Các hệ thống phân tán – Nhóm 14

# Truyền thông nhóm tin cậy
# Cam kết phân tán
# Phục hồi


| | |
|---|---|
| Nguyễn Thanh Hùng | B24CHHT076 |
| Quản Trường Huy | B24CHHT078 |
| Tô Thanh Thái | B24CHHT092 |

# Truyền thông nhóm tin cậy

## Reliable group communication

# Truyền thông nhóm tin cậy

- Đảm bảo 1 thông điệp được thực hiện:

Được gửi đến tất cả thành viên trong nhóm

Đảm bảo không lỗi, gửi chính xác và đúng thứ tự

# Truyền thông đang tin cậy – Ví dụ

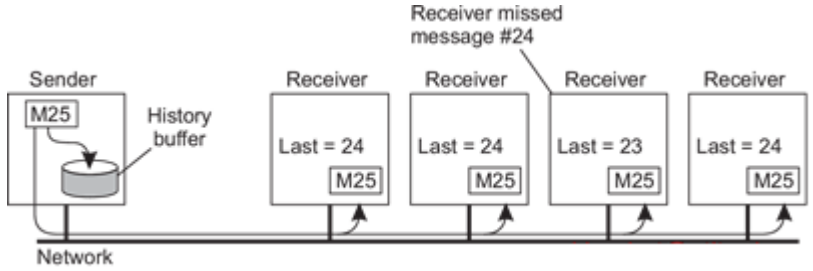
- Trong các ứng dụng như Zoom, Whatsapp group,… khi một người gửi tin nhắn thì mọi thành viên đều nhận được tin đó đúng nội dung.
- Trong các game online như DOTA, Fornite,… các sự kiện và trạng thái trong trò chơi cần được đồng bộ hóa giữa tất cả người chơi để tạo ra một trải nghiệm chơi game mượt mà và công bằng.

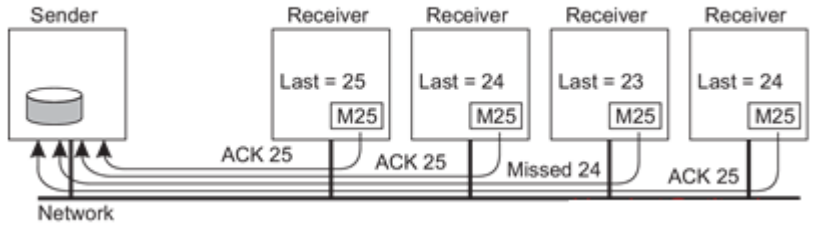
# Truyền thông nhóm tin cậy – 2 loại dựa trên quy mô hệ thống

Truyền thông nhóm tin cậy cơ bản

Truyền thông nhóm tin cậy trong hệ thống lớn

# Truyền thông nhóm tin cậy cơ bản




**Ưu điểm**

- Áp dụng cho hệ thống nhỏ
- Không đòi hỏi các giao thức đồng thuận phức tạp
- Dễ dàng giao tiếp giữa các nút

**Nhược điểm**

- Thiếu khả năng mở rộng
- Thiếu khả năng phục hồi sau lỗi
- Không đảm bảo tính nhất quán giữa các nút

# Truyền thông nhóm tin cậy trong hệ thống lớn

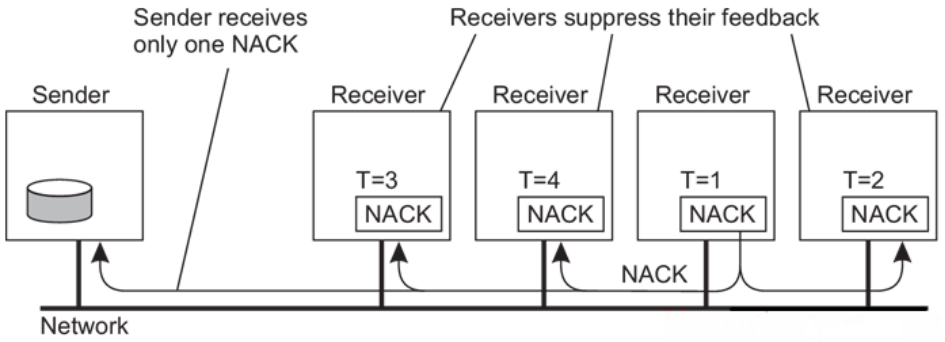
**Ưu điểm**

- Khả năng mở rộng
- Khả năng phục hồi và chịu lỗi
- Tính nhất quán và đồng bộ

**Nhược điểm**

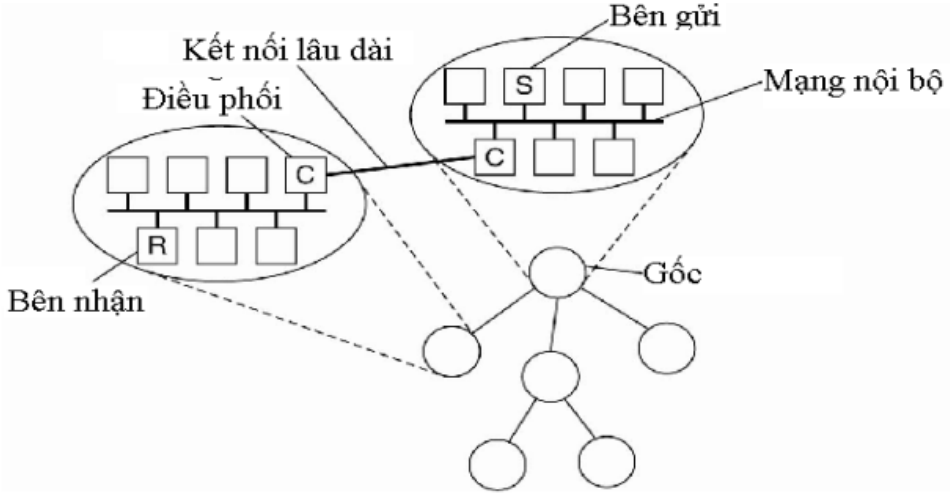
- Chi phí cao
- Độ trễ
- Phức tạp trong triển khai và bảo trì

# Phản hồi không phân cấp




- Sender gửi thông điệp đến các Receiver, Receiver chỉ phản hồi thông điệp NACK
- NACK được gửi cho toàn bộ thành viên trong nhóm NACK và Sender
- Sử dụng giao thức tự triệt tiêu thông điệp NACK
- Thiết lập lịch phản hồi NACK cho từng Receiver

# Phản hồi phân cấp




- Sử dụng cho nhóm với số lượng các tiến trình rất lớn
- Thông điệp chỉ gửi đến nhóm các tiến trình cao nhất, chúng tự gửi đến các phân cấp nhỏ hơn
- Trong các tiến trình có phân cấp nhỏ hơn có thể sử dụng các phản hồi ACK hoặc NACK
- Cần có bộ đệm riêng của mỗi nhóm phục vụ việc không nhận được thông điệp sẽ gửi yêu cầu lên cấp cao hơn

