Programowanie Współbieżne

Wstęp

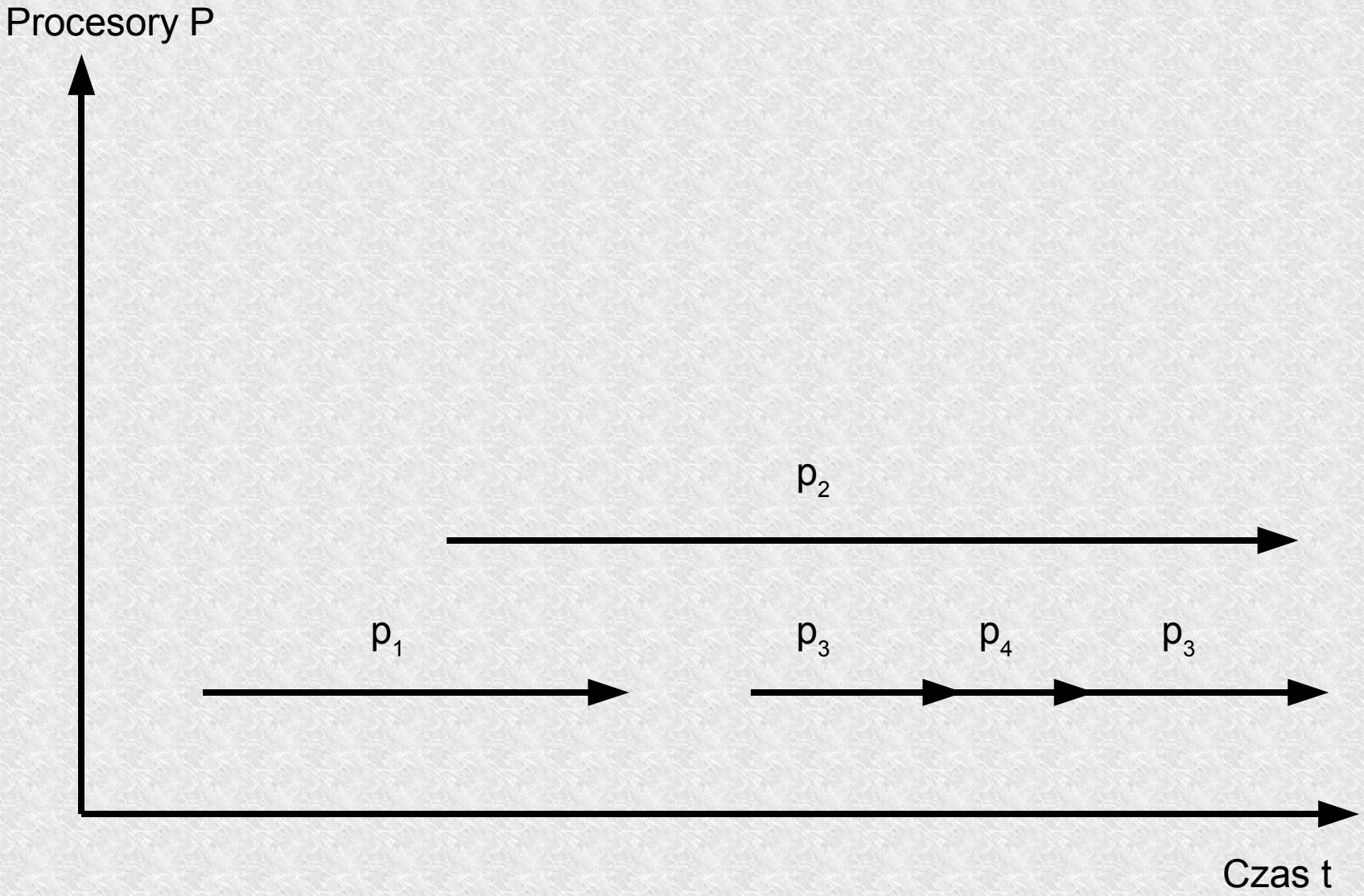
Literatura

- M. Ben-Ari, *Podstawy programowania współbieżnego i rozproszonego*
- *W. Richard Stevens – Programowanie zastosowań sieciowych”*
- A.S. Tanenbaum, *Rozproszone systemy operacyjne*, PWN, 1997.
- M.J Bach, *Budowa systemu operacyjnego UNIX, WNT, 1995*
- Z. Manna, A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems (Specific ation)*, Springer-Verlag, 1992
- W. Iszkowski M. Maniecki. - Programowanie współbieżne.

Podstawowe pojęcia

- **Proces** – program sekwencyjny w trakcie wykonywania
- **Procesy współbieżne –** procesy, których wykonanie może (ale nie musi) przebiegać równolegle. Jeden proces zaczął się przed końcem drugiego
- **Procesy równoległe** – procesy współbieżne wykonywane w tym samym czasie

Równoległość a współbieżność



Podstawowe pojęcia

- **Program współbieżny** – program składający się z kilku procesów sekwencyjnych które zazwyczaj przesyłają między sobą jakieś dane lub tylko się synchronizują
- **Programowanie współbieżne** jest tworzeniem programów, których wykonanie powoduje uruchomienie pewnej liczby procesów współbieżnych (zazwyczaj procesy te są zależne)

Podstawowe pojęcia

- **Zdarzenia**
  - **Synchroniczne –** takie na które czkamy
  - **Asynchroniczne –** występują niespodziewanie w dowolnej chwili
- **Instrukcja atomowa** - niepodzielna

Podstawowe pojęcia

- **Procesy zależne –** dwa procesy nazywamy zależnymi, jeżeli fakt wykonywania któregokolwiek z nich wpływa na wykonywanie drugiego
- **Zmienna dzielona** – zmienna wspólna, wykorzystywana przez kilka współbieżnych procesów.
- **Sekcja krytyczna** – fragment procesu, w którym korzysta on ze zmiennej dzielonej
- **Synchronizacja** – uporządkowanie akcji poszczególnych procesów w czasie

Podstawowe pojęcia

W literaturze można spotkać, określenia.

- Współbieżny (concurrent)
- Równoległy (parallel)
- Rozproszony (distributed)

Podstawowe pojęcia

Program współbieżny jest **poprawny** jeśli posiada własności **bezpieczeństwa** i **żywotności.**

- **Własność Bezpieczeństwa -** proces utrzymuje system w pożądanym stanie.
- **Własność żywotności** – jeżeli któryś z procesów czeka na jakieś zdarzenie, to ono w końcu zajdzie. Szczególny rodzaj własności żywotności nazywa się **własnością uczciwości**

Podstawowe pojęcia

- **Uczciwość słaba** –Jeśli proces nieprzerwanie zgłasza żądanie, to kiedyś będzie ono obsłużone.
- **Uczciwość mocna** – Jeśli proces zgłasza żądanie nieskończenie wiele razy to kiedyś będzie ono obsłużone.
- **Oczekiwanie liniowe** – Jeśli proces zgłasza żądanie, będzie ono obsłużone zanim dowolny inny proces zostanie obsłużony więcej niż raz.
- **FIFO** – Jeśli proces zgłasza żądanie, to będzie ono obsłużone przed dowolnym żądaniem zgłoszonym później.

