which is the majority of the resolved values for all children of the root. For each child j of the root where p_j is nonfaulty, Lemma 5.9 implies that $\mathsf{resolve}_i(j)$ equals $tree_j()$, which is p_j's input v. Since a majority of the children of the root correspond to nonfaulty processors, p_i decides v.

The next lemma is used to show the agreement condition. A node π is *common* in an execution if all nonfaulty processors compute the same reduced value for π, that is, $\mathsf{resolve}_i(\pi) = \mathsf{resolve}_j(\pi)$, for every pair of nonfaulty processors p_i and p_j. A subtree has a *common frontier* if there is a common node on every path from the root of the subtree to its leaves.

Lemma 5.10 *Let π be a node. If there is a common frontier in the subtree rooted at π, then π is common.*

Proof. The lemma is proved by induction on the height of π. The base case is when π is a leaf and it follows immediately.

For the inductive step, assume that π is the root of a subtree with height $k+1$ and that the lemma holds for every node with height k. Assume in contradiction that π is not common. Since by hypothesis the subtree rooted at π has a common frontier, every subtree rooted at a child of π must have a common frontier. Since the children of π have height k, the inductive hypothesis implies that they are all common. Therefore, all processors resolve the same value for all the children of π and the lemma follows since the resolved value for π is the majority of the resolved values of its children. □

Note that the nodes on each path from a child of the root of the tree to a leaf correspond to different processors. Because the nodes on each such path correspond to $f+1$ different processors, at least one of them corresponds to a nonfaulty processor and hence is common, by Lemma 5.9. Therefore, the whole tree has a common frontier, which implies, by Lemma 5.10, that the root is common. The agreement condition now follows. Thus we have:

Theorem 5.11 *There exists an algorithm for n processors that solves the consensus problem in the presence of f Byzantine failures within $f+1$ rounds using exponential size messages, if $n > 3f$.*

In each round, every processor sends a message to every processor. Therefore, the total message complexity of the algorithm is $n^2(f+1)$. Unfortunately, in each round, every processor broadcasts a whole level of its tree (the one that was filled in most recently) and thus the longest message contains $n(n-1)(n-2)\cdots(n-(f+1)) = \Theta(n^{f+2})$ values.

5.2.5 A Polynomial Algorithm

The following simple algorithm uses messages of constant size, takes $2(f+1)$ rounds, and assumes that $n > 4f$. It shows that it is possible to solve the consensus problem

Algorithm 16 A polynomial consensus algorithm in the presence of Byzantine failures: code for p_i, $0 \leq i \leq n-1$.

```
Initially pref[i] = x                          // initial preference for self is for own input
and pref[j] = v_⊥, for any j ≠ i               // default for others

1:  round 2k − 1, 1 ≤ k ≤ f + 1:               // first round of phase k
2:      send ⟨pref[i]⟩ to all processors
3:      receive ⟨v_j⟩ from p_j and assign to pref[j], for all 0 ≤ j ≤ n − 1, j ≠ i
4:      let maj be the majority value of pref[0], ..., pref[n − 1] (v_⊥ if none)
5:      let mult be the multiplicity of maj

6:  round 2k, 1 ≤ k ≤ f + 1:                   // second round of phase k
7:      if i = k then send ⟨maj⟩ to all processors   // king of this phase
8:      receive ⟨king-maj⟩ from p_k (v_⊥ if none)
9:      if mult > n/2 + f
10:       then pref[i] := maj
11:       else pref[i] := king-maj
12:     if k = f + 1 then y := pref[i]         // decide
```

with constant-size messages, although with an increase in the number of rounds and a decrease in the resilience.

The algorithm contains $f+1$ phases, each taking two rounds. Each processor has a preferred decision (in short, *preference*) for each phase, initially its input value. At the first round of each phase, all processors send their preferences to each other. Let v_i^k be the majority value in the set of values received by processor p_i at the end of the first round of phase k. If there is no majority, then a default value, $v_\perp$, is used. In the second round of the phase, processor p_k, called the *king* of the phase, sends its majority value v_k^k to all processors. If p_i receives more than $n/2 + f$ copies of v_i^k (in the first round of the phase) then it sets its preference for the next phase to be v_i^k; otherwise, it sets its preference to be the phase king's preference, v_k^k, received in the second round of the phase. After $f+1$ phases, the processor decides on its preference.

Each processor maintains a local array *pref* with n entries. The pseudocode appears in Algorithm 16.

The following lemmas are with respect to an arbitrary admissible execution of the algorithm. The first property to note is *persistence of agreement*:

Lemma 5.12 *If all nonfaulty processors prefer v at the beginning of phase k, then they all prefer v at the end of phase k, for all k, $1 \leq k \leq f+1$.*

Proof. Since all nonfaulty processors prefer v at the beginning of phase k, each processor receives at least $n-f$ copies of v (including its own) in the first round of phase k. Since $n > 4f$, $n - f > n/2 + f$, which implies that all nonfaulty processors will prefer v at the end of phase k. □

This immediately implies the validity property: If all nonfaulty processors start with the same input v, they continue to prefer v throughout the phases (because the preference at the end of one phase is the preference at the beginning of the next); finally, they decide on v at the end of phase $f + 1$.

Agreement is achieved by the king breaking ties. Because each phase has a different king and there are $f + 1$ phases, then at least one phase has a nonfaulty king.

Lemma 5.13 *Let g be a phase whose king p_g is nonfaulty. Then all nonfaulty processors finish phase g with the same preference.*

Proof. Suppose that all nonfaulty processors use the majority value received from the king for their preference (Line 11). Since the king is nonfaulty, it sends the same message and thus all the nonfaulty preferences are the same.

Suppose that some nonfaulty processor, say p_i, uses its own majority value, say v, for its preference (Line 10). Thus p_i receives more than $n/2 + f$ messages for v in the first round of phase g. Consequently every processor, including the king p_g, receives more than $n/2$ messages for v in the first round of phase g and sets its majority value to be v. Thus, no matter whether it executes Line 10 or Line 11 to set its preference, every nonfaulty processor has v for its preference. □

Therefore, at phase $g + 1$ all processors have the same preference, and the persistence of agreement (Lemma 5.12) implies that they will decide on the same value at the end of the algorithm. This implies that the algorithm has the agreement property and solves the consensus problem.

Clearly, the algorithm requires $2(f + 1)$ rounds and messages contain one bit. Thus we have:

Theorem 5.14 *There exists an algorithm for n processors that solves the consensus problem in the presence of f Byzantine failures within $2(f+1)$ rounds using constant size messages, if $n > 4f$.*

5.3 IMPOSSIBILITY IN ASYNCHRONOUS SYSTEMS

We have seen that the consensus problem can be solved in synchronous systems in the presence of failures, both benign (crash) and severe (Byzantine). We now turn to asynchronous systems. We assume that the communication system is completely reliable and the only possible failures are caused by unreliable processors. We show that if the system is completely asynchronous, then there is no consensus algorithm even in the presence of a single processor failure. The result holds even if processors fail only by crashing. The asynchronous nature of the system is crucial for this impossibility proof.

This impossibility result holds both for shared memory systems, if only read/write registers are used, and for message-passing systems. We first present the proof for shared memory systems in the simpler case of an $(n - 1)$-resilient algorithm (also called a *wait-free* algorithm), where all but one of the n processors might fail. Then