# Cam kết phân tán

## Distributed commit

# Cam kết phân tán

Đảm bảo một hoạt động được thực hiện

Bởi tất cả các thành viên trong một nhóm xử lý

Hoặc không được thực hiện bởi bất kỳ thành viên nào

# Cam kết phân tán – Ví dụ

Đặt vé máy bay

Dịch vụ A: Đặt chỗ ngồi trên máy bay

Dịch vụ B: Thanh toán tiền vé

Dịch vụ C: Gửi email xác nhận

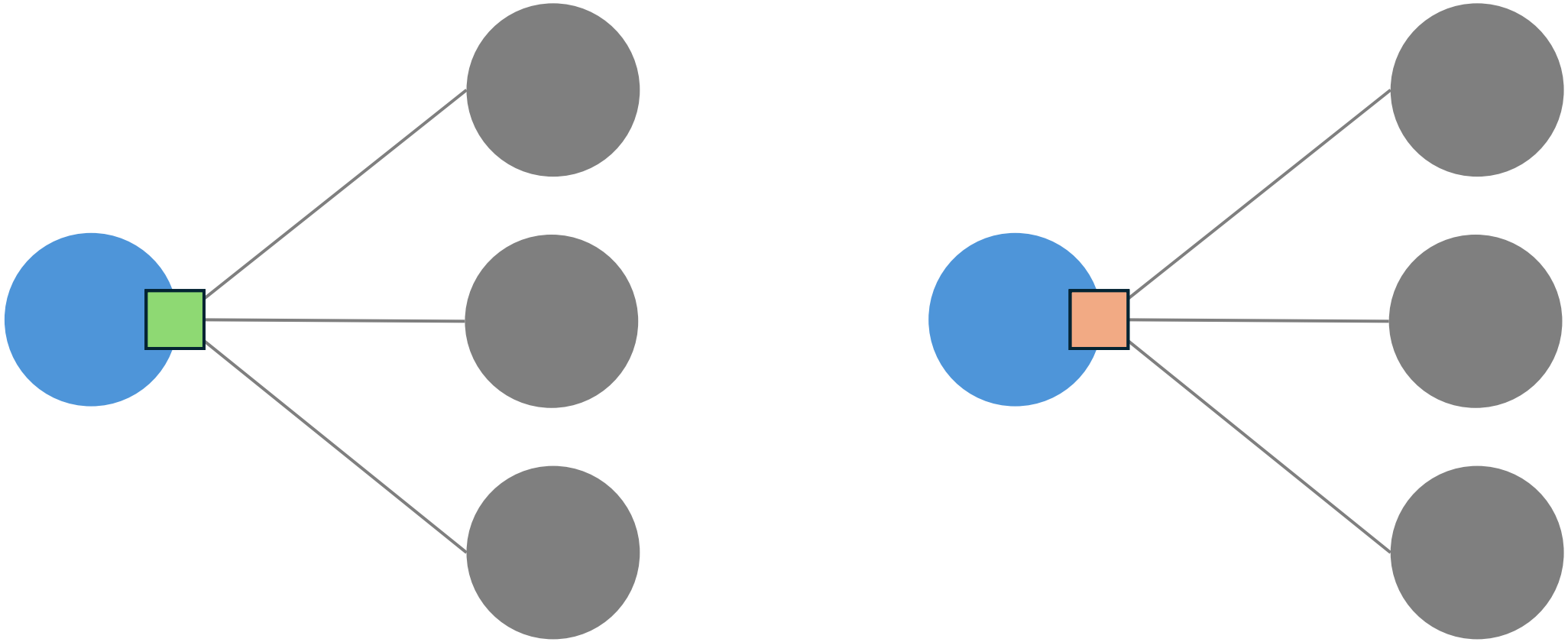
# Cam kết phân tán – Giao thức

Giao thức cam kết một pha (1PC)

Giao thức cam kết hai pha (2PC)

Giao thức cam kết ba pha (3PC)

# Cam kết phân tán – Giao thức **1PC**



# Cam kết phân tán – Giao thức 1PC

**Ưu điểm**

- Đơn giản
- Hiệu suất cao
- Phù hợp môi trường tin cậy

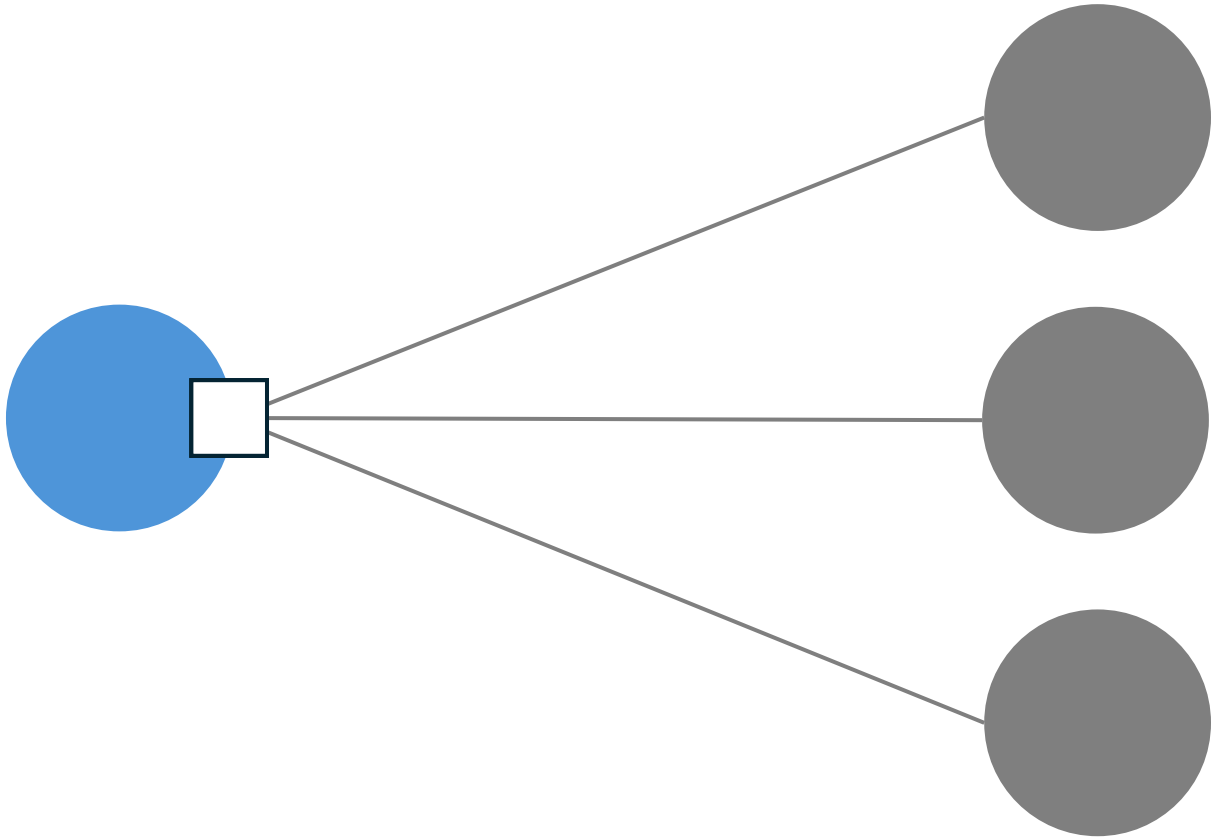
**Nhược điểm**

- Không chịu lỗi tốt
- Không đảm bảo tính nhất quán
- Không phù hợp hệ thống phân tán

# Cam kết phân tán – Giao thức 2PC

Phase 1: Prepare
(Voting Phase)

Phase 2: Commit/Abort
(Decision Phase)

