CS186 Discussion #2

(External sorting & hashing, Single-table SQL)

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

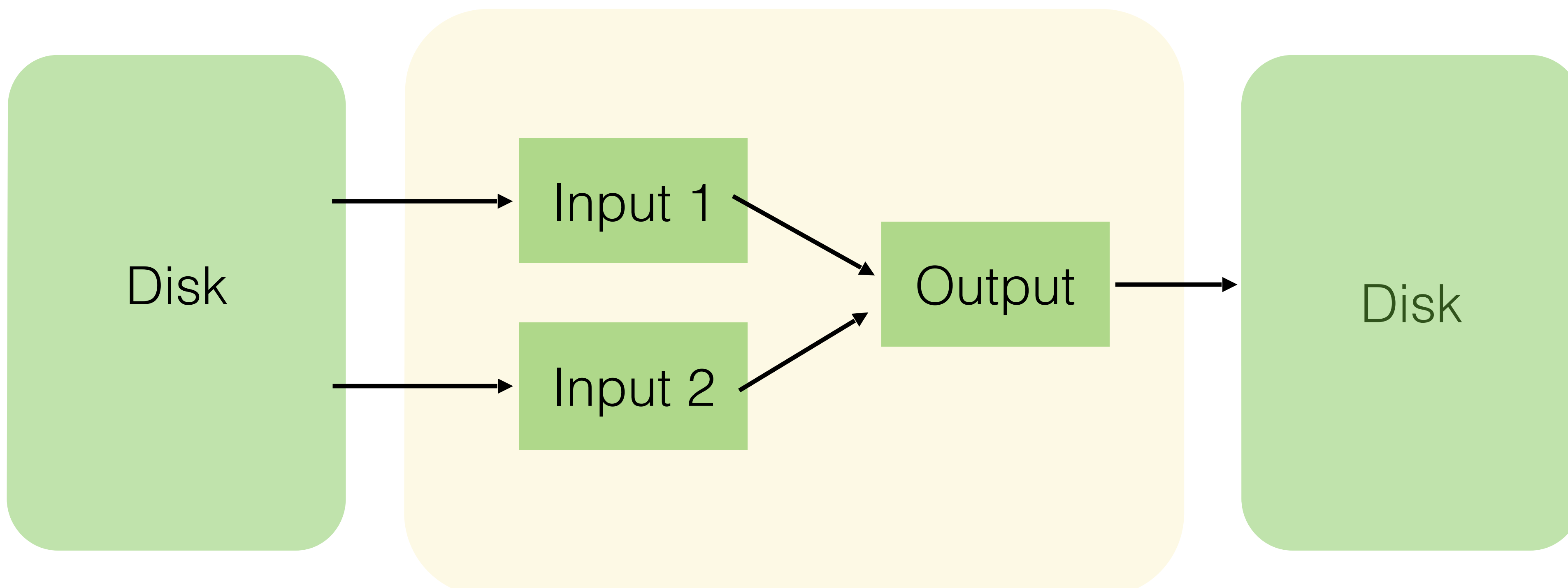
- A sorted subset of a table
- Size is denoted by how many pages it spans

(name = Bob; sid = 1)	(name = Sam; sid = 2)
(name = Joe; sid = 2)	(name = Jen; sid = 4)
(name = Ann; sid = 3)	(name = Dan; sid = 5)
(name = Jill; sid = 6)	(name = Ned; sid = 6)
(name = Mia; sid = 9)	(name = Ed; sid = 10)
(name = Ted; sid = 10)	(name = Lou; sid = 11)
(name = Bill; sid = 12)	(name = Al; sid = 14)
(name = Van; sid = 13)	(name = Kev; sid = 15)
(name = Jon; sid = 15)	(name = Sue; sid = 20)

