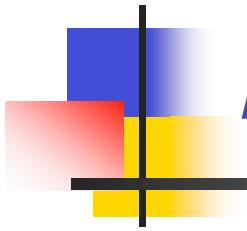
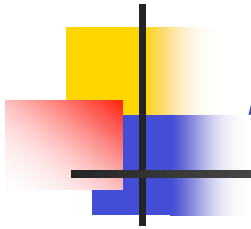
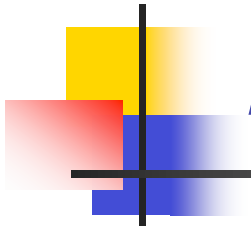


# List Ranking



# *List Ranking* (1)

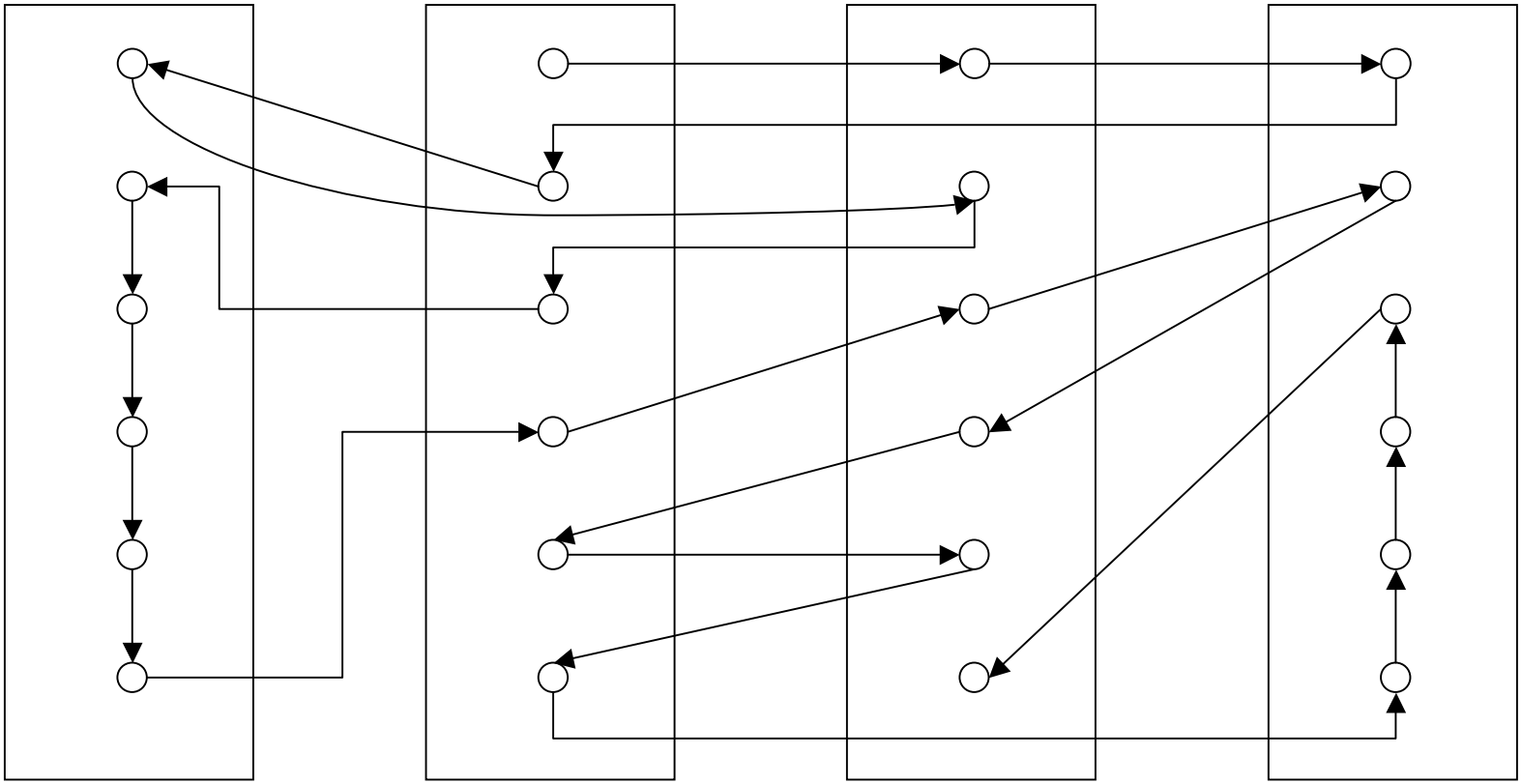
- Seja *L* uma lista representada por um vetor *s* tal que *s[i]* é o nó sucessor de *i* na lista *L*, para *u*, o último elemento da lista *L*, *s[u]=u*.
- A **distância** entre *i* e *j*, $d_L(i,j)$, é o número de nós entre *i* e *j* mais 1.
- O problema do **list ranking** consiste em computar para cada $i \in L$, a distância entre *i* e o último elemento *(j)*, denotado $rank_L(i)=d_L(i,j)$.

