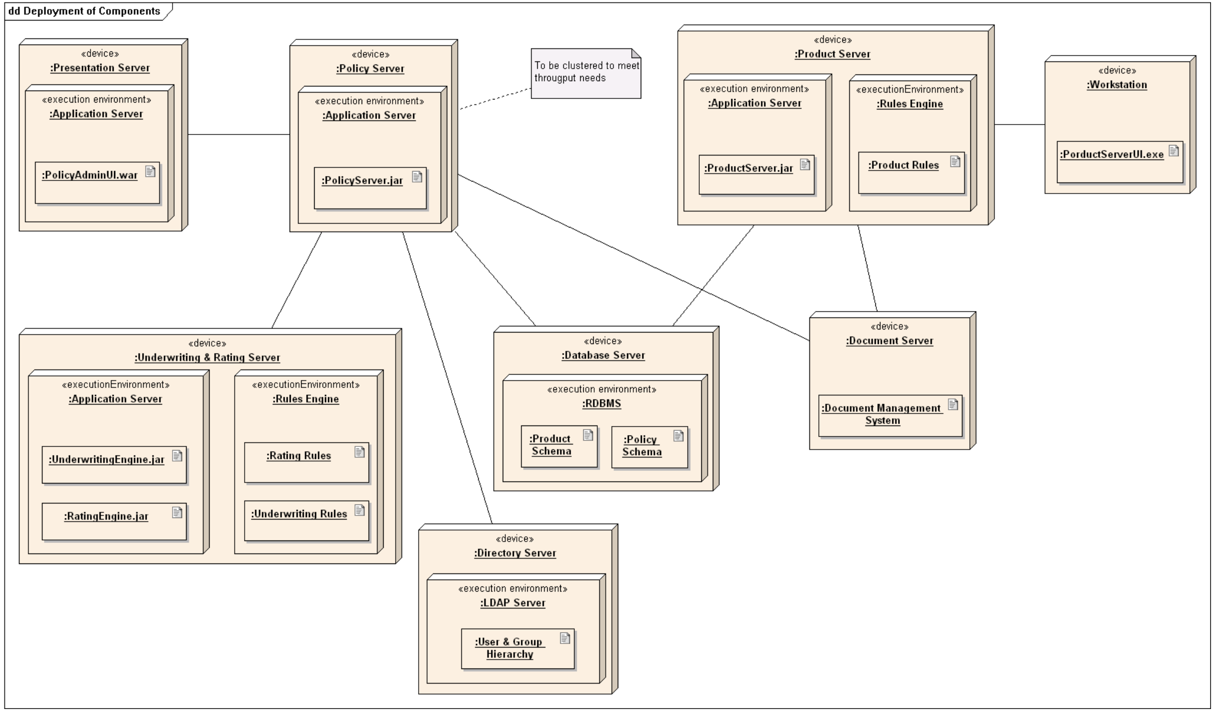
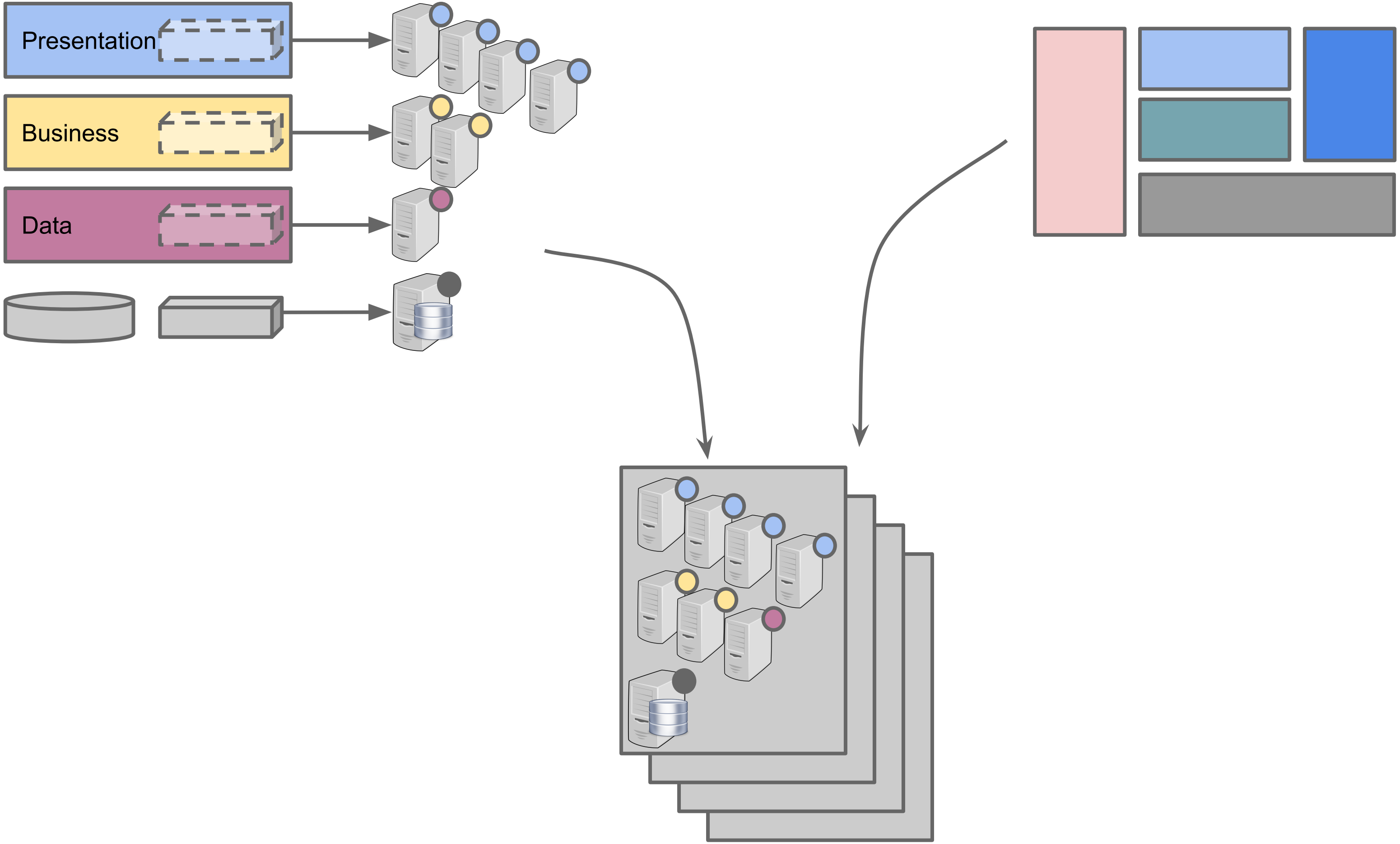
Stateful Services on Mesos

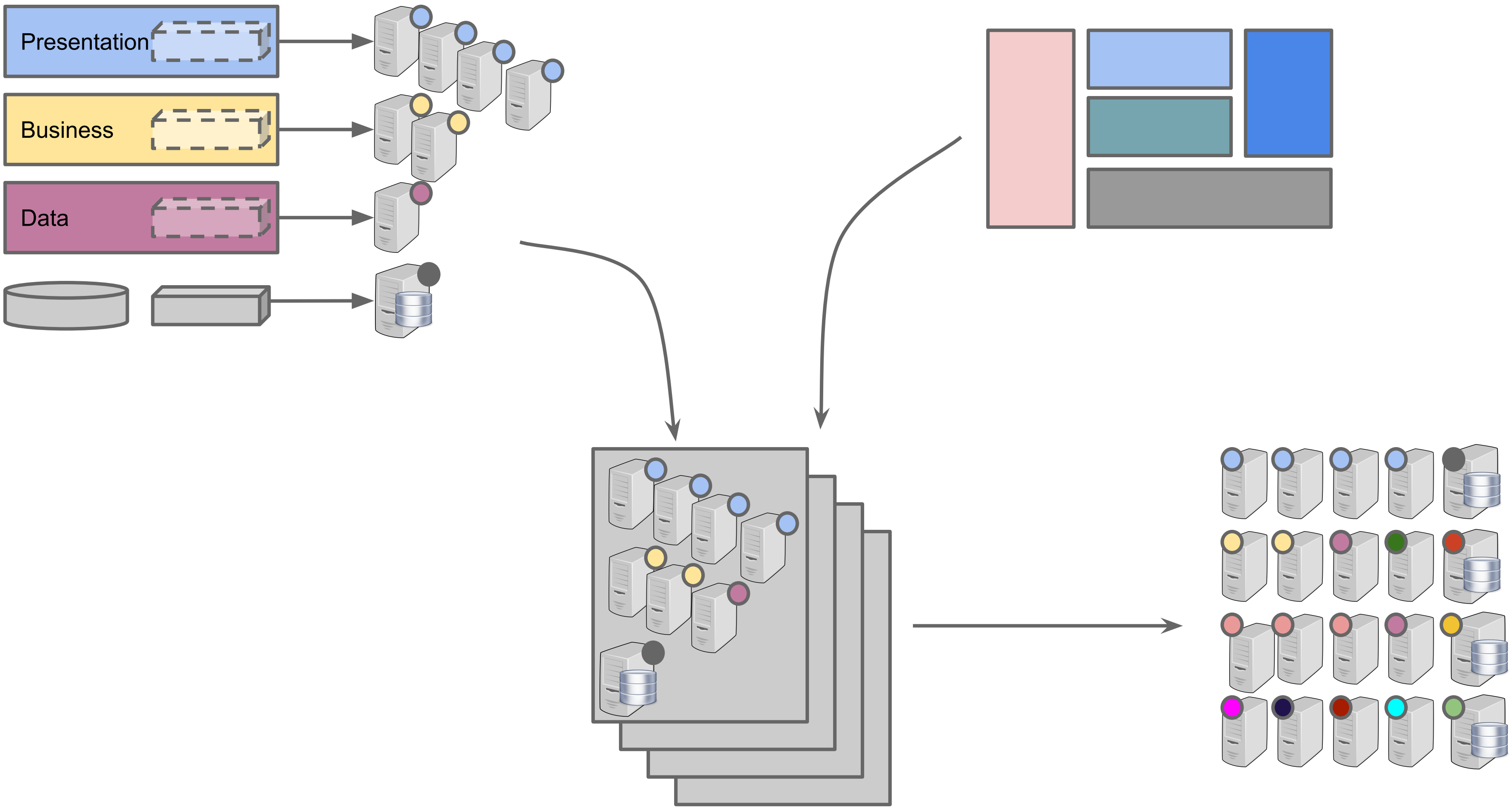
Ankan Mukherjee (ankan@moz.com)
Arunabha Ghosh (agh@moz.com)

