CS186 Discussion #1

(Introduction, external sorting & hashing)

Michelle Nguyen

Google

CS 160

Berkeley cs186

FUNG INSTITUTE for ENGINEERING LEADERSHIP
UC BERKELEY COLLEGE OF ENGINEERING

Master of Engineering

BiD

Now it’s your turn!

- Meet the people around you: name, major, year, interests, etc.

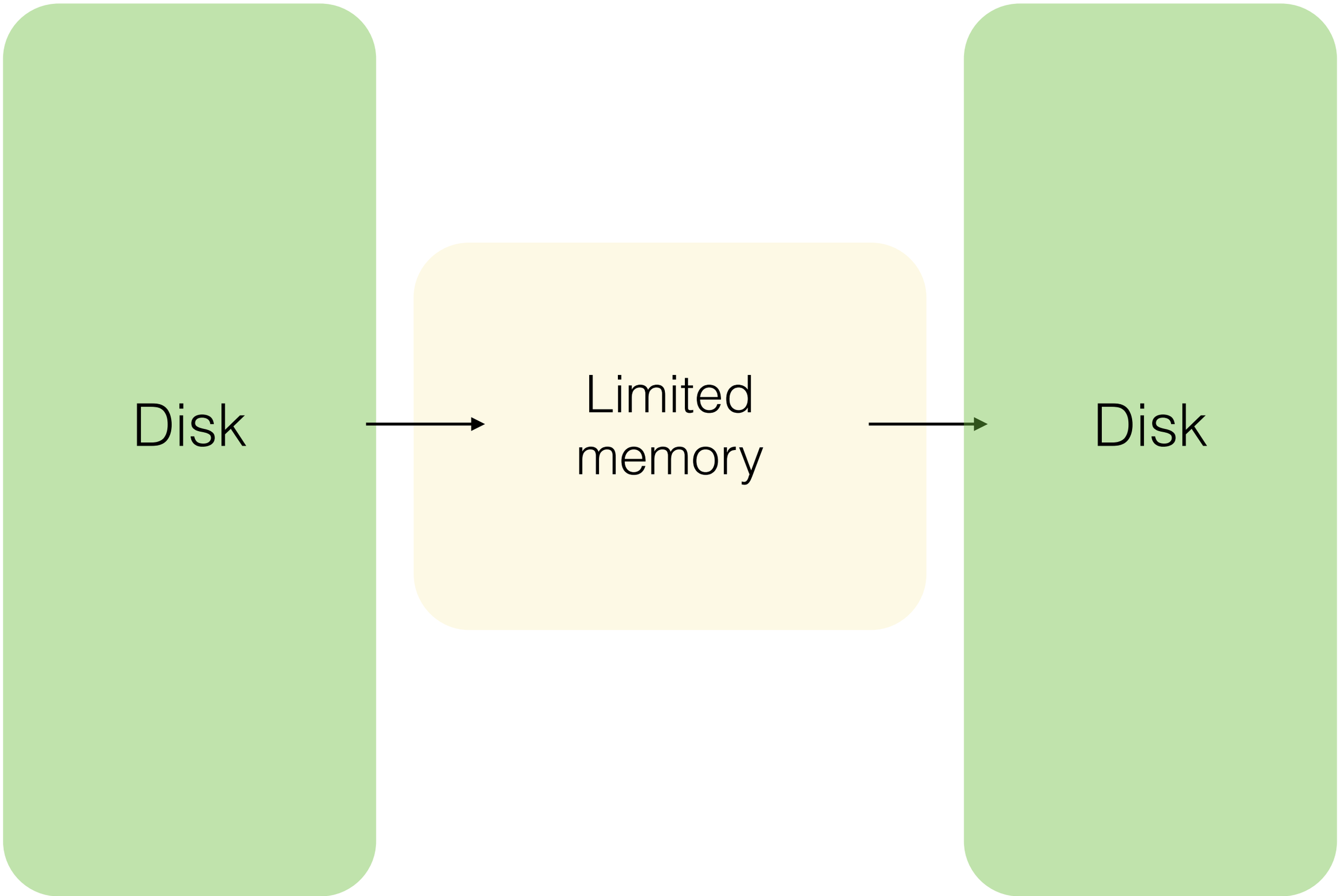
- Partners are **recommended** in this course. If you don’t have one, take this opportunity to find one!

Enrollment

- We will not be expanding the course.
- We cannot control the processing of the waitlist.
- CS186 will be offered again Spring 2016
  - (Joe Hellerstein is awesome!)

Logistics questions?

Out-of-core? External?


Disk
Limited memory
Disk

Time-space Rendezvous

- When items are in memory at the same time
  - Ex: Aggregation or eliminating duplicates
- Implemented through:
  - External sorting
  - External hashing

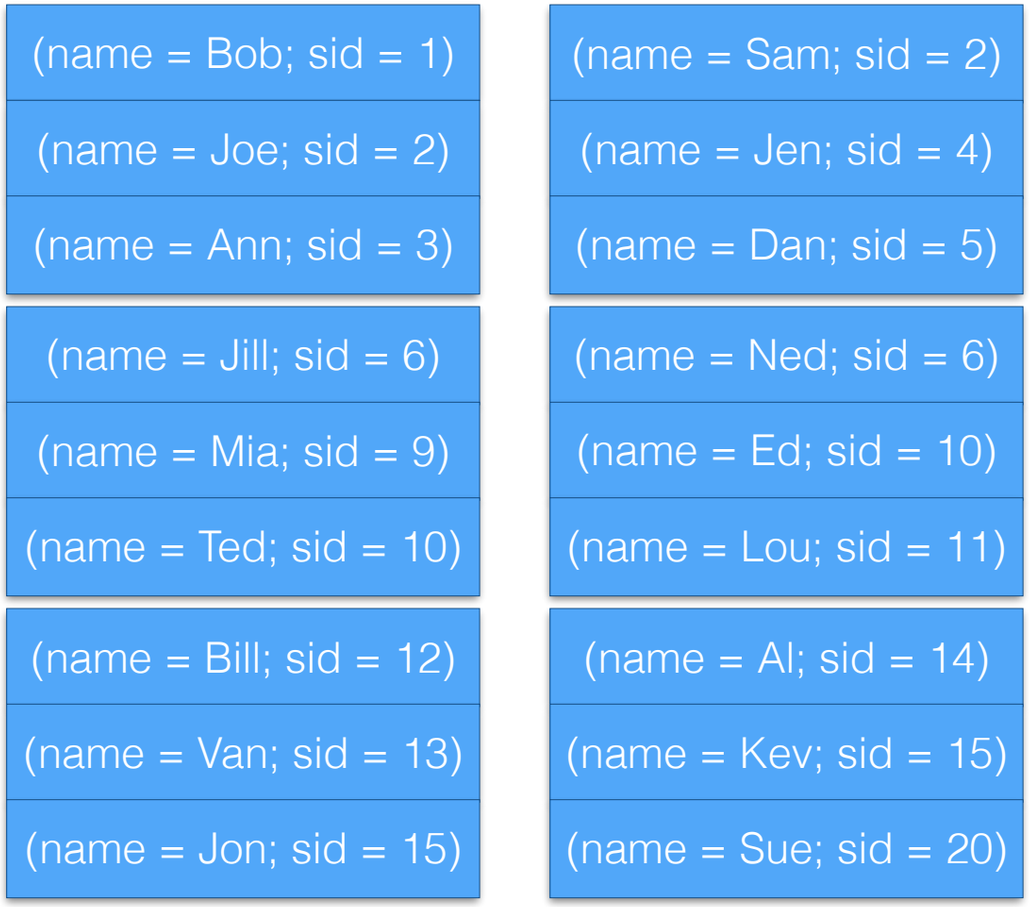
External Sorting

External Sorting

- Want to sort data that does not fit in memory
- Minimize number of I/O's (especially random I/O's)

Terminology: Sorted Runs

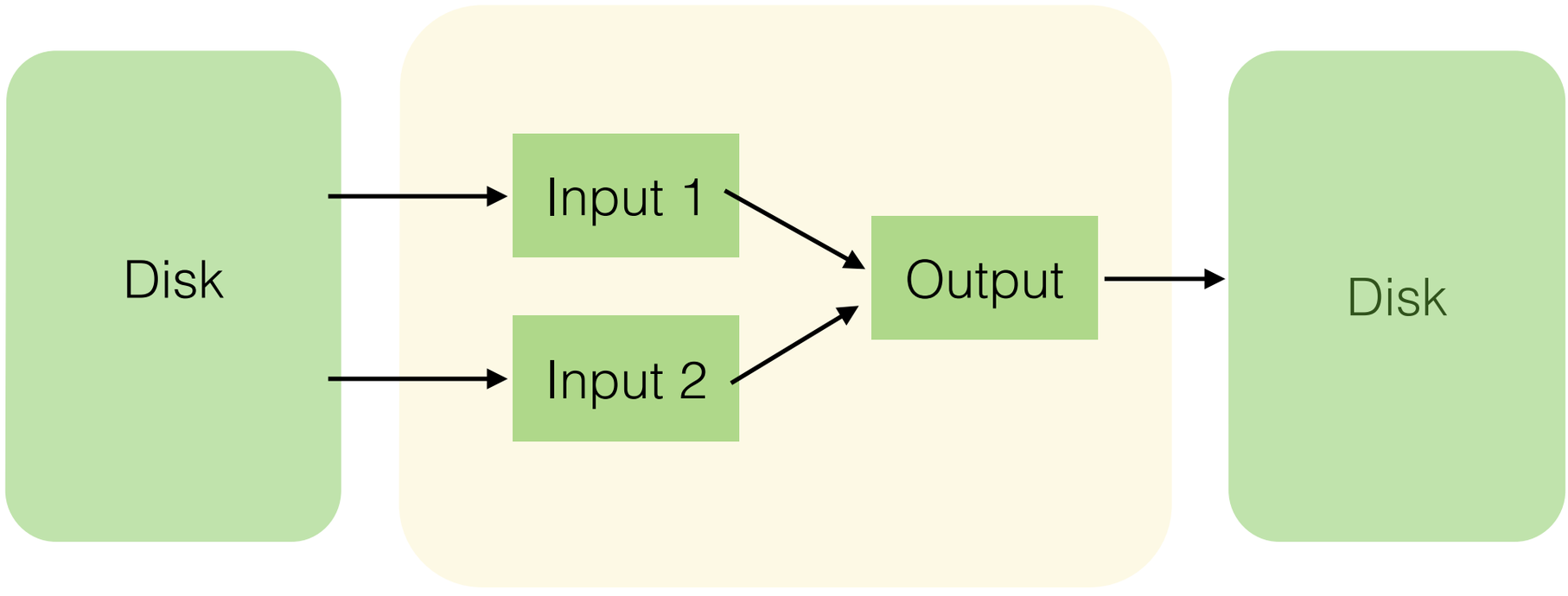
- A sorted subset of a table
- Size is denoted by how many pages it spans



Pages with tuple size = 3

There are two sorted runs, both with a length of 3 pages.

2-Way Merge Sort



Use 3 pages of buffer

Input

3,4

6,2

9,4

8,7

5,6

6,5

1,4

4,2

Input

Pass 0

Output

3,4

6,2

9,4

8,7

5,6

6,5

1,4

4,2

Input

Pass 0

Output

6,2

9,4

8,7

5,6

6,5

1,4

4,2

3,4

Input

Pass 0

Output

6,2

9,4

8,7

5,6

6,5

1,4

4,2

3,4

Input

Pass 0

Output

9,4

8,7

5,6

6,5

1,4

4,2

6,2

3,4

Input

Pass 0

Output

9,4

8,7

5,6

6,5

1,4

4,2

2,6

3,4

Input

Pass 0

Output

9,4

8,7

5,6

6,5

1,4

4,2

3,4

2,6

Input

Pass 0

Output

8,7

5,6

6,5

1,4

4,2

9,4

3,4

2,6

Input

Pass 0

Output

8,7

5,6

6,5

1,4

4,2

4,9

3,4

2,6

Input

Pass 0

Output

8,7

5,6

6,5

1,4

4,2

3,4

2,6

4,9

Input

Pass 0

Output

3,4

2,6

4,9

7,8

5,6

5,6

1,4

2,4

Input

3,4

6,2

9,4

8,7

5,6

6,5

1,4

4,2

Pass 0

3,4

2,6

4,9

7,8

5,6

5,6

1,4

2,4

1 page runs

Input

Pass 1

Output

3,4

2,6

4,9

7,8

5,6

5,6

1,4

2,4

Input 1

Input 2

Output

Input

Pass 1

Output

4,9

7,8

5,6

5,6

1,4

2,4

Input 1

3,4

Input 2

2,6

Output

Input

Pass 1

Output

4,9

7,8

5,6

5,6

1,4

2,4

Input 1

3,4

Input 2

6

Output

2

Input

Pass 1

Output

4,9

7,8

5,6

5,6

1,4

2,4

Input 1

4

Input 2

6

Output

2,3

Pass 1

Input

4,9

7,8

5,6

5,6

1,4

2,4

Input 1

4

Input 2

6

Output

Output

2,3

Input

Pass 1

Output

2,3

4,9

7,8

5,6

5,6

1,4

2,4

Input 1

Output

4,6

Input 2

Input

Pass 1

Output

4,9

7,8

5,6

5,6

1,4

2,4

Input 1

Input 2

Output

2,3

4,6

Input

Pass 1

Output

5,6

5,6

1,4

2,4

Input 1

4,9

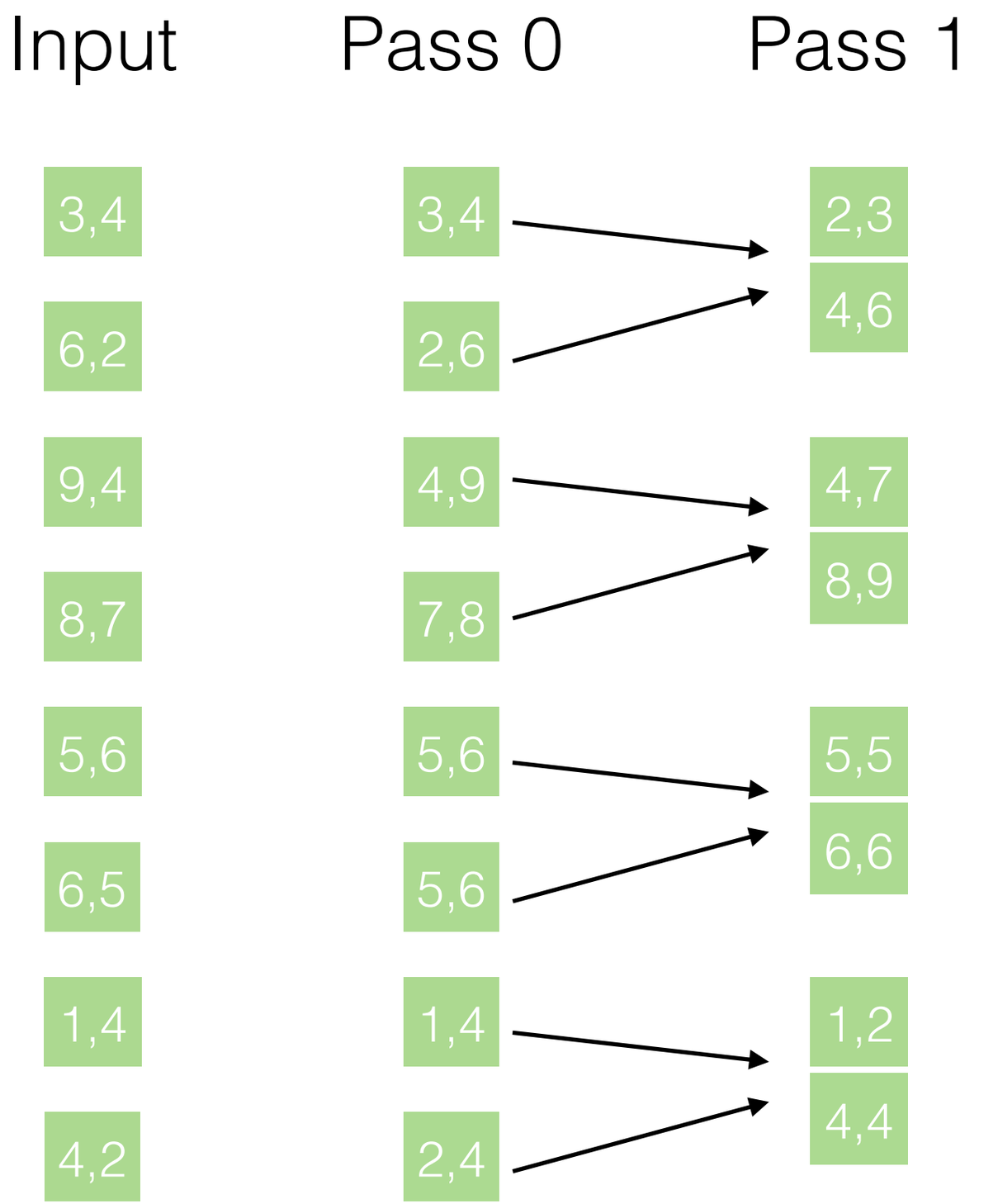
Input 2

7,8

Output

2,3

4,6


Input
Pass 0
Pass 1
3,4
6,2
9,4
8,7
5,6
6,5
1,4
4,2
3,4
2,6
4,9
7,8
5,6
5,6
1,4
2,4
2,3
4,6
4,7
8,9
5,5
6,6
1,2
4,4
1 page runs
2 page runs

Pass 2

Input

Output

2,3

4,6

4,7

8,9

5,5

6,6

1,2

4,4

Input 1

Output

Input 2

Pass 2

Input

Output

4,6

8,9

5,5

6,6

1,2

4,4

Input 1

2,3

Input 2

4,7

Output

Input

Pass 2

Output

4,6

8,9

5,5

6,6

1,2

4,4

Input 1

Output

2,3

Input 2

4,7

Input

Pass 2

Output

2,3

8,9

5,5

6,6

1,2

4,4

Input 1

4,6

Output

Input 2

4,7

Pass 2

Input

8,9

5,5

6,6

1,2

4,4

Input 1

6

Input 2

7

Output

4,4

Output

2,3

Pass 2

Input

8,9

5,5

6,6

1,2

4,4

Input 1

6

Input 2

7

Output

Output

2,3

4,4

Input

Pass 2

Output

2,3

4,4

Input 1

Output

6,7

Input 2

8,9

5,5

6,6

1,2

4,4

Pass 2

Input

8,9

5,5

6,6

1,2

4,4

Input 1

Input 2

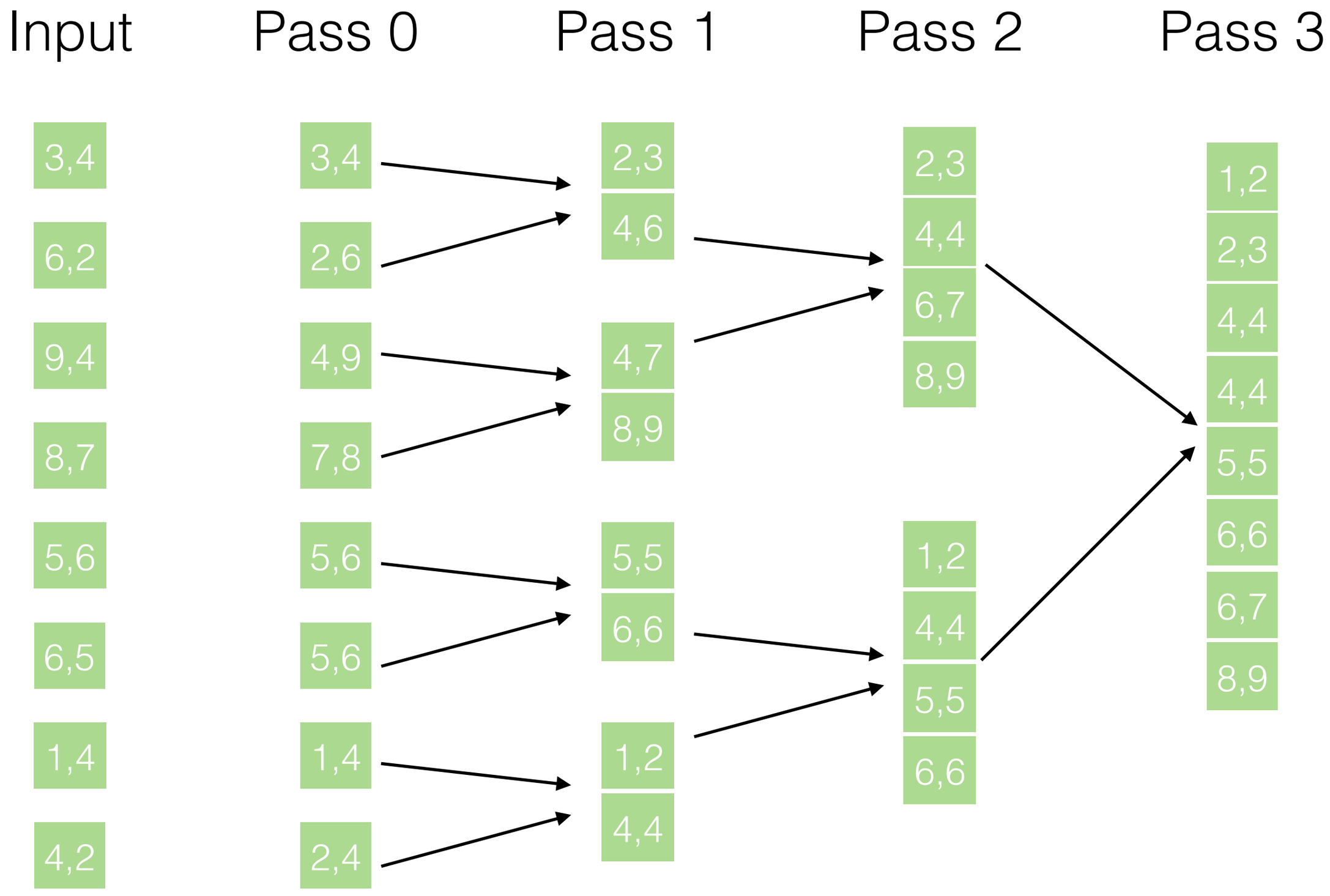
Output

Output

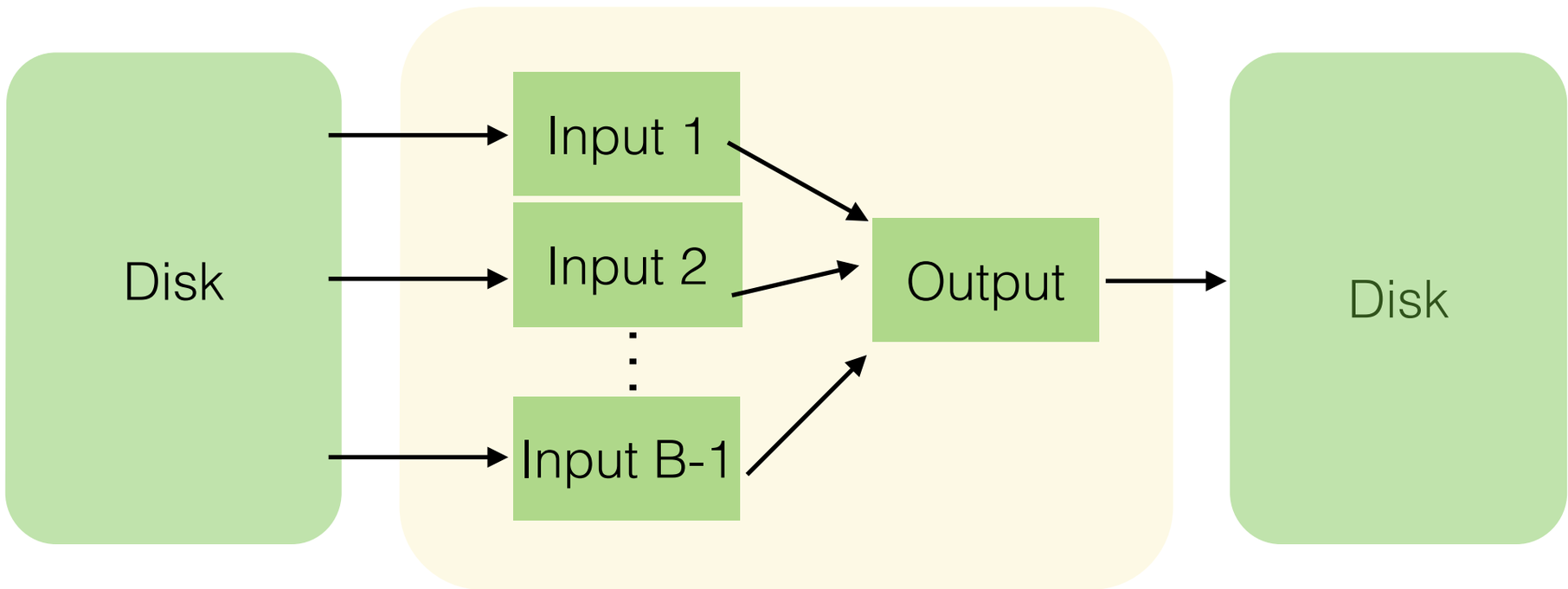
2,3

4,4

6,7


Input
Pass 0
Pass 1
Pass 2
Pass 3
3,4
6,2
9,4
8,7
5,6
6,5
1,4
4,2
3,4
2,6
4,9
7,8
5,6
5,6
1,4
2,4
2,3
4,6
4,7
8,9
5,5
6,6
1,2
4,4
2,3
4,4
6,7
8,9
1,2
4,4
5,5
6,6
1,2
2,3
4,4
4,4
5,5
6,6
6,7
8,9
1 page runs
2 page runs
4 page runs
8 page runs

Generalized Merge Sort



Buffer size of B pages

Generalized Merge Sort

- Pass 0: Use all B buffers to sort, giving N/B sorted runs
- Pass 1, 2, …, etc: Merge B-1 runs
- # Passes: `ceil(log_{B-1}(ceil(N/B)) + 1`
- # I/O's: `2N*(ceil(log_{B-1}(ceil(N/B)) + 1)`

Pass 0

Input

Output

3,4

6,2

5,3

9,4

1,2

8,7

3,3

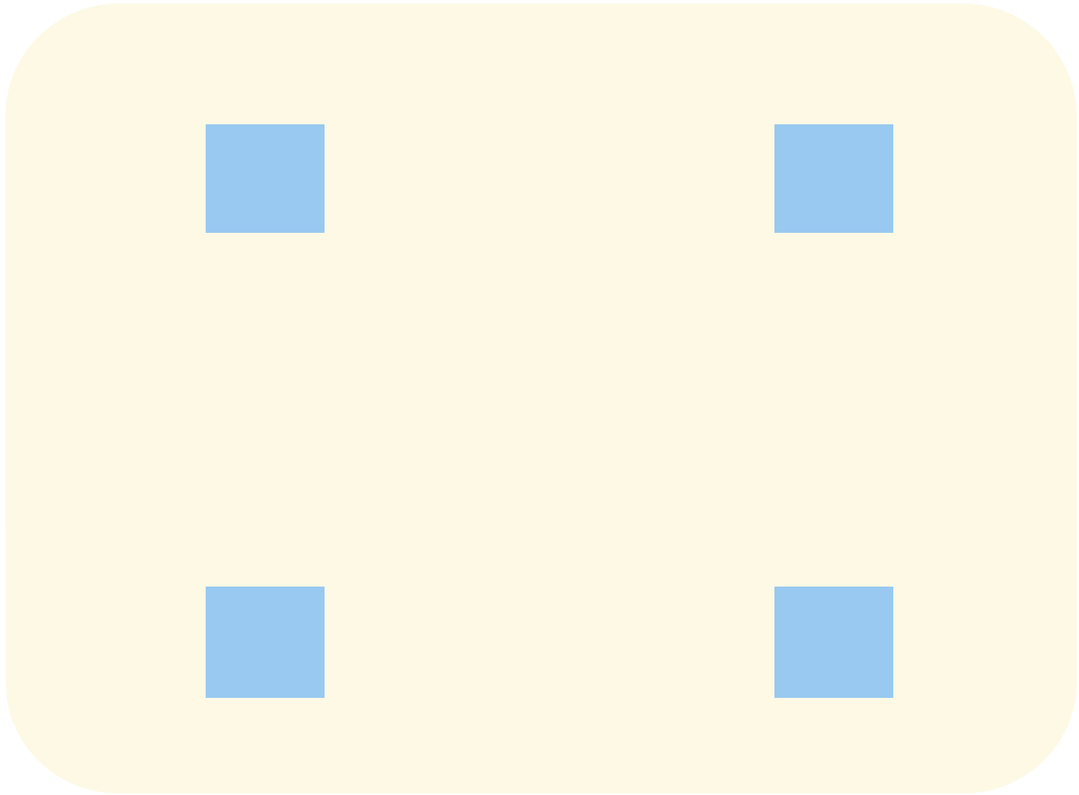
5,6

9,2

6,5

1,4

4,2



B = 4

Input

Pass 0

Output

5,3

1,2

3,3

5,6

9,2

6,5

1,4

4,2

3,4

6,2

9,4

8,7

B = 4

Input

Pass 0

Output

(Use a sort algorithm from 61B!)

