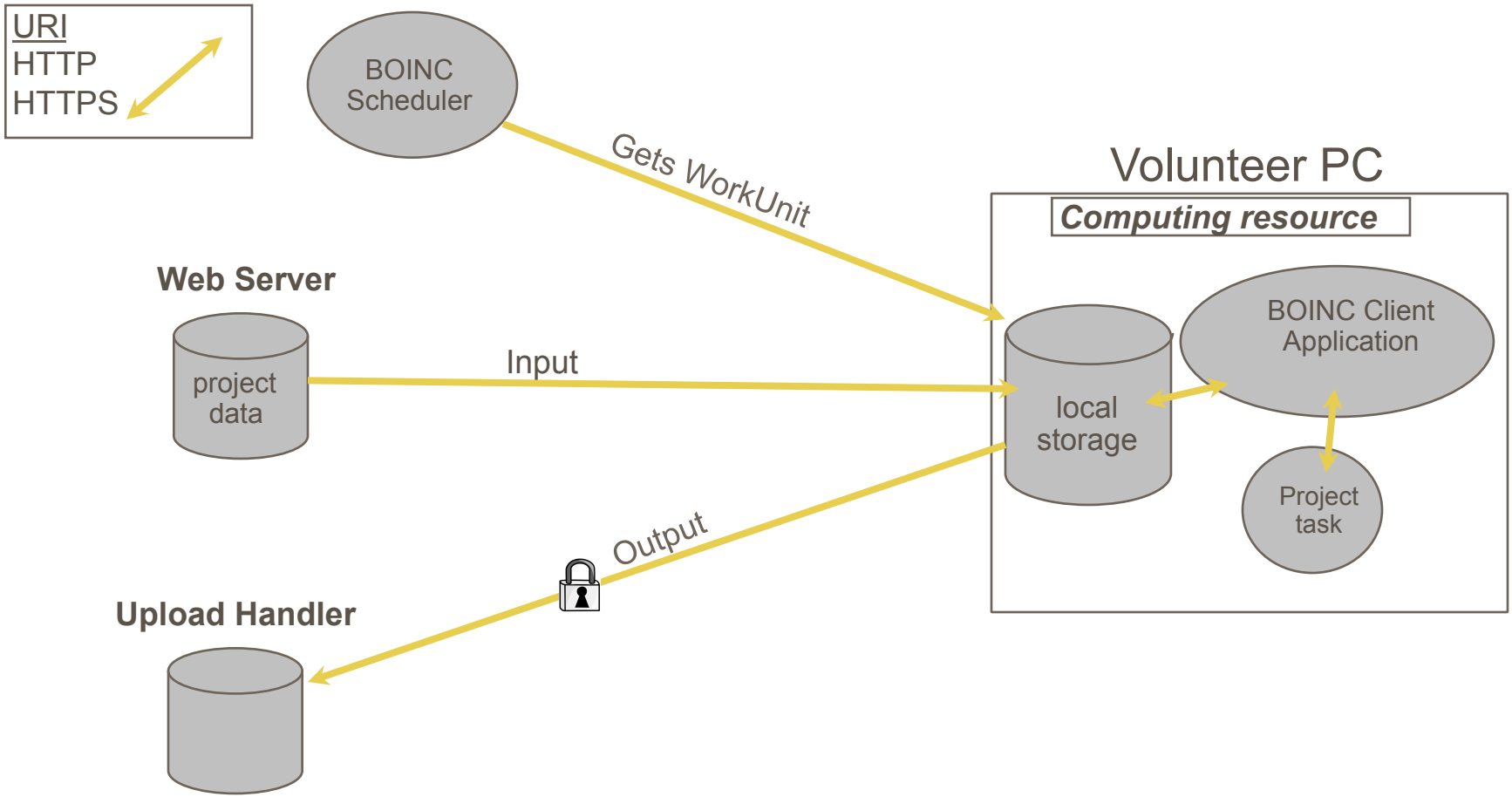
Attic Data Distribution Framework

Presented By: **Ian Kelley**

- School of Computer Science
  Cardiff University, United Kingdom
- E-mail: I.R.Kelley@cs.cardiff.ac.uk

BOINC Data Access



Typical BOINC Applications

- **Rosetta@Home**
  - Size of a Work Unit: 3 MB
  - Processing Time of a Work Unit: 3h
  - Size of Initial Data: 17 MB
- **SETI@Home**
  - Size of a Work Unit: 340 KB
  - Processing Time of a Work Unit: 2h
  - Size of Initial Data: 2.5 MB
- **Einstein@Home**
  - Size of a Work Unit: 3.2 MB
  - Processing Time of a Work Unit: 5h
  - Size of Initial Data: 40 MB

EDGI Data Requirements

- **Fusion Physics Application**
  - Institute for Biocomputation and Physics of Complex Systems
  - Execution time: ~30 minutes
  - Input files: ~10 MB
- **Material Science Applications**
  - G.V. Kurdyumov Institute for Metal Physics
  - Execution time: ~30 min per scenario
  - Input files: 1 – 10 MB
  - Jobs: 10^3 – 10^4 per day
- **Signal-and Image Processing**
  - Forschungszentrum Karlsruhe
  - Execution time: 4 days
  - Input files: ~20 GB

