MARCIN COPIK <MARCIN.COPIK@INF.ETHZ.CH>

DPHPC: Prefix Scan

Recitation session, 14.11.2019

Org

- First project reports have to be sent by tomorrow! Please send directly to your supervisor. And cc Timo ☺

Agenda for today:

- Prefix scan
- Homework

Recap: oblivious algorithms

“*An algorithm is* ***data-oblivious*** *if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs*”

```
int reduce(int n, arr[n]) {
 for(int i=0; i<n; ++i)
  sum += arr[i];
}
```

Recap: oblivious algorithms

“An algorithm is ***data-oblivious*** *if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs”*

```
int reduce(int n, arr[n]) {
 for(int i=0; i<n; ++i)
  sum += arr[i];
}
```

```
int findmin(int n, a[n]) {
 for(int i=1; i<n; i++)
  if(a[i]<a[0]) a[0] = a[i];
}
```

Recap: oblivious algorithms

“An algorithm is ***data-oblivious*** *if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs”*

```
int reduce(int n, arr[n]) {
 for(int i=0; i<n; ++i)
  sum += arr[i];
}
```

```
int findmin(int n, a[n]) {
 for(int i=1; i<n; i++)
  if(a[i]<a[0]) a[0] = a[i];
}
```

```
int finditem(list_t list)
 item = list.head;
 while(item.value!=0 && item.next!=NULL)
  item=item.next;
}
```

Recap: oblivious algorithms

*“An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs”*

```
int reduce(int n, arr[n]) {
 for(int i=0; i<n; ++i)
  sum += arr[i];
}
```

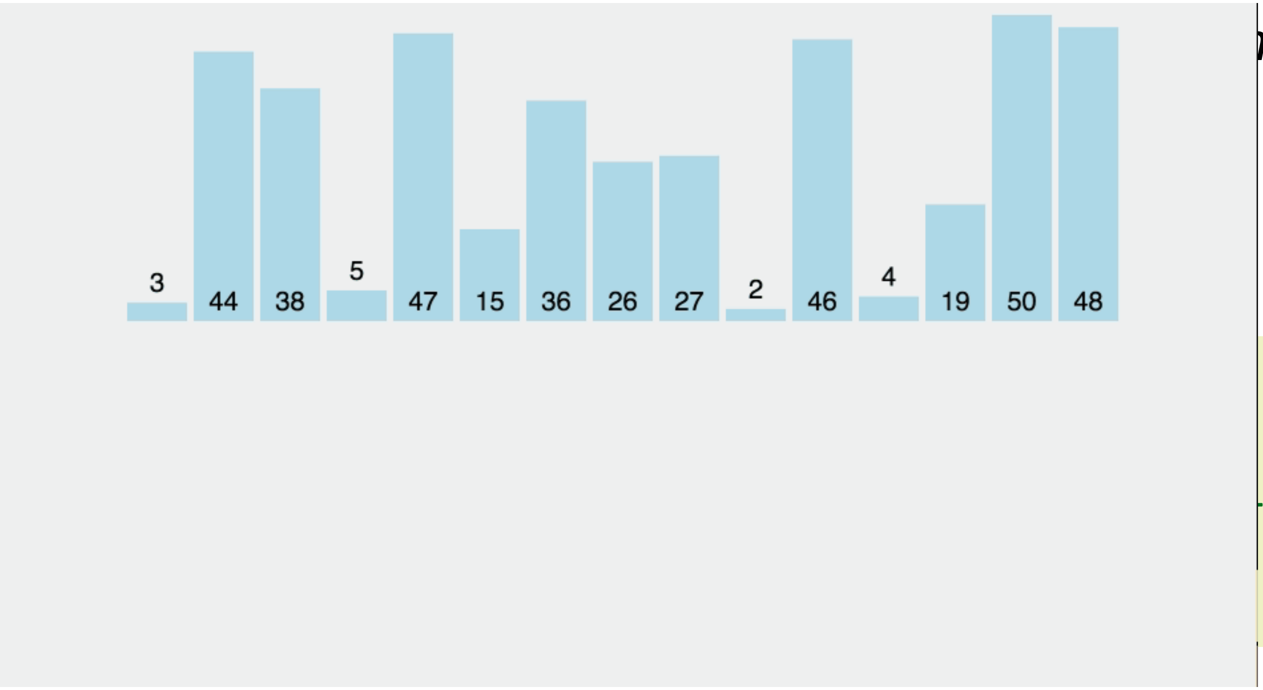
```
int findmin(int n, a[n]) {
 for(int i=1; i<n; i++)
  if(a[i]<a[0]) a[0] = a[i];
}
```

```
int finditem(list_t list)
 item = list.head;
 while(item.value!=0 && item.next!=NULL)
  item=item.next;
}
```

- Insert sort
- Prefix sum on an array
- Dense Cholesky decomposition ($A = LL^T$)
- Gradient descent
- Model checking: exhaustive verification

Recap: oblivious algorithms

*“An algorithm is **data-oblivious** if, for each p* *he*
set of memory locations read and the set of *are*
determined by the input size and are indepe



```
int reduce(int n, arr[n]) {
 for(int i=0; i<n; ++i)
  sum += arr[i];
}
```

```
int findmin(int
 for(int i=1; i<
  if(a[i]<a[0]) a
}
```

```
)
```

- Insert sort
- Prefix sum on an array
- Dense Cholesky decomposition (A = LL^T)
- Gradient descent
- Model checking: exhaustive verification

Recap: oblivious algorithms

*“An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs”*

```
int reduce(int n, arr[n]) {
 for(int i=0; i<n; ++i)
  sum += arr[i];
}
```

```
int findmin(int n, a[n]) {
 for(int i=1; i<n; i++)
  if(a[i]<a[0]) a[0] = a[i];
}
```

```
int finditem(list_t list)
 item = list.head;
 while(item.value!=0 && item.next!=NULL)
  item=item.next;
}
```

- Insert sort
- Prefix sum on an array
- Dense Cholesky decomposition ($A = LL^T$)
- Gradient descent
- Model checking: exhaustive verification

NO. Memory locations and instructions depend on data distribution.

Recap: oblivious algorithms

*“An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs”*

```
int reduce(int n, arr[n]) {
 for(int i=0; i<n; ++i)
  sum += arr[i];
}
```

```
int findmin(int n, a[n]) {
 for(int i=1; i<n; i++)
  if(a[i]<a[0]) a[0] = a[i];
}
```

```
int finditem(list_t list)
 item = list.head;
 while(item.value!=0 && item.next!=NULL)
  item=item.next;
}
```

- Insert sort
- Prefix sum on an array
- Dense Cholesky decomposition (A = LL^T)
- Gradient descent
- Model checking: exhaustive verification

NO. Memory locations and instructions depend on data distribution.
YES.

Recap: oblivious algorithms

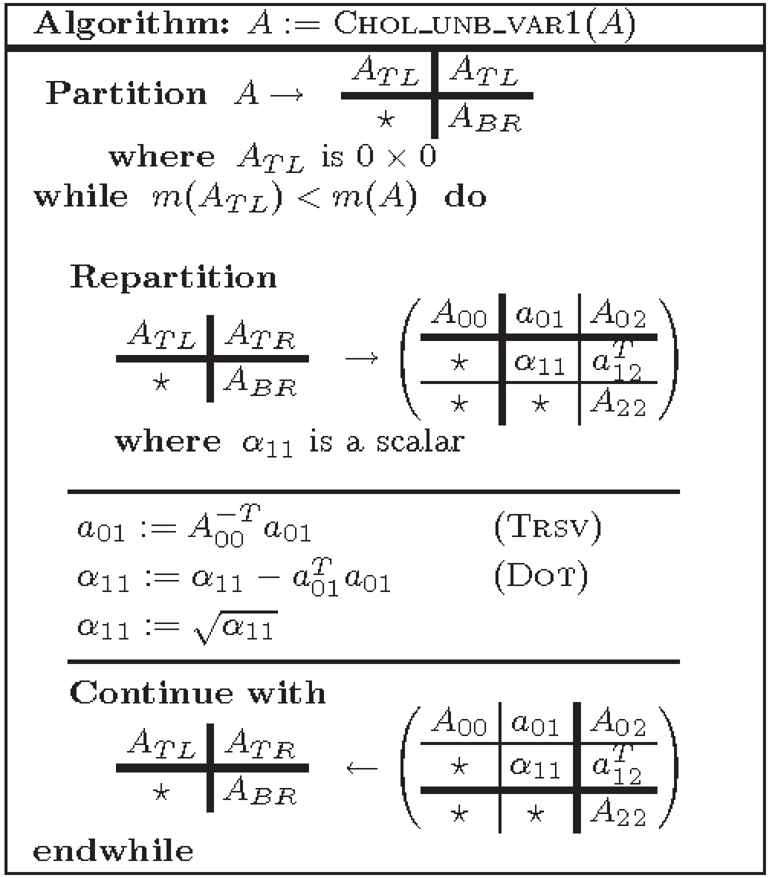
*“An algorithm is **data-oblivious** if, for each problem size, the s* … *, the set of memory locations read and the set of memory locations w* … *ion are determined by the input size and are independent of the values*

```
int reduce(int n, arr[n]) {
  for(int i=0; i<n; ++i)
    sum += arr[i];
}
```

```
int findmin(int n, a[n]) {
  for(int i=1; i<n; i++)
    if(a[i]<a[0]) a[0] = a[i];
}
```

```
ir
                    ULL)

}
```



```
Algorithm: $A := \text{CHOL\_UNB\_VAR1}(A)$
Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TL} \\ \hline \star & A_{BR} \end{array}\right)$
    where $A_{TL}$ is $0 \times 0$
while $m(A_{TL}) < m(A)$ do
    Repartition
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline \star & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline \star & \alpha_{11} & a_{12}^T \\ \hline \star & \star & A_{22} \end{array}\right)$
        where $\alpha_{11}$ is a scalar

    $a_{01} := A_{00}^{-T} a_{01}$                (TRSV)
    $\alpha_{11} := \alpha_{11} - a_{01}^T a_{01}$      (DOT)
    $\alpha_{11} := \sqrt{\alpha_{11}}$

    Continue with
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline \star & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline \star & \alpha_{11} & a_{12}^T \\ \hline \star & \star & A_{22} \end{array}\right)$
endwhile
```

- Insert sort
- Prefix sum on an array
- Dense Cholesky decomposition (A = LL^T)
- Gradient descent
- Model checking: exhaustive verification

NO. Memory locations and instructions depend on data distribution.
YES.

Recap: oblivious algorithms

*“An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs”*

```
int reduce(int n, arr[n]) {
 for(int i=0; i<n; ++i)
  sum += arr[i];
}
```

```
int findmin(int n, a[n]) {
 for(int i=1; i<n; i++)
  if(a[i]<a[0]) a[0] = a[i];
}
```

```
int finditem(list_t list)
 item = list.head;
 while(item.value!=0 && item.next!=NULL)
  item=item.next;
}
```

▪ Insert sort	**NO.** Memory locations and instructions depend on data distribution.
▪ Prefix sum on an array	**YES.**
▪ Dense Cholesky decomposition ($A = LL^T$)	**YES.**
▪ Gradient descent	
▪ Model checking: exhaustive verification	

Recap: oblivious algorithms

*“An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs”*

```
int reduce(int n, arr[n]) {
 for(int i=0; i<n; ++i)
  sum += arr[i];
}
```

```
int findmin(int n, a[n]) {
 for(int i=1; i<n; i++)
  if(a[i]<a[0]) a[0] = a[i];
}
```

```
int finditem(list_t list)
 item = list.head;
 while(item.value!=0 && item.next!=NULL)
  item=item.next;
}
```

▪ Insert sort	**NO.** Memory locations and instructions depend on data distribution.
▪ Prefix sum on an array	**YES.**
▪ Dense Cholesky decomposition ($A = LL^T$)	**YES.**
▪ Gradient descent	**NO.** Stopping condition depends on numerical result.
▪ Model checking: exhaustive verification	

Recap: oblivious algorithms

*"An algorithm is **data-oblivious** if, for each problem size, the sequence of instructions executed, the set of memory locations read and the set of memory locations written by each executed instruction are determined by the input size and are independent of the values of the other inputs"*

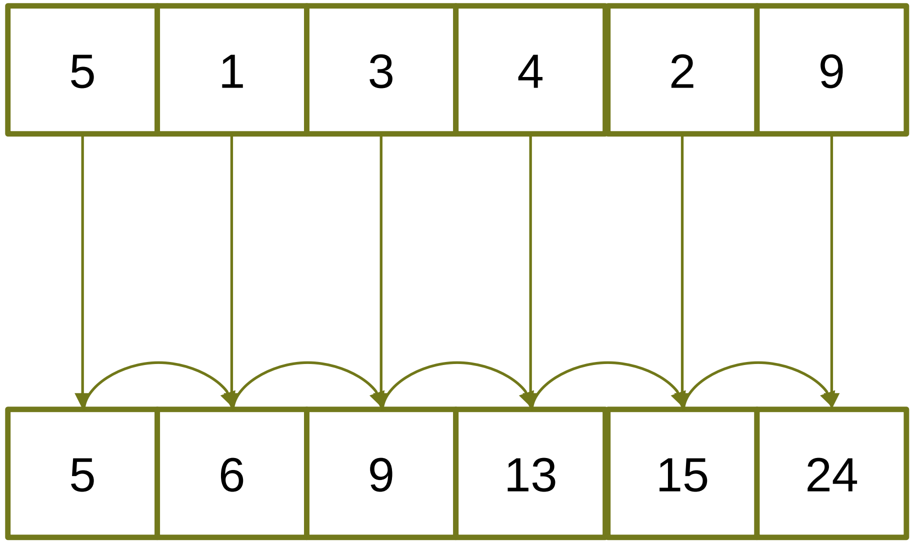
```
int reduce(int n, arr[n]) {
 for(int i=0; i<n; ++i)
  sum += arr[i];
}
```

```
int findmin(int n, a[n]) {
 for(int i=1; i<n; i++)
  if(a[i]<a[0]) a[0] = a[i];
}
```

```
int finditem(list_t list)
 item = list.head;
 while(item.value!=0 && item.next!=NULL)
  item=item.next;
}
```

Insert sort	**NO.** Memory locations and instructions depend on data distribution.
Prefix sum on an array	**YES.**
Dense Cholesky decomposition ($A = LL^T$)	**YES.**
Gradient descent	**NO.** Stopping condition depends on numerical result.
Model checking: exhaustive verification	**NO.** Counterexample position depends on model.

