4. Raspoređivanje poslova

Drugi dio – raspoređivanje na višeprocesorskim sustavima

4.6. Višeprocesorsko raspoređivanje

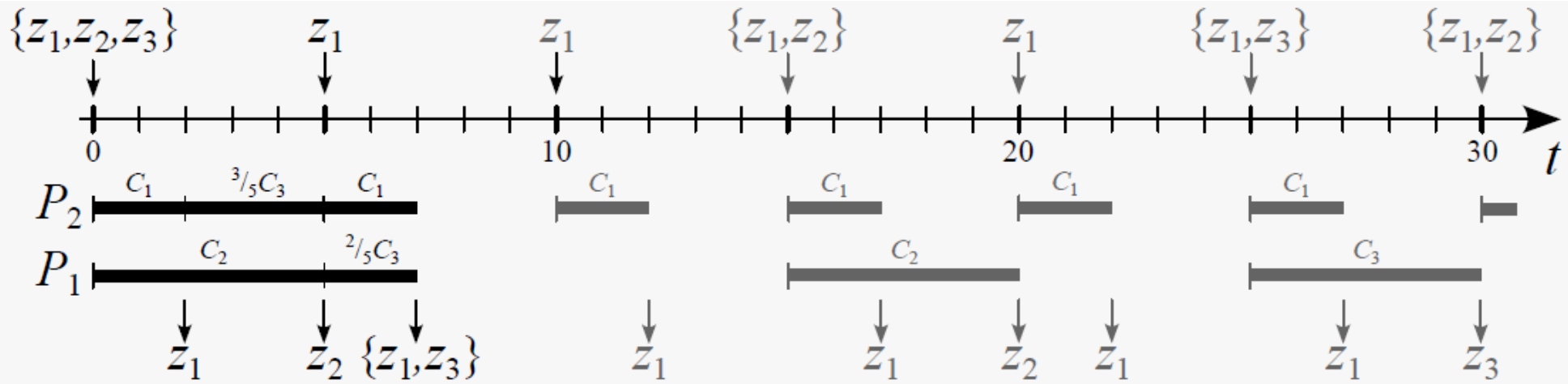
- prikazani algoritmi za jednoprocesorska rade i na višeprocesorskim
  - jedino što se odabire više zadataka za izvođenje, kad god se pojavi potreba za raspoređivanje
  - npr. sustav koji ima M procesora i treba obavljati N zadataka (N > M)
  - kod RMPA – M pripravnih zadataka najvećeg prioriteta se izvodi
  - kod EDF – M pripravnih zadataka s najbližim rokovima se izvodi

- očekivano, višeprocesorski sustav može više toga rasporediti
  - ali koliko više? NIJE M puta više
  - formule za rasporedivost RMPA/EDF na višeproc. sustavu su složenije
  - nisu navedene u ovim materijalima (mogu se pogledati u literaturi)

Primjer 4.10.

Prikažimo rad postupka **mjere ponavljanja** kada se primjenjuje na dvoprocesorskom računalu nad sustavom iz primjera 4.4.

$$
\begin{aligned}
Z_1: &\quad T_1 = 5 \text{ ms}, &\quad C_1 = 2 \text{ ms} \\
Z_2: &\quad T_2 = 15 \text{ ms}, &\quad C_2 = 5 \text{ ms} \\
Z_3: &\quad T_3 = 25 \text{ ms}, &\quad C_3 = 5 \text{ ms}
\end{aligned}
$$



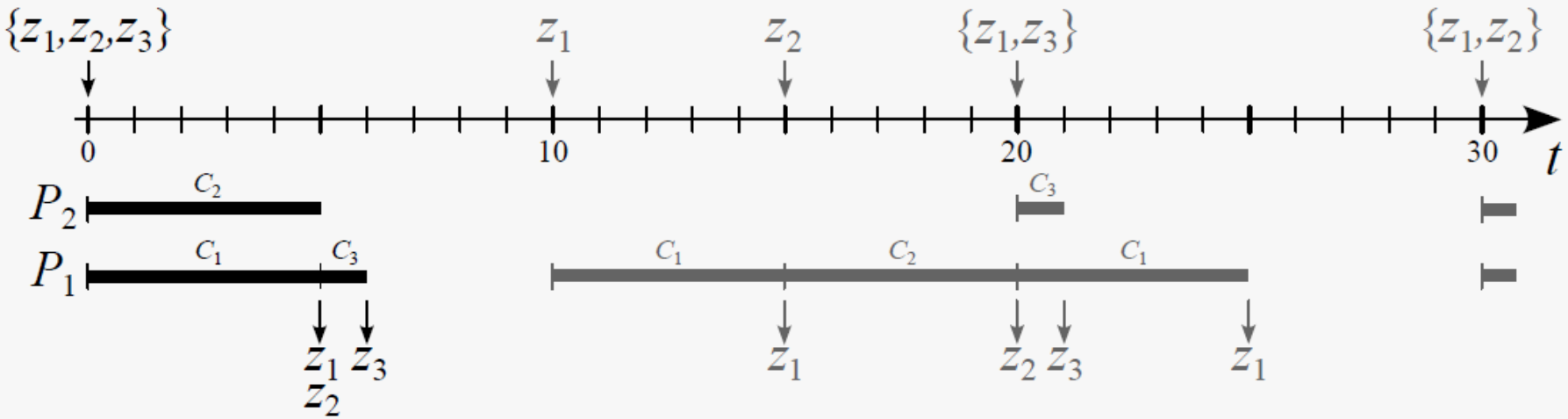
Slika 4.15.

Već se u 7. ms može zaključiti da je sustav raspoređiv na dvoprocesorskom računalu (i stati s daljnjom provjerom). Očito je problem olakšan, tj. takvi sustavi mogu zadovoljiti veći skup zadataka.

Primjer 4.11.

Prikažimo korištenje **raspoređivanja prema rokovima** za sustav zadataka iz primjera 4.5. na dvoprocesorskom sustavu:

$$
\begin{aligned}
Z_1: &\quad T_1 = 10 \text{ ms}, \quad C_1 = 5 \text{ ms} \\
Z_2: &\quad T_2 = 15 \text{ ms}, \quad C_2 = 5 \text{ ms} \\
Z_3: &\quad T_3 = 20 \text{ ms}, \quad C_3 = 1 \text{ ms}
\end{aligned}
$$



Slika 4.16.

I u ovom je primjeru provjera rasporedivosti mogla stati prije, u 6. ms s obzirom na to da je sustav razmatran u kritičnom slučaju.