Desktop Grid Data Issues

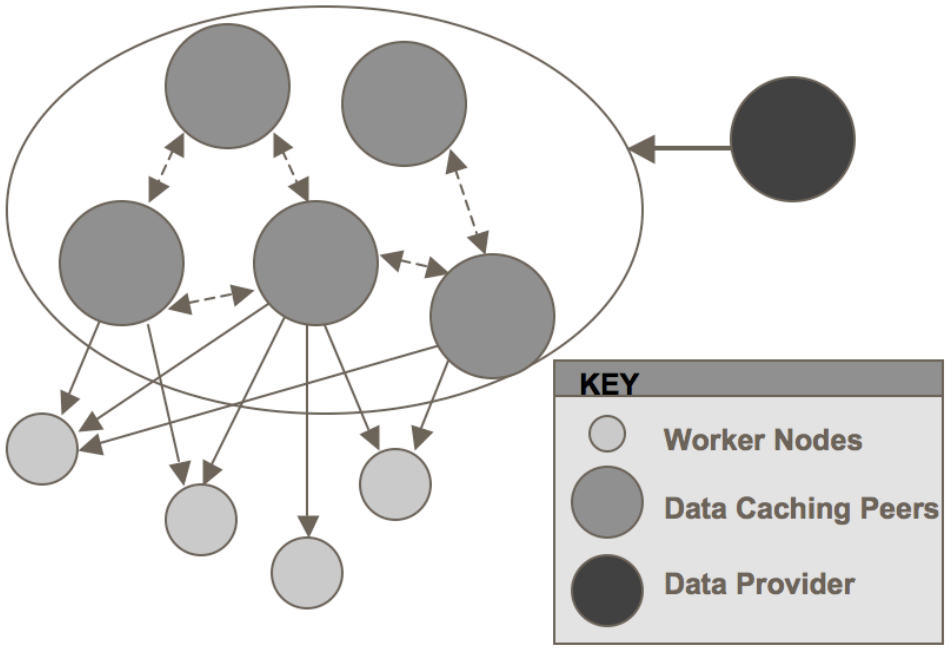
- Overall bandwidth requirements can be high, especially with replicated jobs
- Project's need persistent data webserver, and potentially a N mirrors to balance load
  - For smaller groups servers might be hard to maintain or mirror
  - For Service Grids, data might be restricted and it would be useful to have a staging ground for DG data.
- Network peak demand problem

➢ Possible to construct a "P2P" system using clients and/or (potentially dynamic) set of project/partner servers to serve and cache input data

Desktop Grid Data Issues

- General architecture requirements
  - Need to protect end-users and have opt-out system
    - compulsory open ports on all workers is not possible
  - Protect the project's data
    - may want limited caching on any given peer to limit exposure
    - need to ensure data integrity and potentially have authentication techniques for data cachers
  - Beneficial to support different network topologies (WAN, LAN)
  - *These requirements discount many established P2P systems such as BitTorrent*

Distributed Data Centers



- ***Data Caching layer***
  - Data Caching peers exchange data amongst themselves and serve client machines
  - Authentication can be turned on between Data Cachers (Data Centers)

Attic
(Previously ADICS)

- Overview of Attic P2P architecture
  - History
  - Overview
  - Message types
  - Protocols
  - Security
  - Features
- Scenario/Use-case outline
  - making data available to DG from Attic network

History

- Started as part of a UK EPSRC proposal in 2005
  - Focus on providing data distribution inside Desktop Grids, with target community being Einstein@home

EPSRC
Engineering and Physical Sciences Research Council

- Continued development under EU FP7 EDGeS (2008-2010) and EDGI (2011-2012) projects
  - Need to provide a way to support data distribution within Desktop Grids for load balancing
  - Additional focus on moving Service Grid data and jobs to Desktop Grids, and legacy application support

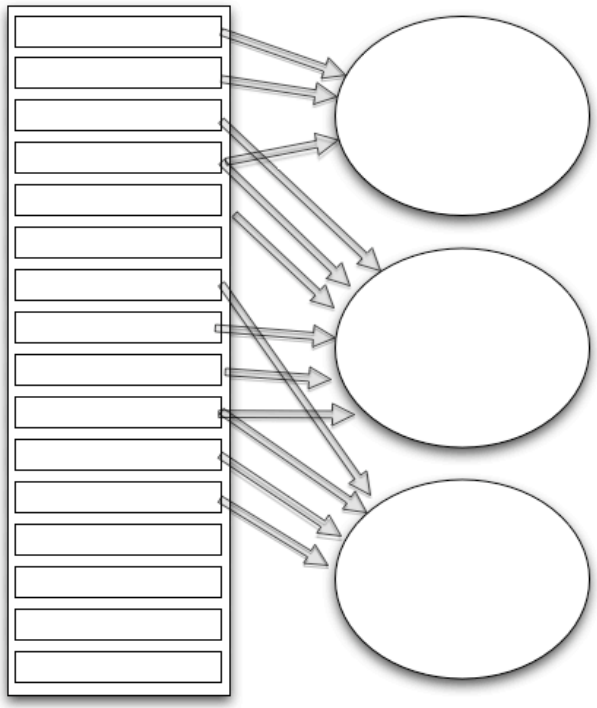
EDGeS EDGI

Project Website: *http://www.atticfs.org*

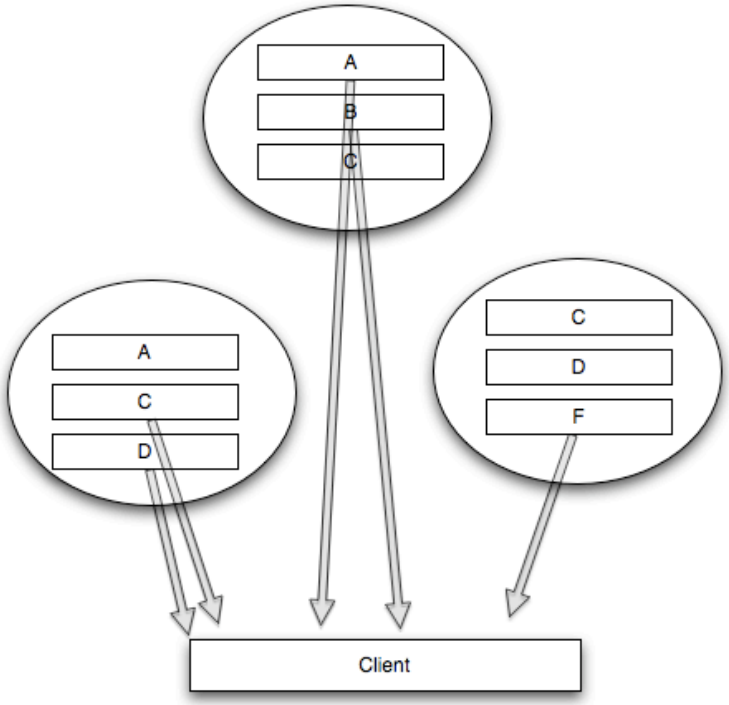
EDGI

Attic

Files CAN be split into individual chunks for distribution to data caching layer.

