# Prisoners, Rooms, and Light Switches

Daniel M. Kane*

Department of Mathematics and Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA, U.S.A.

dakane@ucsd.edu

Scott Duke Kominers†

Harvard Business School and Department of Economics
Harvard University
Boston, MA, U.S.A.

kominers@fas.harvard.edu

Submitted: Sep 28, 2020; Accepted: Oct 11, 2020; Published: Feb 12, 2021


**Abstract**

We examine a new variant of the classic prisoners and light switches puzzle: A warden leads his $n$ prisoners in and out of $r$ rooms, one at a time, in some order, with each prisoner eventually visiting every room an arbitrarily large number of times. The rooms are indistinguishable, except that each one has $s$ light switches; the prisoners win their freedom if at some point a prisoner can correctly declare that each prisoner has been in every room at least once. *What is the minimum number of switches per room, $s$, such that the prisoners can manage this?* We show that if the prisoners do not know the switches' starting configuration, then they have no chance of escape—but if the prisoners do know the starting configuration, then the minimum sufficient $s$ is surprisingly small. The analysis gives rise to a number of puzzling open questions, as well.

**Mathematics Subject Classifications:** 68M14, 00A08, 91A06, 91A12

*Kane gratefully acknowledges the support of an NSF Postdoctoral Fellowship, NSF CAREER Award 1553288, and a Sloan Research Fellowship.

†Kominers gratefully acknowledges the support of a Harvard Mathematics Department Highbridge Fellowship, an NSF Graduate Research Fellowship, National Science Foundation Grants CCF-1216095 and SES-1459912, an AMS-Simons Travel Grant, the Harvard Milton Fund, and the Ng Fund and the Mathematics in Economics Research Fund of the Harvard Center of Mathematical Sciences and Applications.

# 1 Introduction

The following puzzle is well-known:

> There are $n$ prisoners in a prison. The warden offers a deal: He will lead the prisoners into a particular room one at a time in some order, with the guarantee that each prisoner will eventually be led into the room arbitrarily many times. At any point, a prisoner may declare that all the prisoners have been in the room. If the declaring prisoner is correct, then the prisoners are freed. Otherwise, they are executed(!).
>
> The prisoners are allowed to confer ahead of time to agree upon a strategy, but are allowed no direct communication after the exercise starts. The room that they are led into is completely featureless except for a light switch, which starts in the OFF position. What switch flipping strategy guarantees the prisoners' freedom?

This "One-Bulb Room" problem appears in Winkler's *Mathematical Puzzles* [8, p. 103]; Winkler remarks that it is also appeared in *The Emissary* [1] and as a puzzler on *Car Talk* [7]. Dehaye, Ford, and Segerman [2] have studied a similar problem, in which the prisoners may synchronize their actions with a global clock.

The second author, Kominers, and Chen [5] posed a generalization of the One-Bulb Room problem, inquiring about what happens when the number of rooms is increased to $r > 1$.[1] This query, in turn, has a number of variations. Some turn out to be surprisingly subtle—to whit, the original solution of [5] contained an error, which was spotted by the first author. Discussions about a corrected version of the problem ([6]) have led to this article, which rigorously investigates the circumstances under which the prisoners may win their freedom (and those in which they are doomed to failure).

## 1.1 A Solution for $n$ Prisoners, One Room

With only one room to track, the prisoners have a fairly simple escape strategy. They select a *leader*, who will keep count of the number of prisoners who have entered the room. The other prisoners signal that they have been in the room by turning the light switch ON the first time they are able to do so; the leader acknowledges these signals by turning the switch OFF again. Given that each prisoner only signals once, the leader will know that all the prisoners have been in the room once he has acknowledged $n - 1$ signals.

Of course, if $n > 1$, the leader has entered the room at least once by the time he has acknowledged $n - 1$ signals; at that time he can declare immediately. (When $n = 1$, the leader must wait until he has been in the room once before declaring.)

[1] The problem was later featured in one of the second author's *Bloomberg Opinion* puzzle columns [4]; the solution presented there and in [6] corresponds to the protocol we present in Section 3.3, although our analysis here is far more formal.

### 1.2 A Formal Framework

Although the $n$ prisoner, one room solution just discussed is fairly straightforward, much of our later discussion will be far more complex. Therefore, we introduce a notation for prisoners' solution protocols at the outset, using the $n$ prisoner, one room solution as an example.

We say that a room with $s$ switches is *in configuration* $(a_1, \ldots, a_s)$ if its switches display the values $a_1, \ldots, a_s$ (in sequence). For example, a room with one switch has two possible configurations: (ON) and (OFF). In general, it will be somewhat cumbersome to describe configurations as lists of switch values, so we often give names to configurations instead.

In the solution described above, all the prisoners who are not the leader follow a very simple protocol: they wait until they see the light switch OFF, and then turn it ON. We notate this procedure as "FLIP((OFF),(ON))." In general, we define

> FLIP($A$,$B$): Wait until you see a room in configuration $A$ and then reconfigure it to $B$.

The leader, meanwhile, must apply a more complicated protocol. First, assuming $n > 1$, the leader must turn the lights off $n - 1$ times. We could express this by writing "FLIP((ON),(OFF))" $n - 1$ times, but this seems cumbersome. We instead write:

> REPEAT($n - 1$)
>  FLIP((ON),(OFF)),

where REPEAT($k$) indicates that the prisoner should repeat the nested actions $k$ times. After the counting phase is completed, the leader must *declare*, announcing that everyone has entered the room; this operation is written "DECLARE." The leader's complete protocol in the solution to the one-room case is therefore:

> REPEAT($n - 1$)
>  FLIP((ON),(OFF))
> DECLARE.

As we have already observed, if $n = 1$, then the leader (who is the only player by default) must follow a slightly different protocol. He must wait until he enters a room, and then must DECLARE. Equivalently, since the switch starts OFF, he must wait until he sees the configuration (OFF). We define the operation SEE($a$), which means that a prisoner waits (i.e., he progresses no further through his protocol) until he enters a room that has configuration $a$. The full solution to the one-room problem when $n = 1$ is therefore:

> SEE((OFF))
> DECLARE.

(Although the SEE($a$) operation is equivalent to the "trivial" flipping operation FLIP($a$,$a$), it is useful to distinguish these two operations for clarity.)

As the solutions we discuss shall often require trivial modifications in the case $n = 1$ (as occurs in the $n$ prisoner, one room problem), we will hereafter assume $n > 1$ except where otherwise noted.

In order to add clarity to these protocols we sometimes add comments delineated by double backslashes:

DECLARE. \\ This is a comment.

### 1.3 A Note on Starting Configuration

Note that protocol we have just described assumes that the room is known to start with its switch in the OFF state. If the room is known to start in the ON state, an analogous protocol may be applied. On the other hand, if the room begins in an unknown configuration, a slightly more complicated approach must be used. In particular, the prisoners may use the following protocol:

Leader's Protocol:
REPEAT($2n - 2$)
  FLIP((ON),(OFF))
DECLARE;

Other Prisoners' Protocol:
REPEAT(2)
  FLIP((OFF),(ON)).

The analysis of this protocol is similar to that of the simpler one for a known starting configuration. The differences here are that each non-leader signals twice, and that, if the room stats in the (ON) state, the leader will "acknowledge" an extra signal. This causes him to declare before all other prisoners have signalled twice—instead, he declares after all prisoners but one have signalled twice, and the remaining prisoner has signalled once. Nonetheless, whenever the leader declares, all prisoners will have entered the room at least once.

### 1.4 $n$ Prisoners, $r$ Rooms

Having thus handled the problem as stated, we consider generalizations in which the prison has $r \geqslant 1$ rooms that the prisoners might be led into. There are several slight variants of this generalization; we discuss them in ascending order of difficulty.

#### 1.4.1 Distinguishable Rooms

If the different rooms are distinguishable, then the prisoners can treat each room as a separate, parallel instance of the original problem. More generally, if the rooms may be partitioned into classes of mutually indistinguishable rooms, then each class may be addressed separately (at least under the assumption that the same prisoner would have

declared in each sub-instance). Hence, to keep the problem interesting, we only examine the case in which the only features distinguishing the rooms are the configurations of their switches.

### 1.4.2 Each Prisoner Visits at least One Room

With multiple rooms, the warden might relax his requirements of the prisoners. In particular, he might ask that the prisoners declare only after each has visited at least one room. However, this problem can be solved using a protocol similar to that used in the one-room case. In particular, even for an unknown starting configuration, the following protocol wins the prisoners freedom:

LEADER'S PROTOCOL:
REPEAT$((r+1)(n-1))$
  FLIP((ON),(OFF))
DECLARE;

OTHER PRISONERS' PROTOCOL:
REPEAT$(r+1)$
  FLIP((OFF),(ON)).

### 1.4.3 Each Prisoner Visits All Rooms

From the preceding analysis, we are narrowed to a case in which the warden can really make trouble for the prisoners: In this case, we have $r$ rooms, distinguishable from each other only by the states of their switches, and the warden requires that each prisoner must have visited every room before some prisoner declares. In the interest of fairness, the warden must grant the prisoners the guarantee that, if they wait long enough, each prisoner will eventually be led into every room an arbitrarily large number of times; we say that a schedule of room visits is *valid* if it has this property. We are now left with the following question:

> What is the minimum number of switches per room, $s$, so that the prisoners have a protocol ensuring that they can win their freedom under any valid schedule of room visits?

**Definition 1.** We say that a protocol is *winning* if for any valid sequence of room visits (1) some prisoner will eventually declare and (2) no prisoner will declare until each prisoner has visited every room at least once.

## 2 A Negative Result when the Starting Configuration is Unknown

We begin by supposing that the rooms' starting configurations are unknown. Unlike the one-room case, in which this difficulty can be circumvented with only a slight modification

of the prisoners' protocol (recall Section 1.3), if there are $r > 1$ rooms and their initial configurations are unknown, then *the prisoners have no protocol that is guaranteed to work*.

