocazione fosse globale?
- in questo caso potrebbe essere scelto un frame di un'altro processo



A questo punto se il numero di pagine attive per P2 cresce, P2 riempirà i propri frame residui ...

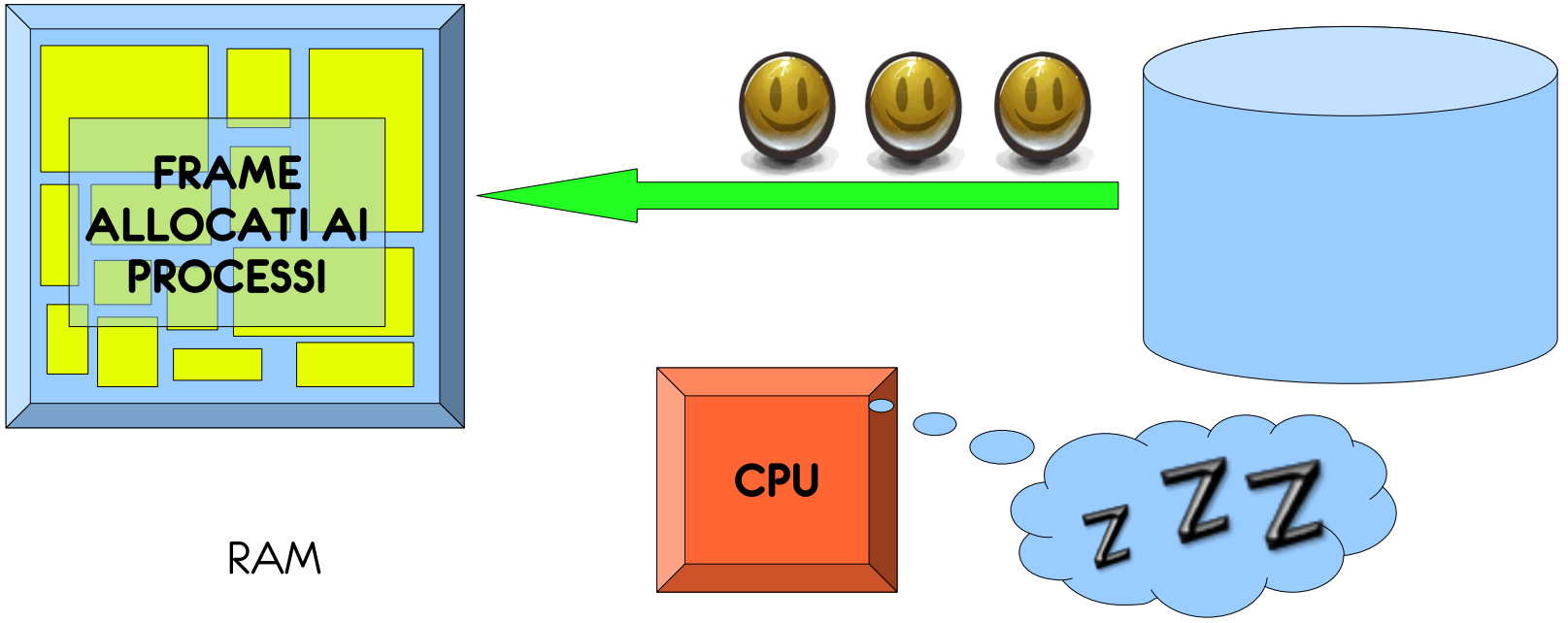
Effetti del thrashing

- quando si ha thashing perché i processi hanno a disposizione meno frame del numero di pagine attive, l'attività della CPU tende a diminuire: i processi tendono a rimanere in attesa del completamento di operazioni di I/O