5,3

1,2

3,3

9,2

5,6

6,5

1,4

4,2

2,3

4,4

6,7

8,9

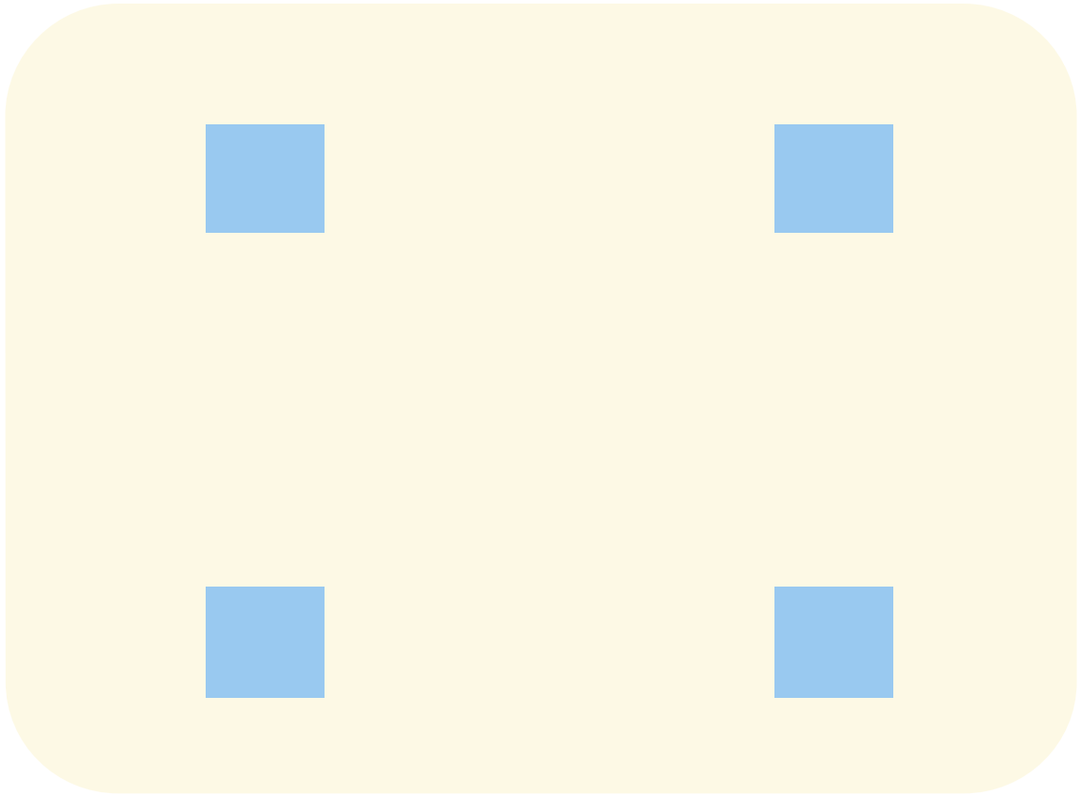
B = 4

Pass 0

Input

Output

(Use a sort algorithm from 61B!)



B = 4

5,3

1,2

3,3

9,2

5,6

6,5

1,4

4,2

2,3

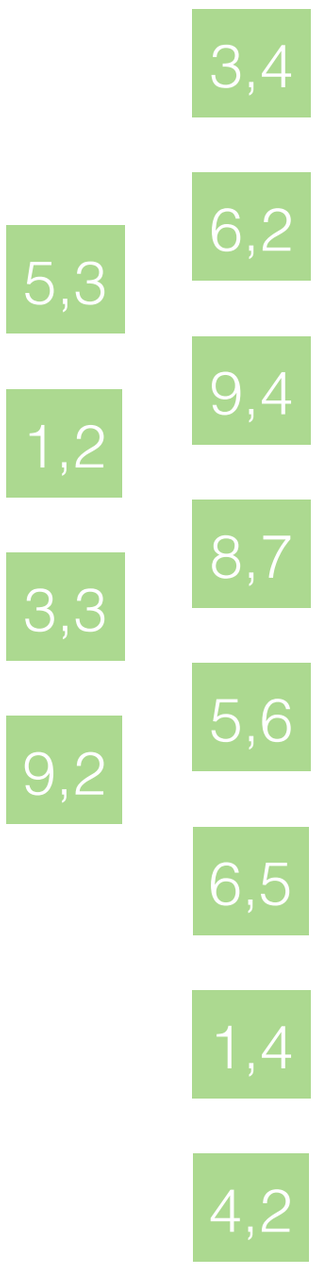
4,4

6,7

8,9

Input

Pass 0



4 page runs

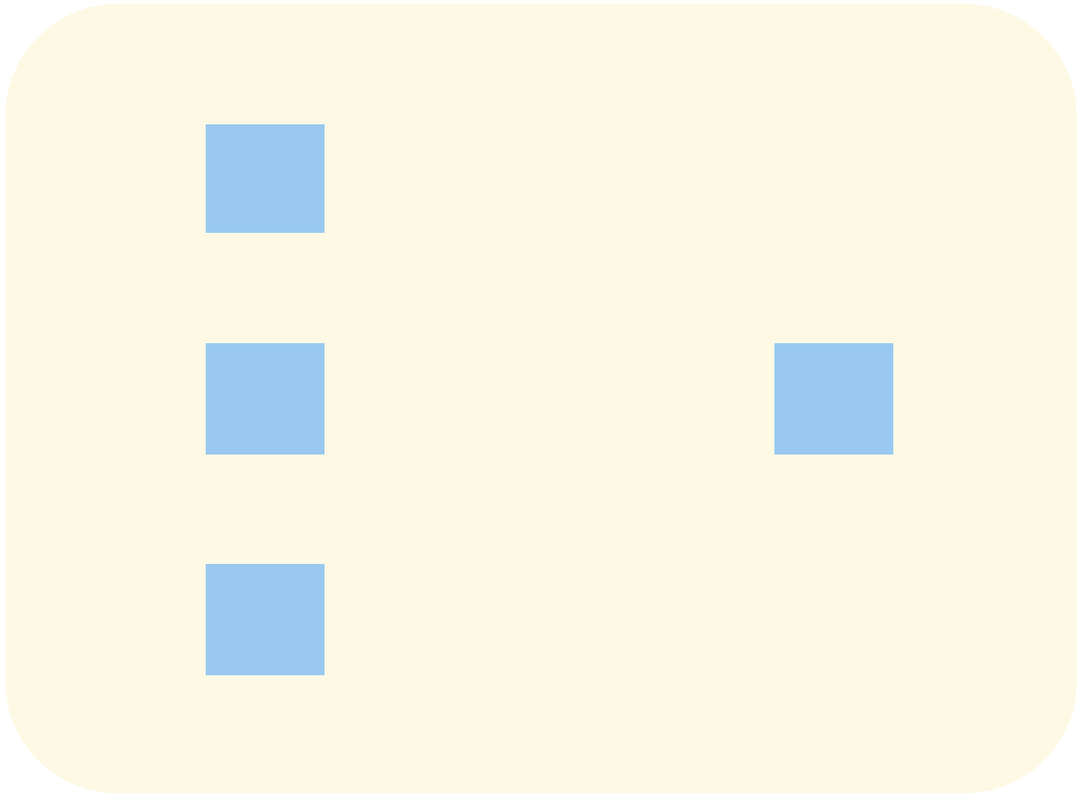
Pass 1

Input

Output

1,2
2,3
3,3
5,9

2,3
4,4
6,7
8,9
1,2
4,4
5,5
6,6



B = 4

Pass 1

Input

Output

4,4

6,7

8,9

2,3

3,3

5,9

4,4

5,5

6,6

2,3

1,2

1,2

B = 4

Pass 1

Input

Output

4,4

6,7

8,9

2,3

3,3

5,9

4,4

5,5

6,6

2,3

2

1,1

2

B = 4

Pass 1

Input

4,4

6,7

8,9

2,3

3,3

5,9

4,4

5,5

6,6

2,3

2

2

Output

1,1

B = 4

Pass 1

Input

4,4

6,7

8,9

2,3

3,3

5,9

4,4

5,5

6,6

2,3

2

2

B = 4

Output

1,1

Pass 1

Input

4,4

6,7

8,9

2,3

3,3

5,9

5,5

6,6

Output

1,1

2,3

4,4

2

2

B = 4

Pass 1

Input

4,4

6,7

8,9

2,3

3,3

5,9

5,5

6,6

3

4,4

2,2

2

Output

1,1

B = 4

Pass 1

Input

4,4

6,7

8,9

2,3

3,3

5,9

5,5

6,6

3

4,4

2

B = 4

Output

1,1

2,2

Pass 1

Input

4,4

6,7

8,9

3,3

5,9

5,5

6,6

3

4,4

2

2,3

Output

1,1

2,2

B = 4

Generalized Merge Sort

- Pass 0: Use all B buffers to sort, giving N/B sorted runs
- Pass 1, 2, …, etc: Merge B-1 runs
- # Passes: `ceil(log_{B-1}(ceil(N/B)) + 1`
- # I/O's: `2N*(ceil(log_{B-1}(ceil(N/B)) + 1)`

Worksheet #3, 4

List the differences between 2-way external merge sort and general external merge sort.

List the differences between 2-way external merge sort and general external merge sort.

- 2-way only utilizes 2 input buffers, general utilizes B-1
- During pass 0, 2-way only uses 1 page to sort files. General external merge sort uses all B pages in its buffer to sort the initial runs.

Buffer: 640 KB
Page: 64 KB

How many pages can your buffer hold?

Buffer: 640 KB
Page: 64 KB

How many pages can your buffer hold?

- 640 KB * (1 page / 64 KB) = **10 pages**

Buffer: 640 KB
Page: 64 KB

How many pages are in a 4 MB file?

Buffer: 640 KB
Page: 64 KB

How many pages are in a 4 MB file?

- 4 MB * (1024 KB / 1MB) * (1 page / 64 KB) = **64 pages**

Buffer: 10 Pages
File: 64 Pages

How many passes would it take to externally merge sort a 4 MB file?

Buffer: 10 Pages
File: 64 Pages

How many passes would it take to externally merge sort a 4 MB file?

- ceil(log_{10-1} ceil(64 / 10)) + 1 = ceil(log_9 (7)) + 1 = 1 + 1 = 2 passes

Buffer: 10 Pages
File: 64 Pages

How many I/O's are needed to to externally merge sort a 4 MB file?

Buffer: 10 Pages
File: 64 Pages

How many I/O's are needed to to externally merge sort a 4 MB file?

- (# of passes) * 2 * (# of pages in file) = 2 * 2 * 64 = 256 I/O's

Buffer: 10 Pages
File: 64 Pages

What is the maximum file size that can be sorted with just 2 passes in this system?

Buffer: 10 Pages
File: 64 Pages

What is the maximum file size that can be sorted with just 2 passes in this system?

- (# of buffer pages) (# of buffer pages - 1) = 10 * 9 = 90 pages = 5760 KB ~= 5.6 MB

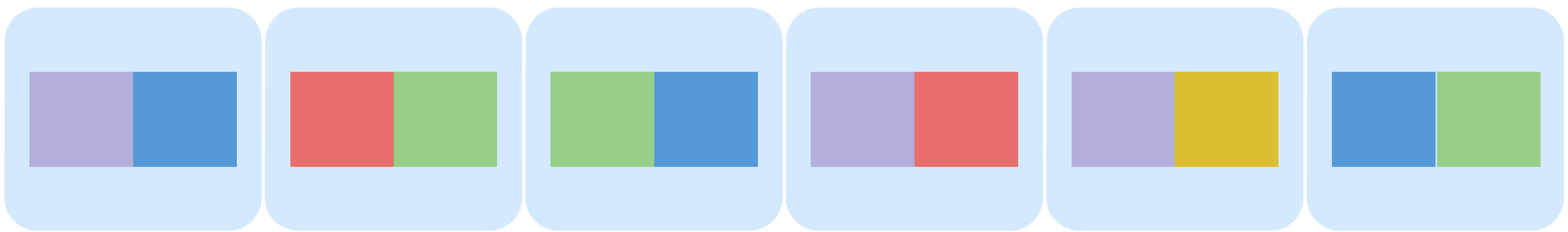
External Hashing

External Hashing

- Want to aggregate data that does not fit in memory
- Minimize number of I/O's (especially random I/O's)

Aggregating Colors

- Goal: Group squares by color
- Setup: 12 squares, 2 can fit per page. We can hold 8 squares in memory.
- N=6, B=4



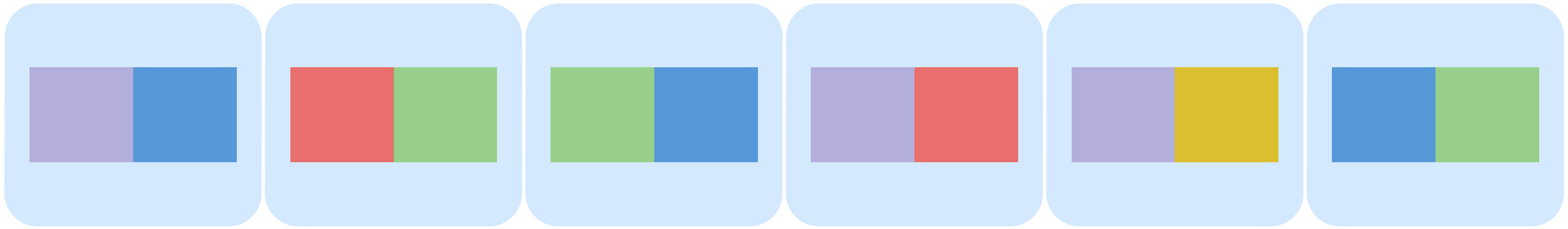
Pass 1: Divide

- Read all pages in, hash to B-1 partitions/buckets so that each group guaranteed to be in same partition.

- May not be a whole partition for each group.

- # I/O's = 2N

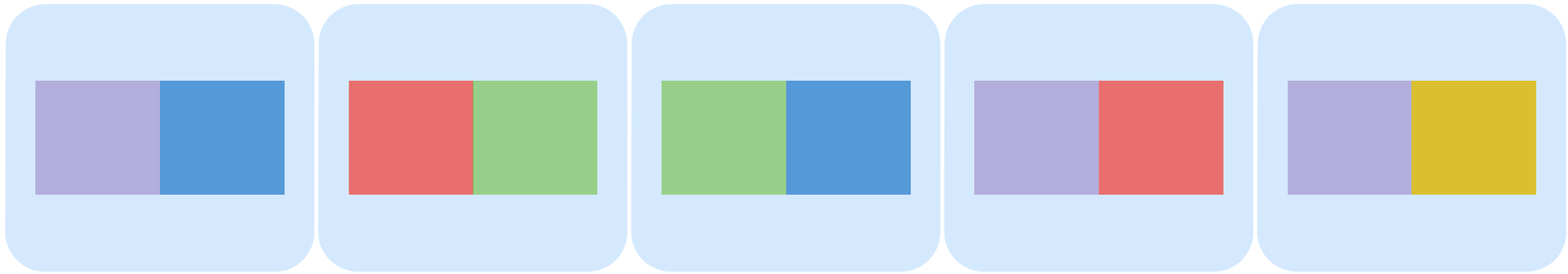
Pass 1: Divide



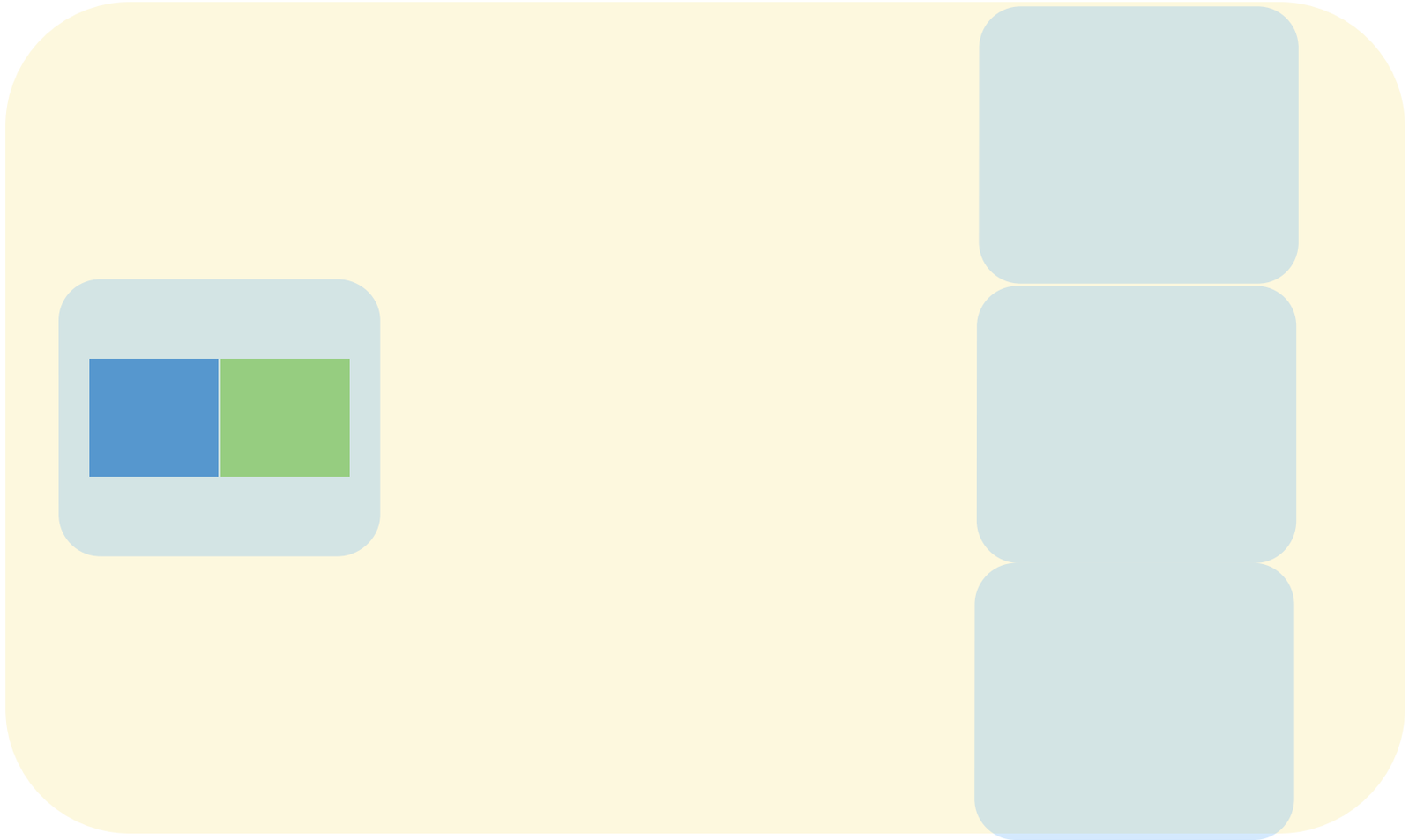
N=6, B=4

Assign colors to 3 partitions using hash function.
Our hash function:
{G,P} -> 1
{B} -> 2
{R, Y} -> 3

Pass 1: Divide

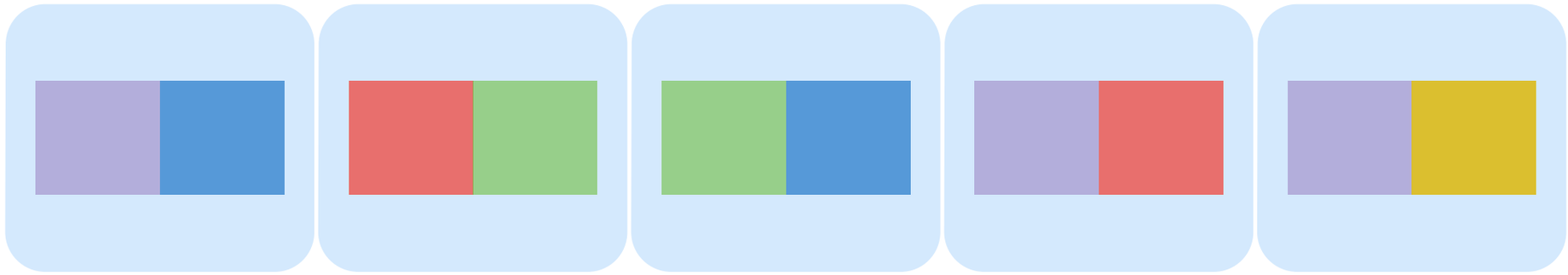


N=6, B=4

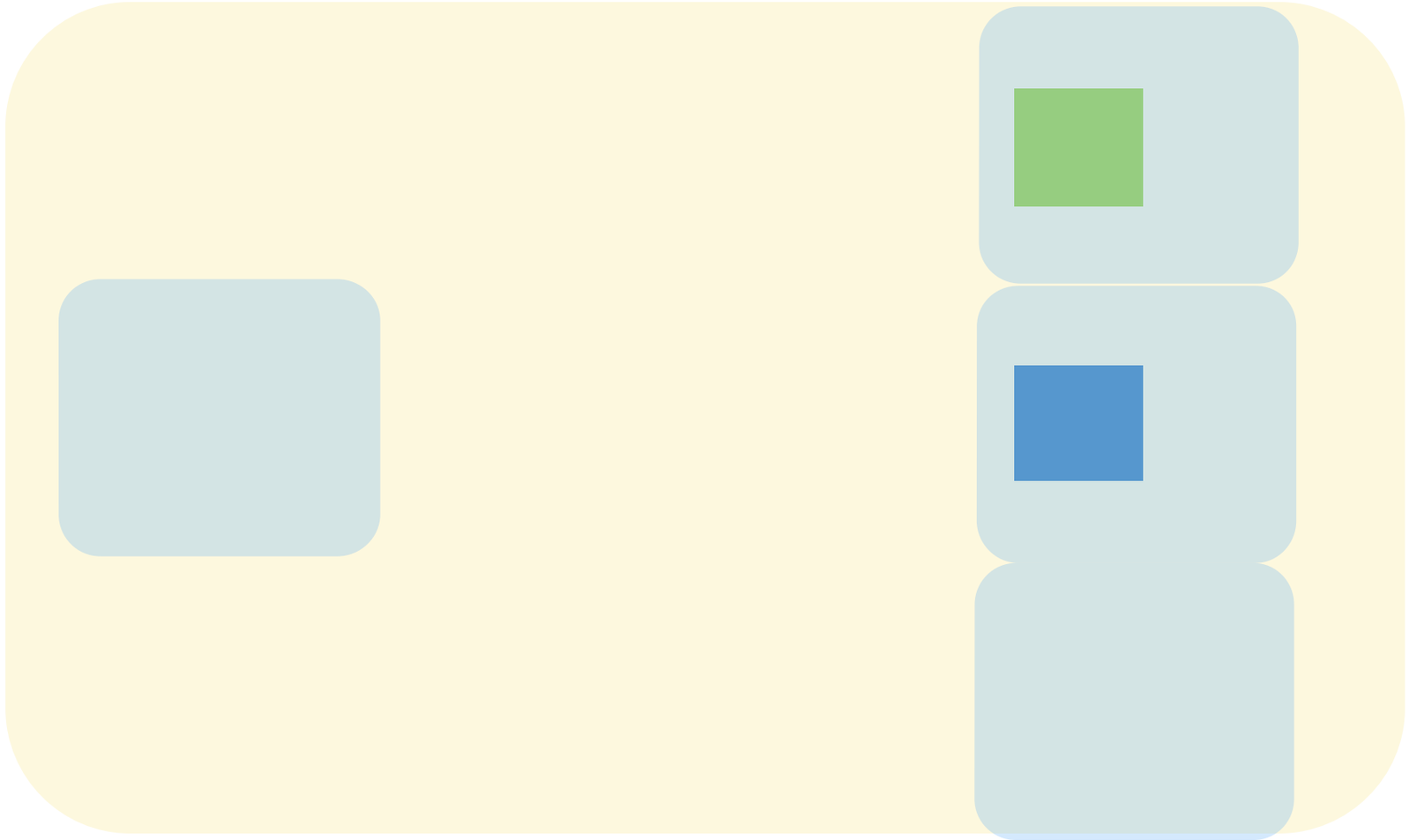


Assign colors to 3 partitions using hash function.
Our hash function:
{G,P} -> 1
{B} -> 2
{R, Y} -> 3

Pass 1: Divide

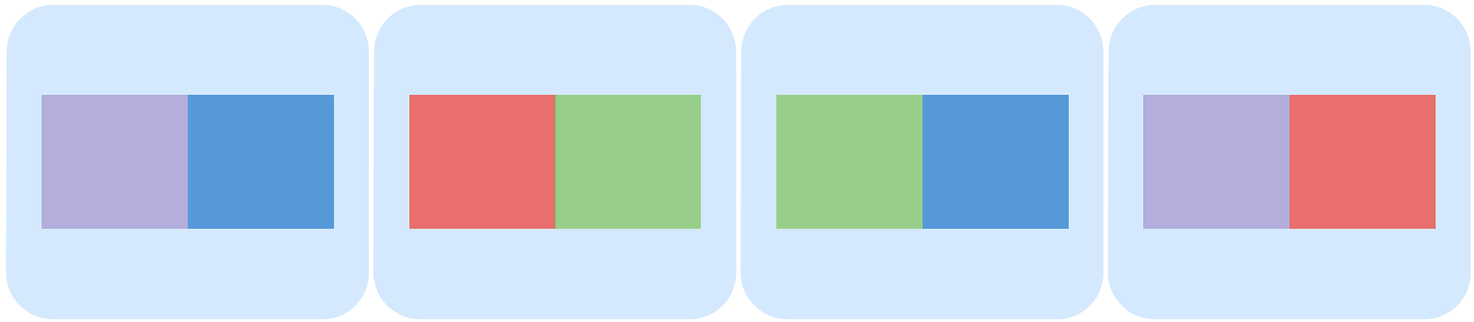


N=6, B=4

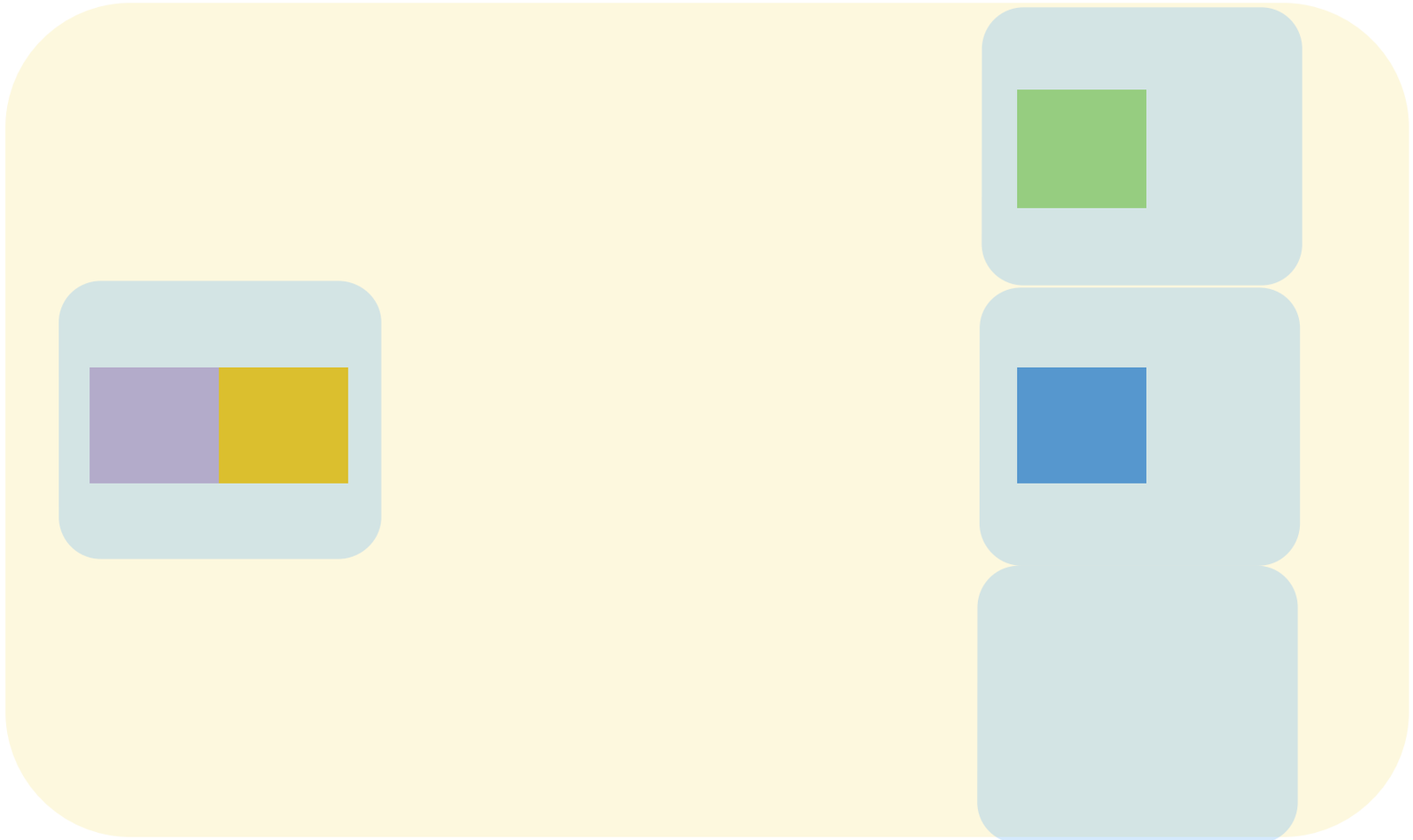


Assign colors to 3 partitions using hash function.
Our hash function:
{G,P} -> 1
{B} -> 2
{R, Y} -> 3

