# COMP90020 Distributed Algorithms

## **Project Demo**: Dynamic Vector Clocks

Team Double-J
**James Sammut** (502030)
**Joel Kenna** (995401)

# Introduction and Background

- The concept of “Logical Time”: mapping the physical phenomenon when utilised with distributed systems

- Logical clocks - the ordering of events within processes part of a distributed system. **Lamport** & **Vector Clocks** [3]-[5] are famous and well-regarded in this context

- Various extensive clock-based algorithms exist:, **Dynamic Vector Clocks** [1], **Interval Tree Clocks** [6], **Prime Clocks** [7], **Chain Clocks** [8], **Matrix Clocks** [1] [9] [10]

- We’ve chosen to implement Dynamic Vector Clocks for this project!

Clocks, clocks and clocks!

# Dynamic Vector Clocks

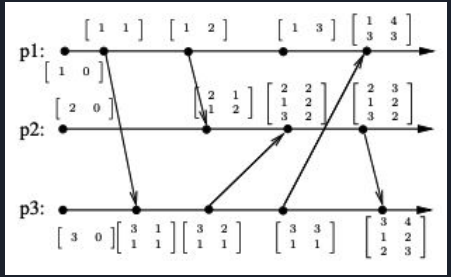
- Vector Clocks: an array (V) of scalar values.
  - V[i] Incremented by $P_i$ upon message send/internal event
  - Received by process $P_{j,}$ who increments V[j]

- **The number of processes (*n*) is pre-determined: not adjustable!**

- Dynamic Vector Clocks (DVCs) are a way of making this requirement unnecessary: processes (peers) join a session and combine their vector clocks on message delivery.

- Peers ($P_i$) hold onto a clock with [$P_i$, <int>] for all $P_j$ : $P_i \neq P_j$ for each other process $P_j$ it knows of after message receive/delivery.

- The DVC algorithms allows flexibility and is the basis of our projects implementation



Dynamic Vector Clock Process Diagram [2]

# Dynamic Vector Clocks - The Rationale

**Causal Broadcast Ordering**

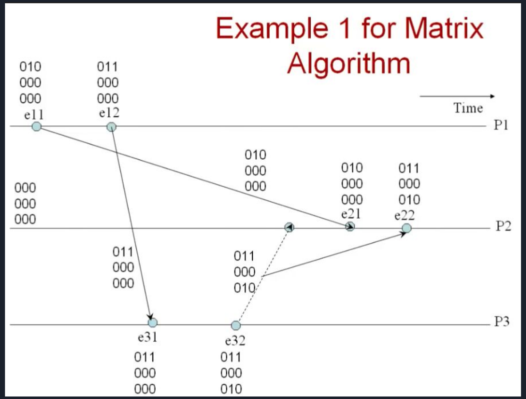
- Non-causal ordering of messages = **confusion**.
- A process lags/network becomes segregated **- unordered.**

**Dynamic Networks and Peer Size**

- For modern P2P Messaging Applications - peer size not fixed.
- Peers **should be able to join/leave whenever they like.**
- A DVC can grow/shrink as needed.

**Simpler Implementation**

- Peers only need to hold onto a clock with [Process_ID, clock]
- Matrix Clocks - **initially considered.**
    - **Huge overhead** with storing sends for each send to $P_j$ from $P_i$
    - Grows exponentially (i.e 20 peers, 400 values)
    - Not applicable for our use case of **broadcast-only**
    - Delivery logic is similar to DVCs.




Matrix Clock and Broadcast between Processes [1]

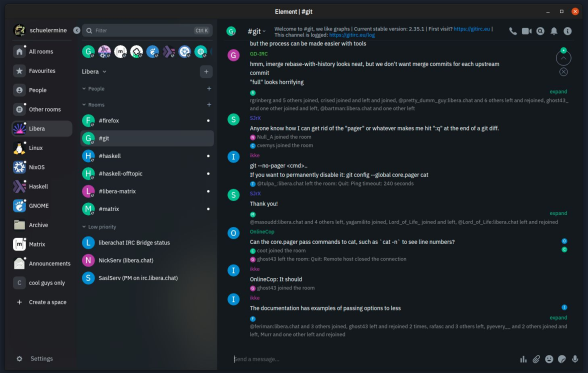
# Our Application - P2P Messaging Application

