CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Practical Implementation of Distributed CARMA: Communication-Optimal MM Algorithm

Author: Marko Kabic
marko.kabic@cscs.ch

Mentors: Dr. Joost VandeVondele
Dr. Raffaele Solcà

April 1st, 2018

Motivation

- Yet another MM algorithm?

Communication-optimal

Motivation

- Yet another MM algorithm?

Communication-optimal

IF

- Specific matrix dimensions (like square matrices)
- Certain memory ranges

Motivation

- Yet another MM algorithm?

Communication-optimal

IF

- Specific matrix dimensions (like square matrices)
- Certain memory ranges

Motivation

- Yet another MM algorithm?

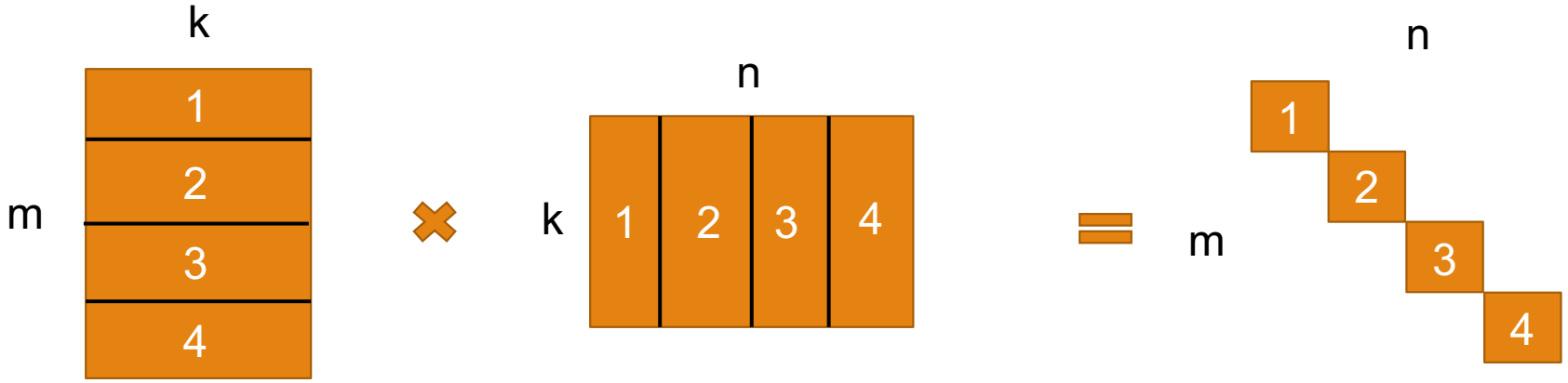
Communication-optimal

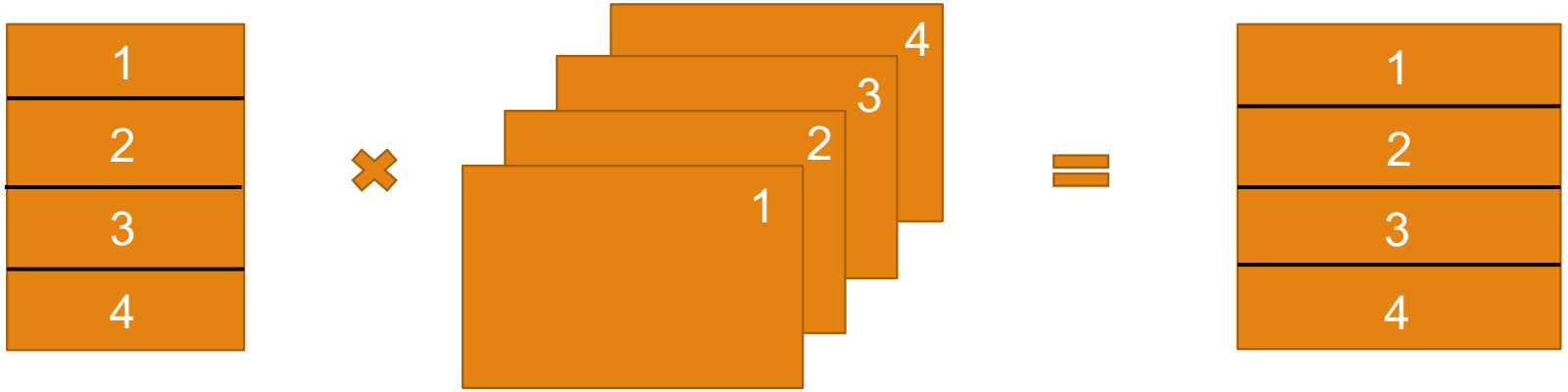
IF

- Specific matrix dimensions (like square matrices)
- Certain memory ranges

- CARMA:
  - **Communication Optimal:** for any dimensions and memory ranges.

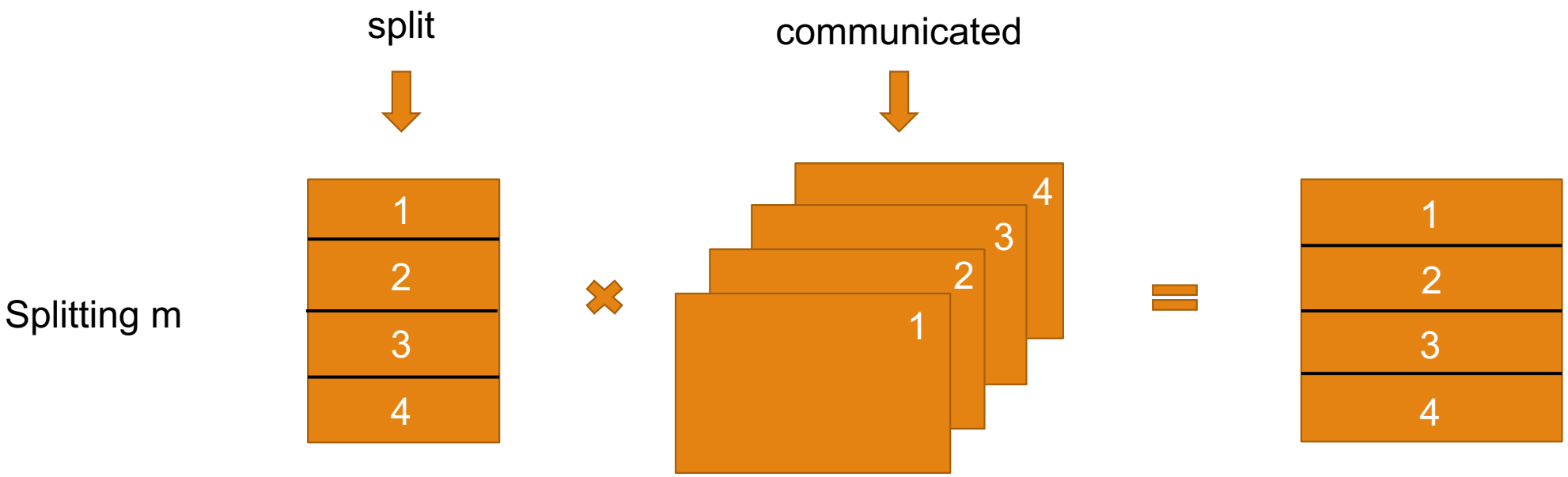
Is communication really necessary?





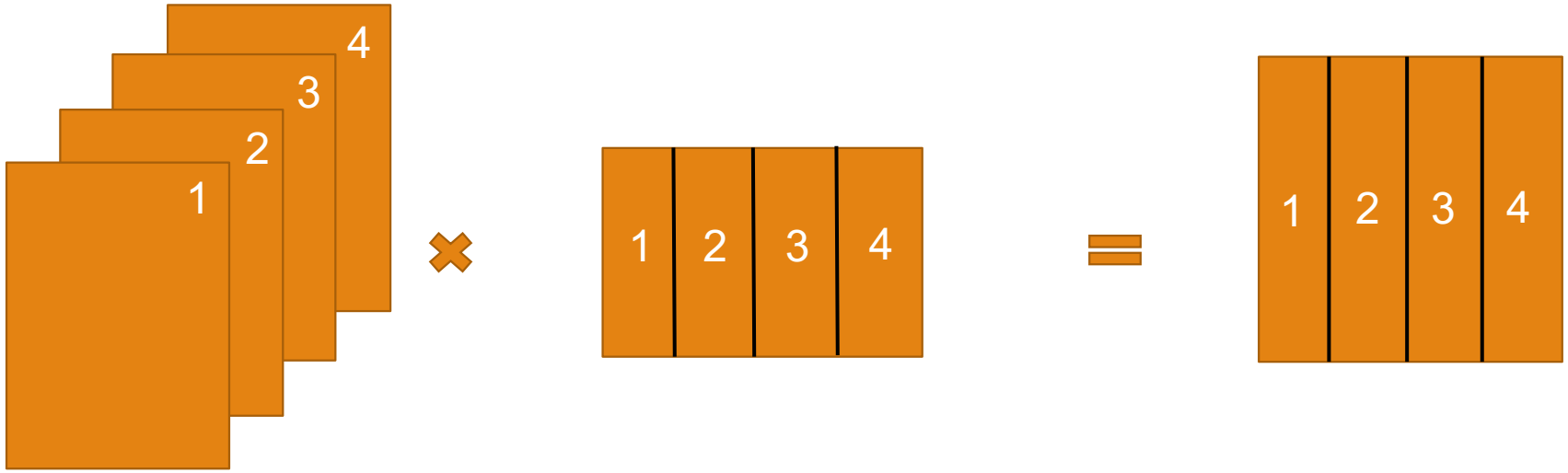
Is communication really necessary?

- Redundancy in data is necessary in order to parallelize MM
- We need communication to make redundancy
- If one matrix is split, the other one has to be communicated

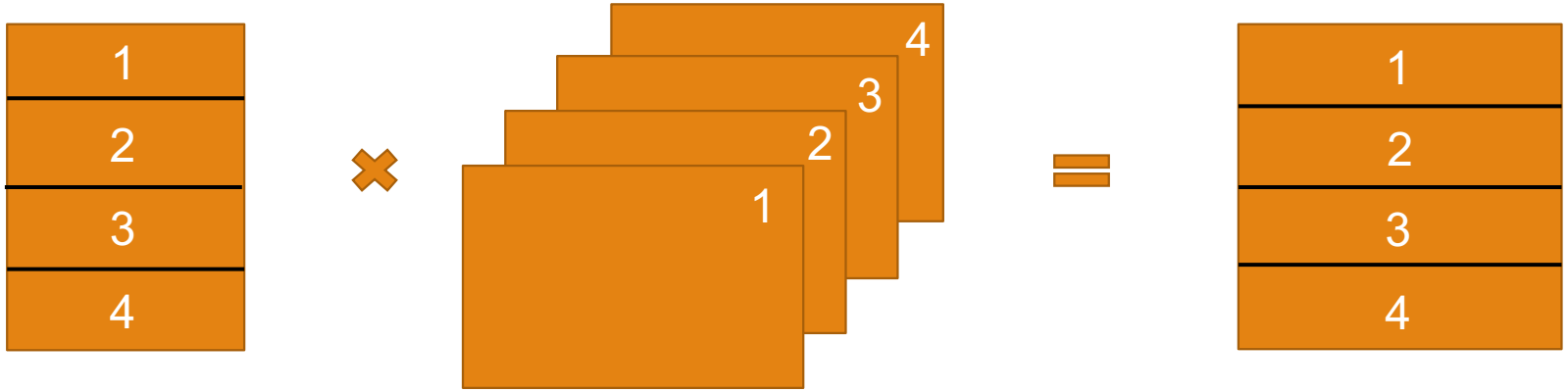


We can split any dimension

Splitting n

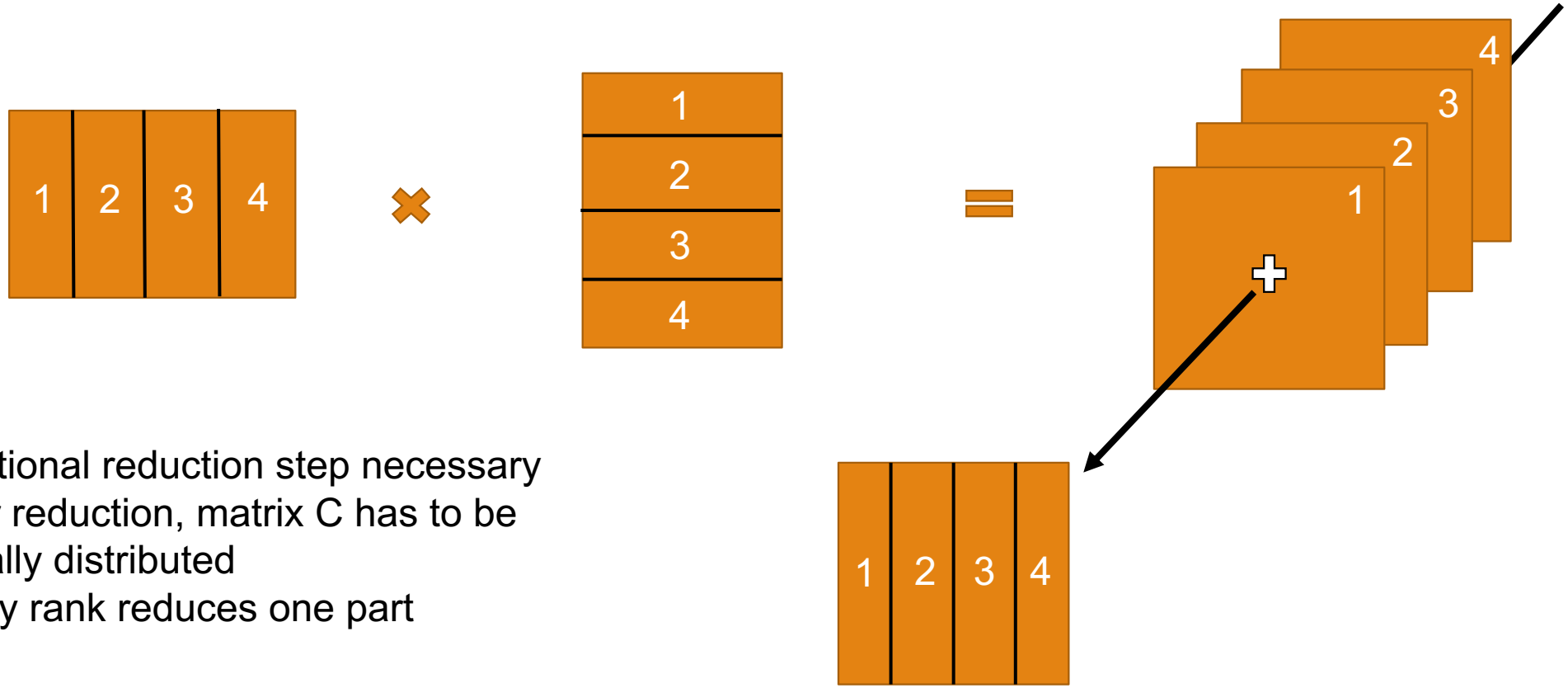


Splitting m



We can split any dimension

Splitting k

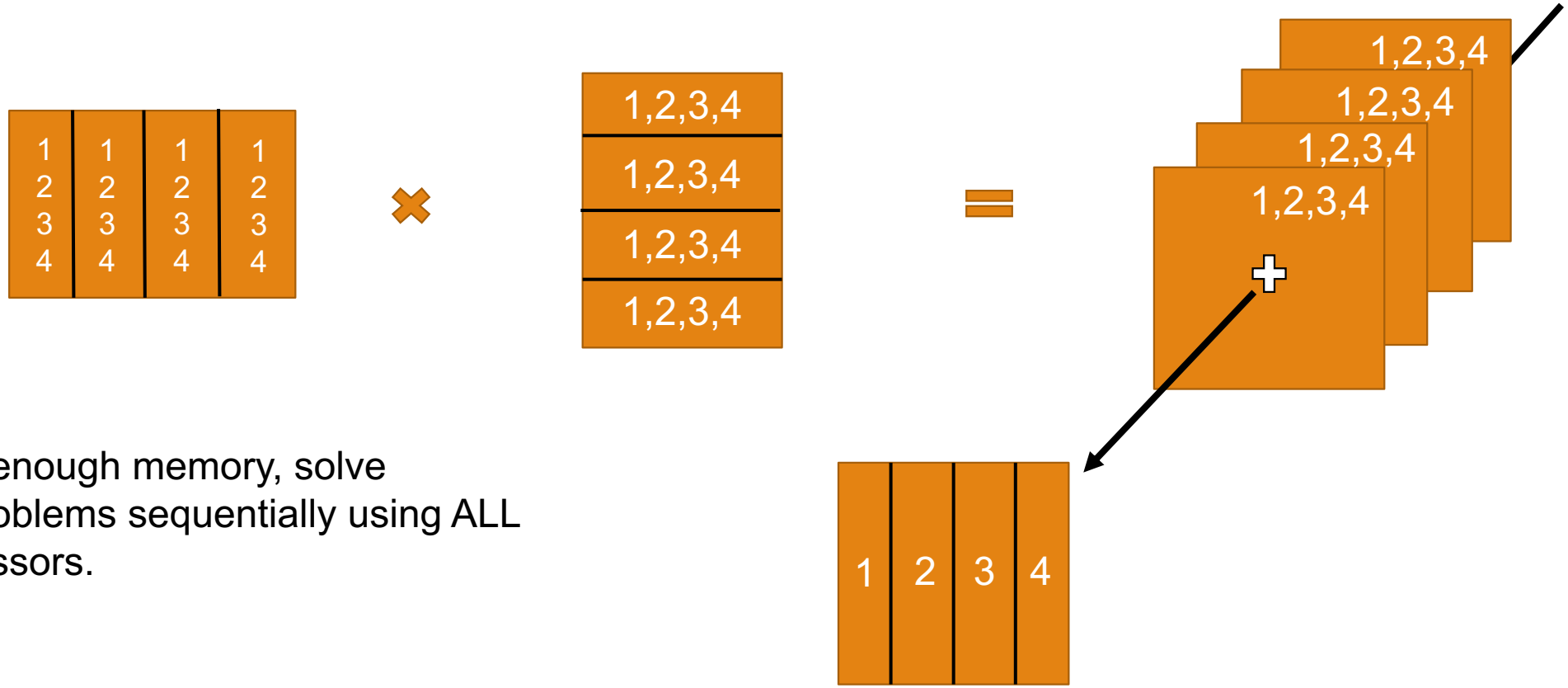


- Additional reduction step necessary
- After reduction, matrix C has to be equally distributed
- Every rank reduces one part

Parallelization requires a lot of additional memory!

If there is not enough memory - serialize

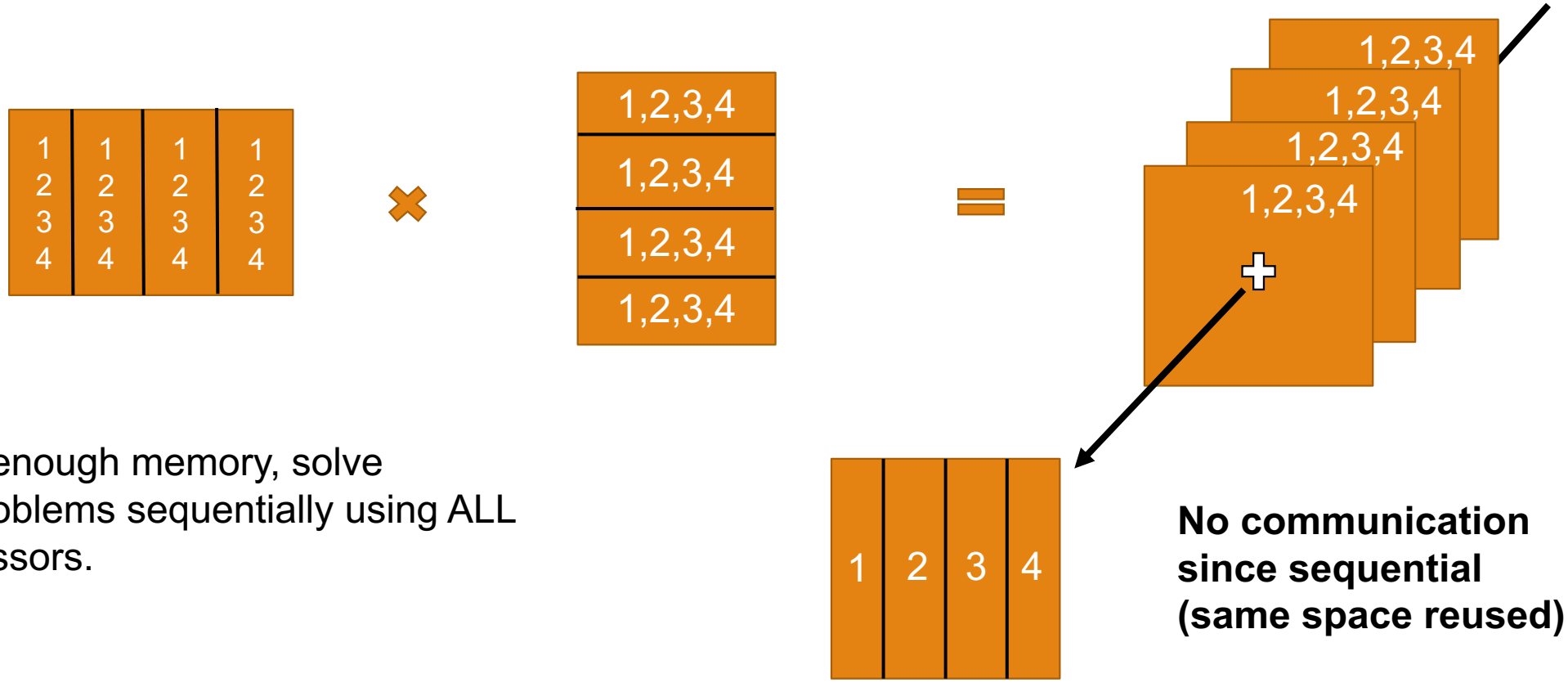
Splitting k



- If not enough memory, solve subproblems sequentially using ALL processors.

If there is not enough memory - serialize

Splitting k



- If not enough memory, solve subproblems sequentially using ALL processors.

