Gestione della Memoria

Introduzione ai Sistemi Operativi

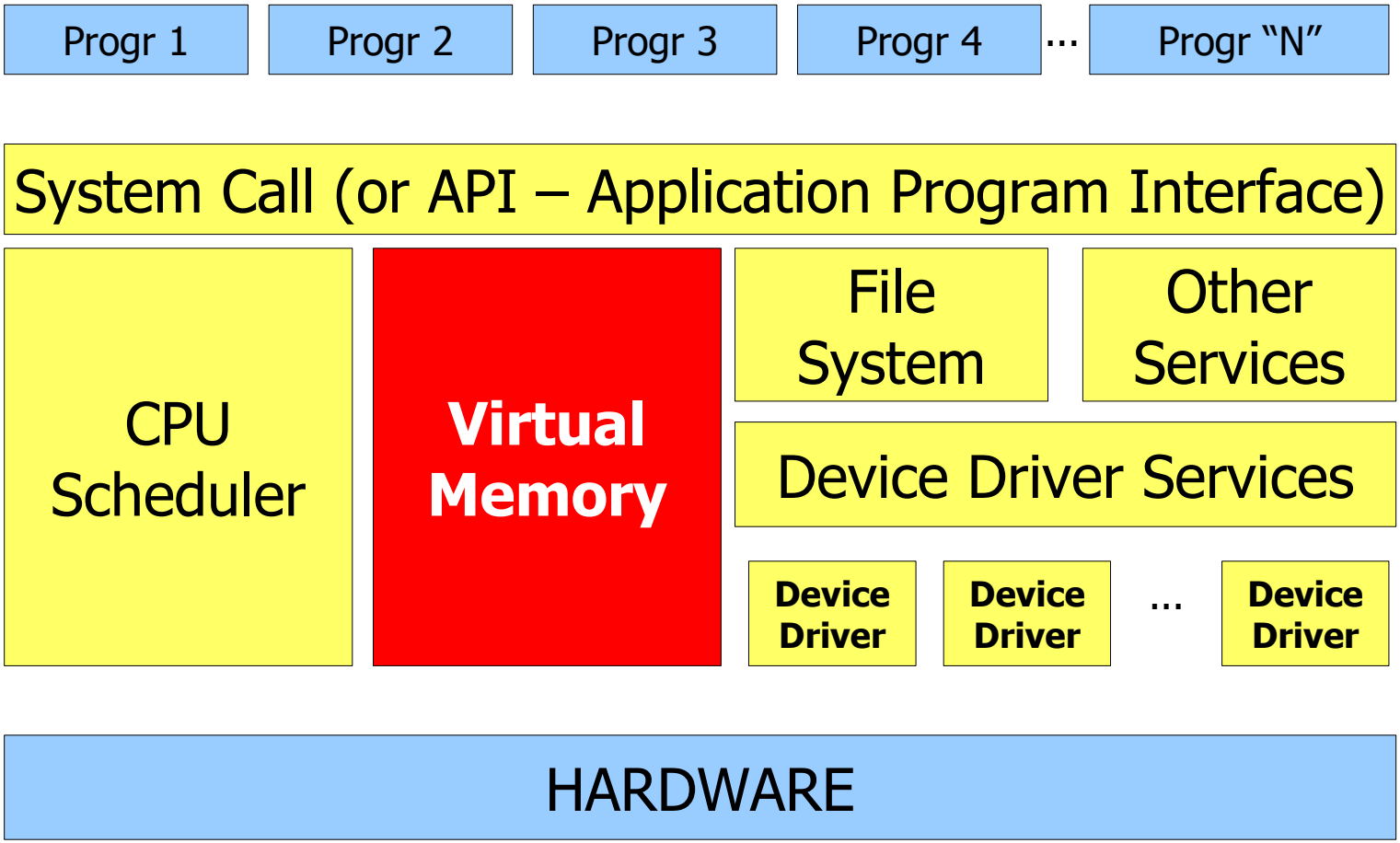
Corso di Abilità Informatiche

Laurea in Fisica

prof. Ing. Corrado Santoro

A.A. 2010-11

Architettura di un sistema operativo

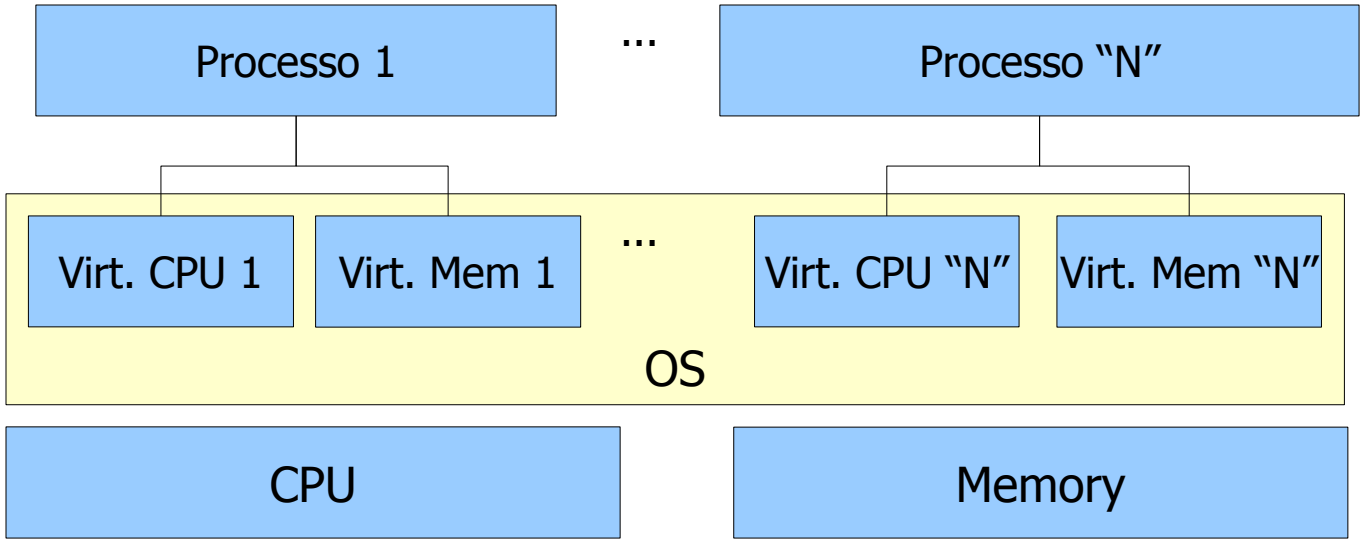


Gestione della memoria

- Ogni programma in esecuzione usa una certa quantità di memoria
- Ogni programma in esecuzione **deve** possedere **la sua** memoria
- La totalità della memoria fisica deve essere opportunamente **"spezzettata"** tra i vari programmi in esecuzione
- Cosa accade se un programma richiede una memoria **maggiore** di quella a lui assegnata?

Gestione della memoria

- Compito del sistema operativo è "virtualizzare" la memoria fisica allo scopo di offrire, ad ogni programma in esecuzione un proprio spazio di indirizzamento **contiguo** e **privato**

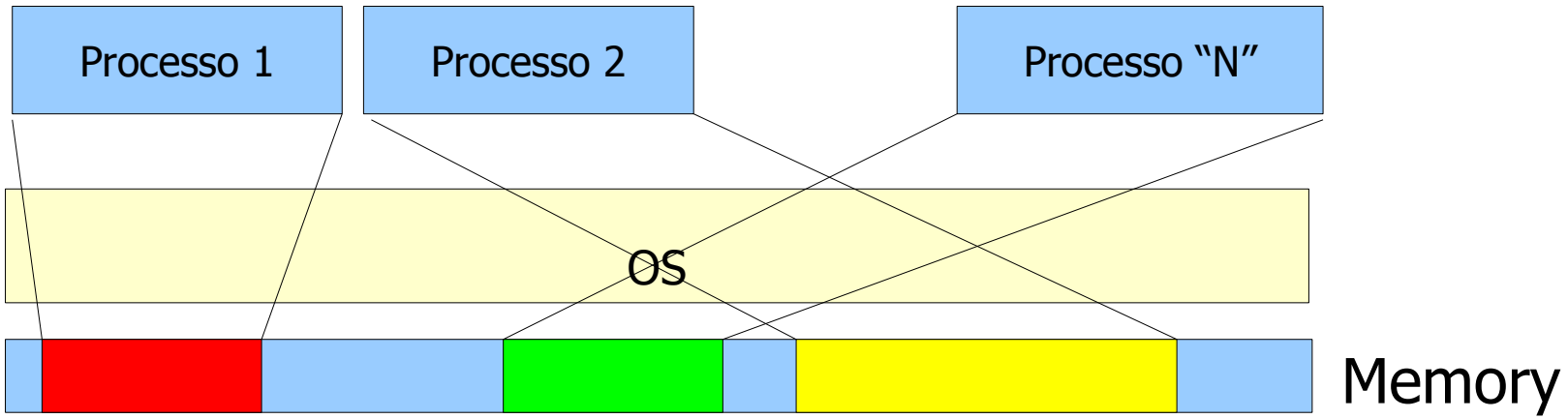


Tecniche di gestione della memoria

- **Allocazione contigua**
  - Molto semplice
  - Poco efficiente
  - Costosa dal punto di vista computazionale
- **Paginazione**
  - Più complessa
  - Molto efficiente
  - Poco costosa computazionalmente
  - Supportata da hardware specifico

