ESAC TAP stateless follow up

IVOA November 2025 Interoperability Meeting
J. Osinde[1], I. León[2]
C. Ríos[1], R. Parejo[1], J. Ballester[1], A.Alonso[1]
R. Bhatawdekar[3]
SCO-08: Archives Software Development

ESAC
Camino Bajo del Castillo s/n, Urb. Villafranca Del Castillo
28692 Villanueva de la Cañada (Madrid) Spain
[1]Starion for ESA, [2]AURORA Technology B.V, [3]ESA

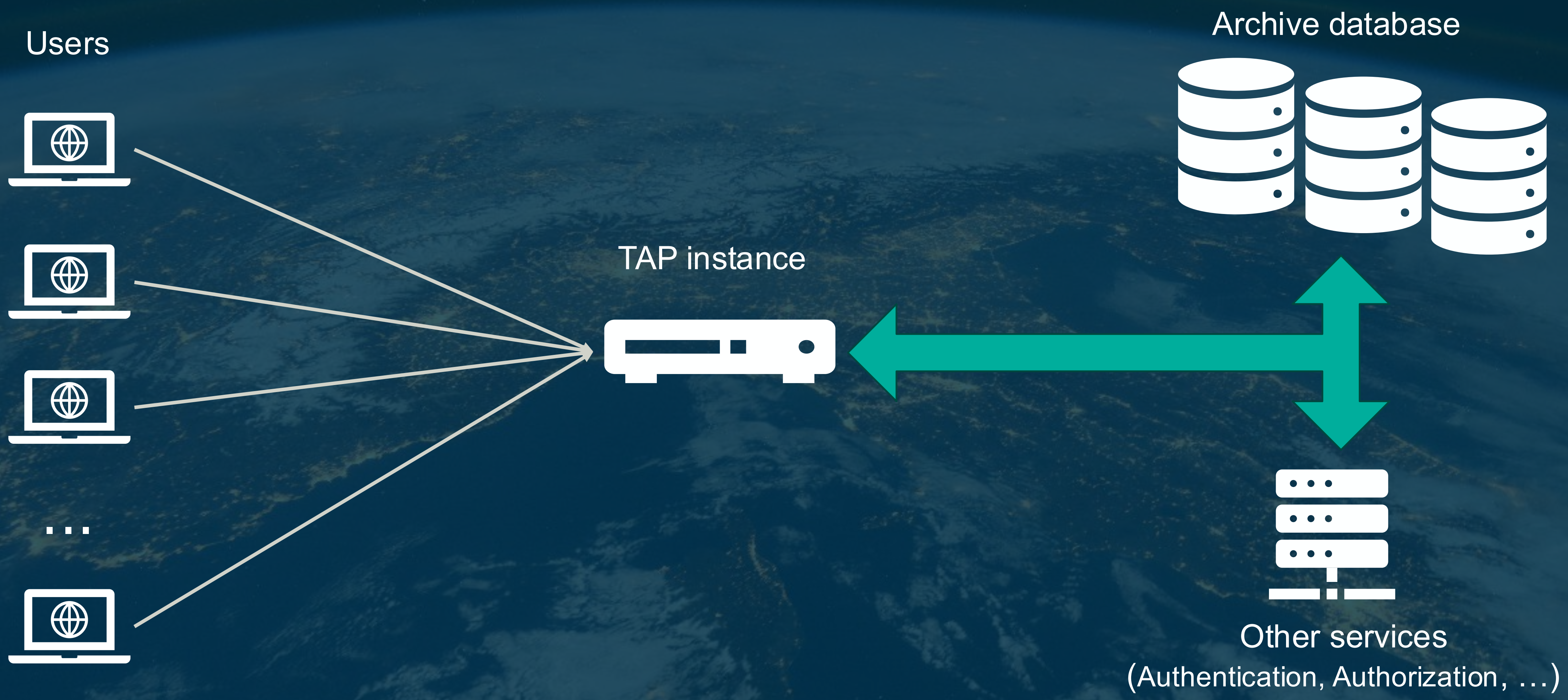
The goal: Key Advantages of Stateless, Scalable Service Architectures

- Easy Horizontal Scalability: Since each instance is independent and doesn't rely on local or local session state, you can easily add or remove instances based on load.
- Simplified Load Balancing: Any request can be routed to any instance, since all instances are functionally identical
- High Availability and Fault Tolerance: If one instance fails, others can continue processing requests without interruption.
- Easier Deployment and Rolling Updates: Stateless services don't depend on persistent in-memory state, so you can deploy new versions without disrupting ongoing sessions

