سیستم عامل جلسه ششم

سلسله مراتب حافظه Storage hierarchy

بین سه ویژگی کلیدی حافظه یعنی، هزینه (Price)، ظرفیت (capacity) ، زمان دسترسی (access time) باید سبک و سنگین کرد. برای این کار نمی‌توان بر یک حافظه یا فن آوری خاصی تکیه کرد و باید از سلسله مراتب حافظه استفاده کرد

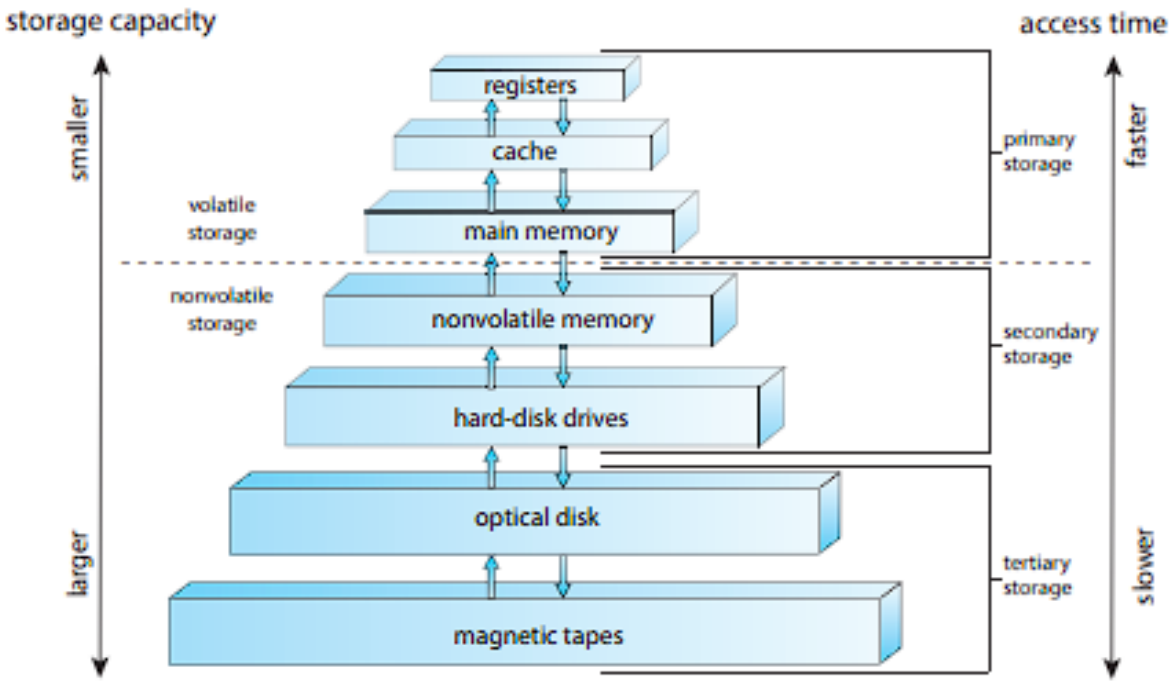


Figure 1.6 Storage-device hierarchy.

سلسله مراتب متدوال حافظه:

1. ثبات‌ها (Register)
2. حافظه نهان (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 پایین‌تر این سلسله مراتب شرایط زیر رخ می‌دهد:

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 حافظه‌های زیر در مقابل آن نوشته شده است:

ثبات‌ها (Register): کامپایلر

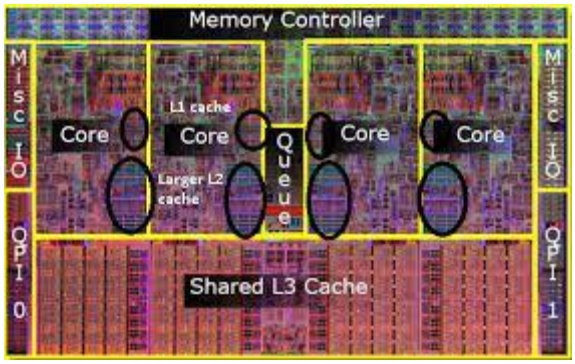
حافظه پنهان (cache): خودکار (توسط خود پردازنده)

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

حافظه پنهان (cache)

