Easy Geo-Redundant Failover with MARS and `systemd`

FrOSCon 2019 Presentation by Thomas Schöbel-Theuer

Easy Geo-Redundant Failover: Agenda

- Motivation: why GEO-redundancy
- Long-distance asynchronous replication
- Cluster management for long distances
- Using `systemd` as a clustermanager
- Current Status / Future Plans

Why GEO-Redundancy

DR = Disaster Recovery
CDP = Continuous Data Protection

- **Example: GALILEO incident (DR / CDP did not work)**
  - Disaster = earthquake, flood, terrorist attack, power outage, ...
- **BSI Paper 12/2018:**

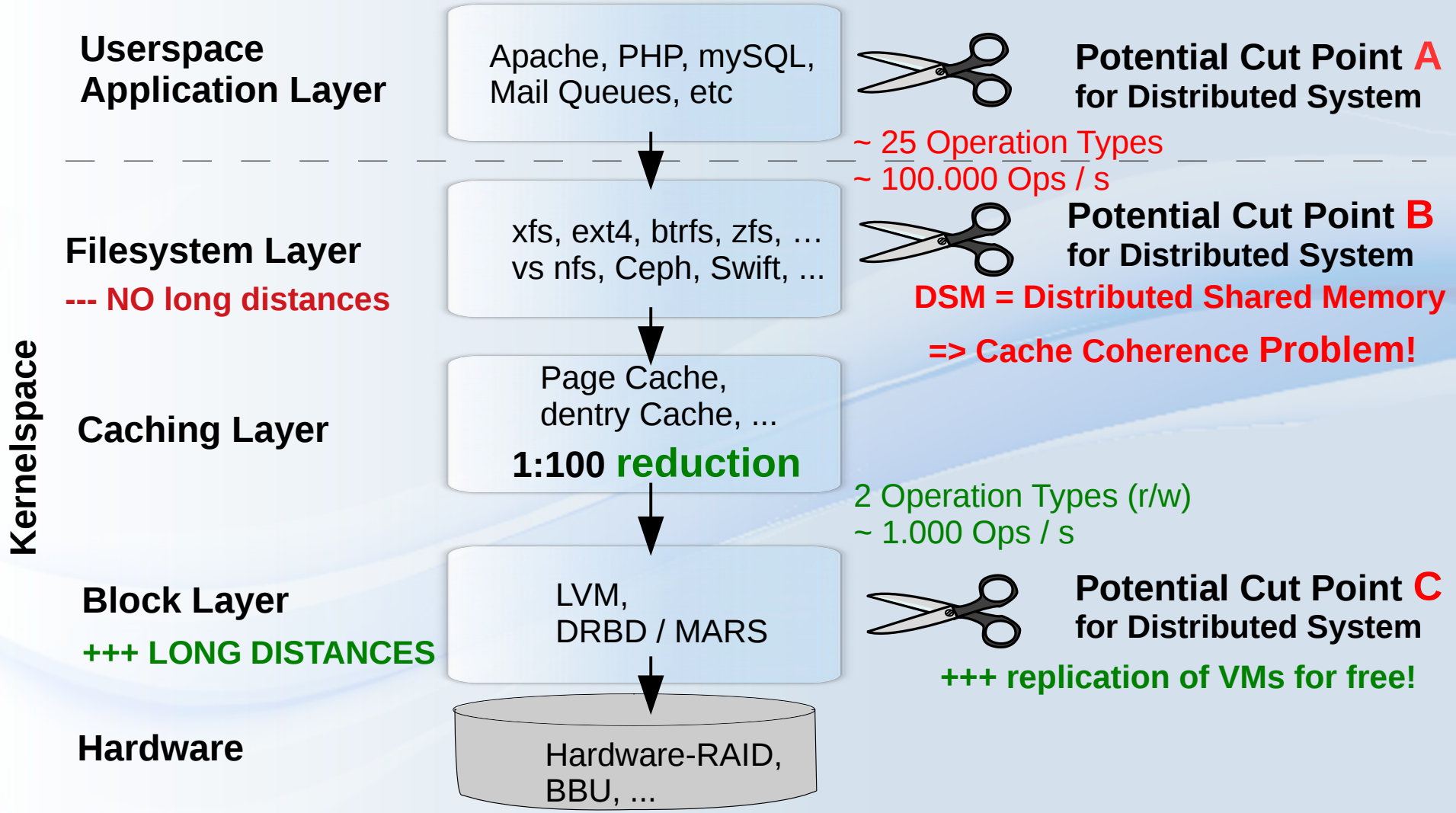
  Kriterien für die Standortwahl höchstverfügbarer und georedundanter Rechenzentren

  `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Sicherheitsberatung/Standort-Kriterien_HV-RZ/Standort-Kriterien_HV-RZ.pdf?__blob=publicationFile&v=5`

  in English: Criteria for Locations of Highly Available and Geo-Redundant Datacenters

  - Stimulated some controversial discussions, but see commentary `https://www.it-finanzmagazin.de/bsi-rechenzentren-entfernung-bafin-84078/`
- **Conclusions: distances > 200 km „recommended“**
- **Might influence future legislation (EU / international)**
- **„Critical Infrastructures“ more important!**

Replication at Block Level vs FS Level



Growth at 1&1 Ionos ShaHoLin = Shared Hosting Linux

- 4 datacenters at 2 continents, pair distance > 50 km
- ~ 9 millions of customer home directories
- ~ 10 billions of inodes
- > 4.7 petabytes *allocated* in ~ 3800 xfs instances

