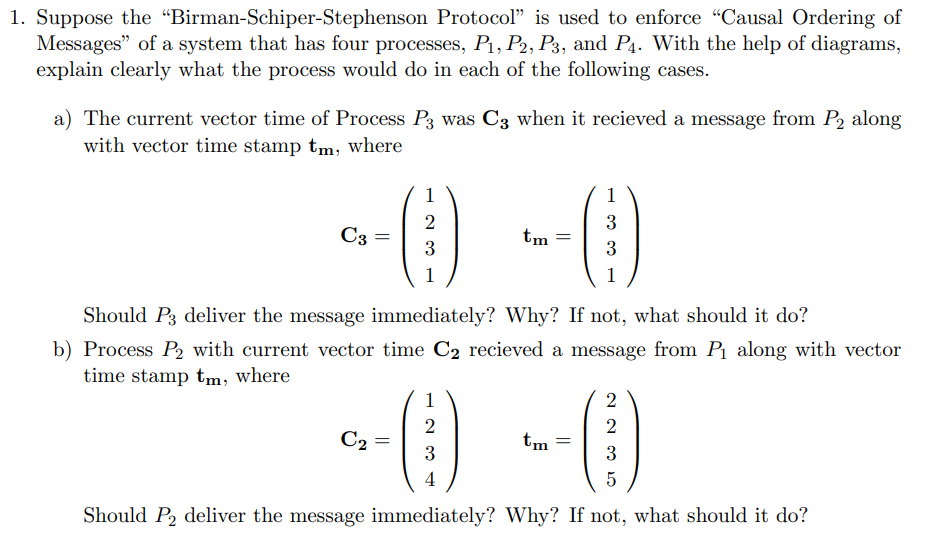
CSE 461 Homework 4

Ken Lin

Total points: Don’t know how many points this is, assuming the max is 50, and it’s 10 points each question.

1, (10 Points.)

(Indexing from 1 rather than 0, C2[1] refers to the 1st entry in C2.)

a, Should P3 deliver the message immediately? Why? If not, what should it do?

Yes. C3[i] = tm[i] for all i except 2, meaning every message P2 has received up to the point of when the message was sent was received by P3.

Additionally, C3[2] = tm[2] – 1, meaning that P3 received every message sent by P2 up to this point.

These two factors combined means it’s time to deliver the message.

b, Should P2 deliver the message immediately? Why? If not, what should it do?

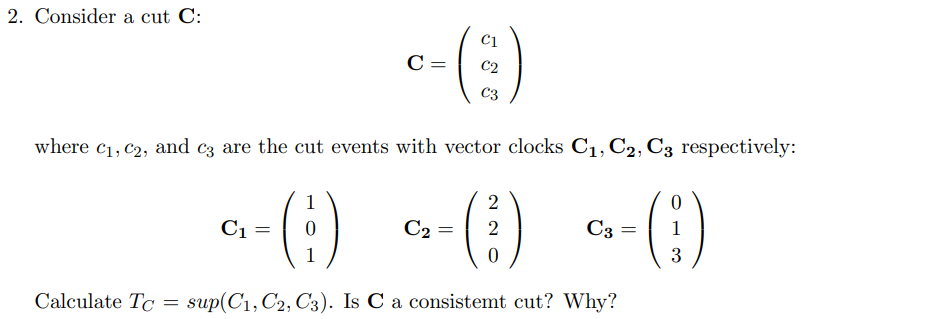
It should not.

C2[1] = tm[1] -1, meaning P2 has received every message from P1 before this, so this condition is satisfied.

But C2[4] != tm[4], specifically, C2[4] = tm[4] – 1, meaning at the time the message was sent, P1 received 1 message from P4 that, at the present time, P2 hasn’t yet seen.

Because of that, what P2 needs to do is to wait for the message from P4 to arrive before delivering this message. (That message might trigger other waits if P4 had seen messages at time of sending that P2 hadn’t.)

2, (10 Points)



No.

C2[1] > C1[1], meaning C2 received a message that C1 hasn’t sent it.

More formally:

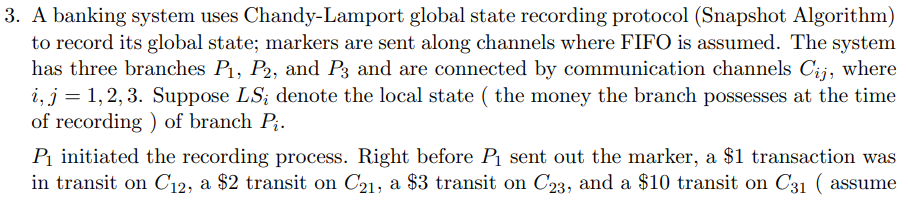
T = sup(C1, C2, C3) = {2, 2, 3}.

