diBELLA 2D

Parallel Algorithms for De Novo Genome Assembly

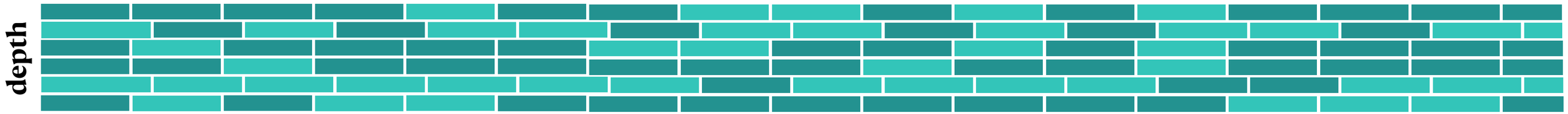
Giulia Guidi —February 25th, 2021
CS267

De Novo Genome Assembly Stages

The knowledge of the **complete DNA makeup** of an organism is fundamental to provide the most detailed resolution of genetic and epigenetic variations

Wheat Genome ~17G nucleotides

Reads ~100-10k nucleotides



Read: short fragment of DNA sequence that can be read by a DNA sequencing technology – **can't read whole DNA in one go** and each region of the genome is sequenced **multiple times** (depth)

***De novo* genome assembly**: reconstruct an unknown genome from a set of reads

De Novo Assembly Is Hard!

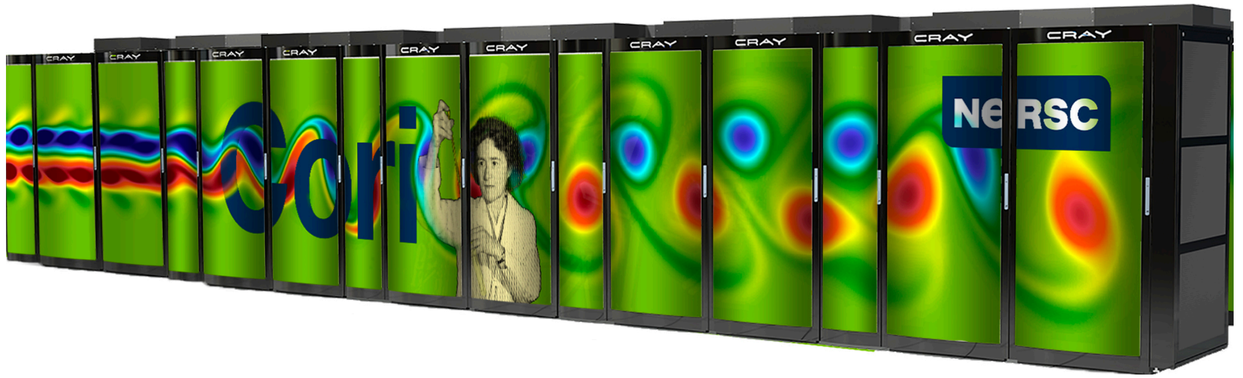
- There is **no reference genome**: In principle we want to reconstruct unknown genome sequence
- Reads are **significantly shorter** than whole genome: Average length from 100 to 15K bases
- Genomes have **repetitive regions**: Repetitive regions increase genome complexity
- Reads include **errors**: Error rates from 0.5% up to 15%

..And Genomes Are Big!

Current state-of-the-art shared-memory software can handle typically handle human and microbial genomes — However, **plant genomes and metagenomes can easily generate many GB or TB of data** which is infeasible to process even on beefy nodes

A distributed memory *de novo* genome assembler **enables characterization of big and complex genomes** as well **reducing the runtime** of human-like size genomes **exploiting higher degree of parallelism** coming from many core machines which potential **impact on many sciences:**

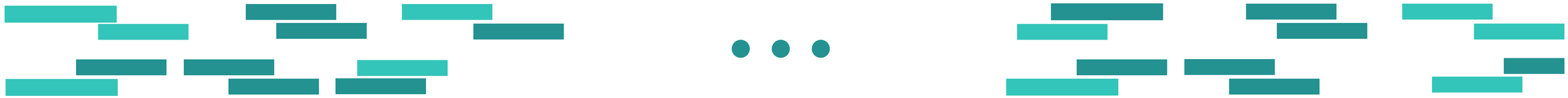
- **Energy**
- **Health and medicine**
- **Environment**



Long Read Assembly Paradigm

diBELLA 2D focuses on **long read sequencing data** and the most common paradigm to assemble long read data is called **Overlap-Layout-Consensus paradigm** (OLC)

Overlap

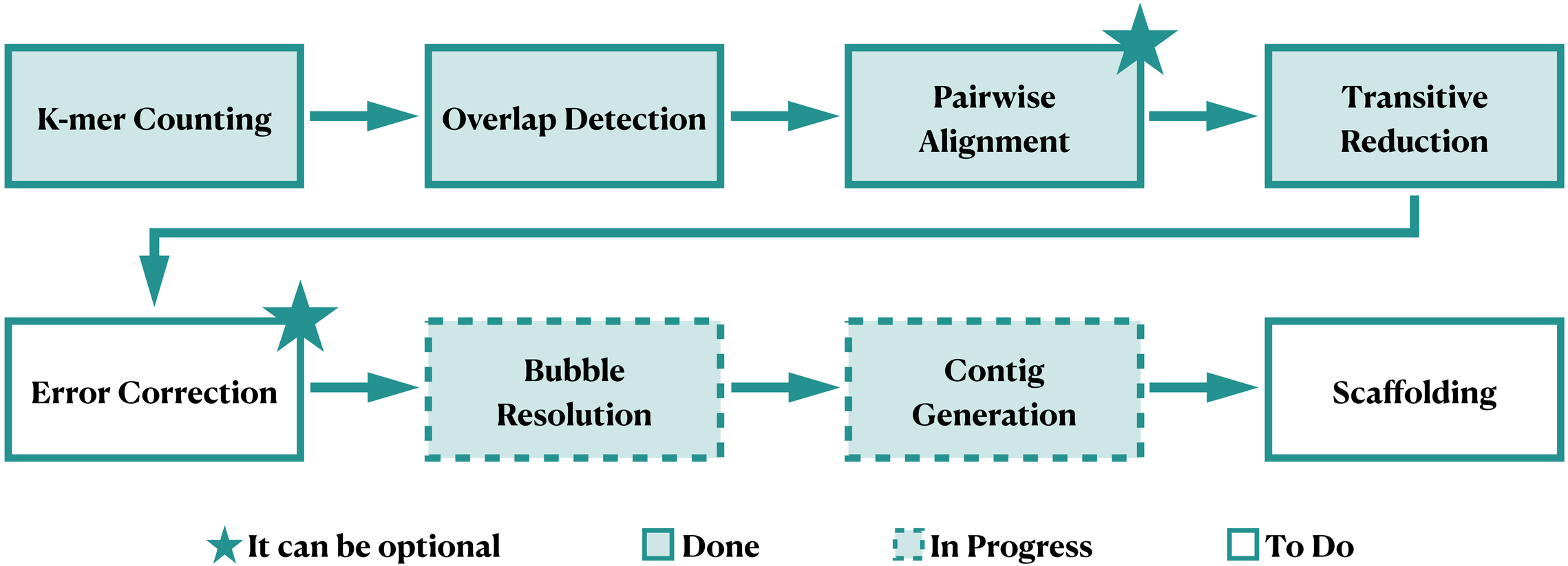


Layout

Consensus

Assembly Paradigm In Detail

And diBELLA 2D Roadmap [3]



K-mer Counting

An Overview

A k-mer is a **substring of fixed length *k***, let's say $k = 4$

TTCAATTTCAAT

TTCA 2

TCAA 2

CAAT 2

AATT 1

ATTT 1

TTTC 1

Based on error rate and coverage of the data set, we choose the optimal k-mer length and k-mer frequency interval (See [1])

Common sizes for 15% error rate long read data are $k = [15\text{-}20]$ while for 0.5% error rate long read data we usually have $k = [30\text{-}50]$

These k-mers are used to **find matches between the sequences** in a dataset:

TTCAATTT**CAAT**
CAATGACATTAC

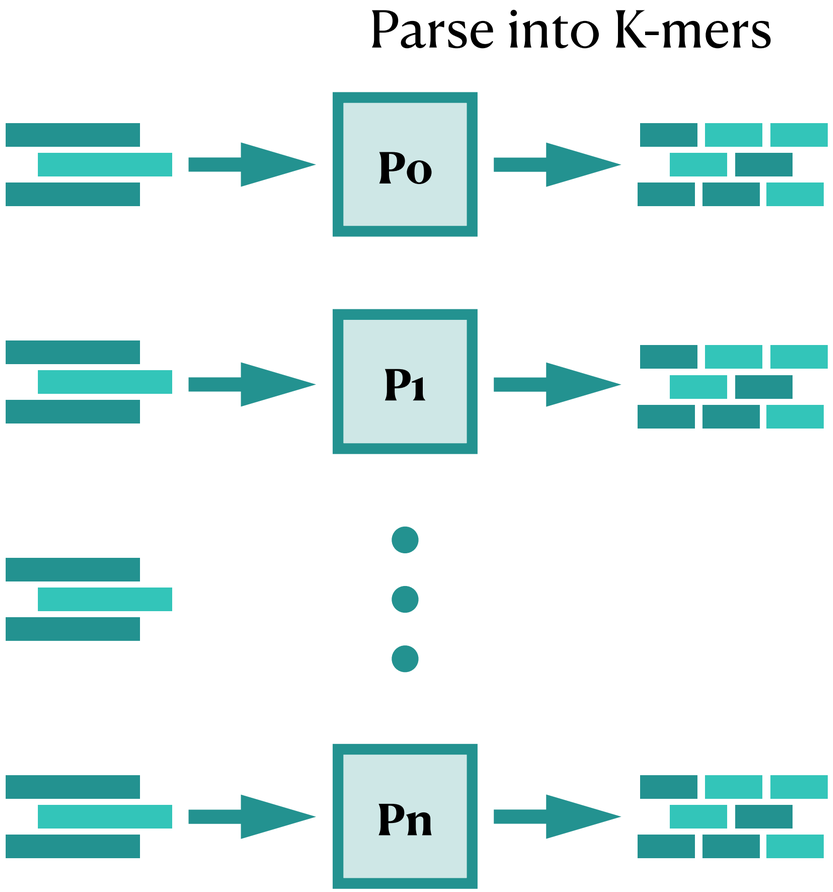
K-mer Counting

An Overview of the First Pass (MPI Only) [2]

