A LAYERED ANALYSIS OF CONSENSUS*

YORAM MOSES † AND SERGIO RAJSBAUM‡

Abstract. This paper introduces a simple notion of *layering* as a tool for analyzing well-behaved runs of a given model of distributed computation. Using layering, a model-independent analysis of the consensus problem is performed and then applied to proving lower bounds and impossibility results for consensus in a number of familiar and less familiar models. The proofs are simpler and more direct than existing ones, and they expose a unified structure to the difficulty of reaching consensus. In particular, the proofs for the classical synchronous and asynchronous models now follow the same outline. A new notion of connectivity among states in runs of a consensus protocol, called *potence connectivity*, is introduced. This notion is more general than previous notions of connectivity used for this purpose, and plays a key role in the uniform analysis of consensus.

1. Introduction. For almost two decades now, the consensus problem has played a central role in the study of fault-tolerant distributed computing, e.g. [34, 18, 17, 14, 19, 27, 21, 11, 12]. It has clearly received the greatest amount of attention of any problem treated in the theoretical literature on distributed computing, and has been studied in a large variety of models and under many types of failure assumptions. The structure of the consensus problem in different settings is frequently based on closely related notions. However, proofs for different models are often based on distinct and somewhat ad hoc techniques. In particular, there have been considerable differences between the study of consensus in asynchronous models, and its study in synchronous and partially synchronous ones.

In order to cope with the proliferation of distributed computing models, researchers have proposed a variety of simulations between models. The aim is to establish a relation (often of equivalence) between the possibility of solving problems of certain types in different models. This is used to establish impossibility results in particular models, or to provide a systematic way to transform protocols written in one model into protocols for another model. Various such simulations have been given, e.g. between shared memory and message passing [3]; between snapshot shared memory and read/write shared memory [2], or between immediate snapshot shared memory and read/write shared memory [7, 8]; between synchronous and asynchronous message passing [1]; between two shared memory models of different resilience [30].

This paper attempts to present a uniform approach to the study of solvability of consensus in various models of computation, in which, intuitively, crash failure behavior can occur. That is, where a process can be silenced from some point on in an execution, and thus appear as if it has crashed. The standard forms of malicious (Byzantine) faults [34], in which faulty processes may behave in an arbitrary manner satisfy the crash-like behavior condition, as do models of omission failures and, of course, the usual models of crash failures. Our results apply even to situations in which there are only link failures [35], provided there are sufficiently many of them to enable the environment to silence a process.

---

* This paper is the first of two parts. A preliminary version reporting results of these papers appeared in [33].

†Dept. of Electrical Engineering, Technion, Israel, yoram@ee.technion.ac.il. Supported in part by an ATS grant, and by a grant from the Fund for advancement of research.

‡ Instituto de Matemáticas, UNAM Ciudad Universitaria, D.F. 04510, México. rajsbaum@math.unam.mx. On leave at Compaq Cambridge Research Laboratory, One Cambridge Center, Cambridge MA 02142. Sergio.Rajsbaum@compaq.com. Partially supported by Conacyt and DGAPA-UNAM Projects.

We start by presenting an alternative proof of the impossibility of consensus in the asynchronous shared-memory model treated by Loui and Abu-Amara [27]. This proof is based on a new notion of connectivity which we call *potence connectivity*, and on an analysis based on a nicely structured subset of the runs. This subset consists of runs that are obtained by imposing a round-by-round *layering* structure on the model. Roughly, we use layerings in the following sense. Given a model of distributed computation, we identify particular legal sequences of actions for the scheduler, each of which we think of as generating a "layer". We require that any way of performing such layers repeatedly starting from a legal initial state will result in a legal run in the model. Thus, in a precise sense, such a layering can be viewed as defining a submodel of the original model.[1] Any protocol for the original model translates directly to one in the submodel. Moreover, the model and submodel generally share many features. In particular, lower bounds and impossibility results proven for the submodel translate directly into the original model. The use of layerings facilitates performing round-by-round analysis: Almost all of our results regarding consensus will follow from analyzing a single layer of computation.

The benefit of working in a submodel or a set of runs with a simpler structure than that of the original model is well known; some recent examples are [4, 7, 8, 11, 25, 36].

In this paper we concentrate on proving lower bounds and impossibility results for the consensus problem; in a sequel paper (as briefly described in [33]) we show how the ideas of this paper extend naturally to other decision tasks, and are useful also to prove solvability results. First, we perform an abstract and model-independent analysis of consensus using layering. Based on this analysis, implications for specific models are obtained by demonstrating that appropriate layerings can be defined in the model. We exemplify this approach by applying it to the following models: Shared-memory asynchronous model with one crash failure, message-passing asynchronous model with one crash failure, message-passing synchronous model with one mobile failure, and message-passing synchronous model with t crash failures.

For the asynchronous models, we describe two styles of layerings: the *synchronic* and the *permutation* layering. The synchronic layerings we consider define submodels of the asynchronous models that are very close in structure to being synchronous, thereby defining "almost synchronous" submodels of the asynchronous models. Indeed, we show that synchronic layerings can be applied to the synchronous message passing model too. The permutation layering is inspired by the immediate snapshot wait-free model of [7, 36], although we define it both for the message passing and for the shared memory models, both 1-resilient. This appears to be the first variant of the immediate snapshot model suggested for a message-passing model.

Regarding consensus, we provide:

- New, simple and uniform lower bound and impossibility proofs for the standard synchronous and asynchronous models. In particular, we show a simple bivalence-style proof for the synchronous case, and a direct round-by-round construction of a bivalent run (not employing a critical state argument) in asynchronous cases.
- Stronger impossibility results with respect to submodels of the full asynchronous models, in which there is only a small degree of asynchrony. These demonstrate how little asynchrony is needed to make consensus unsolvable.

In a sequel paper we show that the asynchronous models are equivalent in terms

[1] Layering saves us the trouble of explicitly defining the submodel as a model of computation with a new transition function, new actions, etc.

of the 1-resilient solvability of decision tasks. In particular, the slightly asynchronous submodels are no more powerful than the fully asynchronous ones. Moreover, in a precise sense, these models are strictly stronger than what can be done t-resiliently in t rounds of the standard synchronous model.

We consider the layering technique to be useful in a number of ways:

- It provides a tool for performing model-independent round-by-round analysis of decision tasks and related problems.
- Results are obtained directly, and not by means of specially tailored reductions.
- Popular topological treatments (e.g. [32, 7, 24, 36]) focus on the local *final* states of processes. We consider states at intermediate stages of the computation as well. Moreover, the state of the environment is represented as an explicit component in the global state, which facilitates the treatment of message-passing as well as shared memory models.
- We make use of a novel notion of *potence connectivity* of a set of states, whose definition depends on the decisions taken by the protocol in the possible futures of these states. In addition we use the more traditional notion of connectivity based on indistinguishability of states. The combined use of the two notions proved very useful for unifying the analysis of consensus in the synchronous and asynchronous models.

Our analysis in this paper concentrates on the consensus problem. By focusing on this "basic" case we obtain a direct and uniform analysis in simple and elementary terms. The proofs of all of our results are short and rather straightforward, which suggests to us that the notions we use are fairly natural. We believe that, with small modifications, the same type of analysis and style of reasoning can be applied to study more general problems involving topological connectivity of higher dimensionality (e.g. [32]).

There are two other papers that attempt to unify synchronous and asynchronous models in a round by round style. The research project of [20], concurrent and independent to ours, is based on a notion of *fault detectors*. Then there is the work of [23], which uses topological techniques in synchronous, asynchronous and partially synchronous message passing systems, with applications to set-consensus. There are two other papers that independently discovered bivalence arguments for the synchronous consensus lower bound: in the randomized setting there is [9], while in the deterministic setting (same as our application in Section 7.2) there is [6]. A different abstract model based on the runs of a distributed system is proposed in [28]. Recently our layering technique was used to prove a synchronous lower bound for uniform consensus [26].

This paper is organized as follows. In Section 2 we define the consensus problem and the basic elements of a distributed computing model, such as processes, environment (which can be used to model different communication mechanisms), states, actions, runs, and failures. In Section 3 we illustrate the layering approach in the concrete setting of an asynchronous shared memory system, by proving the impossibility of consensus. The notion of potence of a state is introduced here. This section provides a complete example of the use of our ideas; a reader uninterested in the full generality of the approach developed in the remaining of the paper could stop here. In Section 4 we present a generic framework for defining models of a distributed system. As an example, we show how the synchronous mobile failures model is captured in our abstract framework. In Section 5 we develop the generic setting for using potence

connectivity to study consensus in the presence of crash-like failure behavior. In addition, the connection between potence connectivity and earlier notions of connectivity is formalized and proven. In Section 6 we introduce layering in the generic setting, and its basic properties. In Section 7.1 we illustrate the ideas with an impossibility result in the mobile failures (synchronous) model, demonstrating that asynchrony is not necessary in order to make consensus impossible. In Section 7.2 we apply our framework to provide a new proof for the classic synchronous message-passing model. More applications are presented in Section 8 for asynchronous systems, where various particular layerings are described. The conclusions are in Section 9.

2. Preliminaries.

2.1. Consensus. In the consensus problem, we start out in an initial state in which the local state of a process i consists of a binary *initial value* variable $\mathtt{v}_i \in \{0,1\}$ and an undefined write-once *decision* variable $\mathtt{d}_i = \perp$. All communication channels (if any exist) are empty, and shared variables (if any exist) all have an undefined value of $\perp$. We denote the set of (all 2^n) initial states of consensus by Con_0. A protocol for consensus is a deterministic protocol D all of whose runs satisfy the following three properties:

Agreement: All nonfaulty processes reach the same decision;
Decision: Every nonfaulty process irrevocably *decides* on a value; and
Validity: In runs in which all processes start with the same initial value w, the value that the nonfaulty processes decide on is w.

As presented, the consensus problem is well-defined only once we have provided a model of computation. In this model, we must, in particular define the structure of local and global states, as well as what protocols are and how they generate runs (computations).

In addition, we need to define when a process is faulty in a run, since the consensus problem distinguishes between the behavior of faulty processes and that of nonfaulty ones. In Section 2.2 we introduce the most basic elements of a distributed system: states, actions and runs.[2] Then in Section 2.3, we consider the notion of a system, which is simply a set of runs of the model with an associated definition of who is faulty in each of the runs.

2.2. States, Actions and Runs. Throughout the paper we will assume there is a fixed finite set of $n \geq 2$ processes, which we shall denote by $1, 2, \ldots, n$, and an *environment,* denoted by e, which is used to model aspects of the system that are not modeled as being part of the activity or state of the processes. For example, we will model the communication channels or shared variables as being part of the environment's state. In addition, we will assume that various nondeterministic choices such as various delays and failures are actions performed by the environment. (What are typically thought of as actions of the "scheduler" or the "adversary" we model as actions of the environment.)

For every $i \in \{e, 1, 2, \ldots, n\}$, we assume there is a set L_i consisting of all possible *local states* for i. The set of *global states,* which we will simply call *states,* will consist of

$$\mathcal{G} \;=\; L_e \times L_1 \times \cdots \times L_n.$$

[2] Our modeling here and in Section 4.1 is based on the work of [31] which in turn extends the modeling style of [16].

We denote by x_i the local state of i in the state x.

In a given setting, every $i \in \{e, 1, 2, \ldots, n\}$ is associated with a nonempty set ACT_i of possible actions. Intuitively, these may include shared memory operations that the process can perform, messages the process sends, and any internal bookkeeping operations or computations the process may perform. In principle, a single action can cause a number of operations to take place. However, the important point is that the decision to perform this action is taken atomically. We find it convenient to model the environment as performing actions as well. Depending on the model, the environment's actions may involve the delivery of messages, the loss of messages, determining what failures happen and when they occur, resolving race conditions, etc. We also think of the environment as being in charge of **scheduling** the processes, determining which processes are to move in each round of the computation. A *scheduling action* is a set $\mathsf{Sched} \subseteq \{1, \ldots, n\}$ of the processes that are scheduled to move next. We assume the existence of a set $\mathsf{act_e}$ describing the aspects of the environment's actions that handle everything other than the scheduling of processes. Without loss of generality we will assume that an environment's action (an element in ACT_e) is a pair (Sched, a) where Sched is a scheduling action and $a \in \mathsf{act_e}$. A *joint action* is a pair $\bar{\mathsf{a}} = (\epsilon, \mathbf{a})$ where $\epsilon = (\mathsf{Sched}, a)$ is in ACT_e, and $\mathbf{a}$ is a function with domain Sched such that $\mathbf{a}(i) \in \mathsf{ACT}_i$ for each $i \in \mathsf{Sched}$. Thus, $\bar{\mathsf{a}}$ specifies an action for the environment, as well as (via $\mathbf{a}$) an action for every process that is scheduled to move. We define the set of joint actions by $\overline{\mathsf{ACT}}$. Clearly, $\overline{\mathsf{ACT}}$ is determined by a collection of action sets $\{\mathsf{ACT}_i\}_{i=1,\ldots,n}$ and a set $\mathsf{act_e}$. Roughly speaking, joint actions are the events that cause the global state to change into a new state. Thus, for example, a joint action in which process i sends j a message m, and the environment delivers the message m' to process i' will typically cause the local state of i' to change, as well as modifying the state of the communication channel between i and j (this state will be part of the environment's local state). This is formally captured by the notion of a *transition function*, which is a function $\tau : \mathcal{G} \times \overline{\mathsf{ACT}} \to \mathcal{G}$ from global states and joint actions to global states, describing how a joint action transforms the global state.

A *deterministic protocol* for i is a function $P_i : L_i \to \mathsf{ACT}_i$ specifying the action that i is ready to perform in every state of L_i. A *nondeterministic protocol* for i is a function $P_i : L_i \to 2^{\mathsf{ACT}_i} \setminus \emptyset$ specifying for every state of L_i a nonempty set of actions one of which i must perform in that state. In this paper, we will focus our attention on the case in which the the processes follow deterministic protocols, while the environment may follow a nondeterministic protocol.[3]

Intuitively, we think of a run as consisting of an infinite sequence of global states and the joint actions that cause the transitions among them. Notice that once we fix a deterministic protocol $D = (D_1, \ldots, D_n)$ for the processes, an action $\epsilon = (\mathsf{Sched}, a)$ of the environment uniquely determines the joint action $\bar{\mathsf{a}} = (\epsilon, \mathbf{a})$ that will be performed in a given (global) state x: the set Sched determines the processes that participate in the joint action, and $\mathbf{a}(i) = D_i(x_i)$ for each $i \in \mathsf{Sched}$. Proving lower bound and impossibility results can often be thought of as showing that the adversary, which here is the environment, always has a strategy that can guarantee the desired bad behavior. With this end in mind, we will represent a joint action by specifying the

[3]The assumption that the environment's protocol may be nondeterministic is necessary for a faithful description of many models of interest. The assumption that processes follow deterministic protocols is without loss of generality in this paper, in which we are interested in worst-case lower bounds and impossibility results. It is well known and straightforward to show that any result of this type for a protocol for consensus in a given model that is proved for deterministic protocols applies to the more general class of nondeterministic protocols as well.

environment action that determines it. Formally, we model a *run* over $\mathcal{G}$ and ACT_e as a pair $R = (r, \alpha)$, where $r : N \to \mathcal{G}$ is a function from the natural numbers to $\mathcal{G}$ defining an infinite sequence of states of $\mathcal{G}$, and $\alpha : N \to \mathsf{ACT}_e$ defines a corresponding sequence of environment actions. The intuition will be that the joint action caused by $\alpha(k)$ and the underlying protocol leads us from a state $r(k)$ to a state $r(k+1)$. As we will see later on, once we fix a model of computation and a protocol D for the processes to follow, there will be additional conditions relating the sequences r and α. These conditions guarantee, for example, that the actions recorded by α do indeed cause the transitions among the corresponding states recorded by r. The state $r(0)$ is called the *initial state* of the run R. We denote by $r(k)_i$ (resp. $r(k)_e$) the local state of process i (resp. of the environment) in $r(k)$.

