

RESEARCH PROJECT

Synchronizing Remote-Calculated Baked Static Lighting at Runtime to Mobile XR Headsets in Unity

— a system of baking a server-authoritative runtime-scene, distributing and applying them on clients.



Table of Content

— Introduction —

Goal & Motivation
Technical Context & Constraints
Requirements & Summary

— System Design & Implementation —

Architecture
Configuration & Challenges
Workflow (Measurement Example)

— Requirement Fulfilment Analysis —

— Future Work & Present Problems —

— Summary —

— Questions —

— Introduction —

Goal & Motivation

Technical Context & Constraints

Requirements & Summary

Introduction

— Goal & Motivation —

Technical Context & Constraints

Requirements & Summary

Goal:

“Investigate and develop a method for modifying baked Global Illumination (GI) at runtime within the Unity Engine, while maintaining performance and memory efficiency on Android mobile devices.”

Why?

- Consumer HMDs → high fidelity expectations
- Mobile SoC*s can't run GI solvers in real-time → baking only
- Baked lighting is by design static → no standard “runtime sync” solution

* SoC = System on a chip; integrated processor in devices like the Meta Quest 3

Introduction

Goal & Motivation

— Technical Context & Constraints —

Requirements & Summary

Technical Context & Constraints

Unity 6000.3f (Editor & Engine)

- Only Baked GI feasible
(lots of dynamics lights may require high computing times)
 - Baking results are not directly exportable
(lightmaps as .exr or .png; light probes as SH coeff., ...)
 - Editor-only
(unavailable at runtime or builds)
 - Lighting is exchangeable
- Baking at “runtime” only in editor in “Edit Mode”*

Android (HMD)

- Strict thermal & compute budgets
 - 13.3 ms (75 Hz) frame budget
- Baking at runtime not feasible on-device

* “Opposite” of “Play Mode”, when running the scene. The mode when editing the scene.

Technical Context & Constraints — Approach

- Server = middleware
- Server authorizes light changes (static lights) and synchronization
- Unity Editor Instance to bake GI
- Server distributes to clients

Unity Editor Instance <> Server <> Clients

Introduction

Goal & Motivation

Technical Context & Constraints

— Requirements & Summary —

Requirements

Hard	Soft
Core functionality	Usability
Unity-exclusive	Security
Server authoritative	
Negligible introduced system latency	
Reliability	
Performance	

Table of requirements

Summary

- Baked GI (lightmaps, light probes, static lights)
- Changeable at runtime
- Server = middleware; Unity Editor = GI Baking
- Latency heavily depends on baking and distribution time
- Mostly functional requirements

— System Design & Implementation —

Architecture

Configuration & Challenges

Workflow (Usage Example)

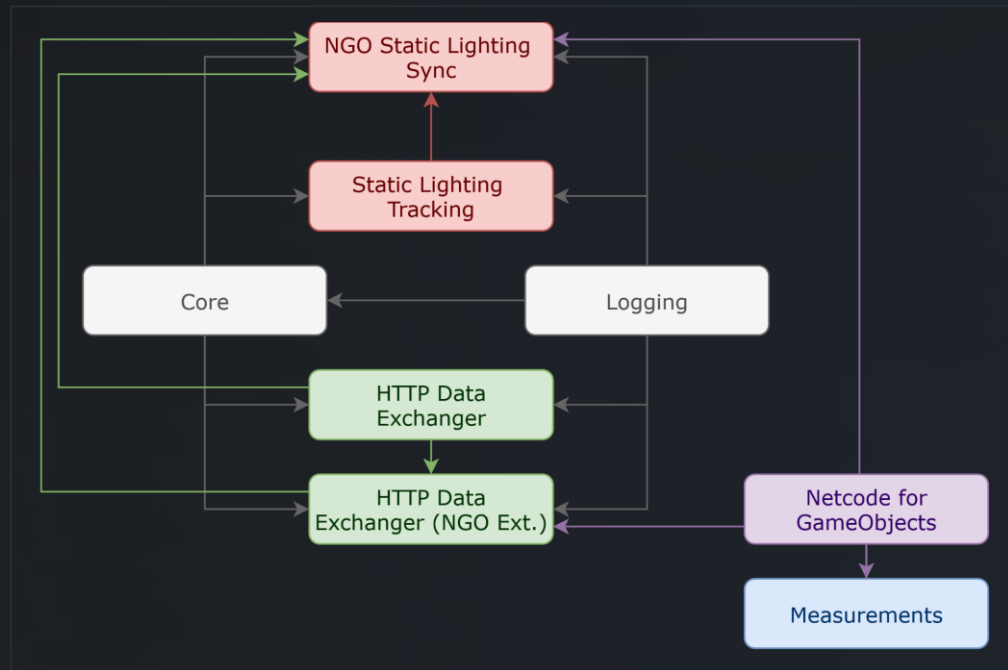
System Design & Implementation

— Architecture —

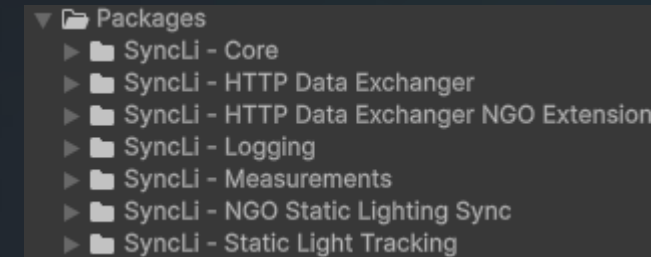
Configuration & Challenges

Workflow (Measurement Example)

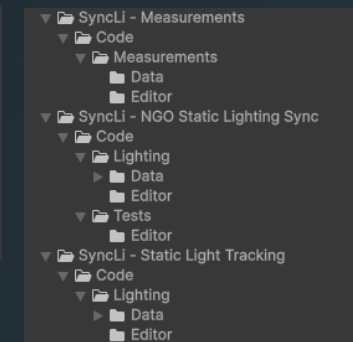
Architecture – Package Based Implementation



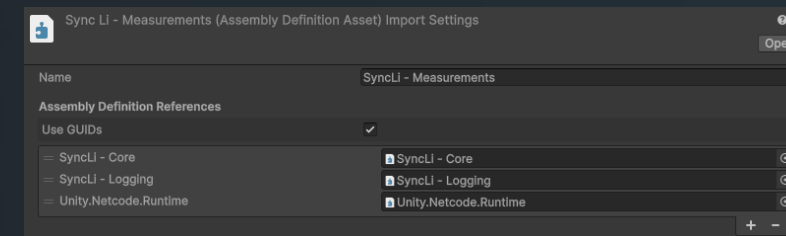
Package Dependency Graph



Package Overview in Project Hierarchy

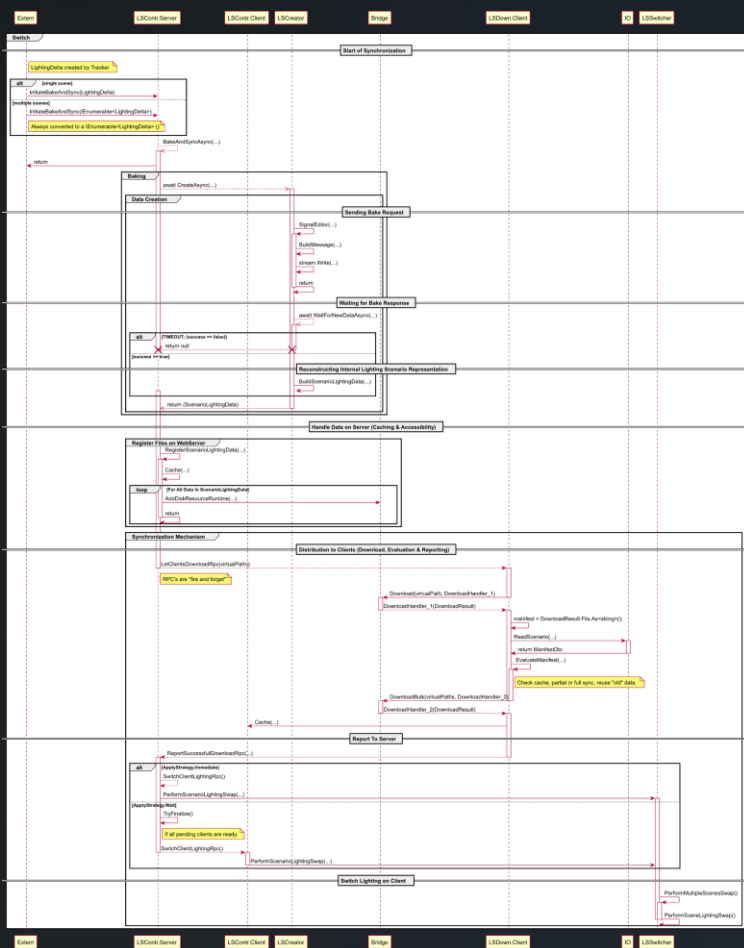


Package Structure



Example Assembly Definition

Architecture – High-Level Sequence



High-Level Sequence Diagram

High-Level Steps:

Server → Unity Editor (Baking)