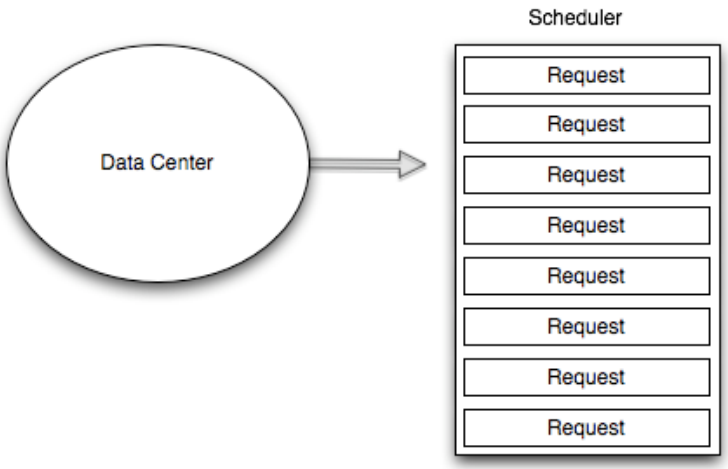


Clients CAN download different parts of the file from multiple data centers.

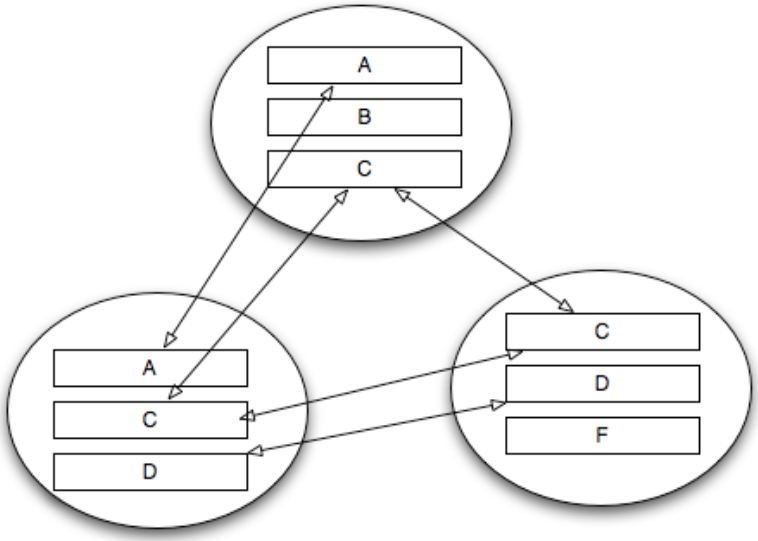


Attic

Data cachers contact ascheduler to receive replication requests.

