Working Paper No. 1928

Clerks

Daniel Fershtman, Kfir Eliaz, and Alexander Frug

August 2025

Clerks[*]

Kfir Eliaz[†] Daniel Fershtman[‡] Alexander Frug[§]

August 10, 2025

Abstract

We study optimal dynamic scheduling of workers to tasks when task completion is privately observed —so that workers can delay the release of finished tasks — and idle time is the only available incentive instrument. We characterize a scheduling rule, and its induced equilibrium, that maximizes expected discounted output. Unless workers are inherently slow, production alternates between efficient phases and delays. Our analysis reveals a trade-off between the quality and the size of the workforce. We also present several extensions, illustrating the versatility of the framework.

Keywords: Strategic servers, non-monetary incentives, optimal scheduling, moral hazard, idle time, multi-server systems.

[*]Financial support from the Foerder Institute and the Sapir Center is gratefully acknowledged.

[†]Tel-Aviv University and NHH; email: kfire@tauex.tau.ac.il

[‡]Tel-Aviv University, Emory University, and CEPR; email: danfershtman@gmail.com

[§]Universitat Pompeu Fabra and Barcelona School of Economics; email: alexander.frug@upf.edu

1 Introduction

*"This job would be great if it wasn't for the ******* customers."* Randal in *Clerks*

In 2018, bed occupancy in the internal wards of most public hospitals in Israel surpassed 100%. This meant that either patients were left in the Emergency Room (ER), or they were put in temporary beds in the halls of the internal wards.[1] There was mounting pressure from the public and the media to resolve this crisis. While most hospitals argued that a resolution demanded higher budgets, one hospital ("Haemek") was able to cut down the occupancy in its internal wards to less than 75% within one year and without any additional expenses! What this hospital administration realized was that the root of the problem was the wards' response to the patient routing rule used by the ER. This involved assigning patients to whichever ward had a vacant bed. Administration suspected the wards were reacting to this by deliberately slowing down their release of patients to keep a high bed occupancy, many of which consisted of "cured" patients which required less treatment and care.

In light of this, the hospital administration decided to change the patient assignment rule to the following: each patient was assigned to a ward according to a fixed order (the first patient to ward 1, the second patient to ward 2, and so on until it was ward 1's turn again) *independently* of the bed occupancy in the wards. If a patient was assigned to ward i which did not have any vacant bed while ward j did, then the patient was put in ward j but was cared for by the staff of ward i. Within two months the average stay in the internal wards fell from 5 days to around 4 days (Linder, 2019).[2]

This anecdote highlights a broader phenomenon: service providers respond to the process of organizing work (e.g., the rules for allocating tasks), and their response may undermine the original objective of the designed work process (which was designed for a fixed behavior of workers). This is especially true in settings with the following characteristics: (i) workers face an inflow of tasks and are not compensated according to their performance, and (ii) workers are the experts on the scene in the sense that they are the ones who know whether a task was successfully completed or whether they put sufficient effort into it. One prominent setting with these features is the public sector where clerks handle public reception or cases/applications/appeals/complaints filed by the public.

[1]Internal wards are responsible for catering to a wide range of internal disorders, providing inpatient medical care to thousands of patients each year. In most hospitals in Israel, patients are routed from the emergency room to one of several internal wards that are similar in structure but have separate medical and administrative staff.

[2]See also Dagan, Lichtman-Sadot, Shurtz, and Zeltzer (2024) for an empirical analysis of this routing reform.

Examples include departments of motor vehicles, social services, unemployment insurance offices, postal offices, and public healthcare services. Even within the private sector, many customer service representatives are paid fixed wages, as is often the case in retail banking.

When wages are fixed, one of the main ways to incentivize workers is by designing the *work process*, which consists of several components. One component is a rule for determining when new tasks are assigned to a worker and what type of tasks. Some tasks are more work intensive or tedious than others (e.g., operating a cashier or a service window may be more laborious than ordering office supplies or filing paperwork). In addition, spacing new task assignments can also reward workers with some idle time. A second component of work-process design determines the relation between the number of servers (e.g., doctors, car mechanics, etc.) and the "capital" they use for work, such as beds in the case of hospitals, mechanic work stations in garages, or "windows" in the case of DMVs, social services, etc.

The workers, in turn, may respond strategically to the work-process design. For example, a worker who anticipates an unpleasant task to be allocated soon might be strategically slow on his current tasks - hoping to see the next task assigned to someone else; redundant, easy paperwork might keep service providers busy and reduce the number of daily appointments; and a physician's motivation to release a patient may depend on whether she expects to be responsible for fewer patients for some duration.

In addition, the designer of the work process often faces an "appearance constraint": the system should have the appearance of being constantly at work. For instance, if there are patients waiting in the ER, there should not be empty beds in any of the wards; if there are people waiting in line in the DMV/social security office/bank, there should not be any service window with a sign "next window please". Consequently, when tasks are waiting to be processed, all the capital that is used for processing them (beds, windows, work-stations) should be in use. This restricts the relation between workers and capital and imposes constraints on how workers can be incentivized with idle time or more rewarding tasks. This paper takes a first step at understanding this design problem and the economic implications of its solution.

We propose a simple stylized model that captures the main forces at work. There are n workers referred to as "servers". Servers work on tasks sequentially: only when they complete one task can they start working on a new one. To complete a task a server needs to work at a "work station", which can accommodate at most one server. A work station can represent a desk, a public reception window, a patient's bed, a phone in a call center or a machine in a factory. There are $m < n$ work stations, which need to be staffed at all times (this is the "appearance constraint" alluded to above). Time is continuous and the completion of a task is modeled as a Poisson process with arrival rate of λ, which represents the servers' productivity. There is an infinite supply of tasks, which are routed to the various servers according to a rule designed by a principal. Once a server completes a task, he decides at each subsequent time whether to release the task or to withhold it. Only the server working on a task observes when it is finished, but the release of a task is observed by all (the principal and all other servers).

The servers' incentives to release completed tasks stem from the following considerations. When a server is assigned a new task he incurs an immediate *set-up cost* of K and he incurs a flow *effort cost* of c at every period in which he works on the task. Idle time is costless (and hence valuable) while withholding a completed task bears a flow cost of $d < c$. The idea is that dwelling on a finished task does not require much effort, yet the server is not completely free to engage in other activities as during idle time (e.g., he cannot make personal calls or surf the web). The principal gets a normalized payoff of one from every released task.

The principal and the servers apply the same discount factor to future payoffs. Our goal is to highlight key features of optimal task allocation rules - ones that attain the highest discounted expected payoff for the principal in (perfect Bayesian) equilibrium - and to characterize equilibria that achieve these payoffs.

When servers are sufficiently slow - i.e., their productivity is below some threshold - the first-best output is achieved by a simple rotation scheme, where servers release tasks immediately upon completion and a server who releases a task is replaced by the non-working server who was idle for the longest time. This is because the relatively long period it takes a server to complete a task provides ample idle time to incentivize a server to release a finished task. However, when servers are fast enough so that their productivity rate exceeds the threshold, there is no equilibrium that attains the first-best: The idle time provided by working servers is just too short to incentivize servers to release a task and incur the future costs of working.

We characterize the second-best, i.e., derive the minimal amount of inevitable delay (inefficiency) in such systems, and find a task allocation scheme and an induced equilibrium that attains the minimal inefficiency. In this equilibrium servers voluntarily sit on finished tasks and delay their release. To prove that the above scheme and its induced equilibrium are optimal, we show that they attain a naive upper bound on the principal's discounted expected payoff. One important message of this result is that some level of inefficiency is necessary for the organization to operate (constrained) optimally. In our model this takes the form of shirking by sitting on completed task. The equilibrium we identify has the feature that the inherent delay is self-enforcing: it is a best response to the delay tactic of the servers.

Our analysis of the second-best also makes the following methodological contribution. We show how a multi-dimensional process involving multiple work stations and multiple servers who can be at multiple "phases" - working, releasing, delaying or being idle - can be reduced to a *unidimensional* representation with a single parameter. This representation considerably simplifies the analysis and proves useful in studying variants of our model.

The equilibrium described above highlights a trade-off that a principal faces with regards to the productivity of its servers. On the one hand, more productive workers can produce more output per unit of time; but on the other hand, faster production requires inefficient delay in order to incentivize workers to release completed tasks. This suggests a natural trade-off between quantity and quality of servers. We analyze this trade-off in our model. First, holding the number of servers and work stations fixed, we find that the total output indeed increases with the servers' productivity rate. However, when the servers' productivity becomes sufficiently high (so that delay becomes inevitable), the increase in output becomes rather slow. Building on this observation, we illustrate that a group of arbitrarily productive servers can be outperformed by a slightly larger group of mediocre servers (i.e., servers who are so slow that simple rotation without delay is incentive compatible). This hints at a tendency to prioritize quantity over quality in organizations with the features described earlier in the introduction.

We conclude our analysis by considering four extensions of our benchmark model. The first extension relaxes the appearance constraint by allowing a work station to be vacated at a flow cost per unit of time (which can capture the negative effect this has on the system's public image). For the clearest illustration of the implication of this relaxation, we focus on the case of a single work station and two servers. We consider "hybrid" mechanisms that allow for both delays and vacating the work station. Our main result here is that the optimal mechanism is *either* our original second-best solution in which the work station is always staffed but servers delay releases, *or* a mechanism with immediate releases but with temporary shut-downs of the work station. The former (latter) is optimal when the cost of vacating a station is above (below) some threshold. Thus, imposing the appearance constraint captures environments in which vacating a work station is quite costly.

The second extension enriches the discretion of each server by allowing him to shirk and sit on a task even before it is finished. Specifically, at any moment of time in which the task has yet to be completed, a server decides whether to exert effort and incur a flow cost of c or shirk and incur the lower cost of d by sitting on the task. The productivity rate is λ if and only if effort is exerted; otherwise, it is zero. Put differently, once assigned to a task, a server decides between two levels of productivity - zero or λ - such that positive productivity is more costly.

A consequence of this new moral hazard dimension of effort exertion is that the allocation scheme, which was optimal in the benchmark model, will now lead to shutdown. The reason is that if a server is willing to put in effort at the time of receiving a task, then he will strictly prefer to immediately release it upon completion. But this means that servers that are productive enough will not receive ample idle time, and hence, will not be motivated to put in effort in the first place. Thus, incentivizing effort is inconsistent with self-enforcing voluntary delays. In particular, an optimal task-assignment scheme cannot solely rely on the servers and requires some form of intervention by the principal. We illustrate one such scheme when there are two servers, a single work station and no set-up costs.

The third extension we consider is that of diverse productivity rates. The presence of this heterogeneity introduces a new complication: to maximize output it is now important to give more tasks to the more productive servers. We illustrate this in the simple case of two servers and a single work station. Here, the first-best is not incentive compatible since it requires the more productive server to constantly work. We characterize the second-best in the case when simple rotation (i.e., servers taking turns working at the work station without any delays) is incentive-compatible. We show that the optimum is attained by a stochastic mechanism in which there is a probability that the more productive server works again after completing a task.

Our final extension introduces the possibility of releasing an unfinished task, which gives zero payoff to the principal. This means that a server now decides at each point in time - both before and after finishing the task - whether to release it. We assume that when an unfinished task is released, there is a positive probability p that it returns (immediately) to be re-processed, and that it is publicly known who released it.

Focusing on the case with two servers and one work station we show that there is an interval of p values such that (i) for values above this interval, the principal can attain the first-best by assigning a returning task and all future tasks to the server who released the unfinished task; (ii) for values below this interval, the principal cannot prevent premature releases; and (iii) for values inside the interval, the principal can prevent premature releases but at a cost of delay.

Our paper makes a methodological contribution by proposing a framework for analyzing dynamic task allocation in organizations where wages are not responsive to output, and workers are privately informed about task completion. As our extensions illustrate, our framework can easily be adapted to capture a variety of frictions and can be applied to a broad range of settings. In addition, our analysis offers an explanation of perceived inefficiencies that are commonly observed in organizations with the above mentioned features, and also suggests possible interventions for improving work processes. For instance, in many cases what may seem as wasteful idleness of workers is actually a necessary feature for sustaining second-best production. Similarly, reducing the amount of capital available to workers can increase total output.

The remainder of the paper is organized as follows. Related literature is discussed below. Section 2 presents the benchmark model. The optimal allocation scheme is described and proven to be optimal in Section 3. Section 4 analyzes the trade-off between quantity and quality of servers. Section 5 considers the four extensions discussed above. Concluding remarks are given in Section 6.

Related literature

The paper is related to several strands of literature. First, it is related to a number of papers studying "strategic service provision." Gavazza and Lizzeri (2007) studies a problem involving servers who face a queue of customers. They consider a game between servers who choose their speed of service taking into account the cost of quicker service, the benefit of free time after all customers have been served, and the demand for service that increases with quicker service (such that more customers means less idle time). The main insight is that disclosing the service speeds of servers lowers the incentives to invest in quicker service and can lead to an equilibrium where customers are worse off relative to a regime without disclosure. In addition to having a very different framework than theirs (more below), slower service in our model is actually *necessary* to avoid shutdown (when servers are sufficiently productive).

