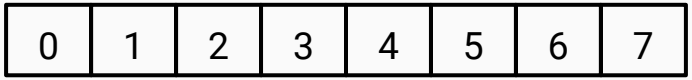
# Laboratorul 9
# Operații distribuite în MPI

# Reduce

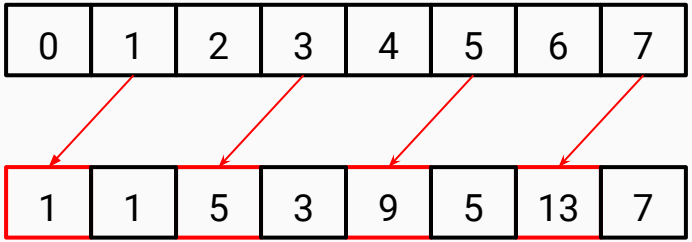
Operația de reduce (cunoscută în limbajele funcționale - Haskell, Racket - drept fold) reprezintă un tip de operație prin care elementele unei colecții sunt restrânse / acumulate într-un singur rezultat, printr-o singură operație aplicată între elementele unei colecții (+, *, min, max etc.)

# Reduce




În acest exemplu, fiecare proces are câte un element (8 elemente - 8 procese)

# Reduce

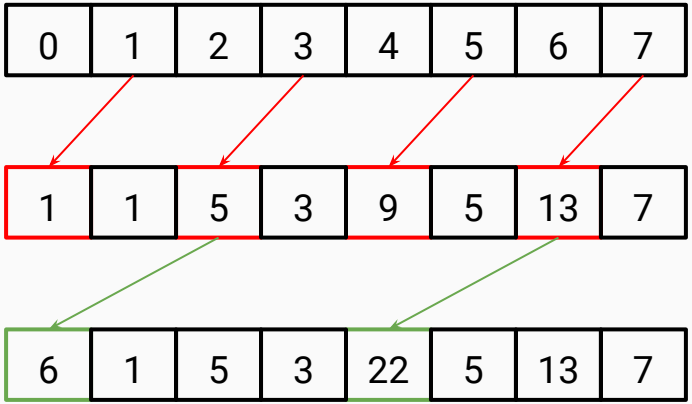



- procesul 1 trimite procesului 0, care adună valoarea primită (1 + 0 = 1)
- procesul 3 trimite procesului 2, care adună valoarea primită (3 + 2 = 5)
- procesul 5 trimite procesului 4, care adună valoarea primită (5 + 4 = 9)
- procesul 7 trimite procesului 6, care adună valoarea primită (7 + 6 = 13)

Deci:

- procesul 0 are valoarea 1
- procesul 2 are valoarea 5
- procesul 4 are valoarea 9
- procesul 6 are valoarea 13

# Reduce

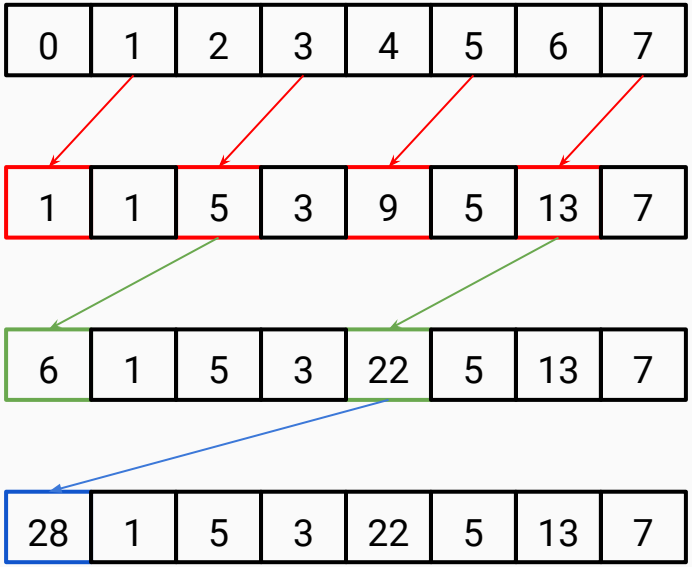



- procesul 2 trimite procesului 0, care adună valoarea primită (5 + 1 = 6)
- procesul 6 trimite procesului 4, care adună valoarea primită (13 + 9 = 22)

Deci:

- procesul 0 are valoarea 6
- procesul 4 are valoarea 22

# Reduce




- procesul 4 trimite procesului 0, care adună valoarea primită (22 + 6 = 28)

Deci:

- procesul 0 are valoarea 28

# Reduce - pseudocod

```
for (pas = 2; pas <= nr_procese; pas *= 2)
    if (rank % pas == 0 && (rank + (pas / 2)) < nr_procese)
        primește la procesul cu rank-ul [rank + (pas / 2)]
        adună
    else if (rank % (pas / 2) == 0 && (rank - (pas / 2)) >= 0)
        trimite la procesul cu rank-ul [rank - (pas / 2)]
```

# Reduce - pseudocod - rulare

pas = 2:

- 1 (1 % (2 / 2) == 0) trimite lui 0 (1 - (2 / 2))
- 0 (0 % 2 == 0) primește de la 1 (0 + (2 / 2))

- 3 (3 % (2 / 2) == 0) trimite lui 2 (3 - (2 / 2))
- 2 (2 % 2 == 0) primește de la 3 (2 + (2 / 2))

- 5 (5 % (2 / 2) == 0) trimite lui 4 (5 - (2 / 2))
- 4 (4 % 2 == 0) primește de la 5 (4 + (2 / 2))

- 7 (7 % (2 / 2) == 0) trimite lui 6 (7 - (2 / 2))
- 6 (6 % 2 == 0) primește de la 7 (6 + (2 / 2))

# Reduce - pseudocod - rulare

pas = 4:

- 2 (2 % (4 / 2) == 0) trimite lui 0 (2 - (4 / 2))
- 0 (0 % 4 == 0) primește de la 2 (0 + (4 / 2))