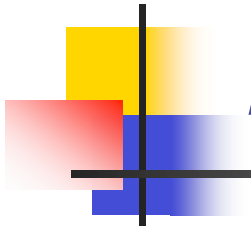


# *List Ranking* (2)

- O número de nós da lista cujos sucessores não estão armazenados no mesmo processador pode variar de 0 a *n/p*.
- Mesmo se todos os sucessores estiverem em um dado processador, após uma aplicação de duplicação recursiva (*pointer jumping)*, não há garantia que isto ainda ocorra nos passos seguintes.

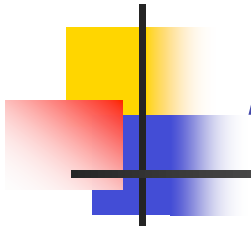
# *List Ranking* (3)





# *List Ranking* (4)

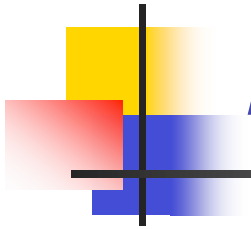
- O número de rodadas de comunicação pode chegar a *O(log n)*, uma vez que pode ser necessária uma comunicação para obter o sucessor de um de seus elementos.
- A simples aplicação de duplicação recursiva não leva a um algoritmo CGM eficiente.



# *List Ranking* (5)

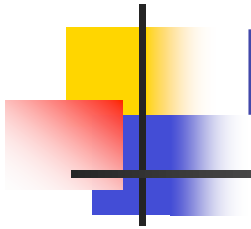
- Selecionar um conjunto de elementos $i^* \in L$, bem distribuídos em $L$, de tal forma que a distância de qualquer $i \in L$ a $i^*$ possa ser computada em $O(\log^k p)$ aplicações de *pointer jumping*.

*r-ruling set*



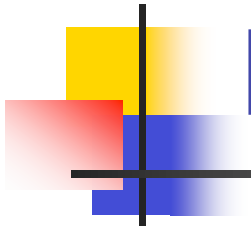
# *r-ruling set*

- Um ***r-ruling set*** de *L* é um subconjunto de elementos selecionados da lista *L* com as seguintes propriedades:
  - Dois vizinhos nunca são selecionados.
  - A distância entre qualquer elemento não selecionado ao próximo elemento selecionado é no máximo *r*.



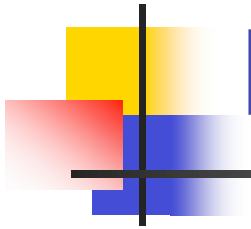
# Estratégia (1)

- Computar um *O(p²)-ruling set R* com */R/=O(n/p)* e difundir *R* para todos os processadores. O subconjunto *R* é representado por uma lista ligada onde cada elemento *i* é atribuído um ponteiro para o próximo elemento *j* de *R* com respeito à ordem induzida por *L*.



