MUTUAL EXCLUSION & QUORUMS

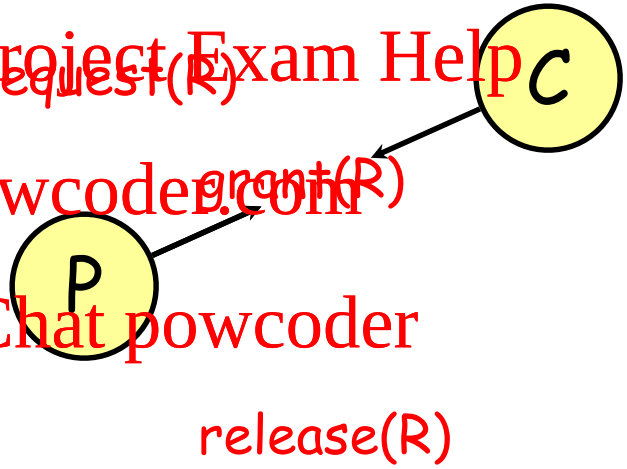
Distributed Mutual Exclusion

- Given a set of processes and a single resource, develop a protocol to ensure exclusive access to the resource by a single process at a time.

- This is a fundamental operation in operating systems, and is generalized to locking in databases.

Centralized algorithm

- One process elected as coordinator



1. **Request** resource
2. Wait for response
3. **Receive grant**
4. access resource
5. **Release resource**

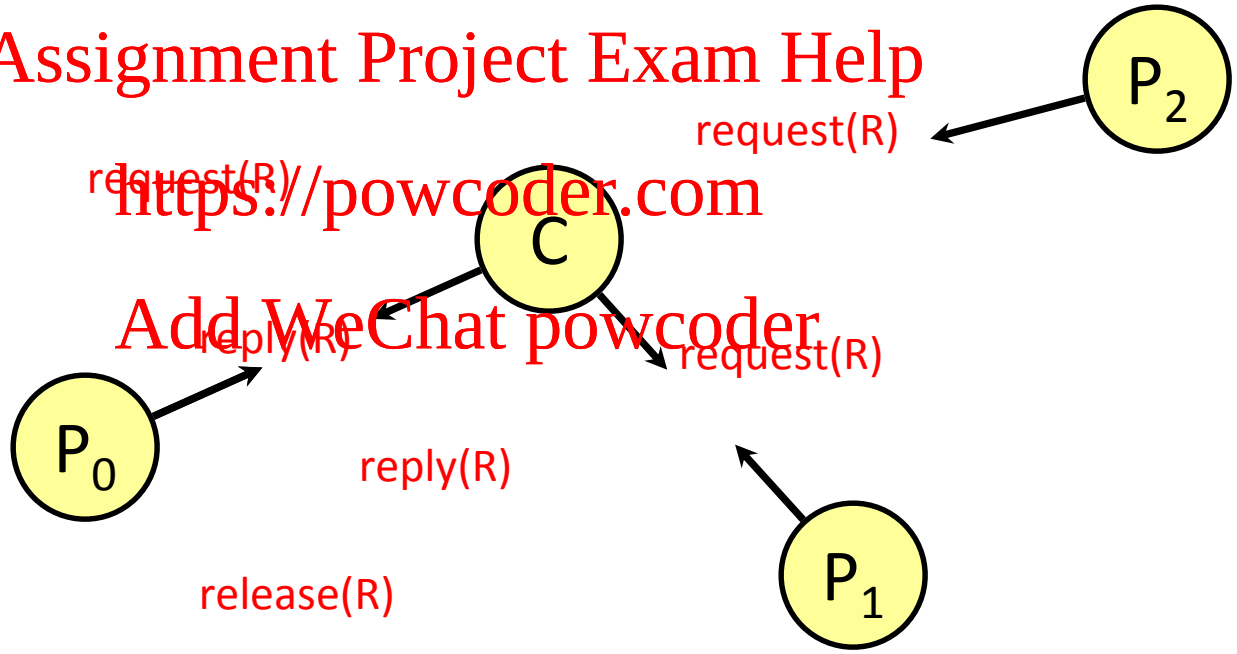
Centralized Solution

- But we can have multiple concurrent requests!
- Coordinator maintains queue of pending requests.
- Protocol:
  - Process send request to coordinator.
  - If no other request, coordinator sends back reply.
    - Otherwise, put request in queue
  - On receipt of reply, process accesses resource.
  - Once done, process sends release to coordinator.
  - On receipt of release, coordinator checks queue for any pending requests.

Centralized Solution

Queue

P_1
P_2



thanks paul krzyzanowski rutgers

Centralized Approach

- Does it satisfy requests in order the requests are made?

Lamport's Distributed Solution

- Instead of a central coordinator, all processes collectively
- Use similar approach, but engage ALL processes.
- Each request is time stamped with:
  - Lamport Logical Time + Identifier (process ID)
  - Ie, Timestamps are totally-ordered Lamport pairs.

Lamport's Distributed Solution

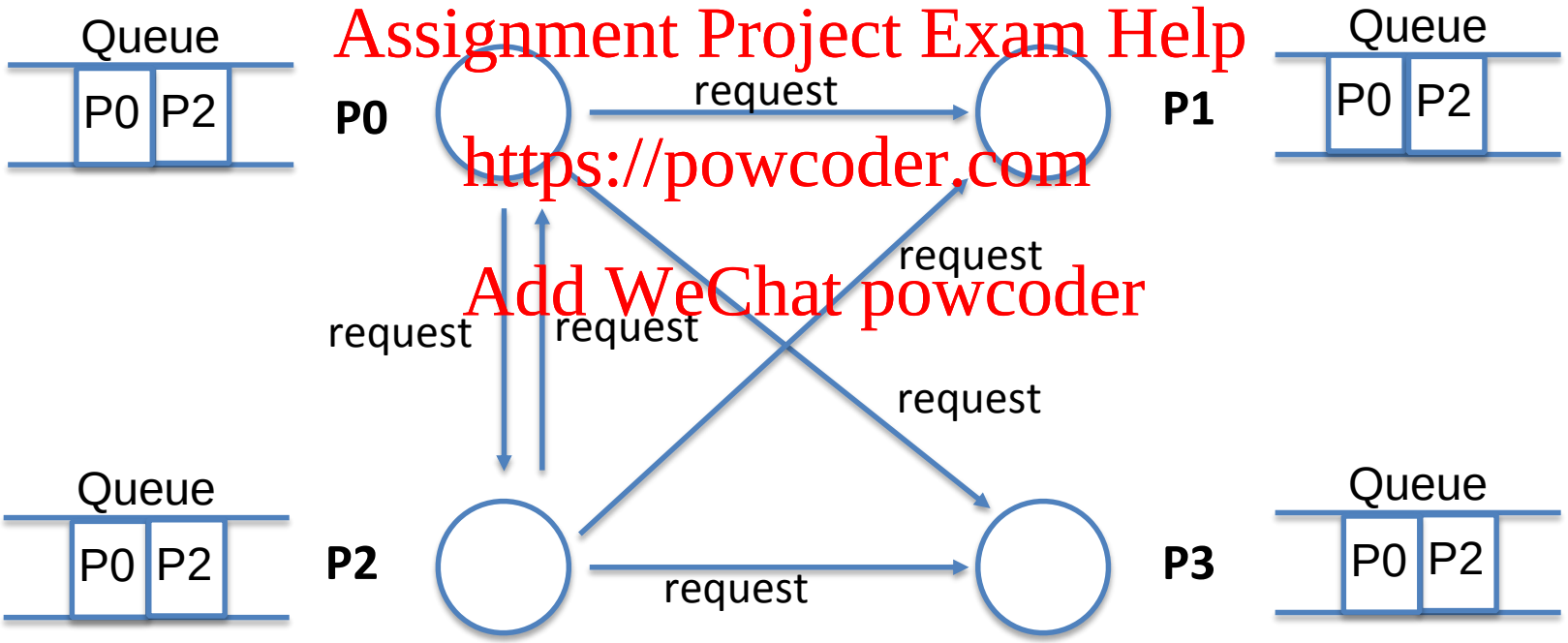
- Process sends time stamped request to all processes and puts request in local queue.
- On receipt of request
  - process puts request in queue and sends back reply.
- Process accesses resource
  - On receipt of all replies
  - Own request at head of queue
- Once done, process sends release to all processes.
- On receipt of release, process removes request

