به نام خدا

سیستم عامل

مدیریت حافظه

حمید فدیشه‌ای، دانشگاه بجنورد

ترم دوم 94-1393

مدیریت اختصاص حافظه

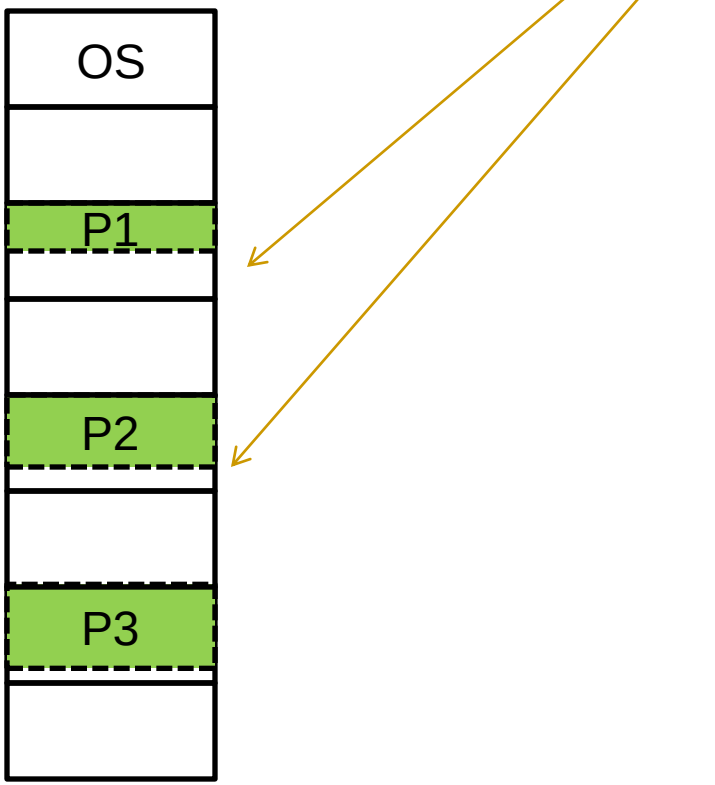
- در مورد نحوه اختصاص فضای حافظه به فرایندها صحبت می‌کند
- رویکردهای مختلفی را بررسی خواهیم کرد
  - بخش‌بندی (Partitioning)
  - صفحه بندی (Paging)
  - قطعه‌بندی (Segmentation)

بخش‌بندی

- بخش‌بندی ایستا (Static)
  - حافظه به چندین بخش با طول ثابت تقسیم می‌شود
  - سیستم عامل معمولا در اولین بخش قرار می‌گیرد (چون روال‌های وقفه معمولا در ابتدای حافظه هستند)
  - به هر فرایند جدید یکی از بخش‌های خالی را اختصاص می‌دهیم
    - اگر کوچک‌تر از اندازه تقسیم بندی باشد اتلاف فضا خواهیم داشت
    - اگر بزرگ‌تر باشد نمی‌توانیم حافظه اختصاص بدهیم

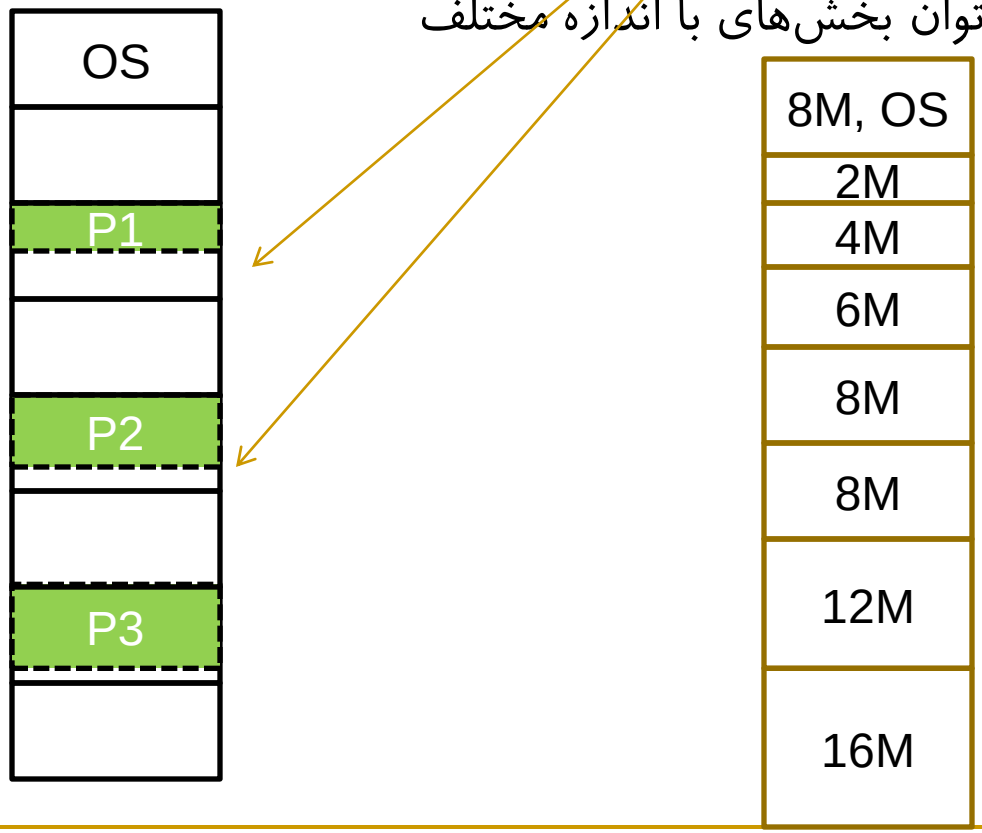
بخش‌بندی

- بخش‌بندی ایستا (Static)
  - به اتلاف حافظه در این روش اصطلاحا تکه تکه شدن داخلی (Internal Fragmentation) گفته می‌شود



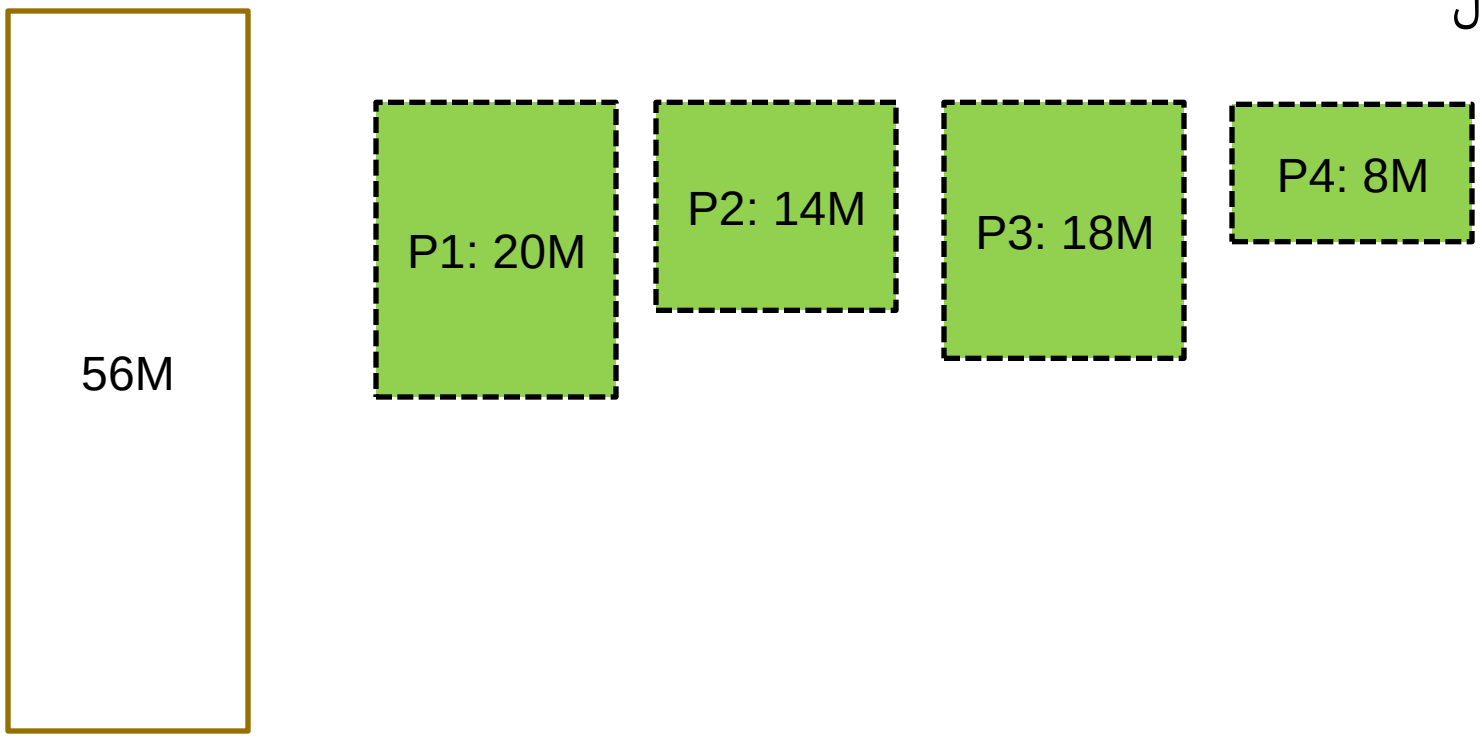
بخش‌بندی

- بخش‌بندی ایستا (Static)
  - به اتلاف حافظه در این روش اصطلاحا تکه تکه شدن داخلی (Internal Fragmentation) گفته می‌شود
  - برای کم کردن این اتلاف می‌توان بخش‌های با اندازه مختلف در نظر گرفت



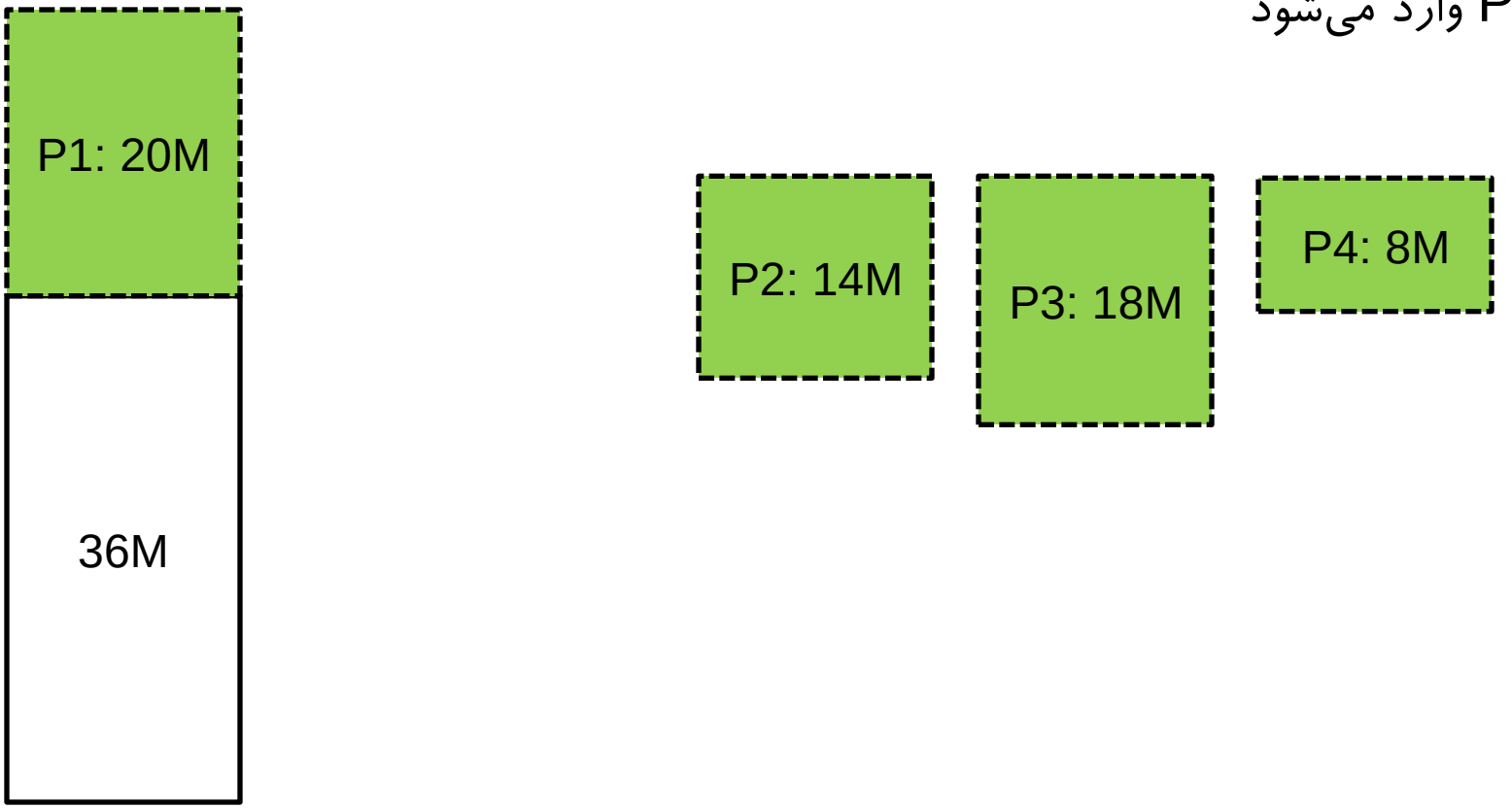
بخش‌بندی

- بخش‌بندی پویا
  - به هر فرایند جدید دقیقا به اندازه نیازش فضا داده شود
  - مثال



