

Sistemi Operativi

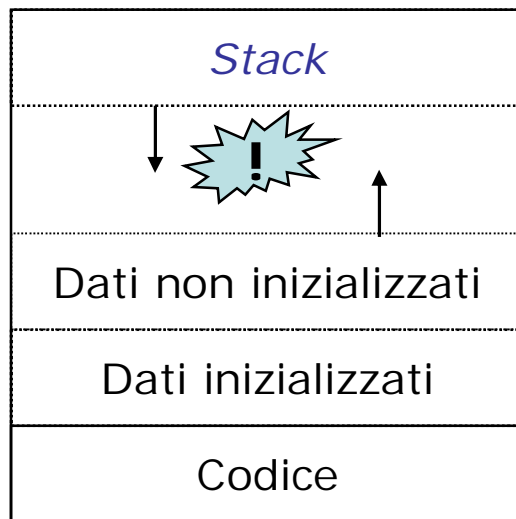
Da Unix a GNU/Linux (parte 2)

Docente: Claudio E. Palazzi
cpalazzi@math.unipd.it

Crediti per queste slides al Prof. Tullio Vardanega

Gestione della memoria – 1

- Massima semplicità per massima portabilità su architetture fisiche diverse
- Ogni processo possiede un proprio spazio di indirizzamento privato (**memoria virtuale**)
 - Suddiviso in 4 sezioni



Stack
Pila dei contesti, dimensione variabile, R/W

Block Storage Segment

Data segment

Text segment, dimensione fissa, R

} dimensione variabile, R/W

Gestione della memoria – 2

- Il segmento dati varia in dimensione a seconda delle attività del programma (cf. `malloc()` di C)
 - POSIX **non** definisce queste chiamate di sistema
 - Una parte del segmento dati può ospitare *file* mappati in memoria
- Il segmento *stack* contiene l'ambiente d'esecuzione corrente (*record* di attivazione) e cresce in direzione **opposta** al segmento dati
- Il segmento codice può essere condiviso tra più processi
 - Ma **non gli altri** segmenti tranne che per processi duplicati da `fork()` fintanto che non vengono modificati
 - Copy-on-write
 - contiene linguaggio macchina (da compilazione)

