RESAR Storage: a System for Two-Failure Tolerant, Self-Adjusting Millions Disk Storage Cluster

Ignacio Corderi, UC Santa Cruz

Thomas M. Kroeger, Sandia National Laboratories

Thomas Schwarz,S.J., Universidad Católica del Uruguay

Darrell D. E. Long, UC Santa Cruz

Where are we?

- World's data production is expanding beyond zettabytes
- Need to manage large numbers of disks
  - Cloud, "Big Data", Exascale computing
- The larger the system the more often components fail
  - Approximately proportional to the number of components
- Component failures leading to disruption of service is unacceptable

Behind the scenes

Currently…

- As data centers grow larger
  - We buy self contained storage units
    - We stack them up
  - Storage containers guarantee tolerance to *k* failures without data loss
  - Recovery is usually slow, often requires partial down time
  - Correlated failures are a big problem

We *can* do better

Key Observations

- Large scale storage organizations should be dynamic
  - Disks enter system in batches
  - Disk capacity changes over lifetime of the system
  - Disks leave the system though failure or decommissioning
- Static (even optimal) layouts for reliability do not adjust well to changes
- The system *must* adapt to this dynamic environment

Failures at large scale

- If you have many things, you will have many failures:
  - Failure rate proportional to number of components (under stochastic assumptions)
  - Correlated (batch) failures can be *much* worse
- Component failure can lead to data loss
- We *mitigate* failure by building *redundancy* into the systems

Redundancy Methods

- Mirroring / Replication
  - Same data stored *n* times
  - Good performance, good reliability, *high* storage overhead
- Parity / Erasure Coding
  - Poor to good performance
    - Requires engineering: caching, large writes, …
  - Good reliability
  - *Low* storage overhead
- Reed-Solomon (error correction) Codes
  - Expensive to compute, expensive to update

Redundancy

- Protecting against a single failure
  - Original data + 1 copy
  - Erasure code with 1 level of protection (RAID5)

What happens when you are recovering and you find out the data on the copy is corrupted?

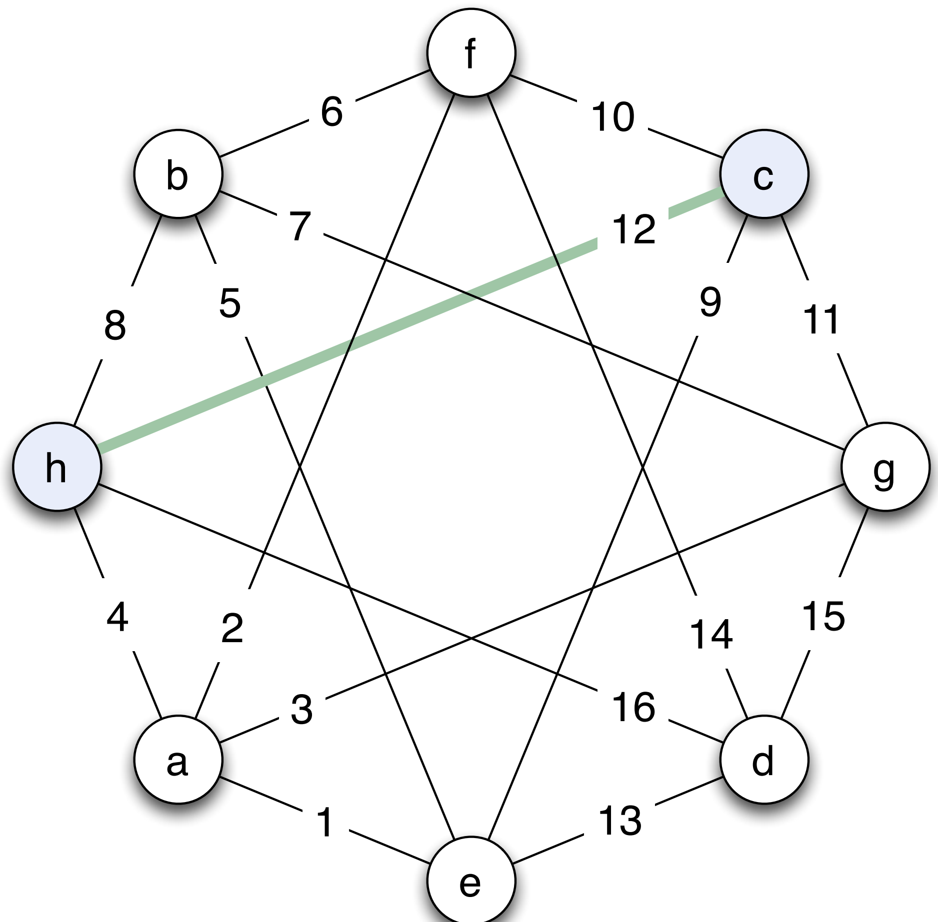
  - Latent sector failures are a problem

- Protecting against 2 failures
  - If when recovering from a failure you encounter some latent failures you can still recover

