
```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 0, slots = 1

i = 0

ProcPerPC = 2

j = 1

((0+1)%1, 0)

wait:

p0.0 = 0

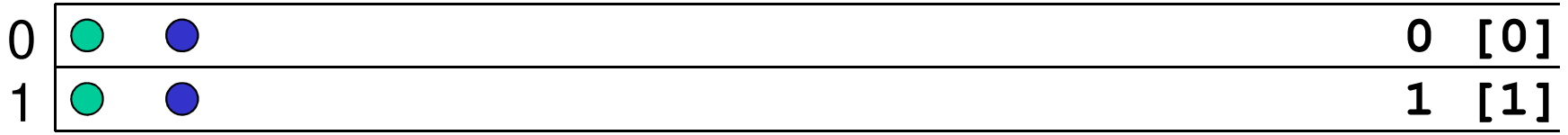
**p0.1 = 0**


```
void ComputeWait() {
        start = 0, slots = 1;          start = 0, slots = 1
        for(i=0; i<=MAXPV; i++) {      i = 0
                for(j=0; j<ProcPerPC[i]; j++){  ProcPerPC = 2
                        wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;  = (0 + 1) % 1 = 0
                start = 2 * start;     = 2 * 0 = 0
                slots = 2 * slots;     = 2 * 1 = 2
        }
}
```

wait:

p0.0 = 0

p0.1 = 0




```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                        wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 0, slots = 2

i = 1

ProcPerPC = 5

pv = 1:

wait:

...

p1.0 = 0

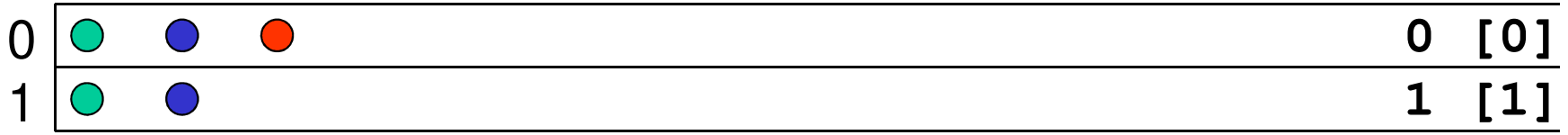
p1.1 = 1

p1.2 = 0

p1.3 = 1

p1.4 = 0

...




```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 0, slots = 2

i = 1

ProcPerPC = 5

**j = 0**

((0+**0**)%2, 1)

pv = 1:

wait:

...

**p1.0 = 0**

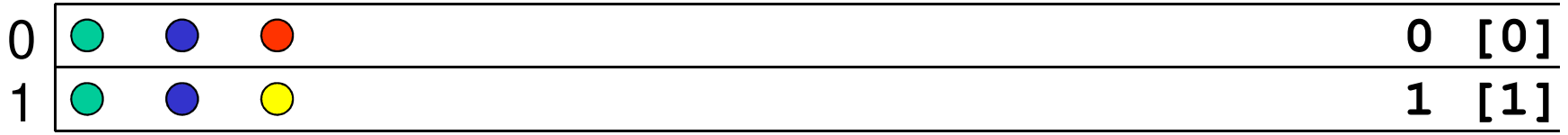
p1.1 = 1

p1.2 = 0

p1.3 = 1

p1.4 = 0

...




```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 0, slots = 2

ProcPerPC = 5

**j = 1**

((0+**1**)%2, 1)

pv = 1:

wait:

...