- 6 (6 % (4 / 2) == 0) trimite lui 4 (6 - (4 / 2))
- 4 (0 % 4 == 0) primește de la 6 (4 + (4 / 2))

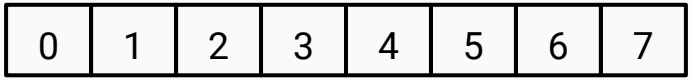
pas = 8:

- 4 (4 % (8 / 2) == 0) trimite lui 0 (4 - (8 / 2))
- 0 (0 % 8 == 0) primește de la 4 (0 + (8 / 2))

# Scan

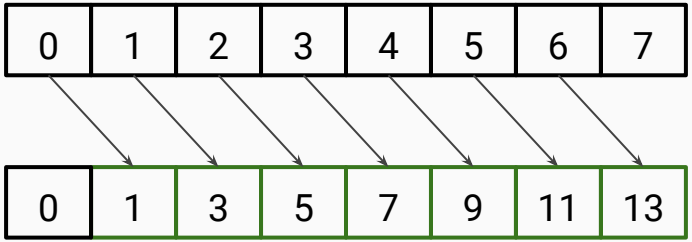
Operația de scan este similară operației de reduce (acumularea elementelor unei colecții într-un singur rezultat). Diferența față de reduce este că ultimul proces din comunicator deține rezultatul final, practic scan reprezintă inversul lui reduce.

# Scan



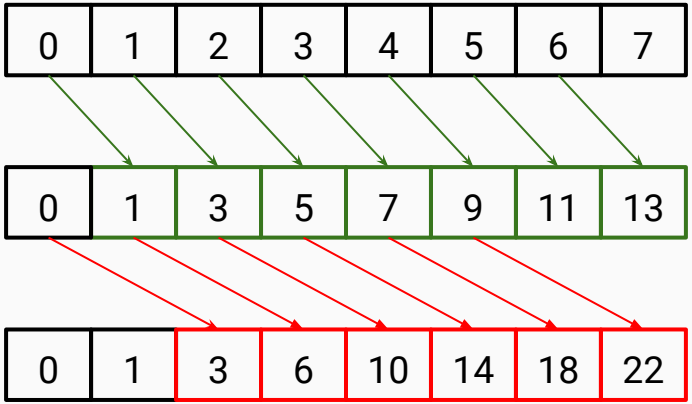

În acest exemplu, fiecare proces are câte un element (8 elemente - 8 procese)

# Scan



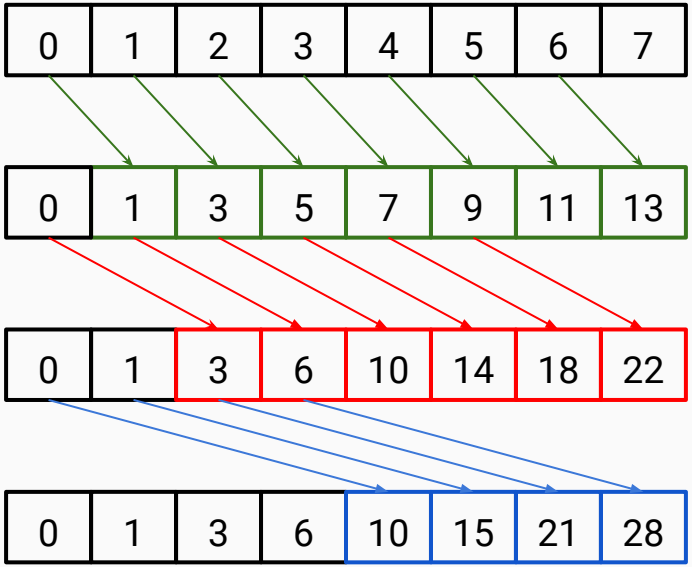

- procesul 0 trimite procesului 1, care adună valoarea primită (0 + 1 = 1)
- procesul 1 trimite procesului 2, care adună valoarea primită (1 + 2 = 3)
- procesul 2 trimite procesului 3, care adună valoarea primită (2 + 3 = 5)
- procesul 3 trimite procesului 3, care adună valoarea primită (3 + 4 = 7)
- procesul 4 trimite procesului 5, care adună valoarea primită (4 + 5 = 9)
- procesul 5 trimite procesului 6, care adună valoarea primită (5 + 6 = 11)
- procesul 6 trimite procesului 7, care adună valoarea primită (6 + 7 = 13)

# Scan




- procesul 0 trimite procesului 2, care adună valoarea primită (0 + 3 = 3)
- procesul 1 trimite procesului 3, care adună valoarea primită (1 + 5 = 6)
- procesul 2 trimite procesului 4, care adună valoarea primită (3 + 7 = 10)
- procesul 3 trimite procesului 5, care adună valoarea primită (5 + 9 = 14)
- procesul 4 trimite procesului 6, care adună valoarea primită (7 + 11 = 18)
- procesul 5 trimite procesului 7, care adună valoarea primită (9 + 13 = 22)

# Scan




- procesul 0 trimite procesului 4, care adună valoarea primită (0 + 10 = 10)
- procesul 1 trimite procesului 5, care adună valoarea primită (1 + 14 = 15)
- procesul 2 trimite procesului 6, care adună valoarea primită (3 + 18 = 21)
- procesul 3 trimite procesului 7, care adună valoarea primită (6 + 22 = 28)

# Scan - pseudocod

```
for (pas = 1; pas < nr_procese; pas *= 2)
    if (rank + pas < nr_procese)
        trimite la procesul cu rank-ul [rank + pas]
    if (rank - pas >= 0)
        primește de la procesul cu rank-ul [rank - pas]
        adună
```

# Scan - pseudocod - rulare

pas = 1:

- 0 trimite lui 1 (0 + 1)
- 1 primește de la 0 (1 - 1)

