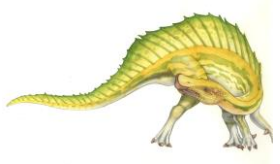
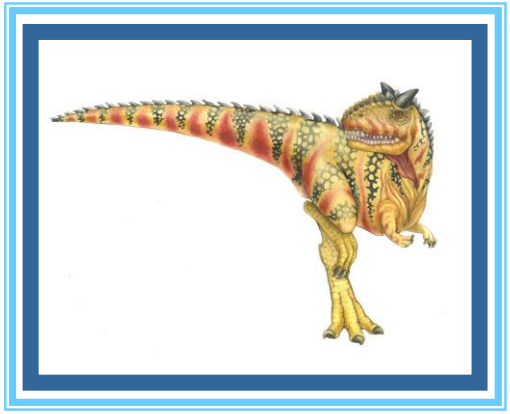
Operacinės sistemos

4. Procesų valdymas: CPU laiko skirstymas (planavimas), procesų sinchronizavimas.

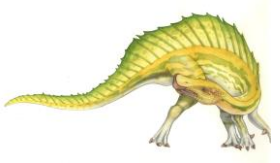


CPU tvarkaraščių sudarymas



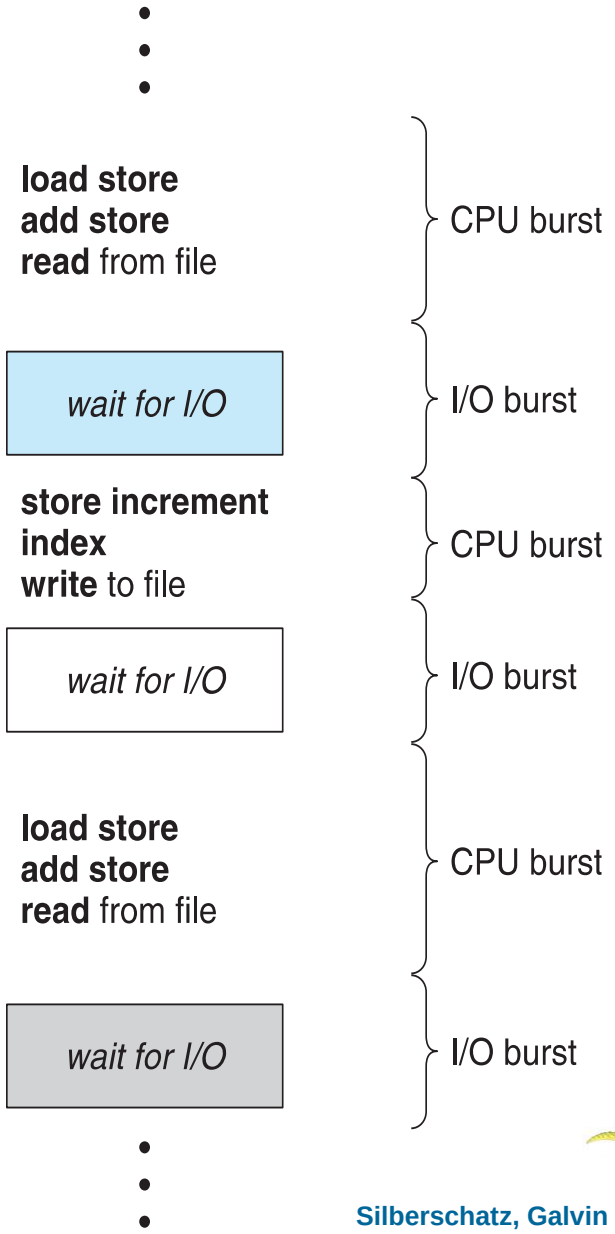
CPU laiko skirstymas potemės

- Pagrindiniai konceptai
- Tvarkaraščių sudarymo kriterijai
- Tvarkaraščių sudarymo algoritmai
- Gijų tvarkaraščiai
- Keleto procesorių tvarkaraščių sudarymas
- Real-Time CPU tvarkaraščiai
- Operacinių sistemų pavyzdžiai

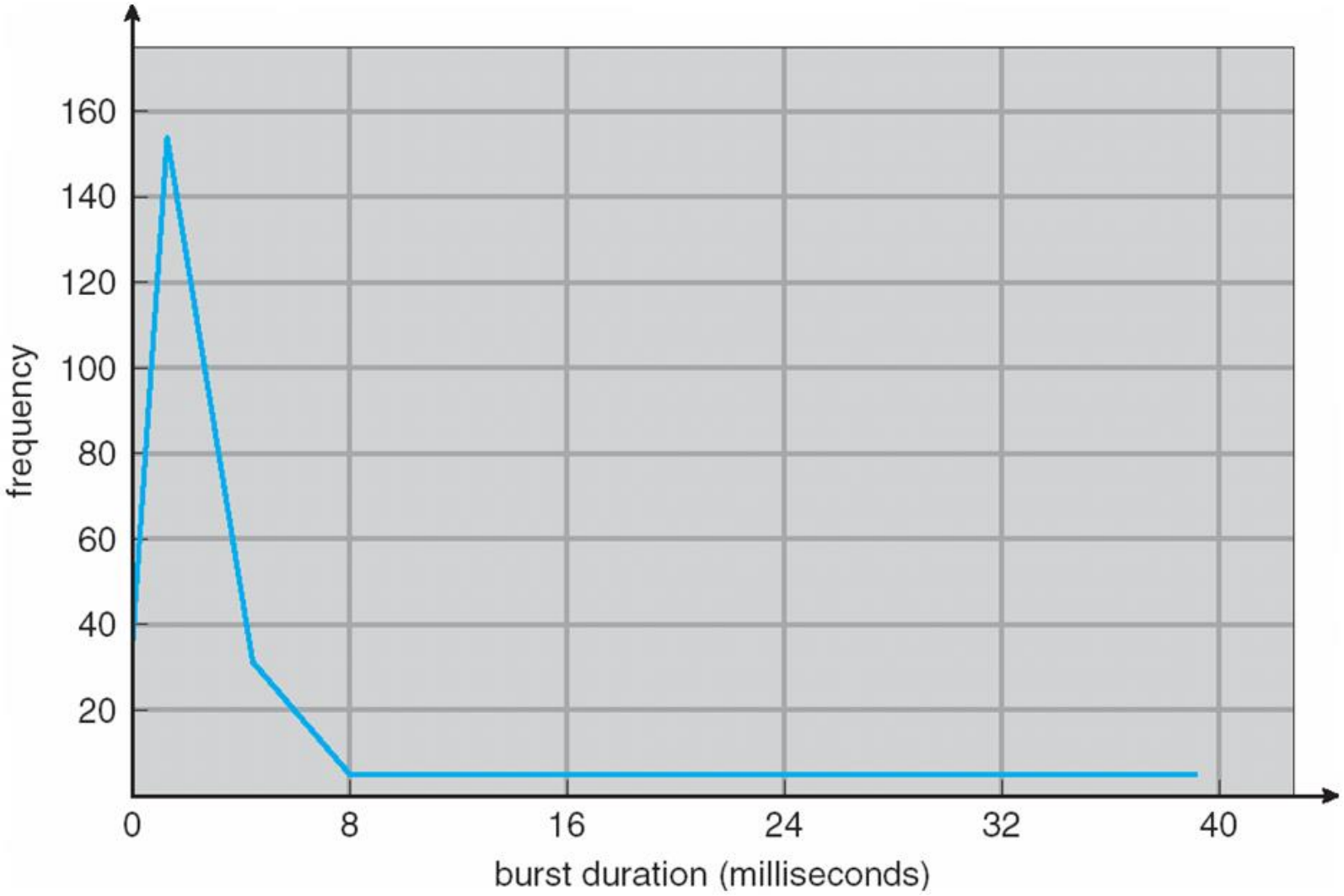


Pagrindiniai konceptai

- Maksimalus CPU išnaudojimas pasiekiamas per multiprogramavimą.
- CPU–I/O protrūkio ciklas (angl. *Burst Cycle*) – proceso vykdymas susideda iš CPU vykdymo ciklo (***cycle***) ir I/O laukimo
- ***CPU burst*** sekamas ***I/O burst***
- CPU burst pasiskirstymas yra viena iš pagrindinių problemų.



CPU-burst laiko histograma

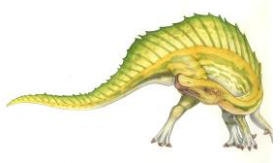


CPU tvarkaraščių sudarytojas

- **Trumpo laiko tvarkaraščių sudarytojas (angl. *Short-term scheduler*)** pasirenka iš procesų, esančių pasiruošusių eilėje ir priskiria vieną iš jų CPU.
  - Eilė gali būti išrikiuota įvairiais būdais.
- CPU tvarkaraščių sprendimai priimami, kai procesas:
  1. Persijungia iš vykdymo į laukimo būseną.
  2. Persijungia iš vykdymo į pasiruošusio būseną.
  3. Persijungia iš laukimo į pasiruošusio.
  4. Nutraukiamas
- Tvarkaraščių sudarymas pagal 1 ir 4 punktą yra ne prevencinis (angl. ***nonpreemptive***).
- Kiti būdai – prevenciniai (***preemptive***)

Dispečeris

- Dispečerio modulis suteikia CPU valdymą procesui, pasirinktam trumpo laiko tvarkaraščių sudarytojo, kas apima:
  - Konteksto perjungimą,
  - Perjungimą į vartotojo režimą,
  - Peršokimą į tinkamą vietą vartotojo programoje, kad perkrauti tą programą
  - **Dispečerio vėlinimas (angl. Dispatch latency)** – laikas, kurį dispečeris trunka sustabdyti vieną procesą ir paleisti kitą.