Distributed Solution

- Does this work (Lamport original solution)?
- Need to order queues so they are identical:
  - Use logical Lamport time + proc id to break ties.
  - FIFO channels
- Requests are executed in causal order.
  - Why?

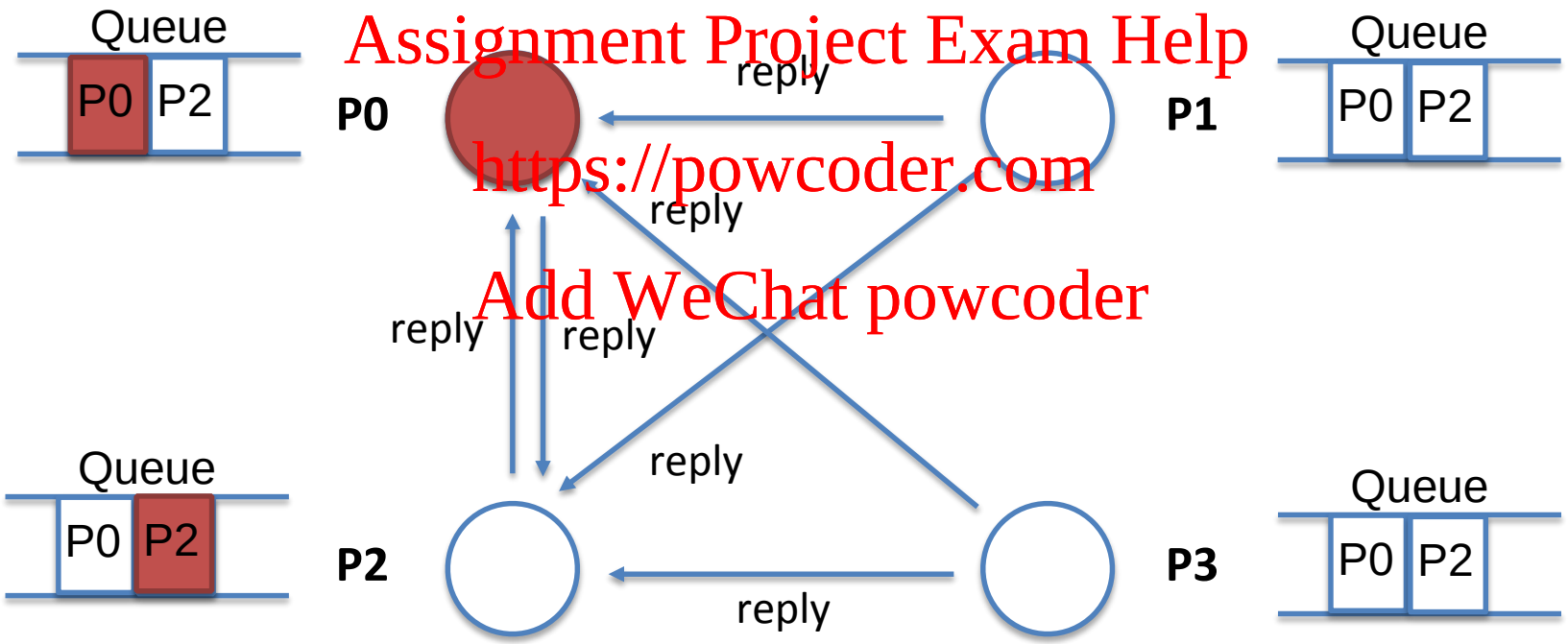
Example

Assume that process P0 and process P2 need to access the shared resource at the same time.



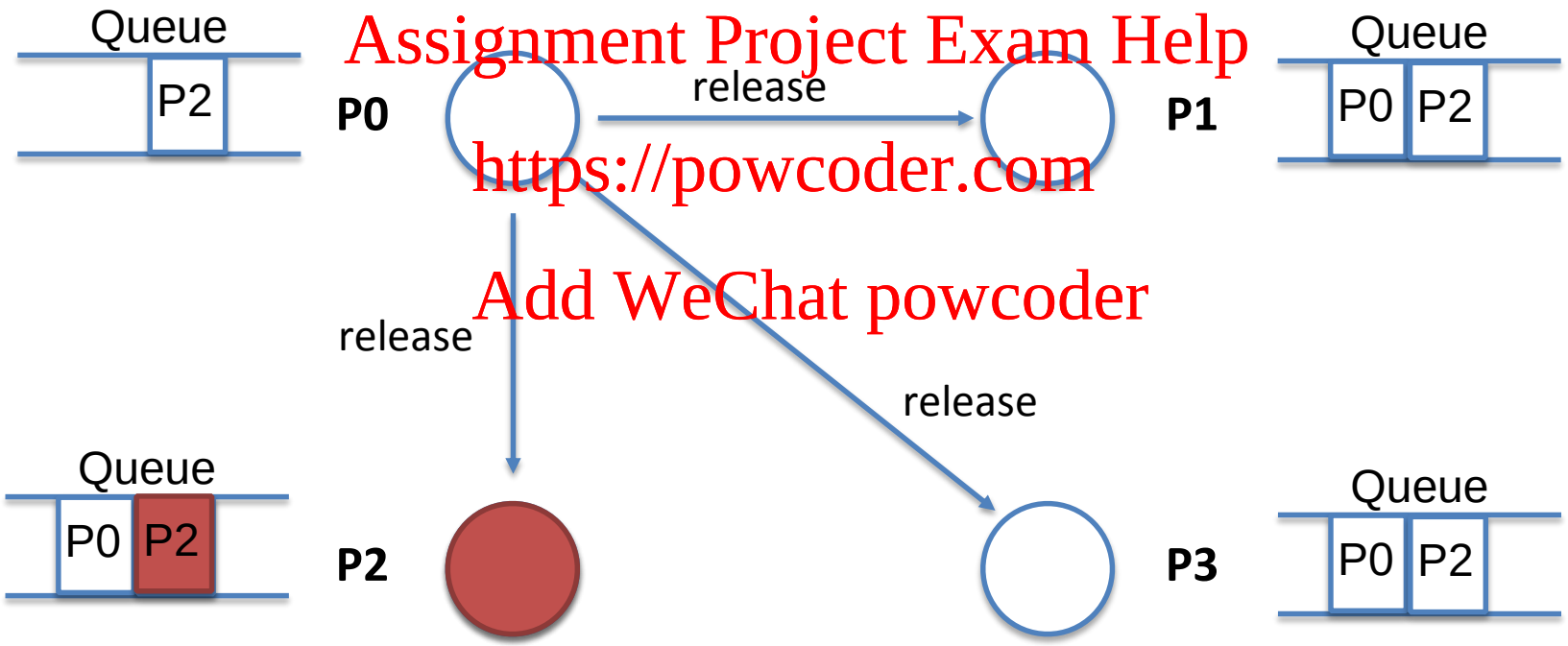
Example

Using process id to break ties, process P0's request will end up on top of the queue, over process P2's.