p1.0 = 0

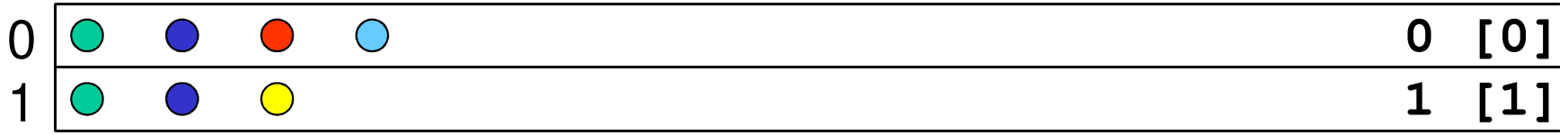
**p1.1 = 1**

p1.2 = 0

p1.3 = 1

p1.4 = 0

...




```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 0, slots = 2

ProcPerPC = 5

**j = 2**

((0+**2**)%2, 1)

pv = 1:

wait:

...

p1.0 = 0

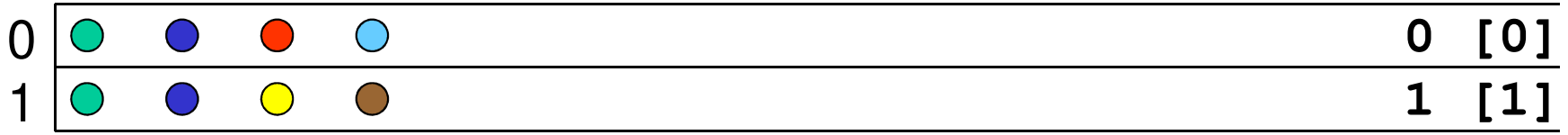
p1.1 = 1

**p1.2 = 0**

p1.3 = 1

p1.4 = 0

...




```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 0, slots = 2

ProcPerPC = 5

**j = 3**

((0+**3**)%2, 1)

pv = 1:

wait:

...

p1.0 = 0

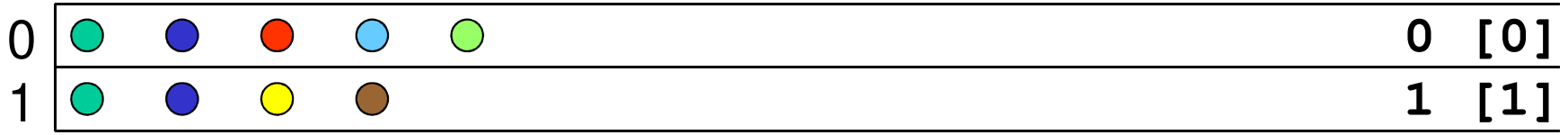
p1.1 = 1

p1.2 = 0

**p1.3 = 1**

p1.4 = 0

...




```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 0, slots = 2

ProcPerPC = 5

**j = 4**

((0+**4**)%2, 1)

pv = 1:

wait:

...

p1.0 = 0

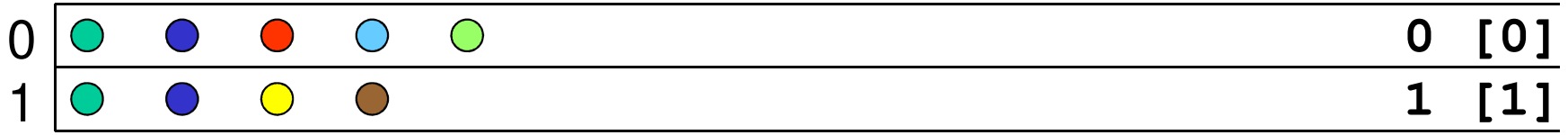
p1.1 = 1

p1.2 = 0

p1.3 = 1

**p1.4 = 0**

...




```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                        wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 0, slots = 2

start = ( 0 + 5 ) % 2 = 1

start = 2 * 1 = 2

slots = 2 * 2 = 4

pv = 1:

ProcPerPC = 5

wait:

...

p1.0 = 0

p1.1 = 1

p1.2 = 0

p1.3 = 1

p1.4 = 0

...


```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                        wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 2, slots = 4

i = 2

ProcPerPC = 1

pv = 2:

wait:

...

p2.0 = 1

...


```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                        wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 2, slots = 4

i = 2

ProcPerPC = 1

**j = 0**

((2+**0**)%4, 2)

pv = 2:

wait:

...

