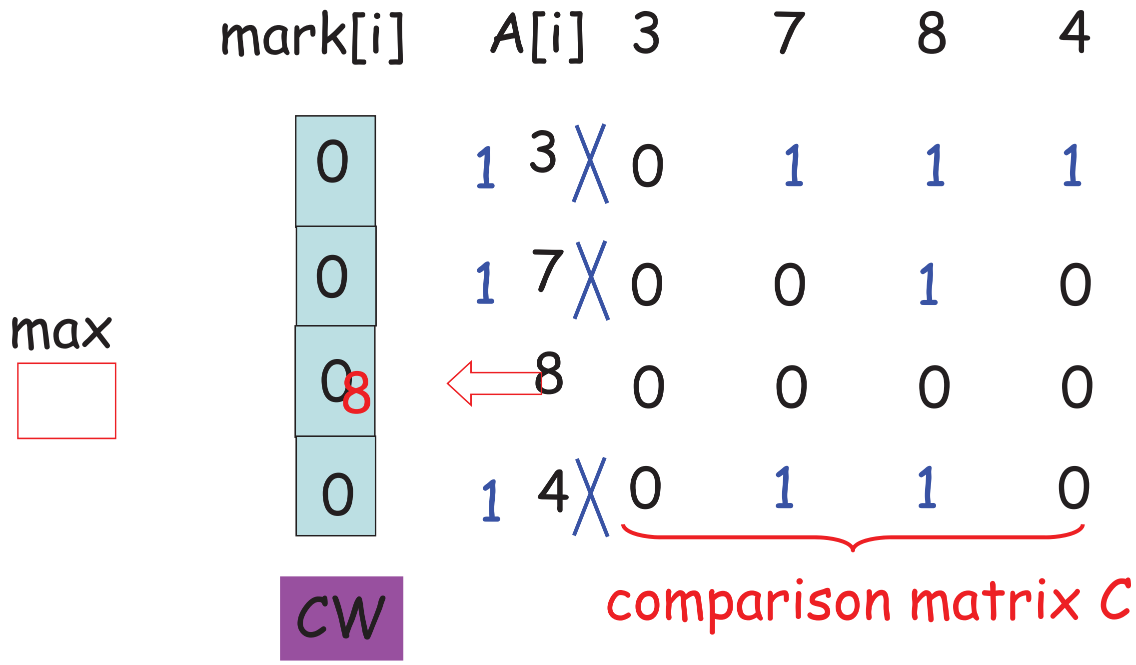



## Finding maximum on the CRCW PRAM



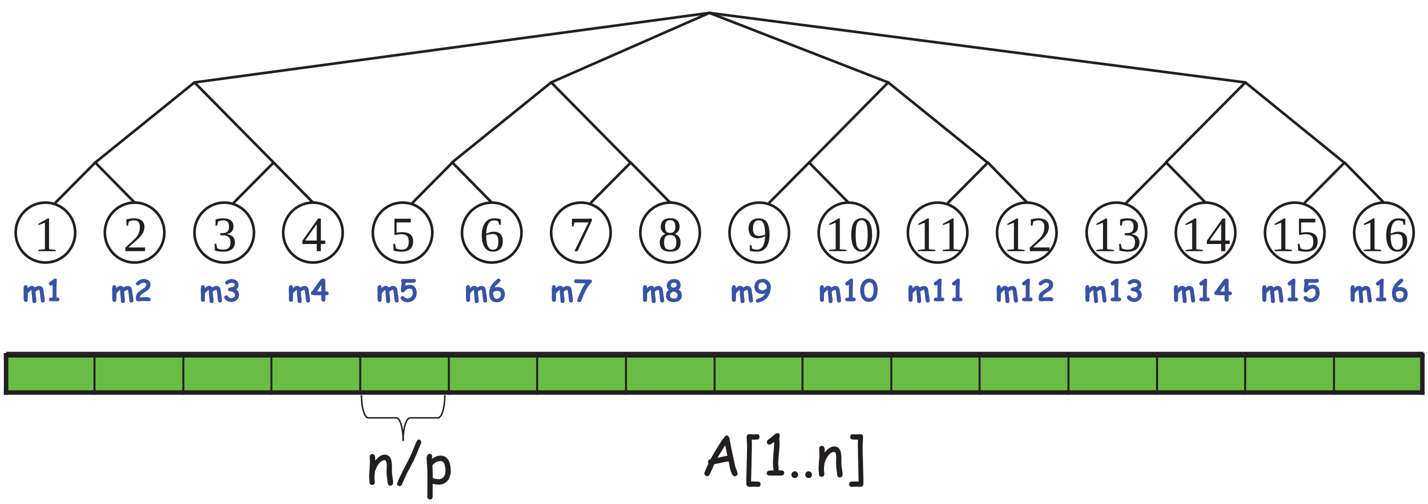
Algo 1: 1 PE, O(n) time, O(n) cost (slow but cheap)

Algo 2: n PE, O(lglg n) time, O(n lglg n) cost (fast but expensive)

Cascading: using p PEs (Example: p = 16)

Stage 1: Algo 1 - O(n/p) time

Stage 2: Algo 2 - O(lglg p) time



Finding maximum
- Algo 1: 1 PE, O(n) time (slow but cheap)
- Algo 2: n PE, O(lglg n) time (fast but expensive)

Cascading: using p PEs

Stage 1: Algo 1 - O(n/p) time

Stage 2: Algo 2 - O(lglg p) time

Time: O(n/p + lglg p)

Cost: Time $\times$ p = O(n + plglg p)

What is the best p?

