Lezione 14

La memoria tampone per i dischi

Sistemi operativi

19 giugno 2012

Marco Cesati

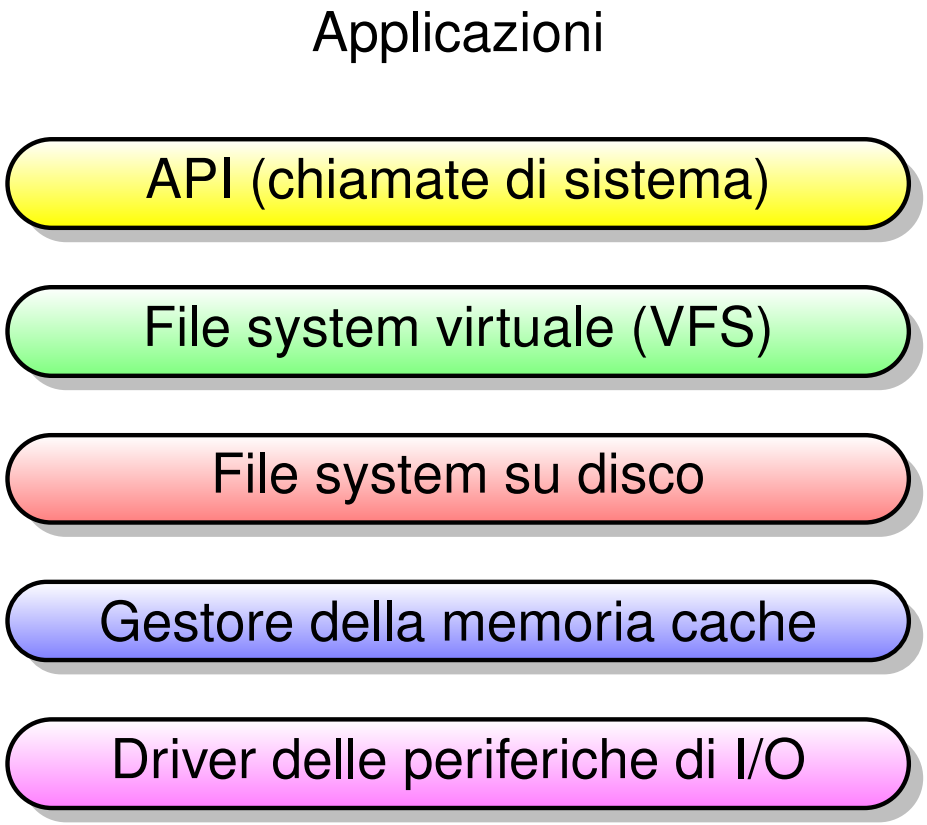
System Programming Research Group
Università degli Studi di Roma Tor Vergata

Di cosa parliamo in questa lezione?

Ottimizzazione degli accessi alla memoria fisica

1. Efficienza e prestazioni della memoria secondaria
2. Memorie tampone per la memoria secondaria
3. Ottimizzazione degli accessi ai file
4. Recupero della memoria fisica

Componenti del SO per la gestione dei file system



Efficienza e prestazioni della memoria secondaria

Le prestazioni del sistema dipendono in modo essenziale dalla gestione della memoria secondaria

Il disco magnetico è uno dei dispositivi hardware più lenti:

- Tasso di trasferimento a regime:
  - Controller SATA: $\sim$300 MB/s
  - Front Side Bus: 12 800 MB/s (Core 2 Extreme)
- Tempi di accesso (latenza):
  - Disco rigido: 16 ms
  - RAM: 80 ns
  - Cache L2: 5 ns
  - Cache L1: 1 ns

1 ns : 16 ms = 1 secondo : 6 mesi

Efficienza e prestazioni della memoria secondaria (2)

Per mitigare la lentezza della memoria secondaria si può:

- Utilizzare opportuni algoritmi di schedulazione del disco
- Organizzare le informazioni sul disco in modo da ridurre gli spostamenti delle testine del disco
- Utilizzare la RAM come memoria tampone (cache) per le informazioni su disco