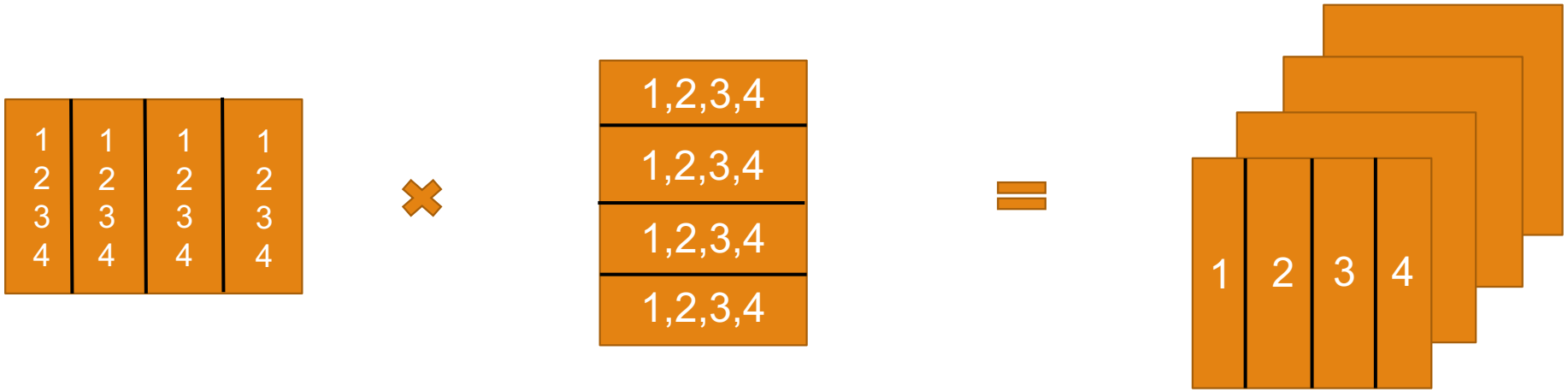
No communication since sequential (same space reused)!

No additional space required!

Reduction on the fly

We can split any dimension

Splitting k



- If not enough memory, solve subproblems sequentially using ALL processors.

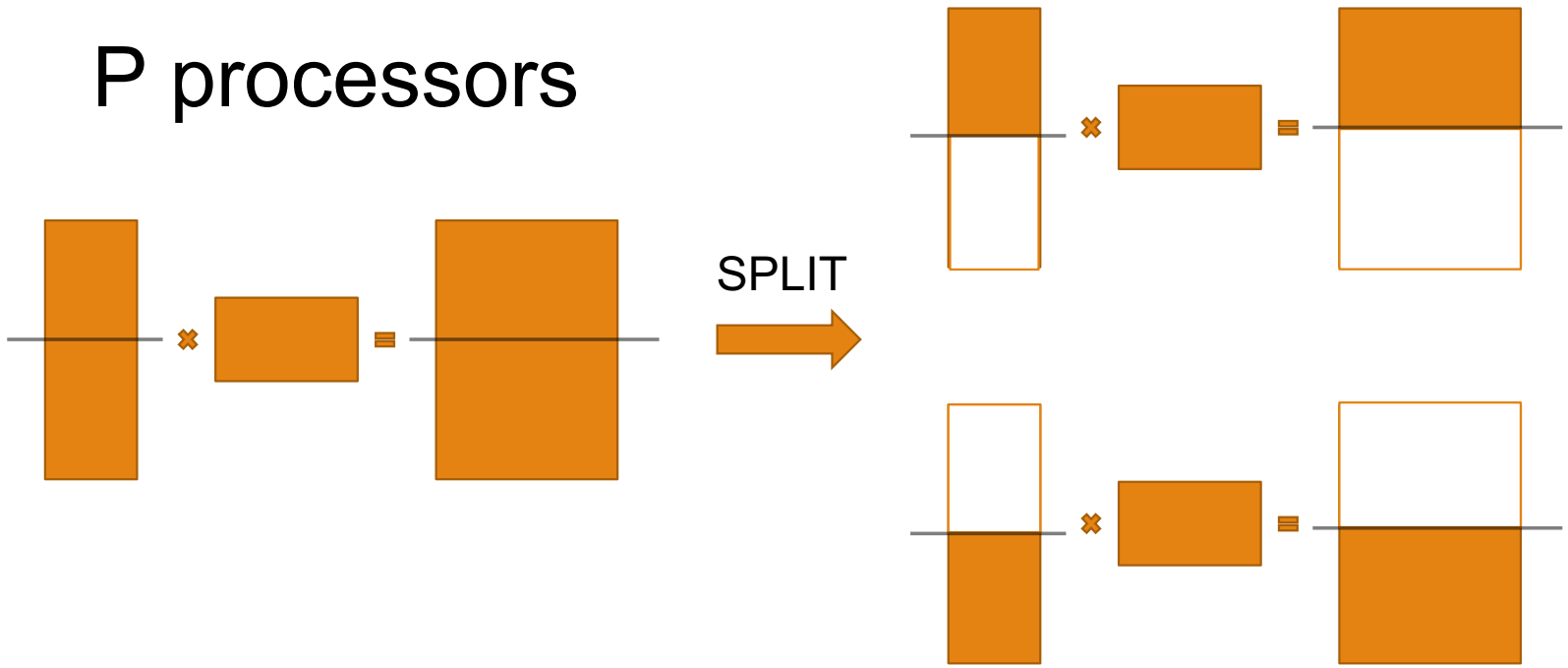
No communication since sequential (same space reused)!

No additional space required!

Reduction on the fly

CARMA

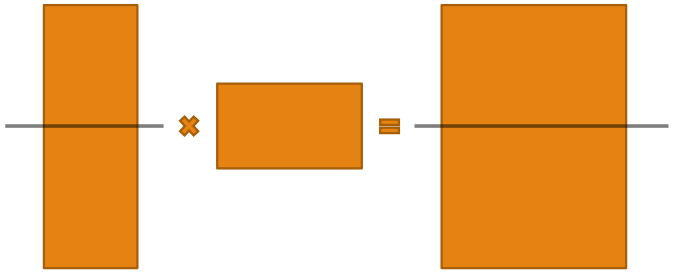
- Split the largest dimension
- Repeat recursively



How does CARMA work?

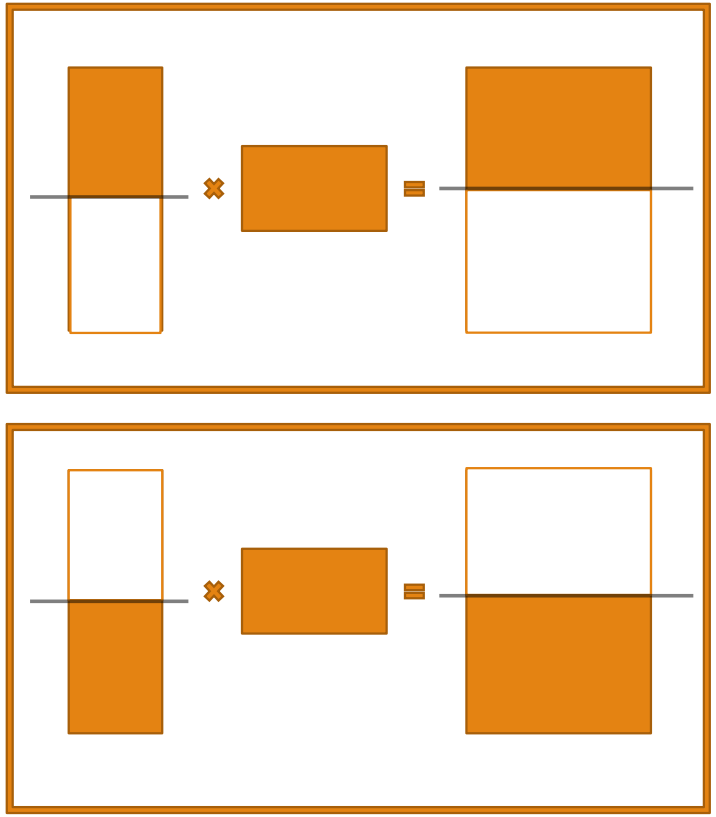
- Split the largest dimension
- Repeat recursively

P processors



SPLIT

P/2 processors



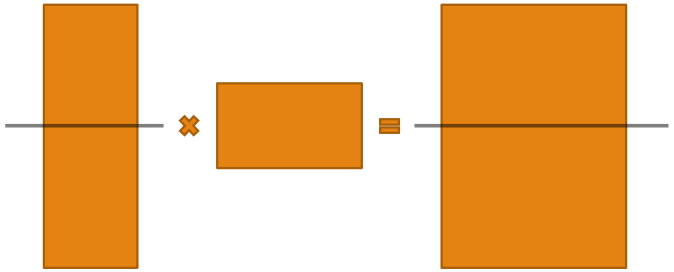
P/2 processors

- If enough memory:
  - Solve the subproblems in parallel

How does CARMA work?

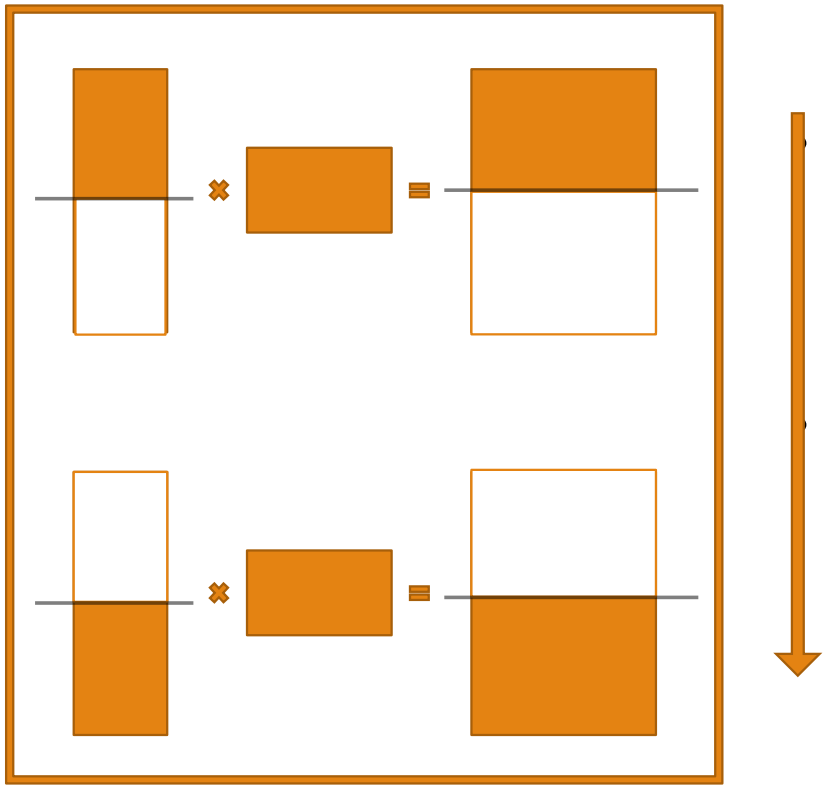
- Split the largest dimension
- Repeat recursively

P processors



SPLIT

P processors



P processors

If enough memory:

- Solve the subproblems in parallel

Otherwise

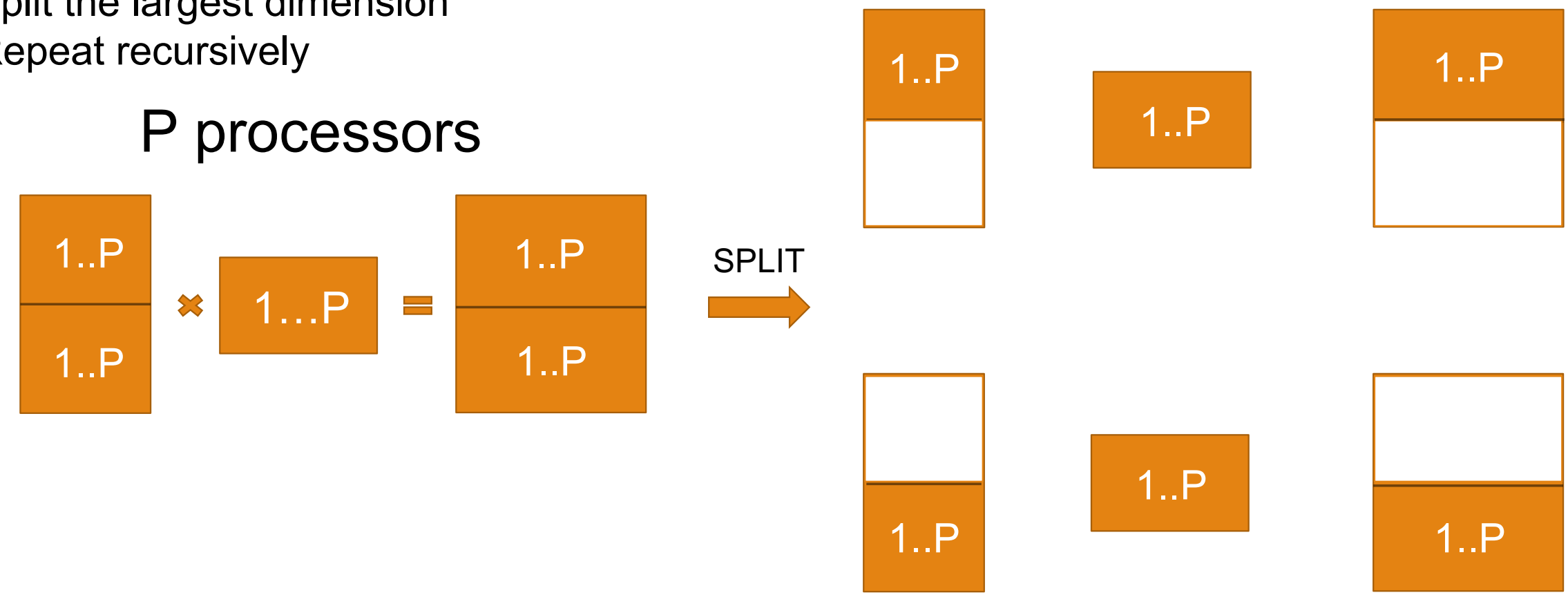
- Solve the subproblems sequentially

Example of Execution

Check out the file **example.pdf**!

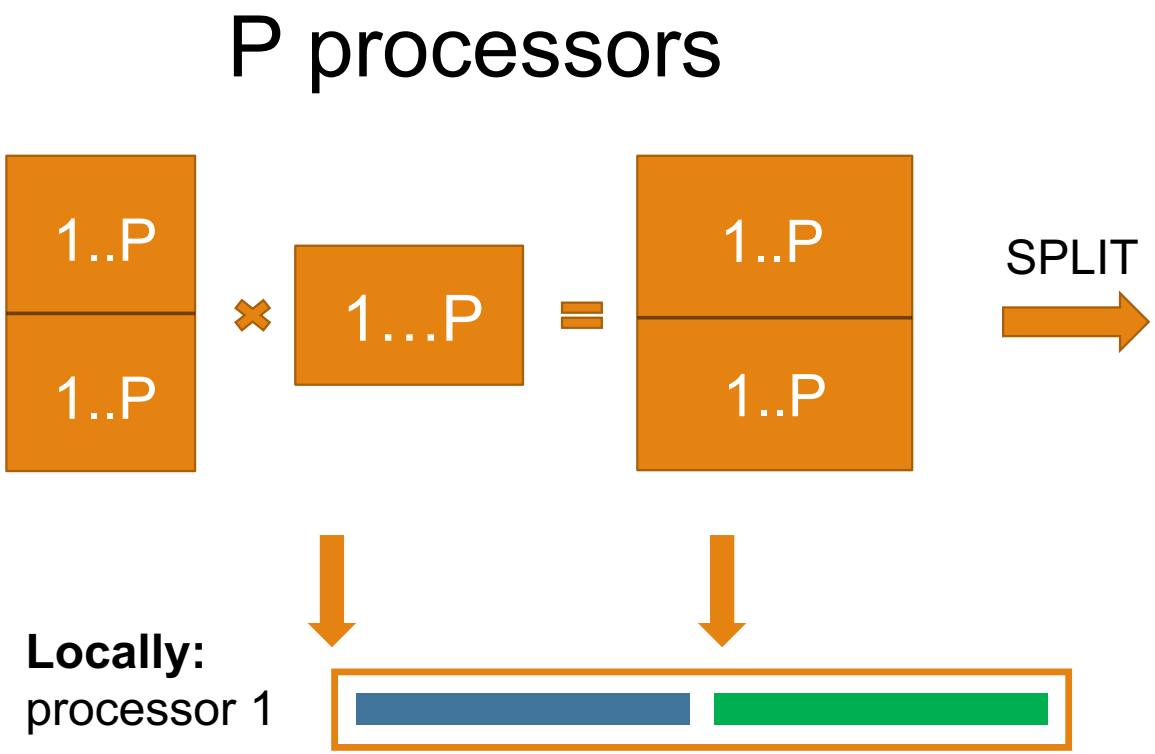
Sequential Step: Closer Look

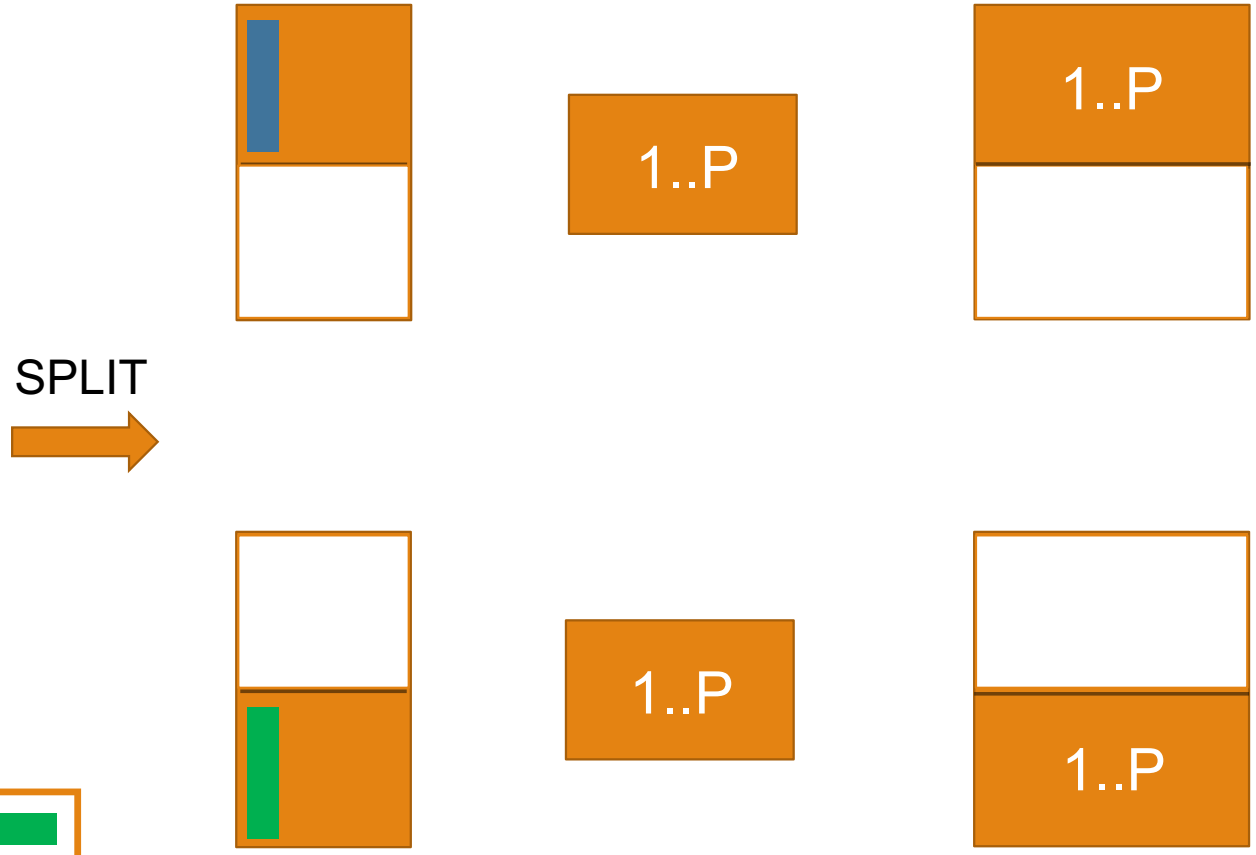
- Split the largest dimension
- Repeat recursively



Sequential Step: Closer Look

- Split the largest dimension
- Repeat recursively

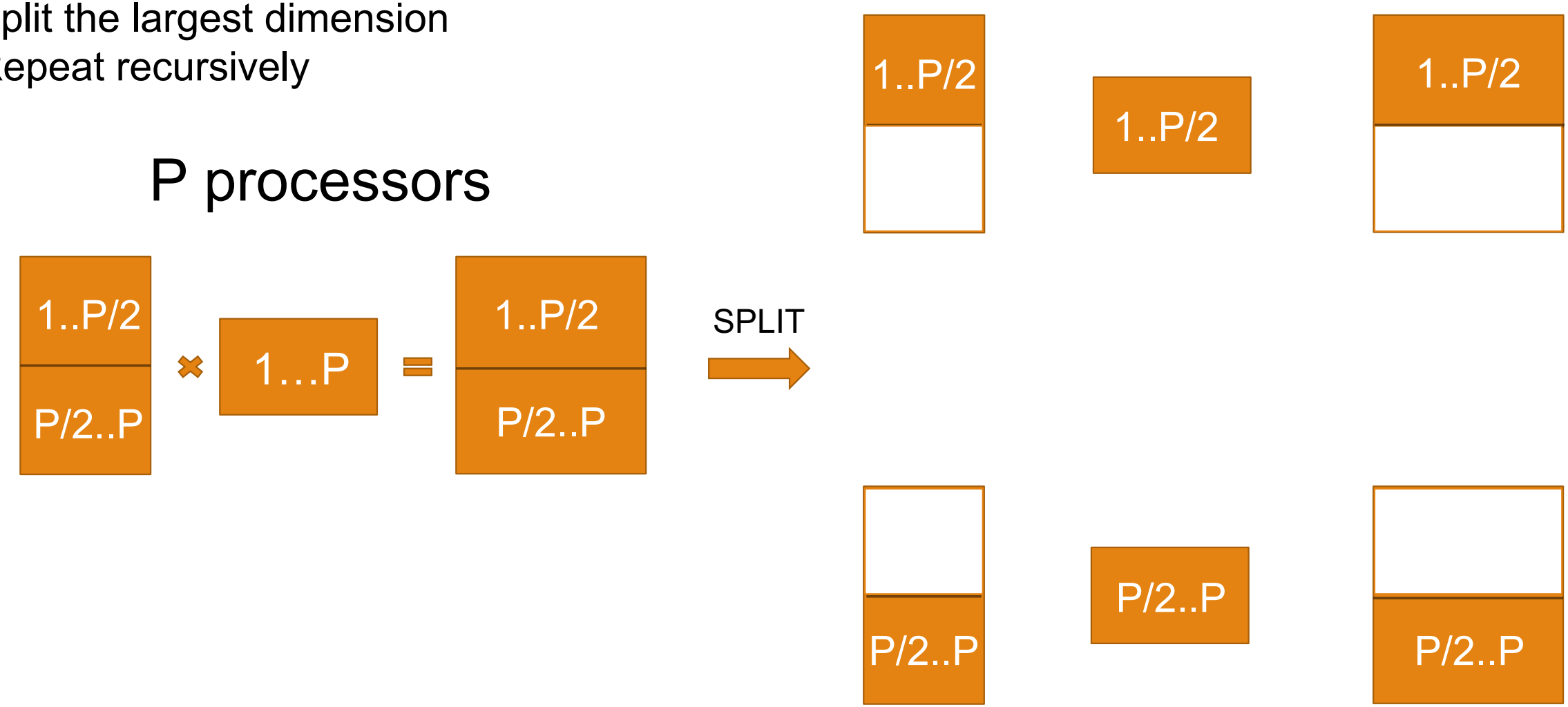




no communication, just moving pointers!

