Vikram Deshmukh
SJSU ID: 010720788

Problem 4. What is in your opinion the critical step in the development of a systematic approach to all-or-nothing atomicity? What does a systematic approach mean? What are the advantages of a systematic versus an ad hoc approach to atomicity? The support for atomicity affects the complexity of a system. Explain how the support for atomicity requires new functions or mechanisms and how these new functions increase the system complexity. At the same time, atomicity could simplify the description of a system; discuss how it accomplishes this task. Support for atomicity is critical for system features that lead to increased performance and functionality, such as virtual memory, processor virtualization, system calls, and user provided exception handlers. Analyze how atomicity is used in each one of these cases.

Answer: The critical step in the development of a systematic approach to all-or-nothing atomicity consists of ensuring
1) only one atomic transaction has access to a shared resources;
2) the system can be restored to its original state if the atomic transaction fails; and
3) the outcome of these atomic transactions yields a consistent result.

A systematic approach means an approach in which contingency plans are in place to recover from undesired outcome arising while executing the atomic transaction. For example, if an atomic transaction fails to complete then a systematic approach dictates that measures should be in place to rollback any changes made to the system by the failed transaction. This way, the system is always in a consistent state regardless of the outcome of the atomic transaction.

A systematic approach is better than an ad-hoc approach simply because it ensures the system does not get corrupted in case undesirable or unwanted events occur while executing the atomic transaction. The contingencies or provisions that are needed to be put in place to ensure complete rollback and mutually exclusive access to shared resources in order to support systematic approach to atomicity add an overhead on top of the existing system, thereby adding to the complexity of the system. Yet, at the same time, adopting a systematic approach frees the system of need to recover from inconsistent state thereby making it more fault-tolerant. Hence, in a way, the systematic approach also simplifies the description of the system.

Let us now review how atomicity is used in the following system features:

- *Virtual Memory:* When the Virtual Memory is preparing pages to fit into frames, it needs to map page addresses from the virtual address space to frame addresses in the physical address space. Once this mapping is done, it needs to move entire pages between the physical memory and the disk. These swapping actions need to be atomic in order to ensure the data being processed in the Virtual Memory does not get corrupted due to one of the Virtual pages being in an inconsistent state. This is done by ensuring that no disk IO operation is permitted while pages and moved between the physical memory and the disk.

Vikram Deshmukh
SJSU ID: 010720788

- *Processor Virtualization:* Processor Virtualization is the process allocating some of the actual CPUs on the user's machine to dynamically created Virtual Machines. These VMs then perform as if the CPUs are actual dedicated CPU of their own system. During Processor Virtualization, it is extremely important that the processes of allocating and deallocating CPUs to the Virtual Machines are atomic in nature. If these processes are not atomic then the system may end up believing that one of the CPUs is still allocated to a VMs while in reality the VM using that CPU ceases to exist. This will lead to the CPU ending up not being utilized and getting ignored by the system.