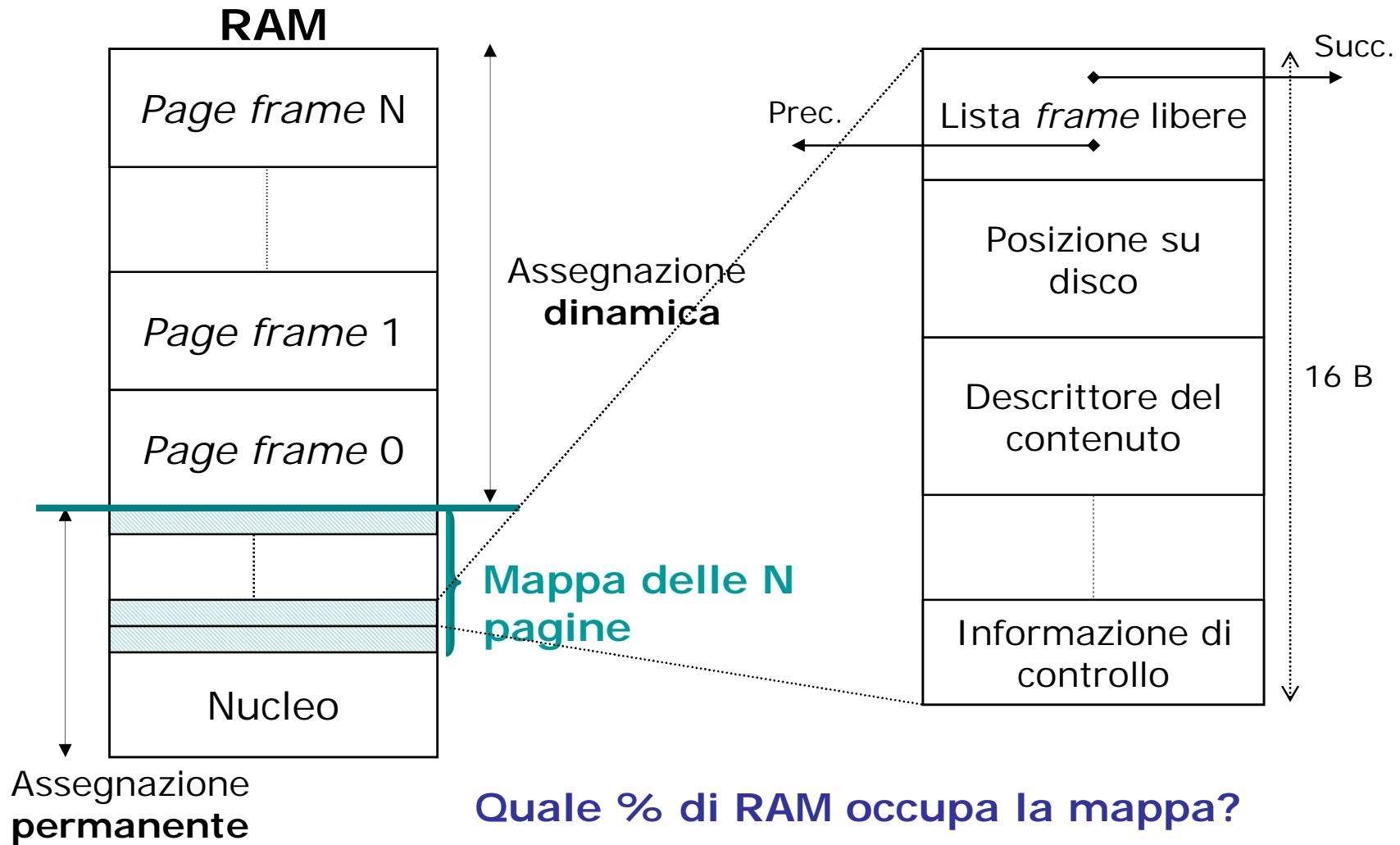
Gestione della memoria (UNIX) – 3

- In origine l'allocazione di memoria principale avveniva mediante **swap** di processi
 - Rimpiazzo di interi processi quando una particolare esecuzione rilevava mancanza di memoria
 - A seguito di **fork ()**
 - A causa di allocazione esplicita richiesta dal programma
 - Per allocazione implicita conseguente a chiamata di procedura
 - Il gestore (**swapper**) creava lo spazio necessario salvando su disco i processi sospesi con più tempo d'esecuzione recente e minor priorità

Gestione della memoria (UNIX) – 4

- In seguito, fu introdotta paginazione con modalità a richiesta (***paging on demand***)
 - Un processo è eseguibile se il suo descrittore e la sua tabella delle pagine si trovano in RAM
 - Il suo spazio di indirizzamento è caricato da disco per ogni riferimento che richiede dati non presenti in RAM
 - Nessun caricamento anticipato di pagine
 - No *working set*
 - Il processo “2” gestisce lo stato dei *page frame* in RAM
 - *Page daemon*
 - Tenendo nucleo di S/O e “mappa delle pagine” (*core map*) **sempre** in RAM
 - Il resto è paginato e ciascuna *page frame* indica il proprio uso
 - Codice, dati, *stack*, tabella delle pagine (altrimenti in lista pagine libere)

Mappa delle pagine (*core map*)

