Rolling OSS Schedules: Ex-Post (ϵ-)Internal Stability and Log-Share Fairness under Churn

Liz Lemma Future Detective

January 16, 2026

Abstract

Stable matching mechanisms with ties face a sharp fairness barrier: even randomized stable matchings can yield linear losses, while allowing (internally stable) lotteries recovers tight $\Theta(\log N)$ share guarantees via the Optimal Stable Share (OSS) ratio. This paper pushes OSS fairness into dynamic markets, where workers and jobs arrive and depart and where preference ties are induced by coarse ratings and statistical indistinguishability. We propose ROSS, a rolling time-sharing mechanism that recomputes an internally stable lottery using a duplication-index construction on each epoch and then implements it as a deterministic rotating schedule. ROSS guarantees ex-post ϵ-internal stability in every round and ensures that each worker's cumulative utility is at least a $1/O(\log N_{\max})$ fraction of their dynamic optimal stable share (defined round-by-round), up to an additive churn term that scales with the number of epoch interruptions experienced by that worker. Our analysis adapts the source paper's duplication-index and directed-forest arguments to a time-indexed setting using a churn-aware potential that quantifies losses from incomplete schedule cycles. The results provide a tractable fairness–stability theory for modern 2026 platforms (gig work, public service rosters, rotating reviewer assignments) where static stability is too brittle and ties are ubiquitous.

Table of Contents

3. 3. Model: time-varying worker/job sets, strict job priorities, worker utilities with ties; definitions of (weak) stability, internal stability, ϵ-stability; define churn/epochs.

4. 4. Benchmarks: per-round OSS $U_t^{\epsilon,*}(w)$ and dynamic OSS $\sum_t U_t^{\epsilon,*}(w)$; discussion of why stronger benchmarks (global dynamic stability) are infeasible/ill-posed under churn.

5. 5. Mechanism (ROSS): epoch construction, duplication-index oracle per epoch, schedule implementation as cyclic rotation; computational complexity and implementability.

6. 6. Main results: ex-post ϵ-internal stability each round; dynamic OSS guarantee with log factor and churn additive penalty; tightness and interpretation.

7. 7. Extensions: (i) bounded utility drift and robustness; (ii) uncertain utilities (rectangular confidence sets) and ϵ calibration; (iii) optional recourse constraints (when numerical methods may be needed).

8. 8. Lower bounds and necessity of churn penalty: impossibility of churn-free guarantees under epoch interruptions; connections to static $\Omega(\log N)$ and $\Omega(N)$ barriers when restricting to stable matchings.

9. 9. Empirical protocol (optional): how to estimate dynamic OSS proxies from logs; simulation design for gig/task markets; metrics for envy incidents and share fairness.

10. 10. Policy and platform implications: rotating schedules, auditability, and stability as a compliance constraint; guidance on choosing epoch length and ϵ.

11. 11. Conclusion and open problems: decentralization, multi-stakeholder objectives, and learning under churn.

1 Introduction and motivation

Many allocation problems that are modeled as two-sided matching are, in practice, *dynamic* and *noisy*. Gig platforms match workers to short-lived tasks; hospitals and staffing agencies reassign shifts as demand fluctuates; cloud systems schedule jobs on heterogeneous machines under changing availability; and even ostensibly "static" assignment problems (e.g., course allocation or public-housing placement) are repeatedly revisited as participants arrive, leave, or update their information. In such environments the platform does not choose a single matching once and for all, but a *sequence* of matchings over time. This repeated choice creates an economic design problem that is distinct from the classical one-shot stable matching model: we must simultaneously control (i) *incentives and legitimacy* in each period (captured here through stability-type constraints), and (ii) *intertemporal fairness* (captured through how utility accrues across rounds).

A key friction is that stability notions are inherently *local*—they constrain the set of matchings at a given time—whereas fairness concerns are inherently *global*—they evaluate a worker's realized utility over many rounds. Static stability alone is therefore not a sufficient policy prescription. To see this, imagine a worker who, in each round, is matched under a stable matching to a "safe" but mediocre job because job priorities favor incumbents. If the platform recomputes a stable matching myopically each round, the same worker may be systematically left out even when there exists another stable matching at that round that would give them a high-utility job. This pathology is not merely a matter of bad tie-breaking: it is structural. When multiple stable matchings exist, picking one deterministically can create persistent winners and losers, and the repeated setting amplifies these disparities.

The second friction is informational: worker values are often observed only coarsely. Platforms typically estimate match quality from sparse feedback, noisy performance measures, or discretized ratings. As a result, worker utilities $U_t(w, a)$ naturally contain *ties*. Ties matter because they enlarge the set of stable outcomes but also complicate the platform's credibility: small perturbations can flip preferences, and strict stability can become unattainable or overly restrictive. In the presence of ties, it is often more realistic to require a tolerance parameter $\epsilon \geq 0$, interpreting ϵ as a minimal gain needed to justify a deviation (or, operationally, as robustness to utility estimation error). This leads to ϵ-stability and its variants, which allow us to separate economically meaningful deviations from those driven by noise.

These two frictions together motivate the central question we study: *Can a platform guarantee each worker a nontrivial fraction of what they could obtain under an ϵ-stable matching in each round, while maintaining (ex-post) stability constraints in every realized round of play?* Our benchmark is deliberately individual-centric. For a worker w present at time t, we consider

their best attainable stable utility,

$$U_t^{\epsilon,*}(w) \;:=\; \max_{\mu \in S_t^\epsilon} U_t\big(w, \mu(w)\big),$$

and we compare the realized cumulative utility to the dynamic sum of these per-round "best stable shares." This benchmark is demanding: it allows the stable matching to vary by worker and by round, and thus captures the strongest individualized notion of "what was stably feasible for me." We view this as an appropriate fairness yardstick in applications where workers evaluate outcomes relative to what the market could have offered them *without undermining stability.*

A natural idea is to use *randomization* across stable matchings. In a static market, if we could draw a stable matching from a distribution each period, then in expectation each worker might receive a fairer share of the stable surplus. However, this is precisely where a fundamental limitation appears. The source material establishes a sharp "logarithmic barrier": even in a *static* one-shot instance, there are markets in which no distribution over stable matchings can guarantee every worker more than an $O(1/\log N)$ fraction of their own best stable utility, where N is the number of workers. Conversely, there exist constructions—based on duplicating opportunities and carefully selecting a small support of stable matchings—that *achieve* a $1/O(\log N)$ share for all workers simultaneously. Economically, this tells us that fairness via time-sharing is possible but unavoidably limited by congestion in the stable set: some workers' best stable outcomes are mutually incompatible, and spreading them across time requires a support whose size must grow logarithmically.

Our contribution is to transport this insight into a dynamic environment with arrivals and departures. Dynamics introduce a new obstacle: even if we have a good distribution over stable matchings for a fixed participant set, churn may interrupt the intended time-sharing schedule before each worker has had a chance to realize their "good" outcome. This is not a second-order concern. In short-lived markets (e.g., tasks that clear quickly, or workers who are active only briefly), naive cycling can perform poorly because it front-loads some workers' gains and back-loads others'. Moreover, in a genuinely dynamic market, requiring full stability with respect to *unmatched* agents can be too stringent: newly arriving agents would retroactively create blocking pairs against a matching that was stable for the previously present population, making it impossible to maintain stability without constant rematching.

For these reasons, we focus on an *internal* stability requirement: we rule out only those blocking pairs (w, a) in which both agents are currently matched to someone else. This captures a practically relevant notion of stability—no *two matched* agents should have a credible deviation—while allowing the platform to manage entry and exit without being forced into

immediate global reoptimization. The distinction is not merely technical. In many platforms, unmatched agents (or agents who just arrived) do not have an enforceable claim to displace an existing match instantaneously; instead, they queue, wait for the next batch, or are integrated at an epoch boundary. Internal stability formalizes this operational reality.

Within this modeling choice, we propose a rolling schedule policy (ROSS) built around two ideas. First, we partition time into *epochs*, maximal contiguous intervals over which the set of participants is constant. Second, at the start of each epoch we compute a small collection of ϵ-internally stable matchings with support size

$$m \;:=\; \lceil \log_2 N_{\max} \rceil + 2,$$

and then *rotate* deterministically through them round by round. The deterministic cycle is an implementation device: it emulates the uniform lottery over this support without requiring per-round randomization, while preserving ex-post internal stability in every realized matching. Intuitively, the policy turns "random time-sharing" into "scheduled time-sharing," which is often preferable in practice because it is auditable and predictable.

The performance guarantee has the economic form we would expect from a time-sharing argument with churn. Within a long epoch, the rotation ensures that each worker obtains, on average, a $1/m$ fraction of their per-round optimal ϵ-stable utility, up to an ϵ slack that accounts for the tolerance in stability. When epochs end early due to arrivals or departures, some workers may miss their scheduled high-utility assignment; we can bound this loss additively by the number of epoch interruptions the worker experiences. This yields a guarantee that decomposes neatly into (i) a multiplicative fairness term governed by $\log N_{\max}$, and (ii) an additive churn penalty governed by the worker's exposure to entry/exit events. The model thus illuminates a tradeoff that practitioners routinely face: frequent recomputation (to reflect churn) improves responsiveness but can harm intertemporal fairness by interrupting rotation; infrequent recomputation improves fairness within an epoch but may ignore important changes.

We also emphasize what our framework does *not* attempt to do. We do not claim to fully solve incentive compatibility in the dynamic setting; rather, we take priorities as exogenous and focus on stability as a legitimacy constraint. We also do not assume that utilities are cardinally meaningful beyond being normalized to $[0, 1]$; the point of the OSS benchmark is comparative—each worker is compared to what was stably achievable for them—not to maximize utilitarian welfare. Finally, while our baseline analysis takes U_t as observed at time t, the motivating interpretation is noisy measurement. This is why ϵ-stability is not simply a mathematical relaxation: it is a robustness knob that can be calibrated to the resolution of the platform's estimates, and it allows us to reason about stable scheduling policies even when preferences are coarsened or drifting.

In summary, the dynamic market setting forces us to marry two desiderata that are typically studied separately: per-round stability and across-round fairness. The static literature tells us that we cannot escape a logarithmic loss when guaranteeing individualized shares relative to stable benchmarks. Our goal is to show that, with an appropriate notion of stability and an explicit accounting for churn, we can extend these log-tight fairness guarantees to dynamic environments using a simple, implementable rolling schedule.

2 Related work

Our setting sits at the intersection of several literatures: dynamic two-sided matching and scheduling, stability under indifferences (ties) and approximate stability, time-sharing and fractional/randomized matchings, online allocation with fairness objectives, and learning-based (bandit) matching. We briefly situate our contribution relative to each.

