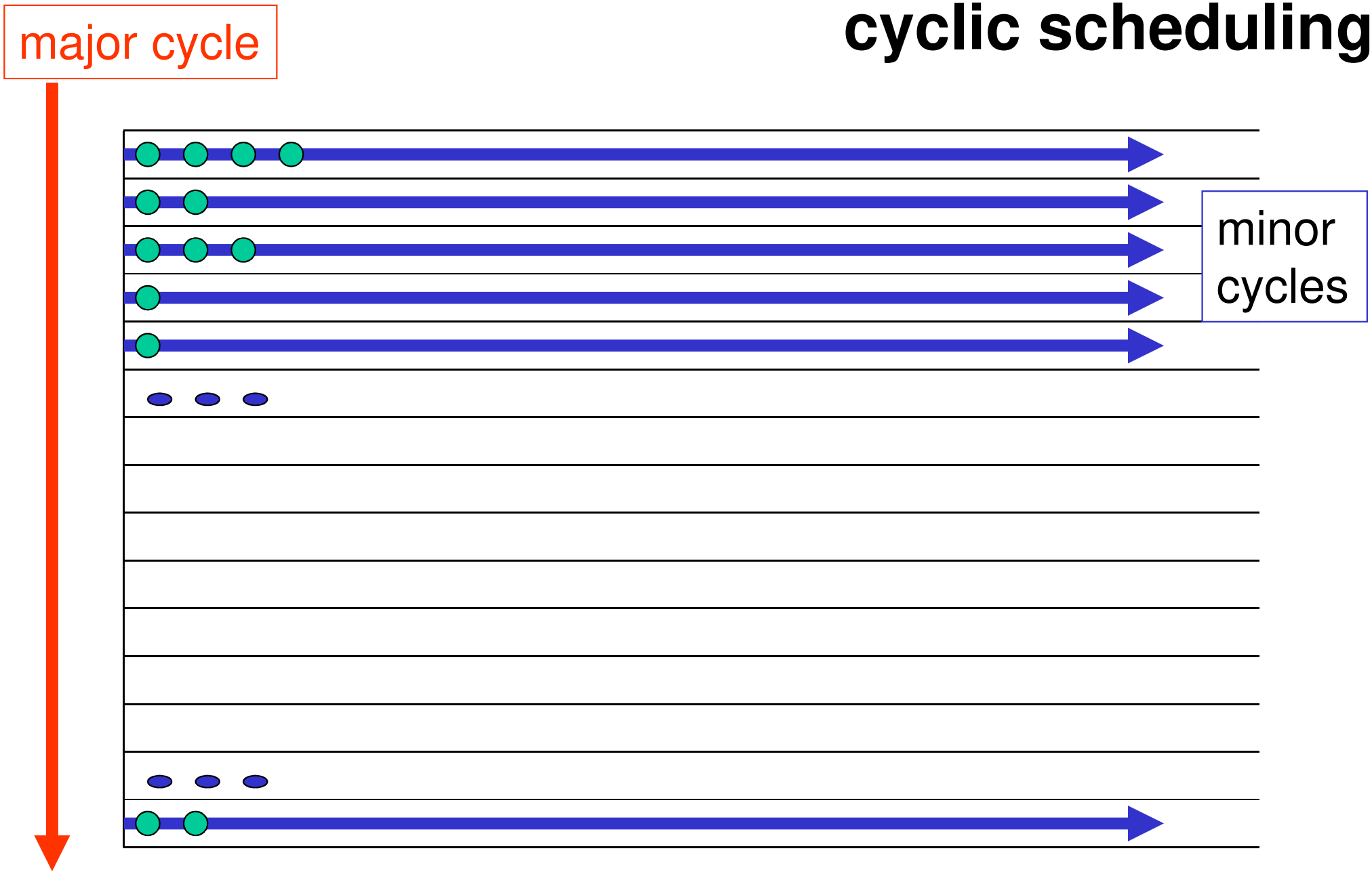
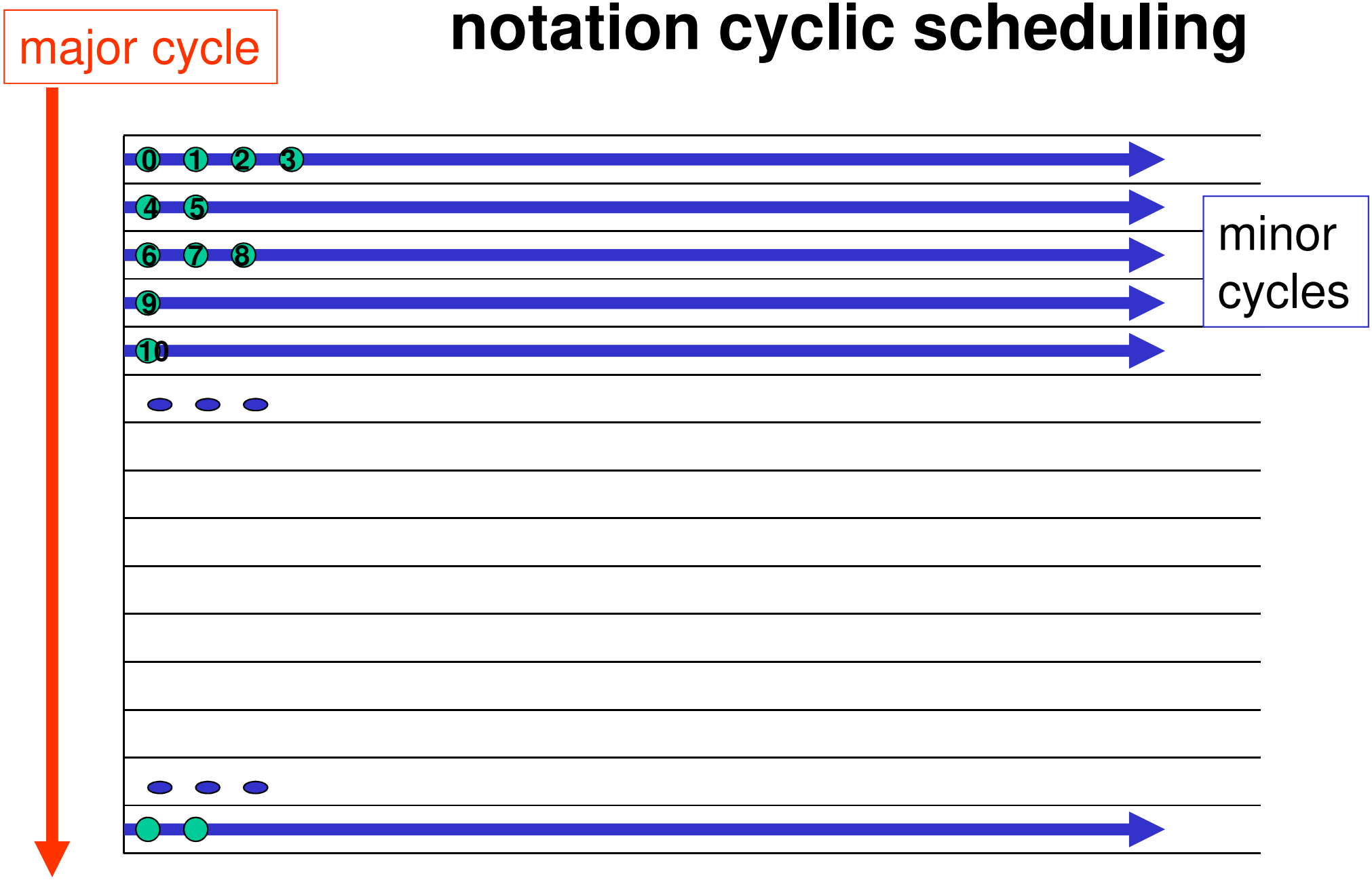
cyclic scheduling



notation cyclic scheduling



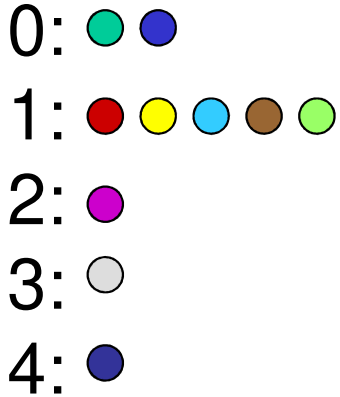
● = execution of a process

(numbers indicate the order of the executions)

Example

- 10 processes
- 5 priority classes

prio[i]

