Parallel Processing

Unit 4

Dr. Ahmed Sallam

www.sallam.cf > (sallamah.weebly.com)

Quiz

On scan of N elements the amount of work is:

- O(log n)
- ✓ O(n)
- O(n log n)
- O(n^2)

On scan of N elements the number of steps is:

- ✓ O(log n)
- O(n)
- O(n log n)
- O(n^2)

Outlines

- Compact
- Compact-like
- Segment scan
- Sorting
  - Odd Even Sort
  - Merge Sort
  - Radix Sort
  - Quick Sort

Compact (Filter)

COMPACT

- Filter red color objects only
  - This help us to keep objects we care about and ignore other objects.
  - This saves space and processing time.

COMPACT MODEL

Input:	a	b	c	d	e	…	
Predicate: e.g. "Is my index even?"	T	F	T	F	T	…	
Output:	a	-	c	-	e	…	Sparse
	a	c	e	…			Dense

WHY WE PREFER DENSE COMPACT?

- Suppose we want to apply toGray() on every red object.

```
//Sparse
if( isRed(object)){

        toGray(object)
}
```

15 threads

```
//Dense
cc=compact(objects, isRed())


map(cc, toGray())
```

15 threads

4 threads

QUIZ

When to use compact?

When the number of elements …..

- Small
- ✓ Large

When the operation on this elements are …..

- ✓ Expensive (complex)
- Cheap (Simple)

Compact Parallelization

- How to compute the scatter address/index in parallel?

Input: a b c d e …

Predicate: T F T F T …
e.g.
"Is my index even?"

Output: a - c - e … Sparse

a c e … Dense

COMPACT PARALLELIZATION

- We can paraphrase the problem as following:

Input:	**T**	**F**	**F**	**T**	**T**	**F**	
Output:	**0**	**-**	**-**	**1**	**2**	**-**	

- And for computer we paraphrase it again as following:

Input:	**1**	**0**	**0**	**1**	**1**	**0**	**???**
Output:	**0**	**1**	**1**	**1**	**2**	**3**	

Exclusive Scan

Compact Parallel Algorithm

1. Generate Predicate array:

```
predicate_array = predicate_function(input_array)
```

2. Generate Scan-in array (1s and 0s)

```
scan-in_array = convert(predicate_array)
```

} map

3. Generate Scatter-addresses array :

```
addresses_array = exclusive_sum_scan(scan-in_array)
```

} scan

4. Scatter input elements to addresses

```
output = scatter(addresses_array, input_array)
```

} scatter

QUIZ

- Suppose we need to compact 1M element with the following predicate functions:
  - A: isDivisiableBy_17( ) Few elements
  - B: isNotDivisiableBy_34( ) Many elements

	A Faster	A = B	B Faster
Map		✓	
Scan		✓	
Scatter	✓		

Outlines

- Compact
- Compact-like
- Segment scan
- Sorting
  - Odd Even Sort
  - Merge Sort
  - Radix Sort
  - Quick Sort