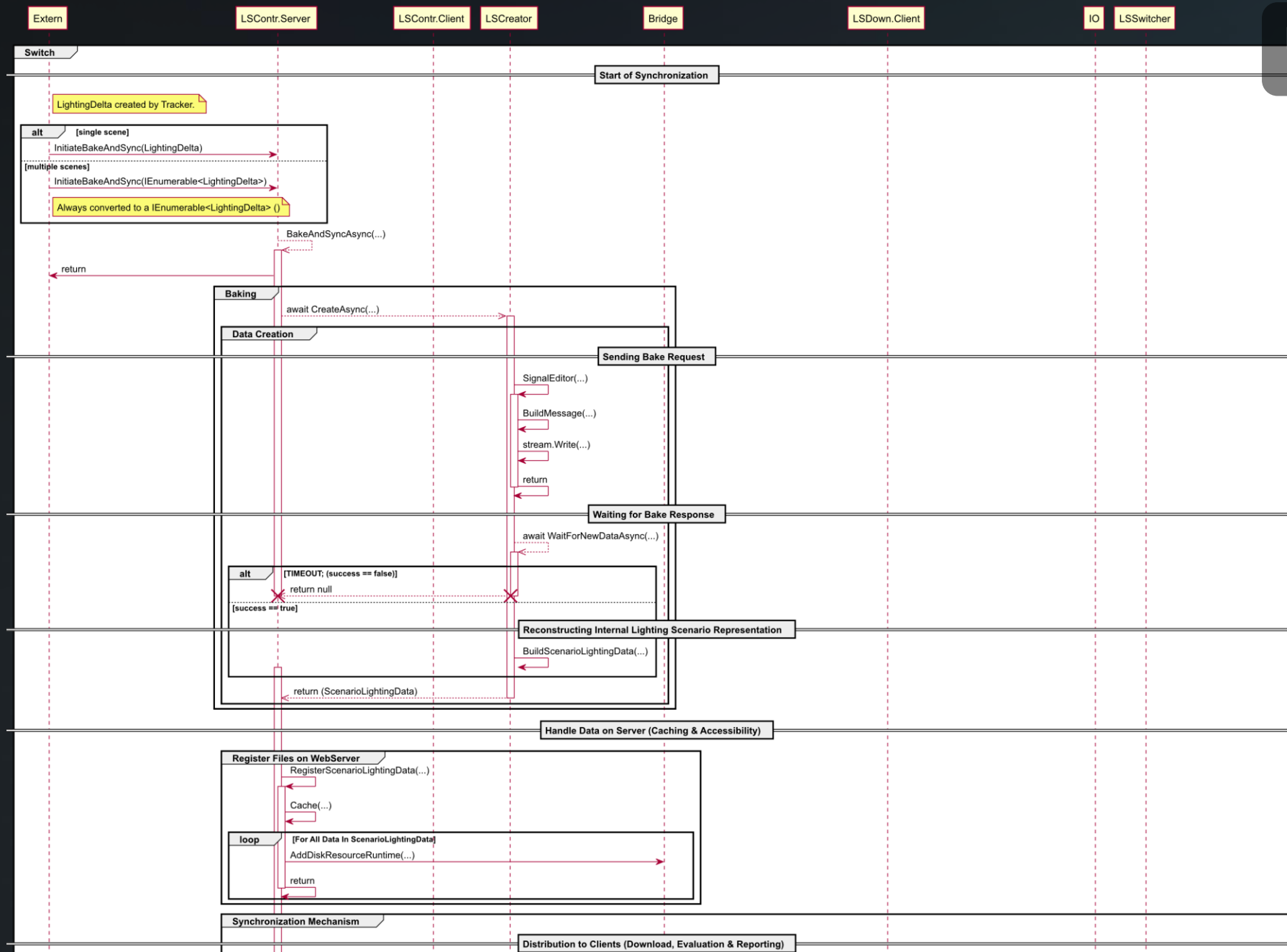
Server $\leftarrow$ Unity Editor (Data)

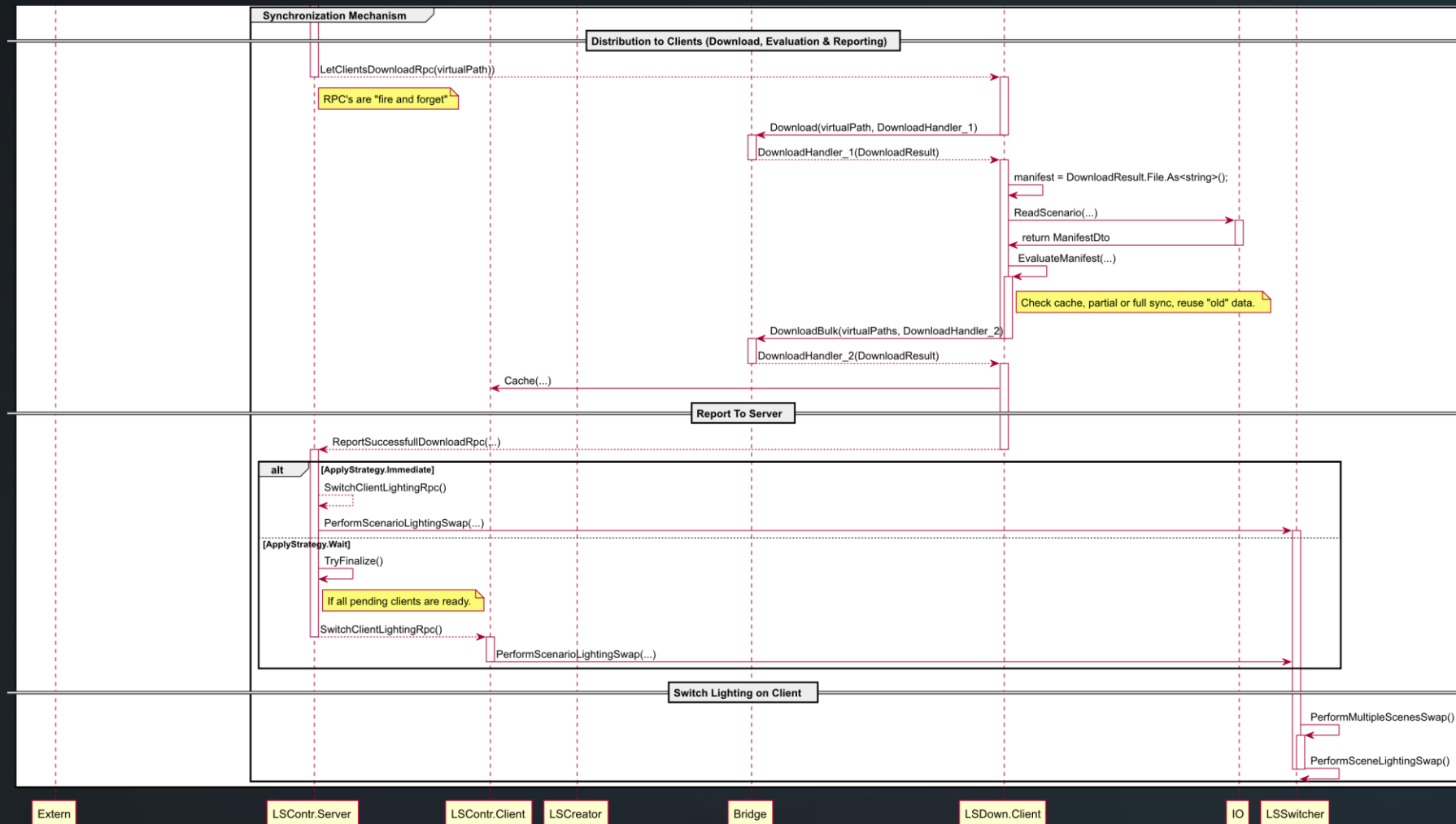
Server Processes Data

Server Distributes Data

Clients Download & Process Data

Servers Signals Switch





Architecture – Network Communication (DTOs)

- Communication is abstracted into Data Transfer Objects (DTOs)
 - Just a JSON serializable object with fields.
 - JSON-IDs easily controllable
- Abstracts the data of the internal objects into well-defined data containers, that are easily be serialized and deserialized.

Used for building the Manifests,
Baking Request and Response objects.

```
public class ColorDto : BaseDto // 7 usages // 2 exposing APIs
{
    [JsonProperty(Globals.JsonFields.RJsonId)]
    public float R { get; set; } // 6 usages

    [JsonProperty(Globals.JsonFields.GJsonId)]
    public float G { get; set; } // 6 usages

    [JsonProperty(Globals.JsonFields.BJsonId)]
    public float B { get; set; } // 6 usages

    [JsonProperty(Globals.JsonFields.AJsonId)]
    public float A { get; set; } // 6 usages

    public static implicit operator Color(ColorDto dto)
    => new Color(dto.R, dto.G, dto.B, dto.A);

    public static implicit operator ColorDto(Color c)
    => new ColorDto { R = c.r, G = c.g, B = c.b, A = c.a };
}
```

Implementation Code of the “Color” Class of Unity as DTO

```
public class BakeRequestDto : BaseDto // 10 usages
{
    [JsonProperty(Globals.JsonFields.ScenarioNameJsonID)]
    public string ScenarioName { get; set; } = Globals.DefaultScenarioName; // 6 usages

    [JsonProperty(Globals.JsonFields.TimestampJsonID)]
    public DateTime Timestamp { get; set; } = DateTime.UtcNow; // 1 usage

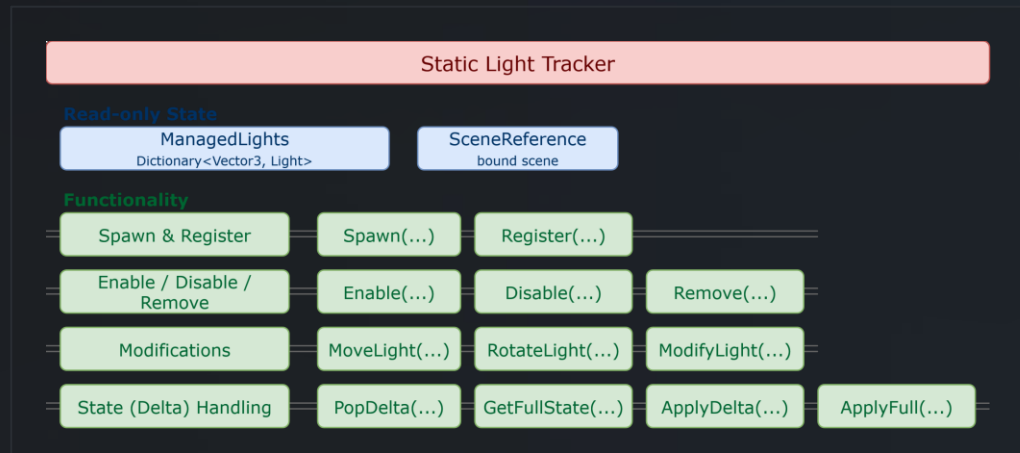
    [JsonProperty(Globals.JsonFields.OutputPathID)]
    public string OutputPath { get; set; } = string.Empty; // 5 usages

    [JsonProperty(Globals.JsonFields.StaticLightFullStateDtosJsonID)]
    public List<StaticLightsFullStateDto> StaticLightFullStateDtos { get; set; } = new(); // 5 usages
}
```

