Programowanie współbieżne i rozproszone

WYKŁAD 1

Krzysztof Pancerz

Wykonywanie rozkazów

- Rozkazy maszynowe wykonywane są przez procesor sekwencyjnie.
- Rozkazy maszynowe posiadają dostęp do danych przechowywanych w pamięci głównej lub pomocniczej.

Program współbieżny

- **Program współbieżny** (*concurrent program*)– zbiór programów sekwencyjnych, które można wykonywać równolegle.
- **Proces** – program wchodzący w skład programu współbieżnego.
- Termin ***Program*** zarezerwowany jest do oznaczenia całego zbioru procesów programu współbieżnego.
- Procesy wchodzące w skład programu współbieżnego mogą ze sobą współdziałać.

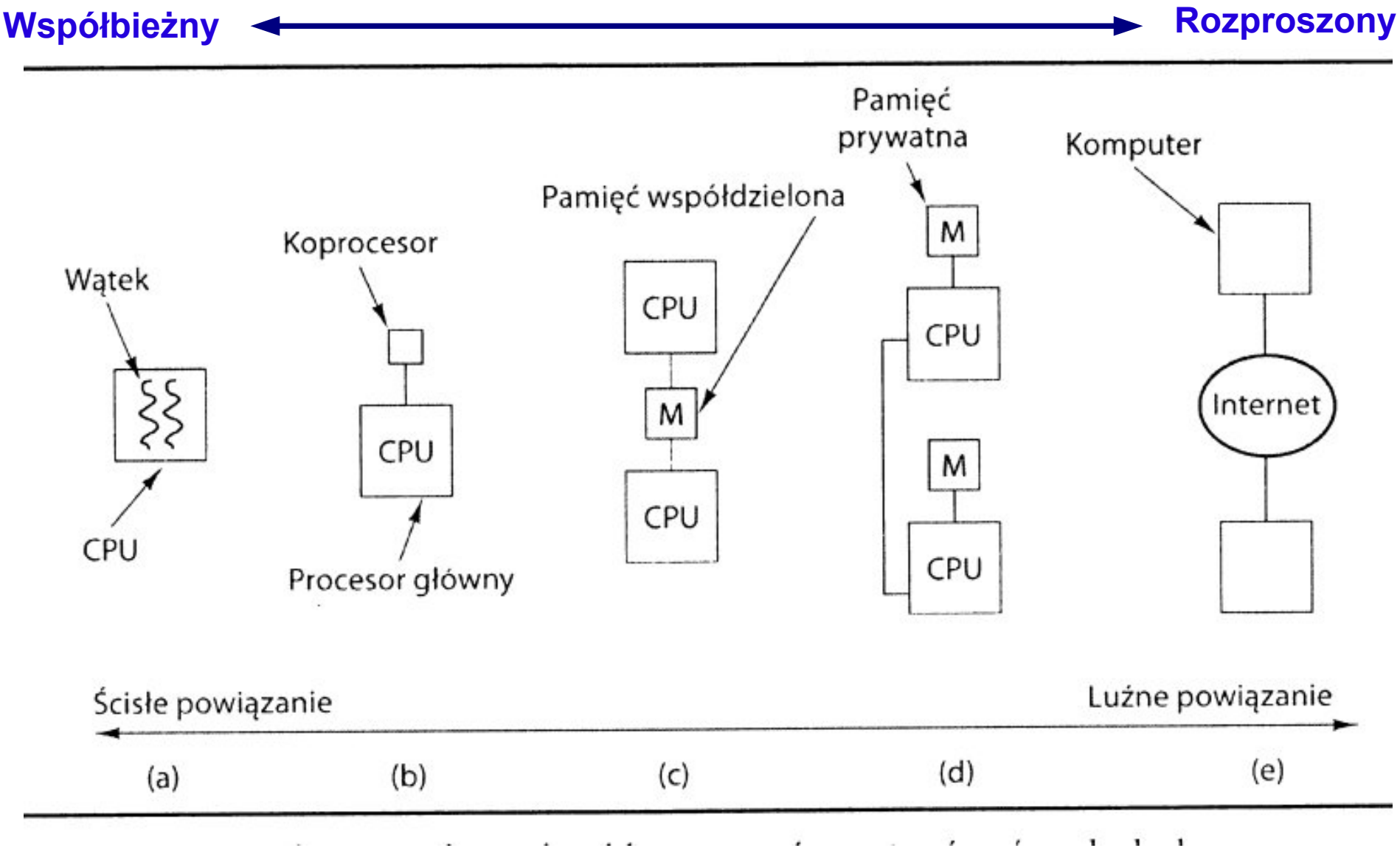
Równoległy, współbieżny, rozproszony

- Przymiotnik ***Równoległy*** stosuje się w odniesieniu do systemów, w których wykonanie wielu programów nakłada się na siebie w czasie (są one wykonywane na różnych procesorach).
- Przymiotnik ***Współbieżny*** oznacza potencjalną równoległość (wykonanie procesów może ale nie musi nakładać się na siebie w czasie).
- Przymiotnik ***Rozproszony*** stosuje się w odniesieniu do systemów, w których wykonanie wielu programów obywa się na rozproszonych procesorach (np. w klastrach obliczeniowych)

Współbieżność

Współbieżność jest bardzo użyteczną abstrakcją, dzięki której można lepiej zrozumieć program, zakładając, że wszystkie procesy wykonują się równolegle.

Powiązania komponentów systemów równoległych



Źródło:A. Tanenbaum „Struktura organizacyjna systemów komputerowych”. Helion, Gliwice, 2006.