**p2.0 = 1**

...


```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                        wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = ( 2 + 1 ) % 4 = 3

start = 2 * 3 = 6

slots = 2 * 4 = 8

pv = 2:

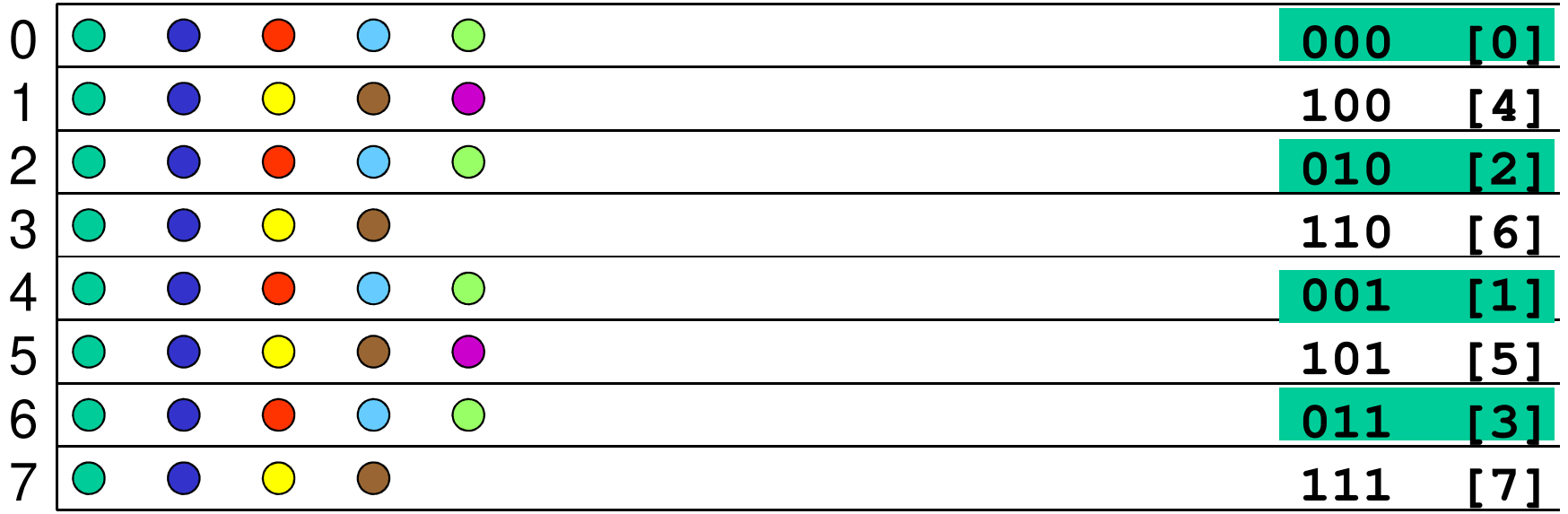
ProcPerPC = 1

wait:

...

p2.0 = 1

...




```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                        wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 6, slots = 8