Prefix scan

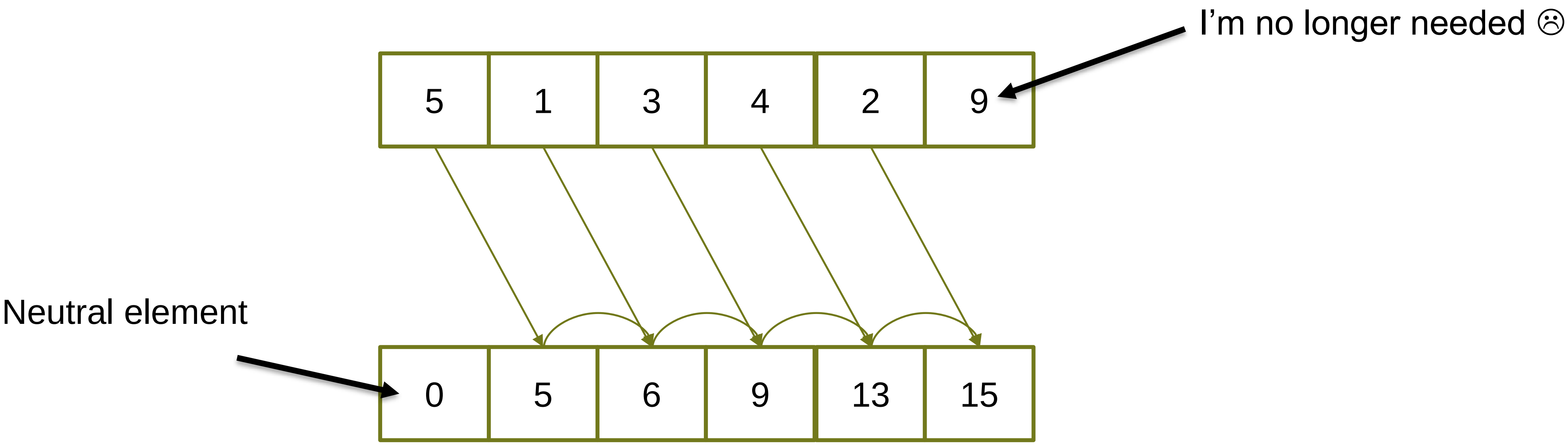


Known as "scan", "prefix scan", "prefix sum"

W(n) = n - 1

D(n) = n -1

Exclusive scan


I'm no longer needed ☹

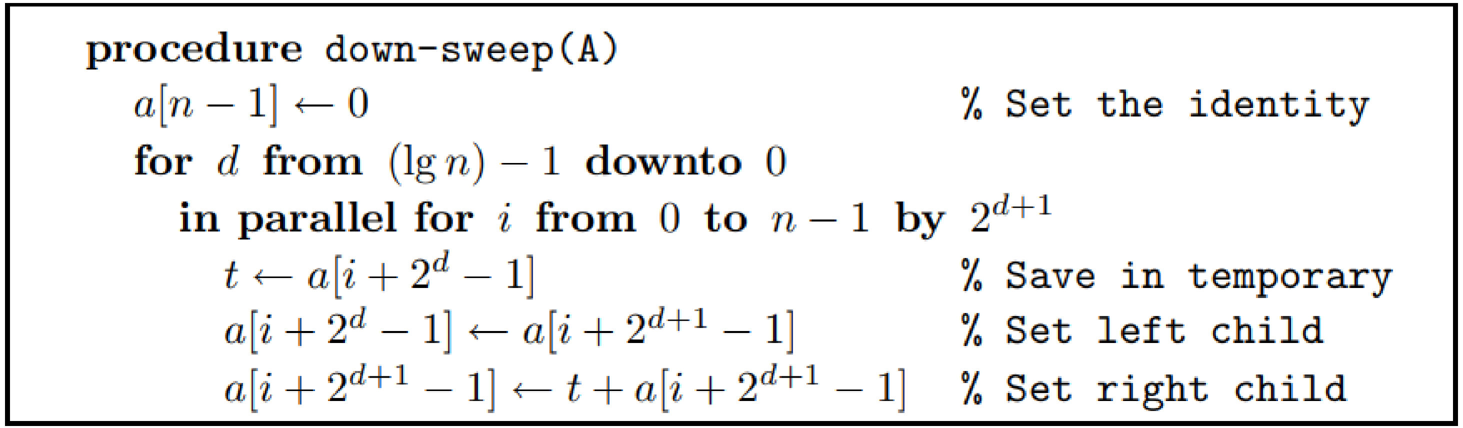
Neutral element

5	1	3	4	2	9
0	5	6	9	13	15

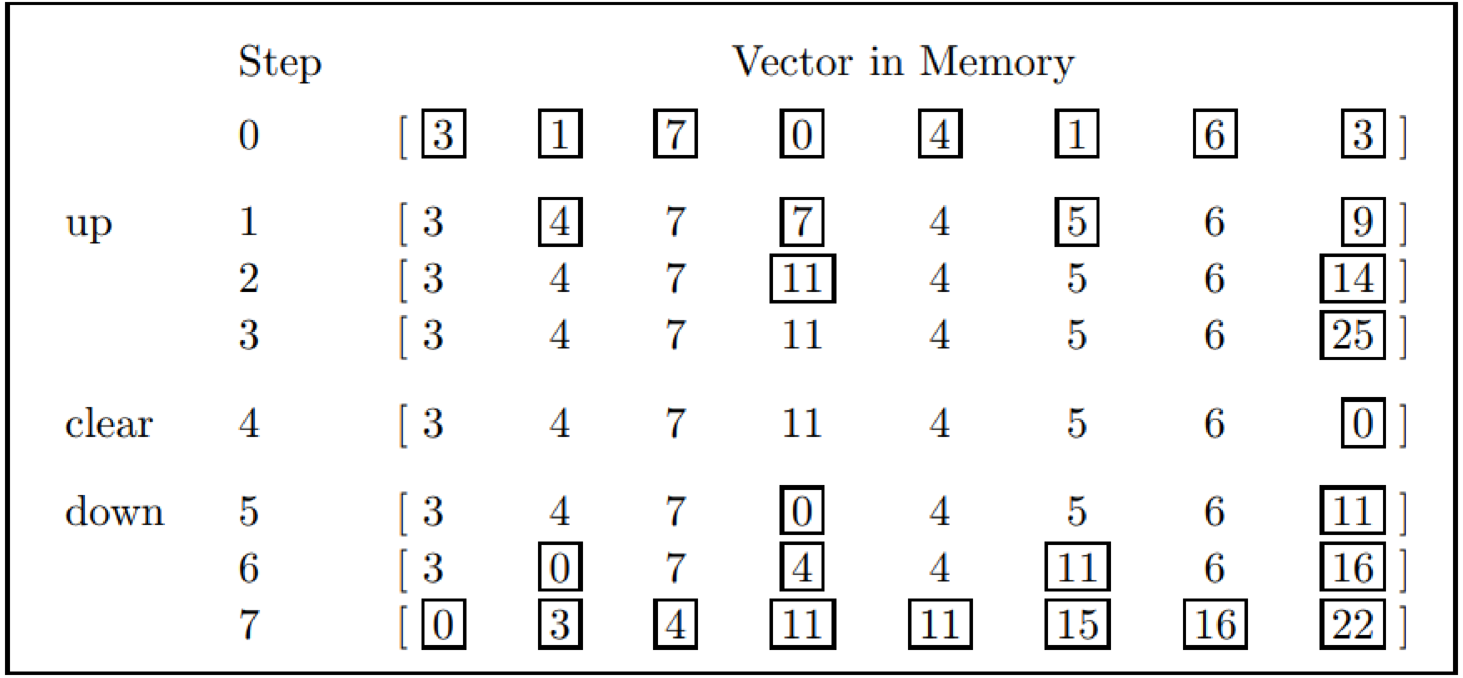
How to make an exclusive scan from an inclusive?

How to make an inclusive scan from an exclusive one?

Blelloch (tree-based) scan



```
procedure down-sweep(A)
  $a[n-1] \leftarrow 0$                                   % Set the identity
  for $d$ from $(\lg n) - 1$ downto $0$
    in parallel for $i$ from $0$ to $n-1$ by $2^{d+1}$
      $t \leftarrow a[i + 2^d - 1]$                       % Save in temporary
      $a[i + 2^d - 1] \leftarrow a[i + 2^{d+1} - 1]$      % Set left child
      $a[i + 2^{d+1} - 1] \leftarrow t + a[i + 2^{d+1} - 1]$  % Set right child
```

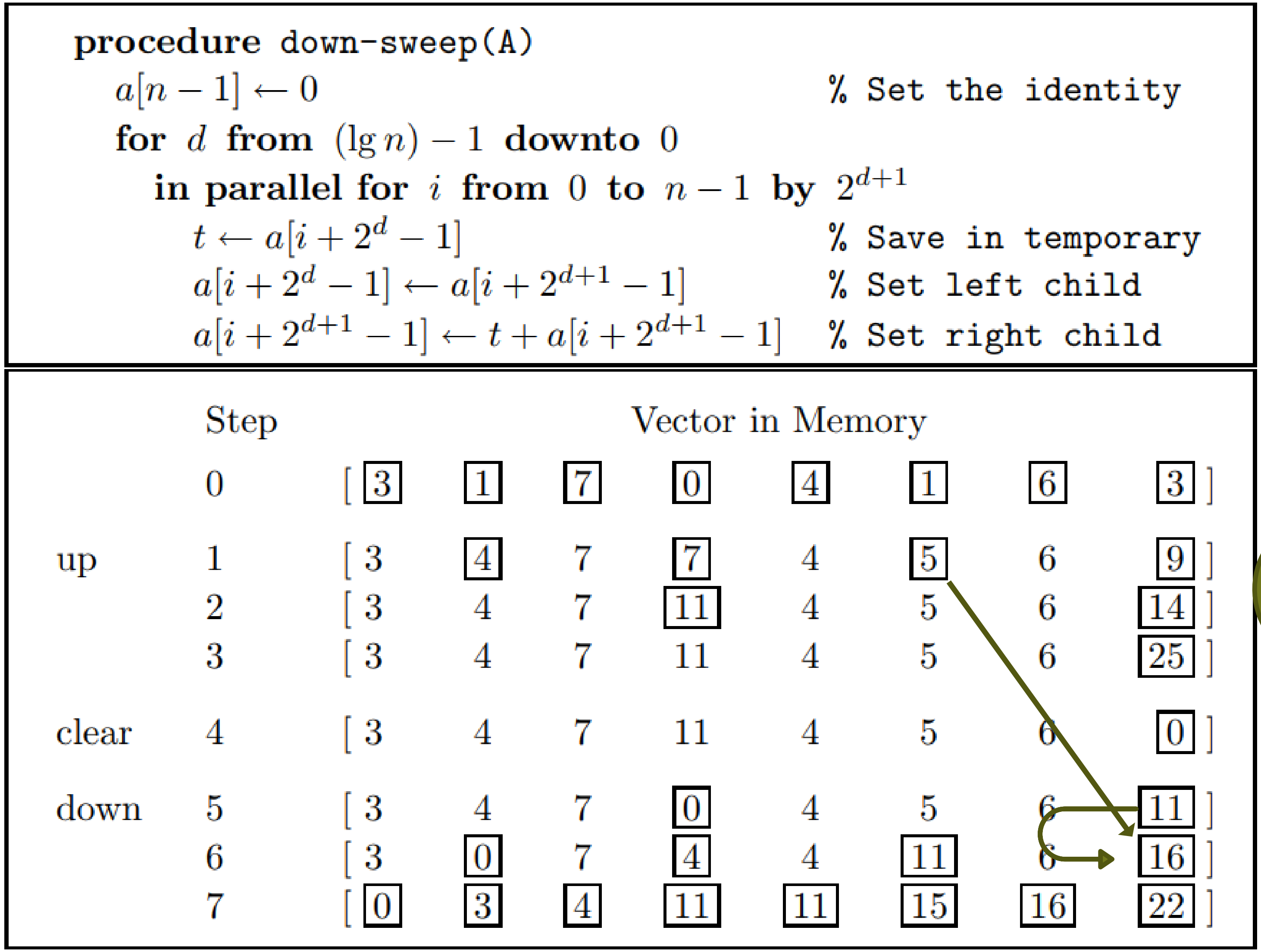


	Step	Vector in Memory							
	0	[$\boxed{3}$	$\boxed{1}$	$\boxed{7}$	$\boxed{0}$	$\boxed{4}$	$\boxed{1}$	$\boxed{6}$	$\boxed{3}$]
up	1	[3	$\boxed{4}$	7	$\boxed{7}$	4	$\boxed{5}$	6	$\boxed{9}$]
	2	[3	4	7	$\boxed{11}$	4	5	6	$\boxed{14}$]
	3	[3	4	7	11	4	5	6	$\boxed{25}$]
clear	4	[3	4	7	11	4	5	6	$\boxed{0}$]
down	5	[3	4	7	$\boxed{0}$	4	5	6	$\boxed{11}$]
	6	[3	$\boxed{0}$	7	$\boxed{4}$	4	$\boxed{11}$	6	$\boxed{16}$]
	7	[$\boxed{0}$	$\boxed{3}$	$\boxed{4}$	$\boxed{11}$	$\boxed{11}$	$\boxed{15}$	$\boxed{16}$	$\boxed{22}$]

G. Blelloch, “Prefix Sums and Their Applications”

(b) Executing a +-prescan on a PRAM.

Blelloch (tree-based) scan



```
procedure down-sweep(A)
  $a[n-1] \leftarrow 0$                                  % Set the identity
  for $d$ from $(\lg n) - 1$ downto $0$
    in parallel for $i$ from $0$ to $n-1$ by $2^{d+1}$
      $t \leftarrow a[i + 2^d - 1]$                      % Save in temporary
      $a[i + 2^d - 1] \leftarrow a[i + 2^{d+1} - 1]$     % Set left child
      $a[i + 2^{d+1} - 1] \leftarrow t + a[i + 2^{d+1} - 1]$   % Set right child
```

	Step		Vector in Memory							
	0	[	$\boxed{3}$	$\boxed{1}$	$\boxed{7}$	$\boxed{0}$	$\boxed{4}$	$\boxed{1}$	$\boxed{6}$	$\boxed{3}$]
up	1	[	3	$\boxed{4}$	7	$\boxed{7}$	4	$\boxed{5}$	6	$\boxed{9}$]
	2	[	3	4	7	$\boxed{11}$	4	5	6	$\boxed{14}$]
	3	[	3	4	7	11	4	5	6	$\boxed{25}$]
clear	4	[	3	4	7	11	4	5	6	$\boxed{0}$]
down	5	[	3	4	7	$\boxed{0}$	4	5	6	$\boxed{11}$]
	6	[	3	$\boxed{0}$	7	$\boxed{4}$	4	$\boxed{11}$	6	$\boxed{16}$]
	7	[	$\boxed{0}$	$\boxed{3}$	$\boxed{4}$	$\boxed{11}$	$\boxed{11}$	$\boxed{15}$	$\boxed{16}$	$\boxed{22}$]

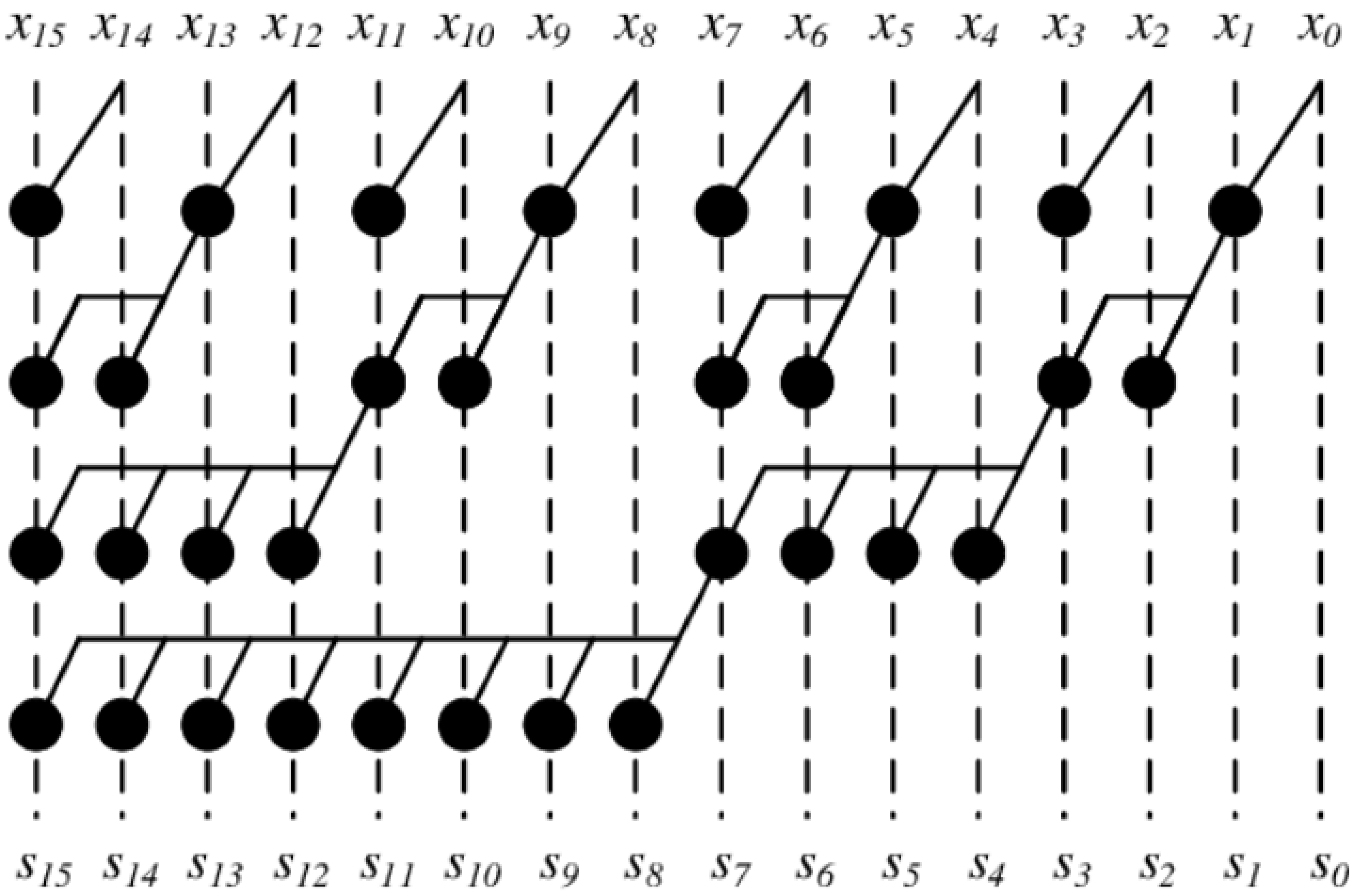
Downsweep uses
a) aggregated result b) last left child result

G. Blelloch, “Prefix Sums and Their Applications”

(b) Executing a +-prescan on a PRAM.

