Catalogue synchronization & ACL propagation

Fabrizio Furano (CERN IT-GT)

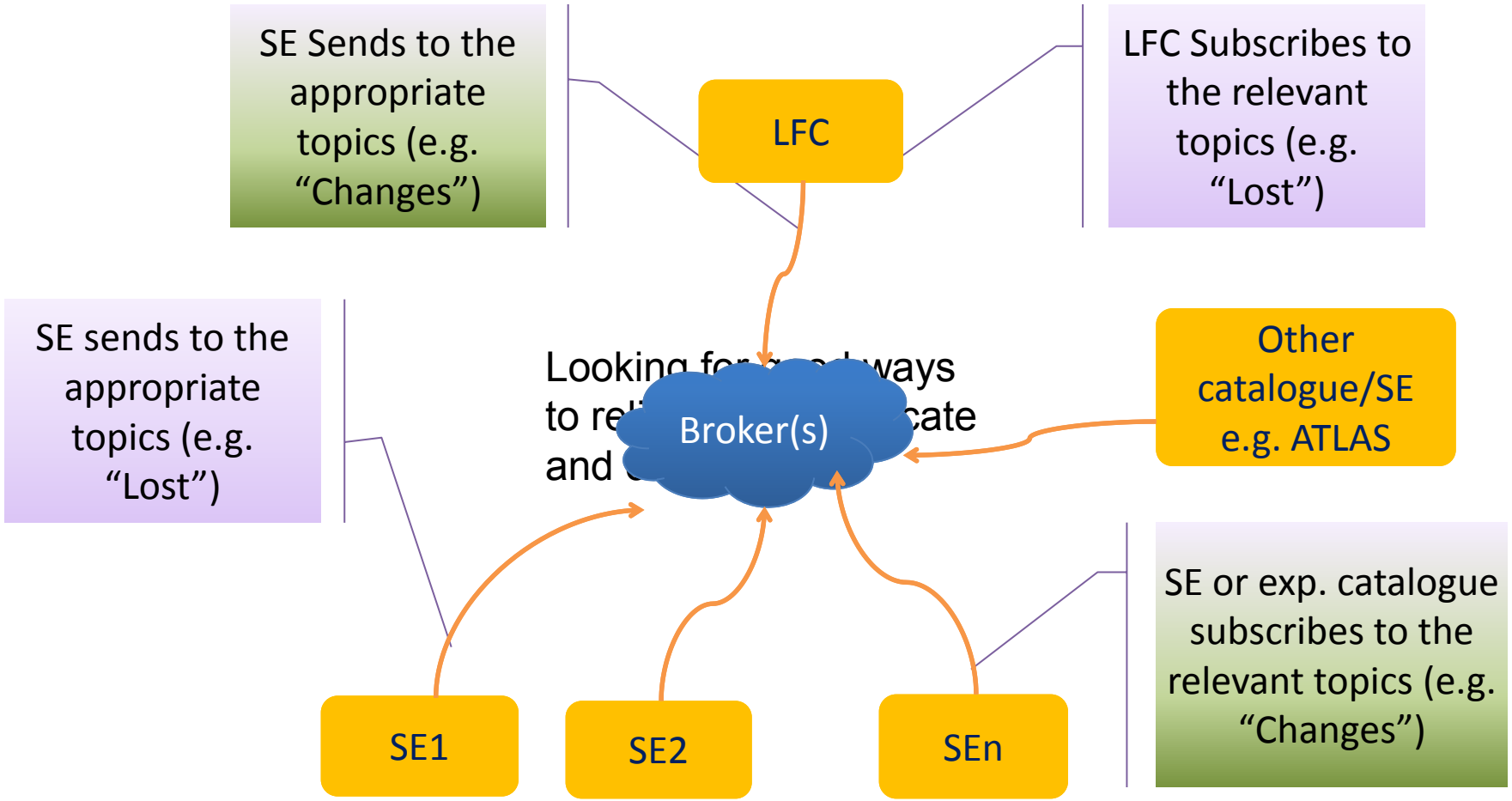
The problem

- 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
- Namespace scanning for diffs is an expensive workaround

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 (downstream)
      - e.g. to propagate changes in the permissions
    - SE->Central Catalogue (upstream)
      - e.g. to propagate info about lost and missing 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?"

