Systemy wbudowane - wykład 10

Przemek Błaśkiewicz

15 maja 2019

Definicja

System czasu rzeczywistego

Real-time system to system, którego poprawność działania zależy nie tylko od poprawności rezultatów (obliczeń, decyzji, akcji), ale również od czasu, kiedy zostaną one wypracowane (*czas reakcji*).

Gotowość

- Program (system) jest zawsze gotowy na napływające dane

Gotowość

- Program (system) jest zawsze gotowy na napływające dane
  - synchronicznie;

Gotowość

- Program (system) jest zawsze gotowy na napływające dane
  - synchronicznie;
  - asynchronicznie.

Gotowość

- Program (system) jest zawsze gotowy na napływające dane
  - synchronicznie;
  - asynchronicznie.
- Proces/system przetwarza dane w sposób przewidywalny, w rozumieniu:

Gotowość

- Program (system) jest zawsze gotowy na napływające dane
  - synchronicznie;
  - asynchronicznie.
- Proces/system przetwarza dane w sposób przewidywalny, w rozumieniu:
  - długości czasu przetwarzania;

Gotowość

- Program (system) jest zawsze gotowy na napływające dane
  - synchronicznie;
  - asynchronicznie.
- Proces/system przetwarza dane w sposób przewidywalny, w rozumieniu:
  - długości czasu przetwarzania;
  - na bieżąco (długości czasu oczekiwania na rozpoczęcie przetwarzania);

Gotowość

- Program (system) jest zawsze gotowy na napływające dane
  - synchronicznie;
  - asynchronicznie.
- Proces/system przetwarza dane w sposób przewidywalny, w rozumieniu:
  - długości czasu przetwarzania;
  - na bieżąco (długości czasu oczekiwania na rozpoczęcie przetwarzania);
  - spełnienia dodatkowych założeń dot. np. kolejności.

Gotowość

- Program (system) jest zawsze gotowy na napływające dane
  - synchronicznie;
  - asynchronicznie.
- Proces/system przetwarza dane w sposób przewidywalny, w rozumieniu:
  - długości czasu przetwarzania;
  - na bieżąco (długości czasu oczekiwania na rozpoczęcie przetwarzania);
  - spełnienia dodatkowych założeń dot. np. kolejności.
- Współdziała ze środowiskiem, reagując na jego niezależne zmiany.

Przypomnienie o RTOS

- Pojęcie “real-time” nie oznacza “szybki”.

Przypomnienie o RTOS

- Pojęcie “real-time” nie oznacza “szybki”.
- PC vs. SW vs. RTOS

Przypomnienie o RTOS

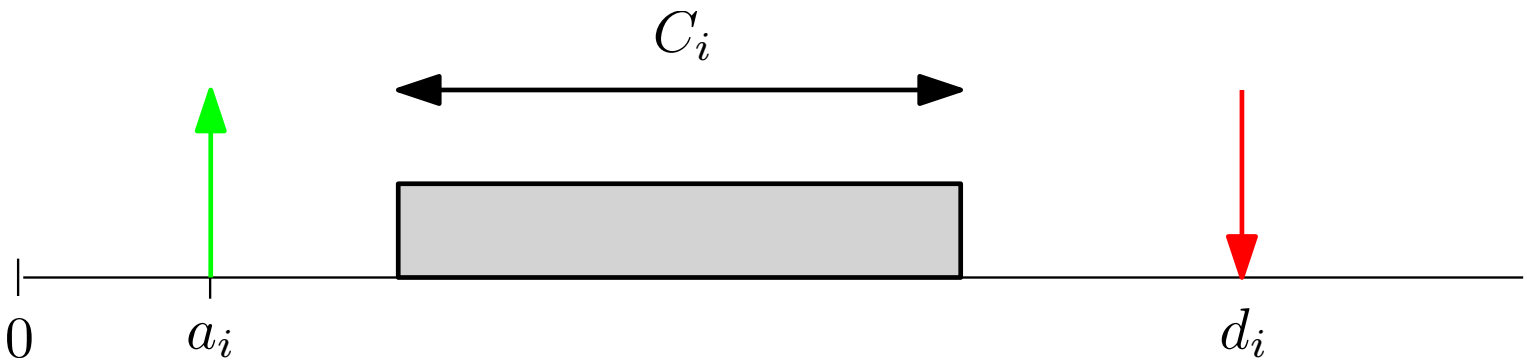
- Pojęcie “real-time” nie oznacza “szybki”.
- PC vs. SW vs. RTOS
- hard/soft/firm OS

Plan

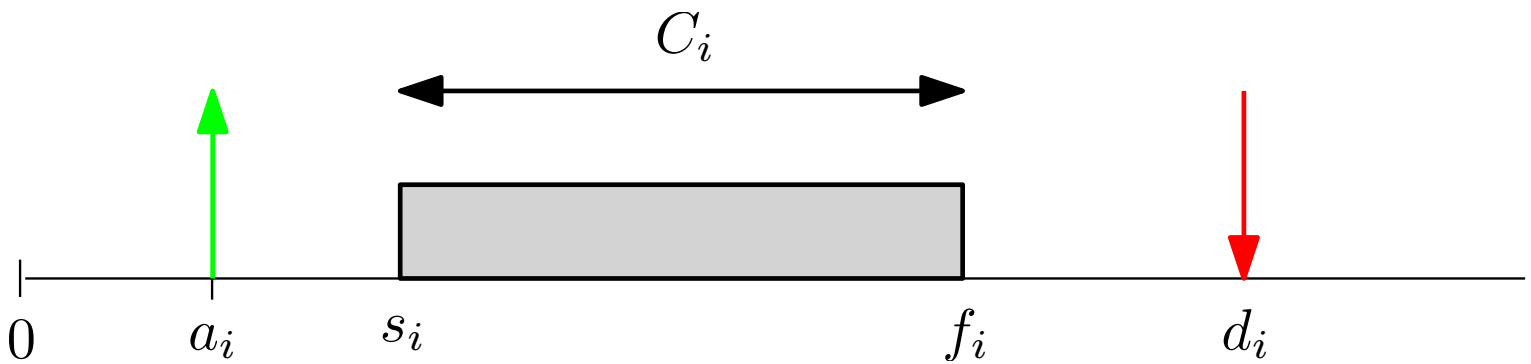
Plan (*schedule*) dla zbioru zadań $(J_1, J_2, \ldots, J_n)$ to pewna funkcja: $\sigma : R^+ \to \{0, \ldots, n\}$, taka, że:

$$\forall t \in R^+, \exists t_1, t_2 \in R^+ : t \in [t_1, t_2), \forall t' \in [t_1, t_2) : \sigma(t) = \sigma(t').$$

Cechy charakterystyczne zadań



Cechy charakterystyczne zadań



Algorytmy planowania (1)

Realizowalność planu

(*feasibility*) – plan jest realizowalny, jeśli jego wykonanie powoduje że wszystkie zadania kończą się nie później niż w swoim terminie wykonania.

Algorytmy planowania (1)

Realizowalność planu

(*feasibility*) – plan jest realizowalny, jeśli jego wykonanie powoduje że wszystkie zadania kończą się nie później niż w swoim terminie wykonania.

- wywłaszczające i niewywłaszczające;

Algorytmy planowania (1)

Realizowalność planu

(*feasibility*) – plan jest realizowalny, jeśli jego wykonanie powoduje że wszystkie zadania kończą się nie później niż w swoim terminie wykonania.

- wywłaszczające i niewywłaszczające;
- statyczne i dynamiczne;

Algorytmy planowania (1)

Realizowalność planu

(*feasibility*) – plan jest realizowalny, jeśli jego wykonanie powoduje że wszystkie zadania kończą się nie później niż w swoim terminie wykonania.

- wywłaszczające i niewywłaszczające;
- statyczne i dynamiczne;
- jednoprocesorowe, wieloprocesorowe;

Algorytmy planowania (1)

Realizowalność planu

(*feasibility*) – plan jest realizowalny, jeśli jego wykonanie powoduje że wszystkie zadania kończą się nie później niż w swoim terminie wykonania.

- wywłaszczające i niewywłaszczające;
- statyczne i dynamiczne;
- jednoprocesorowe, wieloprocesorowe;
- optymalne, heurystyczne;

Algorytmy planowania (1)

Realizowalność planu

(*feasibility*) – plan jest realizowalny, jeśli jego wykonanie powoduje że wszystkie zadania kończą się nie później niż w swoim terminie wykonania.

- wywłaszczające i niewywłaszczające;
- statyczne i dynamiczne;
- jednoprocesorowe, wieloprocesorowe;
- optymalne, heurystyczne;
- dla zadań okresowych (periodycznych) i asynchronicznych;

Lemat o wywłaszczaniu

Jeśli czasy nadejścia zadań są synchroniczne, to mechanizm wywłaszczenia nie polepsza maksymalnego opóźnienia. Czyli jeśli istnieje algorytm wywłaszczeniowy o opóźnieniu L_{max} to istnieje również algorytm niewywłaszczeniowy o tym samym opóźnieniu dla tego samego zestawu zadań.

Zadania aperiodyczne, synchroniczne

- Dany jest zestaw n zadań: $J_1, \ldots, J_n$, gdzie:

Zadania aperiodyczne, synchroniczne

- Dany jest zestaw n zadań: $J_1, \ldots, J_n$, gdzie:
  - dla wszystkich zadań $a_i = 0, i = 1, \ldots n$;

Zadania aperiodyczne, synchroniczne

- Dany jest zestaw n zadań: $J_1, \ldots, J_n$, gdzie:
  - dla wszystkich zadań $a_i = 0, i = 1, \ldots n$;
  - każde zadanie ma określony deadline d_i i czas obliczenia C_i;

Zadania aperiodyczne, synchroniczne

- Dany jest zestaw n zadań: $J_1, \ldots, J_n$, gdzie:
  - dla wszystkich zadań $a_i = 0, i = 1, \ldots n$;
  - każde zadanie ma określony deadline d_i i czas obliczenia C_i;
  - zadania są niezależne (nie ma zależności wykonania).

Zadania aperiodyczne, synchroniczne

- Dany jest zestaw n zadań: $J_1, \ldots, J_n$, gdzie:
  - dla wszystkich zadań $a_i = 0, i = 1, \ldots n$;
  - każde zadanie ma określony deadline d_i i czas obliczenia C_i;
  - zadania są niezależne (nie ma zależności wykonania).
- system bez wywłaszczenia

Zadania aperiodyczne, synchroniczne

- Dany jest zestaw n zadań: $J_1, \ldots, J_n$, gdzie:
  - dla wszystkich zadań $a_i = 0, i = 1, \ldots n$;
  - każde zadanie ma określony deadline d_i i czas obliczenia C_i;
  - zadania są niezależne (nie ma zależności wykonania).
- system bez wywłaszczenia
- jednoprocesorowy

Zadania aperiodyczne, synchroniczne

- Dany jest zestaw n zadań: $J_1, \ldots, J_n$, gdzie:
  - dla wszystkich zadań $a_i = 0, i = 1, \ldots n$;
  - każde zadanie ma określony deadline d_i i czas obliczenia C_i;
  - zadania są niezależne (nie ma zależności wykonania).
- system bez wywłaszczenia
- jednoprocesorowy
- zadanie: znajdź optymalny plan

Planista EDD (earliest due date)

Planista EDD (earliest due date)

Wykonuje zadania w kolejności niemalejącego czasu zakończenia.

Planista EDD (earliest due date)

Wykonuje zadania w kolejności niemalejącego czasu zakończenia.

Przykład 1

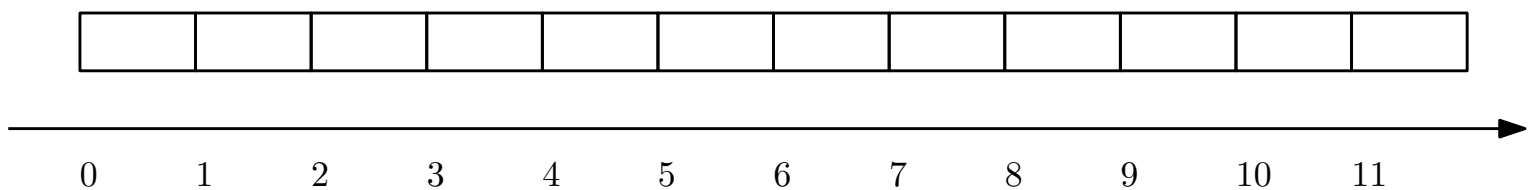
	J_1	J_2	J_3	J_4	J_5
C_i	1	1	1	3	2
d_i	3	10	7	8	5

Planista EDD (earliest due date)

Wykonuje zadania w kolejności niemalejącego czasu zakończenia.

Przykład 1

	J_1	J_2	J_3	J_4	J_5
C_i	1	1	1	3	2
d_i	3	10	7	8	5

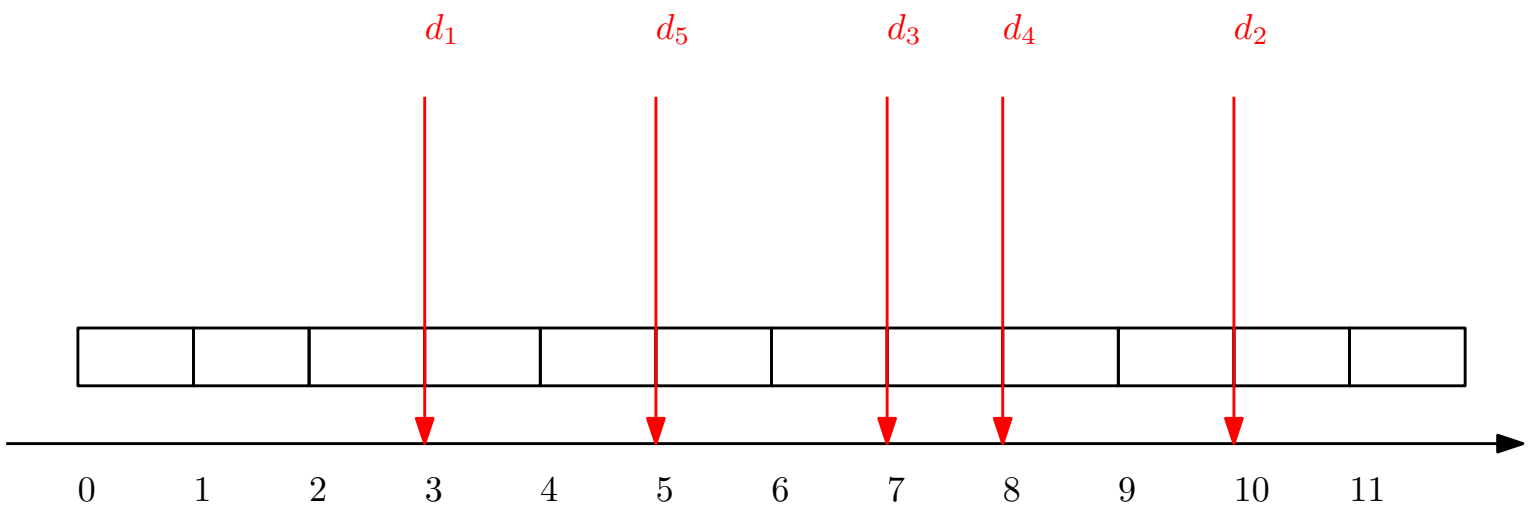


Planista EDD (earliest due date)

Wykonuje zadania w kolejności niemalejącego czasu zakończenia.

Przykład 1

	J_1	J_2	J_3	J_4	J_5
C_i	1	1	1	3	2
d_i	3	10	7	8	5

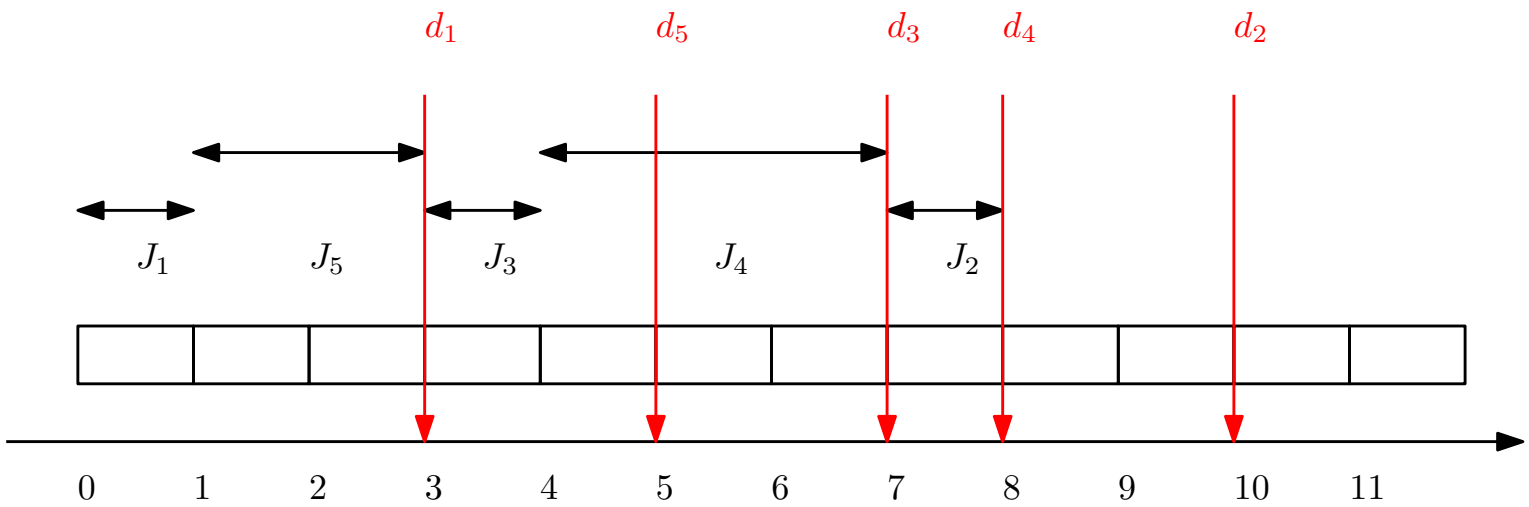


Planista EDD (earliest due date)

Wykonuje zadania w kolejności niemalejącego czasu zakończenia.