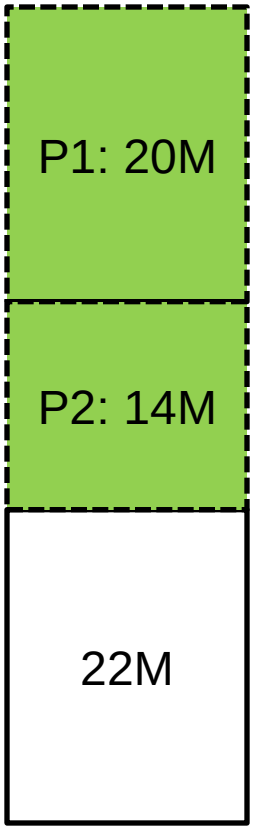
بخش‌بندی

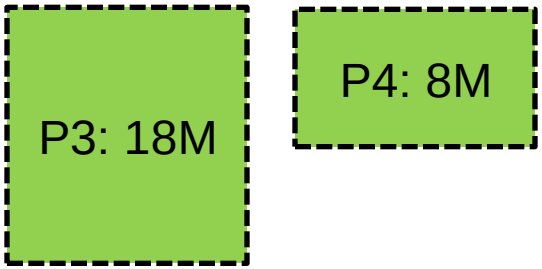
- بخش‌بندی پویا
  - P1 وارد می‌شود



بخش‌بندی

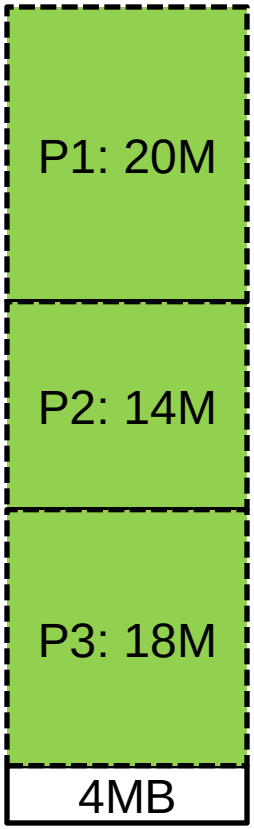
- بخش‌بندی پویا
  - P2 وارد می‌شود

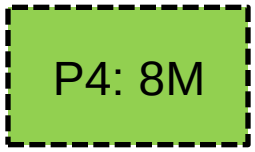




بخش‌بندی

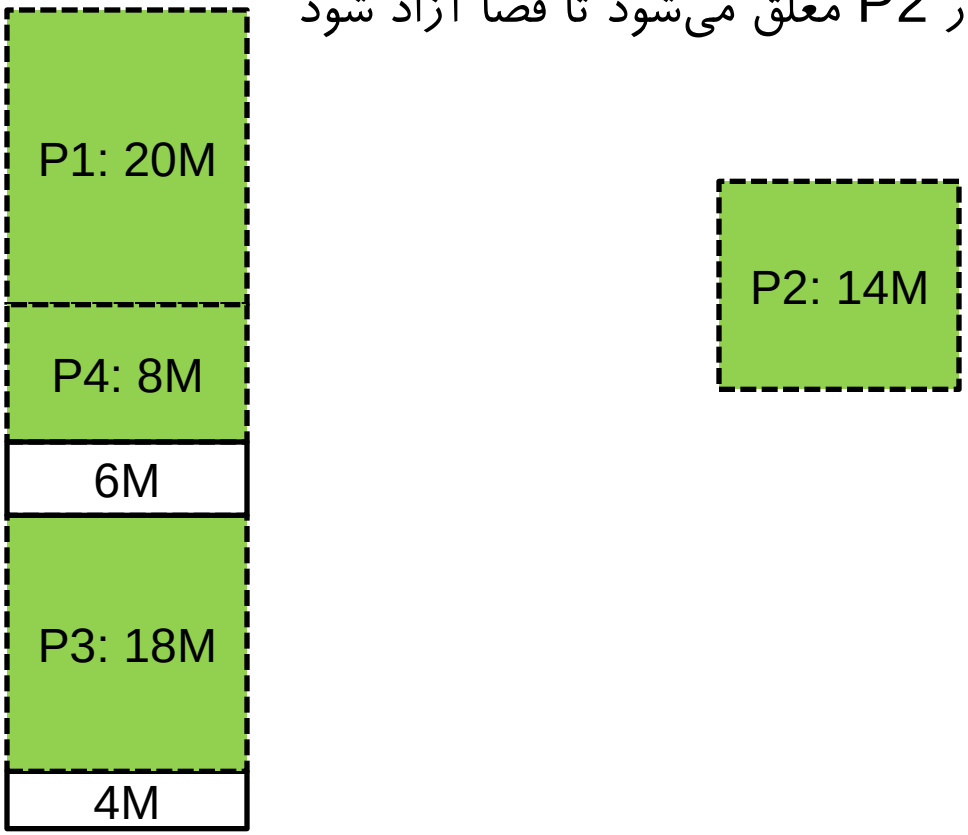
- بخش‌بندی پویا
  - P3 وارد می‌شود





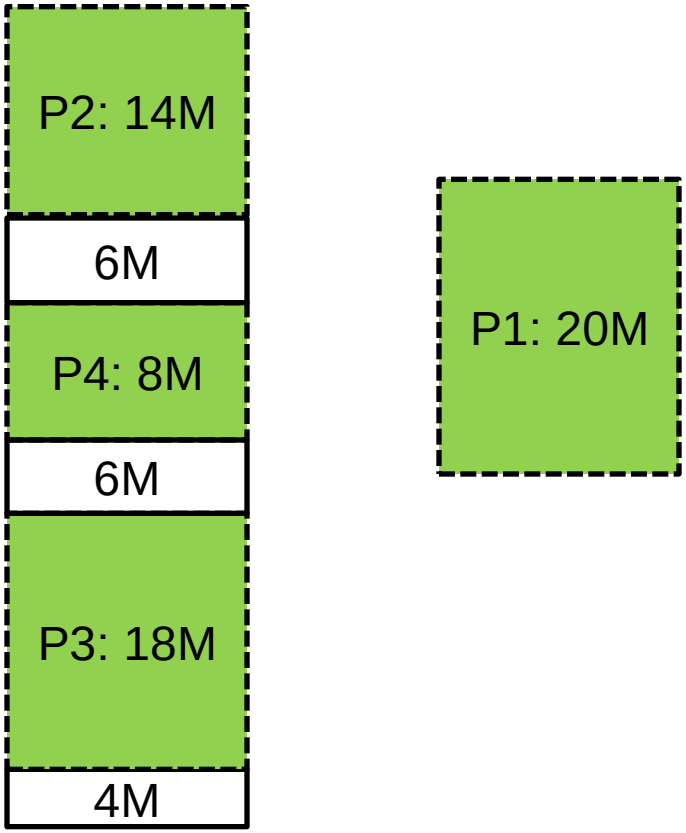
بخش‌بندی

- بخش‌بندی پویا
  - برای P4 فضای کافی نیست. بناچار P2 معلق می‌شود تا فضا آزاد شود

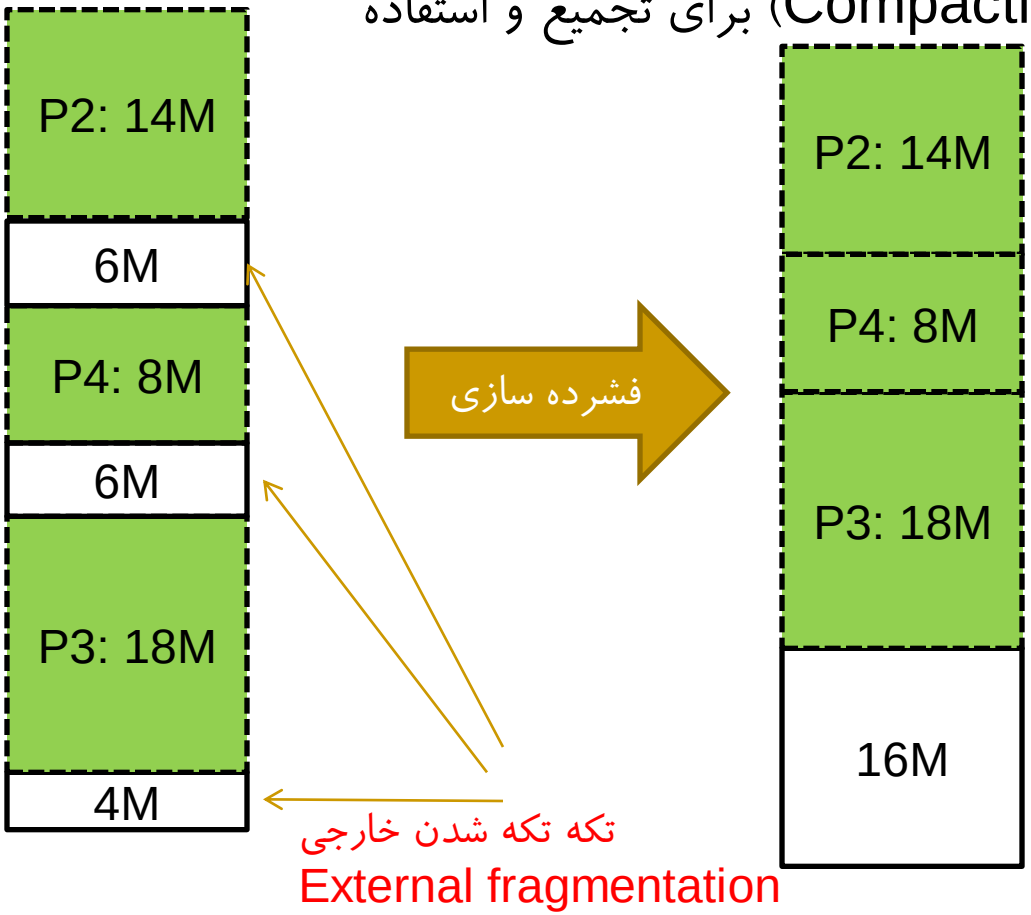


بخش‌بندی

- بخش‌بندی پویا
  - برای بازگشت P2 فضا نیست. P1 معلق می‌شود



- بعد از مدتی تکه تکه شدن خارجی زیادی مشاهده می‌شود
  - نیاز به عمل فشرده سازی (Compaction) برای تجمیع و استفاده از حفره‌ها (فضای خالی)
    - عملی پرهزینه است



بخش‌بندی

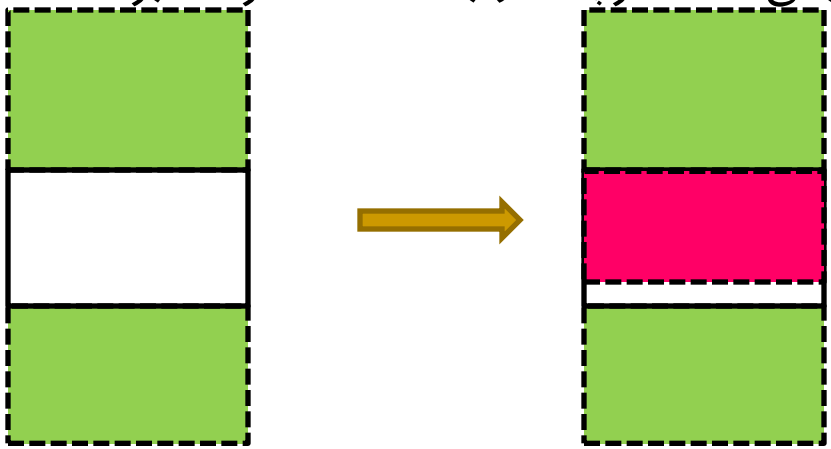
- سیستم عامل لیست حفره‌ها را دارد
  - فرایند جدید در کدام حفره قرار بگیرد؟ سیاست‌های مختلف ممکن است
    - اولین برازش (First Fit)
    - بهترین برازش (Best Fit)
    - بدترین برازش (Worst Fit)
    - برازش بعدی (Next Fit)

بخش‌بندی

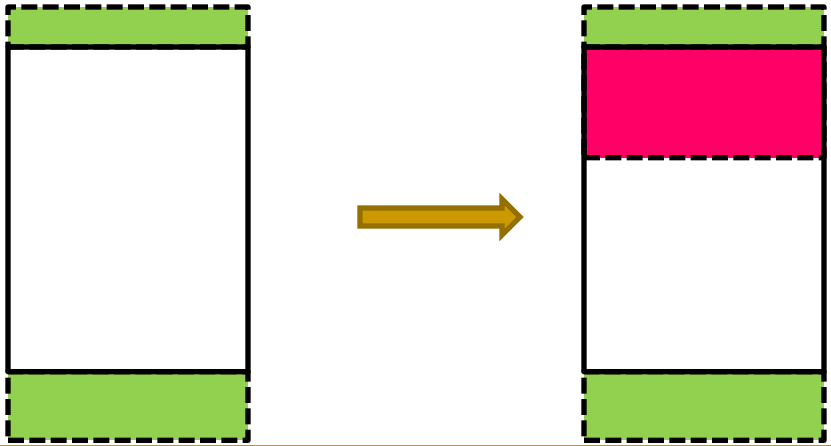
- سیستم عامل لیست حفره‌ها را دارد
  - فرایند جدید در کدام حفره قرار بگیرد؟ سیاست‌های مختلف ممکن است
    - اولین برازش: اولین جای خالی (از ابتدای حافظه) که فرایند قابل جاگرفتن در آن باشد انتخاب می‌شود
    - بهترین برازش: کوچکترین جای خالی که فرایند قابل جا گرفتن در آن باشد انتخاب می‌شود
    - بدترین برازش: بزرگترین جای خالی که ...
    - برازش بعدی: اولین جای خالی (از محل برازش قبلی) که ...