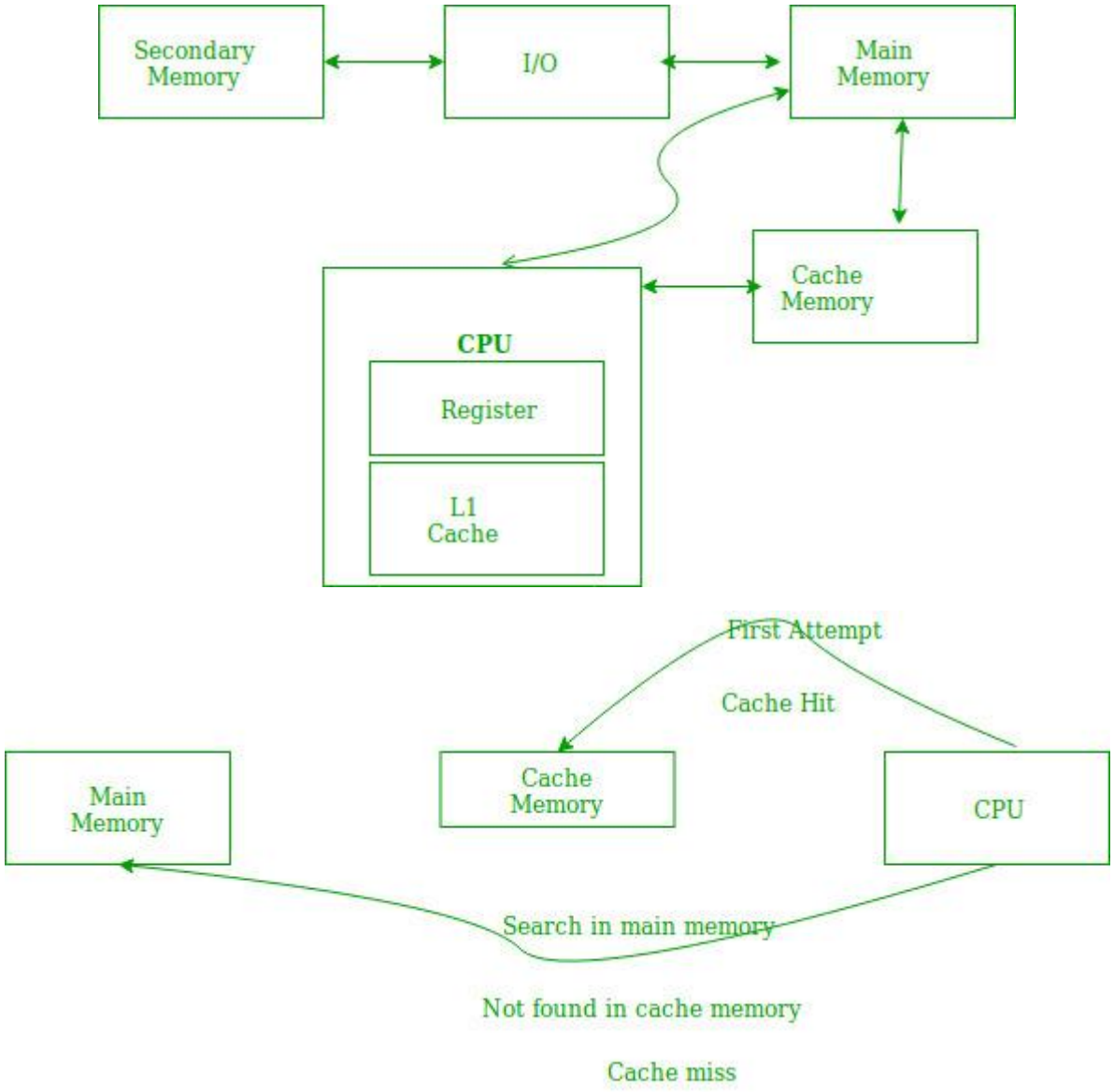
A CPU cache is a hardware cache used by the central processing unit (CPU) of a computer to reduce the average cost (time or energy) to access data from the main memory. A cache is a smaller, faster memory, located closer to a processor core, which stores copies of the data from frequently used main memory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

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



وقتی پردازنده می‌خواهد کلمه ای (Word) (32 or 64 bit) از حافظه را بخواند وجود آن را در حافظه‌ی پنهان بررسی می‌کند. اگر وجود داشته باشد به پردازنده تحویل داده می‌شود در غیر این صورت یک بلوک از حافظه اصلی شامل تعداد ثابتی از خانه‌های حافظه به حافظه پنهان (cache) منتقل می‌شود و سپس کلمه مورد نظر به پردازنده تحویل داده می‌شود. هنگامی که یک بلوک از داده‌ها به حافظه پنهان آورده می‌شود تا یک مراجعه به حافظه انجام شود، به دلیل پدیده‌ی محلی بودن مراجعات، احتمالاً به زودی به دیگر کلمات آن بلوک نیز مراجعه خواهد شد.

(Locality of Reference and Cache Operation in Cache Memory)



در یک سیستم کامپیوتری، برای افزایش کارایی از حافظه‌های چند سطحی استفاده می‌کنند به گونه ای که سطوح نزدیک‌تر 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 می‌شود.

دلیل آن این است که بر اساس اصل محلی گرایی (Locality of Reference) گفته می‌شود هر گاه داده‌هایی مورد استفاده قرار گیرند به زودی در آینده نیز مورد استفاده خواهد بود. قرار دادن این داده‌ها و دستورالعمل‌ها در حافظه پنهان موجب افزایش سرعت دستیابی می‌شود بنابراین زمانی که داده یا دستورالعملی نیاز باشد ابتدا به حافظه پنهان مراجعه می‌شود و در صورت وجود آن، استفاده می‌شود.در غیر این صورت به حافظه اصلی مراجعه شده و داده‌ها و دستوراعمل های مورد نظر، مورد استفاده قرار می‌گیرند.

از آن جایی که ظرفیت حافظه پنهان نسبت به حافظه اصلی بسیار کمتر است بنابراین نخواهیم توانست همه داده‌هایی را که در حافظه اصلی هستند را به حافظه پنهان ببریم پس ممکن است گاهی به داده‌هایی 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 الگوریتم‌های جایگزینی حافظه استفاده می‌شود که دقیقاً همان الگوریتم‌های جایگزینی صفحه در حافظه اصلی هستند

یادداشت:

صفحه (Page) در حافظه اصلی (RAM) به عنوان یک بخشی از فضای آدرس‌پذیر 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 اصلی‌ترین وظایف سیستم عامل است

سخت افزایه پایه (Base Hardware)

حافظه اصلی (Memory) و ثبات‌های ساخته شده در خود پردازنده (Register) ، تنها فضای ذخیره سازی همه منظوره ای هستند که پردازنده مستقیماً می‌تواند به آن‌ها دسترسی داشته باشد

هر دستور العمل در حال اجرا و داده ای که توسط آن استفاده می‌شود، باید در یکی از این دو دستگاه ذخیره سازی با دستیابی مستقیم (حافظه‌ی اصلی و ثبات‌ها) واقع باشد.