4.6.1. Višeprocesorsko statičko raspoređivanje

- ideja je da se neki zadatci mogu izvoditi paralelno i tako prije obaviti neki složeniji posao
- razmatramo sustav zavisnih zadataka
  - za neke (ili sve) zadatke postoji veza „mora se obaviti prije”
  - zavisni zadaci se ne mogu izvoditi paralelno, samo nezavisni mogu
- kako odabrati koje zadatke kada izvoditi?
  - pokazati ćemo dva algoritma:
    - opće raspoređivanje
    - raspoređivanje stablastom strukturom

4.6.1.1. Opće raspoređivanje

- engl. *general scheduling*, (Nissanke, 1997.)

Definicija 4.18. Opće raspoređivanje

Opće raspoređivanje sustava zavisnih zadataka $\mathcal{S} = \{Z_1, Z_2, \ldots, Z_N\}$ na m procesora raspodjeljuje zadatke po procesorima u svrhu njihova što skorijeg završetka tako da se raspoređivanje razmatra **u intervalima Δt_k u kojima vrijedi:**

1. **podskup zadataka koji se izvode $\mathcal{S}' = \{Z_a, Z_b, \ldots\}$ čine međusobno nezavisni zadatci**
2. **svaki zadatak Z_i iz $\mathcal{S}'$ koristi se udjelom procesorske snage $\alpha_{i,k}$ za koji vrijedi:**

$$0 \leq \alpha_{i,k} \leq 1$$

3. **zadatci koji nisu u $\mathcal{S}'$ imaju:** $\alpha_{j,k} = 0$
4. **svi zadatci iz $\mathcal{S}'$ koriste dio raspoložive procesorske snage:**

$$\sum_{i=1}^{N} \alpha_{i,k} \leq m.$$

Primjeri općeg raspoređivanja

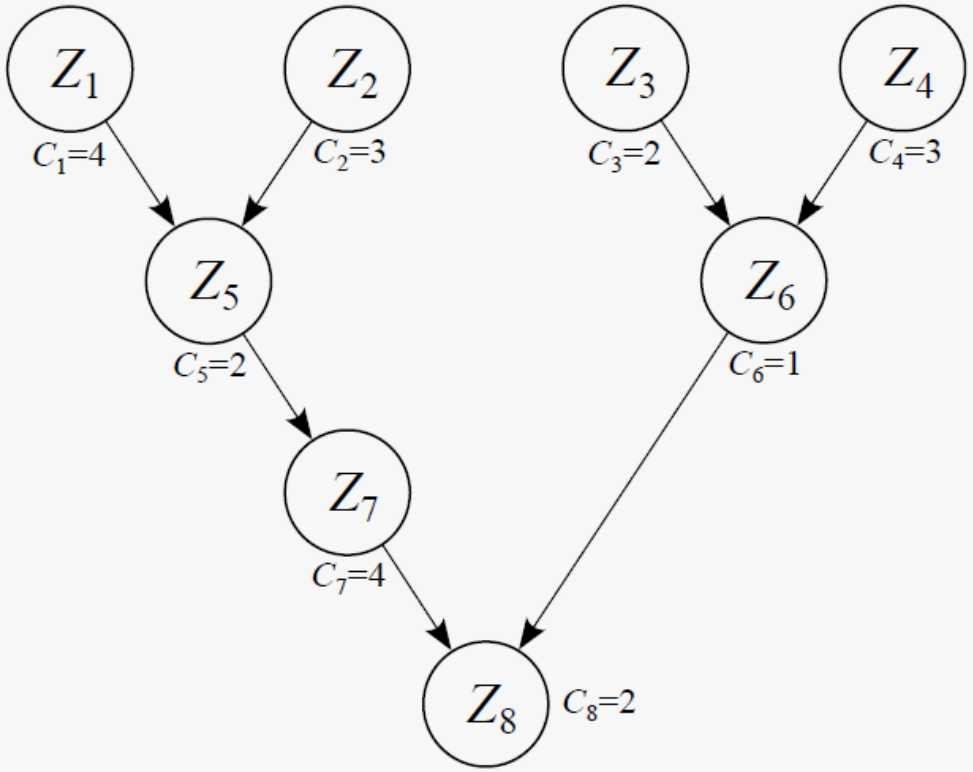
- u skripti, primjeri 4.12. – 4.14.

4.6.1.2. Postupak sa stablastom strukturom

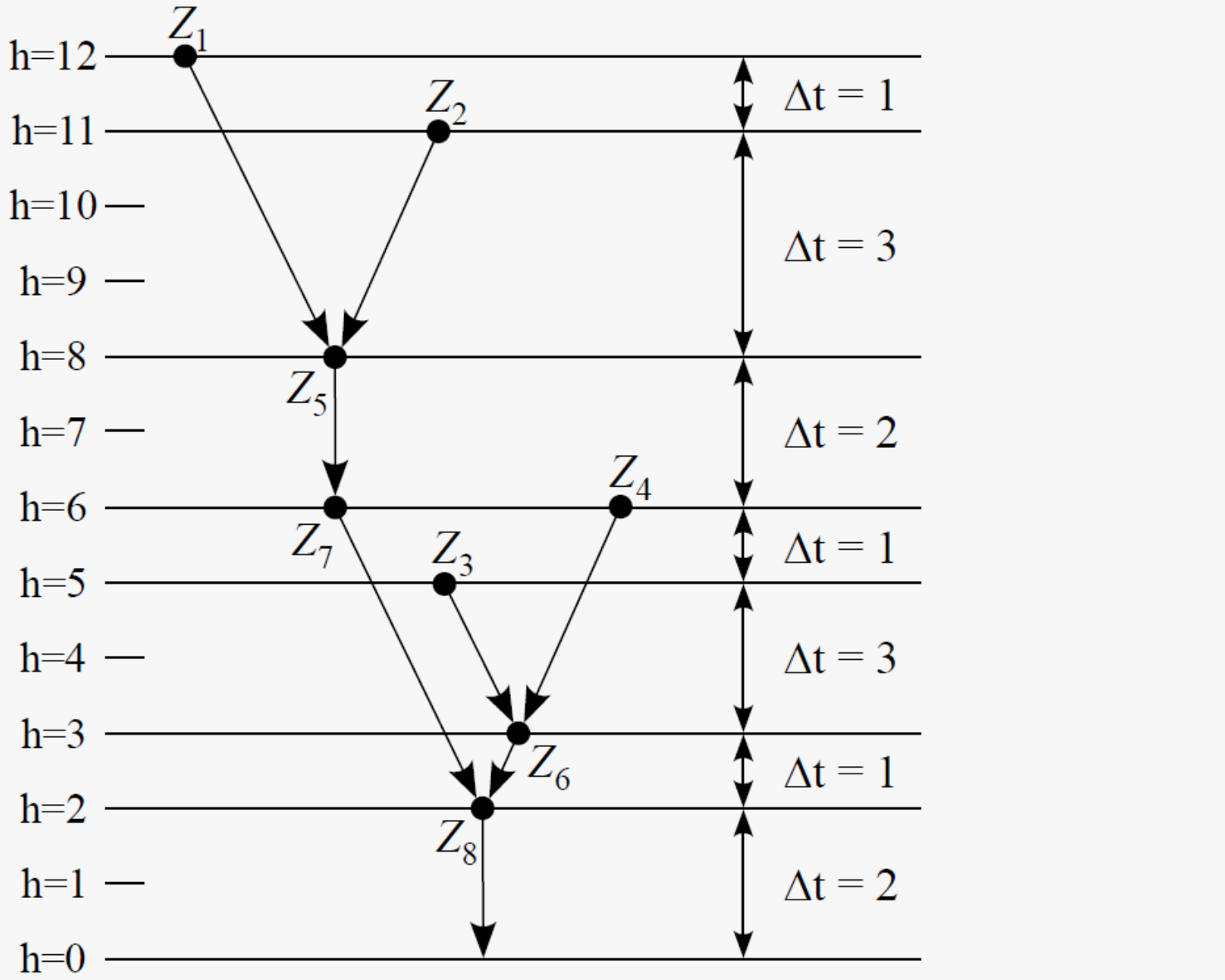
- graf koji prikazuje ovisnost je stablo
- listovi (na vrhu) su početni zadatci, korijen je završni zadatak
- svaki zadatak ima najviše jednog neposrednog sljedbenika
  - jedino završni zadatak nema sljedbenika
- postupa kreće od „kraja” prema „početku”
  - kraj: kad sve završava – kad zadatak u korijenu stabla bude gotov
  - početak: kada prvi zadatci u listovima kreću
- postupak neće uvijek stvoriti optimalno rješenje kao opće raspoređivanje, ali je zato mnogo jednostavniji
- idući primjer je grafički riješen primjenom ovog postupka