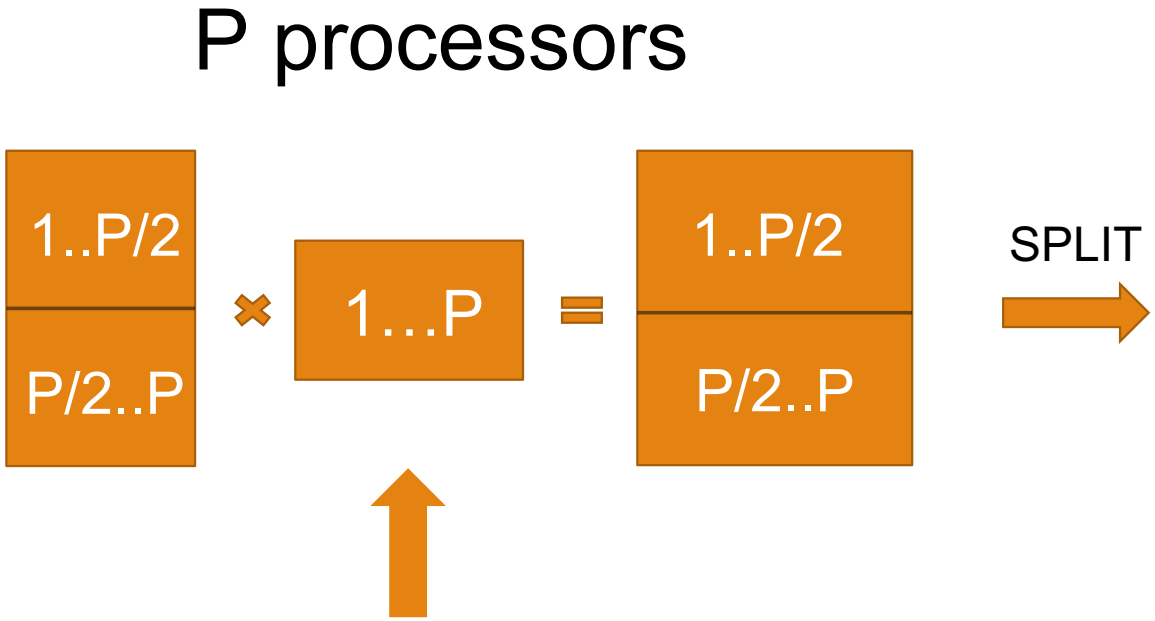
Parallel Step: Closer Look

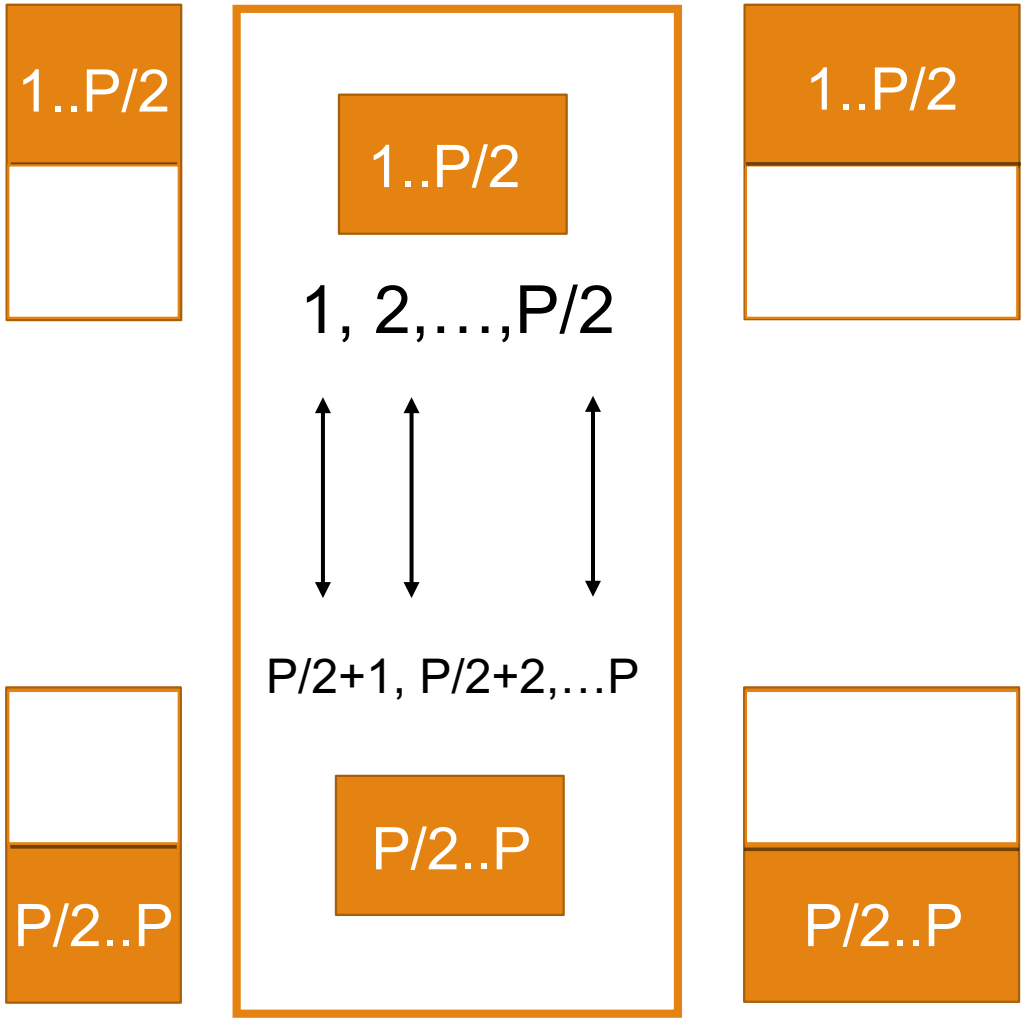
- Split the largest dimension
- Repeat recursively



Parallel Step: Closer Look

- Split the largest dimension
- Repeat recursively

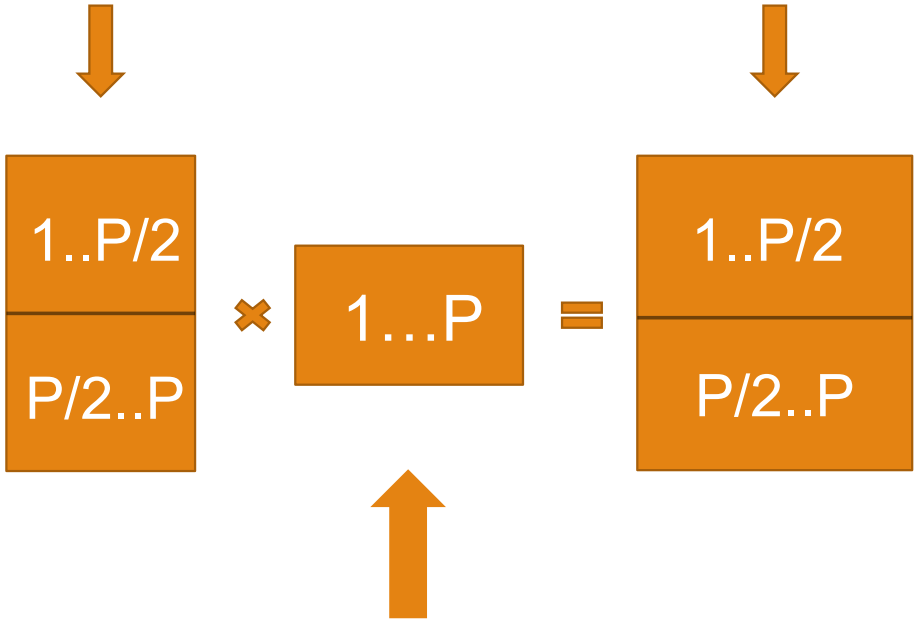




- non-splitted matrix communicated
- splitted matrices must NOT be communicated
- non-powers of 2 more complicated

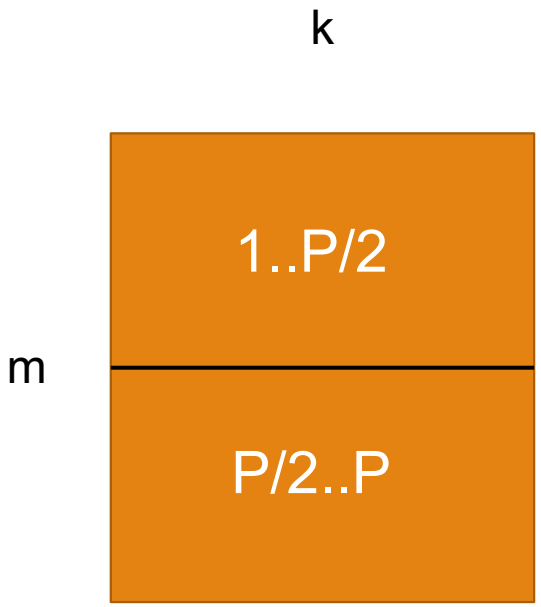
Data Layout

split ⟹ not communicated (data should initially be split like this)



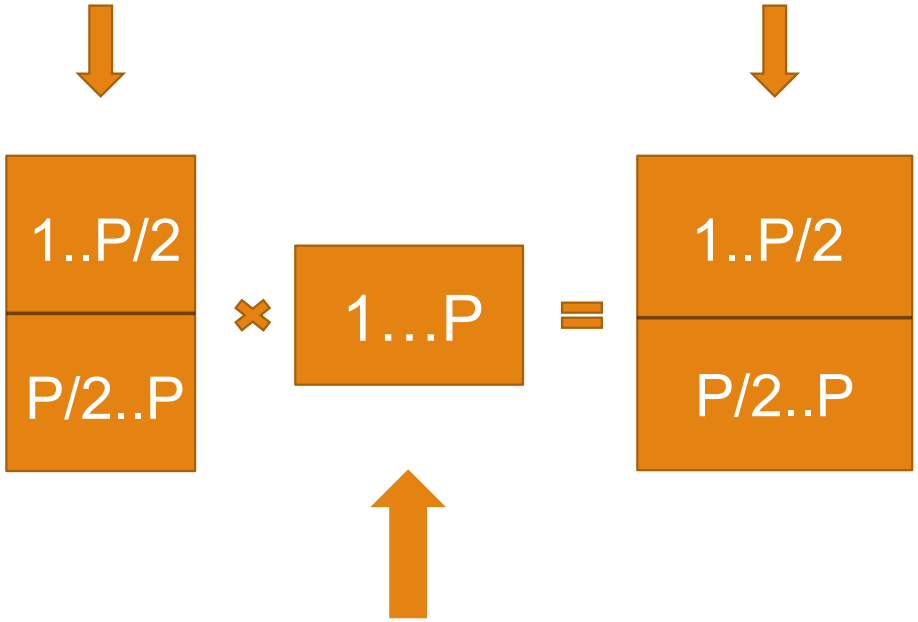
not split ⟹ communicated

1. Parallel (m/2)

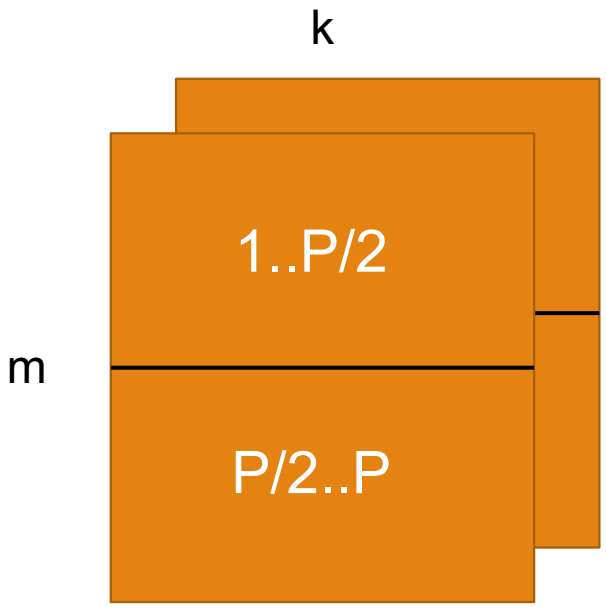


Data Layout

split ⟹ not communicated (data should initially be split like this)



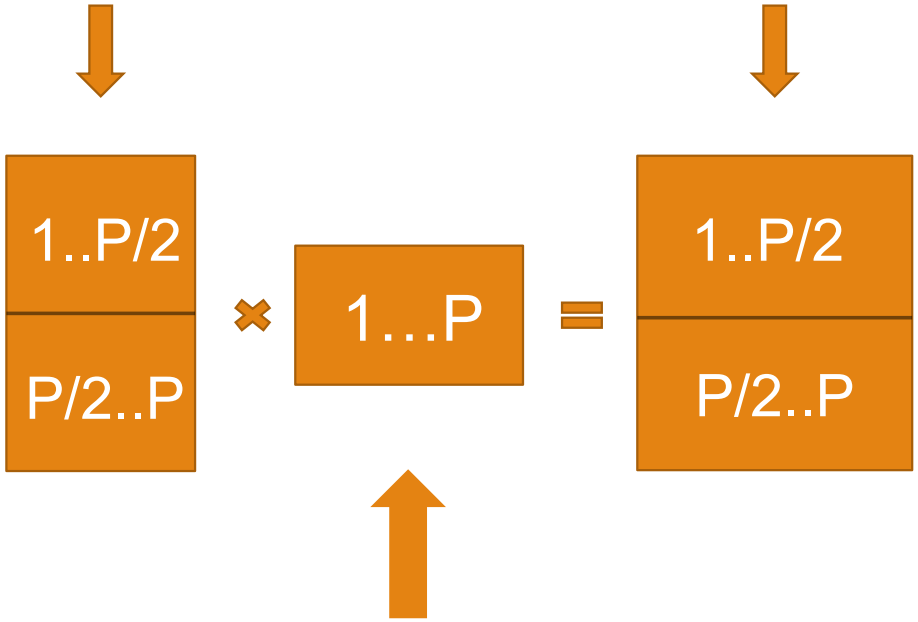
not split ⟹ communicated



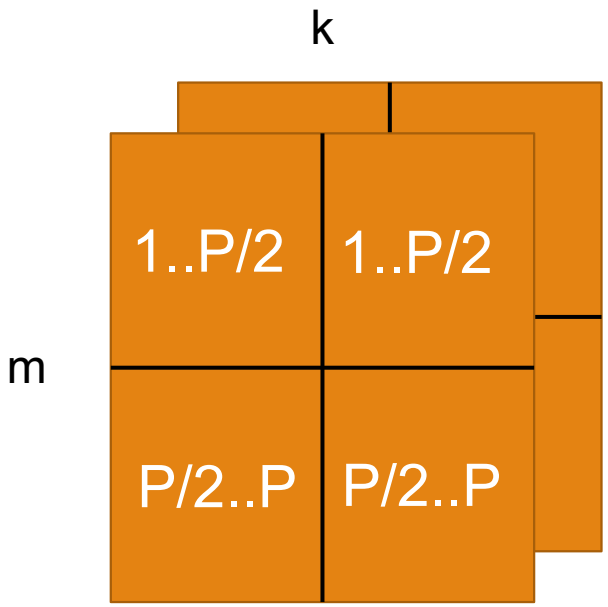
1. Parallel (m/2)
2. Parallel (n/2)

Data Layout

split ⟹ not communicated (data should initially be split like this)



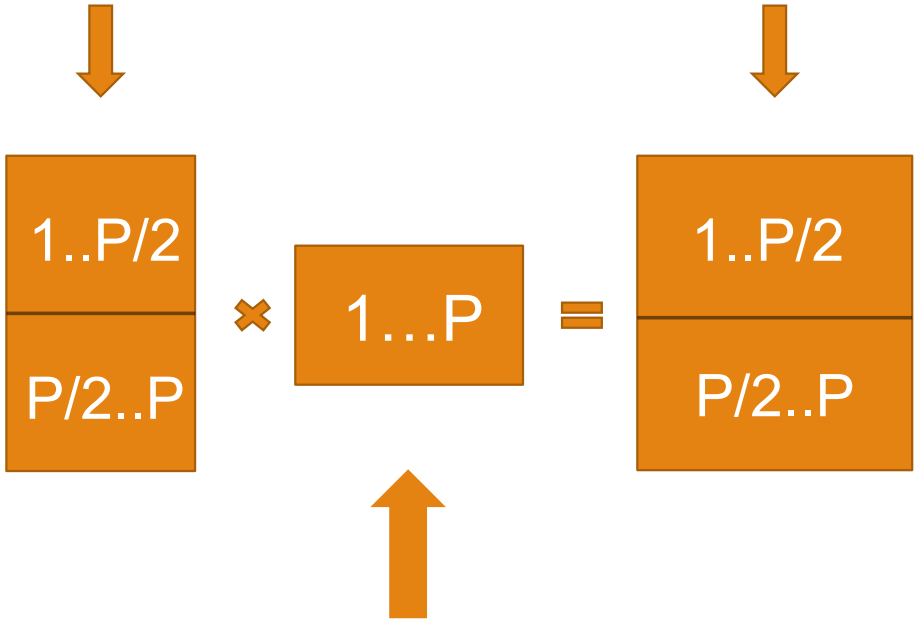
not split ⟹ communicated



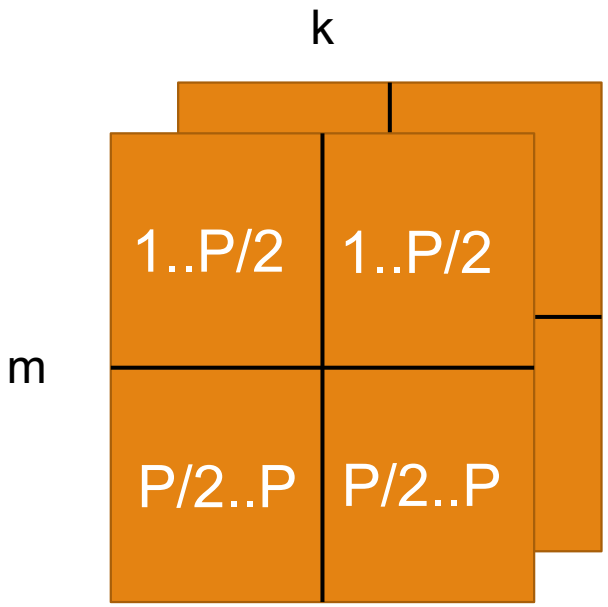
1. Parallel (m/2)
2. Parallel (n/2)
3. Sequential(k/2)

Data Layout

split ⟹ not communicated (data should initially be split like this)



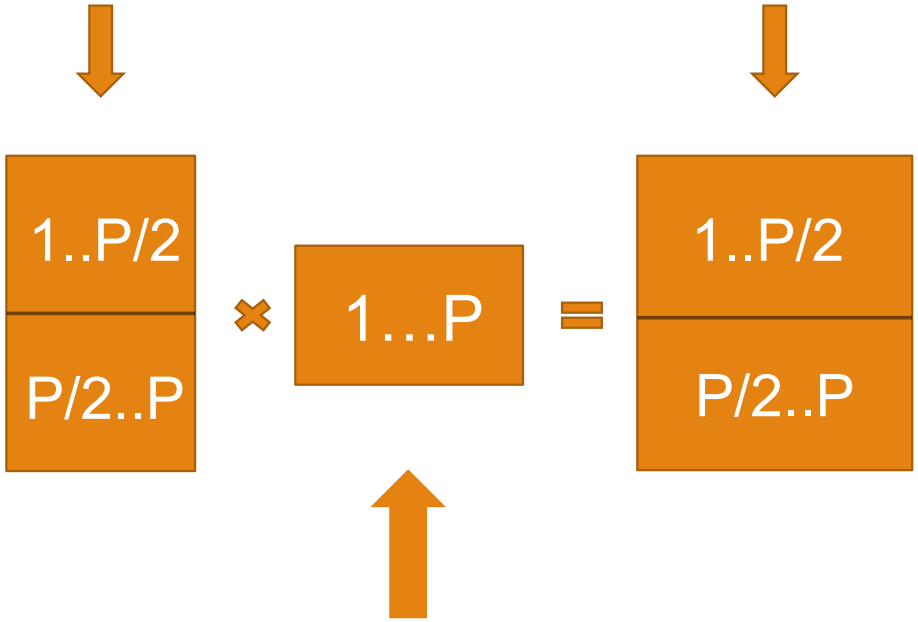
not split ⟹ communicated



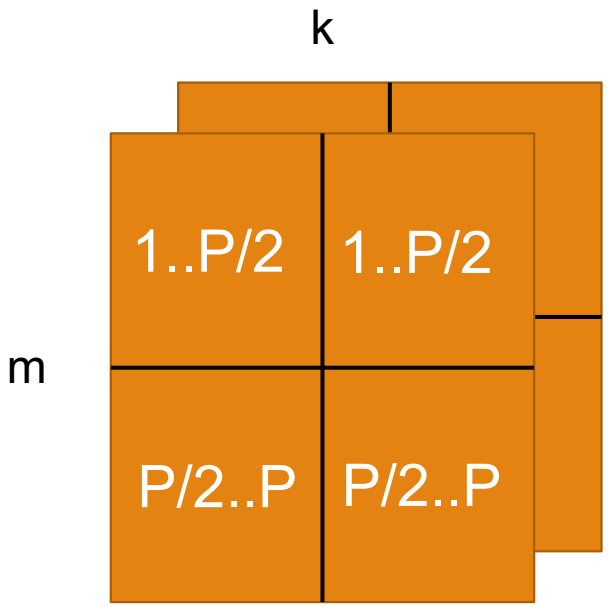
1. Parallel (m/2)
2. Parallel (n/2)
3. Sequential(k/2)
4. Base case: any load-balanced layout