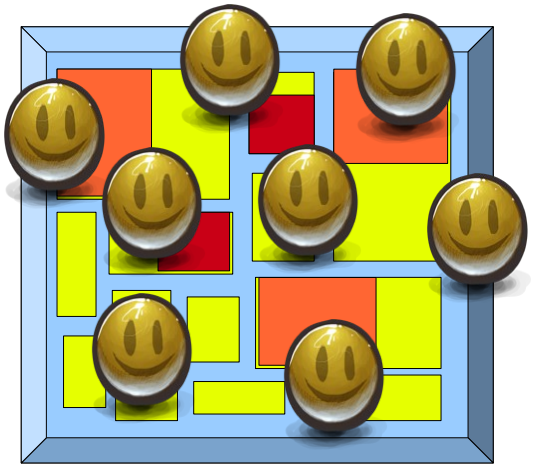
Thrashing e multiprogrammazione

- Molti SO monitorizzano l'uso della CPU: quando questo cala, se ci sono processi conservati in memoria secondaria, portano in RAM qualche processo in più ...



Thrashing e multiprogrammazione

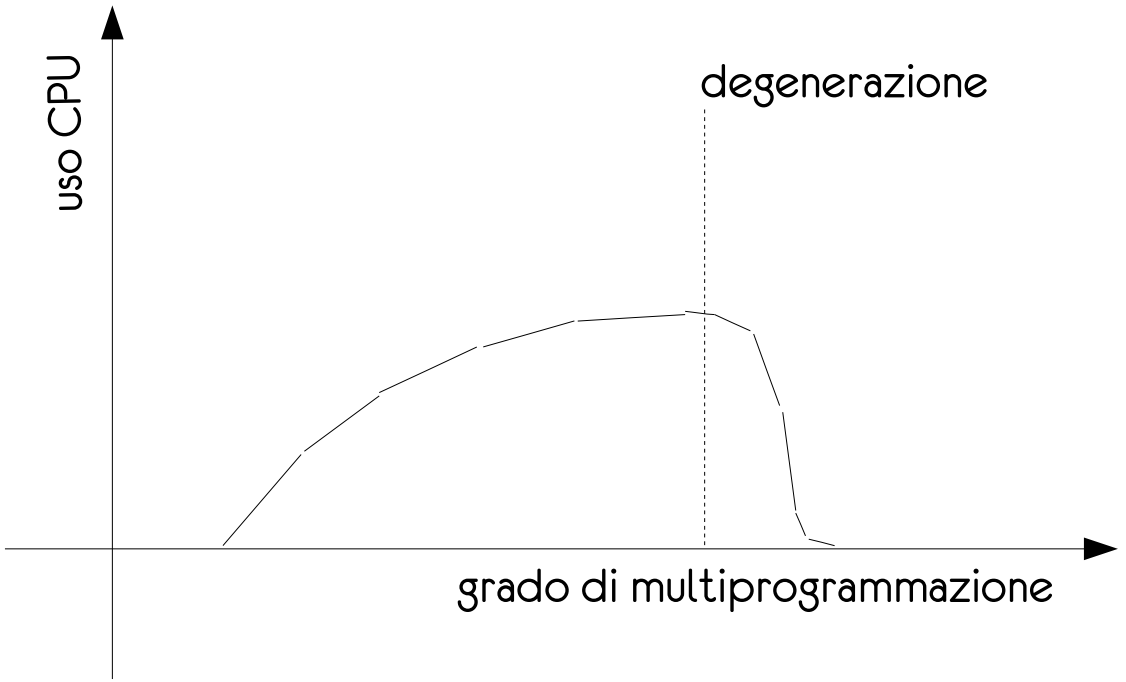
- quando si ha thashing perché i processi hanno a disposizione meno frame del numero di pagine attive, l'attività della CPU tende a diminuire (molto I/O)
- Molti SO monitorizzano l'uso della CPU: quando questo cala, se ci sono processi conservati in memoria secondaria, portano in RAM qualche processo in più ...
- Ai nuovi processi vengono allocati alcuni frame ...
- ... sottratti ad altri processi in RAM se ...
- ... la strategia di allocazione è globale
- aumenta la frequenza di page fault
- diminuisce l'utilizzo della CPU
- ecc. fino al blocco totale



RAM

Andamento

- se si monitora l'uso della CPU, in presenza di thrashing si ha questo andamento



Cambiando strategia ...

