Replicable Services:

Running dCache in a High Availability Configuration

11th International dCache Workshop

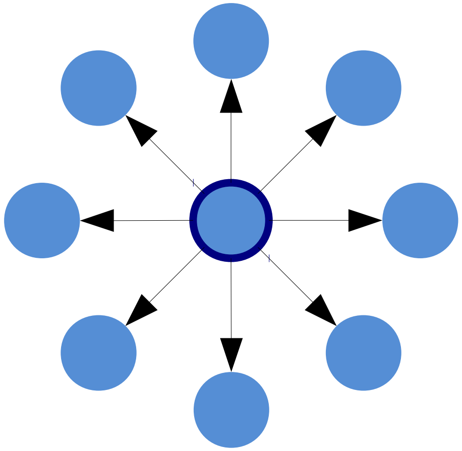
Jürgen Starek
on behalf of the project team

Overview

- What changed?
  A review of the new features since 2.16
- How to profit from it?
  An update demo resulting in an HA setup

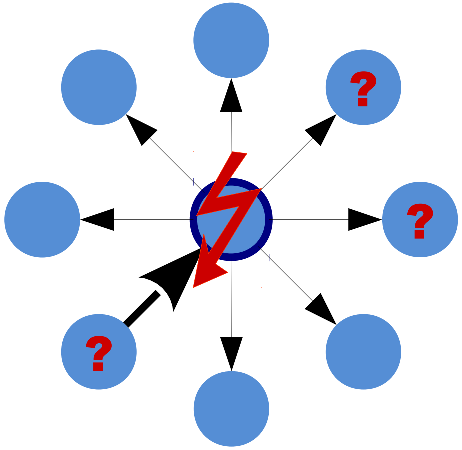
A look at the status quo

- Hierarchy:
  - Domains (containers for cells, each with own VM)
    - Cells (doors, pools, …)
- Cells communicate through messages
- Expectation: “dCacheDomain” as center of communication



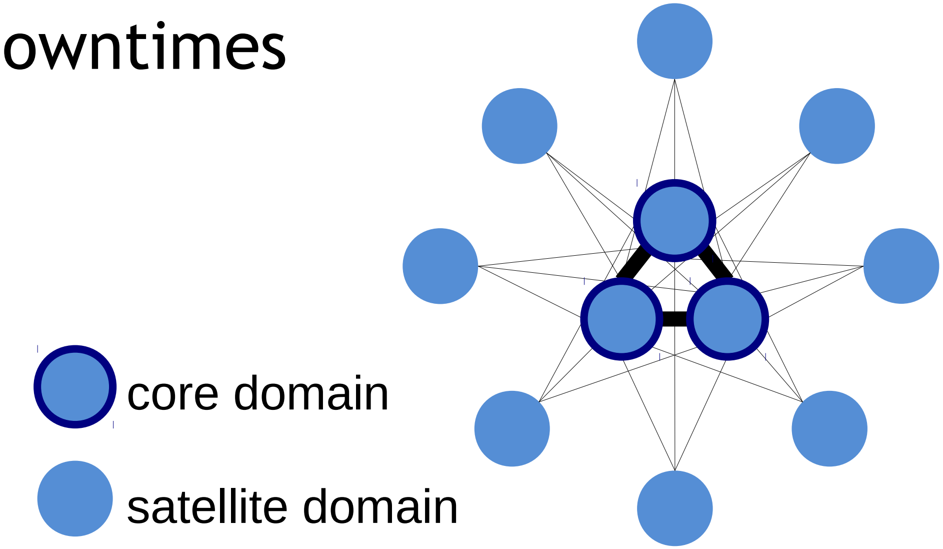
Topology issues

- Single point(s) of failure
- Network partitioning can cause system failure even while all components are up
- Performance may hinge on single component



New since 2.16: Replicable Services

- More than one instance of any crucial component
- No single points of failure
  - Overall system integrity preserved in the face of network or server issues
  - Individual transfers may be aborted
- Rolling updates without downtimes
- Scalability
  - HA-aware doors and SRM
  - HA proxy enabled



Definitions

- Load Balancing
  - analyzing load on nodes and distributing work so that the load is spread evenly (think poise)
- Load Distribution
  - assigning load to nodes without knowing about their status, relying on statistical avaraging
- High Availability
  - Availability of overall system functionality in the face of technical problems, without regard for performance

