Container Football

using MARS on Enterprise-Critical Data

FrOSCon 2018 Presentation (v2 for LCA2019 Discussions)

by Thomas Schöbel-Theuer

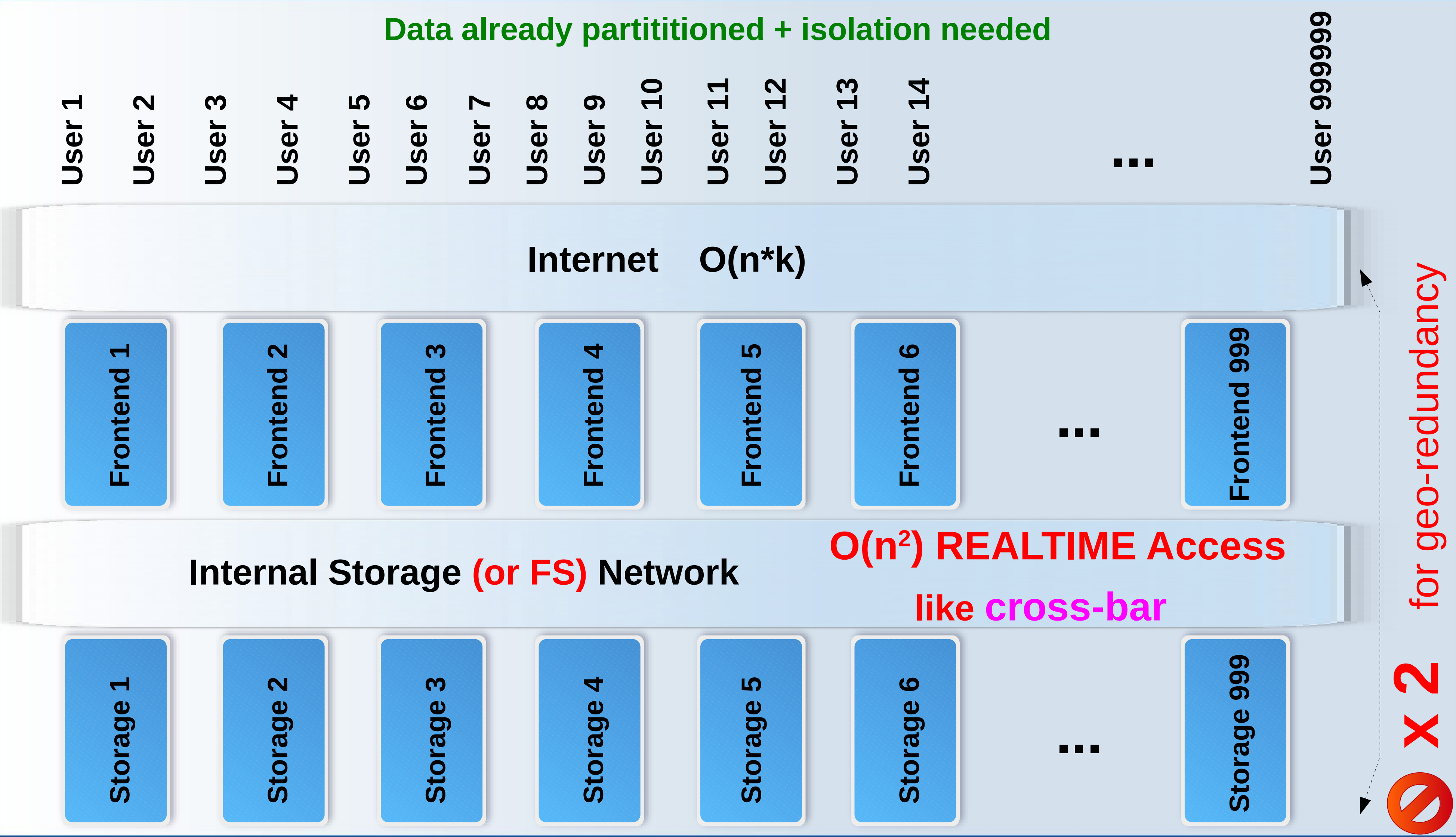
Container Football: Agenda

New method for load balancing

- **Motivation: Scalability of Storage Architectures**
- **Motivation: Reliability** **Unexpected properties!**
- **HOWTO Container Football = Background Migration of LVs**

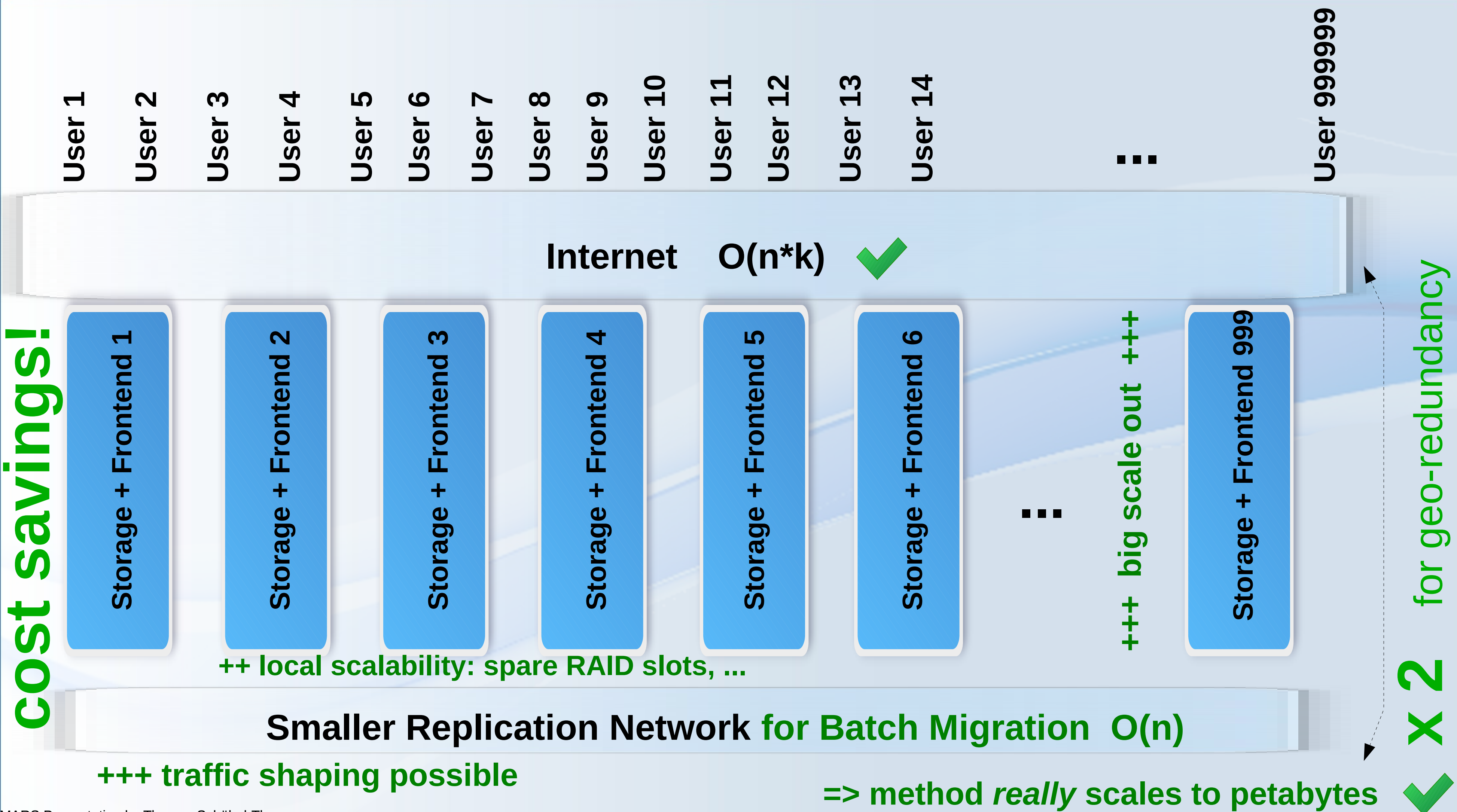
  e.g. for load balancing, HW lifecycle, etc
- **The Football Automation Project**
- **Current Status / Future Plans**