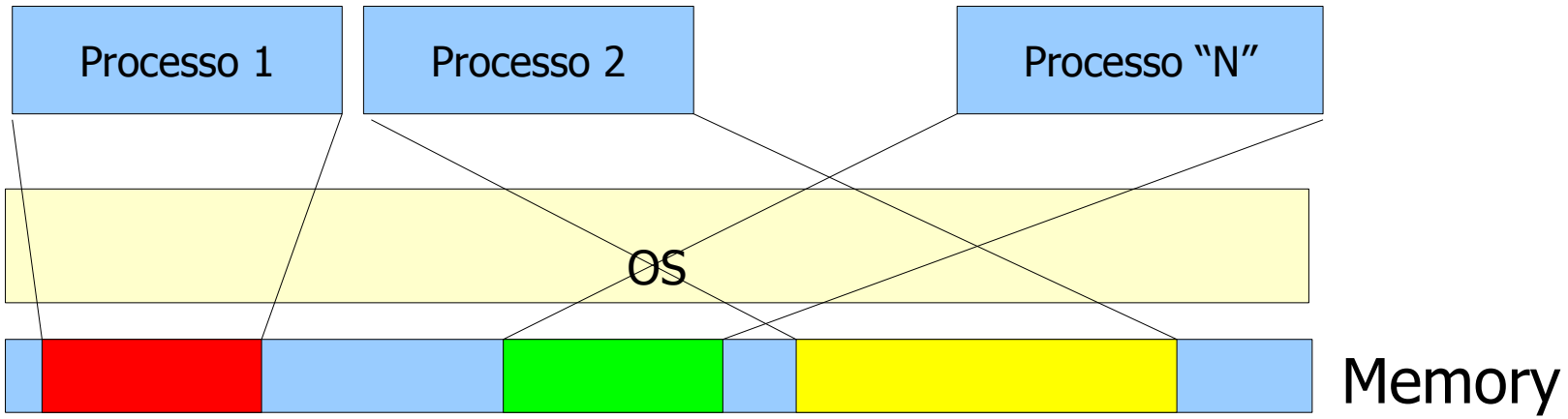
Allocazione contigua

- In funzione delle esigenze dei processi, il sistema operativo suddivide la memoria centrale e ne assegna una partizione ad ogni processo
- Il sistema operativo tiene traccia:
  - delle zone di memoria assegnate
  - dei processi a cui ha assegnato ogni zona
  - delle zone di memoria libere

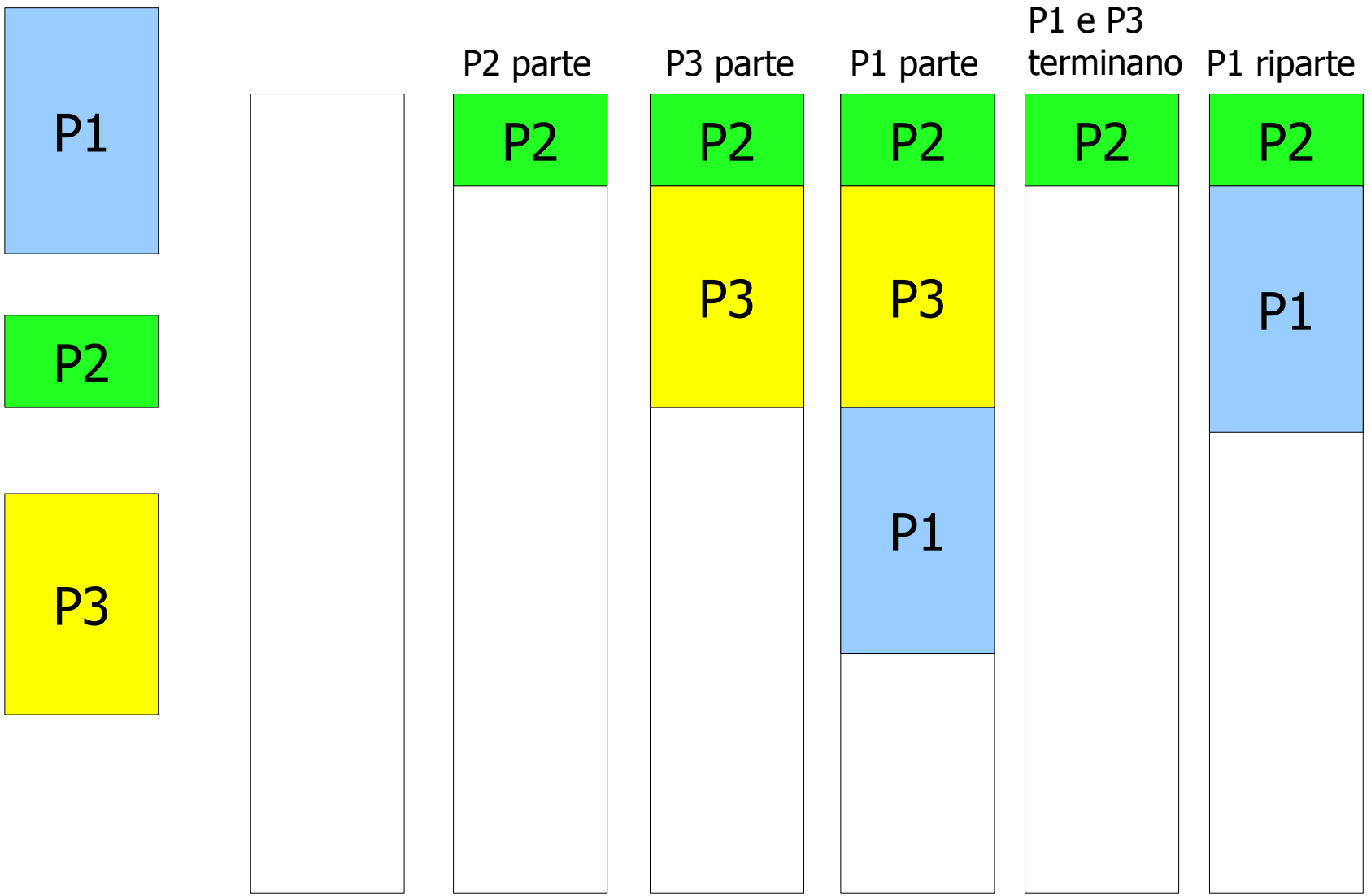


Allocazione contigua

- La memoria è quindi "bucherellata" in
  - zone contigue assegnate ai processi
  - zone libere (buchi)
- La zona di memoria **fisica** assegnata ad un processo è stabilita a **runtime**
- **Quindi per ogni esecuzione dello stesso programma, la zona di memoria assegnata può cambiare**