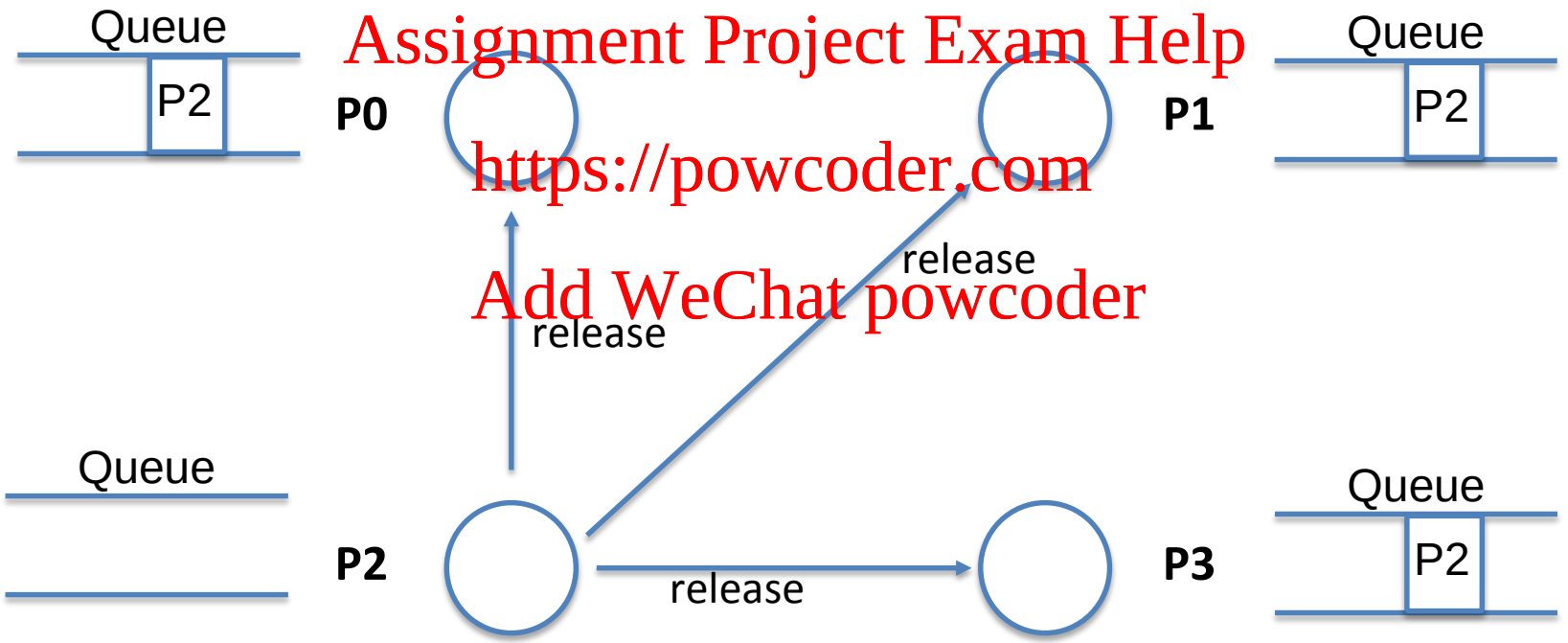
Example

Process P0 is done and broadcasts a release message.



Example

When process P2 is done with the shared resource, it also broadcasts the release message.



Complexity of Lamport's Solution

- This algorithm creates 3(N – 1) messages per request:
  - (N – 1) total number of requests
  - (N – 1) total number of replies
  - (N – 1) total number of releases
- Can we reduce the number of messages while maintaining the same guarantees?

Quorums

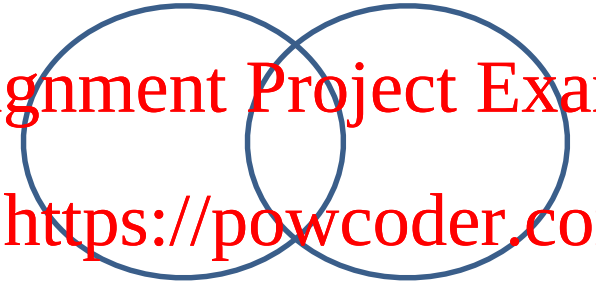
- What if there are failures?
  - Mostly network failures
  - No failure of the resource holder
- Do we need to communicate with ALL processes?
- A quorum is the minimum number of votes that a process has to obtain in order to be allowed access to the shared resource.

Quorums

- Any two requests should have a common process to act as an arbitrator.
- V_i is called a quorum.
- Let process p_i (p_j) request permission from V_i (V_j), then

