Welcome to CS 186!

TA: Bryan Munar
OH: Mondays 11-12pm and Thursdays 2:30-3:30pm (651 Soda)
DISC: Tuesdays 11-12am (136 Barrows) and Wednesdays 10-11am (130 Wheeler)

dat quality doe

About Me

Industry Experience:

- Lawrence Berkeley Lab: Quality Assurance Engineer (2012-2013)

- Guidewire Software: Software Engineering Intern (2014)

- Mozilla Corporation: Firefox Mobile Intern (2015)

Teaching Experience:

- iOS Game Development DeCal (2013)

Campus Involvements:

- UPE (Upsilon Pi Epsilon)

ICEBREAKER!!!

- Get into groups of 4 (or 3)
- Tell your name, favorite comfort food, and why you’re taking this class

Logistics and Course Info.

Class Assignments:

- Vitamins: Assigned every Thursday, due every Monday (with the exception of Midterm weeks)

- Projects: 5 projects throughout the semester (dates due TBA)

Class Exams:

- Midterms: Oct. 5, Nov. 9, Dec. 2 —> All in class!

- NO FINAL!!11!!1! (yer welcomz)

Miscellaneous:

- Project partners are optional (but **highly recommended** if you don't have experience working with people in a programming setting)

- Any questions about Logistics?

What to Expect

From the course:

- Learning what happens “under the hood” of a database (e.g. out-of-core algorithms, indexing, query optimization)

- Implementing simple database functionality (e.g. recovery, text search) and learning SQL

- Information to talk about in interviews, projects to add to your resume

From me:

- I’m here to help, teach, and be there for you! (aka come to my OH and talk to me)

- Albeit, I am not a database wizard who knows the answers to everything database-y

- I do mah best

Tips for Doing Well in CS 186

- Engaging as you learn the material (e.g. drawing diagrams to help you understand, talk to the TAs and answer Piazza posts, the material)

- Know what study methods work best! (e.g. writing an extensive cheat sheet, hand-writing notes, reviewing and redoing discussion problems and vitamins)

- Start all projects EARLY, even if they aren't too bad (srsly, something sketch always happens if you don't)

- Use Piazza! (crowdsource answers, don't be shy!)

Discussion 1: Sorting and Hashing

Overview:

1. External Merge Sort
2. Worksheet exercises
3. External Hashing
4. Worksheet exercises
5. (If there's time, Tournament Sort b/c not super important)

Know These Terms!

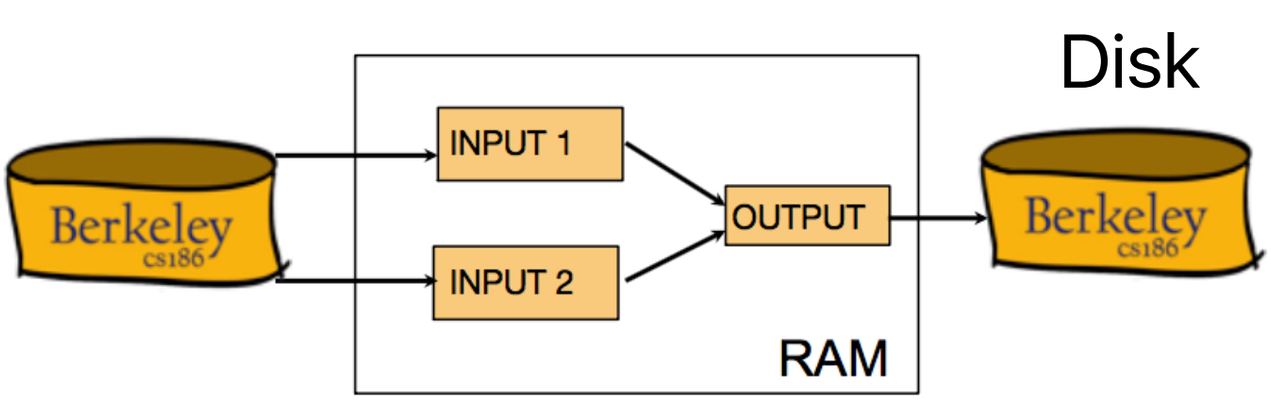
- **I/O:** unit of reading/writing to disk
- **Tuple:** entry in a page (e.g. (name = Joe, age = 35))
- **Page:** stores tuples (e.g. {(Joe, 35), (Michelle, 18), (Richie, 24)})
- **Buffer:** "region of a physical memory storage used to temporarily store data while it is being moved from one place to another," part of RAM
- **"In Memory":** referring to RAM ("allows data items to be accessed (read or written) in almost the same amount of time irrespective of the physical location of data inside the memory")
- **Sorted Runs:** A sorted subset of a table, size denoted by # of pages it spans
- **Out-of-core algorithm:** Sorts/hashes data from disk that can't be sorted/hashed in RAM (N pages >> B pages)

Why study external sorting and hashing?

This is because we want to sort/hash HUGE amounts of data from disk with minimal cost (do computations in RAM, remember the galaxy analogy from lecture)

Two-Way External Merge Sort

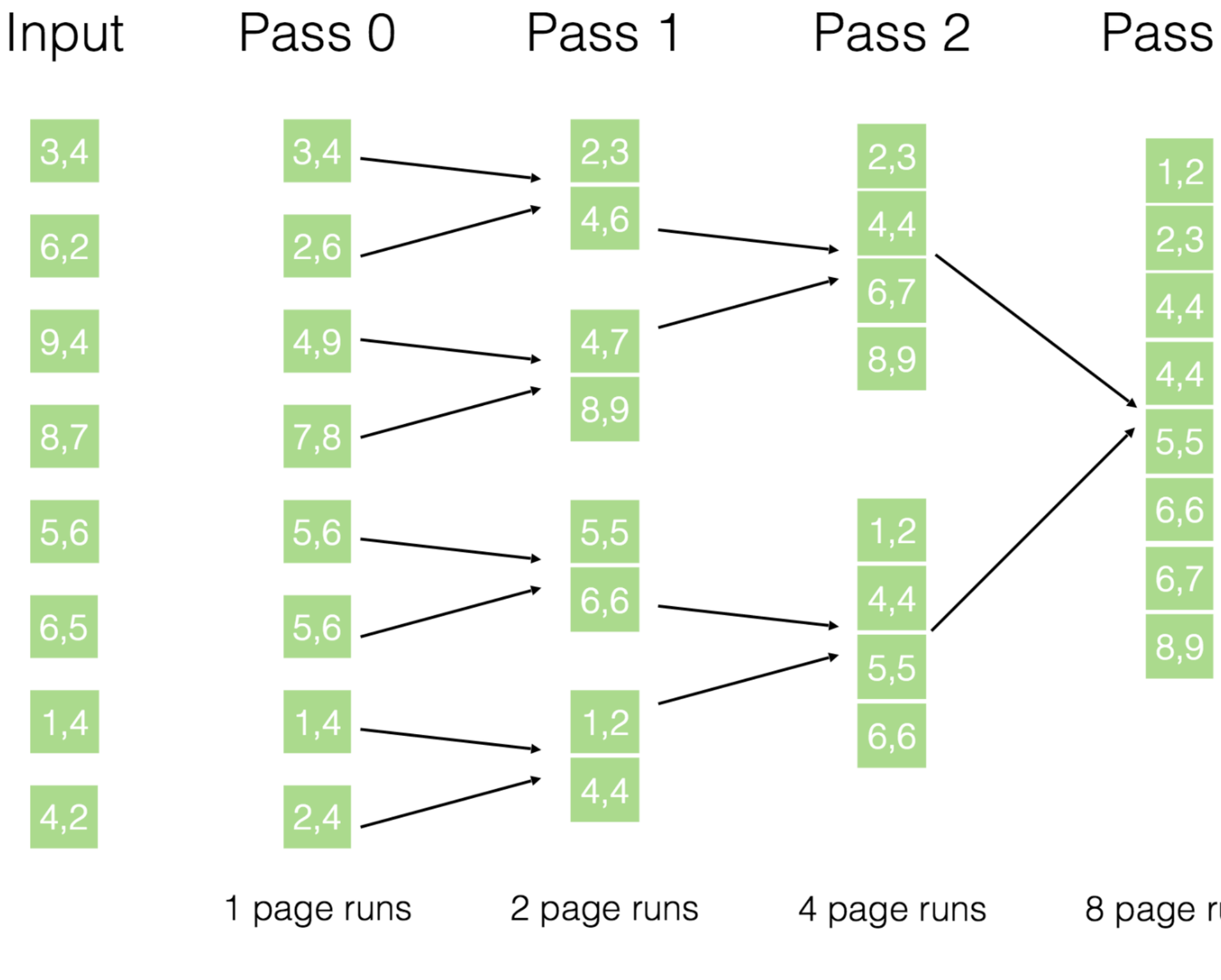
- **Pass 0 (conquer step):**
    - Read a page, sort it, then write it
    - Only ONE buffer page is used
- **Pass 1, 2, 3… etc. (merge step):**
    - Requires THREE buffer pages
    - Merge pairs of runs into runs twice as long
    - A streaming algorithm (uses limited memory)



3 buffer pages!

Thanks Joe!

Two-Way External Merge Sort



Thanks Michelle!

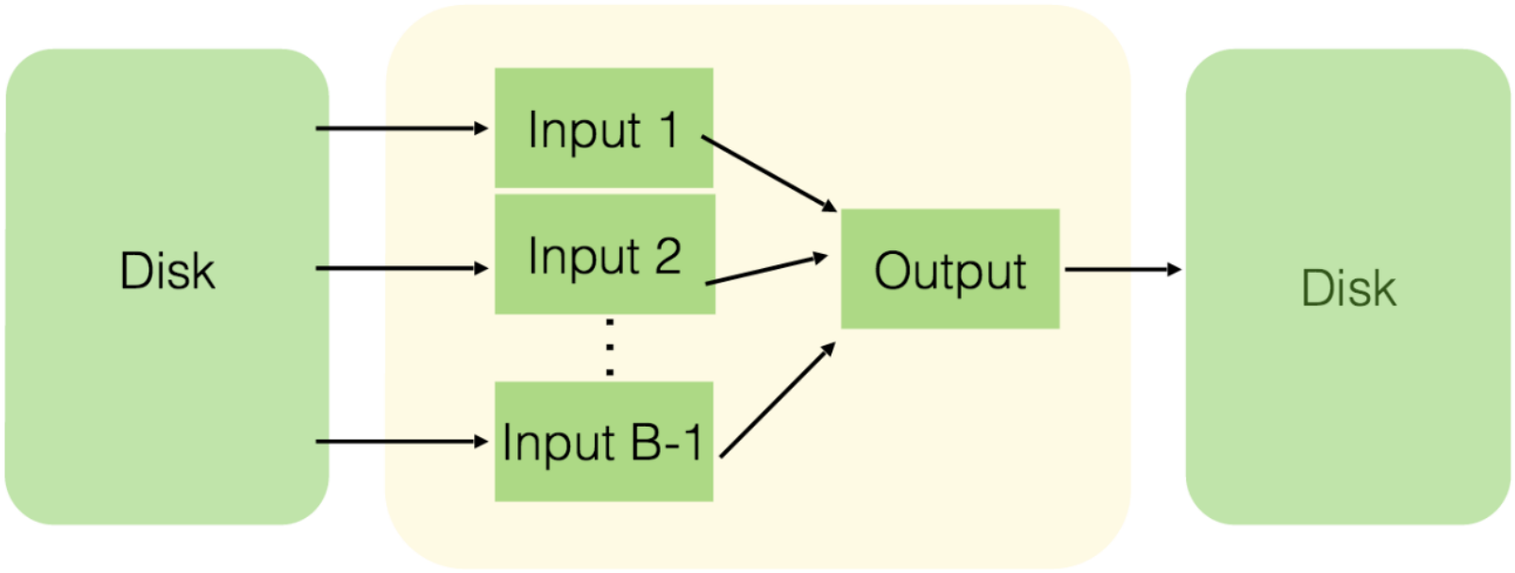
Cost of Two-Way External Merge Sort

- **Number of passes:** $1 + \log_2(N)$, where N = # of pages

- **Number of I/Os:** 2N * # of passes

- Think about how these formulas came about!

General External Merge Sort



Buffer size of B pages

Thanks
Michelle!

Cost of General External Merge Sort

- **Number of passes:**
    - ceil(1 + $\log_{B-1}$(ceil(N / B)))
- **Number of I/Os:**
    - 2N * # of passes
- Again, think about how these formulas came about! (Thinking about these things will help in harder cases e.g. external hashing)

External Merge Sort: Memory Requirement

- How big of a table can we sort in 2 passes?
  - Each sorted run after Phase 0 is of size B
  - Can merge up to B-1 sorted runs in Phase 1
- **Answer: B(B-1)**, can sort N pages of data in about $\sqrt{N}$ space

Is Two-Way External Merge Sort the same as General External Merge Sort with B=2?

nuuuu, do not confuse the two! (because people often do). One of the two sorts uses more buffers to create the initial sorted runs!

Do questions #3 and #4

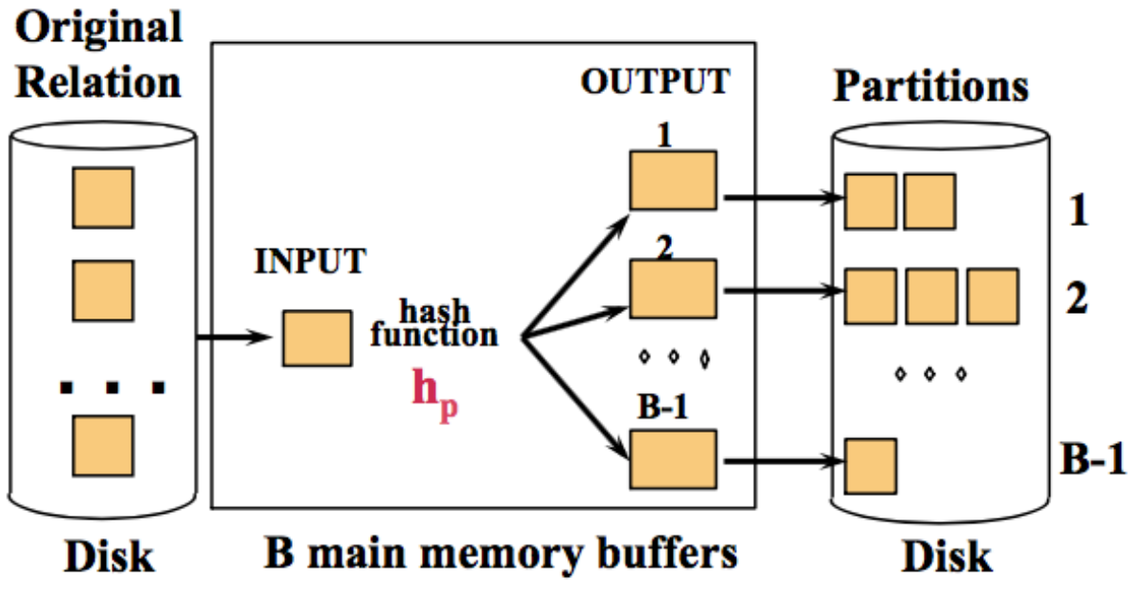
External Hashing

- Sometimes we don’t require order (e.g. removing duplicates, forming groups)
- What if we want to create a hash table, where you have certain keys that match to values? What if we want to aggregate data that doesn’t fit in memory?
- Sometimes it can be cheaper than sorting to just hash
- Minimize number of I/Os (especially random I/Os)