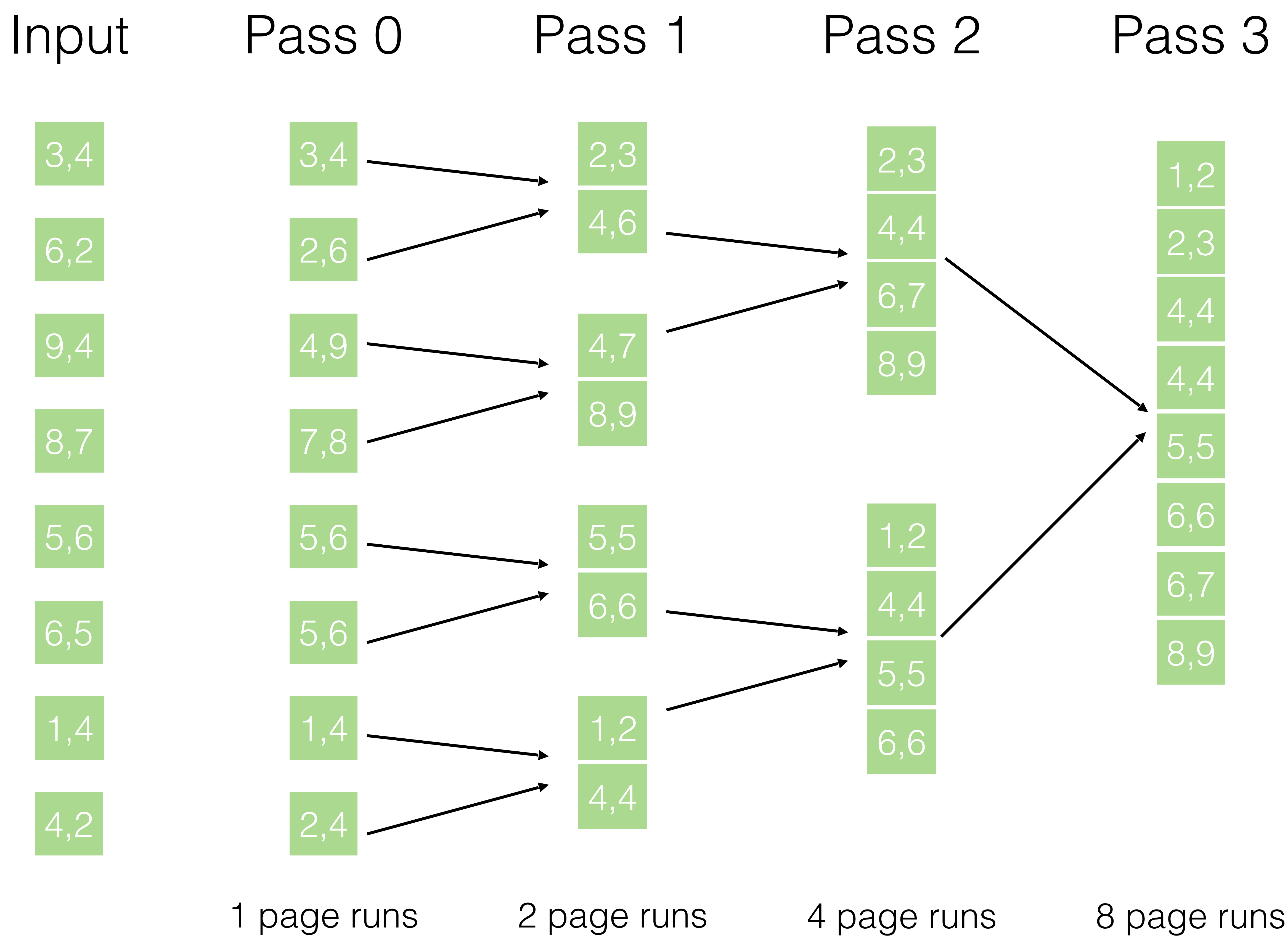
Pages with tuple size = 3

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

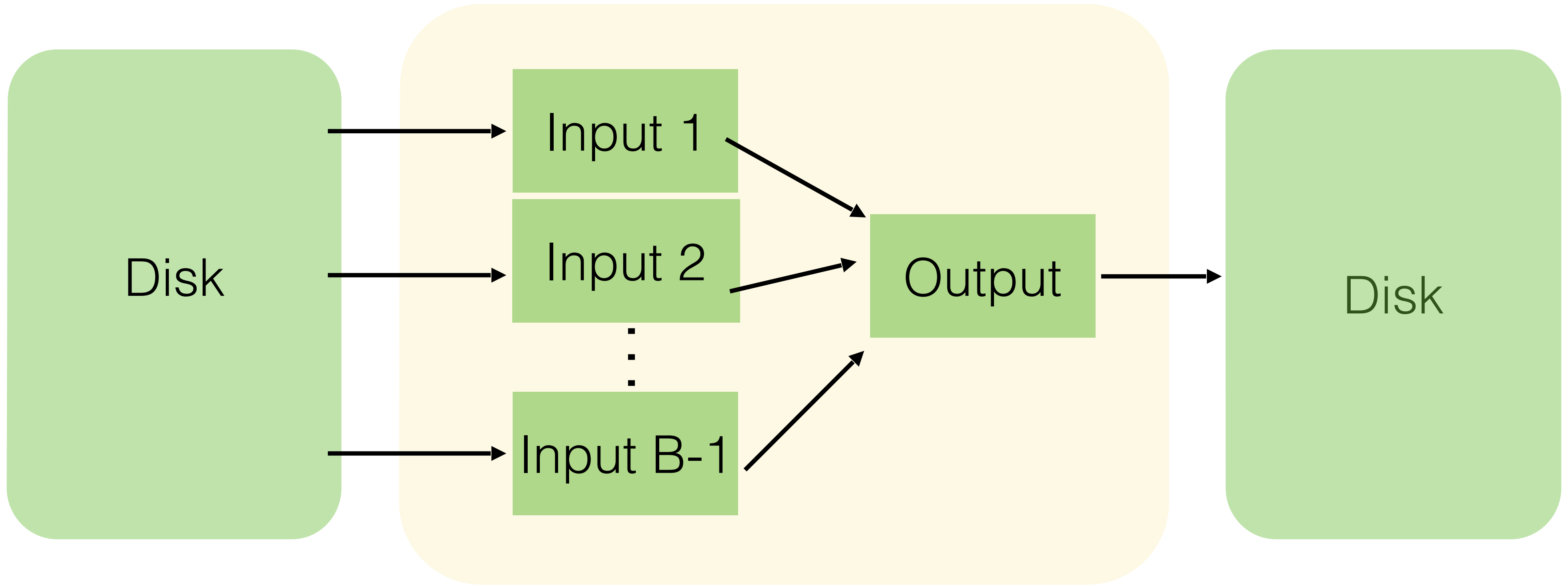
2-Way Merge Sort



Buffer size of 3 pages


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)`

Worksheet #1, 2

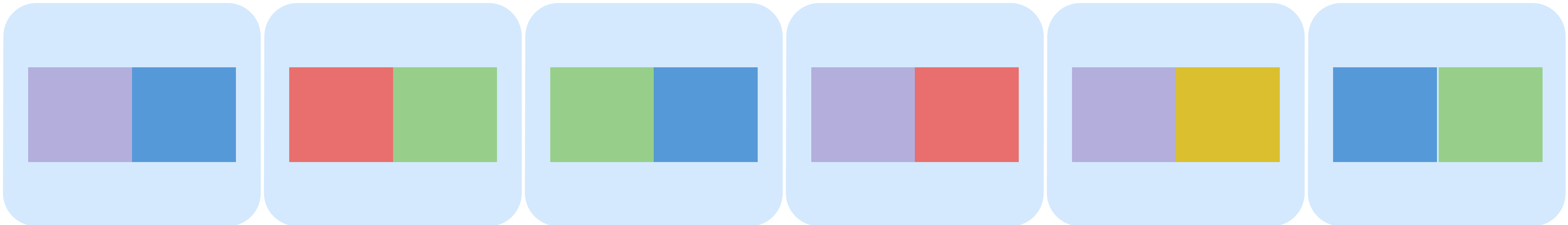
External Hashing

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



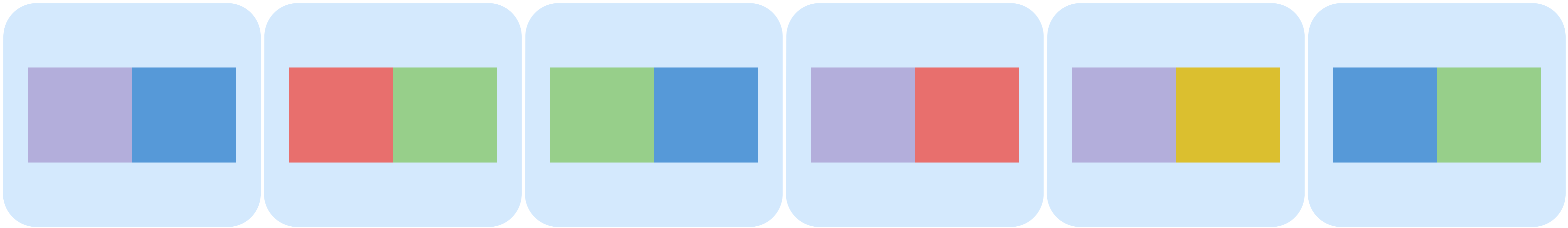
Pass 1: Divide

- 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

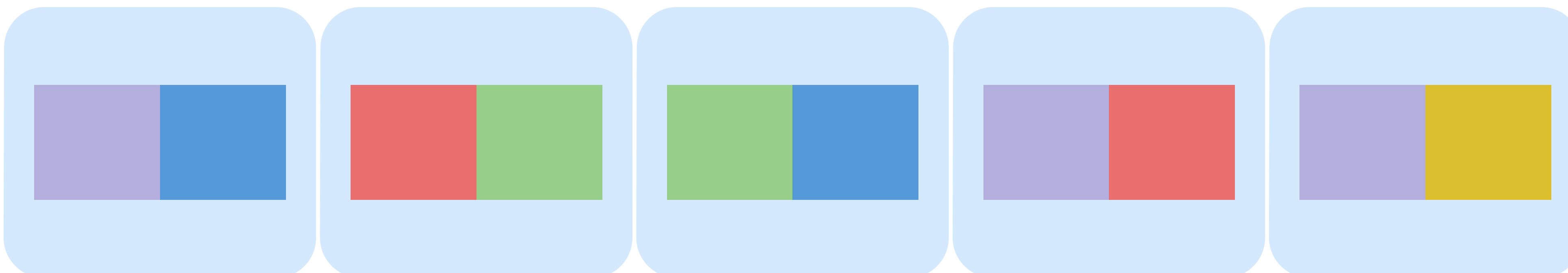
Pass 1: Divide



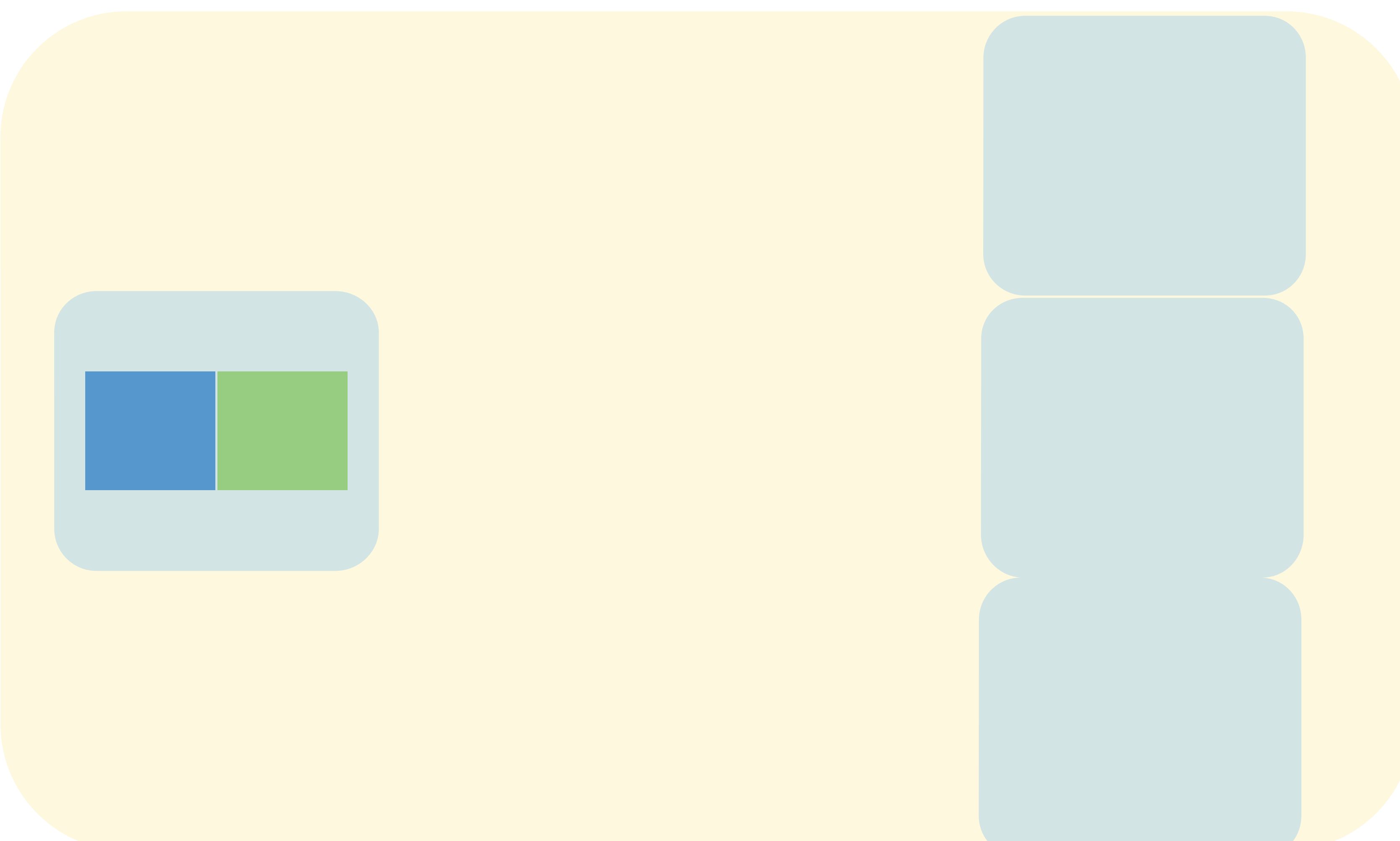
N=6, B=4

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

Pass 1: Divide

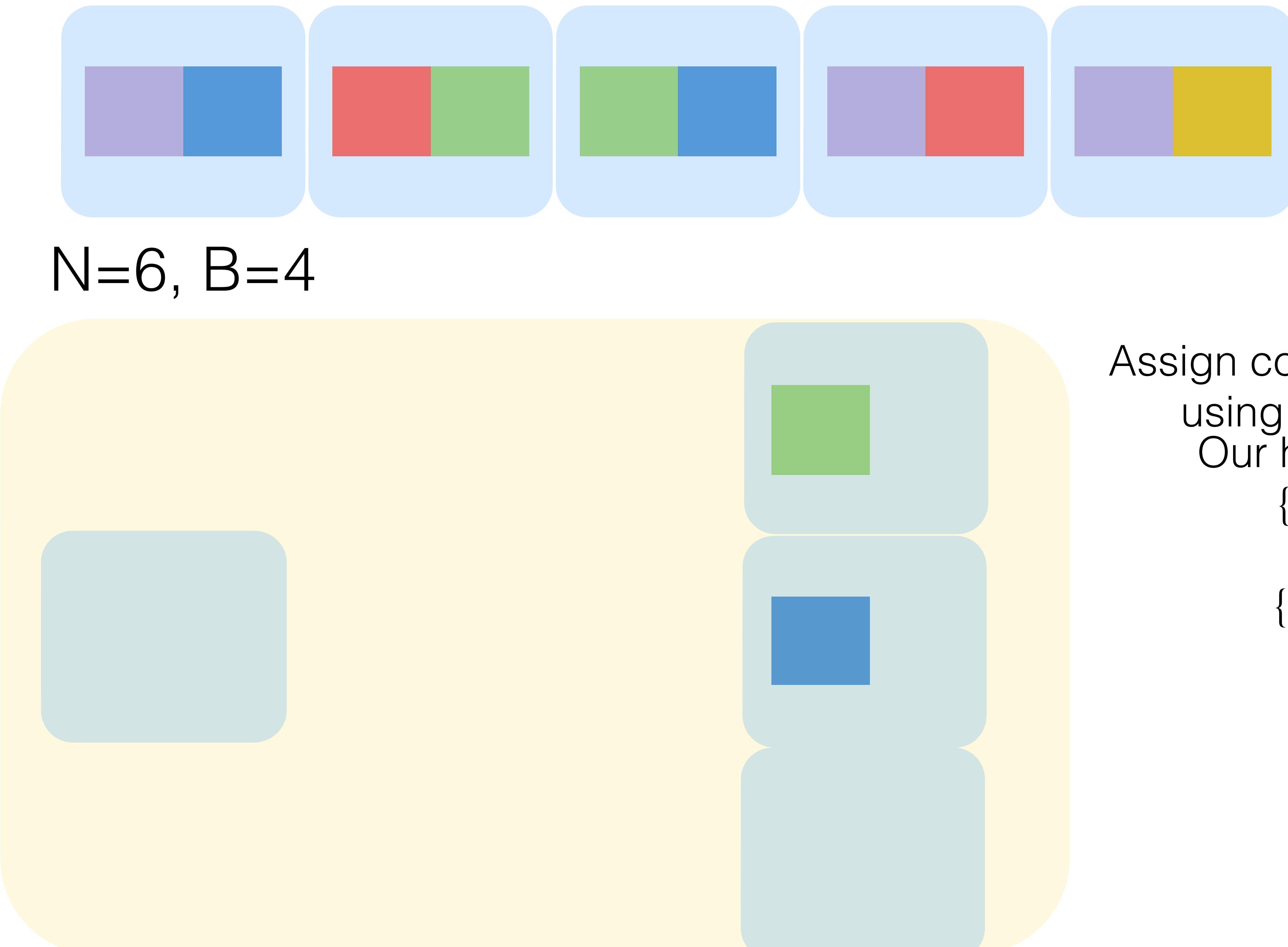


N=6, B=4



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

Pass 1: Divide



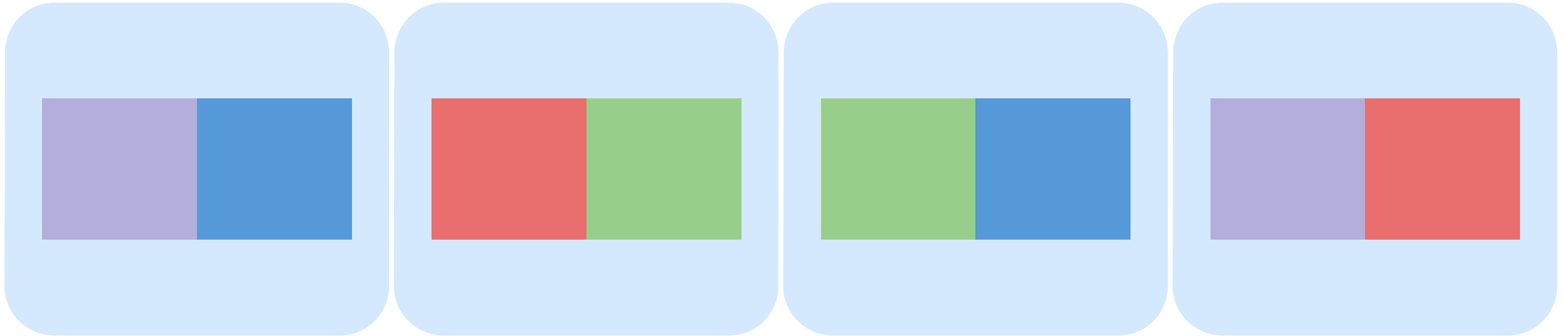
Assign colors to 3 partitions using hash function.
Our hash function:

{G,P} -> 1

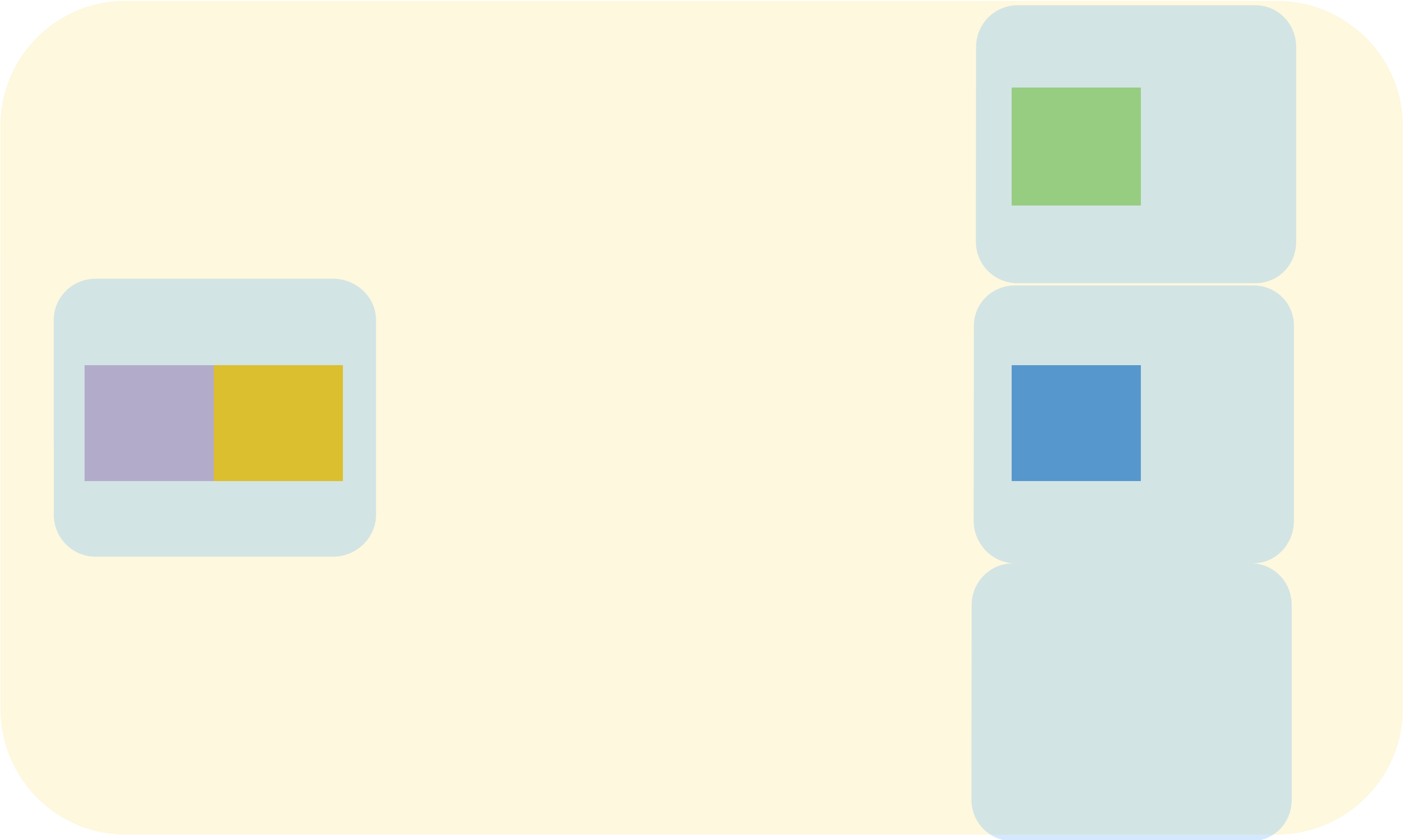
{B} -> 2

{R, Y} -> 3

Pass 1: Divide



N=6, B=4



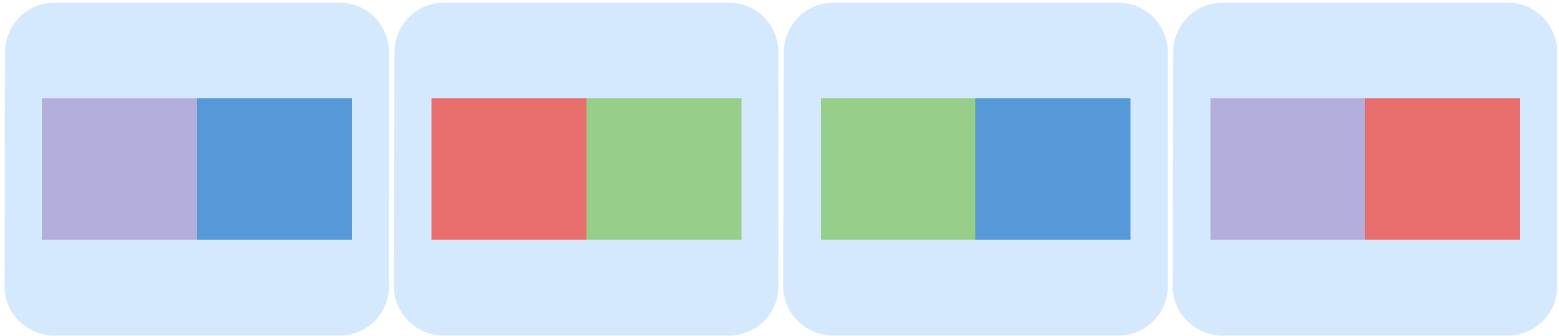
Assign colors to 3 partitions using hash function.
Our hash function:

{G,P} -> 1

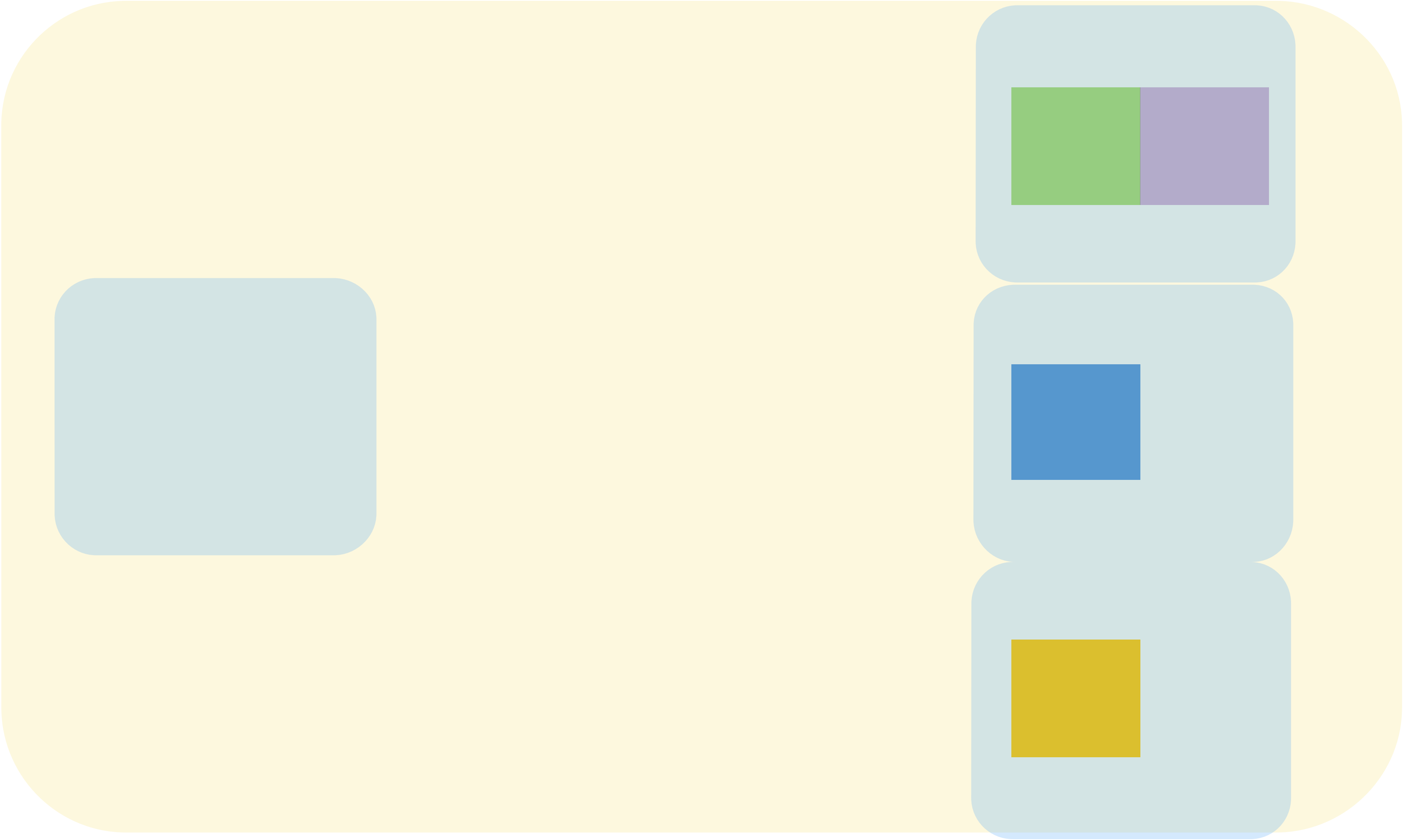
{B} -> 2

{R, Y} -> 3

Pass 1: Divide



N=6, B=4

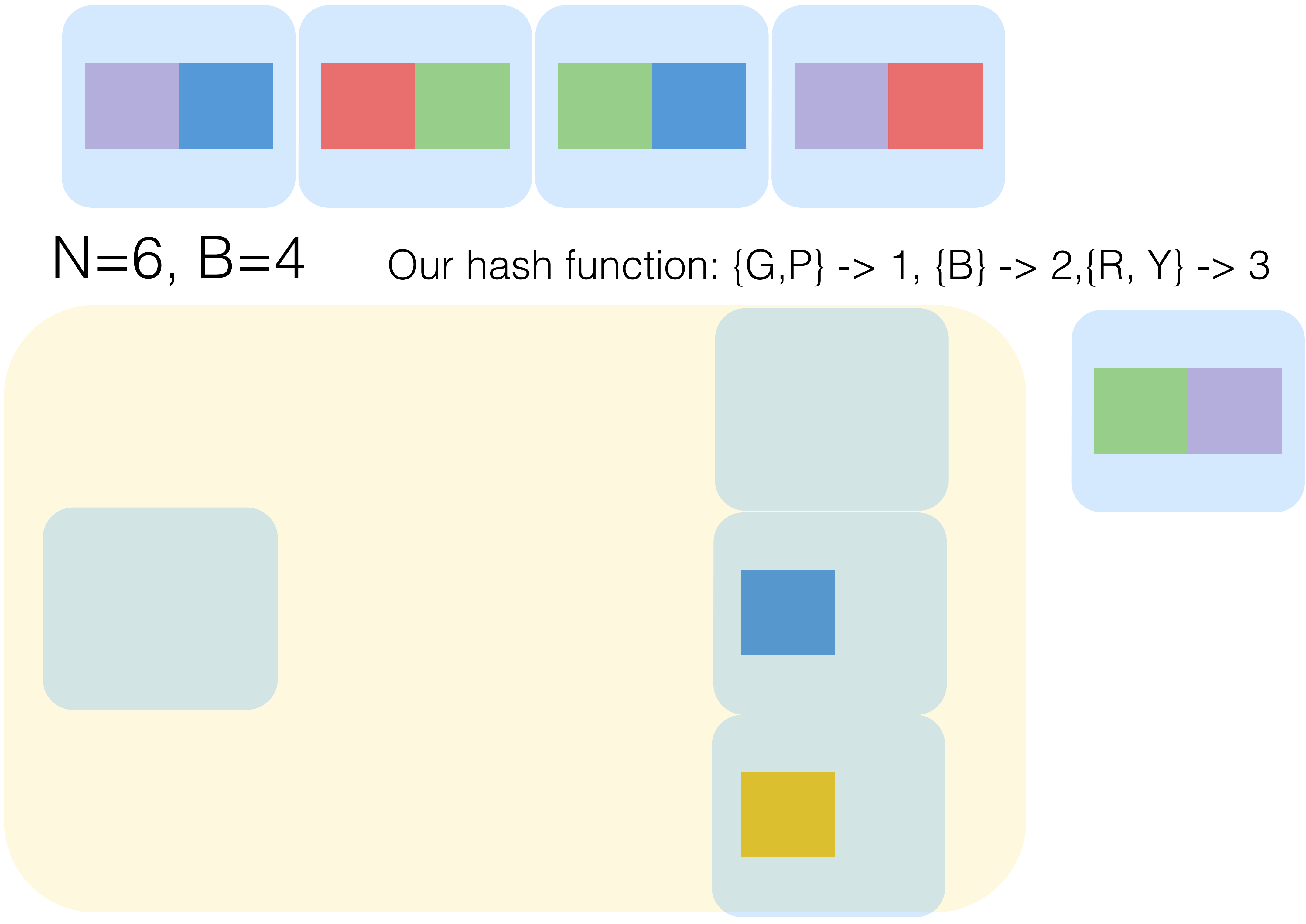


Assign colors to 3 partitions using hash function.
Our hash function:
{G,P} -> 1
{B} -> 2
{R, Y} -> 3

