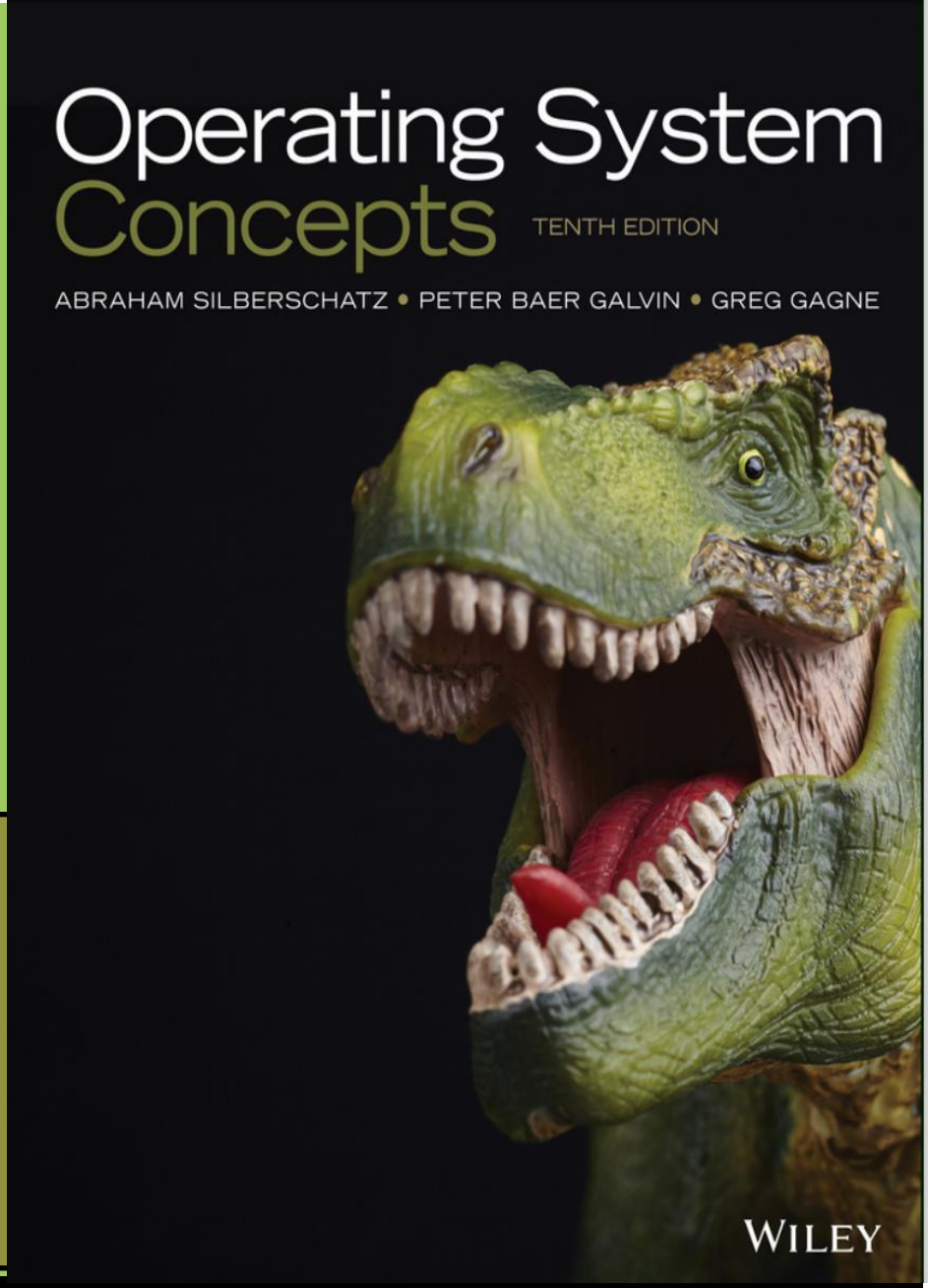
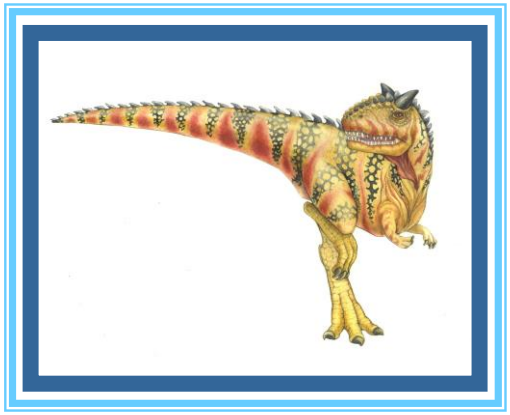
سیستم‌عامل

Dr. A. Taghinezhad



Website: ataghinezhad.github.io, Email:a0taghinezhad@gmail.com

فصل ۹: حافظه اصلی



فصل ۹: حافظه اصلی

- **پیش زمینه**
- تخصیص حافظه پیوسته
- صفحه‌بندی
- ساختار جدول صفحه
- جایگزینی (Swapping)
- مثال :معماری‌های ۳۲ و ۶۴ بیتی
- اینتل مثال :معماریARMv8

Objectives اهداف

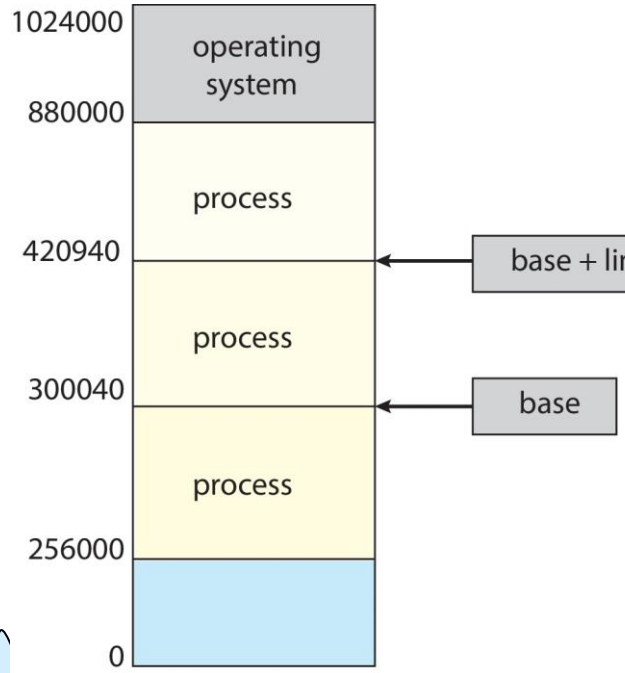
- این بخش به توصیف جزئیات روش‌های مختلف سازماندهی سخت‌افزار حافظه می‌پردازد
- همچنین به بحث در مورد تکنیک‌های مدیریت حافظه، توضیحات دقیق در مورد پردازنده اینتل پنتیوم که از هر دو حالت تقسیم‌بندی خالص و تقسیم‌بندی با صفحه‌بندی پشتیبانی می‌کند، می‌پردازد.

پیش زمینه

- برای اجرای یک برنامه، آن باید از دیسک به حافظه آورده شده و در فرآیندی قرار گیرد .حافظه اصلی و رجیسترها تنها حافظه‌هایی هستند که پردازنده می‌تواند به طور مستقیم به آن‌ها دسترسی داشته باشد .
- واحد حافظه تنها جریان‌های زیر را می‌بینید:
  - آدرسها + درخواست خواندن، یا
  - آدرس + داده و درخواست نوشتن
- دسترسی به رجیستر در یک سیکل ساعت پردازنده (یا کمتر) انجام می‌شود
- حافظه اصلی می‌تواند چندین سیکل طول بکشد و باعث توقف Stall پردازنده شود
  - کش بین حافظه اصلی و رجیسترهای پردازنده برای کمتر کردن توقف قرار می‌گیرد.
- برای اطمینان از عملکرد صحیح، حفاظت از حافظه ضروری است.

