Ada – pakiety – ogólna struktura

```
-- Specyfikacja pakietu:
package Nazwa is
    [Deklaracje_Stałych]
    [Deklaracje_Typów]
    [Deklaracje_Zmiennych]
    [Specyfikacje_Podprogramów]
[private
    [Deklaracje_Stałych]
    [Deklaracje_Typów]
]
end Nazwa;

-- Treść pakietu:
package body Nazwa is
    [Wewnętrzne_Deklaracje]
begin
    [Instrukcje_Inicjalizujące_Pakiet]
end Nazwa;
```

Ada – pakiety – przykład

```
-- Specyfikacja pakietu (plik stack.ads):
package Stack is
    procedure Push(X:Integer);
    function Pop return Integer;
end Stack;
```

Ada – pakiety – przykład

```
-- Treść pakietu (plik stack.adb):
package body Stack is
    size: constant := 100;
    A: array(1..Size) of Integer;
    Top: Integer range 0..Size;
    procedure Push(X: Integer) is
    begin
        If Top = size then return;
        end if;
        Top:=Top + 1;
        A(Top):=X;
    end Push;
    function Pop return Integer is
    begin
        Top:= Top - 1;
        return A(Top+1);
    end Pop;
begin
    top:=0; -- nadanie wartości poczatkowej
end Stack;
```

Ada – pakiety – przykład

```
-- Zastosowanie pakietu (plik main.adb):

with Ada.Text_IO; use Ada.Text_IO;
with ada.integer_text_io; use ada.integer_text_io;
with Ada.Float_Text_IO;  use Ada.Float_Text_IO;

with Stack; use Stack;

procedure main is
M: Integer :=12;
L: Integer :=125;

begin
    Push(M);
    Push(L);
    Put(Pop);
    Put(pop);
end Main;
```

Ada – wskaźniki

```
-- Typy wskaźnikowe alokowane w specjalnej puli pamięci (heap):

type Item;                              -- niepełna deklaracja typu
type Item_Ptr is acces Item;            -- wskaźnik do typu

type Item is                        -- element listy jednokierunkowej
   record
      Value: Integer;
      Next: Item_Ptr;
   end record;
```

Ada – wskaźniki

```
-- Podstawowe operacje na wskaźnikach:

I1: Item_Ptr: new Item'(10,null); -- powołanie i inicjalizacja
                                  -- nowego obiektu
I2: Item_Ptr;

I2:= new Item'(5,I1);             -- dołączenie elementu listy

I1:=I2;                           -- kopiowanie wskaźników (oba
                                  -- pokazują na ten sam obiekt)
I1.Value:=I2.Value;               -- kopiowanie pól
I1.Next :=I2.Next;

I1.all  :=I2.all;                 -- Kopiowanie wszystkich pól

I1 = I2                           -- Sprawdzenie czy oba
                                  -- wskaźniki pokazują na
                                  -- ten sam obiekt

I.all = I2.all                    -- Porównanie, czy wszystkie
                                  -- pola rekordów są równe
```

Programowanie współbieżne

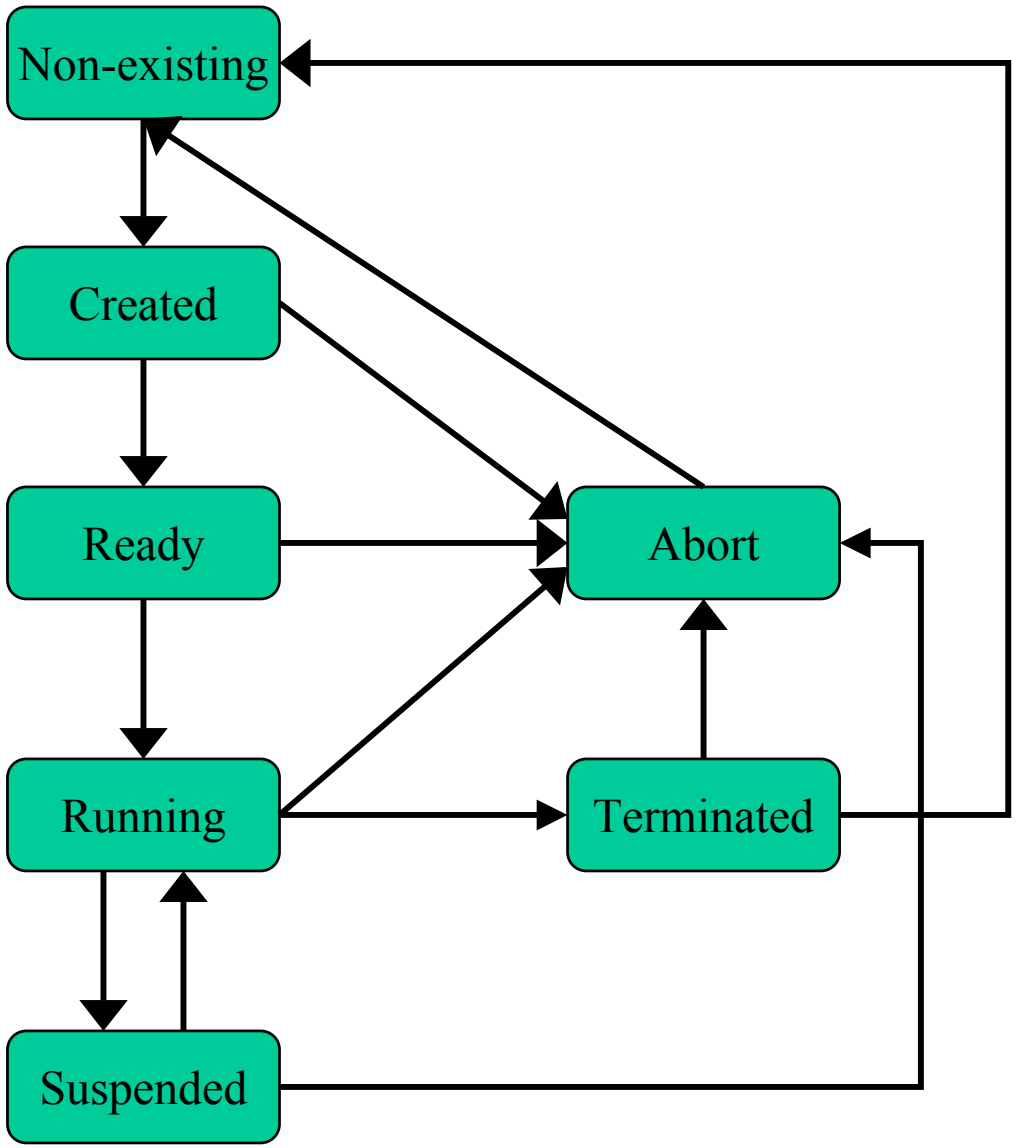
- **System czasu rzeczywistego powinien mieć możliwość współbieżnej obsługi zdarzeń ponieważ zwykle jego otoczenie składa się z wielu obiektów, które działają współbieżnie i generują odpowiednie zdarzenia wymagające obsługi. Wymagania jednoczesnej obsługi narzucają współbieżną strukturę systemu.**
- **Własność współbieżności istotnie zmienia podejście do projektowania i programowania. Pojawia się tu eksplozja kombinatoryczna możliwych zachowań procesów współbieżnych uniemożliwiająca pokrycie testami wszystkich stanów systemu.**

