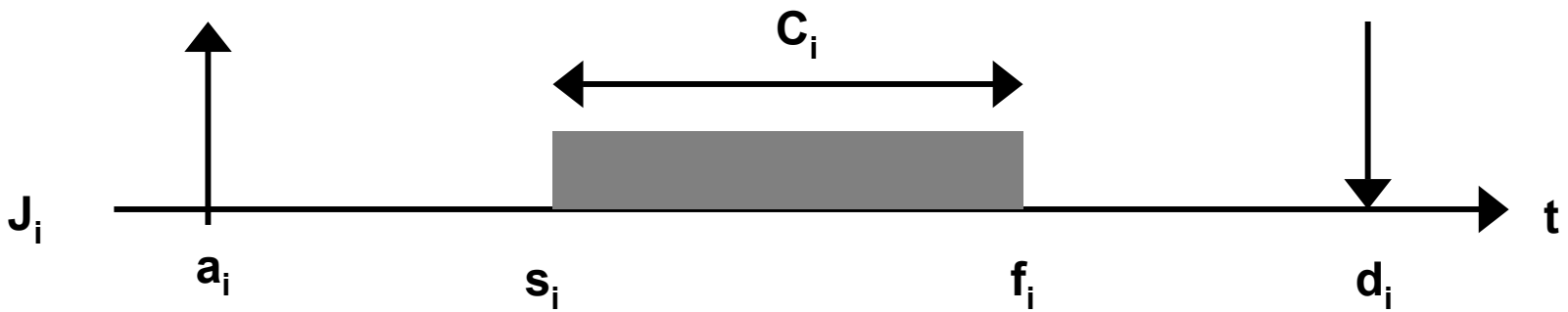
Priorytety zadań - wprowadzenie

- **Podstawowe cechy systemów czasu rzeczywistego, takie jak**
  - **Współbieżność**
  - **Przewidywalność i punktualność**

  są przeciwstawne - niedeterminizm wynikający ze współbieżności ogranicza zapewnienie przewidywalności i punktualności.
- **Niedeterminizm ogranicza się zamykając sekwencje akcji obsługi w jednym zadaniu (por. typowe struktury programowe sterowników mikroprocesorowych)**
- **Często jednak nie można zastosować wyłącznie sekwencyjnego przetwarzania danych z uwagi na niedeterminizm występowania zdarzeń w otoczeniu.**
- **W celu uporządkowania zbioru współbieżnych zadań nadaje się im priorytety.**
- **Odpowiednio dobrane priorytety zapewnić mogą zarówno przewidywalność jak punktualność danemu zbiorowi zadań.**
- **Priorytet określa względną ważność zadań. Na tej podstawie jądro systemu operacyjnego czasu rzeczywistego wybiera wśród zadań, które są w stanie gotowe (*ready*) zadanie „najważniejsze” (o najwyższym priorytecie).**
- **Strategia doboru priorytetów oraz zasada posługiwania się nimi w wykonywaniu zbioru zadań nazywana jest algorytmem szeregowania.**

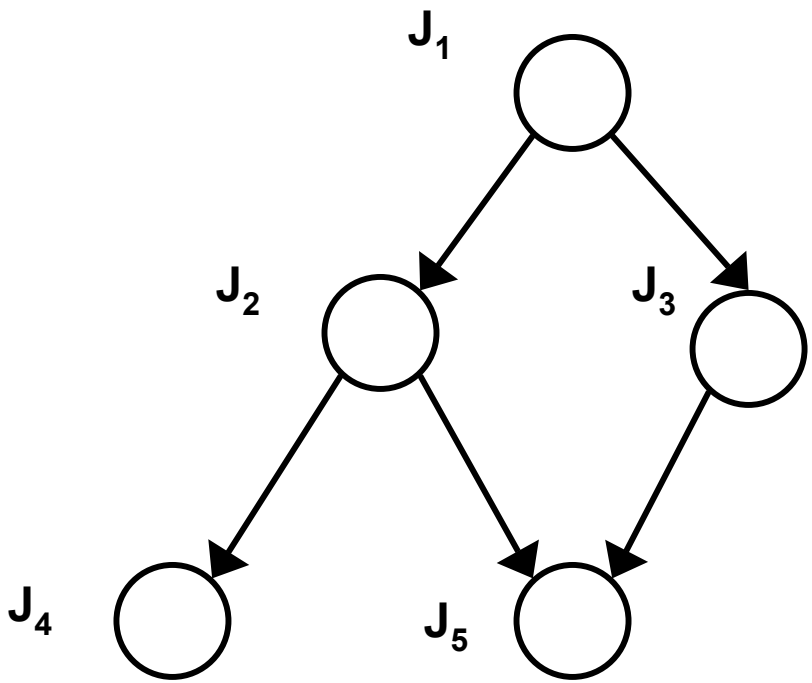
Typy ograniczeń zadań czasu rzeczywistego

- **Ograniczenia czasowe (twarde, miękkie, solidne)**
  - **Parametry opisujące parametry czasowe zadań czasu rzeczywistego:**
    - **Czas napłynięcia zadania (Arrival time) a_i,**
    - **Czas trwania obliczeń (Computation time) C_i,**
    - **Ostateczny termin zakończenia obliczeń (Deadline) d_i,**
    - **Czas rozpoczęcia obliczeń (Start time) s_i,**
    - **Czas zakończenia obliczeń (Finishing time) f_i,**
    - **Krytyczność zadania (twarde, miękkie),**
    - **Wartość zadania v_i,**
    - **Spóźnienie zadania (Lateness) $L_i = f_i - d_i$,**
    - **Luźny czas zadania (Laxity) $X_i = d_i - a_i - C_i$,**



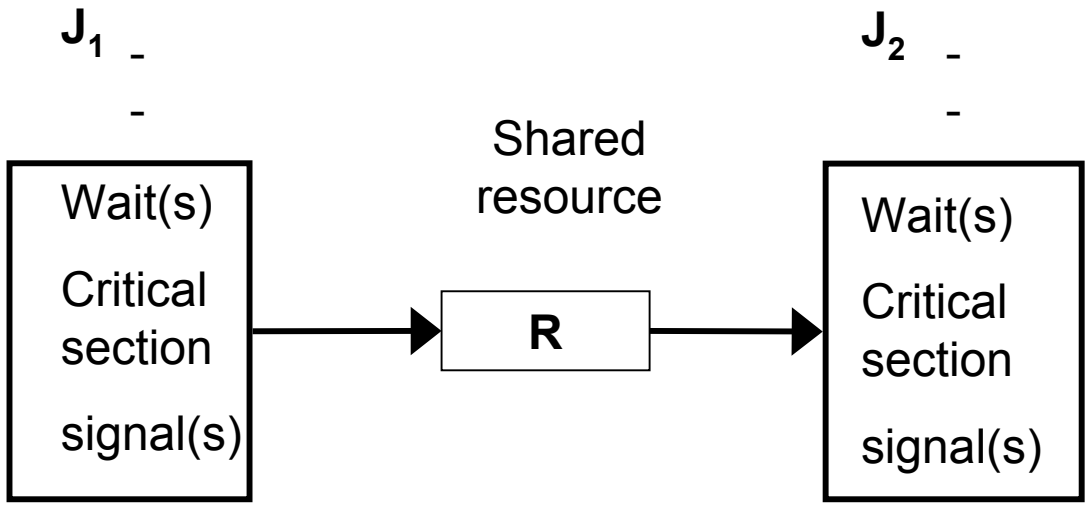
Typy ograniczeń zadań czasu rzeczywistego