Pass 1: Divide

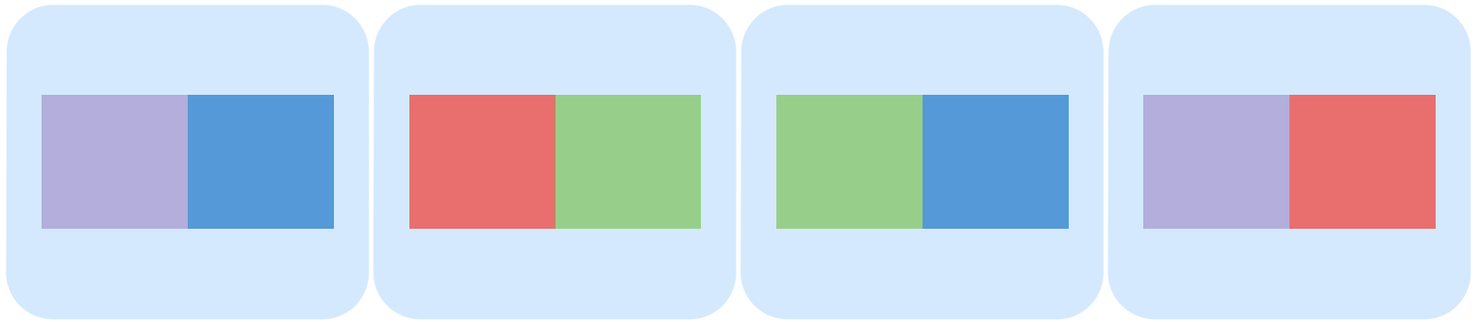


N=6, B=4

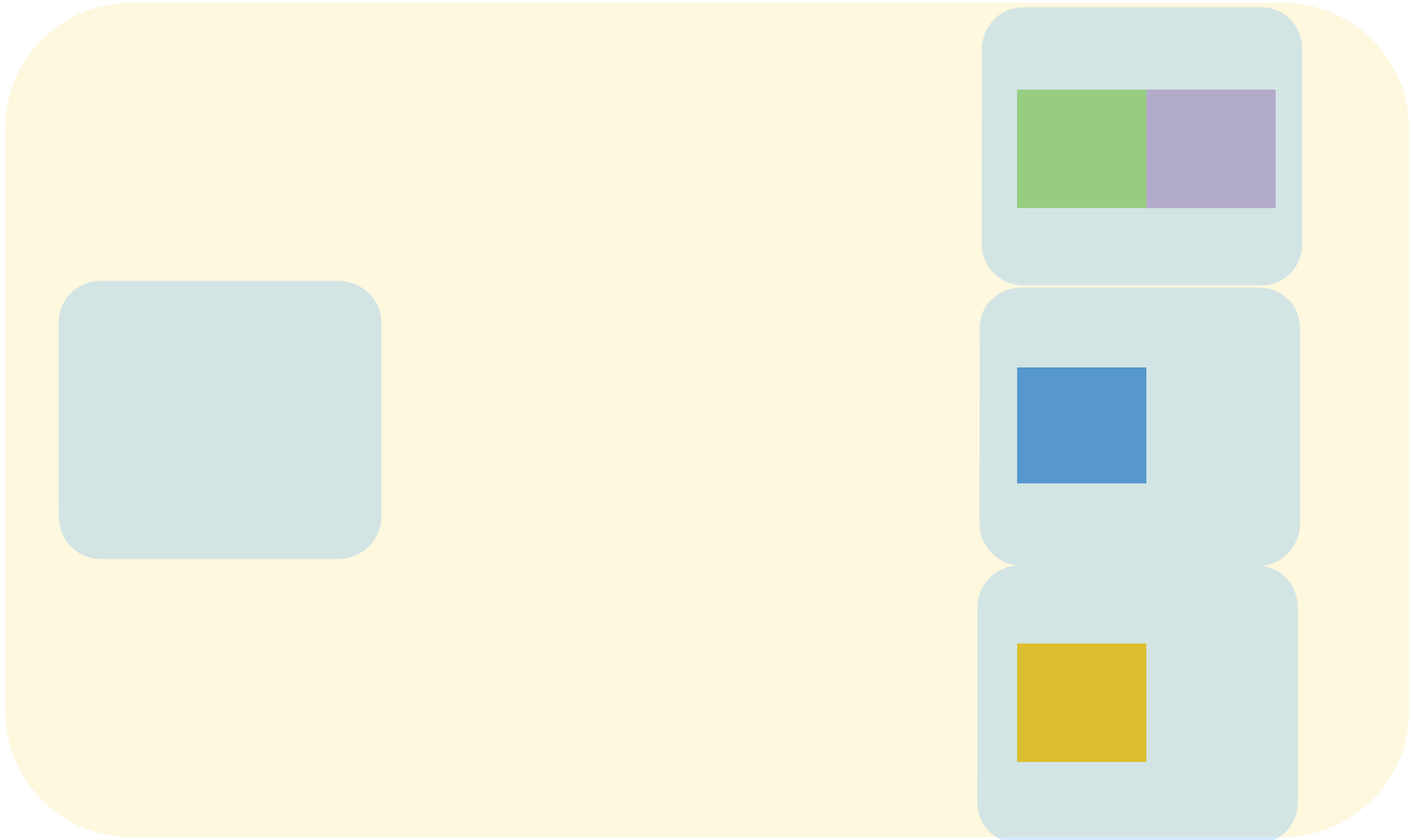


Assign colors to 3 partitions using hash function.
Our hash function:
{G,P} -> 1
{B} -> 2
{R, Y} -> 3

Pass 1: Divide

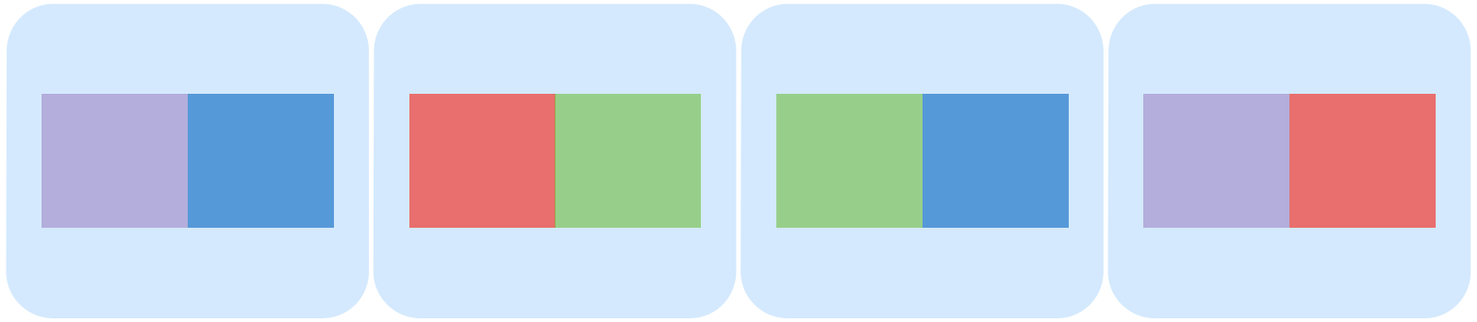


N=6, B=4



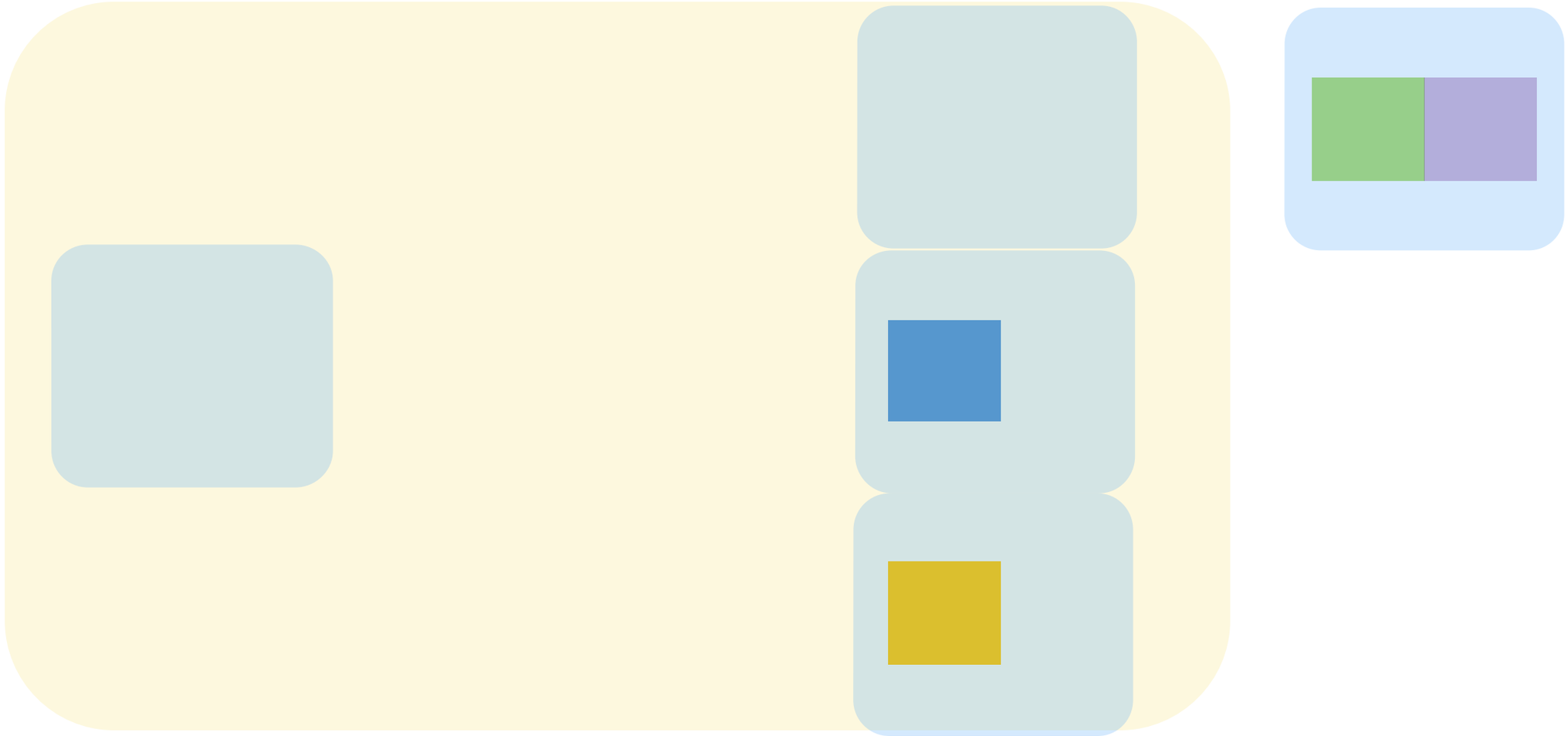
Assign colors to 3 partitions using hash function.
Our hash function:
{G,P} -> 1
{B} -> 2
{R, Y} -> 3