  LVM ~ 8 PB x 2 for geo-redundancy via MARS

  `https://github.com/schoebel/mars`

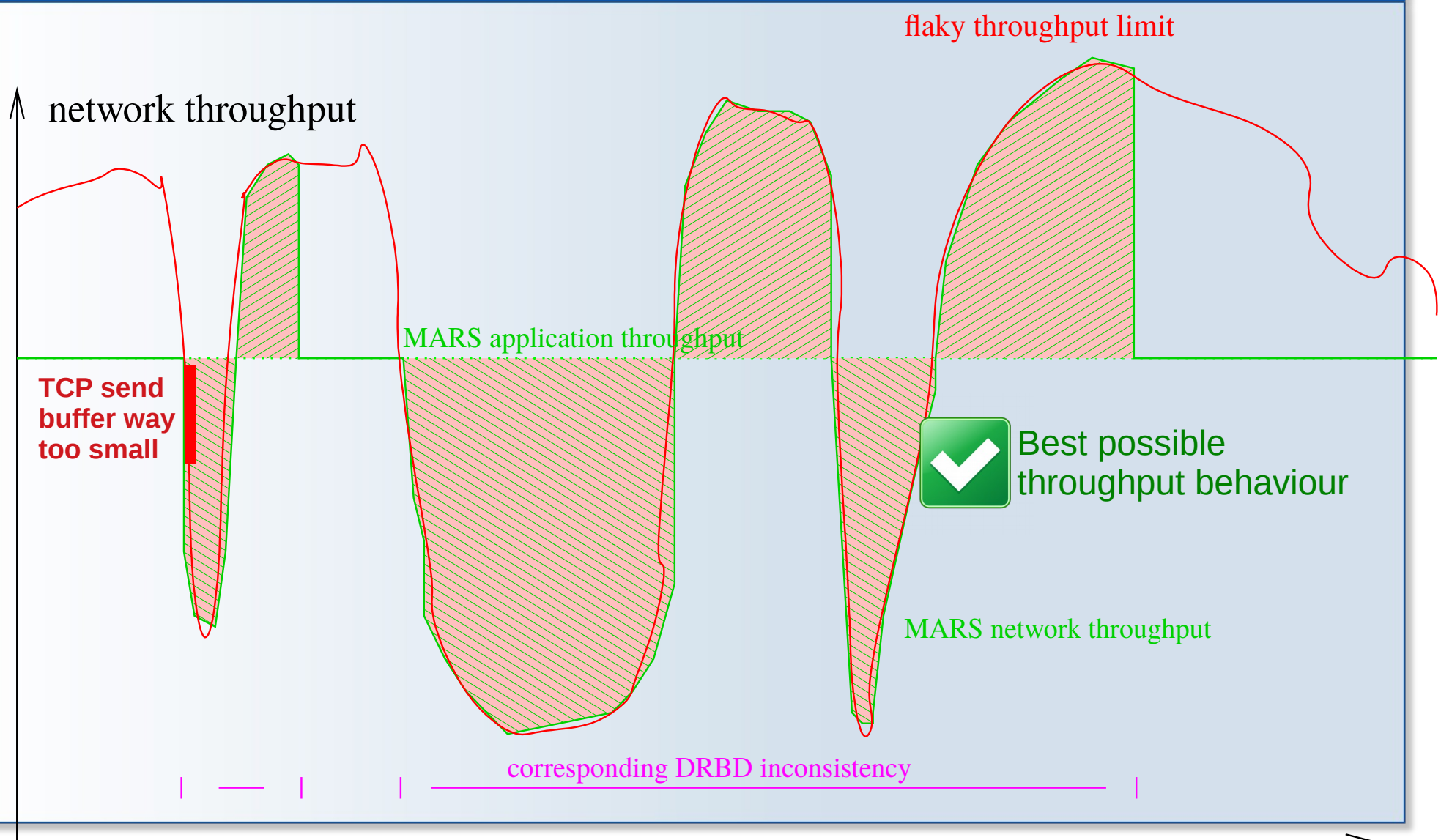
- Growth rate ~ 21 % / year
- Solution: Container Football on top of MARS

  `https://github.com/schoebel/football`

Long-Distance Asynchronous Replication

- **Synchronous** does not *generally* work over ≈50 km
  - like iSCSI over 50 km
- Need Asynchronous Replication
  - Application specific, e.g. mySQL replication ✔
  - Commercial appliances: $$$ €€€ 
  - OpenSource ✔
    - plain DRBD is NOT asynchronous
      - commercial DRBD-Proxy: RAM buffering
    - MARS: truly asynchronous + persistent buffering + transaction logging + MD5 checksums + Anytime Consistency

Network Bottlenecks: MARS



Cluster Management for > 50 km

- Proprietary e.g. 1&1 cm3 (no GPL) 
- Pacemaker & co typically don‘t work as expected
  - original HeartBeat DSM model: shared disk
    cannot *really* handle Split Brain
  - explainable by CAP theorem
    network failures: then no strict consistency + availability at the same time
- Using `systemd` as a Linux clustermanager
  - already in use almost everywhere e.g. startup of VMs
  - itself somewhat „monolithic“, but
  - easily extensible via Unit Files
- MARS dynamic resource creation / deletion
  `marsadm join-resource / leave-resource`
- Solution: marsadm internal macro processor
  creates / deletes systemd units „on the fly“

systemd Unit Example (Template)

```
bash> cat systemd/\^\{unit\}-@\{res\}.mount
@eval{%let{mount}{%subst{%{unit}}{-}{/}}}
[Unit]
Description=MARS local mount on /@{mount}/@{res}
Documentation=https://github.com/schoebel/mars/docu/mars-manual.pdf
Requires=mars.service
After=mars.service

ConditionPathIsSymbolicLink=/mars/resource-@{res}/systemd-want
ConditionPathExists=/mars/resource-@{res}/userspace/systemd-want-@{host}
ConditionPathExists=/dev/mars/@{res}
ConditionPathIsDirectory=/@{mount}/@{res}

[Mount]
What=/dev/mars/@{res}
Where=/@{mount}/@{res}

[Install]
WantedBy=mars.service
```

Usage of systemd unit templates

- **Activation of template** (once after resource creation, for the whole cluster)

```
marsadm create-resource $resource /dev/$vg/$resource
mkfs.xfs /dev/mars/$resource
marsadm set-systemd-unit $resource $start_unit $stop_unit
    => automatic instantiation via macro processor
```

- **Usage at planned handover:**

  `marsadm primary $resource` (or `marsadm primary all`)

  - **Automagically** (independently for each resource):
    - Old primary: `systemctl stop $stop_unit`
    - Old primary: MARS goes to secondary mode
    - New primary: MARS becomes primary /dev/mars/$resource will appear
    - New primary: `systemctl start $start_unit`

- **Usage at unplanned failover:**
  - `marsadm disconnect all ; marsadm primary --force all`

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 Ionos geo-redundancy feature
- MARS status August 2019:

  > 2600 servers (shared hosting)
  biggest ~300 TB

  > 2x12 petabyte total (at RAID level)

  ~ 10 billions of inodes in > 3800 xfs instances,
  biggest ~ 40 TB

  up to 12 LXC Containers on 1 Hypervisor

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` as cluster manager
  - 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 1&1 Ionos
  - get rid of old hardware (project successfully finished)
  - load balancing
  - >50 „kicks“ per week
    - limited by hardware deployment speed
    - Proprietary Planner (for HW lifecycle)

MARS Future Plans

- LTS kernel 4.14 (almost done)
- Faster checksumming (CRC32c | CRC32 | SHA1 | MD5)
- Logfile compression (LZO | LZ4 | ZLIB)
- Optional network transport compression
  - may help for some very slow networks
- Better scalability
  - more resources per host
  - more hosts per cluster
    - get rid of workaround `marsadm join-cluster / split-cluster`
    - eventually: BigCluster at metadata level
- Linux kernel upstream
  - requires a *lot* of work!
- More tooling, integration into other OpenSource projects

Collaboration sought

=> Opportunities for other OpenSource projects!