Podstawowe pojęcia

Najczęstsze błędy

- **Blokada –** (deadlock) – każdy proces ze zbioru P czeka na zdarzenie, które może być spowodowanie wyłącznie przez inny proces z tego zbioru.
- **Zagłodzenie** – (starvation) - sytuacja, w której proces jest nieskończenie wstrzymywany, gdyż zdarzenie na które oczekuje powoduje wznawianie innych procesów
- **Aktywne czekanie** – proces czekający na zdarzenie bez przerwy sprawdza czy ono już wystąpiło angażując niepotrzebnie czas procesora.

Podstawowe pojęcia

- **Przeplot** – Abstrakcja programowania współbieżnego polega na badaniu przeplatanych ciągów wykonań atomowych instrukcji procesów sekwencyjnych

Podstawowe pojęcia

Rozważamy tylko dwa przypadki

- **Współzawodnictwo** - Dwa procesy ubiegają się o ten sam zasób: zasób obliczeniowy, komórka pamięci, lub kanał
- **Komunikacja** - Dwa procesy mogą chcieć się porozumieć, by przesłać dane od jednego do drugiego

Zależności czasowe

Procesy mogą działać z dowolną szybkością i mogą reagować na dowolne zewnętrzne sygnały!
Zależności czasowych brak!

Ciągi wywołań i instrukcje atomowe

- LOAD n;
- ADD 1;
- STORE n;

- LOAD n;
- ADD 1;
- STORE n;

LOAD n; LOAD n; ADD 1; ADD 1; STORE n; STORE n;

LOAD n; ADD 1; LOAD n; ADD 1; STORE n; STORE n;

Klasyczne problemy

- Wzajemnego wykluczania
- Producenta i konsumenta
- Czytelników i pisarzy
- Pięciu filozofów
- Bizantyjskich generałów

Modele programowania rozproszonego

- **Komunikacja synchroniczna** – do wymiany komunikatów niezbędny nadawca i odbiorca
- **Komunikacja asynchroniczna** – nadawca po wysłaniu komunikatu nie musi czekać aż odbiorca go odbierze.

Identyfikowanie procesów i przepływ danych

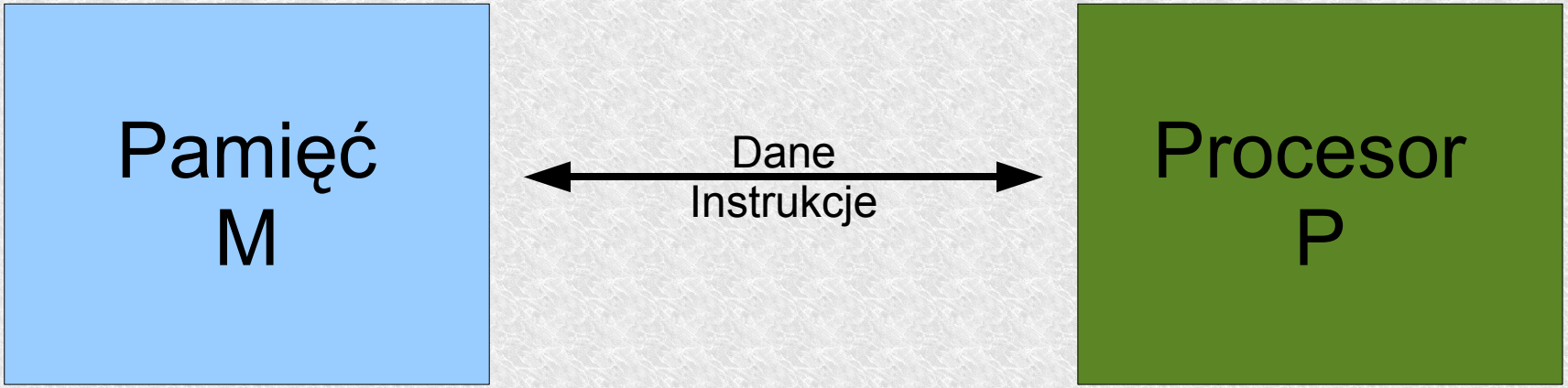
- **Kanały dedykowane** – zarówno nadawca jak i odbiorca znają swoje identyfikatory. Każda wiadomość jest przekazywana bez dodatkowych kosztów związanych np. z przeliczaniem adresu.
- **Komunikacja asymetryczna** – Nadawca zna adres odbiorcy ale odbiorca nie musi go znać. Nadaje się bardzo dobrze do systemów typu *klient – serwer*
- **Rozsyłanie komunikatów** – (BroadCast) Ani odbiorca nie wie od kogo ma odebrać komunikat ani nadawca nie zna odbiorcy, więc wysyła do wszystkich.

Tworzenie procesów

- Dynamiczne
  - Elastyczność
  - Dynamiczne używanie zasobów
  - Wyrównywanie obciążenia
- Statyczne
  - Szybka inicjacja
  - Wyspecjalizowane zadania