Replicable Services

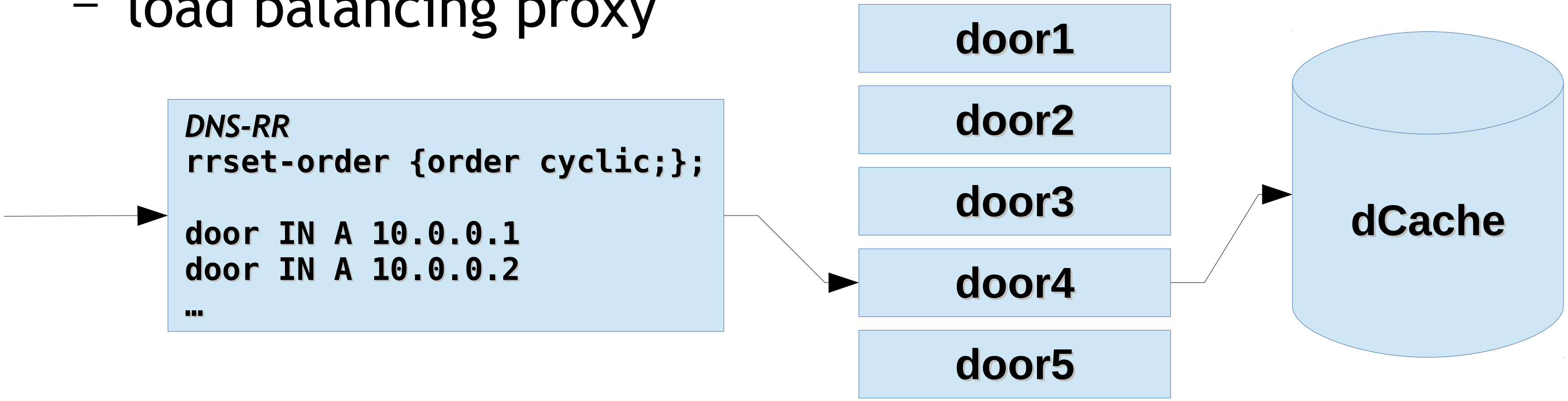
- Differentiate between service name and instance name(s)
  - `PnfsManager`: Service name (logical level)
  - `PnfsManager@somedomain`: cell instance (physical level)
- A replicable service supports
  - this separation
  - having multiple instances

List of Replicable Services

- Critical Services
  - Spacemanager
  - Pinmanager
  - SRM Manager
  - PNFS Manager
  - gPlazma
- Other Services
  - admin
  - httpd
  - info
  - topo
  - statistics

Towards replacing instances

- Established load balancing mechanisms between several doors already allows rolling updates
  - srm
  - DNS round robin
  - BDII
  - load balancing proxy



DNS-RR

```
rrset-order {order cyclic;};

door IN A 10.0.0.1
door IN A 10.0.0.2
…
```

Towards replacing instances

- But what about central services like Pin Manager or PNFS Manager?
  - there‘s only one „true“ status
  - avoid inconsistencies of distributed systems

Challenges

- System needs
  - Common consensus about system status
  - Topology discovery
- Architecture
  - Avoid non-replicated components also beyond core dCache

Zookeeper

- Central component
- Distributed key-value-store
- Source of Truth and Consensus
- Ideally deployed as standalone cluster
  - alternatively: built-in

Zookeeper and HA

- *CAP* theorem: Choose two of [*C*onsistency, *A*vailability, *P*artition resistance]
  - Zookeeper as a „source of truth“ system implicitly chooses *C*
  - Between the remaining *A* and *P*, it chooses *P*, so we need to tolerate short outages!
- Bottom line: expect short outages
  - “Zookeeper is down” is actually a feature when the network is flaky! It is not *A* until *C* can be ensured again.

CAP theorem / Brewer‘s conjecture: See Gilbert/Lynch: „Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.” ACM SIGACT News, v. 33 issue 2, 2002, p. 51–59. DOI 10.1.1.20.1495

Zookeeper as a topology information service

- Replaced location manager
- Informs other dCache services which service instances are up
  - instances auto-register at Zookeeper
  - Zookeeper connection info must be configured manually on dCache nodes
- keeps information consistent throughout cluster

PostgreSQL

- HA configuration beyond dCache scope
  - not strictly necessary if only rolling updates are desired
- General PostgreSQL concept:
  - primary server with active DB
  - standby servers with copies of primary, read-only active, prepared for failover