Ladner-Fischer scan

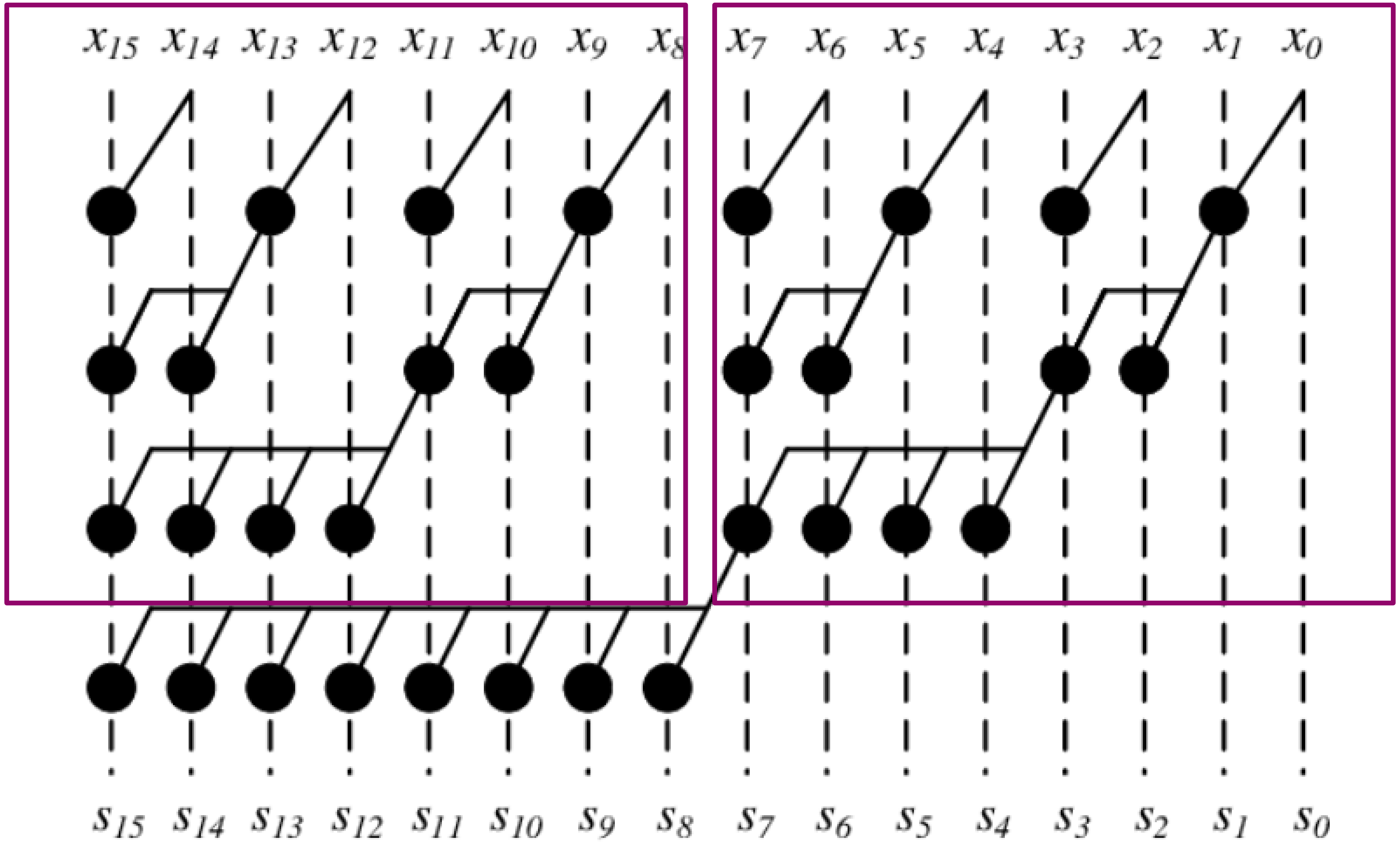
Efficient work for any depth **$D(n) = \log(n) + k$**



N. Abbas et al., “Accelerating HMMER on FPGA using parallel prefixes and reductions”

Ladner-Fischer scan

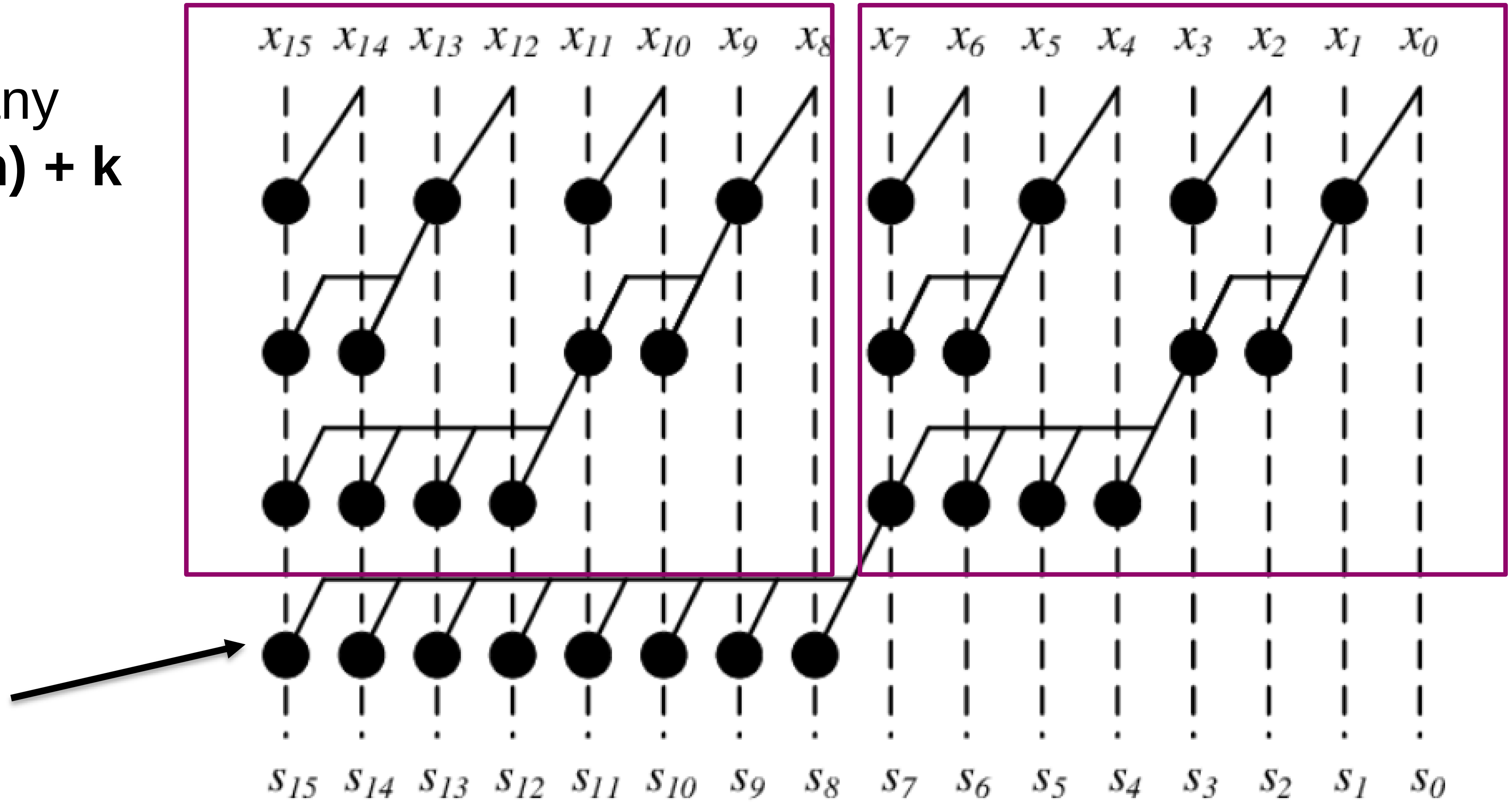
Efficient work for any depth **D(n) = log(n) + k**



N. Abbas et al., “Accelerating HMMER on FPGA using parallel prefixes and reductions”

Ladner-Fischer scan

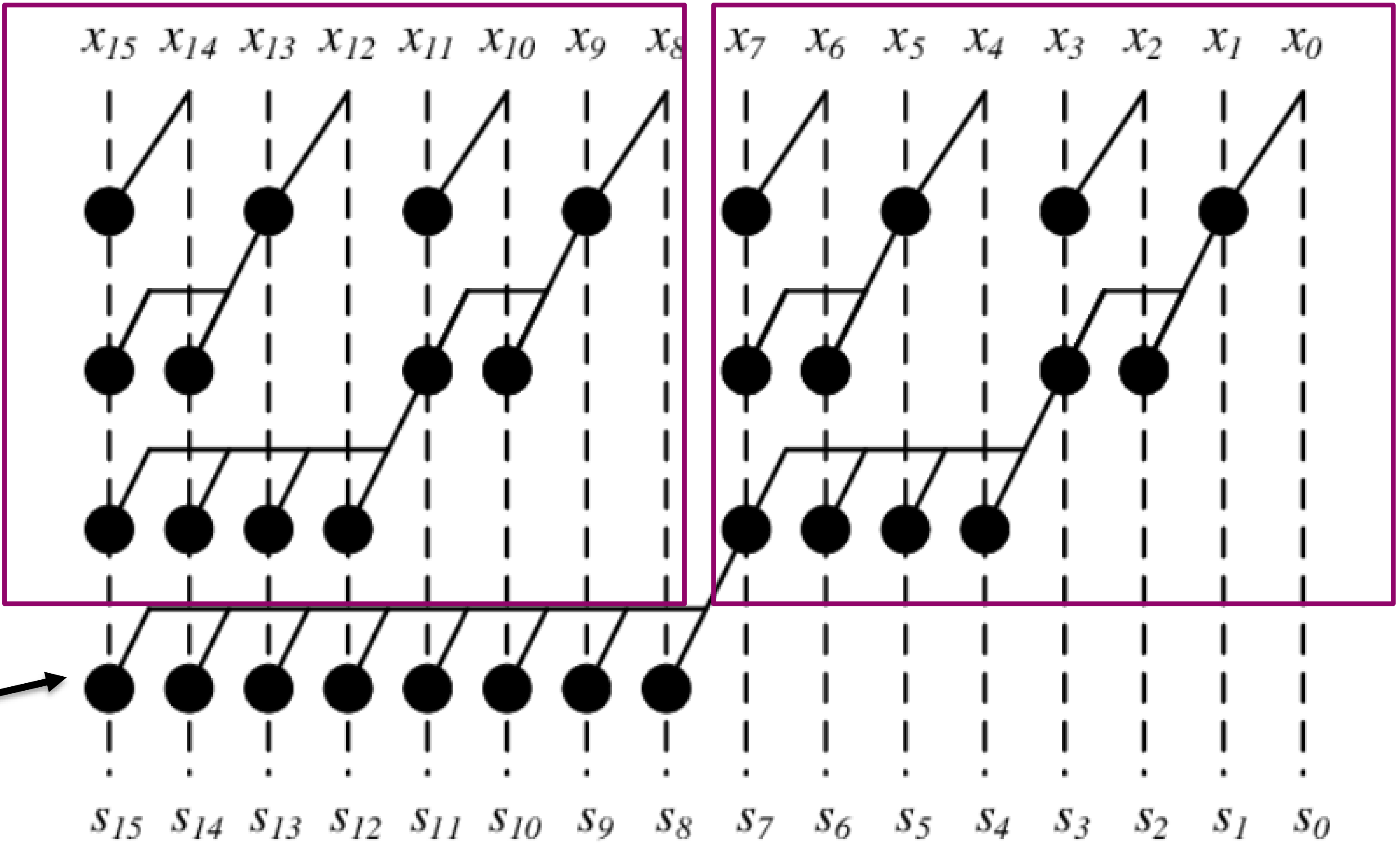
Efficient work for any depth **D(n) = log(n) + k**



N. Abbas et al., “Accelerating HMMER on FPGA using parallel prefixes and reductions”

Ladner-Fischer scan

Efficient work for any depth **D(n) = log(n) + k**



Broadcast

N. Abbas et al., “Accelerating HMMER on FPGA using parallel prefixes and reductions”

Parallel scan

What if p << n ?

1. **Buy more processors? log(n) runtime ☹**

Parallel scan

What if p << n ?

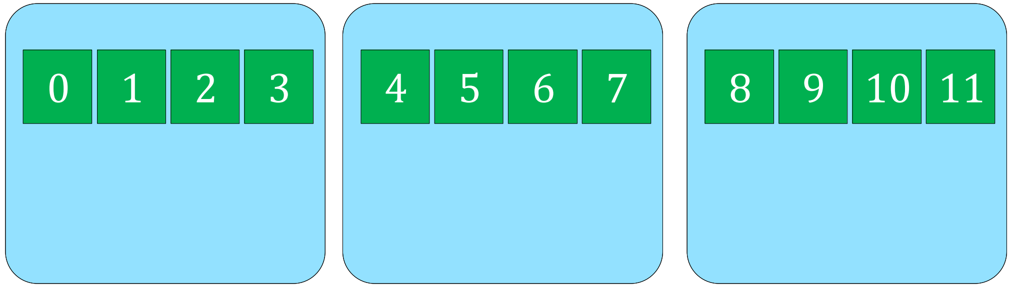
1. Buy more processors? log(n) runtime ☹
2. Run multiple iterations?

Parallel scan

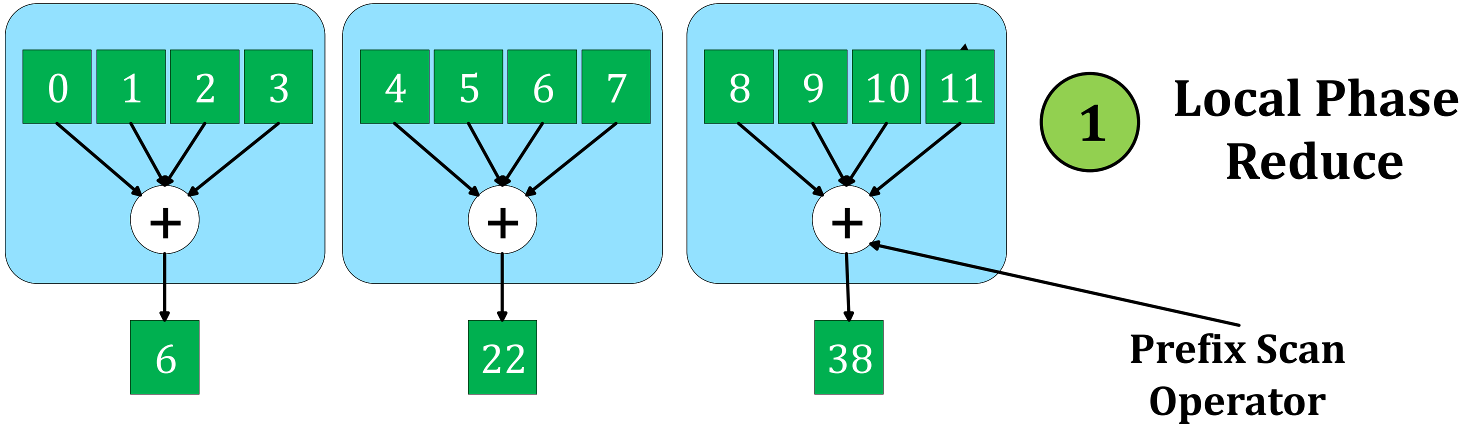
What if p << n ?