Data Layout

split ⟹ not communicated (data should initially be split like this)



not split ⟹ communicated

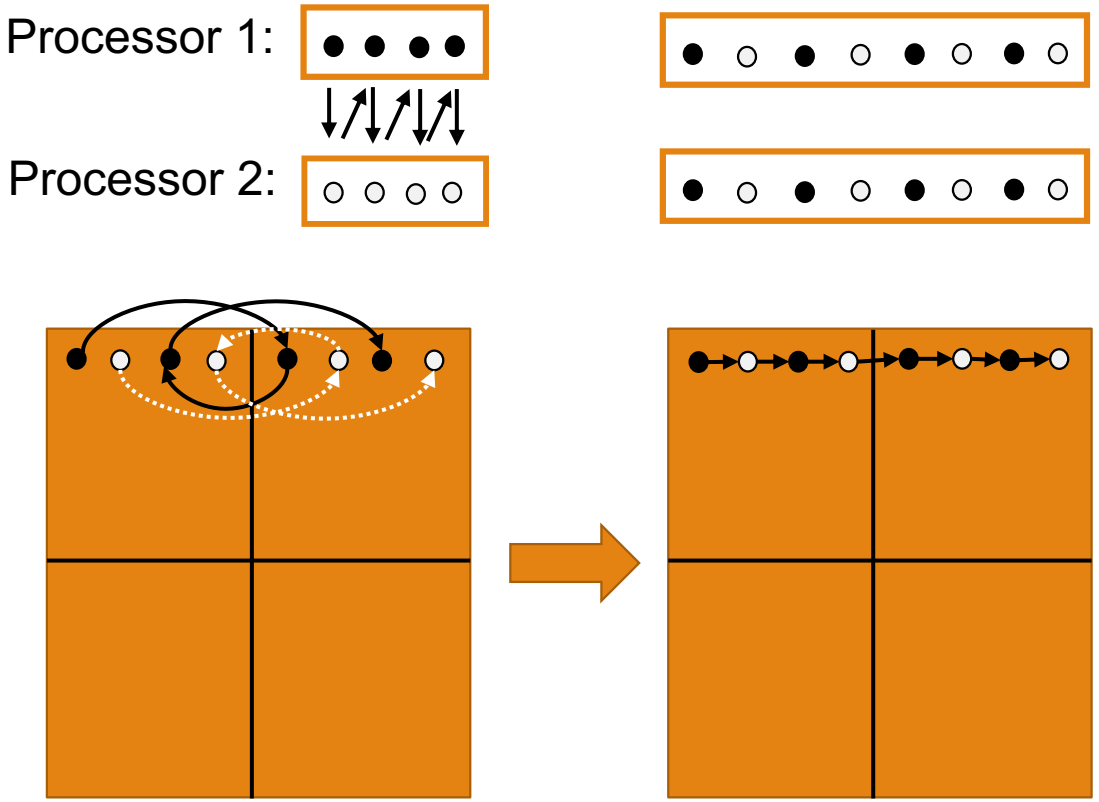


1. Parallel (m/2)
2. Parallel (n/2)
3. Sequential(k/2)
4. Base case: any load-balanced layout

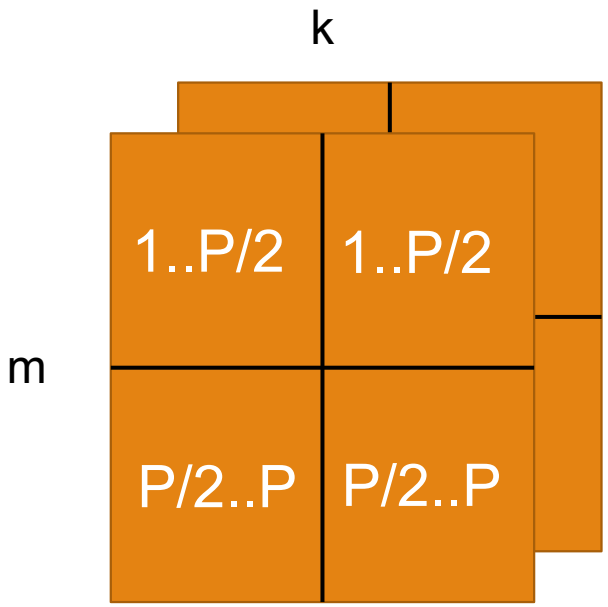
Problem: After each communication the data should be in row-major or column-major order.

Data Layout

Original implementation: cyclic layout in base case



After each communication => reshuffle

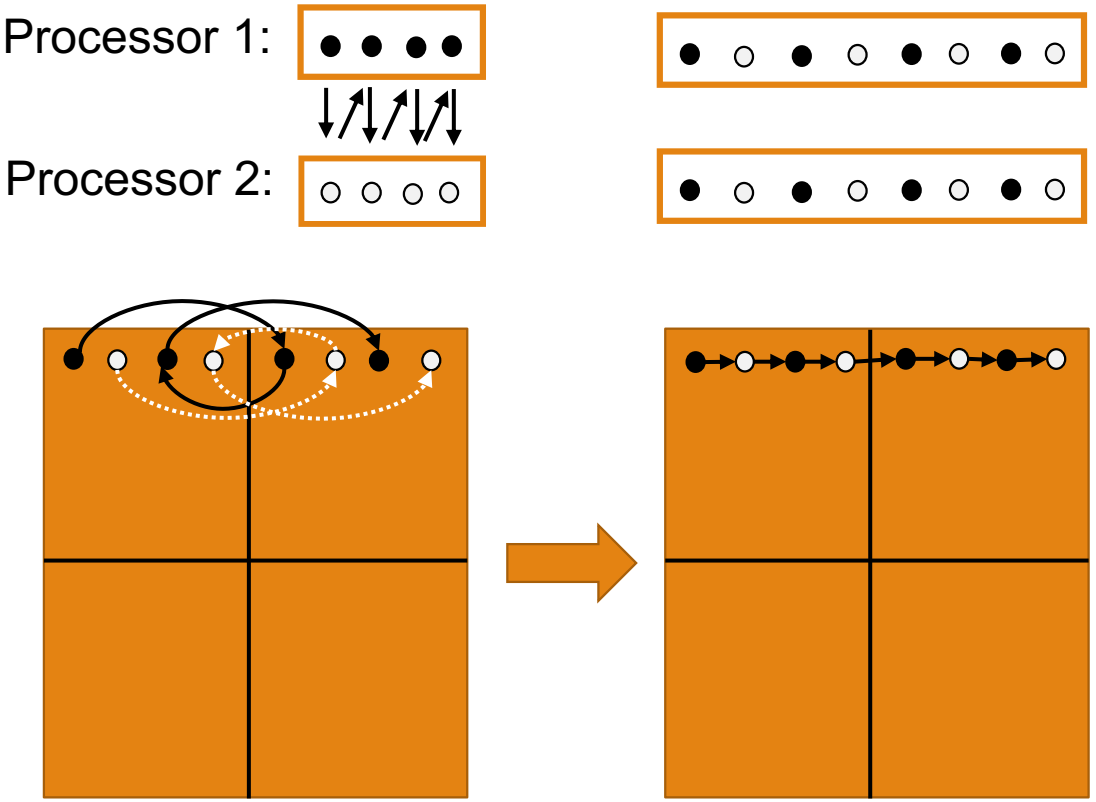


1. Parallel (m/2)
2. Parallel (n/2)
3. Sequential(k/2)
4. Base case: any load-balanced layout

Problem: After each communication the data should be in row-major or column-major order.

Data Layout

Original implementation: cyclic layout in base case



After each communication => reshuffle

Free to choose the layout in the base case

BUT!

After all the communication is performed
all the matrices must be row/column-major

Is there a different base-case layout that requires less data reshuffling?

Problem: After each communication the data should be in row-major or column-major order.

CARMA before

- **Powers of 2:** (m, n, k, P) are all powers of 2, divisions always by 2
- **Data Layout:** Initial/Final data layouts known only in special cases
- **Data Reshuffling:** all elements have to be reshuffled locally after each communication
- **K-split:** Reduction when dividing by k not performed ⇒ wrong results
- **Blocking:** Every communication blocks all ranks
- **Limited division strategy:** all Seqential steps have to be before all Parallel steps
- **Many bugs**

Main Improvements

- **Non-powers of 2:**
  - **Generalization:** algorithm generalization
  - **Asynchrony:** block only the subgroup of ranks
- **K-split:** implementation (requires reduction)
- **Data Layout:**
  - **New Layout:** introducing the new data layout
  - **Mapper:** finding the mapper $global(x, y) \leftrightarrow (rank, local_id)$
- **Any strategy:** allow any order of Sequential and Parallel steps

  (not necessarily all sequential steps must be before parallel steps)

Main Improvements

- **Non-powers of 2:**
  - **Generalization:** algorithm generalization
  - **Asynchrony:** block only the subgroup of ranks
- **K-split:** implementation (requires reduction)
- **Data Layout:**
  - **New Layout:** introducing the new data layout
  - **Mapper:** finding the mapper $global(x, y) \leftrightarrow (rank, local_id)$
- **Any strategy:** allow any order of Sequential and Parallel steps

  (not necessarily all sequential steps must be before parallel steps)

Generalizing for non-powers of 2

Before

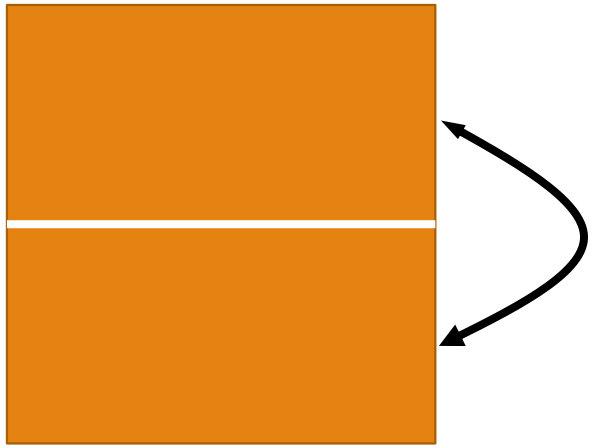
(m, n, k, P)

$(m/2, n, k, P/2)$ $(m/2, n, k, P/2)$

n

$m/2$

$m/2$



symmetric: no need to differentiate between the subproblems

Now

$(m_1, m_2, n_1, n_2, k_1, k_2, P_1, P_2)$

$\dots$

$(m_1, m_2', n_1, n_2', k_1, k_2', P_1, P_2')$ $\dots$

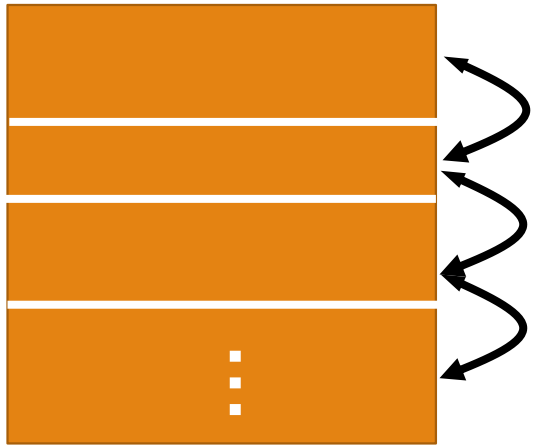
n_1 n_2

m_1

m_2'

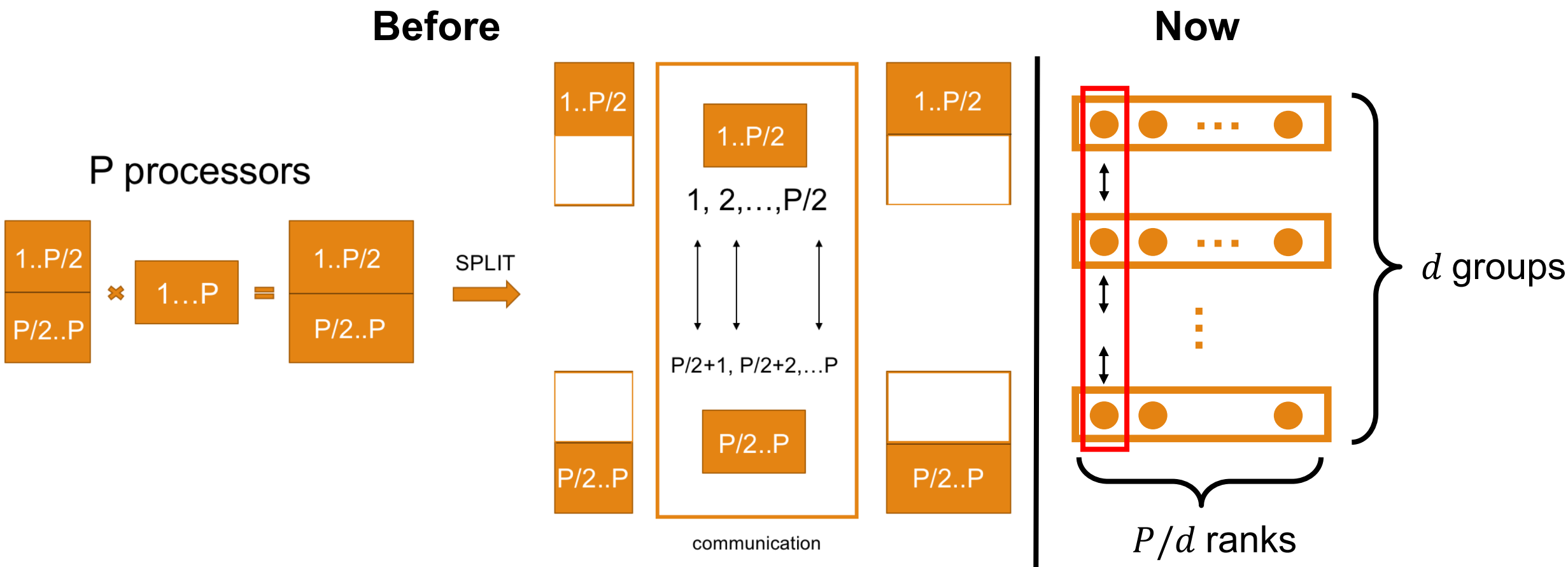
m_2''

m_2'''



non-symmetric: exact intervals (rather than just lengths) break the symmetry

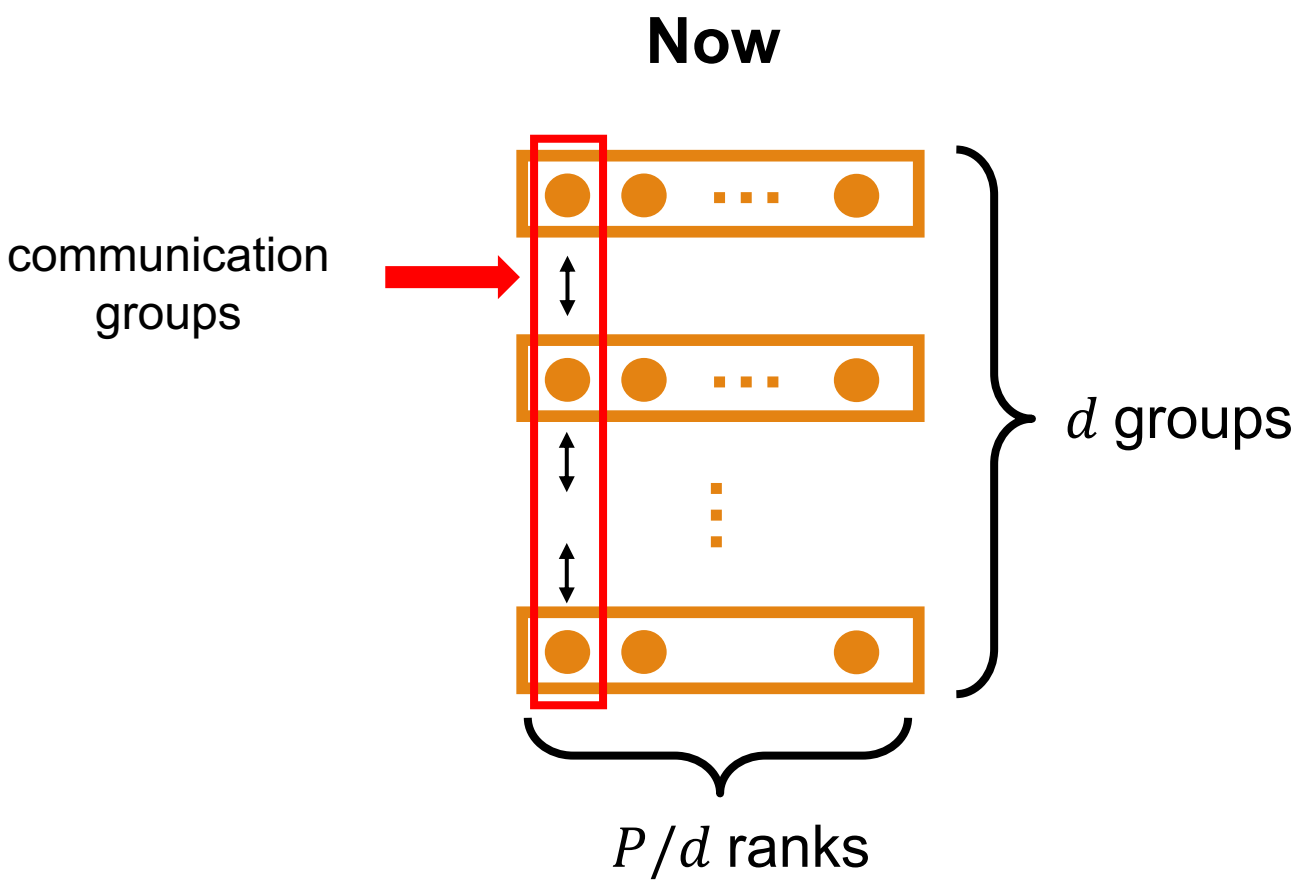
Generalizing for non-powers of 2



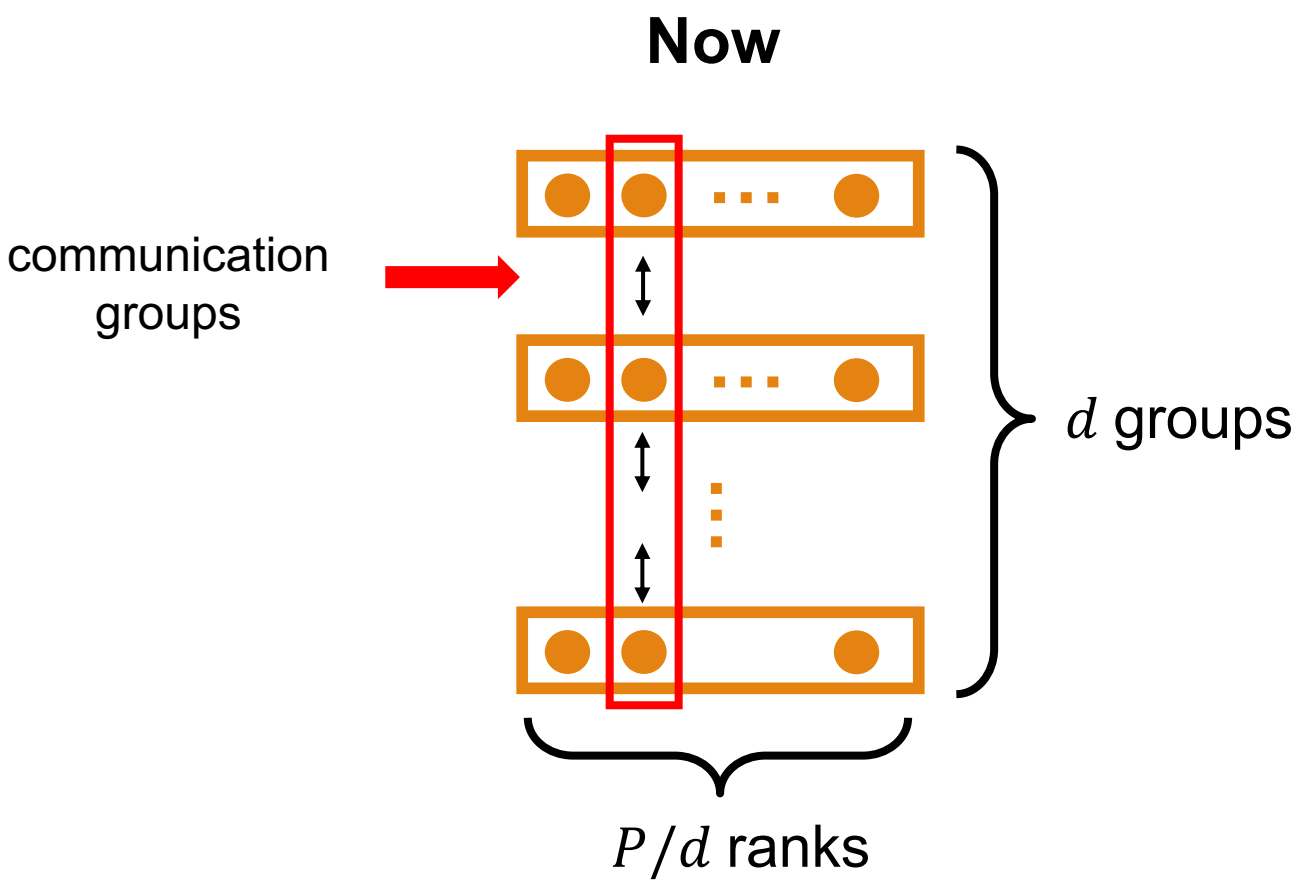
division always by 2 → ranks communicate in groups of 2