Input Sequences

K-mer Counting

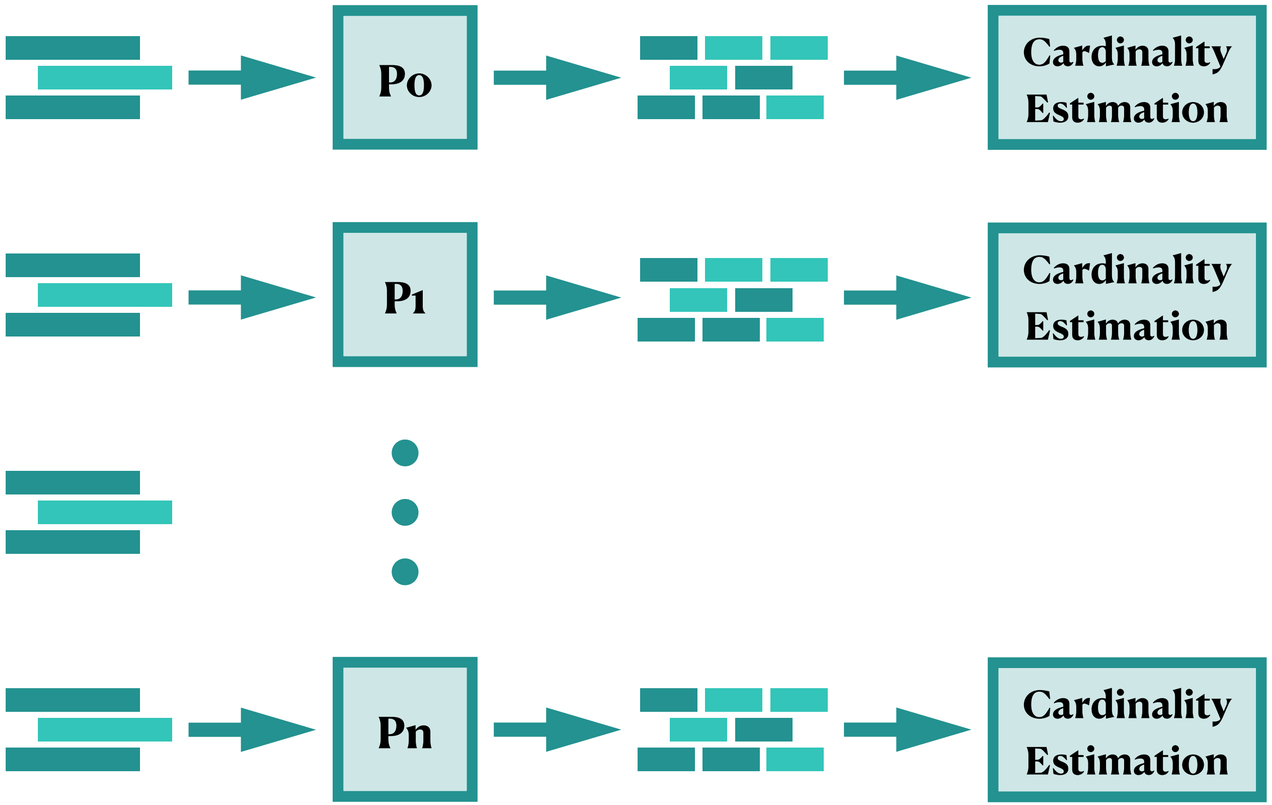
An Overview of the First Pass (MPI Only) [2]



K-mer Counting

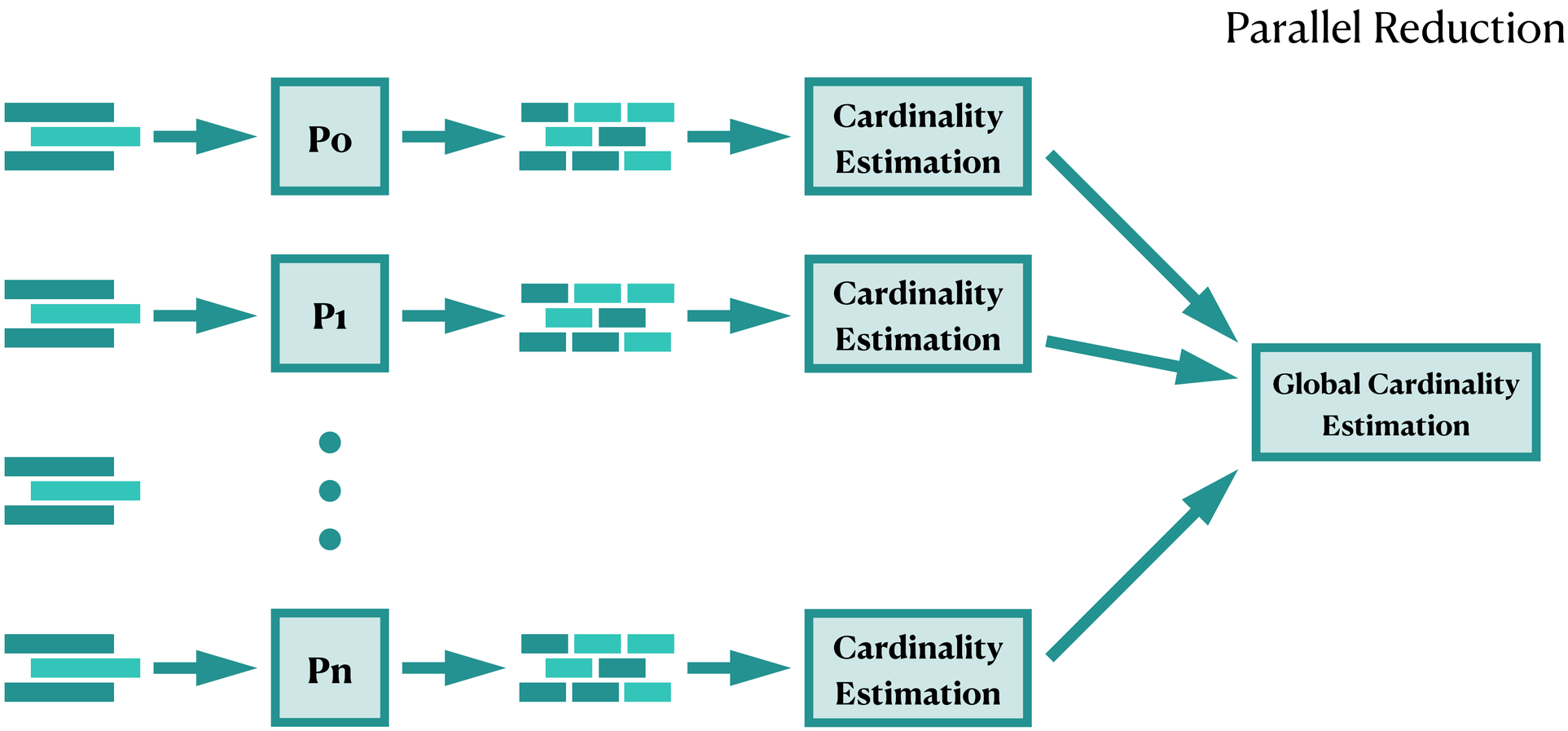
An Overview of the First Pass (MPI Only) [2]

K-mers Cardinality Estimation



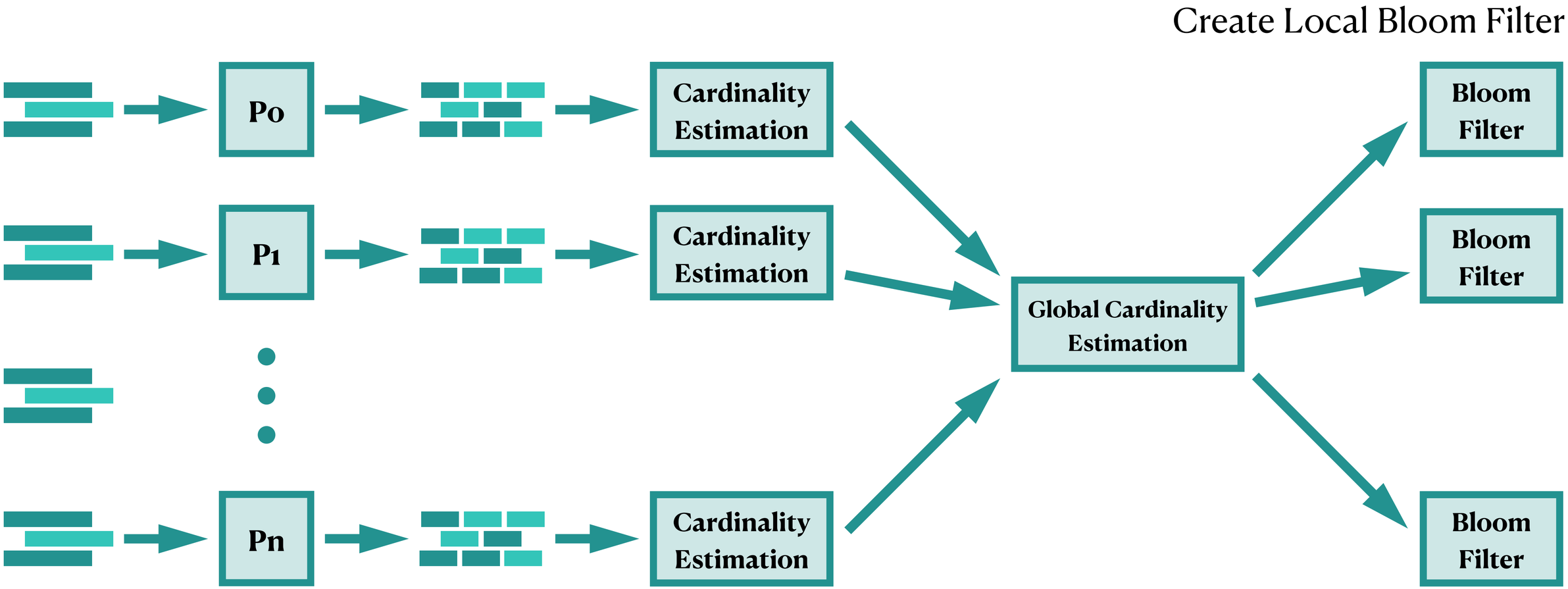
K-mer Counting

An Overview of the First Pass (MPI Only) [2]



K-mer Counting

An Overview of the First Pass (MPI Only) [2]



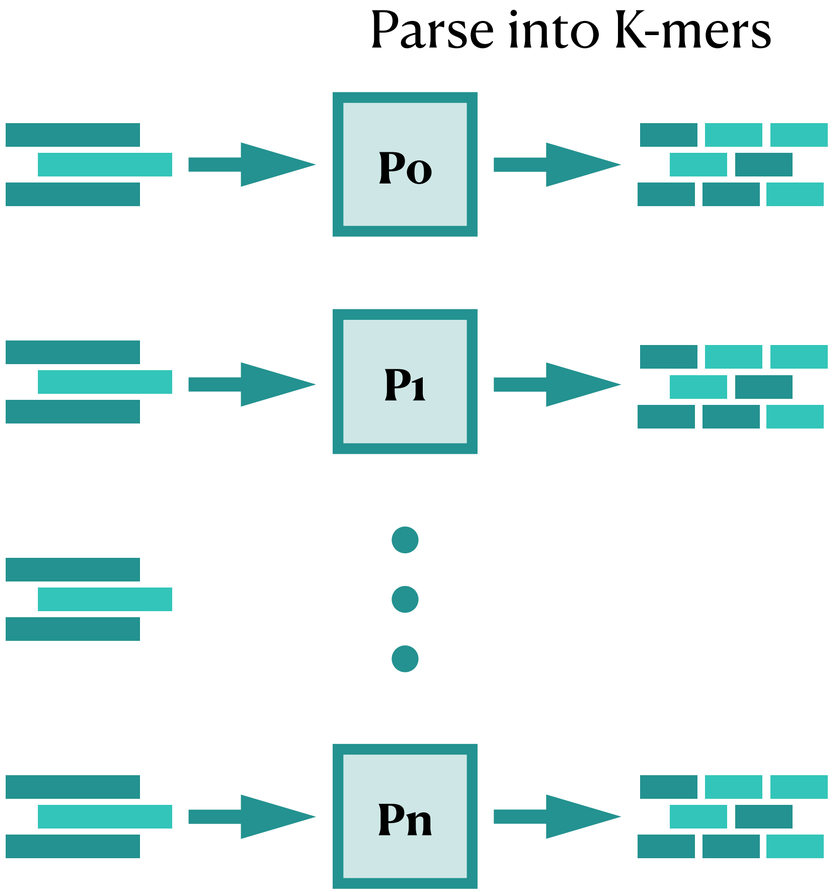
K-mer Counting

An Overview of the Second Pass (MPI Only) [2]