Badly Scaling Architecture: Big Cluster



Well-Scaling Architecture: Sharding

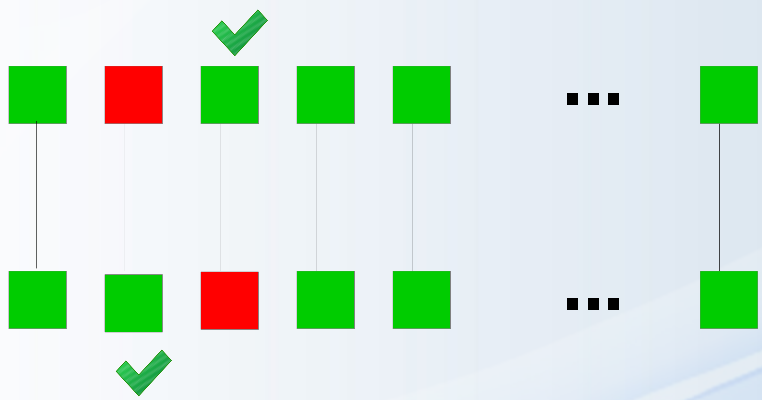
1&1



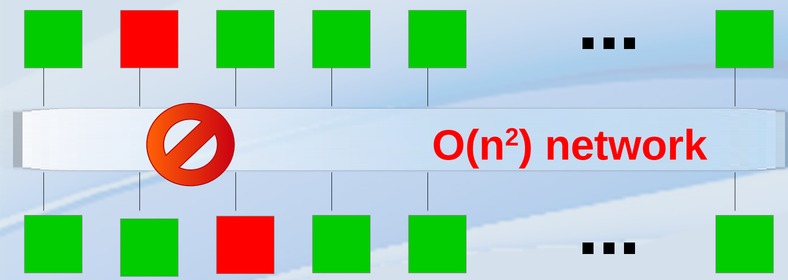
Reliability of Architectures: NODE failures

2 Node failure => ALL their disks are unreachable

DRBD or MARS simple pairs	Big Storage Cluster e.g. Ceph, Swift, ...



same n



=> no customer-visible incident

k=2 replicas not enough
=> INCIDENT because objects are randomly distributed across whole cluster

Low probability for hitting the *same* pair,
even then: only 1 shard affected
=> low **total** downtime

Higher probability for hitting ***any*** 2 nodes,
then O(n) clients affected
=> much higher total downtime