i = 3

ProcPerPC = 1

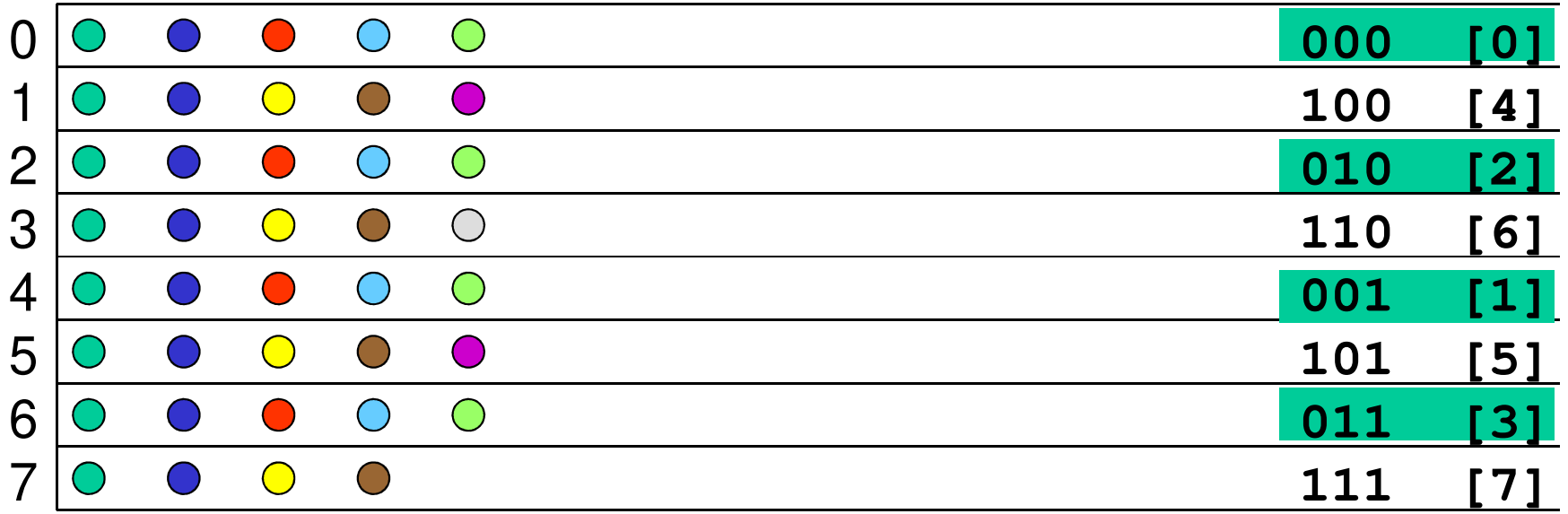
pv = 3: ○

wait:

...

p3.0 = 3

...




```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                        wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 6, slots = 8

i = 3

ProcPerPC = 1

**j = 0**

((6+**0**)%8, 3)

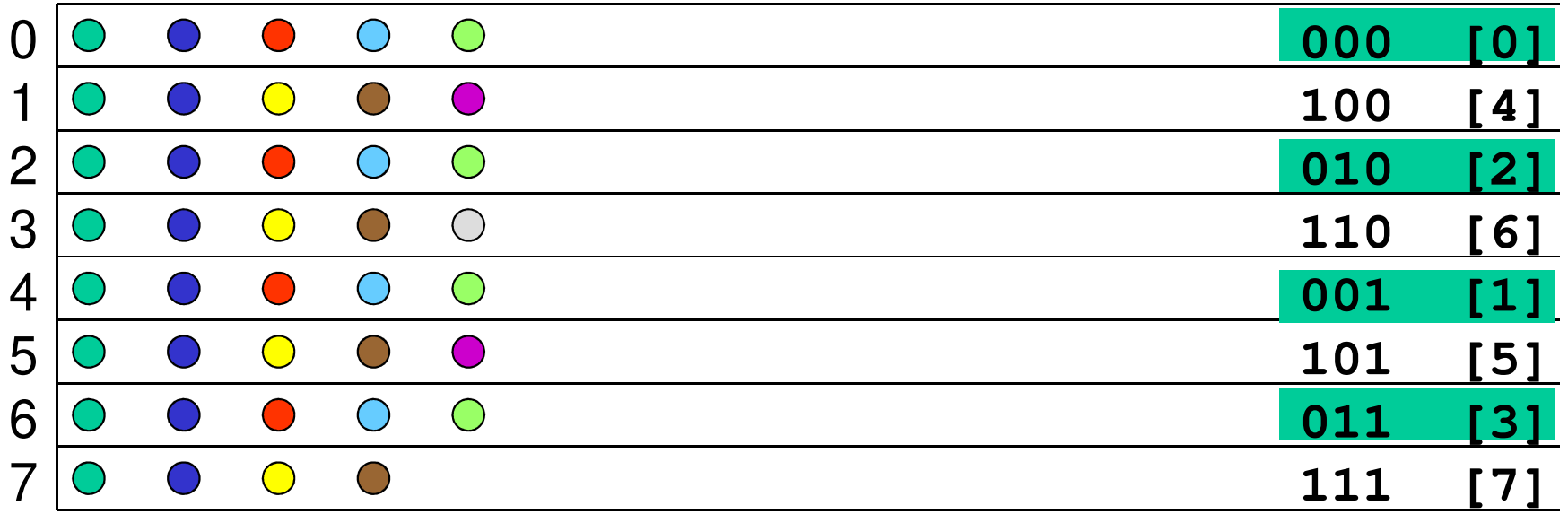
pv = 3: ⊗

wait:

...

**p3.0 = 3**

...




