Puffin Secure Browser

Technical White Paper

Part II: Distributed Compositor Architecture for Responsive Remote Browsing

CloudMosa, Inc.

February, 2026

Executive Summary

Remote Browser Isolation (RBI) solutions traditionally impose a hard tradeoff: strong isolation degrades interactivity, increases latency, and consumes significant bandwidth. Puffin Secure Browser eliminates this compromise by extending Chromium's native process architecture across client and cloud while preserving a near-local user experience.

Puffin's foundation is **RemoteMojo**—a network-extended implementation of Chromium's Mojo IPC boundary. RemoteMojo relocates untrusted web execution (renderer, network, and storage subsystems) into a disposable server-side environment while retaining a native client-side presentation stack.

Built on top of RemoteMojo is Puffin's second architectural pillar: the **Distributed Compositor (Client/Server CC)**, internally known as CC-NG. This is not an overlay, proxy technique, or "acceleration SDK." It required reinterpreting Chromium's compositor frame lifecycle across a network boundary while preserving scheduling invariants, frame production semantics, and determinism.

The Distributed Compositor replicates Chromium's compositor ("cc") semantics across both halves of the system:

- The **server-side compositor** remains tightly coupled to Blink's layout pipeline and is the authoritative source of page state.
- The **client-side compositor** maintains a synchronized rendering model and can respond immediately to many user interactions—such as scrolling and compositor-driven animations—without waiting for a network round trip.

The governing principles are:

- **Predict locally when safe**, removing round-trip latency from interactive operations.
- **Converge asynchronously to authoritative state**, correcting mismatches without breaking security, determinism, or compatibility.

This document explains why distributed compositing is structurally superior for RBI, how the architecture works at a high level, and why it represents durable, infrastructure-grade browser IP rather than an incremental feature.

The RBI Problem: Isolation Without Usability Collapse

Modern browsers represent the primary enterprise attack surface. RBI reduces endpoint risk by relocating untrusted execution off-device. However, most RBI systems feel inherently remote:

- Scroll and gesture responsiveness are bounded by server frame production.
- Animations degrade under network RTT and jitter.
- Continuous pixel streaming consumes substantial bandwidth.

Structural Limits of Common RBI Approaches

1. Video / Pixel Streaming

A remote machine renders frames and transmits compressed video to the endpoint.

- Latency includes render + encode + network + decode.
- Bandwidth scales with frame rate and visual change.
- Visual fidelity is constrained by compression.
- The endpoint behaves as a terminal.

2. DOM / Content Reconstruction

A server sanitizes or transforms content and re-renders locally.

- Compatibility diverges as web APIs evolve.
- Security depends on perfect transformation logic.
- Rendering behavior may drift from Chromium semantics.

Both approaches treat interactivity as a transport artifact. Puffin treats interactivity as a property of Chromium's architecture itself. Puffin preserves Chromium semantics at the IPC boundary and restores responsiveness at the compositor boundary.

Core Insight: Separate the Interaction Loop from the Authority Loop

In Chromium, responsiveness is primarily produced by the compositor subsystem ("cc"), not by layout or JavaScript execution. The compositor:

- Maintains a scene graph (layers, transforms, clips, effects)
- Executes compositor-eligible animations
- Drives scroll updates
- Produces frames at display cadence

In a local browser, many interactions never require a full layout + JS round trip. Most RBI systems lose this property by turning the endpoint into a thin terminal.

The Distributed Compositor restores it by providing the endpoint with a **real compositor instance** capable of immediate presentation updates—while keeping all untrusted web execution strictly server-side.

Architecture Overview

Client–Server Responsibilities

Client-Side (Trusted Rendering – Endpoint)	Server-Side (Untrusted Execution – Cloud)
● Compositing & rasterization ● GPU presentation ● Input capture ● Display scheduling & frame submission ● Predictive compositor updates	● HTML parsing ● JavaScript / WebAssembly execution ● Layout & style computation ● Network requests ● Storage operations ● Authoritative state evolution

The server runs the **web engine loop**. The client runs the **presentation loop**.