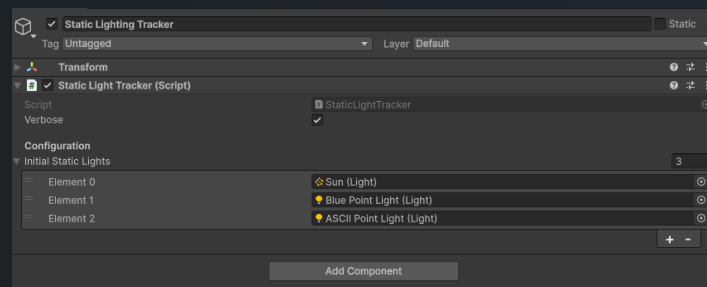
Implementation Code of the “Bake Request” DTO

Architecture

- Tracking Static Lights



Static Light Tracker API Overview



Static Light Tracker GameObject with its Components

```

/// <summary>
/// Centralized place to handle how lights change per iteration.
/// </summary>
// Frequently called
private void ApplyLightingChangeForIteration(int iteration) [1 usage]
{
    if (iteration == 1)
    {
        // Spawn 2 static lights
        _tracker.Spawn(_staticLightSpawnedConfig1, out Light staticLightSpawned1);
        _tracker.Spawn(_staticLightSpawnedConfig2, out Light staticLightSpawned2);
        _staticLightSpawned1 = staticLightSpawned1;
        _staticLightSpawned2 = staticLightSpawned2;

        // Change color of in-scene static light
        _tracker.ModifyLight(_inSceneStaticLight, light =>
        {
            light.color = Random.ColorHSV(0f, 1f, 0.5f, 1f, 0.5f, 1f);
        });
    }
    else
    {
        // Modify the first spawned light
        _tracker.ModifyLight(_staticLightSpawned1, light =>
        {
            light.color = Random.ColorHSV(0f, 1f, 0.5f, 1f, 0.5f, 1f);
        });

        // Modify in-scene static light
        _tracker.ModifyLight(_inSceneStaticLight, light =>
        {
            // Assign a completely random, vibrant color
            // Parameters: Hue(0-1), Saturation(0.5-1), Value(0.5-1)
            light.color = Random.ColorHSV(0f, 1f, 0.5f, 1f, 0.5f, 1f);

            light.intensity = Random.Range(0.5f, 10f);
            light.range = Random.Range(5f, 15f);
        });

        // Modify in-scene the sun (static light)
        _tracker.ModifyLight(_inSceneStaticSun, light =>
        {
            light.intensity = Random.Range(0.5f, 1.5f);
        });
    }
}

```

Code Example how Light Components are: Spawned, Modified and Tracked

Architecture – Debugging Connected Android Devices

Custom Logger

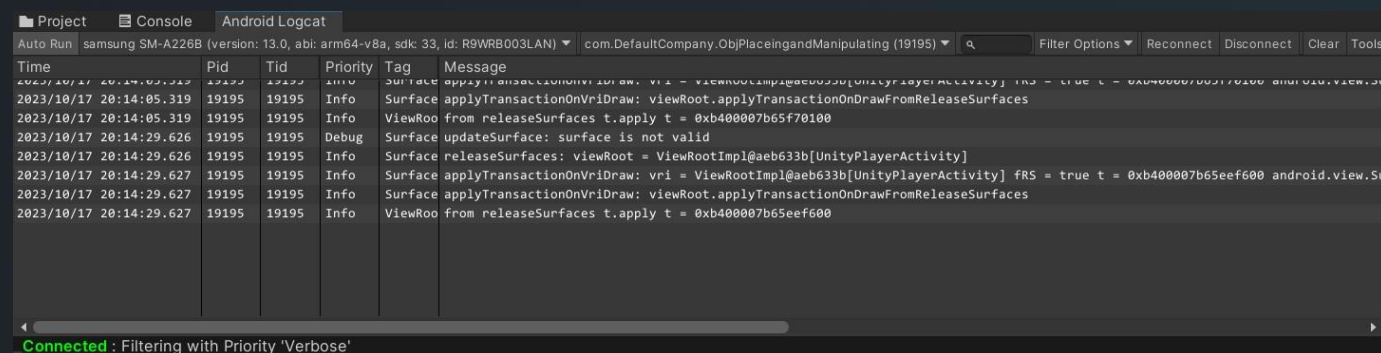
(static and non static, coloring, log categories, hooks into Unity Log Handler)

```
[11:30:59] [CORE] [NET] Local Server started.  
— [NetworkManagerGUI.RegisterClientCallbacks]  
[11:30:59] [DATA] Starting the creation of the runtime directory cache.  
— [HttpFileHandler.InitializeAndFillResourceDictCache]  
[11:30:59] [DATA] Finished creating the runtime directory cache. 0 item(s) available. Took 0 ms.  
— [HttpFileHandler.InitializeAndFillResourceDictCache]  
[11:30:59] [NET] Initialized HTTP server on port 7778 with ip 192.168.0.214 (http://192.168.0.214:7778/).  
— [HttpFileProvider.Activate]  
[11:30:59] [NET] Starting HTTP server.  
— [HttpFileProvider.StartListening]  
[11:30:59] [NET] HTTP server will now listen on port 7778.  
— [HttpFileProvider.ListenLoopAsync]
```

Unity Log with custom log handler

Android Logcat for Meta Quest 3

Important: Only available in development build with “Script Debugging” enabled.



Android Logcat in Unity (https://www.francescogarofalo.it/en/post/log-android-in-unity-with-android-logcat/androidlogcat_huf4f77bfb07927f1ccDca63622e7e9912_121113_1600x0_resize_q75_box.jpg)

System Design & Implementation

Architecture

— Configuration & Challenges —

Workflow (Measurement Example)

Challenges - Android

Android lightmaps differed from PC despite identical bake settings.

Checked Quality Settings, Graphics Settings, Static Batching, shader stripping, compression, and color space.

Android Logcat debugging pointed to texture decoding/loading issues.

Challenges - Android

EXR (and baked PNG workflow) was unreliable on Android during runtime loading.

Implemented custom .rawlm format: Header + raw GetRawTextureData() payload.

Direct restoration on Android without problematic image decoding.

Consistent lightmap results across PC and Android.

Problem: Uncompressed File Format
is large (empty pixels)

	10 texels/unit	50 texels/unit
.exr	53 mb	310 mb
as .rawlm	61 mb	416 mb
.png	6 mb	31 mb
as .rawlm	31 mb	208 mb

Comparison of exported files sizes between two lighting scenarios

Challenges - Android

Loading many lightmaps in one frame caused RAM spikes and allocation failures.

Introduced time-sliced processing: Decode/load one lightmap per frame.

Reduces memory spikes and allows Android garbage collection.

Trade-off: small loading latency, but stable runtime behavior.

Project Configuration Summary

Disable Static Batching.

Enable HTTP (Unsafe).

Use the same Quality and Graphics Settings on PC and Android.

Keep Lighting Settings identical across platforms.

Use the same Lightmap Compression Quality (e.g., High).

Do not change any related settings in a “session”.