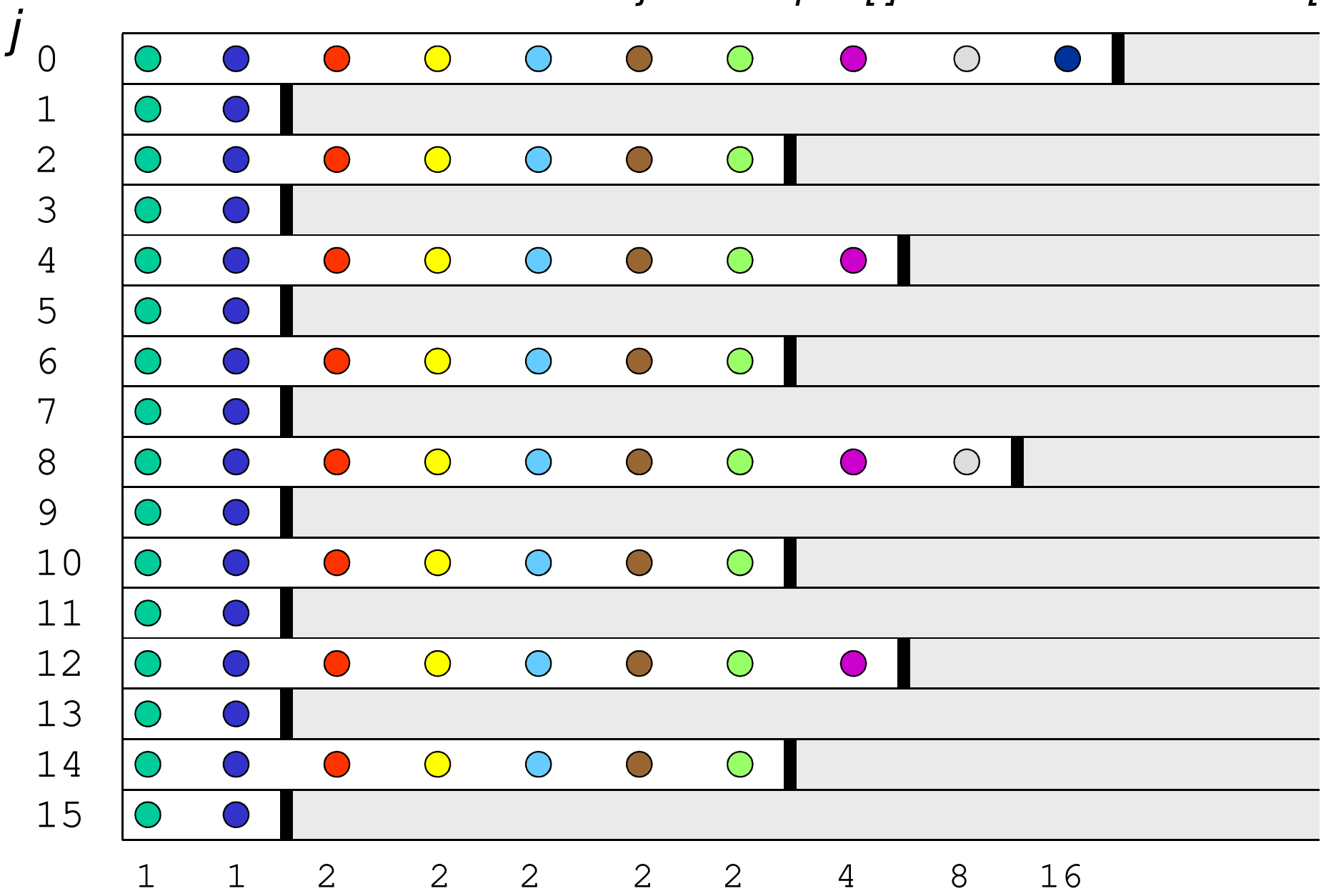


direct scheduling

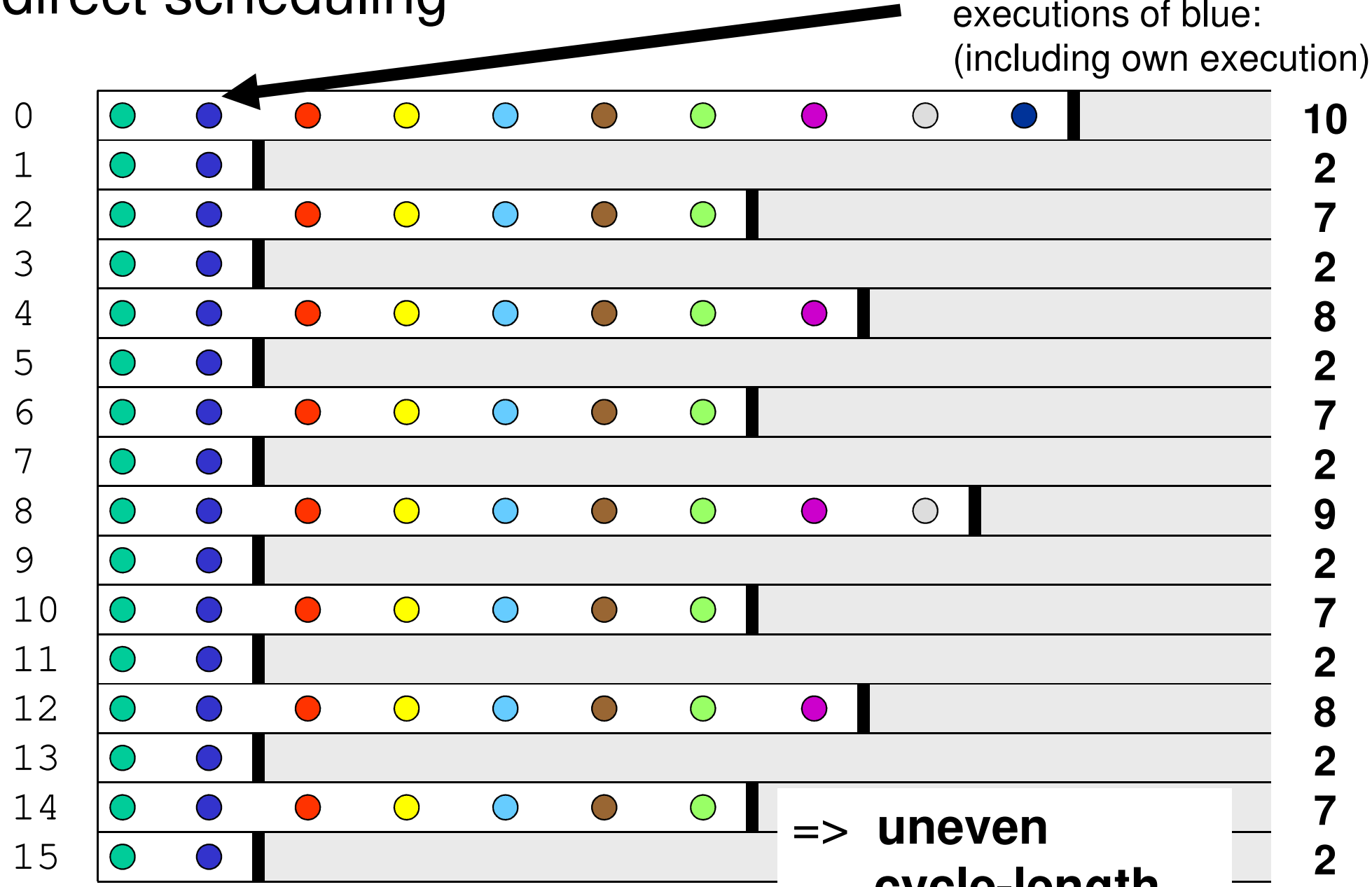
```
1  /* Execute the Major Cycle */
2  for(round = 0; round < n_mic; round = round + 1) {
3          /* Execute the Minor Cycle */
4          ∀p_id ∈ P: {
5                  if(round modulo 2^pv[p_i] == 0) {
6                          execute p_id
7                  }
8          }
9  }
```

direct scheduling

minor cycle *j* :
if *j mod 2^prio[i] == 0* then *execute P[i]*



direct scheduling



Formal Approach

balance

- schedule S
- *distance: dist(p_i,z)* #processes between execution p_i in cycle z and its next execution

