1-

1. Due to many devices on the network communication between them become difficult without some sort of agreed standards. (Think each machine speaks its own language, if we don’t have a unified dictionary for all of them they won’t understand each other)
2. Because there is latency over the network, to transfer data from one point to another speed is determined by speed of light, type of materials used to transfer the data, and computers at each node, hence we can’t ignore the latency, else all sort of evil things happen.
3. Publish subscriber model consist of two main components:  
   Publishers: push their data under certain tags to shared data space.  
   Subscriber: subscribe to tags and when those tags receive an update it get pulled to the subscriber.
4. Provide an abstraction that allow communication between softwares that don’t share the language or even the physical space. (IDL stands for interface description language, think about what you java implements interface, and how you have to fill in blanks)
5. Copying the content that pointer is pointing to, to an array of characters in a stub. It is used to allow sending parameter from a client to server and vice versa in a RPC and replace call by reference.
6. Just send a message and forget about it.
7. Information are sent but failed to be received due to loss of the packets (network is unreliable), information are sent and received but content are changed from what the general requested (network is not secure).
8. Given a sufficiently long period of time over which no changes are sent, all updates are expected to propagate and eventually all replicas will be consistent.
9. It’s how a server makes a remote object available to clients, by bounding a string to a remote object the client can interrogate the registry using the string.
10. Send many requests that I don’t care about using as many threads and computer as I can without any delay. Forcing the server to deal with my meaningless requests and denying the computational resources from other legitimate requests.

2-

1. To allow communication between the client and server, by replacing the call by reference with stubs we can send parameters from client to server. And send back the result in a way both system understand without caring if for example one system use big endian or little endian.
2. Clock used by distributed system to provide ordering on events executed by different processes. They increase every time a component perform an action.  
   Logical clock provide a partial ordering, every time a component perform an action it increment its logical clock and if an action is affecting another process we transmit the logical time and set the logical time of the recipient to max between current logical and received, then increment the recipient time by 1 (since it received a message).  
   vector clocks provide total ordering. By storing the each component clock locally in a vector, and clocks are updated separately for every process, when a process send something to another, we update the relevant vector index.   
     
   Logical clock can infer if event e1 -> e2, that L(e1) < L(e2), however can’t say if L(e1) < L(e2) that event e1 -> e2.  
   With vector clocks you can.
3. ACID:  
   Atomic: ALl or nothing, either all operations execute correctly or none happen.  
   Consistent: All commits leave the system in a consistent state respecting the system integrity.   
   Isolated: components don’t interfere with one another.  
   Durable: Updates are persistent once they are committed
4. 1. We will require a coordinator which can be elected from a set of servers.   
      Then the coordinator will oversee the process of giving tokens to systems requesting access, those systems give the token back to the coordinator when they finished. When the coordinator is out of tokens it start queuing the incoming requests until it gets its token back.
   2. Ring-based election: Assuming all components are arranged in logical order that each component knows which one is next. And no failure in connection.  
      All components at the beginning of the algorithm aren’t participants.   
      Starting the process, the component change itself to participant and send its identifier to the next in the ring.  
      When a component receives an identifier, compares it with its own:  
      if the message’s is larger:  
       forward the message as it is.  
      Else if already participant:  
       Discards the message.   
      Else // must be larger   
       Forward the message with own identifier   
      And declare self as participant.  
      If it receive its own identifier, then the component is now elected, and sends a message to everyone in the ring again.   
      Totaling 3n-1 messages are sent.
   3. In ring-based election, worst case there are 3n-1 messages are sent (2n if process is the one prior to the one that started it, and then n-1 messages to declare leadership).  
      In polling approach: If one of the component initialise it the number of messages sent at beginning is n-1, then n-1 messages received back from each component, that is 2(n-1), then informing each service of the decision is n, so 3(n-1).  
      The main advantage polling has is if a component fail it is easier to recover as long as it is not the coordinate nor the coordinate to be.

3-

1. When a client wants to initialise a RPC it first prepare the arguments and wrap it in a stub. (calling client stub procedure)   
   The stub act as an interface between client and server to allow communication between different systems. The client stub marshall the message into a serial form.  
   After that the client send the stub to the server, where the server unmarshall and arguments are used to call whatever procedure the client requested. After calculation are done result are also marshalled in a stub, and sent back to the client, the client call client stub to unmarshall the returned value and get the result.
2. Availability = 1 - p(failure)   
   = 1 - (0.5/12.5 \* 0.5/12.5)  
   = 0.9984  
   Meaning highly available
3. By using Amdahl’s law,   
   Speedup = 1 / 1-x, where x is what can be parallelized, in this question it is the 90 services.  
     
   = 1 / 1 - 0.9  
   = 1 / 0.1   
   = 10 speed up
4. Setting a bounded time on request isn’t feasible with only one computer.   
   This could be best explained in a situation where the server is receiving too many requests at any one time, the requests will have to be added to the in-queue and have to wait until the server done with other requests, the wait in the in-queue could take longer than any bounded time.   
   Usually this issue is solved by throwing more servers to balance the load, but if we are bound to one server one way to deal with it is having clever estimation of run time and reject requests that will take longer than the bounded time to execute.
5. 1. Z = 0 or 1.   
      Can be done with P1, P1, P2, P2, resulting in z = 1  
      Or P1 P2 P1 P2 resulting in z = 0
   2. Again z = 0 or 1 but also 2.   
      To get a 0 follow P1, P1, P2, P2   
      To get a 1.  
      P1, P1, P2, P2 (or P2, P2, P1, P1)  
      To get a 2:  
      P1, P1(don’t read P2(y)), P2, P2 ( don’t read P1(x))

Question 4-

1. No ACID means requests don’t respect database integrity, more than one request can change same state at the same time, this cause a lot of issues, one is having duplicated booking (the other is corrupting the whole database :))   
   This can be done by say 2 requests coming from different process (thread/computer)  
   For simplicity going to ignore msg id 96 from jerry (it is going to ge rejected anyway).  
     
   First process will dequeue msg id 90, second will dequeue msg id 86.   
   Both will start processing it, and attempt to reserve 540 and 542 respectively.   
   Now at the same time they both commit, because there is no ACID this will propagate just fine, the max\_res will be updated twice but never in time for the cehck to stop either process from booking. Resulting in 3 total booked slot for Tom.
2. At ‘Reserve slot for student’. A revert to the last checkpoint will get in progress, any item since the checkpoint that left the inqueue will be sent back to the inqueue, all messages in the outqueue will be removed, until the full database come back to work.
3. When the server need to ensure the message is sent and received.   
   For database, revert back to final stable state.   
   For middleware If its coordinator it can leave everything in inconsistent state, if it isn't it will have the same fate as database.
4. Msg per second = msg per hour / 3600 = 7200 / 3600 = 2 msg/second  
   2 threads, each process of 50 milliseconds = 0.05   
   Service time = 0.05/ 2 = 20 msg per thread   
   Arrival rate = 2msg/second   
   Service time is way lower than arrival rate thus there is no wait time
5. Hotel reservation might succeed but band didn’t, the check doesn’t check if only one fail, now since one of them failed it is going to go back in teh loop, if this happen twice, it will fail everytime after (because hotel reservation will reach the max causing no enough slots error)