CS186 Discussion 1

(Rendezvous, External Sorting, External Hashing)

Matthew Deng

Matthew Deng

UC Berkeley B.S.'16 EECS
matthew.deng@berkeley.edu
Office Hours: Mondays 2-4PM @ Soda 611

TubeMogul

Berkeley
cs186

LinkedIn

Icebreaker

- Get to know the people around you:
  - Name
  - Year
  - Major
  - Interesting fact about yourself

Warm-Up

Worksheet #1, 2

Rendezvous Exercises

1. What is time-space rendezvous and why do we use it?

Rendezvous Exercises

1. What is time-space rendezvous and why do we use it?

- records are in the same place (RAM) at the same time
- useful when certain records need to be co-resident but are not guaranteed to be in the same input chunk
- implemented through out-of-core algorithms
    - external sorting
    - external hashing

Rendezvous Exercises

2. What is the difference between external sorting and external hashing?

Rendezvous Exercises

2. What is the difference between external sorting and external hashing?

- Sorting:
  - conquer and merge
  - good if we need to sort or already sorted
- Hashing:
  - divide and conquer
  - good if we need to remove duplicates
- Same memory requirement, i/o cost (for 2 passes)

External Sorting

External Sorting

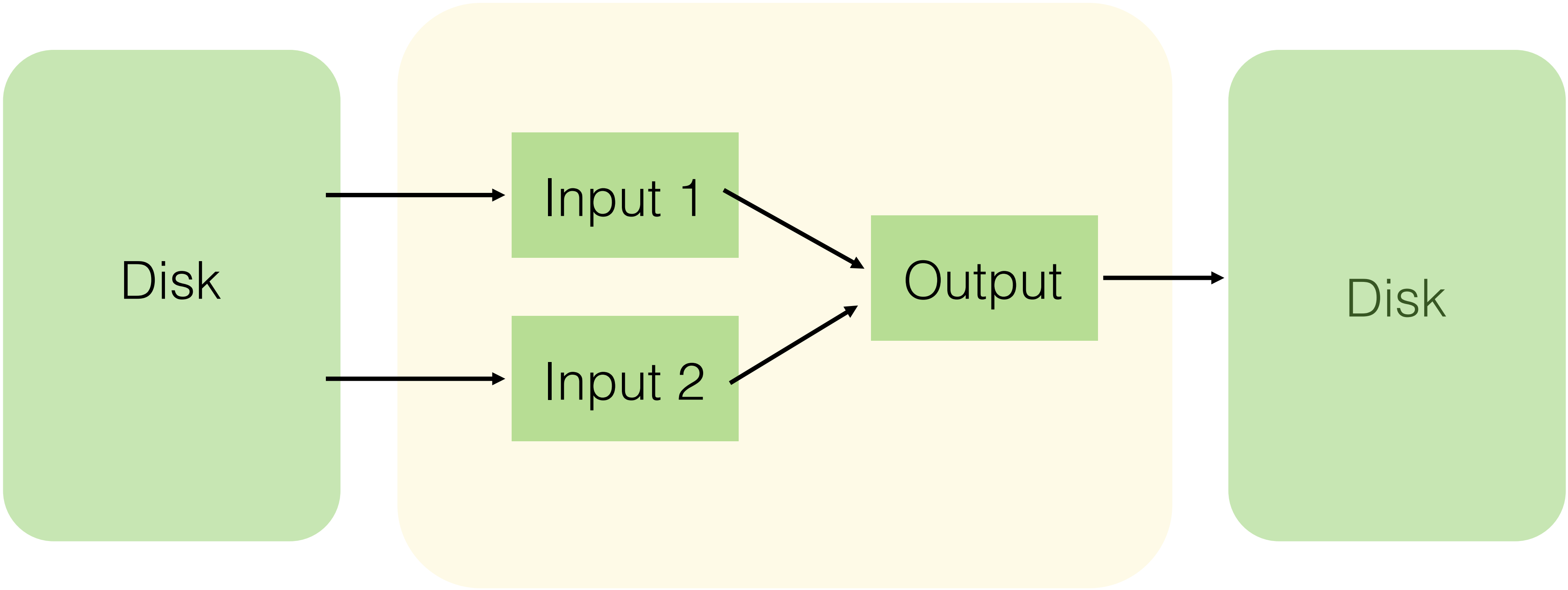
Want to sort data that does not fit in memory

2-Way Merge Sort

- Pass 0 (Conquer):
    - Sort 1 page at a time in memory
- Pass 1, 2, …, etc. (Merge):
    - Merge 2 runs