In order to prove this kind of impossibility result, it will be important to describe the adversarial strategy for the warden. We begin with the following lemma:

**Lemma 2.** *Suppose that each room has a finite number, $s$, of switches. Fix a deterministic protocol for the prisoners. For that protocol, there is a starting configuration for the rooms and a pair of schedules, $\Sigma_1$ and $\Sigma_2$, having the following properties:*

1. *Under $\Sigma_1$, the prisoners will only ever visit one of the rooms.*
2. *Under $\Sigma_2$, each prisoner will visit each room arbitrarily often, i.e., the schedule is valid in the sense described in Section 1.4.3.*
3. *The schedules $\Sigma_1$ and $\Sigma_2$ are indistinguishable from the prisoners' perspectives. In particular, if the prisoners execute their protocol for $\Sigma_1$, then each prisoner will see the same sequence of room configurations as if they had instead executed their protocol for $\Sigma_2$ (and vice versa).*

*Proof.* In order to produce the desired schedules $\Sigma_1$ and $\Sigma_2$, we start with some given room configuration, say $c$. Consider a room in configuration $c$. Fix an ordering of the prisoners, and consider sending them into that room repeatedly in that order. After each pass through such a cycle, record the current configuration of the room in question. Since there are finitely many configurations, some configuration, $d$, must show up infinitely often.

We/the warden start/s with the rooms configured so that one room—the "first room"—is in configuration $c$ and the rest are in configuration $d$. In schedule $\Sigma_1$, we send all the prisoners through the first room in the specified order repeatedly. It is clear that under this schedule $\Sigma_1$, the prisoners will only ever visit one of the rooms.

To construct schedule $\Sigma_2$, we maintain an ordered list of the rooms and a *current room* indicator, initially set to indicate the first room. The warden repeatedly sends the prisoners into the current room in order; however, if the current room is in configuration $d$ at the end of a cycle, then he switches the current room indicator to the next room on the ordered list.

To prove that schedules $\Sigma_1$ and $\Sigma_2$ are indistinguishable, we note first that in both cases, the order in which prisoners are chosen to enter rooms is the same. Furthermore, we claim that for each $k$, the configuration of the current room in step $k$ of $\Sigma_2$ is the same as the configuration of the first room at step $k$ of $\Sigma_1$. We prove this by induction on $k$. For $k = 1$, the current room is in configuration $c$ in either case. If the current rooms were in the same configuration at each step leading up to $k$, then they will be the same at step $k$ because the $k$-th prisoner will have the same history in either schedule and will visit a room in the same configuration, and thus will make the same change to it. The one slight twist that needs to be added is that if we have completed a pass through the prisoners and the current room is in configuration $d$, then the warden will change the current room

indicator in schedule $\Sigma_2$. However, this will necessarily change the current room from one room in state $d$ to another, and will not affect our claim.

We have left to prove that $\Sigma_2$ is valid. For this, we note that in $\Sigma_1$, by assumption, it is the case infinitely often that at the end of a cycle through the prisoners, the warden finds the first room in configuration $d$. Therefore, by the indistinguishability result already proven, in schedule $\Sigma_2$, the warden will infinity often at the end of a cycle find the current room in configuration $D$. Therefore, in $\Sigma_2$, the current room indicator will change infinitely often. However, each time the current room indicator changes, every prisoner will visit the new current room at least once before the indicator changes again. Since the indicator changes infinitely often, and since the warden will change it in a sequence that cycles through all of the rooms infinitely often, each room will become the current room arbitrarily many times. Thus, the schedule $\Sigma_2$ will send each prisoner into each room arbitrarily many times; hence, it is valid. □

Note that if the warden leads the prisoners through the indistinguishable schedules $\Sigma_1$ and $\Sigma_2$ constructed in Lemma 2, the prisoners must either eventually declare under both or never declare under either. But this would mean they either declare incorrectly under $\Sigma_1$ or never declare under the valid schedule $\Sigma_2$. This then leads naturally to our impossibility result.

**Theorem 3.** *Assume $r > 1$ rooms and that the number of switches per room, $s$, is finite. Then for any deterministic protocol for the prisoners, there is a set of initial room configurations and an associated valid schedule $\Sigma$ so that if the prisoners visit rooms according to $\Sigma$, they will either declare incorrectly or fail to declare.*

*Proof.* We use the initial room configurations specified by Lemma 2. In order to find the schedule, we first consider what happens if the prisoners are sent into rooms according to schedule $\Sigma_2$. If they do not declare, then we have a valid schedule for which the prisoners never declare, and we are done. If they do declare, then we note that by the indistinguishability property that the prisoners must also declare under schedule $\Sigma_1$. This declaration will necessarily be incorrect, as $\Sigma_1$ only involves visits to a single room (and $r > 1$). However, $\Sigma_1$ is not a valid schedule. We construct a valid schedule from $\Sigma_1$ by noting that the declaration under $\Sigma_1$ will occur after some finite number $\ell$ of visits. We thus pick a schedule $\Sigma_1'$ that agrees with $\Sigma_1$ for the first $\ell + 17$ visits, and after that sends all prisoners to all rooms infinitely many times in whatever order is desired. It is clear that $\Sigma_1'$ is valid; however, since $\Sigma_1'$ agrees with $\Sigma_1$ on the first $\ell + 17$ visits, the prisoners will still declare on step $\ell$—before they have all visited all the rooms. □

## 3 A Solution for When the Starting Configuration Is Known

Given the impossibility result presented in the prior section, we henceforth focus on the case in which the rooms' starting configurations are known in advance.

### 3.1 Arbitrary Starting Configuration

We start with the general case, in which the starting configuration, while known, may be arbitrary. Arbitrary starting configurations have the potential to make the prisoners' task difficult, since seeing a room in a given configuration could just mean that the room started in that configuration.

The prisoners would prefer to work in a simple, "canonical" starting configuration, for example the one in which where all of the switches start in the OFF position. Fortunately, there is an approach that allows one to use a protocol that works for the all-OFF starting configuration (satisfying some mild extra conditions) to produce a protocol that works for an arbitrary, known starting configuration.

**Lemma 4.** *Suppose that, given $n$, $r$, and $s$, the prisoners have a winning protocol if all of the switches start in the* OFF *position. Suppose additionally that*

1. *the winning protocol makes use of two room configurations, here denoted* 0 *and* 1*, where* 0 *is the all-*OFF *configuration (and* 1 *is some other configuration); and*

2. *one of the prisoners is designated as the leader, and all non-leader prisoners will ignore all rooms they are sent into until they see a room in a configuration other than* 0 *or* 1*.*

*Then, the prisoners have a winning protocol for any (known) set of starting configurations.*

The idea of this protocol is to run the old winning protocol preceded by a protocol that puts all switches in the OFF position. We suppose that of the $r$ rooms, it is known that $r_0 > 0$ start in configuration 0 and $r_1 > 0$ start in configuration 1. In fact, we will not need to know the multiplicities of the other configurations. (If there are not two configurations showing in a strictly positive number of rooms at the start, then all the rooms must be in the same starting configuration—and the prisoners can simply re-label that as "all-OFF.")

The protocol starts with the leader changing the configurations of the $r - r_0 - r_1$ rooms not in configuration 0 or 1 to configuration 1. We next need a way of informing the other prisoners that these rooms have been cleared out. This is done by changing rooms between configurations 1 and 0. In particular, each non-leader will attempt to change rooms from configuration 0 to configuration 1 a total of $r_0 + 1$ times before starting on his old protocol. This number is chosen so that the prisoner cannot possibly see that many rooms in configuration 0 without someone changing rooms to configuration 0. In the meantime, the leader will change rooms from configuration 1 to configuration 0. He will do this $r - r_0 + (n-1) \cdot (r_0 + 1)$ times. This ensures that each other prisoner has changed $r - r_0$ rooms from 0 to 1 and that all rooms are now in configuration 0. After this point we are ready to begin the old protocol. To summarize, here are the prisoners' protocols—where we use a new directive FLIP($*$,1) meaning "wait until you see a room not in configuration 0 or 1 and change that room to configuration 1."

Leader's Protocol:
REPEAT$(r - r_0 - r_1)$
  FLIP(∗,1)
REPEAT$(r - r_0 + (n-1) \cdot (r_0 + 1))$
  FLIP(1,0)
RUN OLD PROTOCOL;

Other Prisoners' Protocol:
REPEAT$(r_0 + 1)$
  FLIP(0,1)
RUN OLD PROTOCOL.

The proof that the preceding protocol works is relatively straightforward; we defer the details to the appendix.

### 3.2 All Switches Start OFF

Given the reduction proven in Lemma 4, we henceforth focus most of our effort on the case in which all rooms start in a specific known configuration—in particular, the case in which all switches start in the OFF position.

#### 3.2.1 Naïve Solutions

Some simple solutions come to mind quickly if we allow potentially large values of $s$. If we had infinitely many switches in each room, the prisoners could use them to encode (in English, translated by some means into binary) any information they want. They could use this to record the complete history of each room—the sequence of visits by prisoners, the names of these prisoners, the rooms previously visited by these prisoners, the configurations of those previously visited rooms, and any proofs of the Riemann Hypothesis that they have discovered in the meantime. Eventually the rooms will become distinguishable, based on the first time that some particular prisoner, say Alvin, visits a given room. Once this has happened, it is trivial for any prisoner to (eventually) verify from each room's history that every room has been visited by every prisoner.

To implement an analogous protocol with only finitely many switches, we note that it suffices to store in each room's configuration:

(a) for each prisoner, $p$, whether or not $p$ has visited the room;

(b) a number distinguishing each room, such as the $k$ such that this room was the $k$-th distinct room visited by Alvin.

We can encode (a) using $n$ switches per room (one for each prisoner), and can encode (b) with $r$ switches using a unary encoding. Hence we only need $s = n + r$ in order for the prisoners to have a winning protocol.

But using unary counters is somewhat inefficient. The value of $k$ can be encoded in binary using $\lceil \log_2(r+1) \rceil$ switches. Furthermore, if prisoners wait to indicate their visits

to a room until after it has been assigned identifiers by Alvin, they only need to store the number of prisoners who have visited the room—and anyone can declare once he verifies that each room has been visited by all $n$ prisoners. Such a counter can be implemented easily with $\lceil \log_2(n+1) \rceil$ switches; hence, we only require

$$s = \lceil \log_2(r+1) \rceil + \lceil \log_2(n+1) \rceil .$$

A slight optimization on the preceding protocol removes the need to identify the rooms, so long as the prisoners only track their presence in each room sequentially. In particular, if we identify $n+1$ distinguished configurations, denoted, $0, 1, \ldots, n$, where 0 is the all-OFF configuration, and assign each prisoner an identifier $i \in \{1, 2, \ldots, n\}$, we can use the following protocol:

Prisoner $i$'s Protocol $(i < n)$:
REPEAT($r$)
  FLIP($i-1$,$i$);

Prisoner $n$'s Protocol:
REPEAT($r$)
  FLIP($n-1$,$n$)
DECLARE.

