Operating Systems
سیستمهای عامل

مجموعه اسلایدهای شماره ۷

دکتر خانمیرزا
h.khanmirza@kntu.ac.ir
دانشکده کامپیوتر
دانشگاه صنعتی خواجه نصیرالدین طوسی

زمان‌بندی

- گفته شد که ریسمانهای آماده اجرا در صف ready قرار دارند و ریسمان بعدی برای اجرا توسط یک ماجول در سیستم عامل به نام زمان بند انتخاب می‌شود

- علاوه بر این اینکه از بین صفهای waiting و در هر صف از بین ریسمانهای در حال انتظار کدام برای پردازش انتخاب شود نیز از وظایف زمان‌بند است

- نهایت هدف این ماجول این است که پردازنده سیستم بنوعی دائم در حال استفاده باشد تا به که برخی پارامترهای سیستمی بهینه شود

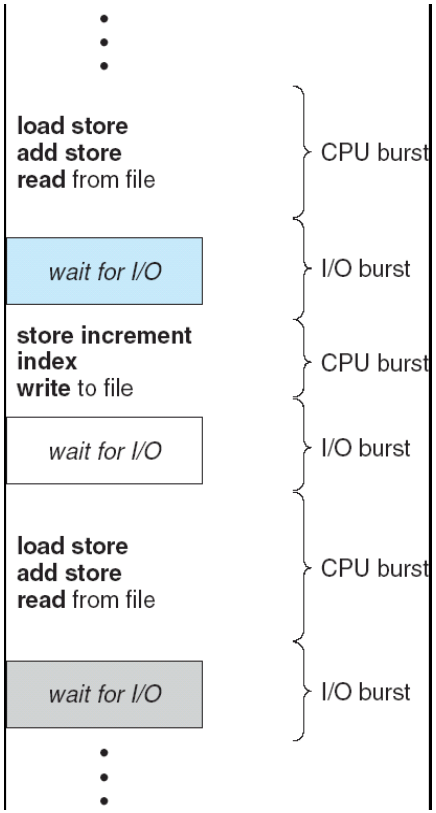
اهداف زمان‌بندی

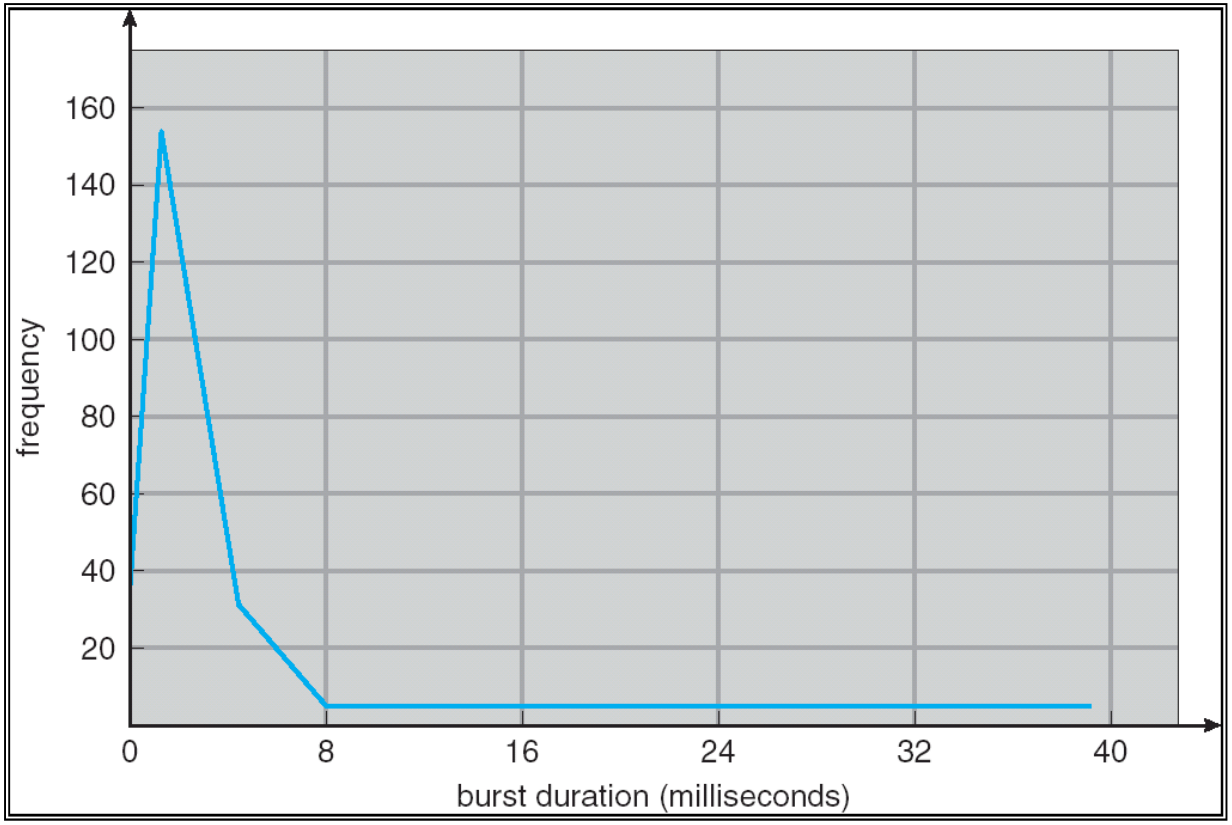
- کاهش زمان پاسخ‌دهی (response time)
  - زمان اولین پاسخ به یک کار (task)
  - این همان چیزی است که کاربر در برنامه‌های تعاملی احساس می‌کند
- افزایش توان عملیاتی (throughput)
  - تعداد کارهای پایان یافته در واحد ثانیه افزایش یابد
  - برای افزایش توان عملیاتی باید دو عامل را بهینه کرد
    - کاهش تعداد تعویض زمینه‌ها
    - استفاده بهینه از منابع نظیر پردازنده، دیسک و ....
- این دو هدف با هم مرتبط هستند ولی ممکن است با هم در تضاد باشند
  - برای پاسخ‌دهی سریعتر ممکن است نیاز به تعویض زمینه بیشتر باشد که این کار باعث کاهش توان عملیاتی می‌شود

اهداف زمان‌بندی

- کاهش زمان پاسخ‌دهی کامل (turnaround time - completion time)
  - مدت زمان ارسال کار برای انجام تا پایان یافتن کامل کار
- کاهش مدت انتظار (waiting time)
  - کل مدت زمانی که یک کار در صف انتظار است
- انصاف (fairness)
  - منابع مخصوصا پردازنده به طور منصفانه به کارها اختصاص داده شود
  - البته انصاف باید در هر سیستم دقیقا تعریف شود:
    - بین کاربران؟ یعنی هر کاربر بازاء برنامه‌هایش روی یک سیستم یک مقدار مساوی با یکی دیگر بگیرد
    - بین فرآیندها؟ اگر یکی از فرآیندها تعداد ریسمان بیشتری داشته باشد و یکی کمتر؟
    - بین ریسمانها؟ اگر فرآیندی چند هزار ریسمان داشته باشد پردازنده بیشتری در اختیار خواهد داشت

رفتار استفاده پی‌درپی (burst) از پردازنده





- ریسمانها دائما بین حالت اجرا و IO تغییر وضعیت می‌دهند
- ریسمانها عموما برای مدت مشخصی با پردازنده کار می‌کنند و بعد کار IO انجام می‌دهند.
  - به بیان دیگر ممکن ریسمانها از تمام زمانی که در اختیار آنها گذاشته می‌شود استفاده نکنند
- به بازه‌ای که ریسمانها از پردازنده به صورت پی‌درپی استفاده می‌کنند CPU Burst گفته می‌شود
- نمودار فوق نشان می‌دهد که burstهای طولانی کمتر مورد استفاده قرار می‌گیرند

اهداف زمان‌بندی

- با مکانیزم تقسیم زمانی (یا همزمانی) ممکن است قبل از آنکه بازه‌ی پردازشی یا استفاده پی‌درپی از پردازنده ریسمان به اتمام برسد پردازنده را از ریسمان گرفته شود.