Sponsoring (MARS + Football)

- Best for > 1 PiB of enterprise-critical data
  - More Football 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

CAP Theorem

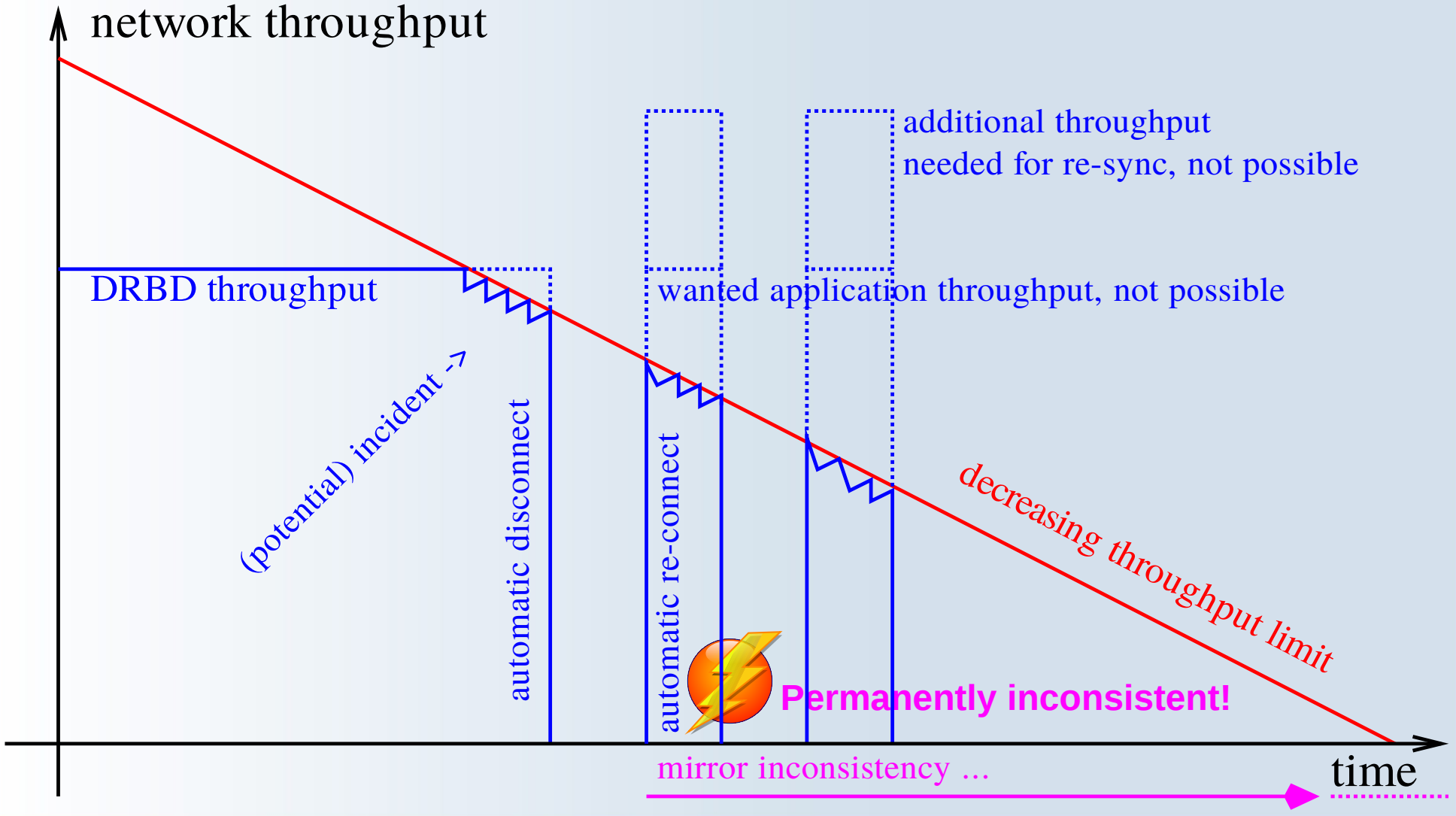
C = Strict Consistency

violated at
- disasters
- LONG distances

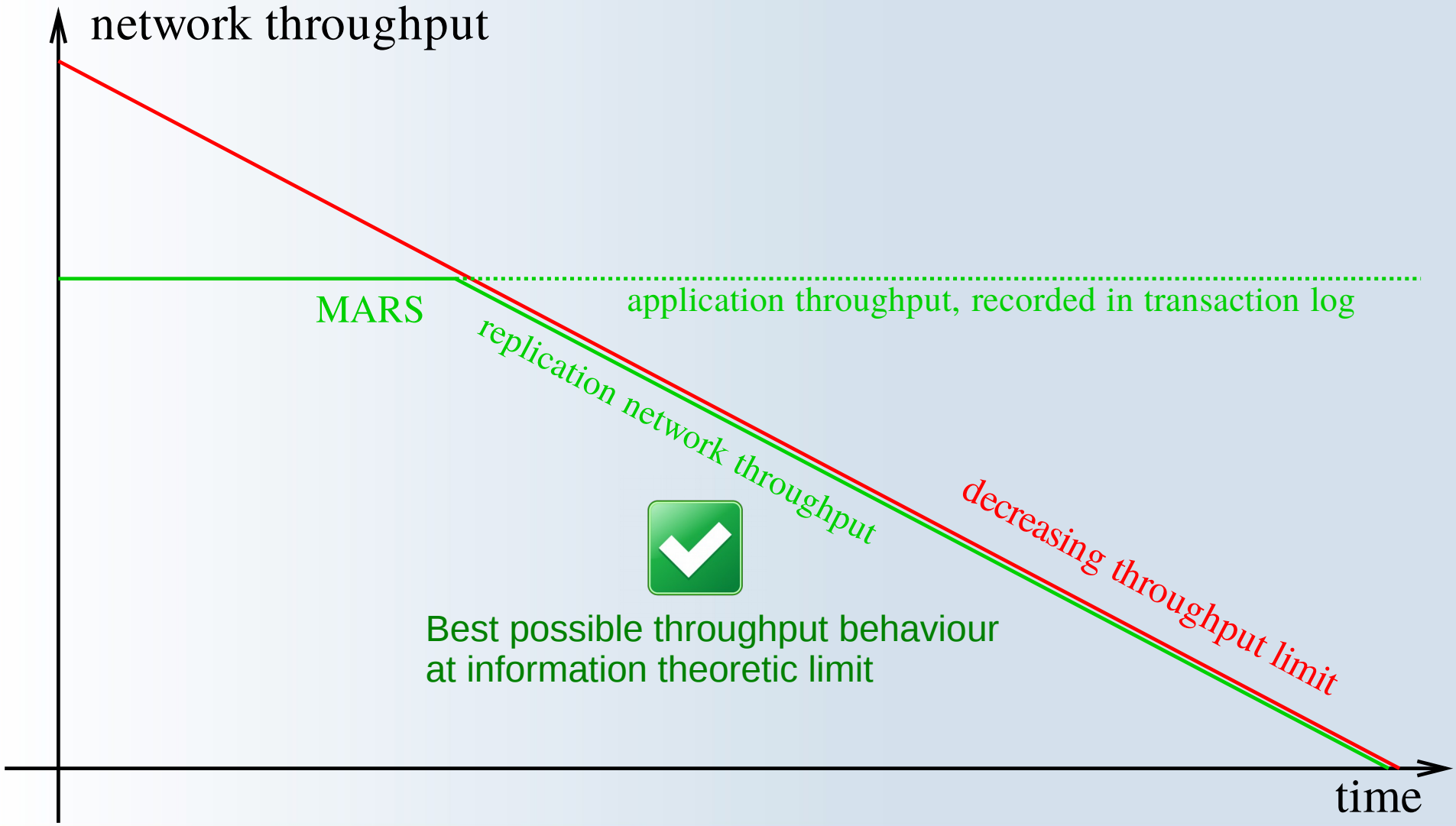
