Outside-In:

O.K.I. Open Service Interface Definitions (OSIDs) as a Native Framework API

Adam Franco - Middlebury College
Alex Chapin - Middlebury College

- Background
- What We Have Built
- Challenges
- Where We Are Going

Background

Our Position in 2003

- We had built 4 curricular systems, and wanted to build more.
- Newer systems grew out of older systems, with commonly used code copied over and then ‘tweaked’.
- Fixes and ‘tweaks’ broke code compatibility and led to many systems with similar bugs.

Why use a framework?

- No reinventing of common parts in each application.
  authentication, group management, etc
- One place to look for bugs in common parts.
- Easier data sharing between applications
  create a group once, it is usable in all apps

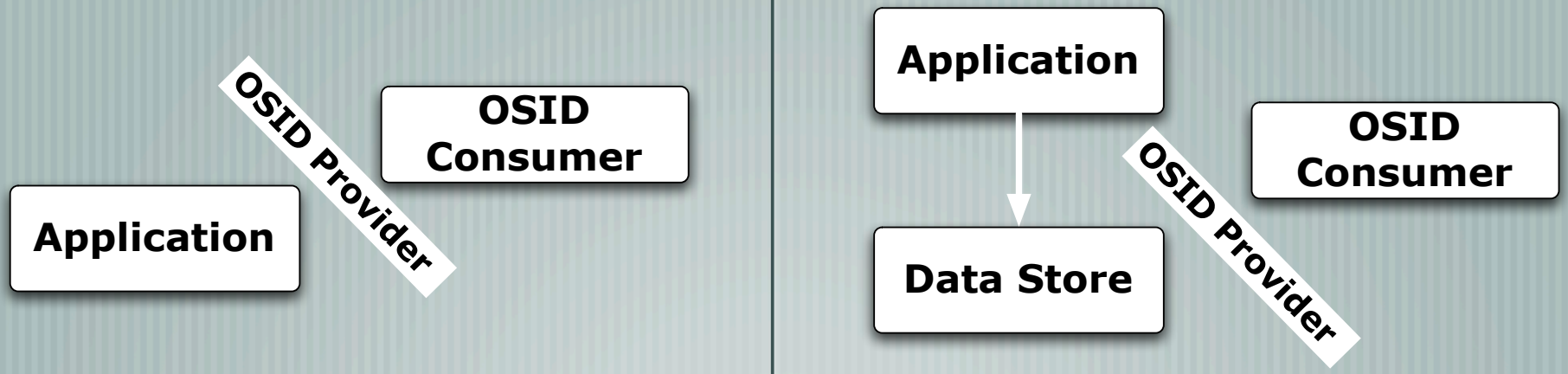
Why did we build our own framework?

- Existing frameworks were focused on execution control and presentation.
- We needed a set of common services as well.

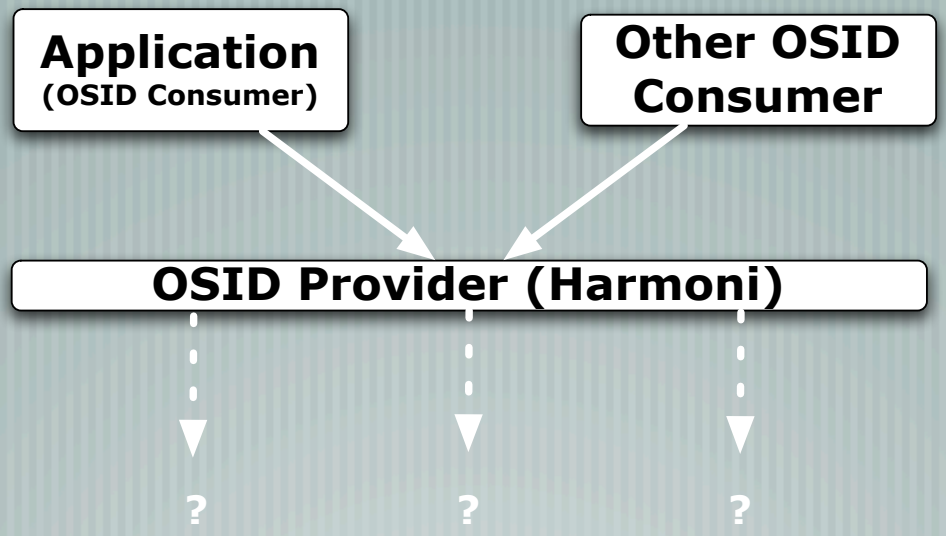
Why use the O.K.I. OSIDs as a framework API?