System Design & Implementation

Architecture

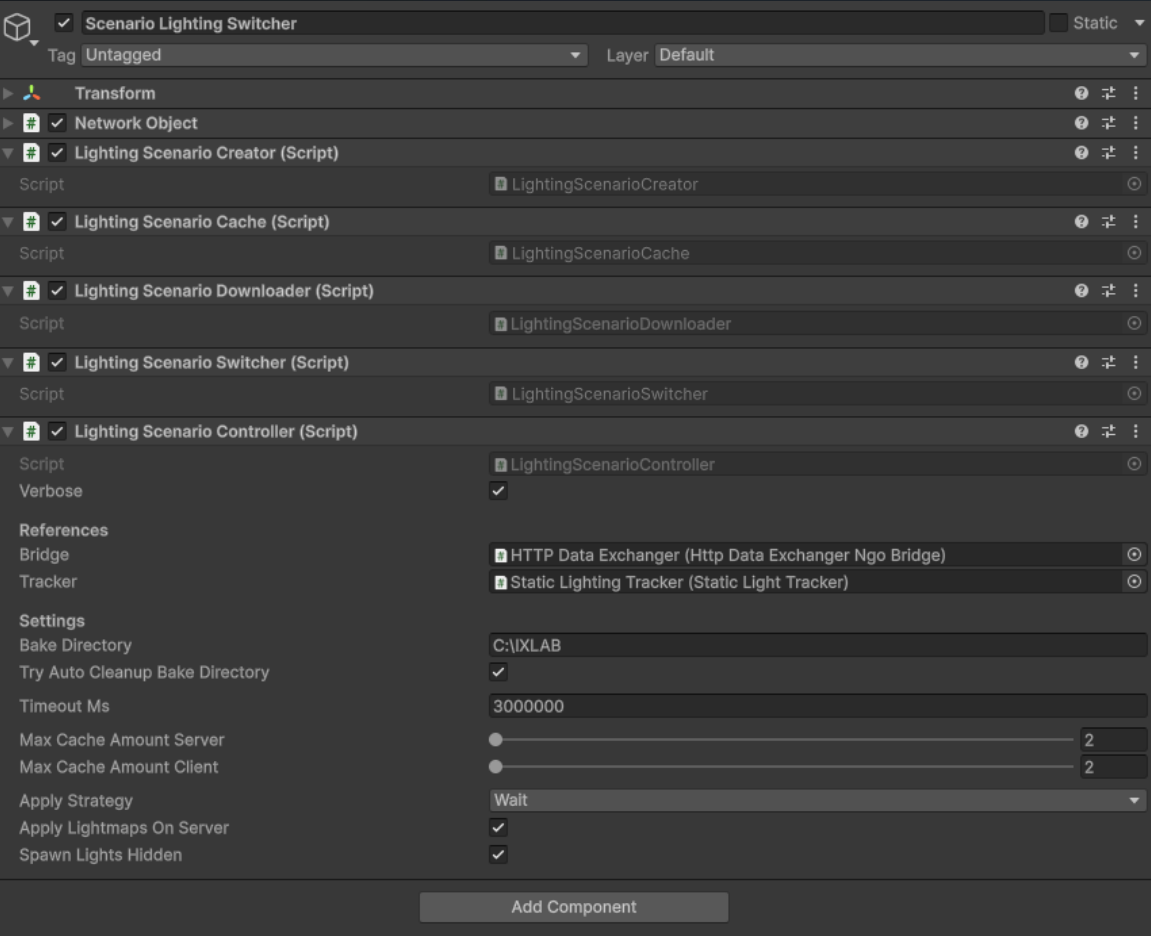
Configuration & Challenges

— Workflow (Measurement Example) —

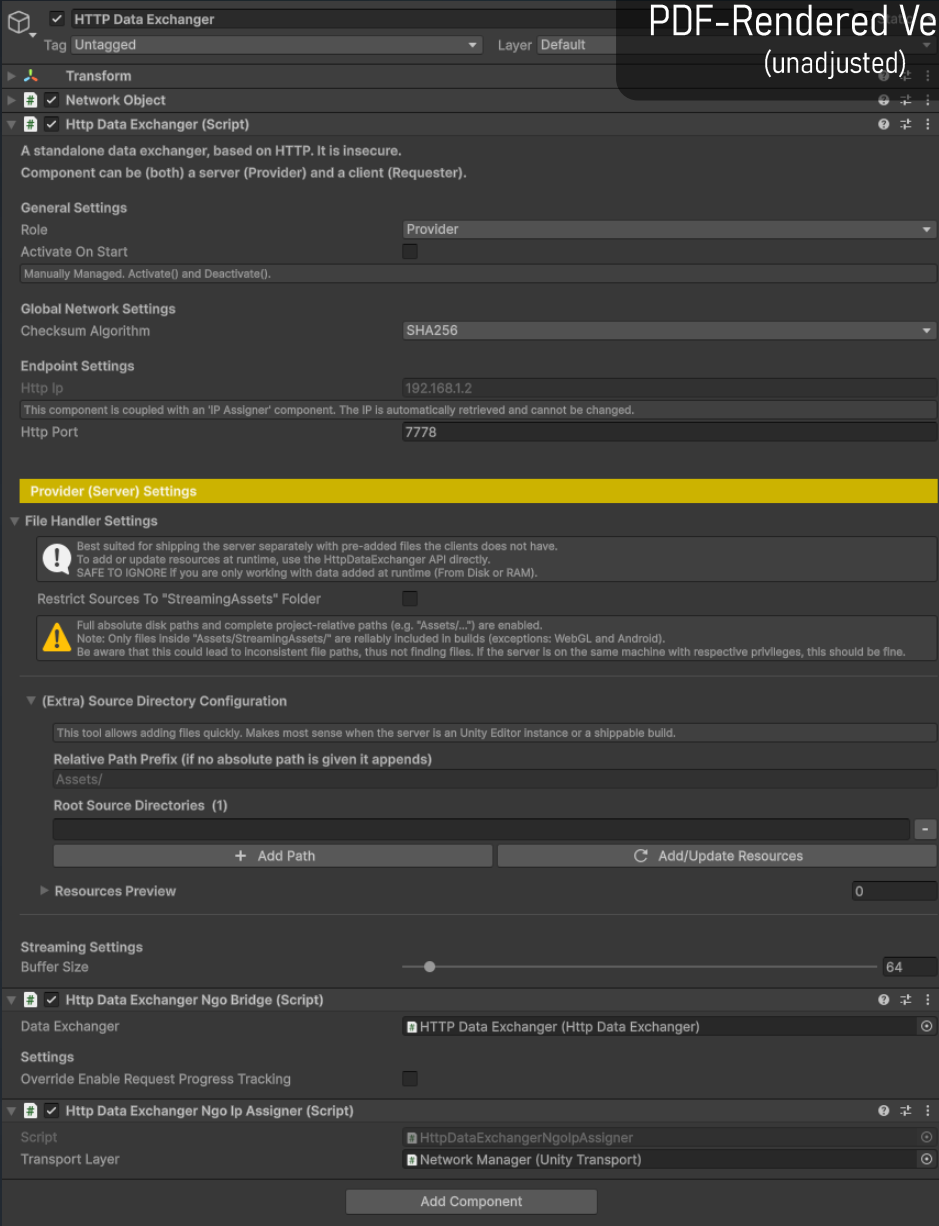
Project Setup

- Configure everything (at least lighting related)
- Copy the project (or use hard links)
- Open copy in a new editor instance