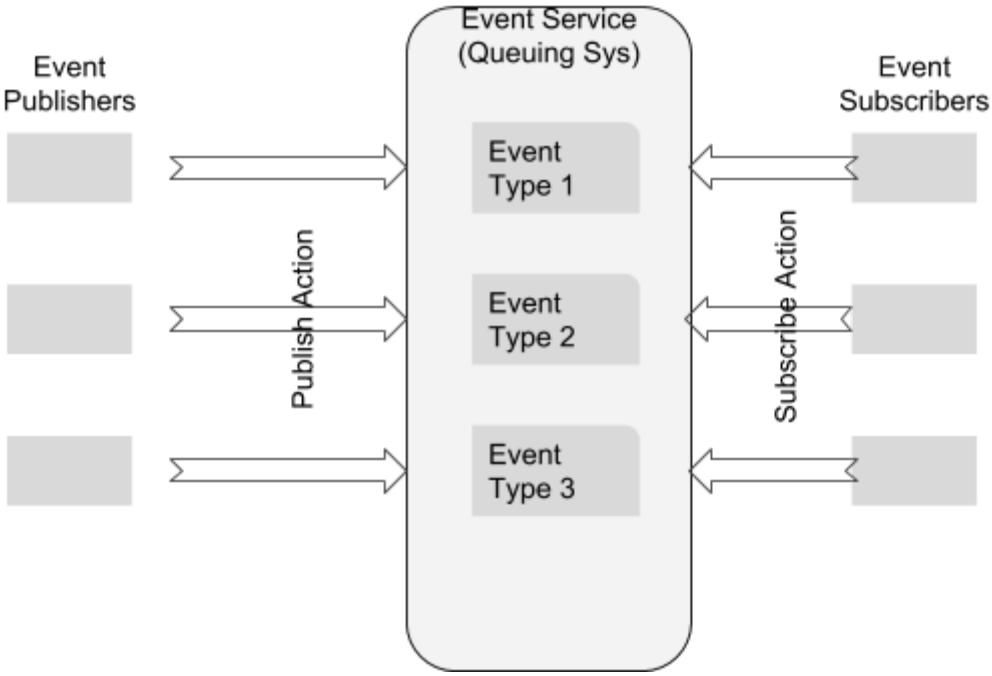
- *System Calls:* or Kernel Calls are calls made via software interrupts to request the kernel to perform operations like accessing the hard disk drive, creating and executing new processes, etc. These are critical, hardware level operation that have major impact on the system as a whole. Hence it needs to be ensured that these actions are atomic in nature. This is done using mutual exclusion and locking mechanism. For example, at any point in time, the system will allow only one application to access a particular portion of the hard drive.

- *User provided Exception Handlers:* In computer applications, provisions are made by the programmer to handle exceptions by means of user-provided exception handling (commonly called "trapping"). This is usually done when the software comes across any unusual, un-expected scenario. In such a case, the execution of code is abruptly stopped and steps are rolled back to the last consistent state. This shows the atomic nature of user provided exception handlers.

Vikram Deshmukh
SJSU ID: 010720788

Problem 6. Explain briefly how the publish/subscribe paradigm works and discuss its application to services such as bulletin boards, mailing lists, and so on. Outline the design of an event service based on this paradigm, as in Figure 2.21(b). Can you identify a cloud service that emulates an event service?

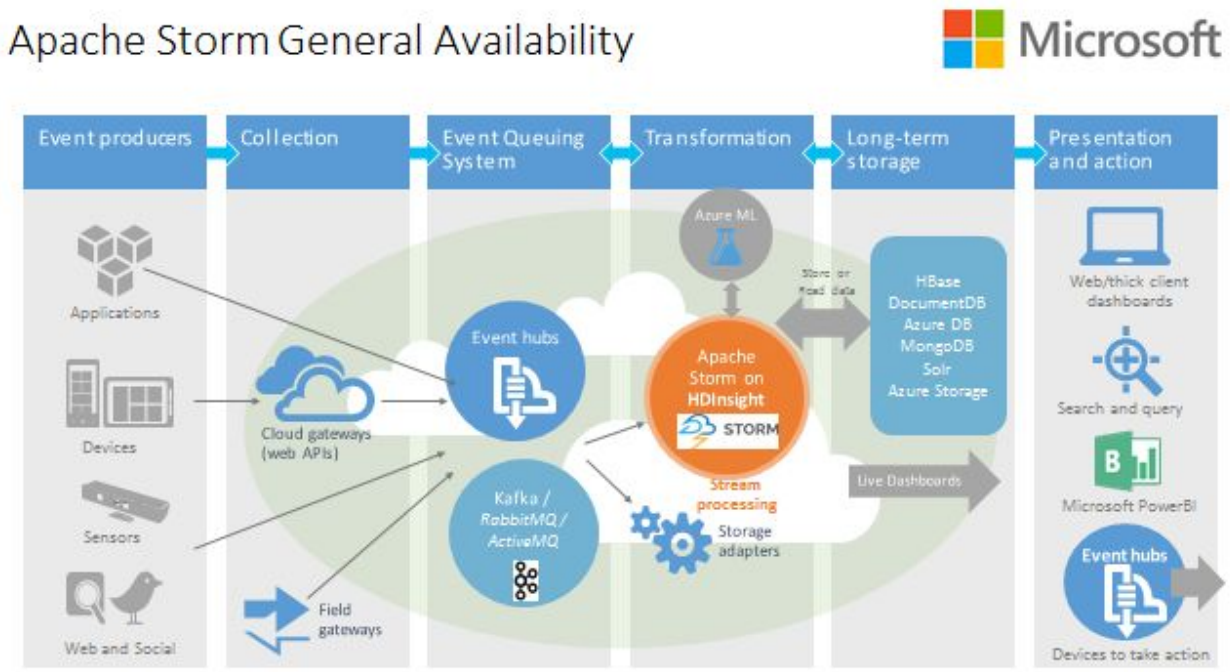
Answer: In an event-based system communication between various components happens by means of events. There is a set of Event Producer components that are tasked with triggering or publishing events, and there are other components that are listening to or are subscribed to these events. Then there is the Event Service itself. It is responsible for queuing all the events being raised in the system and notifying all subscribers of a particular event when that event is triggered. Given below is the outline of the publish/subscribe paradigm.