Wykonywanie procesów współbieżnych

- Generalnie wykonanie procesów współbieżnych może przebiegać na trzy podstawowe sposoby:

1. Procesy wykonywane są przez pojedynczy procesor rzeczywisty metodą przeplotu.
2. Każdy proces wykonywany jest przez odrębny procesor, przy czym procesory mają dostęp do wspólnej pamięci.
3. Procesy wykonywane są przez odrębne, rozproszone procesory połączone kanałami komunikacyjnymi.

Klasyfikacja Flynna

Klasyfikacja Flynna architektur komputerów opiera się na liczbie strumieni rozkazów i liczbie strumieni danych, które mogą być przetwarzane równolegle.

• **SISD** (*Single Instruction Stream – Single Data Stream*) – pojedynczy strumień rozkazów- pojedynczy strumień danych.

Architektura von Neumanna (sekwencyjnie pobierany jest pojedynczy zestaw rozkazów operujący na pojedynczych danych).

• **MIMD** (*Multiple Instruction Stream – Multiple Data Stream*) – wiele strumieni rozkazów- wiele strumieni danych.

Systemy wieloprocesorowe, w których co najmniej dwa procesory wykonują oddzielne strumienie rozkazów operujące na różnych danych.

Klasyfikacja Flynna

• **SIMD** (*Single Instruction Stream – Multiple Data Stream*) – pojedynczy strumień rozkazów- wiele strumieni danych.

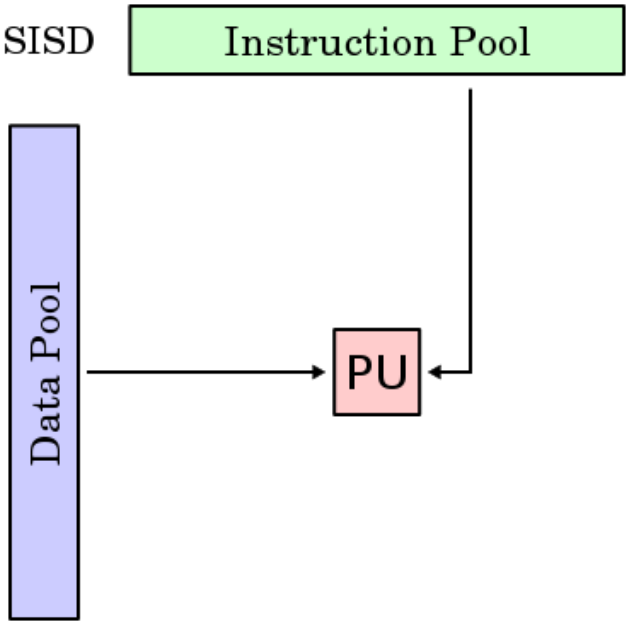
Procesory wektorowe, w których przez jeden procesor sterujący generowany jest pojedynczy strumień rozkazów, przekazywany do wielu oddzielnych procesorów arytmetycznych. Procesory arytmetyczne wykonują jednocześnie na różnych danych te same rozkazy.

