Metaverse
STANDARDS FORUM™

Open Metaverse Browser Initiative (OMBI)

An open-source project that brings spatial computing to the open standards-based internet

The Case for a Metaverse Browser Engine

omb.wiki/sneeze

W3C Immersive Web Discussion

New York, May 2026

The Spatial Web Needs Service Composition

Emerging spatial devices need what the web already does for documents

Computer vision and sensor fusion are enabling devices to understand physical context. Two categories of spatial web client are emerging:

- Human users: AR glasses, phones, headsets interacting with digital content anchored to real-world locations
- Autonomous systems: Vehicles, drones, robots consuming spatial services to navigate and operate in physical environments

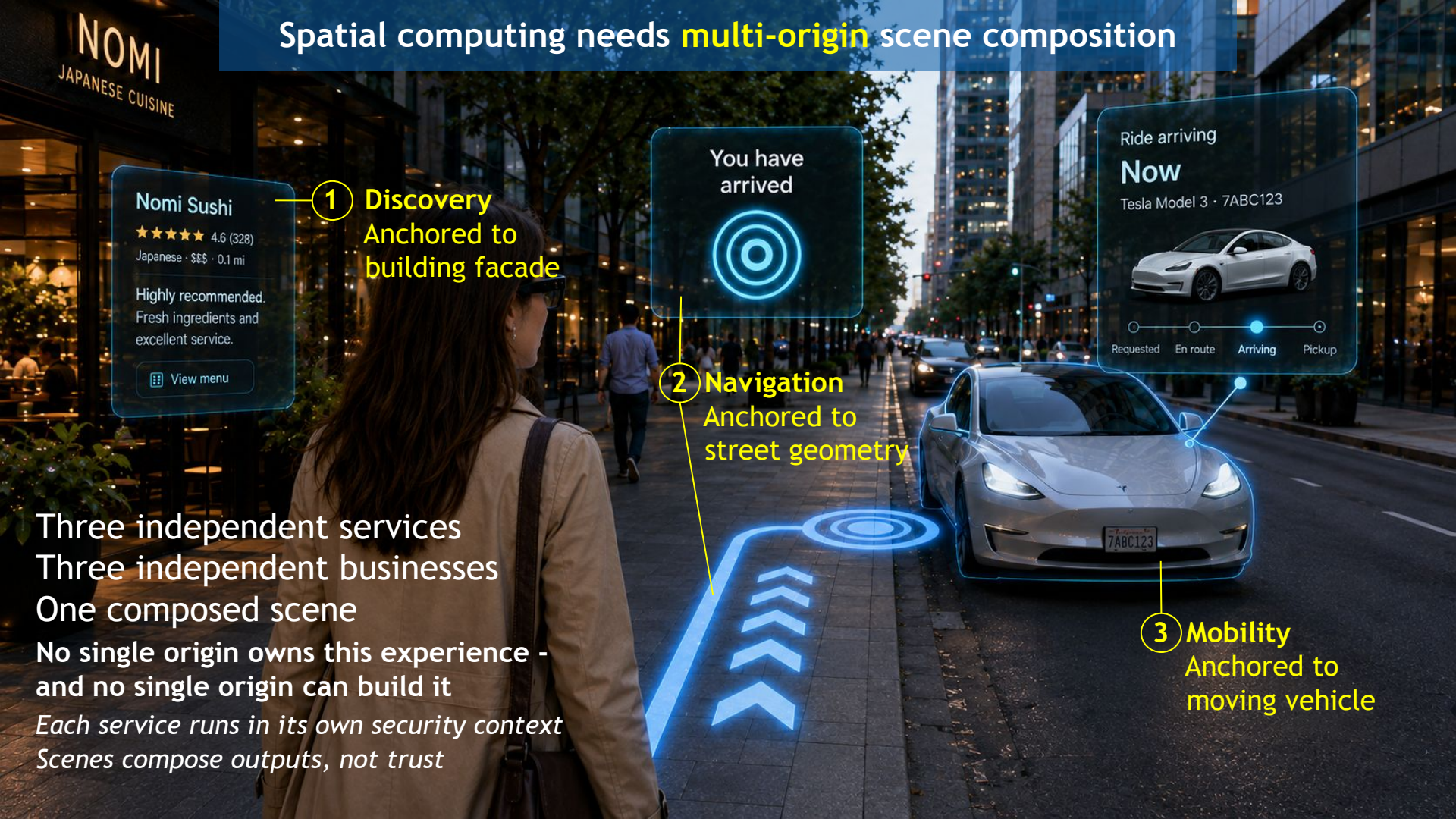
Both require the same thing: multiple independent, geolocated services from many origins, composed into a single 3D scene around them.

OMBI · A COLLABORATIVE OPEN-SOURCE TESTBED Vendor-neutral, hosted at the Metaverse Standards Forum - accessible to all

1. Build consensus on the need and use cases for spatial service composition
2. Explore deployment of spatial services using current web technologies
3. Identify where current web standards fall short and what new capabilities are needed

Smart glasses are a concrete starting point - a device category where emerging capabilities and interoperable standards can meaningfully intersect. *(Though spatial services should be deployable on any device to the extent of its capabilities.)*

Spatial computing needs multi-origin scene composition



Three independent services
Three independent businesses
One composed scene

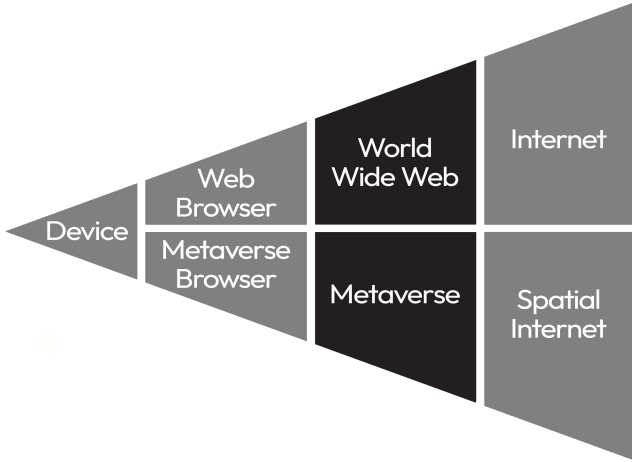
No single origin owns this experience - and no single origin can build it

Each service runs in its own security context
Scenes compose outputs, not trust

We set out to build a metaverse browser using a web browser

RP1 built a global prototype metaverse browser using WebXR.
It showed us where the opportunities and needs are to build new spatial capabilities

∞	**An endless map** spanning 1:1 map of Earth
100k+ CONCURRENT	**Solving for scale using 1000x less compute. Concurrent users with full 6DOF and spatial audio** using RP1's proprietary state-sync architecture. Scalable to billions of unsharded concurrent users
90 fps CO-PRESENCE	**Spatial audio + co-presence** across phone, tablet, desktop and VR headset in one shared scene.



The decision to prototype and discuss a METAVERSE BROWSER ENGINE is inspired and informed by the challenges encountered during this hands-on project

OMBI is Solving For These Requirements

- Companies/people need to be able to host their own content using the same hosting paradigm that they use to serve 2D web content to web browsers **This requirement says: closed walled garden platforms are not acceptable. This includes WebXR instances, which run within an individual origin that would be needed to "host" a metaverse session.**
- Companies/people have a need to publish 3D spatial content using the same delivery/display abstraction paradigm that they use to deliver 2D web content of web browsers **This requirement says: Content published for the metaverse is dropped onto a server and a browser will be responsible for transport and layout/display of the content. WebXR merely provides a surface to displaying pixels, but does not provide any form of "browsing" capabilities.**
- While the web browser can provide geospatial information to individual web pages upon request, the browser doesn't participate in the process of loading spatial content based on your location **This requirement says: Not only is the onus of VPS implementation placed on the web page to provide its own code, but any GPS data provided to an application is done so via a pull request initiated from the application. The use of this data is limited to the application's own design. Such knowledge is not used by the browser to bring other services (urls) into the scene as the users moves about.**
- Multi-origin services/content need to be able to share a single scene – with a level of trust and security that far exceeds the current web browser.**This requirement says: The browser's natural single-origin obsession makes any form of shared scene graph all but impossible. It's nearly [if not totally] impossible for a single company to share data between two highly correlated origins.**
- "10-Foot Rule" - People will not want to download a new, untrusted app every ten feet: **This requirement says: Solutions that attempt to merge independently downloaded apps still require users to download and install apps, which is antithetical to intended XR behavior.**
- The metaverse is expected to have a level of shared user co-presence unlike anything current available in the world wide web. **This requirement says: Thousands of AR & VR participants (combined) need to be co-present in a single space without publishers needing to provide their own individual solutions with their content.**