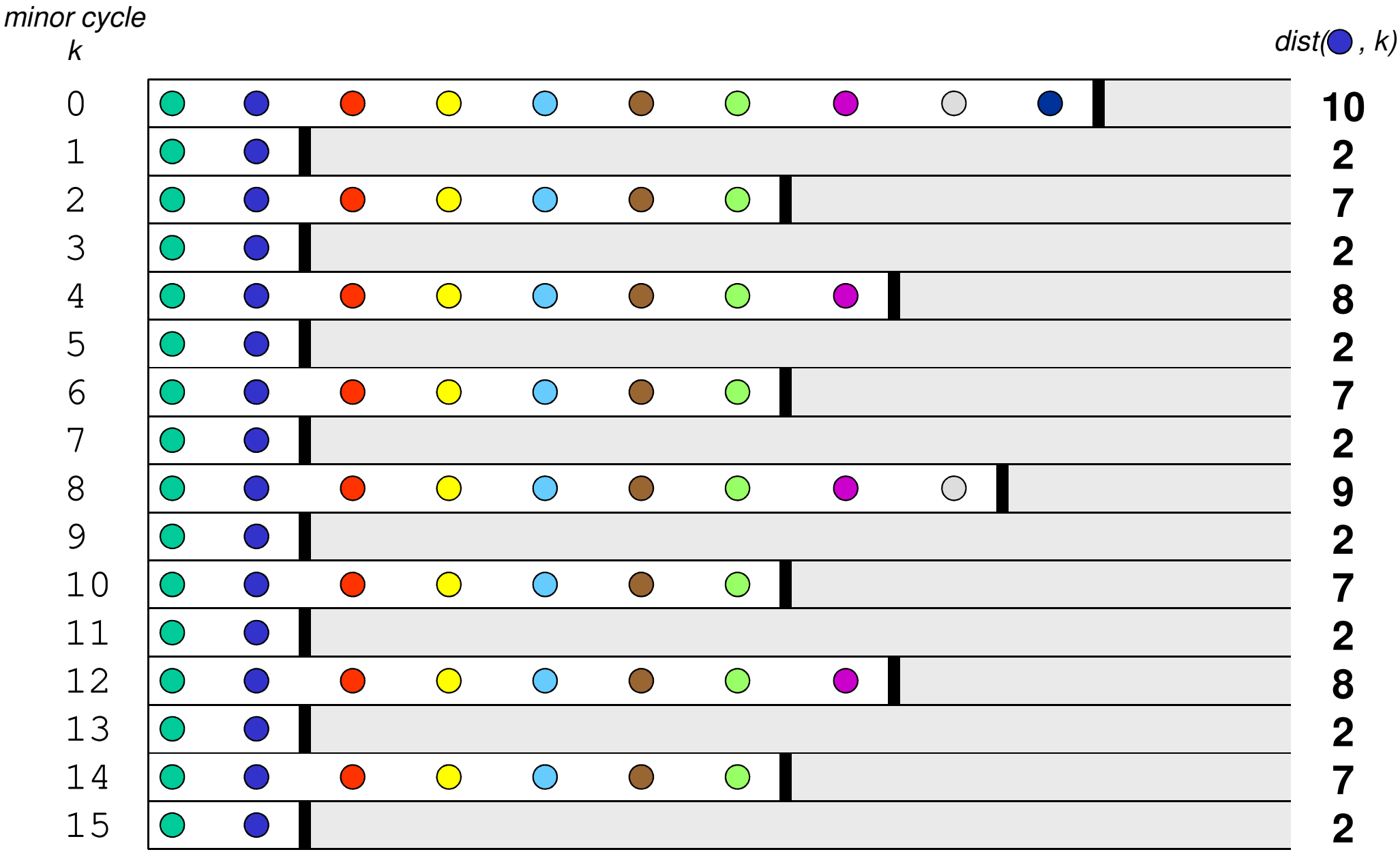
$$balance(S) = \min_{p_i} \frac{\min_x dist(p_i, x)}{\max_y dist(p_i, y)}$$

direct scheduling bad

- example S_1
  - 1 process priority 0
  - n-1 processes priority 1

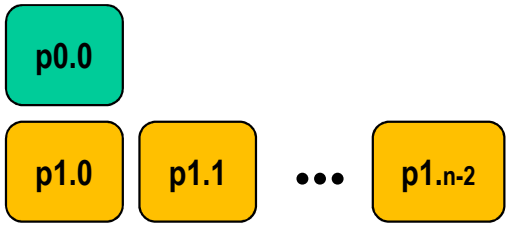
$$balance(S_1) = \frac{1}{n} = 0 \text{ for } n \to \infty$$

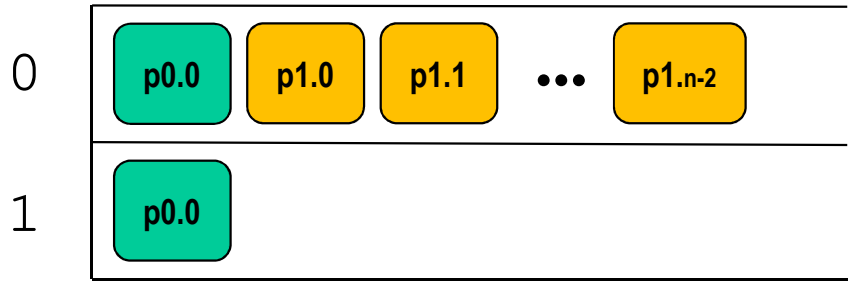
Note: dist() includes counting the own execution of the process



Problem: Unbalanced Execution (Jitter) of S_1

- 1 process priority 0
- n-1 processes priority 1





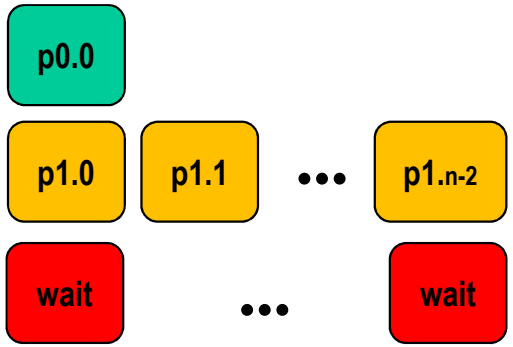
$$dist(p0.0,0) = n \qquad dist(p1.i,0) = n$$

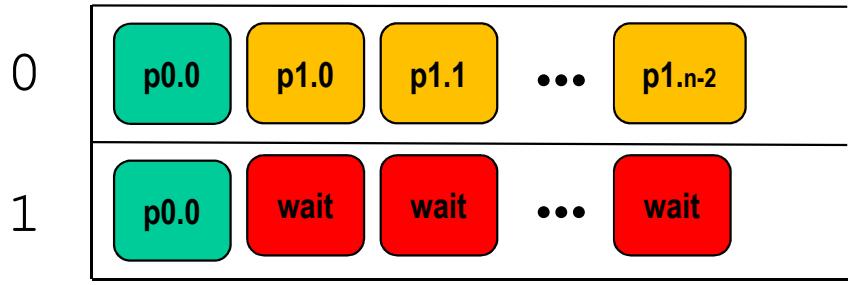
$$dist(p0.0,1) = 1$$

$$balance(S_1) = \min_{p_i} \frac{\min_x dist(p_i, x)}{\max_y dist(p_i, y)} = \frac{1}{n}$$

Pseudo Solution: Idle Time (Do Nothing Processes)

- 1 process priority 0
- n-1 processes priority 1
- n-1 wait processes





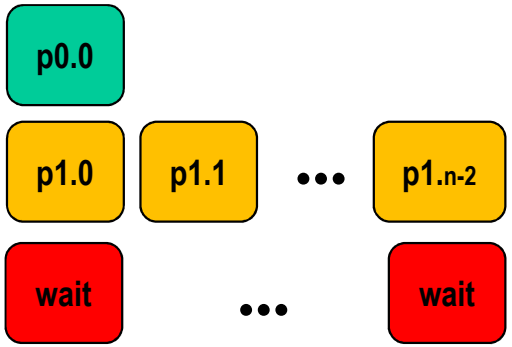
$$dist(p0.0,0) = n \quad dist(p1.i,0) = 2n$$

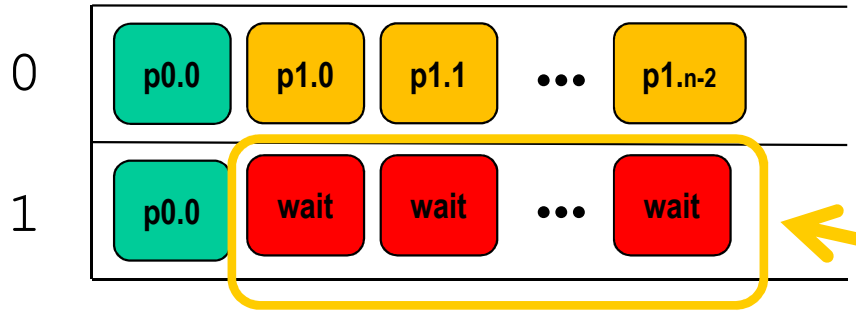
$$dist(p0.0,1) = n$$

$$balance(S_{wait}) = \min_{p_i} \frac{\min_x dist(p_i, x)}{\max_y dist(p_i, y)} = \frac{n}{n} = 1$$