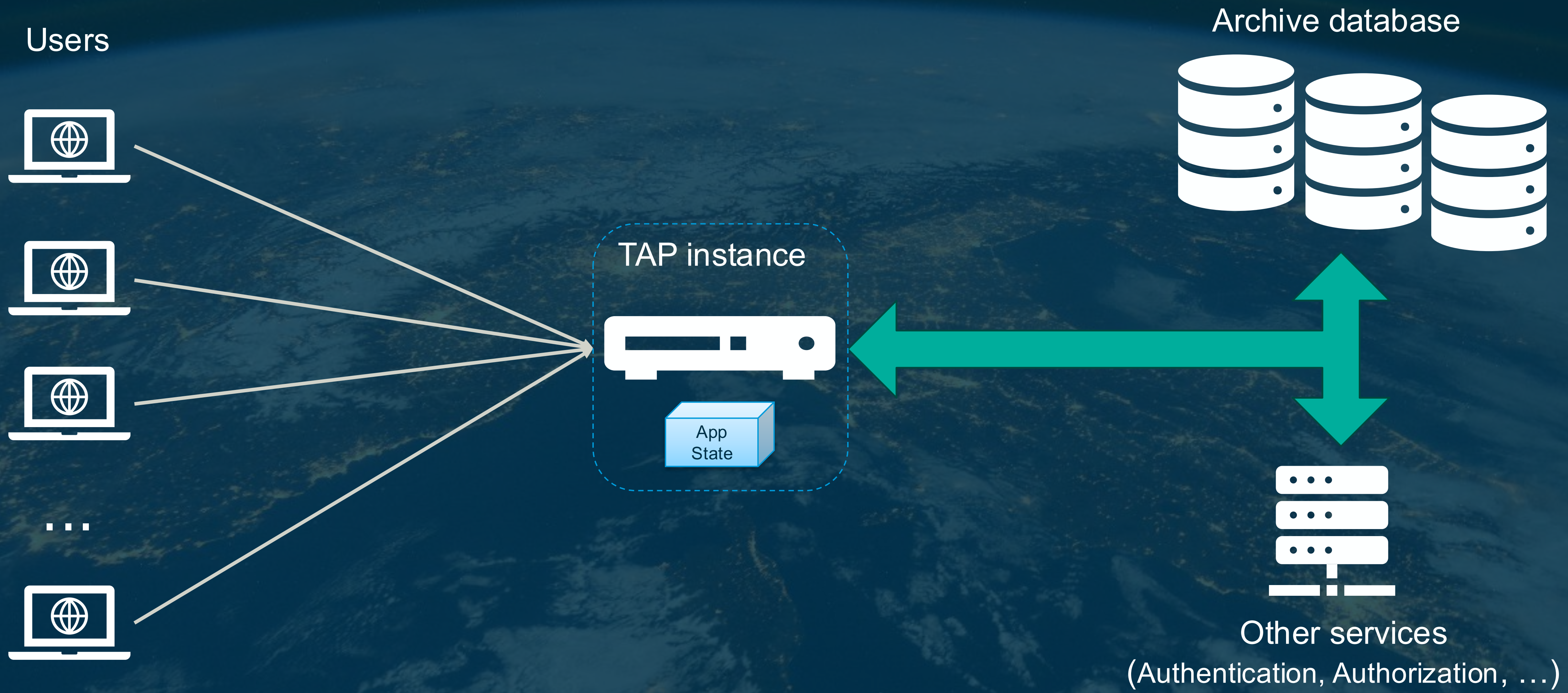
But … TAP is not a stateless service

- Active session
  - Functionality managed by Spring Session library
- Events
  - User log, Job creation/running/done, shared resources, quota updates and system wide notifications
- User status
  - Task monitor feedback (table uploads, x-match jobs, format conversions, etc)
- User parameters
  - Roles, timeout values, disk and memory quotes
  - One entry per user in the system