Dynamic matching and repeated assignment. Classical matching theory focuses on one-shot outcomes—most prominently the Gale–Shapley model and its many variants—and provides a rich understanding of stability, lattice structure, and comparative statics under fixed populations and preferences **??**. Many applied markets, however, operate through repeated assignment with changing availability. This has motivated dynamic models in labor markets (e.g., re-matching over time), school choice with moving cohorts, and operational settings such as shift scheduling and job allocation. Several strands are relevant. One line studies *dynamic* or *intertemporal* matching where agents arrive and depart and the planner chooses a policy to trade off immediate matching against waiting for future participants **??**. Another line, closer to operations, views the platform as a scheduler repeatedly selecting matchings under feasibility constraints; these models often prioritize throughput or welfare and treat strategic or stability considerations as secondary **??**. Our focus differs in that we treat stability-type constraints as a per-round legitimacy requirement and ask what intertemporal fairness guarantees remain feasible under such constraints. In particular, by organizing time into epochs with fixed participant sets, we separate the operational source of nonstationarity (churn) from the within-epoch problem of selecting among multiple stable matchings.

Stability, priorities, and ties. We take strict job-side priorities as primitives, consistent with many institutional environments (hospitals ranking residents, schools ranking students by policy rules, or platforms ranking workers by reliability scores). The theory of stable matching with priorities underlies much of the market design literature **??**. At the same time, practical

implementations often face coarse information and thus indifferences on one or both sides. Stability with ties has been extensively studied both for its algorithmic complexity and for its welfare/fairness implications; the "stable marriage with ties" and related problems highlight that ties enlarge the stable set but can also make selection more delicate **??**. In applications, ties can arise from discretized scores, measurement error, or deliberate policy coarsening; these motivate relaxations such as weak stability, super-stability, and approximate notions that ignore negligible gains. Our use of an ϵ-tolerance is aligned with this robustness interpretation: deviations must clear a minimal utility improvement threshold to be considered credible. This is also in the spirit of "approximate stability" and perturbation-based arguments used to make stability robust to noise **??**. Conceptually, ϵ plays a dual role: it reflects estimation resolution and it expands the feasible set so that a platform can commit to a stable-enough policy without overreacting to small fluctuations.

Internal versus external stability in dynamic environments. A recurring challenge in dynamic markets is that full stability with respect to all present agents can be too brittle: new arrivals can instantly create blocking pairs, forcing constant rematching and undermining predictability. Several papers address this by weakening stability, for example by restricting which deviations are actionable (e.g., only certain coalitions, or only deviations that respect timing/contracting frictions) **??**. Our "internal" stability requirement—ruling out only blocking pairs among agents who are currently matched—can be interpreted as formalizing the operational frictions that prevent immediate displacement in many platforms (queues, batching, onboarding delays, or explicit non-preemption rules). Related notions appear in scheduling and matching with commitments, where previously assigned matches cannot be arbitrarily overturned without incurring costs or violating constraints **??**. The point of this restriction is not to weaken incentives indiscriminately, but to capture a realistic boundary on who can credibly deviate at a given moment.

Time-sharing, randomization, and fractional matchings. A central tool in our analysis is the idea of *time-sharing* across feasible matchings to improve intertemporal fairness. In assignment problems, randomization and fractional allocations are classical devices: Birkhoff–von Neumann decompositions represent fractional matchings as distributions over integral matchings, and mechanisms such as probabilistic serial implement ordinally fair random allocations **?**. In two-sided markets, there is also a literature on randomized or fractional stability concepts (ex ante stability, fractional stable matchings) and on selecting among multiple stable matchings to achieve distributional objectives **??**. Our contribution connects these ideas to a

worst-case, individual-centric fairness benchmark: we compare each worker to their own best ϵ-stable outcome and ask what guarantee is possible uniformly over workers. The static source result we build on establishes a sharp logarithmic limitation and a matching constructive upper bound. In the dynamic setting, the analogous issue is not only whether a good distribution exists, but whether it can be implemented over time when the population changes and the schedule can be interrupted.

Online allocation with fairness objectives. Beyond matching theory, there is a growing body of work on fairness in online resource allocation, often framed through competitive ratios, max–min objectives, proportional fairness, or regret relative to offline benchmarks **??**. Many of these models study online bipartite matching, ad allocation, or scheduling where the planner seeks to maximize total welfare subject to fairness constraints across groups or individuals. Our benchmark is distinct: we do not compare to a utilitarian optimum, but to the per-round ϵ-stable opportunity frontier for each worker. This creates a normative standard that is tightly linked to legitimacy: the platform is only held accountable relative to what could have been delivered without creating (approximate) blocking incentives. The resulting guarantees resemble online-fairness bounds in that they combine multiplicative approximation (a share factor) with additive penalties that reflect nonstationarity—here captured by churn/epoch interruptions rather than adversarial arrivals.

Learning and bandit matching. In many platforms, utilities are not observed directly and must be learned from interaction. This motivates bandit and reinforcement-learning models with matching constraints, including "matching bandits," combinatorial semi-bandits on bipartite graphs, and contextual bandit formulations where edges have unknown rewards **???**. The typical objective is to minimize regret to the best fixed matching or to an offline optimal policy, and the main difficulty is exploration under combinatorial structure. Our baseline analysis abstracts from learning by treating U_t as observed (or as lying in a known uncertainty set), because our focus is on the feasibility and fairness limits imposed by stability. Nevertheless, the ϵ-stability framework is naturally compatible with learning: ϵ can be chosen to dominate estimation error so that the platform maintains credible outcomes while exploring, and our epoch-based recomputation can be viewed as a batching device that reduces instability from frequent policy changes. Incorporating full bandit feedback and endogenizing ϵ is an important direction, but orthogonal to the structural question we isolate: how far time-sharing can go when we insist on per-round stability.

Summary. In short, the existing literature offers powerful tools for either (i) ensuring per-period stability in static environments, or (ii) optimizing welfare/fairness in dynamic and online allocation without stability constraints, or (iii) learning utilities under matching constraints. Our goal is to bridge a specific gap between these strands: we ask for a policy that is ex-post stable (up to ϵ) in every realized round, yet still guarantees each worker a quantifiable fraction of their individualized stable opportunity over time, with explicit degradation under churn. This framing clarifies which fairness losses are inherent (the logarithmic barrier) and which are operational (epoch interruptions), thereby connecting theoretical limits to design choices that platforms actually face.

3 Model

We study a repeated one-to-one matching environment in which a platform must make an assignment decision in every round while the set of available participants changes over time. The goal of the model is to isolate two distinct sources of difficulty that arise in practice: (i) even in a fixed population, stability restrictions limit how much the platform can "time-share" desirable assignments across workers; and (ii) when workers or jobs arrive and depart, any attempt to implement time-sharing can be interrupted, creating an additional and operationally meaningful loss.

Agents, rounds, and feasibility. Time is discrete, indexed by rounds $t \in \{1, \dots, T\}$. In each round t, a set of workers W_t and a set of jobs/tasks A_t are present and eligible for matching. We write $N_t := |W_t|$, $K_t := |A_t|$, and $N_{\max} := \max_t N_t$. A *matching* at round t is a partial one-to-one assignment between W_t and A_t with an explicit outside option $\perp$. Formally, μ_t maps each worker $w \in W_t$ to either a job $\mu_t(w) \in A_t$ or $\mu_t(w) = \perp$; symmetrically, each job $a \in A_t$ is mapped to $\mu_t(a) \in W_t$ or $\mu_t(a) = \perp$, with the consistency constraint $\mu_t(w) = a$ if and only if $\mu_t(a) = w$. We interpret $\mu_t(w) = \perp$ as being unmatched (idle, waiting, or assigned to an outside alternative), and we normalize the utility from $\perp$ to be zero.

This feasibility constraint captures operational settings where each worker can handle at most one task per round and each task can be served by at most one worker (e.g., a shift, a job ticket, a delivery, or a micro-task). While richer capacity constraints are relevant in many applications, the one-to-one case already reveals the core stability–fairness tradeoffs and allows the strongest analytical guarantees.

Preferences, priorities, and cardinal utilities with ties. Jobs have *strict priorities* over workers. For each job a, let P_a denote a strict total order over the universe of workers (or, equivalently, over the workers who

may ever appear). We write $w \succ_a w'$ if job a strictly prioritizes worker w over w'. In many platforms, these priorities reflect a policy rule or score (tenure, reliability, certifications, or compliance checks) and are taken as exogenous constraints rather than strategic choices.

Workers evaluate jobs using cardinal utilities. In each round t, worker $w \in W_t$ has a utility $U_t(w, a) \in [0, 1]$ for each job $a \in A_t$, with ties allowed (so distinct jobs may yield the same value). We set $U_t(w, \perp) = 0$. Allowing ties is important empirically: platforms often discretize relevance scores, workers may have coarse preferences across similar tasks, and measurement noise can make strict rankings unrealistic. The use of cardinal utilities also lets us formulate approximate stability in a way that cleanly separates "small" deviations from meaningful ones.

Timing and what the platform controls. Each round proceeds in three steps:

1. Nature reveals the current participant sets (W_t, A_t) and the contemporaneous utility matrix U_t (or, in extensions, the platform observes an uncertainty set or noisy feedback).

2. The platform selects a feasible matching μ_t on $W_t \times A_t$.

3. Utilities realize and accrue: worker w receives $U_t\big(w, \mu_t(w)\big)$.

The platform's decision rule can depend on the history and on current information. Our analysis focuses on policies that must satisfy a per-round stability-type constraint, reflecting legitimacy, compliance, or non-preemption requirements.

Stability notions and blocking pairs. Because jobs have priorities and workers have utilities (with possible ties), we use a stability definition tailored to priority-based institutions. Fix a round t and a matching μ feasible for (W_t, A_t). A worker–job pair $(w, a) \in W_t \times A_t$ is said to *block* μ if two conditions hold: (i) job a strictly prefers w to its current assignee, i.e., $w \succ_a \mu(a)$ (where we interpret $\mu(a) = \perp$ as the lowest-priority outcome); and (ii) worker w strictly prefers a in utility terms to their current assignment, i.e., $U_t(w, a) > U_t\big(w, \mu(w)\big)$. When utilities admit ties, this is the standard "weak" improvement requirement on the worker side: the worker must gain strictly in utility for the deviation to be compelling.

We will also work with an ϵ-tolerant version. For a fixed $\epsilon \geq 0$, we say that (w, a) *ϵ-blocks* μ at round t if

$$w \succ_a \mu(a) \qquad \text{and} \qquad U_t(w, a) > U_t\big(w, \mu(w)\big) + \epsilon.$$

This relaxation treats deviations with negligible gains (below ϵ) as non-credible, which is natural when utilities are estimated, discretized, or when

switching costs make small improvements irrelevant. Conceptually, ϵ will play two roles in the paper: it enlarges the feasible set of matchings that qualify as "stable enough," and it provides robustness to small perturbations of U_t.

External versus internal stability. In a static model, one typically requires that no blocking pair exists among all present agents. In a dynamic environment, that requirement can be brittle: an arriving worker may instantly create a blocking pair with an already-matched job, forcing immediate displacement if we insist on full (external) stability each round. Many platforms in fact cannot (or choose not to) preempt assignments in this way, either due to contractual commitments, batching, onboarding delays, or fairness norms about not bumping currently served workers.

We therefore distinguish two notions.

External ϵ-stability (the stronger requirement) rules out ϵ-blocking pairs (w, a) regardless of whether w or a are currently matched.