Pseudo Solution: Idle Time (Do Nothing Processes)

- 1 process priority 0
- n-1 processes priority 1
- n-1 wait processes





$$dist(p0.0,0) = n \quad dist(p1.i,0) = 2n$$

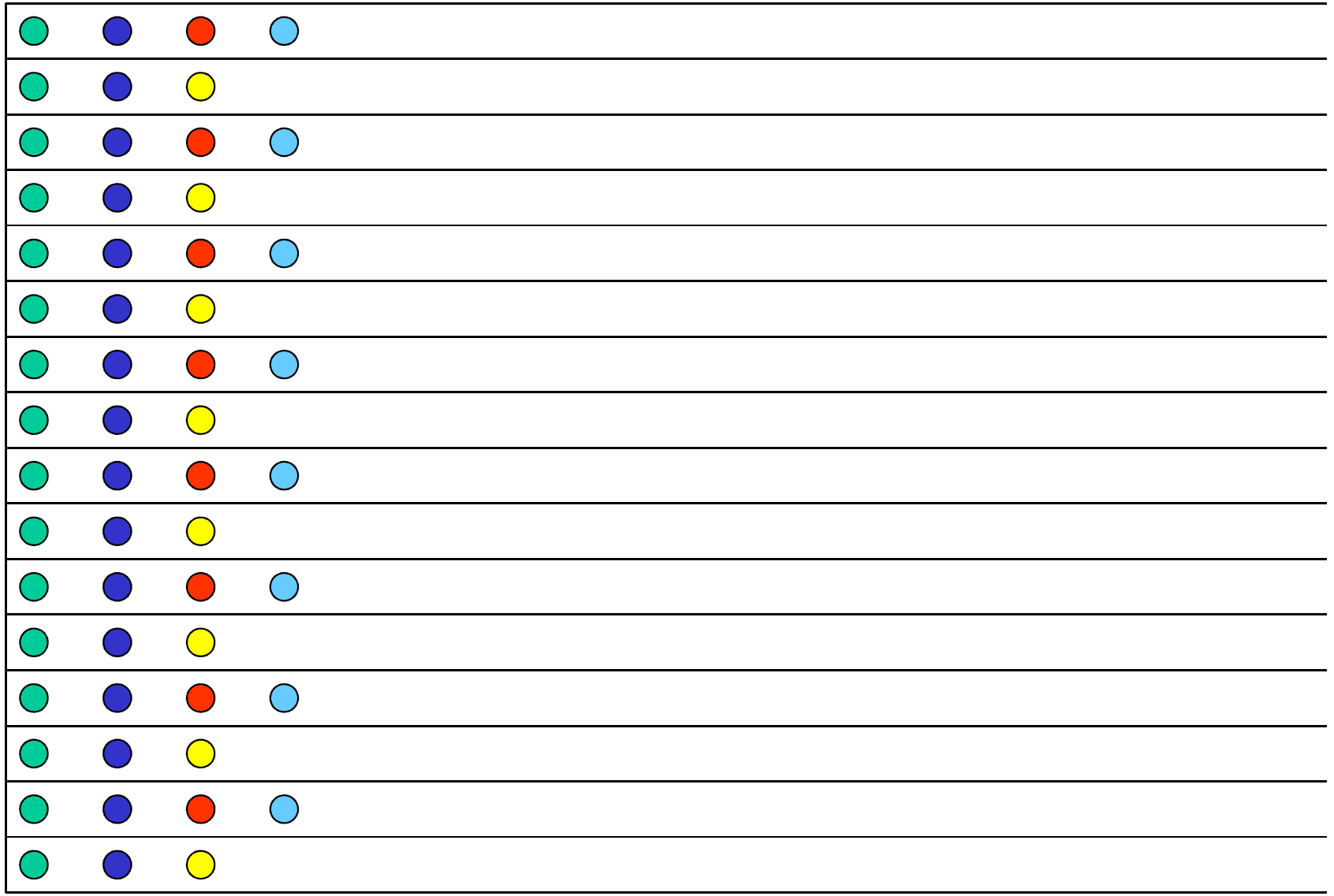
$$dist(p0.0,1) = n$$

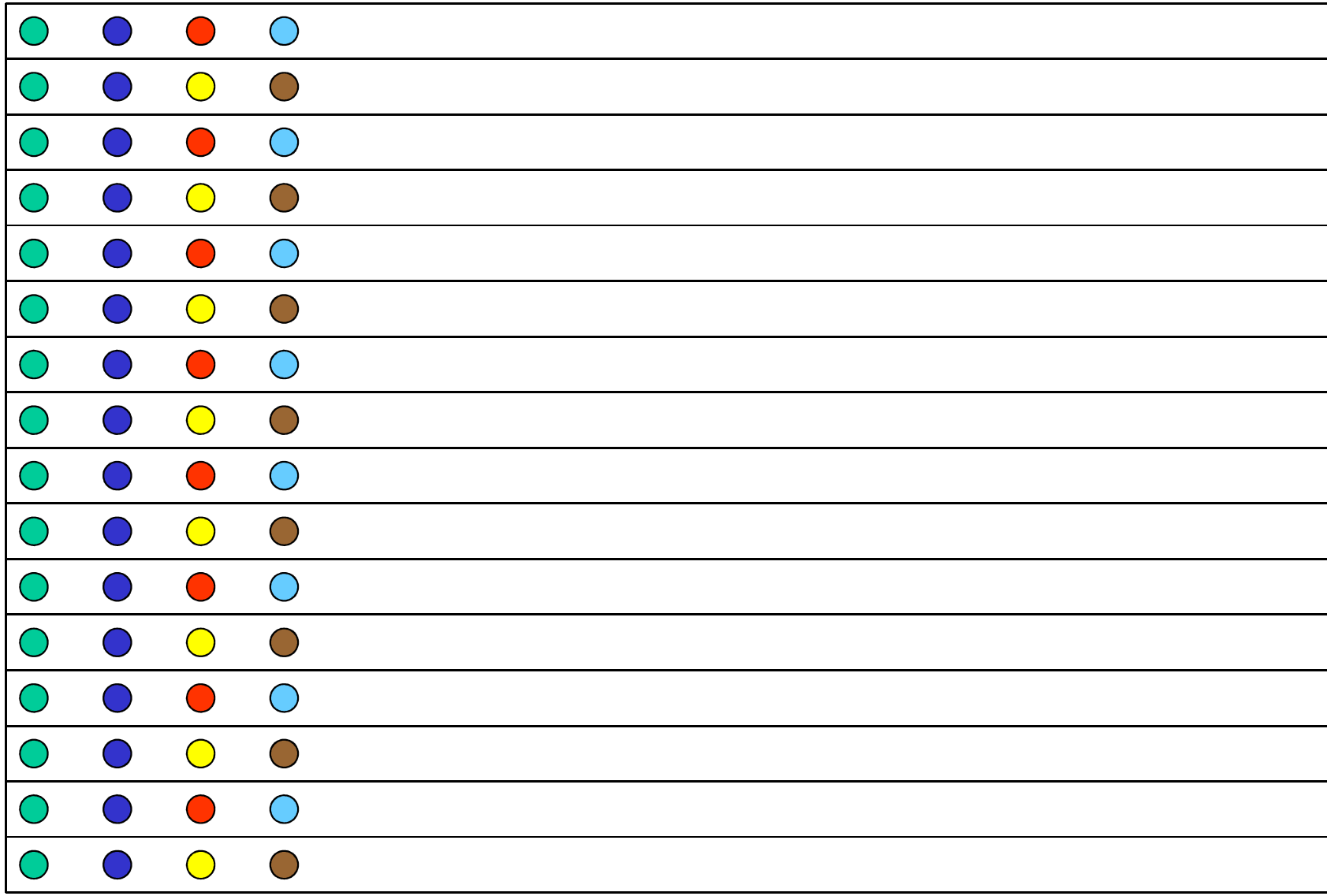
optimal balance
but unlimited waste
of processing time!!!

