# Programowanie Współbieżne

Semafory i bufory

# Historia

Mechanizm semaforów gwarantuje niepodzielny dostęp do zasobów. Rozwiązanie zaproponowane przez Holenderskiego matematyka Dijkstrę w 1965 r.

# Definicja typu semaforowego

- Semafor jest zmienną całkowitoliczbową
- przyjmuje wartości nieujemne
- Jedyne operacje, za wyjątkiem inicjacji, to
  - WAIT, P(holenderskie słowo *passeren* przejść, *proberen* próbować), opuszczanie
  - SIGNAL, V(*vrijmaken* – zwolnić, *verhogen* - zwiększyć), podnoszenie.
- Operacje te są nie podzielne.
- Inicjacja jest dokonywana poza procesami, które z niego korzystają
- Na semaforach nie wykonujemy operacji sprawdzania.

# Semafor ogólny Dijkstry

**P**:czekaj aż S>0; s:=S-1;
**V**:S:=S+1;

# Semafor ogólny

**P(S)**: jeśli S>0 to S:= S-1, w przeciwnym przypadku wstrzymaj działanie procesu wykonującego tę operację.

**V(S)**: jeśli są procesy wstrzymane w wyniku P(S), to wznów jeden z nich, w przeciwnym przypadku S:=S+1.

# Semafor binarny

Przyjmuje wartości 1 lub 0.

**PB(S)**: jeśli S=1 to S:=0, w przeciwnym przypadku wstrzymaj działanie procesu wykonującego tę operację
**VB(S)**: jeśli są procesy wstrzymane w wyniku PB(S), to wznów jeden z nich, w przeciwnym wypadku S:=1;

Zazwyczaj wykonanie VB(S) dla S=1 kończy się błędem

# Semafor dwustronnie ograniczony

**PD(S)**: jeśli S>0 to S:=S-1 w przeciwnym wypadku czekaj

**VD(S)**: jeśli S<k to S:=S+1 w przeciwnym przypadku czekaj

# Semafor OR

**$P_{OR}(S_1,S_2)$**: jeśli $S_1>0$ lub $S_2>0$ to $S_k$-1 (k=1 lub k=2) w przeciwnym przypadku czekaj

**$V_{OR}(S_1,S_2)$**:$S_k:=S_k+1$

# Semafor AND

**$P_{AND}$ ($S_1$,$S_2$)**: jeśli $S_1>0$ i $S_2> 0$ to $S_1:=S_1-1$ i $S_2:=S_2-1$ w przeciwnym przypadku  czekaj

**$V_{AND}$ ($S_k$)**:$S_k:=S_k+1$

# Semafor uogólniony

**PG(S,t)**: jeśli S>=t to S:=S-t w przeciwnym przypadku czekaj

**VG(S,t)**: S:=S+t

# Semafory Agerwalii

**PE($S_1$,...$S_n$,$S'_{n+1}$ ,...,$S'_{n+m}$ )** powoduje zawieszenie procesu do chwili, gdy dla wszystkich $S_k$(k=1,...,n) będzie spełnione $S'_i$=0;
for i:=1 to n do $S_i$:=$S_i$-1;

**VE($S_1$ ,...,$S_r$)**:for i:=1 to r do $S_i$:=$S_i$+1;

# Sekcja krytyczna z zastosowaniem semaforów

```
program wykluczanie_sem;


procedure p1;
begin
   repeat
      wait(s); {P(s)}
      kryt1;
      signal(s); {V(s)}
      lok1;
   until false
end;
```