Geng, Huh, and Nagarajan (2015) considers a game between two servers who simultaneously choose their production rates in response to a routing policy. The servers face a Poisson arrival of customers, and each server's payoff is a sum of two functions: one that is *inversely* related to the amount of its steady-state idle time, and another that decreases with the difference between the servers' workload (measured as the ratio of idle time to production rate). The authors explore a given set of routing policies, and for each policy in this set, they characterize properties of the Nash equilibrium in the game between the servers. Gopalakrishnan, Doroudi, Ward, and Wierman (2016) embeds into a canonical $M/M/N$ queuing framework servers who choose their productivity rate in order to maximize the difference between idle time and the cost associated with their productivity. Focusing on a random task allocation rule, the authors give necessary and sufficient conditions for the existence of a solution to the first-order conditions of a symmetric Nash equilibrium.

Armony, Roels, and Song (2021) also studies a simultaneous game between two servers who choose their production rate. The innovation in that paper is that the cost of each server is the sum of three components: the cost of its production rate, the expected time it takes to service a customer and the expected time that each of its customers needs to wait until they are serviced. The main result of the paper characterizes properties of the Nash equilibrium for two configurations of queues: (i) two independent $M/M/1$ queues, where each server has her own infinite-buffer first-come-first-served (FCFS) queue and customers are routed randomly and uniformly between the two queues, and (ii) a single $M/M/2$ with a single infinite-buffer FCFS queue.

There are two key differences between all of these papers and ours. First, all these papers consider static complete-information games where workers simultaneously choose their production rate. In contrast, we analyze a dynamic model where the production rates are fixed (though we relax this in one of our extensions) but completion times are private information and workers decide when to release tasks. Second, all of the above papers compare the output of *exogenously* given allocation rules. We, however, characterize the maximal expected discounted output that can be attained.

A second related strand of literature considers the problem of scheduling tasks or "work-design." Coviello, Ichino, and Persico (2014, 2015) study a decision-maker facing a growing queue of tasks that arrive at an exogenous rate. Their 2014 paper characterizes the production function, which relates the output rate to the effort rate (that governs completion time) and the activation rate (at which tasks are started). Their 2015 paper applies this production function to a dataset of judges' handling of court cases to estimate the effect of increased case load. Bray, Coviello, Ichino, and Persico (2016) models case-scheduling by judges as a classic multi-armed bandit problem, and argues that prioritizing the oldest hearing (case) is optimal when the case completion hazard rate function is decreasing (increasing). In contrast to our work, these papers do not consider the moral hazard problem inherent in a worker's discretion of when to release a completed task. In Eliaz, Fershtman, and Frug (2022), we study a scheduling problem in which a decision-maker sequentially chooses among alternatives – which can be interpreted as tasks being assigned to workers – when the resulting periodic payoffs depend on both chosen and unchosen alternatives in that period. Besides considering a different type of scheduling problem, agents are not strategic in that paper.

Mylovanov and Schmitz (2008) studies a model of task scheduling in the presence of moral hazard. They study a principal who needs three tasks to be completed within two periods. The principal can assign at most two tasks to one of many identical agents, who each chooses whether to privately exert costly effort. High effort contributes to a higher probability of completing a task, which is publicly observed. Unlike in our framework, the principal has a rich set of instruments that include payments, which can be made contingent on task completion, and a task assignment scheme that depends on whether agents completed the tasks assigned to them.

Finally, the paper is also part of a growing literature studying dynamic delegation without monetary transfers; See, e.g., Frankel (2016), Guo (2016), Li, Matouschek, and Powell (2017), Bird and Frug (2019), Forand and Zápal (2020), Guo and Hörner (2020), and Lipnowski and Ramos (2020). All these papers consider a principal and a single agent, whereas we study a dynamic scheduling problems where multiple agents are assigned to multiple tasks over time.

A recent paper that does consider a dynamic assignment problem among multiple agents is De Clippel, Eliaz, Fershtman, and Rozen (2021). However, the incentive problem in that paper stems from *adverse selection* rather than moral hazard. More specifically, each period a principal wishes to assign a task to one of two agents, depending on who is most qualified to complete the task. Since agents privately learn whether they are qualified for the current period's task, and since they both want to be selected regardless of their qualification, the principal's problem is to design an optimal assignment rule that depends on the public history of task completion.

2 Model

We consider a continuous-time model with one principal, n "servers" and $m < n$ "work-stations". The servers face an infinite inflow of tasks. The principal assigns these tasks to the servers. Completing a task requires one server to work at one of the work-stations for a stochastic amount of time. Specifically, the completion rate of server $i \in \{1, 2, ..., n\}$ is $\lambda > 0$. At any moment of time, one and only one server occupies each work-station.

Only the server working on a task observes when it is completed, and the only decision he faces is when to release a completed task. That is, a server cannot reject a task that is allocated to him and cannot abort an incomplete task. However, once the task is completed, he decides at any moment of time whether to release it or to "sit on it," delaying its release. When a task is released, its release is publicly observed.

When a server receives a task, he incurs an immediate set-up cost of $K > 0$. While working on an (unfinished) task, he incurs a flow effort cost of $c > 0$. While sitting on a completed task, the server incurs a flow delay cost of $d \in (0, c)$. The principal receives a payoff of 1 for each completed task when the task is released. There is a common discount rate of $r > 0$.

An allocation rule selects m servers at time zero and specifies a history-dependent probability distribution over available servers (i.e., servers that are not busy working on a task) that determines which server will work on the next task, whenever some work-station becomes vacant. An allocation rule induces an extensive game of imperfect information among the servers. The principal's payoff from an allocation rule is defined as the principal's highest expected discounted payoff over all perfect Bayesian equilibria (PBEs) of the induced game. We say that an allocation rule is optimal if there is no other allocation rule that attains a higher payoff for the principal.

3 Optimal allocation and delay

The first-best expected discounted payoff for the principal is attained when all stations are occupied at all times and tasks are released immediately upon their completion. Hence, the principal's expected payoff under the non-strategic, first-best benchmark is given by $m\lambda/r$.

When servers are sufficiently slow (low enough λ), the first-best can be attained by *simple rotation without delay*: servers take turns occupying stations, each server immediately releases a task once it is finished and joins the end of the line of idle servers. The reason is that the servers' slow production rate gives ample idle time to compensate for the future cost of working. However, when the servers are sufficiently productive (high enough λ), the idle time that each server would get may not be enough. Consequently, some delay must be injected into the system.

In the baseline case with $n = 2$ and $m = 1$, a simple allocation and delay scheme can be employed: Servers take turns working on tasks, and from the second task onward, they withhold the release of each task for a fixed amount of time, say τ (the release of the first task is not delayed). When τ is set such that upon completing a task, a server is indifferent between withholding the task and releasing it, this scheme attains the second-best output.[3]

This result, which will follow as a special case of Proposition 1 below, can be understood intuitively. Server i is willing to release a completed task because the expected value of idle time he will enjoy – while server j is working and delaying a release of a task – is just long enough to compensate for the cost of working on the next task assignment. Delaying the very first task is not needed since the role of such (costly) delays is only to incentivize previous task releases.

To generalize the above scheme to any $n \geq 2$ and $1 \leq m < n$, we begin by making several preliminary observations. First, as in the case of $n = 2$ and $m = 1$, it is suboptimal

[3]We emphasize that this scheme is not *uniquely* optimal. For example, instead of fixed-time delays, delay times may be random. Also, for some parameters, delays may not be uniform over time and begin at a later stage.

to delay the release of any task after which a "new" server is called to work on his very first task. Second, with many servers and work stations, the order in which tasks are allocated and completed need not coincide. Hence, whenever a work station is vacated, a server with the longest current break is assigned a task. Third, when delays begin to take place, it is suboptimal to have each server delay the release of a task by the same amount of time. To see this, let i be the first server to be assigned a task for the second time. Any delay that is added to incentivize server i before he is called to work again also provides *the same amount* of idle time for *all* the servers who are idle at that time. Thus, the next server to be called to work has already received enough idle time so any additional delay will over-compensate some servers.

To build some intuition for the general delay strategy our scheme will use, note that in the baseline case of $n = 2$ and $m = 1$, delaying a release of a completed task is similar to temporarily freezing the production process of the system. Since the production process is fully captured by the type of activity (production/delay) at the single workstation, the analysis becomes essentially "unidimentional" and therefore, tractable. When $m > 1$ and task completion is privately observed, delaying a release of a specific task does not stop work on other work stations. Since the activities at different work stations are not synchronized, the natural description of the production process becomes high-dimensional, which complicates the analysis.

The scheme that we propose below restores the unidimensional (and hence, tractable) structure of the production process. Although our scheme generates rich dynamic patterns with spillovers over time within and across work stations, these patterns can nevertheless be described as if there is *coordinated* temporal freezing of the entire production process of the system (even though task completion is privately observed).

We now turn to present a general scheme of allocation and delay. We then illustrate the dynamics it generates and discuss two related properties of the scheme that simplify its analysis. Finally, we prove that this scheme is optimal for any number of servers and work stations.

3.1 Rotation with Delay Scheme

The *Rotation with Delay Scheme* (RDS) is described by several steps, where each release of a completed task brings the allocation scheme to the next step. Each step can be characterized by the *state* of the system, where a state is a pair: (i) a set I of servers that are *in* work stations, and (ii) a vector O that specifies the queue of servers that are *out* of work stations, where earlier vector entries correspond to servers at earlier positions in the queue.

Initially, the first m servers are "in", and whenever a working server releases a task, he exits and is replaced by the earliest server to have entered the "out"group.

Formally, these steps can be described as follows.

- *Step 1*: Let the initial state be $(I_1, O_1) = (\{1, ..., m\}, (m+1, ..., n))$, and let $k \in I_1$ be the first server that releases a task in Step 1. The allocation rule then transitions to Step 2, where the state is

$$(I_2, O_2) = (\{1, ..., m, m+1\} - \{k\}, (m+2, ..., n, k)) .$$

- *Step ℓ*: Given the state $(I_\ell, O_\ell) = (\{\ell_1, ..., \ell_m\}, (\ell_{m+1}, ..., \ell_n))$, let $\ell_k \in I_\ell$ be the first server that releases a task in this step. The allocation rule then transitions to Step $(\ell + 1)$, in which the state is

$$(I_{\ell+1}, O_{\ell+1}) = (\{\ell_1, ..., \ell_m, \ell_{m+1}\} - \{\ell_k\}, (\ell_{m+2}, ..., \ell_n, \ell_k)) .$$

The above dynamics create "natural" breaks for the servers given by the idle time between releasing a task upon completion and being assigned the next one. If these breaks provide sufficient incentives to release, the above dynamics collapse to simple-rotation without delay. Otherwise, these dynamics prescribe delays according to a structure that is fully characterized by a single parameter τ.

We denote rotation with delay of τ by $RDS(\tau)$, where delays are arranged as described below.

First, delays *initiate* in cycles: They start in some step σ, and new delays can begin only every $n - m$ steps. Furthermore, the delay strategy of each server is composed of two forms of delay: First, delay begins by withholding the release of a completed task for τ units of time after which the task is released, *except* if before reaching the τ-th unit of delay time, some other working server happens to release a completed task. In the latter case, upon an observed release by another server, the delay switches to being a Poisson process with rate λ – i.e., the server delays the release of a task for a random duration that terminates at the same rate as the servers' production. If the "$\lambda-$delay" ends, the server releases the task unless the system is again in one of the steps that initiate delay, in which case, the server moves to "$\tau-$delay" again, and so forth.[4]

Formally, let $\Sigma = \{\sigma_j\}_{j \in \mathbb{N}}$ denote the subset of steps that initiate delay, where $\sigma_j = 1 + j(n - m)$. At each moment, each server is in one of four possible phases:

[4]Recall that whenever a task is released, the system transitions to the next step. Hence, while a server is delaying a task, the system may proceed to next steps.

- *Idle phase*: The server does not work, pays no cost, and enters the productive phase (described below) according to the transition rule specified above.
- *Productive phase*: The server works on a task, pays a flow cost of c, and completes the task according to a Poisson process with rate λ. Upon completing a task, in every step $\sigma \notin \Sigma$, the server then releases the task and enters his idle phase, whereas in every step $\sigma \in \Sigma$, he enters a τ-delay phase (see below).
- *τ-delay phase*: The server sits on a finished task, pays a flow cost of $d < c$, and exits this phase once the earliest of the following events occur: (i) τ units of time have elapsed, in which case the server releases the task and enters his idle phase, or (ii) some other working server releases a task, in which case the server enters the λ-delay phase (see below).
- *λ-delay phase*: The server sits on a finished task, pays a flow cost of $d < c$, and exits this phase according to a Poisson process with rate λ. In every step $\sigma \notin \Sigma$, the server then releases the task and enters the idle phase, whereas in every step $\sigma \in \Sigma$, he enters the τ-delay phase.

3.2 Illustration of the RDS

Figure 1 below illustrates possible dynamics under *RDS* for the case of four servers and two work stations.

The bold horizontal time-axis in the middle of Figure 1 divides it into two interrelated panels; the lower panel indicates for each server ($S1$, $S2$, $S3$, and $S4$) the times at which he is idle, and the upper panel specifies, for each point in time, whether the server currently assigned to each of the work stations ($M1$ and $M2$) works on a task or delays the release of a completed task according to either $\tau-$delay or $\lambda-$delay.

At the start, $S1$ and $S2$ are assigned to $M1$ and $M2$, respectively, while $S3$ and $S4$ are idle. Until the first release of a task, the system is at state $1 \notin \Sigma$ as denoted below the time-axis. The system moves to state 2 when the first task is released. Under the specified realization, $S2$ completes a task first. This is indicated by the left-most $\star$ symbol at the work-line of $M2$. The arrow from that $\star$ down to the time-axis indicates the immediate release of the completed task. At that point, $S2$ and the currently idle $S3$ switch positions. State 2 of the system is similar to state 1. The next server to complete a task (in this case, $S1$) releases it without delay since $2 \notin \Sigma$. $S1$ is then replaced by $S4$ – the idle server with the longest current break, and the system moves to state 3.