- در واقع نمودار قبل نشان می‌دهد که عمدتا این طور نیست
  - مثلا خیلی از برنامه‌ها که تعاملی هستند عموما منتظر یک کار کاربر هستند و بعد از آن کاری انجام میدهند و بعد منتظر واکنش بعدی کاربر میمانند.

- زمان بندها معمولا یک ترکیبی از انواع کارها را دارند و باید از پس همه برآیند.

الگوریتمهای زمان‌بندی

الگوریتم صف (FCFS)

- عبارت FCFS مخفف First Come First Serve است که همان مفهوم صف عادی را دارد
- در این روش عملاً فرآیندها به ترتیب قرار گرفتن در صف در پردازنده اجرا شده و تا پایان در پردازنده می‌مانند.
- دقت کنید که با این مدل از زمان بندی امکان همزمانی وجود ندارد

الگوریتم صف (FCFS)

Process	Burst Time	Order
P1	24	0
P2	3	1
P3	3	2

- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: (0 + 24 + 27)/3 = 17
- Average Completion time: (24 + 27 + 30)/3 = 27

الگوریتم صف (FCFS)

Process	Burst Time	Order
P1	24	3
P2	3	1
P3	3	2

- Waiting time for $P_1 = 6; P_2 = 0, P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3 VS. 17
- Average Completion time: (3 + 6 + 30)/3 = 13 VS. 27
- بهترین زمان پاسخ‌دهی برای صف زمانی بدست می‌آید که کارها به ترتیب طول مدت اجرا از کوچک به بزرگ مرتب شده باشند

الگوریتم صف (FCFS)

- ویژگیهای الگوریتم صف

- ساده است

- ولی دارای اثر کاروانی (convoy effect) است
- در این اثر فرآیندهای کوتاه پشت سر فرآیندهای بلند گیر می‌کنند

الگوریتم نوبتی (round robin)

- فرآیندها به ترتیب اجرا شده ولی به هر فرآیند یک مقدار مشخص از زمان اختصاص داده می‌شود
- این مدت زمان ثابت با نام کوانتوم (quantum) شناخته می‌شود.

- در حقیقت برای از بین بردن اثر کاروانی در مواجهه با ریسمانهایی با burst طولانی این روش پیشنهاد شده است

- در صورتی که پردازش یک ریسمان در مدت زمان کوانتوم پایان نیابد پردازنده از آن گرفته شده و ریسمان به انتهای صف آماده‌ها گذاشته می‌شود و باید منتظر دور بعدی شود
- بنابراین برای هر بار اجرا باید برای حداکثر q(n-1) واحد زمانی صبر کند

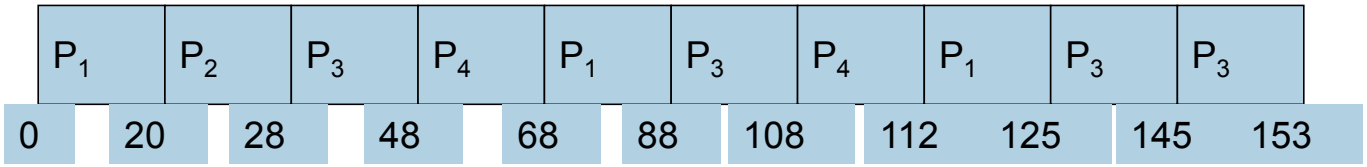
الگوریتم نوبتی (round robin)

- مقدار کوانتوم در کارآیی این الگوریتم نقش مهمی دارد

- کوانتوم بزرگ
  - عملا به سمت رفتار الگوریتم صف نزدیک می‌شود
  - زمان پاسخ‌دهی افزایش می‌یابد
  - توان عملیاتی بهتر می‌شود
    - چون تعداد کارهایی که در یک کوانتوم اجرا شده و به پایان می‌رسند افزایش می‌یابد
  - سربار اجرایی کمتر می‌شود
    - تعداد تعویض زمینه‌ها کاهش می‌یابد
    - از منابع استفاده بهینه می‌شود

الگوریتم نوبتی (round robin)

Process	Burst Time	Order
P1	53	1
P2	8	2
P3	68	3
P4	24	4



- q = 20
- Waiting time for
  - P_1=(68-20)+(112-88)=72
  - P_2=(20-0)=20
  - P_3=(28-0)+(88-48)+(125-108)=85
  - P_4=(48-0)+(108-68)=88
- Average waiting time = (72+20+85+88)/4=66¼
- Average completion time = (125+28+153+112)/4 = 104½

مقایسه الگوریتم نوبتی و صف

- فرض کنید ۱۰ کار داریم که هر کدام ۱۰۰ واحد زمانی طول میکشد. اگر مقدار کوانتوم ۱ باشد و سربار تعویض زمینه صفر باشد:

زمان اتمام		
Job #	FIFO	RR
1	100	991
2	200	992
...	...	...
9	900	999
10	1000	1000

- هر دو زمان پایان کارهای یکسانی دارند
- اما زمان پاسخ‌دهی الگوریتم نوبتی بسیار بد است

مقایسه الگوریتم نوبتی و صف

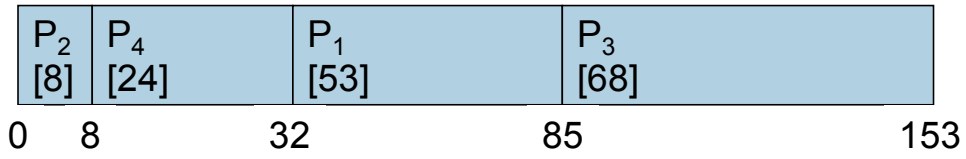
- برای صف:
  - $W_{p1} = 0$
  - $W_{p2} = 100$
  - $W_{p3} = 200$
  - ...
  - $W_{p10} = 900$
- $W_{avg} = 4500/10 = 450$
- $C_{avg} = 5500/10 = 550$

- برای نوبتی:
  - if q = 1 ➔
    - Wp1 = 0 + (10-1) + (20-11) + ... □
    - Wp1 = 0 + 99 * 9 = 891
    - Wp2 = 1 + 99 * 9 = 892
    - Wp3 = 2 + 99 * 9 = 893
    - ....
    - Wp10 = 9 + 99 * 9 = 900
  - $W_{avg} = \frac{\sum Wp_i}{no\ of\ processes} = \frac{8955}{10} = 895.5$
  - $C_{avg} = \frac{\sum c_i}{no\ of\ processes} = \frac{9955}{10} = 995.5$