Internal ϵ-stability (our baseline requirement) only considers deviations among agents who are currently matched. Formally, a feasible matching μ at round t is *ϵ-internally stable* if there does not exist a pair (w, a) such that both w and a are matched in μ (i.e., $\mu(w) \neq \perp$ and $\mu(a) \neq \perp$), with

$$w \succ_a \mu(a) \qquad \text{and} \qquad U_t(w, a) > U_t\big(w, \mu(w)\big) + \epsilon.$$

This requirement enforces a credible "no justified envy with significant gain" condition among the participants who are actually trading in that round. It is weaker than full stability, but it aligns with non-preemption: unmatched workers cannot claim an immediate right to displace an ongoing assignment, while matched workers and matched jobs should not have an obvious mutually beneficial swap that violates the priority rule.

Churn, epochs, and the structure of nonstationarity. The participant sets (W_t, A_t) may change arbitrarily over time. To model this churn in a way that is both operationally interpretable and analytically useful, we partition time into *epochs*: maximal contiguous intervals over which the sets (W_t, A_t) remain constant. Let E denote the number of epochs, indexed by $e \in \{1, \ldots, E\}$. For each epoch e, let $T_e \subseteq \{1, \ldots, T\}$ be its set of rounds (a contiguous block), and let W^e and A^e be the fixed worker and job sets throughout that epoch. We write $L_e := |T_e|$ for the epoch length.

Epochs provide a natural unit for recomputation: when the pool of eligible participants changes, the platform may need to re-run a matching procedure or re-compute a schedule. This recomputation, in turn, can interrupt any deterministic rotation that was intended to share opportunities across workers. For an individual worker w, define the set of active rounds $\mathcal{T}(w) := \{t : w \in W_t\}$, and let $E(w)$ be the number of epochs e such that

$w \in W^e$. The quantity $E(w)$ serves as a worker-specific churn exposure measure: it counts how many times w experiences an "epoch boundary" while present, and thus how many times a time-sharing schedule can be cut short. When we summarize churn at the market level, we may also refer to aggregate measures such as $E-1$ (the number of boundaries) or a symmetric-difference volume $\sum_t |W_t \Delta W_{t-1}| + |A_t \Delta A_{t-1}|$; however, the guarantees we derive will be indexed most cleanly by $E(w)$.

Finally, while our baseline exposition treats stability relative to the contemporaneous utilities U_t, later results allow for slow within-epoch drift in utilities. Intuitively, if U_t changes gradually, then a matching computed at the start of an epoch remains approximately stable throughout the epoch provided ϵ is calibrated to dominate the drift. This connects the model to practice: platforms typically recompute infrequently and rely on stability notions that are robust to small score changes.

4 Benchmarks: per-round OSS and its dynamic extension

To state meaningful guarantees in a changing market, we need a benchmark that (i) respects the institutional constraint encoded by job priorities, (ii) is well-defined even when the set of participants varies across rounds, and (iii) is strong enough to make "time-sharing" nontrivial, yet not so strong that it becomes infeasible or conceptually ill-posed under churn. We therefore benchmark performance worker-by-worker against an *optimal stable share* computed *round-by-round*, and then aggregate this quantity over the rounds in which the worker is present.

Per-round optimal stable share (OSS). Fix a round t and worker $w \in W_t$. Let S_t^ϵ denote the set of feasible matchings at time t that satisfy the (external) ϵ-stability notion. We define worker w's *per-round optimal ϵ-stable share* as

$$U_t^{\epsilon,*}(w) \;:=\; \max_{\mu \in S_t^\epsilon} U_t\big(w, \mu(w)\big),$$

with the convention $U_t(w, \perp) = 0$. This is the best utility that worker w could obtain in round t among matchings that clear the market subject to the priority constraints (up to ϵ). Importantly, this benchmark is *individualized*: two workers may have very different $U_t^{\epsilon,*}(\cdot)$ in the same round, because priorities and feasibility can make certain jobs unattainable in any stable outcome for one worker while remaining attainable for another.

Two features of $U_t^{\epsilon,*}(w)$ are worth emphasizing. First, it is stability-aware rather than welfare-optimal: if worker w could achieve utility 1 by taking a job that would immediately be "justifiably" reclaimed by a higher-priority worker, then that assignment is excluded from S_t^ϵ, and hence excluded from

the benchmark. Second, the benchmark is deliberately permissive about which stable outcome is chosen: we do not privilege the worker-optimal or job-optimal stable matching, nor any particular selection rule. Instead, $U_t^{\epsilon,*}(w)$ asks only whether *some* stable matching can award worker w a given utility level. This aligns with our goal of understanding what can be achieved through rotation or randomization over stable matchings: if there exists at least one stable matching that gives w a high-value job, then in principle a platform could try to allocate that job to w occasionally while staying within the stability constraint each round.

Dynamic OSS (dOSS): aggregating across the worker's active rounds. In a dynamic environment, workers are not present in every round, and the feasible set changes when participants arrive or depart. For this reason, the natural extension of the per-round OSS is additive over the worker's active rounds. Let $\mathcal{T}(w) := \{t : w \in W_t\}$. We define the *dynamic optimal stable share* benchmark as

$$\mathrm{dOSS}_\epsilon(w) \;:=\; \sum_{t \in \mathcal{T}(w)} U_t^{\epsilon,*}(w).$$

This quantity is an *offline* benchmark in the sense that it uses the realized utilities U_t in each round. It is also *non-strategic* and *round-local*: it does not attempt to couple decisions across time, but instead measures the total "stable opportunity" available to w in the realized sequence of markets.

We stress that $\mathrm{dOSS}_\epsilon(w)$ is not the value of any single matching or policy; rather, it upper bounds what a worker could hope to obtain if, in every round t, the platform selected whichever ϵ-stable matching most benefited that worker in that round. As such, it is an intentionally strong yardstick for fairness-style guarantees: if a policy can secure a constant (or logarithmic) fraction of $\mathrm{dOSS}_\epsilon(w)$ for every worker simultaneously, then it is allocating, over time, a significant share of each worker's *stable* opportunities.

Why we do not benchmark against a globally coupled "dynamic stability" notion. One might ask whether a more ambitious benchmark is appropriate—for example, optimizing a worker's cumulative utility subject to a stability constraint that is enforced *across time*, or requiring that the entire sequence $(\mu_t)_{t=1}^T$ be stable in some intertemporal sense. Under churn, such benchmarks quickly become problematic for three related reasons.

First, feasibility and existence become delicate. If we require stability with respect to all agents who are present at any time (a "grand market"), then most worker–job pairs are never simultaneously feasible, and stability constraints become ambiguous: should an absent worker be treated as having an outside option in that round, or as being unable to deviate? Different modeling choices lead to different, often incompatible, notions of blocking.

Even if one formalizes absences by adding outside options, the resulting notion can force the platform to behave as if it could commit future capacity to workers not yet present (or retain commitments to departed agents), which is not operationally meaningful.

Second, strong dynamic stability can be incompatible with non-preemption under churn. Consider a simple two-round example with a single job a. In round 1, only worker w_1 is present, so any stable matching assigns a to w_1. In round 2, a higher-priority worker w_2 arrives with $w_2 \succ_a w_1$. Any notion of per-round external stability in round 2 would assign a to w_2, displacing w_1. A global benchmark that tries to preserve w_1's round-1 advantage into round 2 (or that penalizes displacement) is no longer a stability benchmark; it is instead a commitment or tenure benchmark. Such commitments may be desirable in some applications, but they are institution-specific and should be modeled explicitly (e.g., via contracts, switching costs, or service-level agreements). Our objective here is different: we take the priorities as the "hard" constraint and study what fairness can be recovered by rotating among stable outcomes, given that churn disrupts rotation.

Third, globally optimal dynamic benchmarks are not decomposable and can hide arbitrary value in the choice of intertemporal constraints. Any benchmark that allows the platform to trade off utility across time (e.g., matching a worker to a low-value job today to secure a high-value job tomorrow) requires specifying what commitments carry over, what information is known when, and whether a job is allowed to "hold" capacity. These are important modeling decisions, but they are orthogonal to the stability–time-sharing tension we wish to isolate. In contrast, $\mathrm{dOSS}_\epsilon(w)$ is deliberately modular: it measures, round by round, what was stably attainable *given the realized market in that round*, and then sums those opportunities over the worker's presence.

A note on internal stability and the role of ϵ. Our algorithmic constraint will be ϵ-*internal* stability, reflecting non-preemption and operational frictions. The benchmark $\mathrm{dOSS}_\epsilon(w)$ is nevertheless defined using (external) ϵ-stable matchings S_t^ϵ, for two reasons. First, external stability provides a conservative, institutionally interpretable notion of what a worker could receive without being justifiably displaced by an unmatched higher-priority worker; thus it serves as a clean "best case" notion of stable opportunity. Second, the parameter ϵ makes the benchmark robust to ties, discretization, and small perturbations in the utilities, which will matter when we later allow within-epoch drift. In our eventual guarantees, the gap between internal implementability and external benchmarking is handled explicitly via additive terms (notably the $\epsilon|\mathcal{T}(w)|$ discount and the churn exposure term).

Connection to time-sharing and the next section. The benchmarks above formalize what it means to give each worker a fair share of the stable assignments available to them over time. The remaining question is mechanistic: how can the platform, in each epoch with fixed participants, construct a small set of ϵ-internally stable matchings whose rotation ensures that every worker attains a controlled fraction of $U_t^{\epsilon,*}(w)$, and how does churn limit the ability to realize that rotation without interruption? Section 5 answers these questions by describing the rolling schedule policy (ROSS), which computes an $O(\log N_{\max})$-sized support of internally stable matchings per epoch and then implements a cyclic schedule across rounds.

5 Mechanism: Rolling Optimal Stable Shares (ROSS)

We now describe the rolling schedule policy (ROSS). The guiding idea is simple: within any time interval in which the participant sets are fixed, we can precompute a *small* collection of ϵ-internally stable matchings that "cover" workers' best ϵ-stable opportunities; we then *rotate* through this collection deterministically, so that time averages mimic the fairness of an explicit lottery while maintaining ex-post implementability each round.

Step 0: epochs as the unit of recomputation. Because arrivals and departures change feasibility and stability constraints, ROSS recomputes only when the active sets change. Formally, we partition the horizon into *epochs*, where an epoch e is a maximal contiguous block of rounds $T_e \subseteq \{1, \dots, T\}$ such that $W_t = W^e$ and $A_t = A^e$ for all $t \in T_e$. Let $t_e := \min T_e$ denote the epoch start. Within an epoch, priorities $(P_a)_{a \in A^e}$ are fixed by assumption, and we take the relevant utility matrix to be U^e (in the baseline version, $U^e = U_{t_e}$ is observed at the epoch start and treated as constant within the epoch).

This "recompute-on-churn" design has an operational interpretation: the platform updates a roster only when the roster becomes invalid (because the set of available workers or jobs changes), but otherwise it can run a simple, transparent rotation rule.

