Building reliable Ceph clusters with SUSE Enterprise Storage

Survival skills for the real world

Lars Marowsky-Brée
Distinguished Engineer
lmb@suse.com

What this talk is not

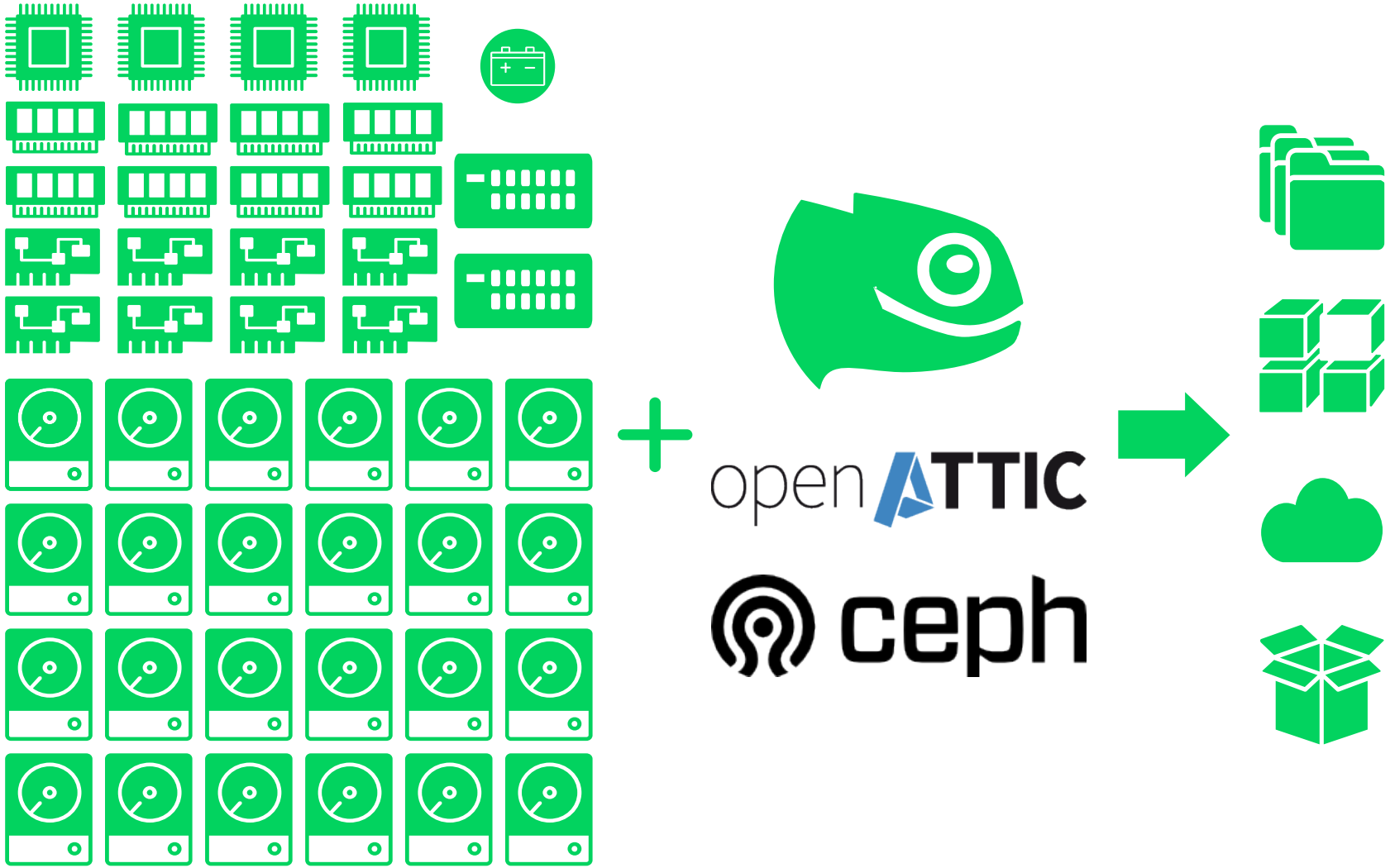
- A comprehensive introduction to Ceph
- SUSE Enterprise Storage roadmap discussion
- A discussion of Ceph performance tuning

