Torculas, Richard O. **02/14/2026**
J3A **Prof. Grachel Ching**

Assignment 2: Simple Distributed Message-Passing Coding Exercise

Why is message passing required in distributed systems?

- In a distributed environment, processes are fundamentally isolated; each operates within its own private memory address space. Because one process cannot "reach into" the RAM of another to read or write data, an explicit mechanism is required to bridge this gap.

What happens if one process fails?

- Without built-in fault tolerance, a basic MPI program will hang indefinitely if a worker process fails before sending its result. This reveals a critical vulnerability where the master remains in a blocking wait, demonstrating that a single point of failure can paralyze the entire distributed computation.

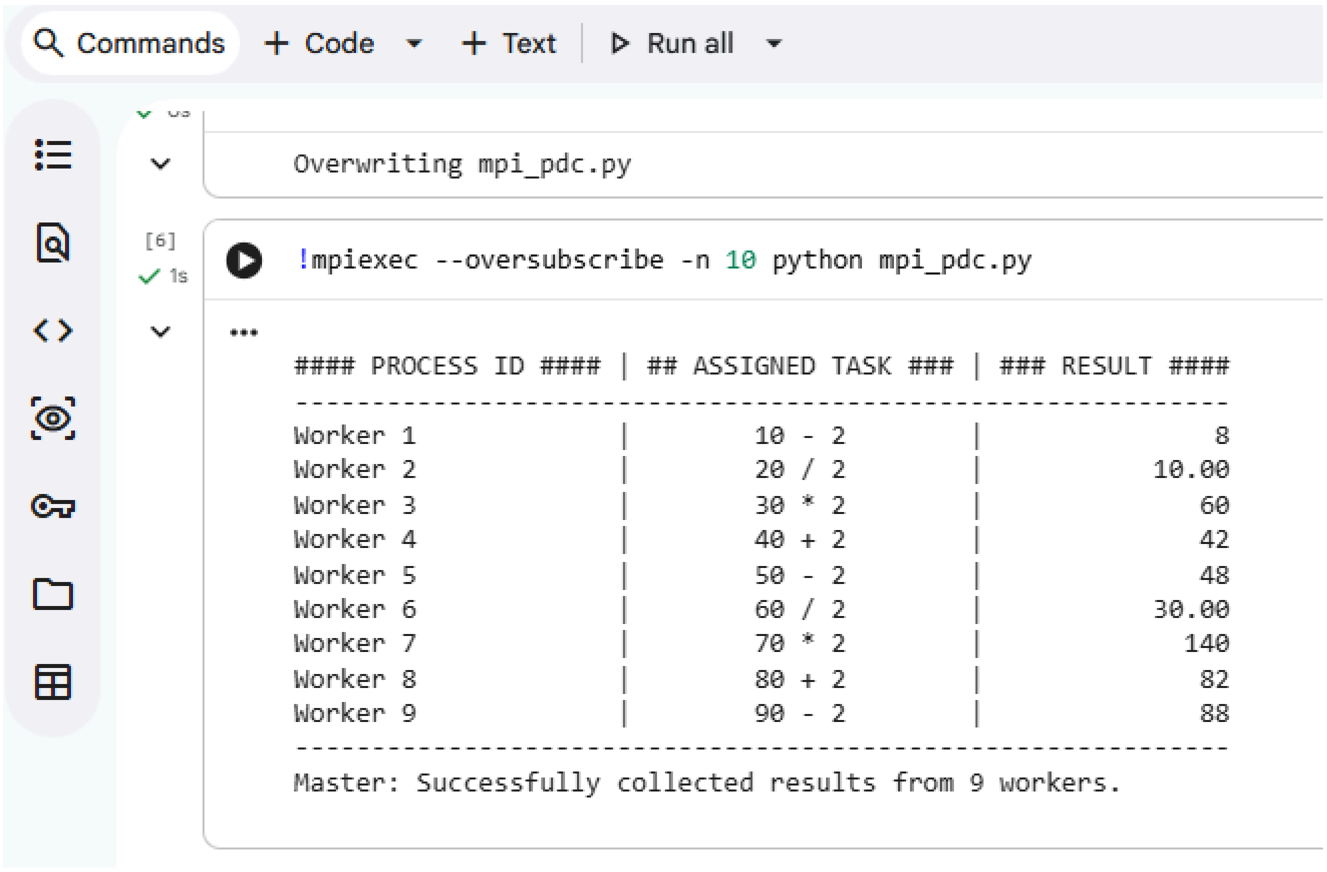
How does this model differ from shared-memory programming?