محافظت

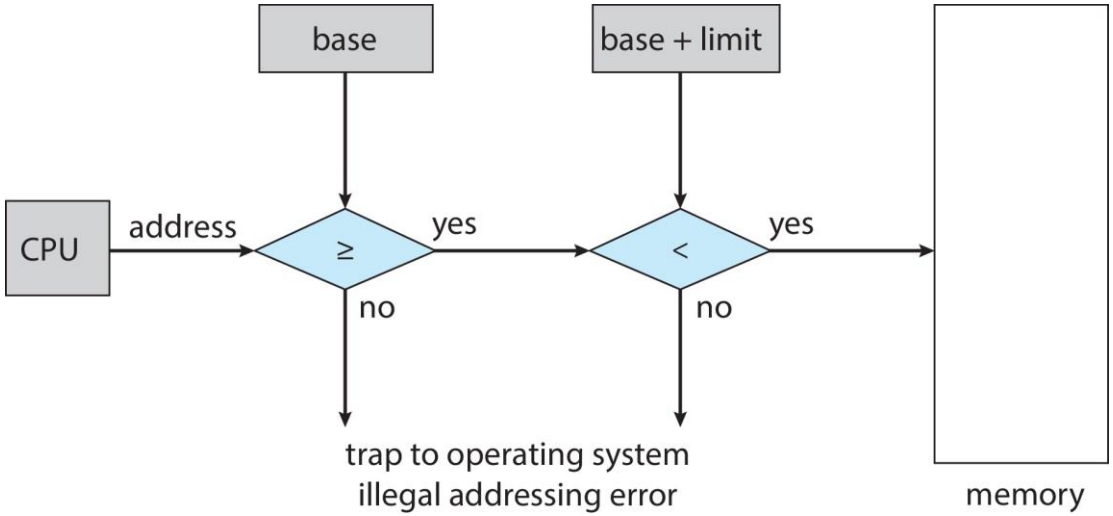
- باید مطمئن شویم که یک فرآیند فقط بتواند به آدرس‌های موجود در فضای آدرس خود دسترسی پیدا کند
- ما می‌توانیم این حفاظت را با استفاده از یک جفت رجیستر **پایه base** و **محدوده limit** ارائه دهیم که فضای آدرس منطقی یک فرآیند را تعریف می‌کنند.



- به عنوان مثال، اگر ثبات پایه ۳۰۰۰۴۰ داشته باشد و رجیستر محدود ۱۲۰۹۰۰ باشد، برنامه می تواند به طور قانونی به همه آدرس ها از ۳۰۰۰۴۰ تا ۴۲۰۹۳۹ دسترسی داشته باشد.

محافظت از آدرس سخت‌افزاری

- پردازنده باید هر دسترسی به حافظه که در حالت کاربر ایجاد می‌شود را بررسی کند تا مطمئن شود بین پایه و محدوده برای آن کاربر قرار دارد .



- دستورالعمل‌های مربوط به بارگذاری رجیسترهای پایه و محدوده دارای امتیاز هستند.

اتصال آدرس Address Bind

- برنامه‌های روی دیسک، آماده برای آمدن به حافظه برای اجرا، **یک صف ورودی** را تشکیل می‌دهند .
  - بدون پشتیبانی، باید در آدرس ۰۰۰۰ بارگذاری شوند
- اینکه آدرس فیزیکی اولین فرآیند کاربر همیشه در ۰۰۰۰ باشد، ناخوشایند است .
- آدرس‌ها در مراحل مختلف عمر برنامه به روش‌های مختلف نشان داده می‌شوند
  - آدرس‌های کد منبع معمولاً نمادین هستند .
  - آدرس‌های کد کامپایل شده به آدرس‌های قابل جابه‌جایی **متصل** می‌شوند
    - به عنوان مثال، "۱۴ بایت از ابتدای این ماژول
    - لینکر (linker) یا لودر (loader) آدرس‌های قابل جابه‌جایی relocatable addressesرا به آدرس‌های مطلق متصل می‌کنند
      - به عنوان مثال، ۷۴۰۱۴
      - هر اتصال binding یک فضای آدرس را به فضای دیگر نگاشت می‌کند.

اتصال دستورالعمل ها در حافظه

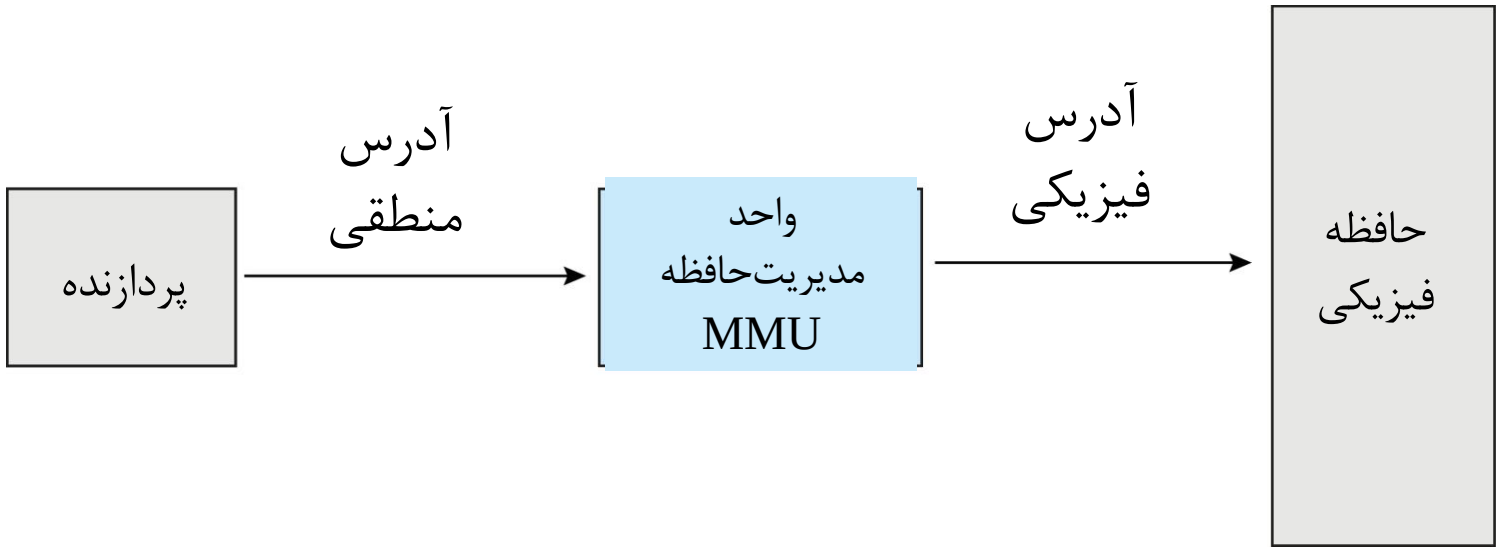
- اتصال یا bind آدرس دستورالعمل‌ها و داده‌ها به آدرس‌های حافظه می‌تواند در سه مرحله‌ی مختلف اتفاق بیفتد:
- **زمان کامپایل (تجمیع) :**اگر موقعیت حافظه از قبل مشخص باشد، کد مطلق (Absolute Code) قابل تولید است؛ اما اگر موقعیت شروع تغییر کند، کد نیاز به کامپایل مجدد دارد.
- **زمان لود یا بارگذاری :**اگر موقعیت حافظه در زمان کامپایل مشخص نباشد، باید کد قابل جابه‌جایی (Relocatable Code) تولید شود.
- **زمان اجرا :**اگر فرآیند در طول اجرا بتواند از یک بخش حافظه به بخش دیگر منتقل شود، اتصال آدرس تا زمان اجرا به تعویق می‌افتد .برای نگاشت‌های آدرس (مانند رجیسترهای پایه و محدوده) به پشتیبانی سخت‌افزاری نیاز است.

فضای آدرس فیزیکی و منطقی

- مفهوم فضای آدرس منطقی که به یک فضای آدرس فیزیکی مجزا متصل شده است، برای مدیریت صحیح حافظه ضروری است.
  - **آدرس منطقی (یا آدرس مجازی) :**توسط پردازنده تولید می‌شود.
  - **آدرس فیزیکی :**آدرسی که واحد حافظه می‌بیند.
- در الگوهای اتصال آدرس در زمان **کامپایل و بارگذاری**، آدرس‌های منطقی و فیزیکی **یکسان** هستند؛ در الگوهای اتصال آدرس در زمان اجرا، آدرس‌های منطقی (مجازی) و فیزیکی با هم تفاوت دارند.
- **فضای آدرس منطقی :**مجموعه‌ی تمام آدرس‌های منطقی تولید شده توسط یک برنامه است.
- **فضای آدرس فیزیکی :**مجموعه‌ی تمام آدرس‌های فیزیکی تولید شده توسط یک برنامه است.