Message content

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

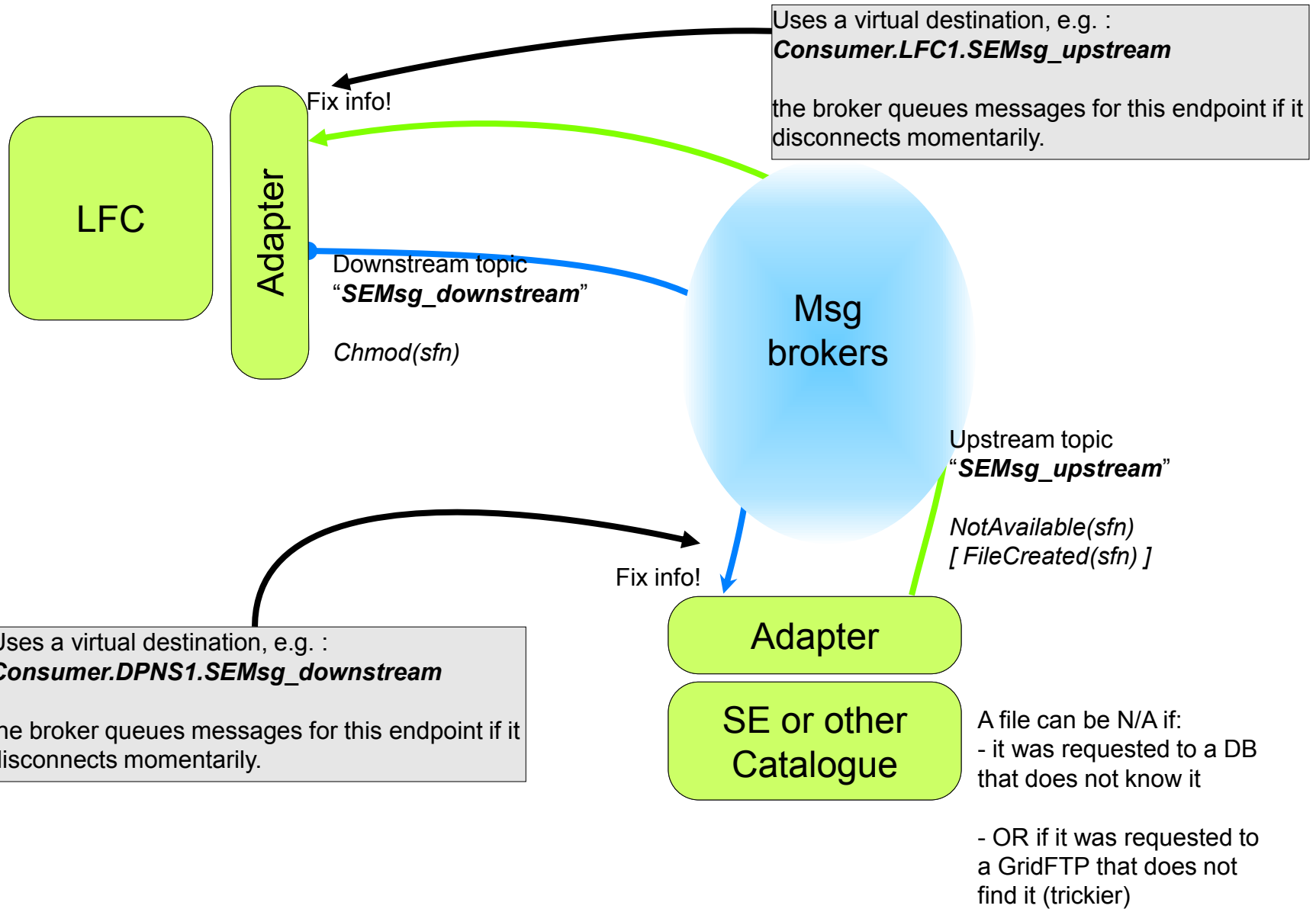
Milestone

- 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 the system
    - Libraries+executables that implements the guidelines
    - Usable for LFC/DPM and anybody else
  - Integrate it into an early LFC/DPM prototype
  - Deploy it for evaluation/testing
  - We have now (Mid Dec 2010) a working prototype