Pass 1: Divide

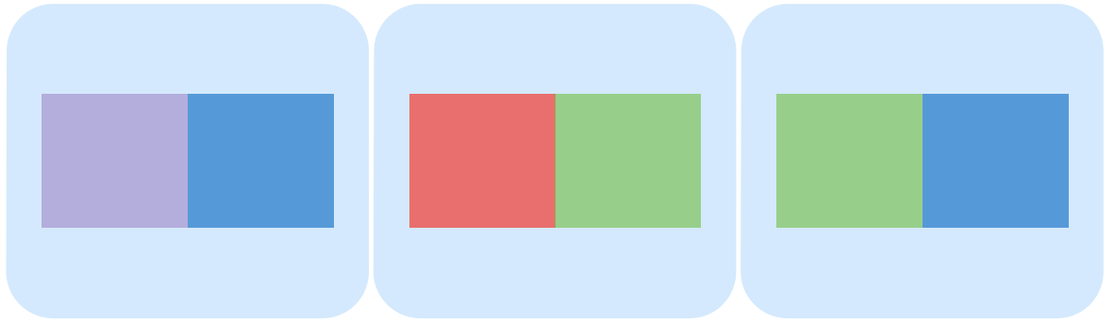


N=6, B=4

Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3

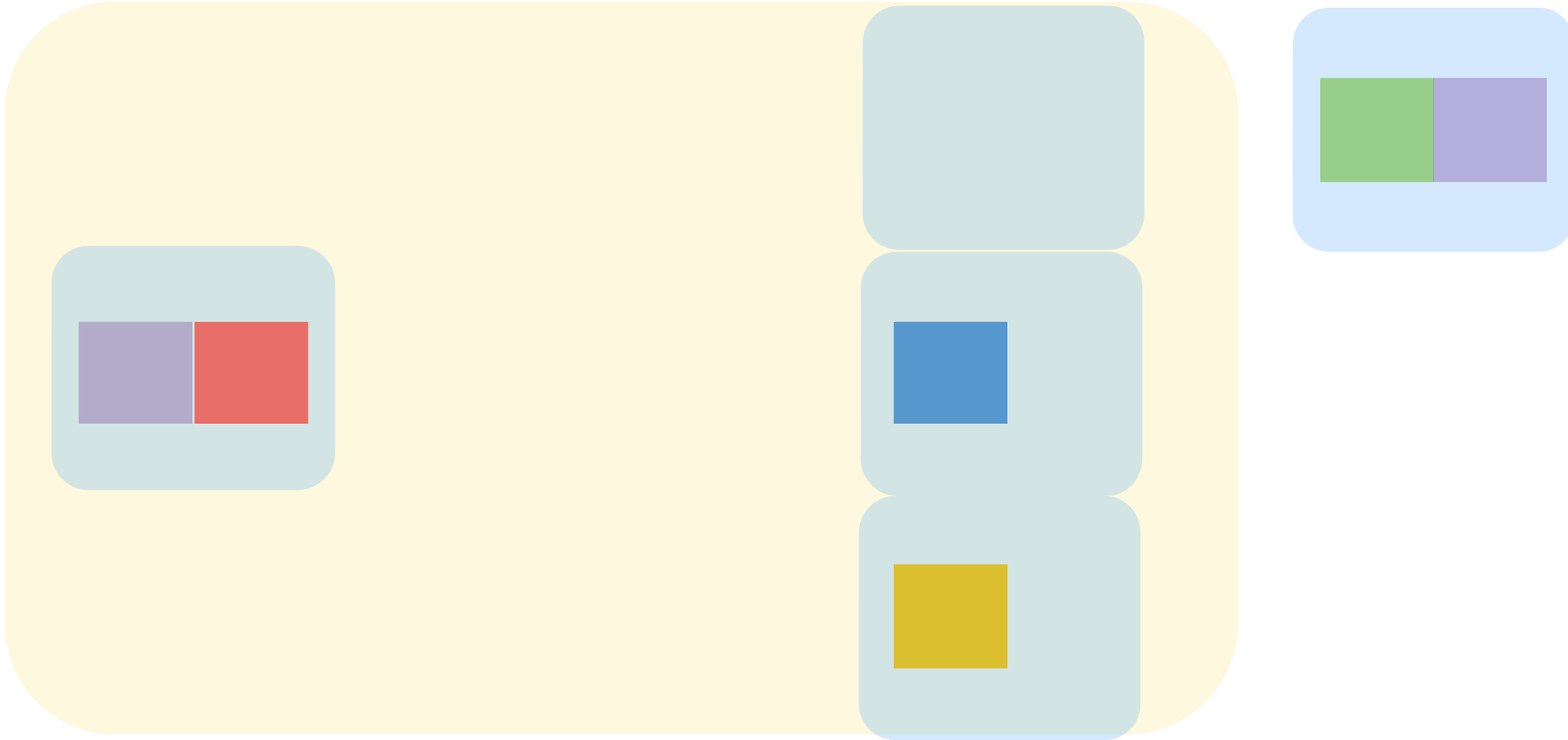


Pass 1: Divide

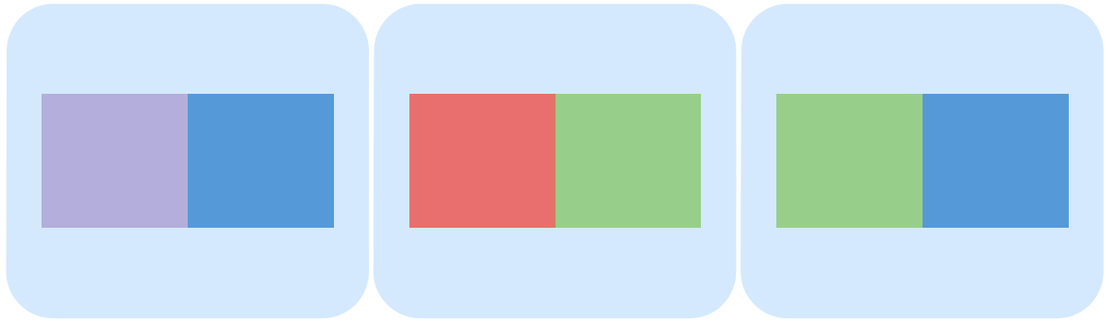


N=6, B=4

Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3

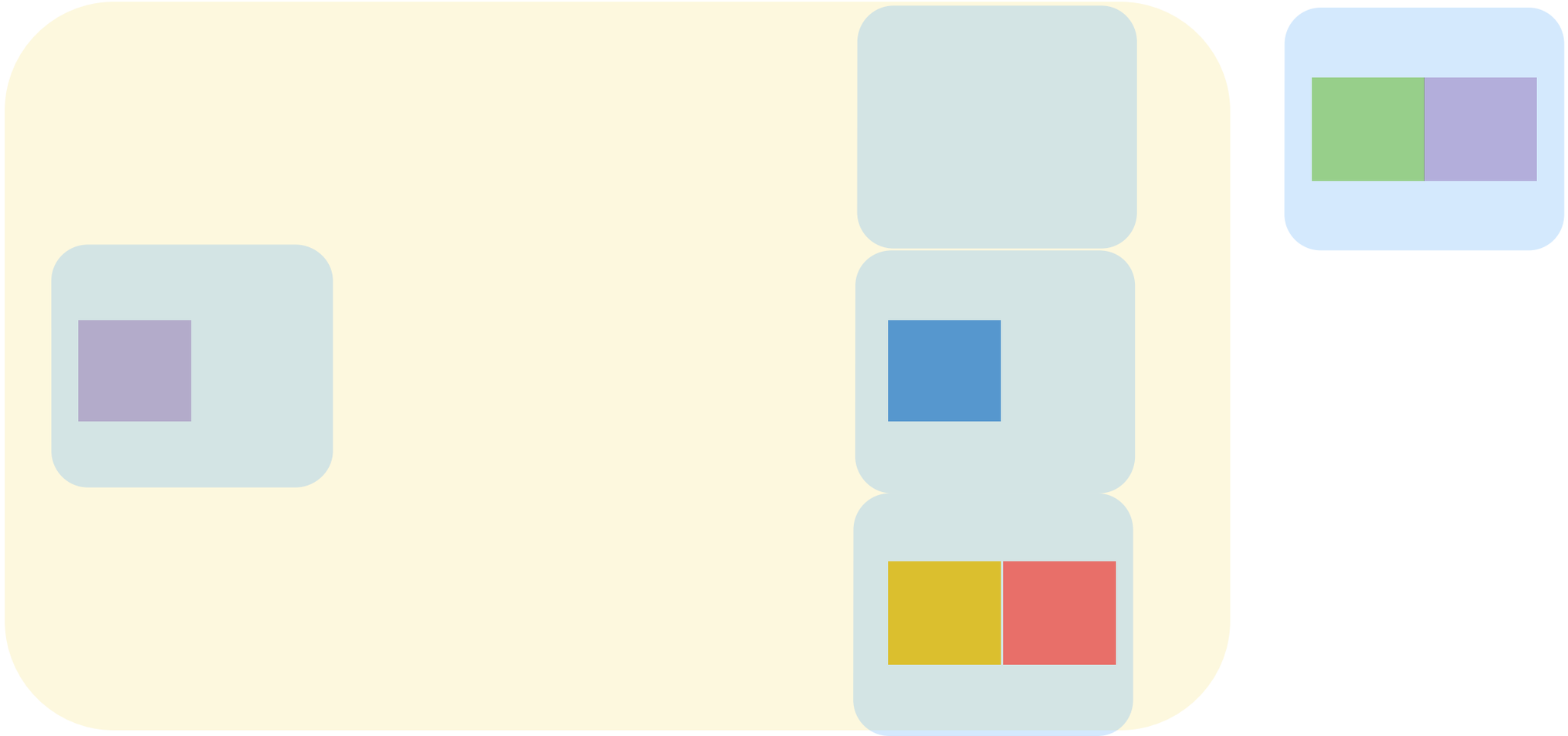


Pass 1: Divide

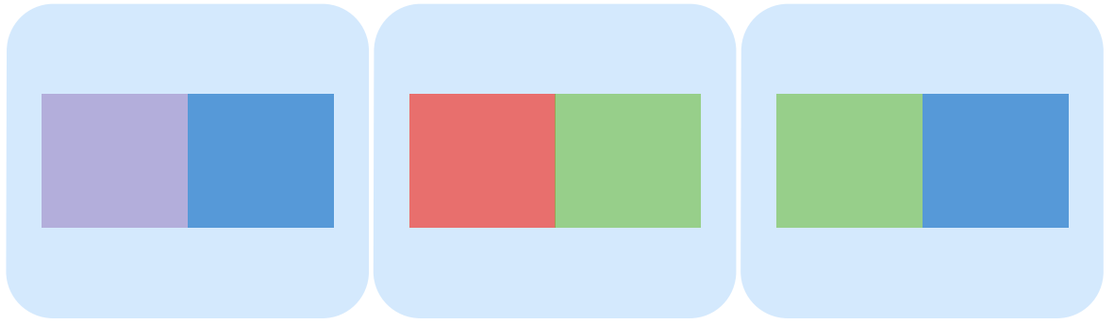


N=6, B=4

Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3

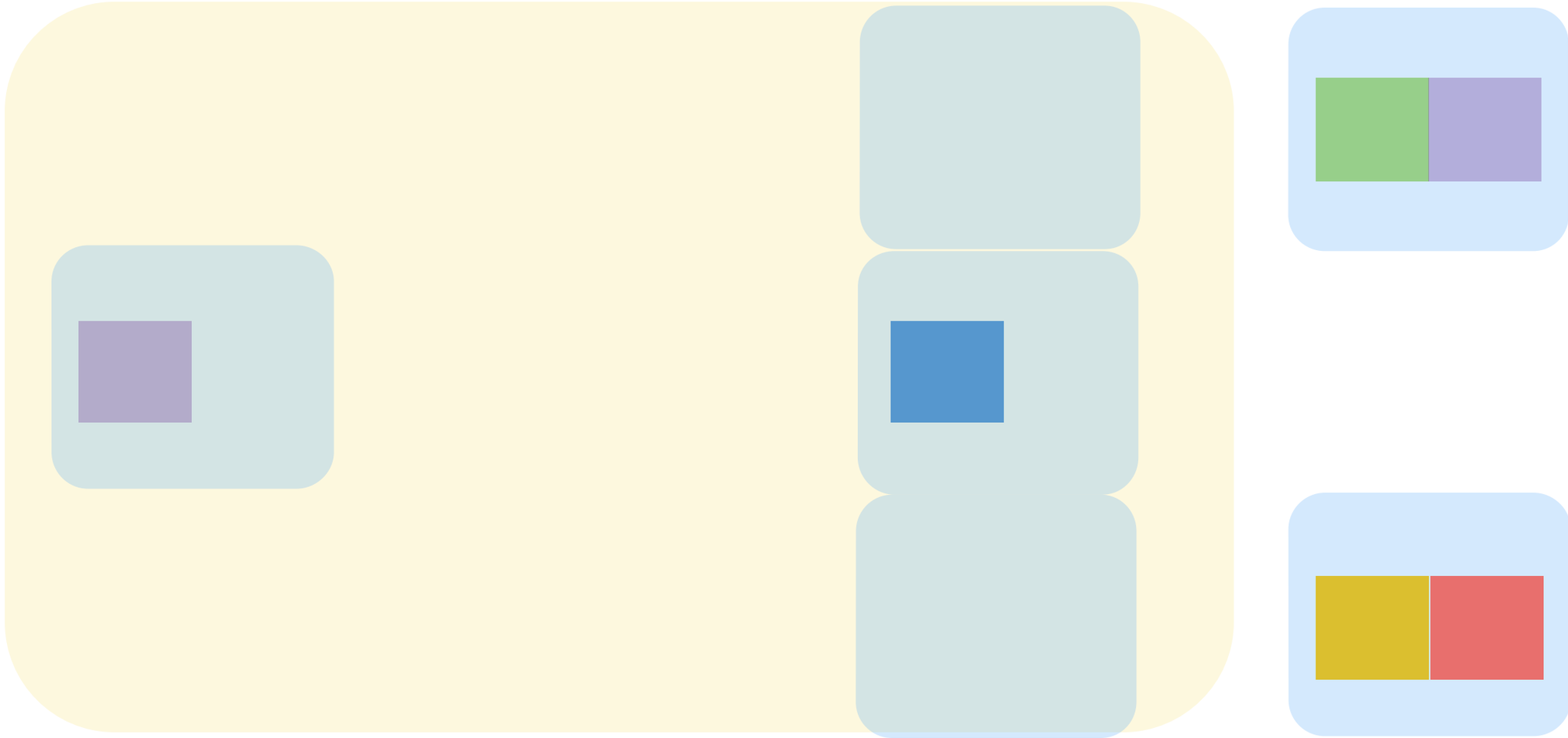


Pass 1: Divide

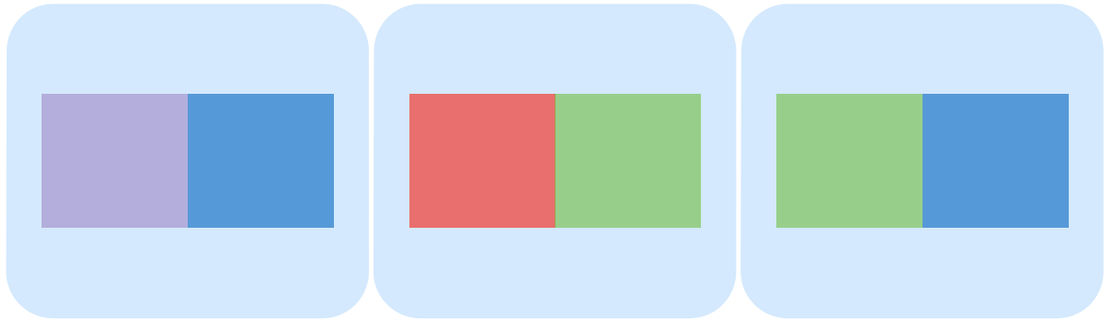


N=6, B=4

Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3

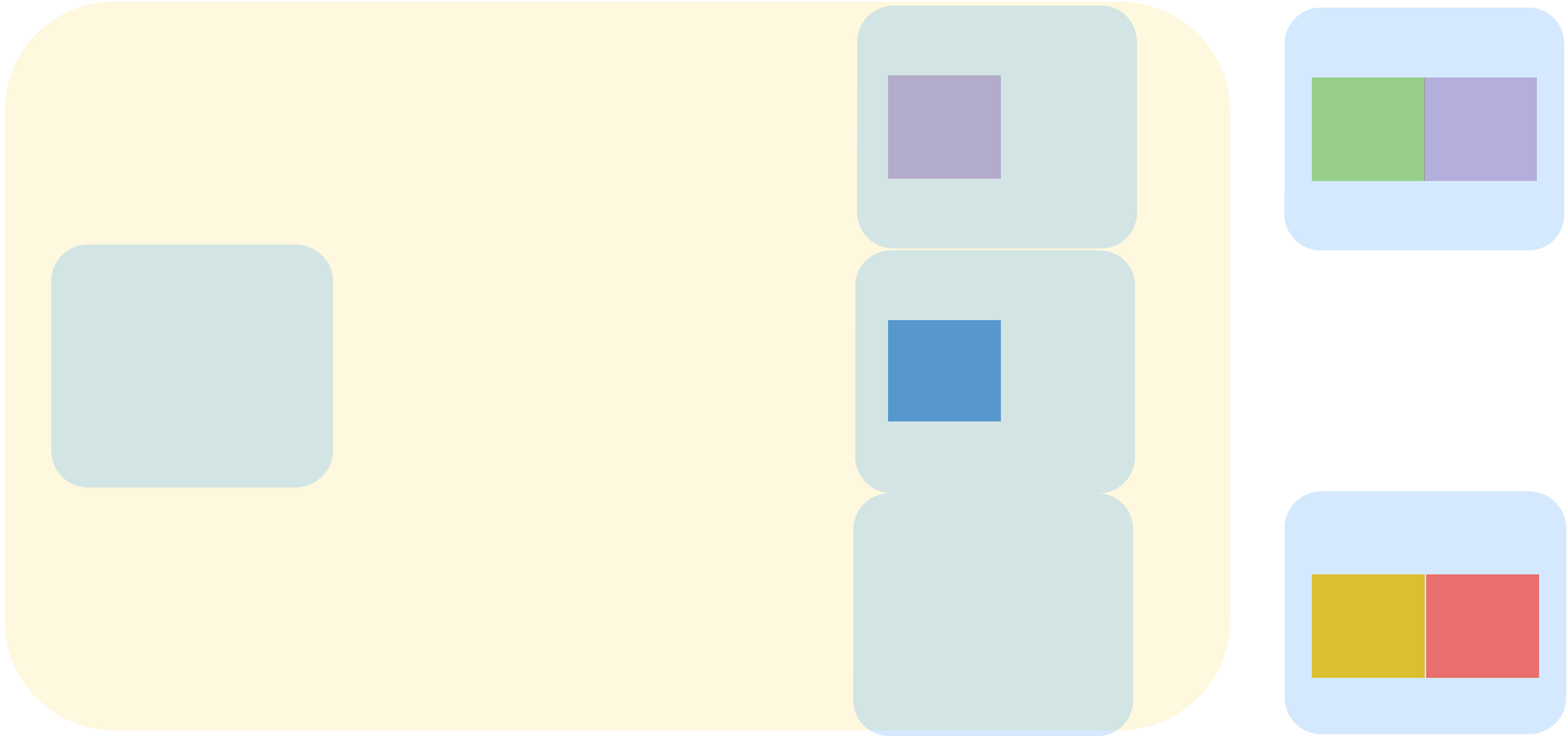


Pass 1: Divide

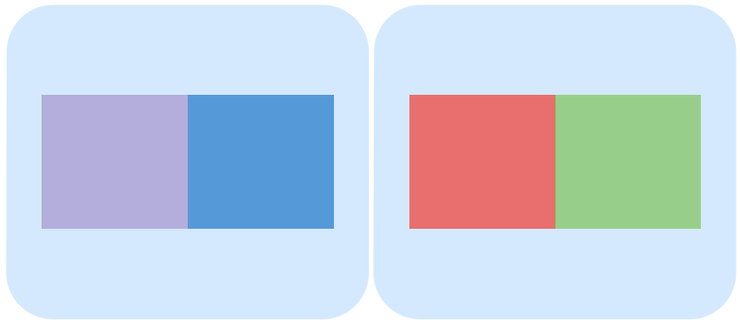


N=6, B=4

Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3

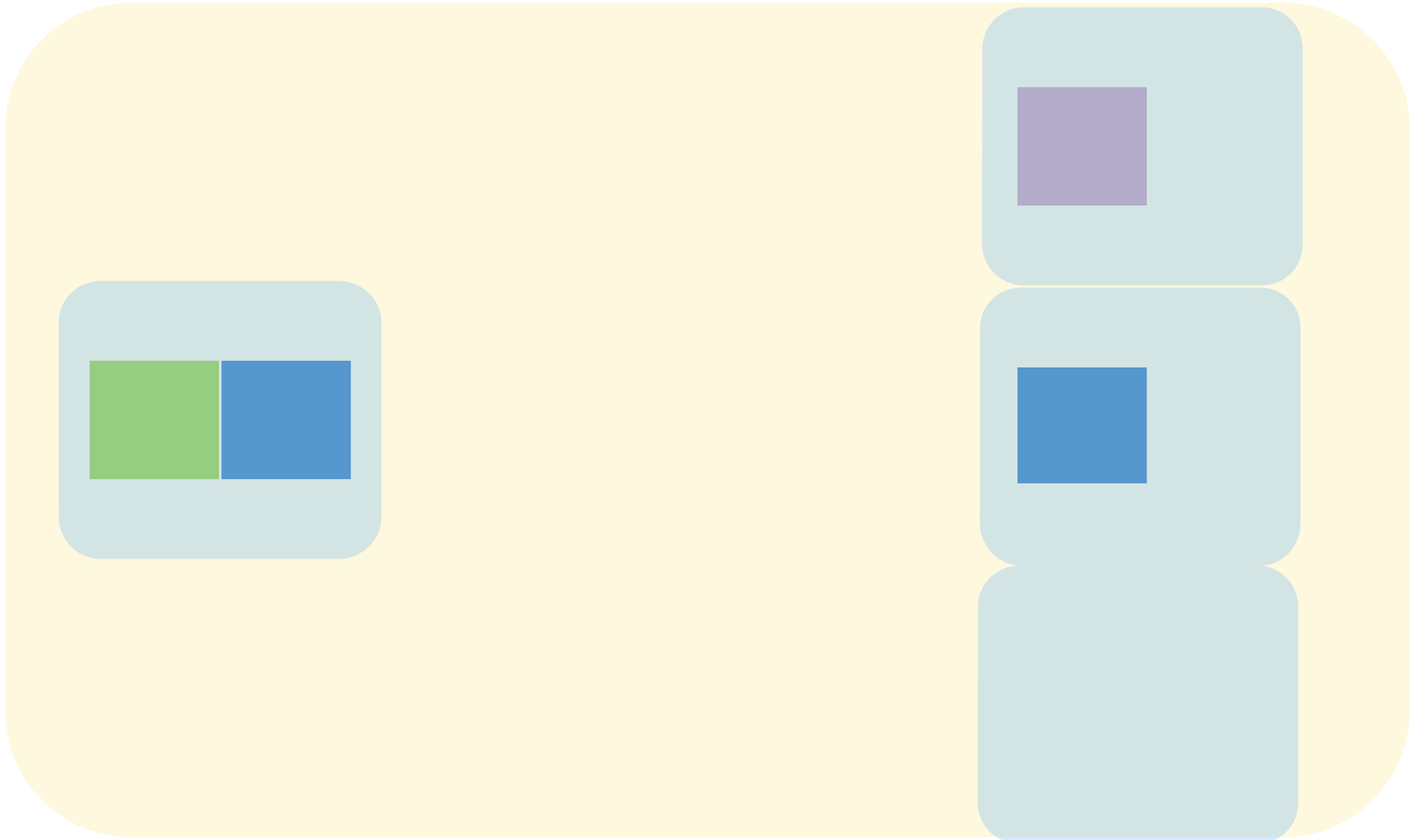


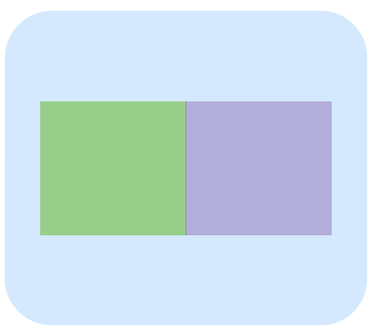
Pass 1: Divide

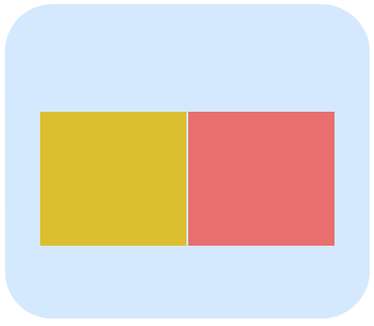


N=6, B=4

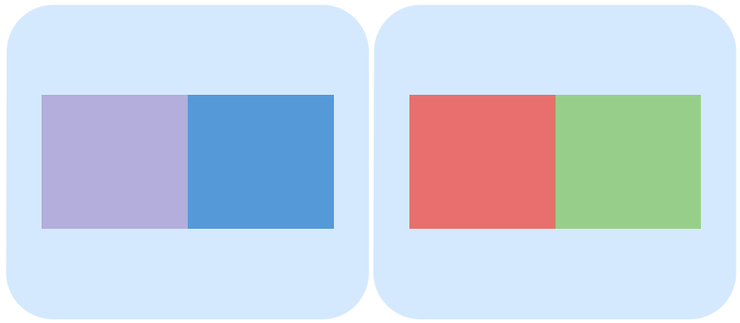
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3





