Consensus

Three Kinds

- The problems of mutual exclusion, electing a nominee and multicast are all instances of the more general problem of consensus.
- Consensus problems more generally then are described as one of three kinds:
  1. Consensus
  2. Byzantine Generals
  3. Interactive Consensus

Global Agreement

Consensus

- A set of processes $\{p_1, p_2, \ldots p_n\}$ each begins in the *undecided* state
- Each proposes a single value v_i
- The processes then communicate, exchanging values
- To conclude, each process must set their decision variable d_i to one value and thus enter the *decided* state
- Three desired properties:
  - Termination: each process sets its $decision_i$ variable
  - Agreement: If p_i and p_j are correct processes and have both entered the *decided* state, then $d_i = d_j$
  - Integrity: If the correct processes all proposed the same value v, then any correct process p_i in the *decided* state has $d_i = v$

Global Agreement

Byzantine Generals

- Imagine three or more generals are to decide whether or not to attack
- We assume that there is a commander who issues the order
- The others must decide whether or not to attack
- Either the lieutenants or the commander can be faulty and thus send incorrect values
- Three desired properties:
  - Termination: each process sets its $decision_i$ variable
  - Agreement: If p_i and p_j are correct processes and have both entered the *decided* state, then $d_i = d_j$
  - Integrity: If the commander is correct then all correct processes decide on the value proposed by the commander
- When the commander is correct, *Integrity* implies *Agreement*, but the commander may not be correct

Global Agreement

Interactive Consensus

- Each process proposes its own value and the goal is for each process to agree on a vector of values
- Similar to consensus other than that each process contributes only a part of the final answer which we call the *decision vector*
- Three desired properties:
  - Termination: each process sets its $decision_i$ variable
  - Agreement: The final decision vector of all processes is the same
  - Integrity: If p_i is correct and proposes v_i then all correct processes decide on v_i as the ith component of the decision vector

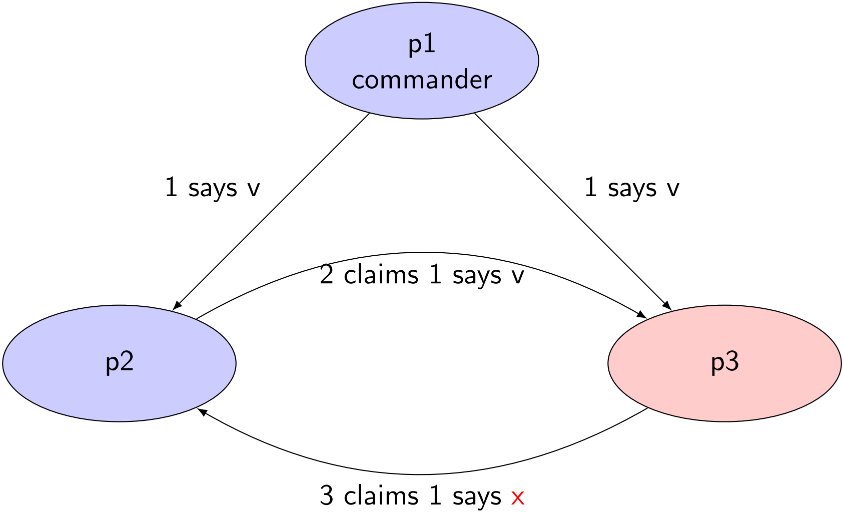
Global Agreement

Relating the three

- Assuming we had a solution to any of the three problems we could construct a solution to the other two
- For example, if we have a solution to Interactive Consensus, then we have a solution to Consensus, all we require is some way consistent function for choosing a single component of the decision vector
  - We might choose a majority function, maximum, minimum or some other function depending on the application
  - It only requires that the function is context independent
- If we have a solution to the Byzantine Generals then we can construct a solution to Interactive Consensus
  - To do so we simply run the Byzantine Generals solution N times, once for each process
- The point is not necessarily that this would be the way to implement such as solution (it may not be efficient)
  - However if we can determine an impossibility result for one of these problems we know that we also have the same result for the others