- Si ha thrashing perché la politica di allocazione dei frame è globale: il SO può sottrarre frame ad un processo per assegnarli ad un altro processo
- un processo degenere, sotteaendo frame agli altri processi, renderà degeneri pure questi
- se adottassimo un algoritmo locale, un processo degenere non potrebbe rendere degeneri altri processi perché non potrebbe sottrarre loro dei frame
- anche in questo caso però, la frequenza di page fault del processo degenere influenzerebbe il tempo di accesso effettivo degli altri processi a causa del sovraccarico causato al dispositivo di paginazione
- ... è il meglio che possiamo fare ?

Pagine attive

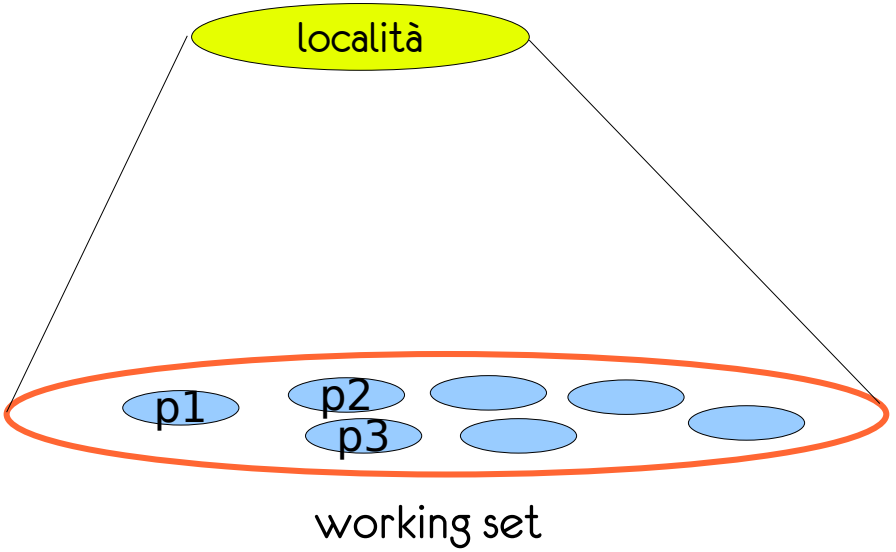
- l'ideale è cercare di prevedere di quante pagine avrà bisogno un processo e assegnargli un numero di frame sufficiente
- L'allocazione non sarà di tipo uniforme né sarà di tipo proporzionale ma dipenderà dal numero di pagine attive per ciascun processo
- non si distribuiscono tutti i frame liberi fra i processi, solo quelli ad essi necessari
- poiché il numero di pagine attive varia nel tempo:
  - se cresce si allocano nuovi frame
  - se decresce si liberano frame
- questo approccio si realizza nel modello a **Working Set**

Working set

- l'esecuzione dei processi può essere descritta sulla base di un **principio di località** che cattura il fatto che ogni processo accede, per periodi di tempo di durata consistente, solo a un sottoinsieme delle variabili globali, a un certo insieme di variabili locali e a un sottoinsieme delle istruzioni che compongono il suo codice
- con il termine **working set** si intende l'insieme delle pagine attive di un processo, cioè l'insieme delle pagine che il processo sta usando
- tale insieme varia nel tempo, per es. invocando una nuova procedura si può focalizzare l'esecuzione su un diverso working set

Working set

- il working set può essere visto come l'insieme delle pagine che implementa una località del processo
- il concetto di località può essere visto come un'astrazione del concetto di working set