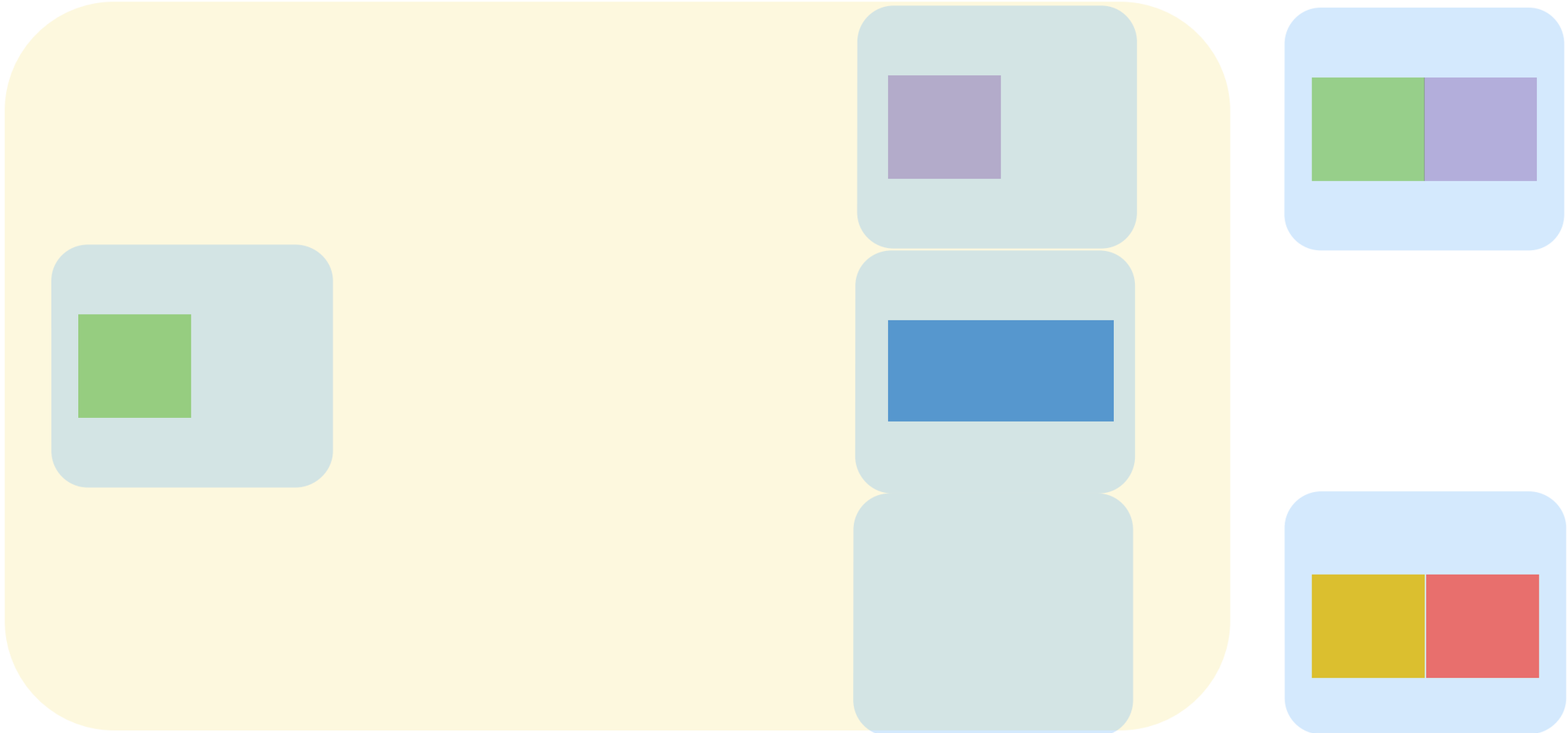


Pass 1: Divide

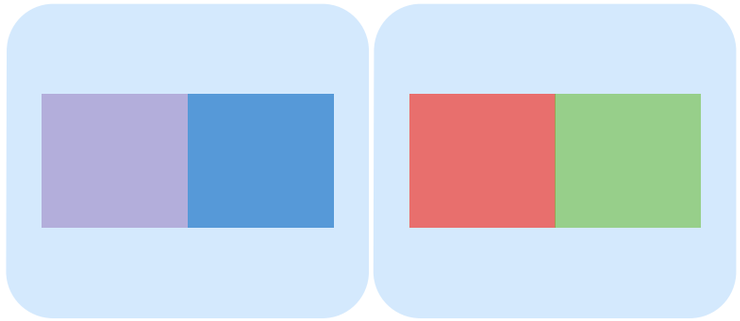


N=6, B=4

Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3

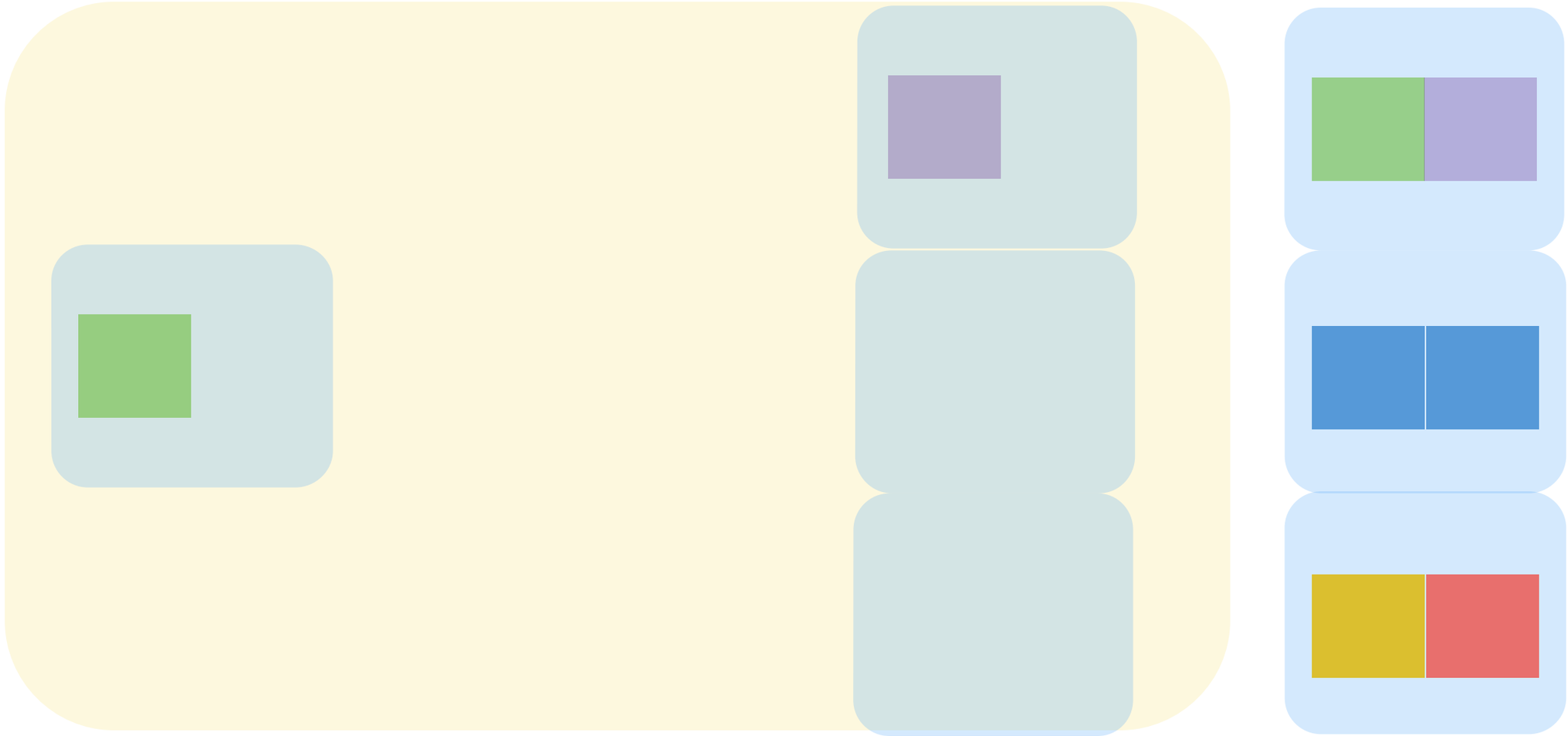


Pass 1: Divide

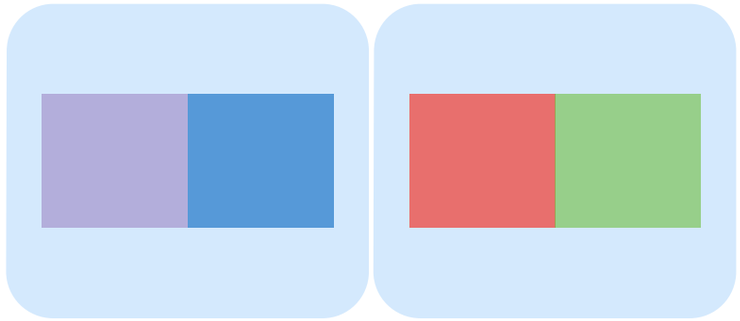


N=6, B=4

Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3

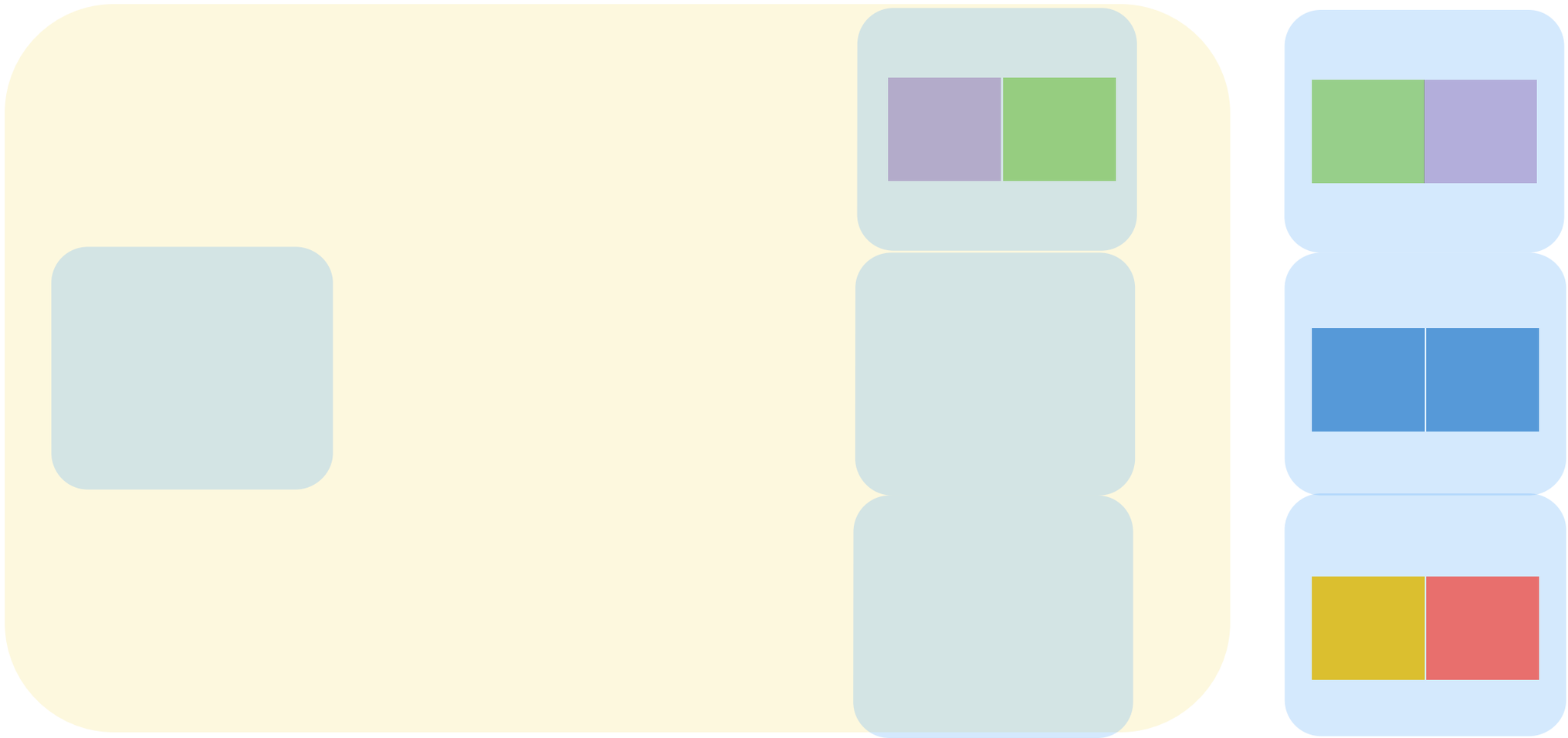


Pass 1: Divide

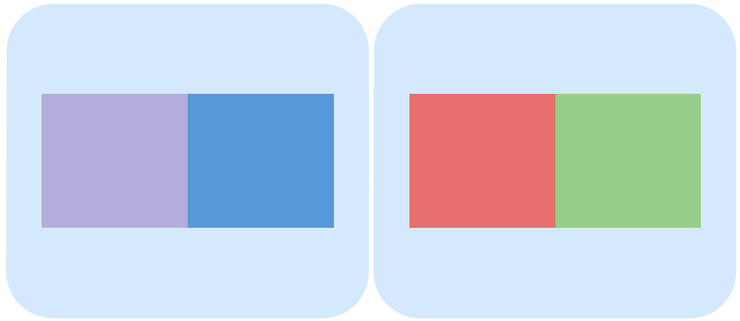


N=6, B=4

Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3

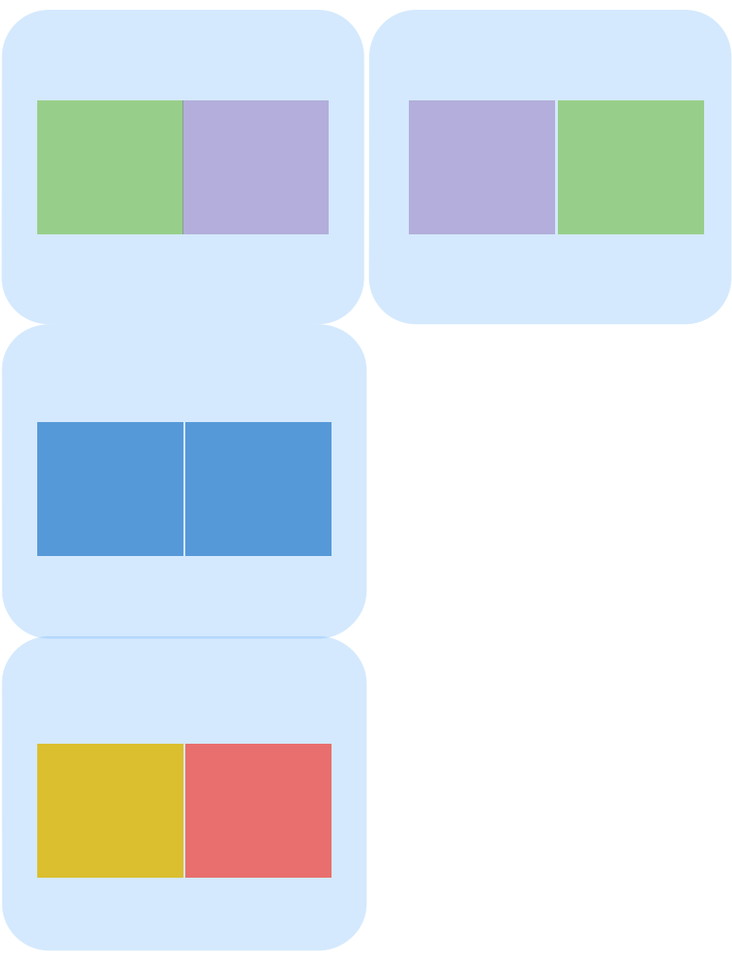


Pass 1: Divide

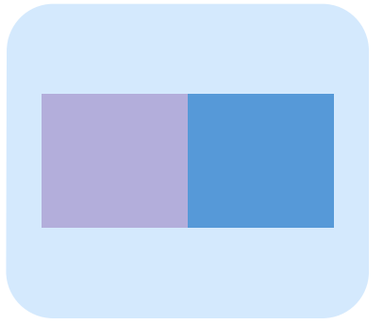


N=6, B=4

Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3

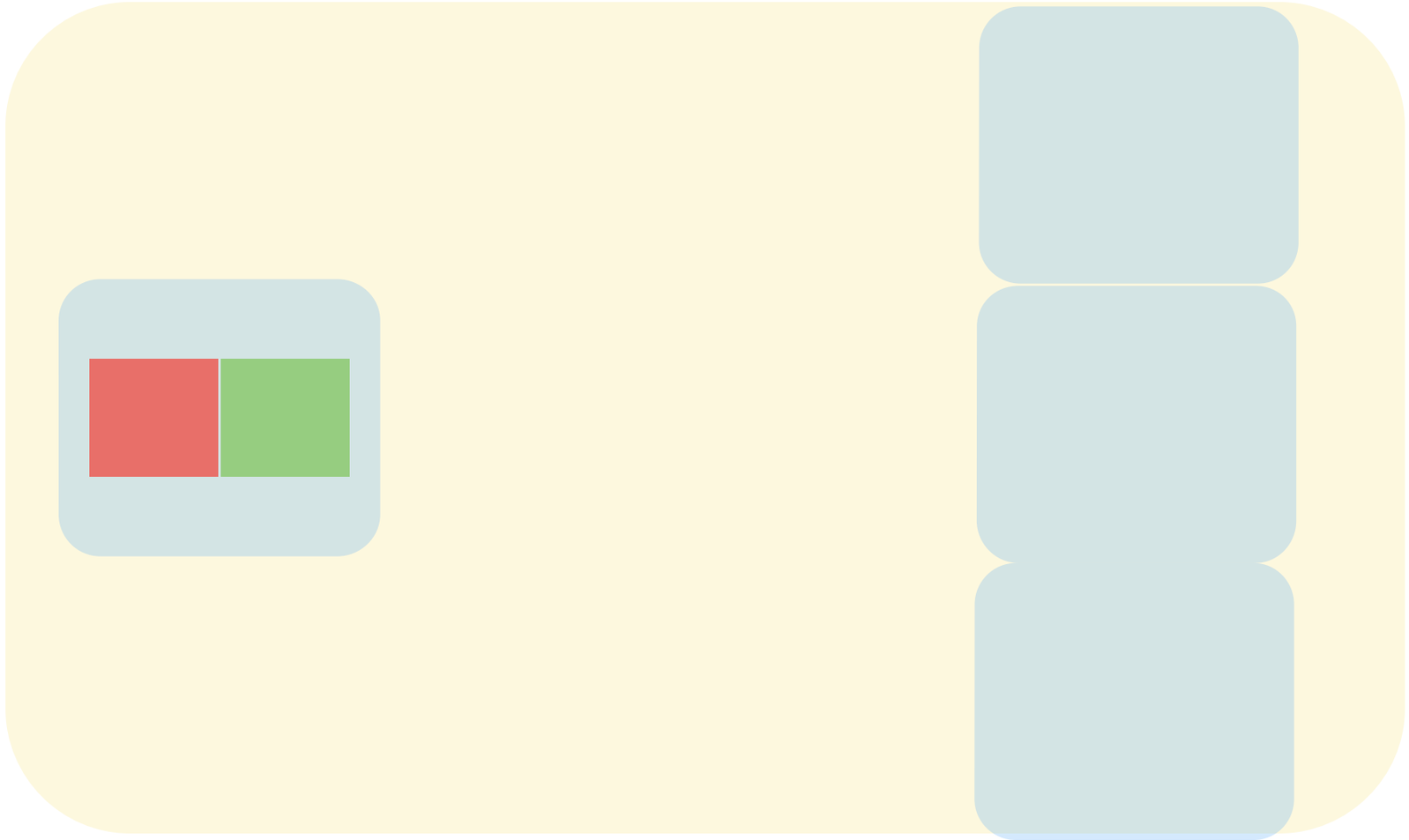


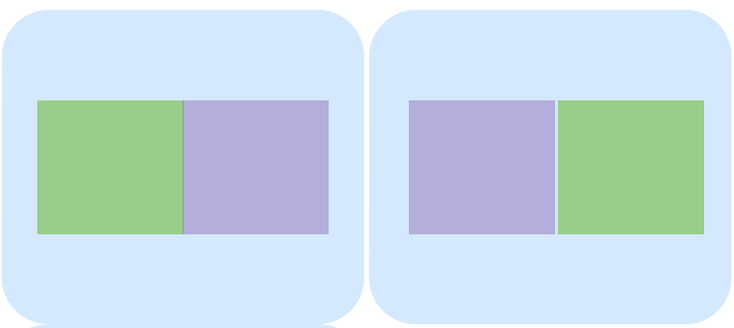
Pass 1: Divide

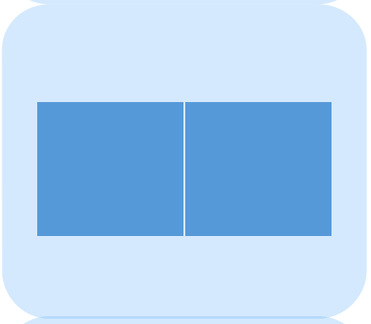


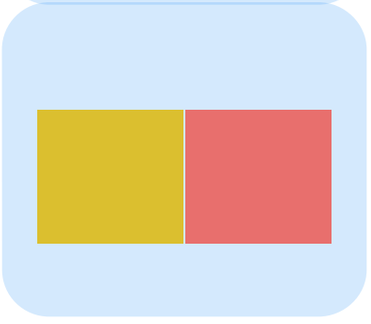
N=6, B=4

Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3

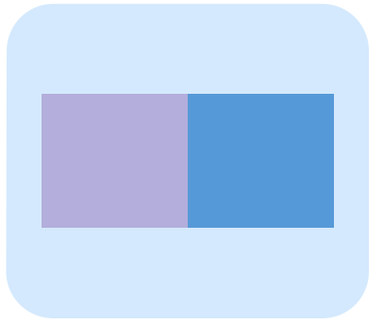






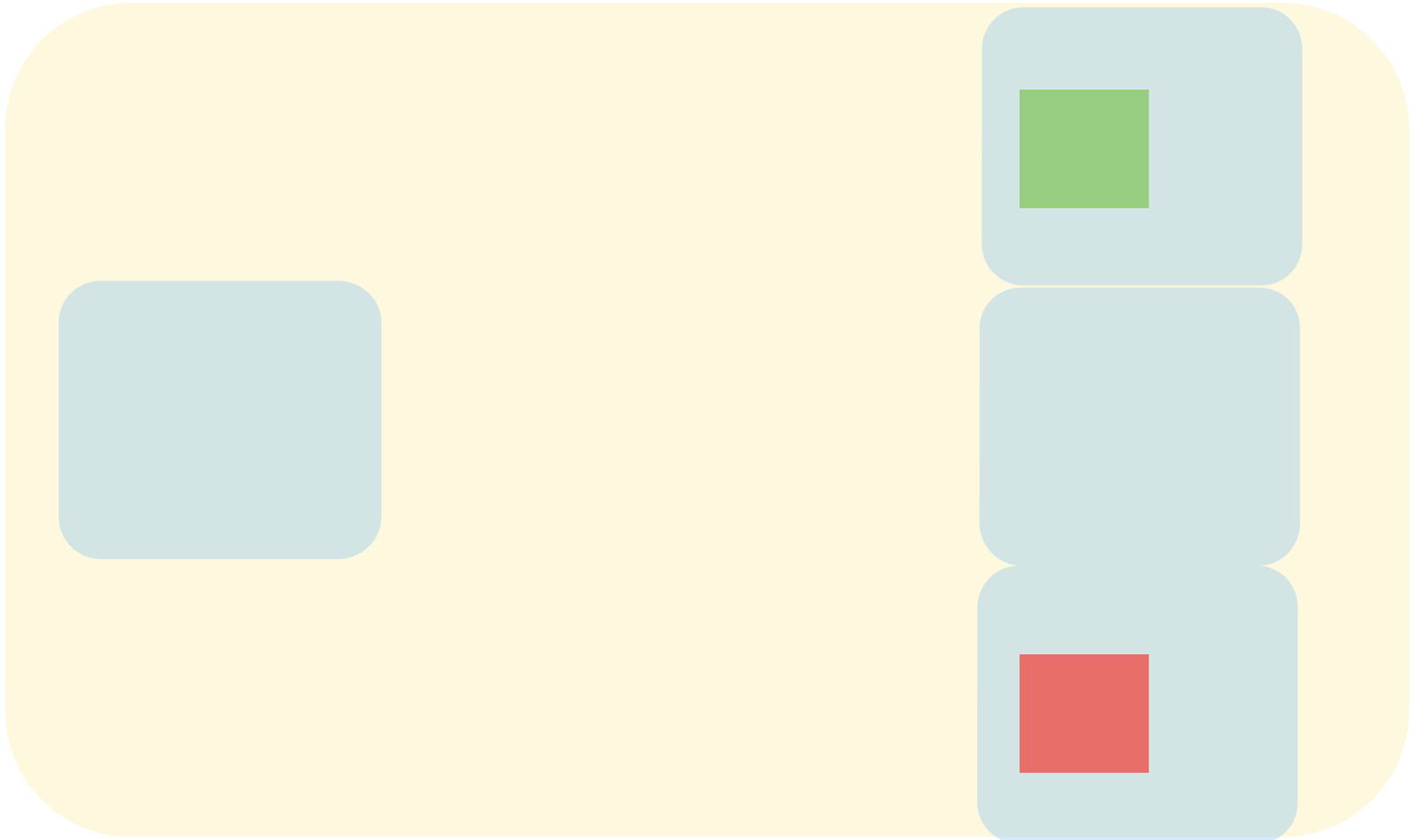


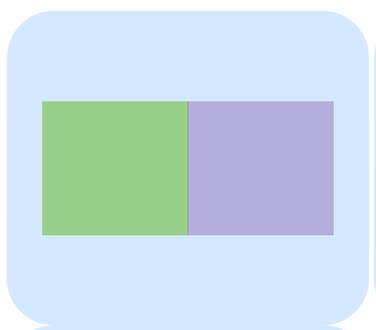
Pass 1: Divide

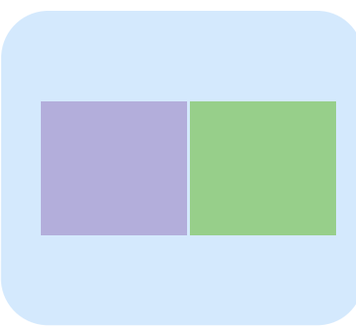


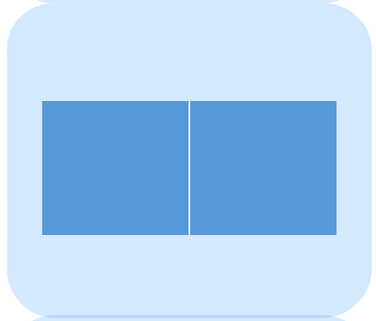
N=6, B=4

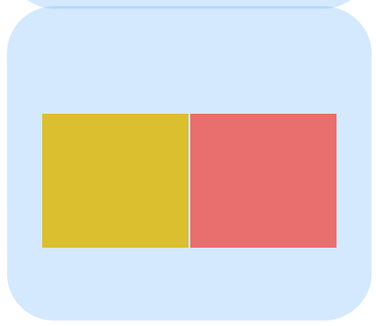
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3







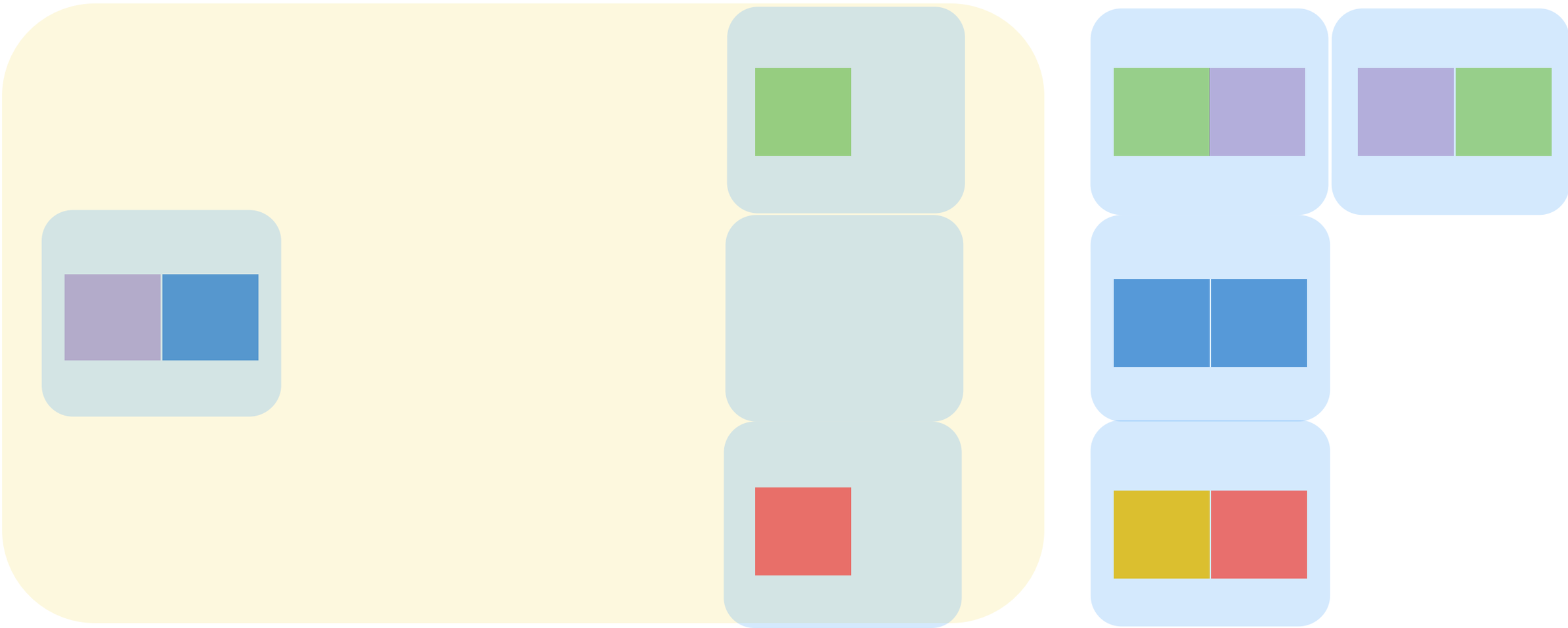




Pass 1: Divide

N=6, B=4

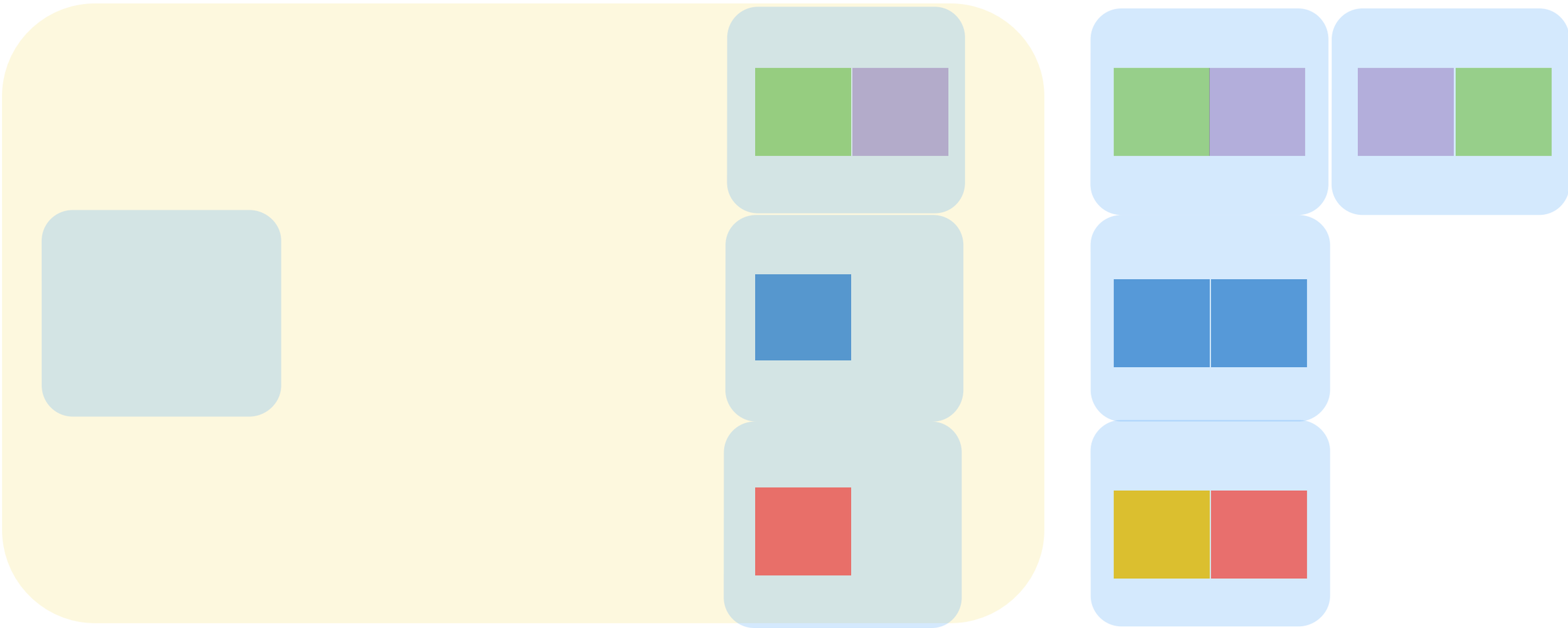
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



Pass 1: Divide

N=6, B=4

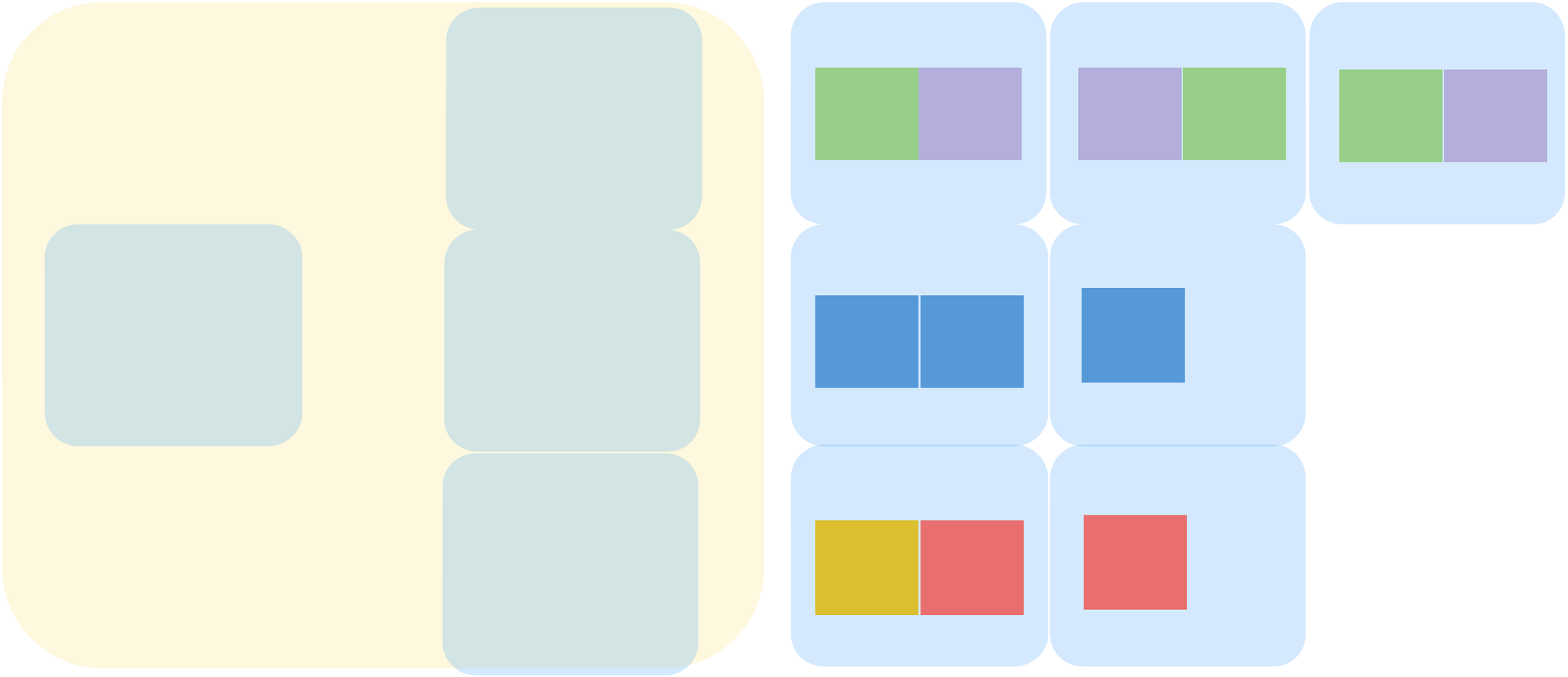
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



Pass 1: Divide

N=6, B=4

Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



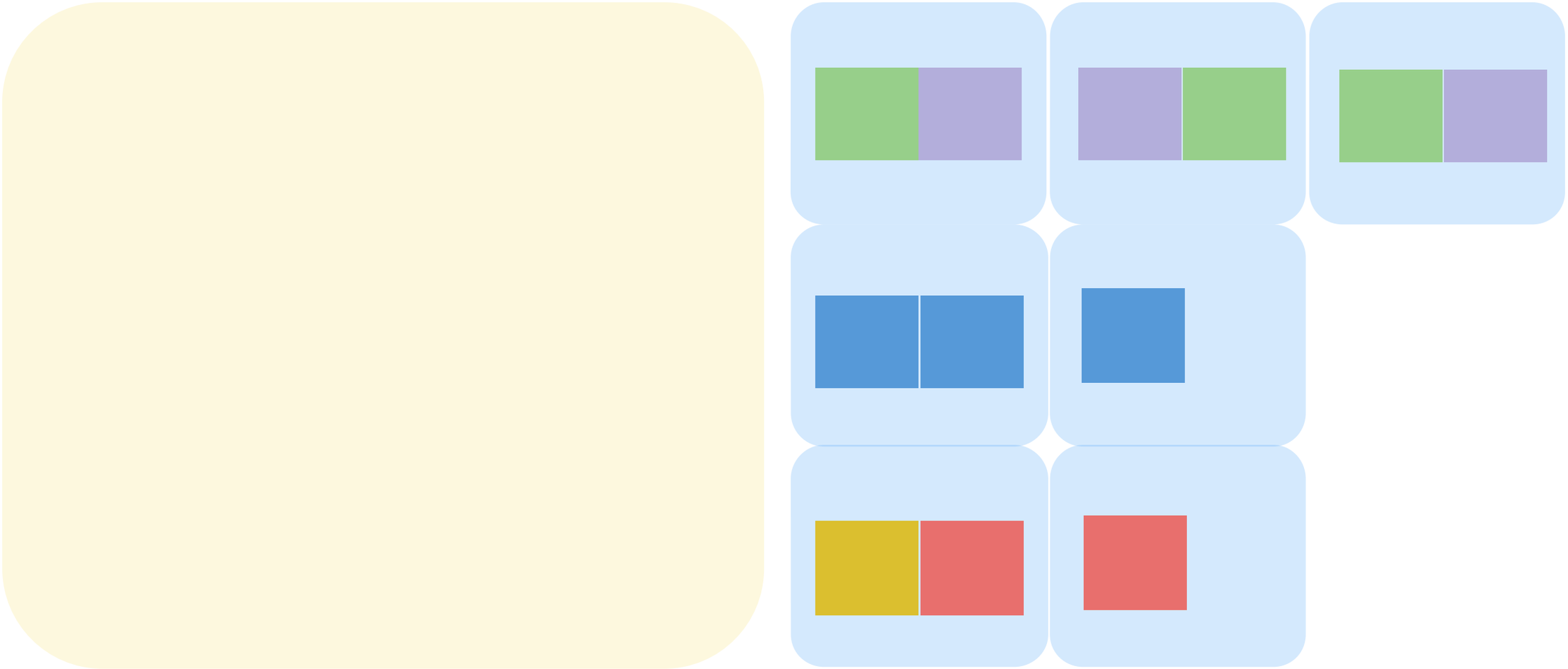
Pass 2: Conquer

- Rehash each partition.
- For a partition to fit in memory, it can only have B pages.
- To hash larger tables, use the partition algorithm recursively until the partition fits into memory
- # I/O's = 2N

Pass 2: Conquer

Create in-memory table for each partition.

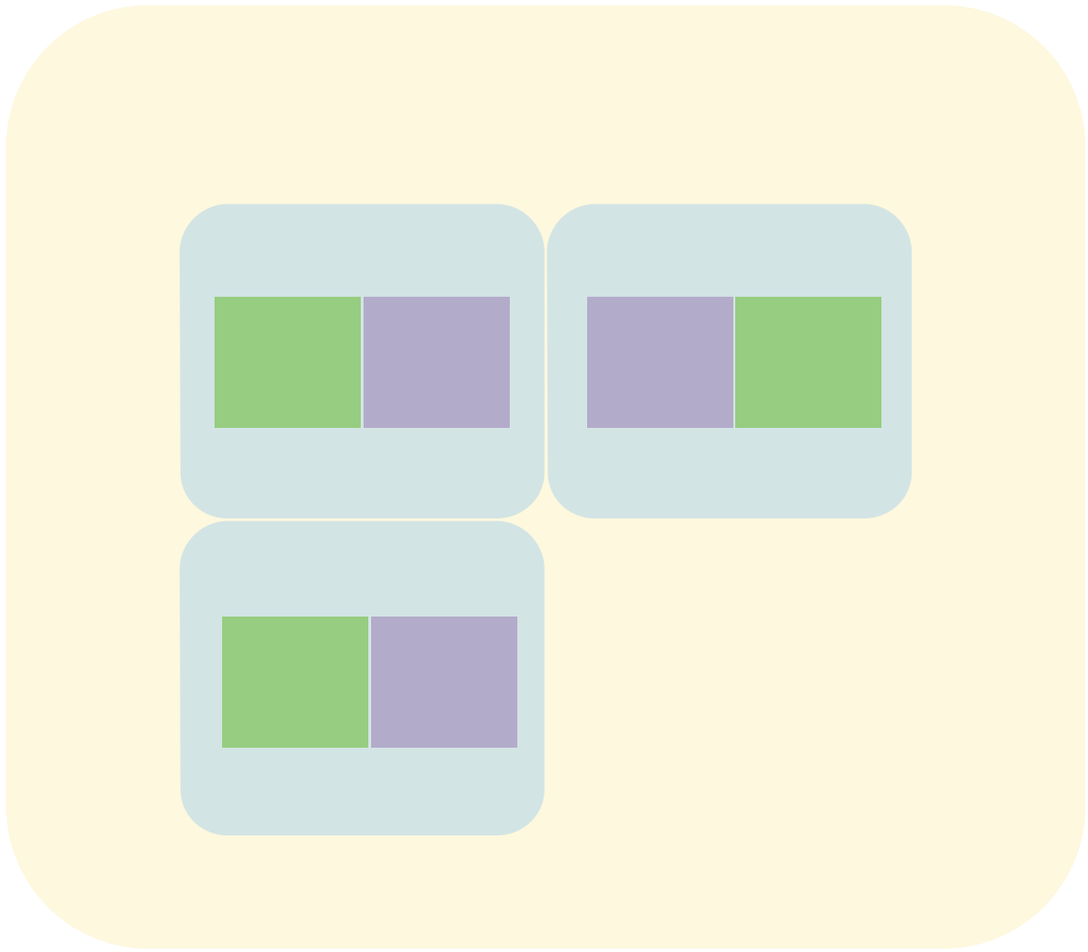
N=6, B=4

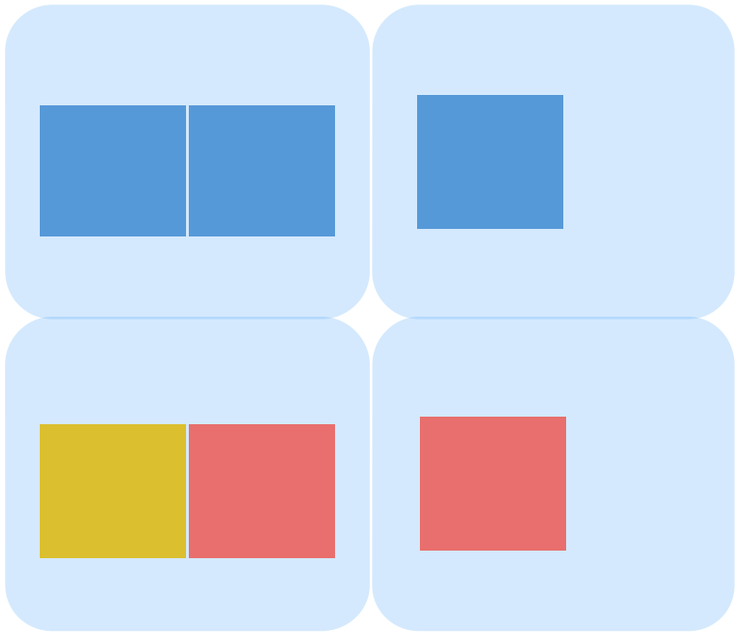


Pass 2: Conquer

Create in-memory table for each partition.