Przykład 1

	J_1	J_2	J_3	J_4	J_5
C_i	1	1	1	3	2
d_i	3	10	7	8	5



Planista EDD (earliest due date)

Wykonuje zadania w kolejności niemalejącego czasu zakończenia.

Przykład 2

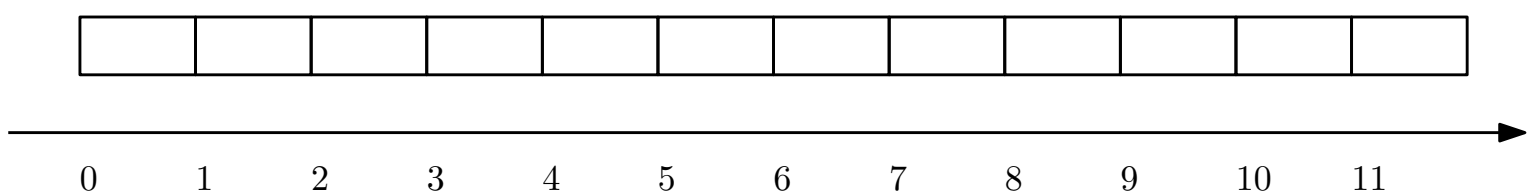
	J_1	J_2	J_3	J_4	J_5
C_i	1	**2**	1	**4**	2
d_i	**2**	**5**	**4**	8	**6**

Planista EDD (earliest due date)

Wykonuje zadania w kolejności niemalejącego czasu zakończenia.

Przykład 2

	J_1	J_2	J_3	J_4	J_5
C_i	1	**2**	1	**4**	2
d_i	**2**	**5**	**4**	8	**6**

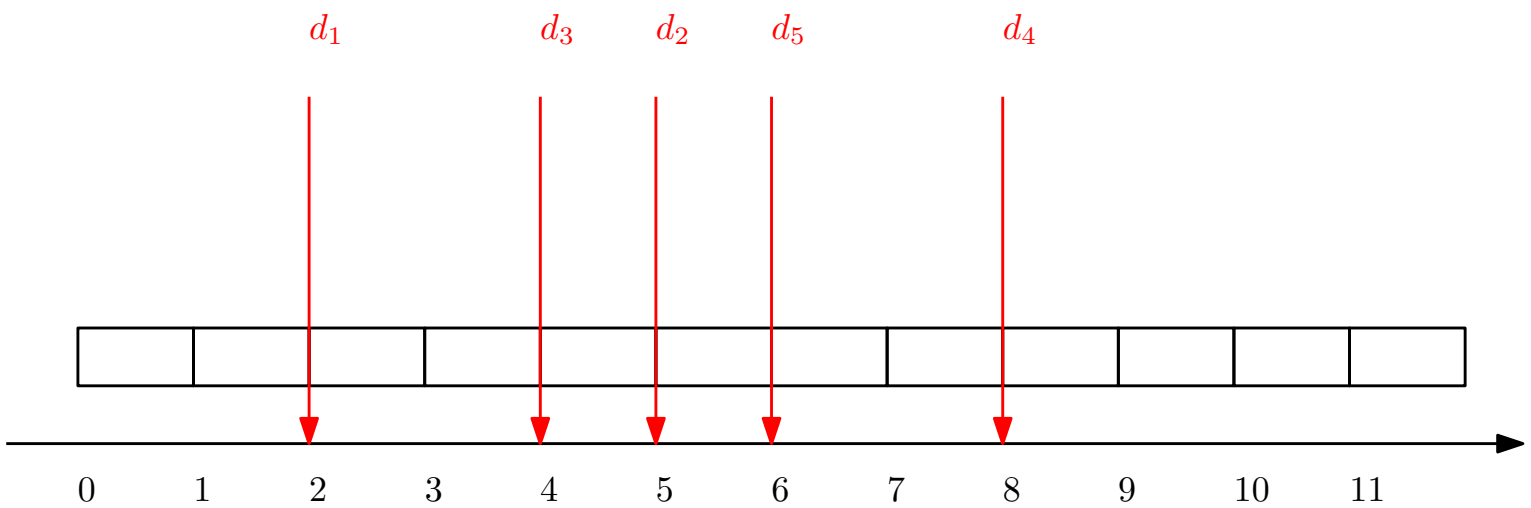


Planista EDD (earliest due date)

Wykonuje zadania w kolejności niemalejącego czasu zakończenia.

Przykład 2

	J_1	J_2	J_3	J_4	J_5
C_i	1	**2**	1	**4**	2
d_i	**2**	**5**	**4**	8	**6**

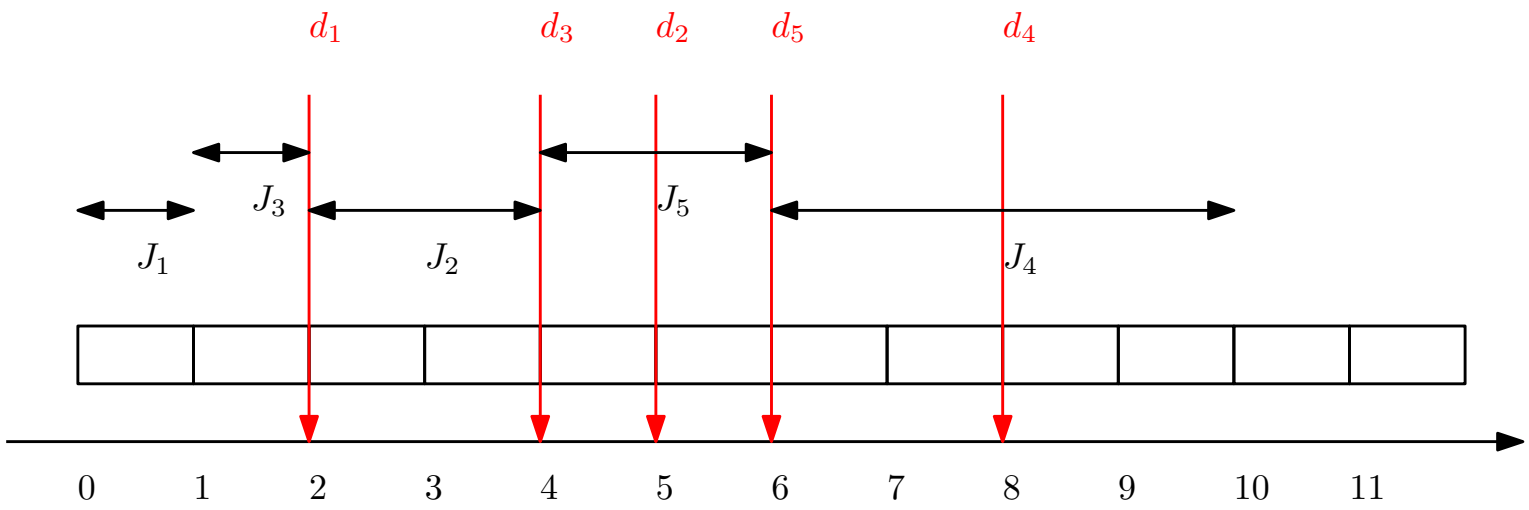


Planista EDD (earliest due date)

Wykonuje zadania w kolejności niemalejącego czasu zakończenia.

Przykład 2

	J_1	J_2	J_3	J_4	J_5
C_i	1	**2**	1	**4**	2
d_i	**2**	**5**	**4**	8	**6**



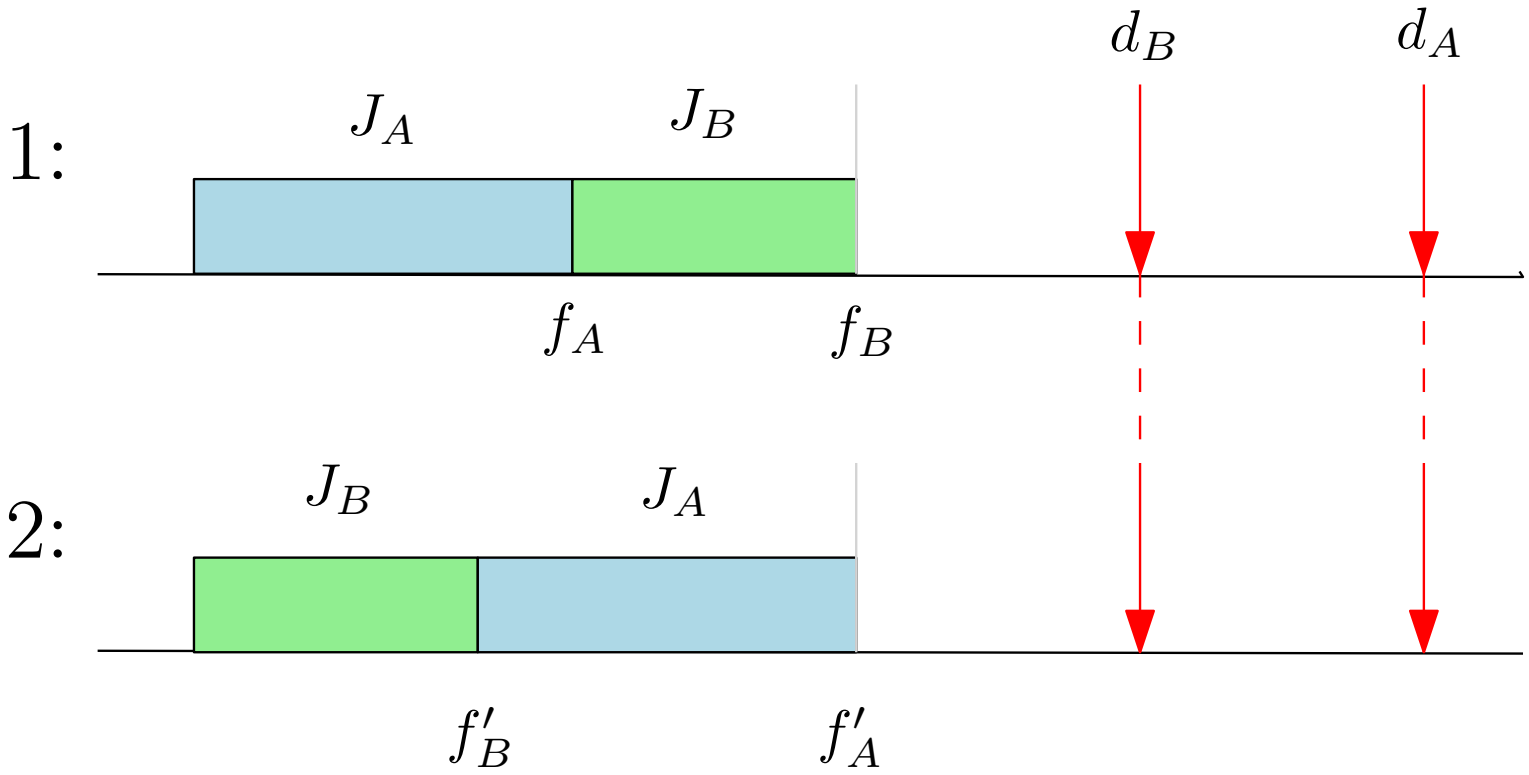
Twierdzenie o optymalności EDD

Twierdzenie o optymalności EDD

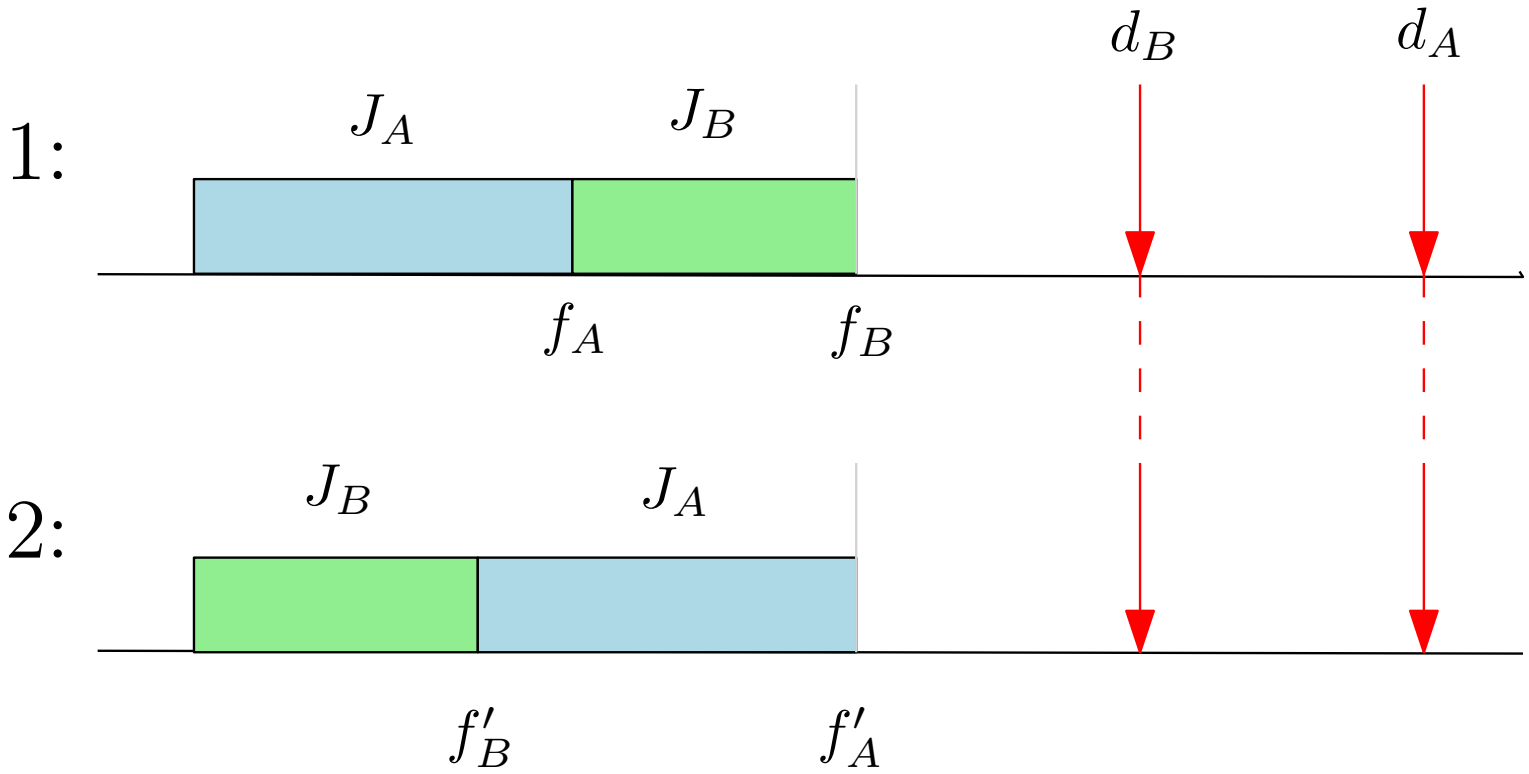
Jackson, 1955

Jeśli dany jest dowolny zestaw niezależnych zadań, pojawiających się w sytemie synchronicznie, to każdy algorytm wykonujący je w kolejności niemalejących terminów wykonania (deadlines) jest algorytmem optymalnym ze względu na minimalizację maksymalnego opóźnienia.

