CS 186 Discussion 1

External Sorting & Hashing
SQL

Original Slides + Diagrams by Michelle Nguyen

Course Logistics

www.cs186berkeley.net

- Enrollment
- Vitamins
  - Weekly, Online
  - Released Thursday, Due Monday
- Homework 1 due Thursday 11:59 pm

Pete Yeh

EECS 2016

peteyeh@berkeley.edu

Discussions (2070 VLSB)

Tue 2-3 pm

Tue 3-4 pm

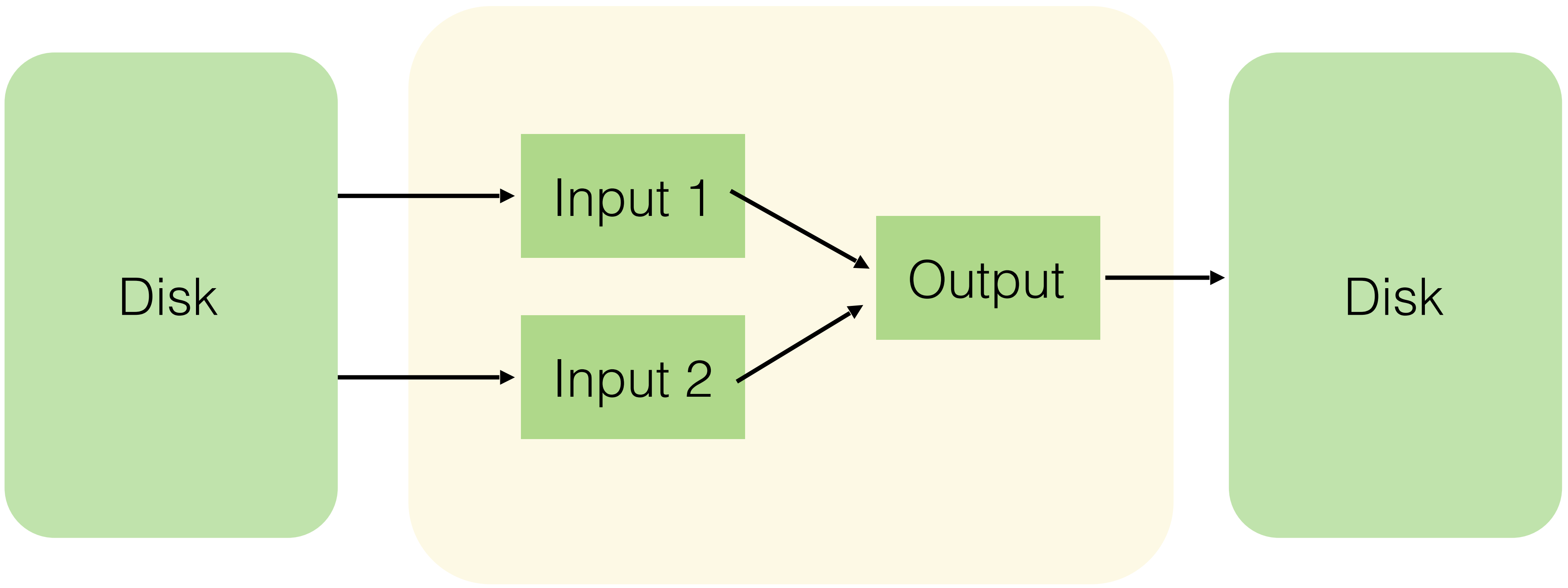
Office Hours (611 Soda)

Thurs 12-2 pm

Meet New Friends!

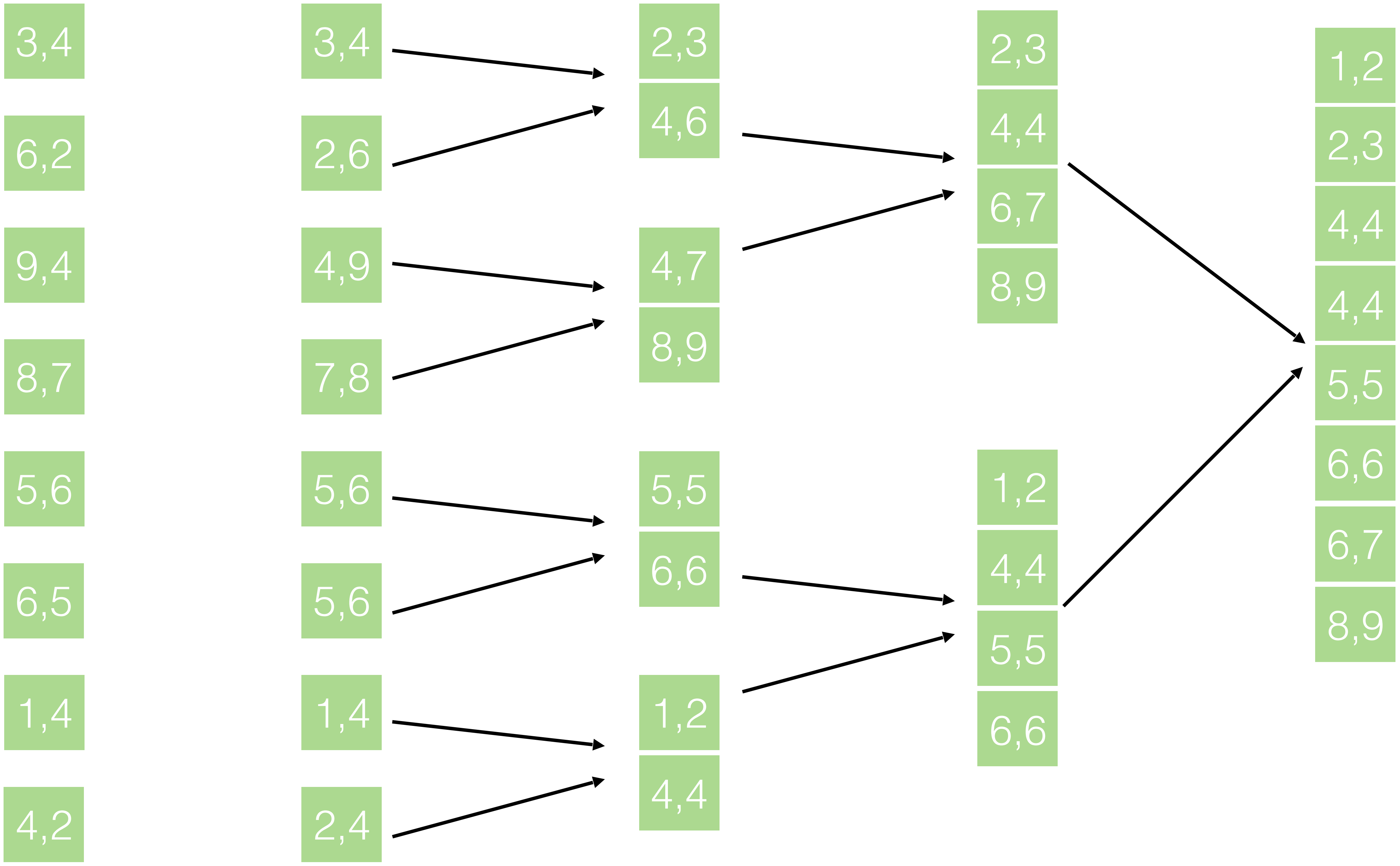
External Sorting

- Two-way Merge Sort (Merge Step)



- Buffer size of 3 pages

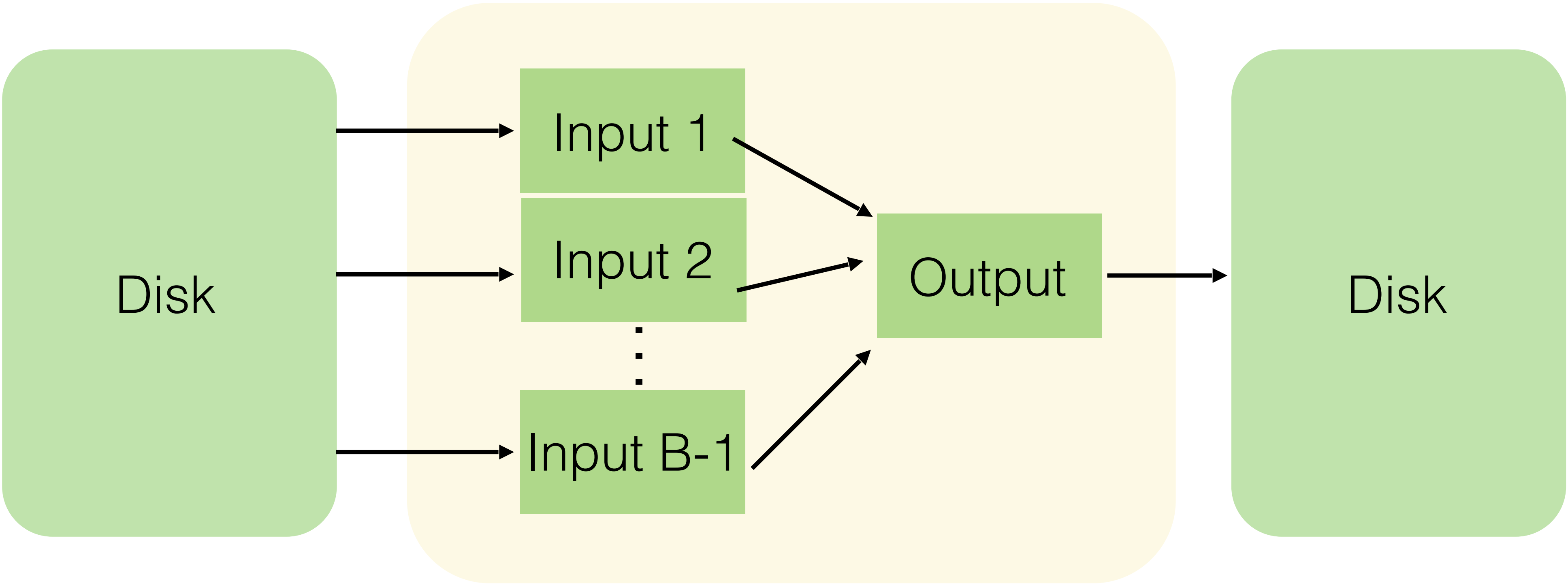
External Sorting



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

External Sorting

- General Merge Sort (Merge Step)



- Buffer size of *B* pages

External Sorting

- *N* blocks in file, *B* blocks in memory
- Number of Passes
  - Two-way $$\lceil \log_2 N \rceil + 1$$
  - Generalized $$\left\lceil \log_{B-1} \left\lceil \frac{N}{B} \right\rceil \right\rceil + 1$$
- Total Cost (I/Os)

2*N* * [# of passes]

External Sorting

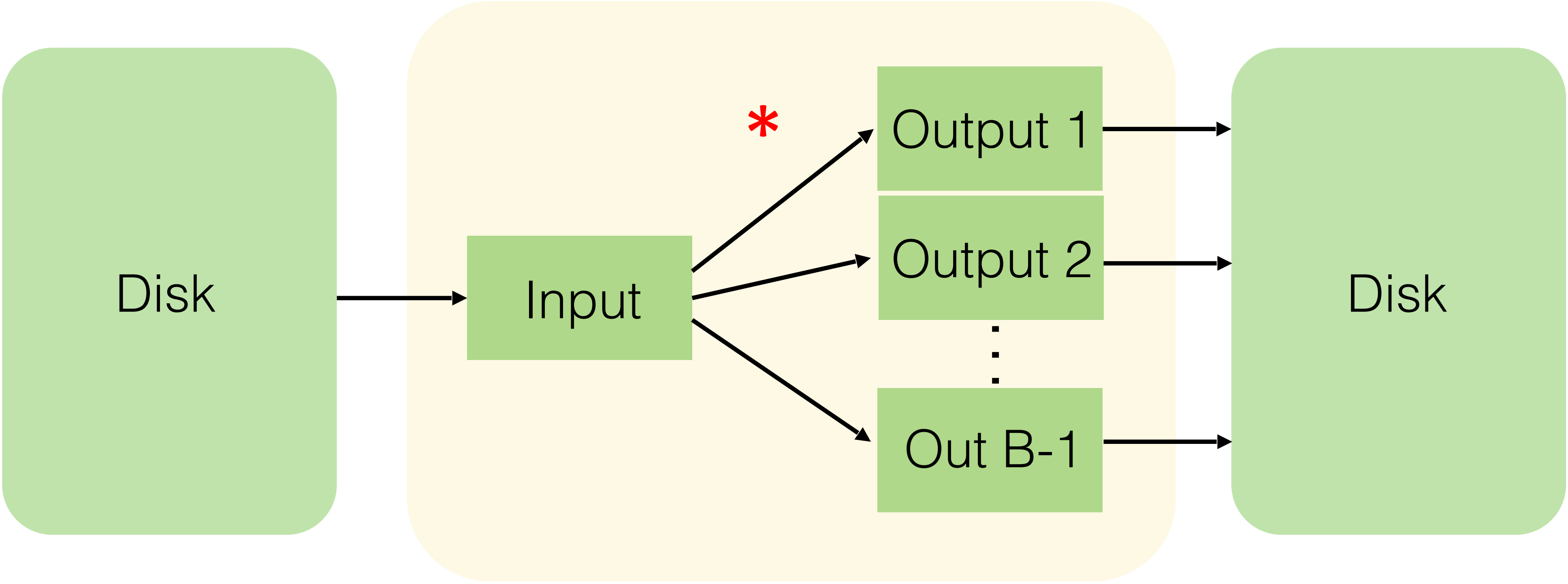
- How big of a file can we sort in two passes?

$$B(B-1)$$

- Why?

External Hashing

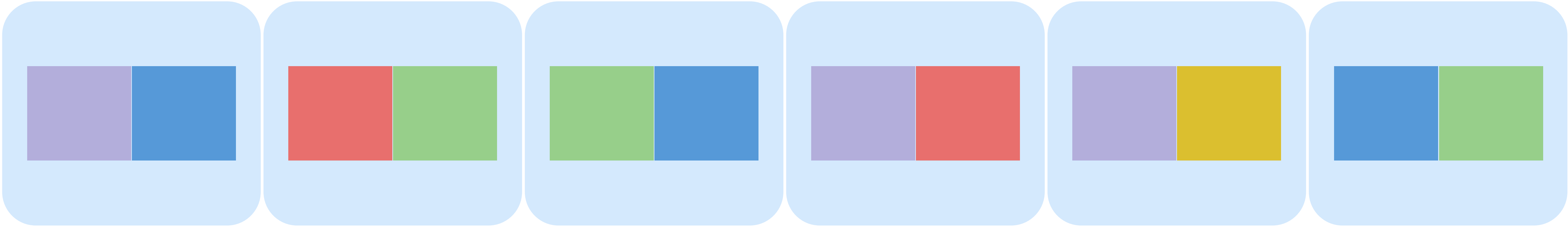
- Partition (Divide) Step



- Buffer size of *B* pages
- * = hash function!