مقایسه الگوریتم نوبتی و صف

- فرض

Best FCFS:



	Quantum	P_1	P_2	P_3	P_4	Average
Wait Time	Best FCFS	32	0	85	8	31¼
	Q = 1	84	22	85	57	62
	Q = 5	82	20	85	58	61¼
	Q = 8	80	8	85	56	57¼
	Q = 10	82	10	85	68	61¼
	Q = 20	72	20	85	88	66¼
	Worst FCFS	68	145	0	121	83½
Completion Time	Best FCFS	85	8	153	32	69½
	Q = 1	137	30	153	81	100½
	Q = 5	135	28	153	82	99½
	Q = 8	133	16	153	80	95½
	Q = 10	135	18	153	92	99½
	Q = 20	125	28	153	112	104½
	Worst FCFS	121	153	68	145	121¾

مقایسه الگوریتم نوبتی و صف

- زمانی که کارها را به ترتیب کوتاهترین به بلندترین مرتب میکنیم میدانیم که بهترین زمان پاسخ دهی و یا کمترین زمان انتظار را برای حالت صف داریم. که ترتیب P2,P4,P1,P3 میشود که در سطر اول نشان داده شده است
- بدترین حالت صف هم ترتیب عکس ترتیب فوق است که در سطر آخر جدول می‌بینیم.
- اجرای نوبتی به ترتیب اندیسهای فرآیندها انجام شده است یعنی P1,P2,P3,P4.
- برای این اجرا با صف مقادیر زیر بدست می‌آید:
- Avg wait time: 0 + 53 + (53 + 8) + (53+8+68) /4 = 60.75
- Avg. compl. time= 53 + (53+8) + (53+8+68) + (53+8+68+24)/4 = 99
- با توجه به جدول مشماهد می‌شود که اجرای نوبتی یک حالت میانی است و هیچوقت به خوبی و یا بدی صف نمی‌شود مگر در حالتی که مدت اجرای همه فرآیندها با هم یکسان باشد.
- در جایی که یک برنامه تعاملی داریم صف خوب نیست ولی برای کارهای محاسباتی خوب است.
- برای برنامه‌های تعاملی اجرای نوبتی به اندازه کافی خوب است.

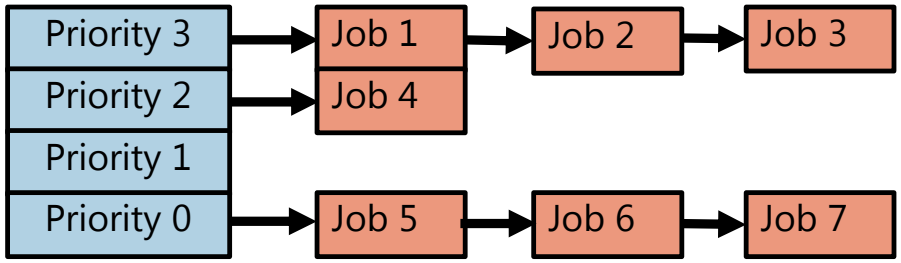
الگوریتم نوبتی وزن دار (weighted round robin)

- به هر فرآیند یک ضریبی از کوانتوم اختصاص داده می‌شود.
- $Q_{P1} = 1 * q$
- $Q_{P2} = 2 * q$
- $Q_{P3} = 3 * q$

- بدیهی است که این روش برای کارهایی که طولانی هستند مفید است زیرا کارهای کوتاه از افزایش زمان کوانتوم نفعی نمی‌برند.
- در این روش توان عملیاتی بهتر ولی زمان پاسخ بدتر می‌شود و عملا کارکرد روش به نوبت را به سمت صف پیش می‌برد.

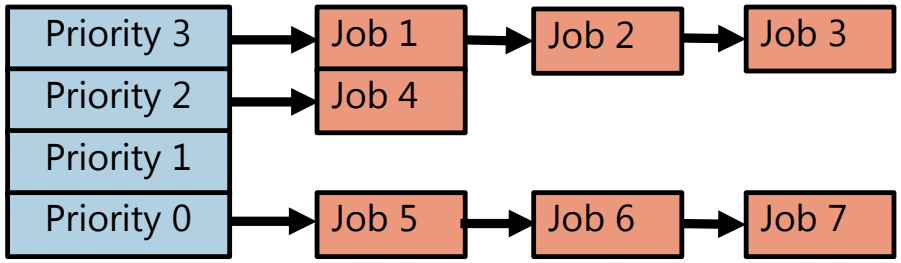
الگوریتم تقدم اکید (strict priority)

- در این الگوریتم ریسمانها همواره بر اساس یک تقدم مشخص شده ثابت اجرا می‌شوند.
- هر ریسمان دارای یک عدد تقدم است که توسط سیستم عامل به آنها نسبت داده می‌شود
- ریسمانها آماده اجرا بر اساس تقدم در یک صف قرار می‌گیرند
  - صف ریسمانهای آماده اجرا خود از چندین صف تشکیل میشود
- هر گاه در صف با تقدم بالاتر کاری برای انجام باشد تا کامل شدن تمامی آن کارها فقط از آن صف با تقدم باید کار انجام شود
- هر گاه صف با تقدم خالی شد حالا نوبت به تقدم پایین‌تر می‌رسد
- برای صف با کمترین تقدم باید تمامی صفهای بالاتر خالی باشند



الگوریتم تقدم اکید (strict priority)

- الگوریتم ساده‌ای است ولی
- مشکل اصلی این الگوریتم قحطی (starvation) است
  - هیچگاه نوبت به صف کم تقدم نرسد چرا که در صف با تقدم همیشه کاری برای انجام هست
- الگوریتم صف نوعی تقدم اکید است که در آن تقدم با فرآیندی است که زودتر آمده باشد.



الگوریتم‌های منصفانه

- اگر تقدم برای کارهای کوتاه باشد بدیهی است که زمان پاسخ‌دهی بهتری خواهیم داشت، اما
  - ممکن است برای کارهای طولانی هیچگاه پردازنده نرسد
  - برای جلوگیری از قحطی و رعایت انصاف باید به فرآیندهای طولانی نیز سهمی برای اجرا قائل شویم تا حداقل پیشرفتی داشته باشند.
- رعایت انصاف با حداقل زمان پاسخ دهی در سیستم در تعارض است

- چطور انصاف را پیاده‌سازی کنیم
  - به هر صف سهمی از زمان پردازنده قائل باشیم. مانند روش نوبتی
  - به شکل مقطعی برخی کارهای با تقدم پایین تقدم بالا بگیرند تا سرویس دهی شوند و بعد دوباره به همان سطح تقدم خودشان برمیگردند!
    - aging: هر چه زمان ماندن یک کار در سیستم افزایش می‌یابد تقدمش بالاتر می‌رود
  - استفاده از روش قرعه‌کشی

الگوریتم قرعه‌کشی (lottery scheduling)

