Catalogue synchronisation

Fabrizio Furano
(CERN IT-GT-DMS)
Presenter: O.Keeble

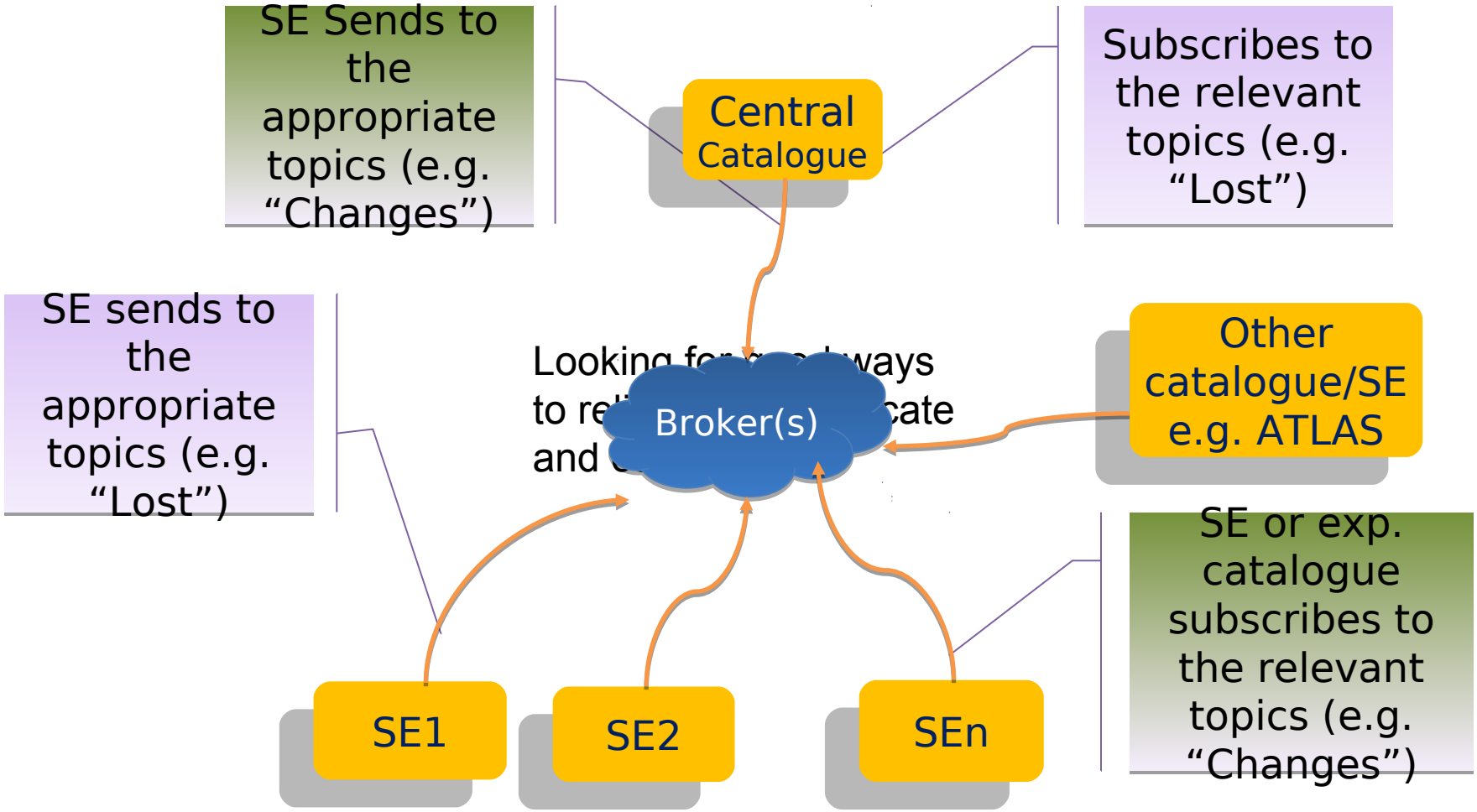
The problem

- 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
    - If a SE looses a file, the LFC does not know
    - If a file is not correctly registered or deleted -> dark data
    - A change in the permissions of a file in LFC is not automatically reflected by the peripheric catalogue
- Keeping them in sync is a very hard problem
- Namespace scanning for diffs is an expensive workaround

The idea

- Make the various catalogues/SEs able to talk to each other
  - In order to exchange messages that keep them synchronised in realtime
  - 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



