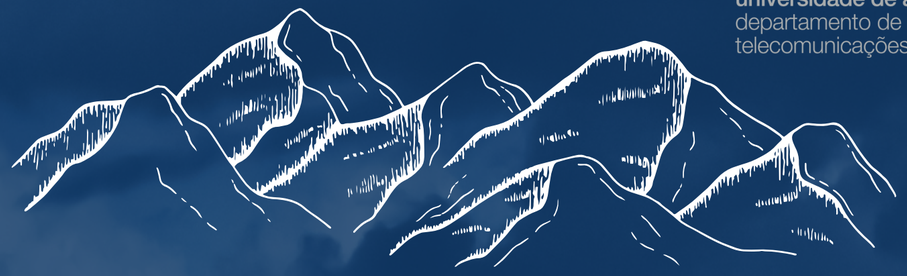
deti
universidade de aveiro
departamento de eletrónica,
telecomunicações e informática



WEATHER STATIONS

DISTRIBUTED-MEMORY PIPELINE WITH MPI (MESSAGE PASSING INTERFACE)

Carolina Reis | n° 131193
Eduardo Lopes | n° 103070

OBJECTIVES

1

Synchronization

Explain the MPI synchronization protocol.

2

Safety

Show why the pipeline cannot deadlock & how starvation is avoided.

3

Performance

Present benchmark evidence and demonstrate the implementation live.

ARCHITECTURE OVERVIEW

MAIN PRODUCER/CONSUMER PIPELINE

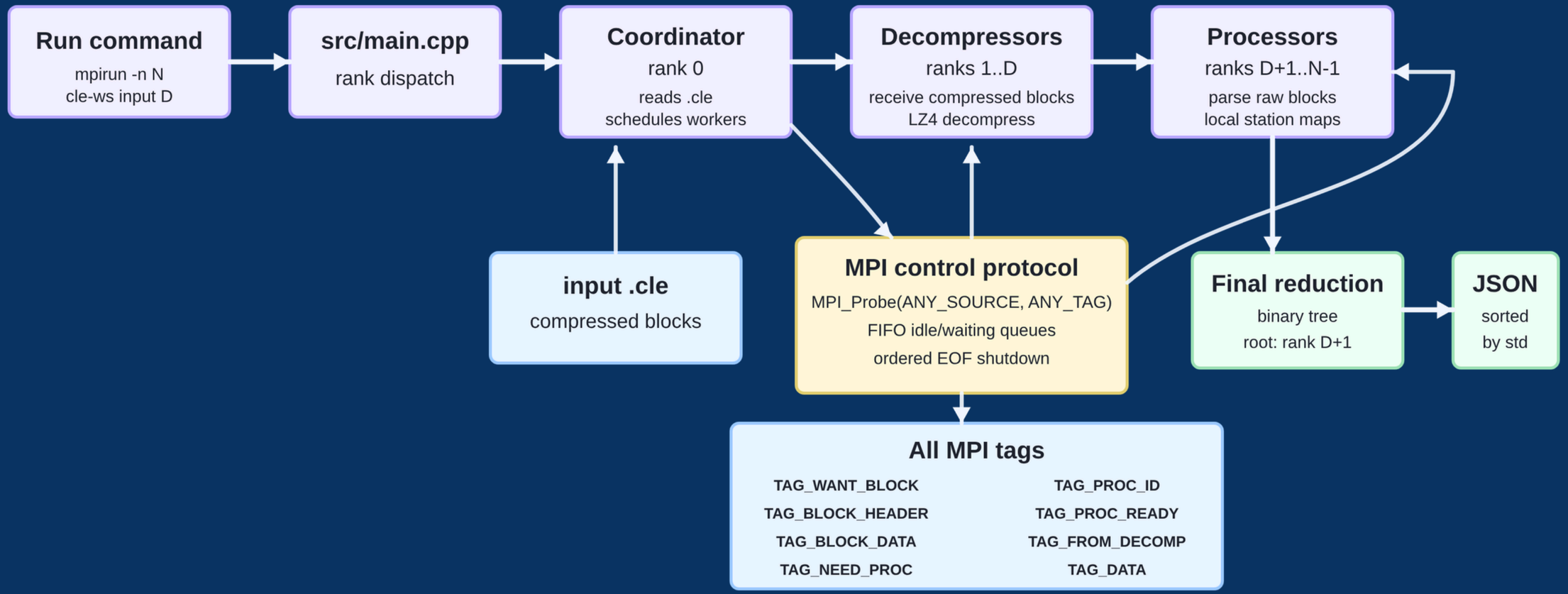
Rank(s)	Role	Responsibility
0	Coordinator	reads .cle, schedules work, sends EOF
1..D	Decompressors	receive compressed blocks, LZ4 decompress
D+1..N-1	Processors	parse raw data, compute stats, reduce

```
mpirun -n N ./build/cle-ws <file.cle> D

Rank 0          → Coordinator    (1 process)
Ranks 1..D      → Decompressors  (D processes)
Ranks D+1..N-1 → Processors     (N - D - 1 processes)
```