2-Way Merge Sort

Credits to Michelle Nguyen



Buffer size of 3 pages

Input

3,4

6,2

9,4

8,7

5,6

6,5

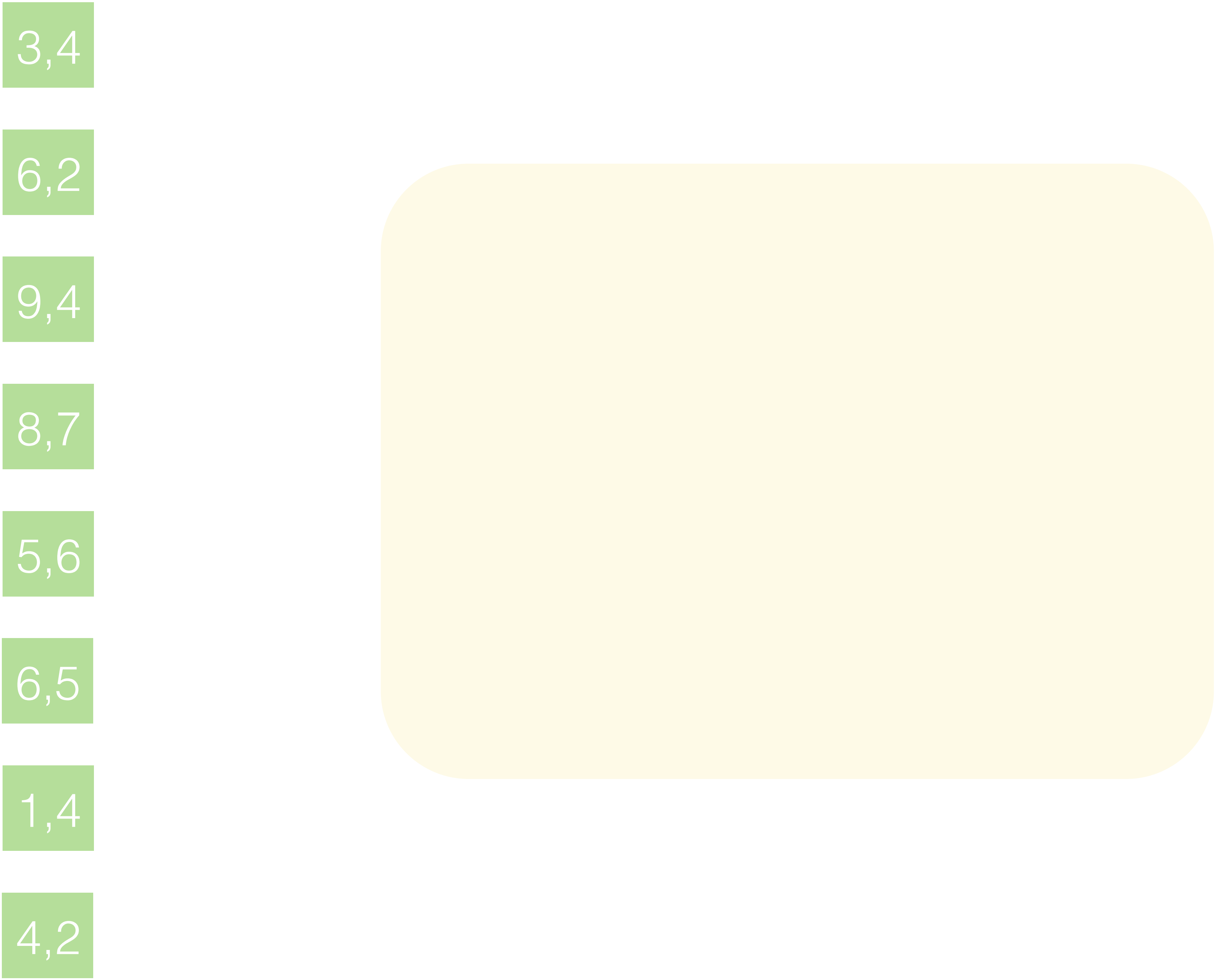
1,4

4,2

Pass 0

Input

Output



3,4

6,2

9,4

8,7

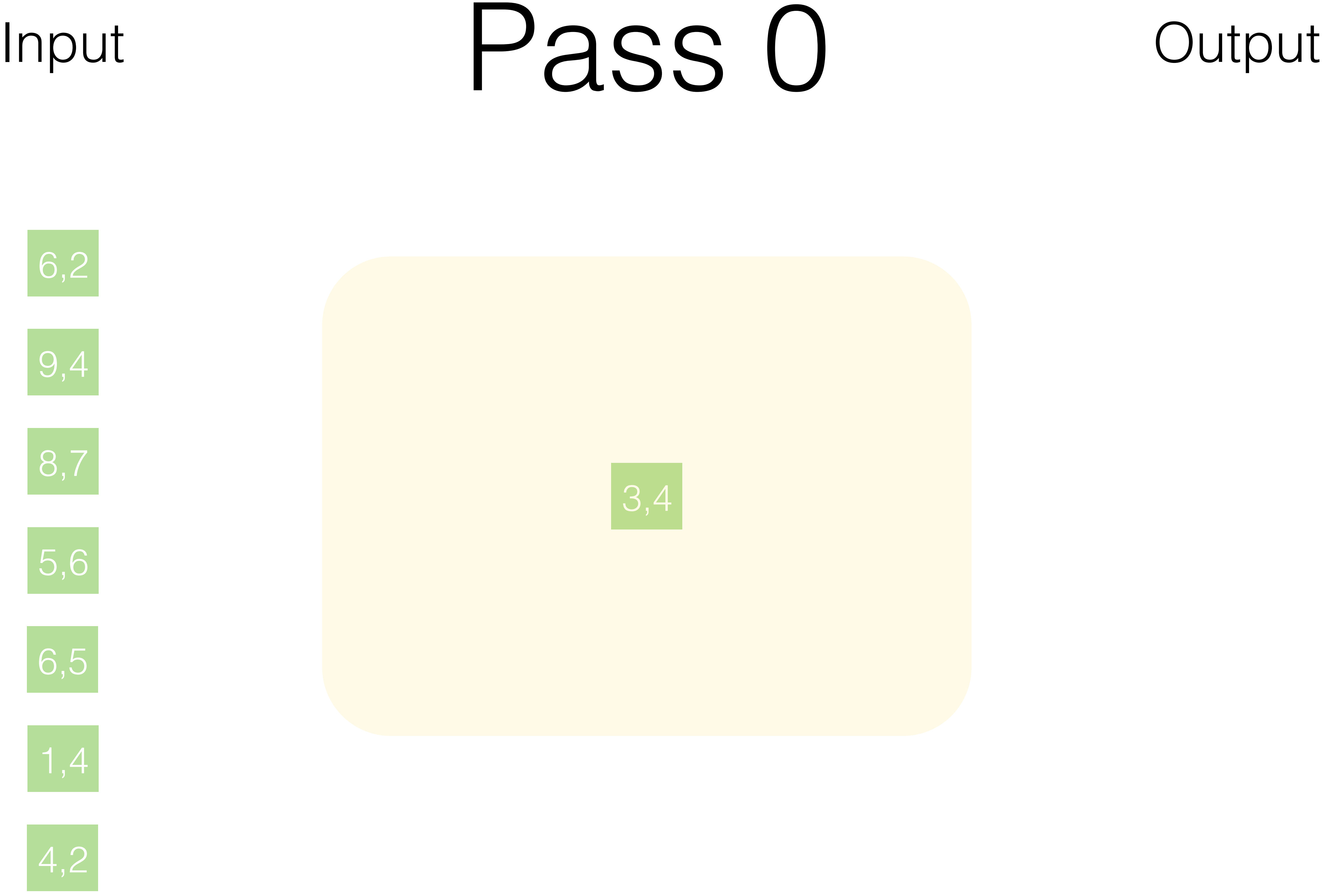
5,6

6,5

1,4

4,2

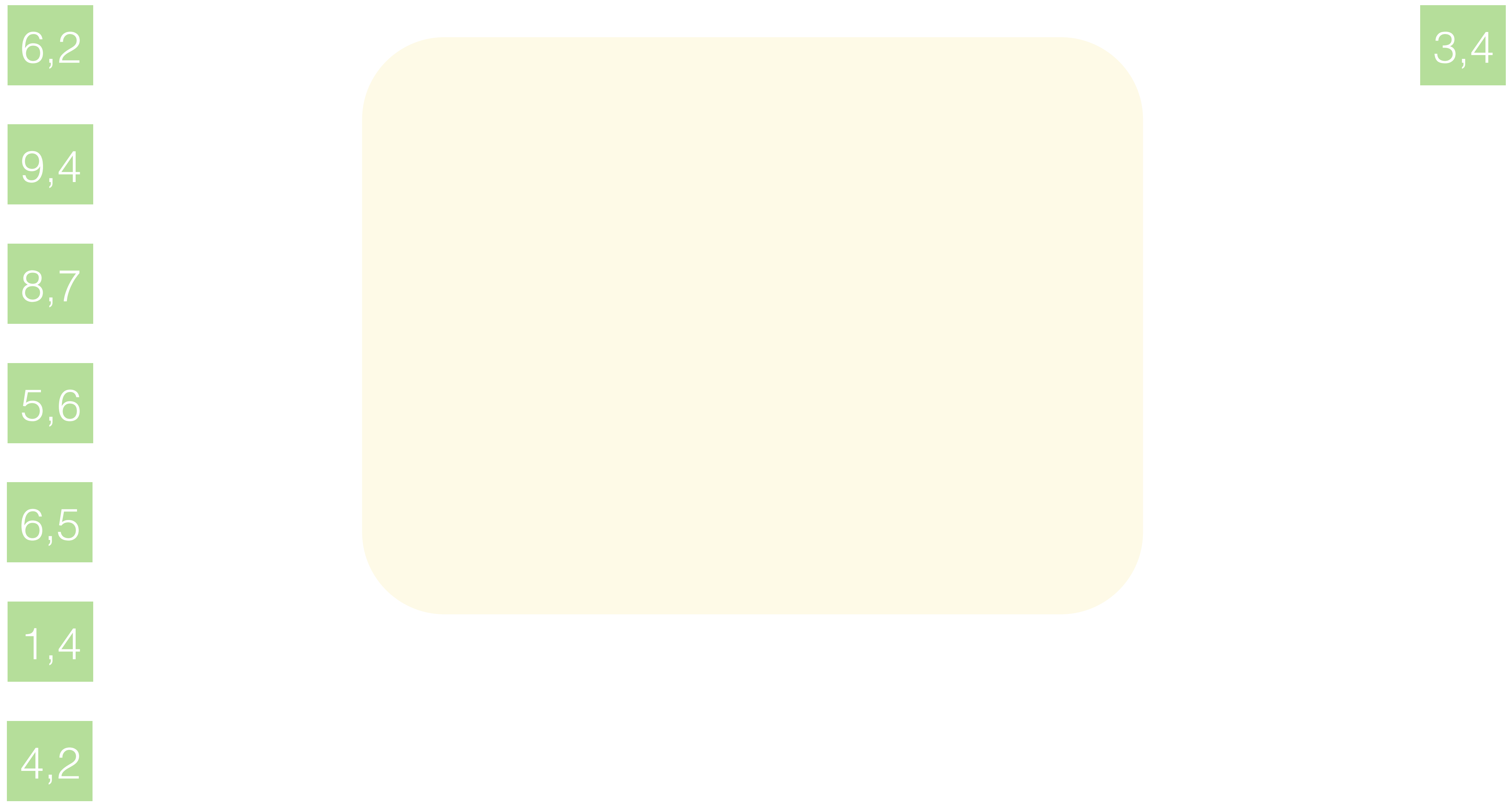
Pass 0



Pass 0

Input

Output



6,2

9,4

8,7

5,6

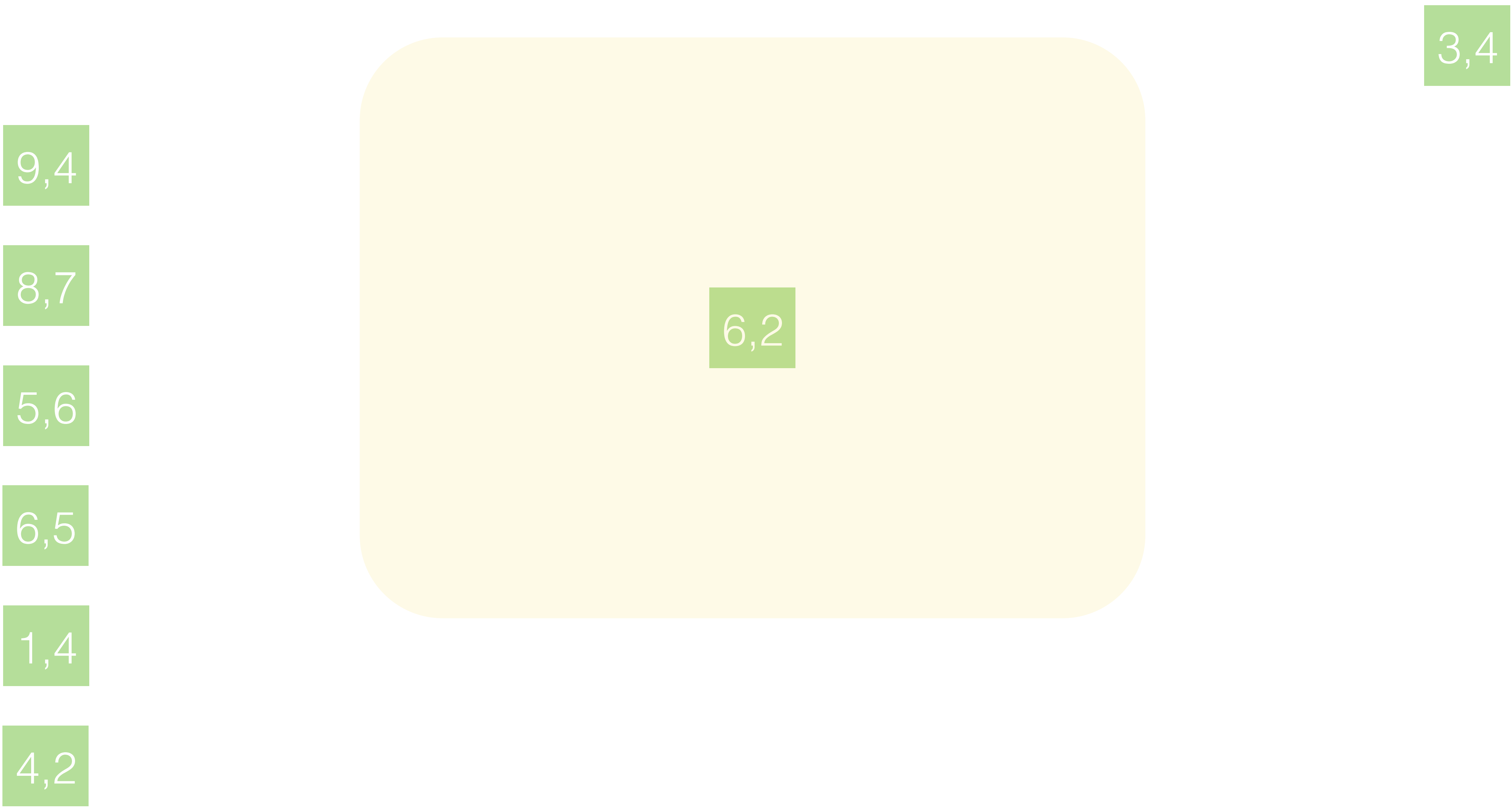
6,5

1,4

4,2

3,4

Pass 0



Input

9,4

8,7

5,6

6,5

1,4

4,2

6,2

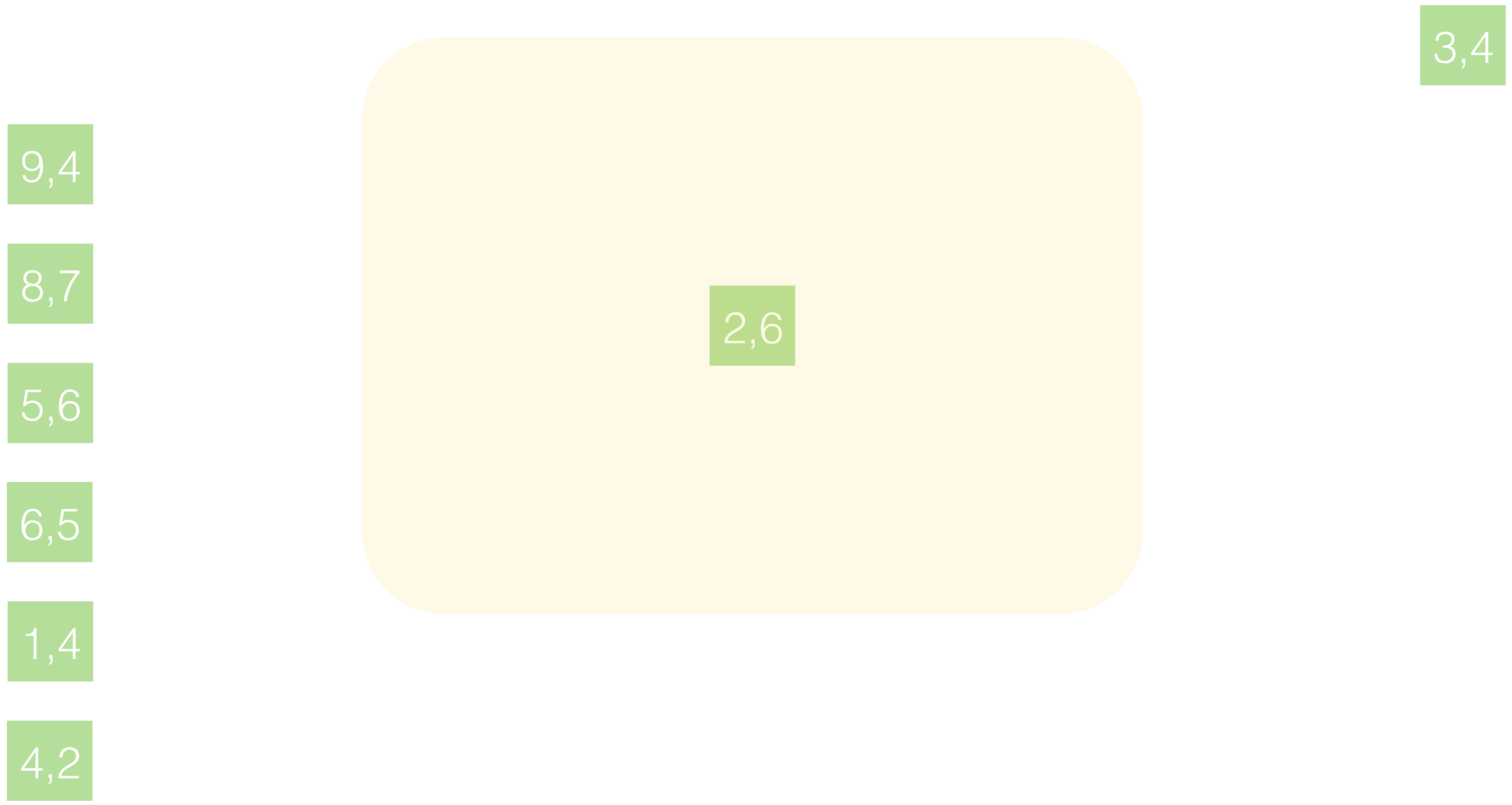
Output

3,4

Pass 0

Input

Output



3,4

9,4

8,7

2,6

5,6

6,5

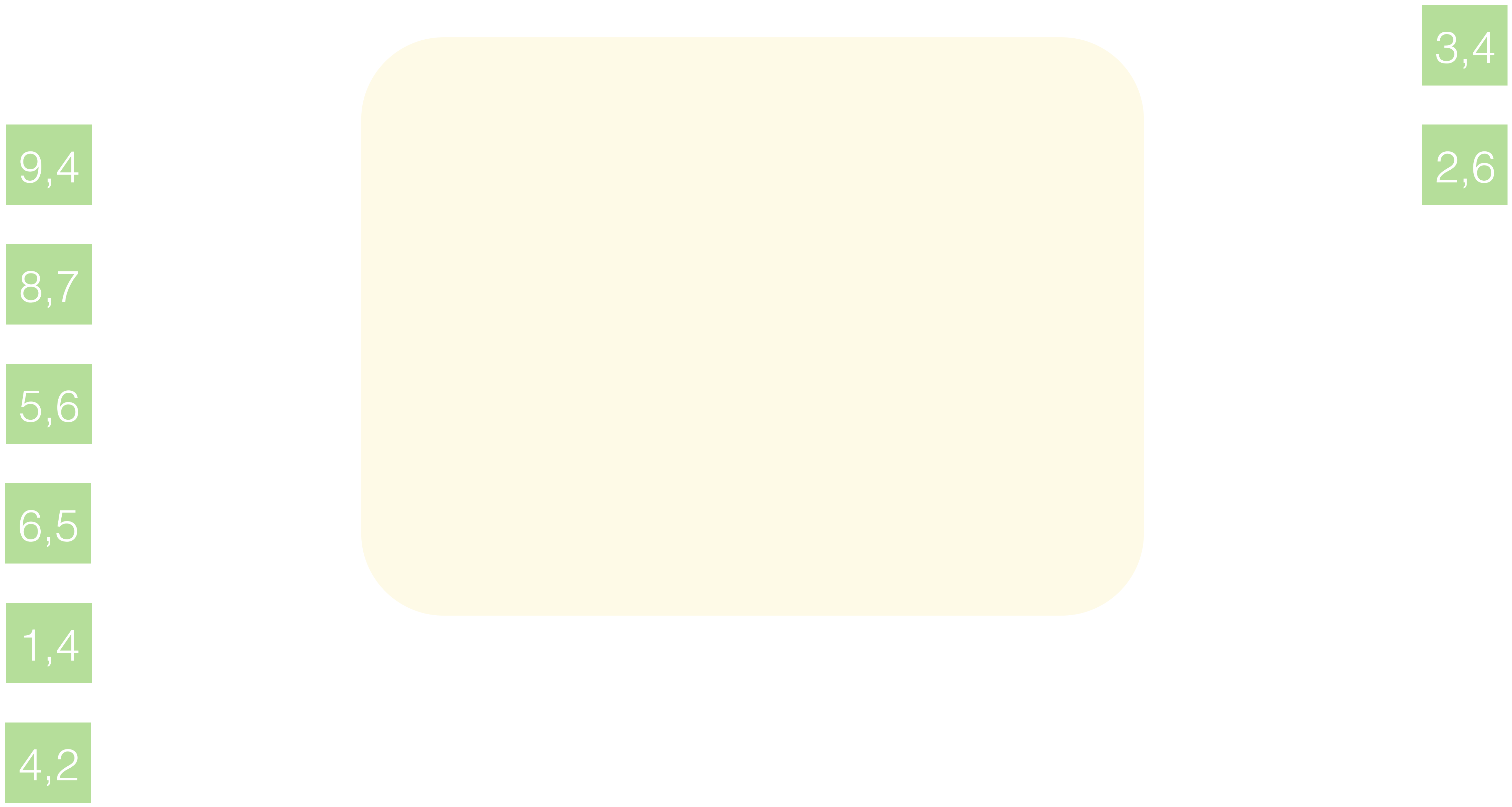
1,4

4,2

Pass 0

Input

Output



3,4

2,6

9,4

8,7

5,6

6,5

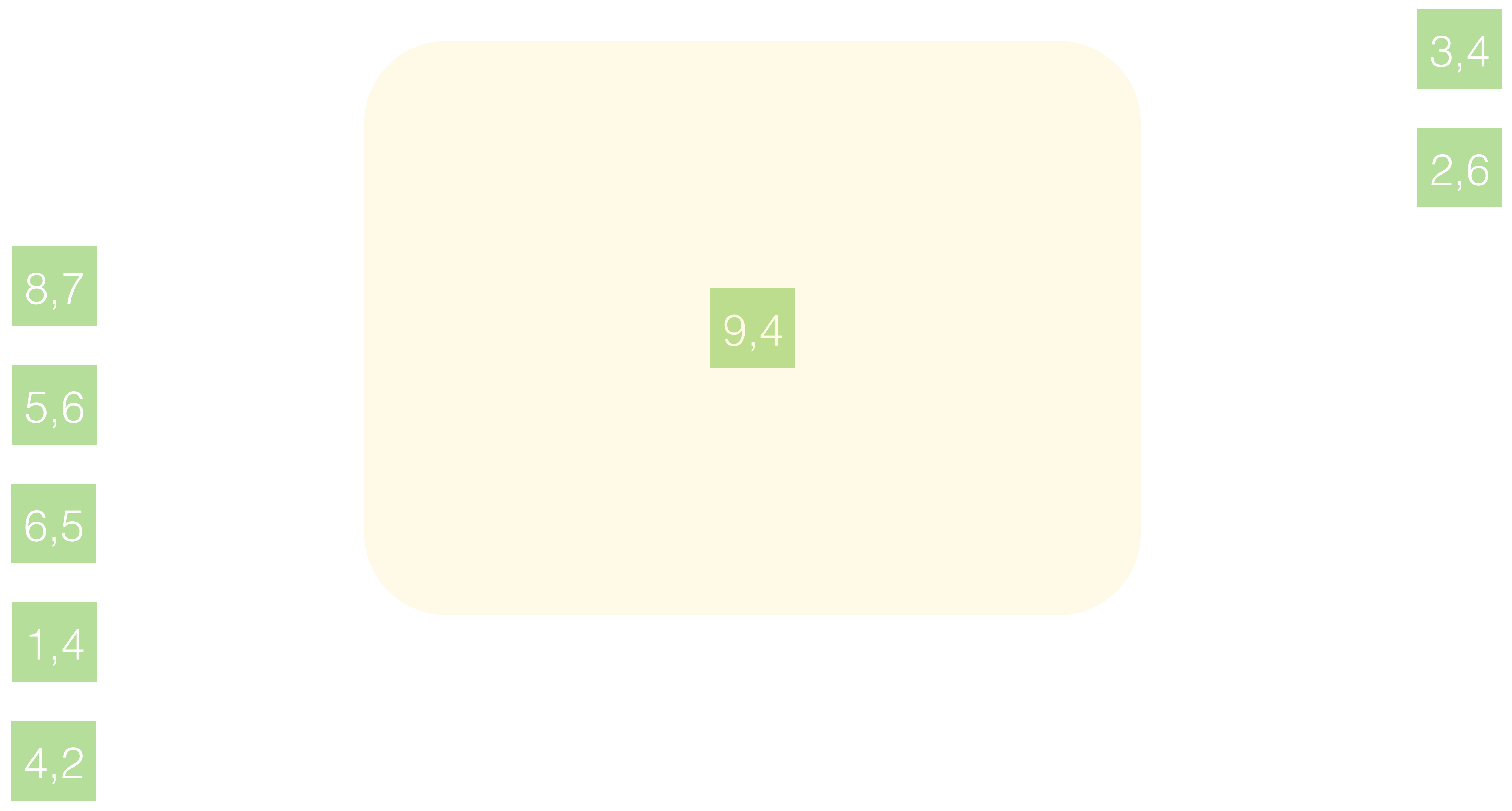
1,4

4,2

Pass 0

Input

Output



8,7

5,6

6,5

1,4

4,2

9,4

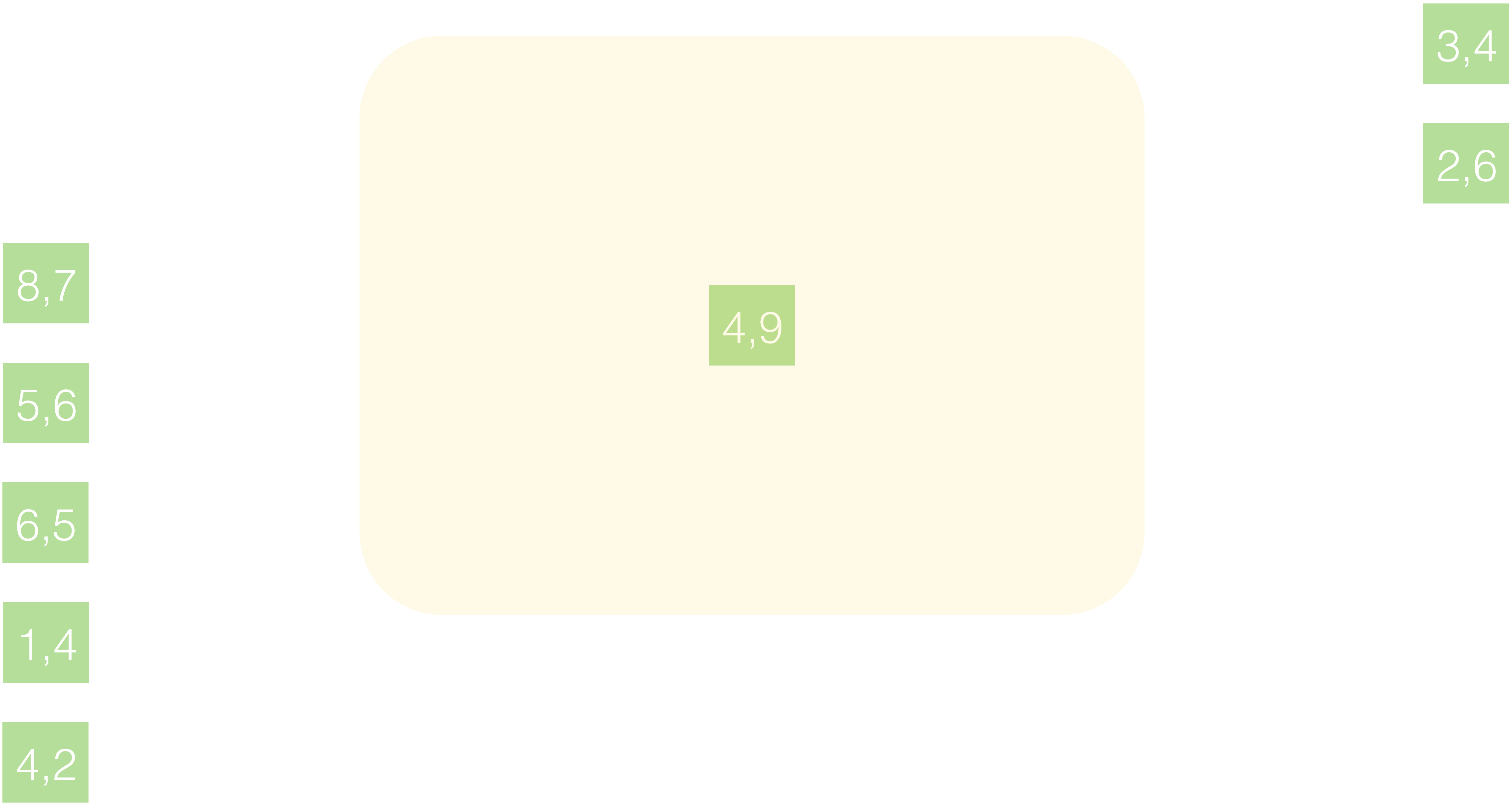
3,4

2,6

Pass 0

Input

Output



3,4

2,6

4,9

8,7

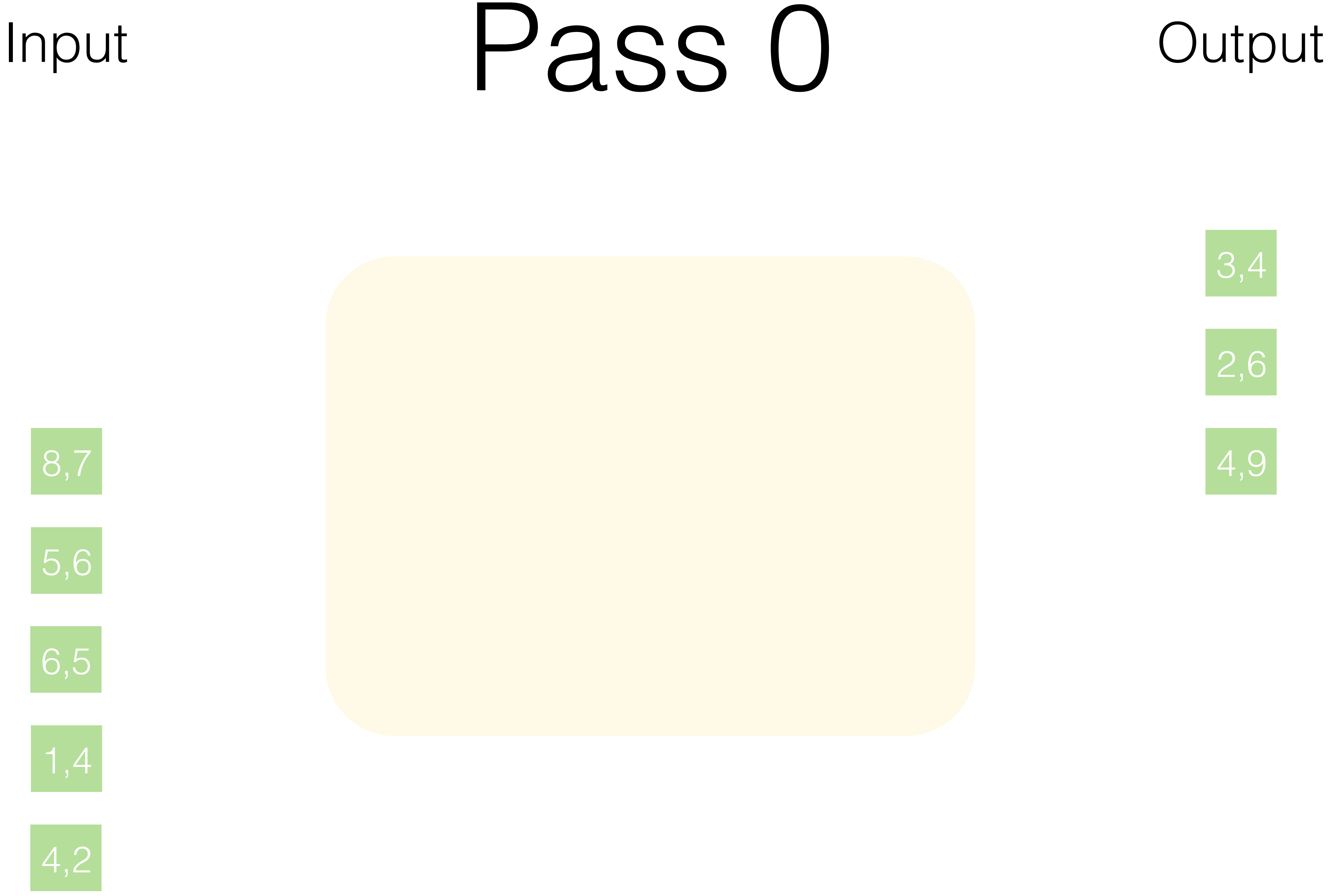
5,6

6,5

1,4

4,2

Pass 0



Input

8,7

5,6

6,5

1,4

4,2

Output

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

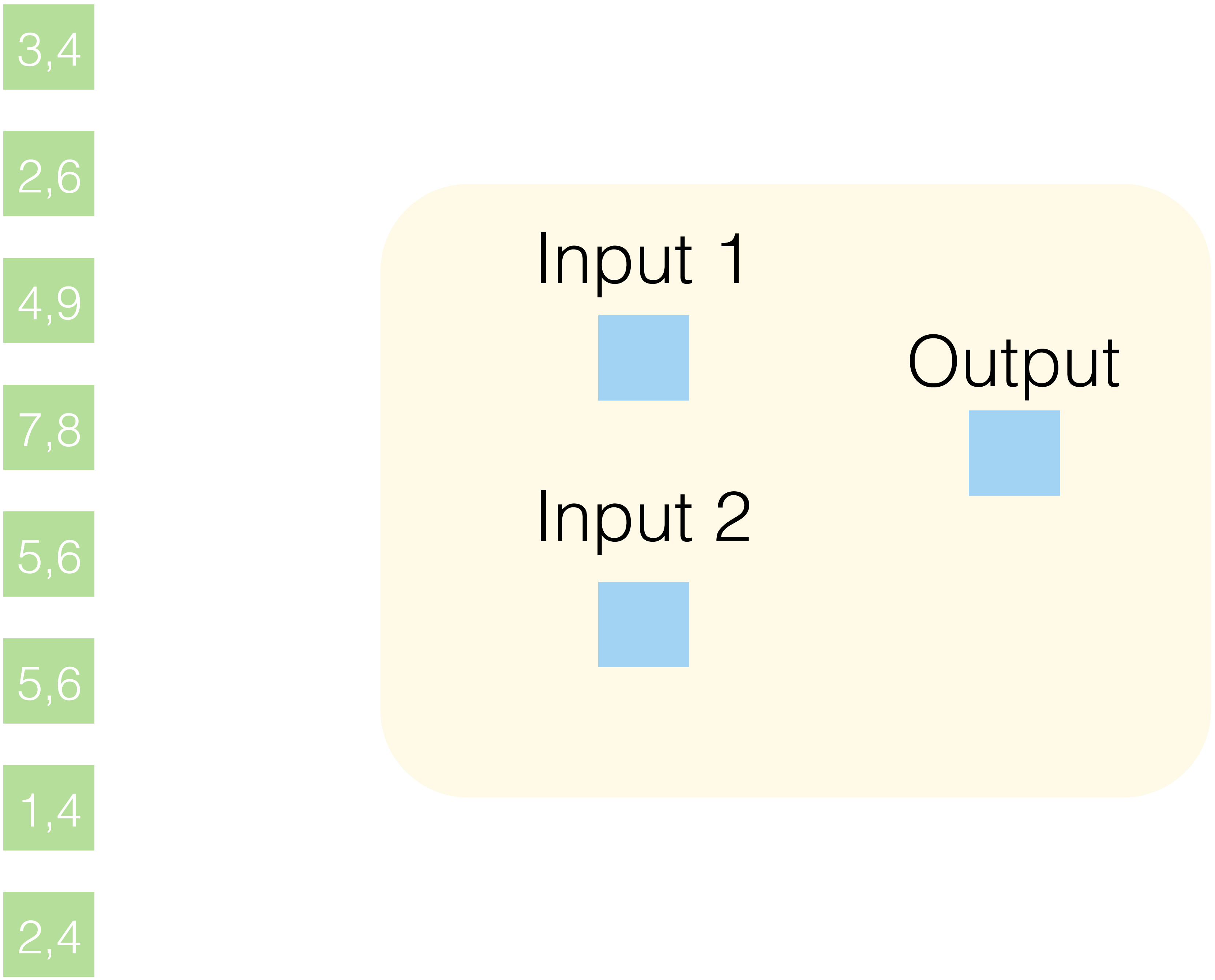
Input	Pass 0
3,4	3,4
6,2	2,6
9,4	4,9
8,7	7,8
5,6	5,6
6,5	5,6
1,4	1,4
4,2	2,4

