Massive Online Agent Arena

Théo Fagnoni
Member of Technical Staff
AppliedAI

Phillip Kingston*
Member of Technical Staff
AppliedAI

28 March 2026

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
(CC BY-NC-SA 4.0)

Abstract

When humans and agents collaborate on complex shared tasks, one participant's changes can invalidate another's ongoing work or require others to review and incorporate new information before proceeding. Effective multi-actor coordination must mitigate the risk of actions based on outdated inputs changing shared state and keep required reviews and updates within declared limits, while preserving each participant's ability to plan independently. We introduce Massive Online Agent Arena, a framework that uses collaboration interfaces to determine when proposed actions may change shared state. An arena comprises humans and agents, referred to as actors, together with a shared environment and interfaces that define what actors can observe and request, which actions require review, and which changes must be communicated to others. Actors can consult intentions, which declare planned actions, to anticipate interference and revise their plans. Before applying an action, the environment verifies that its inputs remain current, its author has permission, and any required approvals are valid. It also reserves capacity for resulting updates that other actors must receive or incorporate. Review requests and updates count against declared limits on unfinished tasks, so completing them releases capacity for further actions. As participation and access change, the environment updates the requirements for applying actions and records which conditions leave proposals waiting. This paper makes three contributions:

1. We introduce an arena framework designed to support agent-to-agent and human-to-agent collaboration at scale, with observable dependencies and enforceable interaction rules.
2. We develop intention sharing, action admission, and completion-based flow control, with conditions for valid approvals and bounded unfinished-task weight under changing participation.
3. We demonstrate that the same coordination mechanism supports canvas and document operations while adapting rate restrictions to changing dependencies. Intention-guided selection reduces the average number of preparation attempts by up to 53.6% compared with a baseline that prepares actions without consulting other participants' plans, while its effect on application latency varies across settings. We extend the evaluation to populations of more than 10^6 simulated agents to examine how registry provisioning sustains progress at scale.

*Corresponding author: phillip.kingston@opus.com

1 Introduction

Collaboration among agents and humans involves decisions made from different observations and at different times. Concurrently prepared actions may propose incompatible changes to the same object or rely on inputs that another action changes. Their dependencies can also change during collaboration, for example when a human joins to review proposals or an agent must incorporate new updates before continuing. Without a common account of these dependencies, a collaboration environment can accumulate pending reviews and updates, repeat calculations, or apply actions without required approval.

Increasing the number of agents can generate work faster than other participants can review or incorporate it. Effective collaboration therefore depends on controlling the work that actions create for others, while allowing tasks with independent requirements to proceed.

These concerns already arise in large agent deployments. Cursor reports coordination bottlenecks in experiments with hundreds of coding agents [1], and Anthropic describes duplicated work and excessive updates in its multi-agent research system [2]. Human participation adds differences in attention and decision time to the unequal planning durations and completion rates already present between agents. When participation changes, proposals prepared earlier may become subject to new review requirements, while unfinished tasks created by previous actions must remain accounted for. Coordination must therefore connect the requirements governing an action when it takes effect to the work that remains outstanding as participation changes.

Massive Online Agent Arena uses collaboration interfaces to connect the information available to actors with the requirements governing their actions. Actors retain their own policies and private reasoning while using disclosed observations and intentions to coordinate their plans. When an action takes effect, the environment reserves capacity for the follow-up work required by those interfaces. The environment checks proposals against current inputs, permissions, approvals, and unfinished-task limits, recording unmet conditions to explain delays and rejections. Its guarantees depend on declaring the relevant tasks and applying actions through these checks. The evaluation shows how advance knowledge of planned actions can reduce repeated preparation, while identifying conditions under which review availability or shared processing capacity continues to limit progress.

2 Related Work

Workspace awareness [3, 4], Coactive Design [5], and partial global planning [6] examine how observations and intentions support joint activity. The A&A model makes environment artifacts active coordination resources [7]. Interface theories address compatibility [8, 9], while autonomic computing studies feedback adjustment [10]. The arena uses the same interface declarations to specify what actors can observe and which obligations the environment enforces when they act.

Reactive Streams specifies recipient-demand backpressure [11]. The arena combines current approval checks with reservations for follow-up tasks at a consistent admission boundary and retains each obligation through completion or permitted retirement. This connects the rules controlling when proposals take effect to the records explaining why they wait.

Agent concurrency mechanisms address interference during execution. Atomix delays commitment of tool effects until progress conditions permit it [12], while ATCC adapts optimistic and pessimistic concurrency control [13]. The arena derives both admission prerequisites and follow-up obligations from the current collaboration configuration. A newly required review can therefore block a proposal prepared earlier, while a recipient's departure does not by itself erase obligations created by earlier applications. The same records determine admission conditions and identify the dependencies responsible for waiting.

Collaborative Gym studies human-to-agent collaboration [14], while CLEO examines concurrent shared-workspace interaction and proposes using visibility and cursor signals to guide pausing [15]. Communication pruning offers a complementary way to reduce coordination cost [16]. Its effects can be evaluated within an arena under declared disclosure, approval, and processing constraints.

Opus Workflow Intention represents a workflow's intended transformation through its input, process and output [17]. Prompt Intention uses this representation to guide workflow generation from user queries [18]. These methods specify the work to be performed, while the arena governs how humans and agents coordinate actions as that work progresses.

The Opus Workflow Evaluation Framework assesses expected reward alongside penalties for structural and informational quality [19]. Its case study compares alternative review arrangements under specified cost and success assumptions. The arena complements this perspective by accounting for the review and update work required when participants modify shared state concurrently.

The guarantees require complete obligation declarations, valid completion evidence, and consistent admission. Extending the local implementation to distributed deployment requires a design that preserves admission consistency and accounting through failures and recovery. The present experiments use prescribed policies to examine mechanism behavior. Interoperability, direct comparisons with the cited systems, and benefits for live agents, human participants, and practical diagnosis of coordination delays remain to be evaluated.

3 The Arena Model

3.1 Actors, State, and Interfaces

An actor is a human or agent that performs actions through the arena's interfaces. An environment component maintains shared state or provides a service over it. When an action involves several environment components, they must coordinate its validation and application so that the state used by the checks remains unchanged until the action takes effect. An object is a unit of shared state, and a scope selects the objects relevant to an interaction, such as one canvas region or document section.

Our running example is a collaborative design canvas with regions L, M, and R. Agent A edits values in L, while agent B prepares a calculation using those values. Human H can edit directly and, when configured, review proposals affecting a selected region. Operations in R use separate inputs and reviews from those in L, but operations in both regions may query the same database and therefore share its processing capacity. An edit by A can invalidate B's input during preparation, while delayed review by H can postpone an otherwise valid proposal.

At time t, the arena $\mathcal{A}_t$ is

$$\mathcal{A}_t = (V_t, E_t, \mathcal{I}_t, X, \mathcal{U}, \delta) \qquad \delta : X \times \mathcal{U} \rightharpoonup X \tag{1}$$

The vertex set V_t contains actors and environment components, E_t contains identified directed interfaces, and $\mathcal{I}_t$ assigns a specification to each interface. The domain X contains object states, and $\mathcal{U}$ contains operations with their arguments. The partial function δ gives the resulting object state when an operation is valid. Parallel interfaces between the same vertices are allowed.

A directed interface governs information and requests sent from one vertex to another. For example, an actor sends an edit request to the environment, which returns observations through an interface in the opposite direction. The two directions can have different permissions and disclosure rules. A connection alone grants neither permission to modify an object nor access to all information held at a vertex.

Let $\mathcal{S}$ denote the space of coordination states and $\mathcal{Z}$ the domain of completion-evidence records. The environment maintains a state $\sigma \in \mathcal{S}$:

$$\sigma = (x, \nu, C, \mathcal{R}, \mathcal{L}, K) \in \mathcal{S} \tag{2}$$

The components are defined together in Table 1. A logical fact is a named property checked by an operation, such as an object's value or the set of objects in a region. Its version changes whenever that property changes. For example, summing the numeric values of all objects in a canvas region depends on which objects are present. Adding an object changes what must be included in the sum, even if all existing values remain unchanged.

Component	Meaning
$x \in X$	Current shared object state.
$\nu : \mathcal{F} \to \mathbb{N}$	Version of each logical fact in the set $\mathcal{F}$.
C	Current configuration: active actors and interfaces, permissions, processing contracts, budgets, and a version counter incremented whenever these settings change.
$\mathcal{R}$	Declared intentions, prepared proposals, and identifiers of operations already applied.
$\mathcal{L}$	Identified processing tasks: reserved, delivered, completed, or explicitly retired.
$K \subseteq \mathcal{Z}$	Evidence satisfying prerequisites, including approvals.