When the preceding protocol is followed, for a room to be in configuration $i$, it must have been visited sequentially by prisoners, $1, 2, \ldots,$ and $i$. The declaration will not be made until every room is in configuration $n$, in which case each room has been visited by every prisoner. For there to be $n+1$ available configurations, there must be at least $\lceil \log_2(n+1) \rceil$ switches per room, and hence

$$s = \lceil \log_2(n+1) \rceil$$

suffices.

### 3.2.2 A Less Naïve Solution

The solutions just described store a large amount of data across the rooms; a lower-overhead solution attempts to run the original one-room protocol for each room sequentially. A simple version of such a protocol requires six distinct configurations, which we will call OFF (the initial room configuration), DONE, 0, 1, $0'$, and $1'$. During the running of the protocol:

- Rooms in the OFF configuration have not yet been modified.
- Rooms in the DONE configuration have been visited by all prisoners and will not be modified again.
- Configurations 0 and 1 are used to implement the one-room protocol discussed in Section 1.1.

- Configurations $0'$ and $1'$ are used to communicate to each prisoner that it is time to move on to the next room.

Formally we use:

Leader's Protocol:
REPEAT($r$)
  FLIP(OFF,0)
  REPEAT($n-1$)
    FLIP(1,0)
  FLIP(0,$0'$)
  REPEAT($n-1$)
    FLIP($1'$,$0'$)
  FLIP($0'$,DONE)
DECLARE;

Other Prisoners' Protocol:
REPEAT($r$)
  FLIP(0,1)
  FLIP($0'$,$1'$).

In the execution of this protocol, the leader selects a room in the OFF configuration, and changes it to the 0 configuration. The prisoners then run the one-room protocol in that room (ignoring all of the other rooms, which are still in the OFF configuration). The leader then flips that room to the $0'$ configuration, and the prisoners run through the one-room protocol in that room *again* to confirm that each prisoner has been in this room (using $0'$ and $1'$ instead of 0 and 1). After this, the leader puts that room in the DONE configuration and moves on to the next room. Note that the alternation between the 0/1 version of the one-room protocol and the $0'/1'$ version of the one-room protocol is necessary here as otherwise the follower prisoners will not know when they have switched rooms.

The protocol just described requires at least $s = 3$ switches per room. We do not provide a full analysis of this protocol here, as in Section 3.4 we discuss a refinement that gets by with only $s = 2$.

### 3.3 A Two-Switch Solution

The following is a relatively simple winning protocol for $s = 2$ that works for arbitrary $n$ and $r$. We name our four configurations (in some order) 0 (the initial configuration), 1, NEXT, and READY. The idea here is that instead of processing the rooms one at a time, we will process the *prisoners* one at a time.

In our protocol, we have at most one "active" prisoner at a time; this prisoner will verify that he has visited every room by first flipping all rooms from the 0 configuration to the 1 configuration, and then flipping them all back. We then need a way to pass the torch to the next active prisoner—and ensure that the prisoner who finished being active

is counted properly. In order to do this, the active prisoner will flip one room to the NEXT configuration, and the designated leader will flip that room to the READY configuration (incrementing a counter in the process). The next prisoner who has not yet been active and sees the room in the READY configuration changes that room's configuration to 0 and becomes active. We continue this process until all prisoners have a chance to be active and visit all rooms—and then be counted.

Formally, the protocol is as follows:

Leader's Protocol:
REPEAT($r$) \\ The Leader is the active prisoner.
  FLIP(0,1)
REPEAT($r$)
  FLIP(1,0)
FLIP(0,NEXT) \\ Sets a room to NEXT to indicate that
a new prisoner can become active.
REPEAT($n$) \\ Counts the number of active prisoners (including himself).
  FLIP(NEXT,READY)
DECLARE;

Other Prisoners' Protocol:
FLIP(READY,0) \\ Waits for a room in READY state
before becoming active.
REPEAT($r$)
  FLIP(0,1) \\ Visits all rooms.
REPEAT($r$)
  FLIP(1,0) \\ Resets all rooms.
FLIP(0,NEXT). \\ Signals the leader that he is done.

In particular we conclude:

**Theorem 5.** *There exists a winning protocol for the prisoners if there are $s = 2$ switches per room and all switches start in the* OFF *configuration.*

The formal proof that the protocol just described is winning is straightforward; hence, we defer it to the appendix.

*Remark* 6. Note that the protocol presented here satisfies the hypothesis of Lemma 4, as no prisoner other than the leader will change the configuration of any room until he has seen a room in the READY configuration. Thus with only $s = 2$ switches in each room, we have a winning protocol for an arbitrary known starting configuration.

### 3.4 Solving the Problem One Room at a Time

As we already noted, the solution we just presented is substantially different from our earlier three-switch solution. Indeed, whereas our two-switch protocol proceeded one prisoner at a time, our three-switch protocol solved the problem one room at a time, with

a single active room that is the only one changing configuration; the prisoners' protocols ensured that each one of them visited that room before changing the configuration of any other room. We might ask whether the one-room-at-a-time protocol can be made to work with only two switches—and in fact with some added complexity, it can be.

To begin, we provide different names for the configurations. We rename the four configurations 0 (the initial configuration), 1, UP and DONE.

At a high level, in our protocol, the leader selects rooms one at a time and plays a game toggling the chosen room's state between two possibilities with the other prisoners (similar to the one-room, one-switch solution), before putting that room in the DONE configuration. It is easy to see how this works for the first room. The leader puts that room in the UP configuration and each other prisoner flips UP to 1 once, while the leader flips it back from 1 to UP $n-1$ times. At the end of this sequence of reconfigurations, the leader flips the room from UP to DONE, establishing that all prisoners have visited the first room.

Naively, then the leader could take another room from the 0 configuration and change it to the UP configuration and he and the other prisoners could play the same game again. Unfortunately, this does not work so easily. The problem is that the non-leaders will not be able to distinguish between the second room being in the UP configuration and the *first* room being in the UP configuration. So the leader needs a way to signal that the first round is over.

To do this, we note that in the first round there is never simultaneously a room in the UP configuration *and* a room in the 1 configuration. So if a prisoner seems a room in UP followed by a room in the 1 configuration, then it must be the case that have seen the active room twice, with that room reconfigured in the interim. However, the active room is only reconfigured a limited number of times. Therefore, no prisoner will ever see—during the first round—a long sequence of a room in the UP configuration followed by a room in the 1 configuration followed by a room in the UP configuration and so on. This provides the leader a way to signal to the other prisoners that they have reached the second round. The leader does this by flipping all non-DONE rooms from the 0 configuration to the 1 configuration, and then flipping one of them to the UP configuration. The other prisoners will then eventually see a long sequence of alternating UP and 1 configurations, and thus know that the second round has started.

Unfortunately, this idea does not work if the prisoners are toggling between UP and 1 in the second round, as then the leader will see *many* rooms in the 1 configuration, which will prevent him from working with just a single room. This is solved by letting the prisoners toggle between UP and 0 in the second round instead of UP and 1—and each round after that, they must alternate between the two. We thus think of the protocol is having to alternating phases: a "0-phase" and a "1-phase."

A final slight complication is that this signaling procedure does not work for the last round. This is because there is only one room left and so the prisoners will not be able to see many alternating configurations between UP and 1, as there is only one non-DONE room. However, it turns out that we will not actually need the last round of the protocol, as the signaling stage in the *previous* round forces the prisoners to visit both of the last

two rooms in order to see the appropriate alternating sequence.

We present the protocols for the leader and the others in the case where $r$ is odd. The case where $r$ is even can be handled with a slight modification.

Leader's Protocol:
REPEAT$((r-1)/2)$
  FLIP(0,UP)  \\ Readies next active room, start of the 0-phase.
  REPEAT$(n-2)$
    FLIP(1,UP)  \\ Counts other prisoners.
  FLIP(1,DONE)  \\ Marks active room as done.
  REPEAT($r$ minus the number of rooms already configured to DONE)
    FLIP(0,1)  \\ Transition – readies rooms for the next phase.
  FLIP(1,UP)  \\ Readies next active room, start of the 1-phase.
  REPEAT$(n-2)$
    FLIP(0,UP)  \\ Counts other prisoners.
  FLIP(0,DONE)  \\ Marks active room as done.
  REPEAT($r$ minus the number of rooms already configured to DONE.)
    FLIP(1,0)  \\ Transition – readies rooms for the next phase.
DECLARE;

Other Prisoners' Protocol:
REPEAT$((r-1)/2)$
  REPEAT$(n)$  \\ Verifies that rooms are set up for the 0-phase.
    SEE(0)
    SEE(UP)
  FLIP(UP,1)  \\ Indicates visit to active room.
  REPEAT$(n)$  \\ Verifies that rooms are set up for the 1-phase.
    SEE(1)
    SEE(UP)
  FLIP(UP,0).  \\ Indicates visit to active room.

Note the SEE(0) and SEE(1) commands above: these make sure that each prisoner stays in step with all of the others. Waiting to see 0s before flipping UP to 1 guarantees that prisoners are in the 0-phase. The number of repetitions is necessary to ensure that prisoners are not just seeing alternations between 0 and UP in the active room. We present a full analysis of the protocol in the appendix.

### 3.5 One Switch Does Not Suffice

We now know that with two switches, there are multiple protocols that allow the prisoners to win their freedom, for arbitrary $n$ and $r$. We now show that one switch is *insufficient* as long as $n \geqslant 2$ and $r \geqslant 5$.

Given the sequence of rooms visited by prisoners and the actions that they take, we define the *observed history* to be the ordered sequence of events describing a particular

prisoner entering a room in some specified initial configuration and then leaving it in some specified configuration. For example, if the exercise starts with prisoner 1 visiting room 1 and changing the configuration from OFF to ON, and then prisoner 2 visiting and not changing the configuration, the observed history would look like this:

- Prisoner 1 enters a room in the OFF configuration and changes it to the ON configuration.
- Prisoner 2 enters a room in the ON configuration and leaves it in the ON configuration.