بخش‌بندی

- سیاست بهترین برازش به نظر بهترین انتخاب است اما لزوما نیست. معمولا اندازه حفره و فرایند دقیقا برابر نیستند و جای خالی باقی‌مانده کوچک و بلااستفاده خواهد بود



- به همین خاطر سیاست بدترین برازش جای مطرح شدن دارد



بخش‌بندی

- سیاست اولین برازش پیاده‌سازی ساده دارد
- سیاست برازش بعدی برای رفع مشکلی از سیاست اولین برازش است: تکه تکه شدن سریع ابتدای حافظه که اکثر جابجایی‌ها آن جا انجام می‌شود

بخش‌بندی

- روش رفاقتی (Buddy)
  - ما بین روش ایستا و پویا
  - حافظه را آن قدر نصف می‌کند تا به اندازه مورد نظر برسد

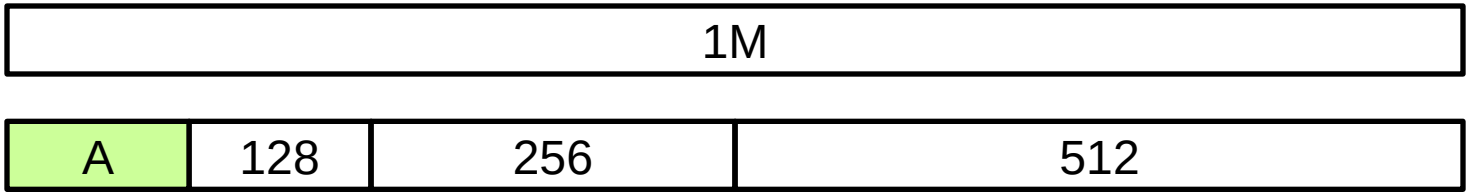
بخش‌بندی رفاقتی

1M

100K درخواست A

64

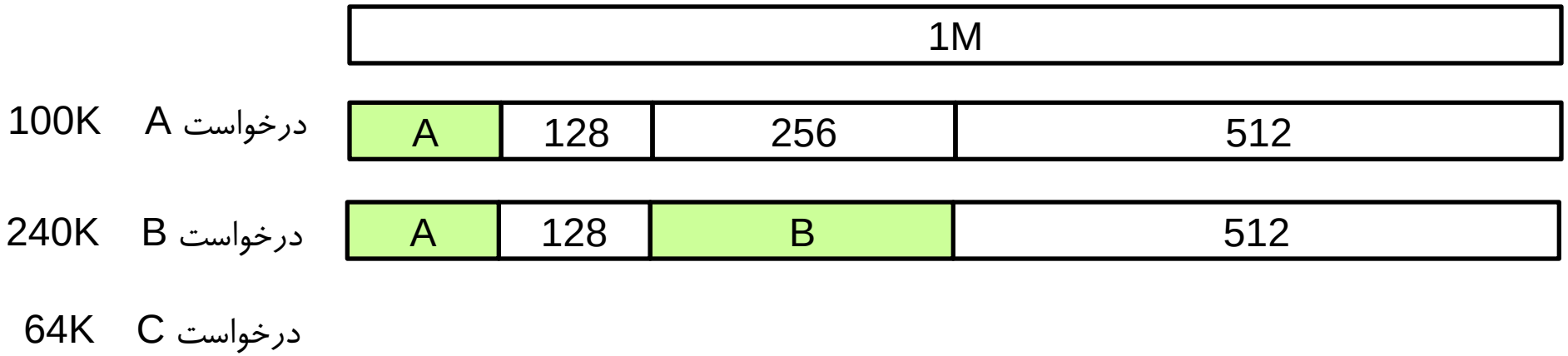
بخش‌بندی رفاقتی



100K درخواست A

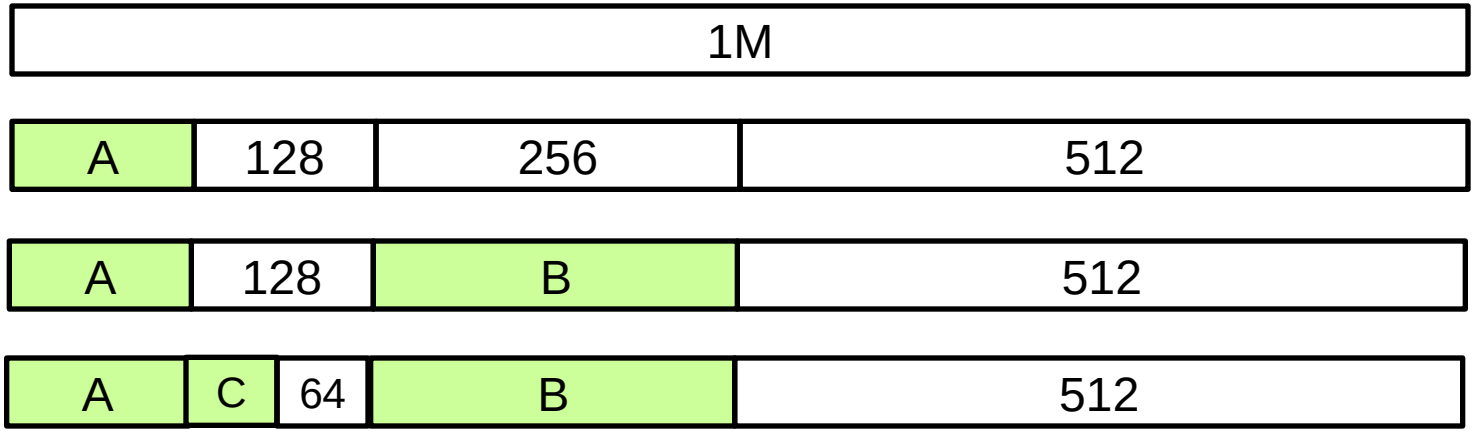
240K درخواست B

بخش‌بندی رفاقتی



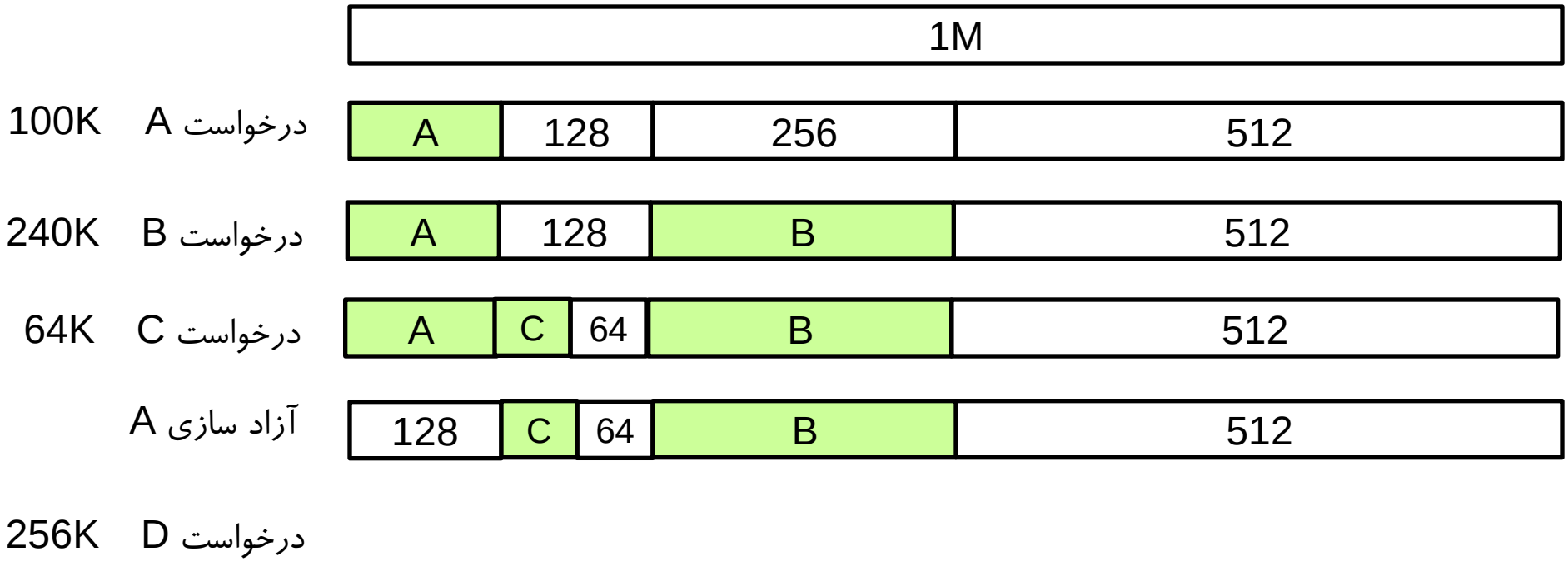
64

بخش‌بندی رفاقتی

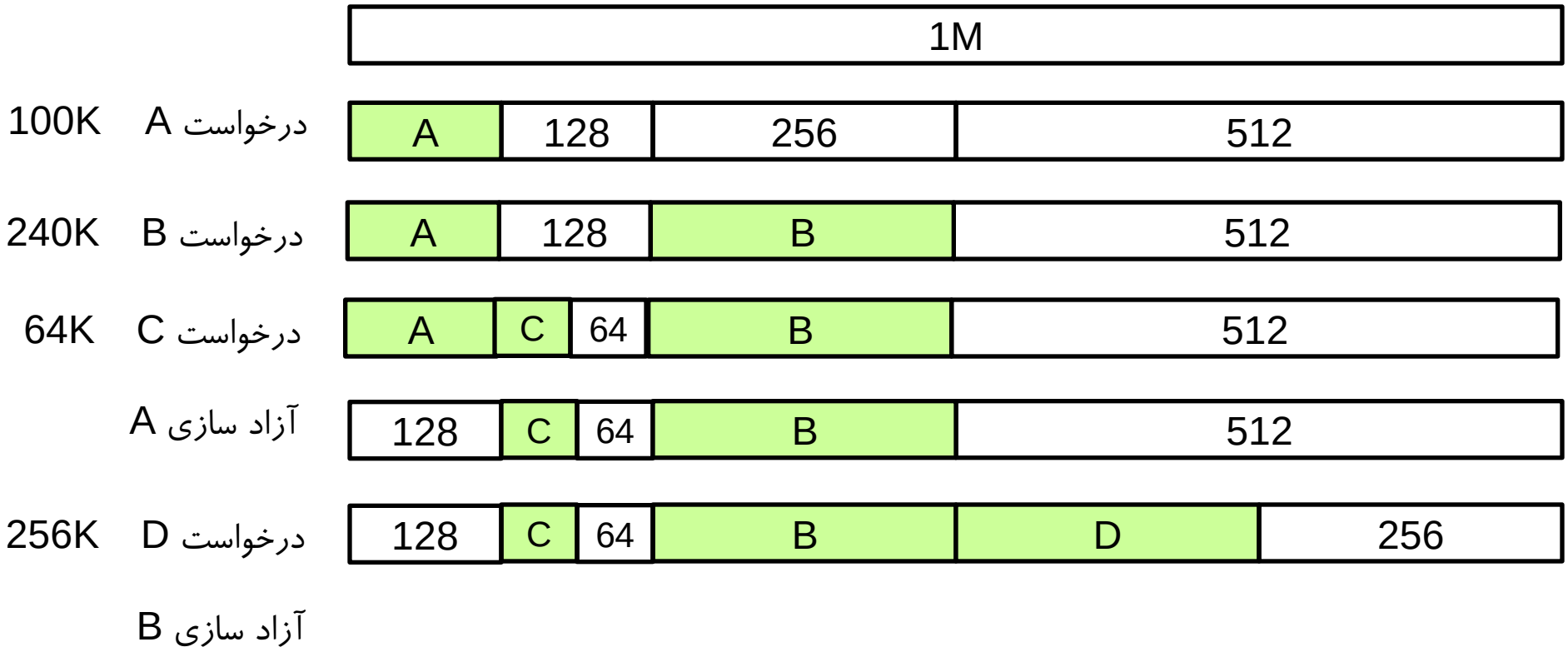


آزاد سازی A

بخش‌بندی رفاقتی

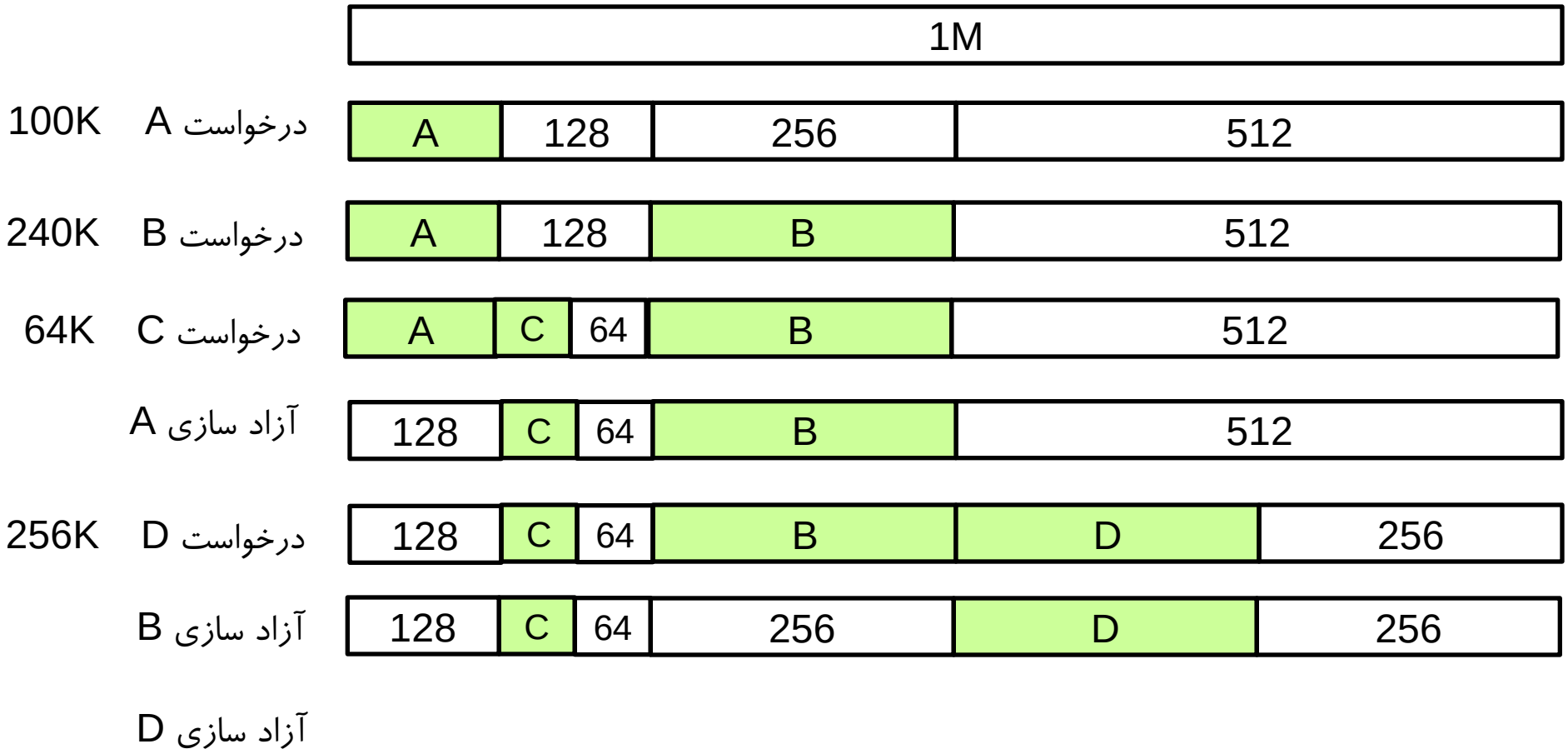


بخش‌بندی رفاقتی



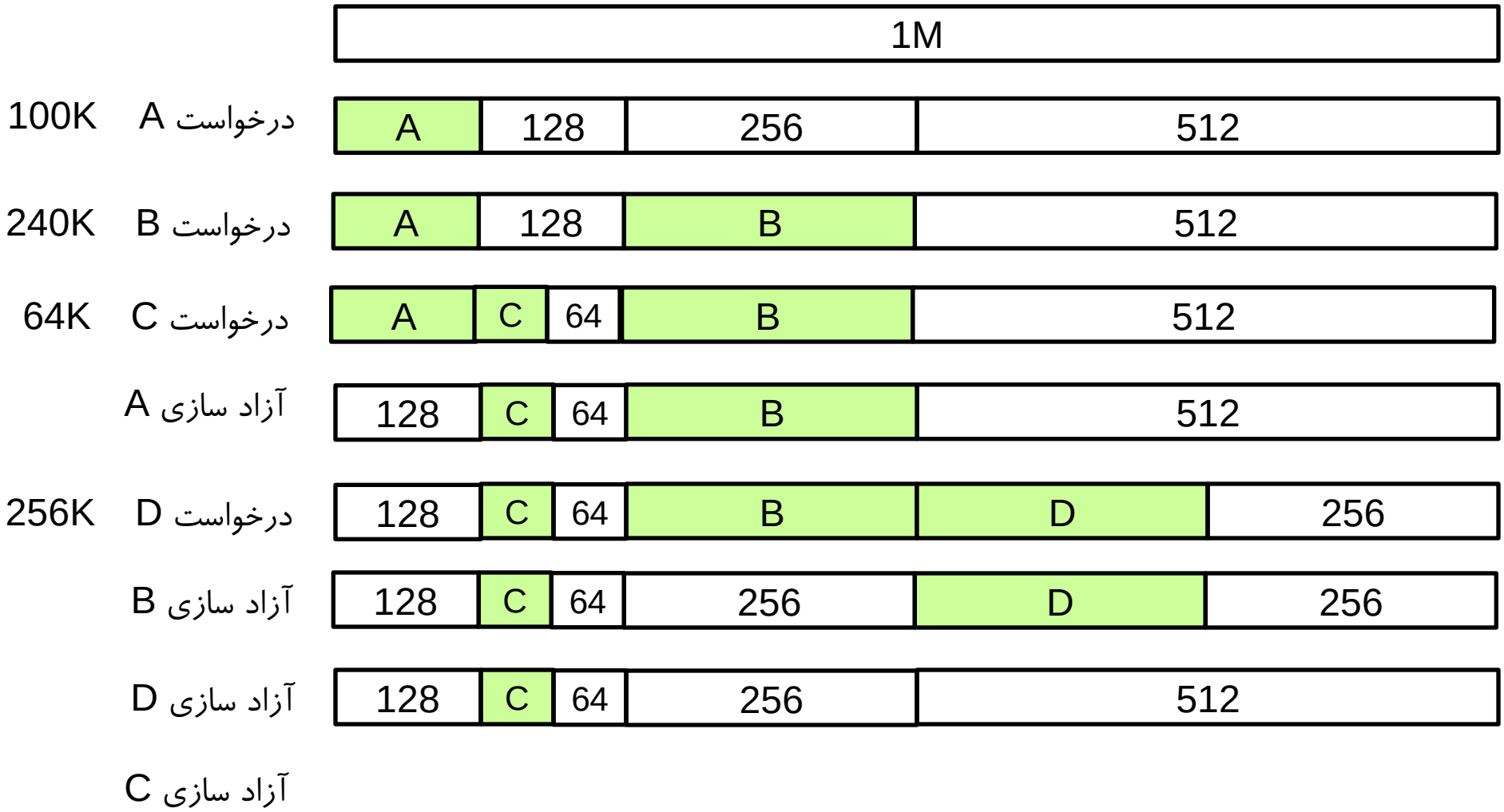
64

بخش‌بندی رفاقتی



