Programmazione Avanzata e parallela

Lezione 11

Luca Manzoni

Memoria di massa

E costo delle operazioni su di essa

- Mappatura di file in memoria con mmap
  - Cosa significa mappare un file in memoria
  - Come si effettua il mapping con mmap
  - Perché vorremmo usare questo metodo?

MMAP

E mappatura di file in memoria

- Sarebbe comodo accedere al contenuto dei file direttamente come se fossero presenti in memoria:
  - Accesso come se si accedesse a un vettore
  - Ogni modifica fatta viene poi salvata sul file
  - Ma caricando in memoria solo le parti del file a cui accediamo
- Per fare questo si utilizza il fatto che abbiamo una memoria virtuale

MMAP

E mappatura di file in memoria

- Lo spazio degli indirizzi è solitamente molto più grande della memoria fisica (e.g., almeno 48 bit di indirizzi su alcune architetture moderne)

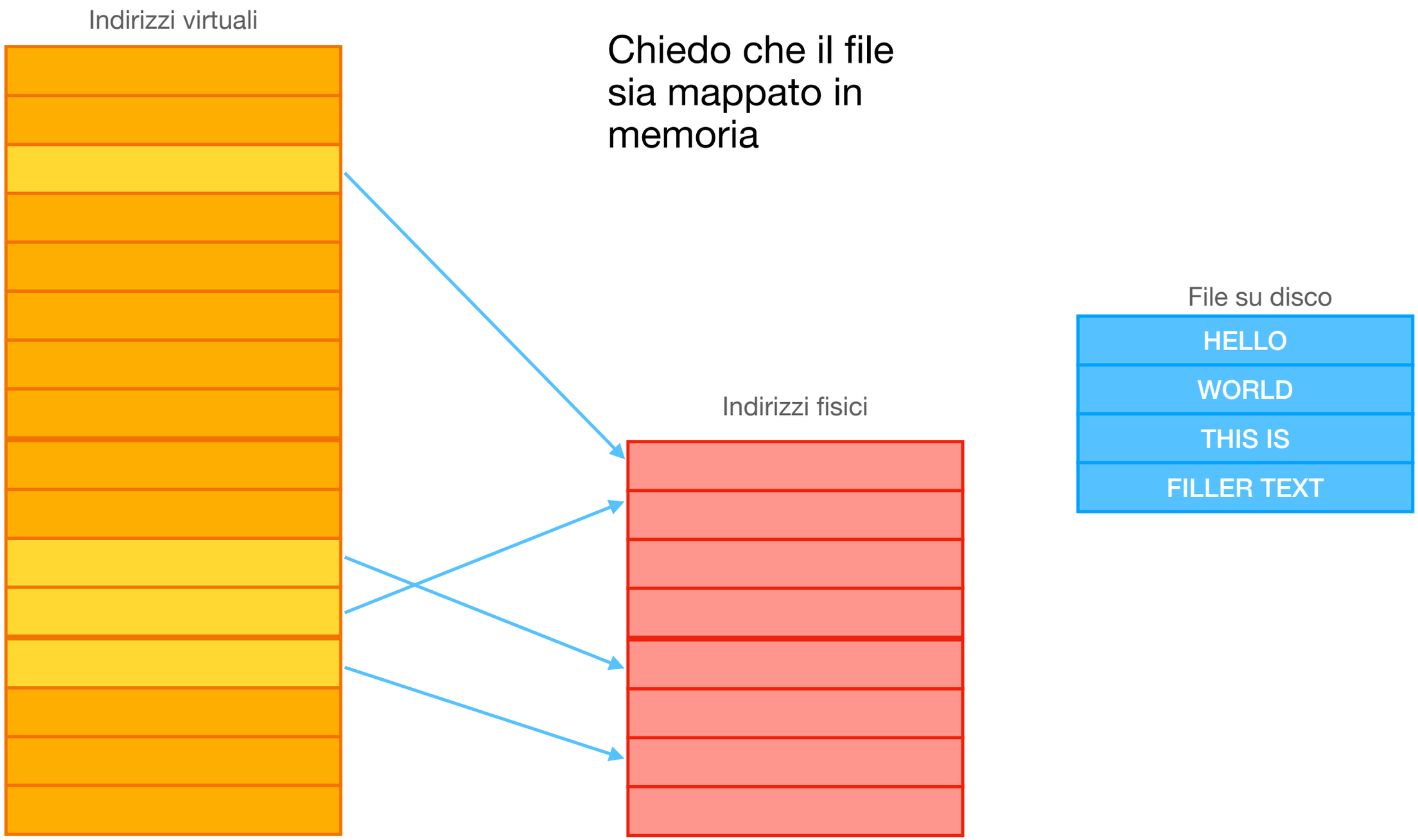
- Possiamo “fare finta” che l’intero contenuto del file sia disponibile in memoria ad alcuni indirizzi virtuali…