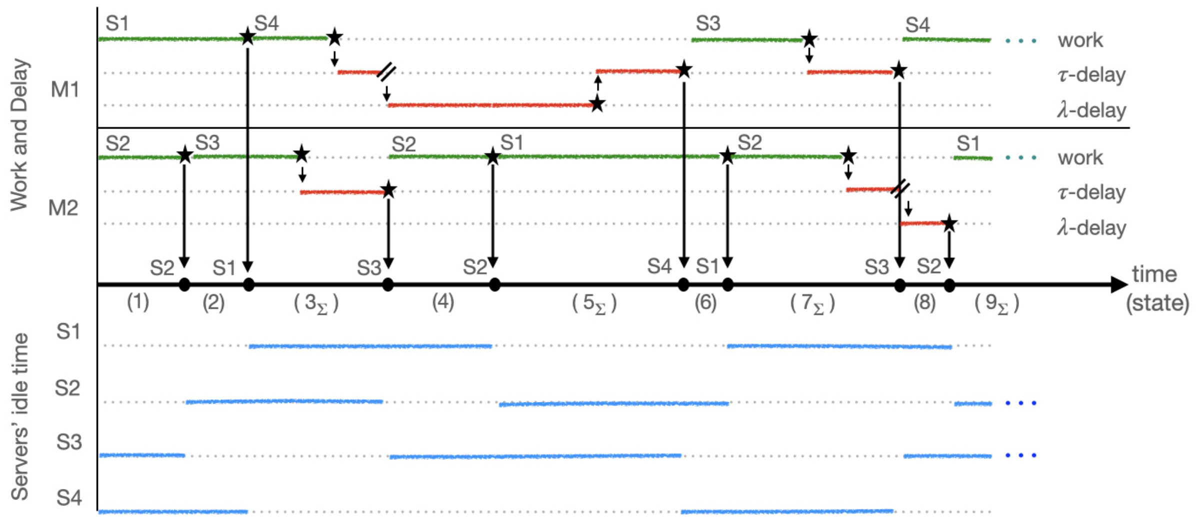


Figure 1: Illustration of dynamics under the rotation-with-delay scheme.

Since $3 \in \Sigma$, the first working server (among $S3$ and $S4$) to complete a task will not release it immediately but only after τ units of time. In the figure, this is reflected in the short down-pointing arrow from the left-most $\star$ symbol at stage 3 (task completion by $S3$ on $M2$) down to the $\tau-$delay line of $M2$. In the depicted realization, before the τ units of time elapsed, server $S4$ on the other work station also completed a task, which puts both servers in the τ-delay phase simultaneously. When a task is finally released by $S3$ (who completed the task first within the current state of the system), $S3$ will be replaced by $S2$ (the idle server with the longest current break) and $S4$ will move from the incomplete $\tau-$delay phase into the $\lambda-$delay phase (in the figure, the unsuccessful completion of the $\tau-$delay phase is marked by the rotated double-slash symbol).

This marks the beginning of state 4 of the system and since $4 \notin \Sigma$, there will be no $\tau-$delays at that state. Note that both servers occupying the work stations are waiting for an event that arrives with Poisson rate λ to release a task (whether it is exiting the $\lambda-$delay phase for $S4$ or completing a task for $S2$.) A released task will move the system to state $5 \in \Sigma$. As can be seen in the figure, the current realization induces a transition of $S4$ from $\lambda-$delay back to a new $\tau-$delay phase after which $S4$ finally releases a task, advancing the system to state 6, and so on.

3.3 The role of two delay phases

Introducing two separate delay components to the server's strategy makes the analysis tractable for the following (related but distinct) reasons. First, it simplifies incentive constraints. Consider a server who has just completed a task. In general, to decide whether to release the task or not, that server needs to estimate the time until he will be reassigned to a new task. This will depend on the history if the expected release time of other servers depends on whether they are still working or delaying the release of a completed task. Introducing the λ-delay that begins with the (publicly observed) release of a task by another server, the expected time between a release of a completed task and the next assignment to that same server is the same *regardless* of the history.

Second, the two delay components in the server's strategy generate dynamics that are payoff-equivalent to a simple "uni-dimensional" cyclical productivity where the system is analyzed as if production on different work stations were perfectly coordinated. This, in turn, implies that despite the richness of the possible dynamic patterns that may arise in $RDS(\tau)$ (as illustrated in the previous section), calculating expected productivity becomes straightforward.

To see this, consider an auxiliary production system that can be in one of two states: *active* or *inactive*. When the system is active, its production rate is $m\lambda$. After completing task number $\sigma \in \Sigma$, but before releasing it, the system freezes for τ units of time.

This auxiliary system is payoff-equivalent to $RDS(\tau)$. At the start, the system's productivity rate under $RDS(\tau)$ is exactly $m\lambda$ (corresponding to the auxiliary system being active). Let server i be the first to initiate the τ-delay upon completing a task at stage σ. He will release the task after exactly τ units of time. This is because all other servers are guaranteed to not release any tasks beforehand since i completed his task before them. It follows that during this period of length τ, the release rate is zero (corresponding to the auxiliary system being inactive).

Now consider the total release rate after i finally releases his task. A server who replaces i, as well as any other working server who did not complete a task during i's delay, will continue to work - each contributing one Poisson process with a release rate of λ. At the same time, any server who did finish a task while i delayed, failed to reach a count of full τ units of time before releasing (since i started this count first). Hence, upon the task release by i, any such server switches to $\lambda-$delay, thereby, contributing again one Poisson process with a release rate of λ. Aggregating these m Poisson processes corresponds to the active state of the auxiliary system, until we reach a step where a new delay is initiated.

Having described the activity dynamics at the level of the whole system, rather than at the level of the work stations, we can represent them in a simple uni-dimensional diagram. Figure 2 illustrates this for $n = 4$ and $m = 2$, which we considered in the previous section. In the diagram, a segment describes the (expected) duration of each state of the system: active (a) or inactive (i) between completions and releases. The release of a completed task is indicated by a full circle and an empty circle indicates finishing working on a task but not releasing it and entering the temporal inactive phase.

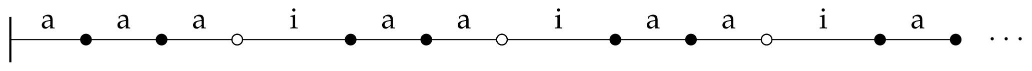


Figure 2: Active and inactive phases.

To further simplify the calculation of the principal's expected payoff, note that the above dynamics can be decomposed into the first two tasks that are not delayed and the indefinitely repeated cycle of two tasks where only the release of the first is delayed (see Figure 3).

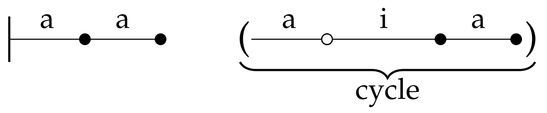


Figure 3: First tasks and cycle structure.

This allows us to easily calculate the expected total productivity (i.e., the principal's payoff). Denote the discounted payoff from the first completion by $x := \frac{2\lambda}{2\lambda+r}$, which also represents the discount until a single task is completed at two work stations. Therefore, the payoff, discounted to time 0, from the first two tasks is $x + x^2$.

If the the duration of the inactive phase is τ, the payoff from a single cycle, discounted to the beginning of the cycle, is $e^{-r\tau}(x + x^2)$. The discounted payoff from each additional cycle $i > 1$ needs to be further discounted by a factor of $(e^{-r\tau}x^2)^i$ to be expressed in the same units as the payoff from the initial cycle. It follows that the expected payoff from the moment the third task is assigned (and we enter the infinitely repeated cycle) is given by:

$$\sum_{j=0}^{\infty} \left(e^{-r\tau}x^2\right)^j e^{-r\tau}(x + x^2) = \frac{e^{-r\tau}(x + x^2)}{1 - e^{-r\tau}x^2},$$

so the total expected payoff discounted to time 0 is $x + x^2 + x^2 \left(\frac{e^{-r\tau}(x+x^2)}{1-x^2e^{-r\tau}}\right)$, or simply

$$\frac{x + x^2}{1 - e^{-r\tau}x^2}.$$

3.4 Optimal delay

Consider a server who completed a task. Set τ (corresponding to the $\tau-$delay phase described above) such that from the perspective of the server, the expected value of idle time until his next task exactly compensates the additional cost of working on that future task, relative to the flow cost of sitting on it until it is completed:

$$\frac{d}{r}\left(1-e^{-r\tau}\left(\frac{m\lambda}{m\lambda+r}\right)^{n-m}\frac{\lambda}{\lambda+r}\right)=e^{-r\tau}\left(\frac{m\lambda}{m\lambda+r}\right)^{n-m}\left(K+\frac{c}{\lambda+r}\right). \tag{1}$$

Note that both expressions on the right and left hand sides of the equality consider periods of length ($\tau +$ *one cycle of others' work without delay* $+$ *own work on a single task*). Specifically, on the right hand side, the future costs of working on a single task, $K+\frac{c}{\lambda+r}$, are discounted based on two variables: the expected discount factor between completing and receiving the next task,

$$\Delta(\lambda)\equiv\left(\frac{m\lambda}{m\lambda+r}\right)^{n-m},$$

and the delay of τ units of time. When a server considers releasing a task, the RHS of (1) therefore represents the expected costs until the server will finish working on the *next* task assigned under $RDS(\tau)$. Alternatively, the server can sit on the completed task and incur a flow cost of d for the same duration. Both courses of behavior will put the server at the end of that time frame at a work station with a completed task at hand which the server can then again decide whether to release or withhold.

The time τ^* that solves (1) guarantees that a server who follows the allocation scheme described above expects to receive a continuation value of $\frac{d}{r}$ at the time of releasing a completed task. That is,

$$\frac{d}{r}=e^{-r\tau^*}\Delta(\lambda)\left(K+\frac{c}{\lambda+r}+\frac{\lambda}{\lambda+r}\frac{d}{r}\right). \tag{2}$$

The strategies defined by $RDS(\tau^*)$ with τ^* defined by (2), therefore constitute an equilibrium. We denote

$$D(\lambda)\equiv e^{-r\tau^*}\Delta(\lambda)=\frac{\frac{d}{r}}{K+\frac{c}{\lambda+r}+\frac{\lambda}{\lambda+r}\frac{d}{r}}.$$

When $\tau^*=0$, $RDS(\tau^*)$ collapses to simple rotation without delay. Note that simple rotation without delay is an equilibrium if the RHS of (2) at $\tau^*=0$ is no greater than the LHS, i.e., $D(\lambda)\leq\Delta(\lambda)$.

Lemma 1. *The equation $\Delta(\lambda) = D(\lambda)$ has a unique solution, $\lambda^* > 0$, and for all $\lambda < \lambda^*$, simple rotation without delay constitutes an equilibrium.*

We refer to servers with productivity rate $\lambda \leq \lambda^*$ as *mediocre servers*.[5] While for mediocre servers simple rotation without delay is incentive compatible, it is no longer the case for teams of servers with $\lambda > \lambda^*$.

Our next result establishes that when simple rotation without delay is not incentive compatible (and thus, the first best cannot be attained), the principal's discounted expected payoff under $RDS(\tau^*)$ is equal to his first-best payoff scaled down by the following ratio: One minus the expected discounted time between tasks under simple rotation, to one minus the expected discounted time between tasks under incentive-compatible delay.

Proposition 1. *For any $\lambda > \lambda^*$, the principal's expected discounted payoff from the $RDS(\tau^*)$ is given by*

$$\frac{m\lambda}{r}\left(\frac{1-\Delta(\lambda)}{1-D(\lambda)}\right). \tag{3}$$

We now show that RDS maximizes the principal's expected discounted payoff.

Proposition 2. *$RDS(\tau^*)$ maximizes the principal's expected discounted payoff.*

We include the proof of this proposition in the main text because it uses an approach that we apply in subsequent results. We first give an overview of this approach before presenting the formal proof.

Since $\min \Sigma = 1 + n - m$, under $RDS(\tau^*)$ the first $n - m$ tasks are released immediately upon completion. Hence, under $RDS(\tau^*)$ the principal's total expected discounted payoff from the first $n - m$ completed tasks is the same as in the first-best output.

Let t_{n-m} denote the (stochastic) point in time at which the $(n-m)$-th task is released. At this time, $RDS(\tau^*)$ enters a phase where tasks will be delayed. The proof derives a "naive" upper bound on the discounted expected output that is attainable from t_{n-m} onward. We then show that this bound is achieved by $RDS(\tau^*)$.

We derive this upper bound in two steps. First, we compute the total available "budget" $\mathcal{B}$ of idle time that can be used to incentivize servers to release all the future completed tasks starting from the $(n-m+1)$-th task. Second, we calculate the net costs $\mathcal{C}$ that need to be compensated to incentivize servers to release tasks. The "naive" upper bound is then the fraction $\frac{\mathcal{B}}{\mathcal{C}}$ of the first-best output. This upper bound is "naive" in two respects: (i) it ignores the "physics" of the environment (i.e., the sequential nature in which servers need

[5]Recall that the value of λ^* depends on other parameters.

to staff stations once tasks are released), and (ii) in computing $\mathcal{B}$ it ignores the idle time that had to be promised to the servers who released the first $n-m$ tasks to incentivize them to release those tasks once they were finished.