- 1 trimite lui 2 (1 + 1)
- 2 primește de la 1 (2 - 1)

- 2 trimite lui 3 (2 + 1)
- 3 primește de la 2 (3 - 1)

- 3 trimite lui 4 (3 + 1)
- 4 primește de la 3 (4 - 1)

pas = 1:

- 4 trimite lui 5 (4 + 1)
- 5 primește de la 4 (5 - 1)

- 5 trimite lui 6 (5 + 1)
- 6 primește de la 5 (6 - 1)

- 6 trimite lui 7 (6 + 1)
- 7 primește de la 6 (7 - 1)

# Scan - pseudocod - rulare

pas = 2:

- 0 trimite lui 2 (0 + 2)
- 2 primește de la 0 (2 - 2)

- 1 trimite lui 3 (1 + 2)
- 3 primește de la 1 (3 - 2)

- 2 trimite lui 4 (2 + 2)
- 4 primește de la 2 (4 - 2)

- 3 trimite lui 5 (3 + 2)
- 5 primește de la 3 (5 - 2)

- 4 trimite lui 6 (4 + 2)
- 6 primește de la 4 (6 - 2)

- 5 trimite lui 7 (5 + 2)
- 7 primește de la 5 (7 - 2)

pas = 4:

- 0 trimite lui 4 (4 + 4)
- 4 primește de la 0 (4 - 4)

- 1 trimite lui 5 (1 + 4)
- 5 primește de la 1 (5 - 1)

- 2 trimite lui 6 (2 + 4)
- 6 primește de la 2 (6 - 4)

- 3 trimite lui 7 (3 + 4)
- 7 primește de la 3 (7 - 4)

# Broadcast

Operația de broadcast este o operație prin care un proces trimite o valoare tuturor proceselor din cadrul comunicatorului. Această operație este reprezentată, în MPI, de MPI_Bcast (remember laboratorul 8).

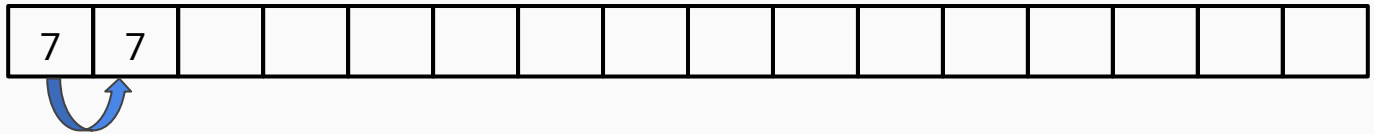
# Broadcast


În acest exemplu, rulăm cu 16 procese (fiecare proces, mai puțin procesul 0, nu are nicio valoare)

```
for (pas = 1; pas < nr_procese; pas *= 2)
    if (rank < pas and rank + pas < nr_procese)
        trimite la procesul cu rank-ul [rank + pas]
    else if (rank >= pas and rank < pas * 2)
        primește de la procesul cu rank-ul [rank - pas]
```

# Broadcast

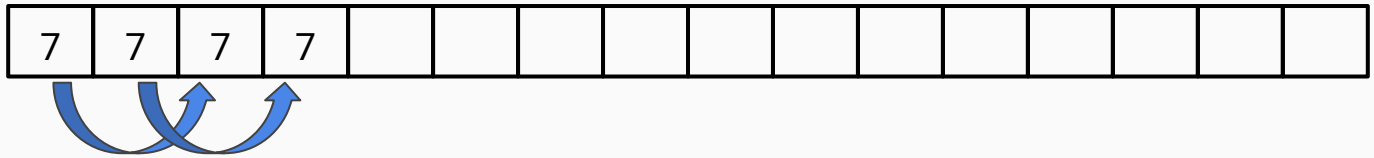
pas = 1




- procesul 0 (0 < 1, 0 + 1 < 16) trimite procesului 1 (0 + 1, 1 >= 1, 1 < 2 * 1)

# Broadcast

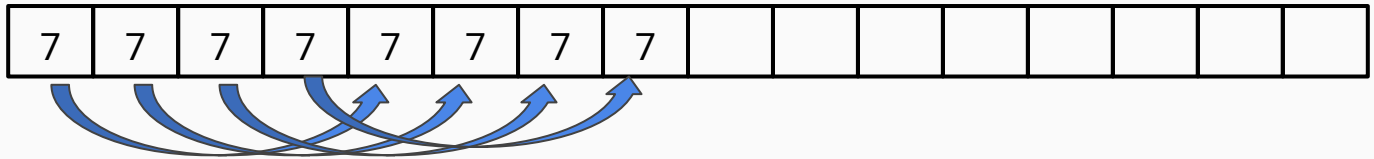
pas = 2




- procesul 0 (0 < 2, 0 + 2 < 16) trimite procesului 2 (0 + 2, 2 >= 2, 2 < 2 * 2)
- procesul 1 (1 < 2, 1 + 2 < 16) trimite procesului 3 (1 + 2, 3 >= 2, 3 < 2 * 2)

# Broadcast

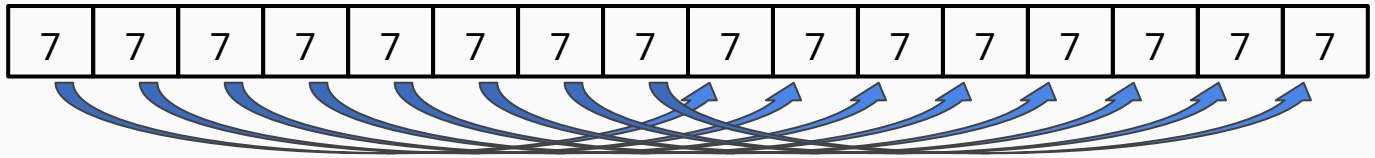
pas = 4




- procesul 0 (0 < 4, 0 + 4 < 16) trimite procesului 4 (0 + 4, 4 >= 4, 4 < 2 * 4)
- procesul 1 (1 < 4, 1 + 4 < 16) trimite procesului 5 (1 + 4, 5 >= 4, 5 < 2 * 4)
- procesul 2 (2 < 4, 2 + 4 < 16) trimite procesului 6 (2 + 4, 6 >= 4, 6 < 2 * 4)
- procesul 3 (3 < 4, 3 + 4 < 16) trimite procesului 7 (3 + 4, 7 >= 4, 7 < 2 * 4)