واحد مدیریت حافظه (ادامه)

- یک سخت‌افزار در زمان اجرا، آدرس مجازی را به آدرس فیزیکی نگاشت می‌کند



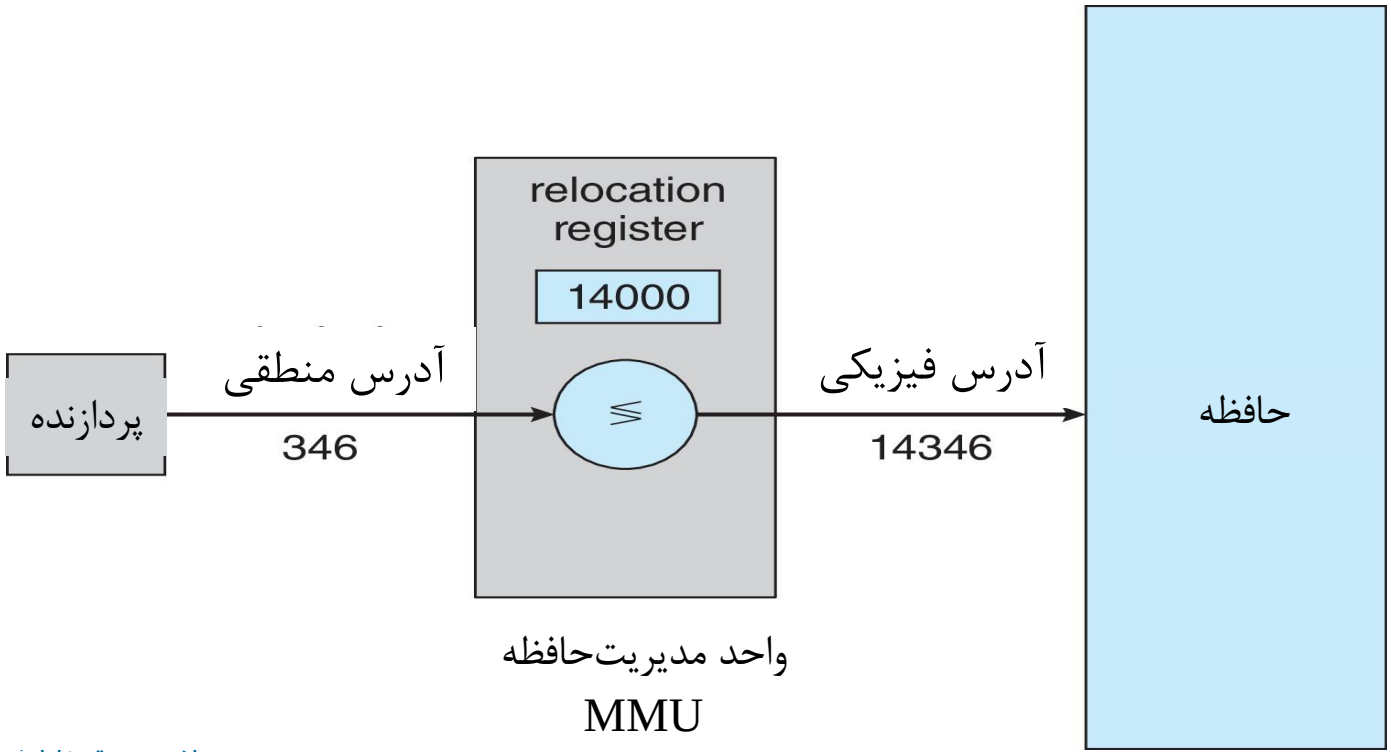
- روش‌های زیادی برای این کار وجود دارد که در ادامه‌ی این فصل به آن‌ها پرداخته می‌شود

واحد مدیریت حافظه (ادامه)

- **یک طرح ساده که تعمیمی از طرح رجیستر- پایه است را در نظر بگیرید:**
- در این طرح، رجیستر **پایه** اکنون رجیستر جابجایی **Relocation** Registerنامیده می‌شود
- مقدار موجود در **رجیستر جابجایی** به **هر آدرسی که توسط یک فرآیند** کاربر در زمان ارسال به حافظه تولید می‌شود، **اضافه می‌شود**.
- برنامه‌ی **کاربر** با آدرس‌های **منطقی** سروکار دارد و هرگز آدرس‌های **فیزیکی** واقعی را نمی‌بیند.
  - اتصال آدرس در زمان اجرا زمانی اتفاق می‌افتد که به یک **موقعیت** در **حافظه** اشاره شود
  - در این لحظه، آدرس **منطقی** به آدرس **فیزیکی** متصل می‌شود.

واحد مدیریت حافظه (ادامه)

- **یک طرح ساده در نظر بگیرید که تعمیمی از طرح رجیستر پایه است:**
- در این طرح، رجیستر پایه اکنون «رجیستر جابجایی (Relocation Register) نامیده می‌شود
- مقدار موجود در رجیستر جابجایی relocation register به هر آدرسی که توسط یک فرآیند کاربر در زمان ارسال به حافظه تولید می‌شود، اضافه می‌شود.



Dynamic Loading بارگزاری پویا

- در بحث ما تا کنون، لازم بوده است که کل برنامه و تمام داده های یک فرآیند در حافظه فیزیکی باشد تا فرآیند اجرا شود. بنابراین اندازه یک فرآیند به اندازه حافظه فیزیکی محدود شده است که بارگزاری ایستا است.
- بارگزاری پویا :همه روال ها در قالب بار قابل جابجایی روی دیسک نگهداری می شوند.
- برنامه اصلی در حافظه بارگذاری شده و اجرا می شود. هنگامی که یک روال نیاز به فراخوانی روال دیگری دارد، روال فراخوانی‌کننده ابتدا بررسی می کند که آیا روال دیگر لود شده است یا خیر.
- در غیر این صورت، لودر پیوندی قابل جابجایی فراخوانی می شود تا روال مورد نظر را در حافظه لود کند و جداول آدرس برنامه را به روز کند تا این تغییر را منعکس کند. سپس کنترل به روال تازه بارگذاری شده منتقل می شود.

Dynamic Loading بارگزاری پویا

- مزایا:
- کل **برنامه** برای اجرا **نیاز نیست که در حافظه باشد**
- روال (Routine) تا زمانی که **فراخوانی نشود، بارگذاری نمی‌شود**
- این روش باعث **استفاده بهینه‌تر از فضای حافظه** می‌شود، زیرا روال‌های بلااستفاده هرگز بارگذاری نمی‌شوند
- تمام روال‌ها در قالب بارگذاری قابل جابه‌جایی روی دیسک نگه داشته می‌شوند
- این روش زمانی مفید است که به حجم زیادی از کد برای مدیریت موارد کم‌اتفاق نیاز باشد
- این روش به پشتیبانی **ویژه‌ای از سیستم‌عامل نیاز ندارد**
  - الف) **لودر دینامیکی** از طریق **طراحی برنامه** پیاده‌سازی می‌شود
  - ب) **سیستم‌عامل** می‌تواند با **ارائه‌ی کتابخانه‌هایی** برای پیاده‌سازی بارگذاری دینامیکی کمک کند.