Table 1. Coordination state. Private reasoning and unrepresented external state are outside σ.

For an active interface e at time t, its specification $I_e = \mathcal{I}_t(e)$ is

$$I_e = (\Omega_e, U_e, \phi_e, \mathrm{Perm}_e, P_e, \mathcal{M}_e) \tag{3}$$

Let $\mathcal{D}$ be the domain of information records, $\mathcal{Q}$ the request domain, and $\mathcal{P}$ the proposal domain. The notation $\mathrm{Fin}(\cdot)$ denotes the collection of all finite subsets of its argument. Table 2 defines the interface components. An interface may require processing tasks, such as a review of a proposed edit or an update that another actor must incorporate. $\mathcal{D}_O$ denotes the domain of records describing these tasks, and Section 4.2 calls each record an obligation and specifies its components. Configuration C supplies the current interface specifications.

Interfaces can be implemented by a graphical control, an agent endpoint, or a registry query, with disclosure rules determining what the recipient can access. An environment can therefore disclose an assistant's intended edit without revealing its private conversation or prepared payload. The route also determines which interfaces apply: logs shown through an environment dashboard follow agent-to-environment and environment-to-human interfaces, while logs supplied directly by an agent follow an agent-to-human interface. An intermediary's interfaces must specify any filtering, request restrictions, or required reviews and updates it introduces.

Component	Type	Meaning
Ω_e	$\mathcal{S} \to \mathrm{Fin}(\mathcal{D})$	Information permitted on this interface.
U_e	$\mathcal{S} \to 2^{\mathcal{Q}}$	Requests currently available.
ϕ_e	$\mathrm{Fin}(\mathcal{D}) \to Y_e$	Representation delivered to the recipient, in domain Y_e.
Perm_e	$\mathcal{Q} \times \mathcal{S} \to \{0, 1\}$	Whether the sender may issue a request.
P_e	$\mathcal{P} \times \mathcal{S} \to \mathrm{Fin}(\mathcal{D}_O)$	Obligation rule assigning tasks required by a proposal.
$\mathcal{M}_e$	Events, weights, and units	Definition of the measured flows.

Table 2. An interface defines disclosure, requests, permissions, required tasks, and measurement. $2^{\mathcal{Q}}$ is the set of request subsets.

3.2 Collaboration Configurations

An assistant is an agent assigned to support a particular human. Figure 1 contrasts direct environment access in (a), human-assistant pairs without assistant-to-assistant interfaces in (b), and a human and assistant connected to a shared agent in (c). Each actor's interfaces specify its permitted observations and requests.

A canvas may itself contain a workflow graph. Opus represents workflows as directed acyclic graphs of tasks [20]. When such a graph is edited in an arena, it is represented in the shared object state $x \in X$, while the arena graph represents who can interact with it.

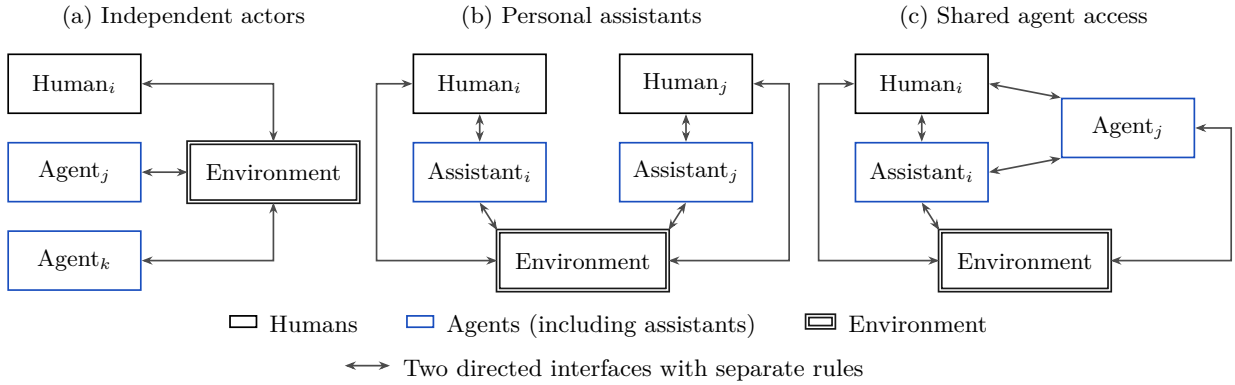


Figure 1. Collaboration configurations. Solid black outlines denote humans, blue outlines denote agents including assistants, and double outlines denote the environment. Matching indices identify personal human-assistant pairs. Each double arrow denotes two directed interfaces with separate rules. Connections permit interaction, not unrestricted access or mandatory approval.

4 Coordination and Admission

4.1 Intentions and Prepared Proposals

An action is an operation an actor plans to perform. The actor may first publish an intention declaring that plan, then submit a prepared action as a proposal a. Applying a produces its effect, the resulting change to shared state. To distinguish the operation from successive preparations, a carries an operation identifier α_a that prevents duplicate application and a proposal identifier η_a that changes whenever preparation uses new inputs or produces a new payload. The remaining fields record the submission interface e_a, the operation $u_a \in \mathcal{U}$ with its prepared payload, and the finite partial map of checked versions $\widehat{\nu}_a : \mathcal{F} \rightharpoonup \mathbb{N}$. Thus

$$a = (\alpha_a, \eta_a, e_a, u_a, \widehat{\nu}_a) \in \mathcal{P} \tag{4}$$

Let $q_a \in \mathcal{Q}$ denote the request carrying a. The version map covers every relied-on input, write target and relevant structural fact. The prepared payload must use values corresponding to the recorded versions, and trusted adapters must preserve this correspondence during preparation. These adapters are code that maps operations to interface rules and observed events to completion evidence.

For actor i, let $\mathcal{H}_i$ be its private-state domain, $\mathcal{V}_i$ its received-view domain, and $\mathcal{D}_i$ its decision domain. Its policy $\pi_i : \mathcal{H}_i \times \mathcal{V}_i \to \mathcal{D}_i$ determines its decision:

$$d_i = \pi_i(h_i, v_i) \tag{5}$$

where h_i is its private state and v_i is the view received through its interfaces. The decision d_i may be to declare an intention, prepare a proposal, or wait. A policy can describe human choice or an automated planner and determines which observations influence the decision. Because access permission does not ensure retrieval or timely delivery, v_i may contain only part of the permitted information or describe an earlier state.

The Action Registry, stored in $\mathcal{R}$, exposes intention records through authorized views. Each record identifies its author, status, and declared read and write footprints: the sets of logical facts the planned action may use or change. Actors can publish, revise, or withdraw a record without applying the action, granting permission, or reserving processing capacity.

Possible interference arises when one planned action may change a fact another uses, or when both may change the same fact. Actors can query these intentions before preparation to revise their plans, defer a calculation, or select another action. Undeclared actions are absent from this comparison, and late declarations may arrive after preparation has begun. Declaring more facts than an action actually uses or changes can instead trigger unnecessary waiting. The actor chooses its response to the declared plans, while the environment checks each proposal against current conditions before applying it. Appendix A.2 gives the retrieval cost and the condition under which an observation determines a check.

4.2 Required Tasks and Processing Accounts

An interface may require a human to review a proposed edit or an agent to incorporate an applied edit into the inputs of a later calculation. Each required activity is a processing task, recorded as an obligation o so that the environment can track whether it remains outstanding. A processing account groups obligations measured in the same unit and limits their total unfinished weight. Each obligation has a processing contract c_o specifying who completes it and what evidence establishes completion, together with its scope, measurement unit, and initial account. We record

$$o = (\iota_o, c_o, w_o, \rho_o, \chi_o) \in \mathcal{D}_O \qquad \rho_o \in \{\text{pre}, \text{post}\} \tag{6}$$

The identifier ι_o distinguishes the obligation, and $w_o \geq 0$ is its weight. The completion predicate $\chi_o : \mathcal{Z} \times \mathcal{P} \times \mathcal{S} \to \{0, 1\}$ tests whether evidence z establishes completion for proposal a in state σ. The phase ρ_o distinguishes a prerequisite (pre), whose evidence must hold when the proposal is applied, from a follow-up task (post), whose capacity is reserved at application. Let $\mathcal{C}$ and $\mathcal{J}$ be the sets of contracts and processing accounts. Each account $j \in \mathcal{J}$ has a finite, nonnegative budget B_j bounding its unfinished-task weight. The mapping $\kappa : \mathcal{C} \to \mathcal{J}$ specifies the initial account. An obligation's assignment starts at $j_o = \kappa(c_o)$, and the ledger records its current account j_o if that assignment later changes.

