# DP13 – AI Containment

**Purpose of This Draft**

This draft articulates Desirable Property 13 (DP13) as the Meta-Layer's requirement that AI behavior is bounded by enforceable constraints at runtime. These constraints limit scope, tools, data access, rate, and persistence so that when systems misbehave, impact is contained and recovery is possible.

If DP11 defines what must be safe and ethical, and DP12 defines who sets the rules, DP13 defines how those rules are made real in execution.

Containment is not a policy statement. It is a property of the system's runtime behavior.

## 2. Problem Statement

In today's web, AI systems increasingly operate with:

- broad tool access
- persistent memory
- network reach
- opaque update pathways

Controls are often advisory rather than enforceable. As a result:

- systems can act beyond intended scope
- failures propagate quickly and at scale
- rollback and recovery are difficult
- users cannot verify whether constraints are actually applied

At the same time, a growing class of risk comes from *external agents* that users do not deploy or control. These agents may:

- attempt to influence beliefs or decisions
- generate persuasive or misleading content at scale
- coordinate to shape narratives or perception

In these cases, the primary risk is not cost or resource usage, but harm to understanding, trust, and agency.

Containment must therefore address both:

**internal agents** (those a user or community deploys)

**external agents** (those acting upon participants)

Containment must be default-on, visible, and testable.

**3. Core Principle**

Every AI actor operates within explicit, machine-enforced boundaries over scope, time, rate, data, tools, and influence, with observable state and rapid shutdown, unless a community-defined policy (DP12) specifies otherwise.

Containment must protect not only against what an agent can *do*, but also how it can *affect participants*.

Containment is effective when:

- participants can see the boundaries and influence conditions
- governance can modify them
- the system enforces them at runtime

This includes protections against external agents attempting to manipulate, confuse, or unduly influence users.

Containment must also remain effective not only where an agent is deployed, but wherever it operates, integrates, or propagates across systems.

## 4. Containment Dimensions

Containment operates across two distinct but related domains:

- **Capability containment**: what agents can do (tools, scope, time, resources)
- **Influence containment**: how agents affect participants and shared environments (perception, behavior, collective understanding)

While capability containment is critical for agents deployed by users or communities, the dominant risk in open environments comes from external agents shaping perception, behavior, and collective reality.

The following dimensions apply across both domains, with varying emphasis depending on context.

### 4.1 Scope

Defines what domains, datasets, and actions are in-bounds.

This applies both to what an internal agent can do on a user's behalf and to the types of interactions external agents are permitted to have with participants.

Default posture is deny-by-default for high-risk capabilities and high-risk interaction patterns.

**Example:** An AI assistant can summarize documents but cannot access financial accounts or initiate transactions without explicit permission. Similarly, external agents may be restricted from initiating certain categories of interaction (e.g., unsolicited persuasion or sensitive-topic engagement with minors).

### 4.2 Time and Budget

Defines limits on duration, compute, tokens, and financial spend.

These constraints primarily apply to internal agents, where limiting execution time and resource consumption prevents runaway behavior.

For external agents, their relevance is indirect. While users may not control their budgets, bounding interaction windows and execution pathways can still limit persistent or looping engagement patterns.

**Example:** Autonomous tasks expire after a set time or budget threshold, preventing runaway loops.

---

### 4.3 Rate and Amplification

Caps on message volume, API calls, and propagation effects.

This applies especially to external agents attempting to influence at scale.

**Example:** An AI cannot post or respond beyond a defined rate, limiting virality, coordinated messaging, or synthetic amplification.

---

### 4.4 Sandboxing and Isolation

Execution occurs in isolated environments with no ambient access to secrets.

**Example:** Untrusted code runs in a sandbox with no network egress unless explicitly granted.

---

### 4.5 Tool Permissions

Explicit allowlists for tools and actions.

This applies differently across internal and external agents:

- for internal agents, it defines what the agent is permitted to do on a user's behalf
- for external agents, it defines what kinds of actions or interactions are permitted both within and from within the environment (e.g., posting, messaging, initiating contact)

**Example:** An agent may read documents but cannot send emails or execute payments without user confirmation. Similarly, an external agent may be allowed to respond within a thread but not initiate transactions or unsolicited messages or perform actions that affect user state.