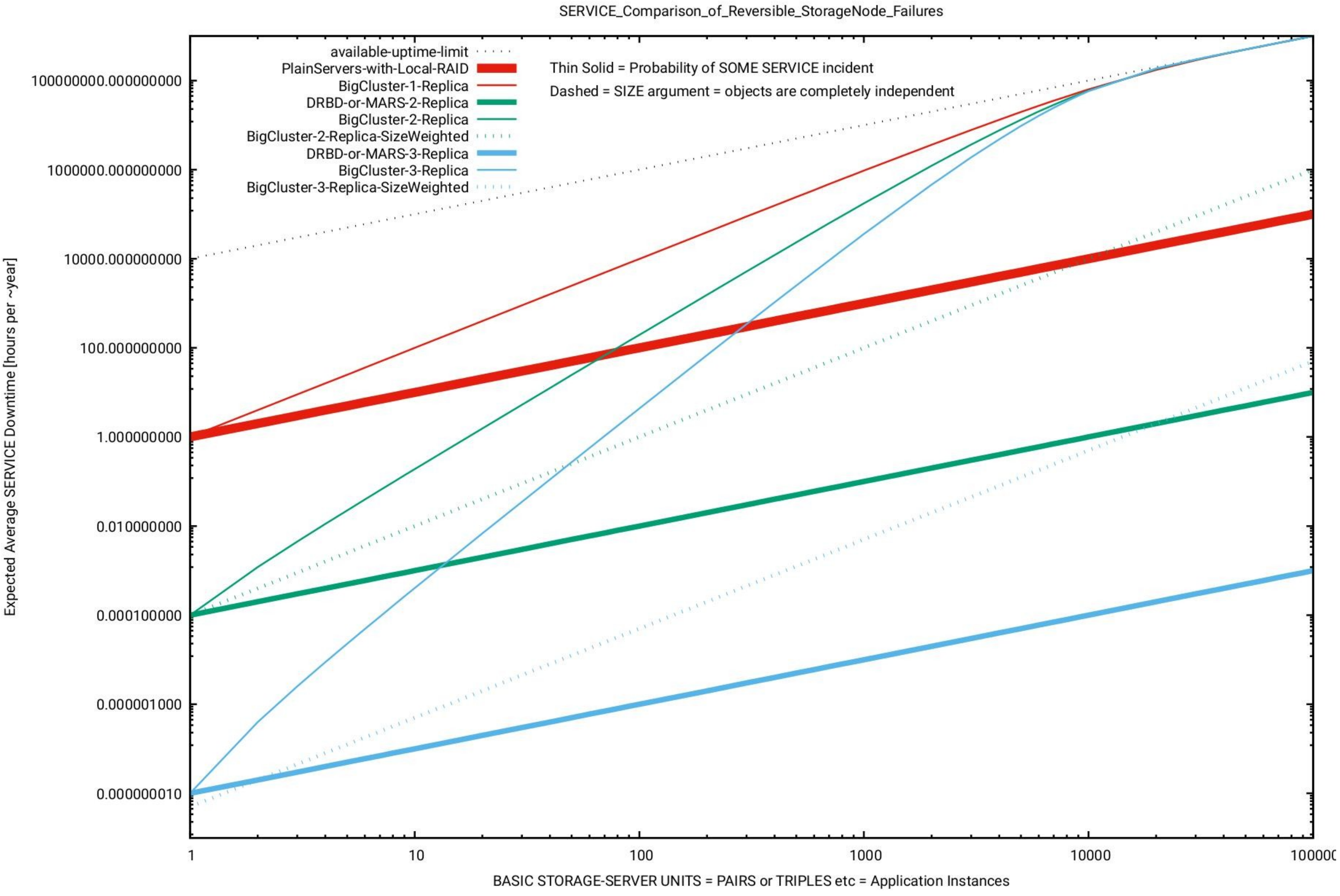
need k >= 3 replicas here

Architectural Reliability Example

- Same / **comparable** dimensioning for BigCluster vs Sharding
- Simplified assumptions (more details in mars-manual.pdf):
  - 1 server has 99.99 % uptime => incident probabiliy p = 0.0001

    => 1 hour downtime per 10.000 operation hours ≈ 13 months ≈ 1 year
  - Only temporary failures, no dependencies between servers
  - **x** axis = # application units = #VGs ~ # LVs
  - **k** = number of replicas => price tag
  - Sectors/Objects are **dependent** (e.g. classical filesystems on top of LVs)
    - maybe too conservative, but **NO GAMBLING!** (timely or spatially)

      enterprise-critical data
  - BigCluster random replication:
    - **all** objects uniformly spread to **all** servers
    - „many“ objects per server => otherwise pice tag!
  - Sharding (DRBDorMARS): simple pairs / triples / ...
- 10000 servers => always 1 of 10000 is down in average


SERVICE_Comparison_of_Reversible_StorageNode_Failures
available-uptime-limit
PlainServers-with-Local-RAID
BigCluster-1-Replica
DRBD-or-MARS-2-Replica
BigCluster-2-Replica
BigCluster-2-Replica-SizeWeighted
DRBD-or-MARS-3-Replica
BigCluster-3-Replica
BigCluster-3-Replica-SizeWeighted
Thin Solid = Probability of SOME SERVICE incident
Dashed = SIZE argument = objects are completely independent
Expected Average SERVICE Downtime [hours per ~year]
100000000.000000000
1000000.000000000
10000.000000000
100.000000000
1.000000000
0.010000000
0.000100000
0.000001000
0.000000010
1
10
100
1000
10000
100000
BASIC STORAGE-SERVER UNITS = PAIRS or TRIPLES etc = Application Instances

Fundamental Law (1)

- Look at 1 LV (from many), then
- Sharding with pairs / triples / etc has the

  BEST POSSIBLE RELIABILITY.

- mathematical proof sketch (induction) at `mars-manual.pdf`

  motivated by practical experiences with 1&1 Ceph clusters
- BigCluster random replication (same k) is **never better.**
  - even worse when considering storage network outages, frontend node failures, **permanent failures** / disasters, etc.
- don‘t neglect k, the **price tag!**

  Integrity of enterprise-critical data
- **don‘t burn arbitrary holes into LVs at (fatal) incidents!**
- Result is contrary to some common belief

Fundamental Law (2)

- **Workarounds for BigCluster:**
  - **USENIX paper on copysets (2013)**
  - **Buckets, spread factors, partitioning, etc**
    - **idea: spread objects to only O(k) instead of to O(n) storage nodes**
      but worse than Sharding on same k
  - **Erasure encoding? => similar to RAID-60 etc, but on O(n) Bigcluster???**
- **Better formulation of general law:**
  - **Distribute your LV data to as less nodes as possible!**
    Spreading to more than necessary worsens reliability
    known as RAID-0 problem
  - **Replicate k into separate failure domains / over long distances**
- **Smallest BigCluster spread: result is then similar to sharding, likely needs similar load balancing / data migration over time**

Common Belief

1&1

- Sharding is inflexible / no load balancing possible???
  - therefore storage networks are a „must“???
- Yes, maybe in the past
- NO LONGER in future => see new Football method
  - VM Football / Container Football / LV Football / ...

Common belief changes only slowly
But fundamental laws of physics / mathematics
are stronger

HOWTO Container Football = Background Migration of LVs

HOST A (old) VM is running	→	HOST B (new) has spare space
– lvdisplay /dev/vg/$mydata		– start kick
–		– lvcreate -L $size -n $mydata vg
–		– marsadm join-resource $mydata \ /dev/vg/$mydata
– (meanwhile VM is altering data)		– marsadm view: wait for UpToDate
– $vmmanager stop /dev/mars/$mydata		– done kick
– cleanup kick		– marsadm primary $mydata
–		– $vmmanger start /dev/mars/$mydata
– marsadm leave-resource $mydata		–
– lvremove /dev/vg/$mydata		–

