Catalogue synchronization & ACL propagation

Fabrizio Furano (CERN IT-GT)

The problem

- Various catalogues keep information that is related
  - E.g. LFC keeps info about the content of remote Storage Elements, each one with its own catalogue
    - A change in the permissions of a file in LFC is not automatically reflected by the peripheric catalogue
    - If a SE looses a file, the LFC does not know
    - If a new file is not correctly registered -> dark data
- Keeping them in sync is a very hard problem

The problem

- Current consistency model is not resilient to failures
  - Storage failures lead to dangling entries to be cleaned up manually. Catalogue failures lead to orphaned files.
- Namespace scanning for diffs is an expensive workaround

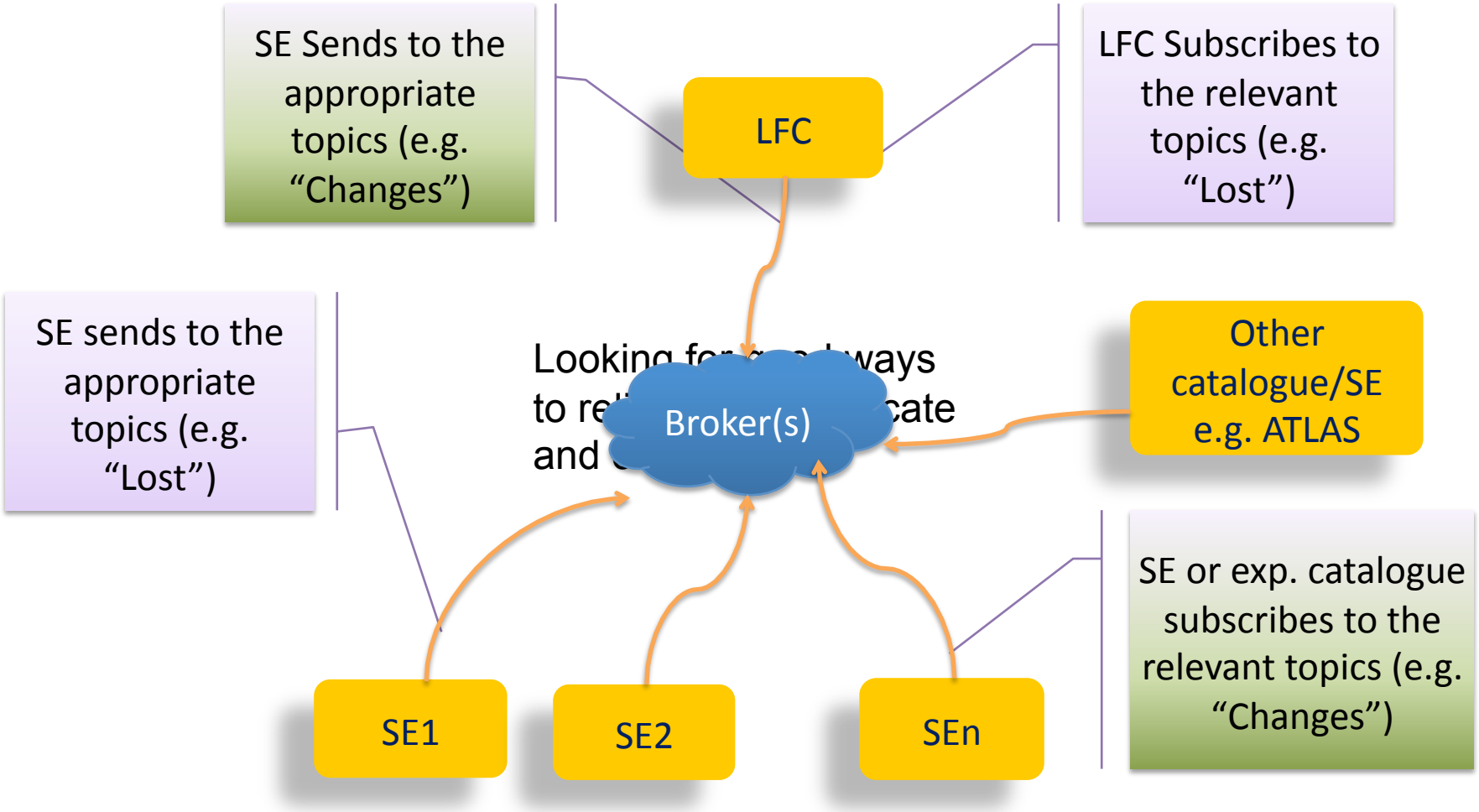
The problem

- Actually, the updates to the various catalogues are performed as external actions
  - By the jobs, as long as they do not die/crash in the wrong moment
  - As scheduled rounds of massive synchronizations