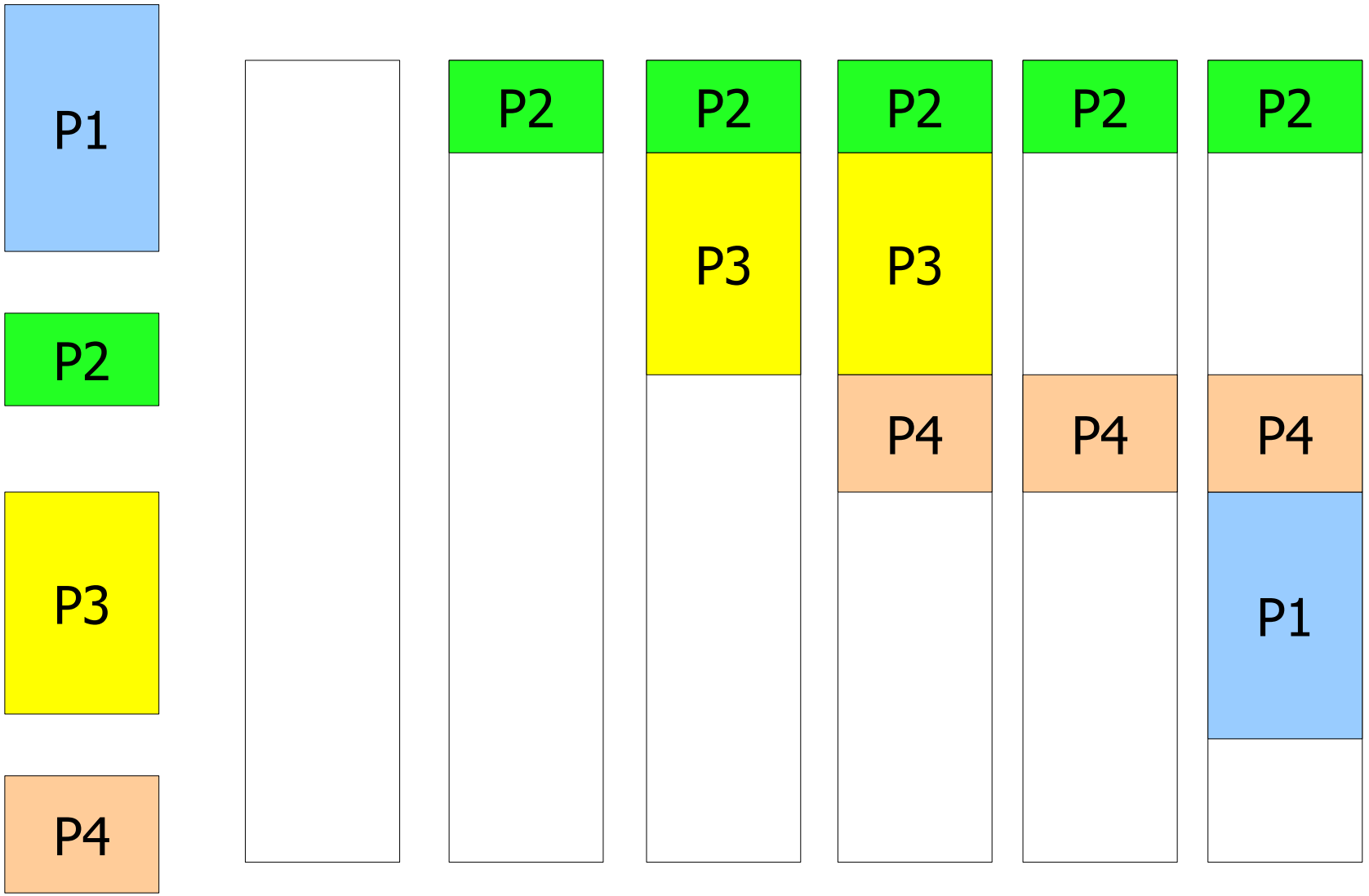
Allocazione contigua



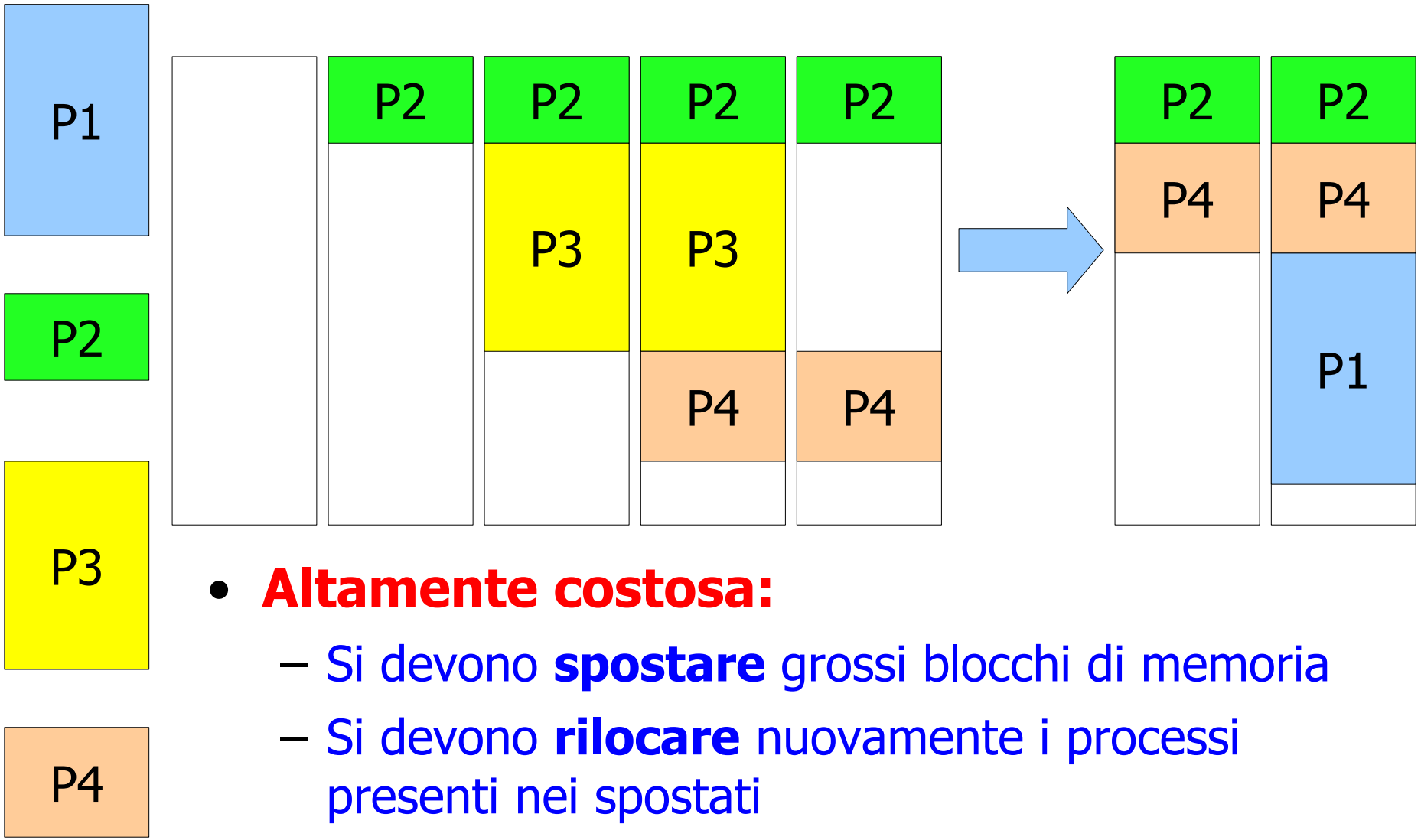
Frammentazione

- Man mano che i processi terminano, la memoria da loro usata viene liberata
- Tuttavia rimane un "buco" (di dimensioni N) che può essere occupato **solo** da un processo di dimensioni <= N
- Con l'andare del tempo, si creano molti buchi piccoli
- Si arriva al paradosso:
  - Processo di dimensioni K
  - Tanti buchi di dimensione $N_i < K$
  - Il processo non può essere caricato nonostante:
    - $\mathbf{\Sigma N_i > K}$