## Application Features

- P2P messaging application for scenarios with low room for confusion
- Peers are able to broadcast messages to one-another
- **Guaranteed** causal message delivery
- Peers can **enter** and **leave** the network dynamically
- Simple GUI (Kivy library) for improved user experience

## Implemented in 2 Phases

- Phase 1: Algorithm development using MPI
    - Dynamic Vector Clock & Matrix Clock algorithm implementations
- Phase 2: Algorithm socket integration and GUI
    - Utilising algorithm implemented from Phase #1
    - Allowance for peers to enter/leave as needed
    - With a peer registry server, or without




Chat Room - just an example!

# **Phase 1**: Algorithm development using MPI

- Implemented using Python & MPI (Message Passing Interface)
- Dynamic Vector Clock and Matrix Clock algorithms
- Simulated environment of *n* processes (whom are 'aware' of each other)
- Both broadcast and unicast messages.
- Input of events on a process (text file) -> output on terminal **with causal delivery checking.**
- Enqueued if deliverability is not adhered.

```
b1, r3, r2
r1, b2, b3
r2, r3, r1
```

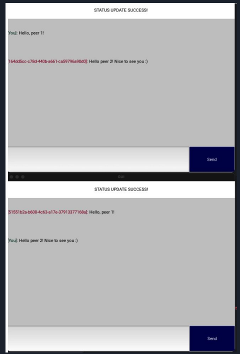
Process events - invalid broadcast example

```
└ ./phase1_invoke.sh -f examples/dynamic_vector_clocks/b1_3_node_simple_valid.txt -a dvc
|----- Process 1: ['b1'] -----|
----------------------
Event #0 -> b1: (0)
----------------------
Broadcast message to process(es) [2, 3]
Process 1 generated message heading for Process(es) [2, 3]. Number: 9.936. DVC of [[1, 1], [2, 0], [3, 0]]
Process 1 broadcasting message to Process(es) [2, 3] @ 22:02:44.117457

|----- Process 2: ['r1'] -----|
----------------------
Event #0 -> r1: (0)
----------------------
Process 2 received number 9.936 from Process 1 @ 22:02:44.117602
This message satisfied the DVC causal deliverability condition. Delivering.
9.936 (message number) + 0 (current sum). Process 2's sum = 9.936

Checking messages in the message/hold back queue for deliverability
No messages are in the message/hold back queue
DVC after r1:   [[1, 1], [2, 0], [3, 0]]
Number Sum:      9.936

|----- Process 3: ['r1'] -----|
----------------------
Event #0 -> r1: (0)
----------------------
Process 3 received number 9.936 from Process 1 @ 22:02:44.117617
This message satisfied the DVC causal deliverability condition. Delivering.
9.936 (message number) + 0 (current sum). Process 3's sum = 9.936

Checking messages in the message/hold back queue for deliverability
No messages are in the message/hold back queue
DVC after r1:   [[1, 1], [2, 0], [3, 0]]
Number Sum:      9.936
```

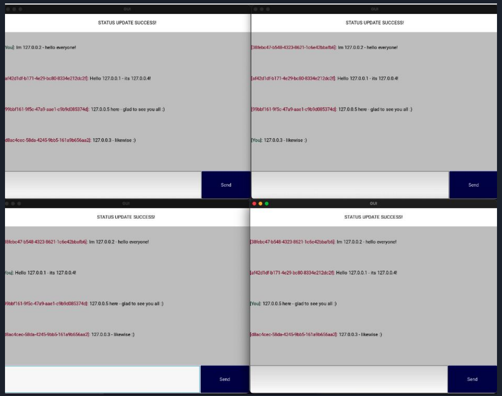
Phase 1 invocation - valid broadcast example

# **Phase 2**: Algorithm socket integration & GUI

- Implemented using Python & sockets
- Dynamic Vector Clock algorithm implemented from Phase 1 integrated
- Processes ("peers") able to dynamically join/leave the system
- All processes able to communicate with one-another: a "chat room"
- Implementation with a **peer registry server** (for handling registration), or not (specify IPs for peers)
- Messages that are broadcast: checked for **causal delivery.** Enqueued if deliverability is not adhered.