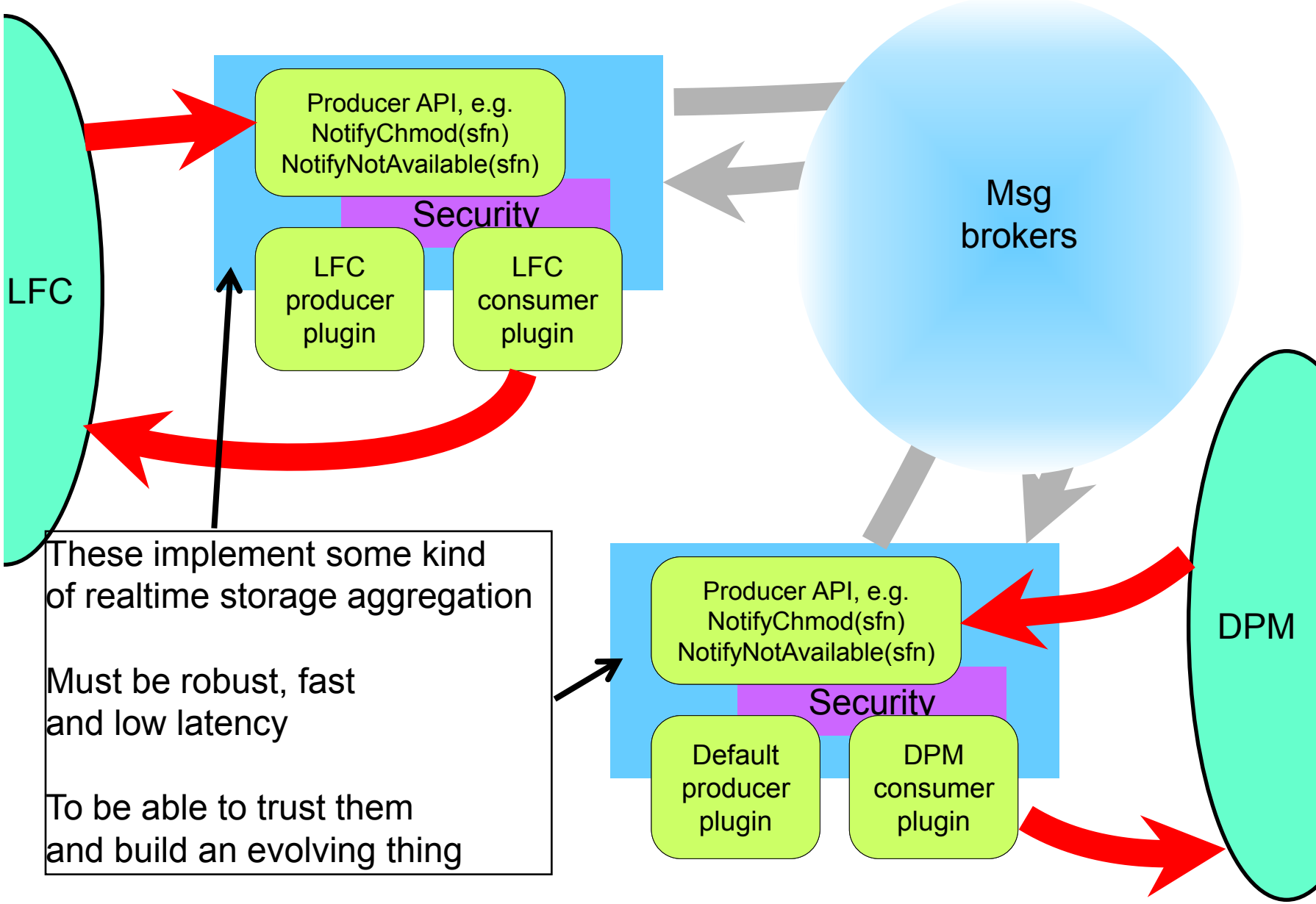
The architecture



The adapter

- The adapter is a component called SEMsg
  - Built to be robust, efficient and easy to integrate
  - Plugin-based (ev. with “null” plugins), loaded at runtime
    - A plugin that performs actions (in the catalogue) when a message comes
    - A plugin that performs SE(Catalogue)-specific actions when a message has to be sent through the API
  - Provides commands to manually send messages
    - As well as a simple C/C++ API to send our messages (hides message composition and the security stuff)
  - The same library is used for the LFC and DPM prototype, but with different sets of plugins
    - Hence, more sets of plugins can be added, to talk to other systems

Detail - SEMsg plugins



SEMsg

- Completely asynchronous, multithreaded design
  - Does everything (in background) to keep the consumer connection UP
  - Never hiccups in case of conn troubles/broker restart
- The consumer may live as a small daemon or be started by another daemon
  - (LFC/DPM use the external daemon by now)
- Also provides commands to send the messages
  - E.g. to manually notify that a set of files is not available
- Natively supports bulk operations
  - Automatic creation of bulk messages from the internal queues.
  - No need for weird APIs or complex implementations.

Security

- Preliminary requirements:
  - Guarantee the identity of the senders
  - Guarantee the correctness of the content
  - Simplicity of use and deployment

- It seems that the best way to do this is to sign the messages at app level (i.e. in SEMsg)
  - 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

What's next

- Write the first version of the protocol spec
  - Message format, conventions etc...
- Prepare for official builds & deployment
  - Still without security by now
  - Implement a more serious configuration subsystem
  - Non-trivial packaging exercise
- Add less trivial notifications (e.g. ACLs)
- Sync with the other SE developers

- Start implementing the security part
- Being able to deploy LFC/DPM+SEMsg for prod test
  - Robustness tests + ev. fixes/additions in SEMsg/DPM/LFC
- At each step, keep an eye on the applicability to the computing models
  - + ev. fixes/additions

Conclusions

- Several 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
  - It's a form of realtime interaction between SEs and catalogues
- The live demonstrator of SEMsg/LFC/DPM is available
  - The messaging (test) infrastructure and the tools seem really OK

Thank you

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