Aggregating Colors

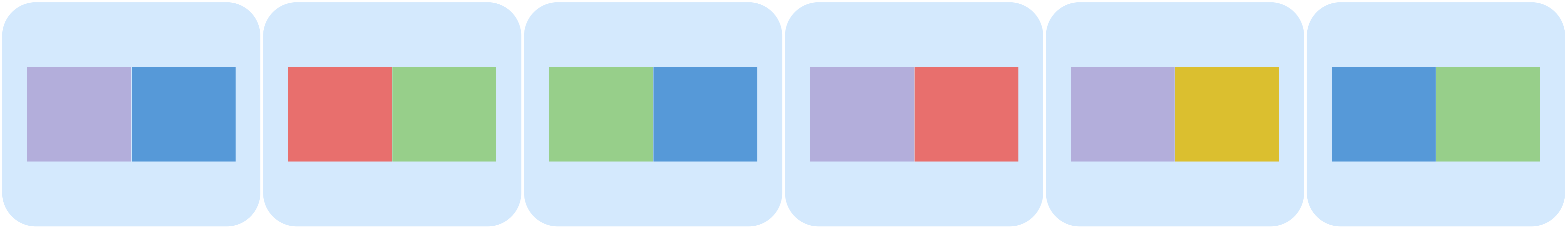
- Goal: Group squares by color
- Setup: 12 squares, each page fits 2 squares. We can hold 4 pages in memory.
- $N = 6$, $B = 4$



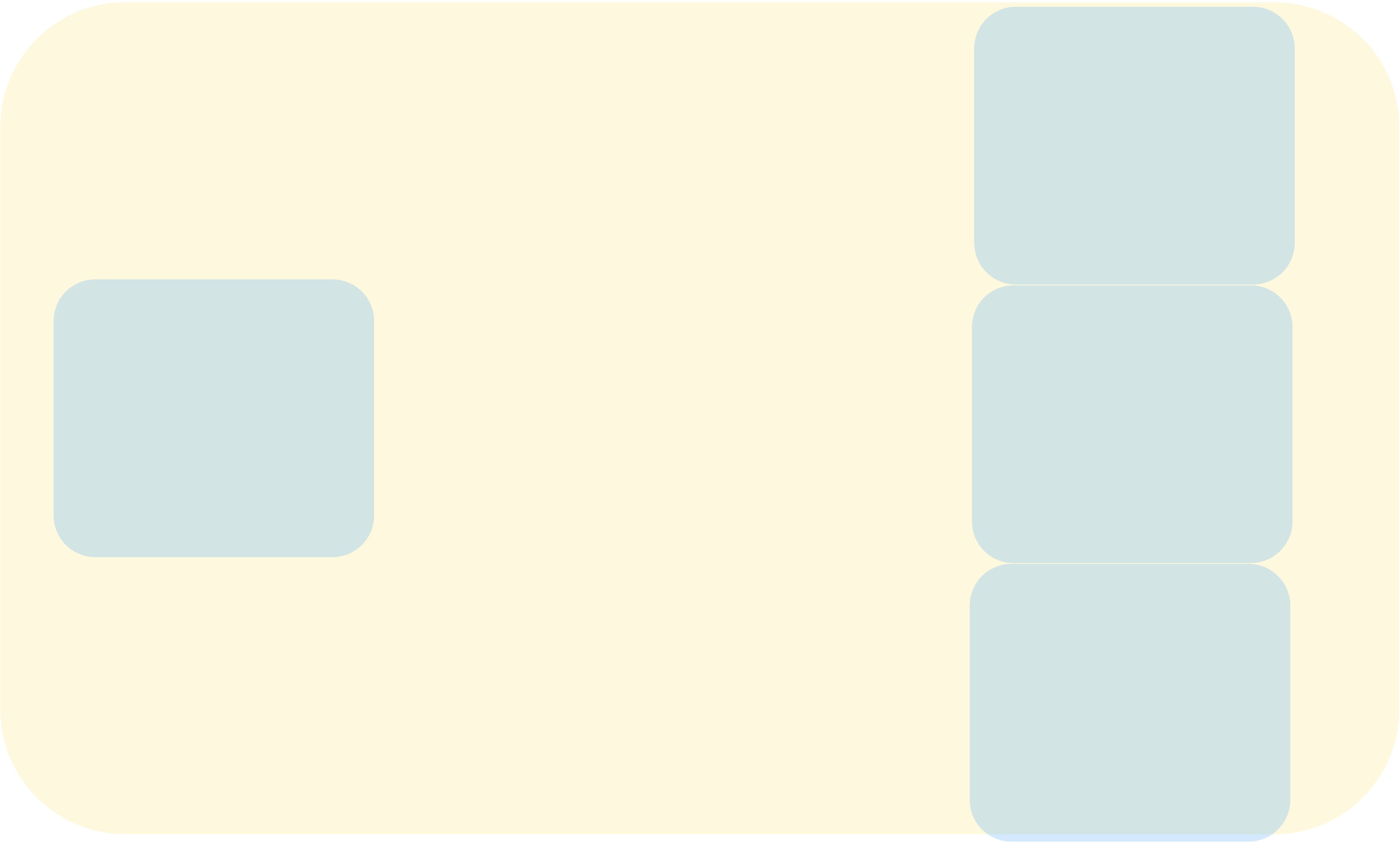
Pass 1: Divide

- 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.

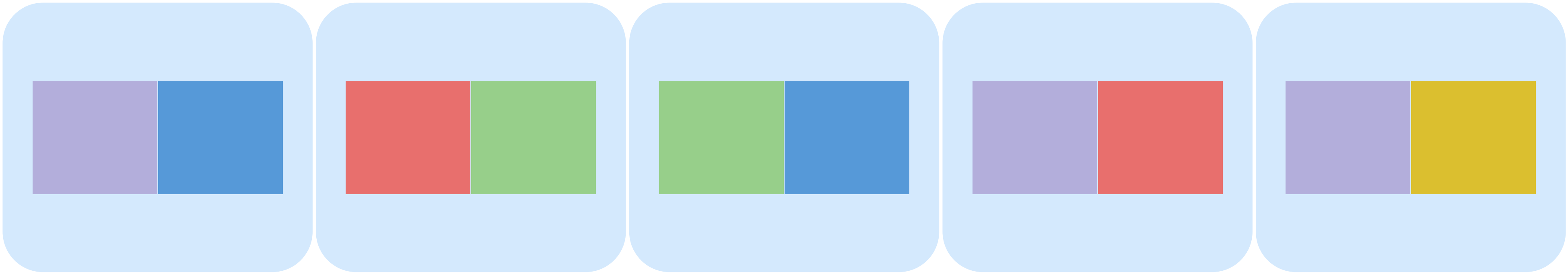
Our hash function:

{G,P} -> 1

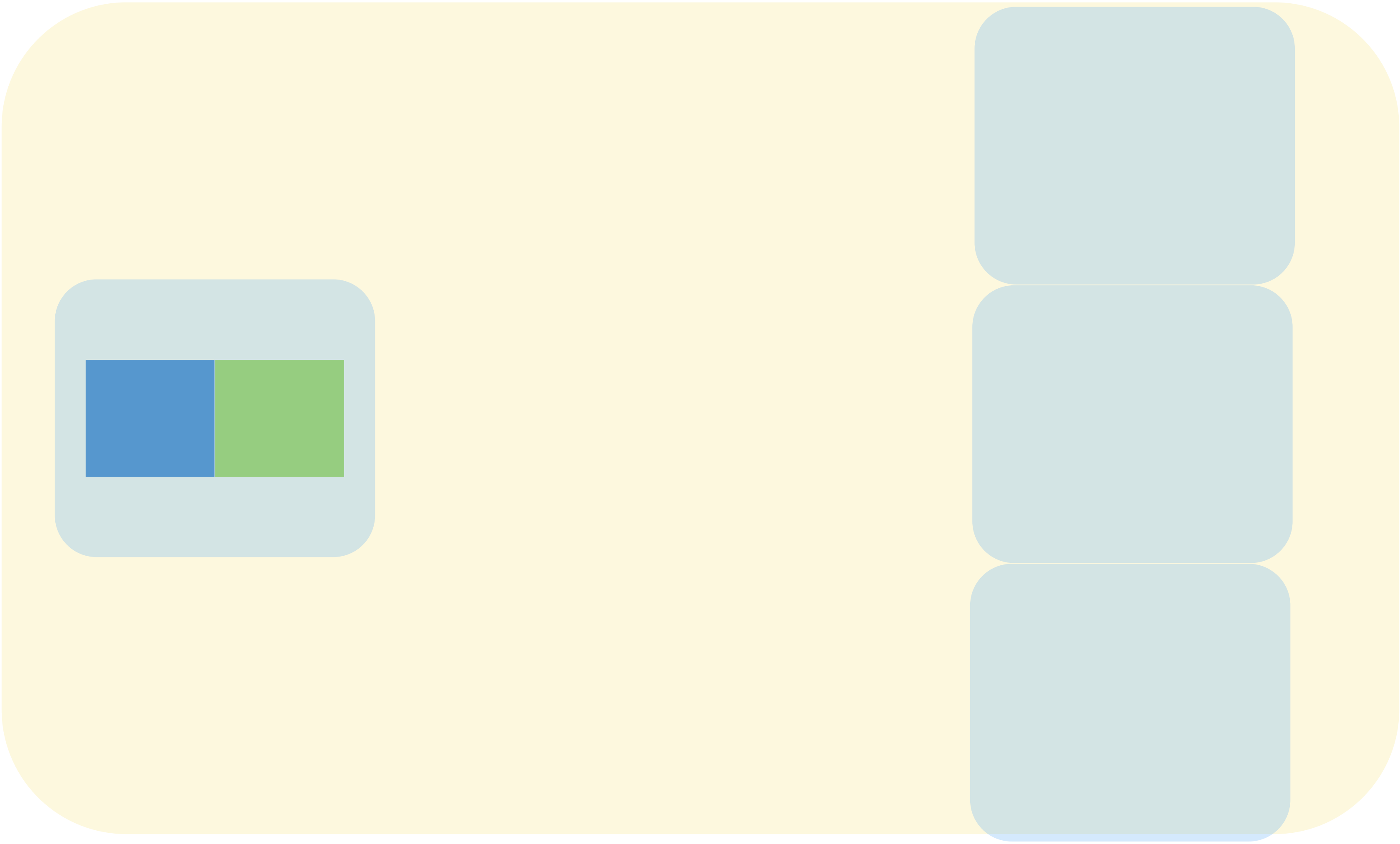
{B} -> 2

{R, Y} -> 3

Pass 1: Divide



N=6, B=4



Assign colors to 3 partitions using hash function.

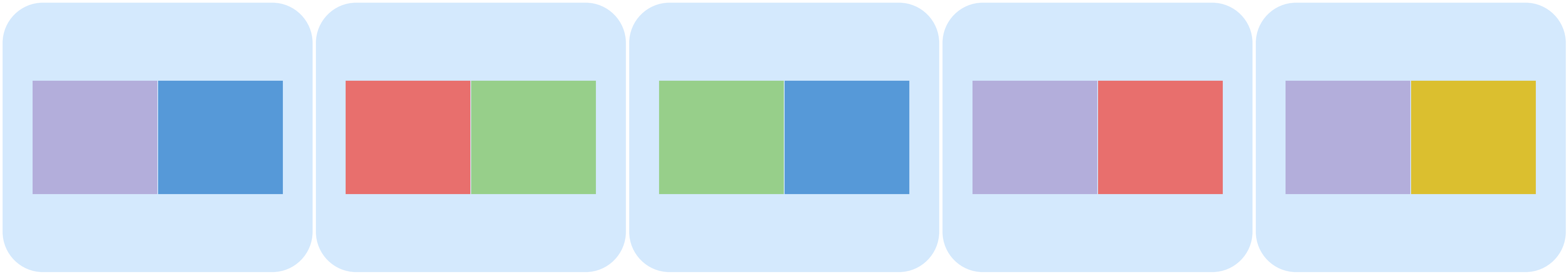
Our hash function:

{G,P} -> 1

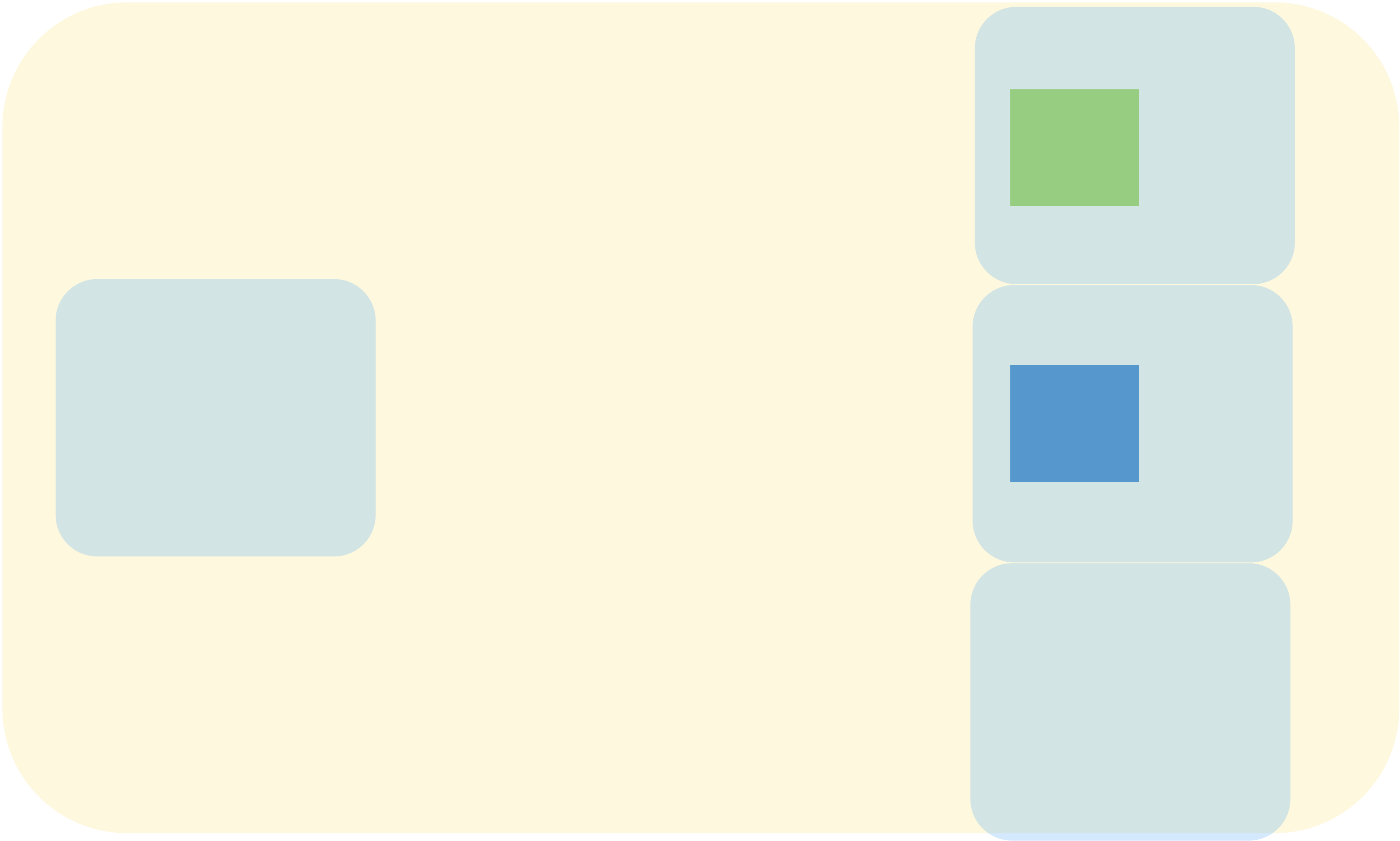
{B} -> 2

{R, Y} -> 3

Pass 1: Divide



N=6, B=4



Assign colors to 3 partitions using hash function.