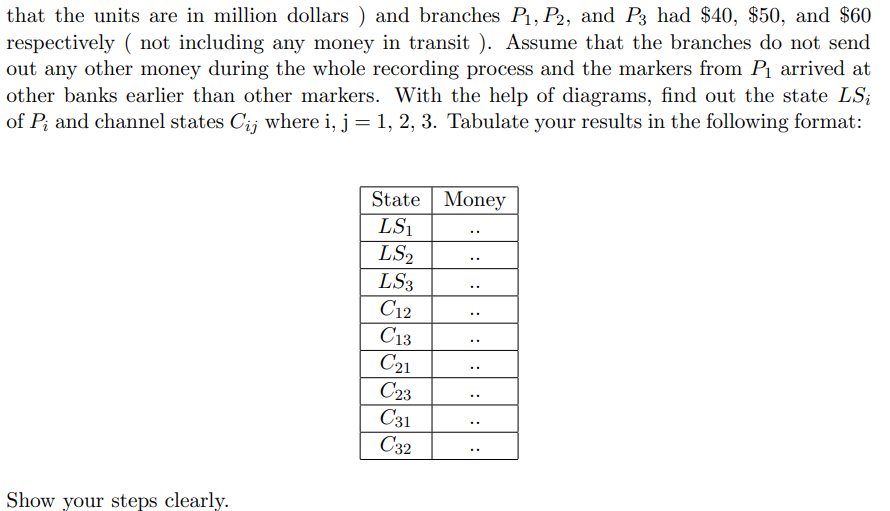
But T[1] > C1[1]. So the cut isn’t consistent.

Sup(C1, C2, C3) needed to equal {C1[1], C2[2], C3[3]}, but that equals {1, 2, 3} not {2, 2, 3}.

C[2][1] > C[1][1], meaning C2 happened after C1.

3, (10 Points assumed)