A = Availability

P = Partitioning Tolerance
= the network can have its own outages

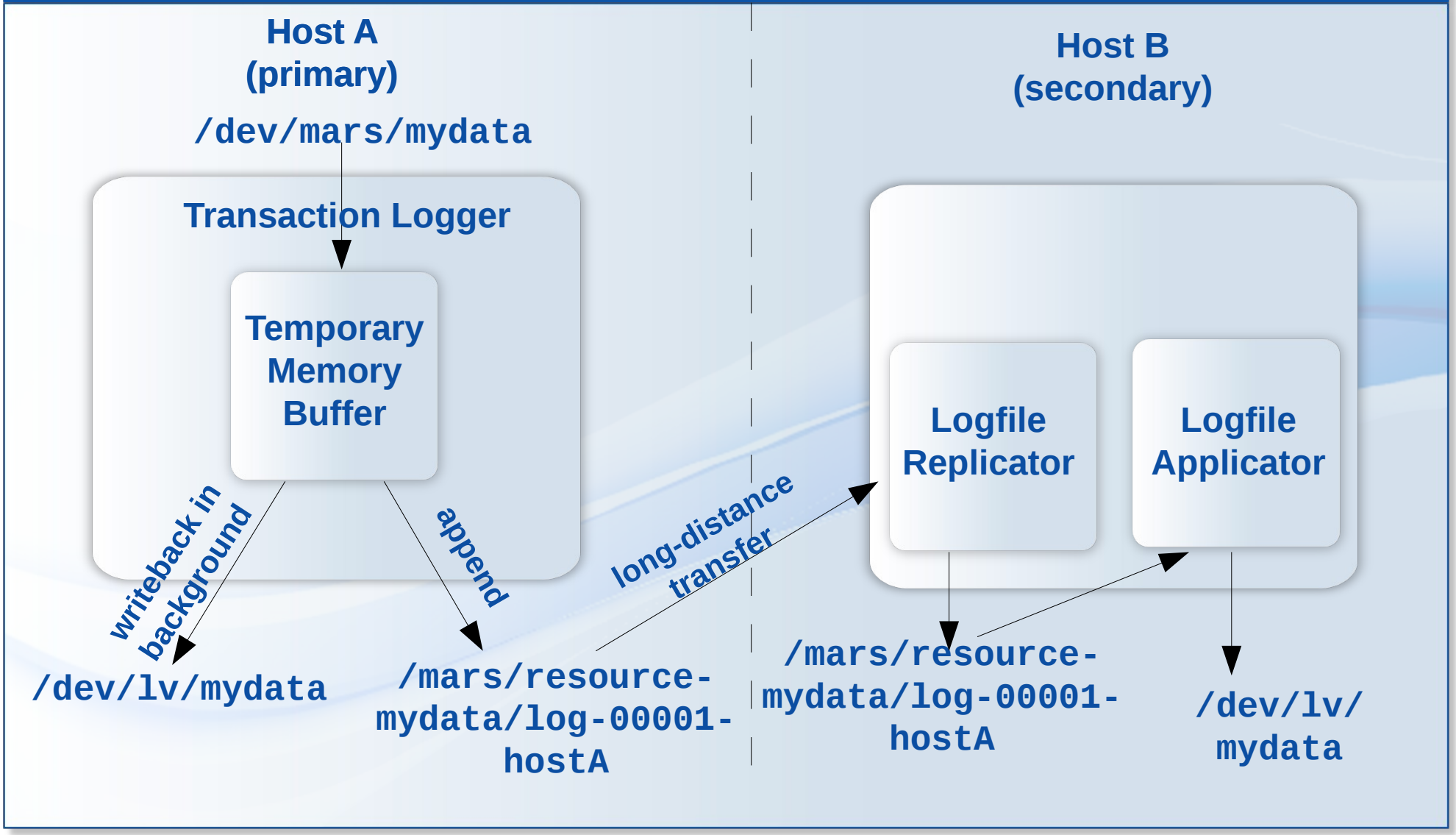
Network Bottlenecks (1) DRBD



Network Bottlenecks (2) MARS



MARS Data Flow Principle



Use Cases DRBD+proxy vs MARS

DRBD+proxy
(proprietary)