Input Sequences

K-mer Counting

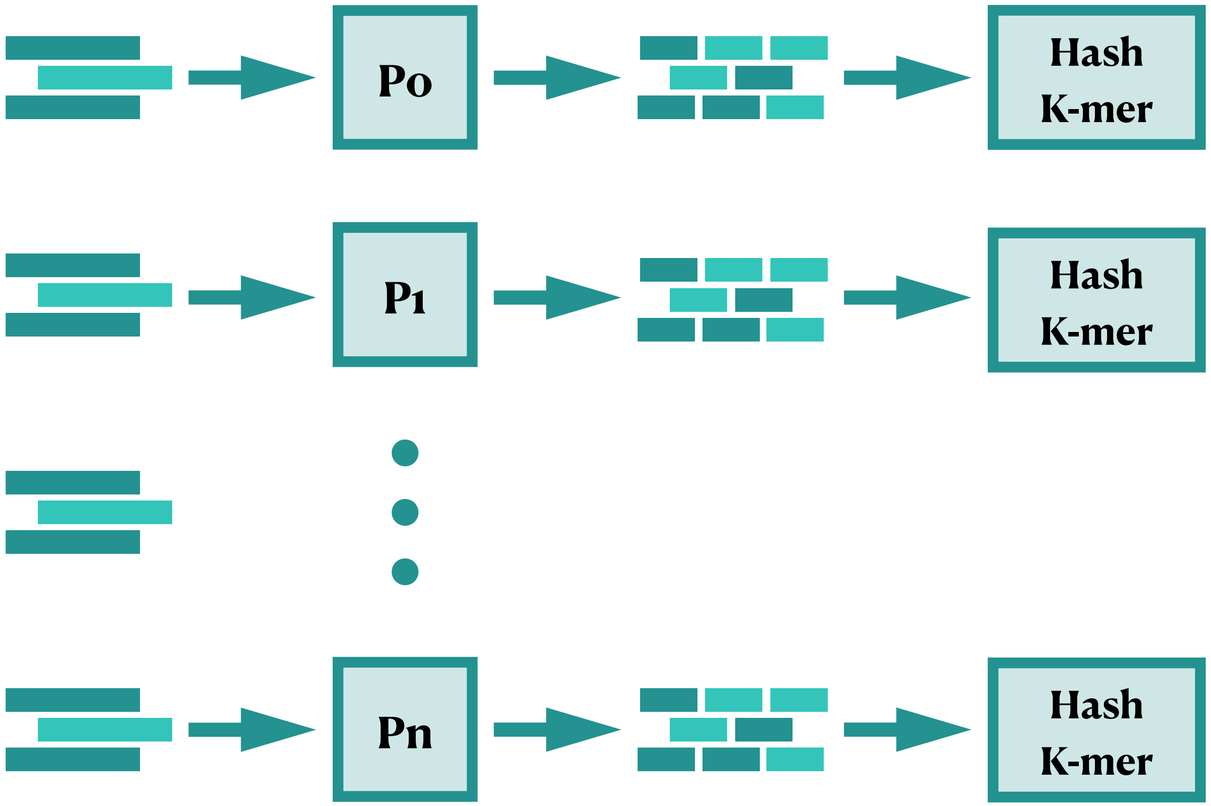
An Overview of the Second Pass (MPI Only) [2]



K-mer Counting

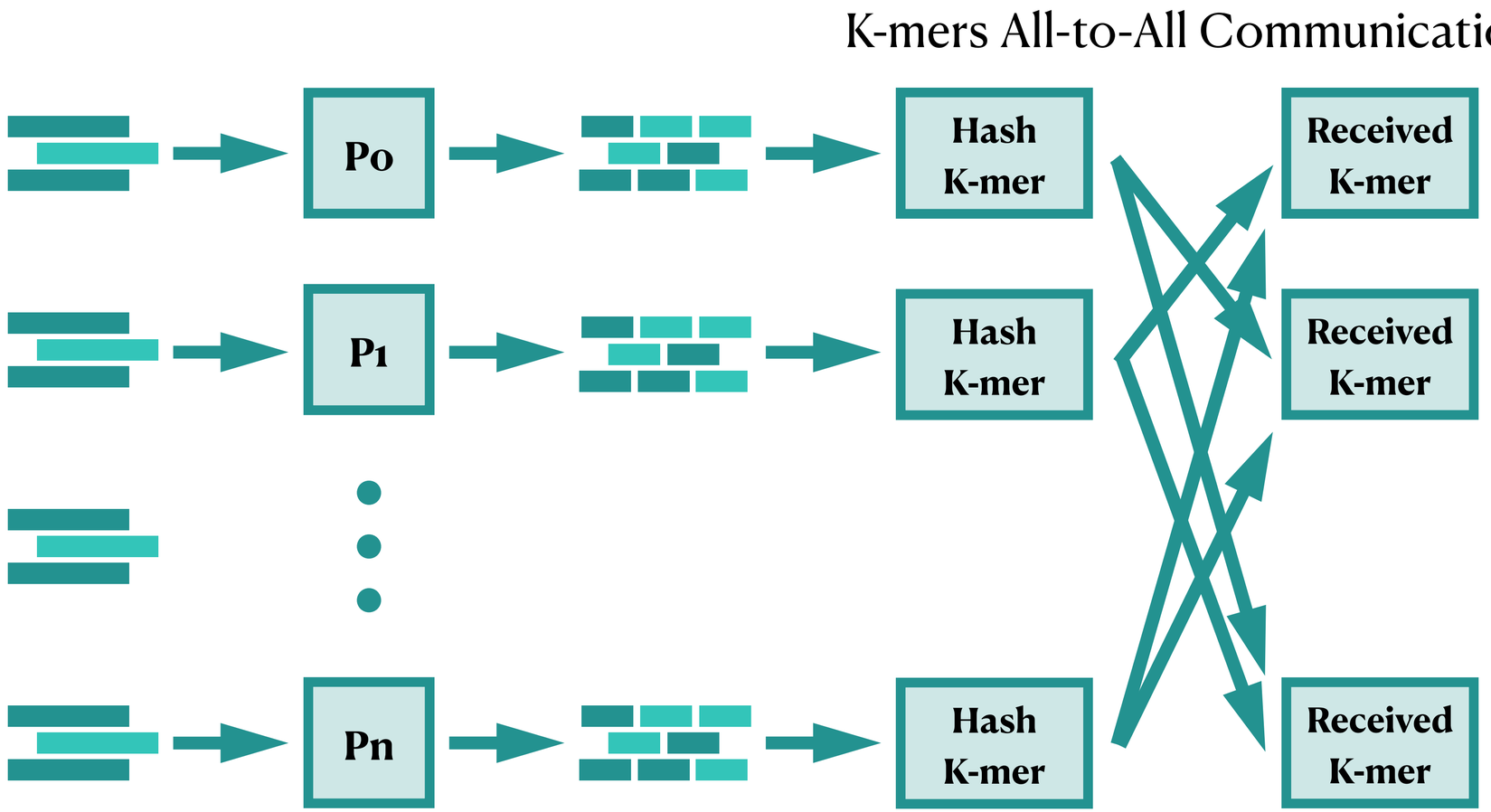
An Overview of the Second Pass (MPI Only) [2]

Hash K-mers and Find Owners



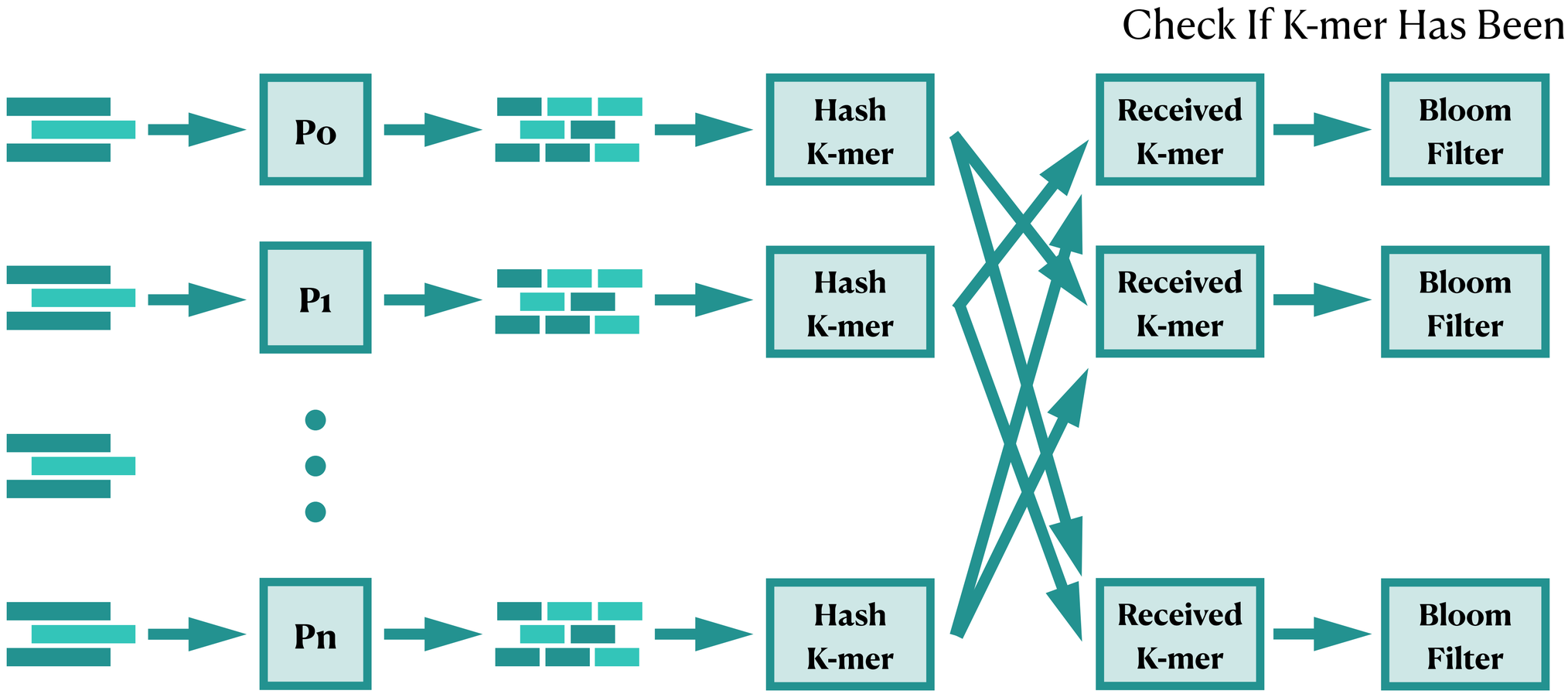
K-mer Counting

An Overview of the Second Pass (MPI Only) [2]