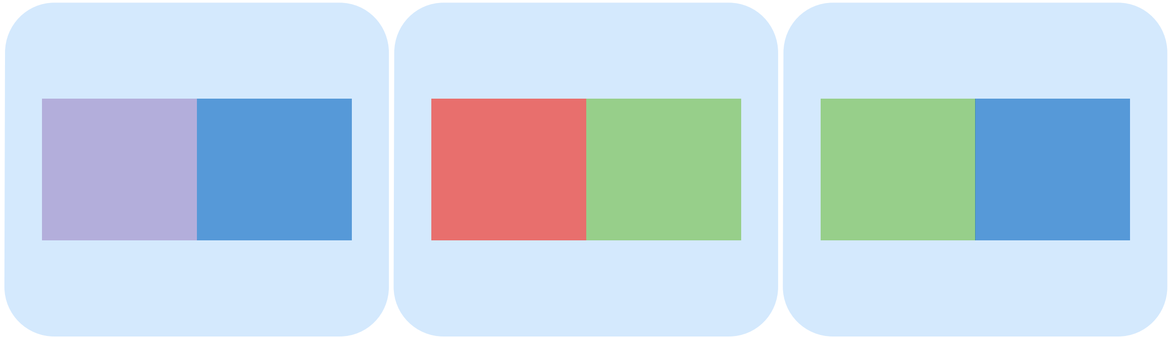
Pass 1: Divide

N=6, B=4

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

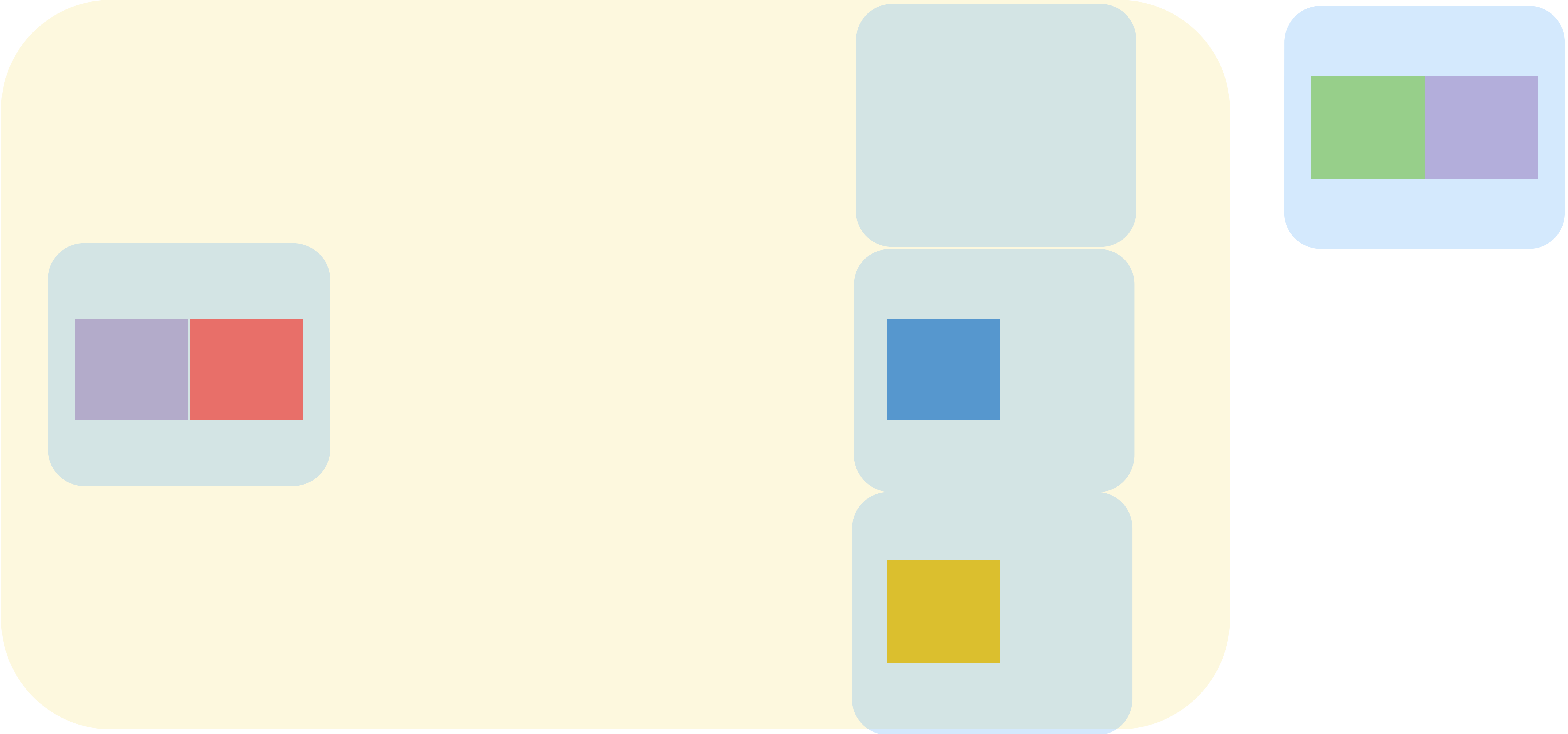


Pass 1: Divide

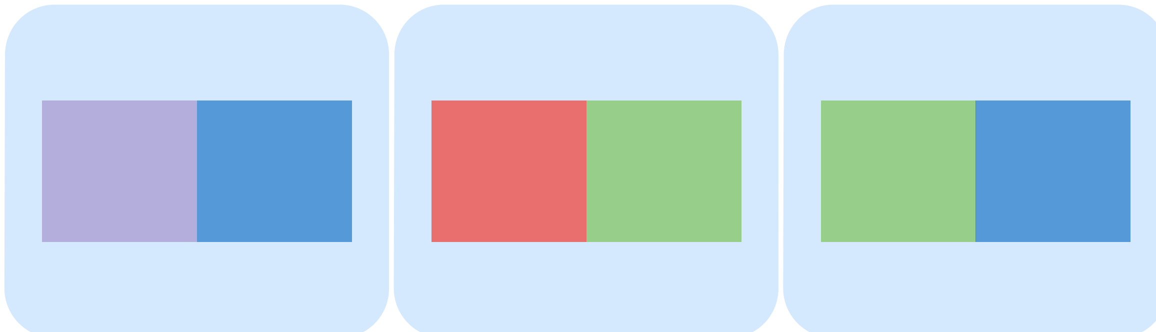


N=6, B=4

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

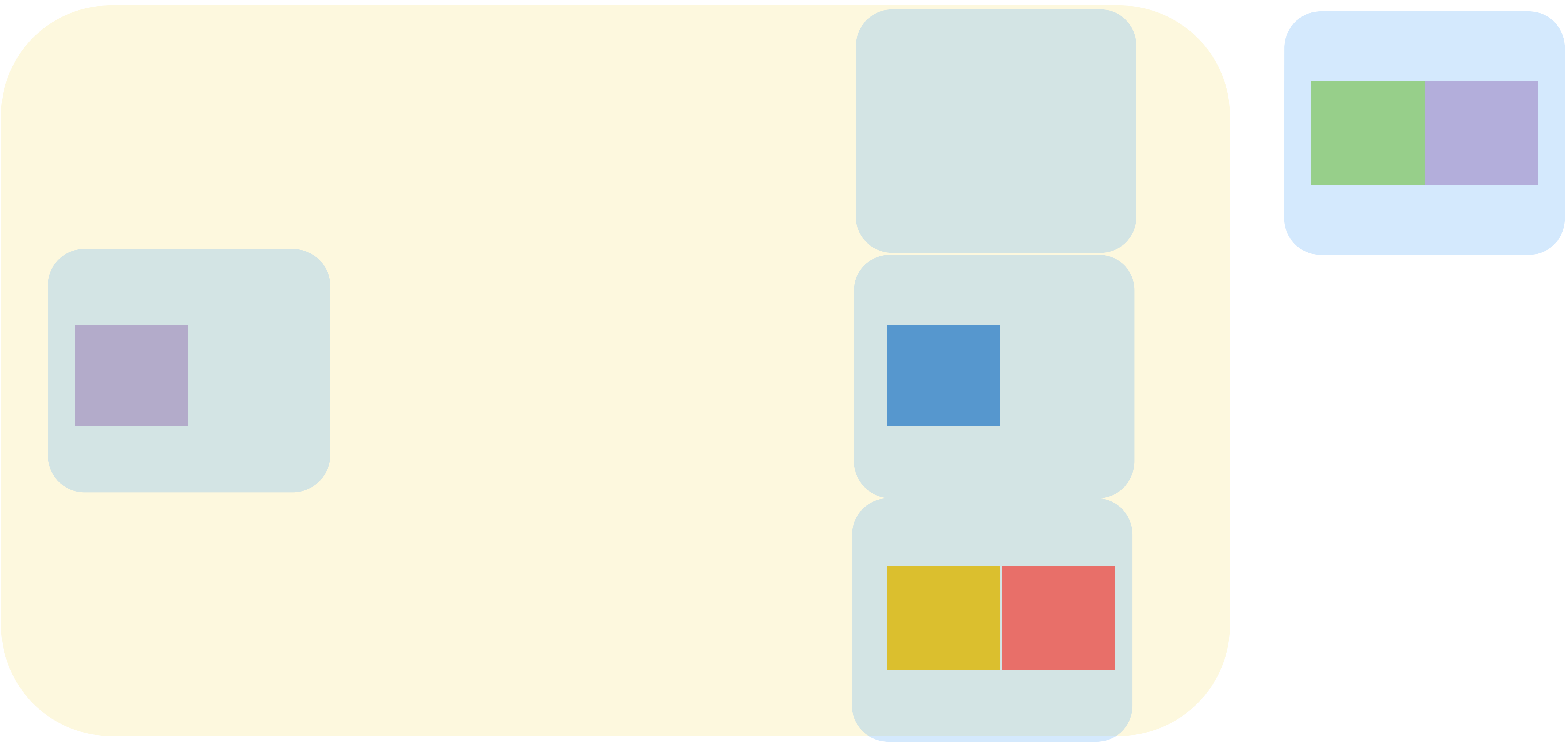


Pass 1: Divide

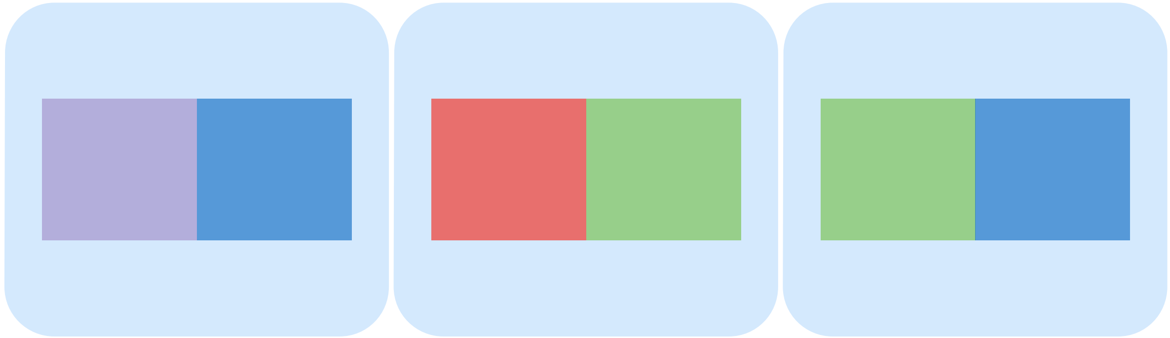


N=6, B=4

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

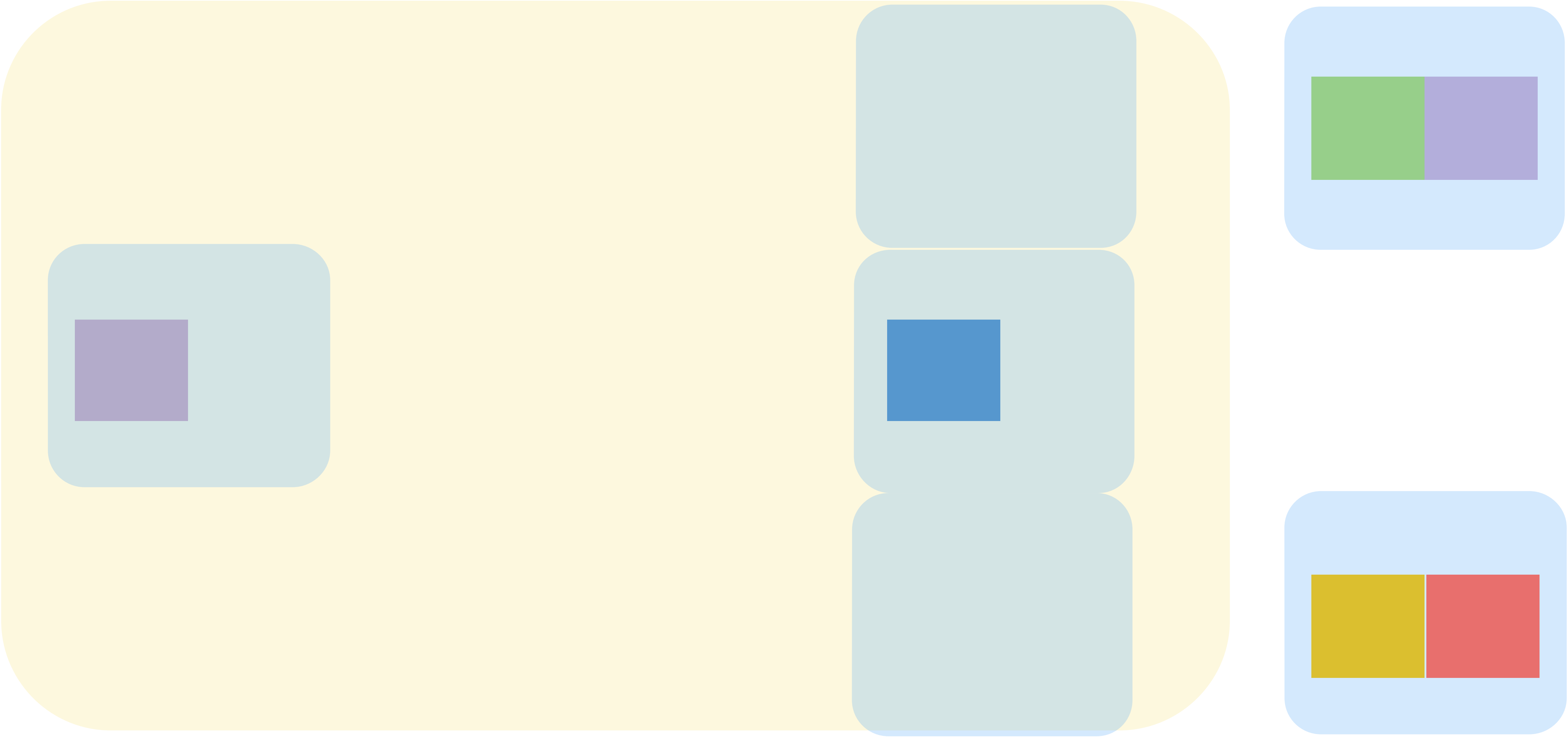