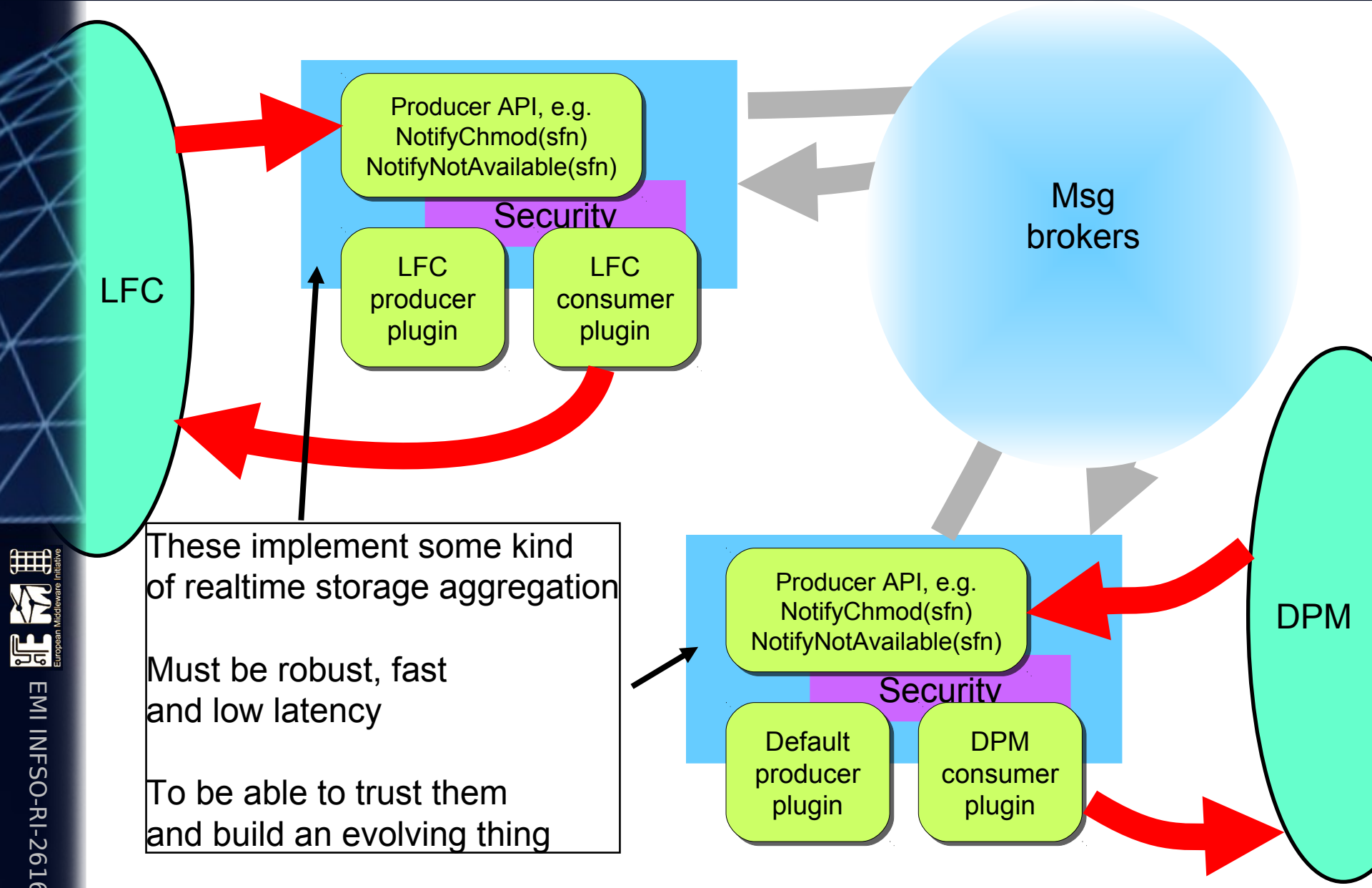
Types of interactions

- 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
    - “SE1 has a (new) file Y”
    - “File X has new access permissions”
  - Interface external systems interested in cross-checking the information against their DBs
    - E.g. an experiment’s database
  - Eventually allow a central system to query the others to synchronise itself
    - “Who has file Z”?
    - “Do you still have file W?”

SEMsg

- Encapsulates the message-related aspects into a clear subsystem
- Simple to use and integrate in other systems
  - Plugin based
  - LFC producer+consumer, DPM producer+consumer, Python consumer
- Completely asynchronous, multithreaded design
- Does everything (in background) to keep the consumer connection UP
  - The API never hiccups in case of conn troubles/broker restart
- Provides CLI to send the notifications
  - E.g. to manually notify that a set of files is not available
- Natively supports the bulk operations defined in the protocol
  - Automatic creation of bulk notifications from the internal queues.
  - No need for weird APIs or complex implementations.