- **Ograniczenia pierwszeństwa**



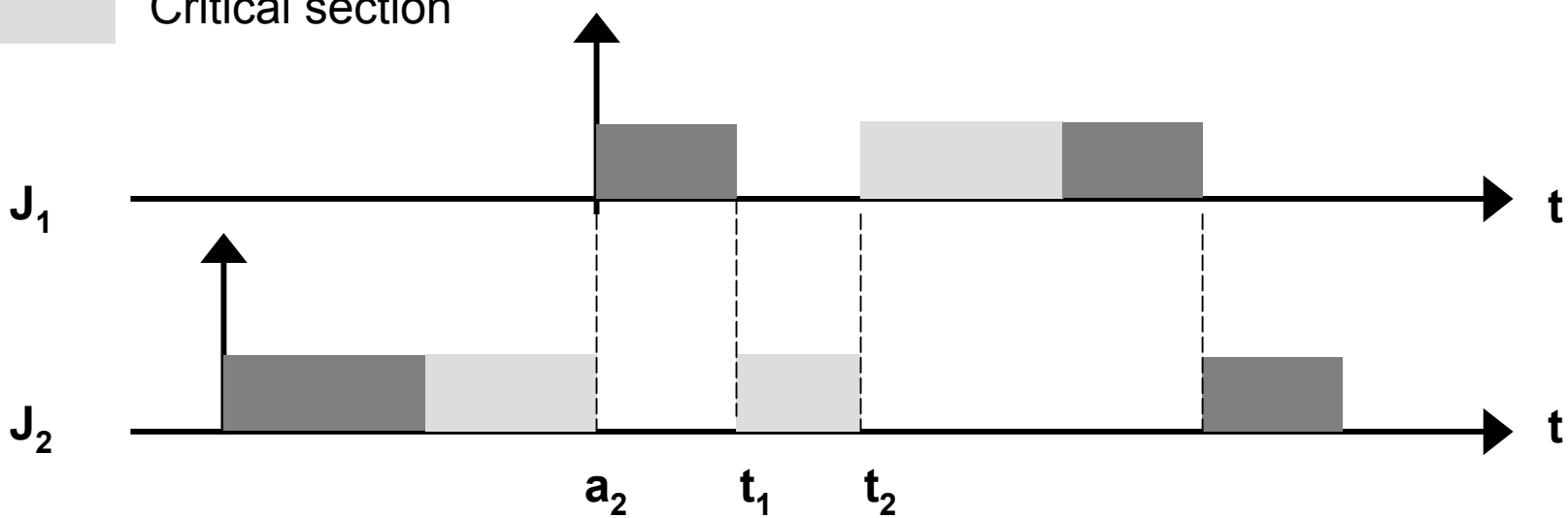
Typy ograniczeń zadań czasu rzeczywistego

- **Ograniczenia na zasobach**




Normal execution

Critical section



Szeregowanie periodycznych zadań czasu rzeczywistego - wprowadzenie

- Wiele aplikacji czasu rzeczywistego (np. sterowanie) składa się ze zbioru zadań wykonywanych cyklicznie (odczyt danych sensorycznych, pętle sterowania, monitorowanie).
- Oczekuje się, że zadania wykonywać się będą cyklicznie z założoną częstotliwością określoną na podstawie wymagań danej aplikacji (np. wolna, szybka pętla obliczeń)
- Kiedy aplikacja ma się składać z wielu współbieżnych zadań z określonymi ograniczeniami czasowymi projektant musi gwarantować, że każda periodyczna instancja jest regularnie aktywowana z prawidłową częstotliwością i kończy swoje obliczenia przed upłynięciem ostatecznego czasu zakończenia obliczeń.
- Jednym z pierwszych opracowanych podejść było:
  - Wykonywanie cykliczne (ang. Cyclic Executive)
- Podstawowe algorytmy szeregowania dla zbioru cyklicznych zadań, to:
  - Rate Monotonic (RM)
  - Earliest Deadline First (EDF)
  - Deadline Monotonic Priority Ordering (DMPO)

Wykonywanie cykliczne

- Jednym z popularnych podejść do konstruowania systemów czasu rzeczywistego jest zastosowanie wykonywania cyklicznego
- Podejście do projektowania jest współbieżne, ale generowany kod aplikacji jest zestawem procedur wywoływanych zgodnie z tablicą
- Cała tablica obejmuje główny cykl systemu podzielony zwykle na pośrednie cykle o stałej długości
- Pośrednie cykle decydują o minimalnym cyklu w systemie
- Główny cykl określa maksymalny cykl systemu

Główna zaleta:

- Taki system jest w pełni deterministyczny

Wykonywanie cykliczne – przykład

- **Rozważany będzie następujący zestaw procesów:**

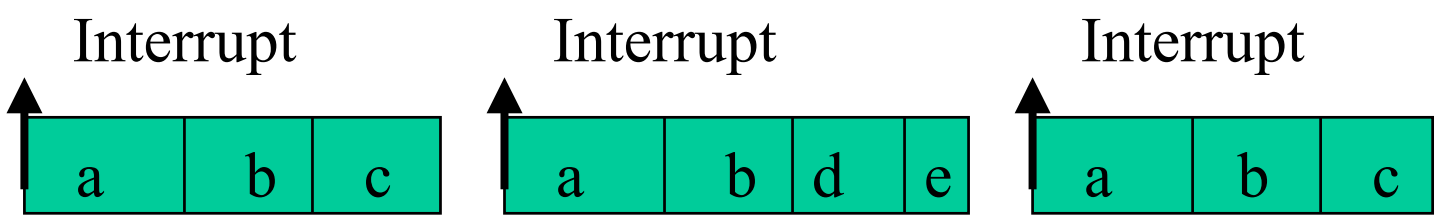
Process	Period, T	Computation Time, C
a	25	10
b	25	8
c	50	5
d	50	4
e	100	2

Wykonywanie cykliczne – przykład

- **Uszeregowanie procesów można zapisać następująco:**

```
loop
   wait_for_interrupt;
   procedure_for_a; procedure_for_b; procedure_for_c;
   wait_for_interrupt;
   procedure_for_a; procedure_for_b; procedure_for_d;
   procedure_for_e;
   wait_for_interrupt;
   procedure_for_a; procedure_for_b; procedure_for_c;
   wait_for_interrupt;
   procedure_for_a; procedure_for_b; procedure_for_d;
end loop;
```

- **Co przekłada się na następujący schemat wykonywania procesów:**



Wykonywanie cykliczne – właściwości

- **Wyeliminowano zestaw procesów (zadań) z systemu. Każdy z pośrednich cyklów jest sekwencją wołań procedur**
- **Procedury współdzielą wspólna przestrzeń adresową stad mogą wymieniać dane. Dane nie muszą być chronione (np. przez semafory), ponieważ współbieżny dostęp do zasobów jest niemożliwy**
- **Wszystkie okresy „procesów” muszą być wielokrotnością cyklu pośredniego.**

Wykonywanie cykliczne – problemy

- **Mogą pojawić się problemy z włączeniem w system procesów o długim okresie. Główny cykl jest zarazem maksymalnym cyklem w systemie bez konieczności włączania dodatkowych modułów szeregujących**
- **Zadania sporadyczne są trudne (jeśli niemożliwe) do włączenia w system**
- **Stworzenie tabeli wykonywania cyklicznego jest trudne. Trudne jest również dokonywanie uaktualnień. Z punktu widzenia matematycznego otrzymuje się problem NP-trudny.**
- **Może zaistnieć konieczność podzielania jednego „procesu" na sekwencję procedur o jednakowym czasie wykonywania, co może powodować dodatkowe błędy (program traci „eleganckość" z punku widzenia inzynierii oprogramowania)**
- **Podejście praktycznie uniemożliwia zastosowanie innych, bardziej elastycznych metod szeregowania**
- **W praktycznych zastosowaniach można zrezygnować z pełnego determinizmu aplikacji z zachowaniem przewidywalności**

Wniosek:

- **Dla prostych, cyklicznych systemów warto rozważyć wykonywanie cykliczne, dla pozostałych zastosować trzeba inne, zaawansowane rozwiązania.**

Założenia dotyczące zbioru zadań przewidzianych do szeregowania (RM, EDF, DMPO)

1. Instancje (kolejne wystąpienia) zadania periodycznego t_i są regularnie aktywowane ze stałą częstotliwością. Przedział T_i pomiędzy kolejnymi aktywacjami nazywany jest okresem zadania.
2. Wszystkie instancje zadania mają ten sam najdłuższy czas wykonywania (WCET- Worst Case Execution Time) C_i.
3. Wszystkie instancje zadania mają tą samą wartość względnego ostatecznego terminu zakończenia obliczeń (Relative deadline) D_i.
4. Wszystkie szeregowane zadania są niezależne tzn. nie ma pomiędzy nimi relacji poprzedzania ani ograniczeń na zasobach.

Dodatkowo:

5. Zadanie nie może zawiesić samego siebie (np. Dla wykonania operacji wej/wyj).
6. Zadania są wprowadzane do wykonywania z chwilą ich nadejścia do kolejki.
7. W rozważaniach pominięte będą czasy na obsługę przełączania zadań itp.