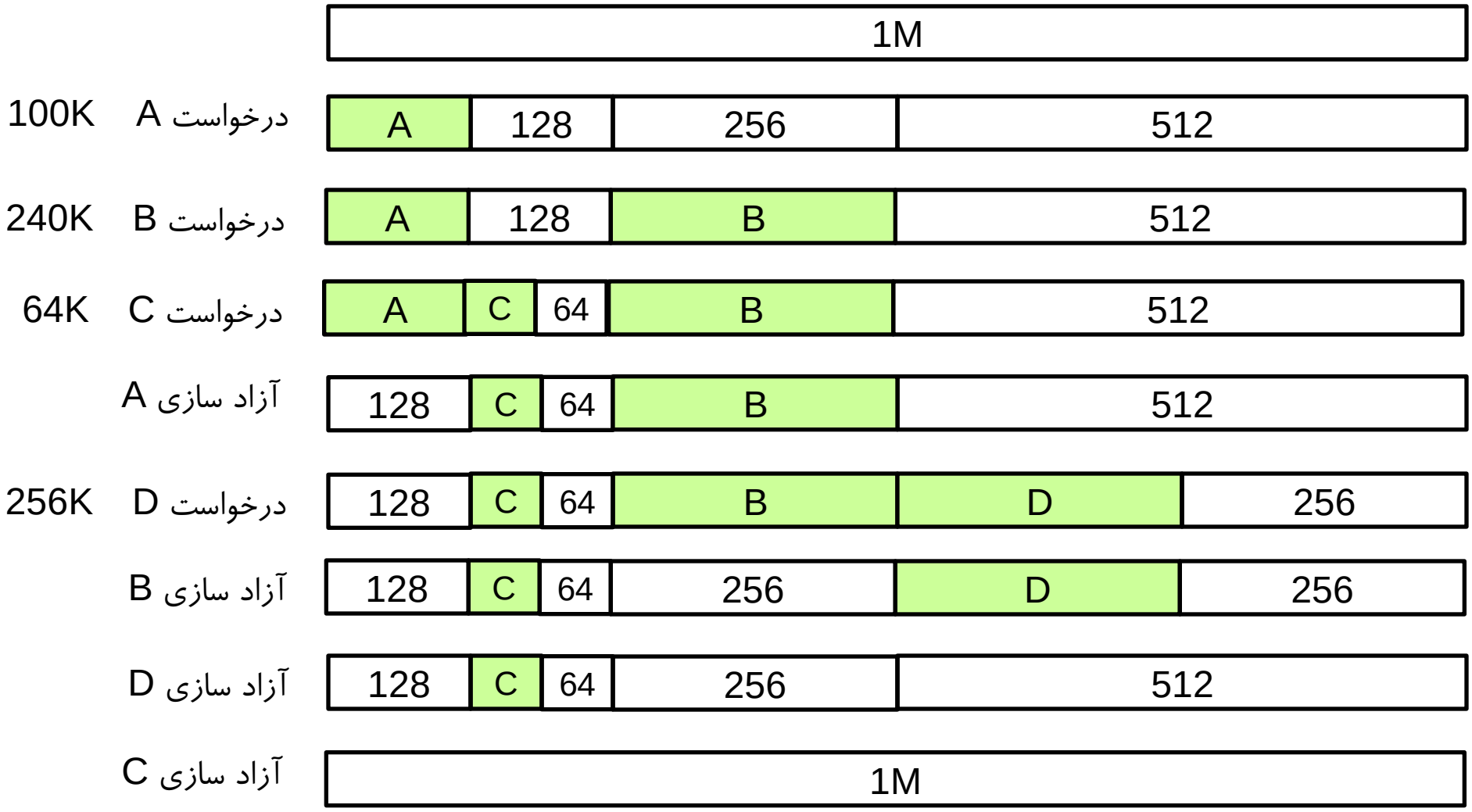
64

بخش‌بندی رفاقتی



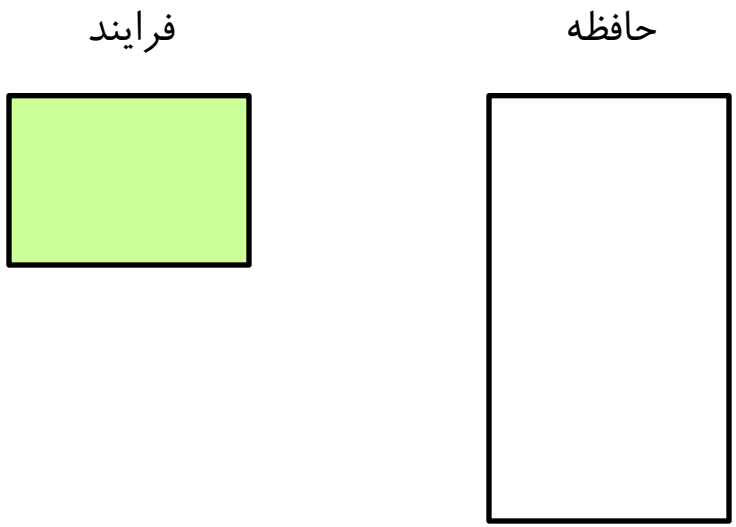
64

بخش‌بندی رفاقتی



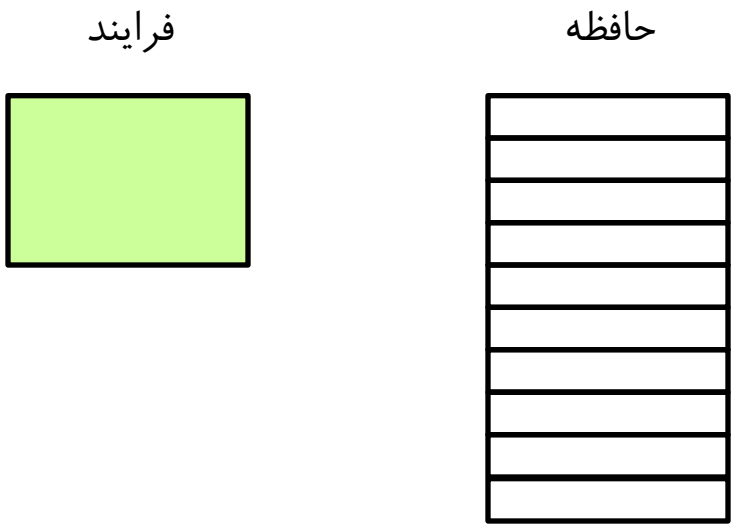
صفحه‌بندی

- در بخش‌بندی، برای فرایند فضای پیوسته می‌خواستیم و همین مشکل‌ساز بود
- این فرض را به هم می‌زنیم
  - یک اندازه صفحه از پیش معلوم داریم (مثلا مقادیر 1K، یا 2K یا 4K انتخاب‌های معمول هستند)



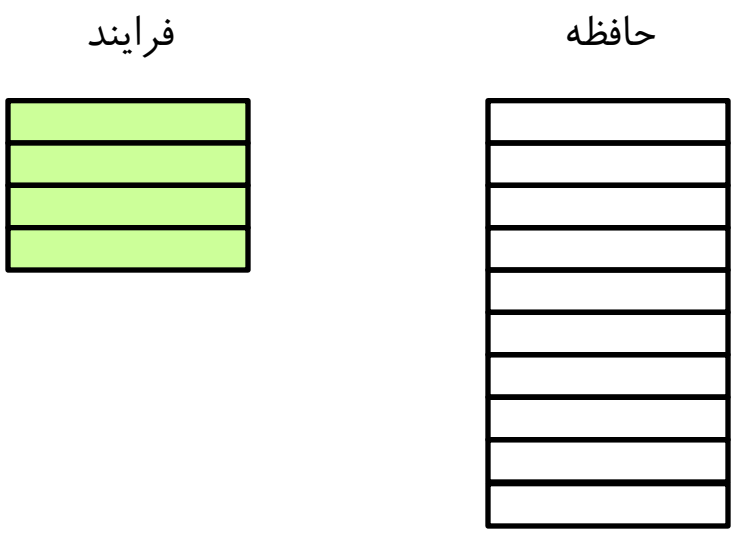
صفحه‌بندی

- در بخش‌بندی، برای فرایند فضای پیوسته می‌خواستیم و همین مشکل‌ساز بود
- این فرض را به هم می‌زنیم
  - یک اندازه صفحه از پیش معلوم داریم (مثلا مقادیر 1K، یا 2K یا 4K انتخاب‌های معمول هستند)
    - تمام حافظه را با این تقسیم بندی در نظر می‌گیریم



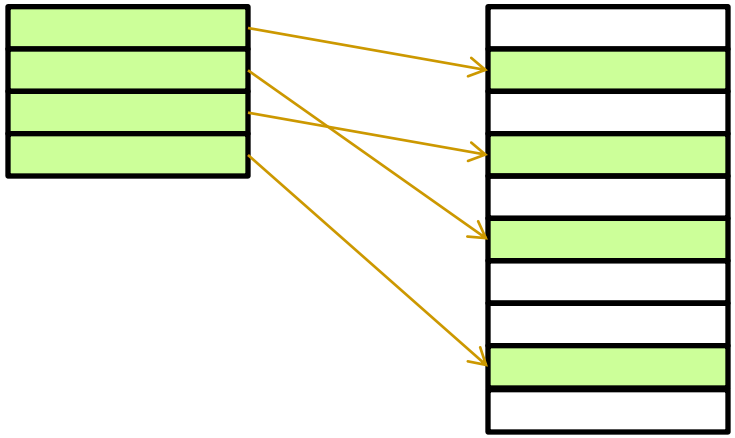
صفحه‌بندی

- در بخش‌بندی، برای فرایند فضای پیوسته می‌خواستیم و همین مشکل‌ساز بود
- این فرض را به هم می‌زنیم
  - یک اندازه صفحه از پیش معلوم داریم (مثلا مقادیر 1K، یا 2K یا 4K انتخاب‌های معمول هستند)
  - تمام حافظه را با این تقسیم بندی در نظر می‌گیریم
  - ضمنا فرایند را هم با همان اندازه صفحه تقسیم بندی می‌کنیم