Application area:

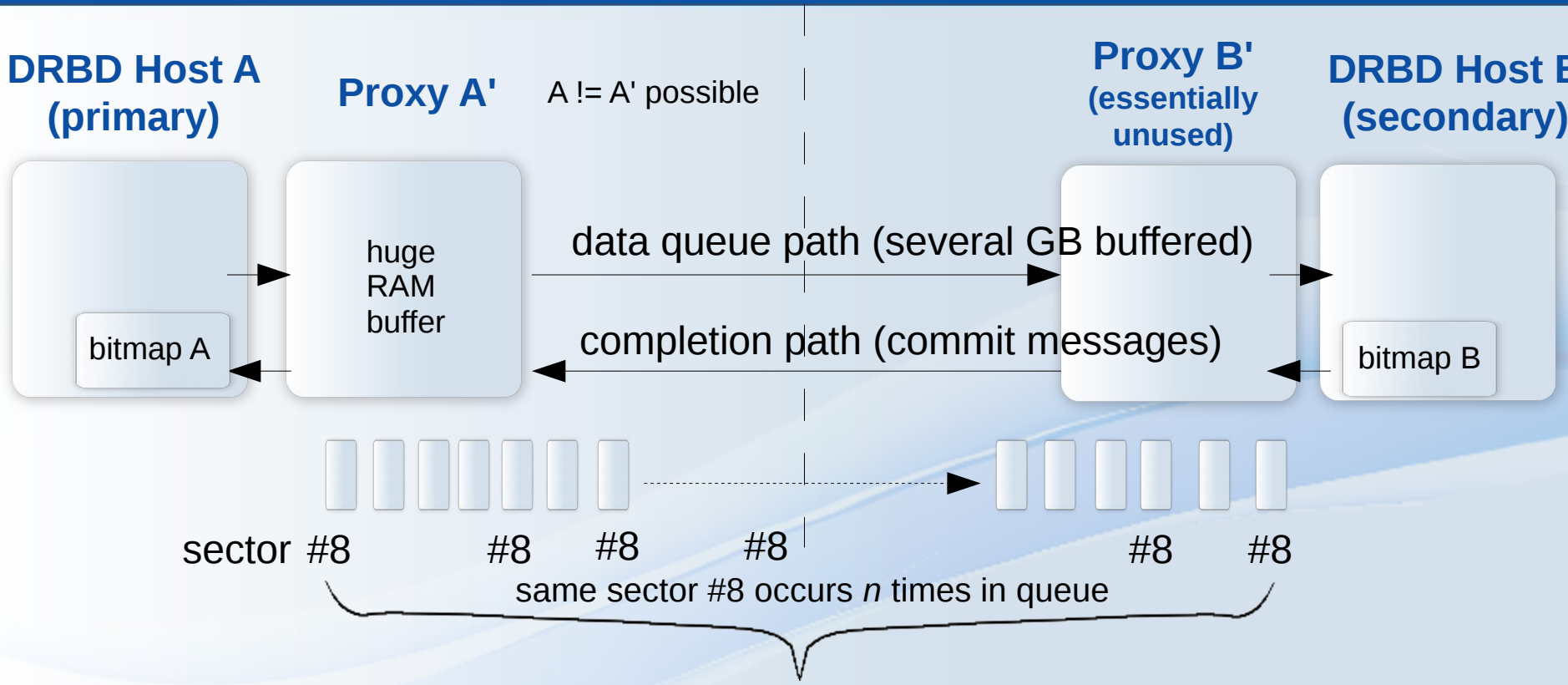
- Distances: any
- Aynchronously
  - **Buffering in RAM**
- Unreliable network leads to **frequent re-syncs**
  - RAM buffer gets lost
  - at cost of actuality
- **Long** inconsistencies during re-sync
- Under pressure: **permanent** inconsistency possible
- High memory overhead
- Difficult scaling to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

MARS
(GPL)

Application area:

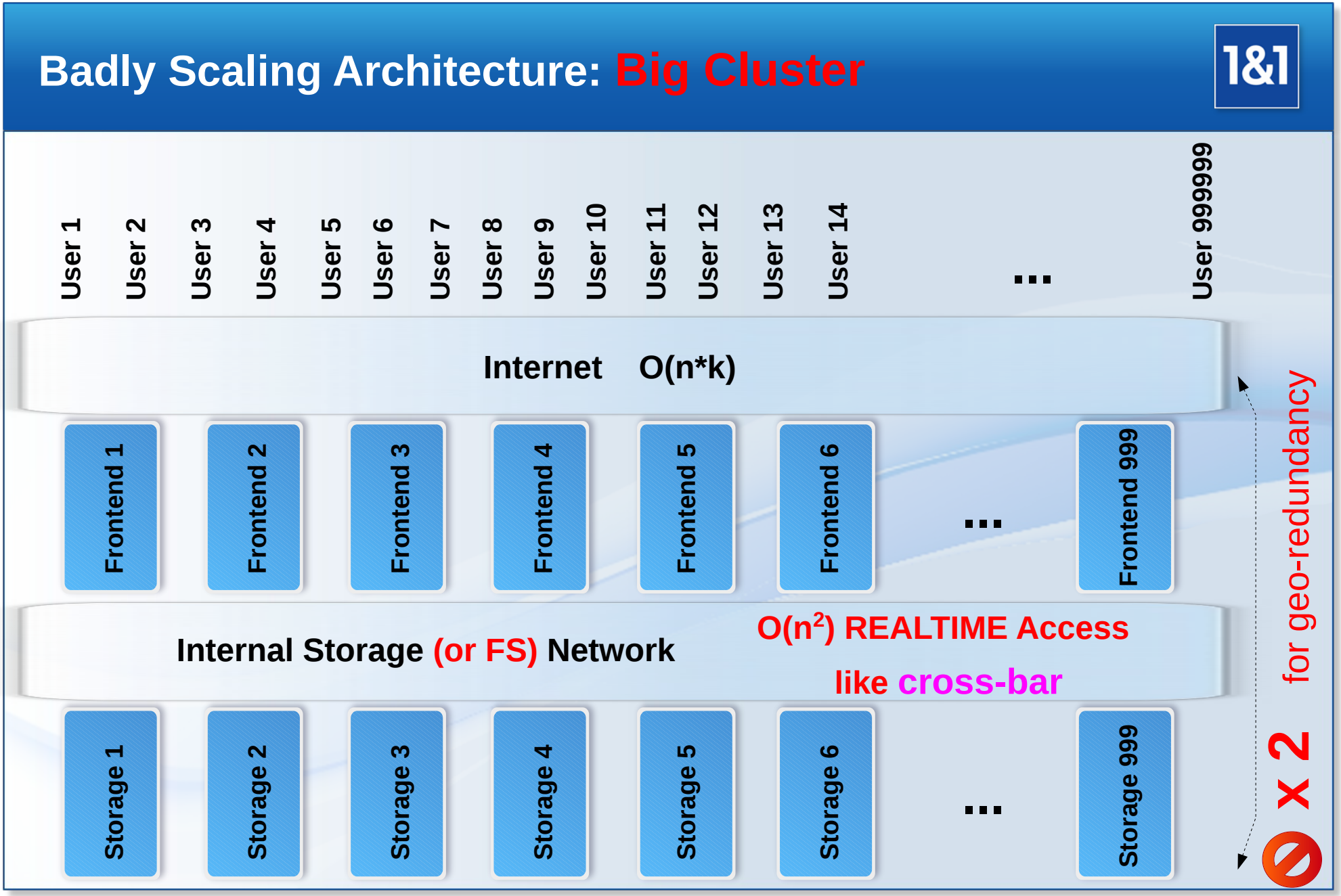
- Distances: **any** (>>50 km)
- Asynchronously
  - near-synchronous modes in preparation
- Tolerates **unreliable network**
- Anytime consistency
  - no re-sync
- Under pressure: no inconsistency
  - possibly at cost of actuality
- Needs >= 100GB in `/mars/` for transaction logfiles
  - dedicated spindle(s)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

