Scalable ESGF Node Manager

Sasha Ames / Prashanth D.

LLNL / LIU

March 3 2015

Contents

Background

The current ESGF Node Manager handles:

- Node membership and status
- Capturing metrics
- Sharing node information across federations: certs, endpoints etc
- A mechanism to share common configuration files.

Drawbacks

- Limited scalability.
- P2P file/data exchange could be more secure, particularly configuration files.

Desirable features for next-gen Node Manager

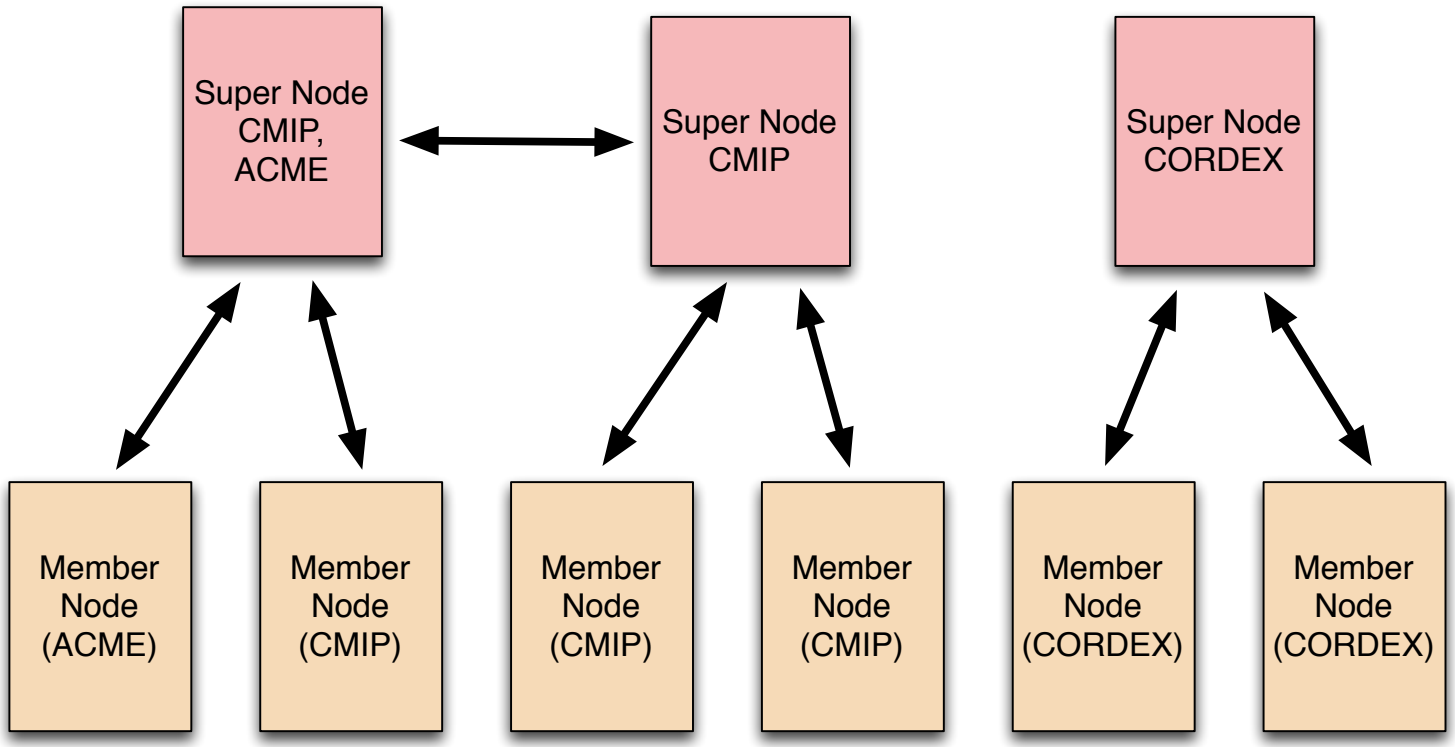
- Fault-tolerant distributed system, without a single point of failure
- High scalability without overloading resources
- Minimise communication overheads
- PAN federation administration: handling cert requests, node memberships, etc.
- Consistent and highly available common configuration files
- Mechanism to enable component redundancy via consistency

Node Manager types

Node managers can be of three different types.