A deployment diagram

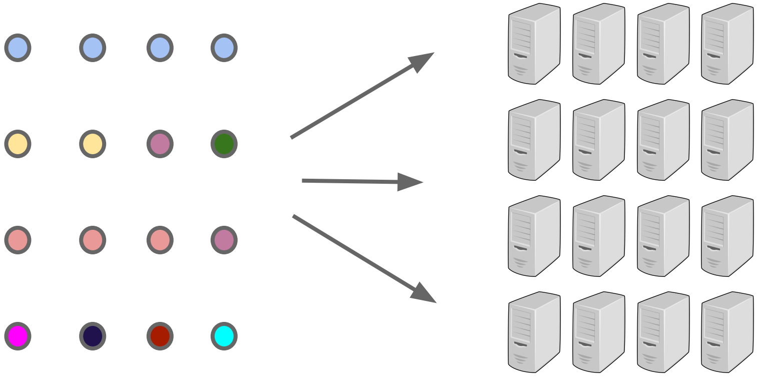


Source: wikipedia


Presentation
Business
Data

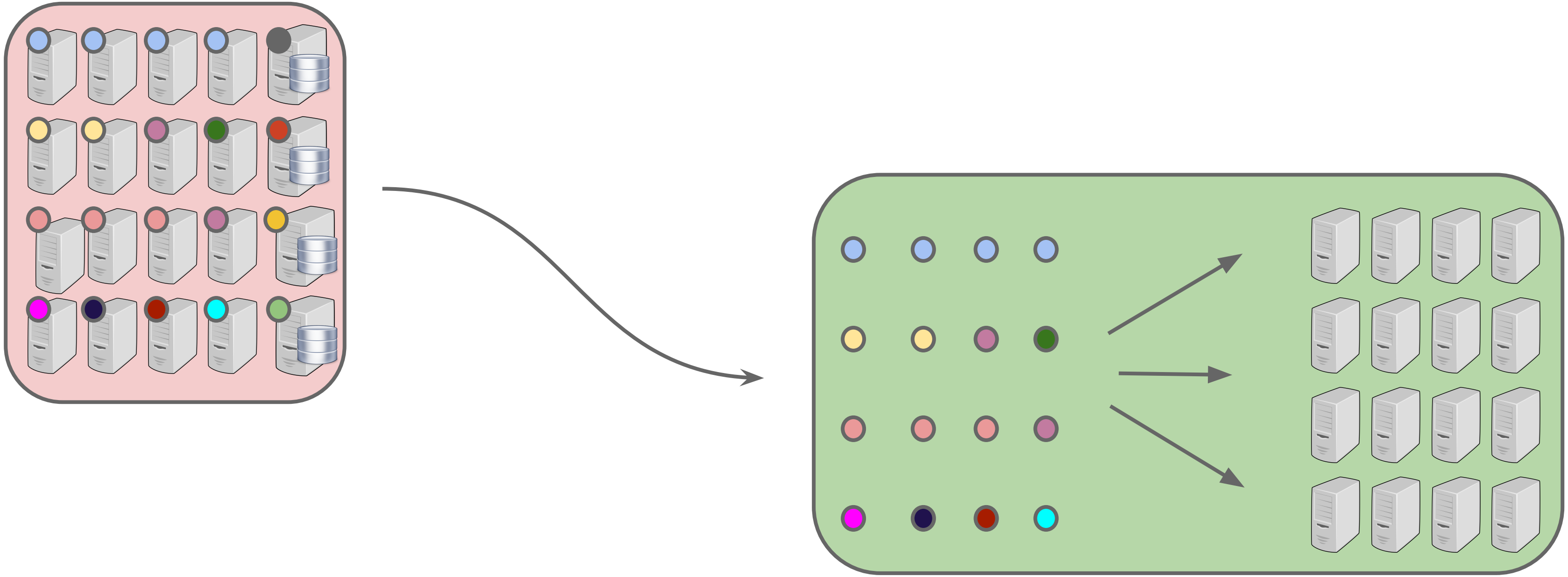

Presentation
Business
Data

Why run on Mesos?



- Services are decoupled from the nodes
- Automatic failover
- Easier to manage/maintain
- Simpler version management
- Simpler environments, staging → deployment
- Lesser complexity of the set of systems

Transition

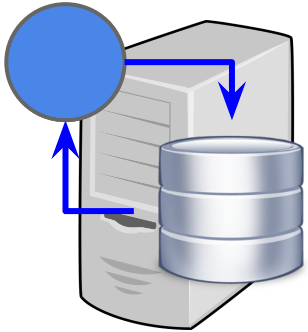


Challenges

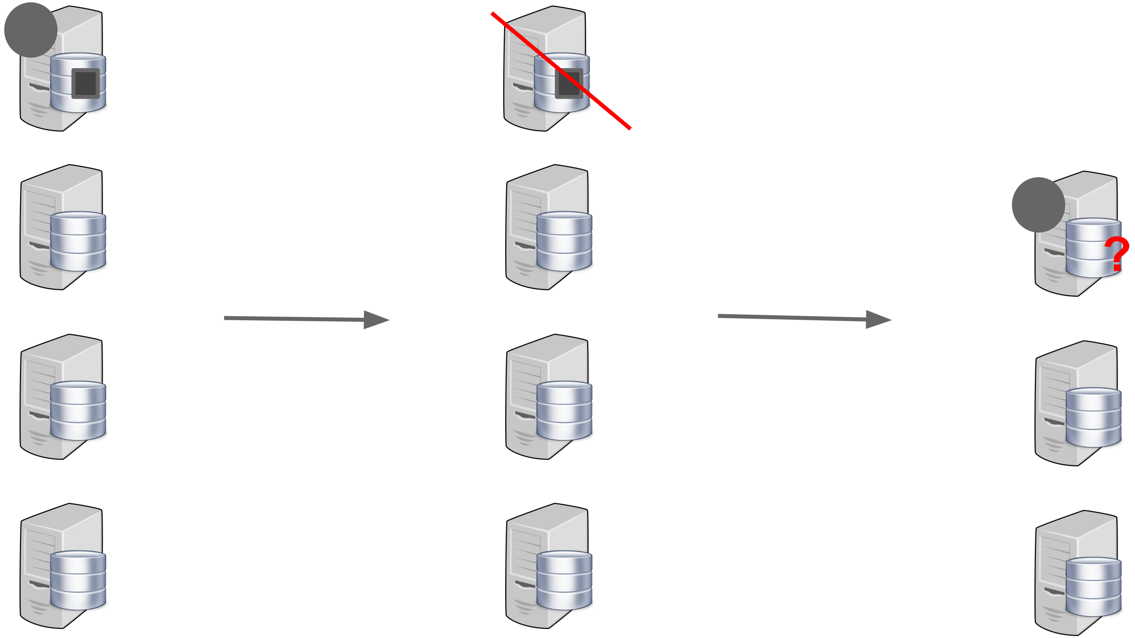
- Packaging/deployment
- Naming/finding services
- Dependency on persistent state

Challenges

- Packaging/deployment
- Naming/finding services
- Dependency on persistent state



The problem



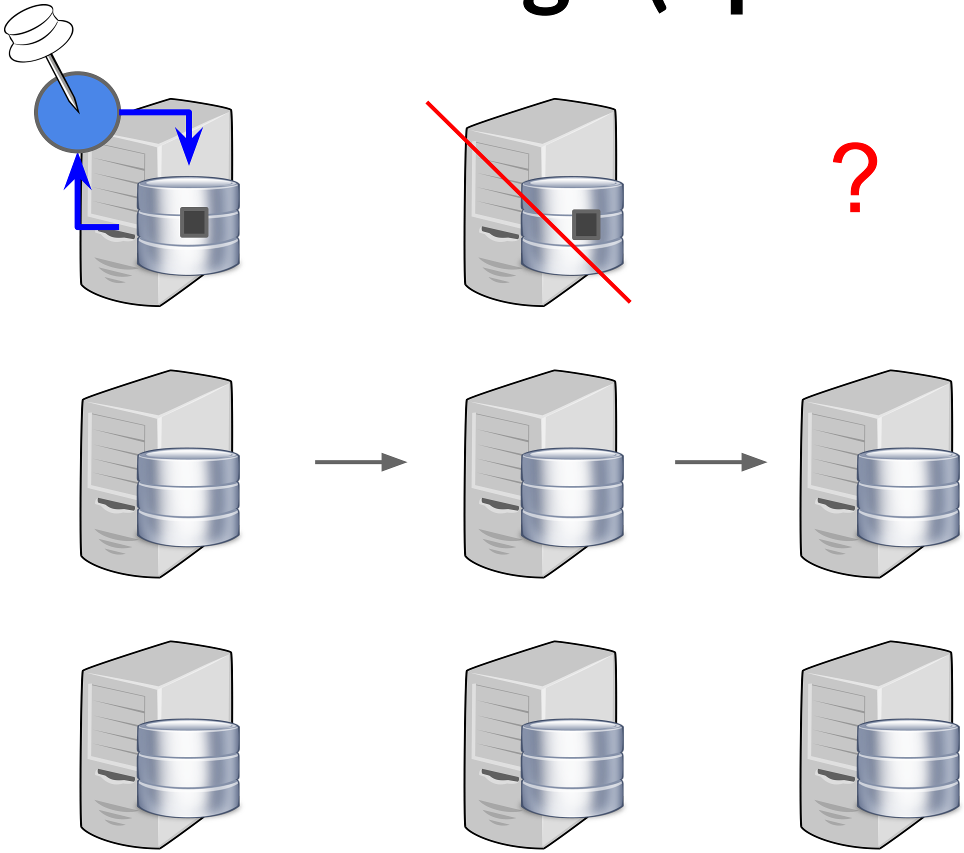
Examples:

- Legacy apps
- Single node SQL databases (mysql, postgres)
- Apps that depend on local storage

Potential Solutions

- Local storage
- Shared storage
- Network block device
- Mesos persistent resource primitives
- Application specific distributed solutions

Local storage (option 1)

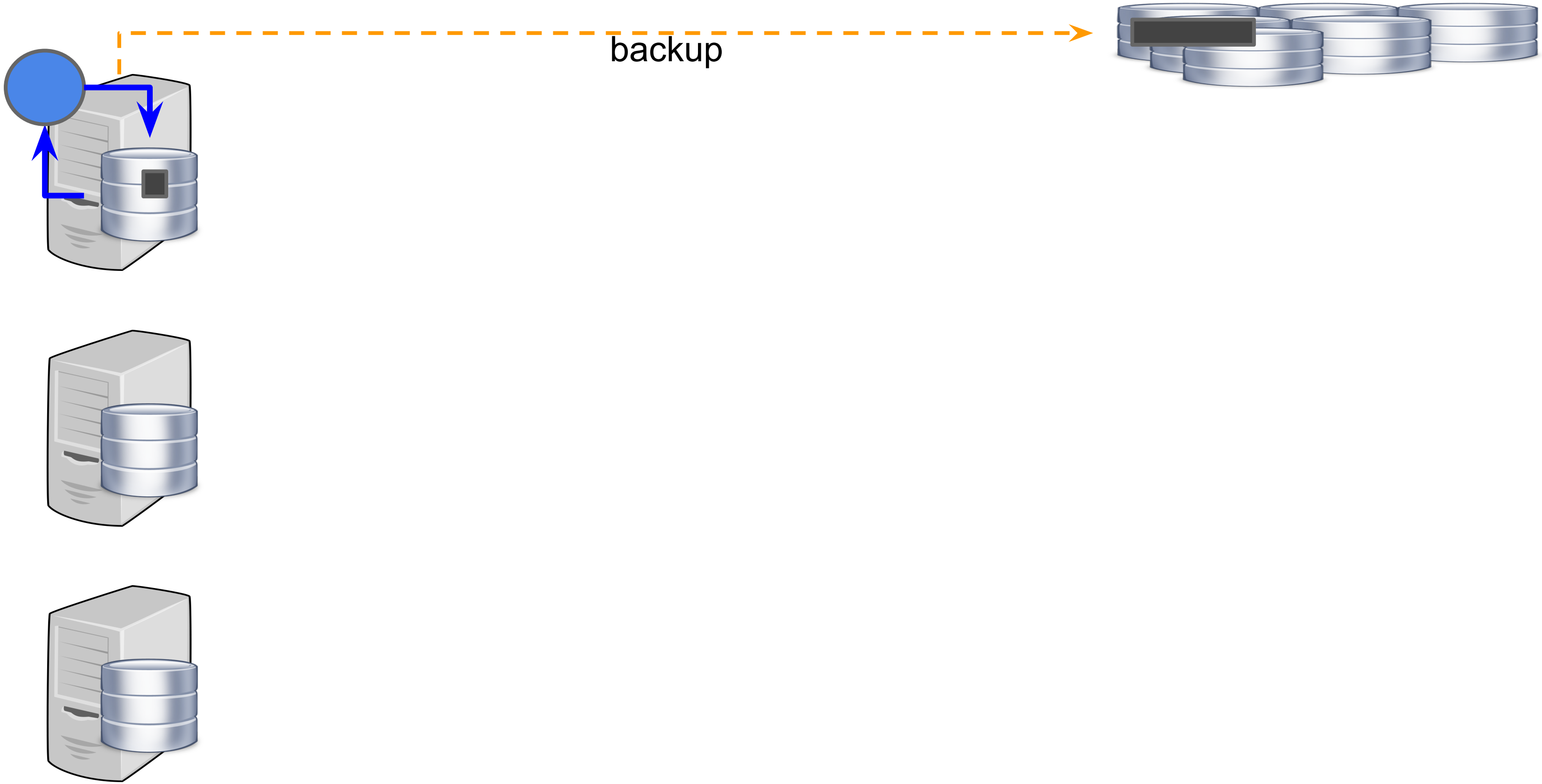


- Pin to node
- On failure
    - Manually bring the node up
    - Rely on existing process

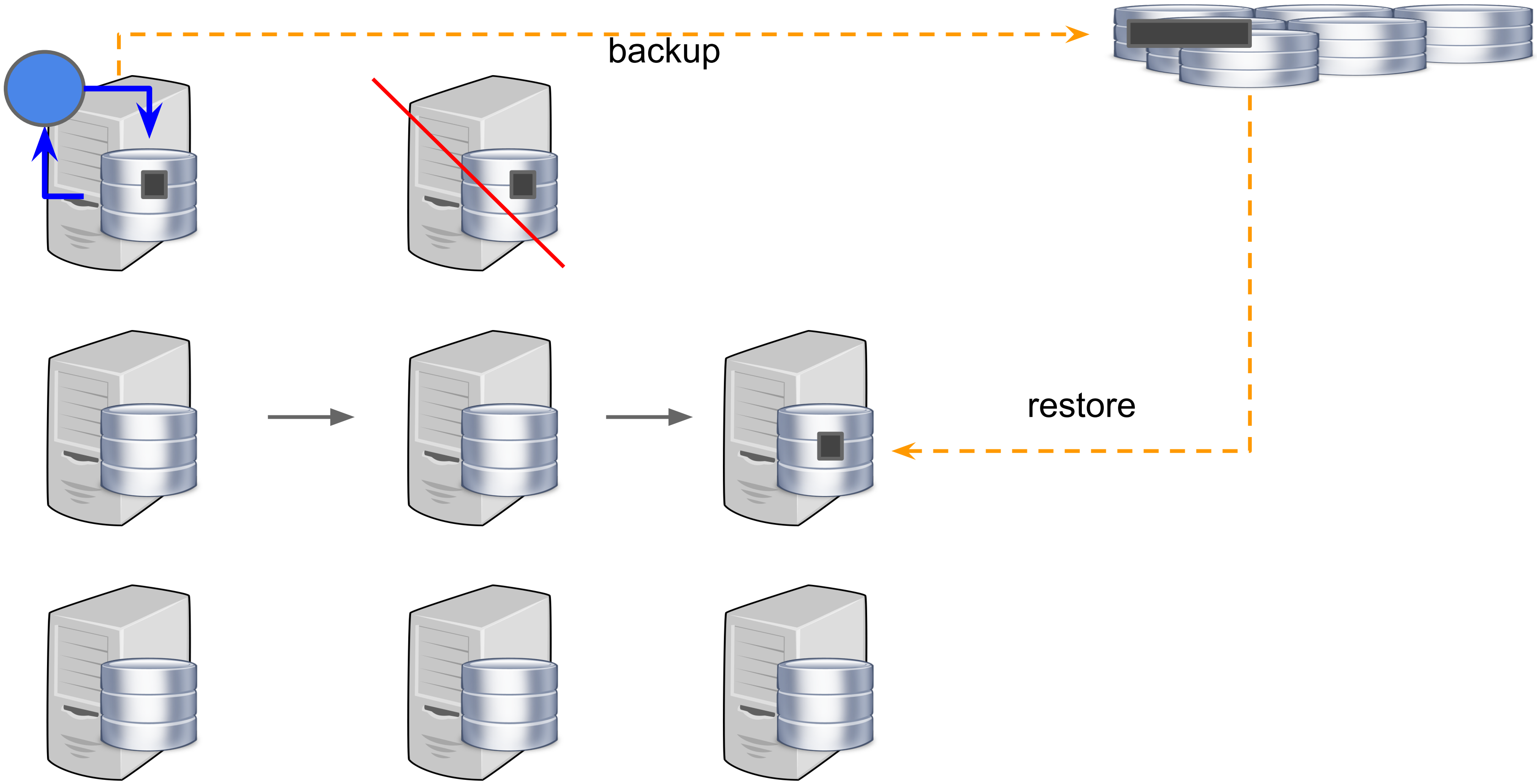
Local storage (option 1)

- Pros
  - Easiest (~ no changes)
  - Share free resources from node
- Cons
  - No auto failover
  - Service still coupled to node
  - *Feels like cheating!*