This same mechanism is followed in bulletin boards and mailing lists as well. In case of the Bulletin board, the board itself acts as an Event Service. The subscribers are entities interested in reading about the events on the bulletin board and the publishers are the event producers of the system. Similarly, in case of mailing lists, the mailing list itself acts as an Event Service. The composers send out an email to the mailing list and not to individual subscribers on the mailing list. The mailing list then broadcasts the email to all ‘subscribers’ of the mailing list.

Azure Event Hub is the best example of a cloud service that emulates an event service. As we can see from the diagram below. Azure Event Hub also has Event Producers, Event Consumer (Presentation and action layer) and Event Service itself (Event Queuing System).

Vikram Deshmukh
SJSU ID: 010720788



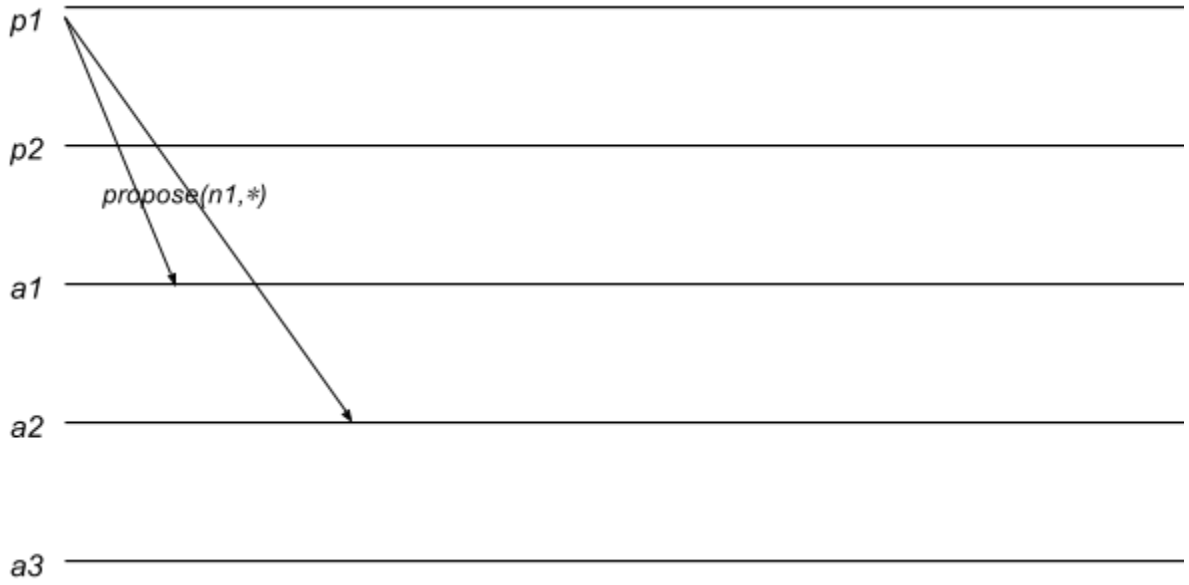
Vikram Deshmukh
SJSU ID: 010720788

Problem. Describe a real-world distributed system that could use Paxos algorithm for solving consensus problem. (Search the web you could find examples of a banking system, MapReduce, and Google Big Table.) Your example should include at least two proposers and 3 acceptors.

Answer: Paxos is widely used in many real world distributed system to solve consensus problems. Paxos ensure that all tasks get executed in a linear fashion. Consider a banking application for example. The points of access may be distributed across geographies. At the bank a teller might wish to process the check issued by the account holder, the account holder may be trying to withdraw some cash from the ATM machine, while the monthly auto-payment set up by the account holder tries to auto-deduct funds from the account to pay the account holder's phone bill. If any of these applications are executed without taking into account the other two actions then it will leave the distributed system in an inconsistent state.

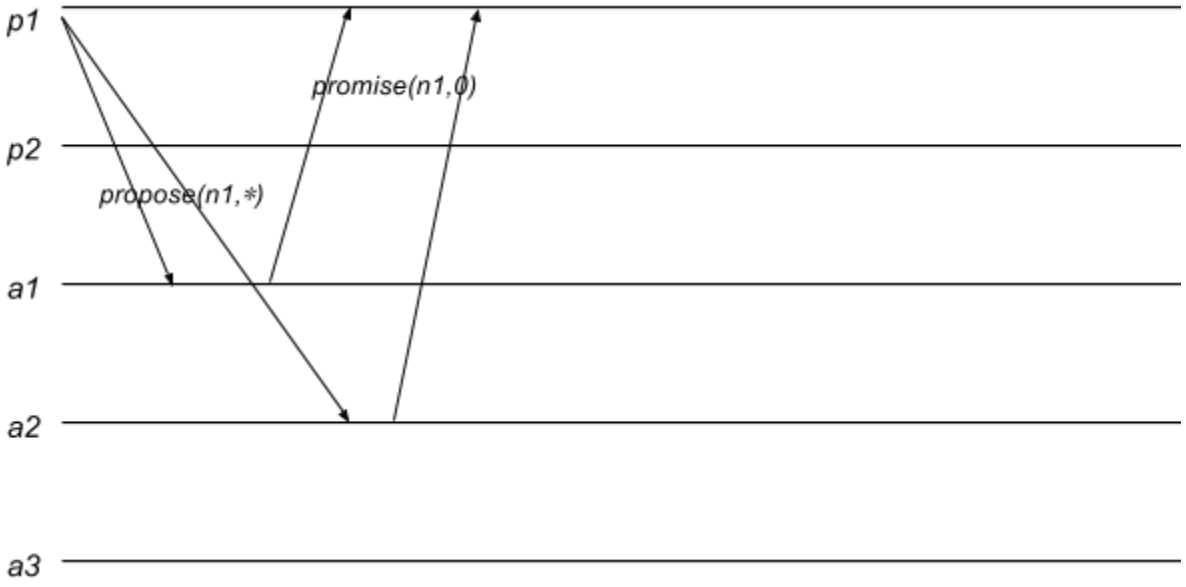
Let us see how Paxos can solve this problem for us. Suppose there are 2 proposers p1 and p2 and three acceptors a1, a2, a3. Then,

1. p1 proposes propose(n1,∗) to a1 and a2.