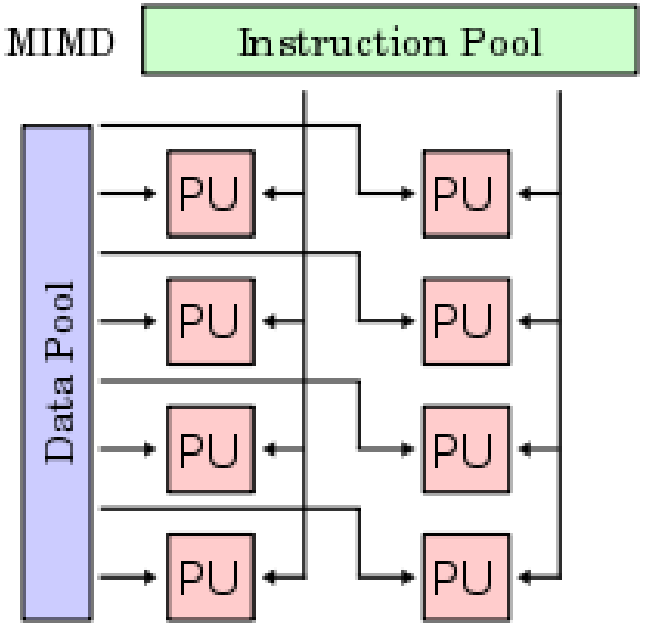
• **MISD** (*Multiple Instruction Stream – Single Data Stream*) – wiele strumieni rozkazów- jeden strumień danych.

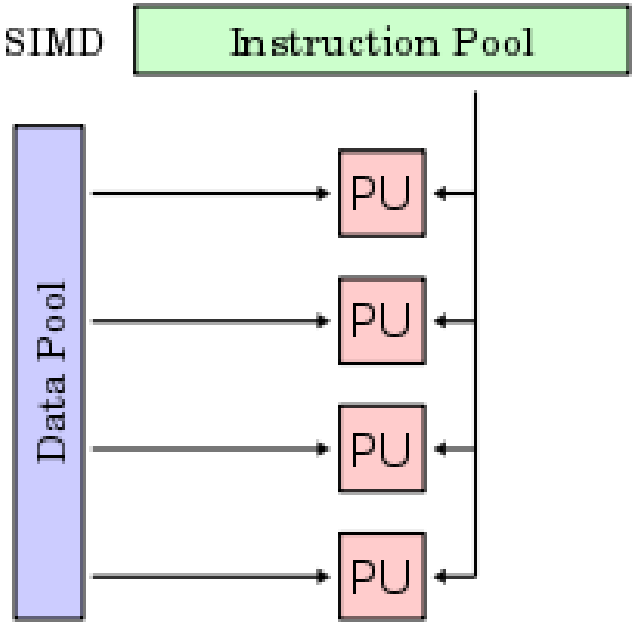
Architektura nie jest wykorzystana w praktyce.

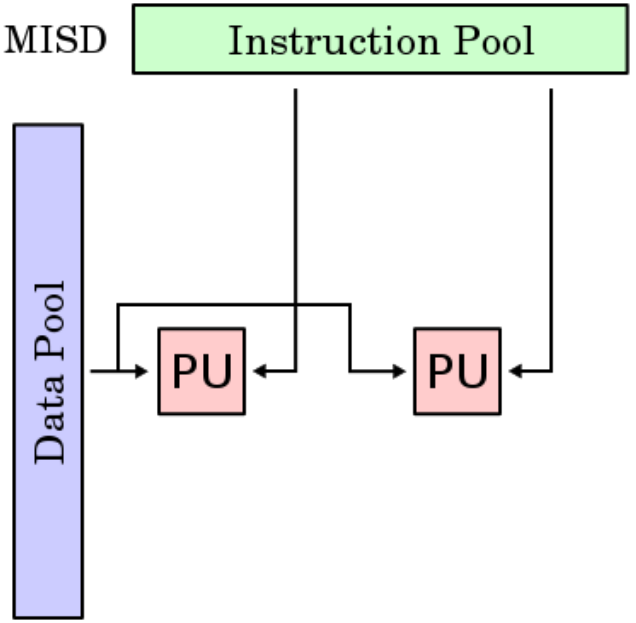
Klasyfikacja Flynna

Źródło:
Wikimedia Commons
author: Colin M.L. Burnett









Przyśpieszenie zadania obliczeniowego

t_j – czas wykonania zadania w systemie jednoprocesorowym.

t_w – czas wykonania zadania w systemie wieloprocesorowym.