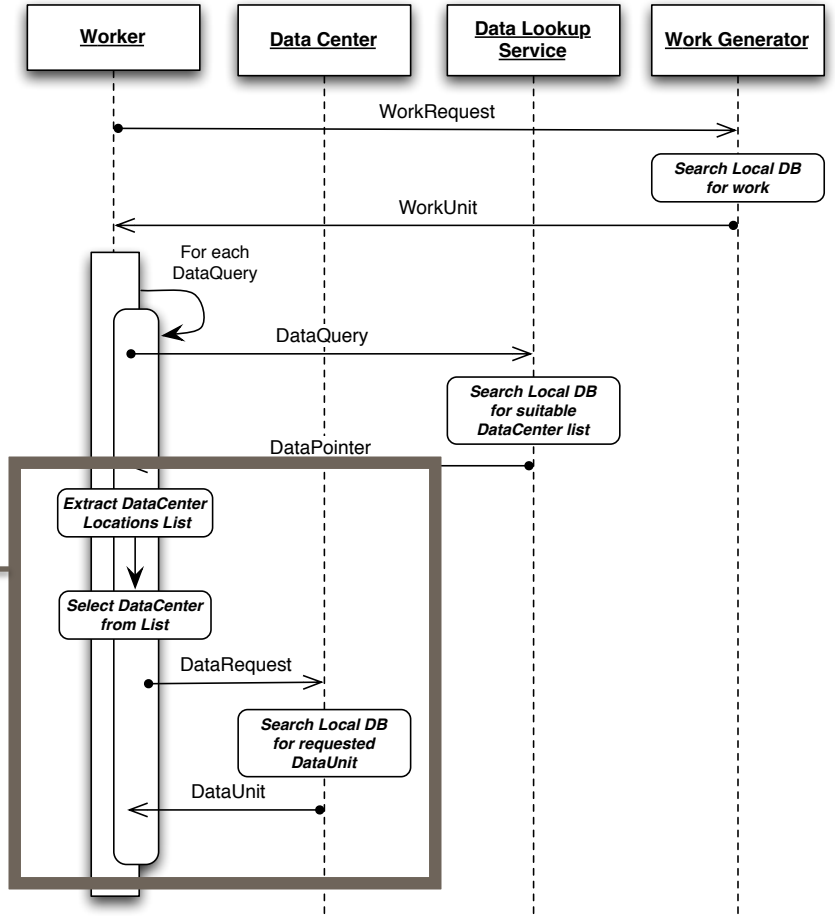


They then download from one-another to propagate data on the network

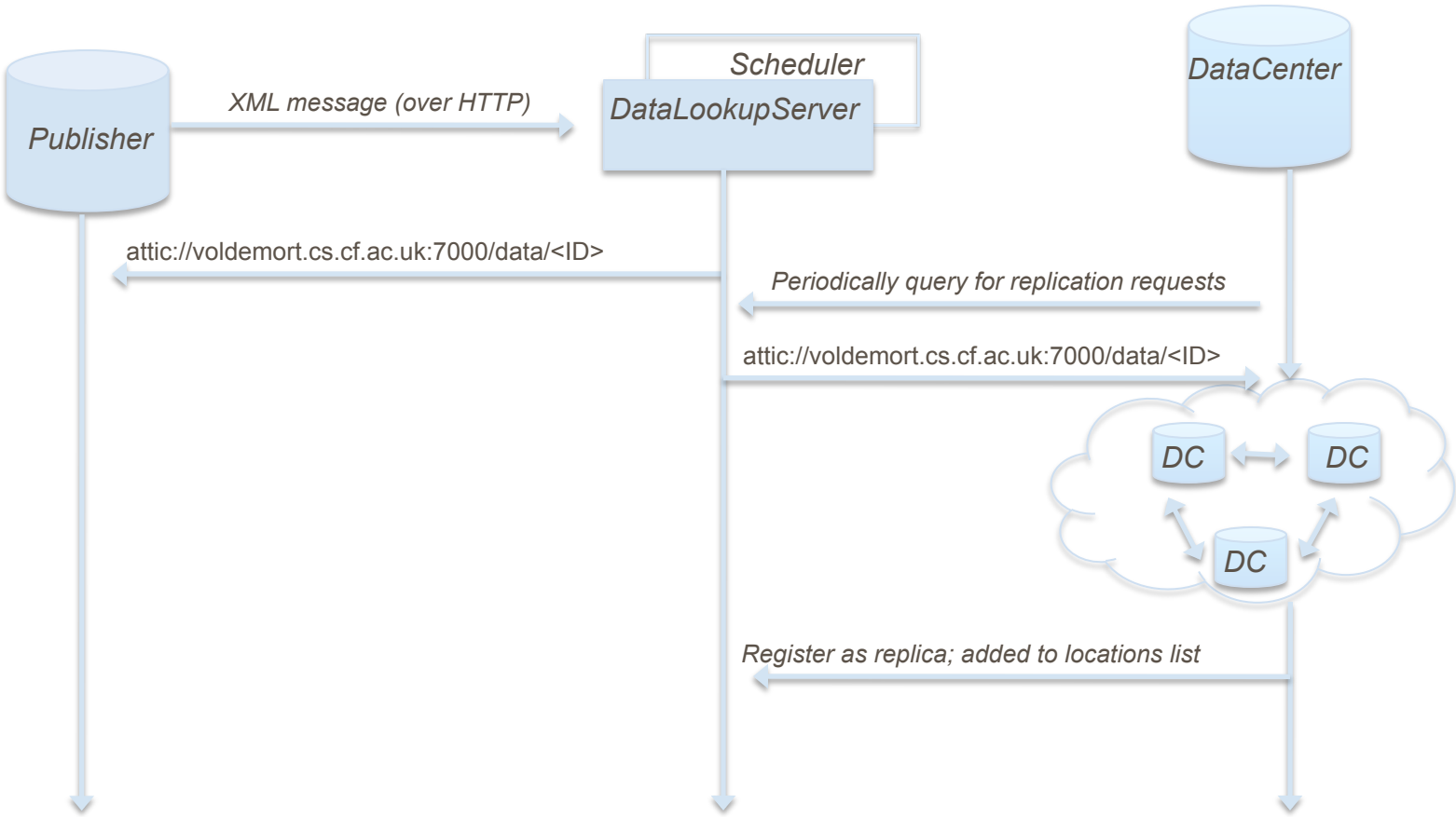


Attic

- Network participants distribute data
- Opt-in strategy
- Restricted publication of data
- URI scheme
- File swarming
  - By simultaneously downloading different chunks from multiple DataCenters



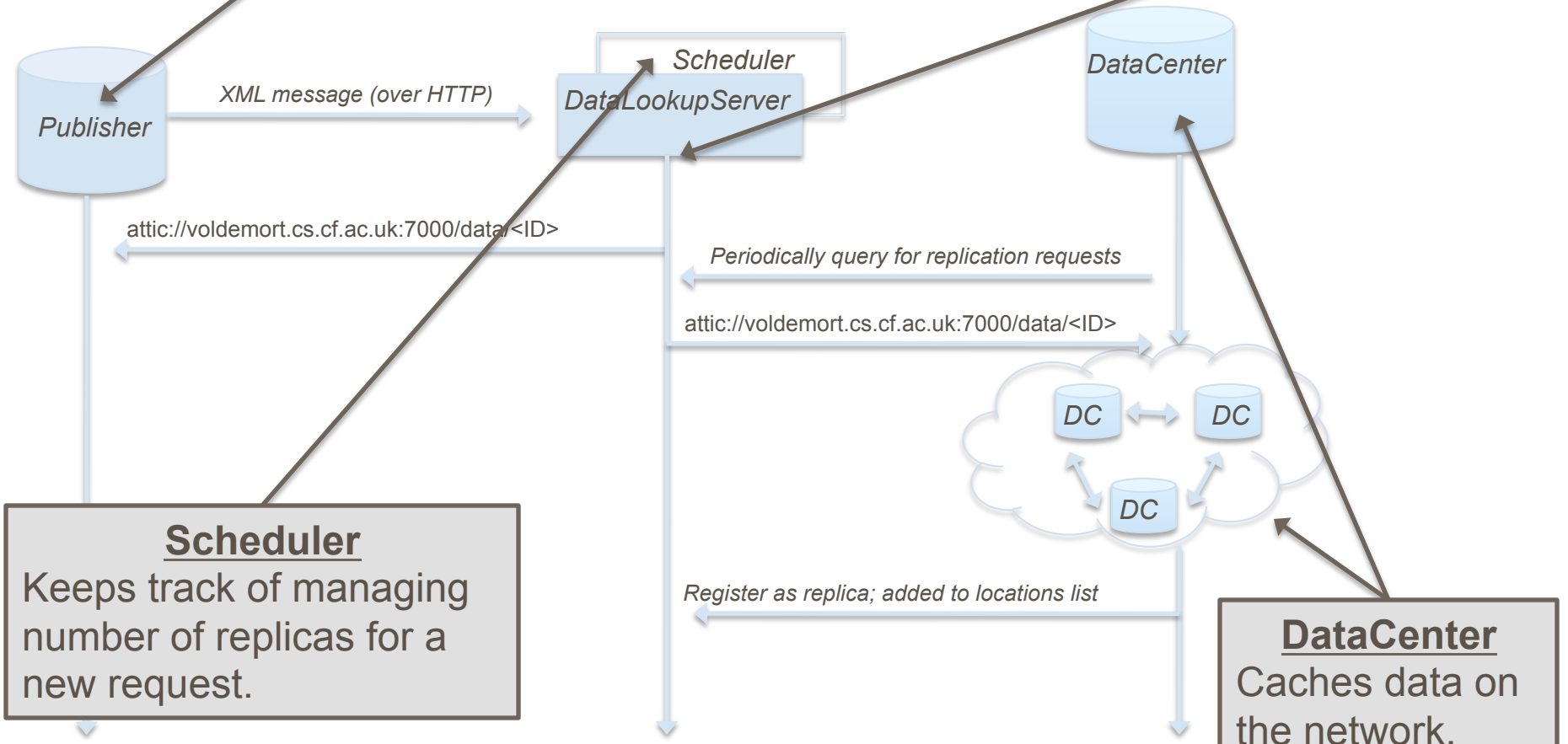
Component Overview



Component O

Publisher
Any entity that publishes a file

DataLookupServer
Manages number of replicas for a new request and responding to cache requests



Scheduler
Keeps track of managing number of replicas for a new request.