DRBD+proxy Architectural Challenge

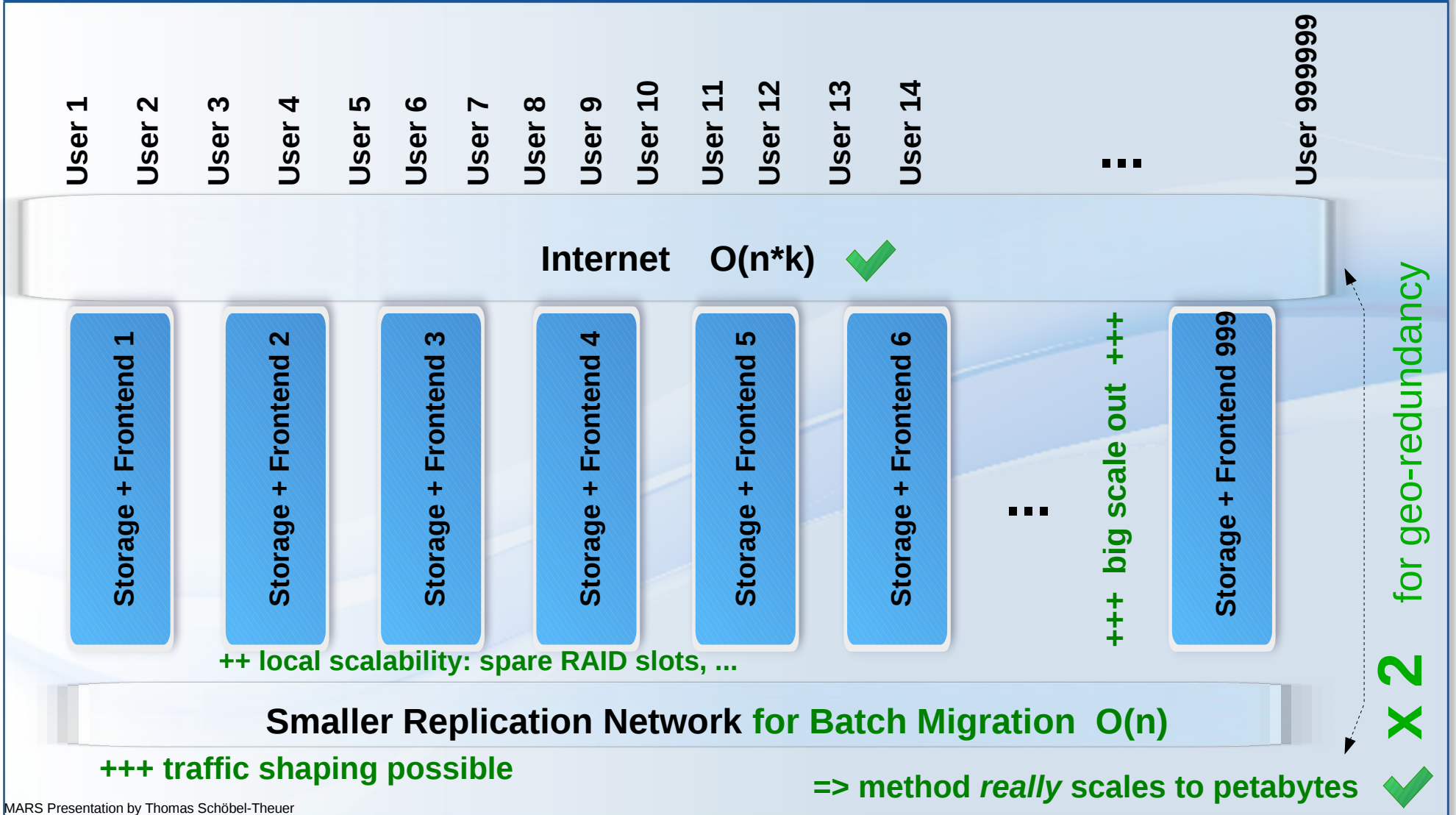


n times
=> need *log*(n) bits for counter
=> but DRBD bitmap has only 1 bit/sector
=> workarounds exist, but complicated
(e.g. additional dynamic memory)