Proof of Proposition 2. The proof proceeds in four steps.

Step 1: Deriving $\mathcal{B}$. Consider time t_{n-m}. As there are m work stations and n servers, $n-m$ servers will be idle at each instant. Since the alternative to releasing a task is to sit on it and pay a flow cost of d, it follows that the total value of idle time that can be allocated to servers *from* t_{n-m} *onwards* is equal to $(n-m)(d/r)$.

However, not all of this budget can be used to provide incentives because some servers will necessarily enjoy idle time *before* being assigned to their first task after time t_{n-m}. As mentioned above, our derivation of $\mathcal{B}$ is naive or permissive in that we treat the tasks that servers worked on before t_{n-m} as sunk, and hence, do not need to be compensated (i.e., we ignore any "promises" that were made to these servers prior to t_{n-m} to incentivize them to release tasks).

Note that up until the release of the $(n-m+1)$-th task, $n-m$ servers consume idle time of $(n-m)\frac{d}{m\lambda+r}$, which cannot incentivize them to release their future assigned tasks. Similarly, between the release of the $(n-m+1)$-th and $(n-m+2)$-th tasks, there will be $n-m-1$ idle servers that have not worked since t_{n-m} and hence consume idle time that cannot be used to incentivise them (for future timely releases) . This "wasted" idle time, in units of time t_{n-m}, is equal to $(n-m-1)\left(\frac{m\lambda}{m\lambda+r}\right)\frac{d}{m\lambda+r}$, where the new component $\frac{m\lambda}{m\lambda+r}$ arises due to discounting to time t_{n-m}. Continuing in this manner, the idle time wasted between the $(n-m-(k-1))$-th release and the $(n-m-k)$-th release, is equal to $(n-m-k)\left(\frac{m\lambda}{m\lambda+r}\right)^{(k-1)}\frac{d}{m\lambda+r}$. It follows that the maximal expected *usable* budget of idle time, in terms of time t_{n-m}, is

$$\mathcal{B}=(n-m)\frac{d}{r}-\frac{d}{m\lambda+r}\sum_{j=1}^{n-m}j\left(\frac{m\lambda}{m\lambda+r}\right)^{n-m-j}=\frac{dm\lambda}{r^2}\left(1-\left(\frac{m\lambda}{m\lambda+r}\right)^{n-m}\right),\quad (4)$$

where the last equality follows from simple but tedious algebra.

Step 2: Deriving $\mathcal{C}$. We begin by computing the difference between the total discounted expected costs that are accrued when m work stations are constantly staffed by servers who work (first-best production), and the total costs that are accrued when the servers at the stations just sit on completed tasks (indefinite delay). In order for the servers to agree to release completed tasks from the $(n-m+1)$-th task onward, this difference in costs must be covered by the total budget of idle time that is available when the $(n-m)$-th task is released. This difference in costs is given by the expression

$$\underbrace{K+m\left(\frac{c+\lambda K}{r}\right)}_{\text{costs of first-best production}} - \underbrace{m\frac{d}{r}}_{\text{cost of indefinite delay}} = K+m\left(\frac{c-d+\lambda k}{r}\right). \tag{5}$$

The expression $(c+\lambda K)/r$ reflects the average production cost for each work station. Note that the station where the $(n-m)$-th task is released requires an immediate (and hence discrete) cost of K (while the servers on the other $m-1$ work stations are still working on their tasks).

The amount in (5) represents the entire difference in the costs incurred between first-best production and indefinite delay of completed tasks. However, since servers cannot reject task assignments, they do not need to be compensated for the costs associated with the production of their first assignment, starting from time t_{n-m} (for each server, the first decision after time t_{n-m} arrives only upon completion of his first assignment).

To calculate the total costs that require compensation to incentivize first-best production, we subtract from (5) the costs associated with the servers' first assigned tasks, *starting from* t_{n-m}. These costs have two components: (i) a cost of $(m-1)\frac{c-d}{\lambda+r}$, reflecting the total expected cost of completing a task for the $m-1$ servers who are already working on a task at time t_{n-m}, and (ii) the expected cost of producing a task for the server who receives his first task at time t_{n-m}, as well as the future expected costs of those $n-m$ servers who are idle after the allocation of the task at time t_{n-m},

$$\left(K+\frac{c-d}{\lambda+r}\right)\left(1+\frac{m\lambda}{m\lambda+r}+...+\left(\frac{m\lambda}{m\lambda+r}\right)^{n-m}\right).$$

Hence, the total net expected discounted cost $\mathcal{C}$ of first-best production requiring compensation, starting from t_{n-m}, is given by:

$$\begin{aligned}\mathcal{C} &= K+m\left(\frac{c-d+\lambda k}{r}\right)-\left((m-1)\frac{c-d}{\lambda+r}+\left(K+\frac{c-d}{\lambda+r}\right)\sum_{j=0}^{n-m}\left(\frac{m\lambda}{m\lambda+r}\right)^{j}\right)\\ &= \frac{m\lambda}{r}\left(\frac{c-d}{r+\lambda}+K\right)\left(\frac{m\lambda}{r+m\lambda}\right)^{n-m}.\end{aligned}$$

Step 3: Deriving the upper bound. Recall that, starting from any point in time, the first-best expected discounted continuation output is given by $m\lambda/r$. As a result, we obtain a naive upper bound of

$$\frac{\mathcal{B}}{\mathcal{C}} \cdot \frac{m\lambda}{r} \tag{6}$$

on the attainable (i.e., incentive compatible) expected discounted output starting from t_{n-m}. That is, the principal cannot hope to obtain a fraction of the first-best payoff greater than $\mathcal{B}/\mathcal{C}$. Substituting into (6) the expression for $\mathcal{B}$ and $\mathcal{C}$, we have the following upper bound on future discounted production starting from t_{n-m} relative to time t_{n-m}:

$$\frac{\mathcal{B}}{\mathcal{C}} \cdot \frac{m\lambda}{r} = \frac{d}{r}\left(\frac{1}{\frac{c-d}{r+\lambda}+K}\right)\left(\frac{1-\left(\frac{m\lambda}{m\lambda+r}\right)^{n-m}}{\left(\frac{m\lambda}{r+m\lambda}\right)^{n-m}}\right)\frac{m\lambda}{r}. \tag{7}$$

Step 4: Comparing the upper bound to $RDS(\tau^*)$ output. We now show that $RDS(\tau^*)$ attains this upper bound. That is, we show that the payoff under this scheme, starting from time t_{n-m}, is equal to (7). Recall that the expected discounted output under $RDS(\tau^*)$, as derived in the proof of Proposition 1, consists of two components. The first corresponds to the output from the first $n-m$ released task, which are released without delay, whereas the second component, given by (21) corresponds to the remaining continuation output. To complete the proof, it is straightforward to verify that the latter continuation output, adjusted to time t_{n-m}, is equal to (7). ■

3.5 Can fewer work stations increase output?

Productivity in our setting is constrained by two upper bounds: (i) a non-strategic "technological" productivity bound, and (ii) a bound reflecting the "incentive budget" that arises due to moral hazard. The former is *increasing* in the number of work stations since in the absence of strategic considerations, additional work stations enable more servers to work in parallel. This is the effective bound when $\lambda < \lambda^*$.

When $\lambda > \lambda^*$, as Propositions 1 and 2 show, the technological upper bound cannot be achieved. However, in the proof of Proposition 2 we show that RDS in fact utilizes *all* of the usable incentive budget $\mathcal{B}$ (see equation (4)). This means that when $\lambda > \lambda^*$, the effective upper bound on productivity (i.e., the above mentioned incentive-budget bound) is attainable.

At the beginning of the interaction, some idle time must be allocated to the servers before they work. With a higher number of work stations, the amount of unusable incentives at the beginning of the interaction decreases since the first completions arrive faster.[6] If the servers are very impatient, this more than compensates the additional long-term incentive budget that arises when there are fewer work stations. The next result shows that when the players are patient enough, reducing the amount of work stations increases the available incentive budget. Thus, as long as there is delay in the optimal allocation rule, adding work stations is strictly worse for the principal.

Proposition 3. *Fix the model parameters $(c, d, K, \lambda, r, n, m)$ such that there is positive delay in RDS. Then, provided that the total expected output is positive, increasing the number of work stations reduces the principal's expected payoff, for sufficiently small r.*

3.6 Discussion and interpretation

In our stylized model, a server is in one of only three states: *work* (efficiently occupying a work-station), *delay* (inefficiently occupying a work-station), or *idle*. More generally, work and inefficiencies in production processes can come in varying intensities and forms.

The idle state in our model represents various alternatives to reward the worker through lowering his work intensity or assigning him more pleasant tasks. Our assumption that only a given number of servers can be idle at each point in time captures the idea that altering the workload of a team of servers is subject to organizational constraints (e.g., the relation between the amount of servers and the capital available to them, and possibly the need to appear constantly at work).

Delays in our model represent various inefficiencies that organizations may inject into work processes. Thus, our model proposes a new perspective on seemingly inefficient project choices or inefficient work processes by interpreting them as possible means to provide incentives. We now illustrate this point in a simple example where instead of delaying task releases, an inefficient production protocol is selected in optimum.

Suppose that there are two servers and one work station, and assume that when a server is called to work on a task he can follow the "**efficient protocol**" – do only the efficient component of the job, or to follow an "**extended protocol**" where he starts with an inefficient component, and upon completion, begins working on the efficient one. The efficient component is completed, as before, according to a Poisson process with arrival rate λ (which does not depend on whether the server worked on the inefficient component first). The inefficient component is completed according to a Poisson process with parameter μ.

When a server is allocated a task, he pays a set-up cost of K and a flow cost of c until

[6] For instance, in the above example, when $m = 2$ the expected time of the first completion is $1/2$, while with $m = 1$ the expected time of the first completion is 1 and that of the second completion is 2.

the efficient component is completed - regardless of the protocol. When the server releases a completed task after the efficient protocol, the principal gets 1. When a completed task is released after the extended protocol, the principal gets $1 + \beta$.

Consider $\lambda > \lambda^*$. That is, the first-best outcome (the non-strategic benchmark) - where the servers only work on the efficient component and release completed tasks without delays - is not incentive compatible. For computational simplicity, suppose that μ solves:

$$\frac{\lambda^*}{\lambda^* + r} = \frac{\mu}{\mu + r}\frac{\lambda}{\lambda + r} \Rightarrow \mu = \frac{(\lambda + r)\lambda^*}{\lambda - \lambda^*}.$$

This implies that μ is set such that the server's considerations under the extended protocol are identical to those in our original model (where only the efficient protocol exists) when the productivity rate of all servers is λ^*. Therefore, simple rotation without delay – provided that the servers follow the extended protocol, is now incentive compatible. Thus, while the principal's payoff from the non-strategic benchmark is $\frac{\lambda}{r}$, her payoff from the extended protocol is $(1 + \beta)\frac{\lambda^*}{r}$. Hence, under the assumption that $(1 + \beta)\frac{\lambda^*}{r} < \frac{\lambda}{r}$, the extended protocol is indeed less efficient.

Since $\lambda > \lambda^*$, the payoff from *RDS* (if the servers follow the efficient protocol) is given by (3). Hence, for values of β such that

$$\frac{\lambda}{r}\left(\frac{1 - \Delta(\lambda)}{1 - D(\lambda)}\right) < (1 + \beta)\frac{\lambda^*}{r},$$

the principal prefers to introduce inefficiency in the work process by adopting the seemingly inefficient – extended protocol.

Finally, we note that in certain real-life scenarios, there are more work stations than servers. In particular, a work station may not be staffed as a result of fluctuation in the demand for service. For example, at certain times of the day, a border-control hall may have multiple unoccupied stations. These additional stations, unoccupied for much of the day, are likely there to accommodate peak hours at which multiple flights arrive. We do not model such fluctuation in demand in order to focus on the agents' moral hazard problem in the simplest possible setting. In the context of the border control example, one can think of our analysis as reflecting a shorter time frame, with m capturing the number of stations scheduled to be open during that time (rather than the total number of stations available).

Similarly, in the hospital scenario, there are more beds than physicians (or wards), but they treat several patients at the same time. This scenario naturally corresponds to a simple modification of our model where each server may occupy several work stations simultaneously. The incentives to release a patient depend on when the next patient is expected to arrive, and "idle time" can naturally be replaced with periods of lower work intensity. More generally, the assumption that $n > m$ is a simplification meant to capture in a simple model the forces that are also present in richer environments.

4 More vs. better servers

A natural question that arises is how does the total output of the system change with the servers' productivity. By Lemma 1, the output coincides with the first-best $m\lambda/r$ for $\lambda < \lambda^*$. For higher levels of productivity, the total output is a fraction $\left(\frac{1-\Delta(\lambda)}{1-D(\lambda)}\right)$ of the first-best. To see that the total output increases with λ consider two production rates $\lambda^* < \lambda_1 < \lambda_2$, and note that upon receiving a task assignment, the more productive server (the one with λ_2) incurs lower expected costs ($K + \frac{c}{\lambda_2} < K + \frac{c}{\lambda_1}$). Hence, this server requires a shorter break upon releasing a completed task. The total output of the system with λ_2 is therefore strictly higher than that with λ_1. Therefore, we can derive a uniform upper bound on the total output of the system with n servers and m work stations, by taking the servers' production rate to infinity.