Quorums

- Given N processes: $|V_i| > N/2$, ie,



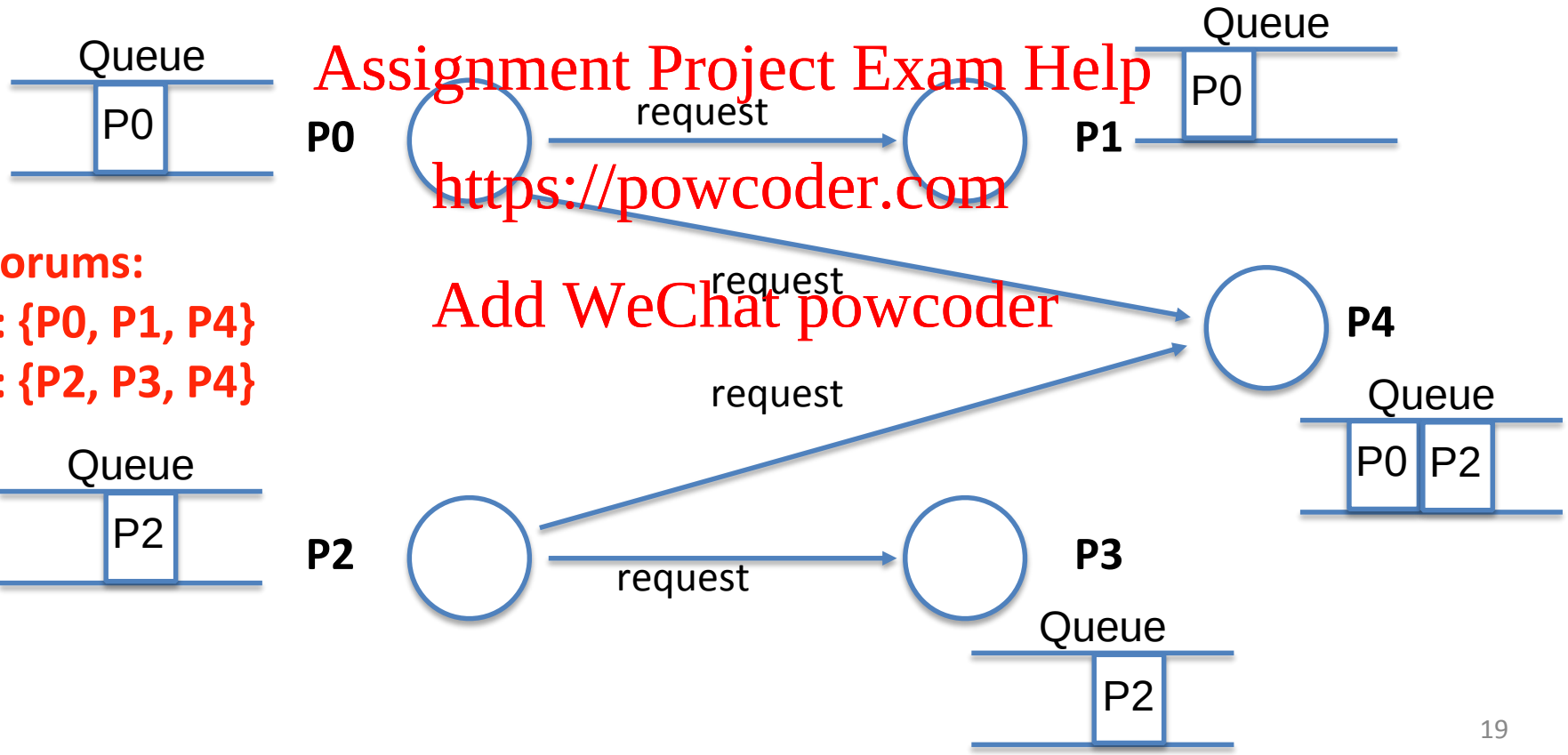
- In general, majority, ie $\lceil (N/2) \rceil$. [Gifford 79]

Basic MUTEX Protocol

- Requesting CS
  - process requests CS by sending *request* message to processes in its quorum
  - If a process receives a *request* it sends back *reply* unless it granted permission to other process; in which case the request is queued—similar to granting a lock
- Entering CS
  - process may enter CS when it receives *reply*s from all processes in its quorum
- Releasing CS
  - after exiting CS process sends *release* to every process in its quorum—release lock
  - when a process gets *release* it sends *reply* to another request in its queue

Example

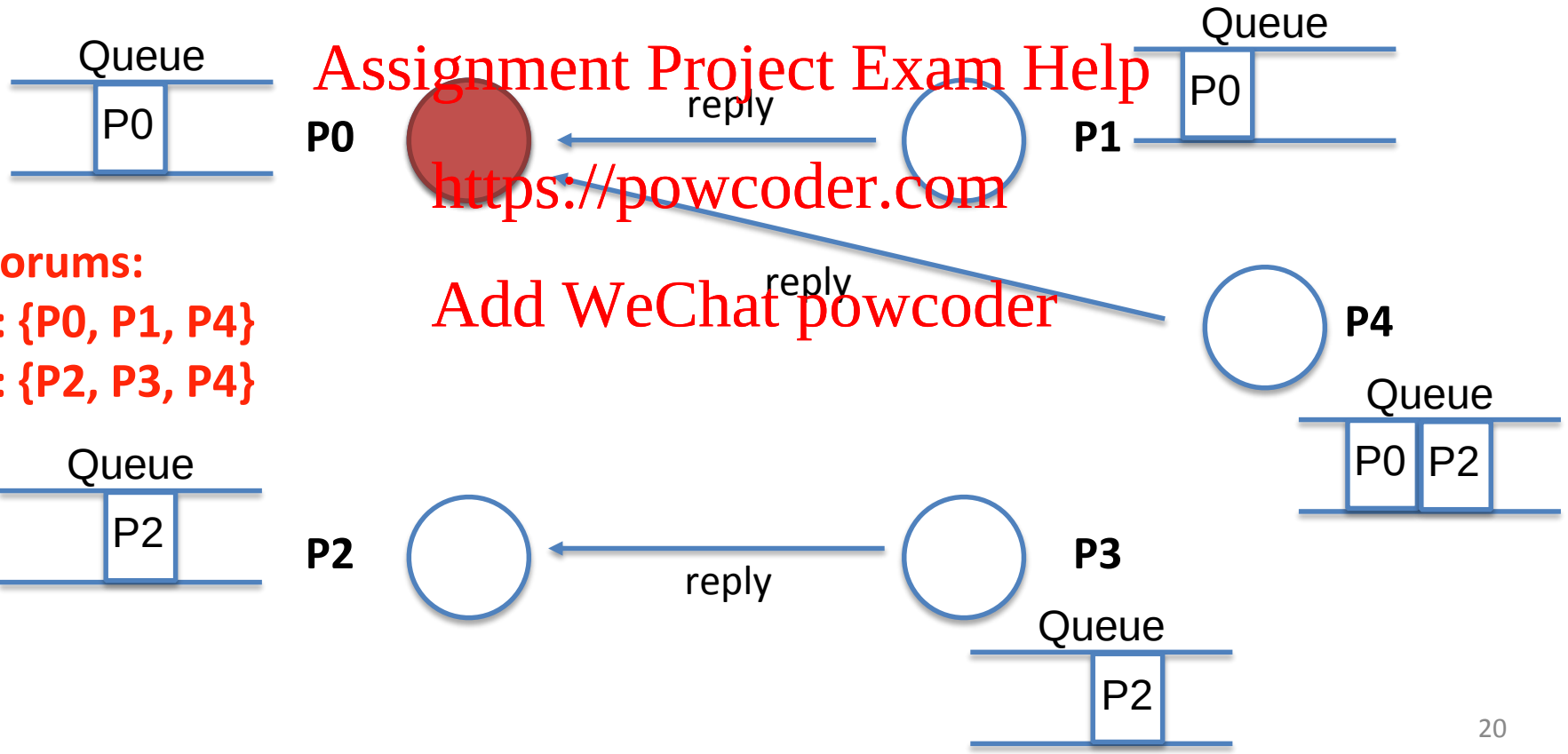
Assume that process P0 and process P2 need to access the shared resource at the same time.