- …ma caricarne il contenuto solo quanto serve

- Questo può essere fatto in modo trasparente dalla gestione della memoria del sistema operativo

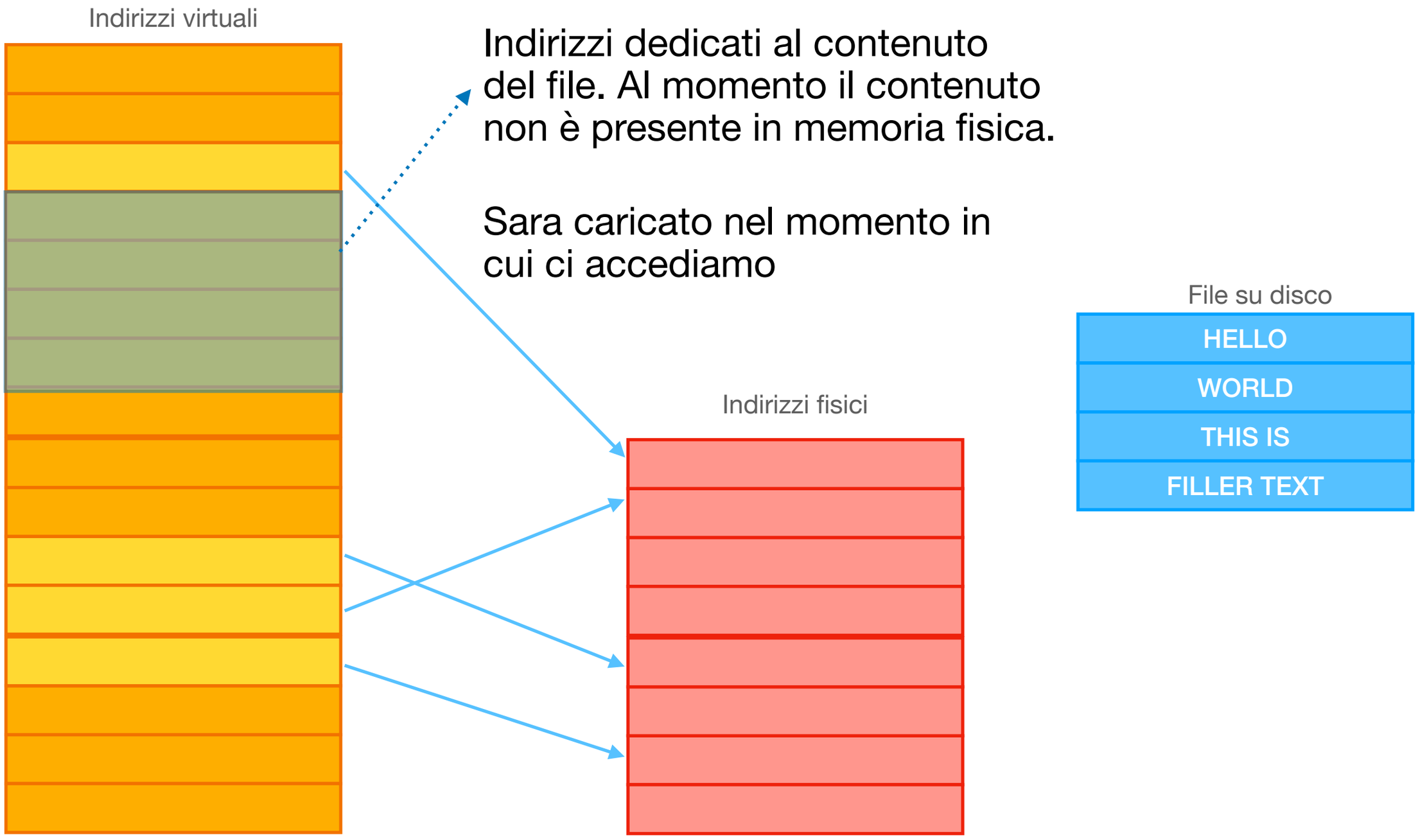