1. Use smallest p to achieve the fastest O(lglg n) time
   $\Rightarrow$ reduce the cost of Algo 2
2. Use largest p to achieve the optimal O(n) cost
   $\Rightarrow$ speedup the time of Algo 1



**Exercise**: Finding maximum on the EREW PRAM

- Algo 1: 1 PE, O(n) time, O(n) cost (slow but cheap)
- Algo 2: n PE, O(lg n) time, O(n lg n) cost (fast but expensive)

Cascading: using p PEs

Stage 1: Algo 1 - O(?)time

Stage 2: Algo 2 - O(?) time

Time: O(?)

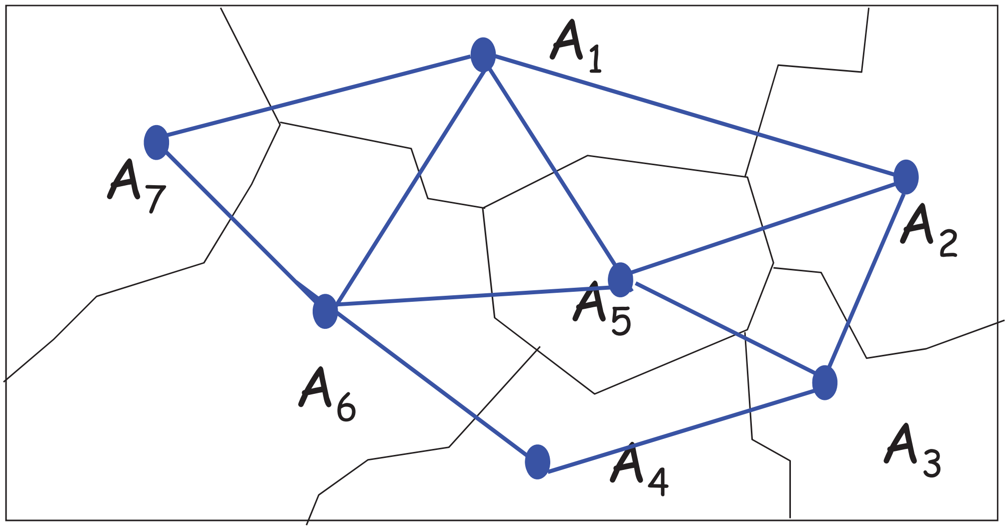
Cost: O(?)

What is the best p?

1. achieve the fastest O(lg n) time: p = ?
2. achieve the optimal O(n) cost: p= ?

# The k-coloring problem

A map of 7 areas ➡ A graph of 7 nodes

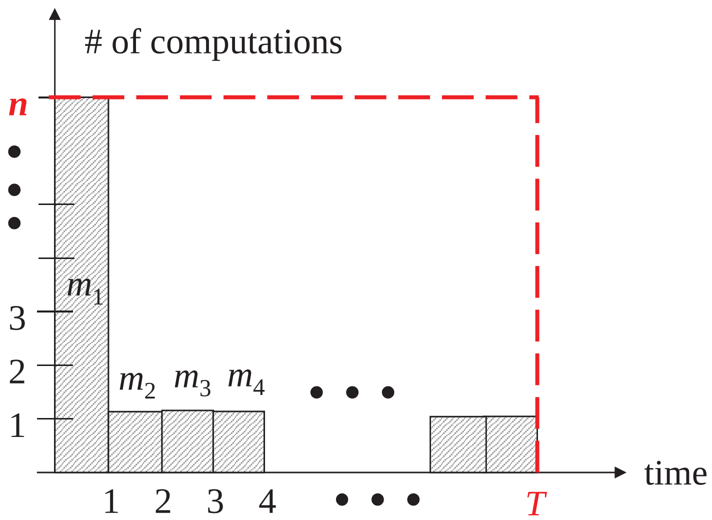



color set = {0, 1, 2, ..., k – 1}

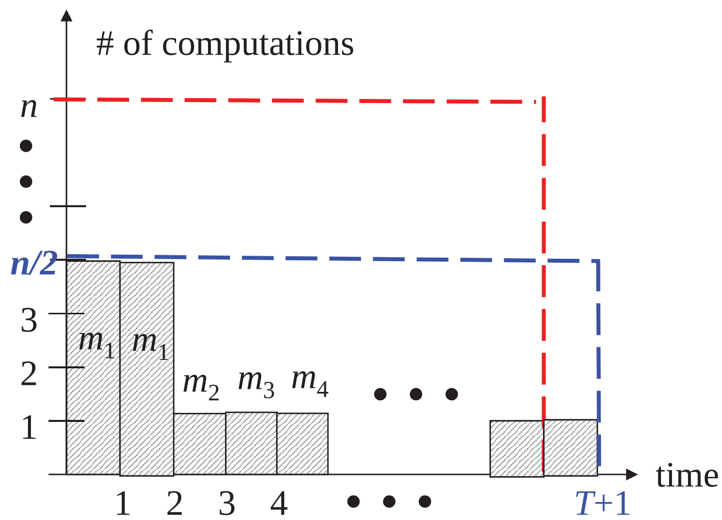


# Usage of Brent's theorem:

reduce cost (make cost close to M) when cost >> M




# of PEs: n
Time = T
Cost = n × T
M = n + T - 1




# of PEs: n/2
Time = T + 1
Cost = n/2 × (T+1)
M = n + T - 1