# Programowanie współbieżne

Podstawowe pojęcia i problemy programowania współbieżnego.

Prowadzący: dr inż. Jarosław Rulka
jaroslaw.rulka@wat.edu.pl
pok. 252C/100

# Sprawy organizacyjne

- Konsultacje w pok. 252C lub zdalnie:
  - termin ????
- Zaliczenie wykładu:
  - Średnia ważona z trzech ocen: kolokwium, laboratorium, projektu;
  - Oceny pozytywne z laboratorium i projektu – wagi 0,3;
  - Ocena pozytywna z kolokwium (ostatni wykład) – waga 0,4;

# Tematyka przedmiotu

| lp | Temat | W | L | Pr |
|---|---|---|---|---|
| 1 | Podstawowe pojęcia programowania współbieżnego. Wzajemne wykluczanie procesów sekwencyjnych. Struktura kodu źródłowego programu współbieżnego. Algorytm Dekkera, algorytm Petersona. | 3 | | |
| 2 | Programowanie współbieżne z zastosowaniem semaforów. Przegląd klasycznych problemów programowania współbieżnego. | 3 | 6 | |
| 3 | Programowanie współbieżne z zastosowaniem monitora procesów sekwencyjnych. | 2 | 6 | |
| 4 | Programowanie współbieżne z zastosowaniem mechanizmów synchronizacji wbudowanych w język Ada. Zarządzanie zadaniami w języku Ada, mechanizm spotkaniowy, obiekty chronione. Implementacja semaforów w Adzie. | 2 | 4 | |
| 5 | Programowanie współbieżne z zastosowaniem mechanizmów synchronizacji wbudowanych w język JAVA / C#. | 2 | 4 | |
| 6 | Realizacja samodzielnych zadań laboratoryjnych z wykorzystaniem mechanizmów synchronizacji języka JAVA / C#. | | | 12 |
| **Razem** | | **12** | **20** | **12** |

# Literatura przedmiotu

**I. Literatura obowiązkowa**


1) M. Ben-Ari: „Podstawy programowania współbieżnego i rozproszonego”, WNT, Warszawa 2009.
2) M. Ben-Ari: „Podstawy programowania współbieżnego i rozproszonego”, WNT, Warszawa 1996, sygn. 53403.
3) A. Karbowski, E. Niewiadomska-Szynkiewicz: „Programowanie równoległe i rozproszone”, Oficyna Wydawnicza PW, Warszawa 2009.
4) Z. Weiss, T. Gruźlewski: „Programowanie współbieżne i rozproszone w przykładach i zadaniach”, WNT, Warszawa 1993.
5) W. Stallings: „Systemy operacyjne. Struktura i zasady budowy”, PWN, Warszawa 2006 (niniejsza książka została również wydana przez Wydawnictwo Robomatic w 2004 roku pt. „Systemy operacyjne”).
6) A. Silberschatz, P. B. Galvin: „Podstawy systemów operacyjnych”, WNT, Warszawa 1993, 2000, 2005 - sygn. 56239, 51409 i in.
7) A.S. Tanenbaum: „Systemy operacyjne”, Helion, 2010.
8) B. Goetz i in.: „Java. Współbieżność dla praktyków”, Helion, 2007.
9) M. Herlihy, N. Shavit: „Sztuka programowania wieloprocesorowego”, PWN, 2010.