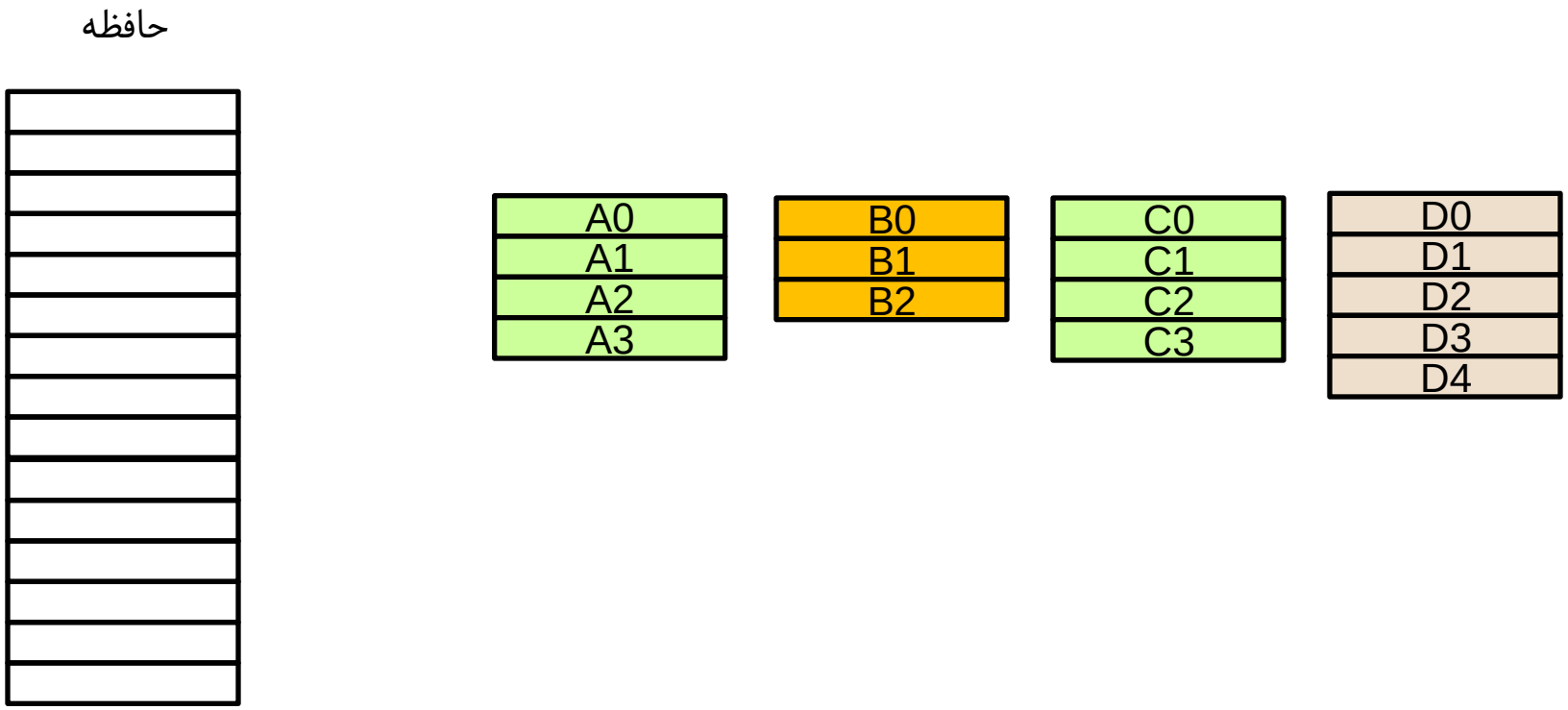
صفحه‌بندی

- در بخش‌بندی، برای فرایند فضای پیوسته می‌خواستیم و همین مشکل‌ساز بود
- این فرض را به هم می‌زنیم
  - یک اندازه صفحه از پیش معلوم داریم (مثلا مقادیر 1K، یا 2K یا 4K انتخاب‌های معمول هستند)
  - تمام حافظه را با این تقسیم بندی در نظر می‌گیریم
  - ضمنا فرایند را هم با همان اندازه صفحه تقسیم بندی می‌کنیم
  - صفحات (Pages) فرایند می‌توانند در قاب‌های (Frames) حافظه (هر کدام که خالی باشد قرار بگیرند (بدون لزوم رعایت ترتیب)