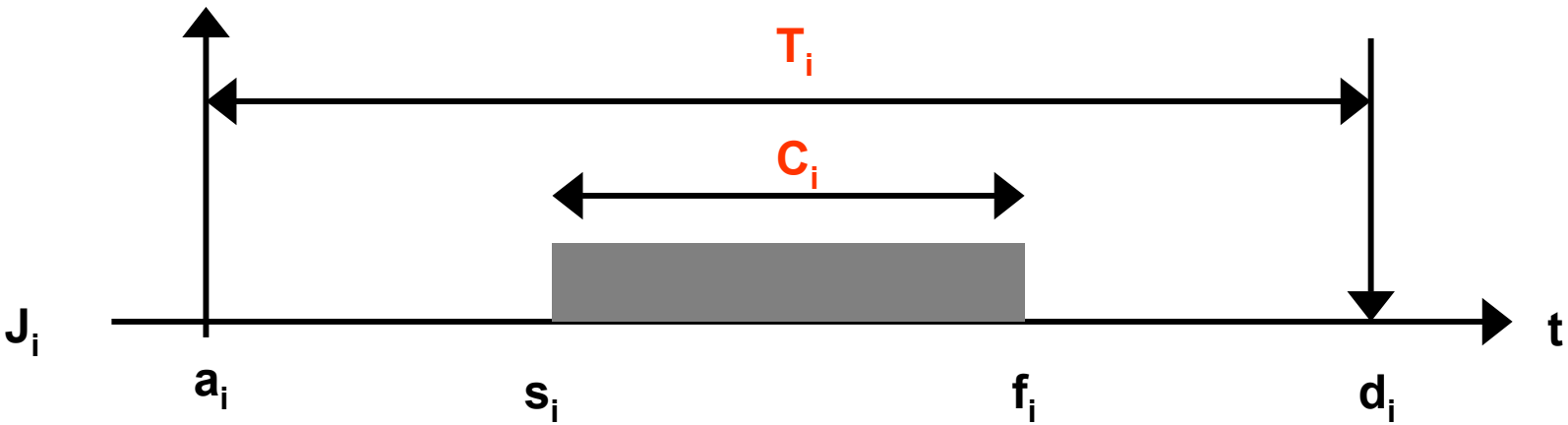
Zadania spełniające założenia 1.-4. Opisać można przez 3 parametry:

- Czas aktywacji pierwszej instancji danego zadania Φ_i,
- Okres danego zadania T_i,
- Najdłuższy czas wykonywania danego zadania C_i.

Współczynnik wykorzystania procesora

- **Wzór:**

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



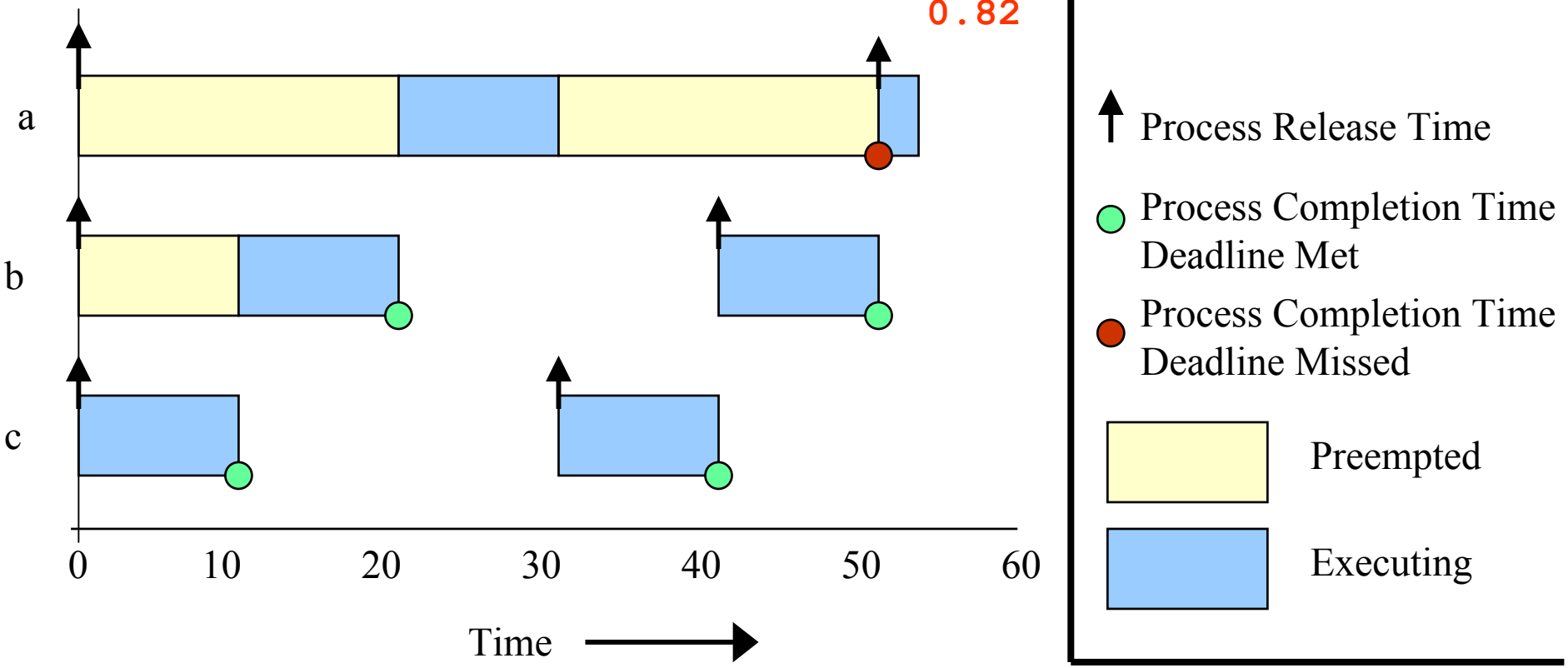
- **Współczynnik opisuje obciążenie procesora podczas wykonywania zbioru cyklicznych zadań.**
- **Dla różnych algorytmów szeregowania i różnych zbiorów zadań współczynnik może przyjmować różne graniczne wartości, dla których dany zbiór zadań jest szeregowalny (może wykonać wszystkie swoje obliczenia w zadanych ograniczeniach czasowych).**
- **Jeśli współczynnik przyjmuje wartość większą od 1, dany zbiór zadań nie może być szeregowany przez żaden algorytm.**

Algorytm szeregowania Rate Monotonic (RM) – podstawowe właściwości

- **Zasada przydziału priorytetów zadaniom:**
  - **Priorytet zadaniom przydziela się w zależności od częstotliwości wznawiania obliczeń (okresu).**
  - **Zadanie najczęściej wznawiane otrzymuje najwyższy priorytet. Kolejnym zadaniom o kolejnych większych okresach przydziela się kolejne niższe priorytety.**
  - **Priorytety przydziela się na stałe, przed rozpoczęciem wykonywania obliczeń przez zadania.**
- **Z założenia algorytm RM przyjmuje możliwość przełączenia (wywłaszczenia) bieżącego zadania przez nowo aktywowane zadanie o wyższym priorytecie (mniejszym okresie).**
- **Wykazano matematycznie, że algorytm RM jest optymalny spośród wszystkich algorytmów szeregowania zadań czasu rzeczywistego ze stałym przydziałem priorytetów.**
- **Wskazano zasadę obliczanie maksymalnej wartości współczynnika wykorzystania procesora, dla której dany zbiór zadań czasu rzeczywistego jest szeregowalny.**

Algorytm szeregowania RM – interpretacja

Process	Period T	ComputationTime C	Priority P	Utilization U
a	50	12	1	0.24
b	40	10	2	0.25
c	30	10	3	0.33
				0.82



Algorytm szeregowania RM – warunek szeregowalności

- **Teoretyczny warunek szeregowlaności (dla małej ilości zadań):**

$$U_{gr} = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{\frac{1}{n}} - 1\right)$$, gdzie n – ilość zadań (1973)

$$\prod_{i=1}^{n}\left(\frac{C_i}{T_i} + 1\right) \leq 2$$, gdzie n – ilość zadań (2003)