ARCHITECTURE OVERVIEW

MAIN PRODUCER/CONSUMER PIPELINE



SYNCHRONIZATION MECHANISMS

All coordination is done exclusively through MPI point-to-point messages!

No shared memory. No locks.

The Coordinator is event-driven:

MPI_Probe(ANY_SOURCE, ANY_TAG) → blocks until any message arrives

Two FIFO queues manage unmatched workers:

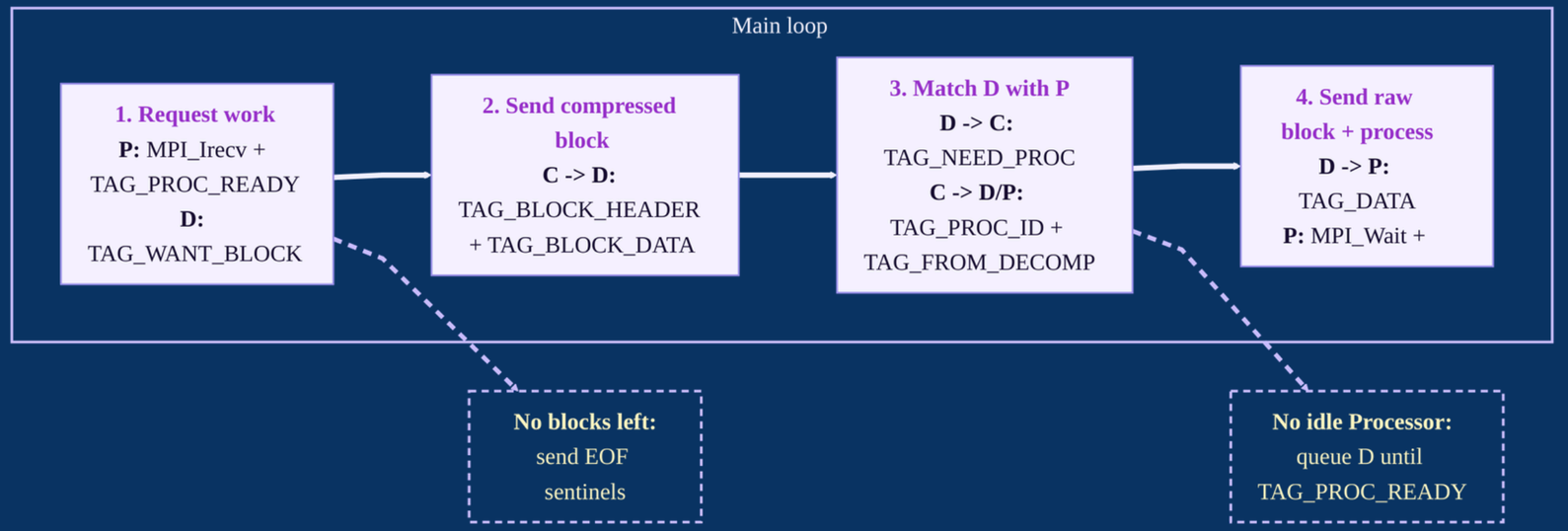
idle_procs → Processors waiting for a block

waiting_decomps → Decompressors waiting for a Processor

SYNCHRONIZATION MECHANISMS

Implementation anchors:

```
src/protocol.h
src/coordinator.cpp   → run(), on_want_block(), on_need_proc(), on_proc_ready()
src/processor.cpp     → run()
```