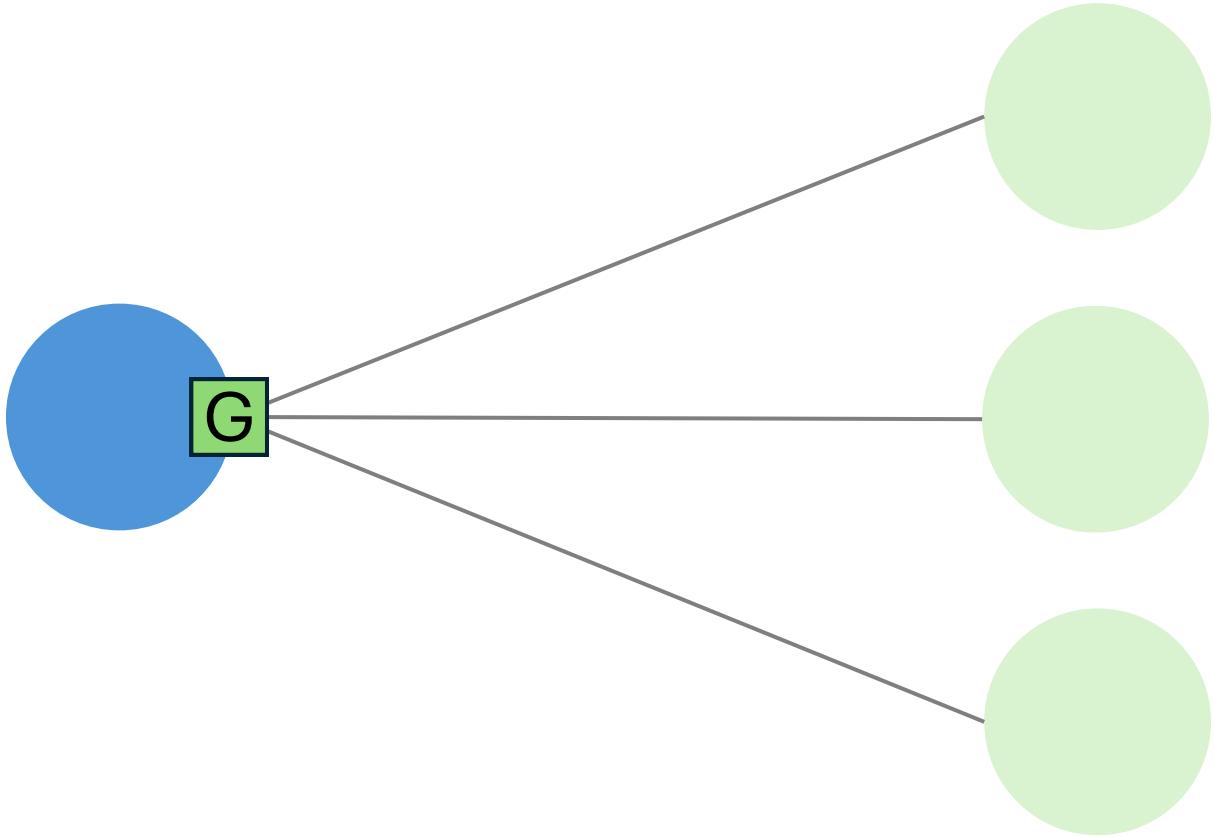


**Trường hợp tất cả vote commit**

# Cam kết phân tán – Giao thức 2PC

Phase 1: Prepare
(Voting Phase)

Phase 2: Commit/Abort
(Decision Phase)

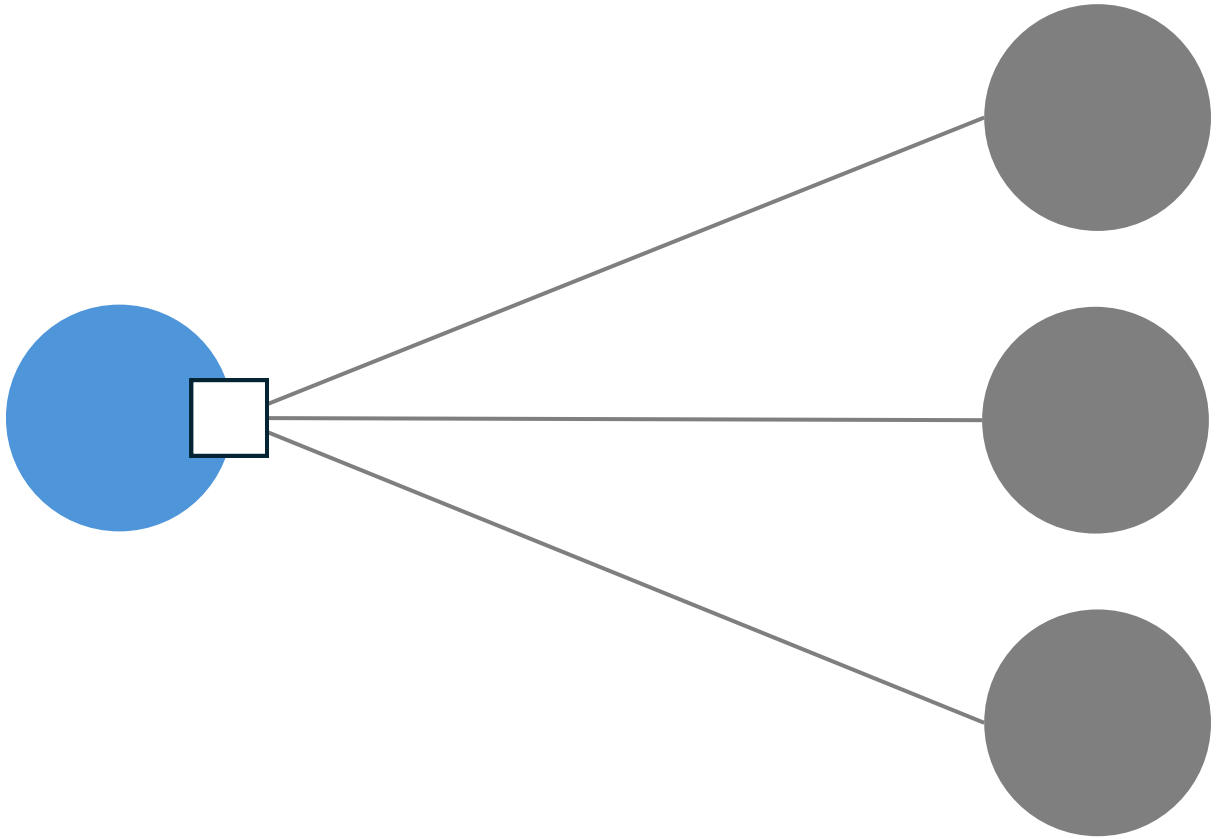



**Trường hợp tất cả vote commit**

# Cam kết phân tán – Giao thức 2PC

Phase 1: Prepare
(Voting Phase)

Phase 2: Commit/Abort
(Decision Phase)