$S(n)$ – przyśpieszenie przy użyciu systemu wieloprocesorowego składającego się z n procesorów.

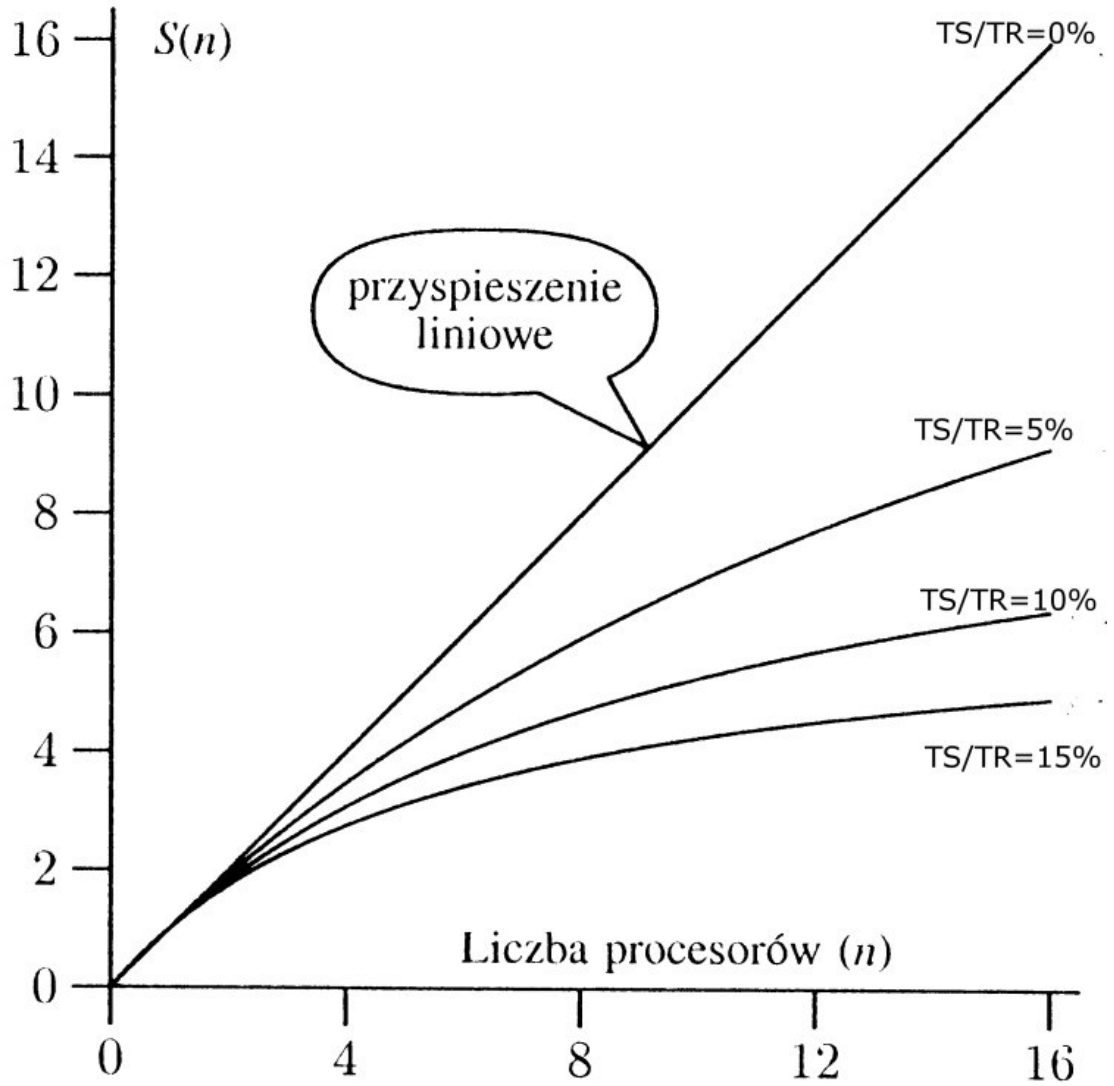
$$S(n) = \frac{t_j}{t_w}$$

Prawo Amdahla

T_S – łączny czas wykonywania ściśle sekwencyjnego kodu danego programu (na jednym procesorze).
T_R – łączny czas wykonywania pozostałego („równoległego”) kodu danego programu (na jednym procesorze).
n – liczba procesorów.
$S(n)$ – przyśpieszenie przy użyciu systemu wieloprocesorowego składającego się z n procesorów.

$$S(n) = \frac{T_S + T_R}{T_S + \frac{T_R}{n}}$$

Prawo Amdahla



Źródło: B.S. Chalk: „Organizacja i architektura komputerów". WNT, Warszawa, 1998.