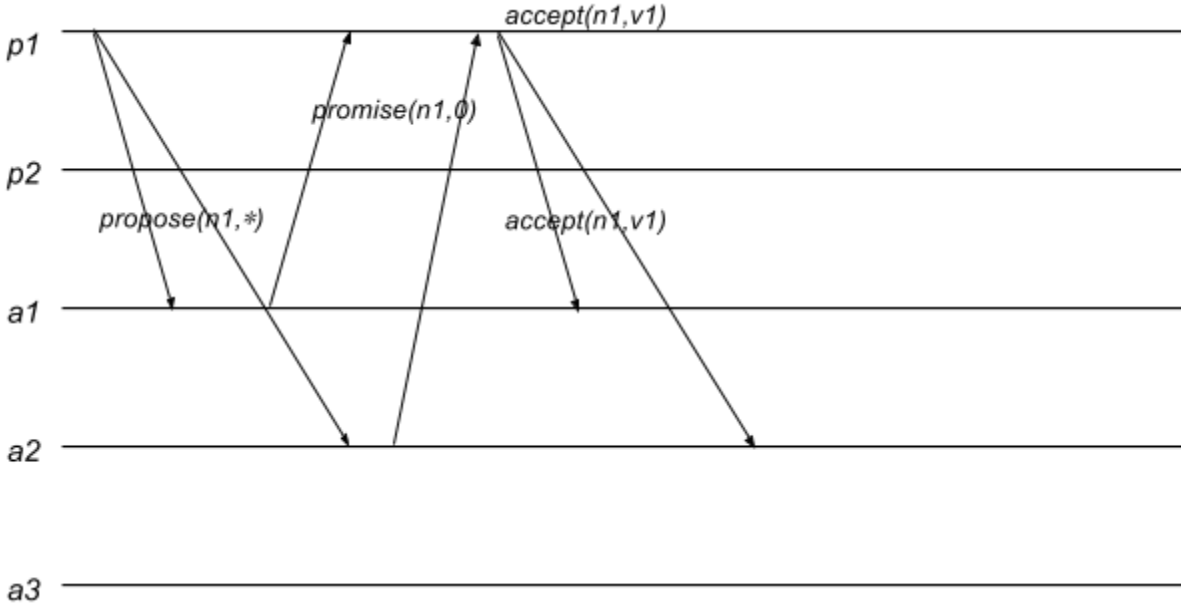


Vikram Deshmukh
SJSU ID: 010720788

2. both a1 and a2 respond to p1 with (a) a promise never again to accept a proposal numbered less than n1 and (b) no proposal because they have not accepted any ones.

