All Storage Must Be Managed!

The Small Catechism of Storage Management

- **Allocation**: All usage of storage space throughout a distributed system is explicitly allocated.
- **Requirements**: Workloads explicitly state their storage space requirements.
- **Policy**: Allocated storage space has a size limit (e.g., byte and object limits), reclamation policies, access control rules, and an owner.
- **Recursive**: Storage space allocations are recursive; an owner of a space can partition it into sub-spaces and assign allocations to others.

Kingfisher: Storage Management for Data Federations

This project is supported by National Science Foundation under Grant OAC-2209645. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Kingfisher

- The idea of the project is to demonstrate the impact of our small catechism of storage management.
  - The project aims to implement this in a reference platform and showing the value for a few science drivers.
- The reference platform is:
  - Using **XCache** (XRootD configured in caching proxy mode, particularly in OSDF) as the space which will be managed.
  - **HTCondor** to communicate workload needs.
  - A new library, **LotMan**, to track allocations and policies.

LotMan

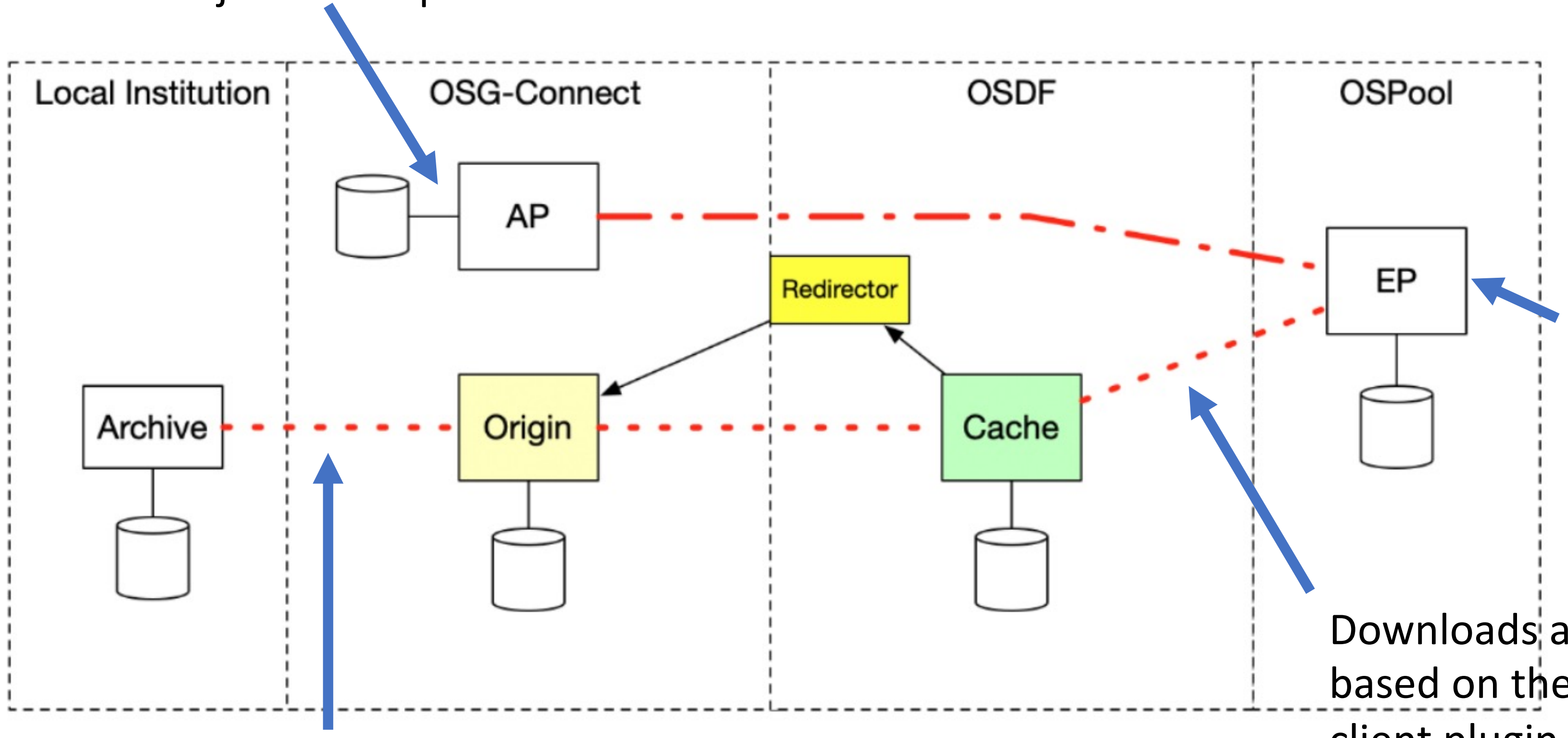
- The LotMan library is the new piece of software:
  - Exposes API to allow creation of “lots”.
  - Each “lot” has an owner, associated storage, reclamation policy, and ACLs.
  - LotMan is only the *accountant*. It does not measure use, move files, or delete files. It is expected to plug in to a larger system (HTCondor, XRootD) that provides the usage information.
- Example reclamation policies:
  - **Classic cache**: Delete any files when needed. (Typical cache setup)
  - **Temporary buffer**: Delete all files after time next Tuesday.
  - **Managed cache**: Delete files, as needed, until the lot is under 1TB of use.