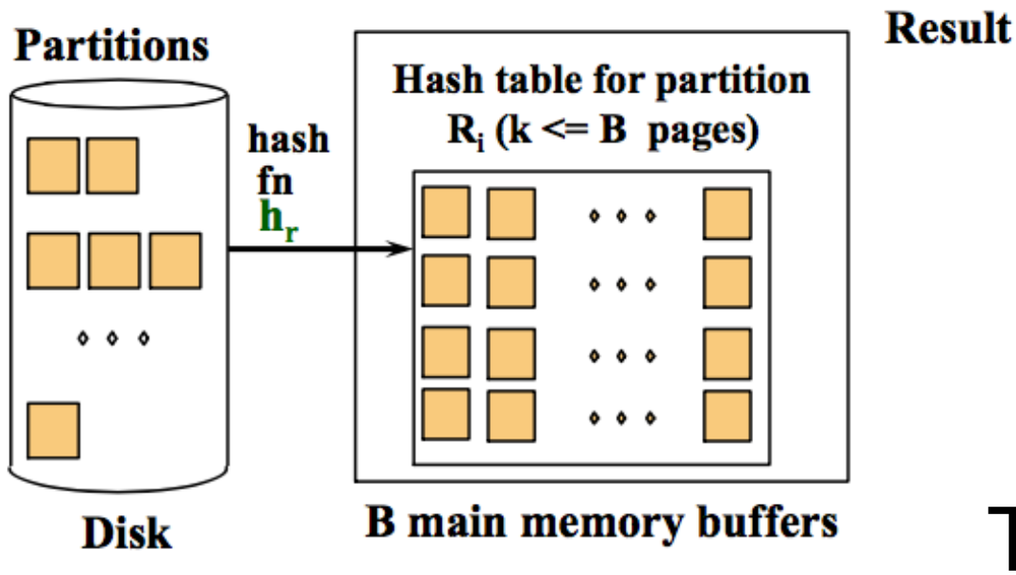
External Hashing

Two Phases

- Partition: (Divide)



- Rehash: (Conquer)

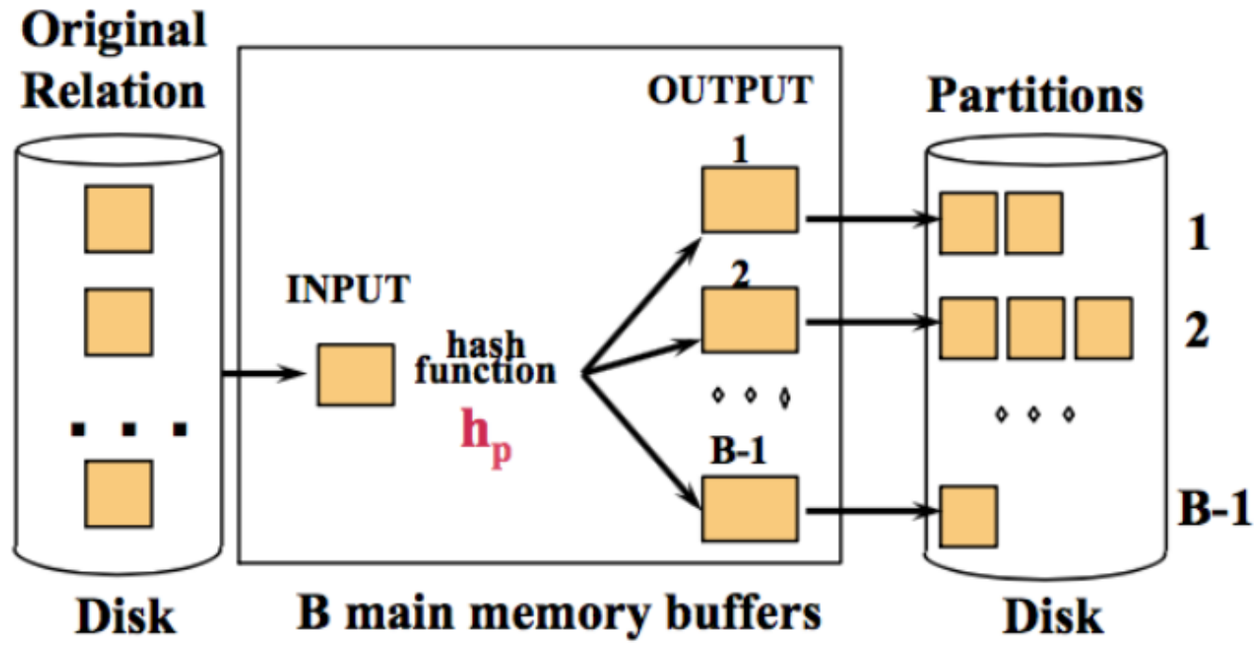


Thanks Joe!

Divide / Partition

Two Phases

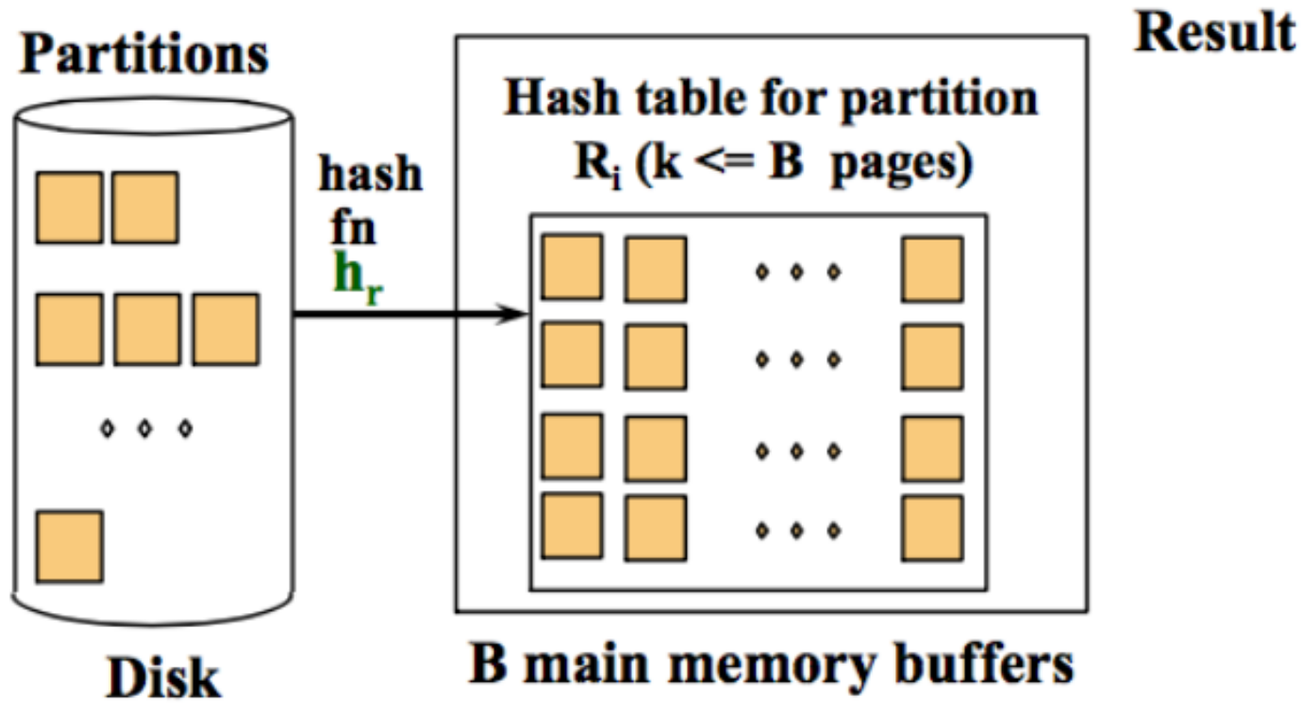
- Partition: (Divide)



- **Streaming Partition (divide):** Use a hash f'n h_p to stream records to disk partitions
  - All matches rendezvous in the same partition.
  - Streaming alg. to create partitions on disk:
    - "Spill" partitions to disk via output buffers

Rehash / Conquer

- Rehash: (Conquer)



- **ReHash (conquer):** Read partitions into RAM hash table **one at a time**, using hash f'n h_r
  - Then go through each bucket of this hash table to achieve rendezvous in RAM (rendezvous in the bucket) —> write back to disk

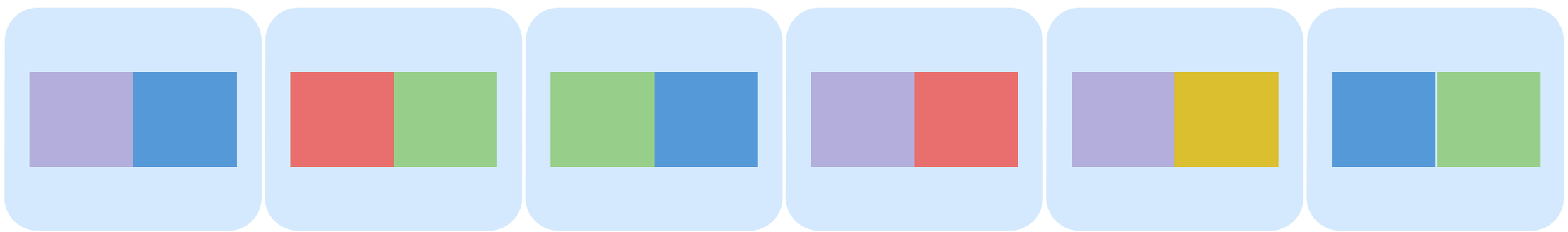
External Hashing example, or nah?

Aggregating Colors (example by Michelle)

- Goal: Group squares by color

- Setup: 12 squares, 2 can fit per page. We can hold 8 squares in memory.