Server-Side Setup



Scenario Lighting Switcher GameObject with its Components

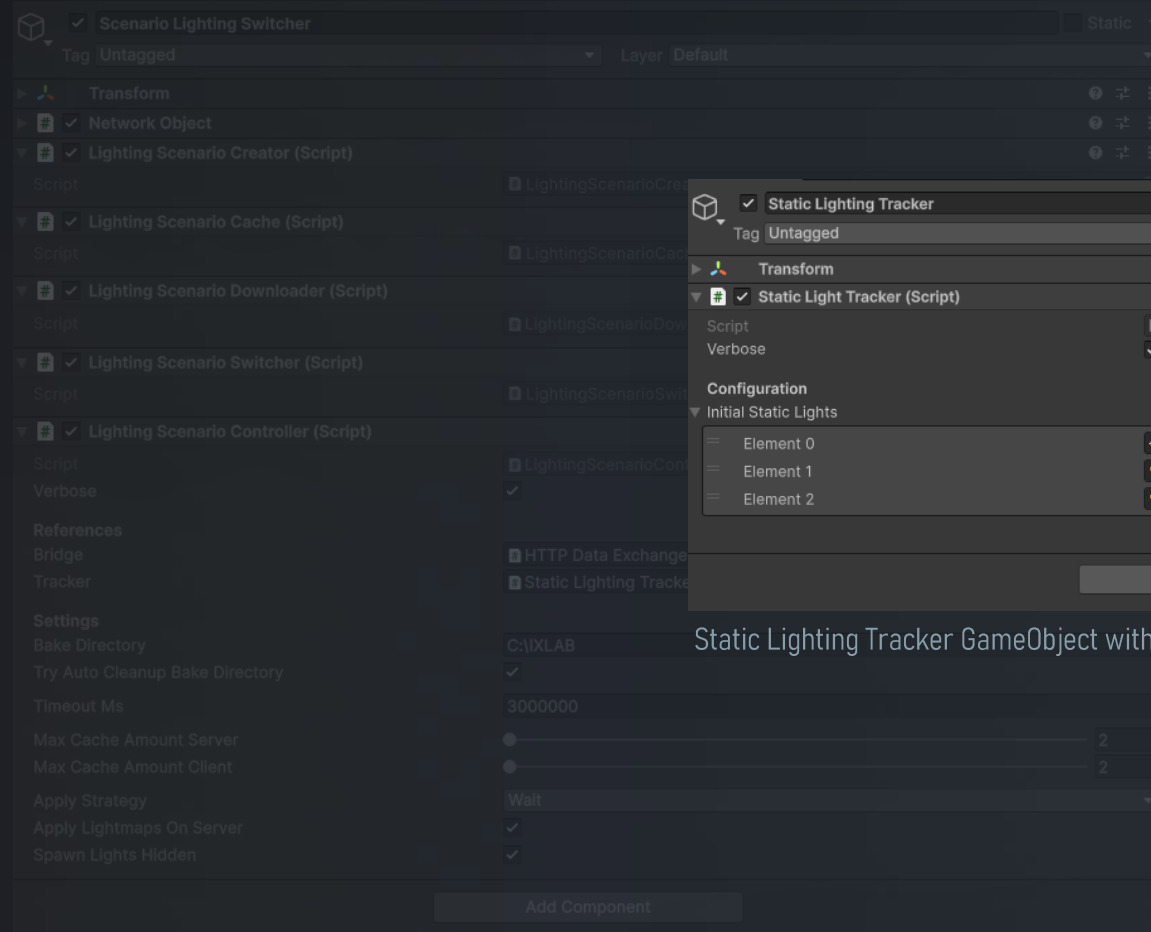


HTTP Data Exchanger GameObject with its Components

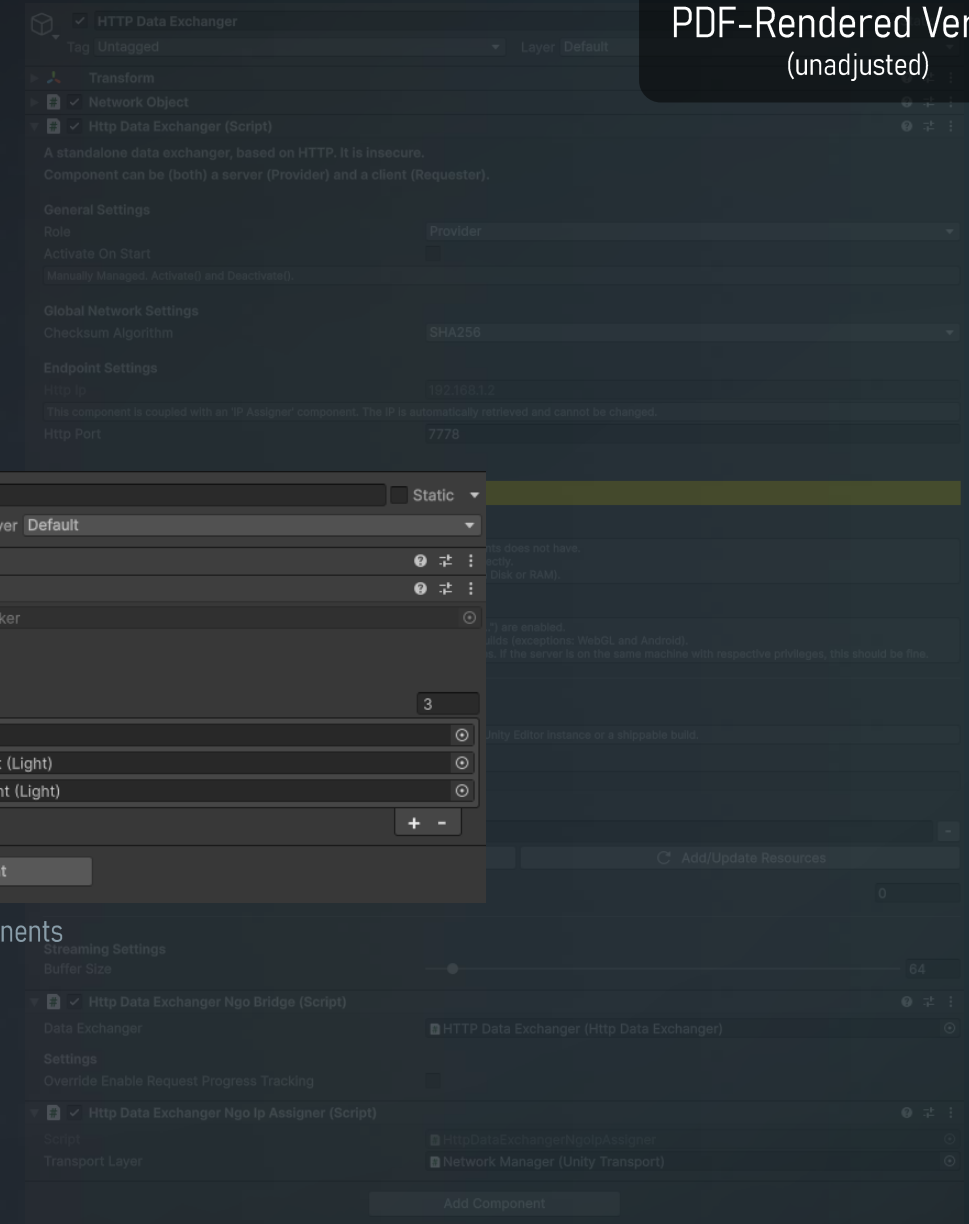
PDF-Rendered Version
(unadjusted)

Server-Side Setup

PDF-Rendered Version
(unadjusted)



Scenario Lighting Switcher GameObject with its Components



HTTP Data Exchanger GameObject with its Components

Editor-Side Setup

Baked Lighting Scenario Exporter

This tool automates the extraction of baked lighting data (Lightmaps and Probes) into a portable folder structure. Textures are automatically imported as 'Lightmap' type to ensure correct linear color space interpretation.

Scenario Name Select Existing Scenario Folder

Scene Name

Scenario Export Location ☐ Restrict to StreamingAssets

Full absolute disk paths and complete project-relative paths (e.g. "Assets/...") are enabled. Note: Only files inside "Assets/StreamingAssets/" are reliably included in builds (exceptions: WebGL and Android). Be aware that this could lead to inconsistent file paths, thus not finding files. If the server is on the same machine with respective privileges, this should be fine.

Browse... ↻

Use Pers. Path

Path Preview

Export Lightmaps As Everything

START BAKE & EXPORT

Unity's built-in baker is a bit of an "all or nothing" system. If you want to bake only one room while keeping the rest of the scene the same, you usually have to use Sub-Scenes or Additive Scene Loading.

Export Current Bake (No Re-Bake)

Enable Verbose Logging ☒

Start SESSION and LISTEN for signals.

Baked Lighting Scenario Exporter Tool Window; No active Session



Baked Lighting Scenario Exporter

This tool automates the extraction of baked lighting data (Lightmaps and Probes) into a portable folder structure. Textures are automatically imported as 'Lightmap' type to ensure correct linear color space interpretation.

Scenario Name Select Existing Scenario Folder

Scene Name

Scenario Export Location ☐ Restrict to StreamingAssets

Full absolute disk paths and complete project-relative paths (e.g. "Assets/...") are enabled. Note: Only files inside "Assets/StreamingAssets/" are reliably included in builds (exceptions: WebGL and Android). Be aware that this could lead to inconsistent file paths, thus not finding files. If the server is on the same machine with respective privileges, this should be fine.

Browse... ↻

Use Pers. Path

Path Preview

Export Lightmaps As Everything

START BAKE & EXPORT

Unity's built-in baker is a bit of an "all or nothing" system. If you want to bake only one room while keeping the rest of the scene the same, you usually have to use Sub-Scenes or Additive Scene Loading.

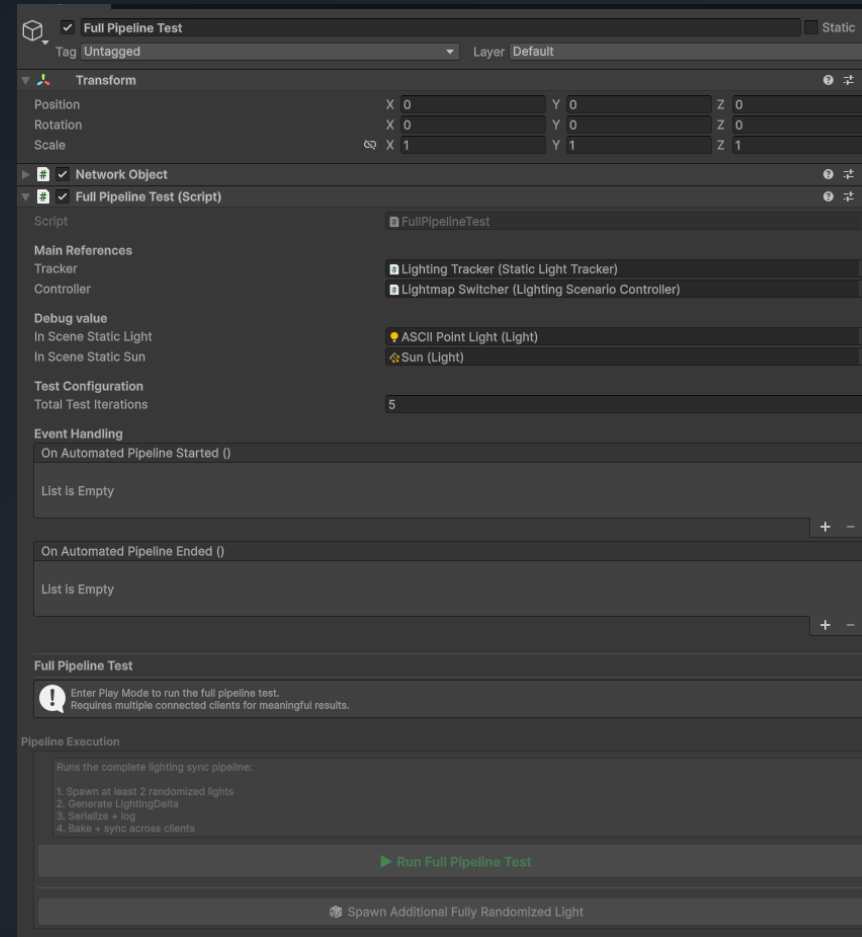
Export Current Bake (No Re-Bake)

Enable Verbose Logging ☒

End Session and Stop listening Listener running on 127.0.0.1:52672
Session is locked.

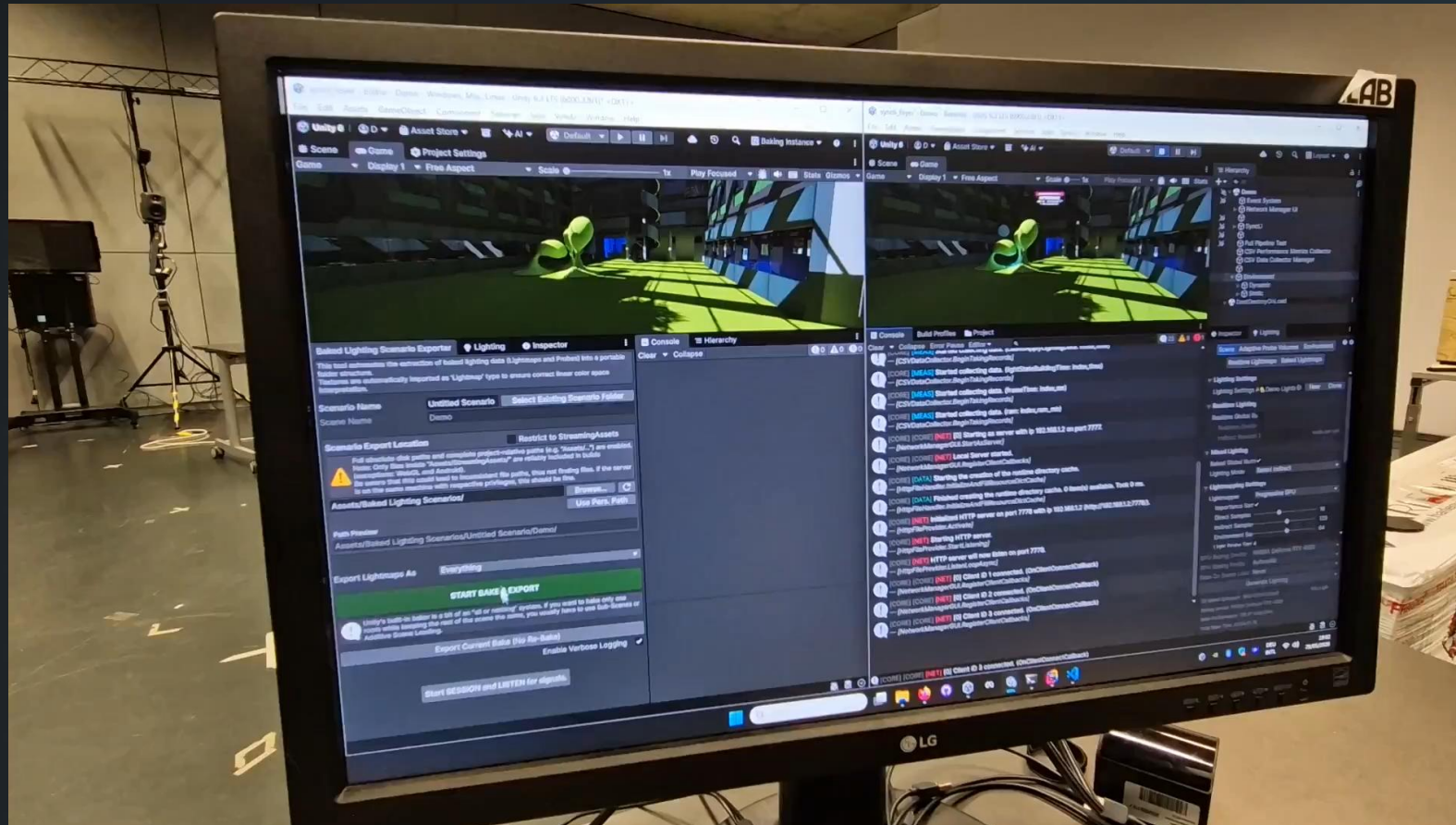
Baked Lighting Scenario Exporter Tool Window; Active Session