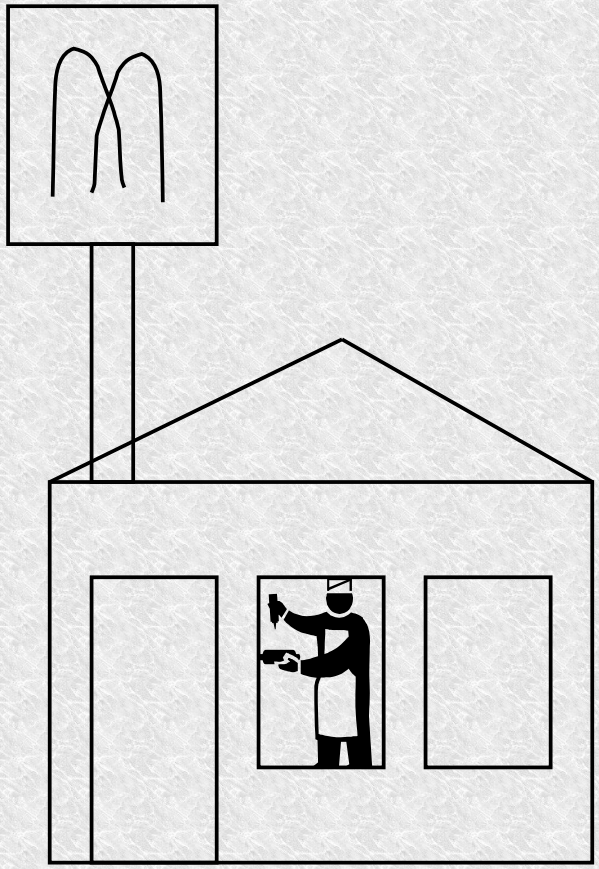
```
Procedure p2;
begin
   repeat
      wait(s);
      kryt2;
      signal(s);
      lok2;
   until false;
end;

BEGIN
   s:=1;
   cobegin;
      p1;p2;
   coend;
END.
```

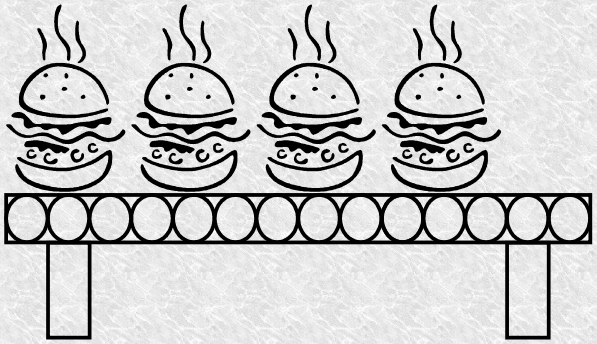
# Wykluczanie n procesów

```
Program wykluczanie_sem_n
const n=5; {5 liczba procesów}
var s:semaphore;
procedure proes (i:integer);
begin
   repeat
      wait(s);
      kryt;
      signal(s);
      lok;
until false
end;

BEGIN
s:=1;
cobegin
   proces(1);
   proses(2);
   ...
   proces(n);
coend
END.
```

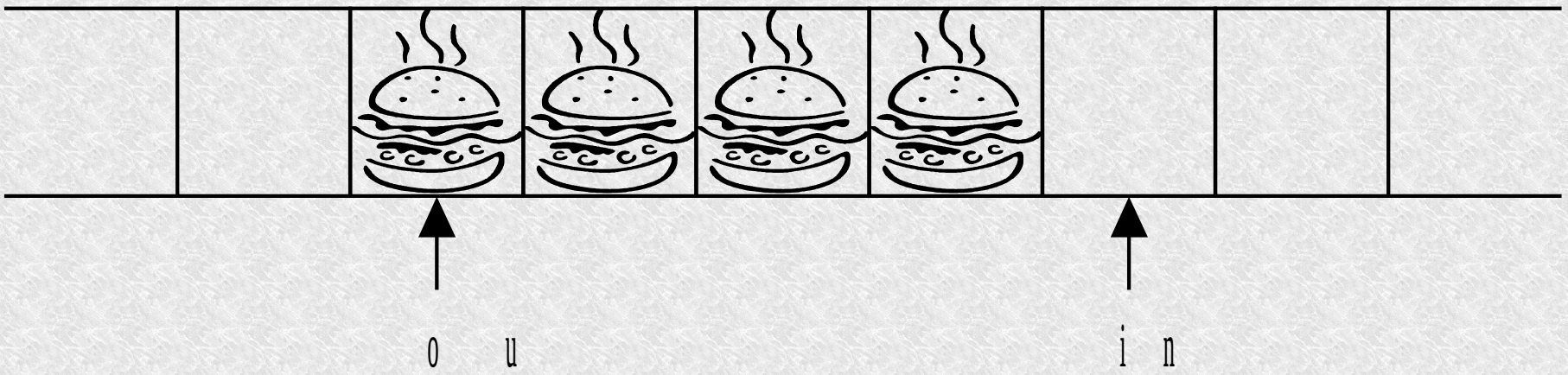
To rozwiązanie dopuszcza możliwość zagłodzenia. Semafory w założeniu nie sprawdzają czy ktoś już był czy nie, można powiedzieć że są one losowo dopuszczane do sekcji krytycznej.