- N=6, B=4



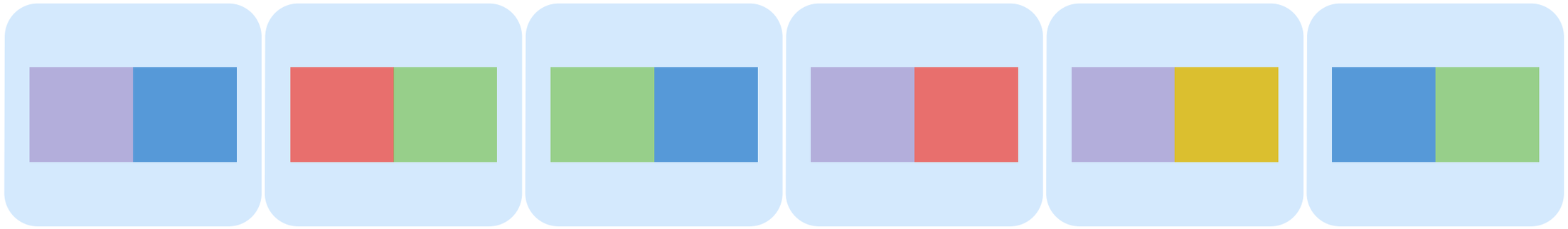
Pass 1: Divide

- 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

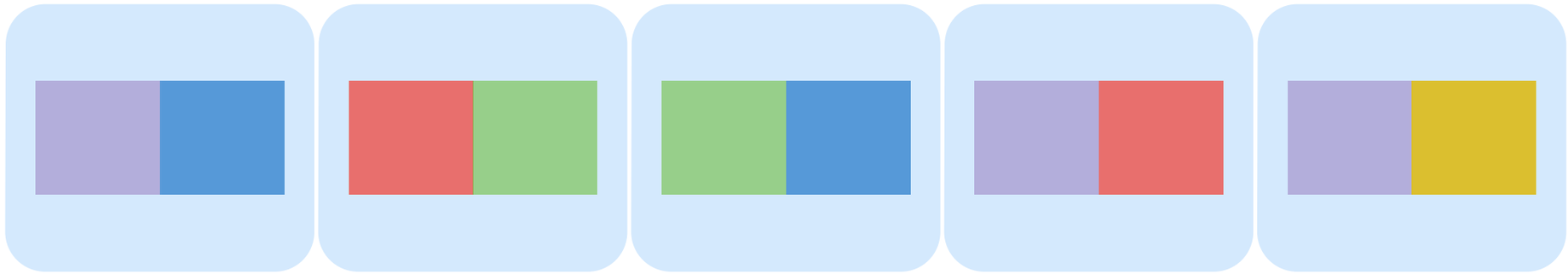
Pass 1: Divide



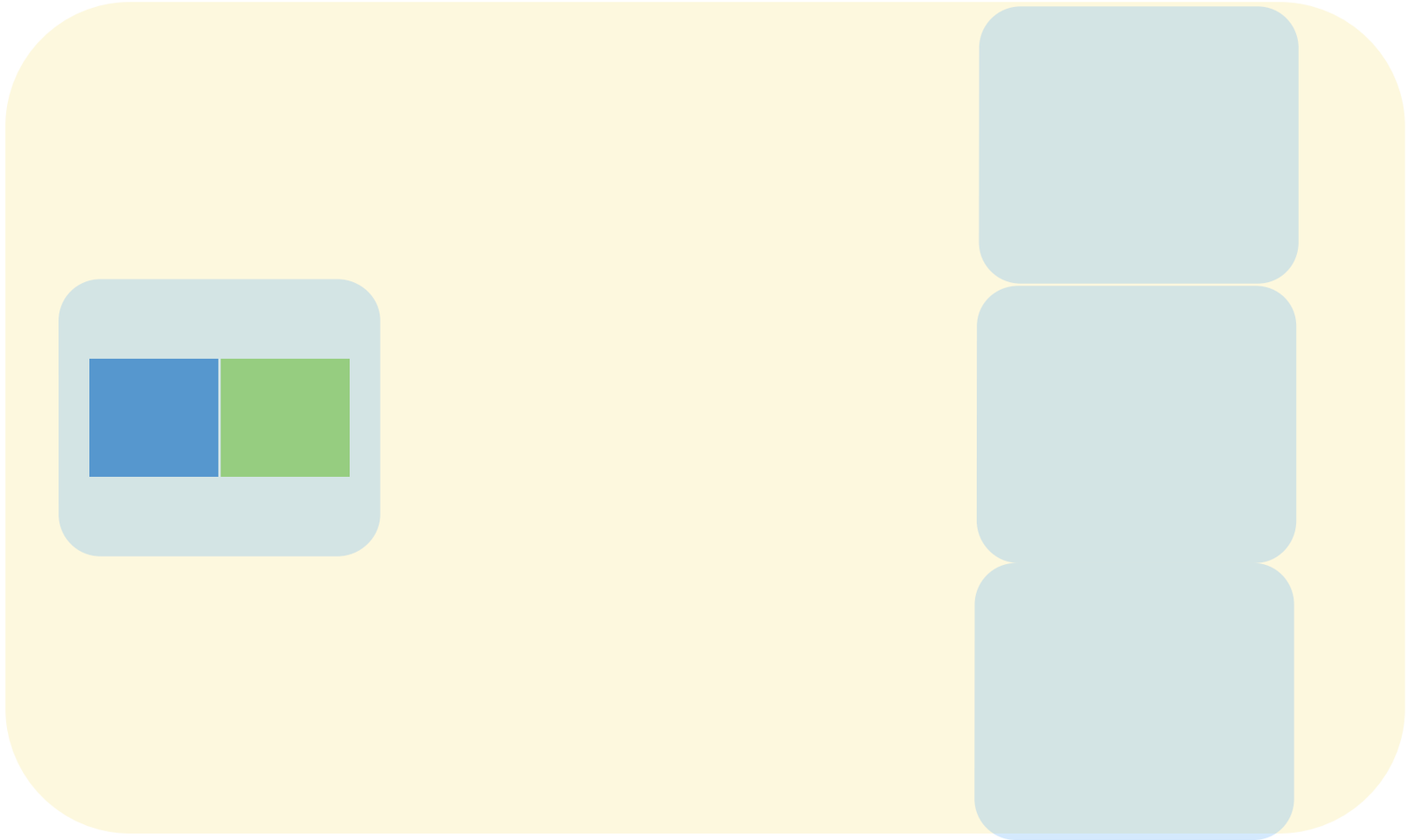
N=6, B=4

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

Pass 1: Divide

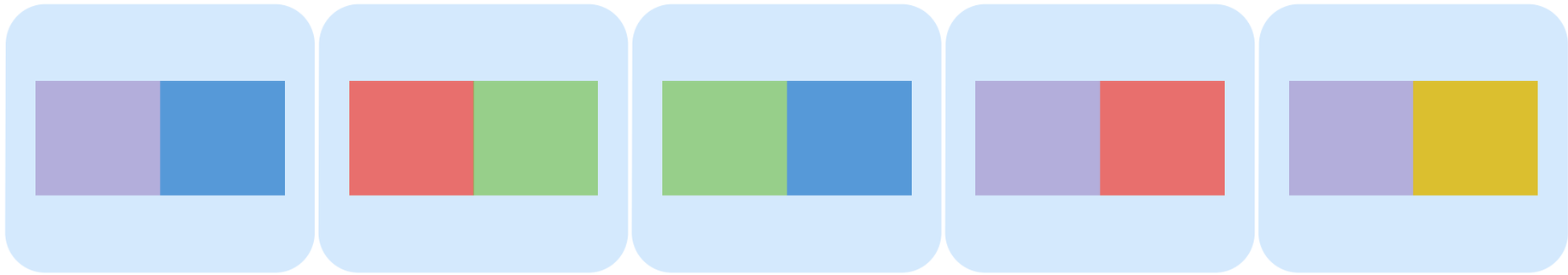


N=6, B=4

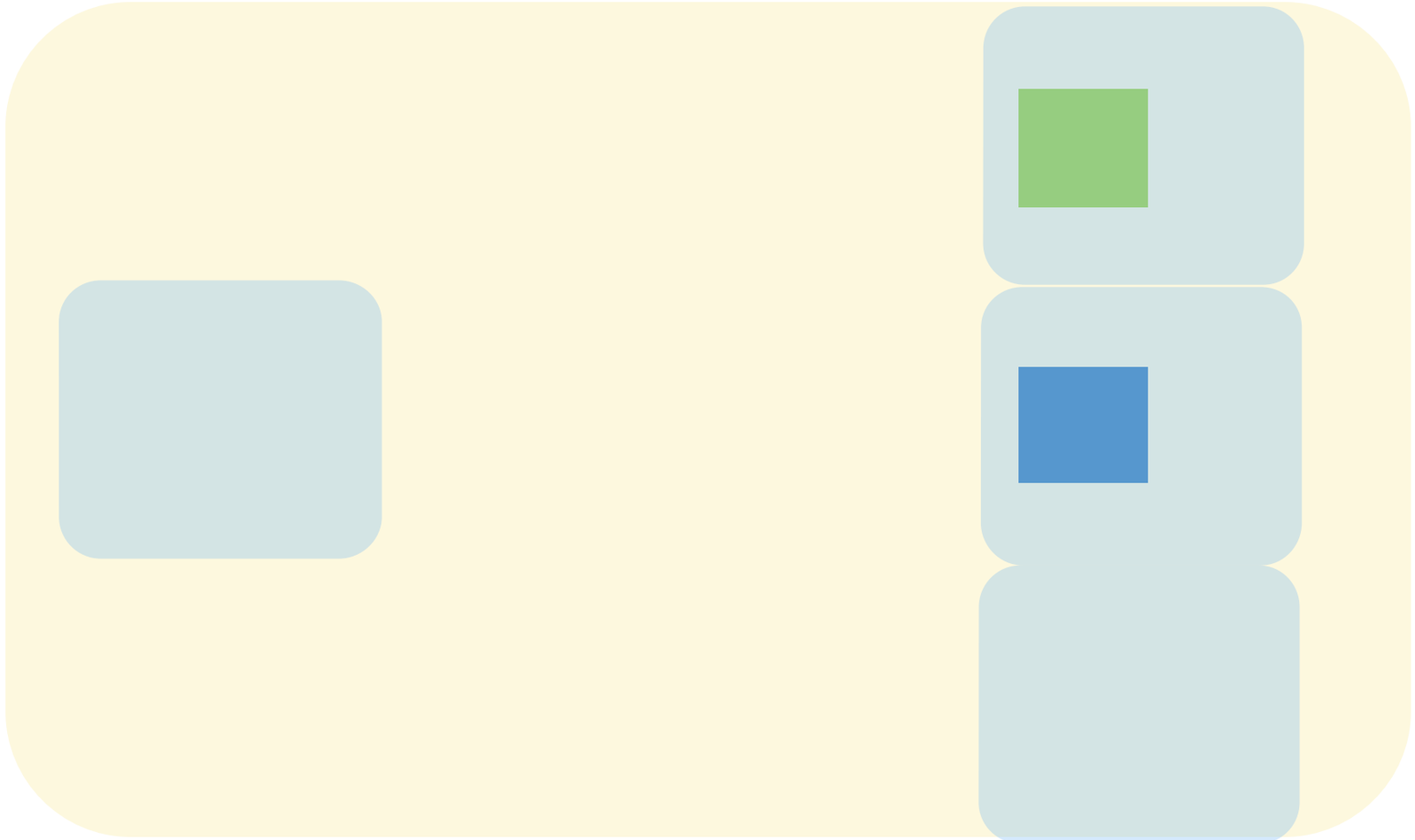


Assign colors to 3 partitions using hash function.
Our hash function:
{G,P} -> 1
{B} -> 2
{R, Y} -> 3

Pass 1: Divide

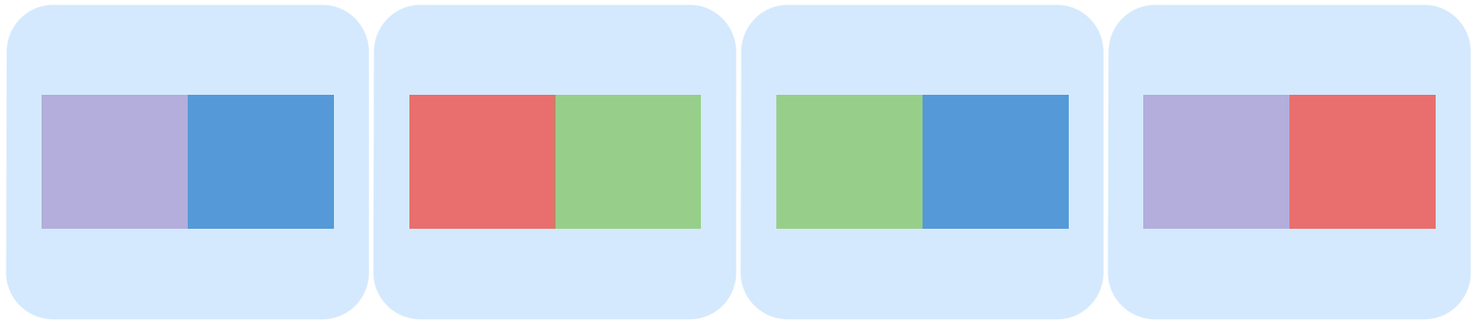


N=6, B=4

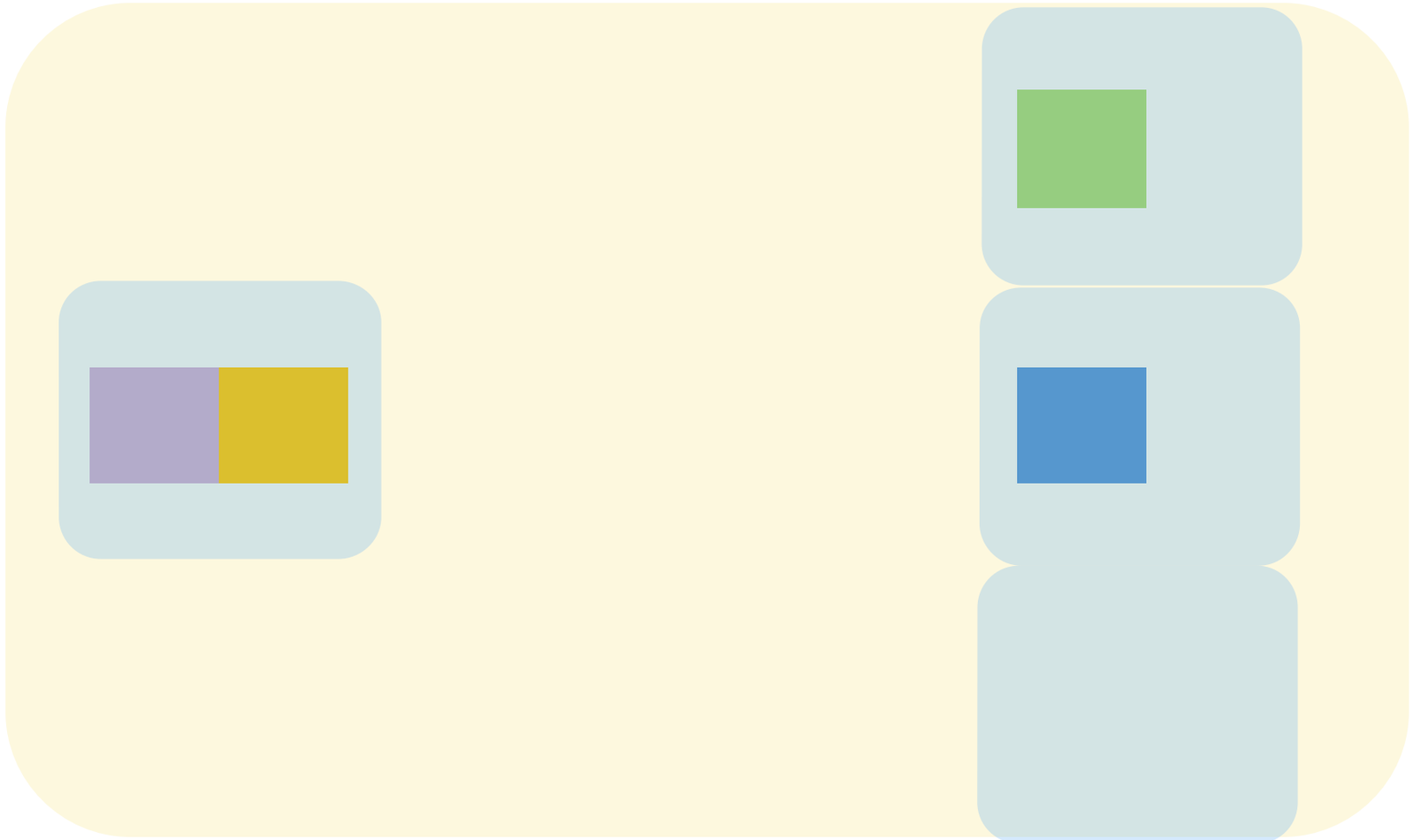


Assign colors to 3 partitions using hash function.
Our hash function:
{G,P} -> 1
{B} -> 2
{R, Y} -> 3

Pass 1: Divide

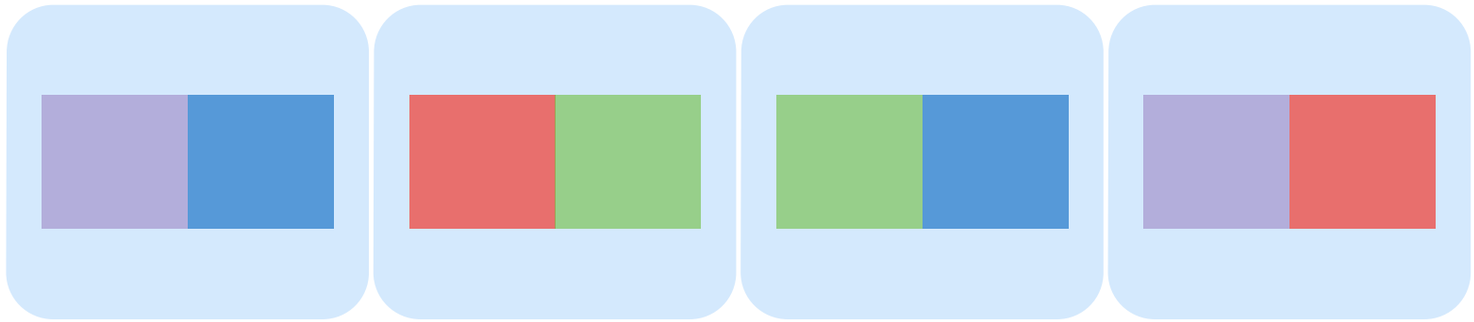


N=6, B=4

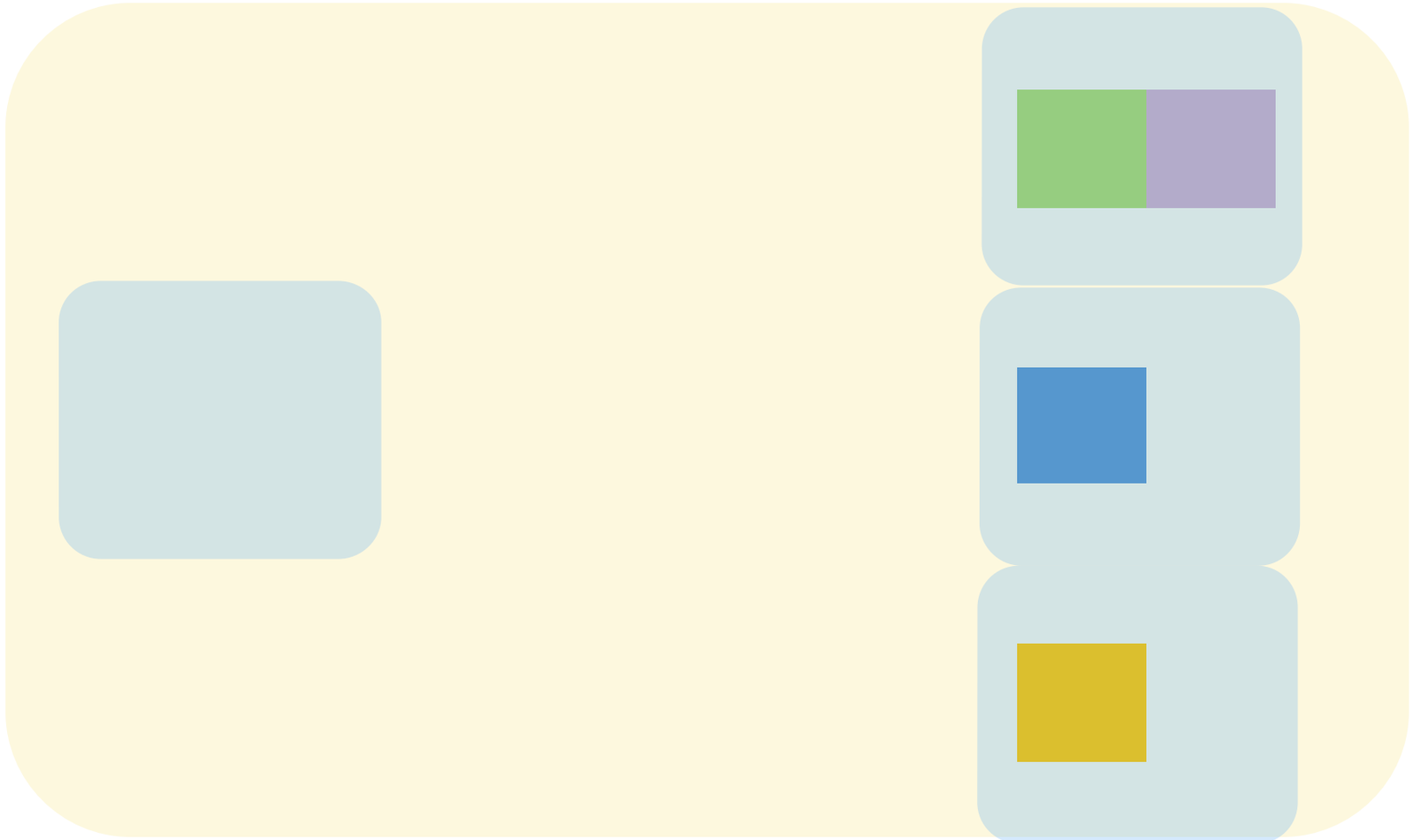


Assign colors to 3 partitions using hash function.
Our hash function:
{G,P} -> 1
{B} -> 2
{R, Y} -> 3