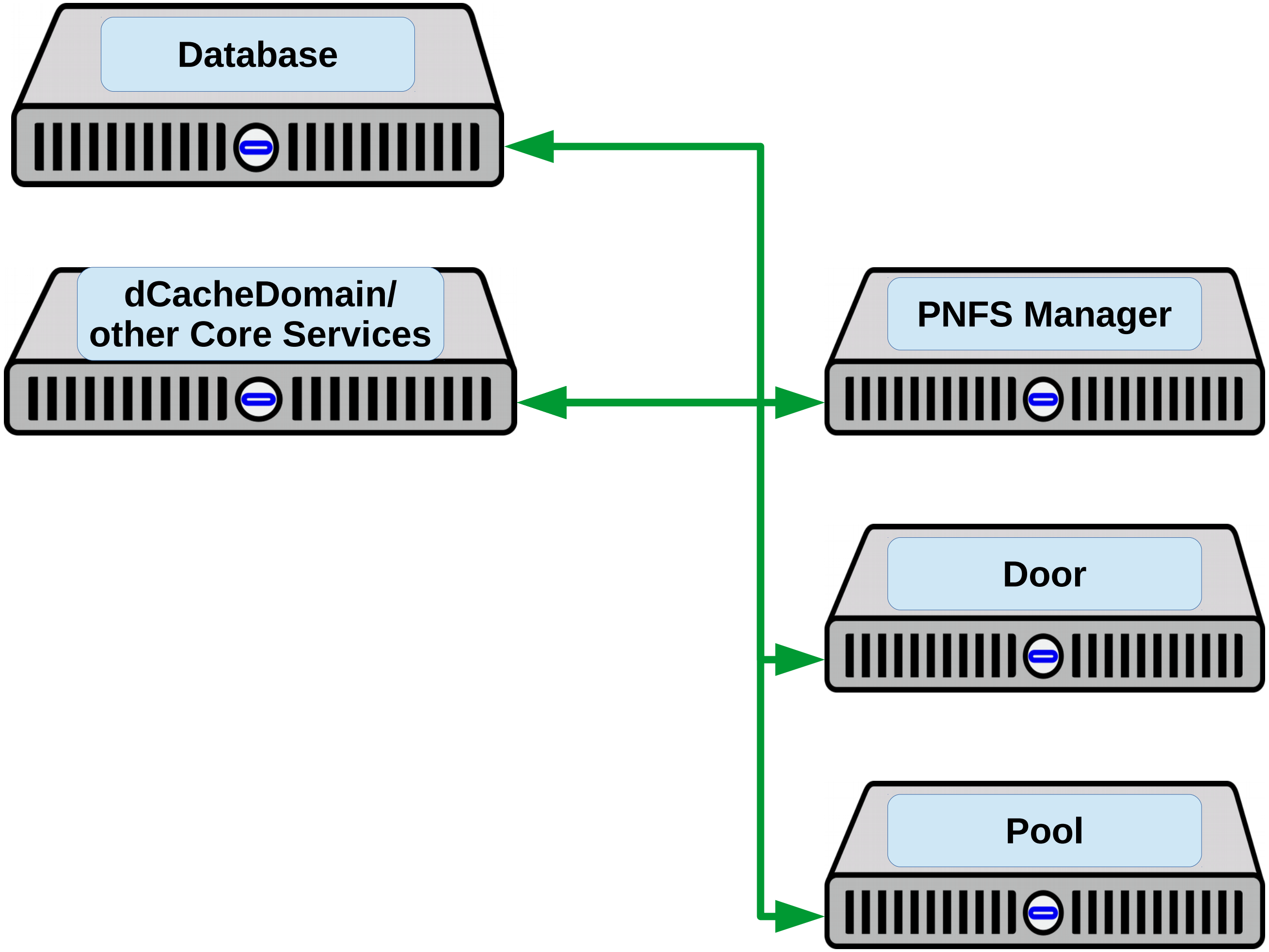
Not yet awesome

- Transfers can’t be interrupted and resumed mid-flight
  - movers only started on request by clients
  - doors can‘t be restarted without clients reconnecting
- We still rely on clients to retry after a reasonably long timeout
- Writing into breaking connection: Client must react to I/O errors