Primjer 4.15.

Sustav zadataka zadan je stablastom strukturom prema slici 4.22.



Slika 4.22.



Slika 4.23. Raspoređivanje stablastom strukturom na dvoprocesorskom računalu

4.6.2. Višeprocesorsko dinamičko raspoređivanje

- razmatramo dva postupka:
  - **raspoređivanje prema rokovima (EDF)**
    - identično jednoprocesorskim, već prikazano i na primeru
  - **raspoređivanje prema labavosti**
    - u jednoprocesorskim sustavima je jednako dobro kao i EDF, a složenije je te ga tamo nismo ni opisivali

Raspoređivanje prema labavosti

- Labavost (engl. *laxity*, *slack time*) označava vrijeme koje zadatak može provesti a da se ne izvodi, no kasnije se ipak stigne obaviti prije svog roka.

Definicija 4.19. Labavost

Neka je zadatak Z u trenutku t zadan sa $Z(t) = \{C(t), d\}$, gdje $C(t)$ predstavlja preostalo potrebno procesorsko vrijeme za završetak zadatka te d rok završetka. Labavost zadatka $\ell(t)$ definira se prema:

$$\ell(t) = d - t - C(t) \tag{4.10.}$$

Vrijeme do roka ("relativan d") često se označava s $đ(t) = d - t$. Uz pomoć $đ(t)$ labavost se definira s $\ell(t) = đ(t) - C(t)$.

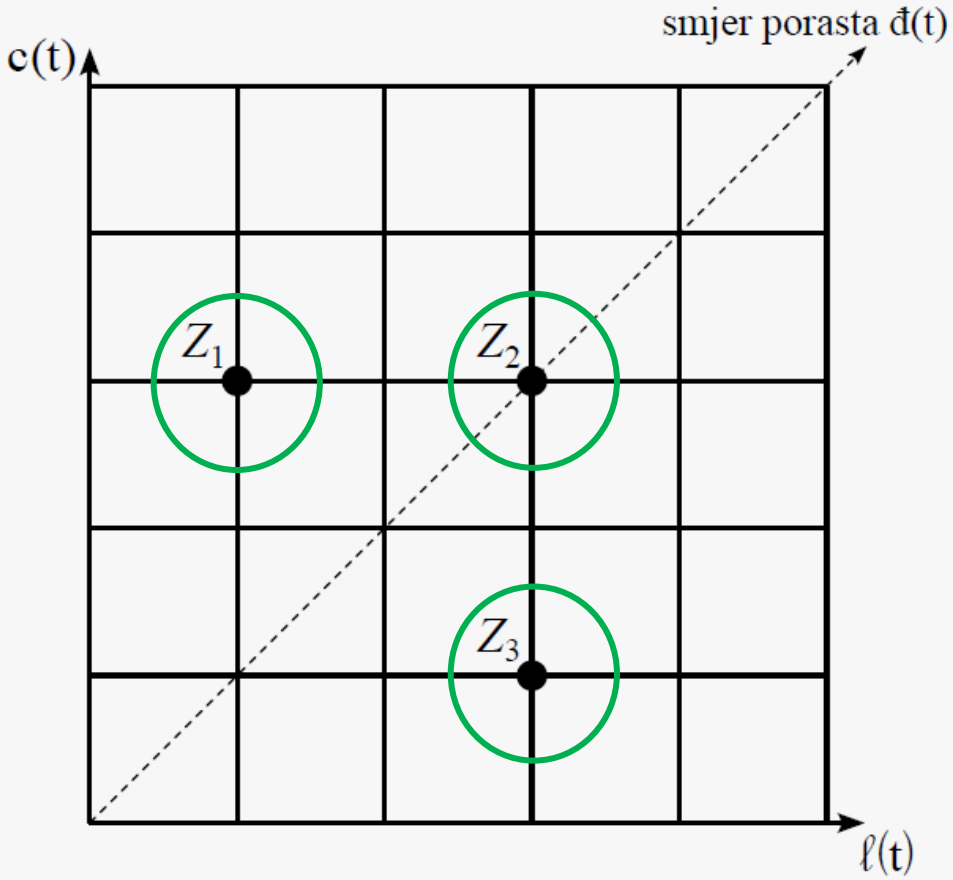
Raspoređivanje prema labavosti (2)

- Labavost $\ell(t)$ – mjera hitnosti pokretanja zadatka
  - veća labavost – manja hitnost
  - manja labavost – veća hitnost
  - labavost je nula – zadatak se odmah mora pokretati i izvoditi bez prekidanje jer inače neće stići do roka
  - labavost je manja od nule – zadatak se više nikako ne stigne obaviti do roka

- Raspoređivanje prema najmanjoj labavosti NL (engl. least laxity first – LLF)
  - prvo se odabire zadatak s najmanjom labavosti

Primjer 4.16.

Slika 4.24. prikazuje primjer ℓ-c grafa s tri zadatka u trenutku t.