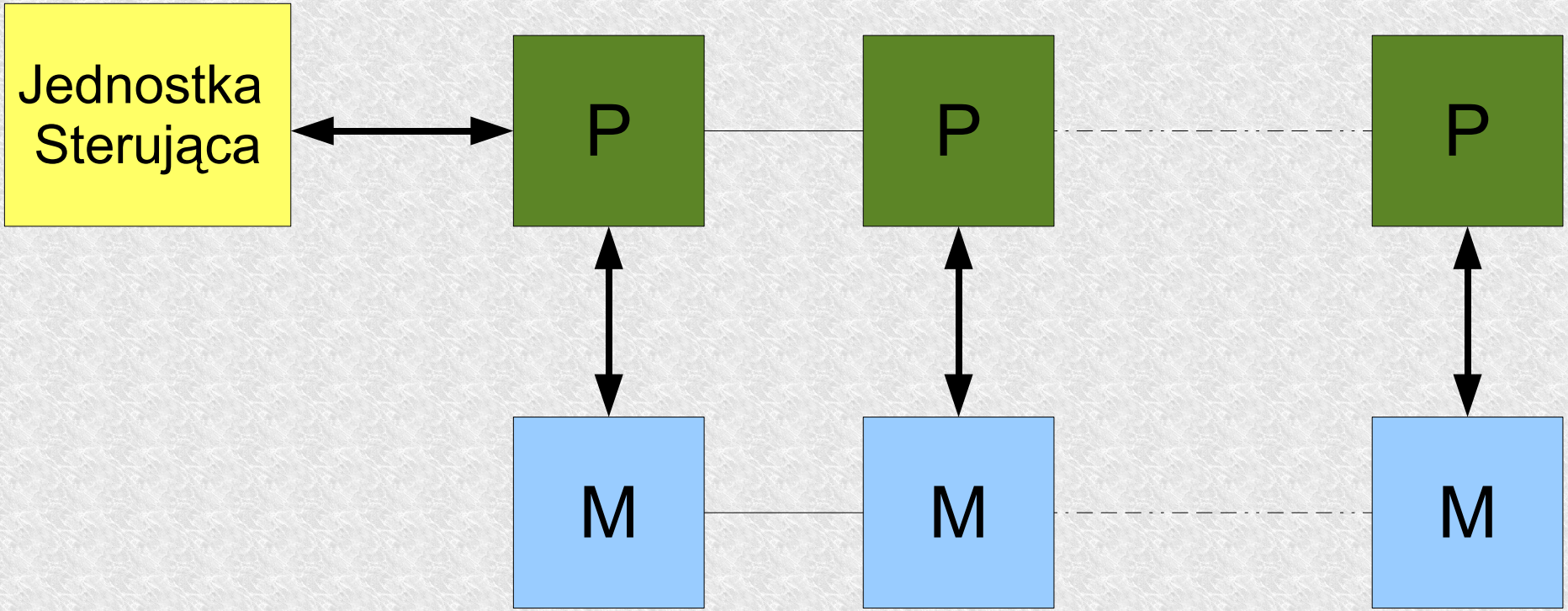
Klasyfikacja maszyn równoległych

SISD – Single Instruction Stream, Single Data Stream



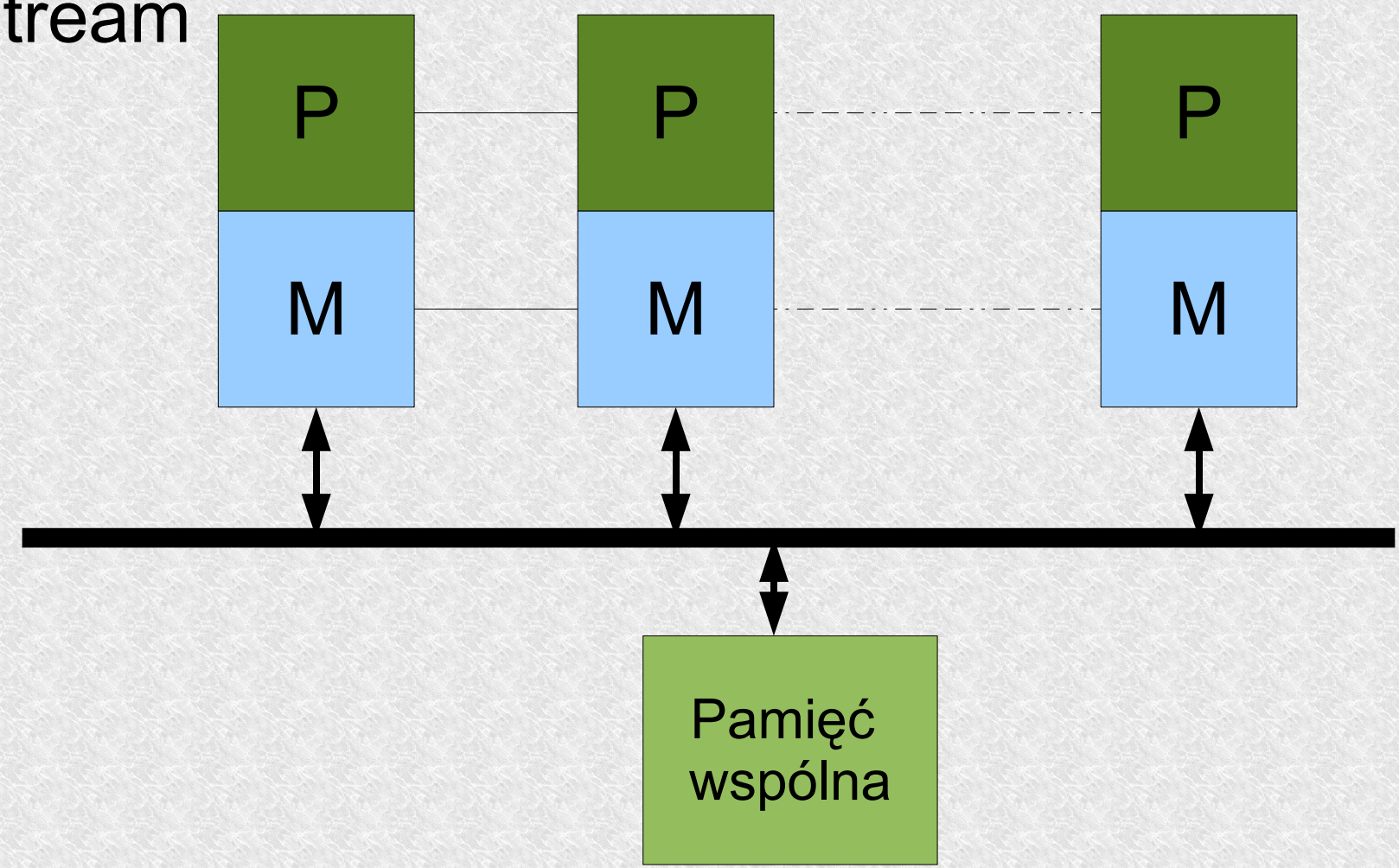
Klasyfikacja maszyn równoległych

SIMD – Single Instruction Stream, Multiple Data Stream