Riduzione degli spostamenti delle testine

- Durante l'installazione di MS Windows, i file necessari per l'avvio del SO vengono registrati in settori consecutivi
  - Via via che il sistema viene utilizzato e il SO aggiornato, la frammentazione aumenta e le prestazioni si riducono
- Nei file system Unix più vecchi gli inode erano posizionati all'inizio del disco/partizione
  - Le operazioni di lettura dei metadati di un insieme di file ("`ls -l`") richiedono piccoli spostamenti delle testine
- Nei recenti file system Unix per dischi di grande capacità gli inode sono "sparpagliati" e posti vicino ai blocchi di dati dei file corrispondenti
  - Accedere al contenuto di un file richiede spostamenti delle testine più brevi
- Il contenuto dei file "piccoli" può essere memorizzato all'interno dell'inode, al posto dei puntatori ai blocchi dati
  - Ad esempio, i file di tipo soft link in Ext2/Ext3

La memoria tampone del disco

Gli accessi alla memoria secondaria tendono a seguire le stesse leggi empiriche degli accessi alla memoria centrale:

Principio di località spaziale

La probabilità di accedere ad un settore di un disco è maggiore se è stato recentemente acceduto un settore vicino

Esempio: accesso sequenziale ad un file su un file system con allocazione dello spazio sequenziale o con estensioni

Principio di località temporale

La probabilità di accedere ad un settore di un disco è maggiore se tale settore è stato già acceduto di recente

Esempio: ciclo di editing / compilazione / esecuzione di un programma

La memoria tampone del disco (2)

I principi di località permettono di migliorare le prestazioni introducendo *memoria tampone* (o *memoria cache*)

Si utilizzano celle di RAM per memorizzare temporaneamente i dati dei dispositivi di memoria secondaria

Tre tipi fondamentali di memoria tampone del disco:

- Memoria tampone di traccia (*track cache*)
- Memoria tampone di buffer (*buffer cache*)
- Memoria tampone di pagina (*page cache*)

Memoria tampone di traccia

- Memorizza il contenuto di intere traccie o cilindri del disco rigido
- Contenuta all'interno del controller del disco rigido
- Invisibile per il SO
- Dimensioni tipiche: da 8 MiB a 64 MiB
- Consente di
  - disaccoppiare la velocità di trasferimento dell'interfaccia del controller da quella delle testine
  - salvare il contenuto dei settori letti dalla testina prima di arrivare a quello effettivamente richiesto dal SO
  - abbreviare la durata delle singole operazioni di scrittura (lato SO)
  - nei dischi SATA e SCSI con *command queueing*, soddisfare una richiesta di lettura utilizzando una precedente richiesta di scrittura

La memoria tampone di buffer

La *buffer cache* è una memoria tampone per i blocchi di dati dei dischi più recentemente letti o scritti

La buffer cache è realizzata dal SO come servizio di base al quale si appoggiano i file system su disco

Per ogni richiesta di trasferimento di un blocco di un file system (superblocco, inode, o blocco dati):

- Il numero logico di blocco viene tradotto in uno o più numeri di settore del disco
- Per ciascun settore si controlla che la buffer cache non contenga già un buffer (memoria di transito) contenente i dati del settore
  - Se presente (cache hit), i dati vengono resi disponibili al componente che ha richiesto il trasferimento
  - Se non presente (cache miss), si alloca un nuovo buffer per il settore, lo si inserisce nella buffer cache e si programma il suo trasferimento

La memoria tampone di buffer (2)

I buffer contenuti nella cache possono essere in diversi stati:

- Free (inutilizzato): il buffer non è associato ad alcun settore
- Invalid (invalido): il buffer è associato ad un settore, ma i dati non sono validi (il settore sul disco è più aggiornato)
  - Le operazioni di lettura che coinvolgono un buffer invalid causano l'immediato trasferimento di dati dal disco
- Dirty (sporco): il buffer è associato ad un settore, ma i dati nel buffer sono più aggiornati di quelli sul disco
  - Le operazioni di scrittura comportano l'aggiornamento dei dati nel buffer ed il suo passaggio allo stato dirty
- Clean (pulito): il buffer è associato ad un settore, ed i dati in memoria e sul disco coincidono

La buffer cache permette di disaccoppiare temporalmente le operazioni di scrittura del file system e la programmazione dei trasferimenti verso il disco

La memoria tampone di pagina

Nei SO con memoria virtuale, alcuni accessi alla memoria secondaria coinvolgono intere pagine di dati:

- L'esecuzione di un programma prevede la mappatura del file eseguibile e delle librerie dinamiche collegate nello spazio di indirizzi virtuali del processo
- I programmi possono richiedere esplicitamente di mappare un file in memoria (es., chiamata di sistema Unix `mmap()`)

La memoria tampone di pagina (o page cache) consente di memorizzare temporaneamente i dati dei file acceduti una pagina alla volta

Ad esempio, alla prima lettura di una pagina corrispondente ad un file mappato in memoria si verifica una eccezione page fault, ed il relativo gestore:

- Alloca una nuova page frame per il processo e la inserisce nelle sue tabelle di paginazione
- Inserisce la page frame nella page cache di sistema
- Programma il trasferimento dei dati dal disco e sospende il processo fino a trasferimento avvenuto

La memoria tampone di pagina (2)

Qual è il vantaggio dato dalla page cache?

- La page frame contenente i dati letti dal disco è inserita nelle tabelle di paginazione di tutti i processi che mappano in modo condiviso il file
- Quando un nuovo processo richiede una mappatura dello stesso file, la page cache consente di determinare rapidamente se la page frame è stata già allocata ed inizializzata con i dati del disco
- Anche quando tutti i processi hanno rilasciato la page frame, questa può rimanere nella page cache, in modo che futuri accessi agli stessi dati possano evitare il trasferimento dal disco
- Le operazioni di scrittura (modifica delle pagine) sono rapide in quanto non è necessario programmare immediatamente il trasferimento verso il disco

Integrazione di buffer cache e page cache

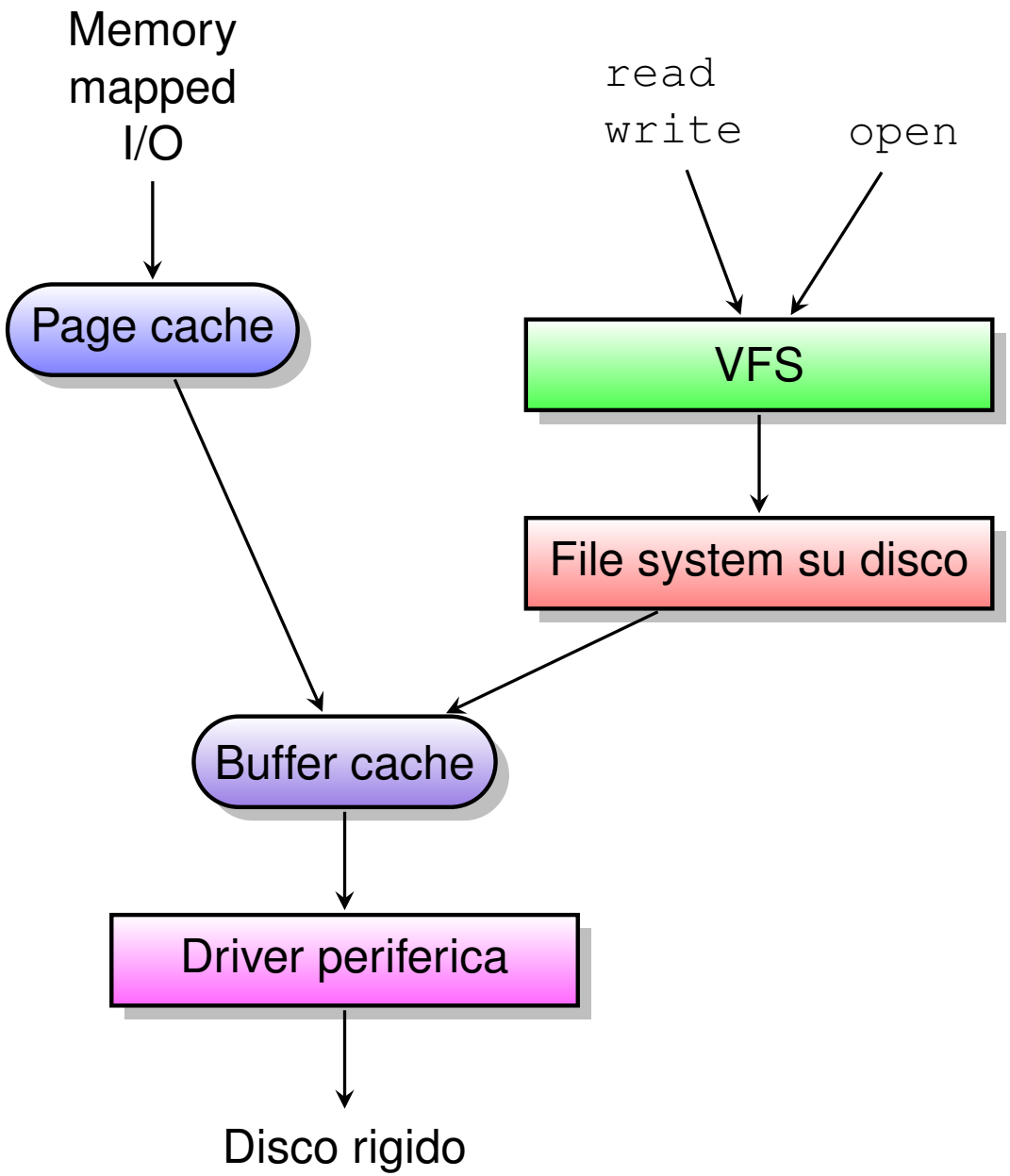
Alcuni SO contengono sia la buffer cache che la page cache