Foundational limitations of current web stack

Metaverse Requirement	Web/WebXR Browser Limitation
Need to flexibly mix geometry, lighting anchors, and behavior from **multiple services** into a **shared 3D scene graph** with robust cross-origin security and permissions and security.	The web has strong cross-origin embedding and isolation, but it does not define a browser-level model for **multiple origins** to contribute governed branches into **one shared spatial scene graph**. WebXR lets a page render XR content; it does not provide multi-origin spatial scene composition.
“10-Foot Rule” - People will not want to download a new, untrusted app every ten feet as they traverse the real world . Need **automatic geolocated service discovery**	The web has URLs, links, DNS, search, and geolocation APIs. What is missing is an interoperable **browser-mediated** lifecycle for **nearby spatial services**: discovery, authorization, loading, composition, and unloading as the user moves.
Geolocated anchoring with shared spatial reference and persistence	Geolocation gives geographic position data, and WebXR anchors can track poses in an XR session/runtime. There is not yet a widely deployed browser primitive for **persistent, shared, 6DoF geospatial anchors** that multiple independent services can agree on.
Services accessible by reference, with real-time state	Web transports can carry real-time data, but there is no **standard spatial service abstraction** for a URL-addressable **live 3D service** that exposes properties, events, actions, subscriptions, permissions, and scene-graph integration to a composing client.
Web access from immersive space. A user can log in to a new service, complete a payment, or grant a permission while remaining inside the immersive session	**No DOM overlays in immersive**: Immersive WebXR cannot be treated like a normal browser viewport. DOM exists, but DOM overlays, popups, cross-origin login dialogs and browser trust UI are constrained and inconsistently supported

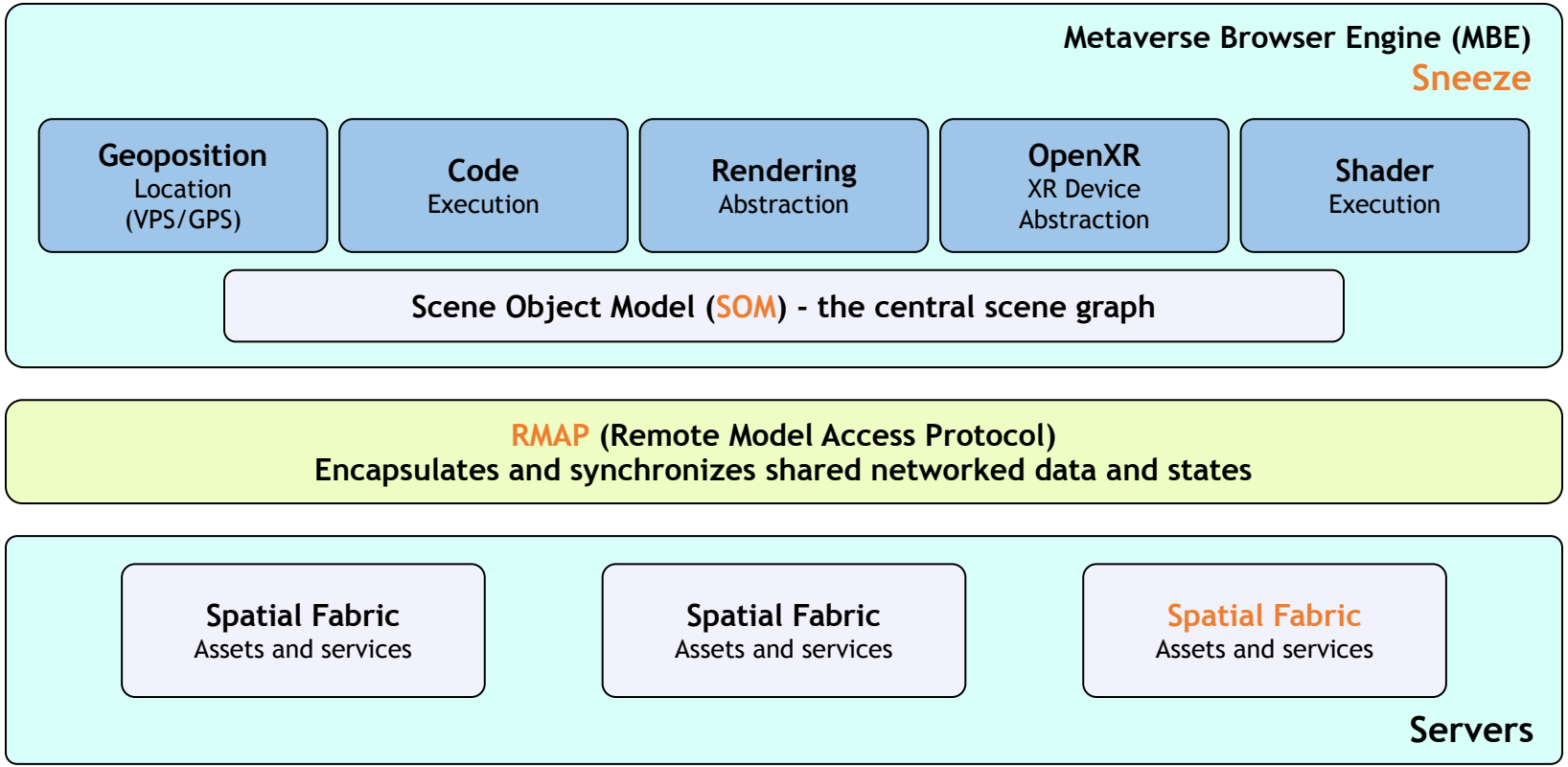
Foundational limitations of current web stack continued

Metaverse Requirement	Web/WebXR Browser Limitation
Multiple high-resolution services. A single session can compose several high-fidelity 3D services simultaneously — e.g. a virtual store, a 3D map, and a video room — each at desktop-class asset quality	**Restricted memory budget**: Standalone XR browsers have tight and device-specific memory budgets, limiting texture resolution, mesh complexity, video, buffers and asset streaming strategies
Increasing compute load. Real-time neural rendering (avatars, Gaussian splats, relighting) runs alongside scene composition and physics on a standalone headset	**Restricted compute capacity**: WebXR can use workers, but rendering and device integration are less thread-controllable than native engines
Consistent frame rate. Device-native frame rate (72/90/120 Hz) need to be maintained for the full session across all composited content, including under thermal load	**Unreliable framerates**: WebXR frame timing has softer guarantees than native XR; GC, browser scheduling, WebGL driver behaviour and thermal limits can produce hitches
Low latency. User body pose, voice, and world state needs synchronization at sub-50 ms latency, tolerating packet loss rather than stalling on retransmits	**Missing "speed-first" datagram protocol**: There is no raw UDP, but WebTransport provides browser-safe unreliable datagrams where supported
App portability. An AR service authored once should runsidentically on Quest 3, Vision Pro, Snapdragon Spaces glasses, Android (ARCore) and iOS (ARKit), with consistent camera, depth, anchor and VPS access	**Immersive-AR gaps**: WebXR AR exists, but camera/depth/scene/VPS access is fragmented across platforms, making "write once" immersive AR unrealistic today

The Metaverse Browser Engine brings spatial computing to the web

Built on open standards for multi-origin compositing, geoposition awareness, self-hosting and hybrid copresence

The metaverse “big picture”



What is Sneeze?

Sneeze is a Metaverse Browser Engine for the metaverse stack. (like WebKit or Blink for the web stack)

What it does:

Sneeze processes 3D spatial content through a Scene Object Model (SOM) - a 3D scene graph (instead of processing HTML/CSS/JavaScript through a DOM like a web browser engine).