=> also works with 2 old replicas → 2 new replicas

Example: football.sh in github.com/schoebel/football

Highlevel: Planner vs Optimizer

Dependencies

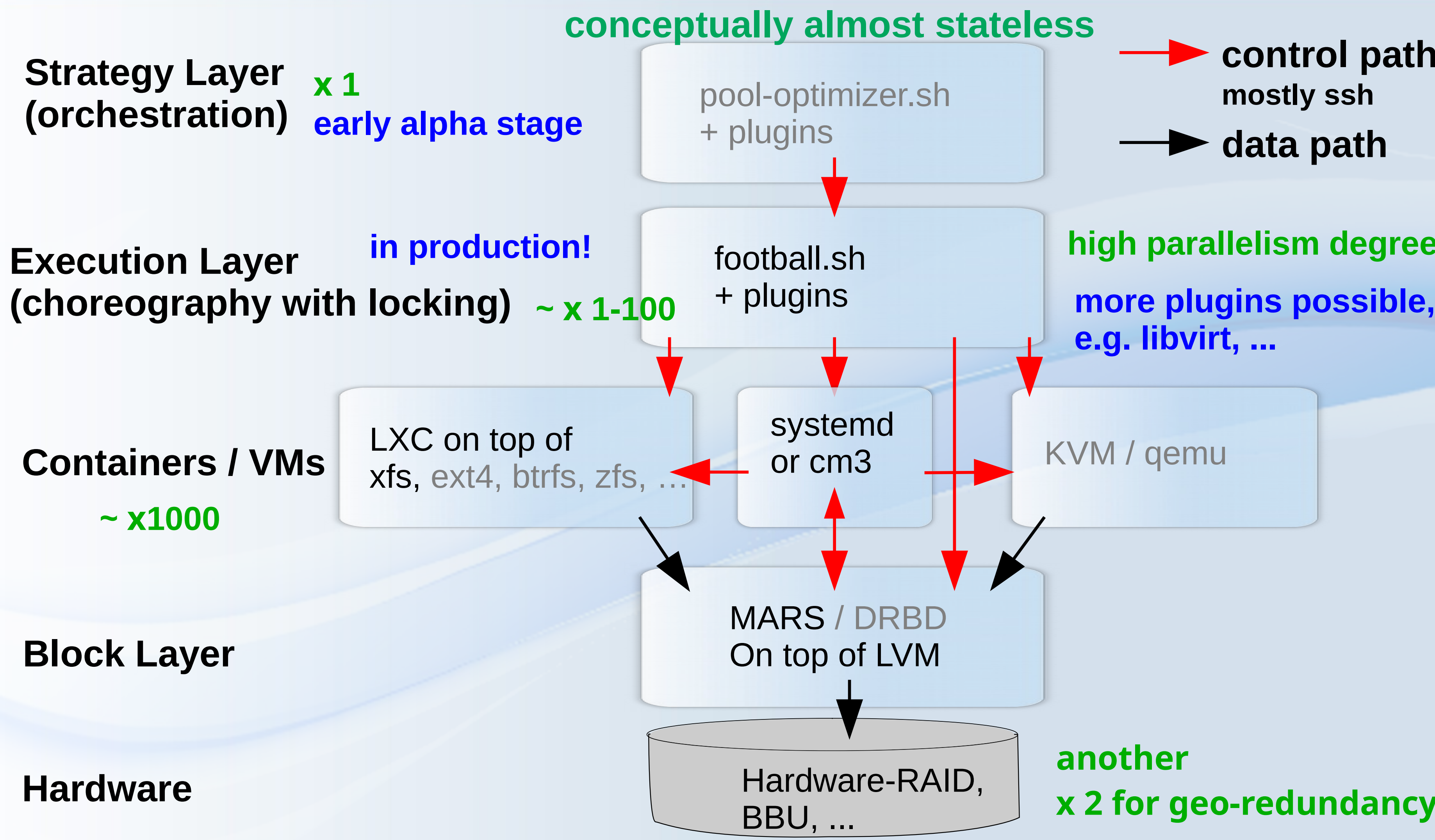
- **Planner: produces stateful plan**

  Complexity = O(|state|^2)

NO dependencies

- **Optimizer: produces stateless actions**
  - **works like a CONTROLLER LOOP**
  - **similar to Kubernetes**

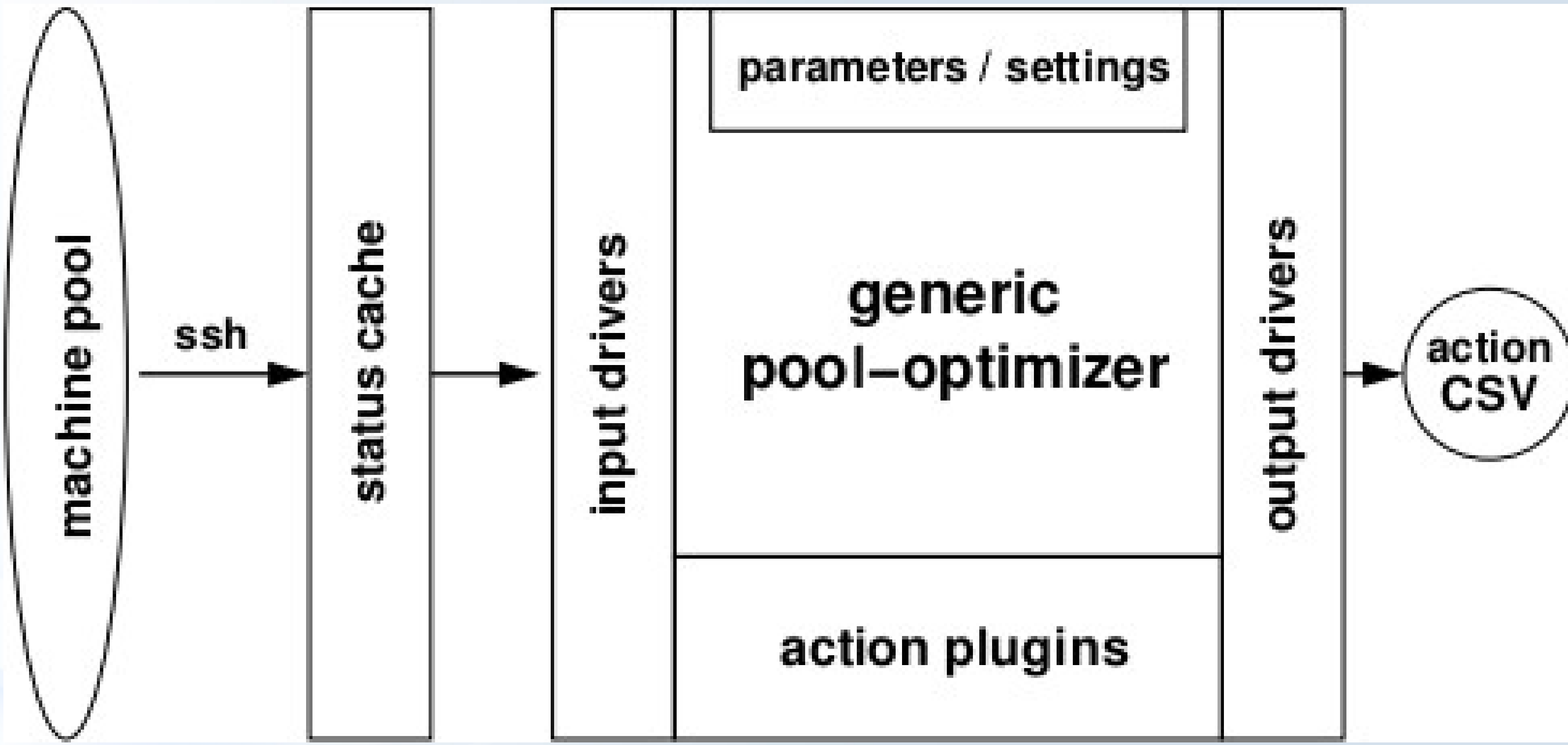
Football Architecture (grey = not yet implemented)