- به هر فرآیند یک یا چند بلیط بخت‌آزمایی نسبت داده می‌شود.
  - بلیط می‌تواند یک عدد تصادفی در یک بازه مشخص باشد
- در هر بازه زمانی از بین بلیطهای اختصاص داده شده یکی از بلیطها به طور تصادفی انتخاب می‌شود. قرعه به نام هر فرآیند باشد آن فرآیند در پردازنده اجرا می‌شود.
- با این روش زمان پردازنده به نسبت بلیطها به فرآیندها اختصاص داده می‌شود.
- برای داشتن زمان پاسخ‌دهی معقول کارهای کوتاهتر تعداد بلیط بیشتری دریافت می‌کنند
- هر فرآیند حداقل یک بلیط دارد بنابراین سیستم دچار قحطی نمی‌شود
- با اضافه شدن و یا کم شدن تعداد فرآیندها، زمان انتظار بتدریج افزایش و یا کاهش می‌یابد و میزان تاثیر به تعداد کل بلیطها بستگی دارد نه به تعداد بلیطهای یک فرآیند.
  - در واقع همه فرآیندها با کم و زیاد شدن تعداد کل فرآیندها تاثیر میگیرند

الگوریتم قرعه‌کشی (lottery scheduling)

- فرض کنید در یک سیستم کارهای کوتاه ۱۰ بلیط و کارهای بلند ۱ بلیط دریافت کنند

# short jobs/ # long jobs	% of CPU each short jobs gets	% of CPU each long jobs gets
1/1	91%	9%
0/2	N/A	50%
2/0	50%	N/A
10/1	9.9%	0.99%
1/10	50%	5%

- اگر تعداد کارهای کوتاه زیاد شود طبیعی است که زمان پاسخ‌دهی خوب نخواهد بود

الگوریتم قرعه‌کشی (lottery scheduling)

- ویژگیها

- ساده
- پیاده‌سازی تقدم بدون قحطی
- عدم وجود تضمین در مورد نحوه اجرا
- عدم تضمین در سیستمهایی با تعداد فرآیند زیاد
- نیازمند تولید کننده عدد تصادفی خوب

- در مقایسه با الگوریتم نوبتی الگوریتم قرعه‌کشی میانگین زمان انتظار بهتری دارد
- در بدترین حالت می‌شود الگوریتم نوبتی و در بهترین حالت می‌شود مثل الگوریتم با تقدم قطعی

الگوریتم‌های با اطلاع از آینده

- تا اینجا متوجه شدیم که اگر کارها با ترتیب طول کوچکتر به بزرگتر به سیستم داده شوند عملکرد کلی سیستم خوب خواهد بود
  - زمان پاسخ‌دهی کم است
  - تعداد کارهای پایان یافته در واحد زمان نیز حداکثر است
- اگر بنوعی از آینده خبر داشتیم میتوانستیم کارها را به ترتیب طول اجرا به پردازنده بدهیم.
- فعلا این فرض را کرده و دو الگوریتم معرفی می‌کنیم

- الگوریتم کوتاهترین کار اول (Shortest Job First - SJF):
  - در واقع نوعی الگوریتم تقدم اکید است که تقدم با کوتاهی زمان اجرا مشخص می‌شود
  - اگر کارها همه یک اندازه باشند این الگوریتم همان روش صف خواهد بود

- الگوریتم کمترین کار باقیمانده اول (Shortest Remaining Time First - SRTF)
  - این الگوریتم نسخه قبضه‌ای الگوریتم بالایی است

الگوریتم‌های با اطلاع از آینده

- الگوریتمهای با تقدم دو نوع دارند:
  - قبضه‌ای (preemptive)
  - غیر قبضه‌ای (non-preemptive)

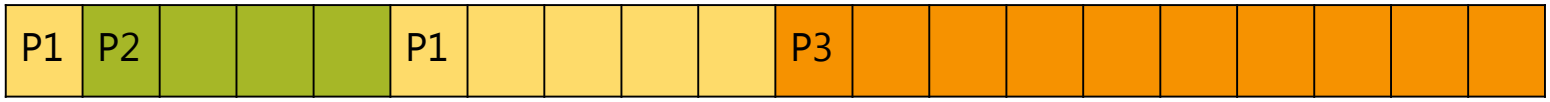
- در الگوریتمهای قبضه‌ای زمانی که کاری با تقدم بالا در حین اجرای یک کار با تقدم پایین می‌رسد اجرای کار فعلی متوقف و اجرای کار با تقدم بالا سریعا آغاز می‌شود.