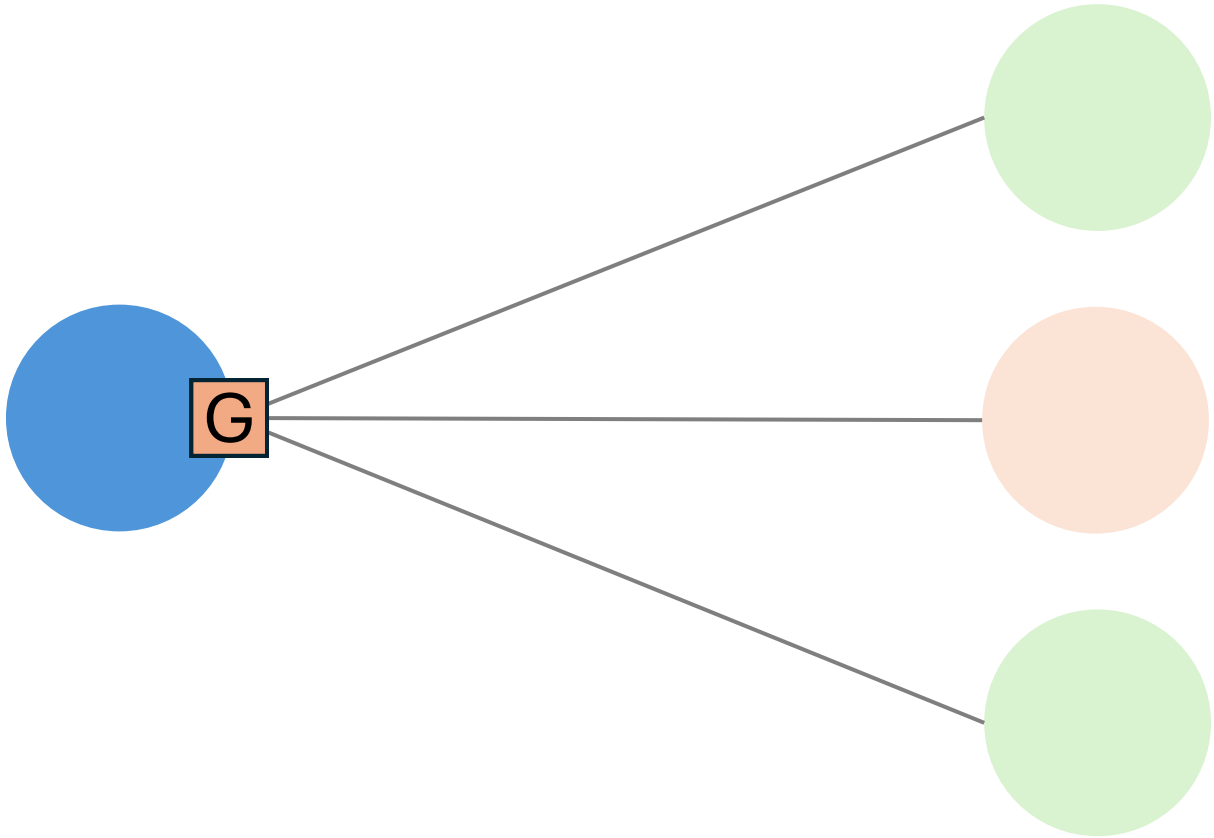


**Trường hợp có thành viên vote abort**

# Cam kết phân tán – Giao thức 2PC

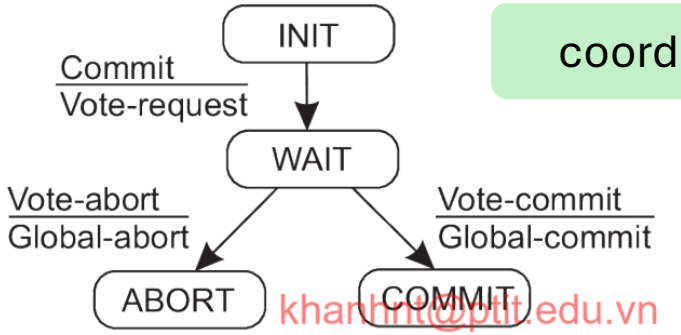
Phase 1: Prepare
(Voting Phase)

Phase 2: Commit/Abort
(Decision Phase)

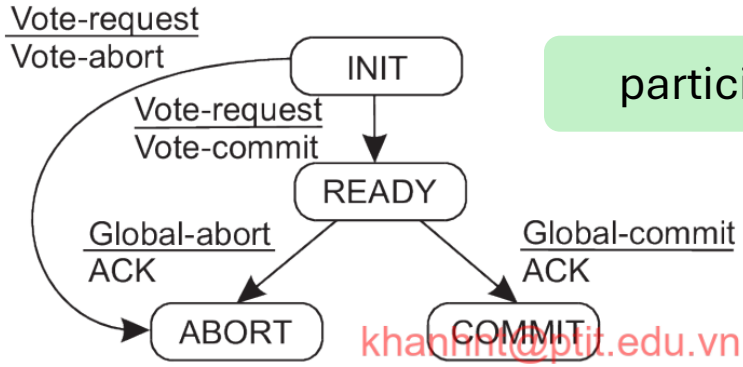



**Trường hợp có thành viên vote abort**

# Cam kết phân tán – Giao thức 2PC




coordinator




participant

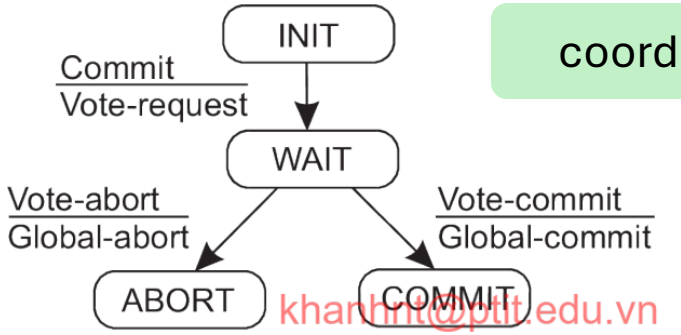


**Các trạng thái bị chặn trong 2PC**

- Participant ở trạng thái INIT
- Coordinator ở trạng thái WAIT
- Participant ở trạng thái READY

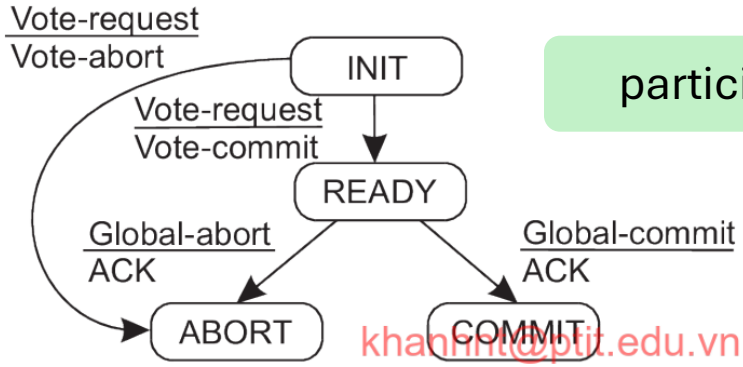
*(Nguồn: M. van Steen and A.S. Tanenbaum, Distributed Systems, 3rd ed., distributed-systems.net, 2017.)*

# Cam kết phân tán – Giao thức 2PC




coordinator






participant



Participant P bị chặn ở trạng thái READY

- P chờ coordinator khôi phục
- P liên hệ với participant Q

| Trạng thái của Q | Hành động của P |
|---|---|
| COMMIT | Chuyển COMMIT |
| ABORT | Chuyển ABORT |
| INIT | Chuyển ABORT |
| READY | Liên hệ participant khác |

*(Nguồn: M. van Steen and A.S. Tanenbaum, Distributed Systems, 3rd ed., distributed-systems.net, 2017.)*

# Cam kết phân tán – Giao thức 2PC

**Ưu điểm**

- Đảm bảo tính nhất quán toàn cục
- Cấu trúc đơn giản
- Phù hợp hệ thống phân tán vừa phải
- Là nền tảng cho các giao thức cao hơn