Pass 1: Divide

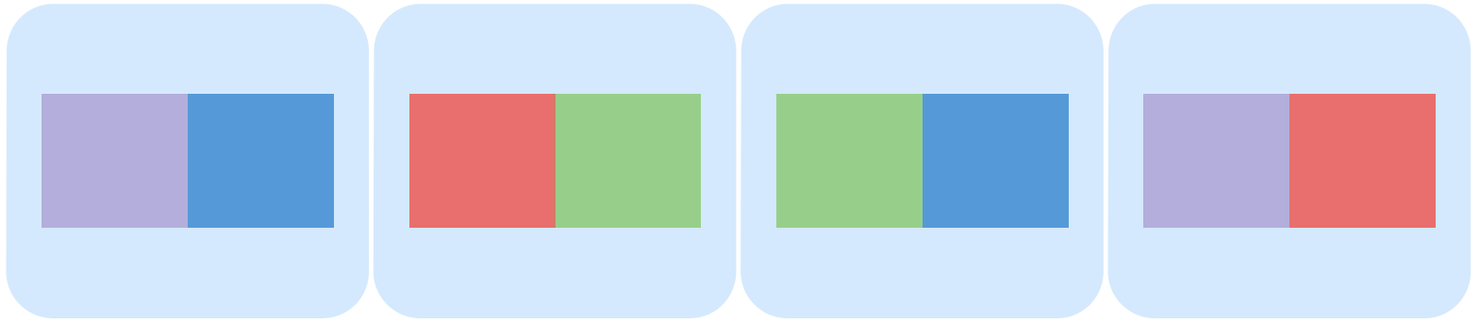


N=6, B=4



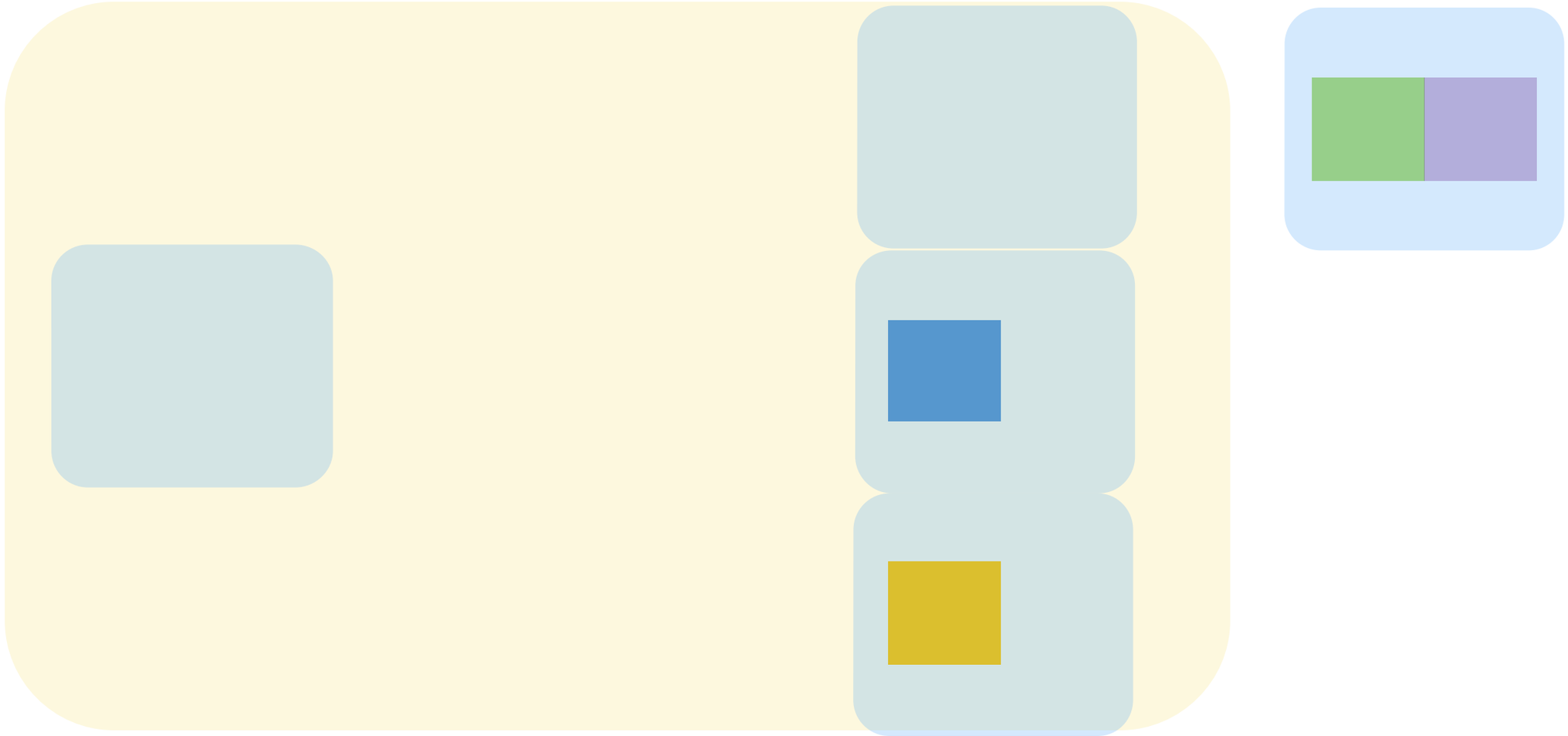
Assign colors to 3 partitions using hash function.
Our hash function:
{G,P} -> 1
{B} -> 2
{R, Y} -> 3