- Message passing requires explicit communication between private memory spaces, which increases coding complexity but naturally prevents data races and allows for massive scaling across clusters. In contrast, shared-memory programming allows threads to access a common memory pool directly, which is simpler for local tasks but requires strict synchronization to avoid data conflicts.

SUMMARY

- In distributed systems, message passing is essential because processes operate in isolated memory spaces and must explicitly exchange data to coordinate tasks. This model is highly scalable and prevents data races, but it lacks inherent fault tolerance, meaning a single process failure can cause the entire system to hang. Ultimately, while it is more complex to implement than shared-memory programming, message passing provides a safer and more robust framework for computing across multiple machines.

OUTPUT

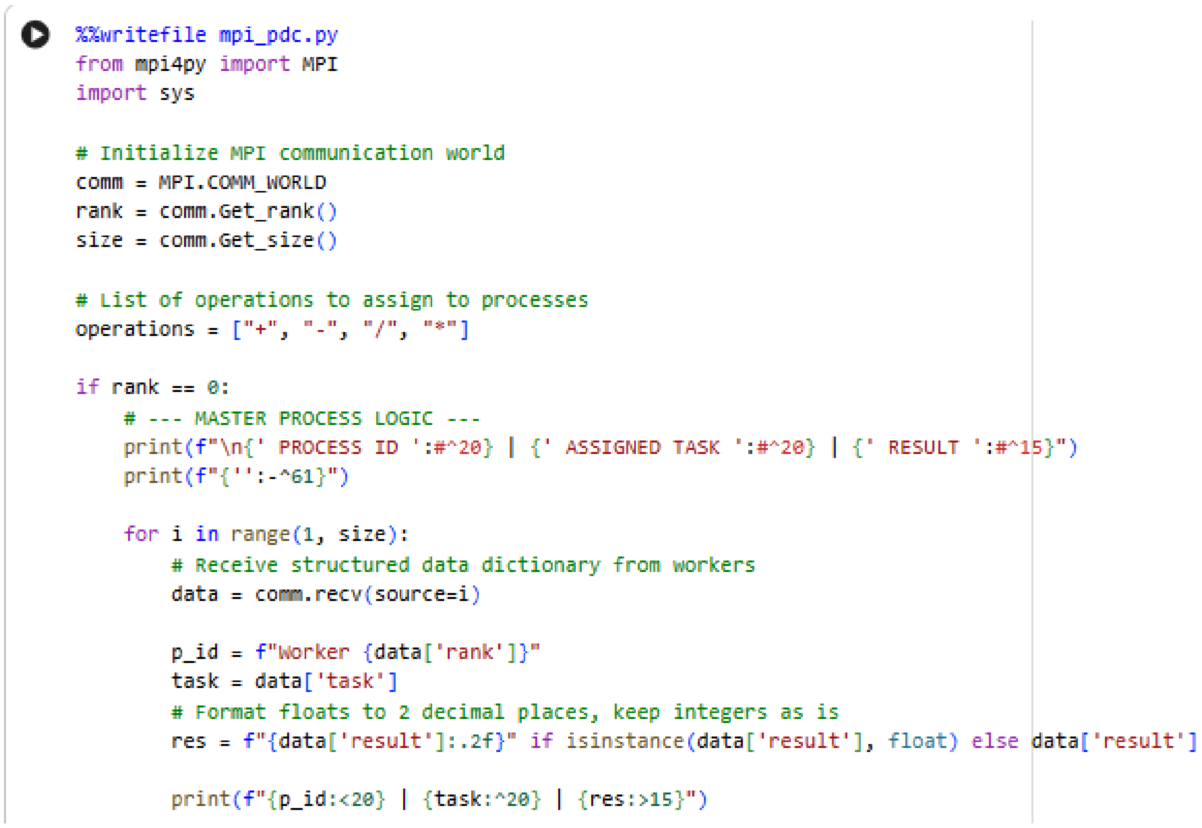


```
Overwriting mpi_pdc.py
```

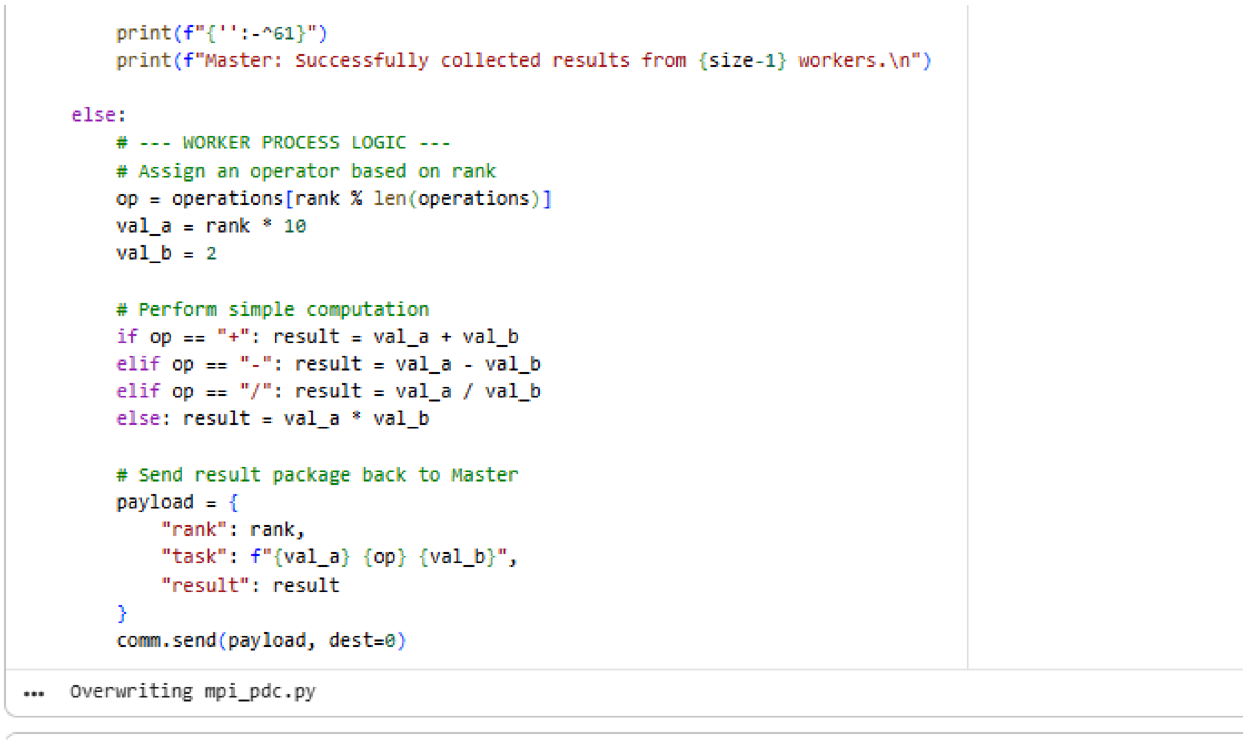
```
!mpiexec --oversubscribe -n 10 python mpi_pdc.py
```

```
#### PROCESS ID #### | ## ASSIGNED TASK ### | ### RESULT ####
-------------------------------------------------------------
Worker 1             |        10 - 2        |              8
Worker 2             |        20 / 2        |          10.00
Worker 3             |        30 * 2        |             60
Worker 4             |        40 + 2        |             42
Worker 5             |        50 - 2        |             48
Worker 6             |        60 / 2        |          30.00
Worker 7             |        70 * 2        |            140
Worker 8             |        80 + 2        |             82