Debugging / Testing Setup



Full Pipeline Test GameObject with its Components

Measurement Functionality Showcase



Video of the functionality and sequence of the system in a measurement context

— Requirement Fulfilment Analysis —

Requirements: Recap

Hard	Soft
Core functionality	Usability
Unity-exclusive	Security
Server authoritative	
Negligible introduced system latency	
Reliability	
Performance	

Table of requirements

Requirement: Core Functionality



Requirement: Unity Exclusive

The system depends on the following Unity packages:

- Newtonsoft JSON (Unity)
- Netcode for GameObjects
- Standard C# Libraries



FULFILLED

Requirement: Server-authoritative

Only the servers can signal the editor and receive signals from it.
Baking GI is therefore only possible as the server.

Only the server's static lights state is used for baking.
Clients can only request changes to it.

FULFILLED

Requirement: Negligible introduced system latency

Measurement Setup



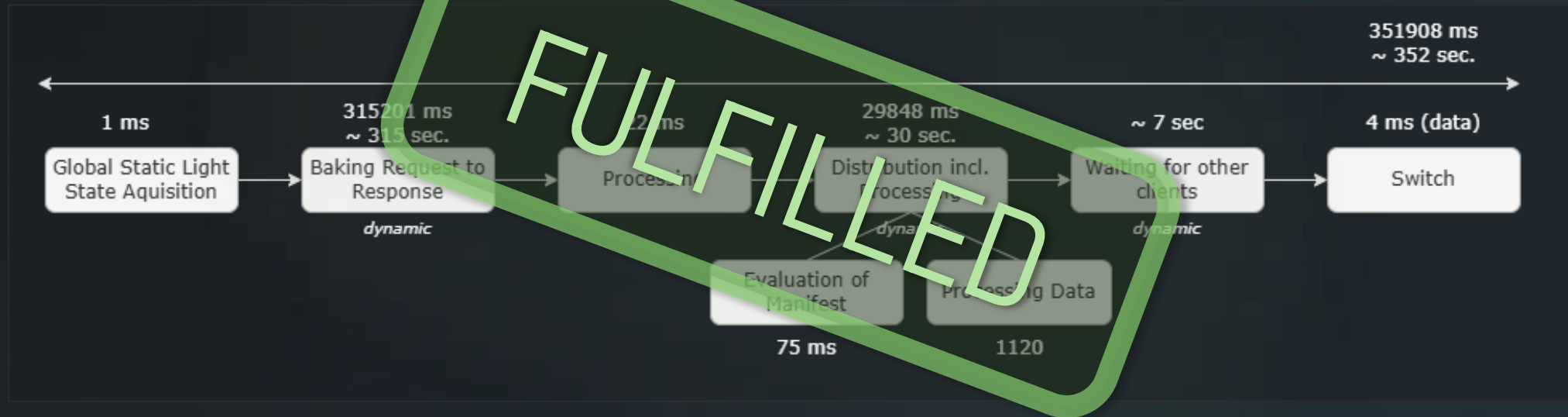
Measurement Setup in the IXLab



Video about the efforts of the measurement setup

Requirement: Negligible introduced system latency

Latency Comparison (50 texels/unit, averages over 3 clients)



Latency overview of different components at 50 texels/units

- 23 ms on server-side
- 1199 ms on client-side

Requirement: Reliability

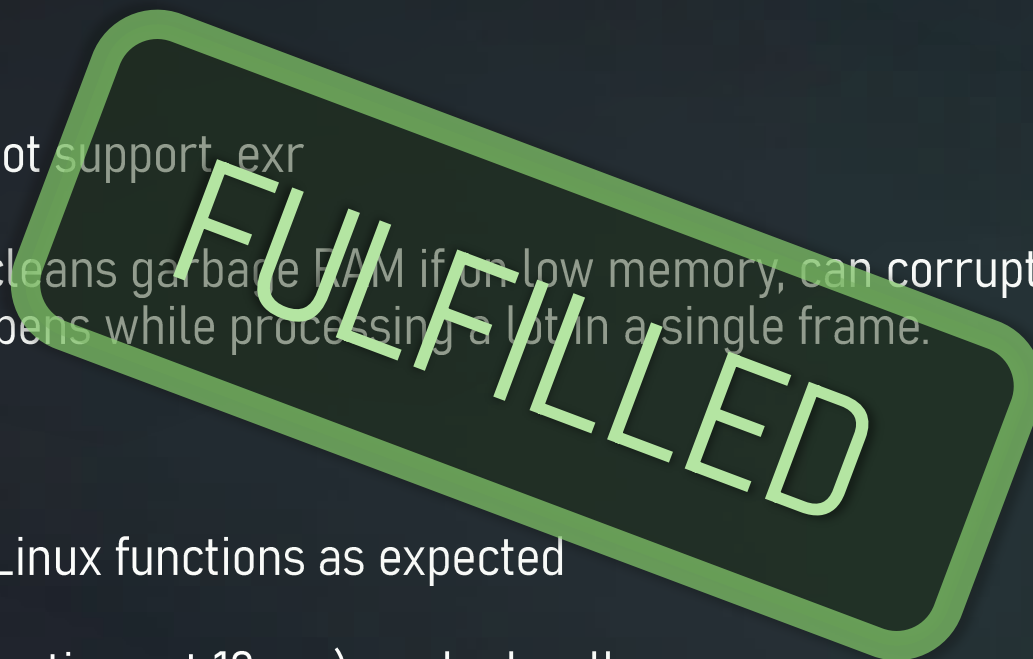
Platform-specific

1. Android does not support `exr`
2. Meta Quest 3 cleans garbage RAM if on low memory, can corrupt data when happens while processing a lot in a single frame.

> Both solved

– Windows/Mac/Linux functions as expected

Measurement tests (20 iterations at 10 res) worked well.



Requirement: Performance (FPS)

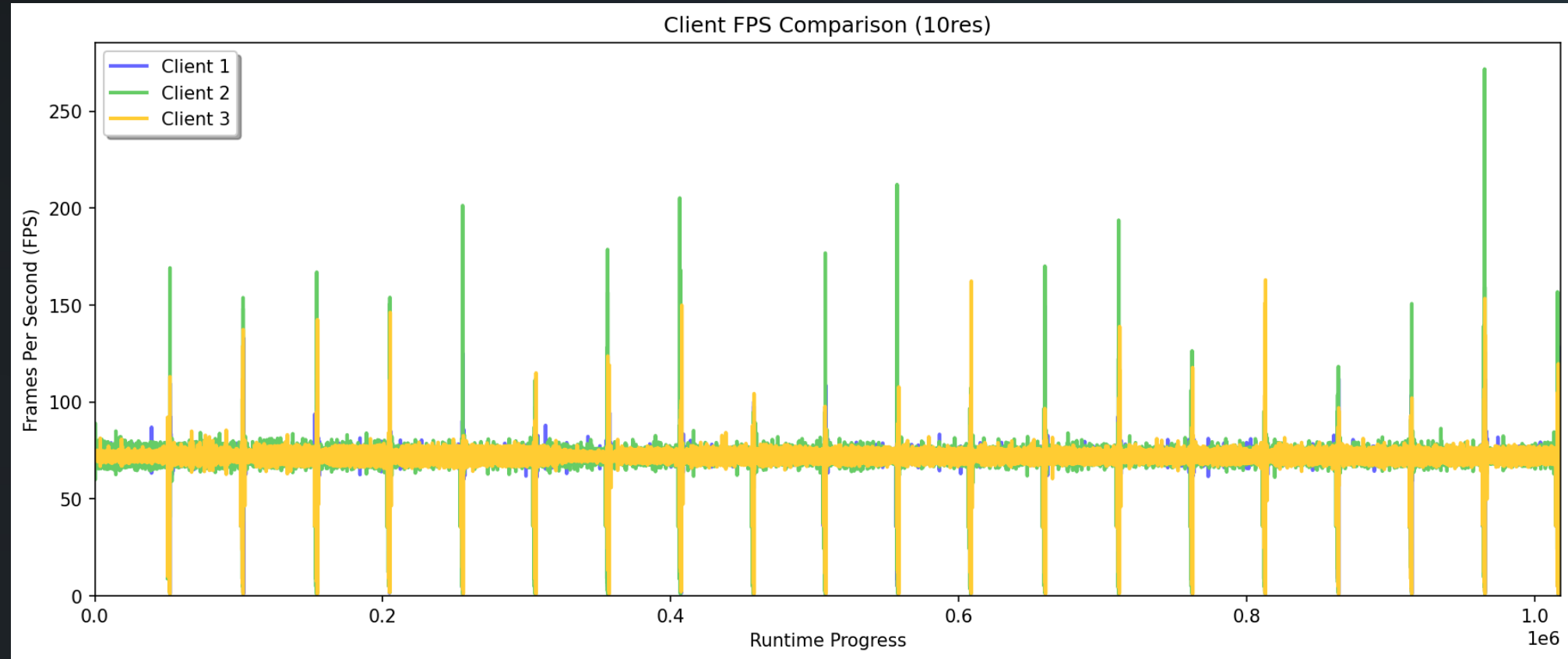
No dynamic lights are used → Arbitrary number of lights possible

Changing data like lightmaps, light probes uses custom data format (.rawlm).