Compact-Like

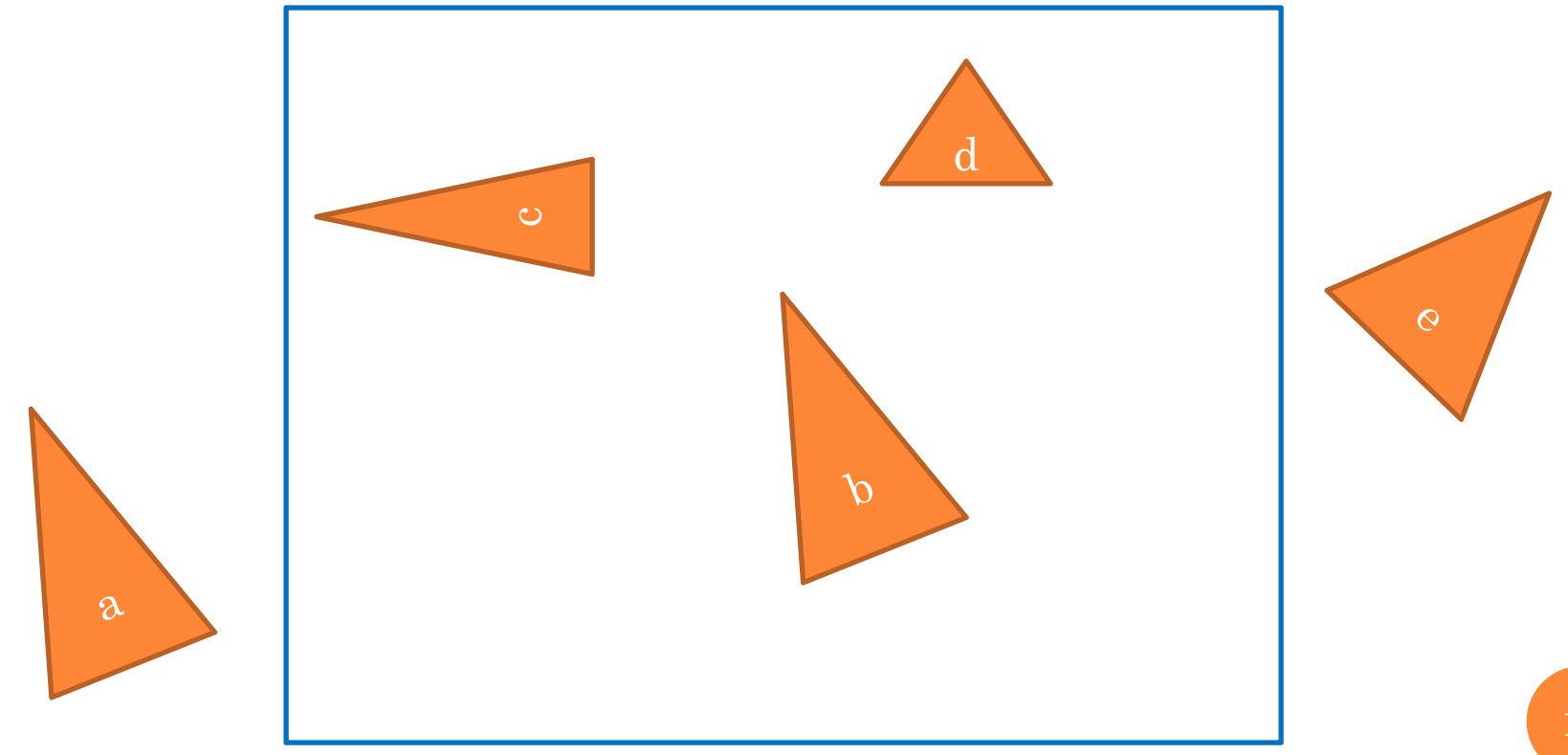
Recap Compact allocation

- Compact allocate 1 output for 1 (true) element and 0 output for 1 (false).

- Can we generalize?? (not only 1s)
  - The number of output can be computed dynamically for each input items.

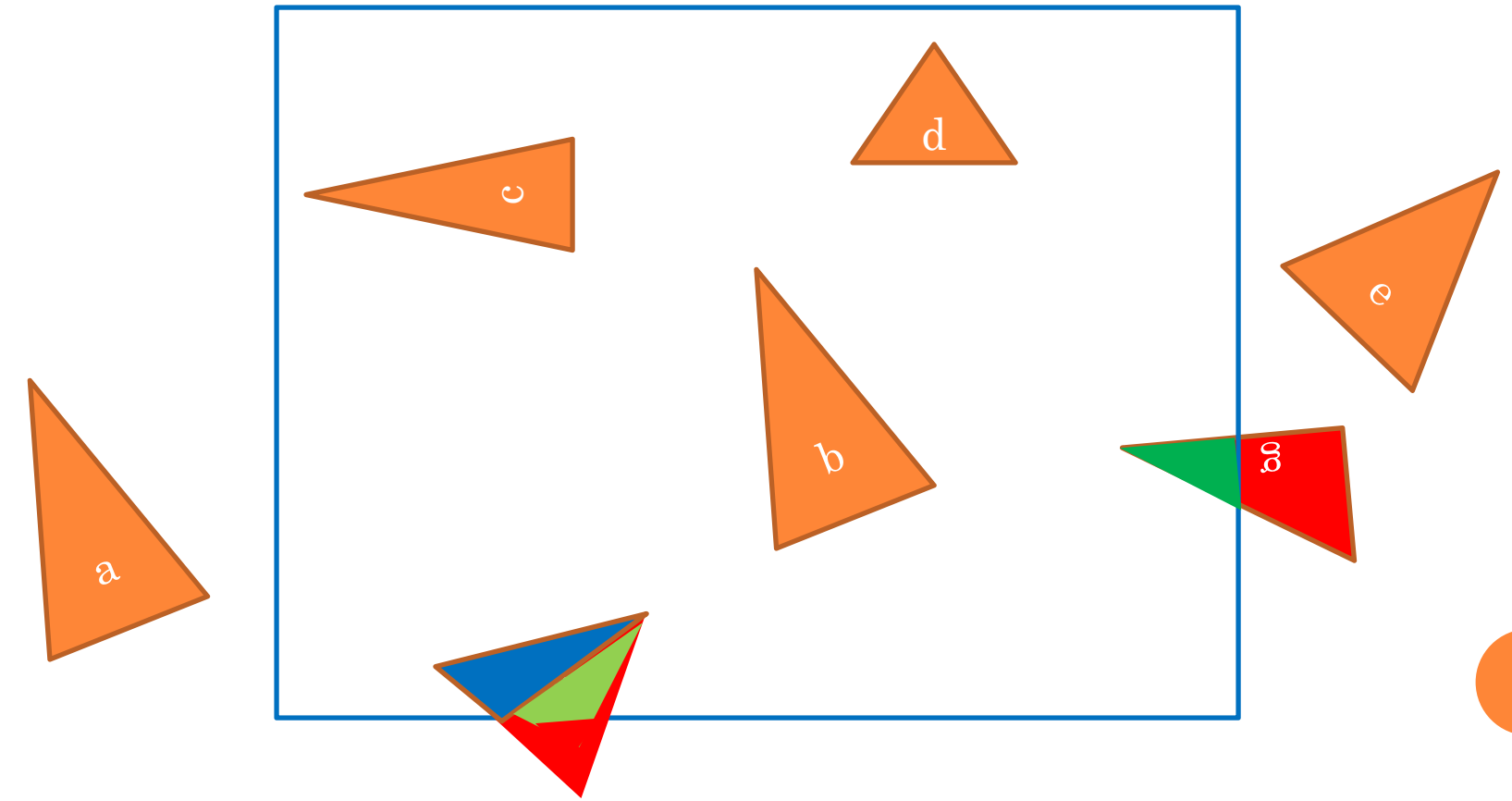
CLIPPING

- Suppose a set of triangles are sent as input to a computer graphics pipeline



Clipping Problem

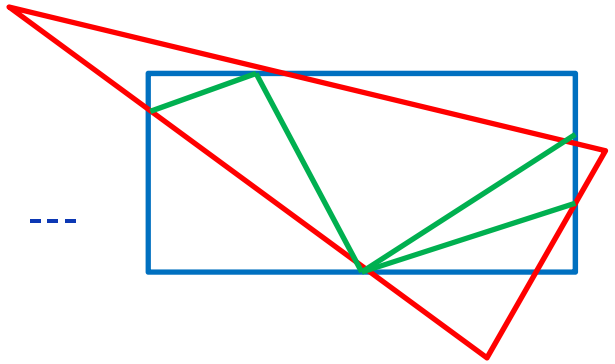
- How to clip triangles at the boundries?