**Nhược điểm**

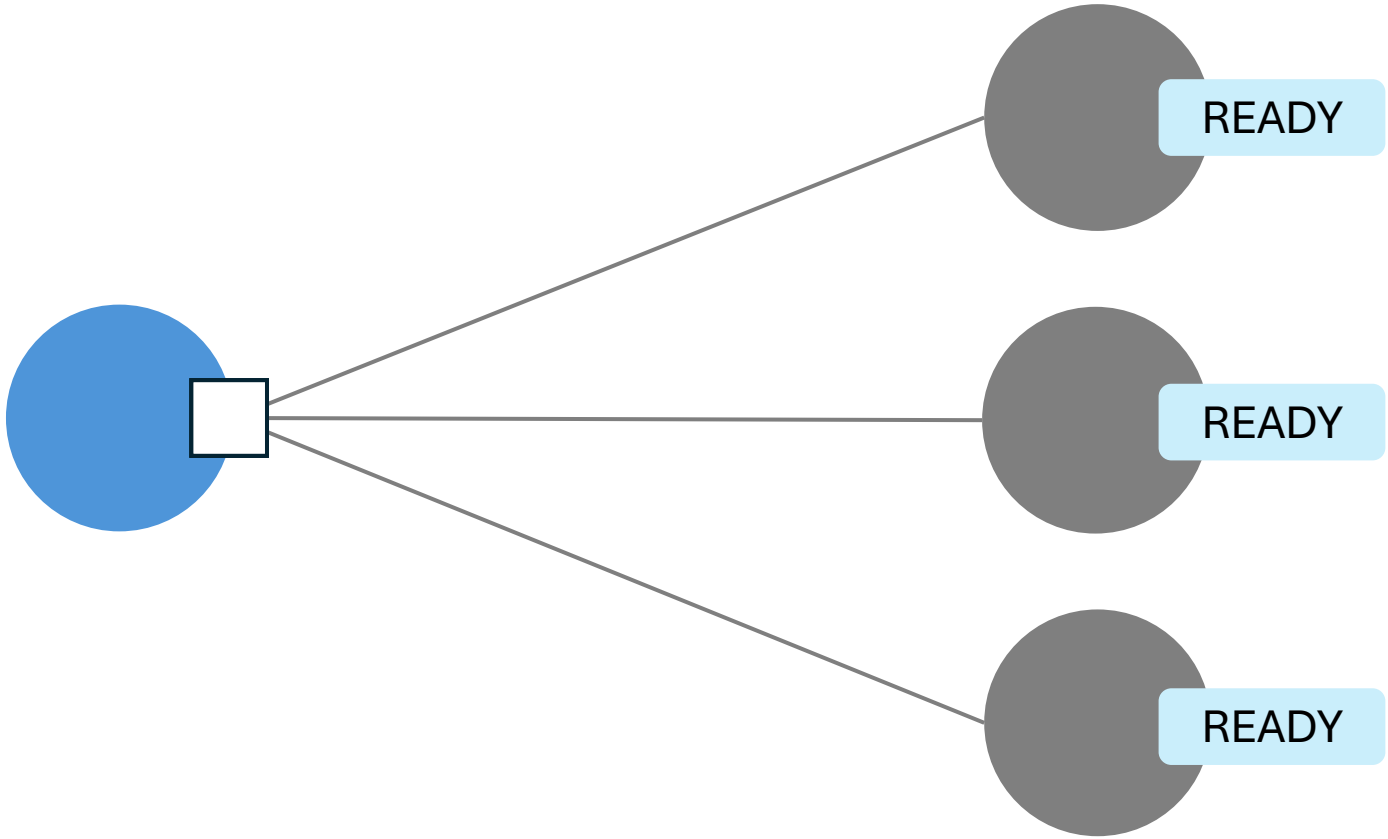
- Không chịu lỗi tốt
- Không thể tự abort
- Chậm hơn 1PC
- Phụ thuộc vào coordinator trung tâm