# Broadcast

pas = 8




- procesul 0 (0 < 8, 0 + 8 < 16) trimite procesului 8 (0 + 8, 8 >= 8, 8 < 2 * 8)
- procesul 1 (1 < 8, 1 + 8 < 16) trimite procesului 9 (1 + 8, 9 >= 8, 9 < 2 * 8)
- procesul 2 (2 < 8, 2 + 8 < 16) trimite procesului 10 (2 + 8, 10 >= 8, 10 < 2 * 8)
- procesul 3 (3 < 8, 3 + 8 < 16) trimite procesului 11 (3 + 8, 11 >= 8, 11 < 2 * 8)
- procesul 4 (4 < 8, 4 + 8 < 16) trimite procesului 12 (4 + 8, 12 >= 8, 12 < 2 * 8)
- procesul 5 (5 < 8, 5 + 8 < 16) trimite procesului 13 (5 + 8, 13 >= 8, 13 < 2 * 8)
- procesul 6 (6 < 8, 6 + 8 < 16) trimite procesului 14 (6 + 8, 14 >= 8, 14 < 2 * 8)
- procesul 7 (7 < 8, 7 + 8 < 16) trimite procesului 15 (7 + 8, 15 >= 8, 15 < 2 * 8)

# Pipeline

- În domeniul de computer science, un pipeline reprezintă un concept folosit în proiectarea procesoarelor și al echipamentelor hardware prin care se crește numărul de instrucțiuni ce sunt executate într-o unitate de timp.
- Acest concept poate fi extins în MPI, unde un proces produce rezultate intermerdiare, care sunt transmise către procesul următor, care procesează și prelucrează rezultatele primite, pe care le trimite mai departe, către procesul următor, totul repetându-se până la ultimul proces.

# Pipeline


În acest exemplu, avem 5 pași, deci 5 procese, unde fiecare prelucrează date și le trimite mai departe, către următorul proces

# Pipeline


Procesul 0 creează rezultate, pe care le trimite procesului 1

# Pipeline


Procesul 1 primește date de la procesul 0, le procesează, aplică operații asupra acestora și trimite rezultatele către procesul 2

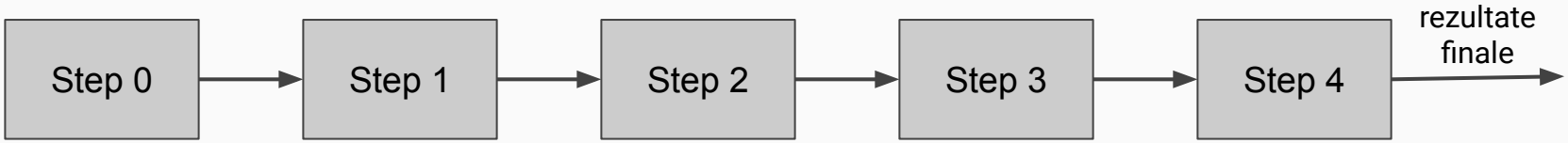
# Pipeline


Procesul 2 primește date de la procesul 1, le procesează, aplică operații asupra acestora și trimite rezultatele către procesul 3

# Pipeline


Procesul 3 primește date de la procesul 2, le procesează, aplică operații asupra acestora și trimite rezultatele către procesul 4

# Pipeline




Procesul 4 primește date de la procesul 3, le procesează, aplică operații asupra acestora și se obțin rezultatele finale din cadrul procesului de pipeline

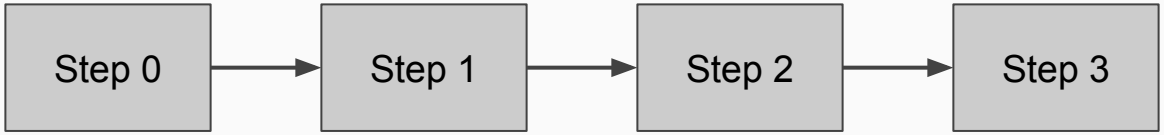
# Calculul unui polinom cu pipeline

- Un polinom, în matematică, reprezintă o ecuație de tipul: f(x) = x^2+ 2 * x + 1, unde x reprezintă variabila din ecuație.
- În MPI, putem calcula un polinom folosind tehnica pipeline în felul următor: numărul de coeficienți va fi egal cu numărul de procese (care este egal cu numărul de pași din procesul de pipeline). În cazul exemplului de mai sus, o să avem 3 procese, fiecare proces corespunzând cu câte un coeficient din ecuație

# Calcularea unui polinom

- Să luăm un exemplu de polinom: f(x) =  3 * x^3 + 4 * x^2 + 2 * x + 1
- Vrem să calculăm f(3) folosind polinom cu pipeline.
- În acest caz, avem un polinom cu 4 coeficienți (de grad 3), astfel vom lucra cu 4 procese (pipeline cu 4 pași de execuție)

# Calcularea unui polinom



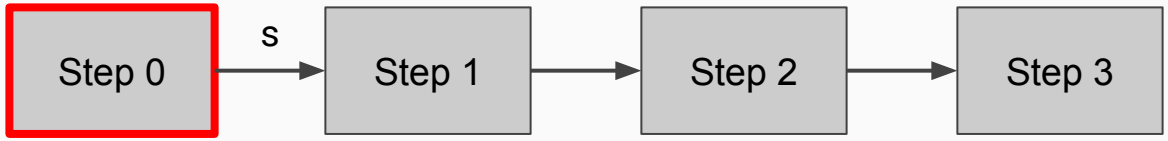

f(x) = 3 * x^3 + 4 * x^2 + 2 * x + 1

Calculăm f(3)

s = 0 (s va reprezenta în final rezultatul pipeline-ului, adică f(3)