division by d

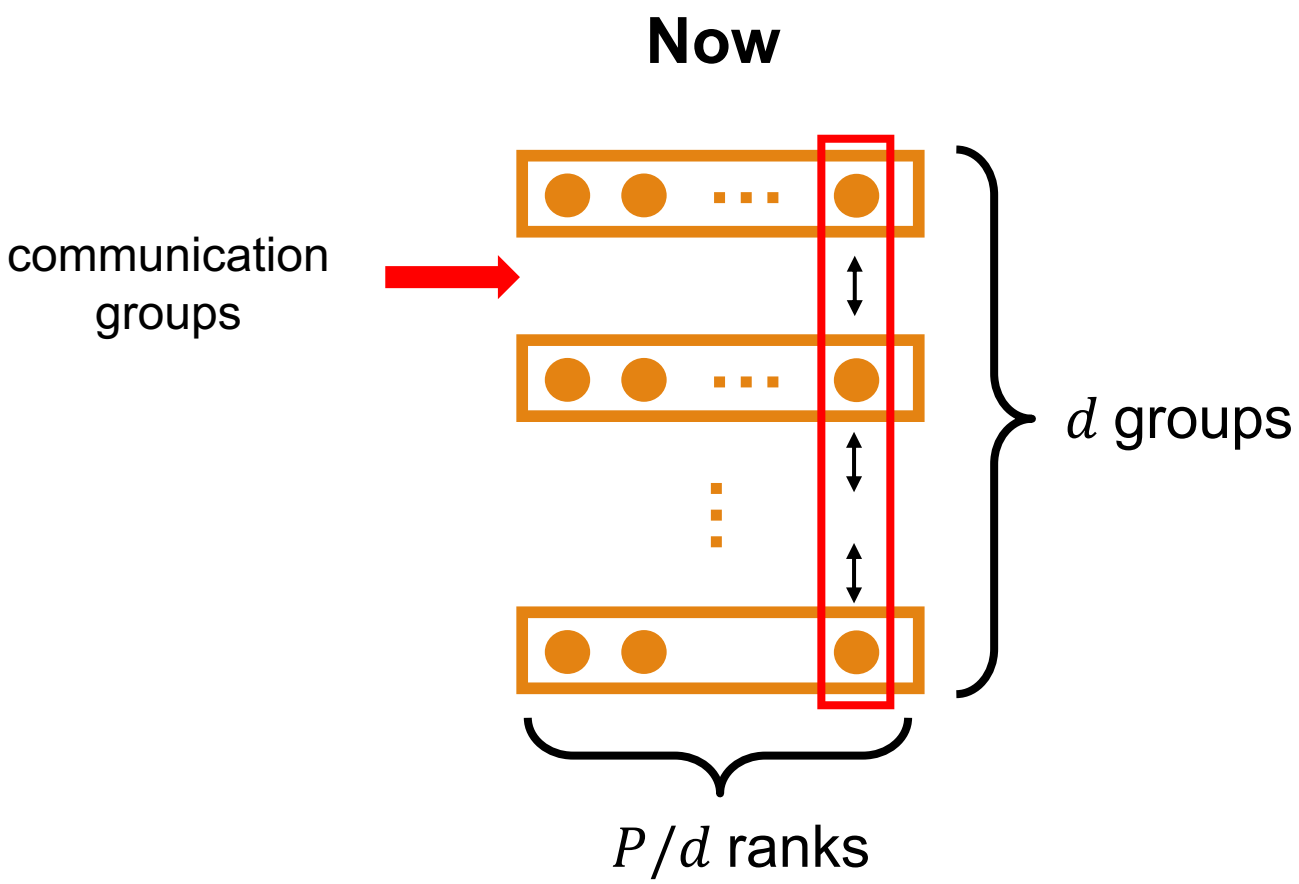
Generalizing for non-powers of 2



Generalizing for non-powers of 2



Generalizing for non-powers of 2

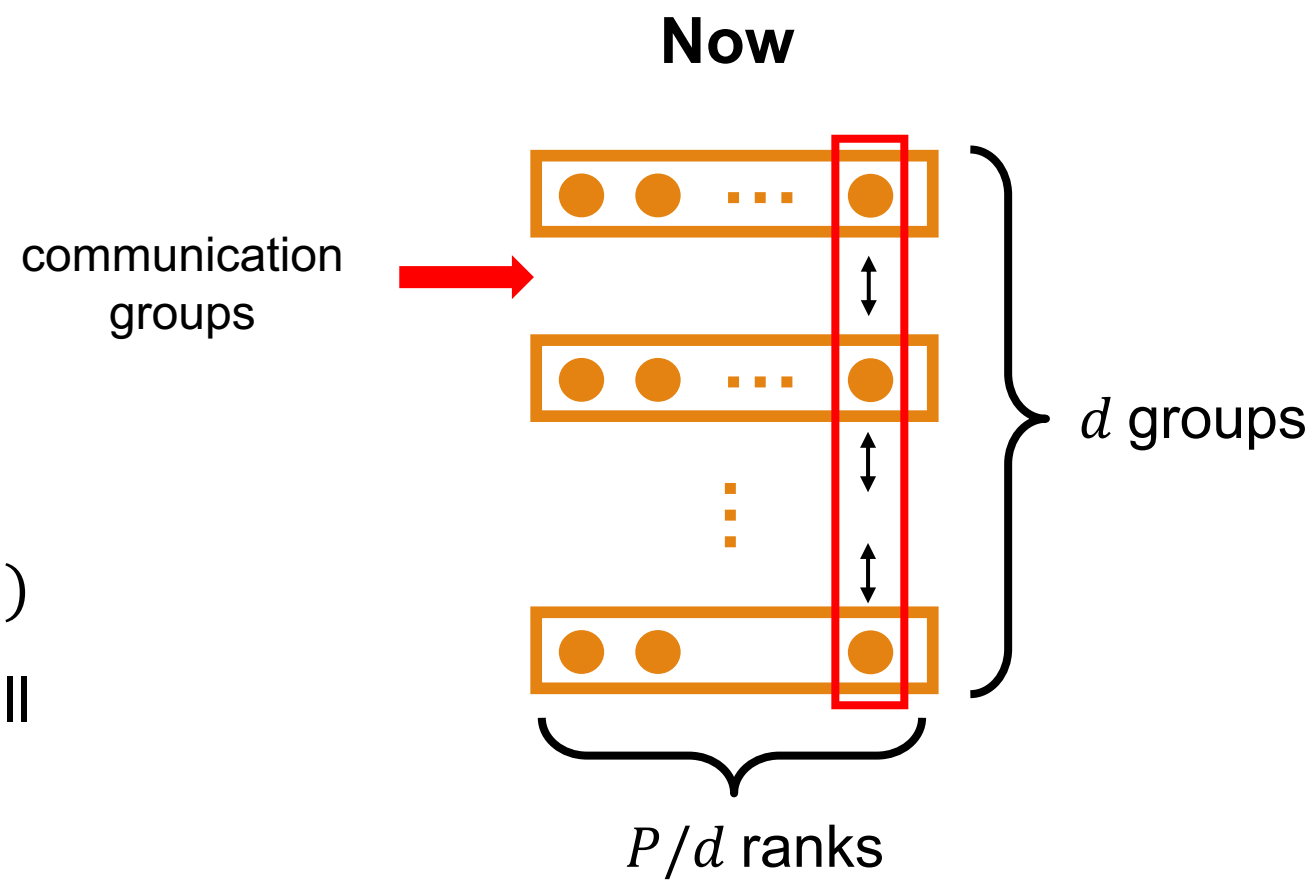


Generalizing for non-powers of 2

- When splitting by d, ranks split into d groups of P/d ranks.
- Use collective MPI routines instead of Send/Receive
- No need to use: $MPI_Barrier(MPI_COMM_WORLD)$

  blocking collective MPI routines will block only the necessary ranks

  → more asynchronous



Main Improvements

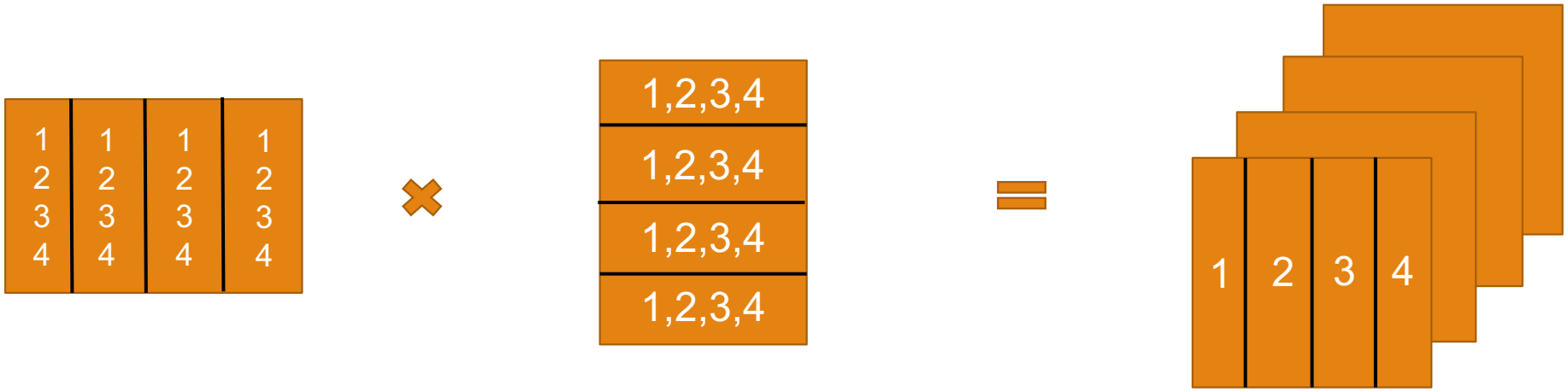
- ✓ **Non-powers of 2:**
  - ✓ **Generalization:** algorithm generalization
  - ✓ **Asynchrony:** block only the subgroup of ranks
- **K-split:** implementation (requires reduction)
- **Data Layout:**
  - **New Layout:** introducing the new data layout
  - **Mapper:** finding the mapper $global(x, y) \leftrightarrow (rank, local_id)$
- **Any strategy:** allow any order of Sequential and Parallel steps

  (not necessarily all sequential steps must be before parallel steps)

Implementing k-split

- No reduction in the original implementation => wrong results

Splitting k



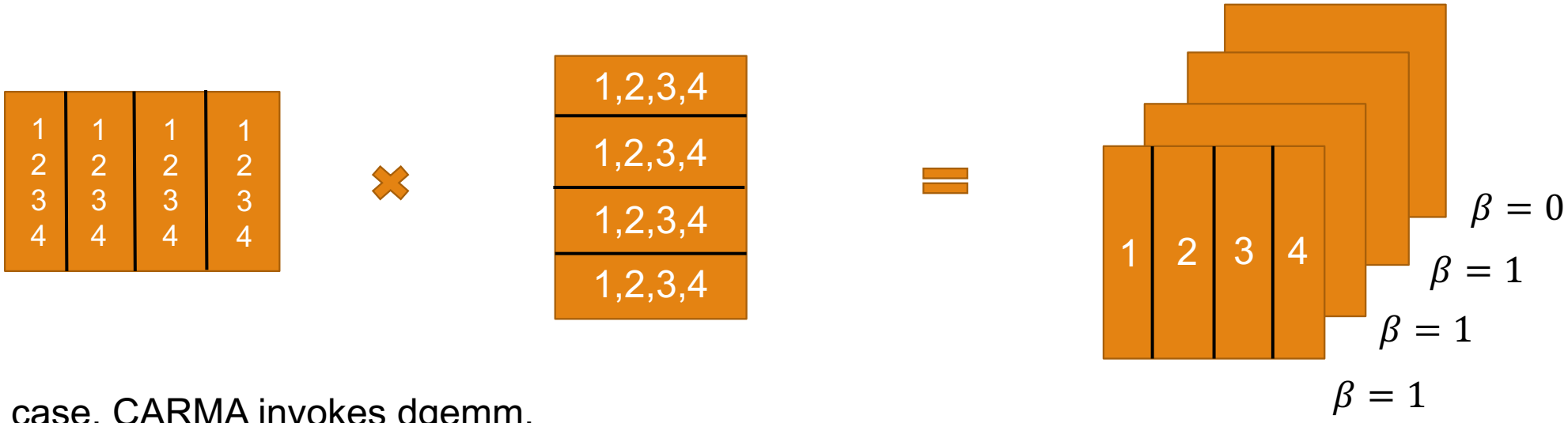
- At the base case, CARMA invokes dgemm.
- Propagate the beta parameter through the recursion to dgemm to perform the reduction on the fly when sequentialy splitting k

No communication since sequential (same space reused)!

Implementing k-split

- No reduction in the original implementation => wrong results

Splitting k



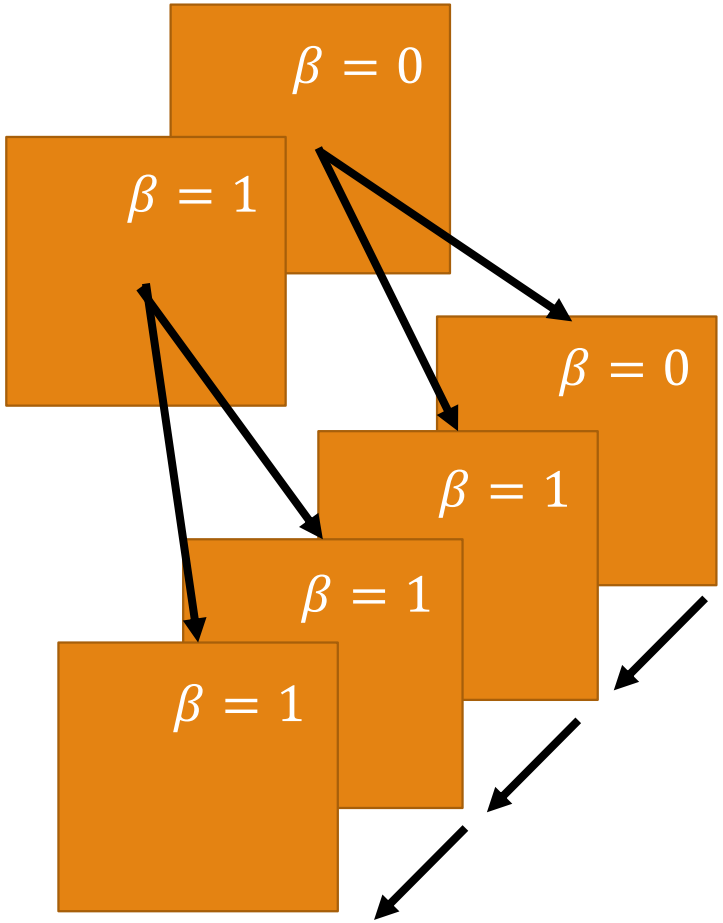
- At the base case, CARMA invokes dgemm.

- **IDEA:** Propagate the beta parameter through the recursion to dgemm to perform the reduction on the fly when sequentialy splitting k

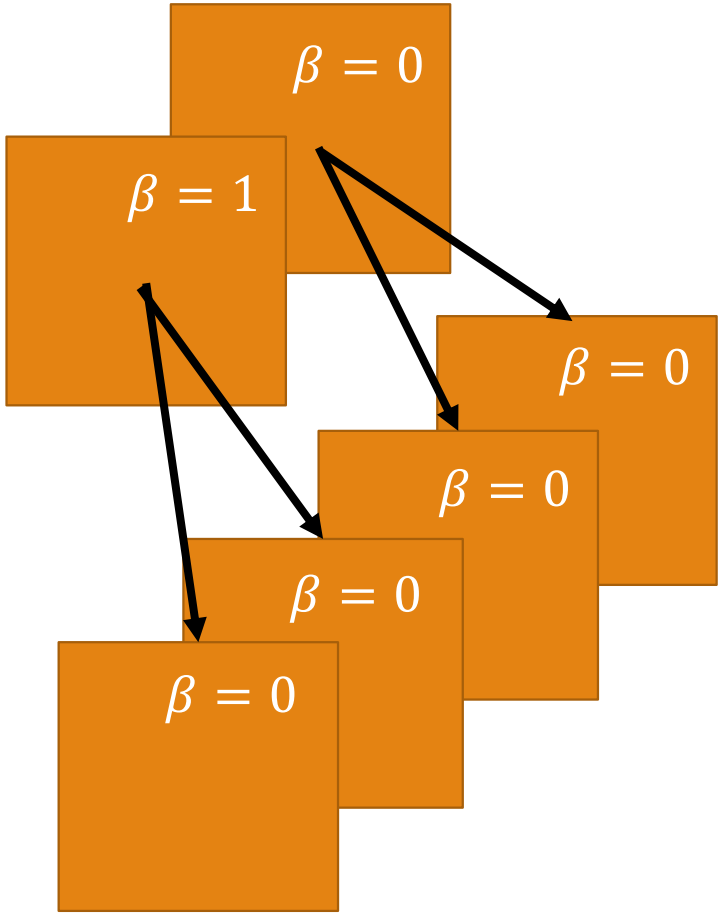
No communication since sequential (same space reused)!