The idea

- Make the various catalogues/SE able to talk to each other
  - In order to exchange messages that keep them synchronized
  - 2 directions:
    - Central Catalogue->SE
      - e.g. to propagate changes in the permissions
    - SE->Central Catalogue
      - e.g. to propagate info about lost files

Communication



Types of interactions

- What can we do with this?
  - Fix inconsistencies as they are found
    - "SE1 apparently lost file X"
  - Prevent inconsistencies by sending messages when something happens
    - "File X has new access permissions"
    - "SE1 has a new file Y"
  - Allow a central system to query the others to synchronize itself
    - "Who has file Z"?
    - "Do you still have file W?"

Messaging APIs and protocols

- Several messaging APIs are available, usable from nearly every language
- Each API talks to the broker(s) using a protocol. Mostly, in our case:
  - OpenWire (binary)
  - STOMP (XML based)
- The broker speaks them all and assures the interoperability
- The idea is not to enforce one protocol, just give usage guidelines so that they all may work
  - Define the message content/semantics
  - Give a reference implementation that can be used by others if they like it
  - Design the things in order to minimize the effort if something changes at the broker side

Message content

- This will be enhanced/agreed in the WG
- The message content/semantics is the building block of the functionalities
- Just to start, we are using a very simple structure
  - Header
    - Proto version
    - Sender hostname
  - Body
    - A set of (tag, value), mandatorily STOMP-compatible, hence just very basic types
    - These are message semantics - related, hence need to use the same set of tags (things like “Filename”)