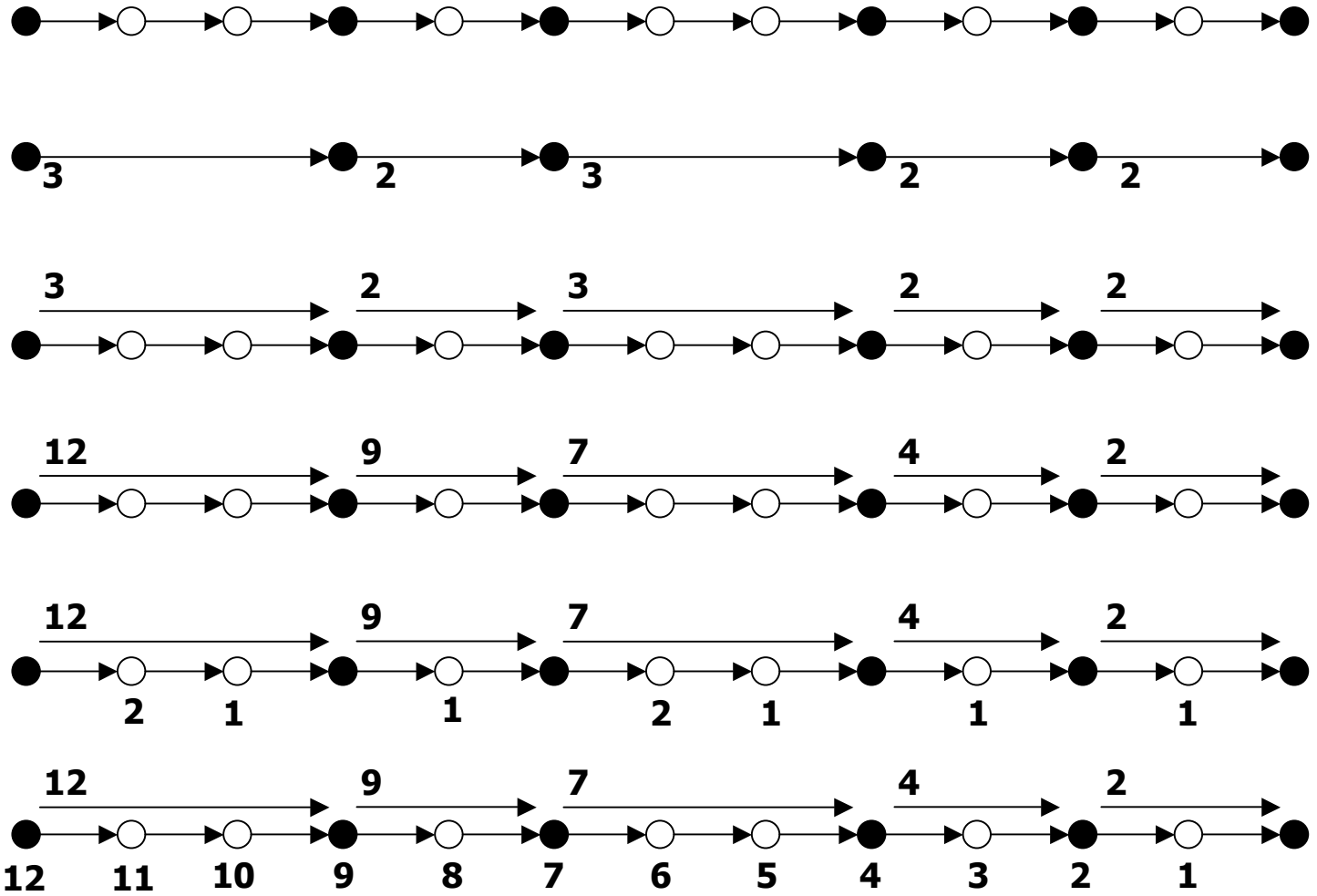
# Estratégia (2)

- Todo processador efetua seqüencialmente o *list ranking* de $R$, computando para cada $i \in R$ sua distância ao último elemento da lista.