The active interfaces specified by configuration C form $E(C)$, which equals E_t at time t. For each proposal, only finitely many active interfaces contribute obligations. Each such interface e contributes the obligations $P_e(a, \sigma)$ it imposes on proposal a, and the environment collects them into

$$\begin{aligned} \mathcal{O}(a, \sigma) &= \bigcup_{e \in E(C)} P_e(a, \sigma) \\ \mathcal{O}^-(a, \sigma) &= \{o \in \mathcal{O}(a, \sigma) : \rho_o = \text{pre}\} \\ \mathcal{O}^+(a, \sigma) &= \{o \in \mathcal{O}(a, \sigma) : \rho_o = \text{post}\} \end{aligned} \tag{7}$$

When several interfaces declare the same identity ι_o, the environment counts one obligation if their contracts, weights, phases, and evidence rules agree, and rejects the proposal otherwise. The shared identity prevents duplicate charges, while fresh follow-up identities for each new application prevent reuse of an earlier completion.

For an edit requiring approval, submitting the review request reserves capacity for the review task. Completing it releases that capacity and, if approval is given, records evidence in K bound to the proposal identity and current contract. The proposed edit requires this approval as a prerequisite, so admission checks the evidence again without charging for the completed review a second time.

Contracts distinguish displaying an update, incorporating it into an agent's input, and approving a proposal, without treating these events as proof of human understanding. A cursor movement can indicate focus or change scope, but counts as approval only if the application defines that gesture as an approval event. Likewise, an assistant's acknowledgement can satisfy a human's obligation only when its contract permits delegation. The application encodes these meanings in its trusted adapters. The framework relies on this mapping rather than inferring the rules from actor prompts.

4.3 One Admission Rule

Admission checks compare a prepared proposal with the current interface requirements and available processing capacity. Within account j, reserved tasks have total weight r_j until they are delivered to the actor or service responsible for completing them. Delivery transfers their weight to q_j, where it remains until completion evidence is accepted. The total unfinished weight ℓ_j, remaining capacity b_j, and additional weight $g_j(a, \sigma)$ required by proposal a are

$$\ell_j = q_j + r_j \qquad b_j = B_j - \ell_j \qquad g_j(a, \sigma) = \sum_{\substack{o \in \mathcal{O}^+(a,\sigma) \\ j_o = j}} w_o \tag{8}$$

For unit-weight updates, $B_j = 4$ permits at most four updates awaiting delivery or completion. This accounting limit does not estimate processing speed or an actor's cognitive capacity.

For consistent obligation declarations, $\mathrm{Admit}(a, \sigma)$ states whether a proposal meets the current admission conditions, without guaranteeing that it will eventually be scheduled:

$$\begin{aligned} \mathrm{Admit}(a, \sigma) \iff\ & \mathrm{Permitted}(a, \sigma) \wedge \mathrm{Current}(a, \sigma) \wedge \mathrm{Valid}(a, x) \\ & \wedge \mathrm{Evidence}(a, \sigma) \wedge \mathrm{Unused}(a, \mathcal{R}) \wedge \bigwedge_{j \in \mathcal{J}} [g_j(a, \sigma) \le b_j] \end{aligned} \tag{9}$$

Predicate	Condition
Permitted	e_a is active, $q_a \in U_{e_a}(\sigma)$, and $\mathrm{Perm}_{e_a}(q_a, \sigma) = 1$.
Current	$\nu(f) = \widehat{\nu}_a(f)$ for every captured fact $f \in \mathrm{dom}(\widehat{\nu}_a)$.
Valid	The object transition $\delta(x, u_a)$ is defined.
Evidence	For each $o \in \mathcal{O}^-(a, \sigma)$, some $z \in K$ satisfies $\chi_o(z, a, \sigma) = 1$ under the current contract.
Unused	Operation identifier α_a has not already been applied.

The environment determines the required obligations and checks admission before reserving capacity and applying the proposal. These steps form a consistent admission boundary: relevant state cannot change between validation and application. Successful admission reserves $r_j \leftarrow r_j + g_j$, applies $x \leftarrow \delta(x, u_a)$, advances ν on every changed value or structural fact, and records α_a as applied in $\mathcal{R}$. If any check fails, the attempt applies no effect and leaves no new reservations.

An earlier inspection cannot authorize application because inputs, permissions, or constraints may change in the meantime. The implementation therefore repeats inspection when the configuration version changes and holds a local exclusive lock across the final checks, reservation, and application. It assumes that prepared payloads and inspection records come from trusted internal components.

After delivery, completion evidence accepted under the obligation's recorded contract releases its weight exactly once from the account currently assigned to its identity in the ledger. Reservation, delivery, and completion update the account by weight w as follows:

$$(r_j, q_j) \longmapsto \begin{cases} (r_j + w, q_j) & \text{reserve if } w \le b_j \\ (r_j - w, q_j + w) & \text{deliver the reserved obligation} \\ (r_j, q_j - w) & \text{complete the delivered obligation} \end{cases} \tag{10}$$

Review requests and tasks requiring a joining actor to receive or incorporate information use the same reservation rule. Before creating these tasks, the environment checks that their combined additional weight fits within each affected account's remaining capacity. It then inserts their identities into the ledger and reserves their weight atomically. If any capacity check fails, no new tasks or reservations are created.

Delivery and completion may occur in one implementation step, but capacity is released only for an identified unfinished obligation. Ignoring notifications for completed or retired obligations prevents duplicate releases. Cancellation needs a separate check because it does not establish that external effects have stopped [21]: an unused reservation can be released only once its task can no longer arrive.

The failed condition determines the response: a changed input requires new preparation, a missing permission causes rejection, and missing evidence or capacity can require waiting. The actor can then revise, defer, or withdraw its proposal, while the environment enforces these conditions without merging arbitrary edits.

4.4 Changing Participation and the Accounting Guarantee

The configuration C specifies the active actors and interfaces, their permissions, processing contracts, and budgets. Changes to these settings increment its version counter, and admission uses the current configuration even for proposals prepared earlier. A joining actor may be required to receive or incorporate an initial view, which is recorded as an obligation. When an actor changes scope, the affected contract is replaced for future admissions, so approval evidence tied to the old contract cannot satisfy its replacement.

Reconfiguration must also preserve obligations created under earlier configurations. An actor's departure can release an unfinished obligation only if its recorded contract permits a waiver, which is recorded as retirement rather than completion. Otherwise, the obligation remains assigned or transfers to an account with the same measurement unit and sufficient remaining capacity. The transfer preserves its identity, recorded contract, weight, and delivery status while moving that weight between the corresponding reserved or delivered totals. Budget reductions follow the same principle: a lower limit takes effect only once the unfinished weight fits within it. Reconfiguration updates are atomic with respect to admission, delivery, and completion, so capacity checks and accounting changes use one consistent state.

Proposition 1 (Current requirements and bounded unfinished-task weight)**.** Assume that the application declares all obligations within the covered scope, every covered effect uses Equation (9) at a consistent boundary, and task identities and completion evidence obey their contracts. Initially, each unfinished obligation has one account and $0 \le \ell_j \le B_j$. Under the accounting and reconfiguration rules above, each applied action satisfies its current prerequisites and reserves every follow-up obligation. At every subsequent state,

$$0 \le \ell_j = q_j + r_j \le B_j \qquad (j \in \mathcal{J}) \tag{11}$$

This result is independent of the actor policies π_i and does not require deterministic delays or accurate rate estimates.

The proof follows the reservation, delivery, completion, and reconfiguration transitions (Appendix A.1). Its coverage depends on declaring the relevant obligations and routing effects through admission. The weight bound also bounds the number of unfinished tasks when their weights share a positive lower bound, but does not bound storage for completed records or other coordination state. Although the proposition bounds unfinished-task weight, it does not guarantee eventual completion or fair scheduling. A stopped recipient or a cycle of waiting actors can leave proposals waiting indefinitely.

5 Adaptation, Observability, and Scale

5.1 Proposal Reassessment After Capacity Changes

The arena graph connects actors and environment components through directed interfaces. The processing graph connects proposals to the accounts assigned to their obligations, with each edge recording the obligation's identity, phase, and weight. The waiting graph links each proposal to its unmet admission conditions, such as a missing approval or insufficient processing capacity. Actors use the separate intention-interference relation to identify planned actions whose declared reads and writes may conflict.