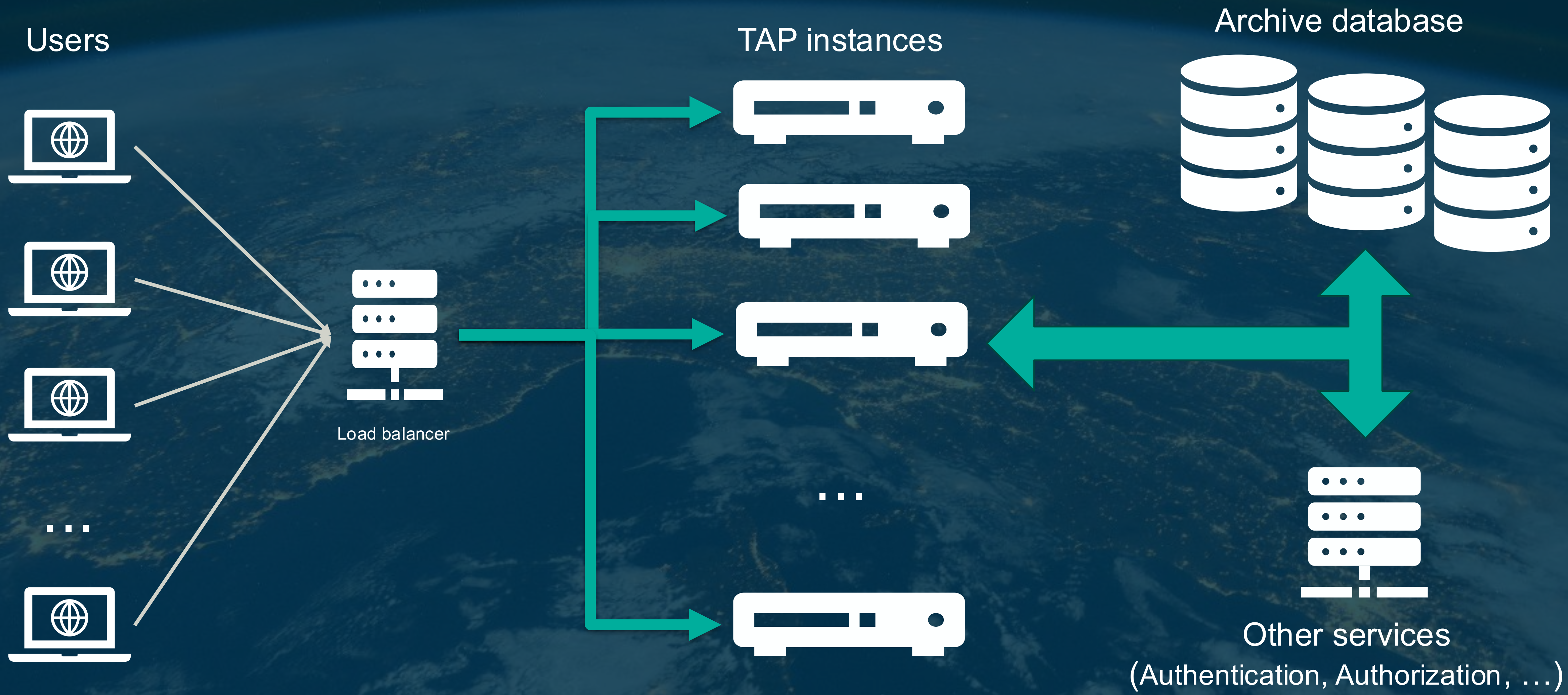
TAP Stateful layout



TAP Stateful layout + Application state



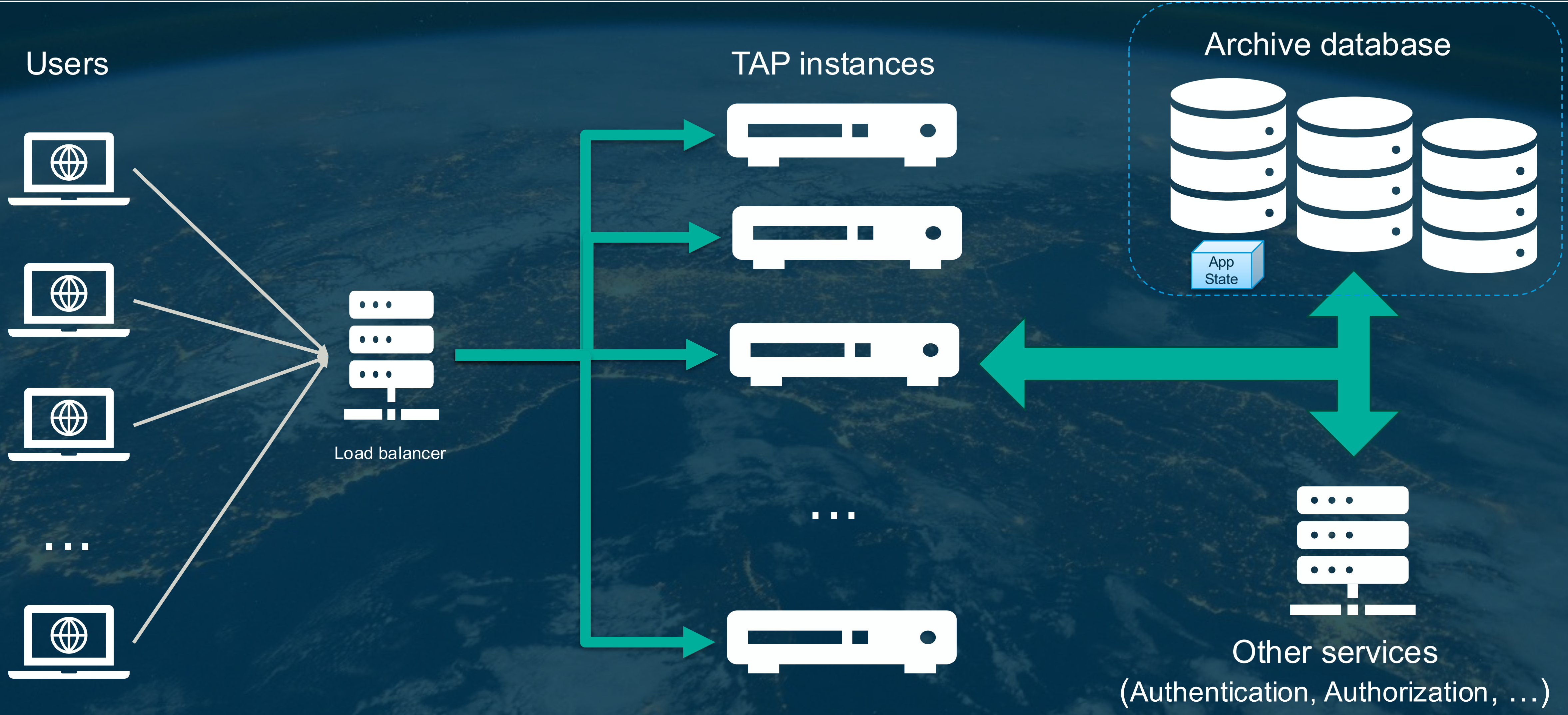
TAP Stateless layout



How to share this information

- The TAP service was refactored to extract this information from the application itself.
- Now the question was: where do we put it?
- The first (and most obvious) option was to store it in the database:
    - It’s structured information and the database is already a central part of the system
    - All instances read and write to the same database, ensuring consistency across the system.
    - Makes synchronization mechanisms unnecessary at the application level
    - Common data (e.g., configuration, metadata, usage counters) is instantly available to all instances.
    - New instances can start up and access the same information without replication or reloading
    - The database guarantees data persistence across restarts, deployments, or crashes
    - Databases handle concurrent access, locking, and transactions reliably.
    - Ensures updates to shared information are atomic and conflict-free
    - Avoids introducing additional systems, simplifies infrastructure and reduces maintenance overhead

TAP Stateless layout + Application state



How to share this information

- The TAP service was refactored to extract this information from the application itself.
- Now the question was: where do we put it?
- The first (and most obvious) option was to store it in the database:
  - It's structured information and the database is already a central part of the system
  - All instances read and write to the same database, ensuring consistency across the system.
  - Makes synchronization mechanisms unnecessary at the application level
  - Common data (e.g., configuration, metadata, usage counters) is instantly available to all instances.
  - New instances can start up and access the same information without replication or reloading
  - The database guarantees data persistence across restarts, deployments, or crashes
  - **Databases handle concurrent access, locking, and transactions reliably.**
  - **Ensures updates to shared information are atomic and conflict-free**
  - Avoids introducing additional systems simplifies infrastructure and reduces maintenance overhead

