

## سیستم عامل جلسه ششم

### سلسله مراتب حافظه

بین سه ویژگی کلیدی حافظه یعنی ، هزینه ، ظرفیت، زمان دسترسی باید سبک و سنگین کرد. برای این کار نمی توان بر یک حافظه یا فن آوری خاصی تکیه کرد و باید از سلسله مراتب حافظه استفاده کرد

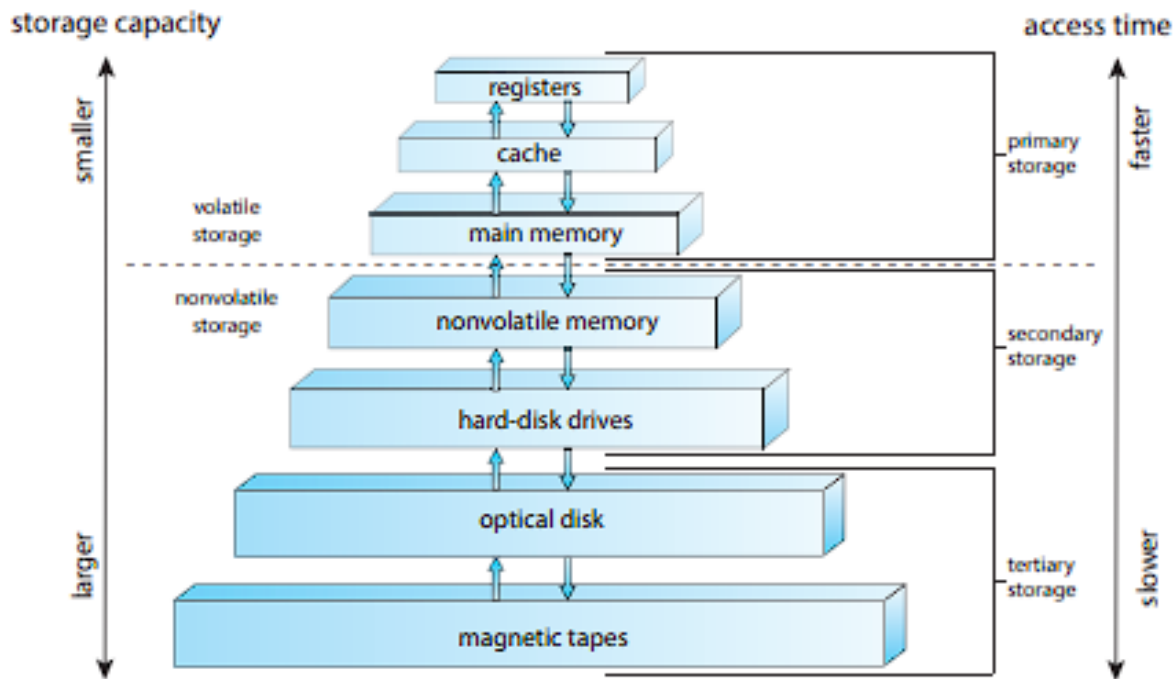


Figure 1.6 Storage-device hierarchy.

## سلسله مراتب متدوال حافظه:

1. ثبات ها (Register)
2. حافظه پنهان (cache)
3. حافظه اصلی (Main Memory)
4. حافظه پنهان دیسک
5. دیسک مغناطیسی (Hard Disk - HDD)
6. رسانه جا به جا پذیر (USB – DVD - CD)

با حرکت به سطوح پایین تر این سلسله مراتب شرایط زیر رخ می دهد:

- 1 – کاهش هزینه در هر بیت 2- افزایش ظرفیت 3- افزایش زمان دسترسی
- 4- کاهش تعداد دفعات دسترسی پردازنده به حافظه

مدیریت منابع هر یک از حافظه های زیر در مقابل آن نوشته شده است:

ثبات ها (Register): کامپایلر

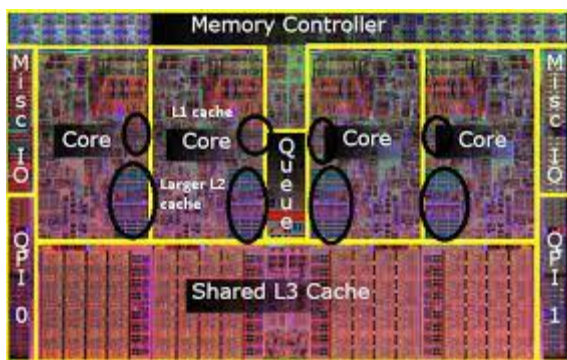
حافظه پنهان (cache): خودکار (توسط خود پردازنده)

حافظه اصلی و دیسک (main Memory and hard disk) : سیستم عامل

## حافظه پنهان (cache)

A CPU cache is a hardware cache used by the central processing unit (CPU) of a computer to reduce the average cost (time or energy) to access data from the main memory. A cache is a smaller, faster memory, located closer to a processor core, which stores copies of the data from frequently used main memory locations.

یک حافظه ی کوچک و سریع بین پردازنده و حافظه اصلی (Main Memory) است. حافظه پنهان حاوی بخشی از حافظه اصلی است.



وقتی پردازنده می خواهد کلمه ای از حافظه را بخواند وجود آن را در حافظه ی پنهان بررسی می کند. اگر وجود داشته باشد به پردازنده تحویل داده می شود در غیر این صورت یک بلوک از حافظه اصلی شامل تعداد ثابتی از خانه های حافظه به حافظه پنهان منتقل می شود و سپس کلمه مورد نظر به پردازنده تحویل داده می شود. هنگامی که یک بلوک از داده ها به حافظه پنهان آورده می شود تا یک مراجعه به حافظه انجام شود ، به دلیل پدیده ی محلی بودن مراجعات ، احتمالاً به زودی به دیگر کلمات آن بلوک نیز مراجعه خواهد شد.