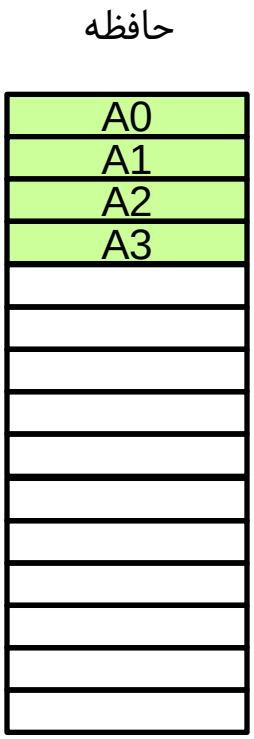
صفحه‌بندی

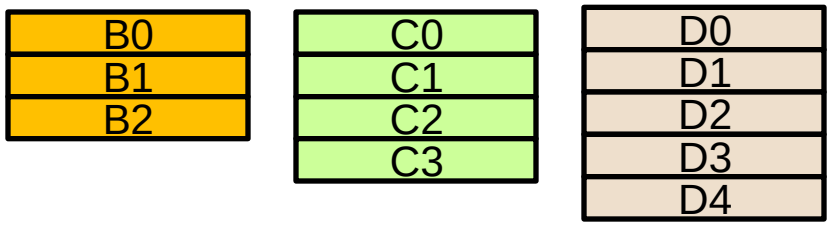
- مثال



صفحه‌بندی

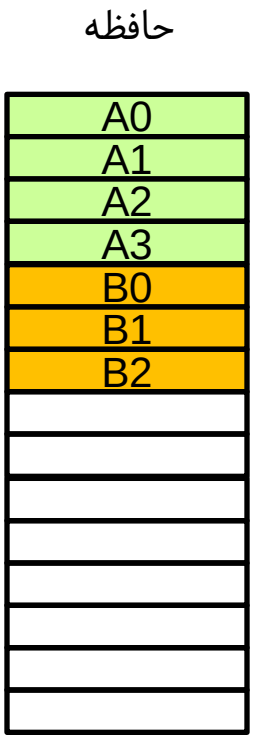
- مثال
  - فرایند A وارد می‌شود

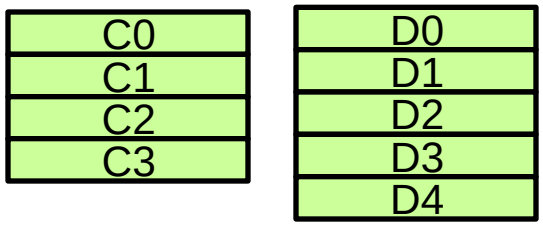




صفحه‌بندی

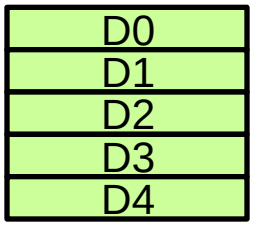
- مثال
  - فرایند B وارد می‌شود

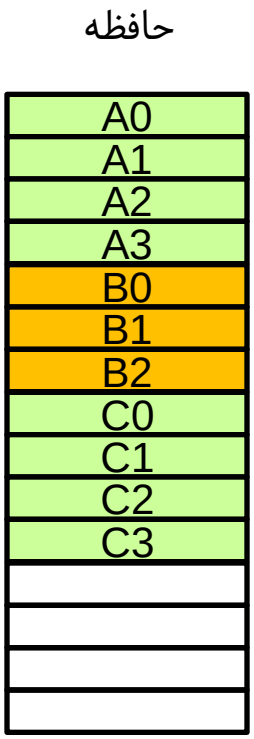




صفحه‌بندی

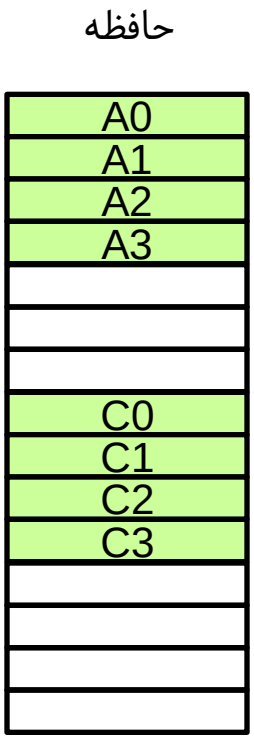
- مثال
  - فرایند C وارد می‌شود





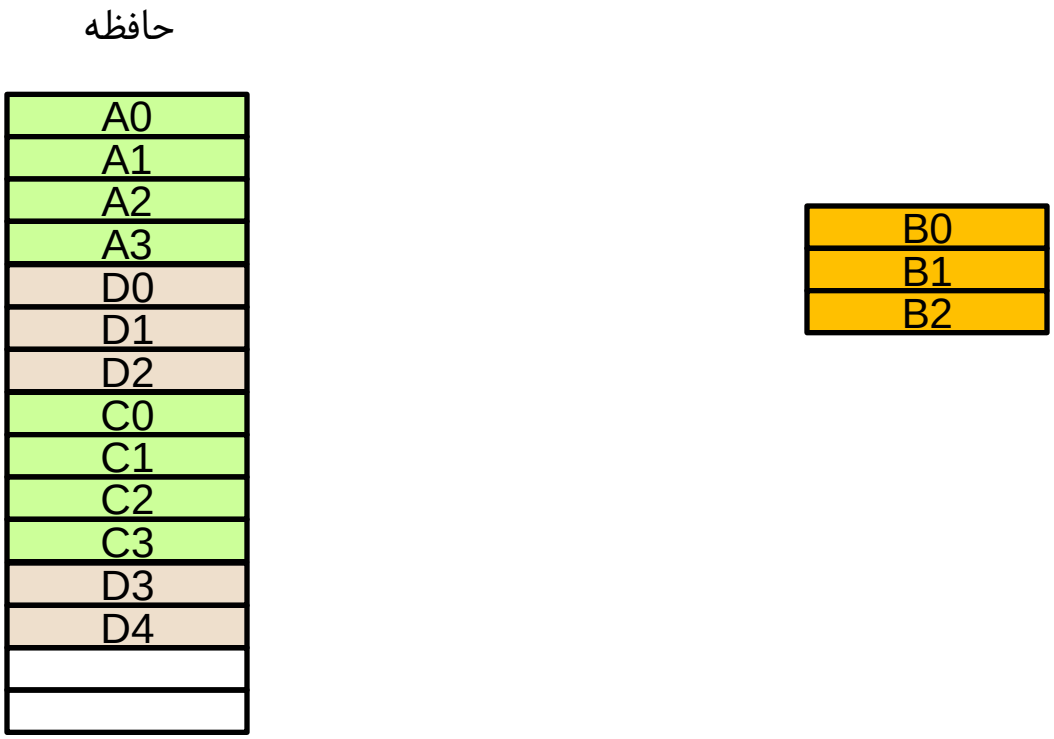
صفحه‌بندی

- مثال
  - برای فرایند D جای کافی وجود ندارد
    - B را خارج می‌کنیم



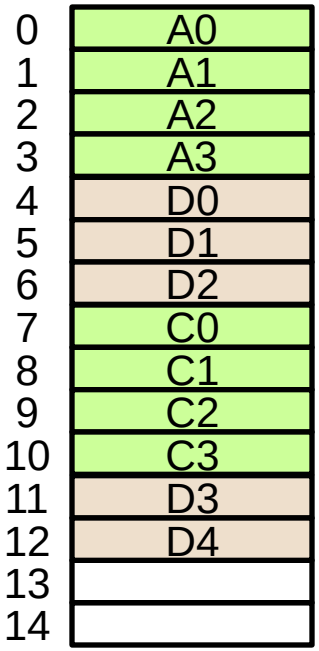
صفحه‌بندی

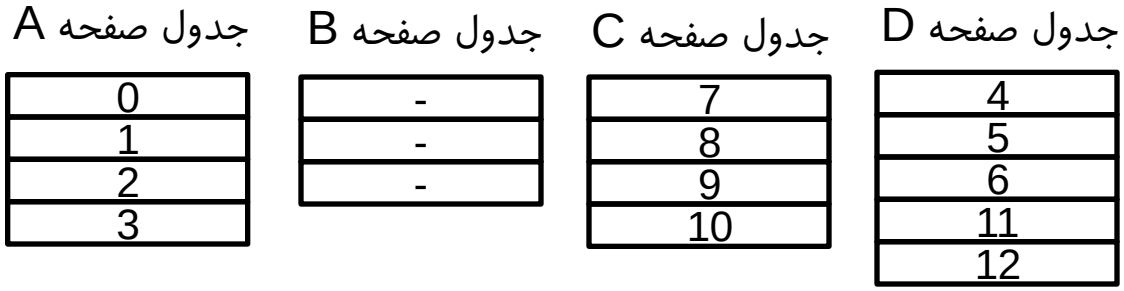
- مثال
  - D وارد می‌شود



صفحه‌بندی

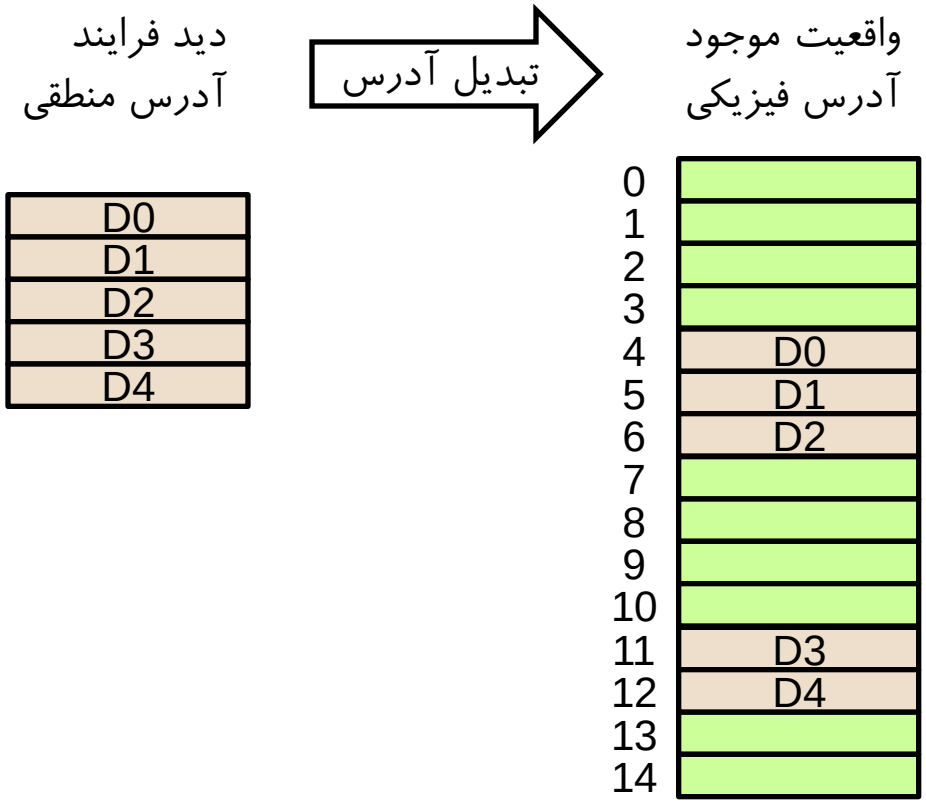
- در این روش صفحات فرایند در حافظه پراکنده هستند
- چطور بفهمیم صفحات فرایند کجا قرار دارند؟
  - یک جدول صفحه (Page Table) برای هر فرایند لازم است





صفحه‌بندی

- دید فرایند یک دید پیوسته است
  - از تکه تکه شدن خودش خبر ندارد
  - نیاز به نوعی تبدیل آدرس وجود دارد



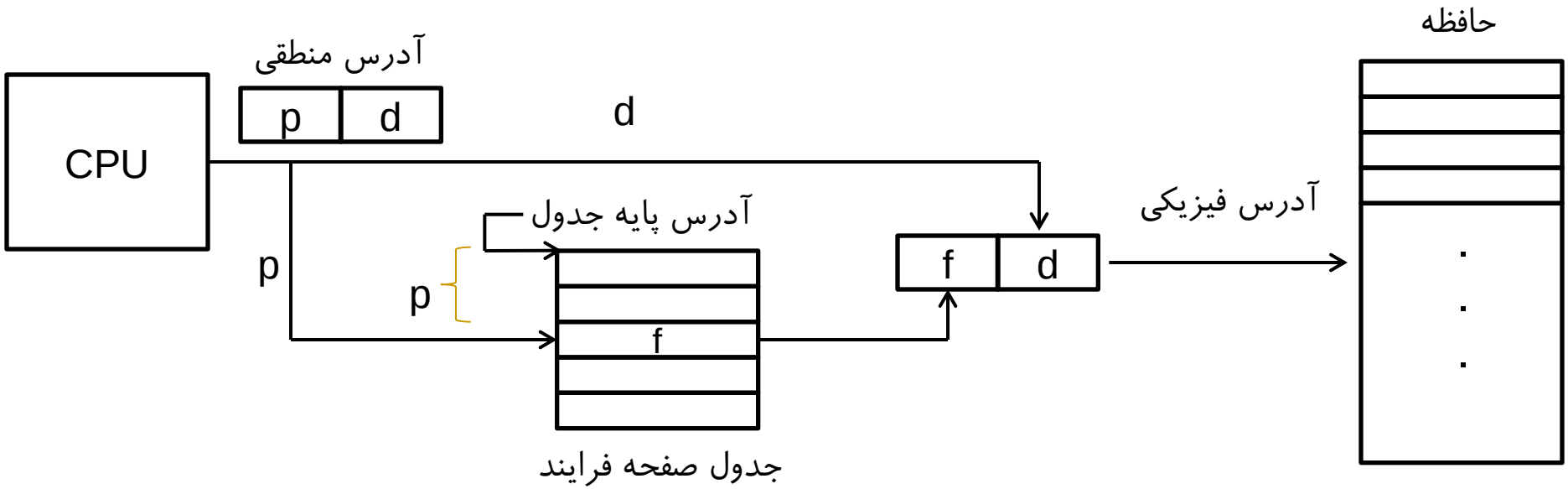
صفحه‌بندی

- آدرس از دید فرایند (آدرس منطقی) شامل دو قسمت است
  - شماره صفحه (Page)
  - مکان داخل آن صفحه (Displacement)

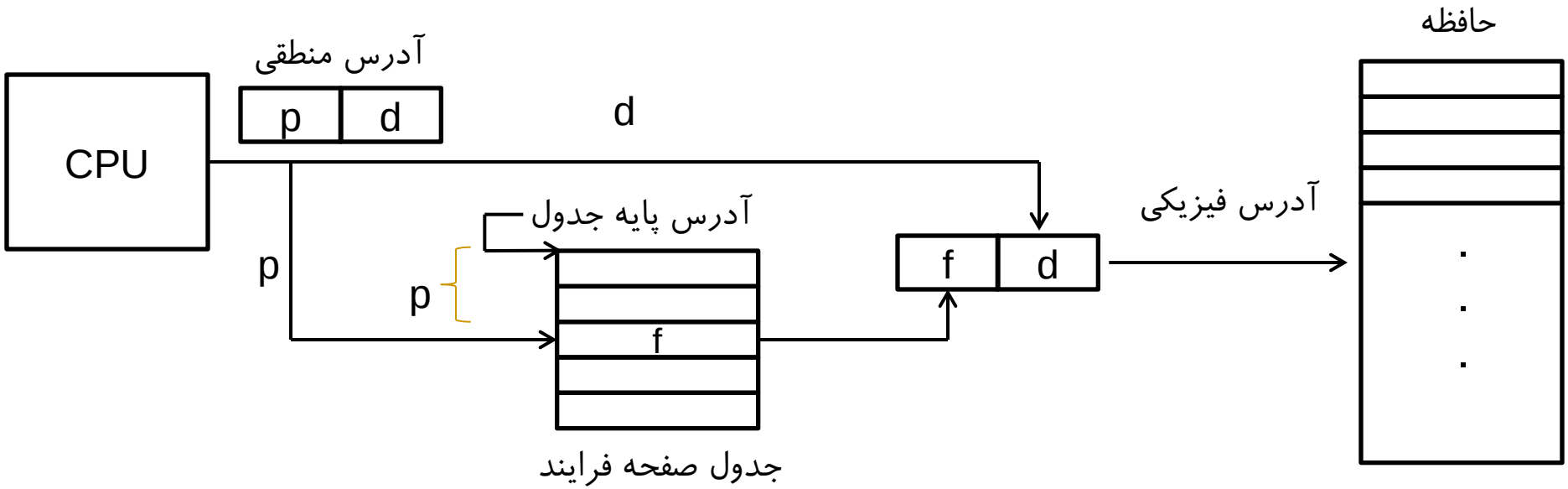
  - مثلا اگر 6 بیت به p و 10بیت به d اختصاص دهیم امکان داشتن حداکثر 64 صفحه به اندازه هر کدام 1K خواهیم داشت
  - در هر مراجعه فرایند به هر مکان از حافظه باید ابتدا تبدیل از آدرس منطقی به فیزیکی انجام شود سپس مراجعه انجام شود

صفحه‌بندی

- تبدیل آدرسی منطقی به فیزیکی



- تبدیل آدرسی منطقی به فیزیکی
  - این عملیات سخت افزاری انجام می‌شود (از دید کاربر مخفی است)
    - وظیفه سیستم عامل در این میان: ساختن جدول صفحه فرایندها
    - سخت افزار جدول را از کجا پیدا می‌کند؟
      - جدول در حافظه قرار دارد
      - هر جدول با آدرس شروعش (آدرس پایه) مشخص می‌شود که جزء PCB فرایند است



صفحه‌بندی

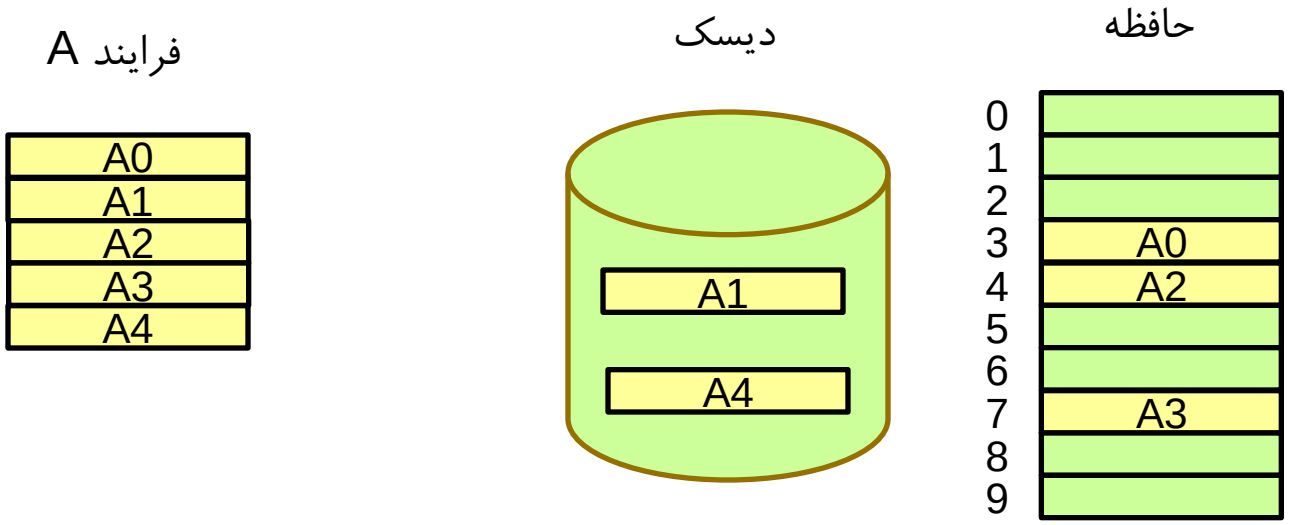
- صفحه‌بندی مشکل قطعه قطعه شدن را حل می‌کند
  - فقط در آخرین صفحه قطعه قطعه‌شدن داخلی کوچکی خواهیم داشت
  - اما آیا مشکل ما به رایگان حل شد؟
    - خیر
    - به جای هر دسترسی به حافظه اینک دو دسترسی نیاز خواهیم داشت
      - اولی برای مراجعه به جدول و محاسبه آدرس فیزیکی
      - دومی برای مراجعه به آن آدرس
    - نحوه مقابله با این مشکل
      - در پردازنده قسمتی به نام TLB (بافر دم دستی ترجمه) (Translation Lookaside Buffer) وجود دارد
      - وظیفه Cache کردن اطلاعات جداول را دارد
      - قبل از مراجعه به حافظه ابتدا به TLB رجوع می‌شود.
      - TLB تعدادی خیلی کم مدخل دارد ولی خیلی سریع است (چند ده مدخل دارد در حالی که جداول صفحه می‌توانند تا هزاران مدخل بزرگ باشند)
      - طبق اصل محلیت در اکثر موارد اطلاعات لازم در TLB یافت می‌شود

صفحه‌بندی

- انتخاب اندازه صفحه
  - خیلی کوچک ← کم شدن تکه تکه شدن داخلی ولی بزرگ شدن جدول صفحه
  - خیلی بزرگ ← کوچک شدن جدول صفحه ولی بزرگ شدن تکه تکه شدن داخلی
  - معمولا مقداری بین 1K الی 4K به صورت توانی از دو انتخاب می‌شود

حافظه مجازی

- مقدمه
  - تا به حال هر فرایندی که به حافظه می‌آوردیم، کل فرایند را به حافظه می‌آوردیم
  - در صورت کمبود حافظه کل فرایند را معلق می‌کردیم
  - حال این فرض را به هم می‌زنیم
    - لازم نیست کل برنامه و داده‌ها در حافظه فیزیکی باشد
    - فقط آن بخش که برای اجرا اکنون نیاز است کفایت می‌کند
  - مثال