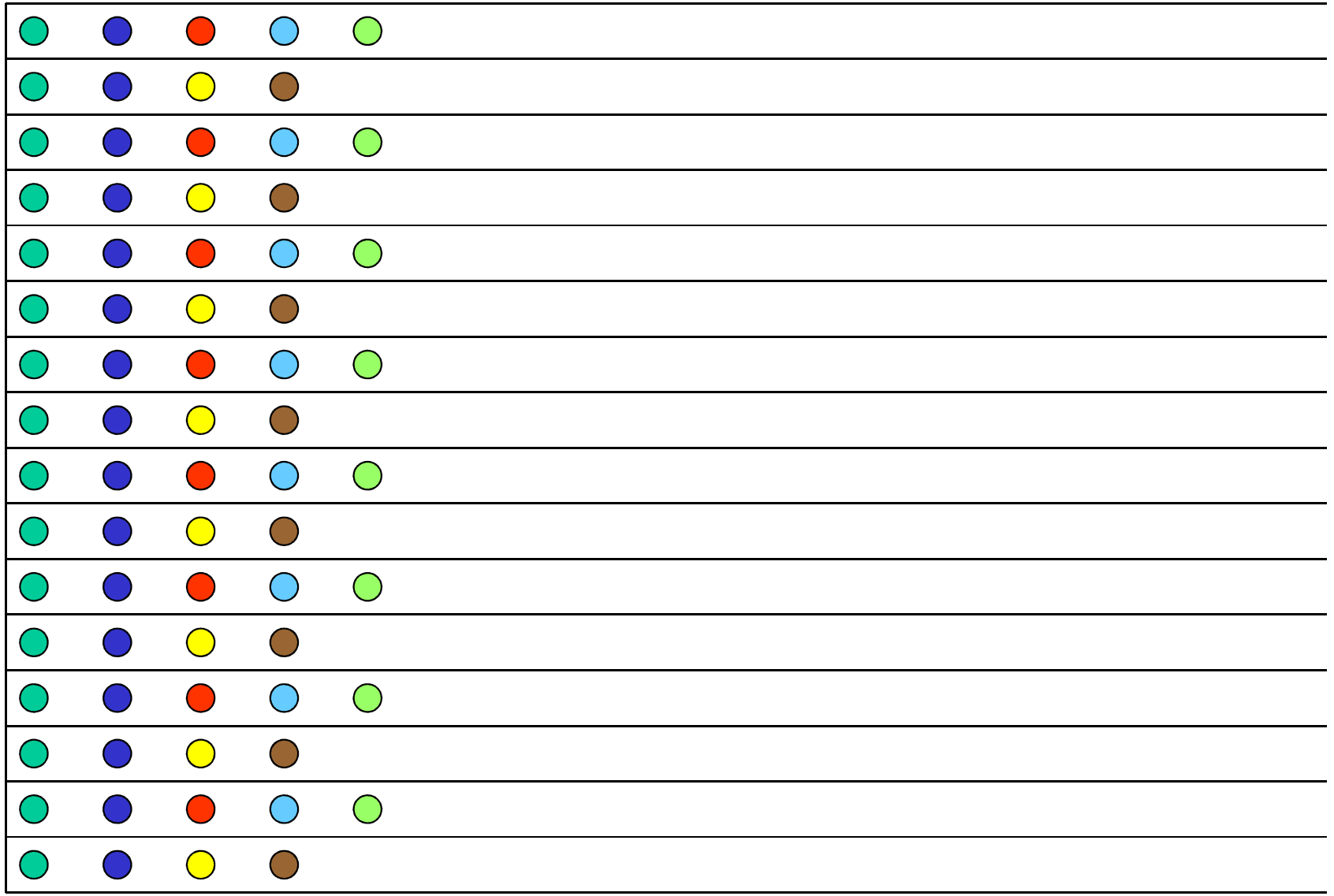
$$balance(S_{wait}) = \min_{p_i} \frac{\min_x dist(p_i,x)}{\max_y dist(p_i,y)} = \frac{n}{n} = 1$$

