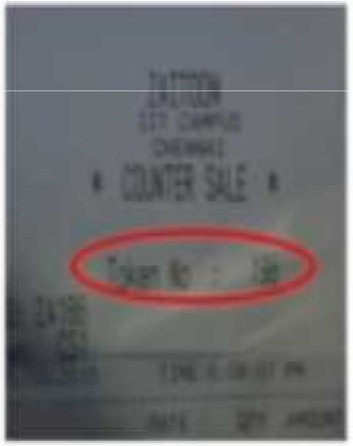
Lamport’s Bakery Algorithm for DSM

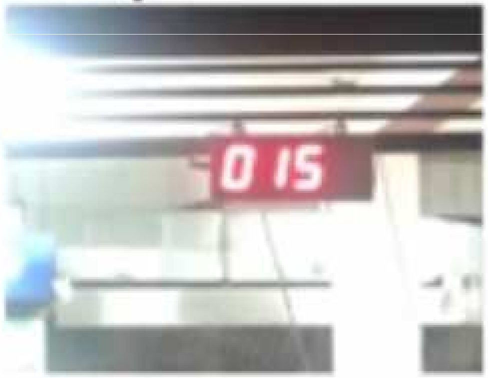
Bakery Algorithm for Mutually Exclusive access to the shared variables stored in DSM

Bakery Algorithm

- Synchronization between N > 2 processes
- By Leslie Lamport

Eat
when 196 displayed





wait your turn!!

http://research.microsoft.com/en-us/um/people/lamport/pubs/bakery.pdf

Simplified Bakery Algorithm

- Processes numbered 0 to N-1
- num is an array N integers (initially 0).
  - Each entry corresponds to a process

```
lock(i){
   num[i] = MAX(num[0], num[1], …., num[N-1]) + 1
   for(p = 0; p < N; ++p){
       while (num[p] != 0 and num[p] < num[i]);
   }
}
```

critical section

```
unlock(i){
   num[i] = 0;
}
```

Simplified Bakery Algorithm (example)

Processes numbered 0 to N-1

num is an array N integers (initially 0).

Each entry corresponds to a process

```
lock(i){
   num[i] = MAX(num[0], num[1], …., num[N-1]) + 1
   for(p = 0; p < N; ++p){
      while (num[p] != 0 and num[p] < num[i]);
   }
}
```

critical section

```
unlock(i){
   num[i] = 0;
}
```

P1	P2	P3	P4	**P5**
0	4	1	2	3

Simplified Bakery Algorithm

- Processes numbered 0 to N-1
- num is an array N integers (initially 0).
  - Each entry corresponds to a process

```
lock(i){
    num[i] = MAX(num[0], num[1], …., num[N-1]) + 1
    for(p = 0; p < N; ++p){
        while (num[p] != 0 and num[p] < num[i]);
    }
}
```

This is at the doorway!!!
It has to be atomic
to ensure two processes
do not get the same token

critical section

```
unlock(i){
    num[i] = 0;
}
```

Simplified Bakery Algorithm (why atomic doorway?)

- Processes numbered 0 to N-1
- num is an array N integers (initially 0).
  - Each entry corresponds to a process

This is at the doorway!!!
Assume it is not atomic

```
lock(i){
    num[i] = MAX(num[0], num[1], …., num[N-1]) + 1
    for(p = 0; p < N; ++p){
        while (num[p] != 0 and num[p] < num[i]);
    }
}
```

critical section

```
unlock(i){
    num[i] = 0;
}
```

P1	P2	P3	P4	**P5**
0	0	0	0	0

Simplified Bakery Algorithm (why atomic doorway?)

- Processes numbered 0 to N-1
- num is an array N integers (initially 0).
  - Each entry corresponds to a process

This is at the doorway!!!
Assume it is not atomic

```
lock(i){
    num[i] = MAX(num[0], num[1], …., num[N-1]) + 1
    for(p = 0; p < N; ++p){
        while (num[p] != 0 and num[p] < num[i]);
    }
}
```

critical section

```
unlock(i){
    num[i] = 0;
}
```

P1	P2	P3	P4	**P5**
0	3	1	2	2

Original Bakery Algorithm

- Without atomic operation assumptions
- Introduce an array of N Booleans: *choosing*, initially all values False.

```
lock(i){
    choosing[i] = True
    num[i] = MAX(num[0], num[1], …., num[N-1]) + 1
    choosing[i] = False
    for(p = 0; p < N; ++p){
        while (choosing[p]);
        while (num[p] != 0 and (num[p],p)<(num[i],i));
    }
}
```

doorway

critical section

```
unlock(i){
    num[i] = 0;
}
```

Choosing ensures that a process
Is not at the doorway
i.e., the process is not 'choosing'
a value for num

(a, b) < (c, d) which is equivalent to: (a < c) or ((a == c) and (b < d))

Original Bakery Algorithm (example)