Corollary 1. *The discounted expected output of n servers and $m < n$ work stations at the limit when $\lambda \to \infty$ is equal to*

$$(n-m)\left(1 + \frac{d}{rK}\right). \tag{8}$$

This follows from Propositions 1 and 2, taking the limit of (3) as $\lambda \to \infty$, noting that $\lim_{\lambda\to\infty} D(\cdot) = \frac{\frac{d}{r}}{K+\frac{d}{r}}$ and that $\lim_{\lambda\to\infty}\left(\frac{m\lambda}{r}(1 - (\frac{m\lambda}{m\lambda+r})^{n-m})\right) = n - m$.

While more productive servers complete more tasks per unit of time, they also shorten the idle time of servers who are not at work stations. Indeed, at an optimal equilibrium, as λ exceeds the threshold λ^*, servers increase the delay until they release completed tasks. This points at a trade-off between quantity and quality of servers. Note that this trade-off exists only because servers are strategic in their decision of when to release completed task. Put differently, higher quality servers amplify the moral hazard problem. In light of this, we ask the following question: Given the number of work stations m, how many mediocre servers (i.e., sufficiently slow servers so that simple rotation without delay is incentive compatible) are needed to outperform a group of $n > m$ unboundedly productive servers?

Recall that the production rate under *RDS* fluctuates over time (see the discussion at the beginning of Section 3). As an intermediate step, it is useful to consider a "uniform-rate system" that outputs a completed task every μ units of time. The uniform rate μ that generates a total output equal to (8) is given by $\mu = (n-m)(r + \frac{d}{K})$.

Consider a system of m work stations and sufficiently many non-strategic servers (i.e., servers who release tasks immediately upon completion). Clearly, if the completion rate of every server at a work station is μ/m, the output equals (8), independently of the exact number of servers. Let λ^*_{n+j} be the value of λ that solves $D(\lambda) = \Delta(\lambda)$ when there are $n+j$ servers. Since λ^*_{n+j} increases in j, we need to find the minimal j for which $\mu/m \leq \lambda^*_{n+j}$.

For the following Proposition, denote

$$D_{\lambda=\frac{\mu}{m}} \equiv \frac{\frac{d}{r}}{K + \frac{c}{\frac{\mu}{m}+r} + \frac{\frac{\mu}{m}}{\frac{\mu}{m}+r} \cdot \frac{d}{r}}.$$

Proposition 4. *Let*

$$j = \left\lceil \frac{\log D_{\lambda=\frac{\mu}{m}}}{\log(\frac{\mu}{\mu+r})} \right\rceil + m - n. \tag{9}$$

A system with m stations and $n+j$ servers with productivity rate μ/m generates the same discounted expected output as a system with m stations and n unboundedly productive servers.

When r is sufficiently close to zero, the expression in (9) becomes significantly simpler:

Corollary 2. *At the $r \to 0$ limit, a system with m stations and $n + m(\lceil c/d \rceil - 1)$ servers with productivity rate equal to μ/m generates the same discounted expected output as a system with m stations and n unboundedly productive servers.*

One implication of Corollary 2 is that when r is sufficiently low and c is close to d, even $n+1$ mediocre servers can generate the same discounted expected output as n unboundedly productive ones. The following example illustrates this.

Example. Let $r = 0.01, c = 2.5, d = 2, K = 1, m = 3$ and $n = 5$. By Corollary 1, the discounted expected output when $\lambda \to \infty$ is equal to $(5-3)(1 + 2/(0.01 \cdot 1)) = 402$. Let us derive the productivity rate that delivers this output when there are 6 servers who immediately release completed tasks under simple rotation. This is given by multiplying 402 by $r = 0.01$ yielding $\mu = 4.02$. Dividing by $m = 3$ we obtain $\lambda = 1.34$, which is the required production rate of each server needed to attain a total uniform production rate of μ for the entire system.

Next, we derive $\lambda^*_{n=6}$ for the above parameters by solving:

$$\frac{\frac{2}{0.01}}{1 + \frac{2.5}{\lambda^*_{n=6}+0.01} + \frac{\lambda^*_{n=6}}{\lambda^*_{n=6}+0.01} \cdot \frac{2}{0.01}} = \left(\frac{3\lambda^*_{n=6}}{3\lambda^*_{n=6} + 0.01} \right)^{6-3}.$$

The solution is $\lambda^*_{n=6} \approx 1.5$. It follows that it is indeed optimal for each of the six servers to immediately release completed tasks under simple rotation. Hence, six servers with $\lambda \approx 1.34$ generate the same discounted expected output as five unboundedly productive servers. $\square$

This result captures a natural tendency in organizations with features similar to our model to prioritize quantity over quality. It is worth noting, however, that this effect would be less pronounced in an alternative model that does not have an infinite supply of tasks. If tasks arrive randomly, a more productive team of workers would enjoy more idle time simply because they finish their work more quickly. While the assumption of the infinite supply of tasks may not seem adequate when the workers' productivity is taken to infinity, if we were to compare workers with different (but finite) productivity levels in settings with abundant tasks, the same qualitative conclusion would still hold.

5 Extensions

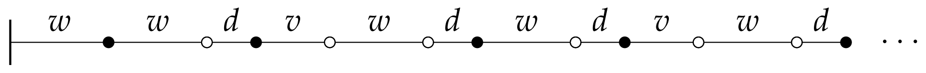
5.1 Vacating workstations

The model in Section 2 assumes an *appearance constraint*, requiring that all workstations remain utilized at all times. This reflects the notion that, for example, ER beds should not sit empty while patients wait, or that bank service windows should not appear idle–whether unstaffed or occupied by clerks on break–when customers are in line. The constraint captures the appearance cost organizations face from public pressure or bad publicity when customers observe idle resources despite delays in service.

While the appearance constraint is natural in many settings, it is useful to consider relaxing it. In this section, we allow workstations to be temporarily vacated at a cost. Specifically, recall that an allocation rule selects m servers at time zero and specifies a history-dependent distribution over available servers to determine who handles the next task when a workstation becomes vacant. We now extend this by allowing the allocation rule to shut down a workstation temporarily at a flow cost $z > 0$. For simplicity, we focus on the case with a single workstation ($m = 1$) and two servers ($n = 2$).

First note that, as in the baseline case, it is without loss to focus on allocation rules that are symmetric and time-invariant. That is, rules where, upon task release, both servers face the same IC constraint, regardless of the number of past releases. This is because any two rules satisfying the following must yield the same expected payoff: (i) the same discounted delay by each server; (ii) the same discounted time the workstation is shut down; (iii) the same order of activity–server 1 works first and releases before server 2 begins; and (iv) the first-release IC constraint binds for each server. Since both servers share the same discount factor, the principal's expected payoff depends only on the total expected compensation required to satisfy the IC constraints–not the precise timing of delays or shutdowns.[7]

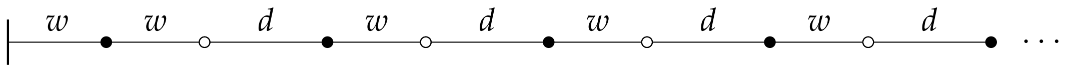
Since delay is possible only when a server completes a task, it is without loss of optimality to restrict attention to the family of mechanisms with the following dynamics. We refer to this family as σ-hybrid mechanisms.



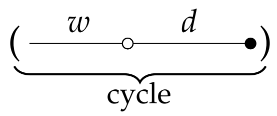
Here w, d, and v indicate phases of work, delay, and vacating the workstation, respectively, and 1 indicates the time at which the principal collects the payoff from a released task. Note that every server expects a break that includes exactly one of each of the phases w, v, and d, until the next task is assigned. The length of each delay and each vacating of a workstation are $\sigma \in (0, \tau)$ and $\tau - \sigma$, respectively, where τ is the delay time under RDS in the baseline model.

When $\sigma = \tau$, the σ-hybrid mechanism is simply the RDS mechanism. When $\sigma = 0$, there are no delays by the servers - a mechanism we refer to as the *VWS (vacating workstation) mechanism*. The optimal σ-hybrid mechanism will depend on the flow cost z of vacating a mechanism. We first compare the two cases of RDS and VWS, and then show that for any given cost z, the optimal mechanism is either one or the other.

It will be convenient to consider the payoffs from RDS and VWS starting from the release of the first task. Under RDS, the dynamics can be described as follows:



and are a repetition of cycles where a worker works and delays,



The payoff from the repeated cycle is $\frac{\lambda}{\lambda+r}e^{-r\tau}$. Recalling that

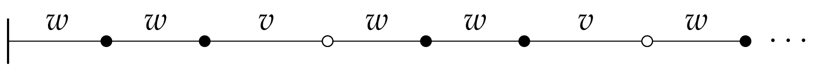
$$e^{-r\tau} = \frac{\frac{d}{r}(\lambda + r)}{\lambda\left(K + \frac{c}{\lambda+r} + \frac{\lambda}{\lambda+r}\frac{d}{r}\right)}, \tag{10}$$

[7]As in the baseline model with RDS, delays must occur only when idle servers have positive debt to ensure delays contribute to incentive provision. In this setting, the same must hold, and in addition, both servers must have positive debt for workstation shutdowns to be effective.

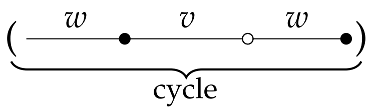
the payoff from RDS (starting from the time at which the first task is released) is

$$\frac{\lambda}{\lambda+r}e^{-r\tau}\sum_{i=0}^{\infty}\left(\frac{\lambda}{\lambda+r}e^{-r\tau}\right)^{i}=\frac{\lambda}{\lambda+r}e^{-r\tau}\frac{1}{1-\frac{\lambda}{\lambda+r}e^{-r\tau}}=\frac{d}{r}\frac{r+\lambda}{c-d+K\lambda+Kr}. \tag{11}$$

Turning to VWS, the dynamics take the form



where (starting from the time at which the first task is released) the following cycle is repeated and the length of time during which the workstation is vacated in any such cycle



is again τ. The payoff from each such cycle is

$$\frac{\lambda}{\lambda+r}+\left(\frac{\lambda}{\lambda+r}\right)^{2}e^{-r\tau}-z\frac{\lambda}{\lambda+r}\frac{1-e^{-r\tau}}{r},$$

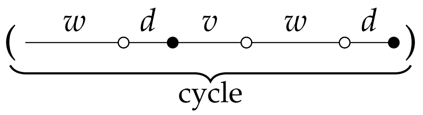
and hence the overall payoff from VWS, starting from the time at which the first task is released, is

$$\left(\frac{\lambda}{\lambda+r}+\left(\frac{\lambda}{\lambda+r}\right)^{2}e^{-r\tau}-z\frac{\lambda}{\lambda+r}\frac{1-e^{-r\tau}}{r}\right)\frac{1}{1-\left(\frac{\lambda}{\lambda+r}\right)^{2}e^{-r\tau}}. \tag{12}$$

Comparing (11) and (12), the cost z^* at which the two mechanisms yield the principal the same payoff is

$$z^{*}=\frac{d\lambda+Kr^{2}+cr+Kr\lambda}{c-d+K\lambda+Kr}. \tag{13}$$

We now show that for every value of z, either RDS or VWS is optimal. Under a σ-hybrid mechanism, after the first task (which is released without delay), the cycle



is repeated indefinitely. The payoff from each cycle is given by:

$$\begin{aligned}&\frac{\lambda}{\lambda+r}e^{-r\sigma}+\frac{\lambda}{\lambda+r}e^{-r\sigma}e^{-r(\tau-\sigma)}\frac{\lambda}{\lambda+r}e^{-r\sigma}-z\frac{\lambda}{\lambda+r}e^{-r\sigma}\frac{1-e^{-r(\tau-\sigma)}}{r}\\&=\frac{\lambda}{\lambda+r}e^{-r\sigma}\left(1+\frac{\lambda}{\lambda+r}e^{-r\tau}\right)-z\frac{\lambda}{\lambda+r}\frac{e^{-r\sigma}-e^{-r\tau}}{r},\end{aligned}$$

and the payoff from repeating this cycle forever is $f(\sigma)-z\cdot g(\sigma)$, where

$$f(\sigma):=\frac{\lambda}{\lambda+r}e^{-r\sigma}\left(1+\frac{\lambda}{\lambda+r}e^{-r\tau}\right)\frac{1}{1-\left(\frac{\lambda}{\lambda+r}\right)^2e^{-r\tau}e^{-r\sigma}}$$

and

$$g(\sigma):=\frac{\lambda}{\lambda+r}\frac{e^{-r\sigma}-e^{-r\tau}}{r}\frac{1}{1-\left(\frac{\lambda}{\lambda+r}\right)^2e^{-r\tau}e^{-r\sigma}},$$

and where $f(\cdot)$ and $g(\cdot)$ are taken for parameters that require delay in RDS and $\tau>0$ is determined accordingly.

Plugging in (10) and (13), a bit of algebra shows that

$$f(\sigma)-z^*\cdot g(\sigma)=\frac{d}{r}\frac{r+\lambda}{c-d+K\lambda+Kr},$$

which is precisely the principal's payoff from RDS and VWS. In other words, for the value of z that equates the payoffs from RDS and VWS, z^*, the principal's payoff is the same for all $\sigma\in(0,\tau)$.