# Calcularea unui polinom

f(3), f(x) = 3 * x^3 + 4 * x^2 + 2 * x + 1



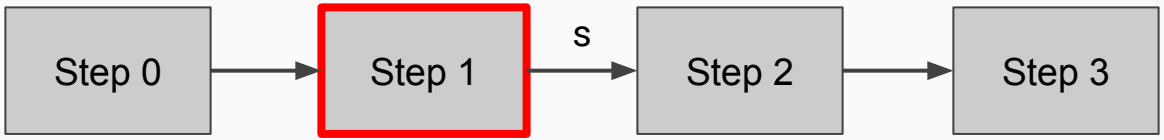

Procesul 0 se ocupă de x^0

```
s += 1 * (3 ^ 0) => s = 1
```

Procesul 0 trimite s către procesul 1

# Calcularea unui polinom

f(3), f(x) = 3 * x^3 + 4 * x^2 + 2 * x + 1



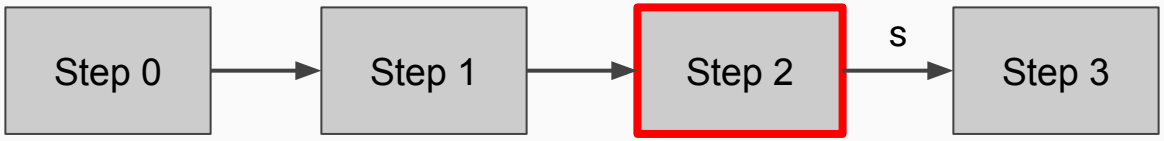

Procesul 1 primește rezultate de
la procesul 0
Procesul 1 se ocupă de x^1

```
s += 2 * (3 ^ 1) => s = 7
```

Procesul 1 trimite s către
procesul 2

# Calcularea unui polinom

f(3), f(x) = 3 * x^3 + 4 * x^2 + 2 * x + 1




Procesul 2 primește rezultate de la procesul 1
Procesul 2 se ocupă de x^2

```
s += 4 * (3 ^ 2) => s = 43
```

Procesul 2 trimite s către procesul 3

# Calcularea unui polinom

f(3), f(x) = 3 * x^3 + 4 * x^2 + 2 * x + 1


Procesul 3 primește rezultate de la procesul 2
Procesul 3 se ocupă de x^3

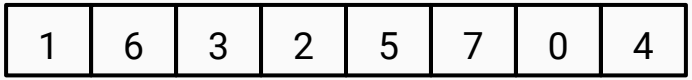
```
s += 3 * (3 ^ 3) => s = 124
```

Procesul 3 este ultimul și întoarce rezultatul final

# Sortarea folosind pipeline

- Dându-se un vector cu n elemente în ordine aleatoare, dorim să-l sortăm prin metoda pipeline, folosind MPI. Pentru aceasta, vom folosi o rețea cu n + 1 noduri (1 master și n workeri).
- Punctul final este momentul în care fiecare proces worker deține stocat unul din numerele din vectorul inițial, iar rank[i]→saved_value < rank[i + 1]→saved_value pentru orice i din [1, n).
- Fiecare worker este inițializat la început cu o valoare (-1 sau depinde de restricții), va primi (n - rank) numere de la rank-ul anterior și va trimite mai departe rank-ului următor (n - rank - 1) numere.
- Atunci când primește un număr, worker-ul îl va compara cu valoarea stocată:
  - dacă recv_value < actual_value, atunci salvăm noua valoare și o trimitem pe cea veche procesului următor
  - dacă recv_value >= actual_value, atunci o trimitem direct procesului următor

# Sortarea cu pipeline

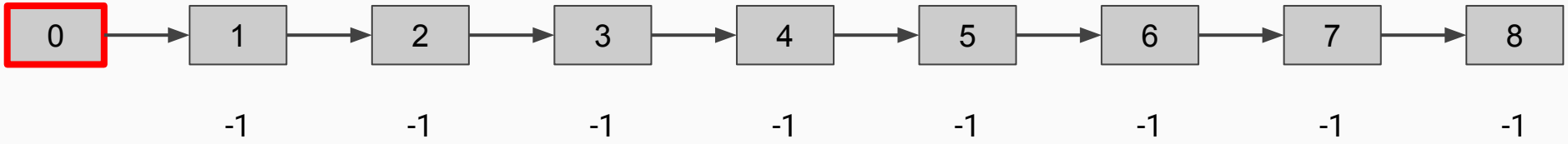



Avem un array nesortat de 8 elemente, așadar vom avea 9 procese:

- un proces master, care începe procesul de pipeline
- 8 procese worker, care se ocupă efectiv de sortarea array-ului

# Sortarea cu pipeline

# Sortarea cu pipeline




Procesul 0 (master):

- asignează o valoare inițială fiecărui proces worker (să spunem -1) - trimite acea valoare proceselor worker
- trimite întreg array-ul către procesul 1 (dim_array - rank, în acest caz dim_array - 0, rank = 0)

# Sortarea cu pipeline


actual_value = -1
primește de la procesul 0: [1, 6, 3, 2, 5, 7, 0, 4]

```
if (actual_value == -1):
     actual_value = recv_value
else if (actual_value <= recv_value)
     trimite recv_value către următorul proces
else
     trimite actual_value către următorul proces
     actual_value = recv_value
```