Twierdzenie o optymalności EDD

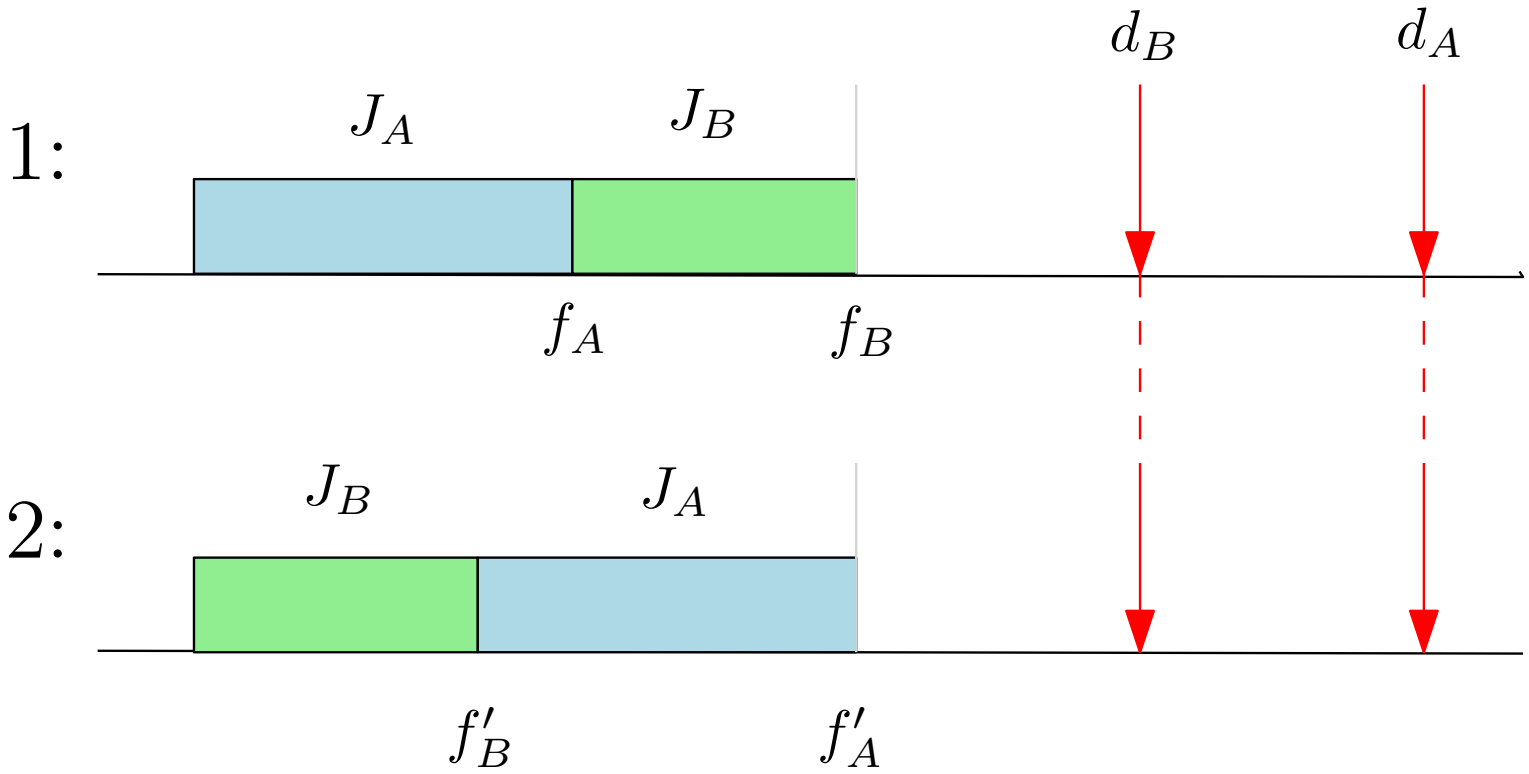


Twierdzenie o optymalności EDD



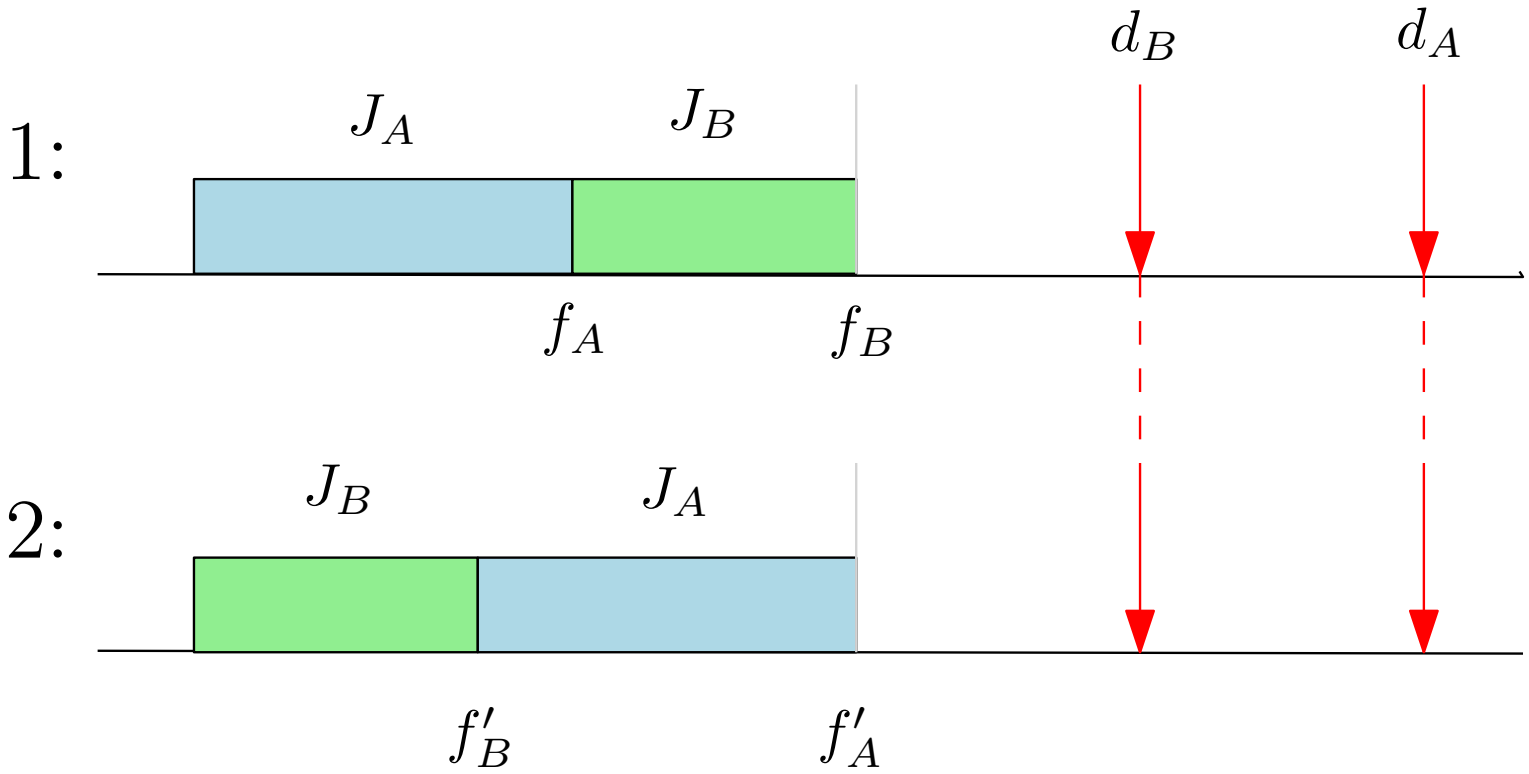
- A $f'_A - d_A = f_B - d_A \leq f_B - d_B$

Twierdzenie o optymalności EDD



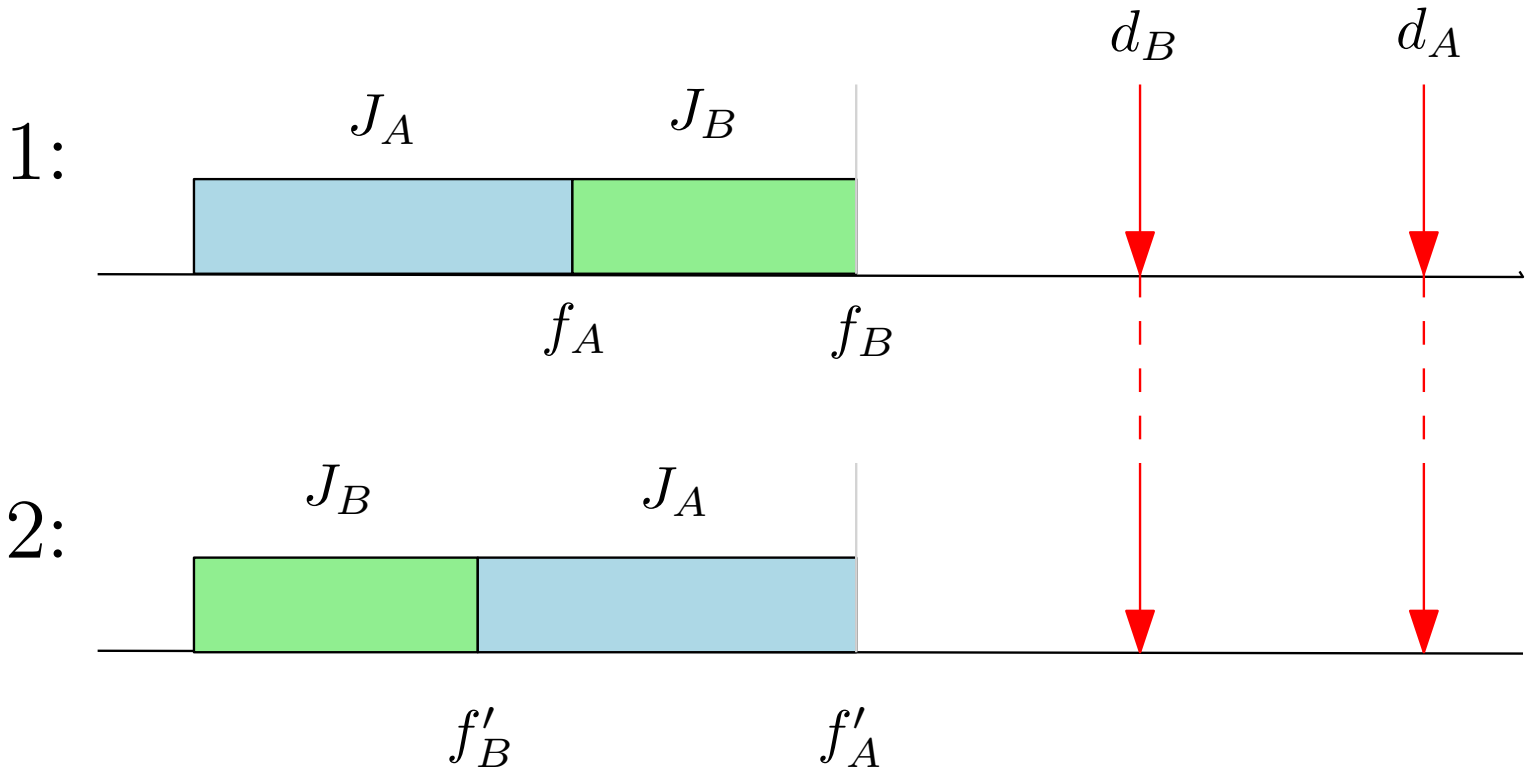
- A $f'_A - d_A = f_B - d_A \leq f_B - d_B$
- B $f'_B - d_B \leq f_B - d_B$

Twierdzenie o optymalności EDD



- A $f'_A - d_A = f_B - d_A \leq f_B - d_B$
- B $f'_B - d_B \leq f_B - d_B$

Twierdzenie o optymalności EDD



A $f'_A - d_A = f_B - d_A \leq f_B - d_B$

B $f'_B - d_B \leq f_B - d_B$

$$L'^{max}_{a,b} \leq L^{max}_{a,b}$$

EDD - właściwości

- złożoność EDD: sortowanie, $\mathcal{O}(n \log n)$

EDD - właściwości

- złożoność EDD: sortowanie, $\mathcal{O}(n \log n)$
- test na *planowalność* zadań $\{J_1, \ldots, J_n\}$:

EDD - właściwości

- złożoność EDD: sortowanie, $\mathcal{O}(n \log n)$
- test na *planowalność* zadań $\{J_1, \ldots, J_n\}$:
  1. posortuj listę zadań niemalejąco względem ich deadlines; (niech to będzie powyższy porządek);

EDD - właściwości

- złożoność EDD: sortowanie, $\mathcal{O}(n \log n)$
- test na *planowalność* zadań $\{J_1, \ldots, J_n\}$:
  1. posortuj listę zadań niemalejąco względem ich deadlines; (niech to będzie powyższy porządek);
  2. sprawdź, czy $\forall i \in 1, \ldots, n : \; f_i \leq d_i$;

EDD - właściwości

- złożoność EDD: sortowanie, $\mathcal{O}(n \log n)$
- test na *planowalność* zadań $\{J_1, \ldots, J_n\}$:
  1. posortuj listę zadań niemalejąco względem ich deadlines; (niech to będzie powyższy porządek);
  2. sprawdź, czy $\forall i \in 1, \ldots, n : \; f_i \leq d_i$;
     - ponieważ $f_i = \sum_{k=1}^{i} C_k$ warunek wystarczający:

$$\forall i \in 1, \ldots, n : \; \sum_{k=1}^{i} C_k \leq d_i$$

EDD - właściwości

- złożoność EDD: sortowanie, $\mathcal{O}(n \log n)$
- test na *planowalność* zadań $\{J_1, \ldots, J_n\}$:
  1. posortuj listę zadań niemalejąco względem ich deadlines; (niech to będzie powyższy porządek);
  2. sprawdź, czy $\forall i \in 1, \ldots, n : \; f_i \leq d_i$;
     - ponieważ $f_i = \sum_{k=1}^{i} C_k$ warunek wystarczający:

$$\forall i \in 1, \ldots, n : \; \sum_{k=1}^{i} C_k \leq d_i$$

- EDD jest optymalny, zatem jeśli zestawu zadań nie można rozplanować przez EDD, to nie można go rozplanować w ogólności.

Zadania aperiodyczne, asynchroniczne

Zadania aperiodyczne, asynchroniczne

- Dany jest zestaw n zadań: $J_1, \ldots, J_n$, gdzie:
  - a_i dla wszystkich zadań jest różne i nieznane;
  - każde zadanie ma określony deadline d_i i czas obliczenia C_i;
  - zadania są niezależne (nie ma zależności wykonania).
- system z wywłaszczeniem;
- jednoprocesorowy
- zadanie: znajdź plan minimalizujący maksymalne opóźnienie.

Planista EDF – Earliest Deadline First

EDF

W każdym momencie wykonuj zadanie z najwcześniejszym czasem wykonania spośród dostępnych zadań w systemie.

Planista EDF – Earliest Deadline First

EDF

W każdym momencie wykonuj zadanie z najwcześniejszym czasem wykonania spośród dostępnych zadań w systemie.

- Jeśli w systemie pojawia się zadanie z czasem wykonania wcześniejszym niż aktualnie wykonywane zadanie - wykonywane zadanie jest wywłaszczone na rzecz nowo przyszłego zadania.

Planista EDF – Earliest Deadline First

EDF

W każdym momencie wykonuj zadanie z najwcześniejszym czasem wykonania spośród dostępnych zadań w systemie.

- Jeśli w systemie pojawia się zadanie z czasem wykonania wcześniejszym niż aktualnie wykonywane zadanie - wykonywane zadanie jest wywłaszczone na rzecz nowo przyszłego zadania.
- Założenie o znikomości czasu na przełączanie zadań!

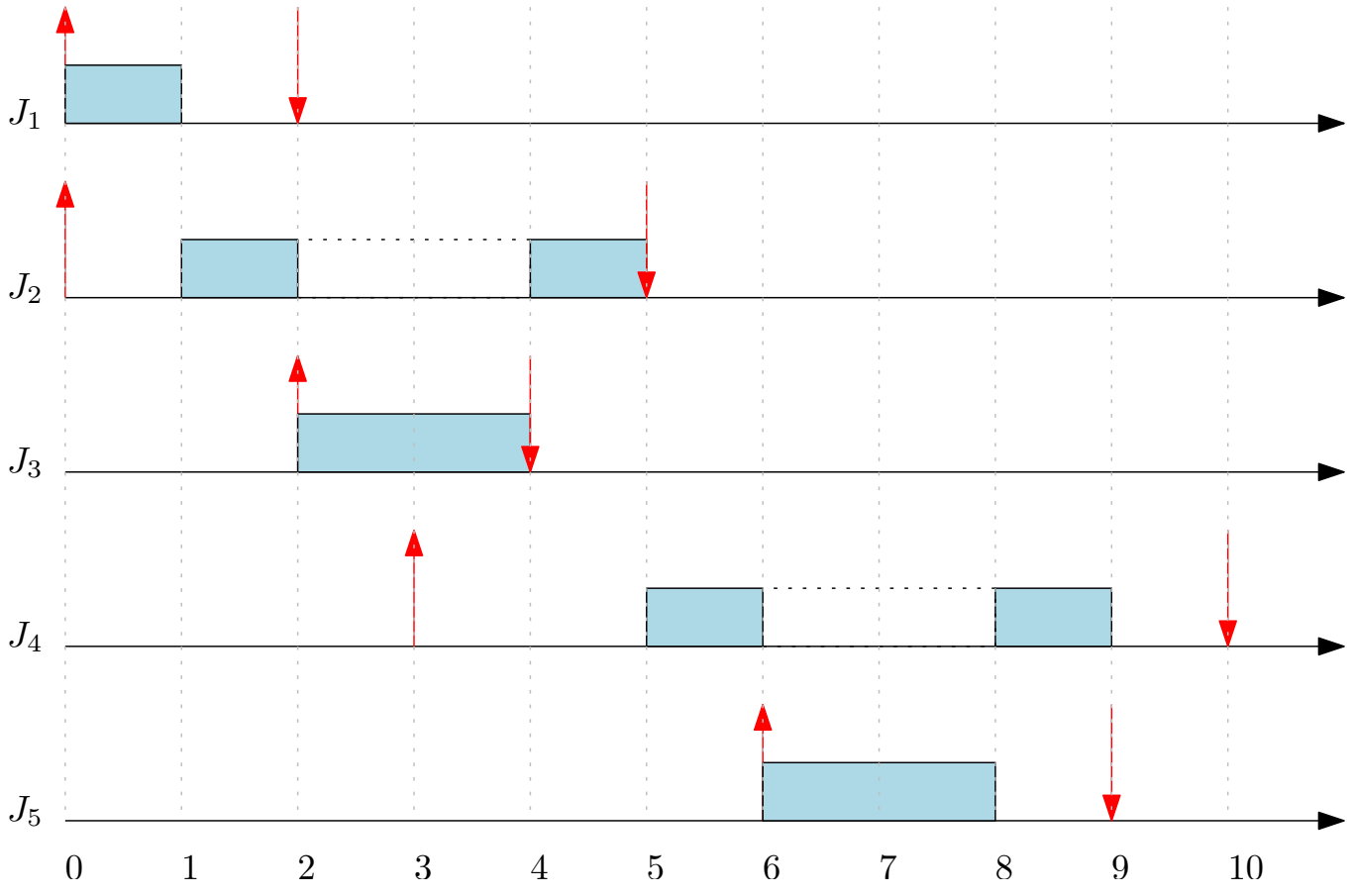
EDF - przykład

EDF - przykład

	J_1	J_2	J_3	J_4	J_5
a_i	0	0	2	3	6
C_i	1	2	2	2	2
d_i	2	5	4	10	9

EDF - przykład

	J_1	J_2	J_3	J_4	J_5
a_i	0	0	2	3	6
C_i	1	2	2	2	2
d_i	2	5	4	10	9



Optymalność EDF

Optymalność EDF

Horn 1974

Jeśli dany jest dowolny zestaw niezależnych zadań, pojawiających się w sytemie asynchronicznie, to każdy algorytm wykonujący je tak, że w każdym momencie wykonywane jest zadanie o najbliższym czasie wykonania (deadline) jest algorytmem optymalnym ze względu na minimalizację maksymalnego opóźnienia.