Slika 4.24. ℓ-c graf u trenutku t

Relativni rokovi:

$$đ_1(t) = \ell_1(t) + C_1(t) = 4$$
$$đ_2(t) = \ell_2(t) + C_2(t) = 6$$
$$đ_3(t) = \ell_3(t) + C_3(t) = 4$$

Odabir raspoređivača prema NL:

1. Z_1
2. Z_2 i Z_3 (ista labavost)

Kada više zadataka ima istu labavost odabir jednog od njih je proizvoljan ili se koristi sekundarni kriterij (npr. prema rokovima)

4.6.2.2. Zalihost računalne snage

- koliko se procesorskog vremena može odvojiti za druge poslove a da zadani sustav zadataka ipak stigne sve prema rokovima
- ideja je da se može procijeniti što ako dođu neki prekidi ili drugi (ne)očekivani (hitni) poslovi – stigne li se i njih obraditi

Definicija 4.20. Zalihost računalne snage – $F(m, \mathcal{S}, t, x)$

U trenutku t zalihost računalne snage u sustavu s m procesora za budući period $[t, t + x]$ iznosi:

$$F(m, \mathcal{S}, t, x) = m \cdot x - \sum_{Z \in R_1} C_Z(t) - \sum_{Z \in R_2} (x - \ell_Z(t))$$

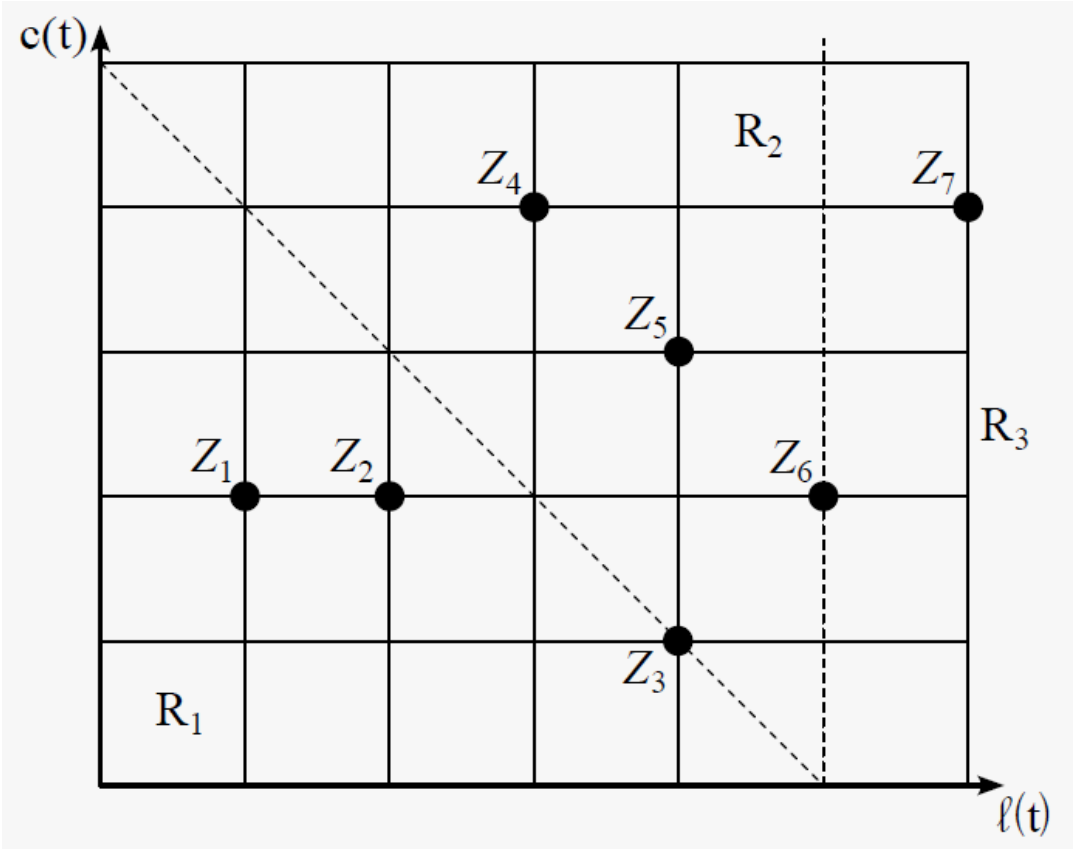
gdje $R_1 \subseteq \mathcal{S}$ predstavlja skup zadataka koji moraju biti gotovi do trenutka $t + x$, tj.

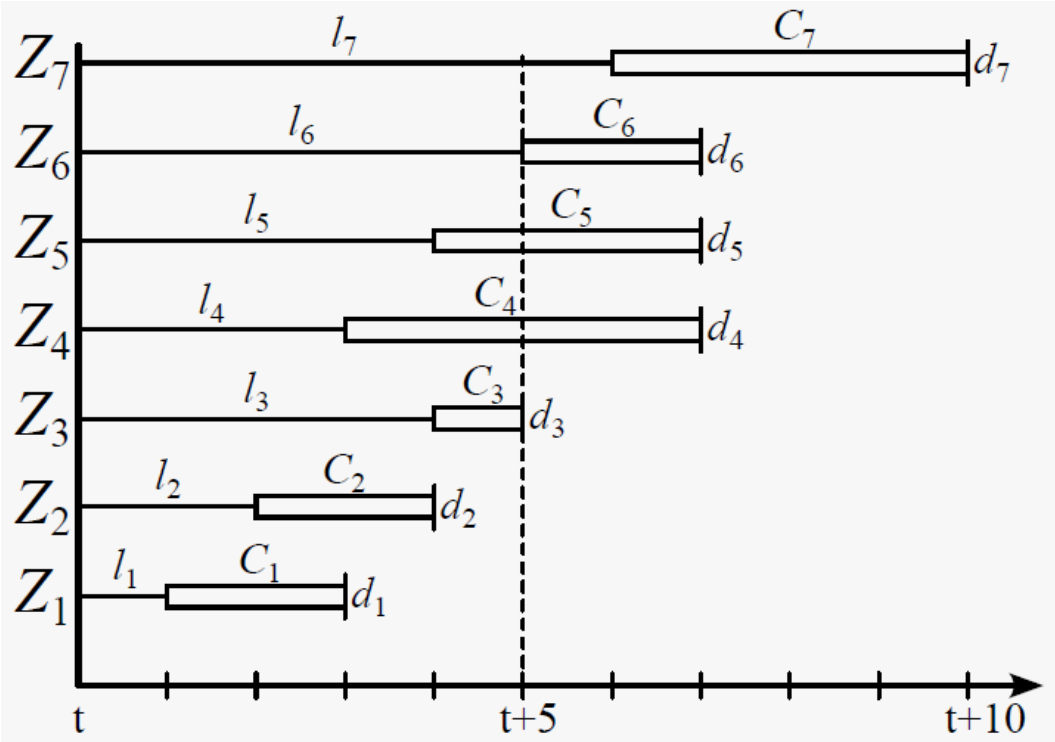
$$đ_Z(t) = C_Z(t) + \ell_Z(t) \leq x \quad \forall Z \in R_1$$

dok $R_2 \subseteq \mathcal{S}$ predstavlja skup zadataka koji trebaju djelomično obaviti svoj posao u intervalu $[t, t + x]$, a da bi stigli završiti prije svog d, tj.