انقیاد آدرس (Address Binding)

معمولاً برنامه بر روی دیسک به صورت یک فایل اجرایی دودویی ذخیره می‌شود. 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 می‌گیرد. این فرایند ممکن است در حین اجرا بین دیسک و حافظه انتقال یابد (Swapping) . فرایندهای موجود در دیسک که منتظرند وارد حافظه و اجرا شوند، صف ورودی را تشکیل می‌دهند.

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

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

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

برنامه‌ی کاربر هیچ گاه آدرس‌های فیزیکی واقعی را نمی‌بیند.

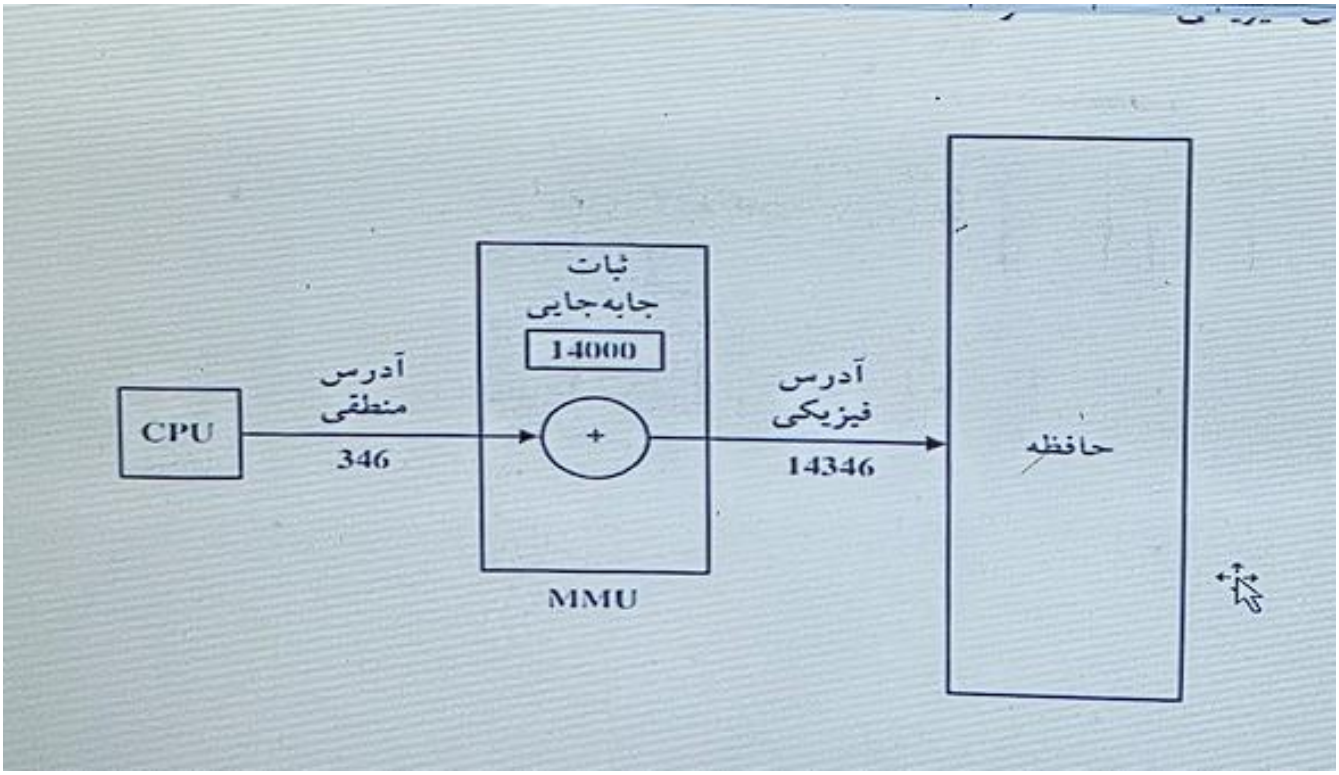
سخت افزار نگاشت حافظه، آدرس‌های منطقی را به آدرس‌های فیزیکی تبدیل می‌کند

دو نو آدرس وجود دارد

آدرس‌های منطقی (از صفر تا max)

آدرس‌های فیزیکی (از R + 0 تا max + R با مقدار پایه‌ی R)