Optymalność EDF

Horn 1974

Jeśli dany jest dowolny zestaw niezależnych zadań, pojawiających się w sytemie asynchronicznie, to każdy algorytm wykonujący je tak, że w każdym momencie wykonywane jest zadanie o najbliższym czasie wykonania (deadline) jest algorytmem optymalnym ze względu na minimalizację maksymalnego opóźnienia.

– bez dowodu (mechanizm zbliżony do dowodu tw. Jacksona) –

Zadania aperiodyczne, asynchroniczne, wersja bez wywłaszczenia

Zadania aperiodyczne, asynchroniczne, wersja bez wywłaszczenia

- Dany jest zestaw n zadań: $J_1, \ldots, J_n$, gdzie:
  - a_i dla wszystkich zadań jest różne i nieznane;
  - każde zadanie ma określony deadline d_i i czas obliczenia C_i;
  - zadania są niezależne (nie ma zależności wykonania).
- system bez wywłaszczeń;
- jednoprocesorowy
- zadanie: znajdź plan minimalizujący maksymalne opóźnienie.

EDF bez wywłaszczeń - przykład

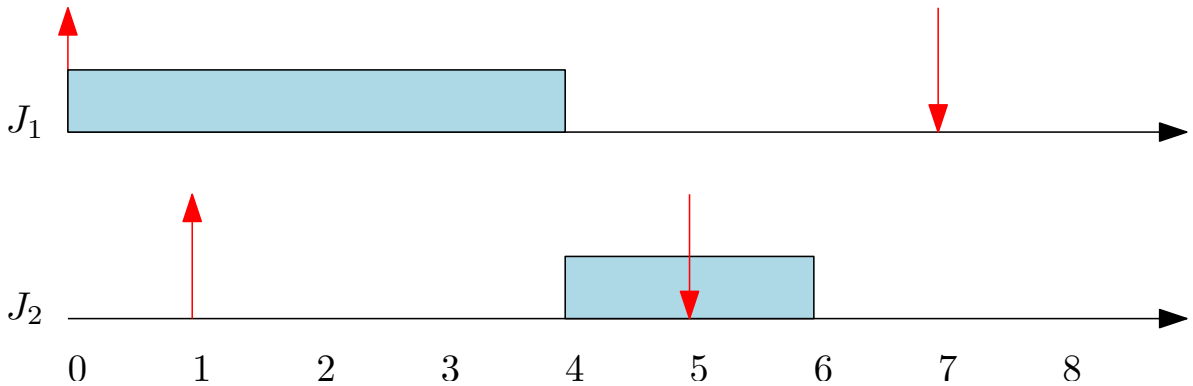
EDF bez wywłaszczeń - przykład

	J_1	J_2
a_i	0	1
C_i	4	2
d_i	7	5

EDF bez wywłaszczeń - przykład

	J_1	J_2
a_i	0	1
C_i	4	2
d_i	7	5

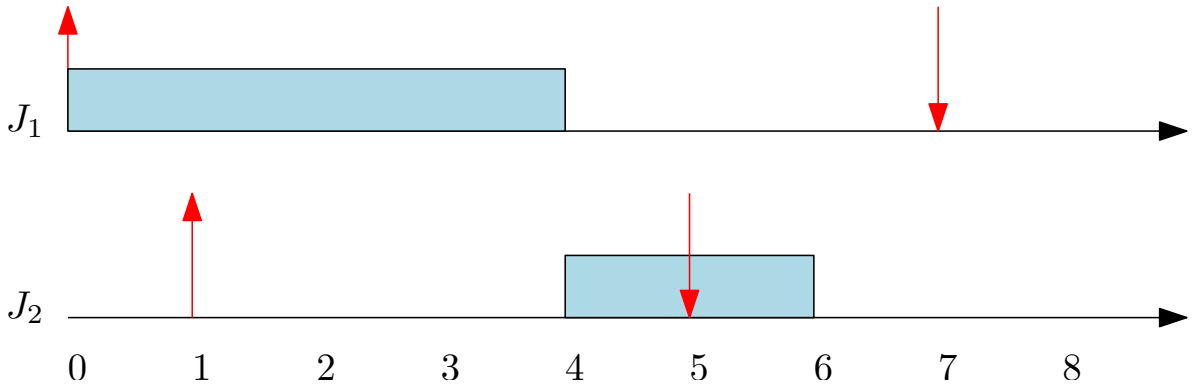
- EDF bez wywłaszczeń:



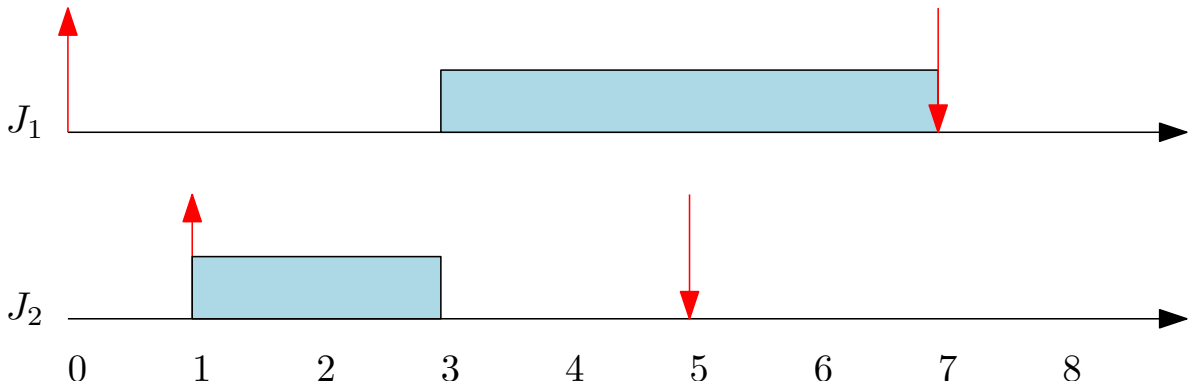
EDF bez wywłaszczeń - przykład

	J_1	J_2
a_i	0	1
C_i	4	2
d_i	7	5

- EDF bez wywłaszczeń:



- optymalnie:



EDF bez wywłaszczeń - własności

- W poprzednim przykładzie CPU jest bezczynne w okresie 0-1, mimo gotowości zadania J_1.

EDF bez wywłaszczeń - własności

- W poprzednim przykładzie CPU jest bezczynne w okresie 0-1, mimo gotowości zadania J_1.
- Jeśli czasy a_i nie są znane, to niemożliwe jest podjęcie decyzji o tej bezczynności.

EDF bez wywłaszczeń - własności

- W poprzednim przykładzie CPU jest bezczynne w okresie 0-1, mimo gotowości zadania J_1.
- Jeśli czasy a_i nie są znane, to niemożliwe jest podjęcie decyzji o tej bezczynności.

Jeffay i in. 1991

Przy braku zezwolenia na okresy bezczynności przy gotowym do wykonania zadaniu, EDF jest optymalny również dla modelu bez wywłaszczeń.

Zadania aperiodyczne, asynchroniczne, wersja bez wywłaszczenia, dozwolone czasy bezczynności

Zadania aperiodyczne, asynchroniczne, wersja bez wywłaszczenia, dozwolone czasy bezczynności

- Dany jest zestaw n zadań: $J_1, \ldots, J_n$, gdzie:
  - a_i dla wszystkich zadań znane;
  - każde zadanie ma określony deadline d_i i czas obliczenia C_i;
  - zadania są niezależne (nie ma zależności wykonania).
- system bez wywłaszczeń;
- system zezwala na czasy bezczynności przy gotowym zadaniu;
- jednoprocesorowy
- zadanie: znajdź plan minimalizujący maksymalne opóźnienie.

Zadania aperiodyczne, asynchroniczne, wersja bez wywłaszczenia, dozwolone czasy bezczynności

- Dany jest zestaw n zadań: $J_1, \ldots, J_n$, gdzie:
  - a_i dla wszystkich zadań znane;
  - każde zadanie ma określony deadline d_i i czas obliczenia C_i;
  - zadania są niezależne (nie ma zależności wykonania).
- system bez wywłaszczeń;
- system zezwala na czasy bezczynności przy gotowym zadaniu;
- jednoprocesorowy
- zadanie: znajdź plan minimalizujący maksymalne opóźnienie.

W ogólności, problem jest **NP-trudny**.

Zadania aperiodyczne, asynchroniczne, wersja bez wywłaszczenia, dozwolone czasy bezczynności

- Dany jest zestaw n zadań: $J_1, \ldots, J_n$, gdzie:
  - a_i dla wszystkich zadań znane;
  - każde zadanie ma określony deadline d_i i czas obliczenia C_i;
  - zadania są niezależne (nie ma zależności wykonania).
- system bez wywłaszczeń;
- system zezwala na czasy bezczynności przy gotowym zadaniu;
- jednoprocesorowy
- zadanie: znajdź plan minimalizujący maksymalne opóźnienie.

W ogólności, problem jest **NP-trudny**. Rozwiązywalny przy użyciu heurystyk lub mechanizmów typu *branch-and-bound*.

Algorytm Bratley'a

- Wykorzystując mechanizm branch-and-bound odnajduje realizowalne rozwiązanie, jeśli istnieje.
- Bazuje na odpowiedniej permutacji zestawu zadań $J_1, \ldots, J_n$.
- Rozpoczyna od pustej listy (kolejności) zadań;
- **Branch:** wybierz do listy kolejne zadanie (z pozostałych)
- **Bound:**

Algorytm Bratley'a

- Wykorzystując mechanizm branch-and-bound odnajduje realizowalne rozwiązanie, jeśli istnieje.
- Bazuje na odpowiedniej permutacji zestawu zadań $J_1, \ldots, J_n$.
- Rozpoczyna od pustej listy (kolejności) zadań;
- **Branch:** wybierz do listy kolejne zadanie (z pozostałych)
- **Bound:**
  1. znaleziono realizowalny plan $\rightarrow$ bieżąca lista realizowalnym rozwiązaniem;

Algorytm Bratley'a

- Wykorzystując mechanizm branch-and-bound odnajduje realizowalne rozwiązanie, jeśli istnieje.
- Bazuje na odpowiedniej permutacji zestawu zadań $J_1, \ldots, J_n$.
- Rozpoczyna od pustej listy (kolejności) zadań;
- **Branch:** wybierz do listy kolejne zadanie (z pozostałych)
- **Bound:**
  1. znaleziono realizowalny plan $\rightarrow$ bieżąca lista realizowalnym rozwiązaniem;
  2. jeśli istnieje zadanie, którego *deadline* minął, a nie zostało jeszcze zaplanowane $\rightarrow$ bieżąca ścieżka to nierealizowalny plan (cofnij do ostatniego dokonanego wyboru).

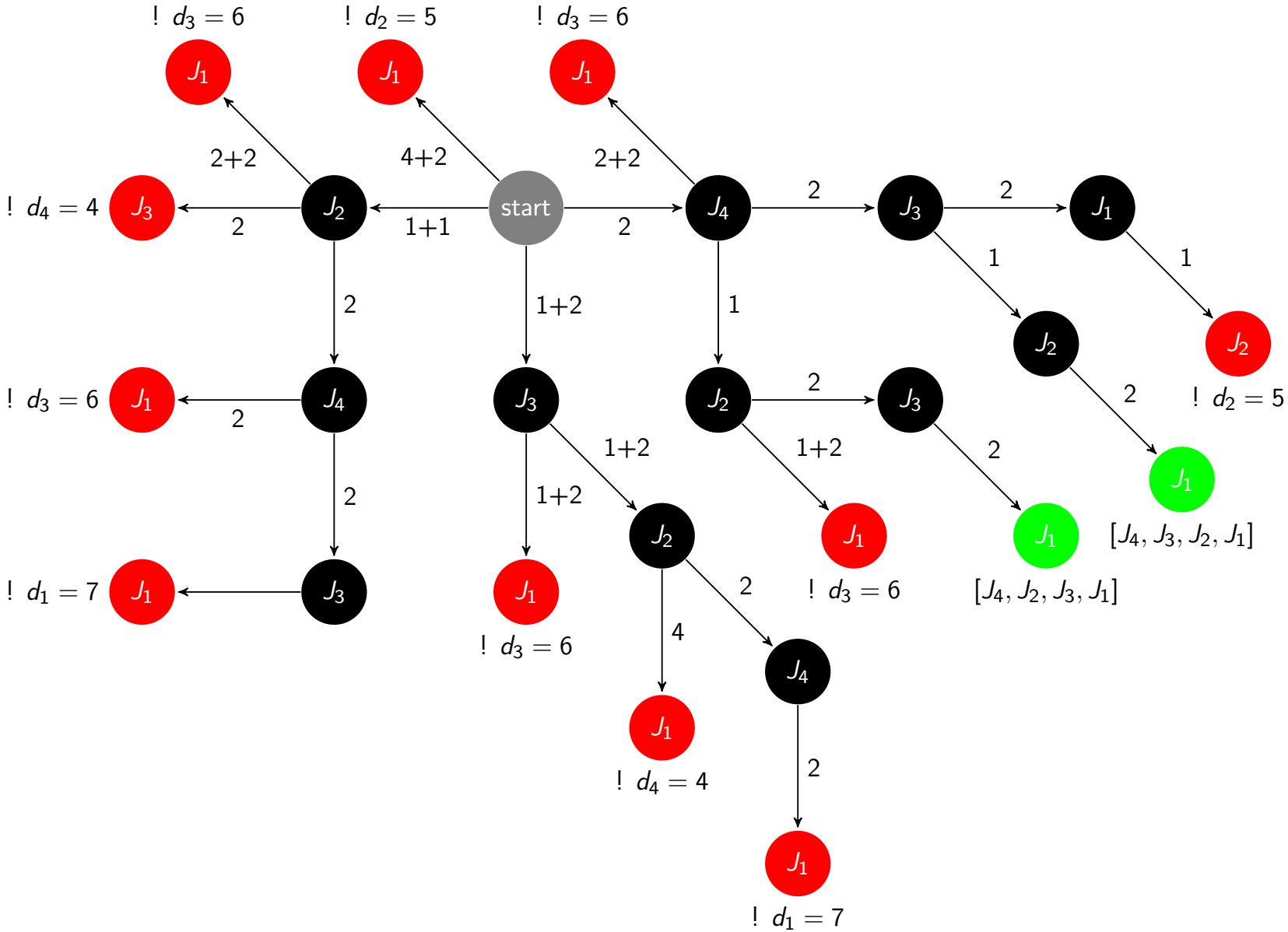
Algorytm Bratley'a

- Wykorzystując mechanizm branch-and-bound odnajduje realizowalne rozwiązanie, jeśli istnieje.
- Bazuje na odpowiedniej permutacji zestawu zadań $J_1, \ldots, J_n$.
- Rozpoczyna od pustej listy (kolejności) zadań;
- **Branch:** wybierz do listy kolejne zadanie (z pozostałych)
- **Bound:**
  1. znaleziono realizowalny plan $\rightarrow$ bieżąca lista realizowalnym rozwiązaniem;
  2. jeśli istnieje zadanie, którego *deadline* minął, a nie zostało jeszcze zaplanowane $\rightarrow$ bieżąca ścieżka to nierealizowalny plan (cofnij do ostatniego dokonanego wyboru).

Algorytm Bratley'a

- Wykorzystując mechanizm branch-and-bound odnajduje realizowalne rozwiązanie, jeśli istnieje.
- Bazuje na odpowiedniej permutacji zestawu zadań $J_1, \ldots, J_n$.
- Rozpoczyna od pustej listy (kolejności) zadań;
- **Branch:** wybierz do listy kolejne zadanie (z pozostałych)
- **Bound:**
  1. znaleziono realizowalny plan $\rightarrow$ bieżąca lista realizowalnym rozwiązaniem;
  2. jeśli istnieje zadanie, którego *deadline* minął, a nie zostało jeszcze zaplanowane $\rightarrow$ bieżąca ścieżka to nierealizowalny plan (cofnij do ostatniego dokonanego wyboru).

	J_1	J_2	J_3	J_4
a_i	4	1	1	0
C_i	2	1	2	2
d_i	7	5	6	4



J_1: a=4, C=2, d=7; J_2: a=1, C=1, d=5;
J_3: a=1, C=2, d=6; J_4: a=0, C=2, d=4

Algorytm Bratley'a - własności

- Złożoność wykładnicza – tylko jako algorytm offline;

Algorytm Bratley'a - własności

- Złożoność wykładnicza – tylko jako algorytm offline;
- Odkryty schemat wykonania zapisany jako lista i odczytywany w kolejności przez planistę w trakcie działania systemu.

Algorytm Bratley'a - własności

- Złożoność wykładnicza – tylko jako algorytm offline;
- Odkryty schemat wykonania zapisany jako lista i odczytywany w kolejności przez planistę w trakcie działania systemu.
- Realizacja – dobry przykład zastosowania struktury drzewa w programowaniu :-)

Planowanie zadań z wymaganą kolejnością wykonania

- Planowanie bez wywłaszczenia, z asynchronicznymi pojawieniami się zadań, czasami wykonania i wymaganą kolejnością jest **NP-trudne**.

Planowanie zadań z wymaganą kolejnością wykonania

- Planowanie bez wywłaszczenia, z asynchronicznymi pojawieniami się zadań, czasami wykonania i wymaganą kolejnością jest **NP-trudne**.
- Dla pewnych rozluźnień problemu można znaleźć optymalne rozwiązania...