- **Dla dużej ilości zadań (1973):**

$$U_{\text{gr}} = \ln 2 \approx \mathbf{0.69}$$

- **Dla przypadkowo dobieranych zbiorów zadań wykazano eksperymentalnie graniczną wartość współczynnika wykorzystania procesora, przy którym zbiór zadań jest szeregowalny:**

$$U_{\text{gr}} = \mathbf{0.88}$$

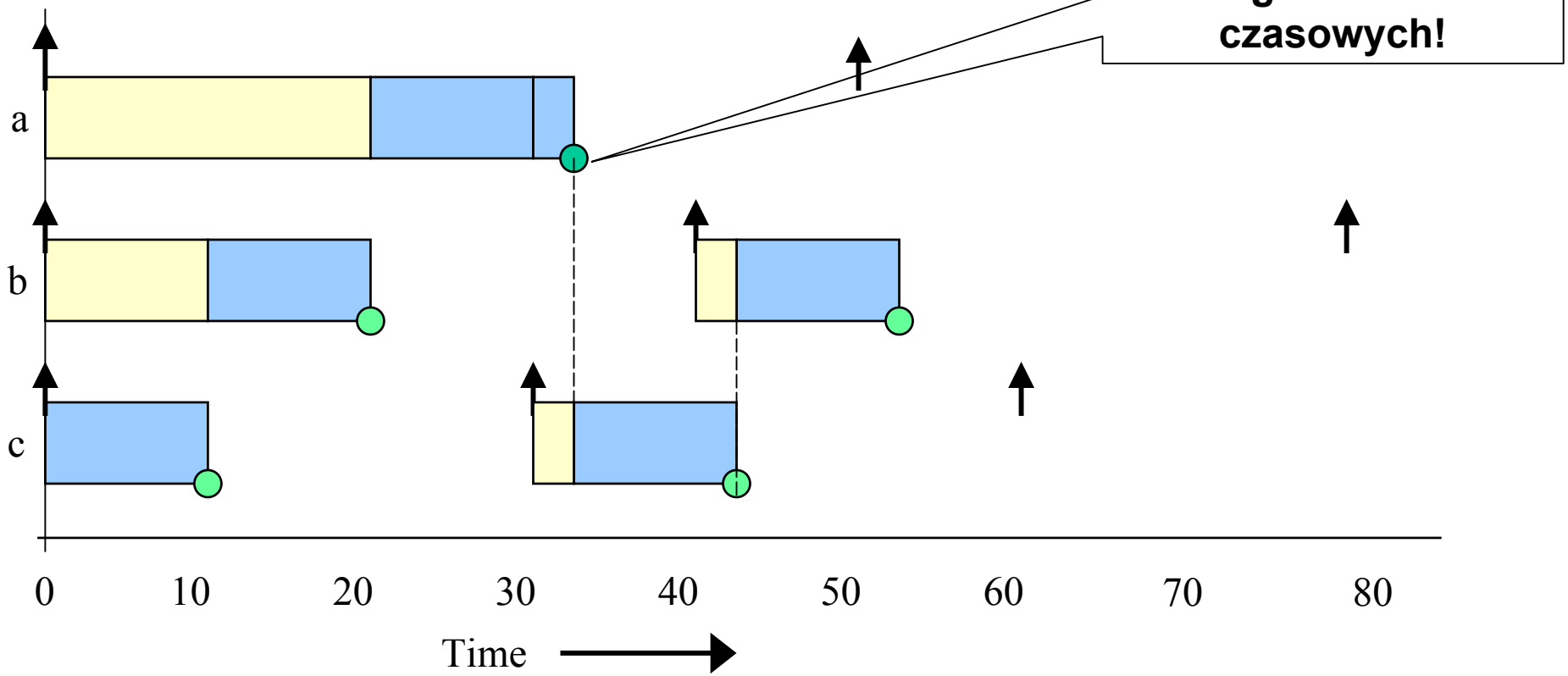
- **Jeśli dla danego zbioru zadań współczynnik wykorzystania procesora mieści się pomiędzy U_{gr} a 1, to nic nie można powiedzieć o wykonalności danego zbioru.**

Algorytm szeregowania Earliest Deadline First (EDF) – podstawowe właściwości

- **Zasada przydziału priorytetów zadaniom:**
  - **Priorytet zadaniom przydziela się dynamicznie w zależności wartości absolutnego ostatecznego terminu zakończenia obliczeń.**
  - **Zadanie, które musi najszybciej zakończyć obliczenia otrzymuje najwyższy priorytet. Priorytety pozostałych zadań przydziela się według kolejnych zbliżających się dla nich ostatecznych terminów zakończenia obliczeń.**
  - **W związku z tym, że absolutny ostateczny termin zakończenia obliczeń w czasie wykonywania każdej z instancji zadania może ulec zmianie, zmianie w sposób dynamiczny ulegają również priorytety danych instancji zadań .**
- **Z założenia algorytm EDF przyjmuje możliwość przełączenia (wywłaszczenia) bieżącego zadania przez nowo aktywowane zadanie o wyższym priorytecie.**
- **Wykazano matematycznie, że algorytm EDF jest optymalny spośród wszystkich algorytmów szeregowania zadań czasu rzeczywistego z dynamicznym przydziałem priorytetów.**
- **Wskazano zasadę obliczanie maksymalnej wartości współczynnika wykorzystania procesora, dla której dany zbiór zadań czasu rzeczywistego jest szeregowalny.**
- **Algorytm jest również optymalny dla zadań nieperiodycznych.**

Algorytm szeregowania EDF – interpretacja

Process	Period T	ComputationTime C	Priority P	Utilization U
a	50	12	1	0.24
b	40	10	2	0.25
c	30	10	3	0.33



Algorytm szeregowania EDF – dyskusja

- **Warunek szeregowlaności:**

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

- **Dlaczego EDF nie jest preferowanym algorytmem szeregowania:**
  - **Algorytmy ze stałymi priorytetami są łatwiejsze w implementacji (mniejszy narzut obliczeniowy dla sytemu operacyjnego).**
  - **W przypadku przepełnienia zachowanie algorytmów ze stałymi priorytetami jest bardziej przewidywalne (procesy o niższym priorytecie nie wykonają się terminowo); Natomiast algorytm EDF może spowodować efekt domino, w którym wiele procesów nie spełni swoich ograniczeń czasowych.**
  - **Zastosowanie współczynnika utylizacji do określenia szeregowalności jest mylące. W przypadku algorytmów ze stałymi priorytetami możliwe jest poprawne działanie systemu pomimo, że przekroczy się teoretycznie wyznaczoną graniczną wartość współczynnika.**

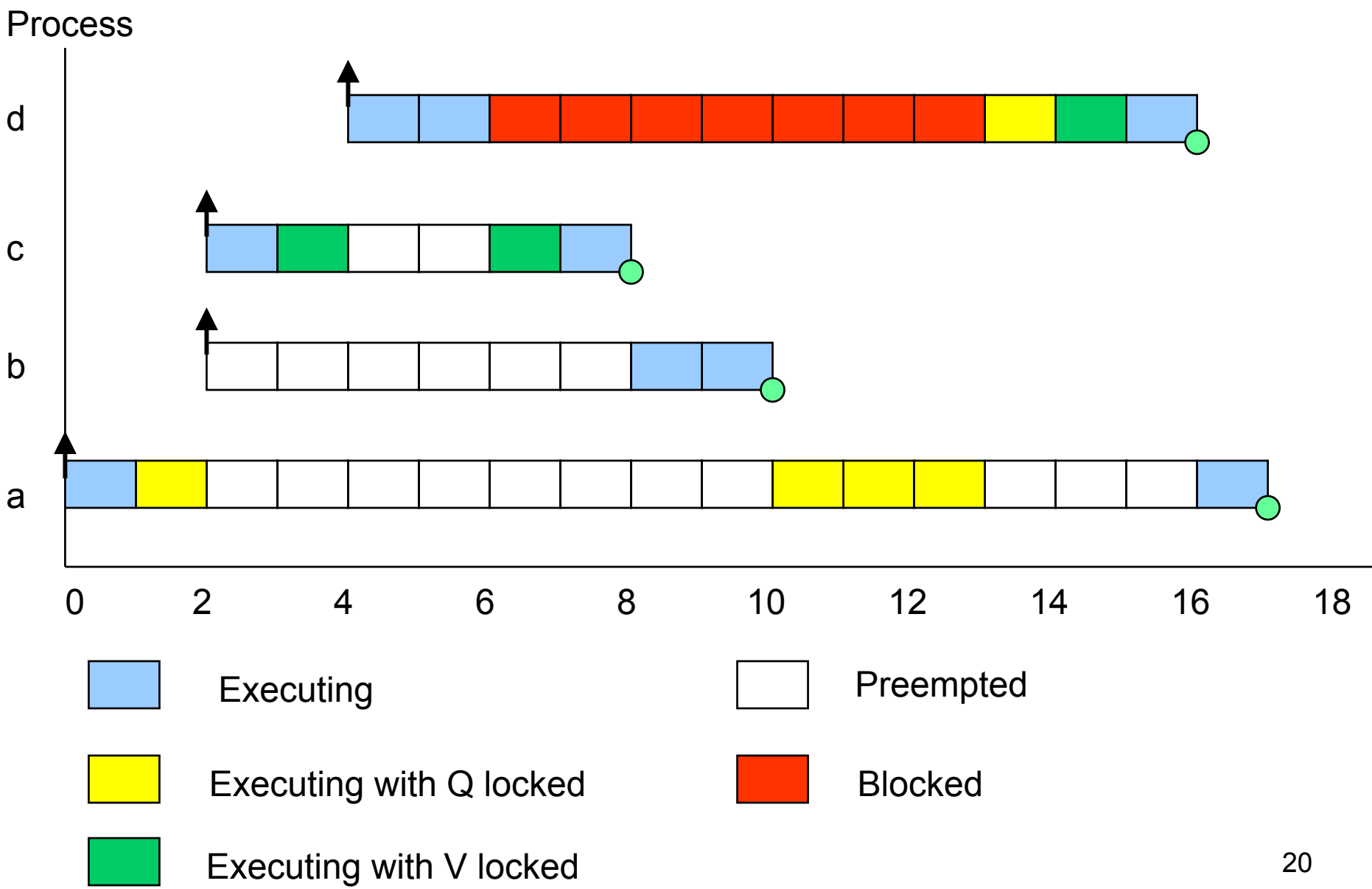
Współdzielenie zasobów - inwersja priorytetów