در یک سیستم کامپیوتری، برای افزایش کارایی از حافظه های چند سطحی استفاده می کنند به گونه ای که سطوح نزدیک تر به پردازنده دارای ظرفیت کمتر اما در عوض سرعت بیشتری هستند. یکی از این موارد استفاده از حافظه پنهان است بدین صورت که هرگاه دستورالعمل یا داده ای در حافظه اصلی مورد استفاده قرار گیرد یک کپی از آن در حافظه پنهان ایجاد می شود. دلیل آن این است که بر اساس اصل محلی گرایی گفته می شود هر گاه داده هایی مورد استفاده قرار گیرند به زودی در آینده نیز لازم خواهد بود. قرار دادن این داده ها و دستورالعمل ها در حافظه پنهان موجب افزایش سرعت دستیابی می شود بنابراین زمانی که داده یا دستورالعملی نیاز باشد ابتدا به حافظه پنهان مراجعه می شود و در صورت وجود آن، استفاده می شود.

در غیر این صورت به حافظه اصلی مراجعه شده و داده ها و دستورالعمل های مورد نظر، مورد استفاده قرار می گیرند.

از آن جایی که ظرفیت حافظه پنهان نسبت به حافظه اصلی بسیار کمتر است بنابراین نخواهیم توانست همه داده هایی را که در حافظه اصلی هستند را به حافظه پنهان ببریم پس ممکن است گاهی به داده هایی نیاز داشته باشیم که در حافظه پنهان نیستند. اگر چنانچه حافظه پنهان پر شده باشد و نیاز به خالی کردن بخشی از آن و جایگزینی آن با داده مورد نظر داشته باشیم از الگوریتم های جایگزینی حافظه استفاده می شود که دقیقاً همان الگوریتم های جایگزینی صفحه در حافظه اصلی هستند

یادداشت:

صفحه (Page) در حافظه اصلی (RAM) به عنوان یک بخشی از فضای آدرس پذیر قرار دارد و به نوعی یک بلاک داده است که شامل اطلاعاتی مانند برنامه‌ها، داده‌ها و سایر اطلاعاتی است که توسط سیستم عامل به حافظه اصلی بارگذاری می‌شود. هر صفحه، دارای یک سایز خاص است و به طور معمول، اندازه یک صفحه در حافظه اصلی، 4 کیلوبایت است.

هنگامی که یک برنامه را اجرا می‌کنید، سیستم عامل صفحات مربوط به برنامه را در حافظه اصلی بارگذاری می‌کند. در فرآیند اجرای برنامه، اگر برنامه به داده‌های جدیدی نیاز داشت، صفحات جدید به صورت پویا در حافظه اصلی ساخته می‌شوند و اگر برنامه به صفحات قبلی دسترسی نداشت، صفحات قبلی از حافظه اصلی حذف می‌شوند.

استفاده بهینه و مناسب از صفحات در حافظه اصلی، باعث افزایش سرعت و کارایی سیستم شما می‌شود. بنابراین، طراحی و پیاده‌سازی الگوریتم‌هایی برای مدیریت صفحات در حافظه اصلی، یکی از اصلی ترین وظایف سیستم عامل است

## سخت افزایه پایه (Base Hardware)

حافظه اصلی و ثبات های ساخته شده در خود پردازنده ، تنها فضای ذخیره سازی همه منظوره ای هستند که پردازنده مستقیما می تواند به آن ها دسترسی داشته باشد

هر دستور العمل در حال اجرا و داده ای که توسط آن استفاده می شود، باید در یکی از این دو دستگاه ذخیره سازی با دستیابی مستقیم (حافظه ی اصلی و ثبات ها) واقع باشد.

## انقیاد آدرس (Address Binding)

معمولا برنامه بر روی دیسک به صورت یک فایل اجرایی دودویی ذخیره می شود. برنامه باید به حافظه بار شود و در داخل فرایندی قرار گیرد تا اجرا شود. بر حسب این که چه مدیریت حافظه ای مورد استفاده قرار می گیرد. این فرایند ممکن است در حین اجرا بین دیسک و حافظه انتقال یابد. فرایند های موجود در دیسک که منتظرند وارد حافظه و اجرا شوند، صف ورودی را تشکیل می دهند.

## فضای آدرس منطقی و فیزیکی (physical and logical address)

### یادداشت:

**A logical address is the virtual address that is generated by the CPU. A user can view the logical address of a computer program. On the other hand, a physical address is one that represents a location in the computer memory. A user cannot view the physical address of a program.**

آدرسی که توسط پردازنده تولید می شود، آدرس منطقی (Logical Address) نام دارد، در حالی که آدرسی که توسط واحد حافظه مشاهده می شود (آدرسی که به ثبات آدرس حافظه بار می شود)، آدرس فیزیکی (Physical Address) نام دارد

مجموعه ای از تمام آدرس های منطقی که توسط برنامه ای تولید می شود، فضای آدرس منطقی (Logical Address Space) نام دارد. مجموعه ای از تمام آدرس های فیزیکی متناظر با این آدرس های منطقی، فضای آدرس فیزیکی (physical Address Space) نام دارد.

نگاشت زمان اجرا از آدرس های مجازی به فیزیکی، توسط واحد مدیریت حافظه Memory Management Unit (MMU) انجام می شود که یک دستگاه سخت افزاری است.

برنامه ی کاربر هیچ گاه آدرس های فیزیکی واقعی را نمی بیند.