$$đ_Z(t) > x \quad \wedge \quad \ell_Z(t) < x, \quad \forall Z \in R_2$$

Primjer 4.17. Zalihost računalne snage





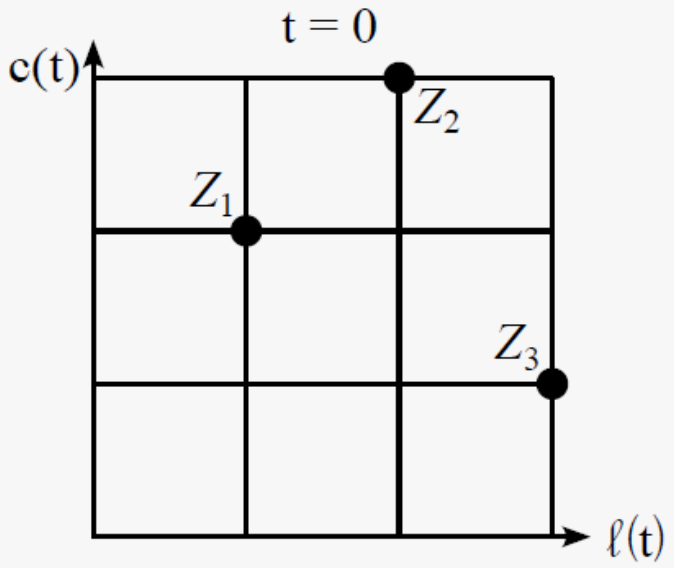
Slika 4.26. Svojstva pojedinih zadataka u trenutku t

$$F(m,\mathcal{S},t,5) = m\cdot 5 - (C_1(t)+C_2(t)+C_3(t)) - ((5-\ell_4)+(5-\ell_5)+(5-\ell_6))$$
$$= m\cdot 5 - (2+2+1) - ((5-3)+(5-4)+(5-5)) = m\cdot 5 - 8$$

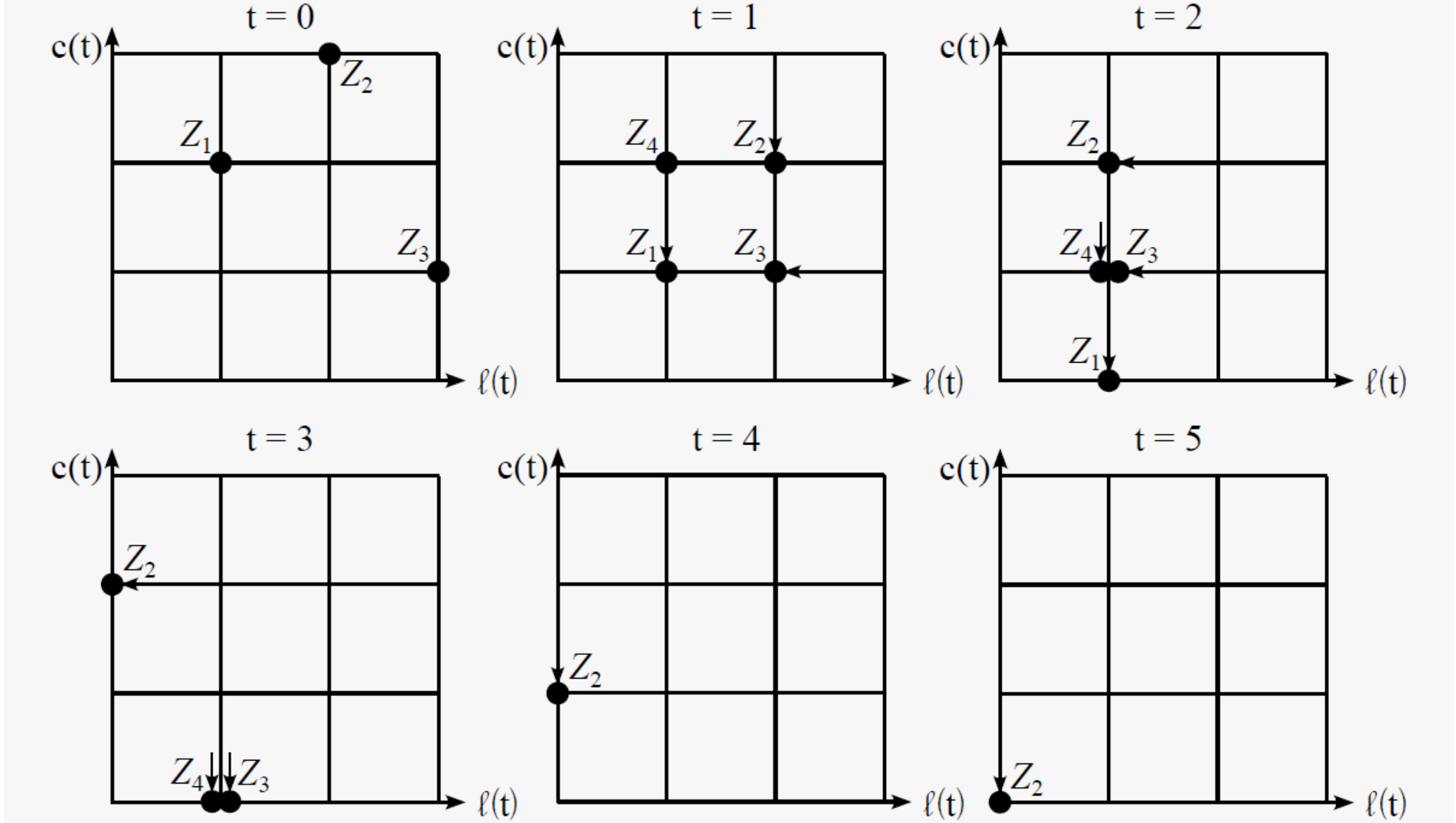
Primjerice, za dvoprocesorski sustav zalihost iznosi: $F(2,\mathcal{S},t,5) = 2\cdot 5 - 8 = 2.$

Primjer 4.18.