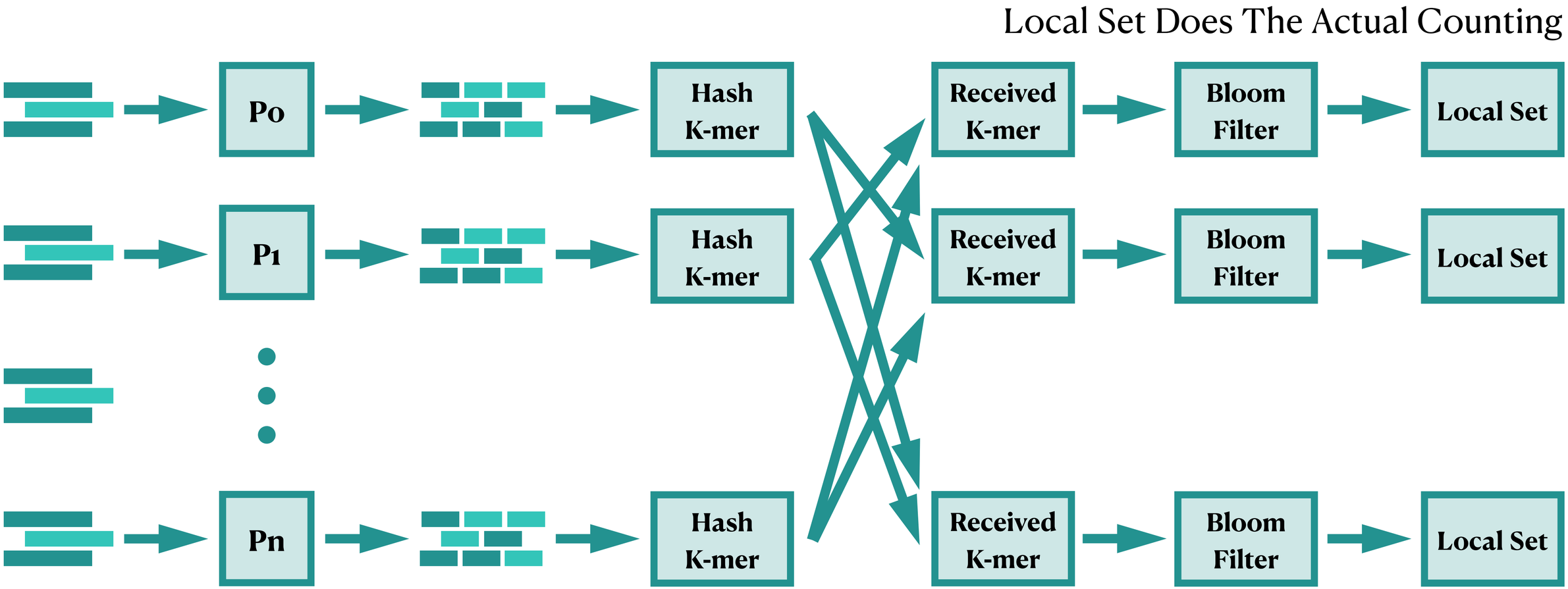
K-mer Counting

An Overview of the Second Pass (MPI Only) [2]



K-mer Counting

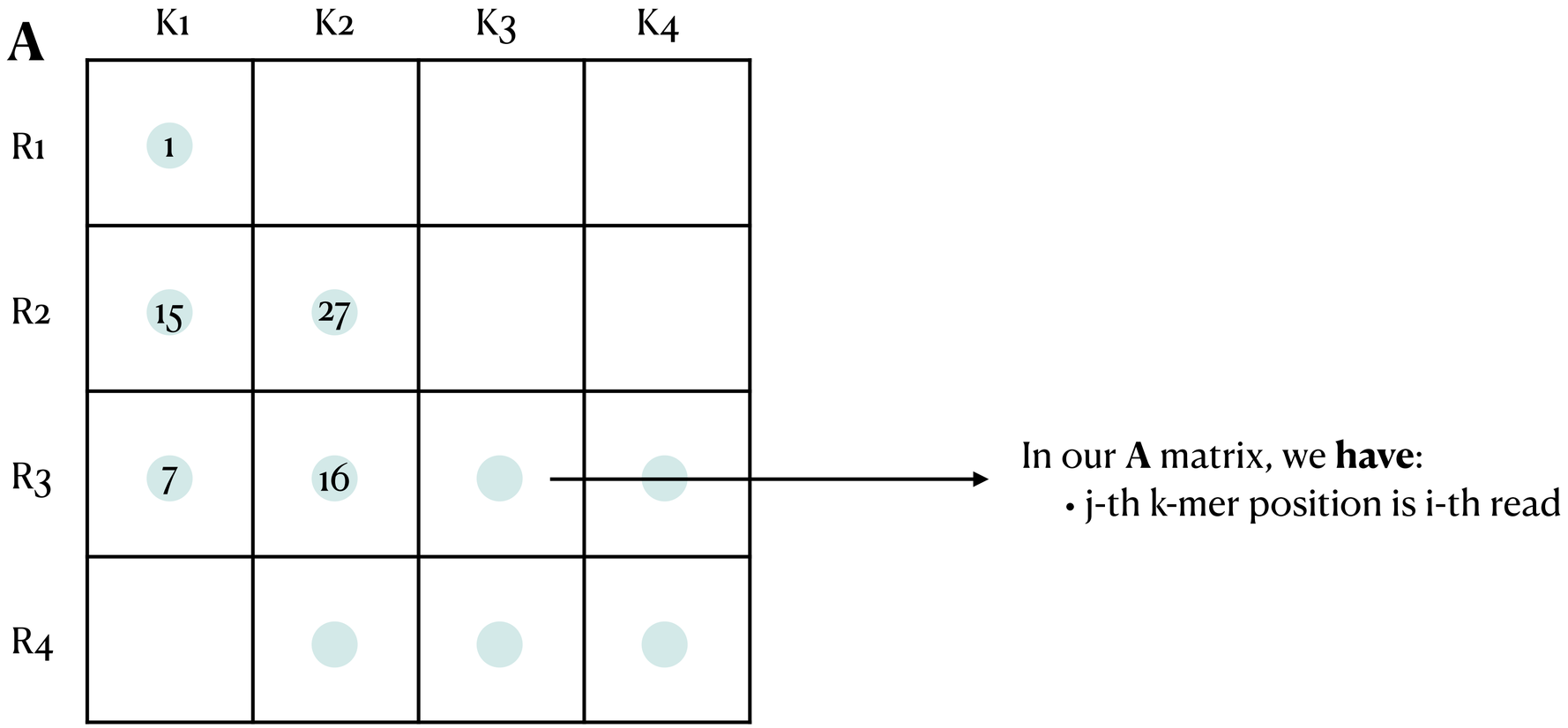
An Overview of the Second Pass (MPI Only) [2]



Overlap Detection

An Overview of Our 2D Algorithm (MPI + OpenMP) [3, 6]

Our |*sequences*|-by-|*k-mer*| matrix:



Overlap Detection

An Overview of Our 2D Algorithm (MPI + OpenMP)

Our |*sequences*|-by-|*k-mer*| matrix:

A

	K1	K2	K3	K4
R1	1			
R2	15	27		
R3	7	16		
R4				

Our |*k-mer*|-by-|*sequence*| matrix:

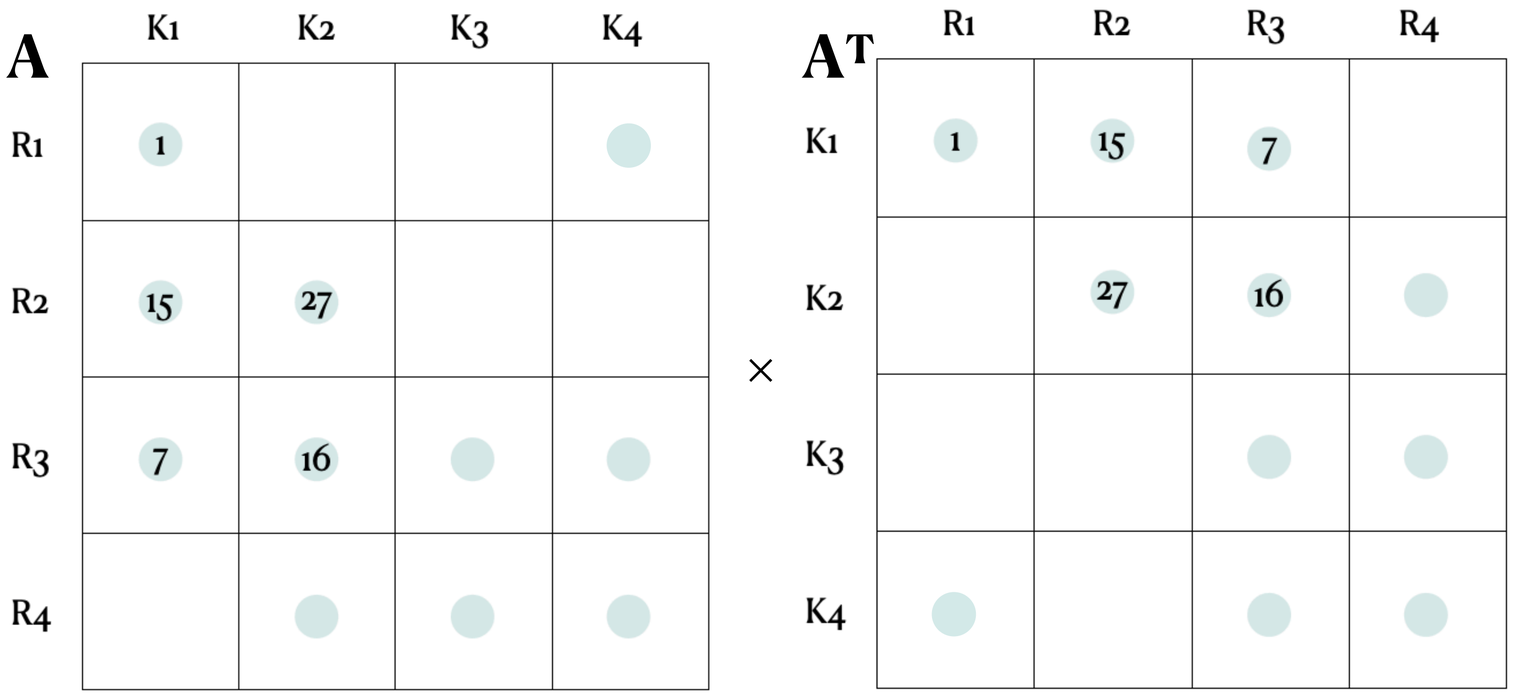
A^T

	R1	R2	R3	R4
K1	1	15	7	
K2		27	16	
K3				
K4				

Overlap Detection

An Overview of Our 2D Algorithm (MPI + OpenMP)

1. AA^T with *custom semiring* to identify the common k-mers to obtain matrix C



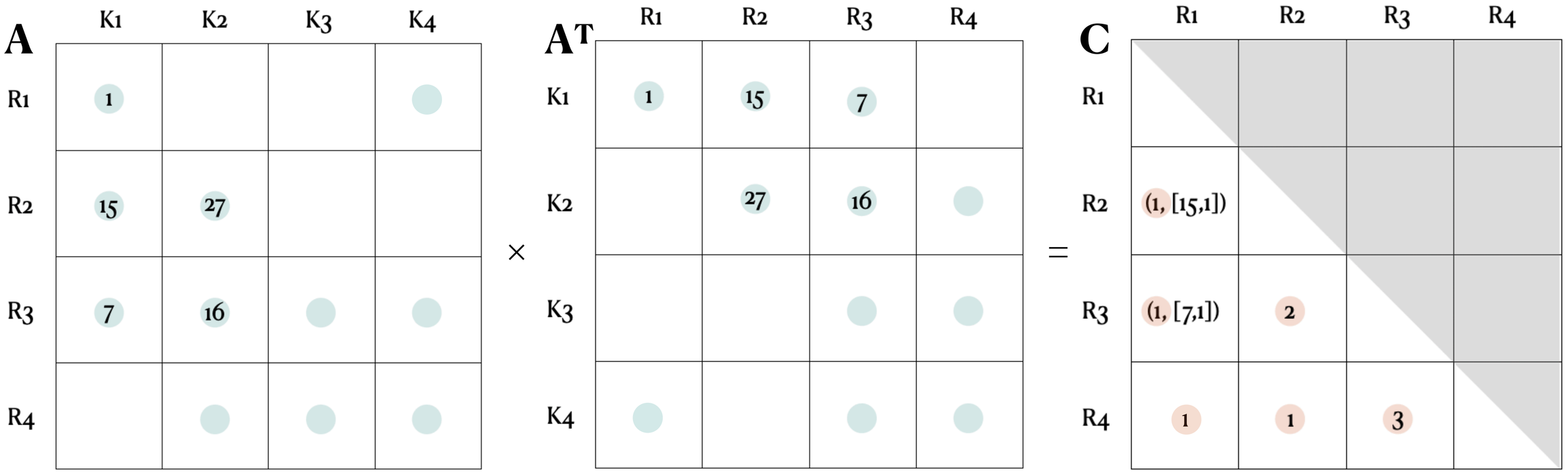
For this multiplication, our **custom multiply is an *assign* operation** and **add is a *add and concatenate* operation**