Propagating β-parameter

Sequential step

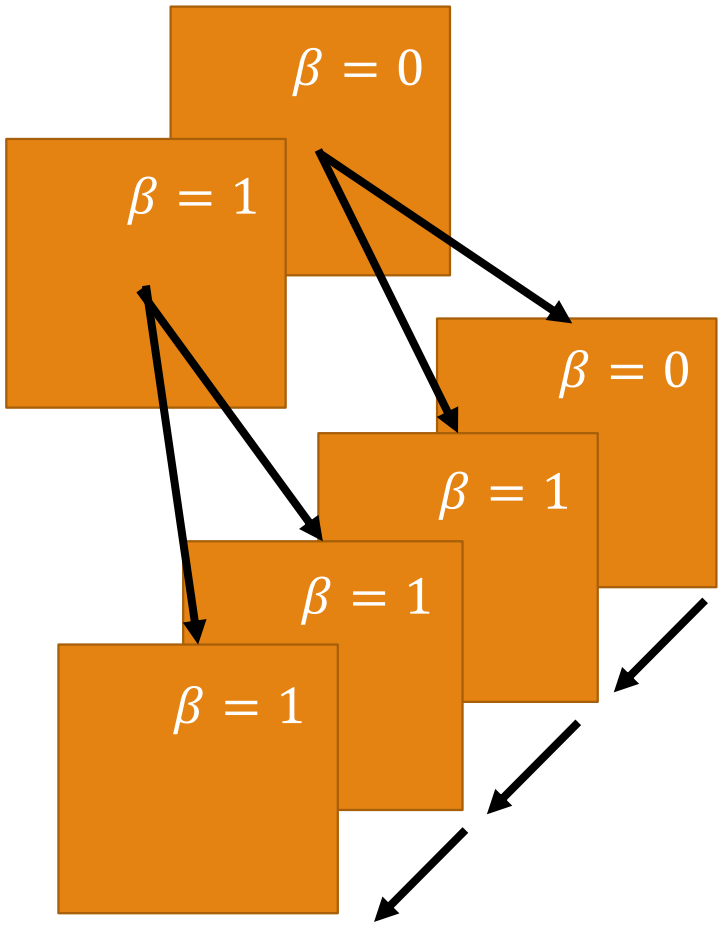


Parallel step

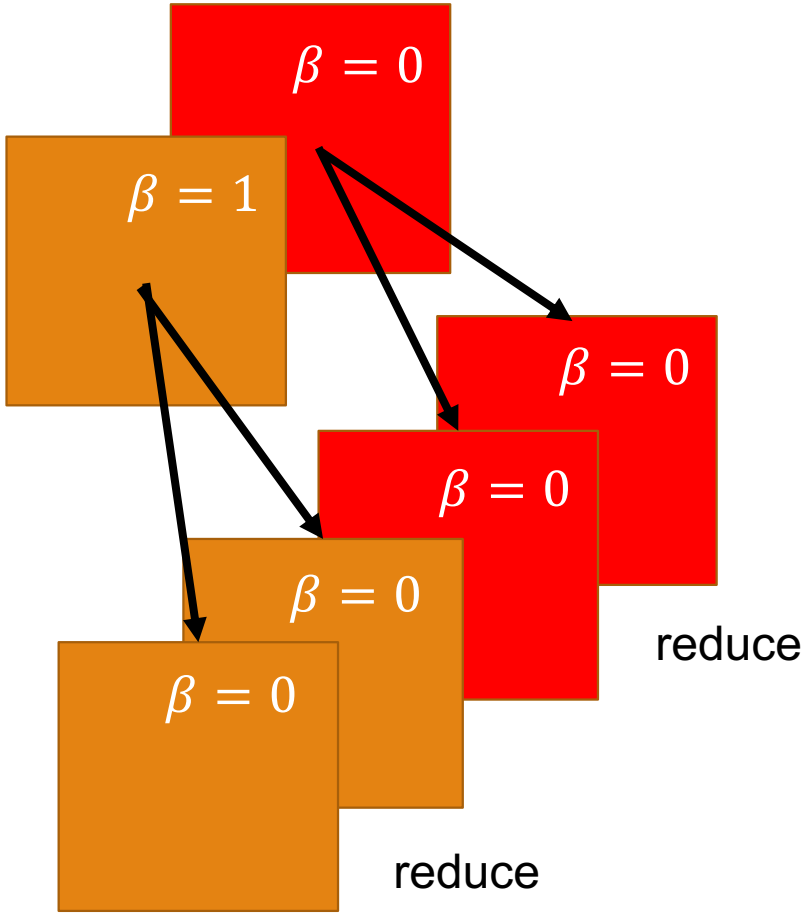


Propagating β-parameter

Sequential step

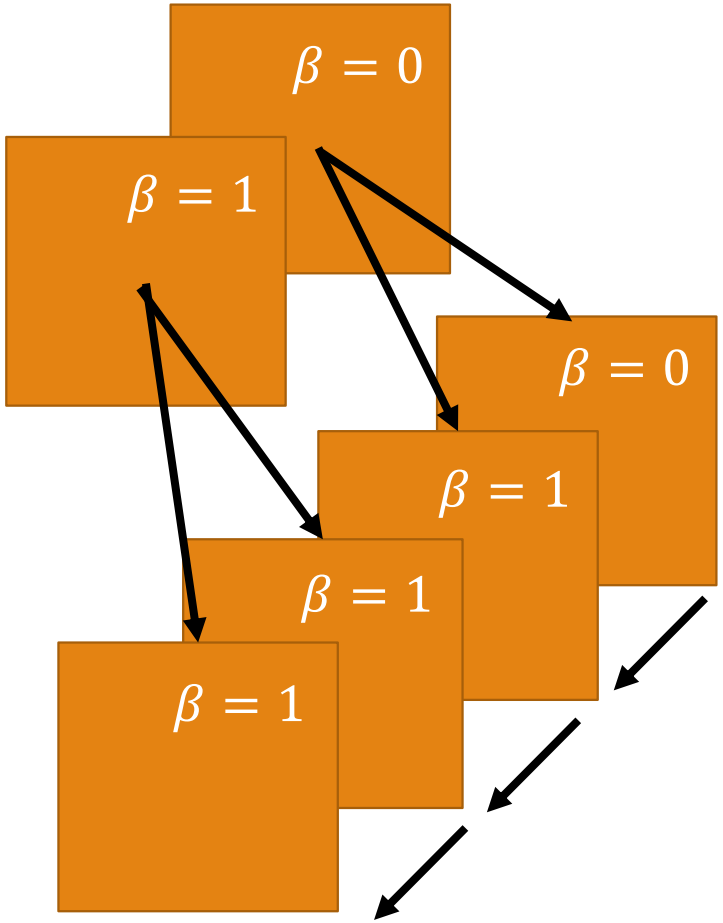


Parallel step

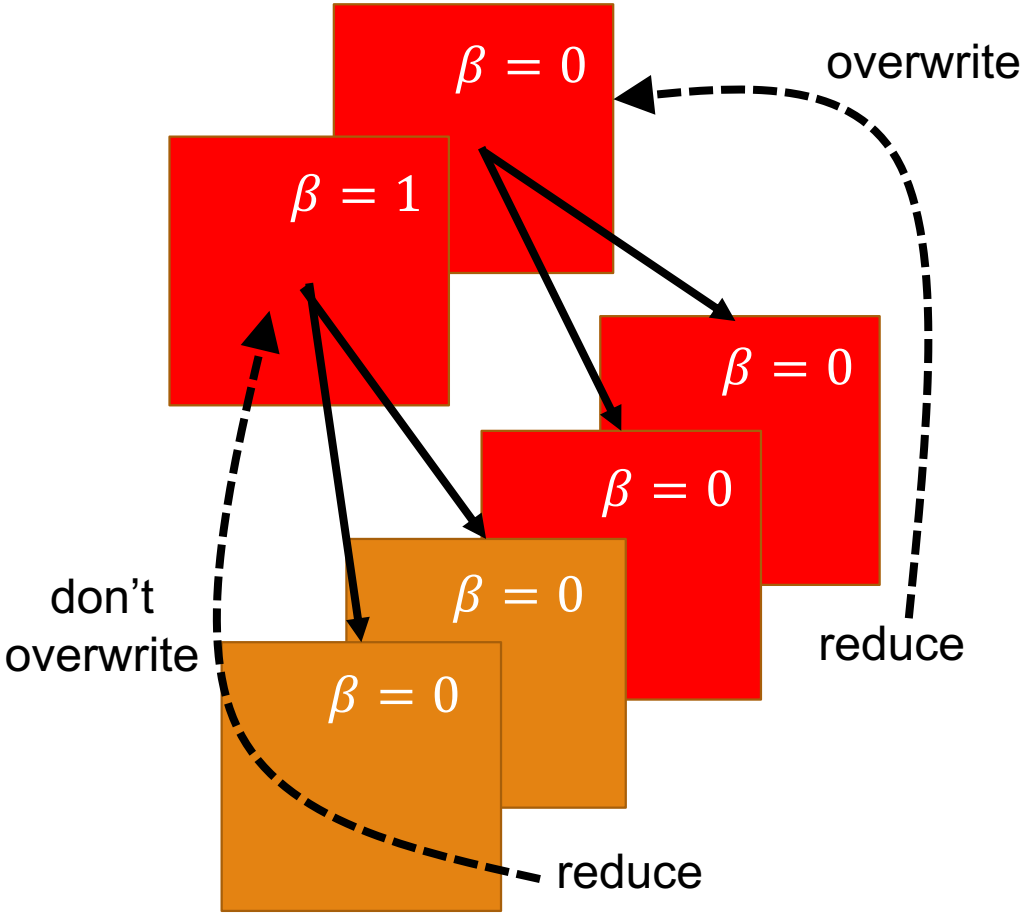


Propagating β-parameter

Sequential step



Parallel step



Main Improvements

- ✓ **Non-powers of 2:**
  - ✓ **Generalization:** algorithm generalization
  - ✓ **Asynchrony:** block only the subgroup of ranks
- ✓ **K-split:** implementation (requires reduction)
- **Data Layout:**
  - **New Layout:** introducing the new data layout
  - **Mapper:** finding the mapper $global(x, y) \leftrightarrow (rank, local_id)$
- **Any strategy:** allow any order of Sequential and Parallel steps

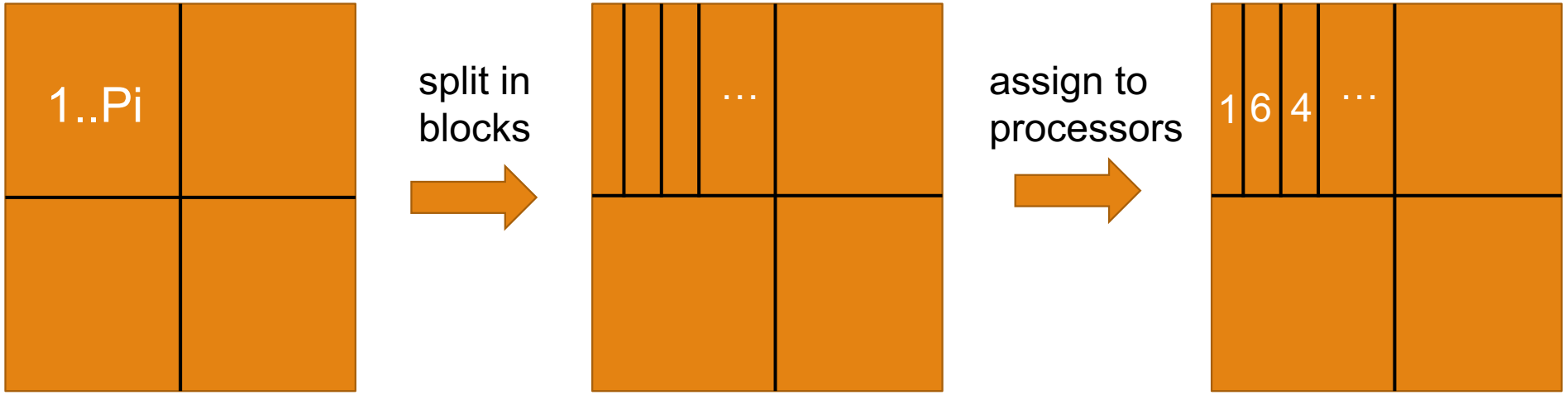
  (not necessarily all sequential steps must be before parallel steps)

Changing the underlying data layout

- Data Layout depends on the divisions that we want to perform

IDEA: take advantage of knowing the communication groups at each step to get the underlying layout which requires no reshuffling.

Permutation of processors depends on the order of performed communication



Changing the underlying data layout

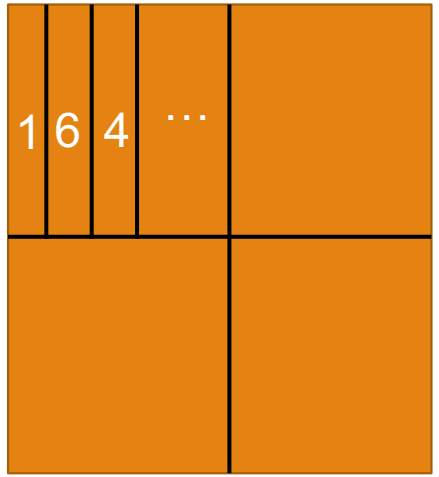
- Data Layout depends on the divisions that we want to perform

→ **IDEA:** take advantage of knowing the communication groups at each step to get the underlying layout which requires no reshuffling.

- Processors which communicate first => adjacent in the permutation.
- Apply the previous condition recursively for each communication step

→ **No need for reshuffling**.

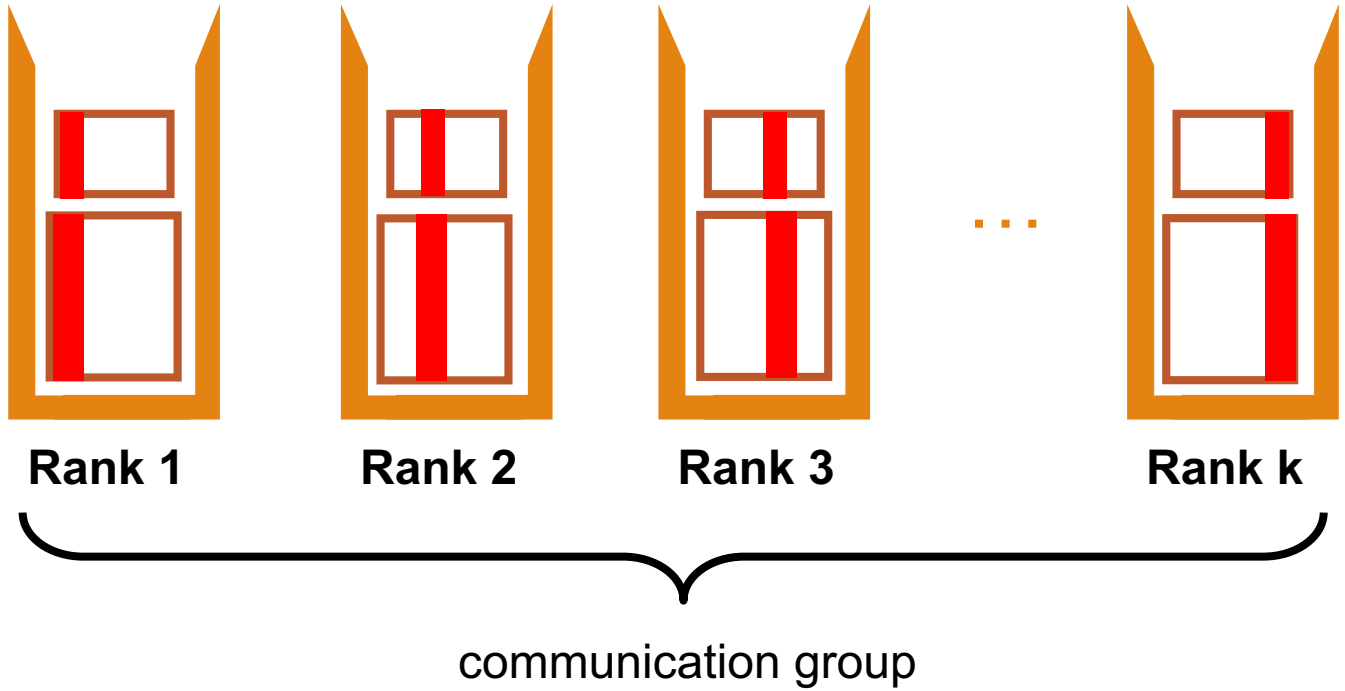
Permutation of processors depends on the order of performed communication



Changing the underlying data layout

Problem: communication groups known only **before** recursion and blocks **after** (not possible to separate these two tasks).

Solution: Stack



Forward propagation:

Feel up stacks with blocks that each rank is dealing with

Backward propagation:

Share the existing blocks among all the ranks in the same communication group

Changing the underlying data layout

Example: (m, n, k, P) = (8, 8, 8, 8)

Steps: Parallel(m/2, n/2, k/2)

A	B	C
0 0 2 2 1 1 3 3	0 0 4 4 2 2 6 6	0 0 1 1 2 2 3 3
0 0 2 2 1 1 3 3	0 0 4 4 2 2 6 6	0 0 1 1 2 2 3 3
0 0 2 2 1 1 3 3	0 0 4 4 2 2 6 6	0 0 1 1 2 2 3 3
0 0 2 2 1 1 3 3	0 0 4 4 2 2 6 6	0 0 1 1 2 2 3 3
4 4 6 6 5 5 7 7	1 1 5 5 3 3 7 7	4 4 5 5 6 6 7 7
4 4 6 6 5 5 7 7	1 1 5 5 3 3 7 7	4 4 5 5 6 6 7 7
4 4 6 6 5 5 7 7	1 1 5 5 3 3 7 7	4 4 5 5 6 6 7 7
4 4 6 6 5 5 7 7	1 1 5 5 3 3 7 7	4 4 5 5 6 6 7 7

Changing the underlying data layout

Example: (m, n, k, P) = (8, 8, 8, 8)

Steps: Parallel(m/2, n/2, k/2)

1. Step: Parallel (m/2)

Communication (B): 0 – 4
1 – 5
2 – 6
3 – 7

A	B	C
0 0 2 2 1 1 3 3	0 0 4 4 2 2 6 6	0 0 1 1 2 2 3 3
0 0 2 2 1 1 3 3	0 0 4 4 2 2 6 6	0 0 1 1 2 2 3 3
0 0 2 2 1 1 3 3	0 0 4 4 2 2 6 6	0 0 1 1 2 2 3 3
0 0 2 2 1 1 3 3	0 0 4 4 2 2 6 6	0 0 1 1 2 2 3 3
4 4 6 6 5 5 7 7	1 1 5 5 3 3 7 7	4 4 5 5 6 6 7 7
4 4 6 6 5 5 7 7	1 1 5 5 3 3 7 7	4 4 5 5 6 6 7 7
4 4 6 6 5 5 7 7	1 1 5 5 3 3 7 7	4 4 5 5 6 6 7 7
4 4 6 6 5 5 7 7	1 1 5 5 3 3 7 7	4 4 5 5 6 6 7 7

0 0 2 2 1 1 3 3	0 0 0 0 2 2 2 2	0 0 1 1 2 2 3 3
0 0 2 2 1 1 3 3	0 0 0 0 2 2 2 2	0 0 1 1 2 2 3 3
0 0 2 2 1 1 3 3	0 0 0 0 2 2 2 2	0 0 1 1 2 2 3 3
0 0 2 2 1 1 3 3	0 0 0 0 2 2 2 2	0 0 1 1 2 2 3 3
	1 1 1 1 3 3 3 3	
	1 1 1 1 3 3 3 3	
	1 1 1 1 3 3 3 3	
	1 1 1 1 3 3 3 3	

Changing the underlying data layout

Example: (m, n, k, P) = (8, 8, 8, 8)

Steps: Parallel(m/2, n/2, k/2)

2. Step: Parallel (n/2)

Communication (A): 0 – 2
1 – 3

A

0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1

B

0	0	0	0	2	2	2	2
0	0	0	0	2	2	2	2
0	0	0	0	2	2	2	2
0	0	0	0	2	2	2	2
1	1	1	1	3	3	3	3
1	1	1	1	3	3	3	3
1	1	1	1	3	3	3	3
1	1	1	1	3	3	3	3

C

0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3

0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

0	0	1	1
0	0	1	1
0	0	1	1
0	0	1	1

Changing the underlying data layout

Example: (m, n, k, P) = (8, 8, 8, 8)

Steps: Parallel(m/2, n/2, k/2)

3. Step: Parallel (k/2)

Communication (C): 0 – 1
with reduction

A

0 0 0 0	1 1 1 1
0 0 0 0	1 1 1 1
0 0 0 0	1 1 1 1
0 0 0 0	1 1 1 1

B

0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1

C

0 0 1 1
0 0 1 1
0 0 1 1
0 0 1 1

0 0 0 0	0 0 0 0	1 1 1 1
0 0 0 0	0 0 0 0	1 1 1 1
0 0 0 0	0 0 0 0	1 1 1 1
0 0 0 0	0 0 0 0	1 1 1 1

0 0 1 1	0 0 0 0	0 0 0 0
0 0 1 1	0 0 0 0	0 0 0 0
0 0 1 1	0 0 0 0	0 0 0 0
0 0 1 1	0 0 0 0	0 0 0 0

reduction

Reshuffling still necessary sometimes

Example: (m, n, k, P) = (8, 8, 8, 8)

Steps: Parallel(m/2, n/2, k/2)

A	B	C
0 0 2 2 1 1 3 3	0 0 4 4 2 2 6 6	0 0 1 1 2 2 3 3
0 0 2 2 1 1 3 3	0 0 4 4 2 2 6 6	0 0 1 1 2 2 3 3
0 0 2 2 1 1 3 3	0 0 4 4 2 2 6 6	0 0 1 1 2 2 3 3
0 0 2 2 1 1 3 3	0 0 4 4 2 2 6 6	0 0 1 1 2 2 3 3
4 4 6 6 5 5 7 7	1 1 5 5 3 3 7 7	4 4 5 5 6 6 7 7
4 4 6 6 5 5 7 7	1 1 5 5 3 3 7 7	4 4 5 5 6 6 7 7
4 4 6 6 5 5 7 7	1 1 5 5 3 3 7 7	4 4 5 5 6 6 7 7
4 4 6 6 5 5 7 7	1 1 5 5 3 3 7 7	4 4 5 5 6 6 7 7

Example: (m, n, k, P) = (8, 8, 8, 4)

Steps: Par(m/2), Seq(n/2), Par(k/2)

- In Sequential steps, same processor can end up in both halves of the matrices.
- Reshuffling of blocks necessary

0 0 0 0 1 1 1 1	0 0 2 2 0 0 2 2	0 0 1 1 0 0 1 1
0 0 0 0 1 1 1 1	0 0 2 2 0 0 2 2	0 0 1 1 0 0 1 1
0 0 0 0 1 1 1 1	0 0 2 2 0 0 2 2	0 0 1 1 0 0 1 1
0 0 0 0 1 1 1 1	0 0 2 2 0 0 2 2	0 0 1 1 0 0 1 1
2 2 2 2 3 3 3 3	1 1 3 3 1 1 3 3	2 2 3 3 2 2 3 3
2 2 2 2 3 3 3 3	1 1 3 3 1 1 3 3	2 2 3 3 2 2 3 3
2 2 2 2 3 3 3 3	1 1 3 3 1 1 3 3	2 2 3 3 2 2 3 3
2 2 2 2 3 3 3 3	1 1 3 3 1 1 3 3	2 2 3 3 2 2 3 3

Reshuffling still necessary sometimes

- Happens only when Sequential step is nested within Parallel step of the same matrix
- Reshuffling of blocks still better than reshuffling of single elements

0: B1, B3

↓ ↗ ↓

2: B2, B4

comm. →

0: B1, B2, B3, B4

2: B1, B2, B3, B4

Example: (m, n, k, P) = (8, 8, 8, 4)

Steps: Par(m/2), Seq(n/2), Par(k/2)

- In Sequential steps, same processor can end up in both halves of the matrices.
- Reshuffling of blocks necessary

0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3

B1 B2 B3 B4

0	0	2	2	0	0	2	2
0	0	2	2	0	0	2	2
0	0	2	2	0	0	2	2
0	0	2	2	0	0	2	2
1	1	3	3	1	1	3	3
1	1	3	3	1	1	3	3
1	1	3	3	1	1	3	3
1	1	3	3	1	1	3	3

0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
2	2	3	3	2	2	3	3
2	2	3	3	2	2	3	3
2	2	3	3	2	2	3	3
2	2	3	3	2	2	3	3

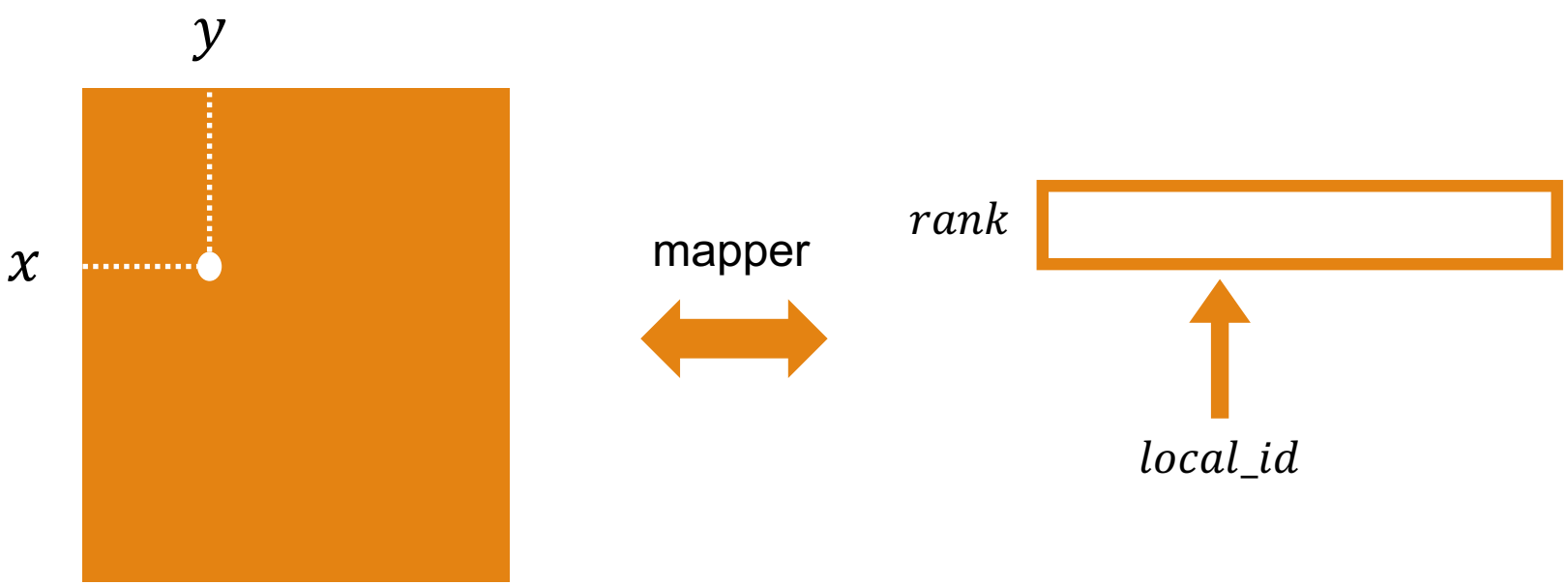
Main Improvements

- ✓ **Non-powers of 2:**
  - ✓ **Generalization:** algorithm generalization
  - ✓ **Asynchrony:** block only the subgroup of ranks
- ✓ **K-split:** implementation (requires reduction)
- **Data Layout:**
  - ✓ **New Layout:** introducing the new data layout
  - **Mapper:** finding the mapper $global(x, y) \leftrightarrow (rank, local_id)$
- **Any strategy:** allow any order of Sequential and Parallel steps