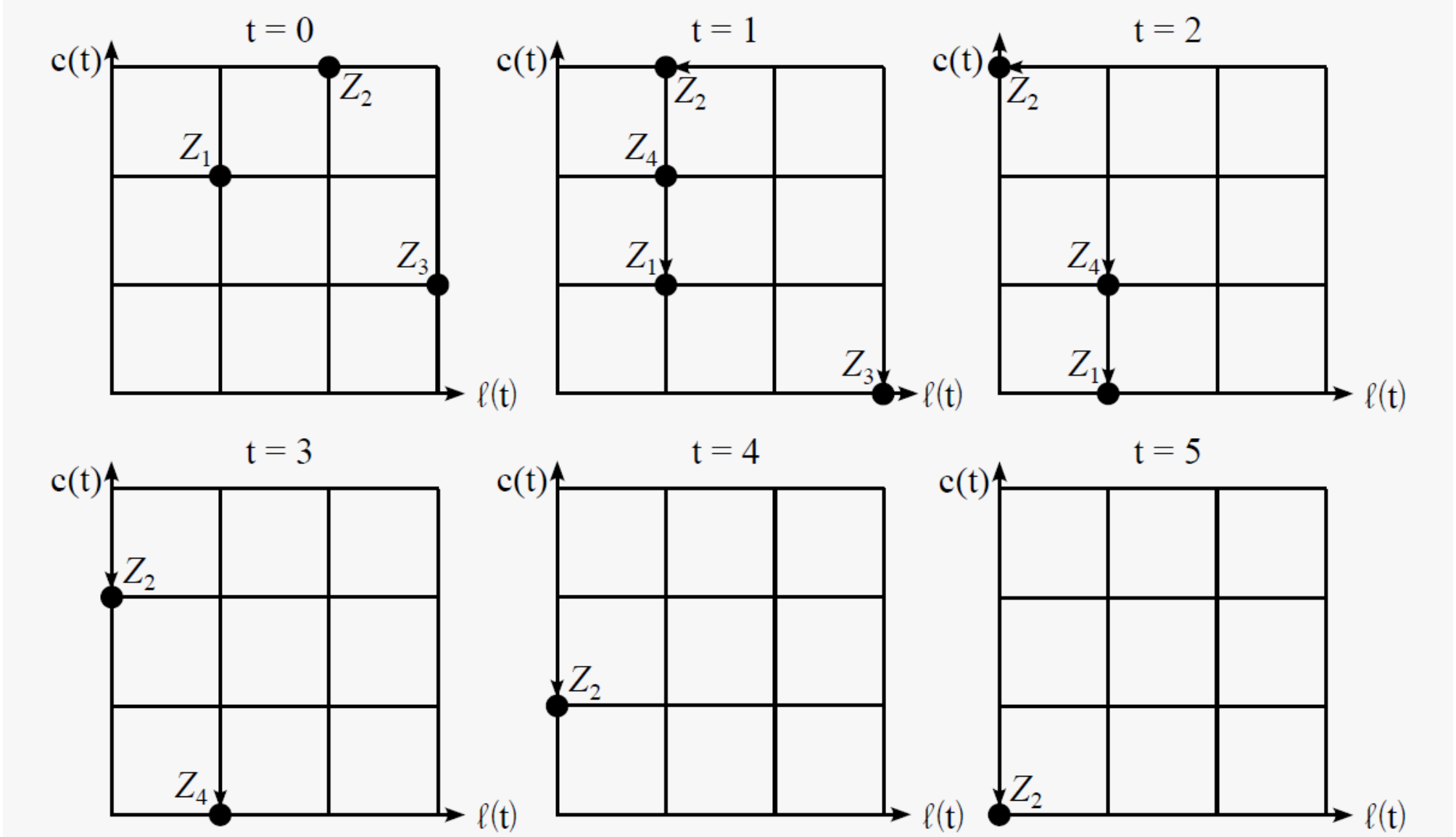
Sustav zadataka u trenutku $t = 0$ zadan je ℓ-c grafom na slici 4.27. Dodatno je poznato da će se u trenutku $t = 1$ pojaviti još jedan zadatak s potrebnim vremenom računanja od 2 jedinice koji mora biti gotov do $t = 4$. Prikažite rad raspoređivača prema najmanjoj labavosti kao primarnom kriteriju i prema rokovima kao sekundarnom, i obratno. Na raspolaganju je **dvoprocesorsko računalo**, a raspoređivač se poziva nakon svake jedinice vremena (za raspoređivanje prema NL).



Slika 4.27. Početno stanje sustava zadataka



Slika 4.28. Raspoređivanje prema najmanjoj labavosti



Slika 4.29. Raspoređivanje prema rokovima

4.7. Postupci raspoređivanja i njihova optimalnost

Optimalnost postupaka raspoređivanja može se promatrati s nekoliko različitih gledišta. Neki od njih su:

1. **zadovoljavanje vremenskih ograničenja zadataka** – izvedivost raspoređivanja
2. **jednostavnost postupka**:
   - radi mogućnosti **ostvarenja u stvarnim sustavima**
   - radi **smanjenja procesne snage utrošene na odlučivanje** o raspoređivanju i kućanske poslove raspoređivača
   - radi malog broja potrebnih obilježja zadataka koji su dostupni unaprijed ili za vrijeme izvođenja
3. **optimiranje završetka skupine zavisnih zadataka**.

4.7.1. Optimalnost prema izvedivosti raspoređivanja

- usporedba ima smisla samo ***unutar iste klase postupaka***
  1. postupci za jednoprocesorske sustave sa statičkom dodjelom prioriteta
  2. dinamički postupci za jednoprocesorske sustave
  3. dinamički postupci za višeprocesorske sustave.

Definicija 4.21. Optimalan postupak raspoređivanja

Za postupak raspoređivanja može se reći da je optimalan ako se njime može raspo-rediti bilo koji skup zadataka koji se može rasporediti i nekim drugim postupkom iz iste klase raspoređivača.

Optimalni postupci za jednoprocesorska računala

- sa statičkom dodjelom prioriteta: mjera ponavljanja (dokaz u [Liu, 1973])
- dinamički: prema rokovima i prema labavosti
  - sve sustave koji zadovoljavaju nužan uvjet ($U \leq 1$) oni rasporede

Optimalni postupci za višeprocesorska računala

- mnogo složenije; nismo ni naveli rasporedivost niti za jedan postupak nekim formulama (do sada)

Definicija 4.22. Izvodljivost raspoređivanja

Nužan i dovoljan uvjet za izvodljivost raspoređivanja sustava zadataka (nekim optimalnim postupkom) jest:

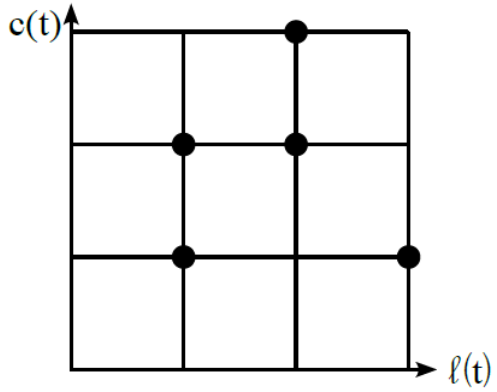
$$F(m, \mathcal{S}, t, x) \geq 0, \quad \forall x > 0$$

gdje $F(m, \mathcal{S}, t, x)$ predstavlja zalihost računalne snage u intervalu $[t, t + x]$ koji se izračunava prema definiciji 4.20.