An *execution* is a finite or infinite subinterval of a run, starting and ending in a state, as described next. For a run $R = (r, \alpha)$ and a pair $m \leq m'$ where m is finite and m' is finite or infinite, we denote by $R[m, m']$ the execution starting at the state $r(m)$ and ending in $r(m')$, and behaving as R does between them. Formally, $R[m, m'] = (\sigma, \beta)$ where σ has domain $[0, m' - m]$ and β has domain $[0, m' - m - 1]$, and they satisfy $\sigma(k) = r(m + k)$ and $\beta(k) = \alpha(m + k)$ for every k in their respective domains. Notice that, in principle, the same execution can occur in different runs, and for that matter even at different times. A *suffix* of a run R is an execution of the form $R[m, \infty]$ for some finite m; similarly, a *prefix* of R is an execution of the form $R[0, m]$.

Given an execution R (possibly consisting of just one state), let us denote by $R \odot \epsilon$ the execution that results from extending R by having the environment perform the action ϵ. In models in which performing a joint action at a state results in a unique next state (which will invariably be the case in this paper), every run of a deterministic protocol D can be represented in the form $x \odot \epsilon_1 \odot \epsilon_2 \odot \cdots$ where x is an initial state and ϵ_i is an environment action, for every integer $i \geq 0$.

2.3. Systems and Failures. We define a *system* to be a pair $(\mathcal{R}, \mathsf{Faulty})$ where $\mathcal{R}$ is a set of runs and Faulty is a predicate on processes and runs of $\mathcal{R}$. In the sequel, $\mathsf{Faulty}(i, R)$ will be taken to mean that i is faulty in the run R. We often focus on the runs of a system $\mathcal{S} = (\mathcal{R}, \mathsf{Faulty})$, and write $R \in \mathcal{S}$ as shorthand for $R \in \mathcal{R}$. Notice that Faulty determines who is faulty in a run as a function of the *whole* infinite run. Obviously, in some models of distributed computation it is possible to determine that a process is faulty by considering only a prefix of the run, sometimes even a single state. In other cases, however, the full history of the run is needed in order to determine who is faulty (this is the case, for example, in the asynchronous models of [19, 27]).

With respect to a system $\mathcal{S}$, a state y is said to *extend* the state x if there is a finite execution in some run of $\mathcal{S}$ that starts in x and ends in y. A run R is said to *contain* a state x if x is one of the global states in R. For conciseness, we will use terminology such as: *a state x of $\mathcal{S}$* when we mean a state x appearing in a run of $\mathcal{S}$, or *an initial state of $\mathcal{S}$* when we mean a state appearing as an initial state in a run of $\mathcal{S}$. By convention, x extends x for every state x of $\mathcal{S}$.

3. Proving Impossibility Using Layering. Our purpose in this section is to give a simple and elegant proof of the FLP impossibility result for consensus [19], based on the notion of layering. This concrete example serves to introduce the more general ideas developed in the rest of the paper. We will present the proof for the asynchronous shared-memory model [27]. Later on in the paper we will describe the properties that play a role in the proof. The standard asynchronous shared memory

model is well known, and detailed formal descriptions can be found in textbooks such as [5, 29]. We now briefly review the features of the model that are relevant for the analysis presented in this section.

We assume the standard asynchronous shared memory model where n processes, $n \geq 2$, communicate by reading and writing to single-writer/multi-reader, shared variables, and one process can crash. A (global) *state* x of the system is a tuple specifying a local state x_i for every process i, and the state of the environment, which in this case consists of the assignment of values to the shared variables, as well as the set of pending shared memory operations, and the set of pending reports for read operations that have been recorded (the value has been read) but not yet reported to the reading process. The pending operations are the read and write operations that have been issued for these variables, and have not yet taken effect.

The sets ACT_i and ACT_e of the actions of the processes and the environment are defined as follows. A process performs an action only when it is scheduled to move. This action is either a local operation, a read of a shared variable (belonging to it or to some other process), or a write to one of its own variables. An action of the environment can have one of three forms: (a) scheduling a process to move — resulting in the process performing an action, (b) performing a pending shared memory operation, or (c) reporting the value read in a recorded read operation to the reading process.

A process that is scheduled to move only a finite number of times in a given run R is said to have *crashed*. We define $\mathsf{Faulty}(i, R)$ to hold, and consider i to be *faulty in* R exactly if i crashes in R. (Notice that in this model, there is no way to determine at a finite state x that a given process is faulty in the run; a process can always "recover" in the future.) As mentioned earlier, we consider deterministic protocols without loss of generality, because any nondeterministic protocol that solves consensus in this model can, by fixing the nondeterministic choices in a fixed arbitrary way, be turned into a deterministic protocol solving it.

A run of a given deterministic protocol D in this model is a run $R = (r, \alpha)$ satisfying:

(i) $r(0) \in \mathsf{Con}_0$,

(ii) each process follows its protocol, and every pair of consecutive states are related according to the operations that take place as scheduled by the environment.

If in addition

(iii) every read and write action issued is eventually serviced appropriately by the environment, and

(iv) at most one process fails in R,

then we say that the run is *admissible*. Let $\mathcal{S}(D)$ the system consisting of the set of all admissible runs of D, and the predicate Faulty described above. Now, the consensus problem is well-defined: D solves consensus if all runs of $\mathcal{S}(D)$ satisfy agreement, decision, and validity with failures as defined by Faulty.

3.1. Layers in the shared-memory model. We define a *layer* to consist of a finite sequence of environment actions. Our intention is to focus on runs of a protocol that are generated by a particular set of layers. If the set of layers is chosen appropriately, these runs can have structural properties that will simplify their analysis.

We define the set L^{rw} of layers in the asynchronous shared-memory model to consists of all layers of the forms

- $[p_1, \ldots, p_n]$ and

- $[p_1, \ldots, p_{n-1}]$,

where the p_i's are process names (elements of $\{1, \ldots, n\}$), and the names appearing in a given layer are pairwise distinct. We think of layers of the first type as *full* layers, since in such a layer every process moves. Layers of the second type enable the environment to "silence" a process from any point in the computation; this will play an important role in determining the "topological" properties of the layered runs (e.g., in Lemma 3.4(b)). A layer specifies a linear ordering in which the environment schedules the processes to move. In a given layer, whenever a process p_i is scheduled to move, it performs a single action, and this action (internal, a read or a write) is serviced (i.e., the read or write action is performed and, in the case of a read, the value that was read is reported to the reader) before the next process moves. Since in our model every process can be scheduled to move at any state (although in some cases the pending operation may just be a "skip" internal operation), the sequence of environment actions described in a layer of L^{rw} can be applied at every state. The intuition behind the definition of L^{rw} is that a layer consists of a "round" in which at least $n-1$ processes get to move (sequentially). Notice that in an infinite sequence of layers, at least $n-1$ process names appear infinitely often. The layers just defined are designed to ensure that each layer contributes towards the fairness conditions (iii) and (iv). As a result, every run generated by an infinite sequence of such layers is admissible. Notice that a layer of the first type is specified by a permutation on $\{1, \ldots, n\}$. We will find it useful to identify a layer of the first type with the corresponding permutation. Because of this connection we call this layering a *permutation layering.*

Once we have fixed the protocol D, we can think of a layer $L \in \mathsf{L}^{rw}$ as an individual "higher-level" action by the environment. An individual environment step in a run would then consist of performing a whole layer. We define *an* L^{rw}*-run of* D to be a run of the form

$$x^0 \odot L_1 \odot L_2 \odot \cdots,$$

where $x^0 \in \mathsf{Con}_0$ and $L_i \in \mathsf{L}^{rw}$ for every $i \geq 1$. More formally, let us denote by $x \cdot L$ the state that is reached at the end of an execution that starts in x, where the processes follow D and the environment performs actions according to L. The L^{rw}-run depicted above is a run $R_{\mathsf{L}} = (r_{\mathsf{L}}, \alpha)$ with $\alpha = (L_1, L_2, \ldots)$ and $r_{\mathsf{L}} = (x^0, x^1, \ldots)$ where for $i \geq 0$ we inductively define $x^{i+i} = x^i \cdot L_{i+1}$. Thus, in an L^{rw}-run, we view the environment as performing actions consisting of whole layers, and we ignore the intermediate states that arise "in the middle" of a layer.

Notice that every L^{rw}-run R_{L} of D corresponds to a unique run $R \in \mathcal{S}(D)$ that is obtained from R_{L} by "expanding" the layers into the detailed sequence of environment actions they describe, and adding the intermediate states. We call a process *faulty* in R_{L} if it is faulty in the corresponding run R. We denote by $\mathcal{S}_{\mathsf{L}}(D)$ the system consisting of the L^{rw}-runs of D, with this definition of failures. The argument above that executing an infinite number of L^{rw} layers yields an admissible run can now be formalized as follows.

LEMMA 3.1. *For every run $R_{\mathsf{L}} \in \mathcal{S}_{\mathsf{L}}(D)$, the run $R \in \mathcal{S}(D)$ corresponding to R_{L} is admissible.*

Proof. By definition of R_{L}, the corresponding run R is clearly a run of D in the model. It is admissible because (a) by definition of the layers, each read and write action that is performed by a process is serviced immediately, and (b) at most one process can fail in R because in each of the infinite sequence of layers that generate R

at least $n-1$ of the n processes is scheduled to perform an action; hence, at most one process can move only a finite number of times in R. □

3.2. Potence Connectivity. Recall that decisions made by the processes appear in their local states from the point of decision on. Therefore, any infinite subsequence of the states of a run will contain the information about the decisions that the different processes perform in the run. Thus, we do not lose information about the decision values by considering the states of an L^{rw}-run instead of looking at the full detail of the run corresponding to it. If a protocol D solves consensus in the asynchronous model just described it must do so in *every* admissible run. It follows that in every L^{rw}-run of D the nonfaulty processes must decide on a value $v \in \{0,1\}$. A crucial role in the proof of impossibility of consensus in the asynchronous shared-memory model is played by the decision values that are possible in the future of a given global state. This is captured by the notion of the *potential valence* (or *potence* for short)[4] of a state with respect to a set of runs.

DEFINITION 3.2. *A state x is w-***potent** *with respect to a system $\mathcal{S}$ if x is a state of a run $R \in \mathcal{S}$ in which at least one nonfaulty process decides w. The state x is* **bipotent** *if it is both 0-potent and 1-potent.*

When discussing potence, we sometimes omit the system $\mathcal{S}$ when it is clear from context. Notice that if a state x is w-potent (resp. bipotent) with respect to $\mathcal{S}_{\mathsf{L}}(D)$ then it is guaranteed to be w-potent (resp. bipotent) with respect to $\mathcal{S}(D)$: If R_{L} is the witness proving that x is w-potent in $\mathcal{S}_{\mathsf{L}}(D)$ then x appears in the run $R \in \mathcal{S}(D)$ corresponding to R_{L}, and R is a witness to w-potence of x with respect to $\mathcal{S}(D)$. Clearly, the converse need not hold. Bipotent states play an important role in delaying (or precluding) consensus, as we shall see in the next section. Our goal will be to show that every consensus protocol must have a run where agreement is not reached. We will do this in the next section, by demonstrating that every protocol for consensus must have a run whose states are all bipotent. Such a run we call a *bipotent run.* The notion of *potence connectivity* is a powerful tool for proving this, and other impossibility results.

DEFINITION 3.3. *With respect to a system $\mathcal{S}$:*

(i) Two states x and y have **shared potence**, *denoted by $x \sim_p y$, if both are w-potent, for some $w \in \{0,1\}$.*

(ii) A set of states X is **potence connected** *if the graph $(X, \sim_p)$ induced by $\sim_p$ on X is connected.*

Potence connectedness is not a very strong condition. Indeed, it is easy to check that a set X of states is potence connected exactly if either (i) for some value w, all states of X are w-potent, or (ii) there exists at least one bipotent state in X. Equivalently, X is not potence connected exactly if X contains both 0-potent and 1-potent states, but does not contain a bipotent state.

The following criterion will serve us to prove that two states have shared potence.

LEMMA 3.4. *Assume the protocol D satisfies the decision property, and let x and x' be states of $\mathcal{S}_{\mathsf{L}}(D)$. If there are states $y, y' \in \mathcal{S}_{\mathsf{L}}(D)$, where y extends x and y' extends x', such that y and y' differ at most in the local state of a single process, then $x \sim_p x'$ with respect to $\mathcal{S}_{\mathsf{L}}(D)$.*

[4] In [33] we used the term *valence* instead of potence. This is changed here because the term valence is used slightly differently in the literature, starting with [19]. Briefly, a state is said to be w-valent, if *all* extensions decide w. (However, bivalent is equivalent to bipotent.) We thank Gerard Tel for suggesting the term.

Proof. Let R_x be an execution in $\mathcal{S}_{\mathsf{L}}(D)$ that ends in state x, and let $R_{x'}$ be one that ends in x'. Since y extends x, there is a finite sequence of layers σ such that $R_x \odot \sigma$ ends in y, and a corresponding sequence σ' such that $R_{x'} \odot \sigma'$ ends in y'. By assumption, we have that y and y' differ at most in the local state of process p. Choose a layer $L \in \mathsf{L}^{rw}$ in which p is not scheduled to move. We extend each state by applying L repeatedly. There are two runs $R, R' \in \mathcal{S}_{\mathsf{L}}(D)$ such that $R = R_x \odot \sigma \odot L^\infty$ and $R' = R_{x'} \odot \sigma' \odot L^\infty$. Both are clearly runs of $\mathcal{S}_{\mathsf{L}}(D)$, and in both runs process p is faulty and the rest are not faulty. A straightforward induction on the number of actions performed by the environment shows that all processes other than p have the same local state history in the suffix of R starting at y as they do in corresponding points in the suffix of R' starting at y'. It follows that the nonfaulty processes reach the same decision in both. Since D satisfies the decision property, decisions are reached by the nonfaulty processes in these runs, and we have that $x \sim_p x'$. □

Lemma 3.4 directly captures as particular instances many useful cases. Thus, for example, it implies that if x extends y then they have a shared potence. Similarly, if a state z extends both x and y, then $x \sim_p y$. A third useful instance is that if x and y differ only in the local state of one process then $x \sim_p y$.