Pass 1: Divide

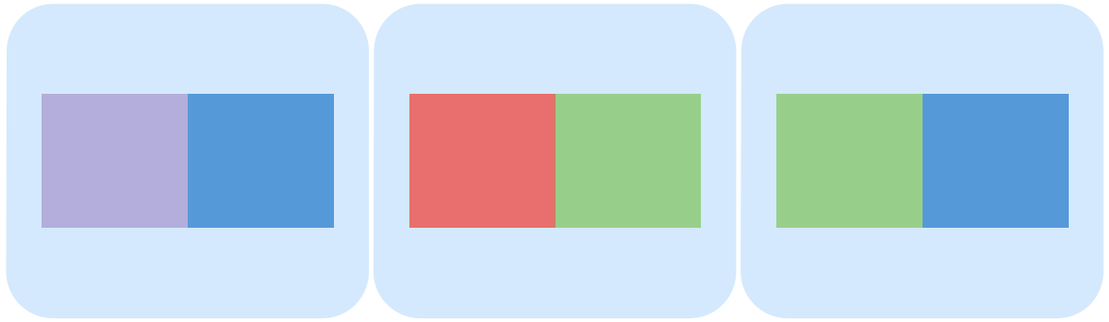


N=6, B=4

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

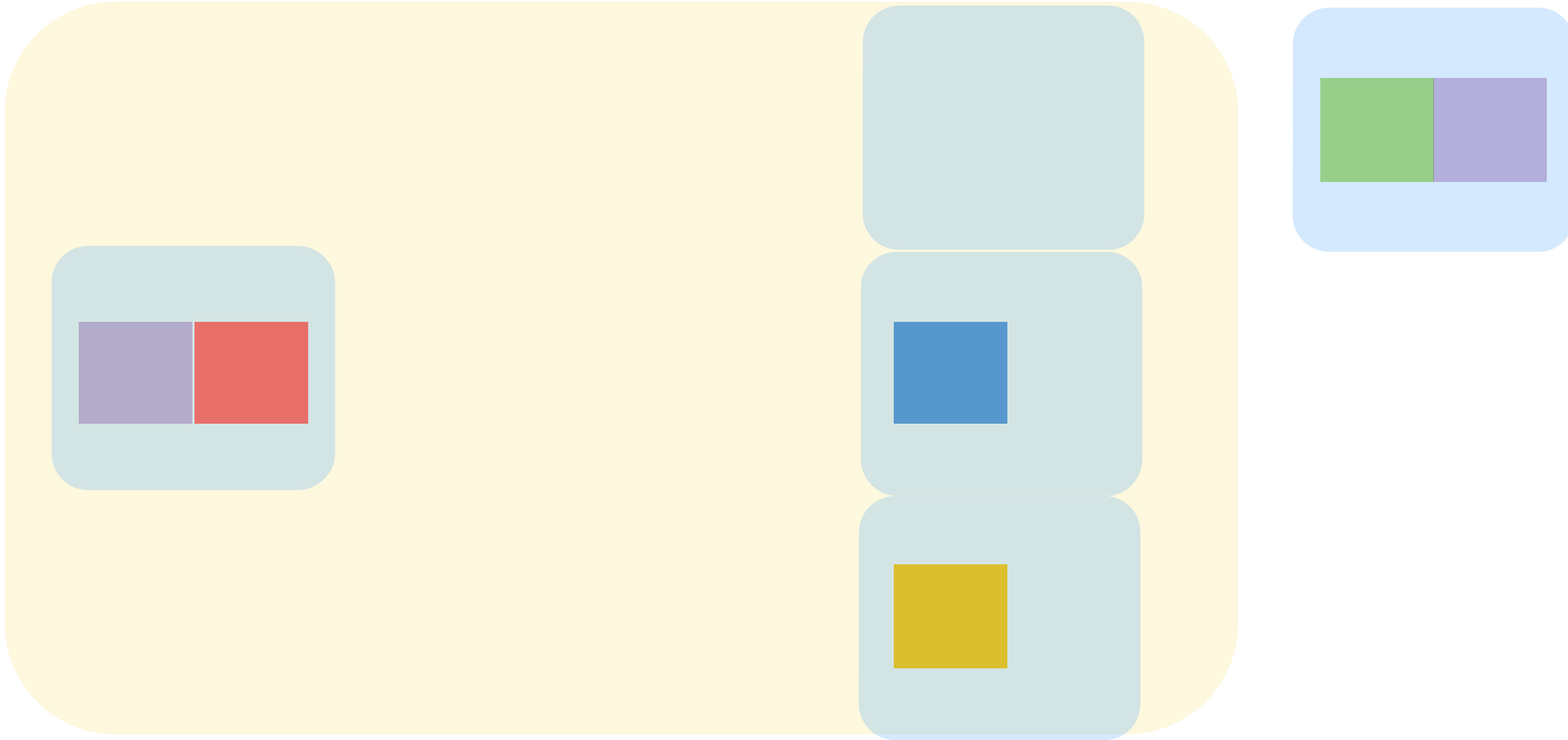


Pass 1: Divide

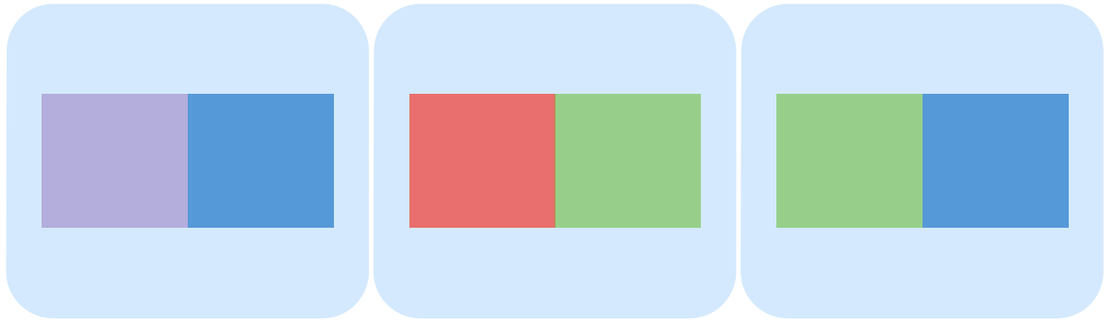


N=6, B=4

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

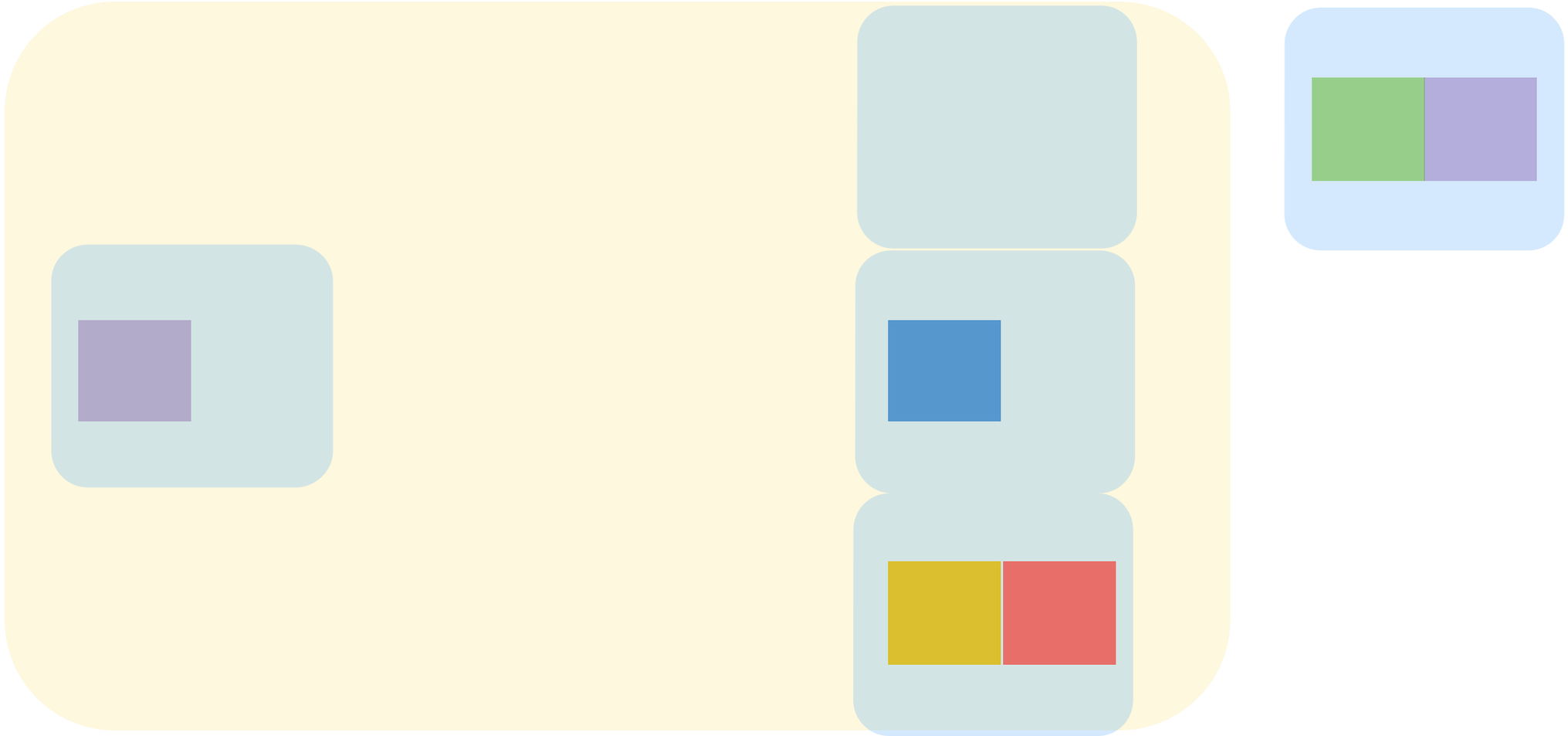


Pass 1: Divide

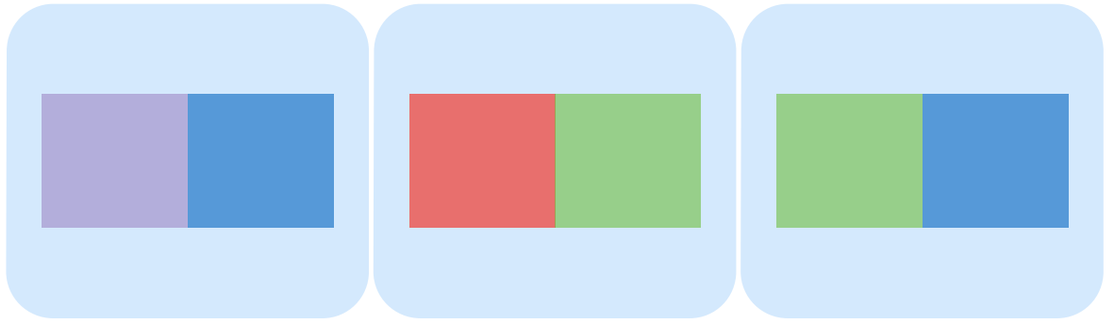


N=6, B=4

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

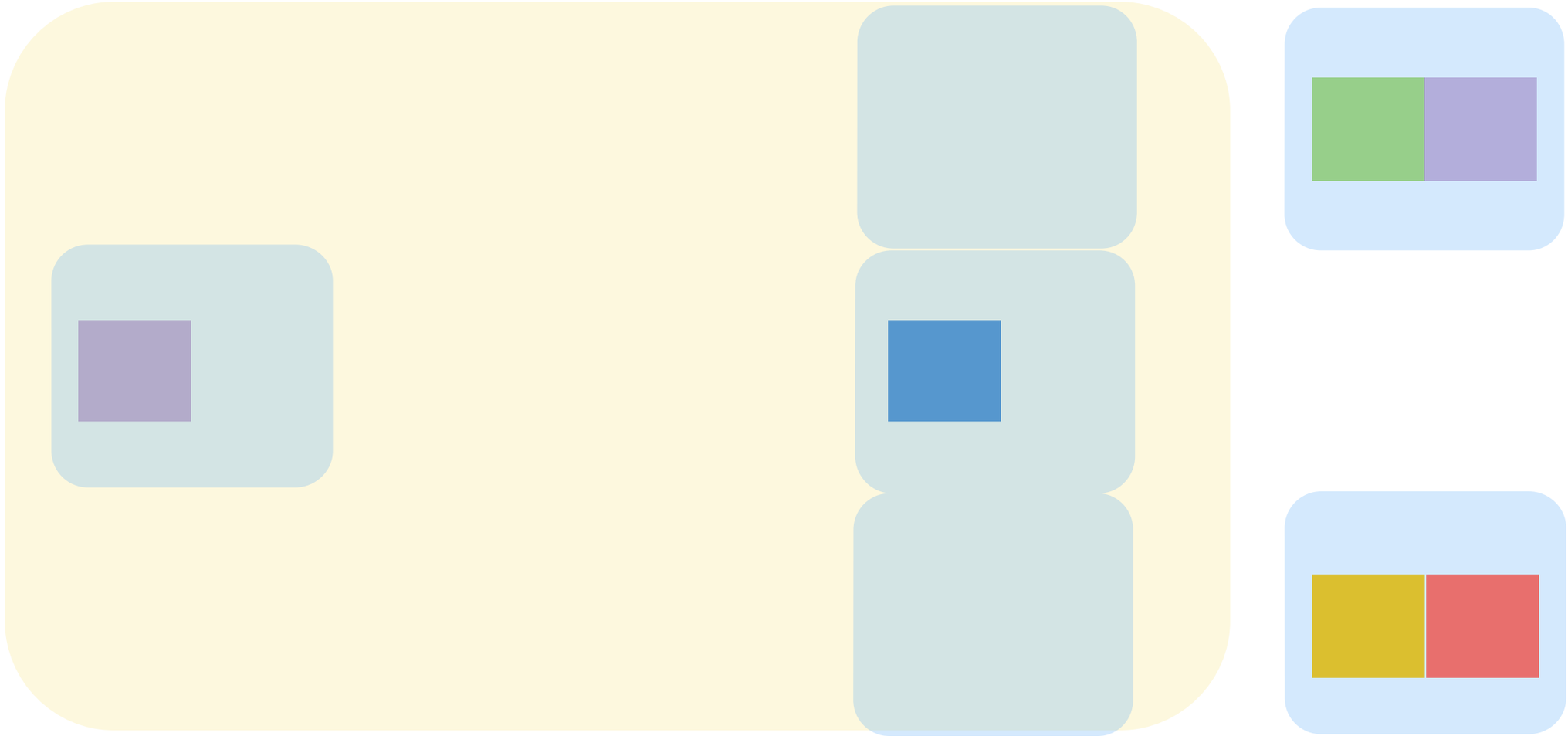


Pass 1: Divide

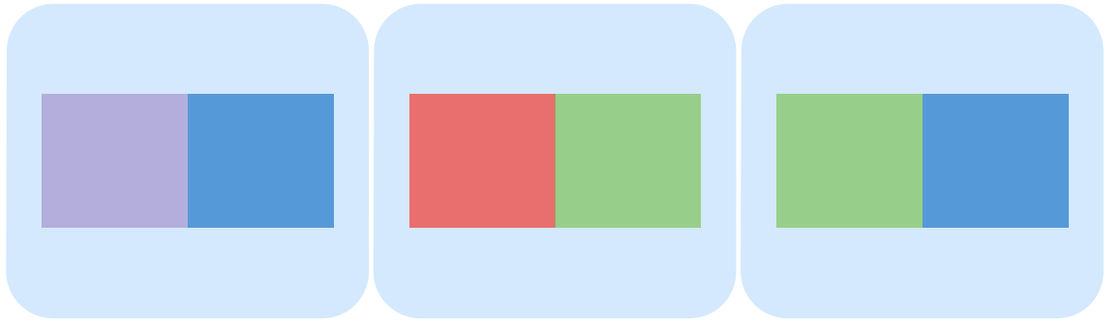


N=6, B=4

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

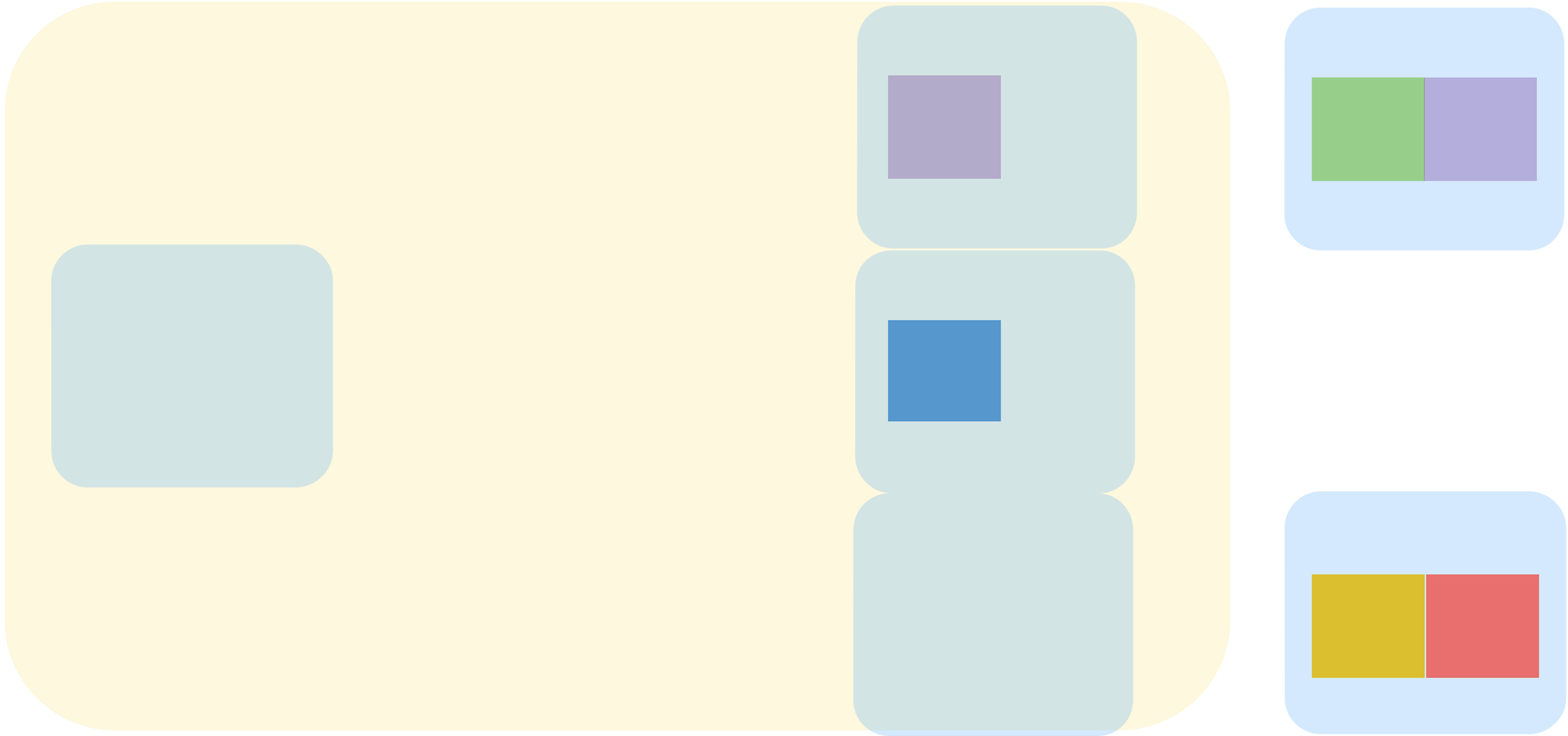


Pass 1: Divide

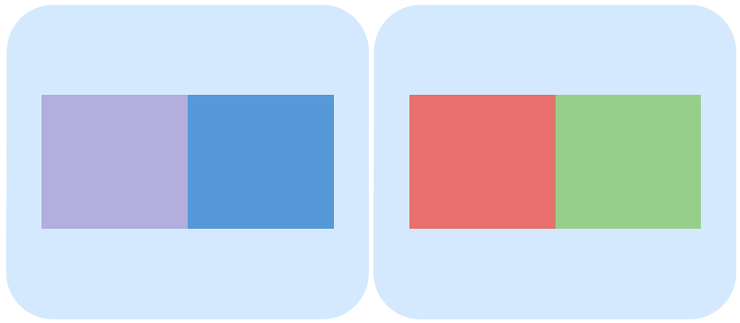


N=6, B=4

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