Step 1: a duplication–index construction within an epoch. Fix an epoch e with worker set W^e, job set A^e, utilities U^e, and priorities P_a. ROSS constructs a list of matchings

$$\widetilde{\mu}_{e,1},\ \widetilde{\mu}_{e,2},\ \dots,\ \widetilde{\mu}_{e,m}, \qquad m := \lceil \log_2 N_{\max} \rceil + 2,$$

each of which is ϵ-internally stable in the epoch. The construction is based on a standard "duplicate capacity to create time slots" reduction: we temporarily enlarge the market by giving each job m identical *copies*, compute

one stable matching in that enlarged market, and then interpret each copy index $i \in \{1, \ldots, m\}$ as a round in the rotation.

Concretely, define the duplicated job set

$$A^{e,\times} \;:=\; A^e \times \{1, \ldots, m\}.$$

Each duplicated job (a, i) inherits job a's priority order P_a over workers. Utilities do *not* depend on the copy index:

$$U^{e,\times}(w, (a, i)) \;:=\; U^e(w, a) \qquad \forall w \in W^e,\ a \in A^e,\ i \in \{1, \ldots, m\}.$$

We then compute a feasible matching $\widehat{\mu}_e$ between W^e and $A^{e,\times}$ that is ϵ-stable in the duplicated instance (with the natural outside option $\perp$). Intuitively, because each job appears m times, $\widehat{\mu}_e$ is a single assignment of workers to *indexed time slots* of jobs, consistent with priorities.

At this point, we define m matchings in the original market by *projecting* $\widehat{\mu}_e$ onto each index. For each $i \in \{1, \ldots, m\}$, the matching $\widetilde{\mu}_{e,i}$ is given by

$$\widetilde{\mu}_{e,i}(w) \;:=\; \begin{cases} a & \text{if } \widehat{\mu}_e(w) = (a, i) \text{ for some } a \in A^e, \\ \perp & \text{otherwise,} \end{cases} \qquad \forall w \in W^e,$$

and $\widetilde{\mu}_{e,i}(a)$ is defined by the one-to-one consistency condition. Feasibility is immediate: since $\widehat{\mu}_e$ matches each worker to at most one copy, each worker appears in at most one $\widetilde{\mu}_{e,i}$ as matched; similarly, each job a has at most one worker assigned in $\widetilde{\mu}_{e,i}$ because $\widehat{\mu}_e$ assigns at most one worker to the specific copy (a, i).

Two remarks clarify why this helps.

First, internal stability is preserved by projection. Any ϵ-blocking deviation under internal stability must involve two agents who are both matched in the realized matching. If $\widetilde{\mu}_{e,i}$ had an internal ϵ-blocking pair (w, a), then in the duplicated market worker w would prefer the copy (a, i) by more than ϵ, and job copy (a, i) would prefer w to its assignee under the inherited priority P_a. This would contradict ϵ-stability of $\widehat{\mu}_e$. Thus, stability of $\widehat{\mu}_e$ is a convenient sufficient condition for each projected schedule slot $\widetilde{\mu}_{e,i}$ to be implementable without internal blocking.

Second, the copy indices create "coverage" over workers. The duplication factor m is chosen so that, in a worst case over preferences and priorities, one can prove that for every worker w there exists some index $i(w)$ for which $\widetilde{\mu}_{e,i(w)}$ gives w a job attaining their per-round optimal ϵ-stable share (relative to the original, unduplicated market). The economic intuition is congestion: if too many workers were denied their best stable opportunities across all m indices, then tracing the implied priority conflicts generates an expanding set of distinct workers, and an exponential-growth argument forces m to scale as $\Theta(\log |W^e|)$. ROSS takes m large enough to dominate $|W^e|$ uniformly over epochs via $N_{\max}$, so that we can use a single rotation length throughout.

Computing $\widehat{\mu}_e$: an oracle view. Algorithmically, ROSS requires a subroutine that outputs an ϵ-stable matching in the duplicated market. We can treat this as an oracle $\textsc{StableMatch}_\epsilon(W^e, A^{e,\times}, U^{e,\times}, P)$. When utilities induce weak worker preferences (ties) and $\epsilon > 0$ creates additional indifference, any standard stable-matching procedure can be adapted by using ϵ-comparisons: a worker regards (a, i) as strictly better than (b, j) only if $U^e(w, a) > U^e(w, b) + \epsilon$, and ties can be broken deterministically (e.g., by job identifiers) without affecting the guarantees stated later, since the analysis is worst-case over instances and does not rely on selecting a particular extremal stable matching.

From an implementability standpoint, the key point is that $\textsc{StableMatch}_\epsilon$ runs once per epoch, not per round. After it returns $\widehat{\mu}_e$, the m projected matchings $\widetilde{\mu}_{e,1}, \ldots, \widetilde{\mu}_{e,m}$ can be stored as explicit maps from workers to jobs (or $\perp$).

Step 2: scheduling within the epoch via cyclic rotation. Having computed $\{\widetilde{\mu}_{e,i}\}_{i=1}^m$, ROSS implements a deterministic rotation over the epoch's rounds. For each round $t \in T_e$, define the index

$$i_t \;:=\; 1 + \big((t - t_e) \bmod m\big),$$

and set the realized matching to be

$$\mu_t \;:=\; \widetilde{\mu}_{e,i_t}.$$

Thus, workers experience a repeating cycle of length m. This deterministic schedule is the simplest way to operationalize the intuition behind time-sharing: if the platform were allowed to randomize independently each round, it could draw i uniformly from $\{1, \ldots, m\}$ and play $\widetilde{\mu}_{e,i}$; the cyclic rule produces the same long-run frequencies without requiring per-round randomness, and it supports auditability (agents can predict which matching will be used on which day of the cycle).

A minor practical refinement is to choose a random *phase* at the epoch start: draw an offset $\phi_e \in \{0, \ldots, m-1\}$ and set $i_t = 1+((t-t_e+\phi_e) \bmod m)$. This preserves all structural properties while reducing systematic bias against agents who tend to arrive just before an epoch ends.

What churn changes, and what ROSS does (and does not) attempt to do. When a new epoch begins, the previously computed schedule may no longer be feasible, so ROSS discards it and recomputes from scratch on the new participant sets. This is deliberately conservative: we do not attempt to "stitch" schedules across epochs or maintain intertemporal commitments, because such commitments would interact sharply with priorities and could violate even internal stability in the new market. Instead, we accept that

churn can truncate a cycle, potentially preventing some workers from reaching the index $i(w)$ at which they would obtain their best stable opportunity in that epoch. Later, this shows up as an additive loss that scales with the number of epochs a worker experiences.

Computational complexity and operational footprint. Within epoch e, the duplicated market has $|W^e|$ workers and $|A^e|m$ job copies. If $\textsc{StableMatch}_\epsilon$ is implemented via a deferred-acceptance-style routine, runtime is polynomial in $|W^e| \cdot |A^e|m$, and memory is linear in the same order to store preference access and the resulting matching $\widehat{\mu}_e$. The projection step to obtain $\widetilde{\mu}_{e,1}, \ldots, \widetilde{\mu}_{e,m}$ is a single pass over workers and is negligible relative to the stable-matching computation.

Per round, the online work is trivial: compute i_t and output $\widetilde{\mu}_{e,i_t}$. In platform terms, the heavy computation occurs only when the market composition changes; the per-round matching decision is then a table lookup. This separation between *epoch-time computation* and *round-time execution* is precisely what makes ROSS plausible as a scheduling primitive in environments where stability is a hard operational constraint but fairness must be delivered through repeated interaction.

5.1 Main results: stability, dynamic optimal-share guarantees, and tightness

Our analysis of ROSS formalizes a basic design goal for repeated matching markets: we want each *realized* round-t assignment to be implementable (no credible within-assignment deviation), while simultaneously guaranteeing that over time each worker receives a meaningful fraction of what they could have obtained from the best ϵ-stable outcome available in each round. The key point is that these objectives are in tension in worst case: stability constraints restrict the set of feasible matchings, and time-variation (churn) limits the extent to which we can smooth or randomize across matchings.

Ex-post ϵ-internal stability each round. The first guarantee is qualitative but operationally central: every matching μ_t output by ROSS is ϵ-internally stable *in that round*. Internal stability is the appropriate notion for dynamic settings in which the platform must commit to an assignment among the agents who are actually matched at time t: it rules out a pair (w, a) such that (i) both would be involved in a deviation that displaces another currently matched agent and (ii) the worker gains more than ϵ from switching, while the job prefers the worker by its strict priority. Because this definition only quantifies over pairs where both sides are currently matched, it is robust to the presence of unmatched agents and to the fact that the market composition changes over time.

Formally, within an epoch e the policy plays a cyclic sequence of matchings $\{\widetilde{\mu}_{e,i}\}_{i=1}^{m}$ that were constructed to be ϵ-internally stable for that epoch's agent sets and priorities. Since the active sets (W_t, A_t) are constant inside the epoch by definition, the same internal-stability certificate applies to any round $t \in T_e$. Thus, ROSS ensures

$$\mu_t \in I_t^{\epsilon} \qquad \forall t \in \{1, \ldots, T\},$$

without requiring any on-the-fly reoptimization. Economically, this is the sense in which ROSS produces *ex-post* stable outcomes: stability is not merely guaranteed in expectation over randomization, but holds for the particular matching actually implemented on that day.

It is important to be explicit about what this does *not* claim. Internal stability is weaker than full (external) ϵ-stability, because it does not prevent a blocking pair involving an unmatched worker or an unmatched job. This asymmetry is deliberate: in many platform settings, an unmatched agent can be interpreted as not participating (or being unavailable) in the operative round, so deviations involving them are not credible. In Section 7 we discuss how one can strengthen the stability requirement (or accommodate uncertainty) at the cost of a larger ϵ or additional computation.

Dynamic OSS benchmark and the $O(\log N_{\max})$ share. To quantify fairness over time, we compare each worker's realized cumulative utility to a per-round ϵ-stable benchmark. Fix a worker w, and recall that their dynamic benchmark is

$$\mathrm{dOSS}_{\epsilon}(w) \;:=\; \sum_{t \in \mathcal{T}(w)} U_t^{\epsilon,*}(w), \qquad U_t^{\epsilon,*}(w) \;:=\; \max_{\mu \in S_t^{\epsilon}} U_t\big(w, \mu(w)\big),$$

i.e., the sum across the rounds in which w is active of the best utility they could obtain in an ϵ-stable matching computed *fresh* for that round's market. This benchmark is demanding: it allows the matching to change adversarially from round to round in a worker-favorable way, subject only to ϵ-stability in that round. We view this as an appropriate "gold standard" for an online platform that wants to provide each worker a stable share of the best stable opportunities available when they show up.

ROSS guarantees a logarithmic approximation to this benchmark, with additive losses that separate (i) the stability tolerance ϵ and (ii) churn. Let $m := \lceil \log_2 N_{\max} \rceil + 2$. Then for every worker w,

$$\mathbb{E}\Big[\sum_{t \in \mathcal{T}(w)} U_t(w, \mu_t(w)) \Big] \;\geq\; \frac{1}{m} \sum_{t \in \mathcal{T}(w)} U_t^{\epsilon,*}(w) \;-\; \epsilon\,|\mathcal{T}(w)| \;-\; E(w).$$

The expectation is with respect to any random phase choice at epoch boundaries (or, equivalently, to a randomized implementation that samples uniformly from $\{\widetilde{\mu}_{e,i}\}$ each round within an epoch). The deterministic cyclic

schedule achieves the same long-run frequencies as uniform randomization; the phase randomization simply avoids systematic alignment between arrival times and the cycle index.