LotMan – toward a REST API

- Today, LotMan provides a C API.
- Next up: Python API (plan is to use CFFI to call into the C library)
- The third phase (later this year), we want to develop a REST API as a XRootD web server plugin to will allow for remote management of lots:
  - Administrator creates top-level lots based on cache policy.
  - Associate a top-level lot with a token issuer.
  - Token issuers can subdivide and set policy for the cache.

Transfer picture as it exists today

The "Access Point" – where all the jobs are kept



The "Execution Point" – where job instances are executed.

Upload to OSDF origin often by hand (scp and friends)

Downloads are based on the OSDF client plugin to HTCondor.

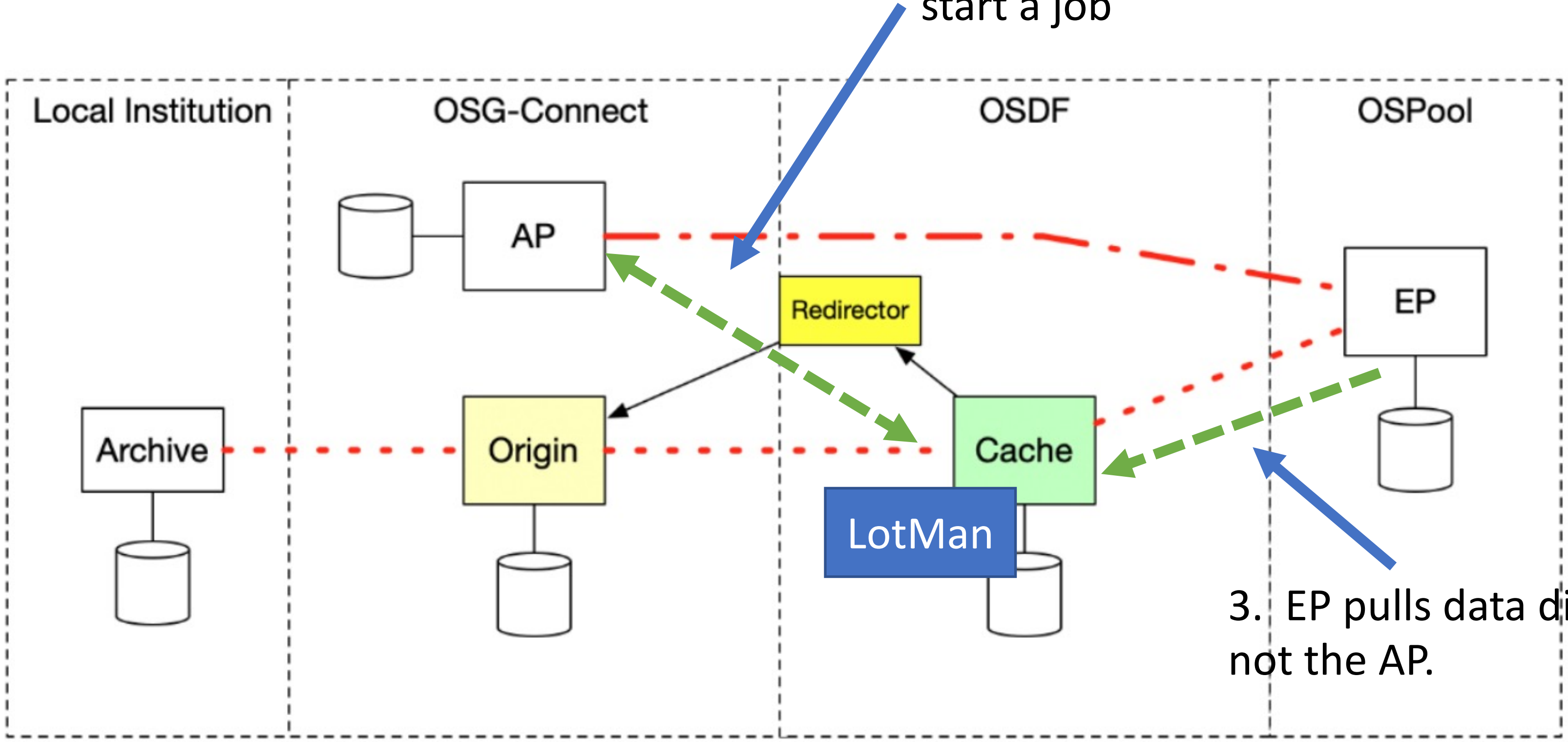
Use cases: AP-managed stage-in

- The AP knows:
  - Where the job is going to run,
  - The nearby caches,
  - what files the job needs,
  - And how many similar jobs it has.
- We would like to leverage this knowledge to push data into XCache.
  - A ‘temporary buffer’-type lot can be allocated by the AP and let expire when the workflow ends.
  - Note: in this case, there is no “origin” where the data lives!

Goal for “tomorrow”