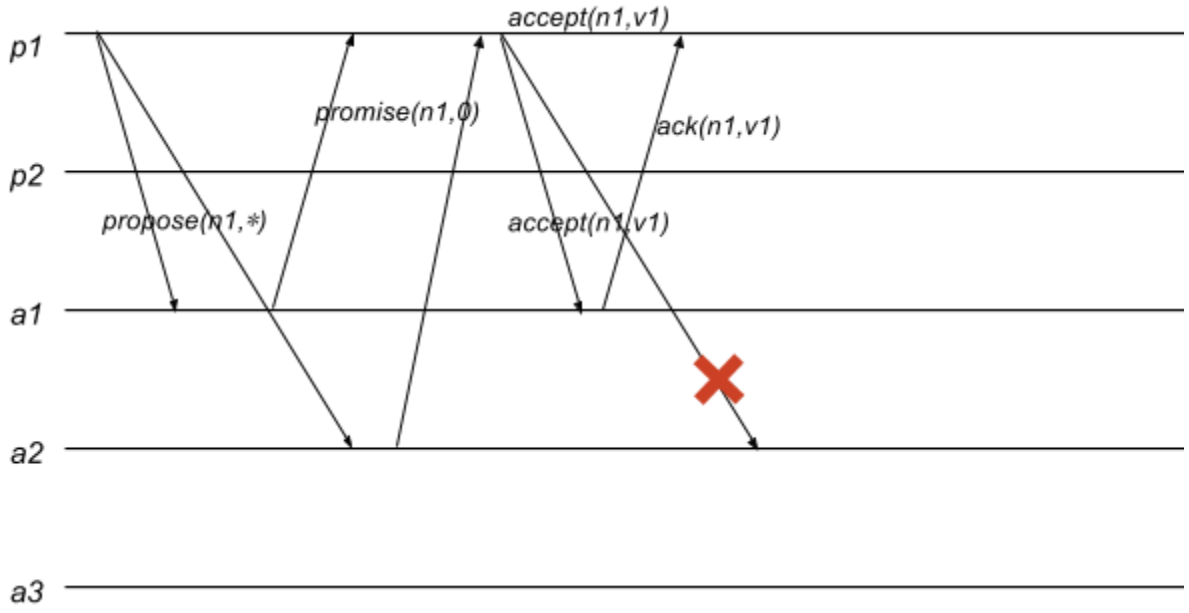


3. p1 receives the two responses from a1 and a2. Now p1 is free to select its own value v1. p1 sends accept(n1,v1) to a1 and a2.