Pass 1: Divide

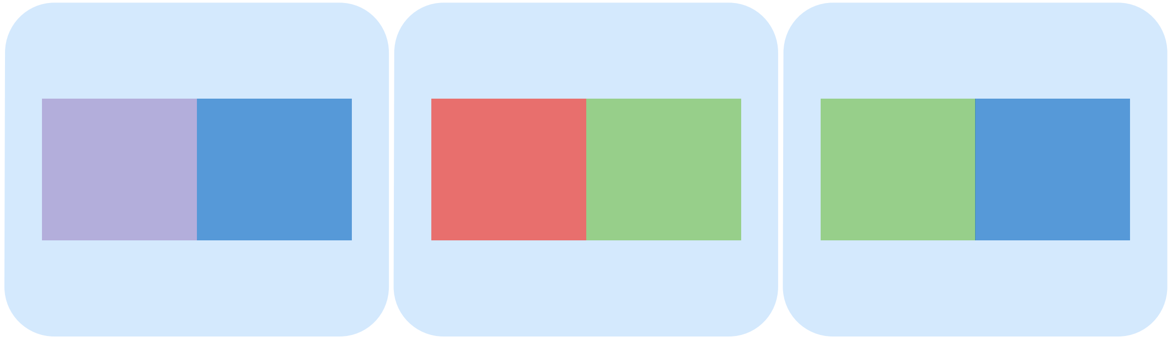


N=6, B=4

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

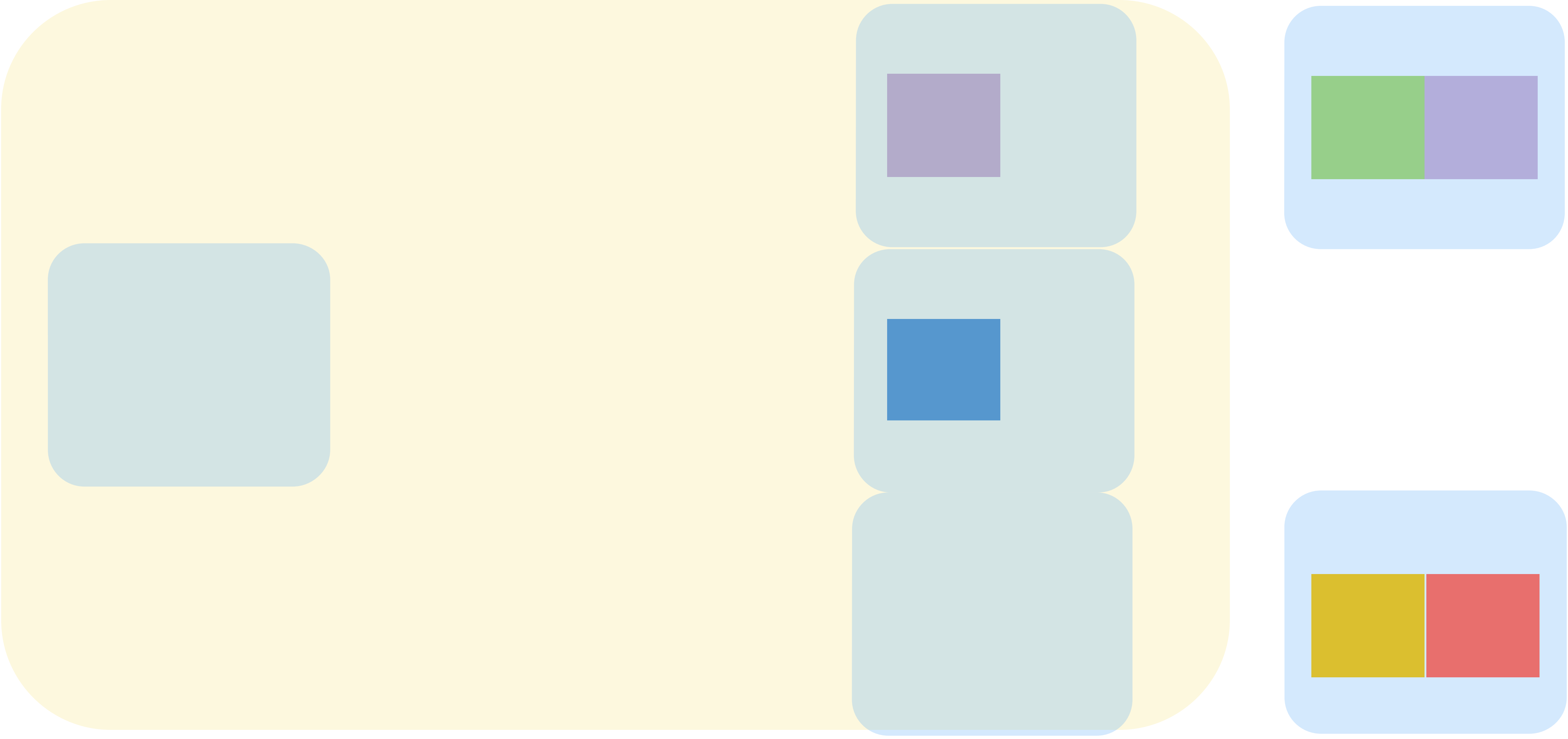


Pass 1: Divide

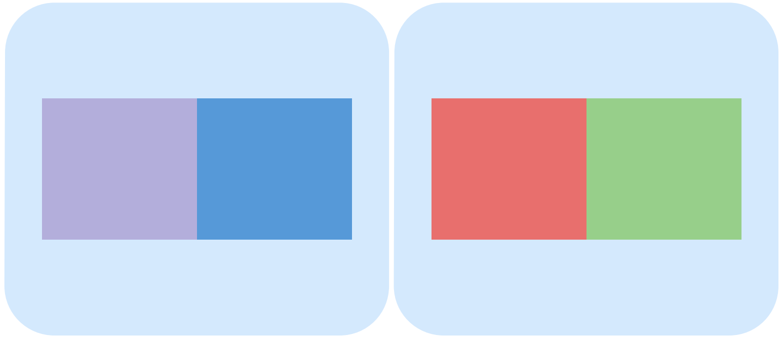


N=6, B=4

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

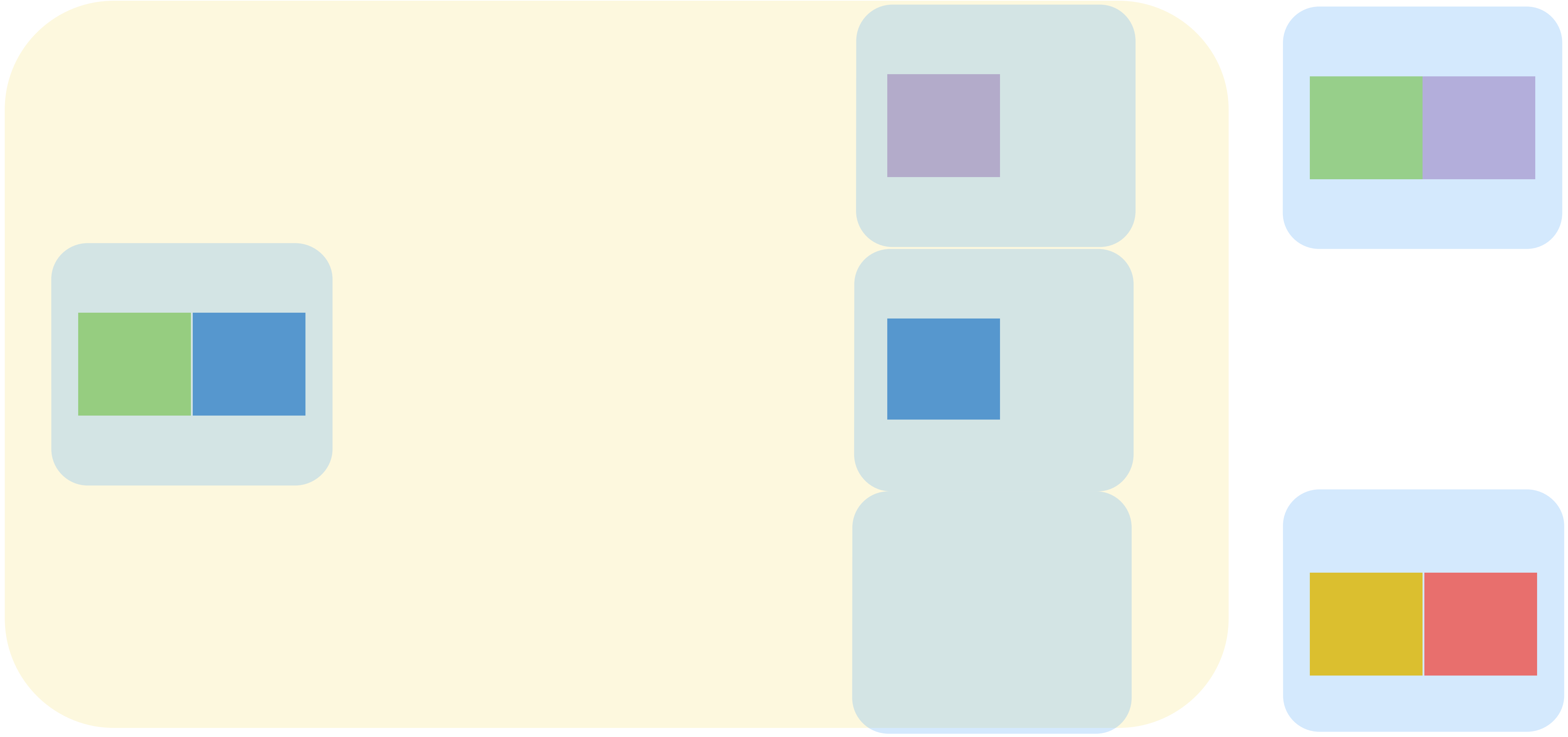


Pass 1: Divide



N=6, B=4

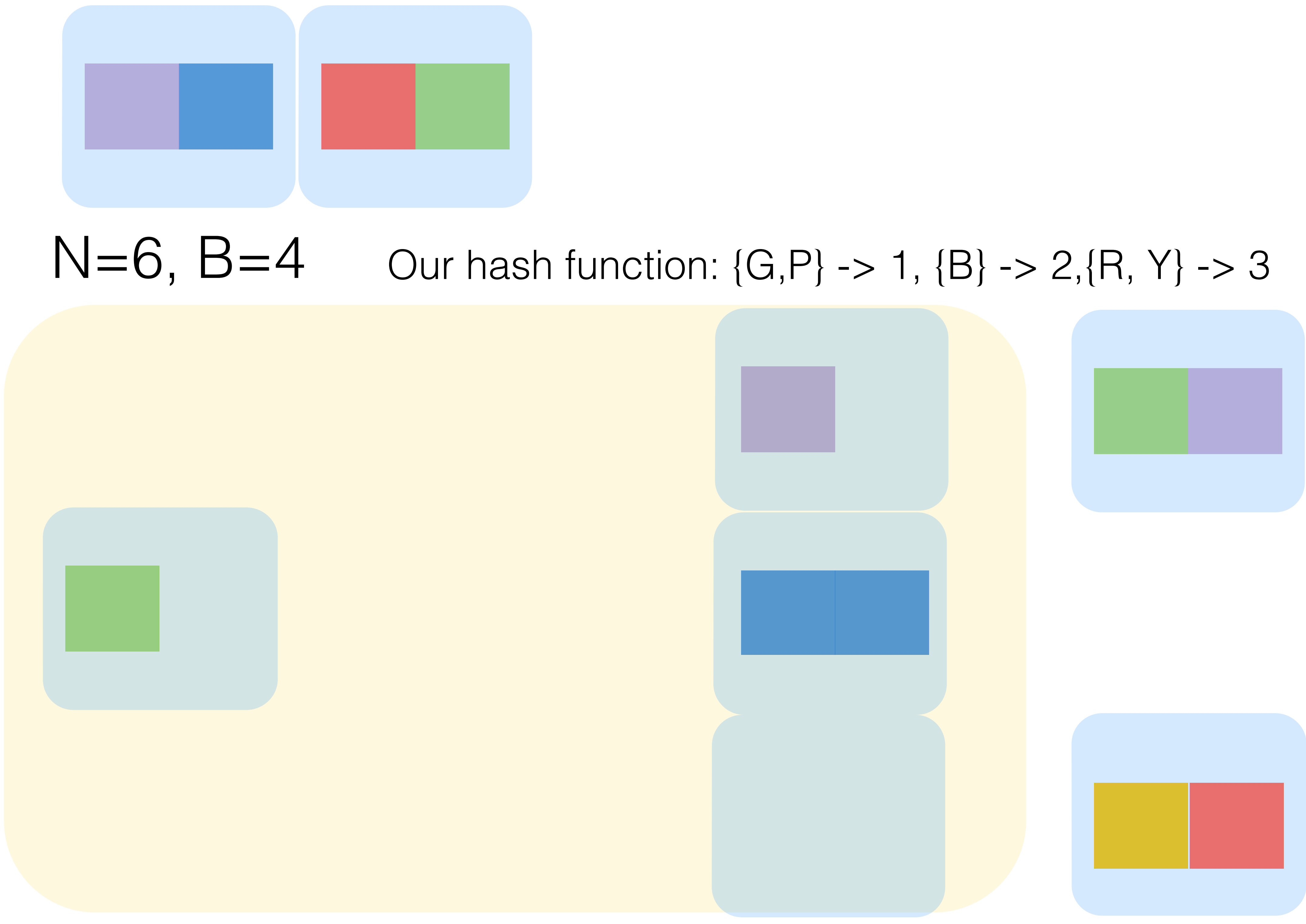
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