SUSE Enterprise Storage - Reprise

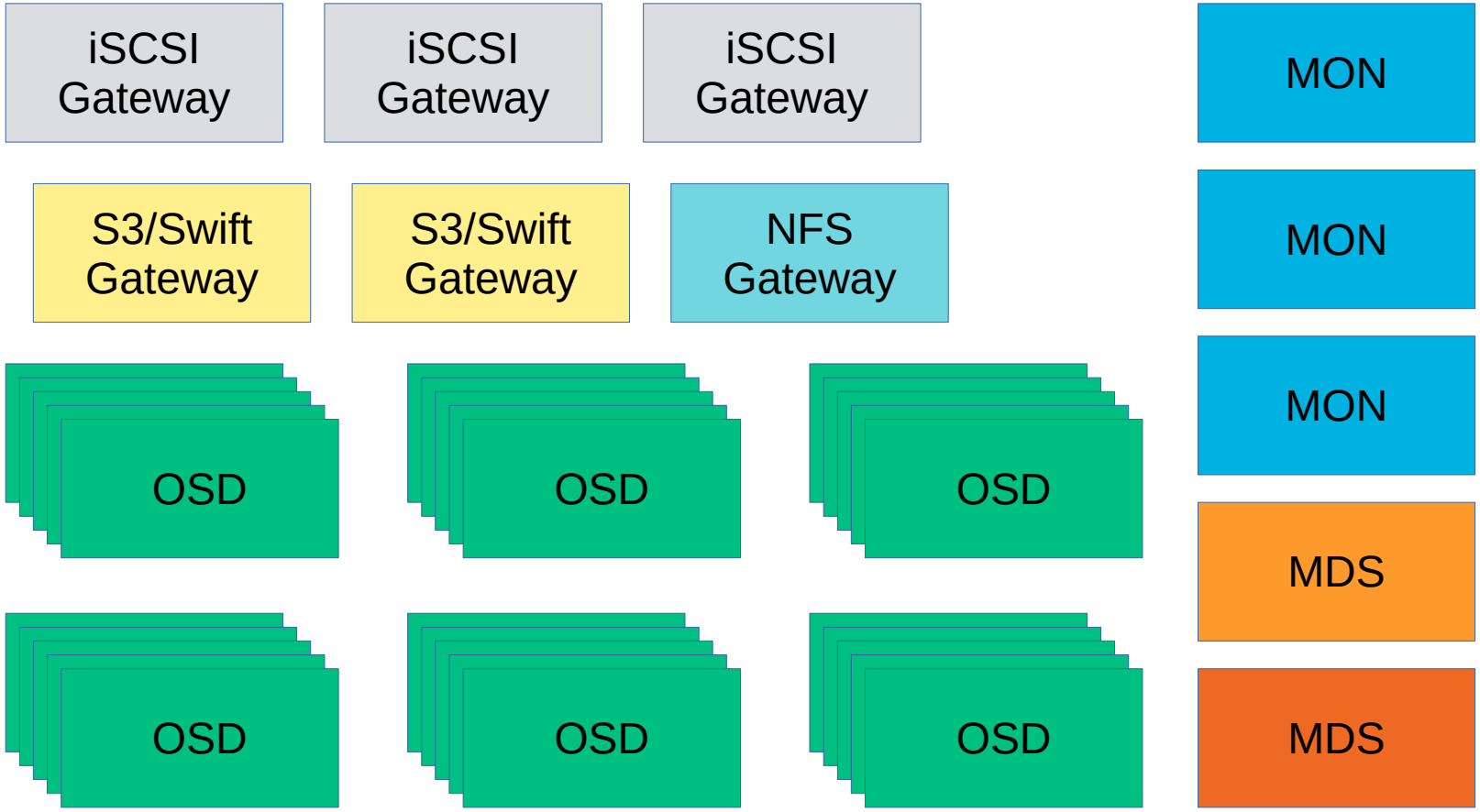
What is Ceph

- An Open Source Software-Defined-Storage project
- Multiple front-ends
  - S3/Swift object interface
  - Native Linux block IO
  - Heterogeneous Block IO (iSCSI)
  - Native Linux network file system (CephFS)
  - Heterogeneous Network File System (nfs-ganesha)
  - Low-level, C++/Python/… libraries
- Common, smart data store (RADOS)
  - Pseudo-random, algorithmic data distribution

Software-Defined-Storage



Ceph Cluster: Logical View



Introducing Dependability

Introducing dependability

- Availability
- Reliability
  - Durability
- Safety
- Maintainability

How much availability do you need?

- Availability and durability are not free
- Cost, Complexity increase exponentially
- Scale makes some things easier

The elephant in the room

- Before we discuss technology ...
- … guess what causes most outages?