Function	Web browser engine	Sneeze spatial engine
Content ingestion	Fetch HTML/CSS from servers	Receive 3D models, shaders from services
Scene composition	Build the DOM	Build the Scene Object Model (SOM)
Rendering	Layout & paint	Delegate to abstracted 3D renderer
Script execution	JavaScript (V8)	WebAssembly sandboxed runtime
Input handling	Mouse/keyboard	Spatial input via OpenXR

Language: C++

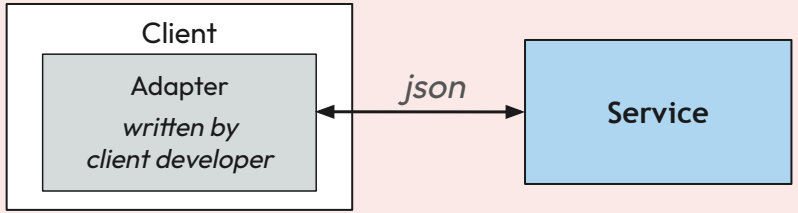
Open-sourced: March 2026, contributed to the Open Metaverse Browser Initiative (OMBI) under the **Metaverse Standards Forum**, where it serves as an exemplar MBE implementation

Rendering: Does not implement its own renderer - delegates through a graphics abstraction layer

Remote Model Access Protocol (RMAP)

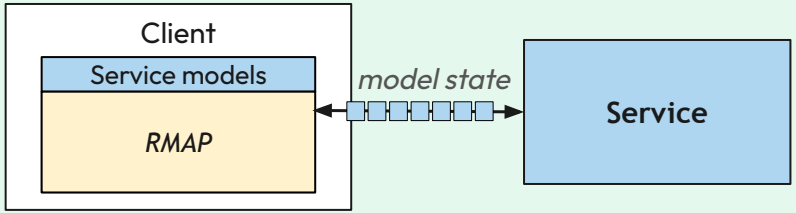
A protocol for synchronizing the state of discoverable models using any preferred network transport

Legacy – REST · GraphQL · gRPC · WebSockets · MQTT



Client developer writes integration code

RMAP



Service ships integration code

The current model

- Every existing service protocol – REST, GraphQL, gRPC, WebSockets, MQTT – publishes an API and asks **every client developer to write the integration code**
- Integration cost is per-client, repeated millions of times, and rises sharply for any non-trivial protocol
- There is **no one standard way** to implement real-time services, which are notoriously difficult to implement and document.

The inversion

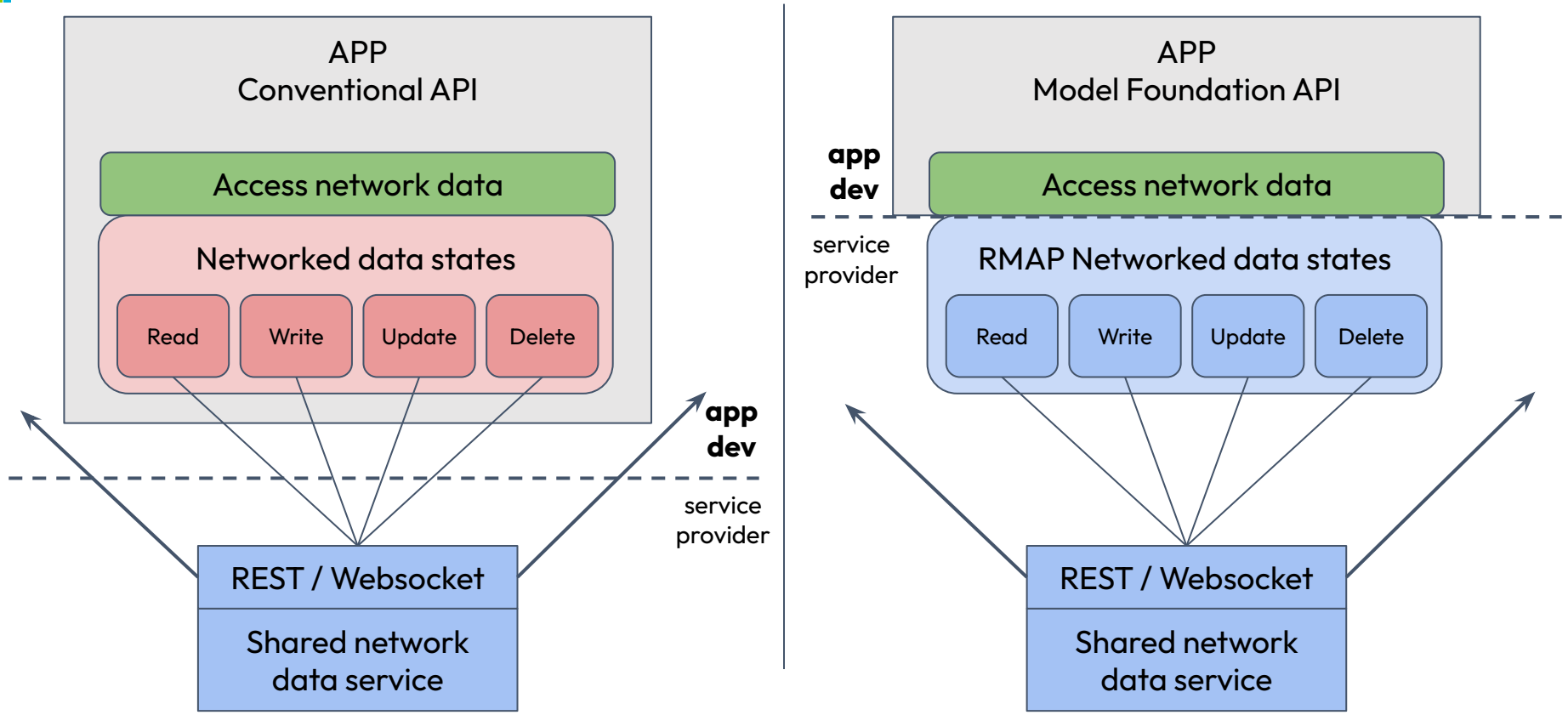
- Services publish RMAP typed model classes that run inside the WASM store alongside client code.
- **Client interacts only with models**, obfuscated from the programmatic details of the service.
- Models are inherently discoverable. One uniform action/event mechanism, queryable for any model. **Programmer can use an unfamiliar service without docs**.

What flows from it

- RMAP works on any software platform, not just metaverse browsers. Can be ported to any language or platform. **Real-time services are easier to use** than current stateless services.
- CORS sidestepped – no cross-origin fetch surface left to police.
- Bug fixes propagate via service updates without requiring client-developer rebuilds.
- Service can switch wire protocols overnight without breaking the client.

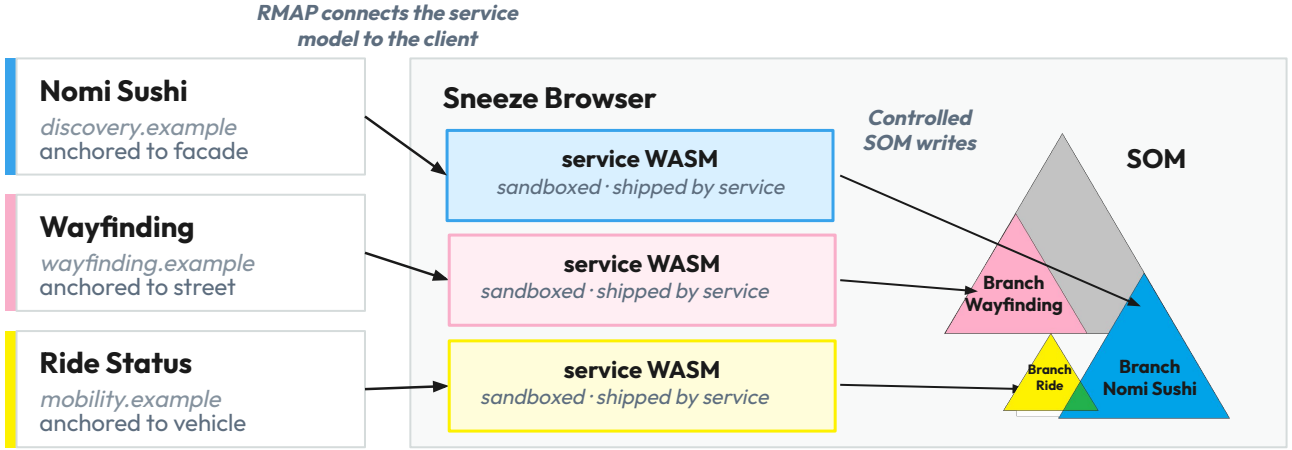
Runtime-agnostic · Additive over REST · No close prior art

RMAP encapsulates and synchronizes shared networked data and states



SOM in OMBI – Multi-Origin Composition

How spatial services deliver state into a shared scene



01 · THE PROBLEM

Three services, three providers, one scene

The SOM composes content from many independent services. Nomi Sushi, the wayfinding arrows, and the ride status each comes from a different operator at a different origin. Service count scales to dozens or hundreds in dense urban contexts.