**II. Literatura uzupełniająca**

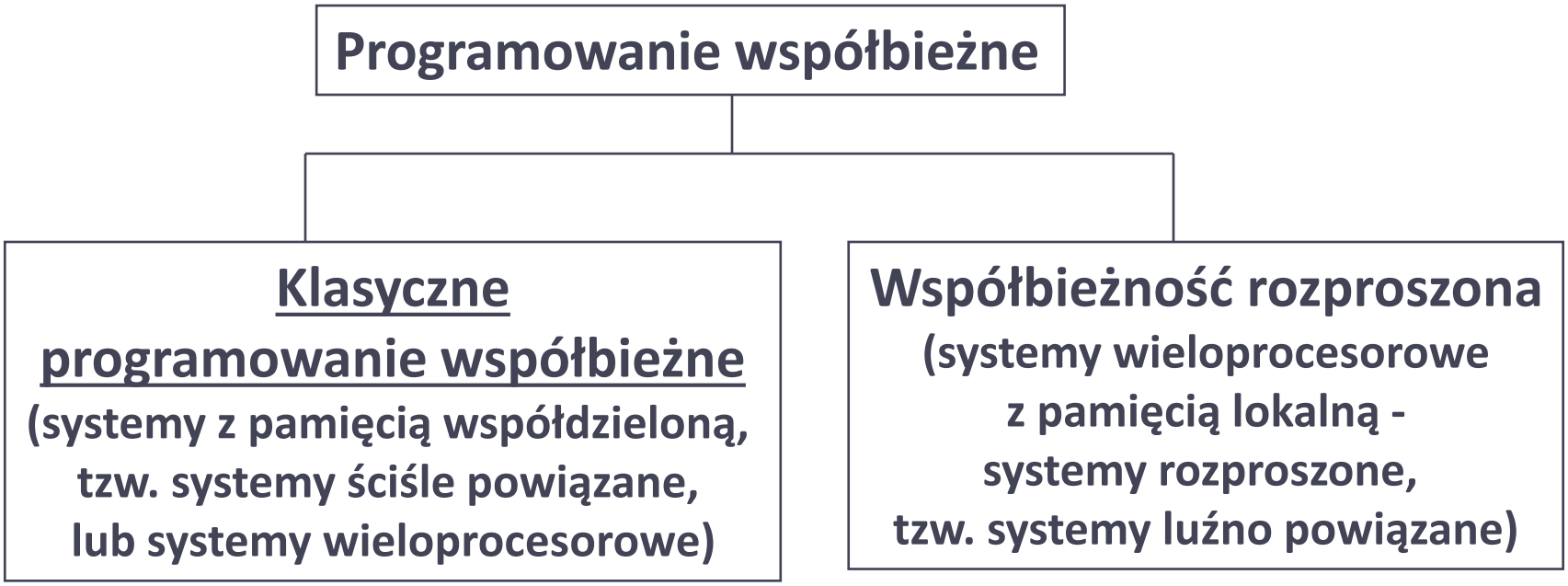

1) G. Gębal: „Programowanie współbieżne w Adzie”, Wydawnictwo Stachowski, 1999.
2) B. Goodheart, J. Cox: „Sekrety magicznego ogrodu. UNIX® System V Wersja 4 od środka”, WNT, Warszawa 2001.
3) U. Vahalia: „Jądro systemu UNIX. Nowe horyzonty”, WNT, Warszawa, 2001.
4) J. Richter: „Programowanie aplikacji dla Microsoft Windows”, Wydawnictwo RM, 2002.
5) C. Breshears: „The Art of Concurrency”, O’Reilly, 2009.
6) D. Harel: „Rzecz o istocie informatyki. Algorytmika”, WNT, 1992 i późn.
7) S. Wrycza, B. Marcinkowski, K. Wyrzykowski: „Język UML 2.0”, Helion, 2005.
8) J. Richter, Ch. Nasarre: „Windows via C/C++”, Microsoft Press, 2007.
9) J. Duffy: „Concurrent Programming on Windows”, Addison-Wesley, 2009.

# Programowanie współbieżne – definicja

- **Programowanie współbieżne** (M. Ben-Ari) – zbiór technik i notacji programistycznych służących do wyrażania potencjalnej równoległości oraz do rozwiązywania zagadnień związanych z powstającymi przy tym problemami: synchronizacji i komunikacji.
- Pozwala rozważać równoległość algorytmu obliczeniowego bez wdawania się w szczegóły implementacyjne.

# Odmiany programowania współbieżnego

(z pominięciem programowania równoległego)



# Programowanie współbieżne na tle innych dziedzin informatyki


- Większość języków programowania nie posiada mechanizmów programowania współbieżnego (równoległego);
- Pra–język **C** także nie został zaprojektowany do programowania zadań do współbieżnej realizacji - jednakże jako język systemu **Unix** może wywoływać funkcje systemowe;
- Pierwszym językiem dedykowanym była **Ada** (~1980r.);
- Kolejnym istotnie ważnym językiem (poza laboratoryjnymi rozwiązaniami) stała się **Java, C#**;

# Program współbieżny

- ***Program współbieżny*** jest zbiorem powiązanych (pod)programów sekwencyjnych wykonywanych abstrakcyjnie „równolegle” w określonym, wspólnym celu.
- Każdy uruchomiony (pod)program sekwencyjny wchodzący w skład programu współbieżnego będziemy nazywali ***procesem współbieżnym*** *(w skrócie procesem)*.
- Każdy proces może być realizowany na abstrakcyjnym (wirtualnym) **procesorze** – fizycznie występują różne architektury;
- Mówimy, że dwa (lub więcej) procesy są **współbieżne**, jeśli jeden z nich rozpoczyna się przed zakończeniem drugiego;