Mapping in memoria

Come funziona



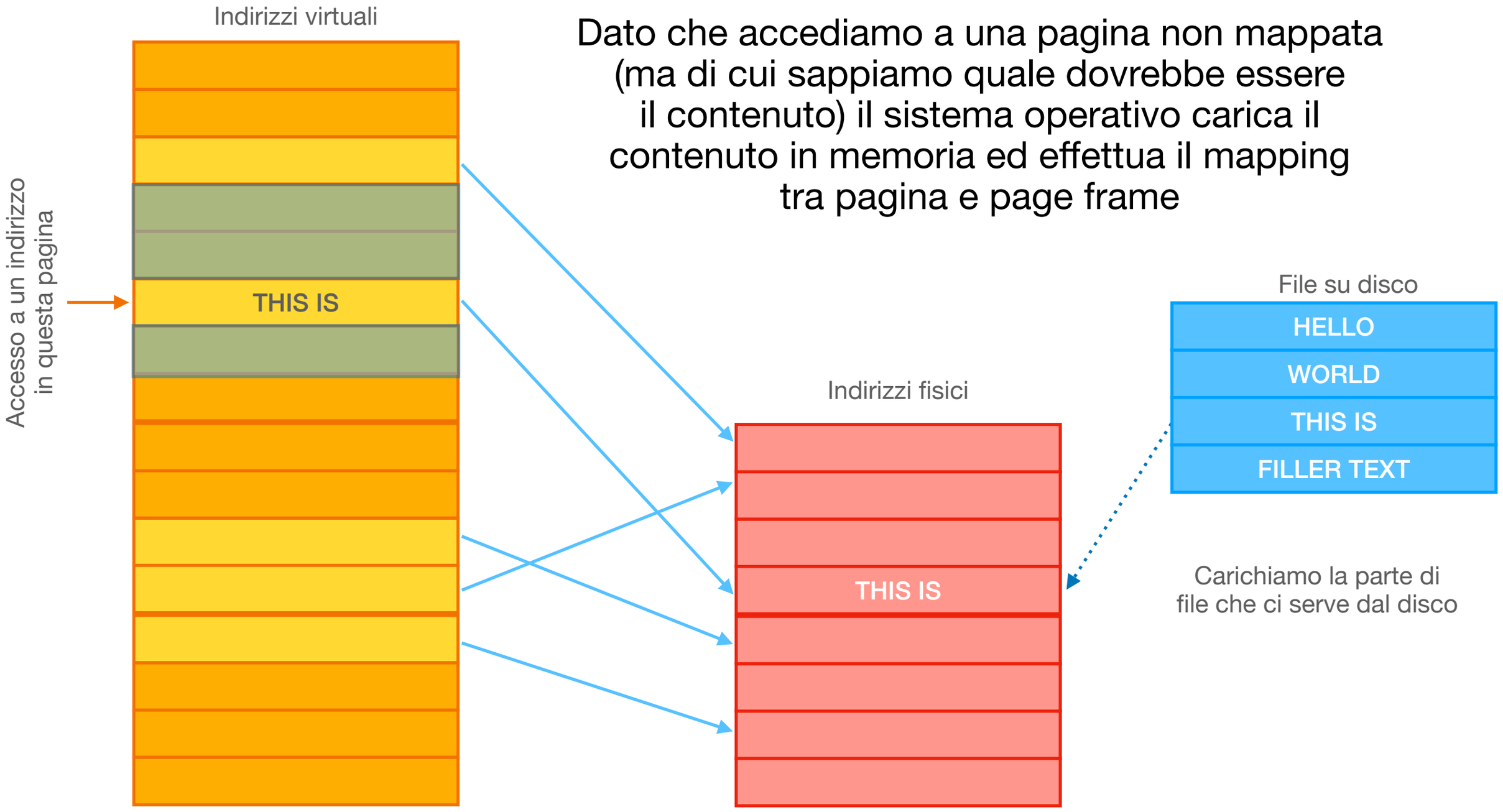
Mapping in memoria

Come funziona



Mapping in memoria

Come funziona

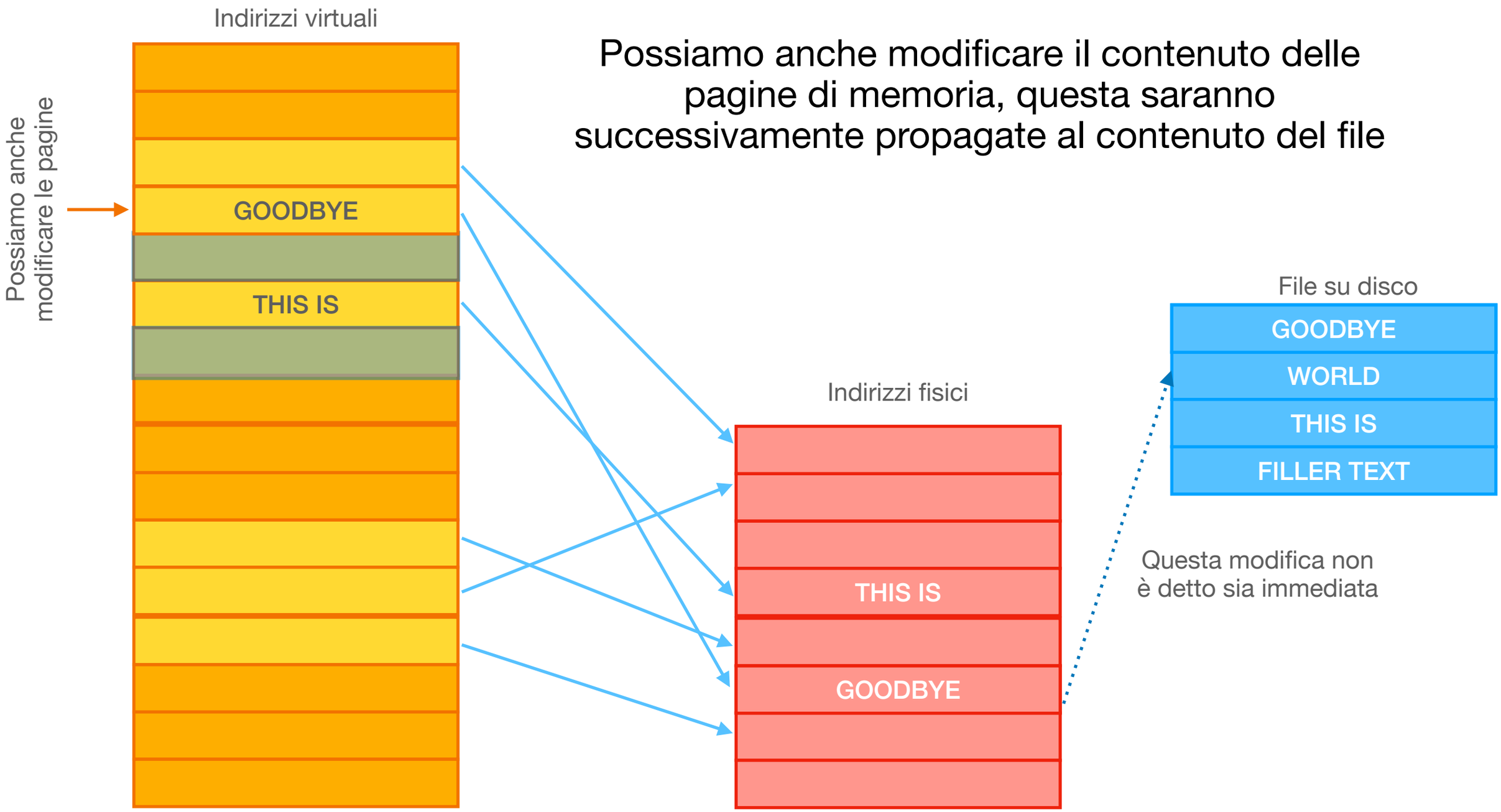


Dato che accediamo a una pagina non mappata (ma di cui sappiamo quale dovrebbe essere il contenuto) il sistema operativo carica il contenuto in memoria ed effettua il mapping tra pagina e page frame