N=6, B=4





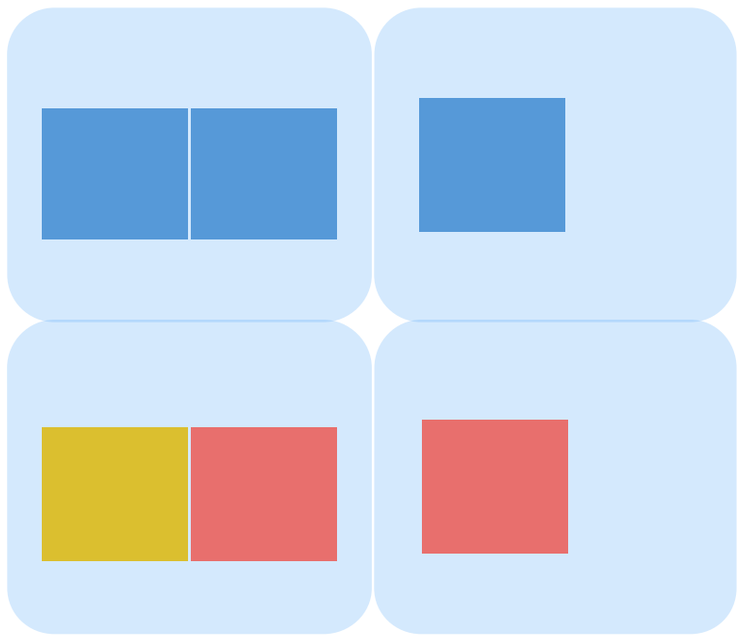
Pass 2: Conquer

Create in-memory table for each partition.

N=6, B=4

Green

Purple



Worksheet #5, 6, 7

Why can we process B * (B - 1) pages of data with external hashing in just two passes (divide and conquer phases)?

Why can we process B * (B - 1) pages of data with external hashing in just two passes (divide and conquer phases)?

- Our main limitation is how big the partitions can be after the partition hashing. Since we need to be able to read in the whole partition into memory, each partition can be at most B pages big.

If you’re processing exactly B * (B - 1) pages of data, is it likely that you’ll have to perform recursive external hashing? Why?

If you're processing exactly B * (B - 1) pages of data, is it likely that you'll have to perform recursive external hashing? Why?

- You would have to have an absolutely perfect hash function that evenly distributes any record into the B-1 partitions. This is almost impossible in practice. Rather, we should expect that some partitions may be larger than B after partition hashing.

While you recursively perform external hashing, you reuse the same hash functions for partitioning. What's the problem with this?

While you recursively perform external hashing, you reuse the same hash functions for partitioning. What's the problem with this?

- The partition that is too big to fit in memory will still be too big to fit in memory if we maintain the same partition hashing strategy.

Tournament Sort

- Used to gather initial runs for external sorting algorithms
- Alternative to quicksort
  - Quicksort faster, but tournament sort longer runs
  - Average length of a run: 2(B-2)

H1

H2

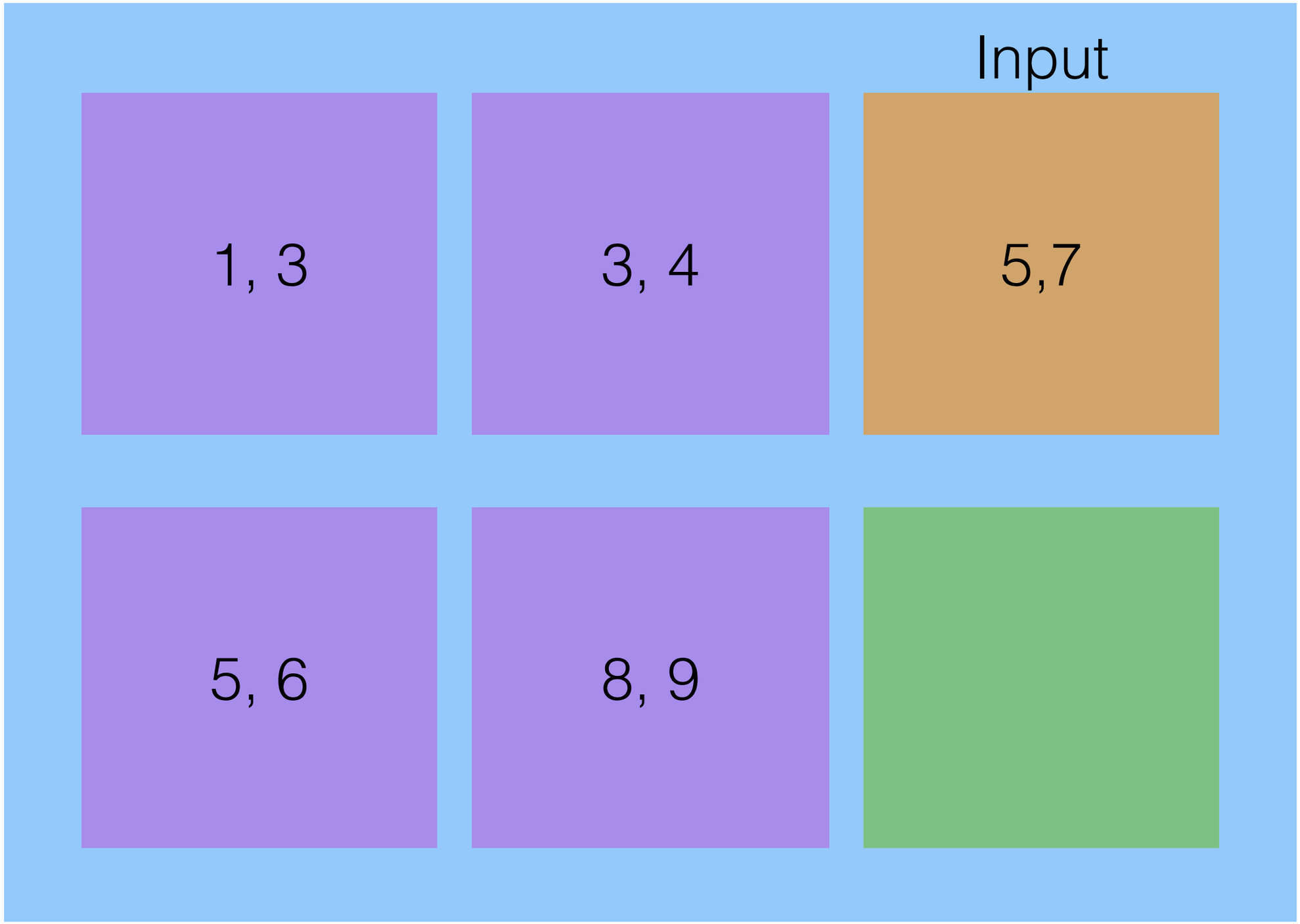
2,1

4,4

9,8

Output:

Tournament Sort



Tournament Sort

H1

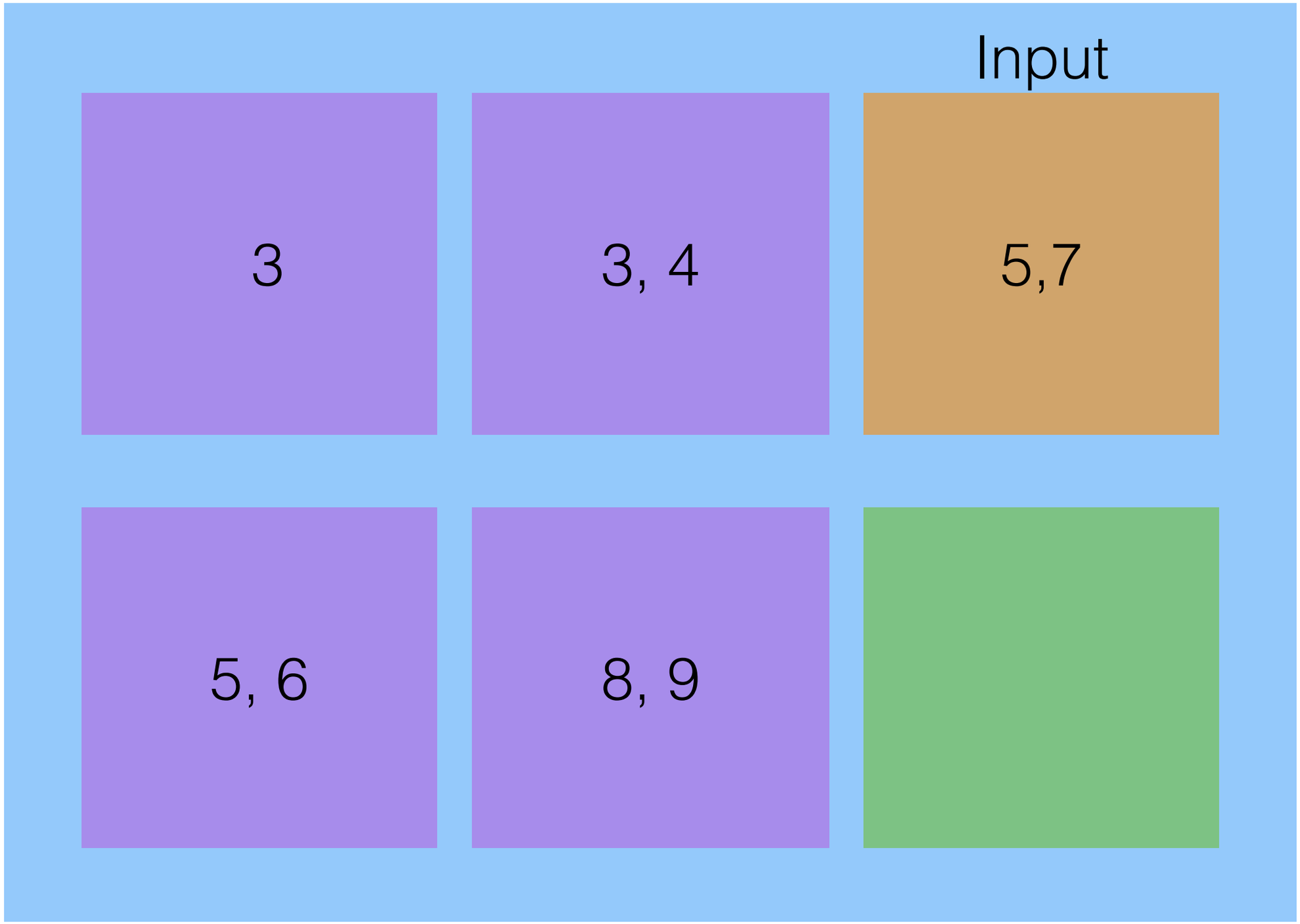
H2

2,1

4,4

9,8

Output:
1



Tournament Sort

H1

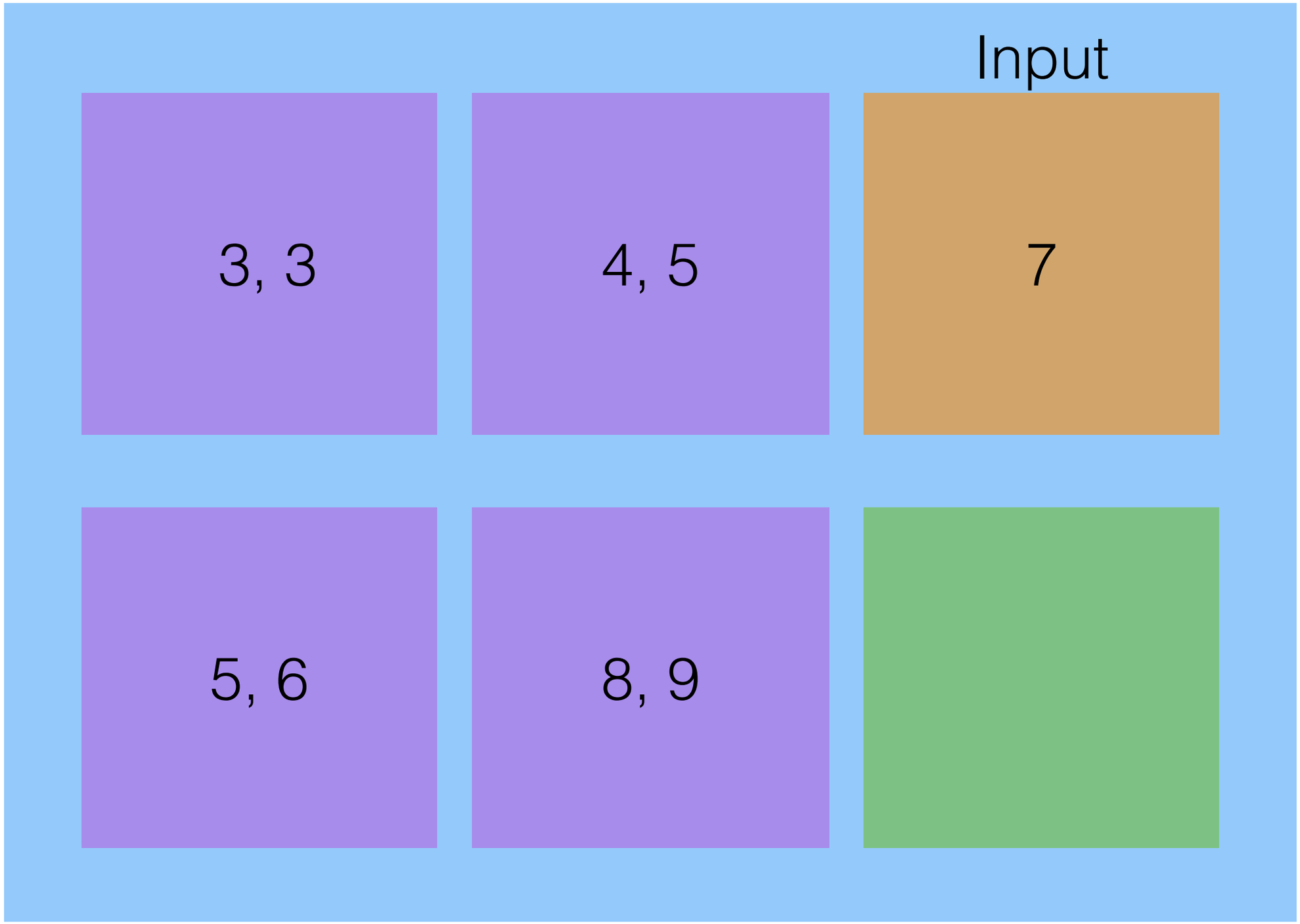
H2

2,1

4,4

9,8

Output:
1



Tournament Sort

H1

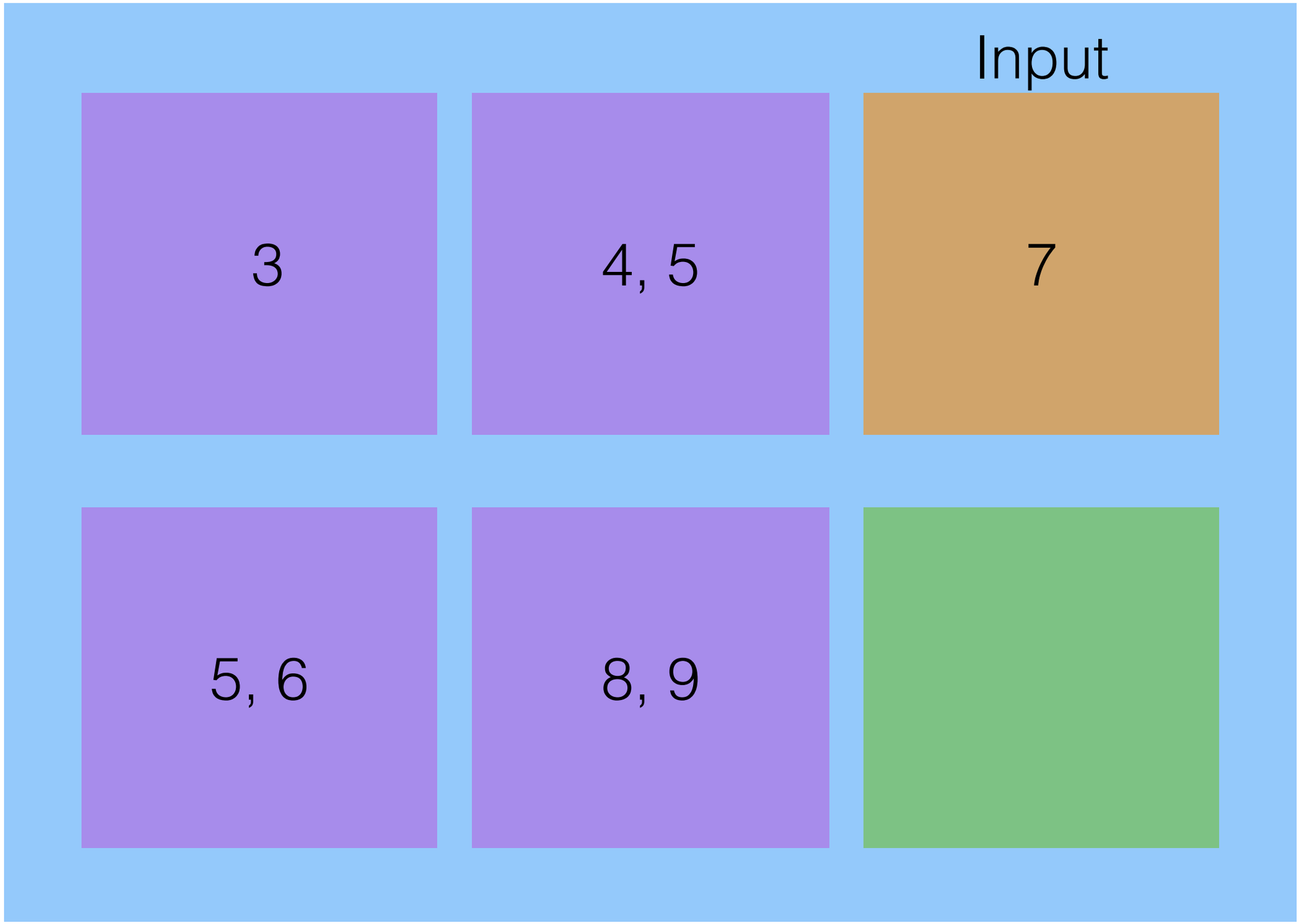
H2

2,1

4,4

9,8

Output:
1, 3



Tournament Sort

H1

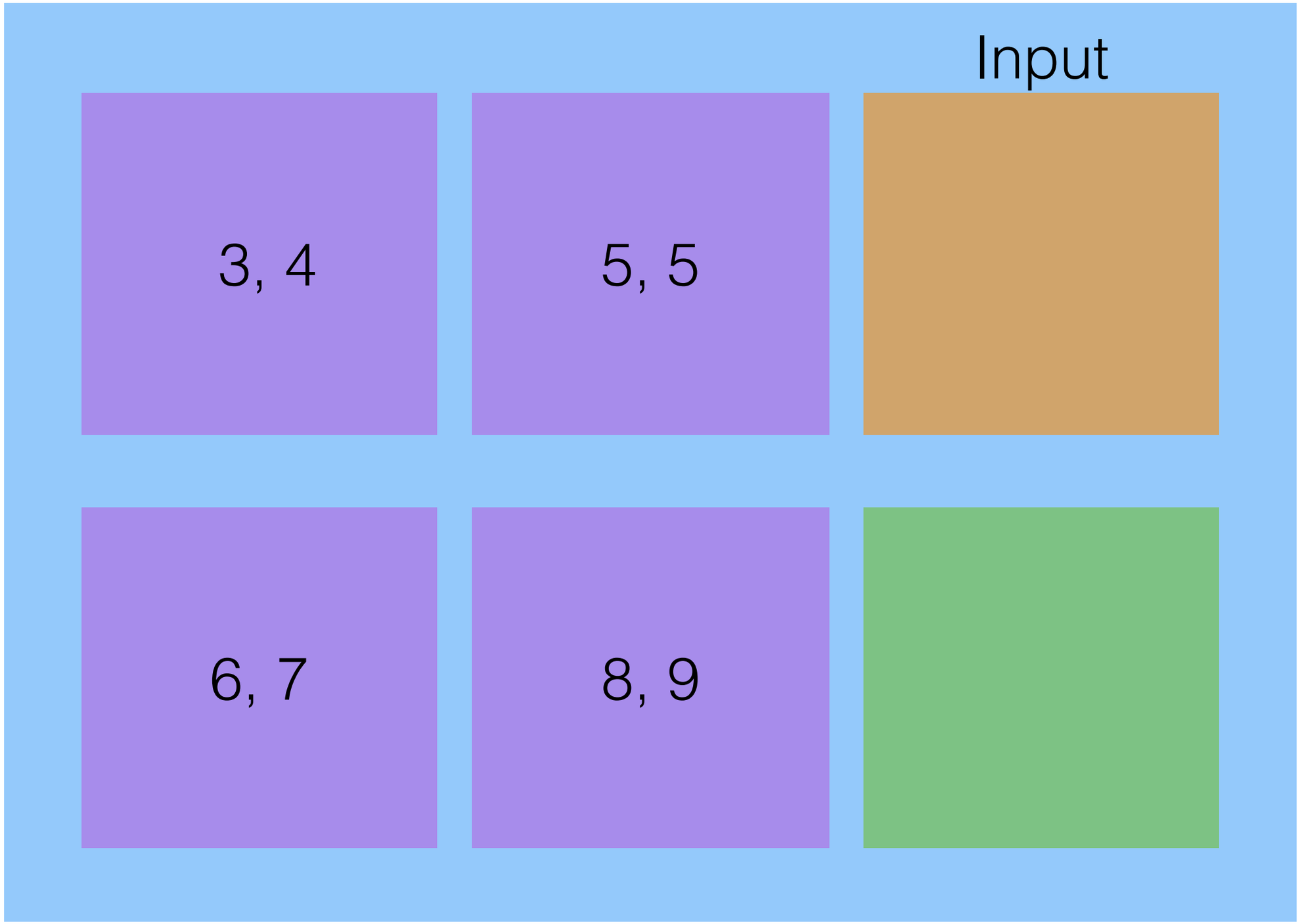
H2

2,1

4,4

9,8

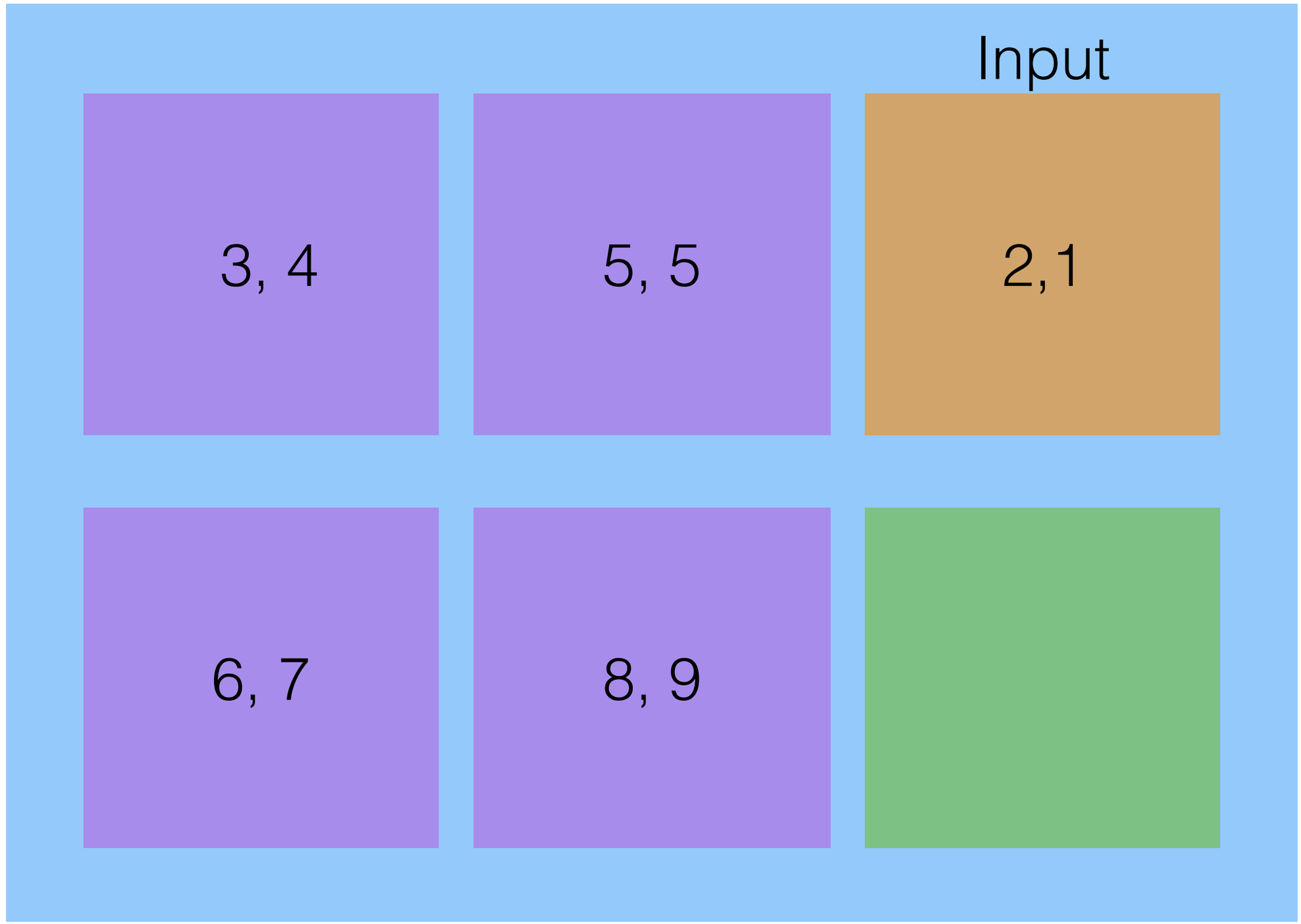
Output:
1, 3



H1

H2

Tournament Sort



4,4

9,8

Output:
1, 3

Tournament Sort

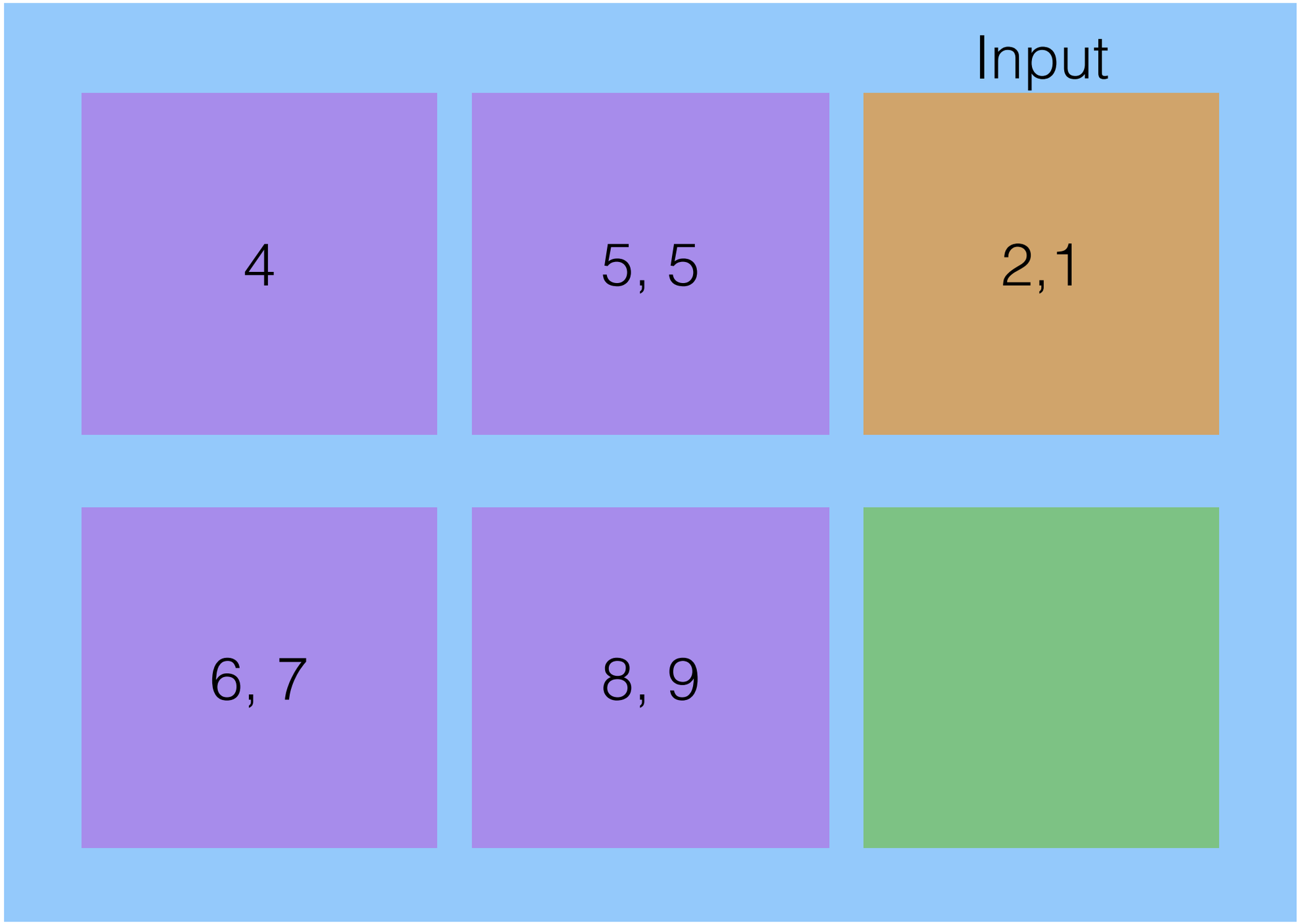
H1

H2

4,4

9,8

Output:
1, 3, 3



Tournament Sort

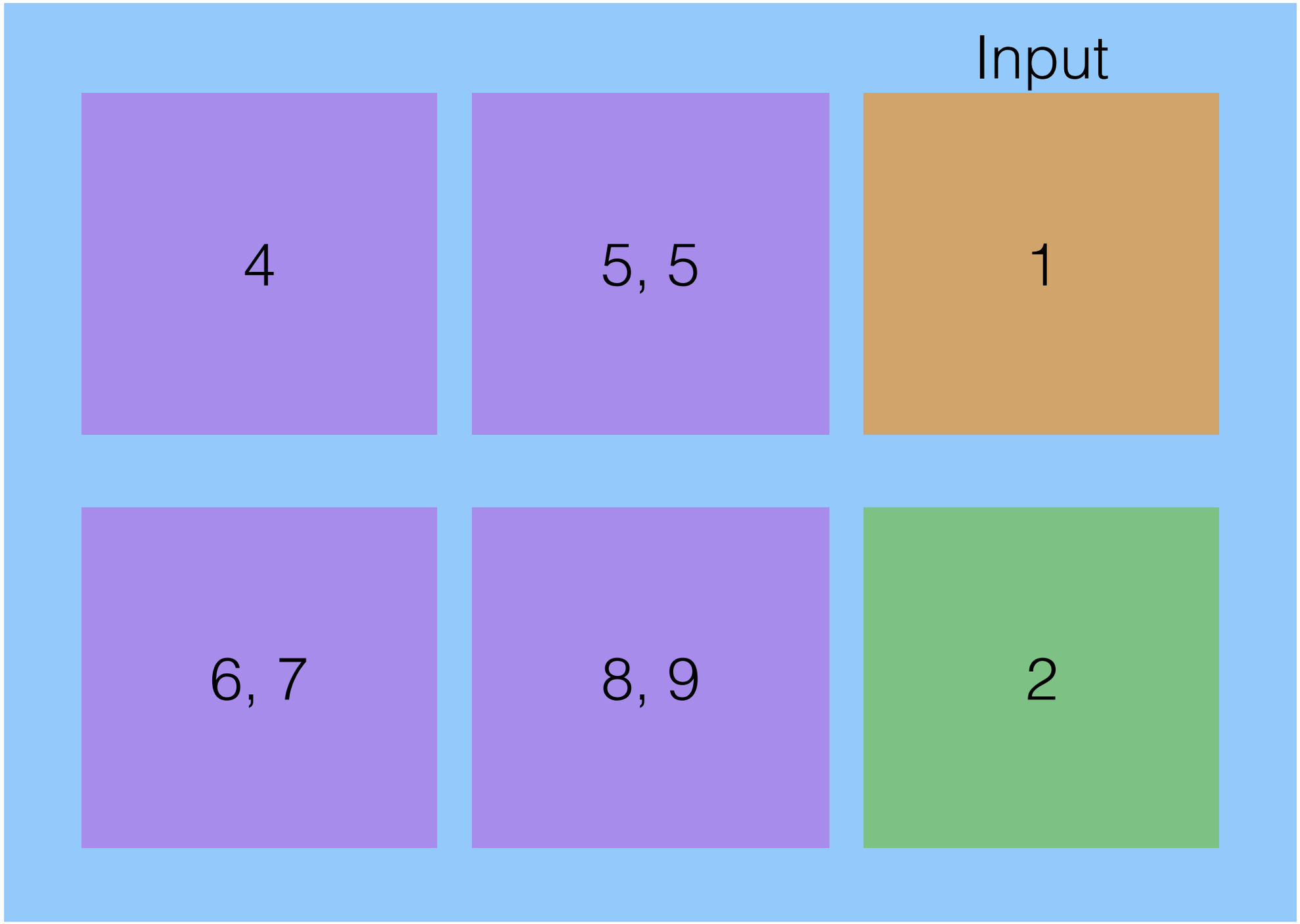
H1

H2

4,4

9,8

Output:
1, 3, 3



H1

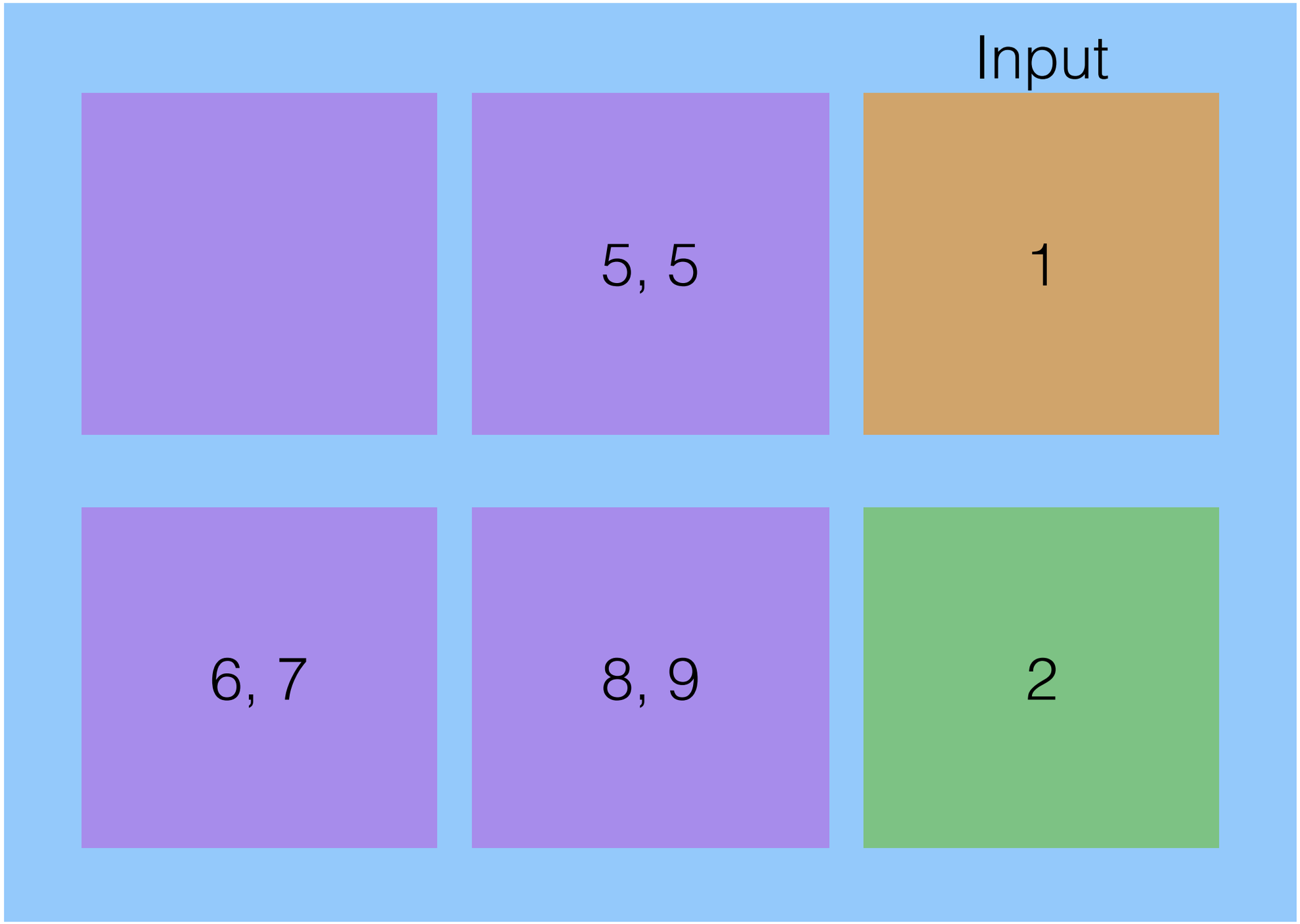
H2

Tournament Sort

4,4

9,8

Output:
1, 3, 3, 4



Tournament Sort

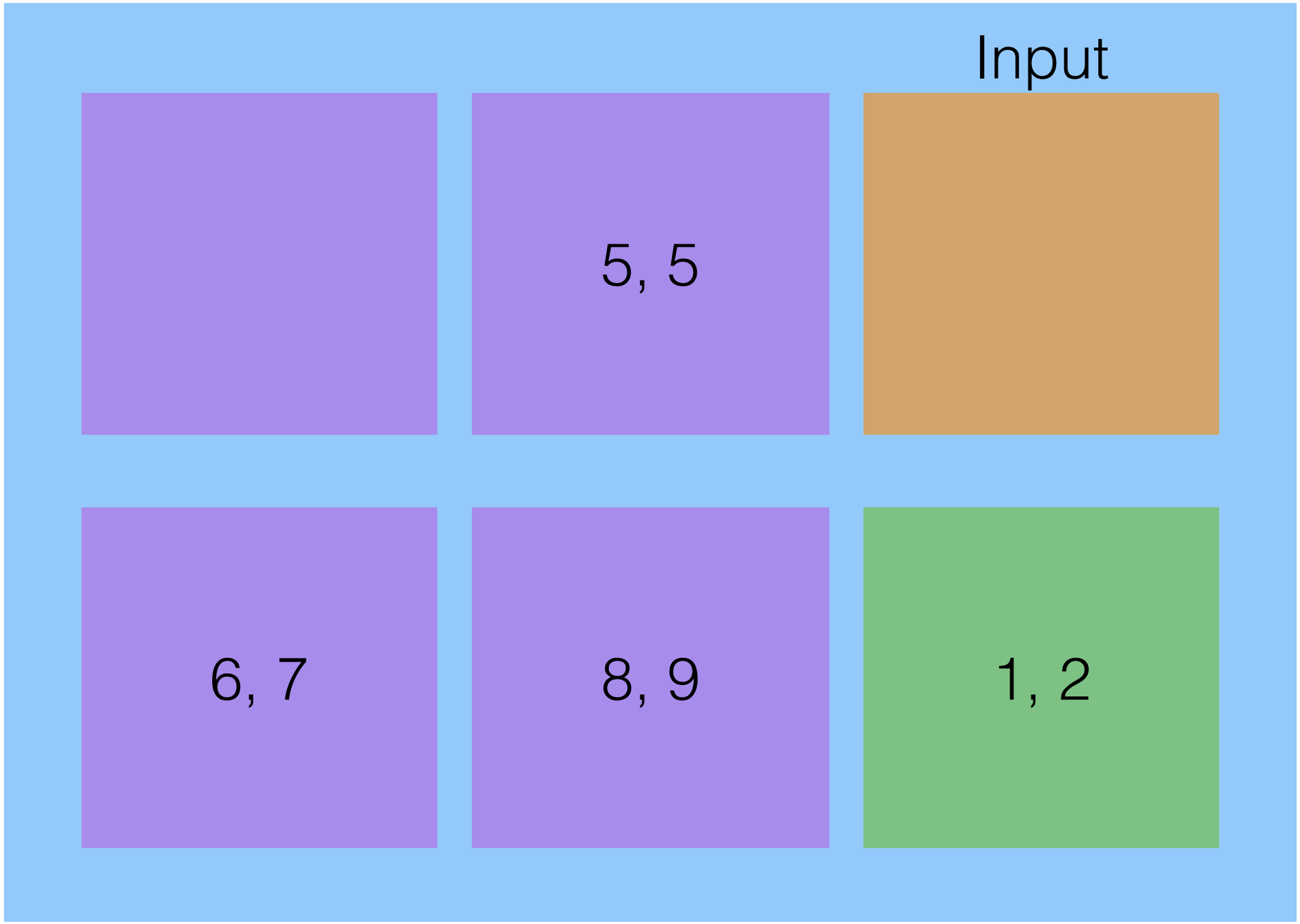
H1

H2

4,4

9,8

Output:
1, 3, 3, 4



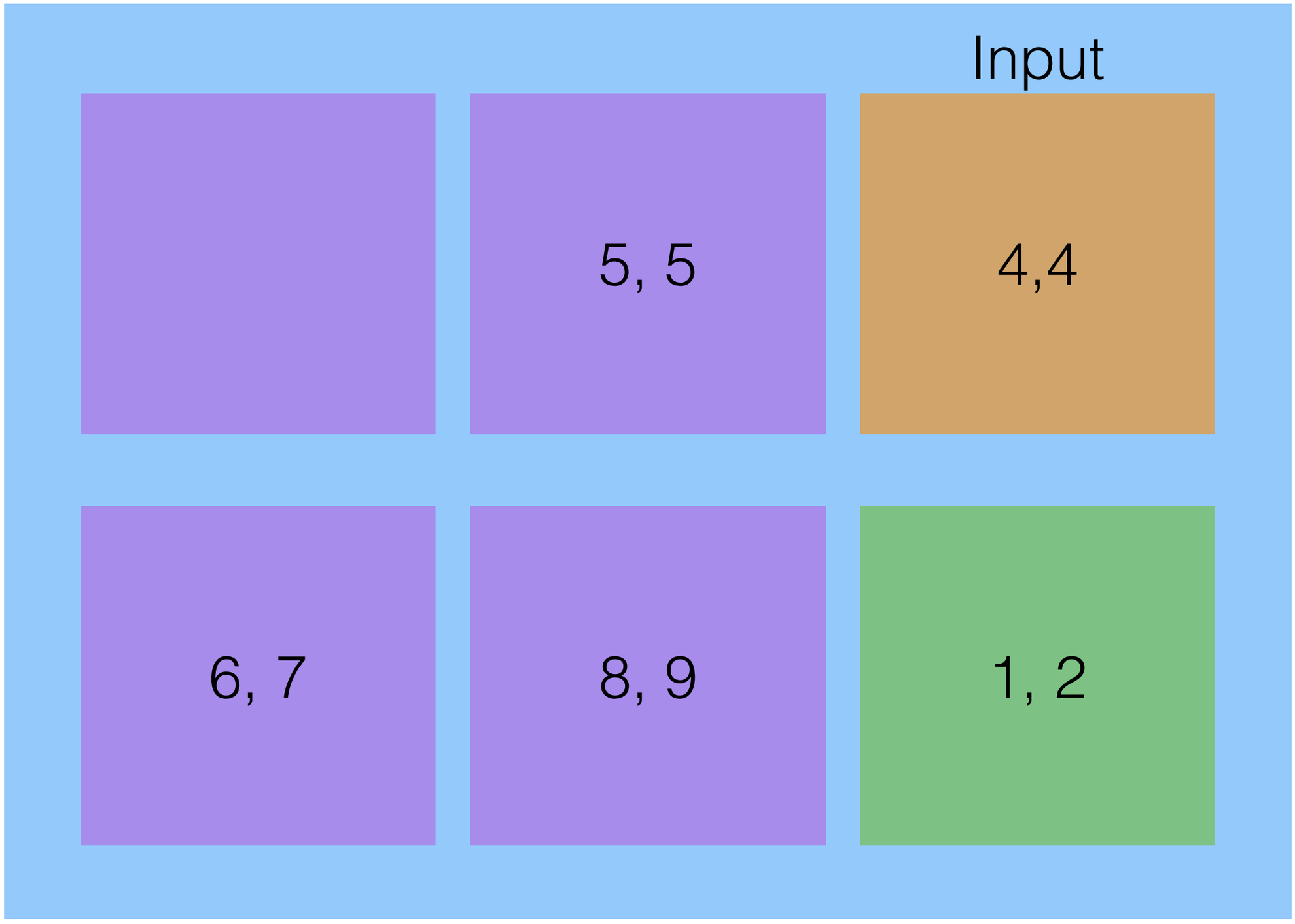
H1

H2

Tournament Sort

9,8

Output:
1, 3, 3, 4



H1

H2

Tournament Sort

Input

5

4,4

9,8

6, 7

8, 9

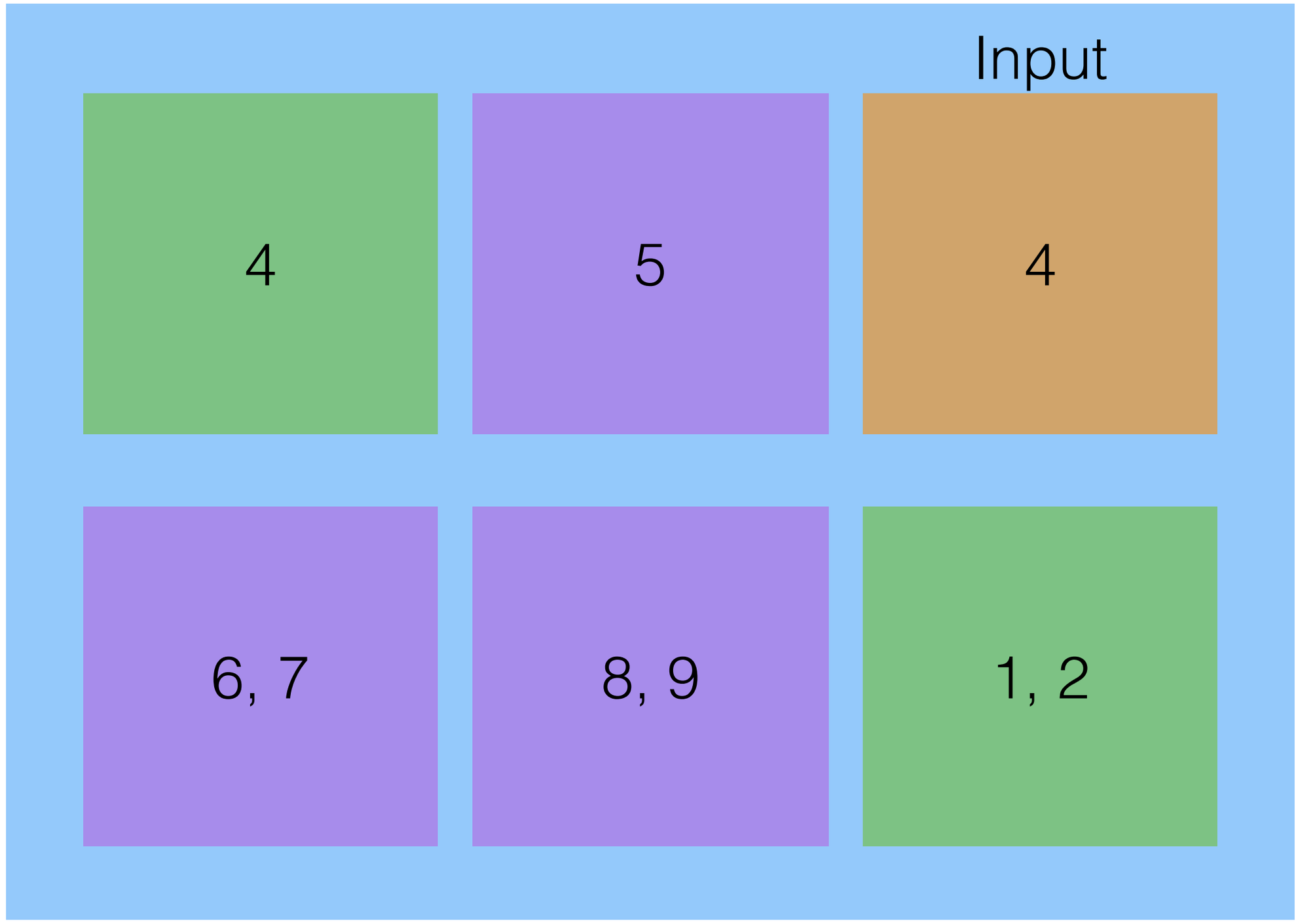
1, 2

Output:
1, 3, 3, 4,
5

H1

H2

Tournament Sort



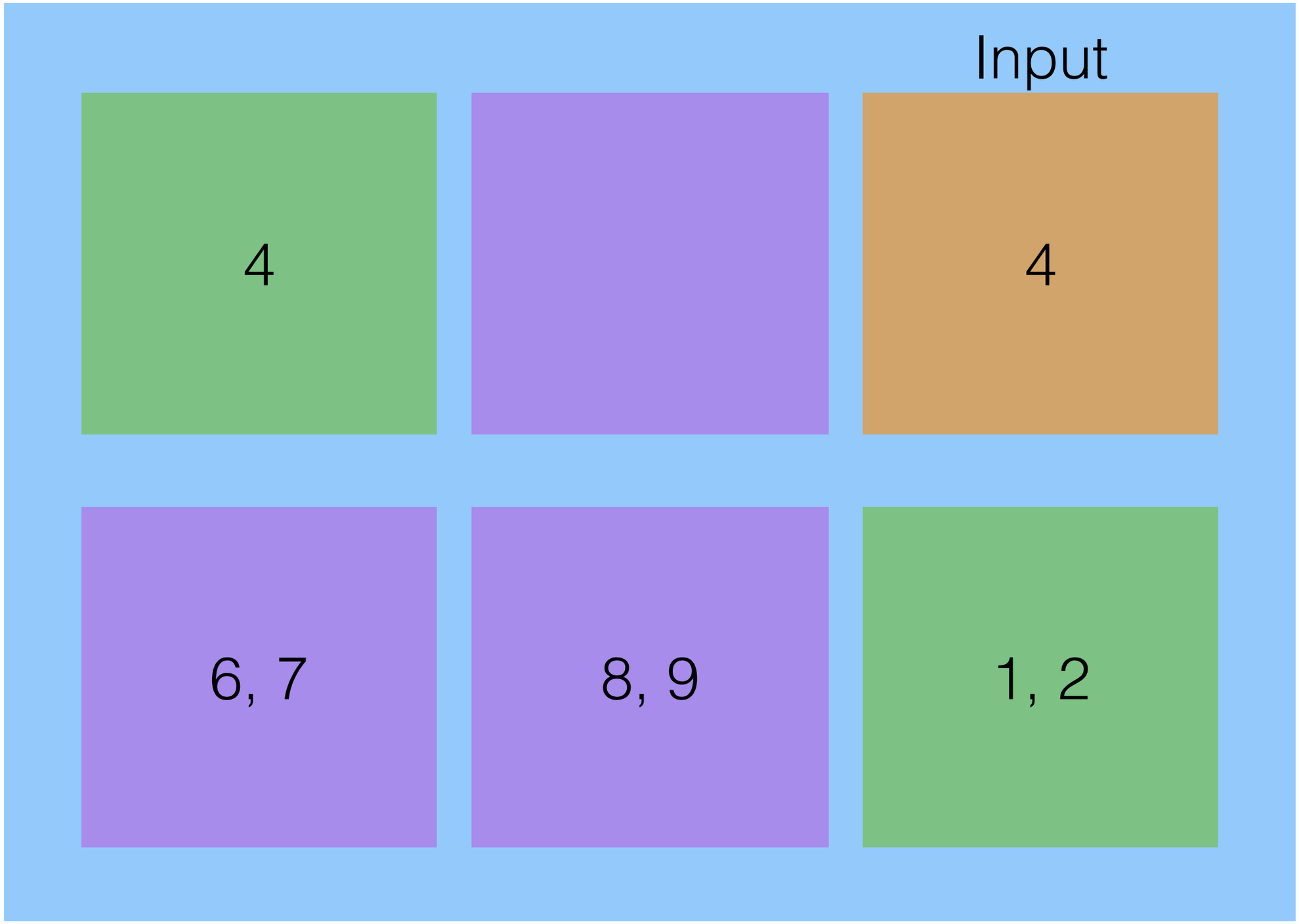
9,8

Output:
1, 3, 3, 4, 5

H1

H2

Tournament Sort



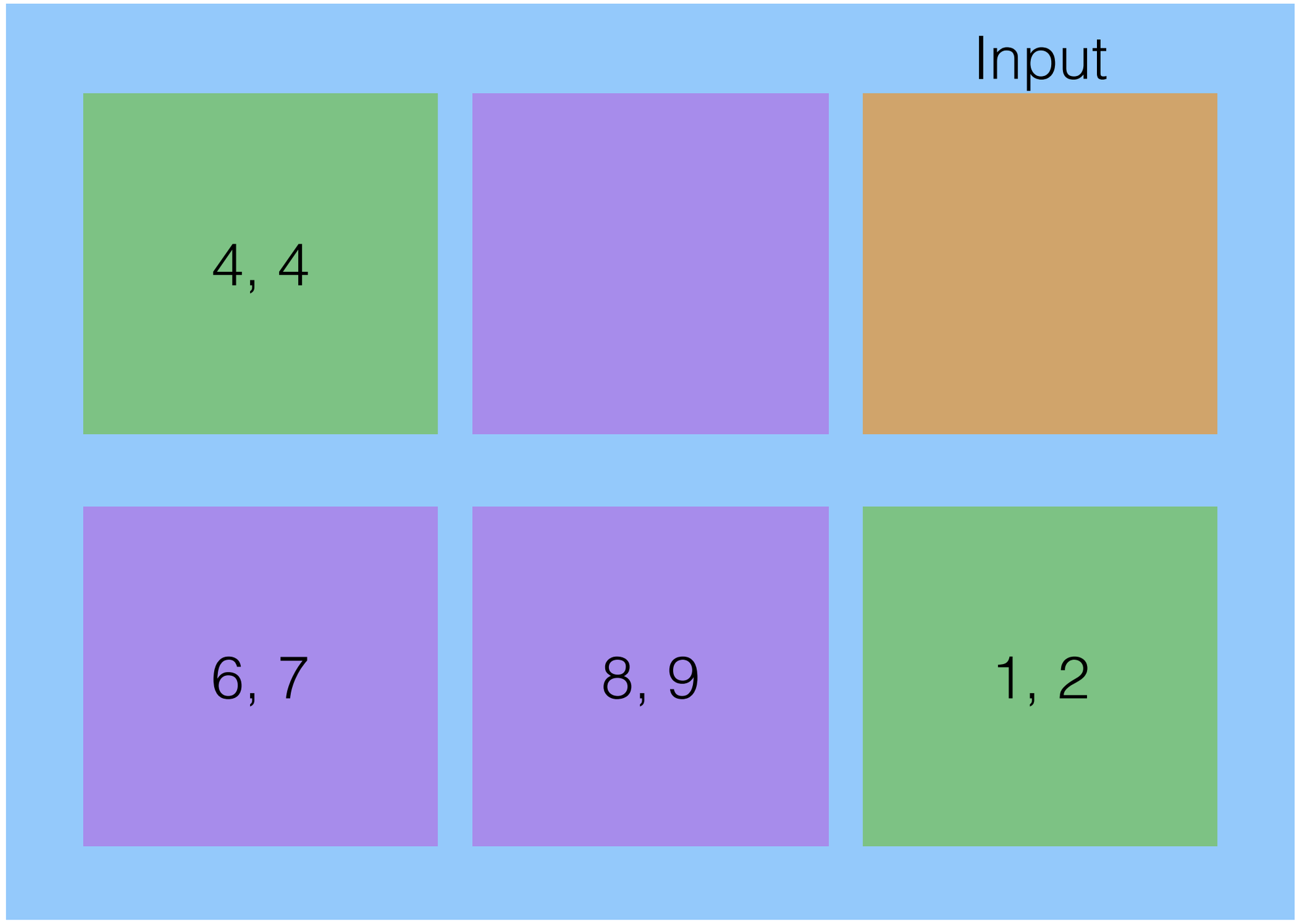
9,8

Output:
1, 3, 3, 4,
5, 5

H1

H2

Tournament Sort



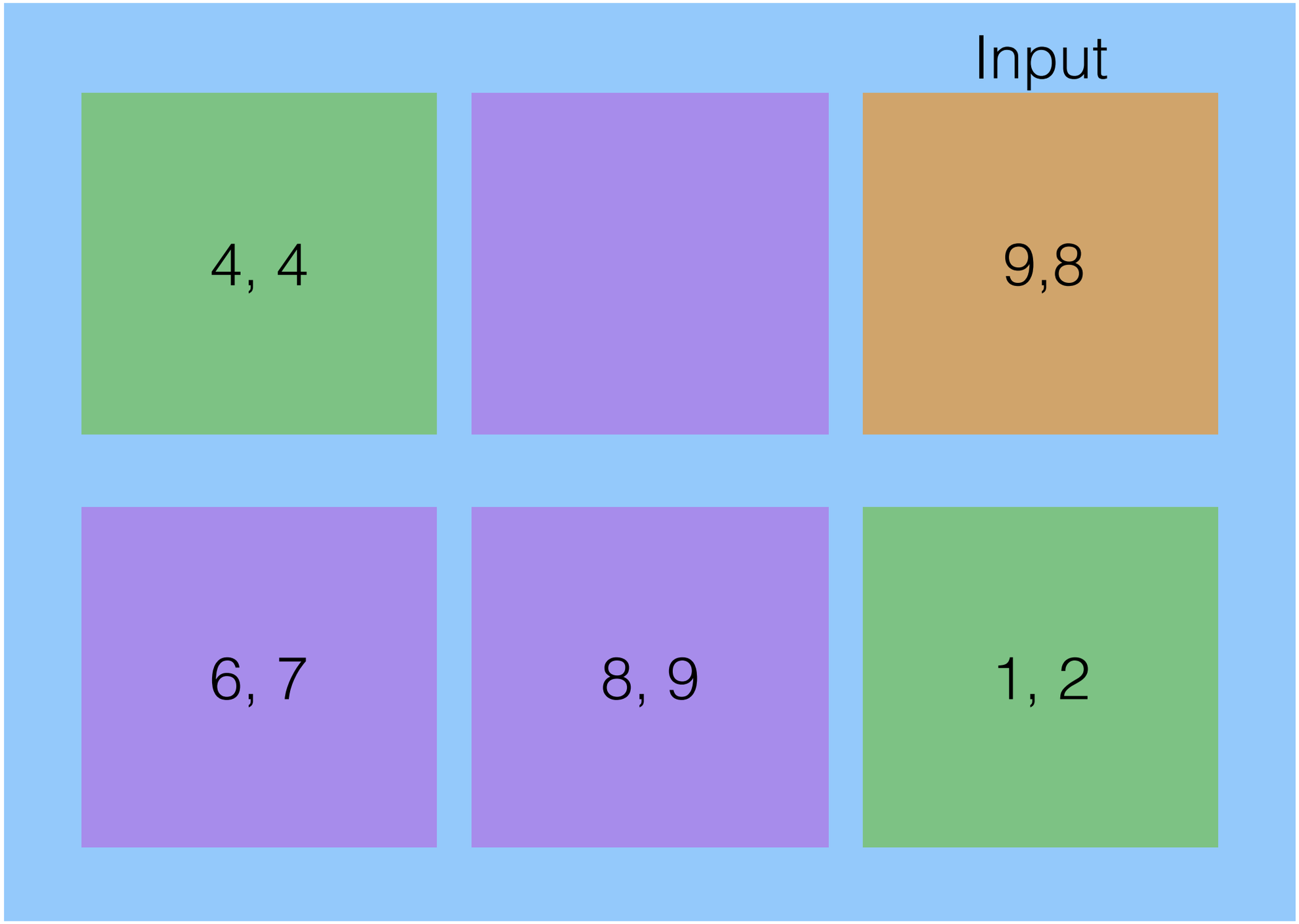
9,8

Output:
1, 3, 3, 4,
5, 5

H1

H2