Dynamic Linking اتصال پویا

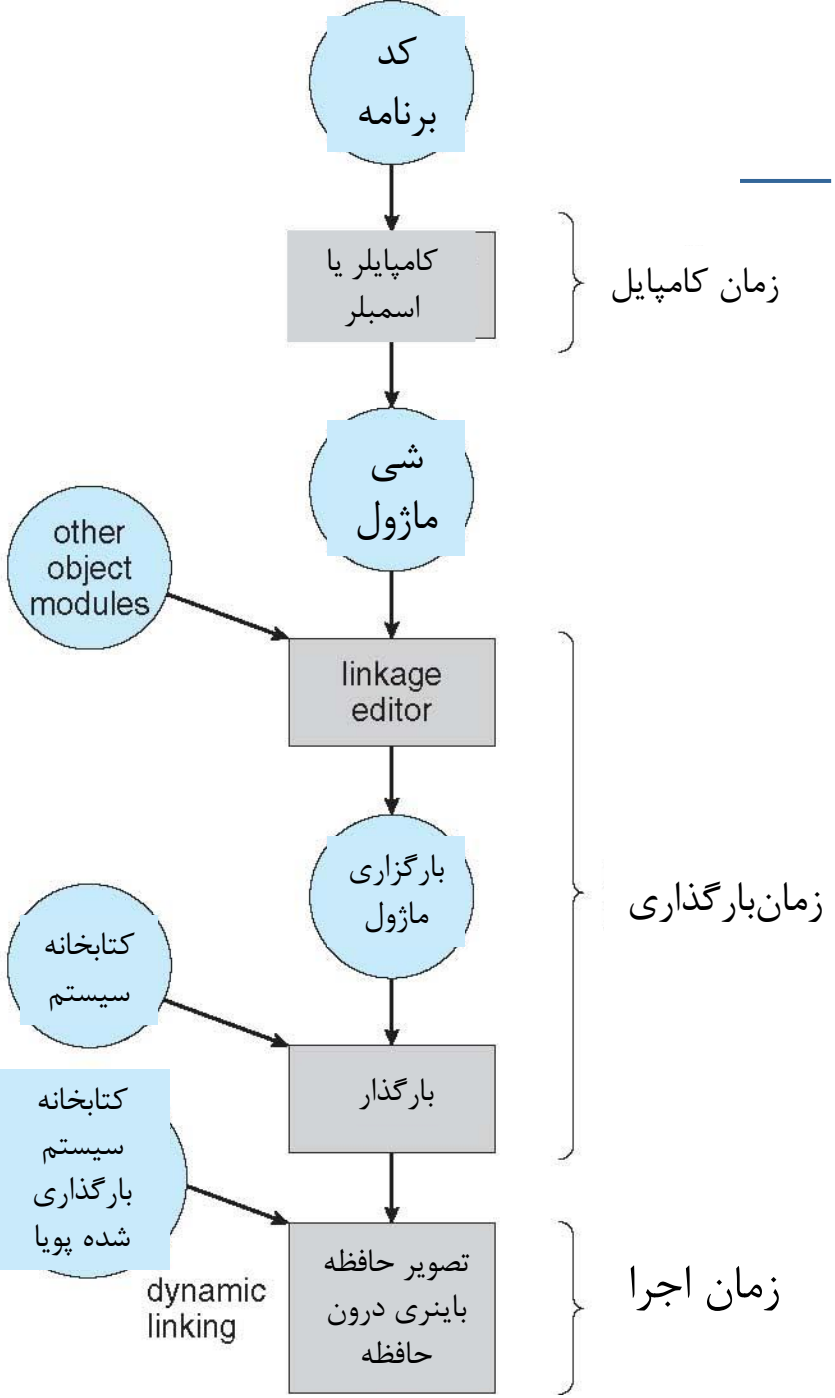
- کتابخانه‌های پیوند پویا (DLL) راهکاری برای **بهینه‌سازی منابع** در سیستم‌عامل هستند .برخلاف کتابخانه‌های سنتی که مستقیما در برنامه گنجانده می‌شوند، DLLها در زمان اجرای برنامه به آن لینک داده می‌شوند .این یعنی هر برنامه فقط به توابع مورد نیاز خود از کتابخانه دسترسی پیدا می‌کند.
- این رویکرد چند مزیت کلیدی دارد:
  - **کاهش حجم برنامه‌های کاربردی :**با حذف کدهای تکراری که در چندین برنامه استفاده می‌شوند، اندازه نهایی برنامه‌ها به شکل قابل توجهی کاهش می‌یابد .این امر به ویژه برای کتابخانه‌های حاوی توابع و کدهای حجیم اهمیت زیادی دارد.
  - **بهینه‌سازی استفاده از حافظه :**از آنجایی که تنها یک نسخه از DLL در حافظه بارگذاری می‌شود، چندین برنامه به طور همزمان می‌توانند از آن استفاده کنند .این کار باعث صرفه‌جویی در حافظه رم شده و عملکرد کلی سیستم را بهبود می‌بخشد.
  - **بروزرسانی ساده‌تر کتابخانه‌ها :**بروزرسانی یک DLL بر خلاف کتابخانه‌های سنتی که مستقیما در برنامه‌ها قرار دارند، نیازی به به‌روزرسانی تک تک برنامه‌ها ندارد .با به‌روزرسانی نسخهDLL ، تمامی برنامه‌های وابسته از قابلیت‌های جدید کتابخانه بهره‌مند می‌شوند.

کتابخانه های پیوندپذیر(DLL)

- **مشابه ماژول‌های شیء** -توسط لودر در تصویر نهایی برنامه (باینری) ترکیب می‌شوند.
- **پیوند دینامیک (برخلاف پیوند ایستا):**
  - پیوند به زمان اجرا موکول می‌شود (شبیه بارگذاری پویا).
  - کاربرد :کتابخانه‌های سیستمی (مثل کتابخانه استاندارد زبانC)
- **مزایای DLL ها:**
  - **کاهش حجم برنامه :**هر برنامه نیازی به کپی جداگانه از کتابخانه ندارد.
  - **بهینه‌سازی حافظه :**یک نسخه از DLL در حافظه اصلی برای چندین فرایند قابل اشتراک است.
- **کاربرد گسترده :**کتابخانه‌های DLL در سیستم‌عامل‌های ویندوز و لینوکس استفاده می‌شوند .

پردازش چند مرحله ای یک برنامه کاربر

- کتابخانه‌های پیوندی پویا DLL کتابخانه‌های سیستمی هستند که هنگام اجرای برنامه‌ها به برنامه‌های کاربر مرتبط می‌شوند.
- برخی از سیستم‌عامل‌ها فقط از پیوند استاتیک پشتیبانی می‌کنند که در آن کتابخانه‌های سیستم این مدلی رفتار می‌شوند.