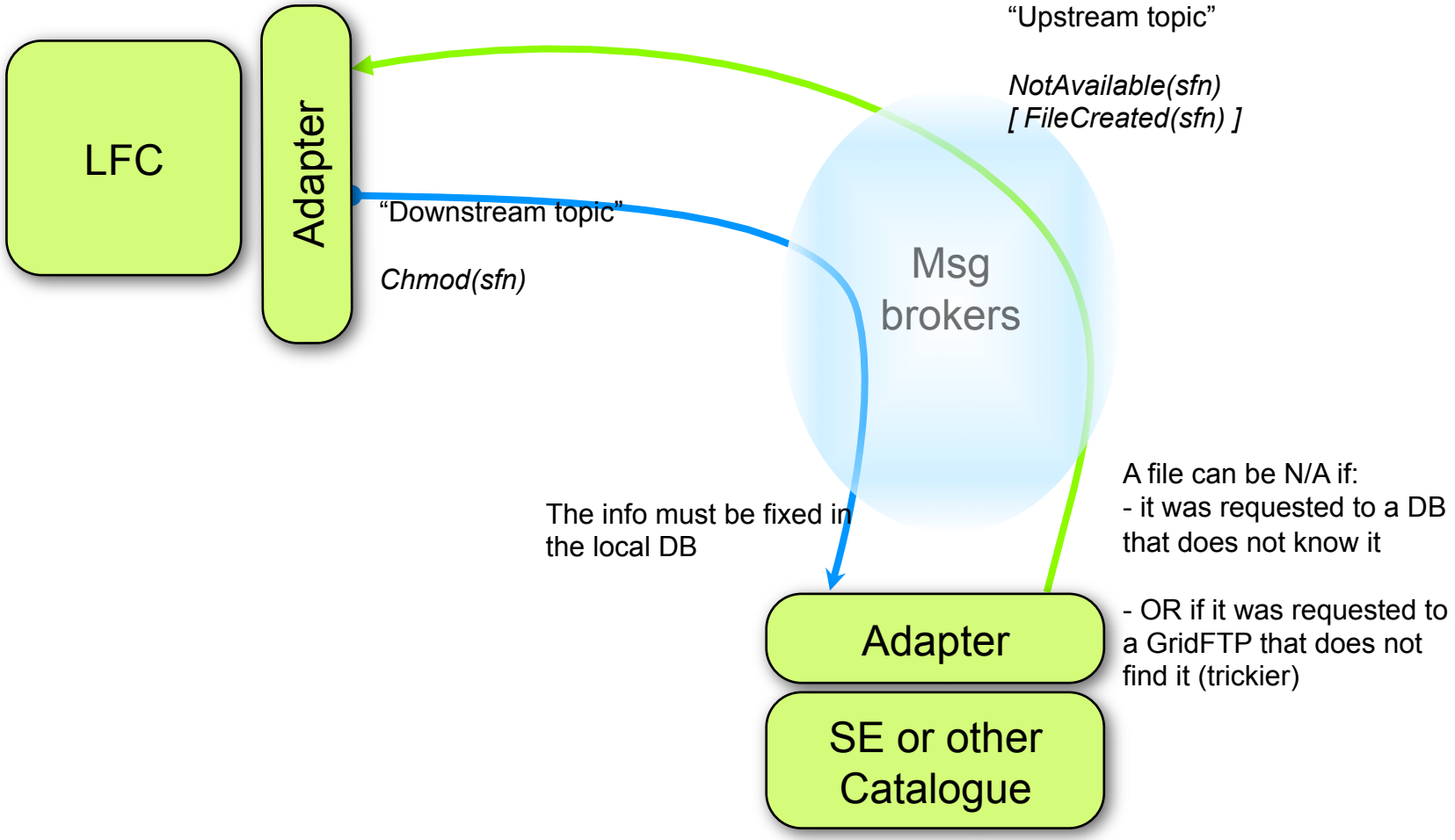
Milestone

- Proposed demonstrators to use reliable message (i.e. industry standard MQ) as backbone of the reliability
  - All interested catalogues can “subscribe” for permissions that changed in the LFC
  - Lost files can be broadcast on the “lost” topic to interested catalogues
    - Note: in general we are talking about synchronizing catalogues
    - Somehow possible also to fix a local catalogue with respect to the content of the local disks
      - Trickier but with an obvious added value

Milestone

- This means
  - Defining the architecture
  - Develop an early LFC/DPM prototype
    - A library that implements the guidelines
    - Usable for LFC/DPM and anybody who's interested
  - Deploy it for evaluation/testing
  - We expect to have a toy prototype by the end of the year