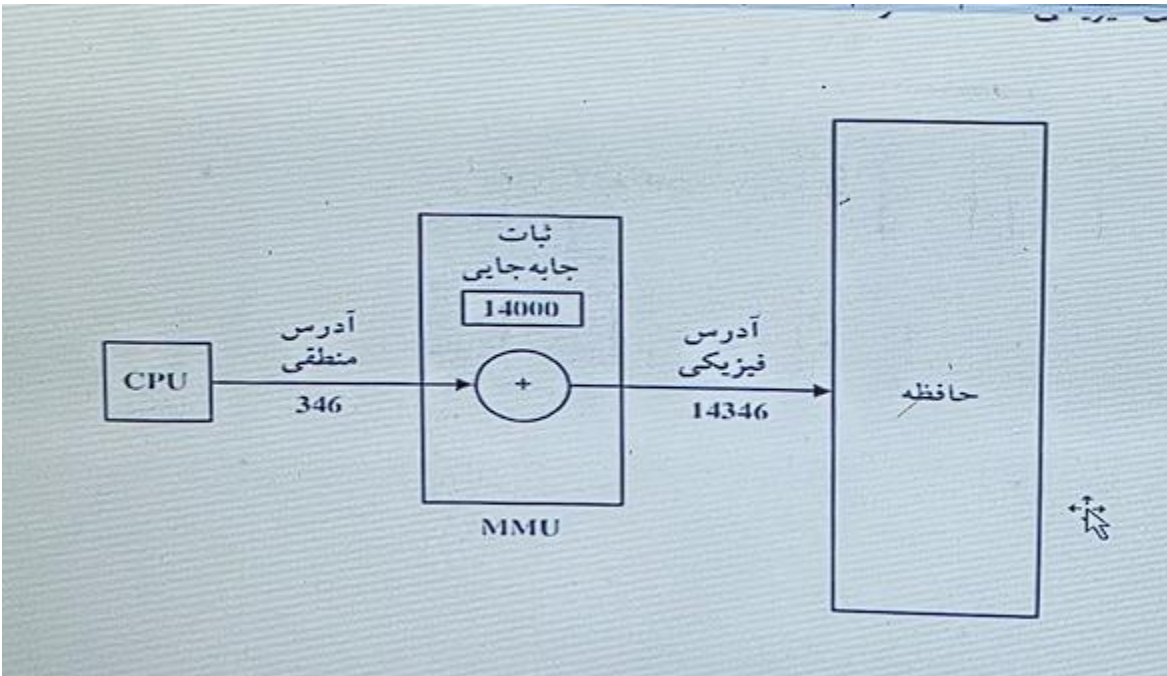
سخت افزار نگاشت حافظه ، آدرس های منطقی را به آدرس های فیزیکی تبدیل می کند

دو نو آدرس وجود دارد

آدرس های منطقی (از صفر تا  $\max$ )

آدرس های فیزیکی (از  $R + 0$  تا  $\max + R$  با مقدار پایه ی  $R$  )

کاربر فقط آدرس های منطقی را تولید می نماید و فکر می کند که فرایند در محل های صفر تا  $\max$  اجرا می شود. برنامه کاربر ، آدرس های منطقی را تولید می کند و این آدرس ها قبل از بکارگیری باید به آدرس های فیزیکی نگاشت شوند.

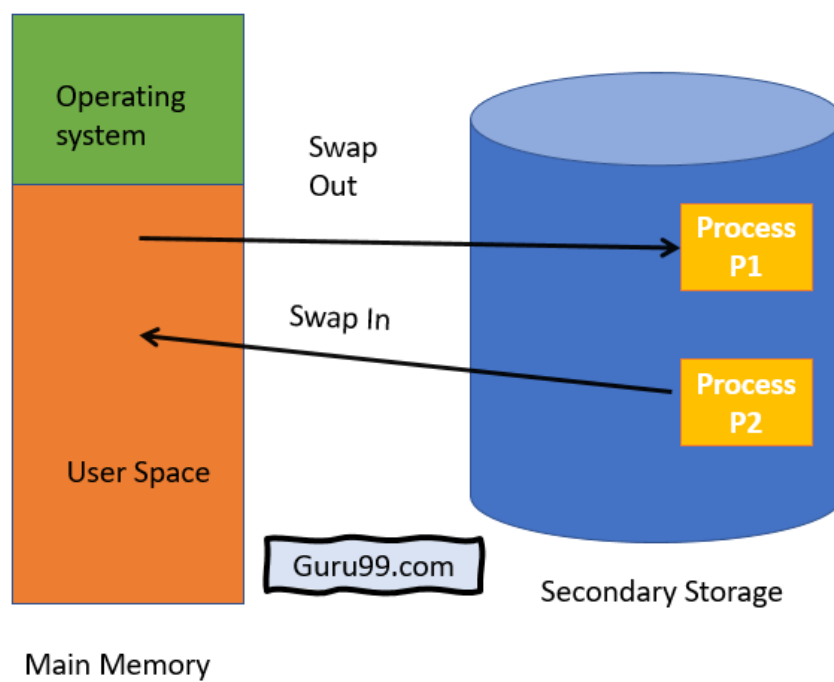
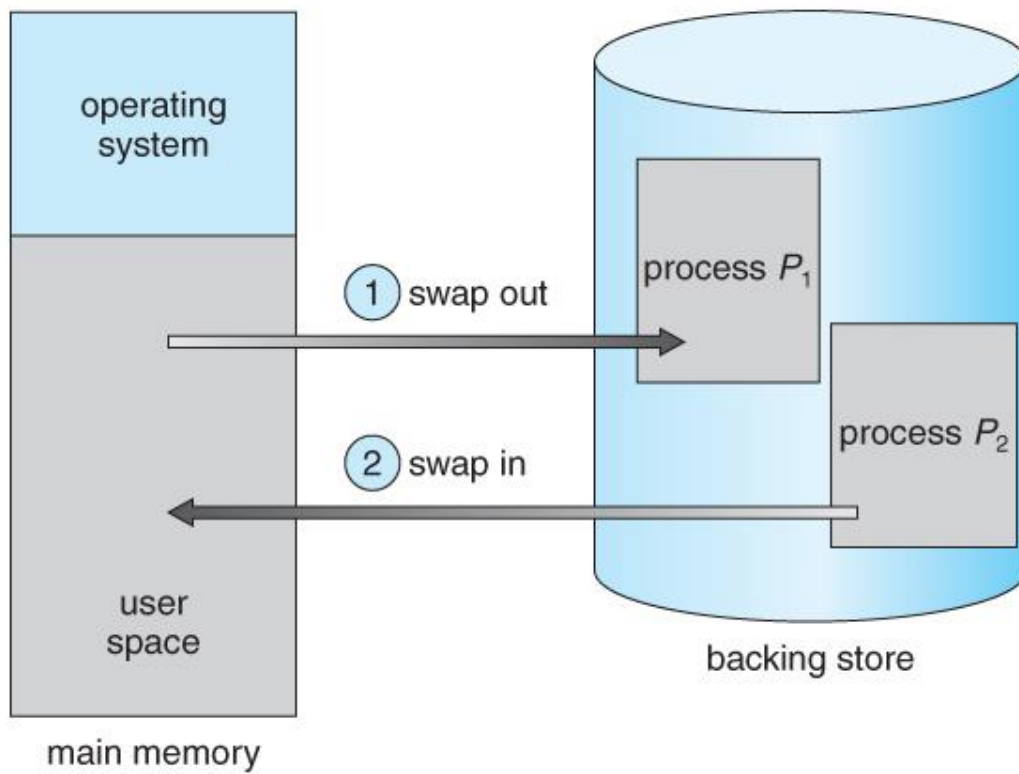