Detail - SEMsg plugins



Security (1/2)

- We can:
  - Guarantee the identity of the senders
  - Make sure that the content has not changed
  - Support SSL connections to the brokers (=comm encryption)
- Everything is X509-based, described in detail in the docs
- Performance is good (hundreds/s), space for improvement, probably not needed now
  - The native bulk messages help a lot here
- The messages are signed using SHA1
  - We treat messages as we do with PGP for e-mails
- Future: implement some form of whitelisting

Security (2/2)

- The communicating entities are in general SEs, hence machines running with:
  - a host certificate
  - a service certificate (possible in principle)
- Now we have a way to be sure about the sender's identity
  - The actions bound to the notifications are never destructive by construction
  - We might like to have a way to filter senders
  - Whitelisting seems a way to go
    - Role for Argus?
  - We need anyway some production experience

The Python plugin

- One more consumer plugin, in the SEMsg distribution, with the DPM and LFC ones
- Associates python funcs to SEMsg notifications
  - Fully configurable in the SEMsg config file and generic
    - e.g. The notification FileNotAvailable invokes the function ‘func1’ from the module ‘module1’ passing its content as individual simple parameters
  - Fast: invokes natively the Python C API, no python spawns are performed
  - Benefits from the SEMsg structure, e.g. the security and the bulk messages
    - The Python script only has to deal with the action to be performed, not with the tech details of the messaging
- Tested with Python 2.4, 2.5, 2.6

Upcoming: ATLAS+LHCb

- ATLAS DDM tools
  - The “DDM problematic file catalogue” can be directly fed with notifications coming from SEMsg
    - Currently, manual invocation of the CLI with lists of files
    - Integration was trivial
- In practice, an instance of the SEMsgdaemon will sniff the notifications and forward them through the Python plugin
  - Step 1: only for files that disappear
  - Step 2: also treat new files that pop up and deletions
- LHCb DM:
  - Missing files
  - Tracking recent registrations and deletes
    - feed dark data realtime crosschecks
- Atlas and LHCb use the same SEMsg tools
  - There’s even potential for a common consumer-side solution

Towards v1.2.0beta

- The Dec demonstrator was OK, so were the tests of 1.1.0 (E.Lanciotti, CERN IT/ES)
- The libs/tools are available in the glite and EMI builds as a pre-release
  - All interested catalogues can use it
  - Documentation available in the TWiki, including the plugins
    - https://twiki.cern.ch/twiki/bin/view/EMI/EmiJra1Syncat
- LFC/DPM were instrumented to use it
- Installed in the DPM nightly testbed
  - Just a matter of installing a couple of RPMs from ETICS and setting a few options
- Rel. 1.2 will contain the enhancements that enable the new features requested by the first 'customers'
- DPM/LFC 1.8.2 will contain these components

What's next

- **Sync with the other SE developers (STORM, dCache)**
- Start deploying the ATLAS+LHCb instances and some real world test instances of LFC/DPM
  - Robustness tests + ev. fixes/additions in SEMsg/DPM/LFC
- Evaluate if the realtime notifications can be used to help Atlas consolidate their LFCs
- At each step, keep an eye on the applicability to the computing models
    - + ev. fixes/additions
- Then, we'll drop the 'beta'

Conclusions

- Making catalogues and SEs interact is a good way to attack the consistency problem
  - It's a form of realtime interaction between SEs and catalogues
  - It won't mathematically kill the inconsistencies, but will help making a much better system
- SEMsg is available as a pre-release until EMI-2
- Protocol and SEMsg documentation in the Wiki
  - https://twiki.cern.ch/twiki/bin/view/EMI/EmiJra1Syncat
- Feedback is welcome
- 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

SEMsg

- Encapsulates the message-related aspects into a clear subsystem
- Simple to use and integrate in other systems
  - Plugin based
  - LFC producer+consumer, DPM producer+consumer, Python consumer
- Completely asynchronous, multithreaded design
  - Does everything (in background) to keep the consumer connection UP
  - The API never hiccups in case of conn troubles/broker restart
- The consumer may live as a daemon or be started within another daemon
  - LFC/DPM use the external daemon
- Also provides commands to send the notifications
  - E.g. to manually notify that a set of files is not available
- Natively supports the bulk operations defined in the protocol
  - Automatic creation of bulk notifications from the internal queues.
  - No need for weird APIs or complex implementations.