- They cover all major service areas.
- Services are designed together as a set without overlapping responsibilities.
- OSIDs are a standard – not a one-off API – increasing the chances of interoperability with 3rd parties.

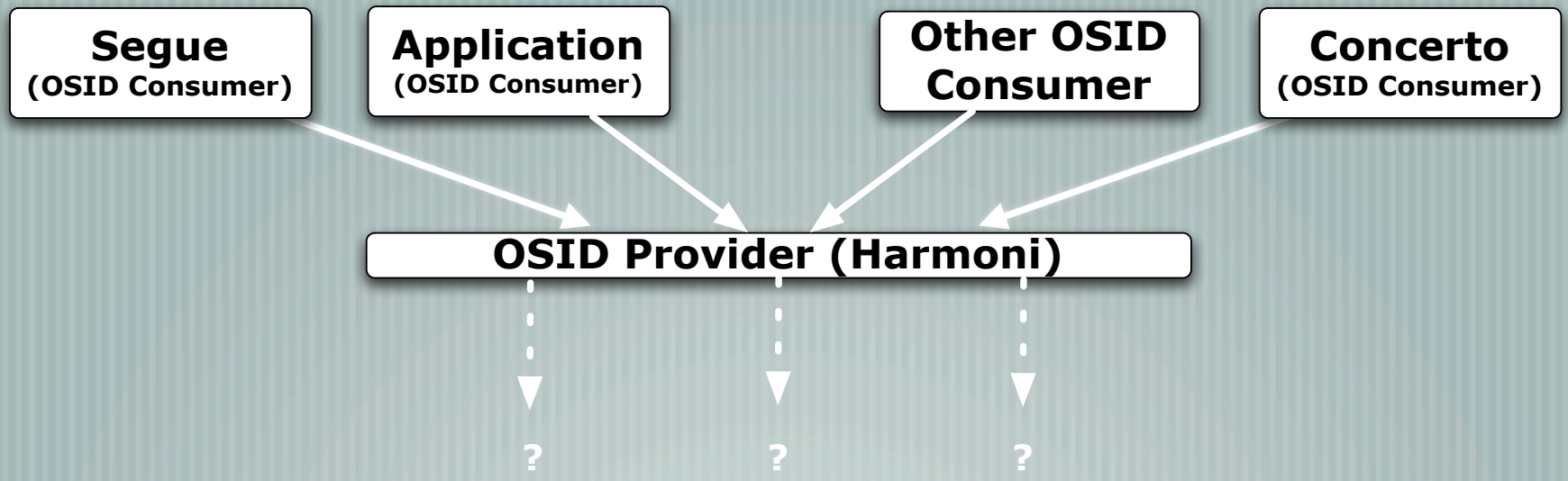
Common Interoperability Patterns



OSIDs as Framework API



OSIDs as Framework API



- Integration point always available, no wrapper needed.
- Provider is general enough to support many applications.
- Enable new storage options or performance characteristics by exchanging OSID Provider implementations.

What We Have Built

Harmoni

- A service-oriented framework
- Includes implementations of most OSIDs as well as other services (tagging, image processing, etc).
- Includes optional controller and UI layout/theming systems.