Programowanie współbieżne dotyczy notacji i technik programowania umożliwiających specyfikację potencjalnej równoległości oraz rozwiązywanie zagadnień synchronizacji i komunikacji.

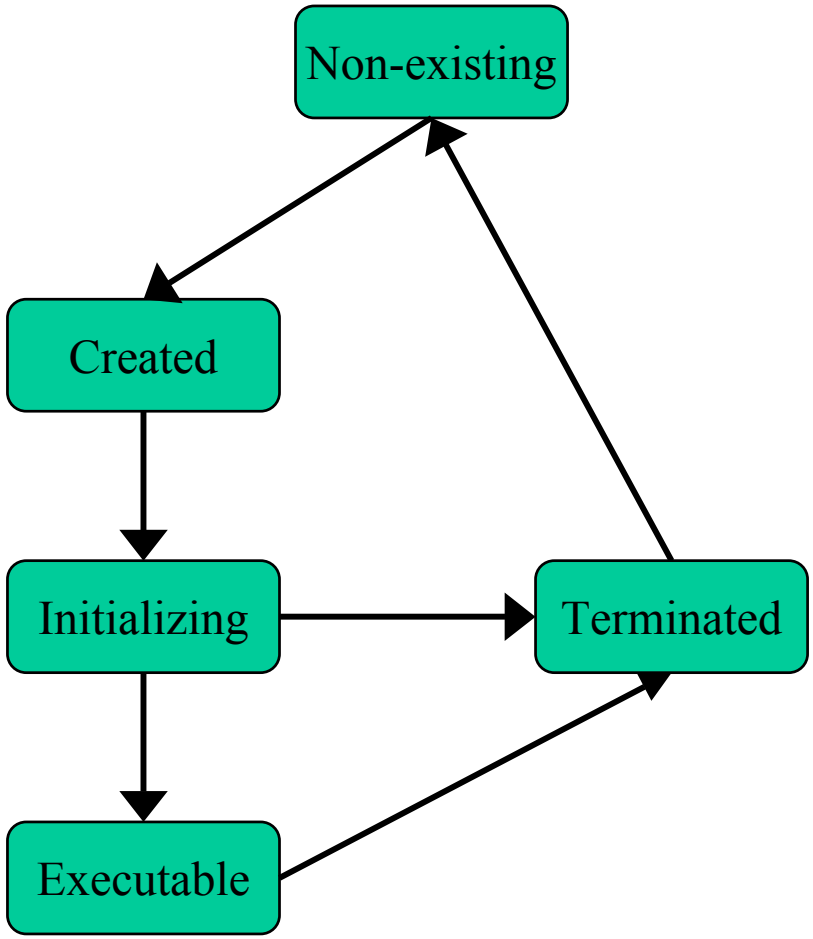
Proces, współbieżność

- **Proces jest pewnym obiektem dynamicznym, charakteryzującym się zbiorem stanów i pewną relacją (funkcją) określającą następstwo stanów.**
- **Dowolne dwa procesy nazywamy współbieżnymi, jeśli jeden z nich może rozpocząć się przed zakończeniem drugiego**

Uproszczony diagram stanów procesu



Fazy zadania w programowaniu współbieżnym



Programowanie współbieżności

- **Konstrukcje umożliwiające programowanie współbieżności:**

1. **Deklarowanie procesów**
2. **Współprogramy (*corutines*)**
3. **Mechanizm *fork-join***
4. **Konstrukcja *cobegin-coend***
5. **Wątki w standardzie POSIX**