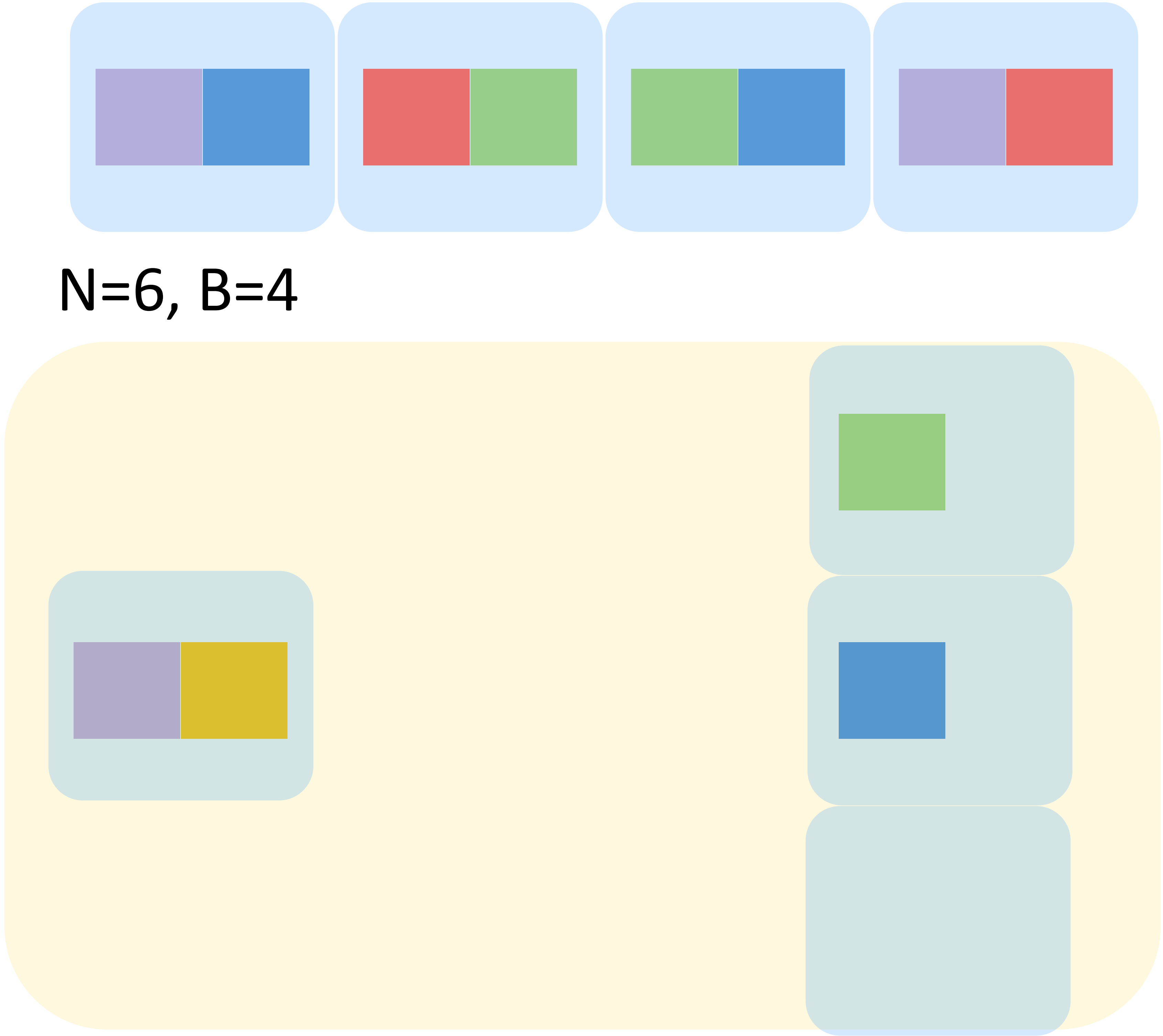
Our hash function:

{G,P} -> 1

{B} -> 2

{R, Y} -> 3

Pass 1: Divide



N=6, B=4

Assign colors to 3 partitions using hash function.

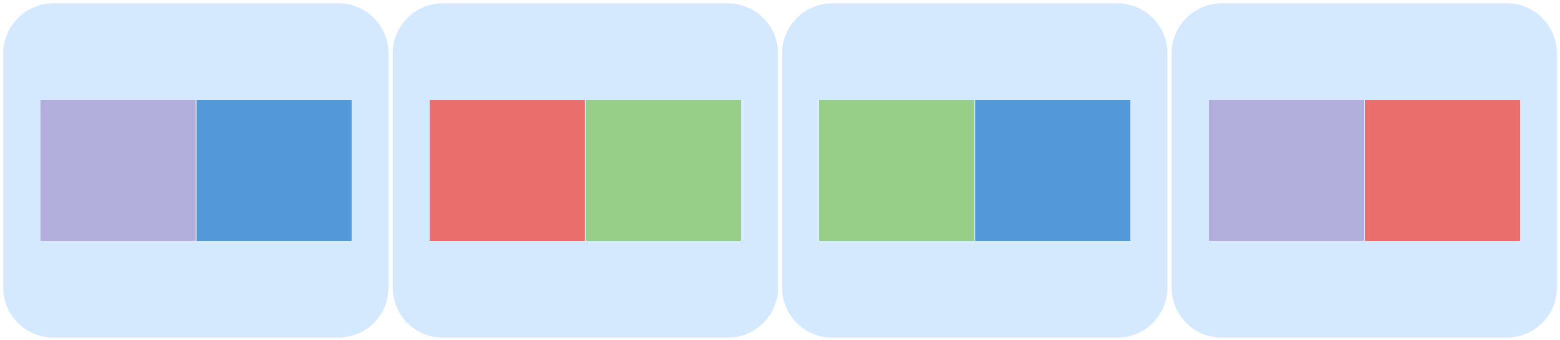
Our hash function:

{G,P} -> 1

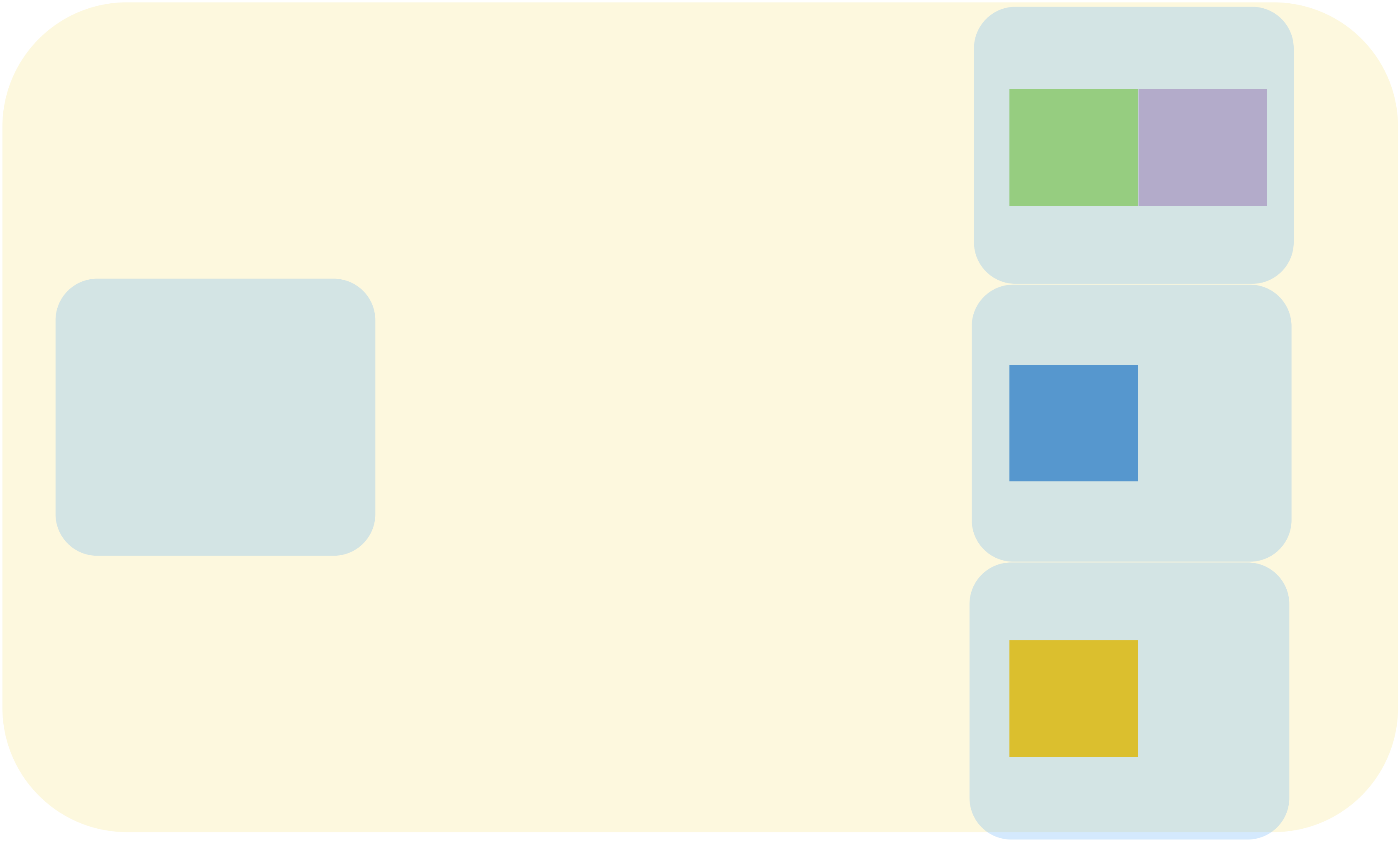
{B} -> 2

{R, Y} -> 3

Pass 1: Divide



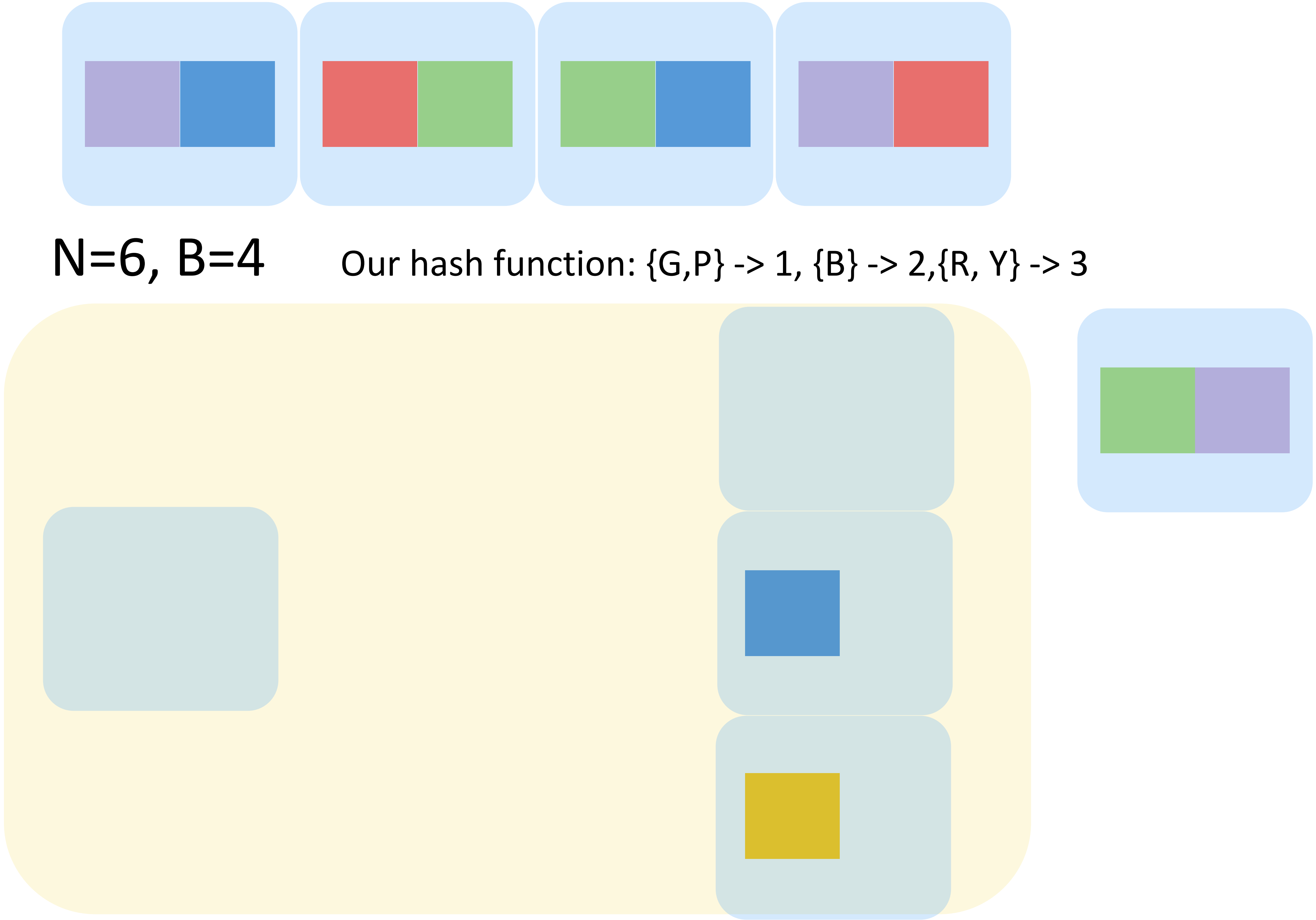
N=6, B=4



Assign colors to 3 partitions using hash function.