The crux of the impossibility proof is captured in the following lemma, which shows that the set of successors of a state in $\mathcal{S}_{\mathsf{L}}(D)$ is guaranteed to be potence connected.

LEMMA 3.5. *Assume D satisfies the decision property. For every state x of $\mathcal{S}_{\mathsf{L}}(D)$, the set of states $\mathsf{L}^{rw}(x) = \{y \mid y = x \cdot L$, for some $L \in \mathsf{L}^{rw}\}$ is potence connected with respect to $\mathcal{S}_{\mathsf{L}}(D)$.*

Proof. We start by showing two cases in which states of $\mathsf{L}^{rw}(x)$ have a shared potence, from which the result will follow. For every permutation $p_1, p_2, \ldots, p_n$ of $\{1, \ldots, n\}$ we claim the following holds:

(i) $x \cdot [p_1, \ldots, p_{n-1}] \sim_p x \cdot [p_1, \ldots, p_n]$.

This is true by Lemma 3.4, because there is a state of $\mathcal{S}_{\mathsf{L}}(D)$ that extends both states. This state results from extending the first state by applying the layer $L = [p_n, p_1, \ldots, p_{n-1}]$, and it also results from extending the second state by $L' = [p_1, \ldots, p_{n-1}]$. Clearly,

$$x \cdot [p_1, \ldots, p_{n-1}] \cdot [p_n, p_1, \ldots, p_{n-1}] \quad = \quad x \cdot [p_1, \ldots, p_n] \cdot [p_1, \ldots, p_{n-1}],$$

since in both cases exactly the same actions take place in the same order in the two layers following x.

For a permutation $\pi = [p_1, \ldots, p_n]$, we denote the permutation $[p_1, \ldots, p_{k-1}, p_{k+1}, p_k, p_{k+2}, \ldots, p_n]$ that is obtained by transposing the k^{th} and $(k+1)^{\text{st}}$ elements of π by $\mathsf{Tr}(k, \pi)$. We claim

(ii) Let π be a full layer and $k \in \{1, \ldots, n-1\}$. Then $x \cdot \pi \sim_p x \cdot \mathsf{Tr}(k, \pi)$.

Let $y = x \cdot \pi$ and $y' = x \cdot \mathsf{Tr}(k, \pi)$. In moving from x to both y and y', every individual process performs the same action. Recall that in our model each shared variable can be written by a single process. Thus, every process that executes a write, writes the same value in both cases, and the state of the shared memory is the same in y and in y'. In addition, every process except possibly for p_k and p_{k+1} end up in the same local state in y and in y'. Now, if both processes perform a read, or both perform a write, each of them will end up in the same local state in y and in y'. Finally, if one of them executes a read and the other a write, only the reading process may end up in a different state in y and in y'. It follows that y and y' differ at most in the local state of one process, so $y \sim_p y'$ by Lemma 3.4.

The potence connectivity of $\mathsf{L}^{rw}(x)$ follows: A well-known property of the group of permutations is that we can transform any permutation π of $\{1,\ldots,n\}$ to any other permutation π' using a finite number of single transpositions $\mathsf{Tr}(k,\pi)$ as in (ii). Thus, by transitivity of connectivity we have that all states $x \cdot \pi$ obtained from x by a full layer are potence connected. Each of the remaining states of $\mathsf{L}^{rw}(x)$ is obtained by a layer $[p_1,\ldots,p_{n-1}]$, and is connected to the rest by part (i).
□

The last connectivity property is implicit in [19]:

LEMMA 3.6. *If D satisfies the decision property, then Con_0 is potence connected with respect to $\mathcal{S}_{\mathsf{L}}(D)$.*

Proof. In this proof, given a state z we denote by z_j the local state of process j in the state z. Let $x, y \in \mathsf{Con}_0$, and for every $0 \leq m \leq n$ define x^m by setting

$$x_j^m = \begin{cases} y_j & \text{for all } j \leq m; \text{ and} \\ x_j & \text{for all } j > m. \end{cases}$$

Clearly, $x^m \in \mathsf{Con}_0$, and it is easy to check that $x^0 = x$ and $x^n = y$. Moreover, for every $0 < l \leq n$ we have that x^{m-1} and x^m differ exactly in the local state of process m. Con_0 is a subset of the states of $\mathcal{S}_{\mathsf{L}}(D)$. Hence, by Lemma 3.4 we have that $x^{m-1} \sim_p x^m$. It follows that x and y are potence connected and we are done. □

3.3. Using bipotence to prove impossibility. In the previous section we used the decision requirement of consensus together with the possibility of having one process crash, to establish connectivity properties. In this section we show how these properties, together with the agreement and validity requirements of consensus, imply the impossibility result. We start by demonstrating that bipotent states can play a role in delaying, or precluding consensus.

LEMMA 3.7. *Assume the protocol D satisfies the agreement and decision properties. If a state x of $\mathcal{S}_{\mathsf{L}}(D)$ is bipotent with respect to $\mathcal{S}_{\mathsf{L}}(D)$, then no process has decided in x.*

Proof. Assume by way of contradiction that i has decided on value w in x. Since x is bipotent, there is a run $R \in \mathcal{S}_{\mathsf{L}}(D)$ containing x in which some process, say j, decides $1-w$. Let y be a state of this run extending x in which j has already decided $1-w$. Thus, at y processes i and j have (irrevocably) decided on different values. Let P be a prefix of this run that ends in y, and let $L \in \mathsf{L}^{rw}$ be any full layer. The run $R' = P \odot L^\infty$ is a run of $\mathcal{S}_{\mathsf{L}}(D)$ in which i decides w, process j decides $1-w$, and both are nonfaulty (they move infinitely often). Processes i and j similarly decide w and $1-w$ respectively in the run $\hat{R} \in \mathcal{S}(D)$ corresponding to R', contradicting the assumption that D satisfies the agreement property. □

The following lemma is the basis for the impossibility proof.

LEMMA 3.8. *Let X be a potence connected (with respect to $\mathcal{S}_{\mathsf{L}}(D)$) set of states of $\mathcal{S}_{\mathsf{L}}(D)$. If X contains both 0-potent and 1-potent states, then there is a bipotent state in X.*

Proof. Let X^0 be the subset of X consisting of 0-potent states, while X^1 is the subset of 1-potent states. By assumption, both subsets are nonempty. It thus follows that there must be an edge $x^0 \sim_p x^1$ with $x^0 \in X^0$ and $x^1 \in X^1$ (otherwise X is not potence connected).

Since $x^0 \sim_p x^1$, the states x^0 and x^1 have a shared potence. If their shared potence is 0 then x^1 is bipotent, while if the shared potence is 1 then x^0 is bipotent. In either case there is a bipotent state in X, as desired. □

We can use Lemma 3.6 to obtain a bipotent initial state x^0, a well known result of [19].

LEMMA 3.9. *If D satisfies the decision and validity properties, then the set Con_0 contains a bipotent state with respect to $\mathcal{S}_{\mathsf{L}}(D)$.*

Proof. Let $x^0 \in \mathsf{Con}_0$ be the initial state in which all initial values are 0, and let x^1 be the corresponding state with all values 1. By the validity condition in every admissible run of D starting in the state x^0 (resp. x^1) the nonfaulty processes decide 0 (resp. 1). Since L^{rw}-runs correspond to admissible runs, and there is an L^{rw}-run starting in x^0 and an L^{rw}-run starting in x^1, x^0 is 0-potent and x^1 is 1-potent. By Lemma 3.6 Con_0 is potence connected and thus by Lemma 3.8 we obtain that there is a bipotent state in Con_0. □

LEMMA 3.10. *Every protocol D that satisfies decision and validity has a bipotent run in $\mathcal{S}_{\mathsf{L}}(D)$.*

Proof. By Lemma 3.9, there is a bipotent state, say x^0, in Con_0. We will construct a sequence of bipotent states $x^1, x^2, \ldots, x^k, \ldots$ and a corresponding sequence of layers $L_1, L_2, \ldots, L_k, \ldots$ such that $x^{i+1} = x^i \cdot L_{i+1}$ for all $i \geq 0$. The desired run will be

$$R_{\mathsf{L}} = x^0 \odot L_1 \odot L_2 \odot \cdots \odot L_k \odot \cdots$$

It remains to define the two sequences. We will define the layers L_i and the states x^i by induction on i. Notice that x^0 is bipotent. Let $k \geq 0$, and assume that we have constructed sequences $L_1, L_2, \ldots, L_k$ and $x^1, x^2, \ldots, x^k$ with the desired properties. In particular, x^k is a bipotent state. Since D satisfies decision, we have by Lemma 3.5 that $\mathsf{L}^{rw}(x^k)$ is potence connected. Since x^k is bipotent, $\mathsf{L}^{rw}(x^k)$ contains both 0-potent and 1-potent states. It follows from Lemma 3.8 that there is a bipotent state $y \in \mathsf{L}^{rw}(x^k)$. Since $y \in \mathsf{L}^{rw}(x^k)$ we have by the definition of $\mathsf{L}^{rw}(x^k)$ that $y = x^k \cdot L$ for some layer L. Set $L_{k+1} = L$ and $x^{k+1} = y$. □

We conclude

THEOREM 3.11. *There is no 1-resilient consensus protocol in the asynchronous shared memory model.*

Proof. We need to show that no protocol D can satisfy all three properties of consensus: decision, validity and agreement. Assume that D satisfies decision and validity. By Lemma 3.10 there is a bipotent run $R \in \mathcal{S}_{\mathsf{L}}(D)$. This run has an infinite number of bipotent states. If D were to satisfy agreement as well, we have by Lemma 3.7 that no process could have decided in a bipotent state. But then no process would ever decide in the run R, and hence no process would ever decide in the corresponding run of $\mathcal{S}(D)$ contradicting the assumption that D satisfies the decision property. □

3.4. Discussion. contributions) In this section we have used layering to provide an alternative proof of the impossibility of consensus in the asynchronous shared-memory model. Our proof differs from those of [19, 27] in a number of ways. First, it does not depend on a "critical state" argument; rather, it constructs a bipotent run inductively one state (or rather one layer) at a time. A subtle aspect of the standard proofs is proving that prefixes of a run can be extended into full runs in a manner that satisfies the fairness (or admissibility) conditions. In our case this is simplified by having each layer contribute "enough" to the fairness conditions to guarantee that any run constructed as a sequence of layers is admissible. We have attempted to provide a proof that makes fairly limited and local use of the particular properties of the model. As a result, as we shall see in the sequel, the same proof outline is applicable to the analysis of consensus in other models. Thus, we believe

that the notions of potence connectivity and layering capture some of the topological structure underlying the consensus problem (and more generally 1-resilient solvability, as described in [33]). For example, the layering approach can be used to prove global connectivity properties of a model, as opposed to the approach described in this section, that shows connectivity of successors of a state; more about this is in Section 9.

We next provide a more general framework and show how the notions of potence and layering and the proof outline just given can be applied more broadly. We start by describing how to model different types of distributed systems in a general fashion.

4. Layering Consensus in General Models. We now consider how layering can be used to analyze consensus for a variety of models. We start by defining models of distributed systems in a more general manner, and show how layering can be applied to consensus in such generic models.

4.1. Models of distributed computation. Using the notions of Section 2.2, we define a generic model of computation. A *model of distributed computation* is determined by sets L_i, $i \in \{e, 1, 2, \ldots, n\}$ of local states for the processes and the environment, and corresponding sets of actions ACT_i, for every $i \in \{e, 1, 2, \ldots, n\}$, and by a tuple $M = (\mathcal{G}_0, P_e, \tau, \Psi, \mathsf{FGen})$, where

- $\mathcal{G}_0 \subseteq \mathcal{G}$ is called the set of *initial states*. The identity of $\mathcal{G}_0$ will depend on the type of analysis for which the model is introduced. When we focus on a particular problem such as consensus, $\mathcal{G}_0$ is the set Con_0 of initial states for consensus,
- P_e is a (nondeterministic) protocol for the environment,
- τ is a transition function,
- Ψ is a set of runs over $\mathcal{G}$ and ACT_e, such that for every pair of runs R and R' that have a suffix in common, $R \in \Psi$ if and only if $R' \in \Psi$. The set Ψ is called the set of *admissible* runs in the model. This is a tool for specifying fairness properties of the model. For example, properties such as "every message sent is eventually delivered" or "every process moves infinitely often" are enforced by allowing as admissible only runs in which these properties hold. The condition we have on Ψ being determined by the suffixes of its runs ensures that admissibility depends only on the infinitary behavior of the run. Finally,
- FGen is a function that, for each protocol D gives a predicate Faulty_D defined on the runs of D in M (defined below). We remark that the dependence of the Faulty_D on the protocol is useful when we want to capture the idea that a process is faulty if it deviates from the protocol it is supposed to follow. This is relevant for handling malicious failures, for example.

We say that a run $R = (r, \alpha)$ is a **run of the protocol** $D = (D_1, \ldots, D_n)$ **in** $M = (\mathcal{G}_0, P_e, \tau, \Psi, \mathsf{FGen})$ in case

(i) $r(0) \in \mathcal{G}_0$, so that R begins in a legal initial state according to M,

(ii) $\alpha(k) \in P_e(r(k)_e)$ for all k,

(iii) $r(k+1) = \tau(r(k), (\alpha(k), \mathbf{a}^k))$ holds for all $k \geq 0$, where the domain of $\mathbf{a}^k$ is the set Sched in $\alpha(k)$, and $\mathbf{a}_i^k = D_i(r(k)_i)$ for every $i \in \mathsf{Sched}$,[5] and

(iv) $R \in \Psi$, so that R is admissible.

[5] Given this choice, any deviations of a process from the protocol, as may happen in a model with malicious failures, will need to be modeled as resulting from the environment's actions. The behavior of faulty processes in such case will be controlled by the environment.

Condition (ii) implies that the environment's action at every state of R is legal according to its protocol P_e, and condition (iii) states that the state transitions in R are according to the transition function τ, assuming that the joint action is the one determined by the environment's action and the actions that the protocol D specifies for the processes that are scheduled to move. A run satisfying properties (i)–(iii) but not necessarily the admissibility condition (iv) is called *a run of D consistent with M*. It is a run in which the initial state and local transitions are according to D and M, but the admissibility conditions imposed by Ψ are not necessarily satisfied. We will find it useful to consider such runs in Section 6.

The notions of models and protocols give us a way of focusing on a special class of systems, resulting from the execution of a given protocol in a particular model. We denote by $\mathcal{S}(D, M, I)$ the system $(\mathcal{R}, \mathsf{Faulty})$, where $\mathcal{R}$ is the set of all runs of protocol D in the model M that start in initial states from a set I, where $I \subseteq \mathcal{G}_0$, and $\mathsf{Faulty} = \mathsf{FGen}^M(D)$.