# Cam kết phân tán – Giao thức 3PC

Phase 1: CanCommit

Phase 2: PreCommit

Phase 3: DoCommit

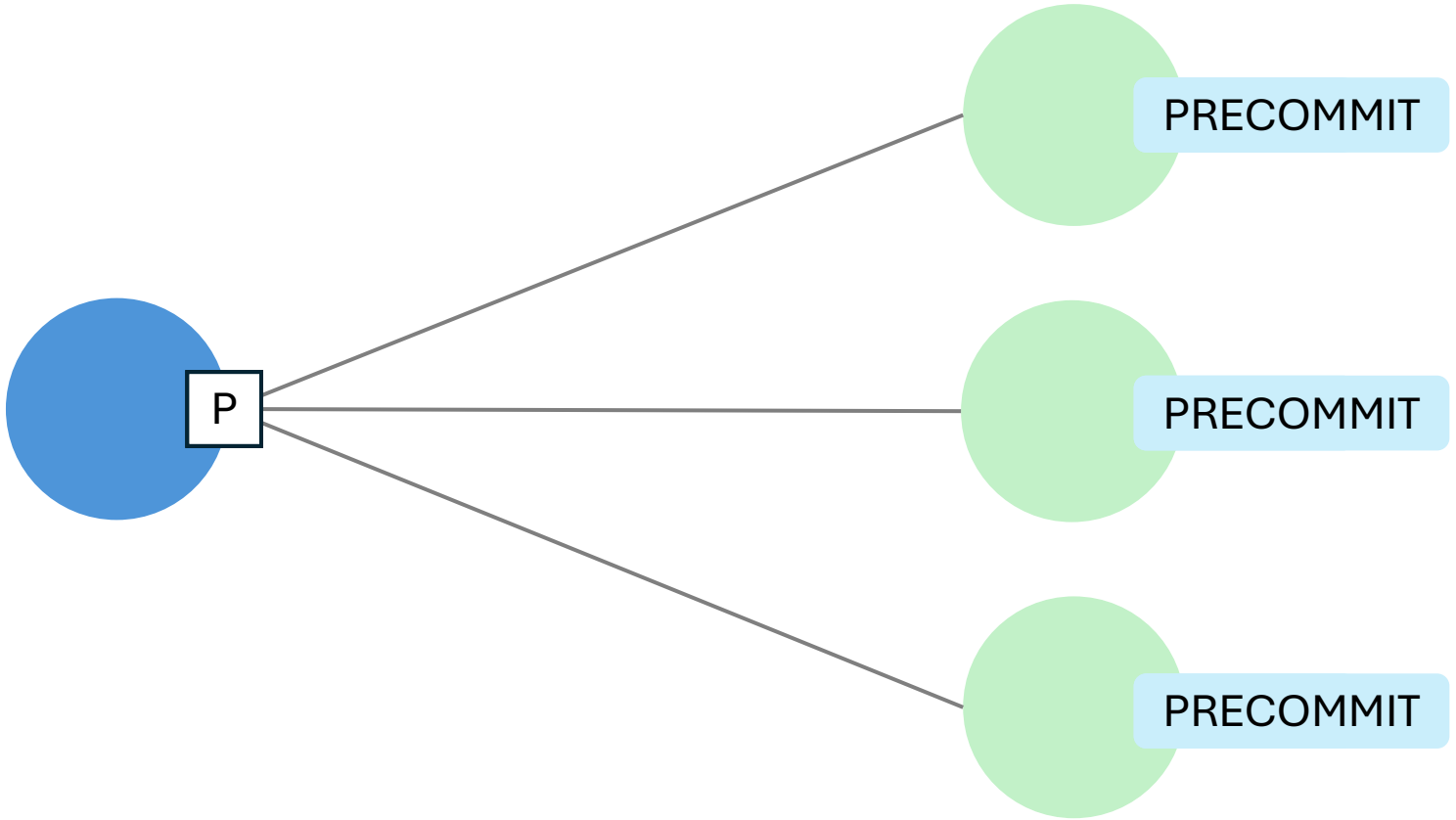


# Cam kết phân tán – Giao thức **3PC**

Phase 1: CanCommit

Phase 2: PreCommit

Phase 3: DoCommit

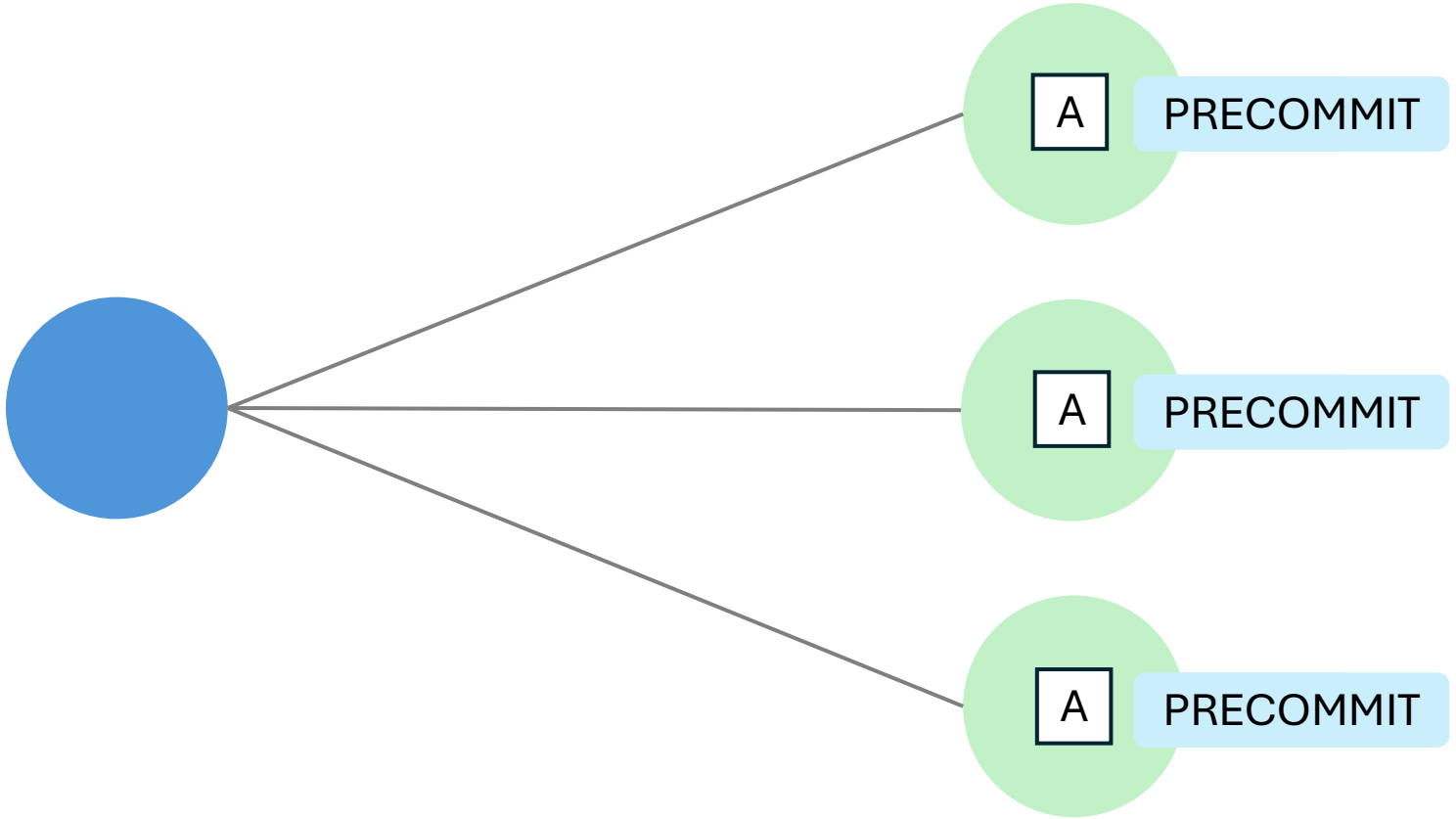


# Cam kết phân tán – Giao thức **3PC**

Phase 1: CanCommit

Phase 2: PreCommit

Phase 3: DoCommit

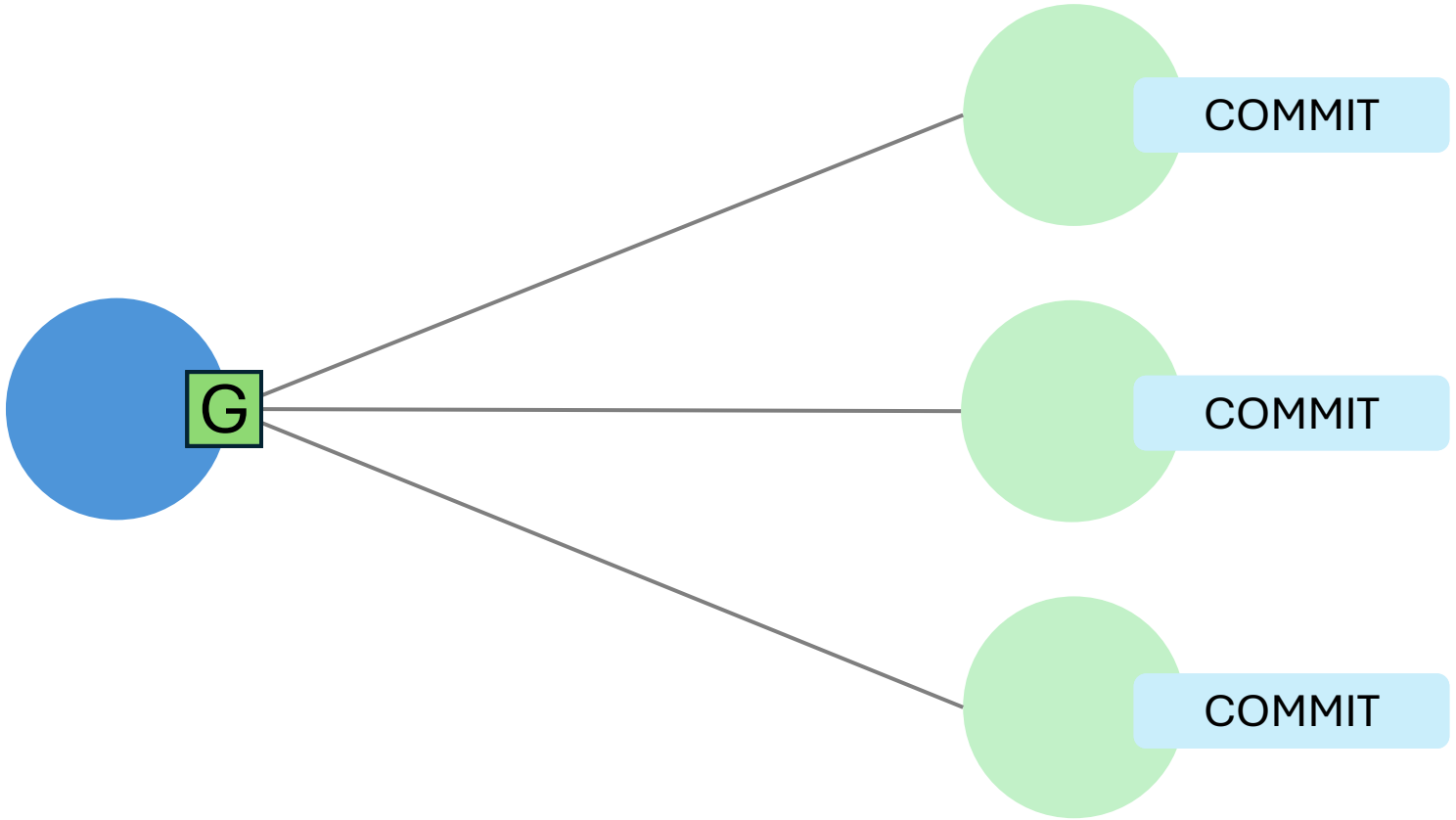


# Cam kết phân tán – Giao thức **3PC**

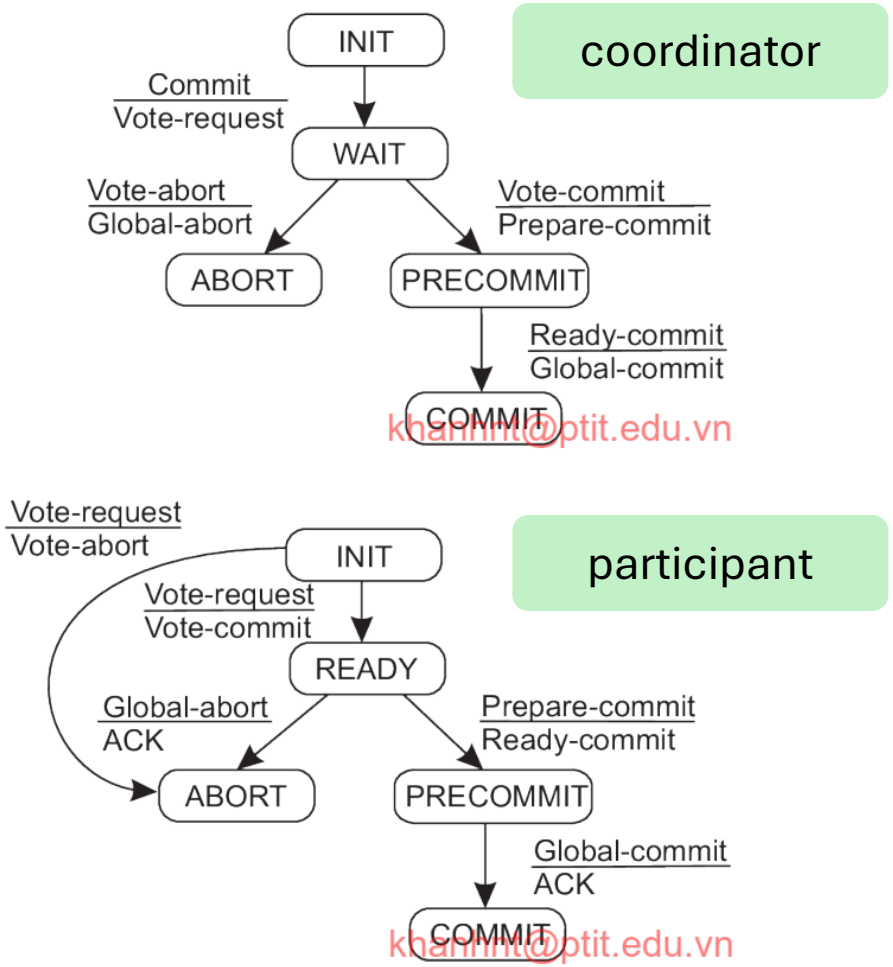
Phase 1: CanCommit

Phase 2: PreCommit

Phase 3: DoCommit



# Cam kết phân tán – Giao thức 3PC





**Các trạng thái bị chặn trong 3PC**

- Participant ở trạng thái INIT