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

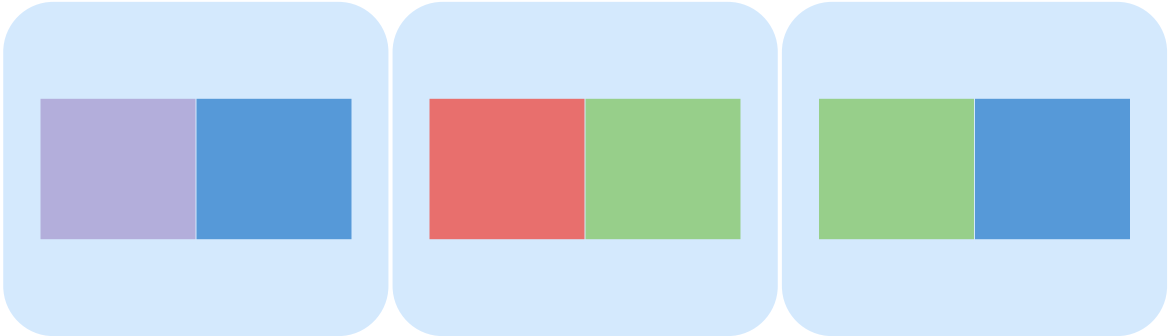
Pass 1: Divide



N=6, B=4

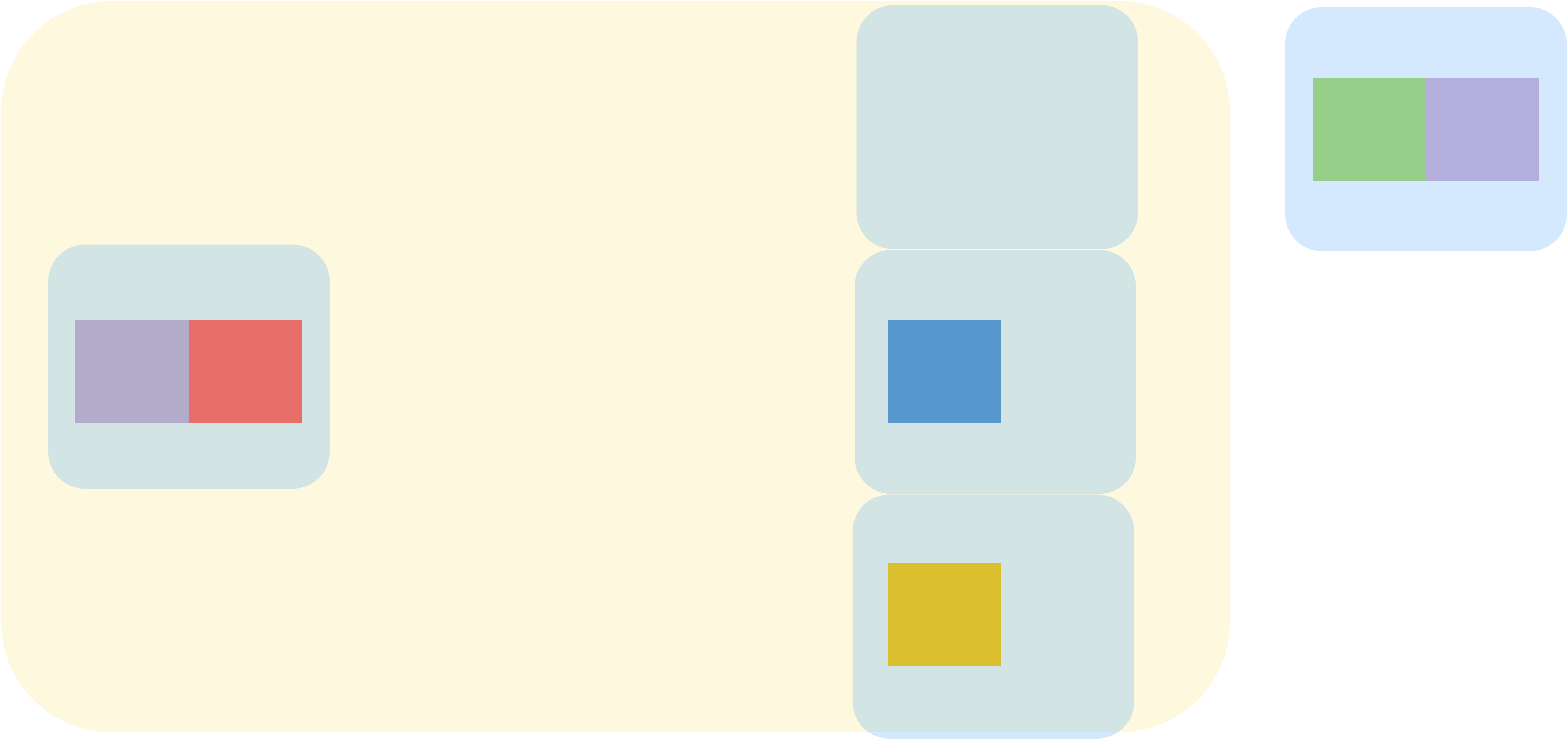
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3