02 · HOW RMAP FITS

Service ships adapter/model definition

Each service ships its WASM program; Sneeze loads it sandboxed on first connection. RMAP delivers model updates to the WASM application via model updates; the application writes only into its owned SOM branch. RMAP and SOM stay architecturally independent - the layering is clean by design .

03 · WHAT THIS ENABLES

One browser-composed scene

No per-service install. No per-service client integration. Each provider can evolve its backend protocol independently and is safely sandboxed within a WASM store. Services can read from any unprotected portion of the SOM for contextual awareness. Write branch ownership and service identity prevent cross-service scene mutation

RMAP standardizes service model access; WASM isolates service logic; SOM branch ownership composes independent providers into one scene

WASM advantages for the Metaverse Browser

1. Sandboxed execution for third-party service logic with strong per-service isolation
2. No JavaScript dependency required in the native browser engine; WASM modules can run directly in an embedded native runtime
3. Language flexibility: services can be authored in C, C++, Rust, and other languages that compile to .wasm
4. Performance and control: supports CPU budgeting, interruption, and independent execution per service
5. Portable execution model aligned with multi-platform, standards-based architecture

WASM vs Javascript

1. WASM is a better sandbox than JavaScript for running many third-party services safely inside the browser engine.
2. Sneeze is designed around a native C++ scene model, not a JavaScript runtime.
3. JavaScript is less suited to predictable real-time engine workloads than native code plus WASM.

Established WASM Languages

C++
Rust
Go
Zig
C#
Python
Javascript (AssemblyScript)

Emerging

MoonBit
Grain
Kotlin
Swift

Certificate-based security for multi-origin content

The Web Problem: The web's trust model is tied to origins -- usually DNS-based domains plus scheme and port. Pages can load cross-origin resources, but active cross-origin access, storage, and permissions are largely origin-scoped, with exceptions like CORS layered on top. The result is that trust and application boundaries are coupled to web infrastructure rather than portable cryptographic identity.

The Metaverse Solution: A spatial fabric is able to host content from multiple independent origins simultaneously. Trust is anchored to organization-based cryptographic identity, not DNS.

- A metaverse spatial fabric (MSF) file contains a JWS-signed JSON structure that declares which services participate in a fabric, with the full X.509 certificate chain embedded in the signature
- Organization identity = SHA-256 of the signing certificate's public key - survives certificate renewal, is deterministic, and is independently verifiable by any party
- Grouped services runs in a WASM store scoped to the organization's fingerprint.
- WASM Stores provide for isolated memory, CPU budgets, controlled host API access
- Persistent storage is scoped to both the individual fabric and to the fabric's organization. Organization A cannot read Organization B's data
- The browser verifies the signature and certificate chain before trusting any content.
- Users can filter out organizations, untrusted or unverified entities
- Standardized content tags (ie. advertisement, adult, game, POS, navigation, safety, entertainment, etc) allows users to filter out content based on content type.

Result: Multiple origins coexist by design, each cryptographically isolated. The fabric publisher signs who participates. The browser enforces the boundaries. No CORS. No domain coupling. Trust is portable and verifiable.

What is a Spatial Fabric (evolution of a website) ?

What is a Spatial Fabric?

1. A shared 3D coordinate space where services and content live, like a website for the metaverse
2. Can be public or private, and multiple fabrics can be connected into one browser session
3. RP1's default fabric is a 1:1 digital twin of Earth and the solar system

Benefits of Spatial Fabric

1. **Composable world model:** multiple services contribute to one shared scene instead of one closed app owning everything
2. **Decentralized hosting**: organizations run their own services and keep control of data and monetization
3. **Real-time interaction**: persistent connections and live state updates support shared, synchronized experiences
4. **Open service integration**: many independent services can plug into the same space through a standardized model/API layer
5. **Massive scale**: designed to support very large spatial datasets and many concurrent users

Sneeze provides geopositioning and location awareness

The Metaverse Browser Engine architecture provides a spatial context so relevant services can be automatically activated based on proximity

- **VPS grounding**:
  Sneeze combines GPS/VPS with OpenXR tracking to know where the user is and anchor the scene to the physical world.
- **Spatial awareness**:
  It keeps a live Scene Object Model (SOM) of nearby fabrics, services, and objects, so content can appear based on location and proximity.
- **Real-time adaptation**:
  As the user moves, Sneeze updates the scene continuously through proximity queries, streaming, and coordinate transforms.
- **Open architecture**:
  VPS is treated as a pluggable positioning layer, so different environments can provide their own localization backends.
- **Privacy by design**:
  Location and input are browser-mediated, with fabric-level ownership of VPS data and permission controls around sensitive signals.

OMBI Spatially Located Services: Where are we?

USE CASE
Sneeze + Spatial Fabric service discovery

User opens Sneeze in spatial context (by specifying a **primary fabric**)

GPS/VPS provide continuous position information to Sneeze, which locates within the SOM

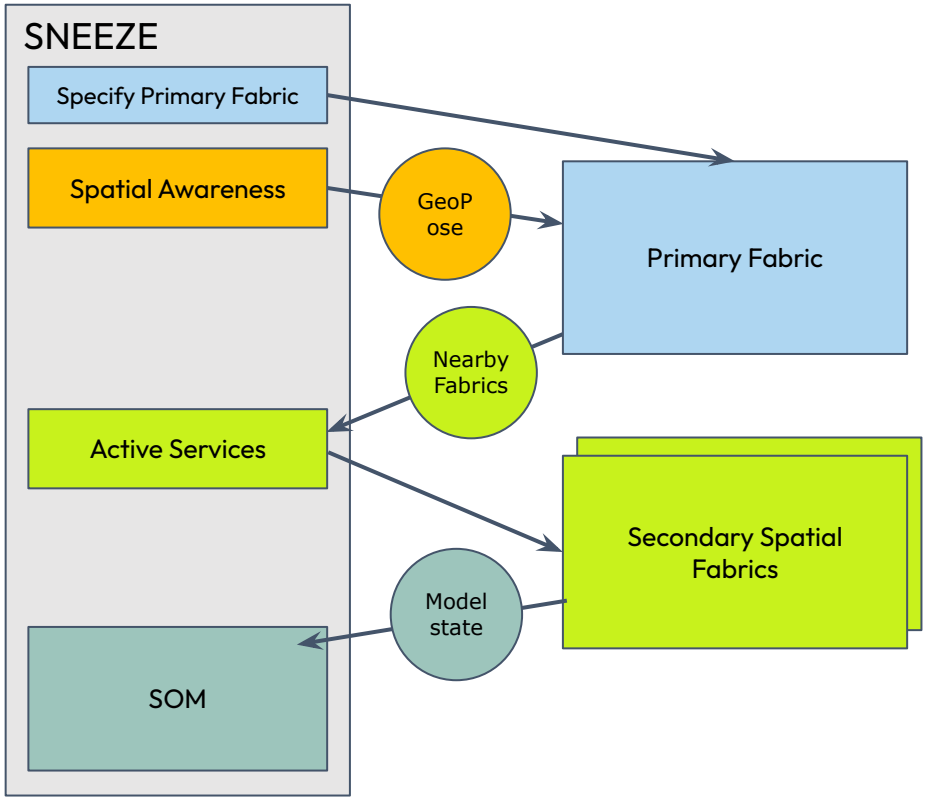
Sneeze fetches proximal map nodes (via RMAP) from loaded fabrics' map services