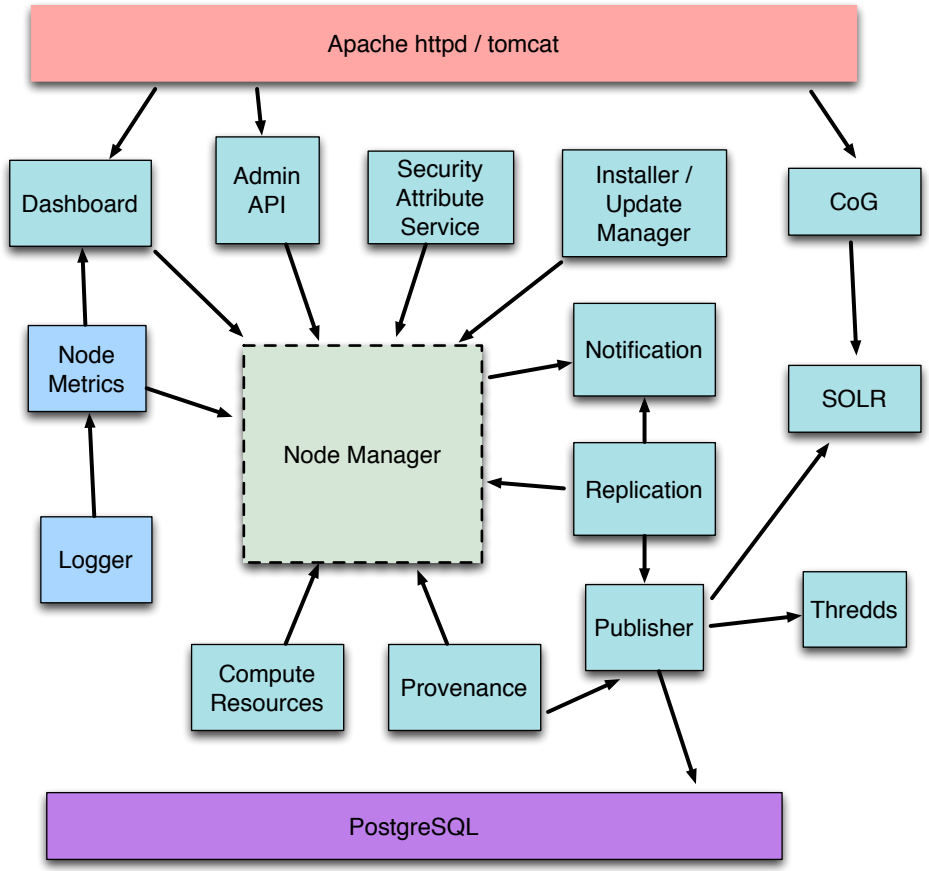
1. Supernodes
   - A validated and reliable source for configuration directives, metrics, information about components, etc., at project level.
   - Multiple concurrent supernodes for scalability, fault tolerance and load sharing.
   - Supernodes query other Node Managers for metrics and status.
   - A single Node Manager can serve as supernode to multiple projects or even as supernode to one and membernode to another etc.
   - Administrator action required to add/remove from ESGF
2. Membernodes
   - Default Node Manager configuration
   - Cannot query other Node Managers.
   - May join or leave ESGF "at will"
3. Standby supernodes
   - The Node Managers run as supernodes only when too few supernodes are operational.