Tvarkaraščių sudarymo kriterijai

- **CPU išnaudojimas** – laikyti CPU kiek įmanoma apkrautą.
- **Pralaidumas (angl. Throughput)** – procesų, kurie užbaigia vykdymą per tam tikrą laiko tarpą skaičius.
- **Apsisukimo laikas (angl. Turnaround time)** – laiko tarpas, įvykdyti tam tikrą procesą.
- **Laukimo laikas** – laiko tarpas, kurį procesas turi laukti pasiruošusių eilėje.
- **Atsako laikas (angl. Response time)** – laiko tarpas, nuo užklausos pateikimo iki pirmo atsako sudarymo (ne rezultato) (bendro laiko aplinkoje).

Tvarkaraščių sudarymo algoritmų optimizavimo kriterijai

- Max CPU išnaudojimas
- Max pralaidumas
- Min apsisukimo laikas
- Min laukimo laikas
- Min atsako laikas

Pirmas atėjęs – primas aptarnaujamas (angl. First-Come, First-Served (FCFS)) tvarkaraščių sudarymas

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Pvz. procesai atkeliauja tokia tvarka: P_1 , P_2 , P_3
  Ganto diagrama:

- Laukimo laikas P_1 = 0; P_2 = 24; P_3 = 27
- Vidutinis laukimo laikas: (0 + 24 + 27)/3 = 17

FCFS tvarkaraščių sudarymas (tęs.)

Jei procesai atkeliauja tokia tvarka:

$$P_2 , P_3 , P_1$$

- Ganto diagrama:

- Laukimo laikas $P_1 = 6; P_2 = 0, P_3 = 3$
- Vidutinis laukimo laikas: (6 + 0 + 3)/3 = 3
- Daug geresnis, nei ankstesniu atveju.

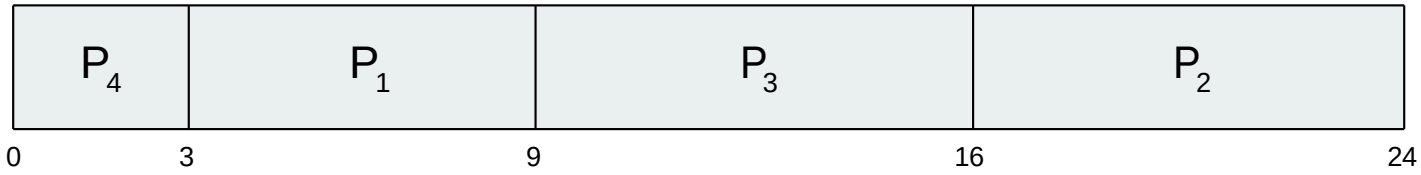
Trumpiausias darbas pirmas (Shortest-Job-First (SJF)) tvarkaraščiai

- Su kiekvienu procesu susietas sekančio CPU burst laikas.
  - Šis laikas naudojamas išdėlioti procesus su trumpiausiu laiku.
- SJF yra optimalus – duoda mažiausią vidutinį laukimo laiką duotam procesų rinkiniui.
  - Sunkumas yra žinoti sekančios CPU užklausos trukmę.

SJF pavyzdys

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF tvarkaraščio diagrama



- Vidutinis laukimo laikas = (3 + 16 + 9 + 0) / 4 = 7

Sekančios CPU Burst trukmės nustatymas

- Galima tik apytiksliai nustatyti trukmę – turėtų būti panaši į prieš tai buvusios.
  - Tuomet, parenkamas procesas su trumpiausia tikėtina trukme.

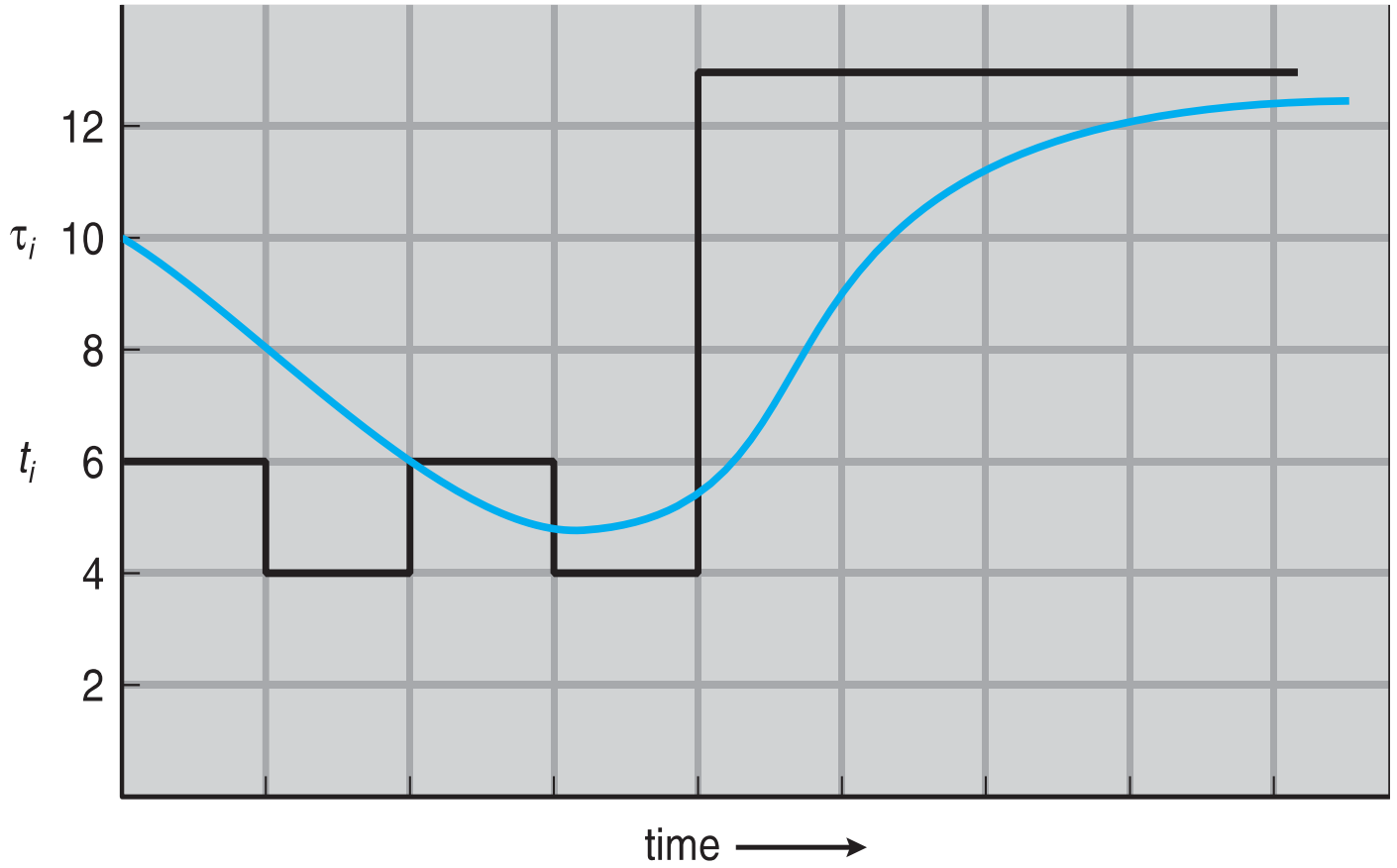
- Gali būti atliekama su ankstesnių CPU burst trukme, naudojant eksponentinį vidurkį.