```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                        wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 6, slots = 8

i = 3

start = ( 6 + 1 ) % 8 = 7

= 2 * 7 = 14

= 2 * 8 = 16

pv = 3: ⊗

ProcPerPC = 1

wait:

...

p3.0 = 3

...

```
void ComputeWait() {
        start = 0, slots = 1;
        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                wait[i][j] = Rev.Bin.((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

start = 14, slots = 16

i = 4 (MAXPV = 4)

ProcPerPC = 1

j = 0

((14+0)%16, 4)

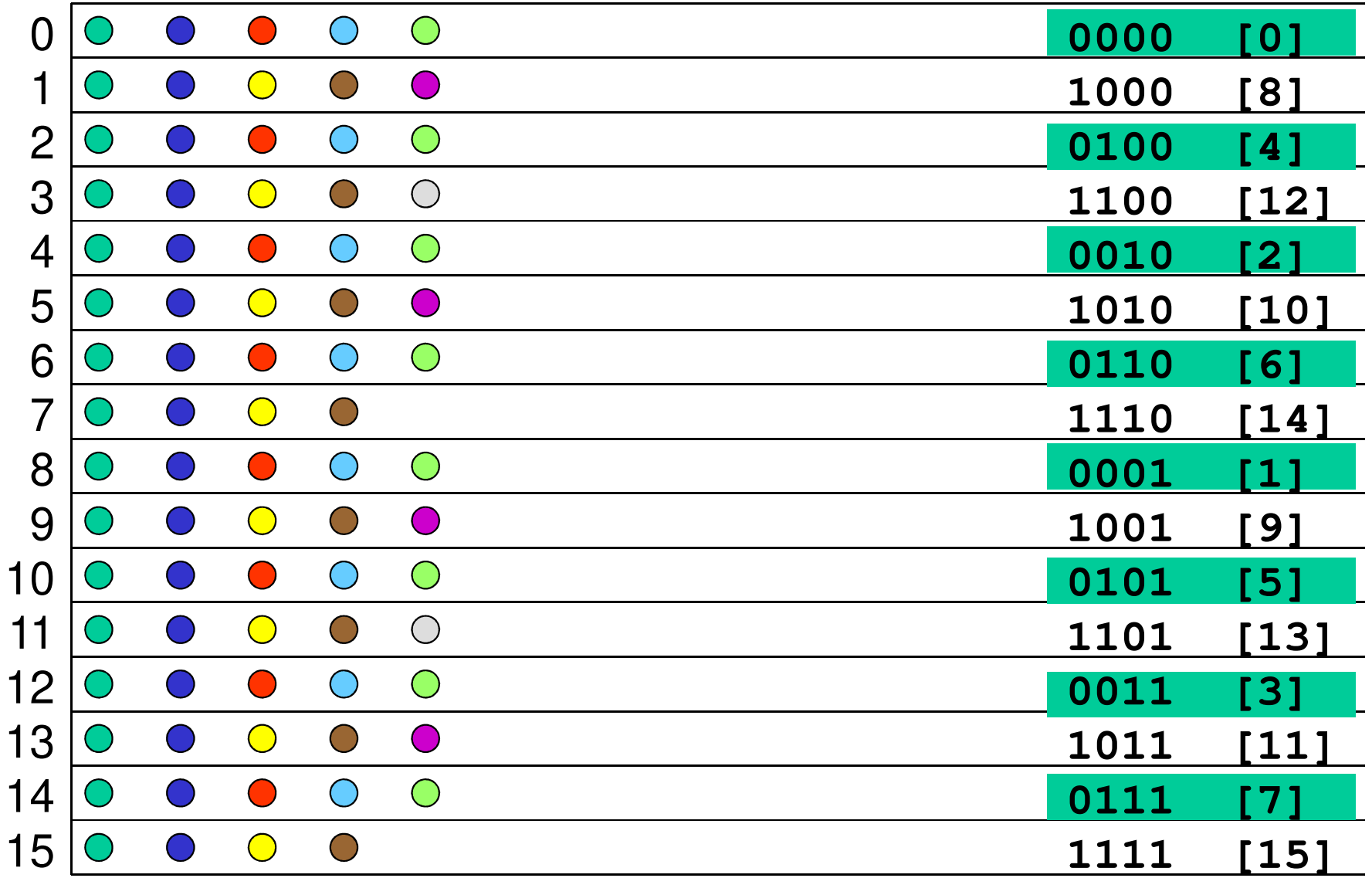
pv = 3:

wait:

...

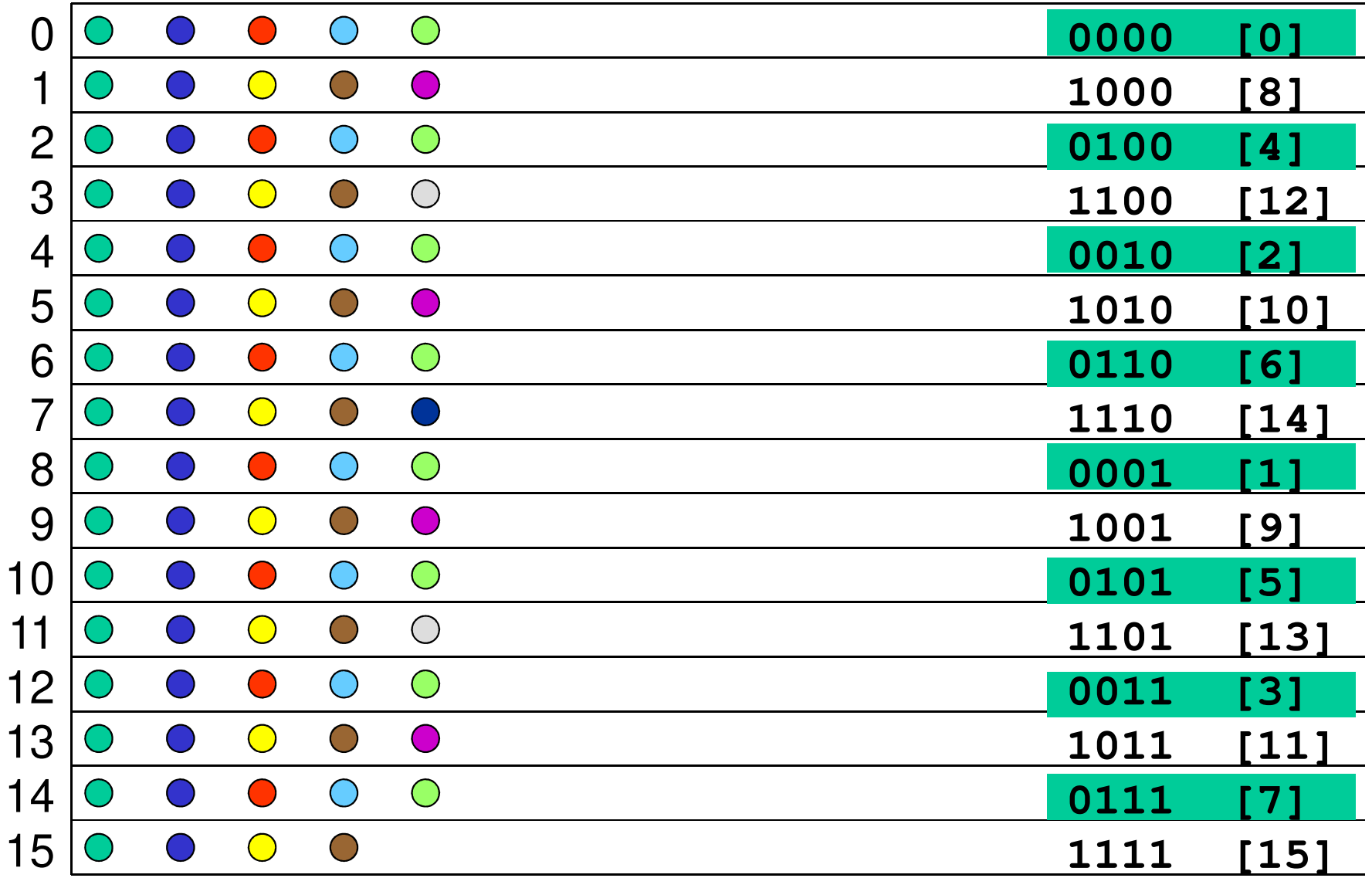
p3.0 = 7

...




wait: ... p4.0 = 7

pv = 4:




wait: ... **p4.0 = 7**

pv = 4:

# B-Scheduling

- perfect execution
  - no idle-time
- optimal frequency
  - provable upper and lower bound for period
- efficiently computable
  - linear in the number of processes
- compact representation
  - two variables per process (wait & priority)

# Basic Analysis

workload WL

- number of all process executions in a schedule

$$WL = \sum_{0 \le i \le \text{maxpv}} \#PC_i \cdot 2^{\text{maxpv}-i}$$

- priority class $PC_i$: all processes with priority i
- maxpv: maximum priority

# Basic Analysis

number of minor cycles $n_{mic}$ :

$$n_{mic} = 2^{\text{maxpv}}$$