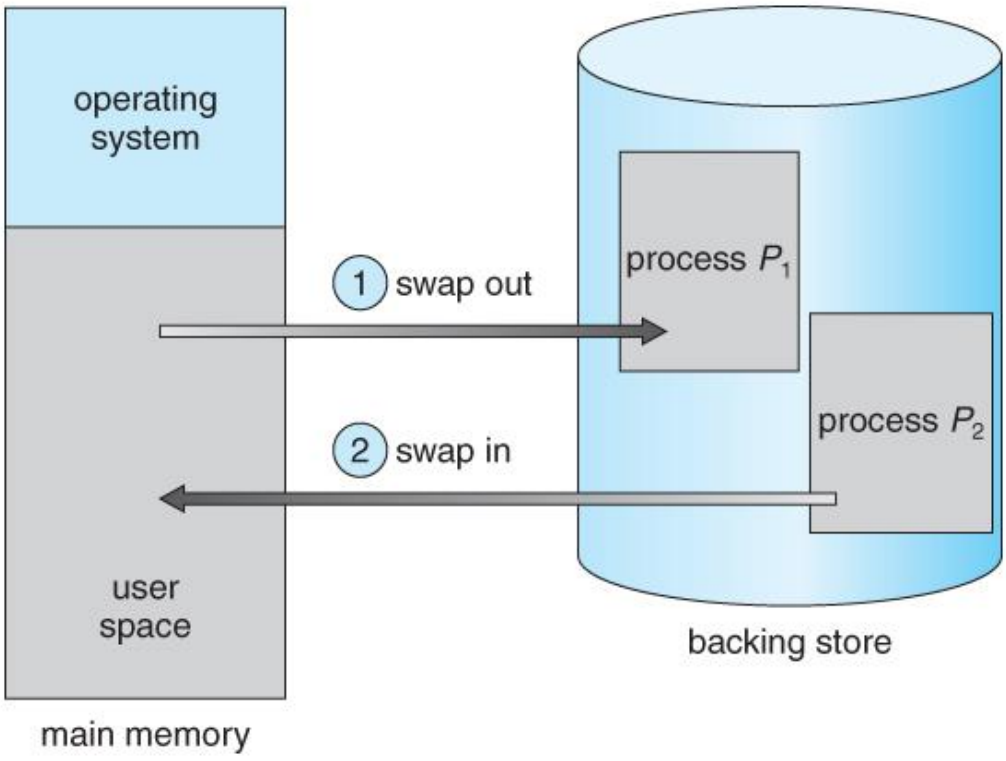
کاربر فقط آدرس‌های منطقی را تولید می‌نماید و فکر می‌کند که فرایند در محل‌های صفر تا max اجرا می‌شود. برنامه کاربر، آدرس‌های منطقی را تولید می‌کند و این آدرس‌ها قبل از بکارگیری باید به آدرس‌های فیزیکی نگاشت شوند.

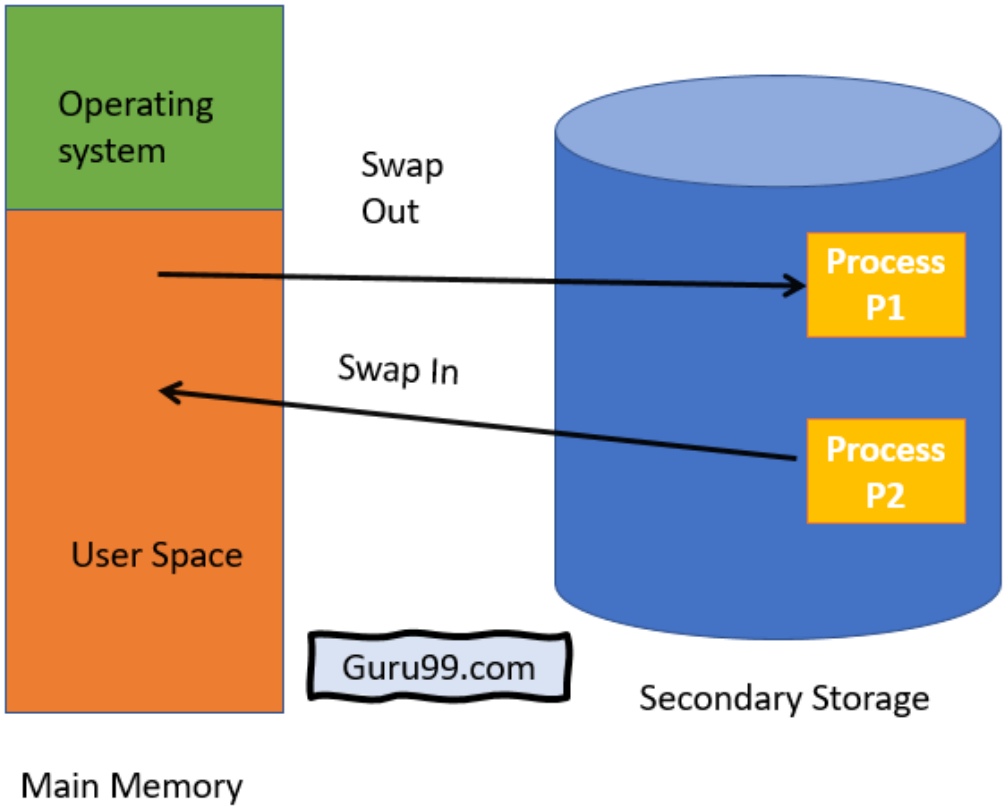


مبادله (Swapping)