1. $t_n = n^{th}$ CPU burst ilgis
2. $\tau_{n+1} =$ spėjama sekanti CPU burst reikšmė
3. $\alpha, 0 \le \alpha \le 1$
4. Aprašyti:

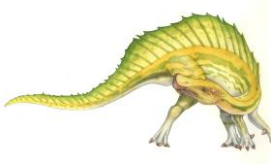
$$\tau_{n=1} = \alpha\, t_n + (1-\alpha)\tau_n .$$

- Paprastai nustatoma ½
- Prevencinė versija vadinama trumpiausias likęs laikas pirmas (angl. ***shortest-remaining-time-first***)

Sekančios CPU Burst trukmės nuspėjimas



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



Eksponentinio vidurkio pavyzdžiai

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Paskutinė istorija nesiskaičiuoja
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Tik paskutinis CPU burst skaičiuojasi
- Jei praplėsime formulę, gausime:

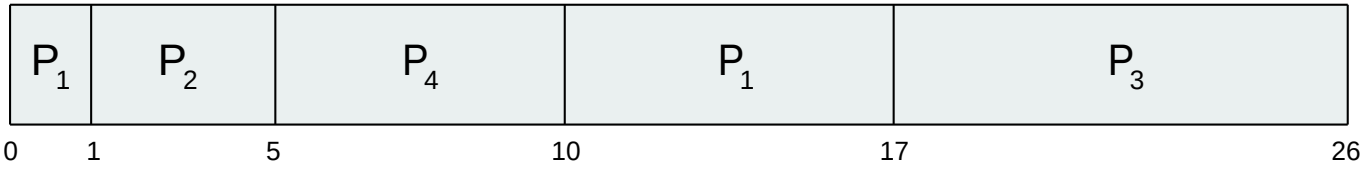
$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j \alpha\, t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

- Kadangi tiek α, tiek $(1 - \alpha)$ yra mažiau arba lygu 1, kiekvienas vėlesnis turi mažesnį svorį, nei prieš tai buvęs.

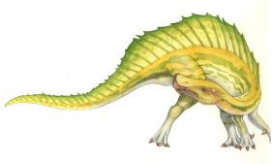
Shortest-remaining-time-first pavyzdys

Process	*Arrival* Time	Burst Time
P_1	0	8 7
P_2	1	4 4
P_3	2	9
P_4	3	5

- *Preemptive* SJF (Shortest Job First) Ganto diagrama



- Vidutinis laukimo laikas = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 ms



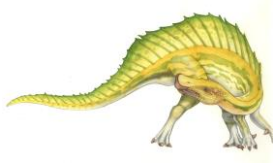
Shortest-remaining-time-first pavyzdys

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Sudarykite* SJF Ganto diagramą
- Apskaičiuokite vidutinį laukimo laiką.

Prioritetinis tvarkaraščių sudarymas

- Prioriteto numeris (sveikasis skaičius) yra susietas su kiekvienu iš procesų.
- CPU yra priskiriamas procesui su aukščiausiu prioritetu (mažiausias skaičius ≡ aukščiausias prioritetas)
  - Preemptive
  - Nonpreemptive
- SJF (Shortest Job First) yra prioritetinis tvarkaraščio sudarymas, kur prioritetas yra priešingas spėjamam sekančio CPU burst laikui.
- Problema ≡ **Starvation** – žemo prioriteto procesai gali būti visai nevykdomi.
- Sprendimas ≡ Senėjimas (**Aging**) – einant laikui, proceso prioritetas didėja.

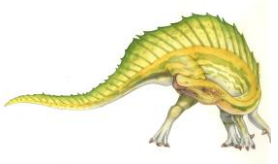


Prioritetinio tvarkaraščio pavyzdys

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart

- Average waiting time = 8.2 msec



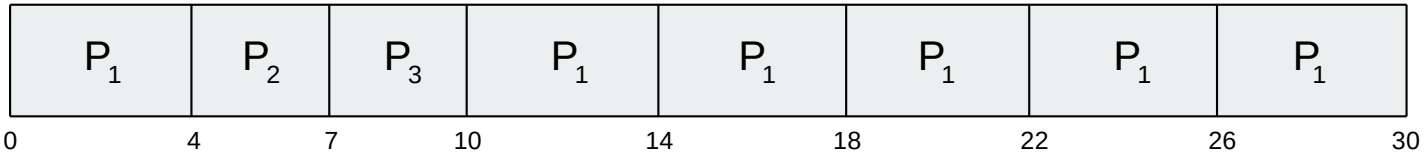
Round Robin (RR)

- Kiekvienas procesas gauna nedidelę CPU laiko dalį (**time quantum** *q*), paprastai 10-100 ms. Po šio laiko, procesas yra įdedamas į pasiruošusių eilės galą.
- Jei yra *n* procesų pasiruošusių eilėje ir laiko kvantas yra *q*, tuomet kiekvienas procesas gauna 1/*n* CPU laiko. Procesai nelaukia daugiau nei (*n*-1)*q* laiko vienetų.
- Taimeris pertraukia kiekvieną kvantą, kad numatyti sekantį procesą.
- Sparta:
  - *q* didelis $\Rightarrow$ FIFO
  - *q mažas* $\Rightarrow$ *q* turi būti didelis dėl konteksto perjungimo, kitaip pridėtinės išlaidos (angl. overhead) yra per didelės.

RR su laiko kvantu = 4 pavyzdys

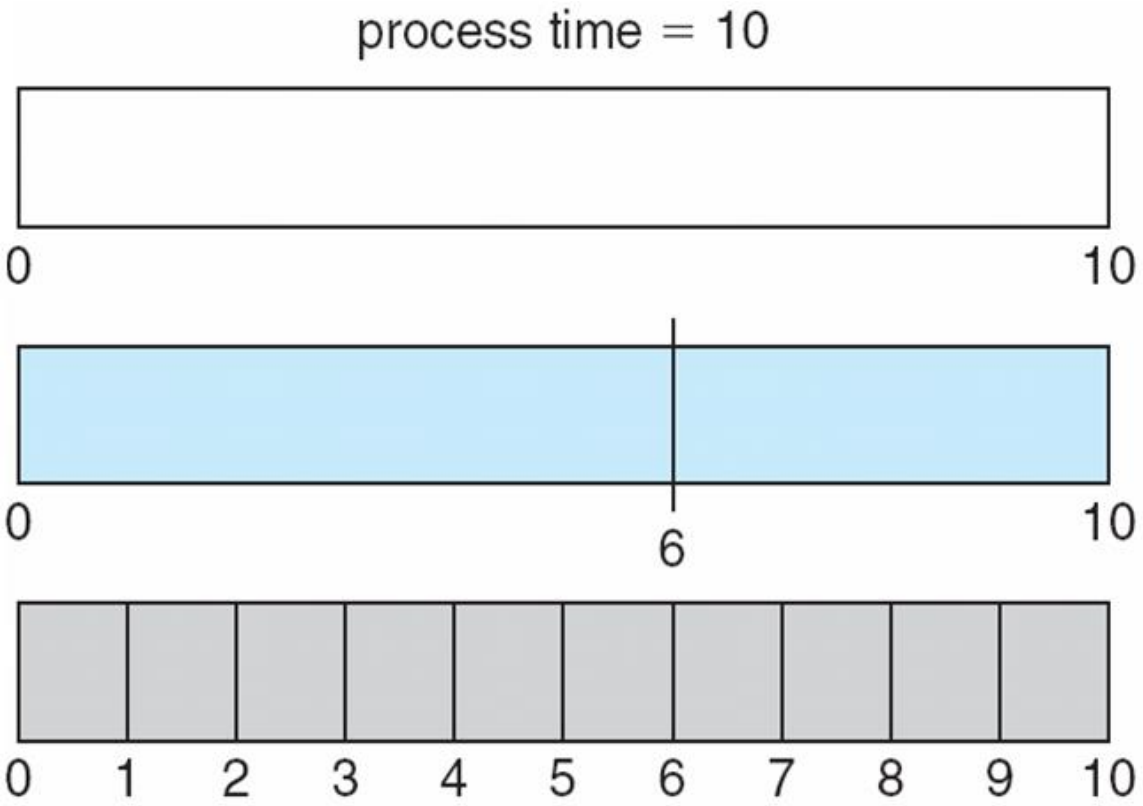
Process	Burst Time
P_1	24
P_2	3
P_3	3

- Ganto diagrama:



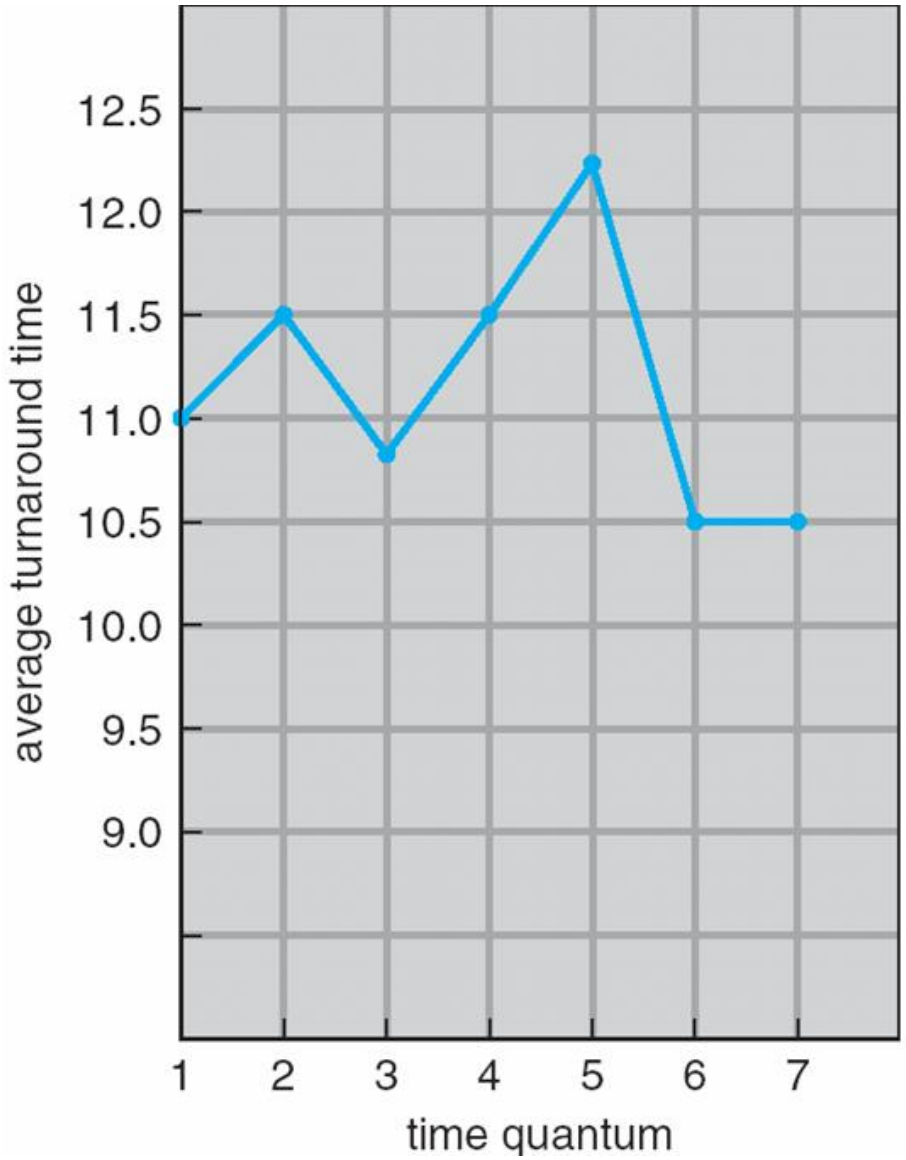
- Paprastai, didesnis vidutinis apsisukimo laikas, nei SJF, tačiau geresnis atsakas.
- q turi būti didelis, palyginus su konteksto perjungimo laiku.
- q paprastai 10ms - 100ms, konteksto perjungimas < 10 us

Laiko kvantas ir konteksto perjungimo laikas



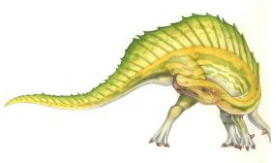
quantum	context switches
12	0
6	1
1	9

Apsisukimo laikas kinta su laiko kvantu



process	time
P_1	6
P_2	3
P_3	1
P_4	7

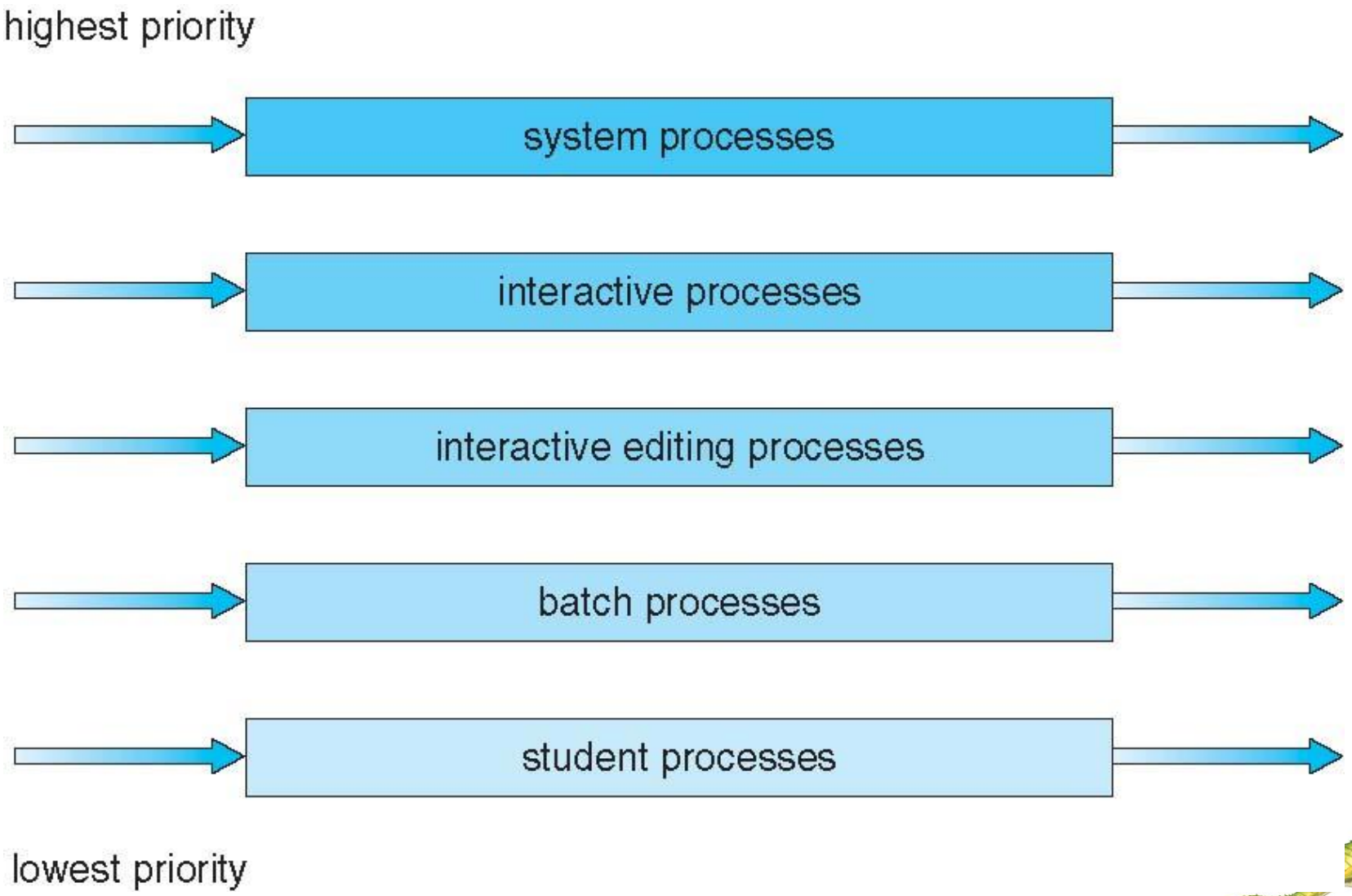
80% CPU bursts turi būti trumpesni nei q



Keleto lygių eilė

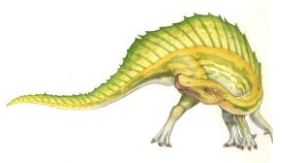
- Pasiruošusių eilė yra padalijama į keletą skirtingų eilių, pvz.:
  - **Priekinė (foreground)** (interactive)
  - **Foninė (background)** (batch)
- Kiekviena eilė turi savo tvarkaraščių algoritmą:
  - priekinė – RR
  - foninė – FCFS
- Tvarkaraščių sudarymas turi būti atliekamas tarp eilių:
  - Fiksuoto prioriteto tvarkaraščiai; (pvz., aptarnauti priekinę eilę prieš foninę). Galimas badavimas.
  - Laiko Time pjūvis – kiekviena eilė gauna tam tikrą CPU laiką kūrį gali padalinti tarp savo procesų; pvz., 80% priekinei RR
  - 20% foninei FCFS

Keleto lygių eilių tvarkaraščiai



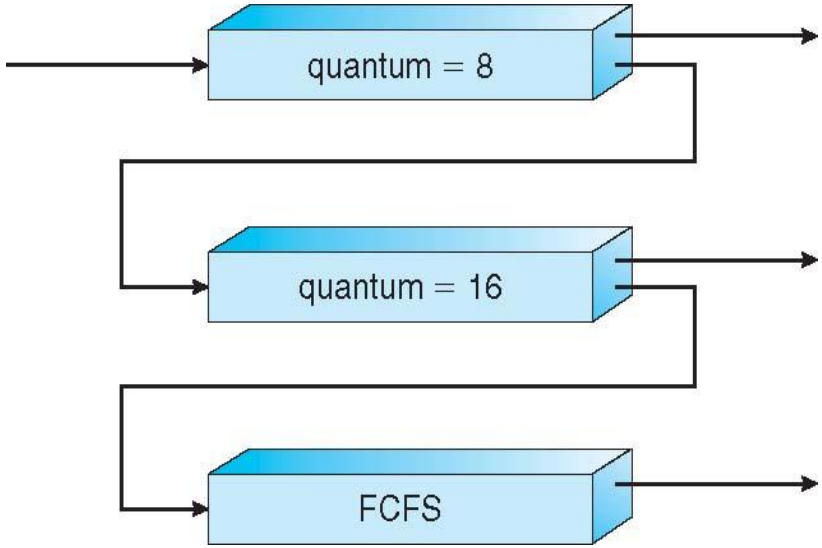
Keleto lygių atsako eilė

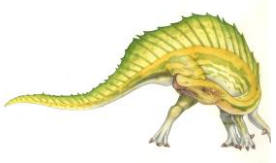
- Procesas gali judėti tarp eilių; tokiu būdu gali būti įgyvendintas senėjimas.
- Keleto lygių atsako eilės tvarkaraščių sudarytojas gali būti aprašomas šiais parametrais:
  - Eilių skaičius.
  - Tvarkaraščių algoritmas kiekvienai eilei.
  - Metodas, naudojamas nustatyti kada atnaujinti procesą.
  - Metodas, naudojamas nustatyti kada pažeminti procesą.
  - Metodas, naudojamas nustatyti į kurią eilę procesas pereis.