We say that a prisoner $p$ *owns* a room configuration $c$ at some particular point in time if he has visited all rooms that are in configuration $c$ at that point in time. And we next say that a prisoner $p$ *provably owns* a room configuration $c$ at some point in time if in all visit sequences with the same observed history, $p$ owns $c$ at that time.

**Lemma 7.** *A winning protocol for the prisoners will never declare unless all prisoners provably own all configurations.*

*Proof.* Suppose that after some sequence of visits, the prisoners declare without prisoner $p$ provably owning configuration $c$. Then there is some sequence of visits $\Sigma$ with the same observed history in which $p$ does not own $c$. Since the prisoners' behaviors (including their declarations and configuration changes) depend only on the observed history, this means that for the sequence of visits $\Sigma$, the prisoners will declare before $p$ has visited all rooms in configuration $c$. Thus the prisoners' protocol cannot be winning. □

In order to reason about the concept of provable ownership, we use the following lemma:

**Lemma 8.** *Whether a prisoner $p$ provably owns a configuration $c$ changes in exactly the following circumstances:*

- *Prisoner $p$ loses provable ownership of configuration $c$ when a room of a configuration not provably owned by $p$ is reconfigured to $c$ by some other prisoner.*
- *Prisoner $p$ gains provable ownership of a configuration $c$ when he visits the only room in configuration $c$ or the only room in configuration $c$ is reconfigured to some other configuration.*

Note that the number of rooms currently in each configuration can be inferred from the observed history. In particular, it can be determined when one of the situations in Lemma 8 has taken place by considering only the observed history.

The basic idea of the proof of Lemma 8 is that $p$ can only lose provable ownership of $c$ if there is some sequence of visits consistent with the observed history in which $p$ loses ownership of $c$. Similarly, $p$ can only gain provable ownership of $c$ if in some sequence of visits consistent with the observed history $p$ did not have ownership of $c$ and after the

extra visit, no matter how it is arranged to be consistent with the observed history, $p$ will gain ownership of $c$. We deferr the full argument to the appendix.

Next we declare a prisoner *finished* if under no circumstances will that prisoner ever again change the configuration of a room or declare. We note the following lemma about when a prisoner may become finished.

**Lemma 9.** *Under a winning protocol for the prisoners, no prisoner may become finished (under any valid sequence of room visits) before he provably owns all configurations at once.*

*Proof.* Assume for sake of contradiction that there is a winning protocol in which, under some sequence of revisits $\Sigma$, there is a prisoner $p$ who becomes finished before he provably owns all configurations. This means that there is some sequence of visits with the same observed history as under $\Sigma$ for which $p$ does not own all the configurations, and thus has not visited all rooms. We extend this sequence of visits arbitrarily until one of the prisoners declares (which they will do if each prisoner is led into each room sufficiently many times). We then remove from that extended sequence all room visits that $p$ made since he became finished. Since after this point, $p$ did not reconfigure any rooms, none of the other prisoners can distinguish these two visit sequences, and hence they will still declare. On the other hand, in this new visit sequence $p$ will not have visited all rooms, and the prisoners thus must have declared incorrectly. (This can be extended to a valid sequence of room visits in which the prisoners declare incorrectly by adding arbitrary visits after the point at which the prisoners declare.) □

We are now ready to prove our main result for this section:

**Theorem 10.** *There is no winning protocol when $s = 1$, $n \geqslant 2$, and $r \geqslant 5$.*

*Proof.* The basic idea of our proof is to construct, for any fixed protocol, a sequence $\Sigma$ of visits with the following properties:

- Until some prisoner becomes finished, no configuration with at least one room in that configuration is ever provably owned by more than one prisoner.
- Until some prisoner becomes finished, no prisoner provably owns any configuration with more than two rooms in that configuration.
- Each prisoner visits each room infinitely often.

We produce the desired sequence as follows. We say that we *extend our visit sequence directly* to mean that we execute the prisoner-room visit that has least recently occurred, with ties broken arbitrarily. We extend directly if:

1. Some prisoner is finished.
2. All rooms or all but one room are in the same configuration.
3. No prisoner provably owns any configuration.

Otherwise:

4. If there is no finished prisoner and there is some configuration $c$ with exactly two rooms in configuration $c$, some prisoner $p$ who provably owns $c$, and no other configuration and no other prisoner who provably owns any configuration: In this case, let $p'$ be a prisoner other than $p$. Since $p'$ is not finished, there is some sequence of visits that will cause him to either reconfigure or a room or declare. In particular, there is some sequence of 0s and 1s so that if $p'$ is led into rooms in those configurations in that order, then $p'$ will either reconfigure a room or declare at the end of that sequence. As there are currently rooms in both the 0 and 1 configurations, we can send $p'$ on such a sequence of visits, and thus do so.

We note that assuming our invariants hold, the above list of possibilities is exhaustive. We have left to verify that this visit sequence satisfies the stated invariants. In other words we claim that at all times, one of the following conditions is true:

1. One of the two possible configurations has at most one room in it, and no prisoner provably owns the other configuration, and no prisoners are finished.

2. There are at least two rooms in each configuration, and no prisoner provably owns any configuration, and no prisoners are finished.

3. One configuration is present in two rooms, and is provably owned by exactly one prisoner—but other than that, no prisoner provably owns any configuration, and no prisoners are finished.

4. Some prisoner is finished.

We show by induction that at least one of the four stated conditions always holds. The base case is straightforward, since at the start either Condition (1) or Condition (2) (or Condition (4), in the event that some prisoner never takes action under the protocol) must hold because $r \geqslant 5$.

If Condition (1) holds, then without loss of generality, there is at most one room in the ON configuration. Condition (1) will continue to hold until a second room is reconfigured into the ON configuration, by some prisoner $p$. At this time, by Lemma 8 no prisoner provably owns any configuration, except for $p$, who might provably own the ON configuration. Hence, either Condition (2) or Condition (3) is satisfied (unless some prisoner becomes finished, leading to Condition (4)).

If Condition (2) is satisfied, then a single visit cannot cause any prisoner to provably own any configuration; hence, after any visit either Condition (1) or Condition (2) is satisfied (unless some prisoner becomes finished, leading to Condition (4)).

If Condition (3) is satisfied, then any visit that does not reconfigure a room cannot change room ownership and so we remain in Condition (3). Otherwise, let $(p, c)$ be the unique pair of a prisoner $p$ who provably owns a configuration $c$, and let $p'$ be any other prisoner. If $p'$ reconfigures a room to configuration $c$, then no prisoner will provably own any configuration, leading to Condition (2) (unless some prisoner becomes finished, leading

to Condition (4)). If $p'$ reconfigures any room away from configuration $c$, then we will be in Condition (1) (again, unless some prisoner becomes finished, leading to Condition (4)). Since our procedure ensures that only prisoners other than $p$ will reconfigure rooms while Condition (3) holds, our invariants are maintained.

Finally, it is automatic that once Condition (4) is satisfied, it remains satisfied henceforth.

Note that if we extend our visit sequence directly infinitely often, then each prisoner visits each room infinitely often. This will happen under our constructed sequence, since after applying Case 4 of our sequence-generating procedure, we are left in one of the other cases, which will cause us to extend our visit sequence directly again. Thus, the sequence we have constructed is valid.

Morover, we note that if we run our constructed sequence, then no prisoner can provably own all configurations until after some other prisoner becomes finished. This means that any protocol either has some prisoner become finished before provably owning all of the configuration (which implies that their protocol is not winning by Lemma 9), or never leads to any prisoner provably owning all configurations. In the latter case, either the prisoners eventually declare (in which case their protocol is again not winning by Lemma 7), or they never do. In that last case, the prisoners never declare despite each of them being led into each room infinitely often, and so their protocol is not winning. □

*Remark* 11. We note that with some additional complications, it is possible to prove a similar negative result for as few as three rooms ($r = 3$). It is clear from the solution to the classic puzzle that that one switch ($s = 1$) suffices for a single room. Whether or not one switch suffices for two rooms ($r = 2$) is unclear.

## 4 Dimmer Switches

Throughout the arguments presented so far, we have found it useful to name our possible room configurations. The observant reader will notice that when given $s$ switches, our real constraint is that we have only $2^s$ possible configurations; thus, for example in the proof that two switches suffice, a total of four configuration names were used. As a generalization of this use of multiple switches, we can instead think of rooms as having a single "dimmer switch" with a number of possible configurations. The arguments so far show that if the dimmer switch has four or more configurations, then the prisoners have a winning protocol—and if the switch has two or fewer configurations, the prisoners do not (assuming that there are sufficiently many rooms and prisoners).

Whether or not the prisoners have a winning protocol with three configurations is an open question. However, we have found some protocols that seemingly come close. For instance, using only three configurations, it is possible to guarantee that each prisoner will eventually know that he has visited all rooms.

For this protocol, we call our room configurations ON, OFF and NEXT. All of the prisoners use the following protocol:

FLIP(NEXT,OFF)
REPEAT($r$)
  FLIP(OFF,ON)
REPEAT($r$)
  FLIP(ON,OFF)
FLIP(OFF,NEXT).

Furthermore, one of the prisoners, who will we call the leader, prepends the following command to his protocol:

FLIP(OFF,NEXT).

In the execution of this protocol, the leader will set one room to the NEXT configuration. Then some prisoner will see a room in the NEXT configuration and change it to OFF. That prisoner will then reconfigure all rooms to ON and then to OFF, before again changing one room to the NEXT configuration, after which some other prisoner begins reconfiguring rooms. Once each prisoner reaches the end of his protocol, he can conclude that he has visited every room. Unfortunately, no prisoner is able to tell whether the other prisoners have visited all the rooms yet.

We note that the protocol just described is essentially our one-prisoner-at-a-time solution—but without the UP configuration, the leader has no way of counting the number of other prisoners who have finished.

### 4.1 A Probability-1 Solution With 3 Configurations

While we do not know of a winning protocol for the case of a three-configuration switch, we have found a protocol almost as good. The following protocol *wins with probability 1*, by which we mean that

1. The prisoners will never declare incorrectly.

2. After any sequence of visits, there is always some possible sequence of future visits of bounded length after which the prisoners will declare.