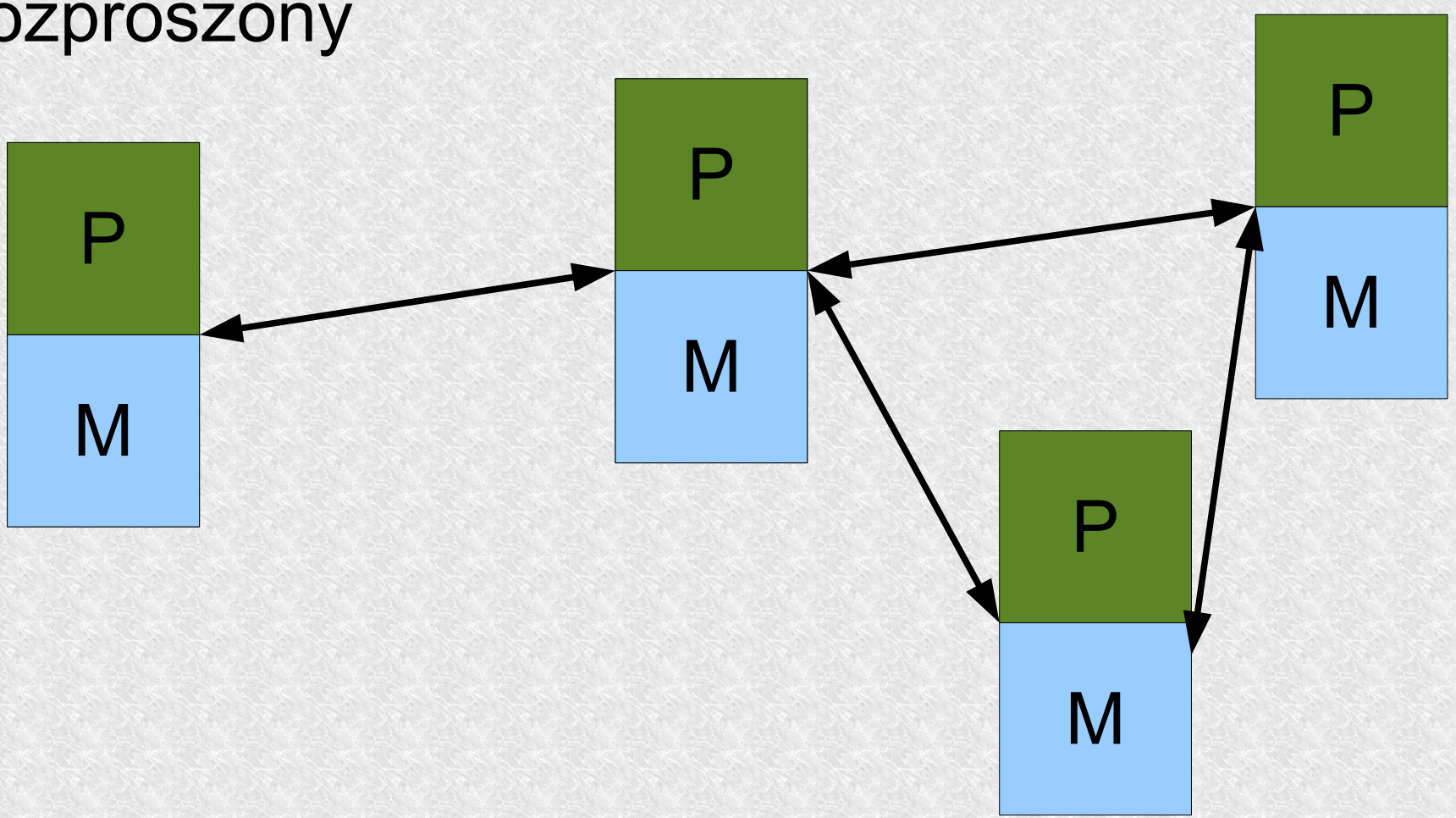
Klasyfikacja maszyn równoległych

MIMD – Multiple Instruction Stream, Multiple Data Stream

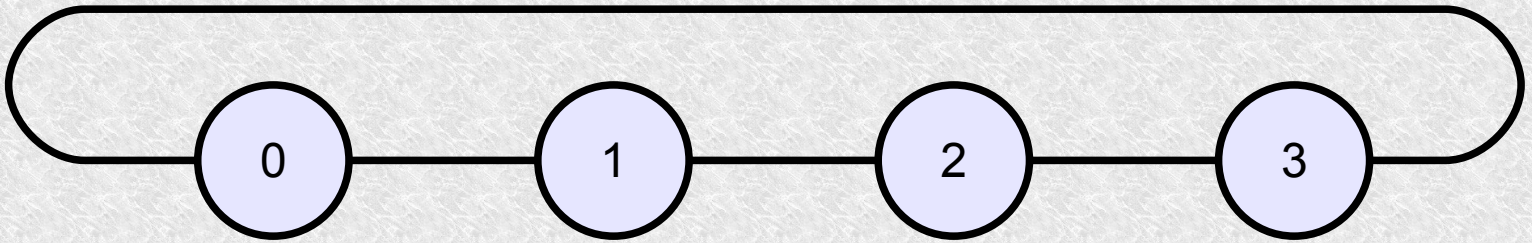


Klasyfikacja maszyn równoległych

MIMD – Bez wspólnej pamięci, system rozproszony