From Transactions to Analytics: Databases in ESDC

- ESDC relies in PostgreSQL databases
  - Fit for OLTP (Online Transactions Processing)
    - Ideal for small, frequent transactions
  - Not so fit for OLAP (Online Analytical Processing)
    - Best fit for data analysis with huge queries
- ESDC Archives have started to store big data
  - GAIA, Euclid, PLATO, …
- DB of choice for big data archives has been Greenplum
  - Very good choice for OLAP (analytical, huge queries)
  - Bad choice for OLTP (small, frequent transactions)

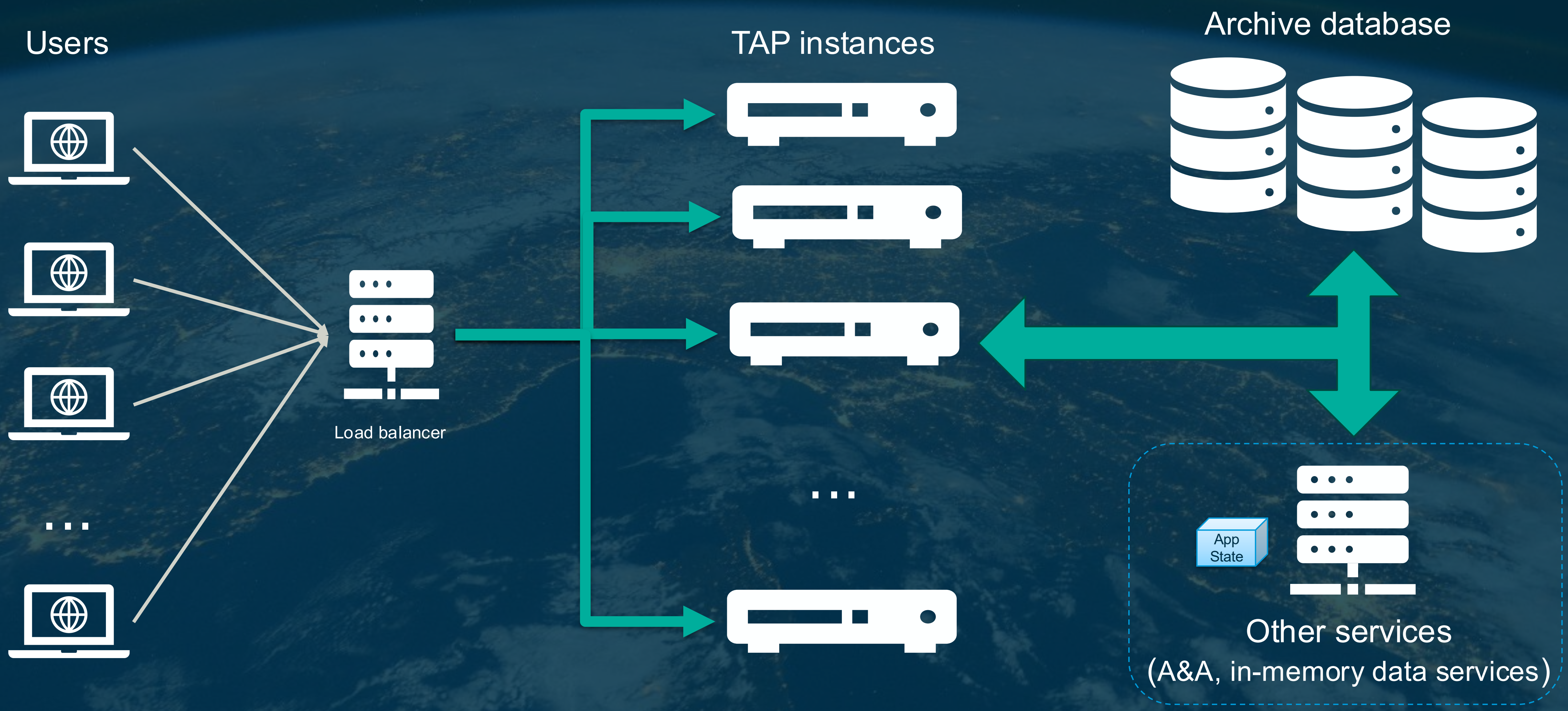
Why the Database Wasn’t the Right Place After All

In Practice, this approach was not as Effective as Expected:

- The CAS login protocol interfered with the Spring session handling, causing duplicate events to be created and stored in the database almost simultaneously.
- Some processes generated multiple events in parallel, and the database couldn’t handle them efficiently, resulting in concurrency conflicts and performance bottlenecks.
- On top of that, matching the speed of an in-memory approach proved difficult. For critical operations, even a small delay became more significant than expected.