Keleto lygių atsako eilės (Multilevel Feedback Queue) pavyzdys

- Trys užklausos:
  - Q_0 – RR su laiko kvantu 8 ms
  - Q_1 – RR laiko kvantas 16 ms
  - Q_2 – FCFS
- Tvarkaraštis
  - Nauja užduotis patenka į eilę Q_0 kuri aptarnaujama FCFS
    - Kai ji pasiekia CPU, užduotis gauna 8 ms
    - Jei neužbaigiama per 8 ms, užduotis perkeliama į eilę Q_1
  - Q_1 užduotis aptarnaujama FCFS ir gauna papildomas 16 ms
    - Jei dar neužbaigiama ji perkeliama į Q_2





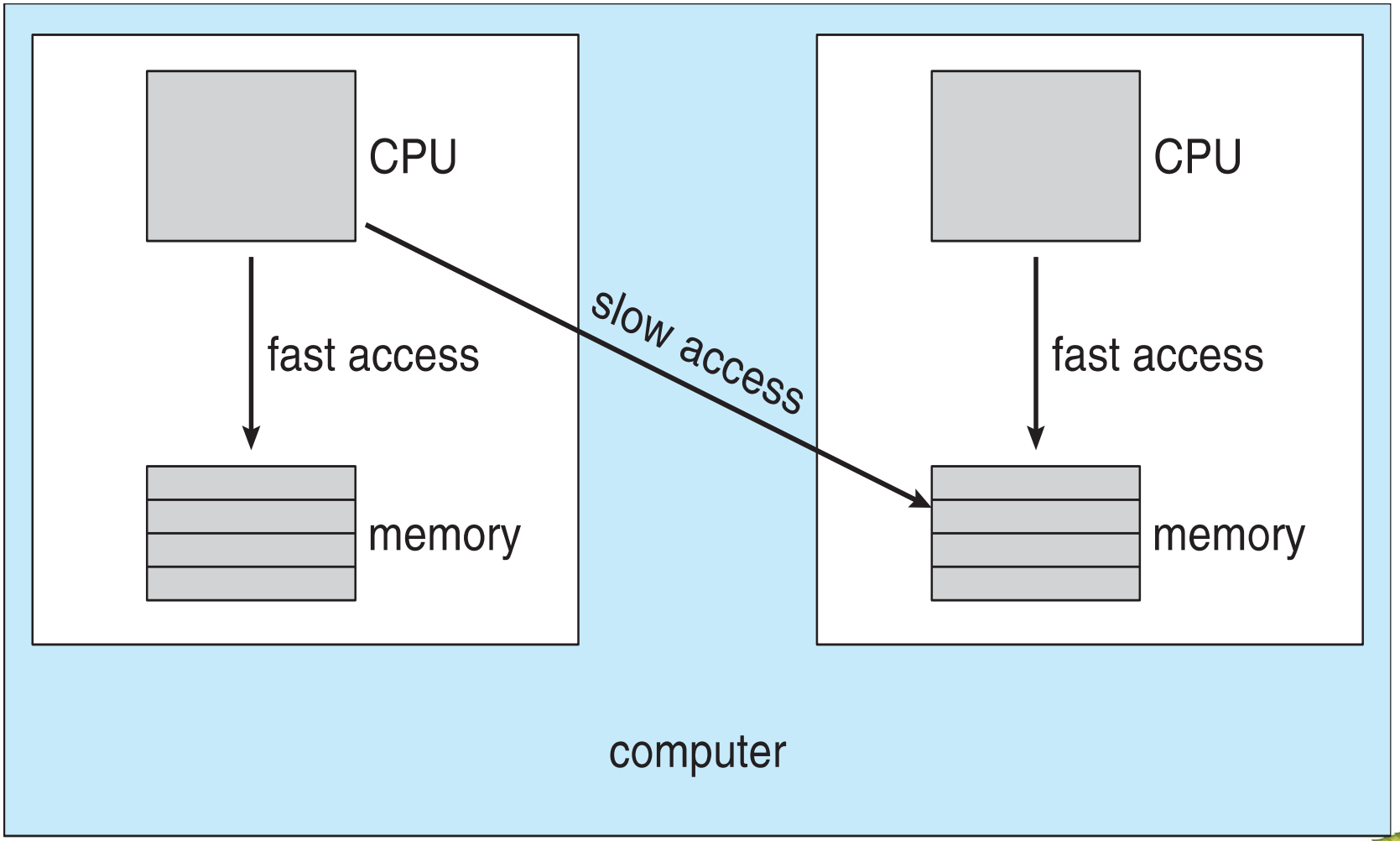
Gijų tvarkaraščių sudarymas

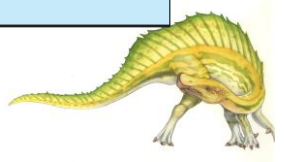
- Atskyrimas tarp vartotojo lygmens ir branduolio lygmens gijų.
- Jei gijos palaikomos, sudaromi gijų tvarkaraščiai, o ne procesų.
- Daug-su-vienu ir daug-su-daug modeliuose, gijų biblioteka sudaro tvarkaraščius vartotojo lygio gijoms vykdyti LWP.
  - Žinoma, kaip procesų konkurencijos rodiklis (angl. **process-contention scope (PCS)).**
  - Paprastai, atliekamas per prioritetų nustatymą programuotojo.
- Branduolio gijos priskiriamos laisvam CPU yra **system-contention scope (SCS)** – varžybos tarp visų gijų sistemoje.

Keleto procesorių tvarkaraščių sudarymas

- Tvarkaraščių sudarymas yra daug sudėtingesnis, kai prieinama keletas CPU.
- **Homogeniniai procesoriai (Homogeneous processors)** multiprocesoriuje.
- **Asimetrinis multiprocessing** – tik vienas procesorius prieina prie sistemos duomenų struktūrų, panaikindamas duomenų bendrinimo poreikį.
- **Simetrinis (Symmetric multiprocessing (SMP))** – kiekvienas procesorius yra sudarantis savo tvarkaraščius (self-scheduling), visi procesai bendroje pasiruošusių eilėje arba kiekvienas procesirius turi savo privačia eilę.
  - Šiuo metu dažniausiai naudojamas
- **Procesoriaus giminingumas (Processor affinity)** – procesas turi giminingumą procesoriui, kuriame veikia.
  - **soft affinity**
  - **hard affinity**
  - Variacijos, įskaitant **processor sets**

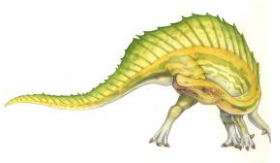
NUMA (Non-uniform memory access) ir CPU tvarkaraščiai