Programowanie współbieżności

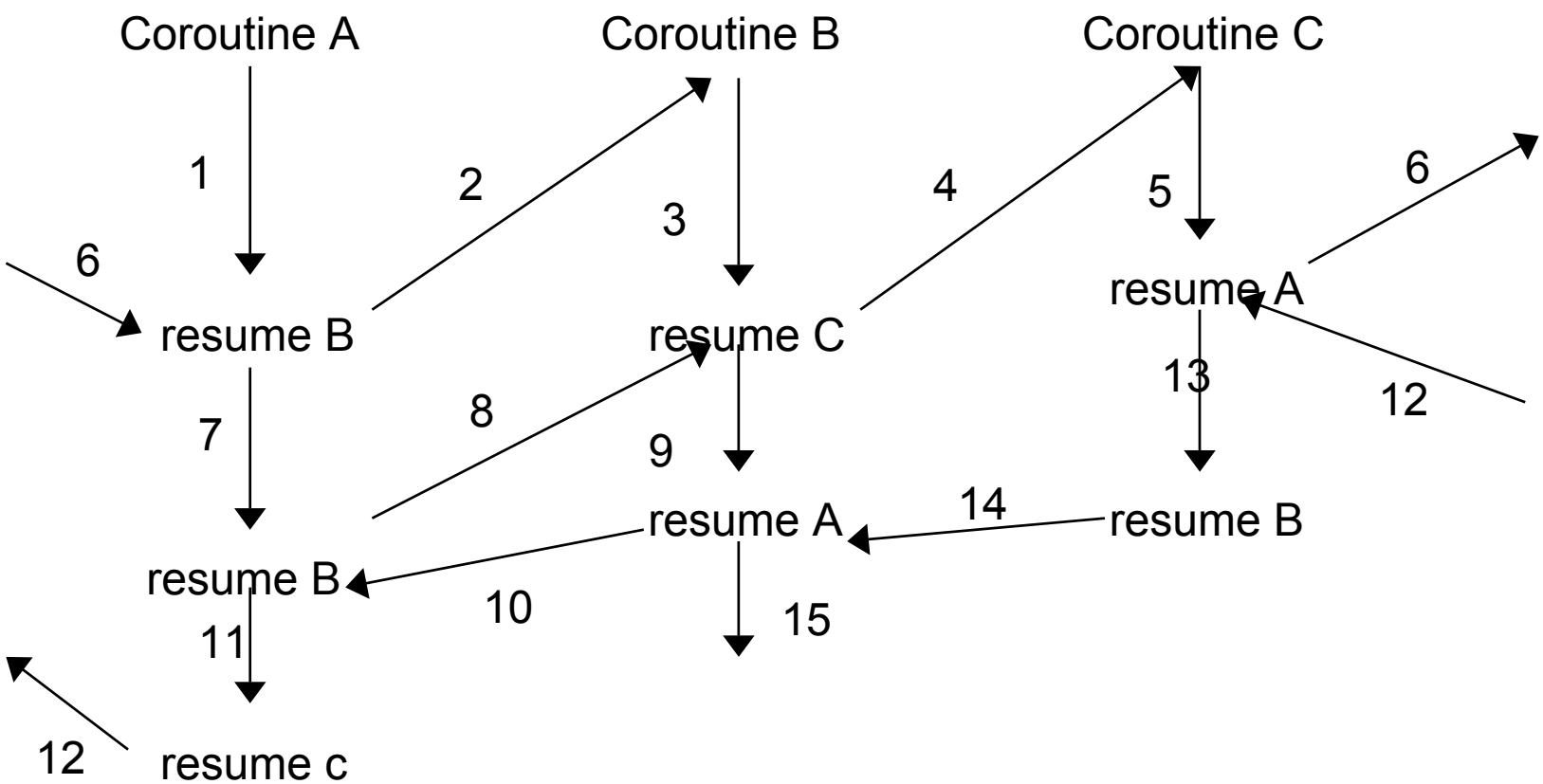
1. **Deklarowanie procesów**

   Istnieje możliwość zdefiniowania sekwencyjnych fragmentów programu (np. procedur/funkcji) w postaci procesów, dla których określa się, czy mogą być wykonywane współbieżnie.