In order to describe our protocol, we need to add another command to our language:

> OSCILLATE($c_1$,$c_2$): Upon entering a room in configuration $c_1$, reconfigure it to configuration $c_2$. Upon entering a room in configuration $c_2$, reconfigure it to configuration $c_1$. Continue this behavior until you have performed the former operation more times than you have performed the latter operation.

For this protocol, we label the prisoners $1, 2, \ldots, n$, and label the room configurations $0, 1, 2$, with 0 being the starting configuration. For $k \neq 1, n$, prisoner $k$'s protocol will be as follows:

Prisoner $k$'s Protocol ($k \neq 1, n$):
REPEAT($k-1$)
  FLIP(1,0)
  FLIP(2,1)
  FLIP(0,2)
SEE(1)  \\ Transitioning.
REPEAT($n+r-1$)  \\ Active, 0-phase.
  FLIP(0,1)
REPEAT($n+r-1$)  \\ 1-phase.
  FLIP(1,2)
REPEAT($n+r-1$)  \\ 2-phase.
  FLIP(2,0)
OSCILLATE(1,0)  \\ Transitioning.
REPEAT($n-k$)  \\ No longer active.
  FLIP(2,1)
  FLIP(0,2)
  FLIP(1,0).

The protocols for prisoners 1 and $n$ are similar. For prisoner 1's protocol, we remove the initial REPEAT loop, and the initial SEE command. To get prisoner $n$'s protocol, we replace the OSCILLATE command by a $DECLARE$ command, and remove the succeeding REPEAT loop:

Prisoner 1's Protocol:
REPEAT($n+r-1$)
  FLIP(0,1)
REPEAT($n+r-1$)
  FLIP(1,2)
REPEAT($n+r-1$)
  FLIP(2,0)
OSCILLATE(1,0)
REPEAT($n-1$)
  FLIP(2,1)
  FLIP(0,2)
  FLIP(1,0);

Prisoner $n$'s Protocol:
REPEAT($n-1$)
  FLIP(1,0)
  FLIP(2,1)
  FLIP(0,2)
SEE(1)
REPEAT($n+r-1$)
  FLIP(0,1)

REPEAT($n + r - 1$)
  FLIP(1,2)
REPEAT($n + r - 1$)
  FLIP(2,0)
DECLARE.

At a high level the execution of this protocol will work as follows. Each prisoner one at a time becomes *active* (when he is between the SEE command and OSCILLATE command in their execution). The active prisoner will turns all the 0s to 1s, then all the 1s to 2s then all the 2s back to 0s. Meanwhile, the other prisoners "resist" this change by flipping rooms in the opposite direction once per prisoner per step. There are two things worth noting about this. First, it guarantees that the active prisoner visits every room because the number of rooms flipped from 0 to 1 (or from 1 to 2 or from 2 to 0) is equal to the number of rooms plus the number of other prisoners flipping them in the other directions. Second, the other prisoners' resistance allows them to keep track of where in the protocol they are. To see this, note that while the active prisoner is reconfiguring 0s to 1s, another prisoner might reconfigure a 1 back to a 0, but he will not be able to execute their next command (flipping a 2 to a 1) until the active prisoner moves to their next phase (flipping 1s to 2s).

The one difficulty with this idea is how we switch from one active prisoner to the next. The issue is that the new active prisoner needs to wait until the previous one is finished turning 2s into 0s before he starts turning 0s into 1s. We note that if given access to a fourth configuration, UP, we could have the previous active prisoner reconfigure a room to UP when he is done, signaling to the new active prisoner that they are ready. This would give a protocol similar to the one-prisoner-at-a-time protocol. Otherwise, a simple way to signal that they are ready is to flip a room into the 1 configuration. This would work, except that the other prisoners are reconfiguring 1s to 0s and might destroy the signal before the new active prisoner sees it. This could be fixed if the old active prisoner reconfigured $r - 2$ rooms from 0 to 1; however, this introduces a new problem. In particular, with some other prisoners reconfiguring 1s to 0s and some reconfiguring 0s to 1s, the active prisoner will not be able to tell whether or not they are all finished, since if one 0 to 1 change were skipped *and* one 1 to 0 change were skipped, there would be no way to know. In order to fix this, we want to instead guarantee that the old active prisoner on net turns more 1s to 0s than 0s to 1s. However, he cannot do this immediately as he might simply flip many 0s to 1s and then flip them back without the new active prisoner seeing the signal. We fix this with the OSCILLATE command. This ensures that the old active prisoner keeps reconfiguring rooms back and forth between 0 and 1 until somebody (who must in this case be the new active prisoner) starts configuring 0s to 1s. We present the full analysis in the appendix.

## 5 A Probability-$\epsilon$ Solution With 2 Configurations

In the last section, we found a probability-1 protocol for three configurations. Unfortunately our impossibility proof for one switch—i.e., two configurations—does not generalize to protocols merely working with probability 1. Although we do not have a two-configuration protocol that succeeds with probability 1, we do have a protocol that satisfies another interesting condition.

We define a protocol to *win with probability* $\epsilon$ if the prisoners never declare incorrectly and do declare in some sequence of visits. The difference between probability 1 and probability $\epsilon$ is that in the latter case it may be possible to become stuck. Essentially, having a probability $\epsilon$ protocol means that you have a way of proving that everyone has been to every room under some sequence of room visits.

Given the two room configurations, 0 and 1, with 0 the starting configuration, we produce the following protocol that wins with probability $\epsilon$. Here, we have labeled the prisoners $p_0, p_1, p_2, \ldots, p_{n-1}$, and, intuitively, in the protocol execution that causes them to win, the prisoners act in order $p_0$ first then $p_1$ and so on:

$p_k$'s Protocol ($k \neq n-1$):
REPEAT($r+k$) \\ Startup phase, "started" after first command executed.
  FLIP(0,1)
REPEAT($r$) \\ Check phase.
  FLIP(1,0)
REPEAT($r$)
  FLIP(0,1)
REPEAT($r+k+1$) \\ Cooldown phase.
  FLIP(1,0); \\ Finished.

$p_{n-1}$'s Protocol:
REPEAT($r+n-1$)
  FLIP(0,1)
REPEAT($r$)
  FLIP(1,0)
REPEAT($r$)
  FLIP(0,1)
DECLARE.

For our analysis of this protocol, we define a few phases: Once a prisoner has executed his first FLIP(0,1) command, we declare him to have *started*. While in the first REPEAT loop, we say that a prisoner is in the *startup phase*. During the next two REPEAT loops, we say that prisoner is in the *check phase*. While in the last loop, we say he is in the *cooldown phase*—and after that, he is *finished*.

First, we note that there is some sequence of visits that cause the prisoners to declare. The required visit sequence is as follows: $p_0$ visits each room in sequence (changing them all to 1s and finishing the startup phase), then visits all the rooms again (changing them

back to 0 and finishing the second REPEAT loop), visiting all rooms a third time (setting them to 1, and finishing the check phase), and then visiting a fourth time (setting all rooms to 0 and finishing all but the last step of the cooldown phase). Then $p_1$ visits a room $R$ followed by $p_0$ visiting $R$. Prisoner $p_1$ flips $R$ to configuration 1 and back, with $p_0$ finishing his cooldown phase and $p_1$ executing the first command in his startup phase. Then $p_1$ visits each room in order four times. As before, this leaves all rooms in the 0 configuration with $p_1$ having finished all but the last two steps of his cooldown phase. Next, we have $p_2$ and $p_1$ alternate visits to some room twice. This finishes $p_1$'s cooldown phase and the first two steps of $p_2$'s startup phase.

Continuing this logic, we eventually reach a state in which all rooms are in the 0 configuration, $p_0, \ldots, p_{k-1}$ have finished, $p_k$ has completed all but the last $k+1$ steps of his cooldown phase, and none of $p_{k+1}, \ldots, p_n$ have started. We then have $p_k$ and $p_{k+1}$ alternate visits to room $R$ a total of $k+1$ times. This causes $p_k$ to finish his cooldown phase and for $p_{k+1}$ to complete the first $k+1$ steps of his startup phase. We then have $p_{k+1}$ visit all rooms in order four times, leaving them all in configuration 0, with $p_{k+1}$ having completed all but the last $k+2$ steps of his cooldown phase (or declaring if $k+1 = n-1$). This leaves us in the same situation as we began with, but for $k+1$. Continuing in this manner, the prisoners eventually reach the point at which $p_{n-1}$ declares.

Now, we now need to verify the more difficult assertion that the protocol only declares after all prisoners have visited all rooms. Most of our argument is based on a single fact: At any time during the execution of the protocol, the difference in the total number of FLIP(0,1) commands executed by all prisoners and the total number of FLIP(1,0) commands executed by all prisoners is between 0 and $r$ inclusive (as this difference is the number of rooms currently in configuration 1). For each prisoner, we thus define the *imbalance* to be the difference in the number of FLIP(0,1) commands he has executed and the number of FLIP(1,0) commands he has executed. By construction, the sum of all prisoner's imbalance is between 0 and $r$ inclusive. We note the following easily verified facts about prisoners' imbalances:

1. After a prisoner starts, his imbalance is positive until his cooldown phase.

2. A prisoner's imbalance is non-negative until that prisoner is finished, at which point that prisoners in balance becomes $-1$.

3. At the end of a prisoner $k$'s startup phase and at the end of a prisoner $k$'s check phase, that prisoner's imbalance is $r+k$.

Note that Conditions (2) and (3) together imply that $p_k$ cannot finish his startup phase until at least $k$ other prisoners have finished. Since a prisoner must end his startup phase before finishing, $p_0$ is the only prisoner who can finish before any other. Similarly, $p_1$ is the only prisoner that can finish after only $p_0$ has finished. Continuing with this logic, we conclude that if the prisoners all finish, then they must do so in the order $p_0, p_1, p_2, \ldots$. Additionally, $p_k$ cannot complete his startup phase until $p_0, \ldots, p_{k-1}$ have finished. Note therefore, that when $p_k$ completes his startup phase or completes his check

phase, only $p_0, \ldots, p_{k-1}$ can have finished and that all other prisoners must have non-negative imbalance. Since $p_k$ has imbalance $r + k$ and the total imbalance is at most $r$, this means that it must be the case that $p_0, \ldots, p_{k-1}$ have imbalance $-1$ (which implies that they have finished) and that $p_{k+1}, \ldots, p_{n-1}$ must have imbalance 0 (which means that they have not started). This means that no other prisoners reconfigure any rooms during $p_k$'s check phase. In particular, it means that during the first REPEAT loop of $p_k$'s check phase, $p_k$ must flip every room from 1 to 0 (and thus must visit every room). Therefore, every finished prisoner must have visited every room. Furthermore, when the prisoners declare, $p_{n-1}$ is finished, which requires that $p_0, \ldots, p_{n-1}$ have all finished. Thus, whenever the prisoners declare under our protocol, we know that every prisoner has visited every room.