It can easily be verified that the functions $f(\sigma)$ and $g(\sigma)$ are both decreasing in σ. Furthermore, for any $\sigma\in(0,\tau)$,

$$\frac{\partial}{\partial\sigma}\left(f(\sigma)-z^*\cdot g(\sigma)\right)=\frac{\partial}{\partial\sigma}f(\sigma)-z^*\cdot\frac{\partial}{\partial\sigma}g(\sigma)=0,$$

that is,

$$\frac{\partial}{\partial\sigma}f(\sigma)=z^*\cdot\frac{\partial}{\partial\sigma}g(\sigma).$$

Therefore, if $z<z^*$, then $\frac{\partial}{\partial\sigma}f(\sigma)<z\cdot\frac{\partial}{\partial\sigma}g(\sigma)<0$, and hence $\frac{\partial}{\partial\sigma}\left(f(\sigma)-z\cdot g(\sigma)\right)<0$. It is therefore profitable for the principal to reduce delay, which implies the corner solution $\sigma=0$. That is, VWS is optimal for all $z<z^*$. For costs $z>z^*$, $z\cdot\frac{\partial}{\partial\sigma}g(\sigma)<\frac{\partial}{\partial\sigma}f(\sigma)<0$, which means $\frac{\partial}{\partial\sigma}\left(f(\sigma)-z\cdot g(\sigma)\right)>0$. Hence, in this case it is profitable for the principal to increase delay, which yields the corner solution $\sigma=\tau$, meaning RDS is optimal.

Proposition 5. *If the workstation can be vacated at a flow cost of z, the optimal σ-hybrid mechanism is RDS(τ^*) RDS when $z > z^*$, and VWS when $z < z^*$.*

The optimal mechanism takes a bang-bang form: it is never optimal to combine server delays with vacating the workstation. Intuitively, the cutoff cost z^* increases with λ, r, and d, and decreases with c and K. A key implication is that the results from earlier sections continue to apply when workstation shutdowns are allowed, provided the *appearance cost* exceeds z^*. If the cost is below z^*, then inefficiency for incentive provision arises solely through workstation shutdowns, without delaying task releases.

5.2 Endogenous effort

In the previous sections we assumed that a server could not affect his rate of production λ and had only one decision to make: release a finished task or just sit on it. The motivation for this assumption was to isolate the effect of this new form of moral hazard on the server's behavior and the overall productivity of a team of servers. In this section we consider an extension of our basic model where the server's rate of production is endogenous. Specifically, a server now has two decisions to make: (1) an *effort decision* – whenever the task is unfinished, the server decides whether to work or to shirk, and (2) a *release decision* – whenever a task is finished, the server decides whether to release it or not (just as in our benchmark model). Working has a completion rate of λ and flow cost c (as before), while shirking has a completion rate of 0 and flow cost of d (like that of sitting on a completed task). To give the most transparent illustration of the additional hidden effort decision and its implications, we focus on the case with two servers, one work station, and $K = 0$.

The introduction of an effort decision has two important implications: (1) it affects the conditions under which the first-best is attainable, and (2) it affects the second-best level of output and the means of achieving it (when the first-best is unattainable). We begin by analyzing the first implication.

First best – Recall that in our benchmark model, we proved (Lemma 1) the existence of a unique cutoff productivity rate $0 < \lambda^* < \infty$ such that simple rotation without delay is incentive compatible (and hence, the first-best is attainable) if and only if the servers' productivity is $\lambda \leq \lambda^*$. This result was shown for $K > 0$. When $K = 0$, it is still true that if simple rotation without delay is incentive compatible for a given productivity rate λ, then it is also incentive compatible for $\lambda' < \lambda$.[8]

[8]In contrast to the $K > 0$ case, when $K = 0$ the threshold productivity level below which simple rotation without delay is incentive-compatible can also be zero or infinity for some parameter values.

Adding an effort decision effectively reverses this conclusion. Specifically, now there is a *lower* bound $0 < \underline{\lambda} \leq \infty$, which depends on the other parameters, such that simple rotation without delay is incentive compatible whenever $\lambda \geq \underline{\lambda}$. The lower bound is strictly positive and $\underline{\lambda} = \infty$ represents the possibility that simple rotation is never incentive compatible.

To see why, consider the *effort incentive constraint* under simple rotation without delay:

$$\frac{c}{\lambda + r} + \frac{\lambda}{\lambda + r} \cdot \frac{0}{\lambda + r} \leq \frac{d}{r}\left(1 - \left(\frac{\lambda}{\lambda + r}\right)^2\right). \tag{14}$$

On the left-hand-side we have the expected cost of exerting effort until a task is completed plus the expected cost of idle time (which is zero) during the period in which the other server works on his task. On the right-hand-side we have the alternative option of shirking and paying a flow cost of d for the same duration of time (i.e., the expected time it takes to complete two tasks). If $c \geq 2d$, the above constraint is violated for all λ, implying that simple rotation without delay is never incentive compatible. However, if $c < 2d$, the constraint holds whenever

$$\lambda \geq \frac{(c-d)r}{2d - c}.$$

Next, consider the *release incentive constraint*:

$$\frac{0}{\lambda + r} + \frac{\lambda}{\lambda + r} \cdot \frac{c}{\lambda + r} \leq \frac{d}{r}\left(1 - \left(\frac{\lambda}{\lambda + r}\right)^2\right). \tag{15}$$

The left-hand-side is the expected cost that is incurred when a server releases a completed task: He enjoys idle time while the other server works and then incurs the cost of effort on his next task. The right-hand-side is the flow cost of sitting on a completed task for the same duration.

From inequalities (14) and (15), it follows that the effort incentive constraint is the more demanding one: While the right-hand-side of both constraints is equal, the left-hand-side in the effort incentive constraint is strictly higher.

If simple rotation without delay is incentive compatible for some λ, why is it not incentive compatible for lower productivity rates when servers also make an effort decision (even though it is incentive compatible if servers only make a release decision)? The reason for this is the relative timing of the cost of effort and the benefit of idle time in the incentive constraints.

As is common in many dynamic economic models, in the effort constraint, the cost of effort *precedes* the reward. When servers are slow (low λ), having a long interval of working time and then an equally long interval of idle time may not be as attractive because of discounting (since the reward comes in the relatively distant future). When the productivity rate increases, the time intervals of work and being idle shrink identically, but the effect of discounting becomes weaker; hence, simple rotation without delay may become incentive compatible. However, in the release constraint, the order is reversed: First, the servers enjoy idle time, and only later they incur the cost of effort (on the next task). Since, as shown above, the effort constraint implies the release constraint, effectively, the latter constraint can be ignored. By contrast, in our benchmark model, the release incentive constraint is the only constraint and hence, the binding one.

Second best — We now turn to analyze the second-best outcome when simple rotation without delay is not incentive compatible. A naive upper bound on the total expected output (discounted to the time of the first task assignment) can be obtained by multiplying the first-best output, $\frac{\lambda}{r}$ by the ratio of the available budget for incentive provision to the total expected costs that must be covered under the first best. The available budget is the value of idle time starting from the end of the first task (the first task wastes idle time for the server who has not worked yet): $\frac{\lambda}{r+\lambda}\frac{d}{r}$. The costs that need to be covered consist of a constant flow cost of effort c net of the inevitable cost of d due to being allocated a task, i.e., $\frac{c-d}{r}$. Hence, the upper bound is given by,

$$\frac{\lambda}{r} \times \left\{ \left(\frac{\lambda}{\lambda + r} \frac{d}{r} \right) \bigg/ \left(\frac{c-d}{r} \right) \right\},$$

which simplifies to

$$\frac{d}{r} \frac{\lambda^2}{(r+\lambda)(c-d)}. \tag{16}$$

This upper bound completely ignores the constraints imposed by the dynamic structure of the environment and simply identifies which fraction of the required budget to incentivize first-best is available, and assumes that all of it can be utilized efficiently to incentivize productive effort.

We now show that this upper bound can be attained by a scheme similar to RDS, but with one important difference: instead of delaying the release of a finished task, the servers shirk before starting work. To develop the intuition for this, we begin by discussing the difference between the first- and second-best analyses in light of the effort and release incentive constraints and the relation between them.

In the first-best analysis above, only two of the server's actions must be incentive-compatible: *working* (when the task is unfinished) and *releasing* (when the task is finished)

When the first-best is unattainable, the second-best requires that servers will also be willing to either *delay* or *shirk* (at least occasionally). Suppose first that, as in RDS, we make the *delay* of a finished task incentive compatible in a way that does not depend on the calendar time. This implies that, at any point in time after finishing the task, the server is indifferent between delaying and releasing it, i.e., the release IC constraint holds with equality. However, the effort incentive constraint remains more demanding than the release incentive constraint even when the first-best outcome is not attainable. As a result, when the release incentive constraint holds with equality, working on an unfinished task becomes suboptimal. On the other hand, making the actions *shirk* and *work* optimal when the server is with an unfinished task (by choosing the expected length of the break following release such that the effort IC constraint holds with equality) makes immediate release upon completion uniquely optimal.

Assume that finished tasks are released immediately and let us first find the shirk time σ^* that makes the effort IC constraint hold with equality. This σ^* satisfies

$$\frac{c}{\lambda + r} + \frac{\lambda}{\lambda + r} e^{-r\sigma^*} \frac{\lambda}{\lambda + r} \frac{d}{r} = \frac{d}{r}. \tag{17}$$

This expression reflects the server's indifference between shirking forever and working on the current task, releasing it upon completion and enjoying idle time until being assigned the next task, at which point the server will receive the continuation value that corresponds to shirking forever. It is easy to see that for such shirk duration σ^*, the release constraint is satisfied and holds with strict inequality since

$$e^{-r\sigma^*} \frac{\lambda}{\lambda + r} \left(\frac{c}{\lambda + r} + \frac{\lambda}{\lambda + r} \frac{d}{r} \right) < \frac{c}{\lambda + r} + \frac{\lambda}{\lambda + r} e^{-r\sigma^*} \frac{\lambda}{\lambda + r} \frac{d}{r} = \frac{d}{r}.$$

Again, the left-hand-side corresponds to the payoff from releasing a finished task – enjoying idle time during the other server's shirking and working followed by own working until completion on the next assigned task and never releasing it. This gives strictly lower expected costs compared to $\frac{d}{r}$, that corresponds to sitting on the current finished task forever. In other words, releasing finished tasks is always strictly preferred.

Solving (17) for $e^{-r\sigma^*}$ yields

$$e^{-r\sigma^*} = \left(\frac{d}{r} - \frac{c}{r + \lambda} \right) \frac{r\,(r + \lambda)^2}{d\lambda^2}. \tag{18}$$

We have the following result.

Proposition 6. *Simple rotation where, from the second task onward, servers shirk prior to start working for σ^* units of time attains the upper bound specified in (16).*

5.3 Heterogeneous production rates

Up until now we maintained the assumption of symmetric production rates among servers. Under this assumption, inefficiency is fully captured by the overall amount of delay. However, with heterogeneous production rates, overall performance depends not only on whether the working servers are delaying completed tasks, but also on which servers are working. Hence, with heterogeneous production rates the goal of finding optimal allocation mechanisms and characterizing the servers' equilibrium behavior is much more challenging. However, achieving this goal - even in simple cases - leads to new insights and novel comparative statics.

To illustrate this, we focus on a simple but important scenario: There is one work station and two servers, and simple rotation without delay is incentive-compatible. When servers are symmetric, this implies that the first best is trivially attained since all finished tasks are released immediately. On the other hand, when the servers have heterogeneous production rates, say $\lambda > \mu > 0$, the first best requires that only the most productive server works at all times, which is clearly not incentive compatible. Our question is, what is the second best in this case?

Proposition 7. *Assume that simple rotation without delay is incentive-compatible. Then the principal's optimal (second-best) expected discounted payoff is given by*

$$\frac{\lambda}{\lambda + r}\left(1 + \frac{\varphi\lambda + (1-\varphi)\mu}{r}\right),$$

where

$$\varphi = \frac{d}{c + K(\lambda + r)}.$$

The expression for the second-best output can be understood as follows. The first task is allocated to the most productive server (say server 1), who releases it without delay (this generates a payoff of $\lambda/(\lambda + r)$). From that point onwards, the system's production rate is the weighted average of the two production rates, where the weights are determined by the parameters as stated in the proposition.

The proof of the proposition proceeds as follows. First, we identify a naive upper bound on the fraction of first-best output that can be produced by server 1 in any incentive compatible scheme, beginning from the first task release. As in previous sections, the idea behind this upper bound is to ignore the constraints arising from the dynamic nature of the problem. This upper bound is equal to the value of φ in the proposition.

Next, we show that this upper bound can be attained via a specific allocation rule. Moreover, under the allocation rule that we propose, the equilibrium behavior prescribes no delays to any of the servers. Hence, at every instant, either server 1 or server 2 will put active effort (no delays), which explains the weighted average form of the servers' productivity rates as stated in the proposition.

To prove that the naive upper bound is attainable we propose the *stochastic repetition scheme* under which: Server 1 is assigned the initial task; whenever server 1 releases a task, the next task is also assigned to server 1 with probability p and it is assigned to server 2 with probability $1 - p$; and whenever server 2 releases a completed task, the next task is assigned to server 1 with certainty.

The above result offers several interesting comparative statics observations. First, if the servers become less patient, or if the costs related to working on a task (c, K) increase, then the amount of tasks that will be allocated to the most productive server will *decrease* – which will decrease the total productivity of the system. Perhaps more interesting is that two systems that are identical in all parameters except for d will perform differently at the second-best even when neither system exhibits any delay (specifically, the system with a higher d will be more productive).

