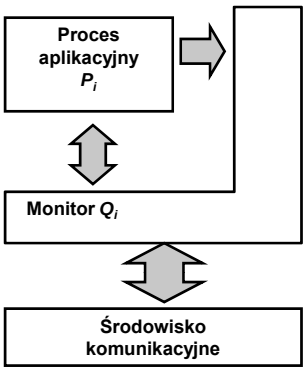
Czas wirtualny, złożoność algorytmów

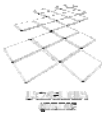
Monitor



Z każdym procesem P_i skojarzony jest **proces monitora Q_i**.

Cechy monitora

- Monitor może odczytywać (obserwować) zmienne lokalne procesu
- Monitor może obserwować i kontrolować zdarzenia komunikacyjne
- Monitor nie ma natomiast możliwości zmiany stanu procesu przez przypisanie jego zmiennym lokalnym nowych wartości



Konwencja zapisu algorytmów

Typy komunikatów:

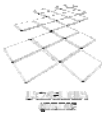
- aplikacyjne
- kontrolne
- sygnały
- pakiety

Wspólne atrybuty:

- identyfikator typu komunikatu
- identyfikator komunikatu
- identyfikator nadawcy
- identyfikator odbiorcy

typ FRAME

```
type FRAME is record of
   tag: ... /* pole identyfikatora typu */
   mId: ... /* pole identyfikatora wiadomości */
   sId: ... /* pole identyfikatora nadawcy */
   rId: ... /* pole identyfikatora odbiorcy */
end record
```

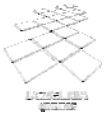



typ MESSAGE

```
type MESSAGE extends FRAME is record of
                              ...
end record
```

typ CONTROL

```
type CONTROL extends FRAME is record of
                        ...
end record
```

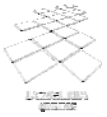



typ SIGNAL

```
type SIGNAL extends FRAME is record of
end record
```

typ PACKET

```
type PACKET extends FRAME is record of
     ...
     data: MESSSAGE
end record
```

Czas wirtualny (1)

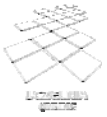
Zegary realizowane w systemach asynchronicznych mają stanowić aproksymację czasu rzeczywistego.

Aproksymacja taka uwzględnia jedynie zachodzące w systemie zdarzenia i dlatego czas ten nazywany jest **czasem wirtualnym** (**logicznym**).

Czas wirtualny (2)

W odróżnieniu od czasu rzeczywistego upływ czasu wirtualnego nie jest więc autonomiczny, a zależy od występujących w systemie zdarzeń i stąd określone wartości czasu wirtualnego mogą nigdy nie wystąpić.

Czas wirtualny wyznacza się za pomocą **zegarów logicznych** (ang. *logical clocks*).



Zegar logiczny - definicja

Zegar logiczny systemu rozproszonego jest funkcją $\mathcal{T}$: $\Lambda \rightarrow \mathcal{Y}$, odwzorowującą zbiór zdarzeń Λ w zbiór uporządkowany $\mathcal{Y}$, taką że:

$$(E \mapsto E') \Rightarrow (\mathcal{T}(E) < \mathcal{T}(E')) \qquad (4.1)$$

gdzie $<$ jest relacją porządku na zbiorze $\mathcal{Y}$.

Należy zauważyć, że w ogólności relacja odwrotna nie musi być spełniona, tzn. $\mathcal{T}(E) < \mathcal{T}(E') \not\Rightarrow E \mapsto E'$.

Zegary logiczne - właściwości

- jeżeli zdarzenie E zachodzi przed E' w tym samym procesie, to wówczas wartość zegara logicznego odpowiadającego zdarzeniu E jest mniejsza od wartości zegara odpowiadającego zdarzeniu E'
- w przypadku przesyłania wiadomości M, czas logiczny przyporządkowany zdarzeniu nadania wiadomości M jest zawsze mniejszy niż czas logiczny przyporządkowany zdarzeniu odbioru tej wiadomości

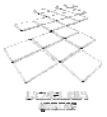
Zegar skalarny – definicja

Jeżeli przeciwdziedzina $\mathcal{Y}$ funkcji zegara logicznego jest zbiorem liczb naturalnych $\mathbb{N}$ lub rzeczywistych $\mathbb{R}$, to zegar nazywany jest **zegarem skalarnym**.