Improve your human factor

- Great, you are already here!
- Training
- Documentation
- Back them up with a world-class support organization

High-level considerations

Advantages of Homogeneity

- Eases system administration
- Components are interchangeable
- Lower purchasing costs
- Standardized ordering process

Murphy's Law, 2016 version

- **"At scale, everything fails."**
- Distributed systems protect against individual failures causing service failures by eliminating Single Points of Failure
- Distributed systems are vulnerable to *correlated* failures

Advantages of Heterogeneity

Everything is broken …

… but everything is broken **differently**

Homogeneity is non-sustainable

- Hardware gets replaced
  - Replacement with same model not available, or
  - not desirable given current prices
- Software updates are not (yet) globally immediate
- Requirements change

- Your cluster ends up being heterogeneous **anyway**
- **… you might as well benefit from it.**

Failure is inevitable; suffering is optional

- If you want uptime, prepare for downtime
- Architect your system to sustain a single or multiple failures
- Test whether the system meets your SLA while
  - degraded and
  - recovering.

A bag of suggestions

Embrace diversity

- Automatic recovery requires a >50% majority
  - Splitting into 3 different categories/models?
  - Hard to do with certain components
    - … but possible for the MONs!
  - Multiple architectures?
  - Mix them across different racks/pods

- A 50:50 split still allows manual recovery in case of catastrophic failures
  - Different UPS and power circuits

Storage diversity

- Avoid desktop HDDs
- Avoid sequential serial numbers
- Mount at different angles if paranoid

- Avoid desktop SSDs
- Monitor wear-leveling
- Remember the journals see **all** writes

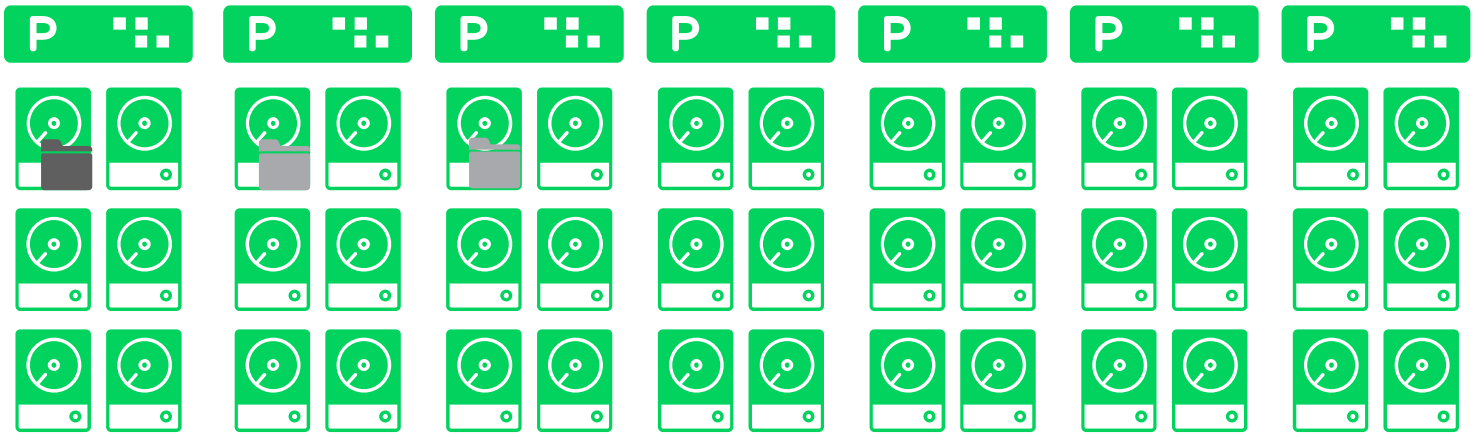
Storage Node Sizing

- Node failures most common granularity
  - Admin mistake, network, kernel crash
- Consider impact on:
  - Performance (degraded and recovery)
  - and capacity!
- Nodes should not be more than 10% of your total capacity
- Free capacity should be larger than largest node

Data availability and durability

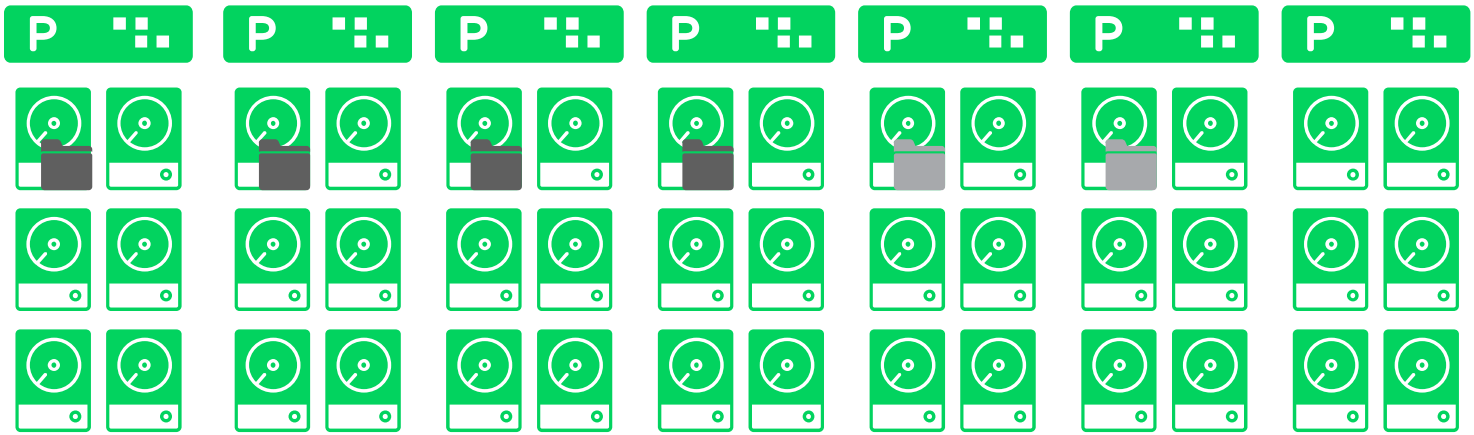
- Replication:
  - Number of copies
  - Linear overhead
- Erasure Coding:
  - Free choice of number of data and coding chunks
  - Can accommodate any number of outages
  - Fractional overhead