Località e working set

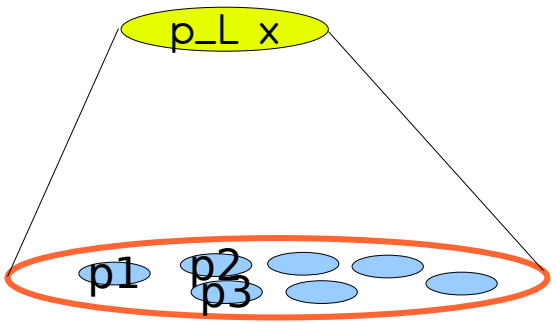
```
void push(lista *p_L, double x)
{
    printf("push %.2f \n",x);

    if(p_L == NULL)
    {
        *p_L = (lista) malloc (sizeof(struct nodo));
        if(p_L == NULL) return;
        (*p_L) -> info = x;
    }

    else ...
```

Finché permango in una località se il SO ha caricato tutto il working set relativo non si avranno page fault

Quando eseguo la funzione push uso le variabili locali p_L e x ed eseguo le istruzioni che in parte vediamo: la località è data dal codice di push e dalle variabili p_L e X. Tale località si implementa in un insieme di pagine che contiene effettivamente il codice in questione e le variabili locali menzionate



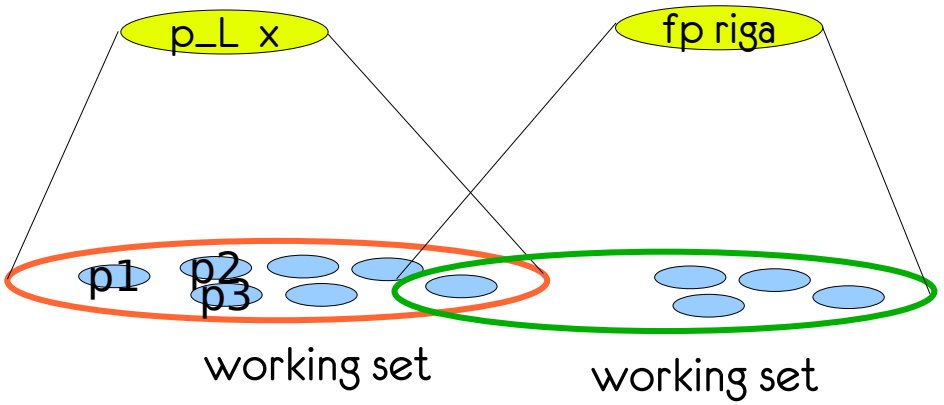
working set

Località e working set

```
FILE *fp = fopen("dati", "r");
char riga[128];

fread(riga, 128, 1);
while(!feof(fp)) {
      printf("%s\n", riga);
      fread(riga, 128, 1);
}
```

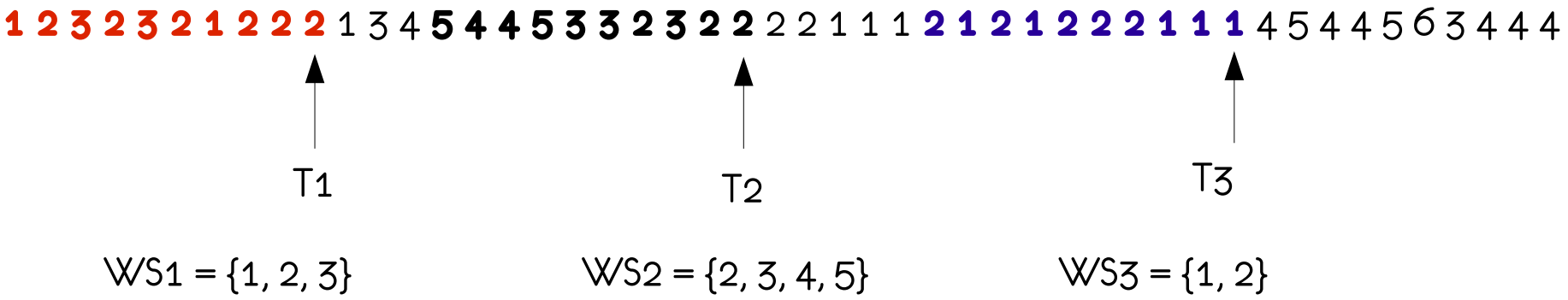
cambiando località
cambierà l'insieme delle pagine
coinvolto nell'esecuzione