We say that a system $\mathcal{S}$ satisfies the *pasting property* if, for every pair $R = (r, \alpha)$ and $R' = (r', \alpha')$ of runs of $\mathcal{S}$ such that $r(m) = r'(m')$ for some integers m, m', there is a run R'' of $\mathcal{S}$ such that $R''[0, m] = R[0, m]$ and $R''[m, \infty] = R'[m', \infty]$. Intuitively, the pasting property says that we can "paste" any prefix ending in a state x with a suffix starting in x, and obtain a run of $\mathcal{S}$. In a sense, this means that all of the information that is relevant to determining the future of a state is included in the state.

It is straightforward to show:

LEMMA 4.1. *Every system of the form $\mathcal{S} = \mathcal{S}(D, M, I)$ has the pasting property.*

Proof. Assume that $R = (r, \alpha)$ and $R' = (r', \alpha)$ are runs of $\mathcal{S}$, and that $r(m) = r'(m')$, and let R'' be defined as in the definition of the pasting property. We need to show that R'' satisfies conditions (i)-(iv) above. For condition (i), $r''(0) = r(0)$ and $r(0) \in \mathcal{G}_0$ since R satisfies condition (i). For $k < m$, we have that $r''(k) = r(k)$, $\alpha''(k) = \alpha(k)$ and $r''(k+1) = r(k+1)$. Therefore properties (ii) and (iii) follow for $k < m$ from the fact that they hold for R. Similarly, for $k \geq m$, $r''(k) = r'(m'+k-m)$, $\alpha''(k) = \alpha'(m'+k-m)$ and $r''(k+1) = r'(m'+k+1-m)$ so that (ii) and (iii) for these values of k follow from the fact that they hold for the run R' (at time $m'+k-m$). Finally, $R''[m, \infty] = R'[m', \infty]$, so that R'' and R' have a suffix in common. Since $R' \in \Psi$, it thus follows that $R'' \in \Psi$ so R'' satisfies (iv) and we are done. □

Finally, recall that the definition of consensus depends in an essential way on the behavior of the nonfaulty processes. Since we have a very general notion of a Faulty predicate that may depend on the model, we will restrict attention to cases in which the notion of failures is not completely out of hand. A predicate Faulty defined for the runs of a system $\mathcal{S}$ induces a notion of a process being *failed at a state* (with respect to $\mathcal{S}$). We say that a process i is *failed at state x* if i is faulty in all runs of $\mathcal{S}$ containing x. Otherwise i is *non-failed* at x.

DEFINITION 4.2. *A system $\mathcal{S} = (\mathcal{R}, \mathsf{Faulty})$ satisfies* **fault independence** *if*

(i) For every state x of $\mathcal{S}$ there is a run $R^x \in \mathcal{S}$ in which x appears, such that the only processes that fail in R^x are those that are already failed at x,

(ii) no process is failed at an initial state of $\mathcal{S}$,

(iii) if R and R' have a common suffix, then the same processes fail in both runs, and

(iv) at most $n-1$ processes fail in any run $R \in \mathcal{S}$.

Part (i) is formally captured by the condition

$$(\forall x \text{ of } \mathcal{S})(\exists R^x \in \mathcal{S})\forall i\,[\mathsf{Faulty}(i, R^x) \text{ iff } i \text{ is failed at } x].$$

The intuition here is that every instance of faulty behavior should be the result of the failure of some component in the system. If there is an extension of a state in which one component fails and another does not, and there is a different execution where the second component fails but the first one does not, then there should be a third execution where neither one fails. Failures are thus independent in this sense. Part (ii) implies that for any initial state x, and every process i, there is a run R^x containing x where i is non-faulty. (Of course there may be another such run where i is faulty.) Part (iii) is included because the failures we are interested in (e.g. crashes) are determined by the infinite part (i.e. suffix) of a run. Notice that there are failure models in which a failure can be committed at a given point in time, and not be reflected in the processes' states at a later time. This could cause two runs to have the same suffix while a particular process behaves in a faulty manner in one of the runs but not in the other. However, by appropriately modeling the environment's state to keep track of such failures, part (iii) can be guaranteed in such models as well. Part (iv) allows us to concentrate on runs where at least one process did not fail.

All systems we will consider are assumed to satisfy fault independence.

4.2. Example: The Mobile Failures Model. We illustrate the use of the abstract framework just described by describing a synchronous model with a *single mobile failure* [35]. The model is the standard synchronous message-passing model in which communication proceeds in lockstep rounds, in every round each process can send a message to each of the other processes, and messages are delivered in the round they are sent with no corruptions. The failure assumption is that in every round m there can be at most one process i_m some of whose messages are lost. The set of messages lost by this process in the round in question is arbitrary. We use the term *mobile failure* because the identity of the process whose messages may be lost can change from one round to the next.

We now sketch how this model, which we denote by $M^{\mathsf{m}} = (\mathcal{G}_0^{\mathsf{m}}, P_e^{\mathsf{m}}, \tau^{\mathsf{m}}, \Psi^{\mathsf{m}}, \mathsf{FGen}^{\mathsf{m}})$, can be represented in terms of our formalism. The environment's state is assumed to be the same in all states of $\mathcal{G}_0^{\mathsf{m}}$. The environment's protocol P_e^{m} is uniform at all states: nondeterministically choose a process $i \in \{1, \ldots, n\}$ and a set $T \subseteq \{1, \ldots, n\}$ and perform the action $(\{1, \ldots, n\}, (i, T))$. Notice that in all cases $\mathsf{Sched} = \{1, \ldots, n\}$ so that, intuitively, all processes are scheduled to move in every round. The action $a = (i, T)$ specifies that any message sent from i to members of T will be lost. The actions a process can perform in this model specify a list of at most one message to be sent to each process in the current round. Thus, given a protocol $D = (D_1, \ldots, D_n)$ for the processes, for every state x the protocol D_i for i defines an action $D_i(x_i)$ that specifies what is its next state, and the messages that i sends in that state.

Given a state x and a joint action $\bar{\mathsf{a}} = (a_e, \mathbf{a})$, $a_e = (\{1, \ldots, n\}, (j_0, T_0))$, the transition function τ^{m} updates the local state of a process i as a function of its current action, $D_i(x_i)$, and the list of messages that are sent to it in the current round that are not blocked (i.e., the list of messages sent to i by processes j according to $D_j(x_j)$ except that if $i \in T_0$ we ignore any message that may be sent by $j = j_0$).

The set Ψ^{m} makes no restrictions whatsoever: it consists of all possible runs consistent with M^{m}. Finally, $\mathsf{FGen}^{\mathsf{m}}(D)(i, R) = \mathit{false}$ for all processes i, runs R and

protocols D; in this model, no process is ever considered faulty.[6]

Note that M^{m} satisfies fault independence. Namely, consider any system $\mathcal{S} = \mathcal{S}(D, M^{\mathsf{m}}, I)$. All properties of Definition 4.2 hold trivially for $\mathcal{S}$, because no processes are ever considered faulty in M^{m}.

5. Abstract Impossibility Framework for Consensus. We are now ready to initiate a general model-independent analysis of the consensus problem. We will attempt to show that the general structure of the proof we presented in Section 3 is widely applicable. As in Section 2.1, we will assume a uniform set Con_0 of initial states for consensus. The local state of every process i in a state x of Con_0 consists of two distinct variables, $\mathtt{v}_i$ and $\mathtt{d}_i$. The first has a binary value and is considered i's *initial value* for the purpose of the consensus procedure. The second is a write-once variable that appears in all of i's local states, and is initially undefined (i.e., initially $\mathtt{d}_i = \perp$). The environment's state is assumed to be the same in all states of Con_0.

In the sequel, we shall focus on systems that are compatible with the consensus problem. We call a system $\mathcal{S} = \mathcal{S}(D, M, I)$ a *system for consensus* if (i) the set I of initial states in $\mathcal{S}$ is Con_0, (ii) the local state x_i of every process i in each state x of $\mathcal{S}$ contains the variable $\mathtt{d}_i$, and in all runs the variable $\mathtt{d}_i$ is write-once. A model $M = (\mathcal{G}_0, P_e, \tau, \Psi, \mathsf{FGen})$ is a *model for consensus* if $\mathsf{Con}_0 \subseteq \mathcal{G}_0$ and every system of the form $S(D, M, \mathsf{Con}_0)$ is a system for consensus.

All Models are models for consensus satisfying fault independence.

The consensus problem in Section 2.1 is now well-defined with respect to a general model M: a protocol D solves consensus if all the runs of the system $\mathcal{S}(D, M, \mathsf{Con}_0)$ satisfy agreement, decision and validity, where faulty processes are defined according to the function $\mathsf{Faulty}_D = \mathsf{FGen}(D)$.

5.1. Potence and bipotence in general models. Recall the definitions of w-potence and bipotence with respect to a system $\mathcal{S}$ of Section 3.2. In particular, they hold for $\mathcal{S} = \mathcal{S}(D, M, I)$. In this section we consider two useful ways to show that $x \sim_p y$ in general models. We start with some specific conditions under which analogues to Lemma 3.4 holds.

LEMMA 5.1. *Assume that $\mathcal{S}$ satisfies decision and let x, y, z be states of $\mathcal{S}$.*

(i) For $v \in \{0, 1\}$, if z is v-potent and z extends y, then y is also v-potent.

(ii) If z extends both x and y then $x \sim_p y$.

Proof. For part (i), assume z is v-potent. Then there exists a run $R = (r, \alpha)$ of $\mathcal{S}$ and a time $m \geq 0$ such that $r(m) = z$ and some nonfaulty process i in R decides v. Since z extends y, there is a run $R' = (r', \alpha')$ and there are times $k' \geq k \geq 0$ such that $r'(k) = y$ and $r'(k') = z$, in which the only processes that fail are those that are already failed at z (fault independence (i) and (iii)). Consider the run R'' obtained by pasting the prefix $R'[0, k']$ with the suffix $R[m, \infty]$. Since the system $\mathcal{S}$ satisfies the pasting property (by Lemma 4.1) R'' is a run of $\mathcal{S}$. Process i reaches the same decisions in R and in R'' (decisions are write-once and the $\mathtt{d}_i$ variable appears in all of i's local states). Since R and R'' have a common suffix, the same processes fail in both runs (fault independence (iii)). It follows that i is a nonfaulty process deciding v in R''. The state y is therefore v-potent and we are done.

[6] This assumption is made for simplicity. Notice that the process i_m some of whose messages may be lost in a given round m still receives all messages sent to it. This process should therefore not be at a disadvantage when it comes to being able to decide on the consensus value. The same analysis and conclusions as we present here would hold if, for example, we would assume that a process that is silenced from some point on in a given run is considered faulty in that run.

We now show part (ii). Since $\mathcal{S}$ satisfies the decision property, every state of $\mathcal{S}$, including z in particular, is either 0-potent or 1-potent (or both). Assume that z is v-potent. By part (i) y is v-potent, and by the same argument x is v-potent as well. Hence $x \sim_p y$ and we are done. □

Recall that a run is bipotent if all of its states are bipotent. An important consequence of the agreement property is that a consensus protocol cannot terminate while in a bipotent state. As a result, if a protocol has a bipotent run, then it cannot solve consensus. This is an important feature underlying impossibility proofs for consensus. We now capture these claims as they apply to general models more formally. One feature of many popular models, including the common asynchronous ones, is that the failure of a process is not determined in finite time. We say that a system $\mathcal{S}$ displays *no finite failure* if, for all states x of $\mathcal{S}$, no process is failed at x. Namely, for every process i, there is a run containing x in which i is nonfaulty. For such systems we have the following lemma, which is a slight generalization of Lemma 3.7 and whose proof has the same structure.

LEMMA 5.2. *Let $\mathcal{S}$ satisfy the agreement requirement and assume there is a bipotent run $R^b \in \mathcal{S}$. If $\mathcal{S}$ displays no finite failure then $\mathcal{S}$ does not satisfy the decision property.*

Proof. We will show that if a state x of $\mathcal{S}$ is bipotent, then no process has decided by x. The claim follows, since in a bipotent run no process will ever decide, and we have by the fault independence assumption part (iv) that there must be at least one nonfaulty process in R^b.

Assume by way of contradiction that x is bipotent and i is decided at x. Let its decision at x be $\mathtt{d}_i = w \neq \perp$. Since x is bipotent, there is a run R containing x in which some process, say j, decides $1 - w$. It follows that there is in R a state y extending x in which $\mathtt{d}_j = 1 - w$. Since $\mathcal{S}$ displays no finite failure, both i and j are non-failed at y. By the fault independence assumption part (i), there is a run R' containing y in which both i and j are nonfaulty. Since $\mathtt{d}_i$ and $\mathtt{d}_j$ are write-once, however, in R' process i decides w while j decides $1-w$, contradicting the assumption that $\mathcal{S}$ satisfies the agreement property. □

Lemma 5.2 clearly demonstrates that consensus cannot be attained at a bipotent state of a system that displays no finite failure. The following lemma shows that a consensus protocol is still unable to terminate in a bipotent state even in systems in which failures can be observed in finite time.

LEMMA 5.3. *Let $\mathcal{S}$ be a system satisfying the agreement requirement and assume that no more than $t < n$ processes fail in any run of $\mathcal{S}$. If x is a bipotent state of $\mathcal{S}$ then at least $n - t$ non-failed processes at x have not decided by x.*

Proof. Since x is bipotent, there is a run R^0 containing x in which at least one nonfaulty process decides 0. The set P_0 of nonfaulty processes in R^0 consists of at least $n - t$ processes. They are all non-failed at x, and by the agreement property none of them has decided 1 by x. By symmetry of 0 and 1, we obtain the existence of a set P_1 of at least $n - t$ non-failed processes at x that have not decided 0 by x. By the fault independence property part (i), there is a run $\hat{R}$ containing x in which the only processes that fail are the ones that are already failed at x. All processes in $P_0 \cup P_1$ are nonfaulty in $\hat{R}$. By the agreement property, there is at most one value $w \in \{0, 1\}$ on which nonfaulty processes decide in $\hat{R}$. If nonfaulty processes do not decide 0 in $\hat{R}$, then no process in P_0 has decided by x, and if they do not decide 1 in $\hat{R}$, then no process in P_1 has decided by x. In either case, the claim holds. □

5.2. Potence connectivity revisited. The central role played by connectivity in the analysis of consensus and decision problems in general has been observed by many authors starting with [18]. The traditional notion of connectivity in the literature ([18, 29, 35]) is based on comparing the local states of processes in the *current* (global) state: Two states are similar if they share enough structure (e.g., equal environment and process' states), and the transitive closure of this binary relation provides a corresponding notion of (similarity-based) connectivity. In contrast, the shared potence of states depends on their possible future extensions, and hence so is potence connectivity. Clearly, a notion of the first kind is independent of the protocol used, while potence connectivity is protocol dependent. It is our view that potence connectivity plays a crucial role in the structure of consensus. In addition to generalizing our treatment of potence connectivity slightly, in this section we will draw a formal connection between similarity connectivity and potence connectivity. Intuitively, we will show that in models in which the environment can always silence an arbitrary process, similarity connectivity yields potence connectivity. Similarity-based connectivity will thus prove to be a useful tool for showing potence connectivity.