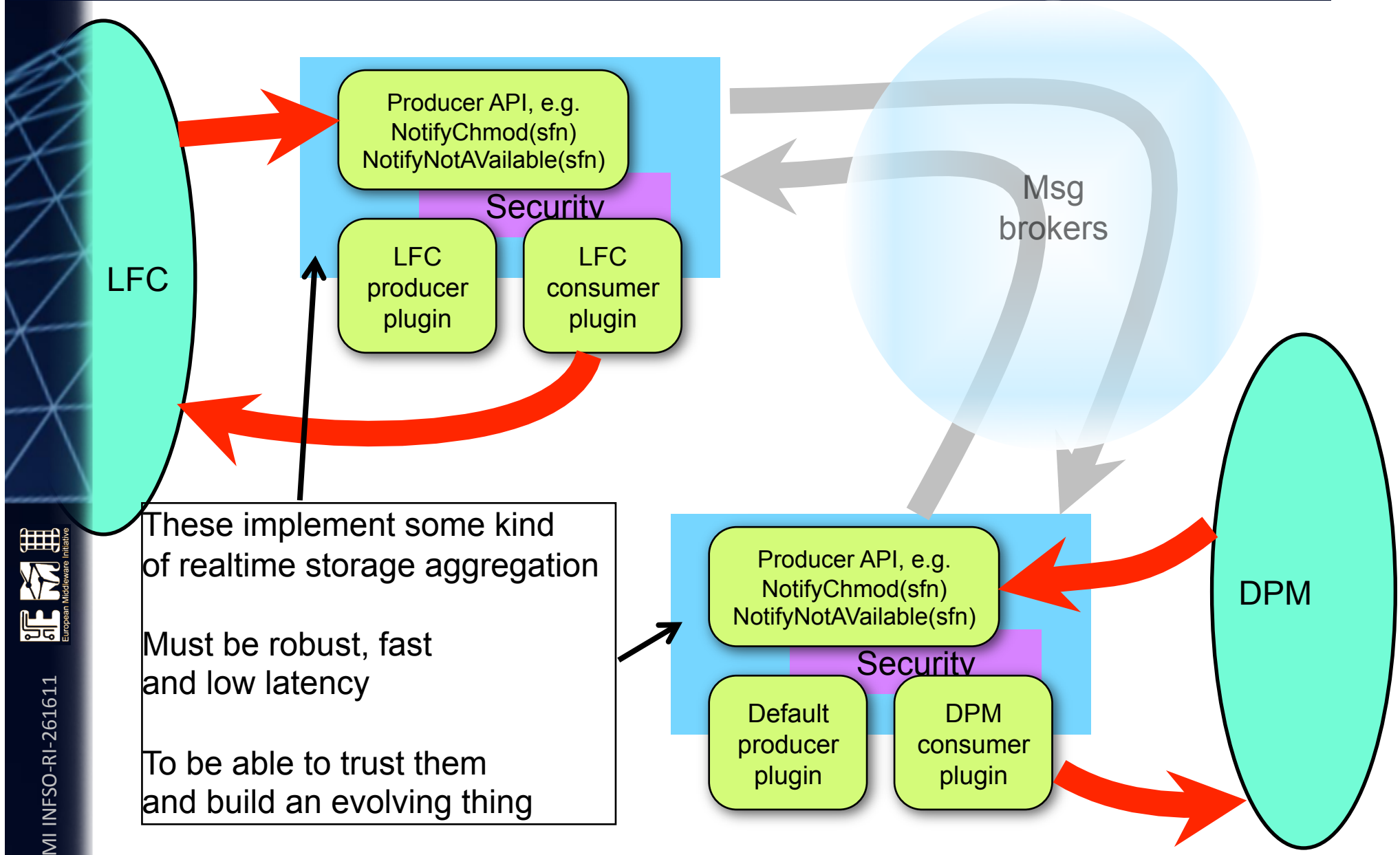
The architecture



LFC/DPM specifics

- The preliminary integration of LFC/DPM uses a new library called SEMsg
  - Built to be efficient and flexible
  - Plugin-based (ev. with “null” plugins)
    - A plugin that performs actions (in the catalogue) when a message comes
    - A simple API to send our messages (hides message composition and the security stuff)
    - A plugin that performs SE(Catalogue)-specific actions when a message has to be sent through the API
  - The same library is used for LFC and DPM, but with different sets of plugins

Detail - SEMsg plugins



Security

- Preliminary requirements:
  - Guarantee the identity of the senders
  - Guarantee the correctness of the content
- It seems that the best way to do this is to sign the messages at app level
  - Like we do with PGP for e-mails
  - This puts an additional difficulty in the dev
    - Not a tragic one if it's done once and for all

APIs and protocols

- Although compatible, each API/Protocol has its own idiosyncrasies. E.g.:
  - STOMP is simply broken in the prod release of the ActiveMQ C++ library (3.2.3). I had to do all the tests with (their) trunk, with good results, at the end.
    - It seems that the fix will be available after the holidays, well compatible with our timescales
  - The reference STOMP implementation that I tested does not support recovery. Any glitch silently makes a zombie client until manual detection and restart.
  - We'll have to deal with the old APR libs shipped with SLC5.
    - Not a big trouble, just install a newer one. Will see the details when the time comes.
  - The app/library must guarantee a full async behavior (must not hiccup on network glitches)
  - Probably many others, we'll see when the tests become more serious.
- This phase is an "advanced investigation", we cannot know everything in advance. We'll also need to gain some operational experience.
- Need to sort out the needed features

Conclusions

- Many aspects have still to be sorted out, but at least we started well.
- Making catalogues and SEs interact seems a good way to attack the consistency problem, but what I like is the idea of realtime interaction between SEs and catalogues.
- The messaging broker tech is not new, at least we know that it works :-)
- The performance seems sufficient, it should also be sufficiently scalable

Thank you

EMI is partially funded by the European Commission under Grant Agreement INFSO-RI-261611