# Problem producenta i konsumenta





# Bufor nieskończony




Odbywać się będą dwie czynności

```
repeat
   wyprodukuj rekord v;
   b[in]:=v;
   in:=in+1;
until false;
```

```
repeat
   wait until in >
out;
   w:=b[out];
   out:=out+1;
   skonsumuj rekord w;
until false;
```

# PK – bufor nieograniczony

```
Program producent_konsument;
var n:semaphore; {ogólny}
procedure producent;
begin
   repeat
      produkuj;
      wloz;
      signal(n);
   until false;
end;
```

```
procedure
konusemuent;
begin
   repeat
      wait(n)
      pobierz;
      konsumuj;
   until false
end;

BEGIN
   n:=0;
   cobegin
      producent;
konsument;
   coend;
END.
```

# PK – ze strefą krytyczną

```
program producent_konsument;
var n:semaphore;
    s:semaphore;
Procedure producent;
begin
   repeat
      produkuj;
      wait(s);
      włóż;
      signal(s);
      signal(n);
   until false
end;
```

```
Procedure konsument;
begin
   repeat
      wait(n);
      wait(s);
      pobierz;
      signal(s);
      konsumuj;
   until false;
end;

BEGIN
   n:=0;
   s:=1;
cobegin

producent;konsument;
coend;
END.
```

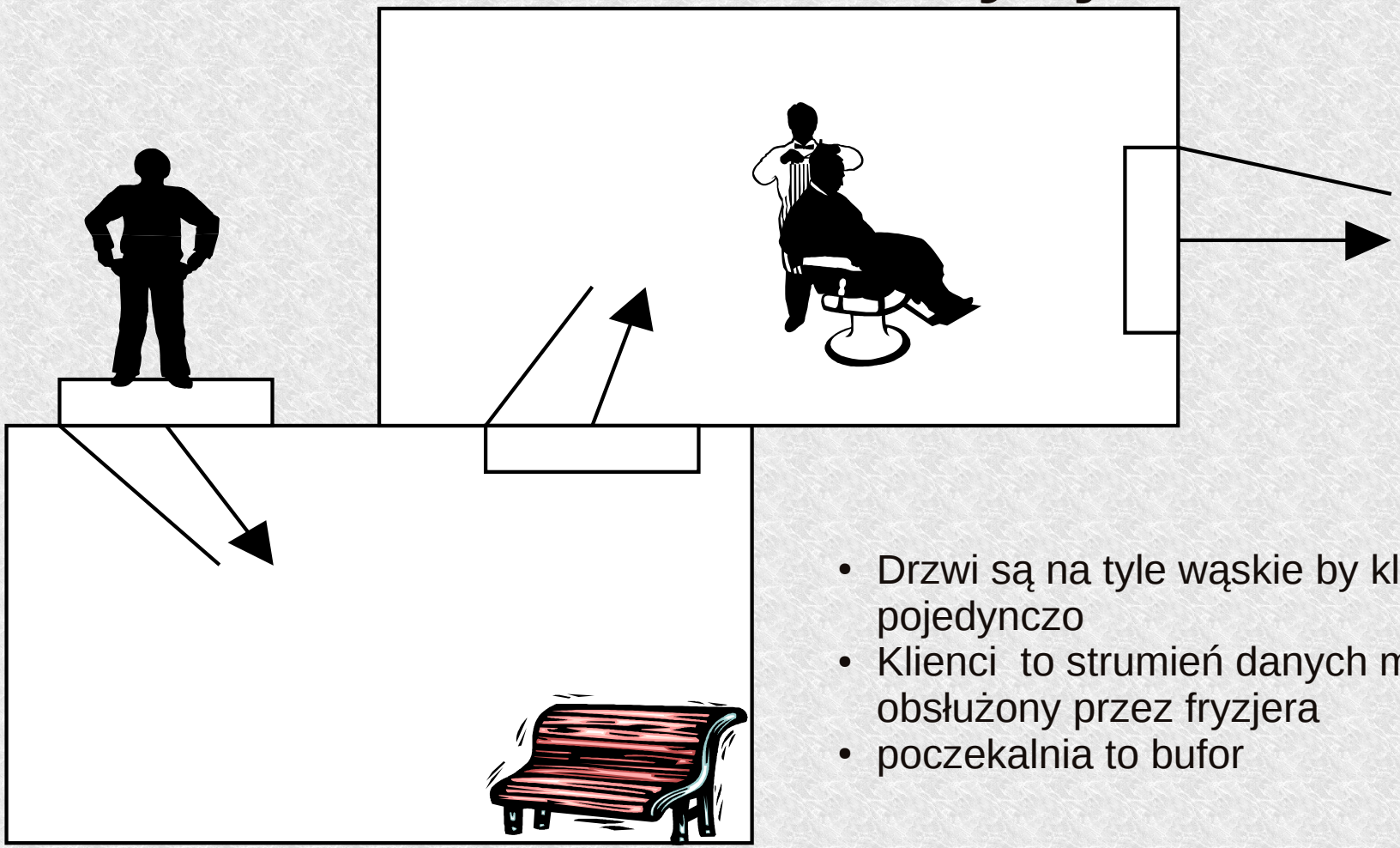
Uwaga na kolejność, odwrotnie może dojść do utraty żywotności