Similarity connectivity + crash-like behavior => potence connectivity. Many failure models considered in the study of fault tolerance allow faulty behavior in which the state of a process is "hidden" from some point on. Usually this happens as a result of a process crash, but it can also be the result of the process's memory being erased, for example. When such hiding can occur, states that differ only in the local state of one process will often have a shared potence. It follows that there is a connection between similarity of states and potence connectivity. We now formalize this connection.

The state of the environment is often best described as a tuple of distinct components, each accounting for a separate aspect of the system. In some models, part of the components of the state of the environment can affect only the state of a single process. For example, in a shared-memory model, if there is a variable that can be read *only* by process j, then it can affect only j. A similar situation occurs in the message-passing model, when a channel contains messages that were all sent to j. We call such components *j-components*. Clearly, in some models the environment state has no j-components. When they do exist, the following notions depend on a clear definition of these components. Two environment states are said to *agree modulo j* if they are the same except, possibly, for their j-components. We say that two states x and y **agree modulo** j, if $x_i = y_i$ for all $i \neq j$ and their environment states x_e and y_e agree modulo j. The intuition is that if process j can somehow be "silenced" or if its local state can be "hidden" and not observed by others in the future, runs resulting from both states will be the same from the point of view of the other processes. In models in which there are no j components to the environment's state, states x and y that agree modulo j differ at most in the identity of process j's local state.

DEFINITION 5.4 (Similarity). *With respect to a system $\mathcal{S}$*

(i) States x and y are **similar**, *denoted by $x \sim_s y$, if there is a process j such that (a) the states x and y agree modulo j, and (b) there exists $i \neq j$ that is non-failed in both x and y.*

(ii) A set of states X is **similarity connected** *if the graph $(X, \sim_s)$ induced by $\sim_s$ on X is connected.*

A well-known and useful property of the initial states for consensus, Con_0, is given by:

LEMMA 5.5. *The set Con_0 is similarity connected.*

Proof. To prove that Con_0 is similarity connected we will show that every two initial states $x, y \in \mathsf{Con}_0$ are connected by a sequence of states in which each pair of neighbors are similar. Choose $x, y \in \mathsf{Con}_0$. For $0 \leq l \leq n$, define x^l by setting

$$x^l_j = \begin{cases} y_j & \text{for all } j \leq l; \text{ and} \\ x_j & \text{for all } j > l. \end{cases}$$

(Recall that $x_e = y_e$ by definition of Con_0.) Clearly, $x^l \in \mathsf{Con}_0$, and it is easy to check that $x^0 = x$ and $x^n = y$. Moreover, for every $0 < l \leq n$ we have that x^{l-1} and x^l agree modulo l, since the local states of the environment and of all processes, except possibly that of l, are equal. The assumption that no process is failed in an initial state (fault independence (ii)), and the fact that $n \geq 2$, imply that there is a process $i \neq l$ that is non-failed in both x^{l-1} and x^l, and hence these two states are similar. □

We now formalize the intuition that similarity connectedness yields potence connectedness when processes may crash.

DEFINITION 5.6 (crash-like behavior). *Let X be a set of states of the system $\mathcal{S}$. We say that $\mathcal{S}$* **displays crash-like behavior** *with respect to X if the following condition holds. For every $x, y \in X$ and process j, if x and y agree modulo j, then there exist in $\mathcal{S}$ runs R^x and R^y and times $m_x, m_y \geq 0$ such that*

(i) $r^x(m_x) = x$ and $r^y(m_y) = y$,

(ii) $r^x(m_x + k)$ and $r^y(m_y + k)$ agree modulo j for all $k \geq 0$, and

(iii) Every process $i \neq j$ that is not failed in x and in y is nonfaulty in R^x and in R^y.

It is worth noting that the abstract definition of crash-like behavior is not restricted to crash failures. What the definition requires is that the state of j may be "hidden" from the rest of the processes indefinitely from some point. This can happen in models of process failures such as crash, omission, or Byzantine failures. It can also happen in models of link failures [35] or even in some cases when a failure may simply change the local state of a process, thereby effectively corrupting or erasing its memory.

A very useful relation between potence connectedness and similarity connectedness is given by the following lemma.

LEMMA 5.7. *Let $\mathcal{S}$ be a system satisfying the decision property, and let X be a similarity connected set of states of $\mathcal{S}$. If $\mathcal{S}$ displays crash-like behavior with respect to X, then X is potence connected.*

Proof. Since similarity connectedness is the transitive closure of $\sim_s$ and potence connectedness is the transitive closure of $\sim_p$, it suffices to show that for all $x, y \in X$, if $x \sim_s y$ then $x \sim_p y$. Assume $x \sim_s y$ and $\mathcal{S}$ displays crash-like behavior with respect to $X \supseteq \{x, y\}$. Then there exists a process j such that (i) the states x and y agree modulo j, and (ii) there exists a process $i \neq j$ that is non-failed in both x and y. Since $\mathcal{S}$ displays crash-like behavior with respect to $\{x, y\}$, part (i) implies that there exist in $\mathcal{S}$ runs R^x and R^y and times $m_x, m_y \geq 0$ such that (a) $r^x(m_x) = x$ and $r^y(m_y) = y$, (b) $r^x(m_x + k)$ and $r^y(m_y + k)$ agree modulo j for all $k \geq 0$, and (c) Every process $i \neq j$ that is nonfaulty in x and in y is nonfaulty in R^x and in R^y. Moreover, by (ii), there is at least one such nonfaulty process i. Since $\mathcal{S}$ satisfies the decision requirement, process i eventually decides in both runs. Let w be the value that i decides in R^x. Because $\mathtt{d}_i$ is write-once and appears in all local states of i, and since $r^x(m_x + k)_i = r^y(m_y + k)_i$ for all $k \geq 0$, we conclude that process i decides on w in R^y as well. It follows that both x and y are w-potent, and hence $x \sim_p y$. □

Notice that Lemma 3.8 holds also for a general model M. Also, recall from the

proof of Lemma 3.9 that when decision and validity hold, the initial state with all initial values 0 is 0-potent, while the one with all values 1 is 1-potent; this holds in the generic setting due to fault independence. Thus, an immediate consequence of Lemmas 3.8, 5.5 and 5.7 is a generalization of the well-known fact from [19] that when even a single process can crash, there must be a bipotent initial state for consensus.

THEOREM 5.8. *Let $\mathcal{S}$ be a system for consensus satisfying the decision and validity conditions. If $\mathcal{S}$ displays crash-like behavior with respect to Con_0, then there is a bipotent initial state in $\mathcal{S}$.*

6. Layering. In Section 3.1 we defined a set of layers to facilitate the analysis of particular well-behaved runs of the shared-memory model. We now consider a similar operation for general models. Recall that given a finite execution R we denote by $R \odot \epsilon$ the execution that results from extending R by having the environment perform $\epsilon = (\mathsf{Sched}, a)$ in its final state. A run of $\mathcal{S} = \mathcal{S}(D, M, I)$ can thus be represented in the form $x \odot \epsilon_0 \odot \epsilon_1 \odot \cdots$ where $x \in I$ and $\epsilon_i \in \mathsf{ACT}_e$ for $i \geq 0$.

In some cases we are interested in single actions of the environment, while in others we may be interested in thinking of sequences of such actions as constituting a "round" or "layer" of the computation. In such cases, we will be interested only in those states that appear at the end of layers, and want to ignore the intermediate states. Given a model $M = (\mathcal{G}_0, P_e, \tau, \Psi, \mathsf{FGen})$, we define a **layer** over ACT_e to be a nonempty finite sequence $\ell = \epsilon_1, \epsilon_2, \ldots, \epsilon_k$ of actions of the environment, $\epsilon_i \in \mathsf{ACT}_e$. If a protocol D is specified, given a state x, the round ℓ would lead from x to the state at the end of $x \odot \epsilon_1 \odot \epsilon_2 \odot \cdots \odot \epsilon_k = x \odot \ell$, provided each ϵ_i is an action that can be executed according to the environment protocol in the state at the end of $x \odot \epsilon_1 \odot \epsilon_2 \odot \cdots \odot \epsilon_{i-1}$. In this case we say that ℓ is *executable* at x. We denote the state at the end of the execution $x \odot \ell$ by $x \cdot \ell$.

One of the principles behind the use of layers and layerings is that we would like to ignore small steps and intermediate states, and center our attention on interesting landmarks in the computation. Given a set of layers L, we define an L-*run* to be a pair $R_{\mathsf{L}} = (r_{\mathsf{L}}, \alpha_{\mathsf{L}})$, where $r_{\mathsf{L}} : N \to \mathcal{G}$ defines an infinite sequence of states of $\mathcal{G}$, and $\alpha_{\mathsf{L}} : N \to \mathsf{L}$ is a sequence of layers. We will be interested in L-runs that describe runs of a system $\mathcal{S} = \mathcal{S}(D, M, I)$. We say that a run $R = (r, \alpha)$ of $\mathcal{S}$ *corresponds to* an L-run $R_{\mathsf{L}} = (r_{\mathsf{L}}, \alpha_{\mathsf{L}})$ if there is an infinite sequence $i_0 < i_1 < \cdots < i_k < \ldots$, such that $i_0 = 0$ and for all $k \geq 0$ we have both (a) $r(i_k) = r_{\mathsf{L}}(k)$ and (b) the sequence $\alpha(i_k), \alpha(i_k + 1), \ldots, \alpha(i_{k+1} - 1)$ is exactly the layer $\alpha_{\mathsf{L}}(k)$.

Intuitively, an L-run is a run that is obtained by starting at some initial state and repeatedly performing layers of L. The run R_{L} keeps track of the specific sequence of layers used, and of the states at the end of layers. Thus, the view presented in R_{L} is that the actions that the environment performs are in the form of whole layers. It is easy to check that there can be at most one run R corresponding to a given R_{L} in the system $\mathcal{S}(D, M, I)$. Suppose that $r_{\mathsf{L}}(0) \in I$ and for every $m \geq 0$ there is an execution $x^m \odot \ell_m$ in $\mathcal{S} = \mathcal{S}(D, M, I)$, where $x^m = r_{\mathsf{L}}(m)$ and $\ell_m = \alpha_{\mathsf{L}}(m)$. In this case, there is a unique run R of D consistent with M that corresponds to R_{L}. Notice, however, that R might still not be a run of D in M, because it may fail to satisfy the fairness condition Ψ of M. We will be interested in sets L of layers in which this cannot happen:

DEFINITION 6.1. *A nonempty set L of layers is called a* **layering** *of a model M if, for every protocol D, the following condition holds. Every run R of D consistent with M that corresponds to an L-run R_{L} is a run of D in M.*

Notice that for every set of layers, L, we can consider the runs of a protocol D in

which the environment performs actions according to the layers in this set. A set of layers is a layering if it satisfies the property that *every* run obtained by performing an infinite sequence of layers of L must satisfy the fairness conditions imposed by M, as specified by its admissibility condition Ψ. Roughly speaking, then, each layer must carry out a sufficient amount of work to guarantee that the fairness requirements imposed by Ψ are ultimately satisfied.

Given a system $\mathcal{S} = \mathcal{S}(D, M, I)$ and a layering L of M, we define the corresponding layered system $\mathcal{S}_{\mathsf{L}} = \mathcal{S}_{\mathsf{L}}(D, M, I)$ by

$$\mathcal{S}_{\mathsf{L}} = \{R_{\mathsf{L}} \mid \text{there is a run } R \text{ of } \mathcal{S} \text{ that corresponds to } R_{\mathsf{L}}\},$$

with Faulty predicate as follows. Since every run R_{L} of $\mathcal{S}_{\mathsf{L}}$ has associated a unique run R of the original system $\mathcal{S}$, the Faulty_D predicate from M can be extended to the runs of $\mathcal{S}_{\mathsf{L}}$ by defining $\mathsf{Faulty}_D(R_{\mathsf{L}}) = \mathsf{Faulty}_D(R)$ for all runs $R_{\mathsf{L}} \in \mathcal{S}_{\mathsf{L}}$. Thus, to figure out who the faulty processes in a run of $\mathcal{S}_{\mathsf{L}}$ are, we check the corresponding run of $\mathcal{S}$. Notice, however, that fault independence of $\mathcal{S}$ does not necessarily imply fault independence of $\mathcal{S}_{\mathsf{L}}$.

DEFINITION 6.2. *A* **layering** L *of a model* M **satisfies fault independence** *if every system of the form* $\mathcal{S}_{\mathsf{L}}(D, M, I)$ *satisfies fault independence.*

There are several obvious close correspondences between $\mathcal{S}_{\mathsf{L}}$ and $\mathcal{S}$, that follow directly from the definition of $\mathcal{S}_{\mathsf{L}}$ in terms of the runs of $\mathcal{S}$. First of all, $\mathcal{S}_{\mathsf{L}}$ inherits a number of "universal" properties from $\mathcal{S}$: Each of the properties of decision, agreement, and validity is satisfied by $\mathcal{S}_{\mathsf{L}}$ if it is satisfied by $\mathcal{S}$. In the other direction, "existential" properties of $\mathcal{S}_{\mathsf{L}}$ pass on to $\mathcal{S}$: If a state x is bipotent with respect to $\mathcal{S}_{\mathsf{L}}$ then it is bipotent with respect to $\mathcal{S}$. Moreover, if R_{L} is a bipotent run of $\mathcal{S}_{\mathsf{L}}$, then the run R of $\mathcal{S}$ that corresponds to R_{L} is a bipotent run with respect to $\mathcal{S}$.

Our use of layering to prove lower bound and impossibility results for consensus will be based on the close correspondence between $\mathcal{S}_{\mathsf{L}}$ and $\mathcal{S}$. The general idea is to assume there is a protocol D solving consensus in some particular model M. An appropriate layering L for M is then defined, for which it can be shown that there is a bipotent run (or in the case of a lower bound a bipotent prefix of a run) in $\mathcal{S}_{\mathsf{L}} = \mathcal{S}_{\mathsf{L}}(D, M, \mathsf{Con}_0)$.

For every state x of $\mathcal{S}_{\mathsf{L}}$ we define $\mathsf{L}(x)$ to be the set of the successors of x in $\mathcal{S}_{\mathsf{L}}$. More formally,

$$\mathsf{L}(x) = \{x \cdot \ell \mid \ell \in \mathsf{L} \text{ and } x \odot \ell \text{ is an execution of } \mathcal{S}_{\mathsf{L}}\}.$$

Provided the layering L guarantees that $\mathsf{L}(x)$ is potence connected, we can use the following theorem to prove the existence of a bipotent run in $\mathcal{S} = \mathcal{S}(D, M, \mathsf{Con}_0)$. We can then apply Lemma 5.2 to show that the protocol D does not solve consensus.