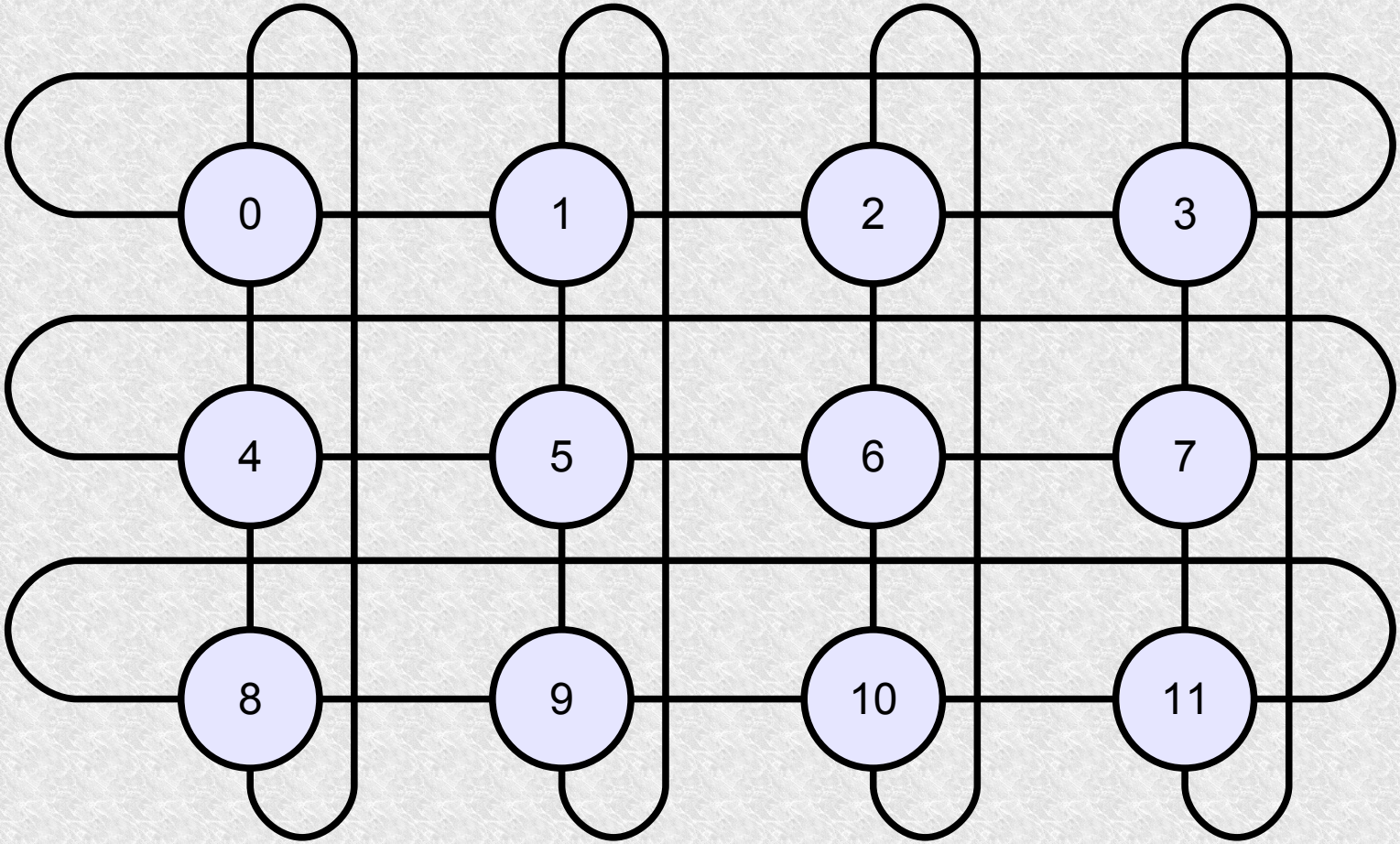


Architektura - pierścień



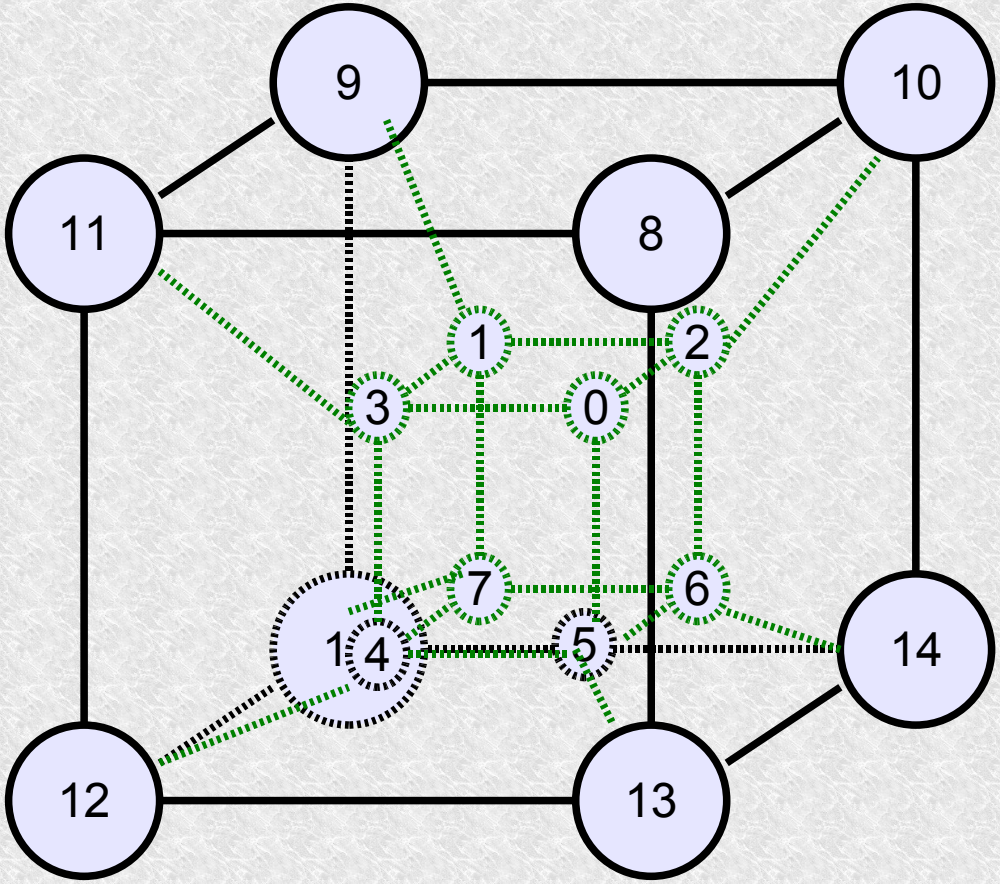
Odległość między dwoma najbardziej odległymi jednostkami = **0,5*p** (z zaokrągleniem w dół) przy przesyłaniu 2 kierunkowym oraz **p-1** przy jednokierunkowym