Pass 1: Divide

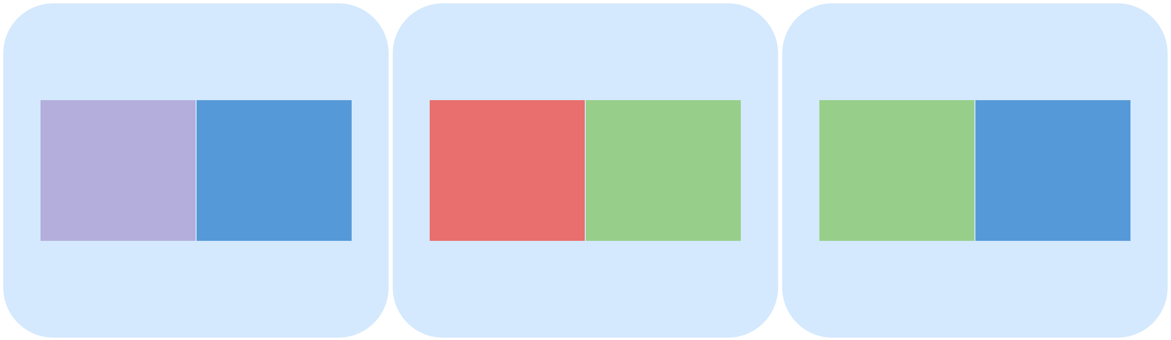


N=6, B=4

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

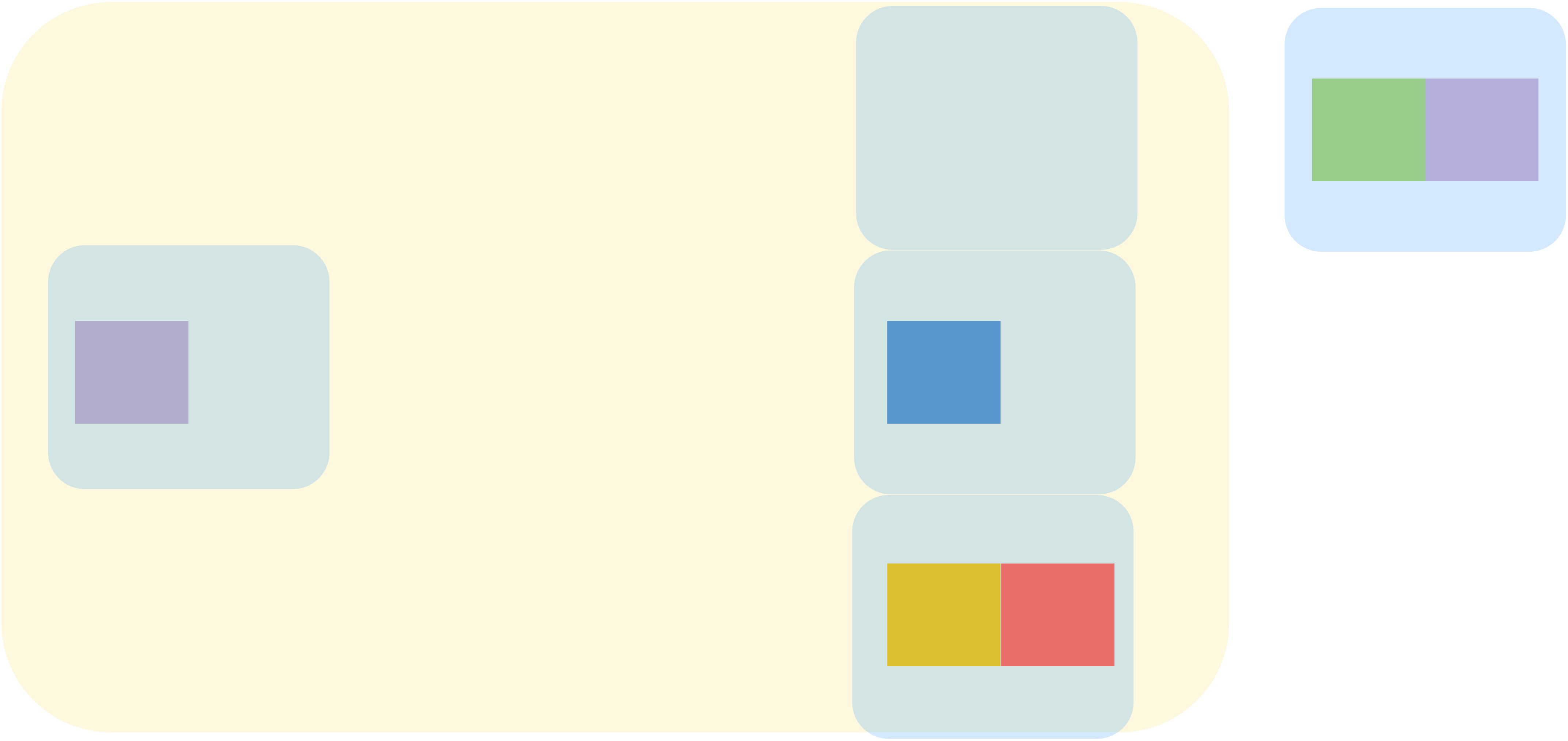


Pass 1: Divide

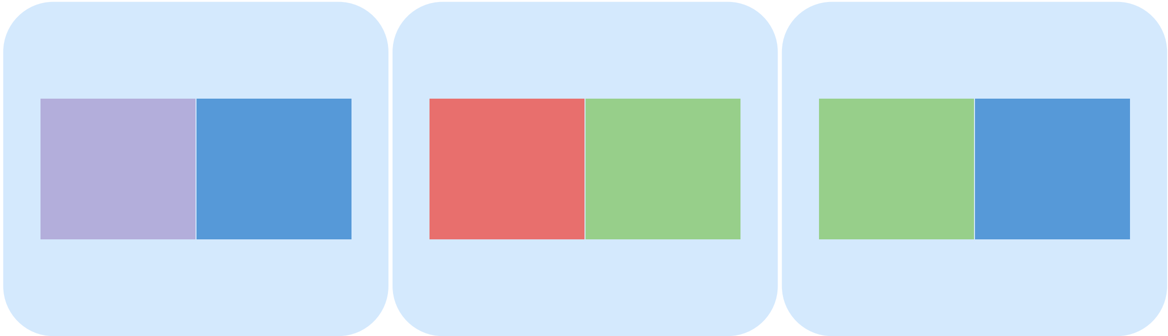


N=6, B=4

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

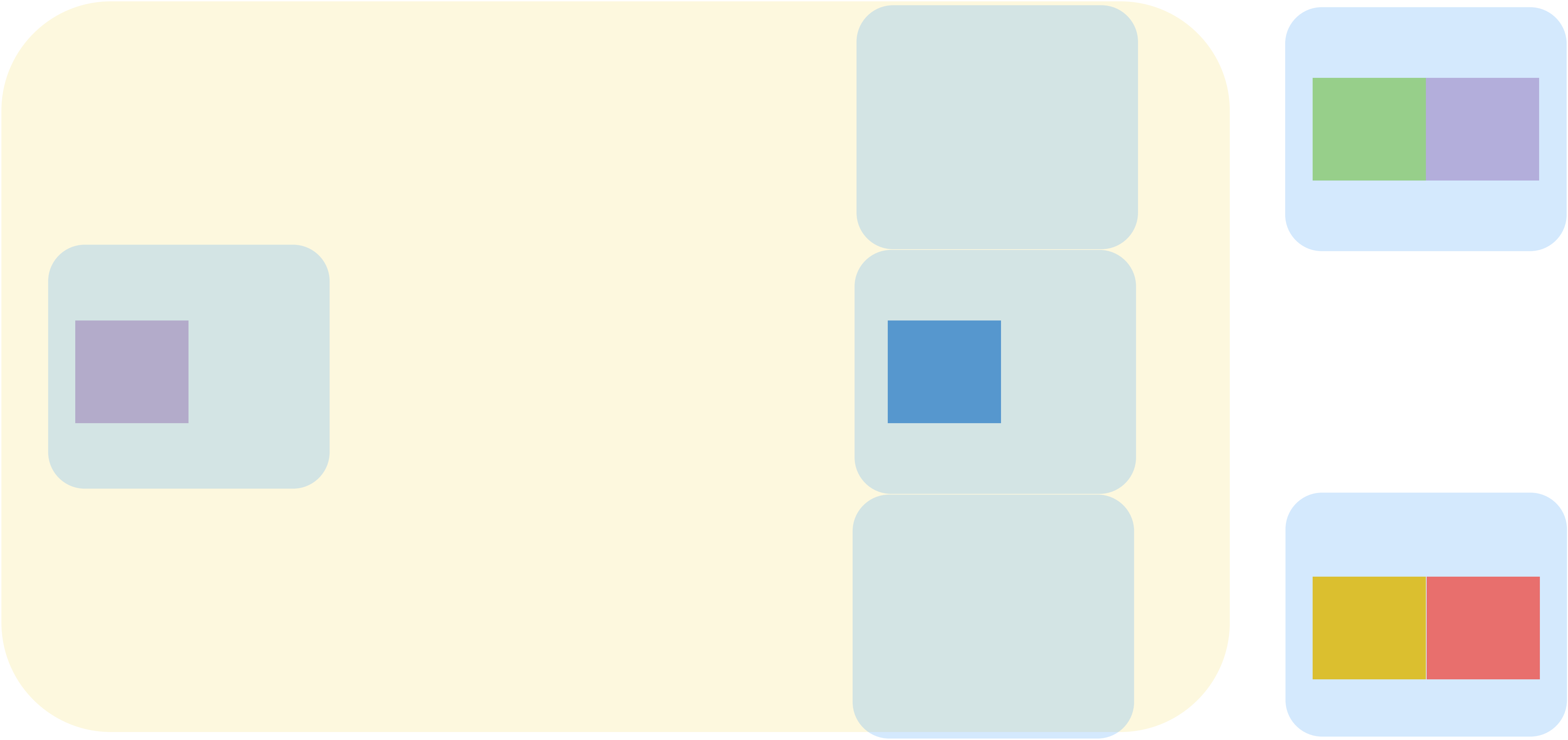


Pass 1: Divide

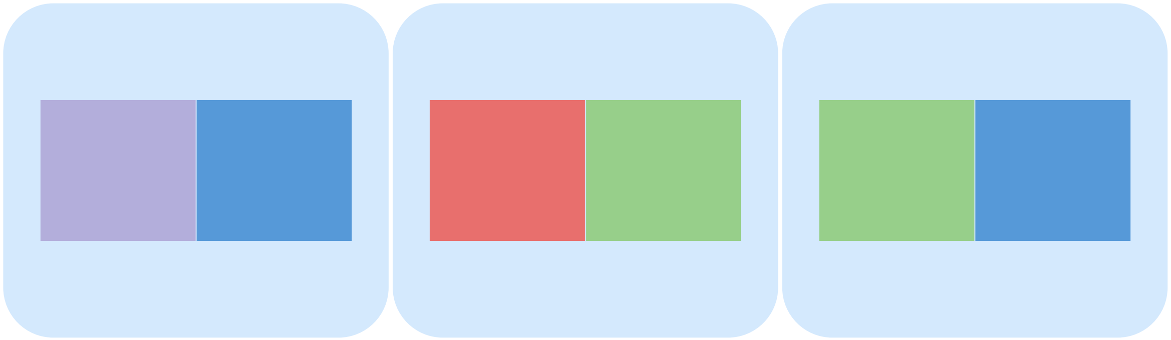


N=6, B=4

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

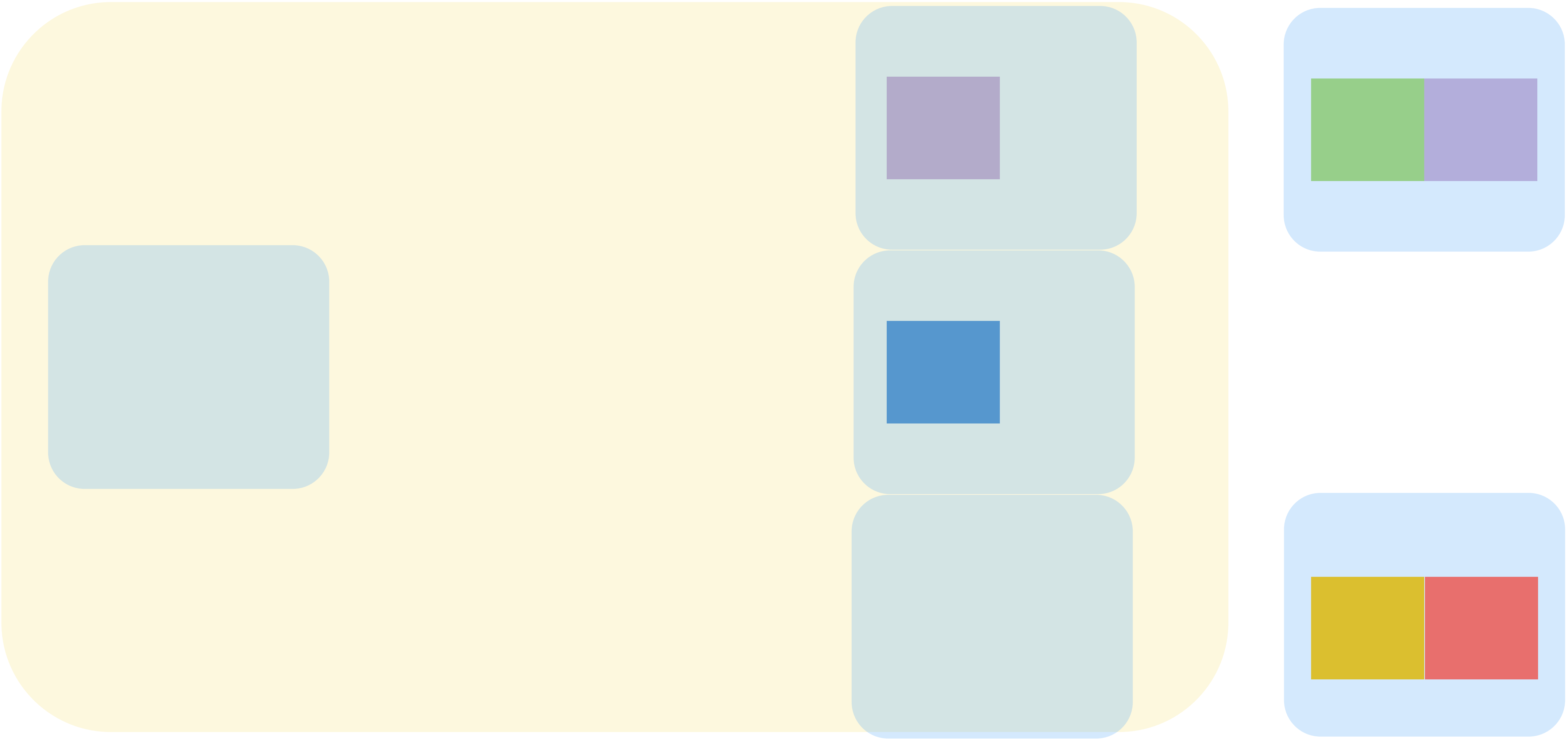