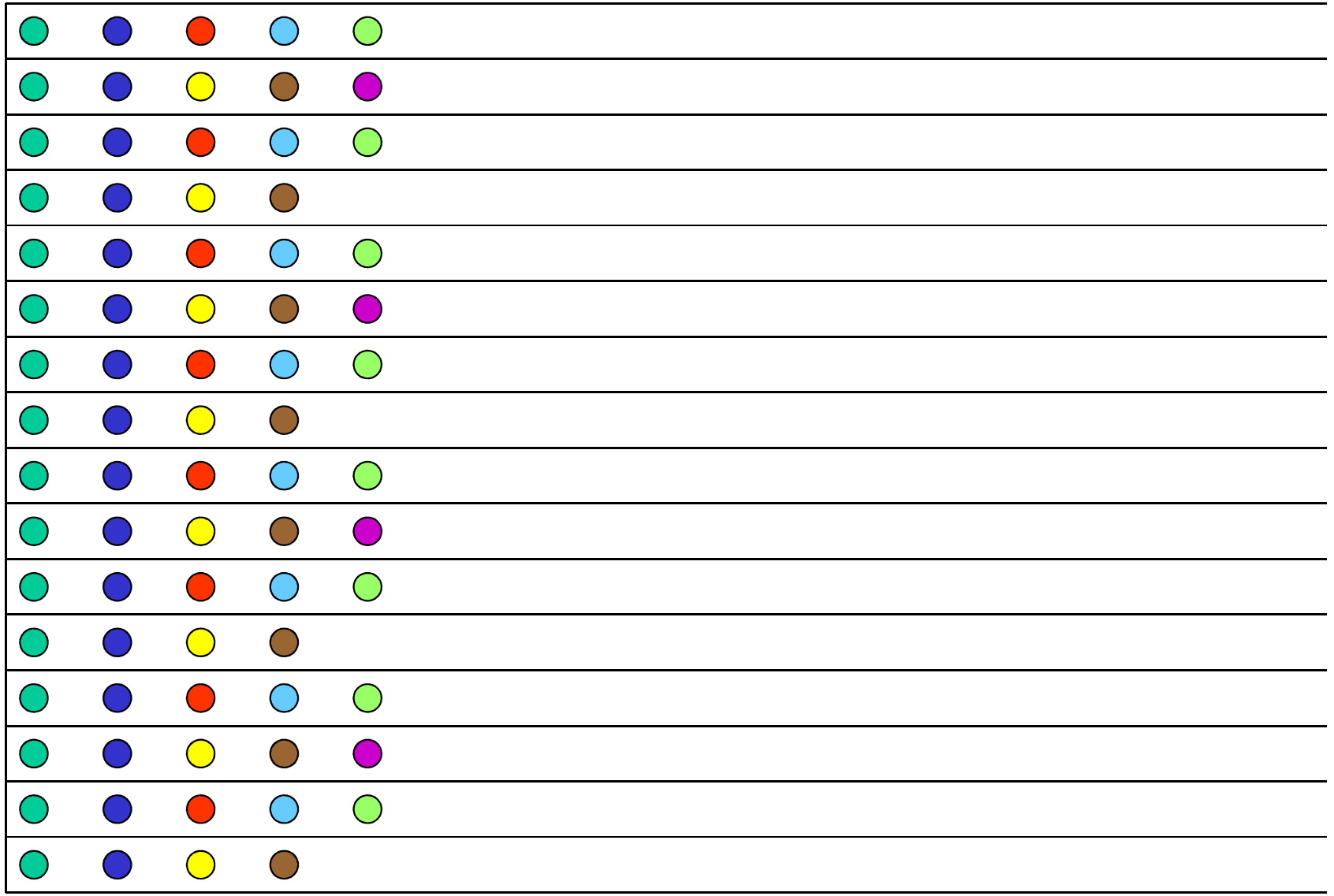
Real Solution: B-Scheduling

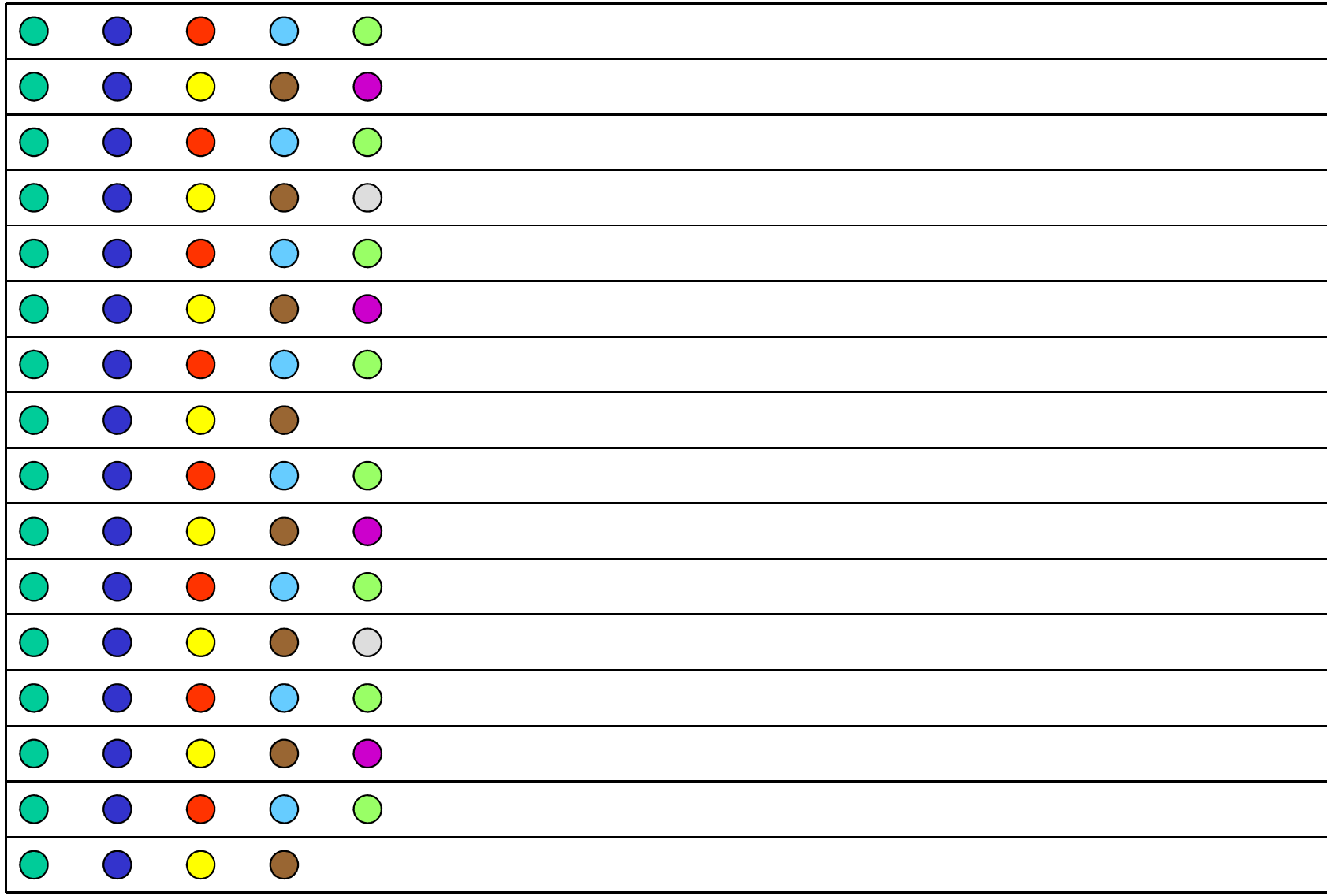
- designed for behaviors
  - assumes same ressource needs for each process
- best effort scheduler
  - no fixed time constraints
  - but tries to maximizes the usage of the processor
- based on exponential effect priorities
- with provable optimality properties