Architektura – tablica 2 wymiarowa



Odległość $0,5*(p_x+p_y)$

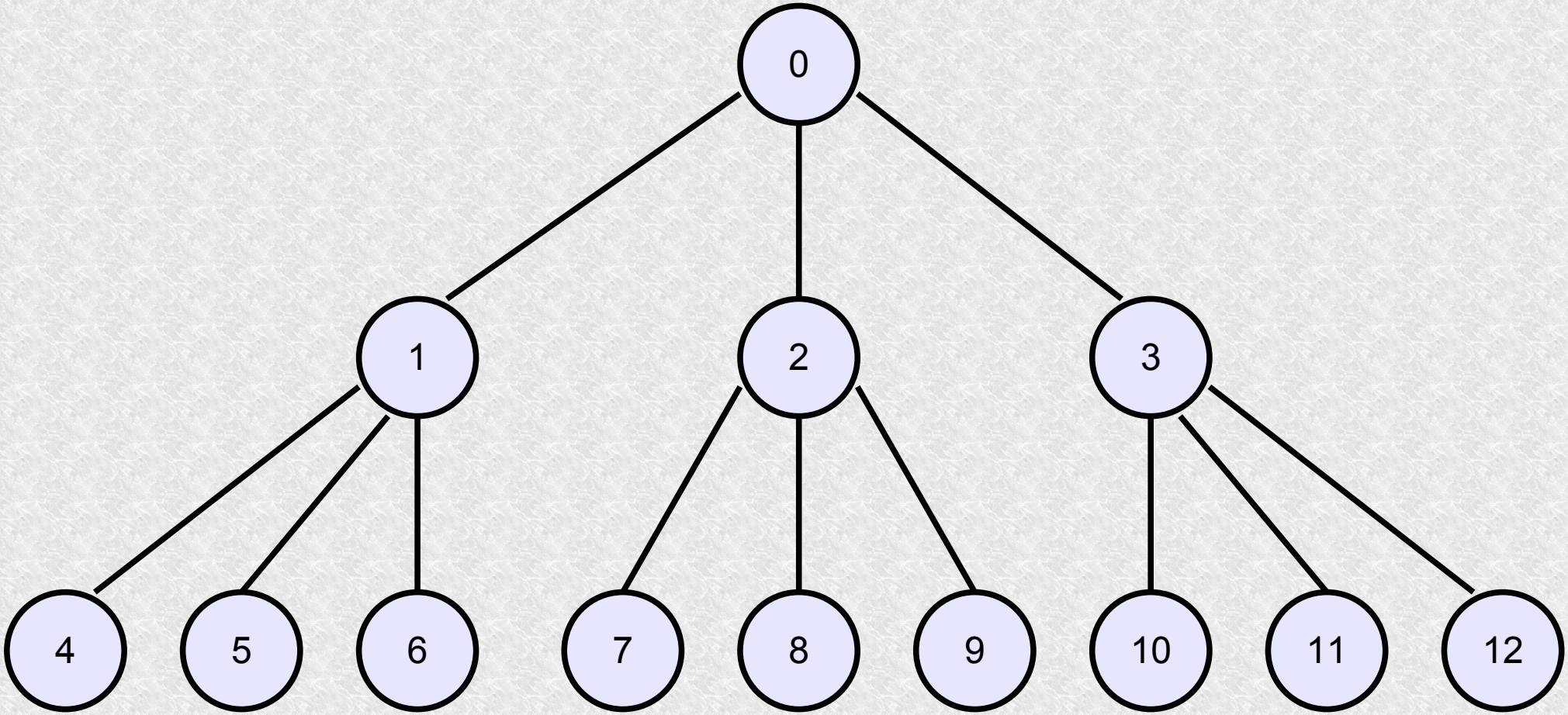
Architektura - Hipersześcian



Architektura - Hipersześcian

- Hipersześcian rzędu n zbudowany jest z 2^n węzłów
- Każda architektura rzędu wyższego zawiera w sobie architekturę rzędu niższego
- W przedstawionym przykładzie 4 wymiarowym mamy odległość $\log_2 p$

Architektura - drzewo



Zaleta: najmniejsza ilość kanałów do komunikacji = p-1
maksymalna odległość 2*il_poziomów
Wada: gdy zawiedzie jeden węzeł to przestaje działać cała gałąź

Klasy równoległości

Ziarnistość

$$G = \frac{czas_{obliczen}}{czas_{komunikacji}}$$

Klasy równoległości

- Równoległość drobnoziarnista (małe G)
- Równoległość gruboziarnista (duże G)

Klasy równoległości

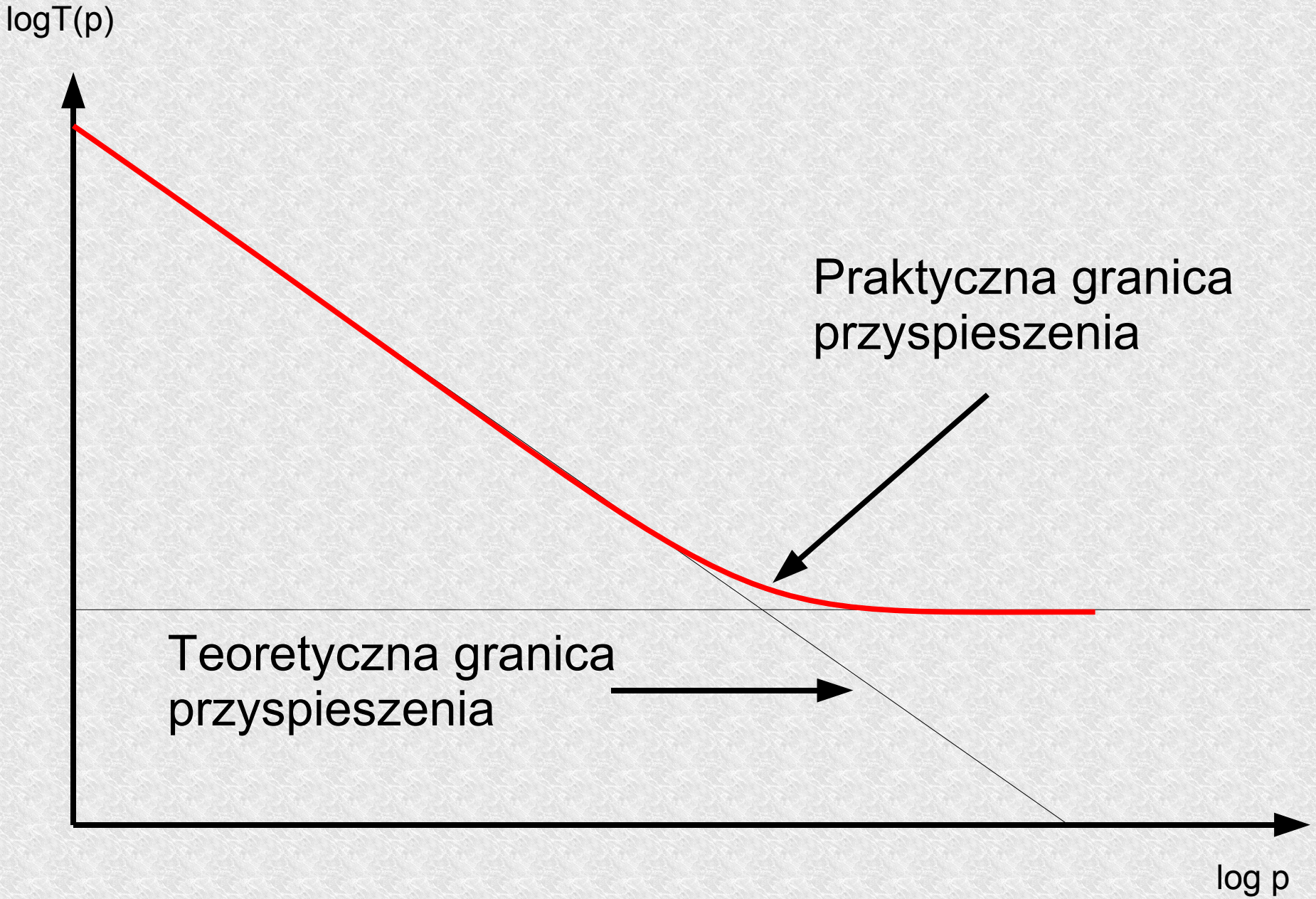
- **Równoległość procesów** – równoległość w kodzie programowym dotyczącym danego problemu obliczeniowego
  - Funkcjonalna – wyspecjalizowane procesy
  - Geometryczna – np. mnożenie macierzy
  - Algorytmiczna – najbardziej drobnoziarnisty podział
- **Równoległość danych** – program jest wykonywany jednocześnie na różnych zbiorach danych.