## مبادله (Swapping)

فرایند باید در حافظه باشد تا اجرا شود. اما، فرایند می تواند موقتا از حافظه اصلی به ذخیره ساز پشتیبان برود و بعد به حافظه برگردد و به اجرایش ادامه دهد. این عمل را مبادله می گویند

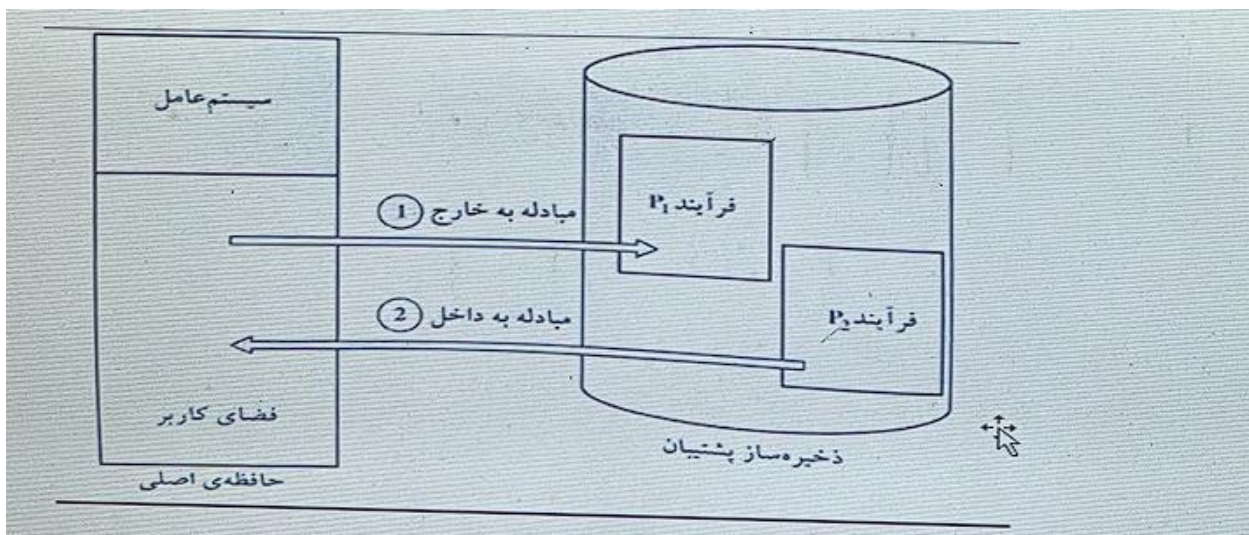
Swapping is a technique used in operating systems to temporarily move pages or entire processes from the main memory (RAM) to secondary storage devices, such as hard disks, to free up space in the main memory. When the system needs to access the data that has been swapped out, it swaps the less frequently used data back into the main memory and swaps out other data to secondary storage again. This process is called swapping because the system is essentially swapping data between the main memory and secondary storage. Swapping enables a system to run larger applications or more applications concurrently than it could if it relied solely on the available physical memory.





## مبادله استاندارد

مبادله ی استاندارد شامل انتقال فرایند بین حافظه ی اصلی و ذخیره ساز پشتیبان است. ذخیره ساز پشتیبان معمولاً یک دیسک سریع است و باید به اندازه کافی بزرگ باشد تا بتواند تمام تصاویر و حافظه را برای تمام کاربران ذخیره کند و دستیابی مستقیم به این ذخیره ساز را فراهم آورد.