Incoming map nodes identify secondary spatial fabrics/services, which are loaded into the SOM

Map nodes that become too far away are unloaded from the SOM

SOM composes the multi-origin scene

OUTCOME Place-aware services rendered, anchored to the user's physical surroundings



Combining the web and spatial scenes

Methods of compositing Blink and Sneeze outputs

Four Ways to Compose a spatial engine (Sneeze) with a web engine (Blink)

A Full-Viewport Handoff

LIKE *Like opening a PDF, entering a WebXR immersive session, or invoking the proposed W3C <model> element*

MECHANISM Browser switches between Blink-rendered web pages and Sneeze-rendered spatial fabrics

USE CASE Fastest path to open a spatial fabric from the web - similar to opening a PDF or immersive document

LIMITATIONS No simultaneous composition; user is either in a web page or a Sneeze scene. Requires navigation, permissions, identity, and lifecycle handoff

B Sneeze Island Inside a Web Page

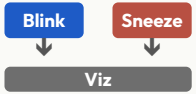
LIKE *Like <iframe>, <canvas>, or a Sketchfab / Google Maps embed - a sandboxed rectangle for 3D*

MECHANISM Blink embeds an opaque Sneeze viewport as a rectangular region inside a normal web page

USE CASE Lets web developers embed live 3D spatial services, fabrics, previews, or mini-worlds in existing websites

LIMITATIONS The 3D scene is boxed inside the page; Blink does not directly access or manipulate the SOM

C Shared Compositor / Sibling Surfaces



LIKE *Like Chrome's Viz combining Blink with browser UI - or an OS window compositor*

MECHANISM Blink and Sneeze each render independent surfaces that are combined by a shared compositor (Viz in Chromium; OS-level on visionOS/Android)

USE CASE Useful for browser chrome, dialogs, permission prompts, side panels, overlays, and mixed UI around a spatial scene

LIMITATIONS Simple rectangular composition only. Does not by itself create true 3D integration unless depth, pose, hit testing, and occlusion are explicitly shared

D Web Surface Inside the Sneeze 3D Scene

LIKE *Like Apple visionOS rendering Safari panels in 3D space, or WebView inside a native AR app*

MECHANISM Blink renders live web content to a surface/texture that Sneeze places as an object in the SOM

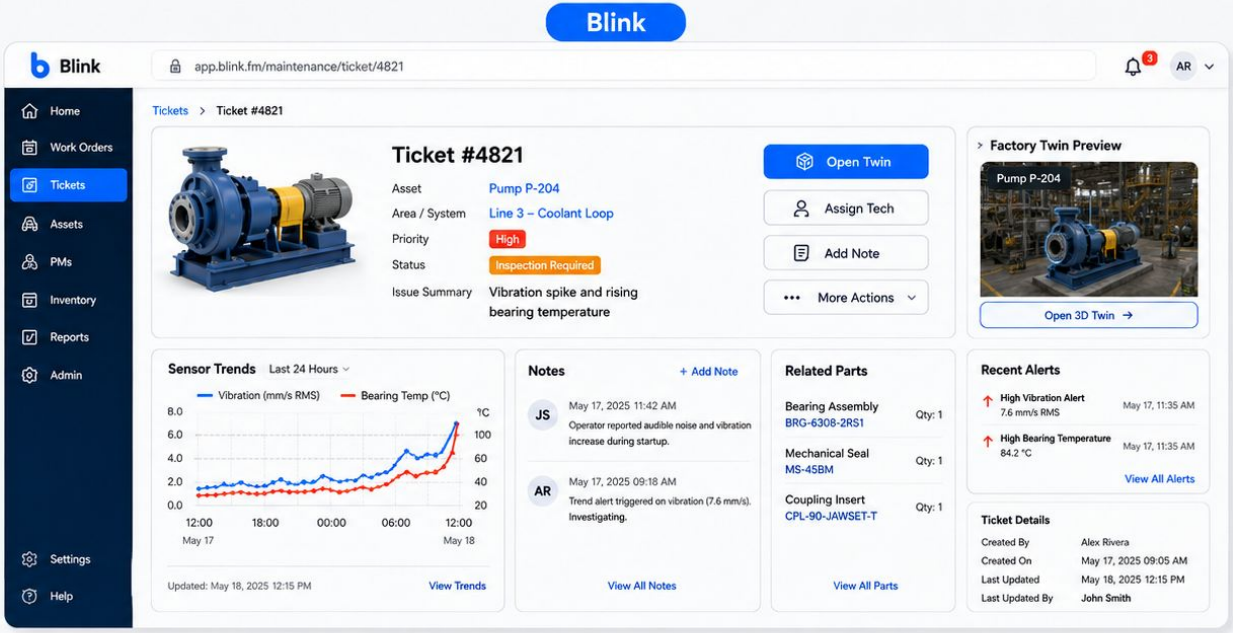
USE CASE Web panels, dashboards, forms, commerce flows, maps, documents, and controls embedded into a true 3D spatial scene with perspective, transforms, occlusion, and lighting

LIMITATIONS Requires careful input routing, focus management, permissions, depth/occlusion handling, and security isolation. Initially likely treats the web page as a rectangle, not individual DOM nodes as SOM objects

Needs more than Viz Composition

1.1 Web Engine standalone view

A standard web workflow captures the maintenance issue before any 3D view is opened



Mechanism

A maintenance issue is recorded and managed in a standard Blink web ticket.

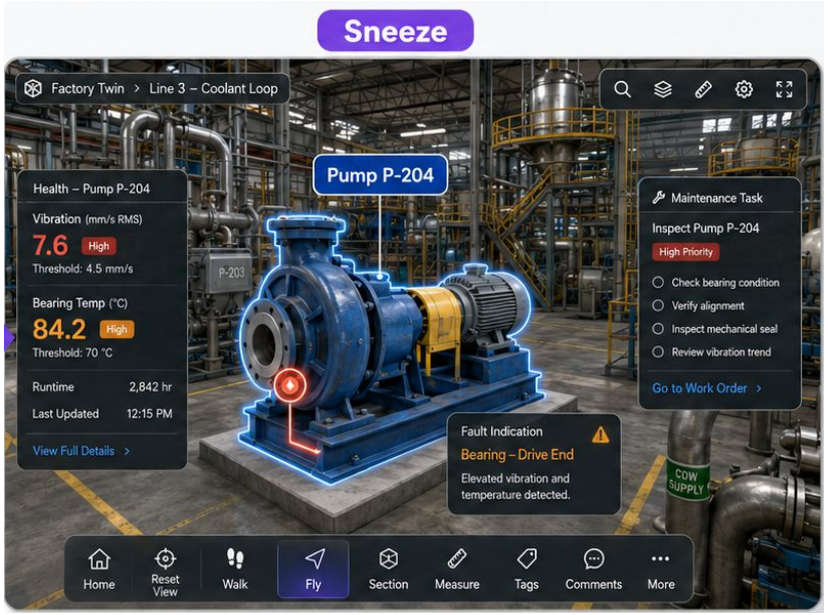
Use Case

Supervisors and technicians can review status, logs, parts, and sensor trends in a familiar 2D workflow.

Limitation

The ticket describes the machine problem, but not its physical context or spatial access constraints.

1.2 Spatial Engine standalone view



Why 3D Helps

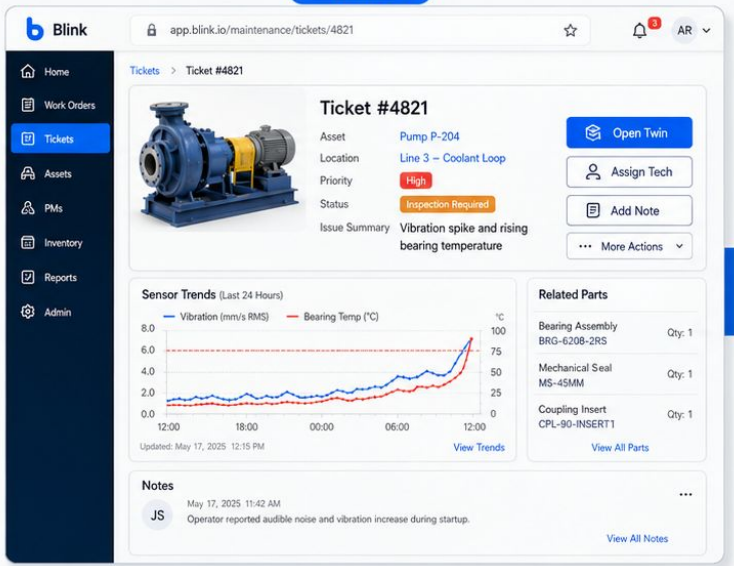
The 3D twin reveals physical context, surrounding equipment, access constraints, and the fault location.