BAD SOLUTION

Input: | a | c | e | d | f | b | g |

- Allocate maximum possible space in intermediate array
  - 5 for each triangle in our case
  - Intermediate: | a | ? | ? | ? | ? | b | ? | ---
- Apply compact

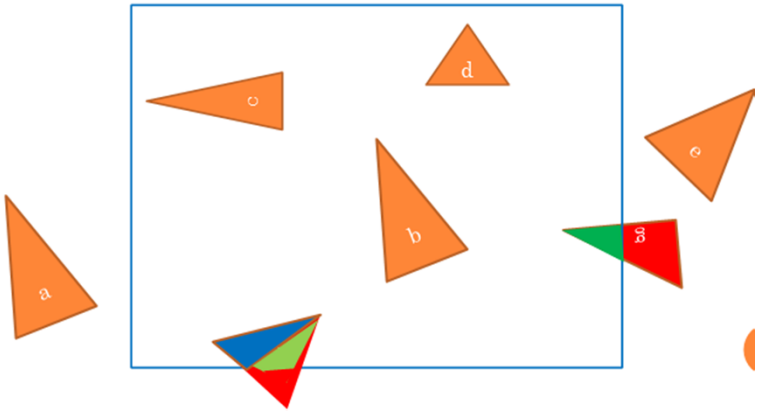


Disadvantages:

- Wasteful in space
- Scanning large intermediate array

GOOD SOLUTION (GENERAL COMPACT)

Input: | a | c | e | d | f | b | g |



- Allocation requests per input element

request: | 0 | 1 | 0 | 1 | 2 | 1 | 1 |

- Apply scan

Addresses: | 0 | 0 | 1 | 1 | 2 | 4 | 5 |

- Allocate output array based with respect to max scan #, and then apply scatter

output: | 0 | 1 | 2 | 3 | 4 | 5 |

Other Application of Scan

- Data Compression
- Collision Detection

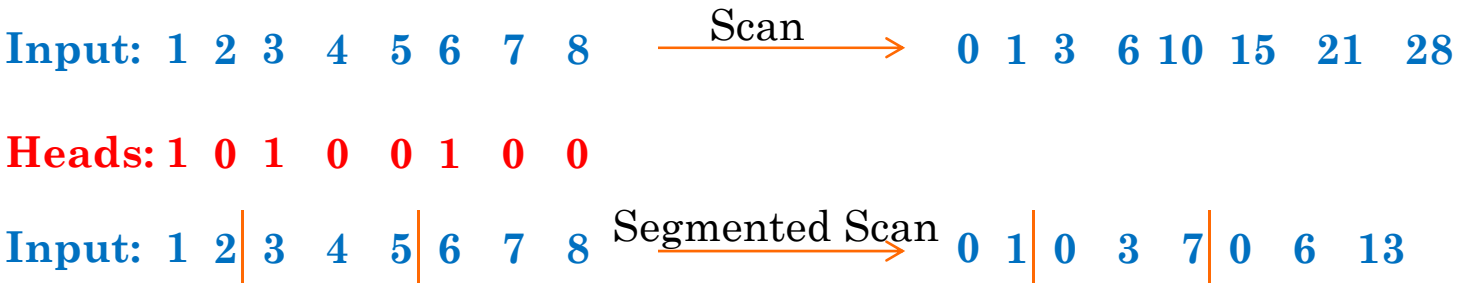
Outlines

- Compact
- Compact-like
- Segment scan
- Sorting
  - Odd Even Sort
  - Merge Sort
  - Radix Sort
  - Quick Sort

Segmented Scan

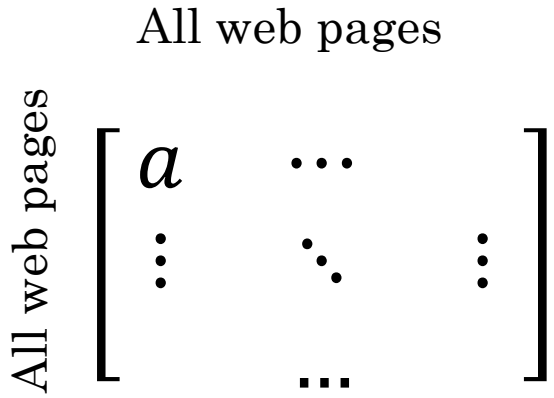
Segmented Scan

- Many small Scans
  - Lunch each independently
  - Combine as segments
- Remember we back all segments in one big array to be processed by one kernel, instead of running a separate kernel over each segment to gain max benefits from GPU power.
- We use a separate array to indicate segments' heads



SEGMENTED SCAN APPLICATION

- "Sparse matrix *Dense vector" multiplication (SpMv)
  - Sparse matrix: contains all elements includes a lot of zeros
  - Dense matrix: doesn't contain zeros
- Sparse matrix multiplication comes with a lot of unnecessary multiplication
- E.g. Google PageRank
  - a is a non-zero value indicate there is a link between webpages indicated by column index and webpage indicated by row index.