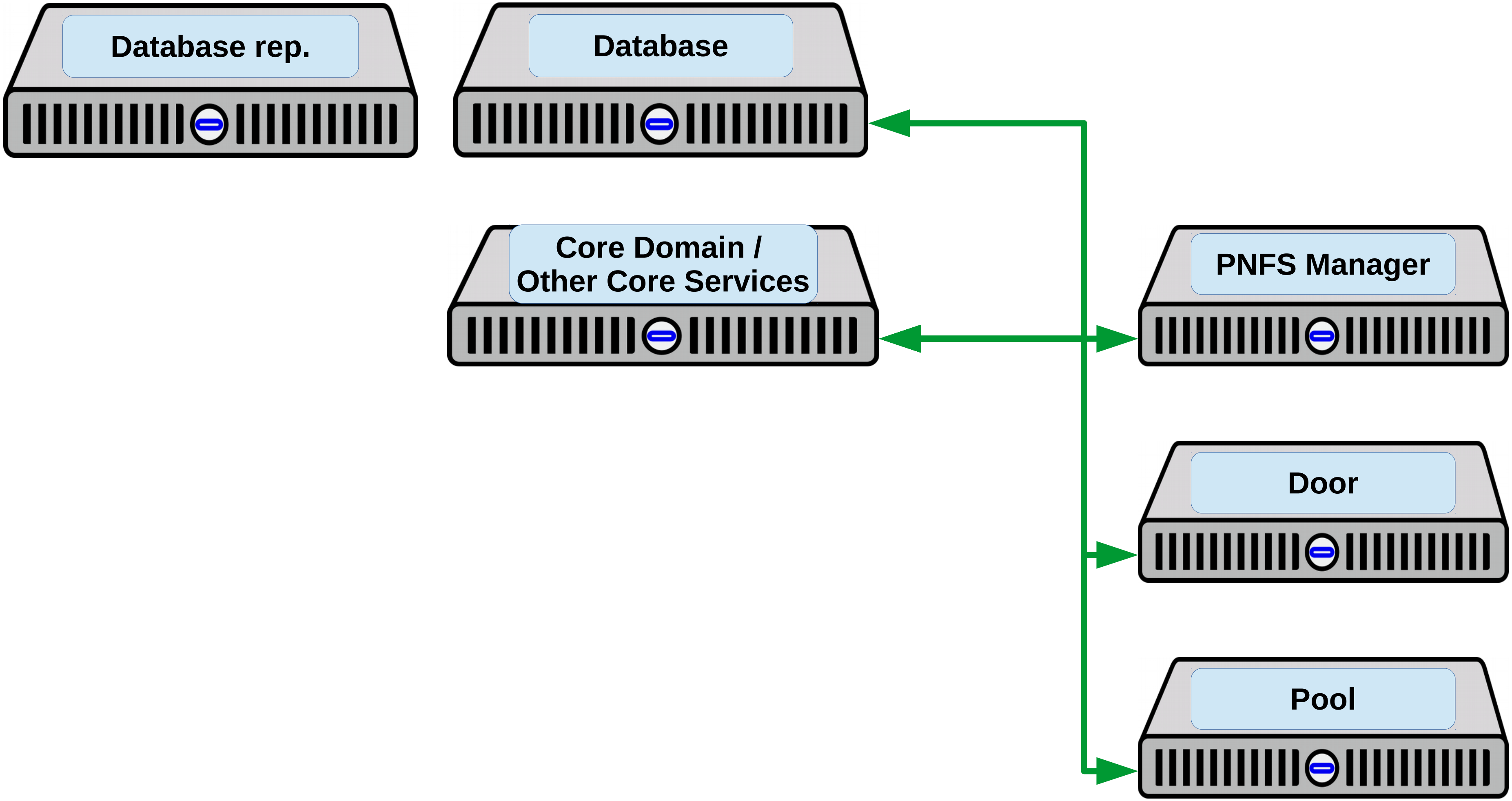
Changing to an HA architecture

- Plan ahead, with network topology in mind
- Set up a ZooKeeper cluster of at least three nodes
- Replicate PostgreSQL Server
- Update dCache to 3.0
  - connect to ZK
- add instances as needed