  W świetle tej definicji **proces nieskończony jest współbieżny** ze wszystkimi procesami, które rozpoczęły się od niego później;

# Wielozadaniowość – ziarnistość obliczeń

- ***Ziarnistość obliczeń*** nie jest pojęciem ścisłym – opisuje średnią liczbę operacji obliczeniowych między punktami synchronizacji procesów oraz synchronizacji dostępu do danych (na maszynach z pamięcią wspólną) lub komunikacji (na maszynach z pamięcią lokalną).
- Im większa jest liczba operacji obliczeniowych, tym większa ziarnistość (grube ziarno, ***gruboziarnistość*** - ang. *coarse grain*).
  - Mniejszy narzut synchronizacyjno-komunikacyjny.
- Ziarnistość mała (***drobnoziarnistość*** - ang. *fine grain*) występuje, gdy mała liczba operacji obliczeniowych przypada na procesor pomiędzy instrukcjami odpowiedzialnymi za punkty komunikacji i synchronizacji.
  - Większy narzut synchronizacyjno-komunikacyjny.

# Abstrakcja programowania współbieżnego (1/2)

- Programowanie współbieżne jest abstrakcją stworzoną w celu modelowania i analizowania zachowania programów współbieżnych.

- Abstrakcja programowania współbieżnego polega na modelowaniu i badaniu **przeplatanych** ciągów wykonań (realizacji) **atomowych** (niepodzielnych) instrukcji procesów współbieżnych.

# Abstrakcja programowania współbieżnego (2/2)

Abstrakcja programowania współbieżnego obejmuje:

1. **proces współbieżny**
2. **model wzajemnych oddziaływań procesów,**
3. **instrukcje atomowe,**
4. **czas,**
5. **przeplot,**
6. **poprawność programu.**