DataCenter
Caches data on the network.

Terms

- DLS – Data Lookup Service
  - receives requests to publish data
  - receives requests to cache data
  - does not store any data, only keeps mappings between endpoints and data
  - acts as a scheduler as well – controls exposure of data according to constraints defined by the publisher
- DP – Data Publisher
  - publishes an advert to the DLS about data
  - typically the DP is also a seed endpoint (but not always)
- DC – Data Center
  - requests data references from the DLS
  - caches data from other endpoints
- Worker
  - downloads data from DCs for processing.

Message Types

- DataDescription
  - contains metadata, e.g., name, description, project
  - file data, e.g. size, MD5, and a list of chunks with byte ranges and MD5s
- DataAdvert
  - contains DataDescription
  - Constraints, e.g., replication count
  - Used when publishing data
- DataQuery
  - contains Constraints
  - Used when Querying for data to cache/replicate
- DataPointer
  - contains DataDescription
  - List of endpoints associated with the description
  - Returned to a query for data
  - The data structure pointed to by an attic:// URL

Message Types

- https://voldemort.cs.cf.ac.uk:7048/dl/meta/pointer/dceae487-bb18-4dfd-9391-3a4b701b1fb7

```
- <DataPointer>
  - <DataDescription>
      <id>dceae487-bb18-4dfd-9391-3a4b701b1fb7</id>
      <name>dceae487-bb18-4dfd-9391-3a4b701b1fb7.dat</name>
      <project>edges</project>
      <description>Test file</description>
    - <FileHash>
        <hash>d6a5f4aae746e18c92f18eaba9d77c61</hash>
        <size>6435839</size>
      - <Segment>
          <hash>44bc74bb4d6225b8b8e68ea6848a57</hash>
          <start>0</start>
          <end>524287</end>
        </Segment>
      - <Segment>
          <hash>bf5b8da3143d769241b21675347691</hash>
          <start>524288</start>
          <end>1048575</end>
        </Segment>
```

Message Types

- https://voldemort.cs.cf.ac.uk:7048/dl/meta/pointer/dceae487-bb18-4dfd-9391-3a4b701b1fb7

```
- <Segment>
    <hash>cdabbd2444a8b3c182a69528cb119c1</hash>
    <start>5767168</start>
    <end>6291455</end>
  </Segment>
- <Segment>
    <hash>1e6fe5fa73723a1cc3b02f8b5a3cc3d5</hash>
    <start>6291456</start>
    <end>6435838</end>
  </Segment>
 </FileHash>
</DataDescription>
- <Endpoint>
 - <url>
    https://d220.cs.cf.ac.uk:7049/dp/data/dceae487-bb18-4dfd-9391-3a4b701b1fb7
  </url>
 </Endpoint>
</DataPointer>
```

Message Types

- https://voldemort.cs.cf.ac.uk:7048/dl/meta/pointer/dceae487-bb18-4dfd-9391-3a4b701b1fb7

```
      - <Segment>
          <hash>1e6fe5fa73723a1cc3b02f8b5a3cc3d5</hash>
          <start>6291456</start>
          <end>6435838</end>
        </Segment>
      </FileHash>
    </DataDescription>
  - <Endpoint>
    - <url>
        https://d220.cs.cf.ac.uk:7049/dp/data/dceae487-bb18-4dfd-9391-3a4b701b1fb7
      </url>
    </Endpoint>
  - <Endpoint>
    - <url>
        https://electricline.cs.cf.ac.uk:7047/dc/data/dceae487-bb18-4dfd-9391-3a4b701b1fb7
      </url>
      <meta>https://electricline.cs.cf.ac.uk:7047/dc/meta</meta>
    </Endpoint>
</DataPointer>
```

Message Types

- https://d220.cs.cf.ac.uk:7049/dp/meta/filehash/dceae487-bb18-4dfd-9391-3a4b701b1fb7

```
– <FileHash>
    <hash>d6a5f4aae746e18c92f18eaba9d77c61</hash>
    <size>6435839</size>
  – <Segment>
      <hash>44bc74bb4d6225b8b8e68ea6848a57</hash>
      <start>0</start>
      <end>524287</end>
    </Segment>
  – <Segment>
      <hash>bf5b8da3143d769241b21675347691</hash>
      <start>524288</start>
      <end>1048575</end>
    </Segment>
  – <Segment>
      <hash>e69da54b2e42c423364640ad490dd4</hash>
      <start>1048576</start>
      <end>1572863</end>
    </Segment>
```

File Chunk info available from meta endpoint

Message Types

- Once a Data Center has downloaded the data and notified the Data Lookup Service, it appears in the DataPointer.
  - i.e., it gets added to the replica list
- The metadata endpoint is where clients can get meta info about data from a Data Center
- Note: the seed does not provide a metadata endpoint
  - Therefore it becomes a fallback endpoint during downloading
  - As more DCs get the data, the seed becomes redundant

Protocols

- Uses HTTP(S) for all exchanges
  - message and data
  - uses HTTP byte ranges to specify chunks
- Message serialization
  - default serialization is JSON (JavaScript Object Notation)
  - also XML (e.g., for demo)
  - JSON is about 1/3 to 1/2 as verbose as XML
    - but still Unicode

Protocols

- Why HTTP?
  - Attic is about data. HTTP is good at data.
  - allows nodes to take part transparently, for example a server without knowledge of Attic may be used as a fall-back during downloading. It exposes no metadata, but responds to byte range requests
  - easy integration with other systems, e.g., BOINC uses curl libs.
  - Allows use of common libraries to directly download data and/or build new clients/servers