http://harmoni.sourceforge.net

Two curricular applications built on *Harmoni*

Concerto

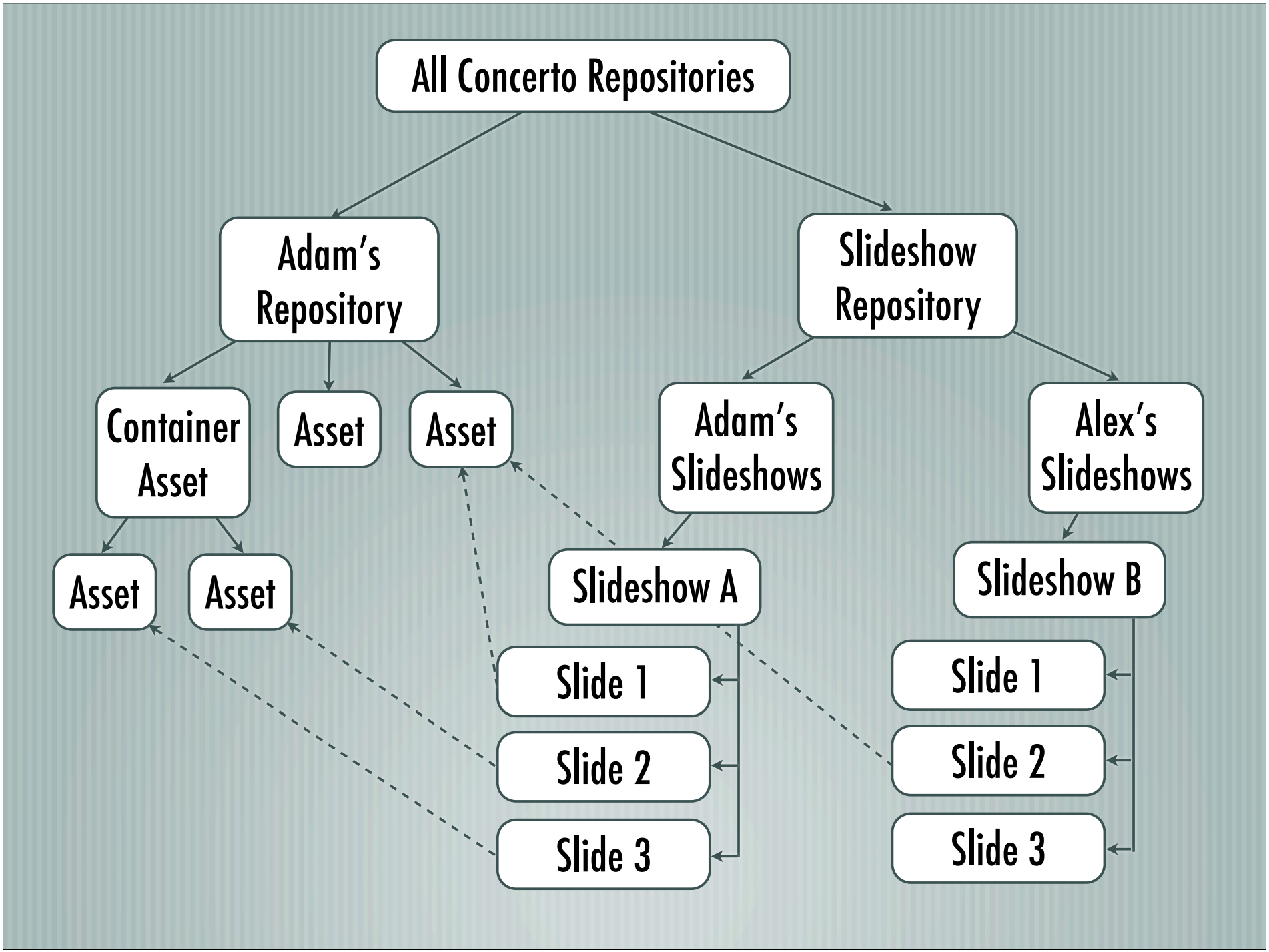
- Multimedia D.A.M.
- Catalog with Dublin Core, VRA, and other metadata
- User-based tagging
- Build slideshows of media
- CMS with curricular focus

Segue

- Build web sites/blogs
- Collaborative editing
- Threaded discussions
- Pluggable content

Concerto

- Media-Asset hierarchy is created by users for their needs (maps directly to the Repository OSID).
- Users can choose and/or define metadata schemas
- Slideshows are a separate Asset hierarchy with a field for the ID of the media Asset to display.