Review matrix multiplication

$$\begin{bmatrix} 1 & 0 & 3 \\ 2 & 4 & -1 \\ 0 & 1 & 5 \end{bmatrix} \text{x} \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} =$$

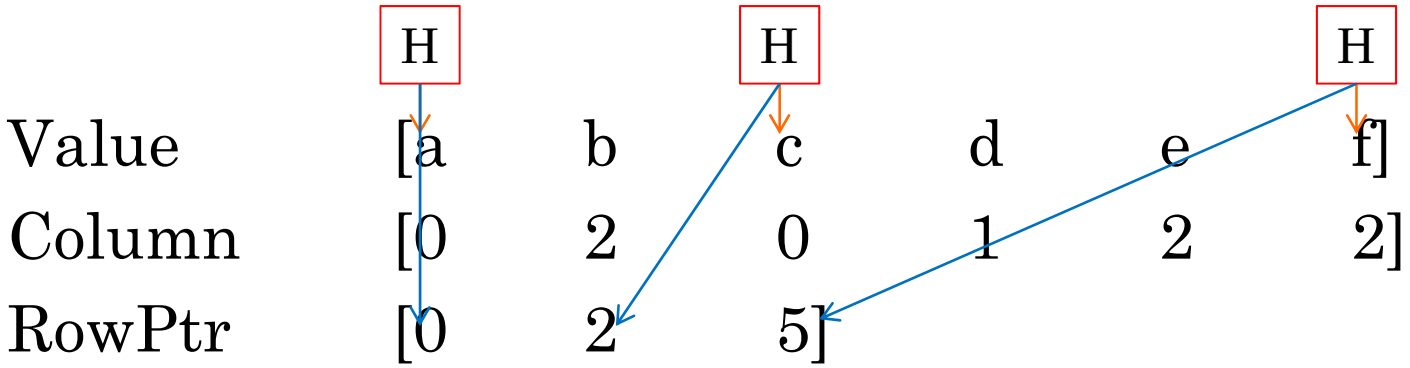
$$\begin{bmatrix} (1*0)+(0*1)+(3*2) \\ (2*0)+(4*1)+(-1*2) \\ (0*0)+(1*1)+(5*2) \end{bmatrix} =$$

$$\begin{bmatrix} 6 \\ 2 \\ 11 \end{bmatrix}$$

Compressed Sparse Row

- We can represent sparse matrix in CPR format as following:

$$\begin{bmatrix} a & 0 & b \\ c & d & e \\ 0 & 0 & f \end{bmatrix}$$



CPR MULTIPLICATION

- We can represent sparse matrix in CPR format as following:
  - Value: [a b c d e f]
  - Column: [0 2 0 1 2 2]
  - RowPtr: [0 2 5]

$$x \begin{bmatrix} x \\ y \\ z \end{bmatrix} \begin{matrix} 0 \\ 1 \\ 2 \end{matrix}$$

1. Create Segments with values and RowPtr

   [a b c d e f] $\xrightarrow[\text{[0 2 5]}]{\text{RowPtr:}}$ [a b | c d e | f]

2. Gather vector values using column

   $\xrightarrow[\text{[0 2 0 1 2 2]}]{\text{column:}}$ [x z x y z z]

3. Pairwise multiply 1 . 2

   [a*x b*z | c*x d*y e*z | f*z]

4. Apply exclusive backward sum scan (at the head)

   Can we apply reduce instead?? out(0) out(1) out(2)

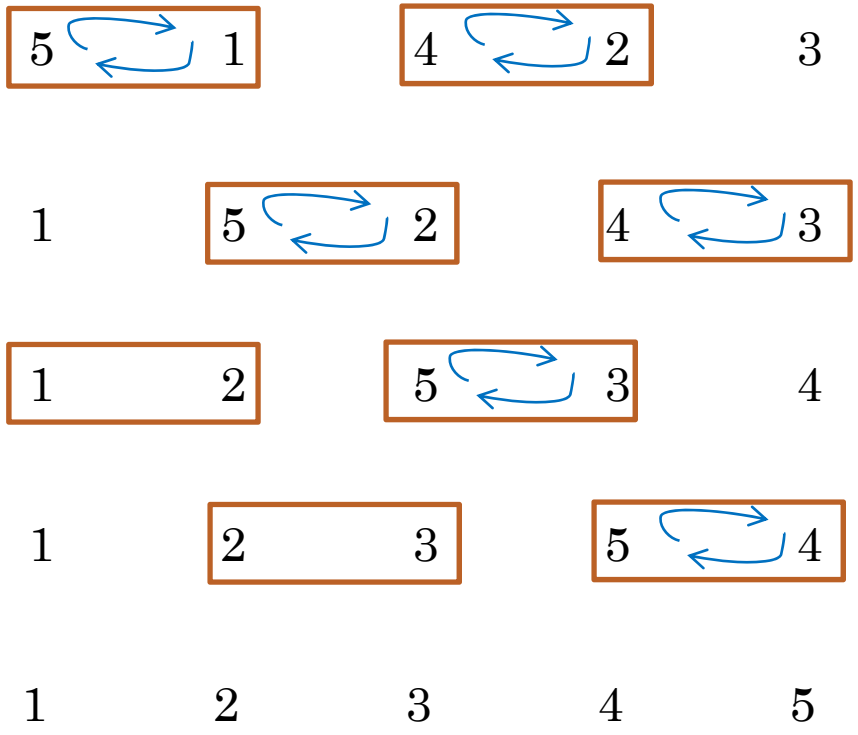
Outlines

- Compact
- Compact-like
- Segment scan
- Sorting
  - Odd Even Sort
  - Merge Sort
  - Radix Sort
  - Quick Sort