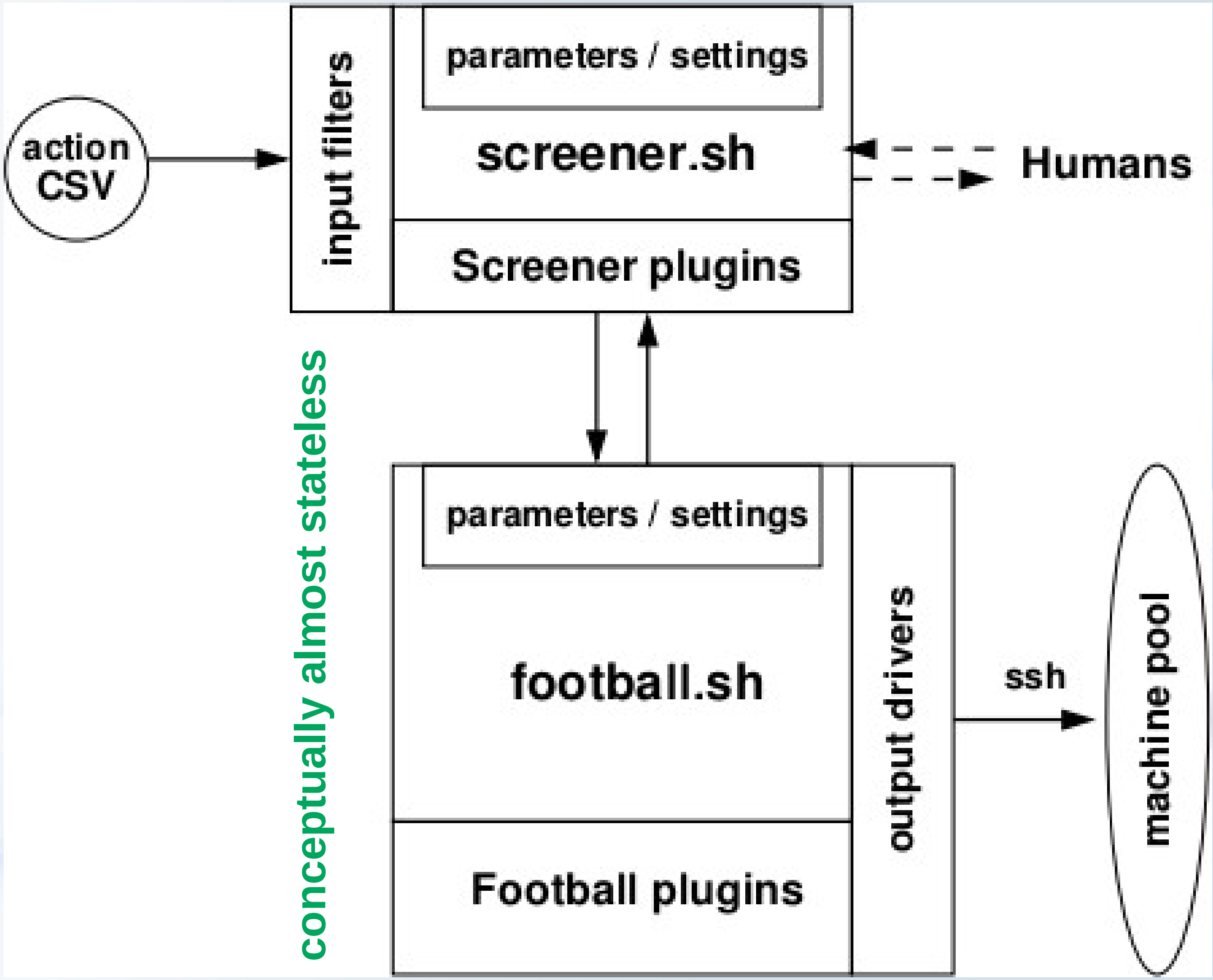
Pool-optimizer (early alpha stage)

conceptually almost stateless

State is ONLY here



football.sh (in production with cm3 plugin)



Football Current Status

- GPL with lots of plugins, some generic, some 1&1-specific
  - about 2/3 of code is generic
  - plugins/football-basic.sh uses `systemd`
  - https://github.com/schoebel/football
  - https://github.com/schoebel/mars
- Multiple operations:
  - migrate $vm $target_cluster
    - low downtime (seconds to few minutes)
  - shrink $vm $target_percent
    - uses local incremental rsync, more downtime
  - expand $vm $target_percent
    - online, no downtime
- In production at internal Efficiency project
  - get rid of old hardware
  - Concentrate ~ 7 LXC containers on 1 hypervisor
  - currently >50 „kicks“ per week
    - limited by hardware deployment speed
    - Proprietary Planner (for HW lifecycle)
  - Almost finished: ~70% of ~1800 blades already migrated (mid of January 2019) and mostly shrunk

Sponsoring (MARS + Football)

- Best for > 1 PiB of enterprise-critical data
  - Example: ShaHoLin (slide3)
  - More plugins in future, e.g. for KVM, ...
- Future pool-optimizer will deliver similar functionality than **Kubernetes**
  - but on stateful storage + containers instead of stateless Docker containers
  - State is in the storage and in the machines, but not in orchestration
- Long-term perspective
  - MARS is largely complementary to DRBD
  - Geo-redundancy with OpenSource components
  - distances > 50km possible, tolerates flaky replication networks
  - **Price / performance** better than anything else (see mars-manual.pdf)
  - Architectural **reliability** better than BigCluster with cheaper hw + network!
- ask me: decades of experience with enterprise-critical data and long-distance replication