2. Full-Viewport Handoff to the Factory Twin

A 2D maintenance ticket opens the corresponding 3D factory digital twin

A

Blink



Handoff

Sneeze

Mechanism

The browser switches from a 2D Blink maintenance ticket to a full Sneeze factory digital twin view.

Use Case

Lets a technician move instantly from a ticket record to spatial understanding of the affected machine.

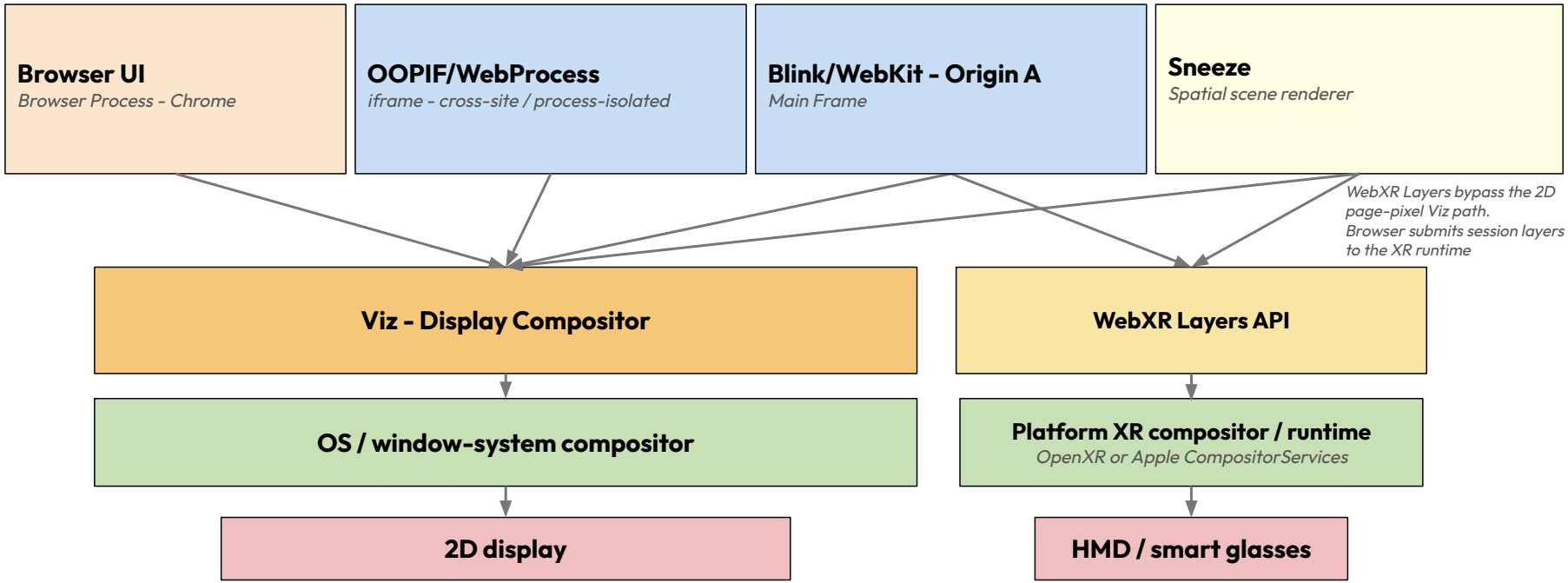
Why 3D Helps

The 3D twin reveals physical context, surrounding equipment, access constraints, and the fault location.

Adding Sneeze to the Web Stack (interim)

Cross-site content may render in separate processes - Viz is the shared browser composition stage for page pixels

B C



The Web is not a shared-scene composition model *- web security, process isolation where used, and browser-process checks prevent unauthorized DOM, resource, GPU resource, and script-readable pixel access across origins*

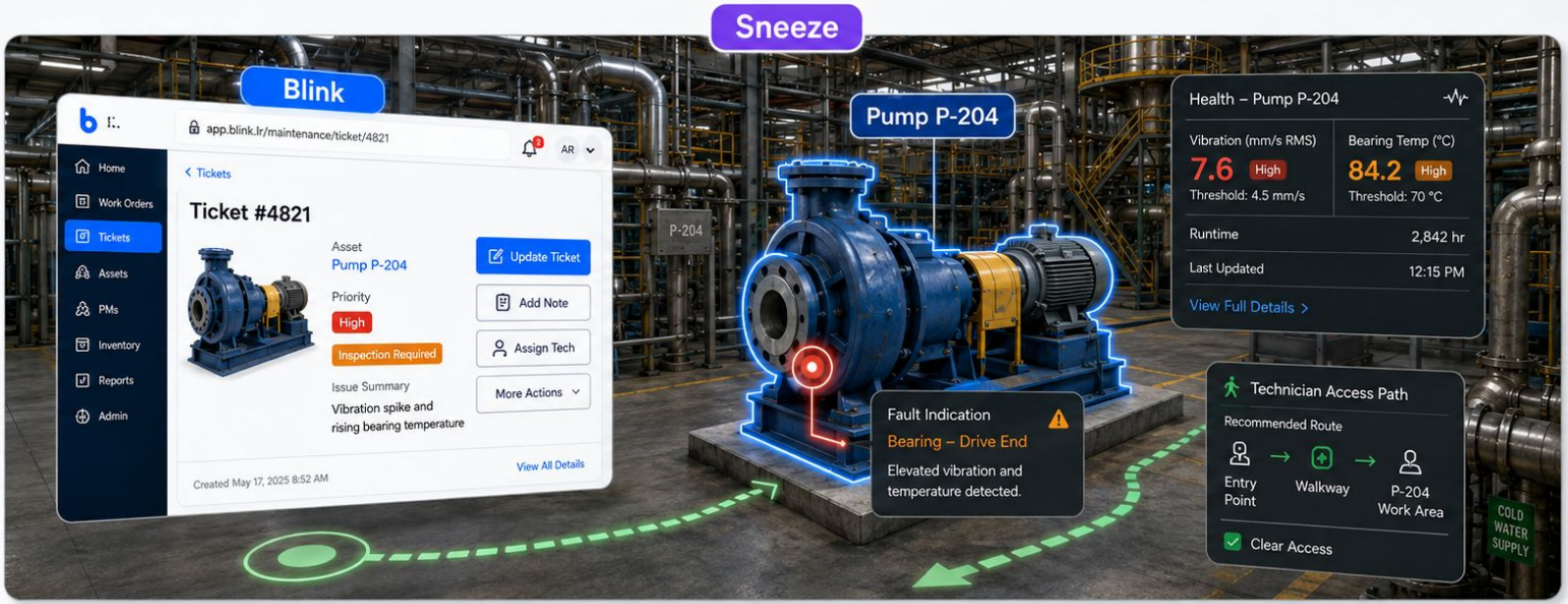
Viz aggregates compositor frames *- surfaces/draw-quads, not as a shared DOM, WebGL/WebGPU, or 3D scene graph*

Immersive XR is single-session *- only one visible immersive XR session owns the XR device display; other origins do not co-compose into that session*

3. Ticket Embedded in the 3D Scene

D

The 2D work order remains available while the machine is examined in spatial context.



Mechanism

Blink renders the maintenance ticket as a live web surface that Sneeze places in the spatial scene.

Use Case

A technician can read and update the work order while viewing the actual machine and nearby equipment.