1 page runs

Pass 1

Input

Output



3,4

2,6

4,9

7,8

5,6

5,6

1,4

2,4

Input 1

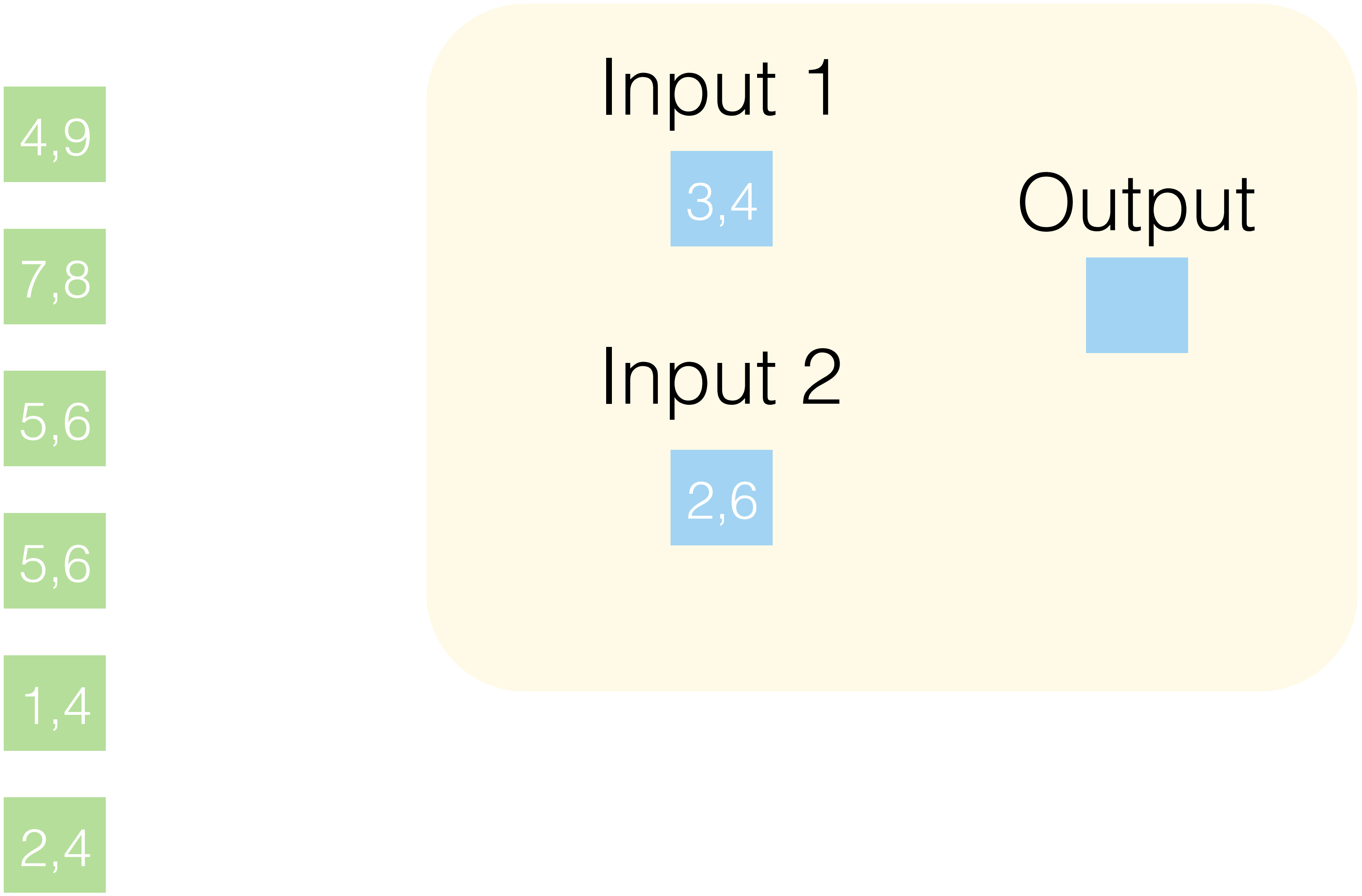
Output

Input 2

Input

Pass 1

Output



4,9

7,8

5,6

5,6

1,4

2,4

Input 1

3,4

Output

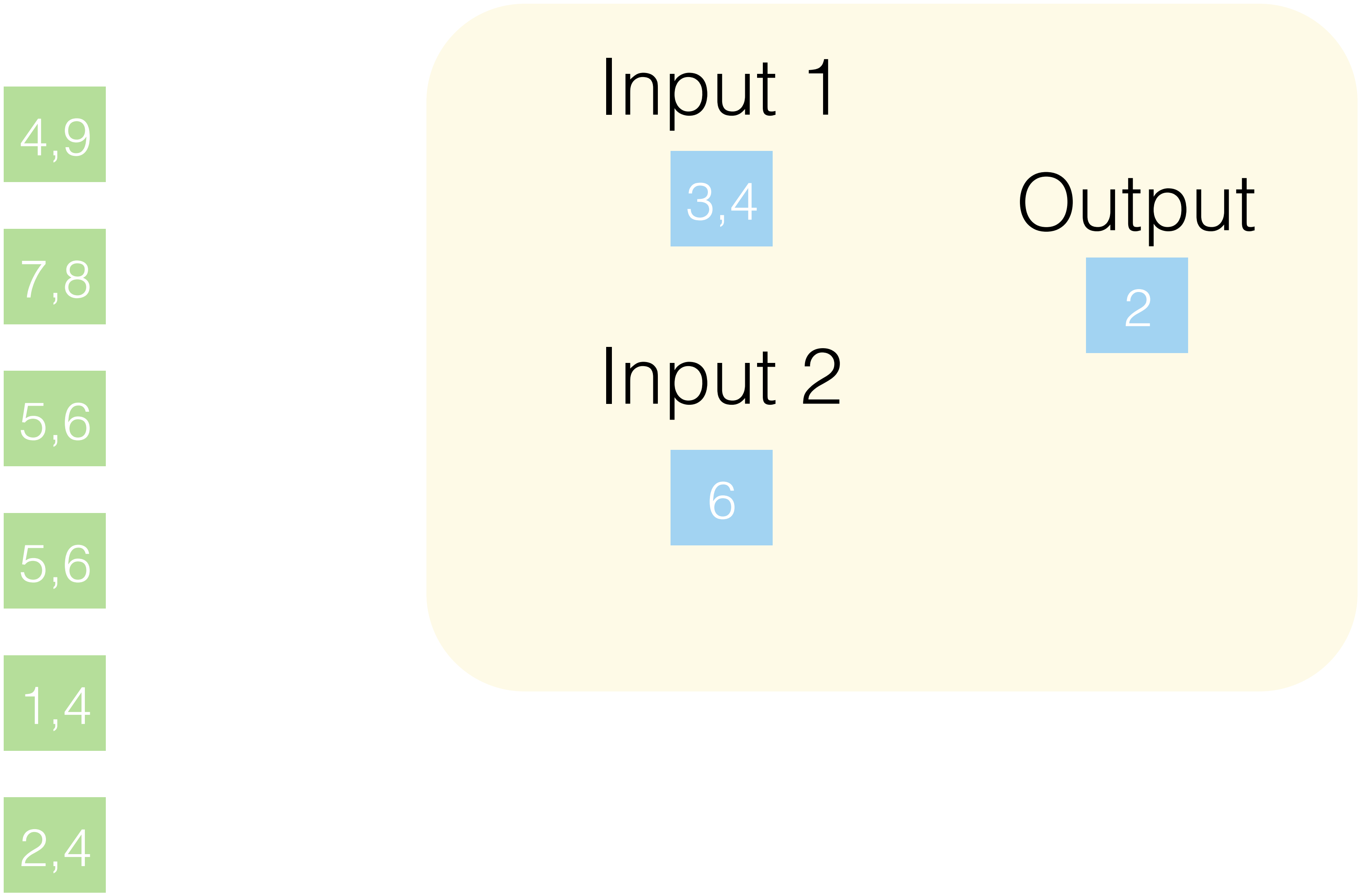
Input 2

2,6

Input

Pass 1

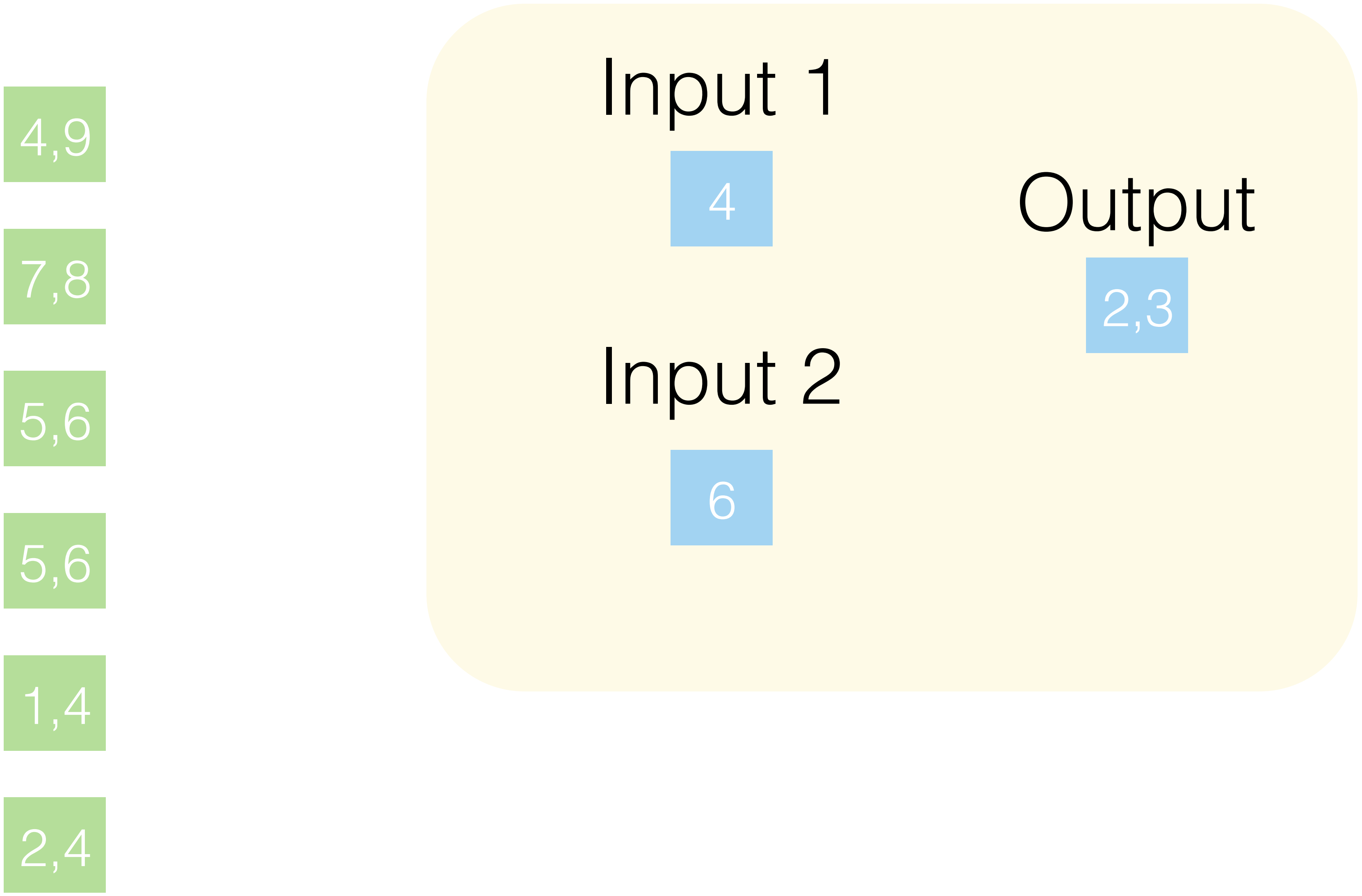
Output



Pass 1

Input

Output



4,9

7,8

5,6

5,6

1,4

2,4

Input 1

4

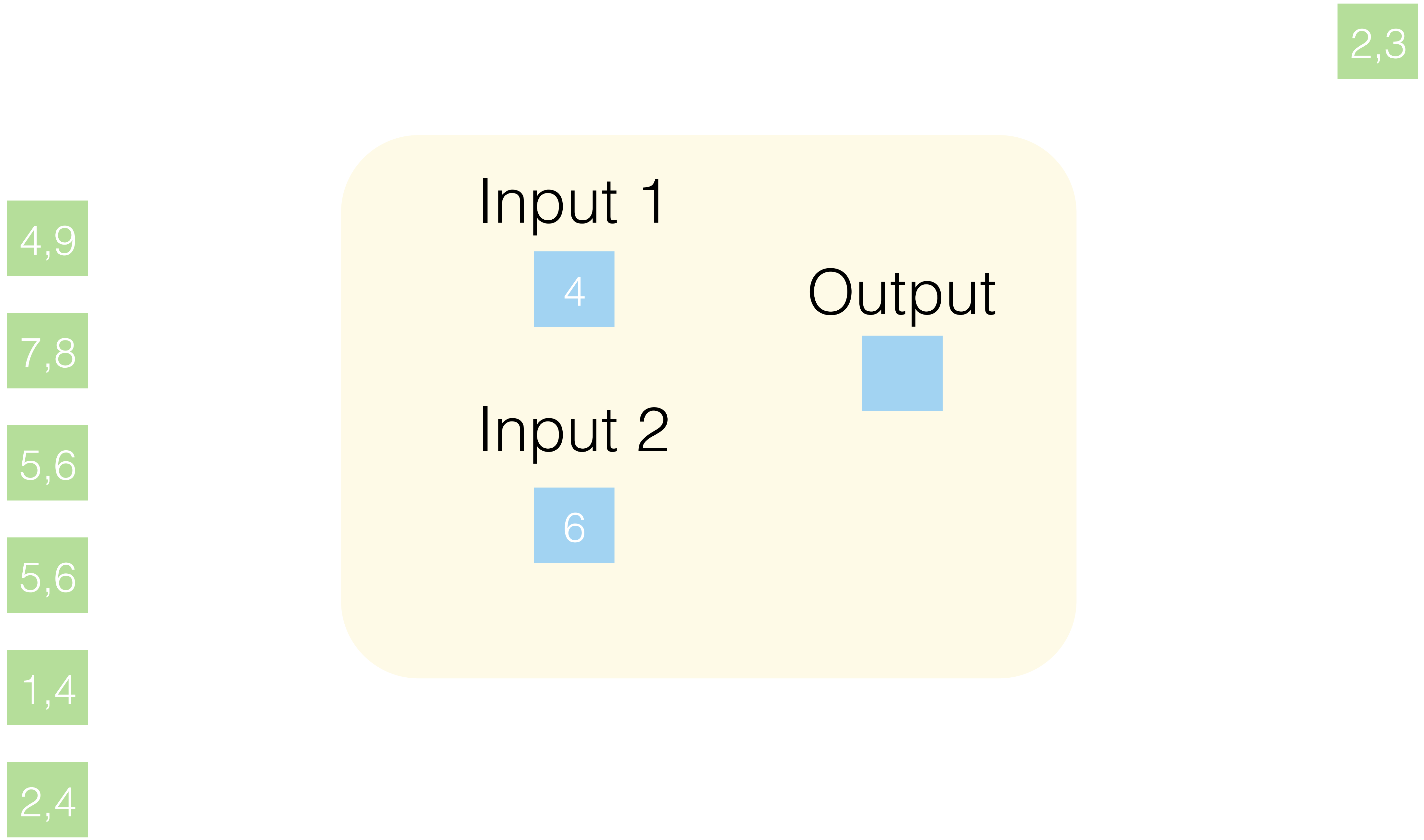
Input 2

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

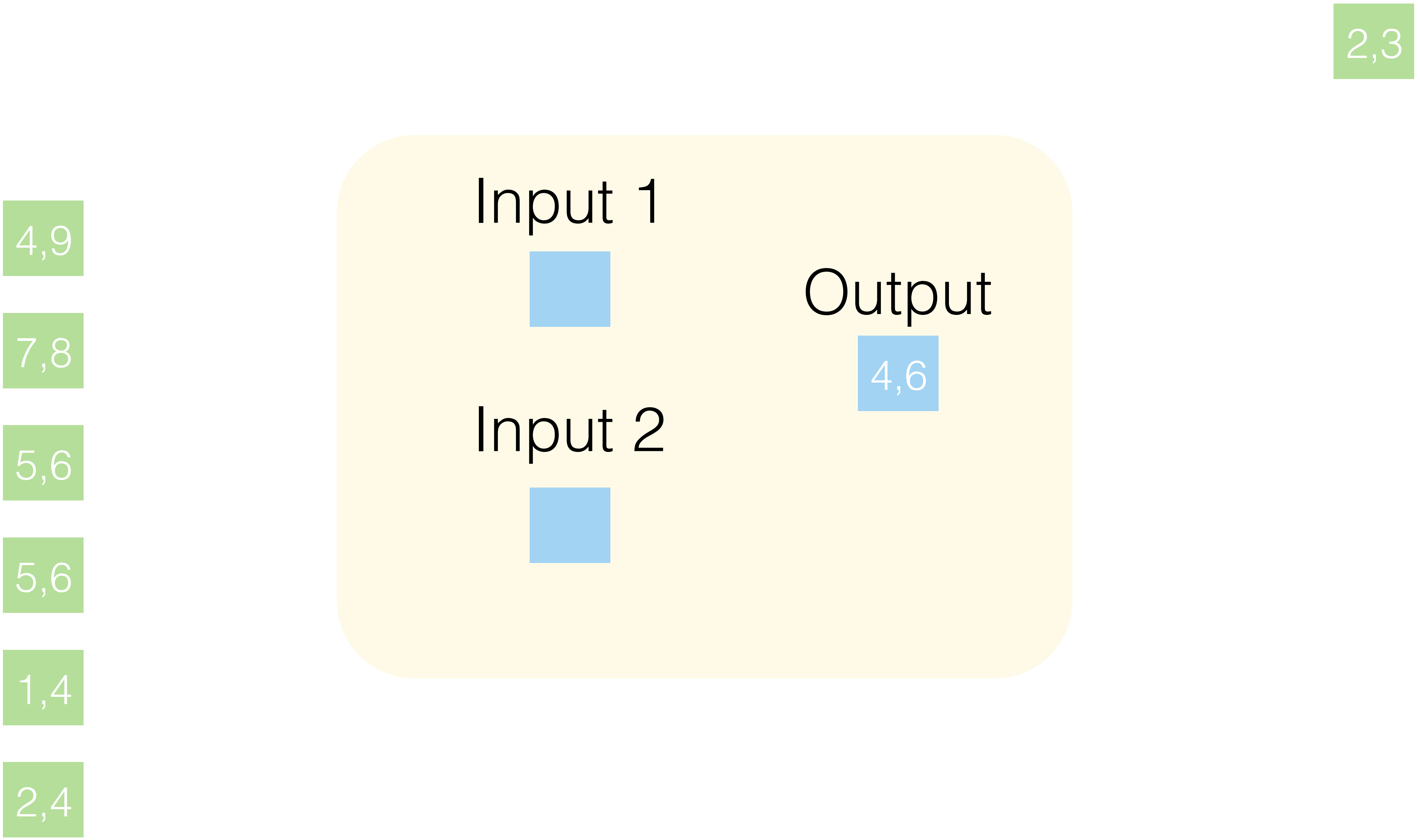
Output

2,3

Pass 1

Input

Output



2,3

4,9

7,8

5,6

5,6

1,4

2,4

Input 1

Output

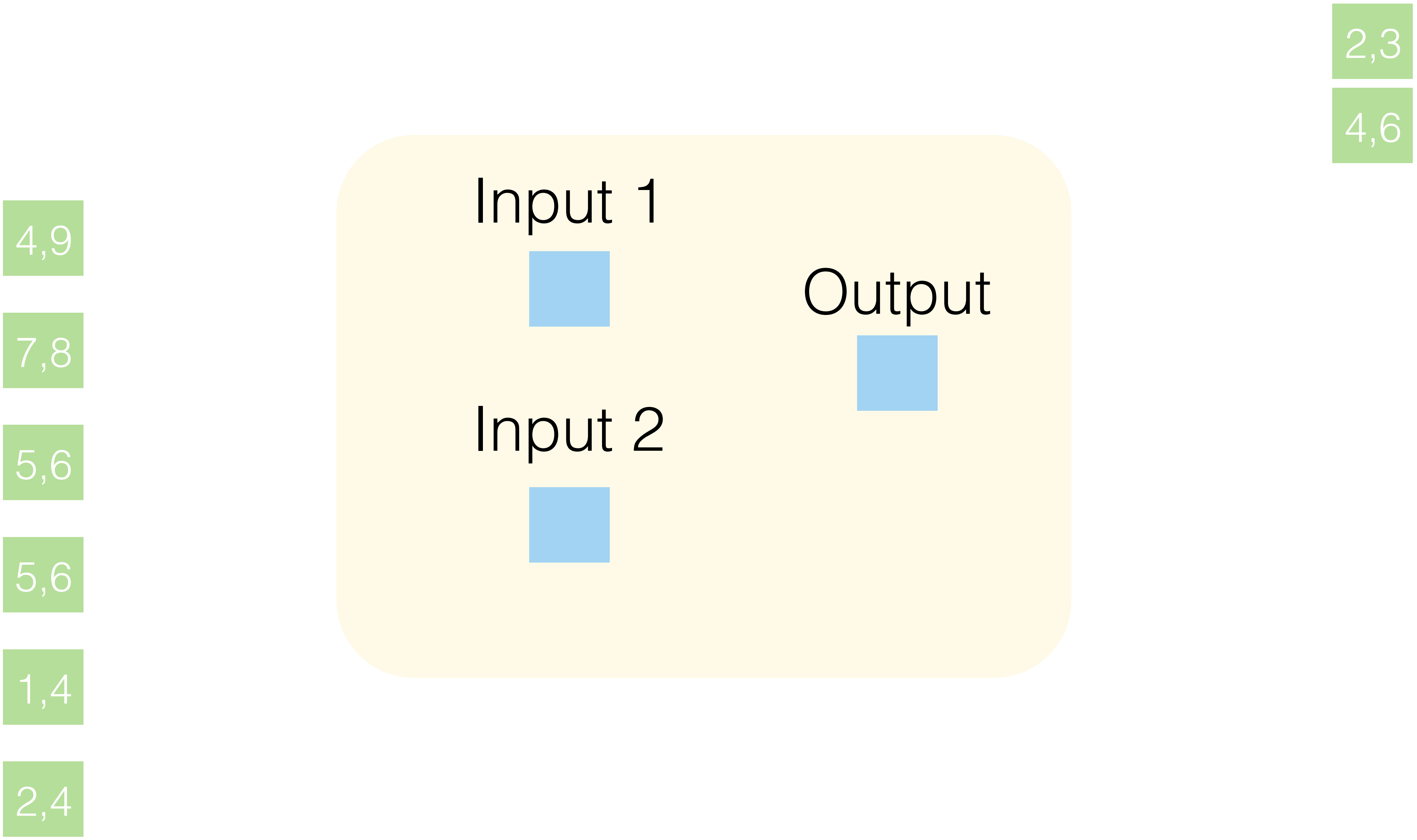
4,6

Input 2

Input

Pass 1

Output



4,9

7,8

5,6

5,6

1,4

2,4

Input 1

Input 2

Output

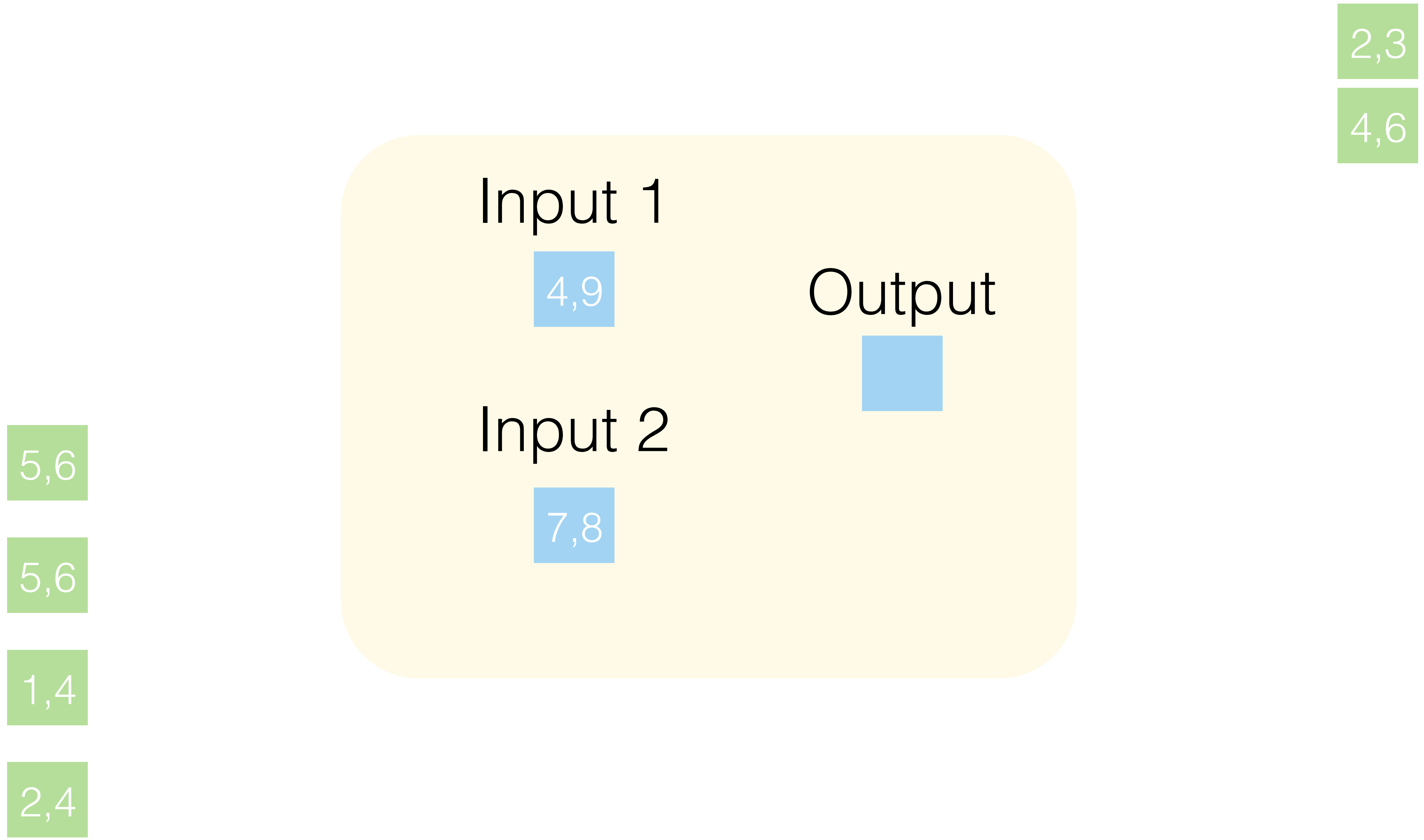
2,3

4,6

Pass 1

Input

Output



2,3

4,6

Input 1

4,9

Output

Input 2

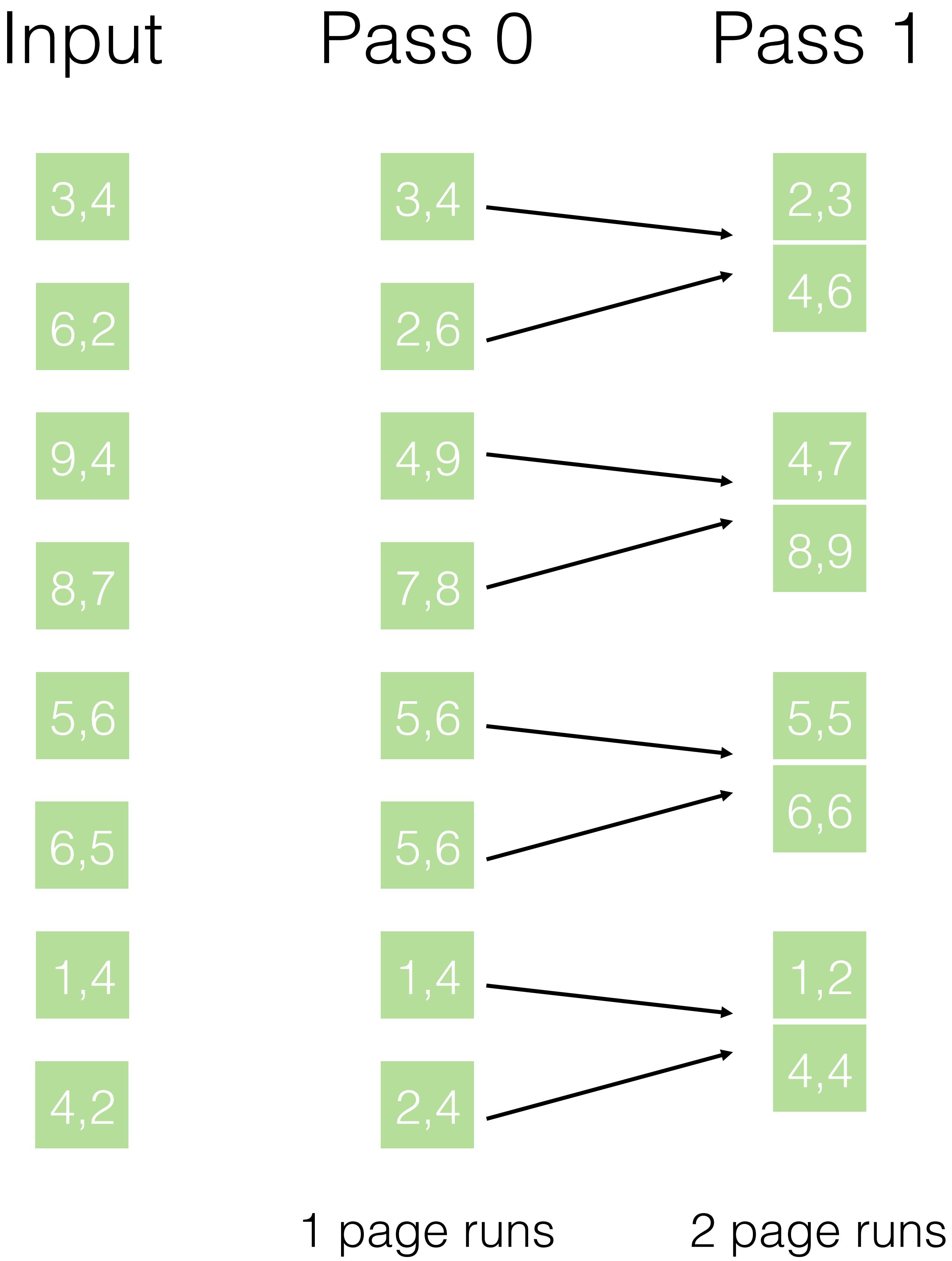
7,8

5,6

5,6

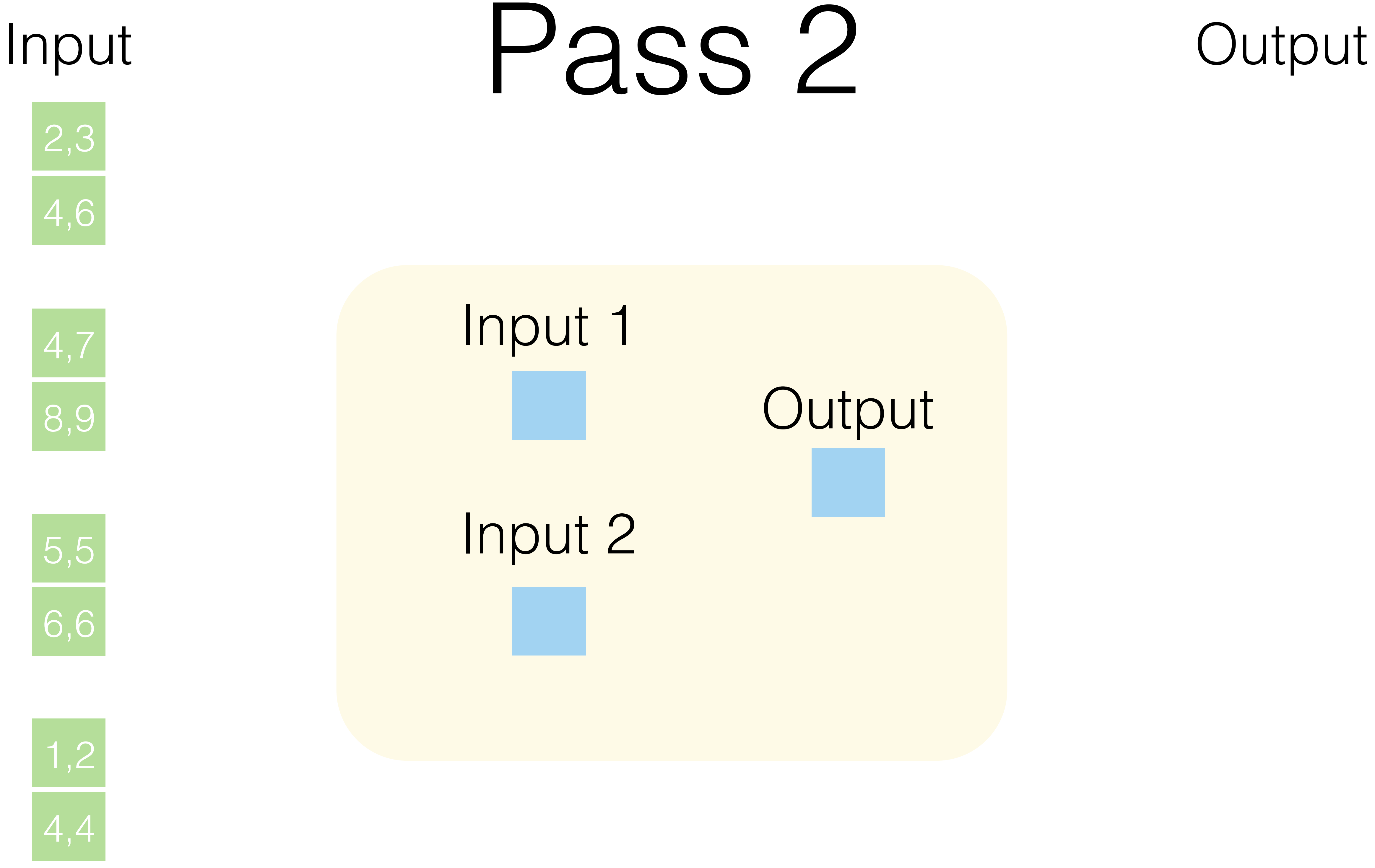
1,4

2,4



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

Pass 2



Input

2,3
4,6

4,7
8,9

5,5
6,6

1,2
4,4

Input 1

Input 2

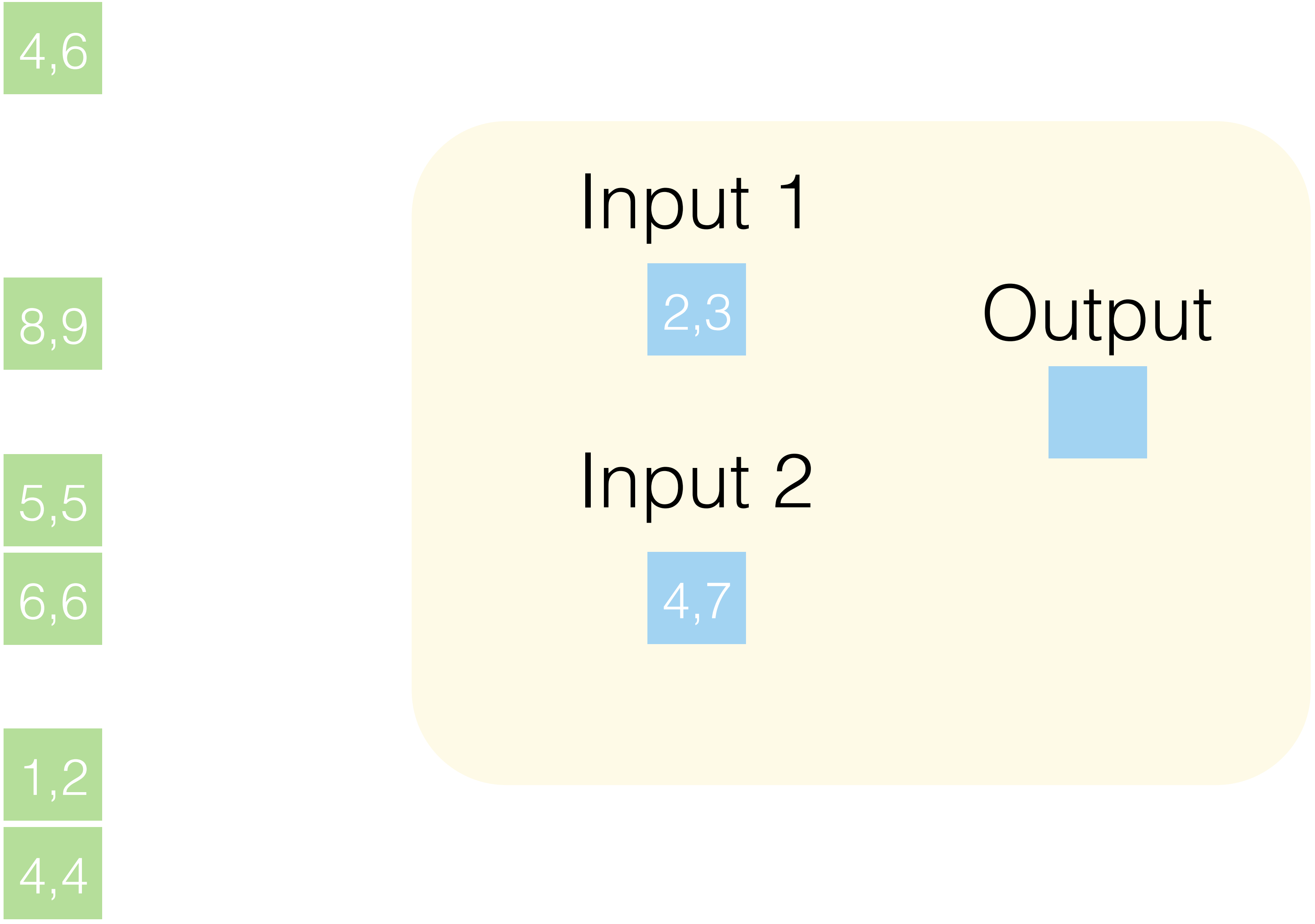
Output

Output

Pass 2

Input

Output



4,6

8,9

5,5

6,6

1,2

4,4

Input 1

2,3

Input 2

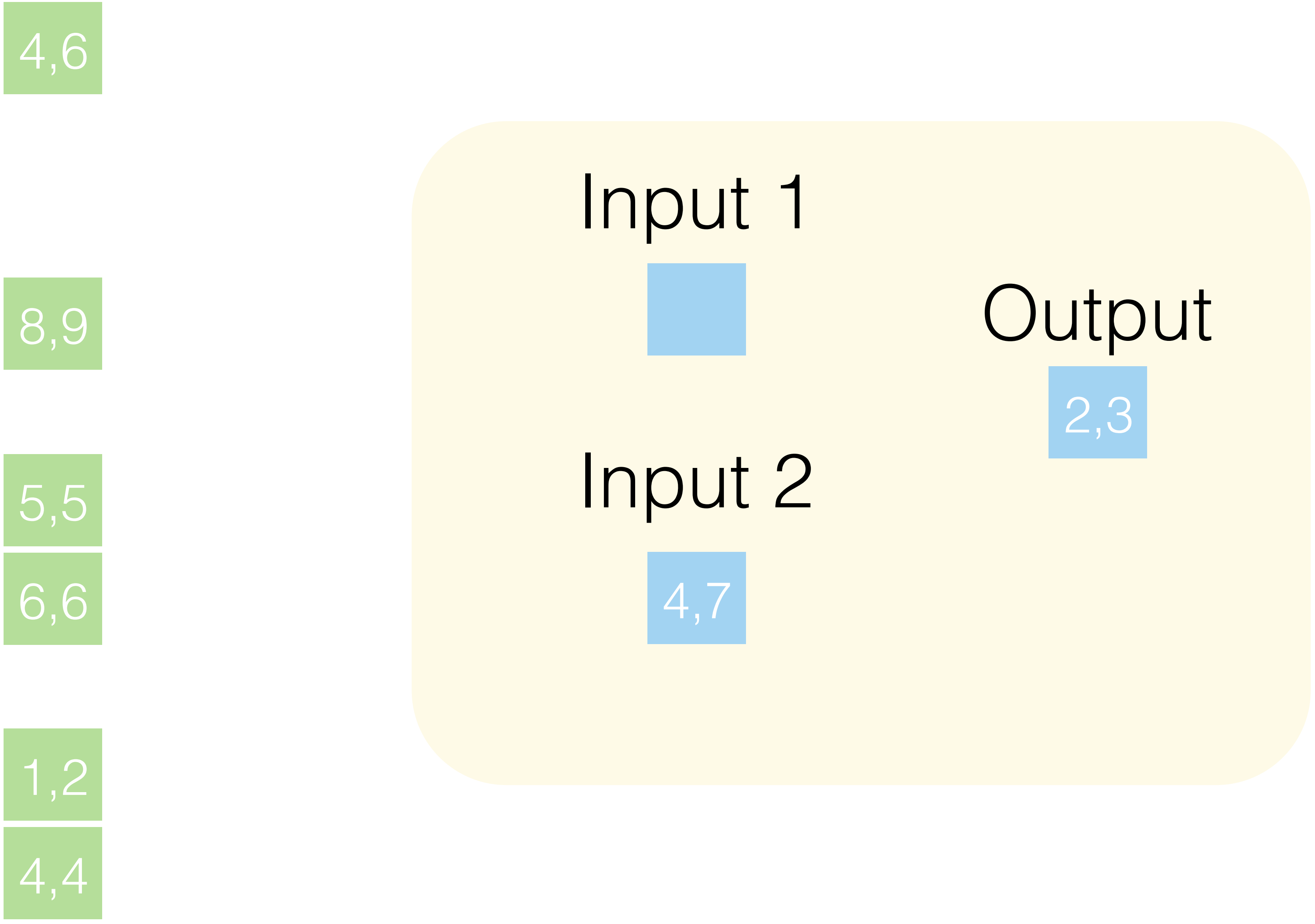
4,7

Output

Input

Pass 2

Output



4,6

8,9

5,5

6,6

1,2

4,4

Input 1

Output

2,3

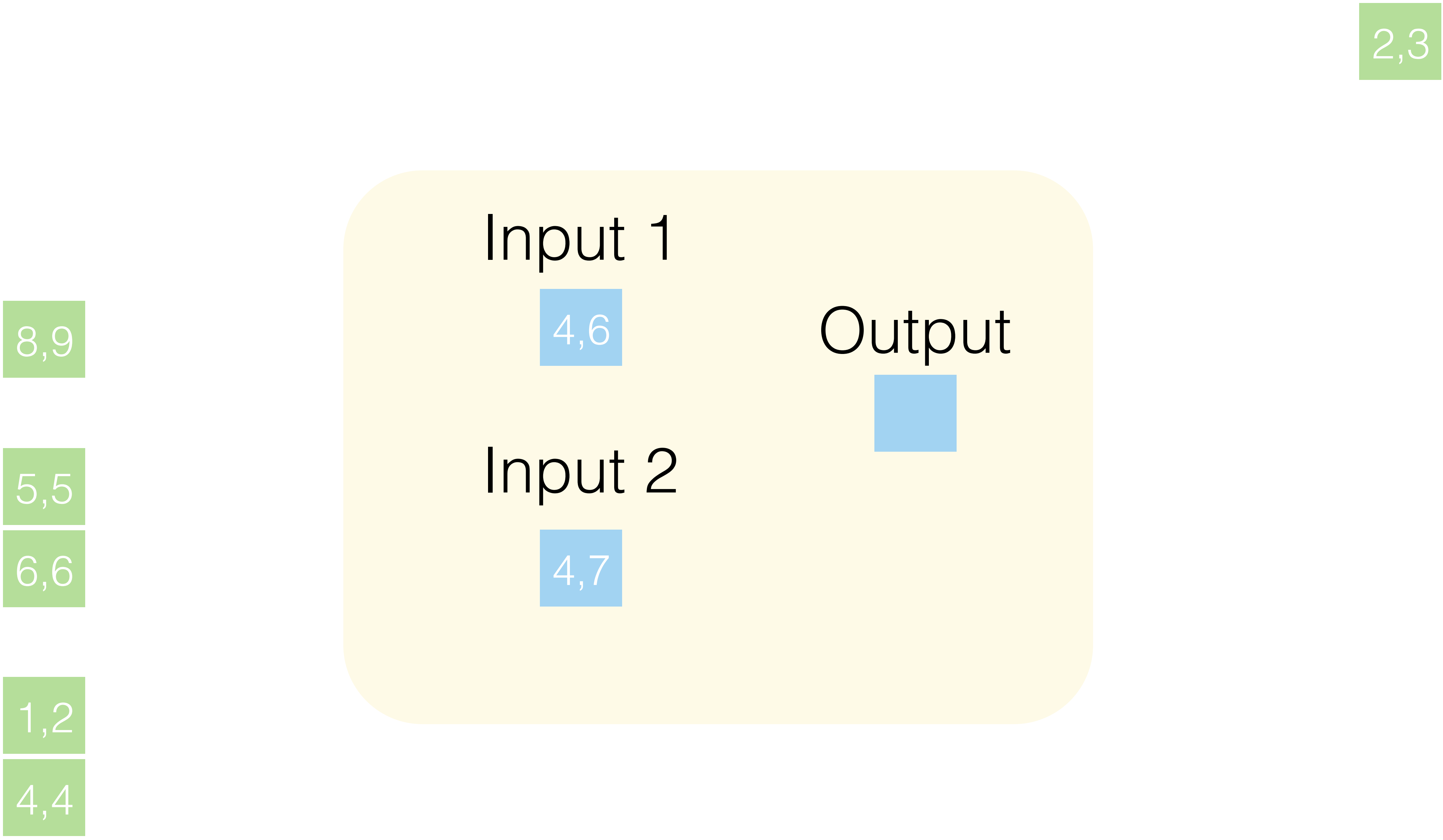
Input 2

4,7

Pass 2

Input

Output



2,3

Input 1

4,6

Output

Input 2

4,7

8,9

5,5

6,6

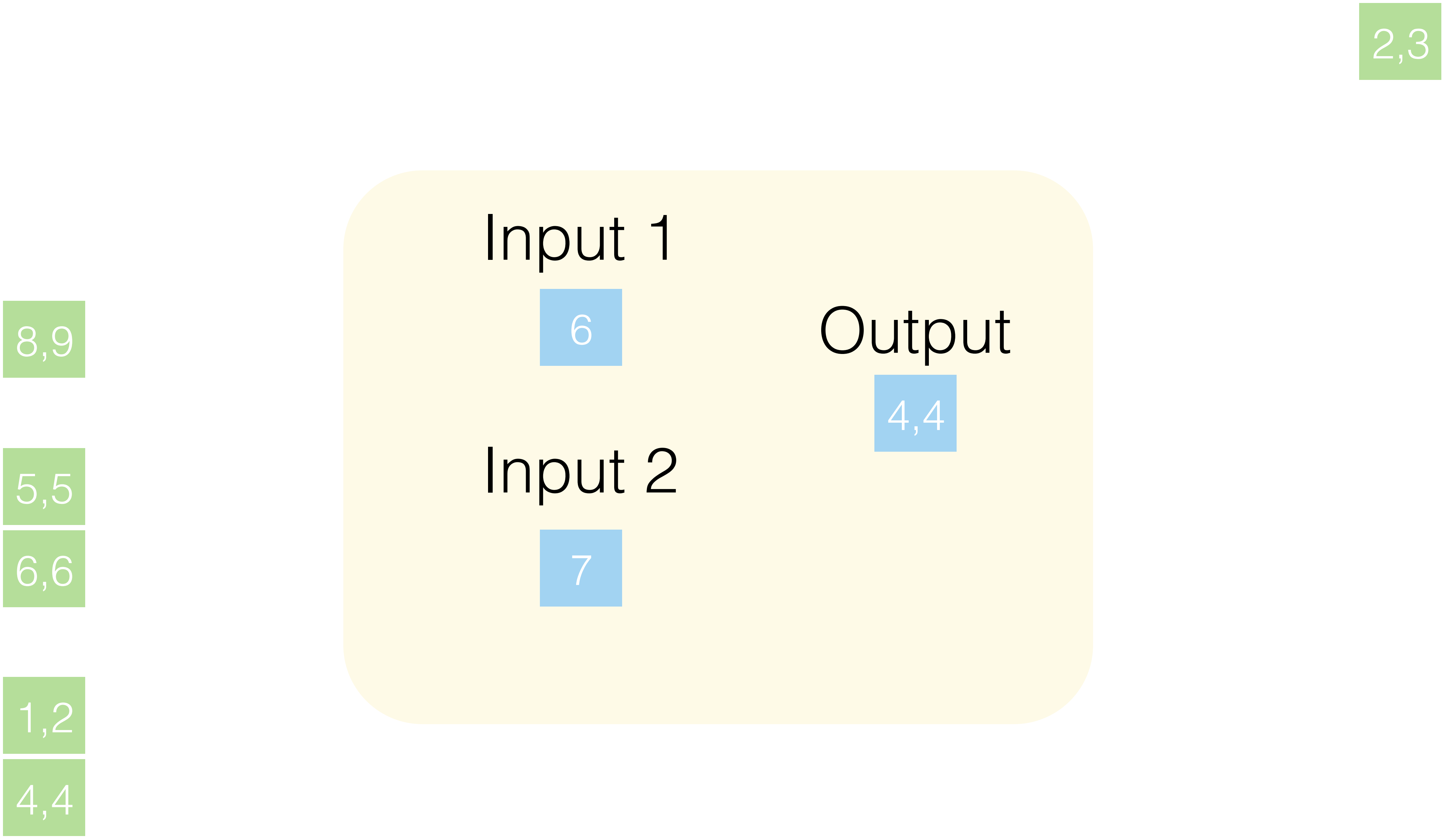
1,2

4,4

Pass 2

Input