THEOREM 6.3. *Let* L *be a layering of* M *satisfying fault independence, and let* $\mathcal{S} = \mathcal{S}(D, M, \mathsf{Con}_0)$ *be a system for consensus satisfying the decision condition. Consider the layered system* $\mathcal{S}_{\mathsf{L}} = \mathcal{S}_{\mathsf{L}}(D, M, \mathsf{Con}_0)$*, and assume that there is a bipotent initial state in* $\mathcal{S}_{\mathsf{L}}$*. If, for every state* x *of* $\mathcal{S}_{\mathsf{L}}$ *the set* $\mathsf{L}(x)$ *is potence connected in* $\mathcal{S}_{\mathsf{L}}$*, then there is a bipotent run in the original system* $\mathcal{S}$*.*

Proof. We will construct a bipotent L-run R^b_{L} in $\mathcal{S}_{\mathsf{L}}$, and the corresponding run R^b will be a bipotent run in $\mathcal{S}$. We obtain this run by starting from a bipotent initial state x^0 and constructing an infinite sequence of layers $\ell_0, \ell_1, \ldots$ from L, such that the state $x^m = x^0 \cdot \ell_0 \cdot \ldots \cdot \ell_{m-1}$ is bipotent w.r.t. $\mathcal{S}_{\mathsf{L}}$ for all $m \geq 0$. By construction, for all $m \geq 0$ we will have that $x^m \odot \ell_m$ is an execution in $\mathcal{S}_{\mathsf{L}}$, since it will be taken from $\mathsf{L}(x^m)$. This means that it is consistent with D and M to execute the sequence of

environment actions ℓ_m, and corresponding protocol actions starting in x^m, leading to x^{m+1}. Hence the run

$$R^b = x^0 \odot \ell_0 \odot \ell_1 \odot \ldots$$

will be a run of D consistent with M. The fact that L is a layering of M will then imply that R^b is a run of $\mathcal{S}(D, M, \mathsf{Con}_0)$. Recall that a bipotent state of $\mathcal{S}_\mathsf{L}$ is necessarily also a bipotent state of $\mathcal{S}$. Moreover, if a state of a run R is bipotent, then all states preceding x in R are also bipotent, by Lemma 5.1(i). Since there are infinitely many bipotent states in R^b (all the states x^m from the construction), it follows that all states of R^b are bipotent, so that R^b is a bipotent run of $\mathcal{S}$.

It remains to define x^0 and the sequence of layers ℓ_k. By assumption, there is a bipotent initial state in $\mathcal{S}_\mathsf{L}$. We shall choose this state to be x^0. Assume inductively we have chosen $\ell_0, \ldots, \ell_{m-1}$ so that $x^m = x^0 \cdot \ell_0 \cdot \ldots \cdot \ell_{m-1}$ is bipotent (with respect to $\mathcal{S}_\mathsf{L}$). Since x^m is bipotent, the set $\mathsf{L}(x^m)$ contains both 0-potent and 1-potent states. By assumption, $\mathsf{L}(x)$ is potence connected for every state x of $\mathcal{S}_\mathsf{L}$, and hence in particular $\mathsf{L}(x^m)$ is potence connected. It follows by Lemma 3.8 that there is a bipotent state $x' \in \mathsf{L}(x^m)$. By definition of $\mathsf{L}(x^m)$ there must also be a layer $\ell' \in \mathsf{L}$ such that $x' = x^m \cdot \ell'$. Set $\ell_m = \ell'$, and let $x^{m+1} = x^0 \cdot \ell_0 \cdot \ldots \cdot \ell_{m-1} \cdot \ell_m$. Clearly, $x^{m+1} = x^m \cdot \ell_m = x^m \cdot \ell' = x'$. It follows that x^{m+1} is bipotent, and we are done. □

We have essentially completed the description of our abstract framework for proving impossibility of consensus. Given a model M, the strategy would be to choose an appropriate layering L for M. With respect to an arbitrary protocol D, we use Theorem 5.8 to prove the existence of a bipotent initial state, and use Theorem 6.3 to extend this state into a bipotent run. Finally, from Lemmas 5.2 or 5.3 we conclude that this run is a counterexample to the decision property of consensus. We now apply this scheme to prove impossibility of consensus and lower bounds for models with synchronous message passing.

7. Synchronous Message Passing. We present two applications of the general framework to synchronous models: first for mobile failures, and then for the classic crash failure model.

7.1. Impossibility for mobile failures. We illustrate the use of the abstract framework just described by proving a new impossibility result for consensus in the presence of a single mobile failure in the synchronous model. This model, denoted $M^\mathsf{m} = (\mathcal{G}_0^\mathsf{m}, P_e^\mathsf{m}, \tau^\mathsf{m}, \Psi^\mathsf{m}, \mathsf{FGen}^\mathsf{m})$, is described in Section 4.2. Recall that M^m is the standard synchronous model with the failure assumption that in every round m there can be at most one process i_m some of whose messages are lost. The environment's protocol P_e^m is uniform in all states: nondeterministically choose a process $i \in \{1, \ldots, n\}$ and a set $T \subseteq \{1, \ldots, n\}$ and perform the action $(\{1, \ldots, n\}, (i, T))$, denoted simply by (i, T); all processes are scheduled to move in every round. The messages from i to members of T will be lost. Recall that Ψ^m makes no restrictions whatsoever: it consists of all possible runs consistent with M^m, and $\mathsf{FGen}^\mathsf{m}(D)(j, R) = \mathit{false}$ for all processes j, runs R and protocols D.

To get the following impossibility result in M^m, we assume for contradiction that there is a protocol D solving consensus, and prove three basic claims:

(i) There exists a layering L for M^m that satisfies fault independence,

(ii) $\mathcal{S}_\mathsf{L} = \mathcal{S}_\mathsf{L}(D, M^\mathsf{m}, \mathsf{Con}_0)$ displays crash-like behavior with respect to every subset X of its states; and

(iii) for every state x in a layer of $\mathcal{S}_\mathsf{L}$, the set $\mathsf{L}(x)$ is potence connected.

The following proof establishes these claims in detail. The general scheme will be the same in the next sections (and we will not include as many details).

THEOREM 7.1. *No protocol solves the consensus problem in M^{m}.*

Proof. We start by choosing a layering for M^{m}. Each layer in this case will consist of a single action of the environment. We use $[k]$ to denote the set $\{1, \ldots, k\}$, with $[0]$ denoting the empty set. Define $\mathsf{L} = \{(i, [k]) : 1 \leq i \leq n \text{ and } 0 \leq k \leq n\}$. To see that L is a layering of M^{m}, observe that every layer is an action of the environment and the admissibility condition Ψ^{m} in this model makes no restrictions. Hence, for every protocol D, every run resulting from an infinite sequence of layers of L is immediately a run of M^{m} as desired. Moreover, L satisfies (trivially) fault independence because no processes are ever considered faulty in M^{m}.

Assume by way of contradiction that there is a protocol D solving consensus in M^{m}. Then the system $\mathcal{S} = \mathcal{S}(D, M^{\mathsf{m}}, \mathsf{Con}_0)$ is a system for consensus that satisfies decision, agreement and validity. Given that no process is ever considered faulty in this model, we have by Lemma 5.2 that if there is a bipotent run in $\mathcal{S}$ then the decision property must fail in $\mathcal{S}$, contradicting the assumption that D solves consensus. We will complete the proof by demonstrating the existence of a bipotent run in $\mathcal{S}$ via Theorem 6.3. Hence, we next consider the layered system $\mathcal{S}_{\mathsf{L}} = \mathcal{S}_{\mathsf{L}}(D, M^{\mathsf{m}}, \mathsf{Con}_0)$.

Our next goal is to show that there is a bipotent initial state in $\mathcal{S}_{\mathsf{L}}$. Notice that, by definition of L, the environment is able to "silence" any given process j from a given state on, simply by performing the layer $(j, [n])$ in all subsequent rounds. The other processes will have no way to learn anything about j's state. It immediately follows that $\mathcal{S}_{\mathsf{L}}$ displays crash-like behavior with respect to every subset X of its states. More precisely, we verify that Definition 5.6 holds. Let x, y be two states in a layer of $\mathcal{S}_{\mathsf{L}}$, such that x and y agree modulo j. Let $\hat{R}^x$ and $\hat{R}^y$ be prefixes of L-runs ending in x and y respectively. Then the L-runs $R^x = \hat{R}^x \odot (j, [n])^\infty$ and $R^y = \hat{R}^y \odot (j, [n])^\infty$ are in $\mathcal{S}_{\mathsf{L}}$. Letting $R^x = (r^x, \alpha^x)$ and $R^y = (r^y, \alpha^y)$, it is straightforward to verify that there are times $m_x, m_y \geq 0$ such that

(i) $r^x(m_x) = x$ and $r^y(m_y) = y$,

(ii) $r^x(m_x + k)$ and $r^y(m_y + k)$ agree modulo j for all $k \geq 0$, and

(iii) Every process $i \neq j$ that is not failed in x and in y is nonfaulty in R^x and in R^y.

Recall that $\mathcal{S}$ is a system for consensus satisfying decision and validity. As mentioned above, $\mathcal{S}_{\mathsf{L}}$ inherits these properties from $\mathcal{S}$. It now follows from Theorem 5.8 that there is a bipotent initial state in $\mathcal{S}_{\mathsf{L}}$.

We are finally in a position to apply Theorem 6.3 to derive the existence of a bipotent run R^b in $\mathcal{S}_{\mathsf{L}}$. To do so, we still need to show that for every state x of $\mathcal{S}_{\mathsf{L}}$, the set $\mathsf{L}(x)$ is potence connected. For every pair of processes j, j' we have that $x \cdot (j, [0]) = x \cdot (j', [0])$ because in both cases no messages are lost in the round following x. It follows that $x \cdot (j, [0]) \sim_s x \cdot (j', [0])$. Moreover, for every $k < n$ we have that $x \cdot (j, [k]) \sim_s x \cdot (j, [k+1])$, because the two states can differ only in the state of process $k + 1$. It follows that $\mathsf{L}(x)$ is similarity connected for all x. That $\mathsf{L}(x)$ is potence connected now follows from Lemma 5.7, since $\mathcal{S}$ displays crash-like behavior with respect to $\mathsf{L}(x)$. We thus obtain that there exists a bipotent run R^b in $\mathcal{S}_{\mathsf{L}}$, and the run of $\mathcal{S}$ that corresponds to R^b is bipotent with respect to $\mathcal{S}$, as desired. □

Theorem 7.1 illustrates the fact that it doesn't take much to make consensus impossible. The adversary in M^{m} has fairly limited powers. Furthermore, we obtained the impossibility with a layering that restricts the adversary even more. We remark that this result can be obtained by a modification of the proof of a theorem by Santoro

and Widmayer in [35]. Their analysis involved attaining consensus in the presence of communication link failures. Their proof is the only one we have found that predates this paper and that uses a bipotence-based argument in the style of Fischer et al. [19] in the synchronous context.

7.2. The Synchronous Lower Bound. The analysis we performed for the mobile failure model M^{m} in the synchronous case should, intuitively, apply equally well to the standard t-resilient case in the synchronous model. In this model there is a bound of t on the total number of processes who may fail in the run, and a process some of whose messages are lost is considered faulty. The well-known lower bound for this case (due originally to [18, 14]) states that every consensus protocol must require at least $t+1$ rounds in its worst case run. Roughly speaking, any prefix consisting of t rounds of a run of M^{m} can be viewed as a prefix of a run in the standard omissions failure model with at most t failures. (Formally, the only modification required would be to have the environment's state record processes that have omitted in the past.) The impossibility of solving consensus in M^{m} therefore immediately implies that there can be no protocol solving consensus in t rounds in the omissions model. For if one existed, it would also solve consensus in M^{m}. This argument immediately implies the $t+1$ round lower bound for the omissions and Byzantine failure models. We can not, however, use the impossibility for M^{m} to derive the lower bound for the crash failure model. Nevertheless, even in the crash failure model one might expect to prove that there will exist a bipotent state at the end of round t, and thus derive the $t+1$-round lower bound just as in our analysis for M^{m}. A close inspection, however, shows that things are not *that* simple. There will typically not need to be a bipotent state at the end of round t. But the essence of this idea still works.

We shall now provide the lower bound analysis for the crash failure model and a number of related failure models at once. We assume that the failure model satisfies (i) that in the first round in which a process fails the environment can block the delivery of an arbitrary subset of its messages, (ii) that the environment can silence a faulty process forever in all rounds after the first one in which it fails, and (iii) the environment's local state keeps track of the processes that have failed. It is easy to check that it satisfies fault independence. The layers we will focus on will consist of single environment actions, each corresponding to a single round of the synchronous model. Specifically, we consider two kinds of actions by the environment:

- clean This environment action, applicable at all states, involves no new process failure. Messages of failed processes are not delivered, while all messages of non-failed processes are delivered.
- (j,k) This action is applicable to a state x only if fewer than t processes are failed at x, and process j is not failed at x. As before, messages of failed processes are not delivered, while all messages of non-failed processes other than j are delivered. The messages of j act as with the action $(j,[k])$ in M^{m}: Those addressed to processes up to and including k are not delivered, while the others are delivered.

We denote the layering consisting of all actions of these types by L^t. Notice that the number of processes that fail in a run of $\mathcal{S}_{\mathsf{L}^t}$ is at most t, since once t processes are failed at a state, all later layers will be clean. It is now straightforward to show:

LEMMA 7.2. *Let M be one of the standard synchronous t-resilient models with either crash, omission, or Byzantine failures; let D be a protocol for the processes in the model M, and let $\mathcal{S} = \mathcal{S}(D, M, \mathsf{Con}_0)$. Finally, let x be a state of $\mathcal{S}_{\mathsf{L}^t}$. Then*

(i) L^t is a layering of M that satisfies fault independence;

(ii) for every state x of $\mathcal{S}_{\mathsf{L}^t}$ the set $\mathsf{L}^t(x)$ is similarity connected; and

(iii) if no more than $t-2$ processes are failed at x, then $\mathsf{L}^t(x)$ is potence connected.

Proof. For part (i), recall that in a run of $\mathcal{S}_{\mathsf{L}^t}$ the number of failed processes does not exceed t. Moreover, every L^t action is a legal action for the environment in the model M. It follows that L^t is a layering of $\mathcal{S}$. Also, it is easy to check fault independence.

Part (ii) follows the pattern from M^{m}: If t processes are failed at x, then the set $\mathsf{L}^t(x)$ consists of the singleton state $x \odot \mathsf{clean}$, and is hence trivially similarity connected. If fewer than t processes are failed at x, then $\mathsf{L}^t(x) = \{x \odot \mathsf{clean}\} \cup \{x \odot (j,k) | j \text{ not failed in } x\}$. It is straightforward to check that $x \odot (j,k) \sim_s x \odot (j,k+1)$ for all j and $k < n$; as well, $x \odot \mathsf{clean} \sim_s x \odot (j,0)$. It follows that $\mathsf{L}^t(x)$ is similarity connected in either case.