Realizacja zegarów skalarnych

Funkcja $\mathcal{T}(E)$ implementowana jest przez zmienne naturalne $clock_i$, $1 \leqslant i \leqslant n$, skojarzone z procesami P_i (monitorami Q_i).

Wartość zmiennej $clock_i$ reprezentuje w każdej chwili wartość funkcji $\mathcal{T}(E_i^k)$ odnoszącą się do ostatniego zdarzenia E_i^k jakie zaszło w procesie P_i, a tym samym reprezentuje upływ czasu logicznego w tym procesie.



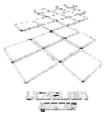
Alg. Lamporta (1)

```
type PACKET extends FRAME is record of
      clock : INTEGER
      data  : MESSAGE
end record


msgIn       : MESSAGE
pcktOut     : PACKET
clock_i     : INTEGER
d           : INTEGER
```

Alg. Lamporta (2)

```
1.  when e_send(P_i, P_j, msgOut: MESSAGE) do
2.     clock_i := clock_i + d
3.     pcktOut.clock := clock_i
4.     pcktOut.data := msgOut
5.    send(Q_i, Q_j, pcktOut)
6.  end when
```

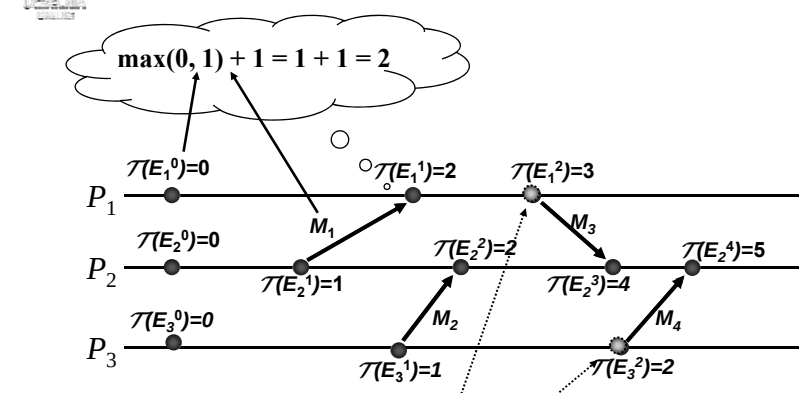



Alg. Lamporta (3)

```
7.  when e_receive(Q_j, Q_i, pcktIn: PACKET) do
8.    clock_i := max(clock_i, pcktIn.clock) + d
9.    msgIn := pcktIn.data
10.   deliver(P_j, P_i, msgIn)
11. end when

12. when e_internal(P_i, *) do
13.    clock_i := clock_i + d
14. end when
```

Przykład synchronizacji zegarów logicznych

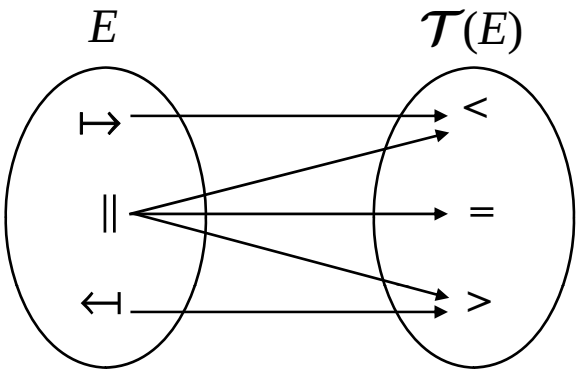


☑ $(E \mapsto E') \Rightarrow (\mathcal{T}(E) < \mathcal{T}(E'))$

☒ $(\mathcal{T}(E) < \mathcal{T}(E')) \Rightarrow (E \mapsto E')$

(19)