## تخصیص حافظه ی همجوار (Contiguous Memory allocation)

حافظه ی اصلی باید سیستم عامل و فرایندهای کاربران را جا دهد. بنابراین لازم است بخش های مختلفی از حافظه اصلی به روش کارآمدی تخصیص یابند.

حافظه معمولاً به دو قسمت تقسیم می شود:

یک قسمت برای سیستم عامل مقیم

قسمت دیگر برای فرایندهای کاربر

یادداشت:

تخصیص حافظه‌ی همجوار یک روش اختصاص حافظه است که در آن، برای یک فرآیند یا برنامه، یک بلوک پیوسته و پی در پی از حافظه به طول مشخصی اختصاص داده می‌شود. در این روش، مسئولیت تخصیص و اداره حافظه به عهده‌ی سیستم عامل است.

استفاده از حافظه‌ی همجوار در برنامه‌ها باعث بهبود کارایی و سرعت اجرای برنامه می‌شود، زیرا با این روش، زمان لازم برای جستجوی فضاهای خالی در حافظه کاهش می‌یابد و برنامه به راحتی می‌تواند به سمت قسمت‌های پیوسته از حافظه حرکت کند.

اما با این حال، تغییر در اندازه یا نوع حافظه‌ای که به یک برنامه اختصاص داده شده است، ممکن است سبب شکستگی برنامه شود و مشکلاتی برای سیستم عامل و پردازنده‌ها ایجاد کند. بنابراین، استفاده از تخصیص حافظه‌ی همجوار در صورتی مناسب است که برنامه به طور دقیق نیازهای خود را مشخص کرده باشد و نیاز به تخصیص و انتقال پویا در حین اجرای برنامه نداشته باشد.

سیستم عامل را می‌توان در آدرس بالای حافظه یا در آدرس پایین حافظه قرار داد. عامل مهمی که در این تصمیم‌گیری نقش دارد، محل بردار وقفه ( Interrupt vector-table) است. هر سطح وقفه یک محل رزرو شده در حافظه دارد که بردار وقفه نامیده می‌شود. چون معمولاً بردار وقفه در آدرس پایین حافظه قرار دارد، برنامه نویسان معمولاً سیستم عامل را نیز در آدرس پایین حافظه قرار می‌دهند.

## تخصیص حافظه (Memory Allocation)

یکی از ساده ترین روش ها برای تخصیص حافظه این است که حافظه را به چندین بخش (پارتیشن) با اندازه ی ثابت تقسیم شود. در هر بخش ممکن است دقیقا یک فرایند قرار گیرد. لذا درجه ی چند برنامه ای توسط تعداد بخش ها محدود می شود. اگر در این روش چند بخشی، یک بخش از حافظه خالی باشد، فرایندی که از صف ورودی انتخاب می شود و در آن بخش خالی قرار میگیرد. وقتی فرایندی خاتمه می یابد، بخشی از حافظه که در اختیار آن است آزاد می شود و فرایند دیگری می تواند در آن قسمت قرار گیرد

در طرح بخش هایی با اندازه های متفاوت، سیستم عامل، جدولی را تشکیل می دهد که مشخص می کند چه بخش هایی از حافظه، آزاد و چه بخش هایی اشغال هستند. در آغاز کل حافظه برای فرایندهای کاربر مهیا است و به عنوان یک بلوک حافظه ی بزرگ به نام حفره (Hole) در نظر گرفته می شود. سرانجام حافظه شامل مجموعه ای از حفره ها با اندازه های مختلف خواهد بود

وقتی فرایندها وارد سیستم می شوند، در یک صف ورودی قرار می گیرند. سیستم عامل نیازمندی های حافظه ی مربوط به هر فرایند و فضای موجود را در نظر می گیرد و مشخص می کند به کدام فرایند ها ، حافظه تخصیص دهد. وقتی به فرایندی حافظه تخصیص می یابد. به حافظه بار می شود و می تواند برای دریافت چرخه های پردازنده رقابت کند . وقتی فرایندی خاتمه می یابد. حافظه اش را آزاد می کند و سیستم عامل می تواند آن را به فرایند دیگر موجود در صف ورودی اختصاص دهد.

در هر زمان، لیستی از اندازه ی بلوک آزاد و صف ورودی داریم. سیستم عامل صف ورودی را با استفاده از یک الگوریتم زمانبندی مرتب می کند. حافظه زمانی به فرایند ها تخصیص می یابد که حافظه ی آزاد باقی مانده نتواند نیازمندی فرایند بعدی را بر آورده کند. یعنی هیچ بلوکی از حافظه موجود(حفره) برای نگهداری فرایند کافی نباشد. سیستم عامل می تواند منتظر بماند تا یک بلوک حافظه به اندازه ی کافی آزاد شود، یا در صف ورودی جستجو کند تا ببیند آیا فرایندی وجود دارد که نیازمندی حافظه ی آن کمتر باشد یا خیر و در صورت وجود آن را انتخاب نماید.

در هر زمان مجموعه ای از حفره ها با اندازه ی مختلف در سراسر حافظه وجود دارد.

وقتی فرایندی از صف ورودی انتخاب می شود و نیاز به حافظه دارد، در این مجموعه جستجو می کنیم تا یک حفره ی به اندازه کافی پیدا کنیم.