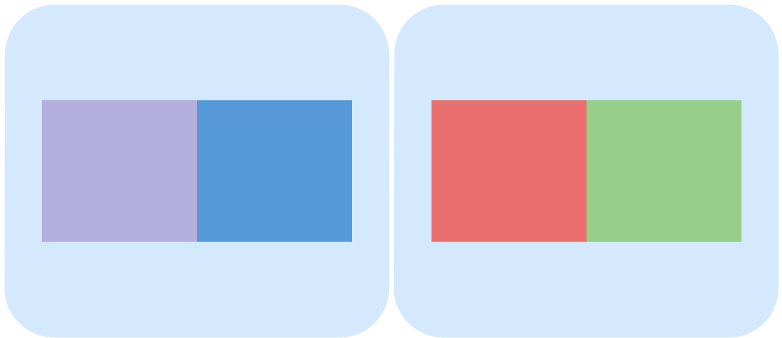
Pass 1: Divide

N=6, B=4

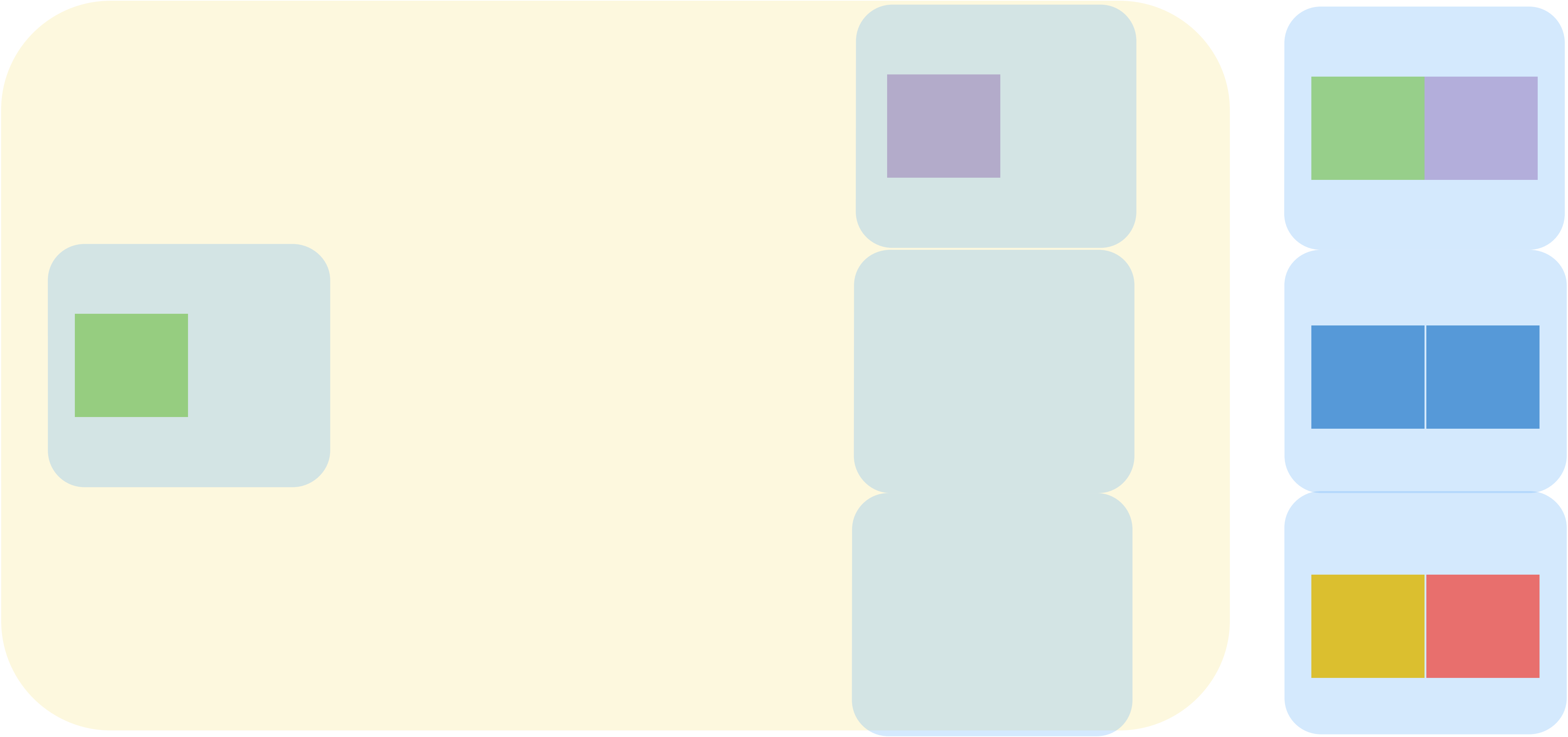
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



Pass 1: Divide



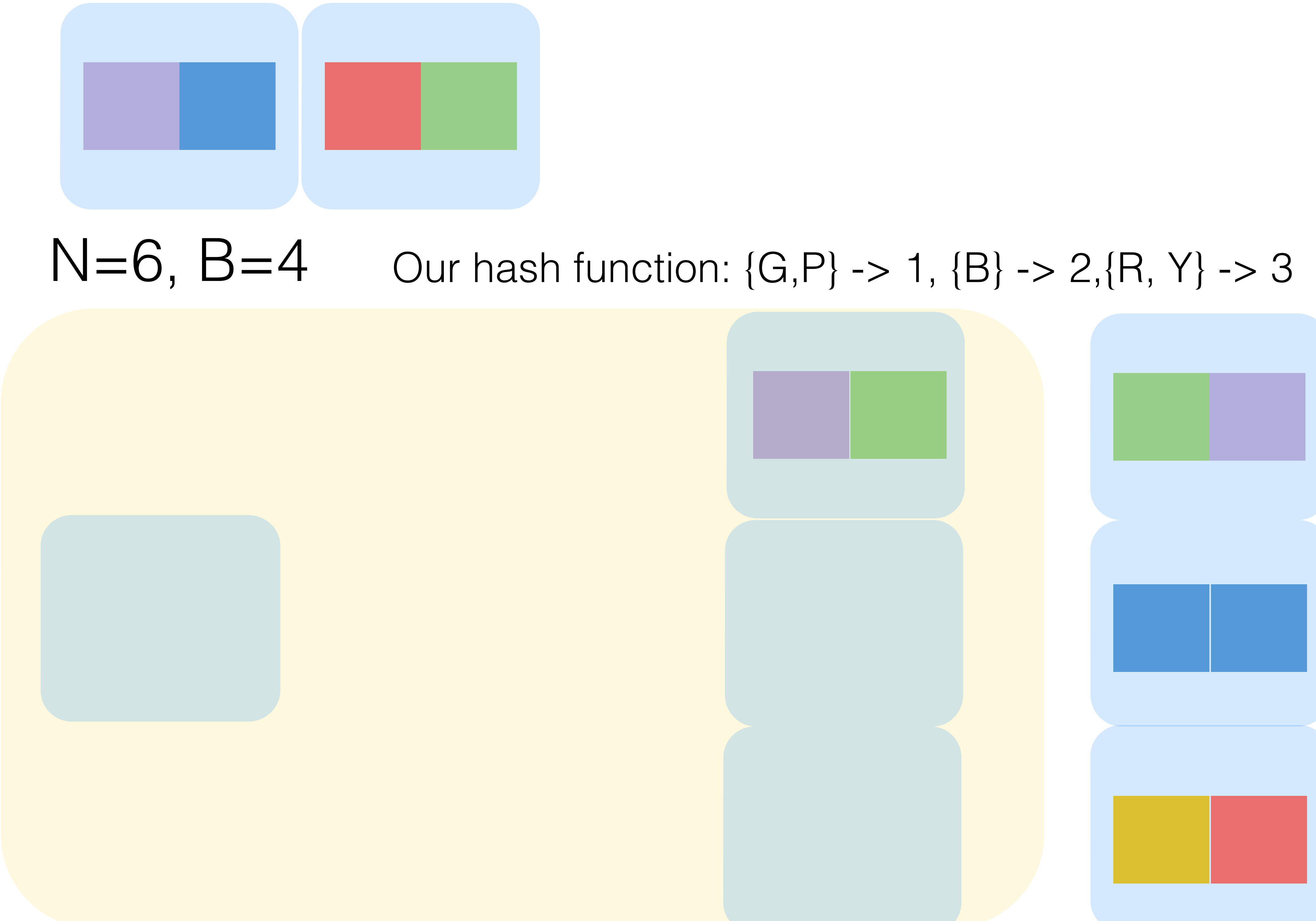
N=6, B=4   Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