- Jeśli dany proces jest zawieszony i oczekuje na zakończenie obliczeń przez proces o niższym priorytecie, wtedy model priorytetów w danym systemie jest w pewnym sensie podważony.
- Mówi się wtedy, że w systemie zachodzi inwersja priorytetów.
- Proces oczekujący na inny proces o niższym priorytecie nazywa się procesem zablokowanym (blocked).

- Dla zilustrowania ekstremalnego przykładu inwersji priorytetów rozważone zostanie wykonywanie 4 periodycznych zadań: a, b, c i d oraz dwu zasobów q i v.

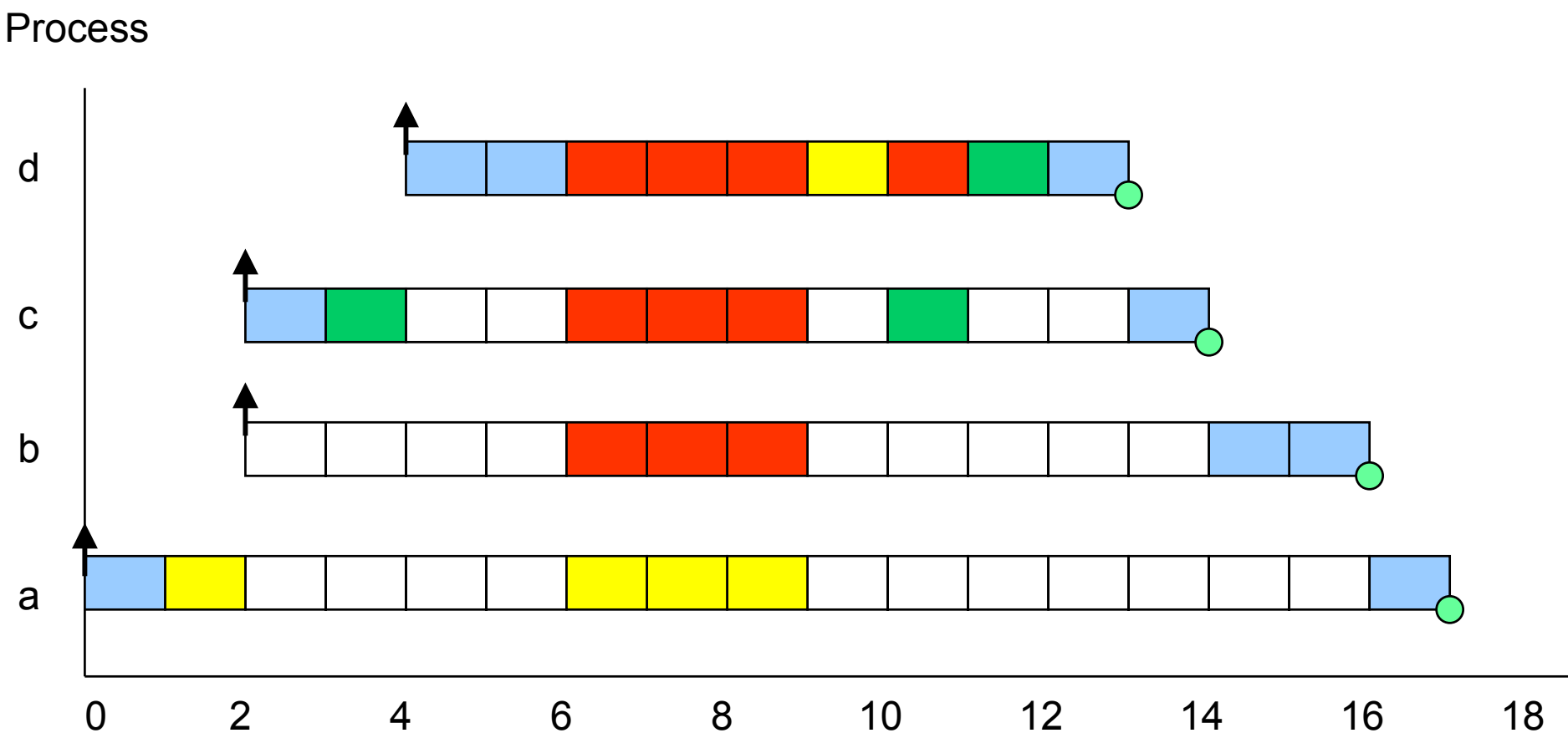
Process	Priority	Execution Sequence	Release Time
a	1	EQQQQE	0
b	2	EE	2
c	3	EVVE	2
d	4	EEQVE	4

Inwersja priorytetów - przykład



Protokół dziedziczenia priorytetów (Priority Inheritance Protocol) - przykład

- **Jeśli proces p jest blokowany przez proces q, wtedy proces q jest wykonywany z priorytetem procesu p.**