Output



8,9

5,5

6,6

1,2

4,4

Input 1

6

Input 2

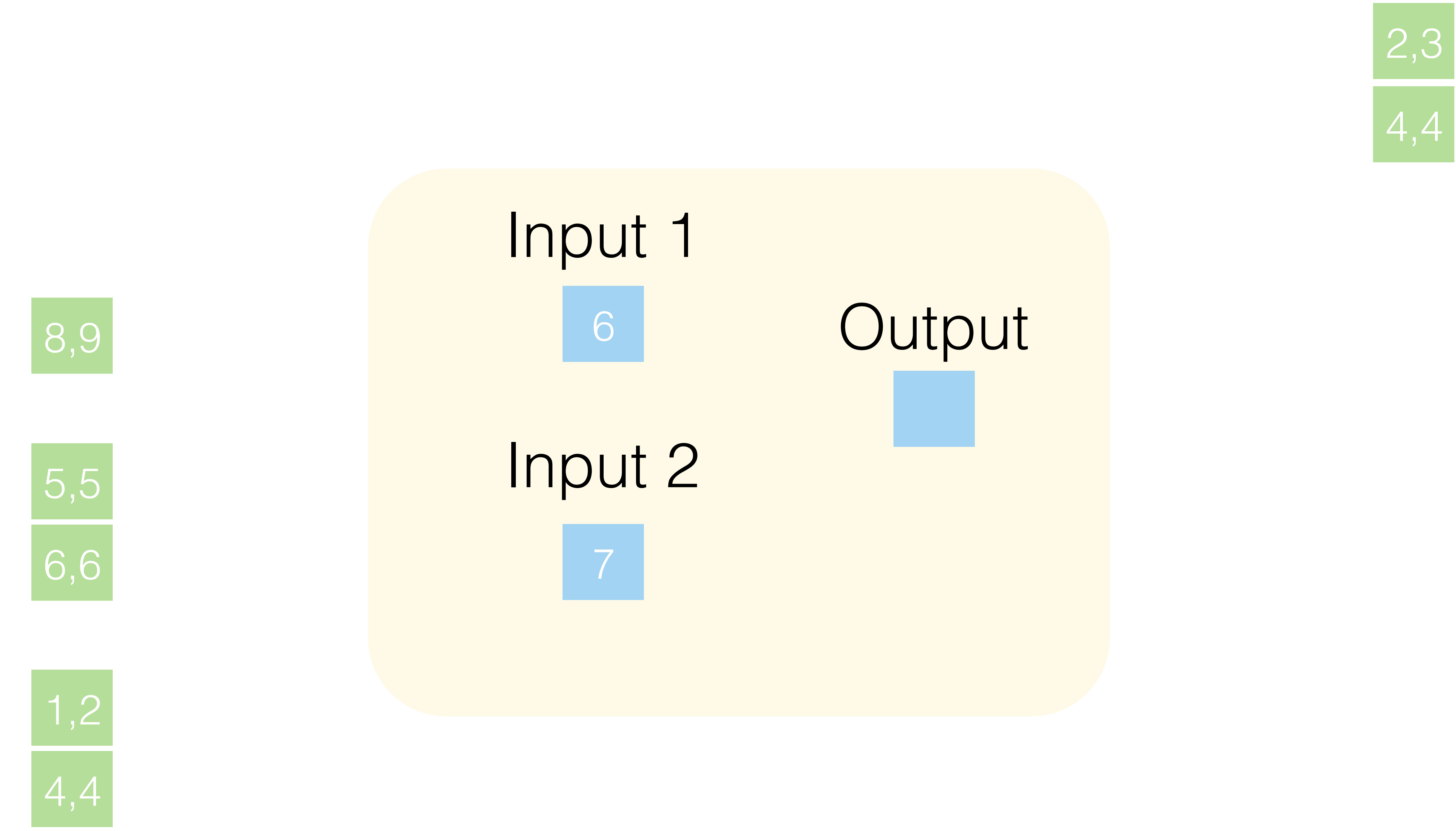
7

Output

4,4

2,3

Pass 2



Input

Output

2,3

4,4

Input 1

6

Output

Input 2

7

8,9

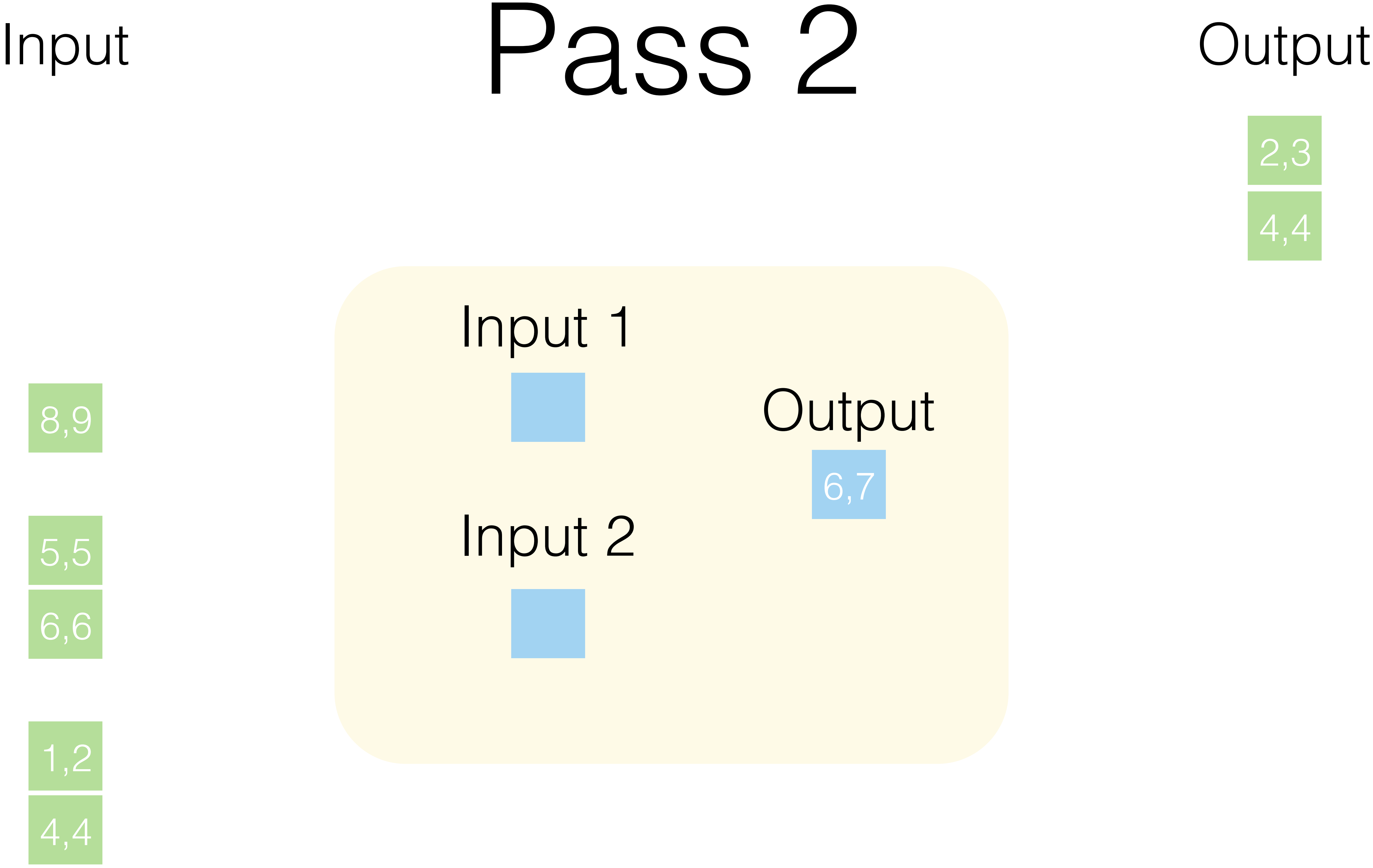
5,5

6,6

1,2

4,4

Pass 2



Input

8,9

5,5

6,6

1,2

4,4

Input 1

Input 2

Output

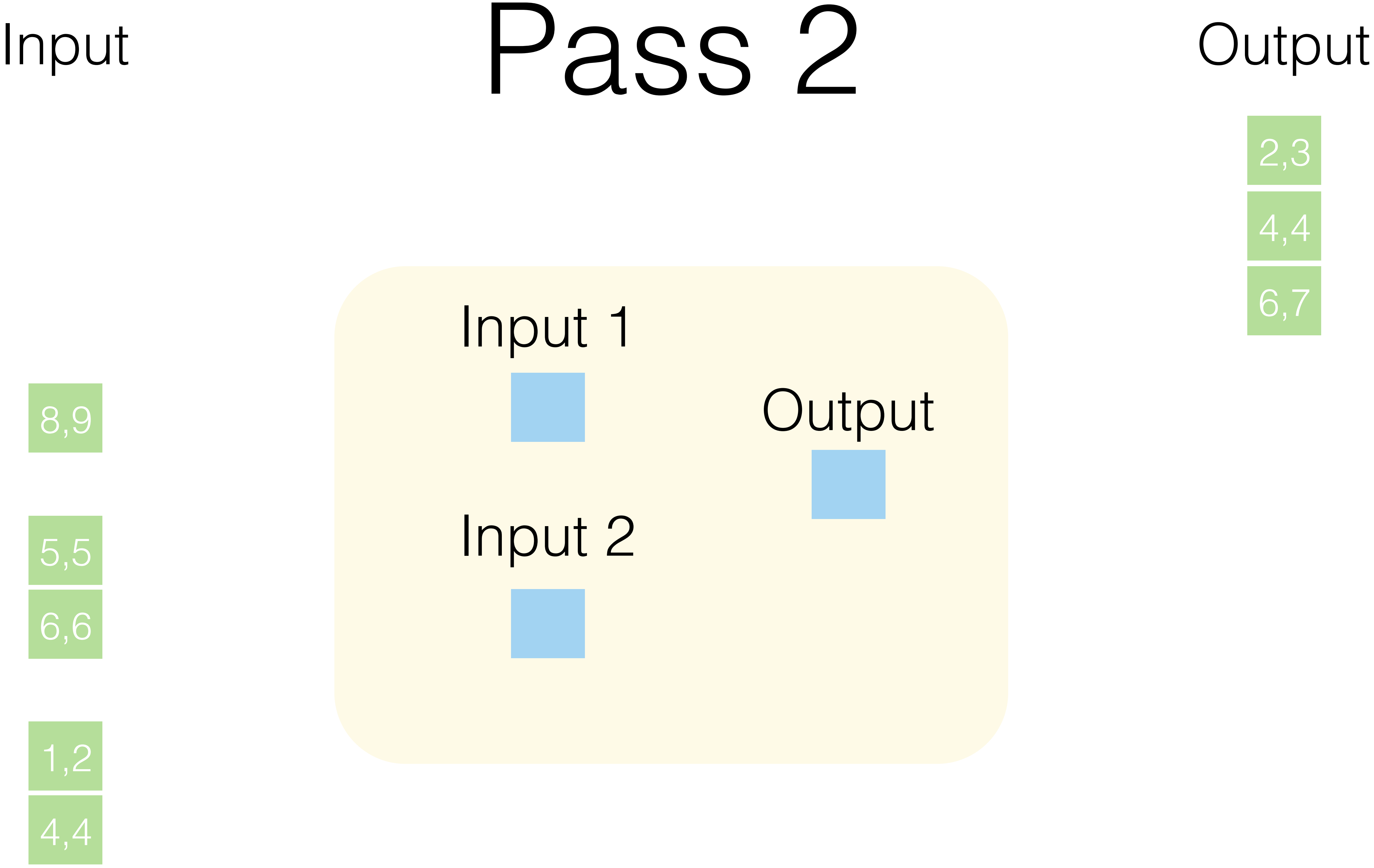
6,7

Output

2,3

4,4

Pass 2



Input

8,9

5,5
6,6

1,2
4,4

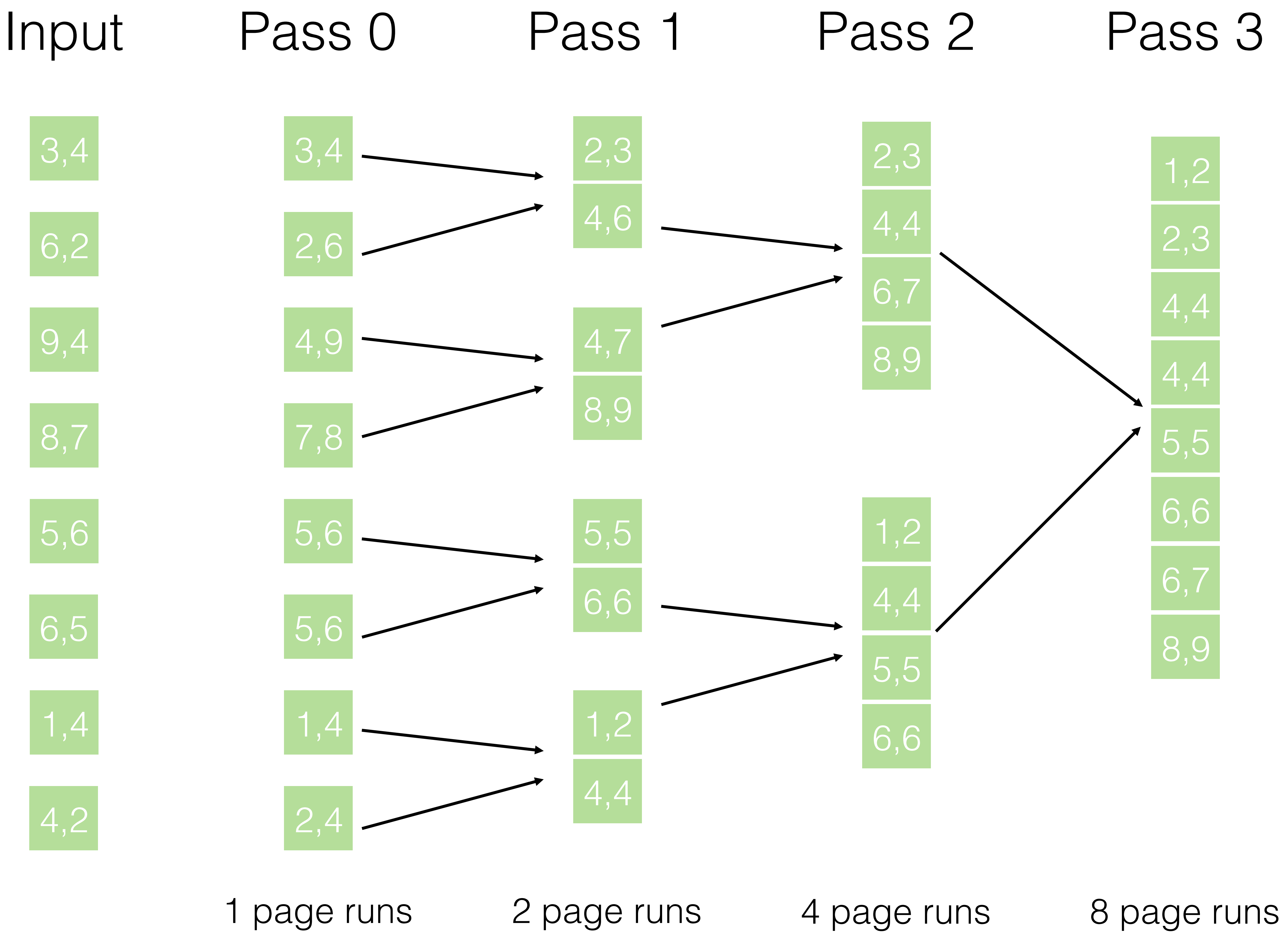
Input 1

Input 2

Output

Output

2,3
4,4
6,7



Input

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

Pass 1

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

2 page runs

Pass 2

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

4 page runs

Pass 3

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

8 page runs

2-Way Merge Sort

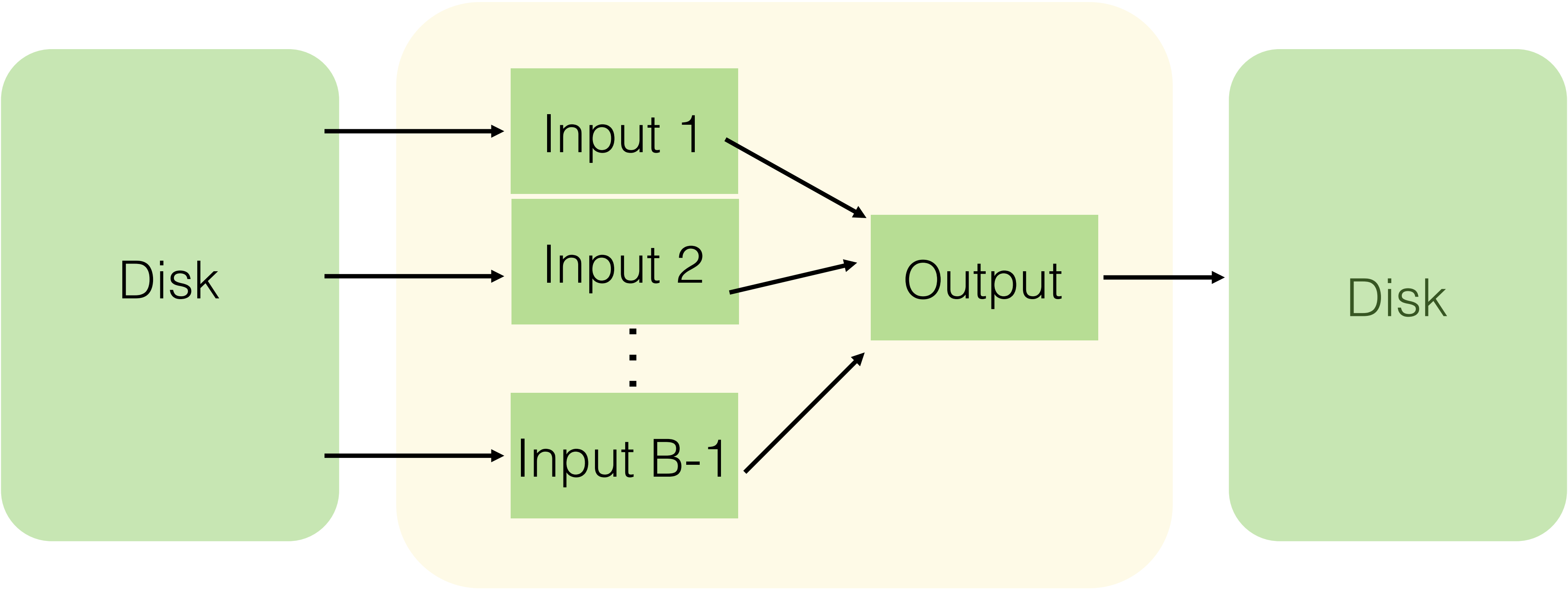
- Number of Passes:
  - $\lceil \log_2 N \rceil + 1$
- Number of I/O's:
  - $2N * (\lceil \log_2 N \rceil + 1)$

Generalized Merge Sort

- Pass 0 (Conquer):
  - Sort B page at a time in memory
- Pass 1, 2, …, etc. (Merge):
  - Merge B-1 runs

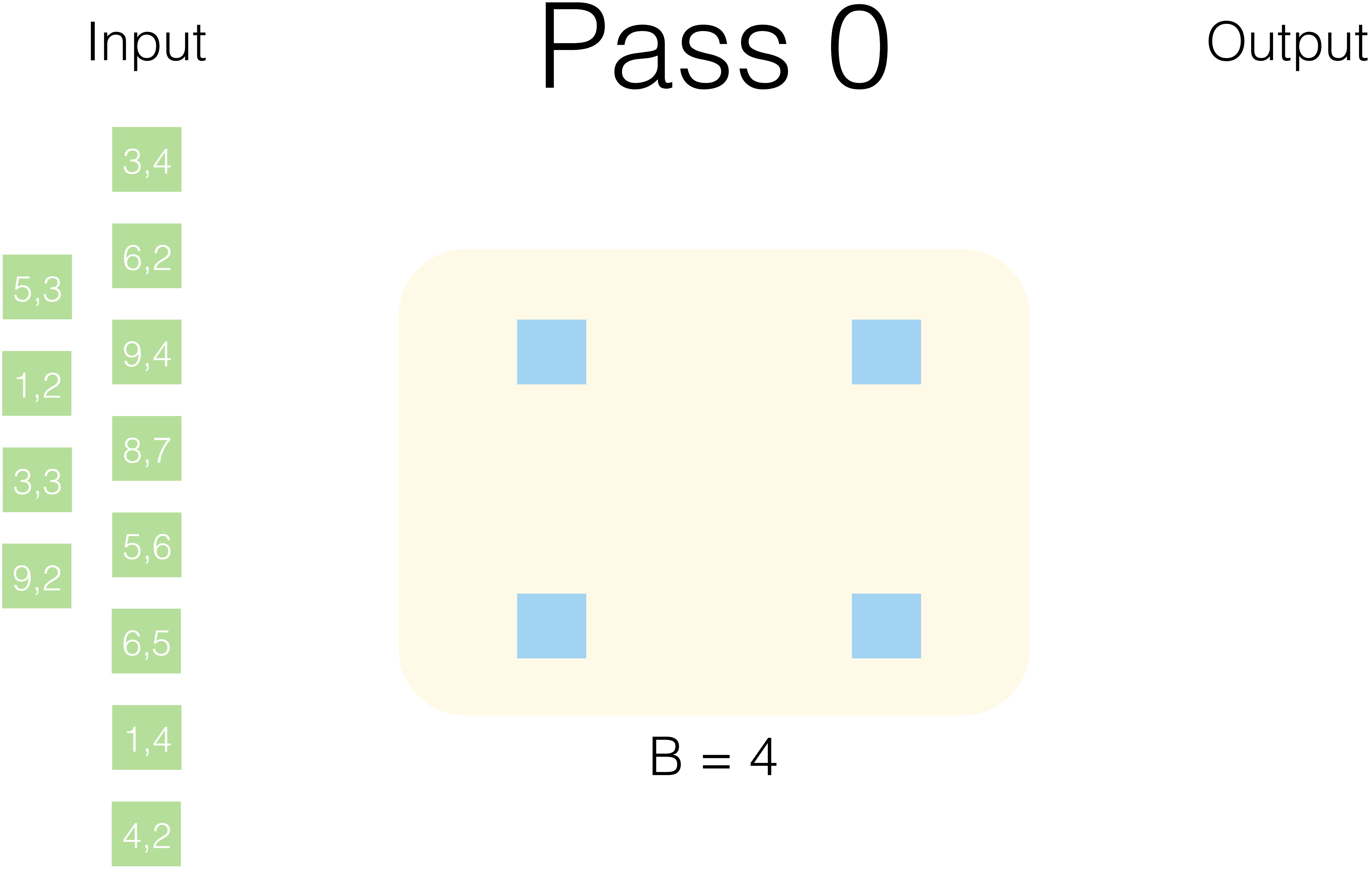
Generalized Merge Sort

Credits to Michelle Nguyen



Buffer size of B pages

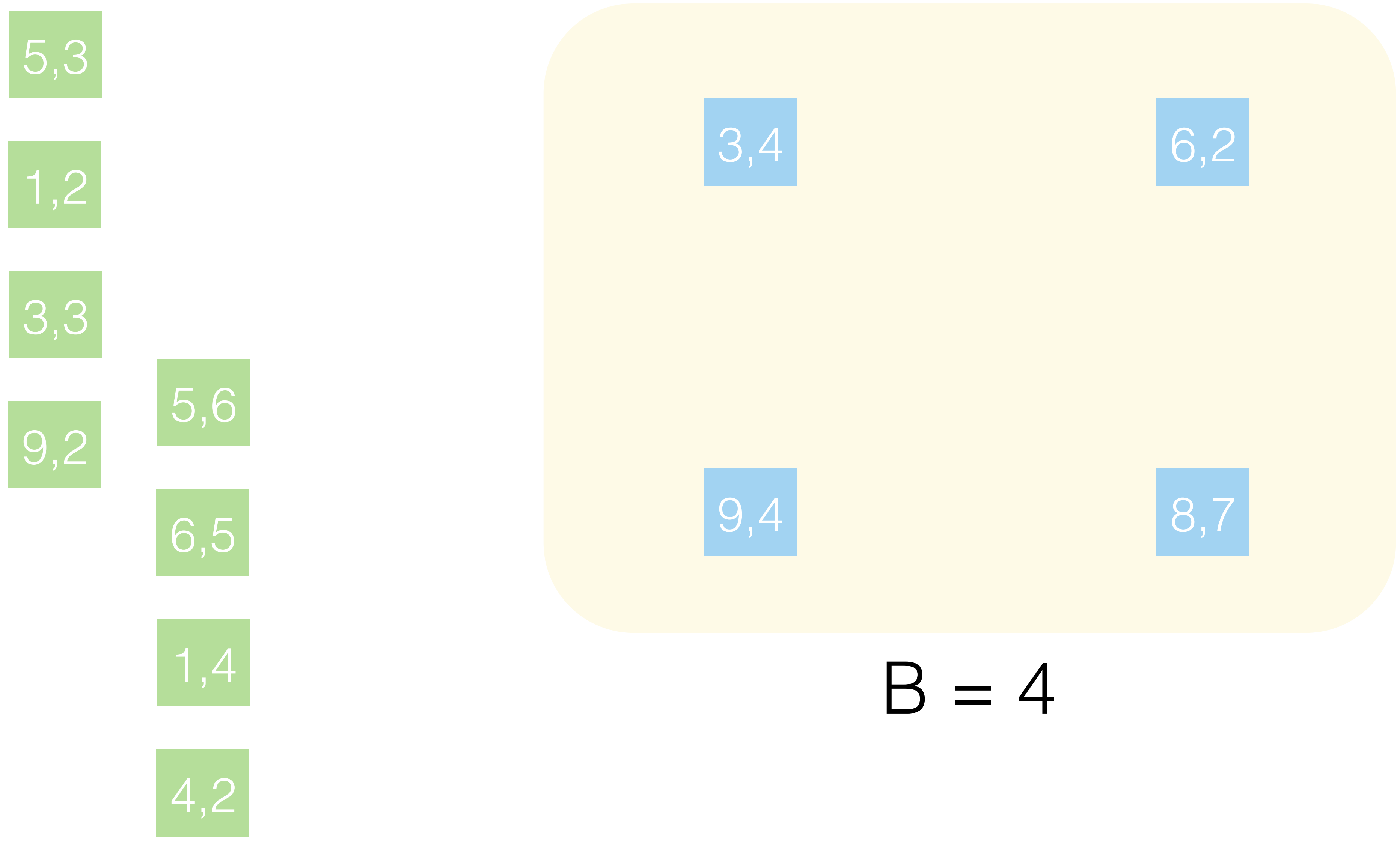
Pass 0



Input

Pass 0

Output



5,3

1,2

3,3

9,2

5,6

6,5

1,4

4,2

3,4

6,2

9,4

8,7

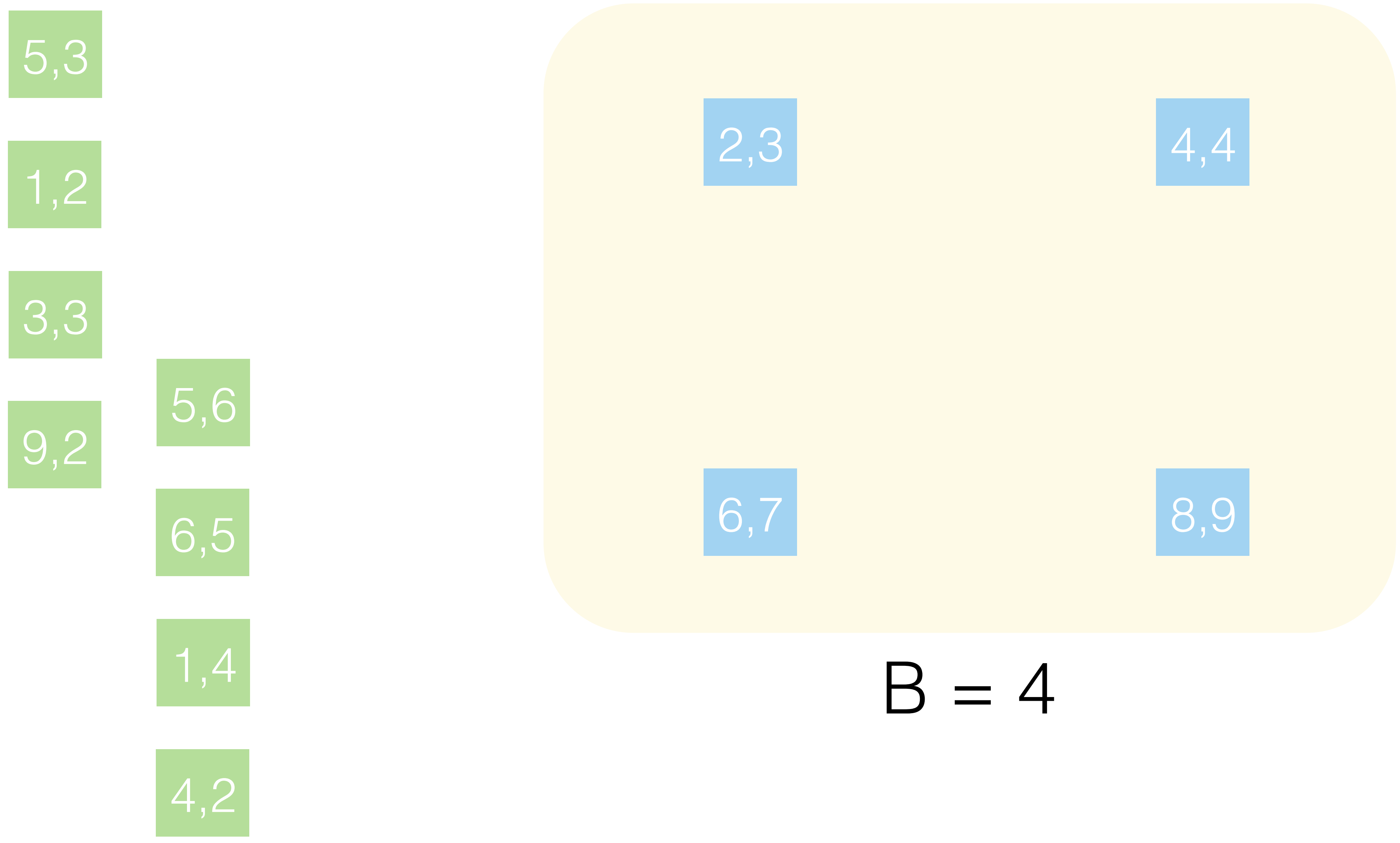
B = 4

Pass 0

Input

Output

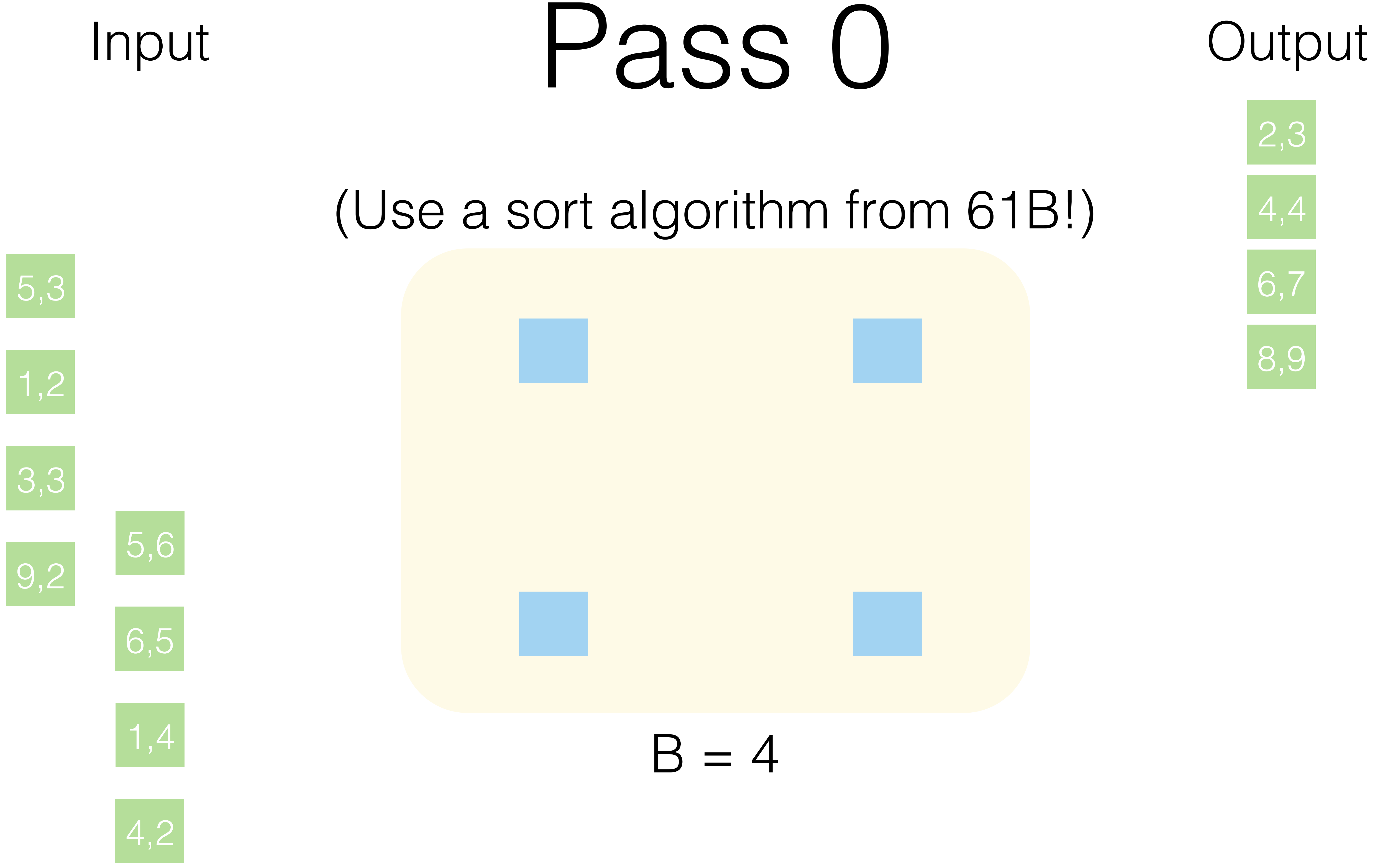
(Use a sort algorithm from 61B!)



B = 4

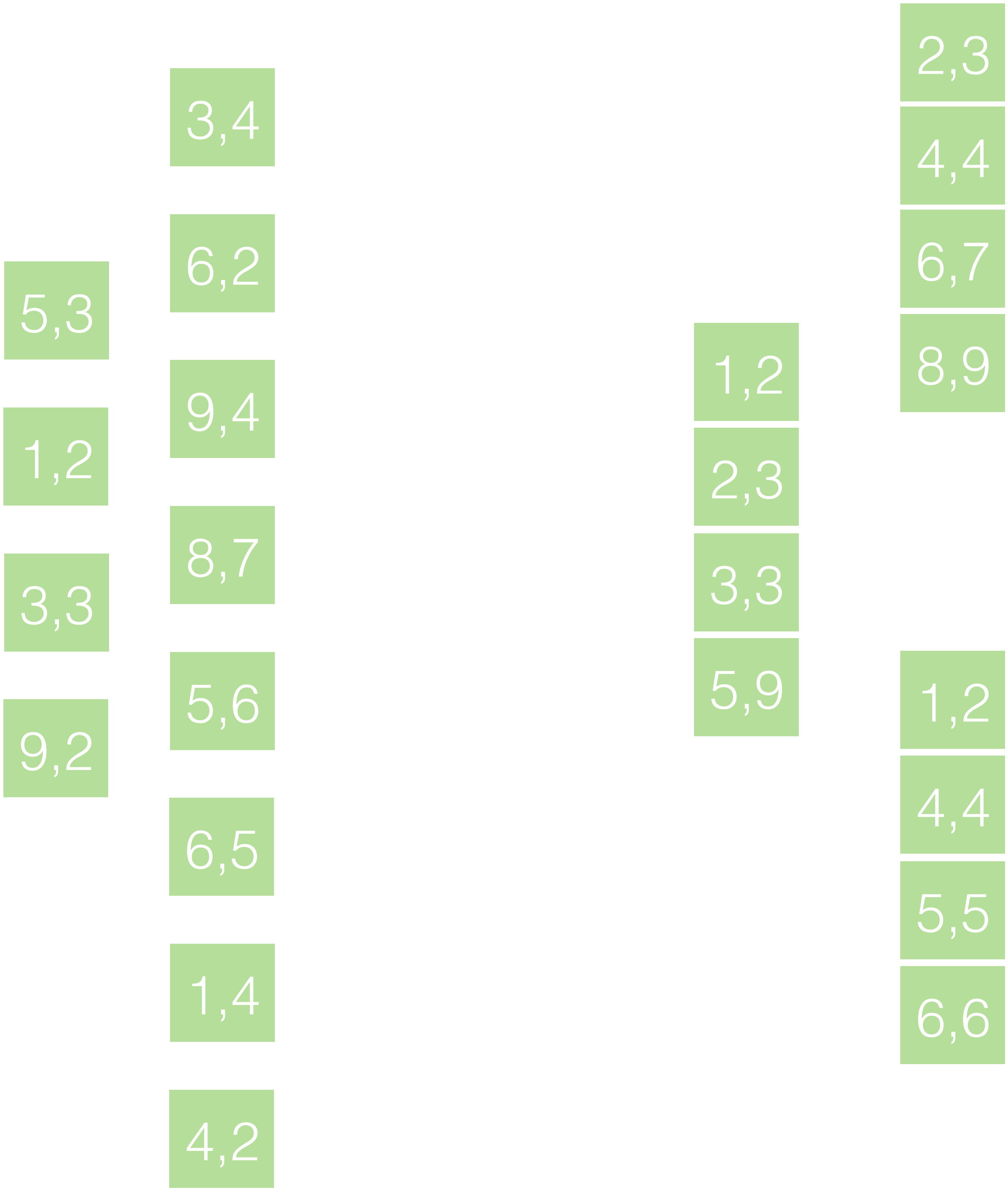
Pass 0

(Use a sort algorithm from 61B!)