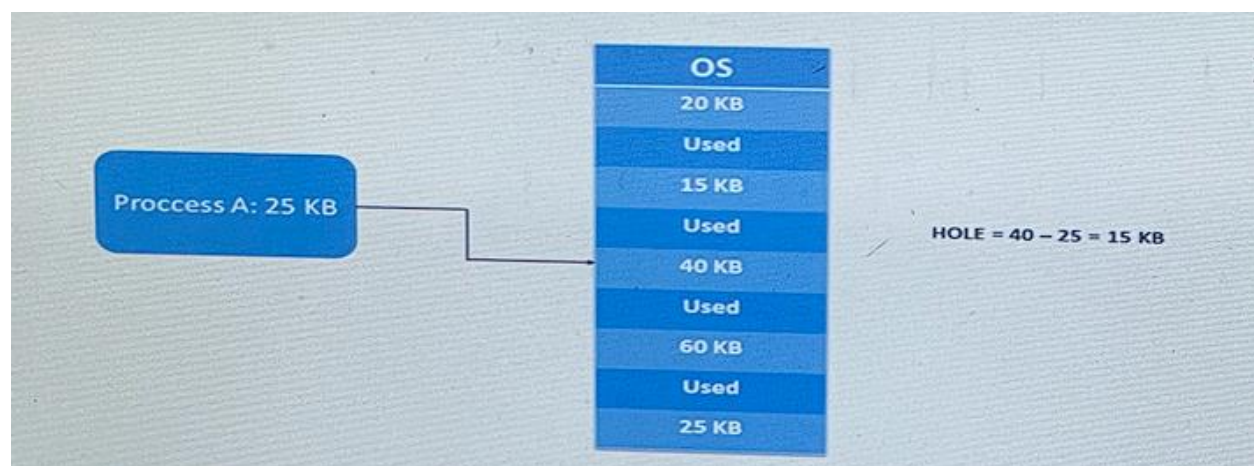
اگر این حفره، بزرگ تر از حافظه ی مورد نیاز برای فرایند باشد به دو بخش تقسیم میشود:

یک بخش به فرایند تخصیص می یابد و بخش دیگر به این مجموعه از حفره ها بر می گردد.

وقتی فرایندی خاتمه می یابد. بلوک حافظه ی خود را آزاد می کند تا به مجموعه ی حفره ها برگردد. اگر حفره ی جدید همجوار حفره ی های دیگری باشد، این حفره های همجوار را ادغام می کنیم تا حفره ی بزرگ تری ایجاد شود. در این نقطه، ممکن است لازم باشد بررسی کنیم آیا فرایند هایی منتظر حافظه هستند؟ و آیا حافظه ی آزاد شده ای که ادغام شده است می تواند تقاضای فرایند های منتظر را برآورده کند؟ یا خیر.

این رویه، حالت خاصی از مساله ی تخصیص حافظه ی پویا است که مشخص می کند که با استفاده از لیستی از حفره های خالی، چگونه می تواند به درخواستی به اندازه  $n$  پاسخ دهد. راه حل های گوناگونی برای این مساله وجود دارد. متداول ترین راهبرد هایی که برای انتخاب یک حفره ی آزاد از مجموعه ای از حفره ها به کار می روند. عبارت ان از : اولین برازش، بهترین برازش، بدترین برازش و برازش بعدی

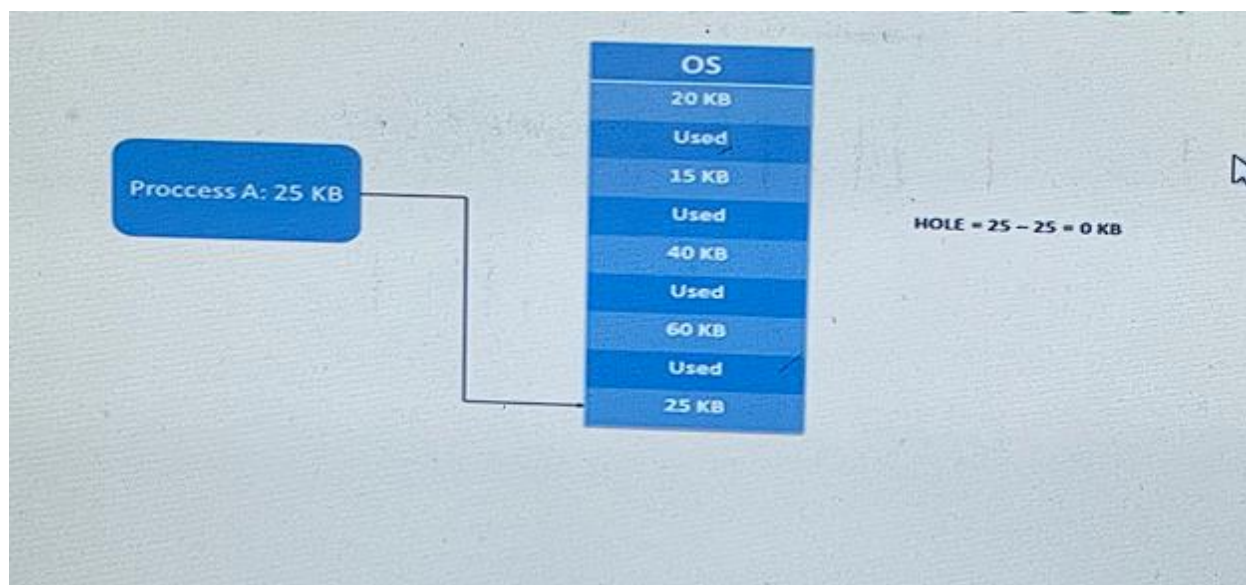
## اولین برازش (First-fit)



در این روش، اولین حفره ای که بتواند به فرایند تخصیص یابد، انتخاب می شود. جستجو می تواند از ابتدای مجموعه ای از حفره های آزاد انجام شود یا می تواند از جایی آغاز شود که جستجو قبلی برای اولین برازش، در آن جا خاتمه یافته است. با یافتن اولین حفره ی که فضای کافی برای فرایند داشته باشد، جستجو خاتمه می یابد.