Finally, for Part (iii), notice that $\mathcal{S}_{\mathsf{L}^t}$ displays crash-like behavior with respect to every set X consisting of states in each of which at most $t-1$ processes are failed. Since an L^t-action fails at most one new process, if at most $t-2$ processes are failed in x, then $\mathsf{L}^t(x)$ is similarity connected by Part (ii), and at most $t-1$ processes are failed in any given state of $\mathsf{L}^t(x)$. It now follows from Lemma 5.7 that $\mathsf{L}^t(x)$ is potence connected and we are done. □ We thus have

LEMMA 7.3. *Consider the system $\mathcal{S}_{\mathsf{L}^t} = \mathcal{S}_{\mathsf{L}^t}(D, M, \mathsf{Con}_0)$. Let x^0 be a bipotent state in a layer of $\mathcal{S}_{\mathsf{L}^t}$ in which no more than f processes are failed. Then there is an L^t-execution with states $x^0, x^1, \ldots, x^{t-f-1}$, such that x^{t-f-1} is bipotent and no more than $t-1$ processes are failed in x^{t-f-1}.*

Proof. We prove by induction on m, for $0 \leq m \leq t-f-1$, that an execution of the desired form exists, with x^m bipotent and where no more than $m+f$ processes are failed in x^m. The basis $m = 0$ holds by assumption. Assume inductively that the claim holds for $m < t-f-1$. Thus, we have that $m+f < t-1$ processes are failed in x^m. By Lemma 7.2(iii) we have that $\mathsf{L}^t(x^m)$ is potence connected, and by an argument similar to the one in the proof of Theorem 6.3, there is a bipotent state $x^{m+1} \in \mathsf{L}^t(x^m)$. By definition of L^t, the number of failed processes in x^{m+1} is at most $m+f+1 \leq t$. □

Since Theorem 5.8 guarantees the existence of a bipotent initial state with $f = 0$ failed processes, Lemma 7.3 immediately implies the existence of a bipotent state x^{t-1} at the end of round $t-1$. By Lemma 5.3, this gives us a t-round lower bound for consensus. The true $(t+1)$-round lower bound is obtained by showing that two rounds are still necessary after a bipotent state:

LEMMA 7.4. *Under the conditions of Lemma 7.2, assume that $t \leq n-2$ and let D be a protocol for consensus. If $\hat{x}$ is a bipotent state in a layer of $\mathcal{S}_{\mathsf{L}^t}$, then there is a state $y \in \mathsf{L}^t(\hat{x})$ in which at least one non-failed process has not decided.*

Proof. Notice that a state x with t failed processes cannot be bipotent, since there is a unique infinite L^t extension starting at x. Hence, to be bipotent, the state $\hat{x}$ can have no more than $t-1$ failed processes. By Lemma 7.2(ii), we have that $\mathsf{L}^t(\hat{x})$ is similarity connected. Since $\hat{x}$ is bipotent, there are states $y^0, y^1 \in \mathsf{L}^t(\hat{x})$ such that y^0 is 0-potent and y^1 is 1-potent. The similarity connectivity of $\mathsf{L}^t(\hat{x})$ implies the existence of states $z^0, z^1 \in \mathsf{L}^t(\hat{x})$ (not necessarily distinct) satisfying $z^0 \sim_s z^1$ that are 0- and 1-potent, respectively. Recall that all states of $\mathcal{S}_{\mathsf{L}^t}$ have at most t faulty processes. Since $t \leq n-2$ and z^0 and z^1 agree modulo j for some j, it follows that there is at least one process $i \neq j$ such that i is not failed in both states and $z_i^0 = z_i^1$. Assume by way of contradiction that every non failed process is decided in both z^0 and z^1. In particular, i is decided, say with value v. Agreement implies that in both states,

every non-failed process decides v. It follows that both z^0 and z^1 are v-potent, and neither of them is $(1-v)$-potent, contradicting the assumption that one of them is 0-potent and the other is 1-potent. □

We can now put the two results together and obtain the desired lower bound:

THEOREM 7.5. *Let $t \leq n-2$. Every t-resilient protocol for consensus in synchronous models where faulty processes can either crash, omit, or behave in a Byzantine fashion, has a run in which decision requires at least $t+1$ rounds. Moreover, for $t = n-1$, every such protocol has a run in which decision requires at least t rounds.*

This result was first proved for crash failures by Dolev and Strong [14], and the latest version of the proof is in [15]. Our proof here is the first one we are aware of that is in the style and spirit of the impossibility proofs for the asynchronous case.[7] Moreover, we feel that it is even simpler than the one of [15]. In addition to generalizing the lower bound for t-resilient consensus, we feel that our proof provides further insight into the structure of consensus protocols in this model. Let us briefly consider an example. It is well-known [34] that there are t-resilient consensus protocols that are guaranteed to decide in precisely $t+1$ rounds. Thus, the worst-case lower bound of Dolev and Strong is tight. We call a protocol in which consensus is always reached in at most $t+1$ rounds *fast*. We can now show

LEMMA 7.6. *Let D be a fast t-resilient consensus protocol. For every execution with states $x^0, x^1, \ldots, x^k, x^{k+1}$ of D, if at most k processes have failed by x^k, and the $(k+1)^{\text{st}}$ round is failure-free, then x^{k+1} cannot be bipotent.*

Proof. By assumption, only k processes have failed by x^{k+1}. If x^{k+1} is bipotent, then by Lemma 7.3 it can be extended to a run with a bipotent state x^t at the end of t rounds. By Lemma 7.4, two more rounds are necessary for agreement in the worst case, contradicting the assumption that D is fast. □

Clearly, Lemma 7.3 also partially describes the situation in runs in which potentially more than one process can crash in a given round. It matches the upper (and lower) bounds given in [15], which show roughly that if in some execution $k+w$ crashes are detected by the end of round k, then agreement can be secured by the end of round $t+1-w$. Hence, by allowing $k+w$ crashes by the end of round k, the environment has essentially "wasted" w faults in its quest to delay agreement. Lemma 7.3 guarantees that the environment has not lost more than w rounds in this case.

8. Asynchronous Message Passing. In Section 3 we illustarted the layering technique by proving impossibility of consensus in the asynchronous shared memory model. The proof was based on a "permutation layering," in which processes move one at a time and there is very little concurrency. Our proof of impossibility for M^{m}, on the other hand, considered every synchronous round to be a layer, and such a layer contains a great deal of concurrency. In this section we apply our framework to prove impossibility for the asynchronous message-passing model. We will give two proofs. One using a permutation layering based on the proof of Section 3, and the other using a "synchronic layering" which is very close to the layering used in the synchronous model M^{m}. We do this to illustrate the closeness of the different models, and to show the choice and flexibility that is often provided by the layering technique. As we have already seen in the shared memory case, in asynchronous models "slow" behavior of processes can be used to imitate the omitting behavior in M^{m}. The small

[7] We have recently been informed that Aguilera and Toueg [6] have independently and slightly later given a proof for this result using a bipotence argument. The structure of their proof is similar to ours. Bar-Joseph and Ben-Or also reported having found some of the arguments in [9].

but crucial difference now will be that, in the asynchronous model, delayed messages will nevertheless eventually be delivered or, similarly, a slow process that is about to write a variable will ultimately write the value. In the synchronous model M^{m}, the lost messages are gone forever. Hence, to perform a careful analysis of the round by round evolution, we need to consider as part of the state (i.e., in the environment's local state) the status of the messages in transit or, similarly, the current values of shared variables. In this sense, our treatment goes slightly beyond the scope of most of the recent work on topological approaches, in which the state of the environment does not play a role, and asynchronous message-passing model are often somewhat subtle to deal with.

Consider the standard (see for example [19, 29]) asynchronous message-passing model M^{mp} in which processes communicate by message passing and both processes and communication are asynchronous. A process is faulty if it is scheduled to move only a finite number of times in an infinite run. The celebrated result of Fischer, Lynch and Paterson [19] proving the impossibility of solving consensus in the presence of a single crash failure was carried out in this model. In the asynchronous message-passing model M^{mp}, an action for i is a pair: it contains a local action on the variables of the local state, and a communication action, which is either skip_i that does nothing, or is a $\mathsf{send}_i(j, m)$ action, specifying the sending of the message m to process j. The effect of a $\mathsf{send}_i(m, j)$ action is to change the environment's state by recording the new message as being in the channel between i and j. For the definition of similarity among states in this model, we consider the j-component of the environment's state to consist of the set of channels whose destinations are process j. Since channels are point-to-point in this model, components corresponding to different processes are disjoint. The environment's actions consist of either scheduling a process i to move, or delivering a set of one or more messages that have been sent and not yet delivered to their destination. The fairness conditions in this model are that (a) at most one process can fail in any given run, and (b) every message that is sent to a process that does not fail in the run must eventually be delivered.

In this model we consider a layering L^{mp} consisting of all layers of the following three types:

- $[p_1, \ldots, p_n]$,
- $[p_1, \ldots, p_{n-1}]$, and
- $[p_1, \ldots, p_{k-1}, \{p_k, p_{k+1}\}, p_{k+2}, \ldots, p_n]$ with $k < n$.

The first two layers are of the same form as in the layering L^{rw} for the shared-memory model. In a layer of L^{mp}, when a process i is scheduled to *move* it first performs its action, and then it receives all the messages that have been sent to i by that point and have not yet been delivered (if any exist). The third type of layer is considered a *full* layer, since it involves all processes as the first type does. In this case, the processes take steps in the linear order given by the sequence, except that p_k and p_{k+1} move *concurrently*. Thus, a message sent by one of these two processes to the other will not be delivered in the current layer. The reason we use the new third type of layer is that with this particular layering, a transposition in a layer of the first two types can change the local state of both p_k and p_{k+1}. Whereas in the case of L^{rw} a move by a process involved either a read or a write, in L^{mp} when a process moves it may both send a message and receive messages. Adding a layer of the third type enables us to take smaller steps, in which at most one process at a time will change its state. We identify a layer of the first type with the corresponding permutation on $\{1, \ldots, n\}$. For a permutation $\pi = [p_1, \ldots, p_n]$, we denote the layer

$[p_1, \ldots, p_{k-1}, \{p_k, p_{k+1}\}, p_{k+2}, \ldots, p_n]$ of the third type by $\pi_{\{k,k+1\}}$. Notice that the set notation in layers of the third type is justified, since the exact same sequence of operations occurs in $\pi_{\{k,k+1\}}$ and $\pi_{\{k+1,k\}}$. Hence, for every state x of $\mathcal{S}_{\mathsf{L}^{mp}}(D)$ and permutation π we have that $x \cdot \pi_{\{k,k+1\}} = x \cdot \pi_{\{k+1,k\}}$.

We can show:

LEMMA 8.1. *Let D be a protocol for the processes in the model M^{mp}.*

(i) L^{mp} is a layering of M^{mp} that satisfies fault independence;

(ii) $\mathcal{S}_{\mathsf{L}^{mp}} = \mathcal{S}_{\mathsf{L}^{mp}}(D, M^{mp}, \mathsf{Con}_0)$ displays crash-like behavior with respect to every subset X of its states; and

(iii) for every state x in a layer of $\mathcal{S}_{\mathsf{L}^{mp}}$, the set $\mathsf{L}^{mp}(x)$ is potence connected.

Proof. (i) Let R be an L^{mp}-run. By definition, R consists of an infinite sequence of layers from L^{mp}. Since in each layer L of L^{mp} at least $n-1$ processes move, in an infinite number sequence there can be at most one process that fails to take an infinite number of steps. It follows that at most one process can be faulty in R. Since a process that moves receives all of the pending messages sent to it, all messages sent to nonfaulty processes in R are delivered. It follows that the run R' that corresponds to R is admissible. It is, in addition, easy to check that L^{mp} satisfies fault independence, since M^{mp} satisfies no finite failure.

Part (ii) follows from the fact that the environment can halt the operation of an arbitrary process j at any given state by repeatedly performing layers of the form $[1, 2, \ldots, j-1, j+1, \ldots, n]$ from that state on.

It remains to show (iii). First notice that just as in the case of L^{rw} we have that

$$x \cdot [p_1, \ldots, p_{n-1}] \cdot [p_n, p_1, \ldots, p_{n-1}] \quad = \quad x \cdot [p_1, \ldots, p_n] \cdot [p_1, \ldots, p_{n-1}],$$

since in both cases exactly the same actions take place in the same order in the two layers following x. Lemma 5.1(ii) implies that $x \cdot [p_1, \ldots, p_n] \sim_p x \cdot [p_1, \ldots, p_{n-1}]$, and it follows that every state of $\mathsf{L}^{mp}(x)$ obtained from x by a layer of the second type is potence connected to a state obtained by a (full) layer of the first type. Let X_f be the subset of $\mathsf{L}^{mp}(x)$ consisting of states of the form $x \odot \mathsf{L}$, where L is a full layer. It remains to show that X_f is potence connected. Since, by part (ii), $\mathcal{S}_{\mathsf{L}^{mp}}(D)$ displays crash-like behavior with respect to X_f, Lemma 5.7 implies that it suffices to show that the set X_f is similarity connected.

We first claim that for a full layer $L = \pi$ of the first type and an index $k < n$, we have that

$$x \cdot \pi \quad \sim_s \quad x \cdot \pi_{\{k,k+1\}}.$$

To see why, recall that according to the environment's actions in a layer of the types we are considering, a process that is scheduled first performs its action and then receives the messages that were sent to it and are still in transit. Thus, the actions of a process p_j in the layer $x \odot \pi$ depends only on p_j's local state in x. It now follows that every process p_i with $i \neq k+1$ performs the same actions and receives the same messages in the layer following x in both $x \odot \pi$ and $x \odot \pi_{\{k,k+1\}}$. The only process whose state at the end of the layer may differ in the two cases is p_{k+1}. If p_k sends p_{k+1} a message, it will be delivered in the first case, and will remain in the channel from p_k to p_{k+1} in the second case. Since the channel from p_k to p_{k+1} is part of the p_{k+1}-component of the environment's state,[8] the two states agree modulo p_{k+1} and

[8] This is the place where we make use of the notion of j-component in the definition of similarity. Intuitively, we expect this will be needed whenever message-passing models are considered in which messages can be delayed in the channels for more than one "round".

are thus similar. The claim follows.