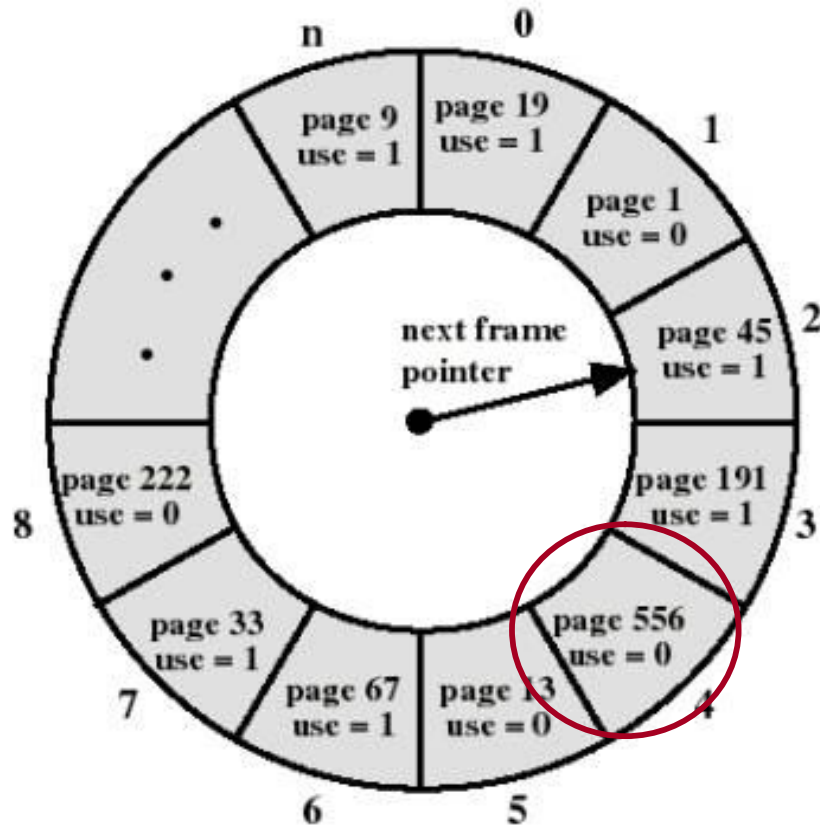


Quale % di RAM occupa la mappa?

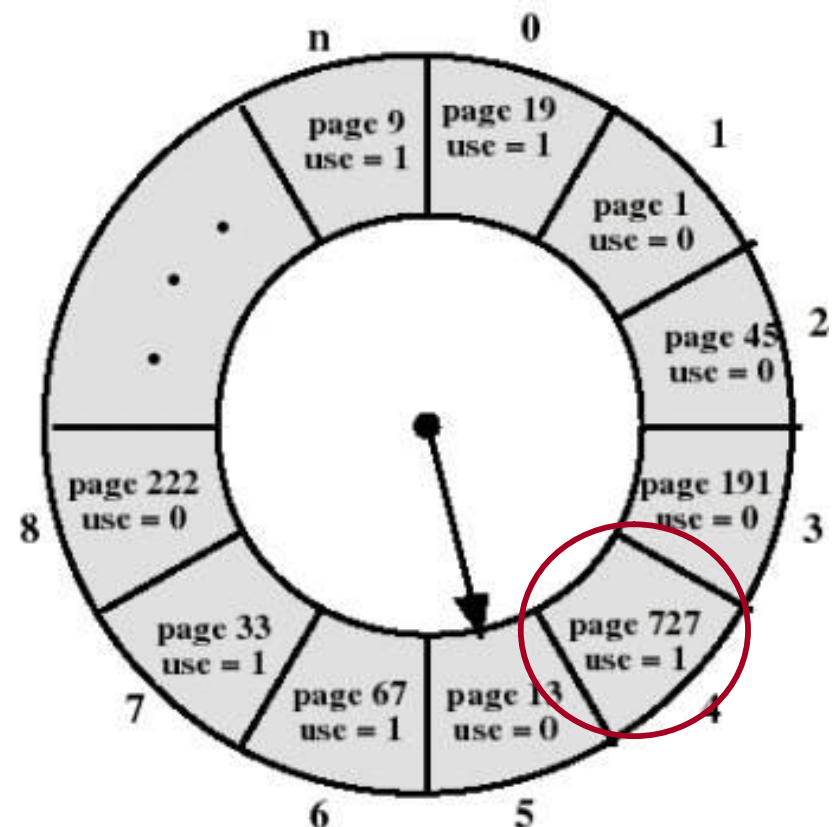
Gestione della memoria (UNIX) – 5

- *Page daemon* verifica con periodico 1/4 s che in RAM vi siano \geq **lotsfree** pagine libere
 - Se ne mancano ne libera quante ne servono salvandone il contenuto corrente su un'area di disco specifica per pagina
 - La selezione delle pagine in uscita usa un algoritmo “**a doppia passata**”
 - *Two-handed clock algorithm*
 - Lista circolare delle pagine
 - La 1^a passata pone a 0 il *bit* di riferimento
 - La 2^a passata, a distanza **programmabile**, rimuove le pagine nel frattempo non riferite (*bit* vale 1 altrimenti)

Orologio a una passata



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Gestione della memoria (UNIX) – 6