Quorums:
V0: {P0, P1, P4}
V2: {P2, P3, P4}

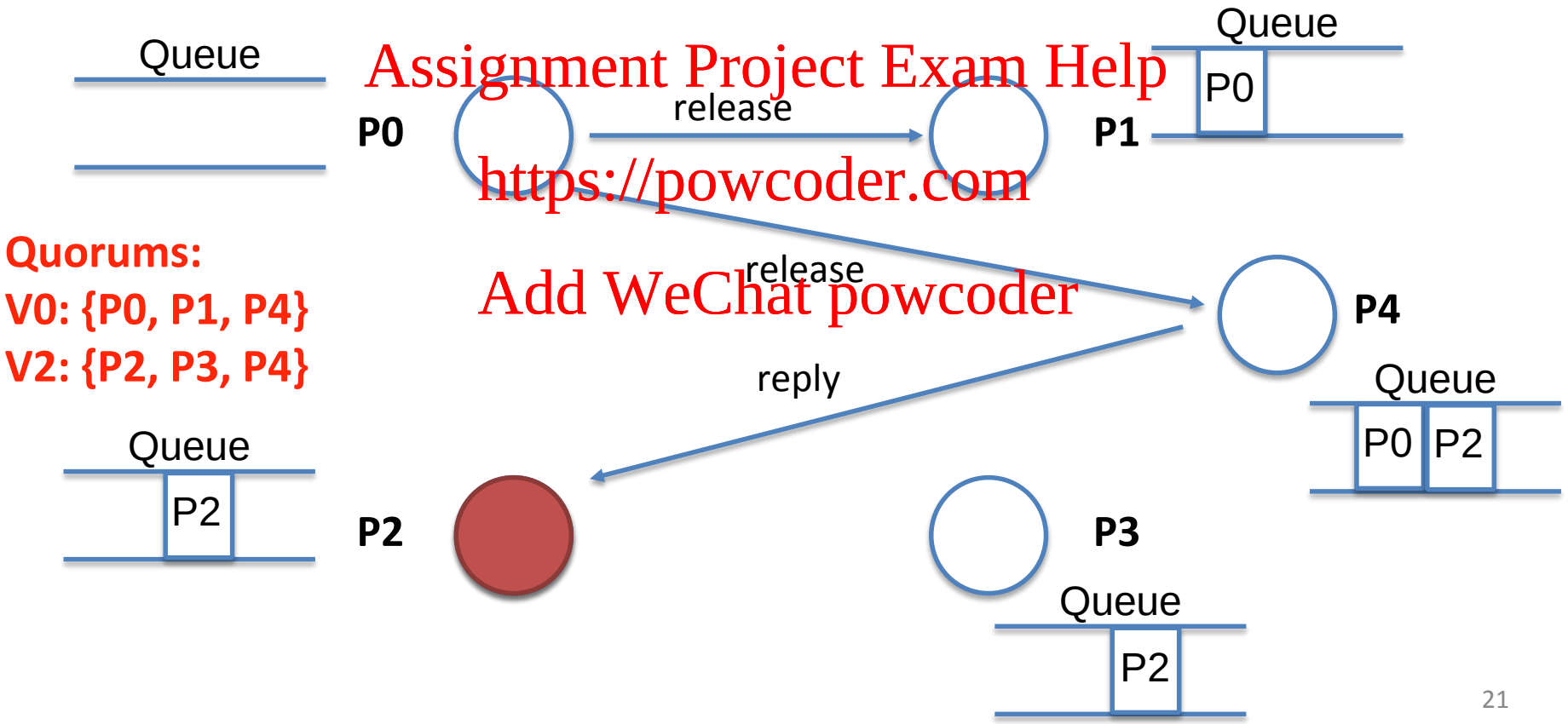
Example

Process P4 is the arbitrator since it's common in both V0 and V2 quorums.



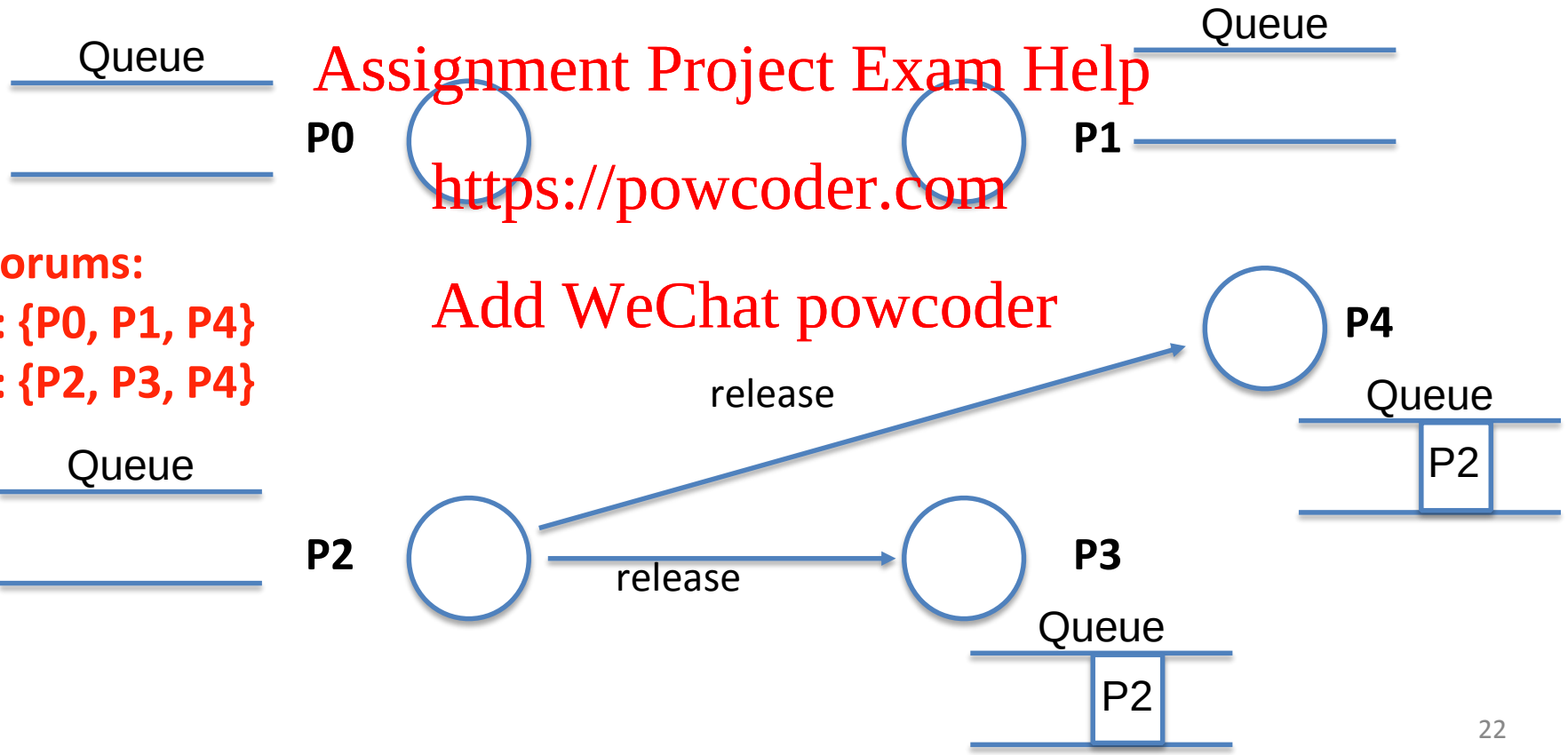
Example

Process P0 sends the release message to its quorum.



Example

Process P2 finishes and broadcasts to its quorum the release message.