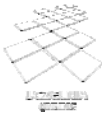
Relacja między zb. zdarzeń i zb. wartości zegara skalarnego



(20)

Zegar wektorowy - definicja

Zegarem wektorowym jest zegar logiczny, dla którego przeciwdziedzina funkcji $\mathcal{T}$, oznaczana dalej dla odróżnienia przez $\mathcal{T}^V$, jest **zbiorem *n*-elementowych wektorów liczb naturalnych lub rzeczywistych.**



Realizacja zegarów wektorowych

Funkcja $\mathcal{T}^V$ implementowana jest przez zmienne tablicowe $vClock_i$, $1 \leqslant i \leqslant n$, skojarzone z poszczególnymi procesami. Zmienna $vClock_i$ jest tablicą $[1..n]$ liczb naturalnych, odpowiadającą pewnej aproksymacji czasu globalnego z perspektywy procesu P_i.

W efekcie aktualna wartość tablicy $vClock_i$ odpowiada w każdej chwili wartości funkcji $\mathcal{T}^V(E_i^k)$ odnoszącej się do ostatniego zdarzenia, jakie zaszło w procesie P_i.

Alg. Matterna (1)

```
type PACKET extends FRAME is record of
      vClock : array [1..n] of INTEGER
      data   : MESSAGE
end record
```

```
msgIn   : MESSAGE
pcktOut : PACKET
vClock_i : array [1..n] of INTEGER
d       : INTEGER
k       : INTEGER
```

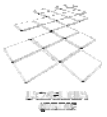

Alg. Matterna (2)

```
1.  when e_send(P_i, P_j, msgOut: MESSAGE) do
2.    vClock_i[i] := vClock_i[i] + d
3.    pcktOut.vClock := vClock_i
4.    pcktOut.data := msgOut
5.    send(Q_i, Q_j, pcktOut)
6.  end when
```

Alg. Matterna (3)

```
7.  when e_receive(Q_j , Q_i, pcktIn: PACKET) do
8.   vClock_i[i] := vClock_i[i] + d
9.   for all k ∈ {1, 2, ..., n} do
10.    vClock_i[k] := max(vClock_i[k], pcktIn.vClock[k])
11.  end for
12.  msgIn := pcktIn.data
13.  deliver(P_j, P_i, msgIn)
14. end when

15. when e_internal(P_i, *) do
16.  vClock_i[i] := vClock_i[i] + d
17. end when
```

Zegary wektorowe (1)

Twierdzenie 4.1

W każdej chwili czasu rzeczywistego

$$\forall i,j :: vClock_i[i] \geqslant vClock_j[i] \qquad (4.2)$$

gdzie zmienna $vClock_i[i]$ reprezentuje skalarny czas lokalny procesu P_i, a zmienna $vClock_j[i]$, $j \neq i$, aktualne wyobrażenie procesu P_j o bieżącym skalarnym czasie lokalnym procesu P_i.

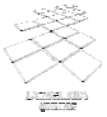
Relacje na etykietach wektorowych

$$vClock_i = vClock_j \Leftrightarrow \forall_k\ vClock_i[k] = vClock_j[k]$$
$$vClock_i \neq vClock_j \Leftrightarrow \exists_k\ vClock_i[k] \neq vClock_j[k]$$

$$vClock_i \leqslant vClock_j \Leftrightarrow \forall_k\ vClock_i[k] \leqslant vClock_j[k]$$
$$vClock_i \not\leqslant vClock_j \Leftrightarrow \exists_k\ vClock_i[k] > vClock_j[k]$$

$$vClock_i < vClock_j \Leftrightarrow vClock_i \leqslant vClock_j \wedge vClock_i \neq vClock_j$$
$$vClock_i \not< vClock_j \Leftrightarrow \neg(vClock_i \leqslant vClock_j \wedge vClock_i \neq vClock_j)$$

$$vClock_i \parallel vClock_j \Leftrightarrow vClock_i \not< vClock_j \wedge vClock_j \not< vClock_i)$$