average #processes per minor cycle *av*:

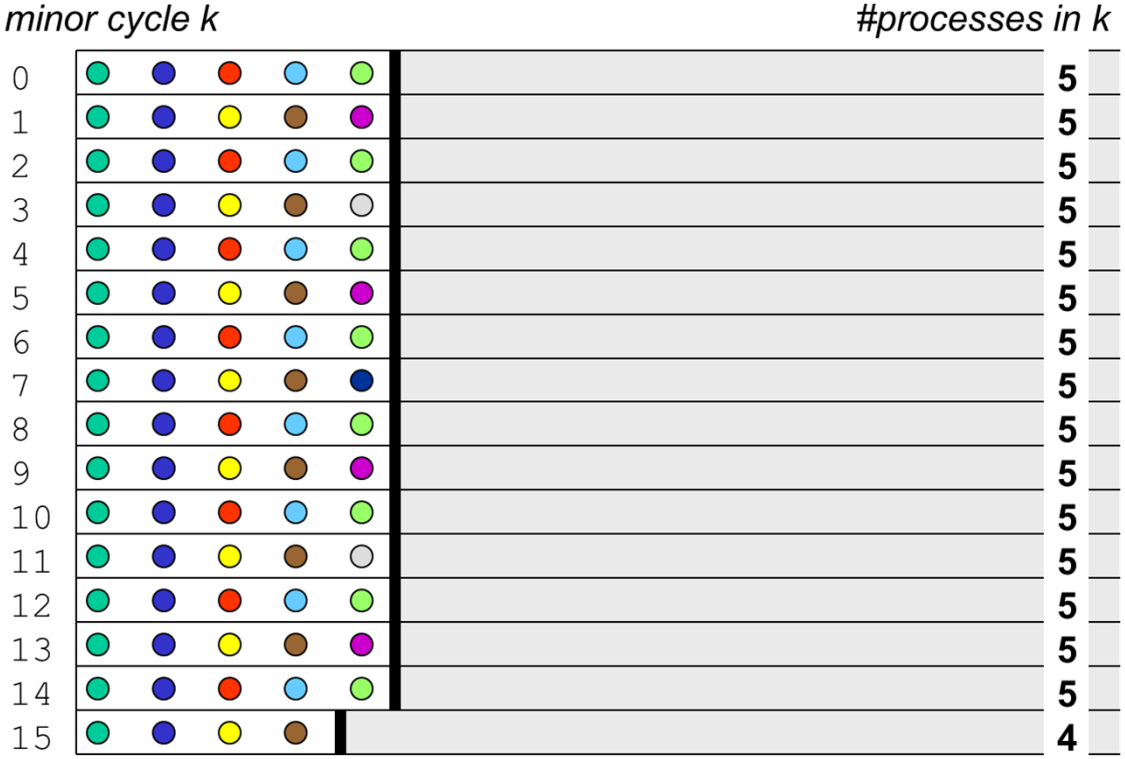
$$av = \frac{WL}{n_{mic}}$$

# Basic Analysis

- *av* is not necessarily an integer
- perfect / dirty minor cycles

$$perfect = \lceil av \rceil$$

$$dirty = \lfloor av \rfloor$$




$$av = \frac{15 \cdot 5 + 1 \cdot 4}{16} = \frac{79}{16} = 4.9375$$

$$perfect = \lceil 4.9375 \rceil = 5$$

$$dirty = \lfloor 4.9375 \rfloor = 4$$

# Basic Analysis

B-schedule S

- S time-optimal (no „waste of time“ through idle processes)
- major cycle consists only of perfect and dirty cycles
- $p_i$ always executed in the same slot in a minor cycle
- $p_i$ exec. in minor cycle $c+2^{pv[pi]}$ *iff* $p_i$ exec. in minor cycle c

therefore:

$$balance(S) = \min_{p_i} \frac{\min_x dist(p_i, x)}{\max_y dist(p_i, y)} = \frac{dirty}{perfect}$$

$$= 1 \text{ for } av \to \infty \text{ , resp. } n \to \infty$$

# Rate Monotic Scheduling

# Rate Monotonic Scheduling

- priorities
  - assigned by the rate monotonic principle
  - i.e., processes with shorter periods get higher priorities
  - note: if two periods are equal, choose an arbitrary one to get a higher priority
- highest priority process *p* that is up is executed
  - i.e., if end of period (= deadline) of *p* is reached
  - currently running process *p'* is interrupted (preemption)
  - and *p* is executed