- Se vi è spazio libero il *page daemon* riporta in RAM processi pronti selezionati con una euristica di “valore”
 - Caricando solo il descrittore di processo e la sua tabella delle pagine
 - Lasciando che il resto sia caricato via *paging on demand*

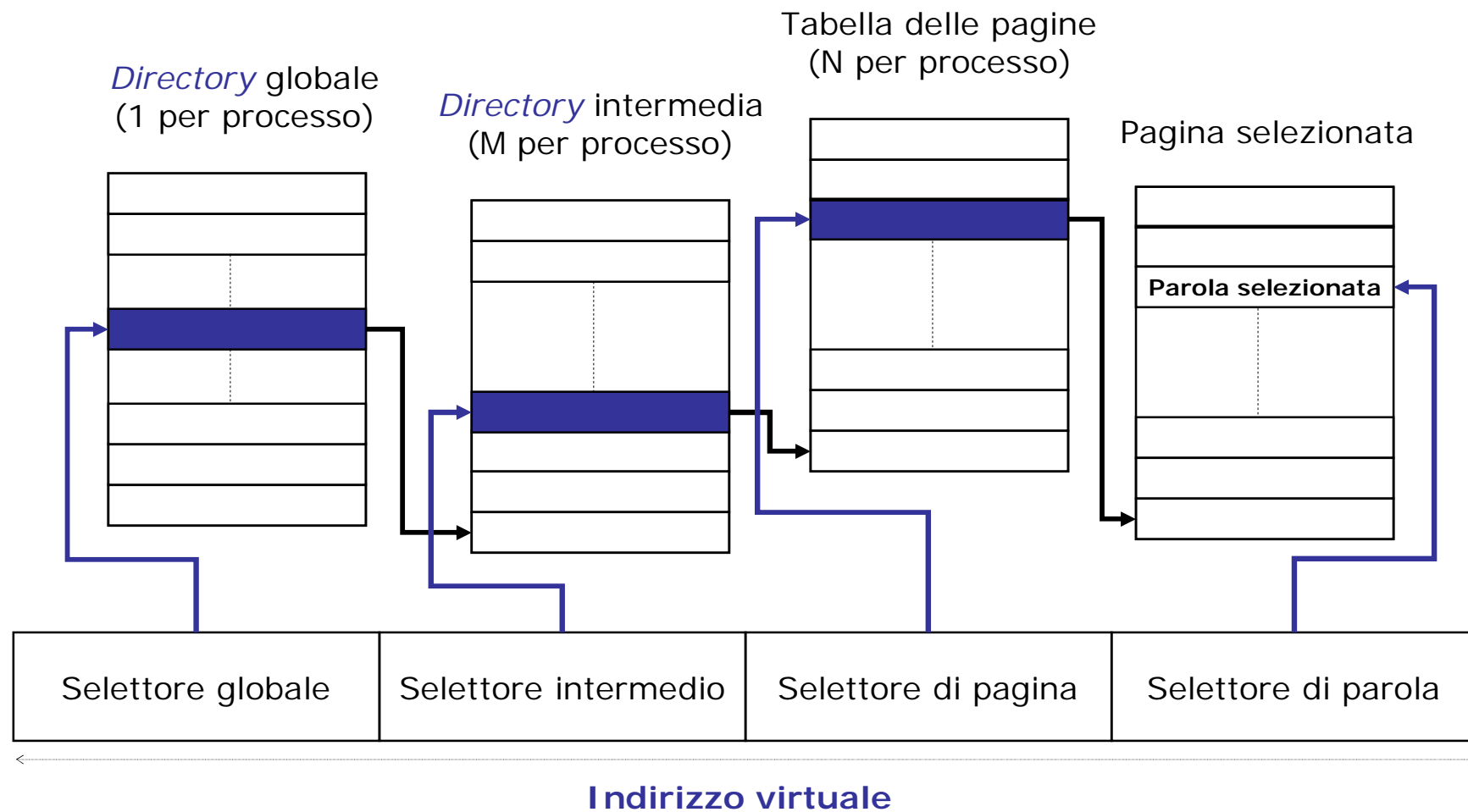
Gestione della memoria (GNU/Linux) – 6

- Per architetture a 32 *bit* la memoria virtuale di processo è ampia 4 GB
 - 1 GB riservato e invisibile al modo operativo normale (*user mode*) per la tabella delle pagine del processo e per altri dati di controllo a uso del nucleo
- Spazio suddiviso in **regioni = sequenze contigue** di pagine
- Ogni regione ha un descrittore noto al nucleo
 - Info su contenuto, pagine attive/inattive, aree libere