Using our custom semiring, we identify the common k-mers to obtain the **candidate overlap matrix C**

Overlap Detection

An Overview of Our 2D Algorithm (MPI + OpenMP)

1. AA^T with *custom semiring* to identify the common k-mers to obtain matrix C

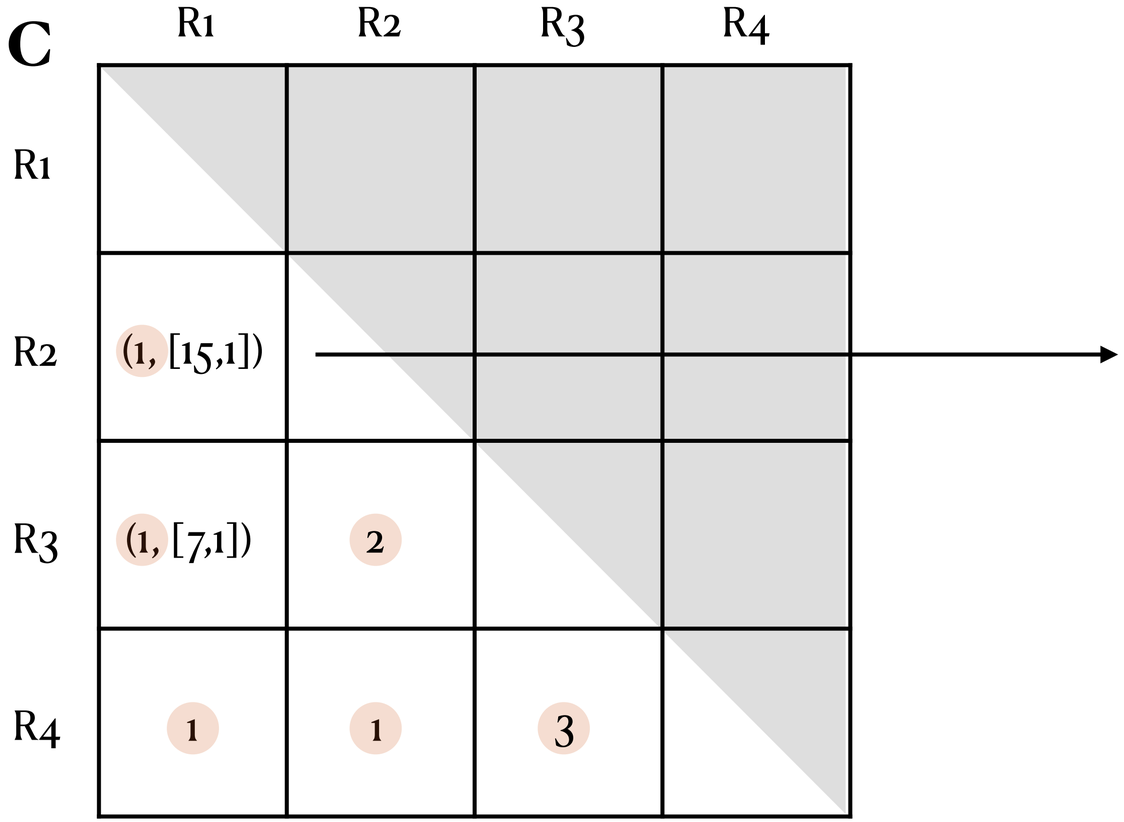


Using our custom semiring, we identify the common k-mers to obtain the **candidate overlap matrix C**

Overlap Detection

An Overview of Our 2D Algorithm (MPI + OpenMP)

Our |*sequences*|-by-|*sequence*| candidate overlap matrix C:



In our **C** matrix, we **have**:

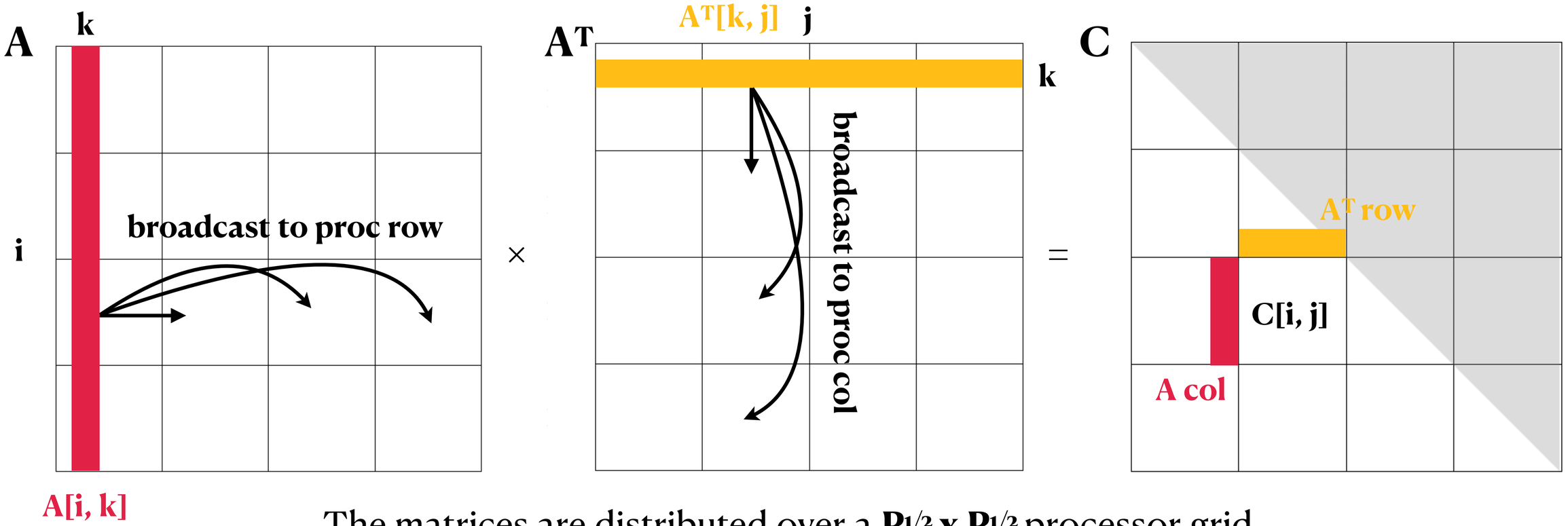
- The number of common k-mers and their positions in the corresponding sequences

In practice the whole matrix is computed but **only the lower triangular entries need pairwise alignment**

Overlap Detection

An Overview of Our 2D Algorithm (MPI + OpenMP) [4, 5]