Security

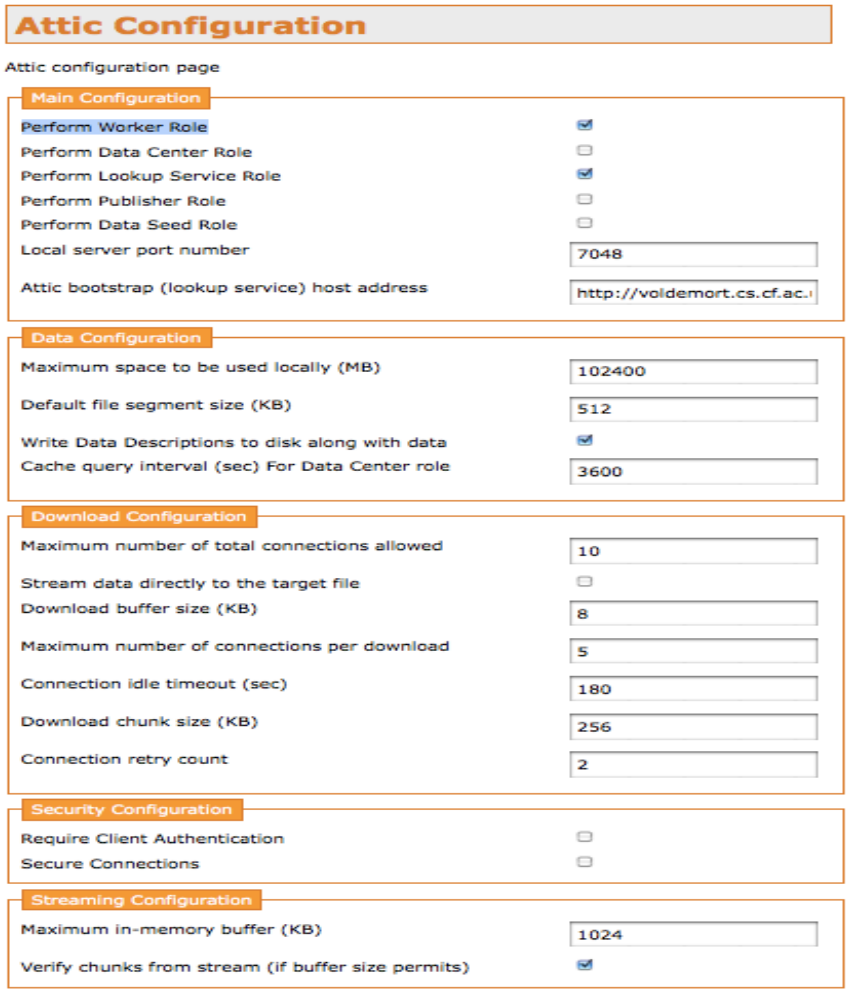
- Authentication (optionally enabled) uses X.509 certificates with TLS
  - mutual
- Authorization is done using the Distinguished Name (DN) in the peer's (e.g., DC) certificate
  - Identities based in DNs are mapped to actions, e.g., PUBLISH, CACHE
  - For example, a Worker may only need a certificate signed by a CA trusted by a DC to download from that DC
  - But a DC may need the above, as well as its DN mapped to the CACHE action on the DLS in order to cache data.

Download Features

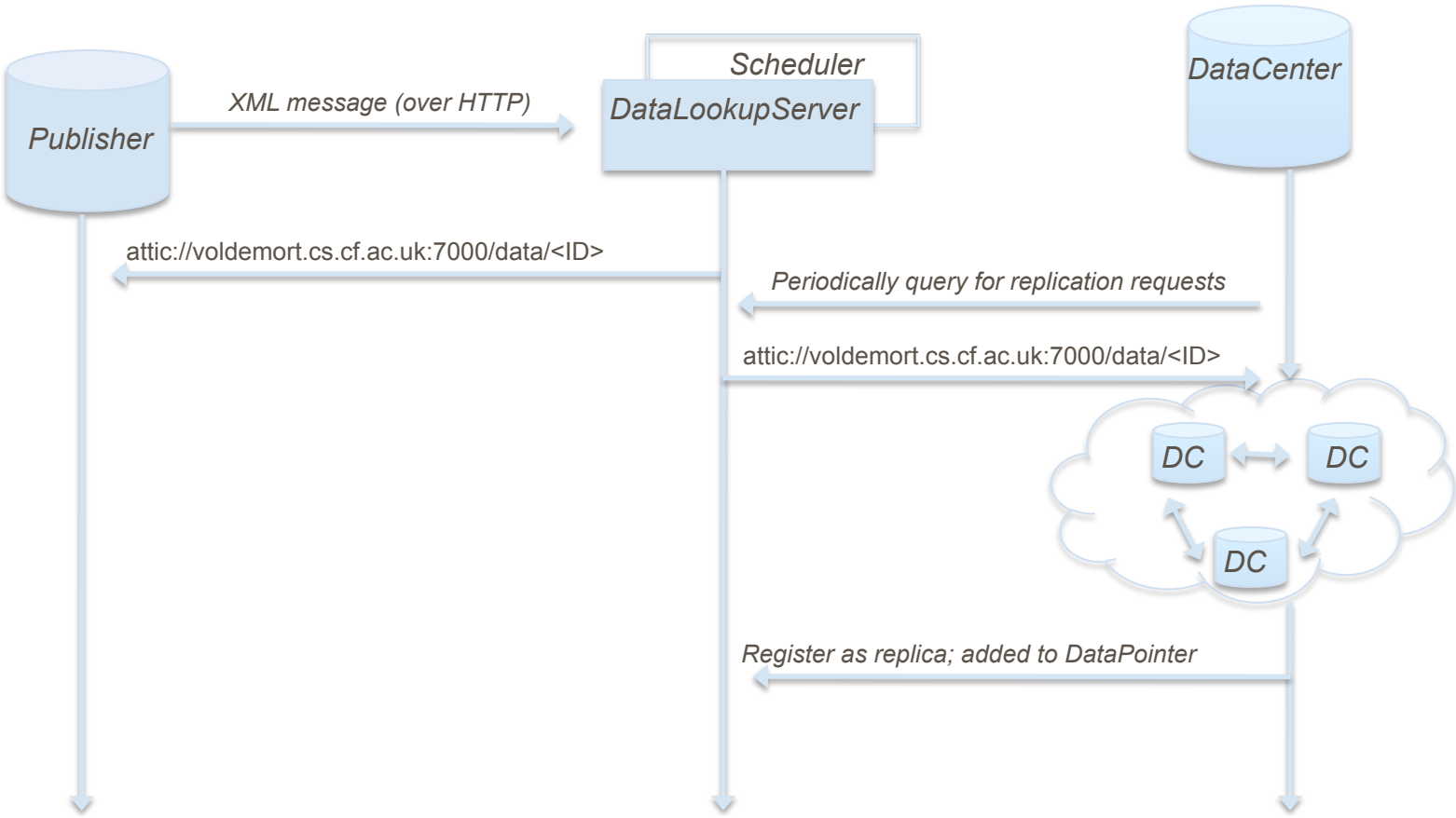
- Rebuilding data from multiple nodes with only partial data
  - before downloading, a metadata request is made to discover chunks at an endpoint
- Endpoint selection based on
  - availability of metadata endpoint
  - RTT of metadata request before download
  - endpoint history
  - duplicate chunks at lower priority endpoints can be used in the event of errors
- Chunk prioritization based on
  - sequentially (used for streaming)
  - endpoint status (fastest first)