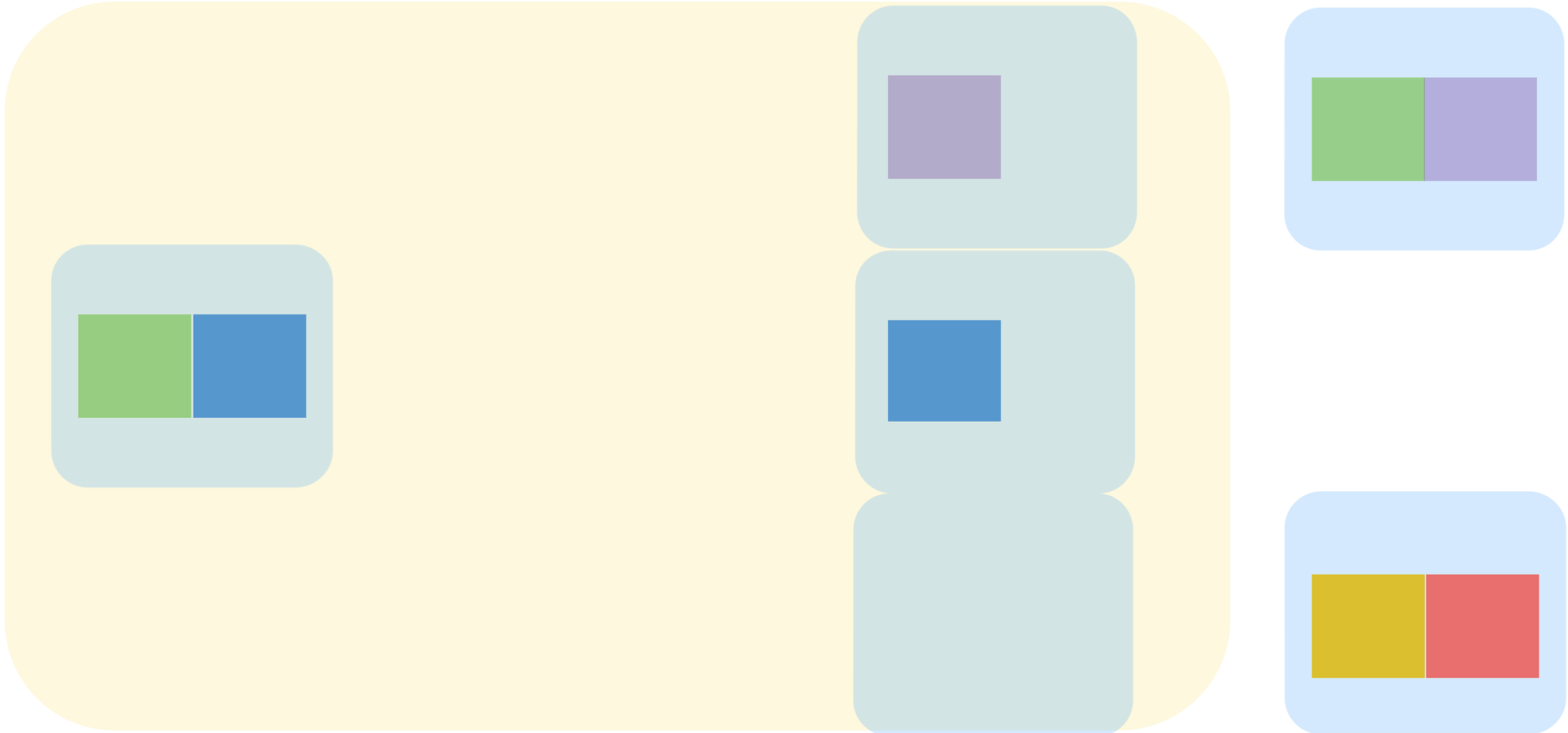


Pass 1: Divide

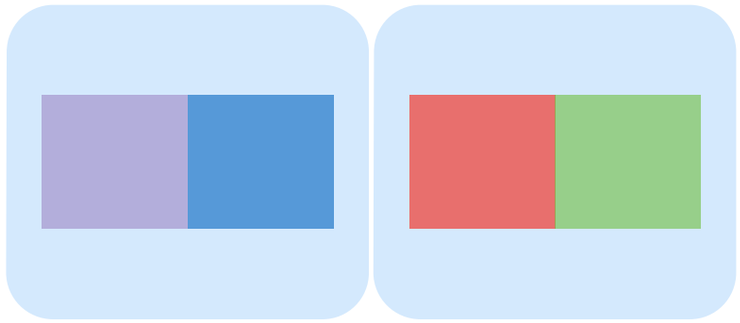


N=6, B=4

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

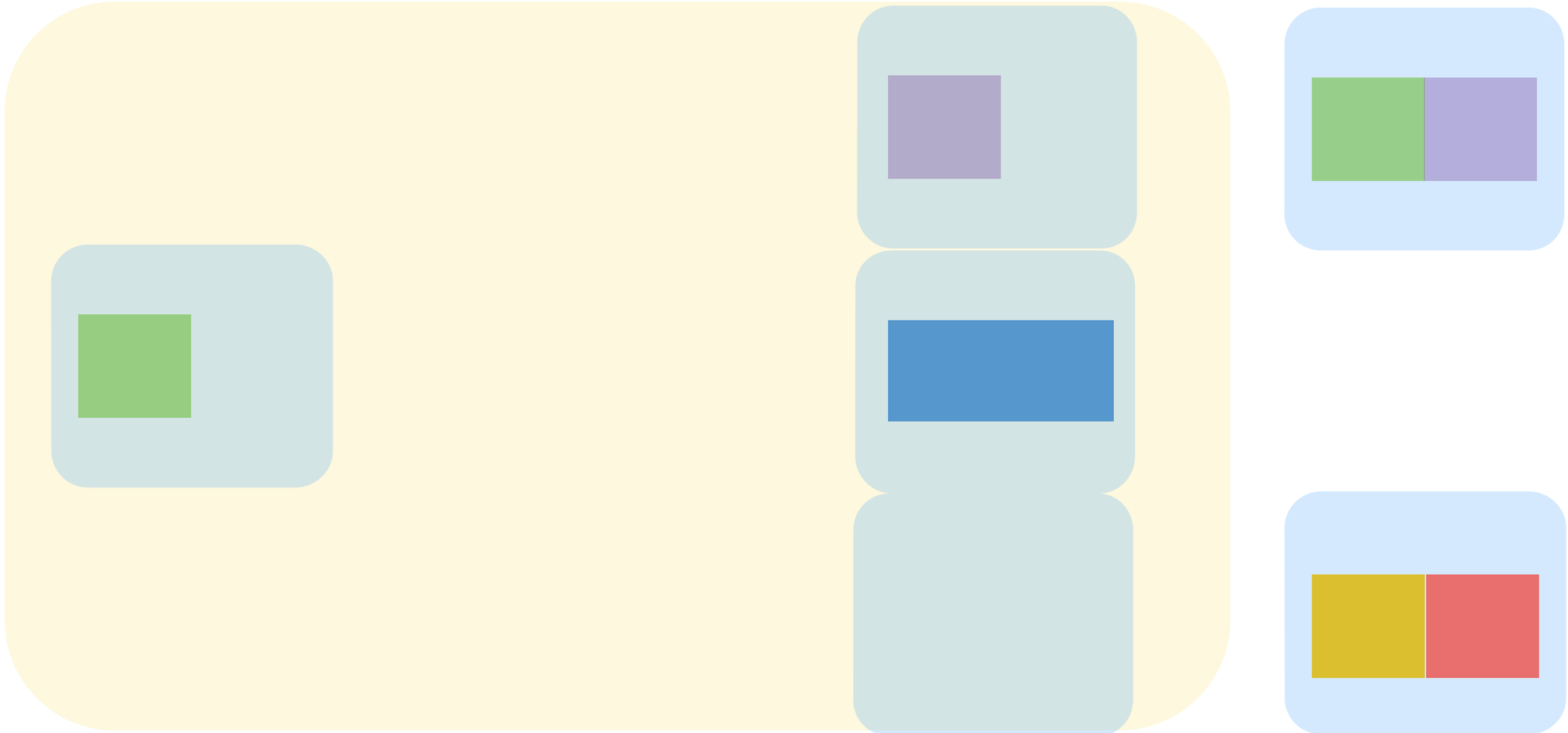


Pass 1: Divide

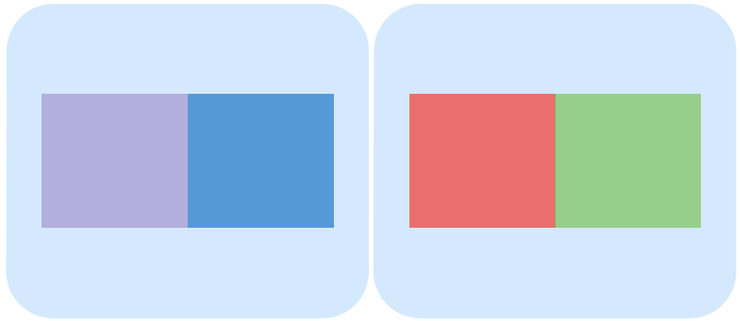


N=6, B=4

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

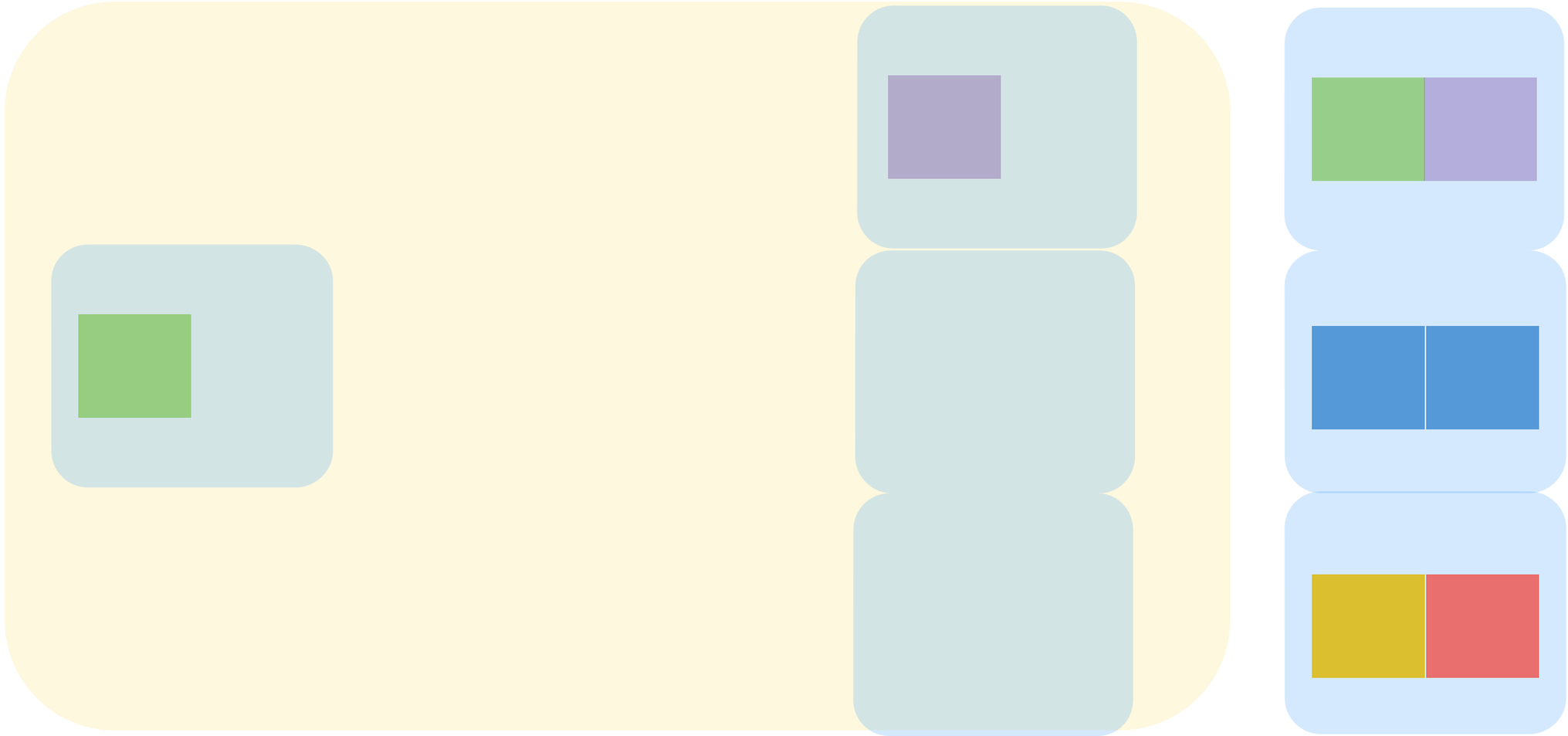


Pass 1: Divide

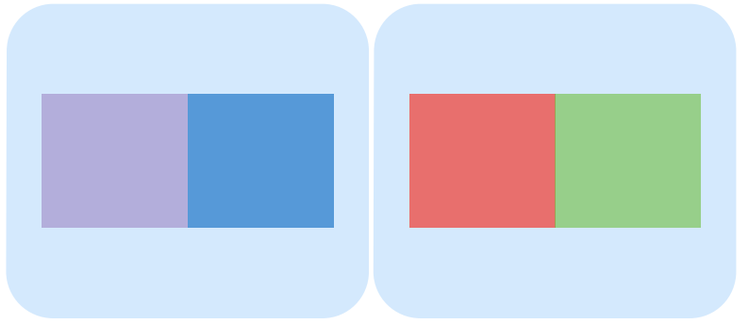


N=6, B=4

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

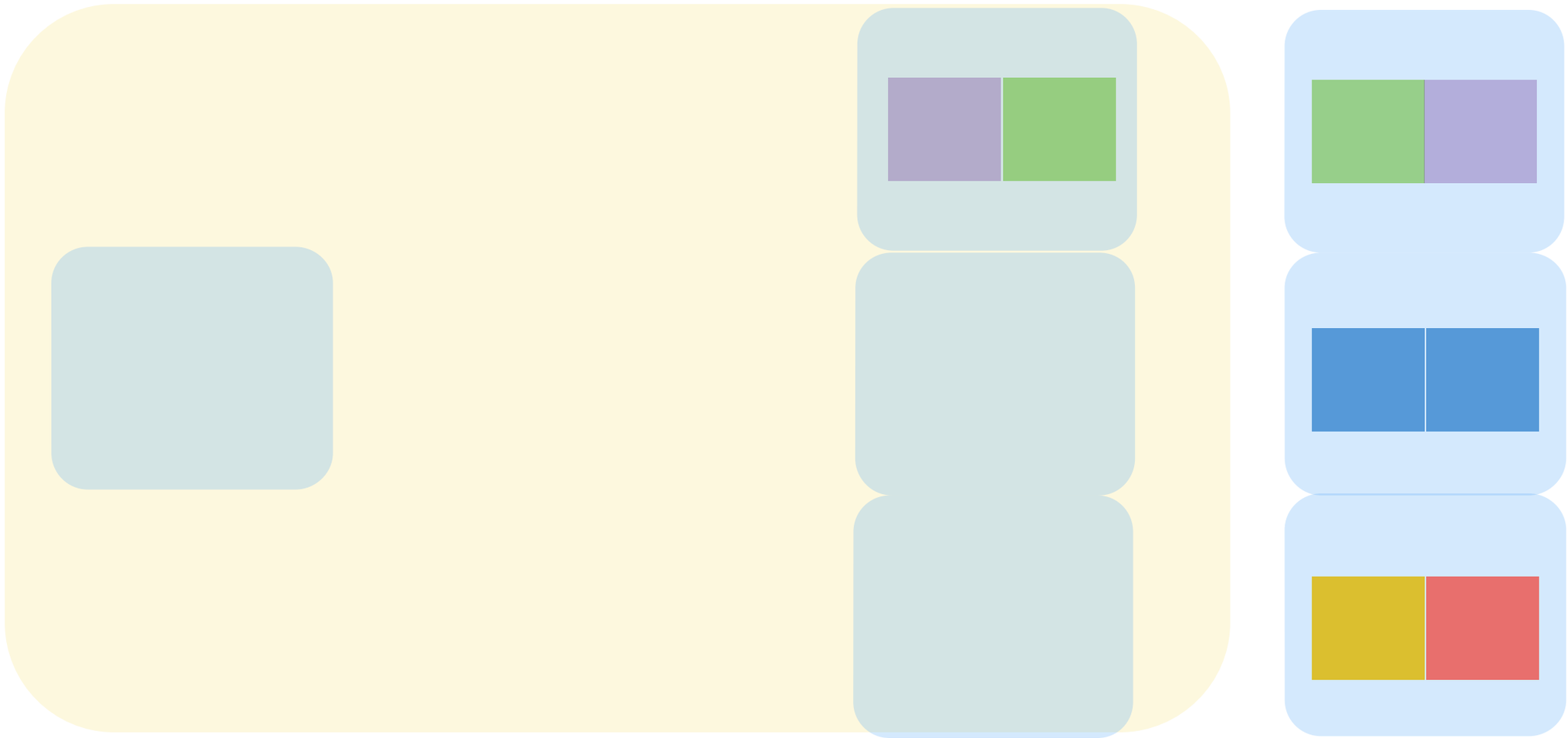


Pass 1: Divide

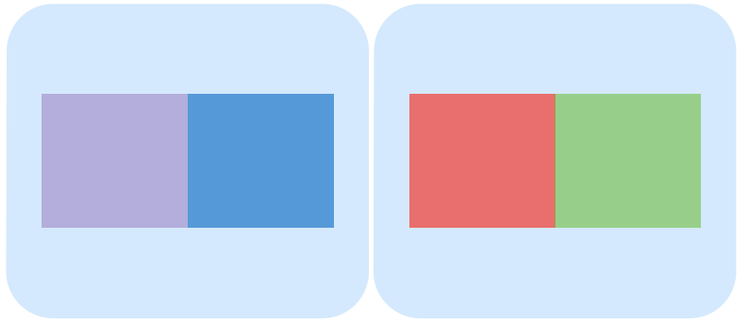


N=6, B=4

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

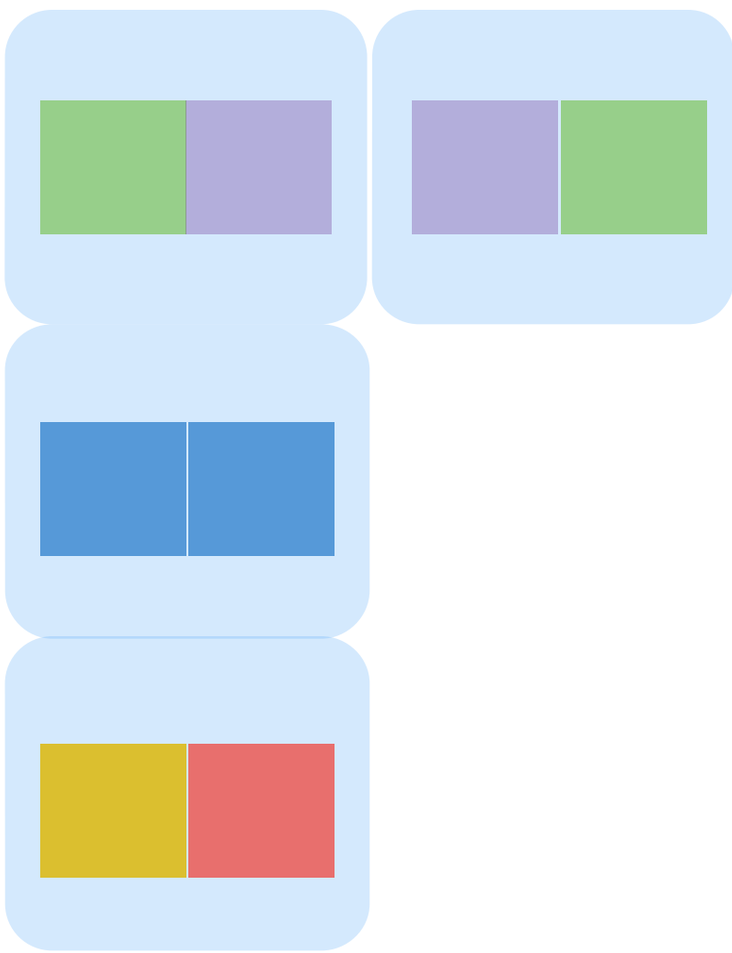


Pass 1: Divide

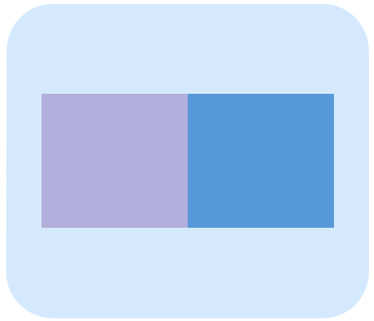


N=6, B=4