Programowanie współbieżności

2. Współprogramy (*corutines*)

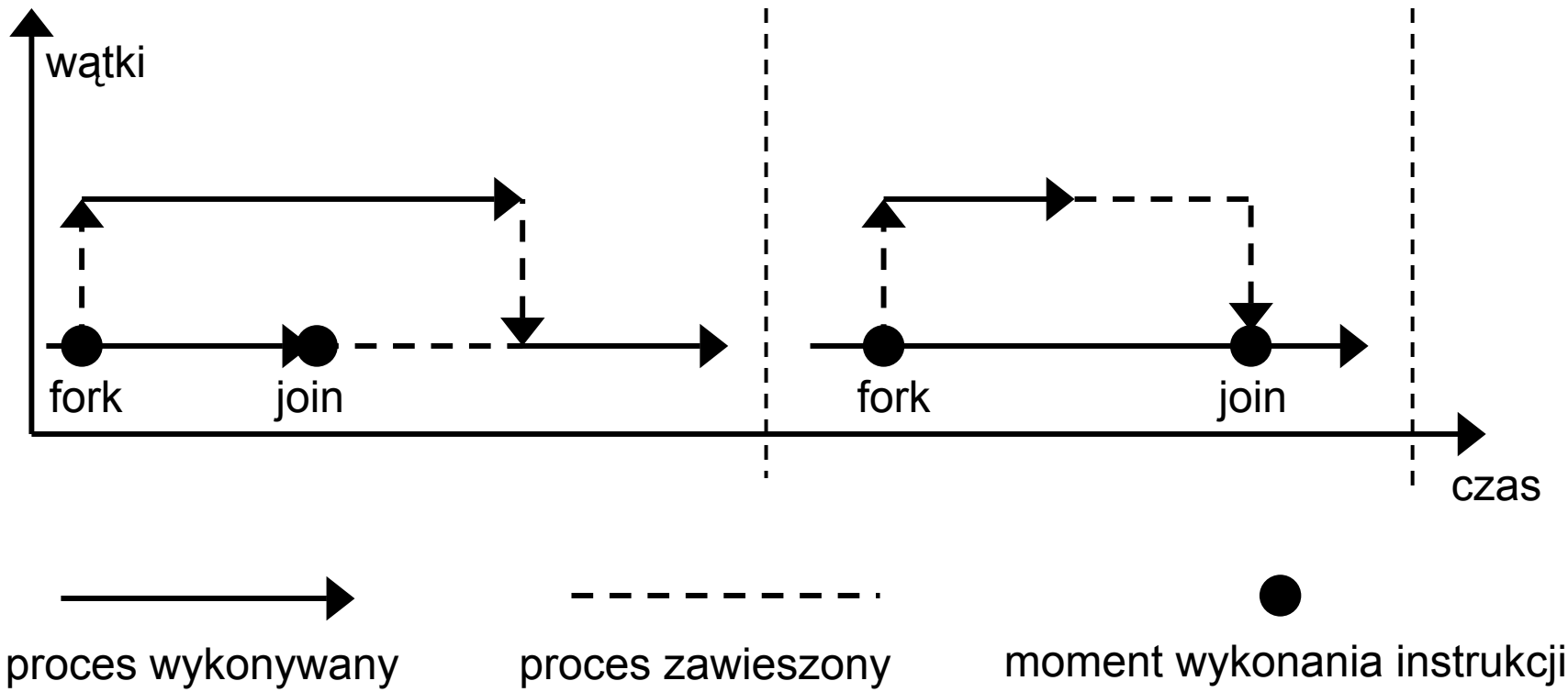
umożliwiają naprzemienne wykonywanie rożnych wątków. Podstawowa zasadą jest tu symetryczny mechanizm wywołania i powrotu miedzy współprogramami.



Programowanie współbieżności

3. Mechanizm *fork-join*

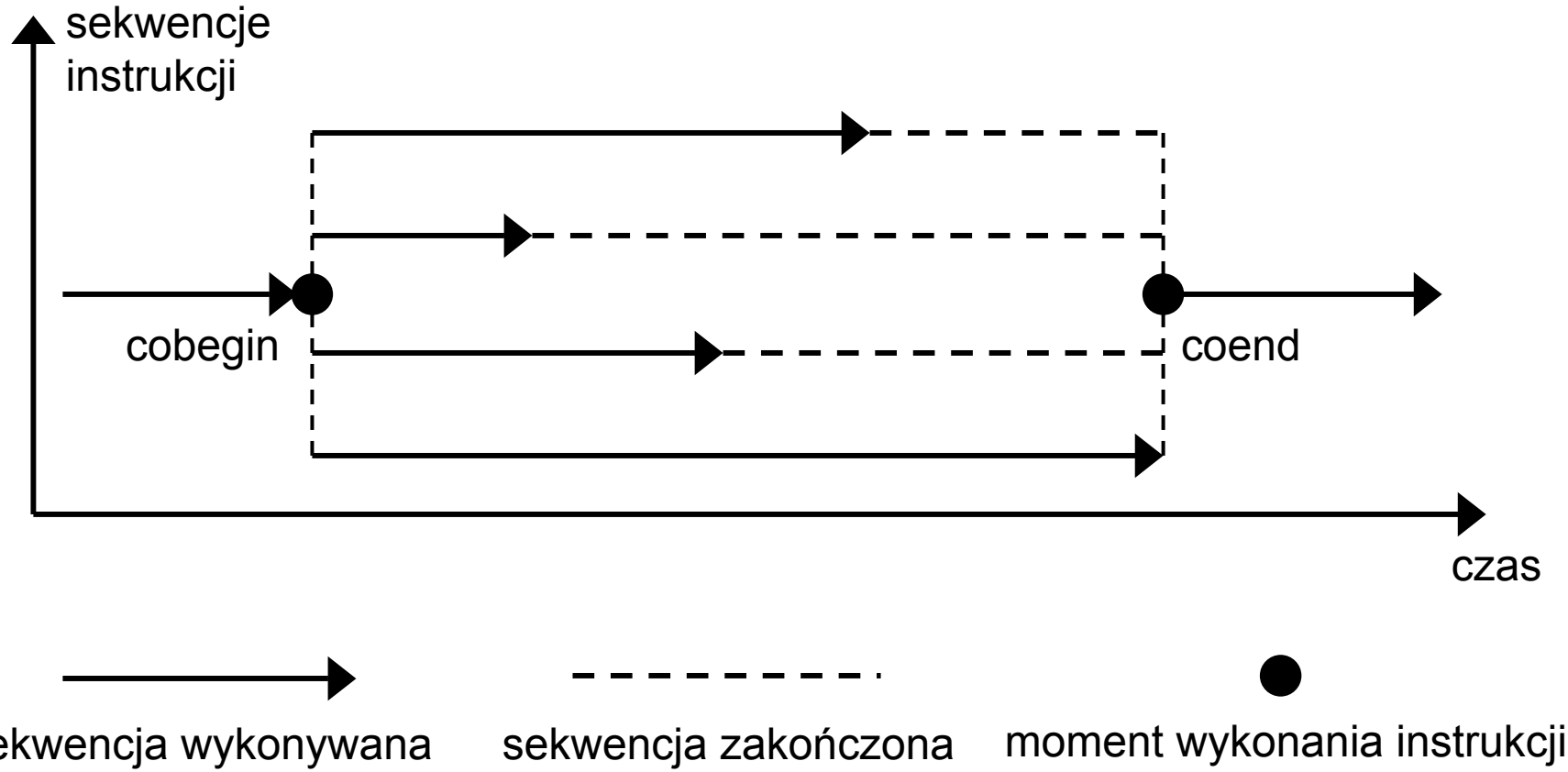
instrukcja *fork* powoduje współbieżną aktywację podprogramu. Instrukcja *join* umożliwia procesowi wykonującemu *fork* synchronizację z zakończeniem wykonywania współbieżnie wykonywanego podprogramu.