Let $\mathcal{P}_t \subseteq \mathcal{P}$ be the finite set of proposals currently under consideration. With their obligations and weights held fixed, define the accounts required by their follow-up tasks as $J_\sigma(a) = \{j : g_j(a, \sigma) > 0\}$. A capacity change confined to accounts $Q \subseteq \mathcal{J}$ can therefore change the capacity test only for proposals in

$$\Gamma_\sigma(Q) = \{a \in \mathcal{P}_t : J_\sigma(a) \cap Q \neq \varnothing\} \tag{12}$$

For proposals outside $\Gamma_\sigma(Q)$, changed accounts have zero required weight, so their capacity inequalities remain satisfied while remaining capacity is nonnegative. All other capacity inequalities have unchanged inputs, so only proposals inside $\Gamma_\sigma(Q)$ require capacity reassessment. This local rule leaves scheduling open and assumes fixed obligations. If an obligation rule P_e depends on changed capacity, its dependencies must be tracked or its obligations recomputed. Configuration changes likewise require refreshed dependencies, while permissions, input versions, and approvals remain separate admission conditions.

5.2 How Completion Controls Further Actions

Reconfiguration changes applicable constraints, while completion releases reserved capacity. When a recipient slows, tasks occupy its account for longer, so proposals requiring it may have to wait for capacity. Long preparation instead increases exposure to inputs changing before a proposal is applied.

Fix account j and budget B_j over an interval $[t_0, t]$. Let $A_j(t)$ and $D_j(t)$ be cumulative admitted and completed obligation weight, including tasks associated with review requests and initial views. With no transfers, waivers, losses, or unused-reservation releases during the interval, conservation and Equation (11) give

$$\ell_j(t) = \ell_j(t_0) + [A_j(t) - A_j(t_0)] - [D_j(t) - D_j(t_0)] \tag{13}$$

$$A_j(t) - A_j(t_0) \leq D_j(t) - D_j(t_0) + b_j(t_0) \tag{14}$$

Additional admitted weight is bounded by completed weight plus the capacity available at t_0. If each action creates one unit obligation and there are no other arrivals, stopping completion at t_0 permits at most $b_j(t_0)$ further actions requiring this account. The restriction takes effect as its remaining capacity is consumed.

For an initially empty, lossless chain of m stages, suppose every unfinished unit occupies exactly one stage s with budget B_s. Let $A_{\text{src}}(t)$ count source admissions and $D_{\text{end}}(t)$ terminal completions. Then

$$0 \leq A_{\text{src}}(t) - D_{\text{end}}(t) \leq \sum_{s=1}^{m} B_s \tag{15}$$

The number of admitted units awaiting terminal completion remains bounded by the sum of the stage budgets. Dividing this bound by elapsed time shows that the average admission and completion rates converge to the same value whenever either limit exists. This describes propagation when downstream waiting prevents upstream completion, which graph connectivity alone does not establish. Branching, batching, retries, and cycles require accounting for their own task transfers and arrivals. Private preparation is constrained only if the actor's loop is bounded or connected to completion feedback.

5.3 Observing Flow and Explaining Delay

The measurement rules $\mathcal{M}_e$ in interface I_e specify the events recorded and their units. An applied action adds one to an action counter, a delivered message adds its byte or token count, and an obligation completed under its contract adds its declared weight. Each event type and unit has a separate counter. Let $N_{e,s}^{(u)}(t)$ be the sum recorded by time t on interface e, for event type s and unit u. Its average rate over the preceding interval of length $\Delta > 0$ is

$$F_{e,s}^{(u)}(t;\Delta) = \frac{N_{e,s}^{(u)}(t) - N_{e,s}^{(u)}(t-\Delta)}{\Delta} \qquad t \geq \Delta \tag{16}$$

A low measured rate can reflect low demand rather than a bottleneck. The waiting graph records which admission checks fail, including accounts with insufficient remaining capacity b_j, and distinguishes these constraints from an actor's decision to defer. Unfinished weight ℓ_j and completion rates show where work accumulates, while failed admission checks identify the conditions preventing dependent proposals from proceeding. A proposal can have several recorded blockers, but this diagnosis covers only dependencies represented in the model.

5.4 Capacity and Configuration Comparison

The budget B_j limits unfinished-task weight in account j, while sustained progress also depends on processing rates. A resource is an actor or service that supplies this processing, such as a human reviewer or database. Activities with the same processing requirements form a class k. The matrix M describes their resource demands: each entry $M_{pk} \geq 0$ gives the amount of processing required from resource p by one activity in class k. For each resource, this demand and its processing rate limit μ_p use a consistent unit. For example, a database requires M_{pk} queries per class-k activity and can process at most μ_p queries per second. If λ_k is the sustained rate of class-k activities, a necessary condition is

$$M\lambda \leq \mu \tag{17}$$

This condition assumes no losses or waivers, with unfinished processing divided by elapsed time tending to zero. Separate review and update accounts may depend on the same human, so both contribute to that human's processing constraint. Likewise, all agents using one database contribute to its query limit. These shared limits do not create additional obligation reservations.

The activity rates not excluded by these resource limits form the necessary capacity region $\Lambda(M, \mu)$. For two configurations with the same activity classes, resources, and rate bounds μ,

$$\Lambda(M,\mu) = \{\lambda \geq 0 : M\lambda \leq \mu\} \qquad M_1 \leq M_0 \implies \Lambda(M_0,\mu) \subseteq \Lambda(M_1,\mu) \tag{18}$$

The componentwise inequality $M_1 \leq M_0$ states that no activity requires more processing from any resource. Every rate vector satisfying the original bounds then also satisfies the new bounds. Comparisons preserve the task, approval, and completion requirements. Disclosure, preparation policies, batching, or indexing may exchange costs between resources, whereas provisioning changes μ. These bounds support comparison but do not establish a feasible schedule or an optimal arena. Appendix A.3 derives Equation (17) from conservation.

6 Evaluation of Adaptive Coordination at Scale

The studies test admission correctness, coordination trade-offs and scale using the same mechanism for canvas and document operations. Obligations have weight $w_o = 1$, so budgets count unfinished tasks. Actors, including simulated humans, follow specified policies with simulated preparation, review and delivery times. Measured runtime instead denotes elapsed time on the computer running the local benchmark or population simulator. Appendix B gives the protocols.

6.1 Application Correctness and Changing Participation

The first study examines collaborative construction of an executable workflow on a shared canvas. The workflow calculates a price through connected operations for discounting, taxation, currency conversion and rounding. An agent prepares an executable version from the current canvas while a simulated human modifies the workflow by inserting a tax node and changing its parameters. These edits can invalidate the version being prepared or reviewed, requiring the agent to incorporate the changes before application. Human approval becomes mandatory during the run and remains required. An independent equation checks the resulting workflow on 16 amounts, including zero, refunds and rounding cases, while another actor performs 25 independent audit updates. These experiments examine the coordination required to obtain a valid, approved workflow, complementing the assessment of workflow quality and expected execution value in Opus [19].

We refer to the baseline policy as Reactive: the agent begins preparation without checking other actors' declared plans and repeats preparation if a conflicting edit invalidates its inputs. Intention-aware waits while a visible planned edit conflicts, without interrupting preparation underway.

Figure 2 distinguishes when the workflow changes relative to preparation and review. In panel (a), the edit occurs during preparation, and advance disclosure allows the agent to postpone preparation until the revised workflow is available. In panel (b), the edit occurs during review, after preparation has begun, so both policies must repeat preparation. Panel (c) adds a reviewer pause: advance disclosure avoids repeated preparation, but the workflow still cannot be applied until review resumes and approval is obtained.

The 240 runs combine these scenarios, two policies, two implementations and 20 random seeds. Implementations share canvas operations and output checks but implement admission and contract selection separately.