الگوریتم‌های SJF, SRTF

Process	Arrival Time	Length (Burst)
P1	0	6
P2	1	4
P3	0	10

- SJF

- SRTF

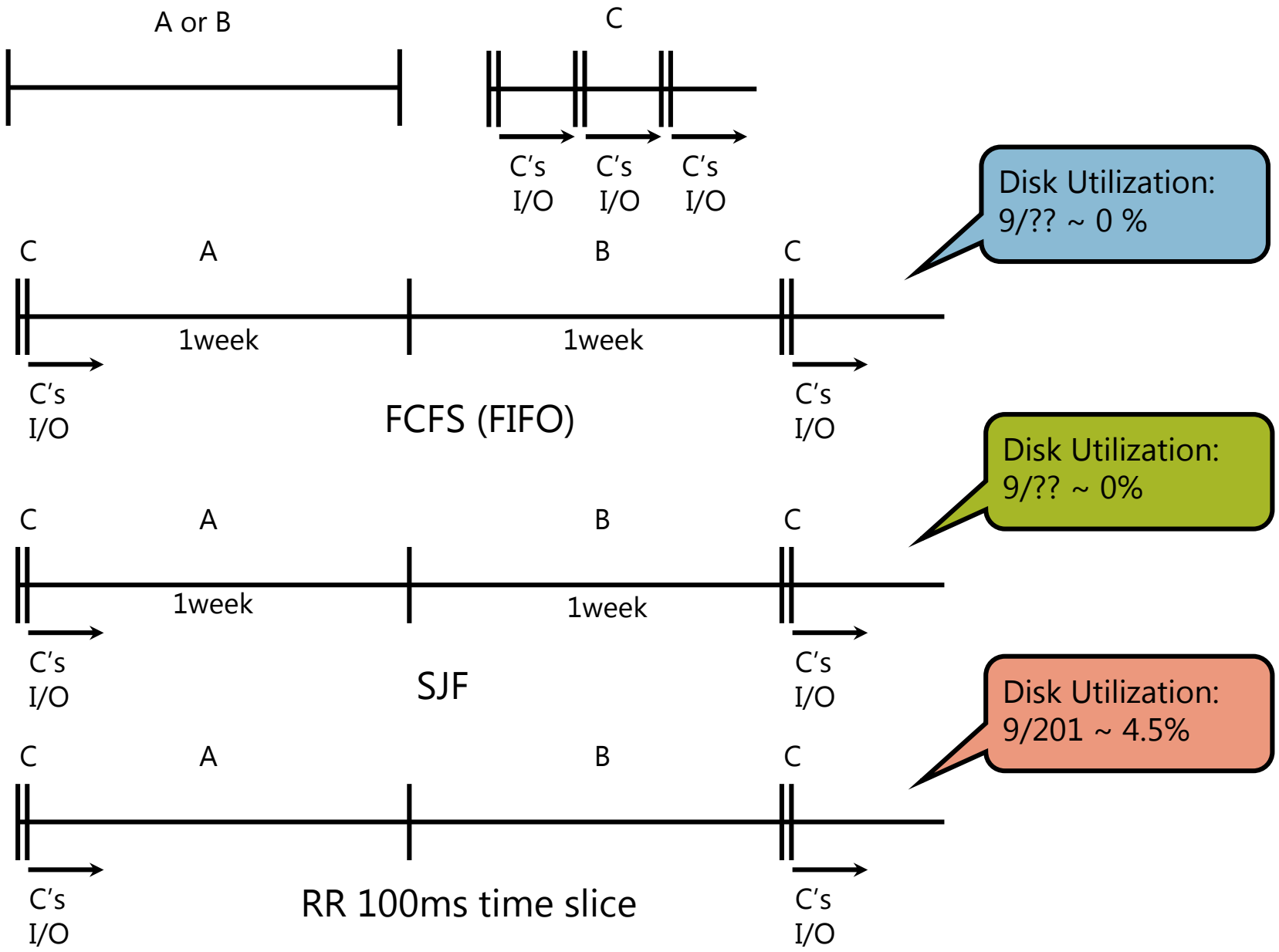


- $Waiting_{SJF}$ = [0 + (6-1) + (10-0) / 3] = 5
- $Waiting_{SRTF}$ = [(0 + (5-1)) + (1-1) + (10-0)] / 3 = 4.7

- این دو الگوریتم بهترین کاری است که می‌توان برای کاهش زمان پاسخ‌دهی داشت
- هر دو الگوریتم می‌توانند موجب قحطی شوند اگر تعداد کارهای کوتاه زیاد باشد

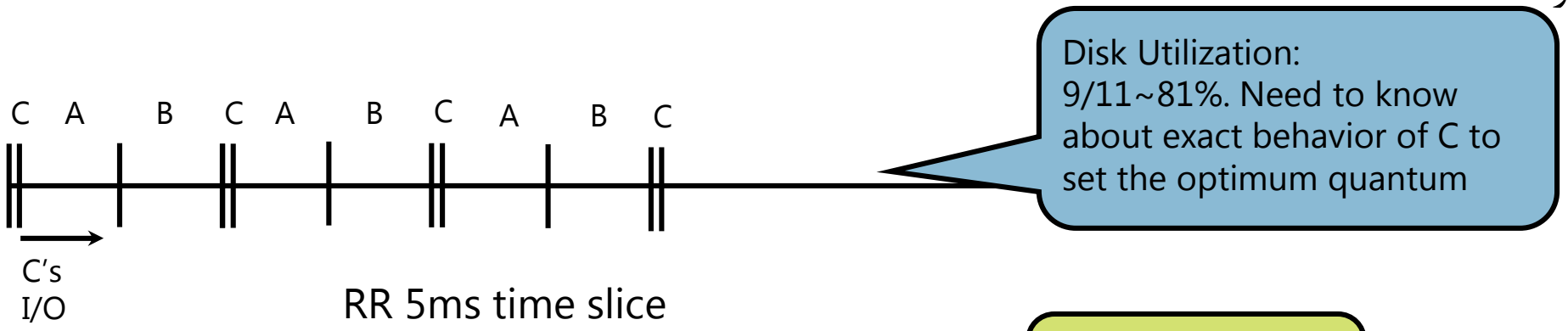
الگوریتم‌های SJF, SRTF

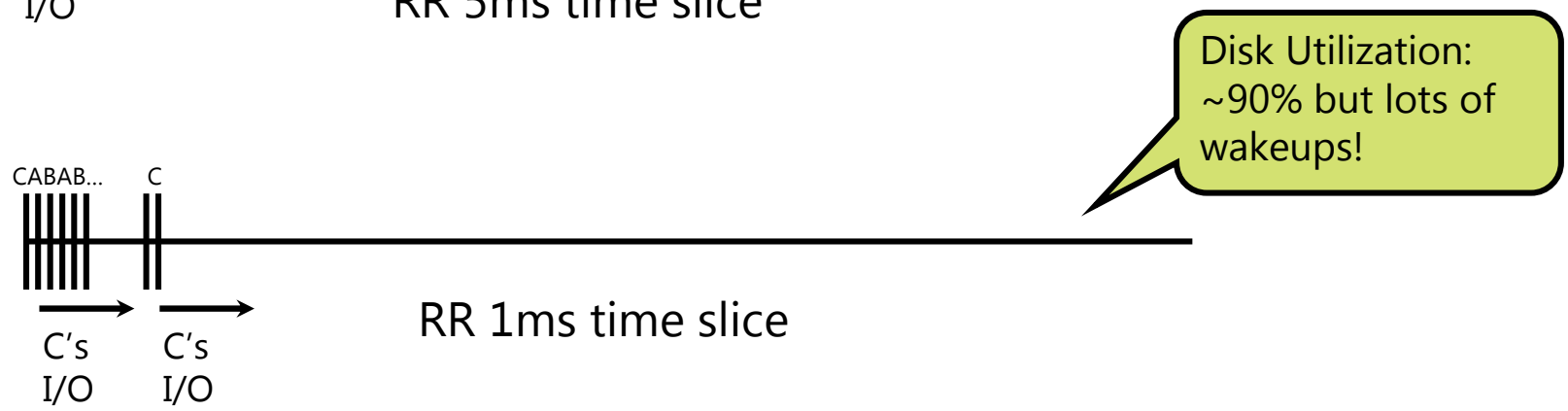
- فرض کنید که دو کار با طول اجرای یک هفته داریم و کار C که 1ms پردازش انجام میدهد و 9ms کار IO دارد

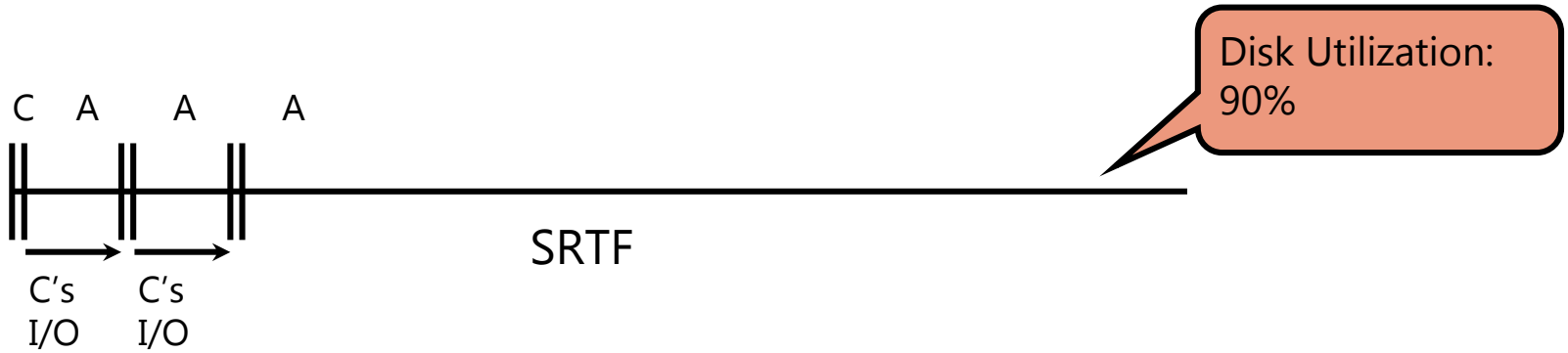


الگوریتم‌های SJF, SRTF

- فرض کنید که دو کار با طول اجرای یک هفته داریم و کار C که 1ms پردازش انجام میدهد و کار IO 9ms دارد