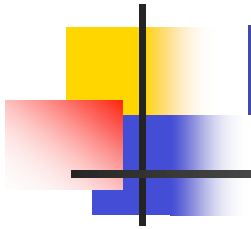


# Estratégia (3)

- A distância dos demais elementos da lista é obtida através da duplicação recursiva até que um elemento de *R* seja alcançado.
- Todos os outro elementos da lista têm no máximo a distância $O(p^2)$ do próximo elemento de *R* na lista.

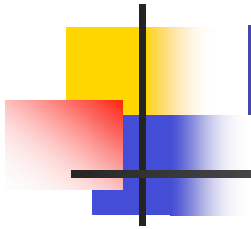

3 2 3 2 2
3 2 3 2 2
12 9 7 4 2
12 9 7 4 2
2 1 1 2 1 1 1
12 9 7 4 2
12 11 10 9 8 7 6 5 4 3 2 1

# Compressão determinística de listas (1)



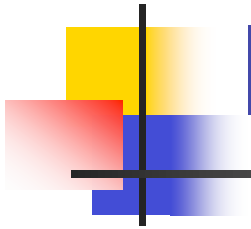
- O objetivo desta compressão é a determinação de um *O(p²)-ruling set* em *O(log p)* passos de comunicação.
- Uma **compressão determinística de listas** é composta de uma seqüência alternada de **fases de compressão** e **de concatenação**.
- Na fase de compressão, seleciona-se um subconjunto de elementos da lista *L*,utilizando um esquema de rotulação (*deterministic coin tossing*).

# Compressão determinística de listas (2)



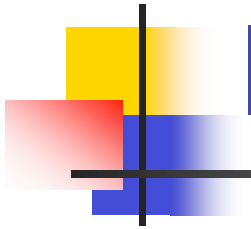
- A fase de concatenação consiste da construção de uma lista ligada, através da duplicação recursiva, com os elementos selecionados na fase de compressão.

# Compressão determinística de listas (3)



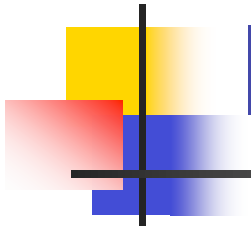
- Um *s*-intervalo de comprimento *k* é uma seqüência $I=(i_1, ..., i_k)$ de elementos da lista tal que $s[i_j] = i_{j+1}$, $1 <= j <= k-1$.
- Os dois vizinhos $n_1$ e $n_2$ do *s*-intervalo *I* são tais que $s[n_1]=i_1$ e $s[i_k]=n_2$.
- Um *s*-intervalo maximal *I* de elementos da lista cujos elementos estão todos no mesmo processador é dito *s*-intervalo local.
- Um *s*-intervalo maximal *I* de elementos da lista tal que quaisquer dois elementos consecutivos não estão no mesmo processador é denominado *s*-intervalo não-local.

# Compressão determinística de listas (3)



- O rótulo $l(i)$, $\forall\, i \in L$, na fase de compressão é o número do processador $p$ que armazena o nó $i$.
- Neste esquema, cada elemento de $L$ tem no máximo $p$ rótulos distintos.
- Seja $M=\{i, i+1, \ldots, i+k\} \subseteq L$, tal que $l(i) \neq l(s([i]), \forall\, i \in L$, onde $s[i]$ é um máximo local se $l(i)<l(s[i])>l(s[s[i]])$.

# Compressão determinística de listas (4)



- Selecionando apenas máximos locais não há garantia de distância menor que $O(p)$.
- Pode haver $L' = \{j, j+1, \ldots, j+k\} \subseteq L$, onde $l(s[j]) = l(j)$, $\forall\, j \in L'$ e $k > p$.
- Selecionamos todos os segundos elementos.