Input

Pass 0

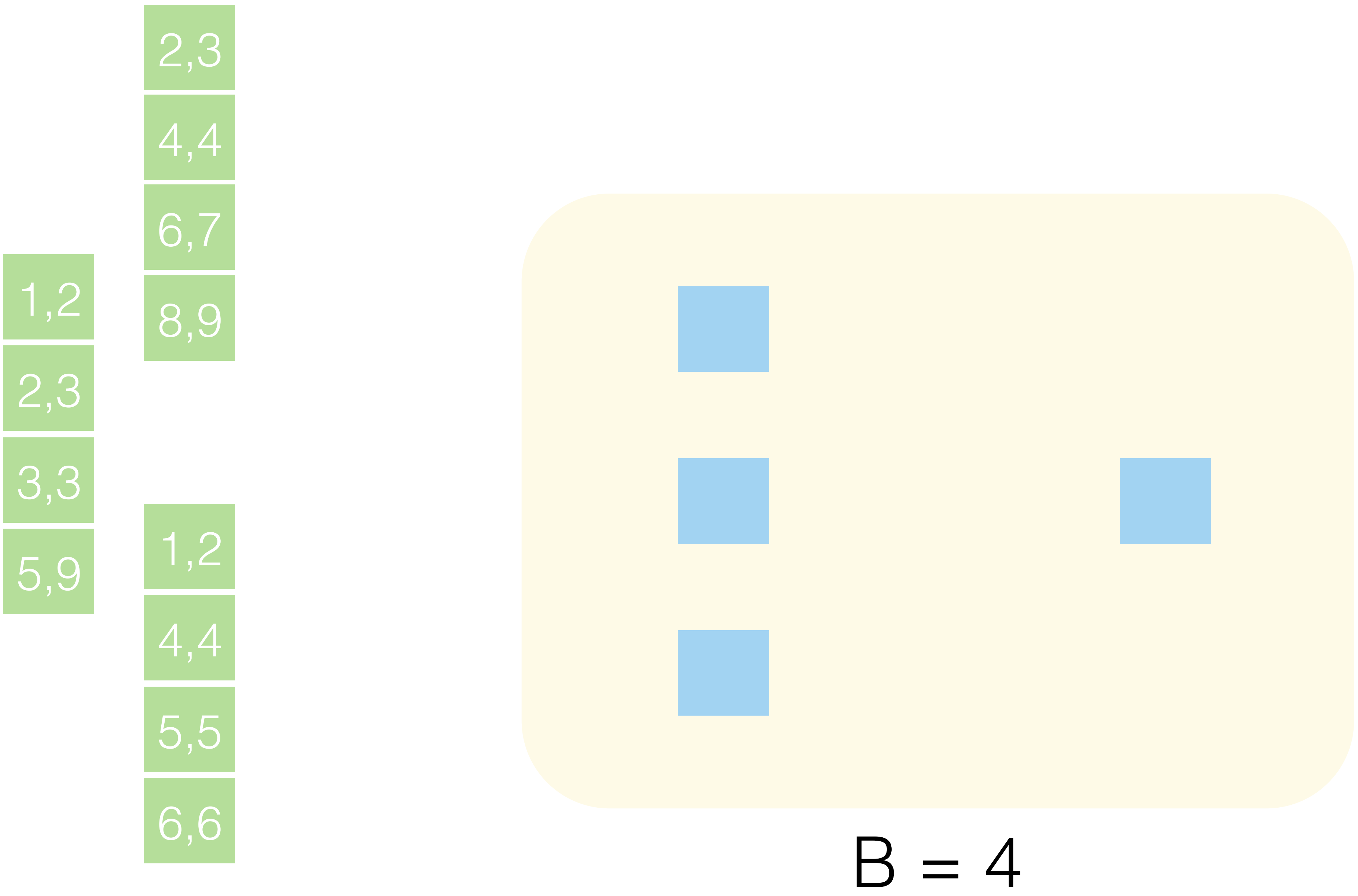


4 page runs

Pass 1

Input

Output



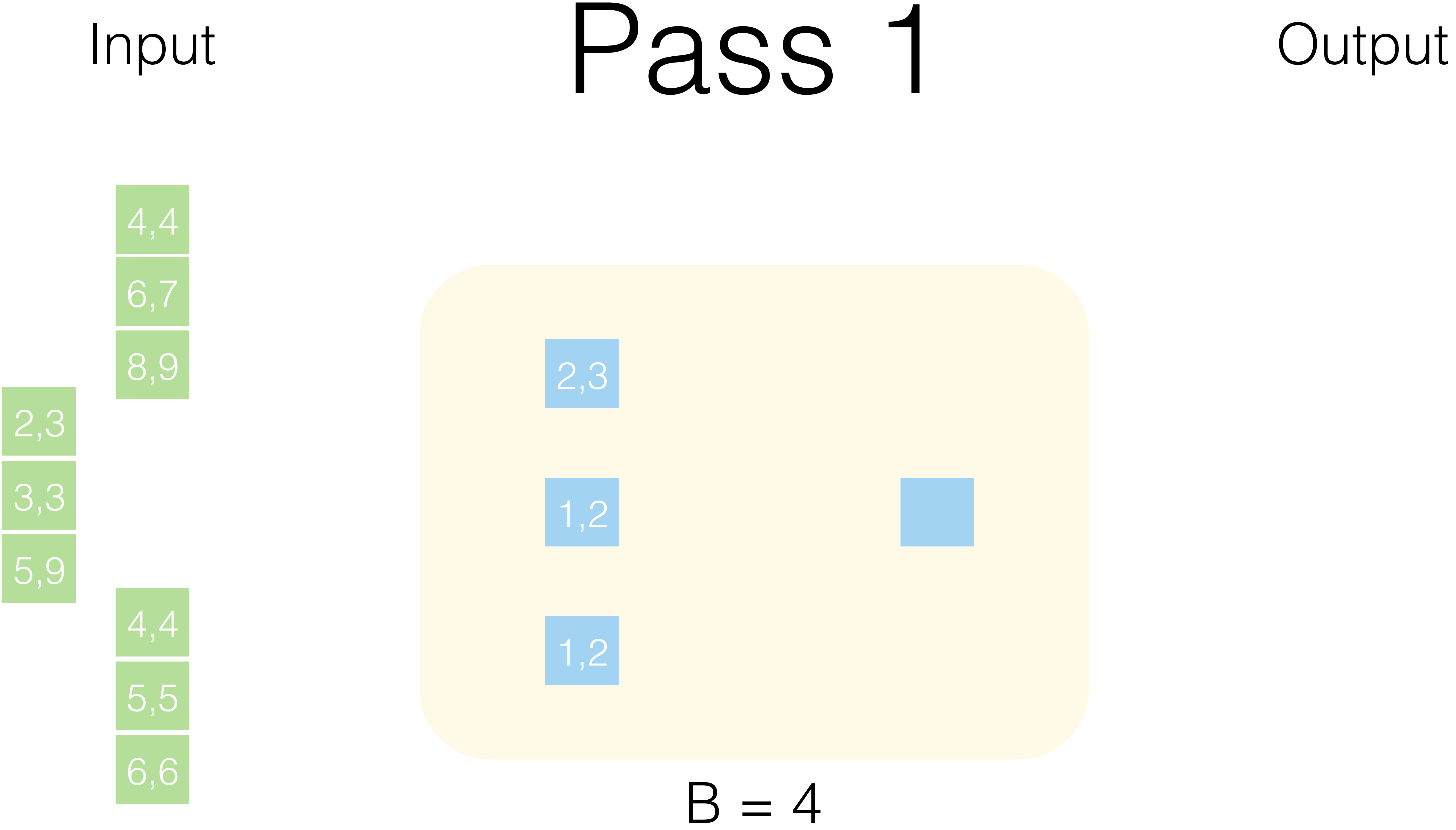
1,2
2,3
3,3
5,9

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

1,2
4,4
5,5
6,6

B = 4

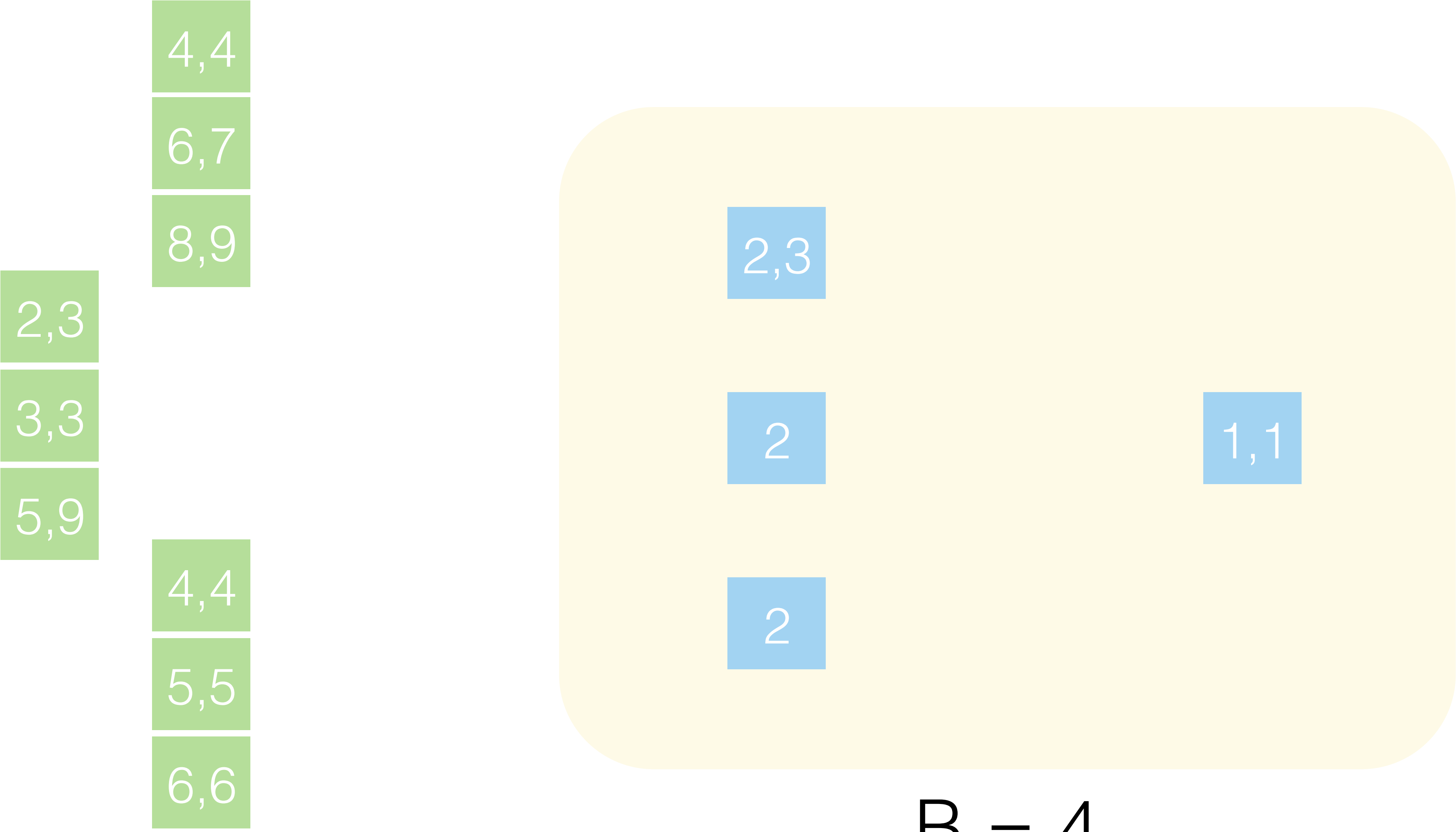
Pass 1



Pass 1

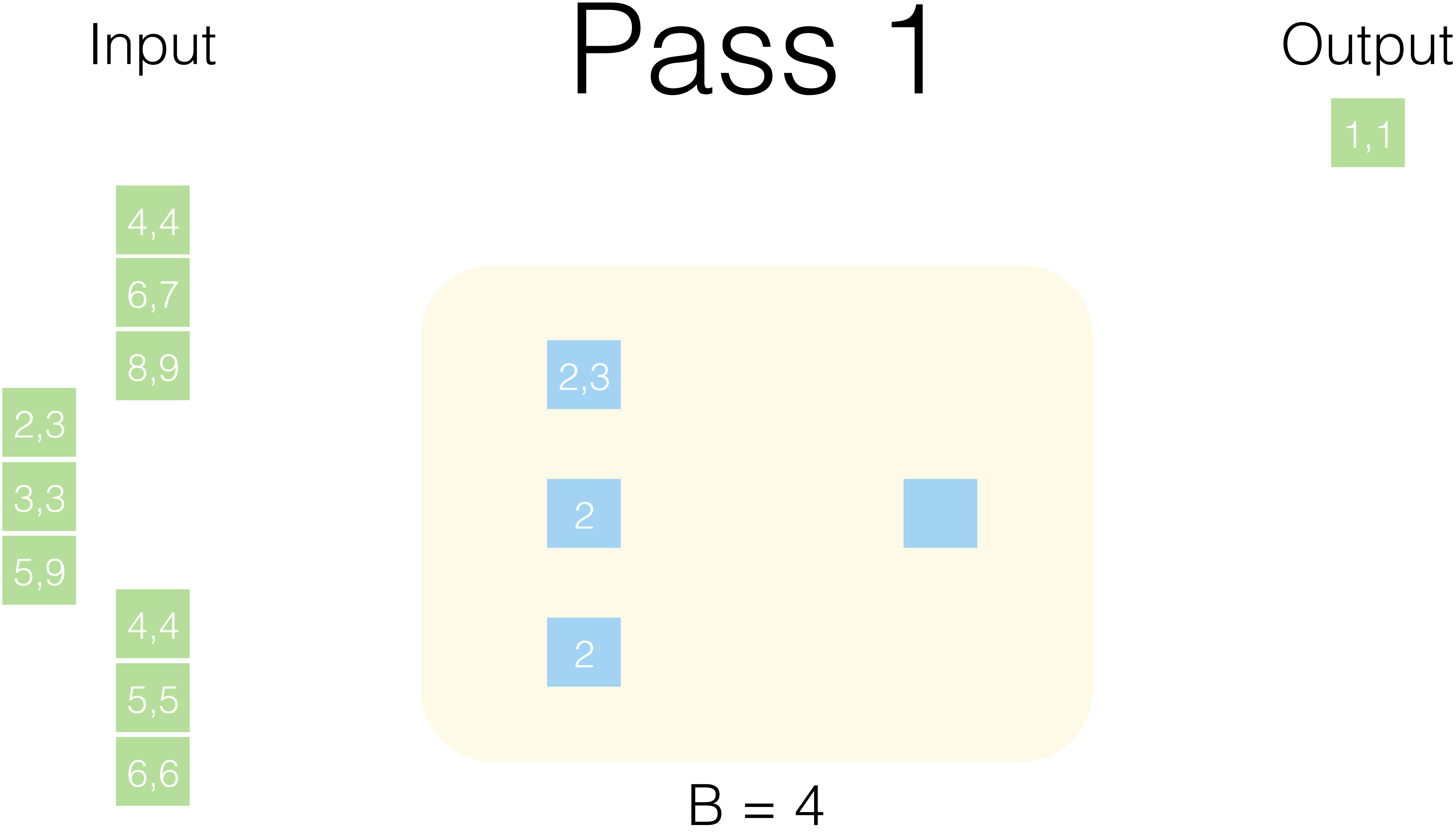
Input

Output



B = 4

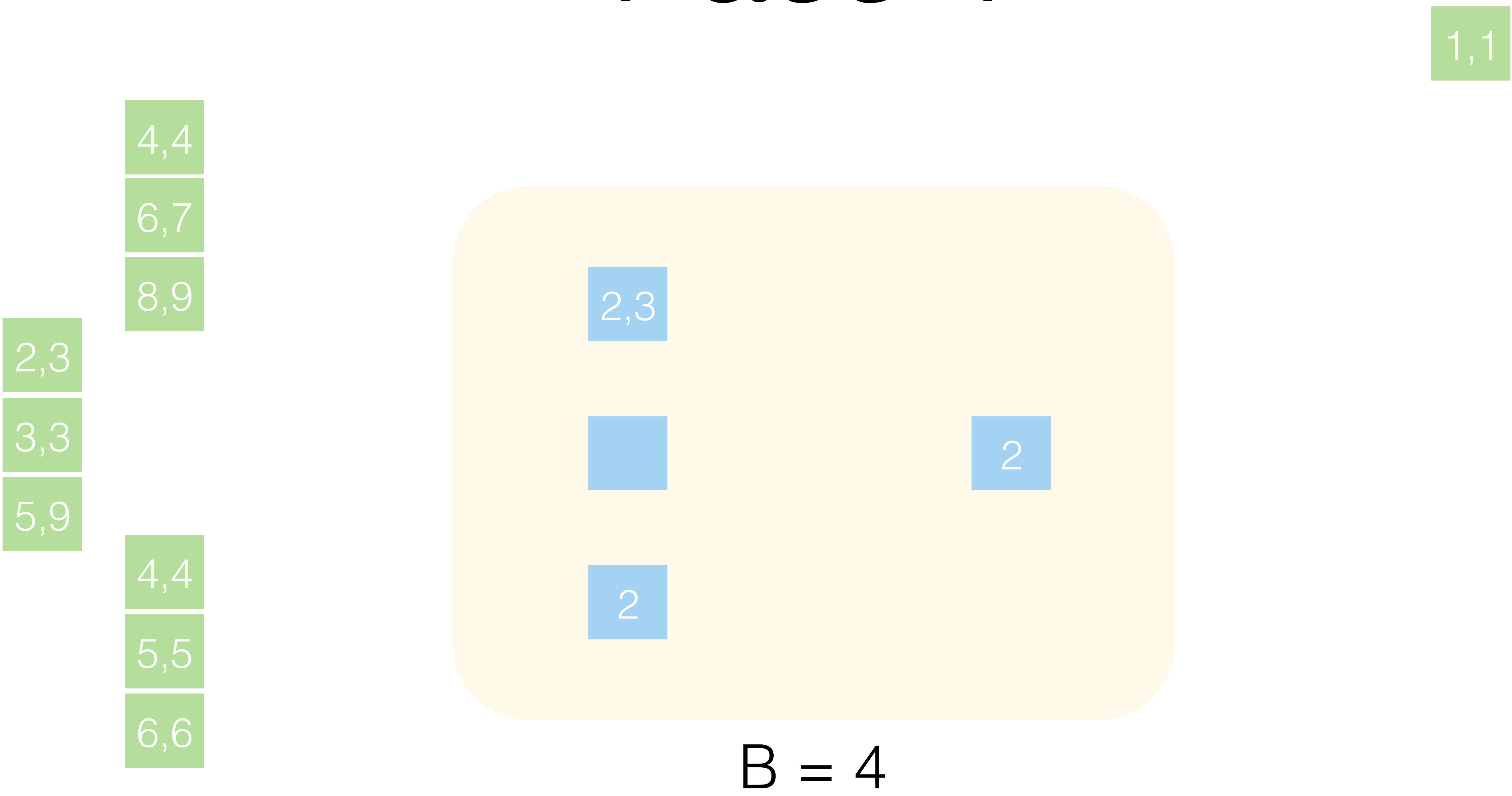
Pass 1



Pass 1

Input

Output

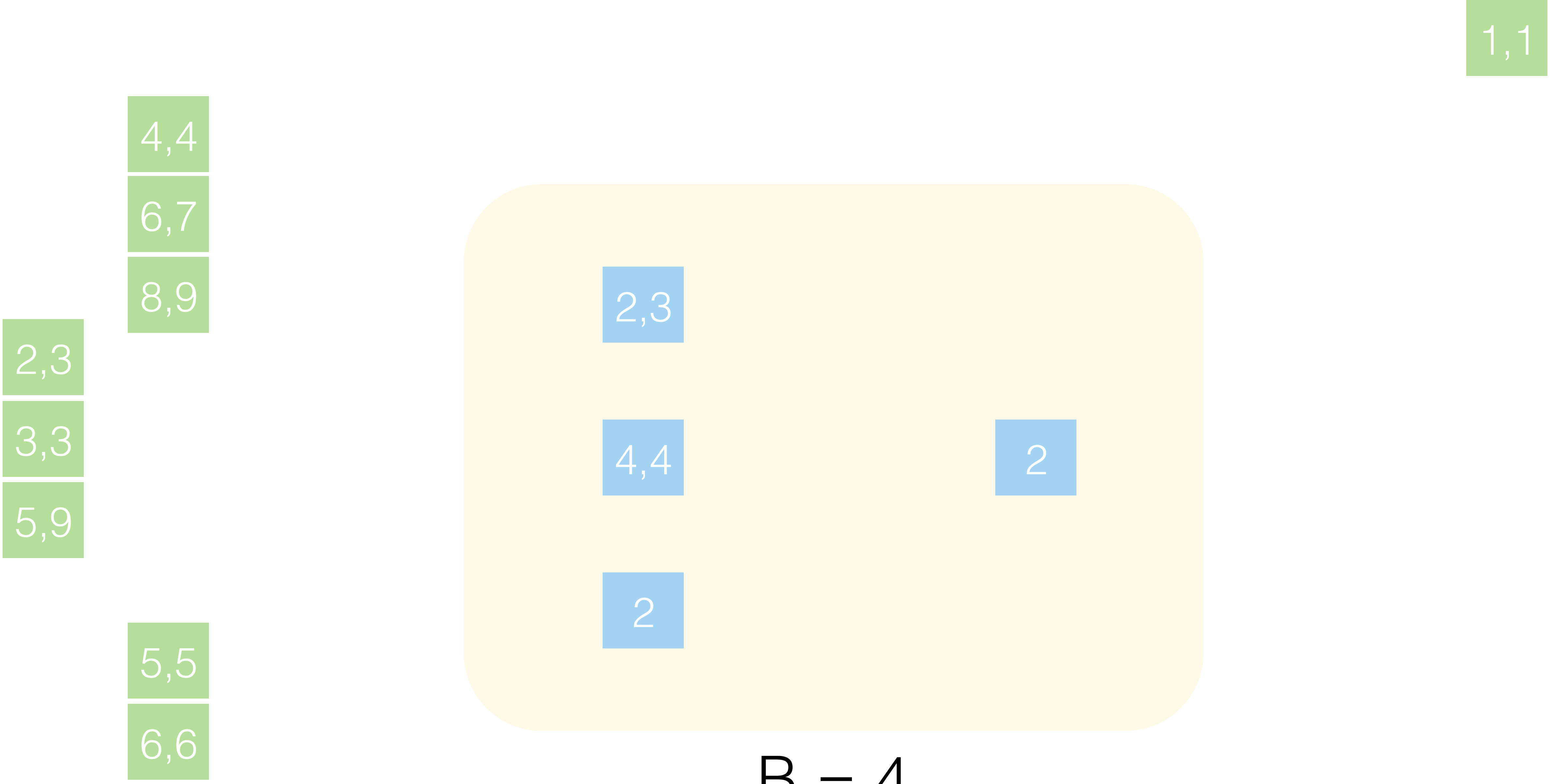


B = 4

Pass 1

Input

Output



1,1

4,4

6,7

8,9

2,3

3,3

5,9

5,5

6,6

2,3

4,4

2

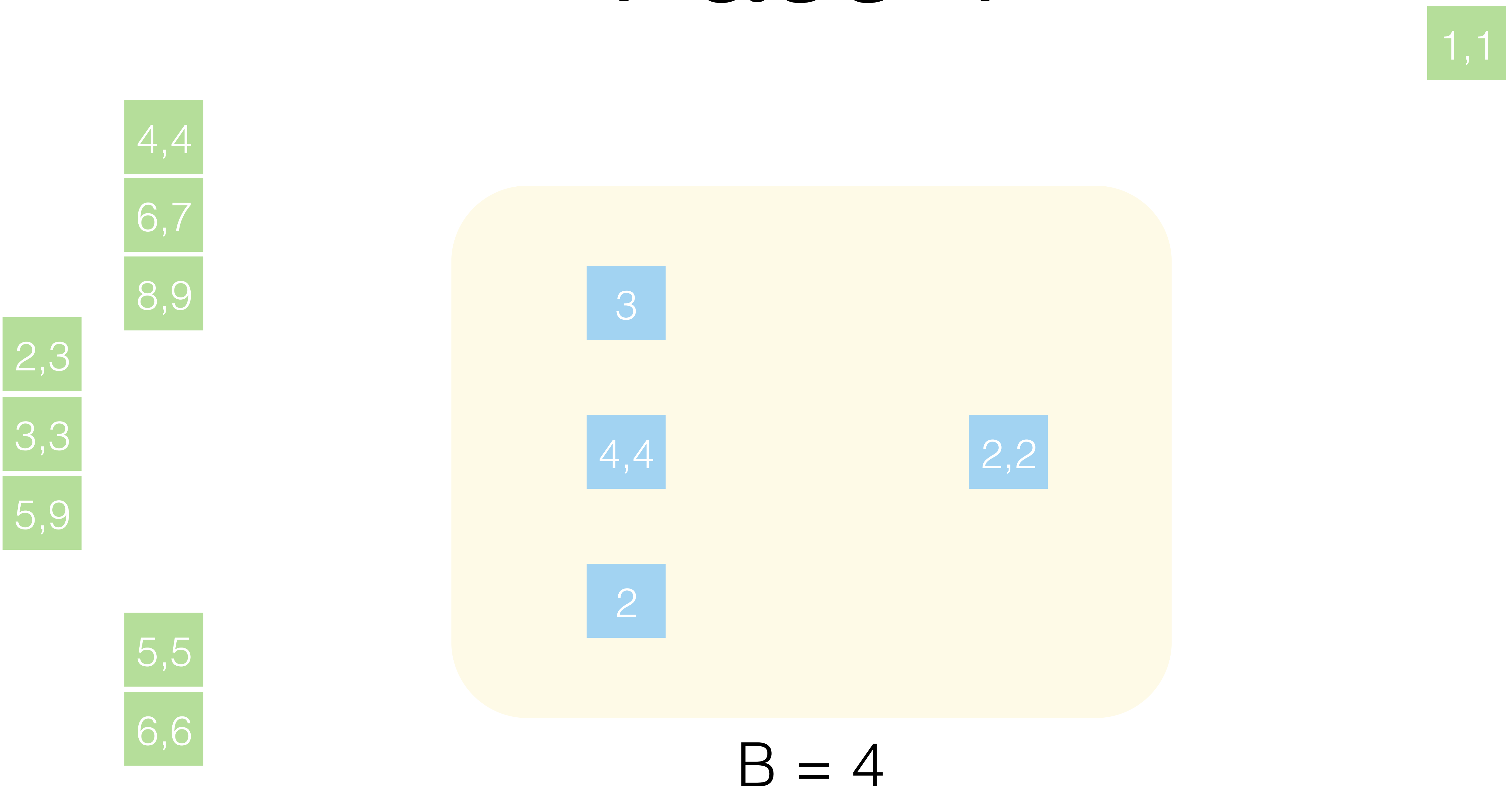
2

B = 4

Pass 1

Input

Output



1,1

4,4

6,7

8,9

2,3

3,3

5,9

5,5

6,6

3

4,4

2

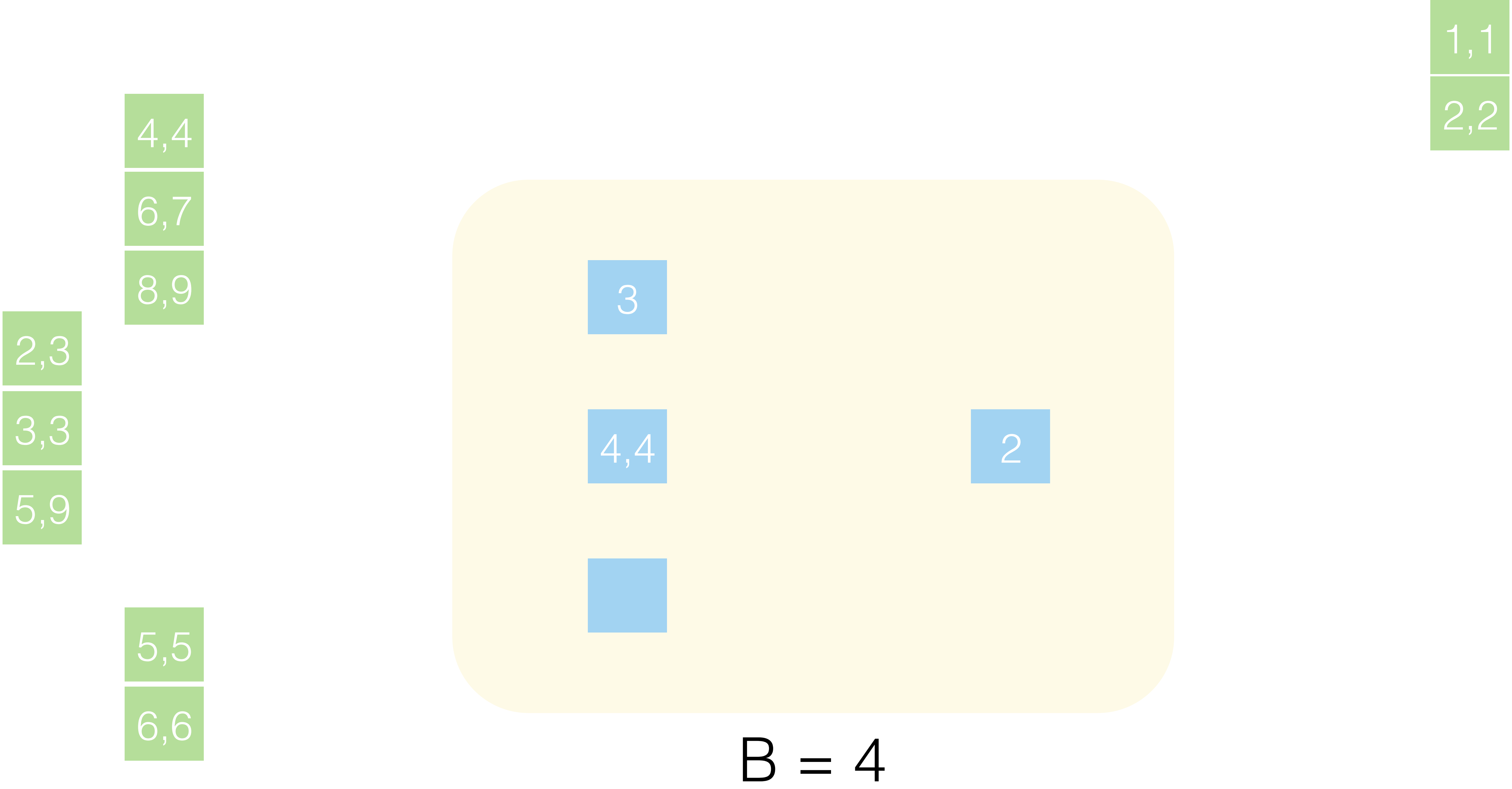
2,2

B = 4

Pass 1

Input

Output

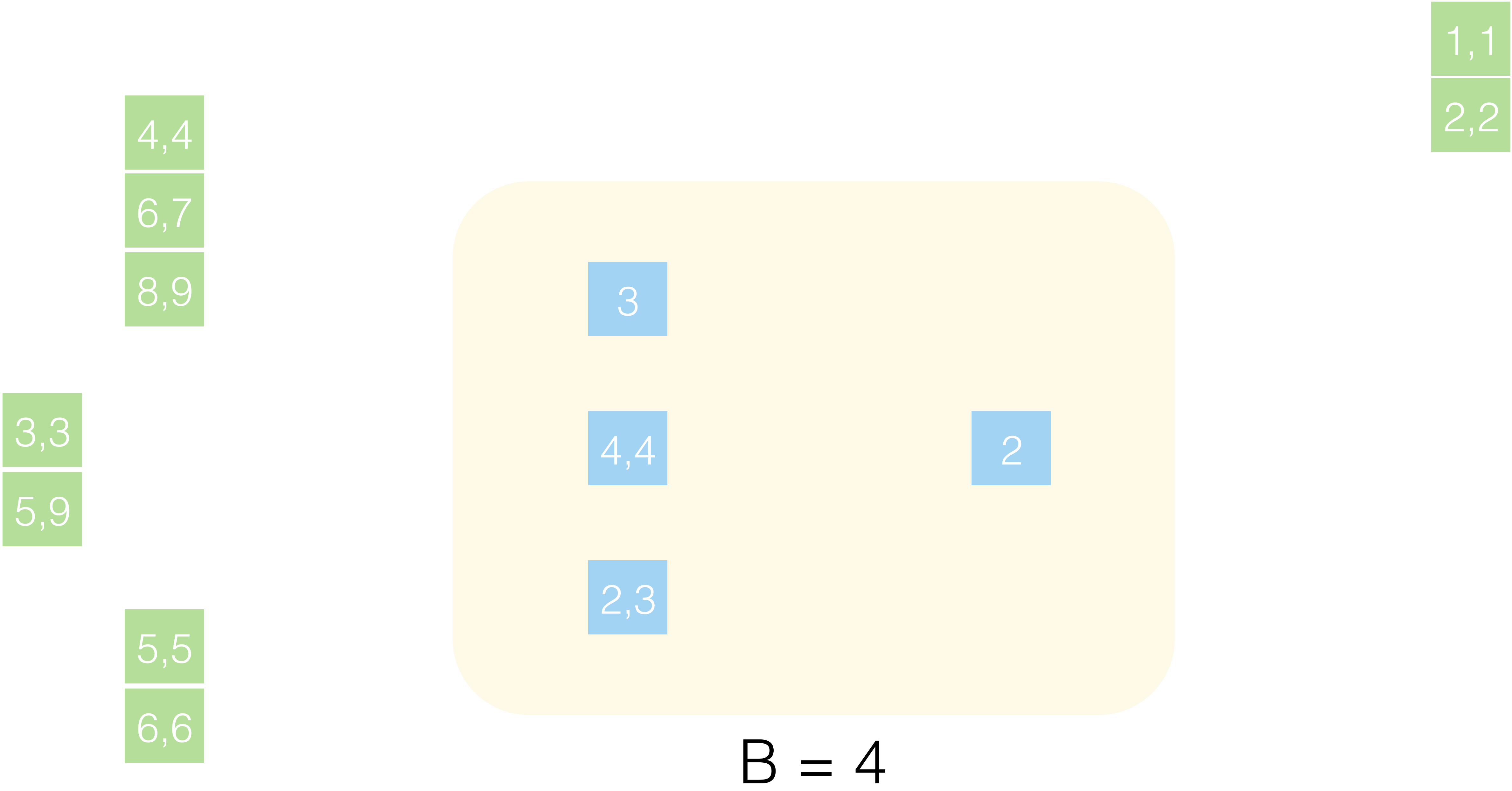


B = 4

Pass 1

Input

Output



1,1

2,2

4,4

6,7

8,9

3,3

5,9

5,5

6,6

3

4,4

2,3

2

B = 4

Generalized Merge Sort

- Number of Passes:
  - $\lceil \log_{B-1} \lceil N/B \rceil \rceil + 1$
- Number of I/O's:
  - $2N * (\lceil \log_{B-1} \lceil N/B \rceil \rceil + 1)$

Worksheet #6, 7

External Sorting Exercises

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

External Sorting Exercises

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

- Sorting: 2-way only utilizes 2 input buffers, general utilizes B-1

- During pass 0, 2-way only uses 1 page to sort files. notice how applying B=3 to the general equation doesn’t yield the correct # of passes. The general external merge sort uses all B pages in its buffer to sort the initial runs in pass 0.

External Sorting Exercises

7. Your system has 640 KB of memory allocated for the buffer for external sorting and you have infinite space for scratch disks. Each page holds 64 KB of data.
Note: 1024 KB = 1 MB.

   1. How many pages can your buffer hold?

External Sorting Exercises

7. Your system has 640 KB of memory allocated for the buffer for external sorting and you have infinite space for scratch disks. Each page holds 64 KB of data.

Note: 1024 KB = 1 MB.

1. How many pages can your buffer hold?

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

External Sorting Exercises

7. Your system has 640 KB of memory allocated for the buffer for external sorting and you have infinite space for scratch disks. Each page holds 64 KB of data.
Note: 1024 KB = 1 MB.