Sorting

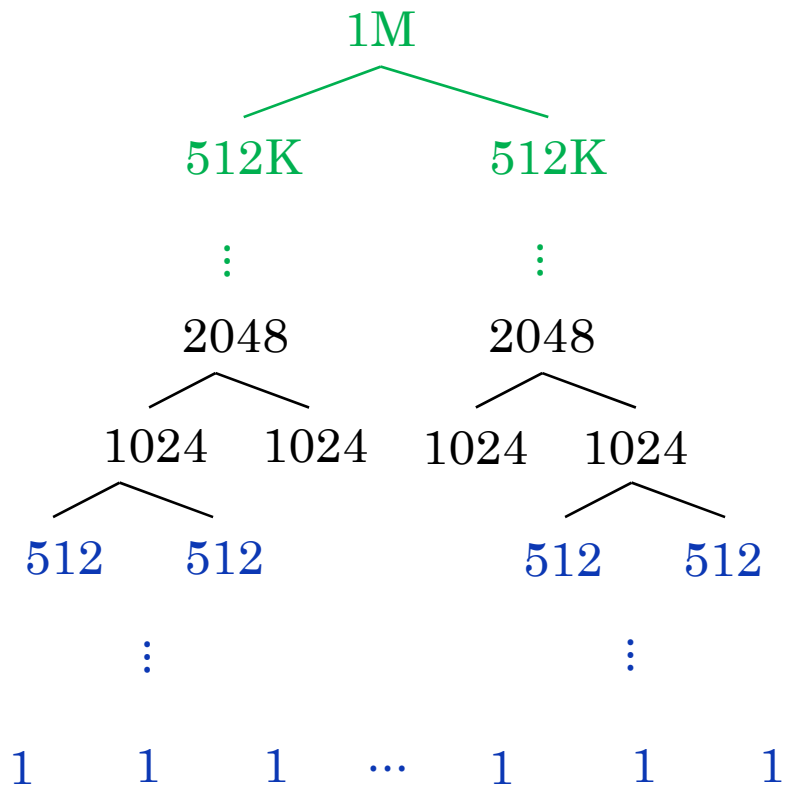
Odd-Even (Brick) Sort

- It's the parallel version of bubble sort



	Step?	Work?
O(1)	☐	☐
O(n)	✓	☐
O(log n)	☐	☐
O(n log n)	☐	☐
$O(n^2)$	☐	✓

Merge Sort



Stage 3:

- We have only 1merge task
- 2 huge sorted lists
- Choose splitters (256th element) in each list. and sort them
- Use merge task in stage 2 to merge elements between splitters

Stage 2:

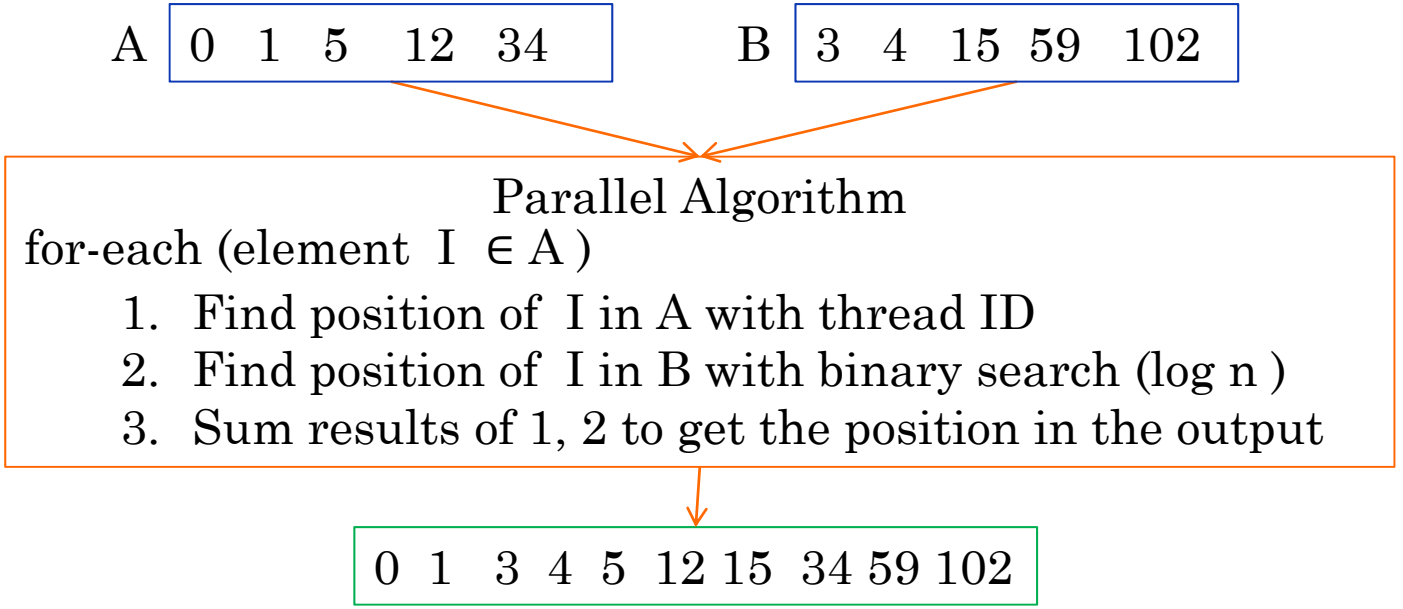
- Bunch of medium merge tasks
- 1 merge to 1 block