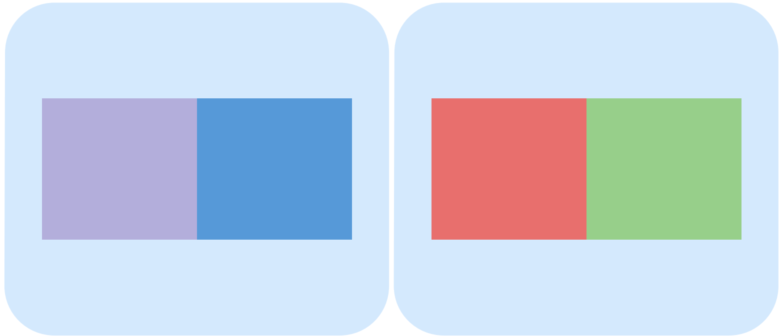
Pass 1: Divide

N=6, B=4

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

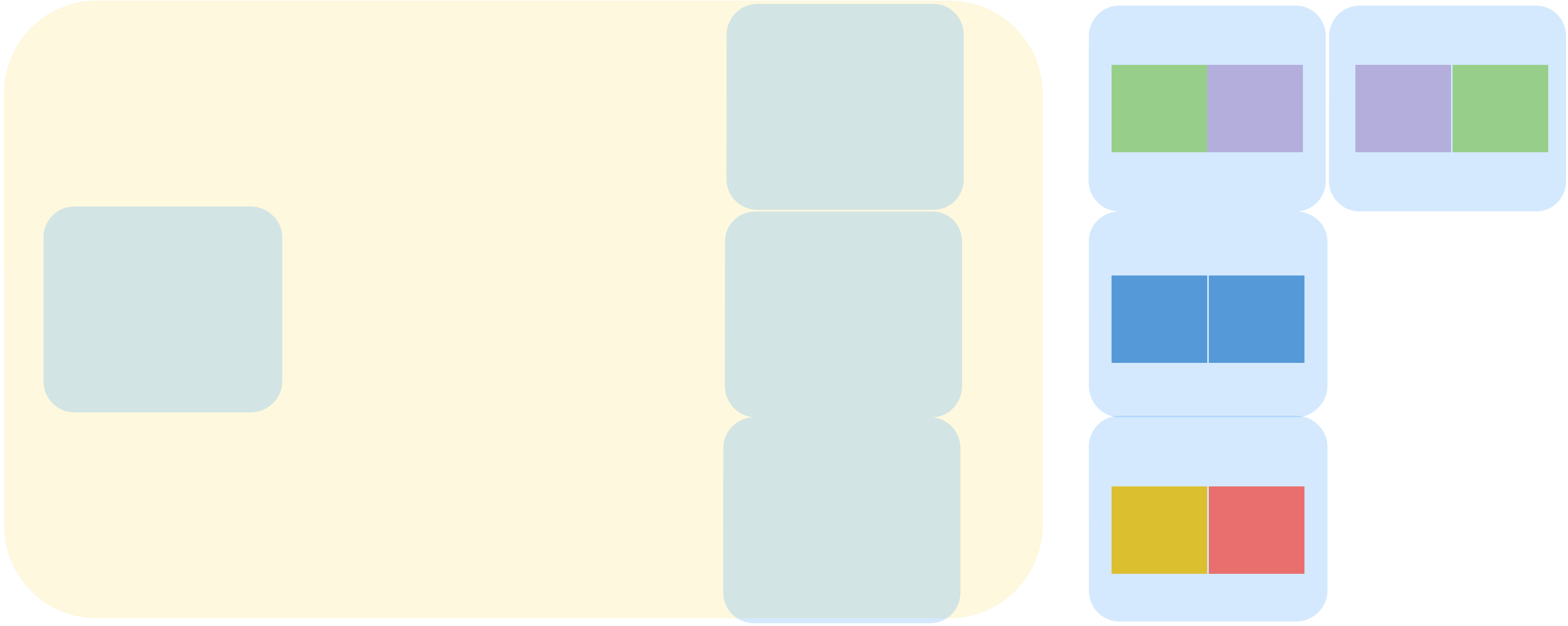


Pass 1: Divide

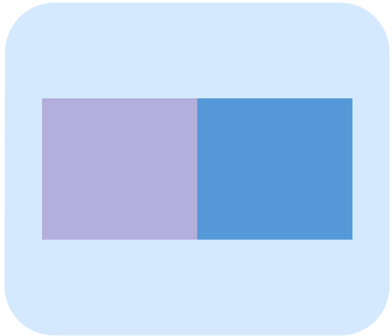


N=6, B=4

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

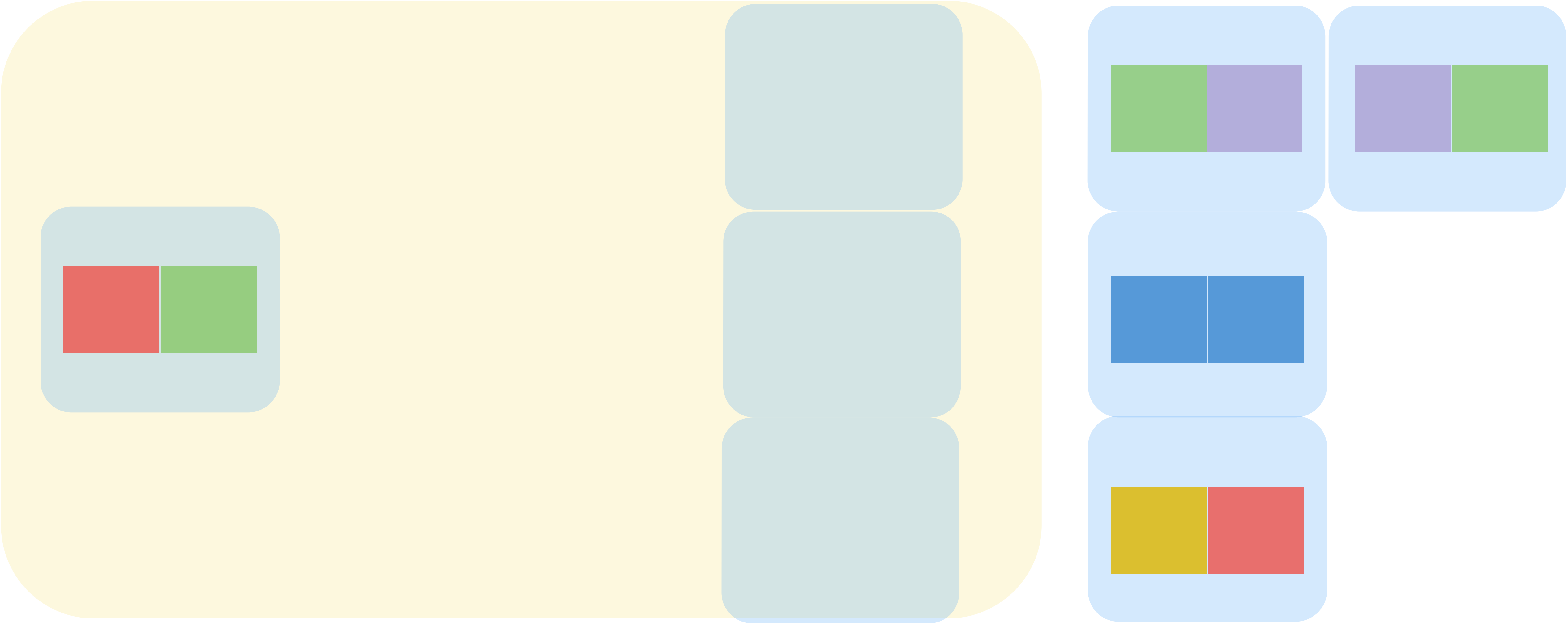


Pass 1: Divide

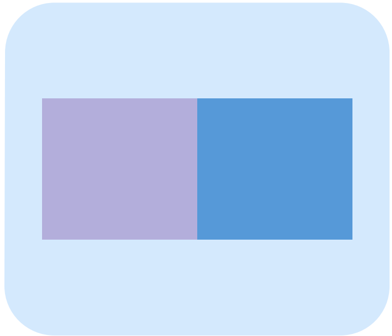


N=6, B=4

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

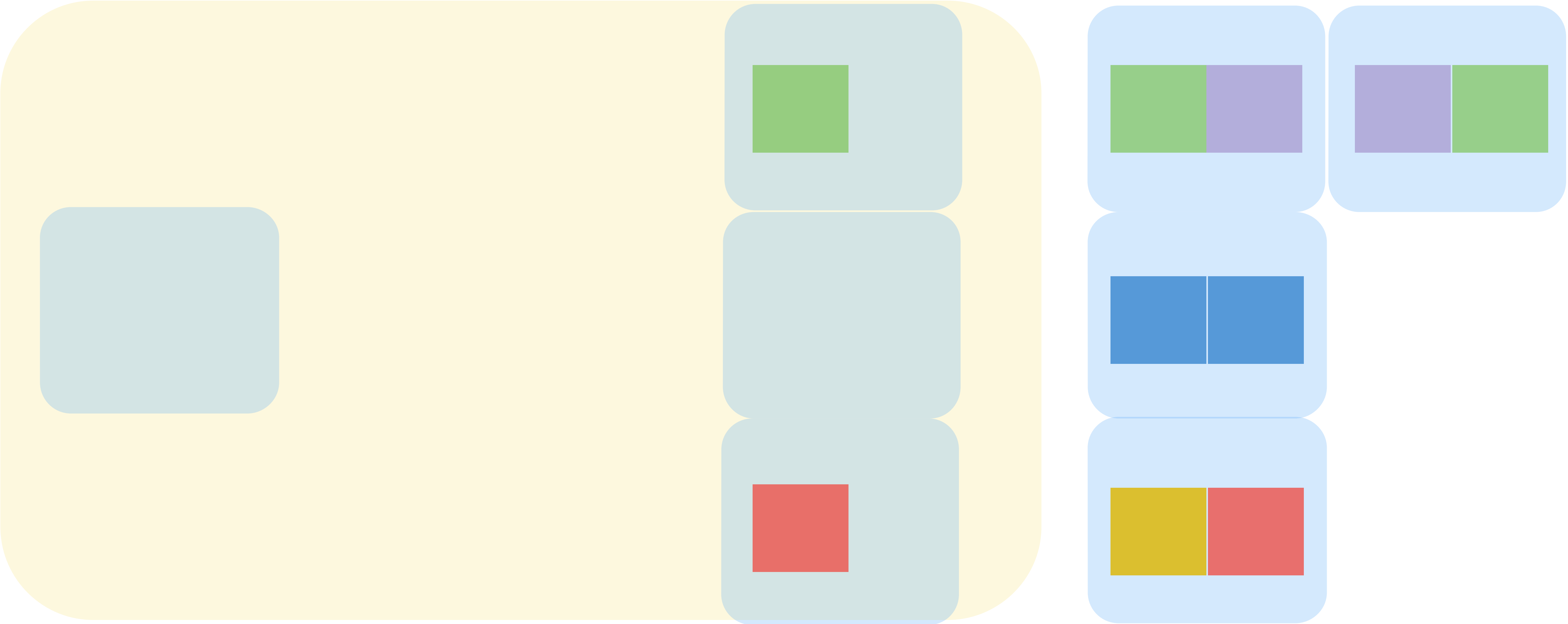


Pass 1: Divide



N=6, B=4

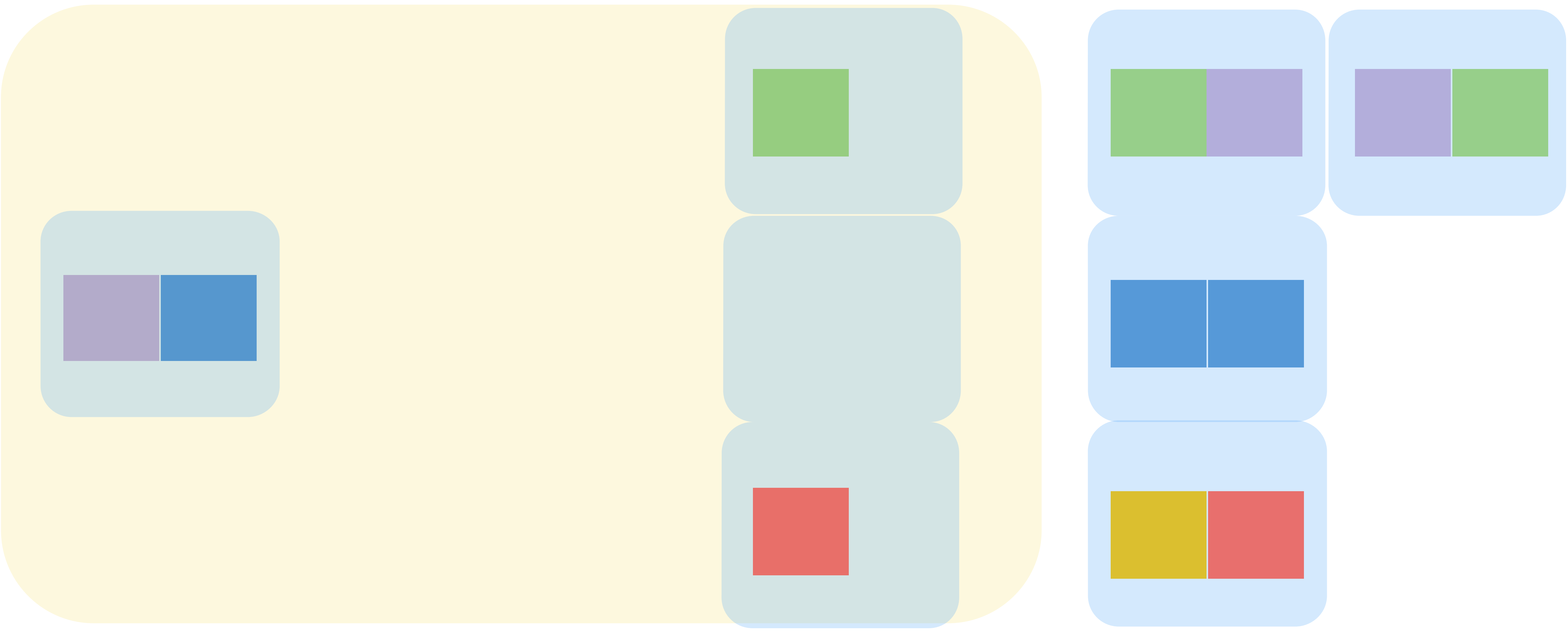
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



Pass 1: Divide

N=6, B=4

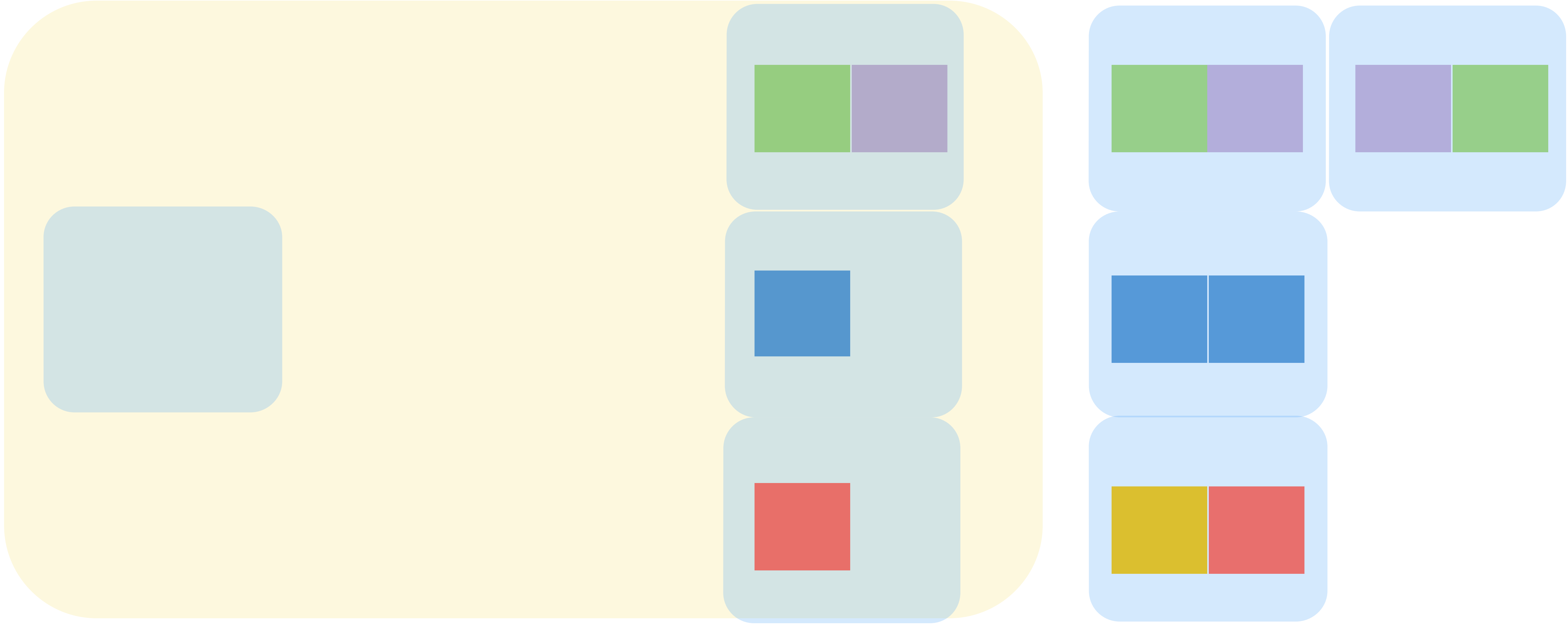
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



Pass 1: Divide

N=6, B=4

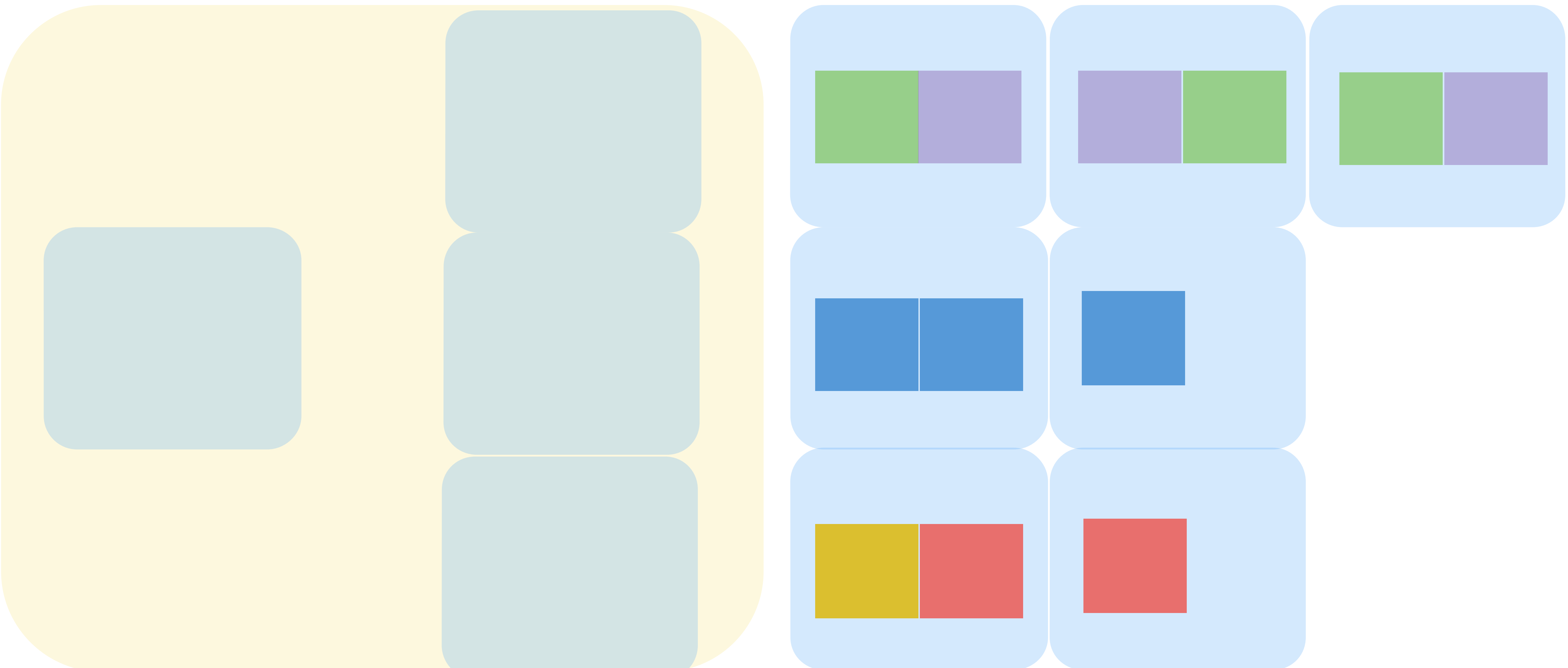
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