# PK – semafory binarne

```
Program producent_konsument;
var   n:integer;
      s:semaphore; {b}
opoznij:semaphore; {b}

Procedure producent;
begin
   repeat
   produkuj;
   wait(s);
   włóż;
   n:=n+1;
   if n=1 then signal(opoznij);
   signal(s);
until false
end;
```

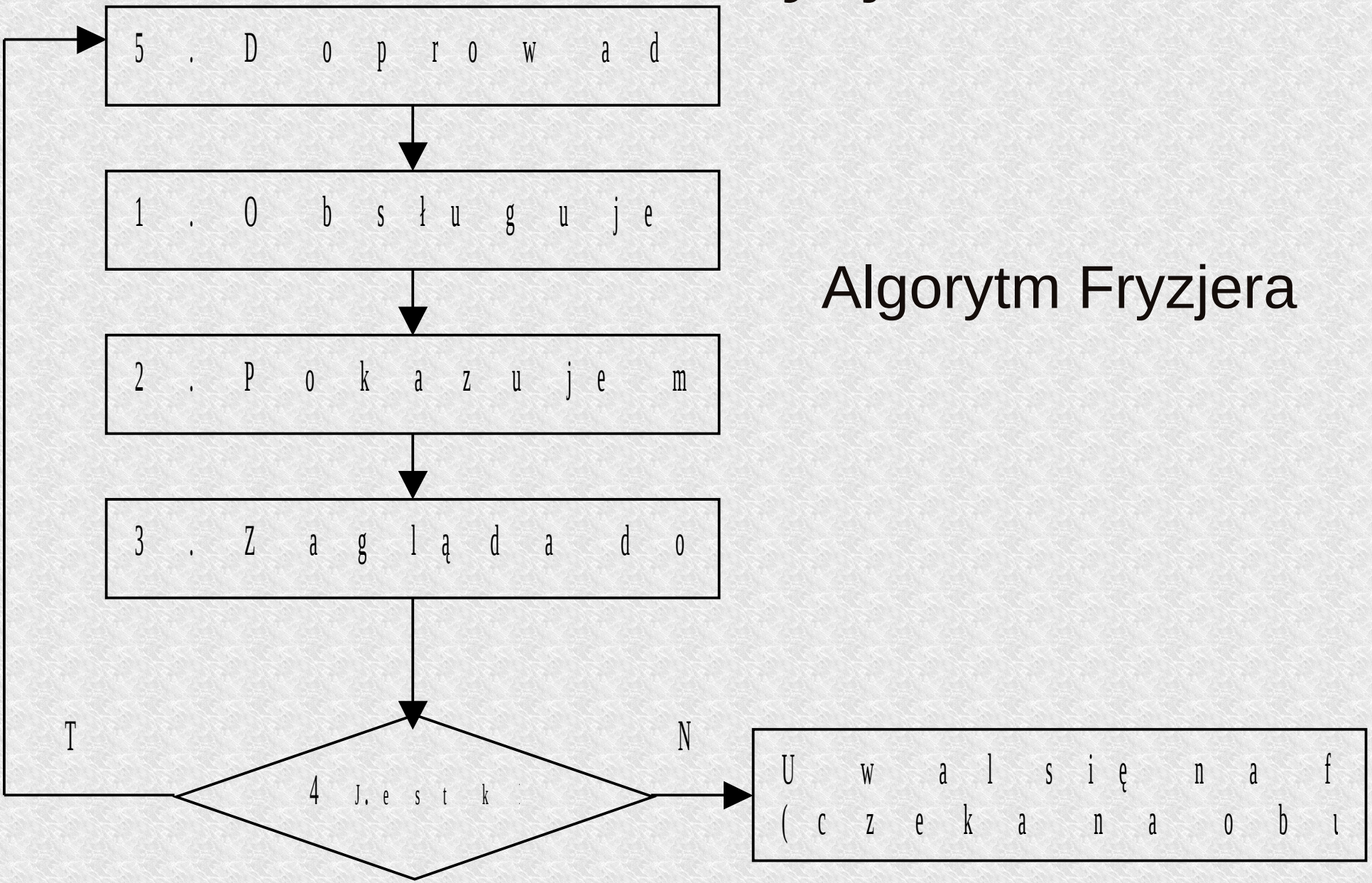
```
Procedure konsument;
var m:integer {lokalna zmienna!}
begin
   wait(opoznij);
   repeat
      wait(s);
      pobierz;
      n:=n-1;
      m:=n;
      signal(s);
      konsumuj;
      if m=0 then wait(opoznij)
   until false
end;

BEGIN
   n:=0;
   s:=1;
   opoznij:=0;
   cobegin;
      producent;konsument;
   coend;
END.
```