Stage 1:

- Tons of small merge tasks
- 1 merge to 1 thread
- We can use shared memory
- Also we can use serial algorithm to sort small block of elements

MERGE TASK

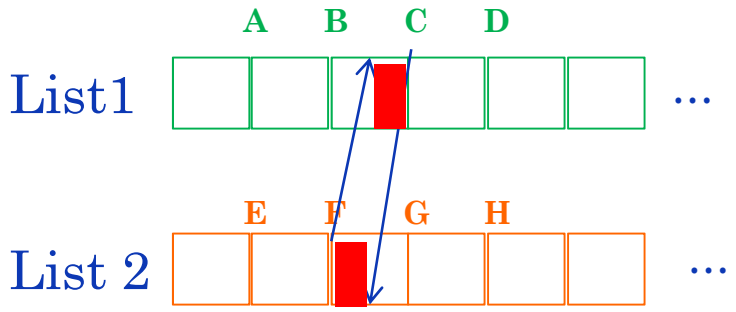

0 1 5 12 34
Sorted
3 4 15 59 102
Serial Algorithm
Compare heads
0 1 3 4 5 12 15 34 59 102


A
0 1 5 12 34
B
3 4 15 59 102
Parallel Algorithm
for-each (element I ∈ A)
1. Find position of I in A with thread ID
2. Find position of I in B with binary search (log n)
3. Sum results of 1, 2 to get the position in the output
0 1 3 4 5 12 15 34 59 102

MERGE TASK (CONT.)

- How to merge two huge lists in parallel?
  - No one can do all of this work

Find n^{th} (e.g. 256^{th}) splitter elements



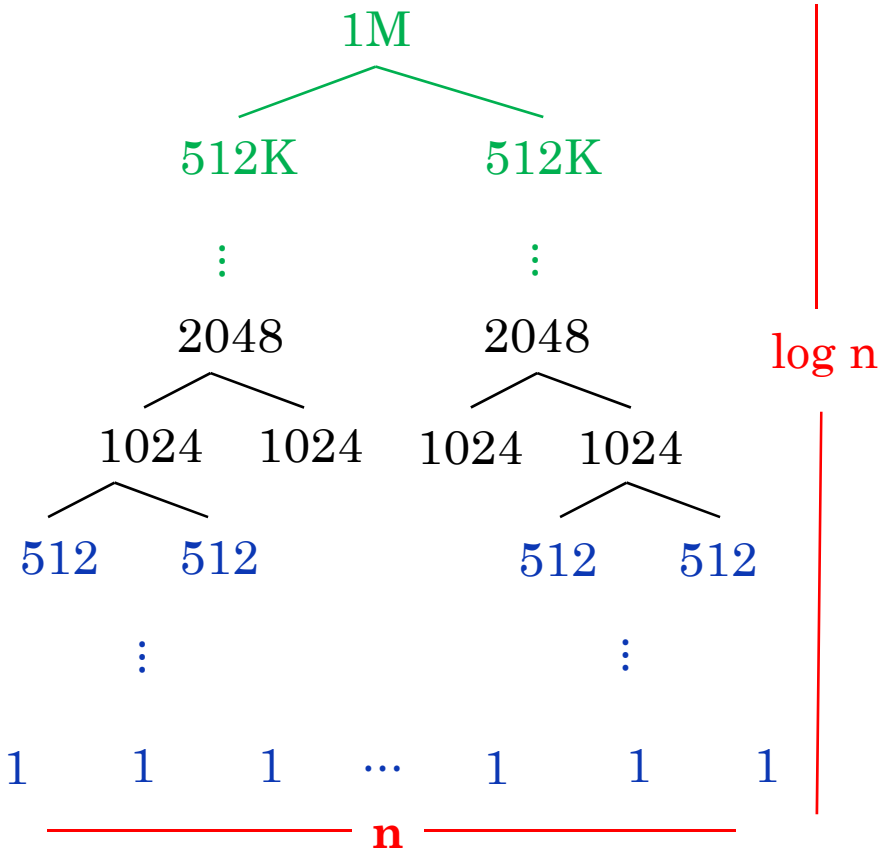
Merge splitters

E A B F C G D H

Merge elements between every two consecutive splitters
e.g. F and C:

1. Find F position in list 1
2. Find C position in list 2
3. Merge elements between these two positions in every list.

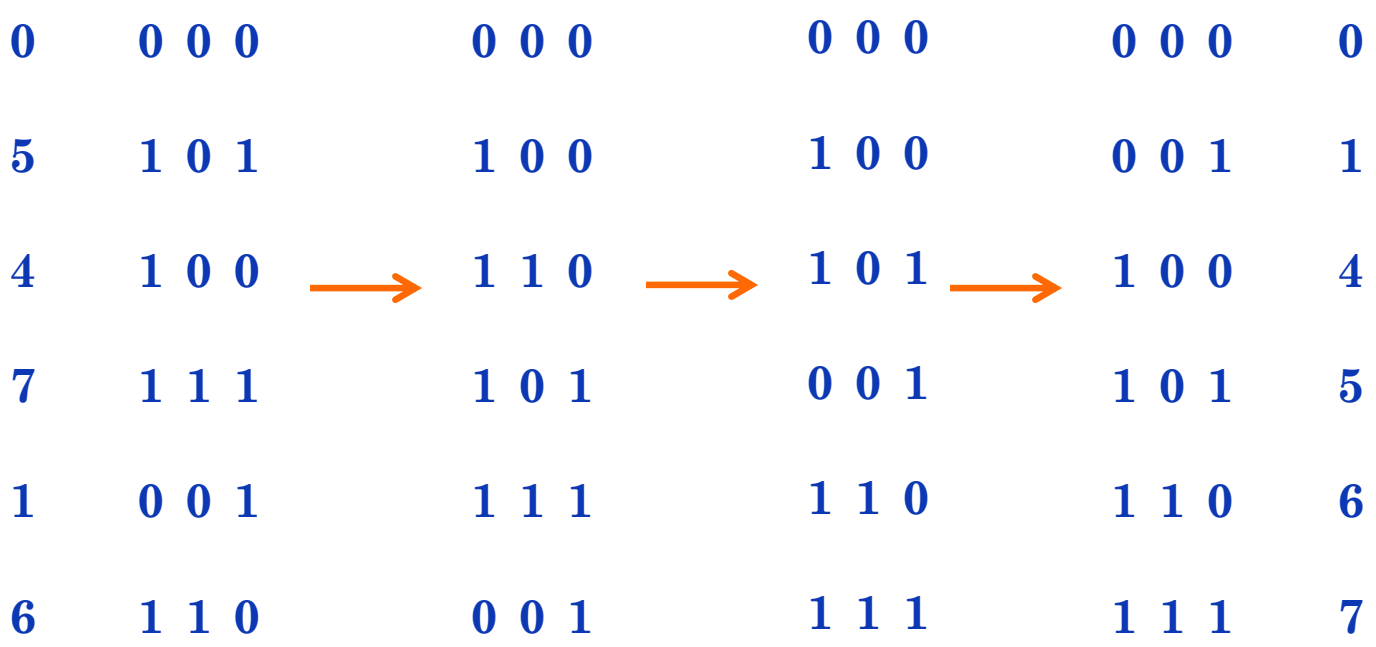
Merge Analysis



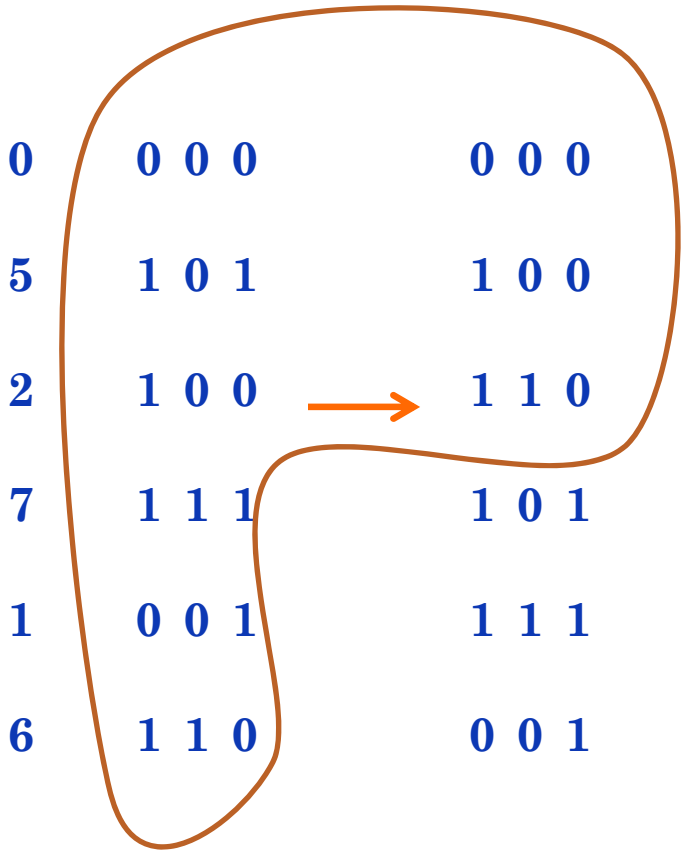
	Step?	Work?
O(1)	☐	☐
O(n)	☐	☐
O(log n)	✓	☐
O(n log n)	☐	✓
$O(n^2)$	☐	☐

Radix Sort

1. Start with LSB
2. Split input into 2 sets based on the current bit
3. Move to next MSB, and repeat 1



Radix Sort Penalization



What is this Algorithm?

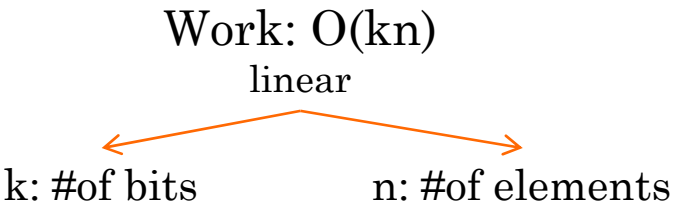
Compact

What is The Predicate?