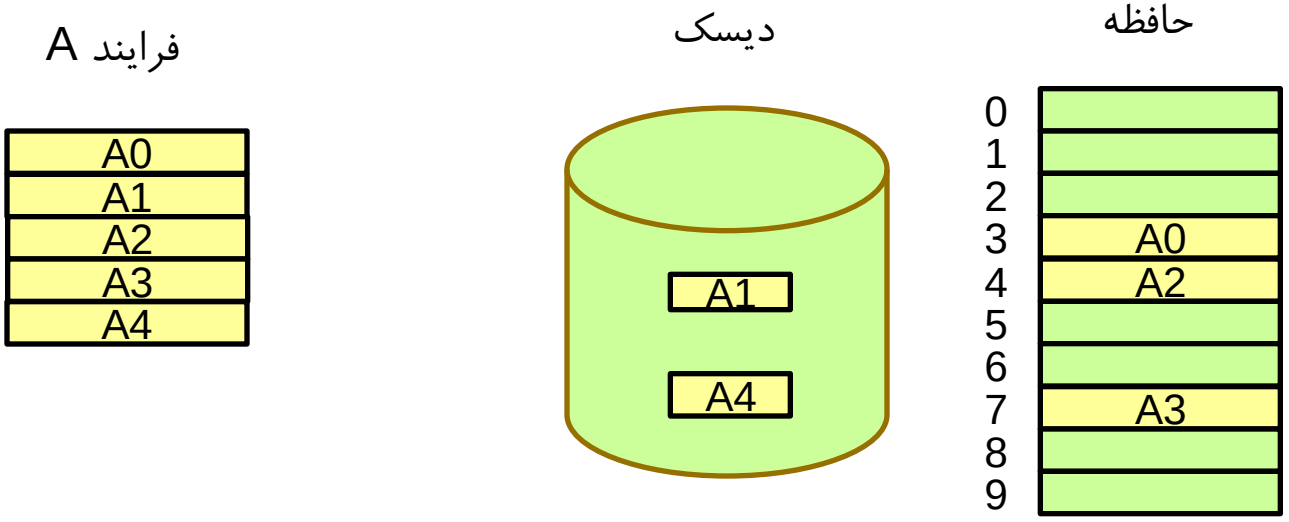


حافظه مجازی

- برای مدیریت این تغییر یک بیت به جدول صفحه می‌افزاییم
  - بیت P حاضر یا غایب بودن صفحه در حافظه

جدول صفحه A

	frame	P
0	3	1
1	-	0
2	4	1
3	7	1
4	-	0

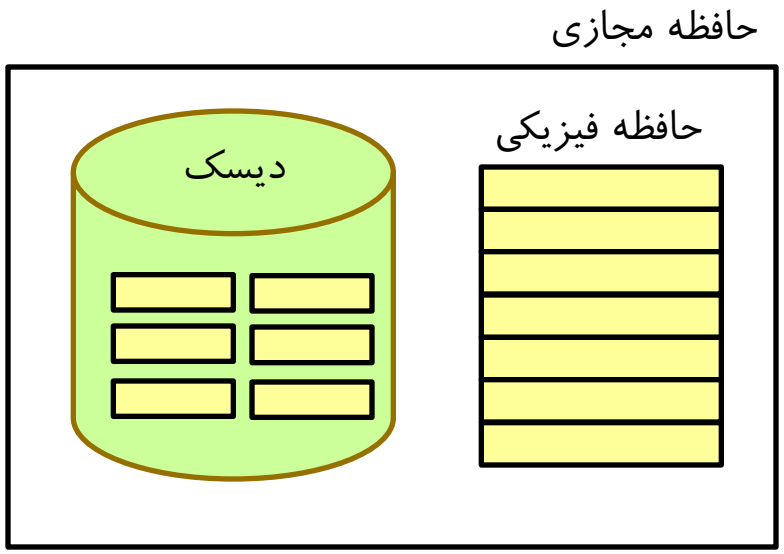


حافظه مجازی

- روند دسترسی به حافظه مجازی
  - ابتدا به TLB مراجعه می‌شود
  - ممکن است شماره قاب در TLB یافته نشود در این صورت به جدول صفحه واقع در حافظه مراجعه می‌شود
  - اگر در آن هم نبود (بیت P صفر بود) دیگر کار سخت‌افزار تمام شده. یک وقفه به نام Page Fault (فقدان صفحه) رخ می‌دهد
  - سیستم عامل وارد عمل می‌شود و صفحه را از دیسک به حافظه بار می‌کند
  - اگر فضای خالی در حافظه نباشد یکی از صفحه‌های موجود از حافظه اخراج می‌شود تا صفحه جدید بتواند به جای آن وارد شود
  - برای انتخاب صفحه اخراجی سیاست‌های مختلفی وجود دارد
  - وقوع فقدان صفحه هزینه زمانی بالایی تحمیل می‌کند. چرا که دیسک از حافظه خیلی کندتر است. این امر باید خیلی کم اتفاق بیفتد و همین طور هم هست. چرا؟

حافظه مجازی

- دیدیم در روند دسترسی ذکر شده گاهی سیستم عامل وارد عمل می‌شود
  - اما این از دید برنامه‌نویس (فرایند) پنهان است. او فقط حافظه مجازی را می‌بیند
  - تعداد بیت‌های آدرس مجازی می‌تواند بیشتر از آدرس فیزیکی باشد
  - این یعنی ساختن یک تصویر بزگ‌تر از حافظه با یاری گرفتن از دیسک
  - مثال
    - پردازنده‌های 32 بیتی اینتل
      - آدرس فیزیکی 32 بیت
        (حداکثر 4GB حافظه فیزیکی)
      - آدرس منطقی 45 بیت
        (حداکثر 32TB حافظه مجازی)



حافظه مجازی

- علاوه بر بیت P می‌توان بیت‌های دیگری به جدول صفحه افزود
  - مثلا بیت M نشان بدهد آیا صفحه تغییری داشته یا نه (چرا مفید است؟)
  - بیت‌های دسترسی که آیا صفحه قابل نوشتن هست یا نه

حافظه مجازی

- کندی دیسک نسبت به حافظه چه اثری بر زمان دسترسی حافظه مجازی خواهد داشت؟
  - بستگی به درصد وقوع فقدان صفحه دارد
- مثال
  - یک سیستم حافظه مجازی با صفحه‌بندی با این خصوصیات داریم
    - زمان دسترسی به حافظه: 60ns
    - زمان انتقال هر صفحه از دیسک به حافظه: 25ms
    - زمان دسترسی به TLB: 5ns
    - نسبت اصابت TLB: 95%
  - میانگین زمان دسترسی را برای دو حالت زیر حساب کنید
    - الف) درصد فقدان صفحه 0.1% باشد
    - ب) درصد فقدان صفحه 0.001% باشد

حافظه مجازی

- زمان دسترسی به حافظه: 60ns
- زمان انتقال هر صفحه از دیسک به حافظه: 25ms
- زمان دسترسی به TLB: 5ns
- نسبت اصابت TLB: 95%

- میانگین زمان دسترسی را برای دو حالت زیر حساب کنید
  - الف) درصد فقدان صفحه 0.1% باشد
  - ب) درصد فقدان صفحه 0.001% باشد

اگر فقدان صفحه رخ ندهد

$$t1 = 0.95(5+60) + 0.05(5+60+60) = 68ns$$

TLB حافظه TLB جدول حافظه

اگر فقدان صفحه رخ دهد

$$t2 = (5ns + 60ns + 25ms + 60ns + 60ns) = 25ms$$

TLB (در آن نیست) جدول (در آن نیست) دیسک دوباره جدول حافظه

الف) تأخیر بیش از حد قابل قبول $t = 0.999t1 + 0.001t2 = 25us$

ب) تأخیر قابل قبول $t = 0.99999t1 + 0.00001t2 = 318ns$

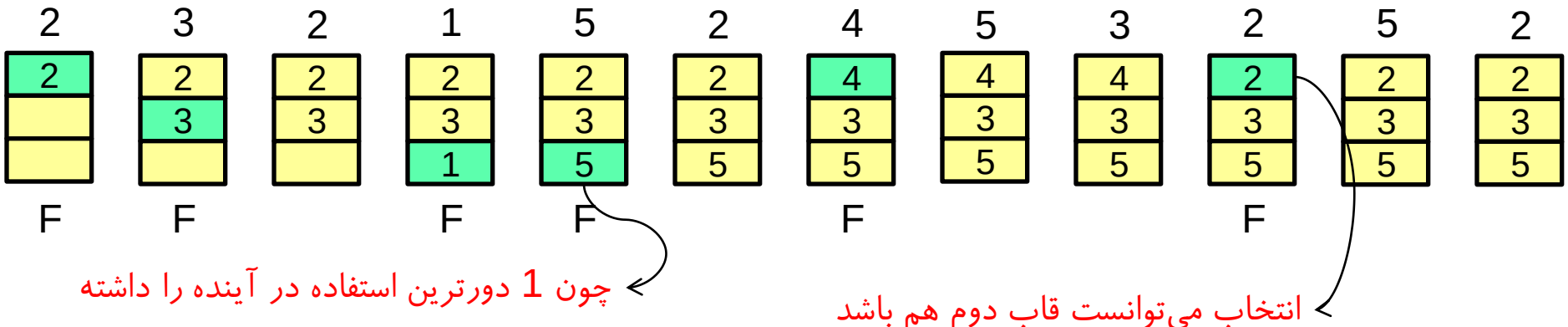
حافظه مجازی

- در مثال دیدیم که درصد وقوع فقدان بر میانگین زمان دسترسی حافظه مؤثر است
- برای جایگزینی صفحه (اگر همه قاب‌ها پر باشند) سیاست‌های وجود دارد
- سیاست‌ها به لحاظ میزان فقدان و نیز هزینه پیاده‌سازی متفاوتند
- سیاست‌های جایگزینی
  - بهینه
  - حداقل استفاده اخیر (LRU) (Least Recently Used)
  - خروج به ترتیب ورود (FIFO)
  - ساعت (Clock)
- بررسی سیاست‌ها در قالب مثال..
  - حافظه با سه قاب (اندکی کوچک است!! فقط برای مثال)
  - دسترسی به صفحات فرایند طبق دنباله روبرو 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

مثال سیاست جایگزینی

2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

- سیاست بهینه
  - در هنگام جایگزینی، آن صفحه‌ای اخراج می‌شود که دورترین استفاده در آینده را داشته باشد
  - بهترین نتیجه را می‌دهد اما عملی نیست چون قدرت پیشگویی لازم دارد
  - به همین دلیل سعی می‌کنند رفتار آینده را از طریق رفتار گذشته تقریب بزنند (اسلایدهای بعد)

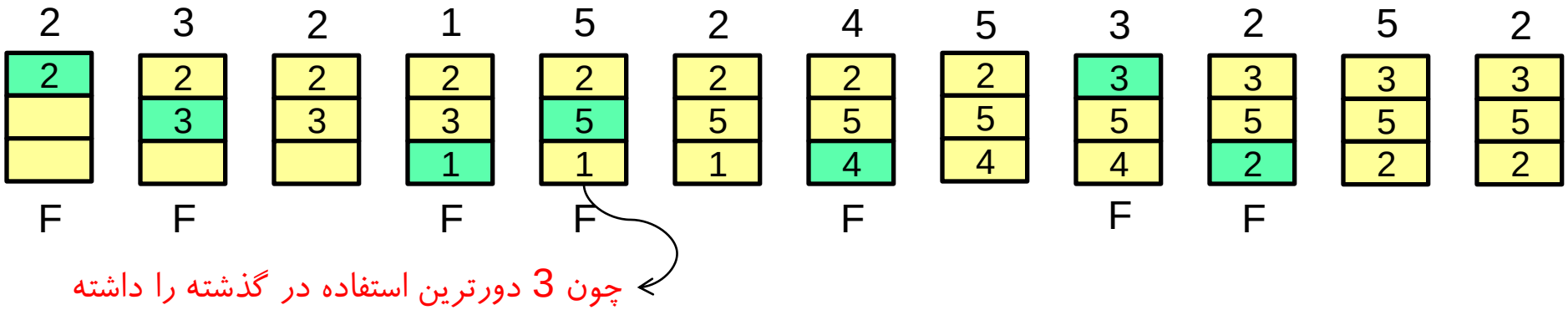


6 Page Faults

مثال سیاست جایگزینی

2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

- سیاست LRU
  - در هنگام جایگزینی، آن صفحه‌ای اخراج می‌شود که دورترین استفاده در گذشته را داشته باشد
  - توجیه: چنین صفحه‌ای احتمالا در آینده نزدیک هم استفاده نخواهد شد
  - نتیجه‌اش خوب است ولی پیاده‌سازی مشکلی دارد (نیاز به برچسب زمانی یا پشته)

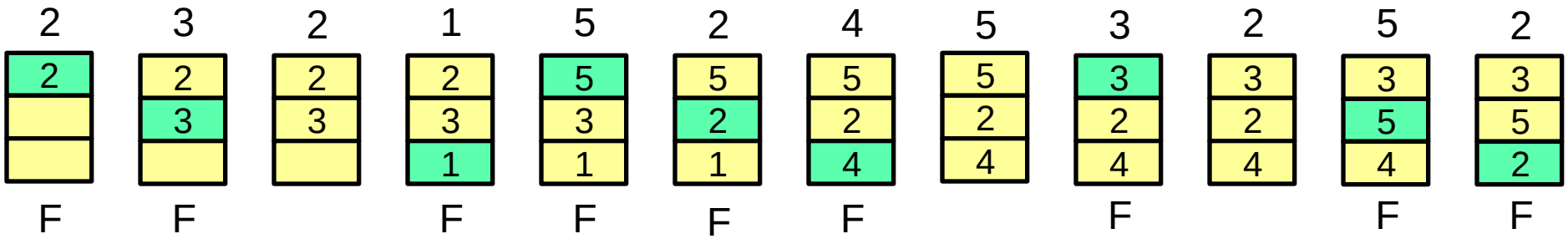


7 Page Faults

مثال سیاست جایگزینی

2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

- سیاست FIFO
  - هرکدام زودتر وارد شده باشد زودتر هم خارج می‌شود
  - توجیه: صفحه‌ای که مدت‌ها پیش به حافظه آورده شده احتمالا اکنون بی‌استفاده است
  - این توجیه اغلب غلط از آب در می‌آید و معمولا بدترین نتیجه را می‌دهد
  - ساده‌ترین پیاده سازی را دارد (با یک اشاره‌گر چرخشی)