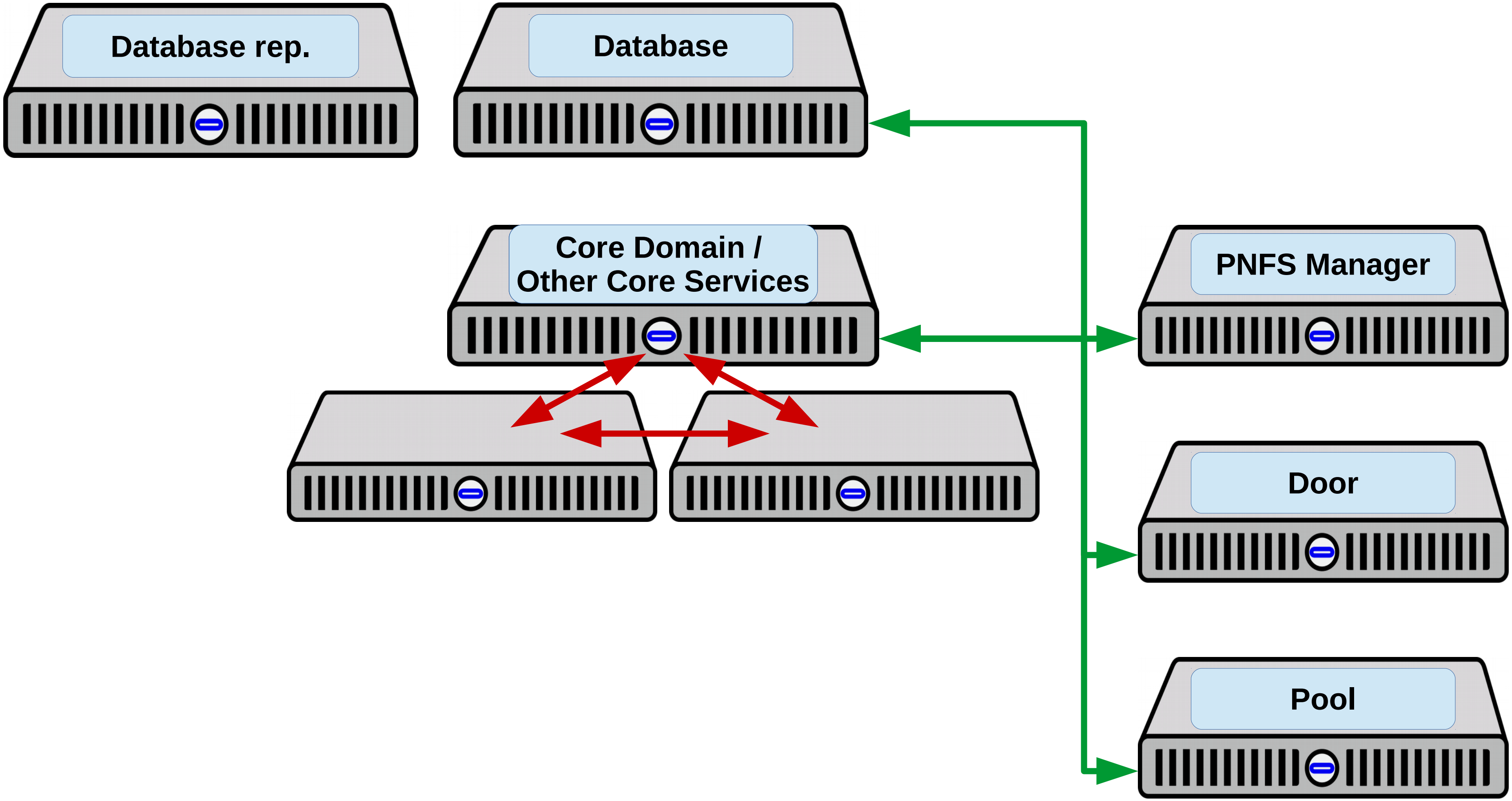
Topology changes for HA



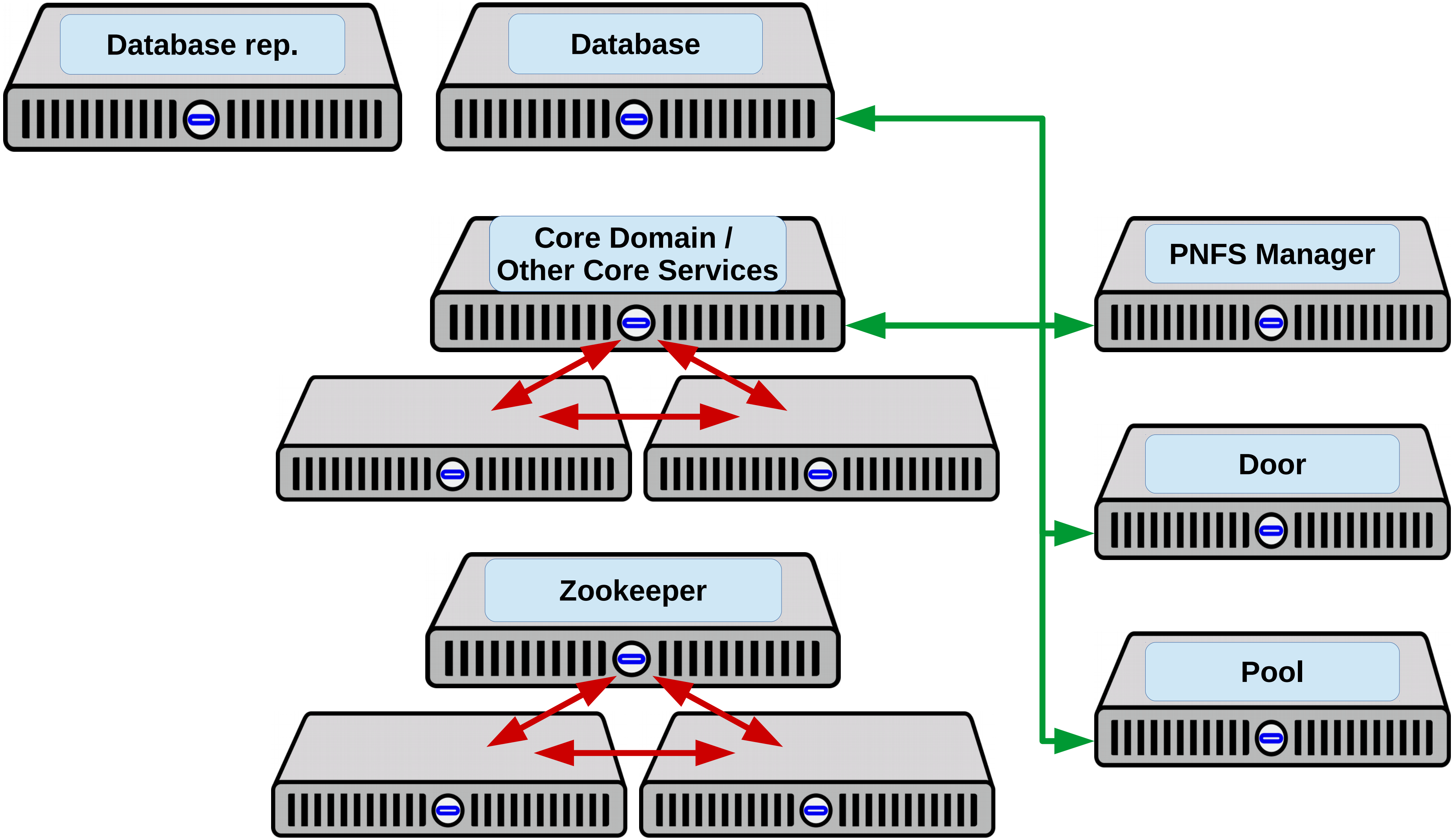
Topology changes for HA