Programowanie współbieżności

4. Konstrukcja *cobegin-coend*

umożliwia specyfikację współbieżnie aktywowanych sekwencji instrukcji. Zakończenie następuje, gdy wszystkie zostaną wykonane.



Programowanie współbieżności

5. Wątki w standardzie POSIX

- **Klasyczny mechanizm systemu UNIX *fork-wait,***
- **Wielowątkowe procesy powoływane i usuwane przez odpowiednie funkcje interfejsu.**

Zadania w języku Ada

- **Podstawowym mechanizmem programowania współbieżności w języku Ada są zadania,**
- **Zadanie jest interpretowane jako proces, który może się wykonywać współbieżnie z innymi procesami (zadaniami),**
- **Każde zadanie może zawierać inne zadania kreowane statycznie lub dynamicznie (możliwość tworzenia hierarchii zadań),**
- **Zadania są aktywizowane po zakończeniu opracowywania ich deklaracji,**
- **Usunięcie zadania jest realizowane, jeśli:**
  - **Istnieją przesłanki do usunięcia (zadanie wykonało ostatnią instrukcję lub osiągnęło instrukcję *terminate*, lub wykonano na nim instrukcję *abort*),**
  - **Spełnione są warunki do usunięcia zadania, bez utraty spójności programu (wszystkie zadania zależne są usunięte lub oczekują na usunięcie).**

Zadania w języku Ada – przykład wstępny

```
procedure Example1 is -- zadanie może być definiowane
                      -- w procedurze lub pakiecie lub zadaniu
   task type A_Type;  -- zdefiniowanie typu zadaniowego
   task type B_Type;
   A: A_Type;         -- utworzenie zadań
   B: B_Type;

   task body A is     -- lokalne deklaracje dla zadania A
   begin              -- sekwencja instrukcji dla zadania A
   end A;

   task body B is     -- lokalne deklaracje dla zadania B
   begin              -- sekwencja instrukcji dla zadania B
   end B;
begin  -- Zadania A i B rozpoczną swoje wykonywanie
       -- przed pierwszą instrukcją sekwencji
       -- instrukcji należących do procedury.
       -- System składa się z 3 (!) współbieżnych procesów:
       -- zadań A i B oraz procedury Example1
end Example1;         -- procedura zakończy się dopiero
                      -- gdy zakończą działanie oba zadania
```

Zadania w języku Ada – przegląd możliwości tworzenia

Zadania w języku Ada mogą być:

- **Zgrupowane w tablice**
- **Składnikami rekordów**
- **Powoływane dynamicznie**
- **Składnikami innych zadań (można tworzyć hierarchię zadań)**

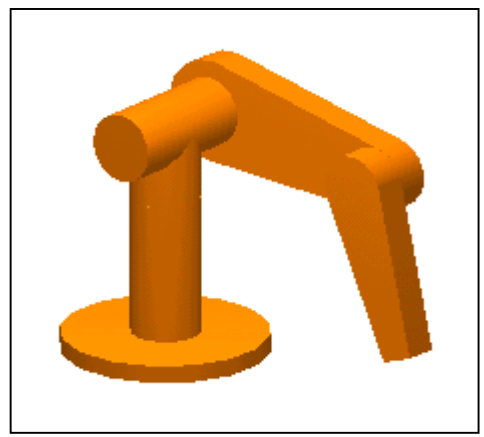
Zadania w języku Ada – fazy wykonywania

- **Aktywacja – wykonywana w trakcie tzw. Opracowywania częsci deklaracyjnej zadania; w wyniku aktywacji zadanie wraz z danymi lokalnymi zostają utowrzone oraz zadanie jest inicjowane.**
- **Wykonywanie – wykonywane są poszczególne instrukcje zapisane we wnętrzu zadania.**
- **Zakończenie**
  - **Osiągnięcie ostatniej instrukcji lub**
  - **Zgłoszenie wyjątku, który nie może być obsłużony lub**
  - **Wykonanie instrukcji *terminate* w rozgałęzieniu *select* (dalsze wykłady) lub**
  - **Wykonanie na zadaniu instrukcji abort.**