Zegary wektorowe (2)

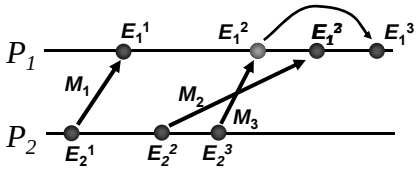
Twierdzenie 4.2

Niech $\mathcal{T}^V(E)$ oraz $\mathcal{T}^V(E')$ będą wartościami zegarów wektorowych zdarzeń E i E'. Wówczas:

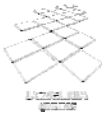
$$(E \mapsto E') \Leftrightarrow (\mathcal{T}^V(E) < \mathcal{T}^V(E')) \qquad (4.3)$$

Kanały FIFO

Kanały gwarantujące porządek odbioru wiadomości zgodny z kolejnością wysyłania będziemy nazywać **kanałami FIFO** (ang. *First-In-First-Out*).



(29)



Alg. Müllender'a (1)

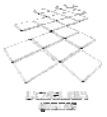
```
type PACKET extends FRAME is record of
      seqNo : INTEGER
      data  : MESSAGE
end record

msgIn        : MESSAGE
pcktOut      : PACKET
delayBuf_i   : array [1..n] of set of PACKET := ∅
seqNo_i      : array [1..n] of INTEGER := 0
delivNo_i    : array [1..n] of INTEGER := 0
delivered_i  : BOOLEAN
```

(30)

Alg. Müllender'a (2)

```
1.  when e_send(P_i, P_j, msgOut: MESSAGE) do
2.    pcktOut.data:= msgOut
3.    seqNo_i[j]:= seqNo_i[j] + 1
4.    pcktOut.seqNo:= seqNo_i[j]
5.    send(Q_i, Q_j, pcktOut)
6.  end when
```

Alg. Müllender'a (3)

```
7.  when e_receive(Q_j, Q_i, pcktIn : PACKET)  do
8.    if pcktIn.seqNo = delivNo_i[j] + 1
9.      then
10.       msgIn:= pcktIn.data
11.       deliver(P_j, P_i, msgIn)
12.       delivNo_i[j]:= delivNo_i[j] + 1
13.       delivered_i:= True
14.     else
15.       delayBuf_i[j]:= delayBuf_i[j] ∪ {pcktIn}
16.       delivered_i:= False
17.     end if
```

Alg. Müllender'a (4)

```
18.   while delivered_i do
19.     delivered_i:= False
20.     for all pckt ∈ delayBuf_i[j] do
21.       if pckt.seqNo = delivNo_i[j] + 1 then
22.         msgIn:= pckt.data
23.         deliver(P_j, P_i, msgIn)
24.         delivNo_i[j]:= delivNo_i[j] + 1
25.         delivered_i:= True
26.         delayBuf_i[j]:= delayBuf_i[j]\{pckt}
27.       end if
28.     end for
29.   end while
30. end when
```

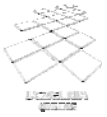

Cechy kanałów FIFO

- są pewnym mechanizmem synchronizacji wymaganym przez wiele aplikacji,
- ułatwiają znalezienie rozwiązania i konstrukcję algorytmów rozproszonych dla wielu problemów,
- ograniczają, w porównaniu z kanałami nonFIFO, współbieżność komunikacji, a tym samym efektywność przetwarzania.

Kanały typu FC

Kanały typu FC (ang. *Flush Channels*), łączą zalety kanałów FIFO i nonFIFO, (pewien stopień synchronizacji i współbieżnej komunikacji).

mechanizmy (operacje) komunikacji	zdarzenia	wiadomości
$send^t$ (ang. *two-way-flush send*)	e_send^t	M^t
$send^f$ (ang. *forward-flush send*)	e_send^f	M^f
$send^b$ (ang. *backward-flush-send*)	e_send^b	M^b
$send^o$ (ang. *ordinary send*)	e_send^o	M^o



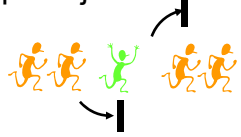
Wyprzedzanie wiadomości

Powiemy, że wiadomość M' **wyprzedza** wiadomość M w kanale $C_{i,j}$, jeżeli wiadomość M została wysłana przez P_i wcześniej niż M', lecz proces P_j najpierw odebrał wiadomość M'.