- iako ispravna, definicija nam je „beskorisna” za provjeru (presložena)

Usporedba: prema rokovima vs prema labavosti?

- za labavost nam treba i C za svaki zadatak pa bi očekivali da je bolji



Slika 4.30. Primjer sustava u ℓ-c grafu gdje RZ ne uspijeva, a NL uspijeva rasporediti sustav zadataka na dvoprocesorskom računalu

- na gornjem primjeru je bolji, ali kako dokazati da vrijedi općenito?
  - „intuitivno”: sadrži informaciju o roku i labavosti, pa implicitno je bar dobar i kao prema rokovima
- je li NL optimalan među dinamičkim? dovoljno je naći kontra primjer

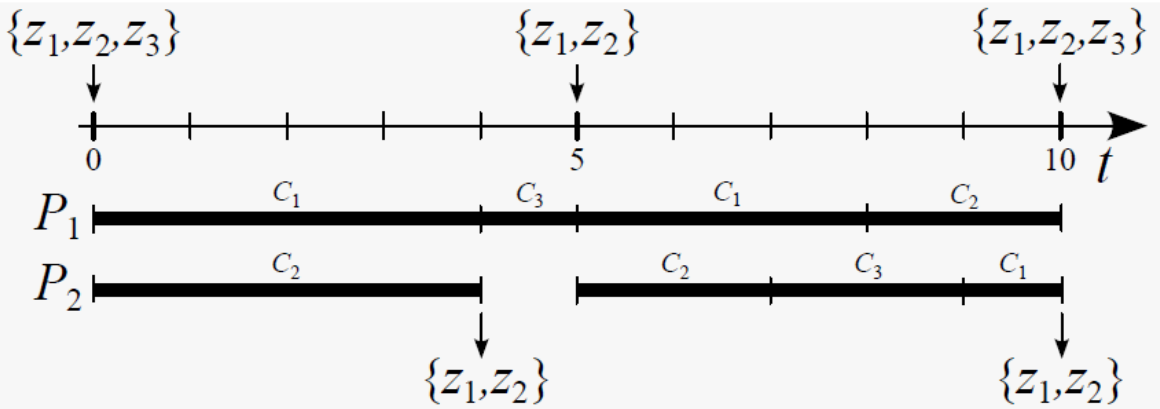
Je li NL optimalan među dinamičkim?

- dovoljno je naći kontra primjer za odgovor NE (ako postoji, a postoji ☹)

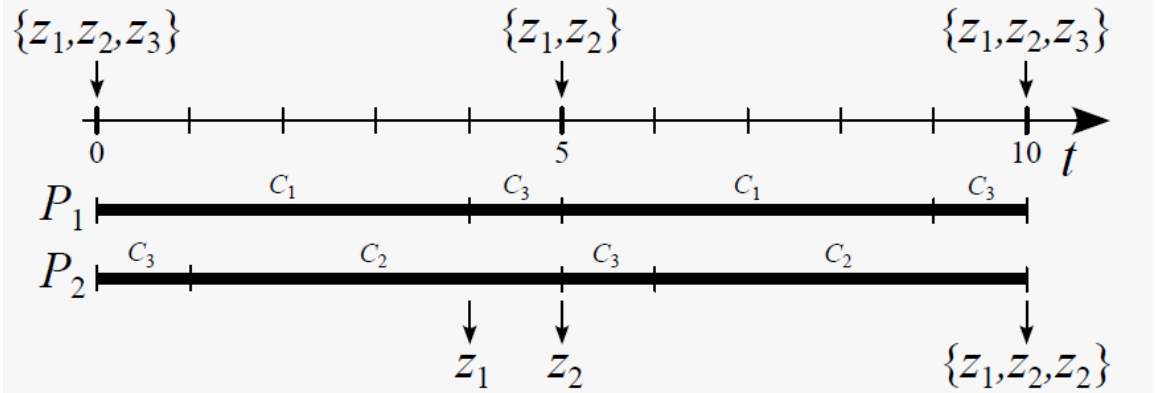
Primjer 4.19. Sustav koji NL ne može rasporediti

Neka je zadan sustav periodičkih zadataka koji treba izvoditi na dvoprocesorskom računalu:

$$Z_1: \quad T_1 = 5 \text{ ms}, \quad C_1 = 4 \text{ ms}$$
$$Z_2: \quad T_2 = 5 \text{ ms}, \quad C_2 = 4 \text{ ms}$$
$$Z_3: \quad T_3 = 10 \text{ ms}, \quad C_3 = 4 \text{ ms}$$



Slika 4.31. NL raspoređivanje uz indeks zadatka kao drugi kriterij



Slika 4.32. Raspored (ručni) koji zadovoljava sva ograničenja

4.8. Problemi određivanja rasporeda u stvarnim sustavima

- u stvarnim sustavima mnoge stvari nisu poznate
  - T – periode uglavnom jesu (osim za aperiodičke i sporadične zadatke)
  - C – vrijeme računanja u periodi – uglavnom je poznat najgori slučaj (max(C))
  - d – rok završetka obično nije trenutak početka iduće periode već neko vrijeme blizu početka same periode
- ponekad se izvođenje (dijela) zadatka ne smije prekidati
- što se onda koristi?
  - raspoređivanje prema prioritetu (uz mjeru ponavljanja za dodjelu prioriteta)
  - raspoređivanje prema rokovima
  - rasporedivost se „rješava” povećanjem procesorske snage

4.8.1. Statičko raspoređivanje za jednostavne zadatke

- jednostavni mikrokontroleri
- jednostavni periodički zadatci
  - sve što im je potrebno je u strukturama podataka
  - ne treba pamtiti kontekst njihovih dretvi od jednog pokretanja do drugog (u različitim periodama)
  - ne blokiraju se, ne koriste sinkronizacije
  - mogu slati i primati poruke, ali bez blokiranja
  - kratko traju – poznato maksimalno vrijeme rada u periodi
  - zadatci se mogu prekidati prekidima, ali ne drugim zadatcima

4.8.1. Statičko raspoređivanje za jednostavne zadatke (2)

- napravi se statički raspored na **hiper-periodi**
  - hiper-perioda – najmanji višekratnik perioda svih zadataka
  - „ručno” se „poslažu” zadaci u tu hiper-periodu

- kako/kada pokretati zadatke? → **periodički prekidi**
- minimalno korištenjem najkraće periode, često i puno kraćom periodom
- na svaki prekid obrađuje se bar jedan zadatak
- prekid može poslužiti i za prekidanje zadataka koji predugo traju (zbog greške ili drugih razloga)

Primjer 4.20. Zadatci po odsječcima

Zadan je sustav koji se sastoji od jednostavnih periodičkih zadataka Z_1–Z_4 s periodama $T_1 = 1$ ms, $T_2 = 2$ ms, $T_3 = 4$ ms te $T_4 = 8$ ms. Očekivana trajanja izvođenja zadataka su do 300 μs po zadatku. Osim navedenih zadataka sustav obrađuje i sporadične zadatke, a obrađuje ih mehanizmom prekida. Uz pretpostavku da sporadični zadatci neće uzeti više od 100 μs unutar svake milisekunde, napravite jedan statički raspored zadataka i pokažite funkciju za obradu alarma koji se javlja svake 1 ms.