1. Buy more processors? log(n) runtime ☹
2. Run multiple iterations?
3. Split work and minimize global exchange to log(p) ☺

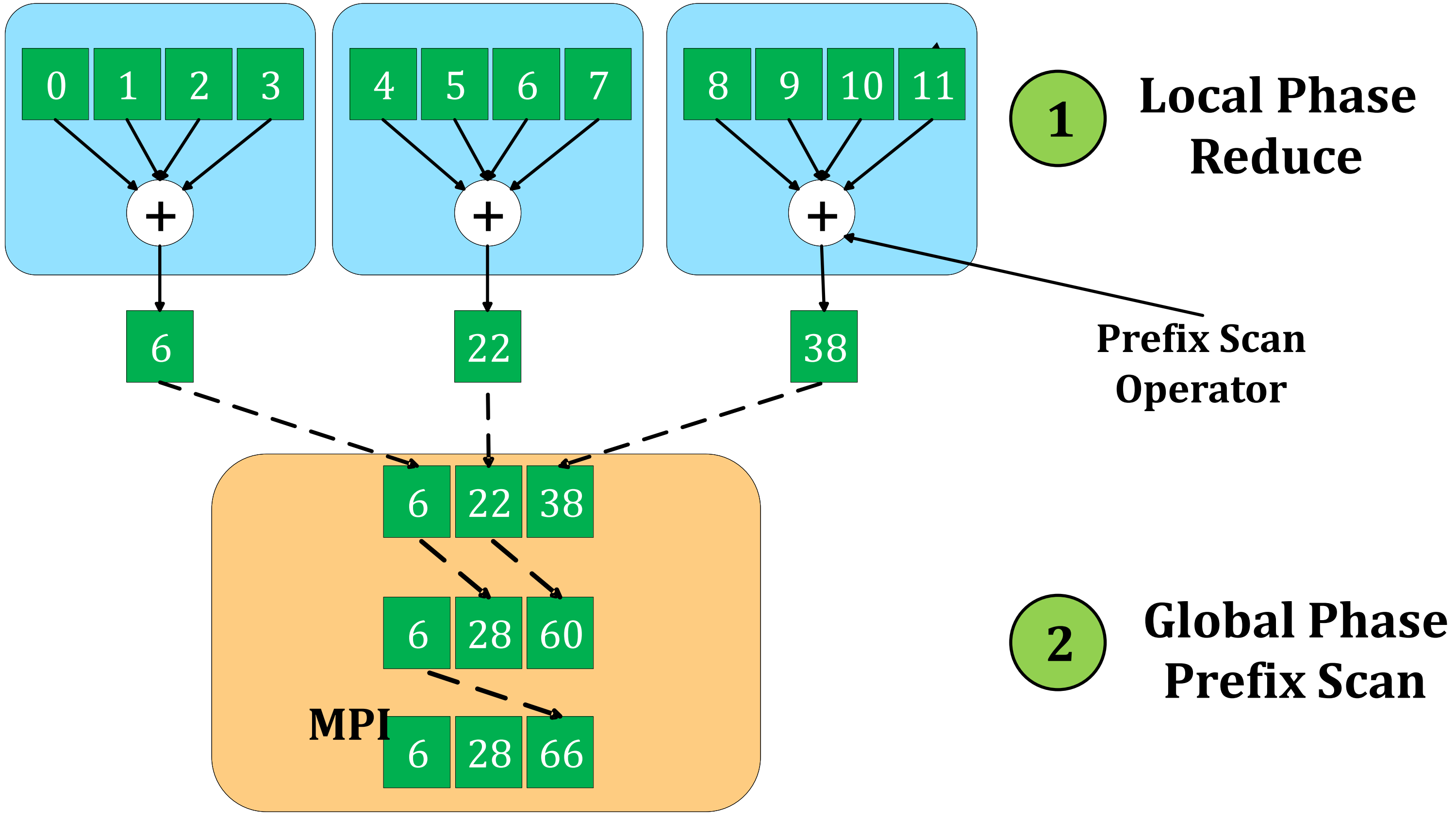
Parallel scan



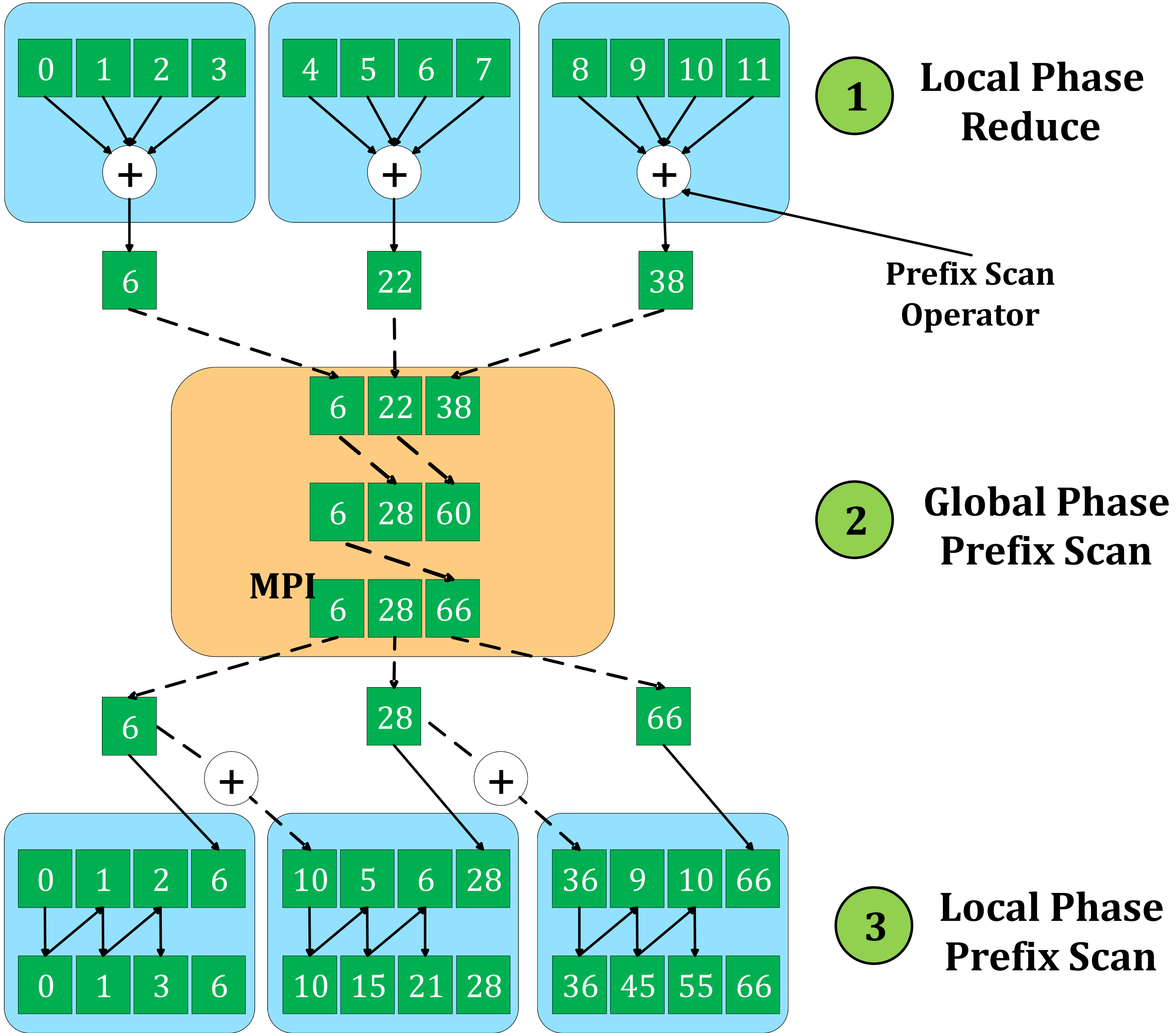
Parallel scan



Parallel scan



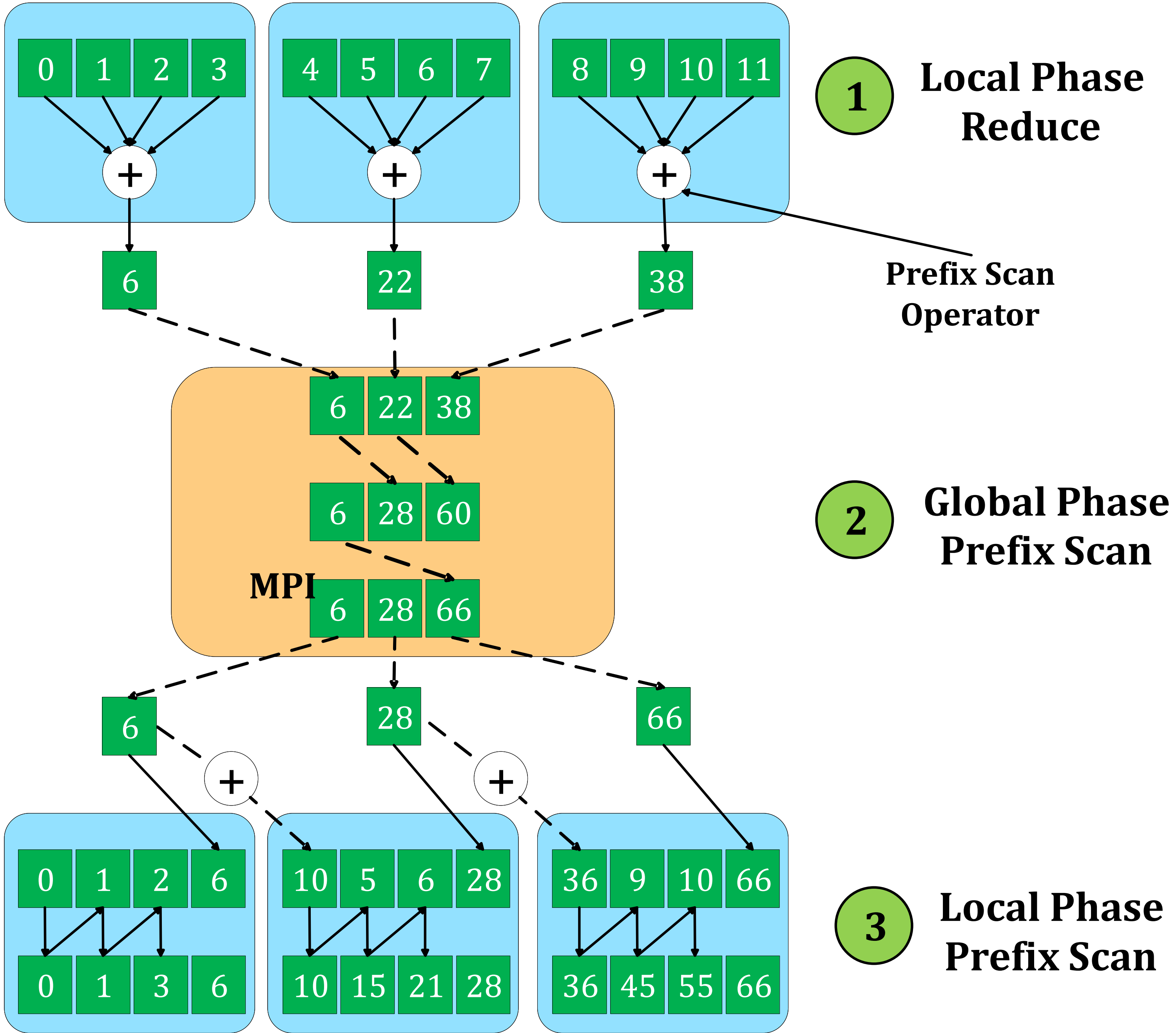
Parallel scan



Parallel scan

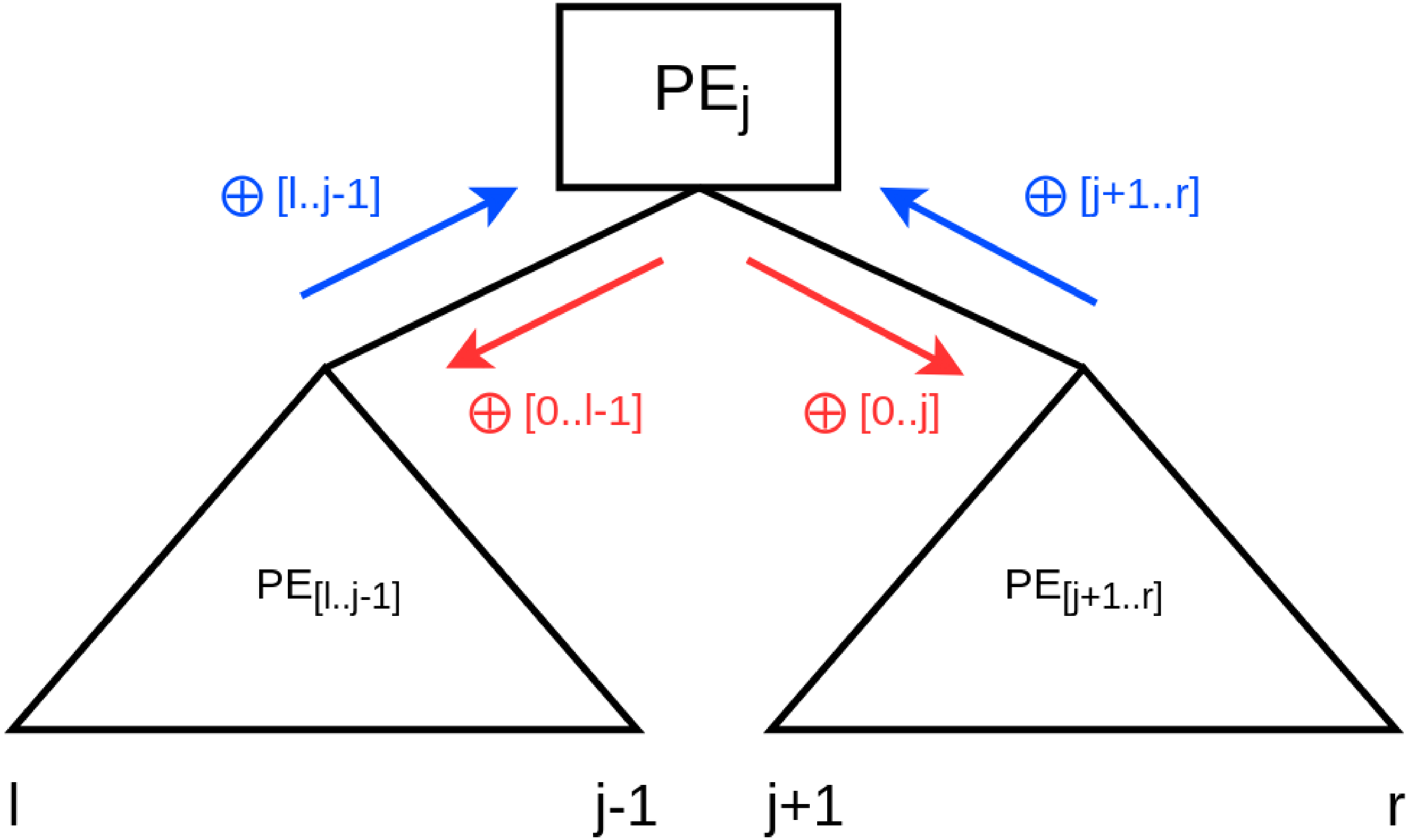
Scan—then—reduce
also possible!