Multiple-Processor Scheduling – apkrovos balansavimas

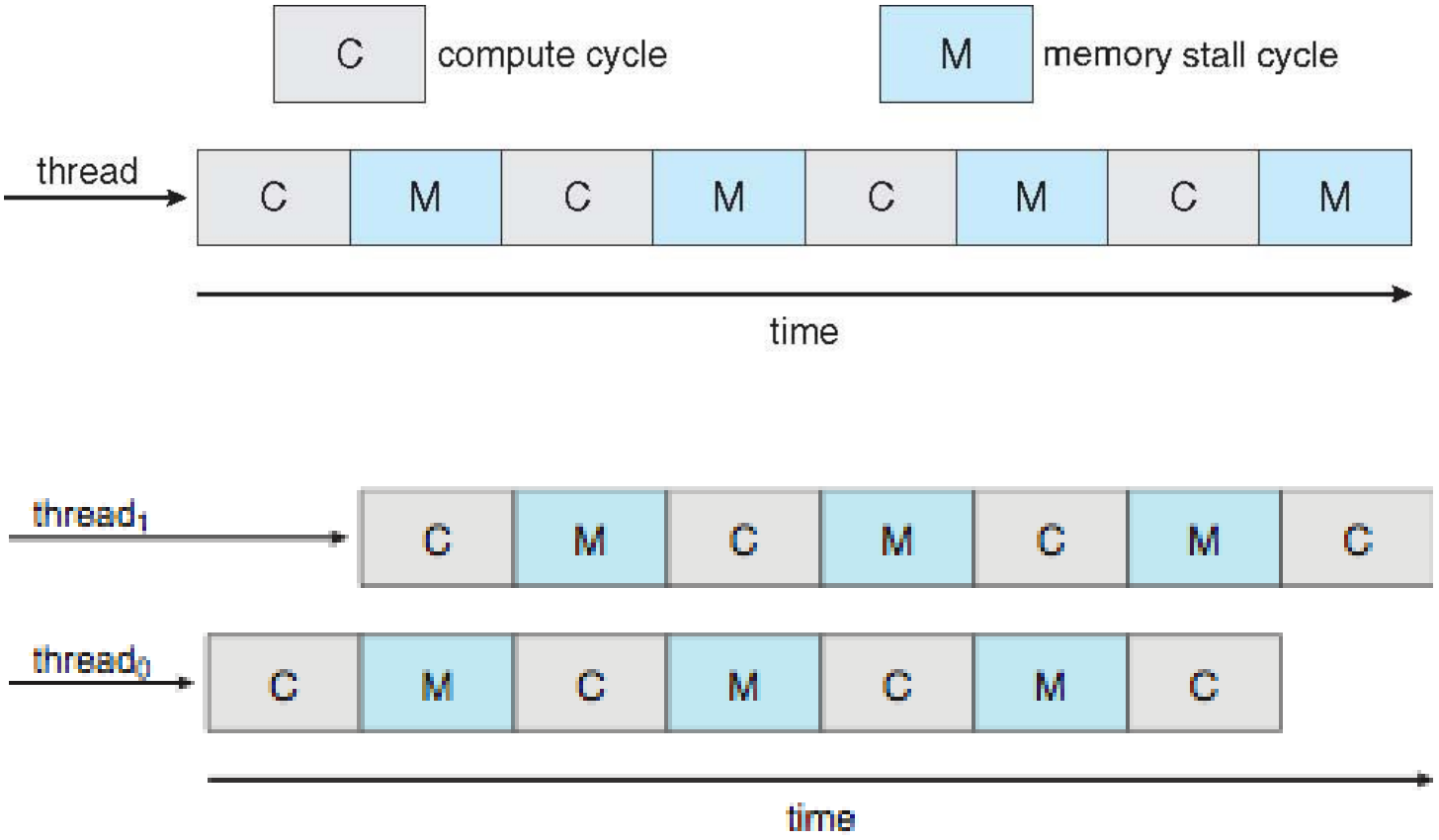
- Jei SMP (Symmetric multiprocessing), reikia išlaikyti visus CPU apkrautus dėl efektuvumo.
- **Apkrovos balansavimas (angl. Load balancing)** siekia tolygiai paskirstyti apkrovą.
- **Push migration** – periodinė užduotis tikrina kiekvieno procesoriaus apkrovą, jei randa perkelia užduotį iš perkrauto CPU į kitus.
- **Pull migration** – laisvi procesoriai paima laukiančias užduotis iš užimtų procesorių.



Keleto branduolių procesoriai

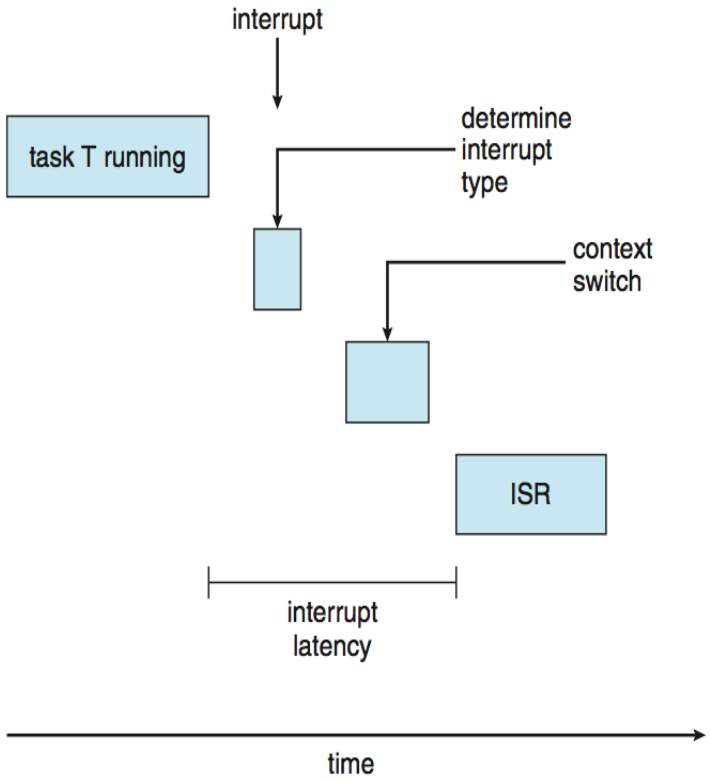
- Paskutiniu metu, siekiama sutalpinti keletą procesoriaus branduolių tame pačiame luste.
- Greitesni ir naudoja mažiau energijos.
- Keletas gijų branduoliui, taip pat, auga.

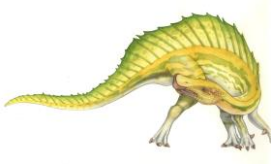
Keleto gijų keleto branduolių sistema



Realaus laiko CPU tvarkaraščio sudarymas

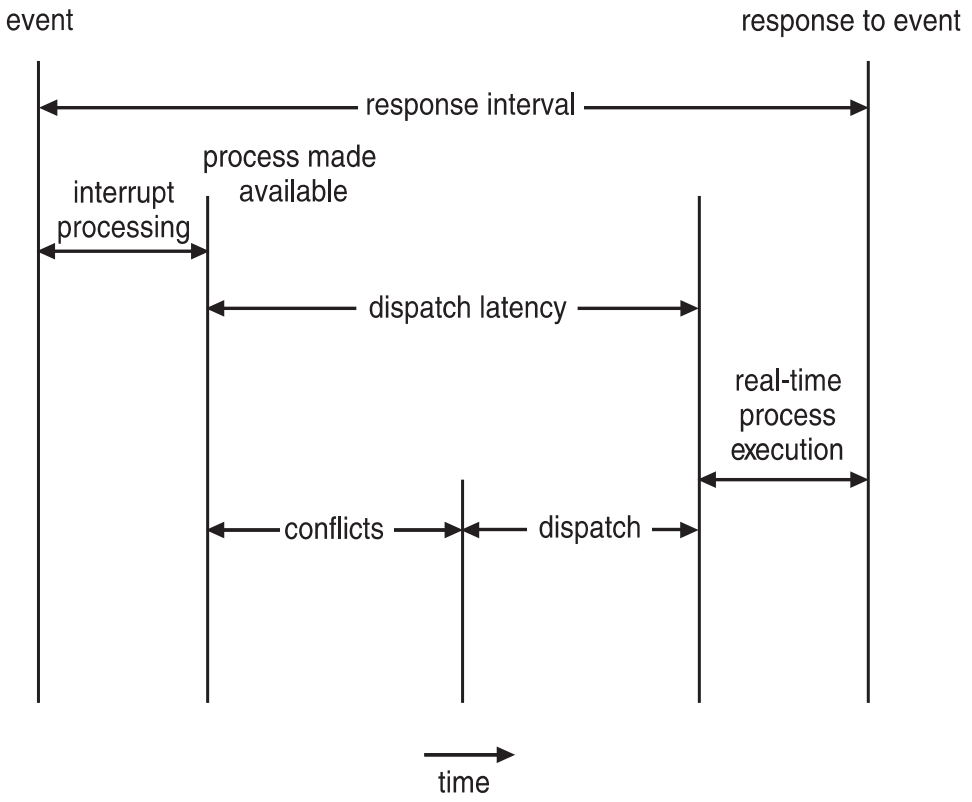
- Susiduriama su dideliais iššūkiais
- **Soft real-time sistemos** – nėra garantijos, kad realaus laiko procesas bus įdėtas į tvarkaraštį
- **Hard real-time systems** – užduotis turi būti įvykdyta iki numatyto laiko.
- Du vėlinimo tipai veikia spartą:
  1. Pertraukimų vėlinimas – laikas nuo pertraukimo atėjimo iki veiksmo kurį pertraukia servisas
  2. Dispečerio vėlinimas – laikas išimti dabartinį procesą iš CPU ir perjungti kitą.

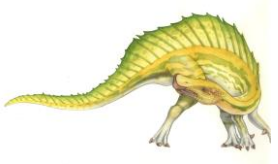




Realaus laiko CPU tvarkaraščio sudarymas (tęs.)