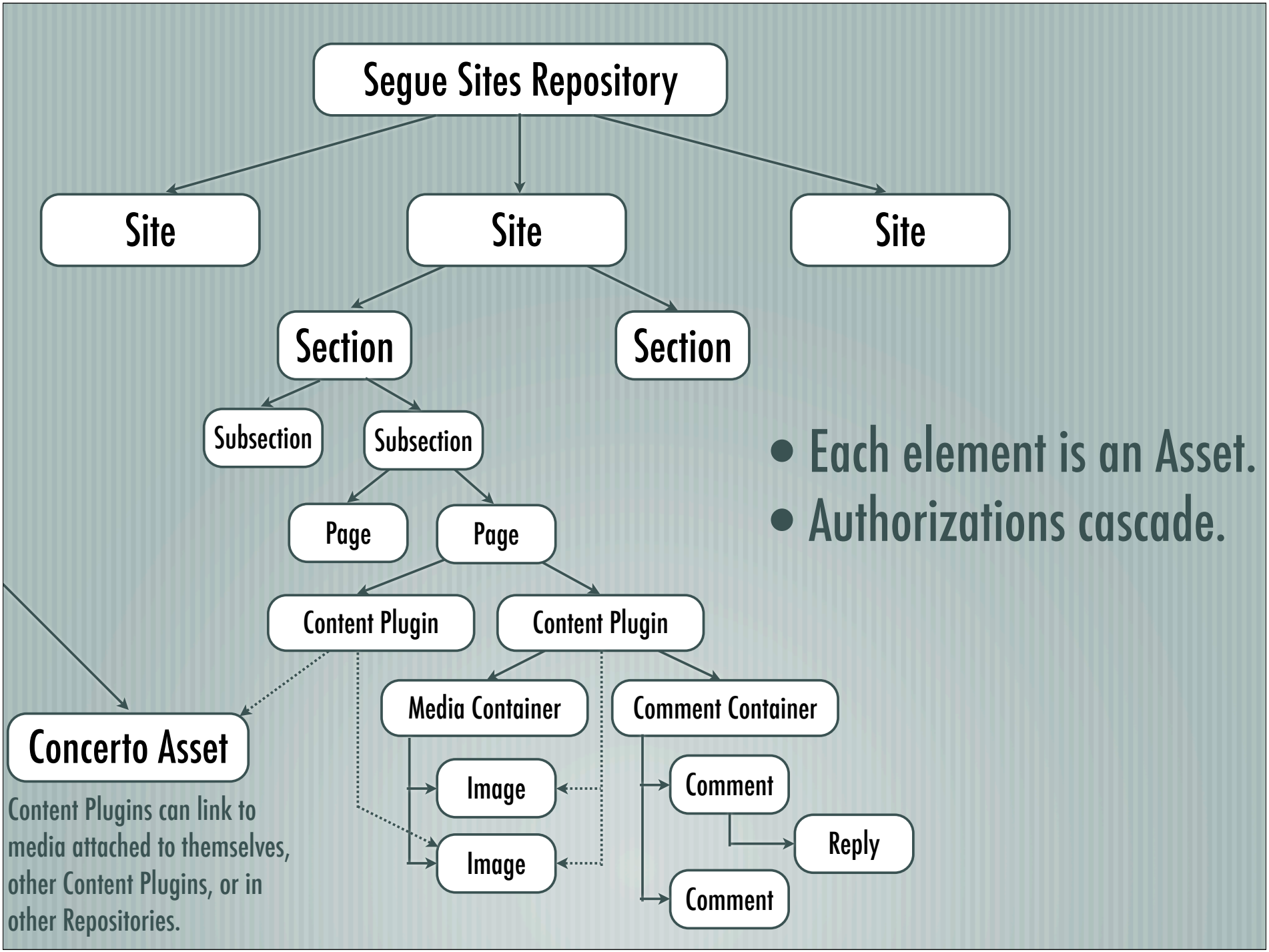

All Concerto Repositories
Adam's Repository
Slideshow Repository
Container Asset
Asset
Asset
Adam's Slideshows
Alex's Slideshows
Asset
Asset
Slideshow A
Slideshow B
Slide 1
Slide 2
Slide 3
Slide 1
Slide 2
Slide 3

Some OSID advantages

- Storage and authorization are taken care of by the Repository.
- Unique IDs allow easy building of slideshow tools.
- User interface is the only remaining challenge.
- Well-defined integration point for other systems.

