# Linux Virtual Memory Management interview

## Các loại bộ nhớ trong máy tính

Bao gồm:

* Các loại bộ nhớ chính:
  + RAM: SRAM, DRAM, DDR SDRAM
  + ROM: PROM, EEPROM, Flash
  + Cache: Cache L1, L2, L3
  + Bộ nhớ ngoài: HDD, SSD, thẻ nhớ,…

### Bổ sung: Phân biệt 3 bộ nhớ Cache: L1, L2 và L3

Cache L1

* **Vị trí**: Tích hợp trực tiếp trong lõi CPU (core)
* **Dung lượng**: Nhỏ nhất, thường từ 32KB đến 128KB cho mỗi lõi
* **Tốc độ**: Nhanh nhất, độ trễ thấp nhất (khoảng 0.9-1.3ns)
* **Cấu trúc**: Thường chia thành Cache lệnh (instruction cache) và Cache dữ liệu (data cache)
* **Truy cập**: CPU truy cập trước tiên vào L1 khi cần dữ liệu

Cache L2

* **Vị trí**: Thường tích hợp trong CPU nhưng bên ngoài lõi, mỗi lõi có thể có L2 riêng hoặc một số lõi chia sẻ
* **Dung lượng**: Lớn hơn L1, thường từ 256KB đến 2MB cho mỗi lõi
* **Tốc độ**: Chậm hơn L1 nhưng nhanh hơn L3 (độ trễ khoảng 3-10ns)
* **Cấu trúc**: Thường là bộ nhớ đệm hợp nhất (unified cache), lưu trữ cả lệnh và dữ liệu
* **Truy cập**: CPU truy cập vào L2 khi không tìm thấy dữ liệu trong L1

Cache L3

* **Vị trí**: Tích hợp trong CPU nhưng được chia sẻ giữa tất cả các lõi
* **Dung lượng**: Lớn nhất, thường từ 4MB đến 64MB (hoặc nhiều hơn trên CPU máy chủ)
* **Tốc độ**: Chậm nhất trong ba loại cache (độ trễ khoảng 10-40ns), nhưng vẫn nhanh hơn RAM
* **Cấu trúc**: Luôn là bộ nhớ đệm hợp nhất, có thể được chia thành các phân đoạn
* **Truy cập**: Hoạt động như lớp trung gian giữa các cache L2 và RAM

So sánh hiệu suất và vai trò

* **Thứ tự truy cập**: CPU → L1 → L2 → L3 → RAM
* **Tỷ lệ truy cập thành công**: L1 > L2 > L3 (cache hit rate)
* **Vai trò của L1**: Tối ưu hóa tốc độ xử lý ngay tại lõi
* **Vai trò của L2**: Giảm áp lực lên L1 và cải thiện hiệu suất khi có nhiều dữ liệu
* **Vai trò của L3**: Giảm độ trễ khi truy cập RAM và hỗ trợ chia sẻ dữ liệu giữa các lõi CPU

Cả ba loại bộ nhớ cache đều sử dụng công nghệ SRAM (Static RAM) để đạt được tốc độ cao hơn nhiều so với bộ nhớ chính RAM (thường là DRAM).

## Tại sao cần sử dụng Virtual memory

Có 3 lý do chính:

* Quản lý phân mảnh bộ nhớ:
  + VD: Xét bộ nhớ RAM 4Gb và có 4 process chiếm mỗi process 1GB ở địa chỉ 1, 2, 3, 4G. Nếu giờ process 1 và 3 đã được giải phóng thì với cơ chế quản lý trực tiếp, hệ điều hành cũng không thể malloc thêm 1 vòng nhớ 2Gb nào cả vì không có dải địa chỉ liền mạch nào đủ dài.
* Tránh phải load toàn bộ chương trình lên RAM ngay từ đầu:
  + VD: Khi chương trình được load sẽ cần phải chừa thêm đủ chỗ cho stack và heap hoạt động ngay từ đầu. Còn nếu có VTM thì các địa chỉ này có thể cấp phát sau.
* Bảo vệ tính toàn vẹn dữ liệu
  + VD: Mỗi process có bảng phân trang riêng nên process không thể truy cập vào vùng nhớ của process khác.