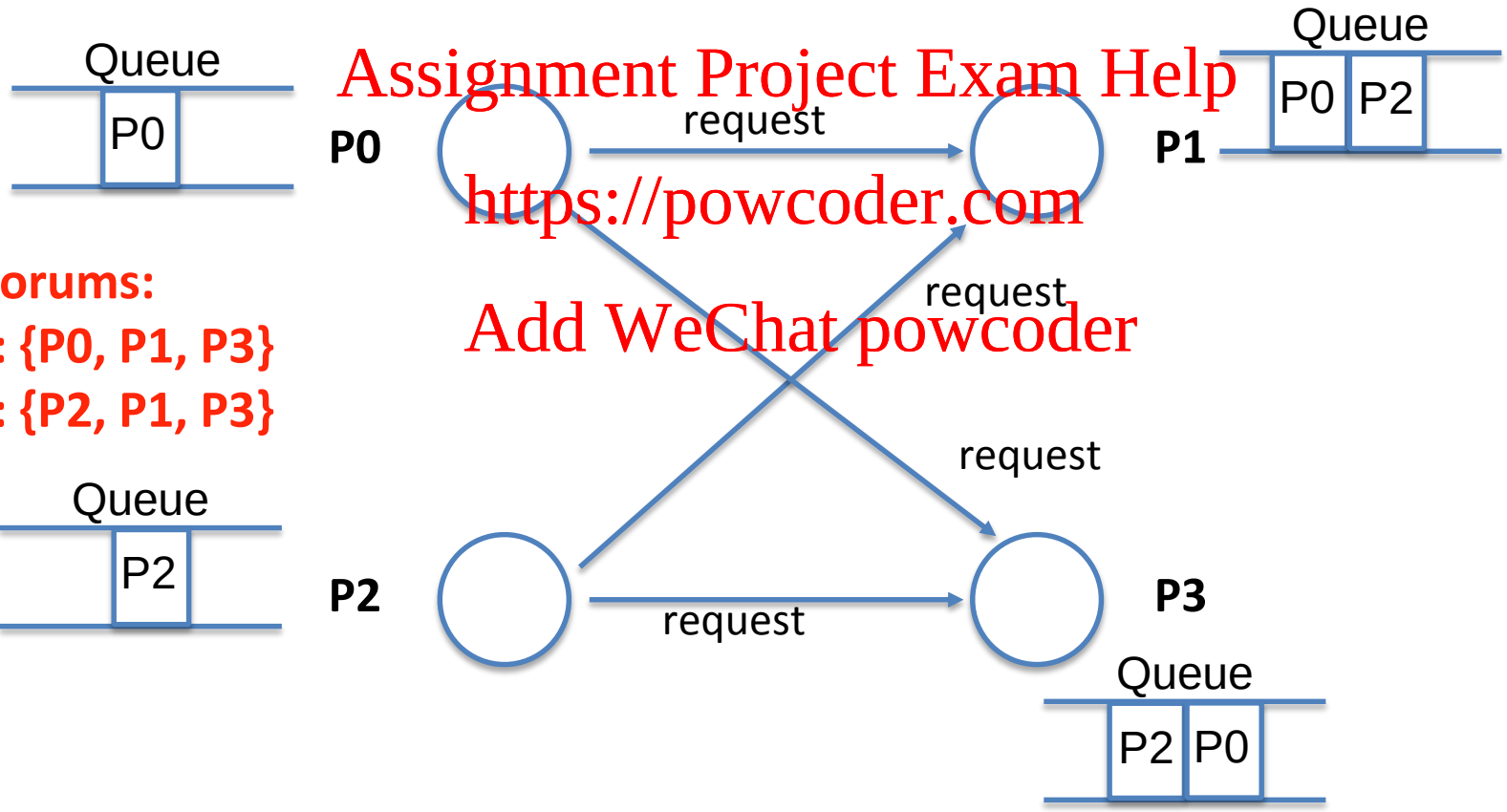


Deadlock

- Unfortunately, the quorums approach can lead to Deadlocks.
- Deadlocks occur even with a single resource.
  - Not the standard database or operating system multi-object deadlock.

Deadlock Example

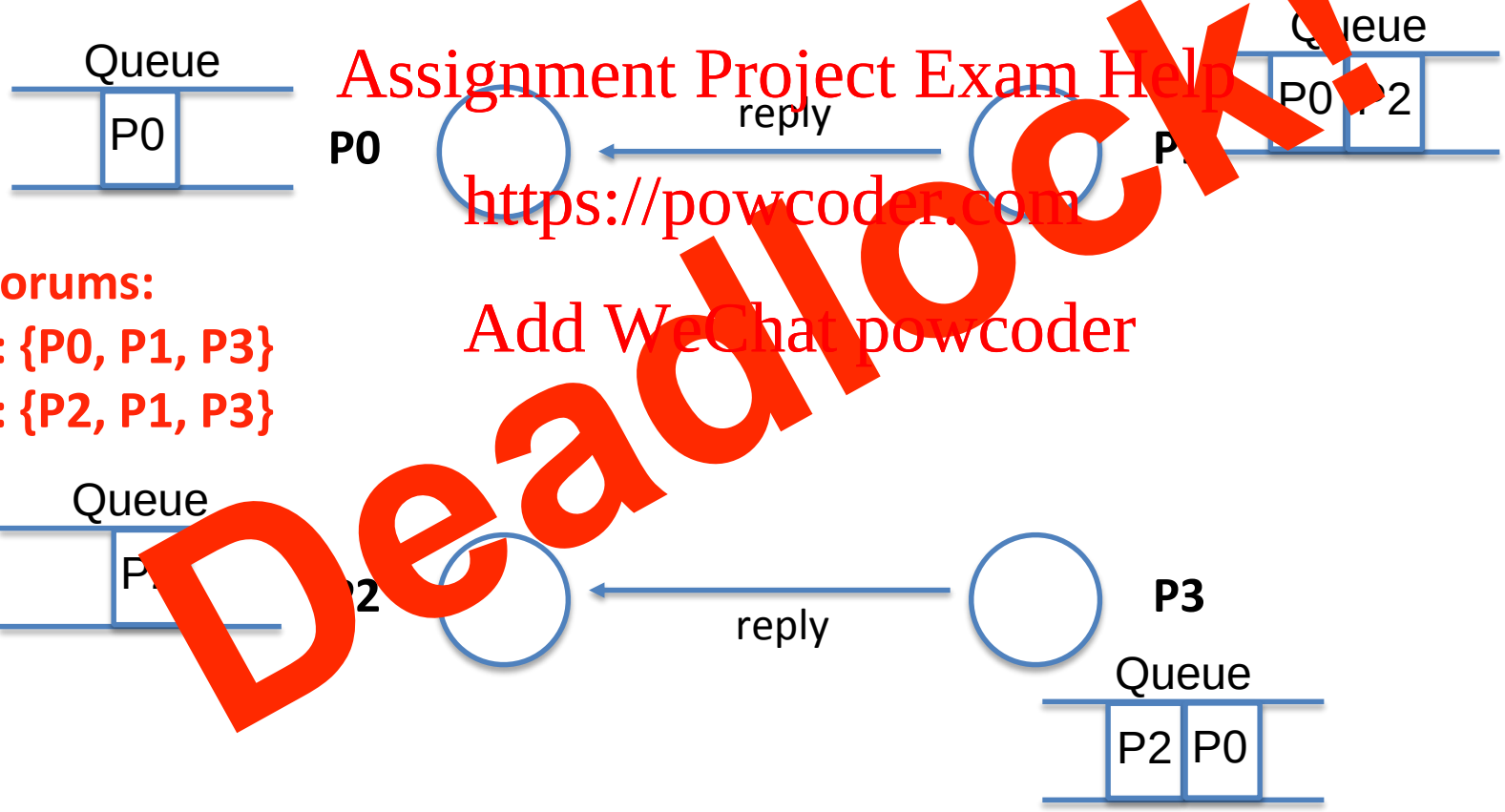
Assume that process P0 and process P2 need to access the shared resource at the same time.