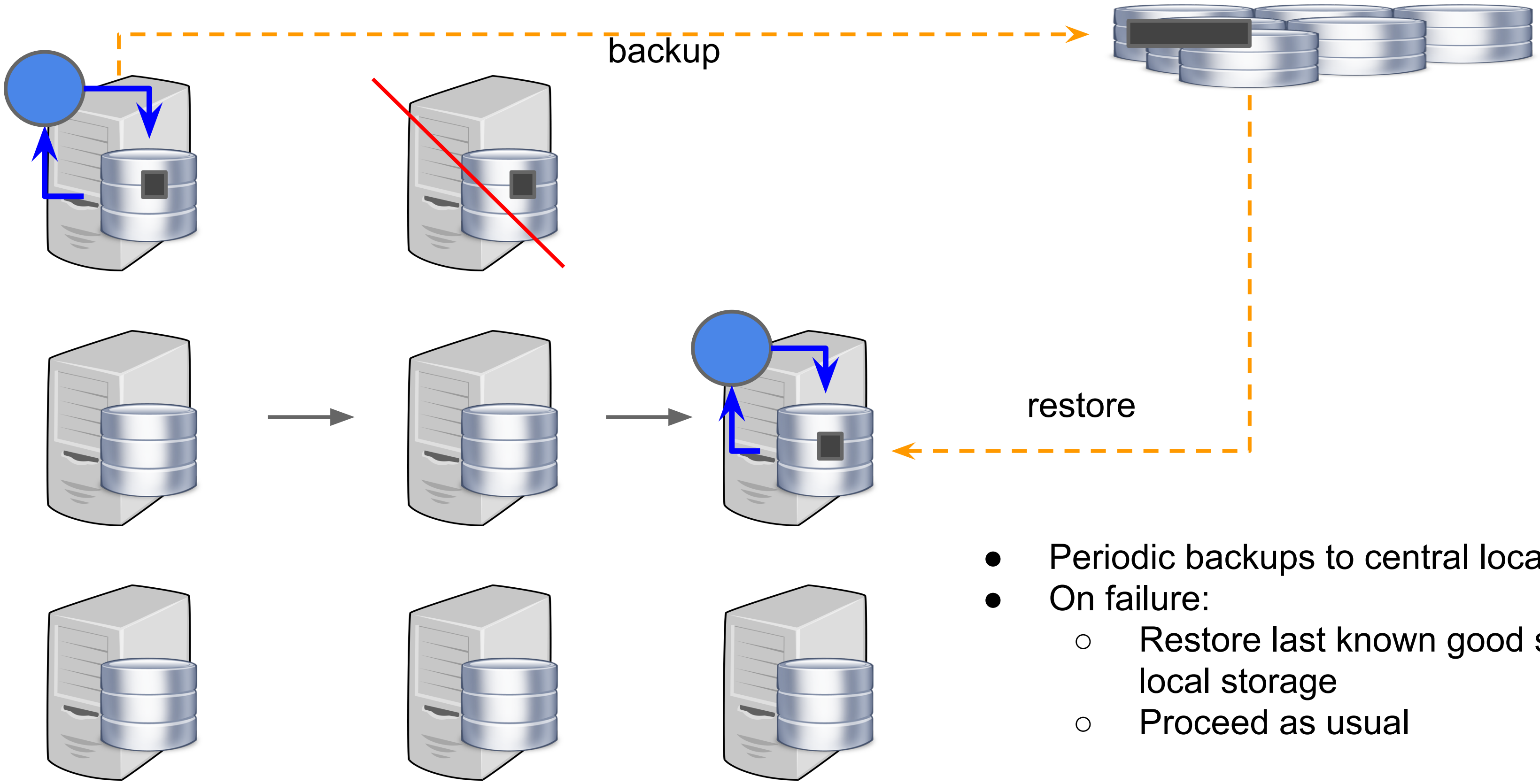
Local storage (option 2)



Local storage (option 2)



Local storage (option 2)

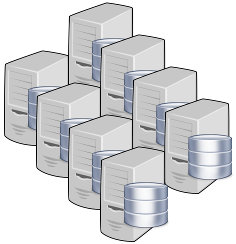


- Periodic backups to central location
- On failure:
  - Restore last known good state to local storage
  - Proceed as usual

Local storage (option 2)

- When and where to backup?

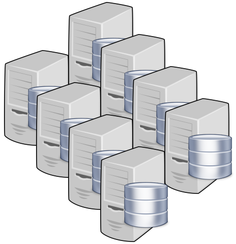
- When and where to restore?
  - Which node?
  - Which backup?



Local storage (option 2)

- When and where to backup?

- When and where to restore?
  - Which node?
  - Which backup?

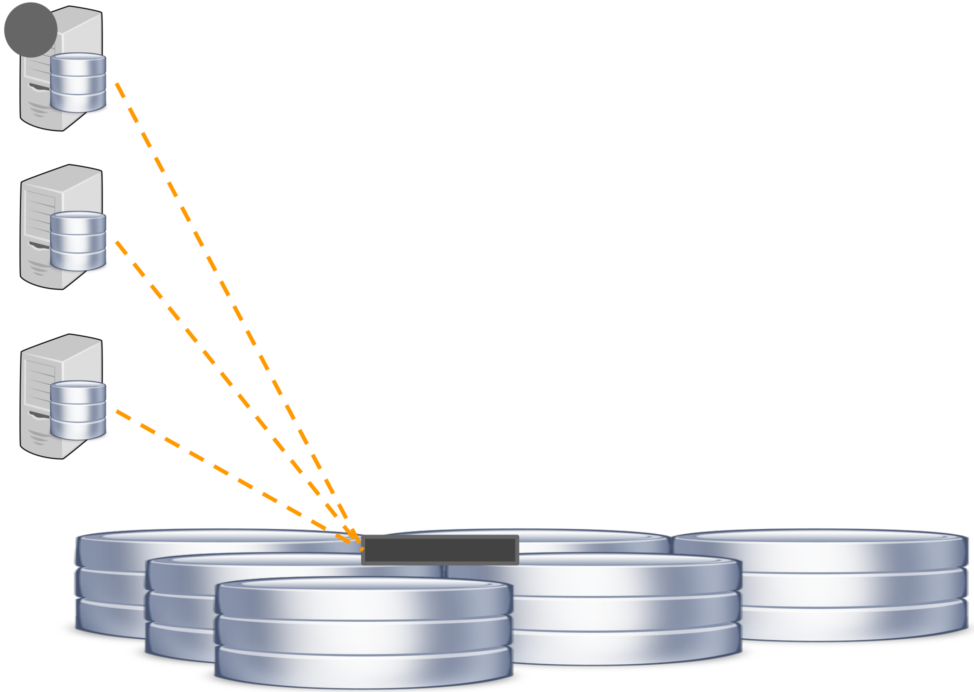


“Automated scripted restore at process start.”

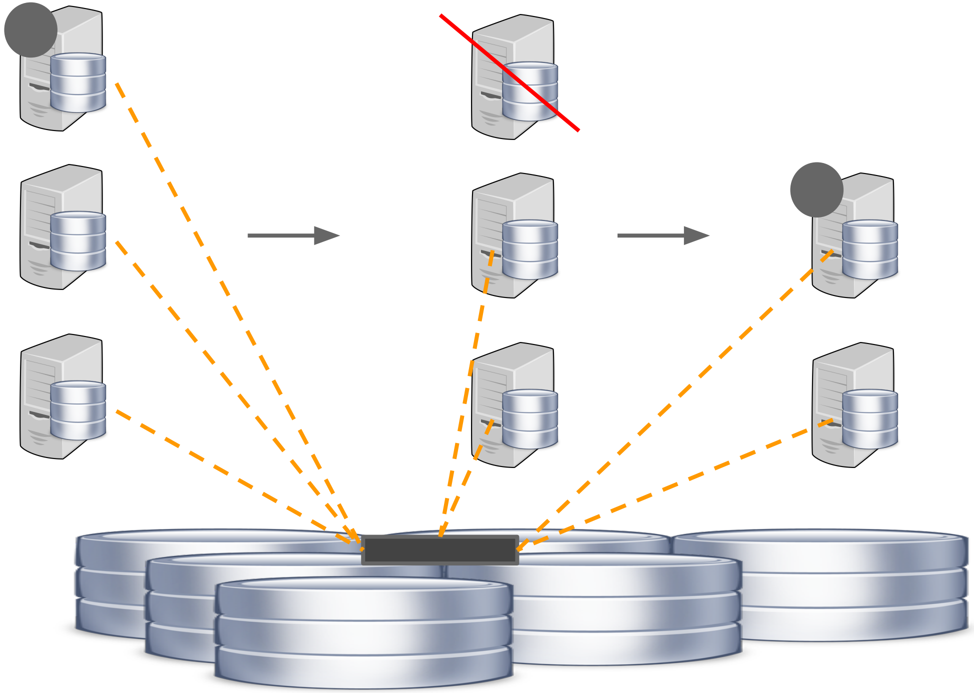
Local storage (option 2)

- Pros:
  - Easy to set up
  - Auto failover
  - Share free resources
- Cons:
  - Scripted restore complexity
  - Adversely affected by system & data volume/type
  - Time to restore
  - Data loss