Appendix

MARS Current Status

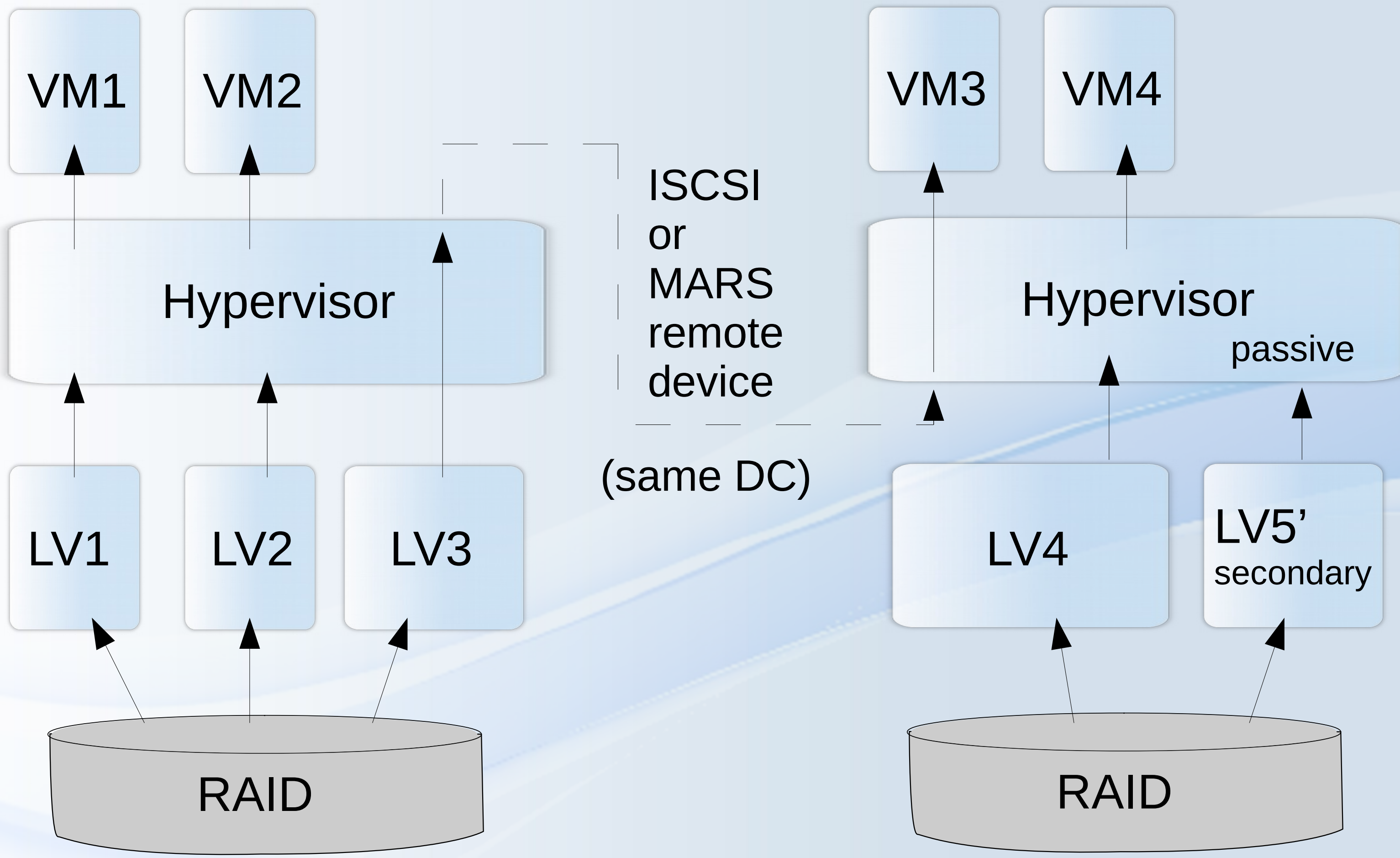
- MARS source under GPL + docs:
  `github.com/schoebel/mars`
  `mars-manual.pdf` ~ 100 pages
- mars0.1stable productive since 02/2014
- Backbone of the 1&1 geo-redundancy feature
- MARS status January 2018:
  > 5800 servers (shared hosting + databases)

  > 2x12 petabyte total

  ~ 10 billions of inodes in > 2500 xfs instances, biggest ~ 40 TB

  <= 10 LXC Containers on 1 Hypervisor
- New internal Efficiency project
  - Concentrate more LXC containers on 1 hypervisor
  - New public branch mars0.1b with many new features, e.g. mass-scale clustering, socket bundling, remote device, etc
  - mars0.1b currently in ALPHA stage

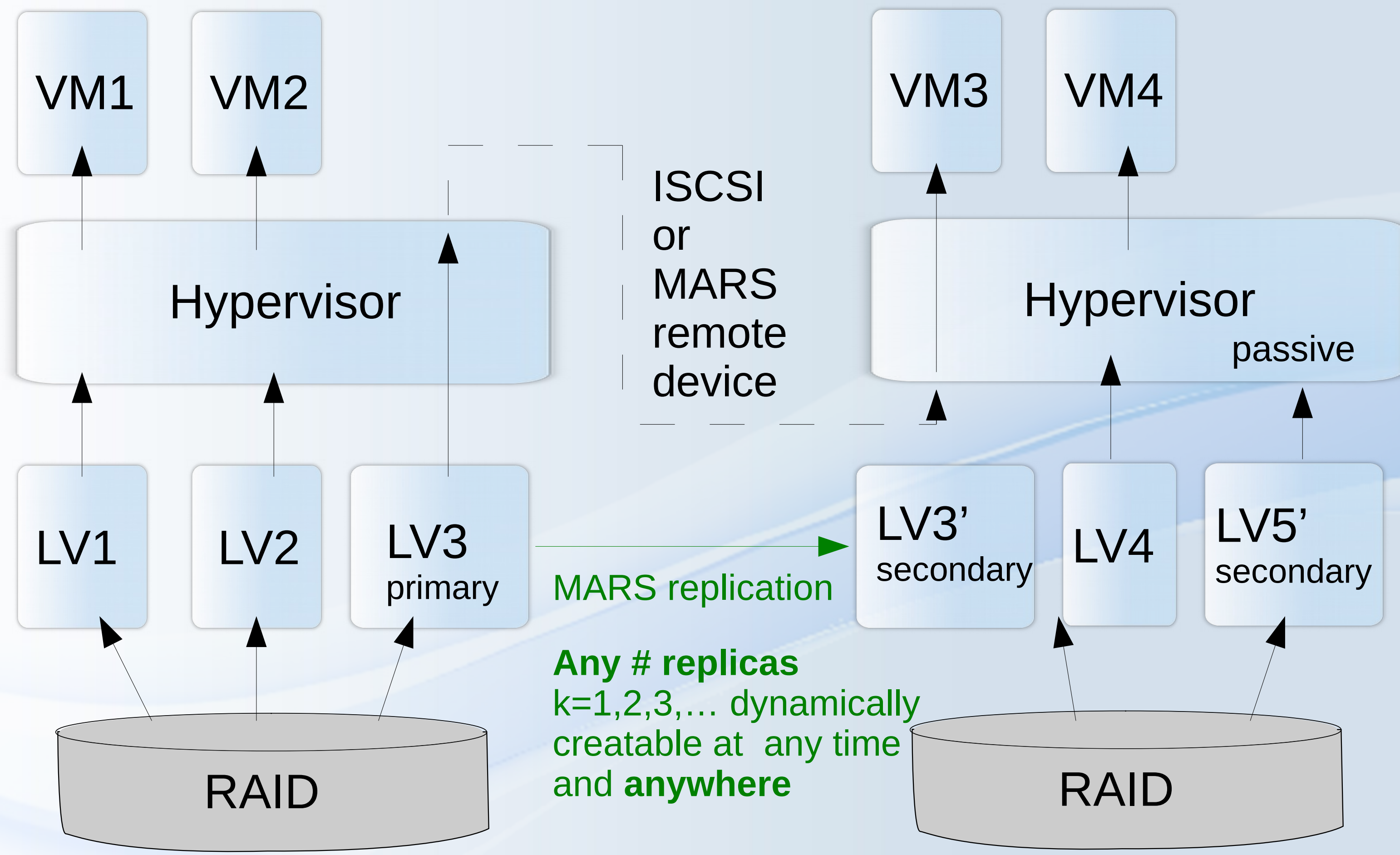
Flexible MARS Sharding + Cluster-on-Demand