Frammentazione

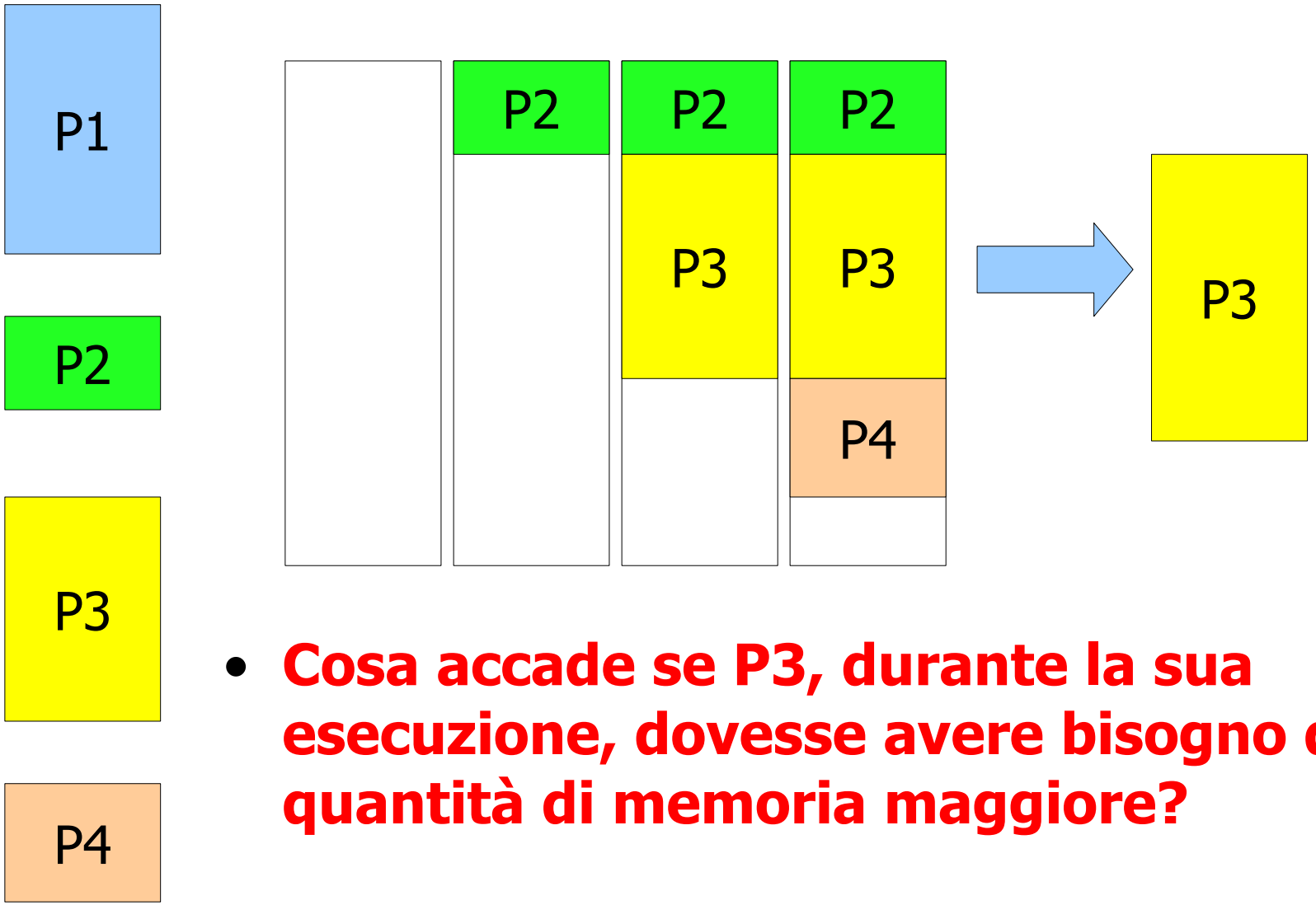


Frammentazione: Compattazione



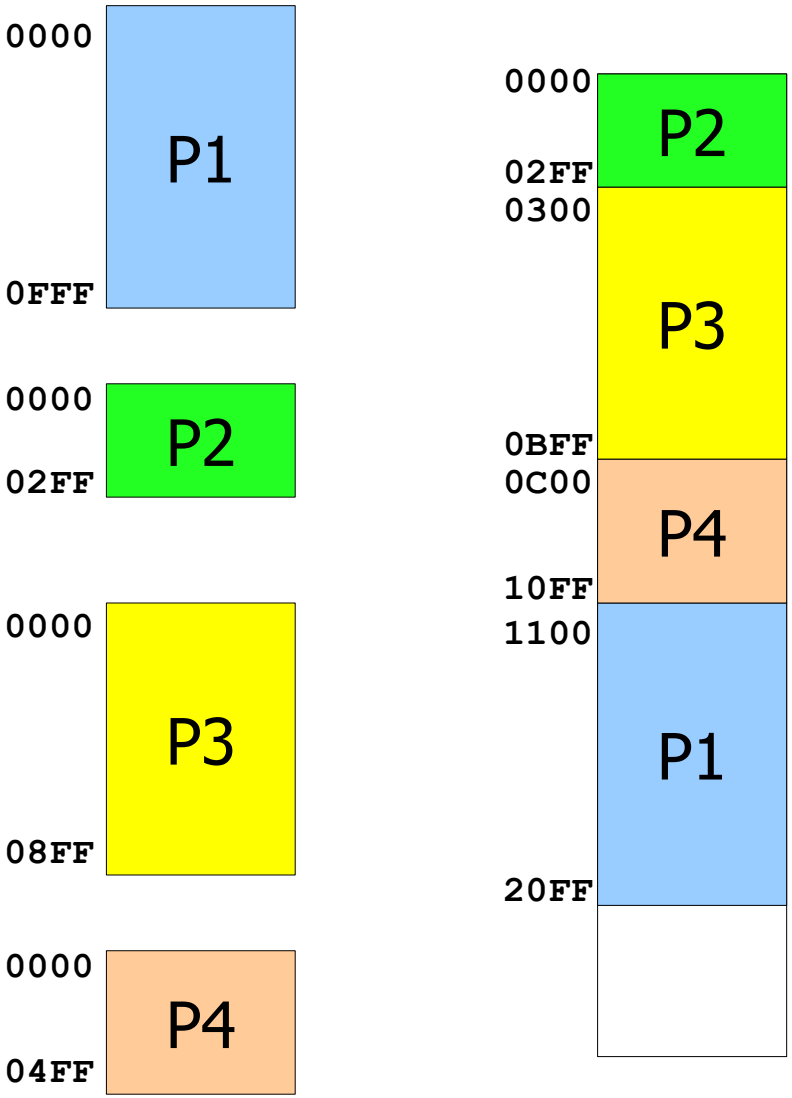
- **Altamente costosa:**
  - Si devono **spostare** grossi blocchi di memoria
  - Si devono **rilocare** nuovamente i processi presenti nei spostati

Crescita della memoria occupata



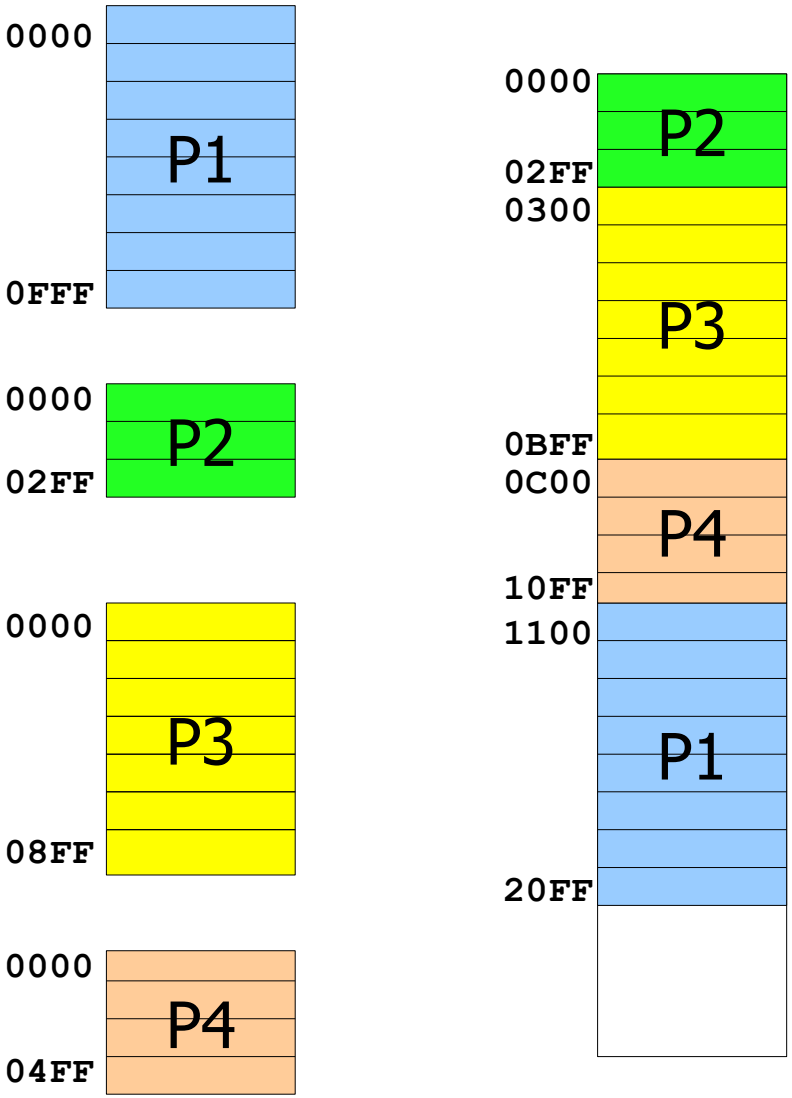
- **Cosa accade se P3, durante la sua esecuzione, dovesse avere bisogno di una quantità di memoria maggiore?**

Indirizzamento virtuale



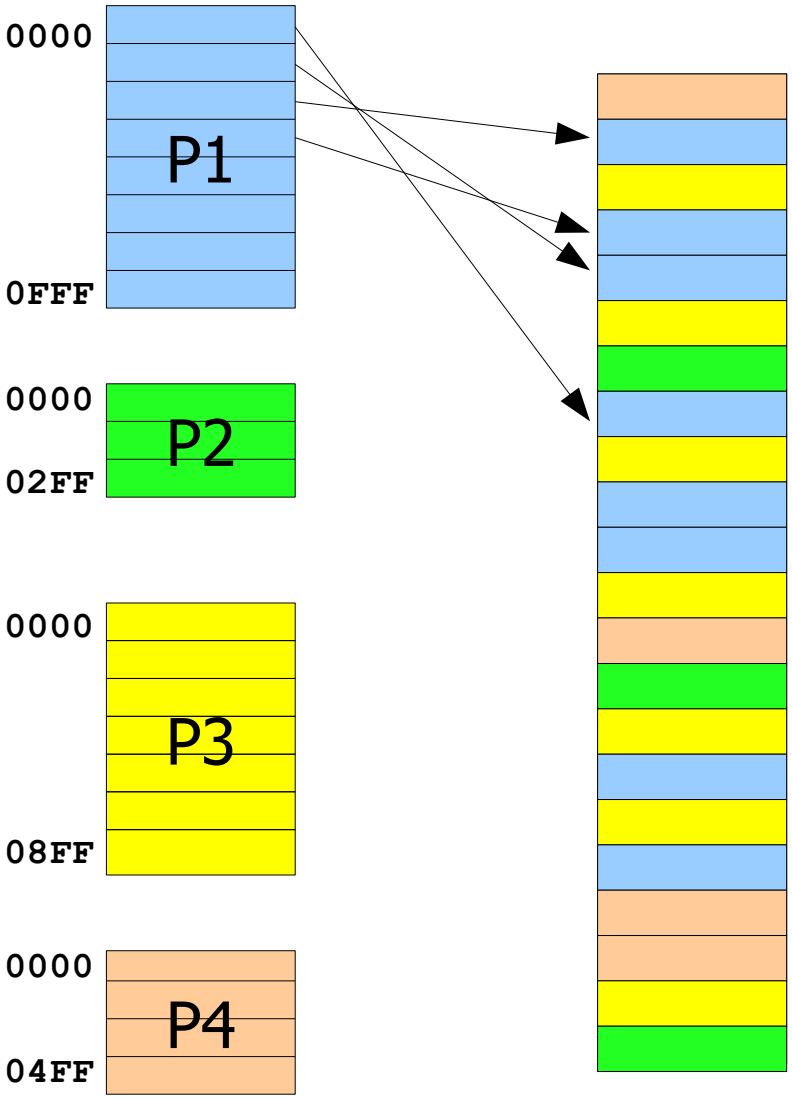
- Ogni programma può essere caricato in qualunque posizione in memoria
- Il modo con cui un processo "vede" la memoria deve essere indipendente dalla sua posizione
- Il processo vede degli indirizzi **logici/virtuali** non **fisici**
- Vi è un mapping 1:1 tra l'**intero spazio logico**/virtuale del processo e l'**intero blocco di memoria fisica** assegnato