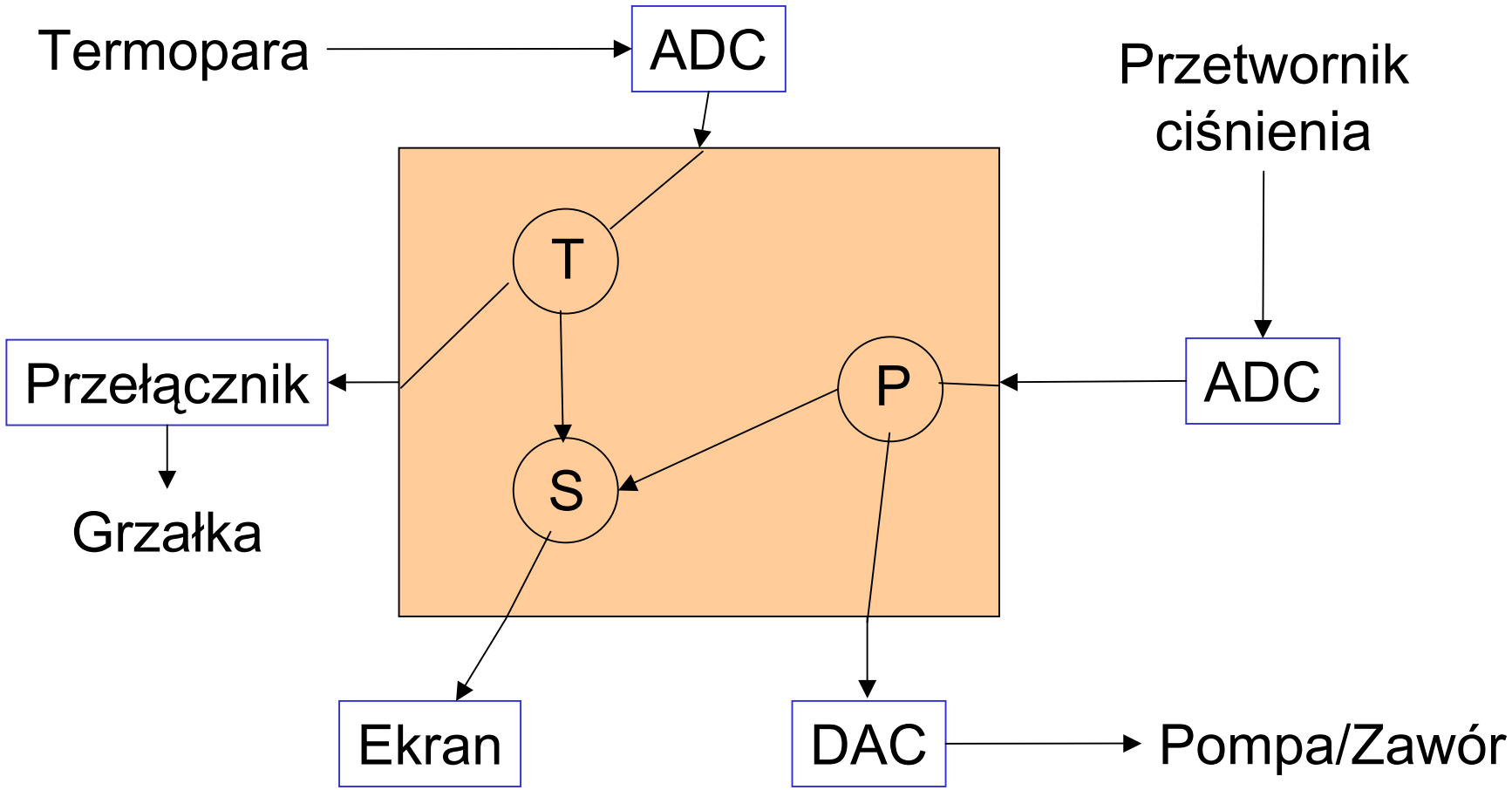
(Usunięcie zadania jest możliwe jedynie, w warunkach, gdy nie zaburzy to spójności aplikacji, tzn. jeśli wszystkie inne zadania są usunięte, lub oczekują na usunięcie)

Zadania w języku Ada – przykład: szkielet sterownika robota

```
procedure main is
    type dimension is (os1, os2, os3);
    task type control(dim : dimension); -- parametr wywołania!
    procedure move_arm(D: dimension; P: Integer) is
        begin null; end;
    procedure new_setting(D: dimension; P: Integer) is
        begin null; end;
    task body control is
    position : integer;      -- bezwzględna pozycja
    setting : integer;       -- względne przemieszczenie
    begin
        position := 0;           -- resetuj pozycję
        loop   move_arm(dim, position);
               new_setting(dim, setting);
               position := position + setting;
        end loop;
    end control;
    C1 : control(os1);
    C2 : control(os2);
    C3 : control(os3);
begin  null; end main;
```



Zadania w języku Ada – przykład: prosty system wbudowany



- **Generalny cel, to utrzymanie temperatury i ciśnienia w zadanych przedziałach i przekazywanie bieżącego stanu zmieżonych wielkości na ekran.**

Prosty system wbudowany – możliwe architektury oprogramowania

- **Można stworzyć pojedynczy program ignorując logiczną współbieżność procesów T, P i S. Nie jest wtedy wymagane wsparcie systemu operacyjnego (Można zaproponować aplikację dla mikrokontrolera jednoukładowego z podłączonymi odpowiedniki peryferiami).**
- **T, P i S mogą być sekwencyjnymi programami lub procedurami i można zastosować składniki systemu operacyjnego do stworzenia współbieżnych procesów i określenia zależności miedzy nimi.**
- **Można utworzyć pojedynczy współbieżny program zachowujący strukturę obiektów T, P i S. Nie trzeba znać wtedy funkcji systemu operacyjnego; wymagane będzie odpowiednie środowisko programowe do uruchomienia aplikacji (dołączony do systemu operacyjnego podsystem uruchomieniowy)**

Które podejście zastosować ?