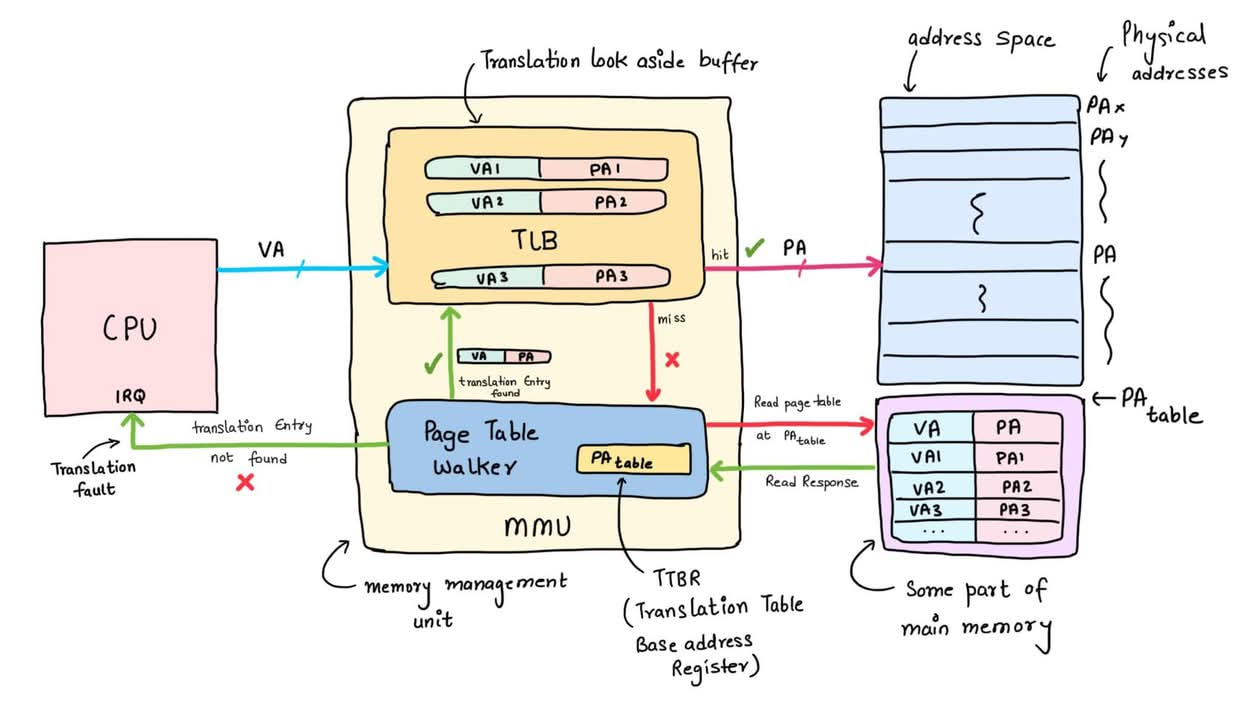
Tính năng thêm:

* Tận dụng bộ nhớ vật lý khi hết RAM (swap space)

## Cơ chế hoạt động của Virtual Memory

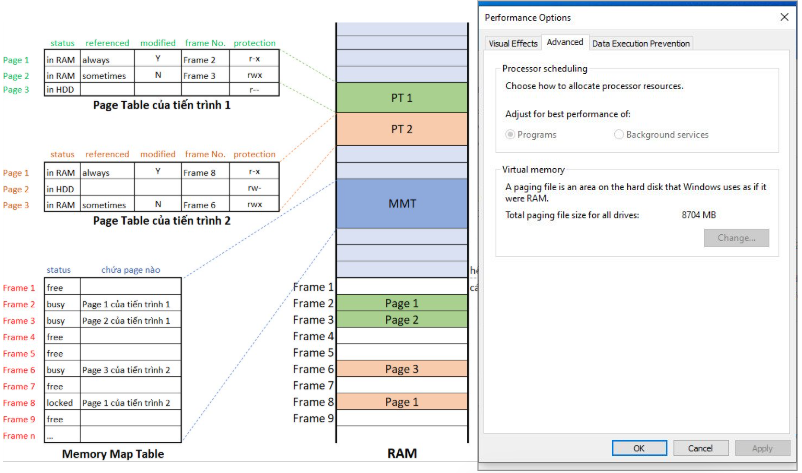
### Virtual Memory được triển khai nhờ cả phần mềm và phần cứng