Carichiamo la parte di file che ci serve dal disco

Mapping in memoria

Come funziona

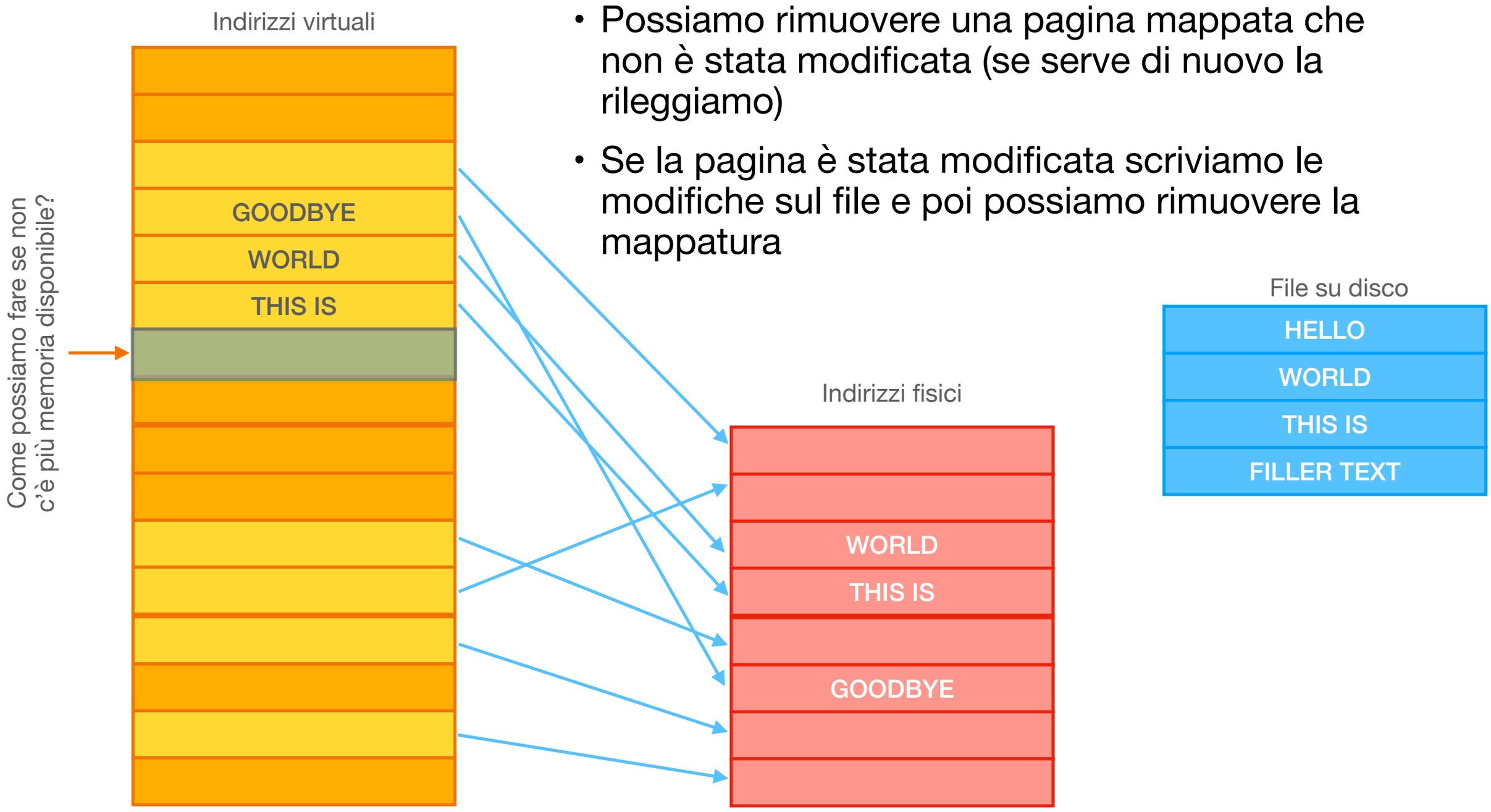


Mapping in memoria

Come funziona

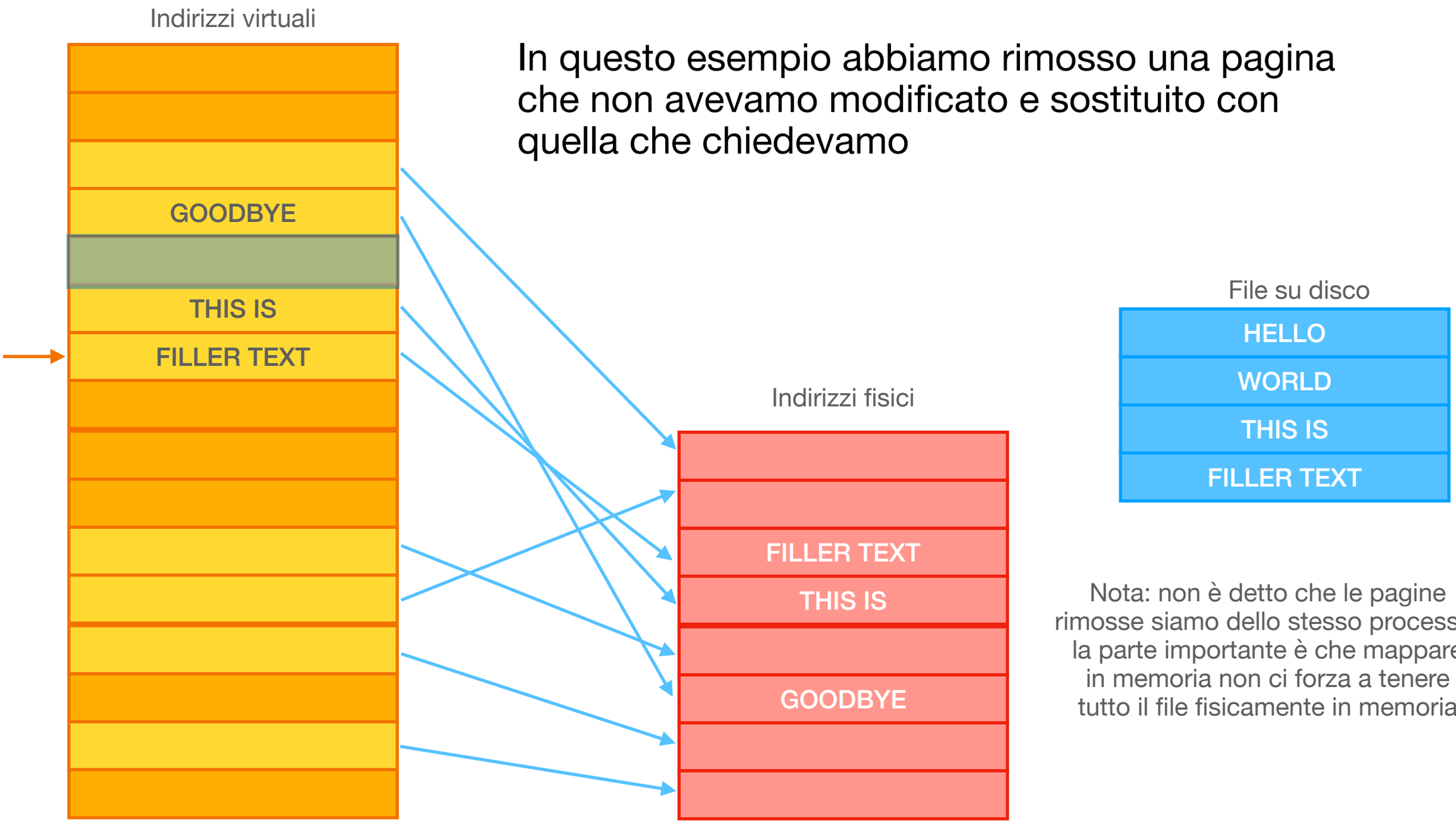
Due possibilità:

- Possiamo rimuovere una pagina mappata che non è stata modificata (se serve di nuovo la rileggiamo)
- Se la pagina è stata modificata scriviamo le modifiche sul file e poi possiamo rimuovere la mappatura



Mapping in memoria

Come funziona



In questo esempio abbiamo rimosso una pagina che non avevamo modificato e sostituito con quella che chiedevamo