*Remark* 12. We note that although the prisoners never declare incorrectly here, it is very easy for them to get stuck. The proof shows that in order for the prisoners to declare, it must be the case that $p_k$ does not start until all of $p_0, \ldots, p_{k-2}$ have finished. Of course if $p_k$ (for some $k \geqslant 1$) is the first prisoner to visit any room, then all of the rooms will be reconfigured to the 1 configuration before any prisoner has finished his startup phase, and there will be no way to make further progress.

# 6 Corner Cases and Related Problems

We close by discussing a number of special cases, in which sharper results can be obtained, along with some variants on our problem.

## 6.1 2 Rooms, 3 Configurations

We have a three-configuration solution for the special case of $r = 2$. For this solution, we call our configurations UP, ON, and OFF—with OFF representing the initial configuration. We have a single leader, whose protocol is as follows:

Leader's Protocol:
FLIP(OFF,UP)
REPEAT($n-1$)
  FLIP(ON,OFF)
DECLARE.

All other prisoners use the following protocol:

Other Prisoners' Protocol:
SEE(UP)
FLIP(OFF,ON).

Essentially, after the leader produces a single room in the UP state, the prisoners execute the standard one-room protocol in the other room, with the proviso that they do nothing until they have seen the room in the UP state.

*Remark* 13. The idea just described also provides us with a somewhat silly protocol for $r = 3$ with four configurations.

### 6.2 Small $n$

In Section 3.2.1, we described a solution using $n+1$ configurations in which prisoner $k$ would change all the rooms from configuration $k-1$ to configuration $k$. Although this is somewhat inefficient for $n \geqslant 3$, it provides new solutions when $n = 1$ or $n = 2$.

### 6.3 Unknown Starting Configuration with Infinitely Many Room Configurations

We note that the proof of Theorem 3 actually requires that each room has only a finite number of possible configurations—as it happens, this is actually necessary. In particular, as we show now, if there are infinitely many configurations, then there *is* a protocol that wins under arbitrary starting configurations.

For simplicity, we assume that there are countably infinitely many configurations and that these configurations correspond to finite-length alphanumeric strings, thought of as writing on the walls of the room. The prisoners' protocol here is actually fairly simple. Upon entering a room, each prisoner appends his name to that room's transcript, followed by the number of rooms that prisoner has visited so far. We claim that with this simple protocol, eventually some prisoner will have enough information to be able to conclude that each prisoner has visited every room.

To start the analysis, we note first that eventually all $r$ rooms will become distinguishable from each other. In particular, if a prisoner ever sees $r$ rooms for which for no pair of rooms is the transcript of one a prefix of the transcript of the other, these rooms must all be distinct (as the transcript of a room can only be modified by appending new text). To show this, we consider some particular prisoner, Barry. Upon visiting his $k$-th room, he will append "Barry$k$" to the transcript of that room. Now some rooms may have strings of the form "Barry$m$" in their initial transcripts, but since these initial transcripts are finite, there is a maximum value of $m$ that ever appears. Call $M$ the largest such value of $m$. Once Barry makes his $k$-th visit to any room for any $k > M$, his text "Barry$k$" in that room (followed up by another prisoner's name rather than more digits for the number) will never appear in any other room. Once he has made such visits to all $r$ rooms, the rooms will thereafter be distinguishable.

So eventually, a prisoner will see rooms with transcripts $T_1, T_2, \ldots, T_r$ none of which is a prefix of any other. At some later point, this prisoner will see rooms with transcript $T_i$ followed by some list of names and numbers that include the names of every prisoner. At that point, it must be the case that every prisoner has visited the room that had transcript $T_i$. Once any prisoner has seen this occur for all $i$ with $1 \leqslant i \leqslant r$, that prisoner can safely declare.

### 6.4 Symmetric Strategies

One interesting modification to our problem would be the additional requirement that the prisoners use identical protocols. Essentially none of our protocols satisfy this property.

For the one-room case Winkler [8, pp. 130–131] gives a solution with symmetric protocols and two switches per room, with reference to [3]. In general, the problem with identical protocols seems to be much harder (although if the prisoners are allowed to specify a starting configuration, they can start with a single room in a special configuration and structure their protocol so that the first person who sees that configuration becomes "leader").

### 6.5 Repeated Entries

A substantially easier modification to the rules requires that each prisoner visit each room $\ell \geqslant 1$ times before the prisoners declare. This problem is not significantly more difficult than the original, as several of our protocols can be easily modified to accommodate it. For example, our original two-switch solution can be modified so that each prisoner flips all rooms on and then all rooms off $\ell$ times. Essentially all of the protocols presented in this paper have similar modifications.

### 6.6 Multiple Declarations

Another modification of the problem is obtained by requiring that all prisoners declare at some point after they have all visited every room. This can be done with four-state switches using a slight modification of the protocol given in Section 3.4: the leader puts a room in the UP state to denote that it is time to declare. In particular, we append

FLIP(DONE,UP)

to the leader's protocol, and append

SEE(UP)
DECLARE

to all other prisoners' protocols.

### 6.7 Forced Flipping of Switches

Another modification would be the require that upon each visit to a room, a prisoner *must* reconfigure that room's state in some way. It is not clear that any of our existing protocols generalize to this alternate setting directly, but any protocol for the original problem can be extended to this case by doubling the number of room configurations. This is done by replacing each configuration with a pair of new configurations, which are treated as equivalent for purposes of the protocol, except that a prisoner can toggle between them if no other configuration change is desired.

Put another way, we can accomplish this by adding an additional switch to each room. This switch will have no effect on the rest of our protocol save that any prisoner visiting a room will always flip that switch in addition to whatever else he was going to do.

### 6.8 Limited Reconfiguration

More generally, the warden could impose essentially arbitrary restrictions on which configurations can be reconfigured into which other configurations in a single visit. We cannot say much about the problem in this level of generality, and leave it to prisoners craftier than us.

**Acknowledgements**

The authors appreciate helpful comments from Mike Nizza and seminar audiences at Harvard and the 2021 Joint Mathematics Meetings.

## A Proofs Omitted from the Main Text

*Proof of Lemma 4.* In order to show that the provided protocol works, we will need to verify that:

1. No prisoner begins to run his old protocol before the leader completes his first REPEAT loop.

2. Between the end of the leader's first REPEAT loop and when he begins to run his old protocol, all rooms are in configuration 0 or 1.

3. When the leader begins to run his old protocol, the other prisoners have all started to run their old protocols, but have ignored all rooms they have seen since they started doing so, and all rooms are in configuration 0.

4. Eventually the leader will reach the RUN OLD PROTOCOL step.

Once we have proven these statements we will be done, since Statement (4) implies that eventually the leader begins to run his old protocol, and Statement (3) implies that at that time

- all non-leader prisoners are acting as if they were at the start of their old protocols and

- all rooms are in state 0.

Therefore, from that point in time, it is as if all prisoners were running the old protocol with the correct starting configuration. Since the warden must send each prisoner into each room arbitrarily many times from that point (in order for the sequence of visits to be valid), the correctness of the old protocol implies the correctness of the new one.

Statement (1) holds because each time a prisoner flips a 0 to a 1, the number of rooms in configuration 0 decreases. The only way this number can increase is either after the leader finishes his first REPEAT loop or after some other prisoner begins running his old protocol. Since the number of starting 0s is less than the number that must be changed,

no non-leader can begin to run his old protocol until some room changes to configuration 0. Therefore, the first prisoner to begin to run his old protocol must do so after the leader completes his first REPEAT loop.

For Statement (2), we first note by Statement (1) that until the leader completes his first REPEAT loop, no other prisoner begins running his old protocol. Therefore, until that time, the only way that the number of rooms not in state 0 or 1 changes is that the number decreases by 1 each time the leader executes his FLIP(∗,1) command. Therefore when the leader finishes his first REPEAT loop, there are no such rooms remaining. Thus, at the end of the leader's first REPEAT loop, all rooms are in configuration 0 or 1. We note that between that time and the end of the leader's second REPEAT loop, the only way that a room can be put into a configuration other than 0 or 1 would be if another prisoner who has started to execute his old protocol does so. However, by assumption, non-leader prisoners who are executing their old protocols will not pay attention to any rooms (much less change their configurations) until they have seen one in a configuration other than 0 or 1. However, there is no way that a (non-leader) prisoner can be the first to do this, as such a prisoner would need to have first seen a room in a state other than 0 or 1, which must have been produced by some even earlier prisoner.

Statement (3) is proven by considering the number of rooms in configuration 0. This number increases by 1 when the leader runs a FLIP(1,0), decreases by 1 when another prisoner runs a FLIP(0,1). Since Statement (2) implies that none of the non-leaders have reconfigured rooms or executed any commands since starting to run their old protocols, these are the only ways the number of rooms in configuration 0 can change until the leader starts to run his old protocol. The number of rooms in configuration 0 starts at $r_0$. In order for the leader to begin the old protocol, this number must increase $r - r_0 + (n-1)\cdot(r_0+1)$ times. However since the number of rooms in configuration 0 can never exceed $r$, this is only possible if it has decreased at least $(n-1)\cdot(r_0+1)$ times. This many decreases can happen only if each of the other prisoners run their FLIP(0,1) the full $r_0+1$ times and begin running their old protocols.