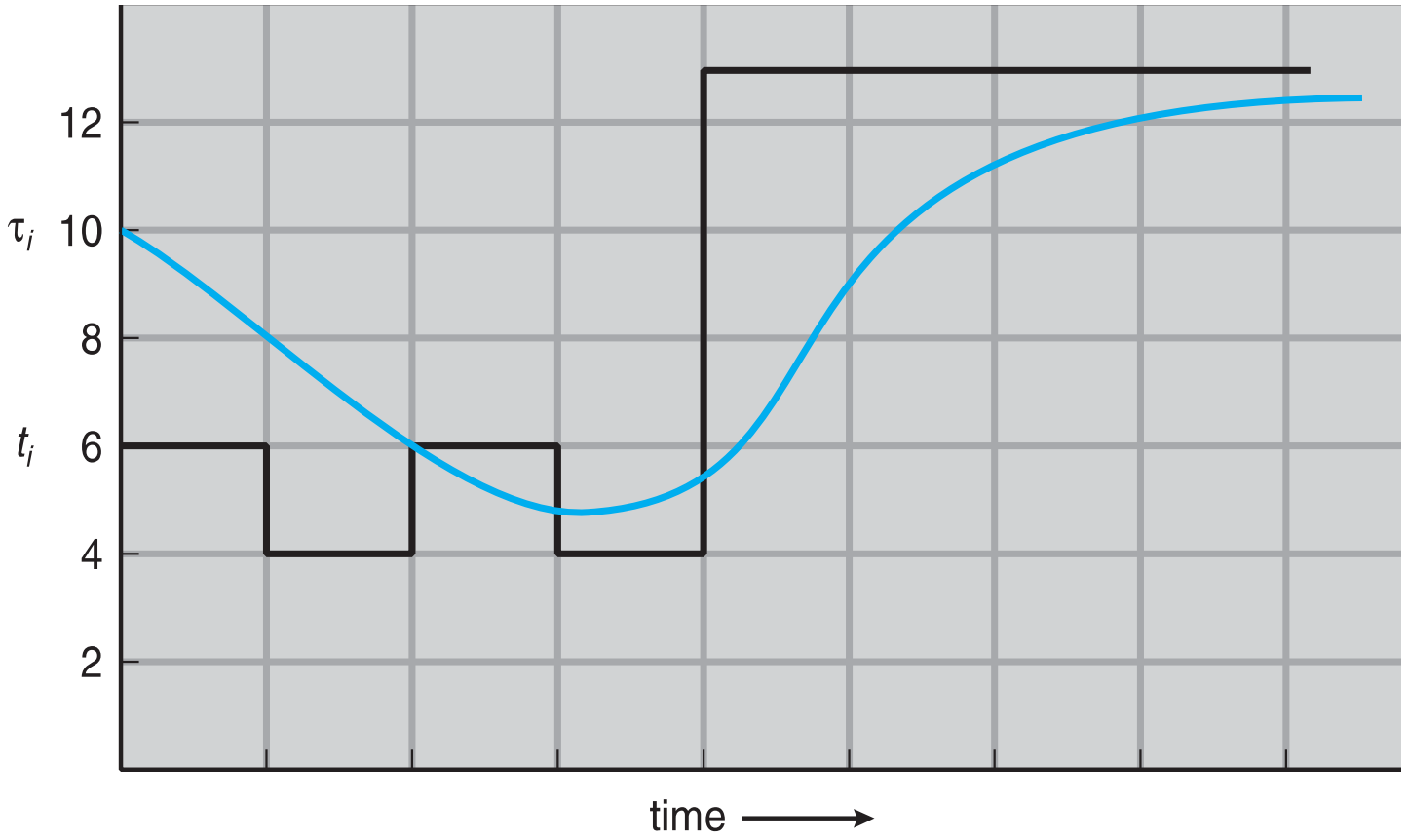
الگوریتم‌های با اطلاع از آینده

- چطور می‌توان از طول اجرای یک فرآیند در آینده اطلاع داشت؟
- برای پیش بینی آینده میتوانیم تاریخچه‌ای از رگبارهای یک فرآیند را نگهداری کرده و بر آن اساس رفتار آنرا در آینده حدس میزنیم.
  - فرض می‌شود اگر برنامه‌ای تابحال IO bound بوده پس از این به بعد هم خواهد بود.
  - اگر فرآیندی تصادفی رفتار می‌کند این کار کمکی به کار ما نخواهد کرد.
- برای پیش‌بینی می‌توانیم از هر تابعی استفاده کنیم مثلا از فیلتر Kalman که یک تابع صاف کننده (smooth) می‌توانیم استفاده کنیم.

- 1. t_n = actual length of n^{th} CPU burst
- 2. τ_{n+1} = predicted value for the next CPU burst
- 3. α, $0 \leq \alpha \leq 1$
- $\tau_{n+1} = \alpha\tau_n + (1-\alpha)t_n$