# Sortarea cu pipeline


actual_value = -1
[1, 6, 3, 2, 5, 7, 0, 4]

recv_value = 1
actual_value == -1 => actual_value = recv_value = 1

actual_value = 1

# Sortarea cu pipeline


actual_value = 1
[1, 6, 3, 2, 5, 7, 0, 4]

recv_value = 6
actual_value <= recv_value => trimite recv_value (6) către procesul 2

actual_value = 1

# Sortarea cu pipeline


actual_value = 1
[1, 6, 3, 2, 5, 7, 0, 4]

recv_value = 3
actual_value <= recv_value => trimite recv_value (3) către procesul 2

actual_value = 1

# Sortarea cu pipeline


actual_value = 1
[1, 6, 3, 2, 5, 7, 0, 4]

recv_value = 2
actual_value <= recv_value => trimite recv_value (2) către procesul 2

actual_value = 1

# Sortarea cu pipeline


actual_value = 1
[1, 6, 3, 2, 5, 7, 0, 4]

recv_value = 5
actual_value <= recv_value => trimite recv_value (5) către procesul 2

actual_value = 1

# Sortarea cu pipeline


actual_value = 1
[1, 6, 3, 2, 5, 7, 0, 4]

recv_value = 7
actual_value <= recv_value => trimite recv_value (7) către procesul 2

actual_value = 1

# Sortarea cu pipeline


actual_value = 1
[1, 6, 3, 2, 5, 7, 0, 4]

recv_value = 0
actual_value > recv_value => trimite actual_value (1) către procesul 2, actual_value = recv_value = 0

actual_value = 0

# Sortarea cu pipeline


actual_value = 0
[1, 6, 3, 2, 5, 7, 0, 4]

recv_value = 4
actual_value <= recv_value => trimite recv_value (4) către procesul 2

actual_value = 0

**Stop: procesul 1 va avea valoarea 0, se trece la procesul 2, iar procesul 1 trimite valoarea procesului master**

# Sortarea cu pipeline


actual_value = -1
primește de la procesul 1: [6, 3, 2, 5, 7, 1, 4]

# Sortarea cu pipeline


actual_value = -1
[6, 3, 2, 5, 7, 1, 4]

recv_value = 6
actual_value == -1 => actual_value = recv_value = 6

actual_value = 6

# Sortarea cu pipeline


actual_value = 6
[6, 3, 2, 5, 7, 1, 4]

recv_value = 3
actual_value > recv_value => trimite actual_value (6) către procesul 3, actual_value = recv_value = 3

actual_value = 3

# Sortarea cu pipeline


actual_value = 3
[6, 3, 2, 5, 7, 1, 4]

recv_value = 2
actual_value > recv_value => trimite actual_value (3) către procesul 3, actual_value = recv_value = 2

actual_value = 2

# Sortarea cu pipeline


actual_value = 3
[6, 3, 2, 5, 7, 1, 4]

recv_value = 5
actual_value <= recv_value => trimite recv_value (5) către procesul 3

actual_value = 3

# Sortarea cu pipeline


actual_value = 3
[6, 3, 2, 5, 7, 1, 4]

recv_value = 7
actual_value <= recv_value => trimite recv_value (7) către procesul 3

actual_value = 3

# Sortarea cu pipeline


actual_value = 2
[6, 3, 2, 5, 7, 1, 4]

recv_value = 1
actual_value > recv_value => trimite actual_value (2) către procesul 3, actual_value = recv_value = 1

actual_value = 1

# Sortarea cu pipeline


actual_value = 1
[6, 3, 2, 5, 7, 1, 4]

recv_value = 4
actual_value <= recv_value => trimite recv_value (4) către procesul 3

actual_value = 1

**Stop: procesul 2 va avea valoarea 1, se trece la procesul 3, iar procesul 2 trimite valoarea procesului master**

# Sortarea cu pipeline


actual_value = -1
primește de la procesul 2: [6, 3, 5, 7, 2, 4]

# Sortarea cu pipeline


actual_value = -1
[6, 3, 5, 7, 2, 4]

recv_value = 6
actual_value == -1 => actual_value = recv_value = 6

actual_value = 6

# Sortarea cu pipeline


actual_value = 6
[6, 3, 5, 7, 2, 4]

recv_value = 3
actual_value > recv_value => trimite actual_value (6) către procesul 4, actual_value = recv_value = 3

actual_value = 3

# Sortarea cu pipeline


actual_value = 3
[6, 3, 5, 7, 2, 4]

recv_value = 5
actual_value <= recv_value => trimite recv_value (5) către procesul 4

actual_value = 3

# Sortarea cu pipeline


actual_value = 3
[6, 3, 5, 7, 2, 4]

recv_value = 7
actual_value <= recv_value => trimite recv_value (7) către procesul 4

actual_value = 3

# Sortarea cu pipeline


actual_value = 3
[6, 3, 5, 7, 2, 4]

recv_value = 2
actual_value > recv_value => trimite actual_value (3) către procesul 4, actual_value = recv_value = 2

actual_value = 2

# Sortarea cu pipeline


actual_value = 2
[6, 3, 5, 7, 2, 4]

recv_value = 4
actual_value <= recv_value => trimite recv_value (4) către procesul 4

actual_value = 2

**Stop: procesul 3 va avea valoarea 2, se trece la procesul 4, iar procesul 3 trimite valoarea procesului master**

# Sortarea cu pipeline


actual_value = -1
primește de la procesul 3: [6, 5, 7, 3, 4]