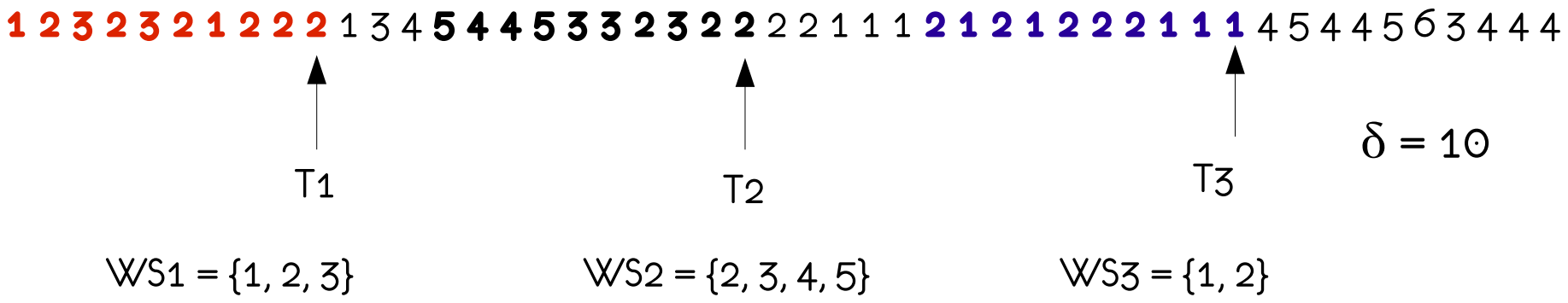
i diversi WS possono avere sovrapposizioni

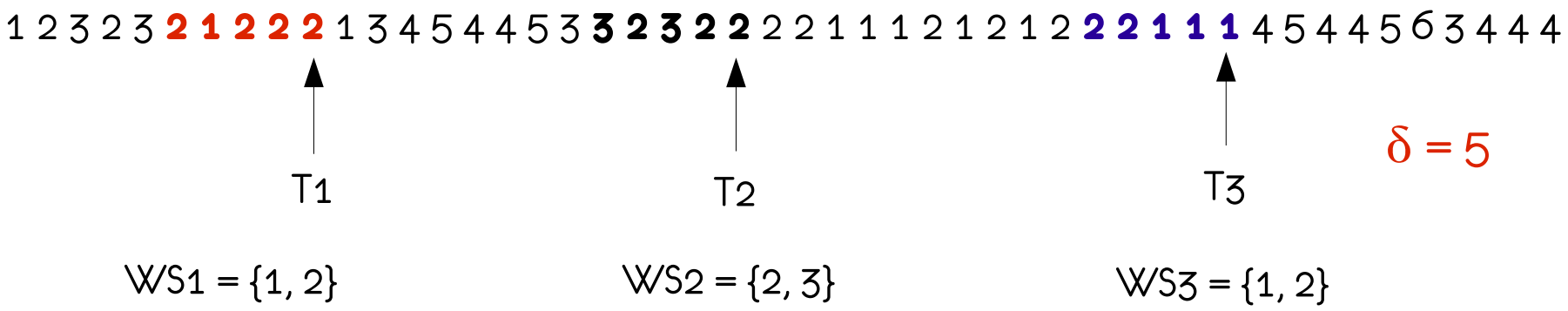
Modello del working set

- si basa sul principio di località
- viene definito un parametro δ (ampiezza della finestra di analisi)
- per ogni processo si analizzano gli ultimi δ riferimenti alle pagine
- le pagine individuate costituiscono il working set corrente
- es. sia $\delta = 10$