Planowanie zadań z wymaganą kolejnością wykonania

- Planowanie bez wywłaszczenia, z asynchronicznymi pojawieniami się zadań, czasami wykonania i wymaganą kolejnością jest **NP-trudne**.
- Dla pewnych rozluźnień problemu można znaleźć optymalne rozwiązania...
- Np. dla przykładu gdzie wszystkie zadania są dostępne od razu.

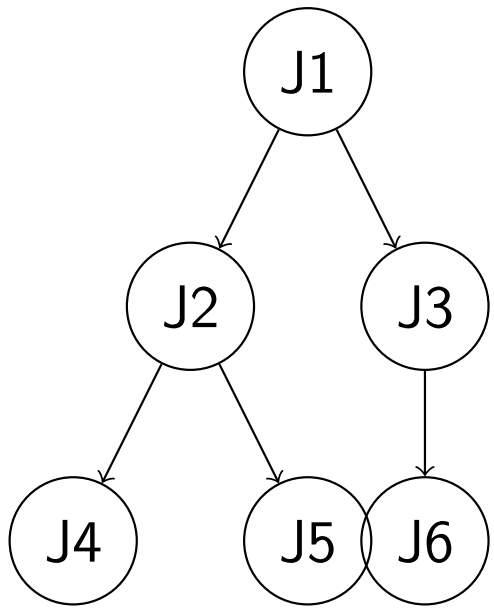
Planowanie zadań z wymaganą kolejnością wykonania

- Planowanie bez wywłaszczenia, z asynchronicznymi pojawieniami się zadań, czasami wykonania i wymaganą kolejnością jest **NP-trudne**.
- Dla pewnych rozluźnień problemu można znaleźć optymalne rozwiązania...
- Np. dla przykładu gdzie wszystkie zadania są dostępne od razu.

Planowanie zadań z wymaganą kolejnością wykonania

- Planowanie bez wywłaszczenia, z asynchronicznymi pojawieniami się zadań, czasami wykonania i wymaganą kolejnością jest **NP-trudne**.
- Dla pewnych rozluźnień problemu można znaleźć optymalne rozwiązania...
- Np. dla przykładu gdzie wszystkie zadania są dostępne od razu.

	J_1	J_2	J_3	J_4	J_5	J_6
a_i	0	0	0	0	0	0
C_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6



Planowanie zadań z wymaganą kolejnością wykonania

- Planowanie bez wywłaszczenia, z asynchronicznymi pojawieniami się zadań, czasami wykonania i wymaganą kolejnością jest **NP-trudne**.

Planowanie zadań z wymaganą kolejnością wykonania

- Planowanie bez wywłaszczenia, z asynchronicznymi pojawieniami się zadań, czasami wykonania i wymaganą kolejnością jest **NP-trudne**.
- Dla pewnych rozluźnień problemu można znaleźć optymalne rozwiązania...

Planowanie zadań z wymaganą kolejnością wykonania

- Planowanie bez wywłaszczenia, z asynchronicznymi pojawieniami się zadań, czasami wykonania i wymaganą kolejnością jest **NP-trudne**.
- Dla pewnych rozluźnień problemu można znaleźć optymalne rozwiązania...
- gdy wszystkie zadania są dostępne od razu – LDF

LDF – latest deadline first

1. utrzymuj zadania do planowania w grafie G, do wykonania na liście P;

LDF – latest deadline first

1. utrzymuj zadania do planowania w grafie G, do wykonania na liście P;
2. spośród wszystkich liści G, wybierz zadanie J o najpóźniejszym terminie wykonania (deadline);

LDF – latest deadline first

1. utrzymuj zadania do planowania w grafie G, do wykonania na liście P;
2. spośród wszystkich liści G, wybierz zadanie J o najpóźniejszym terminie wykonania (deadline);
3. wstaw J na początek P i usuń je z G

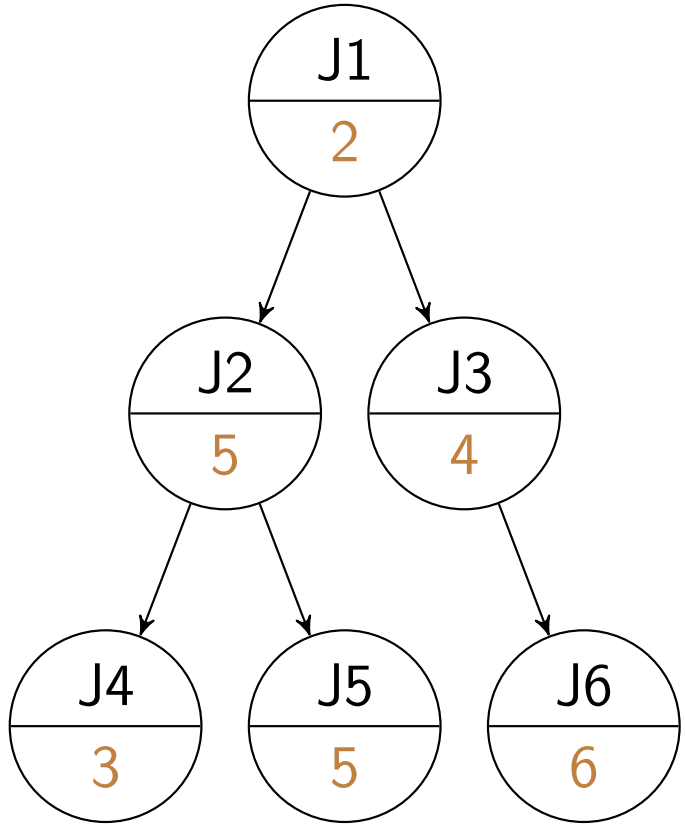
LDF – latest deadline first

1. utrzymuj zadania do planowania w grafie G, do wykonania na liście P;
2. spośród wszystkich liści G, wybierz zadanie J o najpóźniejszym terminie wykonania (deadline);
3. wstaw J na początek P i usuń je z G
4. wykonuj ponownie od 2.

LDF – przykład

	J_1	J_2	J_3	J_4	J_5	J_6
a_i	0	0	0	0	0	0
C_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6

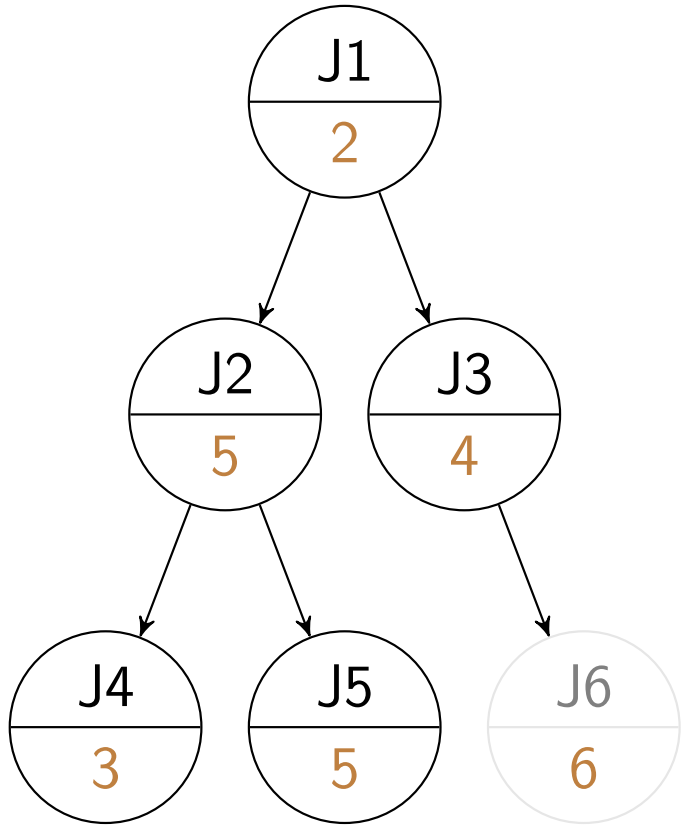
$$P = ()$$



LDF – przykład

	J_1	J_2	J_3	J_4	J_5	J_6
a_i	0	0	0	0	0	0
C_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6

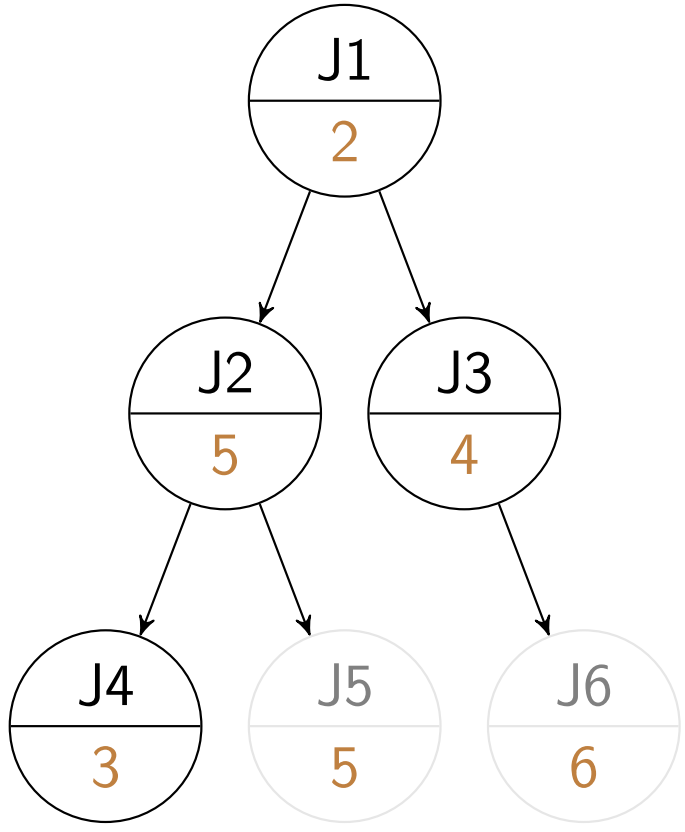
$$P = (J_6)$$



LDF – przykład

	J_1	J_2	J_3	J_4	J_5	J_6
a_i	0	0	0	0	0	0
C_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6

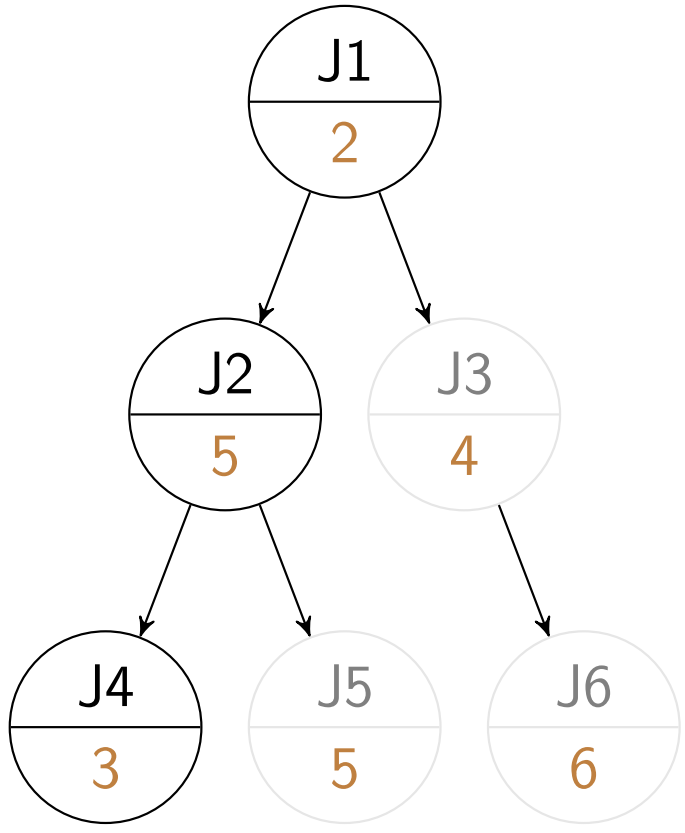
$$P = (J_5, J_6)$$



LDF – przykład

	J_1	J_2	J_3	J_4	J_5	J_6
a_i	0	0	0	0	0	0
C_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6

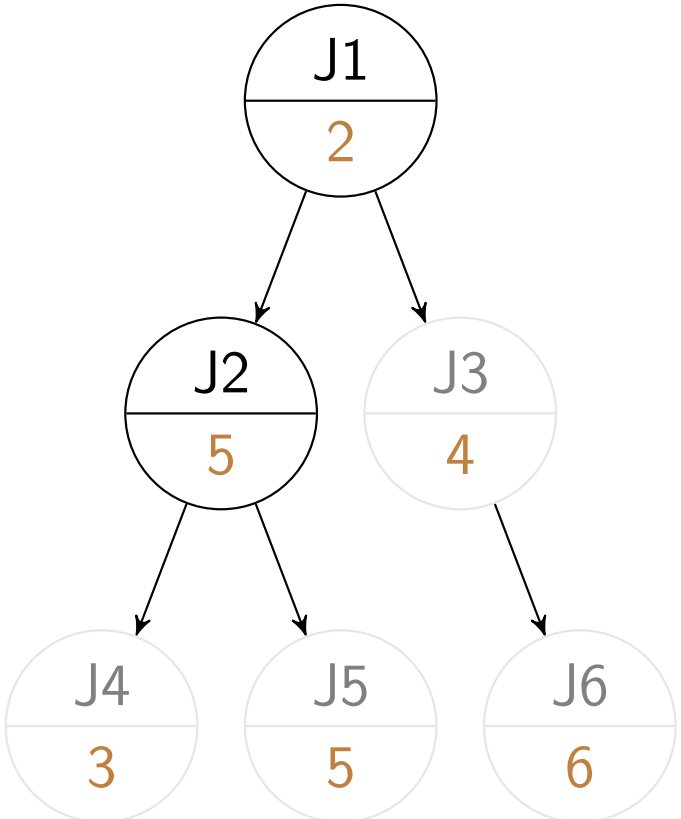
$$P = (J_3, J_5, J_6)$$



LDF – przykład

	J_1	J_2	J_3	J_4	J_5	J_6
a_i	0	0	0	0	0	0
C_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6

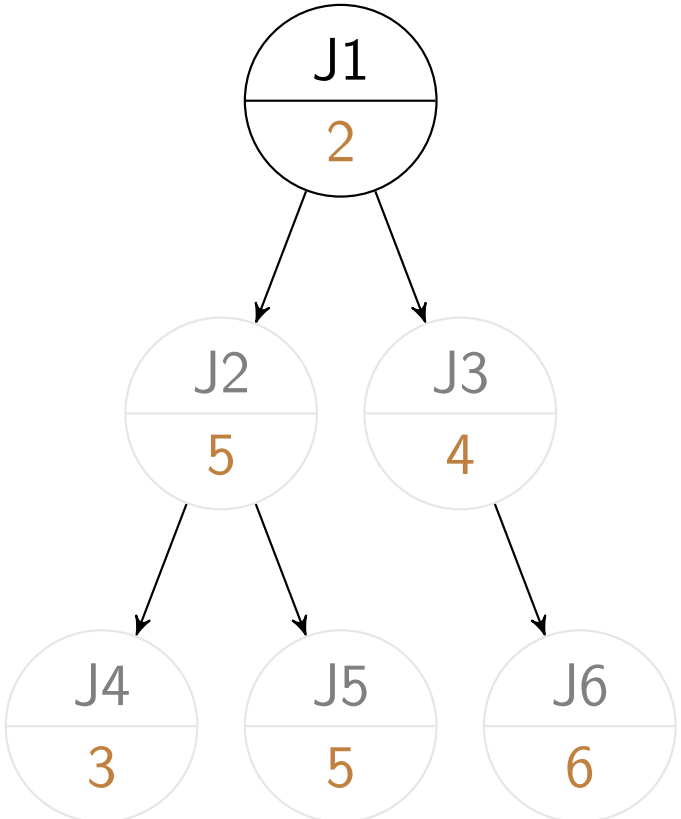
$$P = (J_4, J_3, J_5, J_6)$$



LDF – przykład

	J_1	J_2	J_3	J_4	J_5	J_6
a_i	0	0	0	0	0	0
C_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6

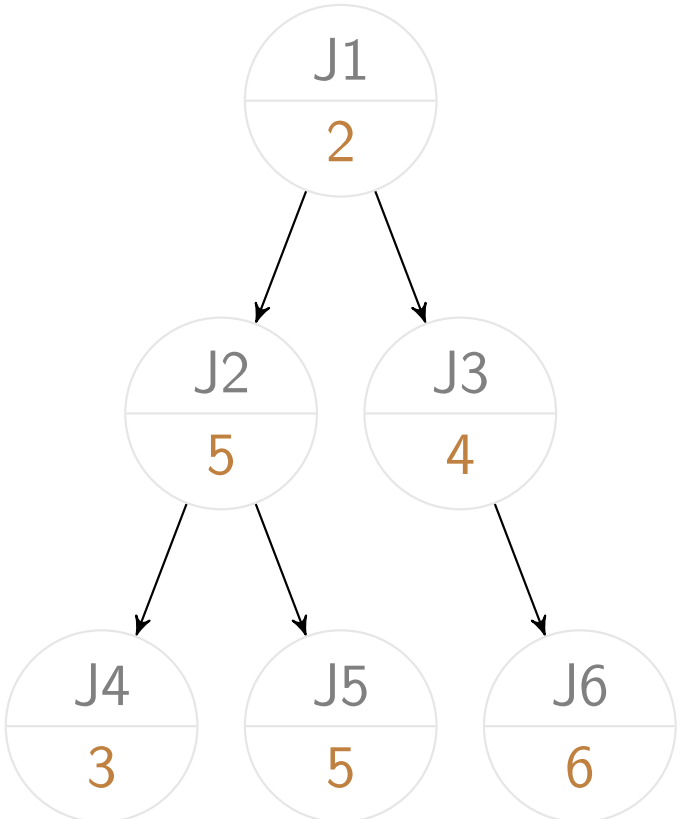
$$P = (J_2, J_4, J_3, J_5, J_6)$$



LDF – przykład

	J_1	J_2	J_3	J_4	J_5	J_6
a_i	0	0	0	0	0	0
C_i	1	1	1	1	1	1
d_i	2	5	4	3	5	6

$$P = (J_1, J_2, J_4, J_3, J_5, J_6)$$



LDF – optymalność

LDF – optymalność

Lawler '73

LDF jest optymalny ze względu na minimalizację maksymalnego opóźnienia.