View of federated Node Managers with projects



Node Manager role in ESGF Node

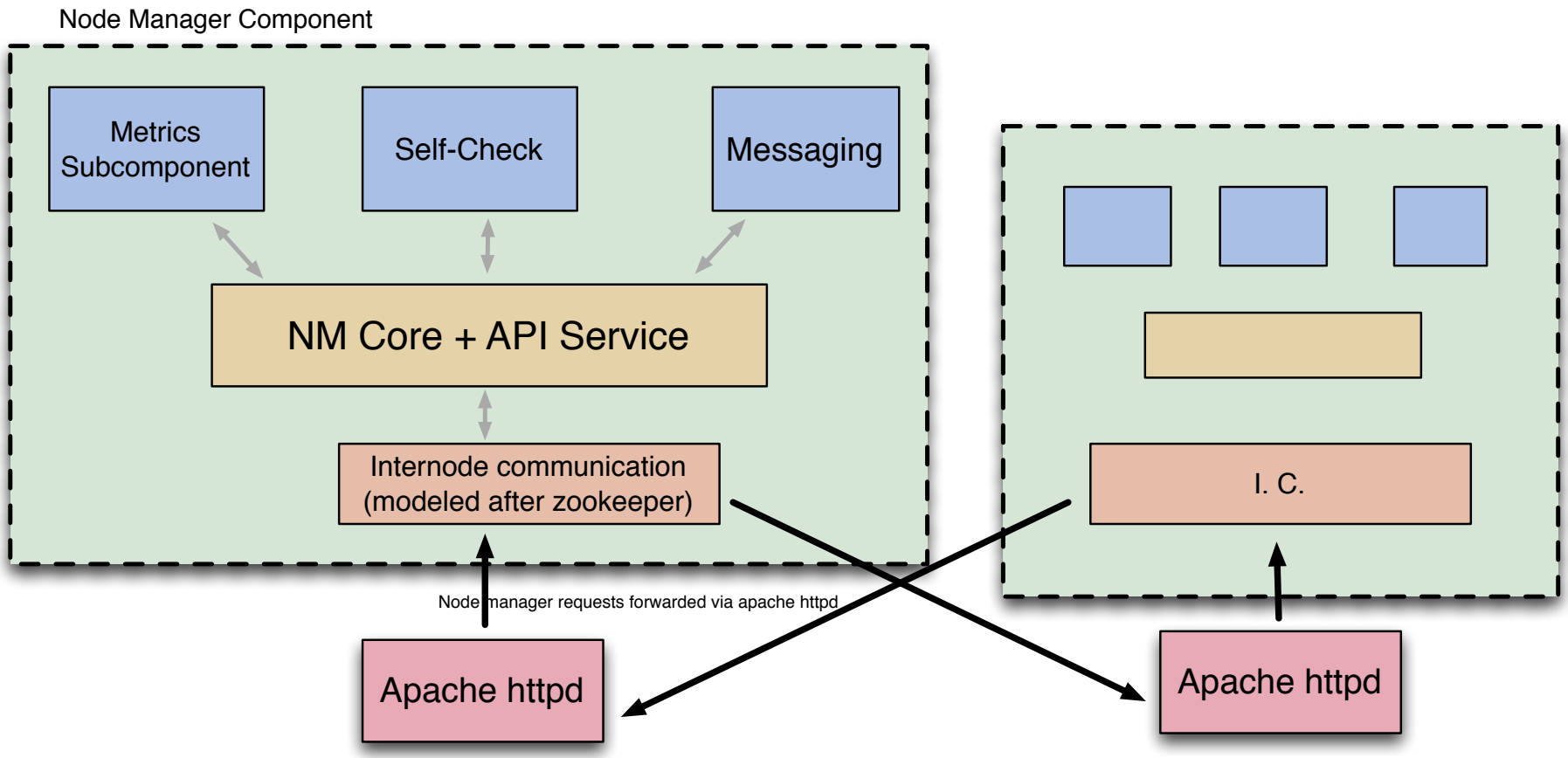
Dashboard	Manage consistent updates to registration.xml
Metrics	Coordinate metric metadata
Compute	Maintain repository of federated compute resources
Security	Replication of attributes for service failover
Update Manager	Consistent record of component versions
Notification	Push alert messages out to administrators / users
Replication	Data set version information
Publication	Consistent project-based configuration
Provenance	Consistent provenance metadata
Thredds	???
SOLR	???
CoG	???
Logger	(metadata)?