* Phần cứng: Cần có khối MMU (Nếu không có sẽ là bản linux nommu)
* Phần mềm: Cần có hệ điều hành hỗ trợ Virtual Memory

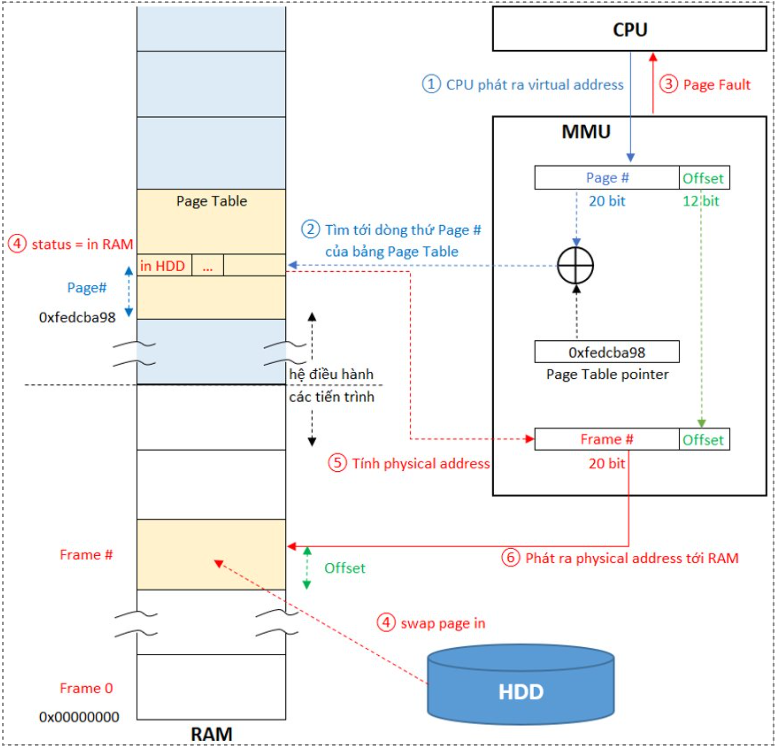


Hệ điều hành chia RAM thành các Frame (thường là 4kb, có thể thay đổi bằng cấu hình PAGE\_SHIFT) và tạo ra bảng Memory Map Table để quản lý các frame. Vùng nhớ của mỗi process chia ra làm nhiều page, có kích thước bằng frame, bảng quản lý các Page này là Page Table.

* Ví dụ:
  + CPU 32 bit, RAM 1GB, Frame 4Kb -> Memory map table = 2^18 line \* 32 bit = 1 MB
  + CPU 64 bit, RAM 16G, Frame 16Kb -> Memory map table = 2^20 line \* 64 bit = 8 MB



### Cơ chế quản lý bộ nhớ

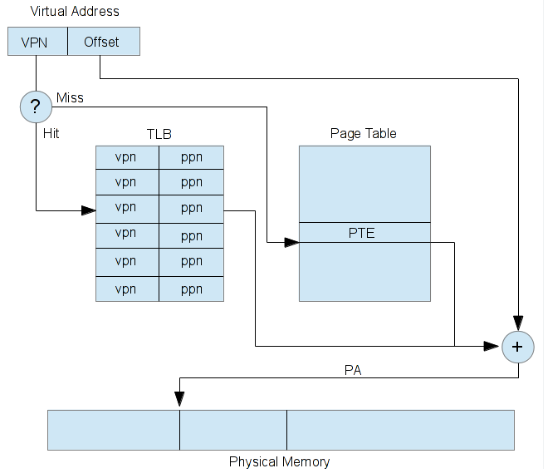


Nhược điểm cần khắc phục:

* Mất thời gian tham chiếu đến bảng page table, tốc độ tìm kiếm dữ liệu giảm
* Nếu số lượng process lớn, kích thước page table sẽ lớn chiếm nhiều RAM

# Các cơ chế tiết kiệm thời gian truy xuất

## Cơ chế bảng TLB (Translation Lookaside Buffer)



Cơ chế này được hỗ trợ bởi phần cứng.