The inequality has a transparent economic interpretation. The first term says that each worker receives at least a $1/m$ fraction of their dynamic stable benchmark, where m grows only logarithmically with the maximum market size. The second term, $-\epsilon|\mathcal{T}(w)|$, reflects the fact that allowing ϵ-blocking slack effectively relaxes comparisons by ϵ each round; we pay for that relaxation in the performance bound. The third term, $-E(w)$, is a churn penalty: each epoch boundary can truncate a rotation and cause the worker to miss (at most) one scheduled "good" index at which they would have attained their best ϵ-stable opportunity for that epoch.

Why a logarithmic factor, and why churn enters additively. The multiplicative $\log N_{\max}$ dependence is not an artifact of our proof technique; it reflects a real congestion phenomenon inherent to stability under priorities. Intuitively, workers' best stable opportunities are mutually incompatible: many workers may regard the same small set of jobs as their best stable outcomes, but job priorities restrict which of them can be served without creating blocking incentives. The duplication–index argument underlying the epoch construction shows that, by expanding each job into m indexed "time slots," we can distribute these best stable outcomes across indices so that each worker attains their own best stable outcome on at least one index. A directed-growth (or forest) argument then implies that ensuring such coverage for *all* workers in the worst case forces m to scale as $\Theta(\log |W^e|)$; choosing m based on $N_{\max}$ ensures a uniform guarantee across epochs.

Churn matters differently: it does not change the intrinsic time-sharing requirement (hence it does not appear inside the logarithm), but it limits how effectively the platform can *realize* the within-epoch coverage over time. When an epoch ends, the schedule must be recomputed because feasibility and stability constraints change with the participant sets. This recomputation resets the rotation, so a worker who is present across many short epochs may repeatedly lose the "tail" of the cycle that would have delivered their favorable index. Bounding utilities by 1 implies that each such truncation costs at most one unit of cumulative utility, yielding an additive penalty proportional to $E(w)$. In practice, this term emphasizes a policy tradeoff: recomputing aggressively keeps matchings aligned with the current market, but frequent resets reduce the ability to deliver time-averaged fairness.

Tightness and limits of improvement. Finally, the guarantee is essentially the best possible in worst case. Even in the static setting (a single epoch with fixed agents and utilities), no policy that selects matchings—even allowing arbitrary mixing over stable matchings—can guarantee every worker a

share better than $\Omega(1/\log N)$ of their optimal stable share in the worst case. By embedding that hard static instance as an epoch in our dynamic environment, we inherit the same lower bound: the $1/m = \Theta(1/\log N_{\max})$ multiplicative dependence is order-tight up to constants. Thus, improvements over ROSS must come from moving beyond worst-case guarantees (e.g., structural assumptions on utilities or priorities), from weakening stability, or from allowing richer intertemporal instruments than per-round matchings.

Taken together, these results characterize the tradeoff the model is designed to illuminate. ROSS delivers ex-post implementability (internal stability) every round, and it converts repeated interaction into time-averaged fairness with only a logarithmic loss in market size. The unavoidable costs appear in intuitive places: a per-round ϵ slack term, and an additive penalty for churn that captures the fundamental difficulty of guaranteeing time-sharing when the set of participants itself is unstable.

6 Extensions: robustness, uncertainty, and recourse

The baseline presentation treats utilities U_t as observed and piecewise constant within an epoch, so that we may precompute an internally stable rotation and then implement it mechanically. In many applications, however, the utility inputs are themselves moving targets (e.g., worker-specific conversion rates, predicted completion times), or they are only estimated with statistical error. Moreover, platforms often face *recourse* constraints: operational, contractual, or fairness considerations limit how sharply assignments can change from round to round. We outline three extensions that preserve the organizing logic of ROSS—time-sharing across a small support of internally stable matchings—while making explicit the additional slack one must pay for robustness or implementability.

(i) Bounded utility drift within an epoch and ϵ-robust internal stability. Suppose that within an epoch e the participant sets (W^e, A^e) and priorities P_a are fixed, but utilities drift gradually over rounds $t \in T_e$. A convenient model is a max-norm Lipschitz condition: for consecutive $t, t{+}1 \in T_e$,

$$\|U_{t+1} - U_t\|_{\max} \;:=\; \max_{w \in W^e,\; a \in A^e} \big|U_{t+1}(w,a) - U_t(w,a)\big| \;\le\; \rho.$$

Operationally, the platform computes the rotation $\{\widetilde{\mu}_{e,i}\}_{i=1}^{m}$ once at the epoch start $t_e := \min T_e$, using U_{t_e}, and then plays it throughout the epoch. The question is what internal-stability guarantee this implements at a later round t, when the true utilities are $U_t \neq U_{t_e}$.

The key observation is that internal stability is a *margin* condition: a blocking pair (w, a) requires a strict improvement for worker w of more than

ϵ relative to their assigned job. If each entry of the utility matrix moves by at most Δ, then the improvement margin moves by at most 2Δ (one term for the deviation utility and one for the incumbent utility). Since $\|U_t - U_{t_e}\|_{\max} \le (t - t_e)\rho$, it follows that if $\widetilde{\mu}_{e,i}$ is ϵ_0-internally stable under U_{t_e}, then the same matching is ϵ-internally stable under U_t for any

$$\epsilon \;\ge\; \epsilon_0 + 2(t - t_e)\rho.$$

A simple and often useful corollary is a within-epoch calibration rule: if we target a fixed tolerance ϵ at all rounds, we may compute the rotation at epoch start with a smaller tolerance ϵ_0 that leaves room for drift. In the common case where drift is small on the time scale of an epoch, choosing ϵ on the order of 2ρ (or, more conservatively, $2\rho L_e$) preserves ex-post ϵ-internal stability without recomputation.

This robustness comes with two economic limitations. First, increasing ϵ widens the set of matchings that qualify as stable but simultaneously weakens the benchmark comparison by introducing larger $-\epsilon|\mathcal{T}(w)|$-type terms in performance bounds. Second, if drift is not small relative to the desired stability margin, then the platform faces a design tradeoff between *reoptimization frequency* (shorter epochs, hence more churn-like resets) and *stability slack* (larger ϵ). A practical compromise is a trigger rule: retain the current rotation as long as an online estimate of $\|U_t - U_{t_e}\|_{\max}$ stays below $\epsilon/2$, and otherwise declare an epoch break and recompute. This converts unmodeled drift into an endogenous churn measure, making explicit that robustness and churn are two sides of the same implementability constraint.

(ii) Uncertain utilities and rectangular confidence sets: robust ϵ-calibration. A second departure from the baseline is informational: the platform may not observe U_t directly, but only an estimate $\widehat{U}_t$ derived from historical data or noisy feedback. A parsimonious and analytically tractable uncertainty model is *rectangular* (entrywise independent) confidence sets. For each round t, let

$$\mathcal{U}_t \;:=\; \Big\{U :\, W_t \times A_t \to [0,1] \;\Big|\; U(w,a) \in [\widehat{U}_t(w,a) - \delta_t(w,a),\; \widehat{U}_t(w,a) + \delta_t(w,a)] \;\; \forall (w,a)\Big\},$$

with truncation at $[0,1]$ implicit. The platform's objective may then be to ensure implementability and fairness *uniformly* over all $U \in \mathcal{U}_t$, reflecting worst-case misspecification within the confidence region.

There are (at least) two natural robust stability notions. The more conservative one requires that the realized matching μ_t be ϵ-internally stable for *every* $U \in \mathcal{U}_t$. This is a direct robustness constraint: no deviation should become profitable by more than ϵ under any plausible utility realization. A less conservative alternative requires stability only for the (unknown) true U_t, but uses $\mathcal{U}_t$ to pick an ϵ that makes violations unlikely. Our framework

accommodates either choice through the same calibration logic: the stability margin must dominate the uncertainty in pairwise utility differences.

Concretely, define the round-t worst-case uncertainty radius

$$\Delta_t \;:=\; \max_{w\in W_t,\; a\in A_t} \delta_t(w,a).$$

If a matching μ is ϵ_0-internally stable under the estimate $\widehat{U}_t$, then for any $U \in \mathcal{U}_t$ the worker's deviation gain satisfies

$$U(w,a) - U(w,\mu(w)) \;\le\; \widehat{U}_t(w,a) - \widehat{U}_t(w,\mu(w)) \;+\; 2\Delta_t,$$

so μ is ϵ-internally stable uniformly over $\mathcal{U}_t$ whenever $\epsilon \ge \epsilon_0 + 2\Delta_t$. Thus, rectangular uncertainty enters exactly as an *additive* slack in ϵ, mirroring the bounded-drift logic but with Δ_t interpreted as statistical error rather than temporal movement.

For fairness comparisons, one must also decide what benchmark is meaningful under uncertainty. A pessimistic benchmark replaces $U_t^{\epsilon,*}(w)$ with a *robust* optimal stable share,

$$U_{t,\mathrm{rob}}^{\epsilon,*}(w) \;:=\; \max_{\mu\in S_t^\epsilon}\; \min_{U\in\mathcal{U}_t} U\big(w,\mu(w)\big),$$

which asks what a worker can guarantee in an ϵ-stable matching when utilities are adversarially selected within $\mathcal{U}_t$. This benchmark is weaker than the nominal one computed from $\widehat{U}_t$, but it aligns with environments where the platform is accountable for performance under misspecification (e.g., when predicted worker-job fit is systematically biased). Alternatively, if $\mathcal{U}_t$ is a high-probability confidence set, the platform can benchmark against the nominal $\widehat{U}_t^{\epsilon,*}(w)$ and interpret the resulting guarantees as holding with high probability, with Δ_t governing the necessary ϵ-buffer.

The limitation here is conceptual as well as quantitative: robustifying stability via larger ϵ can be economically meaningful (it prevents fragile assignments that hinge on thin predicted differences), but it may also legitimize outcomes that are materially blocked under the true utilities. This makes the calibration step a policy parameter: platforms may wish to report the chosen ϵ (or its implied Δ_t) as a transparency measure, since it encodes how conservative the assignment is relative to estimation risk.

(iii) Optional recourse constraints and when numerical methods become necessary. A third extension concerns operational feasibility. ROSS as stated rotates among m internally stable matchings, and this rotation may reassign many workers from one round to the next. In settings such as school placement across terms, shift bidding with training requirements, or gig platforms that value continuity, the platform may face hard or soft

recourse constraints that penalize reassignment. A canonical hard constraint is a per-round cap

$$\left|\{w \in W_t : \ \mu_t(w) \neq \mu_{t-1}(w)\}\right| \ \leq \ R,$$

or, more generally, a switching cost added to the objective:

$$\sum_{t=2}^{T} \sum_{w \in W_t \cap W_{t-1}} c_w \cdot \mathbf{1}\{\mu_t(w) \neq \mu_{t-1}(w)\}.$$

Introducing such constraints changes the nature of the design problem. Without recourse constraints, we can analyze each epoch independently, and the principal difficulty is distributing each worker's best stable outcome across a small index set. With recourse constraints, we must additionally construct a *path* through the space of stable matchings whose adjacent elements are close.