LDF – optymalność

Lawler '73

LDF jest optymalny ze względu na minimalizację maksymalnego opóźnienia.

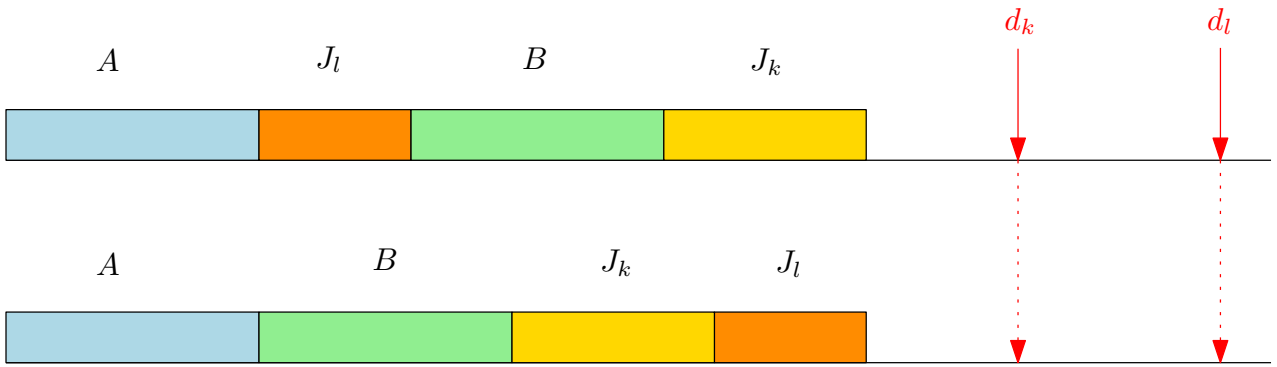
- dane są zadania $(J_1, \ldots, J_n)$
- niech J_l nie ma zadań "poniżej" siebie w drzewie zależności
- niech J_l ma najpóźniejszy termin wykonania

LDF – optymalność

Lawler '73

LDF jest optymalny ze względu na minimalizację maksymalnego opóźnienia.

- dane są zadania $(J_1, \ldots, J_n)$
- niech J_l nie ma zadań "poniżej" siebie w drzewie zależności
- niech J_l ma najpóźniejszy termin wykonania

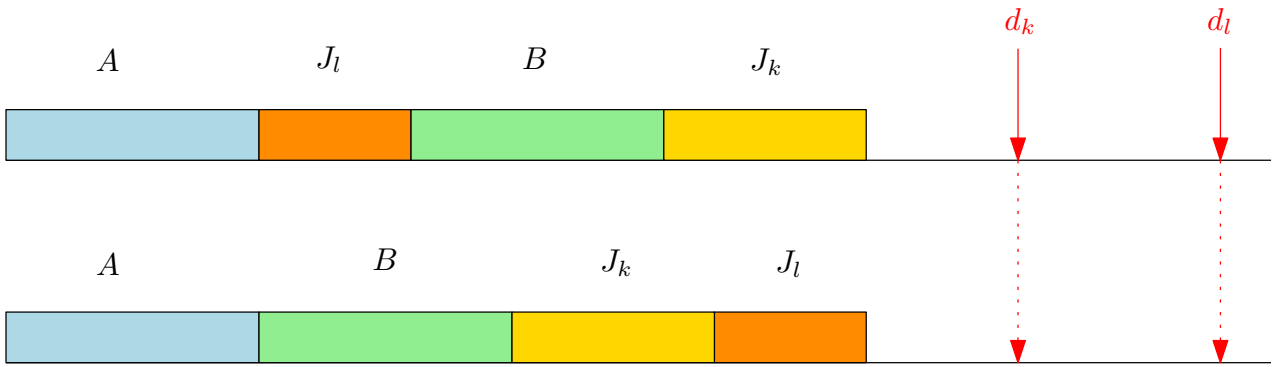


LDF – optymalność

Lawler '73

LDF jest optymalny ze względu na minimalizację maksymalnego opóźnienia.

- dane są zadania $(J_1, \ldots, J_n)$
- niech J_l nie ma zadań "poniżej" siebie w drzewie zależności
- niech J_l ma najpóźniejszy termin wykonania



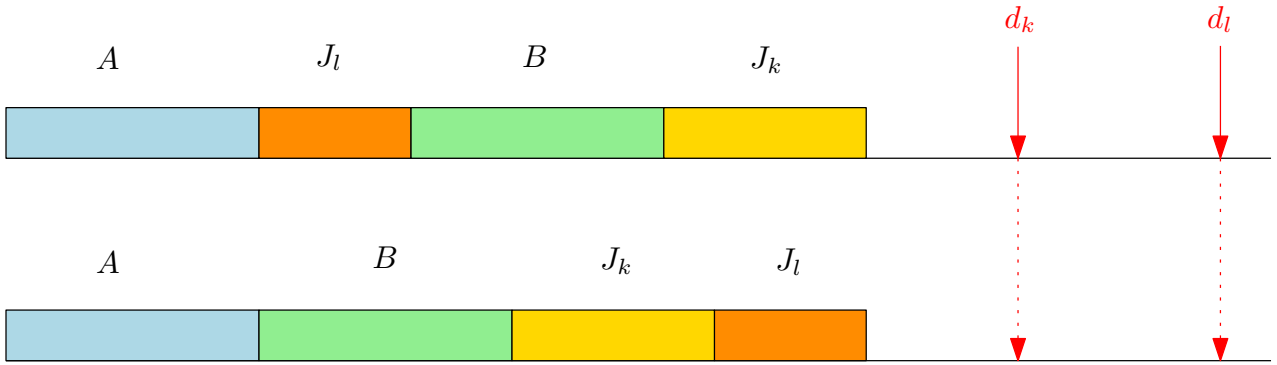
1. przesuń J_l na koniec kolejki wykonania (bo $d_l > d_k$);

LDF – optymalność

Lawler '73

LDF jest optymalny ze względu na minimalizację maksymalnego opóźnienia.

- dane są zadania $(J_1, \ldots, J_n)$
- niech J_l nie ma zadań "poniżej" siebie w drzewie zależności
- niech J_l ma najpóźniejszy termin wykonania



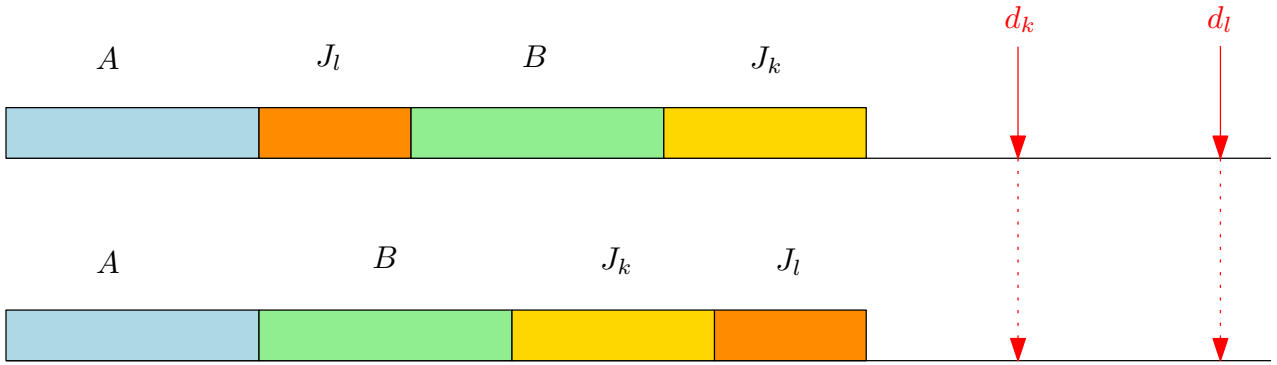
1. przesuń J_l na koniec kolejki wykonania (bo $d_l > d_k$);
2. pokaż, że kolejność wykonania jest zachowana

LDF – optymalność

Lawler '73

LDF jest optymalny ze względu na minimalizację maksymalnego opóźnienia.

- dane są zadania $(J_1, \ldots, J_n)$
- niech J_l nie ma zadań "poniżej" siebie w drzewie zależności
- niech J_l ma najpóźniejszy termin wykonania



1. przesuń J_l na koniec kolejki wykonania (bo $d_l > d_k$);
2. pokaż, że kolejność wykonania jest zachowana
3. pokaż, że maksymalne opóźnienie nie zwiększa się

LDF – właściwości

- działa jedynie jako algorytm off-line

LDF – właściwości

- działa jedynie jako algorytm off-line
- złożoność:

LDF – właściwości

- działa jedynie jako algorytm off-line
- złożoność:
  - szukanie liści w grafie G – $\mathcal{O}(|E|)$

LDF – właściwości

- działa jedynie jako algorytm off-line
- złożoność:
  - szukanie liści w grafie G – $\mathcal{O}(|E|)$
  - utrzymywanie posortowanej (wg. d_i) listy liści – $\mathcal{O}(\log n)$

LDF – właściwości

- działa jedynie jako algorytm off-line
- złożoność:
  - szukanie liści w grafie G – $\mathcal{O}(|E|)$
  - utrzymywanie posortowanej (wg. d_i) listy liści – $\mathcal{O}(\log n)$
  - ogółem: $\mathcal{O}(n * \max(|E|, \log n))$

Planowanie zadań z wymaganą kolejnością wykonania

- Planowanie bez wywłaszczenia, z asynchronicznymi pojawieniami się zadań, czasami wykonania i wymaganą kolejnością jest **NP-trudne**.

Planowanie zadań z wymaganą kolejnością wykonania

- Planowanie bez wywłaszczenia, z asynchronicznymi pojawieniami się zadań, czasami wykonania i wymaganą kolejnością jest **NP-trudne**.
- Dla pewnych rozluźnień problemu można znaleźć optymalne rozwiązania...

Planowanie zadań z wymaganą kolejnością wykonania

- Planowanie bez wywłaszczenia, z asynchronicznymi pojawieniami się zadań, czasami wykonania i wymaganą kolejnością jest **NP-trudne**.
- Dla pewnych rozluźnień problemu można znaleźć optymalne rozwiązania...
  - wszystkie zadania są dostępne od razu – LDF

Planowanie zadań z wymaganą kolejnością wykonania

- Planowanie bez wywłaszczenia, z asynchronicznymi pojawieniami się zadań, czasami wykonania i wymaganą kolejnością jest **NP-trudne**.
- Dla pewnych rozluźnień problemu można znaleźć optymalne rozwiązania...
  - wszystkie zadania są dostępne od razu – LDF
  - dla schematów wywłaszczających – modyfikowany EDF

Zmodyfikowany EDF – przykład

Aby zaadaptować przypadek, gdy zadania pojawiają się w systemie asynchronicznie ($\exists i, j : a_i \neq a_j$):

Zmodyfikowany EDF – przykład

Aby zaadaptować przypadek, gdy zadania pojawiają się w systemie asynchronicznie ($\exists i, j : a_i \neq a_j$):

1. zaktualizuj czasy nadejścia a_i:

Zmodyfikowany EDF – przykład

Aby zaadaptować przypadek, gdy zadania pojawiają się w systemie asynchronicznie ($\exists i, j : a_i \neq a_j$):

1. zaktualizuj czasy nadejścia a_i:
   - każde zadanie w "korzeniu": $a_i^* = a_i$

Zmodyfikowany EDF – przykład

Aby zaadaptować przypadek, gdy zadania pojawiają się w systemie asynchronicznie ($\exists i, j : a_i \neq a_j$):

1. zaktualizuj czasy nadejścia a_i:
   - każde zadanie w "korzeniu": $a_i^* = a_i$
   - dla każdego zadania, którego czas nadejścia "rodzica" J_r został zaktualizowany: $a_i^* = \max\{a_i, a_r^* + C_r\}$

Zmodyfikowany EDF – przykład

Aby zaadaptować przypadek, gdy zadania pojawiają się w systemie asynchronicznie ($\exists i, j : a_i \neq a_j$):

1. zaktualizuj czasy nadejścia a_i:
   - każde zadanie w "korzeniu": $a_i^* = a_i$
   - dla każdego zadania, którego czas nadejścia "rodzica" J_r został zaktualizowany: $a_i^* = \max\{a_i, a_r^* + C_r\}$
2. zaktualizuj terminy wykonania:

Zmodyfikowany EDF – przykład

Aby zaadaptować przypadek, gdy zadania pojawiają się w systemie asynchronicznie ($\exists i, j : a_i \neq a_j$):

1. zaktualizuj czasy nadejścia a_i:
   - każde zadanie w "korzeniu": $a_i^* = a_i$
   - dla każdego zadania, którego czas nadejścia "rodzica" J_r został zaktualizowany: $a_i^* = \max\{a_i, a_r^* + C_r\}$
2. zaktualizuj terminy wykonania:
   - dla każdego zadania w "liściu": $d_i^* = d_i$

Zmodyfikowany EDF – przykład

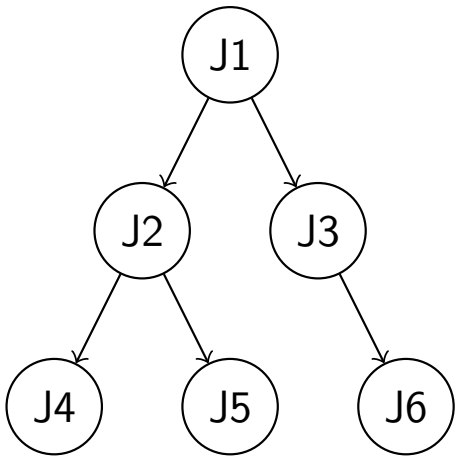
Aby zaadaptować przypadek, gdy zadania pojawiają się w systemie asynchronicznie ($\exists i, j : a_i \neq a_j$):

1. zaktualizuj czasy nadejścia a_i:
   - każde zadanie w "korzeniu": $a_i^* = a_i$
   - dla każdego zadania, którego czas nadejścia "rodzica" J_r został zaktualizowany: $a_i^* = \max\{a_i, a_r^* + C_r\}$
2. zaktualizuj terminy wykonania:
   - dla każdego zadania w "liściu": $d_i^* = d_i$
   - dla każdego zadania J_i, którego "potomkowie" J_p zostali zaktualizowani: $d_i^* = \min\{d_i, d_p^* - C_p\}$

Zmodyfikowany EDF – przykład

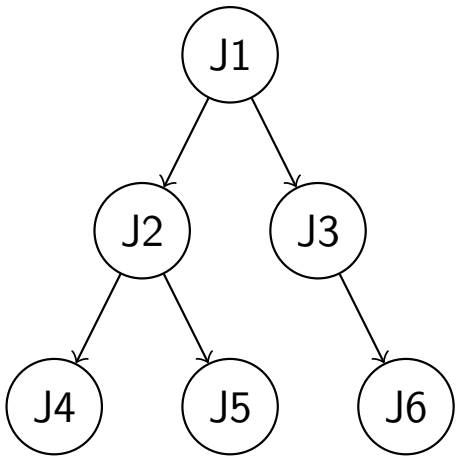
Zmodyfikowany EDF – przykład

	J_1	J_2	J_3	J_4	J_5	J_6
a_i a_i^*	1	2	3	3	3	4
C_i	1	2	1	1	1	3
d_i d_i^*	2	5	4	6	5	7



Zmodyfikowany EDF – przykład

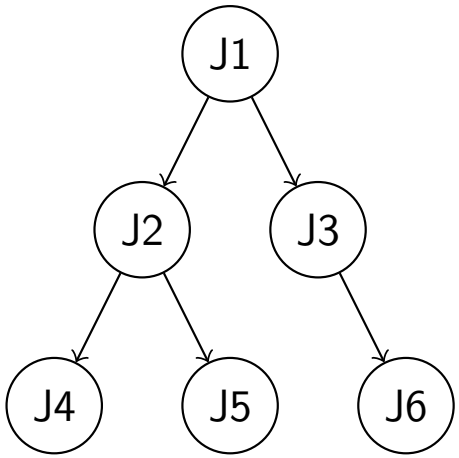
	J_1	J_2	J_3	J_4	J_5	J_6
a_i	1	2	3	3	3	4
a_i^*	1	2	3	4	4	4
C_i	1	2	1	1	1	3
d_i	2	5	4	6	5	7
d_i^*						



- $a_1^* = a_1 = 1$
- $a_2^* = \max\{a_2, a_1^* + C_1\} = 2$
- $a_3^* = \max\{a_3, a_1^* + C_1\} = 3$
- $a_4^* = \max\{a_4, a_2^* + C_2\} = 4$
- $a_5^* = \max\{a_5, a_2^* + C_2\} = 4$
- $a_6^* = \max\{a_6, a_3^* + C_3\} = 4$

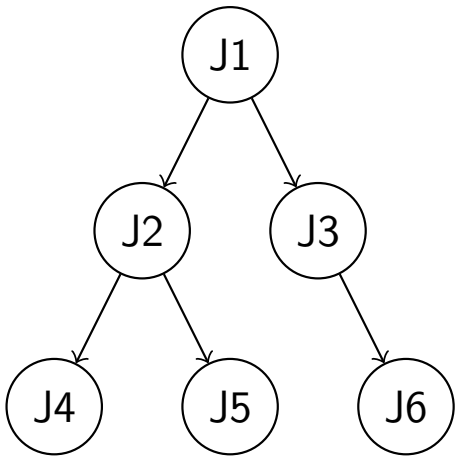
Zmodyfikowany EDF – przykład

	J_1	J_2	J_3	J_4	J_5	J_6
a_i	1	2	3	3	3	4
a_i^*	1	2	3	4	4	4
C_i	1	2	1	1	1	3
d_i	2	5	4	6	5	7
d_i^*						



Zmodyfikowany EDF – przykład

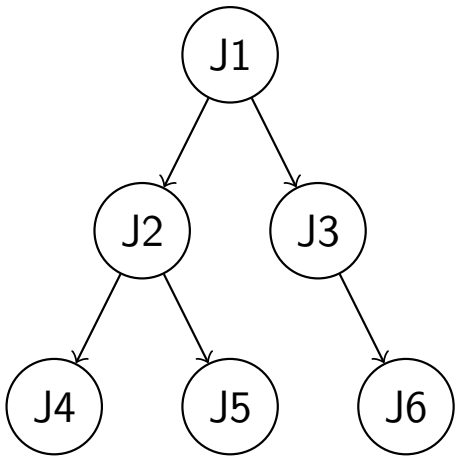
	J_1	J_2	J_3	J_4	J_5	J_6
a_i	1	2	3	3	3	4
a_i^*	1	2	3	4	4	4
C_i	1	2	1	1	1	3
d_i	2	5	4	6	5	7
d_i^*	0	4	4	6	5	7



- $d_6^* = d_6 = 7$
- $d_5^* = d_5 = 5$
- $d_4^* = d_4 = 6$
- $d_3^* = \min\{d_3, d_6^* - C_6\} = 4$
- $d_2^* = \min\{d_2, d_5^* - C_5, d_4^* - C_4\} = 4$
- $d_1^* = \min\{d_1, d_2^* - C_2, d_3^* - C_3\} = 0$

Zmodyfikowany EDF – przykład

	J_1	J_2	J_3	J_4	J_5	J_6
a_i	1	2	3	3	3	4
a_i^*	1	2	3	4	4	4
C_i	1	2	1	1	1	3
d_i	2	5	4	6	5	7
d_i^*	0	4	4	6	5	7



Jak zmodyfikować zadania, by plan był wykonalny?