(i&1)==0

Radix Sort Analysis

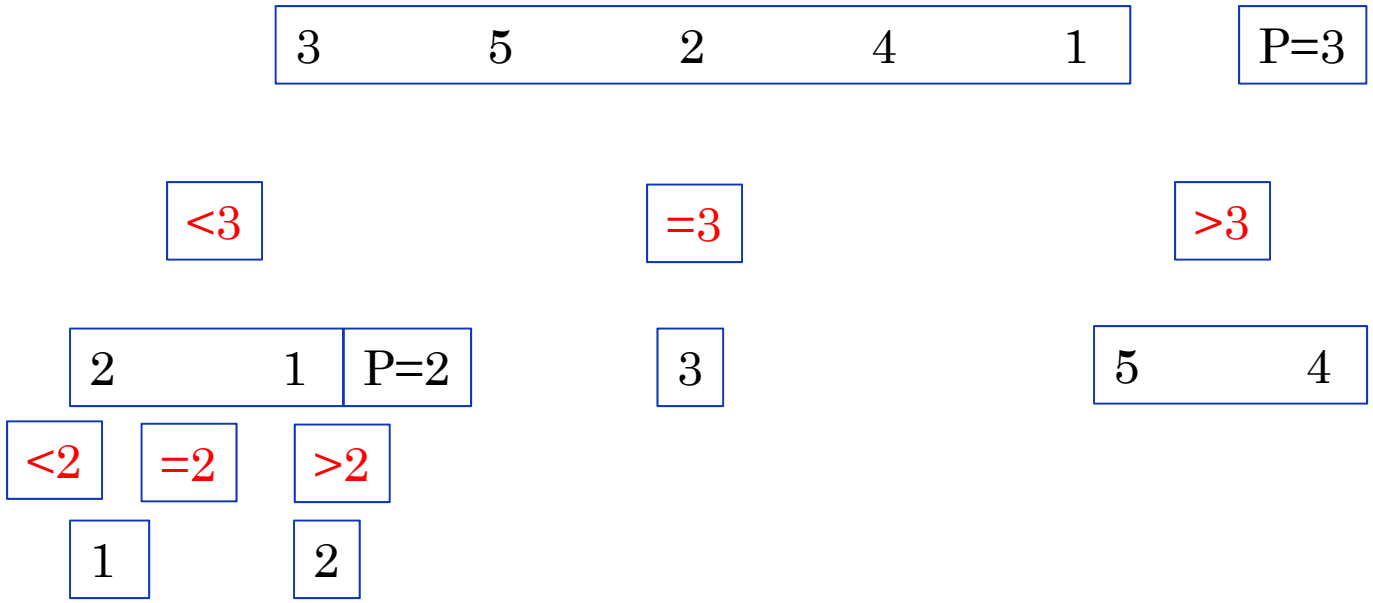
0	0 0 0	0 0 0	
5	1 0 1	1 0 0	
2	1 0 0	1 1 0	
7	1 1 1	1 0 1	
1	0 0 1	1 1 1	
6	1 1 0	0 0 1	



Steps: O(k)

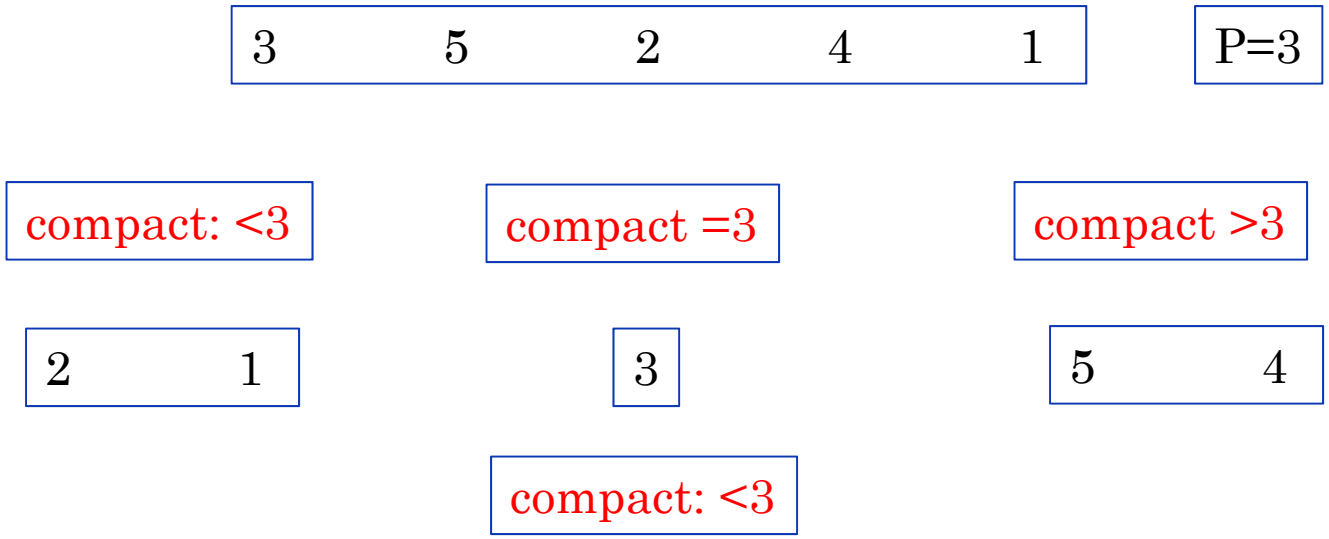
Quick Sort

- Choose Pivot element
- Compare all elements with Pivot
- Split into 3 Arrays <p, =p, >p
- Recursion on each array



QUICK SORT PENALIZATION

- Old GPUs doesn't support recursion



- Current GPUs support recursion

NOTE

- All sort algorithms that we have studied are key value sorts where we usually depend on an integer key to sort.

- However if you have items with different data item (e.g. structure with many value).
  - Use a key or a pointer to this value to apply sorting

Outlines

- Compact
- Compact-like
- Segment scan
- Sorting
  - Odd Even Sort
  - Merge Sort
  - Radix Sort
  - Quick Sort

Red Eye removal

- Stencil
- Sort
- Map