2. How many pages are in a 4 MB file?

External Sorting Exercises

7. Your system has 640 KB of memory allocated for the buffer for external sorting and you have infinite space for scratch disks. Each page holds 64 KB of data.
Note: 1024 KB = 1 MB.

2. How many pages are in a 4 MB file?

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

External Sorting Exercises

7. Your system has 640 KB of memory allocated for the buffer for external sorting and you have infinite space for scratch disks. Each page holds 64 KB of data.
Note: 1024 KB = 1 MB.

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

External Sorting Exercises

7. Your system has 640 KB of memory allocated for the buffer for external sorting and you have infinite space for scratch disks. Each page holds 64 KB of data.
Note: 1024 KB = 1 MB.

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

$$\lceil \log_{B-1} \lceil N/B \rceil \rceil + 1$$
$$= \lceil \log_{10-1} \lceil 64/10 \rceil \rceil + 1$$
$$= \lceil \log_{9} 7 \rceil + 1$$
$$= 1 + 1$$

= 2 passes

External Sorting Exercises

7. Your system has 640 KB of memory allocated for the buffer for external sorting and you have infinite space for scratch disks. Each page holds 64 KB of data.
Note: 1024 KB = 1 MB.

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

External Sorting Exercises

7. Your system has 640 KB of memory allocated for the buffer for external sorting and you have infinite space for scratch disks. Each page holds 64 KB of data.
Note: 1024 KB = 1 MB.

4. 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

External Sorting Exercises

7. Your system has 640 KB of memory allocated for the buffer for external sorting and you have infinite space for scratch disks. Each page holds 64 KB of data.
Note: 1024 KB = 1 MB.

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

External Sorting Exercises

7. Your system has 640 KB of memory allocated for the buffer for external sorting and you have infinite space for scratch disks. Each page holds 64 KB of data.
Note: 1024 KB = 1 MB.

5. 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

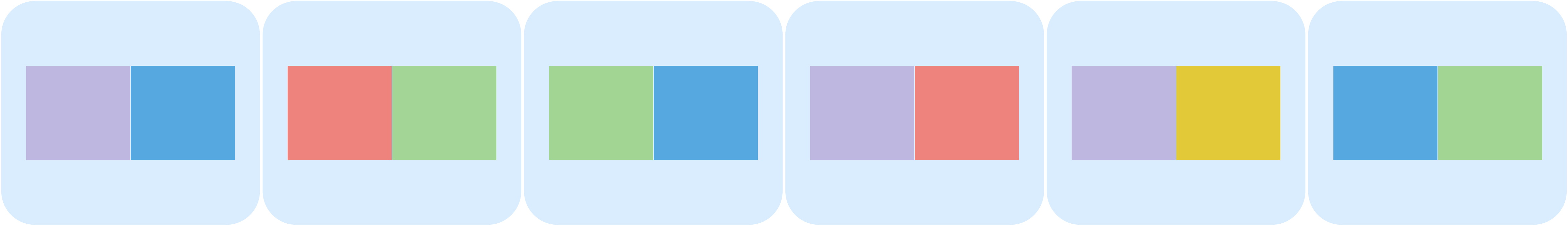
External Hashing

Want to aggregate data that does not fit in memory

Aggregating Colors

Credits to Michelle Nguyen

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



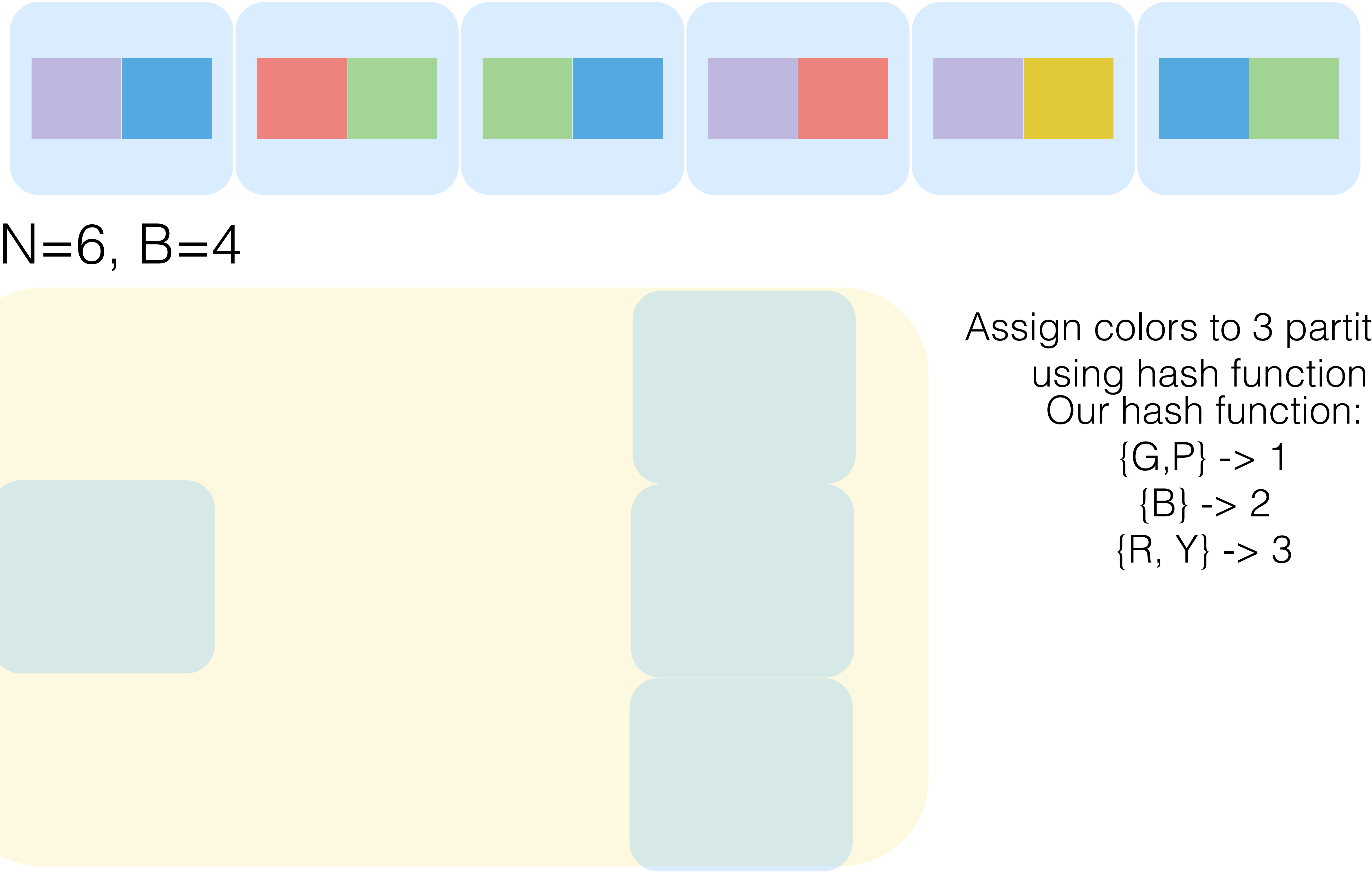
Pass 1: Divide

- 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.

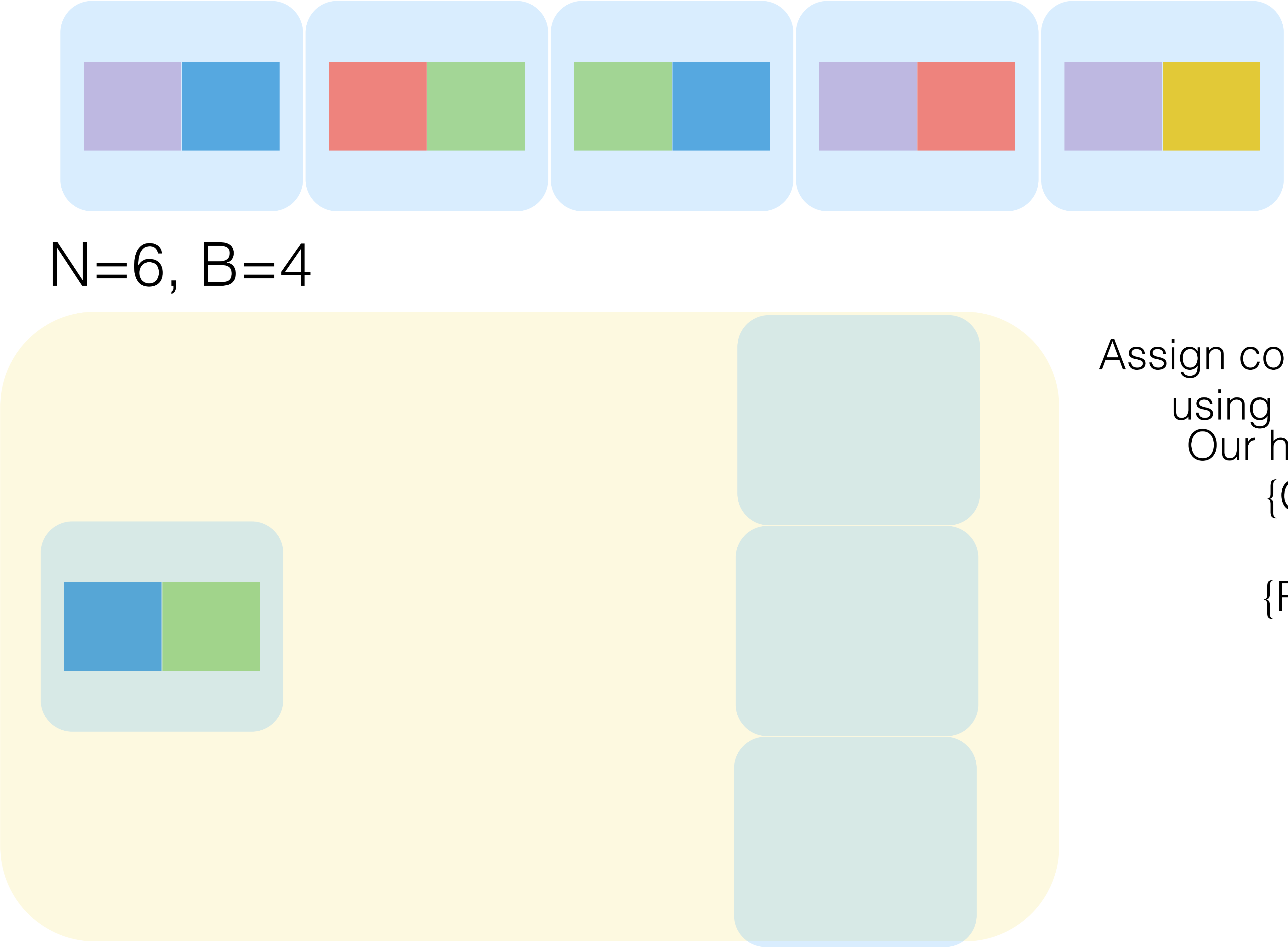
Our hash function:

{G,P} -> 1

{B} -> 2

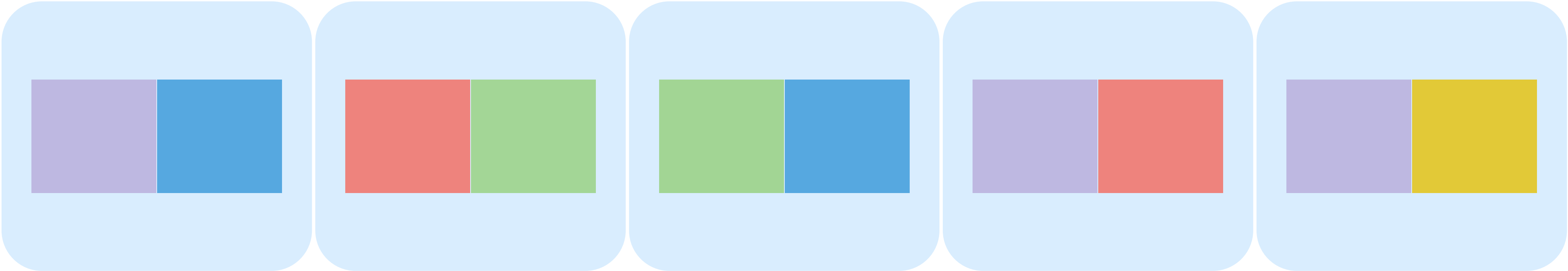
{R, Y} -> 3

Pass 1: Divide

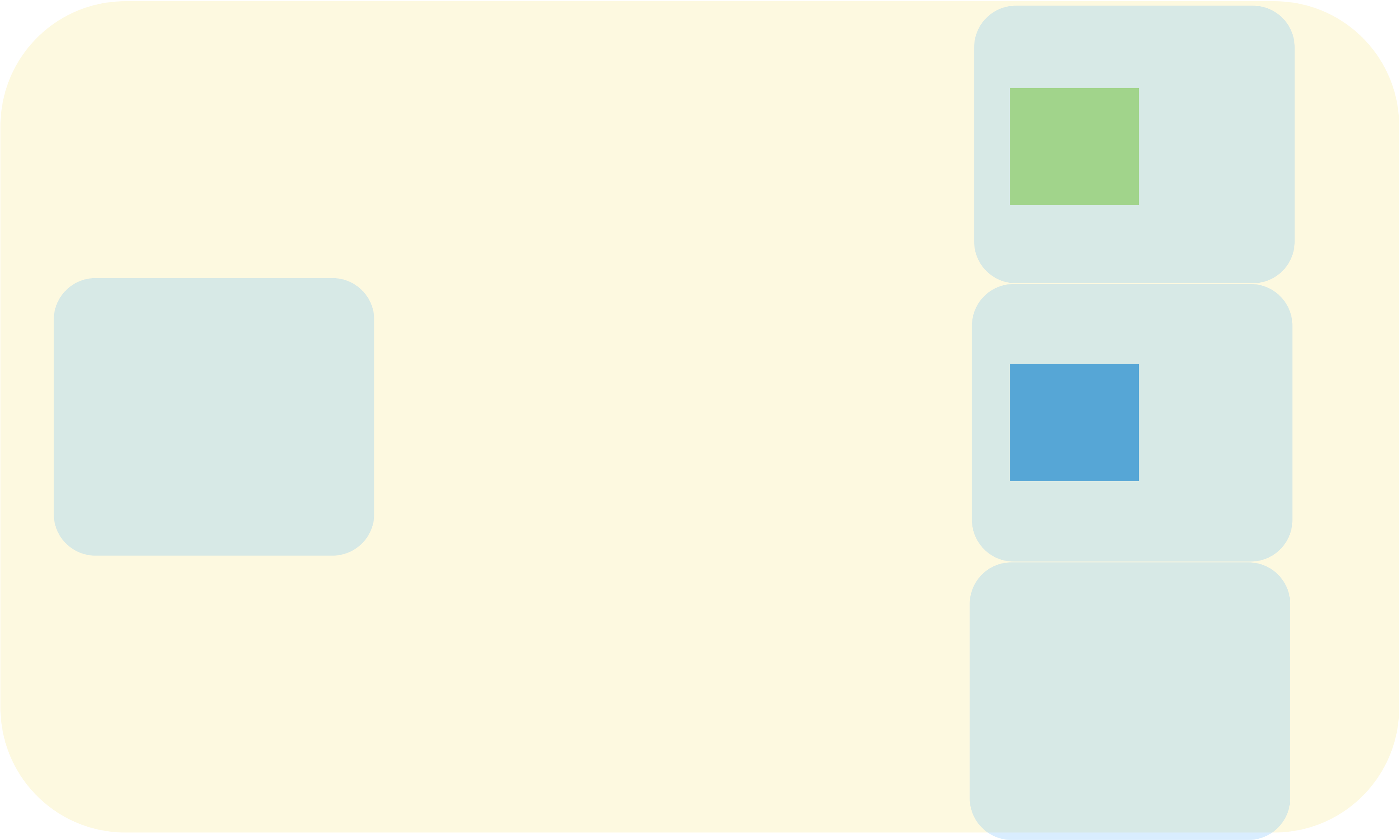


Assign colors to 3 partitions using hash function.
Our hash function:
{G,P} -> 1
{B} -> 2
{R, Y} -> 3

Pass 1: Divide

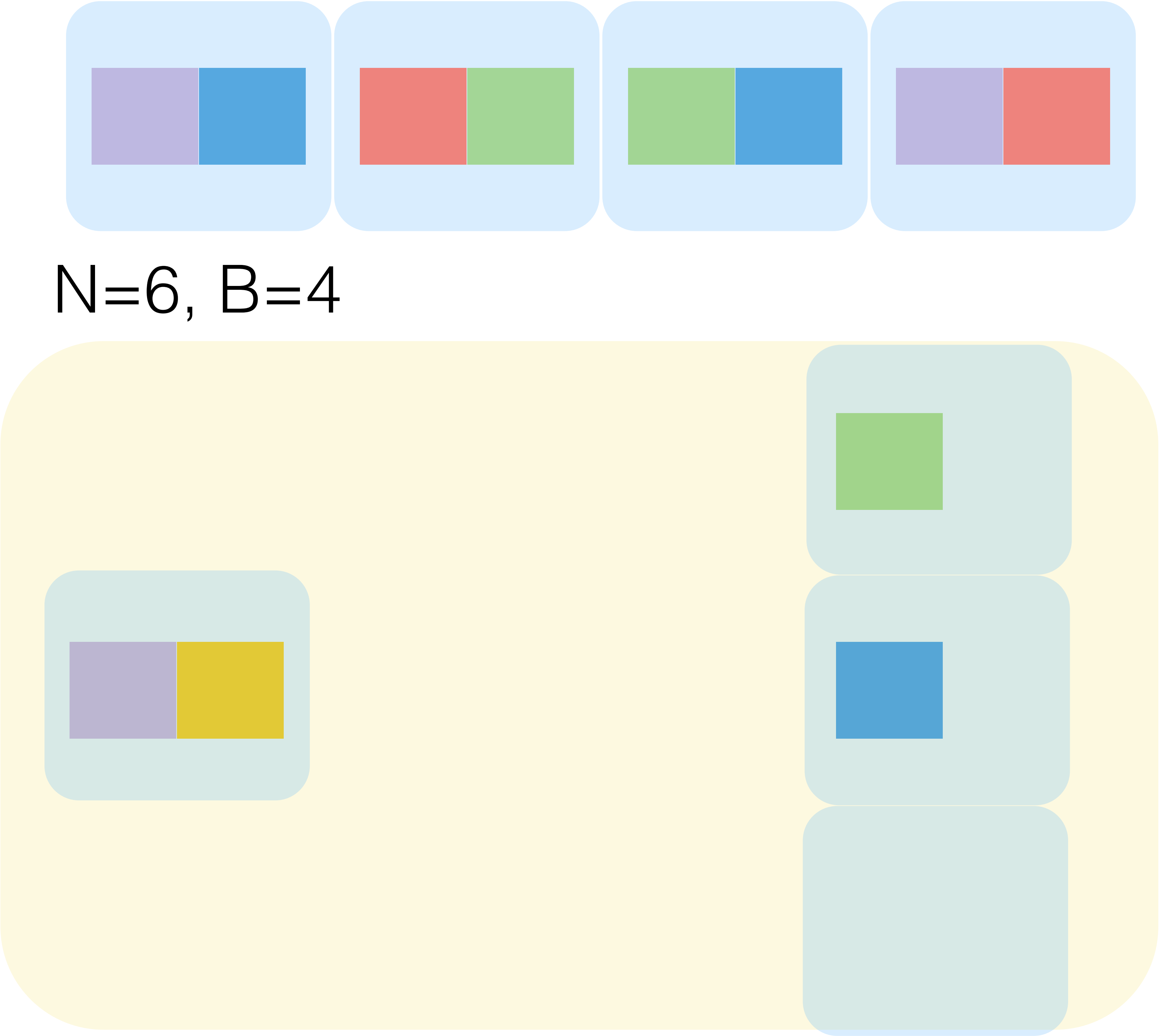


N=6, B=4



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

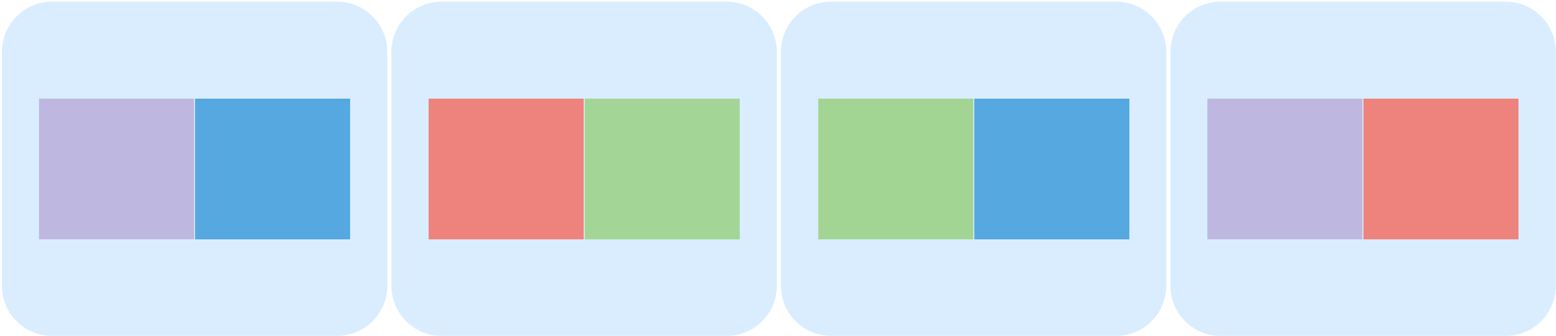
Pass 1: Divide



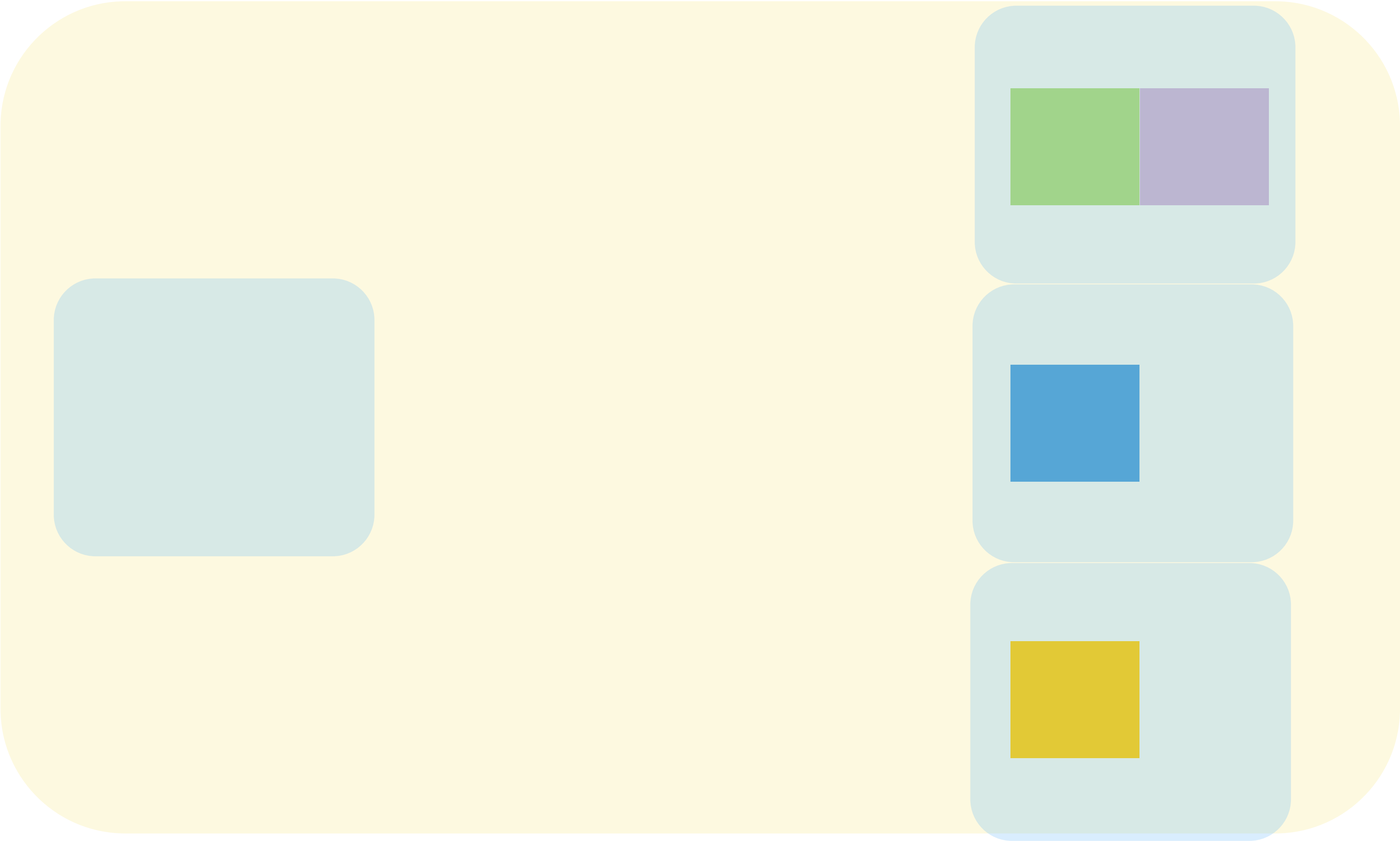
N=6, B=4

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

Pass 1: Divide

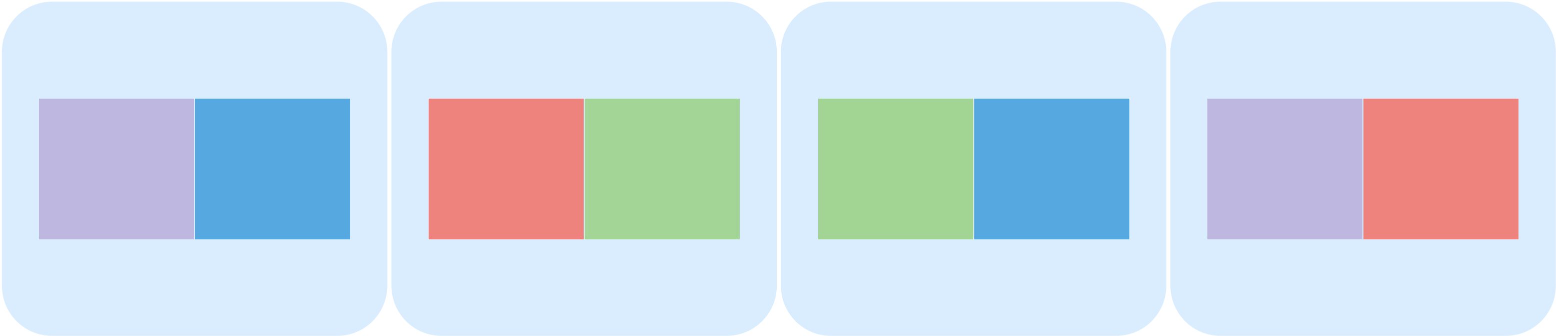


N=6, B=4



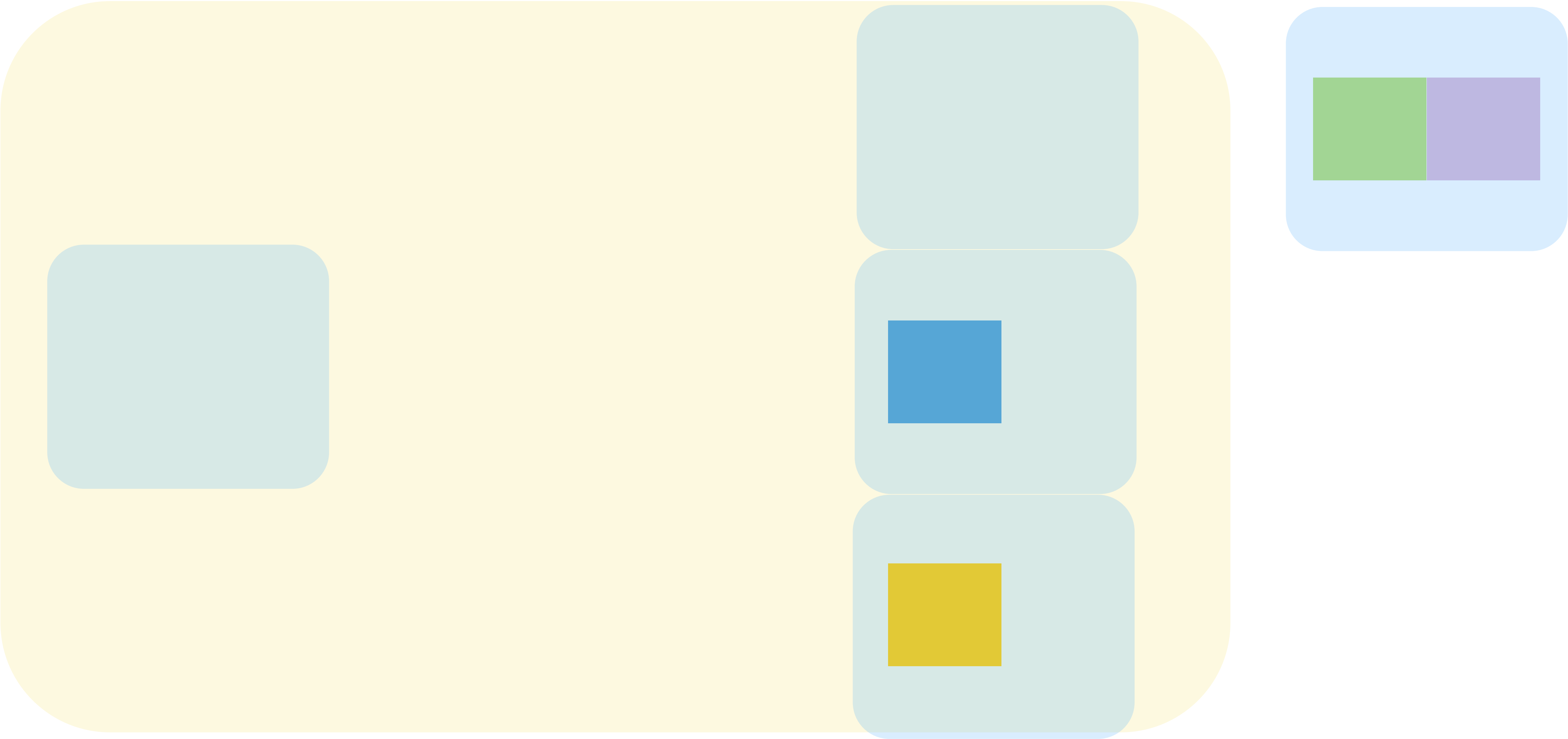
Assign colors to 3 partitions using hash function.
Our hash function:
{G,P} -> 1
{B} -> 2
{R, Y} -> 3