Zmodyfikowany EDF – przechodniość planowalności

Przechodniość planowalności

Zbiór zadań jest planowalny z zachowaniem wymogów następstwa wykonania zadań wtedy i tylko wtedy, gdy zbiór zadań o zmodyfikowanych parametrach (a_i^*, d_i^*) jest planowalny przez EDF.

Zmodyfikowany EDF – przechodniość planowalności

Przechodniość planowalności

Zbiór zadań jest planowalny z zachowaniem wymogów następstwa wykonania zadań wtedy i tylko wtedy, gdy zbiór zadań o zmodyfikowanych parametrach (a_i^*, d_i^*) jest planowalny przez EDF.

1. Dla każdego planu σ zachowującego zależności musi być:

Zmodyfikowany EDF – przechodniość planowalności

Przechodniość planowalności

Zbiór zadań jest planowalny z zachowaniem wymogów następstwa wykonania zadań wtedy i tylko wtedy, gdy zbiór zadań o zmodyfikowanych parametrach (a_i^*, d_i^*) jest planowalny przez EDF.

1. Dla każdego planu σ zachowującego zależności musi być:
   - $s_i \geq \max\{a_i, a_r^* + C_r\}$

Zmodyfikowany EDF – przechodniość planowalności

Przechodniość planowalności

Zbiór zadań jest planowalny z zachowaniem wymogów następstwa wykonania zadań wtedy i tylko wtedy, gdy zbiór zadań o zmodyfikowanych parametrach (a_i^*, d_i^*) jest planowalny przez EDF.

1. Dla każdego planu σ zachowującego zależności musi być:
   - $s_i \geq \max\{a_i, a_r^* + C_r\}$
   - $f_i \leq \min\{d_i, d_p^* - C_p\}$

Zmodyfikowany EDF – przechodniość planowalności

Przechodniość planowalności

Zbiór zadań jest planowalny z zachowaniem wymogów następstwa wykonania zadań wtedy i tylko wtedy, gdy zbiór zadań o zmodyfikowanych parametrach (a_i^*, d_i^*) jest planowalny przez EDF.

1. Dla każdego planu σ zachowującego zależności musi być:
   - $s_i \geq \max\{a_i, a_r^* + C_r\}$
   - $f_i \leq \min\{d_i, d_p^* - C_p\}$
2. Jeśli zadania $J_i \prec J_j$ planowalne są przez EDF, to:

Zmodyfikowany EDF – przechodniość planowalności

Przechodniość planowalności

Zbiór zadań jest planowalny z zachowaniem wymogów następstwa wykonania zadań wtedy i tylko wtedy, gdy zbiór zadań o zmodyfikowanych parametrach (a_i^*, d_i^*) jest planowalny przez EDF.

1. Dla każdego planu σ zachowującego zależności musi być:
   - $s_i \geq \max\{a_i, a_r^* + C_r\}$
   - $f_i \leq \min\{d_i, d_p^* - C_p\}$
2. Jeśli zadania $J_i \prec J_j$ planowalne są przez EDF, to:
   - $a_i^* < a_j^*$: J_i pojawiło się wcześniej niż J_j;

Zmodyfikowany EDF – przechodniość planowalności

Przechodniość planowalności

Zbiór zadań jest planowalny z zachowaniem wymogów następstwa wykonania zadań wtedy i tylko wtedy, gdy zbiór zadań o zmodyfikowanych parametrach (a_i^*, d_i^*) jest planowalny przez EDF.

1. Dla każdego planu σ zachowującego zależności musi być:
   - $s_i \geq \max\{a_i, a_r^* + C_r\}$
   - $f_i \leq \min\{d_i, d_p^* - C_p\}$
2. Jeśli zadania $J_i \prec J_j$ planowalne są przez EDF, to:
   - $a_i^* < a_j^*$: J_i pojawiło się wcześniej niż J_j;
   - $d_i^* < d_j^*$: J_i ma wcześniejszy termin wykonania;

Zmodyfikowany EDF – przechodniość planowalności

Przechodniość planowalności

Zbiór zadań jest planowalny z zachowaniem wymogów następstwa wykonania zadań wtedy i tylko wtedy, gdy zbiór zadań o zmodyfikowanych parametrach (a_i^*, d_i^*) jest planowalny przez EDF.

1. Dla każdego planu σ zachowującego zależności musi być:
   - $s_i \geq \max\{a_i, a_r^* + C_r\}$
   - $f_i \leq \min\{d_i, d_p^* - C_p\}$
2. Jeśli zadania $J_i \prec J_j$ planowalne są przez EDF, to:
   - $a_i^* < a_j^*$: J_i pojawiło się wcześniej niż J_j;
   - $d_i^* < d_j^*$: J_i ma wcześniejszy termin wykonania;

Przechodniość planowalności

Zbiór zadań jest planowalny z zachowaniem wymogów następstwa wykonania zadań wtedy i tylko wtedy, gdy zbiór zadań o zmodyfikowanych parametrach (a_i^*, d_i^*) jest planowalny przez EDF.

1. Dla każdego planu σ zachowującego zależności musi być:
   - $s_i \geq \max\{a_i, a_r^* + C_r\}$
   - $f_i \leq \min\{d_i, d_p^* - C_p\}$
2. Jeśli zadania $J_i \prec J_j$ planowalne są przez EDF, to:
   - $a_i^* < a_j^*$: J_i pojawiło się wcześniej niż J_j;
   - $d_i^* < d_j^*$: J_i ma wcześniejszy termin wykonania;

   zatem:

Zmodyfikowany EDF – przechodniość planowalności

Przechodniość planowalności

Zbiór zadań jest planowalny z zachowaniem wymogów następstwa wykonania zadań wtedy i tylko wtedy, gdy zbiór zadań o zmodyfikowanych parametrach (a_i^*, d_i^*) jest planowalny przez EDF.

1. Dla każdego planu σ zachowującego zależności musi być:
   - $s_i \geq \max\{a_i, a_r^* + C_r\}$
   - $f_i \leq \min\{d_i, d_p^* - C_p\}$
2. Jeśli zadania $J_i \prec J_j$ planowalne są przez EDF, to:
   - $a_i^* < a_j^*$: J_i pojawiło się wcześniej niż J_j;
   - $d_i^* < d_j^*$: J_i ma wcześniejszy termin wykonania;

   zatem:
   - J_j nie może się rozpocząć przed J_i, oraz J_j nie wywłaszczy J_i;

Zmodyfikowany EDF – przechodniość planowalności

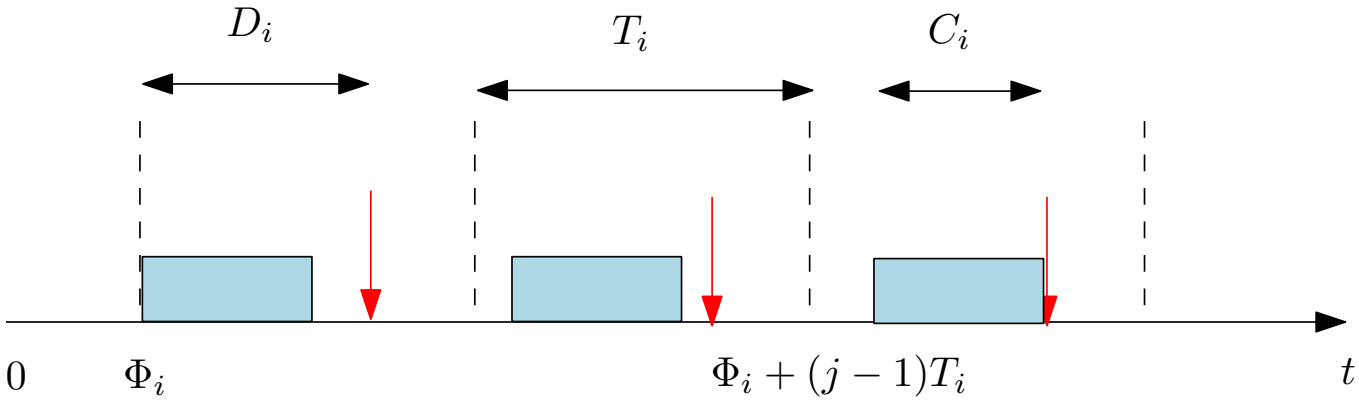
Przechodniość planowalności

Zbiór zadań jest planowalny z zachowaniem wymogów następstwa wykonania zadań wtedy i tylko wtedy, gdy zbiór zadań o zmodyfikowanych parametrach (a_i^*, d_i^*) jest planowalny przez EDF.

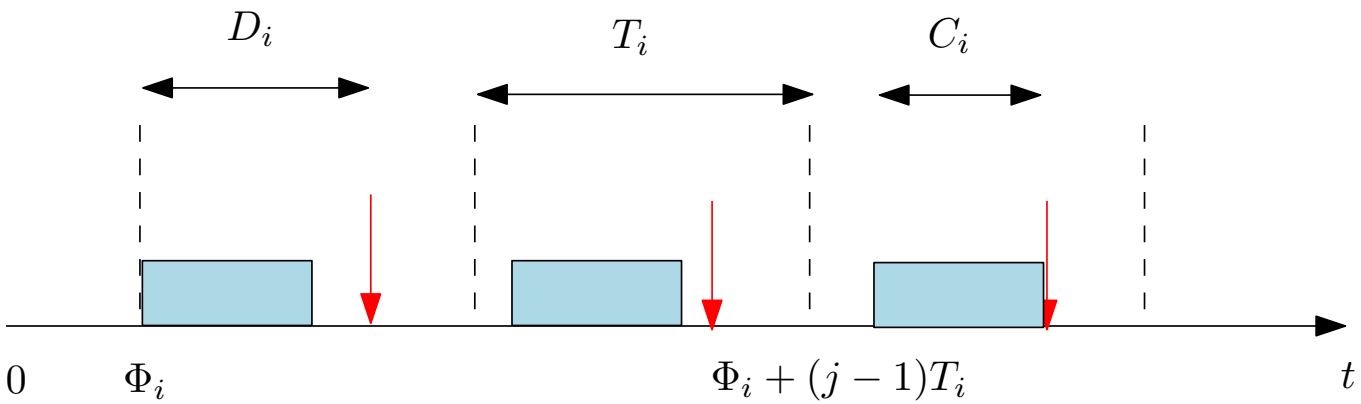
1. Dla każdego planu σ zachowującego zależności musi być:
   - $s_i \geq \max\{a_i, a_r^* + C_r\}$
   - $f_i \leq \min\{d_i, d_p^* - C_p\}$
2. Jeśli zadania $J_i \prec J_j$ planowalne są przez EDF, to:
   - $a_i^* < a_j^*$: J_i pojawiło się wcześniej niż J_j;
   - $d_i^* < d_j^*$: J_i ma wcześniejszy termin wykonania;

   zatem:
   - J_j nie może się rozpocząć przed J_i, oraz J_j nie wywłaszczy J_i;
   - $a_i^* \geq a_i$ oraz $d_i^* \leq d_i$

Planowanie zadań okresowych


D_i
T_i
C_i
0
Φ_i
$\Phi_i + (j-1)T_i$
t

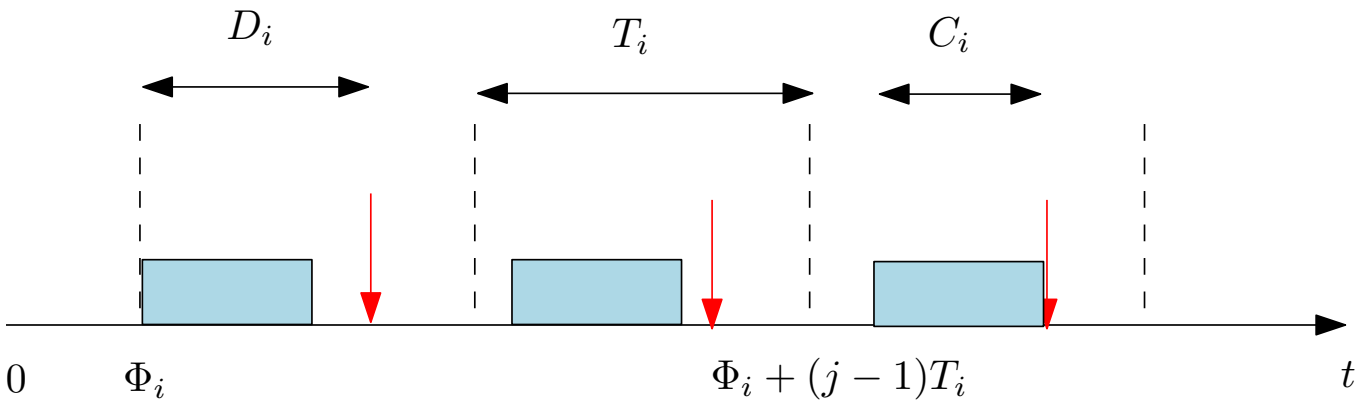
Planowanie zadań okresowych



Każde zadanie okresowe t_i określone jest przez:

- fazę Φ_i – pierwszy moment wystąpienia;

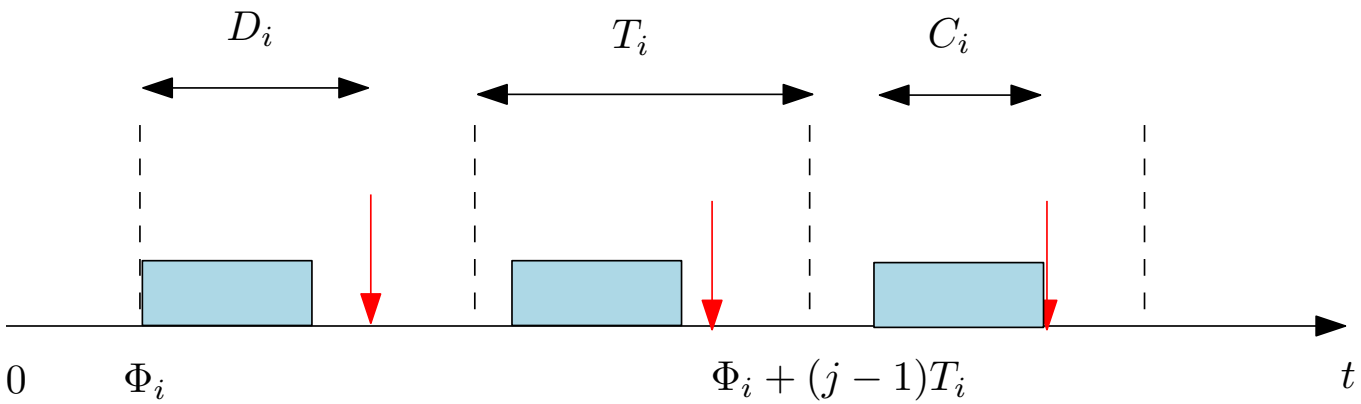
Planowanie zadań okresowych



Każde zadanie okresowe t_i określone jest przez:

- fazę Φ_i – pierwszy moment wystąpienia;
- okres T_i – czas jaki upływa między dwoma kolejnymi wystąpieniami;