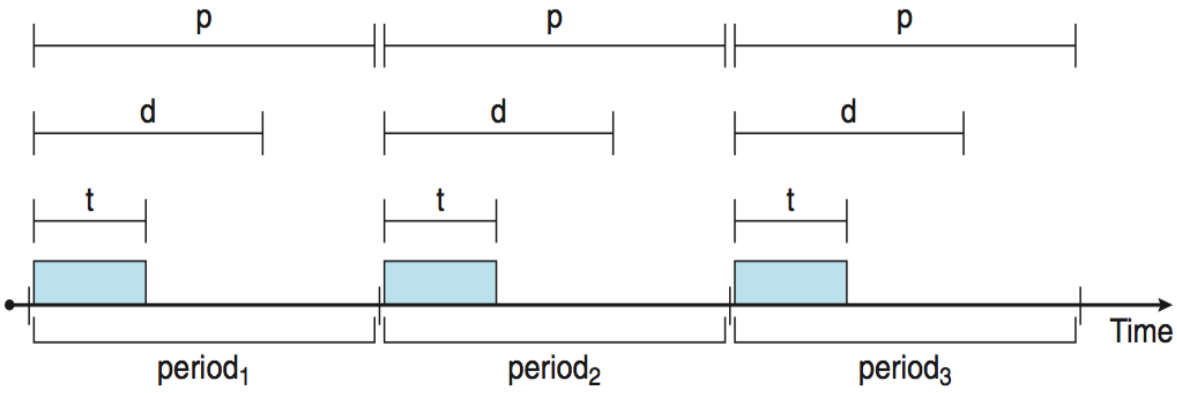
- Dispečerio vėlinimo konflikto fazė:
  1. Išėmimas bet kokių procesų, veikiančių branduolio režimu.
  2. Atlaisvinti žemo prioriteto procesų resursus, reikalingus aukštesnio prioriteto procesams.





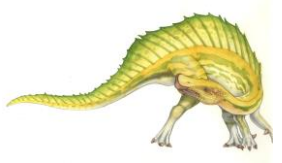
Prioritetais grindžiami tvarkaraščiai

- Realaus laiko tvarkaraščiams, jų sudarytojas turi palaikyti prevencinį, prioritetais grindžiamą tvarkaraščių sudarymą.
  - Tačiau garantuojamas tik soft real-time
- Hard real-time sistemoms turi būti suteikiama galimybė pasiekti deadlines.
- Procesai turi naujas charakteristikas: **periodiniams** reikia CPU nustatytais periodais
  - Turi apdorojimo laiką *t*, deadline *d,* periodą *p*
  - $0 \leq t \leq d \leq p$
  - **Rate** of periodic task is 1/*p*



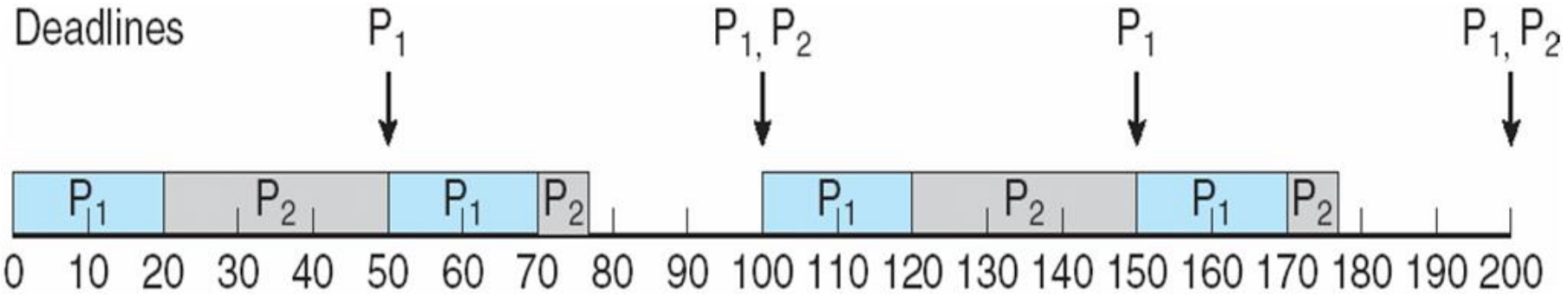
Virtualizavimas ir tvarkaraščių sudarymas

- Virtualizavimo programinė įranga sudaro CPU tvarkaraščius keletui OS.
- Kiekviena svečio OS turi savo tvarkaraštį
  - Nežino, kad jai nepriklauso CPU
  - Gali lemti prastą atsako laiką
- Gali pakenkti gero tvarkaraščių algoritmo naudai.

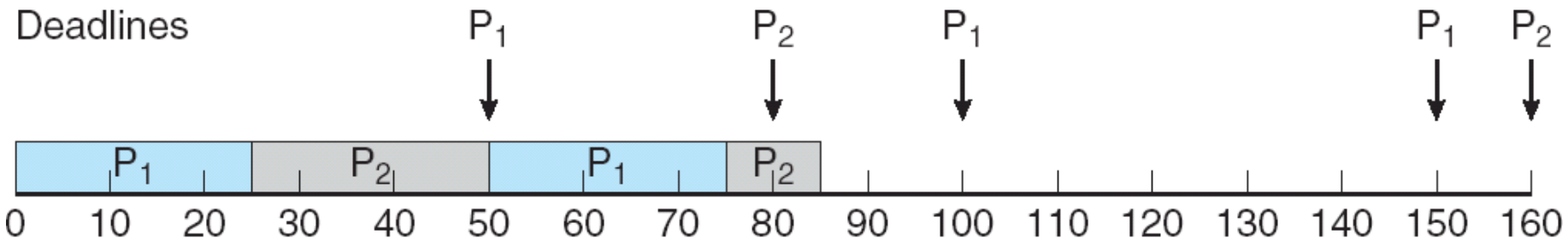


Įvertinamas monotoninis tvarkaraštis (Rate Monotonic Scheduling)

- Prioritetai priskiriami, atsižvelgiant į periodo inversiškumą.
- Trumpesni periodai = aukštesnis prioritetas.
- P_1 yra priskiriamas aukštesnis prioritetas nei P_2.



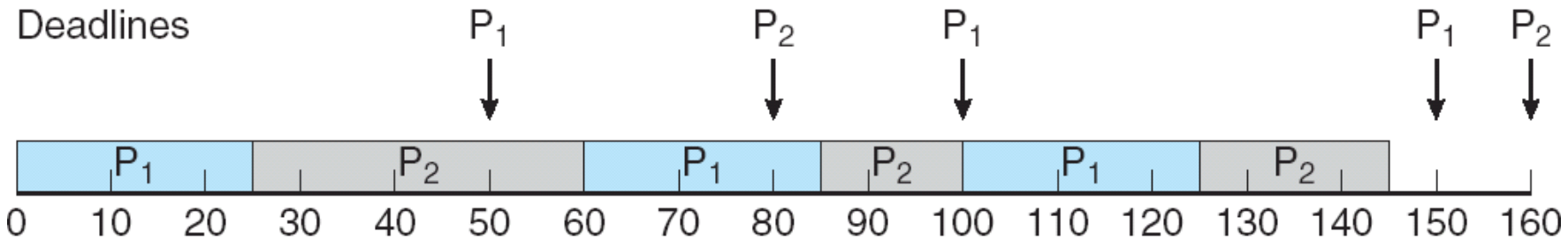
Praleistas terminas su Rate Monotonic Scheduling

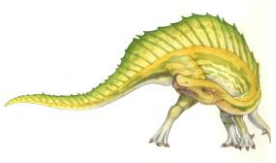


Anksčiausias terminas pirmas (Earliest Deadline First (EDF)) tvarkaraštis

- Prioritetai priskiriami pagal terminus:

  kuo ankstesnis terminas – tuo aukštesnis prioritetas;





OS pavyzdžiai

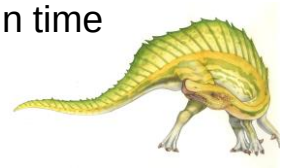
- Linux tvarkaraščiai
- Windows tvarkaraščiai

Linux tvarkaraščiai iki versijos 2.5

- Iki 2.5, naudojamo standartinio UNIX algoritmo variacija.
- Nuo 2.5 pereita prie constant order *O*(1) tvarkaraščių laiko
  - Prevencinis, prioritetais grindžiamas
  - Du prioritetų tipai: time-sharing ir real-time
  - **Real-time** varijuoja tarp 0 ir 99 ir tarp 100 to 140
  - Žemesnės vertės – aukštesnis prioritetas.
  - Aukštesnis prioritetas gauna didesnius q.
  - Užduotis vykdoma, kol liko laiko kvante (time slice) (**active**)
  - Jei laiko neliko (**expired**), nevykdoma, kol kitos užduotys išnaudos savo kvantus.
  - Visos vykdomos užduotys yra sekamos CPU **runqueue** duomenų struktūroje
    - Du prioritų masyvai (active, expired)
    - Užduotys indeksuojamos pagal prioritetą
  - Veikia gerai, tačiau prastas atsako laikas, dėl interaktyvių procesų.