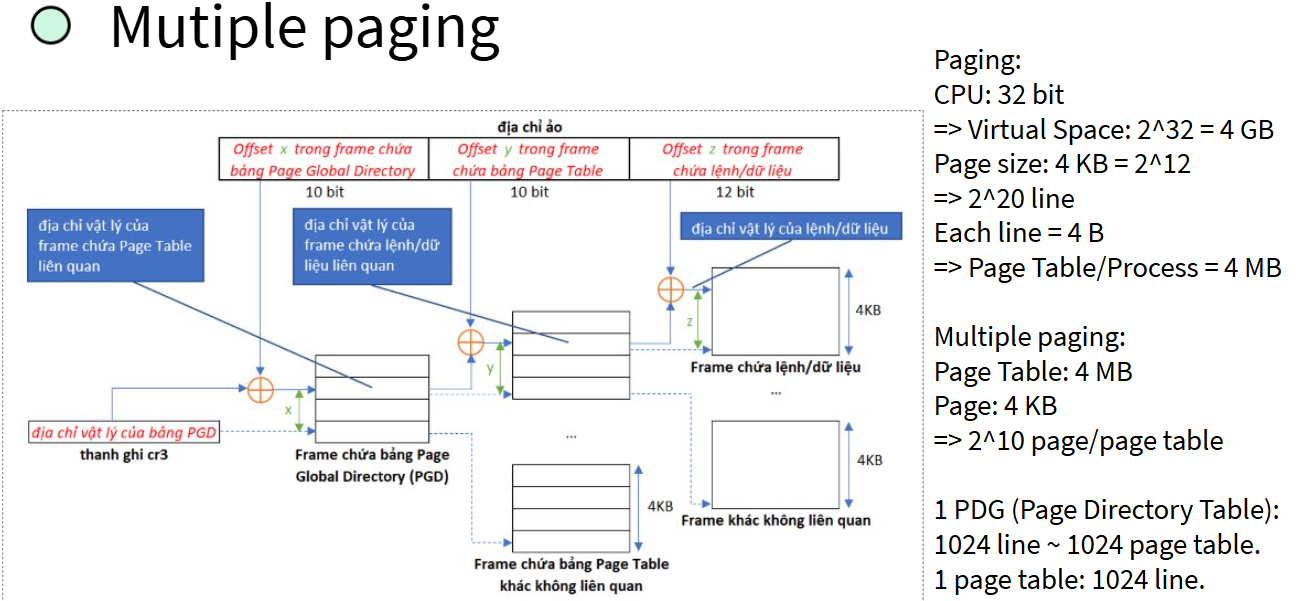
Dựa vào tính chất LOR (Locality of Reference) tức là các chương trình có xu hướng tham chiếu đến 1 hoặc 1 vài page gần nhau trong khoảng thời gian ngắn, bộ đệm TLB giúp MMU không phải tra cứu bảng Page Table quá nhiều. Khi nhận được 1 virtual address, MMU sẽ tra cứu thông tin trong TLB, nếu thông tin bị miss thì mới tra cứu bảng Page table.

Ưu điểm: Tăng tốc độ tra cứu đến các địa chỉ hoặc các page đã được truy cập gần đây

Nhược điểm: Mất thêm thời gian để tra cứu đến các page chưa có trong TLB

* + Tăng TLB size để giải quyết nhược điểm nhưng sẽ làm MMU trở nên đắt tiền hơn

## Cơ chế multi-level page



Cơ chế này cho phép tạo các bảng Page Table phân cấp. Khi MMU tìm kiếm dữ liệu chỉ cần thực hiện tìm kiếm dạng thư mục phân cấp thay vì phải XOR với tất cả các Entry.

Ưu điểm: giúp tăng tốc độ tra cứu bảng Page Table

Nhược điểm: Tăng bộ nhớ cần sử dụng để chứa PGD, PMD,..