Shared file system - centralized



Shared file system - centralized

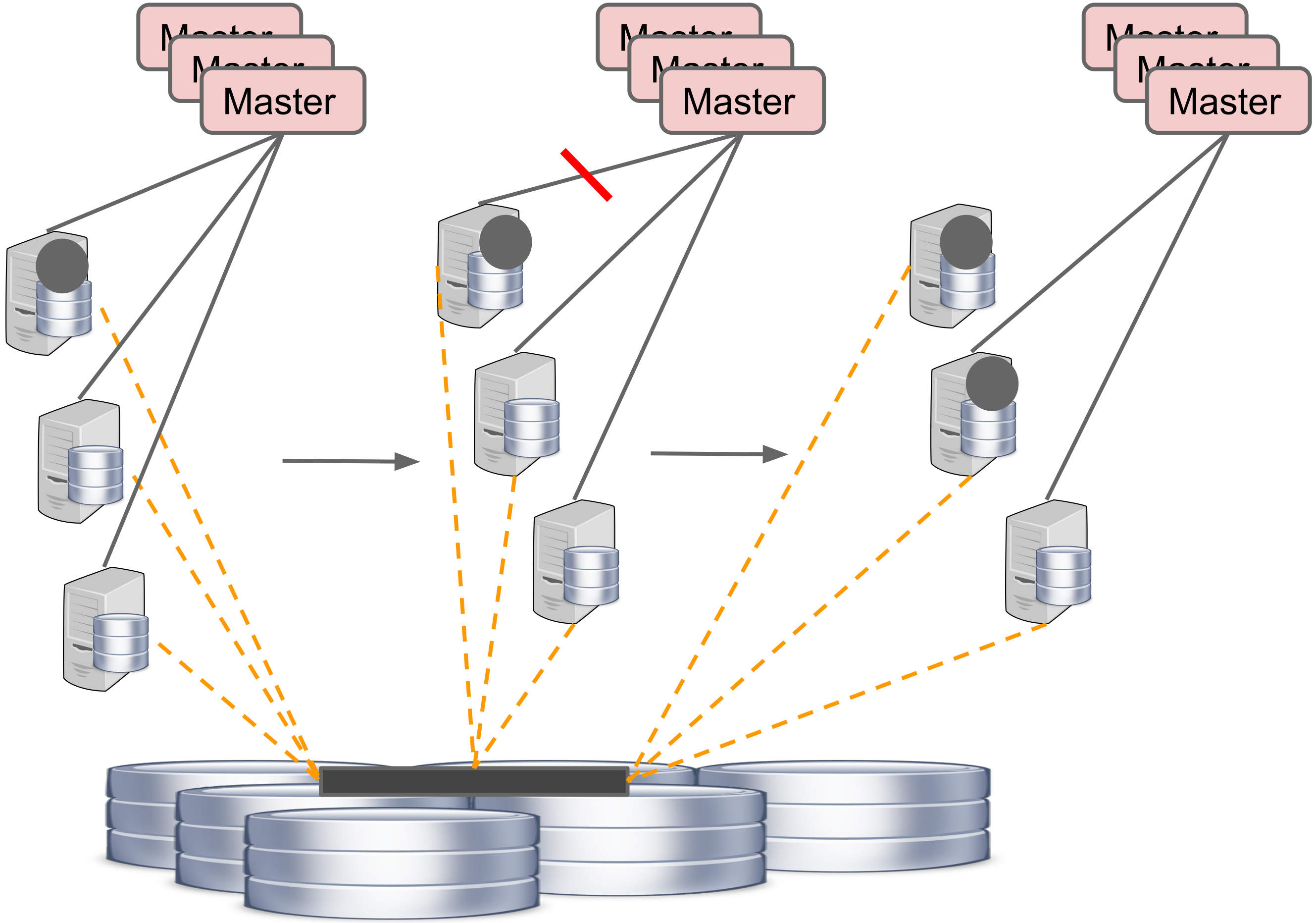


- POSIX compliant centralized shared FS
- Example: NFS
- Mounted to same path across all nodes
- On failure:
    - Let Mesos start new instance on any available node

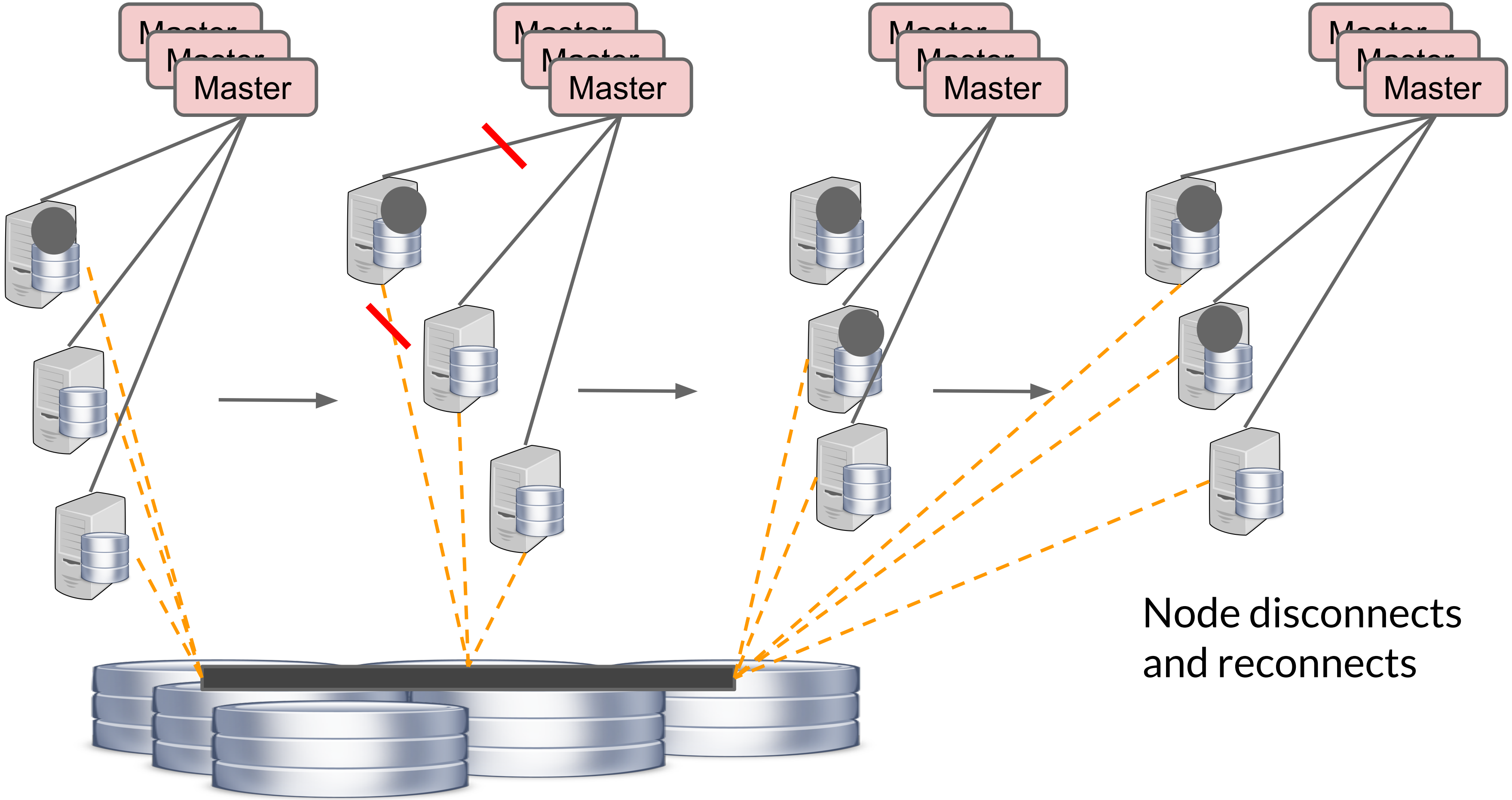
Shared file system - centralized

What can go wrong?

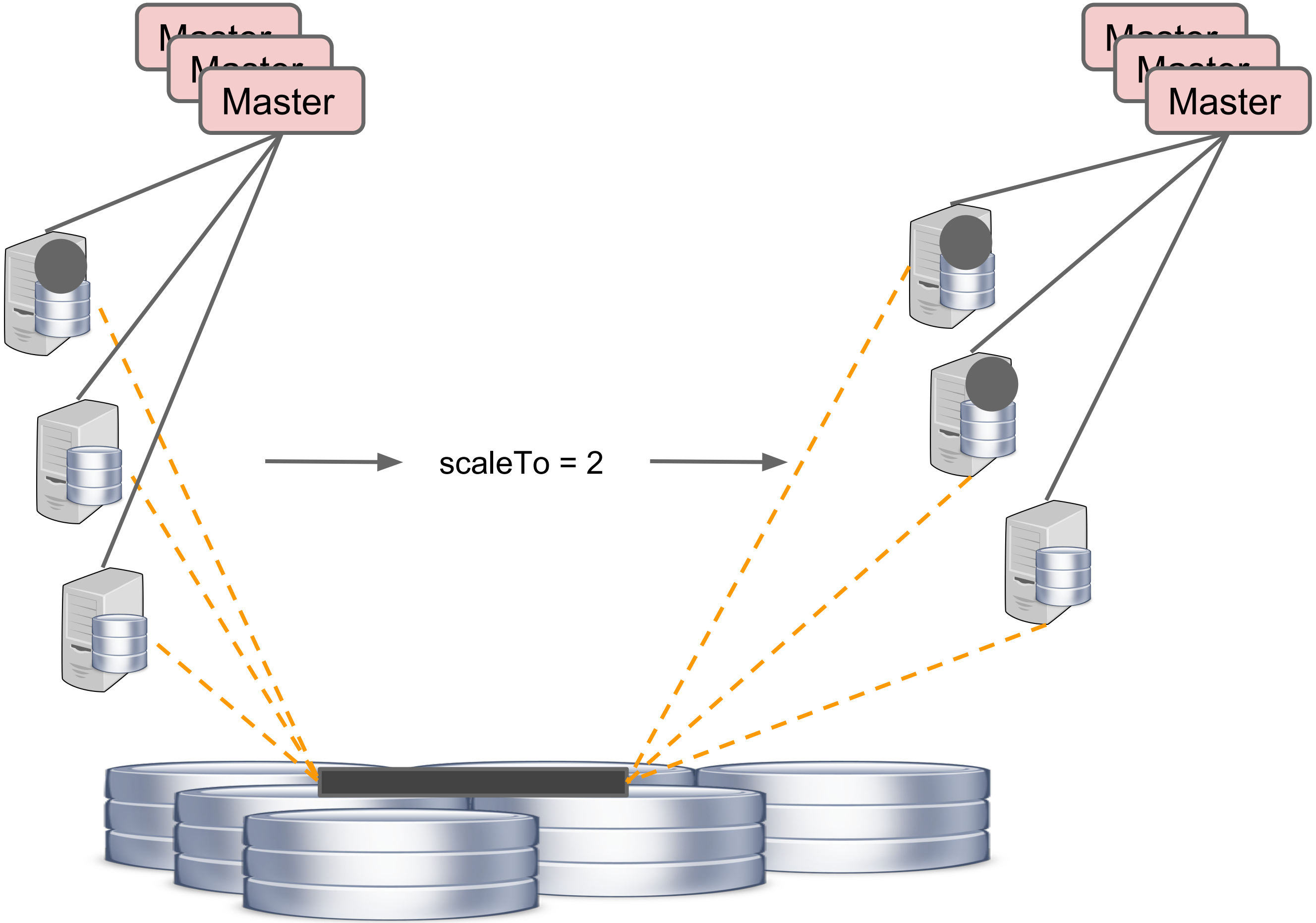
- What did we just do?
    - Added network between the process and the storage