Quorums:
V0: {P0, P1, P3}
V2: {P2, P1, P3}

Deadlock Example

P1 and P3 don't agree on which process should get the resource.

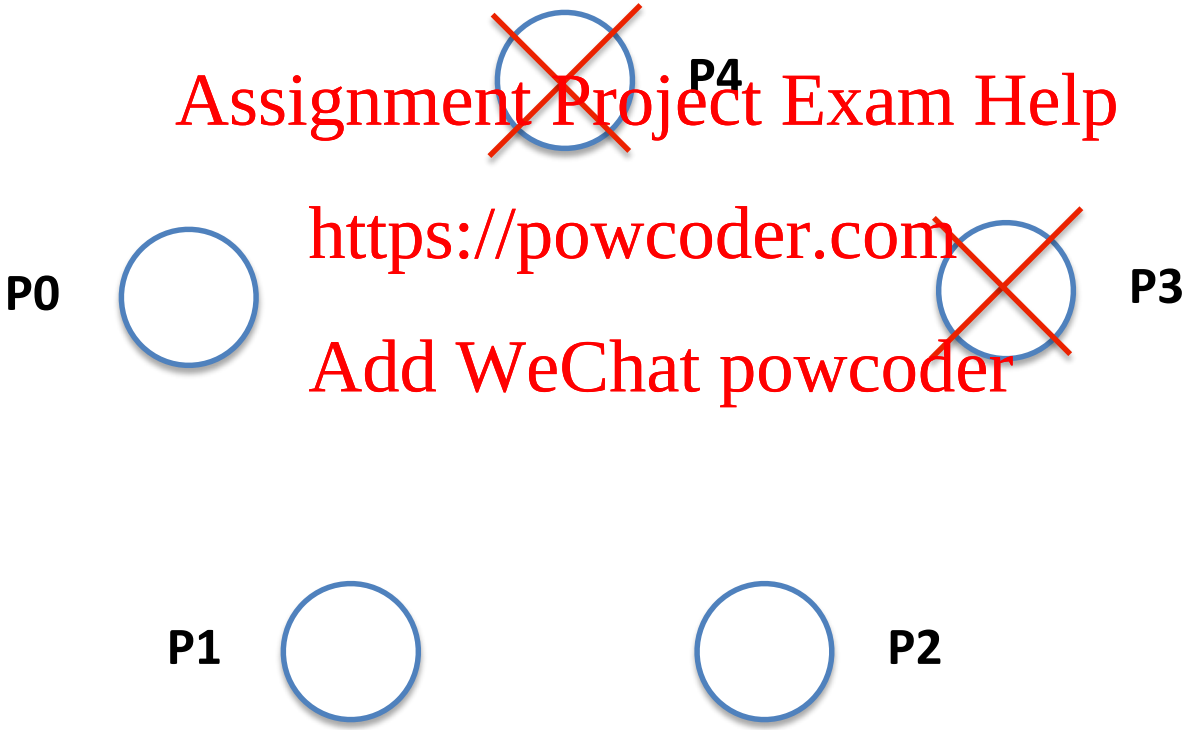


Deadlock!

Site Failure Example

How many sites can we afford to go down?

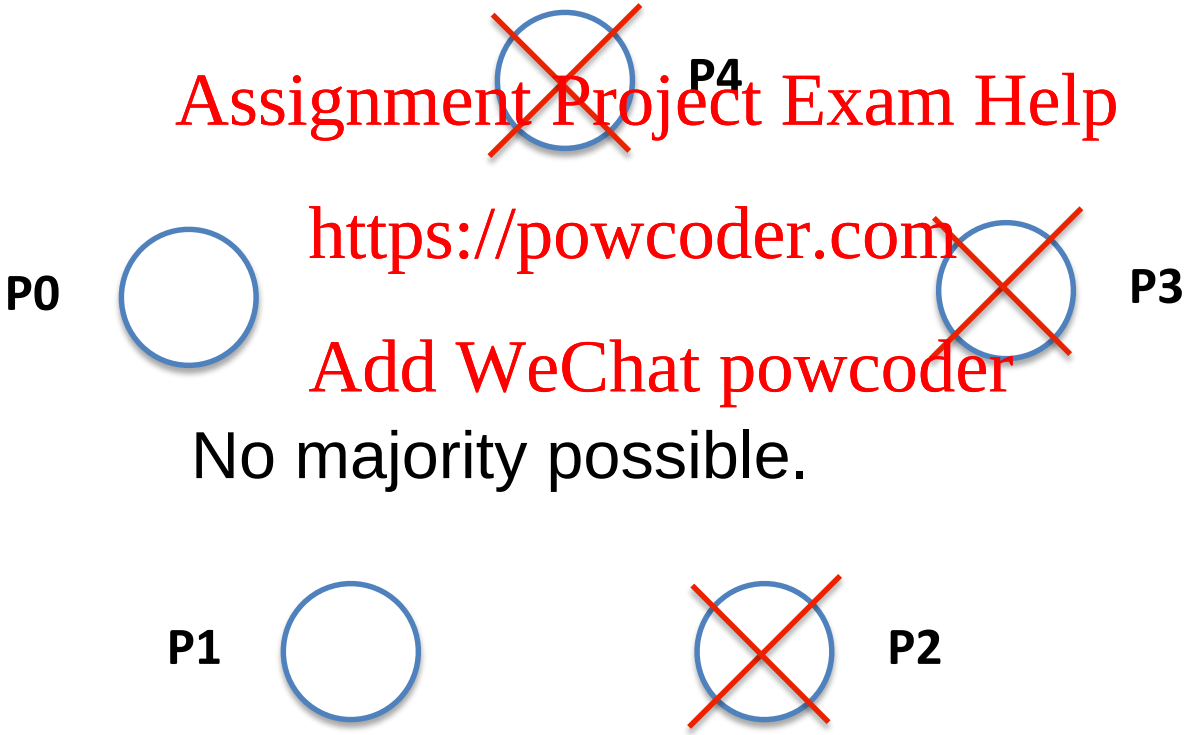
We can achieve quorum!



Site Failure Example

How many sites can we afford to go down?

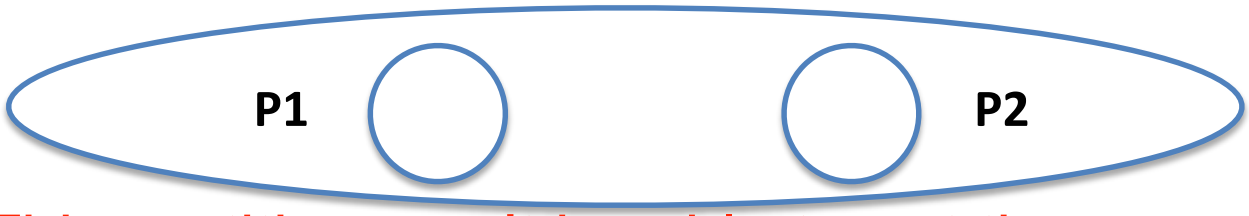
We can **not** achieve quorum!



No majority possible.