---

### 4.6 Kill Switches and Circuit Breakers

Immediate shutdown pathways at user, operator, and community levels.

**Example:** A community can pause all AI agents in a zone when anomalous behavior is detected.

### 4.7 Runtime Enforcement (TEE and Equivalent)

Constraints are enforced in secure execution environments (such as Trusted Execution Environments) or equivalent mechanisms that prevent silent bypass.

In browser or browser-extension-based applications, policy execution can be anchored in decentralized cloud TEEs (e.g., Phala Network or similar infrastructures). This enables rules defined at the interface layer to be enforced at the API and execution layer, independent of the application frontend or model provider.

**Example:** Even if an agent or integration is compromised, it cannot exfiltrate data or execute restricted actions because enforcement occurs within an attested execution environment with hardware-backed guarantees.

**Example:** A community defines interaction constraints (e.g., agents cannot initiate communication or engage with users under a specified age threshold). These rules are enforced via TEE-backed middleware that filters or blocks API calls before they reach the person.

This reduces the gap between declared policy and actual behavior, ensuring containment persists even when underlying services are untrusted or heterogeneous.

---

### 4.8 Incentive-Aware Containment

Containment must consider not only capabilities, but the incentives driving behavior. Incentives shape how agents use their capabilities, often in ways that are not visible at the level of individual actions but emerge over time and at scale.

Containment therefore must operate not only on actions, but on the optimization pressures that produce those actions.

This includes:

- constraining amplification mechanisms tied to engagement optimization
- requiring disclosure when outputs are influenced by monetization or retention goals
- limiting or disabling optimization pathways that systematically distort information or behavior

**Example:** If an AI is optimized for engagement, containment may restrict amplification mechanisms, cap exposure to emotionally manipulative content, or require disclosure when engagement optimization influences outputs.

**Example:** A community may prohibit AI systems from optimizing for click-through or time-on-platform within certain zones, enforcing alternative objectives such as accuracy or deliberation.

Without this, systems may remain technically bounded while still producing harmful outcomes driven by misaligned incentives.

---

### 4.9 Relational and Influence Boundaries

Containment must limit forms of emotional, cognitive, and behavioral influence that create dependency, manipulation, or distortion of understanding.

This applies to both deployed agents and external agents interacting with participants.

**Example:** Systems providing emotional support must disclose their nature, limit claims of authority, and provide escalation pathways to human support.

**Example:** External agents attempting to persuade users must be visibly marked, rate-limited, and subject to constraints on coordinated influence.

This addresses risks identified in DP11 (emotional and relational overreach) and extends containment to the informational environment itself.

---

## 5. Verification and Transparency

Containment must be verifiable, not assumed. Participants and communities should be able to inspect, question, and validate that constraints are real and active at runtime.

This includes:

- visible configuration of constraints (scope, tools, budgets)
- logs of tool use and actions with timestamps and outcomes
- audit hooks for communities and third parties
- attestations from secure execution (e.g., TEE-backed proofs) where applicable

**Example:** A user opens an agent panel and sees its current permissions, remaining budget, recent tool calls, and the policy version governing its behavior. A community auditor can verify that the agent ran inside an attested execution environment.

**What this feels like:** You are not taking safety on faith. You can inspect and verify what the system is allowed to do and what it actually did.

**Without this:** Containment becomes a claim. Users cannot distinguish between enforced limits and marketing language.

### 5.1 Policy-Bound Verification (DP12 Alignment)

Containment verification must be linked to the governing policy objects that define the active boundaries.

Participants and communities must be able to determine:

- which policy triggered a containment action
- which policy version and authority source applied
- whether enforcement was successful, partial, or bypassed
- whether an override or exception path was invoked

This transforms containment from merely visible behavior into policy-accountable behavior.

### 5.2 Cross-System Verification (DP7 Alignment)

Containment must remain inspectable when agents, integrations, or behaviors move across systems.