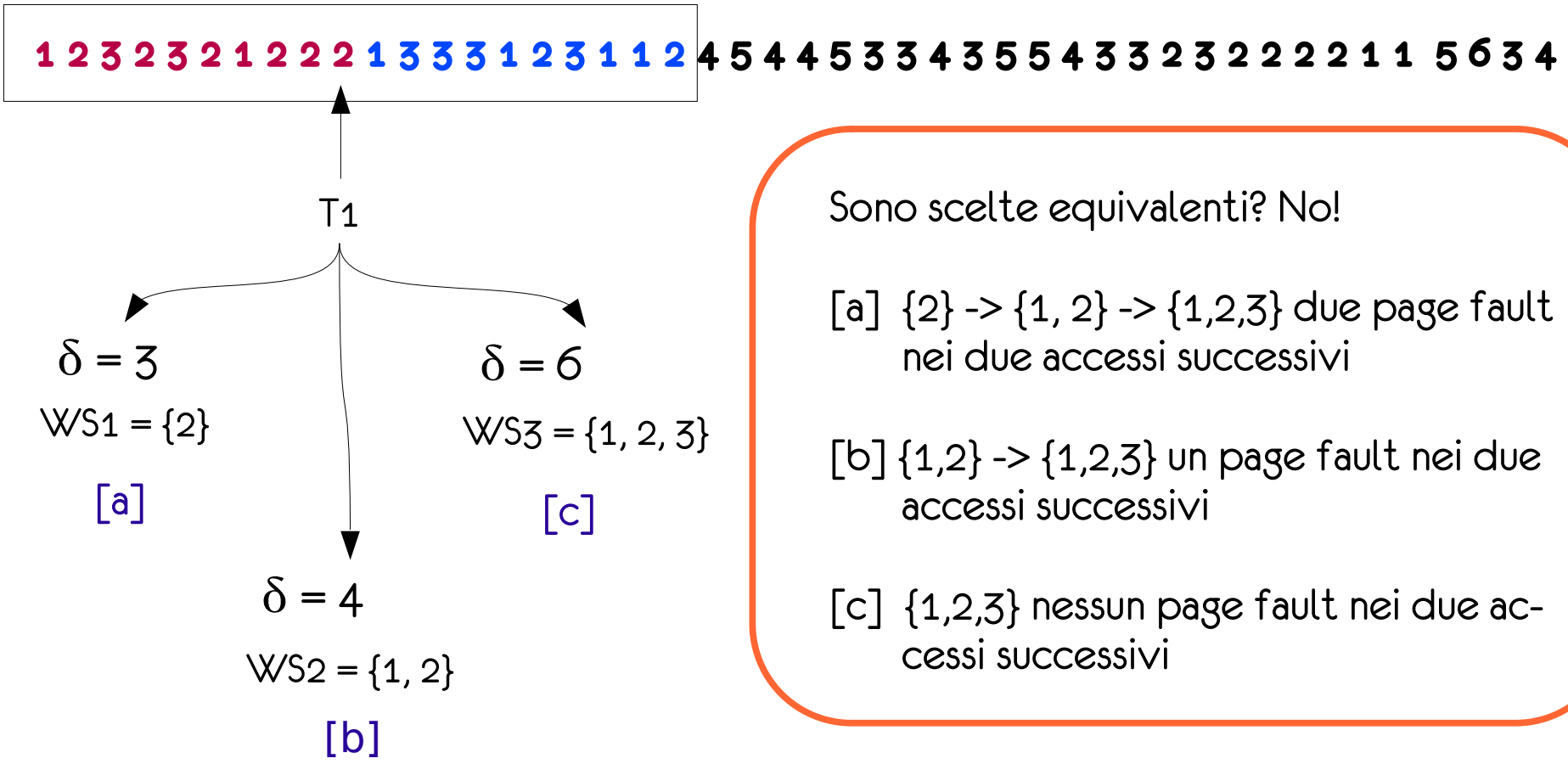


Working set e δ





Working set e δ



Sono scelte equivalenti? No!