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

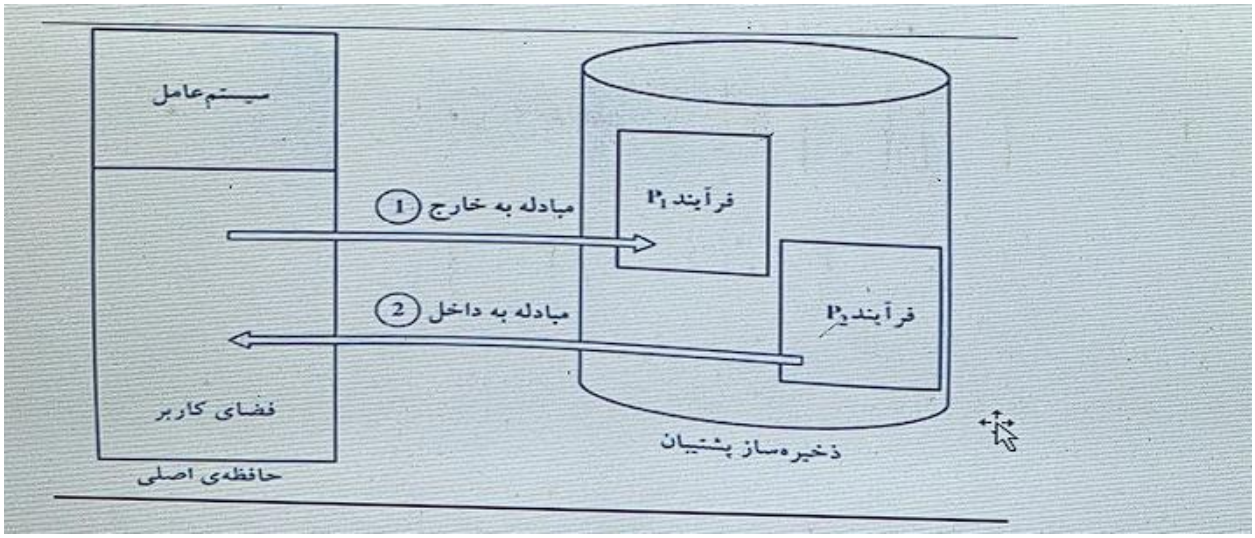
Swapping is a technique used in operating systems to temporarily move pages or entire processes from the main memory (RAM) to secondary storage devices, such as hard disks, to free up space in the main memory. When the system needs to access the data that has been swapped out, it swaps the less frequently 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.


operating system
1 swap out
process P_1
process P_2
2 swap in
user space
backing store
main memory


Operating system
Swap Out
Process P1
Swap In
Process P2
User Space
Guru99.com
Secondary Storage
Main Memory

مبادله استاندارد

مبادله‌ی استاندارد شامل انتقال فرایند بین حافظه‌ی 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 حافظه‌ی همجوار (Contiguous Memory allocation)

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

حافظه معمولاً به دو قسمت تقسیم می‌شود:

1.یک قسمت برای سیستم عامل مقیم

2.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 تصمیم گیری 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 ساده‌ترین روش‌ها برای تخصیص حافظه این است که حافظه را به چندین بخش (پارتیشن) با اندازه‌ی ثابت تقسیم شود. در هر بخش ممکن است دقیقاً یک فرایند قرار گیرد. لذا درجه‌ی چند برنامه ای توسط تعداد بخش‌ها محدود می‌شود. اگر در این روش چند بخشی، یک بخش از حافظه خالی باشد، فرایندی که از صف ورودی انتخاب می‌شود و در آن بخش خالی قرار می‌گیرد. وقتی فرایندی خاتمه می‌یابد، بخشی از حافظه که در اختیار آن است ازاد می‌شود و فرایند دیگری می‌تواند درآن قسمت قرار گیرد

در طرح بخش‌هایی با اندازه‌های متفاوت، سیستم عامل، جدولی را تشکل می‌دهد که مشخص می‌کند چه بخش‌هایی از حافظه، آزاد و چه بخش‌هایی اشغال هستند. در آغاز کل حافظه برای فرایندهای کاربر مهیا است و به عنوان یک بلوک حافظه‌ی بزرگ به نام حفره (Hole) در نظر گرفته می‌شود. سرانجام حافظه شامل مجموعه ای از حفره‌ها با اندازه‌های مختلف خواهد بود

وقتی فرایندها وارد سیستم می‌شوند، در یک صف ورودی قرار می‌گیرند. سیستم عامل نیازمندی‌های حافظه‌ی مربوط به هر فرایند و فضای موجود را در نظر می‌گیرد و مشخص می‌کند به کدام فرایندها، حافظه تخصیص دهد. وقتی به فرایندی حافظه تخصیص می‌یابد. به حافظه بار می‌شود و می‌تواند برای دریافت چرخه‌های پردازنده رقابت کند . وقتی فرایندی خاتمه می‌یابد. حافظه اش را آزاد می‌کند و سیستم عامل می‌تواند آن را به فرایند دیگر موجود در صف ورودی اختصاص دهد.

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

در هر زمان مجموعه ای از حفره‌ها با اندازه‌ی مختلف در سراسر حافظه وجود دارد.

وقتی فرایندی از صف ورودی انتخاب می‌شود و نیاز به حافظه دارد، در این مجموعه جستجو می‌کنیم تا یک حفره‌ی به اندازه کافی پیدا کنیم.

اگر این حفره، بزرگ‌تر از حافظه‌ی مورد نیاز برای فرایند باشد به دو بخش تقسیم می‌شود:

یک بخش به فرایند تخصیص می‌یابد و بخش دیگر به این مجموعه از حفره‌ها بر می‌گردد.