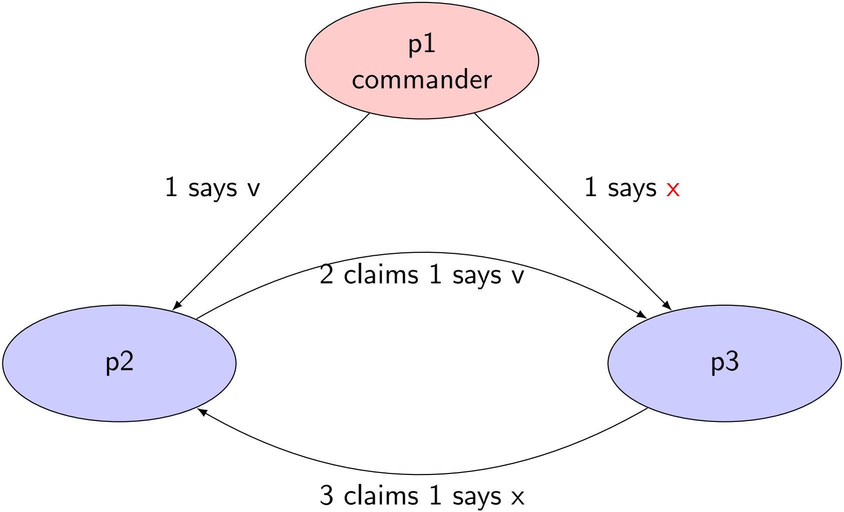
Global Agreement

Byzantine Generals in a Synchronous System



Global Agreement

Byzantine Generals in a Synchronous System



Global Agreement

Impossible

- Recall:
  - Agreement: If p_i and p_j are correct processes and have both entered the *decided* state, then $d_i = d_j$
  - Integrity: If the commander is correct then all correct processes decide on the value proposed by the commander
- In both scenarios, process p_2 receives different values from the commander p_1 and the other process p_3
- It can therefore know that one process is faulty but cannot know which one
- By the *Integrity* property then it is bound to choose the value given by the commander
- By symmetry the process p_3 is in the same situation when the commander is faulty.
- Hence when the commander is faulty there is no way to satisfy the *Agreement* property, so no solution exists for three processes

Global Agreement

$N \leq 3 \times f$

- In the above case we had three processes and at most one incorrect process, hence $N = 3$ and $f = 1$
- It has been shown, by Pease *et al* that more generally no solution can exist whenever $N \leq 3 \times f$
- However there can exist a solution whenever $N > 3 \times f$
- Such algorithms consist of *rounds* of messages
- It is known that such algorithms require at least $f + 1$ message rounds
- The complexity and cost of such algorithms suggest that they are only applicable where the threat is great
- That means either the threat of an incorrect or malicious process is great
- and/or the cost of failing due to inability to reach consensus is large

Global Agreement

Consensus in an Asynchronous System

- Fisher *et al* have shown that it is impossible to design an algorithm which is guaranteed to reach consensus in an asynchronous system, under the following condition:

Global Agreement

Consenus in an Asynchronous System

- Fisher *et al* have shown that it is impossible to design an algorithm which is guaranteed to reach consensus in an asynchronous system, under the following condition:
  - We allow a single process crash failure
- Even if we have 1000s of processes, and the failure is a crash rather than an arbitrary failure of just a single process, any consensus algorithm is not guaranteed to reach consensus
- Clearly this is a pretty benign set of circumstances
- We therefore know that there is no solution in an asynchronous system to either:
  1. Byzantine generals (and hence consensus or interactive consensus)
  2. Totally order and reliable multicast

Consensus in an Asynchronous System

So what to do?

- The important word in the previous impossibility result is: guarantee
- There is no algorithm which is guaranteed to reach consensus
- Consensus has been reached in asynchronous systems for years
- Some techniques for getting around the impossibility result:
  - Masking process failures, for example using persistant storage such that a crashed process can be replaced by one in effectively the same state
    - Thus meaning some operations appear to take a long time, but all operations do eventually complete
  - Employ failure detectors:
    - Although in an asynchronous system we cannot achieve a reliable failure detector
    - We can use one which is “perfect by design”
    - Once a process is deemed to have failed, any subsequent messages that it does send (showing that it had not failed) are ignored
    - To do this the other processes must agree that a given process has failed