Pass 1: Divide

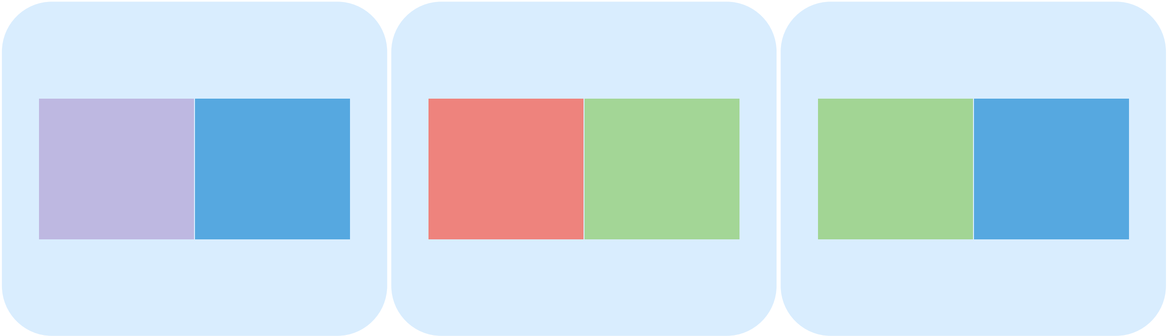


N=6, B=4

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

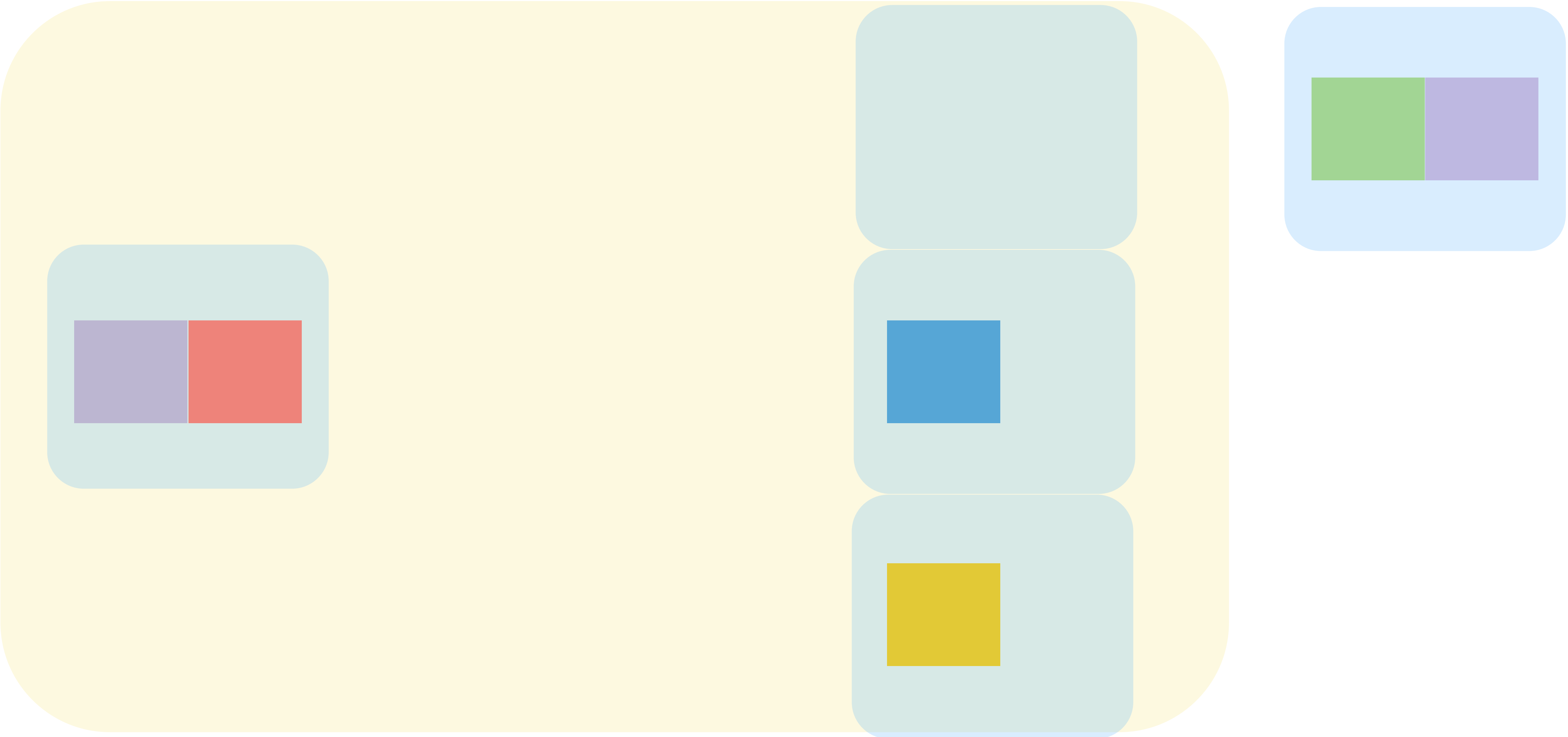


Pass 1: Divide

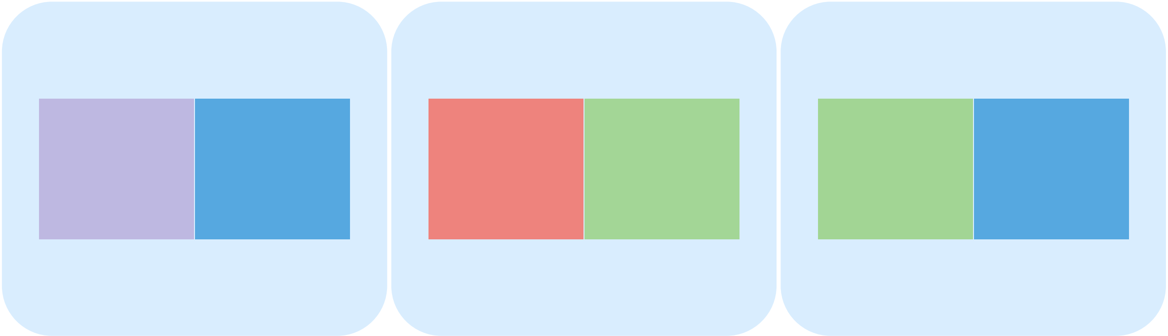


N=6, B=4

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

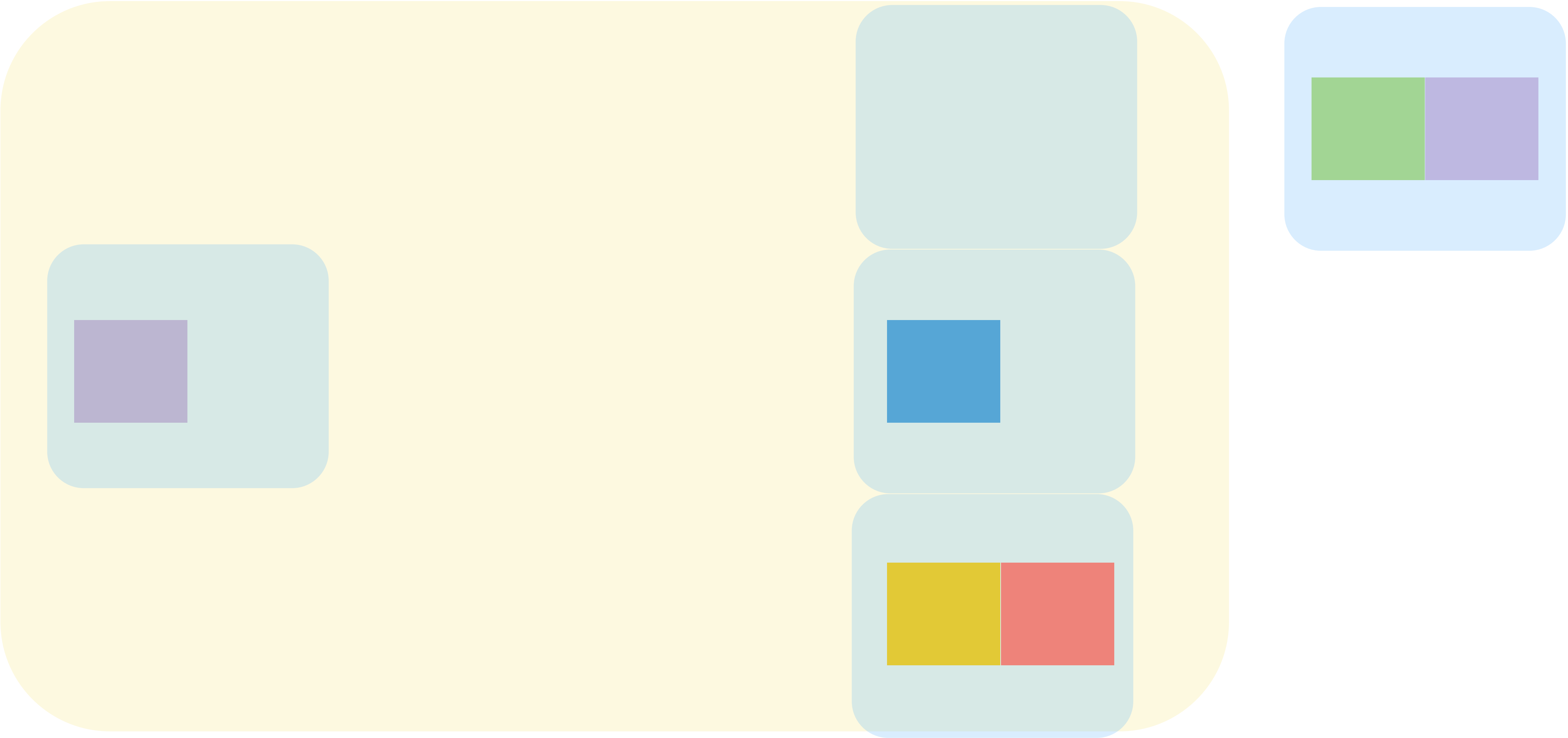


Pass 1: Divide

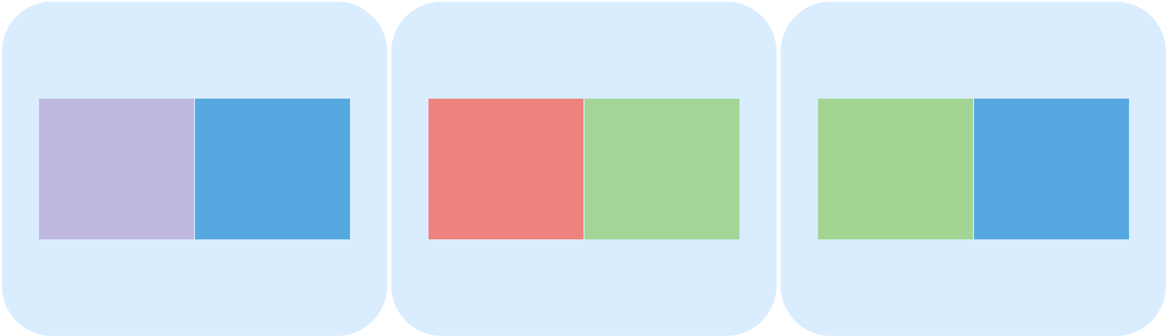


N=6, B=4

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

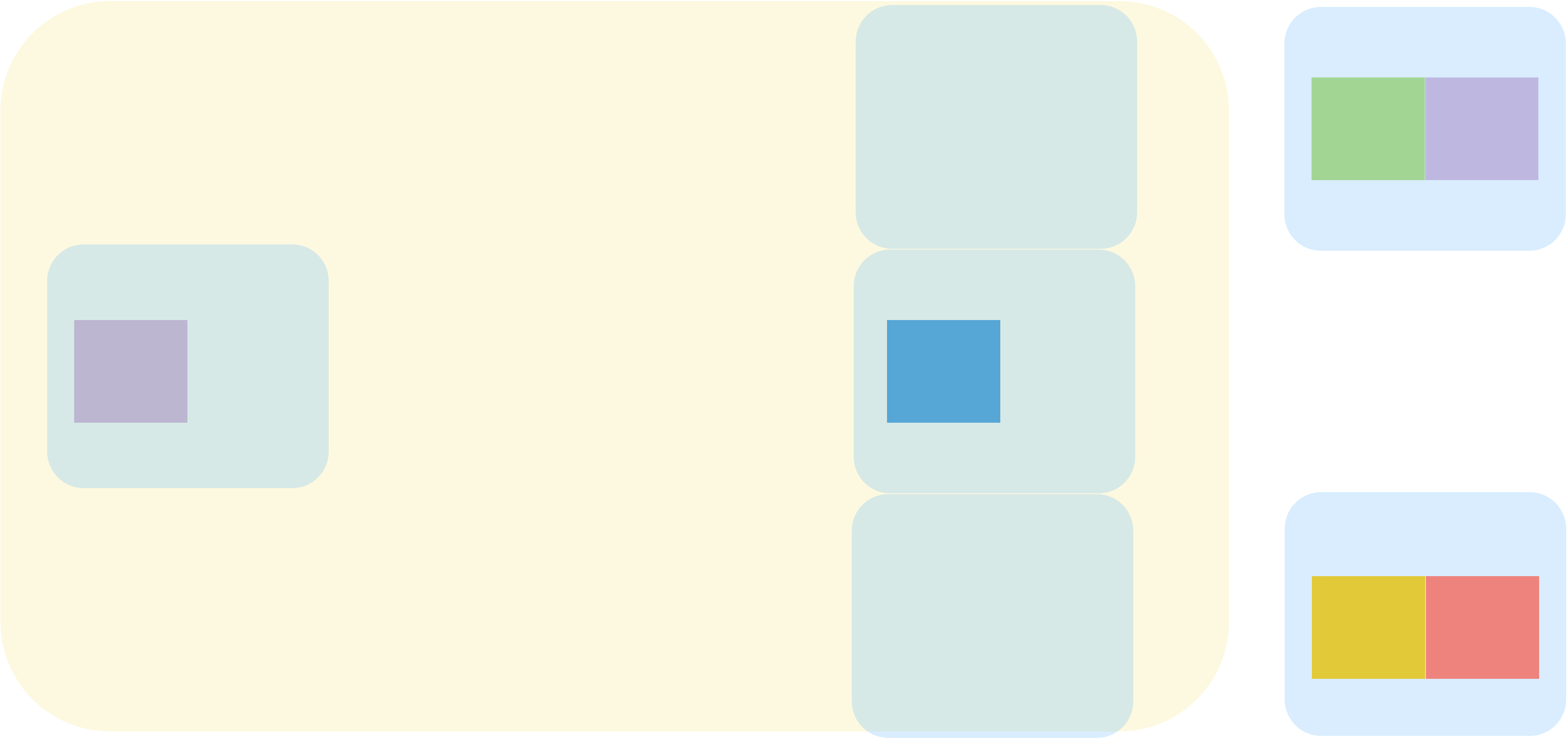


Pass 1: Divide

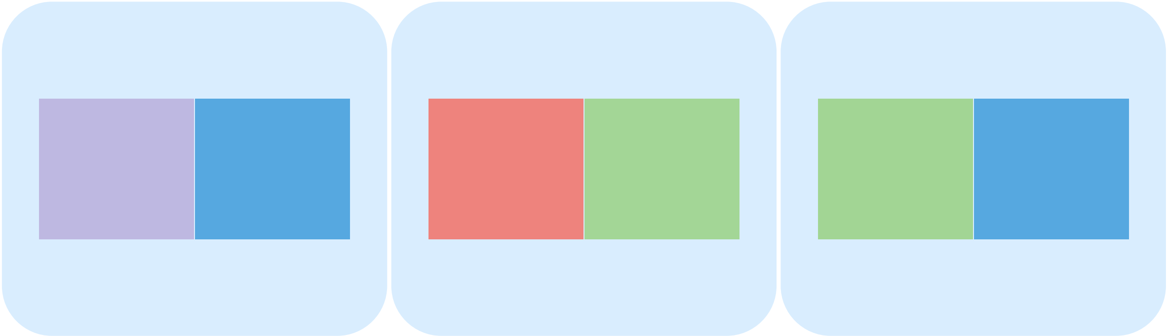


N=6, B=4

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

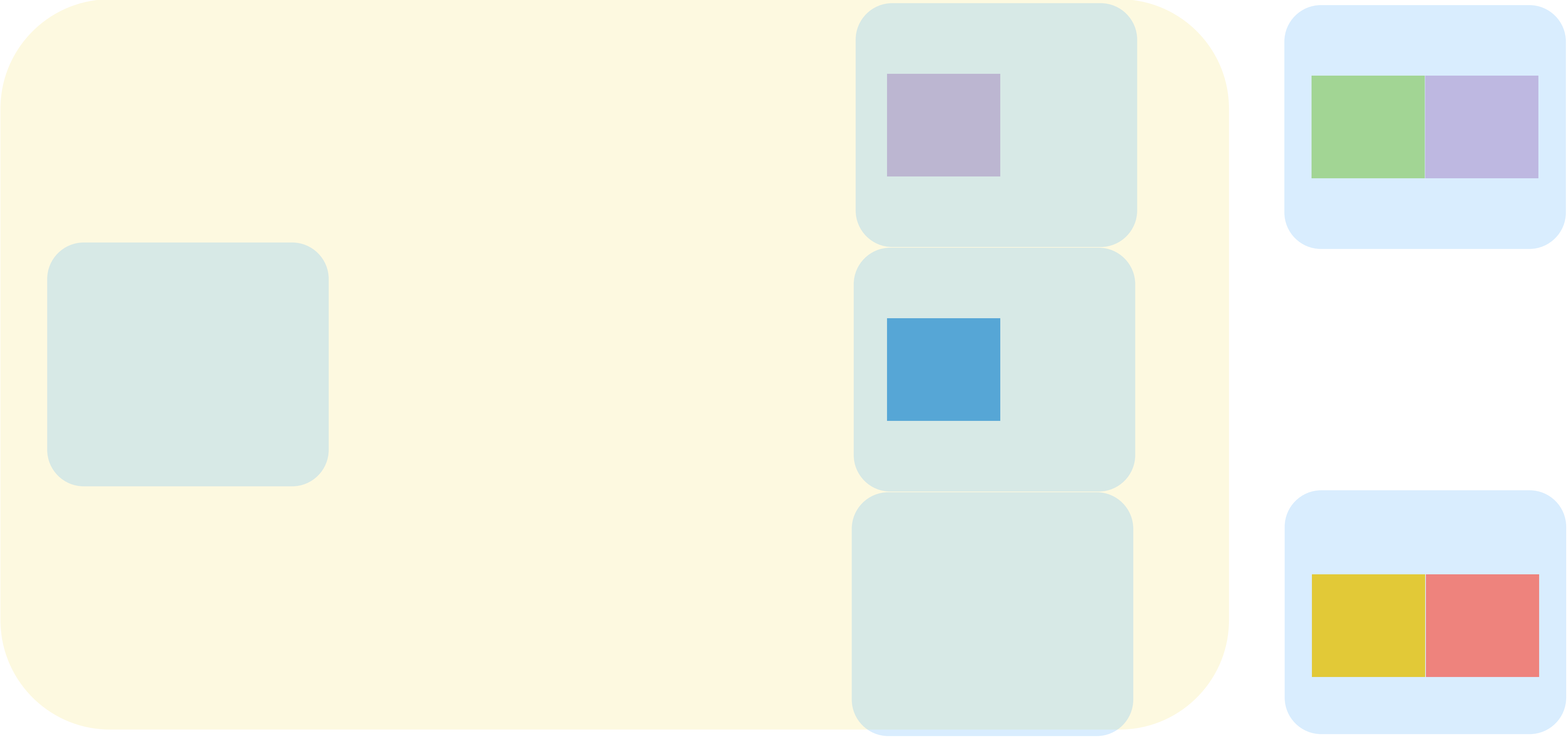


Pass 1: Divide

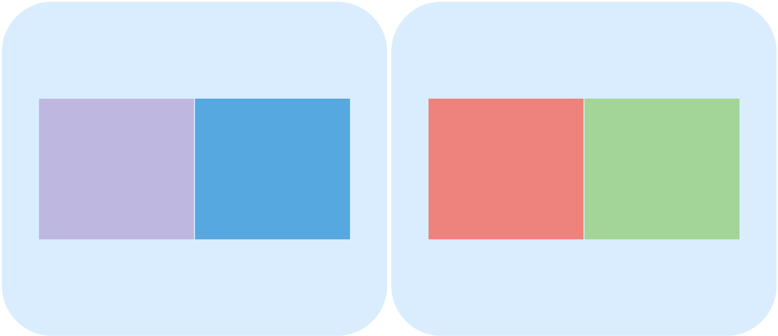


N=6, B=4

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

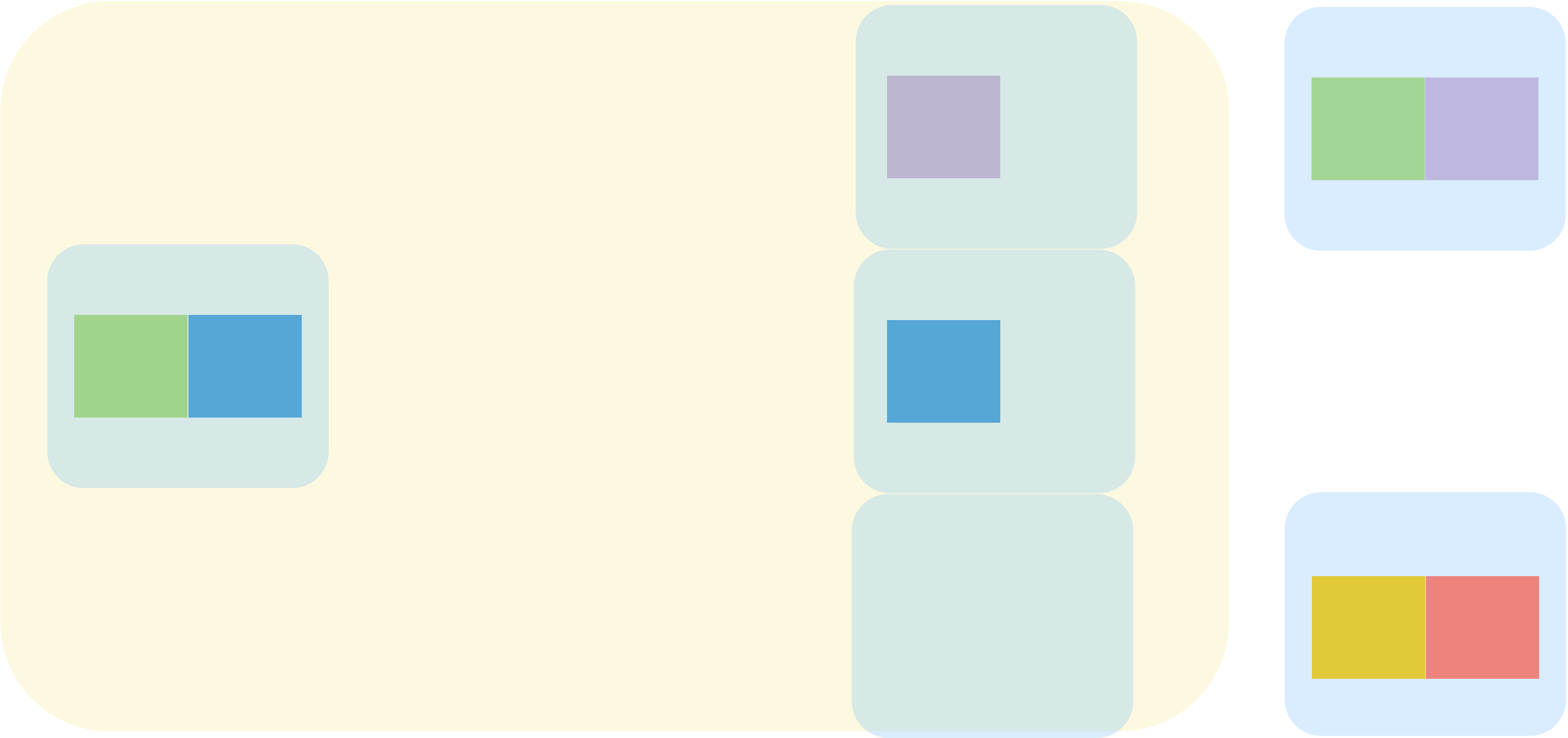


Pass 1: Divide

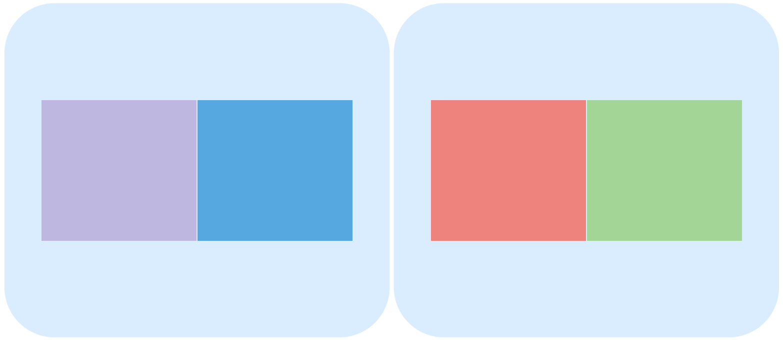


N=6, B=4

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

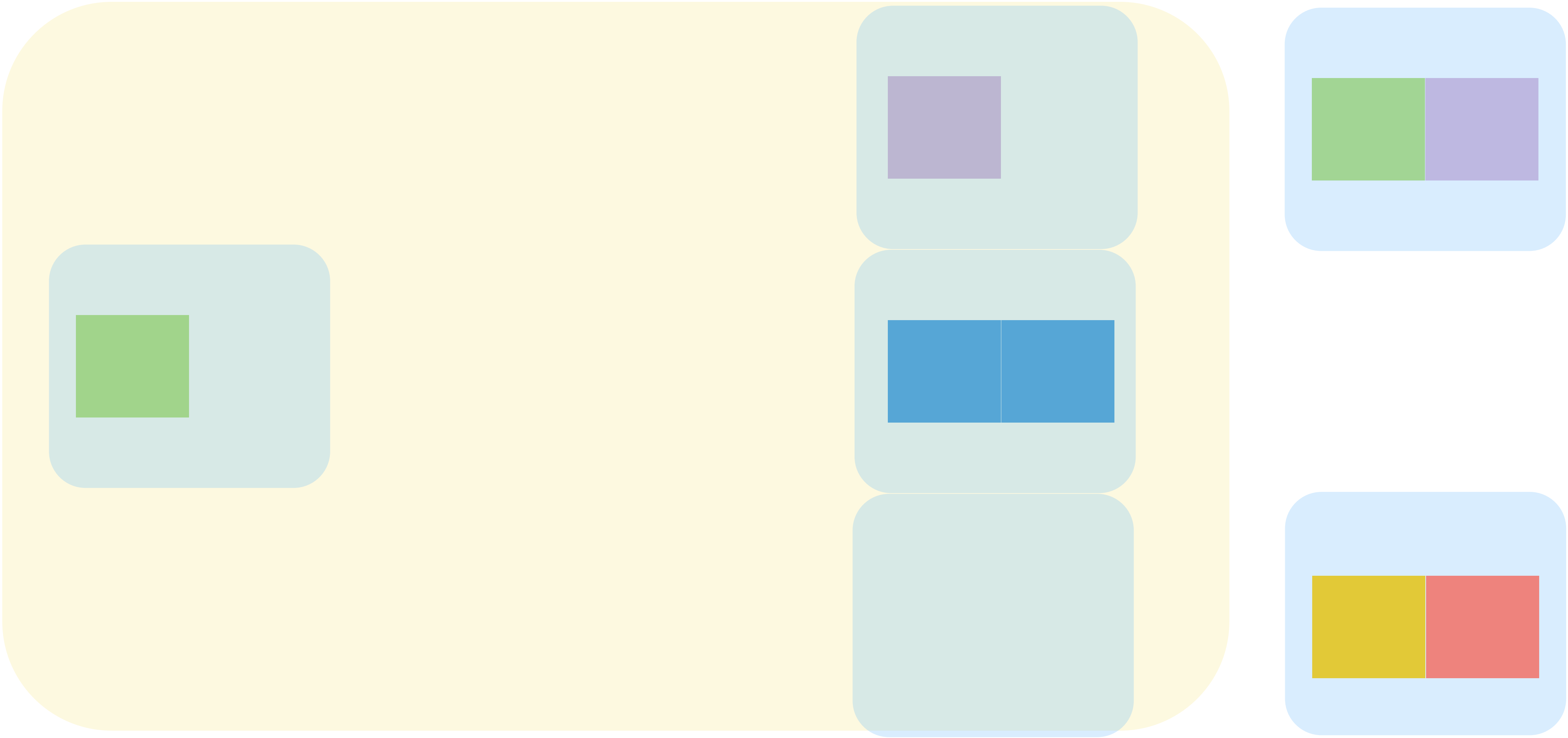


Pass 1: Divide

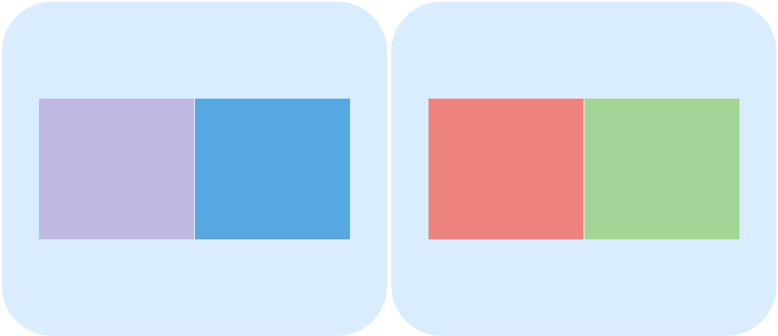


N=6, B=4

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

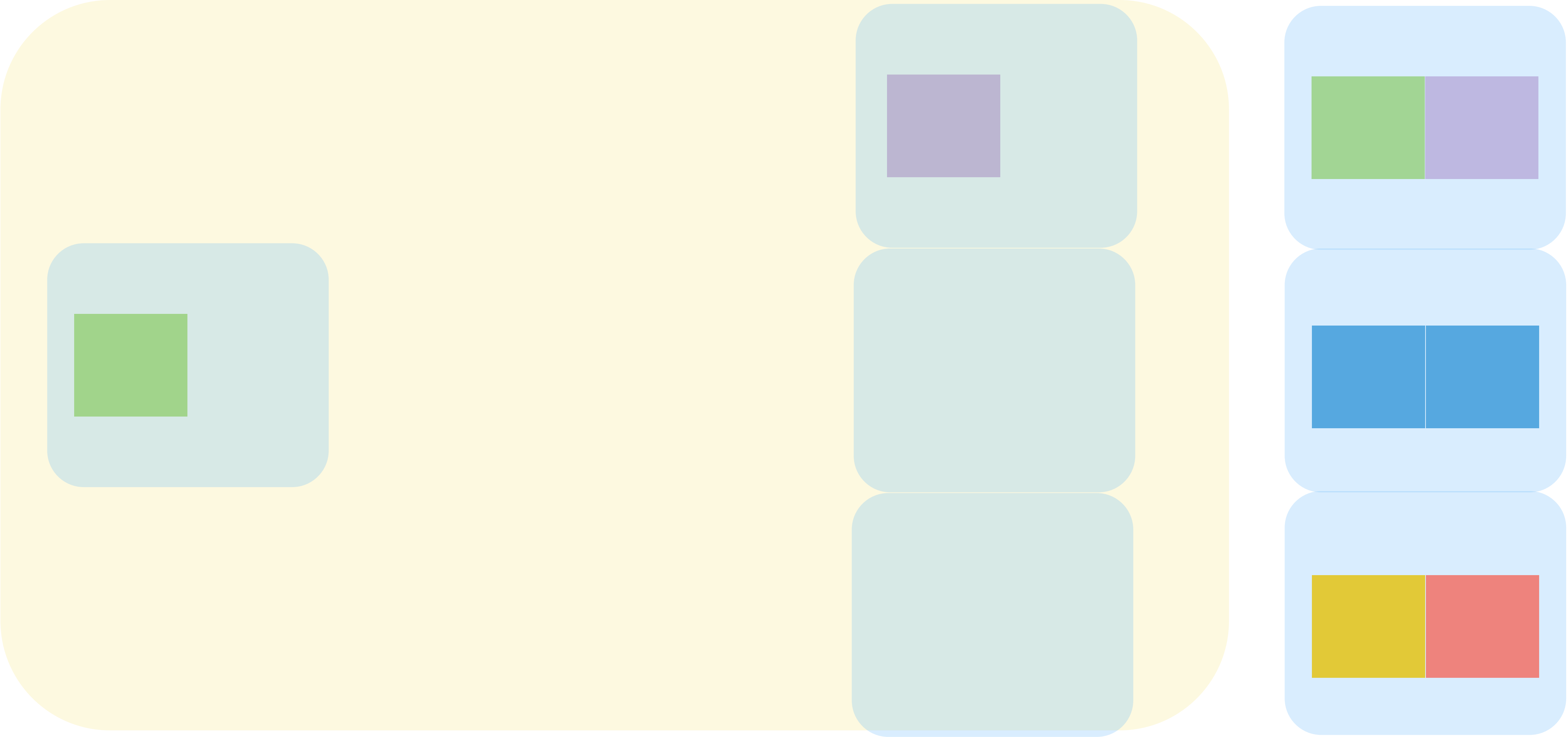


Pass 1: Divide

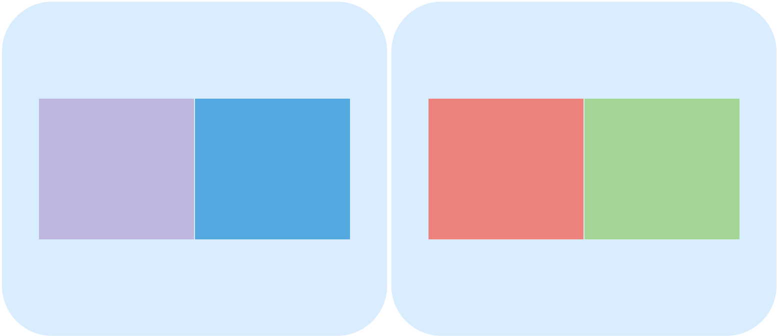


N=6, B=4

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

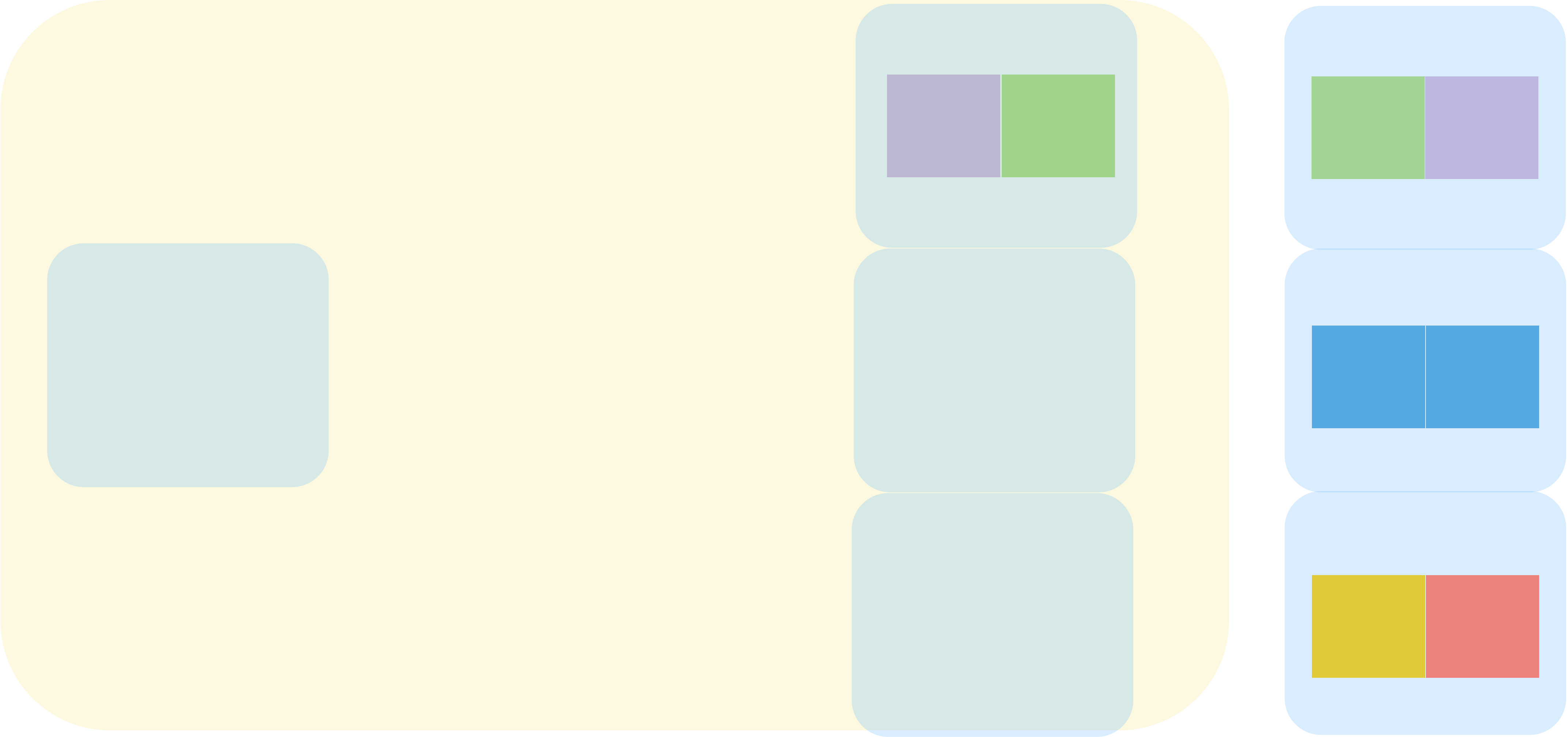


Pass 1: Divide

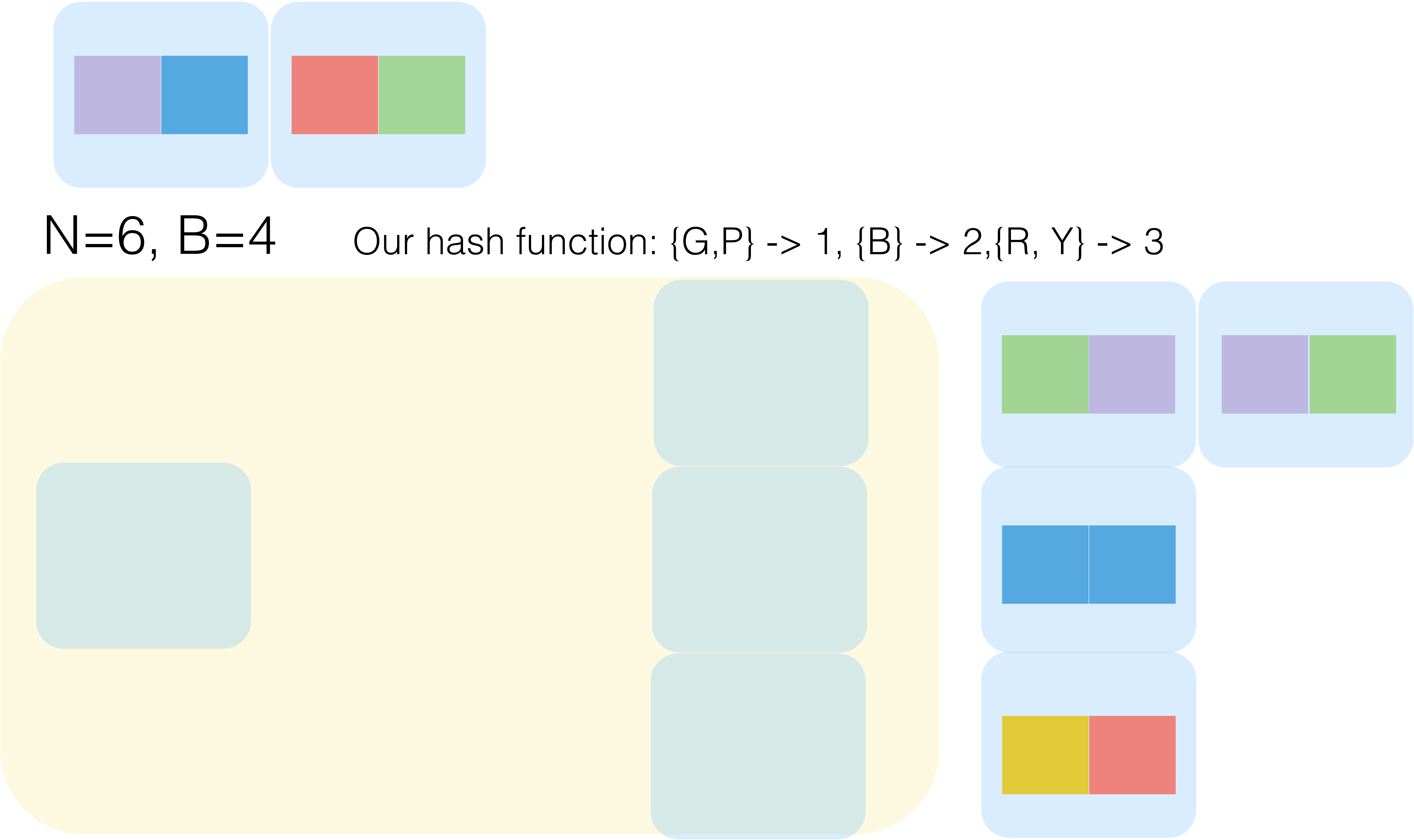


N=6, B=4

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



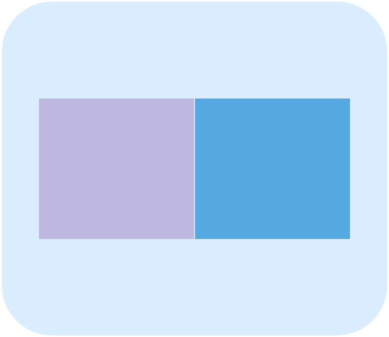
Pass 1: Divide



N=6, B=4

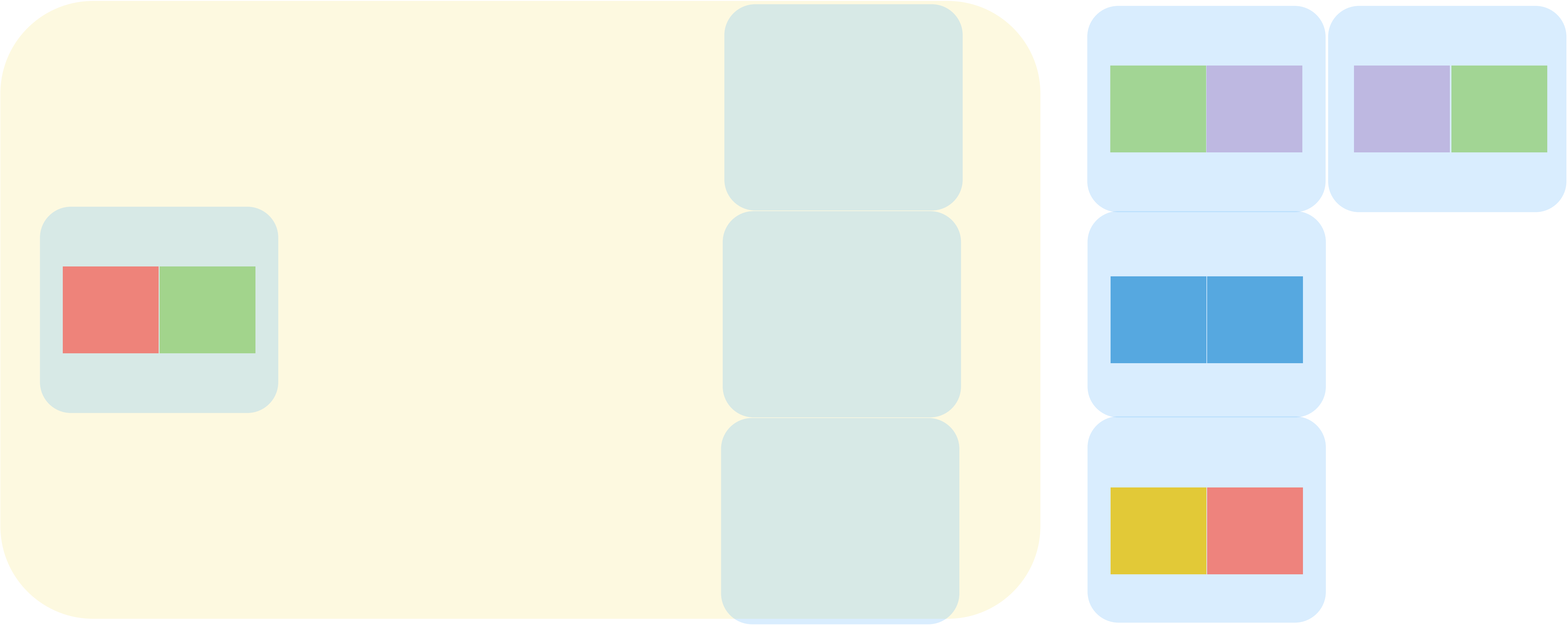