Which one is better?



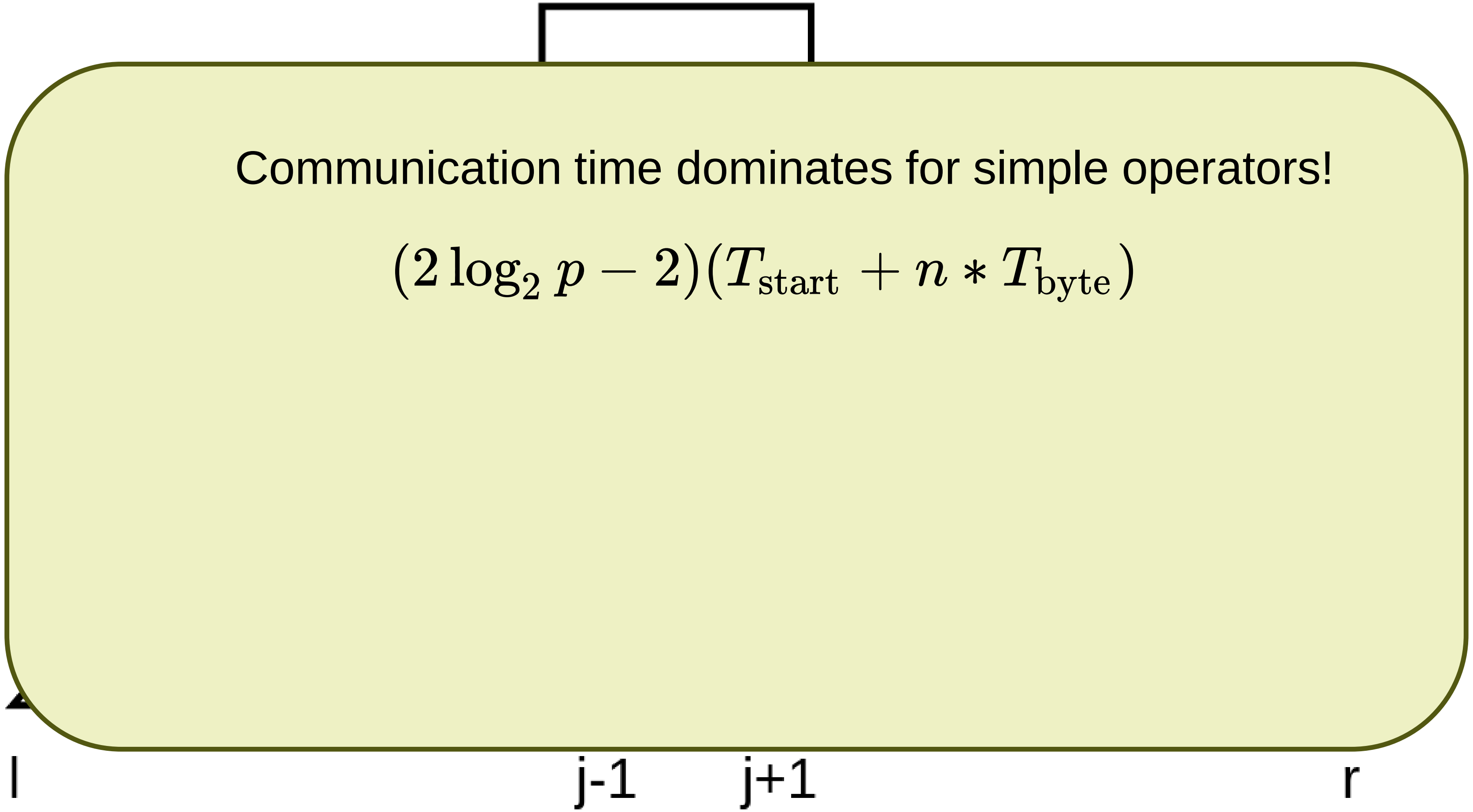
Scan in message passing

Pipelined binary tree – split message of size **n** into **k** packets.



Scan in message passing

Pipelined binary tree – split message of size **n** into **k** packets.

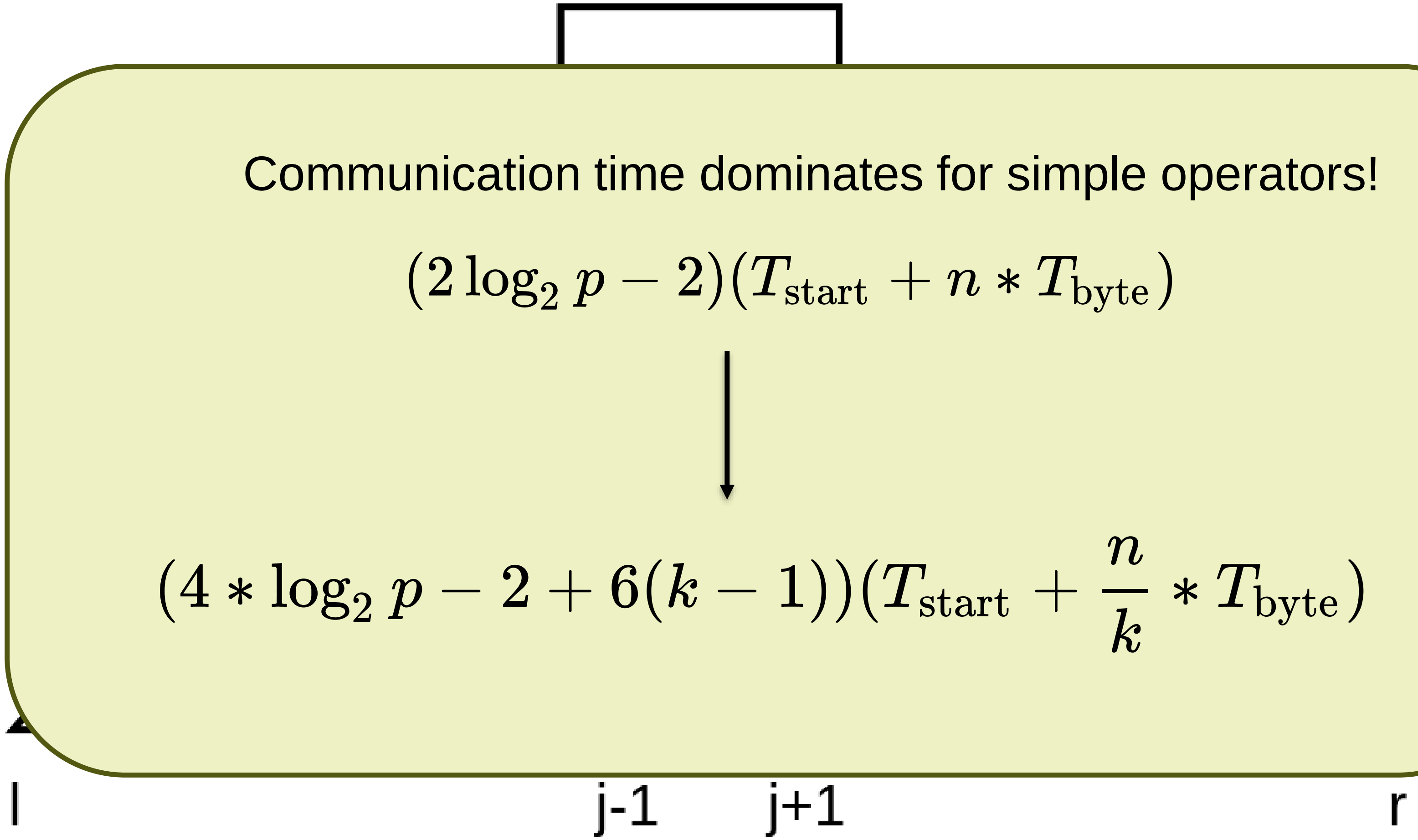


Communication time dominates for simple operators!

$$(2\log_2 p - 2)(T_{\text{start}} + n * T_{\text{byte}})$$

Scan in message passing

Pipelined binary tree – split message of size **n** into **k** packets.



Communication time dominates for simple operators!

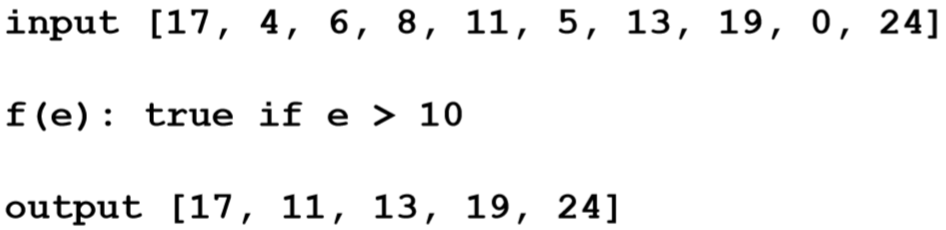
$$(2\log_2 p - 2)(T_{\text{start}} + n * T_{\text{byte}})$$

$$\downarrow$$

$$(4 * \log_2 p - 2 + 6(k-1))(T_{\text{start}} + \frac{n}{k} * T_{\text{byte}})$$

Parallel filter

Given an array, produce an array containing only elements for which predicate is true.



```
input [17, 4, 6, 8, 11, 5, 13, 19, 0, 24]

f(e): true if e > 10

output [17, 11, 13, 19, 24]
```

Can we parallelize that?

Parallel filter

```
input [17, 4, 6, 8, 11, 5, 13, 19, 0, 24]
```

- **Parallel map with filter f**

D(n) = 1

```
bits   [1,  0, 0, 0,  1, 0,  1,  1, 0,  1]
```

- **Parallel prefix sum on bit vector**

D(n) = log(n)

```
bitsum [1,  1, 1, 1,  2, 2,  3,  4, 4,  5]
```

- **Parallel map with filter**

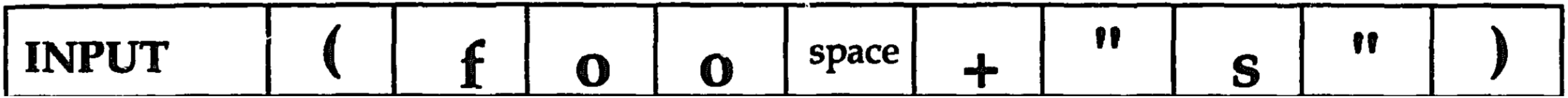
```
output [17, 11, 13, 19, 24]
```

D(n) = 1

```
def f(i):
  if bitsum[i]:
    output[bitsum[i -1]] = input[i]
```

wiki.rice.edu/confluence/download/attachments/4435861/comp322-s18-lec37-slides-v1.pdf

Parallel lexical analysis



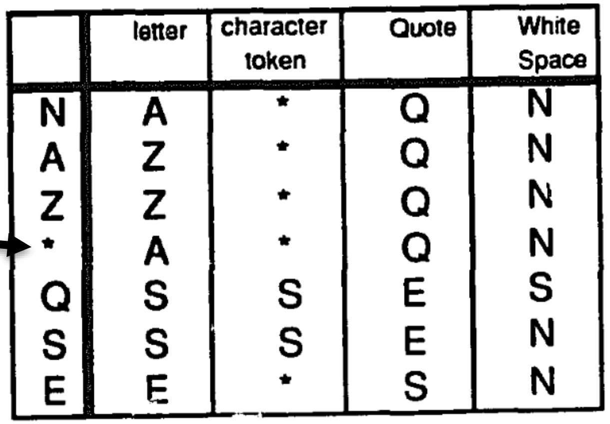
INPUT	(	f	o	o	space	+	"	s	"	)

Whitespace

Identifier

Character token

Quote strings

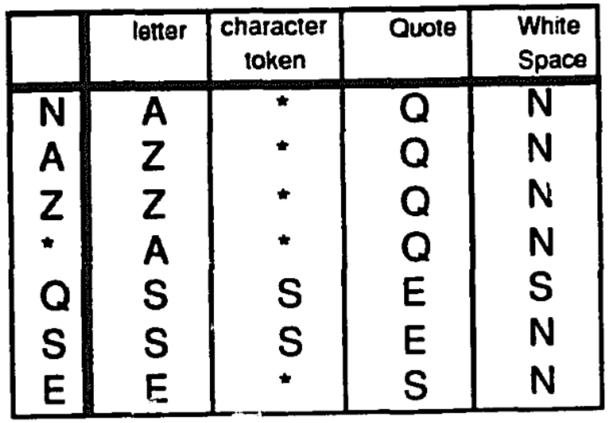


	letter	character token	Quote	White Space
N	A	*	Q	N
A	Z	*	Q	N
Z	Z	*	Q	N
*	A	*	Q	N
Q	S	S	E	S
S	S	S	E	N
E	E	*	S	N

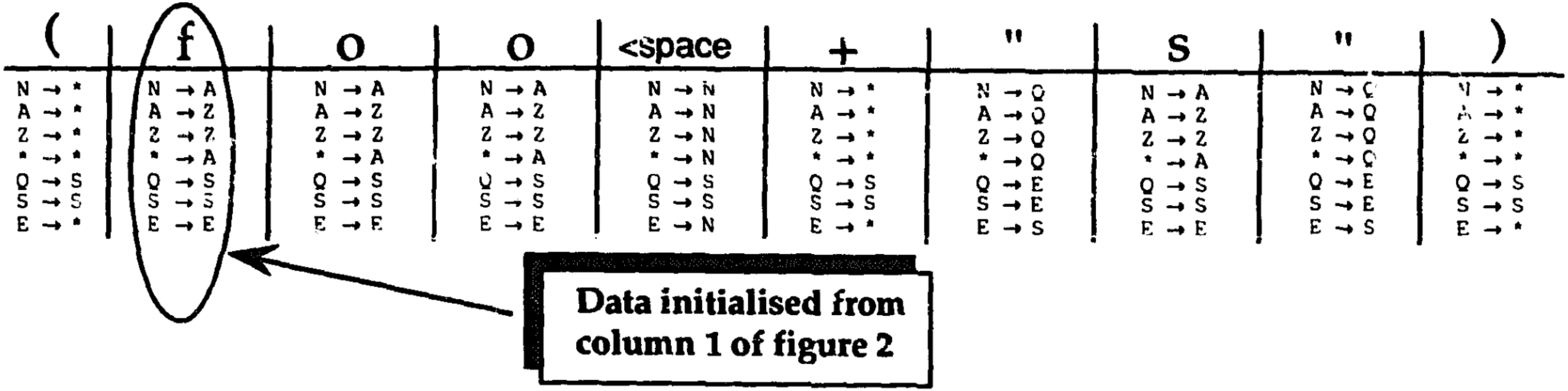
Finite-State Machine

Parallel lexical analysis

Map appropriate transition for each character



	letter	character token	Quote	White Space
N	A	*	Q	N
A	Z	*	Q	N
Z	Z	*	Q	N
*	A	*	Q	N
Q	S	S	E	S
S	S	S	E	N
E	E	*	S	N



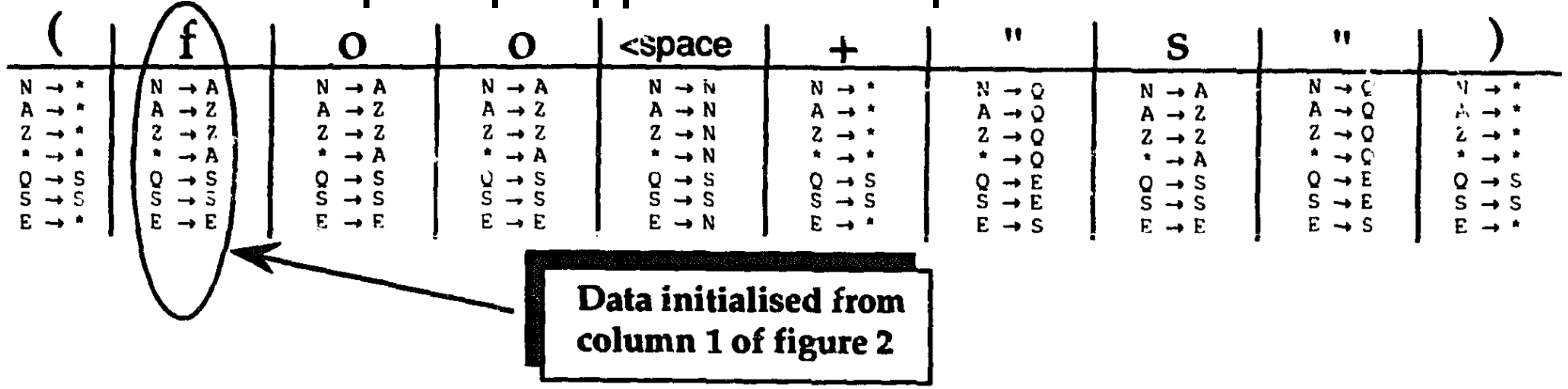
(	f	o	o	<space	+	"	s	"	)
N → *	N → A	N → A	N → A	N → N	N → *	N → Q	N → A	N → Q	N → *
A → *	A → Z	A → Z	A → Z	A → N	A → *	A → Q	A → Z	A → Q	A → *
Z → *	Z → Z	Z → Z	Z → Z	Z → N	Z → *	Z → Q	Z → Z	Z → Q	Z → *
* → *	* → A	* → A	* → A	* → N	* → *	* → Q	* → A	* → Q	* → *
Q → S	Q → S	Q → S	Q → S	Q → S	Q → S	Q → E	Q → S	Q → E	Q → S
S → S	S → S	S → S	S → S	S → S	S → S	S → E	S → S	S → E	S → S
E → *	E → E	E → E	E → E	E → N	E → *	E → S	E → E	E → S	E → *

Data initialised from column 1 of figure 2

Fig. 3. The state of the processors after the input ‘(foo + “s”)’ has been initialised.