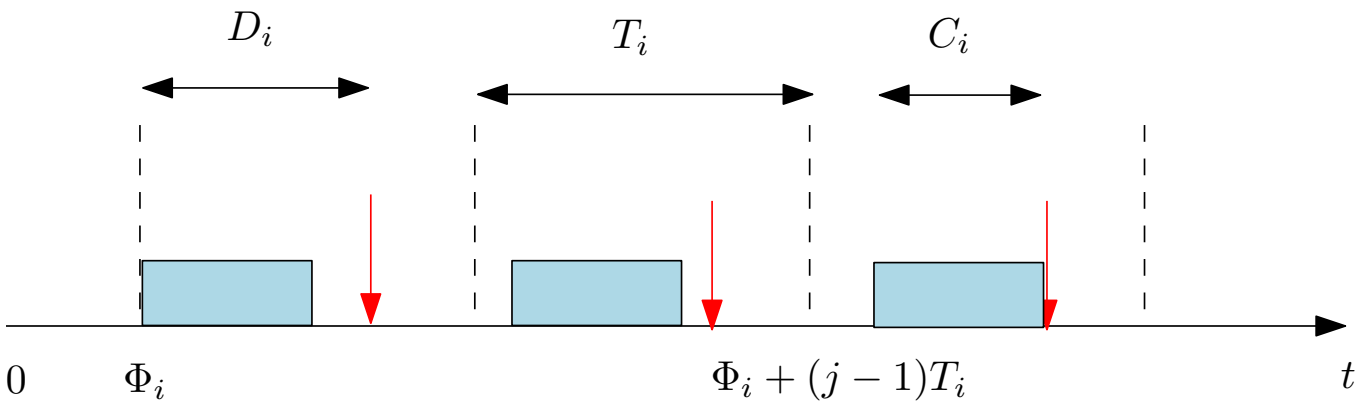
Planowanie zadań okresowych



Każde zadanie okresowe t_i określone jest przez:

- fazę Φ_i – pierwszy moment wystąpienia;
- okres T_i – czas jaki upływa między dwoma kolejnymi wystąpieniami;
- względny termin wykonania (deadline) D_i – okres czasu przeznaczony na wykonanie zadania od momentu jego wystąpienia;

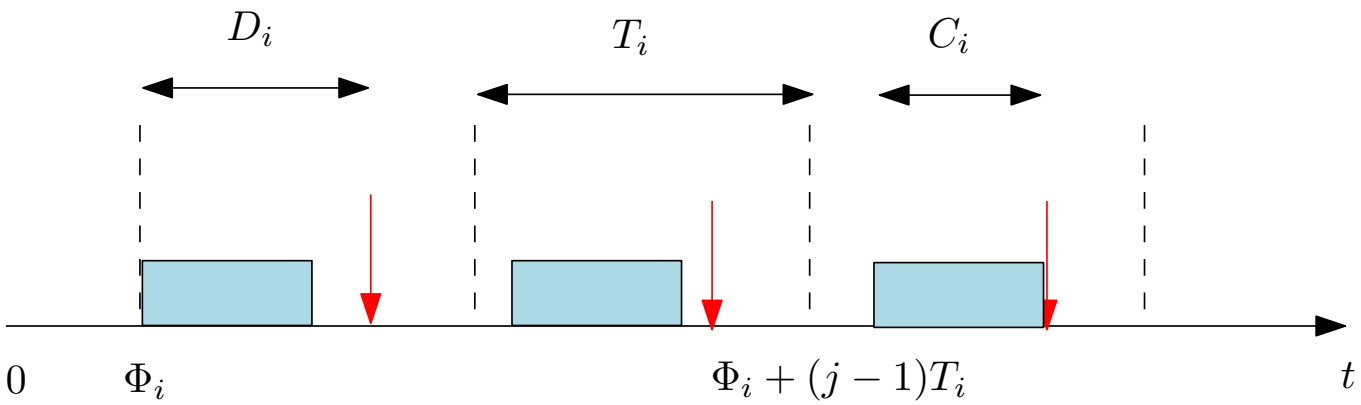
Planowanie zadań okresowych



Każde zadanie okresowe t_i określone jest przez:

- fazę Φ_i – pierwszy moment wystąpienia;
- okres T_i – czas jaki upływa między dwoma kolejnymi wystąpieniami;
- względny termin wykonania (deadline) D_i – okres czasu przeznaczony na wykonanie zadania od momentu jego wystąpienia;
- czas wykonania C_i – czas trwania obliczeń zadania.

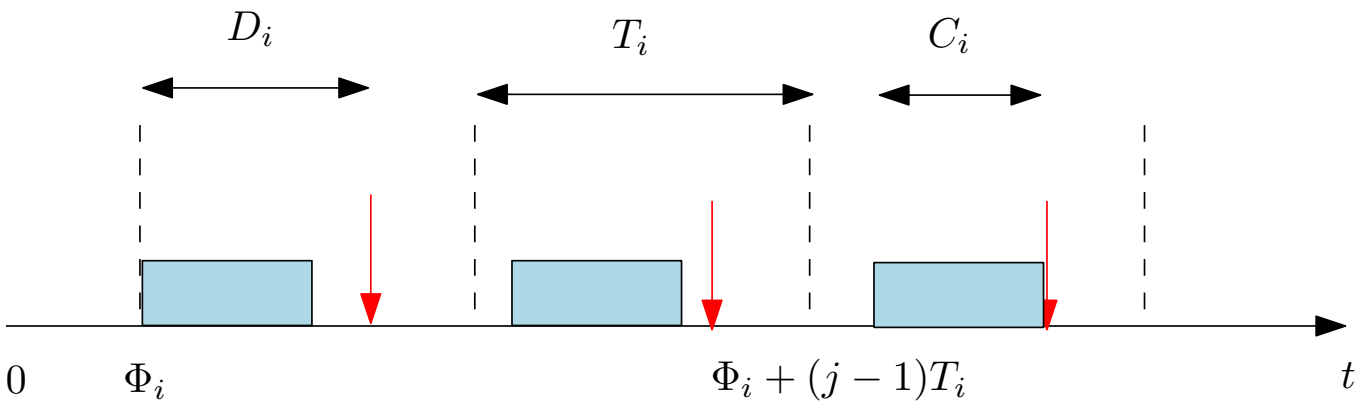
Planowanie zadań okresowych



Każde zadanie okresowe t_i określone jest przez:

- fazę Φ_i – pierwszy moment wystąpienia;
- okres T_i – czas jaki upływa między dwoma kolejnymi wystąpieniami;
- względny termin wykonania (deadline) D_i – okres czasu przeznaczony na wykonanie zadania od momentu jego wystąpienia;
- czas wykonania C_i – czas trwania obliczeń zadania.

Planowanie zadań okresowych



Każde zadanie okresowe t_i określone jest przez:

- fazę Φ_i – pierwszy moment wystąpienia;
- okres T_i – czas jaki upływa między dwoma kolejnymi wystąpieniami;
- względny termin wykonania (deadline) D_i – okres czasu przeznaczony na wykonanie zadania od momentu jego wystąpienia;
- czas wykonania C_i – czas trwania obliczeń zadania.

Plan realizowalny składa się z czasów startu $s_{i,j}$ i ukończenia $f_{i,j}$ dla każdego zadania i w każdej jego instancji (pojawieniu się) j.

Planista RM - rate-monotonic

(Liu, Layland, 1973):

- każde zadanie otrzymuje priorytet $1/T_i$ (proporcjonalny do częstotliwości pojawiania się);

Planista RM - rate-monotonic

(Liu, Layland, 1973):

- każde zadanie otrzymuje priorytet $1/T_i$ (proporcjonalny do częstotliwości pojawiania się);
- wykonywane jest gotowe zadanie z aktualnie najwyższym priorytetem;

Planista RM - rate-monotonic

(Liu, Layland, 1973):

- każde zadanie otrzymuje priorytet $1/T_i$ (proporcjonalny do częstotliwości pojawiania się);
- wykonywane jest gotowe zadanie z aktualnie najwyższym priorytetem;
- system z wywłaszczeniami;

RM – przykład (1)

	t_1	t_2	t_3
Φ_i	0	0	0
T_i	4	6	12
C_i	2	1	4
D_i	4	6	12

RM – przykład (1)

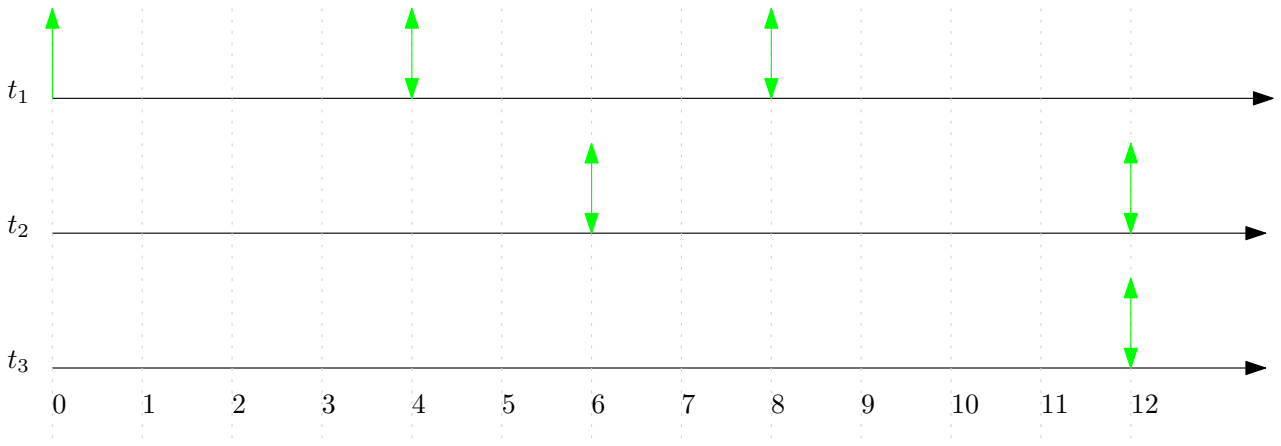
	t_1	t_2	t_3
Φ_i	0	0	0
T_i	4	6	12
C_i	2	1	4
D_i	4	6	12

Policzmy: $C_1/T_1 + C_2/T_2 + C_3/T_3 = 2/4 + 1/6 + 4/12 = 1$.

RM – przykład (1)

	t_1	t_2	t_3
Φ_i	0	0	0
T_i	4	6	12
C_i	2	1	4
D_i	4	6	12

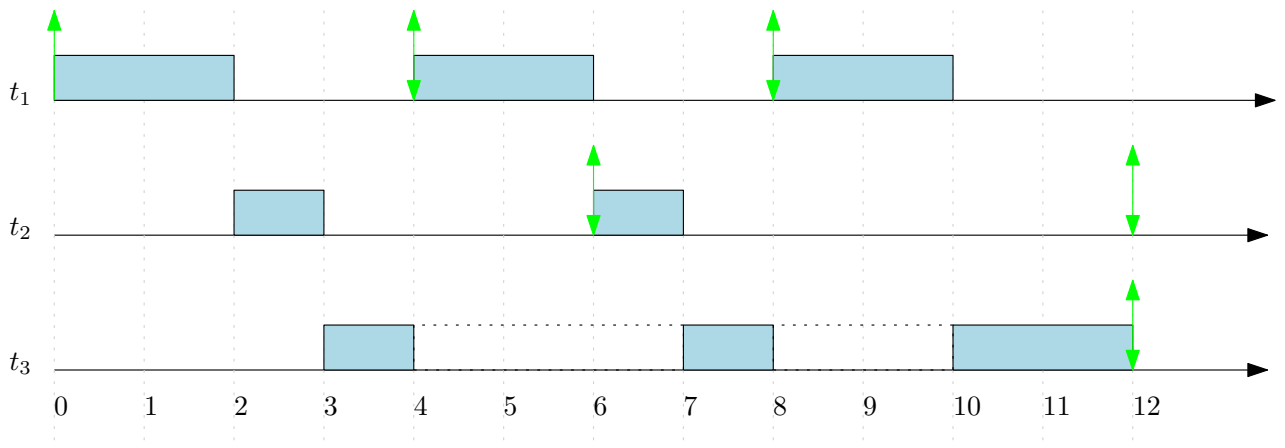
Policzmy: $C_1/T_1 + C_2/T_2 + C_3/T_3 = 2/4 + 1/6 + 4/12 = 1$.



RM – przykład (1)

	t_1	t_2	t_3
Φ_i	0	0	0
T_i	4	6	12
C_i	2	1	4
D_i	4	6	12

Policzmy: $C_1/T_1 + C_2/T_2 + C_3/T_3 = 2/4 + 1/6 + 4/12 = 1$.



RM – przykład (2)

	t_1	t_2	t_3
Φ_i	0	0	0
T_i	4	5	10
C_i	2	2	1
D_i	4	5	10

RM – przykład (2)

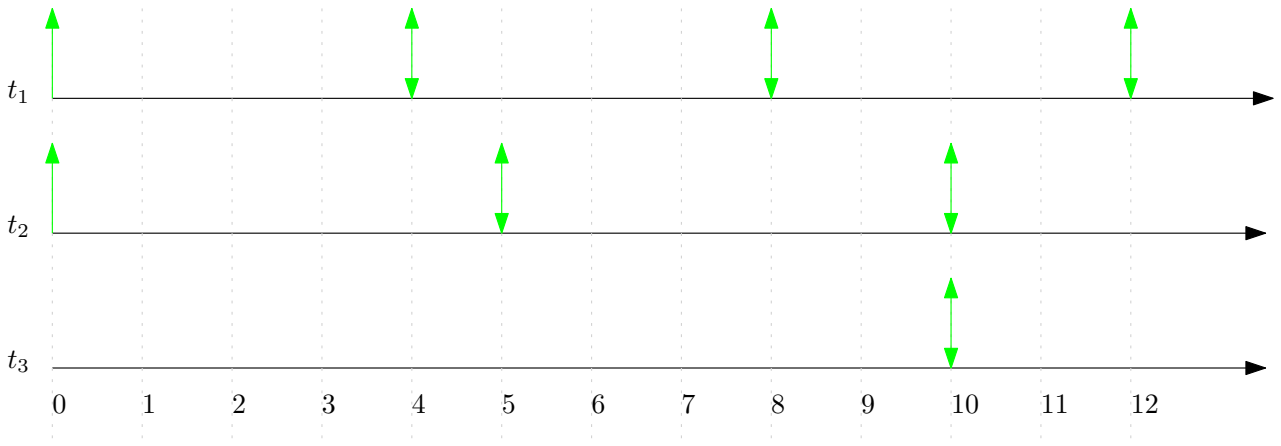
	t_1	t_2	t_3
Φ_i	0	0	0
T_i	4	5	10
C_i	2	2	1
D_i	4	5	10

Policzmy: $C_1/T_1 + C_2/T_2 + C_3/T_3 = 2/4 + 2/5 + 1/10 = 1$.

RM – przykład (2)

	t_1	t_2	t_3
Φ_i	0	0	0
T_i	4	5	10
C_i	2	2	1
D_i	4	5	10

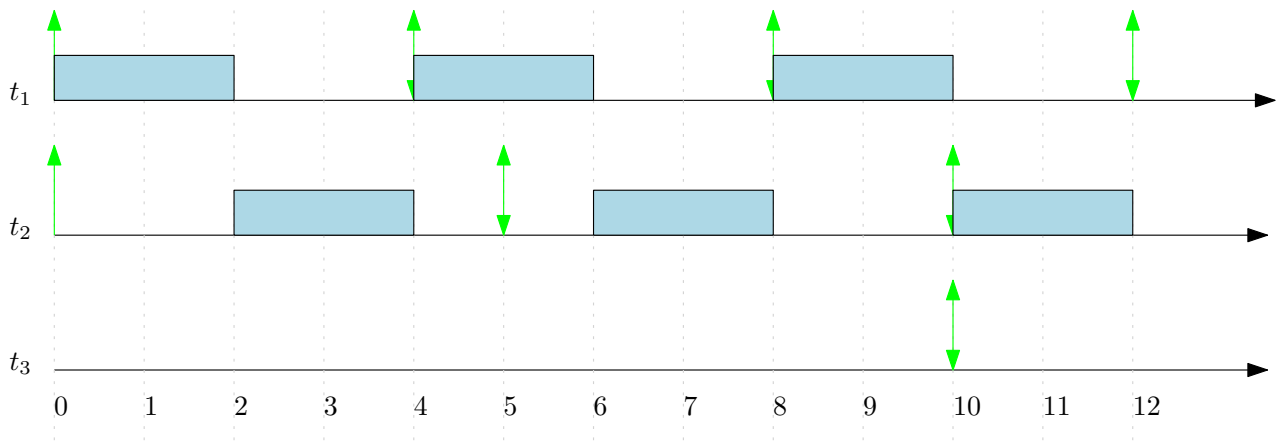
Policzmy: $C_1/T_1 + C_2/T_2 + C_3/T_3 = 2/4 + 2/5 + 1/10 = 1$.



RM – przykład (2)

	t_1	t_2	t_3
Φ_i	0	0	0
T_i	4	5	10
C_i	2	2	1
D_i	4	5	10

Policzmy: $C_1/T_1 + C_2/T_2 + C_3/T_3 = 2/4 + 2/5 + 1/10 = 1$.



Stopień wykorzystania procesora - warunek planowalności

Stopień wykorzystania procesora

Dla danego zbioru n zadań okresowych, stopień wykorzystania procesora to:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}.$$

- warunek konieczny dla stworzenia realizowalnego planu: $U \leq 1$

Stopień wykorzystania procesora - warunek planowalności

Stopień wykorzystania procesora

Dla danego zbioru n zadań okresowych, stopień wykorzystania procesora to:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}.$$

- warunek konieczny dla stworzenia realizowalnego planu: $U \leq 1$
- dla algorytmu RM w pesymistycznym wypadku, dla $U \geq 2 \cdot (2^{1/n} - 1)$ brak gwarancji poprawnego rozwiązania.

Rzeczy do zapamiętania

- różnice między RTOS a “PC-OS”;
- właściwości czasowe zadań w systemie (w tym okresowych);
- rodzaje planistów, jakie warunki biorą pod uwagę;
- algorytm EDD, EDF, LDF, Bratley’a, RM