Prosty system wbudowany – zdefiniowanie pakietów pomocniczych

```
package Data_Types is
   type Temp_Reading is new Integer range 10..500;
   type Pressure_Reading is new Integer range 0..750;
   type Heater_Setting is (On, Off);
   type Pressure_Setting is new Integer range 0..9;
end Data_Types;
```

Potrzebne typy danych

```
with Data_Types; use Data_Types;
package IO is
   procedure Read(TR : out Temp_Reading); -- z ADC
   procedure Read(PR : out Pressure_Reading);
   procedure Write(HS : Heater_Setting);-- do przeł.
   procedure Write(PS : Pressure_Setting); -- do DAC
   procedure Write(TR : Temp_Reading);  -- do Ekranu
   procedure Write(PR : Pressure_Reading);-- do Ekranu
end IO;
```

Procedury wymiany danych z otoczeniem

Prosty system wbudowany – procedury sterujące

```
with Data_Types; use Data_Types;
package Control_Procedures is
   -- procedury odczytujące stan otoczenia
   -- i generujące odpowiednie ustawienia.
   procedure Temp_Convert(TR : Temp_Reading;
                          HS : out Heater_Setting);
   procedure Pressure_Convert(PR : Pressure_Reading;
                              PS : out Pressure_Setting);
end Control_Procedures;
```

Prosty system wbudowany – ROZWIĄZANIE SEKWENCYJNE

```
with Data_Types; use Data_Types; with IO; use IO;
with Control_Procedures;  use Control_Procedures;

procedure Controller is
   TR : Temp_Reading;
   PR : Pressure_Reading;
   HS : Heater_Setting;
   PS : Pressure_Setting;
begin
   loop
      Read(TR);     -- z ADC
      Temp_Convert(TR,HS);
      Write(HS);    -- do przełącznika
      Write(TR);    -- na ekran
      Read(PR);
      Pressure_Convert(PR,PS);
      Write(PS);
      Write(PR);
   end loop; -- nieskończona pętla
end Controller;
```

Nie jest wymagany system operacyjny

Prosty system wbudowany – ROZWIĄZANIE SEKWENCYJNE - DYSKUSJA

- **Odczyt temperatury i ciśnienia następuje z jednakową częstotliwością**
- **Zastosowanie liczników i instrukcji warunkowych może poprawić sytuację**
- **Może zaistnieć potrzeba rozdzielanie procedur konwertujących Temp_Convert i Pressure_Convert, aby umożliwić niezależne wykonywanie obliczeń i odpowiednie rozłożenie prac**
- **W czasie czekania na odczyt temperatury nie można przekazać uwagi oprogramowania na ciśnienie (i na odwrót)**
- **Uszkodzenie np. kanału odczytu temperatury powodujące zawieszenie procedury odczytującej Dead spowoduje równocześnie blokadę możliwości odczytu stanu ciśnienia.**

Prosty system wbudowany – POPRAWIONE ROZWIĄZANIE SEKWENCYJNE

```
with Data_Types; use Data_Types; with IO; use IO;
with Control_Procedures; use Control_Procedures;
procedure Controller is
  TR : Temp_Reading;    PR : Pressure_Reading;
  HS : Heater_Setting; PS : Pressure_Setting;
  Ready_Temp, Ready_Pres : Boolean;
begin
  loop
    if Ready_Temp then
      Read(TR); Temp_Convert(TR,HS);
      Write(HS);  Write(TR);
    end if;
    if Ready_Pres then
      Read(PR); Pressure_Convert(PR,PS);
      Write(PS);  Write(PR);
    end if;
  end loop;
end Controller;
```

Jakie tu są problemy?

Prosty system wbudowany – POPRAWIONE ROZWIĄZANIE SEKWENCYJNE - DYSKUSJA

- **Rozwiązanie bardziej niezawodne**
- **Program spędza większość czasu na odpytywanie urządzeń wejściowych, czy są gotowe**
- **Takie aktywne oczekiwanie zdecydowanie obniża wydajność i w szeregu systemach jest nie do przyjęcia.**
- **Programy o takiej strukturze są trudne do zaprojektowania, zrozumienia i udowodnienia poprawności.**

Głównym zarzutem wobec podejścia sekwencyjnego jest fakt, że program nie odzwierciedla, że podsystem sterowania ciśnieniem i podsystem sterowania temperaturą są niezależne. Mając do dyspozycji środowisko do tworzenia aplikacji współbieznych można by było zaprogramować każdy z tych podsystemów jako zadanie.