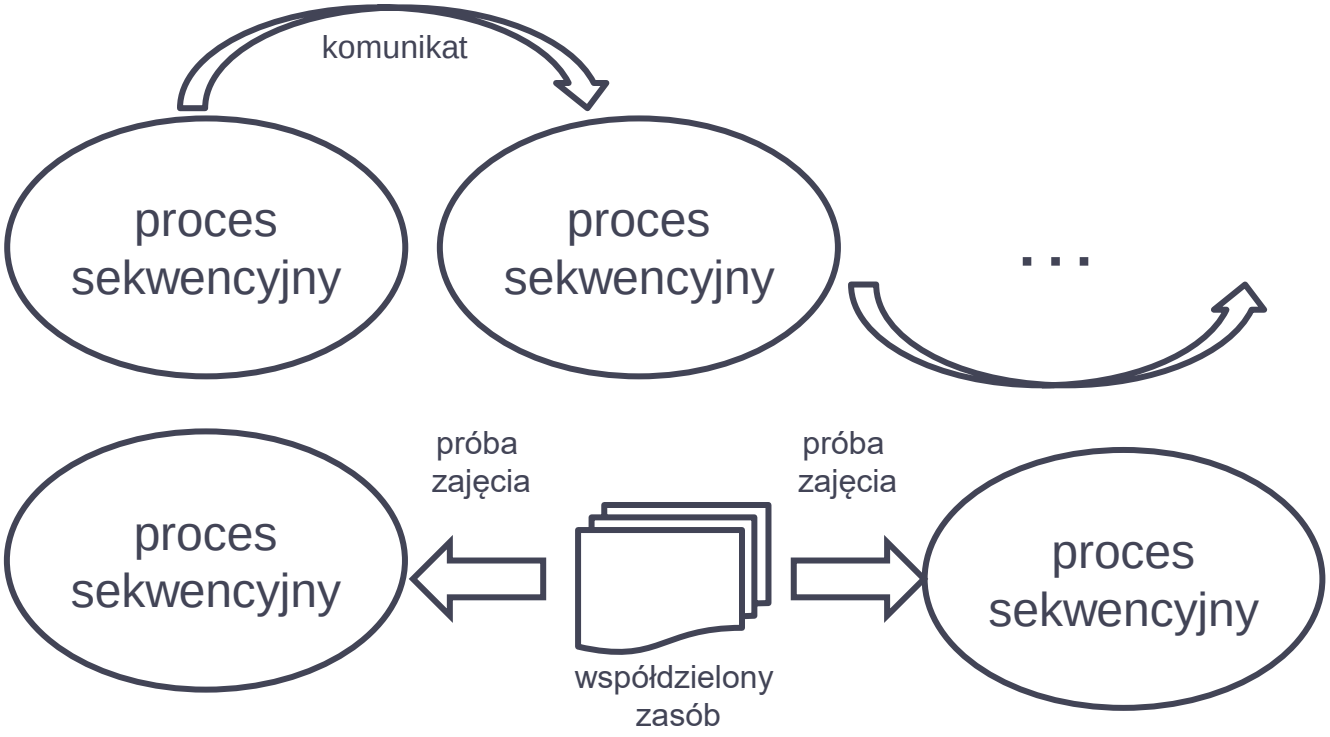
# Proces a wątek

- Abstrakcja procesu współbieżnego jest fizycznie realizowana przez:
  - **Proces** systemu operacyjnego:
    - program z danymi wykonywany pod kontrolą systemu operacyjnego, posiadający przydzielone zasoby (pamięć, urządzenia we/wy) i rywalizujący o dostęp do procesora;
  - **Wątek** *(thread)* – lekki proces *(lightweight process)*:
    - Wątek ma własne sterowanie z czym wiąże się tzw. kontekst wątku obejmujący licznik rozkazów, stan rejestrów procesora oraz stos;
    - kod i dane są współdzielone między wątkami;
    - pracą wątków steruje planista wątków (*thread scheduler*) przydzielając im procesor na określony kwant czasu;
      - wiąże się z tym pojęcie wielowątkowości *(multithreading);*

*Uwaga: np. w Javie każdy program jest wielowątkowy!*

# Model wzajemnych oddziaływań procesów (1/2)

- Zajmujemy się abstrakcjami, w których procesy ze sobą **współpracują** lub między sobą **współzawodniczą** (wg M. Ben-Ari);



# Model wzajemnych oddziaływań procesów (2/2)

- **Współpraca** (kooperacja):
  - występuje, gdy procesy realizują oddzielne zadania, lecz w sumie działają, aby osiągnąć wspólny cel;
  - wymaga komunikacji:
    - synchroniczna, asynchroniczna;
    - często realizowana poprzez wspólną pamięć, co prowadzi do współzawodnictwa – wymaga wówczas wzajemnego wykluczania procesów będących stronami w komunikacji;
- **Współzawodnictwo**:
  - dwa procesy ubiegają się o ten sam zasób – np. obliczeniowy, dostęp do pewnej komórki pamięci lub kanału komunikacyjnego;
  - współzawodnictwo zawsze wymaga synchronizacji, w tym rozwiązania problemu wzajemnego wykluczania;

# Instrukcje atomowe (1/2)

- Przez **instrukcje atomowe** rozumie się podzbiór wybranych rozkazów procesora realizujących algorytm współbieżny.
- Atomowość instrukcji obejmuje jej:
  1. niepodzielność – stan „przed” i stan „po”,
  2. nieprzerywalność – jeżeli się rozpoczęła, to musi się zakończyć.
- Atomowość instrukcji oznacza, że niezależnie od sposobu implementacji instrukcji w procesorze **jest ona wykonywana w całości z zachowaniem spójności danych.**
  - W trakcie jej wykonywania nie są obsługiwane żadne przerwania, co oznacza, że musi być dokończona zanim nastąpi przełączenie kontekstu.
  - Wykorzystywane są mechanizmy sprzętowe (dostępu do pamięci) lub synchronizacyjne zapewniające spójność danych.