Verso il paging (paginazione)



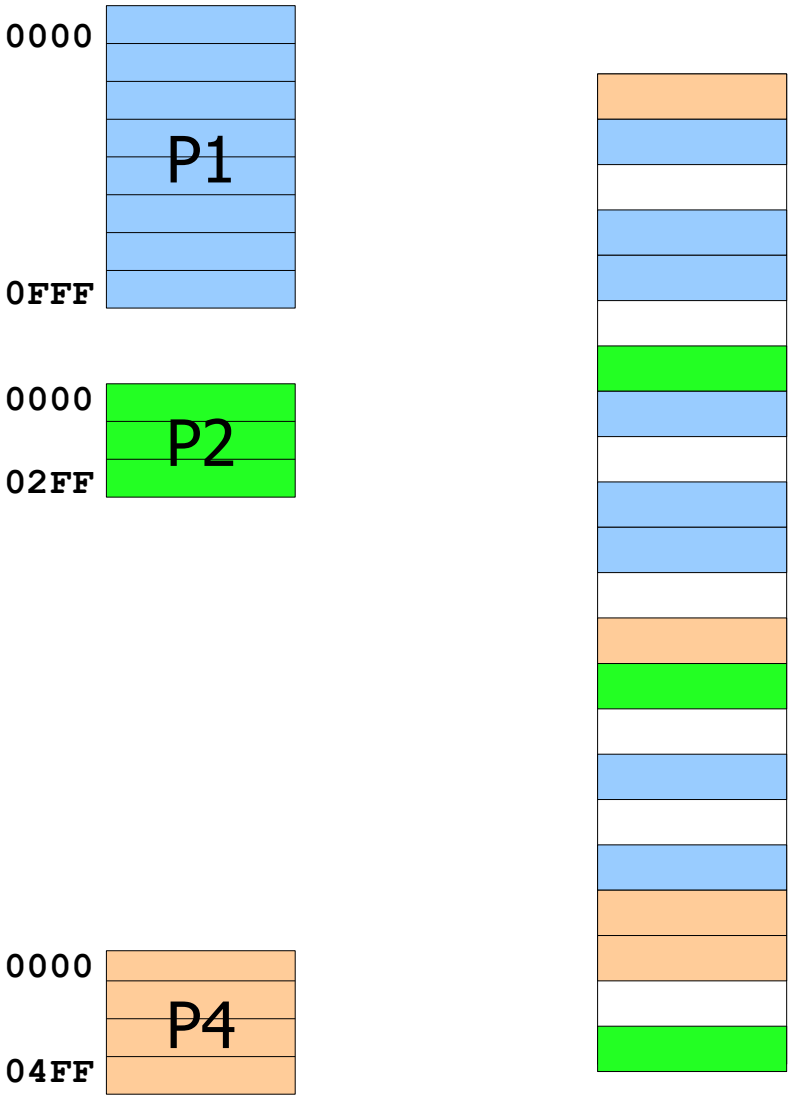
- Ora supponiamo di **suddividere** lo spazio logico e fisico in blocchetti di **dimensione fissa**
- ... mantenendo lo stesso tipo di mapping tra blocco logico e blocco fisico

Verso il paging (paginazione)



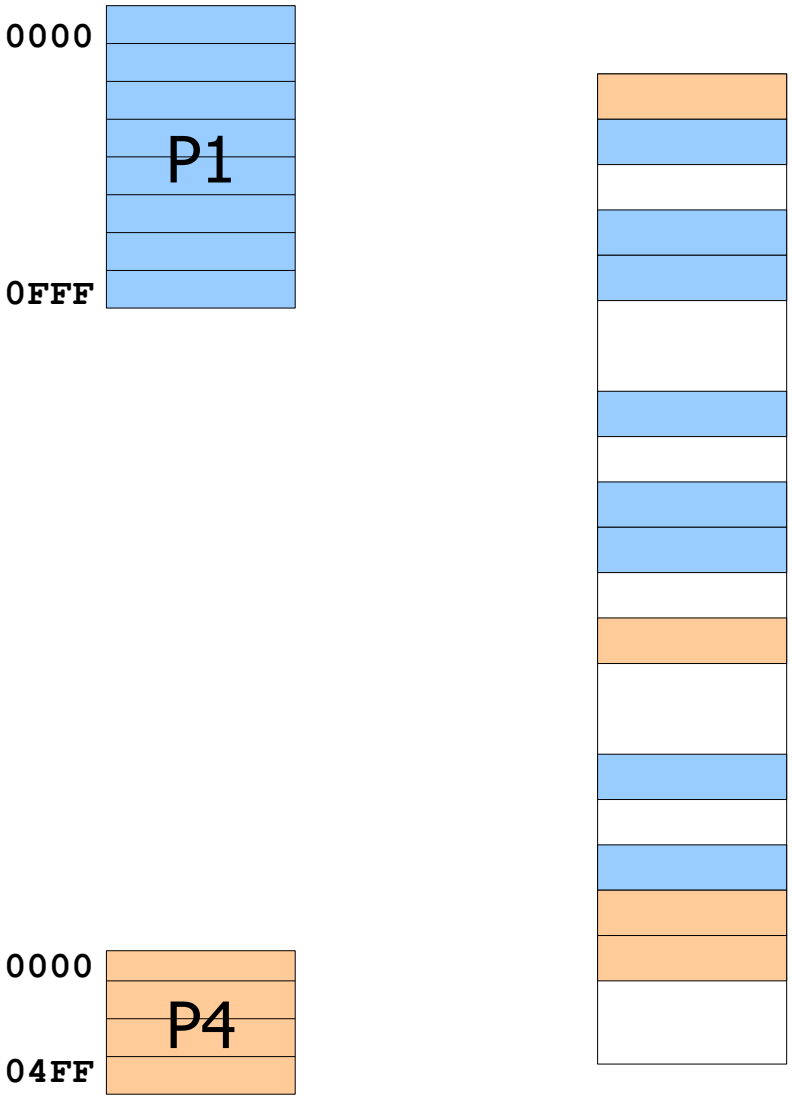
- Ora supponiamo di poter effettuare il mapping tra i **singoli blocchetti** e non più tra i blocchi di memoria interi
- I blocchetti di memoria fisica **non devono più essere contigui**
- **Si eliminano i problemi di allocazione e frammentazione**

Va via P3 ...