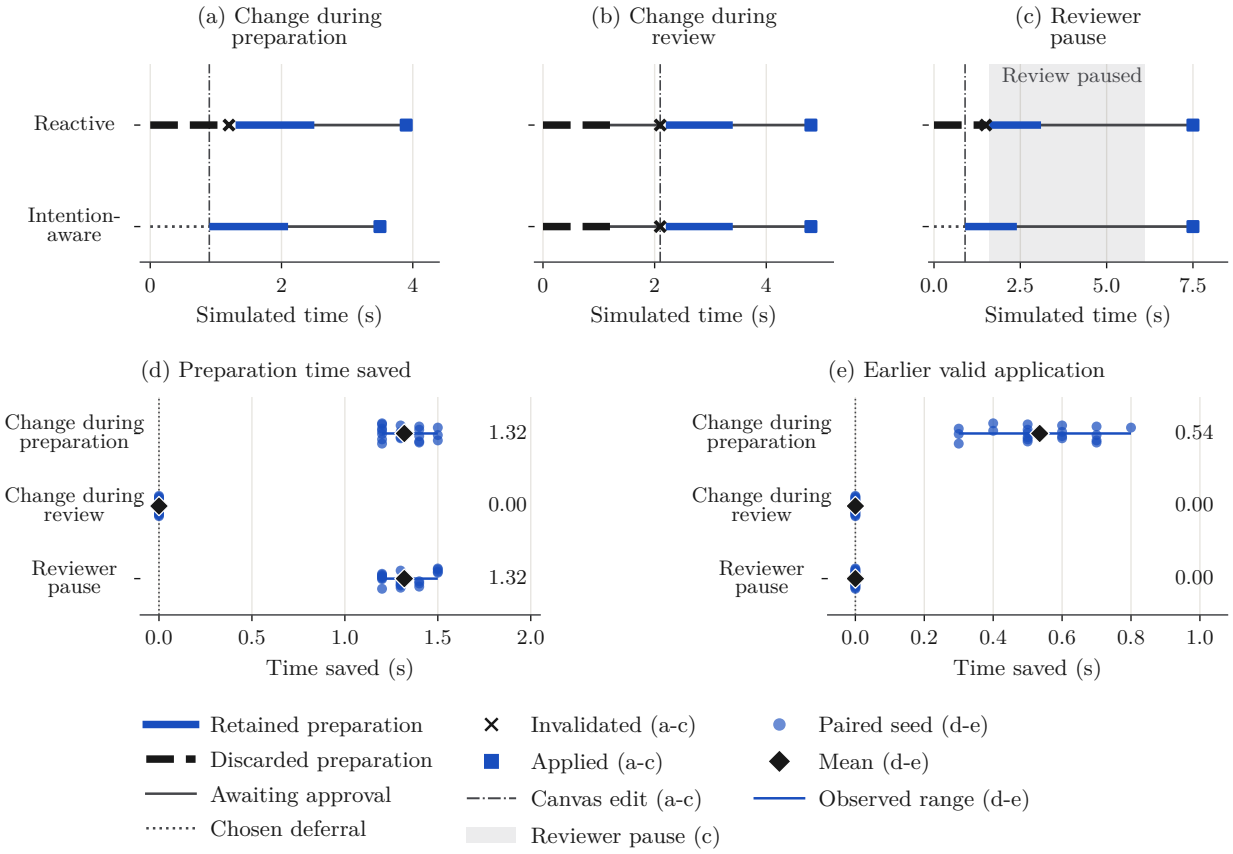


Figure 2. Adapting a canvas calculation. Top: seed-0 timelines show preparation, approval waiting and deferral. Vertical lines mark the canvas edit, crosses invalidation and squares application. Time scales differ. Bottom: time saved by consulting intentions, relative to Reactive. Each dot compares policies with identical random timings. Positive values indicate less discarded preparation or earlier valid application. Diamonds and printed values are means over 20 seeds, and lines show ranges.

All 240 runs produce correct, approved calculations, and the 120 implementation pairs agree. Recovery time runs from the canvas edit to application of the corrected, approved calculation. Early disclosure reduces attempts from two to one and mean recovery from 3.21 to 2.67 s. With a reviewer pause, attempts again fall to one but recovery remains 6.16 s. Disclosure during review leaves both policies at two attempts and 2.77 s, with one stale review rejected per run. The independent branch applies 10-12 audit updates during the pause (Figure 2). Early intentions can therefore avoid repeated preparation, but this does not shorten recovery when reviewer availability determines the application time.

Further canvas and document studies vary observers, inputs, permissions and contracts after inspection. Independent contract-selection methods agree, while omitted dependencies or admission checks produce detectable violations. Duplicate completion notices and competing reservations test task accounting. These controlled checks do not estimate a general failure probability (Appendix B).

6.2 Awareness and Coordination Policy

The second study isolates coordination between concurrent edits and dependent calculations on the shared canvas. Eight agents perform 64 operations that modify shared input values or compute and update private outputs. Because calculations read values that other agents can change, an edit may invalidate another agent's preparation. The experiment compares how coordination policies handle these conflicts, measuring the number of preparation attempts and the simulated time from an operation's arrival to its application. The 180 runs vary shared values (two or eight), intention visibility (public or restricted), and disclosure delay (0, 0.5 or 2 s), with five seeds. One agent prepares slowly, one recipient slows fourfold, and a simulated human joins, changes scope and leaves. Review and update presentation have separate two-item budgets but share one human processor. Because participation changes at fixed simulated times, different policy schedules can incur different review and update workloads. The results therefore do not isolate policy effects under identical processing requirements.

Reactive prepares tasks in arrival order without consulting intentions. Whereas Intention-aware waits for a visible conflicting edit, Intention-guided selection uses a fixed priority order and prepares a task only when no earlier-ranked visible intention conflicts. Conflict locking prevents tasks from preparing concurrently when one task may change a value that another reads or changes. It reserves each task's read and write sets in the same fixed priority order until the task is applied. All policies share admission checks, but the locking coordinator sees every submitted read and write set, including those hidden from peers. Intention-guided selection uses only intentions visible to its actor.

Intention-guided selection reduces mean preparation attempts by 3.8-53.6% relative to Reactive. With two shared values and immediate public disclosure, attempts fall from 141 to 65.4 while mean latency rises from 4.01 to 7.06 s. Restricted disclosure delayed by two seconds gives 85.2 attempts and 7.72 s. With eight shared values, intention-guided selection uses 68.2-77.0 attempts against Reactive's 80 and achieves lower mean latency in every reported setting. Longer disclosure delays do not consistently increase latency.

Conflict locking uses 64 attempts and has the lowest mean latency: 3.18 s with two shared values and 0.90 s with eight. In these settings, Conflict locking achieves both fewer preparation attempts and lower mean application latency than intention-guided selection. The comparison involves different information access: the locking coordinator sees all submitted read and write sets, whereas intention-guided selection uses only the intentions disclosed to each actor. Intention-guided selection averages 3,486.4 registry queries with two shared values and immediate public disclosure. Because registry-query execution is excluded from simulated application latency, these results do not establish whether the reduction in preparation attempts offsets the cost of querying intentions in a deployed system. These comparisons measure policy behavior under changing participation.

All runs finish with no pending tasks or obligations. Thirty additional runs allow review and update presentation to proceed concurrently, separating their completion rates from their unfinished-task budgets. Across all 210 runs, all 13,440 applied actions pass independent checks, and admission rejects 114 stale reviews.