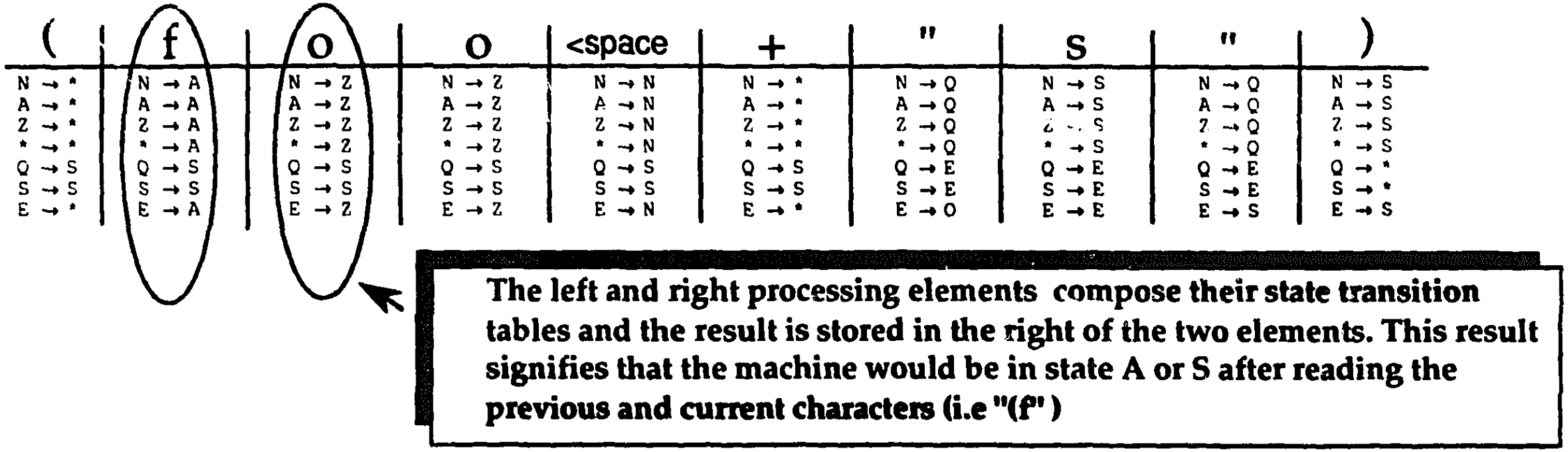
Parallel lexical analysis

Compose transitions: α → β and β → γ produces α → γ



(	f	o	o	<space	+	"	S	"	)
N → *	N → A	N → A	N → A	N → N	N → *	N → Q	N → A	N → Q	N → *
A → *	A → Z	A → Z	A → Z	A → N	A → *	A → Q	A → Z	A → Q	A → *
Z → *	Z → Z	Z → Z	Z → Z	Z → N	Z → *	Z → Q	Z → Z	Z → Q	Z → *
* → *	* → A	* → A	* → A	* → N	* → *	* → Q	* → A	* → Q	* → *
Q → S	Q → S	Q → S	Q → S	Q → S	Q → S	Q → E	Q → S	Q → E	Q → S
S → S	S → S	S → S	S → S	S → S	S → S	S → E	S → S	S → E	S → S
E → *	E → E	E → E	E → E	E → N	E → *	E → S	E → E	E → S	E → *

Data initialised from column 1 of figure 2



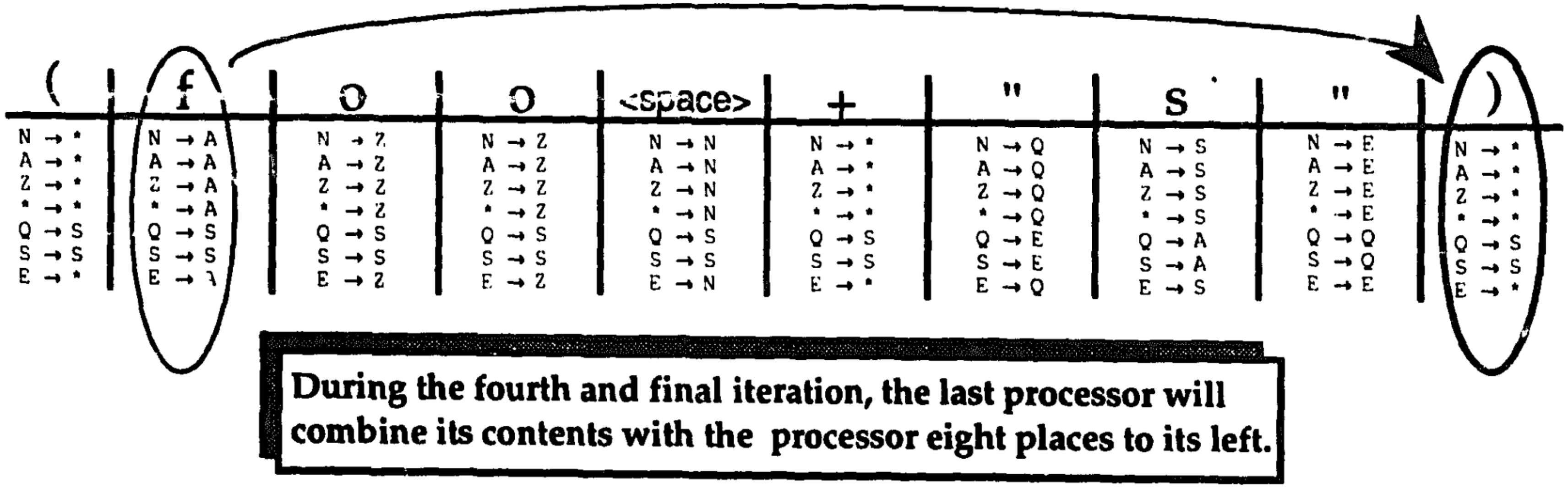
(	f	o	o	<space	+	"	S	"	)
N → *	N → A	N → Z	N → Z	N → N	N → *	N → Q	N → S	N → Q	N → S
A → *	A → A	A → Z	A → Z	A → N	A → *	A → Q	A → S	A → Q	A → S
Z → *	Z → A	Z → Z	Z → Z	Z → N	Z → *	Z → Q	Z → S	Z → Q	Z → S
* → *	* → A	* → Z	* → Z	* → N	* → *	* → Q	* → S	* → Q	* → S
Q → S	Q → S	Q → S	Q → S	Q → S	Q → S	Q → E	Q → E	Q → E	Q → *
S → S	S → S	S → S	S → S	S → S	S → S	S → E	S → E	S → E	S → *
E → *	E → A	E → Z	E → Z	E → N	E → *	E → O	E → E	E → S	E → S

The left and right processing elements compose their state transition tables and the result is stored in the right of the two elements. This result signifies that the machine would be in state A or S after reading the previous and current characters (i.e "(f")

Parallel lexical analysis

Parallel prefix sum!

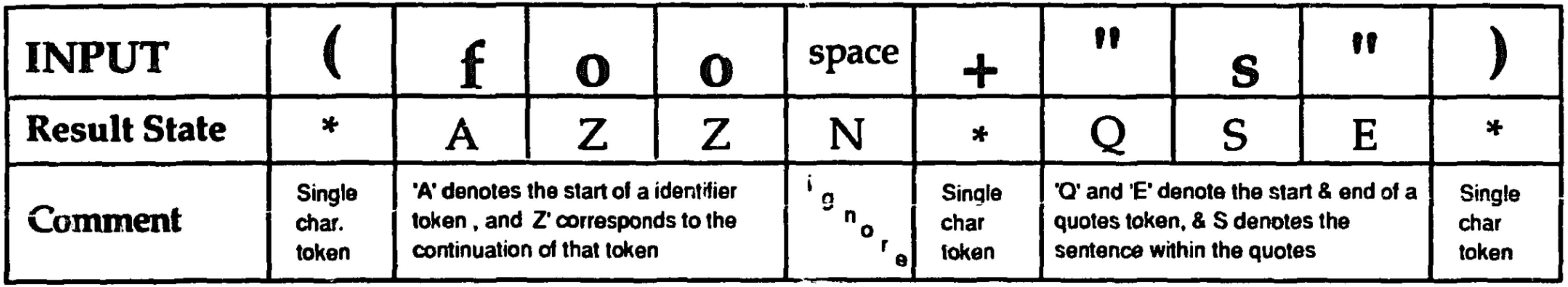
Last value is FSM state after reading the entire input



(	f	o	o	<space>	+	"	S	"	)
N → *	N → A	N → Z	N → Z	N → N	N → *	N → Q	N → S	N → E	N → *
A → *	A → A	A → Z	A → Z	A → N	A → *	A → Q	A → S	A → E	A → *
Z → *	Z → A	Z → Z	Z → Z	Z → N	Z → *	Z → Q	Z → S	Z → E	Z → *
* → *	* → A	* → Z	* → Z	* → N	* → *	* → Q	* → S	* → E	* → *
Q → S	Q → S	Q → S	Q → S	Q → S	Q → S	Q → E	Q → A	Q → Q	Q → S
S → S	S → S	S → S	S → S	S → S	S → S	S → E	S → A	S → Q	S → S
E → *	E → A	E → Z	E → Z	E → N	E → *	E → Q	E → S	E → E	E → *

During the fourth and final iteration, the last processor will combine its contents with the processor eight places to its left.

Fig. 5. The fourth and final iteration of the parallel prefix technique.



INPUT	(	f	o	o	space	+	"	S	"	)
Result State	*	A	Z	Z	N	*	Q	S	E	*
Comment	Single char. token	'A' denotes the start of a identifier token , and Z' corresponds to the continuation of that token			ignore	Single char token	'Q' and 'E' denote the start & end of a quotes token, & S denotes the sentence within the quotes			Single char token

"Parallel lexical analysis and parsing on the AMT distributed array processor." Parallel computing 18.6 (1992)

Radix sort

- **Idea: split keys into multiple digits according to the radix**

101	111	010	011	110	001

Radix sort

- **Idea: split keys into multiple digits according to the radix**

101	111	010	011	110	001

- **In each iteration sort according to a single digit.**

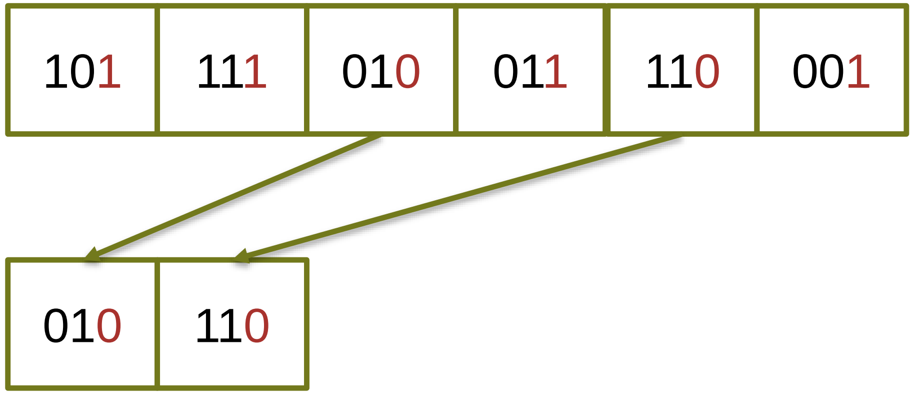
101	111	010	011	110	001

Radix sort

- **Idea: split keys into multiple digits according to the radix**

101	111	010	011	110	001

- **In each iteration sort according to a single digit.**



101	111	010	011	110	001

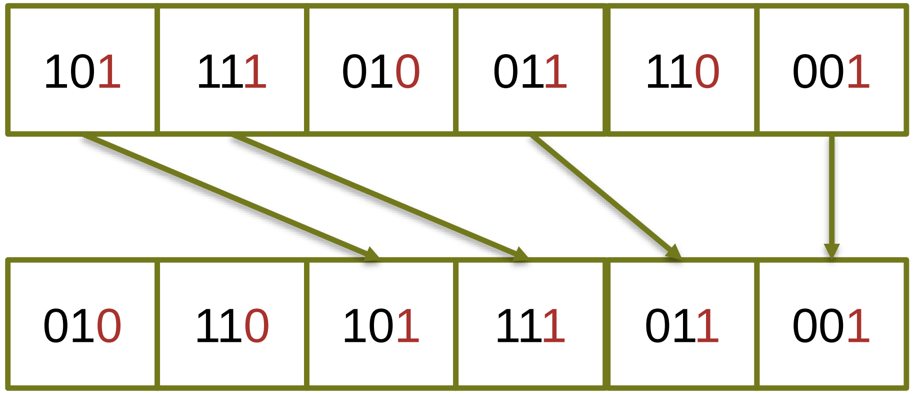
010	110

Radix sort

- **Idea: split keys into multiple digits according to the radix**

101	111	010	011	110	001

- **In each iteration sort according to a single digit.**



101	111	010	011	110	001

010	110	101	111	011	001

Radix sort

- **Idea: split keys into multiple digits according to the radix**

101	111	010	011	110	001

- **In each iteration sort according to a single digit.**

101	111	010	011	110	001

010	110	101	111	011	001

- **Stability (preserver the order inside group) + process each digit => full sort!**

101	001	010	110	111	011

001	010	011	101	110	111

Radix sort

- **Idea: split keys into multiple digits according to the radix**

101	111	010	011	110	001

- **In each iteration sort according to a single digit.**

101	111	010	011	110	001

010	110	101	111	011	001	**Iter 0**

- **Stability (preserver the order inside group) + process each digit => full sort!**