- Coordinator ở trạng thái WAIT
- Coordinator ở trạng thái PRECOMMIT
- Participant ở trạng thái READY hoặc PRECOMMIT

*(Nguồn: M. van Steen and A.S. Tanenbaum, Distributed Systems, 3rd ed., distributed-systems.net, 2017.)*

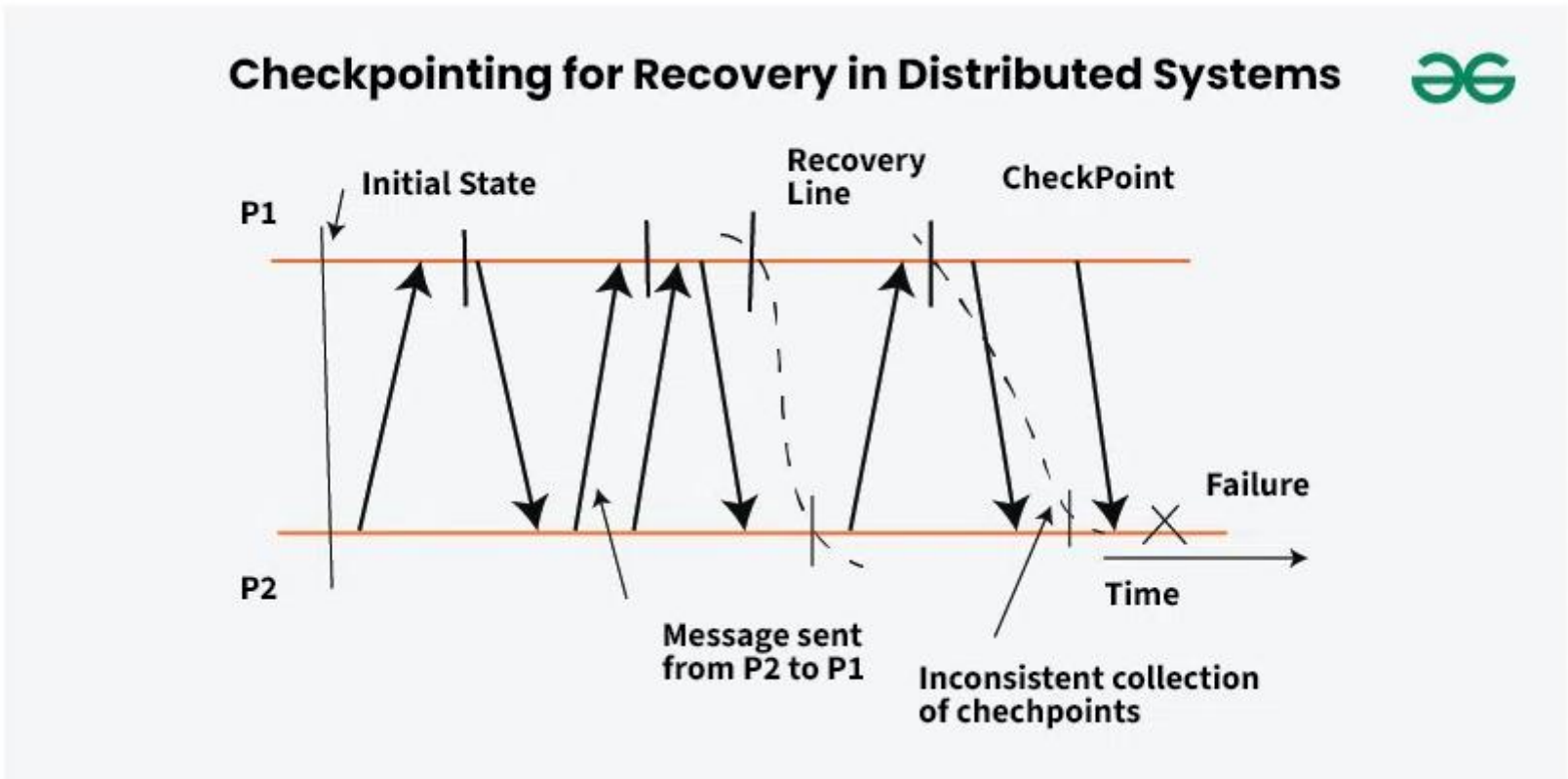
# Hồi phục

## Recovery

# Recovery - Hồi phục

- Mục tiêu của khôi phục (Recovery): đảm bảo hệ thống tiếp tục hoạt động sau khi xảy ra lỗi.
- Hai dạng lỗi chính:
- *Process failure*: một tiến trình dừng bất ngờ.
- *Communication failure*: mất gói tin, mất kết nối giữa các tiến trình.
- Chiến lược cơ bản: lưu trạng thái hệ thống & tái thiết trạng thái hợp lệ sau lỗi.