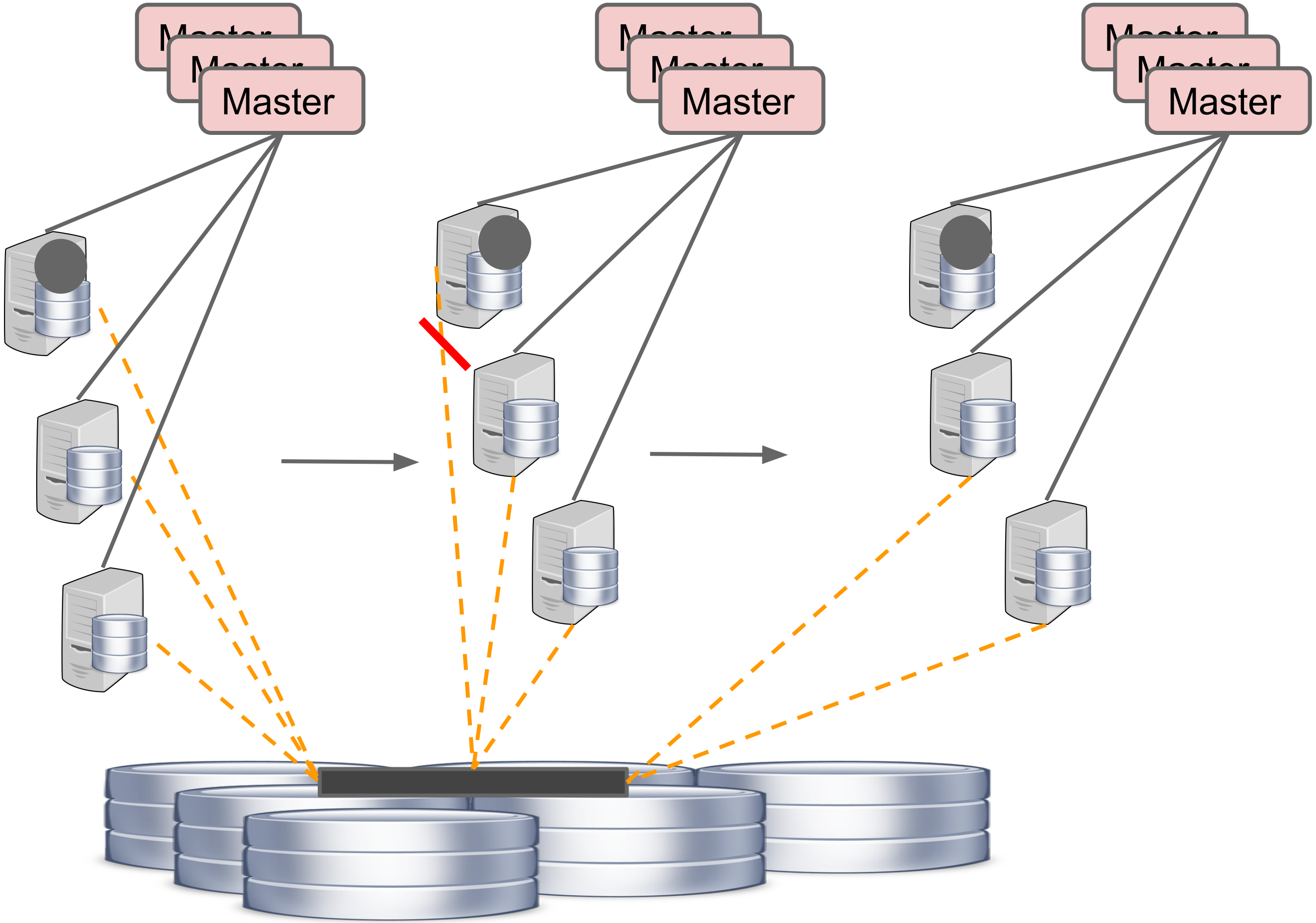
Node disconnects from master



Node disconnects
and reconnects



Task is scaled to >1



Node disconnects from FS

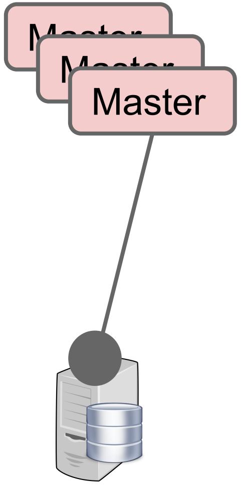
Shared file system - centralized

To summarize, we could end up with…

- Possibly corrupted data if
    - Node disconnects from master but is connected to FS
    - Node disconnects from network & then connects back
    - Somehow the task is “scaled” to >1 instances
- Possibly undesired state of process/service if
    - Node is connected to master but disconnects from FS

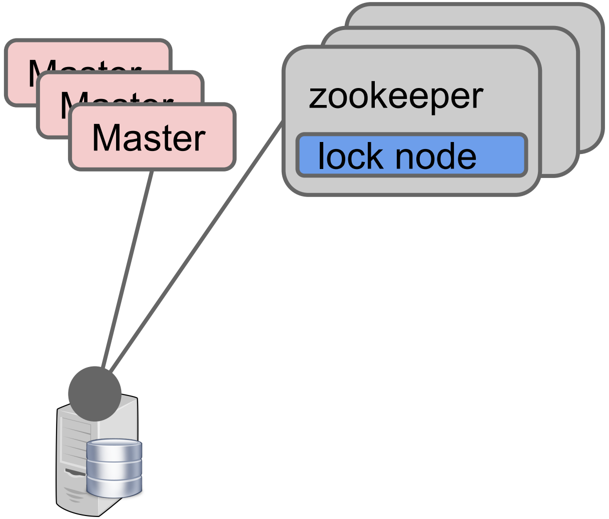
Shared file system - centralized

How do we fix this?



Shared file system - centralized

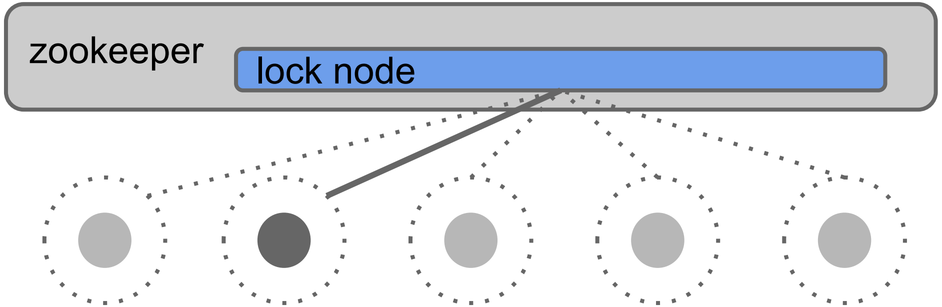
How do we fix this?



- Use zookeeper exclusive lock
- The process should
  - start only if it has acquired the zk lock (exit otherwise)
  - exit at any point it loses the zk lock
- Check for FS mount and exit if NA

Shared file system - centralized

- How without changing orig app?
  - New startup app/script (wrapper)
  - entrypoint/startup → wrapper → orig app



Shared file system - centralized

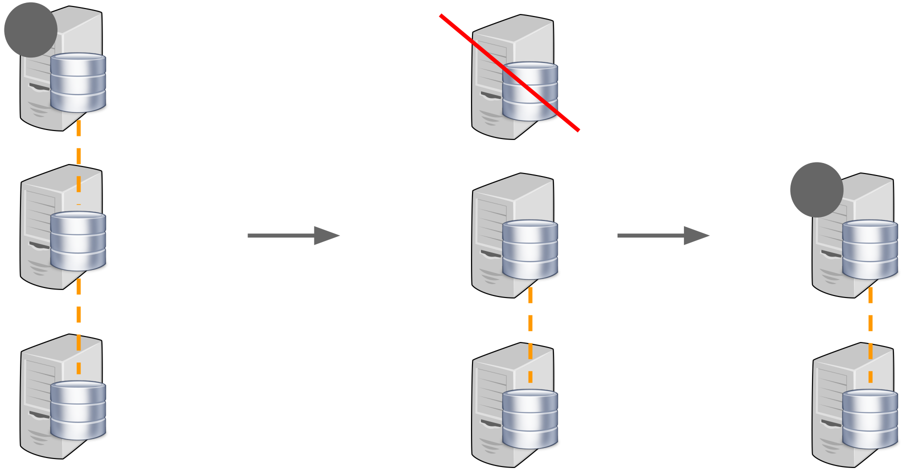
Check:

- Possibly corrupted data if
    - Node disconnects from master but is connected to FS
    - Node disconnects from network & then connects back
    - Somehow the task is “scaled” to >1 instances
- Possibly undesired state of process/service if
    - Node is connected to master but disconnects from FS

Shared file system - centralized

- Pros:
  - Easy to set up
  - Process benefits from most features (except scaling)
- Cons:
  - Handle mutual exclusion (but this is fairly simple)
  - Depends on network speed/latency