Pass 1: Divide

N=6, B=4

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



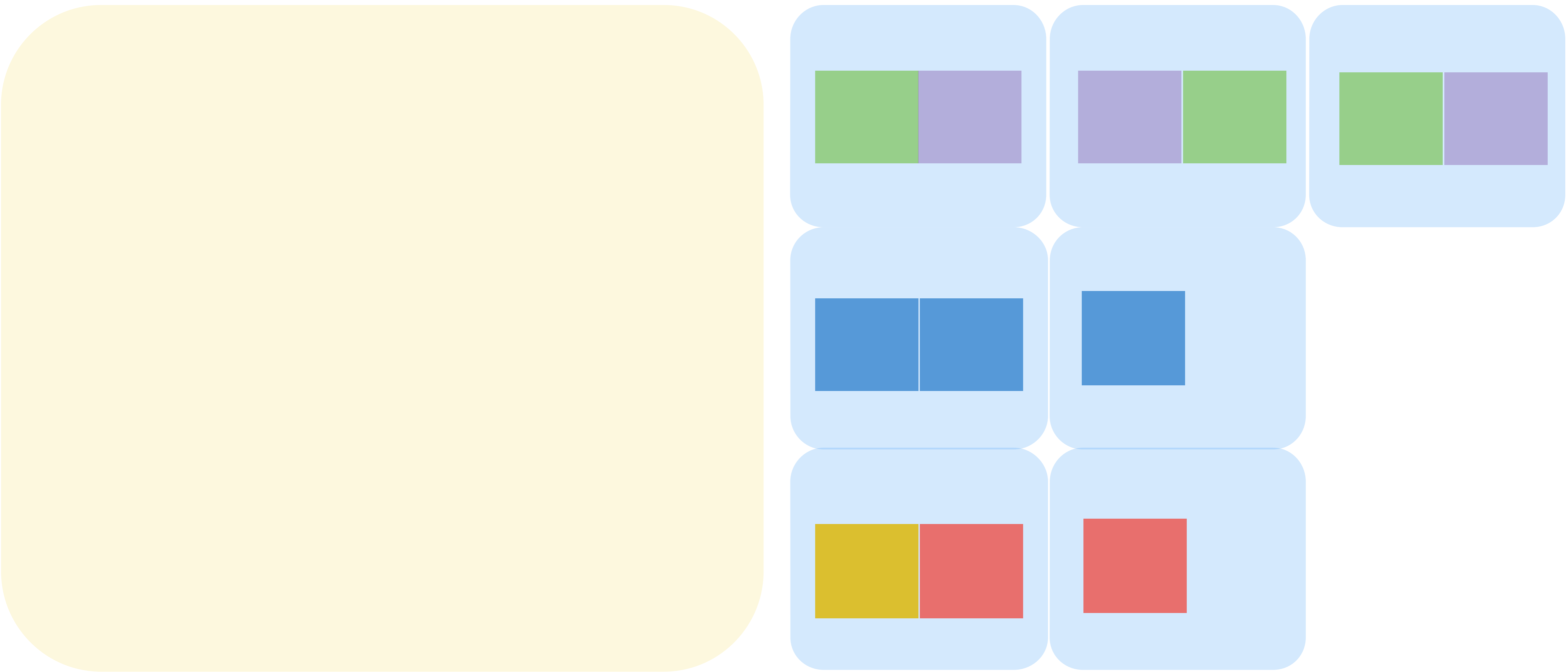
Pass 2: Conquer

- 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.

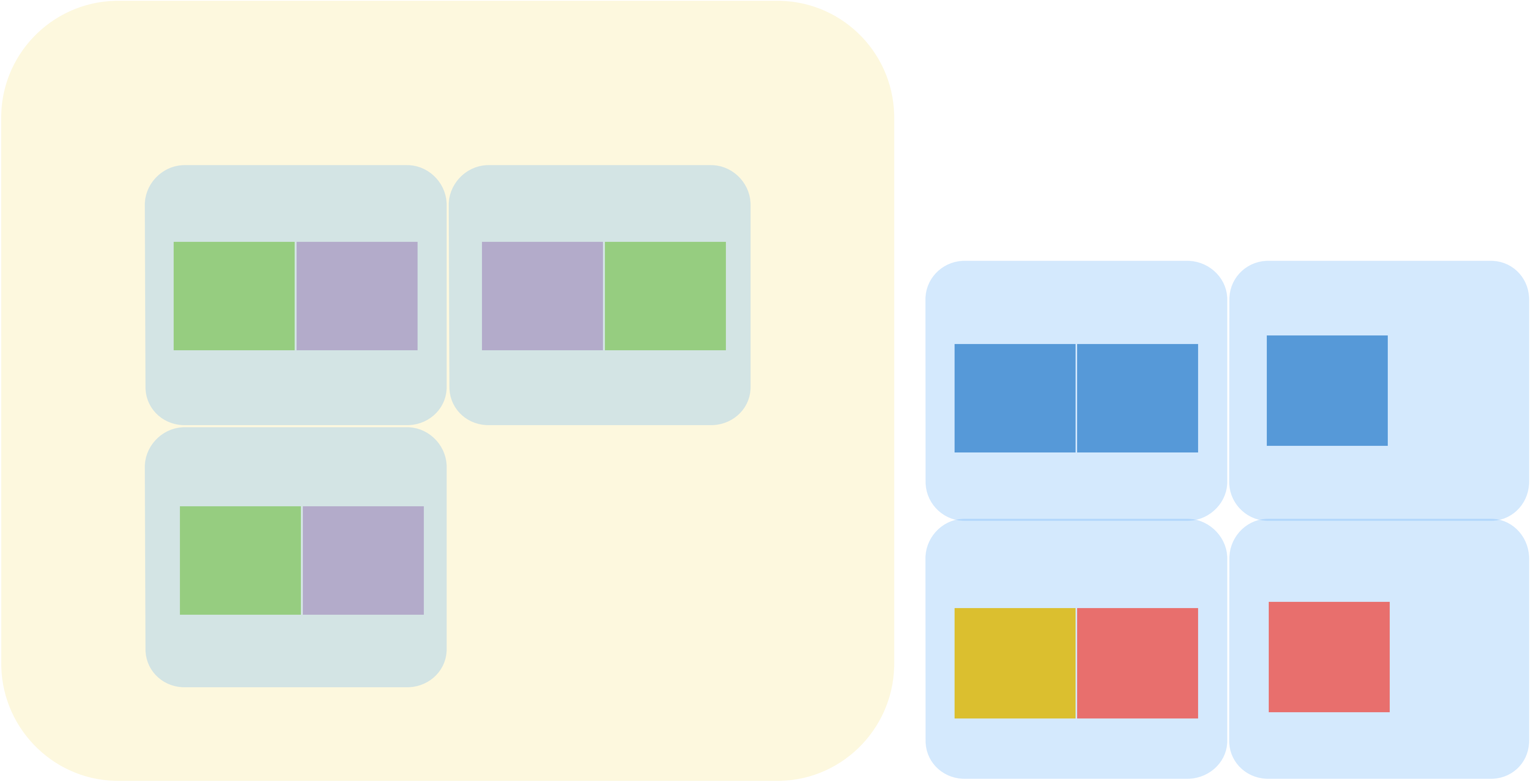
N=6, B=4



Pass 2: Conquer

Create in-memory table for each partition.

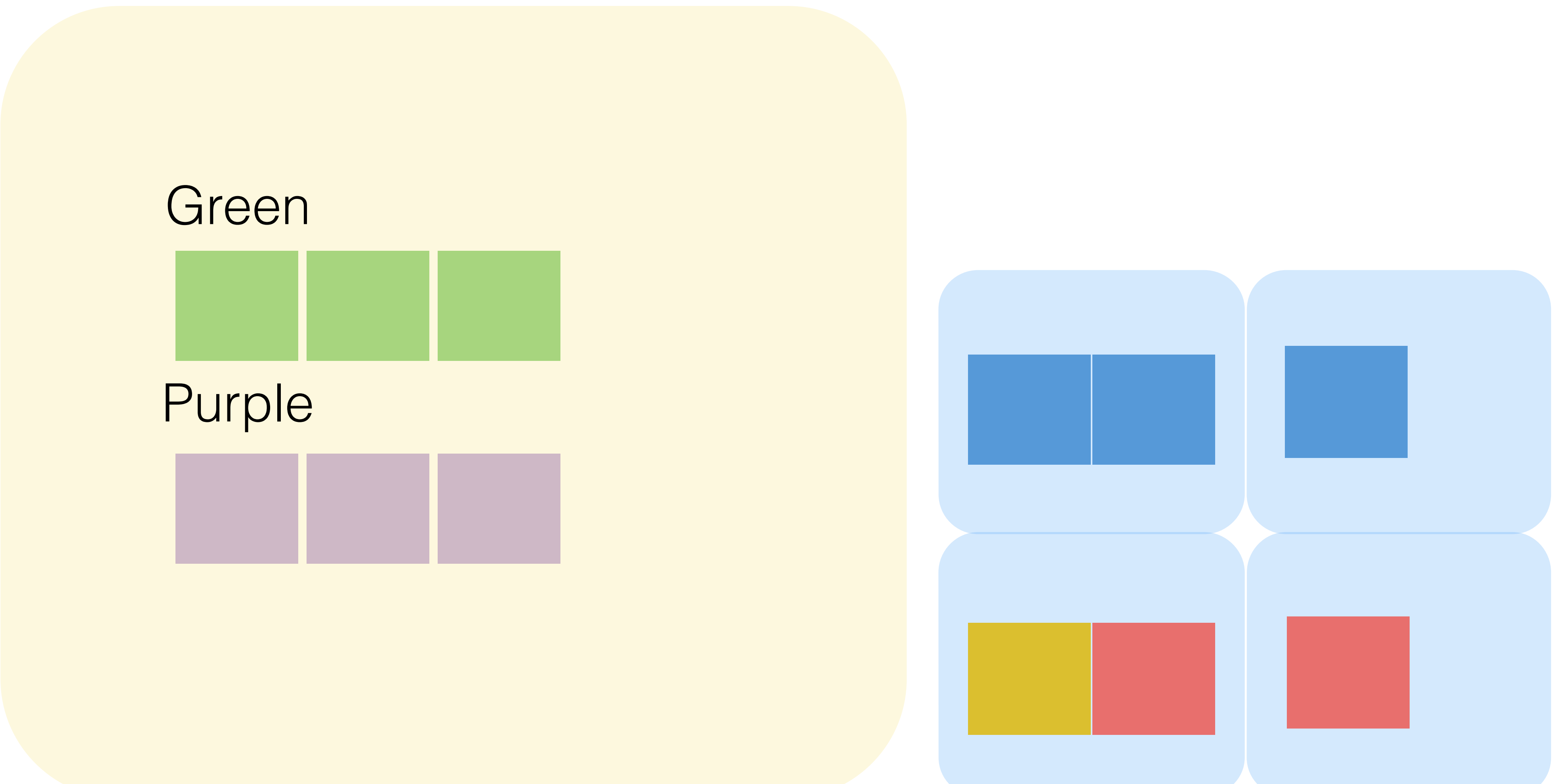
N=6, B=4



Pass 2: Conquer

Create in-memory table for each partition.

N=6, B=4



Worksheet #3, 4, 5

Single-Table SQL

```
SELECT [DISTINCT]  <column list>
FROM <relation list>
[WHERE <predicate>]
[GROUP BY <column list> [HAVING <predicate>]]
[ORDER BY <column list>];
```

```
SELECT standing, gpa, COUNT(*)
FROM Students
WHERE sname STARTS_WITH ‘A’
GROUP BY standing, gpa
HAVING COUNT(*) > 3;
```

```
SELECT year_released, COUNT(*)
FROM Albums
WHERE year_released < 2000
GROUP BY year_released;
```

SELECT year_released, COUNT(*)
Output total # of albums in each release year

FROM Albums
Query on albums table

WHERE year_released < 2000
Only include albums released before 2000

GROUP BY year_released;
Group by year it was released

Worksheet #6, 7, 8, 9