What “Distributed Compositor” Means

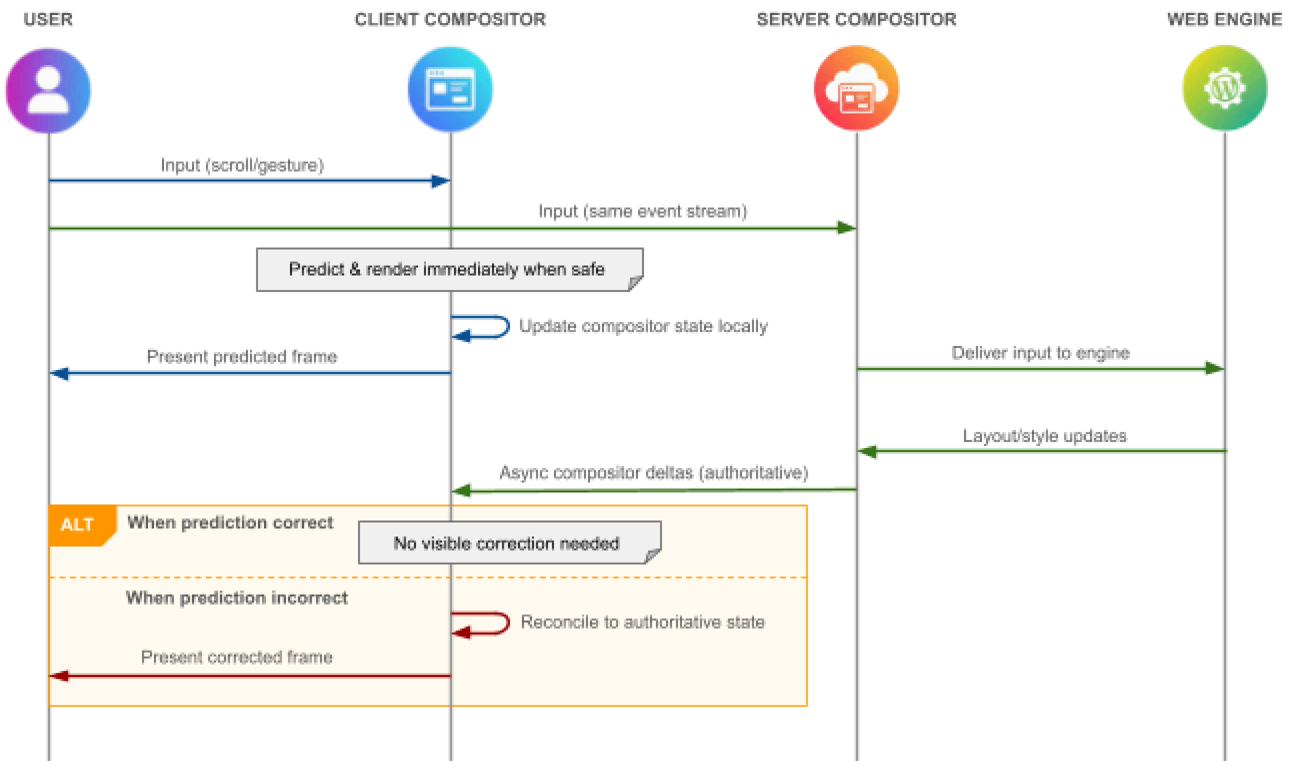
Puffin maintains two synchronized compositor instances:

- **Server Widget (Authoritative Compositor)**
  Coupled to Blink; produces canonical compositor state.
- **Client Widget (Predictive Compositor)**
  Maintains a synchronized replica of compositor state and produces frames locally.

The server continuously transmits **incremental compositor deltas**.
The client:

- Rasterizes locally
- Applies compositor-safe predictions immediately
- Continues compositor-driven animations at display cadence
- Asynchronously reconciles to authoritative state

High-Level Data Flow


USER
CLIENT COMPOSITOR
SERVER COMPOSITOR
WEB ENGINE
Input (scroll/gesture)
Input (same event stream)
Predict & render immediately when safe
Update compositor state locally
Present predicted frame
Deliver input to engine
Layout/style updates
Async compositor deltas (authoritative)
ALT
When prediction correct
No visible correction needed
When prediction incorrect
Reconcile to authoritative state
Present corrected frame

This dual-delivery of input is intentional. It ensures:

- The server remains authoritative (security and compatibility)
- The client can be responsive (experience)

Architectural Invariants Preserved

Distributed compositing is governed by strict invariants:

- Blink remains authoritative.
- Security authority remains server-side.
- Compositor state is eventually consistent.