Prosty system wbudowany – ROZWIĄZANIE WSPÓŁBIEŻNE Z ZASTOSOWANIEM MECHANZMÓW SYSTEMU OPERACYJNEGO 1

```
package OSI is
   type Thread_ID is private;
   type Thread is access procedure;

   function Create_Thread(Code : Thread)
            return Thread_ID;
   -- other subprograms
   procedure Start(ID : Thread_ID);
private
    type Thread_ID is ...;
end OSI;
```

Prosty system wbudowany – ROZWIĄZANIE WSPÓŁBIEŻNE Z ZASTOSOWANIEM MECHANZMÓW SYSTEMU OPERACYJNEGO 2

```
package Processes is
   procedure Temp_C;
   procedure Pressure_C;
end Processes;

with IO; use IO;
with Control_Procedures; use Control_Procedures;
package body Processes is
   procedure Temp_C is
      TR : Temp_Reading;  HS : Heater_Setting;
   begin
      loop
         Read(TR); Temp_Convert(TR,HS);
         Write(HS); Write(TR);
      end loop;
   end Temp_C;
```

Prosty system wbudowany – ROZWIĄZANIE WSPÓŁBIEŻNE Z ZASTOSOWANIEM MECHANZMÓW SYSTEMU OPERACYJNEGO 3

```
  procedure Pressure_C is
    PR : Pressure_Reading;
    PS : Pressure_Setting;
  begin
    loop
      Read(PR);
      Pressure_Convert(PR,PS);
      Write(PS);
      Write(PR);
    end loop;
  end Pressure_C;
end Processes;
```

Prosty system wbudowany – ROZWIĄZANIE WSPÓŁBIEŻNE Z ZASTOSOWANIEM MECHANZMÓW SYSTEMU OPERACYJNEGO 4

```
with OSI, Processes; use OSI, Processes;
procedure Controller is
   TC, PC : Thread_ID;
begin
    TC := Create_Thread(Temp_C'Access);
    PC := Create_Thread(Pressure_C'Access);
    Start(TC);
    Start(PC);
end Controller;
```

Lepsze, bardziej niezawodne rozwiązanie

Dla realnych SO rozwiązanie może okazać się trudne do odczytu

Prosty system wbudowany – ROZWIĄZANIE WSPÓŁBIEŻNE Z ZASTOSOWANIEM ZADAŃ JĘZYKA ADA

```
with Data_Types; use Data_Types; with IO; use IO;
with Control_Procedures; use Control_Procedures;
procedure Controller is

   task Temp_Controller;
   task body Temp_Controller is
     TR : Temp_Reading;
     HS : Heater_Setting;
   begin
     loop
       Read(TR);
       Temp_Convert(TR,HS);
       Write(HS); Write(TR);
     end loop;
   end Temp_Controller;

begin
   null;
end Controller;
```

```
task Pressure_Controller;
task body Pressure_Controller is
     PR : Pressure_Reading;
     PS : Pressure_Setting;
   begin
     loop
       Read(PR);
       Pressure_Convert(PR,PS);
       Write(PS); Write(PR);
     end loop;
   end Pressure_Controller;
```

Zalety podejścia współbieżnego

- **Zadania sterujące wykonują się współbieżnie w nieskończonych pętlach**
- **Jeśli zadanie oczekuje na odczyt danych, to inne może zostać wykonywane; jeśli oba zadania są zawieszone, to nie jest wykonywana „pusta” pętla**
- **Struktura logiczna aplikacji jest odzwierciedlona w kodzie programu. Naturalna współbieżność zewnętrznych procesów jest odwzorowana w strukturze współbieżnych zadań.**

Wady zaproponowanego podejścia

- **Oba zadania wysyłają dane do wyświetlacza, ale wyświetlacz jest zasobem, do którego tylko jeden proces powinien mieć w danej chwili prawo do zapisu**
- **Wymagane jest powołanie trzeciego elementu systemu rozwiązującego problem współbieżnego dostępu do niewspółbieżnego zasobu**
- **Powinno być również zapewnione, że dane od każdego sterownika zostaną przekazane do ekranu**
- **Ekran musi zapewnić wzajemne wykluczenie dostępu do siebie**
- **Całe podejście wymaga wsparcia od systemu uruchomieniowego.**

Co stosować – współbieżność oferowaną przez SO, czy współbieżność zaszytą w języku programowania?

- **Argumenty za współbieżnością na poziomie języka programowania**
  - **Program jest łatwiejszy do śledzenia i pielegnacji**
  - **Jest wiele systemów operacyjnych i zastosowanie „nadrzędnego" języka jest bardziej przenośne (por. Java)**
  - **W komputerach wbudowanych może nie być systemu operacyjnego, a może istnieć system uruchomieniowy dla języka Ada**
- **Argumenty przeciw współbieżności na poziomie języka programowania**
  - **Prościej jest skomponować programy napisane w różnych językach,jeśli stworzono je pod dany system operacyjny**
  - **Mogą pojawić się trudności w stworzeniu efektywnego współbieżnego środowiska uruchomieniowego ponad warstwą systemu operacyjnego**
  - **Pojawiają się standardy dla systemów operacyjnych**

To co stosować?

PODSUMOWANIE

- **Większość aplikacji czasu rzeczywistego jest z natury współbieżna**
- **Zawarcie procesu/zadania wewnątrz języka programowania czasu rzeczywistego znacząco zmienia podejście do tworzeia aplikacji**
- **Bez współbieżności oprogramowanie czasu rzeczywistego ma zwykle postać pojedynczej pętli sterującej**
- **Struktura takiej pętli nie oddaje realnej struktury zewnętrznego (współbieżnego) środowiska systemu. Trudno jest zweryfikować czasowe i niezawodnościowe wymagania aplikacji bez możliwości skorzystania z pojęcia procesu/zadania.**