Recall that when simple rotation without delay is incentive compatible, $p > 0$ and so server 1 is assigned more tasks than server 2. Even when simple rotation is not incentive compatible, there are some situations in which the above stochastic-repetition scheme still induces both servers to immediately release finished tasks. To see this, suppose that the idle time that server 1 gets in simple rotation without delay is enough to incentivize him to release completed tasks immediately (i.e., server 2 is sufficiently slow). If this is the case, we can find $p \geq 0$ that makes server 1 indifferent between releasing a finished task or not. If given this p, the idle time that server 2 gets is enough to incentivize him to immediately release finished tasks, we are done (i.e., the scheme induces immediate releases of finished tasks and gives more tasks to server 1). If the idle time for server 2 is not enough, we need to inject delay into the scheme. This raises the following questions: Is it optimal only for server 2 to delay? Should server 1 be the only one delaying? It is optimal for both servers to delay? These questions are left for future research.

5.4 Premature release

In this subsection we consider the possibility that servers can release unfinished tasks. For example, in the hospital setting, a patient may be discharged before sufficient time has elapsed during which his symptoms did not recur; in a setting of a public sector office, an application may proceed to the next step before all forms were filled and signed. Such premature releases are costly to the principal if only finished tasks contribute to his payoff. In addition, unfinished tasks may return to be properly finished, causing delays in the processing of new awaiting tasks. Continuing the above examples, a patient who is discharged prematurely is more likely to become sick again (perhaps even more severely than before), and may return to the ER or to another hospital for further treatment; incomplete applications by individuals who are eligible for the service they request may lead to wrongful rejections, or may be sent back to the department that released it.

The possibility to release unfinished tasks means that upon receiving a task, a server now decides at each point in time (both before and after finishing a task) whether to release it. For a server, releasing an unfinished task is tempting as it saves the flow cost of working on a task and allows the server to enjoy the idle time sooner. For the principal, unfinished tasks do not contribute to her payoff and she would therefore like to prevent their release. A key insight in this section is that even when the principal can use threats of punishments (within the allocation rule) to make premature releases an off-path phenomenon, such threats may require inefficient delays that occur on-path.

To illustrate this we focus on the following environment. There are two servers and a single work station ($n = 2, m = 1$). After a server is assigned a task (and pays the cost of K) he decides at each point in time whether to release it. If an unfinished task is released, then with probability p it returns immediately and it is publicly known which server released it. The principal gets a payoff of 1 whenever a *finished* task is released and a payoff of 0 if a task is released prematurely (suppose that these payoffs are not immediately observed so the principal cannot learn from them). We focus on the case where both servers have productivity $\lambda \leq \lambda^*$ so that in the benchmark where premature releases are ruled out, simple rotation without delay is incentive compatible and achieves the first-best.

To deter premature releases, the harshest punishment that the principal can commit to is to allocate all future tasks to the server who was found to release an unfinished task (which happens when an unfinished task returns). The severity of this punishment is limited for two reasons: (i) a server can avoid the costs associated with all future tasks by sitting indefinitely on the returning unfinished task (which a server must accept), and (ii) premature releases are caught only with probability p. Still, for a sufficiently high p, the threat of this punishment can incentivize the servers to attain the first-best by following a simple rotation without delay.

Proposition 8. *The principal can attain the first-best if and only if p is at least*

$$p^* = \frac{\frac{c}{\lambda+r} + \frac{r}{\lambda+r}u}{K + \frac{c}{\lambda+r} + \frac{\lambda}{\lambda+r}\frac{d}{r} - u},$$

where

$$u = \frac{\lambda\left(c + K\left(\lambda+r\right)\right)}{r(2\lambda+r)}.$$

While the exact characterization of the threshold p^* requires a formal argument, the existence of a range of p values sufficiently close to 1, for which the first best can be achieved, is quite intuitive. Similarly, since the threat of punishment for a returning task is limited, when p is sufficiently close to 0, premature releases cannot be deterred. Interestingly, there is a range of intermediate p values for which premature releases can be completely deterred. However, this requires deliberately reducing the system's productivity. Specifically, we show below that in addition to p^* there is another threshold p^{**} such that premature releases can be deterred only if $p \geq p^{**}$ and importantly, $p^{**} < p^*$. Thus, as can be seen in Figure (4), for intermediate values of p, premature releases can be made an off-equilibrium event but task releases will be inefficiently delayed. Hence, from the perspective of an outside observer who is aware of the workers' abilities, the system will appear inefficiently slow.

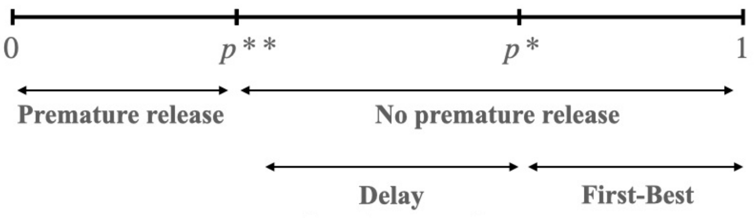


Figure 4: Release and Delay as a function of p.

To understand the intuition for the intermediate range, consider $p = p^*$ and suppose that a server is now working on an unfinished task. Since $p = p^*$, the server is indifferent between following simple rotation without delay and prematurely releasing the task. Consider a slightly lower value of p. Now, the expected payoff from premature release is higher. Since the principal is already using the most severe punishment for returning tasks, the only way to deter premature releases for this lower p is to raise the expected payoff from releasing a finished task. This can only be done if other servers produce less efficiently, or, strategically delay their task releases.

However, delay cannot be voluntary as in *RDS* because servers who are incentivized not to release early have a strict incentive to release finished tasks. One way to inject delay is with the following scheme, which we call *simple rotation with minimal release time*. In this scheme servers take turns working on tasks, but if a server releases a task before time T, or if the task returns, then that server is assigned all future tasks. Thus, the minimal release time T introduces delay into the system.

The minimal delay T that deters premature releases is derived as follows. Let s^* be a strategy that calls for a server to work on a task until completion. If it takes less than T units of time to finish the task, then the server sits on the finished task until T units of time have passed at which point he released it. Otherwise, he releases it immediately upon completion. Consider a server who did not finish a task after T units of time following a history in which both servers followed s^*. Let $\tilde{s}$ be a deviation from s^* which calls for this server to release the unfinished task, and to continue playing according to s^* if the task does not return. But if the task does return, then $\tilde{s}$ calls for the server to sit on the returning task indefinitely. The minimal delay time T satisfies that the server is indifferent between deviating to $\tilde{s}$ and continuing to play according to s^*.

To express this formally, suppose both servers follow s^*. Let u^* denote the continuation cost of a server at the point in time when a task is completed, and let v^* be the continuation cost at the point in time in which the task is allocated to the server. Then

$$\begin{aligned} v^* = K + \int_0^T \lambda e^{-\lambda t}\left(\int_0^t e^{-rs}cds + \int_t^T e^{-rs}d\,ds\right) + \int_T^\infty \lambda e^{-\lambda t}\int_0^t e^{-rs}cdsdt \\ + \left(\int_0^T \lambda e^{-\lambda t}e^{-rs}ds + \int_T^\infty \lambda e^{-\lambda t}\int_0^t e^{-rs}dsdt\right)u^* \end{aligned}$$

and

$$u^* = \left(e^{-rT} - \frac{r}{\lambda + r}e^{-(\lambda+r)T}\right)v^*$$

It follows that the minimal delay T satisfies

$$p\left(K + \frac{c}{\lambda + r} + \frac{\lambda}{\lambda + r}\frac{d}{r}\right) + (1-p)u^* = \frac{c}{\lambda + r} + \frac{\lambda}{\lambda + r}u^*. \tag{19}$$

There is a finite T that solves this equation if p is greater than

$$p^{**} = \frac{\frac{c}{\lambda+r}}{K + \frac{c}{\lambda+r} + \frac{\lambda}{\lambda+r}\frac{d}{r}}.$$

To see why, note that when $T \to \infty$, $u^* \to 0$ and p^{**} is the value of p that satisfies equation (19) in this case.

It follows that when $p \in (p^{**}, p^{*})$ there is a minimal amount of delay for which simple rotation with minimal release time deters premature releases. An interesting feature of this mechanism (and any other mechanism that deters premature releases) is that when $p < p^{*}$ only finished tasks are released and yet there is inefficient delay even though $\lambda < \lambda^{*}$. Thus, for an outside observer who knows that $\lambda < \lambda^{*}$ the system will appear to be unnecessarily inefficient. Note that this variation of our model offers an additional rationale for delays as means to attain the second-best output. Specifically, delays are needed to generate incentives even when the servers are sufficiently slow. This suggests that delays (or, more generally, inefficient work practices) can be even more widespread than what they seem to be in our original model.

6 Concluding remarks

We opened the paper with the example of Haemek Hospital, where the administration realized that doctors at the internal wards were delaying the release of patients in response to the allocation rule that assigned a new patient to whichever ward had an empty bed. This meant that when wards were working at full capacity, the first ward to release a patient was then assigned a new patient from the ER. To address this problem, the administration severed the link between releasing a patient and being assigned a new one: the responsibility for new patients was assigned to wards using simple rotation regardless of the number of empty beds in that ward.

The insight from this story is that the inherent moral hazard of strategically delaying the release of finished tasks can be addressed by weakening the relation between which server is assigned a task and which server just finished working on one. Indeed, this is a key feature of the optimal allocation rules that we characterize. When servers are identical in their productivity, it is optimal for a server who releases a finished task to enter a queue, and he will be assigned a new task when he gets to the front of that queue. When servers have diverse productivity levels, optimality requires a balance between efficiency - which calls for the most efficient server to always work - and incentive compatibility - which calls to let servers have idle time once they finish working on a task. This means that compared with the slowest server, the fastest server is more likely to be immediately assigned a new task upon releasing a finished one, but this probability must be strictly lower than one.

Optimally addressing the moral hazard problem that we highlight in this paper gives rise to interesting (and somewhat unintuitive) features of service systems: when servers are sufficiently productive and patient, more work stations reduce total output, and in some cases it is better to have more servers of lower quality.

References

Armony, M., G. Roels, and H. Song (2021). Pooling queues with strategic servers: The effects of customer ownership. *Operations Research 69*(1), 13–29.

Bird, D. and A. Frug (2019). Dynamic non-monetary incentives. *American Economic Journal: Microeconomics 11*(4), 111–150.

Bray, R. L., D. Coviello, A. Ichino, and N. Persico (2016). Multitasking, multiarmed bandits, and the Italian judiciary. *Manufacturing & Service Operations Management 18*(4), 545–558.

Coviello, D., A. Ichino, and N. Persico (2014). Time allocation and task juggling. *American Economic Review 104*(2), 609–623.

Coviello, D., A. Ichino, and N. Persico (2015). The inefficiency of worker time use. *Journal of the European Economic Association 13*(5), 906–947.

Dagan, O., S. Lichtman-Sadot, I. Shurtz, and D. Zeltzer (2024). Strategic effort and congestion reduction: Evidence from an emergency department routing reform. Working paper.

De Clippel, G., K. Eliaz, D. Fershtman, and K. Rozen (2021). On selecting the right agent. *Theoretical Economics 16*(2), 381–402.

Eliaz, K., D. Fershtman, and A. Frug (2022). On optimal scheduling. *American Economic Journal: Microeconomics, Forthcoming*.

Forand, J. G. and J. Zápal (2020). Production priorities in dynamic relationships. *Theoretical Economics 15*(3), 861–889.

Frankel, A. (2016). Delegating multiple decisions. *American economic journal: Microeconomics 8*(4), 16–53.

Gavazza, A. and A. Lizzeri (2007). The perils of transparency in bureaucracies. *American Economic Review 97*(2), 300–305.

Geng, X., W. T. Huh, and M. Nagarajan (2015). Fairness among servers when capacity decisions are endogenous. *Production and operations management 24*(6), 961–974.

Gopalakrishnan, R., S. Doroudi, A. R. Ward, and A. Wierman (2016). Routing and staffing when servers are strategic. *Operations Research 64*(4), 1033–1050.

Guo, Y. (2016). Dynamic delegation of experimentation. *American Economic Review 106*(8), 1969–2008.

Guo, Y. and J. Hörner (2020). Dynamic allocation without money. Working paper.

Li, J., N. Matouschek, and M. Powell (2017). Power dynamics in organizations. *American Economic Journal: Microeconomics 9*(1), 217–241.

Linder, R. (2019). Without a single shekel: The secret of the hospital that managed to reduce the load in the ER and wards. *TheMarker*.

Lipnowski, E. and J. Ramos (2020). Repeated delegation. *Journal of Economic Theory 188*, 105040.

Mylovanov, T. and P. W. Schmitz (2008). Task scheduling and moral hazard. *Economic Theory 37*, 307–320.