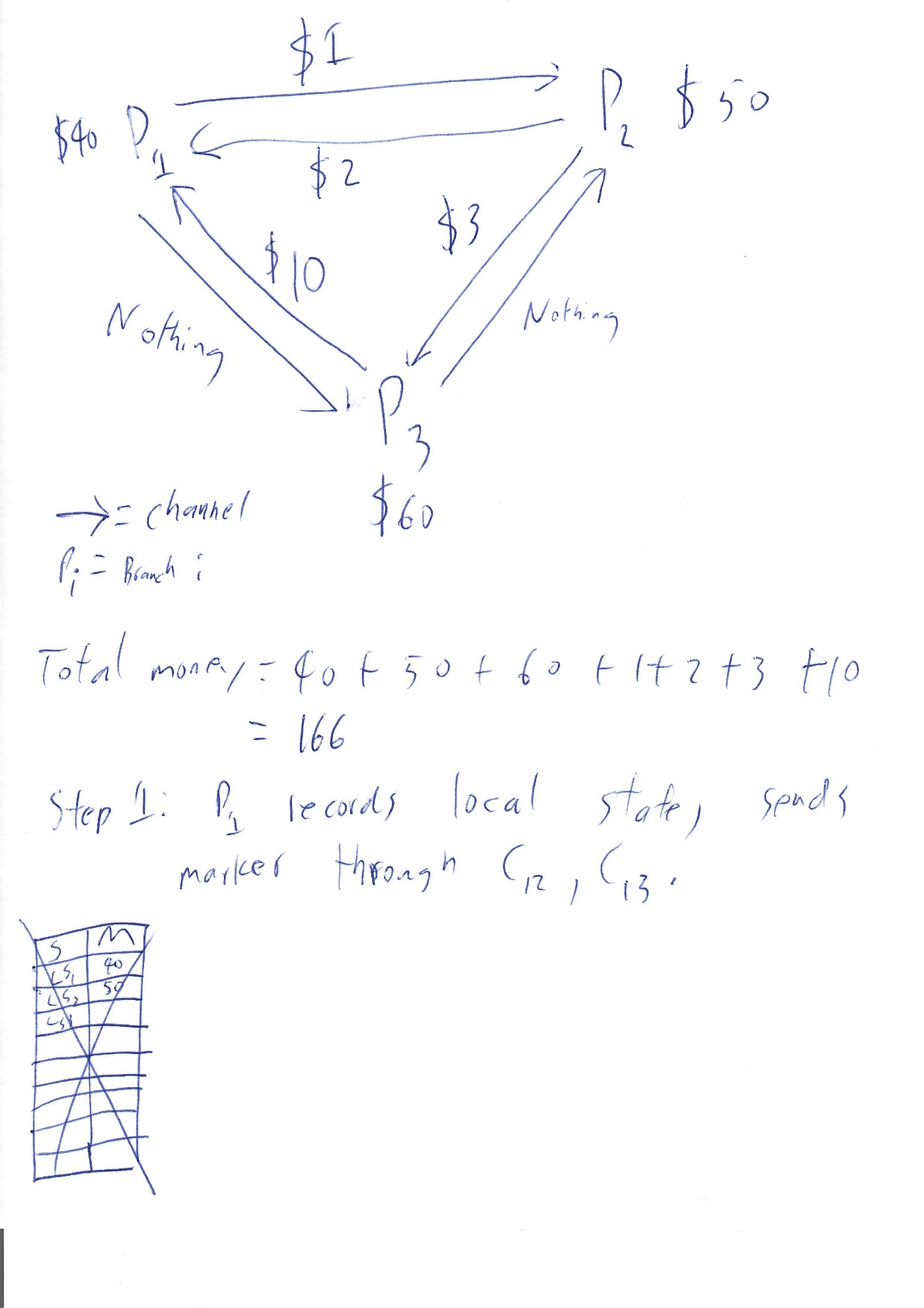


Assuming C12 denotes the channel P1 uses to send messages to P2. Also assuming that a $1 transaction in transit on C12 means that $1 was already deducted from P1 but not yet added to P2.

That is, the total money in the system = P1 + P2 + P3 + Money in transit.

Also assuming that markers will be treated as regular transactions in terms of how they move in a communication channel. (That is, a marker sent after a transaction in the same channel will arrive after that transaction. E.x. the $1 in C12 will arrive before the initial marker P1 sends to P2.)

Additionally, assuming that processes do not keep track of messages sent before they started recording state. (I.e. the branches don’t know of the money currently in transit.) Also, markers being received are not saved in local states. (That is, local states don’t treat markers received as messages received, so they don’t record them as something in a channel.)



Step 1 state:

|  |  |
| --- | --- |
| State | Money & Marker(s) |
| LS1 | 40 |
| LS2 | Not Yet Recorded (50) |
| LS3 | Not Yet Recorded (60) |
| C12 | 1, Marker |
| C13 | None (0), Marker |
| C21 | 2 |
| C23 | 3 |
| C31 | 10 |
| C32 | None (0) |

At step 1, C12 contains $1 and a marker from P1.

C13 contains a marker from P1.

C21 contains $2.

C23 contains $3.

C31 contains $10.

C32 contains nothing.

The P1 marker contains the fact that it has $40. (But not the fact that it sent $1 on the channel C12.)

Step 2 State:

We will process things in terms of markers. That is, we will assume money in transit stays in transit until a marker is sent in the channel. Because of FIFO, the marker will need to wait for the money to be processed before being processed itself.

The marker from P1 arrives. C13 was empty, so P3 receives only the marker. C12 on the other hand contained $1, so P2 receives $1 before receiving the token.

Consequently, when P3 records its own state it recorded itself has having $60, and C13 as empty. It then sent out markers containing this fact through C31 and C32.

However, when P2 recorded its own state it recorded itself as having $51, and C12 as empty. (C12 is treated as empty because the money from C12 arrived before the marker.)

After this is done, P2 sends its marker ($51, C12 & C32 empty) through C21 and C23.

P3 sends its marker ($60, C13 and C23 empty) through C31 and C32.

|  |  |
| --- | --- |
| State | Money & Marker(s) |
| LS1 | 40 |
| LS2 | 51, C12 Empty |
| LS3 | 60, C13 Empty |
| C12 | None (0) |
| C13 | None (0) |
| C21 | 2, LS2 Marker |
| C23 | 3, LS2 Marker |
| C31 | 10, LS3 Marker |
| C32 | None (0), LS3 Marker |

Step 3 State:

At P1, the money from P2 and P3 arrives at P1 before the markers, then the markers arrive.