Przyspieszenie i efektywność

- **Przyspieszenie**
  - To(1) – optymalny czas jednoprocesorowego rozwiązania
  - T(p) – czas wykonania zadania przez p procesorów

$$S(p) = \frac{\mathrm{To}(1)}{\mathrm{T(p)}}$$

Granica przyspieszenia



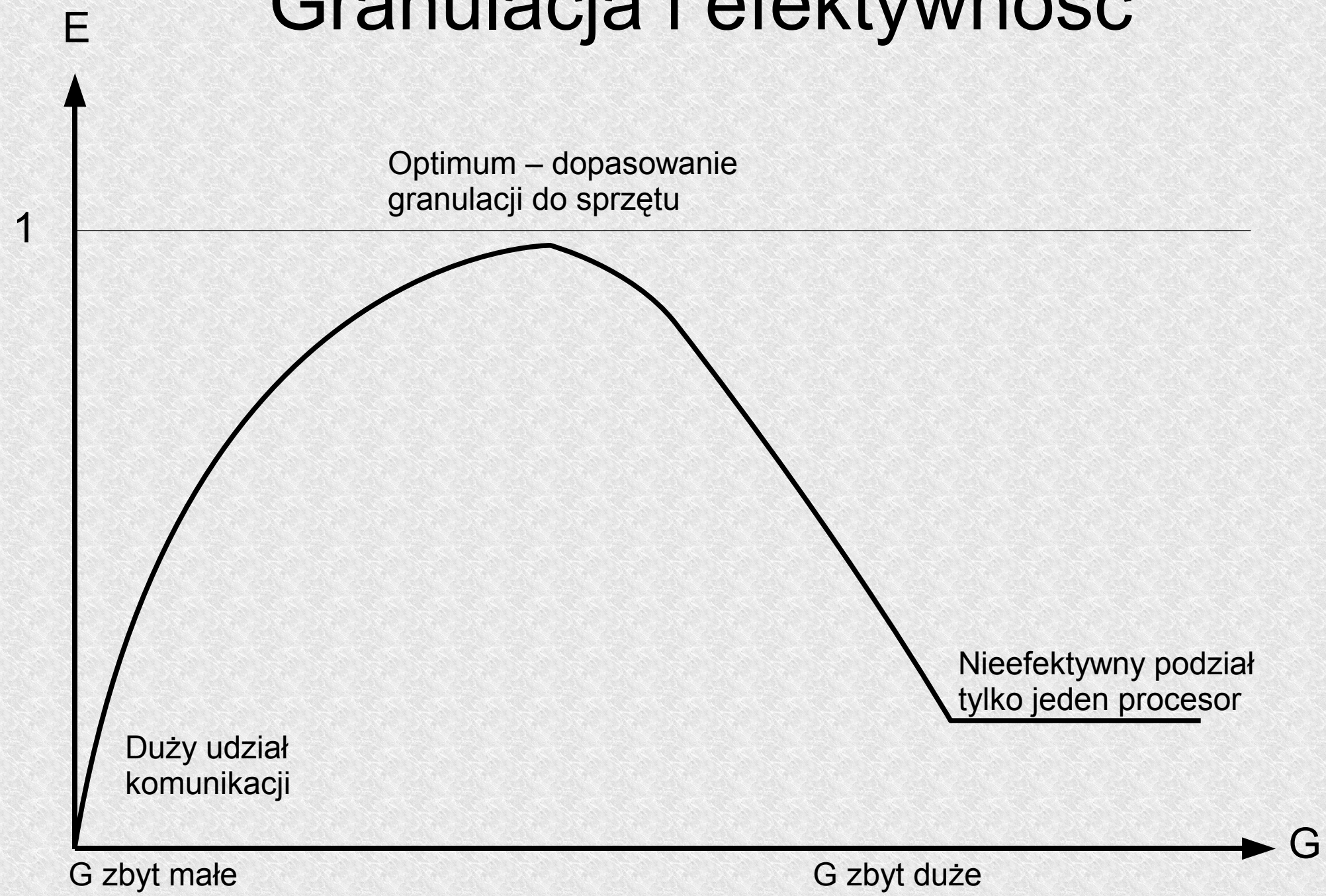
Przyspieszenie i efektywność

- **Efektywność**

$$E(p) = \frac{\mathrm{S(p)}}{p} = \frac{\mathrm{To(1)}}{\mathrm{T(p)*p}}$$

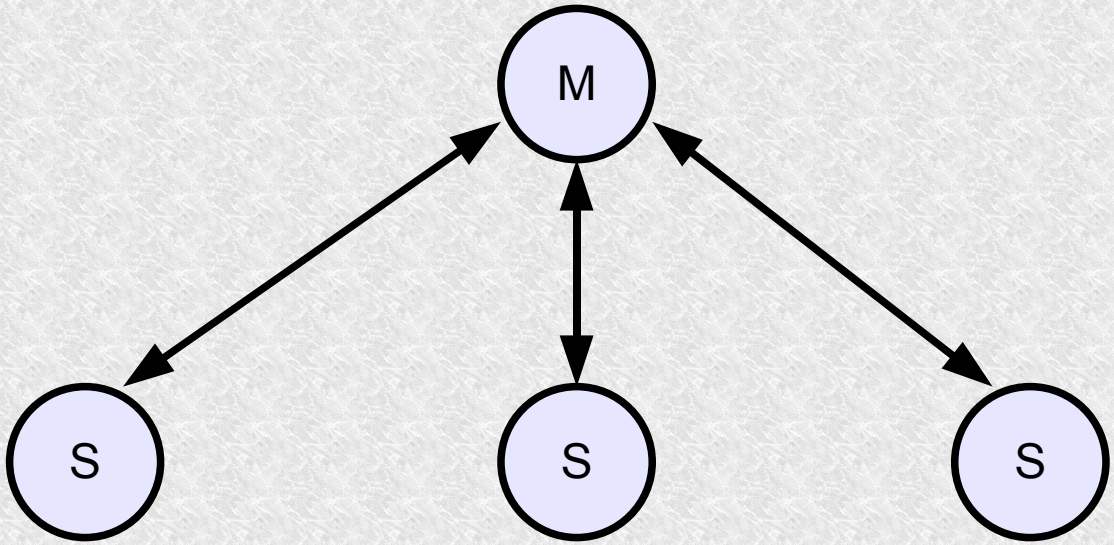
W idealnym przypadku S(p)=p i E(p)=1

Granulacja i efektywność



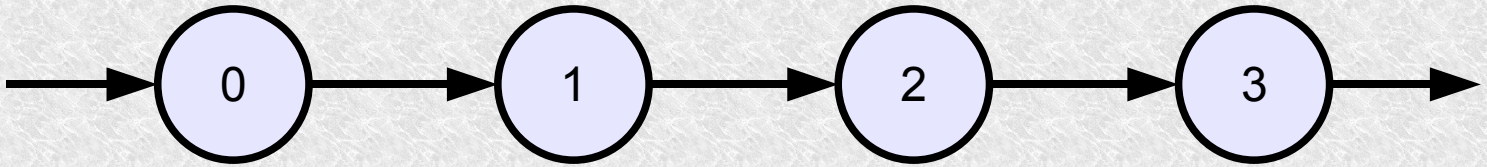
Organizacja obliczeń

Układ z procesem nadrzędnym (Master-Slave)



Organizacja obliczeń

Przetwarzanie potokowe



- Każdy procesor musi czekać aż otrzyma dane
- nie mogą rozpocząć jednocześnie
- cały układ działa w tempie najwolniejszego
- niewielka elastyczność

$$E = \frac{N*p*T}{p*(N+p-1)*T} = \frac{N}{N+p-1}$$

Dla czego warto zrównoleglać

Sortowanie przez zamianę prostą

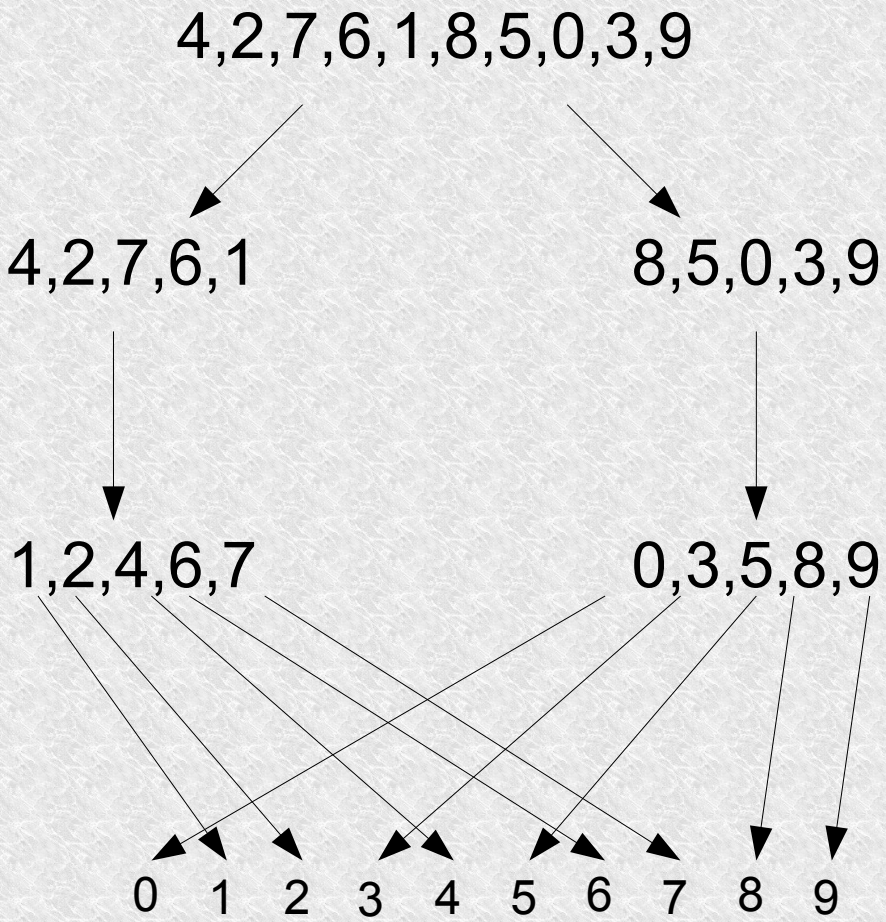
```
program sortowanie
const n=40;
var a:array[1..n] of integer;
    k:integer;
procedure sort(lewy,prawy:integer);
    var i,j,pom:integer;
begin
    for i:=lewy to prawy - 1 do
        for j:=i+1 to prawy do
            if a[j]<a[i] then
                begin
                    pom:=a[j];
                    a[j]:=a[i];
                    a[i]:=pom;
                end;
end;

BEGIN {program główny}
for k:=1 to n do read(a[k]);
sort(1,n);
for k:=1 to n do write(a[k])
END.
```

Sortowanie przez zamianę prostą

Koszt: $(n-1)+(n-2)+\ldots+1=\frac{n(n-1)}{2}$

Czyli około: $\frac{n^2}{2}$