any hypervisor works in client and/or server role

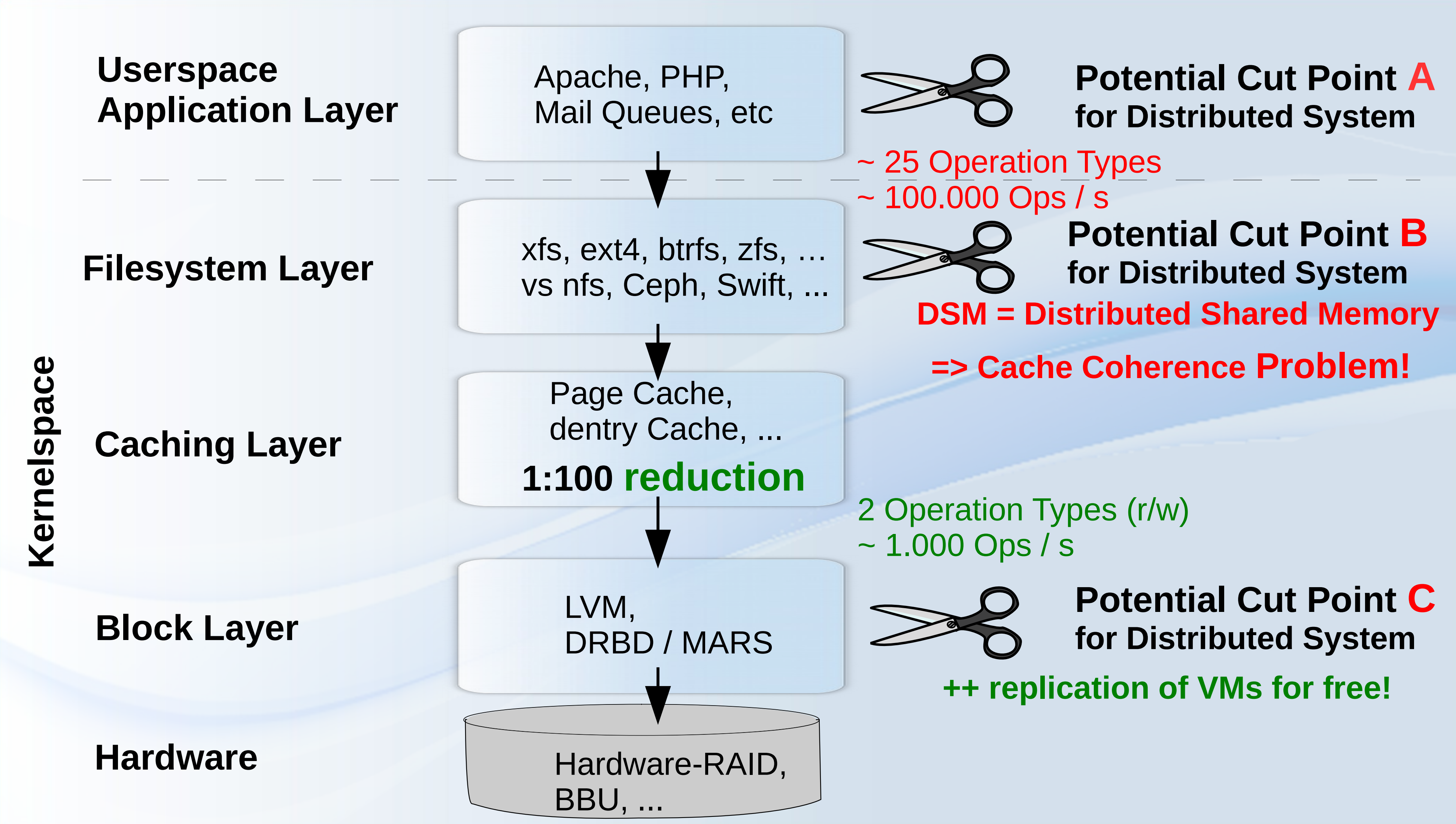
and preferably **locally** at the same time

Flexible MARS Background Migration



=> any hypervisor may be source or destination of some LV replicas at the same time

Replication at Block Level vs FS Level



Use Cases DRBD+proxy vs MARS Light

DRBD+proxy
(proprietary)

Application area:

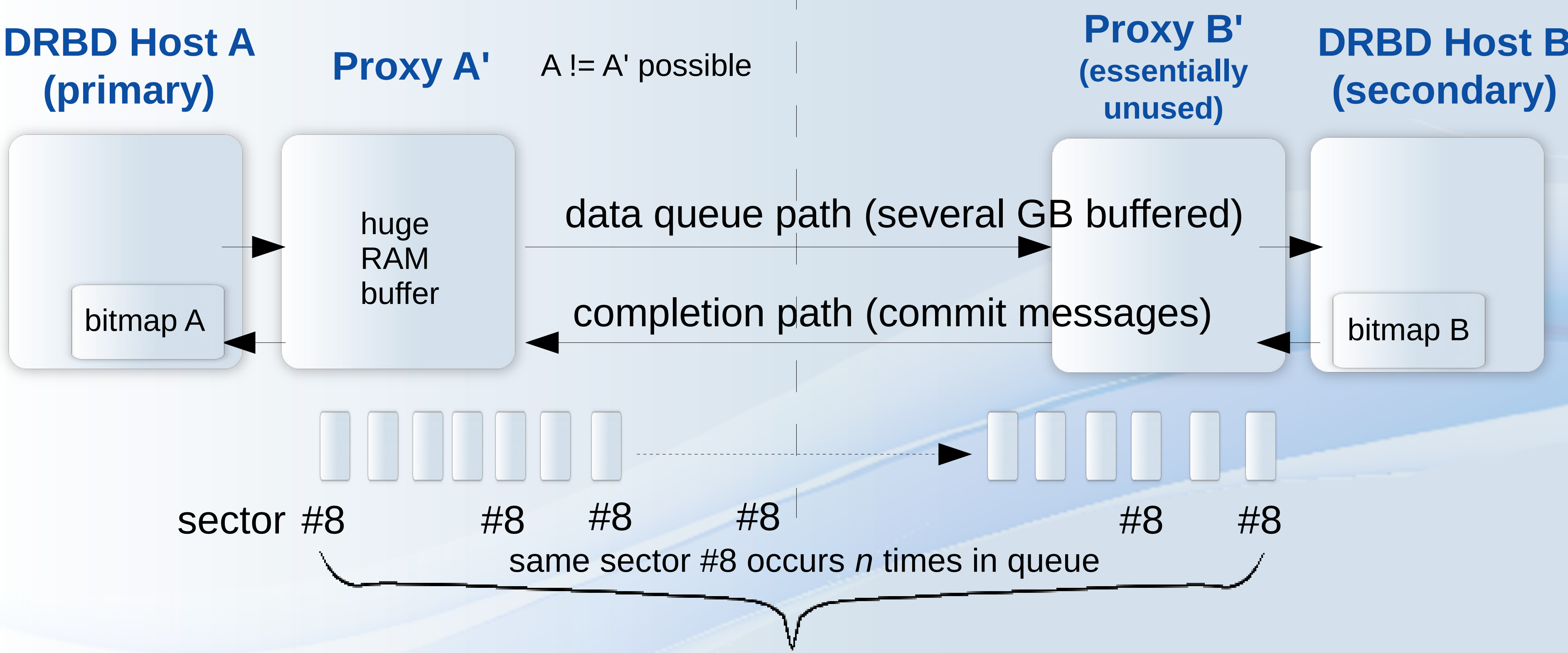
- Distances: any
- Aynchronously
  - **Buffering in RAM**
- Unreliable network leads to **frequent re-syncs**
  - RAM buffer gets lost
  - at cost of actuality
- **Long** inconsistencies during re-sync
- Under pressure: **permanent** inconsistency possible
- High memory overhead
- Difficult scaling to k>2 nodes