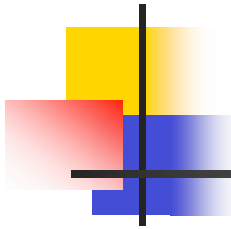
# Algoritmo *p²-ruling set*

- Entrada: *L* representada pelo vetor *s*.
- Saída: $R \subset L$ de nós selecionados.

☞ Cada processador localmente marca todos seus elementos como **não-selecionados**.

☞ Cada processador executa para cada um dos elementos armazenados:

  ☞ se *l(i)<l(s[i])>l(s[s[i]])* *então s[i]* é **selecionado**.

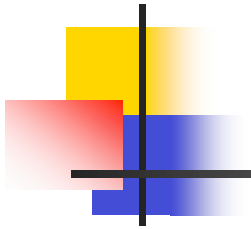


# Algoritmo *p²-ruling set*

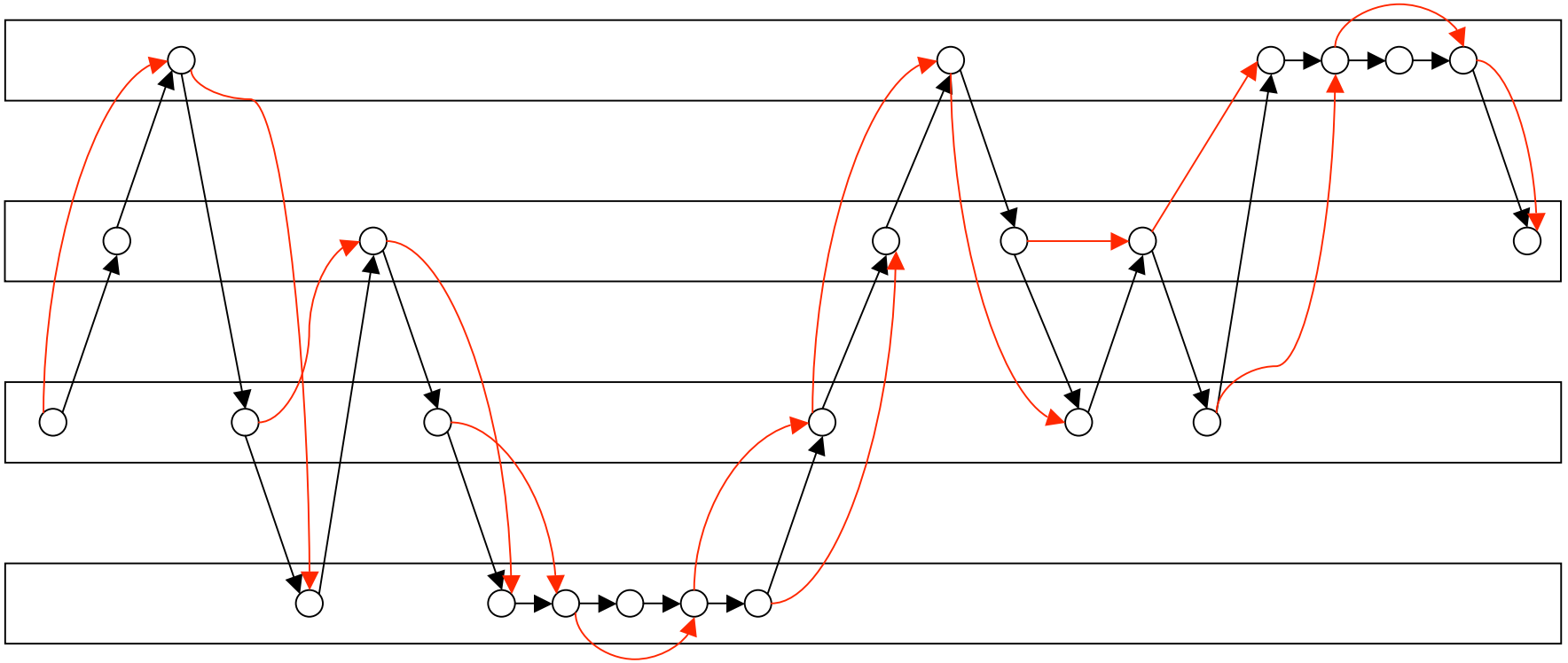
- Cada processador localmente determina seus *s*-intervalos locais. Para cada *s*-intervalo local de comprimento maior que 2, todo segundo elemento é marcado como **selecionado**. Se algum *s*-intervalo tem comprimento menor que 2 e nenhum de seus vizinhos tem um rótulo menor, então ambos elementos são marcados como **não-selecionados**.