- Ad esempio: MS Windows XP, vecchie versioni di Linux ($\leq$ 2.2), Sun Solaris
- In pratica, la page cache è un componente realizzato “al di sopra” della buffer cache
- Accessi ai file tramite mappatura in memoria fanno uso della page cache
- Tutti gli altri accessi (letture e scritture dei file tramite `read()` e `write()`, accesso a superblocchi ed inodi, ...) utilizzano direttamente la buffer cache

Quali sono gli svantaggi di questo approccio?

- È necessario mantenere la coerenza dei dati nelle due cache
- L'overhead della page cache si aggiunge a quello della buffer cache

Integrazione di buffer cache e page cache (2)



Page cache unificata

In alcuni SO la buffer cache viene eliminata

- Esempio: versioni recenti di Linux ($\geq$2.4.10)
- Tutte le operazioni di accesso al contenuto dei file regolari avvengono tramite la page cache
- Nel caso di operazioni di accesso ad un singolo blocco di un file tramite `read()` e `write()`, il SO trasferisce l'intera pagina di dati contenente il blocco

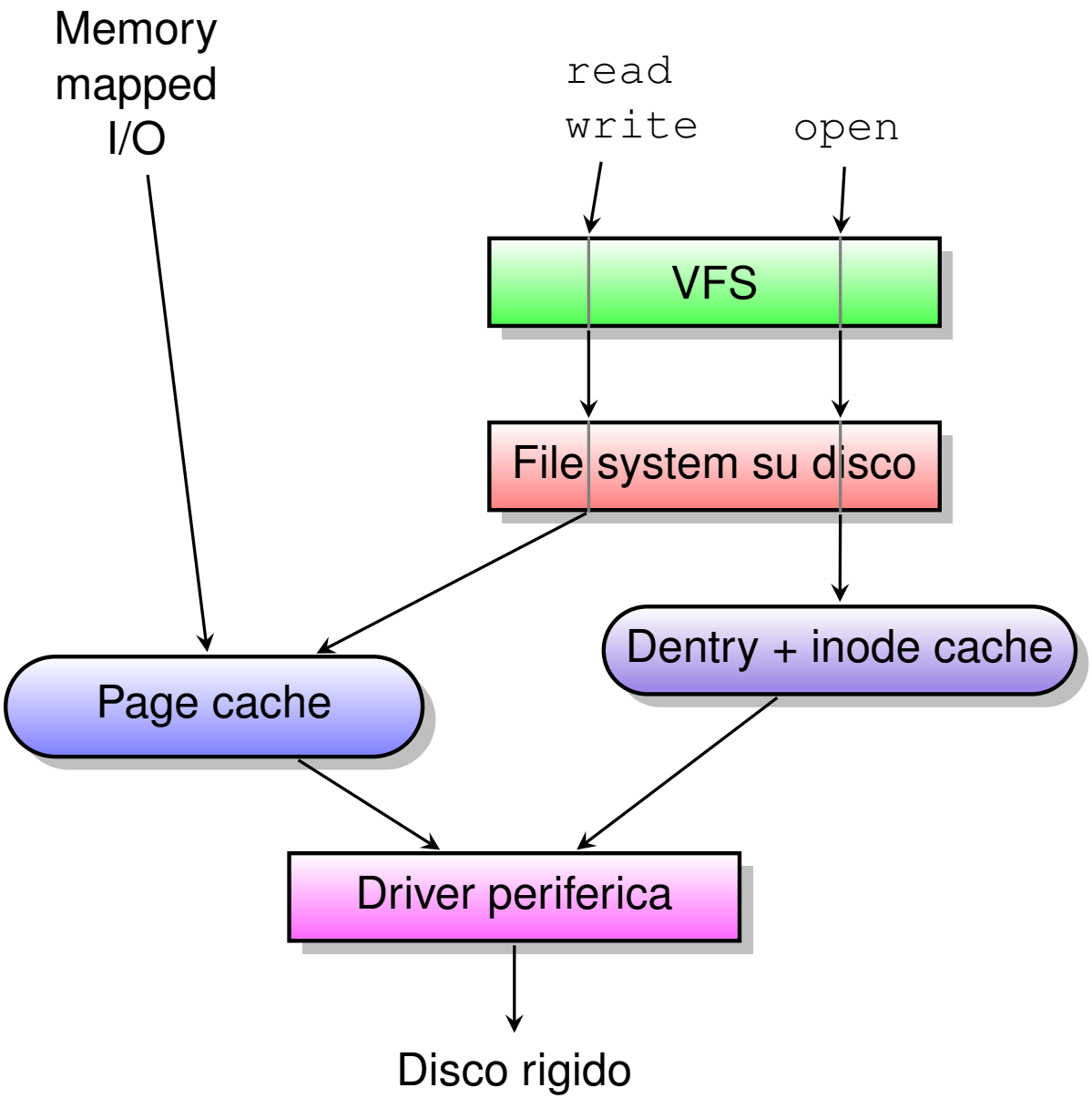
Qual è il problema di questo approccio?