At P2, the marker from P3 arrives.

At P3, the money from P2 arrives before the marker from P2.

Consequently, P1 records C21 as containing $2, and C31 as containing $10. LS1 is now: $40, C21 $2, C31 $10. (That is, P1 has $40, C21 had $2 in transit, and C31 had $10 in transit.) P1’s cash isn’t updated because this is a snapshot.

P2 receives a marker from P3, LS2 had been recorded, but it didn’t see a message before seeing the marker, consequently C32 is recorded as empty. LS2 is now $51, C12 empty (known from before), C32 empty.

P3 receives the money ($3) from C23, then the marker from P2. Because LS3 had been recorded, LS3 records C23 as containing $3. LS3 is now $60, C13 empty, C23 $3.

State after all markers have been received.

|  |  |
| --- | --- |
| State | Money & Marker(s) |
| LS1 | 40, C21 $2, C31 $10 |
| LS2 | 51, C12 empty, C32 empty |
| LS3 | 60, C13 empty, C23 $3 |
| C12 | None (0) |
| C13 | None (0) |
| C21 | None (0) |
| C23 | None (0) |
| C31 | None (0) |
| C32 | None (0) |

Final state after assembling all local states. (LS1, LS2, and LS3 are combined to form this.)

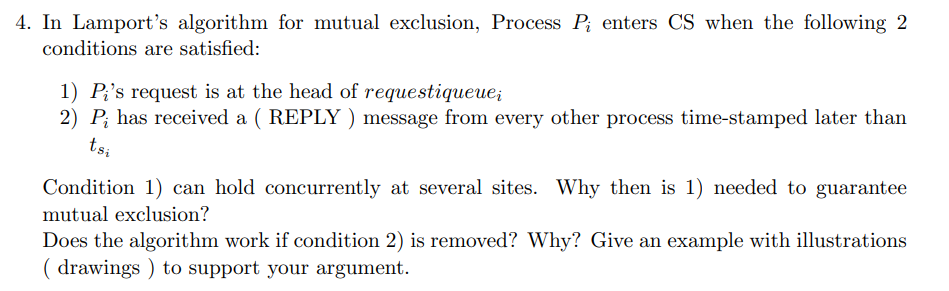
|  |  |
| --- | --- |
| State | Money |
| LS1 | 40 |
| LS2 | 51 |
| LS3 | 60 |
| C12 | None (0) |
| C13 | None (0) |
| C21 | 2 |
| C23 | 3 |
| C31 | 10 |
| C32 | None (0) |

^Above is the answer to this problem.

The snapshot shows $40 at P1, $51 at P2, $60 at P3, C21 as having had $2 in transit, C23 as having had $3 in transit, and C31 as having had $10 in transit.

The other channels are empty.

4, (10 points assumed)



Condition 1) can hold concurrent at several sites. Why then is 1) needed to guarantee mutual exclusion?

Condition 1 is necessary, but not sufficient, for mutual exclusion. That is, if Condition 1 is false then a process very definitely *should not* enter the critical section, because another process is very likely to be inside the critical section if a different process is at the head of a queue. There will be instants of time after a process leaves the critical section but before its RELEASE message arrives at other processes where the process at the head of queue isn’t inside the CS, but to guarantee only 1 process is inside the critical section at a time every process needs to assume the process at the head of the queue is inside the critical section. Consequently, processes not at the heads of their own queues must not enter the critical section.

That is, only those processes at the head of their queues should be inside critical sections, but not all processes at head of queues should be inside critical sections. However, if a process is not at the head of its own queue, then it definitely *shouldn’t* be inside a critical section.