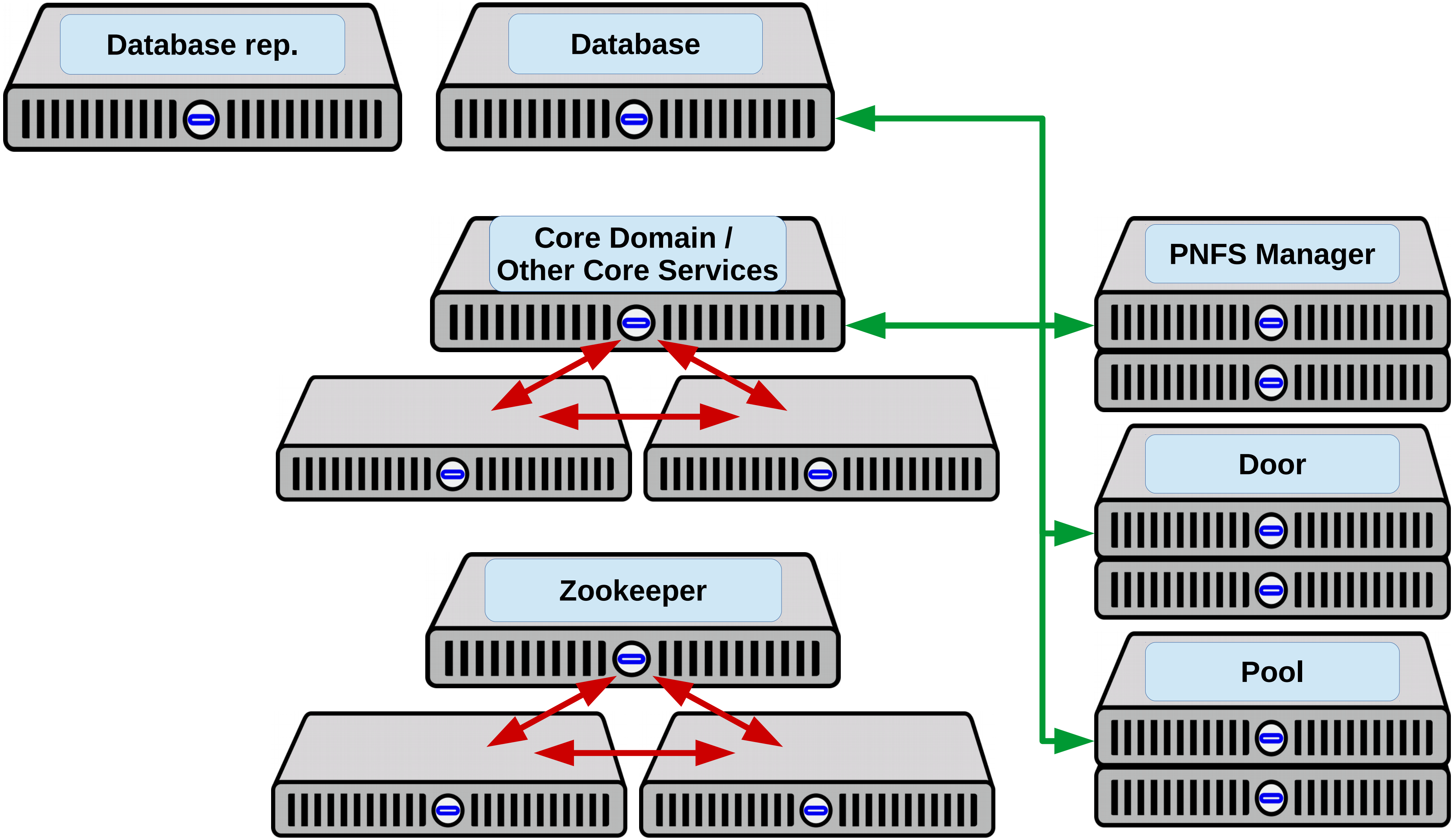
Topology changes for HA



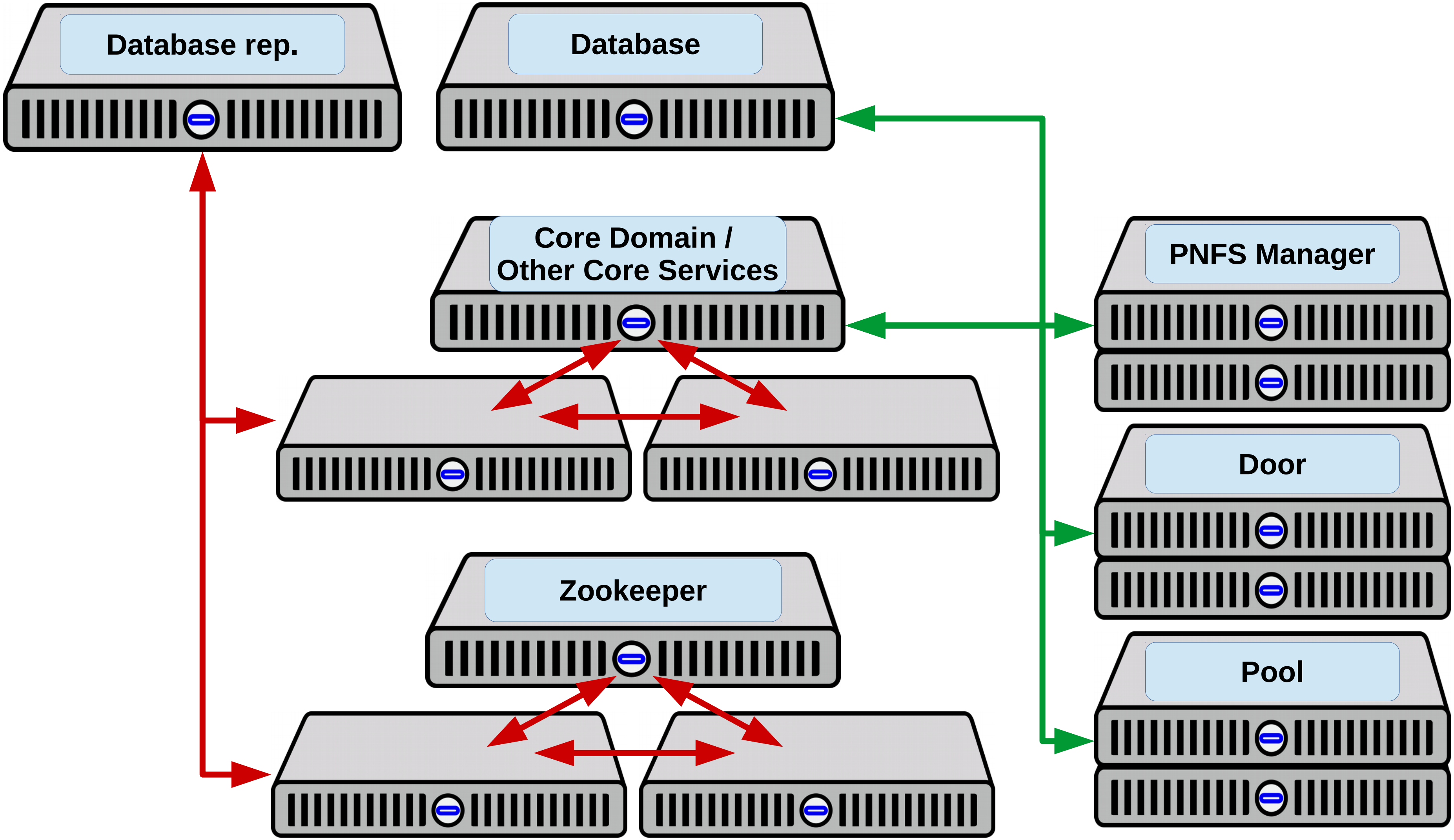