- Without atomic operation assumptions
- Introduce an array of N Booleans: *choosing*, initially all values False.

```
lock(i){
    choosing[i] = True
    num[i] = MAX(num[0], num[1], …., num[N-1]) + 1
    choosing[i] = False
    for(p = 0; p < N; ++p){
        while (choosing[p]);
        while (num[p] != 0 and (num[p],p)<(num[i],i));
    }
}
```

doorway

critical section

```
unlock(i){
    num[i] = 0;
}
```

P1	P2	P3	P4	**P5**
0	0	1	2	2

(a, b) < (c, d) which is equivalent to: (a < c) or ((a == c) and (b < d))

Original Bakery Algorithm (example)

- Without atomic operation assumptions
- Introduce an array of N Booleans: *choosing*, initially all values False.

```
lock(i){
    choosing[i] = True
    num[i] = MAX(num[0], num[1], …., num[N-1]) + 1
    choosing[i] = False
    for(p = 0; p < N; ++p){
        while (choosing[p]);
        while (num[p] != 0 and (num[p],p)<(num[i],i));
    }
}
```

doorway

critical section

```
unlock(i){
    num[i] = 0;
}
```

P1	P2	P3	P4	**P5**
0	3	1	2	2

(a, b) < (c, d) which is equivalent to: (a < c) or ((a == c) and (b < d))

Original Bakery Algorithm (example)

- Without atomic operation assumptions
- Introduce an array of N Booleans: *choosing*, initially all values False.

```
lock(i){
   choosing[i] = True
   num[i] = MAX(num[0], num[1], ...., num[N-1]) + 1
   choosing[i] = False
   for(p = 0; p < N; ++p){
      while (choosing[p]);
      while (num[p] != 0 and (num[p],p)<(num[i],i));
   }
}
```

doorway

critical section

```
unlock(i){
   num[i] = 0;
}
```

P1	P2	P3	P4	**P5**
0	3	0	2	2

(a, b) < (c, d) which is equivalent to: (a < c) or ((a == c) and (b < d))

Original Bakery Algorithm (example)

- Without atomic operation assumptions
- Introduce an array of N Booleans: *choosing*, initially all values False.

```
lock(i){
    choosing[i] = True
    num[i] = MAX(num[0], num[1], …., num[N-1]) + 1
    choosing[i] = False
    for(p = 0; p < N; ++p){
        while (choosing[p]);
        while (num[p] != 0 and (num[p],p)<(num[i],i));
    }
}
```

doorway

critical section

```
unlock(i){
    num[i] = 0;
}
```

check p<i

P1	P2	P3	P4	**P5**
0	3	0	2	2

(a, b) < (c, d) which is equivalent to: (a < c) or ((a == c) and (b < d))

Original Bakery Algorithm (example)

- Without atomic operation assumptions
- Introduce an array of N Booleans: *choosing*, initially all values False.

```
lock(i){
   choosing[i] = True
   num[i] = MAX(num[0], num[1], …., num[N-1]) + 1
   choosing[i] = False
   for(p = 0; p < N; ++p){
      while (choosing[p]);
      while (num[p] != 0 and (num[p],p)<(num[i],i));
   }
}
```

doorway

critical section

```
unlock(i){
   num[i] = 0;
}
```

P1	P2	P3	P4	P5
0	3	0	0	2

(a, b) < (c, d) which is equivalent to: (a < c) or ((a == c) and (b < d))

Original Bakery Algorithm (example)

- Without atomic operation assumptions
- Introduce an array of N Booleans: *choosing*, initially all values False.

```
lock(i){
   choosing[i] = True
   num[i] = MAX(num[0], num[1], …., num[N-1]) + 1
   choosing[i] = False
   for(p = 0; p < N; ++p){
       while (choosing[p]);
       while (num[p] != 0 and (num[p],p)<(num[i],i));
   }
}
```

doorway

critical section

```
unlock(i){
   num[i] = 0;
}
```

P1	P2	P3	P4	**P5**
0	3	0	0	0

(a, b) < (c, d) which is equivalent to: (a < c) or ((a == c) and (b < d))

Original Bakery Algorithm (example)

- Without atomic operation assumptions
- Introduce an array of N Booleans: *choosing*, initially all values False.

```
lock(i){
    choosing[i] = True
    num[i] = MAX(num[0], num[1], ...., num[N-1]) + 1
    choosing[i] = False
    for(p = 0; p < N; ++p){
        while (choosing[p]);
        while (num[p] != 0 and (num[p],p)<(num[i],i));
    }
}
```

doorway

critical section

```
unlock(i){
    num[i] = 0;
}
```

P1	P2	P3	P4	**P5**
0	0	0	0	0

(a, b) < (c, d) which is equivalent to: (a < c) or ((a == c) and (b < d))

References

- Leslie Lamport's Bakery Algorithm Paper.
  http://lamport.azurewebsites.net/pubs/bakery.pdf
- NPTEL Video Lecture:
  https://www.youtube.com/watch?v=YHQxp-XduS0