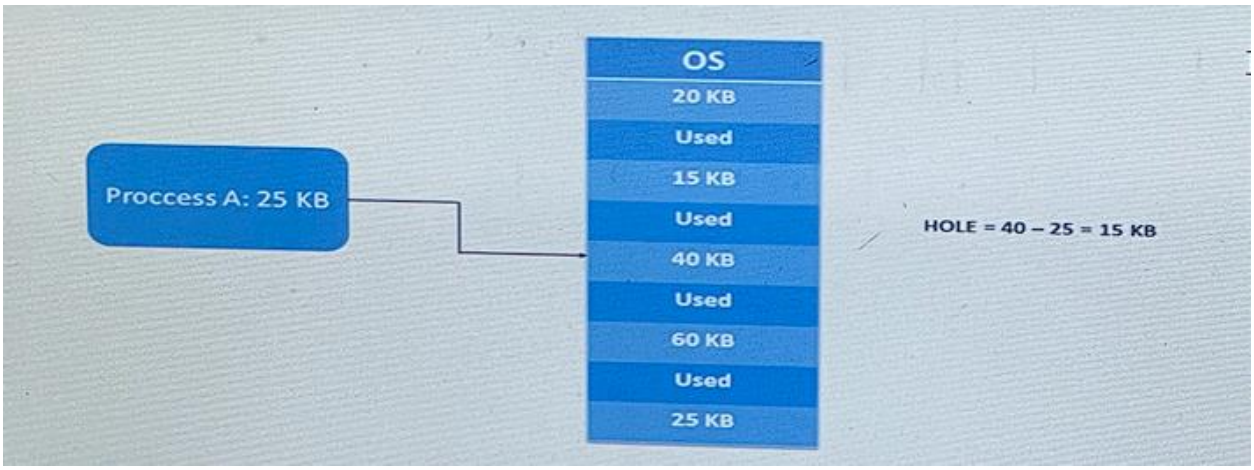
وقتی فرایندی خاتمه می‌یابد. بلوک حافظه‌ی خود را آزاد می‌کند تا به مجموعه‌ی حفره‌ها برگردد. اگر حفره‌ی جدید همجوار حفره‌ی های دیگری باشد، این حفره‌های همجوار را ادغام می‌کنیم تا حفره‌ی بزرگ‌تری ایجاد شود. در این نقطه، ممکن است لازم باشد بررسی کنیم ایا فرایندهایی منتظر حافظه هستند؟ و ایا حافظه‌ی ازاد شده ای که ادغام شده است می‌تواند تقاضای فرایندهای منتظر را براورده کند؟ یا خیر.

این رویه، حالت خاصی از مساله ی تخصیص حافظه‌ی پویا است که مشخص می‌کند که با استفاده از لیستی از حفره‌های خالی، چگونه می‌تواند به درخواستی به اندازه n پاسخ

دهد. راه حل‌های گوناگونی برای این مساله وجود دارد. متداول‌ترین راهبردهایی که برای انتخاب یک حفره‌ی آزاد از مجموعه ای از حفره‌ها به کار می‌روند. عبارت ان از:

اولین برازش First-fit ، بهترین برازش Best-Fit ، بدترین برازش Worst-Fit و برازش بعدی Next-Fit

اولین برازش (First-fit)

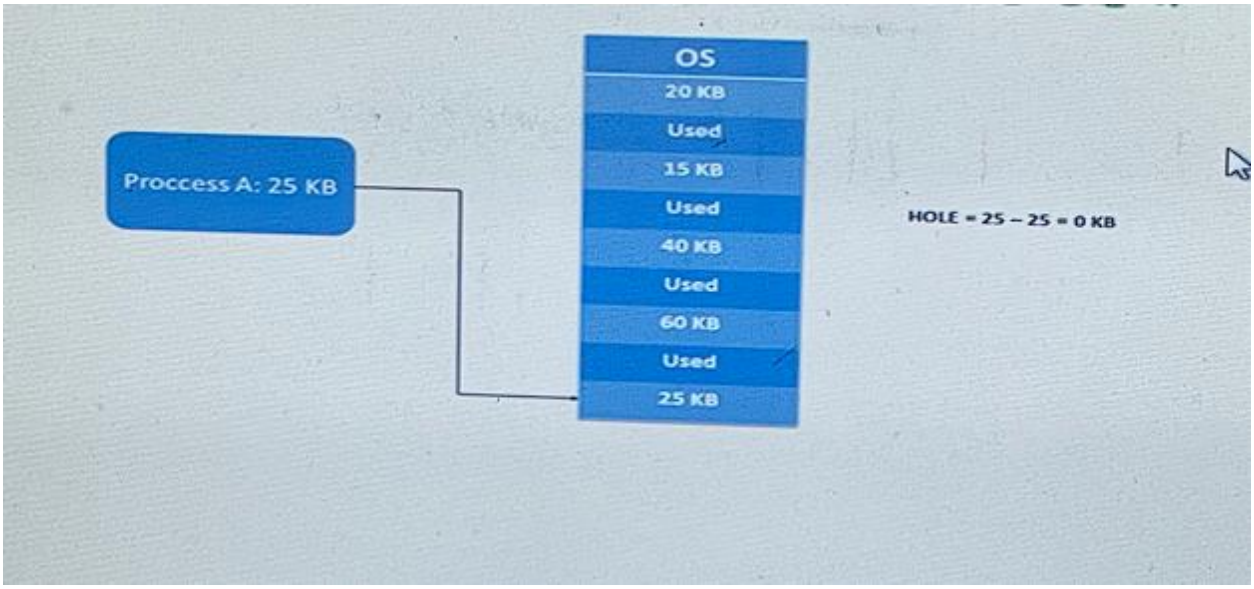


در این روش، اولین حفره ای که بتواند به فرایند تخصیص یابد، انتخاب می‌شود.