- Client prediction never alters web semantics.
- Frame lifecycle semantics are preserved end-to-end.

BeginFrame cadence, frame deadlines, and submission semantics remain aligned with Chromium’s lifecycle expectations. This is essential for determinism and upgrade sustainability.

Prediction + Asynchronous Convergence

Prediction is intentionally constrained to compositor-native operations, including

- Scroll offset updates
- Compositor-driven CSS animations and transitions
- Visual feedback loops already eligible for compositor execution

These are executed immediately at display cadence.

The server processes the same input stream, advances authoritative state, and transmits deltas. The client converges without visual instability. This ensures:

- **Responsiveness** (first visible response not RTT-bound)
- **Correctness** (authoritative state always wins)
- **Security** (no client-side semantic execution)

Compositor State Synchronization

Preserving compositor semantics across machines is fundamentally different from transmitting pixels. The client must reconstruct:

- Scene graph structure
- Property trees (transform, clip, effect)
- Damage/invalidation regions
- Paint/raster inputs

- Animation timelines
- Scroll-linked state

Critical Engineering Challenges

Distributed compositing requires maintaining:

- BeginFrame alignment and scheduling coupling
- Deterministic property tree reconstruction
- Frame token and surface identity consistency
- Raster invalidation correctness under prediction
- GPU resource lifetime synchronization

This is not merely “state replication.” It is lifecycle preservation under network conditions.

Delta Synchronization

Full-state transfer would approximate pixel streaming in bandwidth cost. Puffin’s model:

- Establish baseline scene
- Transmit only incremental state changes
- Preserve frame-to-frame stability

Under interactive SaaS workloads, only a small subset of compositor state changes per frame, making delta transmission structurally more efficient.

Security Posture

Distributed compositing does not move web execution to the endpoint.

- JavaScript/Wasm remain server-side.
- DOM state remains server-side.
- No script execution primitives exist client-side.

- No DOM reconstruction occurs.
- The client receives only graphics-oriented primitives required for rasterization and presentation.

The endpoint's attack surface remains within graphics and presentation layers—not web execution.

Engineering Sustainability

Chromium evolves continuously. RBI systems that deeply fork rendering or content logic accumulate merge debt and patch lag. Puffin confines architectural changes to two controlled boundaries:

- **IPC boundary (RemoteMojo)**
- **Compositor synchronization boundary (Distributed Compositor)**

This minimizes invasive modification of Blink and core renderer logic. Strategically, this enables:

- Faster upstream security patch adoption
- Higher compatibility with modern SaaS
- Lower long-term maintenance risk

Sustaining a browser fork over multiple years is often the dominant cost of ownership. This architecture is explicitly designed to avoid that trap.

Structural Performance Wins

Perceived performance in RBI is dominated by input-to-photon latency and animation smoothness under jitter.

Scroll

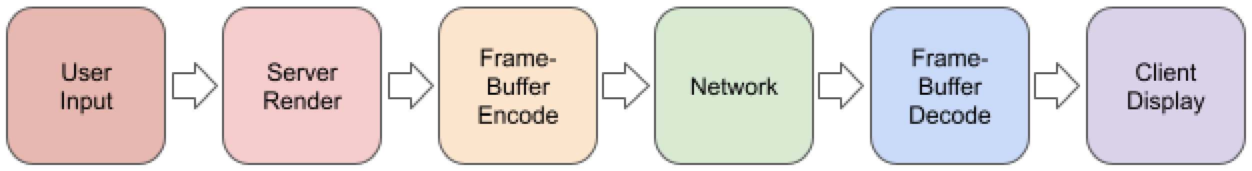
The client applies scroll deltas immediately, and the server computes authoritative evolution in parallel. Reconciliation occurs asynchronously. This removes one RTT from initial scroll response.

Compositor-Driven Animations

Eligible animations tick locally at display cadence. Network jitter does not directly degrade animation smoothness. The server remains authoritative and streams state for convergence.