Statement (4) is a liveness condition that can be proven by looking carefully at the analysis thus far. First, we show that the leader will eventually finish his first REPEAT loop. This is because he executes a FLIP(∗,1) once whenever he enters any of the rooms that did not start as 0 or 1 for the first time. Since there are $r - r_0 - r_1$ of these, eventually he has visited all of them and completed the loop. Next, we note that there will never be a time at which no prisoner can make progress on his REPEAT loop. Indeed, our analysis thus far shows that if all of the non-leaders have completed their REPEAT loops, there will be as many rooms in configuration 1 as iterations left in the leader's loop. Therefore, if the leader enters the appropriate room, he will make progress through his protocol. If both the leader and some non-leader have FLIPs to perform, then either there is a 1 for the leader to FLIP to a 0 or a 0 for the non-leader to FLIP to a 1. As validity guarantees that every prisoner will be sent to every room as many times as we need, if the prisoners wait long enough, then eventually one of them will complete one of their FLIP commands—and this can only happen a bounded number of times before everyone starts to run their old protocols. □

*Proof of Theorem 5.* Recall that we are analyzing the protocol presented in Section 3.3.

In order to analyze this protocol, we introduce some terminology. We say that a prisoner is *exhausted* if he is either a non-leader who has reached the end of his protocol, or is the leader and has completed the first five lines of his protocol. We define an *active* prisoner to be one who is either a non-leader who has completed the first line of his protocol but is not exhausted, or a leader who is not exhausted. We define a prisoner to be *waiting* if he is neither active nor exhausted. It is clear that each prisoner progresses sequentially from waiting to active to exhausted (except for the leader, who is never waiting).

The correctness of our protocol depends heavily on the following invariant. At all times exactly one of the following holds:

- all rooms are in either the 0 or 1 configuration, and there is exactly one active prisoner; or
- all rooms are in the 0 configuration, except for a single room in the NEXT or READY configuration, and there is no active prisoner.

We show that our invariant holds by induction. It is easy to check that the first condition holds in the initial configuration. Now, when a prisoner becomes active, all rooms are in or are changed to the 0 configuration. As this prisoner remains active, no other prisoner will alter room configuration because non-active prisoners ignore rooms in state 0 or 1. Therefore, while active, this prisoner will change all rooms to the 1 configuration and then change all rooms back to 0 before becoming inactive. As this prisoner becomes inactive, he sets one room to the NEXT state, maintaining our invariant. The invariant continues to hold when the leader reconfigures this room from NEXT to READY (and this is the only reconfiguration that can be performed by any of the inactive prisoners); this new state then holds until the next prisoner becomes active.

In order to show that our protocol never declares incorrectly, we observe two more properties. The first property is that exhausted prisoners have visited all rooms; this follows from the preceding analysis and the fact that exhausted prisoners must have once been active. Second, we claim that at the end of the $k$-th iteration of the final repeat loop on of the leader's protocol, there are exactly $k$ exhausted prisoners. We prove this by noting that our protocol cycles through the following three stages:

1. There is an active prisoner.
2. There is a room in the NEXT configuration.
3. There is a room in the READY configuration.

The claim follows from the fact that we increment the number of exhausted prisoners exactly when we transition from Stage 1 to Stage 2, and that we increment the counter on the repeat loop exactly when we transition from Stage 2 to Stage 3. Together, our claims imply that declaration occurs only once all $n$ prisoners are exhausted—and thus only when each prisoner has visited every room. Thus, the protocol never declares incorrectly.

To prove that the protocol always terminates, we show that it always eventually progresses to the next stage (or the leader declares). Since we can only transition from Stage 2 to 3 a total of $n$ times, this proves that the protocol will eventually declare. To show that the protocol always progresses from Stage 1, we observe the following. As a prisoner becomes active, all room are in the 0 configuration. Since no other prisoner will alter any configurations during this stage, the active prisoner will switch every room to the 1 configuration as he visits that room. He will then switch each room to the 0 configuration as he visits it. He will then switch the next room he visits to the NEXT configuration and move the protocol to Stage 2. Stage 2 will always progress to stage 3 when the leader finds the room in the NEXT configuration. Stage 3 will progress to Stage 1 when any waiting prisoner reaches the room in the READY configuration. This will always happen eventually—unless there are no waiting prisoners, which only happens when all prisoners are exhausted—at which point the leader has reached the last line of his protocol and is ready to declare. □

*Analysis of the Protocol Described in Section 3.4.* To show that our protocol is winning, we need some definitions. During some parts of the protocol, the leader is flipping rooms between 0 and 1; we call these periods *transition phases*. When not in a transition phase, some rooms are in the DONE configuration and are called *finished*. Otherwise, either all but one of the unfinished rooms are in the 0 configuration or all but one of the unfinished rooms are in the 1 configuration; we call these periods the 0-phase and 1-phase respectively, and they correspond to the sections of the leader's protocol in which he is running FLIP(1,UP) and FLIP(0,UP) respectively (as indicated). During one of these phases there is one unfinished room, which we call the *active room*, which is toggled between UP and 1 in the 0-phase or between UP and 0 in the 1-phase. The remaining rooms, are not reconfigured at all during this phase.

We have left to prove that the description just given holds, and that the protocol works. In particular, we need to prove the following:

1. During a 0-phase, all rooms are in the 0 or DONE configuration except for a single active room in the 1 or UP configuration. Likewise, during a 1-phase all rooms are in the 1 configuration or the DONE configuration, except for a single active room in the 0 or UP configuration. Furthermore, at the start of this phase, the active room is in the UP configuration.

2. During the 0-phase, no non-leader is at the FLIP(UP,0) line of their protocol, and during the 1-phase no non-leader is at the FLIP(UP,1) line.

3. During the 0-phase, each non-leader will flip the active room from UP to 1 exactly once and will reconfigure no other rooms. During the 1-phase, each non-leader will flip the active room from UP to 0 exactly once and will reconfigure no other rooms.

4. During the transition phase, each room that has ever been an active room is in the DONE configuration. And the leader reconfigures all other rooms from 0 to 1 or from 1 to 0 while no other room reconfigurations take place.

We show that these invariants hold by induction. We note that the statements about the 0-phase and 1-phase are symmetric, so we will only prove the former and under the assumption that these invariants hold for all previous phases.

We begin by showing that at the start of the 0-phase all rooms are in the DONE or 0 configuration with one in the UP configuration. This clearly holds after the first line or the leader's protocol. Otherwise, invariant 4 implies that the leader reconfigured all non-DONE rooms to 0 in the previous transition phase and reconfigured one of the 0s to UPs at the start of the phase. We also note that at the start of the phase, each non-leader is between their FLIP(UP,0) command and their FLIP(UP,1) command. This is true at the start of the protocol, and on later iterations, by assumption they executed their FLIP(UP,0) in the last 1-phase and have not executed FLIP(UP,1) since.

From here, we claim that during the 0-phase, no room other than the active room is reconfigured. This is because reconfiguring a different room would require reconfiguring a room not in the 1 or UP configuration. The leader does not do this until the transition phase. The non-leaders will not do this until they have seen a 1 followed by an UP at least $n$ times; we claim that no non-leader sees this during the 0-phase. Indeed, the first non-leader to see a 1 followed by an UP at least $n$ times must see the active room in these configurations (as no other room is in either the 1 or UP configuration during the 0-phase). This in turn would imply that the leader must have reconfigured it from 1 to UP at least $n$ times (since no non-leader is reconfiguring in this direction). However the leader reconfigures in this way at most $n-1$ times during the phase.

Next we show that during the 0-phase, the active room will be reconfigured between the 1 configuration and UP configuration $n-1$ times. We know that the leader will not progress with his protocol until he has reconfigured it from 1 to UP a total of $n-1$ times. Furthermore, each of the $n-1$ non-leaders will have an opportunity to reconfigure the active room from UP to 1 once during this phase. We claim that if the active room has not been reconfigured between UP and 1 the full $n-1$ times, that it will eventually (assuming that each prisoner is lead into each room enough times) be reconfigured more. If the active room is currently in the 1 configuration, the leader will eventually see it there and reconfigure it. If the active room is currently in the UP configuration and has been flipped from UP to 1 fewer than $n-1$ times, there is at least one non-leader who has not reconfigured the active room during this phase. This prisoner may still have some SEE(0) and SEE(UP) commands to execute before his FLIP(UP,1) command. However, if no other prisoner reconfigures the active room in the interim, he will eventually see the active room in the UP configuration followed by one of the unfinished rooms in the 0 configuration enough times to finish his SEE commands. His next visit to the active room will cause it to be reconfigured.

The preceding argument implies that the protocol will eventually progress from the 0-phase to the next transition phase. We note that it also implies that every prisoner visits the active room before this transition. This is because the leader must have reconfigured the active room from 1 to UP a total of $n-1$ times. This is only possible if it was reconfigured from UP to 1 this many times. However, each non-leader can only do so once. Therefore, by the end of the phase, each non-leader must have reconfigured the

active room from UP to 1.

We now discuss the transition phases. We consider the case of a transition phase after a 0-phase; the case of a transition phase after a 1-phase is analogous. At the start of the transition phase, the leader has just reconfigured the previous active room to the DONE configuration, and all other rooms are in the DONE or 0 configurations.

We next show that no non-leader reconfigures any room during this transition phase. This is because a non-leader will only reconfigure rooms found in the UP configuration. However, during the transition phase, no room is in the UP configuration, nor does the leader reconfigure any room into the UP configuration. During this transition phase, the leader does reconfigure a number of 0 rooms to 1 equal to the number of rooms in the 0 configuration at the start of the phase. This is because at the start of the phase every room is in the 0 or DONE configuration, so this number should be $r$ minus the number of rooms in the DONE configuration. However, since only the leader reconfigures rooms into the DONE configuration and since no prisoner reconfigures rooms out of the DONE configuration, the number of iterations in the leader's REPEAT loop is the number of rooms in the 0 configuration. Since no rooms are being reconfigured by other prisoners, the leader will reconfigure each 0 room to 1 as he finds it, and then reconfigure the next 1-room to UP, starting the next phase. We note that this leaves the rooms in the configurations needed at the start of the 1-phase.

The preceding analysis shows that our invariants hold and that this protocol will eventually terminate. We have left to show that at the end of the protocol that every prisoner will have visited every room. First, as we have already discussed, the active room in any 0- or 1-phase must be visited by every prisoner before we progress. Since the rooms in the DONE configuration are exactly the previously-active rooms, this means that every room in the DONE-configuration was visited by every prisoner. We note that exactly $r-1$ rooms are put into the DONE-configuration by the end of the protocol. This leaves one remaining room to consider.