Node Manager components

- **Communication management**
  - influenced by Zookeeper
  - Supernodes health check each other and member nodes
  - handles node failue
- **ESGF Node Manager API** two subtypes
  - P2P api - part of communication management
  - Services API - repository for ESGF components to share information, look up status
- **Metric collector**: query member nodes and aggregate them (supernodes)
- **Self-check component**: run sanity checks on self.
- **Admin console (local node)**: submit membership requests, CSRs, volunteer for supernode role etc
- **Admin console (supernode admin mode)**: sign CSRs, manage membership and volunteering requests etc.

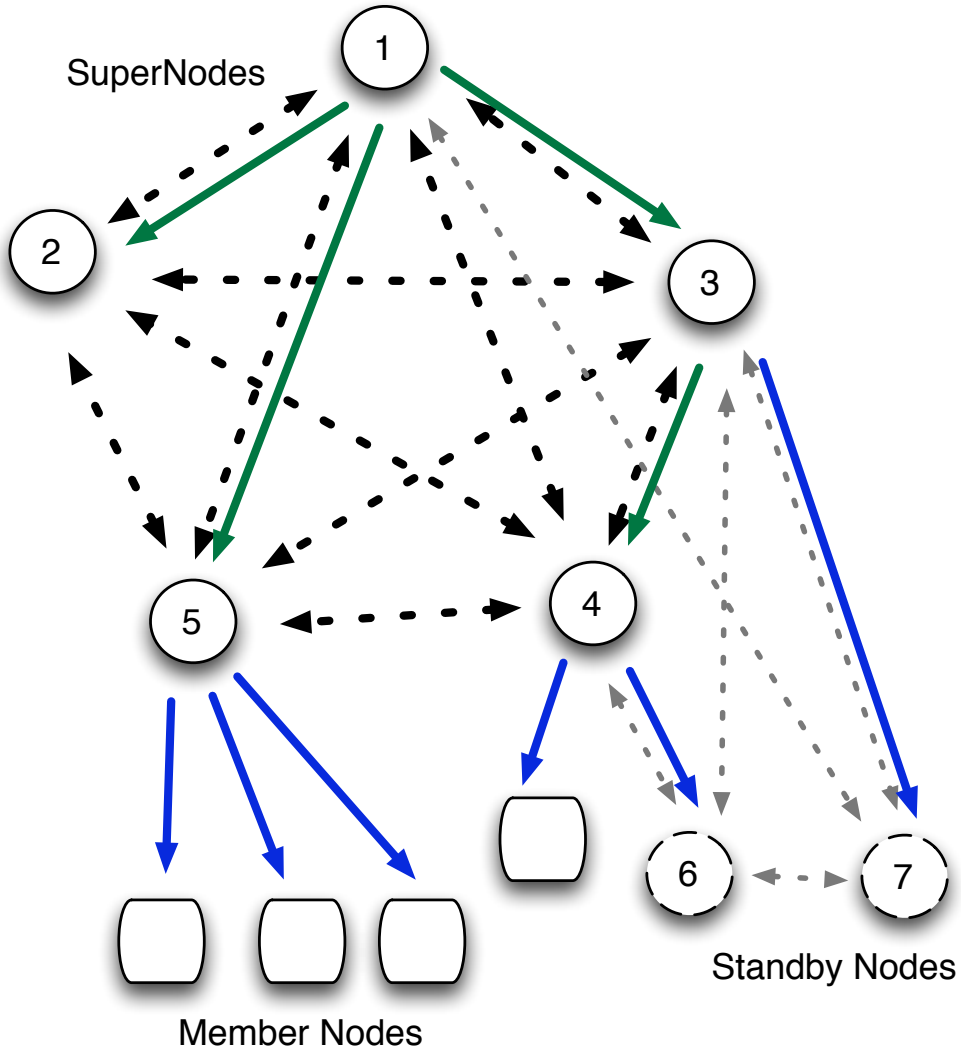
Node Manager component diagram



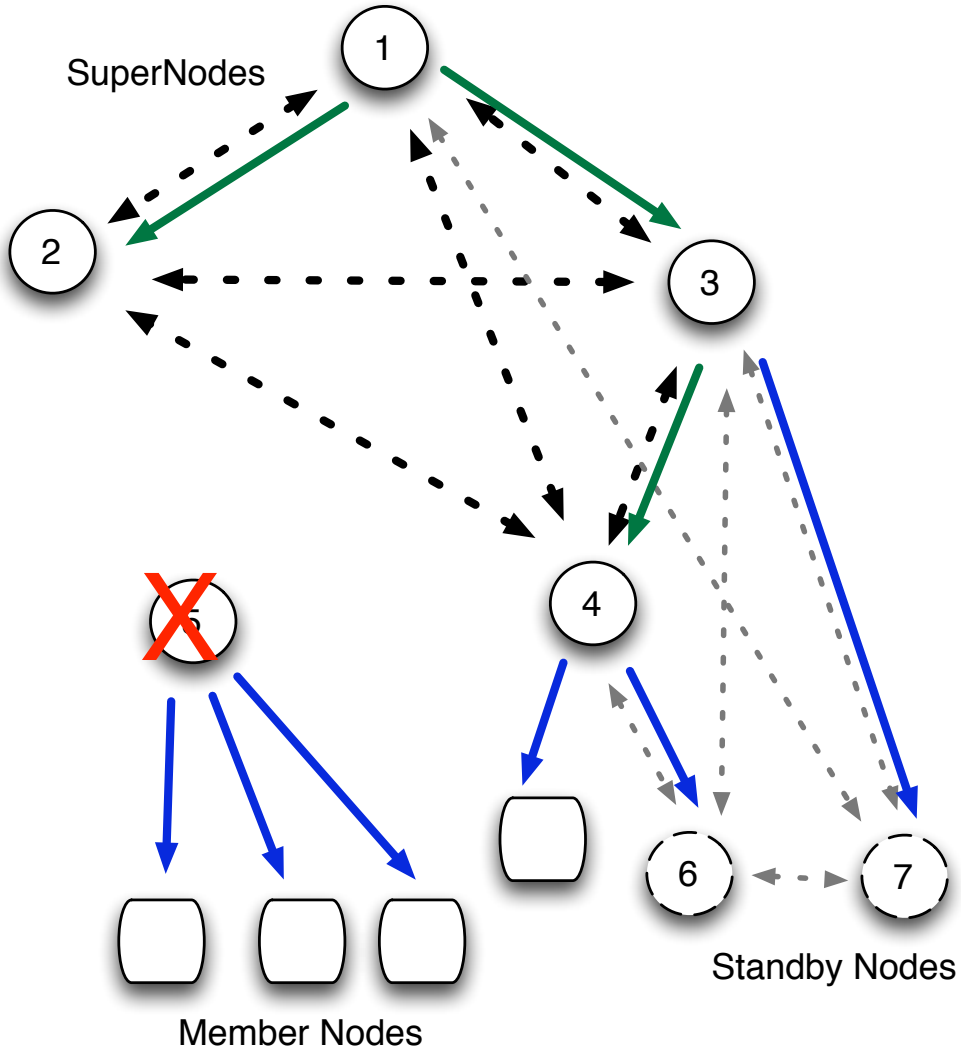
Supernode failure mitigation

- Member nodes need to be redistributed for node-checks, file distribution in event the assigned supernode fails
- Option 1. Distribute to active super nodes
- Option 2. Promote a standby node. Distribute remaining nodes to "new" supernode, place of promoted node.
- Choice of options depends on setting, resources.