Appendix

Proof of Lemma 1. We first show that there is at most one $\lambda > 0$ that solves the equation $\Delta(\lambda) = D(\lambda)$, or equivalently,

$$\left(\frac{m\lambda}{r+m\lambda}\right)^{n-m} = \frac{\frac{d}{r}}{K + \frac{c}{\lambda+r} + \frac{\lambda}{\lambda+r}\cdot\frac{d}{r}}. \tag{20}$$

To do so, we rewrite this equation as a polynomial in λ that equals zero:

$$0 = -\left(m^{n-m}K\right)\lambda^{n-m+1} + m^{n-m}\left(\frac{n}{m}d - Kr - c\right)\lambda^{n-m}$$
$$+ \sum_{i=1}^{n-m-1}\left[d\binom{n-m}{i}m^i r^{n-m-i} + \frac{d}{r}\binom{n-m}{i-1}m^{i-1}r^{n-m-i+1}\right]\lambda^i + d.$$

Note that the coefficient on the largest exponent $\left(\lambda^{n-m+1}\right)$ is negative and the coefficients on all the positive exponents from $n-m-1$ to 1 are positive. The coefficient on λ^{n-m}, the second largest exponent, can be either negative or positive. Descartes' rule of signs states that if the nonzero terms of a single-variable polynomial with real coefficients are ordered by descending variable exponent, then the number of positive roots of the polynomial is at most the number of sign changes between consecutive (nonzero) coefficients. Since the sign changes in the above polynomial are either $(-,-,+,...,+)$ or $(-,+,...,+)$, there is at most one positive root (i.e., at most one positive solution for λ).

We now show that there must exist a positive solution to (20). Note that both sides are continuous in λ. When $\lambda = 0$, the LHS is zero whereas the RHS is positive. When $\lambda \to \infty$, the LHS tends to one while the RHS tends to a constant below one. Hence, there must be some positive λ where both sides equate.

Finally, note that $\lambda < \lambda^*$ implies $D(\lambda) < \Delta(\lambda)$, which guarantees that simple rotation without delay is incentive compatible whenever $\lambda < \lambda^*$. ■

Proof of Proposition 1. Let $x := \frac{m\lambda}{m\lambda+r}$. The proof extends the arguments discussed in Section 3.3 to the case of general n and m. The total expected payoff can be written as the expected payoff from the first $n-m$ tasks that are released without delays, and an infinitely repeated cycle of $n-m$ tasks where only the first is delayed. The payoff, discounted to time 0 from the first $n-m$ tasks is $x + x^2 + ... + x^{n-m} = x\frac{1-x^{n-m}}{1-x}$ and the expected payoff from a single cycle, discounted to the beginning of the cycle) is $e^{-r\tau}x\frac{1-x^{n-m}}{1-x}$.

The discounting between the beginning of a given cycle and the next one is $x^{n-m}e^{-r\tau}$, so the payoff from infinite repetition of the cycle, from the beginning of the first cycle, is

$$\sum_{j=0}^{\infty}(x^{n-m}e^{-r\tau})^j \cdot \left(e^{-r\tau}x\frac{1-x^{n-m}}{1-x}\right) = \frac{1}{1-x^{n-m}e^{-r\tau}}\left(e^{-r\tau}x\frac{1-x^{n-m}}{1-x}\right). \tag{21}$$

Hence, the total expected discounted payoff from $RDS(\tau)$, discounted to time 0, is

$$\left(x\frac{1-x^{n-m}}{1-x}\right) + x^{n-m}\frac{1}{1-x^{n-m}e^{-r\tau}}\left(e^{-r\tau}x\frac{1-x^{n-m}}{1-x}\right) = \frac{x}{1-x}\frac{1-x^{n-m}}{1-x^{n-m}e^{-r\tau}}$$

which, since $\frac{x}{1-x} = \frac{m\lambda}{r}$, simplifies to (3). ■

Proof of Proposition 3. We begin with two simple observations. First, note that if the number of work stations is weakly greater than the number of servers, total output is zero. Hence, to examine the effects of adding a work station, suppose $m < n-1$.

Second, the assumption that RDS features positive delay implies that, from the proof of Proposition 2, the upper bound representing the total incentive budget $\mathcal{B}$ (given by the RHS of equation (4)) is attainable. Hence, the proof consists of showing that the RHS of (4) decreases with m when r is sufficiently small.

To establish this, it is sufficient to show that, for r sufficiently small, $\mathcal{B}(m)/\mathcal{B}(m+1) > 1$, where $\mathcal{B}(m)$ denotes the total incentive budget with m work stations (holding the other model parameters fixed). Substituting the expression from the RHS of equation (4) for m and $m+1$, the above inequality is equivalent to

$$\frac{m}{m+1}\frac{\left(\frac{m\lambda}{m\lambda+r}\right)^{n-m}-1}{\left(\frac{(m+1)\lambda}{(m+1)\lambda+r}\right)^{n-(m+1)}-1} > 1.$$

Taking the limit as r approaches zero. we obtain the condition

$$\frac{m}{m+1}\left(\frac{m+1}{m}\frac{m-n}{m-n+1}\right) > 1.$$

This condition indeed holds since $\frac{m-n}{m-n+1} > 1$ if and only if $m < n-1$, where the latter holds by assumption. ■

Proof of Proposition 4. Recall that *RDS* reduces to simple rotation (without delay) when $\lambda = \lambda_n^*$, i.e., when servers are exactly indifferent between releasing a completed task and sitting on it indefinitely when using a simple rotation scheme. In light of this, we proceed by finding the minimal integer j that satisfies $\lambda_{n+j}^* \geq \mu/m$. Recall that λ_{n+j}^* is the solution to

$$\frac{\frac{d}{r}}{K + \frac{c}{\lambda_{n+j}^* + r} + \frac{\lambda_{n+j}^*}{\lambda_{n+j}^* + r} \cdot \frac{d}{r}} = \left(\frac{m\lambda_{n+j}^*}{m\lambda_{n+j}^* + r} \right)^{n+j-m},$$

where the LHS captures the idle time between tasks required to make a server indifferent between releasing or sitting on a completed task (it is the discount factor that multiplies the server's expected future costs at the time of task completion), while the RHS captures the idle time between tasks (via the discount factor that multiplies future expected costs) under simple rotation. Hence, $\lambda_{n+j}^* \geq \mu/m$ if $j = \lceil x \rceil$, where x is the solution to

$$\frac{\frac{d}{r}}{K + \frac{c}{\mu/m + r} + \frac{\mu/m}{\mu/m + r} \cdot \frac{d}{r}} = D_{\lambda = \frac{\mu}{m}} = \left(\frac{\mu}{\mu + r} \right)^{n+x-m}. \tag{22}$$

Taking the logarithm of both sides of the equation yields the desired result. ■

Proof of Corollary 2. From (22) it follows that at the $r \to 0$ limit,

$$j = \left\lceil \lim_{r \to 0} \frac{D_{\lambda = \frac{\mu}{m}}}{\frac{\mu}{\mu + r}} \right\rceil + m - n.$$

Since

$$\lim_{r \to 0} D_{\lambda = \frac{\mu}{m}} = \lim_{r \to 0} \frac{\log \left(\frac{d(n-m) + Knr}{d(n-m) + \left(\frac{K^2 nr^2}{d^2} \right) + \frac{c}{d} Kmr + 2Kr(n-m)} \right)}{\log \left((n-m) \frac{d + Kr}{Kr + d(n-m) + Kr(n-m)} \right)} = m\frac{c}{d} + (n - 2m),$$

we obtain the desired result. ■

Proof of Proposition 6. The following is the principal's expected payoff (discounted to time zero) from the second task onward when each server shirks for a fixed duration of

$\sigma^* > 0$ prior to starting work:

$$\frac{\lambda}{\lambda + r} \sum_{n=0}^{\infty} \left(e^{-r\sigma^*} \frac{\lambda}{\lambda + r} \right)^n .$$

Since $e^{-r\sigma^*} \frac{\lambda}{\lambda+r} < 1$, this expected payoff can be written as

$$\frac{\lambda}{\lambda + r} \cdot \frac{1}{1 - e^{-r\sigma^*} \frac{\lambda}{\lambda+r}},$$

which simplifies to $\lambda / (r + \lambda - \lambda e^{-r\sigma^*})$. Plugging (18) into this expression yields

$$\frac{\lambda}{r + \lambda - \lambda \left(\frac{d}{r} - \frac{c}{r+\lambda} \right) \frac{r(r+\lambda)^2}{d\lambda^2}},$$

which simplifies to (16). ■

Proof of Proposition 7. We first derive a naive upper bound on the output that can be attained from server 1. Suppose that at the time in which server 1 decides whether to release a completed task, we conduct the following lottery: with probability φ server 1 will continue working forever (releasing all completed tasks upon completion), and with the complementary probability server 1 will get idle time indefinitely. Consider the value of φ that makes server 1 indifferent between accepting the lottery and sitting on the task indefinitely, i.e., the value of φ that solves the equation:

$$\varphi \left(K + \int_0^{\infty} e^{-rt} \left(c + \lambda K \right) dt \right) = \frac{d}{r}.$$

The solution is given by

$$\varphi = \frac{d}{c + K(\lambda + r)}.$$

Hence, an upper bound on the expected future output, discounted to the time of the first release (i.e., the time when the lottery is performed) is $\varphi\lambda / r$.

Let us now compute the total expected discounted output of server 1 following the release of the first task (discounted to this point in time) in the stochastic repetition scheme where p is chosen to make server 1 indifferent between releasing a finished task or not. To do this, we first derive the expected discount factor between the time at which a task was assigned to server 1 and the earliest subsequent time at which a task was assigned to server

2. This is given by the following expression:

$$(1-p)\frac{\lambda}{\lambda+r}+p(1-p)\left(\frac{\lambda}{\lambda+r}\right)^2+\quad ... \quad = \frac{\lambda(1-p)}{r+\lambda(1-p)}.$$

Next, we compute the expected discount factor from the time at which a task was assigned to server 2 and the earliest subsequent time at which a task is assigned to server 1. This is given by $\mu/(r+\mu)$. Finally, the discounted expected output generated by server 1 from the time at which he was assigned a task until the earliest time at which a task is assigned to server 2 is given by:

$$\frac{\lambda}{\lambda+r}+p\left(\frac{\lambda}{\lambda+r}\right)^2+p^2\left(\frac{\lambda}{\lambda+r}\right)^3+\quad ... \quad = \frac{\lambda}{r+\lambda(1-p)}.$$

It follows that the total expected output generated by server 1, discounted to the time at which the first task is released, is given by:

$$p\frac{\lambda}{r+\lambda(1-p)}\left(1+\frac{\mu}{r+\mu}\frac{\lambda(1-p)}{r+\lambda(1-p)}+\left(\frac{\mu}{r+\mu}\frac{\lambda(1-p)}{r+\lambda(1-p)}\right)^2+\ ...\right)$$
$$+(1-p)\frac{\mu}{r+\mu}\frac{\lambda}{r+\lambda(1-p)}\left(1+\frac{\mu}{r+\mu}\frac{\lambda(1-p)}{r+\lambda(1-p)}+\left(\frac{\mu}{r+\mu}\frac{\lambda(1-p)}{r+\lambda(1-p)}\right)^2+\ ...\right),$$

which reduces to

$$\frac{\lambda}{r}\frac{\mu+pr}{\lambda(1-p)+\mu+r}. \tag{23}$$

The value of p that makes server 1 indifferent between releasing a finished task and not doing so solves the equation:

$$\frac{d}{r}=p(K+\frac{c}{\lambda+r}+\frac{\lambda}{\lambda+r}\frac{d}{r})+(1-p)\frac{\mu}{\mu+r}(K+\frac{c}{\lambda+r}+\frac{\lambda}{\lambda+r}\frac{d}{r}),$$

for which the solution is

$$p=\frac{d(\lambda+r)-\mu(c-d)-K\mu(\lambda+r)}{\lambda r(\frac{d}{r}+K)+r(c+Kr)}.$$

Plugging this into (23) yields the desired upper bound $\varphi\lambda/r$.

Finally, note that our assumption that simple rotation is incentive compatible implies that, in optimum, $p>0$, which in turn implies that it is strictly optimal for server 2 to release tasks immediately upon completion under the stochastic repetition scheme, completing the proof. ■

Proof of Proposition 8. Assume that whenever a server i releases an unfinished task and that task returns, server i is assigned all future tasks. Let s be the strategy that calls for a server to work on each task until completion and then release it immediately. The first-best output is achieved when both servers follow s. Let s' be a strategy with the following features: when a server is first assigned a task, he releases it immediately; if the task returns (in which case, all subsequent tasks will be assigned to this server), he sits on the task indefinitely; if the task does not return, the server follows strategy s. Let h be some history in which server i is assigned a task after both servers played according to s. The profile s is an equilibrium if and only if there is no history h after which a server gains by deviating to s'. I.e., if and only if

$$p\left(K+\frac{c}{\lambda+r}+\frac{\lambda}{\lambda+r}\frac{d}{r}\right)+(1-p)u \geq \frac{c}{\lambda+r}+\frac{\lambda}{\lambda+r}u, \tag{24}$$

where u is the continuation cost from following s evaluated at the point of releasing a task; i.e., u solves

$$u=(\frac{\lambda}{\lambda+r})(K+\frac{c}{\lambda+r}+\frac{\lambda}{\lambda+r}u),$$

which yields the expression for u given in the statement of the proposition.

It follows that the LHS of inequality (24) can be written as

$$p\left[\frac{r}{\lambda+r}\left(K+\frac{c}{\lambda+r}\right)+\frac{\lambda}{\lambda+r}\left(\frac{d}{r}-\frac{\lambda}{\lambda+r}u\right)\right]+u.$$

Since $\lambda \leq \lambda^*$ it follows that $\frac{d}{r} \geq u$, and hence, the LHS of inequality (24) increases in p, and is greater or equal than the RHS when $p=1$. It follows that there exists $p=p^*$ at which (24) holds with equality, which yields the expression in the statement of the proposition.

If $p<p^*$, then inequality (24) is violated. This implies that s is not an equilibrium and hence, the first-best cannot be attained. ■