Sortowanie przez scalanie



Sortowanie przez scalanie

```
program sortowanie;
const n = 8;
       dwaen = 16;
var a:array[1..dwaen] of integer;
    k:integer;
procedure scal (lewy,srodkowy,prawy:integer);
var licznik1,licznik2:integer;
    k,indeks1,indeks2:integer;
begin
     licznik1:=lewy;
     licznik2:=srodkowy;
     while licznik1 <srodkowy do
           if a[licznik1] < a[licznik2] then
              begin
                   write(a[licznik1],' ');
                   licznik1:=licznik1+1;
                   if licznik1>=srodkowy then
                      for indeks2:=licznik2 to prawy do
                          write(a[indeks2],' ')
              end
           else
               begin
                    write (a[licznik2],' ');
                    licznik2:=licznik2+1;
                    if licznik2>prawy then
                       begin
                            for indeks1:= licznik1 to srodkowy -1 do
                                write (a[indeks1],' ');
                            licznik2:=srodkowy {zakoncz}
                       end
               end
end;
```

Sortowanie przez scalanie

```
begin {program glowny}
      for k:=1 to dwaen do read(a[k]);
   { tu dodajemy cobegin }
      sort(1,n);
      sort(n+1,dwaen);
   { tu dodajemy coend }
      scal(1,n+1,dwaen);
end.
```

Sortowanie przez scalanie

Szacunkowy koszt:

Aby posortować $\frac{n}{2}$ wykonujemy tylko $\frac{(\frac{n}{2})^2}{2} = \frac{n^2}{8}$ porównań.

Dla dwóch podciągów to jest to $\frac{n^2}{4}$ porównań + ***n*** aby scalić.