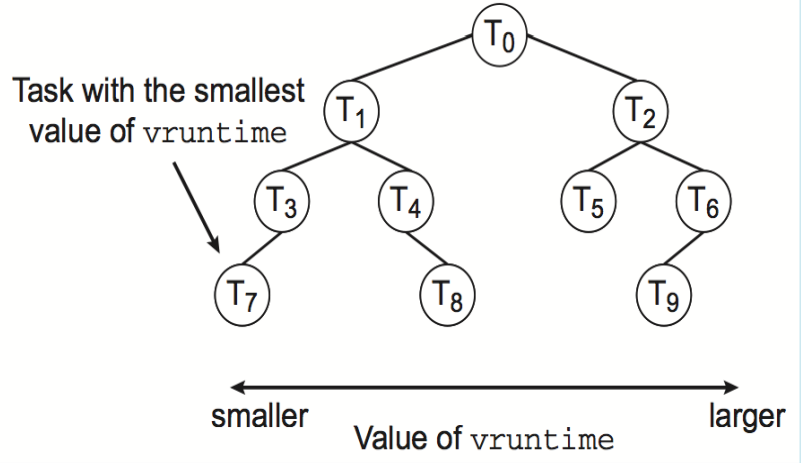
Linux tvarkaraščiai nuo 2.6.23 +

- ***Completely Fair Scheduler*** (CFS)
- **Tvarkaraščių klasės**
  - Kiekviena turi specifinį prioritetą.
  - Tvarkaraščio sudarytojas pasirenka aukščiausio prioriteto užduotį aukščiausioje tvarkaraščių klasėje.
  - Vietoje kvantais grindžiamos su fiksuotu laiku, remiamasi CPU laiko proporcija.
  - 2 tvarkaraščių klasės įtrauktos, kitos gali būti pridėtos.
    1. default
    2. real-time
- Kvantų skaičiavimas remiasi **nice value** nuo -20 iki +19
  - Žemesnė reikšmė – aukštesnis prioritetas.
  - Skaičiuojamas tikslo vėlinimas (**target latency**) – laiko intervalas, per kurį užduotis turi būti vykdoma nors kartą.
  - Šis laikas gali padidėti, jei aktyvių užduočių skaičius padidėja.
- CFS sudarytojas kiekvienai užduočiai laiko **virtual run time** in variable **`vruntime`**
  - Susietas su slopimo faktoriumi, remiantis užduoties prioritetu – žemesnis prioritetas – aukštesnis slopimo faktorius
  - Paprastai numatytas prioritetas užleidžia virtualiam vykdymo laikui = actual run time
- Kad pasirinktų sekančią užduotį, tvarkaraščio sudarytojas pasirenka su mažiausiu virtualiu vykdymo laiku.



CFS sparta

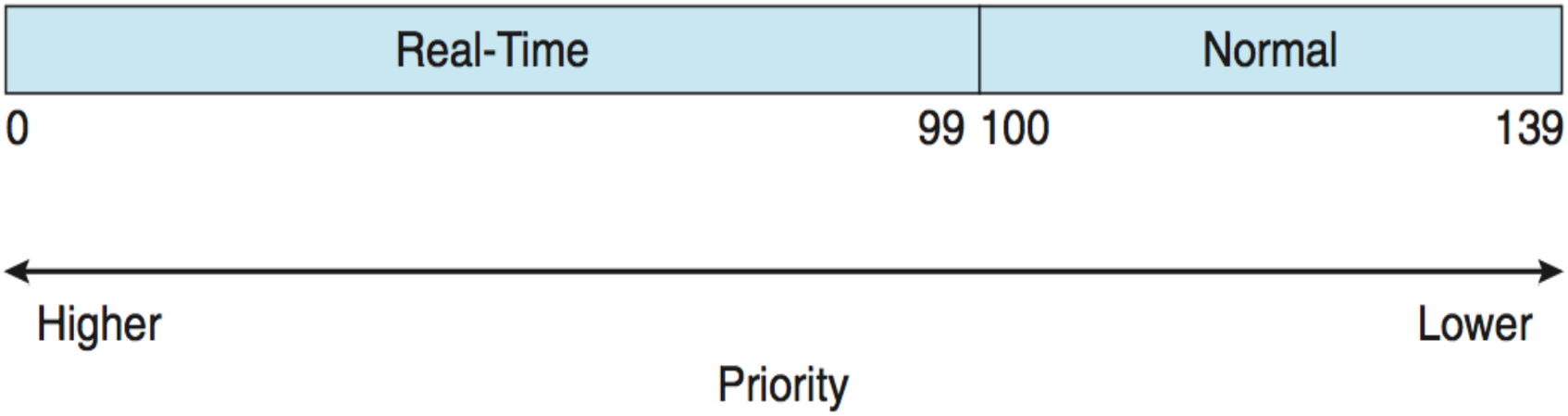
The Linux CFS scheduler provides an efficient algorithm for selecting which task to run next. Each runnable task is placed in a red-black tree—a balanced binary search tree whose key is based on the value of `vruntime`. This tree is shown below:



When a task becomes runnable, it is added to the tree. If a task on the tree is not runnable (for example, if it is blocked while waiting for I/O), it is removed. Generally speaking, tasks that have been given less processing time (smaller values of `vruntime`) are toward the left side of the tree, and tasks that have been given more processing time are on the right side. According to the properties of a binary search tree, the leftmost node has the smallest key value, which for the sake of the CFS scheduler means that it is the task with the highest priority. Because the red-black tree is balanced, navigating it to discover the leftmost node will require $O(\lg N)$ operations (where N is the number of nodes in the tree). However, for efficiency reasons, the Linux scheduler caches this value in the variable `rb_leftmost`, and thus determining which task to run next requires only retrieving the cached value.

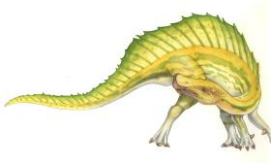
Linux tvarkaraščiai (tęs.)

- Real-time tvarkaraščiai remiasi POSIX.1b
  - Real-time užduotys turi statinius prioritetus
- Real-time plius normalus atvaizdavimas į globalią prioritetų schemą
- Nice value -20 atvaizduojama į globalų prioritetą 100
- Nice value +19 atvaizduojama į prioritetą 139



Windows tvarkaraščiai

- Naudojama prioritetais grindžiamas prevencinis tvarkaraščių sudarymas.
- Aukščiausio prioriteto gija vykdoma sekanti
- **Dispatcher** yra tvarkaraščių sudarytojas
- Gija vykdoma kol (1) blokuojama, (2) sunaudoja laiko kvantą, (3) išstumiama aukštesnio prioriteto gijos
- Real-time gijos gali išstumti ne-real-time
- 32-lygių prioritetų schema.
- **Variable class yra** 1-15, **real-time class** yra 16-31
- Prioritetas 0 yra atminties valdymo gija.
- Eilė kiekvienam prioritetui.
- Jei nėra vykdomos gijos, vykdo **idle thread.**



Windows prioritetų klasės

- Win32 API aprašo keletą prioritetų klasių, kurioms gali priklausyti procesas
  - REALTIME_PRIORITY_CLASS, HIGH_PRIORITY_CLASS, ABOVE_NORMAL_PRIORITY_CLASS,NORMAL_PRIORITY_CLASS, BELOW_NORMAL_PRIORITY_CLASS, IDLE_PRIORITY_CLASS
- Gija duotoje prioriteto klasėje gali turėti santykinį prioritetą
  - TIME_CRITICAL, HIGHEST, ABOVE_NORMAL, NORMAL, BELOW_NORMAL, LOWEST, IDLE
- Prioriteto klasė ir santykinis prioritetas apjungiami ir gaunamas prioriteto numeris.
- Bazinis prioritetas yra NORMAL toje klasėje
- Jei kvantas praeina, prioritetas sumažinamas, tačiau ne mažiau nei bazinis.

Windows prioritetų klasės (tęs.)

- Jei įvyksta laukimas, prioritetas padidinamas, remiantis tuo, kas laukia.
- Priekinis langas turi 3x prioriteto padidinimą.
- Windows 7 pridėjo **user-mode scheduling** (**UMS**)
  - Taikomosios programos kuria ir valdo gijas, nepriklausomas nuo branduolio.
  - Dideliam skaičiui gijų daug efektyvesnis.

Windows prioritetai

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Klausimai

1. Kodėl tvarkaraščių sudarytojui svarbu atskirti I/O apribotas programas nuo CPU apribotų programų?
2. Paaiškinkite, kaip sekančios tvarkaraščių sudarymo kriterijų poros konfliktuoja tam tikrais atvejais:
   - CPU išnaudojimas ir atsako laikas.
   - Vidutinis apsisukimo laikas ir maksimalus laukimo laikas.
   - I/O įrenginių išnaudojimas ir CPU išnaudojimas.
3. Procesai atkeliauja laiko momentu 0 tokia tvarka: P1, P2, P3, P4, P5.
   - Nubrėžkite 4 Ganto diagramas, iliustruojančias šių procesų vykdymą skirtingais algoritmais: FCFS, SJF, nonpreemtive priority, RR (q=1).
   - Koks kiekvieno proceso apsisukimo laikas (turnaround)?
   - Koks kiekvieno proceso laukimo laikas su kiekvienu algoritmu?
   - Kuris algoritmas turi mažiausią vidutinį laukimo laiką?

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

Klausimai

4. Kurie algoritmai gali kentėti nuo badavimo (starvation)?
   - First-come, first-served
   - Shortest job first
   - Round robin
   - Priority
5. Sistemoje naudojamas keleto lygių tvarkaraščių sudarymas. Kokios strategijos vartotojas turi laikytis, kad maksimizuoti CPU laiką, skirtą vartotojo procesams?
6. Didesnis proceso prioritetą nurodantis skaičius reiškia aukštesnį prioritetą?