What are now the alternatives?

TAP Stateless layout + Application state



In-Memory Databases: Fast, Reliable, and Scalable I

- Store and access data directly in memory for ultra-fast read/write operations.
- Use Cases:
  - Caching frequently accessed data
  - Session storage for web applications
  - Message brokering (pub/sub systems)
- Advantages:
  - Extremely low latency and high throughput
  - Reduces load on traditional databases
  - Supports high concurrency and parallel access
- Considerations:
  - Data is volatile unless explicitly persisted

In-Memory Services: Options and Considerations

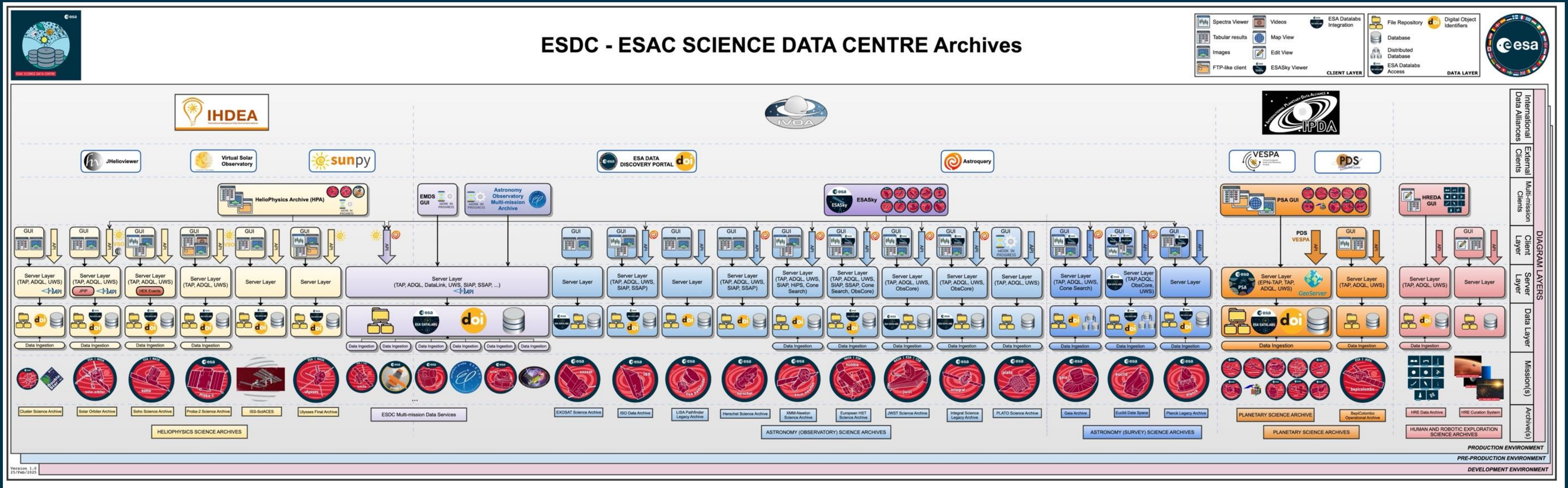
- Popular Services:
  - Redis – Key-value store, supports persistence, pub/sub, and advanced data structures.
  - Memcached – Simple caching for temporary data.
  - Hazelcast / Apache Ignite – In-memory data grids for distributed applications.

- Implications of Adding an In-Memory Service
  - New service added → introduces an additional point of failure.
  - Data strategy: Decide carefully what data goes in-memory vs. what stays in the database.
  - Management overhead: Start/stop, monitoring, and upgrades must be handled.
- Deployment strategy:
  - One instance per Archive
    - Isolates failures between archives
  - One instance for the entire ESDC
    - Reduces management burden.
    - This component is more critical to system reliability, although similar approaches are used for other services (e.g., LDAP for authentication)

System Tests — Still a Crucial Part of the Software

- Always include stress tests as part of the validation process:
  - Reproduce heavy-load scenarios to assess real-world behavior.
  - The system must remain robust under stress.
  - Performance degradation should stay within tolerable limits.
- Use cases should be tested in realistic environments:
  - Even the database version used during testing can have a significant impact.
- Performance considerations alone can cause a release to fail:
  - Define clear and measurable performance requirements early on.

What is the scenario we want to deal with?



Questions / feedback

Thank you for your attention