  - iff *prio(p) > prio(p')*

# Utilization

- processes $p_i$
  - periods $T_i$
  - resource (computation) requirements $C_i$
- utilization
  - per process $U_i$
  - overall U

$$U_i = \frac{C_i}{T_i}$$

$$U = \sum_{i=0}^{n-1} U_i$$

# Rate Monotonic Scheduling Theorem

pre-conditions

- no overhead for context switch
- preemptive
  - highest priority with deadline up
  - gets immediately executed

# Rate Monotonic Scheduling Theorem

example

| | p0 | p1 | p2 |
|---|---|---|---|
| Ti (msec) | 6 | 8 | 12 |
| Ci (msec) | 1 | 2 | 4 |
| priority | 3 | 2 | 1 |
| Ui | 0.17 | 0.25 | 0.33 |

| | | | | | | INT | | | | | | INT | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| executed process | p0 | p1 | p1 | p2 | p2 | p2 | p0 | p2 | p1 | p1 | p2 | p2 | p0 | p2 | p1 | p1 | ... |
| time (msec) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | ... |
| | | | | | | | | | | | | | | | | | |
| p0 | x | 5 | 4 | 3 | 2 | 1 | x | 5 | 4 | 3 | 2 | 1 | x | 5 | 4 | 3 | ... |
| p1 | 7 | x | x | 7 | 6 | 5 | 4 | 3 | x | x | 7 | 6 | 5 | 4 | x | x | ... |
| p2 | 11 | 10 | 9 | x | x | x | - | x | 11 | 10 | x | x | - | x | 11 | 11 | ... |

# Rate Monotonic Scheduling Theorem

theorem: processes can be scheduled if

$$U \leq U_{RMS} = n(2^{1/n} - 1)$$