Pass 1: Divide

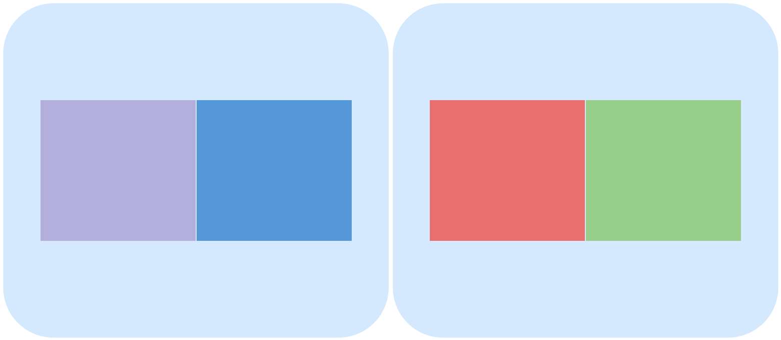


N=6, B=4

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

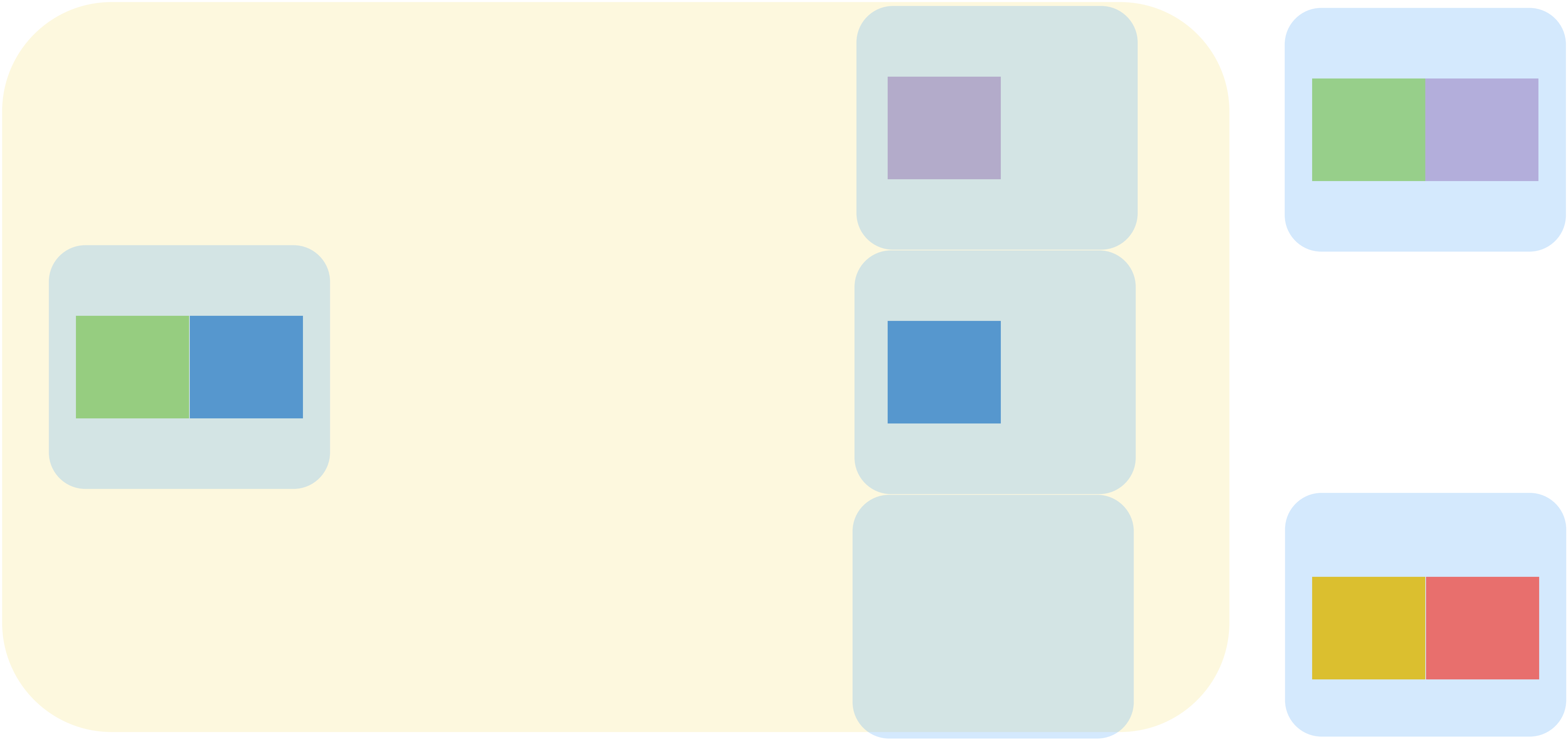


Pass 1: Divide

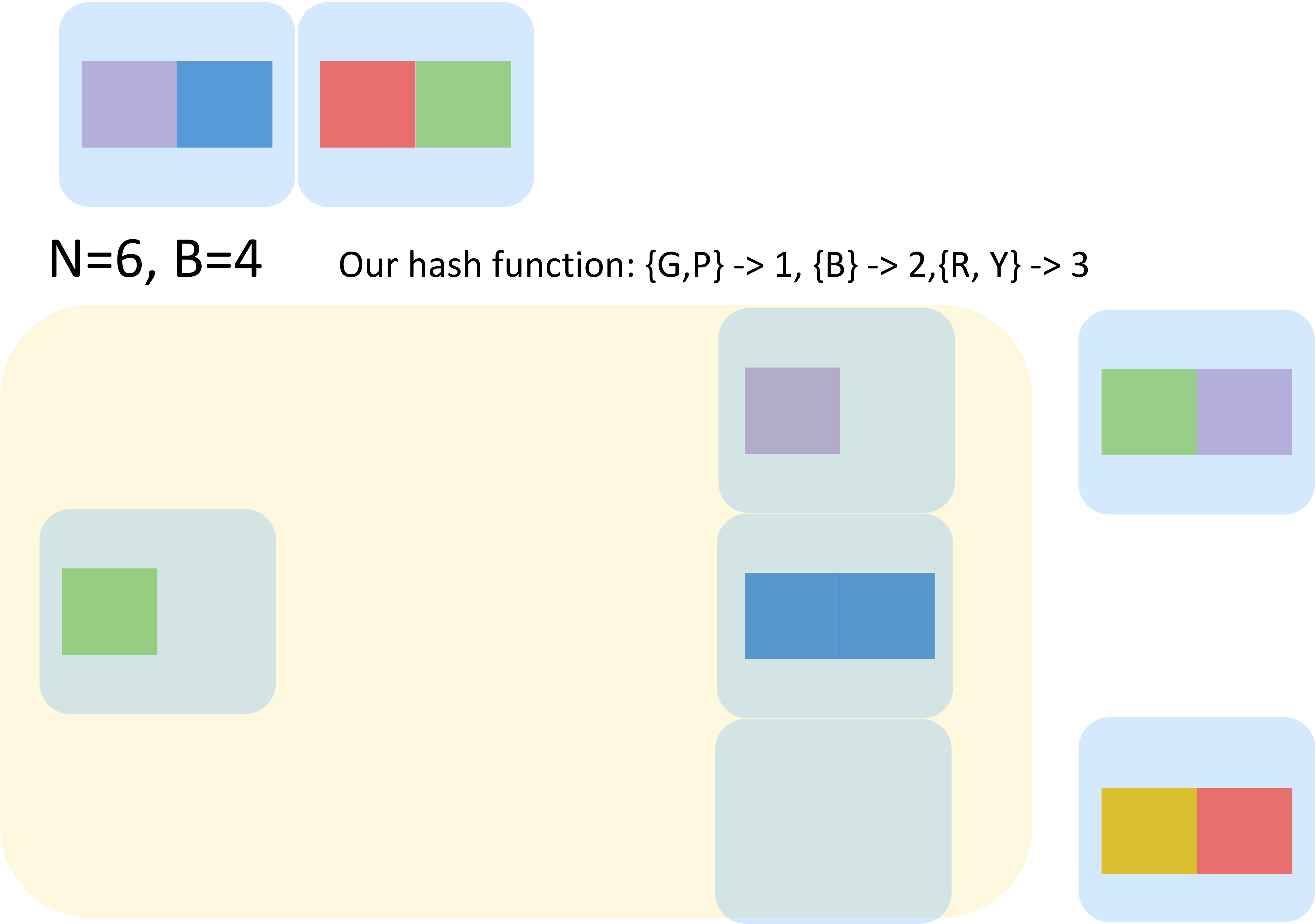


N=6, B=4

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



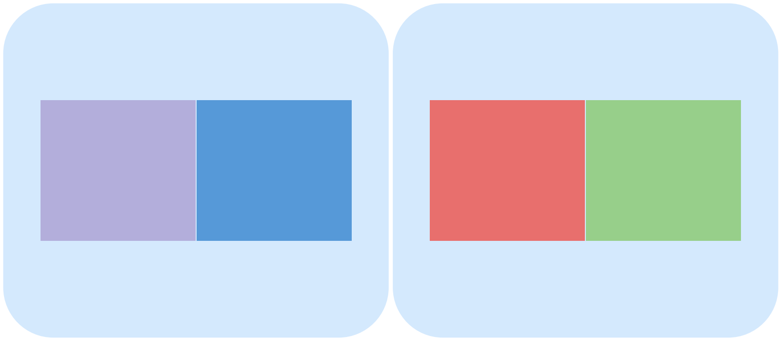
Pass 1: Divide



N=6, B=4

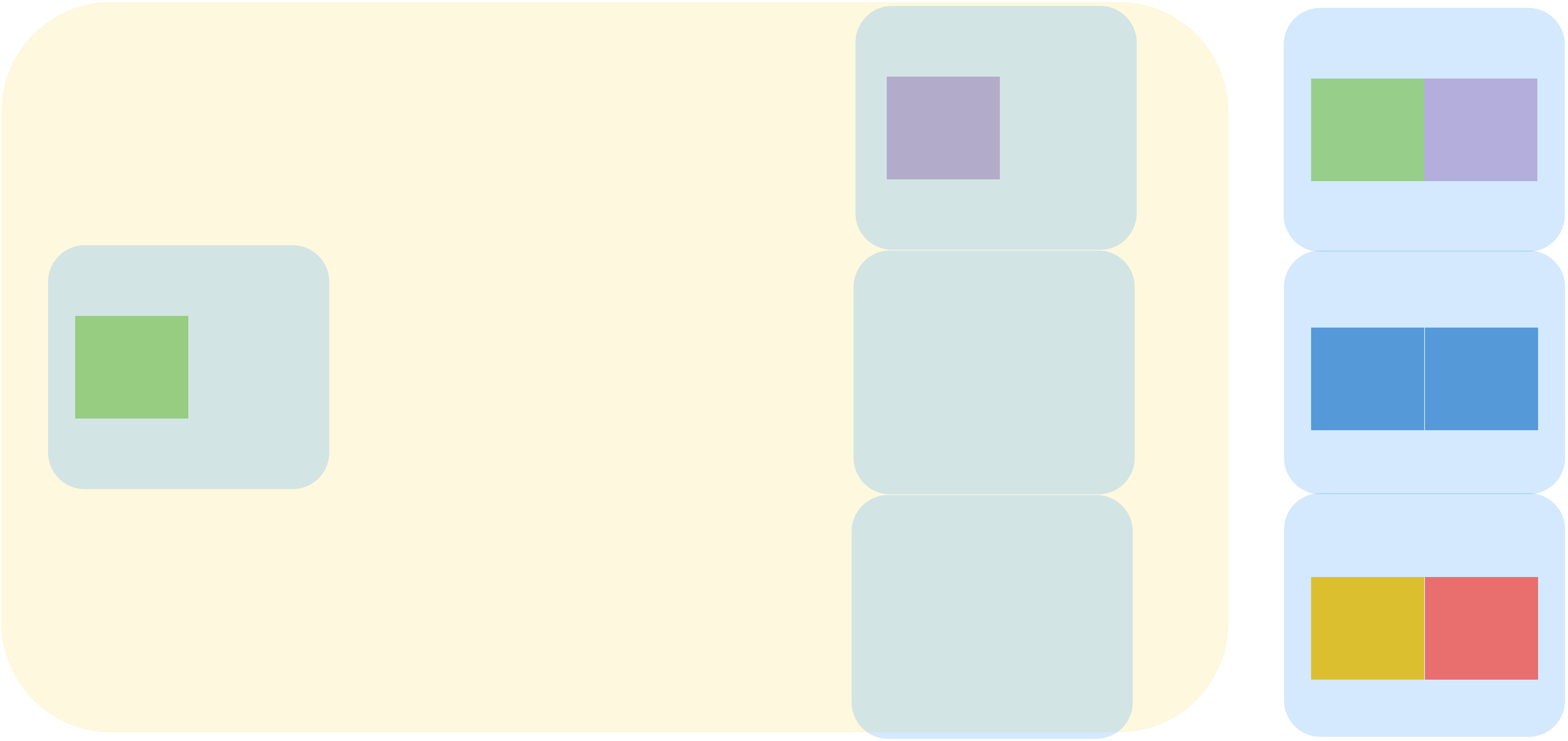
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3