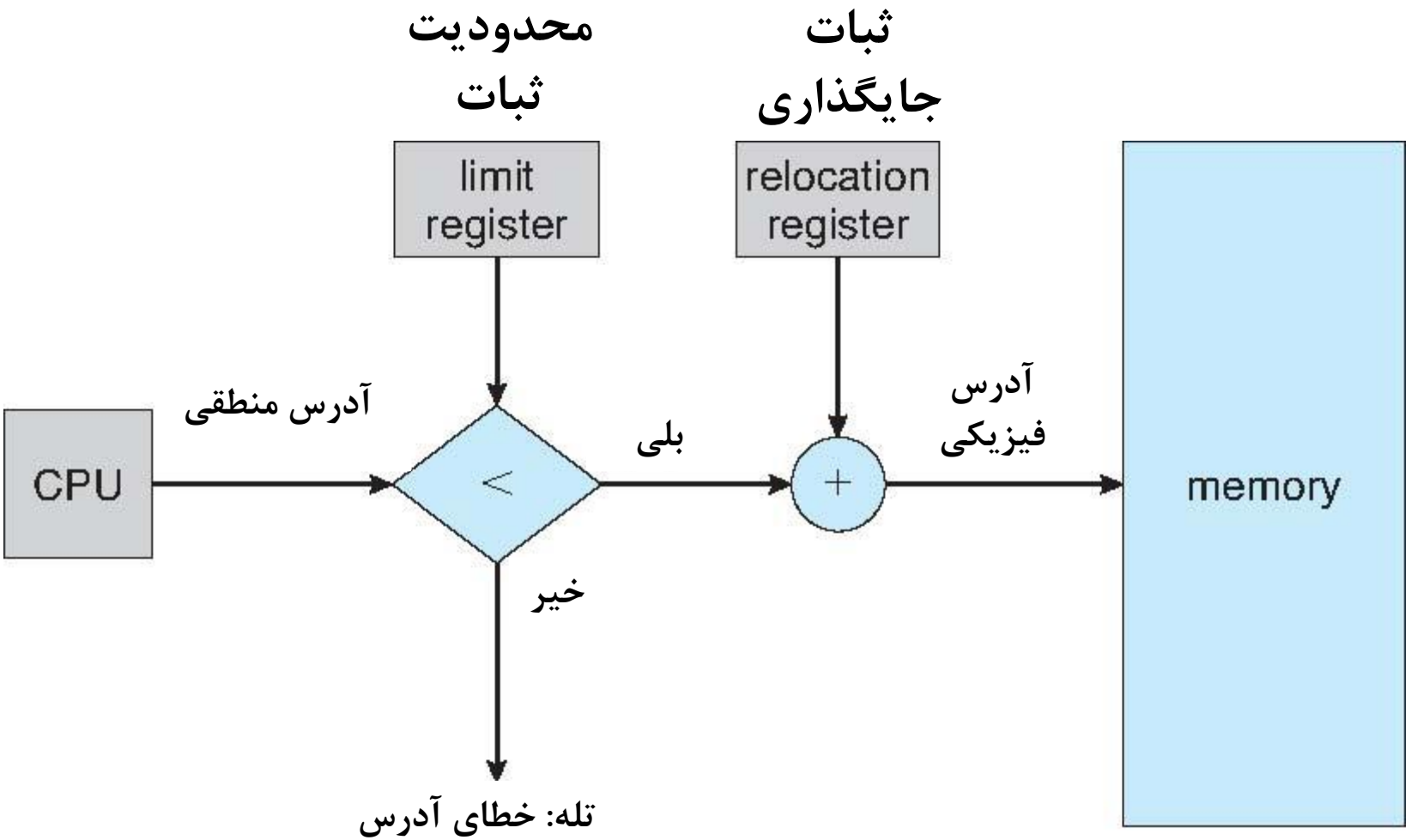
Contiguous Allocation تخصیص پیوسته

- حافظه اصلی باید هم از سیستم‌عامل و هم از فرآیندهای کاربر پشتیبانی کند .این یک منبع محدود است و باید به طور کارآمد تخصیص داده شود.
  - **تخصیص پیوسته :(Contiguous Allocation)** یکی از روش‌های اولیه‌ی تخصیص حافظه است.
- حافظه اصلی معمولاً به دو بخش تقسیم می‌شود:
- قرارگیری هسته در حافظه می‌تواند در آدرس‌های پایین یا بالا باشد .این تصمیم به عوامل مختلفی بستگی دارد، مانند محل **قرارگیری بردار وقفه** (interrupt vector) ،
  - بسیاری از سیستم‌عامل‌ها (از جمله لینوکس و ویندوز) هسته را در آدرس‌های بالای حافظه قرار می‌دهند.
- فرآیندهای کاربر :سپس در پایین آدرس حافظه قرار می‌گیرند.
  - هر فرآیند در یک بخش پیوسته‌ی مجزا از حافظه قرار دارد.

تخصیص پیوسته (ادامه)

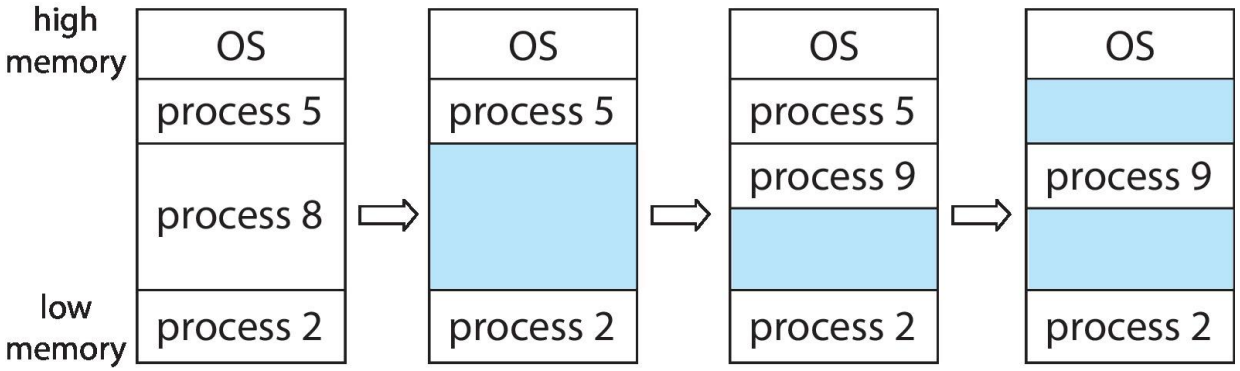
- از رجیسترهای جابجایی (Relocation Registers) برای محافظت فرآیندهای کاربر از دسترسی به یکدیگر و همچنین از تغییر کد و داده‌های سیستم‌عامل استفاده می‌شود.
  - **رجیستر پایه :**مقدار کوچکترین آدرس فیزیکی را در خود نگه می‌دارد.
  - **رجیستر محدوده :**دامنه‌ی آدرس‌های منطقی را در خود نگه می‌دارد؛ هر آدرس منطقی باید کمتر از مقدار رجیستر محدوده باشد.
  - واحد مدیریت حافظه (MMU) به صورت پویا آدرس‌های منطقی را نگاشت می‌کند.
  - این روش می‌تواند به اقداماتی مانند **موقت** بودن کد هسته (Kernel Code) و تغییر اندازه هسته اجازه دهد.

Hardware Support for Relocation and Limit Registers



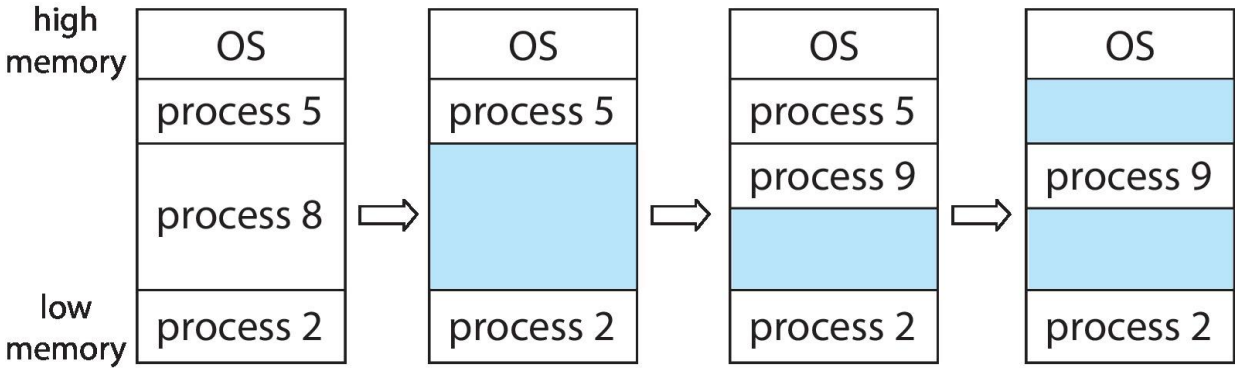
اندازه پارتیشن متغیر

- **تخصیص چندبخشی (Multiple-Partition Allocation):**
- در این روش، درجه چندبرنامگی (Multiprogramming) توسط تعداد پارتیشن‌ها محدود می‌شود.
- برای کارایی بیشتر، **اندازه‌های پارتیشن‌ها** متغیر هستند (متناسب با نیازهای یک فرآیند خاص).
- **سوراخ (Hole):** بلوکی از حافظه در دسترس؛ سوراخ‌ها با اندازه‌های مختلف در سراسر حافظه پراکنده‌اند.



اندازه پارتیشن متغیر

- هنگامی که فرآیندی وارد می‌شود، حافظه‌ای از یک سوراخ به اندازه‌ی کافی برای جای دادن آن به آن اختصاص داده می‌شود.
- خارج شدن یک فرآیند، پارتیشن آن را آزاد می‌کند و پارتیشن‌های مجاور آزاد با هم ترکیب می‌شوند.
- سیستم‌عامل اطلاعات زیر را نگهداری می‌کند:
  - الف) پارتیشن‌های اختصاص‌یافته ب) پارتیشن‌های آزاد :سوراخ‌ها