# Sortarea cu pipeline


actual_value = -1
[6, 5, 7, 3, 4]

recv_value = 6
actual_value == -1 => actual_value = recv_value = 6

actual_value = 6

# Sortarea cu pipeline


actual_value = 6
[6, 5, 7, 3, 4]

recv_value = 5
actual_value > recv_value => trimite actual_value (6) către procesul 5, actual_value = recv_value = 5

actual_value = 5

# Sortarea cu pipeline


actual_value = 5
[6, 5, 7, 3, 4]

recv_value = 7
actual_value <= recv_value => trimite recv_value (7) către procesul 5

actual_value = 5

# Sortarea cu pipeline


actual_value = 5
[6, 5, 7, 3, 4]

recv_value = 3
actual_value > recv_value => trimite actual_value (5) către procesul 5, actual_value = recv_value = 3

actual_value = 3

# Sortarea cu pipeline


actual_value = 3
[6, 5, 7, 3, 4]

recv_value = 4
actual_value <= recv_value => trimite recv_value (4) către procesul 5

actual_value = 3

**Stop: procesul 4 va avea valoarea 3, se trece la procesul 5, iar procesul 4 trimite valoarea procesului master**

# Sortarea cu pipeline


actual_value = -1
primește de la procesul 4: [6, 7, 5, 4]

# Sortarea cu pipeline


actual_value = -1
[6, 7, 5, 4]

recv_value = 6
actual_value == -1 => actual_value = recv_value = 6

actual_value = 6

# Sortarea cu pipeline


actual_value = 6
[6, 7, 5, 4]

recv_value = 7
actual_value <= recv_value => trimite recv_value (7) către procesul 6

actual_value = 6

# Sortarea cu pipeline


actual_value = 6
[6, 7, 5, 4]

recv_value = 5
actual_value > recv_value => trimite actual_value (6) către procesul 6, actual_value = recv_value = 5

actual_value = 5

# Sortarea cu pipeline


actual_value = 5
[6, 7, 5, 4]

recv_value = 4
actual_value > recv_value => trimite actual_value (5) către procesul 6, actual_value = recv_value = 4

actual_value = 4

**Stop: procesul 5 va avea valoarea 4, se trece la procesul 6, iar procesul 5 trimite valoarea procesului master**

# Sortarea cu pipeline


actual_value = -1
primește de la procesul 5: [7, 6, 5]

# Sortarea cu pipeline


actual_value = -1
[7, 6, 5]

recv_value = 7
actual_value == -1 => actual_value = recv_value = 7

actual_value = 7

# Sortarea cu pipeline


actual_value = 7
[7, 6, 5]

recv_value = 6
actual_value > recv_value => trimite actual_value (7) către procesul 7, actual_value = recv_value = 6

actual_value = 6

# Sortarea cu pipeline


actual_value = 6
[7, 6, 5]

recv_value = 5
actual_value > recv_value => trimite actual_value (6) către procesul 7, actual_value = recv_value = 5

actual_value = 5

**Stop: procesul 6 va avea valoarea 5, se trece la procesul 7, iar procesul 6 trimite valoarea procesului master**

# Sortarea cu pipeline


actual_value = -1
primește de la procesul 6: [7, 6]

# Sortarea cu pipeline


actual_value = -1
[7, 6]

recv_value = 7
actual_value == -1 => actual_value = recv_value = 7

actual_value = 7

# Sortarea cu pipeline


actual_value = 6
[7, 6]

recv_value = 6
actual_value > recv_value => trimite actual_value (7) către procesul 8, actual_value = recv_value = 6

actual_value = 6

**Stop: procesul 7 va avea valoarea 6, se trece la procesul 8, iar procesul 7 trimite valoarea procesului master**

# Sortarea cu pipeline


actual_value = -1
primește de la procesul 7: [7]

# Sortarea cu pipeline


actual_value = -1
[7]

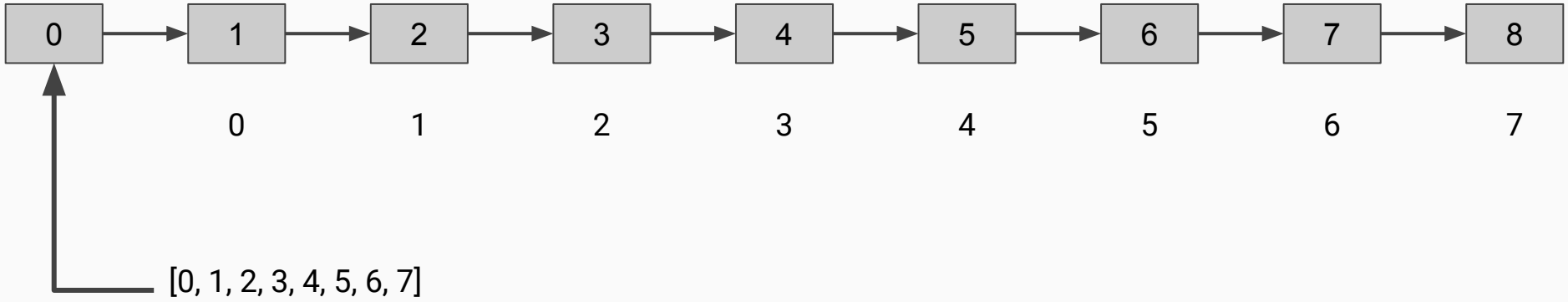
recv_value = 7
actual_value == -1 => actual_value = recv_value = 7

actual_value = 7

**Stop: procesul 8 va avea valoarea 7, fiind ultimul proces pipeline-ul se oprește, iar procesul 7 trimite valoarea procesului master**