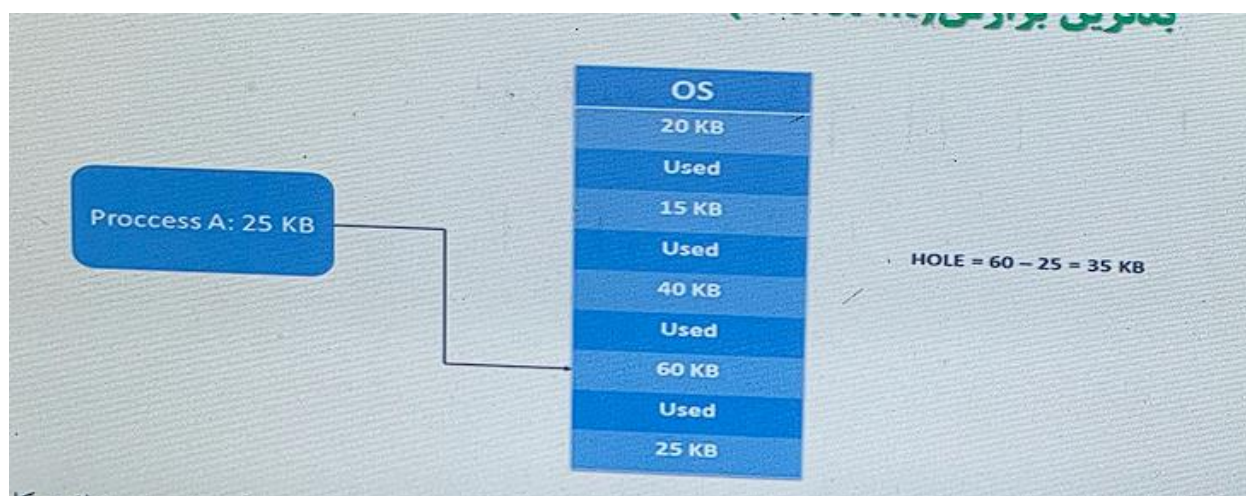


## بهترین برازش (Best-fit)



در این روش کوچک ترین حفره ای که بتواند فرایند را در خود جای دهد تخصیص می یابد. اگر لیست بر حسب اندازه ی حفره ها مرتب نباشد کل لیست باید جستجو شود. این راهبر کوچک ترین حفره ها را باقی می گذارد.

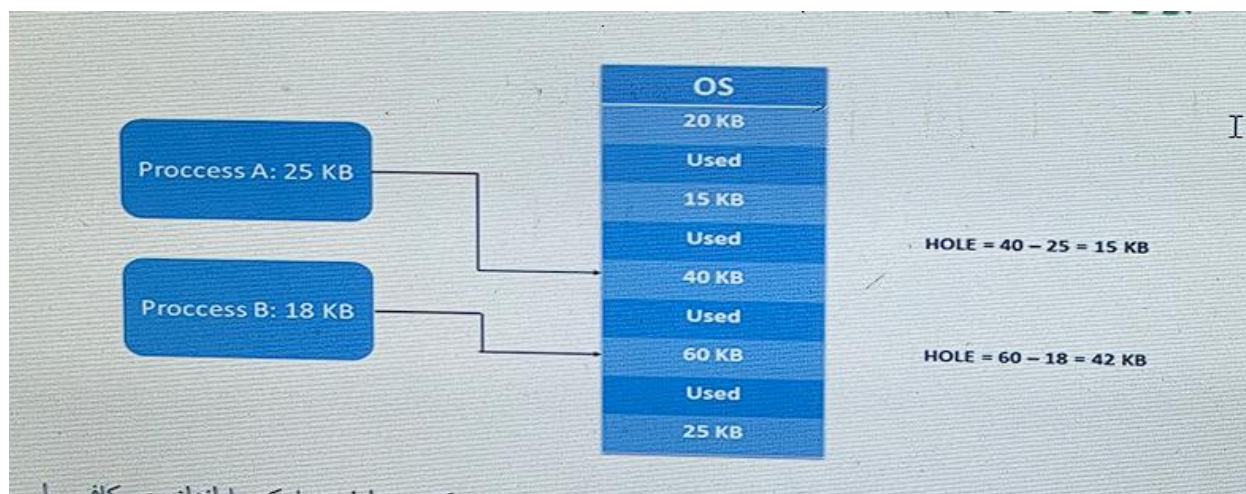
## بدترین برازش (Worst-fit)



در این روش بزرگ ترین حفره انتخاب می شود. اگر لیست بر حسب اندازه ی حفره ها مرتب نباشد، کل لیست باید جستجو شود. این راهبرد بزرگ ترین حفره ها را باقی می گذارد که ممکن است نسبت به کوچک ترین حفره ها که در روش "بهترین برازش" باقی می ماند مفیدی تر باشند

روش های اولین و بهترین برازش، بر اساس بهره وری از فضای حافظه و کاهش زمان، از روش بدترین برازش بهتر است. اولین برازش و بهترین برازش از نظر بهره وری حافظه بر دیگری ترجیح ندارد ولی روش بهترین برازش سریع تر است

## برازش بعدی (Next-fit)



برازش بعدی، حافظه را از مکان آخرین جاگذاری به بعد، مرور می کند و اولین بلوک با اندازه ی کافی را انتخاب می کند.

الگوریتم اولین برازش نه تنها ساده ترین، بلکه معمولاً بهترین و سریع ترین نیز است. نتیجه الگوریتم "برازش بعدی"، کمی بدتر از نتایج اولین برازش است.