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

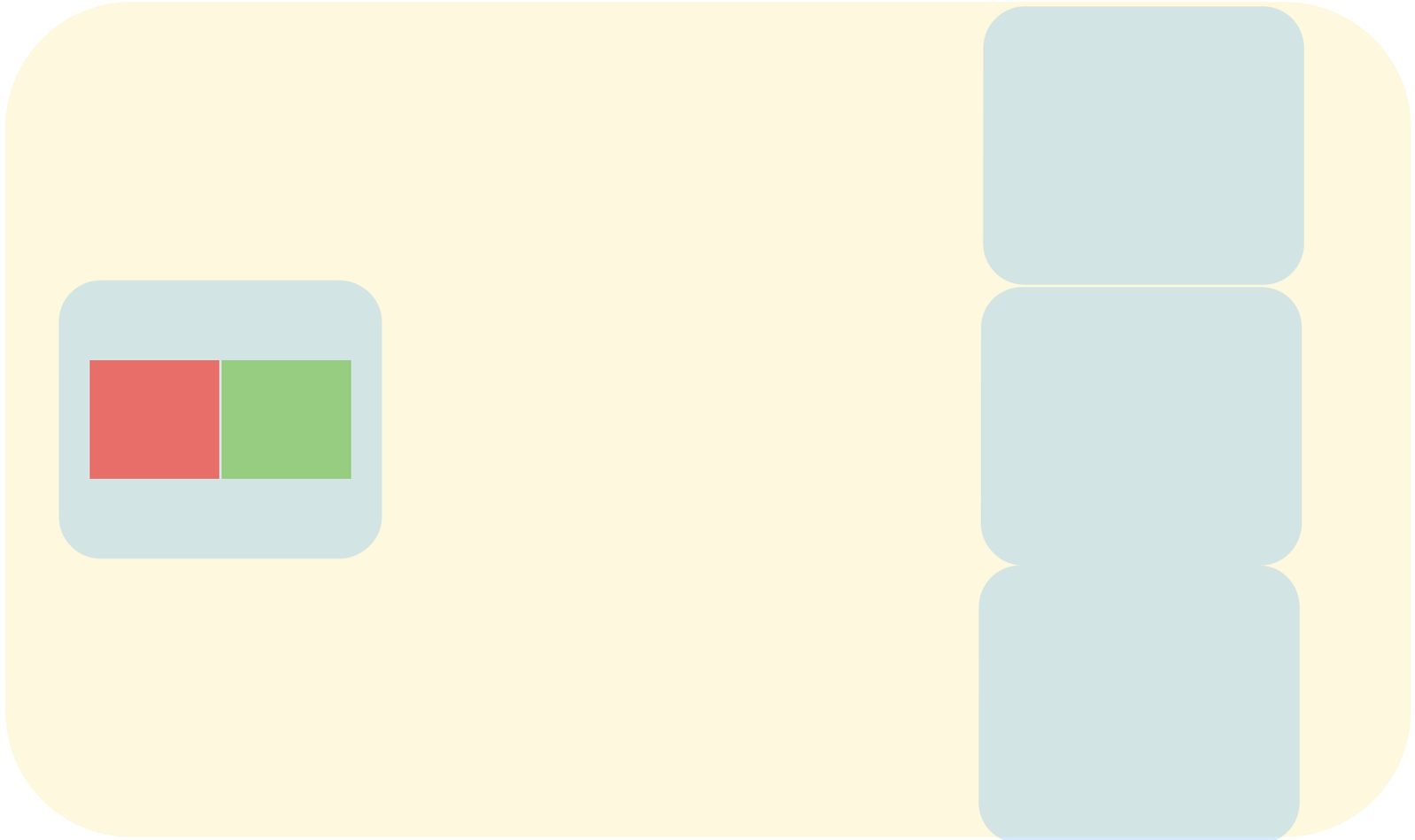


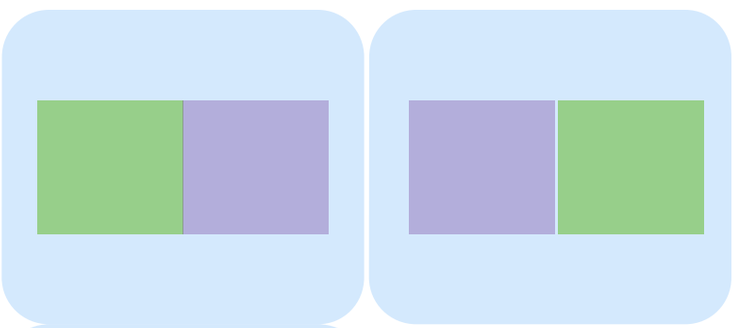
Pass 1: Divide

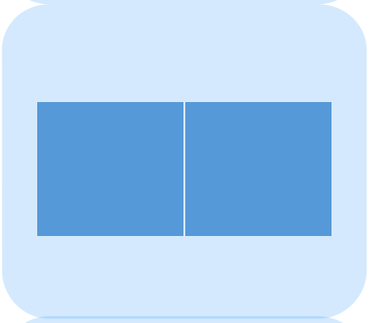


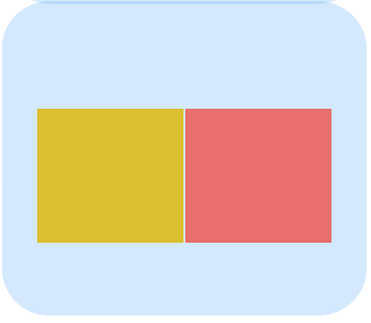
N=6, B=4

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

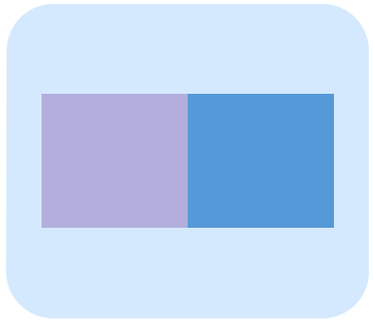






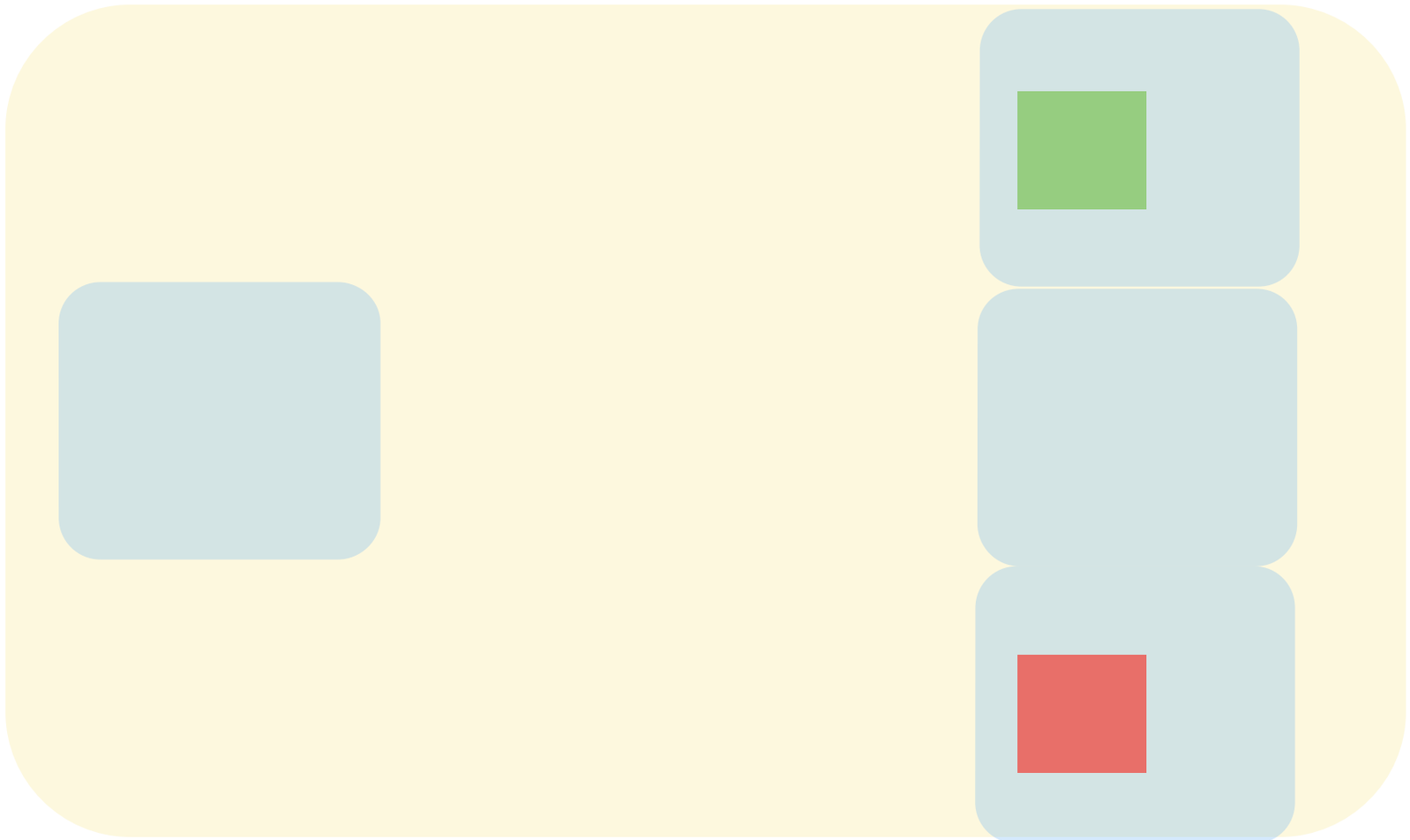


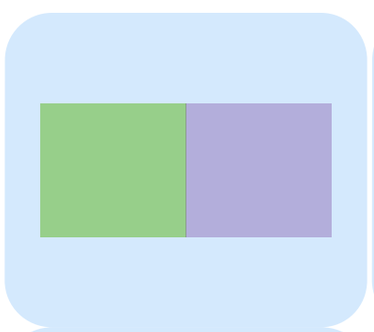
Pass 1: Divide

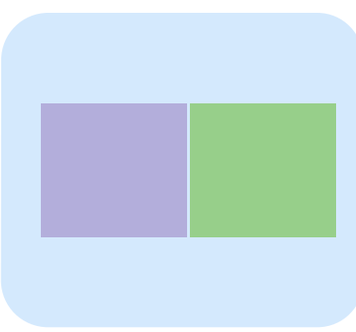


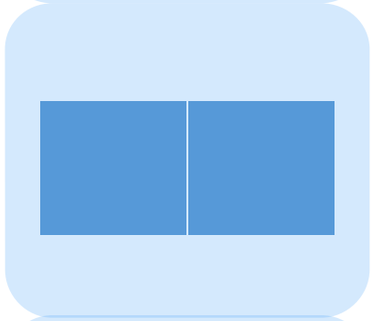
N=6, B=4

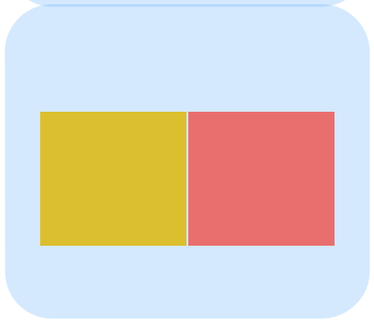
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3







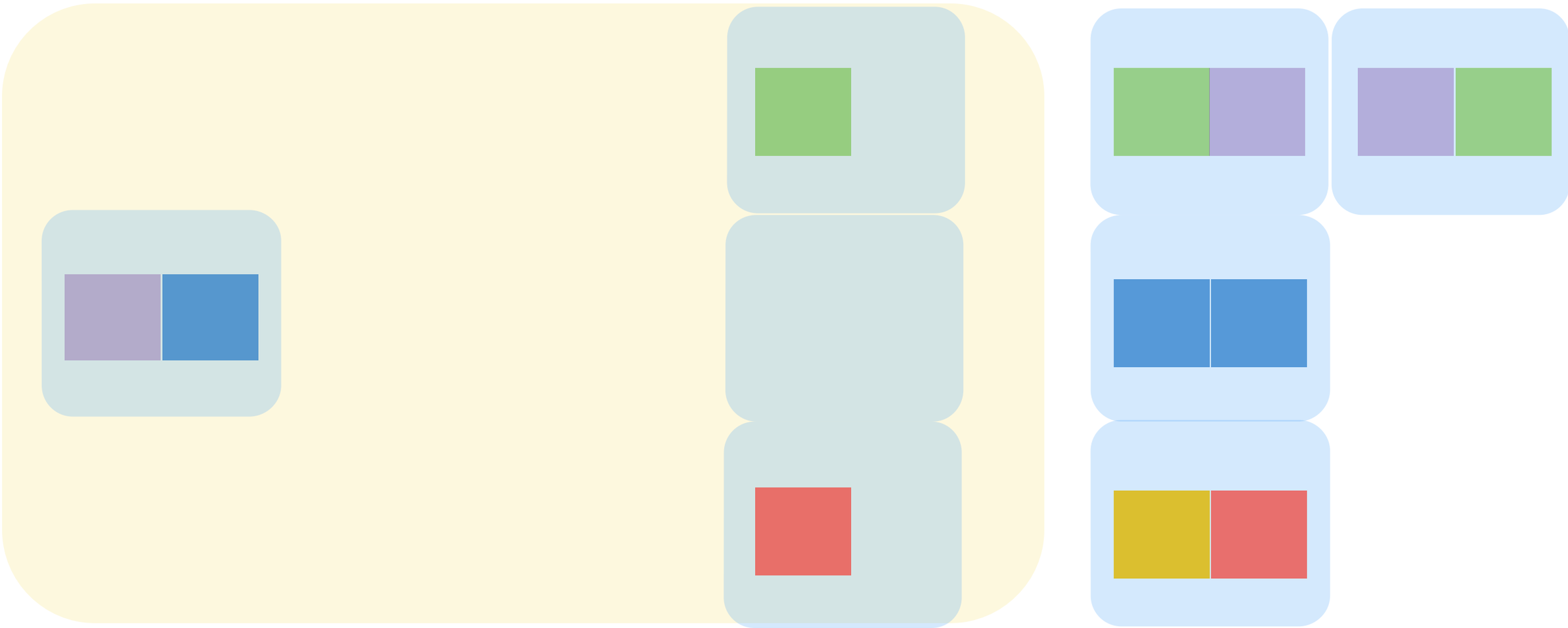




Pass 1: Divide

N=6, B=4

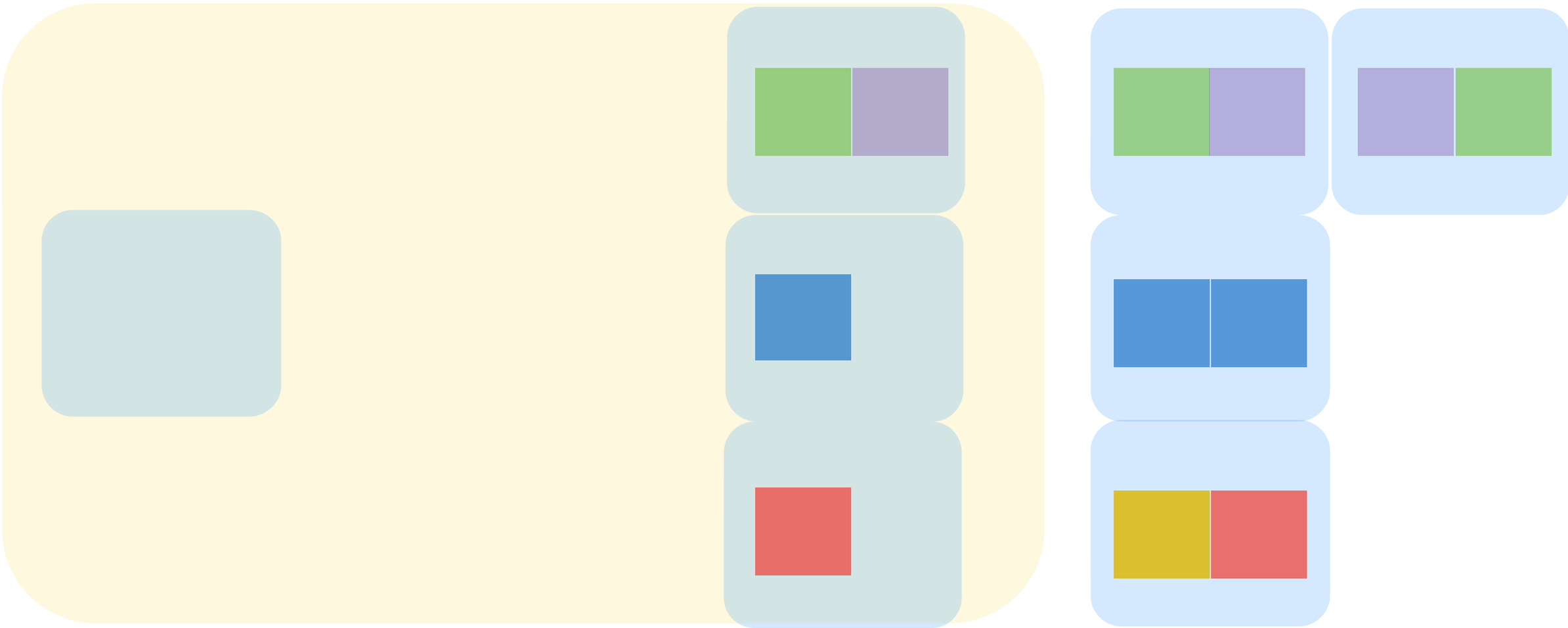
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