1. AP uses the REST API to create a lot at the cache for a workflow.

2. Pushes data into a lot once it’s ready to start a job



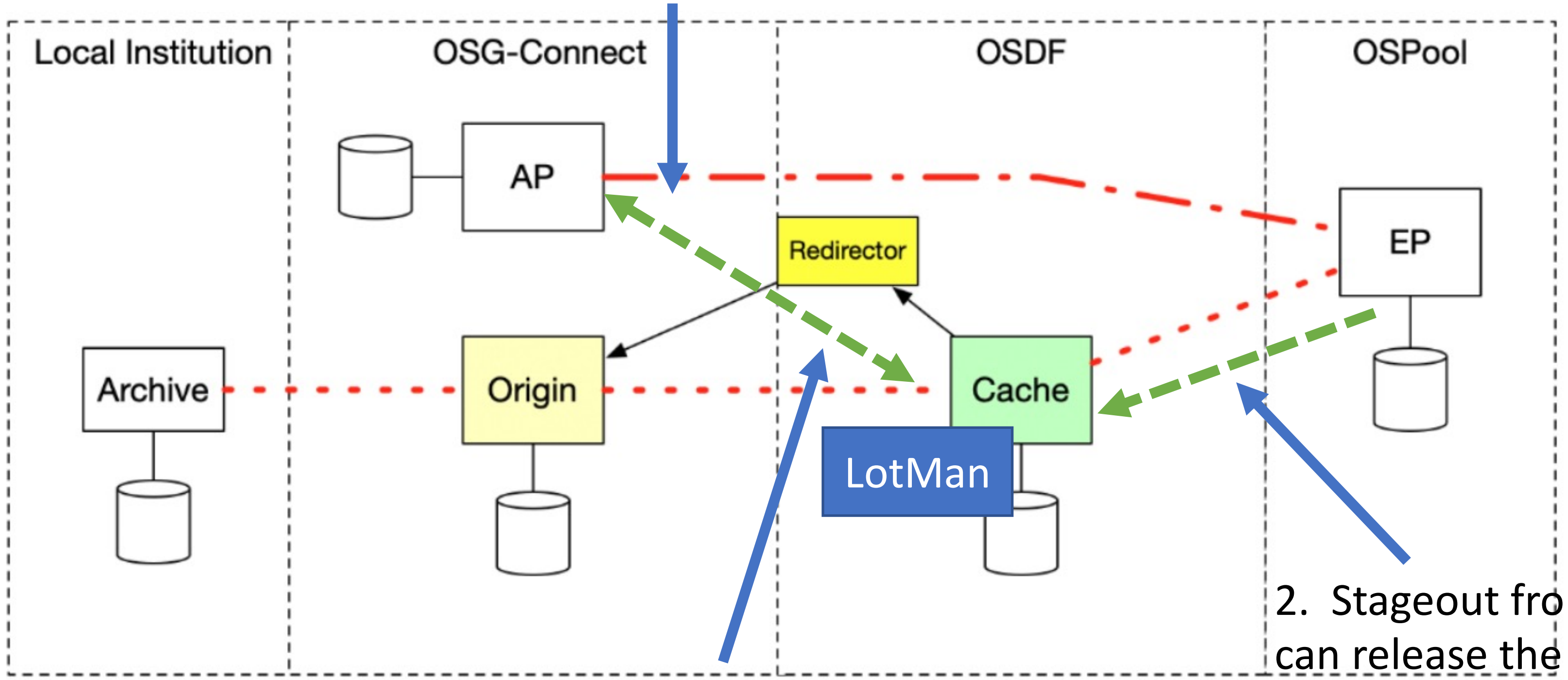
3. EP pulls data directly from the cache, not the AP.

Use cases: Managed Stageout

- Currently, the EP must “hang out” until the data is moved back to the origin.
  - We potentially could start running another job.
  - But if we’re preempted while (slow) stageout is occurring … too bad, job lost.
- Opportunity:
  - Use the cache as a temporary buffer.
  - As soon as the object is moved to the cache, the EP can disappear.
  - The AP still needs to coordinate the transfer from the cache to the origin; the job is incomplete until then.
- Interesting challenges:
  - We’ve now created a potential “mismatched rate” problem. The AP must ensure it doesn’t fill up the buffer with results faster than it can move the data back.

Goal for "tomorrow"

1. AP ensures there's enough space in the lot to hold the job output



2. Stageout from EP to cache. Job can release the CPU resources and 'completes' when the output makes it to the resources.

3. Data is transferred from cache to AP (or origin).

Use cases: Multi-tenant caches

- The OSDF, today, has multi-tenant caches:
  - Each unique entity (OSPool, IGWN, DUNE, etc) using the caches can potentially thrash the entire cache.
- Goal for this use case is the cache administrator configures lots per entity in the top-level namespace. E.g., if we have a 50TB cache:
  - OSPool gets 10TB space.
    - OSPool's monitoring directory gets 1GB space.
  - IGWN can use 30TB.
  - DUNE can use 10TB
- Impact: If DUNE starts streaming 500TB of data through the cache, it will thrash its own area but *not* evict the IGWN data.

Kingfisher

- The project is just getting ramped up!
- We aim to use ideas from "translational CS" – we use the OSDF as a platform for our ideas and feedback from the user community to guide the research.
  - Aim to see the impact on the OSPool user community from the 3 outlined use cases.
- The intent is to build on the existing base – XCache, HTCondor – and enhance them with a library (LotMan) implementing the new concepts.

All Storage Must Be Managed!