Necessitates: Texture2D.LoadRawTextureData(), that **MUST** run on the main thread.

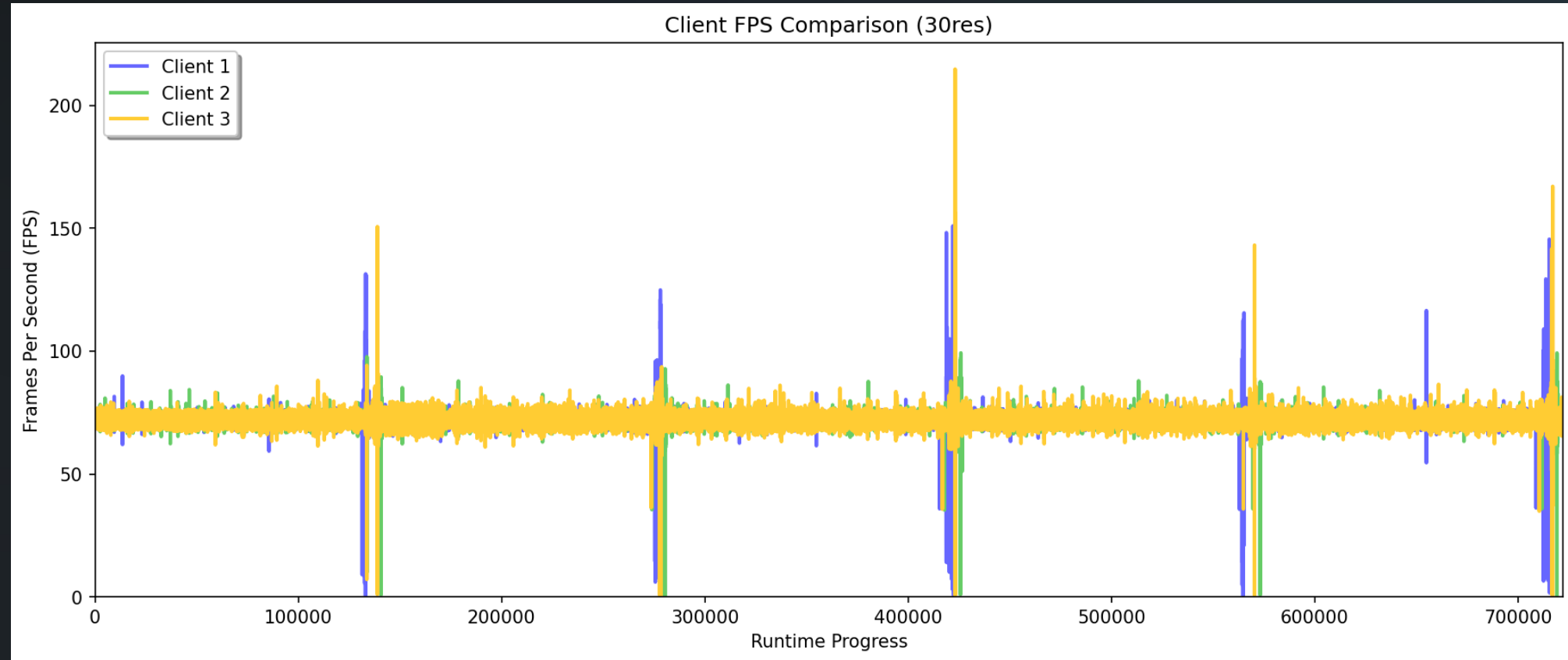
Can we detect spikes?

Requirement: Performance (FPS Spikes)



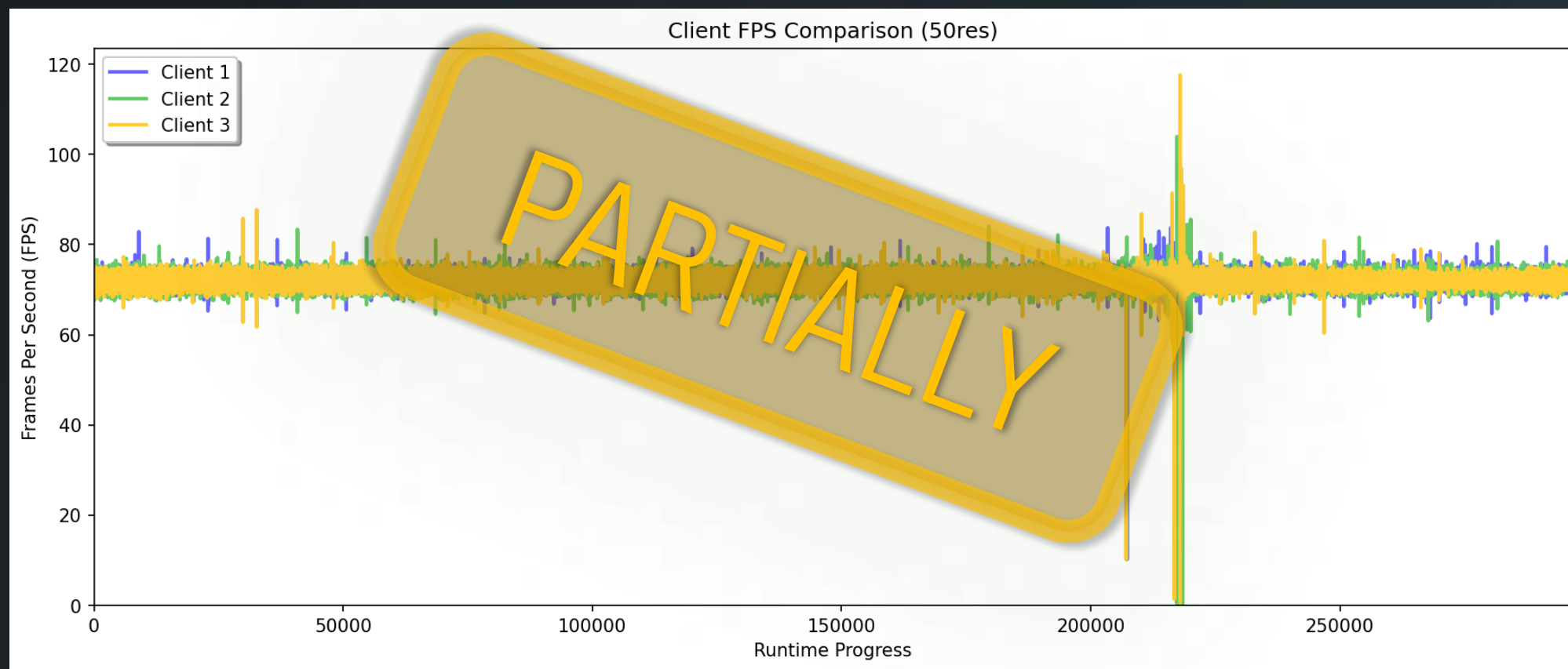
Comparison of FPS between server and 3 clients at a lightmap resolution of 10 over 20 iterations

Requirement: Performance (FPS Spikes)



Comparison of FPS between server and 3 clients at a lightmap resolution of 30 over 5 iterations

Requirement: Performance (FPS Spikes)



Comparison of FPS between server and 3 clients at a lightmap resolution of 50 over 1 iteration

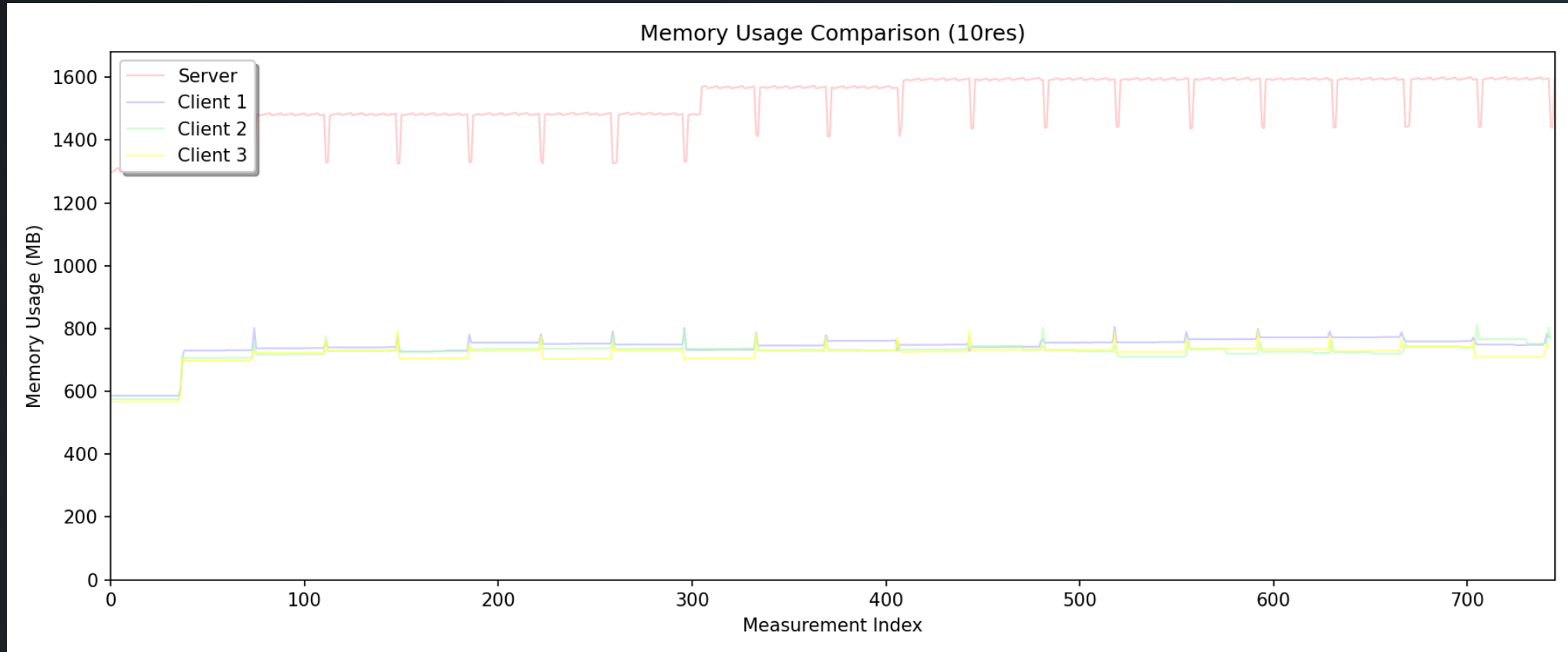
Requirement: Performance (Memory)

The currently used light is cached.