- I blocchi entro la stessa pagina di dati non sono necessariamente contigui sul disco!
- Quindi è necessario usare una struttura di dati che colleghi ciascuna pagina nella page cache con i corrispondenti blocchi del file system

In pratica i SO utilizzano anche altre memorie cache specializzate, ad es. in Linux:

- dentry cache: i percorsi dei file recentemente risolti
- inode cache: info sugli inode acceduti recentemente

Page cache unificata (2)



Scritture sincrone e asincrone

Scrittura sincrona

Operazione di trasferimento programmata e avviata quando viene richiesta la scrittura

Scrittura asincrona

Operazione di trasferimento programmata nel momento che il SO ritiene più conveniente per ottimizzare le prestazioni

- In entrambi i casi, la procedura di richiesta della scrittura si può concludere prima che il trasferimento sia terminato
  - la differenza sta in quando avviare il trasferimento
- Le scritture asincrone sono facili da realizzare in presenza di memorie tampone
  - Teoricamente le scritture possono essere differite fino a quando si deve rilasciare un buffer o una pagina dirty
- Per limitare gli effetti di un crash o di un blackout, il SO scandisce ad intervalli di tempo regolari le cache ed avvia i trasferimenti in scrittura degli oggetti dirty

Ottimizzazione degli accessi sequenziali

Molti accessi ai file sono di tipo sequenziale, ad esempio

- Copia di un file
  - Verso un altro file
  - Su di un socket di rete
- Calcolo di una somma di controllo
- Integrazione del file in un archivio (tar, zip, . . .)
- Lettura di un file di configurazione

La presenza di memoria tampone consente al SO di utilizzare due tecniche per ottimizzare gli accessi sequenziali:

- La lettura anticipata
- Il rilascio indietro

Gli accessi in scrittura possono in genere essere ricondotti a quelli in lettura, in quanto scritture parziali di un blocco o pagina richiede la preventiva lettura dei dati dal disco

La lettura anticipata

La tecnica della *lettura anticipata* (*read ahead*) consente al SO di avviare i trasferimenti dei dati di un file in anticipo rispetto alle richieste effettivamente avute dai processi

Ad esempio:

- Il SO riceve una richiesta per leggere il blocco 0 di un file
- Successivamente riceve una richiesta per leggere il blocco 1 dello stesso file
- Decide quindi di avviare il trasferimento del blocco 2 prima della effettiva richiesta
- Quando riceve la richiesta per il blocco 2, avvia il trasferimento dei blocchi 3 e 4
- Quando riceve le richieste per i blocchi 3 e 4, avvia i trasferimenti per i blocchi da 5 a 8

Quali sono vantaggi e svantaggi della lettura anticipata?