# Problem fryzjera



- Drzwi są na tyle wąskie by klienci wchodzili pojedynczo
- Klienci  to strumień danych mający być obsłużony przez fryzjera
- poczekalnia to bufor

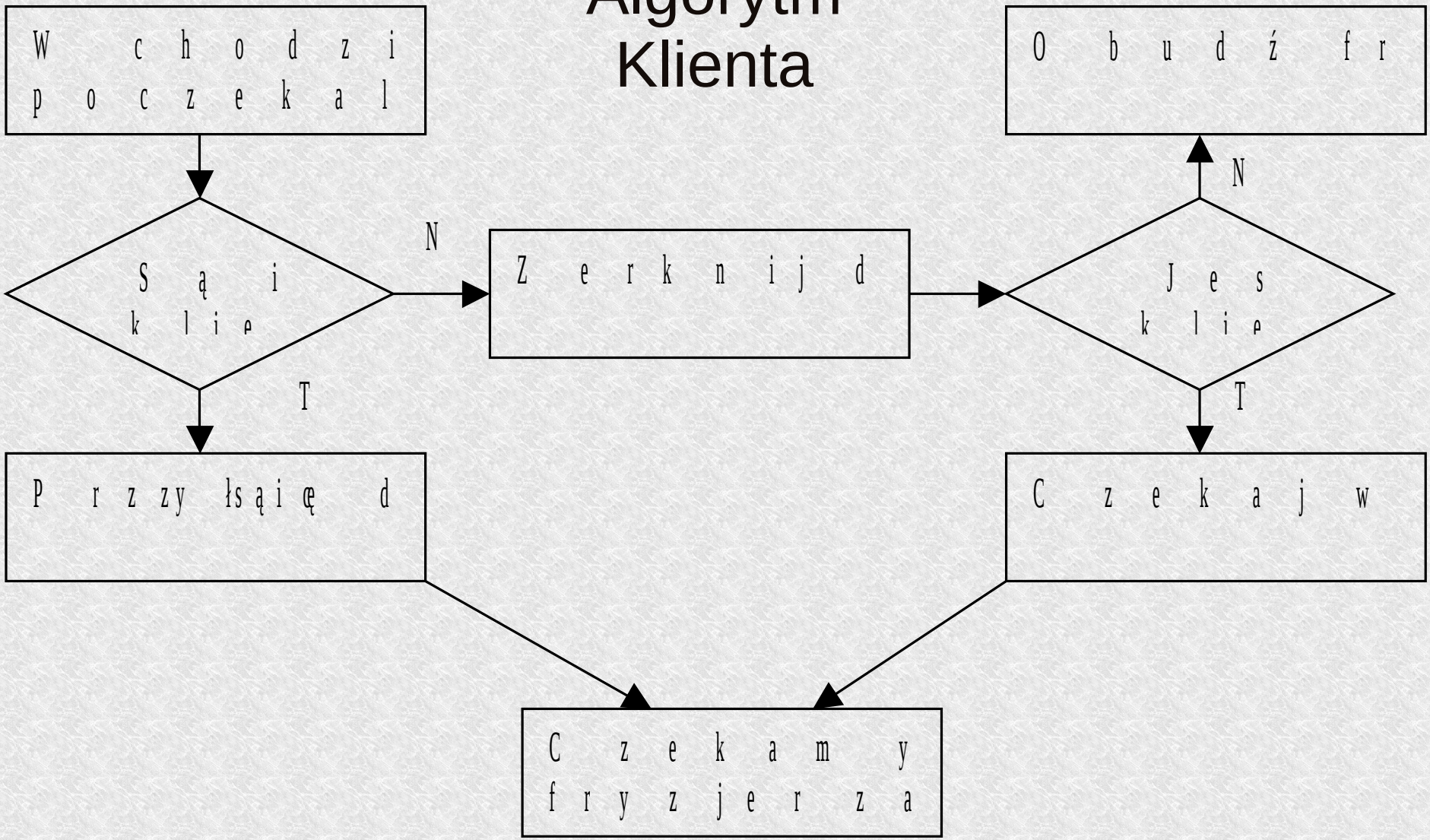
# Problem fryzjera




Algorytm Fryzjera

# Problem fryzjera

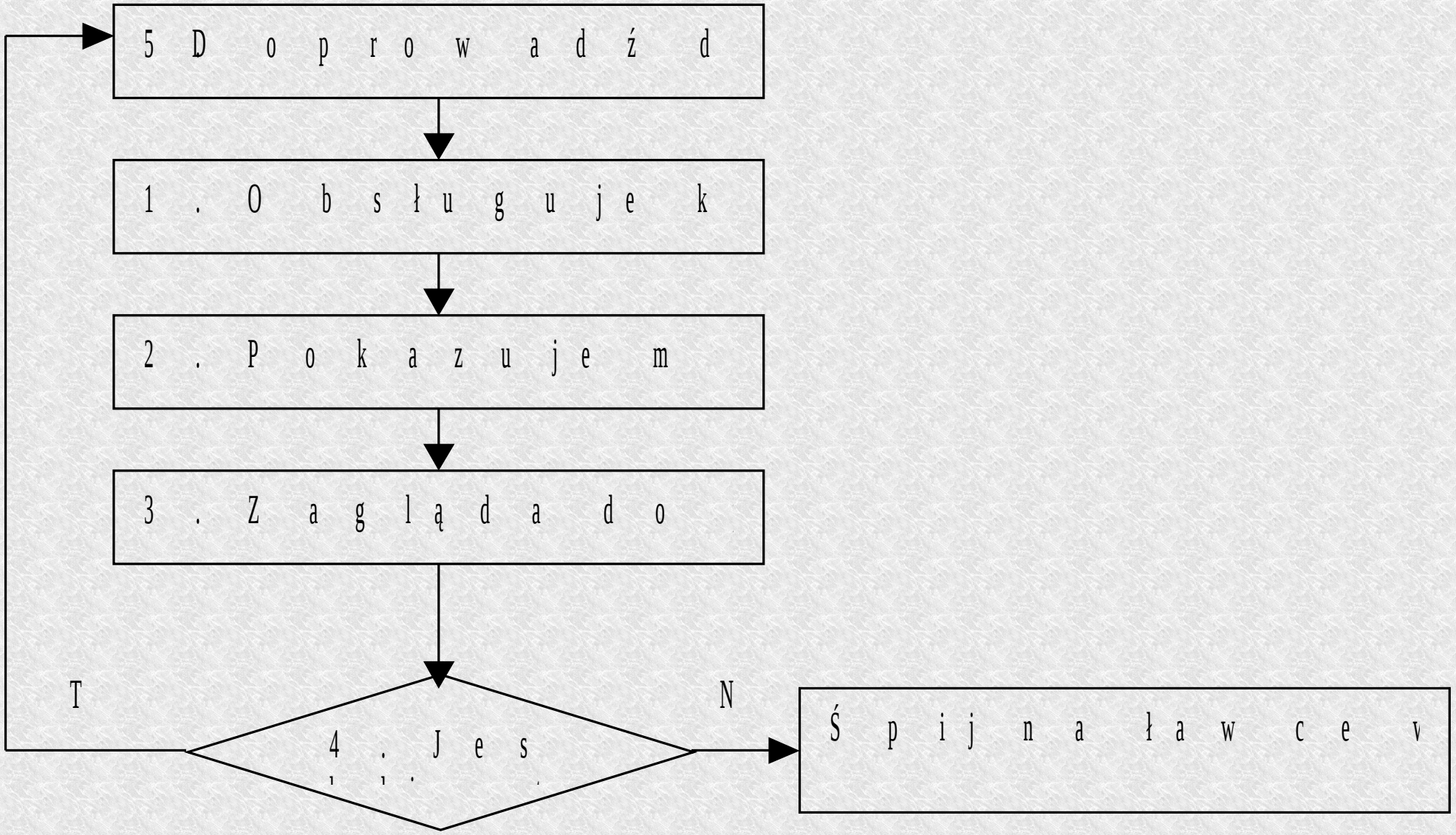
## Algorytm Klienta



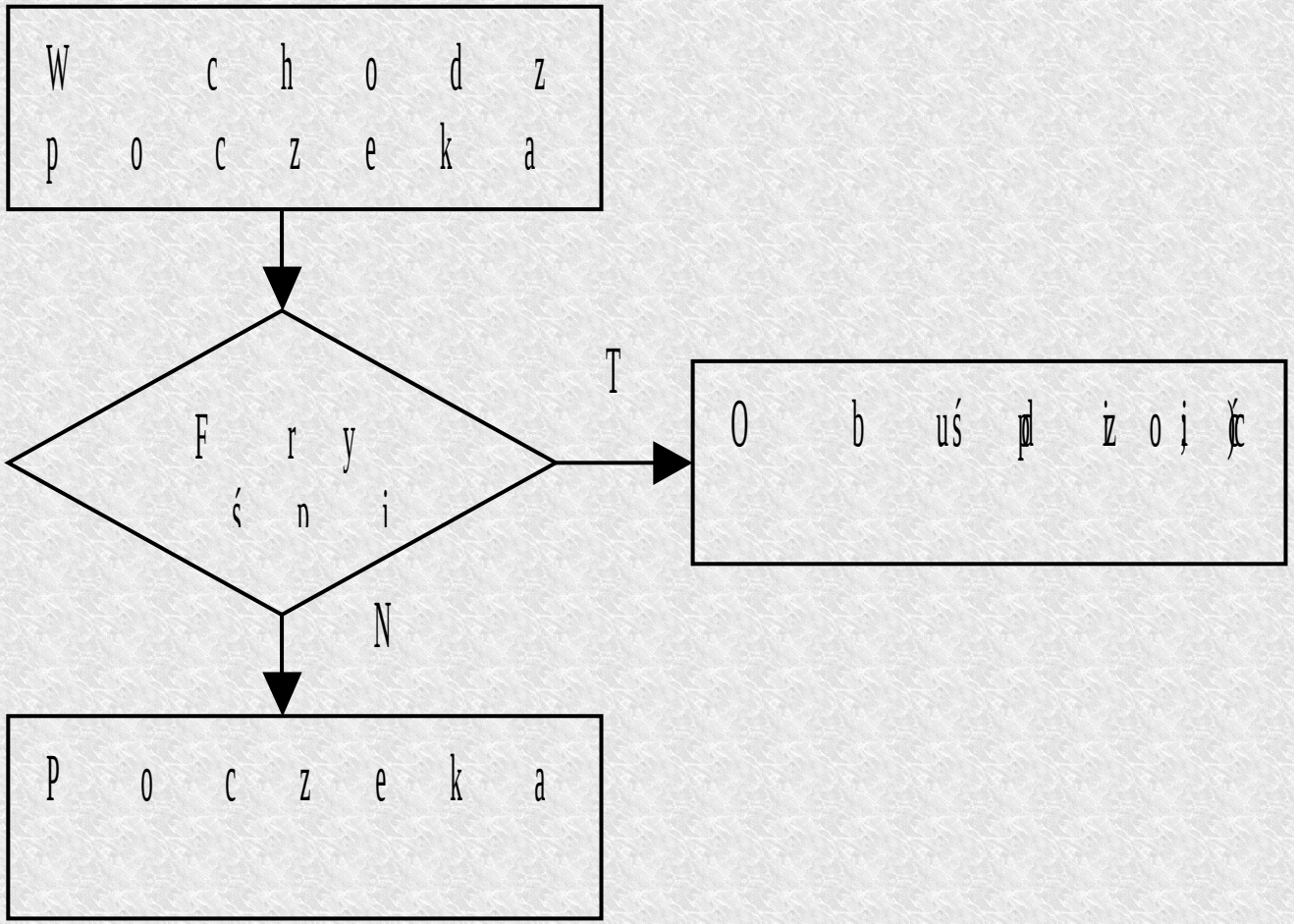
# Problem fryzjera

Gdy fryzjer będzie wyrabiał się z pracą to klienci zbyt często będą zaglądać

# Problem Fryzjera

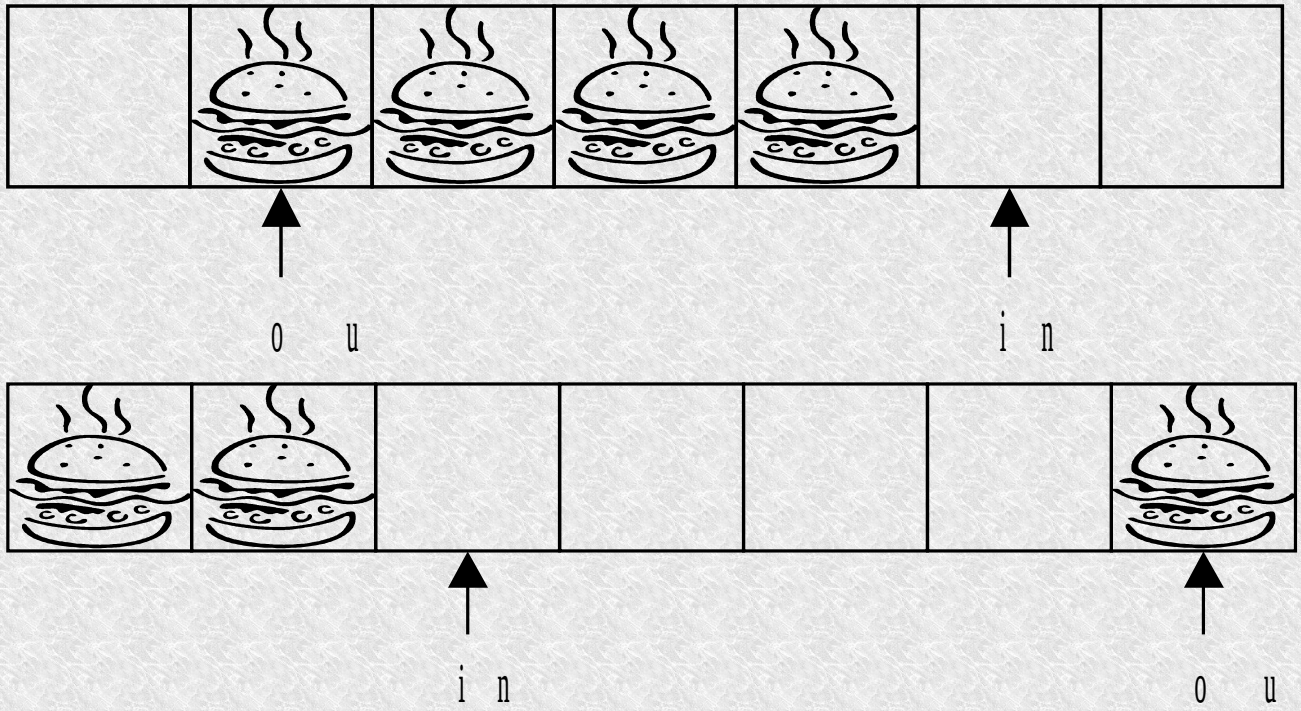