Protokół dziedziczenia priorytetów - dyskusja

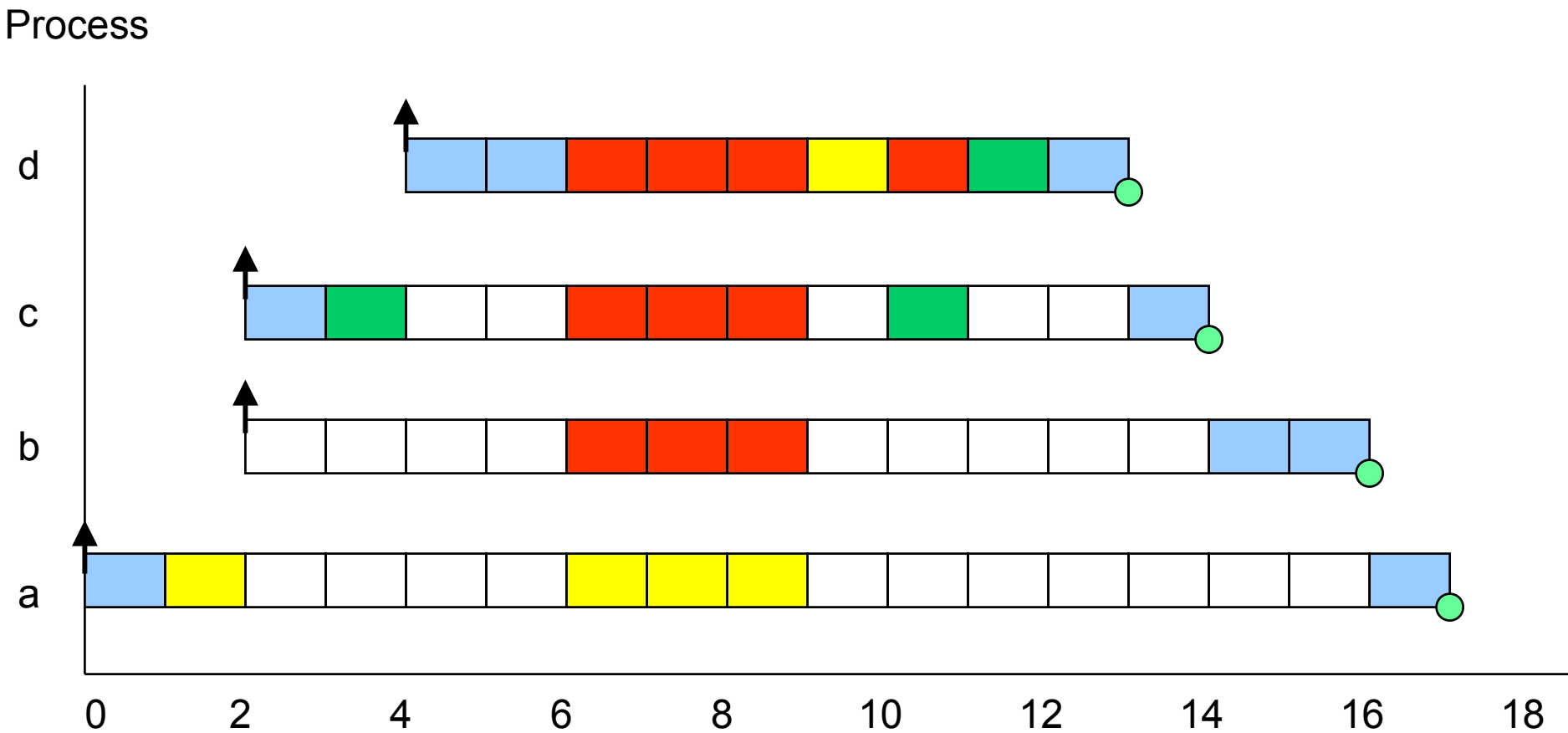
- **Warunek szeregowalności :**

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \max\left(\frac{B_1}{T_1},...,\frac{B_n}{T_n}\right) \leq n(2^{1/n} - 1)$$

- **Algorytm wyliczania czasów zablokowania podano m. in. w:**
  G. C. Butazzo: Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications, Kluver Academic Publishers 1997.

- **Algorytm nie rozwiązuje dwu problemów:**
  - **Łańcucha zablokowań**
  - **Deadlock**

Protokół dziedziczenia priorytetów – łańcuch zablokowań



Protokół dziedziczenia priorytetów – deadlock

a

-

-

Wait(s_Q)

Wait(s_V)

Signal(s_V)

Signal(s_Q)

b

-

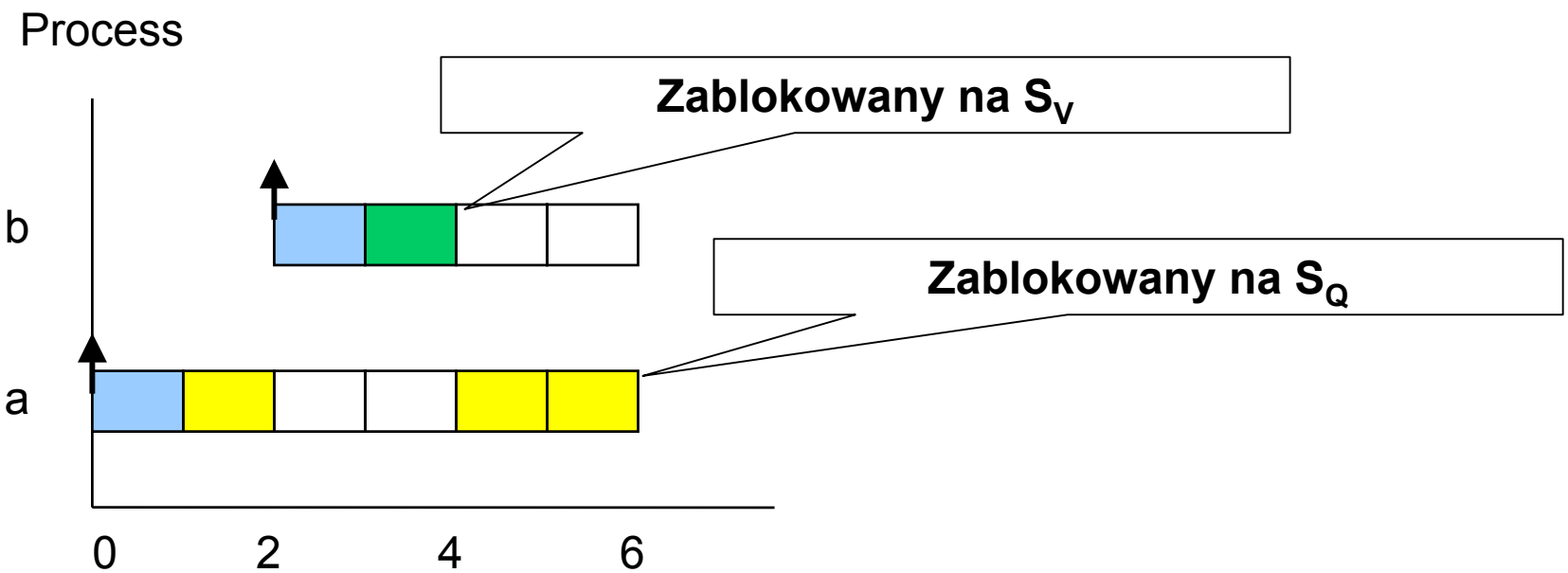
-

Wait(s_V)

Wait(s_Q)

Signal(s_Q)

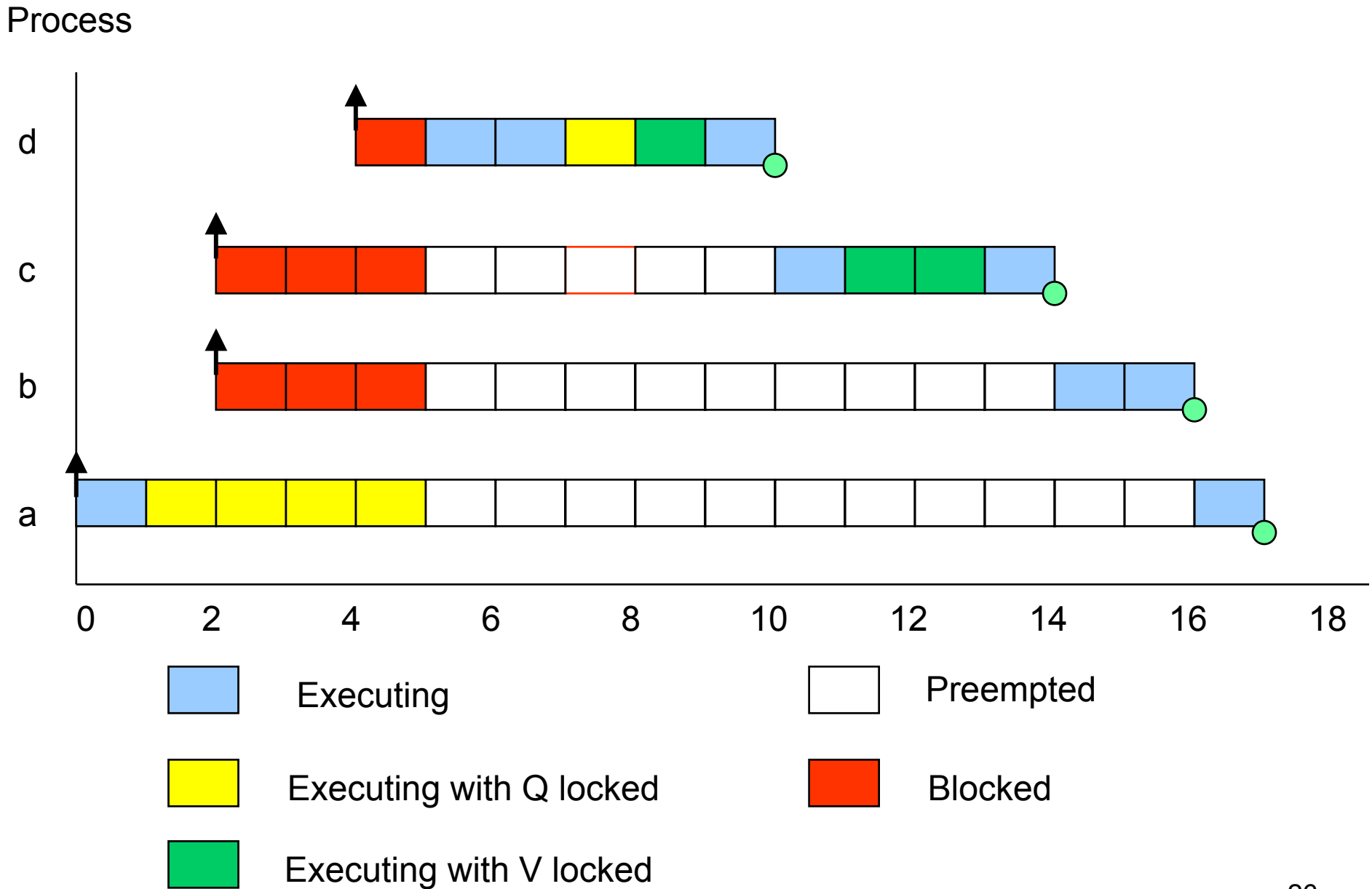
Signal(s_V)