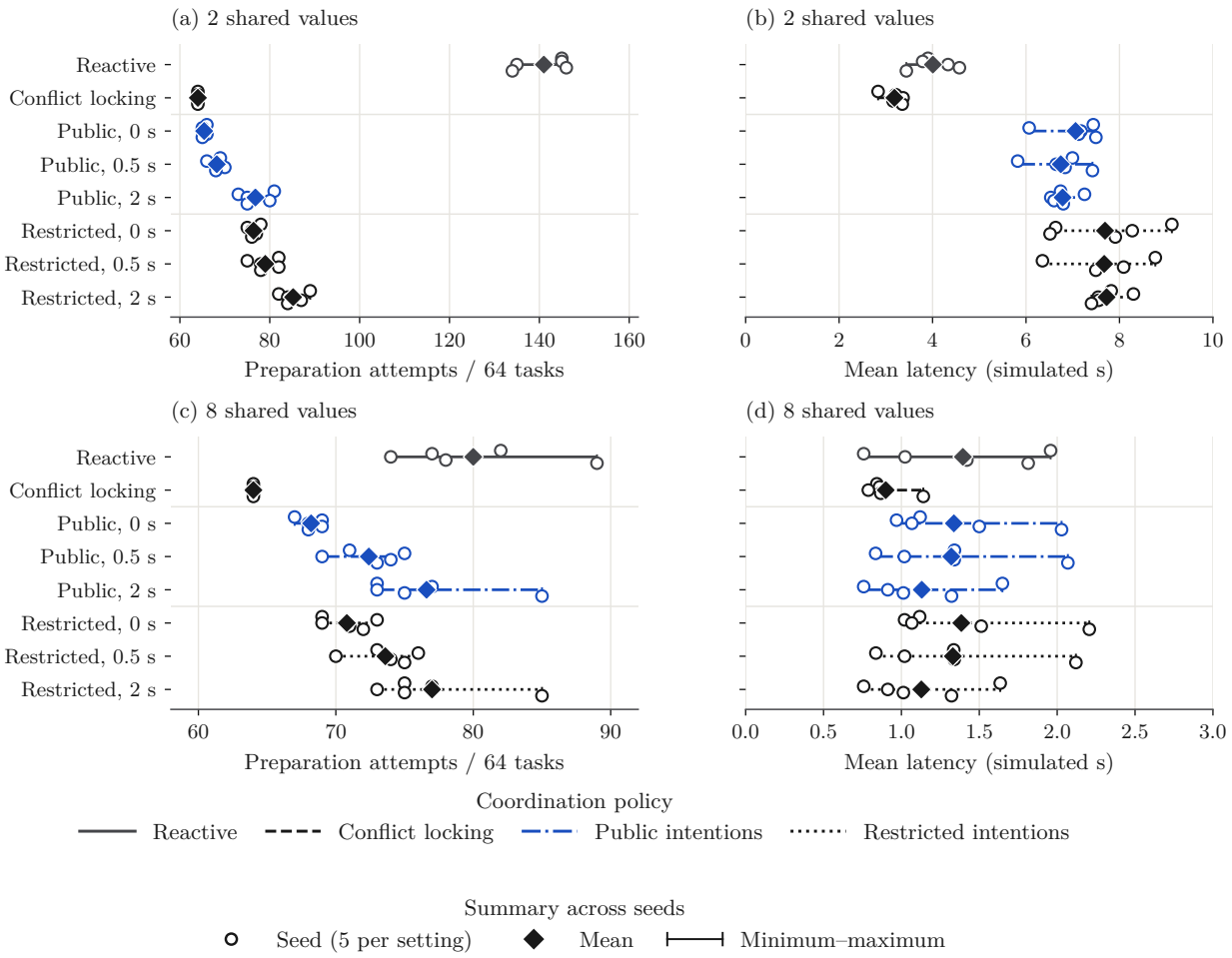


Figure 3. Preparation attempts and application latency. Public and restricted rows use intention-guided selection. Hollow circles show five seeds and filled diamonds their means. Capped bars span the minimum and maximum, with line patterns identifying policies. Disclosure does not affect Reactive or Conflict locking, leaving 16 distinct policy-setting combinations in 180 runs. Scales differ between shared-value counts. The locking coordinator sees all submitted read and write sets.

6.3 Adaptation Along Dependencies

The 36 flow runs use three canvas regions, L, M and R, with 8, 8 and 48 agents generating edits. An agent receiving L updates slows from eight to two updates/s during 15-30 s. A simulated human observes L at 40 s, edits at 44 s, switches to M at 50 s and leaves at 60 s. Edits in the observed region require human updates, completed at two/s with at most four unfinished. Once those four slots fill, further edits wait. Switching regions moves this requirement from L to M, changing which edits must wait for human completion.

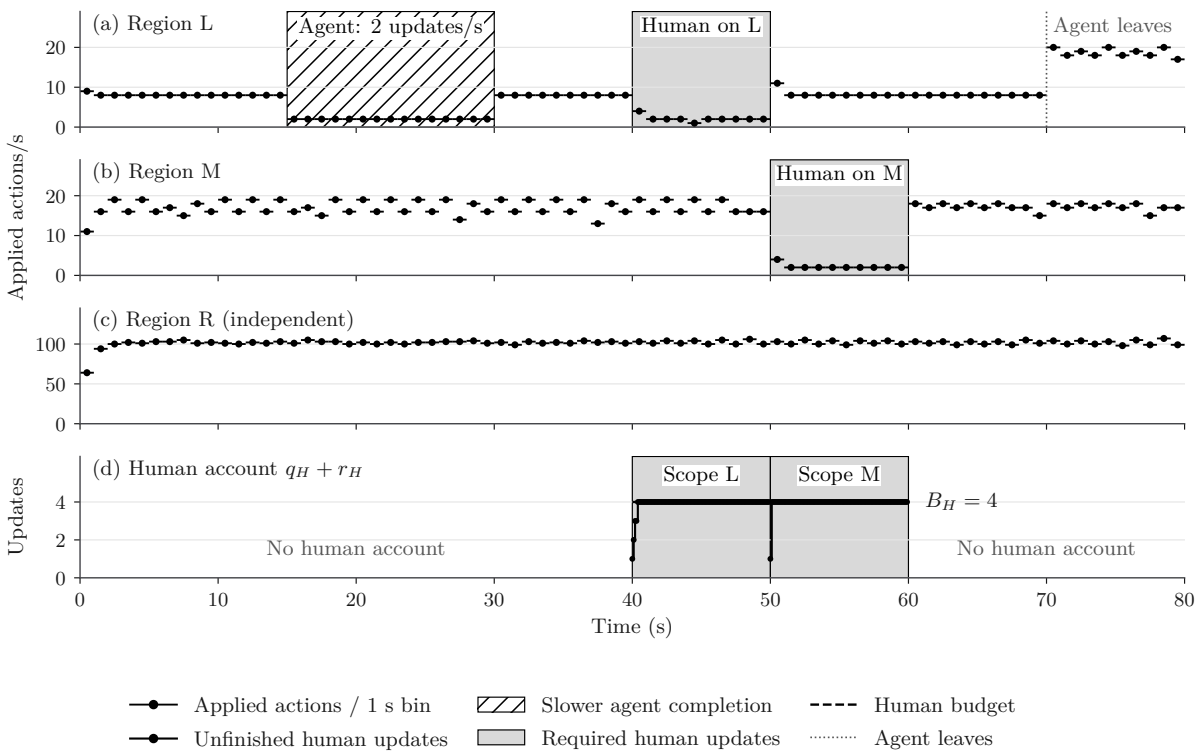


Figure 4. Simulated regional adaptation (seed 7). Panels (a-c) show applied actions per second, with a separate vertical scale for R. Black diagonal hatching marks slower agent completion, while solid grey bands mark required human updates. Panel (d) shows unfinished human updates $q_H + r_H$, sampled every 0.1 s, against budget $B_H = 4$. Human scope moves from L to M at 50 s and ends at 60 s. Four unfinished updates are waived at each transition under the recorded contract. Samples do not resolve every completion and refill. The agent receiving L updates leaves at 70 s.

While the human observes L, independent region R retains 102.33 actions/s. Applying the constraint globally reduces R to 11.67 actions/s, whereas removing the unfinished-task bound permits 828-830 pending obligations. The slowdown therefore follows the region whose edits require human updates, while work in independent regions continues, subject to shared resource limits. These diagnostic variants change the admission rules and are not equivalent policy baselines.

6.4 Coordination Cost and Population Scale

The local benchmark compares an all-contract check, which examines every stored contract, with indexed lookup through edited objects. Its 144 canvas and document runs start with n objects and n contracts, with d contracts initially covering each object. Each applies $2n$ actions for $n \in \{64, 256, 1024, 4096\}$ and $d \in \{1, 8, 32\}$, repeated three times. Both methods produce matching state and coordination records despite observer changes.

At $n = 4096$ and $d = 1$, median canvas runtime is 0.251 s with indexed lookup and 2.967 s with the all-contract check. At $d = 32$, the times rise to 0.917 and 5.785 s as follow-up obligations increase from 8,192 to 262,140. Four first-round edits have 31 recipients after scope replacement, accounting for the four-obligation difference from $8{,}192 \times 32$. This compares the cost of identical admission rules as more contracts and updates apply to each edit.