# Instrukcje atomowe – przykłady – języki programowania (2/2)

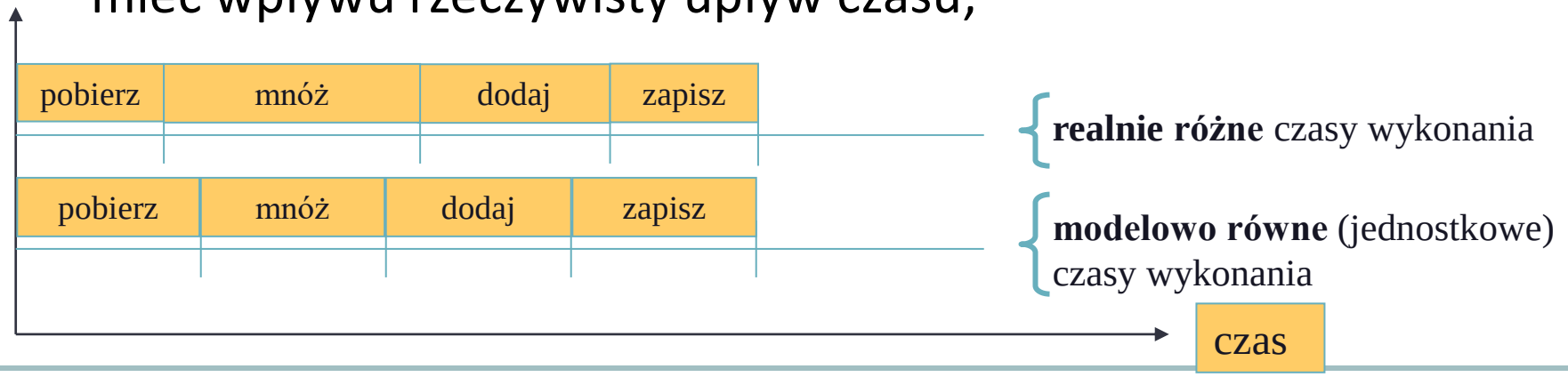
- W języku Java pakiet **`java.util.concurrent.atomic`** dostarcza atomowych typów danych:
  **`AtomicBoolean`**,
  **`AtomicInteger`**,
  **`AtomicLong`**,
  **`AtomicReference`**

- Metoda **`compareAndSet()`** w Java implementuje prymityw synchronizacyjny **CAS** (porównaj aktualną wartość z zadaną i jeśli są równe, to podstaw nową wartość);

- C# dostarcza analogiczną funkcjonalność poprzez metodę **`Interlocked.CompareExchange`**;

# Czas

- **Czas wykonania** instrukcji na różnych procesorach może być różny;
- Abstrakcja programowania współbieżnego zakłada **ignorowanie czasu wykonywania instrukcji** – jednostkowy czas wykonywania (krok);
- **Niedopuszczalna jest zależność czasowa** między procesami!
- Interesuje nas poprawność wykonania ciągu instrukcji;
- Na wynik analizy poprawności algorytmu nie ma i nie może mieć wpływu rzeczywisty upływ czasu;



# Przeplot *(interleave)* (1/3)

- **Przeplot** – jeden z wielu możliwych ciągów (sekwencji, permutacji) realizacji instrukcji atomowych stanowiący złożenie ciągów instrukcji poszczególnych procesów współbieżnych programu z zachowaniem ich kolejności w ramach każdego procesu.
- Założenie:

  Jeżeli rozważymy przypadek dwóch współbieżnych instrukcji **`i11`** i **`i21`** w dwóch procesach **`P1`** i **`P2`**, to muszą one spełniać warunek:

  - Wynik dwu jednocześnie wykonywanych instrukcji musi być taki sam, jak wynik każdego z dwu możliwych ciągów uzyskanych przez wykonanie tych instrukcji jedna po drugiej tzn. **`i11, i21`** lub **`i21, i11`**;

# Przeplot *(interleave)* (2/3)

- Procesory, które przetwarzają procesy programu współbieżnego mogą działać z różną szybkością - powoduje to losowość przeplotu;
- Poprawnie skonstruowany program współbieżny musi być poprawny przy wszystkich poprawnych przeplotach powstałych na różnych procesorach;
- Niech algorytm procesu sekwencyjnego P1 składa się z instrukcji:
  **`P_1: i11, i12, i13`**
- Niech algorytm procesu sekwencyjnego P2 składa się z instrukcji:
  **`P_2: i21, i22, i23`**
- Możliwe są przeploty:
  **`PRZ_1: i11, i21, i12, i22, i13, i23`**
  **`PRZ_2: i11, i12, i13, i21, i22, i23`**
  **`PRZ_3: i21, i22, i11, i12, i13, i23`**
- Przeplot:
  **`PRZ_4: i11, i22, i21, i12, i23, i13`**
  jest niedopuszczalny (błędny). Dlaczego?