Two approaches are natural. The first is structural: restrict attention to rotations that change only a small set of assignments per step (for instance, by decomposing the desired lottery over matchings into a sequence connected by local exchanges). This resembles designing a mixing path on the graph of ϵ-internally stable matchings, with edge weights capturing recourse costs. The second is computational: formulate the within-epoch schedule as an optimization problem over matchings and time indices, balancing (i) internal stability constraints $\mu_t \in I_t^\epsilon$ and (ii) coverage constraints that deliver each worker a target fraction of $U_t^{\epsilon,*}(w)$, subject to explicit recourse limits. Even for a single epoch with fixed participants, this joint design problem can become a mixed-integer program, because stability constraints are combinatorial and the schedule couples decisions across rounds.

We view this as an appropriate point to acknowledge a boundary of the analytic guarantee: the clean $\Theta(\log N_{\max})$ factor is driven by worst-case coverage under stability, but once recourse is imposed, additional lower bounds may emerge from the geometry of the stable set itself (e.g., if stable matchings are disconnected under small Hamming moves). In practice, one can still use the ROSS construction as a *starting point*—it provides a small support of candidate stable matchings with good per-worker coverage—and then solve a secondary routing problem that orders these matchings to minimize reassignment, possibly repeating matchings for multiple rounds when the recourse budget is tight. This hybrid approach preserves the economic interpretation of ROSS (time-sharing across stable outcomes) while adapting it to environments where continuity is a first-class constraint rather than an incidental preference.

7 Lower bounds and the necessity of churn penalties

Our guarantees combine two qualitatively different ideas: a *congestion* term (the unavoidable $O(\log N_{\max})$ multiplicative loss, even in a perfectly static environment) and an *interruption* term (the additive dependence on how often the market composition changes around a worker). It is useful to separate these forces, because they speak to different design levers. Congestion is structural—it is present even with full information, no recourse limits, and a single epoch—whereas interruption losses arise from the simple fact that time-sharing requires time.

(a) The static $\Omega(\log N)$ barrier is inherited by the dynamic model. Even if we set churn to zero (a single epoch, fixed utilities, fixed priorities), a policy that is required to output (approximately) stable matchings cannot simultaneously give every worker a constant fraction of their per-worker optimal stable utility. Proposition 5 formalizes this by embedding the hard static instance directly as a one-epoch dynamic instance. Economically, this lower bound reflects a familiar "bottleneck" phenomenon: stability pins down who can displace whom, and in worst-case instances the workers who are "locally" entitled (by priority) to high-value jobs form an exponentially branching structure. Any lottery over stable matchings must spread its probability mass across many incompatible local entitlements, and a $\log N$ loss is the price of distributing these entitlements broadly enough.

This point matters for interpretation. Without the $\Omega(\log N)$ obstruction, one might read the $m = \lceil \log_2 N_{\max} \rceil + 2$ rotation length as an artifact of our construction. Proposition 5 says it is not: even with unlimited time to rotate and no arrivals or departures, $\Theta(\log N)$ is the right order in worst case when benchmarking against per-worker OSS.

(b) Why any churn-free guarantee must fail under epoch interruptions. The additive term $E(w)$ in Proposition 3 is not merely a byproduct of the proof technique; it captures a genuine impossibility created by short epochs. The core tension is that the fairness logic of ROSS is fundamentally a *coverage* statement: within a fixed instance, we identify a small set of internally stable matchings and ensure that across a full cycle each worker is covered (i.e., receives a matching that attains their per-round stable benchmark up to the $1/m$ factor). When an epoch ends early, the cycle may be cut before that worker's "covering" index is played.

We can make this precise via a simple scheduling argument that is agnostic to how the matchings are constructed.

Interruption lower bound (informal). Fix a participant set (W, A), priorities P, and utilities U. Suppose there exists a collection of internally

stable matchings $\{\nu_1, \ldots, \nu_m\} \subseteq I^\epsilon$ and a map $i(\cdot)$ such that each worker w achieves their per-round benchmark in at least one of them, i.e.,

$$U(w, \nu_{i(w)}(w)) \;\geq\; U^{\epsilon,*}(w).$$

Any policy that plays one matching per round can realize this coverage for all workers only if it has at least m rounds in which to deploy (a permutation of) these matchings. If the epoch length is $L < m$, then regardless of the policy, there exist workers whose covering index is not played. For such a worker w, one can have

$$\sum_{t=1}^{L} U\big(w, \mu_t(w)\big) \;=\; 0 \qquad \text{while} \qquad \sum_{t=1}^{L} U^{\epsilon,*}(w) \;=\; L,$$

simply by designing utilities so that w obtains value 1 only in their covering matching and value 0 in all other stable matchings the policy might play.

The point is not that the worker gets literally zero in realistic markets, but that *the gap scales with the number of interrupted opportunities to complete a coverage cycle*. If we concatenate many short epochs, the losses compound. For example, consider a worker w who is present for $E(w)$ epochs of length one. If in each epoch the platform is forced (by stability) to choose one matching from a set in which w is "served" only by a particular index, then w may miss that index in every epoch. In that case, the worker's cumulative benchmark is $\sum_{t \in \mathcal{T}(w)} U_t^{\epsilon,*}(w) = E(w)$, but the realized utility can remain 0. Any guarantee of the form

$$\mathbb{E}[\Pi_w] \;\geq\; \frac{1}{m}\,\mathrm{dOSS}_\epsilon(w) \;-\; o(E(w))$$

must therefore fail on such a path: the right-hand side grows linearly in $E(w)$ while the left-hand side need not. This is exactly why Proposition 3 pays an additive term proportional to $E(w)$: each time an epoch ends, the platform may lose (for some workers) up to one unit of utility relative to what a full rotation would have delivered, and there is no way to amortize this loss if the environment repeatedly denies the platform enough rounds to complete a cycle.

Two further remarks sharpen the interpretation.

First, the lower bound does *not* rely on the platform being ignorant of the epoch length. Even if the platform knows an epoch will last $L < m$ rounds, it cannot generally order L stable matchings so as to cover *all* workers' benchmark-achieving indices: the coverage requirement itself demands at least m distinct "slots" in the worst case. Knowledge can help choose *which* workers to prioritize early in the cycle, but it cannot remove the combinatorial scarcity of stable opportunities.

Second, the dependence on $E(w)$ is inherently worker-specific. Some workers may experience low effective churn because they remain present in

long epochs; others (e.g., occasional gig workers) may appear only in short bursts. A uniform global churn measure C can be a useful summary, but the impossibility is pinned to how often the *individual* worker's participation is fragmented.

(c) When stability restrictions become tighter, barriers can strengthen to $\Omega(N)$. The $\Omega(\log N)$ barrier concerns the best possible guarantee when we are allowed to randomize (or equivalently time-share) over stable outcomes. If we impose additional restrictions that shrink the feasible set of matchings or the feasible set of lotteries, much stronger impossibilities can emerge. One salient case is insisting on exact stability with no slack (or, more generally, restricting attention to a thin subset of stable matchings with special structure). In such environments, it is possible to construct instances where different workers' optimal stable assignments occur in essentially disjoint stable matchings, so that any lottery supported on fewer than $\Omega(N)$ stable matchings leaves some worker with vanishing probability of receiving their OSS-attaining assignment. In the extreme, one can have a family of instances where for each w there exists a stable matching μ^w that gives w utility 1 but gives many other workers utility 0, and no single stable matching can simultaneously deliver utility 1 to more than $O(1)$ workers. Then, for any distribution $\mathcal{D}$ over stable matchings,

$$\min_{w} \ \mathbb{E}_{\mu\sim\mathcal{D}}\big[U(w,\mu(w))\big] \ \le \ O(1/N),$$

yielding an $\Omega(N)$ multiplicative barrier relative to the per-worker OSS benchmark (which equals 1 for all workers in this construction). The economic message is straightforward: if the stable set fragments into highly specialized outcomes, then fairness-by-time-sharing requires a large support, and short epochs make that requirement operationally infeasible.

This is one reason we regard ϵ not as a purely technical relaxation, but as an implementability parameter. Allowing $\epsilon > 0$ can "thicken" the feasible set I^ϵ enough to recover small-support coverage (and thus logarithmic rotation length), whereas insisting on razor-thin exact stability can force either very long rotations or very unequal treatment.

Takeaway. Putting these pieces together, we should read Proposition 3 as a sharp decomposition of what can and cannot be improved. The $\Theta(\log N_{\max})$ factor is the irreducible congestion cost inherited from the static model. The additive churn term is the irreducible interruption cost that appears whenever participation is fragmented into many short epochs. The platform can trade between the two only by changing modeling commitments: permitting slack in stability (ϵ), recomputing more often (thereby increasing effective churn), or changing the benchmark (e.g., adopting a recourse-aware or robust benchmark). In this sense, churn is not merely a nuisance parameter;

it is the dynamic analogue of market thickness, and any fairness guarantee that ignores it must fail in worst case.

8 Empirical protocol (optional): estimating dynamic OSS proxies, simulation design, and fairness diagnostics

The theory above is benchmarked to the (unobserved) dynamic OSS quantity $\mathrm{dOSS}_\epsilon(w) = \sum_{t \in \mathcal{T}(w)} U_t^{\epsilon,*}(w)$. In applied settings, a platform rarely observes U_t directly; instead it observes a log of feasible pairs, recommended/assigned matches, acceptance/completion outcomes, payments, and worker/job attributes. This section outlines a pragmatic protocol for (i) constructing *proxies* for $\mathrm{dOSS}_\epsilon(w)$ from logs, (ii) designing simulations that mirror gig/task marketplaces, and (iii) reporting interpretable metrics for ϵ-stability violations (envy incidents) and share fairness.

Step 0: reconstruct the per-round instances from logs. For each round t, we require an empirical analogue of (W_t, A_t, P, U_t). The participant sets (W_t, A_t) are typically observed (active workers online; open tasks). Priorities P_a can be (a) an explicit rule (e.g., rating tiers, seniority, response-time buckets), in which case it is directly recoverable, or (b) an implicit policy (e.g., a learned ranking score), in which case we recommend fixing P_a to the *auditable* ordering the platform claims to use for compliance. The main modeling choice is utilities: we recommend estimating a bounded cardinal score $\widehat{U}_t(w, a) \in [0, 1]$ that predicts realized worker surplus (or a monotone transform of acceptance probability) as a function of observables (wage, distance, duration, skill match, time-of-day, etc.). Concretely, one can set

$$\widehat{U}_t(w, a) \;:=\; \sigma\Big(\beta^\top x_t(w, a)\Big)\,,$$

where σ is a squashing map to $[0, 1]$ (logit/probit or a calibrated isotonic regression), and $x_t(w, a)$ are logged features. When only bandit feedback is available (utilities observed only for served pairs), we recommend counterfactual modeling with inverse-propensity weighting or doubly robust estimation; the output remains a proxy, and this limitation should be reported explicitly.