جستجو می‌تواند از ابتدای مجموعه ای از حفره‌های آزاد انجام شود یا می‌تواند از جایی آغاز شود که جستجو قبلی برای اولین برازش، در آن جا خاتمه یافته است.

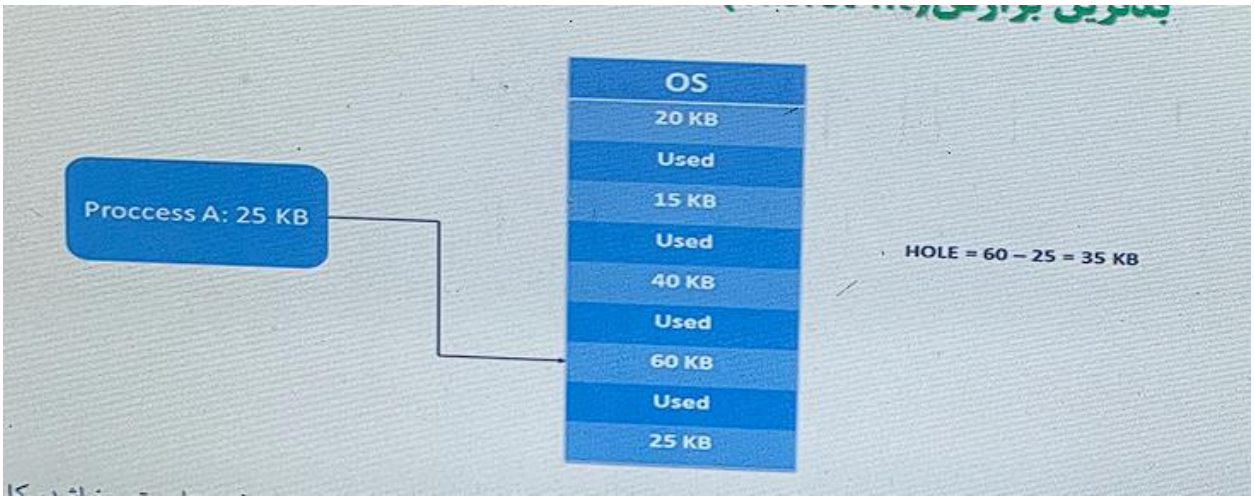
با یافتن اولین حفره‌ی که فضای کافی برای فرایند داشته باشد، جستجو خاتمه می‌یابد.

بهترین برازش (Best-fit)



در این روش کوچک‌ترین حفره ای که بتواند فرایند را در خود جای دهد تخصیص می‌یابد. اگر لیست بر حسب اندازه‌ی حفره‌ها مرتب نباشد کل لیست **باید** جستجو شود. این راهبر کوچک‌ترین حفره‌ها را باقی می‌گذارد.

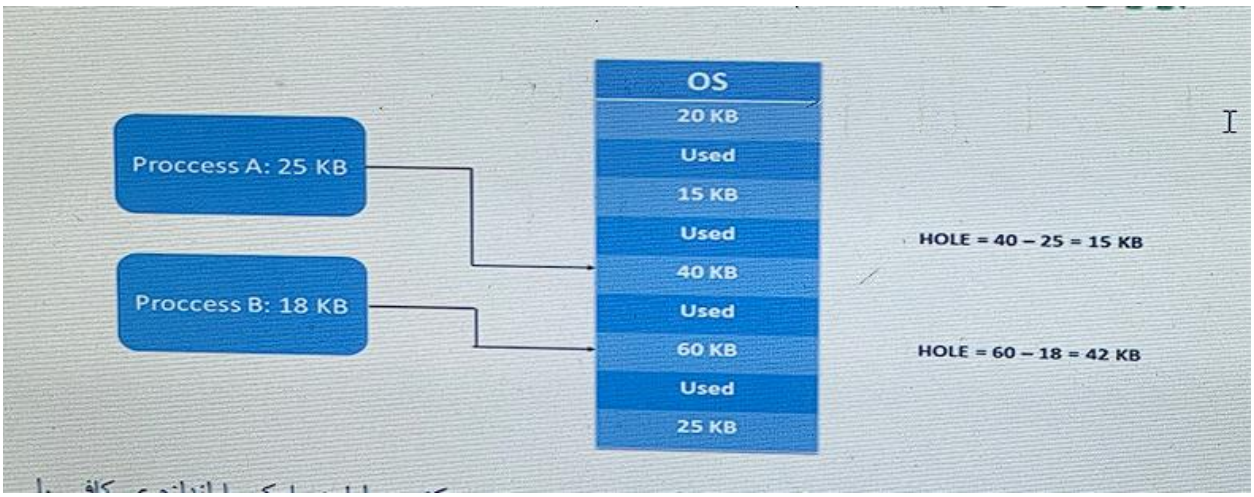
بدترین برازش (Worst-fit)



در این روش بزرگ‌ترین حفره انتخاب می‌شود. اگر لیست بر حسب اندازه‌ی حفره‌ها مرتب نباشد، کل لیست باید جستجو شود. این راهبرد بزرگ‌ترین حفره‌ها را باقی می‌گذارد که ممکن است نسبت به کوچک‌ترین حفره‌ها که در روش " بهترین برازش " باقی می‌مانند مفیدی تر باشند

روش‌های اولین و بهترین برازش First and best ، بر اساس بهره وری از فضای حافظه و کاهش زمان، از روش بدترین worst برازش بهتر است. اولین برازش و بهترین برازش از نظر بهره وری حافظه بر دیگری ترجیح ندارد ولی روش بهترین برازش سریع‌تر است

برازش بعدی (Next-fit)



برازش بعدی، حافظه را از مکان آخرین جاگذاری به بعد، مرور می‌کند و اولین بلوک با اندازه‌ی کافی را انتخاب می‌کند.

