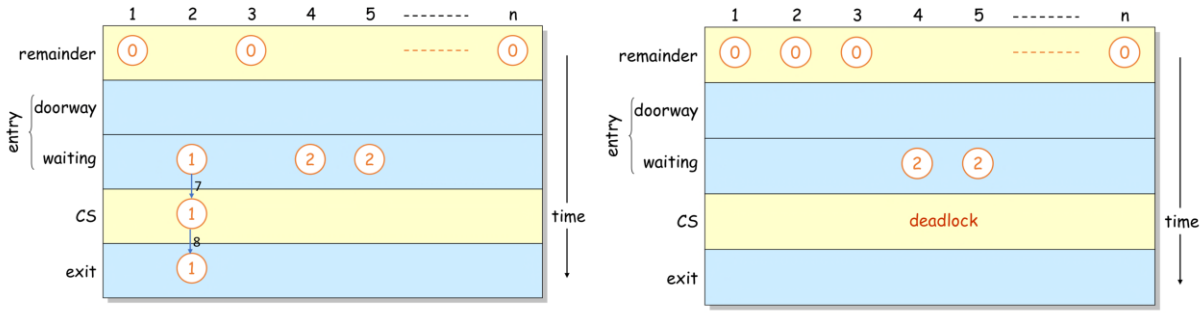
Algoritmo dei panettieri

Concetto base → quando vado in un negozio affollato prendo il ticket per avere l'ordine in cui posso fare gli acquisti; se non c' è nessuno, il ticket non interessa

<u>1 implementazione</u> → il processo prende il ticket più grande e poi entra nel for per controllare che non ci sia nessun altro in sezione critica

```
while (1){
  /*NCS*/
  number[i] = 1 + max {number[j] | (1 ≤ j ≤ N) except i}
  for j in 1 .. N except i {
    while (number[j] != 0 && number[j] < number[i]);
  }
  /*CS*/
  number[i] = 0;
}
```

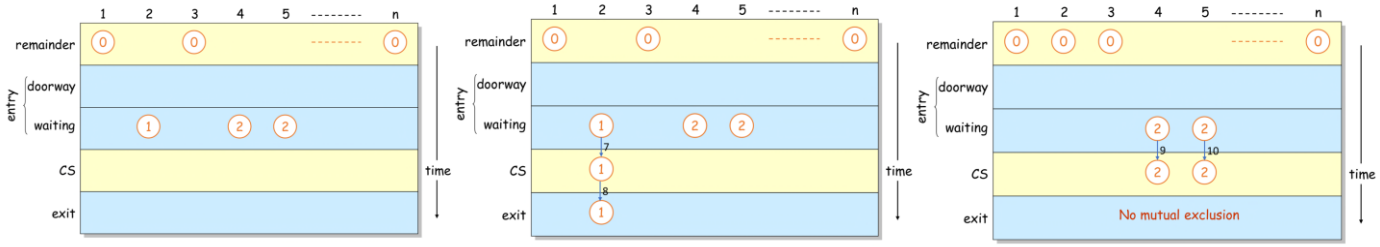
problema ⇨ non garantisce il no-deadlock perché se 2+ processi hanno lo stesso ticket nessuno avrà il numero più piccolo in assoluto (non si verifica mai la seconda condizione del while interno al for) quindi nessuno entrerà in sezione critica



<u>2 implementazione</u> → il deadlock si può risolvere cambiando il < con il <=

```
while (1){
  /*NCS*/
  number[i] = 1 + max {number[j] | (1 ≤ j ≤ N) except i}
  for j in 1 .. N except i {
    while (number[j] != 0 && number[j] <= number[i]);
  }
  /*CS*/
  number[i] = 0;
}
```

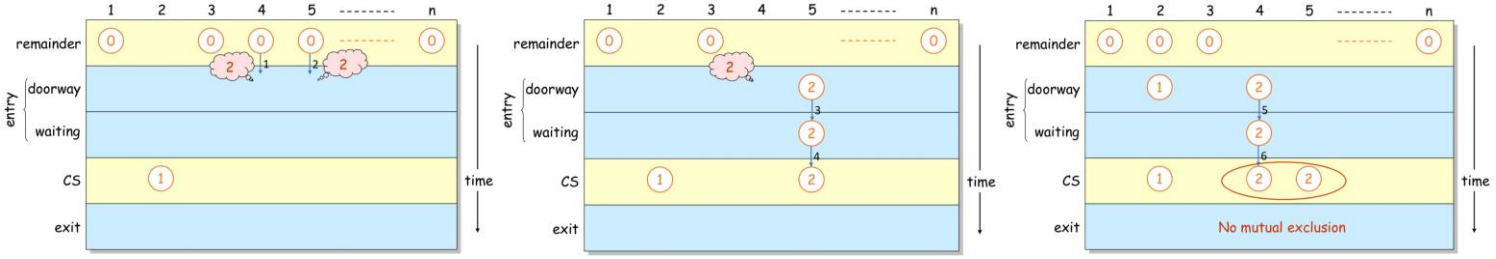
problema ⇨ non garantisce la mutua esclusione perché ora 2+ processi con lo stesso turno potrebbero entrare insieme in sezione critica