Update: Ở thời điể Linux v6.4 64 bit thì Mutiple Paging đã dùng 5 cấp phân trang bao gồm: PGD, P4D, PUD, PMD, PTE

## Bộ nhớ NUMA

# Các cơ chế tiết kiệm/mở rộng không gian lưu trữ

## Cơ chế Demand Paging

<https://offlinemark.com/demand-paging/>

Hệ điều hành sẽ không phân bổ bộ nhớ ngay lập tức (Lazy load) khi allocate memory cho process. Nó chỉ đánh dấu vùng nhớ đó trong process là có thể truy cập, tạo ra bảng Page Table cho vùng nhớ chứ không thực sự cấp phát Frame trên Ram cho vùng nhớ. Chỉ khi nào vùng nhớ bị sửa đổi (ghi vào) thì mới tạo vùng nhớ trên RAM. Nếu chỉ đơn thuần đọc thì không trigger việc tạo vùng nhớ mà thay vào đó tham chiếu đến “Zero page” trên RAM đã được tạo sẵn (pre-allocated)

Lợi ích:

* Giảm sử dụng RAM không cần thiết
  + VD RAM 1G cũng có thể malloc ra vùng nhớ 100G

Nhược điểm:

* Truy cập vào các vùng nhớ chưa được load sẽ bị chậm hơn

## Cơ chế Swapping Space

Có thể được bật/tắt bằng cách cấu hình trong hệ điều hành. Swap Space là không gian bộ nhớ vật lý trên ổ đĩa như HDD hoặc SSD hoạt động như là 1 phần của RAM. Hệ điều hành sẽ quyết định chuyển 1 số page sang Swap Space để làm trống RAM cho các tiến trình khác hoạt động.

Số lượng hoán đổi dựa trên chỉ số swapiness được hệ điều hành quản lý. Nếu chỉ số này cao thì hệ điều hành càng tích cực đẩy các page sang disk và ngược lại.

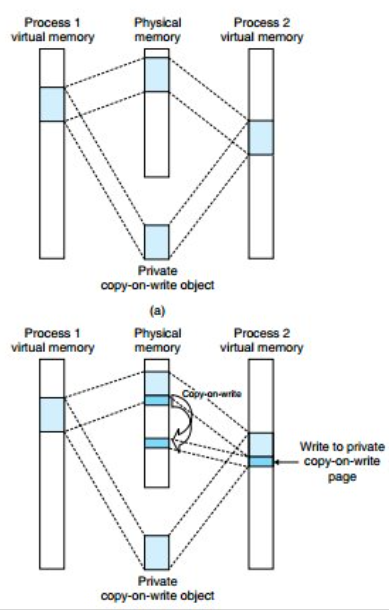
Ưu điểm:

* Tránh quá tải RAM
* Hỗ trợ tốt hơn cho các chương trình lớn và multitasking

Nhược điểm:

* Các loại HDD và SSD rất chậm so với RAM nên tốc độ thực thi chương trình có thể bị giảm đáng kể khi RAM đầy
* Giảm dung lượng bộ nhớ
* Giảm tuổi thọ ổ cứng

## Chia sẻ bộ nhớ (COW)



## Kernel Same-page Merging (KSM)

Là tính năng chống trùng lặp giúp tiết kiệm bộ nhớ, có thể được bật bằng CONFIG\_KSM=y

KSM daemon sẽ quét vùng bộ nhớ đã được đăng kí với nó, thay thế các page có nội dung trùng lặp bằng 1 trang duy nhất, chống ghi và có nội dung tương tự. Nếu process muốn thực hiện thay đổi trên các trang này sẽ COW.

KSM có thể được điều khiển bằng hàm madvise().

Ưu điểm: Tối ưu bộ nhớ, giảm trùng lặp dữ liệu

## Huge Page

Do RAM tăng lên dẫn đến số lượng Page cũng tăng, dẫn đến tăng áp lực quản lý cho bộ đệm TLB (cache miss nhiều làm chậm quá trình xử lý)

* + Tăng size của Page
  + Huge Page

Lợi ích:

* Giảm áp lực lên TLB, tăng cache hit
* Giảm áp lực lên việc quản lý bộ nhớ Page Table