- معمولا $\alpha = \frac{1}{2}$ قرار داده میشود

الگوریتم‌های با اطلاع از آینده



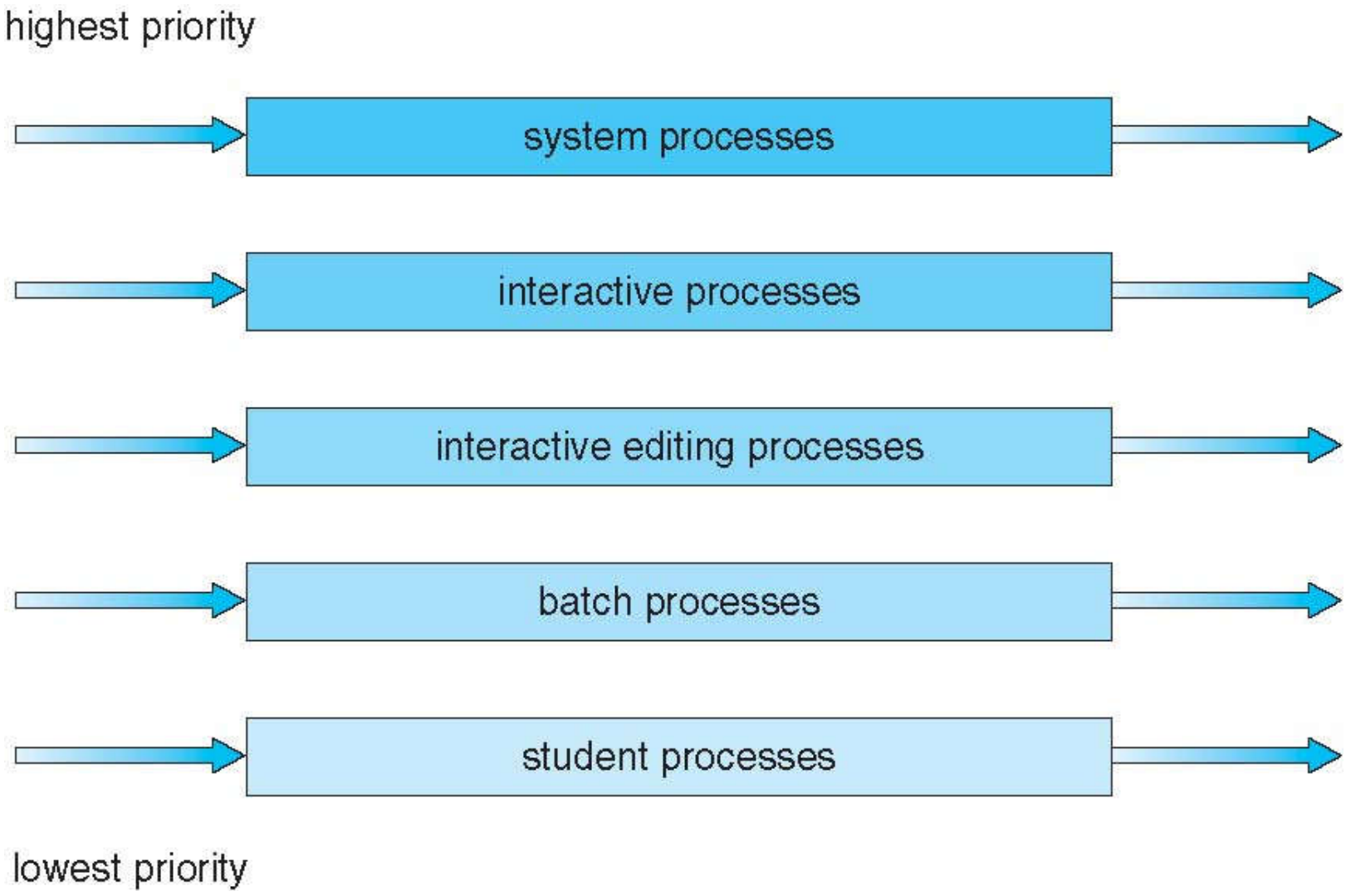
CPU burst (t_i)		6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

الگوریتم‌های زمان‌بندی چند سطحی

- در این روشها صف ریسمانهای آماده اجرا به چند صف تقسیم می‌شود
- ریسمانها بر اساس ویژگیهای مشخصی همواره در یکی از صفها گذاشته می‌شود
  - یک ریسمان هر بار که برای اجرا روی پردازنده در صف قرار داده می‌شود همیشه در یک صف مشخص قرار می‌گیرد
- هر صف الگوریتم زمان‌بندی خود را دارد
- بین صفها یک الگوریتم زمان‌بندی دیگری اجرا می‌شود

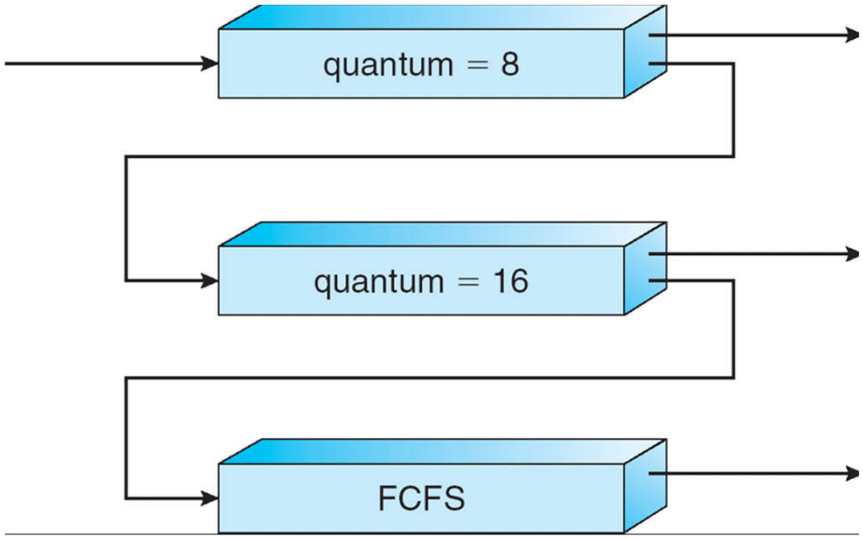
- مثال: سیستم عامل ریسمانهای پیش‌زمینه را در یک صف و ریسمانهای پس‌زمینه را در صف دیگر قرار می‌دهد
  - در صف اول بین ریسمانها الگوریتم نوبتی و در صف دوم بین ریسمانها الگوریتم صف اجرا می‌شود
  - بین دو صف می‌توان یکی از دو الگوریتم زیر را استفاده کرد
    - تقدم اکید
    - نوبتی وزن‌دار

الگوریتم‌های زمان‌بندی چند سطحی



الگوریتم‌های زمان‌بندی چند سطحی با بازخورد

- همانند روش‌های چند سطحی است منتها ریسمانهای بر اساس مدت زمان اجرا شدن بین صفها جابجا می‌شوند.
  - در واقع ریسمانها در این روشها تقدم متغیر دارند
- اجرای الگوریتم
  - تمامی کارها از اول صف اول و در بالاترین تقدم شروع به کار می‌کنند
  - اگر در مرحله اول تمام نشود تقدم یک مرتبه کاهش می‌یابد و به صف دوم می‌رود
  - اگر در مرحله بعدی نیز تمام نشد به مرتبه آخر سقوط می‌کند
  - هر بار که ریسمان کار خود را در زمان مقرر تمام می‌کند یک مرتبه تقدم ریسمان افزایش می‌یابد



- این مکانیزم بنوعی تقریب روش SRTF است
  - کار کوتاه زود تمام می‌شود ولی کاری که با دوبار شانس اجرای با تقدم تمام نشده احتمالا طولانی است و باید تقدم کمتری بگیرد

الگوریتمهای زمان‌بندی

- ویژگیهای الگوریتمهای زمان‌بندی

- منصفانه – غیر منصفانه
- بدون تقدم - با تقدم
- تقدم ثابت – تقدم متغیر
- یک سطحی – چند سطحی

الگوریتمهای زمان‌بندی بلادرنگ

Real-Time Scheduling Algorithms

سیستم‌های بلادرنگ

- در یک سیستم بلادرنگ باید پاسخ یک رویداد در زمان مناسب انجام شود
  - مثلا سیستم ترمز ABS زمانی که قفل شدن چرخها را تشخیص میدهد ۳-۵ میلی ثانیه وقت دارد تا سیستم ضد قفل را اجرا کند و گرنه وسیله از کنترل خارج خواهد شد.