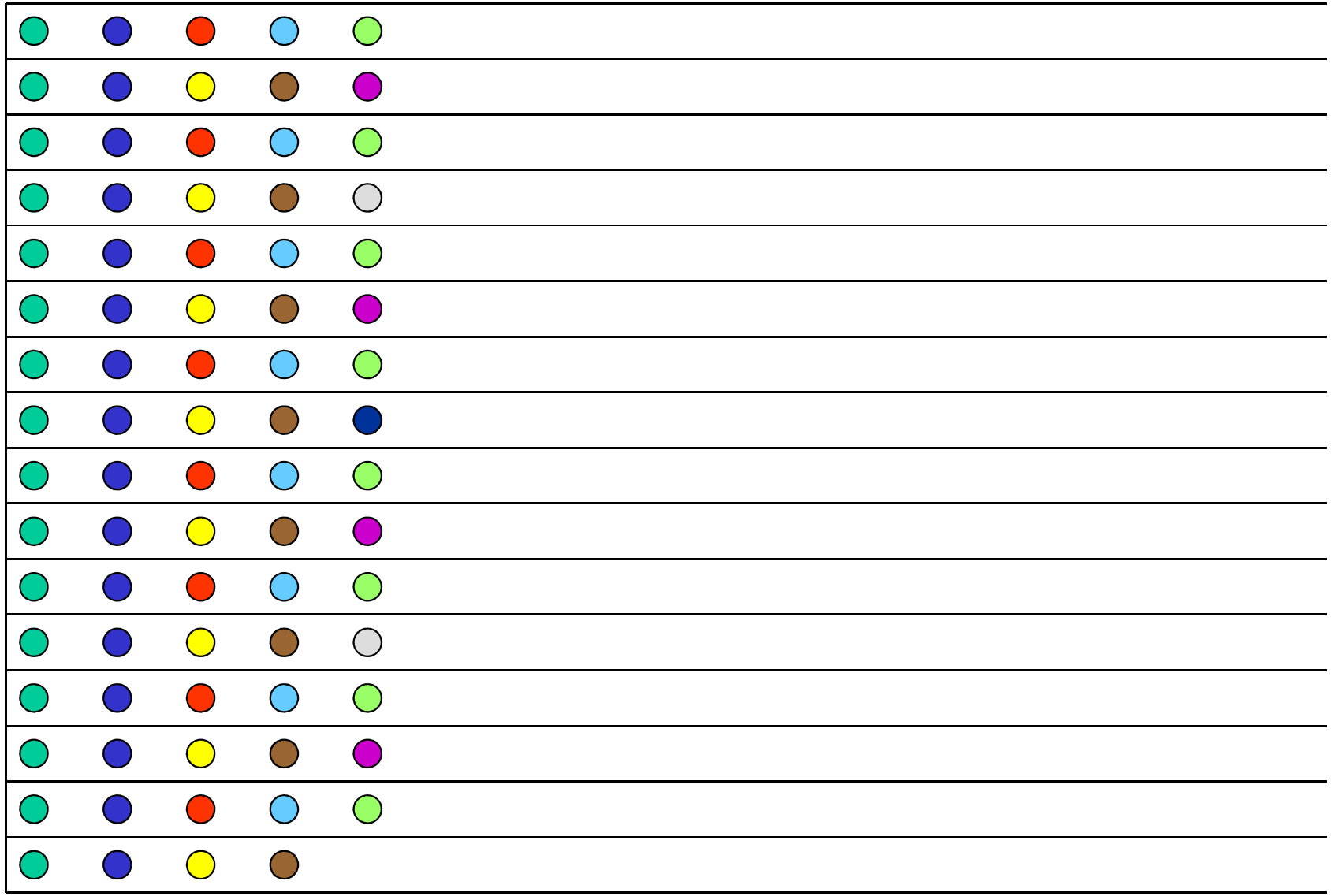


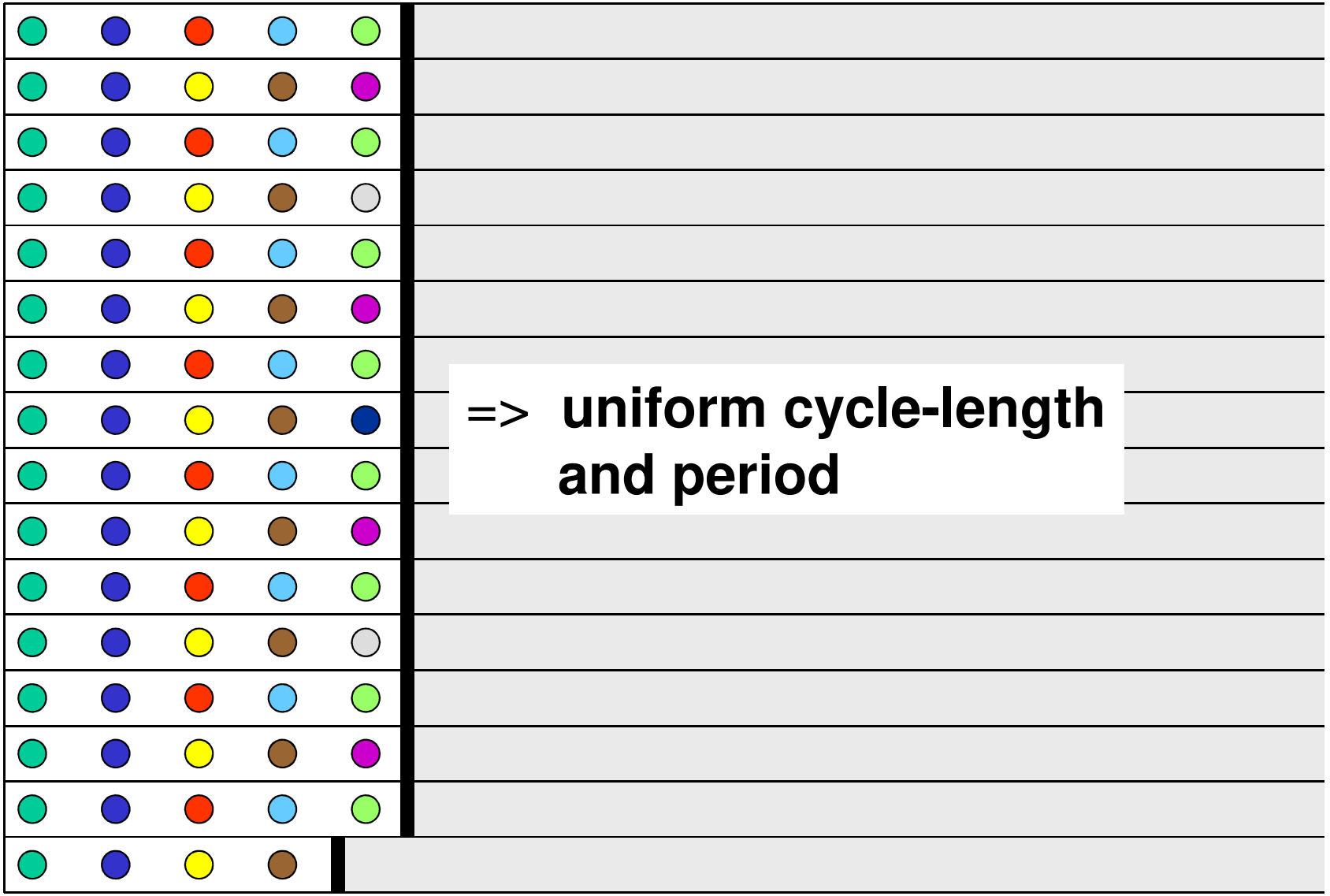


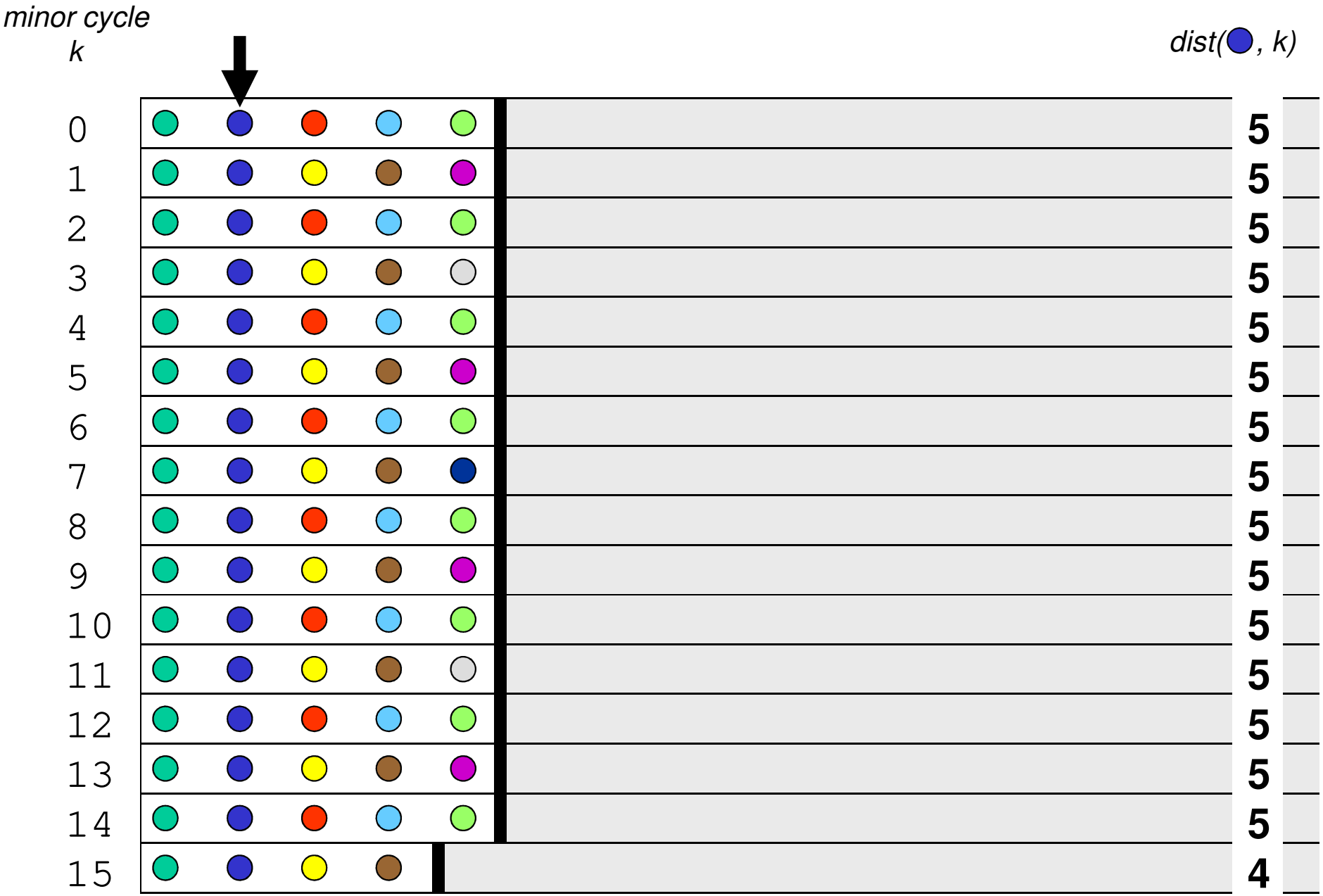









=> uniform cycle-length
and period


minor cycle
k
dist(, k)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
5
5
5
5
5
5
5
5
5
5
5
5
5
5
5
4

Reverse Encoded Binary

two bit-strings with n digits:

X = x(n-1), ... ,x(0)

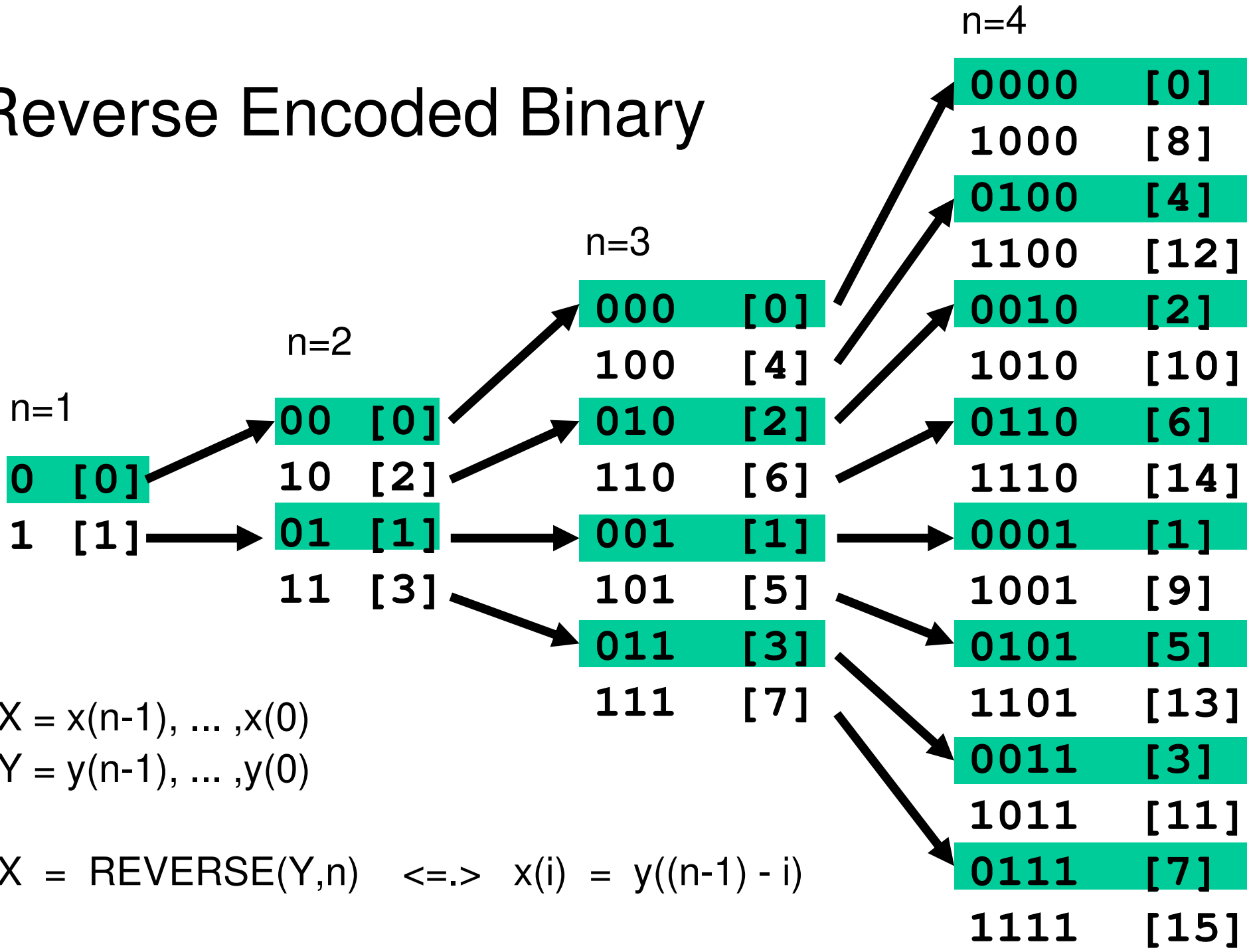
Y = y(n-1), ... ,y(0)