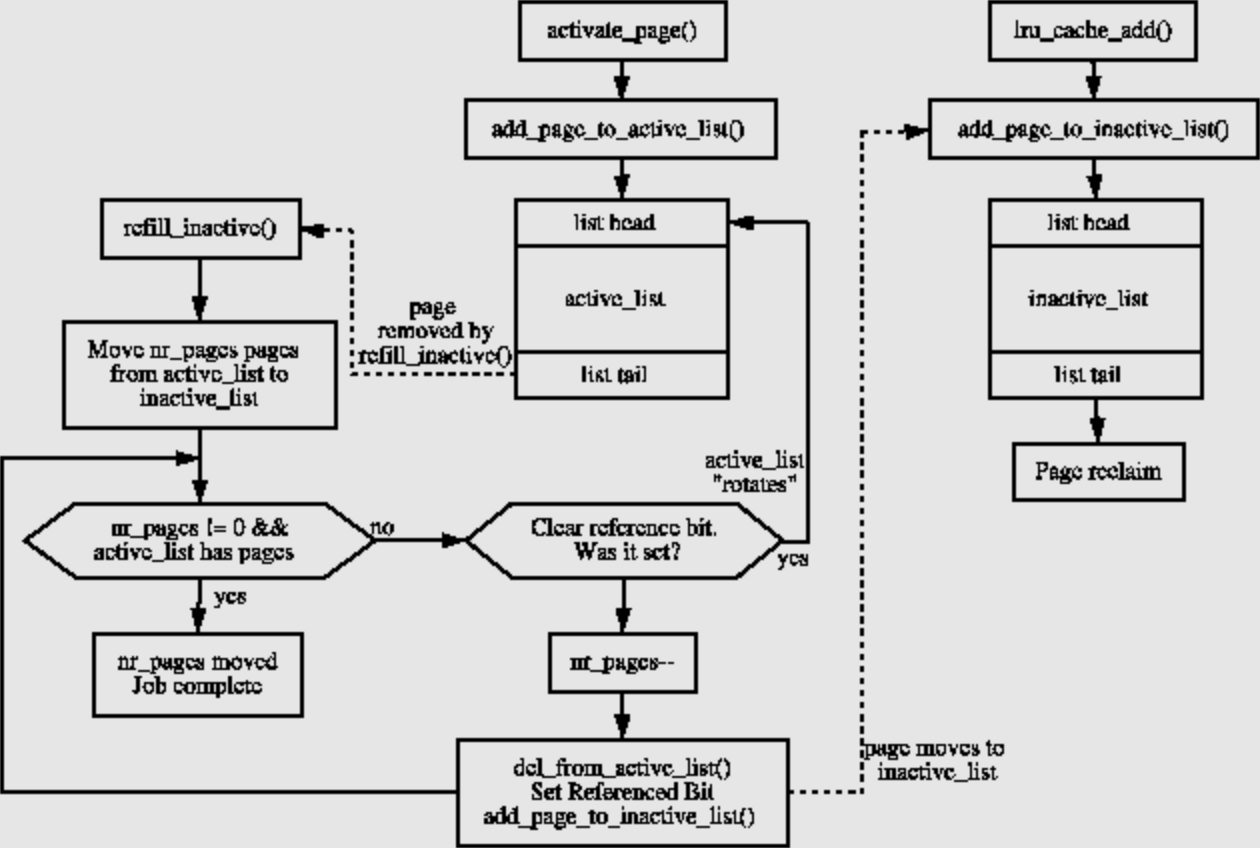
Nhược điểm:

* Tăng phân mảnh nội bộ
* Độ trễ cao khi cấp phát và giải phóng

# Các cơ chế giải phóng và thu hồi bộ nhớ

## Thuật toán LRU (Least Recently Used)

LRU trong linux chỉ là xấp xỉ LRU bao gồm active\_list và inactive\_list. Mỗi trang sẽ có 1 bit active. Định kì kiểm tra lần lượt các page, nếu page đã được gọi đến gần đây sẽ có bit này bằng 1 thì đưa lên đầu active\_list, nếu không cho page này vào inactive list. Page reclaim sẽ ưu tiên lấy lại các page từ inactive\_list (swap sang ổ cứng).



Ưu điểm :

* Giúp xác định page nào nên được reclaim

Nhược điểm :

* Tốn thời gian vào việc quét các page theo chu kì
* Có thể bị ảnh hưởng bởi các page chỉ gọi 1 lần

## OOM Killer (Out-Of-Memory Killer)

## Control groups (cgroups)

## Direct Reclaim