Step 1: partition time into epochs and compute churn summaries. Given (W_t, A_t), we compute epochs T_e as maximal contiguous intervals with constant participant sets. Empirically, one may also use *approximate* epochs by thresholding the symmetric difference $|W_t \Delta W_{t-1}| + |A_t \Delta A_{t-1}|$ to avoid treating minor fluctuations as full recomputations. For each worker w, we compute $E(w)$ (the number of epochs in which w appears) and $|\mathcal{T}(w)|$. These quantities are not merely descriptive: they determine, in the theory, the scale

at which interruption losses can appear, so we recommend reporting them alongside fairness outcomes (e.g., stratifying by high- vs. low-churn workers).

Step 2: estimate a per-round OSS proxy $\widehat{U}_t^{\epsilon,*}(w)$. Exact computation of $U_t^{\epsilon,*}(w) = \max_{\mu \in S_t^\epsilon} U_t(w, \mu(w))$ may be intractable at scale, especially under ties and ϵ-slack. We therefore recommend two complementary proxies.

(A) Optimization-based proxy (small/medium instances). Fix t and a worker w. Solve a mixed-integer program that maximizes $\widehat{U}_t(w, a)$ over matchings subject to feasibility and ϵ-internal stability constraints induced by P_a. Using binary variables $x_{ua} \in \{0, 1\}$ and outside option $x_{u\perp}$, feasibility is linear. ϵ-internal stability can be enforced by forbidding internally blocking pairs: for any a and any pair of workers $u \succ_a v$, we rule out allocations in which a is assigned to v while u is assigned to some b that makes u ϵ-prefer a. With discretization (or by precomputing preference relations $\widehat{U}_t(u, a) > \widehat{U}_t(u, b) + \epsilon$), these constraints can be written as linear inequalities of the form

$$x_{va} + \sum_{b:\ \widehat{U}_t(u,a) > \widehat{U}_t(u,b)+\epsilon} x_{ub} \ \leq \ 1.$$

The objective pins down the best stable job for w, yielding $\widehat{U}_t^{\epsilon,*}(w)$. This proxy is computationally heavy but has the advantage of being explicit and auditable.

(B) Sampling-based proxy (large instances). Generate a diverse collection of ϵ-internally stable matchings for round t (or for epoch e) by running a stable-matching routine under random perturbations/tie-breaks. For example: add i.i.d. noise η_{ua} to utilities, compute a worker-proposing deferred acceptance outcome with workers ranked by $\widehat{U}_t(u, a) + \eta_{ua}$ and jobs ranked by P_a, then post-check ϵ-internal stability under $\widehat{U}_t$ and keep only feasible matchings. Repeat R times, obtaining $\{\mu^{(r)}\}_{r=1}^R$, and define

$$\widehat{U}_t^{\epsilon,*}(w) \ := \ \max_{r \leq R} \ \widehat{U}_t\big(w, \mu^{(r)}(w)\big).$$

This estimator is a *lower bound* on the true $\widehat{U}_t^{\epsilon,*}(w)$ (it can miss the best stable outcome), but it scales and aligns with the time-sharing logic: it directly produces a small support of stable matchings that can also serve as candidate schedules.

Step 3: compute dynamic OSS and realized-share diagnostics. Given a policy's realized assignments μ_t and utilities $\widehat{U}_t$, compute each worker's realized cumulative utility

$$\widehat{\Pi}_w \ := \ \sum_{t \in \mathcal{T}(w)} \widehat{U}_t\big(w, \mu_t(w)\big), \qquad \widehat{\mathrm{dOSS}}_\epsilon(w) \ := \ \sum_{t \in \mathcal{T}(w)} \widehat{U}_t^{\epsilon,*}(w).$$

We then report a *share* statistic,

$$\widehat{\text{Share}}(w) \;:=\; \frac{\widehat{\Pi}_w}{\widehat{\text{dOSS}}_\epsilon(w) + 10^{-9}},$$

along with its distribution across workers (minimum, 10th percentile, median). Because $\widehat{\text{dOSS}}_\epsilon(w)$ is itself estimated, we recommend sensitivity analyses over (i) the number of samples R in the sampling proxy, (ii) alternative utility models $\widehat{U}_t$, and (iii) alternative ϵ values (especially if ϵ is interpreted as a compliance tolerance).

Step 4: measure envy incidents and stability violations. To connect to the stability constraints, we recommend logging and reporting *blocking-pair* statistics. For any round t, define the ϵ-blocking gain of pair (w, a) as

$$\Delta_t^\epsilon(w, a) \;:=\; \widehat{U}_t(w, a) - \widehat{U}_t\big(w, \mu_t(w)\big) - \epsilon.$$

An *external* ϵ-envy incident occurs if $\Delta_t^\epsilon(w, a) > 0$ and $w \succ_a \mu_t(a)$ (treating $\mu_t(a) = \perp$ as lowest priority). An *internal* ϵ-envy incident additionally requires that both w and a are matched in μ_t, matching our internal-stability constraint. Report:

1. the rate of incidents, e.g. $\frac{1}{\sum_t |W_t||A_t|} \sum_{t,w,a} \mathbf{1}\{\Delta_t^\epsilon(w, a) > 0 \wedge w \succ_a \mu_t(a)\}$;
2. an intensity metric, $\sum_{t,w,a} \big(\Delta_t^\epsilon(w, a)\big)_+$, which distinguishes many tiny violations from a few large ones;
3. a per-worker exposure metric, $\sum_{t,a} \mathbf{1}\{w \text{ is } \epsilon\text{-envied at } t \text{ via } a\}$, to detect systematic unfairness concentrated on specific cohorts.

These diagnostics translate the abstract notion of stability into audit-friendly quantities: *how often* could a worker plausibly claim a justified displacement under the stated priority rules, and *how large* are the gains at stake.

Step 5: simulation design for gig/task markets. For controlled evaluation, we recommend simulating a sequence of rounds with (i) stochastic arrivals/departures that generate epochs, (ii) structured utilities, and (iii) explicit priorities. A minimal design is:

1. Arrivals: each worker appears according to an on/off Markov process (capturing shift-like availability); tasks arrive as a Poisson process with service times and deadlines.
2. Utilities: let $\widehat{U}_t(w, a)$ be increasing in pay and decreasing in distance and expected duration, with heterogeneity in worker-specific weights; optionally add drift ρ by evolving task characteristics or worker location.

3. Priorities: set P_a by a composite score (e.g. rating tier first, then response speed), fixed across time to mimic a compliance rule.

Compare ROSS-style rotations to baselines such as (a) myopic stable matching each round (e.g. worker-optimal stable matching under $\widehat{U}_t$), (b) max-weight matching ignoring stability, and (c) stability-constrained max-weight matching. Report tradeoffs among (i) share fairness $\widehat{\mathrm{Share}}(w)$, (ii) incident rates, and (iii) churn exposure $E(w)$, since the theory predicts that short epochs can dominate the welfare loss even when congestion is mild.

Limitations and interpretation. Any empirical $\widehat{\mathrm{dOSS}}_\epsilon(w)$ is a model-based construct; it should be interpreted as an *auditable counterfactual* under declared priorities and an estimated utility model, not as a ground-truth entitlement. Nevertheless, the protocol is useful precisely because it decomposes performance into (i) a stability/compliance dimension (envy incidents) and (ii) a fairness-by-time-sharing dimension (realized share relative to an ϵ-stable benchmark), which can then be linked to concrete operational levers such as recomputation frequency (epoch length) and the permissible slack ϵ.

9 Policy and platform implications: rotating schedules, auditability, and stability as a compliance constraint

The central practical lesson of the model is that *time-sharing over stable matchings* can serve as an operationally simple bridge between two objectives that are often in tension in platform design: (i) respecting a declared priority rule (for compliance, legitimacy, or contractual reasons), and (ii) delivering a meaningful notion of long-run fairness when the market is congested and participants churn. ROSS makes this bridge explicit. Rather than interpreting stability as a one-shot outcome concept ("compute a stable matching and stop"), we treat it as a *per-round constraint*—each realized matching must be (internally) stable up to slack ϵ—and then use rotations across a small set of such matchings to share scarce "good" opportunities over time.

Rotating schedules as a design primitive. Many platforms already rotate exposure implicitly (e.g., cycling who is shown first in a ranking, or who is offered the next job). Our framework suggests making this rotation *structured* and *verifiable*. Within an epoch e (a period of stable participation sets), ROSS precomputes a collection $\{\widetilde{\mu}_{e,1}, \ldots, \widetilde{\mu}_{e,m}\}$ of ϵ-internally stable matchings and then cycles through them. The theoretical point of the construction is not that any particular $\widetilde{\mu}_{e,i}$ is optimal, but that the *support size*

$m = \lceil \log_2 N_{\max} \rceil + 2$ is small while still guaranteeing each worker a nontrivial fraction of her per-round best ϵ-stable outcome in expectation.

From an engineering perspective, this "small support" feature matters: platforms can store and reason about a small menu of matchings per epoch (or per recomputation window), rather than relying on a continuously changing, hard-to-audit ranking model. Moreover, deterministic cycling can be implemented without continuous randomness, which can simplify incident investigations (one can trace which index i was scheduled at time t) while still approximating the fairness benefits of random time-sharing.

Auditability and explainability: why stability helps. A recurring challenge in applied matching systems is that stakeholders want an answer to the question: *"Why was I skipped?"* Stability provides a crisp and auditable response when priorities P_a are part of the platform's public commitments. If the platform asserts that jobs follow a strict priority order P_a (e.g., seniority, licensing tier, or a legally protected ordering), then ϵ-internal stability implies there is no pair (w, a) of *currently matched* agents for which (i) job a would rank w above its assigned worker and (ii) w would gain more than ϵ in utility by switching to a. In other words, among those currently being served, there is no justified displacement that simultaneously respects the job's declared priority and yields a material gain to the worker.

This is a compliance-oriented notion. It does not claim global optimality, nor does it preclude envy by unmatched agents (which would be ruled out by external stability). But it is often the relevant legal or contractual constraint: disputes frequently involve two served parties ("this task should have gone to me instead of them"), and internal stability directly addresses that class of claims.

ROSS also encourages a *separation of concerns* in governance. Priorities P_a can be treated as a policy object chosen for transparency and compliance; within that policy, the platform uses rotation to allocate opportunities fairly over time. This separation makes it easier to audit changes: if priorities change, that is an explicit policy revision; if only the rotation phase changes, that is an operational adjustment without altering the underlying entitlement structure.

Stability as a constraint, not an objective. A practical temptation is to treat stability as a performance target and attempt to "optimize stability" directly (e.g., minimize the number of blocking pairs). Our results instead motivate stability as a *hard constraint* (up to tolerance ϵ) and fairness as the design objective subject to that constraint. This framing matches how many platforms operate: the platform cannot violate certain priority rules, but it has discretion in how to schedule feasible stable allocations over time.

This constraint-based view also clarifies what the fairness guarantee is

and is not. The benchmark $\sum_{t\in\mathcal{T}(w)} U_t^{\epsilon,*}(w)$ is already conditioned on the stability rule; it is *not* a claim about the best unconstrained matchings. Consequently, when a worker receives a low share, the diagnosis is not necessarily "the algorithm is unfair," but may be "the declared priorities and the market thickness together imply that no stable policy can do much better." This distinction is valuable for policy discussions because it ties fairness outcomes to explicit institutional choices.