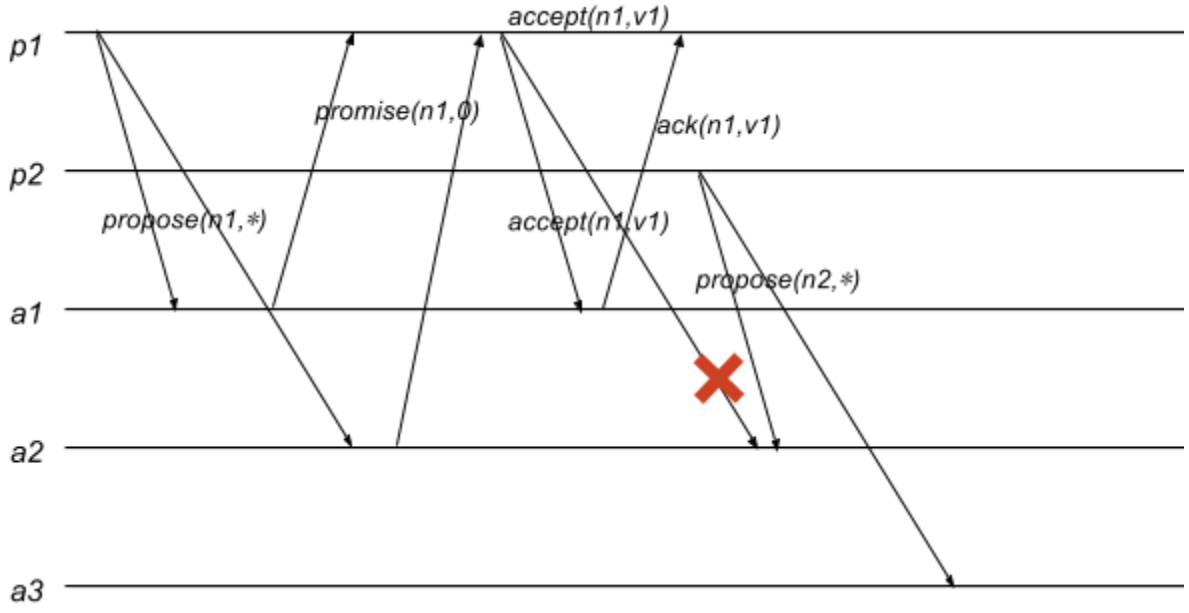


Vikram Deshmukh
SJSU ID: 010720788

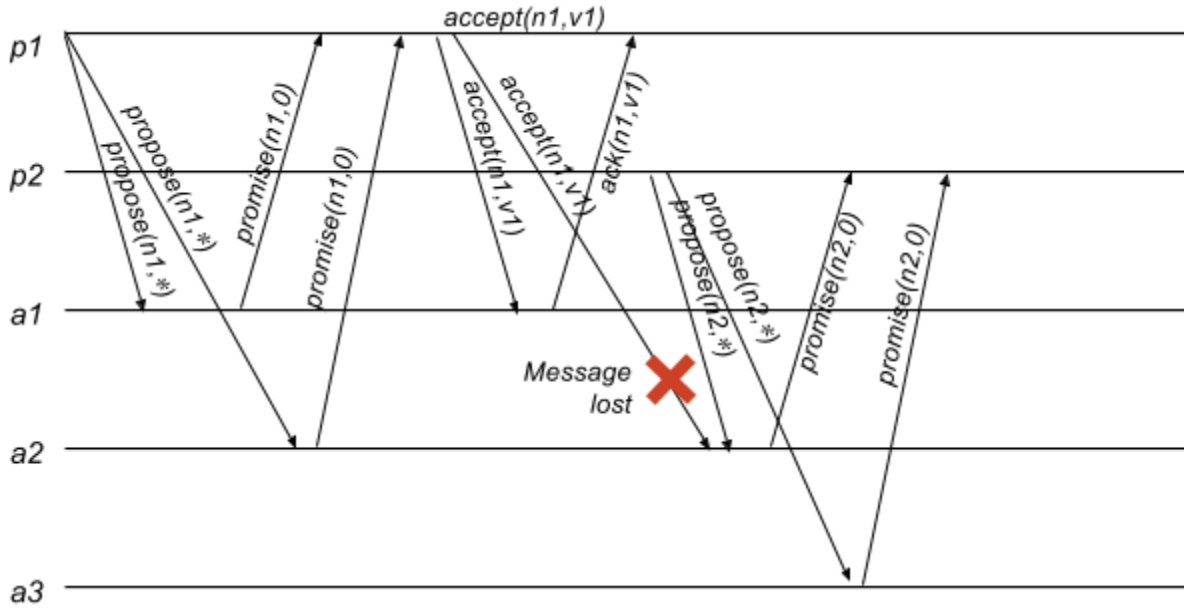
4. a1 accepts (n1,v1) but a2 does not (maybe this message to it