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

الگوریتم برازش بعدی غالباً منجر به تخصیص بلوک‌های آزاد آخر حافظه می‌شود. نتیجه اش این است که بزرگ‌ترین بلوک از حافظه‌ی آزاد، که در انتهای فضای حافظه ظاهر می‌شود، سریعاً به تکه‌های کوچک تقسیم شود. لذا در الگوریتم برازش بعدی، ممکن است فشرده سازی بیشتر تکرار شود. از طرف دیگر، الگوریتم بهترین برازش، بلوک‌های ابتدای حافظه را به تکه‌های کوچکی تقسیم می‌کند که بارها باید جستجو شوند. الگوریتم بهترین برازش، بر خلاف نامش معمولاً بدترین کارایی را دارد. چون این الگوریتم برای براورده کردن نیاز، کوچک‌ترین بلوک ممکن را جستجو می‌کند. تضمین می‌شود که تکه‌ی باقی مانده کوچک باشد. در این روش گرچه هر درخواست حافظه همیشه کوچک‌ترین مقدار حافظه را به هدر می‌دهد، نتیجه اش این است که حافظه‌ی اصلی سریعاً به بلوک‌های کوچکی تقسیم شوند که نمی‌توانند به درخواست‌های تخصیص حافظه پاسخ دهند.

لذا در این روش، نسبت به روش‌های دیگر فشرده سازی به دفعات بیشتری تکرار می‌شود

تکه تکه شدن

هر دو راهبرد اولین برازش و بهترین برازش برای تخصیص حافظه، منجر به تکه تکه شدن خارجی (External Fragmentation) می‌شوند. وقتی فرایندها به حافظه بار می‌شوند و از حافظه حذف می‌شوند، فضای حافظه به تکه‌های کوچک تقسیم می‌شود.

تکه تکه شدن خارجی وقتی به وجود می‌آید که حافظه‌ی کافی برای پاسخگویی به یک درخواست وجود دارد ولی کل این حافظه همجوار نیست. یعنی حافظه به تعداد زیادی از حفره‌های کوچک تقسیم شده است که همجوار نیستند.

این مسأله ی تکه تکه شدن می‌تواند جدی باشد. در بدترین حالت، می‌توانیم 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 می‌تواند مساله ی مهم یا ناچیزی باشد. به عنوان مثال، تحلیل آماری اولین برازش نشان می‌دهد که حتی با بهینه سازیِ ممکن است به ازای تخصیص n بلوک، 0.5N بلوک به دلیل تکه تکه شدن به هدر می‌رود. این ویژگی را قاعده‌ی 50 درصد می‌نامند

تکه تکه شدن حافظه علاوه بر خارجی بودن، می‌تواند به صورت داخلی نیز باشد. طرح تخصیص چند قسمتی را در نظر بگیرید که حفره ای به اندازه‌ی 18،464 بایت دارد. فرض کنید فرایند بعدی 18،462 باید را درخواست کند، اگر دقیقاً بلوک درخواست شده را تخصیص دهیم، یک حفره‌ی 2 بایتی خالی می‌ماند. بدیهی است که هزینه نگهداری این حفره گران‌تر از خود حفره است

رویکرد کلی برای اجتناب از این مساله تقسیم حافظه فیزیکی به بلوک‌هایی با اندازه‌ی ثابت و تخصیص حافظه بر حسب واحدهای مبتنی بر اندازه‌ی بلوک است.

بدین ترتیب، حافظه ای که تخصیص می‌یابد ممکن است کمی بیش از حافظه‌ی درخواستی باشد.

تفاوت بین حافظه‌ی درخواستی و حافظه تخصیص یافته را تکه تکه شدن داخلی (internal fragmentation) می‌گویند یعنی حافظه ای که در داخل یک بخش از حافظه است، ولی مورد استفاده قرار نمی‌گیرد.

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

در این روش، حفره‌های کوچک را با یکدیگر ادغام می‌شوند تا یک بلوک بزرگ از حافظه ایجاد شود. اما فشرده سازی همیشه، ممکن نیست. اگر جا به جایی به صورت استا باشد و در زمان اسمبل کردن یا در زمان باز کردن انجام شود فشده سازی امکان پذیر نیست. یعنی فشرده سازی فقط وقتی ممکن است که جا به جایی به صورت پویا در زمان اجرا صورت گیرد. اگر آدرس‌ها 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 آدرس‌های پایه‌ی جدید منعکس شود. وقتی فشرده سازی ممکن باشد، هزینه‌ی آن باید محاسبه شود

ساده‌ترین الگوریتم فشرده سازی این است که تمام فرایندها به یک طرف حافظه منتقل شوند، یعنی تمام حفره‌ها به یک طرف می‌روند تا حفره‌ی بزرگی از حافظه آزاد را تشکیل دهند. این طرح می‌تواند 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 تکنیک‌ها می‌توانند با هم ترکیب شوند