Phase 2 invocation - without peer registry server




Phase 2 invocation - with peer registry  server

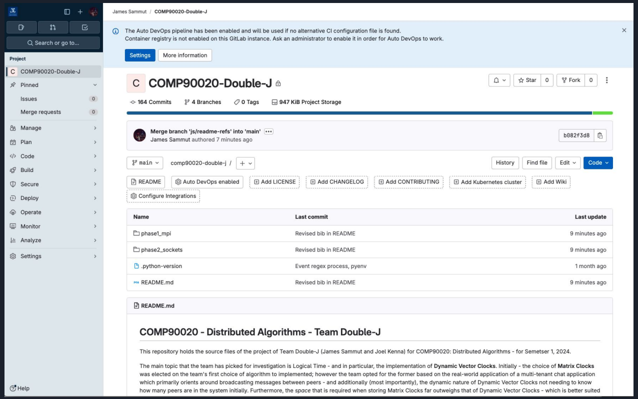
# COMP90020-Double-J Repository

- This project's implementation of both phases has been implemented within a [repository]() on the Melbourne School of Engineerings' GitLab instance.

- `phase1_mpi/` and `phase2_sockets/` directories - implementations for each phase separated

- The `README.md` within the repository contains detailed information for both phases:
  - Approach and background
  - Implementation specifics
  - Invocation

- **Our code submission includes all repository files!**




COMP90020-Double-J Repository

# 🎬 Demo Time 🎬

Thank you!

# References


[1] T. Landes. (2006). Dynamic Vector Clocks for Consistent Ordering of Events in Dynamic Distributed Applications. Presented at Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications & Conference on Real-Time Computing Systems and Applications [Online]. Available: https://vs.inf.ethz.ch/edu/HS2017/VS/exercises/A3/DVC_Landes.pdf

[2] N. Meghanathan. Module 6.2.3 Matrix Algorithm Causal Delivery of Messages. (Nov. 12, 2013). Accessed: Mar. 13, 2024. [Online video]. Available: https://www.youtube.com/watch?v=WgTx7BHWzts.

[3] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” in *Communications of the ACM,* vol.21, no. 7, Jul. 1, 1978, pp. 558-565, doi: 10.1145/359545.359563.

[4] P. Redmond, G. Shen, N. Vazou and L. Kuper, “Verified Causal Broadcast with Liquid Haskell,” in *IFL '22: Proceedings of the 34th Symposium on Implementation and Application of Functional Languages,* Copenhagen, Denmark, 2022, pp. 1-13, doi: 10.1145/3587216.3587222.

[5] M. Raynal and M. Singhal, "Logical time: capturing causality in distributed systems," in *Computer*, Feb. 1996, vol. 29, no. 2, pp. 49-56,, doi: 10.1109/2.485846.

[6] P. Sérgio Almeida, C. Baquero and V Fonte, “Interval Tree Clocks,” in *OPODIS '08: Proceedings of the 12th International Conference on Principles of Distributed Systems,* Luxor, Egypt, Dec. 2008, pp. 259-274. doi: 10.1007/978-3-540-92221-6_18.

[7] A. D.  Kshemkalyani, M Shen and B. Voleti, “Prime clock: Encoded vector clock to characterize causality in distributed systems” in *Journal of Parallel and Distributed Computing,* vol. 140, pp. 37-51, doi: 10.1016/j.jpdc.2020.02.008.

[8] A. Agarwal and V. K. Garg, “Efficient Dependency Tracking for Relevant Events in Shared-Memory Systems” in PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing, New York, NY, USA, Jul 17, 2005, pp. 19-28, doi: 10.1145/1073814.1073818.

[9] A. Singh and N. Badal, "An efficient implementation of multi matrix clocks in the distributed system," in *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, Greater Noida, India, Oct. 8-10, 2015, pp. 209-213, doi: 10.1109/ICGCIoT.2015.7380459.

[10]] H. Guerreiro, L. Rodrigues, N. Preguiça and N. Quental, "Causality Tracking Trade-offs for Distributed Storage," in *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*, Cambridge, MA, USA, Jun. 24-27, 2020, pp. 1-10, doi: 10.1109/NCA51143.2020.9306734.