  (not necessarily all sequential steps must be before parallel steps)

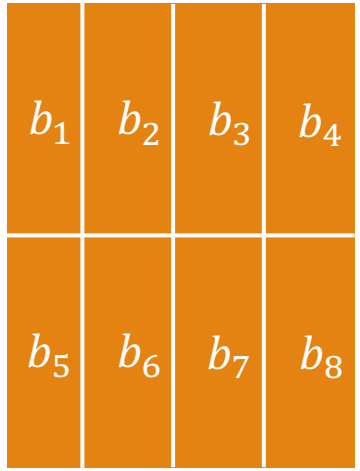
Mapper

$$global(x, y) \leftrightarrow (rank, local_id)$$



Mapper

0	0	2	2	0	0	2	2
0	0	2	2	0	0	2	2
0	0	2	2	0	0	2	2
0	0	2	2	0	0	2	2
1	1	3	3	1	1	3	3
1	1	3	3	1	1	3	3
1	1	3	3	1	1	3	3
1	1	3	3	1	1	3	3



rank	blocks
0	$\{b_1, b_3\}$
1	$\{b_5, b_7\}$
2	$\{b_2, b_4\}$
3	$\{b_6, b_8\}$

block	rank
b_1	0
b_2	2
b_3	0
b_4	2
b_5	1
b_6	3
b_7	1
b_8	3

Given: $global(x, y)$, find $(rank, local_id)$

IDEA: use **interval tree** to find which block a point (x, y) belongs to

Lemma:

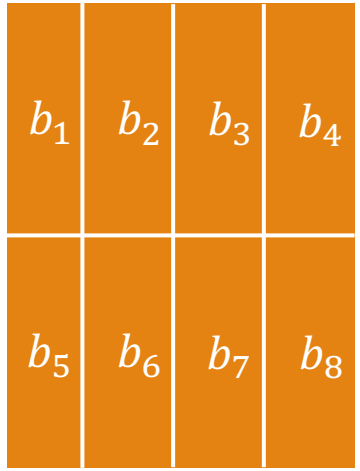
$$\#blocks\ per\ rank = \#Seq\ steps$$

Corollary :

$$no\ Seq\ steps \Rightarrow each\ rank\ owns\ a\ single\ block$$

Mapper

0	0	2	2	0	0	2	2
0	0	2	2	0	0	2	2
0	0	2	2	0	0	2	2
0	0	2	2	0	0	2	2
1	1	3	3	1	1	3	3
1	1	3	3	1	1	3	3
1	1	3	3	1	1	3	3
1	1	3	3	1	1	3	3



rank	blocks
$0 \rightarrow$	$\{b_1, b_3\}$
$1 \rightarrow$	$\{b_5, b_7\}$
$2 \rightarrow$	$\{b_2, b_4\}$
$3 \rightarrow$	$\{b_6, b_8\}$

block	rank
$b_1 \rightarrow$	0
$b_2 \rightarrow$	2
$b_3 \rightarrow$	0
$b_4 \rightarrow$	2
$b_5 \rightarrow$	1
$b_6 \rightarrow$	3
$b_7 \rightarrow$	1
$b_8 \rightarrow$	3

Given: $global(x, y)$, find $(rank, local_id)$

IDEA: use **interval tree** to find which block a point (x, y) belongs to

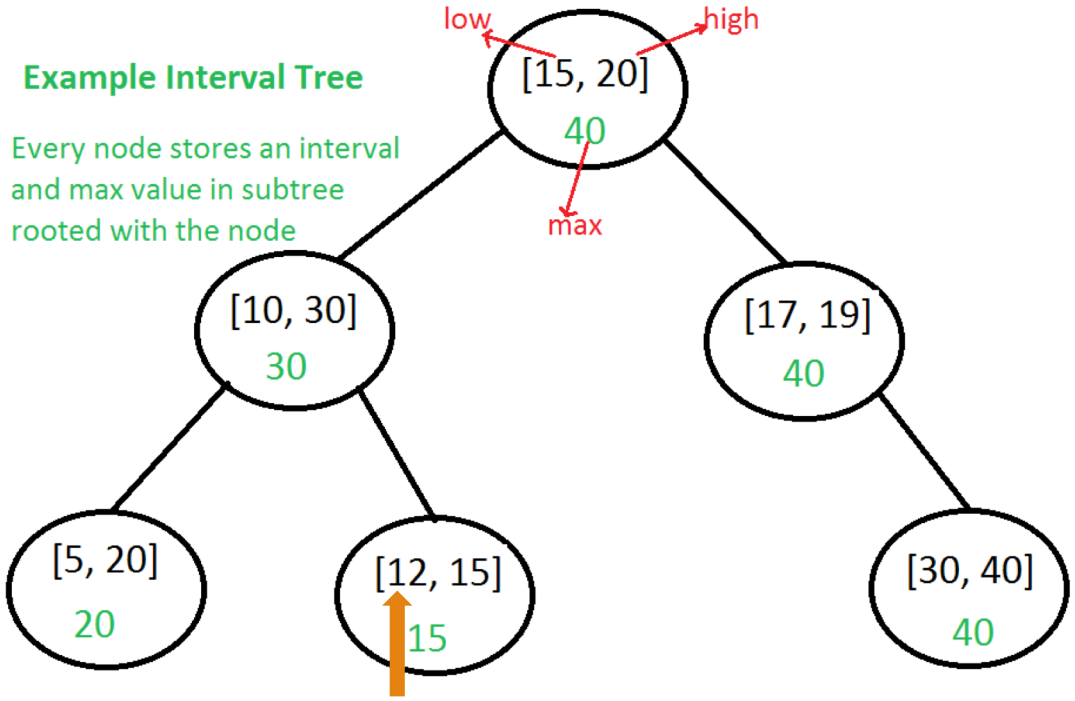
Lemma:

$$\#blocks\ per\ rank = \#Seq\ steps$$

Corollary :

$$no\ Seq\ steps \Rightarrow each\ rank\ owns\ a\ single\ block$$

Interval Tree



Self-balancing Binary Search Tree (BST) with respect to left endpoint

Used when we want to easily:

- add/delete interval
- find interval containing some given point/interval q

Query algorithm:

If $q \in root \Rightarrow return\ root$

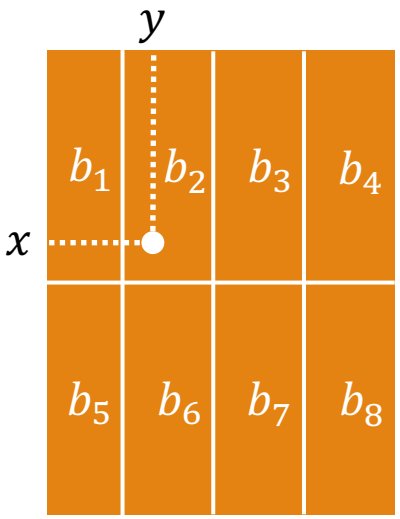
If $q \leq left.max \Rightarrow search\ left$
else
$search\ right$

Query complexity: $O(\log \#nodes)$

Mapper

Columns Interval Tree

Rows Interval Tree



rank	blocks
$0 \rightarrow$	$\{b_1, b_3\}$
$1 \rightarrow$	$\{b_5, b_7\}$
$2 \rightarrow$	$\{b_2, b_4\}$
$3 \rightarrow$	$\{b_6, b_8\}$

block	rank
$b_1 \rightarrow$	0
$b_2 \rightarrow$	2
$b_3 \rightarrow$	0
$b_4 \rightarrow$	2
$b_5 \rightarrow$	1
$b_6 \rightarrow$	3
$b_7 \rightarrow$	1
$b_8 \rightarrow$	3

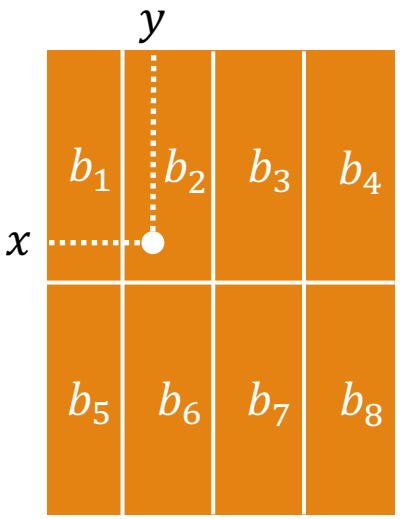
Given: $global(x, y)$, find $(rank, local_id)$

IDEA: use **rows interval tree** to locate a block containing $\boldsymbol{x}$ and use **columns interval tree** to local a block containing $\boldsymbol{y}$

Mapper

Columns Interval Tree

Rows Interval Tree



rank	blocks
$0 \rightarrow$	$\{b_1, b_3\}$
$1 \rightarrow$	$\{b_5, b_7\}$
$2 \rightarrow$	$\{b_2, b_4\}$
$3 \rightarrow$	$\{b_6, b_8\}$

block	rank
$b_1 \rightarrow$	0
$b_2 \rightarrow$	2
$b_3 \rightarrow$	0
$b_4 \rightarrow$	2
$b_5 \rightarrow$	1
$b_6 \rightarrow$	3
$b_7 \rightarrow$	1
$b_8 \rightarrow$	3

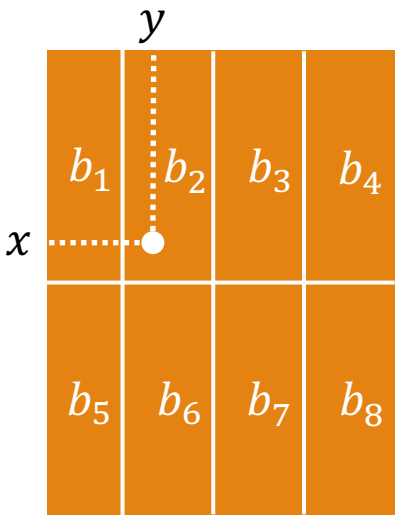
Given: $global(x, y)$, find $(rank, local_id)$ → $\boldsymbol{O(\log \#steps) = O(\log \log P)}$

IDEA: use **rows interval tree** to locate a block containing $\boldsymbol{x}$ and use **columns interval tree** to local a block containing $\boldsymbol{y}$

Mapper

Columns Interval Tree

Rows Interval Tree



rank	blocks
$0 \rightarrow$	$\{b_1, b_3\}$
$1 \rightarrow$	$\{b_5, b_7\}$
$2 \rightarrow$	$\{b_2, b_4\}$
$3 \rightarrow$	$\{b_6, b_8\}$

block	rank
$b_1 \rightarrow$	0
$b_2 \rightarrow$	2
$b_3 \rightarrow$	0
$b_4 \rightarrow$	2
$b_5 \rightarrow$	1
$b_6 \rightarrow$	3
$b_7 \rightarrow$	1
$b_8 \rightarrow$	3

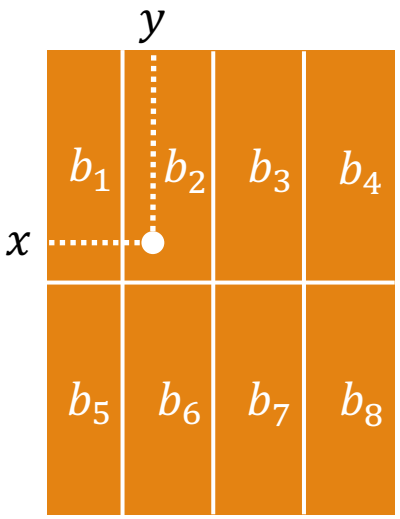
Given: $global(x, y)$, find $(rank, local_id)$ → $\boldsymbol{O(\log \#steps) = O(\log \log P)}$

Previous solution for powers of 2: → $\boldsymbol{O(\#steps)}$

Mapper

Columns Interval Tree

Rows Interval Tree



rank	blocks
$0 \rightarrow$	$\{b_1, b_3\}$
$1 \rightarrow$	$\{b_5, b_7\}$
$2 \rightarrow$	$\{b_2, b_4\}$
$3 \rightarrow$	$\{b_6, b_8\}$

block	rank
$b_1 \rightarrow$	0
$b_2 \rightarrow$	2
$b_3 \rightarrow$	0
$b_4 \rightarrow$	2
$b_5 \rightarrow$	1
$b_6 \rightarrow$	3
$b_7 \rightarrow$	1
$b_8 \rightarrow$	3

Given: $global(x, y)$, find $(rank, local_id)$ → $\boldsymbol{O(\log \#steps) = O(\log \log P)}$ ✓

Previous solution for powers of 2: → $\boldsymbol{O(\#steps)}$

Data Layout Summary

- Less data reshufflings needed, and if needed reshufflings are on the level of blocks
- More efficient and simpler mapper
- Could probably be reduced to Scalapack blocked layout which is more compatible with other linear-algebra libraries.

0	0	2	2	1	1	3	3
0	0	2	2	1	1	3	3
0	0	2	2	1	1	3	3
0	0	2	2	1	1	3	3
4	4	6	6	5	5	7	7
4	4	6	6	5	5	7	7
4	4	6	6	5	5	7	7
4	4	6	6	5	5	7	7

relabel ranks →

0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3
4	4	5	5	6	6	7	7
4	4	5	5	6	6	7	7
4	4	5	5	6	6	7	7
4	4	5	5	6	6	7	7

Main Improvements

- ✓ **Non-powers of 2:**
  - ✓ **Generalization:** algorithm generalization
  - ✓ **Asynchrony:** block only the subgroup of ranks
- ✓ **K-split:** implementation (requires reduction)
- ✓ **Data Layout:**
  - ✓ **New Layout:** introducing the new data layout
  - ✓ **Mapper:** finding the mapper $global(x, y) \leftrightarrow (rank, local_id)$
- **Any strategy:** allow any order of Sequential and Parallel steps

  (not necessarily all sequential steps must be before parallel steps)

Allowing any splitting strategy

- Originally, all Sequential steps had to be before all Parallel steps

$$Seq, Seq, Seq, Par, Par, Par, Par \ldots$$

- If Sequential and Parallel steps interleaved → complicated splitting schemes

$$Seq, Par, Seq, Par, Par, Seq, Par \ldots$$

- We want to be able to interleave step types because:

$$Interleaved\ Seq\ and\ Par\ steps$$

$First\ all\ Seq\ then\ all\ Par$ ← ●—●—●—●—●—●—●—● → $All\ Parallel\ steps$

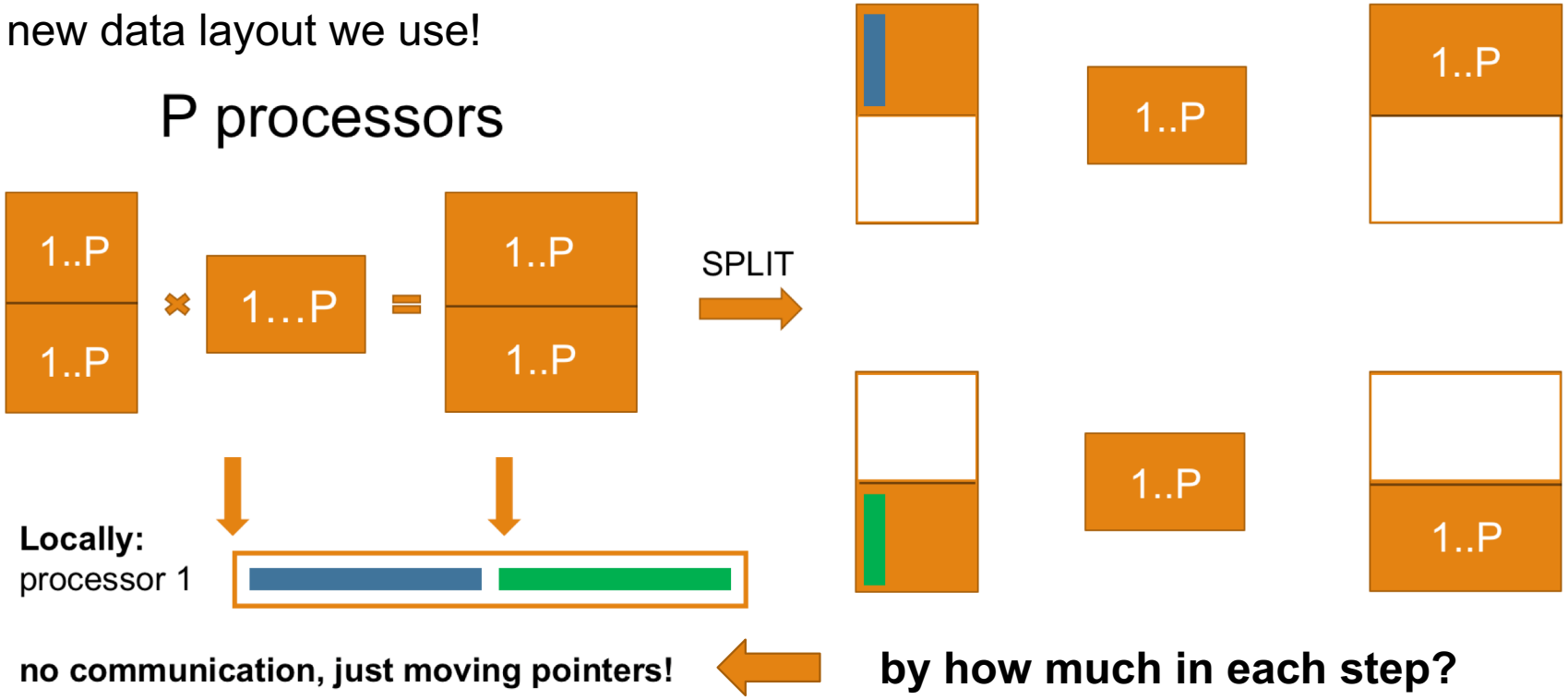
- Requires **least** memory
- Requires **most** communication

tradeoff between memory and comm

- Requires **most** memory
- Requires **least** communication

Sequential Steps Complications

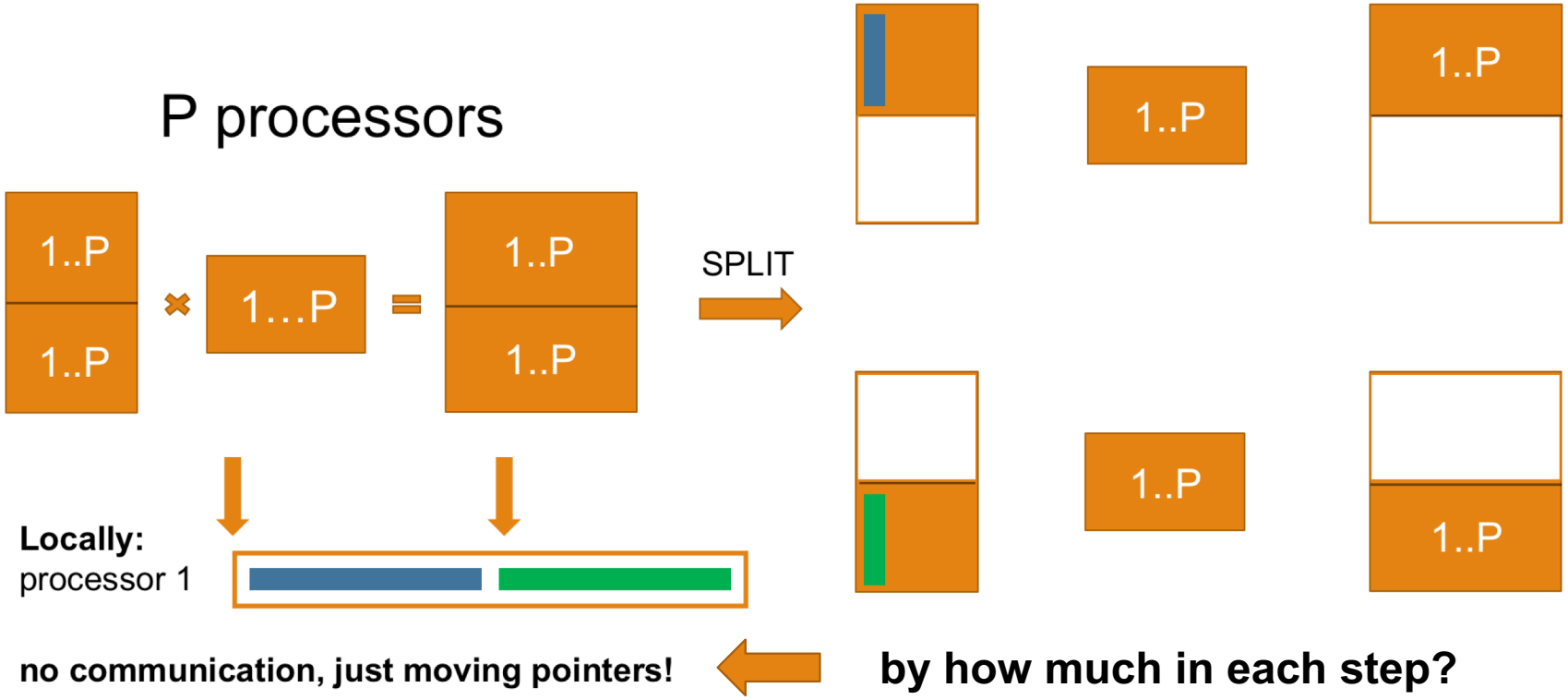
Specific to the new data layout we use!