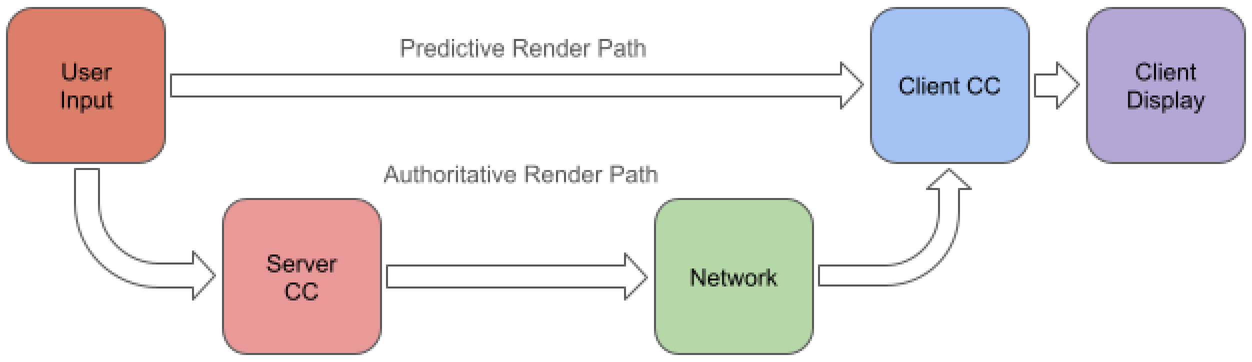
Performance Characteristics

Pixel-streaming pipelines require:



Distributed compositing eliminates:

- Video encode/decode overhead
- The requirement that every visual update be server-produced



Interactive operations that can be satisfied compositor-side are no longer RTT-bound for first visible response.

Compared to pixel streaming:

- Static regions are not retransmitted continuously.
- Many frames correspond to small state deltas.
- Transmission scales with state evolution rather than pixel count.

The structural advantage: **state deltas instead of continuous video**.

Competitive Comparison

Versus Video-Based RBI

Video-based RBI:

- Endpoint as terminal
- Always RTT + codec bound
- Continuous pixel transport

Distributed Compositor:

- Endpoint as real presentation engine
- Immediate interaction response
- State-driven transport

Versus DOM Reconstruction RBI

DOM reconstruction:

- Introduces compatibility risk
- Relies on sanitization correctness
- Re-implements portions of browser semantics

Distributed Compositor:

- Keeps Chromium authoritative
- Avoids client-side semantic reimplementation
- Preserves compatibility with evolving web API

Why This Is Hard

Chromium’s compositor was never designed for cross-machine semantics. It assumes:

- Rich in-process object graphs

- Shared memory data paths
- Tight coupling to scheduling and GPU presentation
- Deterministic frame lifecycle behavior

Distributing it without degrading determinism or maintainability requires solving:

1. Scheduler Coupling and Frame Lifecycle Preservation

BeginFrame dispatch, input sampling, animation ticks, frame submission, surface aggregation, and GPU present timing must remain coherent across machines.

2. Determinism Under Prediction

Property trees, animation timelines, and scroll deltas must converge identically. Failure produces visible jank, tearing, flicker, or snap-back artifacts.

3. Correct Delta Modeling Under WAN Conditions

Ordering, backpressure, and jitter tolerance must prevent subtle divergence.

4. Resource Identity and Lifetime Management

Fonts, images, raster inputs, and GPU resources require consistent identity mapping and invalidation semantics.

5. Visual Stability During Reconciliation

Prediction errors must converge without visible instability.

6. Continuous Upstream Evolution

Chromium’s compositor, scheduling model, and presentation pipeline evolve regularly. A distributed compositor must track these changes without becoming an unmaintainable fork.

This is multi-year systems engineering. It is not achievable through proxy adaptation or simple protocol design.

Strategic Value

The Distributed Compositor is infrastructure-grade browser IP. It represents an architectural reinterpretation of Chromium designed to:

- Preserve security isolation
- Deliver near-local interactivity
- Sustain long-term upgrade velocity

It is not a feature layer and not replaceable with streaming middleware.
It requires deep, invasive understanding of compositor lifecycle semantics, scheduling invariants, and deterministic convergence behavior.

Together with RemoteMojo, Puffin Secure Browser distributes Chromium itself—relocating risk without degrading experience and without reducing the endpoint to a terminal.