SYNCHRONIZATION MECHANISMS

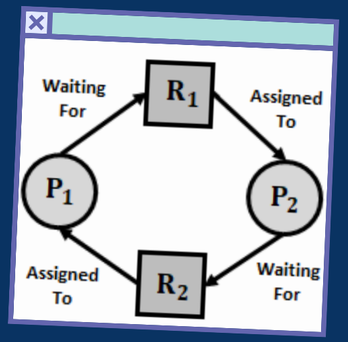
Protocol table:

Tag	Direction	Meaning
TAG_WANT_BLOCK	Decompressor → Coordinator	request compressed block
TAG_BLOCK_HEADER	Coordinator → Decompressor	{compressed_size, block_size} or EOF
TAG_BLOCK_DATA	Coordinator → Decompressor	compressed bytes
TAG_NEED_PROC	Decompressor → Coordinator	raw block ready, need a Processor
TAG_PROC_ID	Coordinator → Decompressor	assigned Processor rank
TAG_PROC_READY	Processor → Coordinator	Processor is idle
TAG_FROM_DECOMP	Coordinator → Processor	source Decompressor rank or EOF
TAG_DATA	Decompressor → Processor	decompressed raw block

DEADLOCK PREVENTION

INVARIANT: NO RAW BLOCK IS EVER SENT UNTIL BOTH ENDPOINTS ARE MATCHED.

1. Coordinator never blocks on a specific rank.
MPI_Probe(ANY_SOURCE, ANY_TAG) → reacts to any message.

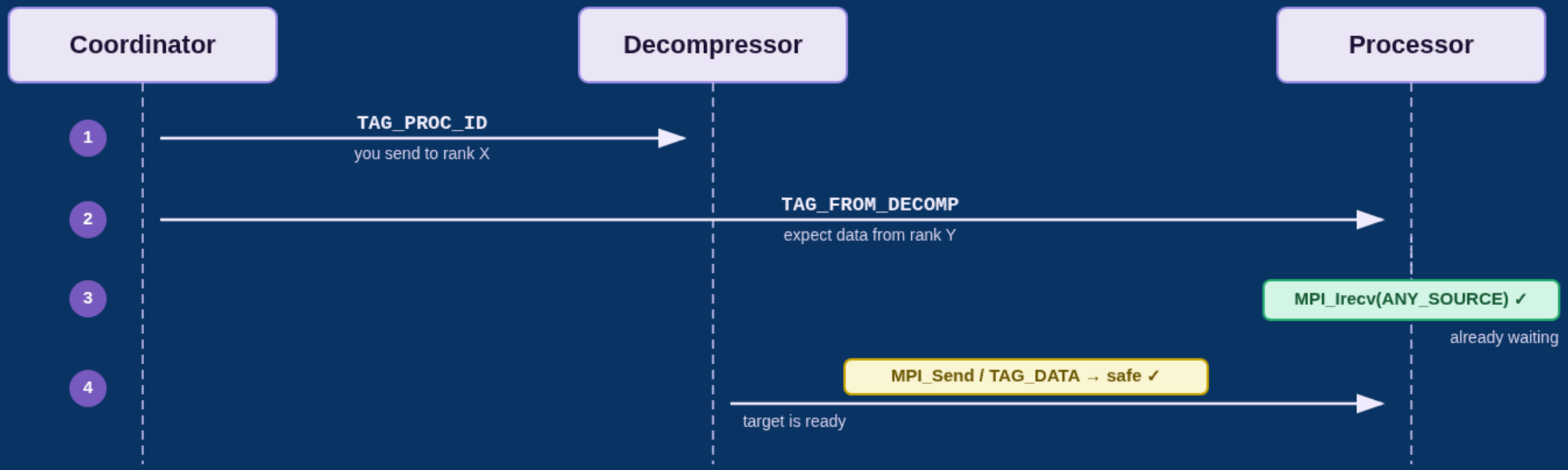


2. Decompressors do not choose a Processor.
They wait for TAG_PROC_ID from the Coordinator.

3. Processors post MPI_Irecv(ANY_SOURCE, TAG_DATA) before signalling ready.
They are already waiting when the Decompressor sends.