# Przeplot *(interleave)* (3/3)

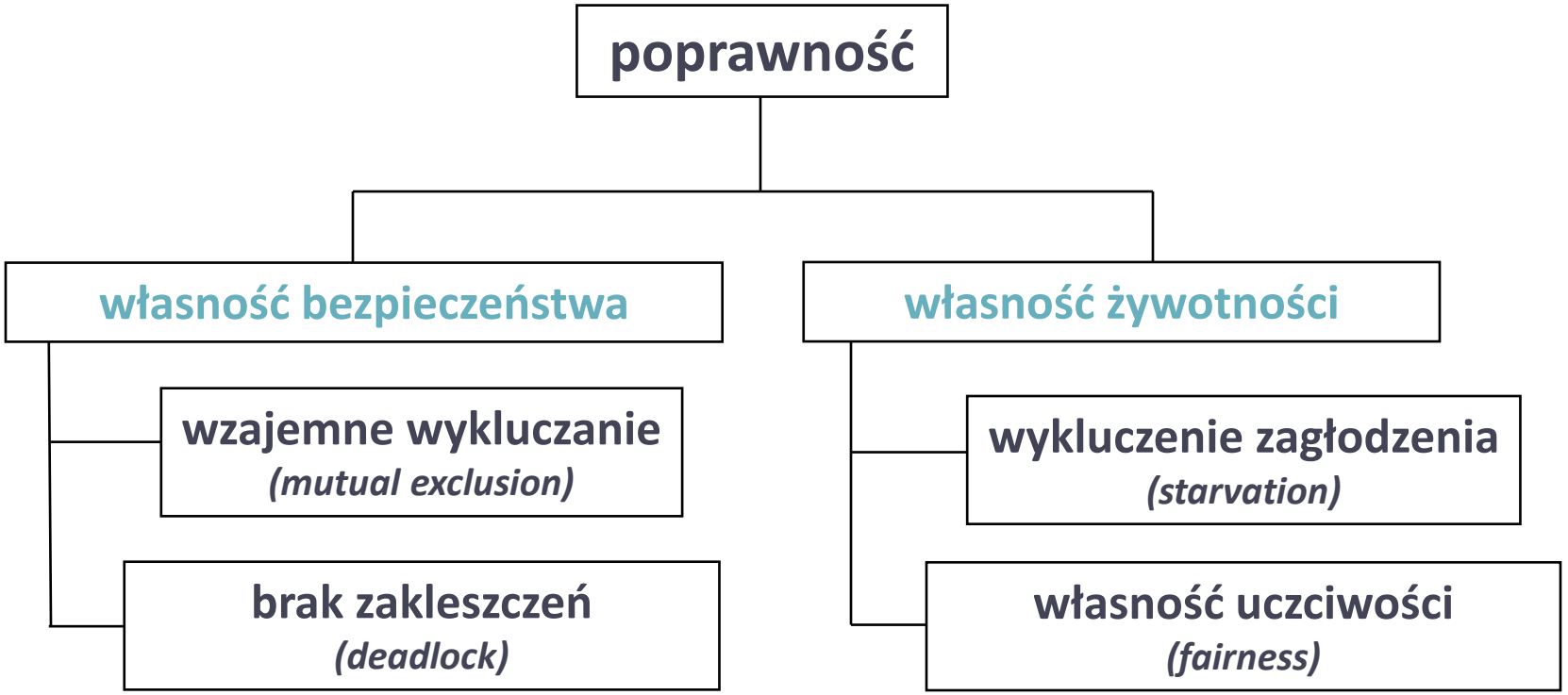
- Uwaga:
  - poprawny program współbieżny działa dobrze dla każdego poprawnego przeplotu;

    czyli program współbieżny nie działa poprawnie, gdy istnieje co najmniej jeden poprawny przeplot, przy którym dochodzi do sytuacji błędnej;

- Istnienie przeplotu w programach współbieżnych:
  - implikuje konieczność specyfikacji zbioru poprawnych przeplotów;
  - jest przyczyną trudności w badaniu poprawności programu: formalnym (teoretycznym) oraz praktycznym;

# Poprawność programu współbieżnego

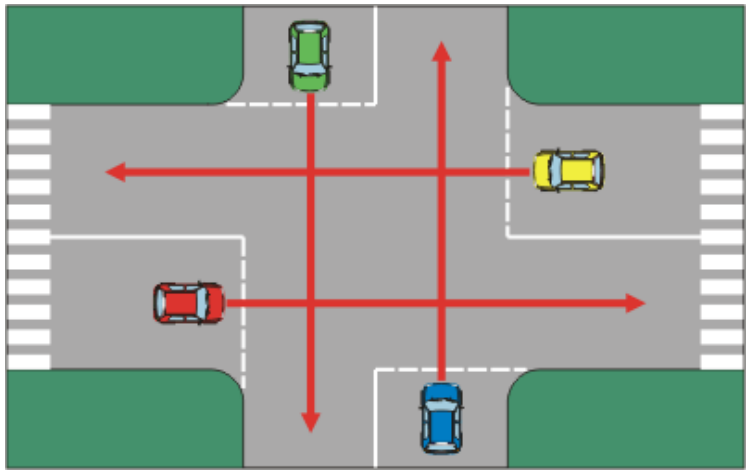
(wg *M. Ben-Ari, „Podstawy programowania współbieżnego i rozproszonego”* )