Tournament Sort

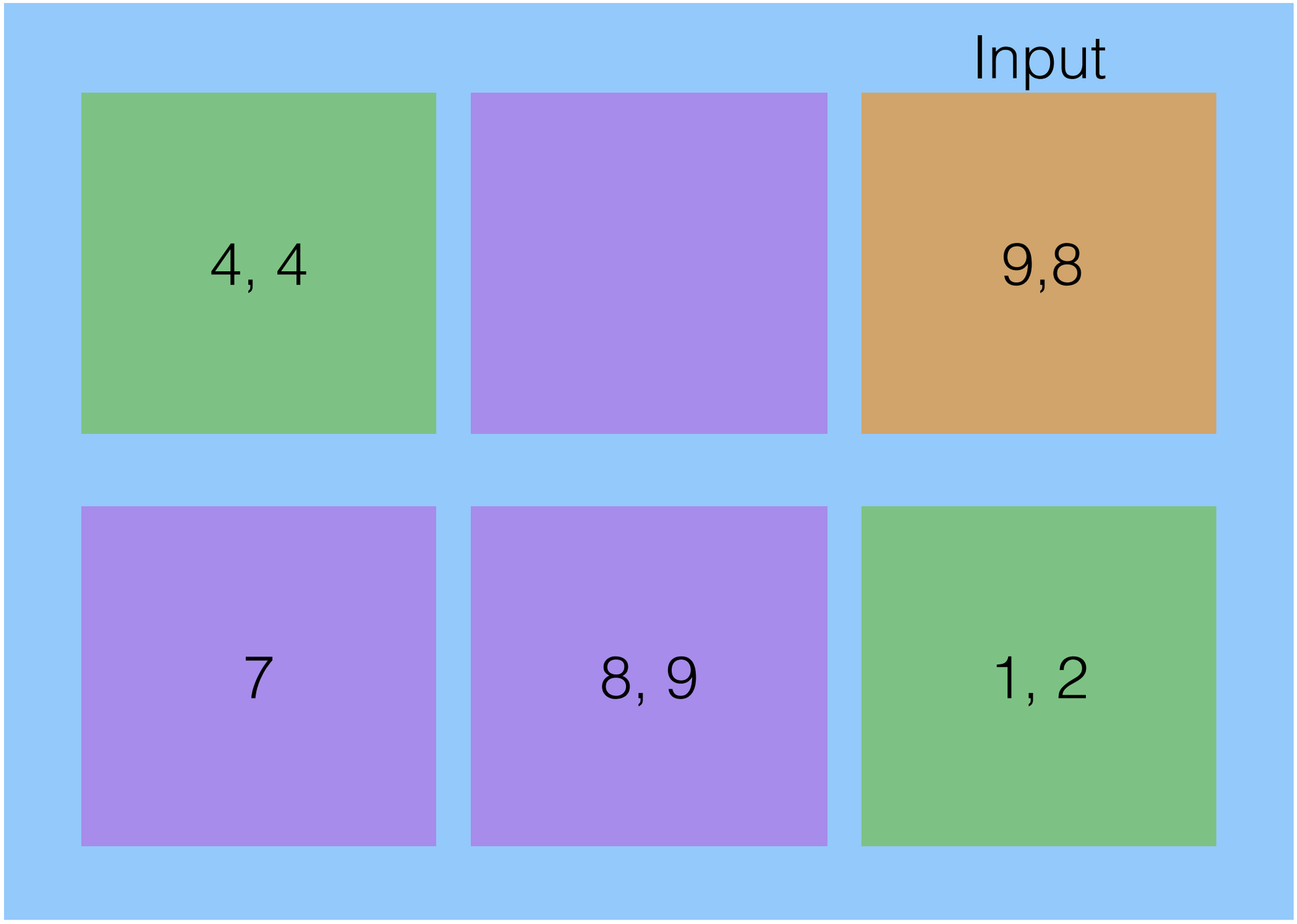


Output:
1, 3, 3, 4,
5, 5

H1

H2

Tournament Sort

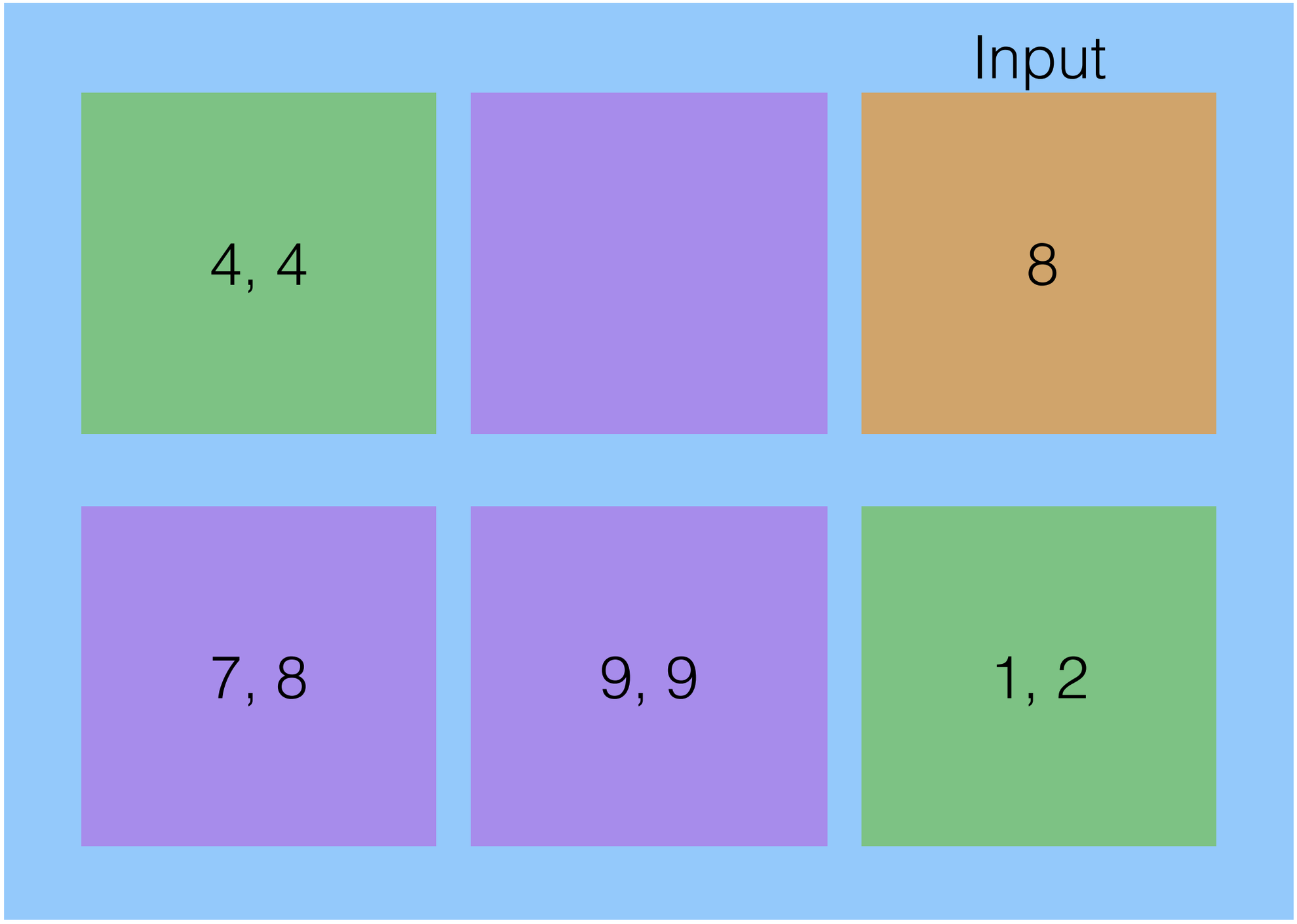


Output:
1, 3, 3, 4,
5, 5, 6

H1

H2

Tournament Sort

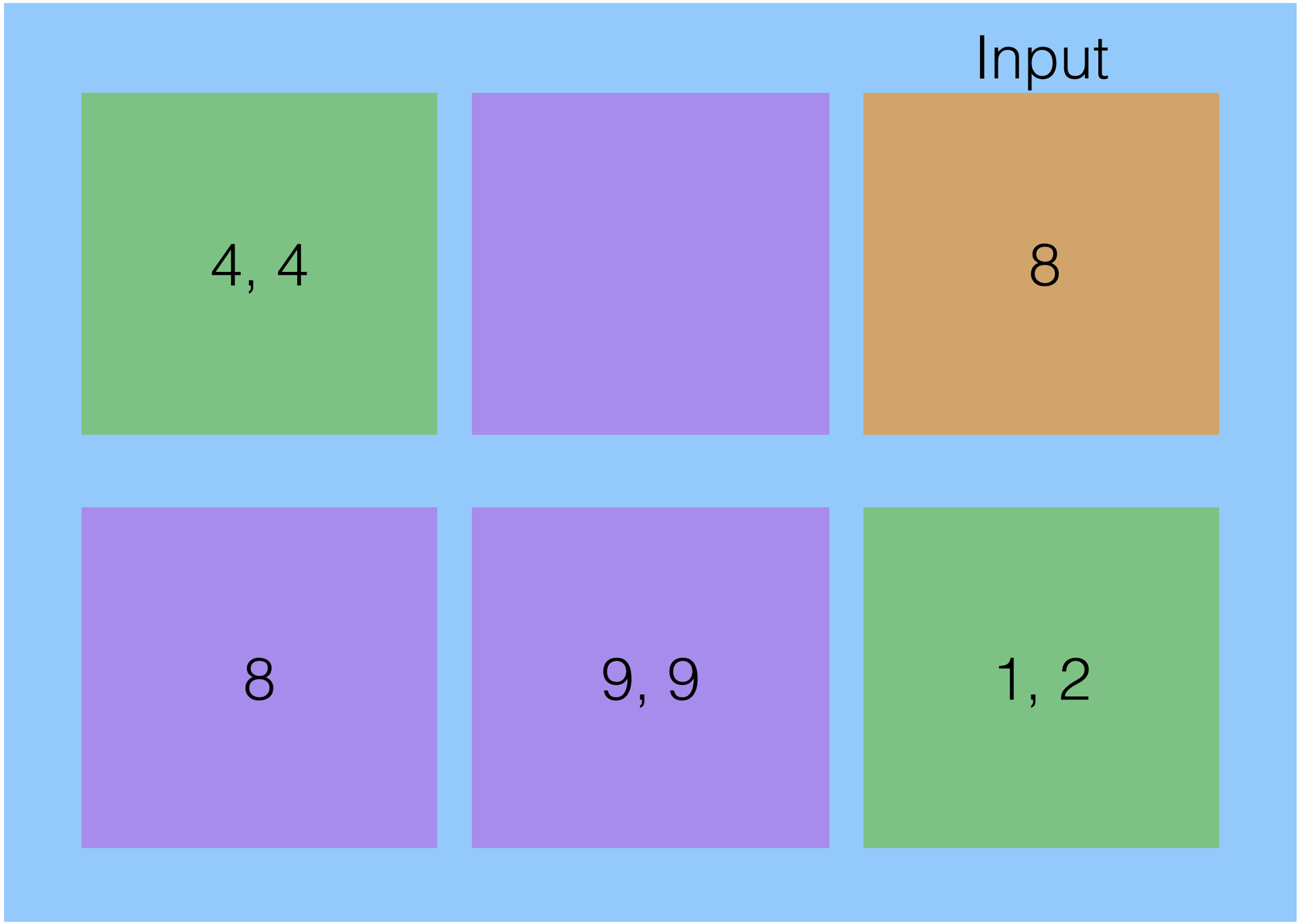


Output:
1, 3, 3, 4,
5, 5, 6

H1

H2

Tournament Sort

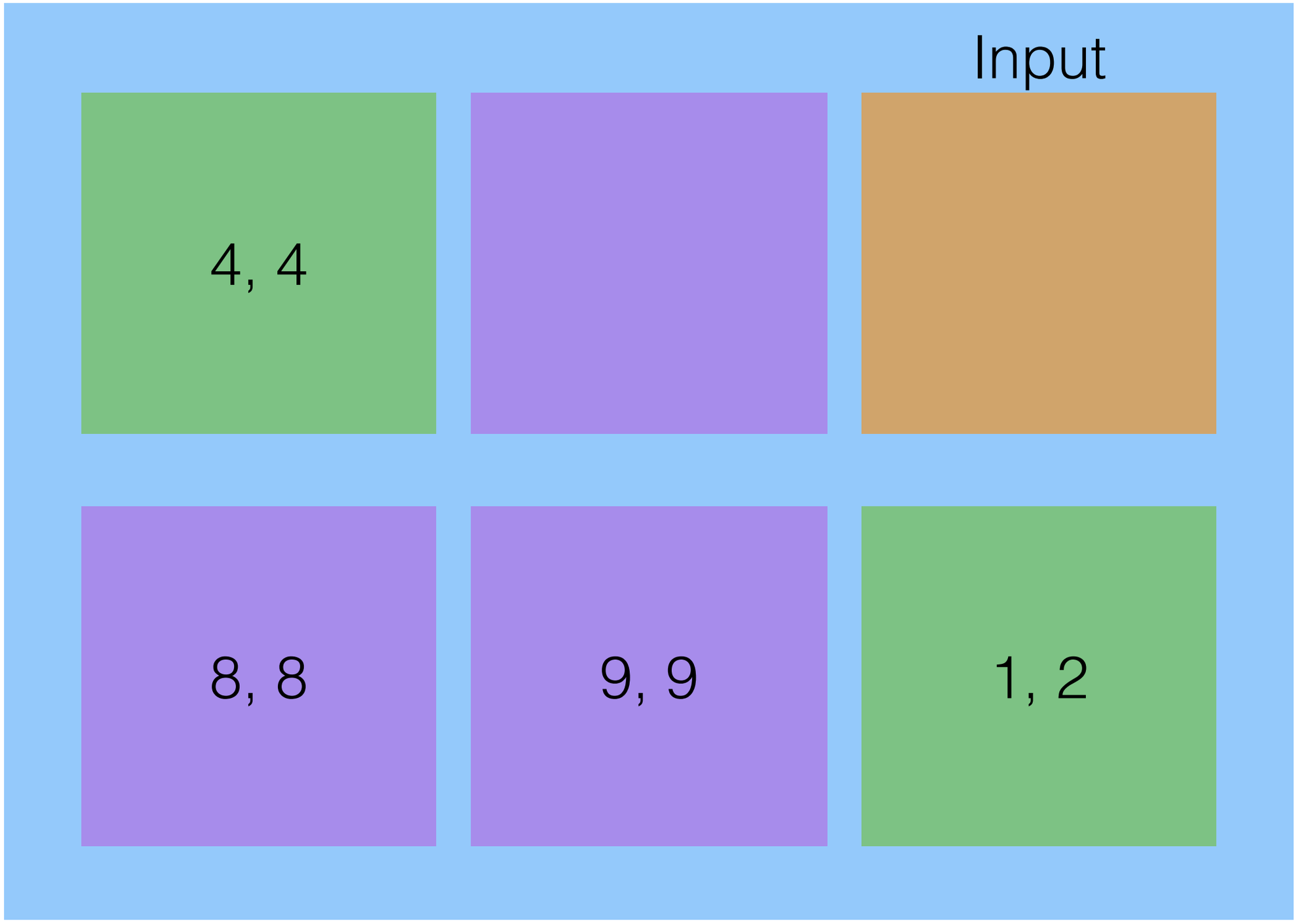


Output:
1, 3, 3, 4,
5, 5, 6, 7

H1

H2

Tournament Sort



Output:
1, 3, 3, 4,
5, 5, 6, 7

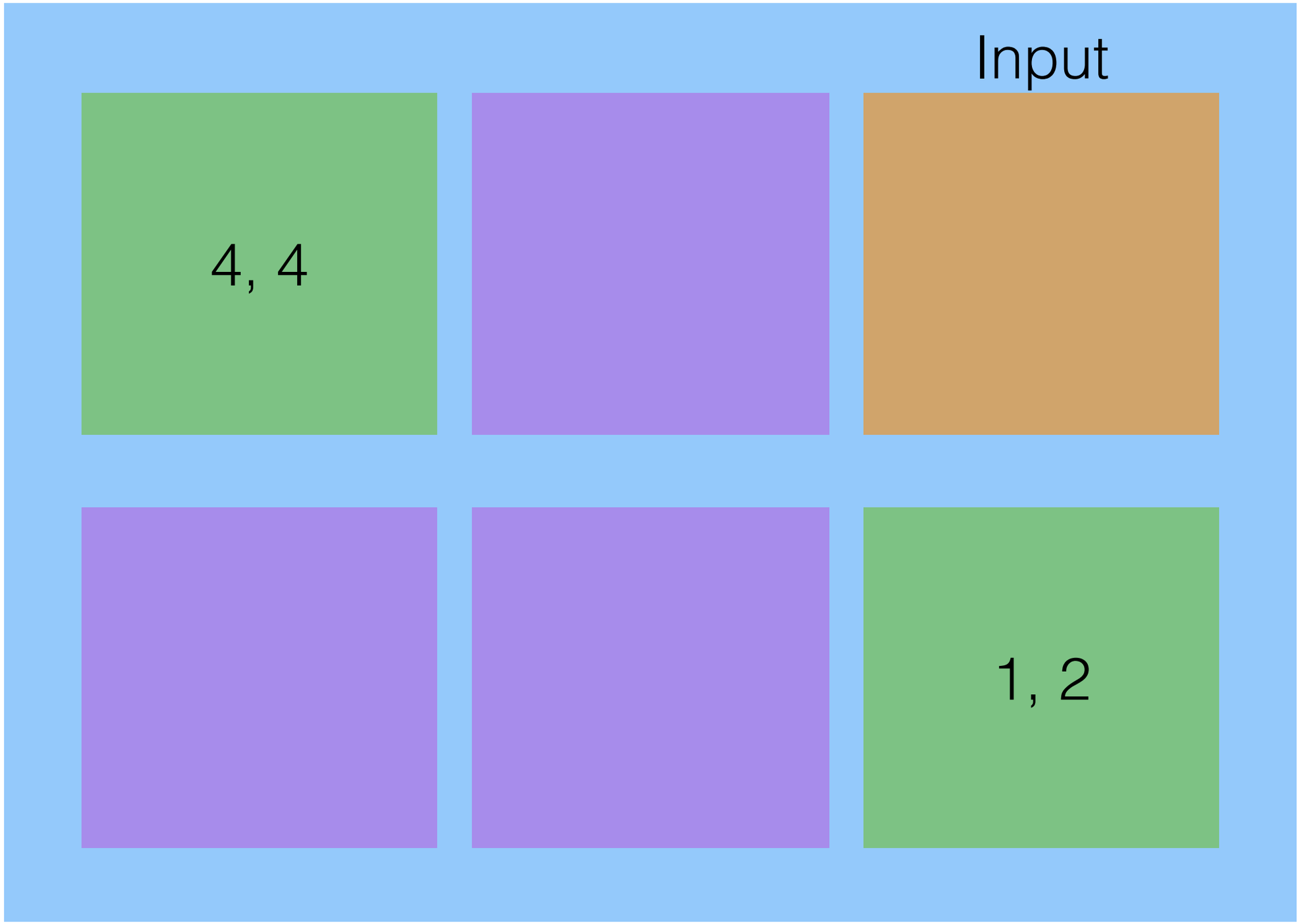
Tournament Sort

H1

H2

Output:

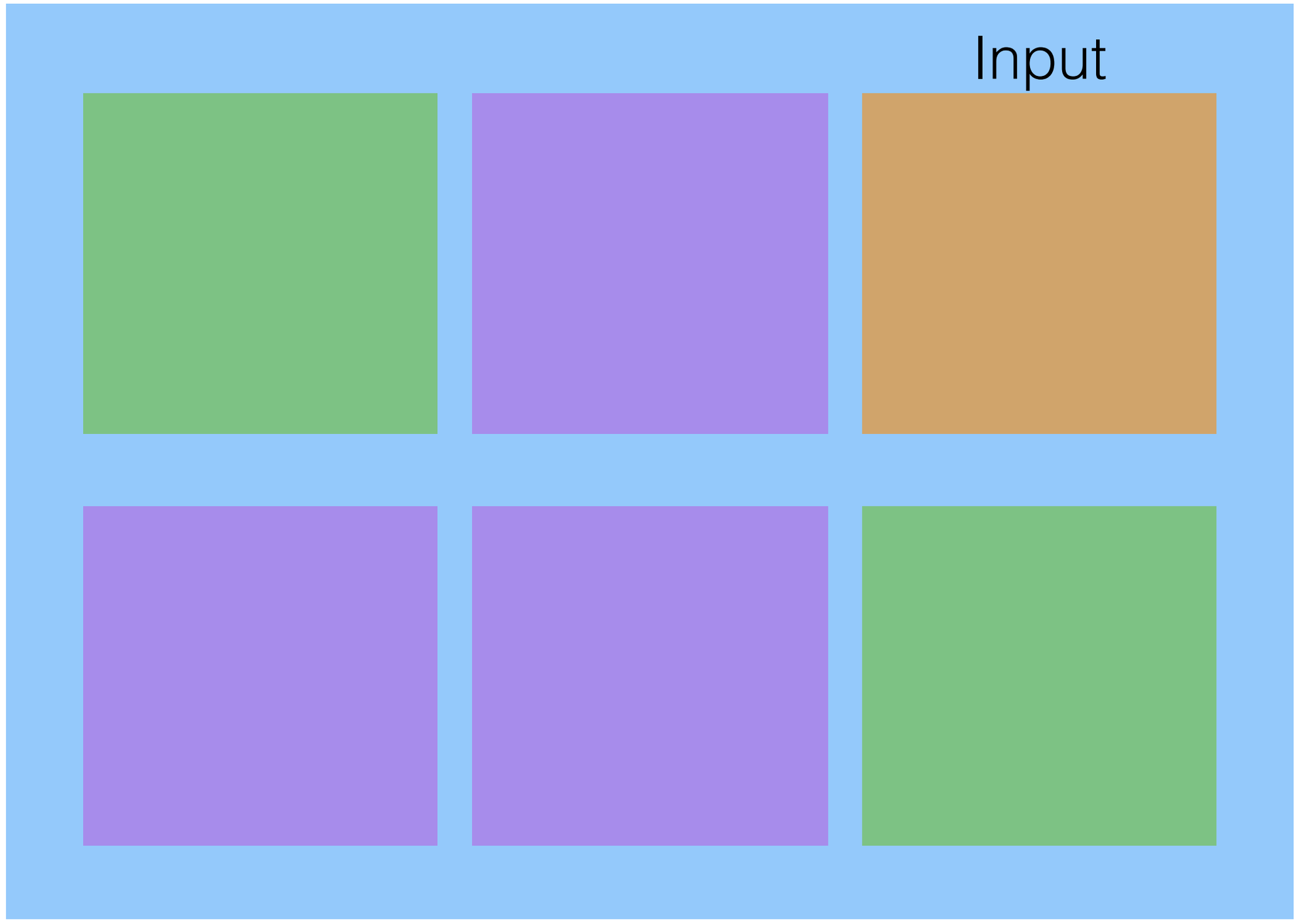
1, 3, 3, 4, 5, 5, 6, 7, 8, 8, 9, 9



H1

H2

Tournament Sort



Output:

1, 3, 3, 4,

5, 5, 6, 7,

8, 8, 9, 9

1, 2, 4, 4

Hashing Midterm Example

- **Hashing Animals:**
  - 4 tuples (animals) a page
  - 101 buffer pages, 32,000 animals
    - Want to group animals by type
- How many times do we have run the Partitioning stage of hashing to hash the animals, assuming all the partitions end up being the same length?

- **Hashing Animals:**
  - 4 tuples (animals) a page
  - 101 buffer pages, 32,000 animals
  - Want to group animals by type
- How many times do we have run the Partitioning stage of hashing to hash the animals, assuming all the partitions end up being the same length?
  - 32,000/4 = 8,000 pages of data
  - 8,000/100 = 80 pages per partition
  - (80 < 101) => each partition will fit in memory => only one partition stage required!