# Conrete Values for $U_{RMS}$

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $U_{rms}$ (in %) | 100 | 82.84 | 77.98 | 75.68 | 74.35 | 73.48 | 72.86 | 72.41 | 72.05 | 71.77 |
| $n$ | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| $U_{rms}$ | 71.77 | 70.53 | 70.12 | 69.91 | 69.78 | 69.72 | 69.66 | 69.62 | 69.58 | 69.56 |

$$\lim_{n\to\infty} U_{RMS} = \lim_{n\to\infty} n(2^{1/n} - 1) = \ln 2 \approx 0.693$$

# $U_{RMS}$ is an upper bound

example

- $U_{RMS}$ = 75.68 (for n=4)
- U = 90.83% > $U_{RMS}$
- but scheduling works

| | p0 | p1 | p2 | p3 |
|---|---|---|---|---|
| $T_i$ (msec) | 3 | 8 | 8 | 10 |
| $C_i$ (msec) | 1 | 1 | 2 | 2 |
| priority | 4 | 3 | 2 | 1 |
| $U_i$ | 0.33 | 0.13 | 0.25 | 0.2 |

| | | | INT | | | INT | | | | | | | | | | | | INT | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| executed process | p0 | p1 | p2 | p0 | p2 | p3 | p0 | p3 | p1 | p0 | p2 | p2 | p1 | p1 | p3 | p0 | p1 | p3 | p0 | p3 | ... |
| time (msec) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | ... |
| p0 | x | 2 | 1 | x | 2 | 1 | x | 2 | 1 | x | 2 | 1 | x | 2 | 1 | x | 2 | 1 | x | 2 | ... |
| p1 | 7 | x | 7 | 6 | 5 | 4 | 3 | 2 | x | 7 | 6 | 5 | 4 | 3 | 2 | 1 | x | 7 | 6 | 5 | ... |
| p2 | 7 | 6 | x | - | x | 7 | 6 | 5 | 4 | 3 | x | x | 7 | 6 | 5 | 4 | 3 | x | - | x | ... |
| p3 | 9 | 8 | 7 | 6 | 5 | x | - | x | 9 | 8 | 7 | 6 | 5 | x | x | 9 | 8 | 7 | 6 | 5 | ... |