# Problem fryzjera



# Bufor ograniczony

**n** – długość tablicy, zakładamy że w buforze jest zawsze jedna komórka pusta aby **in=out** reprezentowało pusty bufor



# Bufor ograniczony

```
Producent –   produkuj;
              wait until((in >= out) and (in – out
< n) or (in < out) and (out –in > 1)
              włóż;
              if in = n then in:=1 else in:=in+1;

Konsument –  wait until (in <> out);
              pobierz;
              if out=n then out:=1 else out:=out+1;
              konsumuj;
```

# Bufor ograniczony

```
Program bufor_ograniczony;
const rozmiarbufora = 10;
var s:semaphore; {b}{dostęp do bufora}
    n:semaphore; { ilość elementów w buforze }
    e:semaphore; { ilość wolnych miejsc w buforze }

Procedure producent;
begin
   repeat
      produkuj;
      wait(e);
      wait(s);
      włóż;
      signal(s);
      signal(n);
   untili false
end;
```

```
Procedure konsument;
   repeat
      wait(n);
      wait(s);
      pobierz;
      signal(s);
      signal(e);
      konsumuj;
   until false;
end;
```

# Bufor ograniczony

```
BEGIN
   s:=1;
   n:=0;
   e:=rozmiarbufora;
   cobegin
      producent;konsument;
   coend;
END.
```

# Bufory rozłączne

- 2 lub więcej, zwykle tej samej długości
- Gdy jeden proces do pierwszego bufora ładuje dane to drugi czyta z drugiego, potem jest zmiana
- spora rozrzutność (średnio połowa pamięci się marnuje)
- po drugie proces odbierający dane musi poczekać aż będzie zmiana a ta zwykle jest po zapełnieniu bufora
- mają zastosowanie wszędzie tam gdzie bufory cykliczne są trudne do zrealizowania