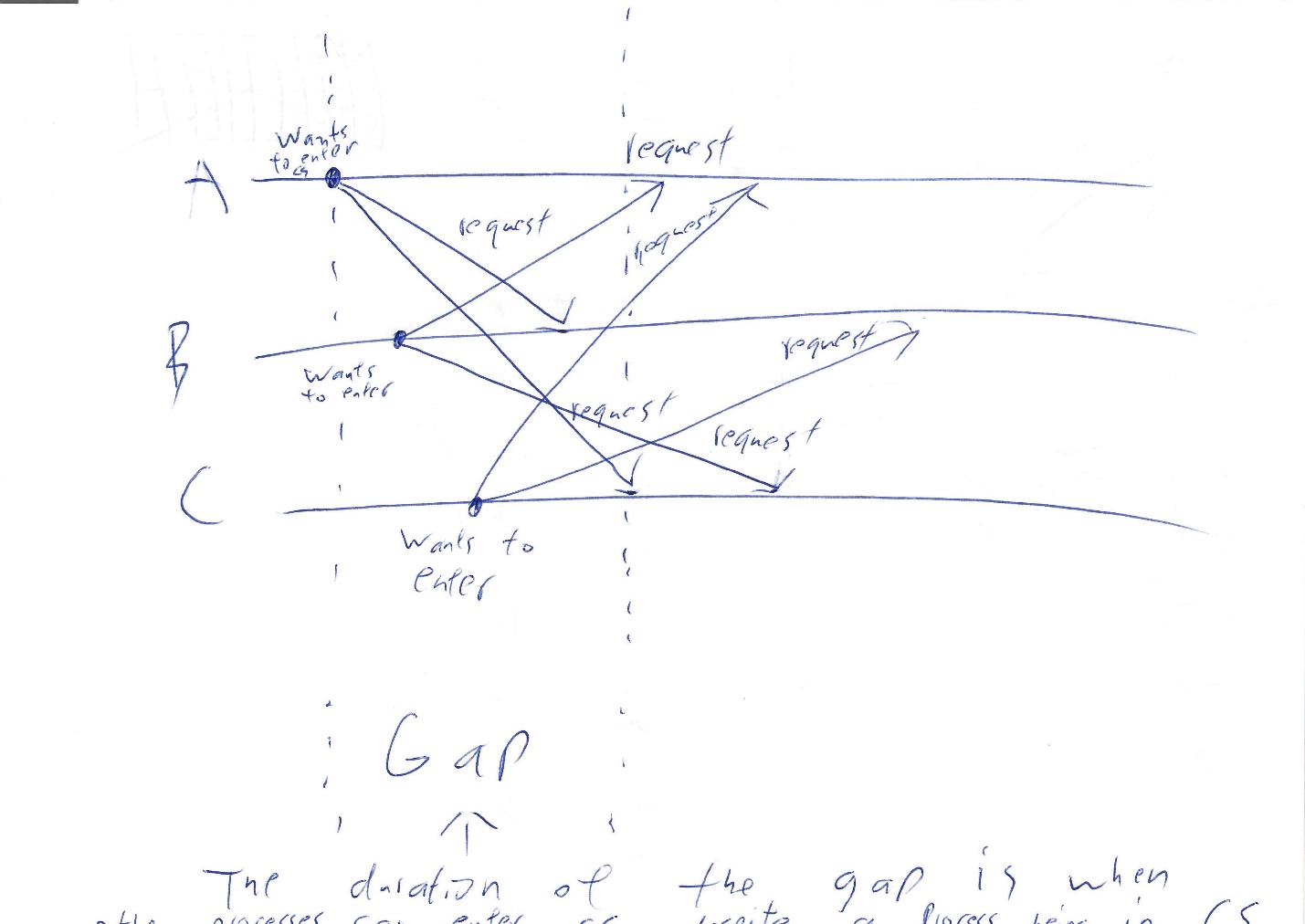
Hence the requirement.

Does the algorithm work if condition 2) is removed? Why? Give an example with illustrations to support your argument.

No.

As stated above, multiple processes can be at the head of their queues at the same time, consequently, if condition 2 is removed, multiple processes might enter the critical section.

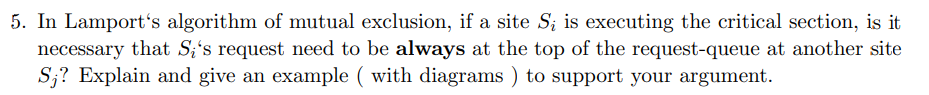
This is because of the delay during the time a process sends out its request to enter the critical section and when its received by other processes. Consequently, if multiple processes sends out the request at similar times then they might all be at the head of their own request queues. Without acknowledgement from other processes there’s no way to know if a process slipped through the ‘gap’ between when it sends out its own request to enter the critical section and when the request from another process that wishes to enter the critical section arrives.



As pictured, the gap is the duration of time where another process can enter the critical section despite a process already being inside the critical section. To avoid this gap, condition 2) is necessary.