Dynamic Storage-Allocation Problem

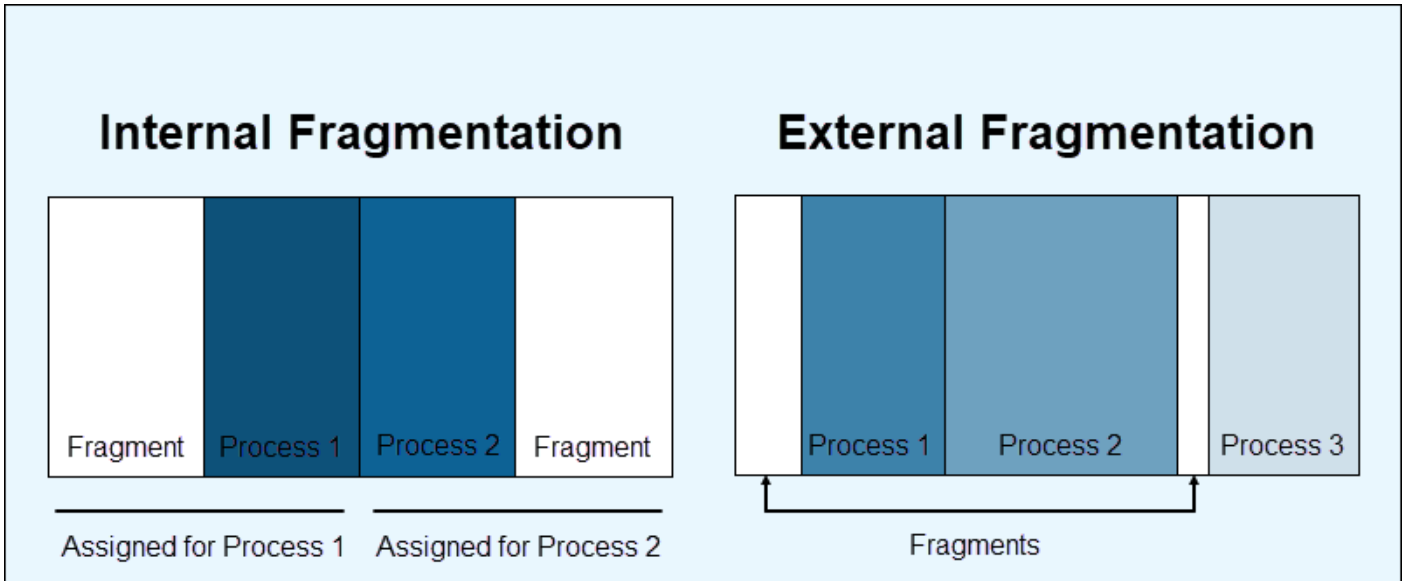
- چگونه می‌توان درخواست فضای به اندازه‌ی n از لیستی از سوراخ‌های آزاد را برآورده کرد؟
  - **اولین برازش (First-fit):** اختصاص اولین سوراخی که به اندازه‌ی کافی بزرگ است.
  - **بهترین برازش (Best-fit):** اختصاص کوچکترین سوراخی که به اندازه‌ی کافی بزرگ است؛ مگر اینکه بر اساس اندازه مرتب شده باشند، کل لیست را باید جستجو کرد.
    - این روش کوچکترین سوراخ باقی‌مانده را ایجاد می‌کند.
  - **بدترین برازش (Worst-fit):** اختصاص بزرگترین سوراخ؛ همچنین باید کل لیست را جستجو کرد.
    - این روش بزرگترین سوراخ باقی‌مانده را ایجاد می‌کند.
- اولین برازش و بهترین برازش از نظر سرعت و استفاده از حافظه نسبت به بدترین برازش بهتر عمل می‌کنند

Fragmentationپراکندگی

- فرض کنید یک طرح **تخصیص چندبخشی(Multiple-Partition Allocation)** با فضایی خالی به اندازه ۱۸۴۶۴ بایت داریم.
- اگر فرآیند بعدی ۱۸۴۶۲ بایت حافظه درخواست کند، چه اتفاقی می‌افتد؟
  - اگر دقیقاً همان بلوک درخواستی را اختصاص دهیم، با یک فضای خالی ۲ بایتی باقی می‌مانیم.
  - مدیریت این فضای خالی کوچک، از خود آن فضا، کارایی کمتری خواهد داشت.
- **پراکندگی خارجی:(External Fragmentation)**
  - زمانی رخ می‌دهد که فضای خالی کافی در حافظه وجود داشته باشد، اما این فضا به قطعات کوچک غیرقابل استفاده تقسیم شود.
  - هیچ یک از قطعات به تنهایی برای برآورده کردن نیازهای حافظه یک فرایند جدید کافی نیست.

Fragmentationپراکندگی

- **پراکندگی داخلی (Internal Fragmentation):** حافظه‌ی اختصاص‌یافته ممکن است کمی بزرگ‌تر از حافظه‌ی درخواست‌شده باشد؛ این اختلاف اندازه، حافظه‌ای داخلی یک پارتیشن است که استفاده نمی‌شود.
- تحلیل اولین برازش نشان می‌دهد که با اختصاص N بلوک، تقریبا نیم N۰.۵ دیگری از بلوک‌ها دست می‌روند که تقریبا یک سوم حافظه می‌شود که غیرقابل استفاده می‌شود.این خاصیت به قانون ۵۰ درصد شناخته می‌شود.



پراکندگی (ادامه)

- **پراکندگی خارجی** را می‌توان با فرآیندی به نام فشرده‌سازی کاهش داد.
  - این کار شامل جابه‌جا کردن محتوای حافظه برای ادغام تمام حافظه‌های آزاد در یک بلوک بزرگ پیوسته است .
  - فشرده‌سازی تنها در صورتی امکان‌پذیر است که relocation یا جابه‌جایی حافظه پویا باشد و در زمان اجرا انجام شود (یعنی آدرس‌های حافظه بتوانند در حین اجرای برنامه تغییر کنند).
- **مشکلات ورودی/خروجی(I/O)** :هنگامی که یک فرایند درگیر عملیات ورودی/خروجی (I/O) باشد، مشکلاتی ایجاد می‌شود .برای جلوگیری از از دست رفتن داده‌ها، می‌توان:
  - کل فرایند I/Oرا در حافظه نگه داشت (که رویکردی ناکارآمد است)
  - عملیات I/O را فقط به بافرهای مدیریت‌شده توسط سیستم‌عامل انجام داد.

صفحه بندی

- فضای آدرس فیزیکی یک فرآیند می‌تواند **غیرپیوسته** باشد. به این معنی که بخش‌های مختلف حافظه به فرآیند اختصاص داده می‌شود و این بخش‌ها الزاما مجاور هم قرار ندارند، به شرطی که فضای خالی در دسترس باشد. این روش مزایای زیر را به همراه دارد:

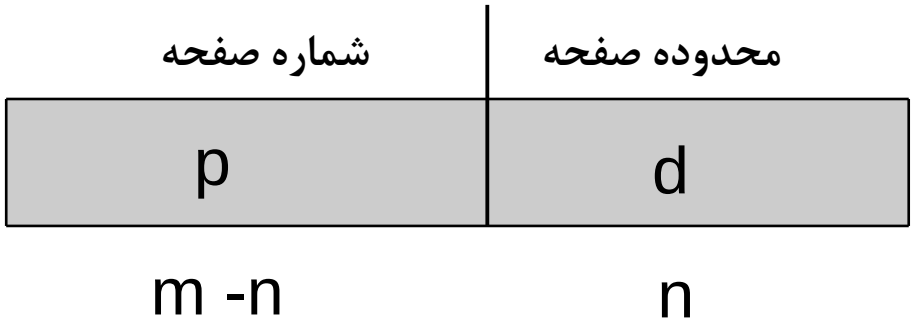
- **جلوگیری از پراکندکی خارجی حافظه (External Fragmentation**
- **سازگاری با اندازه‌های مختلف حافظه:** این روش دیگر با مشکل قطعه‌های حافظه با اندازه‌های نامناسب مواجه نیست و می‌تواند حافظه را به صورت بهینه‌تری مدیریت کند.