Protokół pułapu priorytetów (Priority Ceiling Protocol)

- **Na pojedynczym procesorze:**
  - **Proces o wyższym priorytecie może być zablokowany przez procesy o niższych priorytetach tylko raz podczas swojego wykonywania.**
  - **Nie występuje deadlock**
  - **Nie występuje łańcuch zablokowań**
  - **Wzajemne wykluczanie podczas dostępu do zasobów jest zapewnione**
- **Algorytm:**
  - **Każdy proces posiada przywiązany statycznie określony priorytet**
  - **Każdy zasób posiada określoną statyczną wartość pułapu będącą maksymalnym priorytetem wśród procesów, które go używają**
  - **Dany proces posiada dynamiczny priorytet który określany jest jako maksimum ze statycznie przydzielonego priorytetu i wartości pułapów wszystkich zasobów z których będzie korzystał**
  - **W konsekwencji proces może być zablokowany jedynie na początku swojego wykonywania**
  - **Jeśli proces rozpoczyna swoje wykonywanie, wszystkie zasoby, których potrzebuje muszą być zwolnione. Jeśli nie były, to jakiś inny proces mógł mieć identyczny lub wyższy priorytet i wykonywanie danego procesu było odłożone.**

Protokół pułapu priorytetów - przykład



Algorytmy szeregowania i protokoły przydziału zasobów - uzupełnienie

- **Krytyczne zadania aperiodyczne (wywoływane przerwaniami) traktuje się podczas projektowania systemów jako periodyczne o pewnej maksymalnej częstotliwości wznawiania.**
- **Niekrytyczne zadania aperiodyczne wykonywane są w tle zadań krytycznych, przy czym wprowadza się tzw. serwer nadzorujący wykonywanie tych zadań.**
- **Przy projektowaniu systemów rozważa się również możliwość przeładowania systemu i konsekwencji takiego stanu.**
- **Dla systemów wieloprocesorowych istnieją odpowiedniki algorytmów RM i EDF.**
- **Dla algorytmu EDF nie sformułowano protokołów dziedziczenia i pułapu priorytetów.**
- **Najprawdopodobniej najlepszym schematem przydziału zasobów jest stosowa strategia zasobów (Stack Resource Policy [Butazzo 1997]).**

Język Ada - priorytety

- **W języku Ada możliwe jest definiowanie priorytetów, jeśli w danej dystrybucji zapewniona jest implementacja aneksu D specyfikacji (Real-Time Systems).**
- **Model priorytetów dla języka Ada umożliwia definiowanie:**
  - **Tzw. bazowych i aktywnych priorytetów,**
  - **Protokółu PCP przydziału zasobów,**
  - **Reguły szeregowania zadań,**
  - **Priorytetów dynamicznych.**
- **Priorytety można przyporządkowywać do zadań i obiektów chronionych.**
- **Priorytety przyjmują wartości z dwu przedziałów:**
  - **Priority – niższe priorytety standardowe – przynajmniej 30 wartości jako minimum implementacyjne,**
  - **Interrupt_Priority – wyższe priorytety przerwań – przynajmniej 1 wartość jako minimum implementacyjne.**

Język Ada– typy wartości priorytetów

```
subtype Any_Priority is Integer
        range Implementation-Defined;
subtype Priority is Any_Priority range
        Any_Priority'First .. Implementation-Defined;
              -- dostępnych powinno być co najmniej 30 wartości

subtype Interrupt_Priority is Any_Priority range
        Priority'Last + 1 .. Any_Priority'Last;
              -- dostępna powinna być przynajmniej 1 wartość

Default_Priority : constant Priority :=
        (Priority'First + Priority'Last)/2;
```

Ada - pragma *priority* – nadawanie bazowych priorytetów

```
task Controller is
  pragma Priority(10); -- ustalenie priorytetu zadania
end Controller;
-------------------------------------------------
task type Servers(Pri : System.Priority) is
  -- każda instancja zadania może mieć inny priorytet:
  entry Service1(...);
  entry Service2(...);
  pragma Priority(Pri);
end Servers;
-------------------------------------------------
with System;
...
task type SEMAPHORE is
      entry ACCESS_TO;
      entry FREE;
pragma Priority(System.Priority'First);   -- priorytet
end SEMAPHORE;              -- niezależny od implementacji
```

Ada – protokoły przydziału zasobów

```
-- Właczenie protokołu PCP w programie:
pragma Locking_Policy(Ceiling_Locking);
```

- Konkretne implementacje języka mogą udostępniać również inne protokoły przydziału zasobów.
- Obiekty krytyczne wymagają utrzymania spójności ich danych.
- Zastosowanie priorytetów powinno gwarantować wzajemne wykluczanie.
- Do każdego obiektu chronionego jest wiązany pułapowy priorytet, który jest większy lub równy najwyższemu priorytetowi zadań wołających dany obiekt.
- Jeśli zadanie woła dany obiekt, to jego priorytet jest natychmiast podnoszony do poziomu priorytetu chronionego obiektu.
- Jeśli zadanie zamierzające wejść do obiektu chronionego jest wykonywane, to obiekt nie może być już zajęty przez inny proces.
- Każdy obiekt chroniony ma przydzielany priorytet z zastosowaniem pragmy.
- Jeśli nie zastosowano pragmy, to zakładany jest priorytet Priority'Last.
- Zgłaszany jest wyjątek Program_Error jeśli aktywny priorytet zadania jest wyższy niż pułap.
- Jeśli do chronionej operacji przywiązana jest obsługa przerwania i ustalony został błędny pułap priorytetu program będzie działał błędnie.

Ada – protokoły przydziału zasobów – przykład

```
protected Gate_Control is
   pragma Priority(28);
   entry Stop_And_Close;
   procedure Open;
private
   Gate : Boolean := False;
end Gate_Control;
```

```
protected body Gate_Control is
   entry Stop_And_Close
            when Gate is
   begin
      Gate := False;
   end;
   procedure Open is
   begin
      Gate := True;
   end;
end Gate_Control;
```

Ada – aktywne priorytety

- **Zadanie uzyskujące dostęp do obiektu chronionego dziedziczy jego priorytet. (bazowy priorytet może być czasowo zwiększany!)**
- **Dodatkowo bazowy priorytet zadania może ulec zmianie podczas:**
  - **Aktywacji zadania – zadanie dziedziczy aktywny priorytet od zadania rodzica (jeśli jest wyższy); jest to konieczne, gdyż zadanie rodzica jest blokowane do zakończenia aktywacji potomka i brak dziedziczenia priorytetu stwarzałby niebezpieczeństwo inwersji priorytetów.**
  - **Podczas spotkania – w trakcie wykonywania instrukcji *accept* serwer dziedziczy priorytet zadania wołającego, jeśli ten priorytet jest wyższy – w przeciwnym wypadku pozostaje priorytet serwera.**

Ada – reguły szeregowania zadań 1

- Kolejność wysyłania zadań do wykonywania zdeterminowana jest ich aktualnym aktywnym priorytetem.
- Domyślnie szeregowanie jest zgodne z priorytetami i dopuszcza się wywłaszczanie zadań.
- Nie wiadomo właściwie, co to oznacza w systemach wieloprocesorowych.
- Zdefiniowane w aneksie D algorytm szeregowania zadań, to
  - `FIFO_Within_Priority --(Ada95)`
  - `Non_Preemptive_FIFO_Within_Priorities --(Ada2005)`
  - `Round_Robin_Within_Priorities --(Ada2005)`
  - `EDF_Across_Priorities--(Ada2005)`

Ada – reguły szeregowania zadań 2

- `FIFO_Within_Priority`
  - Zadania są szeregowane zgodnie z ich priorytetami, ustawiając zadania o tym samym priorytecie w kolejce FIFO.
- `Non_Preemptive_FIFO_Within_Priorities`
  - Na danym poziomie priorytetu zadanie wykonuje się do końca lub do momentu zablokowania (na zasobie) lub do wykonania instrukcji delay. Zadanie nie może zostać przełączone przez inne zadanie o wyższym priorytecie.
- `Round_Robin_Within_Priorities`
  - Na każdym poziomie priorytetu zadania wykonywane są w ustalonych odcinkach czasowych, a potem przełączane. Okres przełączania może być ustalony. Dostępna jest procedura:

```
procedure Set_Quantum (Pri     : in System.Priority;
                       Quantum : in Ada.Real_Time.Time_Span);
```

- `EDF_Across_Priorities`
  - Możliwe jest szeregowanie zgodne z EDF. Najogólniej w pewnym zakresie priorytetów dla każdego zadania określany jest termin ostatecznego zakończenia obliczeń (deadline) i zadanie z najmniejszą wartością deadline jest wybierane pierwsze do wykonywania.