Pass 1: Divide

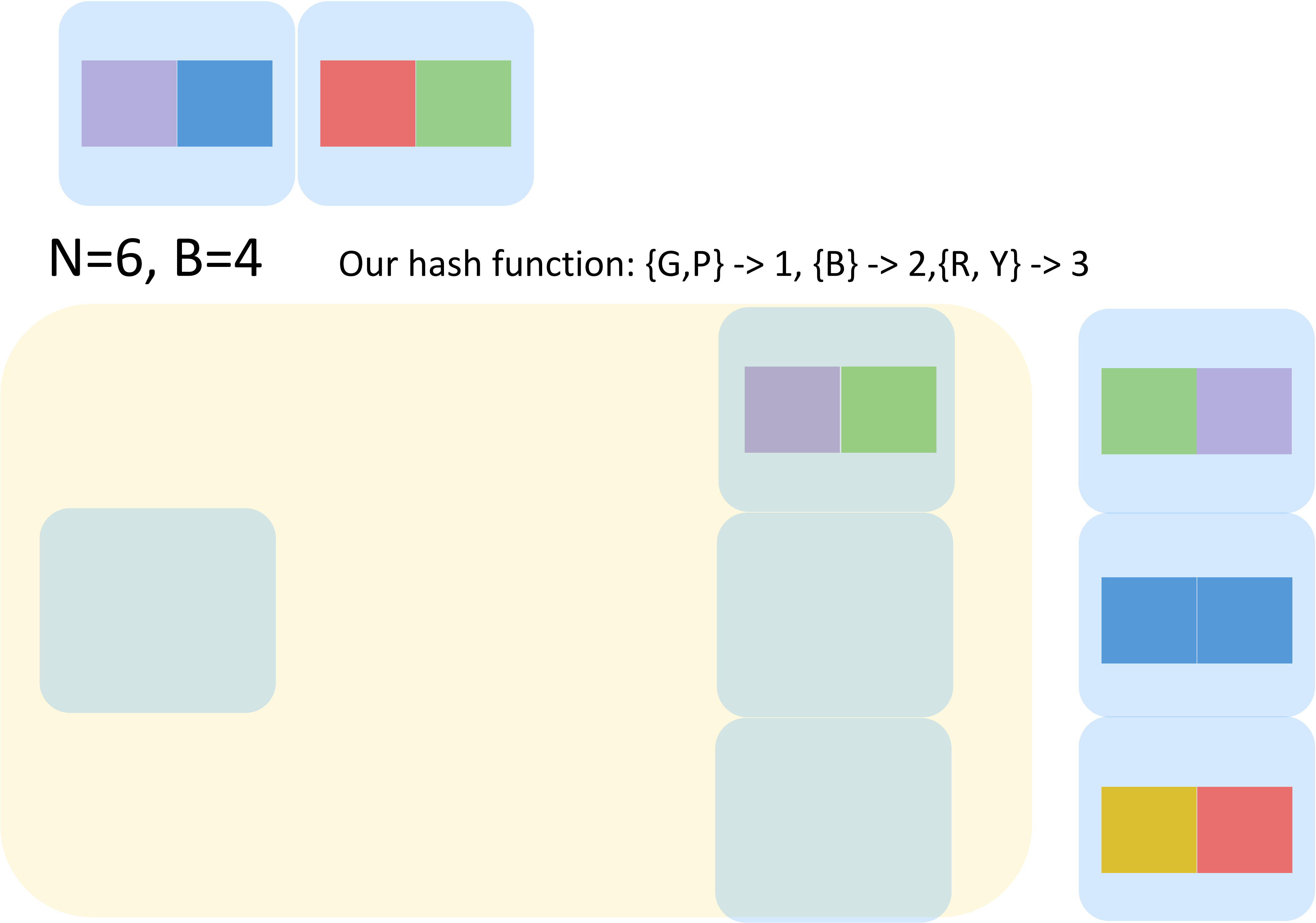


N=6, B=4

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



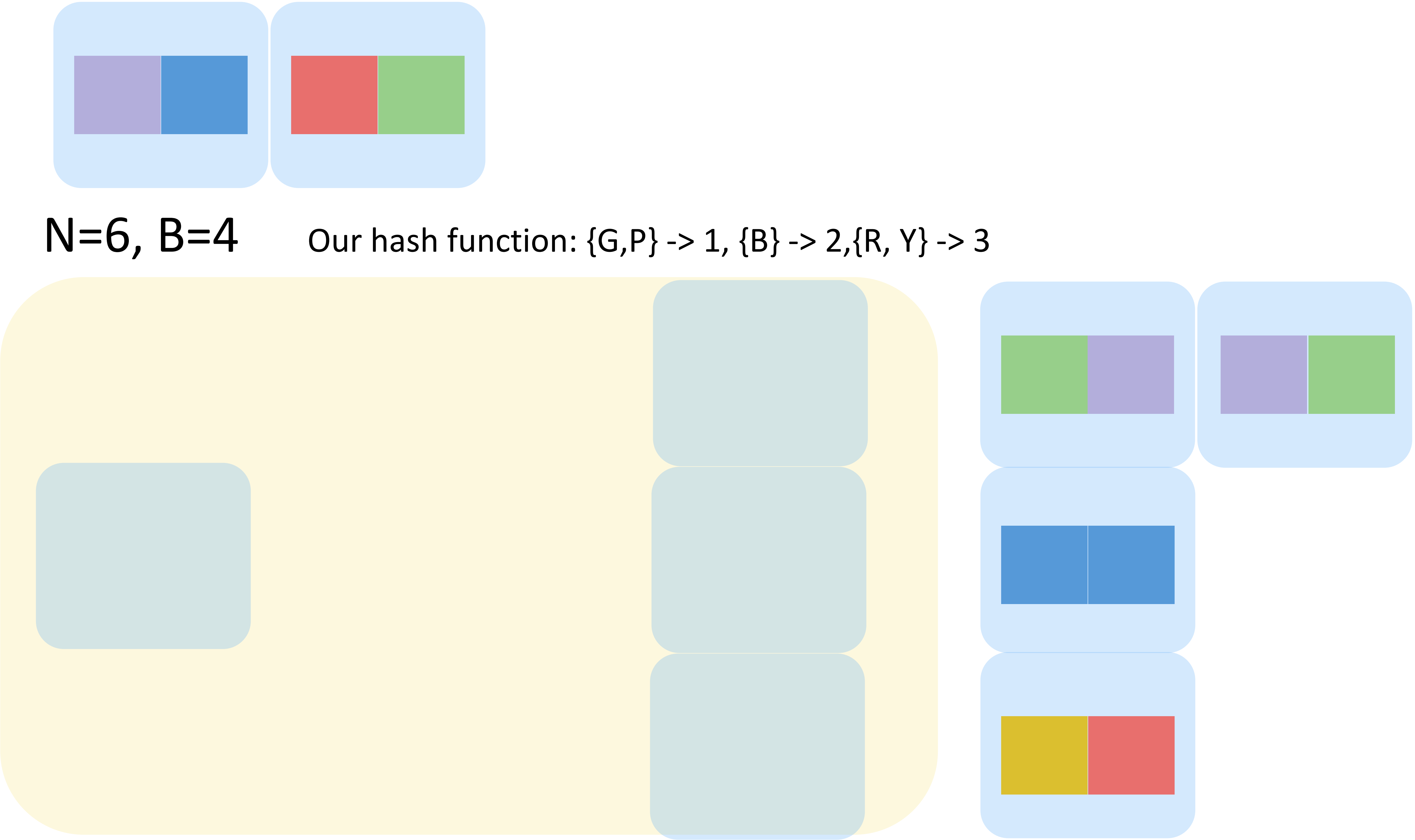
Pass 1: Divide



N=6, B=4

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

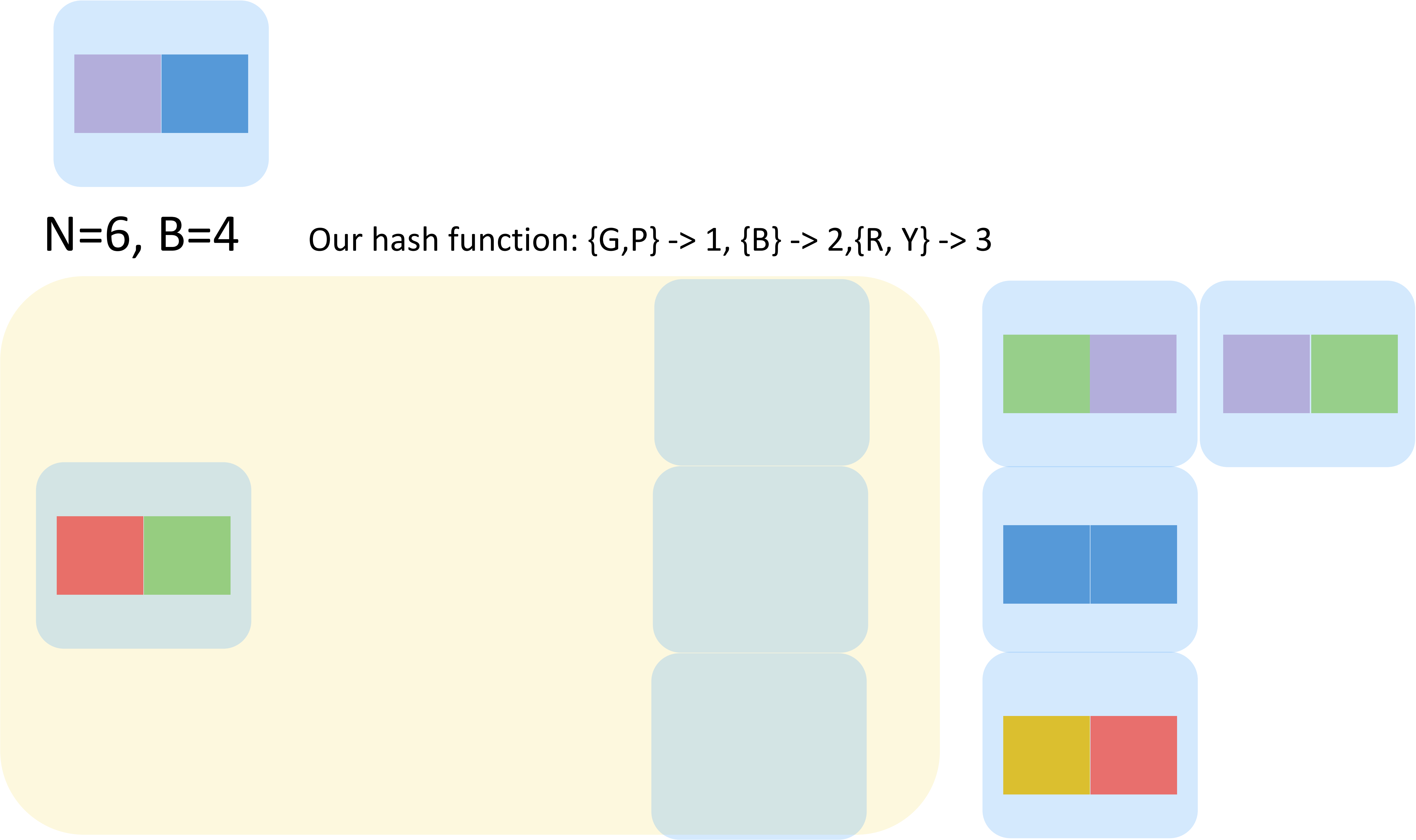
Pass 1: Divide



N=6, B=4

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

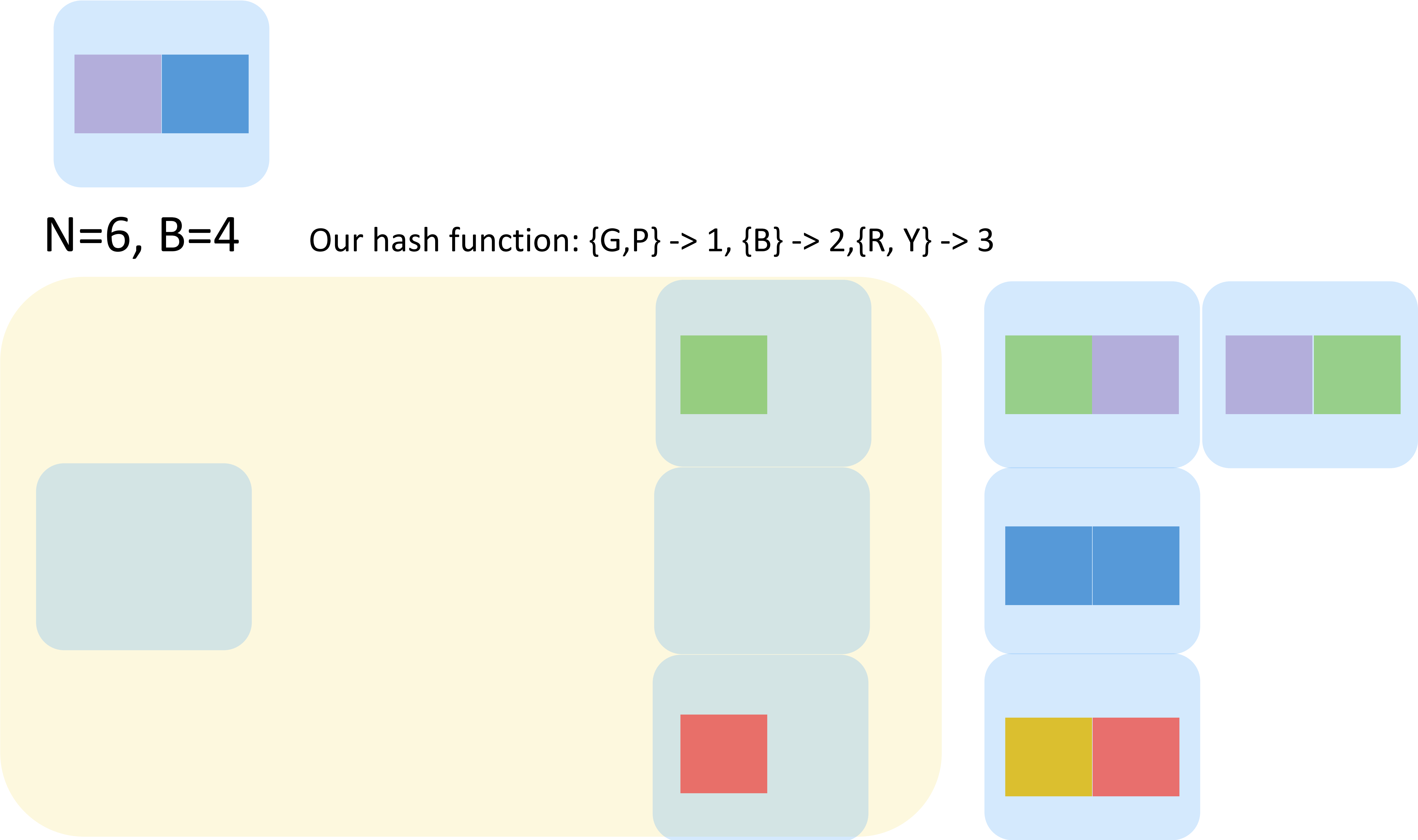
Pass 1: Divide



N=6, B=4

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

Pass 1: Divide



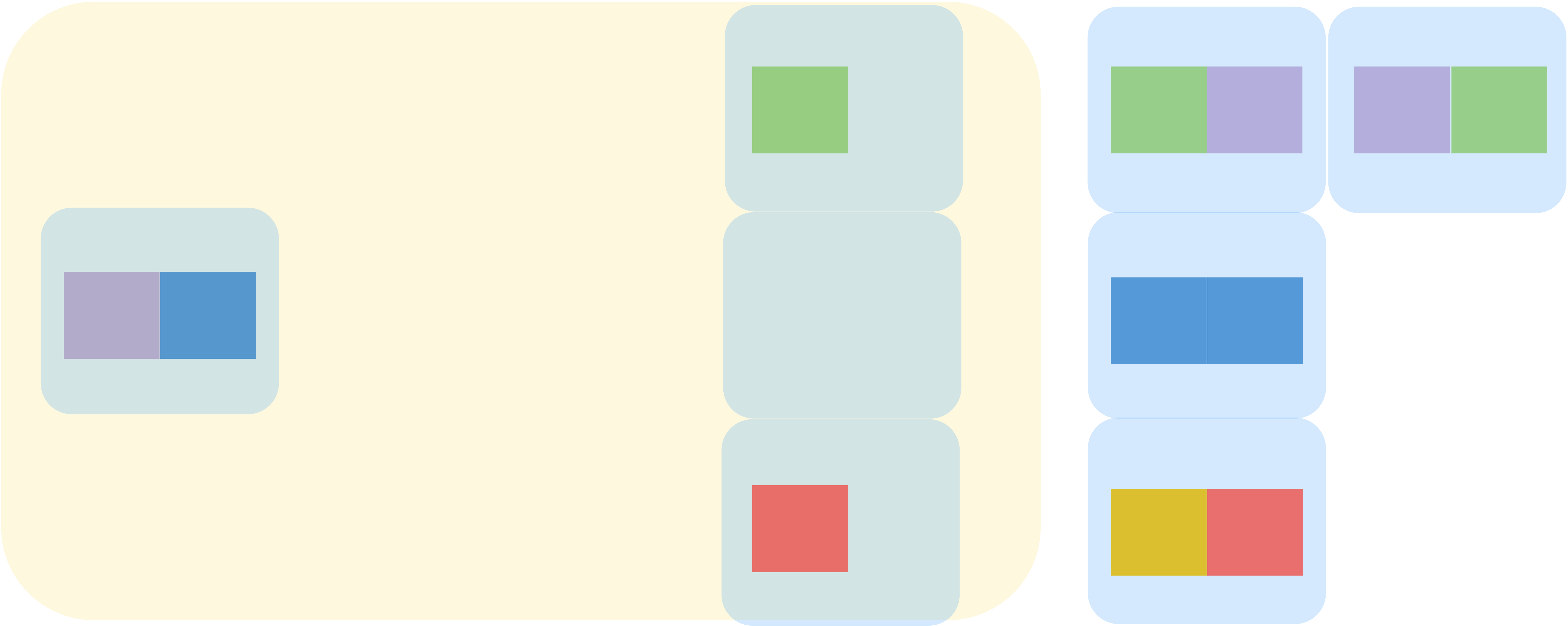
N=6, B=4

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

Pass 1: Divide

N=6, B=4

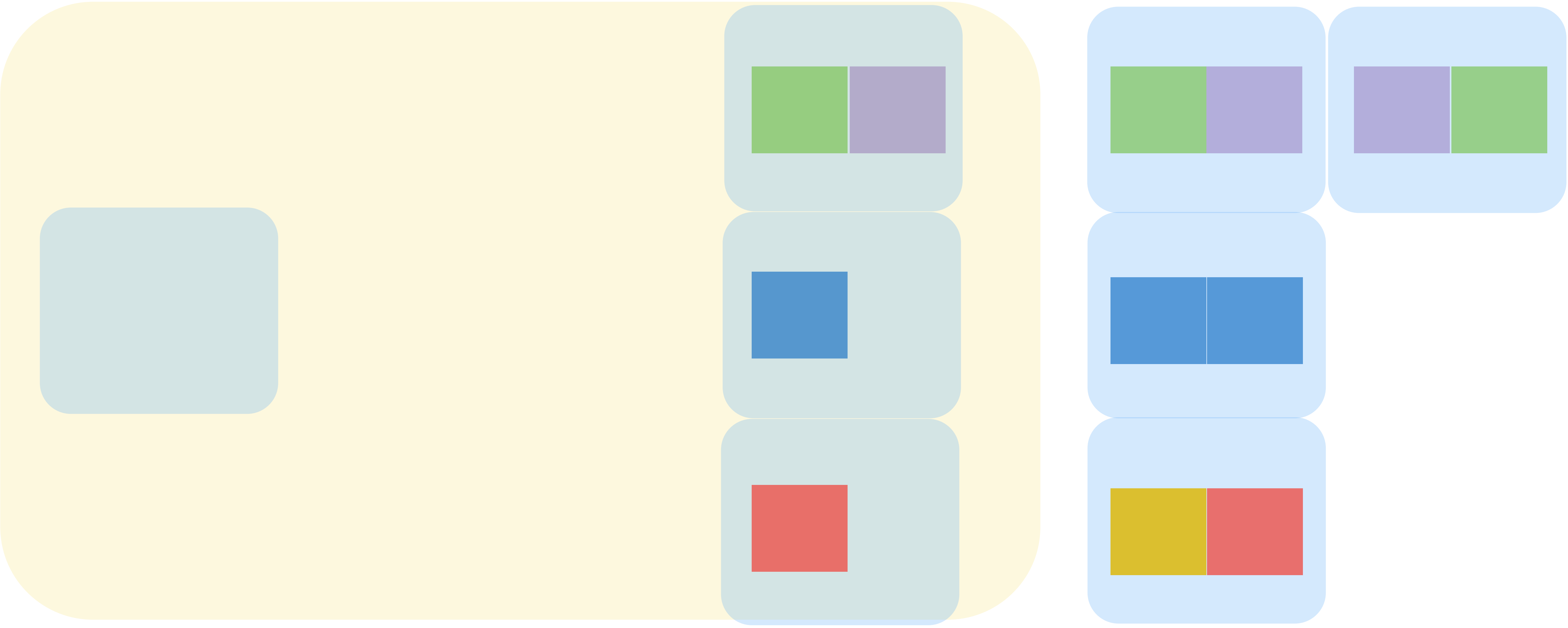
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



Pass 1: Divide

N=6, B=4

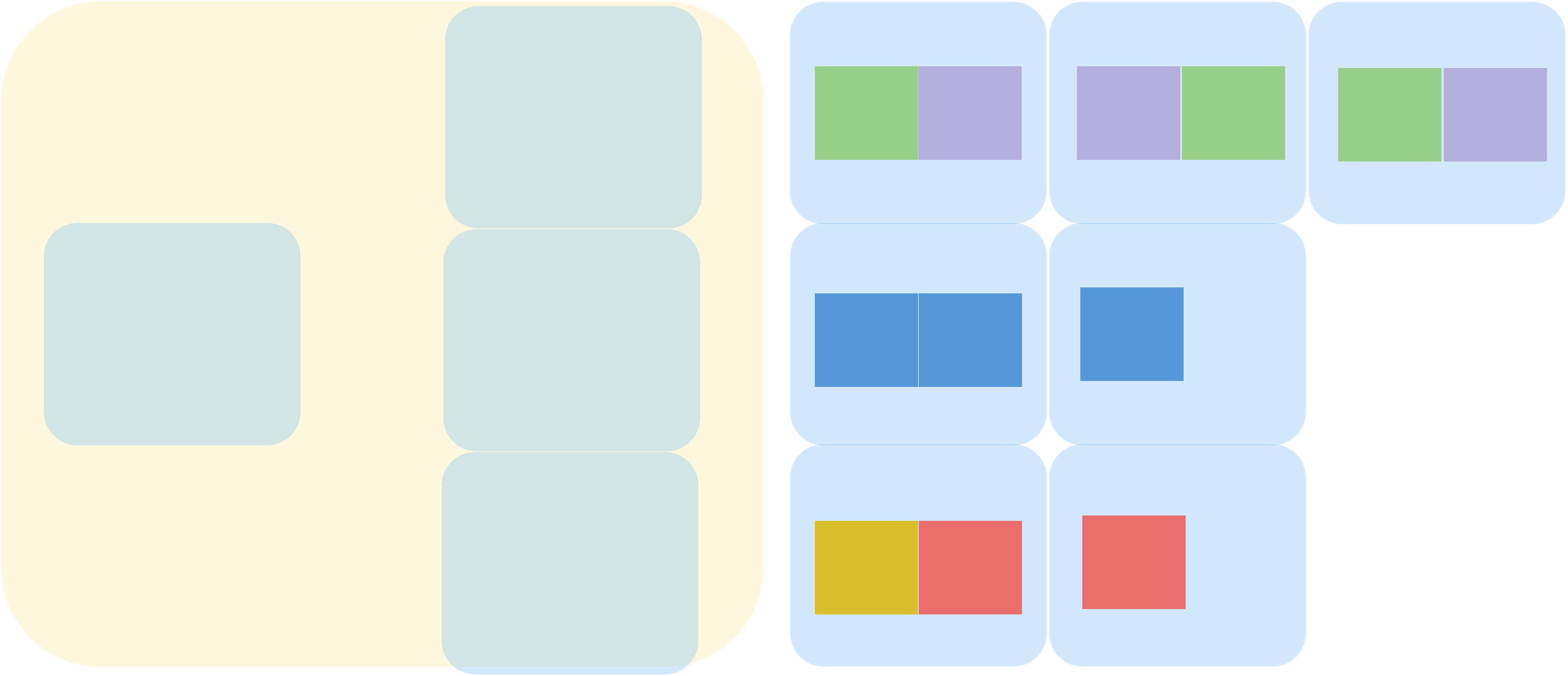
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