Shared file system - distributed

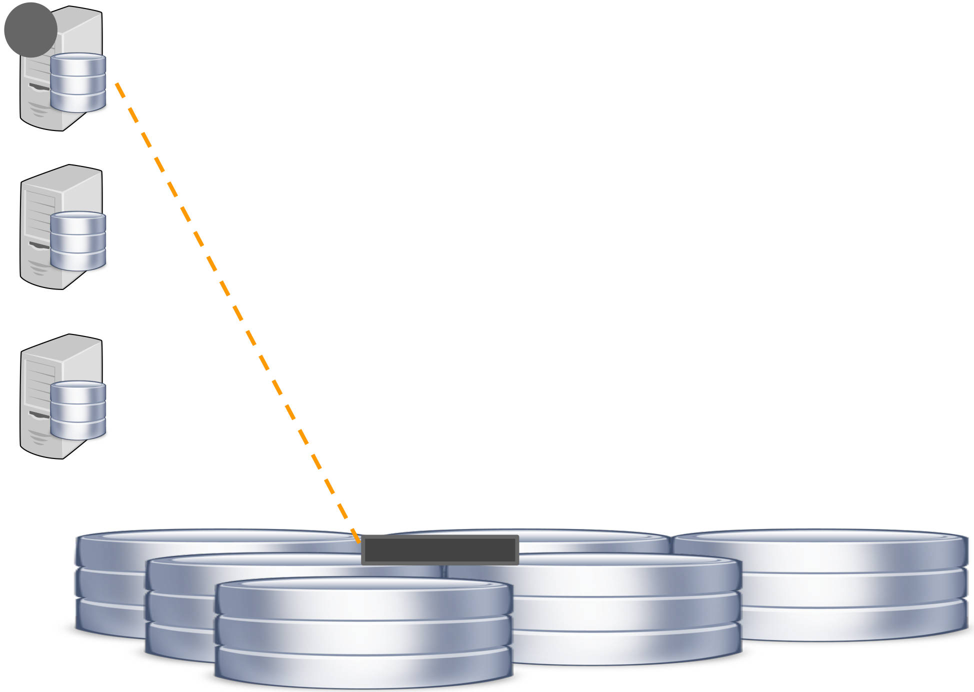


- POSIX compliant distributed shared FS
- Examples: glusterfs, MooseFS, Lustre
- Mounted to same path across all nodes
- On failure:
  - Let Mesos start new instance on any available node

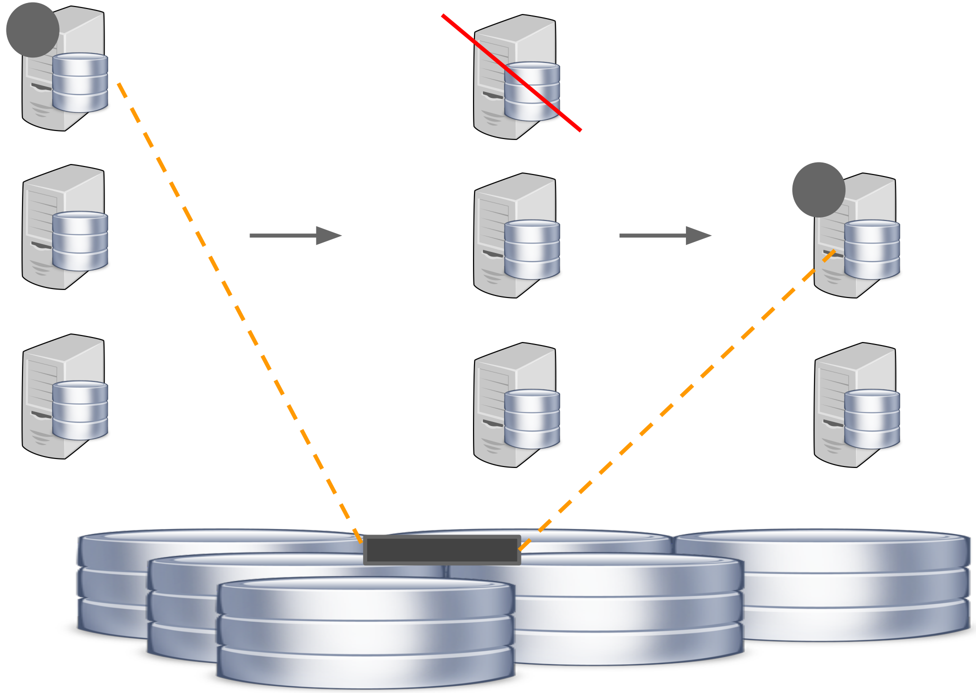
Shared file system - distributed

- Similar to centralized shared FS
- Pros:
  - Process benefits from most features (except scaling)
- Cons:
  - Similar as centralized shared FS
  - Setup may be complex
  - Replication, data distribution, processing overhead, etc.

Network Block Device



Network Block Device



- Somewhat between local and shared FS
- Device mounted to only 1 node at a time
- On node failure:
    - Repair & mount device to new node
    - Proceed as usual

Network Block Device

- Pros
  - Lesser overhead than a high level protocol like NFS.
- Cons
  - Slightly more difficult to manage.
  - Failover is not automatic
    - Need to mount to new node (scripted).
  - May need to repair the FS on the NBD at startup (run fsck before mount)

Persistent State Resource Primitives

- New features
  - Storage as a resource
  - Keep data across process restarts
  - Process affinity to data with node (on node restarts)
- Easier to work with storage

Application Specific Solutions

- For mysql:
  - Vitess
  - Mysos (Apache Cotton)
- Pros
  - Replication and availability built in
  - Scalable
- Cons
  - Relatively more involved setup
  - NA for most applications

Stateful services we’re running

- mysql
- postgresql
- mongodb (single, clustered soon)
- redis
- rethinkdb
- elasticsearch (single, clustered)

Best Practices / Lessons Learnt

- Mount dir at the same point (path)
- Multi-level backup as storage may be SPOF
  - Disk based ones like RAID
  - App specific ones like mysqldump
- Leverage services like zookeeper for mutual exclusion

Best Practices / Lessons Learnt

- Isolate applications at this layer
    - Based on
        - disk space & usage
        - disk iops & usage
        - network bandwidth & usage
    - Use multiple mounts, specific allocation, etc.
- Set up adequate monitoring & alerting

Conclusion

- Although not a natural fit, it is possible to gainfully run stateful services in Mesos.

- Should be approached as an engineering problem rather than one with a generic or ideal solution.

Performance Test

- Disclaimer
  - Very much dependent on the setup, network, etc.
  - YMMV!
- Setup
  - `local* : ~ 2000r / 1000w IOPS`
  - `nfs500 : ~  500 IOPS`
  - `nfs1000: ~ 1000 IOPS`

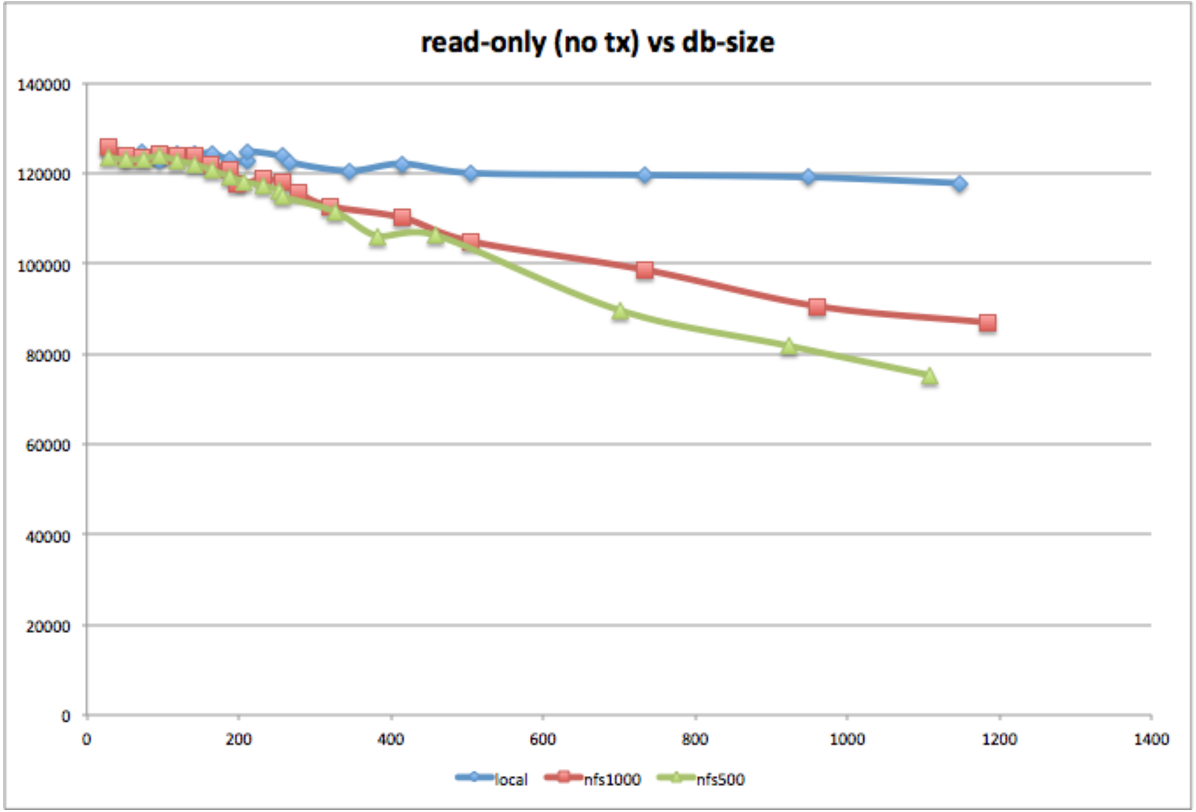
`*24 10k SAS disks in RAID 10`

Performance Test

- System
  - Single node mysql server
  - Buffer pool size: 128 M
- Tests
  - sysbench tests run for 300 seconds
    - default RO & RW tests
    - custom WO tests with no reads
    - single thread