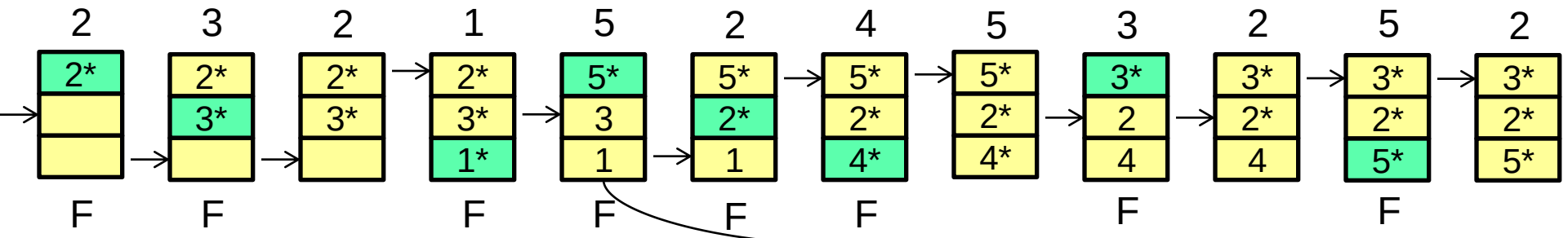


9 Page Faults

مثال سیاست جایگزینی

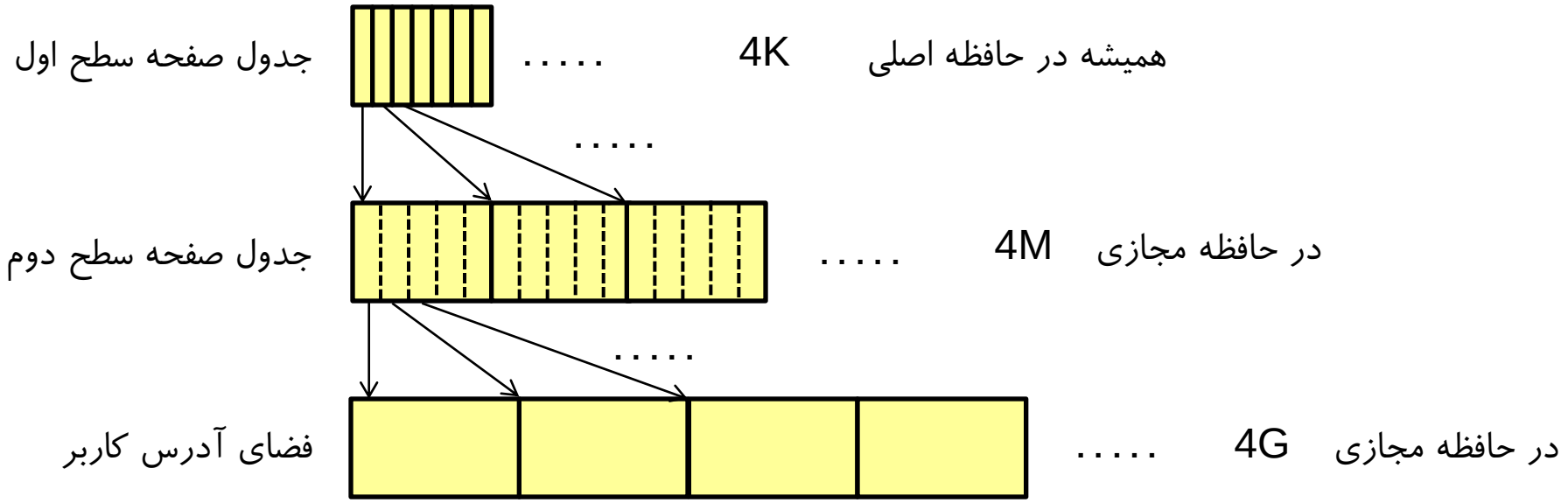
2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

- سیاست ساعت
  - تقریبی از LRU
  - یک بیت used برای هر صفحه در نظر می‌گیرد. وقتی به صفحه مراجعه شود یک می‌شود
  - یک عقربه اخراج به طور گردشی به نوبت به صفحات اشاره می‌کند
  - اگر نیاز به جایگزینی باشد صفحه‌ای که عقربه به آن اشاره کند اخراج می‌شود. مگر این که بیت used آن یک باشد که در این صورت خطر از بیخ گوش صفحه گذشته است
  - البته با وجود عدم اخراج، بیت used صفحه صفر می‌شود و تا دور بعدی فرصت دارد در اثر مراجعه یک شود وگرنه اخراج
  - اگر جایگزینی نداشته باشیم عقربه حرکت نمی‌کند
  - مثالش باید در عمل بررسی شود چون بیت used به رفتار برنامه بستگی دارد. در اینجا فرض شده بعد از آوردن صفحه به حافظه دسترسی به آن انجام می‌شود



صفحه‌بندی دو سطحی

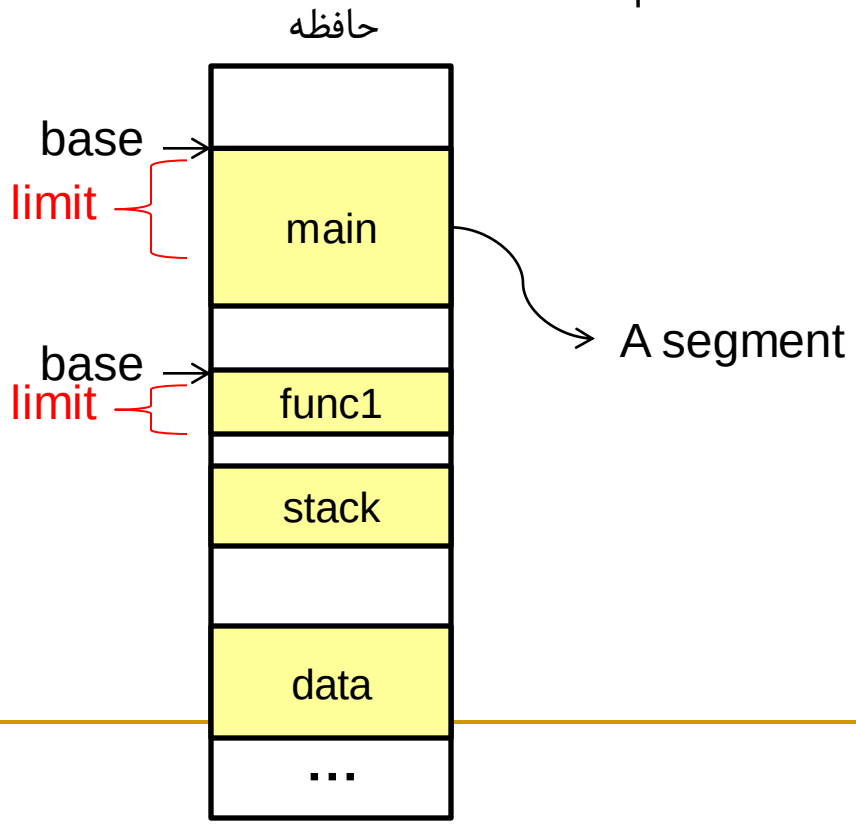
- بعضی فرایندها ممکن است تا چندین گیگابایت هم بزرگ شوند
- این یعنی جدول صفحه تا اندازه چندین مگابایت بزرگ می‌شود
- به همین خاطر گاهی لازم می‌شود جداول صفحه خود در حافظه مجازی باشند
- یعنی آن‌ها هم در معرض صفحه‌بندی قرار بگیرند
- این طراحی را مشکل می‌کند. برخی از ساختار دوسطحی استفاده می‌کنند. مثل Pentium



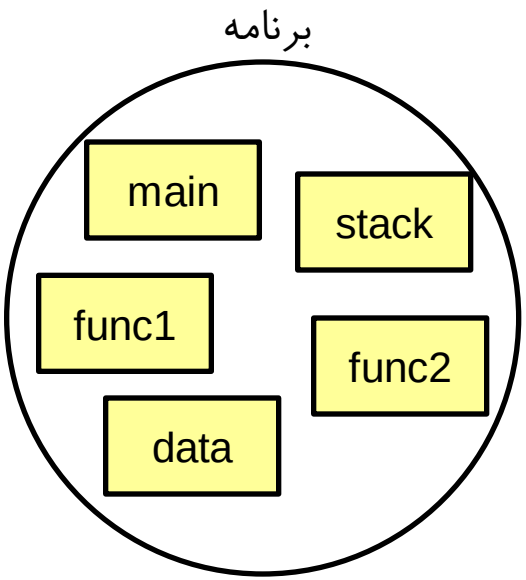
- یک مراجعه به حافظه می‌افزاید (سربار بیشتر)

قطعه بندی

- در صفحه‌بندی، فرایند را به صفحات کوچک تقسیم می‌کردیم
  - قطعات برابر بود
  - از دید برنامه‌نویس پنهان بود
- در قطعه‌بندی، فرایند با نظر برنامه‌نویس به قطعاتی تقسیم می‌شود

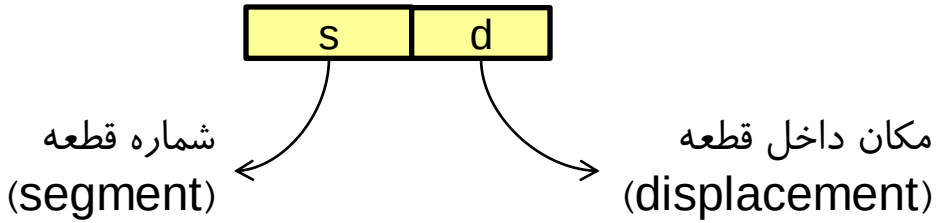


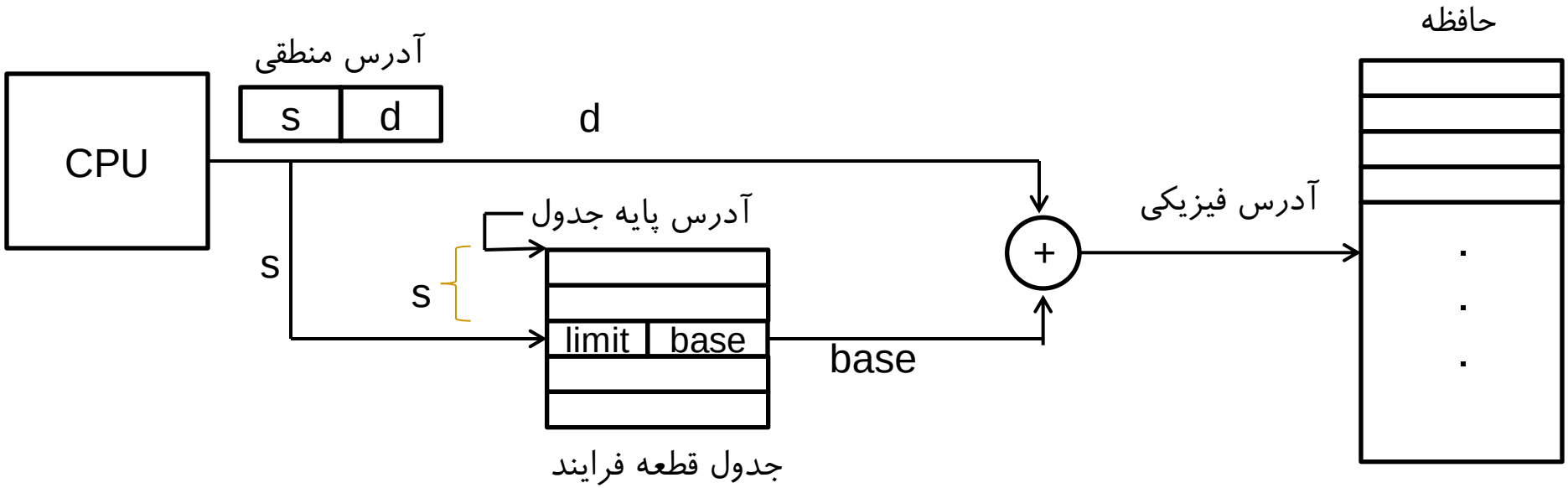
- در قطعه بندی انداز قطعات لزوما برابر نیست
- مزیت: با منطق برنامه‌نویس بیشتر جور در می‌آید و از دید کاربر پنهان نیست
- ایراد: تکه تکه شدن خارجی (اما نه به شدت زمانی که به کل فرایند فضایی پیوسته می‌دادیم. چون به قطعات کوچک‌تر تقسیم شده است)



قطعه‌بندی

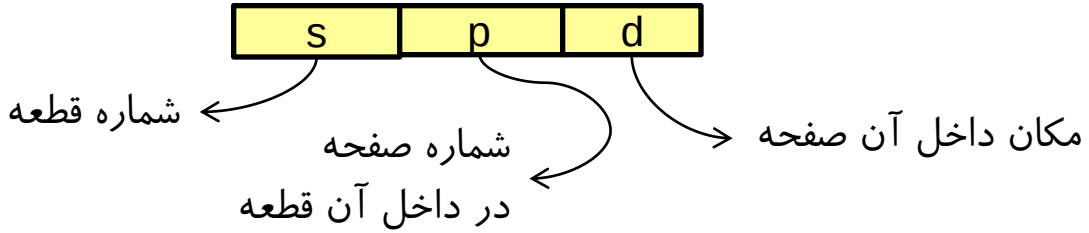
- در روش قطعه‌بندی آدرس منطقی شامل دو قسمت است

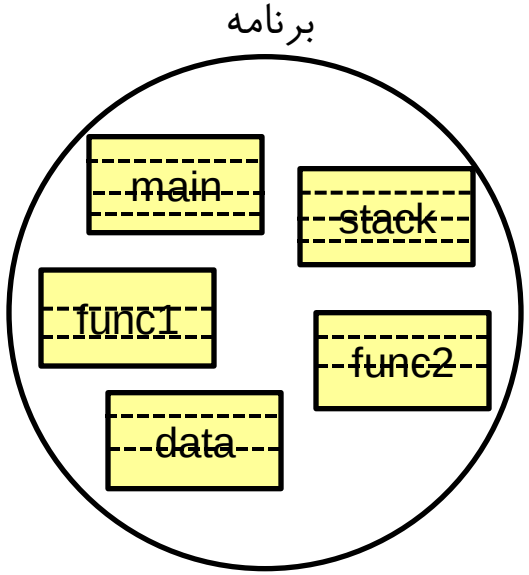




ترکیب قطعه‌بندی با صفحه‌بندی

- در قطعه‌بندی فضایی پیوسته در اختیار هر سگمنت قرار می‌دادیم
- در این جا هر سگمنت را خود تقسیم شده به صفحاتی در نظر می‌گیریم
- آدرس منطقی شامل سه قسمت خواهد بود





روند دسترسی در ترکیب قطعه‌بندی با صفحه‌بندی