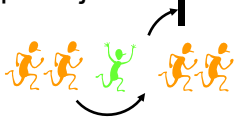
Typy wiadomości w kanałach *FC*

Wiadomość M^t typu ***TF***
(ang. *two-way-flush-send*)
operacja $send^t$

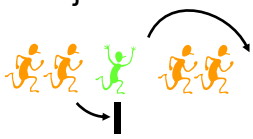
FIFO



Wiadomość M^f typu ***FF***
(ang. *forward-flush-send*)
operacja $send^f$

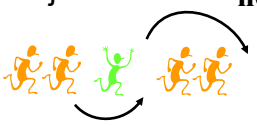


Wiadomość M^b typu ***BF***
(ang. *backward-flush-send*)
operacja $send^b$



Wiadomość M^o typu ***OF***
(ang. *ordinary-send*)
operacja $send^o$

nonFIFO



Implementacja kanałów *FC*

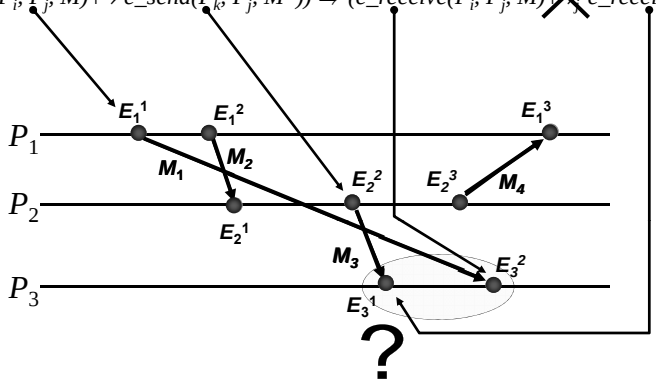
Kanały typu *FC* mogą być implementowane z użyciem różnych mechanizmów:

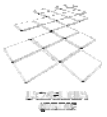
- selektywnego rozgłaszania
- liczników
- potwierdzeń
- …

Środowisko zachowujące uporządkowanie przyczynowe

(4.10)

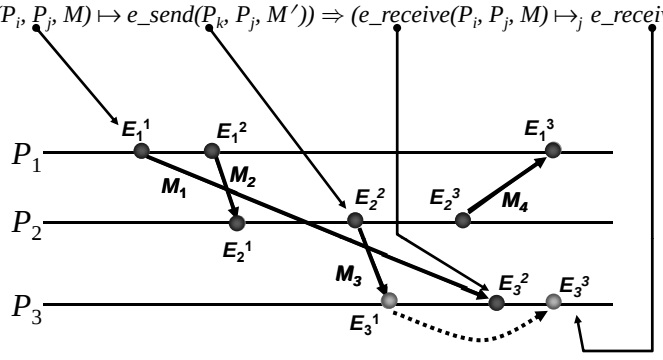
$$(e\_send(P_i, P_j, M) \mapsto e\_send(P_k, P_j, M')) \Rightarrow (e\_receive(P_i, P_j, M) \mapsto_j e\_receive(P_k, P_j, M'))$$





Środowisko zachowujące uporządkowanie przyczynowe

$$(e\_send(P_i, P_j, M) \mapsto e\_send(P_k, P_j, M')) \Rightarrow (e\_receive(P_i, P_j, M) \mapsto_j e\_receive(P_k, P_j, M'))$$



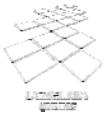
Funkcje kosztu – oznaczenia

Δ^{δ}_{A} zbiór wszystkich poprawnych danych wejściowych δ algorytmu *A*,

$\mathcal{Z}^{*}_{A}(\delta)$ **koszt wykonywania** algorytmu A dla danych δ, gdzie $\delta \in \Delta^{\delta}_{A}$ i $\mathcal{Z}^{*}_{A} : \Delta^{\delta}_{A} \rightarrow \mathbb{R}$

μ rozmiar danych wejściowych δ (rozmiar zadania), taki że $\mu = \mathcal{W}(\delta)$, gdzie $\mathcal{W} : \Delta^{\delta}_{A} \rightarrow \mathbb{N}$, jest zadaną funkcją.