101	001	010	110	111	011	**Iter 1**
001	010	011	101	110	111	**Iter 2**

k digits
k iterations

Radix sort

101	111	010	011	110	001
1	1	0	1	0	1

Radix sort

101	111	010	011	110	001

1	1	0	1	0	1

1. **Build k-element histogram of each digit.**

Digit	0	1
Count	2	4

k digits
k elements

Radix sort

101	111	010	011	110	001
1	1	0	1	0	1

1. **Build k-element histogram of each digit.**

Digit	0	1
Count	2	4

k digits
k elements

2. **Exclusive prefix scan on histogram**

0	1
0	2

Radix sort

101	111	010	011	110	001

1	1	0	1	0	1

1. Build k-element histogram of each digit.

Digit	0	1
Count	2	4

k digits
k elements

2. Exclusive prefix scan on histogram

0	1
0	2

3. Exclusive prefix scan on each group of digits

0	1	0	2	1	3

Radix sort

101	111	010	011	110	001

1	1	0	1	0	1

1. Build k-element histogram of each digit.

Digit	0	1
Count	2	4

k digits
k elements

2. Exclusive prefix scan on histogram

0	1
0	2

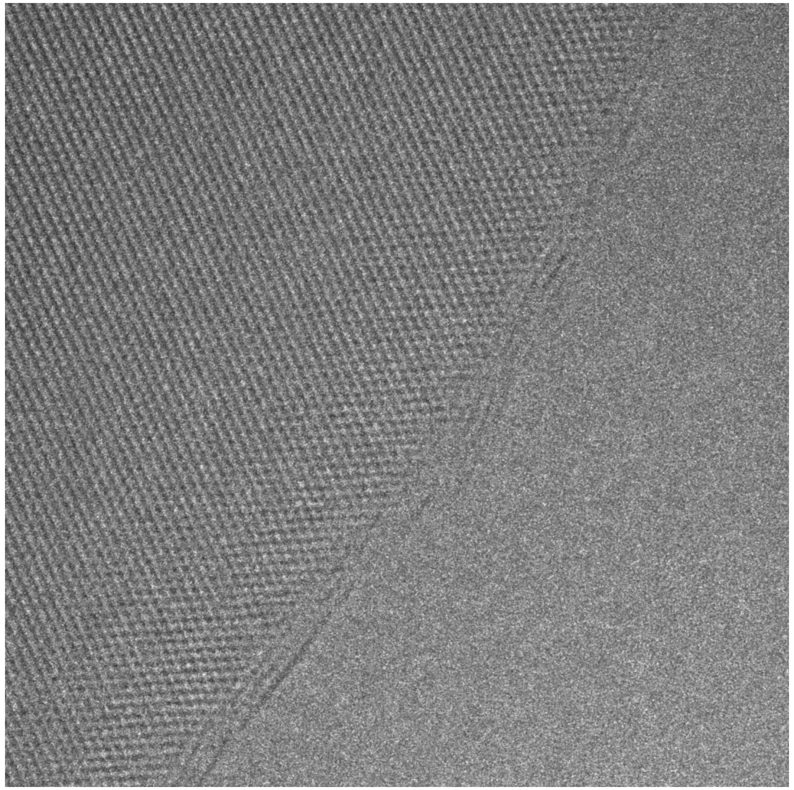
3. Exclusive prefix scan on each group of digits

0	1	0	2	1	3

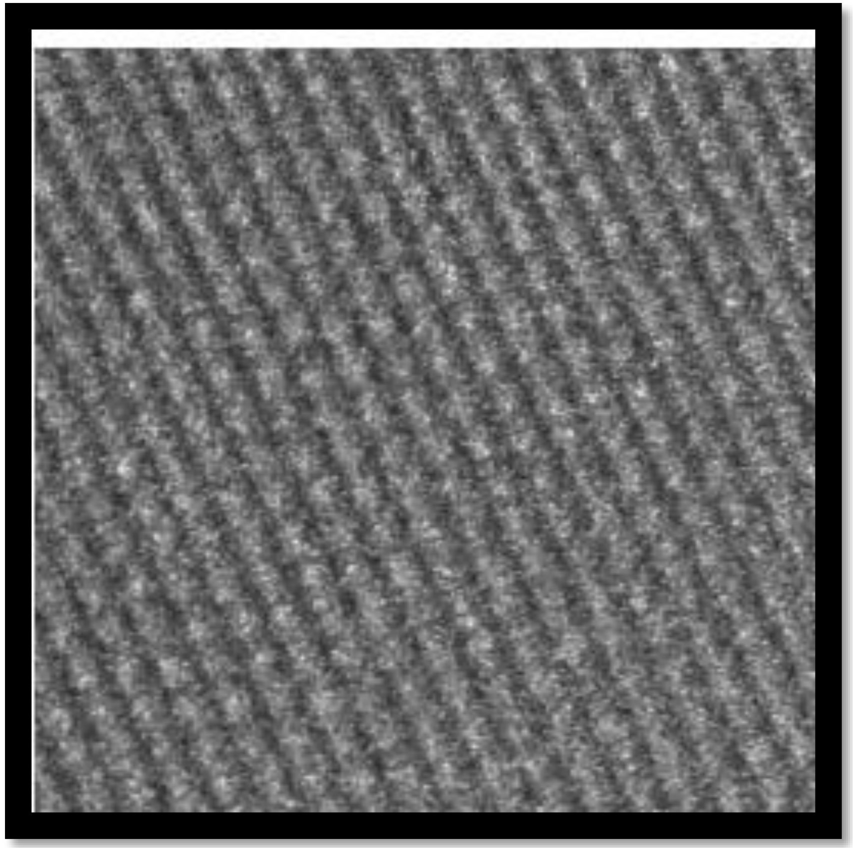
4. Summarize to get new indices

0+2	1+2	0+0	2+2	1+0	3+2

Image Registration



Electron microscopy frame
f_0



The image after deformation (magnification).
$f_{25} \cdot \phi_{0,25}$

Image Registration

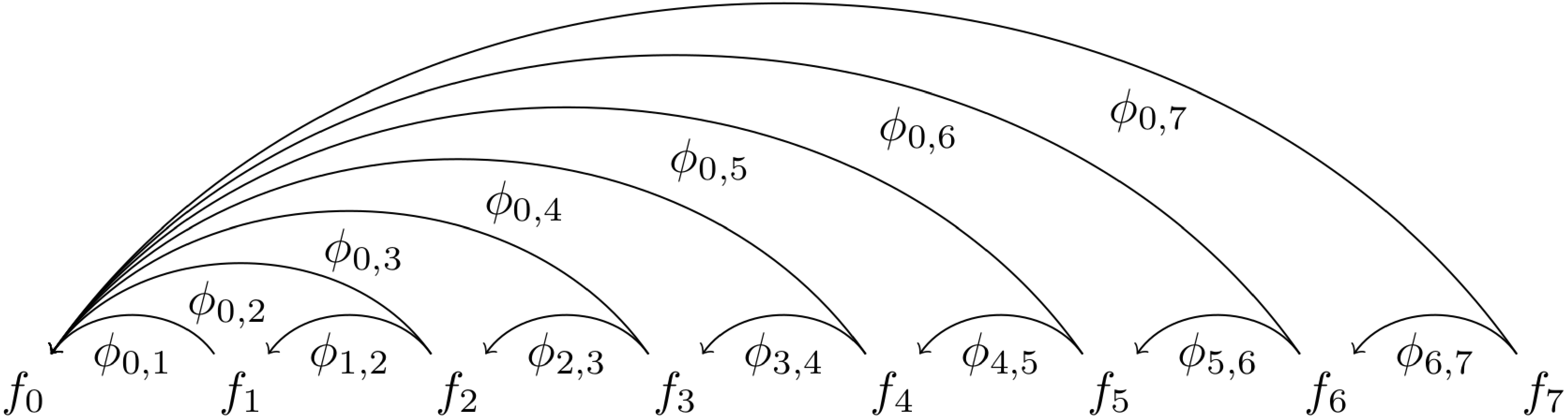
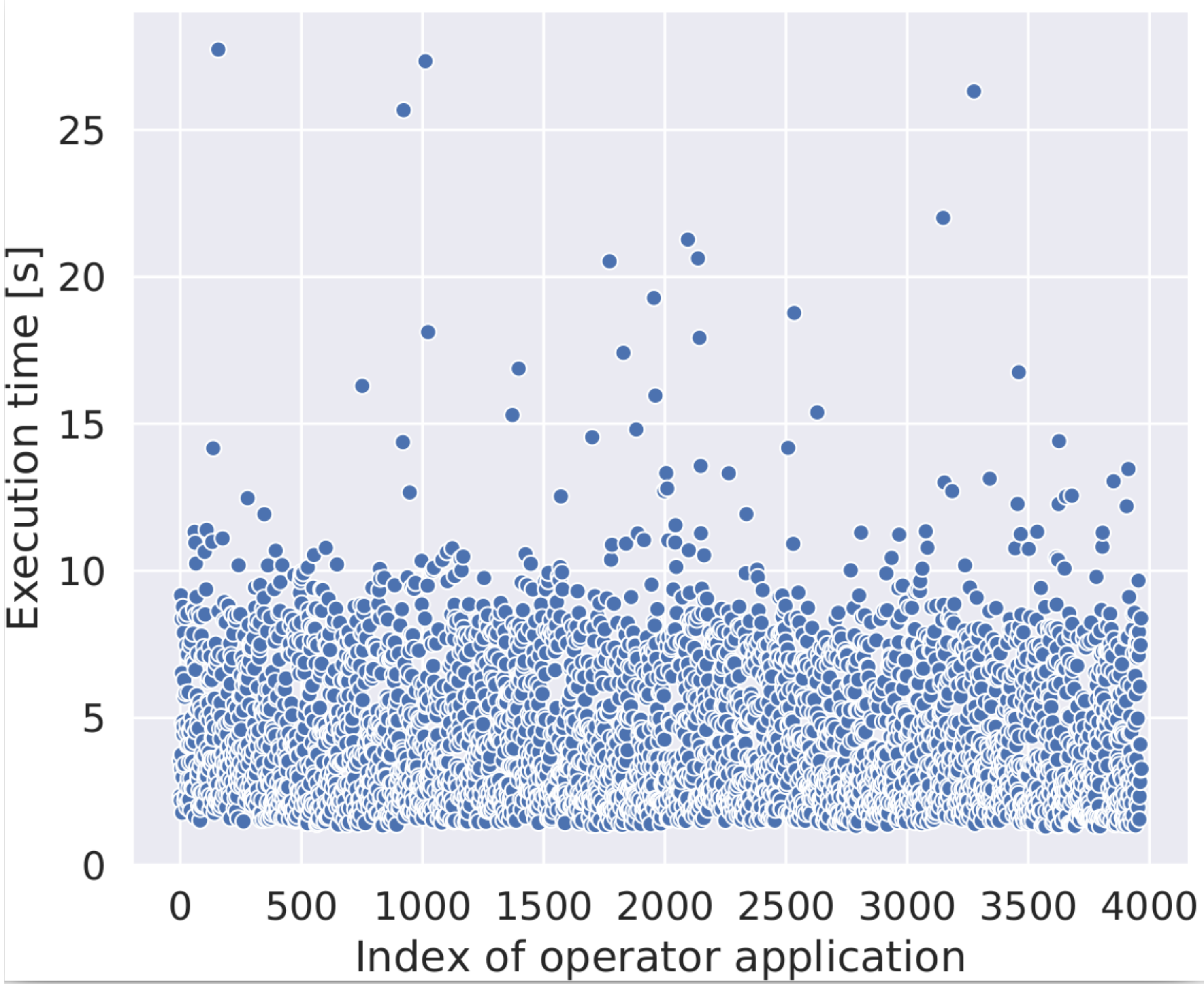


Image Registration



Not every prefix scan problem is…

… trivial to compute
… stable
… perfectly associative

Linked list prefix scan

Recap: breaking symmetry

- We can run parallel computation on independent sets. How to find them?

- Introduce randomness to create local differences!
    - Each node tosses a coin → 0 or 1
    - Let I be the set of nodes such that v drew 1 and $v.next$ drew 0!
        - What is the probability that $v \in I$?

Linked list prefix scan

Recap: breaking symmetry

- We can run parallel computation on independent sets. How to find them?

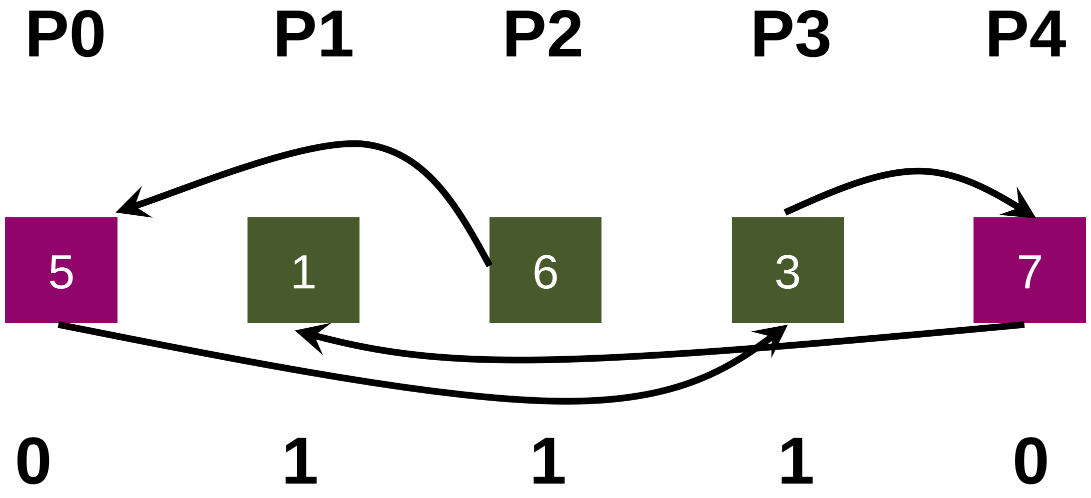
- Introduce randomness to create local differences!
    - Each node tosses a coin → 0 or 1
    - Let I be the set of nodes such that v drew 1 and $v.next$ drew 0!
        - What is the probability that $v \in I$?