# Algoritmo *p²-ruling set*

- para *k=1* até *log p* faça
  - Cada processador localmente executa para cada elemento *i* da lista:
    se *s[i]* é **não-selecionado** então *s[i]=s[s[i]]*.
  - Cada processador localmente executa para cada elemento *i* da lista:
    se (*i,s[i]* e *s[s[i]]* estão **selecionados**) E (*l(i)<l(s[i])>l(s[s[i]])*) E (*l(i)* ≠ *l(s[i])*) E (*l(s[i]* ≠ *l(s[s[i]])* então marcar *s[i]* como **não-selecionado**.



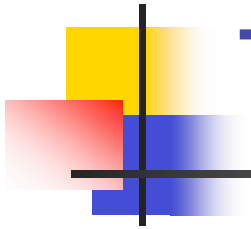
- Cada processador examina seus *s*-intervalos locais. Para cada *s*-intervalo de comprimento maior que dois, todo segundo elemento é marcado como **não-selecionado**. Se um *s*-intervalo tem comprimento dois  e nenhum de seus vizinhos tem um rótulo menor, então ambos elementos são marcados como **não-selecionados**.
- O processador que armazena o último elemento de *L* é marcado como selecionado.



# Algoritmo *p²-ruling set* - desempenho

- Teorema

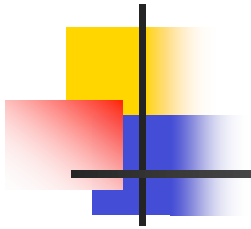
  O Algoritmo computa um *p²-ruling set R*, onde $|R|=O(n/p)$ usando $O(\log^2 p)$ rodadas de comunicação e $O(n/p)$ computação local por rodada.



# Tempo

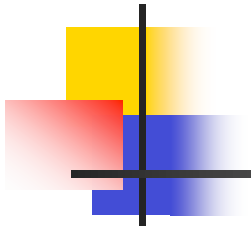
- Lema 1

  Após a *k*-ésima iteração no Passo 2, não existem mais que dois elementos selecionados entre quaisquer $2^k$ elementos subseqüentes.



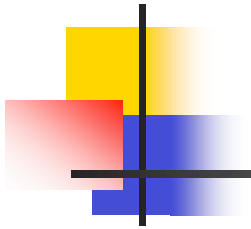
- Lema 2

  Após cada execução do Passo 2.3, a distância de dois elementos subseqüentes selecionados, com respeito aos ponteiros atuais (representado pelo vetor *s*), é no máximo *O(p)*.



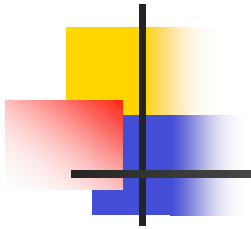
- Lema 3

  Após a *k*-ésima iteração do Passo 2.3, dois elementos subseqüentes com respeito aos ponteiros originais (representado pelo vetor *s*) têm distância $O(2^k)$ com respeito à lista original *L.*