مراحل صفحه‌بندی

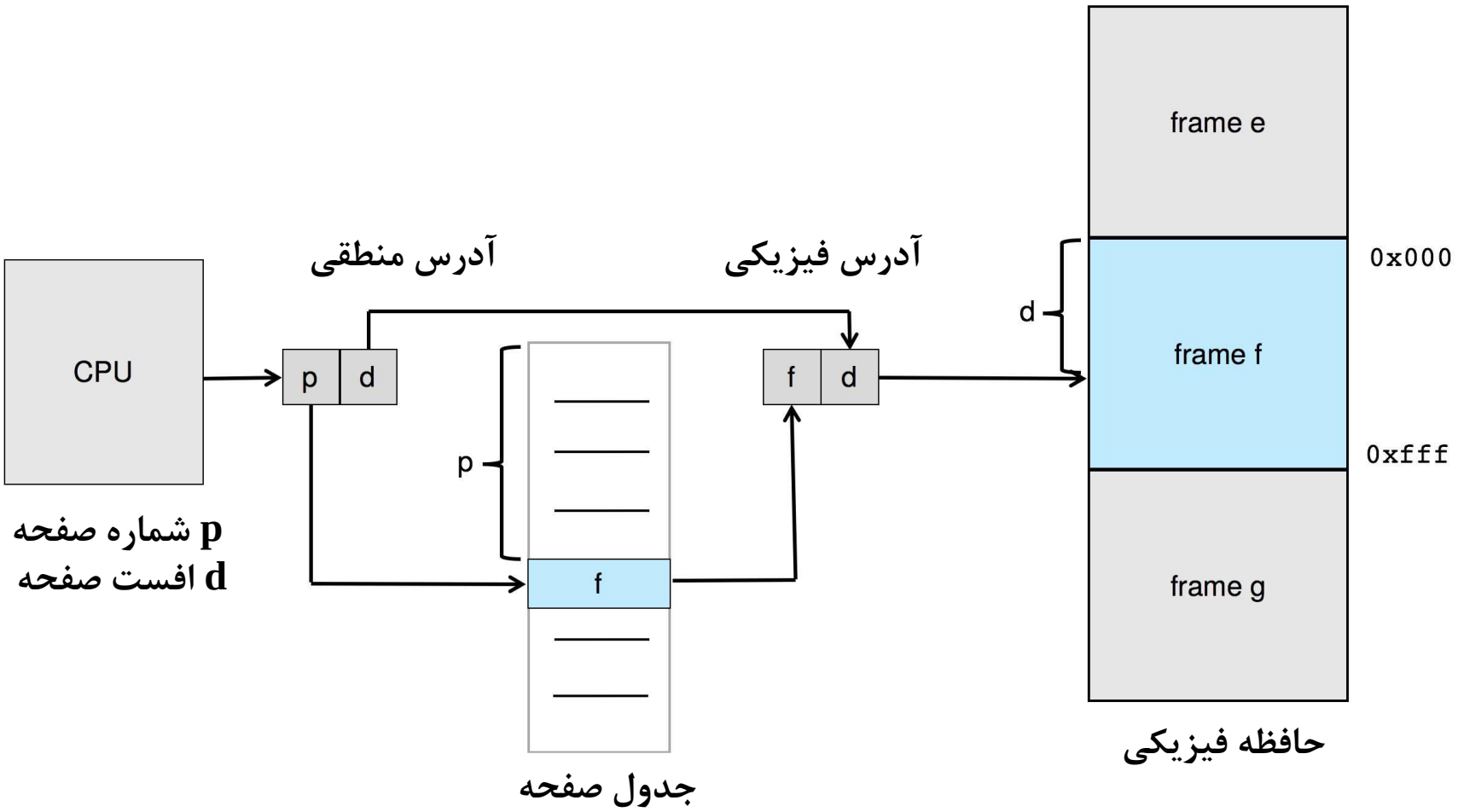
- ۱. **تقسیم حافظه اصلی:** حافظه اصلی به بلوک‌های با اندازه ثابت به نام **فریم** (**Frame**) تقسیم می‌شود. اندازه فریم معمولا توانی از ۲ است و مقداری بین ۵۱۲ بایت تا ۱۶ مگابایت دارد.
- ۲. **تقسیم حافظه منطقی:** حافظه منطقی فرآیند نیز به بلوک‌های با اندازه مشابه به نام **صفحه** (**Page**) تقسیم می‌شود.
- ۳. **ردیابی فریم‌های خالی:** سیستم‌عامل لیستی از تمام فریم‌های خالی در حافظه را نگهداری می‌کند.
- ۴. **اجرای برنامه:** برای اجرای یک برنامه با اندازه N صفحه، سیستم‌عامل باید N فریم خالی پیدا کند و برنامه را در آن فریم‌ها بارگذاری نماید.
- ۵. **جدول صفحه‌بندی:** برای ترجمه آدرس‌های منطقی برنامه به آدرس‌های فیزیکی حافظه، از یک جدول صفحه‌بندی (Page Table) استفاده می‌شود.
- ۶. **حافظه جانبی** (**Backing Store**): حافظه جانبی نیز به صفحاتی با اندازه مشابه تقسیم می‌شود. این صفحات محل ذخیره شدن برنامه‌هایی هستند که در حال حاضر در حال اجرا نیستند.
- ۷. **پراکندی داخلی** :با وجود مزایای روش صفحه‌بندی، همچنان ممکن است با مشکل درون‌پارگی حافظه مواجه شویم.

الگوی ترجمه آدرس

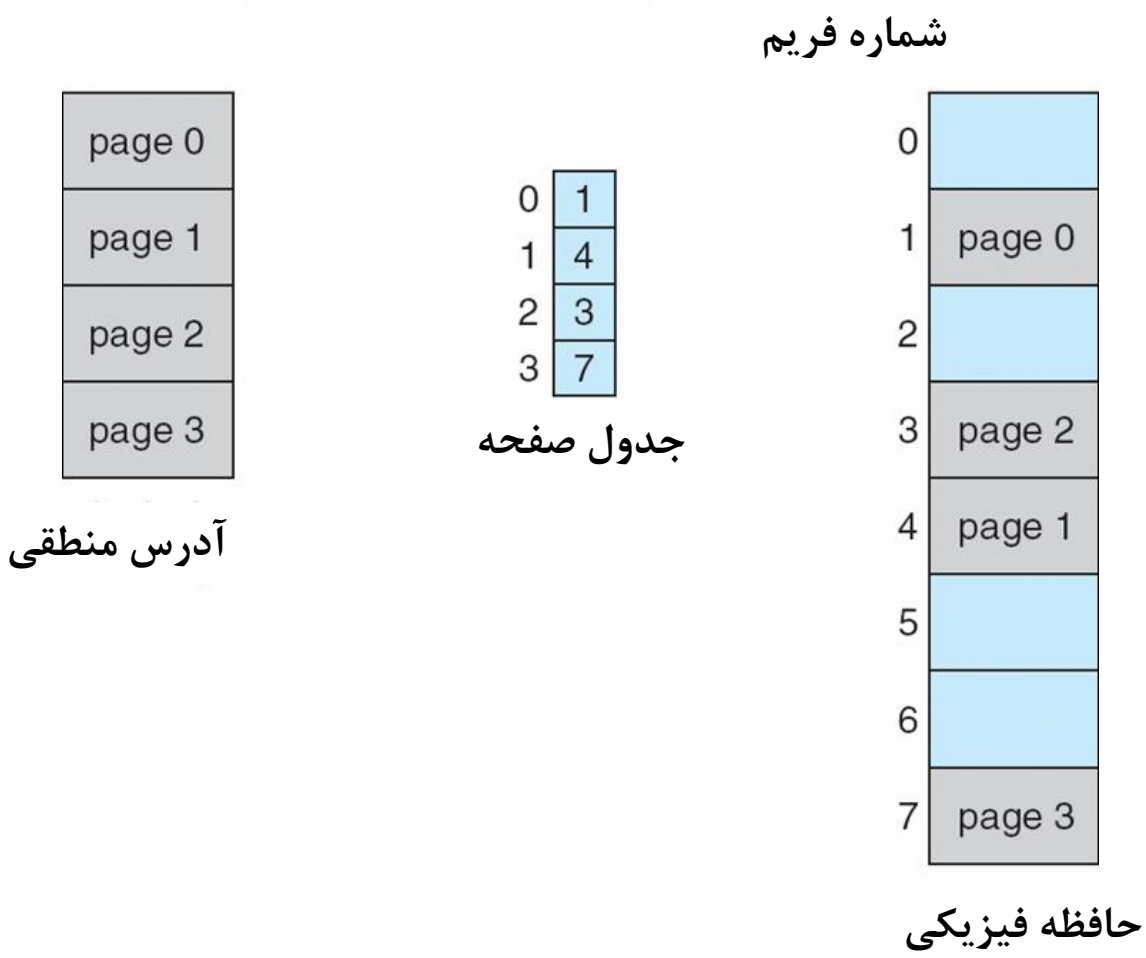
- وقتی پردازنده یک آدرس منطقی تولید می‌کند، این آدرس به دو بخش تقسیم می‌شود:
- **شماره صفحه (p) :** به عنوان یک شاخص در جدول صفحه عمل می‌کند که حاوی آدرس پایه هر صفحه در حافظه فیزیکی است.
- **آفست (فاصله از مبدا) صفحه (d) :** این مقدار با آدرس پایه ترکیب می‌شود تا آدرس نهایی حافظه فیزیکی را که به واحد حافظه ارسال می‌گردد، تعیین کند.
- فرض کنید فضای آدرس منطقی ۲^nبایت و اندازه صفحه ۲^mبایت باشد