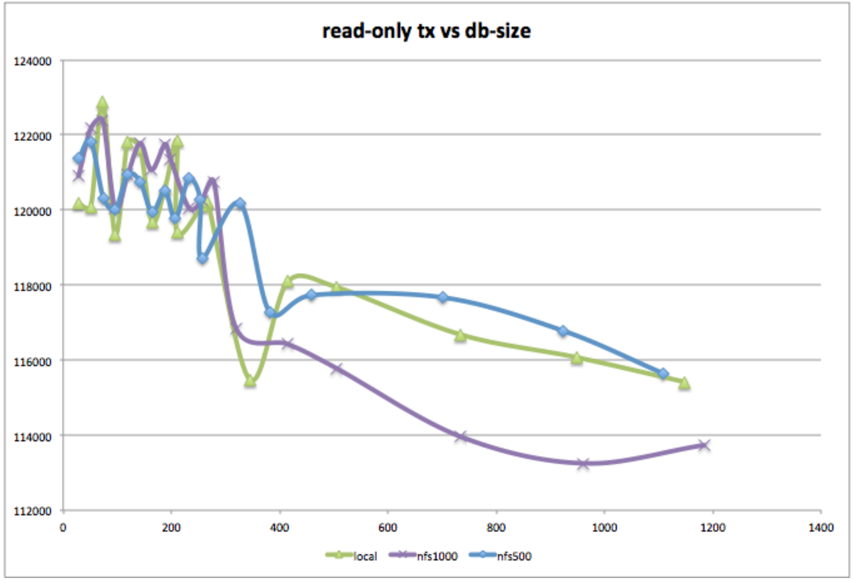
Performance Test

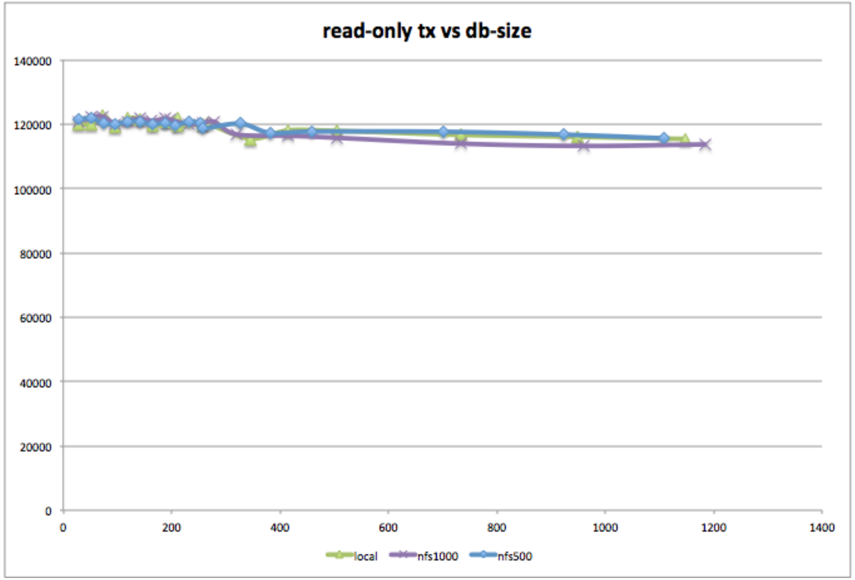
- Read only queries
- No Begin/Commit



Performance Test

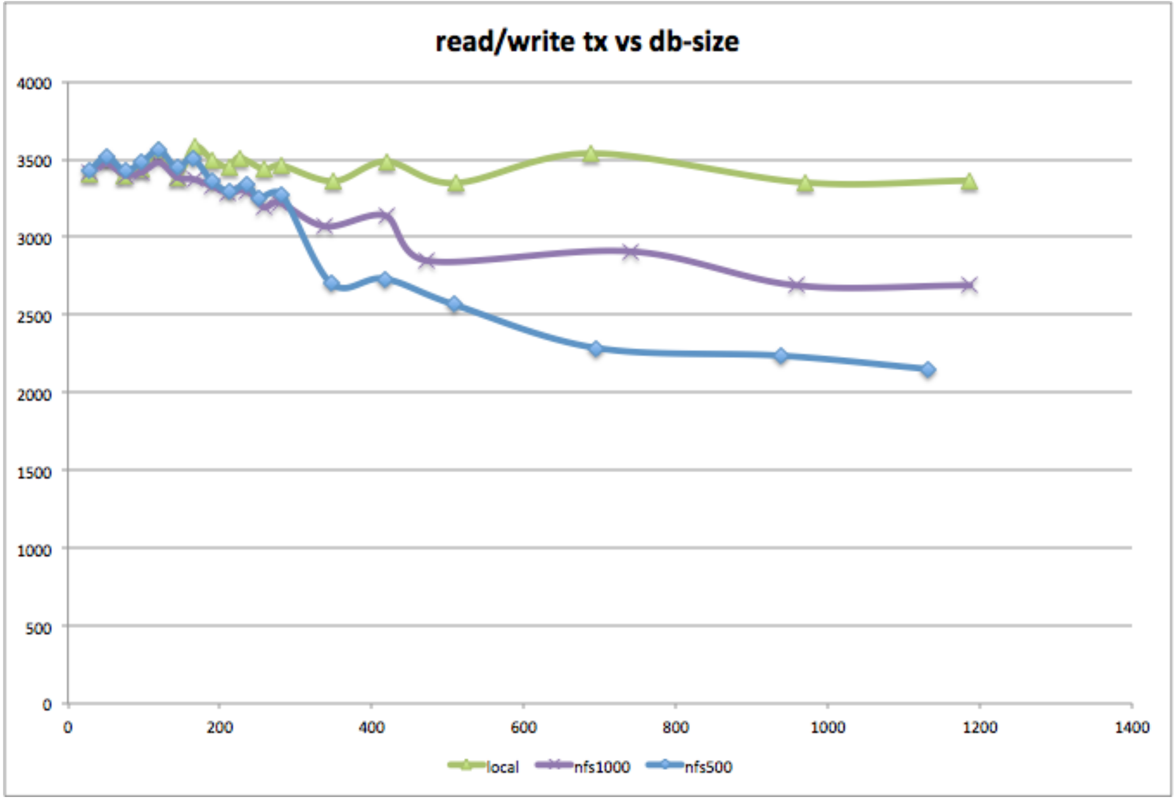
- Read only queries
- With Begin/Commit





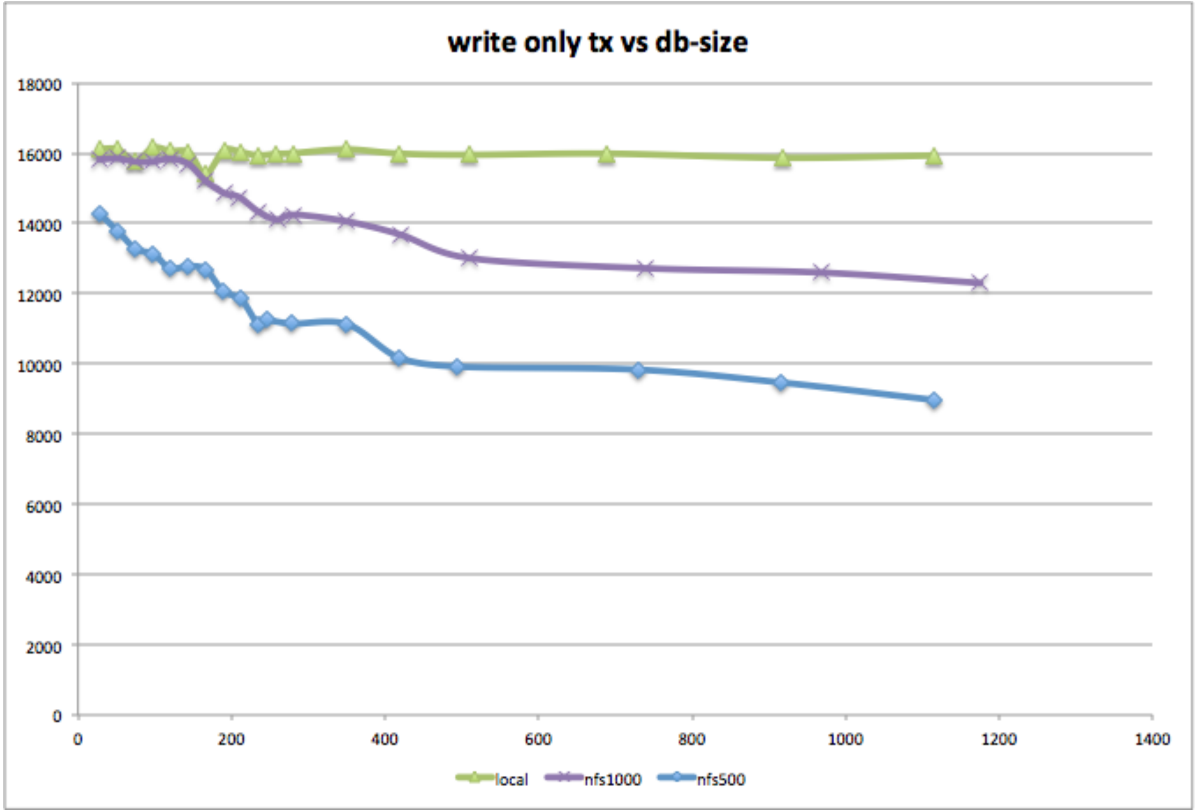
Performance Test

- Read/Write queries
- With Begin/Commit
- 26% write queries



Performance Test

- Write only queries
- With Begin/Commit



Performance Test

- For read heavy queries
  - increasing buffer pool size may compensate for performance decrease with network FS.

- For write heavy queries
  - memory size is less relevant as these are disk bound.

Thanks!