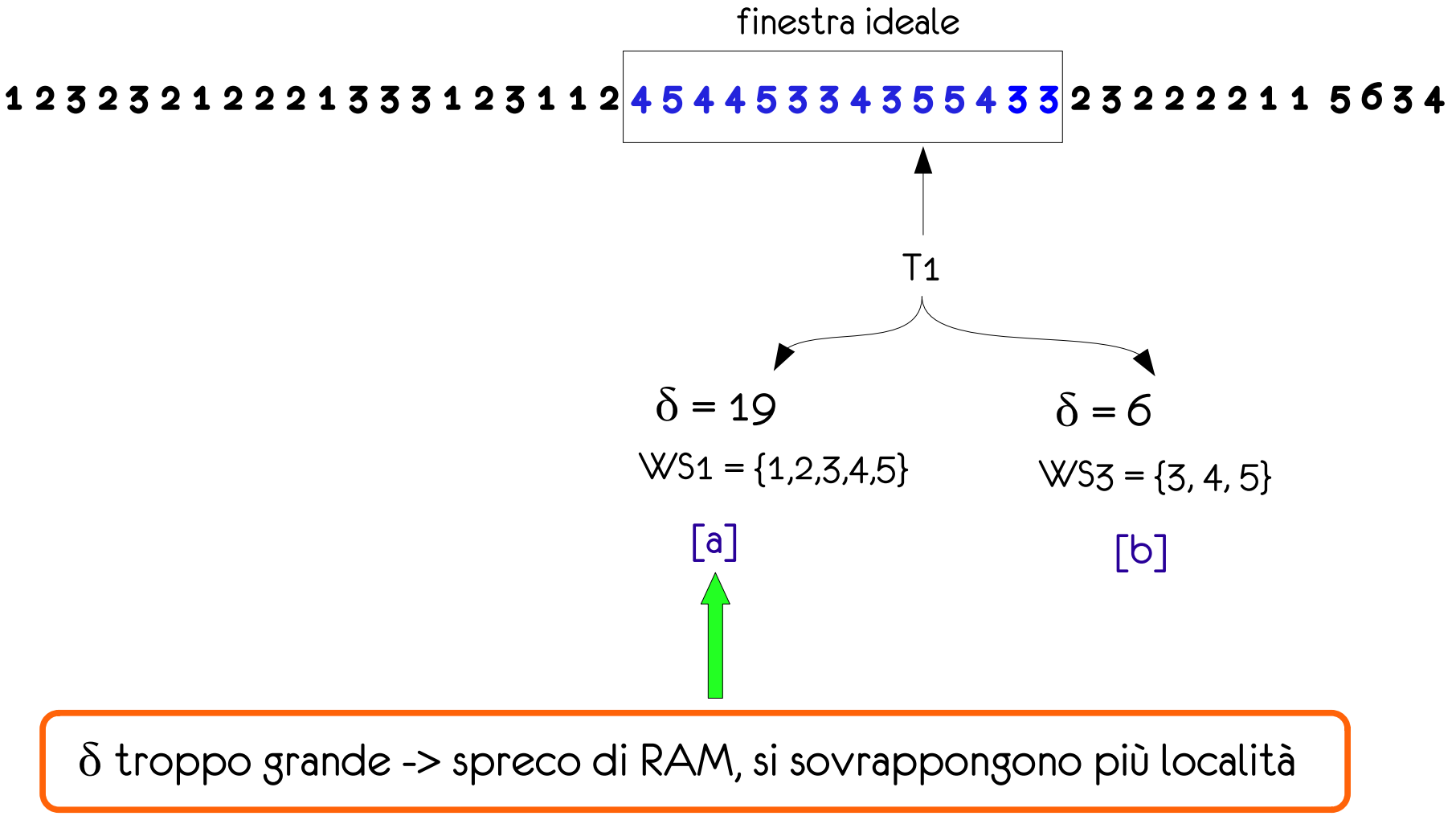
[a] {2} -> {1, 2} -> {1,2,3} due page fault nei due accessi successivi