# Checkpointing



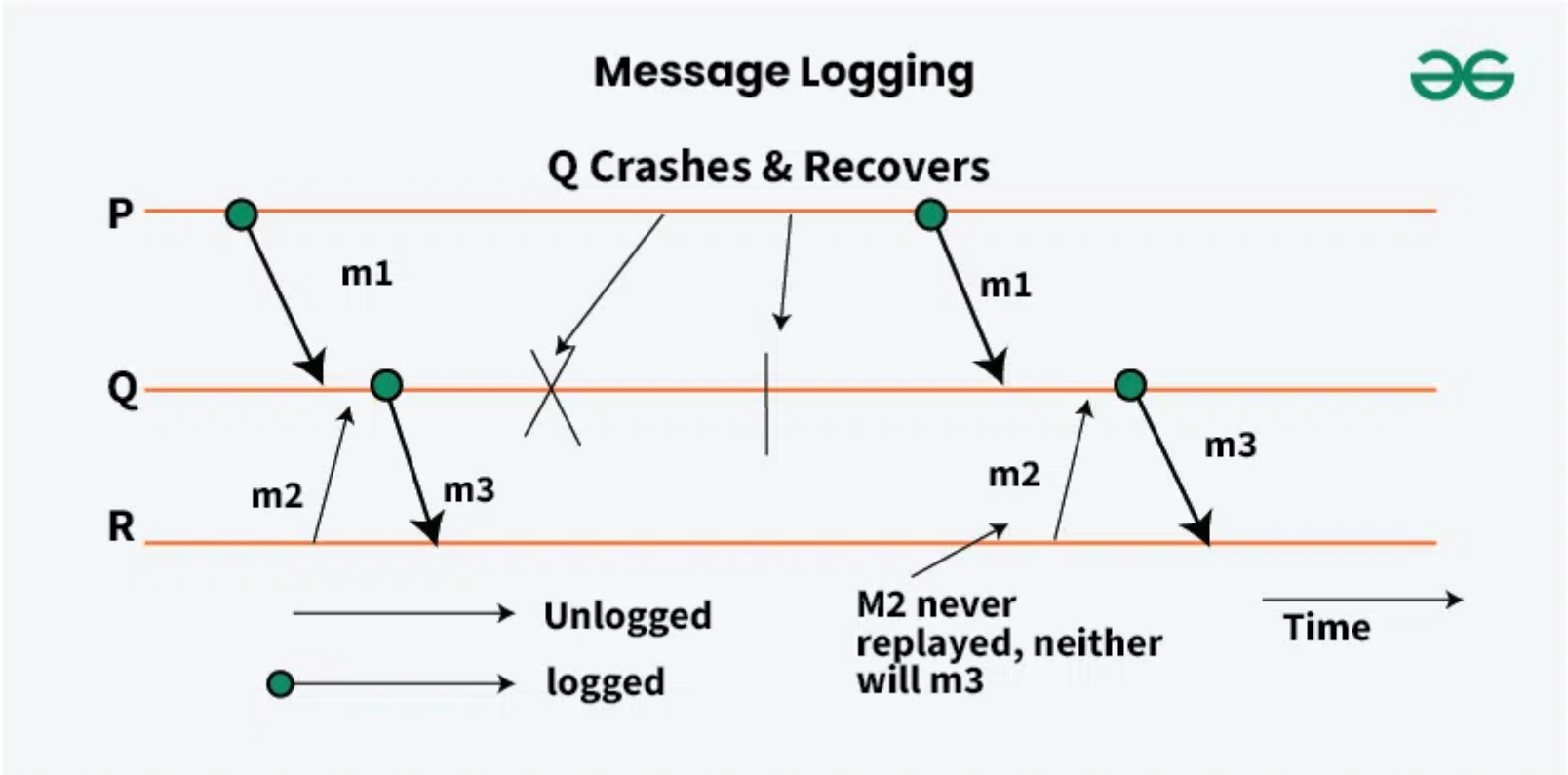
# Checkpointing - Điểm kiểm tra

- *Checkpoint*: ghi lại trạng thái của tiến trình tại một thời điểm.
- Mục tiêu: giảm chi phí khôi phục bằng cách quay lại trạng thái đã lưu thay vì khởi động lại từ đầu.
- Loại checkpoint:
- Independent checkpointing
- Coordinated checkpointing
- Communication-induced checkpointing
- Vấn đề: *domino effect* — mất nhiều trạng thái do checkpoint không được phối hợp.

# Checkpointing – Ví dụ

**Tình huống:**
Một job xử lý dữ liệu chạy trong 5 giờ, nhưng hệ thống bị tắt điện sau 4 giờ.
**Ví dụ minh họa:**
Nếu hệ thống sử dụng **checkpointing** mỗi 30 phút, thì sau sự cố, job chỉ cần chạy lại từ checkpoint gần nhất, ví dụ từ giờ thứ 3.5 → tiết kiệm thời gian, chi phí so với việc chạy lại từ đầu.
**Mô tả cụ thể:**
- Spark hỗ trợ RDD Checkpointing để lưu trạng thái biến đổi của dữ liệu.
- Hadoop hỗ trợ NameNode checkpoint để phục hồi trạng thái HDFS.

# Message Logging



# Message Logging – Lưu trữ tin nhắn

- Ghi lại các thông điệp được gửi và nhận.
- Mục tiêu: giúp tiến trình phát lại chính xác trạng thái trước lỗi.

- Các kỹ thuật logging:
  - *Pessimistic logging*: đảm bảo an toàn trước khi tiếp tục.
  - *Optimistic logging*: ghi thông điệp song song, khôi phục nhanh hơn nhưng phức tạp.
  - *Causal logging*: cân bằng giữa hiệu suất và khả năng khôi phục.

# Message Logging – Ví dụ

**Tình huống:**
Một người dùng gửi tin nhắn lúc 10:00 nhưng server xử lý tin nhắn bị crash lúc 10:01.
**Ví dụ minh họa:**
Nếu hệ thống **message logging**, server có thể khôi phục lại bằng cách:
- Load trạng thái từ checkpoint cũ.
- Replay các tin nhắn được log lại để "bắt kịp" trạng thái mới.

**Chi tiết:**
- WhatsApp dùng hệ thống ghi log phân tán (write-ahead logs).
- Zalo có thể dùng hệ thống log để bảo đảm tin nhắn không mất khi có sự cố server.

# Recovery-Oriented Computing

**Tính toán định hướng khôi phục**

# Recovery-Oriented Computing

- Triết lý: lỗi là điều chắc chắn xảy ra.
- Mục tiêu: thiết kế hệ thống dễ khôi phục, chứ không chỉ tránh lỗi.
- Các nguyên tắc:
  - Tách biệt và giới hạn lỗi (isolation)
  - Hồi phục nhanh (fast recovery)
  - Kiểm tra định kỳ (proactive fault injection, testing)

# Recovery-Oriented Computing – Ví dụ: Dịch vụ Google Search

**Tình huống:**
Một máy chủ trong cụm server của Google bị lỗi bất ngờ.
**Ví dụ minh họa:**
Google thiết kế hệ thống theo hướng **Recovery-Oriented Computing**, nghĩa là:

- Máy chủ lỗi được cô lập tự động khỏi hệ thống.
- Một bản sao dịch vụ khác sẽ tiếp quản ngay lập tức.
- Đồng thời, Google chạy kiểm tra định kỳ (chaos testing) để chắc chắn hệ thống có thể khôi phục khi có lỗi.