Configuration Features

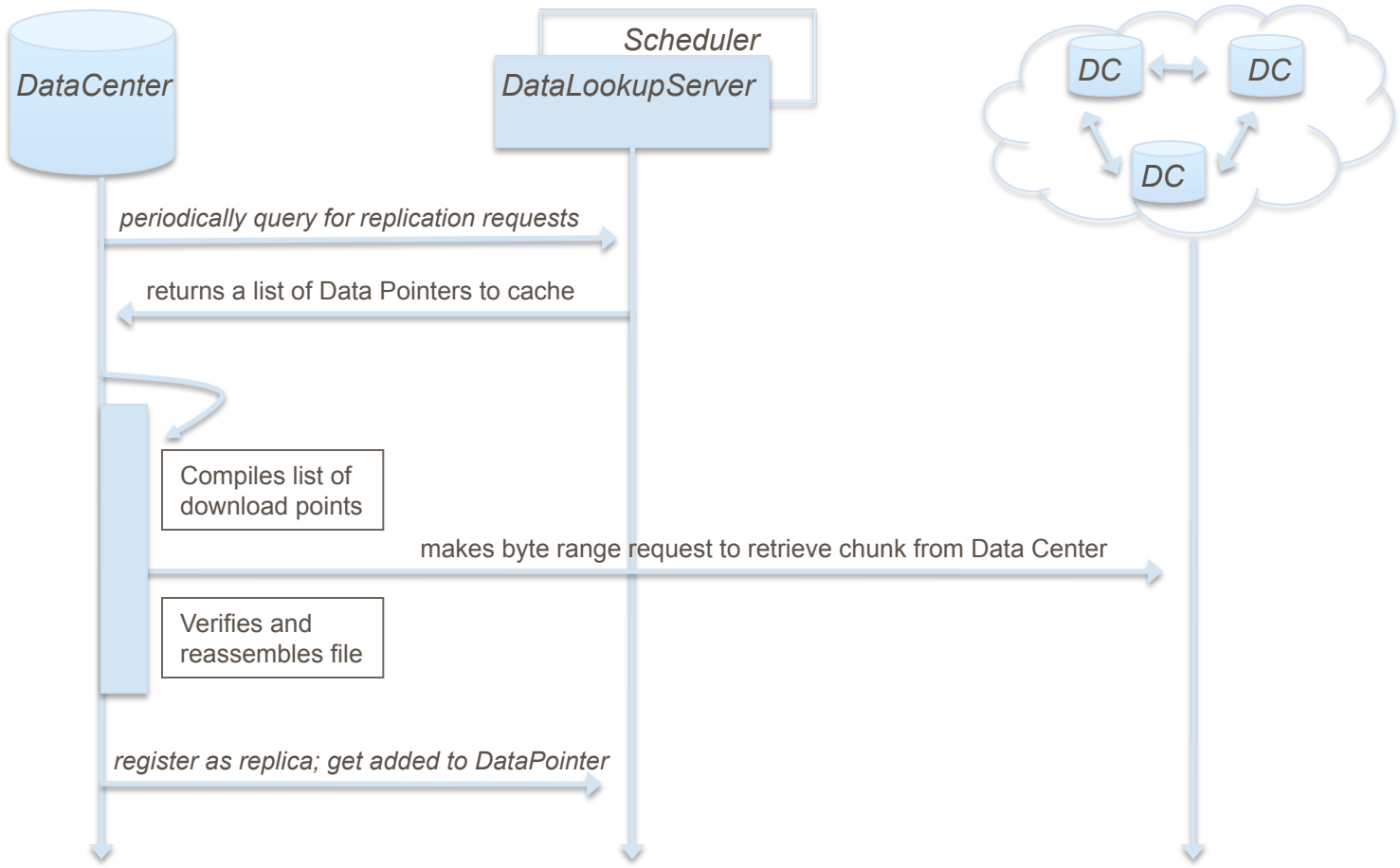
- Web access (TLS mutual authentication)
- Options include:
  - Role(s)
  - Disk space usage and download file type (single file & multiple files that are rebuilt)
  - Connection settings
    - chunk size, number of connections overall/per download, memory footprint, security
  - + *(coming soon)*
    - database configuration
    - up/down bandwidth settings



Attic: Publishing



Attic: Data Center Overlay

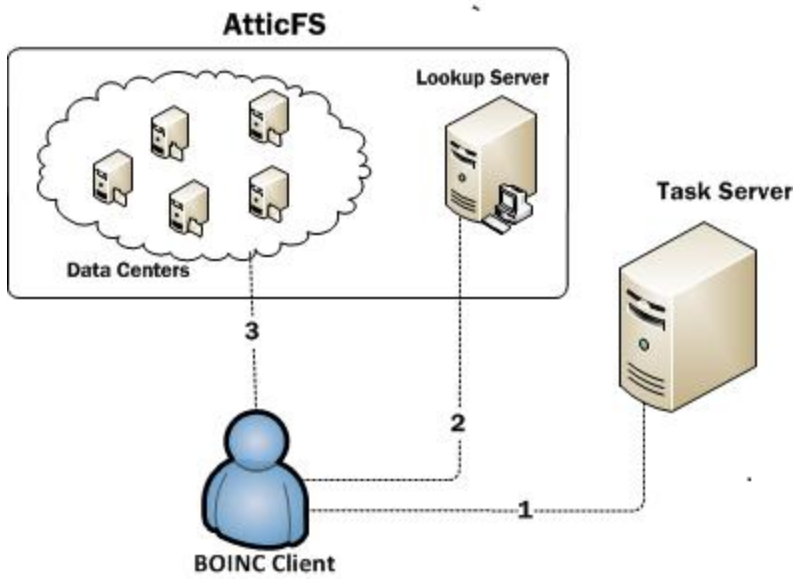


Publish Data to Attic

- Option 1: Use native Java Libraries
- Option 2: Use curl-based CLI w/ Data Seed node
  - Used by the EDGI 3GBridge
    - needs no knowledge of Attic protocol
    - Just a single .sh script to register and send data
    - requires curl 7.x.x on the $PATH
    - main parameters
      - local file to send
      - seed HTTP endpoint
      - certs/keys for mutual authentication
      - others (project, expiry, replica, etc)
    - Outputs Attic URL e.g., `attic://dls.org/1234`