- *Własność bezpieczeństwa* określa własności statyczne poprawnego programu współbieżnego – zawsze muszą być spełnione;
- *Własność żywotności* dotyczy własności dynamicznych – kiedyś muszą być spełnione (teraz lub w przyszłości);

# Poprawność – bezpieczeństwo i żywotność

- Ilustracją dla własności bezpieczeństwa i żywotności może być sytuacja na skrzyżowaniu drogowym:
  - Własność **bezpieczeństwa**: na poprawnie działającym skrzyżowaniu nigdy jednocześnie nie znajdą się pojazdy jadące w kierunkach „wschód-zachód” i „północ-południe”;



  - Własność **żywotności**: każdy pojazd, który zamierza przejechać przez skrzyżowanie, kiedyś przez nie przejedzie;


# Podstawowe problemy synchronizacji w programowaniu współbieżnym

# Problem wzajemnego wykluczania

- Problem (i potrzeba) wzajemnego wykluczania w programowaniu współbieżnym występuje, gdy co najmniej dwa procesy nie mogą:
  1. przeplatać pewnych ciągów instrukcji,
  2. jednocześnie korzystać z tych samych danych (współdzielonych).
- Wspólny zasób nosi nazwę **zasobu krytycznego**;
- **Sekcja_krytyczna**
  - Ciąg instrukcji programu, który jest wykonywany w ramach jednego procesu sekwencyjnego oraz jest niepodzielny dla wszystkich dopuszczalnych przeplotów przez instrukcje z innych sekcji krytycznych.
  - Fragment programu, który może być wykonywany w tym samym czasie przez co najwyżej jeden proces.
  - Sekcje krytyczne w procesach dotyczące tego samego punktu synchronizacji nie muszą być identyczne.

# Przeplot dla sekcji krytycznej

- Niech algorytm procesu sekwencyjnego P1 składa się z instrukcji:
  **P_1: i11, i12, i13, i14, i15, i16**
- Niech algorytm procesu sekwencyjnego P2 składa się z instrukcji:
  **P_2: i21, i22, i23, i24, i25, i26**
- Podkreślone instrukcje są instrukcjami sekcji krytycznej
- Możliwe są przeploty:
  **PRZ_1: i11, i21, i12, i13, i14, i15, i16, i22, i23, i24, i25, i26**
  **PRZ_2: i11, i12, i13, i21, i14, i15, i22, i23, i24, i16, i25, i26**

# Przeplot dla sekcji krytycznej

- Niech algorytm procesu sekwencyjnego P1 składa się z instrukcji:
  **P_1: i11, i12, i13, i14, i15, i16**
- Niech algorytm procesu sekwencyjnego P2 składa się z instrukcji:
  **P_2: i21, i22, i23, i24, i25, i26**
- Podkreślone instrukcje są instrukcjami sekcji krytycznej
- Możliwe są przeploty:
  **PRZ_1: i11, i21, i12, i13, i14, i15, i16, i22, i23, i24, i25, i26**
  **PRZ_2: i11, i12, i13, i21, i14, i15, i22, i23, i16, i24, i25, i26**

# Przeplot dla sekcji krytycznej

- Niech algorytm procesu sekwencyjnego P1 składa się z instrukcji:
  **P_1: i11, i12, i13, i14, i15, i16**
- Niech algorytm procesu sekwencyjnego P2 składa się z instrukcji:
  **P_2: i21, i22, i23, i24, i25, i26**
- Podkreślone instrukcje są instrukcjami sekcji krytycznej
- Możliwe są przeploty:
  **PRZ_1: i11, i21, i12, i13, i14, i15, i16, i22, i23, i24, i25, i26**
  **PRZ_2: i11, i12, i13, i21, i14, i15, i22, i23, i16, i24, i25, i26**
- Przeplot:
  **PRZ_3: i11, i21, i12, i13, i14, i22, i15, i16, i23, i24, i25, i26**
  jest niedopuszczalny (błędny). Dlaczego?