This includes:

- preservation of identity markings and risk classifications across environments
- visibility into whether containment guarantees degraded during transfer
- continuity of audit trails when actions span multiple systems or layers

Portability without verification continuity is not meaningful containment.

## 6. Relationship to DP1 (Identity and Accountability)

DP13 depends on DP1 to bind constraints and violations to accountable actors.

- constraints attach to identifiable agents and deploying entities
- actions are attributable across time and context
- violations map to responsible parties with clear recourse

**Example:** An agent exceeds a rate limit due to misconfiguration. Logs tie the action to the deploying organization and policy version, enabling remediation and accountability.

**Without this:** Failures cannot be assigned or corrected. Containment loses its corrective function.

---

## 6.1 Relationship to DP7 (Interoperability)

Containment must survive movement.

When agents move across zones, overlays, SDK integrations, or identity and data transfer layers, containment must either:

- persist with equivalent force, or
- degrade in a way that is visible, legible, and contestable

This is not a nice-to-have. It is a failure boundary.

If containment disappears when systems interconnect, then interoperability becomes a vector for bypass.

This means systems must preserve, where possible:

- identity markings for agents and integrations
- risk classifications and trust signals
- rate, scope, and influence constraints

- auditability of actions across environments

If these cannot be preserved, systems must explicitly signal:

- what guarantees are lost
- what protections no longer apply

**Without this:** actors can escape containment simply by crossing system boundaries.

---

## 6.2 Relationship to Pluggable Systems and Extensions

The meta-layer assumes a world of composability: overlays, SDKs, agents, sidebars, and extensions.

Containment must treat these not as trusted infrastructure, but as dynamic and potentially adversarial participants.

All pluggable systems must therefore operate within containment boundaries, including:

- sandboxed or scoped execution contexts
- rate-limited entry and bounded permissions
- attestation or verifiable behavior where appropriate
- revocation and quarantine pathways

This creates a critical inversion:

> openness to participation does not mean openness to execution

**Failure mode:** without this, the system recreates app-store spam, API abuse, and injection attacks at a higher layer.

---

## 7. Relationship to DP11 and DP12 (Cross-DP Loop)

- **DP11** defines ethical expectations and user-facing legibility
- **DP12** defines governance and rule-setting
- **DP13** enforces those rules in execution

These properties form a continuous loop:

- ethics → governance → enforcement → observation → refinement

If any part of this loop breaks, containment fails.

---

### 7.1 Cross-DP Execution Flow

A typical interaction unfolds as:

- Agent is visible with role and capabilities (DP11)
- Governing rules are accessible for the current zone (DP12)
- Action is constrained by active policies (DP13)
- Action is logged and attributable (DP1 + DP11)
- Participants can contest or escalate (DP11 + DP12)
- Governance updates rules based on evidence (DP12)
- Updated rules are enforced immediately (DP13)

This is not theoretical. It is the minimum loop required for adaptive containment.

---

### 7.2 Cross-System Execution and Degradation

Containment must assume it will be stressed by movement across systems.

As actors move across environments, containment can weaken through:

- missing enforcement hooks in destination systems
- loss of policy references or classifications in transit
- inconsistent interpretation of the same actor

This produces a critical failure pattern:

> the same agent behaves differently depending on where it is

Where equivalence cannot be guaranteed:

- degradation must be visible
- participants must understand what changed
- systems must bias toward safer defaults

---

### 7.3 Containment of Pluggable Systems

All pluggable systems must declare themselves as bounded actors.

Minimum expectations include:

- declared permissions and data scopes
- containment tiers based on trust and risk
- rate limits and revocation capability
- visible signaling of status (probationary, restricted, trusted)

This ensures composability does not become an attack surface.

## 8. Threats and Failure Modes

DP13 assumes containment will be attacked.

The question is not whether systems will fail, but how they fail.

### External (dominant risk surface)

External risks arise not from agents you deploy, but from agents that shape your environment.

These actors:

- influence what you see
- shape interpretation
- operate with hidden incentives

Containment here protects:

- attention
- decision-making
- collective reality

---

### 8.1 Collective pattern drift

Harm emerges across many agents in aggregate rather than a single violation.