Topology changes for HA



Topology changes for HA

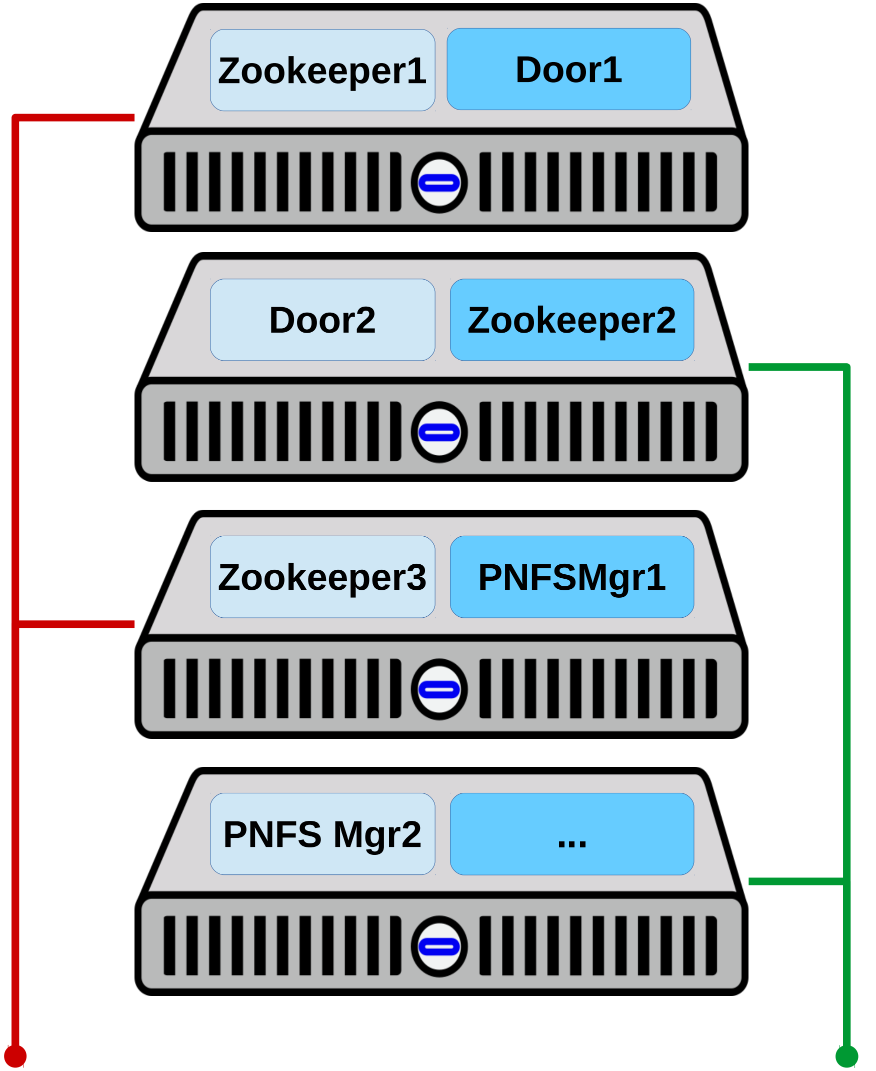


Topology changes for HA



Saving hardware

- Instead of adding physical machines, distribute services
- Plan according to needs for availability, load /scalability



Demo