Gestione della memoria (GNU/Linux) – 7

- La `fork()` di GNU/Linux replica per il figlio l'intera lista di descrittori del padre
- Le pagine del figlio sono fisicamente duplicate **solo** in caso di modifica (*copy on write*)
 - La regione è marcata **R/W**
 - Le sue pagine dati sono inizialmente marcate **R**
 - Ogni richiesta di scrittura causa eccezione così il nucleo duplica la pagina richiesta e marca la copia come **R/W**

Gestione della memoria (GNU/Linux) – 8



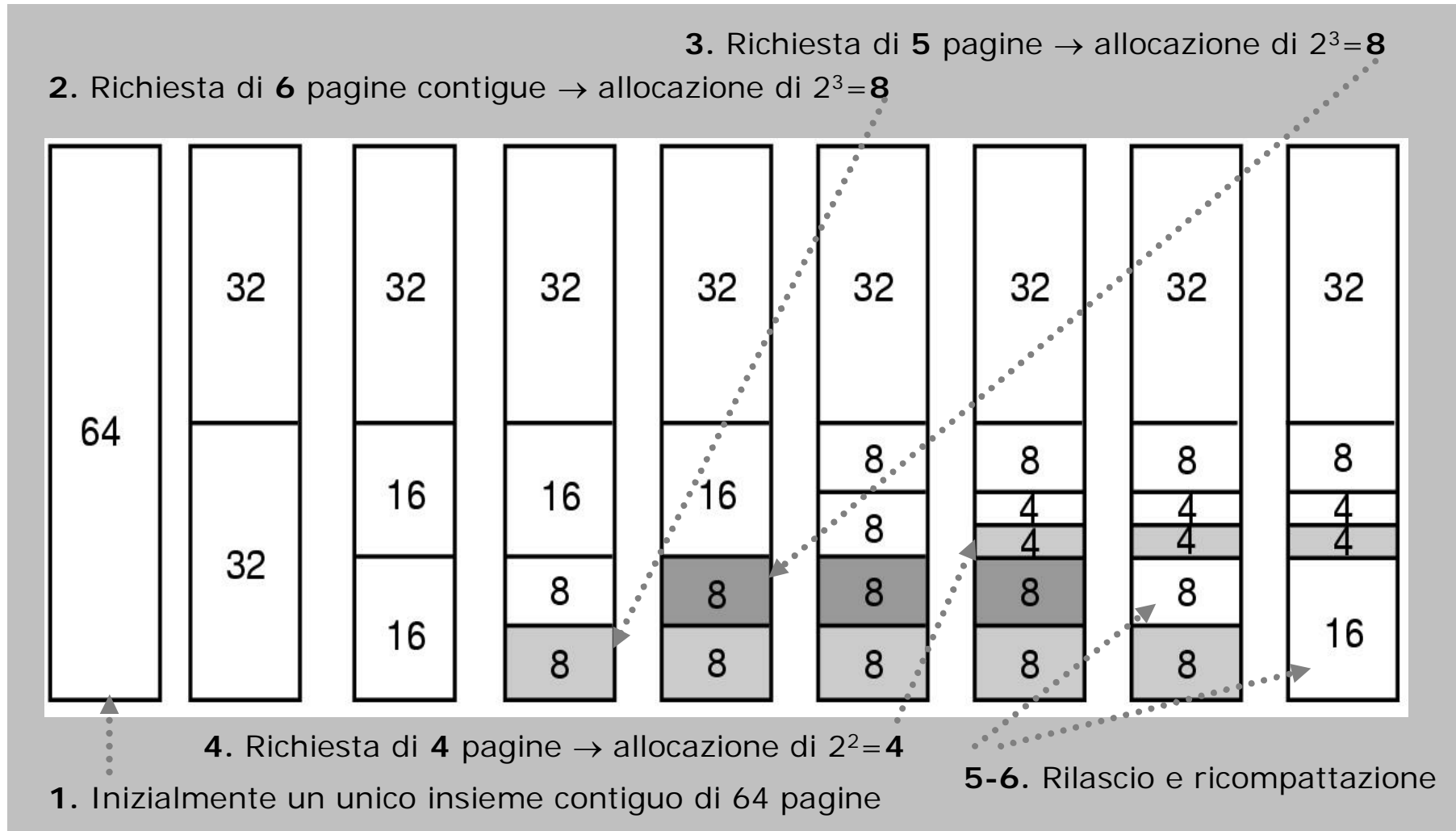
Gestione della memoria (GNU/Linux) – 9

- Il nucleo rimane **sempre** in RAM
- Ad ogni istante nella RAM rimanente possono trovarsi:
 - Le pagine attive dei processi utente
 - Un insieme di pagine utente inattive ma presenti
 - Una *cache* di blocchi di *file* usata dal *file system*
 - Dimensione **variabile** organizzata per pagine
- **kswapd** è il *page daemon* (periodo 1 s) che cerca pagine da spostare su disco
- Il *daemon* **bdf1ush** gestisce la riscrittura delle pagine modificate

Gestione della memoria (GNU/Linux) – 10

- **La RAM è allocata in maniera dinamica e variabile**
- **Algoritmo di allocazione primario (*buddy*)**
 - Ogni richiesta di ampiezza N è arrotondata a $2^m \geq N$
 - La memoria disponibile viene frazionata in metà successive fino a frazioni di ampiezza 2^m
 - Una singola frazione viene assegnata al richiedente
 - Una struttura ausiliaria contiene la testa di liste predefinite di frazioni di ampiezza 2^i ($i=0, \dots, m$) per velocizzare la ricerca
 - La memoria disponibile usata per l'allocazione è sempre la frazione libera di minore dimensione
 - Al rilascio ogni frazione tornata libera si unisce con la frazione vicina se libera (il suo *buddy*)
 - algoritmi sussidiari cercano di ridurre la frammentazione causata dall'algoritmo primario
 - Strutture *slab* per evitare di usare 128 pagine per allocarne 65