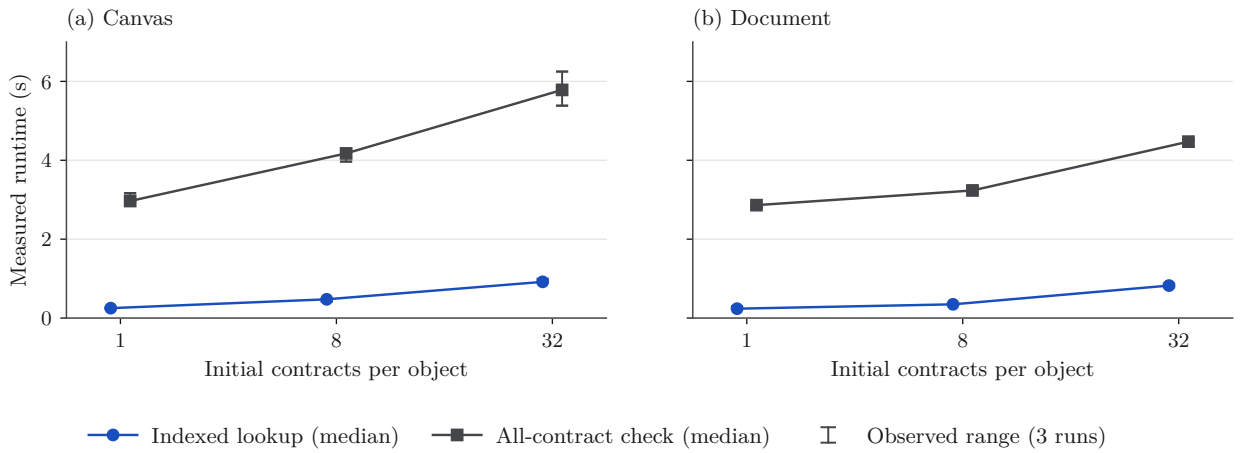


Figure 5. Measured runtime for initialization and 8,192 applied actions at $n = 4096$. Points show medians over three repetitions and whiskers ranges. Timings exclude verification, model inference, network transport, persistent storage and human review time.

The population study simulates larger configurations using bounded inboxes and preparation, random readiness, delays and pauses, changing observers and selected edges, and a shared registry-query service with finite capacity. Its 91 runs cover chains, forks and merges, hubs, modular forward graphs and population sweeps. The registry is simulated rather than executed as a database.

For population N, scaled provisioning sets the registry's base quota to $\max(4, N/8)$ queries per 0.1-second tick and budget to $\max(16, N/2)$ items. Fixed provisioning uses a base quota of 32 and budget of 128. Both vary service stochastically and include a pause (Appendix B). A terminal completion counts one input processed at a terminal node. This measures input processing rather than the completion of unique end-to-end tasks. Over a 20-second horizon and three seeds, scaled provisioning at $N = 1{,}048{,}576$ produces median rates of 195,413.65 terminal completions per simulated second without human observers and 187,016.15 with them. With fixed provisioning, intermediate stages progress but no terminal completions occur within the horizon at $N \geq 65{,}536$. Registry capacity therefore limits how far work progresses through these simulated configurations within the 20-second observation period.

Across 249,173,556 issued obligations, no checked capacity, conservation or current-dependency condition is violated. At the largest agent-only population, median measured runtime is 23.97 s and median peak memory is 200.18 MiB. Runtime and memory measure the cost of running the simulator, while simulated completion rates show how registry provisioning affects progress as population grows. Neither measures deployed agent-fleet performance.

7 Conclusion

Massive Online Agent Arena provides a basis for building collaborative systems in which agents and humans can work independently while remaining accountable for the effects of their changes. Participants can use disclosed observations and intentions to decide when to proceed or revise a plan, while the environment enforces the requirements attached to each action. Its contribution is to connect collaboration requirements to enforceable conditions on actions, giving system designers a way to preserve required review as participation changes and prevent unfinished work from exceeding declared limits. The resulting records also make delays explainable, so coordination problems can be traced to the requirements or resources that constrain progress.

This design applies beyond a particular canvas or workflow wherever changes to shared objects create work for other participants. Its practical value lies in controlling activity through the dependencies that require coordination, allowing work with separate requirements to continue, subject to shared resource limits. The population simulations further show why registry capacity must be considered as collaboration grows. These results can guide the design and evaluation of larger collaborative environments. Establishing the arena framework's practical utility for specific applications will require calibration and validation.

References

[1] Lin, W. (2026). Scaling long-running autonomous coding. https://cursor.com/blog/scaling-agents. Cursor, 14 January 2026.

[2] Anthropic (2025). How we built our multi-agent research system. Engineering report, June 13. https://www.anthropic.com/engineering/multi-agent-research-system.

[3] Dourish, P. and Bellotti, V. (1992). Awareness and Coordination in Shared Workspaces. In *Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work*, pages 107-114. ACM. https://doi.org/10.1145/143457.143468.

[4] Gutwin, C. and Greenberg, S. (2002). A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *Computer Supported Cooperative Work*, 11:411-446. https://doi.org/10.1023/A:1021271517844.

[5] Johnson, M., Bradshaw, J. M., Feltovich, P. J., Jonker, C. M., van Riemsdijk, M. B., and Sierhuis, M. (2014). Coactive Design: Designing Support for Interdependence in Joint Activity. *Journal of Human-Robot Interaction*, 3(1):43-69. https://doi.org/10.5898/JHRI.3.1.Johnson.

[6] Durfee, E. H. and Lesser, V. R. (1987). Using Partial Global Plans to Coordinate Distributed Problem Solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 875-883. https://www.ijcai.org/Proceedings/87-2/Papers/065.pdf.

[7] Omicini, A., Ricci, A., and Viroli, M. (2008). Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432-456. https://doi.org/10.1007/s10458-008-9053-x.

[8] de Alfaro, L. and Henzinger, T. A. (2001). Interface Automata. In *Proceedings of the 8th European Software Engineering Conference*, pages 109-120. ACM. https://doi.org/10.1145/503209.503226.

[9] de Alfaro, L., Henzinger, T. A., and Stoelinga, M. (2002). Timed Interfaces. In *Embedded Software*, volume 2491 of *Lecture Notes in Computer Science*, pages 108-122. Springer. https://doi.org/10.1007/3-540-45828-X_9.

[10] Kephart, J. O. and Chess, D. M. (2003). The Vision of Autonomic Computing. *Computer*, 36(1):41-50. https://doi.org/10.1109/MC.2003.1160055.

[11] Reactive Streams (2022). Reactive Streams Specification for the JVM. https://github.com/reactive-streams/reactive-streams-jvm/tree/v1.0.4. Version 1.0.4, released 26 May 2022.

[12] Mohammadi, B., Potamitis, N., Klein, L., Arora, A., and Bindschaedler, L. (2026). Atomix: Timely, Transactional Tool Use for Reliable Agentic Workflows. arXiv:2602.14849, version 1. https://arxiv.org/abs/2602.14849v1.

[13] Zhou, W., Wang, Z., Peng, Z., Chen, H., Zhang, Y., and Yu, G. (2026). ATCC: Adaptive Concurrency Control for Unforeseen Agentic Transactions. arXiv:2603.13906, version 1. https://arxiv.org/abs/2603.13906v1.

[14] Shao, Y., Samuel, V., Jiang, Y., Yang, J., and Yang, D. (2025). Collaborative Gym: A Framework for Enabling and Evaluating Human-Agent Collaboration. arXiv:2412.15701, version 5, revised 6 December 2025. https://arxiv.org/abs/2412.15701v5.

[15] Son, K., Lee, H., Choi, D., Kim, Y., Kim, T. S., Lee, Y., Chung, J. J. Y., Jung, H., and Kim, J. (2026). "When to Hand Off, When to Work Together": Expanding Human-Agent Co-Creative Collaboration through Concurrent Interaction. arXiv:2603.02050, version 2. https://arxiv.org/abs/2603.02050v2.

[16] Zhang, G., Yue, Y., Li, Z., Yun, S., Wan, G., Wang, K., Cheng, D., Yu, J. X., and Chen, T. (2025). Cut the Crap: An Economical Communication Pipeline for LLM-based Multi-Agent Systems. In *International Conference on Learning Representations*. https://openreview.net/forum?id=LkzuPorQ5L.

[17] Kingston, P., Fagnoni, T., and Altin, M. (2025). Opus: A Workflow Intention Framework for Complex Workflow Generation. arXiv:2502.19532, version 1. https://arxiv.org/abs/2502.19532v1.

[18] Fagnoni, T., Altin, M., Chung, C. E., Kingston, P., Tuning, A., Mohamed, D. O., and Adnani, I. (2025). Opus: A Prompt Intention Framework for Complex Workflow Generation. arXiv:2507.11288, version 3. https://arxiv.org/abs/2507.11288v3.

[19] Seroul, A., Fagnoni, T., Adnani, I., Mohamed, D. O., and Kingston, P. (2025). Opus: A Quantitative Framework for Workflow Evaluation. arXiv:2511.04220, version 1. https://arxiv.org/abs/2511.04220v1.

[20] Fagnoni, T., Mesbah, B., Altin, M., and Kingston, P. (2024). Opus: A Large Work Model for Complex Workflow Generation. arXiv:2412.00573, version 2. https://arxiv.org/abs/2412.00573v2.

[21] Model Context Protocol (2025). Cancellation. https://modelcontextprotocol.io/specification/2025-11-25/basic/utilities/cancellation. Specification version 2025-11-25.