Pass 1: Divide

N=6, B=4

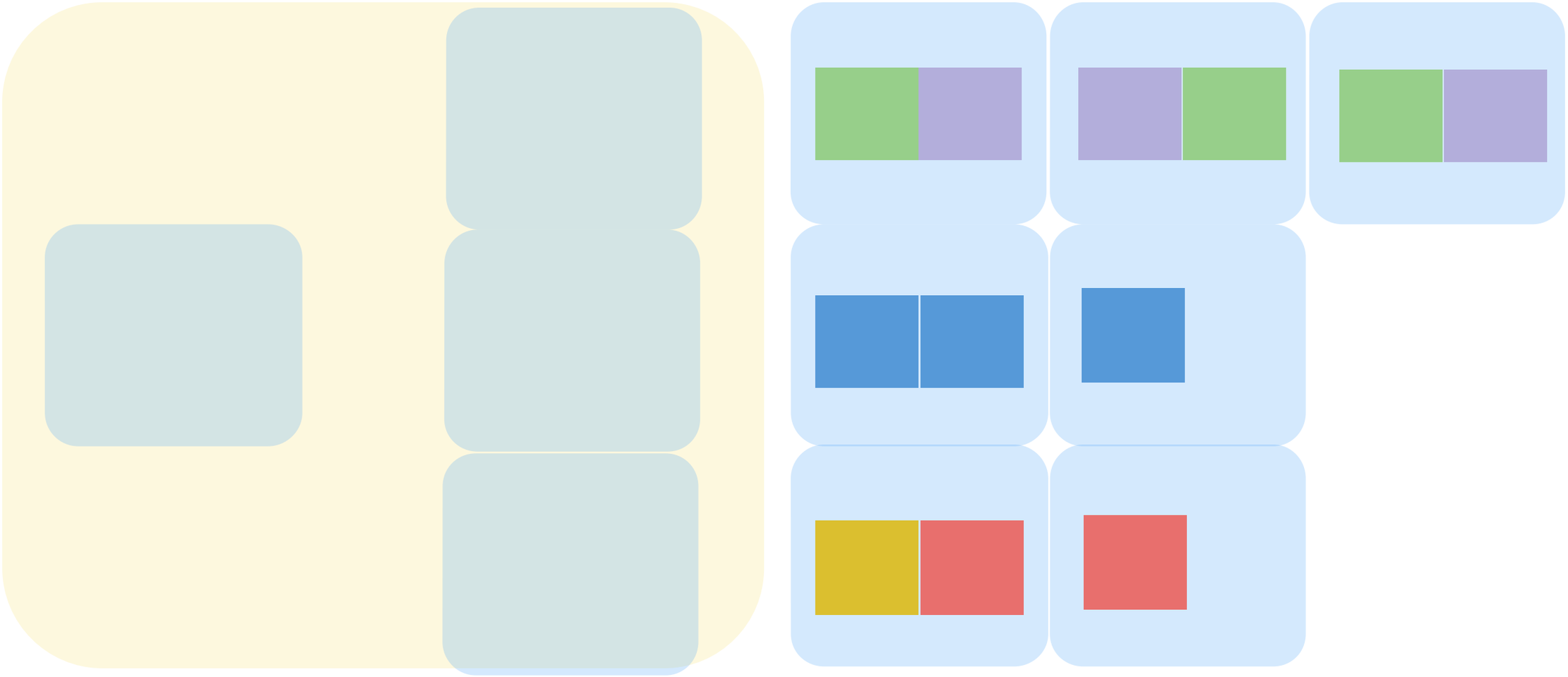
Our hash function: {G,P} -> 1, {B} -> 2,{R, Y} -> 3



Pass 1: Divide

N=6, B=4

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



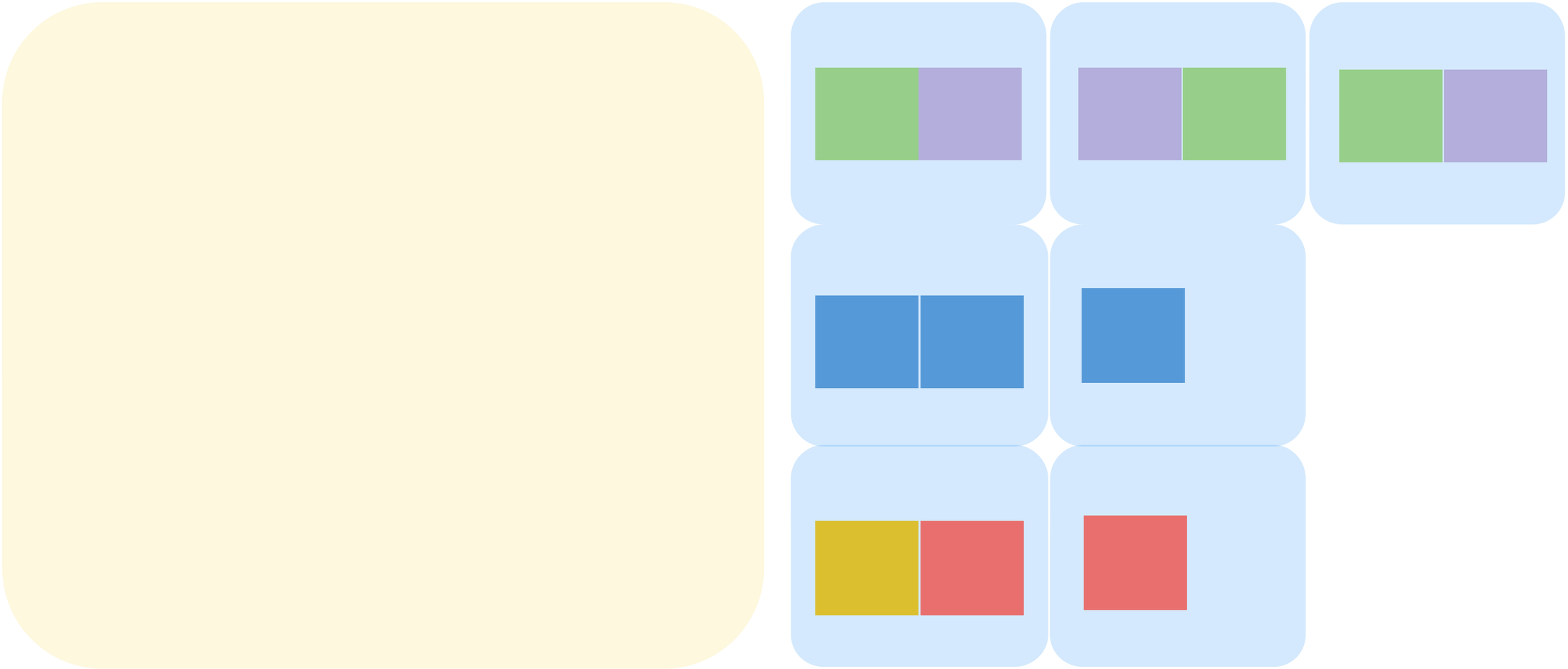
Pass 2: Conquer

- 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

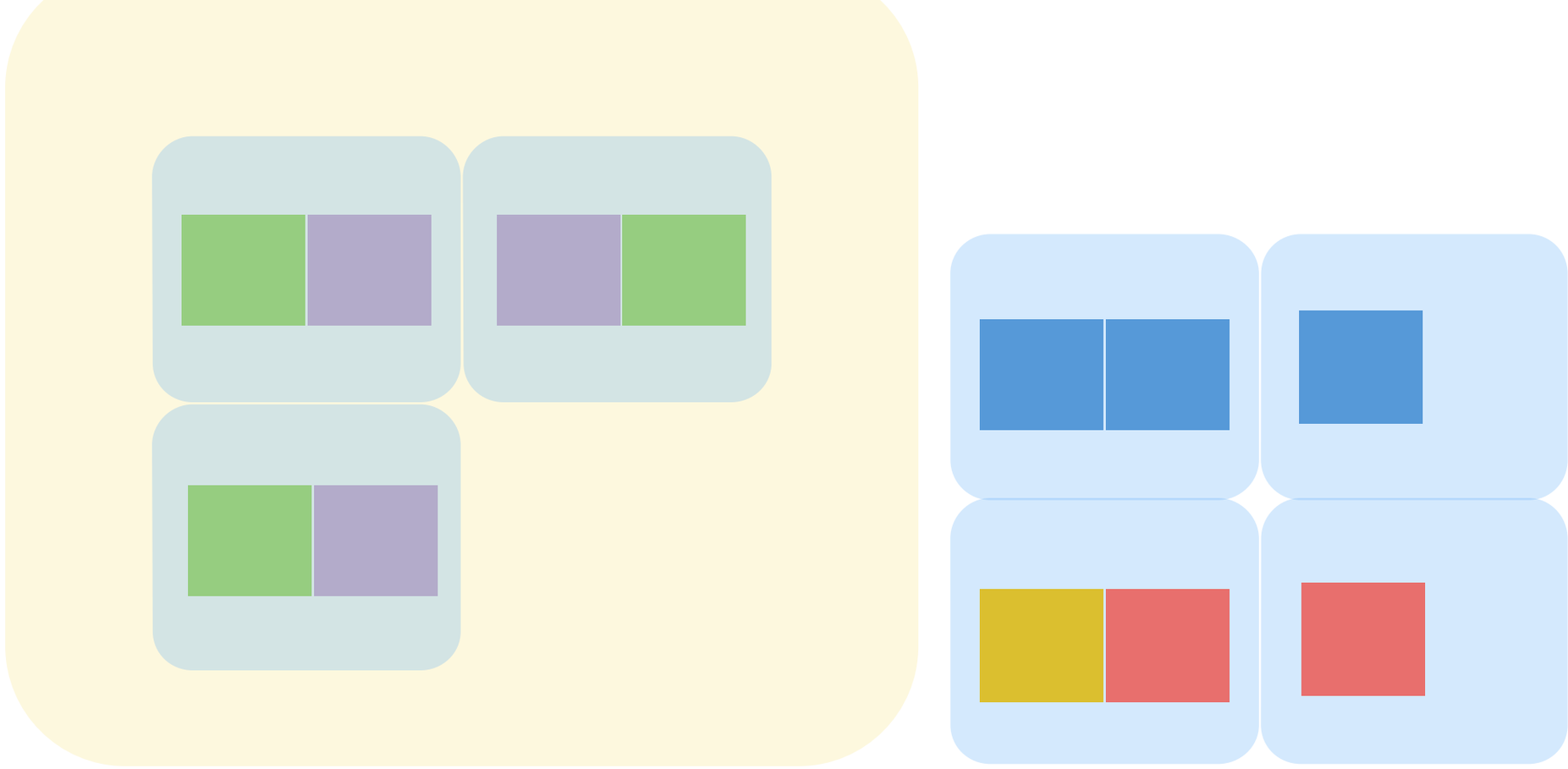


Pass 2: Conquer

Create in-memory table for each partition.

Assume there's another hash function than the one before

N=6, B=4



Pass 2: Conquer

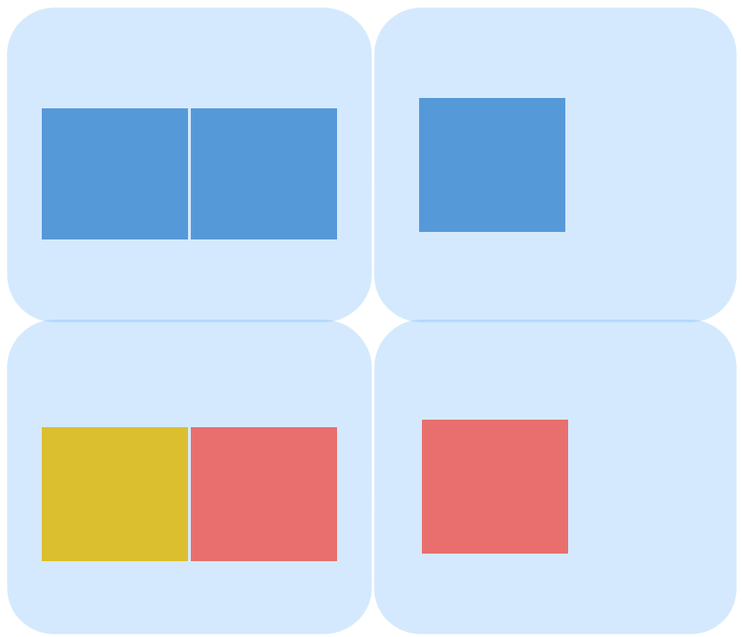
Create in-memory table for each partition.

Assume there's another hash function than the one before

N=6, B=4

Green

Purple



What happens when a partition is too big to fit in memory?

RECURSIVE PARTITIONING!!

"Recursive partitioning is used to **reduce the partition size** so they fit in memory. So we should be fine just rehashing just the partitions that are larger than memory, as long as **our hash functions are increasingly finer-grained** (i.e. if h1 maps A and B to separate partitions, then no later hash function will map them to the same partition).

If the previous condition doesn't hold, then we will have equal elements in two different partitions and the hash table we build in the final re-hashing phase won't be consistent."

Cost of External Hashing

- **Number of I/Os** = 4N (for 2 pass external hashing)

- Gotta use what you know about the partitions and how they're divided, passed into RAM, etc. to calculate the cost!

- For example, what happens if a partition doesn't fit in memory? Then you have to account for the fact that the partition will be divided and conquered again

External Hashing: Memory Requirement

- How big of a table can we hash in 2 passes?
    - B-1 partitions result from Pass 1
        - Each should be no more than B pages in size
- **Answer: B(B-1)**, assuming that the hash function distributes records evenly, we can hash a table of size N pages in about $\sqrt{N}$ space.

Sorting vs. Hashing

- Great if input is already sorted
- Great if output needs to be sorted anyway
- Not sensitive to "data skew" or "bad" hash functions
- Overkill for rendezvous

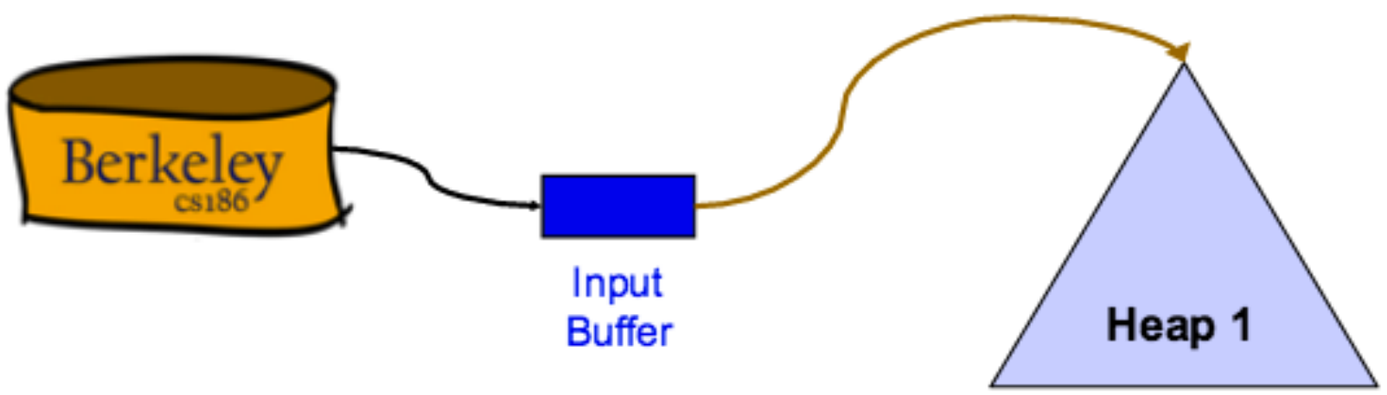
- For duplicate elimination, scales with # of values (not items)
- Can simply conquer sometimes

Do questions #1, #2, #5, #6 and #7

Aside: Tournament Sort

- Used to gather the initial runs for external sorting algorithms (source: Wikipedia)
- Quicksort is faster, but longer runs often means fewer passes
- Important thing to know: average length of a run is 2(B-2)

First, load a heap with B-2 pages of records



This is a priority heap, so it is in sorted order (hence called Heapsort)

Aside: Tournament Sort

```
while (records left on input) {
        m = H1.removemin(); // get smallest value
        Output(m)           // put m in output buffer;
        if (H1 NOT empty)
            r = InputRecord()
            if (r < m)   H2.insert(r);
            else         H1.insert(r);




    else
        H1 = H2;  H2.reset();
        start new output run;
}
```