3 implementazione → uso l'ordinamento lessicografico; a parità di turno ha precedenza il processo con l'identificatore più basso

```
while (1){
  /*NCS*/
  number[i] = 1 + max {number[j] | (1 ≤ j ≤ N) except i}
  for j in 1 .. N except i {
    while (number[j] != 0 && (number[j],j) < (number[i],i));
  }
  /*CS*/
  number[i] = 0;
}
```

problema ⇨ non c'è atomicità delle istruzioni nel protocollo quindi la mutua esclusione non è garantita (calcolare il massimo e aggiungere 1 non è atomica); es. ho 2 processi che leggono contemporaneamente i loro turni e quindi decidono entrambi di essere il più basso



Implementazione finale→ simile a Dijkstra; uso una variabile choosing (segnala che sto scegliendo); posso anche avere 2+ processi con lo stesso turno perché verrà comunque considerato l'ordine lessicografico

```
while (1){
  /*NCS*/
  choosing[i] = true;
  number[i] = 1 + max {number[j] | (1 ≤ j ≤ N) except i}
  choosing[i] = false;
  for j in 1 .. N except i {
    while (choosing[j] == true);
    while (number[j] != 0 && (number[j],j) < (number[i],i));
  }
  /*CS*/
  number[i] = 0;
}
```

come funziona l'algoritmo per trovare il massimo? 3 variabili locali:

```
number[i] = 1 + max {number[j] | (1 ≤ j ≤ N)}
```

```
local1 = 0;
for local2 in 1 .. N {
      local3 = number[local2];
      if (local1 < local3)
            local1 = local3;
}
number[i] = 1 + local1
```

1. local1 → è il vero e proprio massimo che trovo
2. local2 → indice per scorrere tra i numeri
3. local3 → massimo relativo nel ciclo for

se c'è un numero massimo di turni assegnabili [es. i turni possono essere rappresentati solo con 10 bit (1023 turni + turno 0) → se il massimo trovato fosse 1023, il processo dovrebbe prendere il 1024 che però non è rappresentabile] ⇨ inserisco un do ... while (turno > massimo) per cui il processo non prende il turno non assegnabile e aspetta che gli altri processi abbiano finito per prendere il turno più basso successivamente

```
while (1){
  /*NCS*/
 while(number[i] == 0){
    choosing[i] = true;
    number[i] = (1 + max {number[j] | (1 ≤ j ≤ N) except i}) % MAXIMUM
    choosing[i] = false;
  }
  for j in 1 .. N except i {
    while (choosing[j] == true);
    while (number[j] != 0 && (number[j],j) < (number[i],i));
  }
  /*CS*/
  number[i] = 0;
}
```

Caratteristiche:

- i processi comunicano tramite variabili condivise
- read/write non sono atomiche
- ogni variabile condivisa appartiene a un processo che la scrive; gli altri la possono leggere
- nessun processo può scrivere contemporaneamente

versione client/server (uso message passing)

```
while (1){ //client thread
  /*NCS*/
  choosing = true; //doorway
  for j in 1 .. N except i {
    send(Pj,num);
    receive(Pj,v);
    num = max(num,v);
  }
  num = num+1;
  choosing = false;
  for j in 1 .. N except i {  //backery
    do{
      send(Pj,choosing);
      receive(Pj,v);
    }while (v == true);
    do{
      send(Pj,v);
      receive(Pj,v);
    }while (v != 0 && (v,j) < (num,i));
  }
  /*CS*/
  num = 0;
}
```

```
//global variable
//inizialization:
int num = 0;
boolean choosing = false;
// and process ip/ports
```

```
while (1){ //server thread
  receive(Pj,message);
  if (message is a number)
    send(Pj,num);
  else
    send(Pj,choosing);
}
```