has been lost; note that an acceptor can ignore any request without compromising safety).



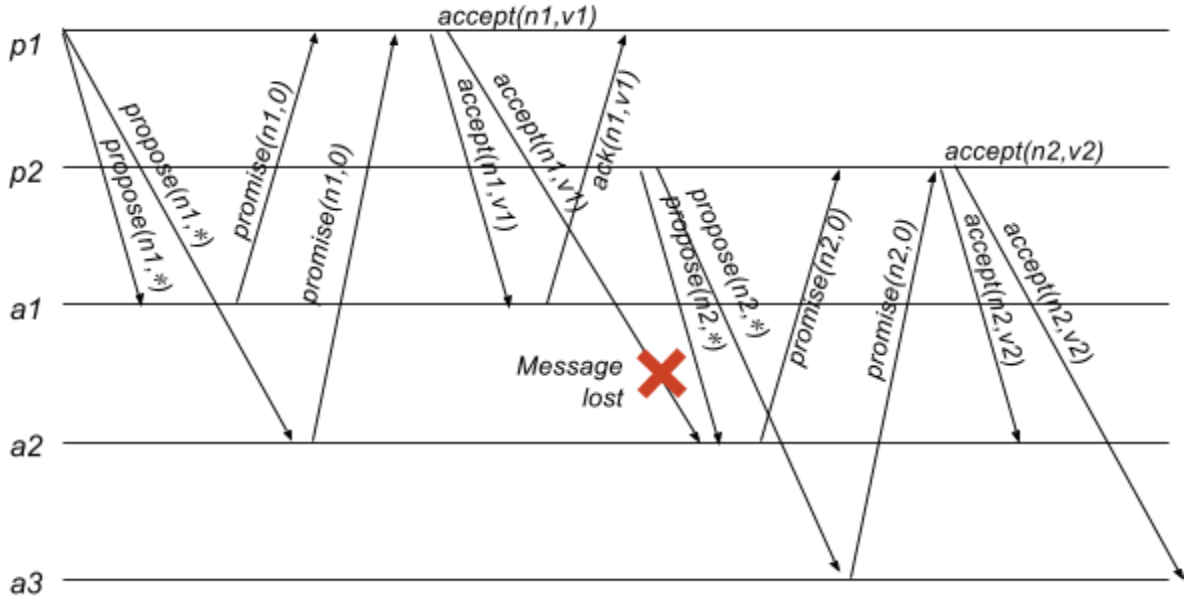
5. p2 proposes propose(n2,*) to a2 and a3 (n2>n1).