# Problem wzajemnego wykluczania

- Schemat pracy procesów współbieżnych może wyglądać następująco:

```
begin
  while true do
  begin
    lokalne_obliczenia;
    protokół_wstępny;
    sekcja_krytyczna;
    protokół_końcowy;
  end
end;
```

# Problem wzajemnego wykluczania

- Należy zapewnić spełnienie zasad:
  - W sekcji krytycznej w tym samym czasie może przebywać co najwyżej jeden proces jednocześnie (bezpieczeństwo).
  - Każdy proces, który chce wykonać sekcję krytyczną, w skończonym czasie powinien do niej wejść (żywotność).
- Protokoły wstępny i końcowy korzysta z dostępnych mechanizmów synchronizacyjnych;
- W przypadku braku wsparcia ze strony systemu operacyjnego należy zastosować algorytmy synchronizacyjne, np. algorytm Petersona, Deckera;

# Zakleszczenia procesów współbieżnych (1/3)

- **Blokada (zastój, zakleszczenie, martwy punkt, impas)** – brak żywotności globalnej – występuje wtedy, gdy każdy proces z danego zbioru procesów jest wstrzymany w oczekiwaniu na zdarzenie, które może być spowodowane tylko przez jakiś inny proces z tego zbioru.
- Zjawisko blokady może być również traktowane jako przejaw braku bezpieczeństwa programu, jest bowiem stanem niepożądanym.

# Zakleszczenia procesów współbieżnych (2/3)

- Warunkiem **zakleszczenia** procesu współbieżnego jest:
  - żądanie równoczesnego dostępu do więcej niż jednego współdzielonego zasobu przez jeden proces.
  - Proces współbieżny po zajęciu jednego z zasobów bezskutecznie usiłuje zająć pozostałe z potrzebnych mu zasobów równocześnie nie zwalniając uprzednio zajętego zasobu(ów).
  - W tym czasie każdy z **zasobów**, który został zajęty przez zakleszczony proces współbieżny **jest blokowany** dla pozostałych procesów oczekujących na ten zasób.
- Wyróżnia się:
  - **zakleszczenia symetryczne**,
  - **cykle zakleszczeń symetrycznych**.

# Zagłodzenie procesów współbieżnych (1/2)

- **Zagłodzenie (wykluczenie)** – brak żywotności lokalnej – występuje wtedy, gdy proces nie zostaje wznowiony, mimo że zdarzenie, na które czeka, występuje dowolną liczbę razy i za każdym razem, gdy proces ten mógłby być wznowiony, jest wybierany jakiś inny czekający proces.
  - W celu oceny żywotności algorytmu z punktu widzenia pojęcia czasu i jego miary (określenia: „w skończonym czasie”, „kiedyś”) wprowadza się pojęcie **uczciwości**.
  - Mimo że możliwość zagłodzenia świadczy o niepoprawności programu, to czasami jest akceptowana (na przykład wszędzie tam, gdzie stosuje się kolejkę priorytetową).
    - Wynika to z faktu, że zagłodzenie jest zwykle mało prawdopodobne oraz nie jest permanentne.

# *Blokada procesu*

- Występują dwa sposoby oczekiwania procesu współbieżnego na udostępnienie współdzielonego zasobu:

  - **aktywne czekanie** (ang. busy-waiting) – proces samoczynnie co pewien interwał czasowy ponawia swoje żądanie dostępu do zasobu:
    - **trwałe aktywne czekanie** – proces ponawia żądania dostępu aż do skutecznego wywłaszczenia zasobu,
    - **nietrwałe aktywne czekanie** – proces ponawia żądanie dostępu do zasobu określoną liczbę razy, po czym przechodzi do pasywnego czekania.

  - **pasywne czekanie** – proces jednorazowo wysyła do tzw. „planisty” żądanie dostępu do zasobu, po czy przestaje być aktywny. Jedynie planista jest w stanie ponownie uaktywnić ten proces.

# Własność uczciwości – rodzaje

- ***Uczciwość słaba*** – jeżeli proces nieprzerwanie zgłasza żądanie, to w końcu będzie ono obsłużone.
- ***Uczciwość mocna*** – jeżeli proces zgłasza żądanie nieskończenie wiele razy, to w końcu będzie ono obsłużone.
- ***Oczekiwanie liniowe*** – jeżeli proces zgłasza żądanie, to będzie ono obsłużone zanim dowolny inny proces zostanie obsłużony więcej niż raz.
- ***FIFO*** (pierwszy wszedł, pierwszy wyjdzie) – jeżeli proces zgłasza żądanie, to będzie ono obsłużone przed dowolnym żądaniem zgłoszonym później.

- Uczciwość słaba i mocna mają znaczenie teoretyczne.
- W praktyce jest stosowane oczekiwanie liniowe lub FIFO.
- Obydwa mogą być implementowane w systemach scentralizowanych – w systemach rozproszonych występują problemy z realizacją algorytmu FIFO.