For a working, most memory efficient approach a cache size of 2 is used.

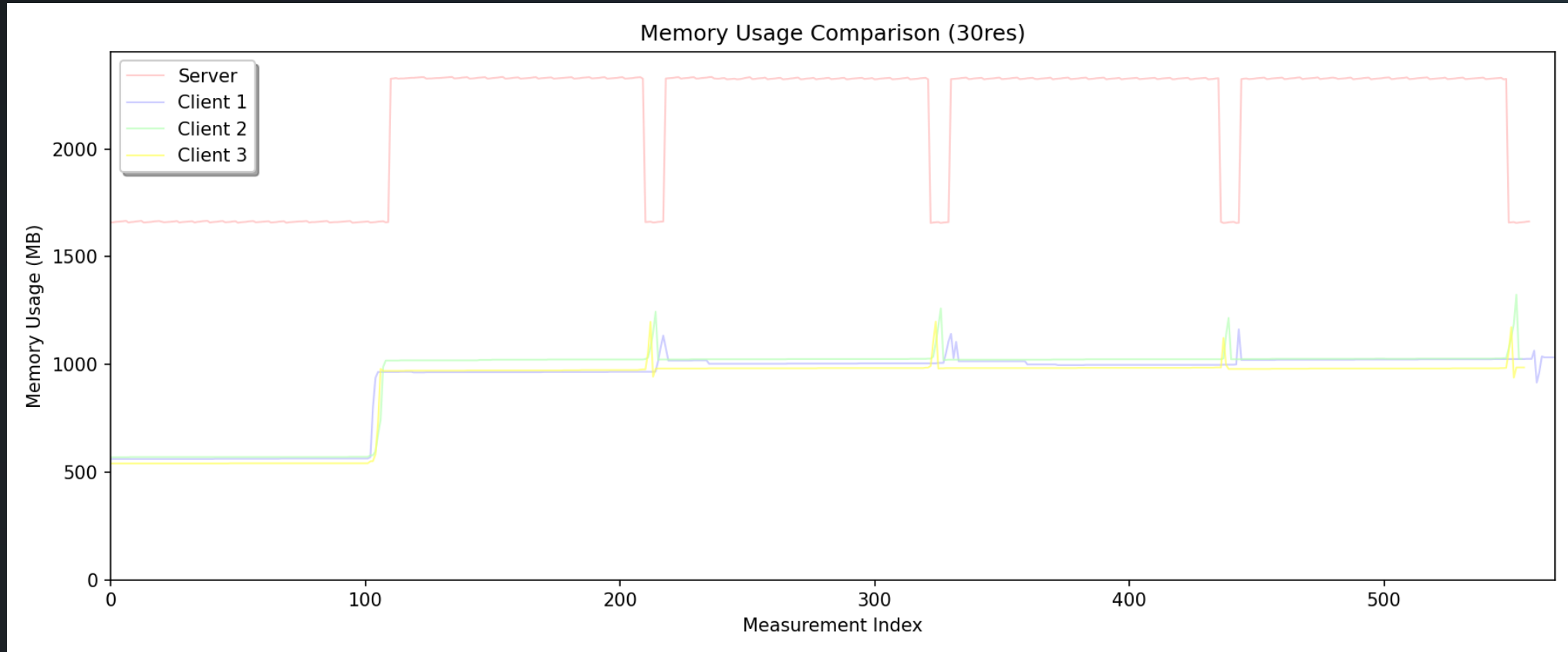
When a new one arrived, it is processed and cached as well to be able to swap on signal.

Requirement: Performance (RAM)



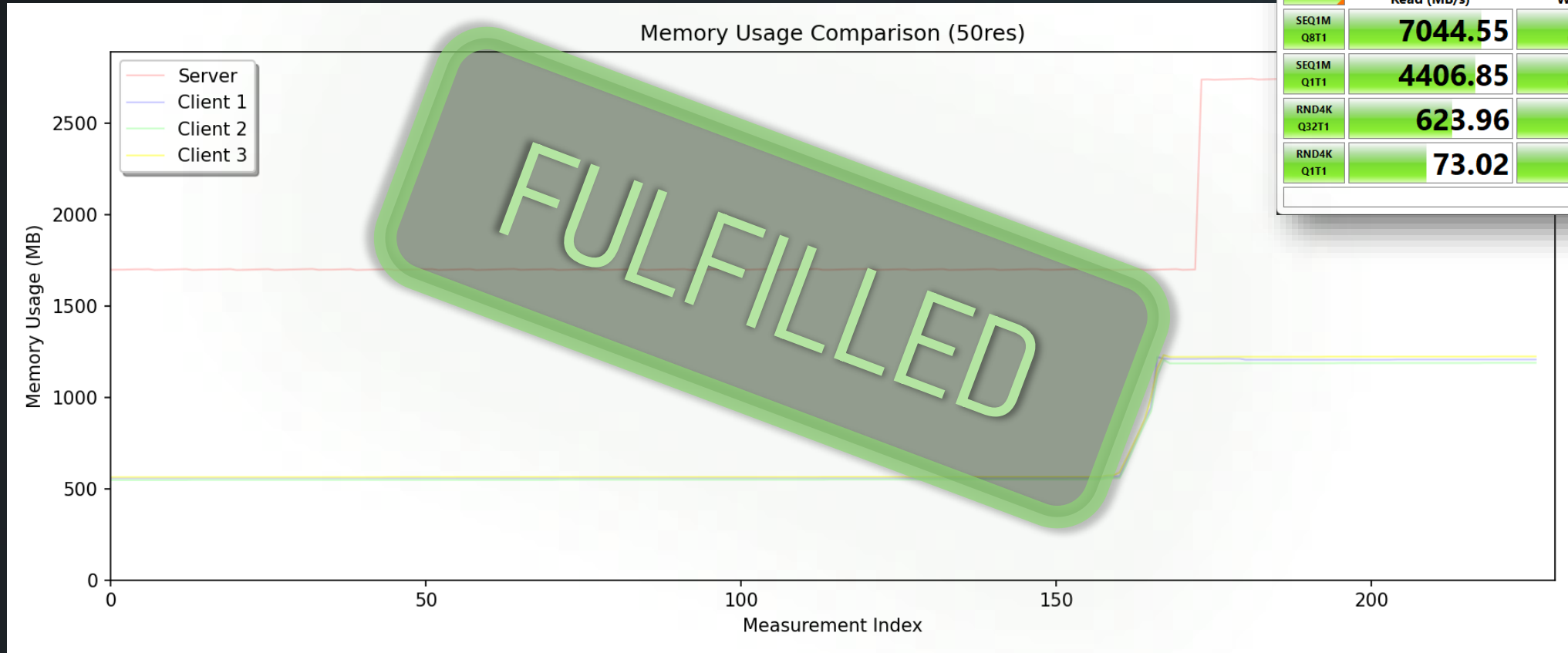
Comparison of RAM usage between server and 3 clients at a lightmap resolution of 10 over 20 iterations

Requirement: Performance (RAM)



Comparison of RAM usage between server and 3 clients at a lightmap resolution of 30 over 5 iterations

Requirement: Performance (RAM)



CrystalDiskMark 9.0.2 x64 [Admin]

File Settings Profile Theme Help Language

All 3 1GiB C: 23% (217/952GiB) MB/s

	Read (MB/s)	Write (MB/s)
SEQ1M Q8T1	7044.55	5272.53
SEQ1M Q1T1	4406.85	5252.78
RND4K Q32T1	623.96	508.15
RND4K Q1T1	73.02	250.07

Comparison of RAM usage between server and 3 clients at a lightmap resolution of 50 over 1 iteration

Soft Requirements

Usability

- extensive editor tooling
- documented, package-based
- custom logging and user friendly interactions
- info boxes for in-editor use

Security

- only local, experimental use
- HTTP must be allowed (unencrypted)
- checksum verification (MD5, SHA256) (extendable)

FULFILLED

Summary

6/6 hard requirements fulfilled

2/2 soft requirements fulfilled

SUCCESS

— Future Work & Present Problems —

Remaining Work

- Compress File format .rawlm for better memory usage
- Additive scene loading & partial sync
- Remote baking for dedicated baking stations
- Reflection probes (pipeline could handle this)

- Field tests in real use-cases
- Is it even practical and feasible to work with such high latencies?

Unity could present a hybrid solution for partially baking async at runtime, but by-design baked lighting is static and is not supposed to be modified.

This trade-off adds quality over control.

– Summary –

Summary

Successfully implemented:

“[...] a method for modifying baked Global Illumination (GI) at runtime within the Unity Engine, while maintaining performance and memory efficiency on Android mobile devices.”

under the presented requirements.

(Functionality; Unity-exclusive; Server authoritative; Reliability; Latency; Performance; Usability; Security)

- High e2e latency (Baking + Downloading) but works.
- There are mostly optimizations left to do, except reflection probes.

Learnings

- How to plan and implement a system package-based.
- File formats (.exr, .png, .rawlm).
- Better understanding of (baked) GI and its tricks.
- Distributing work-load asynchronously.

RESEARCH PROJECT

Synchronizing Remote-Calculated Baked Static Lighting at Runtime to Mobile XR Headsets in Unity

— a system of baking a server-authoritative runtime-scene, distributing and applying them on clients.

Thanks for Listening!
Questions



Live Demo

Steps:

1. Let's modify the scene first on the server (server-authoritative, clients could request)
2. Put HMD (Meta Quest 3) on. (App should be launched already)
3. Enjoy virtual APB
4. Wait for modifications to take place.

Sources

Background Image

https://connect-mediagw.unity.com/h1/20211123/learn/images/c8d6c07e-c130-48b2-8a0f-c96ea01dab9f_CC_Lighting_Cover.png