- If interleaved steps allowed, two consequtive Seqential steps might consist of many nested and unequal substeps

- blocks might change their sizes inside the recursive call

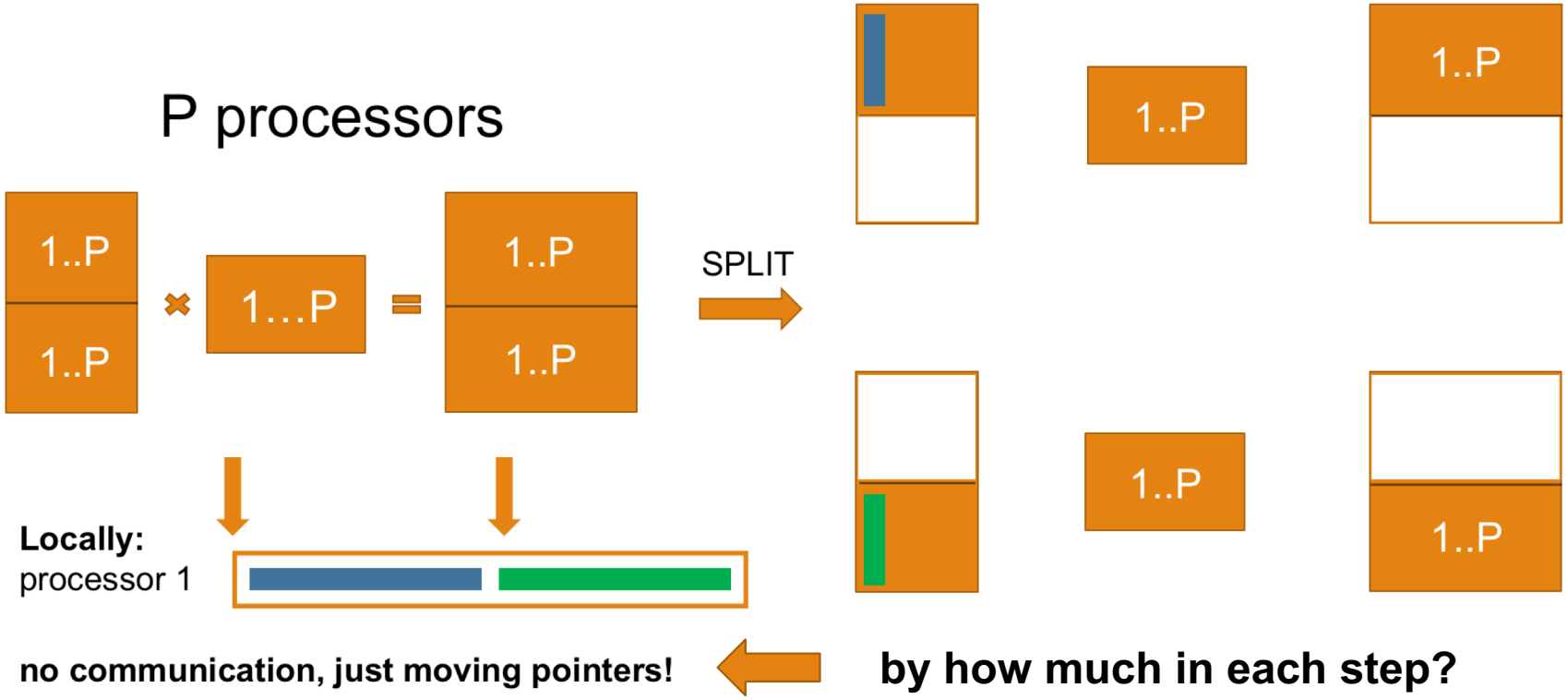
Sequential Steps Complications



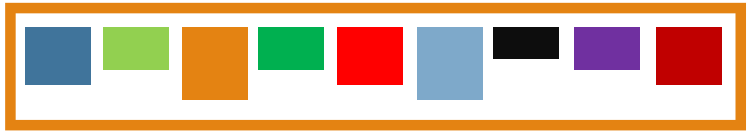
- If interleaved steps allowed, two consequtive Seqential steps might consist of many nested and unequal substeps

- blocks might change their sizes inside the recursive call

Sequential Steps Complications

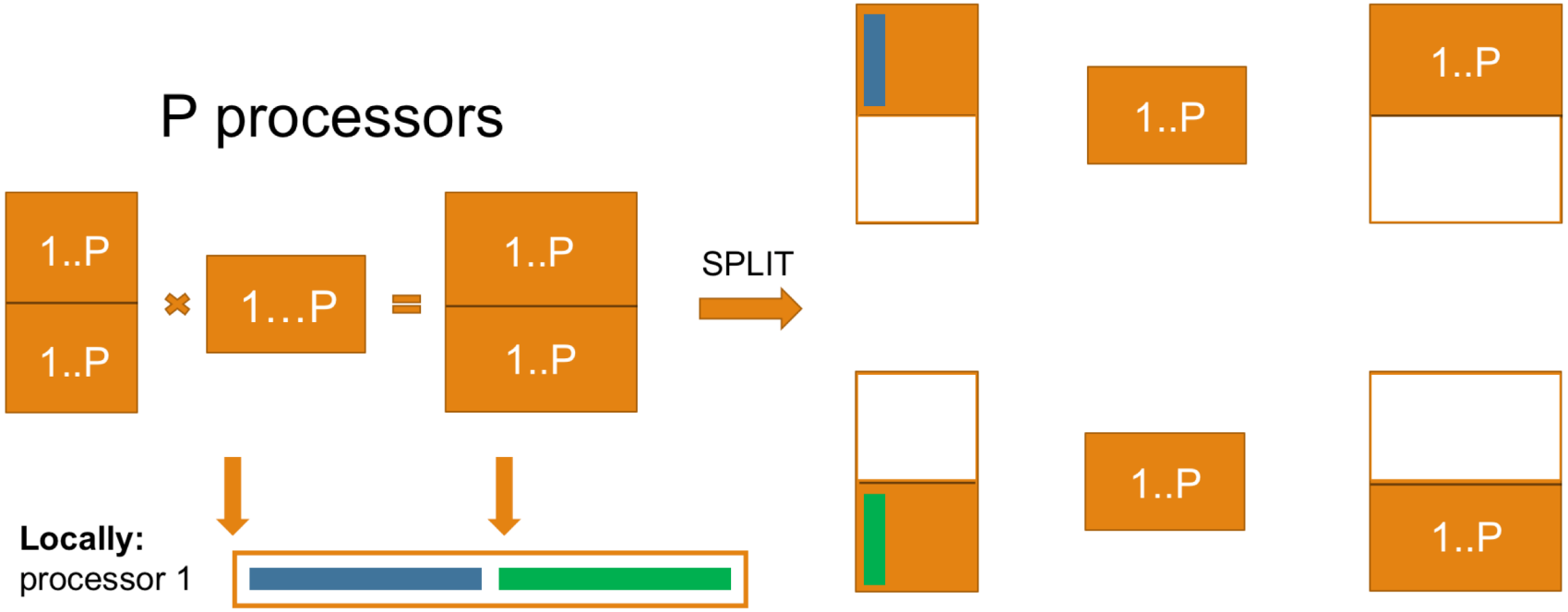


- If interleaved steps allowed, two consequtive Seqential steps might consist of many nested and unequal substeps



- blocks might change their sizes inside the recursive call

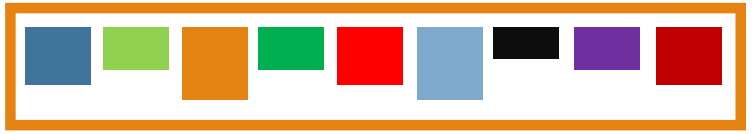
Sequential Steps Complications



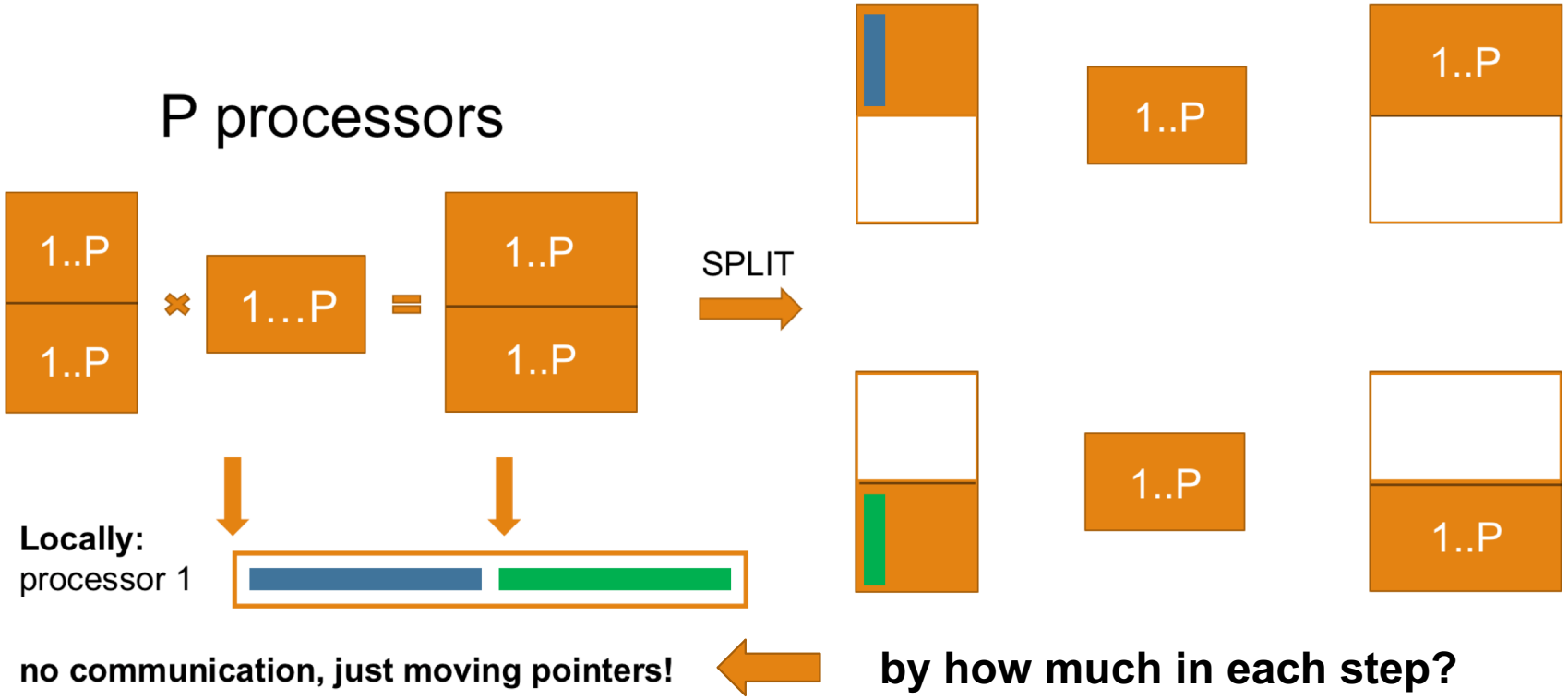
no communication, just moving pointers!

by how much in each step?

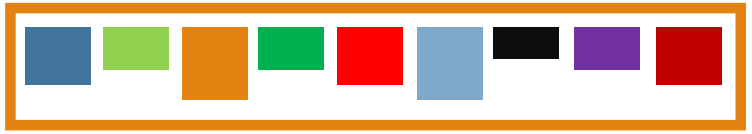
We cannot just jump to the next block!!!



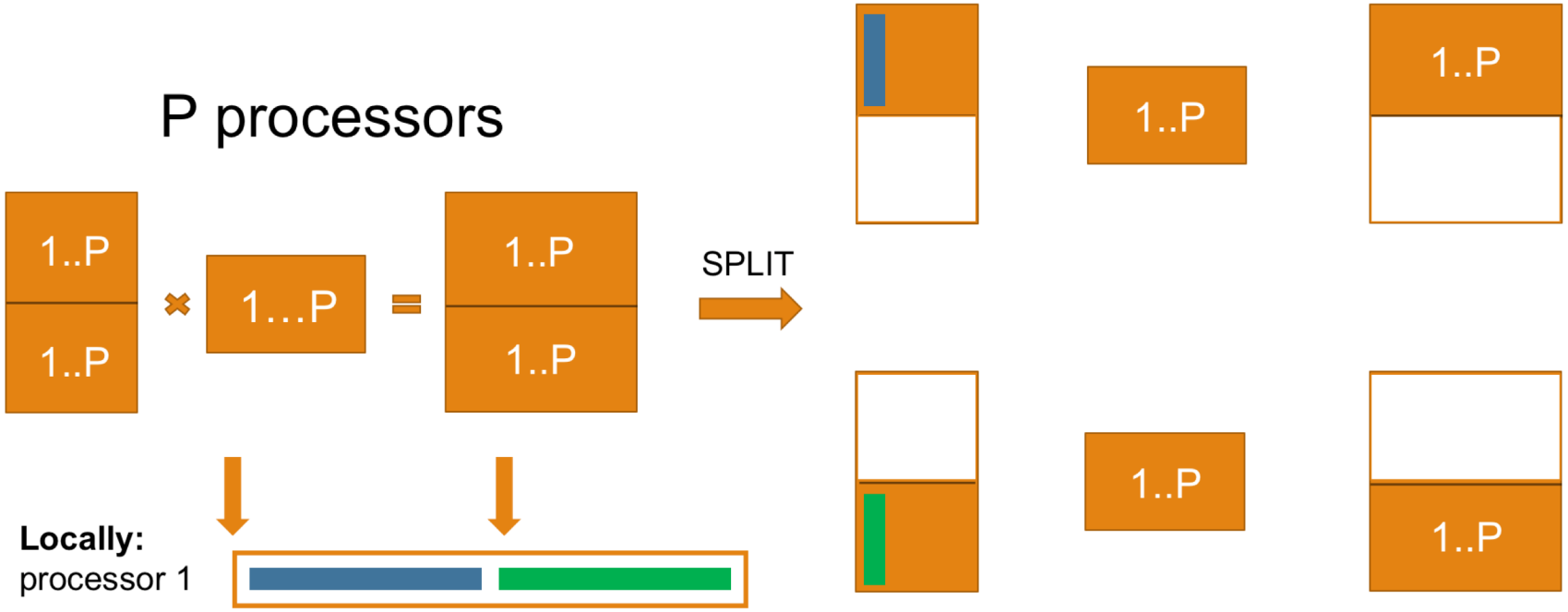
Sequential Steps Solution



We cannot just jump to the next block!!!



Sequential Steps Solution



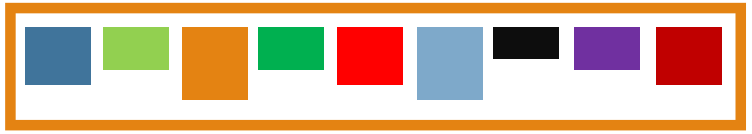
no communication, just moving pointers!

by how much in each step?

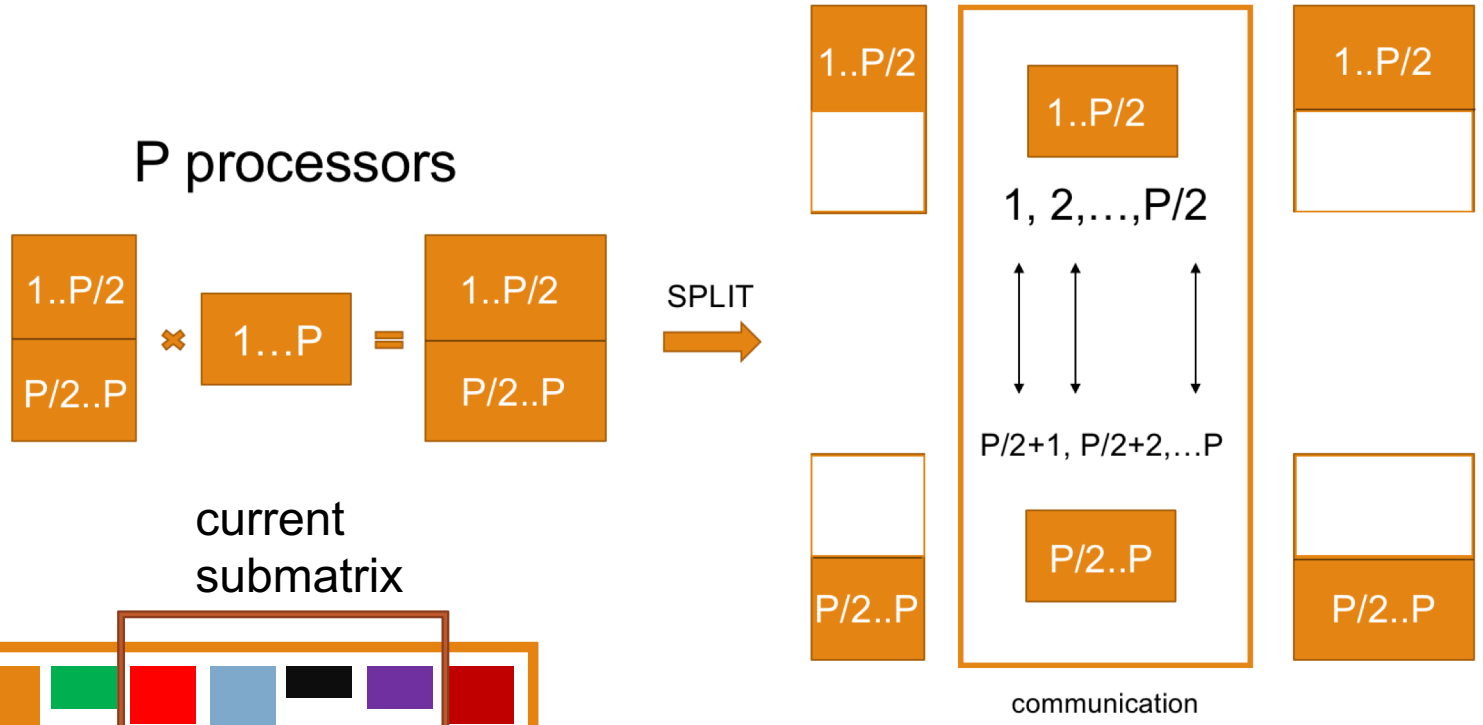
Introduce relation:

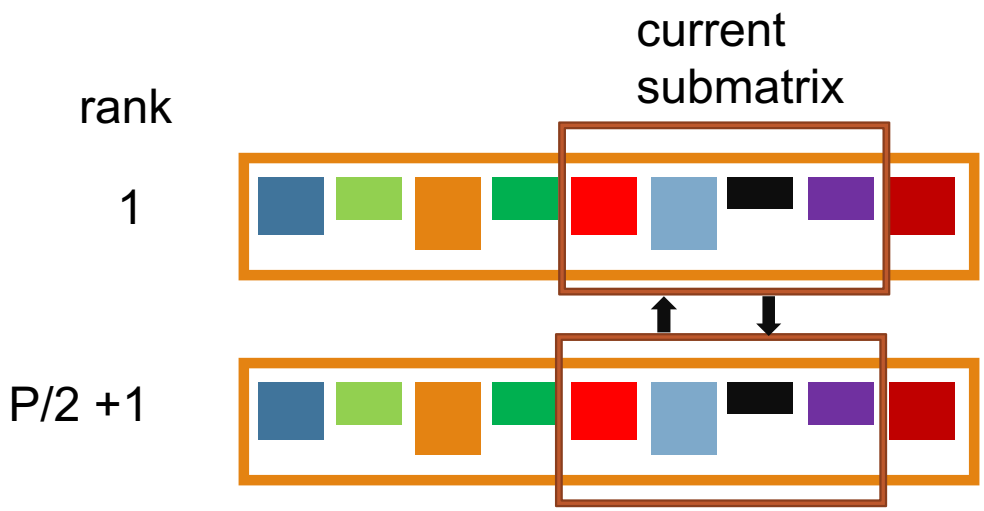
$$\text{block}_i < \text{block}_j \Leftrightarrow \frac{\text{block}_i}{\text{block}_j} \quad \vee \quad \text{block}_i | \text{block}_j$$

Skip all blocks less than the current submatrix



Parallel Steps Complications





Introduce relation:
$\text{block}_i \in \text{block}_j \Leftrightarrow \text{block}_i$ inside block_j

Copy only the blocks inside the current submatrix

CARMA NOW

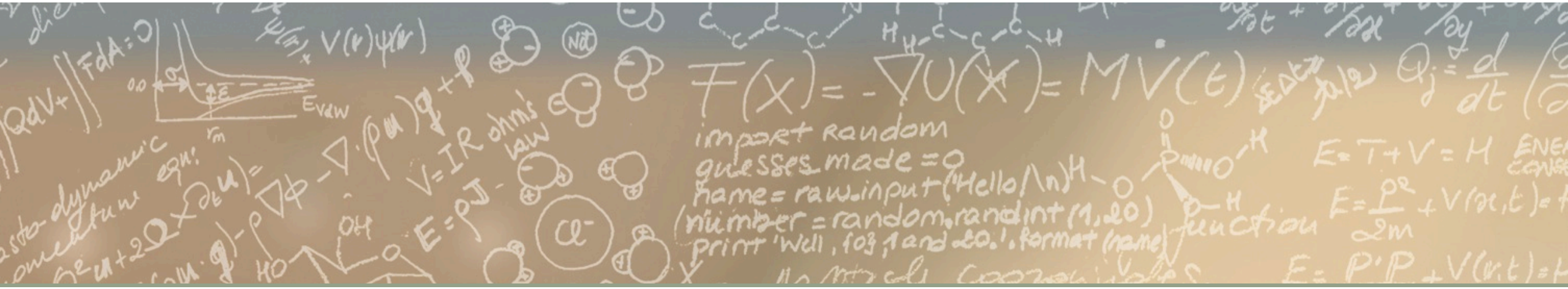
- ✓ **Non-powers of 2:**
  - ✓ **Generalization:** algorithm generalization
  - ✓ **Asynchrony:** block only the subgroup of ranks
- ✓ **K-split:** implementation (requires reduction)
- ✓ **Data Layout:**
  - ✓ **New Layout:** introducing the new data layout
  - ✓ **Mapper:** finding the mapper $global(x, y) \leftrightarrow (rank, local_id)$
- ✓ **Any strategy:** allow any order of Sequential and Parallel steps

  (not necessarily all sequential steps must be before parallel steps)

Further Research

- **Find optimal strategy**
  - **Default Strategy seems best**
  - **But it also depends on dgemm**

- **Make it completely asynchronous**



Thank you for your attention.