Vantaggio: le richieste di lettura sono esaudite rapidamente poiché i dati richiesti sono già nella memoria tampone

Svantaggio: se il processo non sta realmente leggendo il file in sequenza, si spreca RAM e banda passante del bus/disco

La lettura anticipata (2)

- Determinare se un processo accede ad un file in modo sequenziale è difficile
  - Il comportamento passato del processo non può fornire indicazioni certe su quello futuro
- Gli algoritmi per la lettura anticipata sono basati su regole euristiche
  - Funzionano bene in generale, ma possono funzionare male in casi particolari
  - In genere l'amministratore di sistema può modificare i parametri di funzionamento dell'algoritmo
- Nei sistemi POSIX: `posix_fadvise()` (descrittori di file) e `[posix_]madvise()` (file mappati in memoria) sono chiamate di sistema che indicano il tipo di accesso:
  - NORMAL: nessuna indicazione
  - SEQUENTIAL: accesso sequenziale
  - RANDOM: accesso non sequenziale
  - WILLNEED: sarà acceduto a breve
  - DONTNEED: non più acceduto a breve
  - NOREUSE: acceduto una sola volta

Il rilascio indietro

La tecnica del rilascio indietro è speculare alla lettura anticipata

Se il SO ritiene che un file sia acceduto in modo sequenziale, ad ogni nuova richiesta rilascia il blocco (o la pagina) richiesta precedentemente

I rischi del rilascio indietro sono maggiori di quelli della lettura anticipata

- Se viene rilasciato il blocco sbagliato, il processo lo richiederà a breve ma dovrà essere ritrasferito dal disco
- Generalmente è più sicuro utilizzare la tecnica del rilascio indietro con le scritture in sequenza piuttosto che le letture in sequenza
  - Sembra meno probabile che un processo torni ad accedere ad un pagina modificata rispetto ad una letta
  - In questo caso il rilascio indietro può essere abbinato alla scrittura differita: il blocco viene rilasciato dalla memoria cache dopo essere stato trasferito sul disco

Utilizzo della memoria fisica

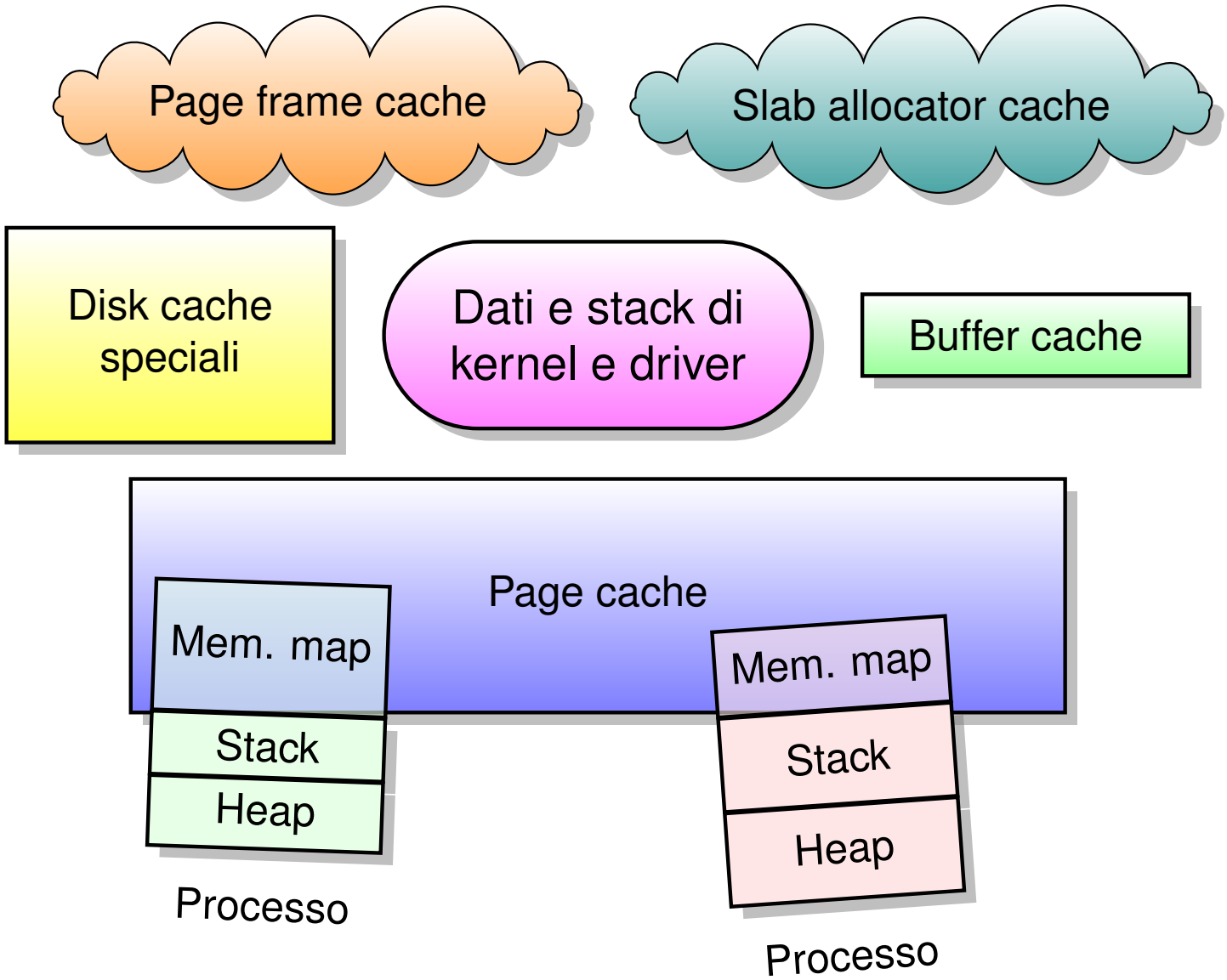
Il kernel di un SO “spende” le pagine di memoria fisica per diversi scopi:

- Stack e strutture di dati per il kernel ed i driver
- Memoria cache per le pagine di dati dei dischi
- Memoria cache per i buffer dei dischi
- Memorie cache specializzate (dentry cache, inode cache)
- Page frame cache (allocate dal Buddy system e libere)
- Slab allocator cache (page frame utilizzate per gli slab)
- Page frame assegnate ai processi per stack e heap
- Page frame assegnate ai processi per file mappati in memoria

Il SO deve possedere un meccanismo che consenta di recuperare memoria fisica:

- Le memorie cache dei dischi tendono ad aumentare di dimensione continuamente
- I processi possono richiedere pagine a volontà e sotto-utilizzarle

Utilizzo della memoria fisica (2)



Recupero della memoria fisica

Ogni SO ha un componente chiamato PFRA (Page Frame Reclaiming Algorithm) per liberare page frame in uso

L'esecuzione del PFRA può essere attivata in caso di esaurimento della memoria fisica, oppure "preventivamente" ad intervalli di tempo regolari

Per il PFRA le page frame sono suddivise a seconda del loro contenuto, ad esempio

Non recuperabili	Pagine libere Pagine utilizzate dal kernel (stack, dati, . . .) Pagine protette (trasferimento, lock)
Anonime	Pagine di stack dei processi Pagine di heap dei processi
Mappate	Pagine mappate su file dei processi Pagine nella page cache o buffer cache Pagine in disk cache specializzate
Scartabili	Pagine nella page frame cache Pagine libere nello slab allocator cache

Recupero della memoria fisica (2)

Il PFRA adotta un insieme di regole euristiche per valutare quali page frame recuperare, ad esempio

- Recupera per prima cosa le pagine scartabili, ossia quelle che non contengono dati utili
- Recupera poi le pagine mappate il cui contenuto è già aggiornato sul disco (clean)
- Poi recupera le altre pagine mappate (poiché sono dirty è necessario avviare i trasferimenti verso il disco)
- Infine recupera le pagine anonime dei processi (salvando il loro contenuto nell'area di swap)
- Usa strategie che non rimuovano le page frame accedute più frequentemente (LRU, con bit di accesso, ecc.)

Per recuperare una pagina si deve togliere la page frame dalle tabelle di paginazione di tutti i processi che la condividono

- Necessaria una struttura di dati per il reverse mapping
  - permette di determinare velocemente tutte le voci di tabelle di paginazione che usano una certa page frame