W praktyce, zamiast kosztu $\mathcal{Z}^{*}_{A}(\delta)$ stosuje się zwykle jego oszacowanie w funkcji rozmiaru zadania $\mu = \mathcal{W}(\delta)$.



Funkcja kosztu - definicja

Funkcją kosztu wykonania algorytmu nazywać będziemy odwzorowanie

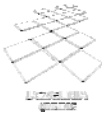
$$\mathcal{Z}_{A} : \Delta^{\mu}_{A} \rightarrow \mathbb{R} \qquad (4.11)$$

Gdzie Δ^{μ}_{A} jest zbiorem wszystkich poprawnych danych wejściowych o rozmiarze μ algorytmu *A*.

Funkcje kosztu wykonania algorytmów

Najczęściej stosowane jest odwzorowanie **pesymistyczne** (najgorszego przypadku), zdefiniowane w sposób następujący:

$$\mathcal{Z}_A(\mu) = \sup\{\mathcal{Z}^*_A(\delta) : \delta \in \Delta^\delta_A \wedge \mathcal{W}(\delta) = \mu\} \quad (4.12)$$



Rząd funkcji (1)

Niech f i g będą dowolnymi funkcjami odwzorowującymi $\mathbb{N}$ w $\mathbb{R}$.

Mówimy, że **funkcja f jest co najwyżej rzędu funkcji g**, co zapisujemy:

$$f = O(g) \quad (4.13)$$

jeżeli istnieje stała rzeczywista $c > 0$ oraz $n_0 \in \mathbb{N}$ takie, że dla każdej wartości $n > n_0$, $n \in \mathbb{N}$ zachodzi:

$$|f(n)| < c\,|g(n)| \quad (4.14)$$

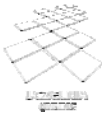
Rząd funkcji (2)

Niech f i g będą dowolnymi funkcjami odwzorowującymi $\mathbb{N}$ w $\mathbb{R}$.

Mówimy, że **funkcja f jest dokładnie rzędu funkcji g**, co zapisujemy:

$$f = \Theta(g) \tag{4.15}$$

jeżeli $f = O(g) \wedge g = O(f)$.



Rząd funkcji (3)

Niech f i g będą dowolnymi funkcjami odwzorowującymi $\mathbb{N}$ w $\mathbb{R}$.

Mówimy, że **funkcja f jest co najmniej rzędu funkcji g**, co zapisujemy:

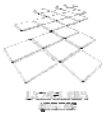
$$f = \Omega(g) \tag{4.16}$$

jeżeli $g = O(f)$.

Złożoność czasowa

W wypadku algorytmów rozproszonych, **złożoność czasowa** jest funkcją kosztu wykonania, wyrażoną przez liczbę kroków algorytmu do jego zakończenia przy założeniu, że:

- czas wykonywania każdego kroku (operacji) jest stały
- kroki wykonywane są synchronicznie
- czas transmisji wiadomości jest stały



Czasy przetwarzania lokalnego i transmisji

W analizie złożoności czasowej algorytmów rozproszonych przyjmuje się też na ogół, że

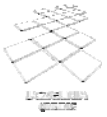
- czas przetwarzania lokalnego (wykonania każdego kroku) jest pomijalny (zerowy)
- czas transmisji jest jednostkowy

Złożoność komunikacyjna

Złożoność komunikacyjna jest funkcją kosztu wykonania algorytmu wyrażaną przez:

- liczbę pakietów (wiadomości) przesyłanych w trakcie wykonywania algorytmu do jego zakończenia
- sumaryczną długość (w bitach) wszystkich wiadomości przesłanych w trakcie wykonywania algorytmu

W konsekwencji wyróżniamy złożoność **pakietową** i **bitową**.