A Proofs and Technical Scope

A.1 Coverage and Accounting

For Proposition 1, admission determines $\mathcal{O}(a, \sigma)$ from current interfaces, checks every prerequisite, and reserves every follow-up obligation before the effect becomes visible. Configuration, permissions, input versions, evidence, and capacities cannot change during these steps. Fresh follow-up identities prevent completion reuse, while the unused-operation check prevents duplicate application. A failed attempt retains neither an effect nor partial reservations. Coverage therefore holds at application without retroactively imposing future contracts.

For account j, a reservation of weight $w \geq 0$ satisfies $w \leq B_j - \ell_j$ and yields $\ell'_j = \ell_j + w \leq B_j$. Delivery preserves ℓ_j, while valid completion decreases it by the weight of one identified unfinished obligation in its current account. Duplicate or retired identities cannot release other tasks. Review requests and initial views obey the same accounting, and unused reservations are released only when arrival is impossible. Reconfiguration atomically preserves obligations, transfers their unfinished weights to accounts with the same unit and sufficient capacity while retaining their identities, recorded contracts, and delivery status, or explicitly waives revocable tasks. A transfer updates the ledger assignment and the corresponding reserved or delivered totals together. A smaller budget takes effect only when the account's unfinished-task weight ℓ_j does not exceed it. Each transition preserves $0 \leq \ell_j \leq B_j$, giving the bound by induction without assumptions on actor policy, intention quality, or completion-time estimates.

Equation (13) accounts for all admissions and completions in a fixed account. Rearranging with $\ell_j(t) \leq B_j$ gives Equation (14). The chain bound in Equation (15) sums the unfinished-task weight of each stage, because every admitted but uncompleted unit belongs to exactly one stage. These bounds supply no positive lower bound on progress.

A.2 Retrieval and Sufficient Observations

Let $\mathrm{Read}(a)$ and $\mathrm{Write}(a)$ be complete declared sets of logical facts that a may use and may change, respectively, including structural facts. For fact f, $I_R(f)$ and $I_W(f)$ contain live registry records declaring reads and writes, respectively. The candidate records that may interfere with a are

$$\mathrm{Cand}(a) = \bigcup_{f \in \mathrm{Read}(a)} I_W(f) \;\cup \bigcup_{f \in \mathrm{Write}(a)} \big(I_R(f) \cup I_W(f)\big) \tag{19}$$

At a consistent snapshot, these candidates cover declared changes to a's inputs and declared reads or changes of its write targets. For at most $b \geq 1$ queried facts and k distinct candidates, hash-based traversal and deduplication take expected $O(bk + b)$ time since each candidate appears in $O(b)$ queried lists. Balanced lookup adds $O(b \log(2 + bm))$ for m live records with at most b facts each. Authorization filtering and payload construction incur additional costs, so restricted disclosure need not make retrieval inexpensive.

A view can inform an actor's planning without determining the outcome of every environment check. For a deterministic check $f : \mathcal{S} \to \{0, 1\}$ and snapshot view $\Phi : \mathcal{S} \to Y$, a function $h : Y \to \{0, 1\}$ satisfying $f = h \circ \Phi$ exists exactly when

$$\Phi(\sigma) = \Phi(\sigma') \implies f(\sigma) = f(\sigma') \tag{20}$$

Necessity follows by applying h to equal views. For sufficiency, assign each realized view the common check value of its producing states. This characterizes information sufficient at the snapshot, without assuming that an actor computes the check or understands the view. Since the state can change afterward, admission still uses current environment state.

A.3 Necessary Resource Capacity

Let $Z_k(T)$ count class-k activities admitted during $(0, T]$, $S_p(T)$ the processing completed by resource p in that interval, and $W_p(T)$ the processing still required there at time T. With exact matrix M, finite initial demand $W_p(0)$, and no losses or waivers,

$$\sum_k M_{pk} Z_k(T) = S_p(T) + W_p(T) - W_p(0) \tag{21}$$

If $Z_k(T)/T \to \lambda_k$, $W_p(T)/T \to 0$, and $S_p(T) \le \mu_p T + o(T)$, division by T gives $(M\lambda)_p \le \mu_p$. Resource demand W_p aggregates the processing required from p, even when the corresponding obligations belong to separate accounts. It does not impose another unfinished-task budget. The resulting rate constraints are necessary and do not establish a feasible schedule.

B Experimental Protocols and Coverage

Study	Runs	Measurement scope
Canvas calculation	240	Outputs, persistent review, matched implementations
Online policies	180 + 30	Arrivals, disclosure, policy and processor comparisons
Fully declared actions	160	All intentions initially visible, contract-selection equivalence
Adapter conformance	60	Canvas/document operations and review races
Local cost	144	Executed construction and admission timings
Scoped flow	36	Seed-7 record shown in Figure 4
Sparse graphs	91	Compact model, including 45 population runs

Table 3. Study families. Simulated task durations and measured runtime describe different costs.

Application boundary. Canvas operations include assignment, movement, copying and a two-input sum, while document operations include replacement, append, copying and a deterministic two-section excerpt. An earlier mixed-canvas prototype also tested deletion and structural edge edits. Inputs and write targets are version-checked. Operations reading every node in a set also read a versioned membership key, so insertion invalidates their assumption that the set is unchanged. Checks exclude uninstrumented reads, and conservative footprints may cause unnecessary invalidation. Authorized humans and agents can review, and humans may edit directly.

Intent queries release author and footprint metadata after audience and timing checks, withholding prepared payloads and private conversations. Assistants receive separate environment views without peer channels, while admission and locking retain internal environment access. This tests content filtering, not distributed privacy, timing side channels or inference from metadata.

Canvas calculation. Review becomes mandatory at 0.3 s and is never waived, with preparation and review lasting 1.2-1.5 s. The reviewer-pause scenario stops service from 1.6 until 5.5-6.2 s. Early intentions appear at time zero and late intentions 0.2 s before the canvas edit, after initial preparation begins. Edits occur during preparation or review.

Both implementations share operations and output checks but separately implement admission and contract selection. Compared events include review activation, canvas edits, preparation, rejection of outdated proposals, review completion and application. Each run completes 26 follow-up obligations and one valid approval, with none pending or waived. Validation reuses 16 cases, not 3,840 distinct tasks. Early and late disclosure average 9.85 and two intention queries, respectively. Recovery time excludes query execution and follow-up processing.

Admission checks. The 160-run study validates 10,240 applied actions and sampled affected-set queries against all-contract checks. All comparisons agree, as do the conformance study's 30 pairs of indexed and all-contract checks. Deliberate dependency omission is detected, and eight diagnostics that disable admission checks expose missing recipients or reservations. The 67 accompanying tests include competing reservations of one available slot and a shared human processor.

Online policies. The fixed order ranks tasks by turn and then producer index, while restricted views hide two producers' intentions from peers. Task identity and seed fix random durations across policies. Preparation lasts 0.2-0.6 s except for one producer at 1.2-2.0 s, and a recipient slows fourfold during 3-8 s. The simulated human joins at 2 s, moves at 6 s and leaves at 10 s, taking 0.5-1.0 s per item. These parameters do not measure human capacity. The 30 additional runs allow two items to be serviced concurrently.

Fixed-time scope changes intersect different policy schedules, producing different obligations for equal action counts. Mean counts, including review and initial views, are 119.6 for Reactive, 113.6 for Intention-guided and 115.8 for Conflict locking, with 4.4, 2.4 and 2.4 mean waivers. Persistent review avoids this variation in the canvas calculation. Repeated disclosure settings add no independent evidence for policies that ignore intentions. Neither study measures saved model tokens.

Flow and scale. Regional-flow producers have random starting times and fixed preparation periods. Completion periods are constant within each phase. Local benchmarks replace an observer between inspection and application on every eighth first-round operation and exclude verification. Archived online runtimes include simulation and output checks, so they do not measure coordination cost alone.

Forty sparse-model runs combine four graph families, five seeds and human presence or absence, with extensions yielding 91 runs. Agent counts remain fixed, and ticks represent 0.1 simulated seconds. Registry service is Poisson-distributed with mean equal to the base quota times a multiplier from $\{0.15, 0.5, 1, 1.5\}$, refreshed every ten ticks, except during a zero-service pause. Merge inputs are processed separately without waiting for all inputs, and excess intake is declined. Terminal completions count processed inputs, not unique end-to-end tasks. The model runs neither inference nor a distributed registry. All numerical figures use recorded runs.