Funzionamento del *buddy algorithm*



Dimensione min e max area allocabile?

Gestione dell'I/O – 1

- UNIX tratta i dispositivi di I/O come *file* di tipo speciale, ciascun con posizione specifica nel FS
 - Per esempio `/dev/...`
- Un gestore (*device driver*) è associato in modo esclusivo a ciascun dispositivo o a famiglia di dispositivi dello stesso tipo
 - Una coppia di indici `<maggiore, minore>` identifica precisamente ciascun dispositivo di I/O
 - Maggiore: tipologia
 - Minore: specifico di quel dispositivo

Gestione dell'I/O – 2

- GNU/Linux consente invece caricamento **dinamico** dei moduli di gestione dei dispositivi
 - Soluzione molto preferibile alla configurazione statica che richiede ogni volta una nuova compilazione dell'intero nucleo
 - Inevitabile a fronte della grande varietà di *hardware* attuale
- Il caricamento dinamico richiede al nucleo di effettuare diverse azioni di configurazione
 - **[1]** Rilocazione dello spazio di indirizzamento del modulo
 - **[1-2]** Allocazione delle risorse necessarie
 - P. es.: interruzione assegnata al dispositivo
 - **[2]** Configurazione del vettore delle interruzioni
 - **[2]** Attivazione e inizializzazione del gestore

Gestione dell'I/O – 3

- Un *file* speciale (detto *socket*) viene utilizzato per la connessione di rete e i relativi protocolli
 - Può essere creato e distrutto dinamicamente
 - Un *socket* è associato a uno specifico indirizzo di rete
- Tre tipi di connessione con scelta alla creazione
 - Connessione affidabile a flusso di caratteri (~ **TCP**)
 - Il gestore garantisce la correttezza della trasmissione
 - Invio e ricezione per blocchi di dimensione variabile
 - Connessione affidabile a flusso di pacchetti (**TCP**)
 - Come sopra, ma con invio e ricezione solo per pacchetti
 - Trasmissione inaffidabile di pacchetti (**UDP**)
 - L'utente deve occuparsi di trattare gli eventuali errori