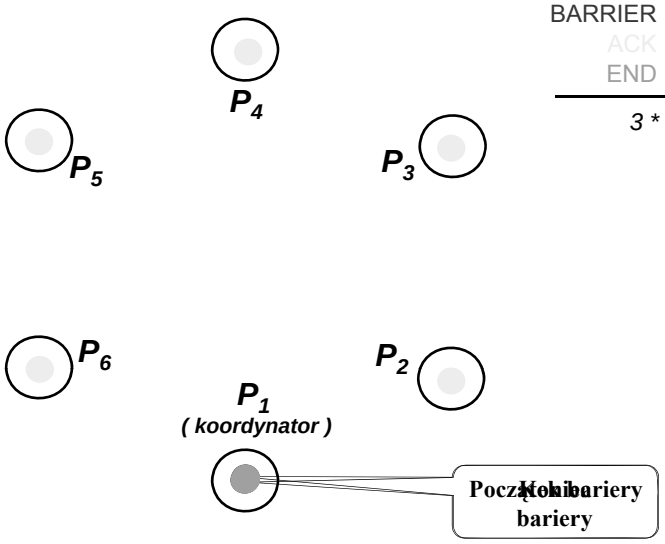
Przykład (1) - bariera

- Topologia połączeń – graf w pełni połączony
- Koordynator rozgłasza komunikat początku bariery
- Wszyscy uczestnicy odbierają komunikat i odsyłają potwierdzenia
- Po otrzymaniu potwierdzeń od wszystkich procesów, koordynator rozsyła komunikat końca bariery

Przykład (1) - rysunek

BARRIER
ACK
END

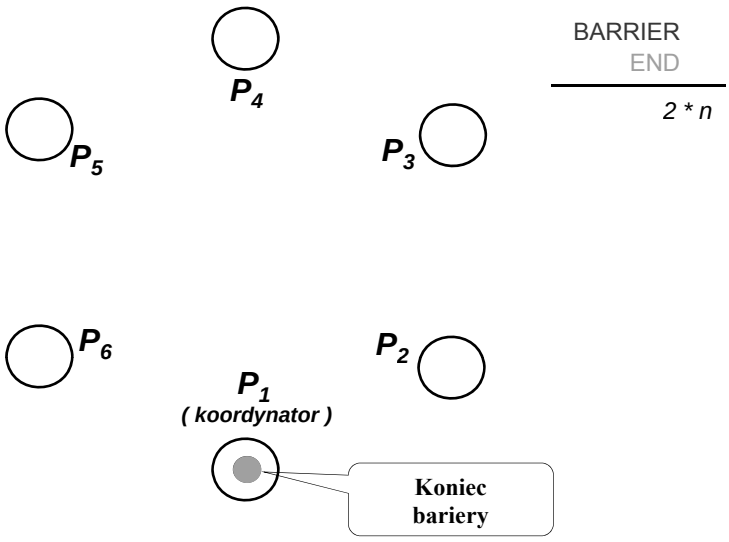
*3 * (n-1)*



Przykład (2) - bariera

- Topologia połączeń – pierścień logiczny
- Koordynator wysyła komunikat początku bariery
- Wszyscy uczestnicy odbierają komunikat i przesyłają go dalej
- Po otrzymaniu komunikatu rozpoczynającego operację bariery, koordynator przesyła komunikat końca bariery

Przykład (2) - rysunek



Warunki poprawności

Analizę poprawności algorytmu rozproszonego (procesu rozproszonego) dekomponuje się zwykle na analizę jego bezpieczeństwa i żywotności.

- właściwość **bezpieczeństwa** (ang. *safety*, *consistency*)
- właściwość **żywotności (postępu)** (ang. *liveness*, *progress*)