Badly Scaling Architecture: Big Cluster



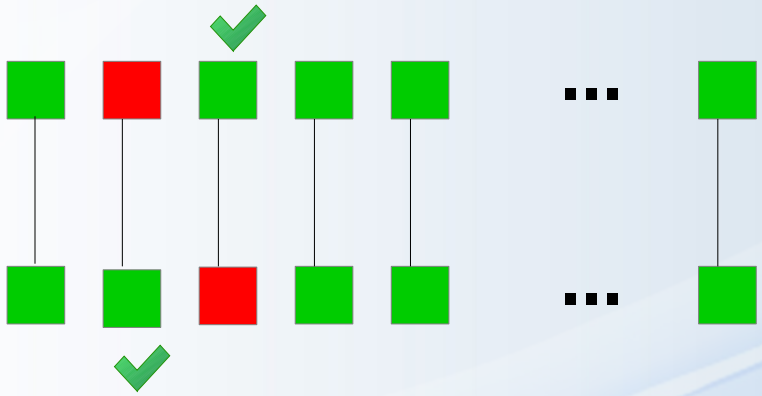
Well-Scaling Architecture: Sharding



Reliability of Architectures: NODE failures

2 Node failure => ALL their disks are unreachable

DRBD or MARS simple pairs

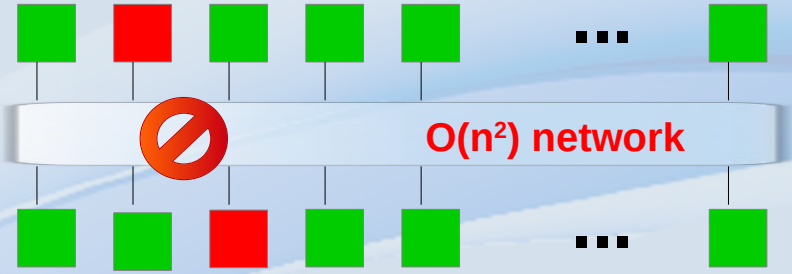


=> no customer-visible incident

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

same n

Big Storage Cluster e.g. Ceph, Swift, ...



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

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

need k >= 3 replicas here

Costs (1) non-georedundant, n>100 nodes

- Big Cluster:
  Typically ≈RAID-10 with k=3 replicas for failure compensation
- Disks: > 300%
- Additional CPU and RAM for storage nodes
- Additional power
- Additional HU

- Simple Sharding:
  Often local RAID-6 sufficient (plus external backup, no further redundancy)
- Disks: < 120%
- Client == Server
  no storage network
  MARS for LV background migration
- Hardware RAID controllers with BBU cache on 1 card
- Less power, less HU

Costs (2) georedundant => LONG Distances

- Big Cluster:
  - 2X ≈ RAID-10 for failure compensation
    (k=6 replicas needed, smaller does not work in long-lasting DC failure scenarios)
- Disks: > 600%
- Additional CPU and RAM for storage nodes
- Additional power
- Additional HU

- Geo-redundant Sharding:
  - 2 x local RAID-6
  - MARS for long distances

    or DRBD for room redundancy
- Disks: < 240%
- Hardware RAID controllers with BBU
- Less power
- Less HU

Costs (1+2): Geo-Redundancy Cheaper than Big Cluster

- Single Big Cluster:
  - ≈RAID-10 with k=3 replicas for failure compensation
- O(n) Clients

  + 3 • O(n) storage servers

  + $O(n^2)$ storage network
- Disks: > 300%
- Additional power
- Additional HU

- Geo-redundant sharding:
  - 2 x local RAID-6
  - MARS for long distances

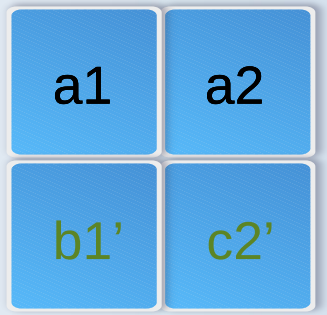
    or DRBD for room redundancy
- 2 • O(n) clients = storage servers

  + O(n) replication network
- Disks: < 240%
- Less total power
- Less total HU

  +++ geo failure scenarios

Costs (3): Geo-Redundancy even Cheaper

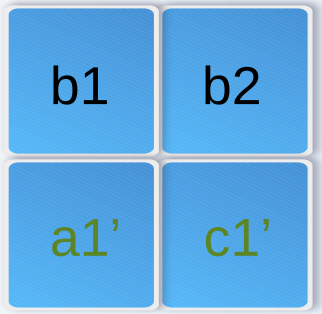
Precondition:
CPU must not be the bottleneck