Worker 9             |        90 - 2        |             88
-------------------------------------------------------------
Master: Successfully collected results from 9 workers.
```

CODE



```python
%%writefile mpi_pdc.py
from mpi4py import MPI
import sys

# Initialize MPI communication world
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# List of operations to assign to processes
operations = ["+", "-", "/", "*"]

if rank == 0:
    # --- MASTER PROCESS LOGIC ---
    print(f"\n{' PROCESS ID ':#^20} | {' ASSIGNED TASK ':#^20} | {' RESULT ':#^15}")
    print(f"{'':-^61}")

    for i in range(1, size):
        # Receive structured data dictionary from workers
        data = comm.recv(source=i)

        p_id = f"Worker {data['rank']}"
        task = data['task']
        # Format floats to 2 decimal places, keep integers as is
        res = f"{data['result']:.2f}" if isinstance(data['result'], float) else data['result']

        print(f"{p_id:<20} | {task:^20} | {res:>15}")
```



```python
    print(f"{'':-^61}")
    print(f"Master: Successfully collected results from {size-1} workers.\n")

else:
    # --- WORKER PROCESS LOGIC ---
    # Assign an operator based on rank
    op = operations[rank % len(operations)]
    val_a = rank * 10
    val_b = 2

    # Perform simple computation
    if op == "+": result = val_a + val_b
    elif op == "-": result = val_a - val_b
    elif op == "/": result = val_a / val_b
    else: result = val_a * val_b

    # Send result package back to Master
    payload = {
        "rank": rank,
        "task": f"{val_a} {op} {val_b}",
        "result": result
    }
    comm.send(payload, dest=0)
```

```
... Overwriting mpi_pdc.py
```