- اجرای بلادرنگ در واقع ایجاد قابلیت پیش‌بینی رفتار و نحوه عملکرد سیستم است و لزوما ارتباطی با سرعت اجرا ندارد

سیستم‌های بلادرنگ

- سیستمهای بلادرنگ نرم (Soft Real-Time)
- در این سیستمها ضمانت اجرا در یک بازه زمانی مشخص داده نمی‌شود بلکه فقط ضمانت داده می‌شود که کار مشخص شده با تقدم بالاتری نسبت به سایر کارها انجام خواهد شد

- سیستمهای بلادرنگ سخت (Hard Real-Time)
- کار تعیین شده باید در مهلت مقرر اجرا شود و در صورت اجرا نشدن در مهلت تعیین شده مانند این است که آن کار اصلا انجام نشده است

- برای داشتن یک زمان‌بند بلادرنگ نرم الگوریتم باید قبضه‌ای باشد

- برای ضمانت سخت بودن باید الگوریتم مناسب طراحی شود

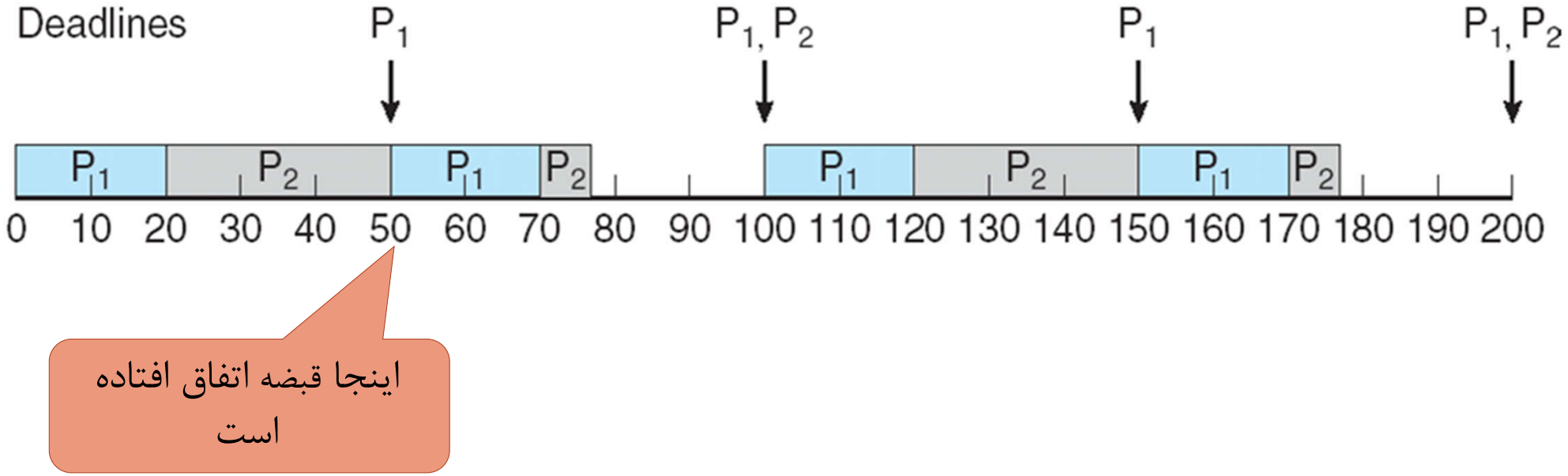
سیستم‌های بلادرنگ

- عموما بررسی زمان‌بندهای بلادرنگ با کارهای تناوبی انجام می‌گیرد
- هر کار
  - با زمان پردازش t
  - و ضرب‌الاجل (یا مهلت) d و
  - و تناوب p به سیستم ارجاع می‌شود
  - بدیهی است که $0 < t \leq d \leq p$
- بنابراین نرخ تناوب این کار $\frac{1}{p}$ است
- در کارهای تناوبی عموما مهلت انجام کار تا دور بعدی ورود همان کار در نظر گرفته می‌شود

الگوریتم نرخ یکنواخت (rate monotonic)

- در این الگوریتم تقدم کارها ثابت و نرخ تناوب ($1/p$) است
- هر چه تناوب بیشتر یا نرخ تکرار کار کمتر باشد در واقع تقدم کار هم کمتر است

	p	t
P1	50	20
P2	100	35



الگوریتم نرخ یکنواخت (rate monotonic)

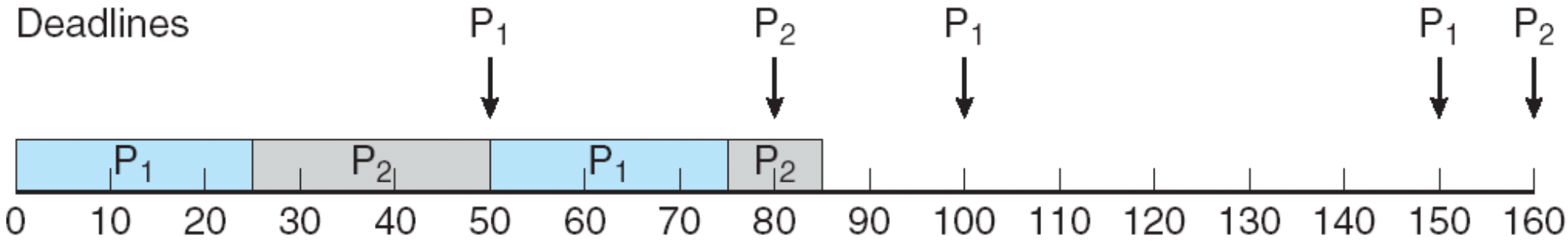
- یک مجموعه کار با این الگوریتم قابل زمان‌بندی است اگر
- $\sum \frac{t_i}{p_i} \leq 1$
- در مثال قبل
- $\frac{20}{50} + \frac{35}{100} = 0.4 + 0.35 = 0.75 < 1$

الگوریتم نرخ یکنواخت (rate monotonic)

- مثال

	p	t
P1	50	25
P2	80	35

- $\frac{25}{50} + \frac{35}{80} = 0.5 + 0.4375 \sim 0.94$



الگوریتم نرخ یکنواخت (rate monotonic)

- در واقع برای این الگوریتم
- CPU Utilization $\approx \sum \frac{t_i}{p_i} \leq N(2^{\frac{1}{N}} - 1)$
- برای یک کار 100%
- برای دو کار 83%
  - در مثال قبلی از این عدد بیشتر شده و برای همین هم قابل انجام نبود

- اگر در سیستم عامل یک کار جدید به سیستم بخواهیم اضافه کنیم سیستم عامل با این فرمول آنرا بررسی کرده و مشخص می:ند آیا کارهای قبلی با این کار جدید قابل انجام هستند یا خیر.
- این فرآیند تحت عنوان admission control شناخته می‌شود

الگوریتم نزدیکترین مهلت (Earliest Deadline First - EDF)

- این الگوریتم از نوع تقدم متغیر است و تقدم را به کاری می‌دهد که نزدیکترین ضرب‌الاجل را دارد
- در این الگوریتم نیازی نیست که کارها تناوبی باشند و یا همیشه مدت زمان اجرای ثابت داشته باشند
- مثال دو فرآیند قبلی که با الگوریتم نرخ ثابت قابل انجام نبود

	p	t
P1	50	25
P2	80	35