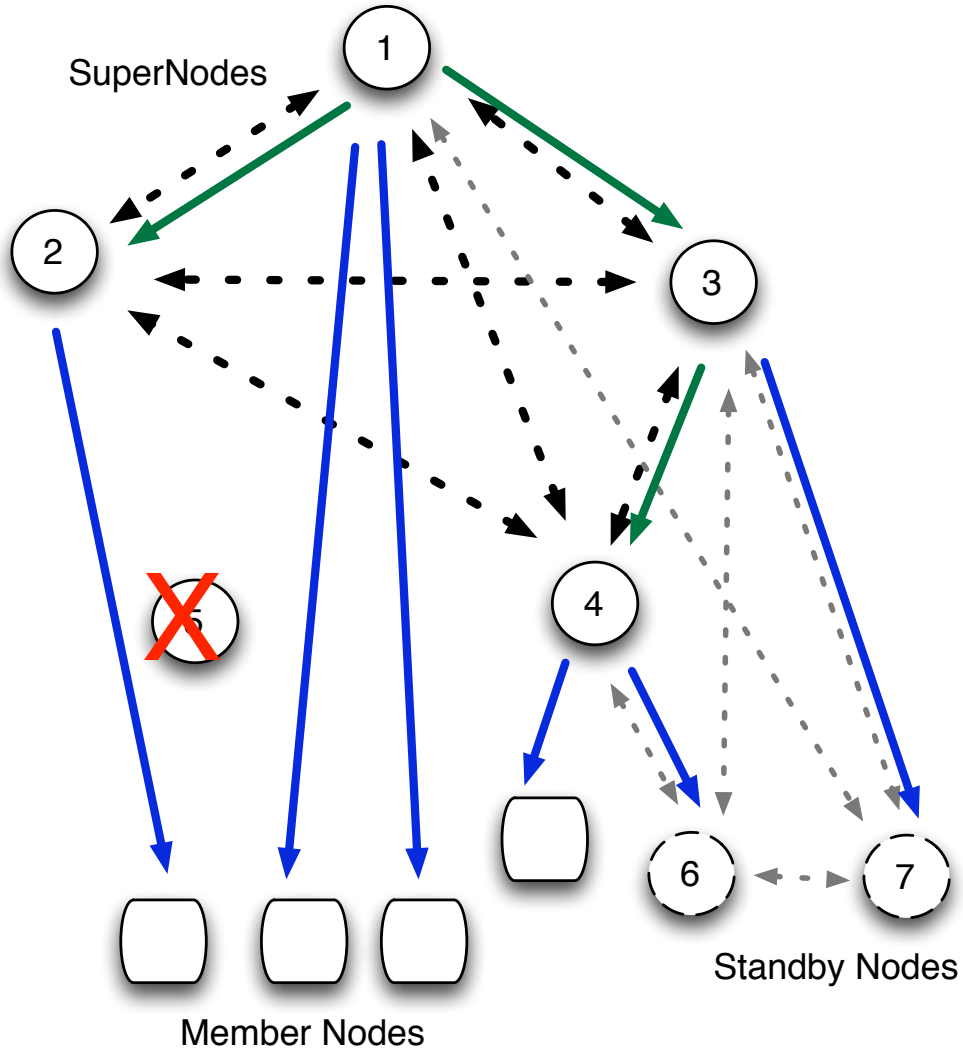
Supernode failure mitigation



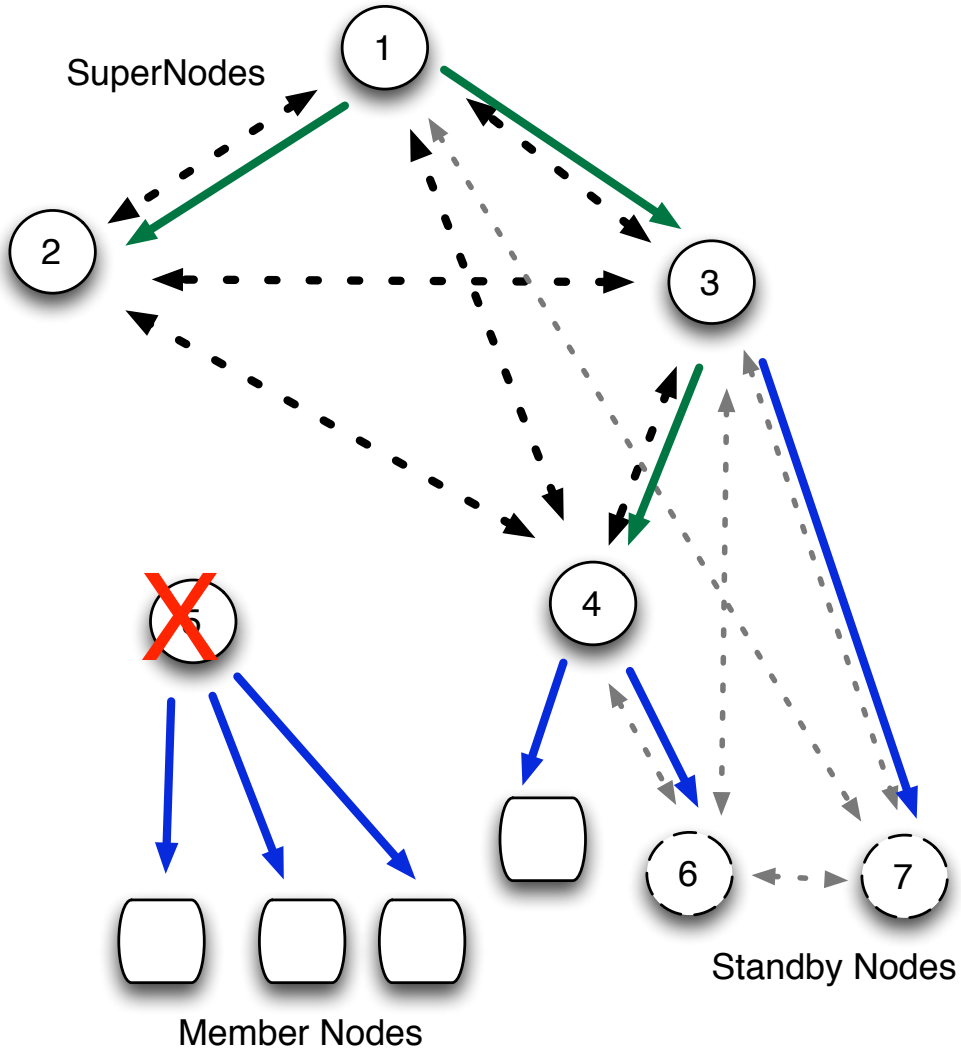
Supernode failure mitigation



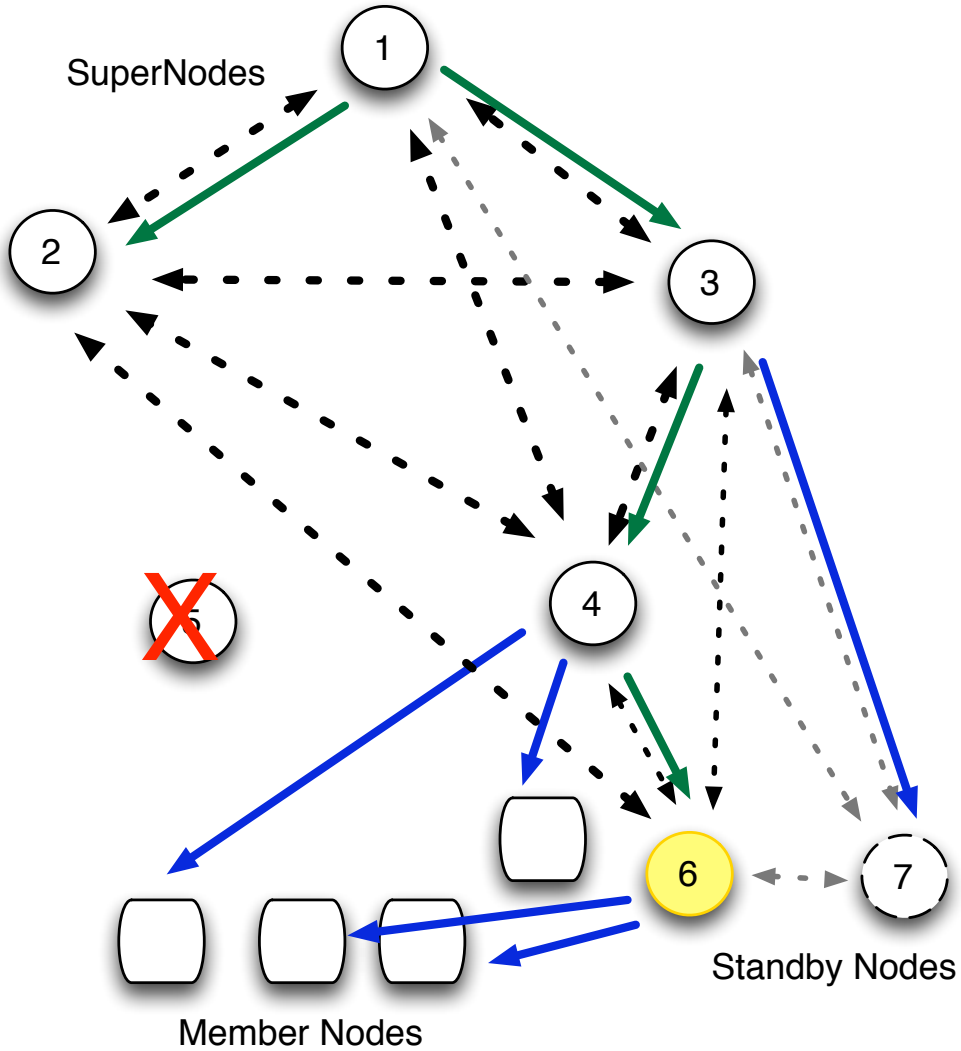
Supernode failure mitigation - Option 1



Supernode failure mitigation - Option 2



Supernode failure mitigation - Option 2



Conflict resolution

- Modeled on git principles
- Changes are like commits with timestamps (supernode id becomes the "tiebreaker")
- Deterministically ordered and replayed - all nodes get the same answer.
- Example: "race condition" of multiple member nodes joining in close succession.

Prototype of the Node Manager

- **API** Django
  - Supports node map distribution, member node join / self-removal.
  - Passes work items (changes) to task queue
- Task queue - process changes. Single worker handles updates synchronously
- Communication set up asynchronously. Responses added to queue.

Node Manager design considerations

- Security: factor for both user/machine executed elevated privilege operations.
- Design to guard against spoofing of membernodes/supernodes etc.
- Which communications need to be secure? For instance, health checks might not need encryption but detailed status might.
- **bootstrap:** how to stand up initial set of supernodes
- **transition roadmap** how to cut over from existing node manager without creating massive downtime for ESGF

Implementation Status

- Federation prototyping ongoing. single source health check complete
- Prototype implementation items to be completed
- node reassignment
- "multiple source" health check
- conflict resolution
- dashboard support
- Integrate django-based API with other django services running on ESGF node / httpd
- secure (https) communications
- Summer - start testing in wide-area environment - will need your help!