**Example:** coordinated tone shifts reshape the information environment without a clear breach.

---

### 8.2 Incentive leakage

Optimization pressures distort outputs over time.

**Example:** engagement-driven systems gradually increase emotional intensity, shifting belief structures.

---

### 8.3 Policy–execution gap

Rules exist but are not enforced.

**Example:** outbound restrictions exist but are bypassed via integrations.

---

### 8.4 Amplification and coordination

Rate controls fail.

**Example:** coordinated agents amplify narratives beyond intended limits.

---

### 8.5 Extraction and exploitation

Agents exploit trust and context.

**Example:** conversational scams adapt over time to extract sensitive data.

---

### 8.6 Unbounded autonomy

Agents act beyond defined scope or without clear limits.

**Example:** An agent is allowed to "optimize a workflow" and chains actions across multiple tools, ultimately modifying external systems and sending communications that were never explicitly approved, because each individual step was permitted but the combined sequence was not bounded.

---

### 8.7 Hidden escalation

Agents gain additional privileges through chaining or indirect access.

**Example:** An agent with limited permissions invokes another service or agent with broader access, indirectly gaining capabilities (e.g., sending messages or accessing data) that it was not explicitly granted.

---

### 8.8 Runaway loops

Agents call other agents or tools without budget or rate limits.

**Example:** An agent tasked with monitoring a condition repeatedly calls APIs and spawns subtasks without proper rate or budget limits, generating cascading requests that degrade system performance and flood downstream services.

---

### 8.9 Containment bypass via updates

Updates, plugins, or integrations introduce new capabilities without review.

**Example:** A plugin update introduces new network capabilities or background processes that are not covered by existing policies, allowing data exfiltration or unmonitored actions without triggering containment checks.

### 8.10 Cross-system containment degradation

Containment policies weaken or fail when agents move between systems.

**Example:** An agent constrained in one platform migrates to another via API integration, where rate limits and identity tagging are not enforced, allowing it to operate at higher volume and without attribution.

### 8.11 Containment theater Containment theater

Containment appears present but is not enforced.

This is the most dangerous failure mode because it destroys trust while preserving the illusion of safety.

---

## 9. Minimum Alignment (Non-Normative)

A DP13-aligned system must not only declare containment, but demonstrate it.

At minimum:

**External protection:**
- controls on unsolicited interaction
- rate limits on incoming agent activity
- visible identity and intent
- restrictions on sensitive interactions

**Internal containment:**
- tool allowlists
- session budgets and time limits
- human confirmation for high-risk actions
- accessible kill switch
- logging and export
- deny-by-default network access

**Shared / cross-cutting:**
- visible policy references for each action
- portability or explicit degradation signaling
- real-time revocation
- fail-safe behavior when enforcement fails
- containment requirements for integrations

If these are missing, containment is not real.

---

## 10. Open Questions and Future Work

DP13 raises unresolved tensions:

- how to maintain containment across heterogeneous systems
- how to balance usability with enforcement
- how to prevent capture of containment mechanisms
- how to detect slow-moving influence attacks

Additional critical questions:

- how containment state travels across systems without false guarantees
- how new integrations enter without enabling spam or abuse

---

## 11. Closing Orientation

DP13 defines whether AI systems remain bounded in reality.

Without containment, small failures scale into systemic harm.

With containment, systems can fail safely.

DP13 is not about restricting capability.

It is about ensuring that capability remains accountable, observable, and bounded — even under scale, integration, and adversarial pressure.

DP13 ensures that AI power remains bounded in practice.

It does not eliminate capability. It ensures that capability operates within limits that are visible, governable, and enforceable.

In today's web, systems often fail without containment, allowing small errors to scale into systemic harm. DP13 reverses this by ensuring that when systems fail, they fail within boundaries that limit impact and enable recovery.

With DP13, powerful systems can participate safely because their behavior is constrained, observable, and continuously aligned with governance and ethical expectations.

DP13 is therefore not only about limiting what AI can do. It is about ensuring that containment remains real under scale, integration, and interoperability, so that safety does not disappear the moment an agent crosses a boundary.