Ada – reguły szeregowania zadań 3

- **Do definiowania reguły szeregowania można się posłużyć *pragmą*:**

```
pragma Task_Dispatching_Policy(policy_identifier);
```

```
Lub:
   pragma Priority_Specific_Dispatching (
                  policy_identifier,
                  first_priority_expression,
                  last_priority_expression);
```

- **W przypadku stosowania opcji**

  `FIFO_Within_Priority,`
  `Non_Preemptive_FIFO_Within_Priorities,`
  `Round_Robin_Within_Priorities`

  można równocześnie aktywować opcję `Ceiling_Locking`**.**

- **Dla zadań szeregowanych z zastosowaniem algorytmu EDF należy określić względną wartość czasu zakończenia obliczeń za pomocą pragmy:**

```
pragma Relative_Deadline (relative_deadline_expression);
```

Ada – reguły szeregowania zadań 4

- **Przykład 1**
  Ustalenie jednej strategii przydziału priorytetów dla zadań:

```
pragma Task_Dispatching_Policy
        (Round_Robin_Within_Priorities );
```

- **Przykład 2**
  Ustalenie kilku strategii przydziału priorytetów w pewnej puli priorytetów:

```
pragma Priority_Specific_Dispatching (
                Round_Robin_Within_Priorities,1,1);
pragma Priority_Specific_Dispatching (
                EDF_Across_Priorities,2,10);
pragma Priority_Specific_Dispatching (
                FIFO_Within_Priority,11,24);
```

- **Uwaga! Zapis:**

```
pragma Priority_Specific_Dispatching (
                EDF_Across_Priorities,2,5);
pragma Priority_Specific_Dispatching (
                EDF_Across_Priorities,6,10);
```

nie jest równoważny zapisowi:

```
pragma Priority_Specific_Dispatching (
                EDF_Across_Priorities,2,10);
```

Ada – reguły kolejkowania wywołań wejścia obiektu chronionego

- Programista może wybrać regułę kolejkowania wywołań wejścia obiektu chronionego i instrukcji *select*.
- Reguła specyfikowana jest przez *pragmę*:

```
pragma Queuing_Policy(Policy_Identifier);
```

- W aneksie D zdefiniowano 2 predefiniowane reguły:
  - `FIFO_Queuing` (domyślne)
  - `Priority_Queuing`
    - Z zastosowaniem reguły `Priority_Queuing` wybierane jest to wejście, które jest otwarte i żąda do niego dostępu zadanie o najwyższym priorytecie.
    - Jeśli o wywołanie konkurują dwa wejścia, do których powiązane są zadnia o tym samym priorytecie, to wywołane zostanie zadanie, które wykonało wołanie pierwsze.
    - Zadania są kolejkowane według aktywnych priorytetów, jeśli aktywny priorytet ulegnie zmianie nie następuje przekolejkowanie, w przypadku zamiany priorytetu bazowego następuje usunięcie zadania z kolejki i i reorganizacja kolejki.

Ada – wymuszanie analizy szeregowania

```
-- (1) wymuszenie szeregowania po każdej obsłudze
task body TT is
begin
  loop
       ... -- instrukcje obsługi
       delay 0.01;
  end loop;
end TT;
-- (2) wymuszenie szeregowania po każdej iteracji
  ...
  for I in A'range loop
            Treatment(I);
            delay 0.01;
       end loop;
```

Ada – priorytety dynamiczne

```
with Ada.Task_Identification; use Ada;
package Ada.Dynamic_Priorities is
  procedure Set_Priority(Priority : System.Any_Priority;
    T : Task_Identification.Task_Id :=
    Task_Identification.Current_Task);
  function Get_Priority(T : T_Identification.Task_Id
     := Task_Identification.Current_Task)
     return System.Any_Priority;
     -- raise Tasking_Error if task has terminated
  -- Both raise Program_Error if a Null_Task_Id is passed
private
  -- not specified by the language
end Ada.Dynamic_Priorities;
```

Przykład – podwójne priorytetowanie (1)

```
pragma Task_Dispatching_Policy(FIFO_Within_Priorities);
pragma Locking_Policy(Ceiling_Locking);

with Ada.Text_IO; use Ada.Text_IO;
with ada.integer_text_io; use ada.integer_text_io;
with Ada.Numerics.Elementary_Functions;
use Ada.Numerics.Elementary_Functions;
with Calendar; use Calendar;
with System;
with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Dynamic_Priorities; use Ada.Dynamic_Priorities;

procedure Main is
   -- Definicja priorytetów zadań:
   Low: constant System.priority := 20;
   Mid: constant System.Priority := 23;
   High: constant System.Priority := 25;
   Very_High: constant System.Priority :=28
```

Przykład – podwójne priorytetowanie (2)

```
-- Deklaracje zadań:

task Minder is
   entry Register(Start: Time);
   pragma Priority(Very_High);
end Minder;

task Example_Hard is
   pragma Priority(High);
end Example_Hard;

task Example_Soft is
   pragma Priority(Mid);
end Example_Soft;
```

Przykład – podwójne priorytetowanie (3)

```
task body Example_Hard is
   Start_Time : Time := Clock;      Stop_Time : Time;
   Cycle : Duration;                Period : Duration := 4.0;
   Dana: Float;
begin
   Minder.Register(Start_Time);     -- Zarejestruj zadanie
   loop
        for I in 1..32000 loop -- Obliczenia
                    for J in 1..3000 loop Dana:=Sqrt(Float(J));
                    end loop;
        end loop;
        Stop_Time := Clock;
        Cycle:=Stop_Time-Start_Time; -- Wyznacz czas obliczeń
        Put("Hard: "); Put(Integer(Cycle)); New_Line;
        Start_Time := Start_Time + Period;
        Set_Priority(Low); -- Zmniejsz swój priorytet
   delay until Start_Time;
   end loop;
end Example_Hard;
```

Przykład – podwójne priorytetowanie (4)

```
task body Minder is
   Start_Time : Time;
   Period : Duration := 4.0;
   Offset : Duration := 2.0;
   Id : Task_Id;
begin
   accept Register(Start : Time) do
        Id := Register'Caller; -- przejmij ID zadania wołającego
        Start_Time := Start;
        Set_Priority(Low,Id); -- obniż priorytet zadania
                              -- wołającego
   end Register;
   Start_Time := Start_Time + Offset;
   loop
        delay Until Start_time;
        Set_Priority(High,Id); -- podnieś priorytet zadania
        Start_Time:=Start_time+Period;
   end loop;
end Minder;
```

Przykład – podwójne priorytetowanie (5)

```
task body Example_Soft is
   Start_Time : Time:= Clock;        Stop_Time : time;
   Cycle : Duration;                 Period : Duration := 6.0;
   Deadline : Duration := 5.0;       Dana: Float;
begin
   loop
        select delay until Start_Time + Deadline;
               Put("Zadanie Soft nie dokonczylo obliczen");
        then abort
               for I in 1..32000 loop
                      for J in 1..4000 loop Dana:=Sqrt(Float(J));
                      end loop;
               end loop;
               Stop_Time := Clock; Cycle:=Stop_Time-Start_Time;
               Put("Soft: "); Put(Integer(Cycle)); New_Line;
        end select;
        Start_Time:=Start_Time + Period;
        delay until Start_Time;
   end loop;
end Example_Soft;                              begin null;end Main;
```