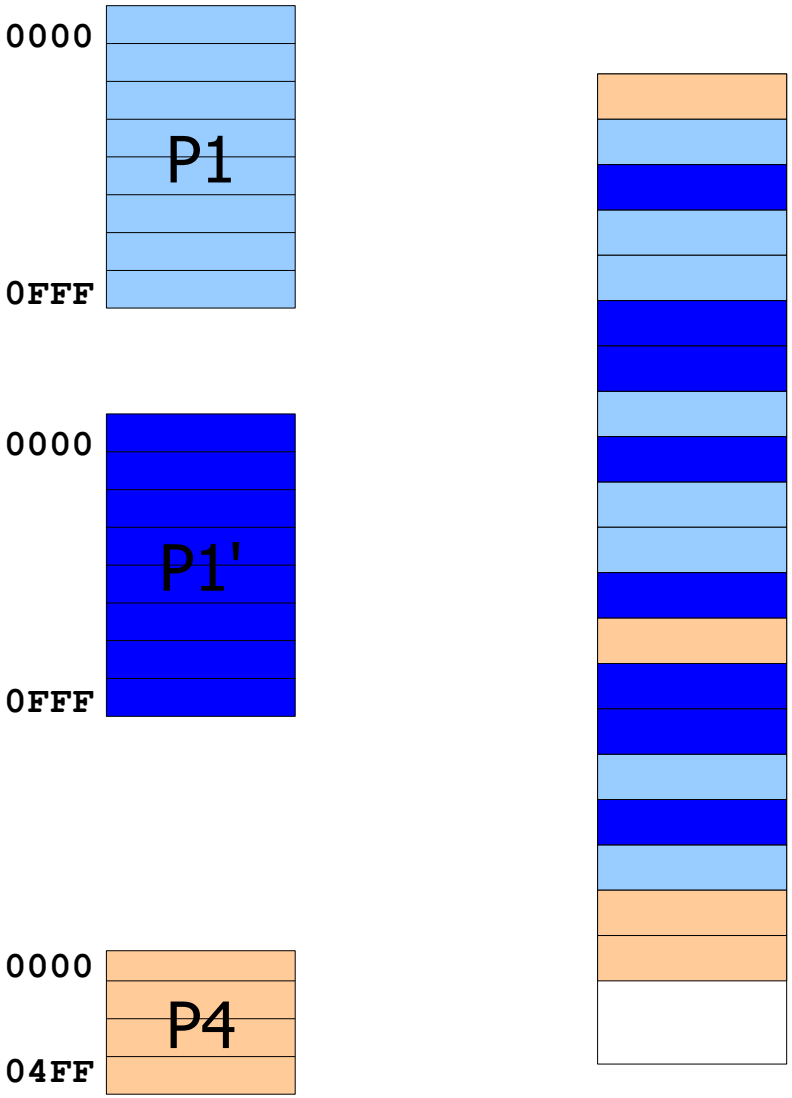
- Liberiamo 7 "blocchettini"

Va via P2 ...



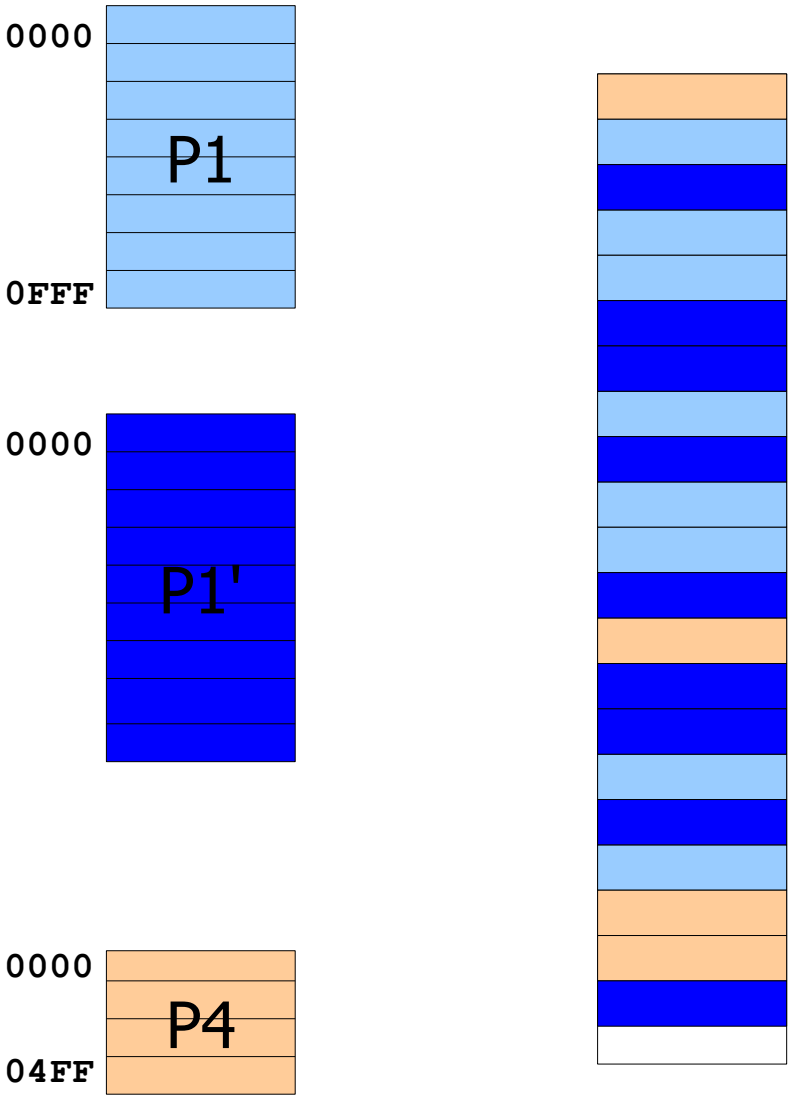
- Liberiamo altri 3 “blocchettini”

Arriva un'altra istanza di P1 ...



- Ci servono 8 "blocchettini", **ma non è necessario che siano contigui**
- **Non si è resa necessaria la compattazione**

P1' cresce ...

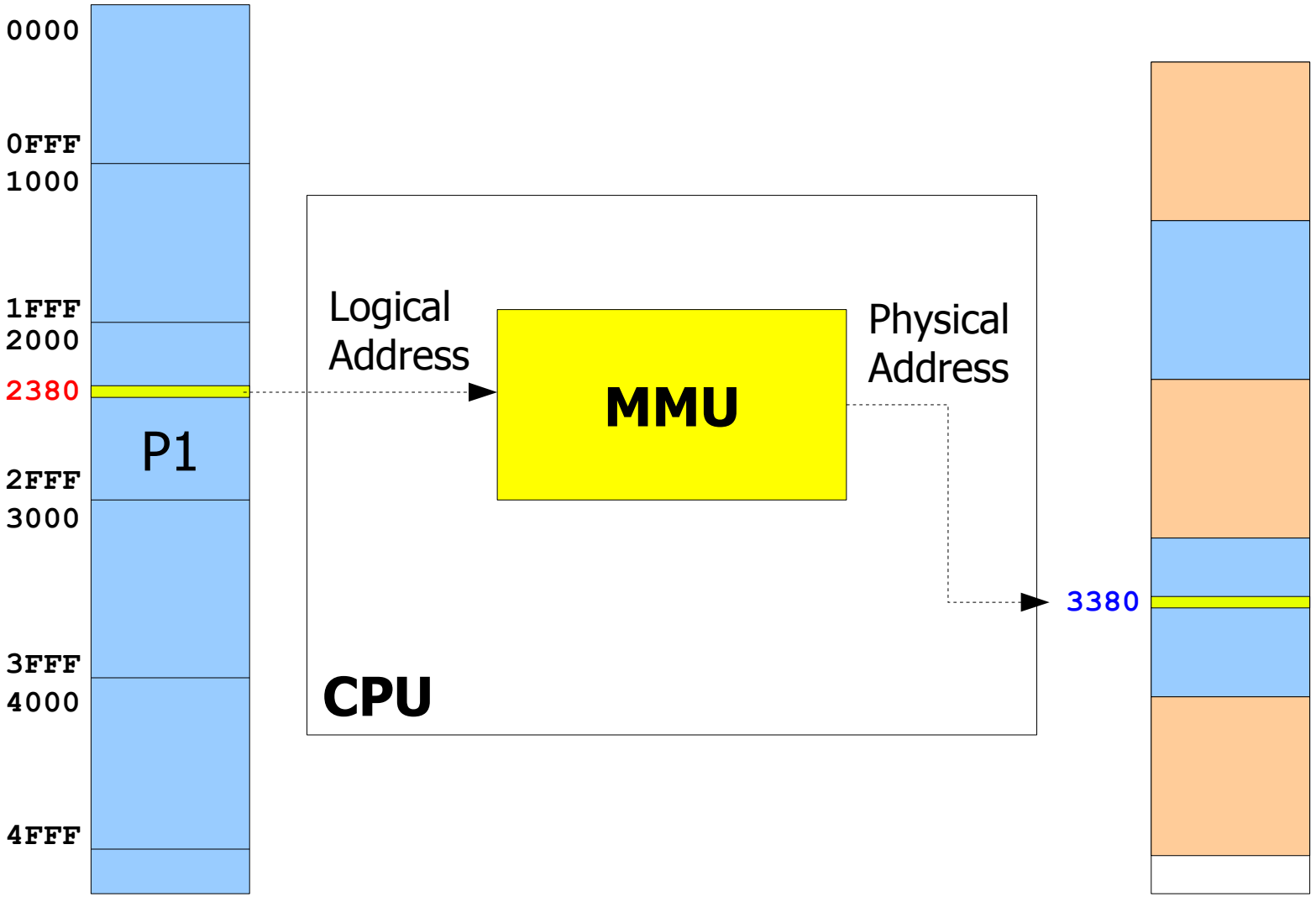


- ... e vorrebbe un altro po' di memoria
- ... gli assegnamo un altro blocchettino che prima era libero

Il paging (paginazione)

- Ogni "blocchetto" è denominato **pagina**
- La pagina ha una dimensione prefissata (una potenza del 2)
  - In genere è 4 Kbytes = 4096 bytes = 2^{12}
- La CPU, tramite la MMU (**Memory Management Unit**) che conosce il mapping delle pagine, traduce gli indirizzi logici in indirizzi fisici