Paging Hardware

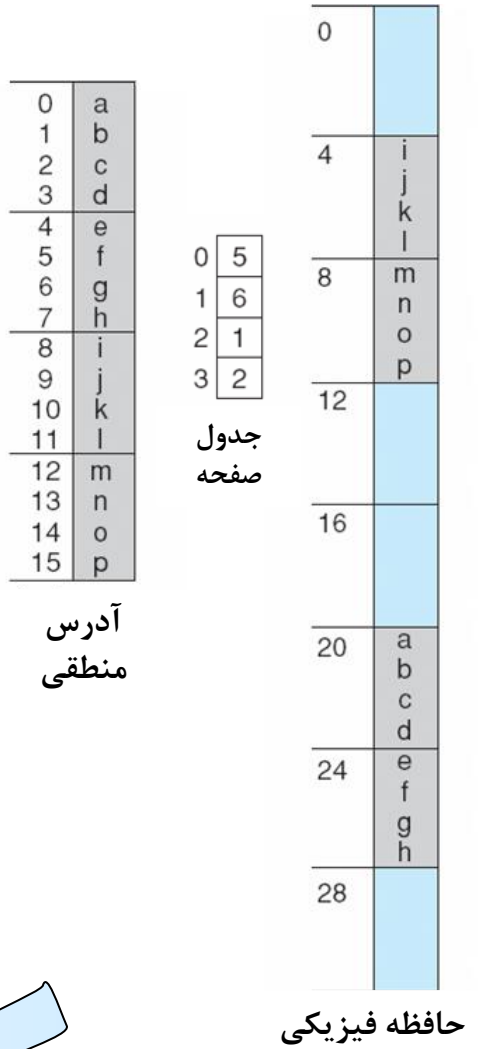


Paging Model of Logical and Physical Memory



مثال صفحه بندی

- آدرس منطقی: مثال n = 2 و m = 4 این به معنی اندازه صفحه 2^2 ۴ بایت و حافظه فیزیکی ۳۲یا 2^4 بایت (شامل ۸ صفحه ۴ بایتی) است.



نحوه‌ی نگاشت حافظه‌ی منطقی کاربر به حافظه‌ی فیزیکی را نمایش می‌دهیم:

- **آدرس منطقی ۰:** صفحه‌ی ۰، افست ۰
  - با مراجعه به جدول صفحه‌بندی، می‌بینیم صفحه‌ی ۰ در فریم ۵ قرار دارد. هر فریم یا صفحه ۴ بایتی
  - بنابراین، آدرس منطقی ۰ به آدرس فیزیکی ۲۰ [= (۵ × ۴) + ۰] نگاشت می‌شود.
- **آدرس منطقی ۳:** صفحه‌ی ۰، افست ۳
  - به آدرس فیزیکی ۲۳ [= (۵ × ۴) + ۳] نگاشت می‌شود.
- **آدرس منطقی ۴:** صفحه‌ی ۱، افست ۰
  - طبق جدول صفحه‌بندی، صفحه‌ی ۱ به فریم ۶ نگاشت می‌شود.
  - پس، آدرس منطقی ۴ به آدرس فیزیکی ۲۴ [= (۶ × ۴) + ۰] نگاشت می‌شود.

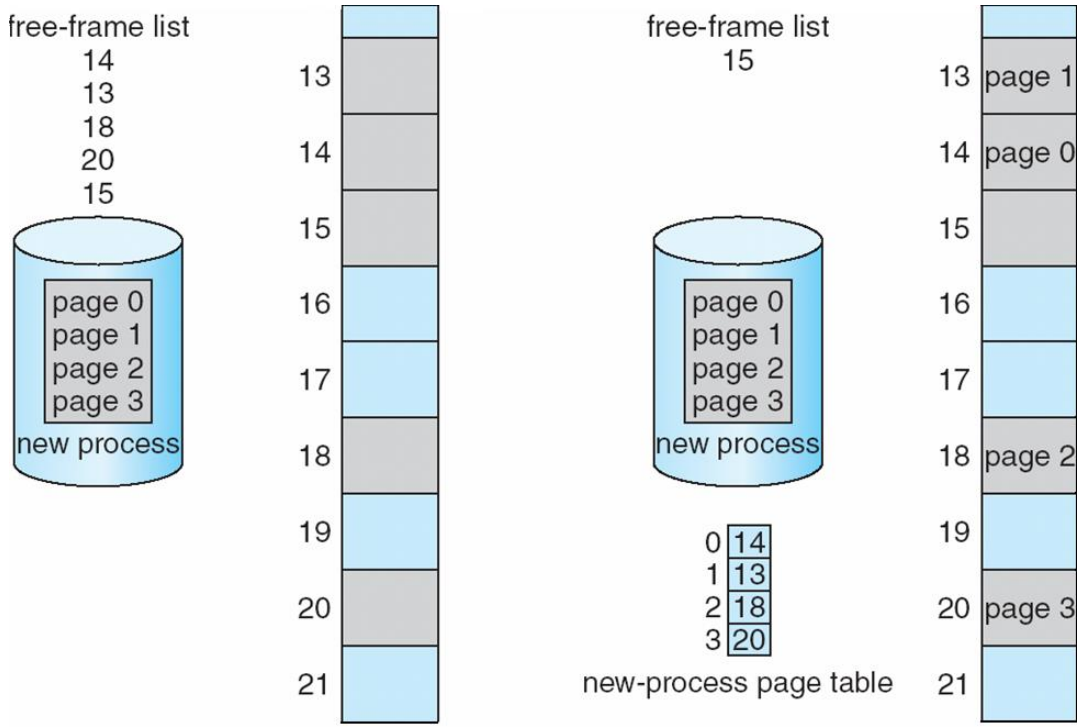
صفحه‌بندی – محاسبه پراکندگی داخلی

- **اندازه صفحه :**۲،۰۴۸ بایت
- **اندازه فرآیند :**۷۲،۷۶۶ بایت
- با تقسیم اندازه فرآیند بر اندازه صفحه $\frac{۷۲۷۶۶}{۲۰۴۸} = ۳۵.۵۳$، یعنی۲۰۴۸ * ۳۶=مشخص می‌شود که این فرآیند به **۳۵ صفحه یعنی ۷۱۶۸۰ بایت** به همراه **۱،۰۸۶ بایت باقی‌مانده** نیاز دارد. در کل به ۳۶ صفحه نیاز دارد.
  - **پراکندگی داخلی = قسمت استفاده‌شده از آخرین صفحه–اندازه صفحه**
    - **پراکندگی داخلی: ۹۶۲ بایت= ۱،۰۸۶ بایت – ۲،۰۴۸ بایت**
- **بدترین حالت پراکندگی :**در بدترین حالت، یک فرایند به n صفحه + ۱ بایت نیاز داشته باشد است. یعنی صفحه n+1 تقریبا به اندازه یک بایت استفاده می شود و مابقی خالی می‌ماند.
- **میانگین پراکندگی :**به طور میانگین، انتظار می‌رود که **پراکندگی** داخلی نصف اندازه صفحه باشد.

انتخاب اندازه صفحه به یک **موازنه** بین کاهش **پراکندگی** داخلی و مصرف حافظه برای جدول صفحه بستگی دارد .به همین دلیل، اندازه صفحه در طول زمان افزایش یافته است.

فریم‌های آزاد

اگر فرآیندی به n صفحه نیاز داشته باشد، حداقل n فریم باید در حافظه در دسترس باشد.
در صورتی که n فریم در دسترس باشد، این فریم‌ها به فرآیند تخصیص داده می‌شوند .
صفحه اول فرآیند در یکی از فریم‌های تخصیص‌یافته (فریم ۱۴ در مثال) بارگذاری می‌شود و شماره فریم مربوطه در جدول صفحه برای این فرآیند ثبت می‌شود .صفحه بعدی در فریمی دیگر بارگذاری شده و شماره فریم آن در جدول صفحه قرار می‌گیرد و این روند به همین ترتیب ادامه می‌یابد مثل شکل:
صفحه صفر در فریم خالی اول (۱۴). صفحه یک در فریم خالی بعدی (۱۳)...



(a) قبل از تخصیص

(b) بعد از تخصیص

End of Essential part