4. EOF is sent only after the pipeline is drained.
Processors stop only when active_decomps reaches zero.

DEADLOCK PREVENTION

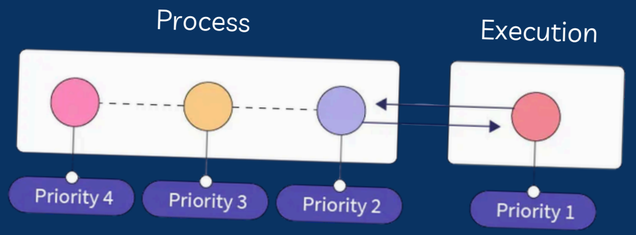



```
Coordinator::on_need_proc()    → matchmaking + parking
Coordinator::on_proc_ready()   → symmetric side
Processor::run()               → MPI_Irecv posted before TAG_PROC_READY
```

STARVATION PREVENTION AND SHUTDOWN

PROGRESS TABLE:

Situation	Coordinator action	Why progress continues
Processor ready, Decompressor waiting	pop **waiting_decomps**	oldest waiting Decompressor moves
Decompressor ready, Processor idle	pop **idle_procs**	oldest idle Processor moves
Processor ready, no work yet	push **idle_procs**	Processor matched later
Last Decompressor done	flush **idle_procs** with EOF	blocked Processors are released



FIFO queues → workers that waited longest are served first.
Every worker is either matched or stored, never dropped.
EOF is sent exactly once per worker, only when safe.

```
Coordinator::on_need_proc()    → matchmaking + parking
Coordinator::on_proc_ready()   → symmetric side
Processor::run()               → MPI_Irecv posted before TAG_PROC_READY
```

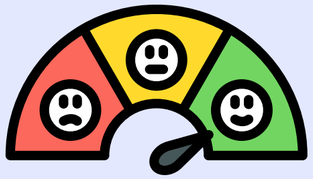
PERFORMANCE SNAPSHOT

Wall time (ms), average of 3 runs

Blocks	Single-thread	D=1, N=3	D=2, N=4	D=2, N=6	D=3, N=7	D=3, N=8	Speedup
10	13253 ms	16161 ms	15337 ms	9870 ms	9134 ms	8713 ms	1.52x
25	33916 ms	39546 ms	38182 ms	20105 ms	19340 ms	18052 ms	1.88x
50	70034 ms	79459 ms	79238 ms	39083 ms	35657 ms	34006 ms	2.06x
100	143108 ms	158928 ms	159813 ms	79668 ms	68338 ms	63223 ms	2.26x

Key numbers:

- Decompressed block size: ~512 MiB
- Coordinator send slots: D + 2
- Aggregation rounds: $\log_2$(num_processors)
- Best configuration: D=3, N=8 (4 processors)
- Bottleneck: Text parsing in Processors



CONCLUSION

MAIN MESSAGE

- The Coordinator serialises only control decisions, not the data path.

Implementation

- Tag-based MPI protocol
- Direct Decompressor → Processor transfer
- D+2 non-blocking send slots
- Parallel Welford merge (numerically stable)
- O(log P) aggregation rounds

Correctness & Liveness

- No deadlock by design
- FIFO matching avoids starvation
- Ordered EOF drains the pipeline
- Only Processors reduce
- Up to 2.26× speedup

DEMO

Live command sequence

```
cd distributed_memory
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build -j$(nproc)

./build/cle-samples 10
./run_timed.sh measurements-10.cle 1 3     # 1 Decompressor, 2 Processors (baseline)
./run_timed.sh measurements-10.cle 3 8     # 3 Decompressors, 4 Processors (best)
```

Self-tests (correctness)

```
./build/cle-stats-selftest
mpirun -n 4 ./build/cle-aggregation-selftest
```

Expected results:

- First run (D=1, N=3) is slower than expected → parsing bottleneck visible.
- Second run (D=3, N=8) is the best config → observe elapsed time on stderr.
- Self-tests validate: Welford update, parallel merge, serialisation, reduction.
- The program always terminates cleanly → EOF propagation works.
- Only rank D+1 prints the final JSON → binary tree converges to root.


THANK
YOU!