Choosing epoch length: recompute less often than you think. The dynamic guarantee highlights a concrete tradeoff in recomputation frequency. On the one hand, frequent recomputation adapts quickly to changing supply/demand and changing estimated utilities. On the other hand, each epoch boundary creates an interruption cost that appears as an additive term $E(w)$ in the worker's guarantee:

$$\mathbb{E}\Big[\sum_{t\in\mathcal{T}(w)} U_t(w,\mu_t(w))\Big] \;\gtrsim\; \frac{1}{m}\sum_{t\in\mathcal{T}(w)} U_t^{\epsilon,*}(w) \;-\; \epsilon|\mathcal{T}(w)| \;-\; E(w).$$

Operationally, this suggests a simple rule-of-thumb: *choose recomputation windows long enough to complete rotations*. Since a full cycle has length $m = O(\log N_{\max})$, epochs shorter than m rounds systematically prevent some workers from reaching their "good" scheduled index before the instance resets. When the platform must recompute frequently (e.g., because supply/demand changes quickly), it can still mitigate the churn penalty by (i) using approximate epochs that ignore small participant fluctuations, (ii) carrying over the rotation phase when changes are minor, or (iii) maintaining a stable "backbone" set of participants for the purpose of rotation while handling marginal arrivals via an auxiliary mechanism.

This perspective also changes how we interpret product requirements such as "real-time matching." Real-time does not necessarily require recomputing the entire stable allocation each round; it often suffices to recompute the *menu* of stable matchings less frequently while assigning in real time according to the current rotation index.

Choosing ϵ: from a mathematical slack to a policy tolerance. The parameter ϵ plays two roles. Mathematically, it enlarges the feasible set of matchings (making stability robust to estimation error or drift) while introducing an additive fairness loss $\epsilon|\mathcal{T}(w)|$. Policy-wise, it can be interpreted as a *tolerance for justified envy*: a worker should not be able to claim a displacement that improves her payoff by more than ϵ while respecting the job's priority.

This suggests choosing ϵ neither as zero by default nor as an arbitrary constant, but rather by anchoring it to measurement and volatility. If utilities are estimated with error on the order of δ (in max norm), then setting ϵ

modestly above that error scale can prevent spurious instability findings that are artifacts of noise. Likewise, if utilities drift within an epoch at rate ρ, the robustness logic motivates ϵ proportional to ρ. The platform can then report ϵ transparently as a compliance tolerance: "we guarantee that no matched worker can be displaced by a higher-priority worker for a gain exceeding ϵ."

At the same time, the bound cautions against setting ϵ too large: a large slack can make stability easier to satisfy, but it effectively licenses larger per-round envy gains and weakens the fairness guarantee linearly in time. In practice, this tradeoff can be managed by monitoring empirical ϵ-envy incident rates as a function of ϵ and selecting the smallest ϵ that yields acceptable robustness (e.g., stable outcomes do not oscillate wildly due to minor estimation changes).

Limitations and stakeholder risks. Two limitations are worth emphasizing in policy discussions. First, internal stability may be insufficient when stakeholders care about claims by *unmatched* agents (e.g., a worker argues she should have been matched at all). Addressing such concerns requires external stability, which is stronger and may reduce feasibility or welfare. Second, rotating schedules can create strategic timing incentives if workers can anticipate the cycle (e.g., logging in only during favorable indices). This is not a failure of the concept, but it means that platforms should consider randomizing the phase, committing to schedules in advance, or designing availability requirements that blunt timing arbitrage.

Overall, the model illuminates a practical tradeoff: stability can make matching systems more legitimate and auditable, but it constrains one-shot efficiency; rotation can restore a long-run fairness notion, but only if recomputation is not so frequent that churn prevents cycles from completing.

10 Conclusion and open problems: decentralization, multi-stakeholder objectives, and learning under churn

We view the main contribution of the framework as conceptual rather than purely algorithmic: it reframes a repeated matching problem as the interaction of (i) a per-round ϵ-internal stability constraint tied to priorities and legitimacy, and (ii) an intertemporal fairness objective benchmarked against what is achievable *subject to that same constraint*. The resulting guarantee is deliberately modest—a $1/O(\log N_{\max})$ share with additive losses from ϵ and churn—because the model makes precise why more ambitious worst-case promises are structurally impossible, even without dynamics. At the same time, the guarantee is operational: it is realized by a small-support rotation over stable matchings, which is exactly the kind of primitive platforms can implement and audit.

Several open problems follow naturally once we take this "stability-as-constraint, fairness-over-time" perspective seriously.

Decentralization and implementation: can rotation be made incentive-compatible? ROSS is a centralized scheduling policy: it precomputes a menu of matchings and then commits to a deterministic (or randomized) rotation. In practice, many markets are not fully centralized. Workers choose when to participate; jobs may have discretion; and the platform may only be able to recommend matches rather than enforce them. A first open problem is therefore *implementation*: under what behavioral or institutional assumptions can a rotation over stable matchings be sustained as an equilibrium of a decentralized process?

One direction is to interpret the rotation index as a form of *time-dependent entitlement*. If workers can observe the index and anticipate future assignments, then they may delay participation to enter on favorable indices, undermining both stability and fairness. This raises a mechanism-design question: can we randomize the phase or conceal the index while preserving auditability? More broadly, what is the right equilibrium concept when (i) priorities are fixed, (ii) utilities may be private information, and (iii) agents can opt in and out? The natural tension is that transparency improves legitimacy, yet predictability creates timing arbitrage. Understanding the minimal information that must be revealed for compliance, while still deterring manipulation, remains open.

A second direction is to search for *distributed analogues* of rotation. For example, one might ask whether repeated applications of deferred acceptance with slowly varying tie-breaking rules can approximate the same time-sharing distribution, or whether there exist local swapping dynamics that converge to a cycle visiting the desired support. Here, internal stability is an advantage: because blocking pairs are only defined among matched agents, one can hope to design decentralized acceptance protocols that preserve internal stability by construction, even if external stability is unattainable.

Multi-stakeholder objectives: beyond the worker-side OSS benchmark. Our benchmark is worker-centric by design: each worker is compared to her best ϵ-stable outcome in each round. This is appropriate when fairness to workers is the salient concern and priorities P_a are non-negotiable. Yet many platforms face multi-stakeholder objectives that do not reduce to a single side. Jobs may care about match quality or completion time; regulators may care about non-discrimination; and platforms may care about revenue, retention, or geographic coverage.

A natural generalization is to define a *vector* of benchmarks—for workers and for jobs—and to ask whether one can rotate over matchings to guarantee simultaneous shares. Even in static markets, such multi-objective guarantees

may be impossible without relaxing feasibility or priorities. In dynamic settings, the question becomes sharper: does churn exacerbate the tradeoff between worker fairness and job-side service quality? Can we characterize Pareto frontiers over long-run shares, e.g.,

$$\left(\frac{\mathbb{E}[\Pi_w]}{\mathrm{dOSS}_\epsilon(w)}\right)_w \quad \text{and} \quad \left(\frac{\mathbb{E}[\Pi_a]}{\mathrm{dOSS}_\epsilon(a)}\right)_a,$$

under a shared stability constraint? A related open problem is the design of stability notions that better reflect job-side concerns when priorities are only a partial ordering, or when jobs have capacity and downstream complementarities (e.g., teams, shifts, or routing constraints). In such environments, internal stability may be too weak to prevent credible complaints on the job side, yet external stability may be too strong to admit any rotation-based fairness guarantees.

More practically, platforms often impose *service-level constraints* (fill rates, response times) that are not captured by stability alone. It is open how to integrate such constraints without losing the small-support structure that makes rotation implementable. One promising approach is to treat these constraints as additional feasibility restrictions defining a smaller set $I_t^{\epsilon,\mathrm{svc}} \subseteq I_t^\epsilon$, and then ask whether an analogue of the $m = O(\log N_{\max})$ support bound still holds.

Learning under churn: stability with partial feedback and changing participants. The analysis assumes that, at the time of matching, the platform observes U_t (or at least an adequate proxy) and priorities P_a. In many applications, utilities are latent: the platform observes noisy acceptance/decline decisions, completion outcomes, ratings, or long-run retention. This raises a central open problem: how do we *learn* utilities while maintaining per-round ϵ-internal stability and providing long-run fairness guarantees?

At a high level, learning introduces an exploration–stability conflict. Exploration requires occasionally deviating from currently estimated best assignments; but such deviations can create blocking pairs relative to the true U_t, violating the compliance constraint. One conceptual resolution is robustification: maintain a confidence set $\mathcal{U}_t$ around the estimated utility matrix and require that μ_t be ϵ-internally stable *for all* $U \in \mathcal{U}_t$. This resembles the drift-robust logic (choose ϵ to dominate uncertainty), but learning makes the uncertainty endogenous and time-varying. The open question is whether one can obtain guarantees of the form

$$\mathbb{E}[\Pi_w] \;\geq\; \frac{1}{m}\mathrm{dOSS}_\epsilon(w) \;-\; (\text{learning regret}) \;-\; (\text{churn penalty}),$$

where the learning term scales sublinearly in $|\mathcal{T}(w)|$ under realistic feedback models.

Churn complicates learning in two distinct ways. First, arrivals and departures change the identity of competitors, so estimates may not transfer cleanly across epochs. Second, the additive $E(w)$ term interacts with exploration: recomputation and experimentation both "reset" effective progress through the rotation. This suggests that the right object may be a *joint* design of (i) epoch segmentation (possibly via change-point detection), (ii) within-epoch exploration schedules, and (iii) a stability slack ϵ calibrated to both statistical error and drift. We do not yet understand the minimal recomputation frequency needed to track nonstationarity while keeping the rotation long enough to deliver its fairness share.

Additional structural extensions. Several modeling extensions are immediate but nontrivial. Many applications are many-to-one (firms with multiple slots) or involve matching with contracts (different hours, pay, or task attributes). It is open whether the logarithmic support phenomenon persists in such richer lattices of stable outcomes, and whether internal stability remains the right compliance proxy. Likewise, priorities may be endogenous or multi-dimensional (e.g., affirmative action constraints, geographic tiers). Embedding such policy constraints into P_a is straightforward formally, but the welfare and fairness implications can be subtle: a rigid priority rule may amplify inequality even as it improves transparency, and the OSS benchmark then measures fairness *conditional on that policy*. Developing diagnostics that separate "policy-imposed scarcity" from "algorithmic allocation" would strengthen the interpretability of any guarantee.

Closing perspective. The broader lesson is that repeated matching platforms should be evaluated not only by one-shot efficiency, but by the long-run distribution of opportunities *under constraints that stakeholders recognize as legitimate*. Rotation over stable matchings is one principled way to operationalize that lesson. The open problems above—decentralized implementability, multi-stakeholder benchmarks, and learning under churn—all ask, in different ways, how far this principle can be pushed before either stability or fairness must be reinterpreted. We view answering those questions as essential for translating theoretical guarantees into robust platform governance.