Procesy i wątki

- W teorii współbieżności używany jest termin ***Proces***.
- W językach programowania spotykany jest termin ***Wątek***.

Procesy i wątki

- **Proces** – wykonujący się program wraz z dynamicznie przydzielonymi mu przez system zasobami.
- **Wątek** – sekwencja działań, która może wykonywać się współbieżnie z innymi sekwencjami działań w kontekście danego procesu (programu).

Procesy i wątki

- **Proces** wykonuje się we własnej przestrzeni adresowej zarządzanej przez system operacyjny.
- **Wątek** wykonuje się w przestrzeni adresowej jednego procesu.

Wykorzystanie wątków

- Wątki umożliwiają programiście tworzenie współbieżnych obliczeń w ramach jednego programu, np. programy interakcyjne zawierają osobny wątek obsługujący zdarzenia związane z interfejsem użytkownika.

Programy wielowątkowe

- W kontekście danego procesu może być wykonywanych wiele wątków. Mówimy wówczas o programie wielowątkowym.

Tworzenie klas wątków

- Sposób 1: Utworzenie klasy wątku, która dziedziczy z klasy **Thread**.
- Sposób 2: Utworzenie klasy wątku, która implementuje interfejs **Runnable**.

Tworzenie i uruchamianie wątku – sposób 1

- Zdefiniować klasę wątku, która dziedziczy z klasy **Thread**.
- Przedefiniować metodę **run**, umieszczając w niej instrukcje, które ma wykonać wątek.
- Stworzyć obiekt utworzonej klasy wątku.
- Wywołać na rzecz utworzonego obiektu wątku metodę **start**.

Tworzenie i uruchamianie wątku – sposób 2

- Zdefiniować klasę wątku, która implementuje interfejs **Runnable**.
- Zdefiniować metodę **run**, umieszczając w niej instrukcje, które ma wykonać wątek.
- Stworzyć obiekt utworzonej klasy wątku.
- Stworzyć obiekt klasy **Thread**, przekazując w konstruktorze referencję do utworzonego obiektu wątku.
- Wywołać na rzecz utworzonego obiektu klasy **Thread** metodę **start**.

Zakończenie pracy wątku

- Sposób naturalny: wątek kończy działanie, gdy kończy się wykonywanie jego metody **run**.
  - Zakończenie metody **run** można uzyskać np. przez sprawdzenie określonego warunku zakończenia pracy wątku i gdy jest on spełniony - wykonanie instrukcji **return**.
- Sposób niezalecany: wywołanie metody **stop** na rzecz obiektu wątku.