- Lema 4

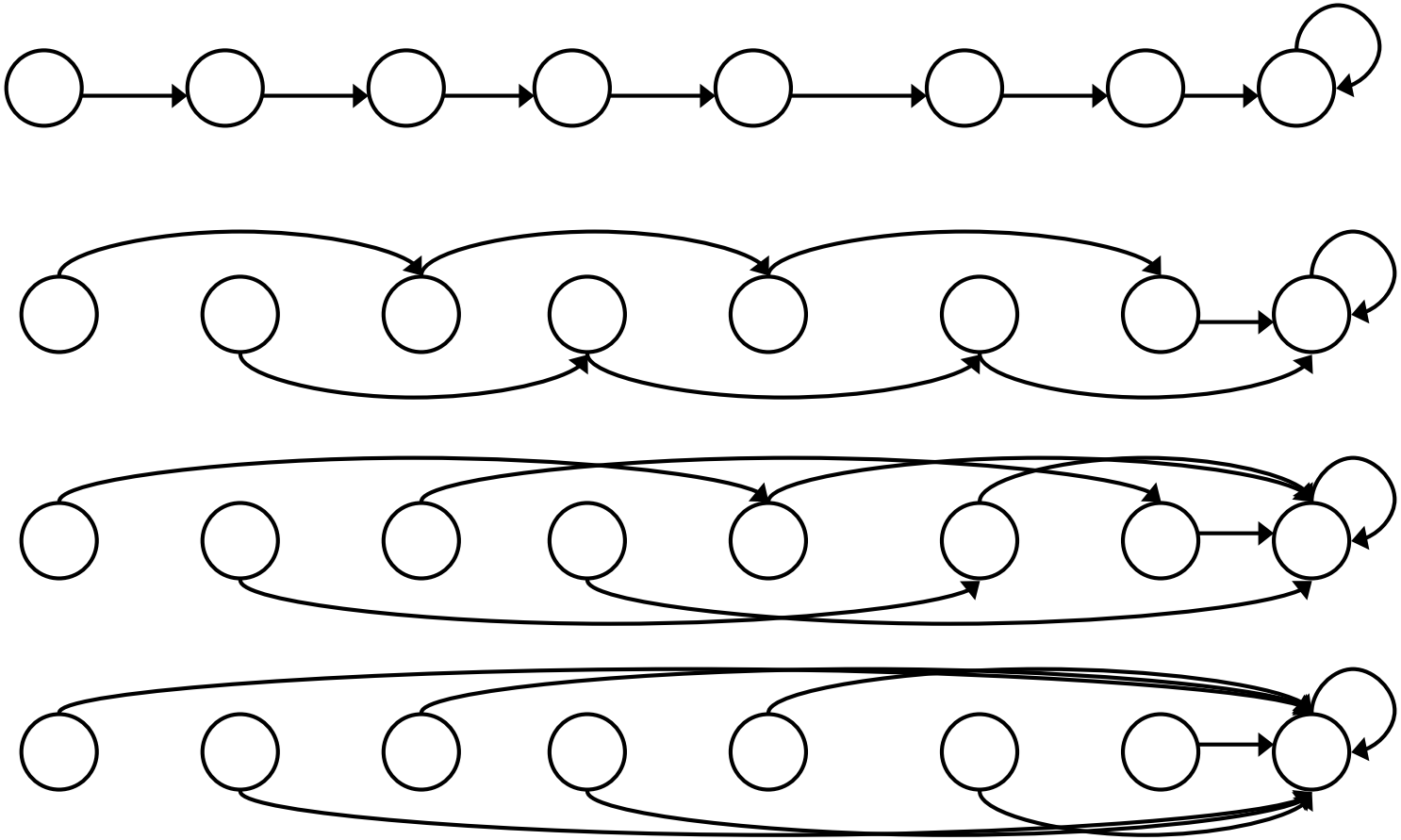
  Quaisquer dois elementos subseqüentes selecionados não distam um do outro mais que $O(p^2)$ com respeito à lista original $L$.



- Teorema

  O problema de *list ranking* para uma lista *L* com *n* vértices pode ser resolvido no modelo CGM com *p* processadores e *O(n/p)* memória local por processador usando *O(log p)* rodadas de comunicação e *O(n/p)* computação local por rodada.

# List Ranking - PRAM



# List Ranking - CGM