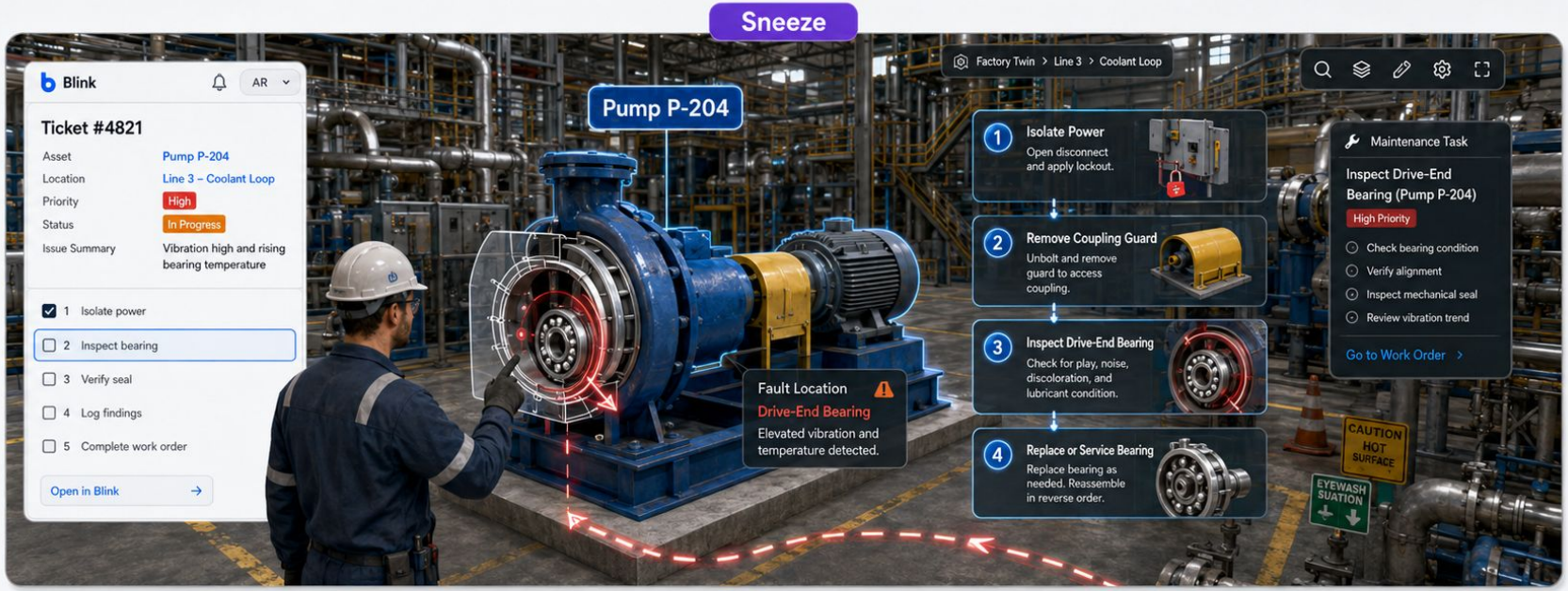
Why It Matters

The browser links recordkeeping and spatial understanding without forcing the user to switch mental context.

4. Spatial Guided Repair Workflow

The factory twin expresses the maintenance task more clearly than the ticket alone

D



Mechanism

Sneeze uses the ticket data to drive step-by-step spatial guidance around the actual machine.

Use Case

Technicians can inspect, repair, and confirm work while remaining grounded in the real factory layout.

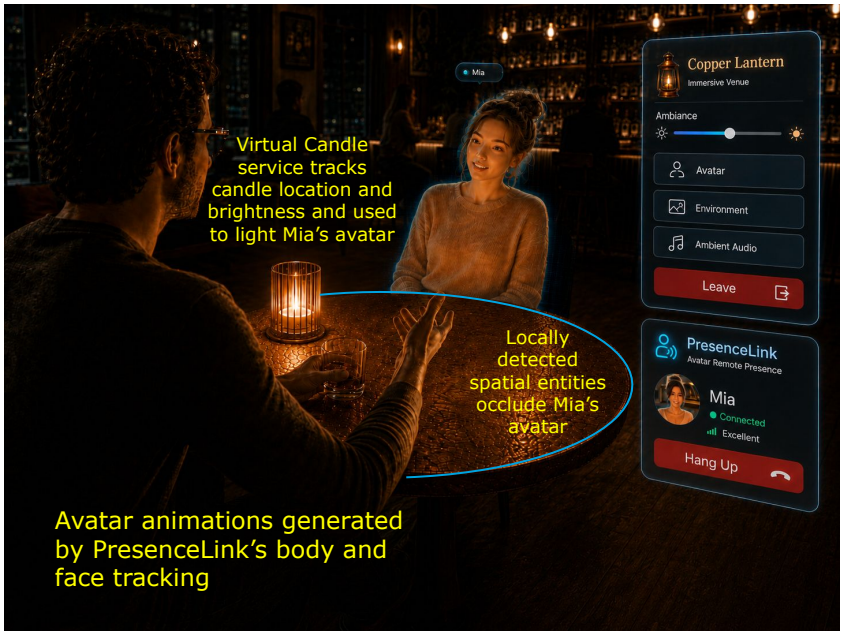
Why 3D Wins

The twin shows fault location, access path, nearby hazards, and repair sequence far more clearly than a flat ticket.

Virtual Presence Example

Mia and Michael are connecting using ‘PresenceLink’ service
The Copper Lantern Bar provides a environmental digital twin service
Mia has selected a higher quality avatar than Michael :)

Michael is in the **real-life** bar using **AR**



Mia is at home using **immersive VR**

Virtual Candle service is an example of cross-origin rendering interaction

Metaverse
STANDARDS FORUM™

OMBI Project Organization

Prototyping Progress

Metaverse Browser Initiative at MSF

Open Source, Open Governance

Open-source project to create ‘Sneeze’ Browser Engine

OSS best practices and permissive Apache 2.0 license

Testbed / exemplar implementation to engage industry and multiple standards organizations

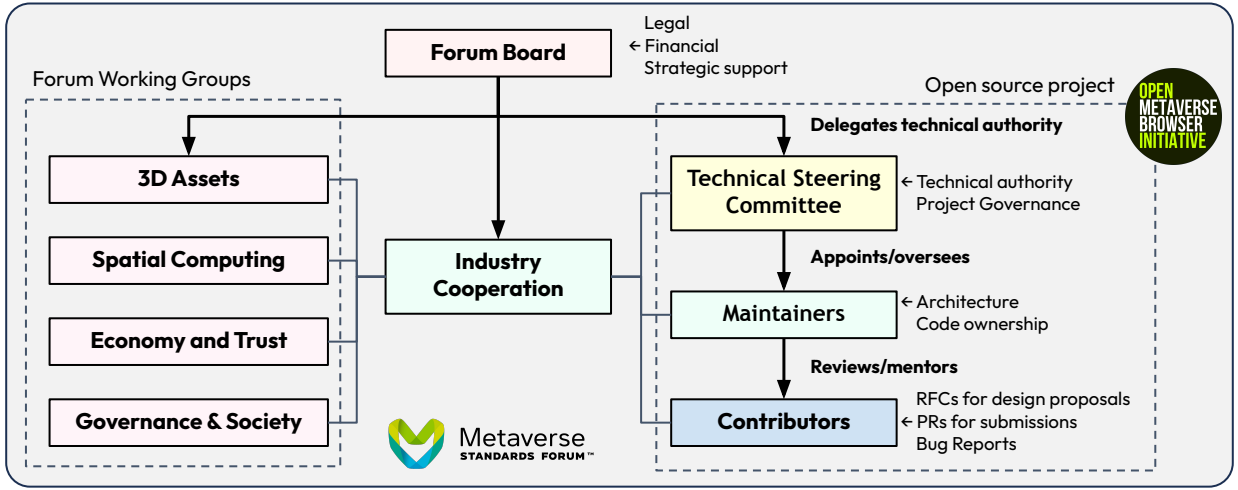
Domain Experts in Forum Working Groups
Provide open source project input guidance and may use OMBI as a testbed. For example:

3D Assets
Avatar formats, including scalability for 1000s of avatars in a scene

Spatial Computing
Portable access to visual positioning systems, indoor positioning, geolocation

Economy and Trust
Identity and transactions, Universal Manifest for persona and possession portability

Governance and Society
EU Digital Identity Wallet (EUDI) alignment under eIDAS 2.0, privacy and child protection compliance



Forum members may participate in both Forum working groups and open-source project

Standalone metaverse browser that runs on XR devices

RP1 is building **APOLLO**, a general-purpose metaverse browser: **a native browser** for spatial computing that connects to **real-time services** and **composes** them into a **shared 3D scene**. It is a browser for the spatial web -- connecting users to many independently hosted spatial services through **open standards**. Not a single-origin app nor a closed virtual world.