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3

Pass 1: Divide

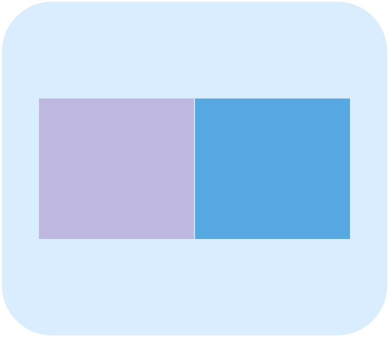


N=6, B=4

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

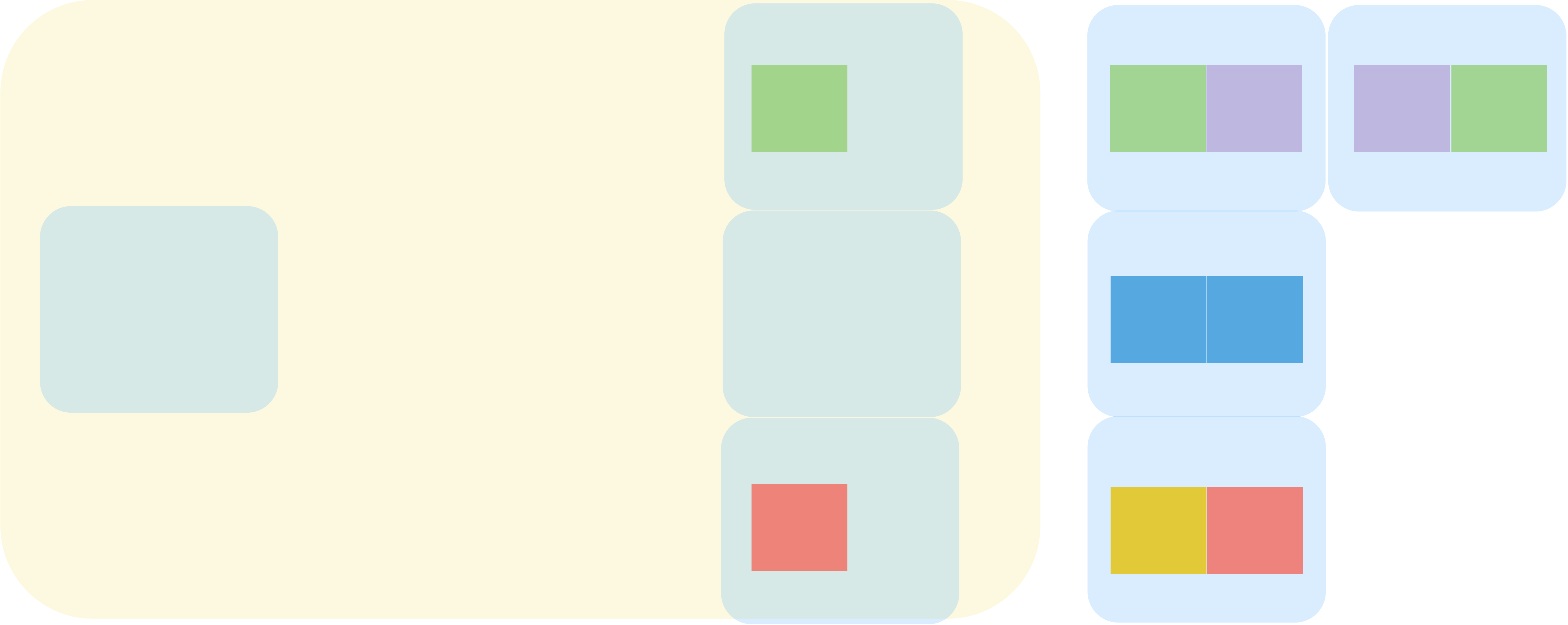


Pass 1: Divide



N=6, B=4

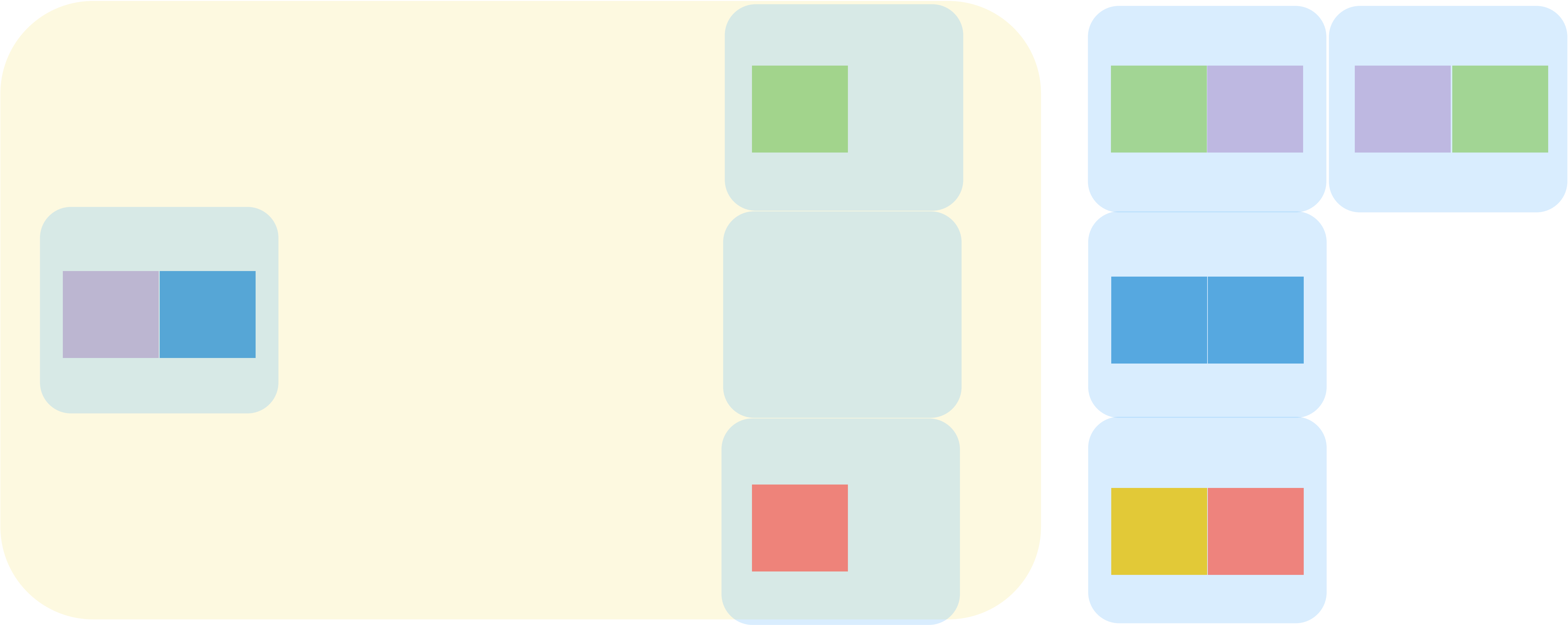
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



Pass 1: Divide

N=6, B=4

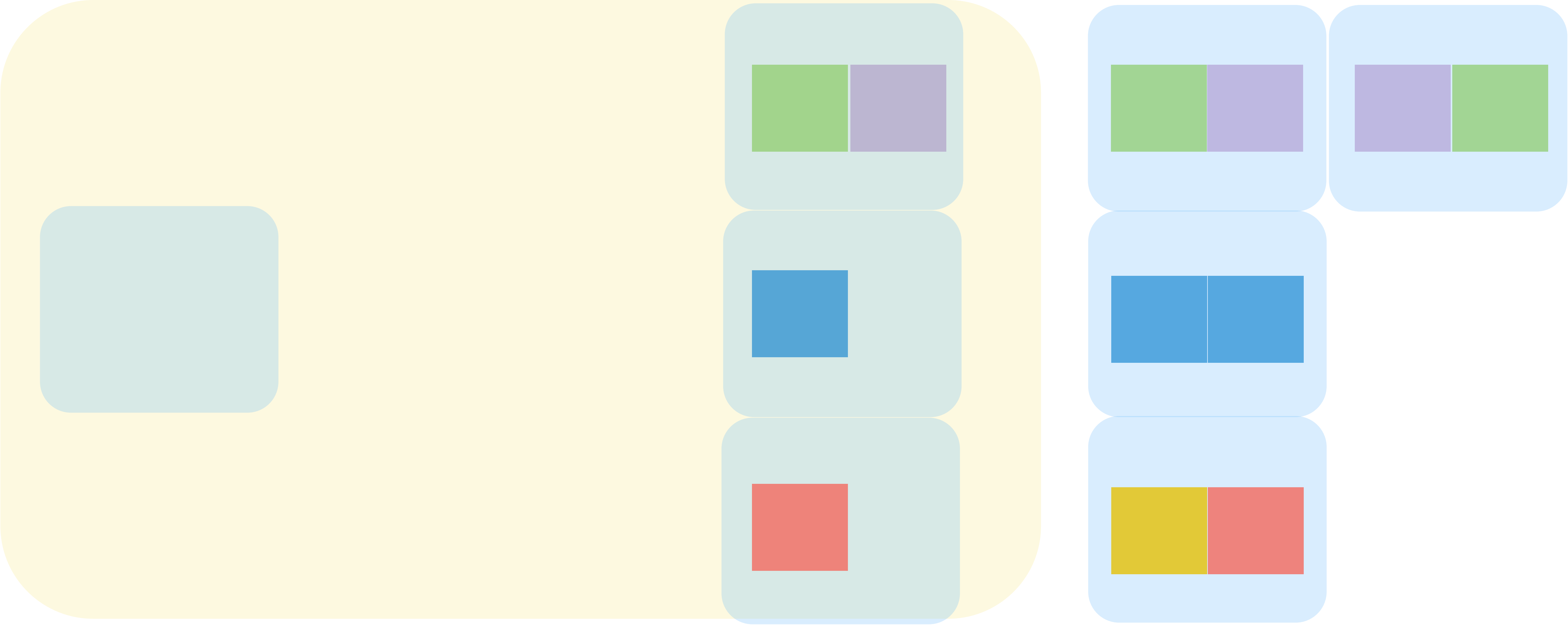
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



Pass 1: Divide

N=6, B=4

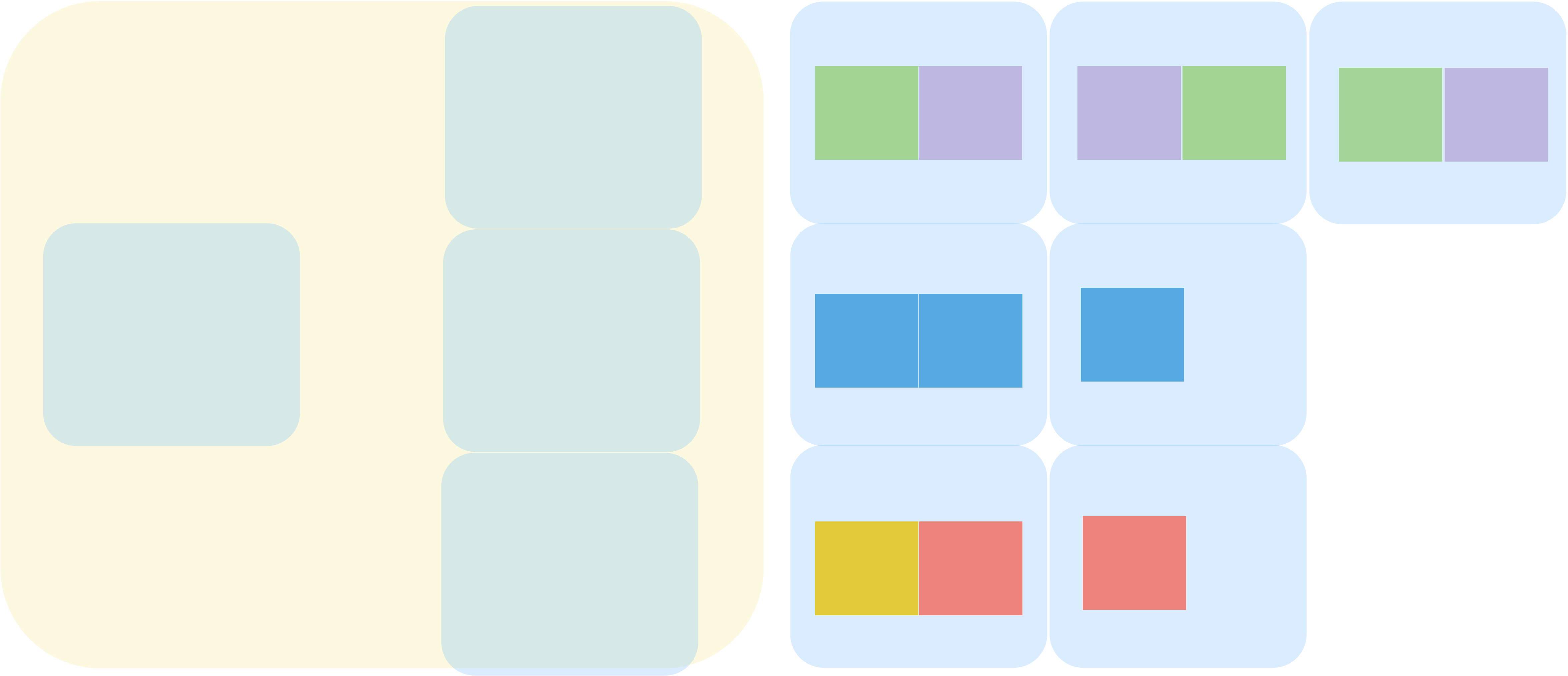
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



Pass 1: Divide

N=6, B=4

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



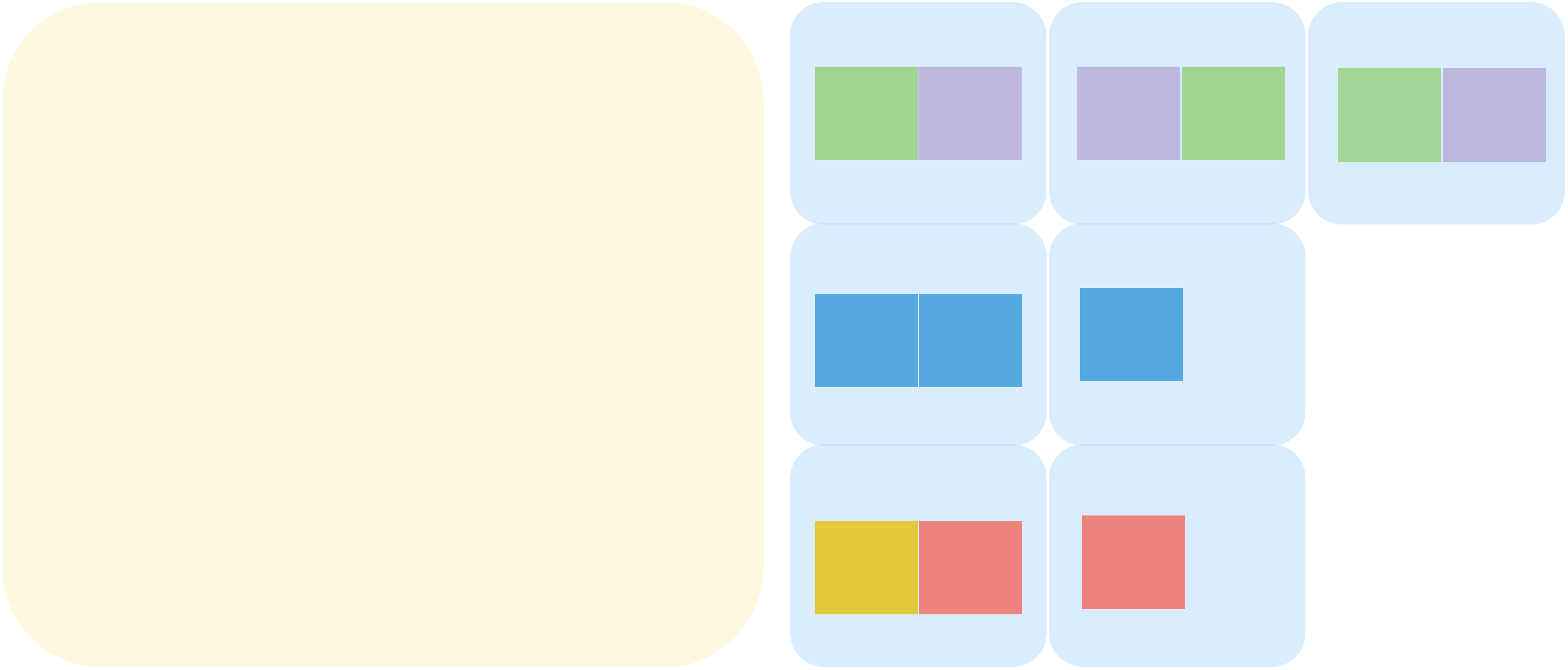
Pass 2: Conquer

- 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.

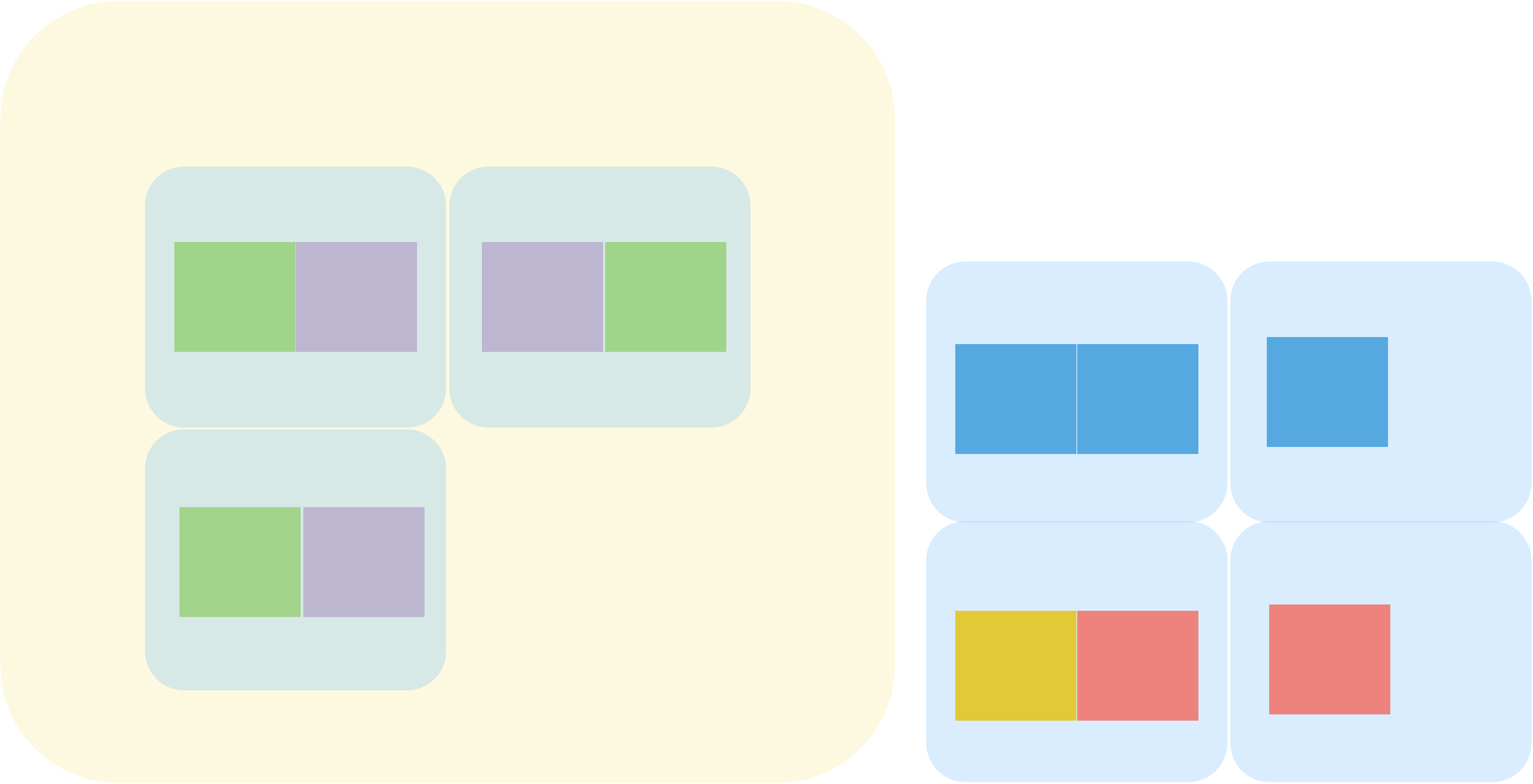
N=6, B=4



Pass 2: Conquer

Create in-memory table for each partition.

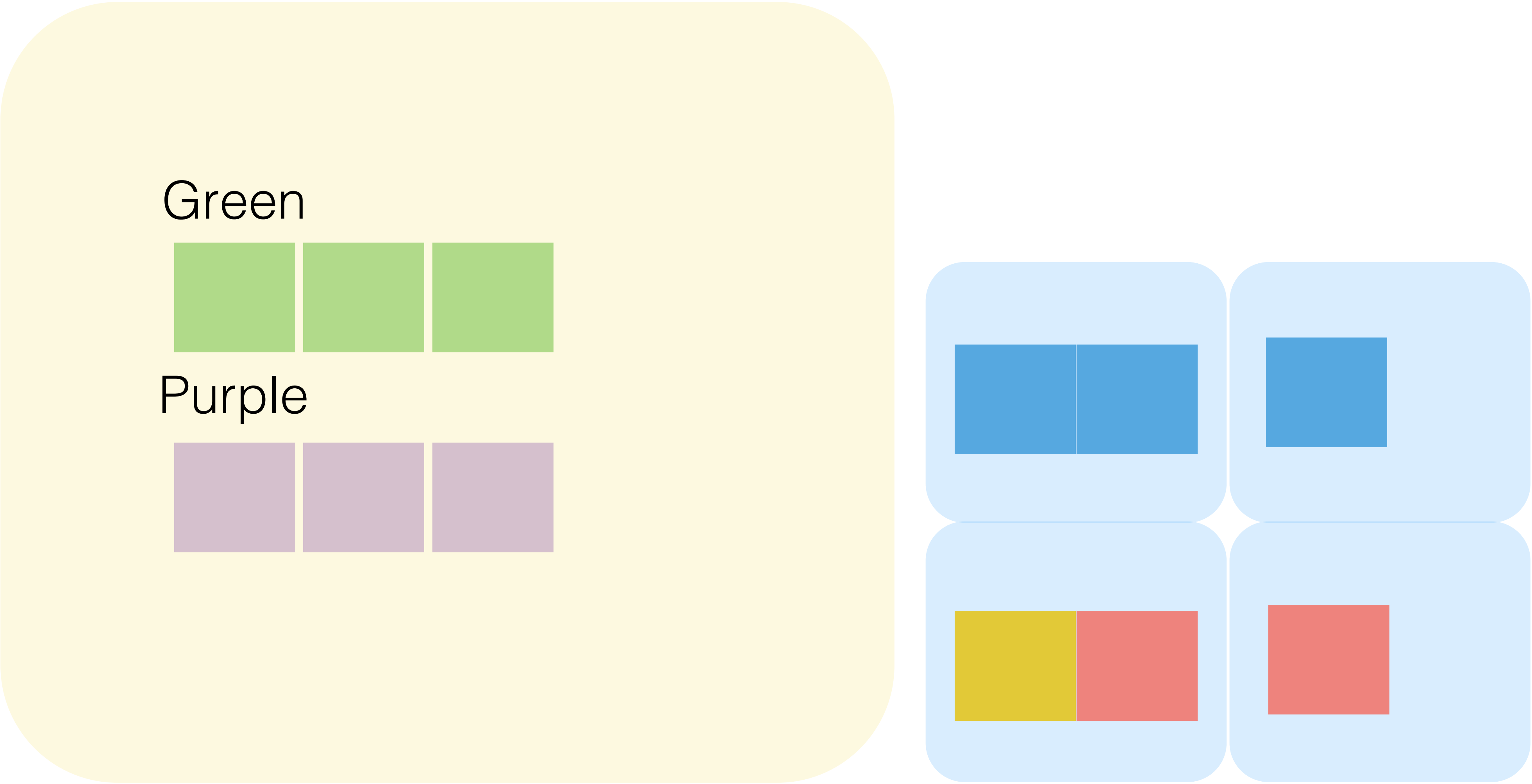
N=6, B=4



Pass 2: Conquer

Create in-memory table for each partition.

N=6, B=4



Worksheet #3, 4, 5

External Hashing Exercises

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

External Hashing Exercises

3. 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.

External Hashing Exercises

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

External Hashing Exercises

4. 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.

External Hashing Exercises

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

External Hashing Exercises

5. 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.