**Procesul master are acum toate valorile primite de la procesele worker, în ordine sortată**

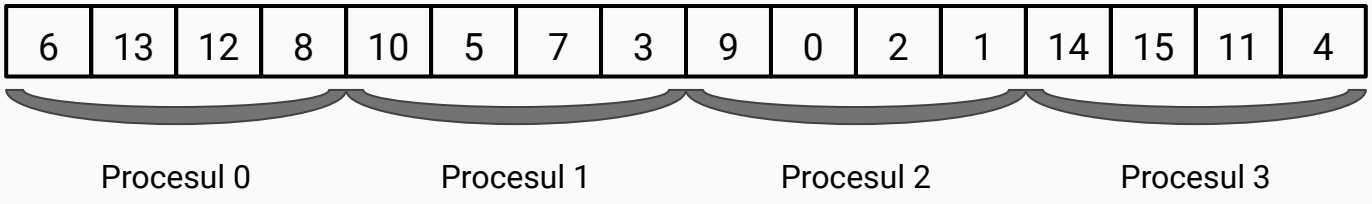
# Sortarea cu pipeline



# Rank Sort

- Rank Sort este un algoritm de sortare a unui vector folosindu-se de rangul fiecărui element din vector. Rangul unui element presupune câte numere sunt mai mici decât el în vector.
- Pași:
  - Se calculează rangul fiecărui element: pentru fiecare element din vector v[i], unde 0 <= i < N (N - dimensiunea vectorului), se numără câte elemente din vector v[j], j != i sunt mai mici: v[j] < v[i], 0 <= i, j < N, i != j.
  - Elementul de pe poziția “i” va fi pe poziția “rang” în noul vector sortat: q[rank] = v[i].
- Algoritmul “Parallel rank sort” are ca scop paralelizarea pasului 1. Rangul fiecărui element poate fi căutat în paralel, adică fiecare proces sau fir de execuție va realiza căutarea rangurilor pentru elementele din intervalul său: [start, end)

# Rank Sort




- În acest exemplu, avem 4 procese.
- Trebuie să vedem, pentru fiecare element din array, câte elemente sunt mai mici decat elementul respectiv.
- Fiecare proces va număra câte elemente sunt mai mici decât un anumit element în bucata sa din array (în acest caz de dimensiune 4 = dim_array / nr_procese).

# Rank Sort

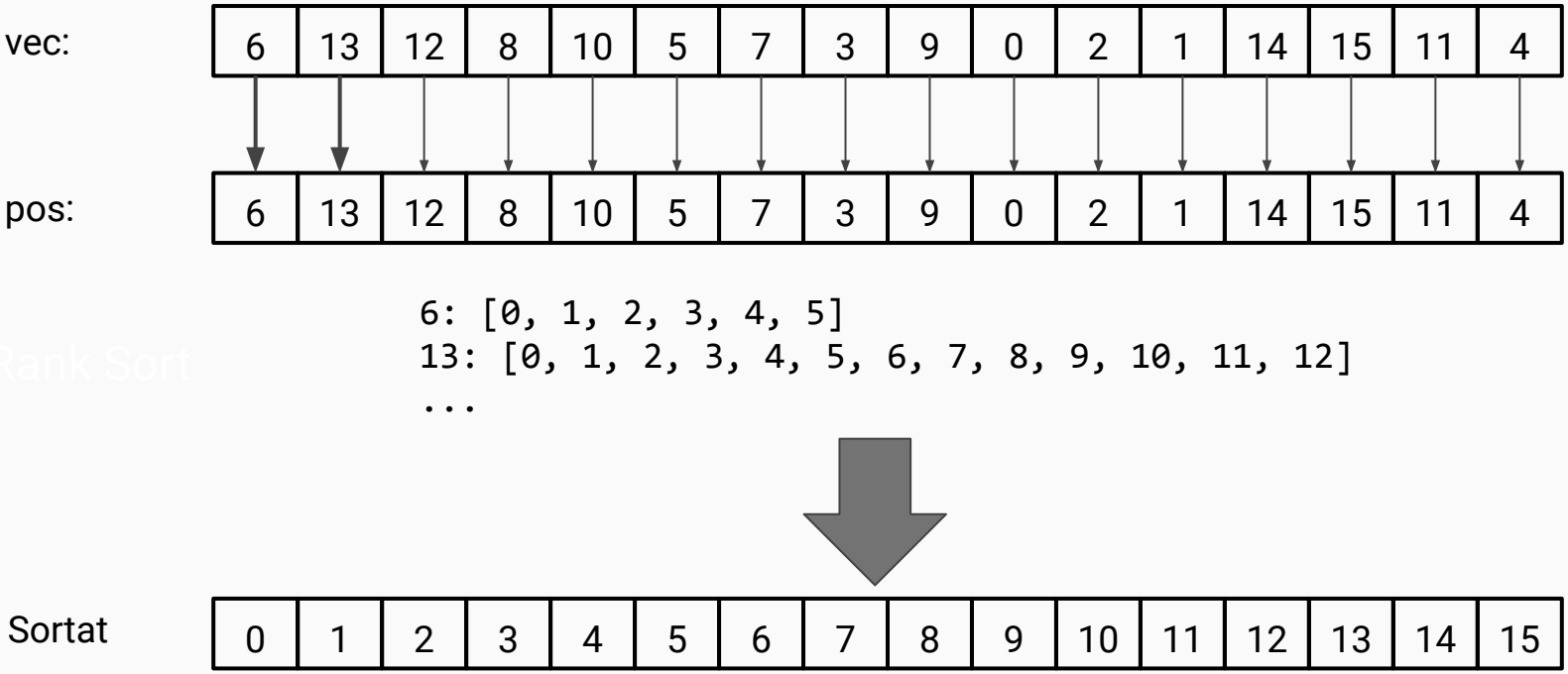
| 6 | 13 | 12 | 8 | 10 | 5 | 7 | 3 | 9 | 0 | 2 | 1 | 14 | 15 | 11 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

- Fiecare proces va avea întreg array-ul și va face numărarea elementelor mai mici în bucata sa de dimensiune dim_array / nr_procese, după cum este în pseudocodul de mai jos:

```
dim_proc = dim_array / nr_procese
for i = 0 to dim_proc
     for j = 0 to dim_array
          if v[rank * dim_proc + i] > v[j]
               pos[rank * dim_proc + i]++
```

- Procesele 1, 2, 3 etc. vor trimite bucata lor din array-ul pos către procesul 0

# Rank Sort




```
6: [0, 1, 2, 3, 4, 5]
13: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
...
```

Rank Sort