The distributed implementation uses CombBLAS's Sparse SUMMA algorithm

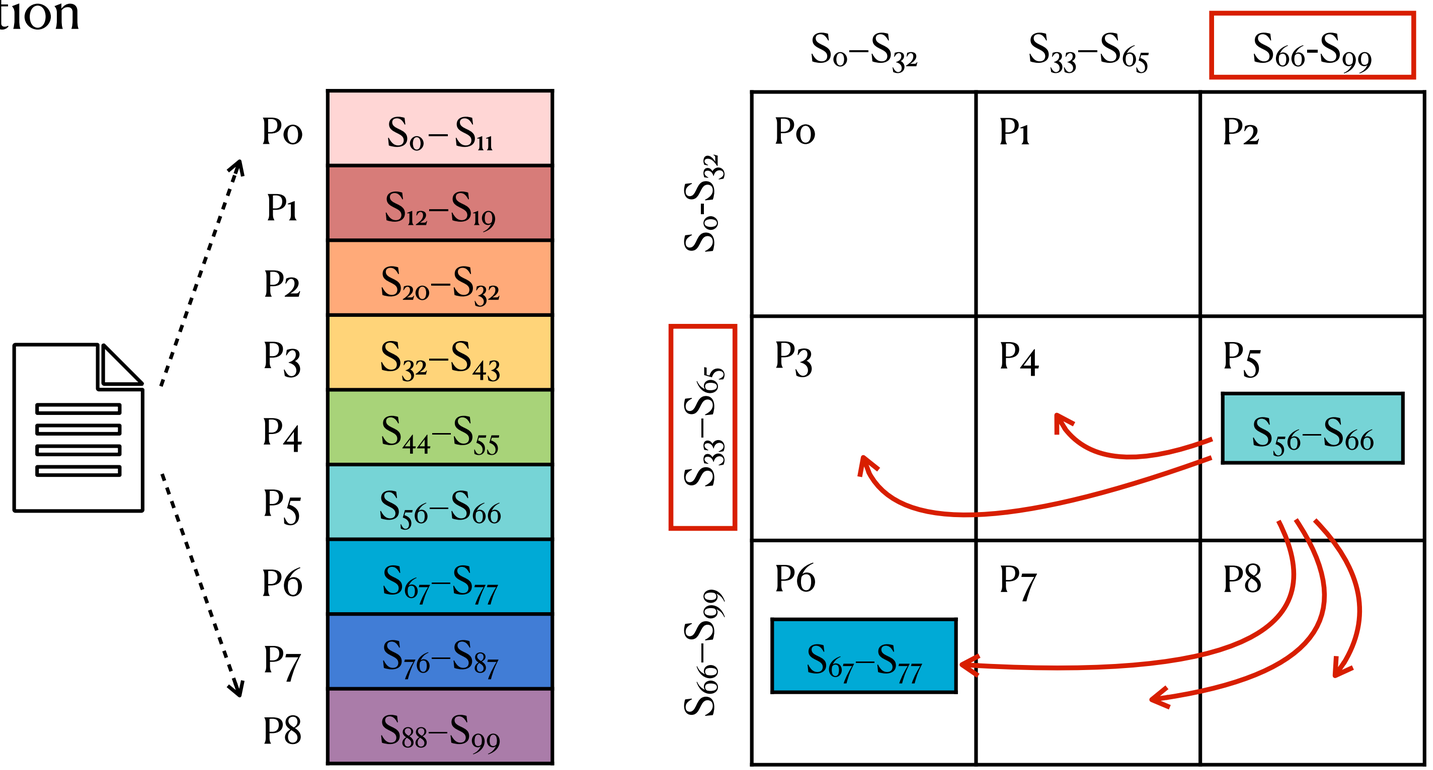


The matrices are distributed over a $P^{1/2} \times P^{1/2}$ processor grid

Sequence Exchange

An Overview of Our 2D Algorithm

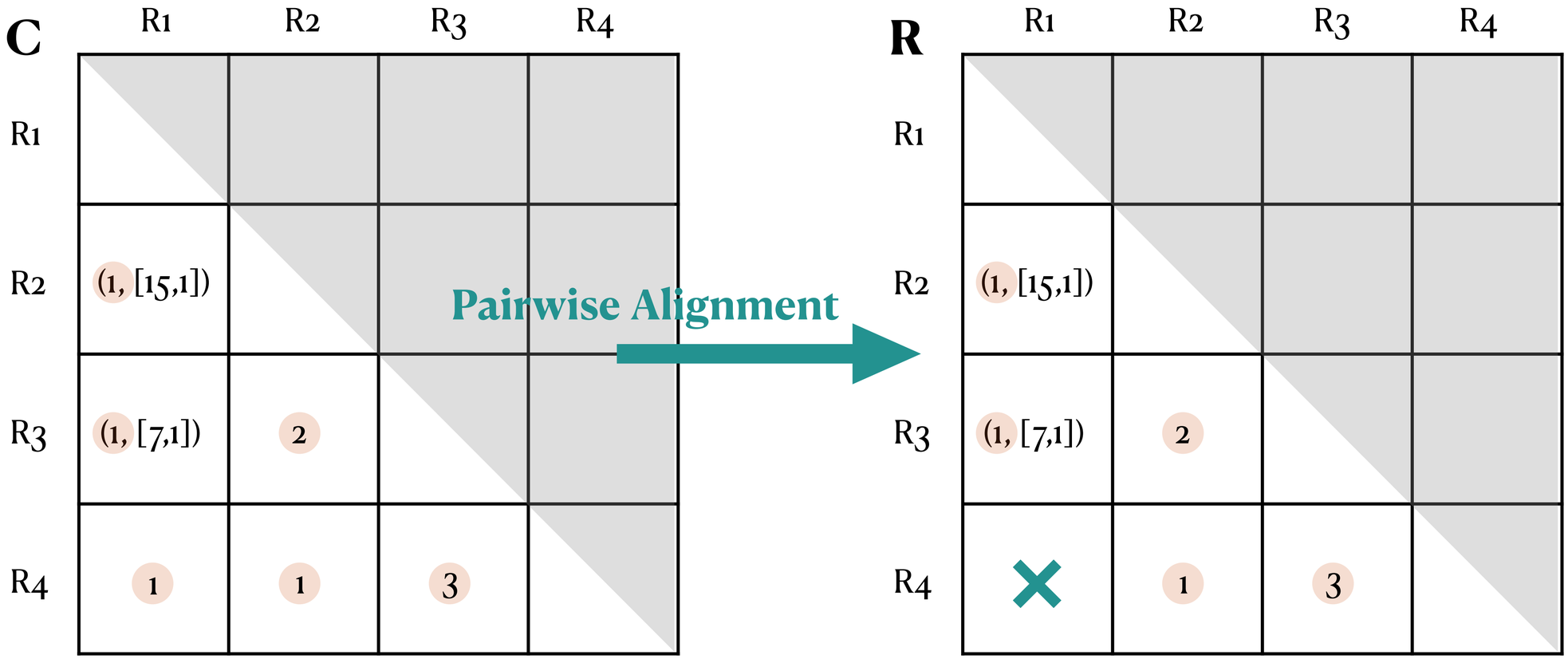
Each processor requests the full range of sequences it might need once the FASTA input file is read from disk — i.e. the sequence exchange begins as soon as we read the input and it overlaps with the SpGEMM computation



Pairwise Alignment

An Overview of Our 2D Algorithm (MPI + OpenMP)

From candidate overlap matrix C to the resulting matrix R post-alignment:



Pairwise Alignment

The kernel [9]

Pairwise alignment is used to **identify regions of similarity** between two biological sequences, it is implemented through **dynamic programming** and it takes **O(mn)** time where m,n are the length of the two sequences to align

diBELLA 2D uses a type of pairwise alignment called seed-and-extend: It starts the alignment from a seed (k-mer) match and extends this seed left and right until it either reaches the end of the sequences or the alignment score falls **X** below the best score seen up to that point

Initial State:

TTCAATTTCAATTGGACTAGCGA
TTCAATTGGAGTAGCATTGACAGGGA

Post Alignment:

TTCAATTTCAATTGGACTAGCGA
| | | | | | | | | | | | | | | |
TTCAATTGGAGTAGC—ATTGACAGGGA

Alignment Score: **13**
If we assume +1 match, -1 mismatch/gap

Project Idea 3: Implement work stealing CPU/GPU for hybrid alignment

Transitive Reduction

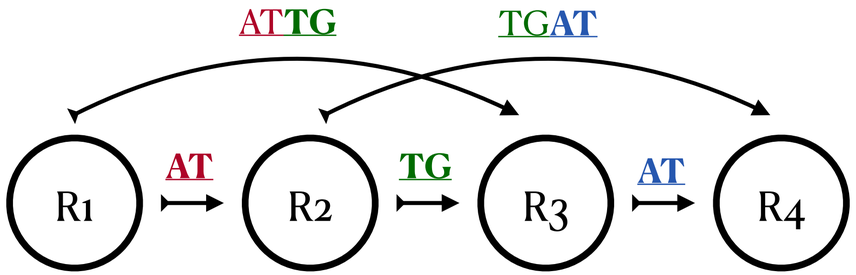
String Graph Definition

The next is to transitively reduce the post-alignment matrix R. To do this, we need the matrix in "string graph" form —**But what's a string graph?**

Set of **overlapping** reads:

R1 TTCAATCGATAC
R2 CAATCGATAC**AT**
R3 ATCGATAC**AT****TG**
R4 CGATACAT**TG****AT**

→ Overhang (**suffix** in *this* case but reality —obviously— is a bit more complex)



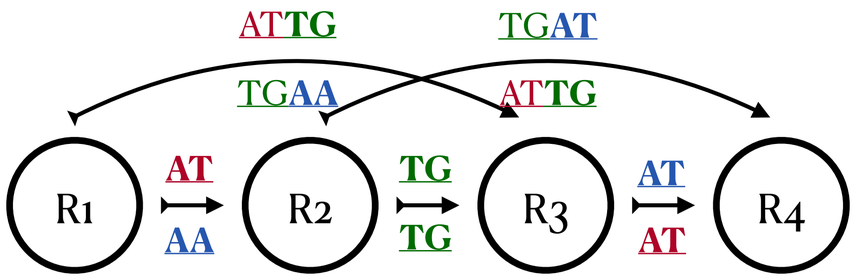
Transitive Reduction

String Graph Definition

Our goal is *retaining as much useful information as we can*, therefore we want to remove the redundant edges with *longer overhangs*

→ Overhang

↦ Bidirected edge



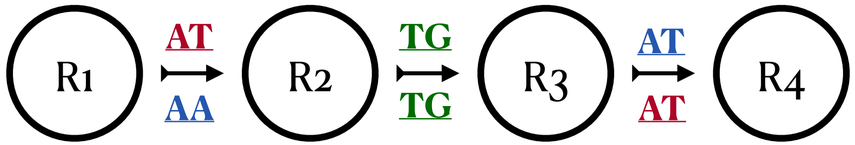
Transitive Reduction

String Graph Definition

Our goal is *retaining as much useful information as we can*, therefore we want to remove the redundant edges with *longer overhangs*

→ Overhang

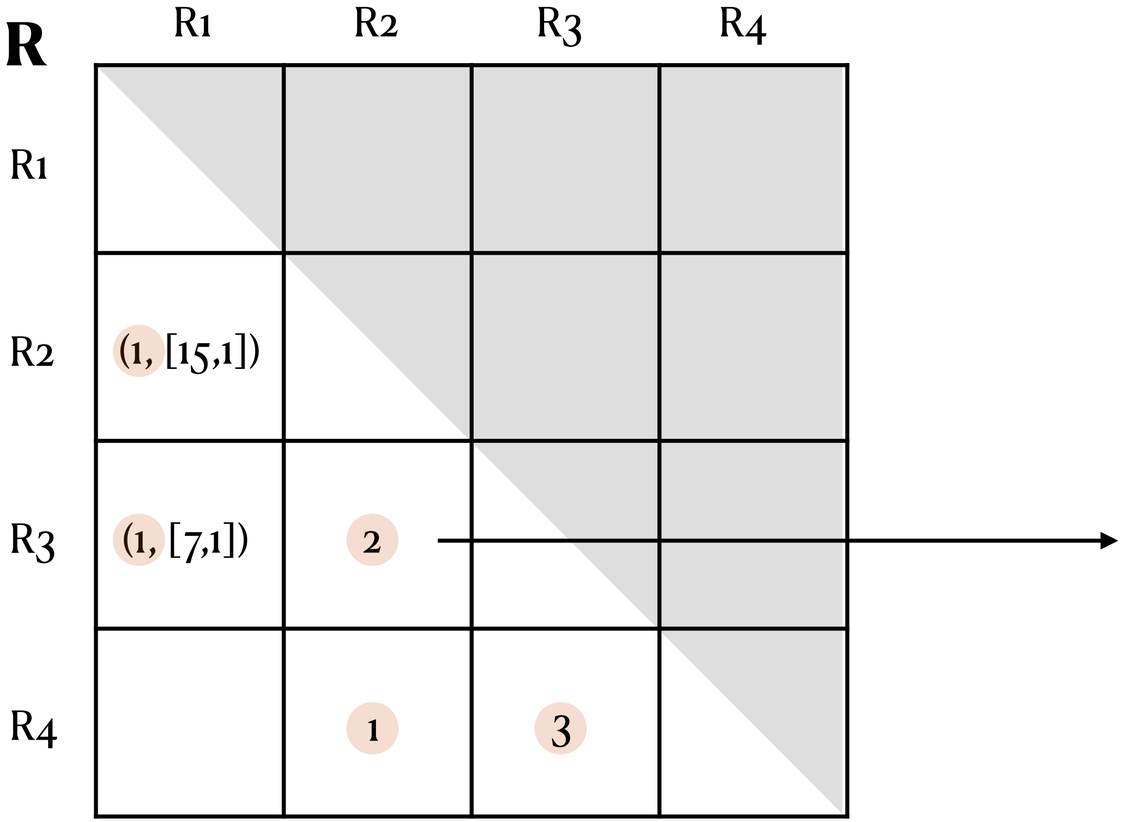
⟼ Bidirected edge

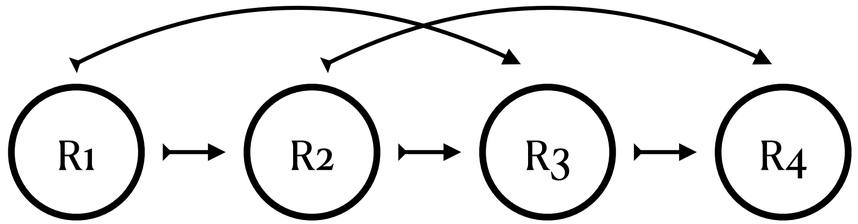


Transitive Reduction

An Overview of Our Algorithm [3]

From the current diBELLA's overlap matrix to a "string graph" matrix





In our *overlap* matrix, we **have**:

- Common k-mers
- K-mers positions in the reads

In our *string* matrix, we **want**:

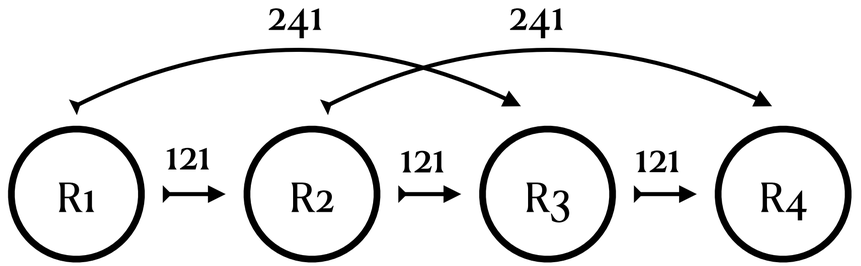
- Overhang length
- Directionality, that is type of edge

Transitive Reduction

An Overview of Our Algorithm

Our string matrix:

R	R1	R2	R3	R4
R1				
R2	121 (30, 01)			
R3	241 (60, 01)	121 (30, 01)		
R4		241 (60, 01)	121 (30, 01)	



In our *string* matrix, we **have**:

- Overhang length
- Directionality, that is type of edge (00, 01, 10, 11)