Jedno moguće rješenje:

Hiperperioda je $T_H = \mathrm{lcm}(1, 2, 4, 8) = 8$ ms. Može se podijeliti u odsječke različitih veličina. U ovom rješenju odabrana je duljina od 1 ms i za svaki takav odsječak napravljen zaseban raspored.

Prvi zadatak Z_1 pojaviti će se u svakom odsječku jednom.

Zadatak Z_2 pojaviti će se u svakom drugom, počevši od prvog.

Zadatak Z_3 bismo također mogli staviti u prvi odsječak, ali onda ne bilo slobodna prostora za eventualne prekide (osim jednog sporadičnog zadatka) ili eventualno prekoračenje izvođenja od 300 μs nekog od zadataka. Stoga je u predloženom rješenju za Z_3 kao prvi odsječak odabran drugi (i svaki četvrti idući, tj. šesti u hiperperiodi).

Zadatak Z_4 je sličnom logikom kao i za Z_3 postavljen u četvrti odsječak.

1			2			3			4			5			6			7			8	
Z_1	Z_2	–	Z_1	Z_3	–	Z_1	Z_2	–	Z_1	Z_4	–	Z_1	Z_2	–	Z_1	Z_3	–	Z_1	Z_2	–	Z_1	–

Slika 4.33. Zadatci po odsječcima hiperperiode

Ostvarenje upravljanja u pseudokodu pretpostavlja da su zadane funkcije za svaki zadatak. Neka su one označene sa `z1`, `z2`, `z3` i `z4`. Tada rješenje može izgledati kao u nastavku.

```
//zadatci po odsječcima
Z[] = {{z1,z2}, {z1,z3}, {z1,z2}, {z1,z4}, {z1,z2}, {z1,z3}, {z1,z2}, {z1}}

t = 0 //idući odsječak; u ovom rješenju indeksi idu od 0 do 7

alarm_svake_1ms
   za i = 0 do duljina_niza(Z[t])-1
      Z[t][i]() //pozovi funkciju zadatka
   t = (t + 1) mod 8

glavni_program
   prazna beskonačna petlja
```

U rješenju se na svaki prekid alarma pokrenu zadatci raspoređeni u taj odsječak (određen varijablom `t`).

Problem može nastati kad neki zadatak ne završi u predviđenom intervalu. Problem je moguće riješiti na razne načine, korištenjem nadzorna alarma, prilagodbom funkcije za obradu alarma ili drukčije. Rješenje jako ovisi o zahtjevima sustava i potrebnim akcijama u slučaju prekoračenja.

Primjer 4.21. Tablični raspored

U sustavu se nalaze zadatci različita tipa (A, B, C) tako da se svaki zadatak treba obaviti jednom unutar svoje periode. Zadatci tipa A su a_1, a_2 i a_3 s periodom $T_A = 1$ ms. Zadatci tipa B su b_1, b_2 $\ldots b_{15}$ s periodom $T_B = 5$ ms. Zadatci tipa C su c_1, c_2 $\ldots c_{17}$ s periodom $T_C = 20$ ms. Očekivano trajanje zadataka je ispod 100 μs. Zadatke treba pokretati na alarm.

Jedno moguće rješenje:

Alarm se javlja periodički svakih 100 μs što daje 10 mogućnosti za pokretanje zadataka unutar svake milisekunde.

Zadatci tipa A svi se moraju pojaviti unutar svake milisekunde.

Zadatci tipa B, njih 15, svi se moraju pojaviti jednom unutar svakih 5 ms. Kada bismo ih jednoliko rasporedili u tom intervalu, tada unutar jedne milisekunde treba izvesti po 3 zadatka tipa B.

Na sličan način možemo zaključiti da bi po jedan zadatak tipa C trebalo izvesti svake milisekunde s time da u tri milisekunde (od 20) taj tip zadatka ne bi bio prisutan. Da se pojednostavi takav problem, najjednostavnije je dodati tri prazna zadatka tipa C tako da se i te tri milisekunde popune takvim tipom zadataka.

Ukupno bi unutar jedne miliskenude trebalo obaviti: 3 zadatka tipa A, 3 zadatka tipa B te jedan tipa C što ukupno čini 7 zadataka. Uz ubacivanje praznih intervala mogući raspored izvođenja zadataka unutar svake milisekunde može biti kao na slici 4.34.

1	2	3	4	5	6	7	8	9	10
A	B	–	A	B	–	A	B	C	–

Slika 4.34. Raspored poziva tipova zadataka unutar jedne milisekunde

U različitim prekidima alarma označenim istim tipom zadatka treba pozvati različite zadatke. Primjerice, prva oznaka B bit će poziv `b1`, druga `b2`, treća `b3`, četvrta (kada krene iduća milisekunda) `b4`, pa `b5` i tako dalje sve do `b15` te onda opet `b1` i dalje. Slično je za zadatke tipa A i C. Primjer rješenja u obliku pseudokoda, uključujući strukture podataka i funkciju alarma, naveden je u nastavku.

```
//kazaljke na funkcije, zasebno po tipovima zadataka
zadA[] = {a1, a2, a3}
zadB[] = {b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, b12, b13, b14, b15}
zadC[] = {c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15,
          c16, c17, ništa, ništa, ništa} // "ništa" je prazna funkcija
zad[] = {zadA, zadB, zadC} //zad[x][y] = kazaljka na y-tu funkciju tipa x

//redoslijed tipova zadataka unutar milisekunde: 0=A, 1=B, 2=C, -1=ništa
slijed[] = {0, 1, -1, 0, 1, -1, 0, 1, 2, -1}

//koji po redu prekid unutar milisekunde (odabir elementa iz slijed[])
t = 0

//idući zadatak prema tipovima (A, B, C)
ind[] = {0, 0, 0} //odabir zadatka iz zad[slijed[t]][]
MAXTIP [] = {3, 15, 20}

alarm_svakih_100us
   tip = slijed[t]                  //koji je tip zadatka na redu
   ako je tip != -1 tada
      zad[tip][ind[tip]]()          //pozovi funkciju koja obavlja zadatak
      ind[tip] = (ind[tip] + 1) mod MAXTIP[tip]
   t = (t + 1) mod 10

glavni_program
   prazna beskonačna petlja
```