We note that the remaining room was the unique room in the 1 configuration during the last 1-phase. The leader must have visited this room because the leader must visit every non-finished room in every transition phase. To show that non-leaders visited this room, we note that each non-leader must have reconfigured the active room during the last 1-phase. However, in order to do this, they must have seen $n$ alternations between rooms in the 1 and UP configurations. But we have already shown that they can have seen at most $n-1$ of these alternations in the previous 0-phase and transition phase. Therefore, they must have seen this final room at least once during the last 1-phase; this completes our argument. □

*Proof of Lemma 8.* Clearly $p$ can lose provable ownership of a configuration $c$ only if there is some possible sequence of visits with the same observed history in which he loses ownership of $c$. This can only happen if some other prisoner reconfigures a room that $p$ has not visited into configuration $c$—which, in turn, can only happen when that other room is in some configuration $c'$ that $p$ does not own, which can happen only if $p$ does not provably own $c'$. Thus, $p$ can only lose provable ownership of $c$ if some other prisoner reconfigures a room from a configuration $c'$, not provably owned by $p$, to configuration $c$.

On the other hand, if $p$ does not provably own $c'$ and some other prisoner reconfigures $c'$ to $c$, then there is some sequence of visits with the same observed history in which $p$ did not own $c'$ before this visit. In this sequence $p$ had not visited all rooms currently in configuration $c'$, and thus without changing the observed history, we may have the last visit reconfigure a room that $p$ has not visited to configuration $c$. In this alternative visit sequence, we have the same observed history, but $p$ does not own $c$ at the end of it. Therefore, after such an event $p$ no longer provably owns $c$.

On the other hand, if $p$ visits the only room currently in configuration $c$, or if the only room currently in configuration $c$ is reconfigured into another configuration, it is clear that $p$ owns $c$ after this takes place. It is also easy to see that it is possible to determine when either of the situations just described has taken place purely by considering the observed history. Therefore, under either of these situations, if $p$ did not previously provably own $c$, he gains such ownership. However, if $p$ did not provably own $c$ before and some visit caused to $p$ to gain provable ownership of $c$, then there must have previously been some sequence of visits with the same observed history in which $p$ did not own $c$ but for which any additional visit with the same observed data would cause $p$ to own $c$. This can happen only if the last room in configuration $c$ that $p$ had not yet visited is either visited by $p$ or reconfigured to another configuration. However, if more than one room was in configuration $c$ before this last visit, then the same observed history will be possible so that the last visit is not to the final room in configuration $c$ not visited by $p$ (either the last visit is to a room in another configuration or it could be made to be to a different room in configuration $c$ without altering the observed history). Therefore, $p$ gains provable ownership of $c$ only if a visit is made to the unique room in configuration $c$ either by $p$ or by another prisoner who reconfigures it to a different configuration. □

*Analysis of the Protocol Described in Section 4.1.* We begin by introducing some terminology: A prisoner executing his SEE or OSCILLATE commands is said to be *transitioning*; a prisoner between those commands in his protocol is called *active*. If a prisoner is in his REPEAT loop, then he is in his 0- 1- or 2-phase as noted in the protocol comments. We make the following claims about the execution of protocol:

1. There is never more than one active prisoner at a time.

2. The prisoners become active in order.

3. During a 0-phase, or while there is no active prisoner, all rooms are in configuration 0 or 1.

4. During a 1-phase, all rooms are in configuration 1 or 2.

5. During a 2-phase, all rooms are in configuration 2 or 0.

6. During a prisoner's 0-phase, each other prisoner is on one of his FLIP(1,0), FLIP(2,1), or OSCILLATE(1,0) commands.

7. During a prisoner's 1-phase, each other prisoner is on one of his FLIP(2,1) or FLIP(0,2) commands.

8. During a prisoner's 2-phase, each other prisoner is on one of his FLIP(0,2) or FLIP(1,0) commands.

9. At the start of the protocol and when a prisoner first switches from active to transitioning, all rooms are in the 0 configuration and all other prisoners are executing a FLIP(1,0) or SEE(1) command, with at most one prisoner in the latter state.

10. At the start of a 1-phase, all rooms are in the 1 configuration, and all non-active prisoners are executing a FLIP(2,1) command.

11. At the start of a 2-phase, all rooms are in the 2 configuration, and all non-active prisoners are executing a FLIP(0,2) command.

To show that the preceding statements hold, we assume that they do at the end of a given phase, and show that they still do at the end of the next phase. The analysis for a 1-phase is easy. At the start of a 1-phase, all rooms are in the 1 configuration and all non-active prisoners are executing FLIP(2,1) and the active prisoner is executing his repeat loop of FLIP(1,2). It is clear that no non-active prisoners will be able to execute their next commands (FLIP(0,2)) until the active prisoner has moved on to the next phase. The active prisoner will not be able to do this until he has flipped $n + r - 1$ rooms from 1 to 2. As there are only $r$ rooms available, he cannot do this unless a total of $n - 1$ rooms (with multiplicity) are flipped from 2 back to 1. This can only happen if each other prisoner completes his FLIP(2,1) command. At the end of this, all rooms will have been changed to state 2 and all other prisoners will be on their FLIP(0,2) commands showing that the state at the start of the next phase is as desired. We also note that this implies that the active prisoner visits every room before the end of this phase.

The analysis for a 2-phase is similar. The one difference is that we note that exactly one prisoner ends with a SEE(1) command rather than a FLIP(1,0) command. This is because each pre-transitioning prisoner executes exactly one command per phase. Therefore if the $k$-th prisoner just finished being active, exactly the $(k + 1)$-st prisoner is on his SEE(1) command.

The analysis for a 0-phase (actually starting from where the previous active prisoner switched to being transitioning) is slightly more complicated. One prisoner started at his SEE(1) command; we call them active, although technically he is transitioning until he sees a room in configuration 1. After seeing that room, he will try to flip $n + r - 1$ rooms from 0 to 1 before moving on to the next phase. Meanwhile, the other prisoners are either executing FLIP(1,0) or OSCILLATE(1,0). We note that after that command, the other prisoner will try to execute FLIP(2,1), but will be unable to do so, as no room will be in the 2 configuration until the next phase. The active prisoner needs to flip $n + r - 1$ rooms from 0 to 1. There are a total of $r$ rooms, initially in the 0-configuration. It will be possible to reconfigure rooms into the 1 configuration $n + r - 1$ times only if the other prisoners in aggregate reconfigure rooms from 1 to 0 at least $n - 1$ times more often than

he reconfigures rooms from 0 to 1. We note that each other prisoner may do so on net at most 1 time. Therefore, we can only transition to the next phase once all non-active prisoners have executed their FLIP(1,0) or OSCILLATE(1,0) commands (but not their next FLIP(2,1) commands) and all rooms have been reconfigured to 1.

From the above analysis, we note that each non-active prisoner executes exactly one command per phase, and that each active prisoner visits all rooms before becoming non-active again. From this it is easy to see that if the $n$-th prisoner declares, then every prisoner must have been active at some point, and therefore every prisoner must have visited every room at least once.

We have left to show that this happens with probability 1. For this we will show that from any reachable state, there is always some continuation that causes the protocol to progress to the next phase. For example, starting at the beginning of a 1-phase, at any point until the next phase, the number of rooms in the 1 configuration plus the number of non-active prisoners on their FLIP(2,1) commands is always the number of iterations left on the active prisoner's loop. This means that there is always either a 2-room to visit for one of the non-active prisoners still on his FLIP(2,1) command or a room in the 1-configuration for the active -prisoner to visit. Therefore, if each prisoner visits each room infinitely often, eventually the non-active prisoners will complete their FLIP(2,1) commands and the leader will complete his loop and the phase will end.

The analysis for the 2-phase is analogous. The analysis starting after the end of the 2-phase is slightly more complicated. First, we show that there is always a way for the next active prisoner to execute his SEE(1) command. This is because until that prisoner does so, the previously active prisoner will be executing his OSCILLATE(1,0) command. This in turn is because he cannot end the OSCILLATE(1,0) command until he has reconfigured more rooms from 1 to 0 than from 0 to 1, but until the next prisoner becomes active, no other prisoner is reconfiguring 0 to 1. While the previous active prisoner is oscillating, there will always be the possibility that he reconfigures a room to 1, which is then seen by the next active prisoner. Once this has happened, we claim that (until the end of the phase) it will always be possible for the active prisoner to reconfigure a 0 to a 1 or a non-active prisoner to complete his current command. Since these can only happen a bounded number of times during the phase, there will always be a way to proceed to the next phase. To show this, if the OSCILLATE command has not been completed, it will always be possible for that prisoner to reconfigure some room to a 0 so that the active prisoner can later reconfigure it to a 1. If the OSCILLATE command has been completed, then the number of rooms in the 0 configuration plus the number of prisoners who have not completed their FLIP(1,0) commands is the number of remaining iterations in the active prisoner's repeat loop. From this it is easy to see that there is always either a room in the 0 configuration for the active prisoner to flip to 1, or a room in the 1 configuration for some non-active prisoner (still on his FLIP(1,0) command) to reconfigure to 0. This shows that it is always possible to make progress, completing our argument. □

## References


[1] Joe P. Buhler and Elwyn R. Berlekamp. Puzzle 4. *The Emissary*, 5(2):11, 2002.

[2] Paul-Olivier Dehaye, Daniel Ford, and Henry Segerman. One hundred prisoners and a lightbulb. *The Mathematical Intelligencer*, 25(4):53–61, 2003.

[3] Michael J. Fischer, Shlomo Moran, Steven Rudich, and Gadi Taubenfeld. The wakeup problem. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, pages 106–116, 1990.

[4] Scott Duke Kominers. Kominers's Conundrums: The warden has a brainteaser. *Bloomberg Opinion*, April 25, 2020.

[5] Scott Duke Kominers, Paul Kominers, and Justin Chen. Problem S08-2. *The Harvard College Mathematics Review*, 2(1):93, 2008.

[6] Scott Duke Kominers, Paul Kominers, and Justin Chen. Problem S08-2 (corrected). *The Harvard College Mathematics Review*, 2(2):96, 2008.

[7] Car Talk Radio Show. Prison switcharoo. National Public Radio, 2003.

[8] Peter Winkler. *Mathematical Puzzles: A Connoisseur's Collection*. AK Peters, 2004.