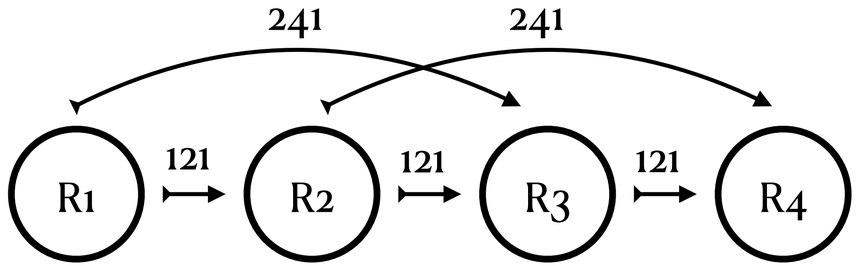
For **simplicity**, I'm going to assume we have only forward edges *for now*

Transitive Reduction

An Overview of Our Algorithm

1. R^2 with *custom semiring* to identify all the 2-hops neighbors

R	R1	R2	R3	R4
R1				
R2	121 (30, 01)			
R3	241 (60, 01)	121 (30, 01)		
R4		241 (60, 01)	121 (30, 01)	



In our *string* matrix, we **have**:

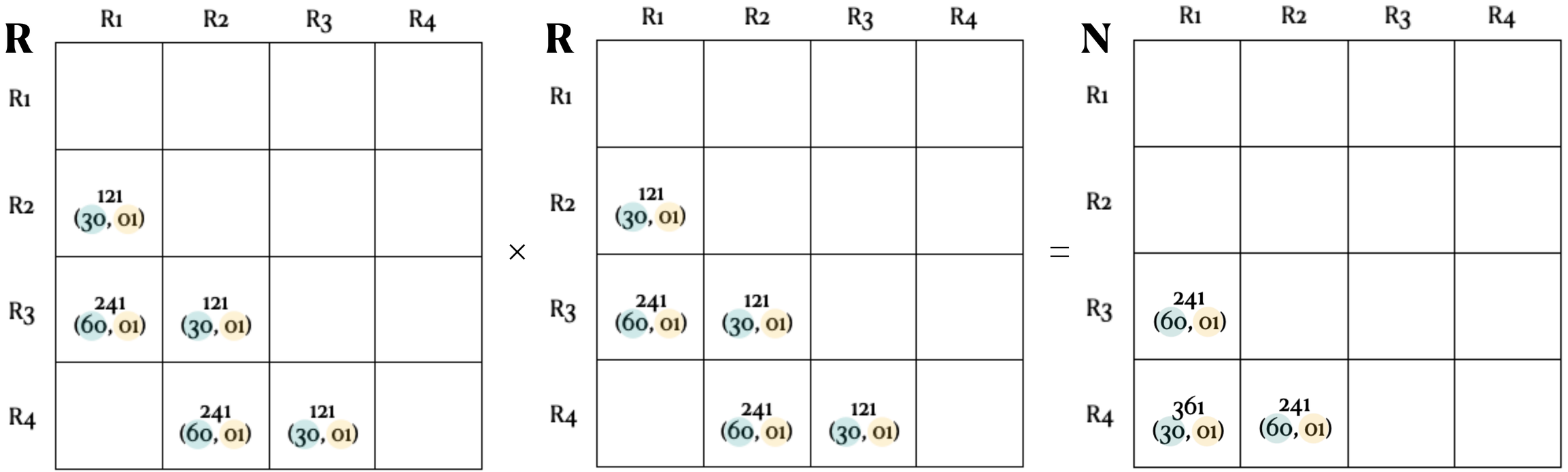
- Overhang length
- Directionality, that is type of edge (00, 01, 10, 11)

For this multiplication, our **custom multiply is a *add* operation** and **add is a *min* operation**

Transitive Reduction

An Overview of Our Algorithm (MPI + OpenMP)

1. R^2 with *custom semiring* to identify all the 2-hops neighbors to obtain matrix N

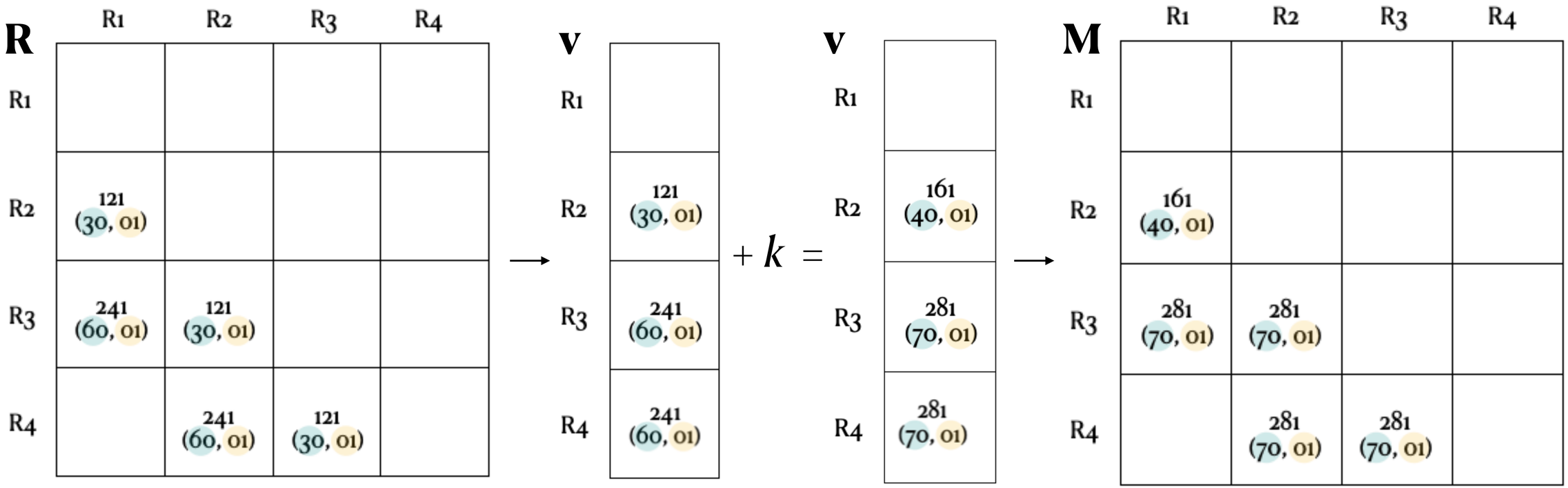


Using our custom semiring, we identify the *shortest* two-hops neighbors

Transitive Reduction

An Overview of Our Algorithm

2. Perform a reduction on R to add a constant *k* to the largest row value

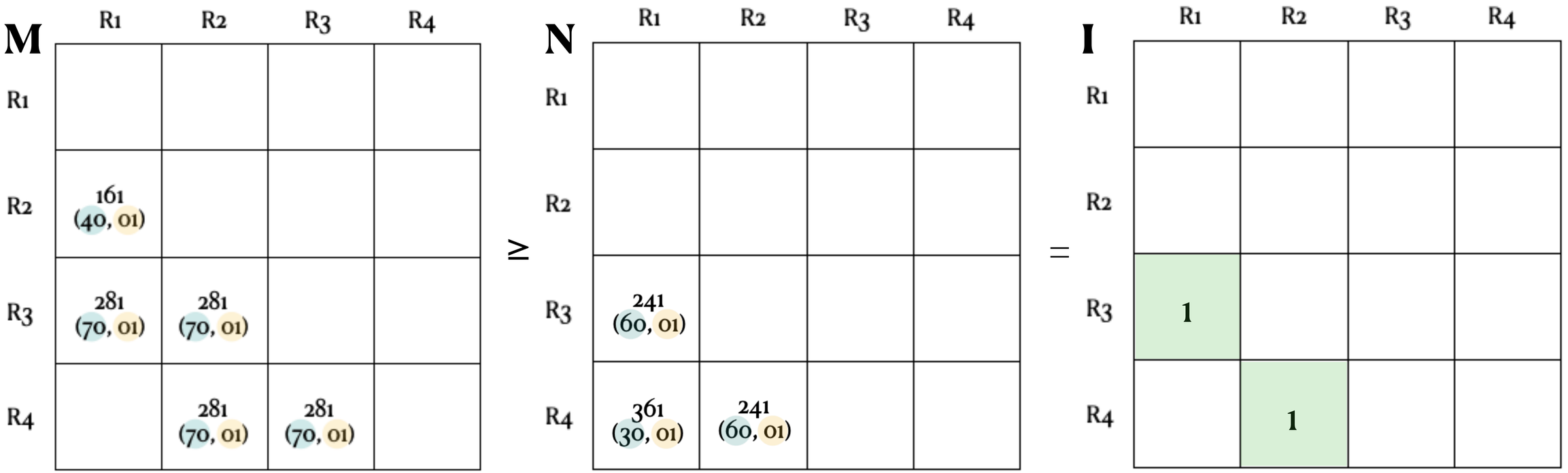


k accounts for possible errors in the reads

Transitive Reduction

An Overview of Our Algorithm

3. Find edges that can be eliminated performing element-wise I = M ≥ N

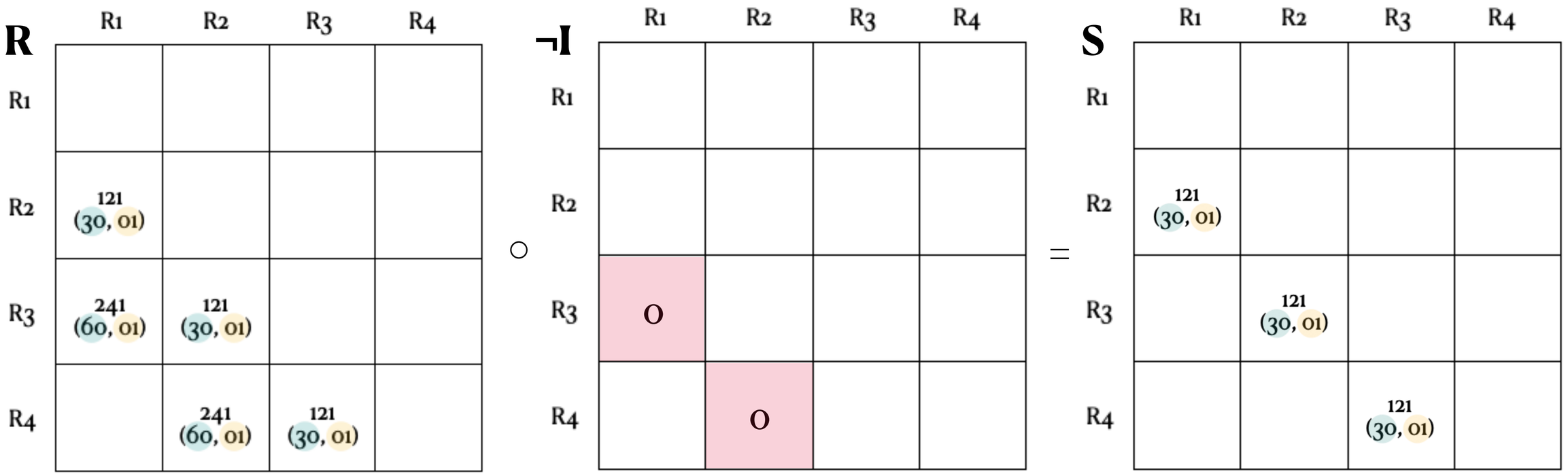


If zero values appear, we prune them

Transitive Reduction

An Overview of Our Algorithm

4. Element-wise S = R .* not(I) to remove transitive edges



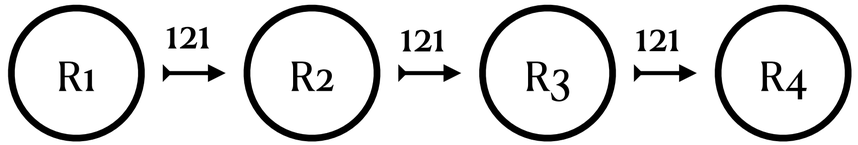
The computation iterates over the reduced matrix **R** until no more reduction is performed —$\mathbf{R}^2$ is the most compute intensive operation in the transitive reduction