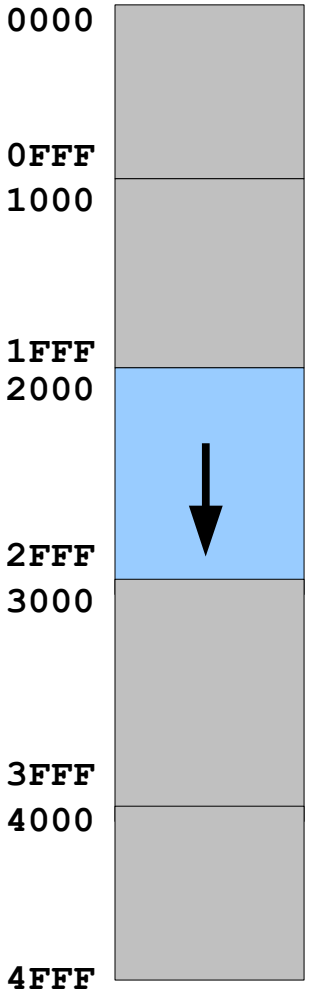
Memory Management Unit



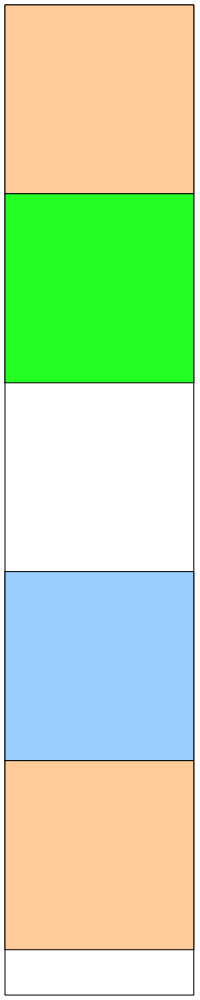
Demand Page

- Adesso abbiamo il controllo dello spazio di indirizzi a livello di pagina
- Tuttavia quando un processo gira esegue le istruzioni contenute in una pagina specifica ...
- ... con l'andare avanti dell'esecuzione, si passa ad eseguire il codice della pagina successiva, etc. etc.
- IN DEFINITIVA, AL PROCESSO SERVE UNA PAGINA PER VOLTA
- ALLORA SE ABBIAMO NECESSITA' DI MEMORIA, E NON C'E' PIU' SPAZIO, POSSIAMO MOMENTANEAMENTE **TOGLIERE AD UN PROCESSO LE PAGINE CHE NON STA USANDO**

Demand Page

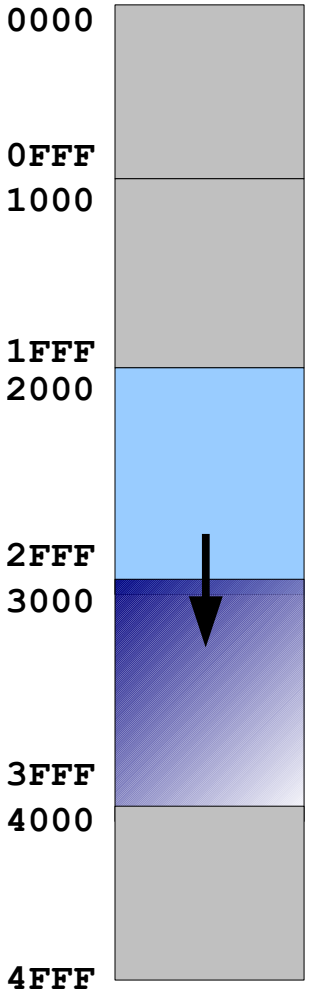


Il processo sta eseguendo il codice presente nella pagina (logica) 2 che è mappata nella pagina (fisica) 3



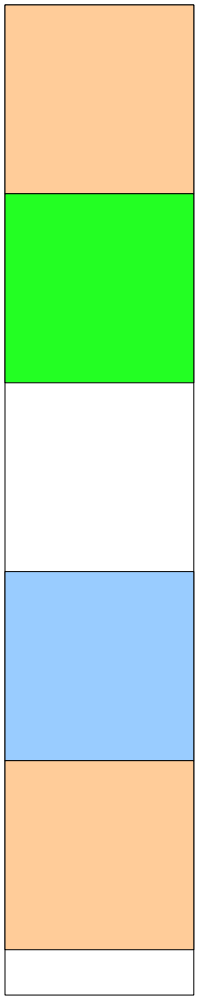
Le altre pagine del processo NON SONO PRESENTI IN MEMORIA FISICA

Demand Page

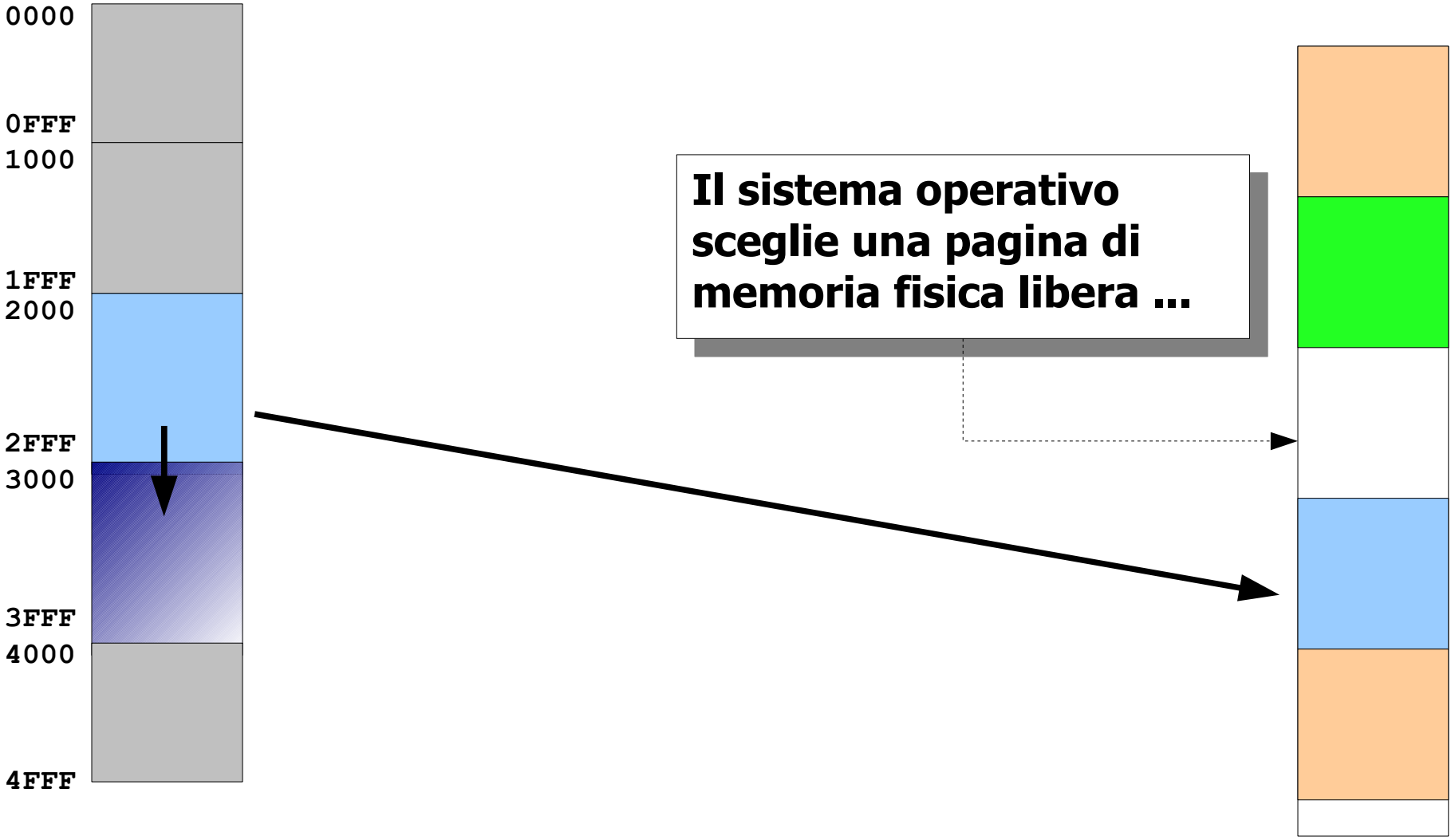


Il processo dovrebbe proseguire con il codice presente nella pagina (logica) 3 che NON E' PRESENTE IN MEMORIA

