ANSWER 1

The outcome of a coin flip can be 0 or 1. Each process has the ability of flipping a coin. A process will get assigned an ID when the outcome of its coin toss is 1. We have to make sure that all IDs are unique and that all processes received such an ID. The procedure is as follows: processes start tossing their coins. The overall outcome can contain any sequence of 0's and 1's. Processes keep tossing the coin and eventually, only one process will have the outcome equal to 1. This process will pick its ID and then broadcast it to all other N-1 nodes. The remaining nodes will proceed similarly, and the next ID to be assigned will increment the current ID value (by 1, let's say). When the process terminates, all nodes will have unique IDs assigned to them.

One can use any token circulating algorithm, also, to name the processes. The idea is that only the process holding the token can get an ID.

ANSWER 2

The idea is to transfer one element of the array at a time, from P to Q via move. The move process has to synchronize with both its sender (P) and its receiver (Q) - because move has to be able to accept a message when P is ready to send, and conversely, Q has to be able to accept the message when move is ready to send. Also we want to make sure that Q receives exactly the value that P sent. Also Q doesn’t receive before P sends.

There is no circular waiting (no circular send-receive dependency) among the processes, so no deadlock.

Process P:

```plaintext
for(i=1; i<=n; i++)
    checks if move is ready to accept value
    move!x[i];
```

Process move:

```plaintext
for(i=1; i<=n; i++)
{
    listens for P
    P?var;
    x[i] = var;
    checks if Q is ready to accept value
    Q!x[i];
}
```
Process Q:

```c
for(i=1; i<=n; i++)  {
   listens for move
   move?y[i];
}
```

**ANSWER 3**

We can consider that the inter-process communication model is a combination of the message-passing model and the knowledge-based model. The second model is needed because there exists an a priori understanding between Alice and Bob (if you don't call, that means you are okay). So no exchange of messages can still lead to a conclusion. Messages are sent instantaneously.

**ANSWER 4**

One idea is to have an array as shared variable, of size equal to the number of processes. Each cell corresponds to one process and the value of each cell represents the current phase of that specific process. Process \( i \) will update its cell to \( k \) after it has completed phase \( k \), and this assignment is done only after \( i \) checks if all the other elements in the array are at least \( k \), and \( i \) can begin phase \( k+1 \). If there is at least one process whose array value is less than \( k \), then process \( i \) writes \( k \) in its corresponding cell, and waits until it can move forward to the next phase. Note that the difference between any two values in the array is at most 1. Also note that checking the values of the remaining processes can be done several times. No process begins phase \( k+1 \) until all processes completed phase \( k \).

Another way is to have a shared variable that represents the number of processes currently in the same phase. Initially, the value of the variable is equal to the number of processes. Each time a process finishes a phase, it decrements this variable. A process can move to the next phase, only when the variable is zero. The first process to encounter the variable equal to zero, it resets it to the number of processes, and the procedure continues.

**ANSWER 5**

Model properties:
- message-passing
- bounded-delay channel
- asynchronous

The idea is to propagate the broadcast. First, \( P \) checks if any of its immediate neighbors is \( Q \) (\( P \) broadcasts a query asking for \( Q \) to all its neighbors.) The next step is that each node other than \( Q \) that has received a query from \( P \) regarding \( Q \), it will forward the query
further to its own neighbors. When node $Q$ receives such a query, it will respond back to $P$. $P$ listens for replies regarding $Q$ for a certain amount of time. $P$ can terminate the process when the first answer regarding $Q$ arrives, or it can accept all the incoming answers until it times out and keeps the answer with the shortest network delay received. It is possible that no answer comes back, even if there exists a path between $P$ and $Q$, and they can theoretically communicate.