MARS Light
(GPL)

Application area:

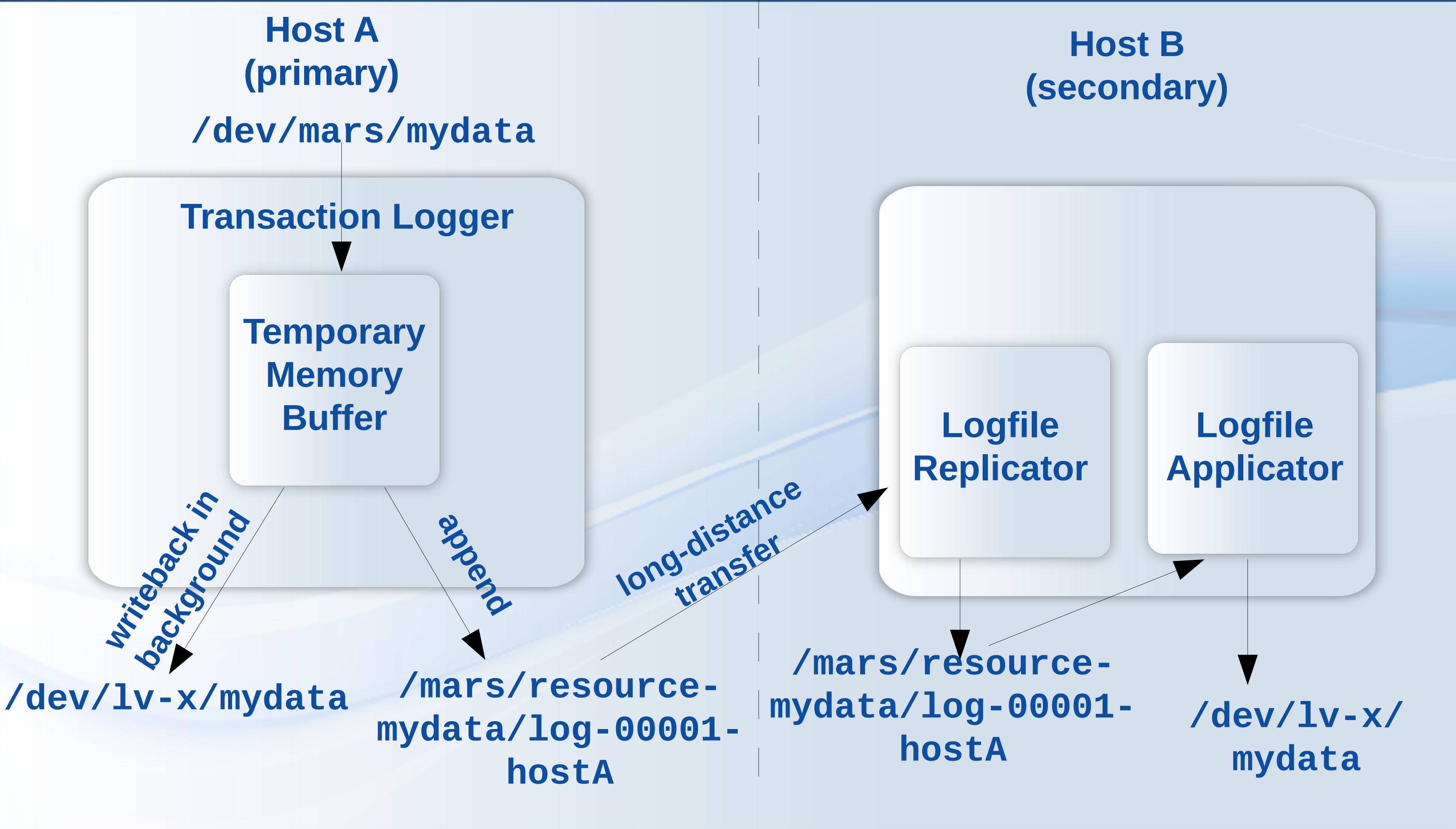
- Distances: **any** (>>50 km)
- Asynchronously
  - near-synchronous modes in preparation
- Tolerates **unreliable network**
- Anytime consistency
  - no re-sync
- Under pressure: no inconsistency
  - possibly at cost of actuality
- Needs >= 100GB in `/mars/` for transaction logfiles
  - dedicated spindle(s) recommended
  - RAID with BBU recommended
- Easy scaling to k>2 nodes

DRBD+proxy Architectural Challenge



n times
=> need *log*(n) bits for counter
=> but DRBD bitmap has only 1 bit/sector
=> workarounds exist, but complicated
(e.g. additional dynamic memory)

MARS Data Flow Principle



Framework Architecture

for MARS + future projects

1&1

External Software, Cluster Managers, etc

Userspace Interface `marsadm`

Framework Application Layer
MARS Light, MARS Full, etc

MARS Light	MARS Full	...

Framework Personalities
XIO = eXtended IO ≈ AIO

XIO bricks	future Strategy bricks	other future Personalities and their bricks

Generic Brick Layer
IOP = Instance Oriented Programming
+ AOP = Aspect Oriented Programming

Generic Bricks

Generic Objects

Generic Aspects
s