Segue

- Web site hierarchy is stored as Asset hierarchy
- Navigational nodes store their layout information in Asset content.
- Micro-Content (plugin instances) store most data in Asset fields.
- Media are Assets attached to a micro-content Asset at time of upload.
- Threaded discussions are Assets in the same site hierarchy.

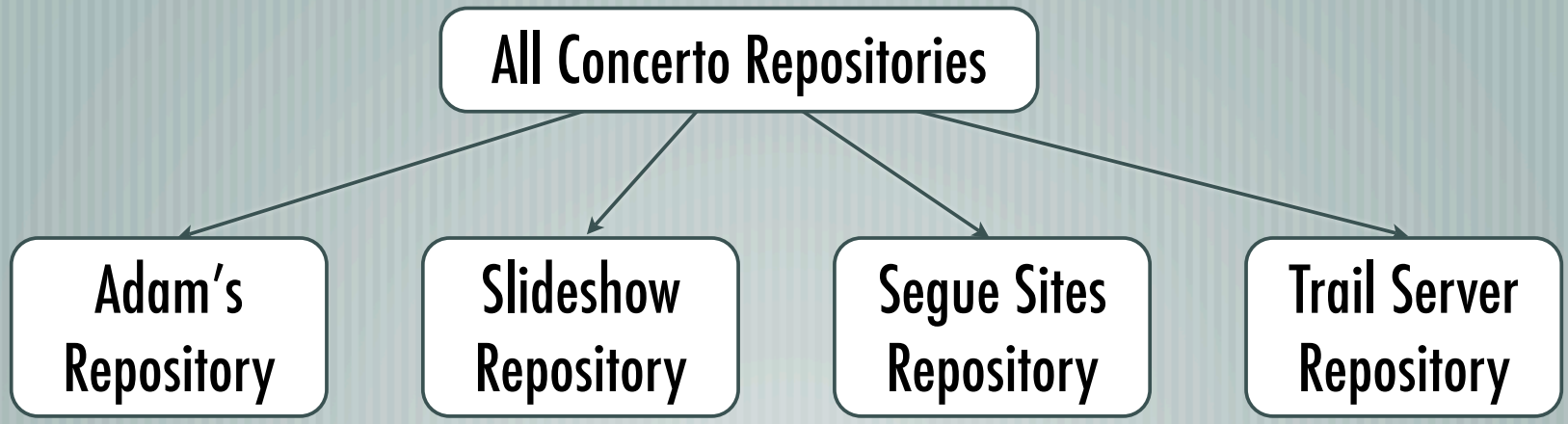

Segue Sites Repository
Site
Site
Site
Section
Section
Subsection
Subsection
Page
Page
Content Plugin
Content Plugin
Concerto Asset
Media Container
Comment Container
Image
Image
Comment
Reply
Comment
Content Plugins can link to media attached to themselves, other Content Plugins, or in other Repositories.
• Each element is an Asset.
• Authorizations cascade.

Some OSID advantages

- Hierarchical authorizations are built in and can be enforced during any access.
- *Segue* data can be accessed for read and write from other applications.
- Browsing media in external repositories is as easy as browsing local media.
- Consistent structure for all data objects.

Sharing Data

- Use Concerto to browse data Assets of Segue and other applications.
- Add other Assets to Concerto Slideshows.
- Use Concerto to add metadata to Assets.



Challenges

OSID v2 Issues
(all fixed in v3)

- Authentication process missing key interactions.
- Numerous "out-of-band agreements"
  i.e. search types and syntax
- Many operations force the retrieval of unneeded data.
- Java-based API didn't always mesh with our PHP4 environment. (*lack of exceptions*)

Other Challenges

- Limited in-language community - no other implementations to test against.
- Delayed gratification - building a framework is a lot of work.

Where We Are Going

XML-RPC OSID Bindings

Talk across networks and/or programming languages.