Finally, we use this last claim to argue as in the shared-memory case. Recall (proof of Lemma 3.5) that $\mathsf{Tr}(k,\pi)$ is the permutation obtained by transposing the k^{th} and $(k+1)^{\text{st}}$ elements in π. Since the order of the concurrent processes in a layer of the third type is immaterial and since $\pi_{\{k+1,k\}} = \mathsf{Tr}(k,\pi)_{\{k,k+1\}}$ we obtain:

$$x \cdot \pi \quad \sim_s \quad x \cdot \pi_{\{k,k+1\}} \quad = \quad x \cdot \pi_{\{k+1,k\}} \quad = \quad x \cdot \mathsf{Tr}(k,\pi)_{\{k,k+1\}} \quad \sim_s \quad x \cdot \mathsf{Tr}(k,\pi),$$

and we are done, since any layer of the first type can be transformed into any other by a sequence of transpositions. □

Given Lemma 8.1, we can use Theorem 6.3 which,together with Theorem 5.8 and Lemma 5.2,now yields:

THEOREM 8.2. *No protocol solves the consensus problem in M^{mp}.*

Observe that the layers in a given layering need not be of the same length. Indeed, we saw both full and non-full layers being used in the permutation layerings above. In addition, in the permutation layerings we discussed above each process performed at most one action in a layer. This was chosen for simplicity. In [33], other variants of permutation layerings are given in which a process may perform many actions in a layer.

8.1. A synchronic layering. We now sketch a second type of layering for the asynchronous models we have considered. This one, which we call the *synchronic* layering, imitates the synchronous model and yields layered systems of M^{mp} whose runs are very close in structure to those of M^{m}. We shall describe it for the message-passing model, and a completely analogous treatment works for the asynchronous shared memory model sketched in Section 3 as well.

The synchronic layering, denoted by L^s, is defined as follows. The layers are denoted by (j,A) or (j,k) with $1 \le j \le n$ and $0 \le k \le n$. In the sequence (j,A), all processes except j move, and then they all receive whatever outstanding messages are addressed to them (including messages that have just been sent). The "A" in (j,A) stands for process j being *absent* from the layer. In an action of the form (j,k), all n processes move simultaneously, and then all outstanding messages to the processes are delivered as before, with one exception: If j's current action is $\mathsf{send}_j(\ell,m)$ and $\ell \le k$, then this message remains outstanding and is not delivered. Notice that the actions $(j,0)$ are all identical and independent of j; they are written this way for ease of exposition. We think of j as being the "slow" process in the layer defined by (j,n) or (j,A).

As with the permutation layering, a layer in the synchronic case contributes sufficiently to the fairness requirements made by the model: At least $n-1$ processes get to move in every layer, and all messages that have been sent by the previous layer to a process that moves are guaranteed to be delivered by the end of the current layer. It follows that L^s is a layering of M^{mp}, and it is easy to check that it satisfies fault independence. Crash-like behavior is immediate as well. To complete the impossibility argument, one needs to show that $\mathsf{L}^s(x)$ is potence connected. The argument for this uses elements from the proof for M^{m} and from the proof in the case of permutation layerings. For every j and every $k < n$, the states $x \odot (j,k)$ and $x \odot (j,k+1)$ agree modulo process $k+1$; they can differ at most in the state of $k+1$ and of the channel from j to $k+1$, which is part of the $k+1$-component of the state. It follows that the set of states of the form $x \odot (j,k)$ is similarity connected, and by Lemma 5.7 this set is also potence connected.

The argument that all of $\mathsf{L}^s(x)$ is potence connected uses the diamond property that

$$x \odot (j,n) \odot (j,A) \quad = \quad x \odot (j,A) \odot (j,0).$$

To see why this equality is true, notice that in both $x \odot (j,n)$ and $x \odot (j,A)$ no process receives a message from j in the last round. These messages are sent, based on j's local state in x, in the last round of $x \odot (j,n)$, but are only received in the following round. In the last round of $x \odot (j,A) \odot (j,0)$ process j sends messages to everyone, and these messages are all received in that round. Notice, however, that these messages are sent before j has received any new messages following the state x. Hence, the messages it sends are again based on $j's$ local state in x. It follows that the same messages are sent by j in the last round of $x \odot (j,A) \odot (j,0)$ and in the last round of $x \odot (j,n)$. Moreover, in both cases these messages are received in the second round following x. It follows that $x \odot (j,n) \odot (j,A)$ and $x \odot (j,A) \odot (j,0)$ are equal.

9. Conclusions. Roughly speaking, we have seen that invariably, when the processes follow a protocol D in a given model M, the environment has a simple best strategy to delay consensus: Start with a bipotent state and, for as long as possible, find a layer that will cause a transition from the current bipotent state to a successor state that is also bipotent. If there exists a layering function L for which $L(x)$ can be shown to be potence connected for all x, then we obtain impossibility. For lower bounds the analogy is almost (but not quite) complete.

Interestingly, some of the layering functions for the different models turn out to be extremely similar. Moreover, we believe that the vast majority of lower bounds and impossibility results for consensus in the different models can be cast in terms of such layering functions, with essentially the same proof. We feel that this uniformity provides new insight into the inherent structure of the consensus problem, and helps pinpoint, in a fairly model-independent way, the reason why consensus is difficult.

Our analysis emphasized and focused on the role of connectivity in the structure of consensus. Indeed, it is straightforward to convert our proofs of the existence of a bipotent state to ones that show the following, roughly: For a set X of states and a layering function L, define $L(X) = \cup_{x \in X} L(x)$. Let $X_0 = \mathsf{Con}_0$, and inductively define $X_{k+1} = L(X_k)$. The proof of Lemma 5.8 shows that X_0 is similarity connected and potence connected (in the presence of a single failure). For the layering functions we have been discussing it is not hard to modify our proofs to show that, roughly, if X_k is potence connected then so is X_{k+1}. Therefore, we have two possible styles of proofs. First is the FLP [19] style, which we followed in the paper. The strategy in this case is to show that there is a bipotent initial state, and then find a sequence of successor bipotent states by proving *locally* the connectivity of the successors of a bipotent state. The other, more *global* type of proof, is closer to the topological approach of works such as [7, 24, 22, 36], and in particular [23]. It shows that each layer X_k is connected, and since it has at least one 0-potent and one 1-potent state, then it must have a bipotent state. The connectivity of these sets is essentially a topological property (although clearly the analysis involves no deep topology). As long as these sets are connected, Lemma 3.8 guarantees that there is a bipotent state in the set.

In this paper we dealt with consensus only, but the connectivity properties we prove hold for arbitrary decision problems as well, and we pursue this line further in a sequel paper (as described in [33]), where we show how our work generalizes

the necessary conditions of [10] to other models, perhaps more remarkably to the synchronous model.

It is often useful to ask what knowledge [16] about the state is required by a process in order to be able to reach a decision. We note that once the protocol is fixed, as long as the state is bipotent, even an observer with complete information about the state cannot determine the final outcome. In a precise sense, no knowledge about the actual state can help the process decide at that point. Indeed, this has been formally captured in Lemma 5.3. Once the state ceases to be bipotent (and becomes "unipotent"), however, information about the state can be of use in determining what value a process should decide on. In fact, the proof of Lemma 7.4 shows that at the first instant along a run in which the state ceases to be bipotent in the t-resilient synchronous case, there are often many processes who have insufficient knowledge to be able to decide. An additional round is sufficient for providing them with this knowledge, in which case they can safely decide. In summary, as long as the state is bipotent, there are "structural" reasons why decision cannot be taken. After that, the reasons merely involve disseminating the relevant information to all relevant parties. In particular, in the $t+1$-round lower bound (Corollary 7.5), t rounds are paid on account of topological reasons (the need to disconnect the set of global states), while an additional round is paid on account of the insufficient knowledge available after t rounds. A similar phenomena may occur in other topology-related problems, such as the k-set consensus problem in the synchronous case [13], where the tight lower bound is $\lfloor t/k \rfloor + 1$ rounds.

Acknowledgments:. We would like to thank Juan Garay for bringing to our attention the mobile failure model and the related reference [35]. Special thanks to Eric Ruppert for pointing out an error in the definition of the message passing asynchronous layering in [33]. We thank Marcos Aguilera, Bernadette Charron-Bost, Idit Keidar and the referees for a very careful reading of an earlier version of the paper and many useful comments.

REFERENCES

[1] Baruch Awerbuch, "Complexity of network synchronization," *J. ACM,* **32:**4, (1985), pp. 804-823.

[2] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, N. Shavit, "Atomic snapshots of shared memory," *J. ACM,* **40:**4, (1993), pp. 873-890.

[3] H. Attiya, A. Bar-Noy and D. Dolev, "Sharing memory robustly in message-passing systems," *J. ACM*, **42:**1, (1995), pp. 124-142.

[4] H. Attiya and S. Rajsbaum, "The combinatorial structure of wait-free solvable tasks," Proc. 10th Intl. Workshop on Distributed Algorithms (WDAG), 1996, (O. Babaoglu and K. Marzullo, Eds.), *Springer-Verlag LNCS # 1151* pp. 321-343. Submitted for journal publication.

[5] Hagit Attiya and Jennifer Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics,* McGraw–Hill, 1998.

[6] M. K. Aguilera and S. Toueg, "A simple bivalency-based proof that t-resilient consensus requires $t+1$ rounds," *Inf. Proc. Let.*, **71**:3-4 (1999), pp. 155-158.

[7] E. Borowsky and E. Gafni, "Generalized FLP impossibility result for t-resilient asynchronous computations," Proc. 1993 ACM Symp. on Theory of Computing (STOC), 1993), pp. 91-100.

[8] E. Borowsky and E. Gafni, "A simple algorithmically reasoned characterization of wait-free computations," Proc. 16th ACM Symp. on Principles of Distributed Computing, (PODC), 1997, pp. 189-198.

[9] Z. Bar-Joseph and M. Ben-Or, "A tight lower bound for randomized synchronous consensus," Proc. 17th ACM Symp. on Principles of Distributed Computing, (PODC), 1998, pp. 193-199.

[10] O. Biran, S. Moran, S. Zaks, "A combinatorial characterization of the distributed 1-solvable tasks," *J. Algorithms*, **11** (1990), pp. 420-440.

[11] O. Biran, S. Moran and S. Zaks, "Tight bounds on the round complexity of distributed 1-solvable tasks," *Theor. Comp. Sci.,* **145** (1995), pp. 271-290.

[12] T. Chandra and S. Toueg, "Unreliable failure detectors for asynchronous systems," Proc. 10th ACM Symp. on Principles of Distributed Computing (PODC) (1991), pp. 257-272.

[13] S. Chaudhuri, M. P. Herlihy, N. Lynch, and M. R. Tuttle, "Tight bounds for k-set agreement," *J. ACM*, **47:**5 (2000), pp. 912 - 943. Preliminary version in IEEE FOCS 1993.

[14] D. Dolev, H. R. Strong, "Authenticated algorithms for Byzantine agreement," *SIAM J. Comput.*, **12:**4 (1983), pp. 656-666.

[15] C. Dwork, Y. Moses, "Knowledge and common knowledge in a byzantine environment: crash failures," *Inf. and Comp.*, **8:**2 (1990), pp. 156-186.

[16] R. Fagin, J. Y. Halpern, Y. Moses and M. Y. Vardi, *Reasoning About Knowledge*, MIT Press 1995.

[17] M. J. Fischer, "The consensus problem in unreliable distributed systems (a brief survey)," Research Report YALE/DCS/RR-273, Yale University Dept. Comp. Sci., June 1983.

[18] M. J. Fischer, N. A. Lynch, "A lower bound for the time to assure interactive consistency," *Inf. Proc. Let.,* **14:**4 (1982), pp. 183-186.

[19] M. J. Fischer, N. A. Lynch, and M.S. Paterson, "Impossibility of distributed commit with one faulty process," *J. ACM*, **32:**2 (1985), pp. 374-382.

[20] E. Gafni, "Round-by-round fault detectors: unifying synchrony and asynchrony," Proc. 17th ACM Symp. on Principles of Distributed Computing, (PODC), 1998, pp. 143-152.

[21] M. P. Herlihy, "Wait-Free Synchronization," *ACM ToPLaS* **13:**1 (1991), pp. 123-149.

[22] M. P. Herlihy and S. Rajsbaum, "A primer on algebraic topology and distributed computing," *Computer Science Today,* Jan van Leeuwen (Ed.), *Springer-Verlag LNCS # 1000* (1995), pp. 203-217.

[23] M. P. Herlihy, S. Rajsbaum, M. R. Tuttle, "Unifying synchronous and asynchronous message-passing models," Proc. 17th ACM Symp. on Principles of Distributed Computing, (PODC), 1998, pp. 133-142.

[24] M. P. Herlihy and N. Shavit, "The topological structure of asynchronous computability," *J. of the ACM,* 46(6), 1999, pp. 858-923. Preliminary version in PODC 1993.

[25] G. Hoest and N. Shavit, "Towards a topological characterization of asynchronous complexity," Proc. 16th ACM Symp. on Principles of Distributed Computing (PODC), 1997, pp. 199-208.

[26] I. Keidar and S. Rajsbaum, "On the Cost of Fault-Tolerant Consensus When There Are No Faults - A Tutorial," MIT Technical Report MIT-LCS-TR-821, May 24 2001. Preliminary version in SIGACT News 32(2), Distributed Computing Column, pp. 45-63, June 2001.

[27] M. C. Loui and H.H. Abu-Amara, "Memory requirements for agreement among unreliable asynchronous processes," In *Parallel and Distributed Computing*, F. P. Preparata, editor, *Adv. Comp. Res.*, **4**, JAI Press (1987) pp 163-183.

[28] R. Lubitch and S. Moran, "Closed schedulers: a novel technique for analyzing asynchronous protocols," *Dist. Comp.,* **8** (1995), pp. 203-210.

[29] N. A. Lynch, *Distributed Algorithms,* Morgan Kaufmann Publishers, Inc. 1996.

[30] Elizabeth Borowsky, Eli Gafni, Nancy Lynch, and S. Rajsbaum, "The BG Distributed Simulation Algorithm," Technical Memo MIT/LCS/TM-573, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, December, 1997. To appear in *Distributed Computing,* Vol. 14, No. 3, July 2001.

[31] R. van der Meyden and Y. Moses, Top-down considerations on distributed computing (Invited Talk), Proc. 12th Intl. Symp. on Distributed Computing (DISC), 1998, S. Kutten, Ed., pp. 16-20.

[32] M. P. Herlihy, S. Rajsbaum, "New perspectives in distributed computing," (Invited Lecture) M. Kutylowski, L. Pacholski, T. Wierzbicki (Eds.) Proc. 24th Intl. Symp. Math. Foundations of Comp. Sci. (MFCS), *Springer-Verlag LNCS # 1672* (1999), pp. 170-186.

[33] Y. Moses and S. Rajsbaum, "The unified structure of consensus: a layered analysis approach," Proc. 17th ACM Symp. on Principles of Distributed Computing, (PODC), 1998, pp. 123-132.

[34] M. Pease, R. Shostak and L. Lamport, "Reaching agreement in the presence of faults," *J. ACM*, **27:**2 (1980), pp. 228-234.

[35] N. Santoro and P. Widmayer, "Time is not a healer," In Proc. 6th Annual Symp. Theor. Aspects of Comp. Sci. (STACS), 1989. *Springer Verlag LNCS # 349* pp. 304-313.

[36] M. Saks and F. Zaharoglou, "Wait-free k-set agreement is impossible: The topology of public knowledge," Proc. 1993 ACM Symp. on Theory of Computing (STOC), 1993, pp. 101-110.