Pass 1: Divide

N=6, B=4

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



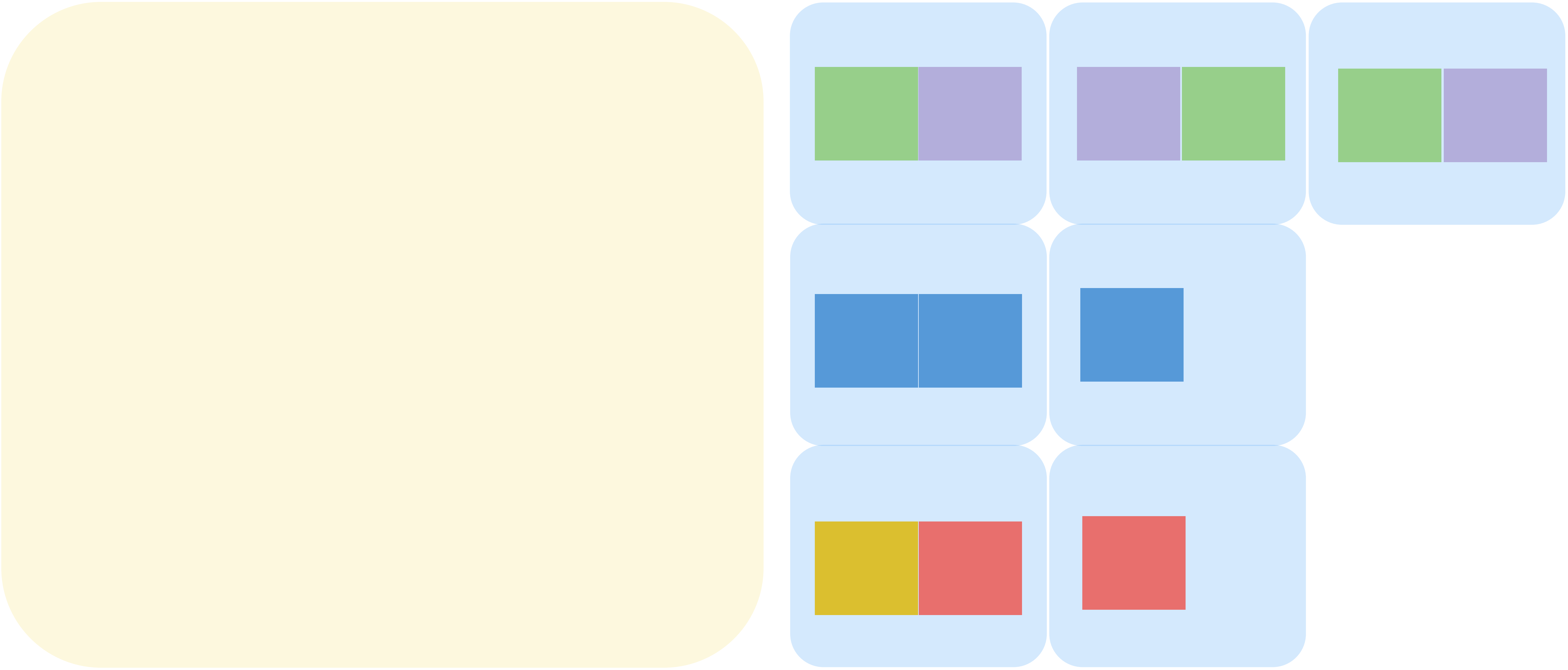
Pass 2: Conquer

- Rehash each partition.
- For a partition to fit in memory, it can only have B pages.
- If a partition is too large… repartition!
    - 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.

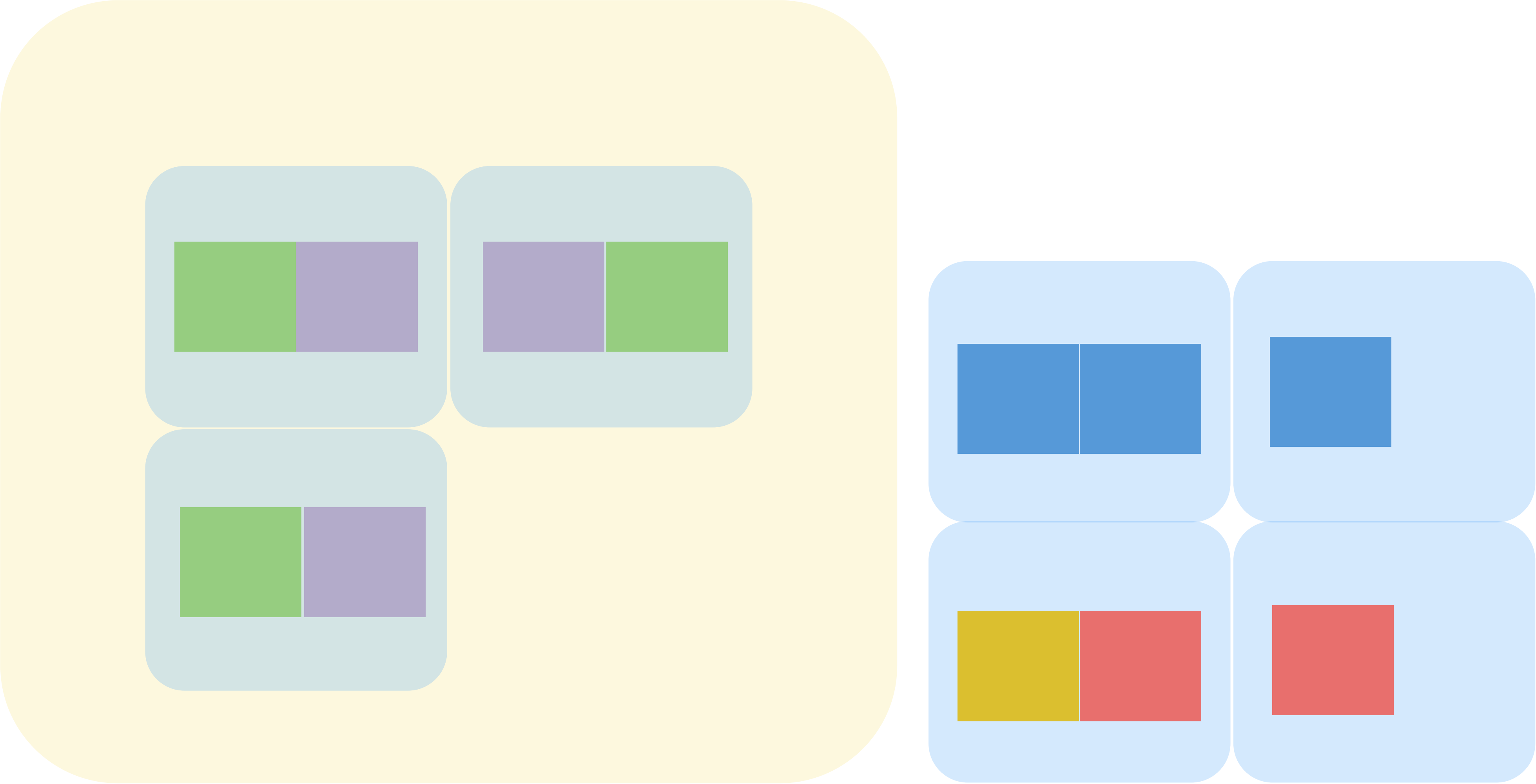
N=6, B=4



Pass 2: Conquer

Create in-memory table for each partition.

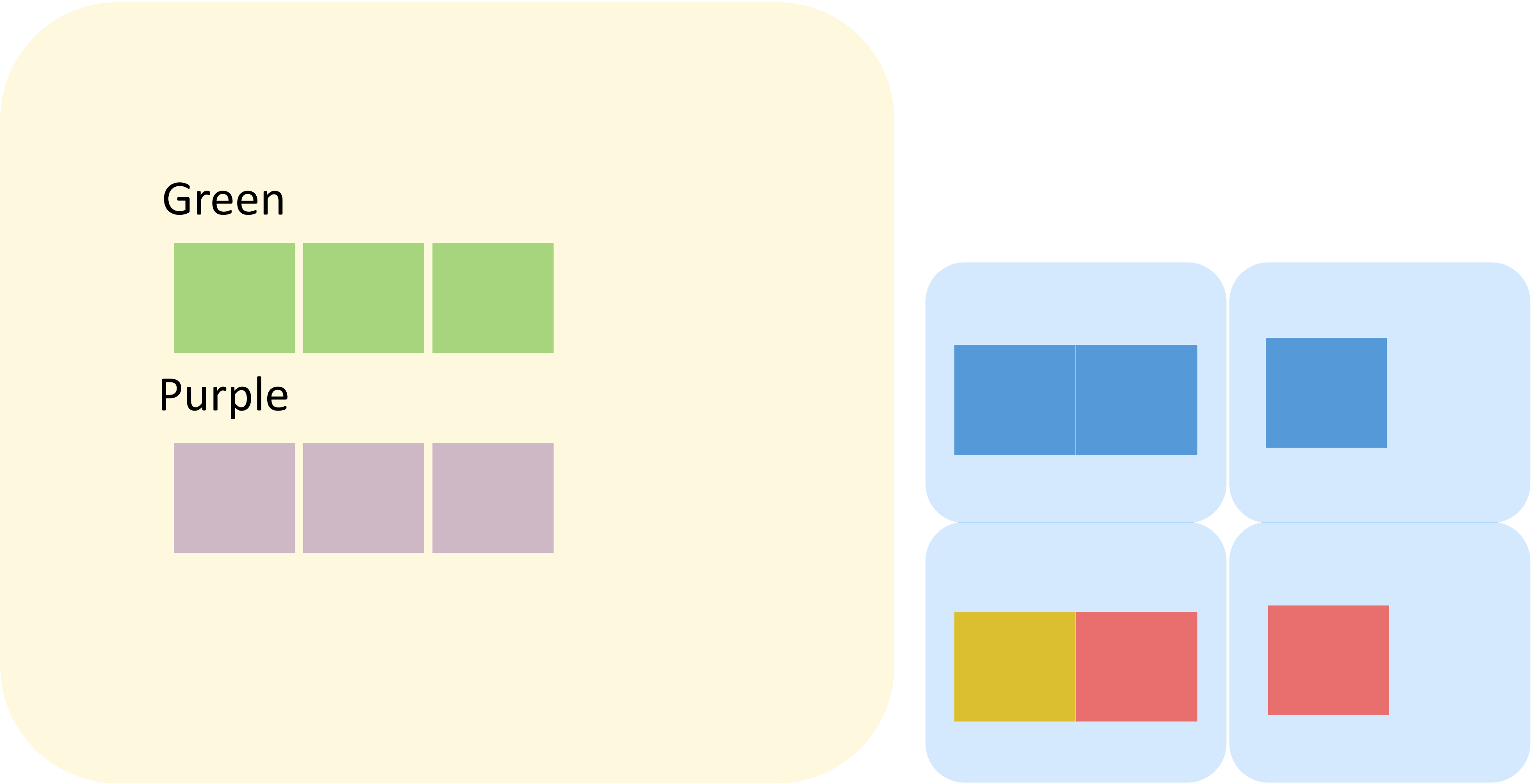
N=6, B=4



Pass 2: Conquer

Create in-memory table for each partition.

N=6, B=4



SQL Queries

```
SELECT [DISTINCT] <column list>
FROM <table1>
WHERE <predicate>
GROUP BY <column list>
HAVING <predicate>
ORDER BY <column list> [DESC/ASC]
LIMIT <amount>
```

Also Review…

- Nested Queries
  - VIEWs, WITH
- UNION/INTERSECT
- Set Comparison Operators
  - IN, EXISTS, ANY, ALL
- Primary Keys
  - And Foreign Keys, Candidate Keys, etc…

Worksheet 1, 2, 3, 4

1. Five songs that spent the most time in the top 40:

Worksheet 1, 2, 3, 4

1. Five songs that spent the most time in the top 40:

```
SELECT song_name
FROM   Songs
ORDER BY weeks_in_top_40 DESC
LIMIT 5;
```

Worksheet 1, 2, 3, 4

2. Name and first year active of every artist whose name starts with ‘B’:

Worksheet 1, 2, 3, 4

2. Name and first year active of every artist whose name starts with ‘B’:

```
SELECT artist_name, first_year_active
FROM   Artists
WHERE artist_name LIKE ‘B%’;
```

Worksheet 1, 2, 3, 4

3. Total number of ‘Techno’ albums released each year:

Worksheet 1, 2, 3, 4

3. Total number of ‘Techno’ albums released each year:

```
SELECT      year_released, COUNT(*)
FROM        Albums
WHERE       genre = ‘Techno’
GROUP BY year_released;
```

Worksheet 1, 2, 3, 4

4. Number of albums per genre, ignoring genres with fewer than 10 albums:

Worksheet 1, 2, 3, 4

4. Number of albums per genre, ignoring genres with fewer than 10 albums:

```
SELECT     genre, COUNT(*)
FROM       Albums
GROUP BY year_released
HAVING     COUNT(*) >= 10;
```