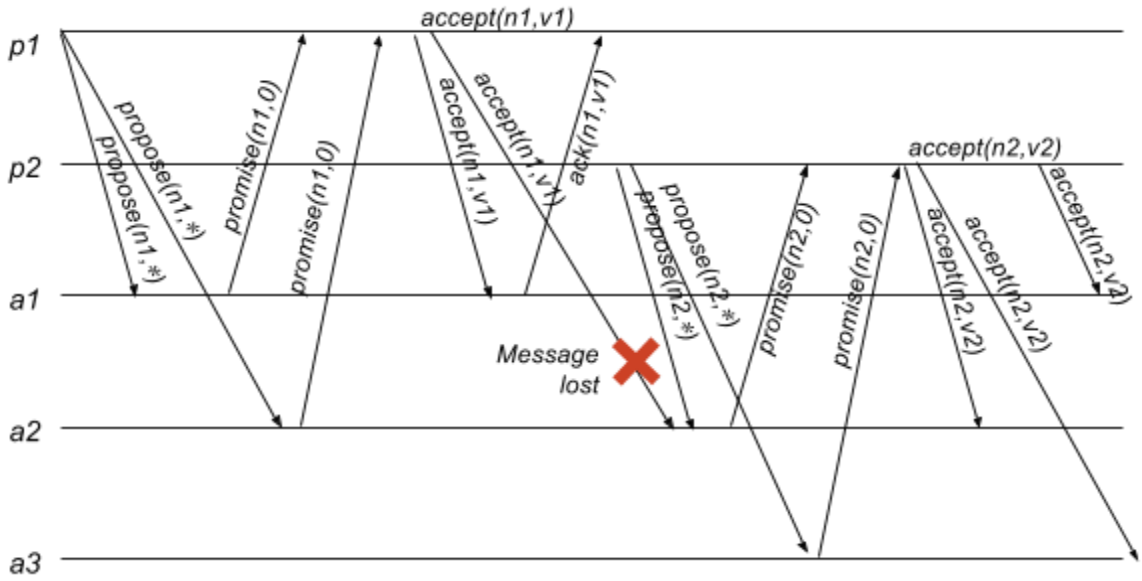
6. Because, in step 4, a2 does not accept accept(n1,v1) from p1, both a2 and a3 respond to p2 with a promise never again to accept a proposal numbered less than n2.



7. p2 receives the two responses from a2 and a3 and is also free to select its own value v2. p2 sends accept(n2,v2) to a1 and a3.



8. a1 can now accept the proposal (n2,v2) from p2.



In this way, by the end of the algorithm all acceptors come to a consensus of agreeing upon a single value.

References:

1. Azure Event Hub (https://azure.microsoft.com/en-us/blog/microsoft-continues-vision-for-broad-big-data-adoption-while-showing-more-love-for-linux/)
2. Paxos example with 2 proposers and 3 acceptors (http://cs.stackexchange.com/a/40973)