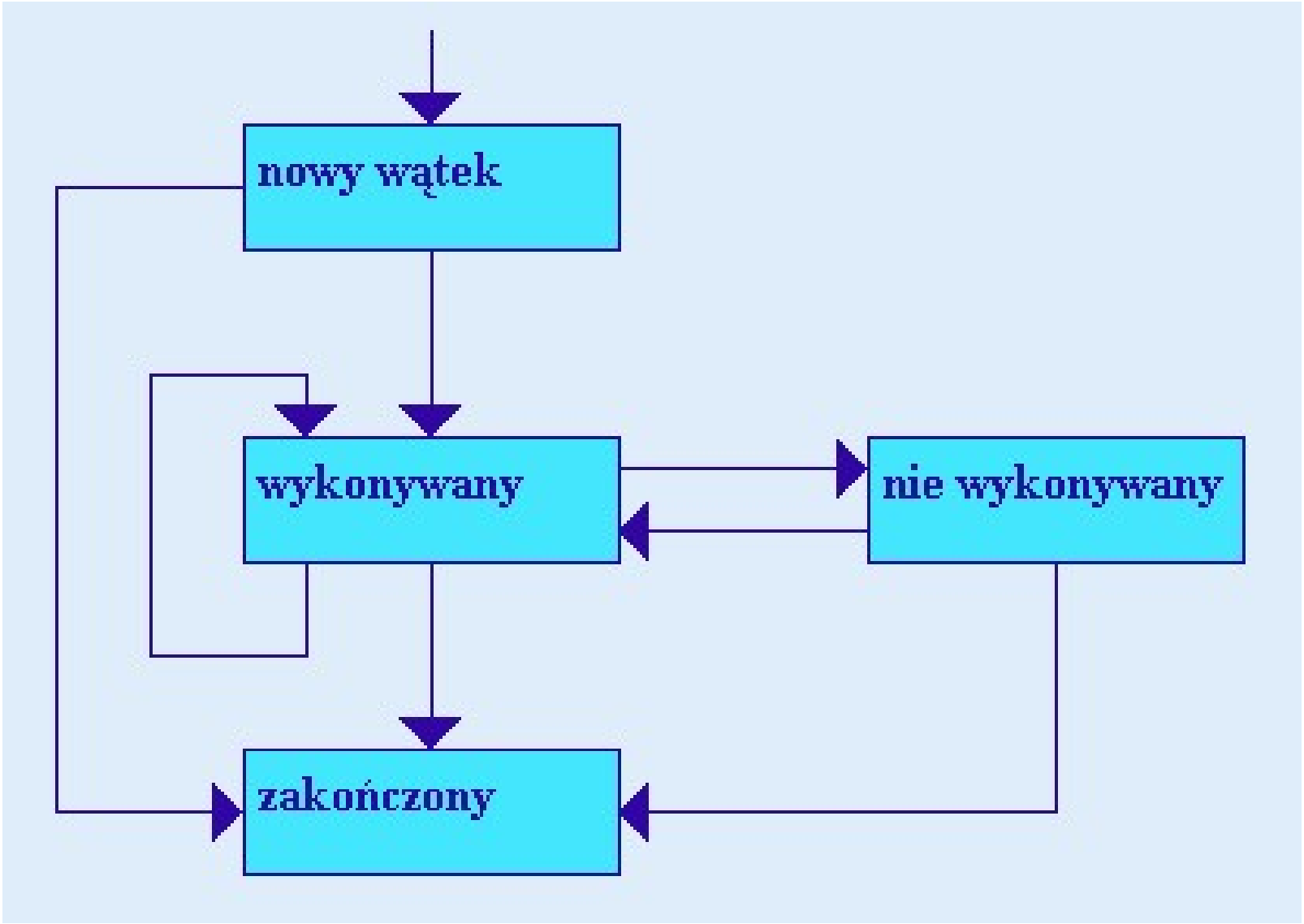
Stany wątku

- **Nowy wątek** (*New Thread*) – stan w momencie stworzenia obiektu wątku.
- **Wykonywany** (*Runnable*) – stan po wywołaniu metody start na rzecz obiektu wątku.
  - Stan ten oznacza potencjalną gotowość wątku do działania, a nie tylko samo wykonywanie wątku. Wątek jest bowiem wykonywany tylko w chwilach czasowych, gdy zostanie mu przydzielony czas procesora.

Stany wątku (cd.)

- **Nie wykonywany** (*Not Runnable*) – stan, do którego wątek przechodzi m.in. na skutek:
  - wywołania metody **sleep**,
  - wywołania metody **wait**,
  - blokowania na operacjach wejścia-wyjścia,
  - blokowania na obiekcie synchronizowanym.

Stany wątku (cd.)



Priorytety wątków

- Każdy wątek posiada priorytet określony przez liczbę naturalną z zakresu od 1 (najniższy priorytet) do 10 (najwyższy priorytet).
- Domyślnie wątek ma taki priorytet jak wątek, który go stworzył. Wątek główny (metoda **main**) ma priorytet 5.
- Priorytety wątków uwzględniane są przy przydzielaniu wątkom czasu procesora.

Priorytety wątków (cd.)

- Priorytet wątku można dynamicznie zmieniać przez wywołanie na rzecz obiektu wątku metody **setPriority**, np.:

watek.setPriority(8);

Przykłady dodatkowych metod klasy Thread

- Metody statyczne (wywoływane na rzecz klasy):
  - **yield( )** - polecenie systemowi aby przerwał wykonywanie bieżącego wątku i przydzielił czas procesora następnemu wątkowi,
  - **sleep(int n)** – uśpienie bieżącego wątku na *n* milisekund.
- Metody wywoływane na rzecz obiektu wątku:
  - **suspend( )** - zwieszenie wykonywania wątku,
  - **resume( )** - wznowienie wykonywania zawieszonego wątku,
  - **setName(String nazwa)** – ustawienie nazwy dla wątku.