Durability: Three-way Replication



Usable capacity: 33%

Durability: 4+2 Erasure Coding



Usable capacity: **66%**

Automatic fault detection and recovery

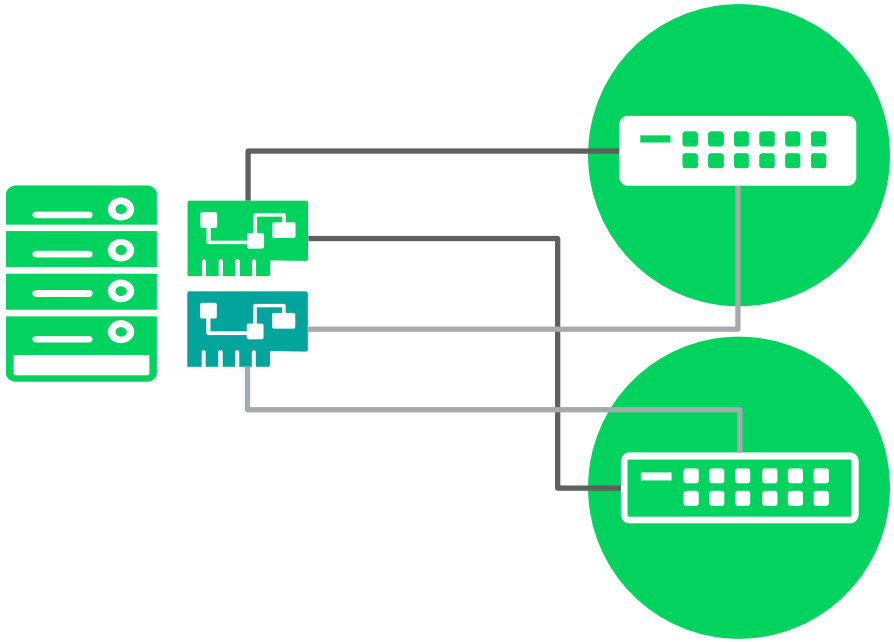
- Do you want this in your cluster?
- Smaller environments may consider running with “noout”

Consider Cache Tiering

- Data in cache tier is replicated
- Backing tier may be slower, but more durable

Network considerations

- Have both the public and cluster network bonded
- Consider different NICs
  - Use last year’s NICs
- One channel from each network to each switch



Avoid configuration drift

- Ensure that systems are configured consistently
  - Installed packages
  - Package versions
  - Configuration (NTP, logging, passwords, …)

- **Avoid manual configuration**
- Use Salt instead

  http://ourobengr.com/2016/11/hello-salty-goodness/

  https://www.suse.com/communities/blog/managing-configuration-drift-salt-snapper/

Update, always (but with care)

- Updates are good for your system
  - Security
  - Performance
  - Stability
- Ceph remains available even while updates are being rolled out

- SUSE’s tested maintenance updates are the main product value

Trust nobody (not even SUSE)

- If you at all possibly can, have a staging system
  - Ideally: a (reduced) version of your production environment
  - At least: a virtualized environment

- Test updates before rolling them out in production
  - Not just code, but also processes!

Trust but verify a.k.a. monitoring

- Performance as the system ages
- SSD degradation / wear leveling
- Capacity utilization
- “Free” capacity is usable for recovery

Disaster ~~can~~ will strike

- Does it matter?
- If it does:
  - Backups.
  - Replicate to other sites.
  - Have fire drills.

Feature selection guideline

- Be aggressive in what you test
- Be conservative in what you deploy


SUSE
We adapt. You succeed.