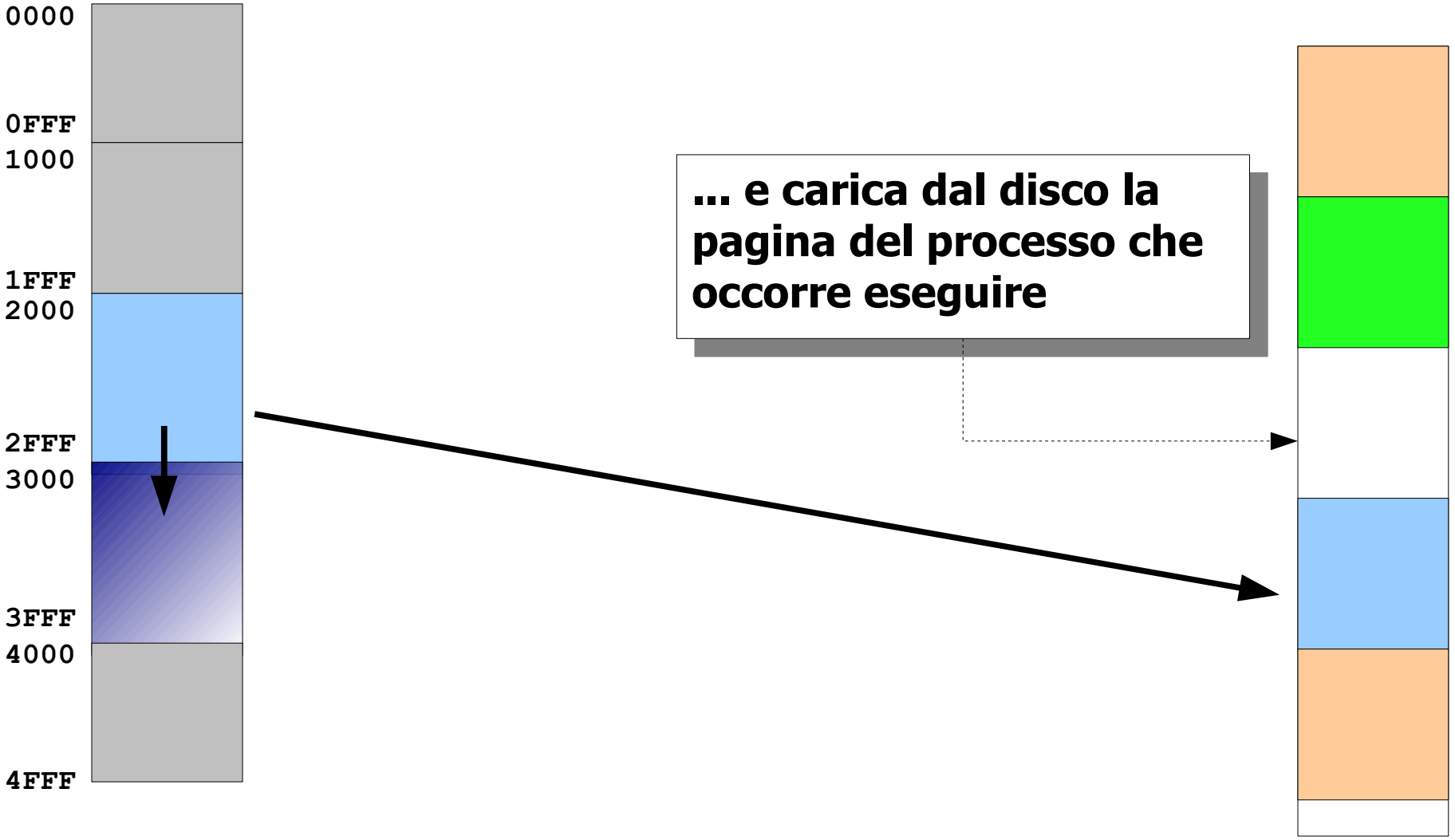
Avviene un evento denominato PAGE FAULT che attiva l'esecuzione del sistema operativo



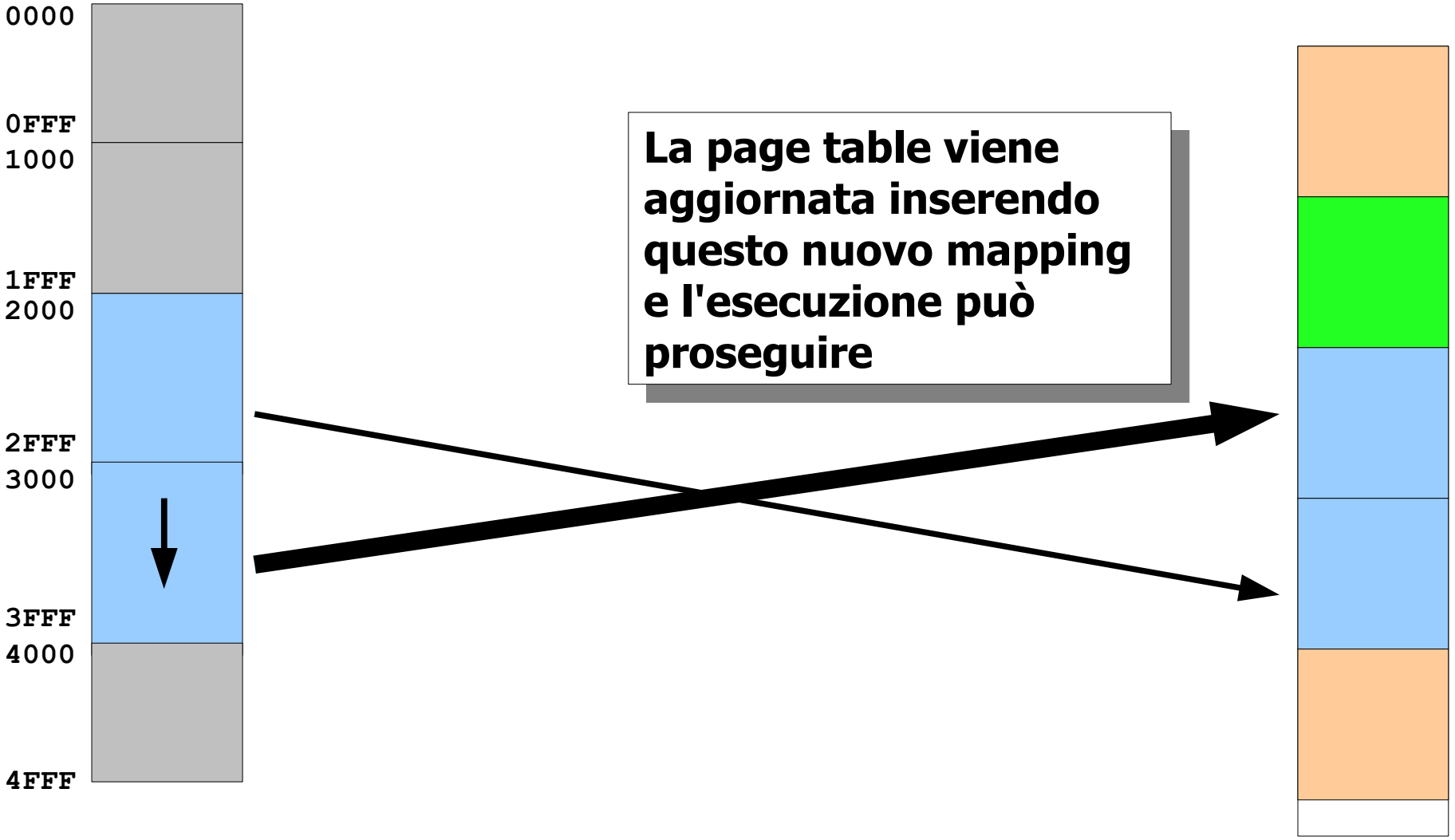
Demand Page


0000
0FFF
1000
1FFF
2000
2FFF
3000
3FFF
4000
4FFF
Il sistema operativo sceglie una pagina di memoria fisica libera ...

Demand Page


0000
0FFF
1000
1FFF
2000
2FFF
3000
3FFF
4000
4FFF
... e carica dal disco la pagina del processo che occorre eseguire

Demand Page


0000
0FFF
1000
1FFF
2000
2FFF
3000
3FFF
4000
4FFF
La page table viene aggiornata inserendo questo nuovo mapping e l'esecuzione può proseguire

Infine ...

- Quando occorre caricare una pagina da disco ...
- Se ci sono pagine fisiche libere, allora nessun problema
- Ma se tutta la memoria è occupata? Occorre "scaricare" una pagina
- Si sceglie una pagina occupata
  - **la si "toglie" al processo proprietario**
  - **la si assegna al nuovo processo**
- Quali pagine sono candidate?
  - **Ovviamente quelle dei processi in WAIT**
  - E poi le altre ...

Conclusioni

- Il paging è la soluzione ottimale per la gestione della memoria
- Si basa sul suddividere la **memoria logica e fisica** in **pagine** di dimensione fissata
- Un componente della CPU, la **MMU (Memory Management Unit)** traduce l'indirizzo virtuale in indirizzo fisico
- Grazie alla MMU, non tutte le pagine devono essere caricate in memoria, ma solo quelle utili al momento
- Se serve memoria, una pagina non usata da tempo può essere "scaricata" su disco per liberare memoria