$$P(v \in I) = \frac{1}{4}$$

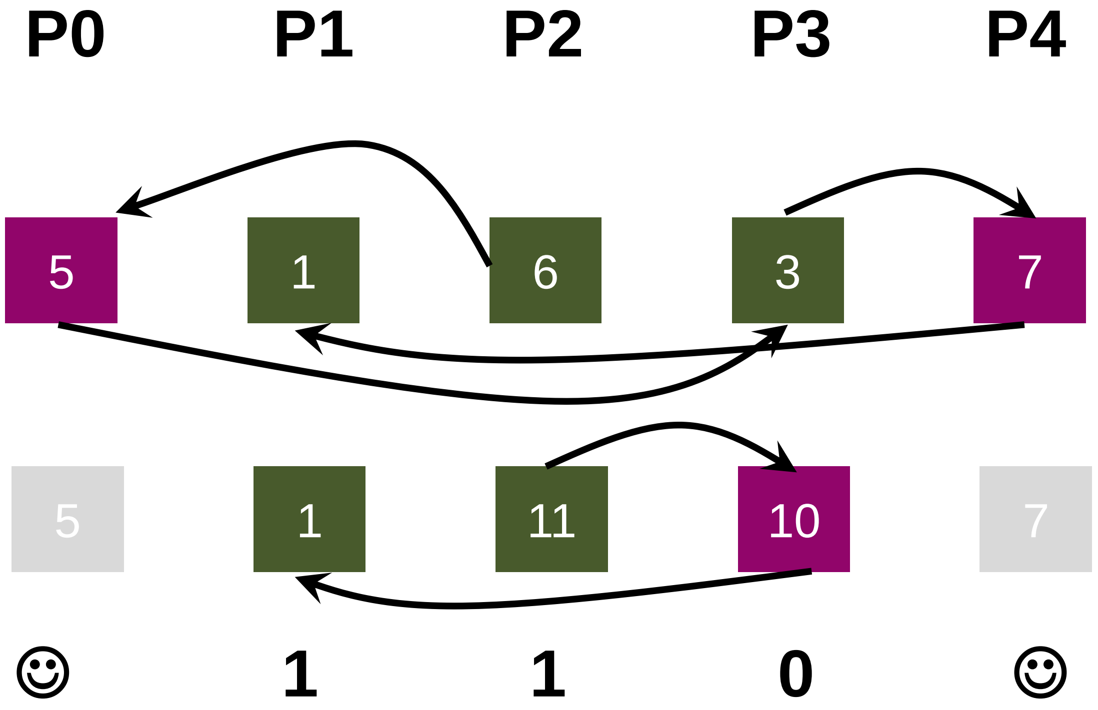
Linked list prefix scan - upward

P0 **P1** **P2** **P3** **P4**

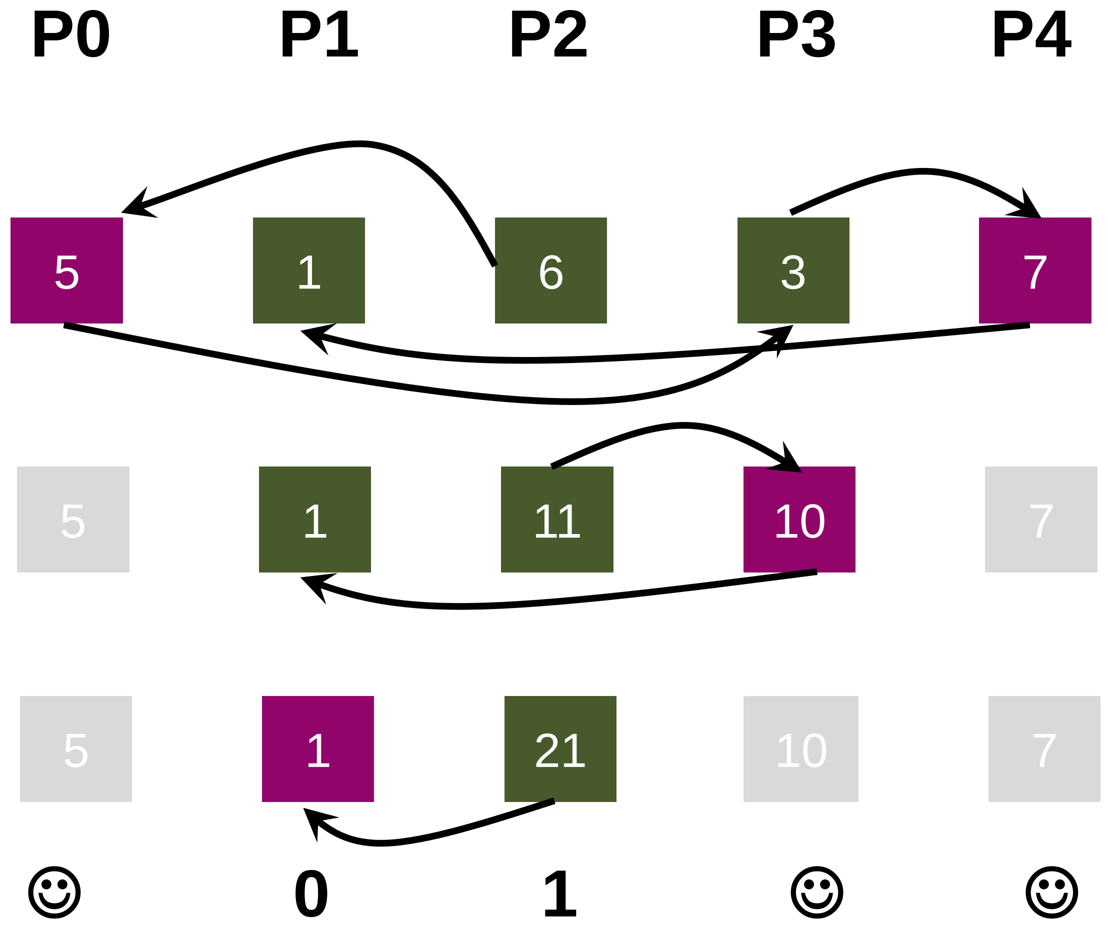
Linked list prefix scan - upward



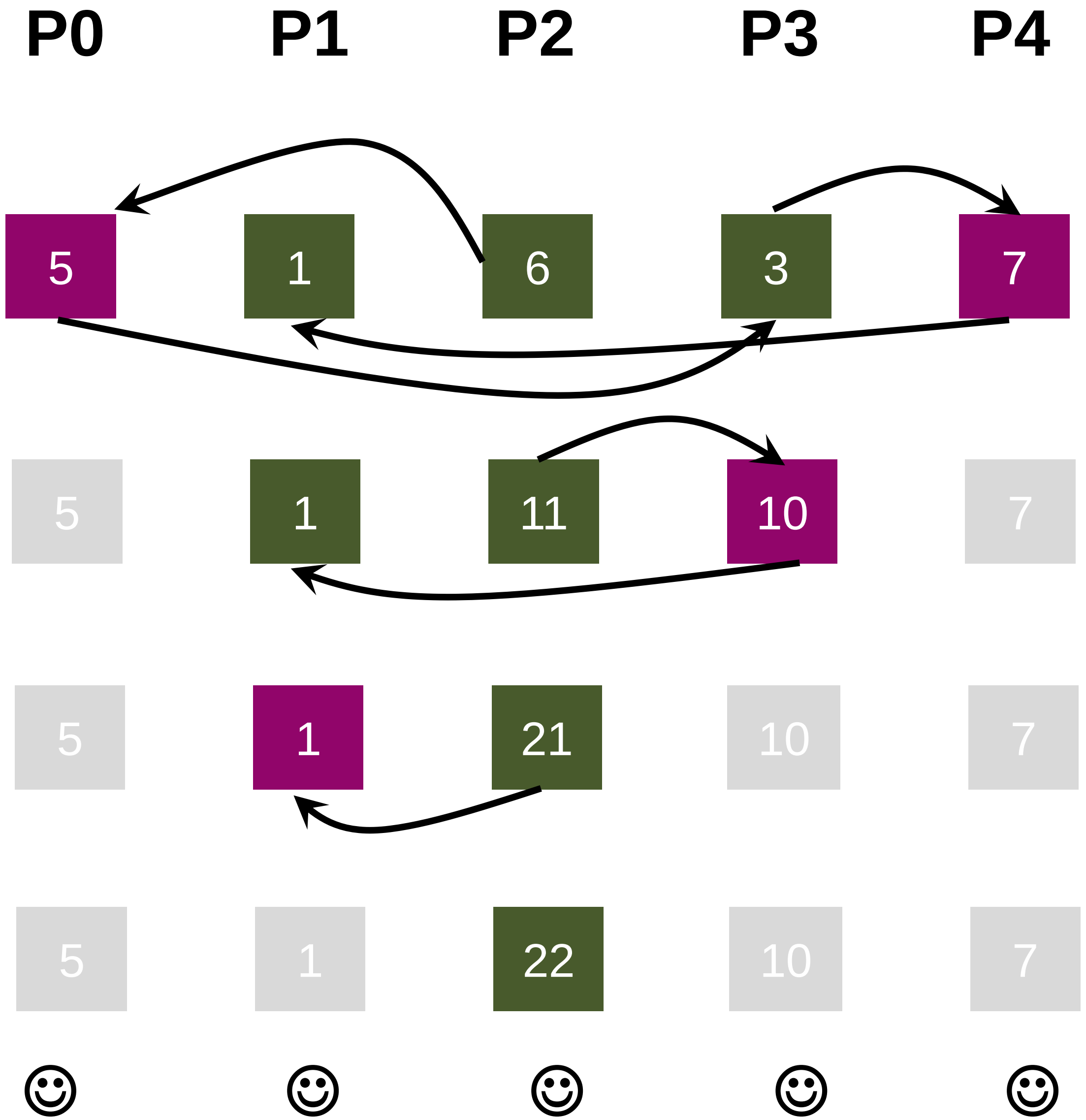
Linked list prefix scan - upward



Linked list prefix scan - upward



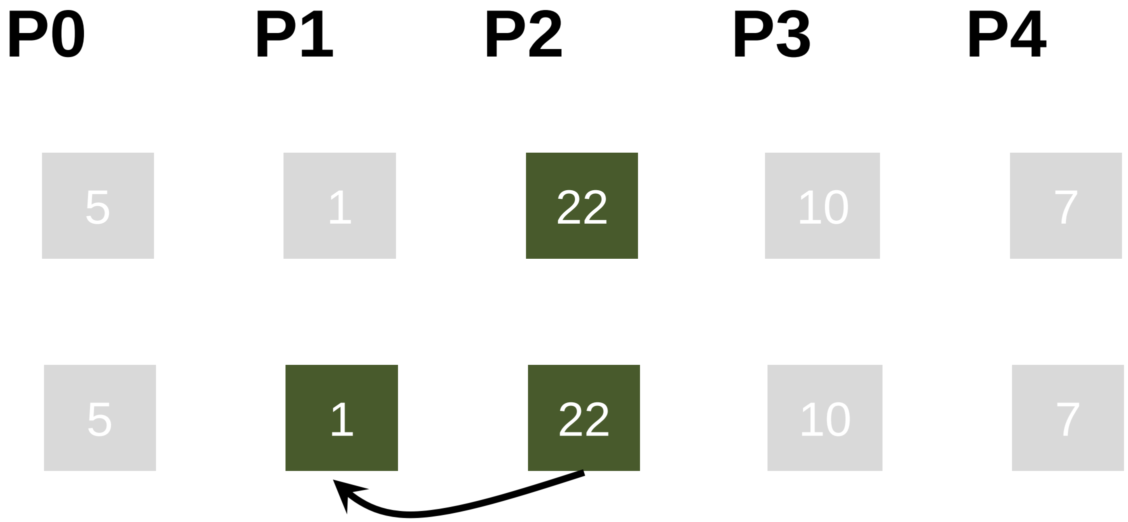
Linked list prefix scan - upward



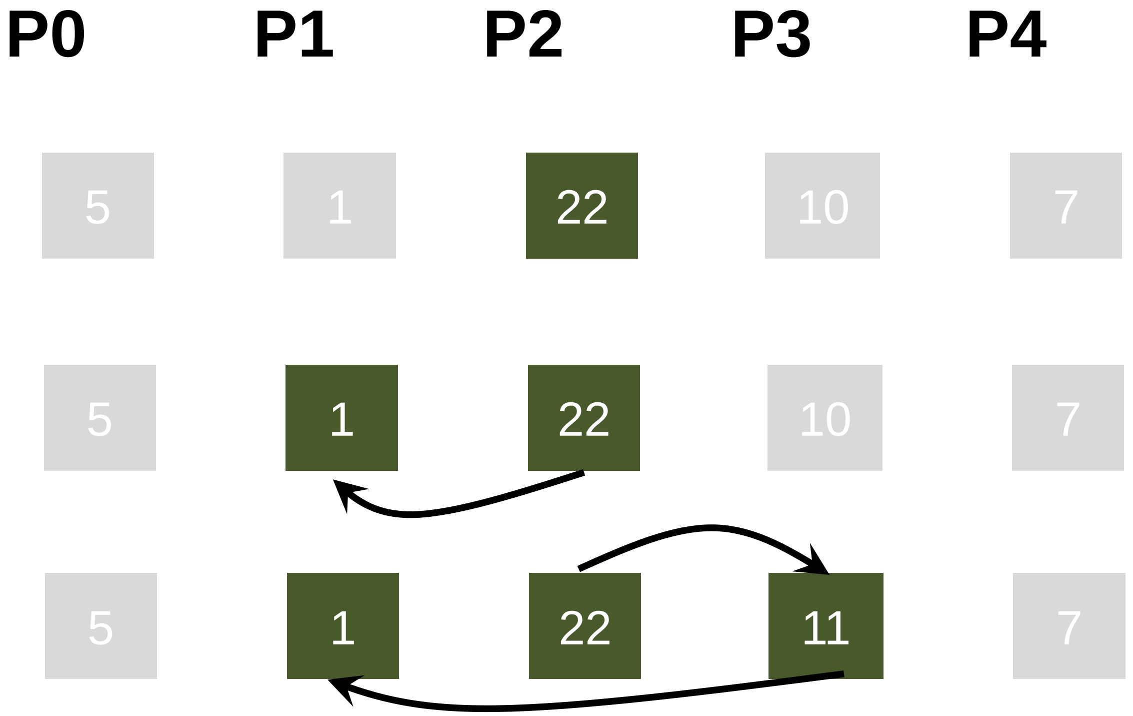
Linked list prefix scan - downward

P0	**P1**	**P2**	**P3**	**P4**
5	1	22	10	7

Linked list prefix scan - downward

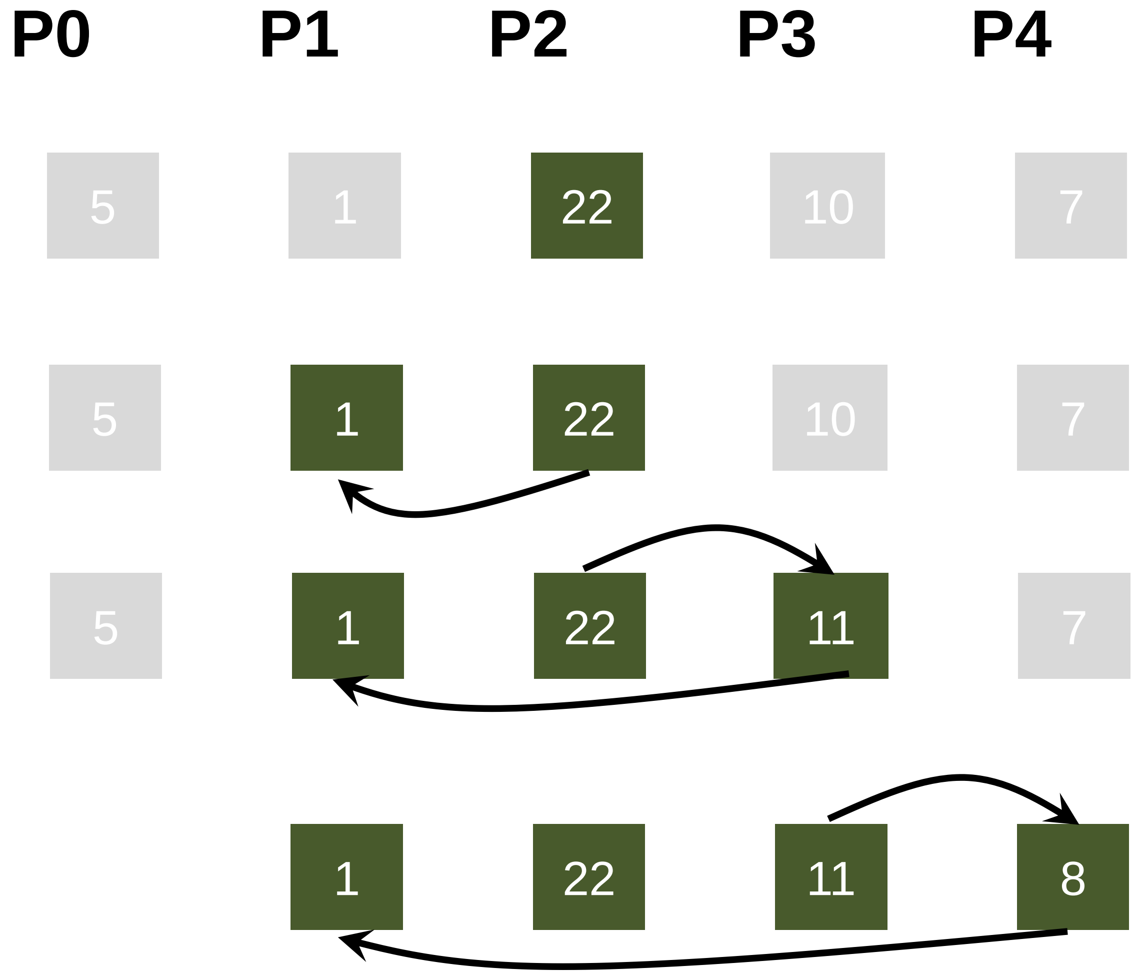


Linked list prefix scan - downward



P0	P1	P2	P3	P4
5	1	22	10	7
5	1	22	10	7
5	1	22	11	7

Linked list prefix scan - downward



Linked list prefix scan - downward