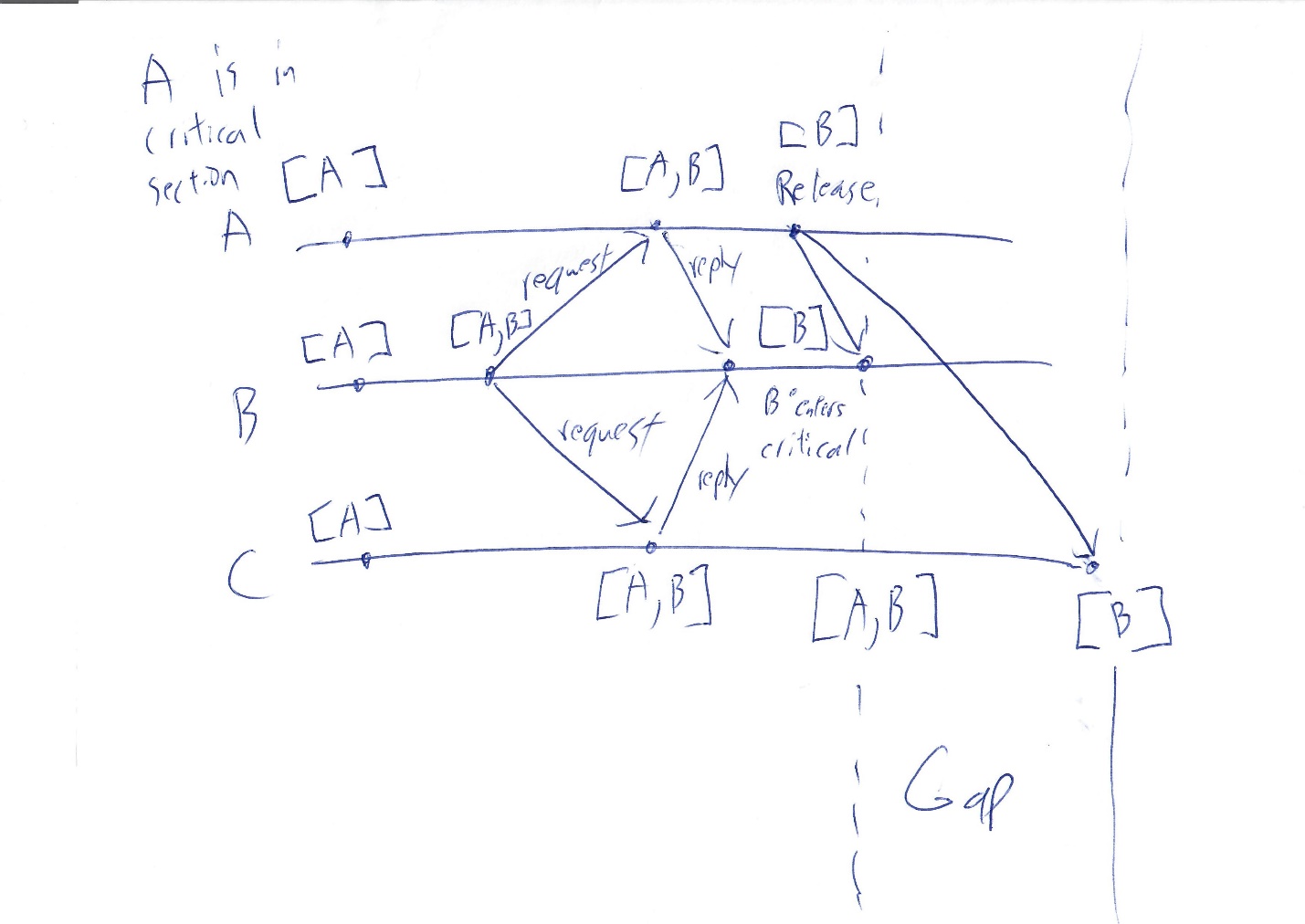
Note how despite the fact that A entered the critical section and sent out messages saying so, B and C would also be able to enter the critical section if rule 2) isn’t applied.

This is assuming the request queue is empty for all three processes at the start of the diagram.

5, 

It’s not necessary. This is because of the delay in the RELEASE message being sent out and the RELEASE message arriving at other processes.

Specifically, the delay isn’t constant, so one process might receive the RELEASE message, and update its own request queue, before other processes.



As you can see, because of the difference in time for the RELEASE message to arrive at process B and process C, there is a gap in time where B is permitted to enter the critical section (received replies from all processes, head of its own queue), yet in process C, the request queue contains a different process at the head of the queue.