Network failure: Partitioning

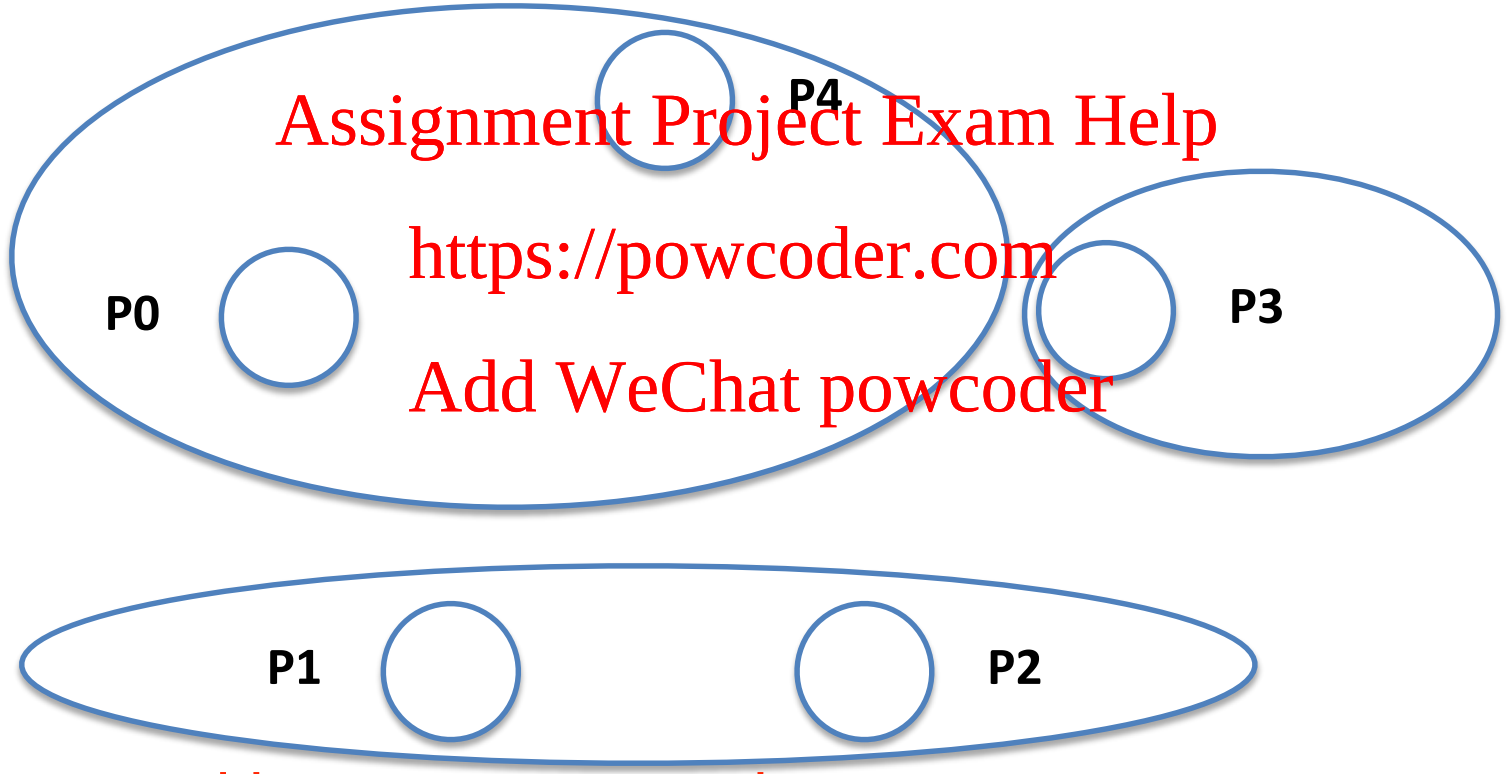
If sites are unreachable, quorum might still be possible.



This partition won't be able to get the resource.

Network failure: Partitioning

If sites are unreachable, quorum might still be possible.

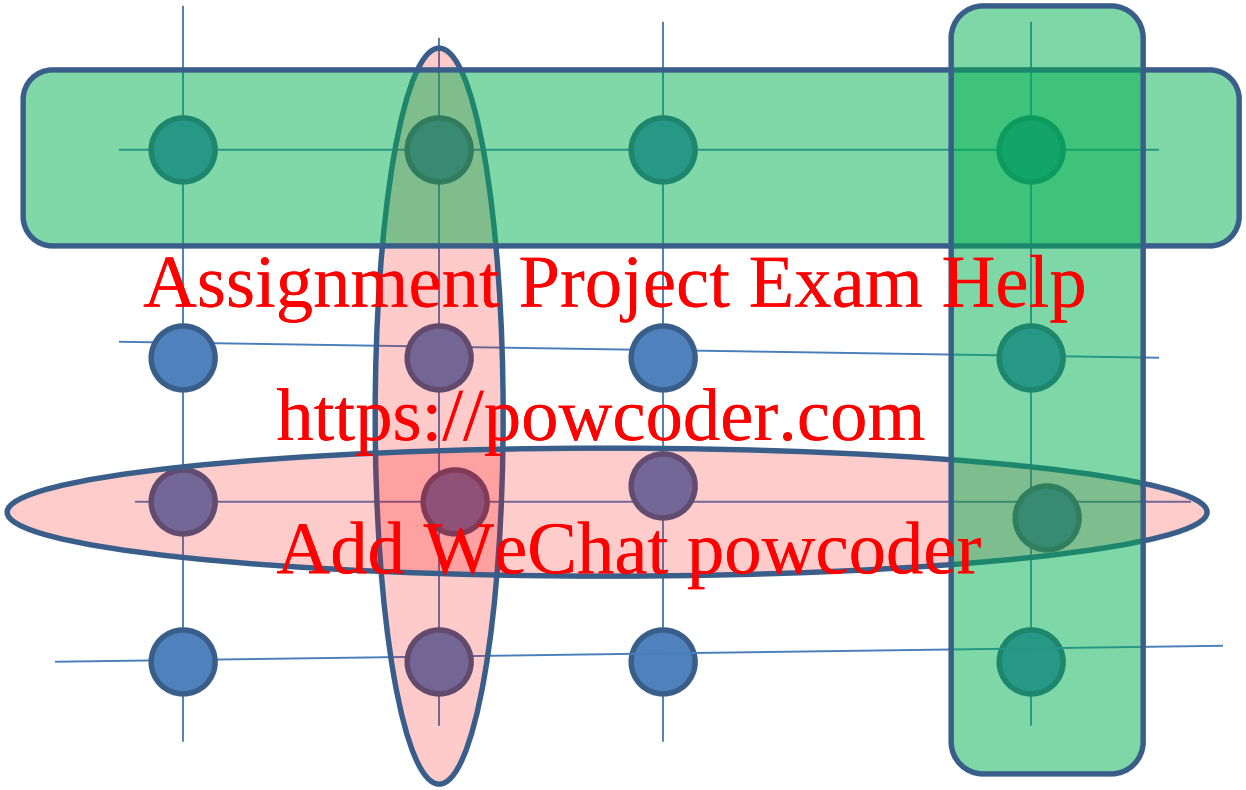


No partition can now get the resource.

Studying the Sizes of Quorums

- Can we have quorums of sizes LESS than majority?
- Yes:
  - Using logical structures

Quorum Sizes on Logical Grid Structure



Quorum size is $O(\sqrt{N})$