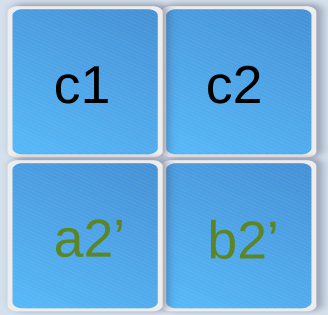
Datacenter 1

Idea: passive LV roles get less CPU

1 datacenter out of 3 may fail



Datacenter 2

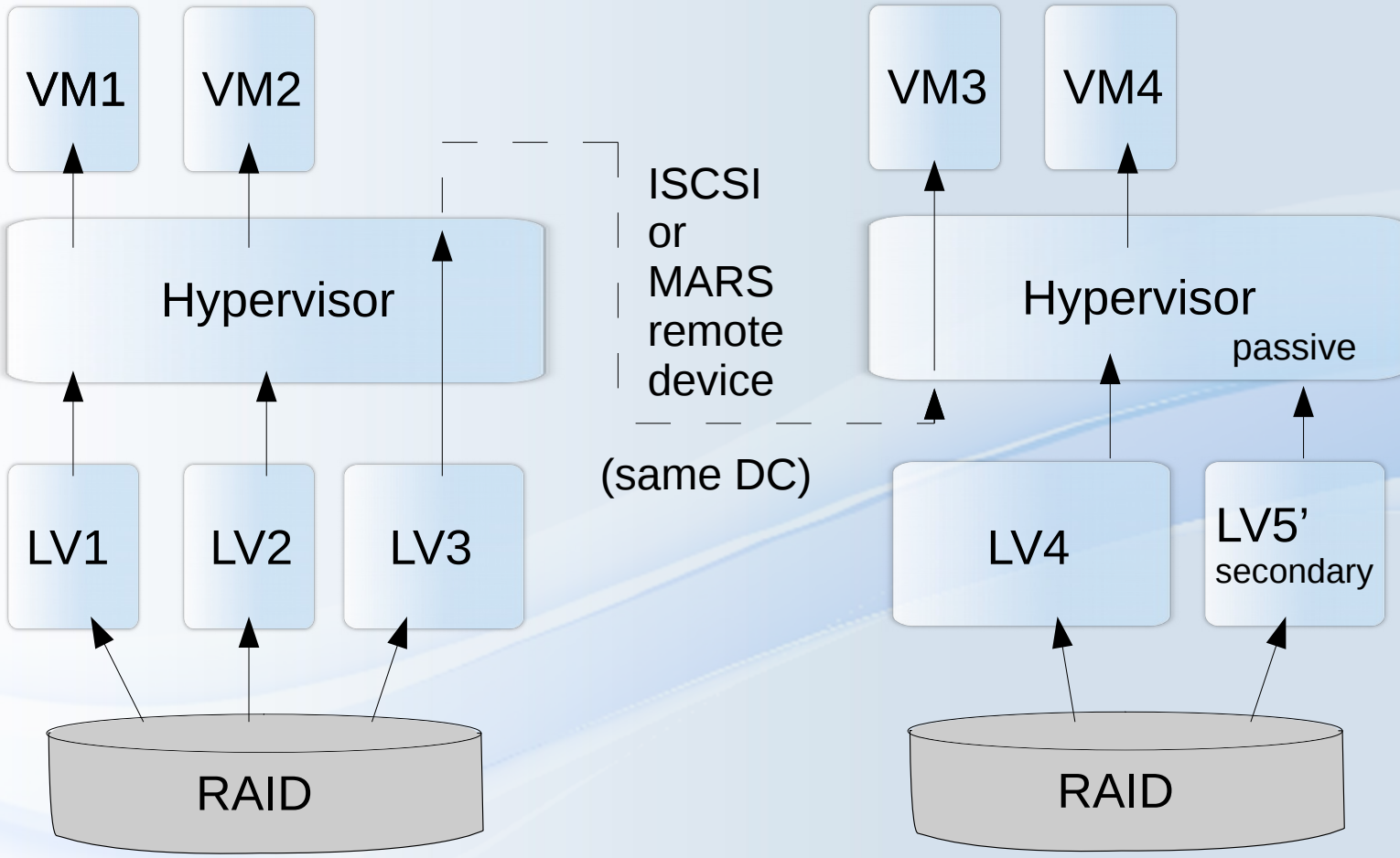


Datacenter 3

Total Storage: x 2
Total CPU: x 1.5
=> 1.5 • O(n)

HOWTO flexible CPU assignment => next slide

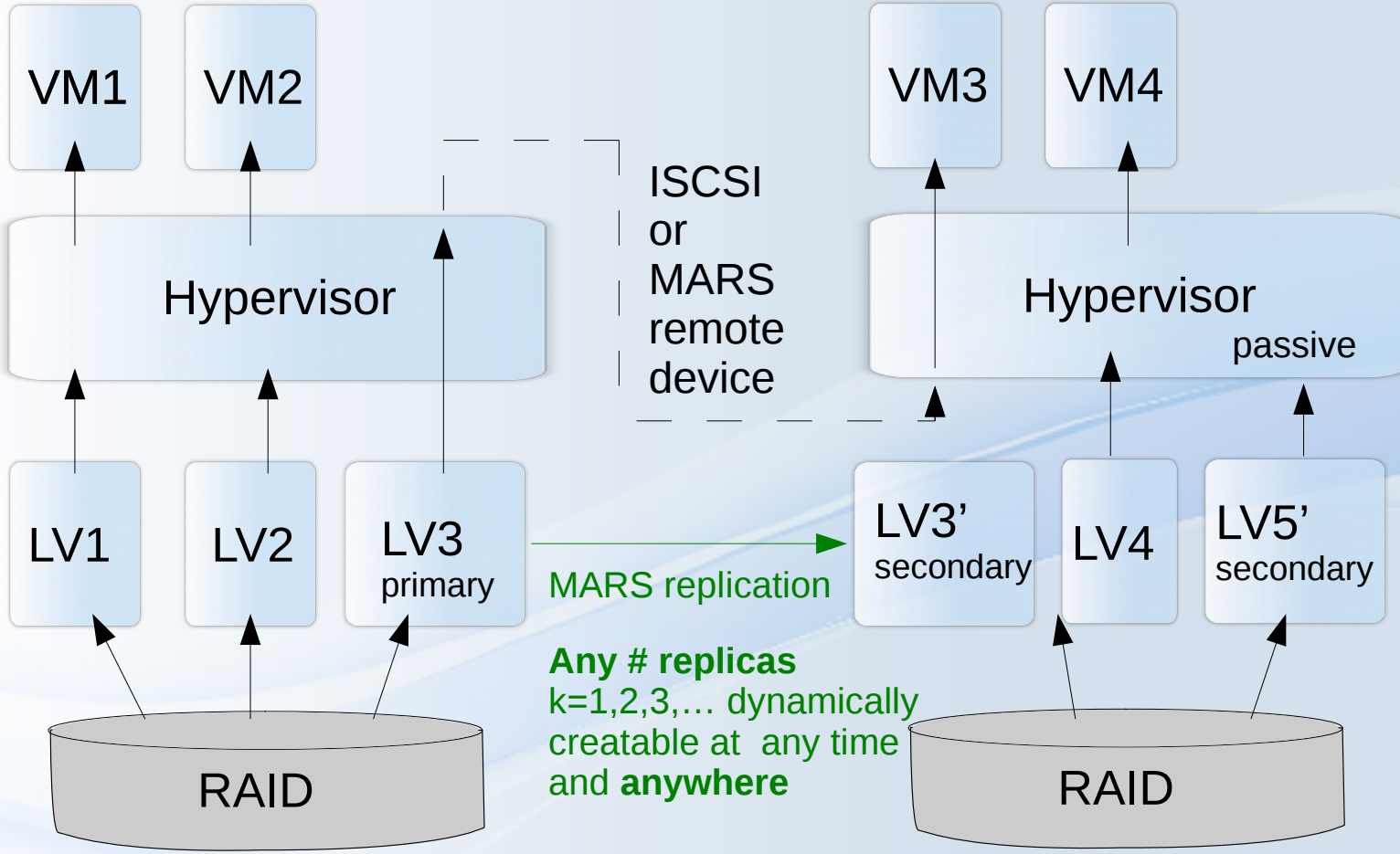
Flexible MARS Sharding + Cluster-on-Demand



any hypervisor works in client and/or server role

and preferably **locally** at the same time

Flexible MARS Background Data Migration



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