Integrating Attic in BOINC Projects

- Using Attic instead of HTTP in the download URL.
- Reference Data Lookup Server for locating data centers that have input data.
- Generate work units using attic:// URL instead of http://

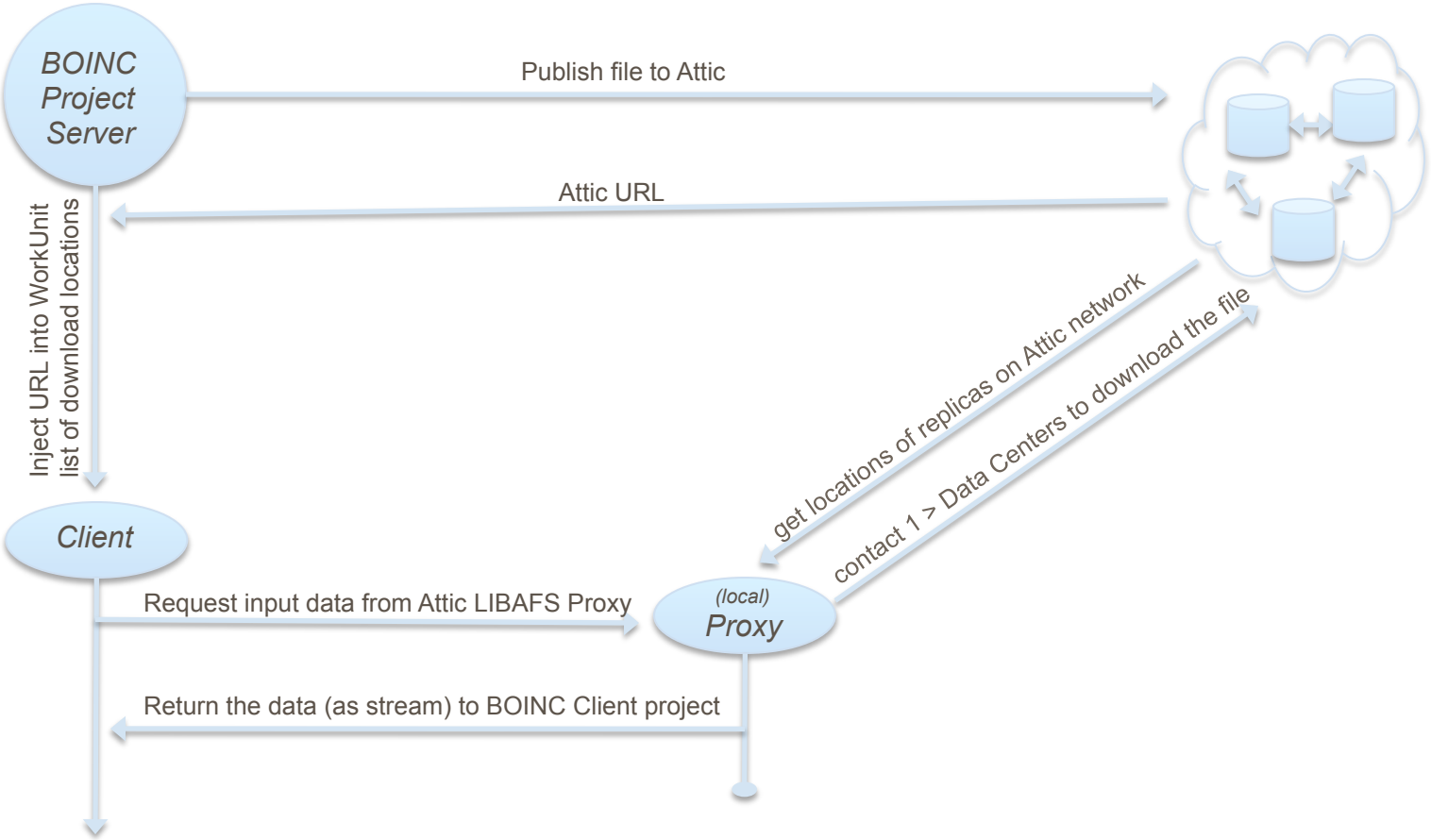


Integrating Attic in BOINC Projects

- Attic "libafs" BOINC proxy client
  - Native-C BOINC project
  - Runs a local web server to intercept URL requests. i.e., `http://localhost:port/<file-identifier>`
  - Requires no additional (project) modification to the BOINC client code, and only minor modification to the server to inject work-unit endpoint and MD5s
    - Except subscription to the new project…
  - Does not "break" anything or endanger BOINC, as there can be automatic fall-over to the next replica URL.
  - Can easily be adapted to intercept attic:// protocol requests (this would require changes to BOINC code)

Integrating Attic in BOINC Projects

http://www.atticfs.org/libafs



Additional Client Integration

- Attic URL Stream Handler
  - Java component that handles URLs with an attic scheme.
  - takes an attic URL e.g., `attic://dls.org/1234`
  - returns a java.io.InputStream for reading the data.
  - Requires only that the application is:
    - written in Java :-) (hopefully we will have a C version by Dec)
    - registers the Attic URL handler.
  - Based on the configuration and data chunks, the stream handler will attempt to verify chunks before passing them to the application

Moving Forward

- Currently deploying Attic within EDGI project as way to distribute Service Grid data
- Each project partner (total: 10) is allocated to deploy a Data Center node
- Files coming from Service Grid Users can be distributed to this Attic layer, giving DG clients a download endpoint.

I'm around tomorrow at the Hackfest to answer any questions, talk about enhancements, future, get Attic working+tested with BOINC, etc.