- **Native metaverse browser** for AR, VR, desktop, and mobile use cases
- Sneeze engine for **scene composition**, rendering management, input handling, and service connectivity
- Scene Object Model (SOM) for **combining content from multiple independent services** into one shared 3D scene
- **Decentralized hosting** so organizations can run their own spatial services and keep control of data and monetization
- **Real-time API / service integration** for AI, payments, games, enterprise tools, and other third-party services
- **1:1 spatial map of Earth and the solar system** as a universal spatial index for attaching environments and services
- **Open standards stack** (built around standards like glTF, OpenXR, ANARI, WebAssembly, and SPIR-V)
- **Sandboxed** third-party logic using **WASM modules**
- **Many simultaneous persistent connections** instead of the one-page-at-a-time web model
- **Cross-platform**, future-oriented architecture intended as an **open browser mode**l rather than a closed world platform

Project Apollo Demo in 4 weeks

Metaverse Browser Using Sneeze at AWE 2026

- Scene consisting of hierarchical (multi-origin) spatial fabrics
- Certificate-based security model
- WASM instances able to manipulate objects in the SOM
- Inspector for Network, Storage, Console
- Universal map browsing (universe -> ground)
- Audio
- Avatar placeholders
- RMAP-based services

Find Out More! Get Involved!

OPEN METAVERSE BROWSER INITIATIVE

Metaverse STANDARDS FORUM™

https://omb.wiki/

Metaverse
STANDARDS FORUM™

Appendices

Geolocation: Collaborative Actions for Interoperability

GOAL *an efficient cross-platform geospatial pose and anchoring interface for WebXR and other web user agents*

TODAY
Current geolocation standards and platforms

T1 **Geolocation API** W3C DAS + WebApps
W3C Rec 2022; CR Snapshot 2026; widely shipped
Lat / lon / optional alt, accuracy, speed, heading/course - no 6DoF pose or XR-frame sync

T2 **WebXR Device API** W3C Immersive Web WG
Candidate Recommendation Draft on Recommendation track
Continuous XR poses in local reference spaces - no standard geospatial/world CRS binding

T3 **WebXR geo-alignment / geospatial reference space** Immersive web incubation
Incubation repo; needs renewed championing
Would align WebXR local reference spaces to geospatial/world frames

T4 **GeoPose 1.0 Data Exchange Standard** OGC
Approved 2022; published 2023; OGC Implementation Standard
Location + orientation of real/virtual objects in Earth-anchored reference frames

T5 **ARKit ARGeoAnchor** Apple
Shipping on supported Apple platforms; "ARGeoTracking / ARGeoAnchor / Location Anchors backed
Geographic anchors using lat/lon/alt; Apple-specific

T6 **ARCore Geospatial API** Google
Shipping in ARCore; Android/NDK, Unity, iOS, Unreal paths; Google VPS / mapped geospatial data backed
Geospatial anchors and device pose using Google VPS / mapped geospatial data

POTENTIAL COLLABORATIVE ACTIONS
Cross-standards-body progress

C1 **Web geospatial pose API**
BUILDS ON T1 + T4 W3C DAS + WebApps; liaise Immersive Web
Extend Geolocation toward continuous geospatial pose - for any web client; requires orientation/device-pose integration, timing guarantees, permissions/privacy, and a GeoPose-compatible profile

C2 **Advance IWCG geo-alignment / geospatial XR Reference Space**
BUILDS ON T2 + T3 W3C Immersive Web
Surface XR poses through a geospatially aligned reference space to WebXR clients. Use OMBI scenarios as motivating requirements

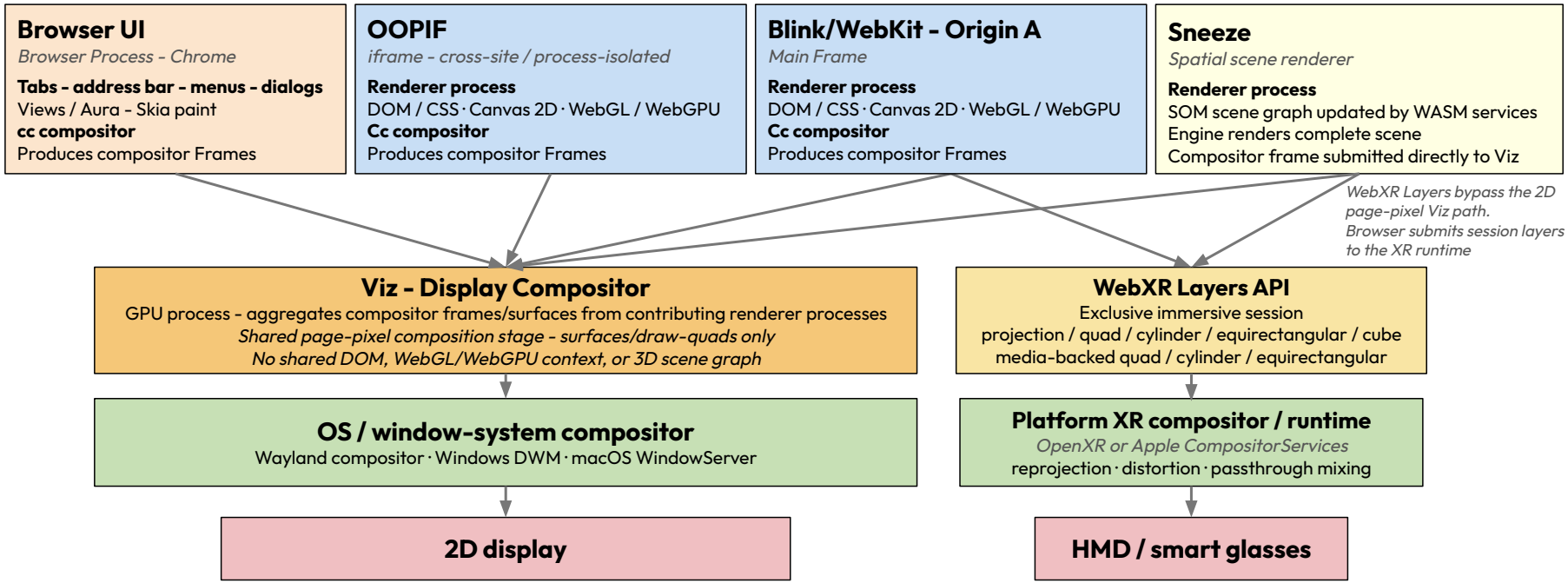
C3 **Adopt GeoPose profile as serialization**
BUILDS ON T3 + T4 W3C IWCG + OGC
Standardize pose format on the wire between client, runtime/OS, and Spatial Fabric. Constrained profile may be needed for privacy, accuracy/confidence metadata, axis conventions, CRS identifiers, and efficient runtime transport

C4 **Explore OpenXR geospatial reference-space / CRS binding extension**
BUILDS ON OpenXR + OGC CRS/GeoPose; informed by T5 + T6 - Khronos
Defines opt-in hooks tying OpenXR spaces to OGC CRS/GeoPose with accuracy/confidence metadata; needs champion + runtime support

Missing today: a standard 6DoF geospatial pose/anchor interface that bridges web permissions, XR reference spaces, and OGC CRS/GeoPose semantics

Adding Sneeze to the Web Stack (interim) details

Cross-site content may render in separate processes - Viz is the shared browser composition stage for page pixels



The Web is not a shared-scene composition model *- web security, process isolation where used, and browser-process checks prevent unauthorized DOM, resource, GPU resource, and script-readable pixel access across origins*

Viz aggregates compositor frames *- surfaces/draw-quads, not as a shared DOM, WebGL/WebGPU, or 3D scene graph*

Immersive XR is single-session *- only one visible immersive XR session owns the XR device display; other origins do not co-compose into that session*