Nota: non è detto che le pagine rimosse siamo dello stesso processo, la parte importante è che mappare in memoria non ci forza a tenere tutto il file fisicamente in memoria

MMAP

E mappatura di file in memoria

- La funzione che ci permette di mappare il contenuto di un file in memoria è data dallo standard POSIX:
  - **mmap**. Disponibile con un #include <sys/mman.h>
  - Dobbiamo dire che file mappare in memoria e ci viene ritornato l’indirizzo nel quale troveremo il contenuto
  - Modificando il contenuto della memoria andremo anche a modificare il contenuto del file

MMAP

Mappare un file

Puntatore all'area di memoria in cui è mappato il file

Indirizzo a cui mappare, usando *(void *)0* si fa decidere al sistema operativo

Dimensione del mapping (sarà poi approssimato in modo da essere un multiplo della dimensione delle pagine)

```
void * mmap(void * addr,
            size_t len,
            int prot,
            int flags,
            int fd,
            off_t offset)
```

Proprietà della memoria mappata. Tra queste:

- **MAP_PRIVATE**.
  Le modifiche non sono visibili all'esterno
- **MAP_SHARED**.
  Modifiche visibili (questo per modificare i file)

Permessi di accesso alla memoria, posso essere messi in OR:

- **PROT_NONE** nessun accesso
- **PROT_READ** accesso in lettura
- **PROT_WRITE** accesso in scrittura
- **PROT_EXEC** quando è possibile eseguire il contenuto della memoria

File descriptor da utilizzare (i.e., che file mappare in memoria)

Offset del file da cui iniziare la mappatura. Deve essere multiplo della dimensione delle pagine

MMAP

Sequenza di passi per mappare

- Aprire il file che vogliamo passare in memoria con fopen
- Ottenerne il file descriptor (un *int*) con **fileno(FILE *)**
- (*Facoltativo*) ottenerne la dimensione del file chiamando **stat(char *, struct stat *)** con argomenti il path e un puntatore a una struttura di tipo *struct stat* che conterrà una serie di informazioni sul file, tra cui anche la dimensione nel campo *st_size*
- Chiamare **mmap** per mappare il file in memoria. Ritorna MAP_FAILED se non è stato possibile effettuare il mapping

MMAP

Sequenza di passi per rimuovere il mapping

- La chiusura di un file non rimuove i mapping che lo coinvolgono, quindi anche la chiusura è in più passi:
  - Chiamare **munmap(void * addr, size_t size)** con l'indirizzo restituito da mmap e la dimensione passata quando si è chiamato mmap
  - Chiudere il file con fclose.
  - Quando viene effettuato munmap le modifiche saranno scritte non necessariamente immediatamente, ma è possibile forzare la scrittura del file con msync

MMAP

Funzioni utili

- Per cambiare la dimensione di un file (e.g., crediamo un file nuovo e vogliamo incrementarne la dimensione a 10MB) possiamo usare **ftruncate(int fd, off_t length)** per cambiare la dimensione del file alla lunghezza indicata. La funzione è in **unistd.h**

- **madvise** può essere utilizzato per indicare come intendiamo accedere a un range di indirizzi (e.g., sequenziale o casuale) o se certi indirizzi non saranno più usati (o necessiteremo di usarli a breve)

MMAP

Vantaggi

- Possiamo accedere al file come se fosse della normale memoria (niente necessità di avere un punto preciso dove fare letture e scritture)

- Invece di utilizzare “read”, “write” e “lseek” come chiamate possiamo affidarci al sistema di gestione della memoria (potenzialmente più efficiente)

- Facilità di accesso quando si hanno più processi o thread che devono lavorare sullo stesso file (attenzione che servono dei lock)

MMAP

Svantaggi

- Per alcune tipologie di accesso (e.g., sequenziale) potrebbe non essere necessariamente più efficiente

- Mappare i file in memoria è più macchinoso

- mmap esiste per i sistemi POSIX, non per ogni sistema operativo

- Non possiamo accedere a file che abbiano dimensioni superiori allo spazio degli indirizzi (che **non** è la memoria fisica). Più che altro un problema su sistemi a 32 bit

- Affidiamo il caching al sistema di gestione della memoria (magari aiutato con madvise) ma questo potrebbe non essere ottimale