[b] {1,2} -> {1,2,3} un page fault nei due accessi successivi

[c] {1,2,3} nessun page fault nei due accessi successivi

δ troppo piccolo non coglie l'intera località

Working set e δ



δ troppo grande -> spreco di RAM, si sovrappongono più località

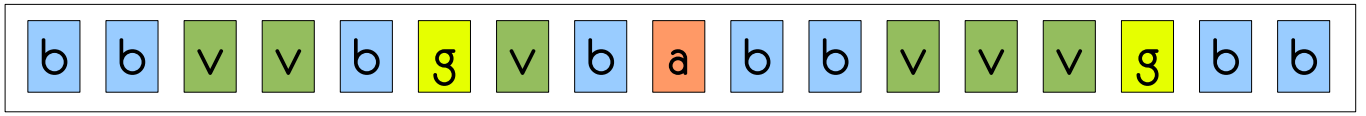
Uso del working set

- Definita la dimensione della finestra δ è possibile calcolare il WS di ogni processo e quindi calcolare la quantità di frame attualmente necessari ai processi in esecuzione:

$$D = \sum^{i \in [1, N]} |WS^i|$$

- Quando D diventa maggiore del numero di frame liberi, si genera il thrashing perché qualche processo non ha a disposizione un numero sufficiente di frame
- Se i frame sono invece sufficienti il SO può assegnare a ciascun processo una quantità di frame adeguata
- Se c'è un surplus di frame liberi è possibile avviare nuovi processi
- Quando il WS di un processo cresce, se non ci sono frame liberi, il SO sceglie uno dei processi, lo copia in memoria secondaria, lo sospende e assegna (parte dei) suoi frame al processo richiedente: così si evita il thrashing

Tener traccia del Working set



- Una pagina fa parte di un WS se esiste almeno un riferimento ad essa all'interno della finestra di analisi
- La verifica viene fatta a intervalli regolari (timer), es. ogni 1000 riferimenti
- δ nella realtà è un valore $\approx$ 10.000 / 15.000
- Occorre determinare un modo efficiente per calcolare il WS di un processo
- Soluzione: si possono usare i bit di riferimento e i registri a scorrimento già descritti parlando di algoritmi di sostituzione delle pagine

Prepaginazione

- **problema inevitabile:**
  - per via del modo in cui sono definite, a ogni cambiamento di località si ha un certo numero di page fault (caricamento della nuova località):
  - es. 1 2 3 2 2 1 1 2 3 2 3 4 5 6 5 5 4 6 6 4 5 5 4
- **problema evitabile**:
  - un problema a cui si può invece ovviare, legato alla paginazione pura (*carico una pagina solo quando viene riferita*), riguarda l'eliminazione dei page fault ad ogni swap in del processo
  - se si memorizza, associato al PCB del processo, anche il WS del processo al momento della sospensione, è possibile caricare subito in memoria tutte le pagine utili, senza dover passare attraverso a una serie di page fault
  - questa tecnica è nota come **prepaginazione**

Page fault frequency

- L'approccio a working set ha riscosso un notevole successo (anche perché consente di effettuare la prepaginazione)
- per evitare il fenomeno del thrashing in sé, esistono tecniche alternative
- in particolare la strategia basata sulla frequenza delle assenze di pagina (page fault frequency)
- La frequenza dei page fault aumenta considerevolmente in presenza di thrashing, un modo per evitare la degegnerazione è porre un limite superiore alla frequenza di page fault accettata
- se un processo supera il limite gli si allocherà un nuovo frame
- d'altro canto, il fatto che tale frequenza scenda molto può essere sinonimo di un WS sovradimensionato