Transitive Reduction

An Overview of Our Algorithm

Our transitive reduced matrix S looks like

	R1	R2	R3	R4
R1				
R2	121 (30, 01)			
R3		121 (30, 01)		
R4			121 (30, 01)	



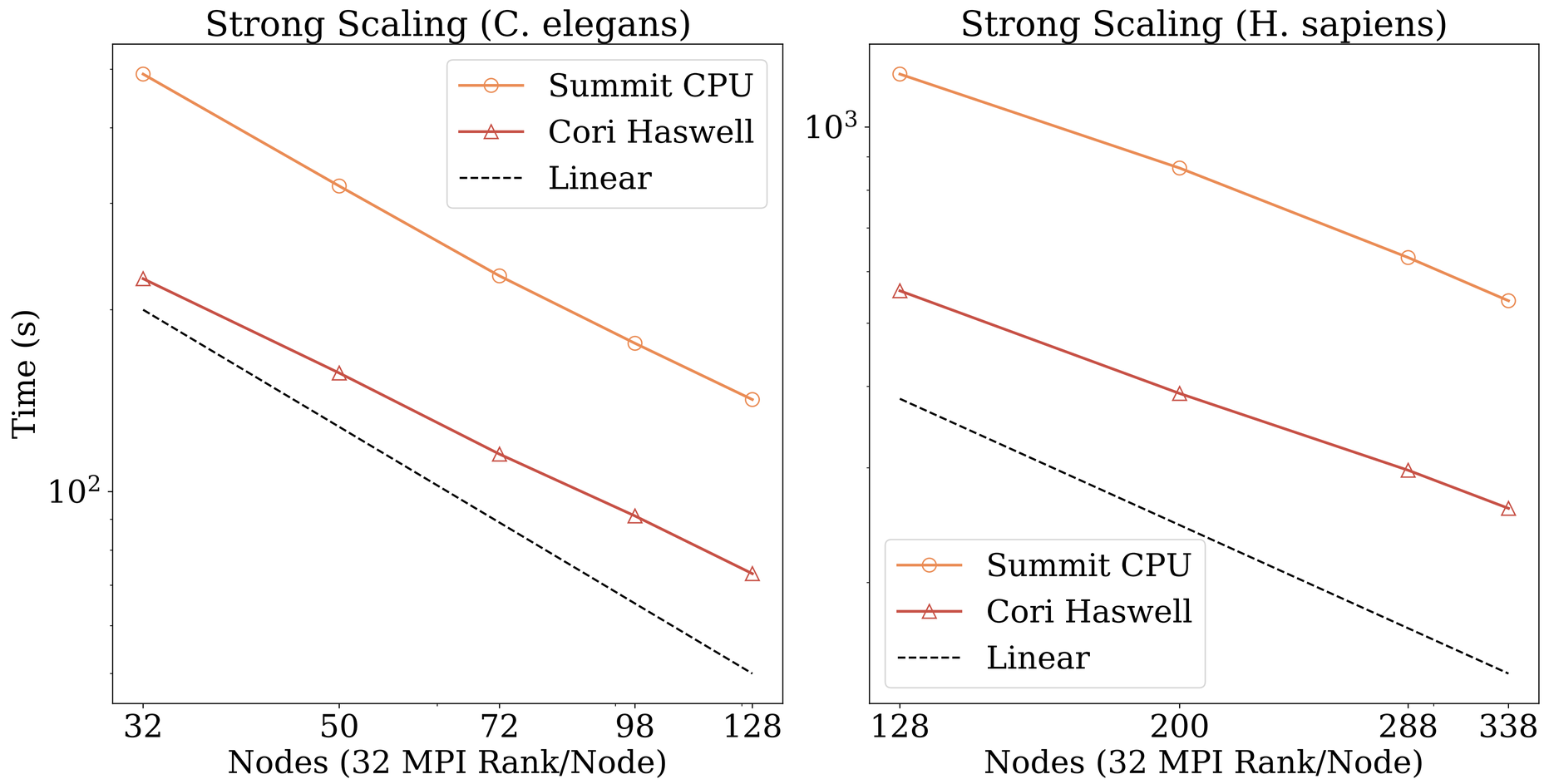
In our *string* matrix, we **have**:

- Overhang length
- Directionality, that is type of edge (00, 01, 10, 11)

For this multiplication, our **custom multiply is a *add* operation** and **add is a *min* operation**

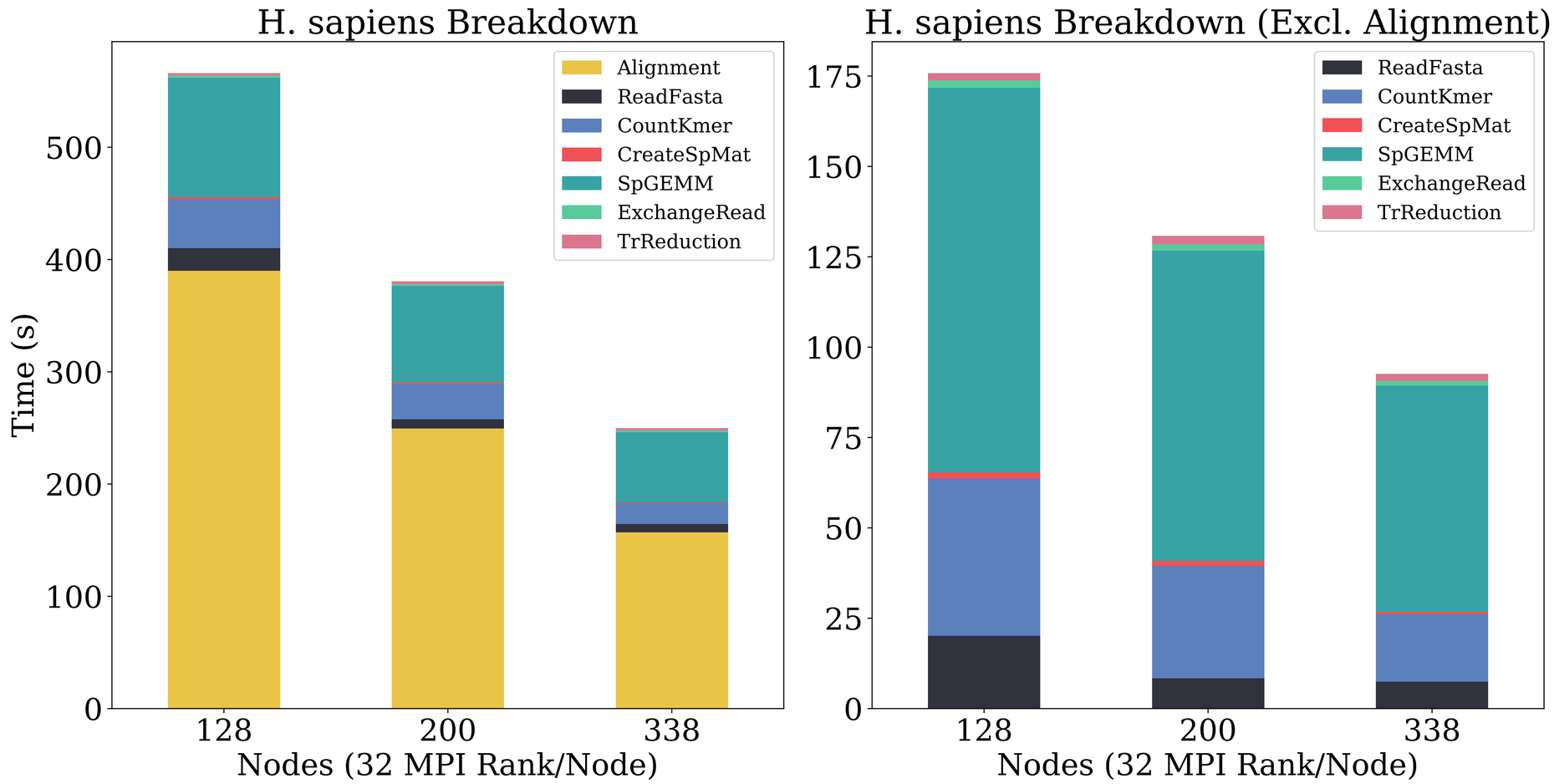
Results

Strong Scaling of diBELLA 2D



Results

Runtime Breakdown of diBELLA 2D on Cori Haswell (H. sapiens)



(A Few) Project Ideas

Hybrid MPI-OpenMP K-mer Counter: This project aims to improve **diBELLA 2D**'s current k-mer counter [2, 3], which uses only MPI and forces the entire pipeline to run with multiple MPI processes per node, even if other parts of the code have OpenMP support.

User-Defined Semiring GPU SpGEMM: The goal of this project is to implement a SpGEMM on GPU [10, 11] running on user-defined semiring operations, such as (min,+) for the shortest path computations in the transitive reduction of diBELLA 2D.

Hybrid CPU-GPU Pairwise Alignment: The idea is to introduce CPU work stealing in the **ADEPT** [8] and **LOGAN** [7] API call. This would mean decomposing the current codes, extracting bare kernels, wrapping them in calls that return the state of the kernel (completed or not), and then based on that state, executing work stealing using a CPU SW library.

If interested, contact: gguidi@berkeley.edu
Class website: https://sites.google.com/lbl.gov/cs267-spr2021

References

[1] Guidi, G., Ellis, M., Rokhsar, D., Yelick, K., & Buluç, A. (2020). BELLA: Berkeley efficient long-read to long-read aligner and overlapper. bioRxiv, 464420.

[2] Georganas, E., Buluç, A., Chapman, J., Hofmeyr, S., Aluru, C., Egan, R., ... & Yelick, K. (2015, November). Hipmer: an extreme-scale de novo genome assembler. In SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-11). IEEE.

[3] Guidi, G., Selvitopi, O., Ellis, M., Oliker, L., Yelick, K., & Buluc, A. (2020). Parallel String Graph Construction and Transitive Reduction for De Novo Genome Assembly. In proceedings of IPDPS21.

[4] Buluç, A., & Gilbert, J. R. (2012). Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. SIAM Journal on Scientific Computing, 34(4), C170-C191.

[5] Buluç, A., & Gilbert, J. R. (2011). The Combinatorial BLAS: Design, implementation, and applications. The International Journal of High Performance Computing Applications, 25(4), 496-509.

[6] Selvitopi, Oguz, Saliya Ekanayake, Giulia Guidi, Georgios Pavlopoulos, Ariful Azad, and Aydin Buluc. "Distributed many-to-many protein sequence alignment using sparse matrices." arXiv preprint arXiv:2009.14467 (2020).

[7] Zeni, A., Guidi, G., Ellis, M., Ding, N., Santambrogio, M. D., Hofmeyr, S., ... & Yelick, K. (2020, May). Logan: High-performance GPU-based x-drop long-read alignment. In 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 462-471). IEEE.

[8] Awan, M. G., Deslippe, J., Buluc, A., Selvitopi, O., Hofmeyr, S., Oliker, L., & Yelick, K. (2020). ADEPT: a domain independent sequence alignment strategy for gpu architectures. BMC bioinformatics, 21(1), 1-29.

[9] Döring, A., Weese, D., Rausch, T., & Reinert, K. (2008). SeqAn an efficient, generic C++ library for sequence analysis. BMC bioinformatics, 9(1), 1-9.

[10] https://github.com/GPUPeople/ACSpGEMM

[11] https://cusplibrary.github.io/