الگوریتم برازش بعدی غالباً منجر به تخصیص بلوک های آزاد آخر حافظه می شود. نتیجه اش این است که بزرگ ترین بلوک از حافظه ی آزاد، که در انتهای فضای حافظه ظاهر می شود، سریعاً به تکه های کوچک تقسیم شود. لذا در الگوریتم برازش بعدی، ممکن است فشردن سازی بیشتر تکرار شود. از طرف دیگر ، الگوریتم بهترین برازش ، بلوک های ابتدای حافظه را به تکه های کوچکی تقسیم می کند که بارها باید جستجو شوند. الگوریتم بهترین برازش ، بر خلاف نامش معمولاً بدترین کارایی را دارد. چون این الگوریتم برای برآورده کردن نیاز، کوچک ترین بلوک ممکن را جستجو می کند. تضمین می شود که تکه ی باقی مانده کوچک باشد. در این روش گرچه هر درخواست حافظه همیشه کوچک ترین مقدار حافظه را به هدر می دهد، نتیجه اش این است که حافظه ی اصلی سریعاً به بلوک های کوچکی تقسیم شوند که نمی توانند به درخواست های تخصیص حافظه پاسخ دهند.

لذا در این روش ، نسبت به روش های دیگر فشردن سازی به دفعات بیشتری تکرار می شود

## تکه تکه شدن

هر دو راهبرد اولین برازش و بهترین برازش برای تخصیص حافظه، منجر به تکه تکه شدن خارجی (External Fragmentation) می شوند. وقتی فرایند ها به حافظه بار می شوند و از حافظه حذف می شوند، فضای حافظه به تکه های کوچک تقسیم میشود.

تکه تکه شدن خارجی وقتی به وجود می آید که حافظه ی کافی برای پاسخگویی به یک درخواست وجود دارد ولی کل این حافظه همجوار نیست . یعنی حافظه به تعداد زیادی از حفره های کوچک تقسیم شده است که همجوار نیستند.

این مسأله ی تکه تکه شدن می تواند جدی باشد. در بدترین حالت ، می توانیم بین هر دو فرایند، یک بلوک آزاد(یا به هدر رفته) داشته باشیم. اگر کل این حافظه در یک بلوک آزاد بزرگ باشد، ممکن است بتوانیم چنین فرایند را جا دهیم و اجرا کنیم.

بسته به میزان کل حافظه و میانگین اندازی فرایندها، تکه تکه شدن خارجی می تواند مساله ی مهم یا ناچیزی باشد. به عنوان مثال، تحلیل آماری اولین برازش نشان می دهد که حتی با بهینه سازی ممکن است به ازای تخصیص  $n$  بلوک،  $0.5N$  بلوک به دلیل تکه تکه شدن به هدر می رود. این ویژگی را قاعده ی 50 درصد می نامند

تکه تکه شدن حافظه علاوه بر خارجی بودن، می تواند به صورت داخلی نیز باشد. طرح تخصیص چند قسمتی را در نظر بگیرید که حفره ای به اندازه ی 18,464 بایت دارد. فرض کنید فرایند بعدی 18,462 بایت را درخواست کند، اگر دقیقاً بلوک درخواست شده را تخصیص دهیم، یک حفره ی 2 بایتی خالی می ماند. بدیهی است که هزینه نگهداری این حفره گران تر از خود حفره است

رویکرد کلی برای اجتناب از این مساله تقسیم حافظه فیزیکی به بلوک هایی با اندازه ی ثابت و تخصیص حافظه بر حسب واحد های مبتنی بر اندازه ی بلوک است.

بدین ترتیب، حافظه ای که تخصیص می یابد ممکن است کمی بیش از حافظه ی درخواستی باشد.

تفاوت بین حافظه ی درخواستی و حافظه تخصیص یافته را تکه تکه شدن داخلی (internal fragmentation) می گویند یعنی حافظه ای که در داخل یک بخش از حافظه است، ولی مورد استفاده قرار نمی گیرد.

یک راه حل برای مساله ی تکه تکه شدن خارجی، فشردن سازی (compaction) است.

در این روش، حفره های کوچک را با یکدیگر ادغام می شوند تا یک بلوک بزرگ از حافظه ایجاد شود. اما فشردن سازی همیشه، ممکن نیست. اگر جا به جایی به صورت استا باشد و در زمان اسمبل کردن یا در زمان باز کردن انجام شود فشردن سازی امکان پذیر نیست. یعنی فشردن سازی فقط وقتی ممکن است که جا به جایی به صورت پویا در زمان اجرا صورت گیرد. اگر آدرس ها به طور پویا جابه جا شوند، جا به جایی مستلزم این است که برنامه وداده ها به محل جدید منتقل شوند و سپس ثبات پایه تغییر کند تا آدرس های پایه ی جدید منعکس شود. وقتی فشردن سازی ممکن باشد، هزینه ی آن باید محاسبه شود

ساده ترین الگوریتم فشردن سازی این است که تمام فرایندها به یک طرف حافظه منتقل شوند، یعنی تمام حفره ها به یک طرف می روند تا حفره ی بزرگی از حافظه آزاد را تشکیل دهند. این طرح می تواند بسیار گران باشد.

راه حل دیگر مساله ی تکه تکه شدن خارجی این است که اجازه دهیم فضای آدرس منطقی فرایندها همجوار نباشد. بدین ترتیب هر جایی از حافظه فیزیکی که آزاد شد، به فرایند تخصیص یابد.

دو تکنیک برای این راه حل وجود دارد که مکمل یکدیگر هستند:

قطعه بندی و صفحه بندی

این تکنیک ها می توانند با هم ترکیب شوند