X = REVERSE(Y,n) <=.> x(i) = y((n-1) - i)

i.e.,

- read/interpret bit-string X backwards as binary number
- Most Significant Bit (MSB) changes most frequently when enumerating reverse encoded binary numbers (instead of LSB)

Reverse Encoded Binary



X = x(n-1), ... ,x(0)
Y = y(n-1), ... ,y(0)

X = REVERSE(Y,n) <=.> x(i) = y((n-1) - i)

Computation of Reverse Binary

```
// computes Reverse Binary
// of number k with d digits
//
int ReverseBinary(int k, int d){
        int tmp = 0;
        int i;

        if(d<1) return(0);
        for(i=0;i<d;i++){
                tmp = tmp<<1;
                tmp += k & 1;
                k = k>>1;
        }
        return(tmp);
}
```

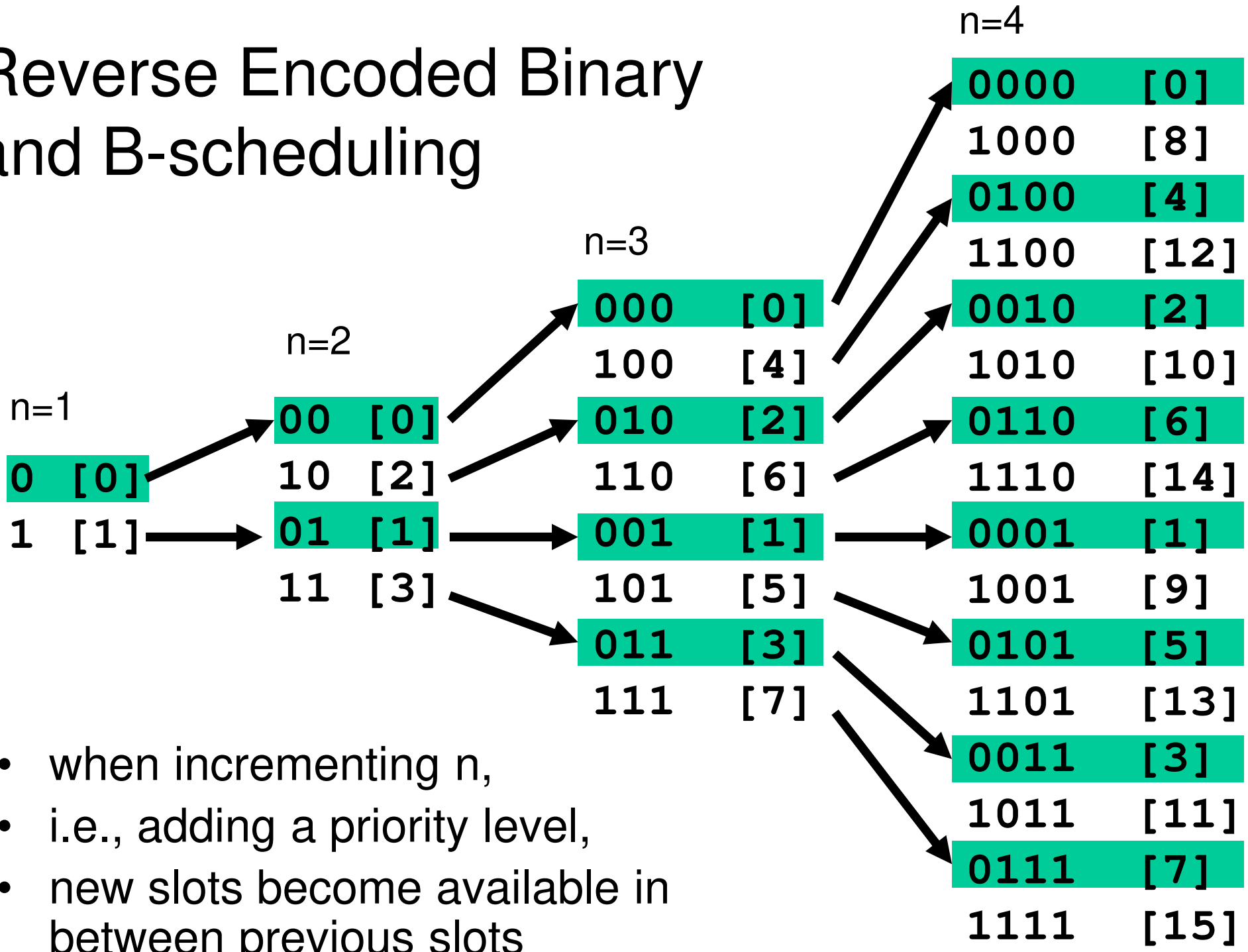
Rev.Bin.(0,0) = 0

Rev.Bin.(0,1) = 0
Rev.Bin.(1,1) = 1

Rev.Bin.(0,2) = 0
Rev.Bin.(1,2) = 2
Rev.Bin.(2,2) = 1
Rev.Bin.(3,2) = 3

Rev.Bin.(0,3) = 0
Rev.Bin.(1,3) = 4
Rev.Bin.(2,3) = 2
Rev.Bin.(3,3) = 6
Rev.Bin.(4,3) = 1
Rev.Bin.(5,3) = 5
Rev.Bin.(6,3) = 3
Rev.Bin.(7,3) = 7

Reverse Encoded Binary and B-scheduling



- when incrementing n,
- i.e., adding a priority level,
- new slots become available in between previous slots

The B-scheduling algorithm (roughly sketched)

- start:
  - $\forall$ *P[i] :* compute *wait[i]* $\in$ *{ 0,..., $2^{prio[i]}$ }*
  - use reverse-code on indices
- execution:
  - if *wait[i]* == 0
  - execute *P[i]*
  - *wait[i]* += $2^{prio[i]}$
  - *wait[i]--*

B-scheduling algorithm

implementation in C

- processes p[i][j]
  - i = priority class of process p
  - j = index of p within the priority class

- wait[i][j]

B-scheduling – compute wait

```
void ComputeWait() {
        int start = 0;
        int slots = 1;

        int i,j;

        for(i=0; i<=MAXPV; i++) {
                for(j=0; j<ProcPerPC[i]; j++){
                        wait[i][j] = ReverseBinary((start+j)%slots, i);
                }
                start = (start + ProcPerPC[i]) % slots;
                start = 2 * start;
                slots = 2 * slots;
        }
}
```

B-scheduling – execution

```
void ExecuteBSchedule() {
        int nmic, int round;
        int i,j;

        // compute the number of minor cycles
        nmic = (1<<MAXPV);
        // execute major cycle
        for(round=0; round < nmic; round++) {
                // execute minor cycle
                for(i=0; i<=MAXPV; i++){
                        for(j=0; j<ProcPerPC[i]; j++){
                                if(wait[i][j]==0) {
                                        // „execute“ p[i][j]
                                        printf ("p%d.%d ", i, j);
                                        wait[i][j] = 1<<i;
                                }
                                wait[i][j]--;
                        }
                }
                printf ("\n");
        }
}
```