- Protecting against more than 2 failures?
  - A bit too much for most applications

RESAR

- **R**obust
- **E**fficient
- **S**calable
- **A**utonomous
- **R**eliable



RESAR

- Adaptive, dynamic, autonomous
- Based on XOR codes, fast to compute
- Broader in scope, can be applied to
  - Reliability, energy efficiency, load balancing
- Key idea:
  - The system is represented as an *undirected graph*

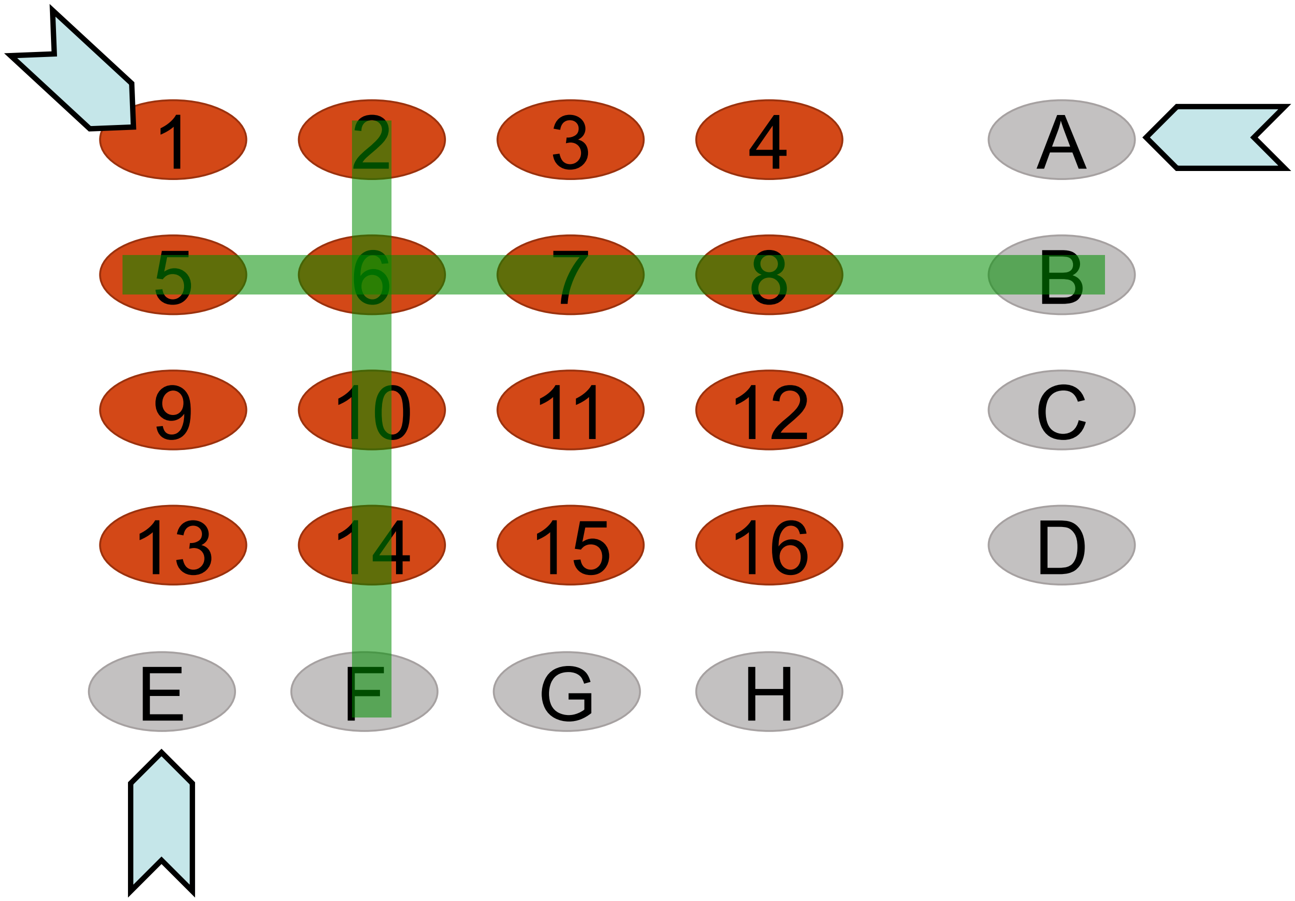
Disklets

- Disks are huge:
  - We use *disklets* of fixed size as basic building blocks
  - Disks have multiple disklets
    - Allows use of disks of different sizes
- Each data disklet is in exactly two parity stripes
  - Higher failure tolerance is usually not needed, but we could use *hypergraphs*
- Disklets are not parts of disks, but an abstraction
  - Low latency disklets could be located on SSD
  - High performance disklets could be stored in RAM

Two-dimensional arrays

- The current solution is a two dimensional RAID layout
  - Each data disk is in two parity blocks
  - Uses a square layout
- What's the problem?
  - Fixed size, rigid layout

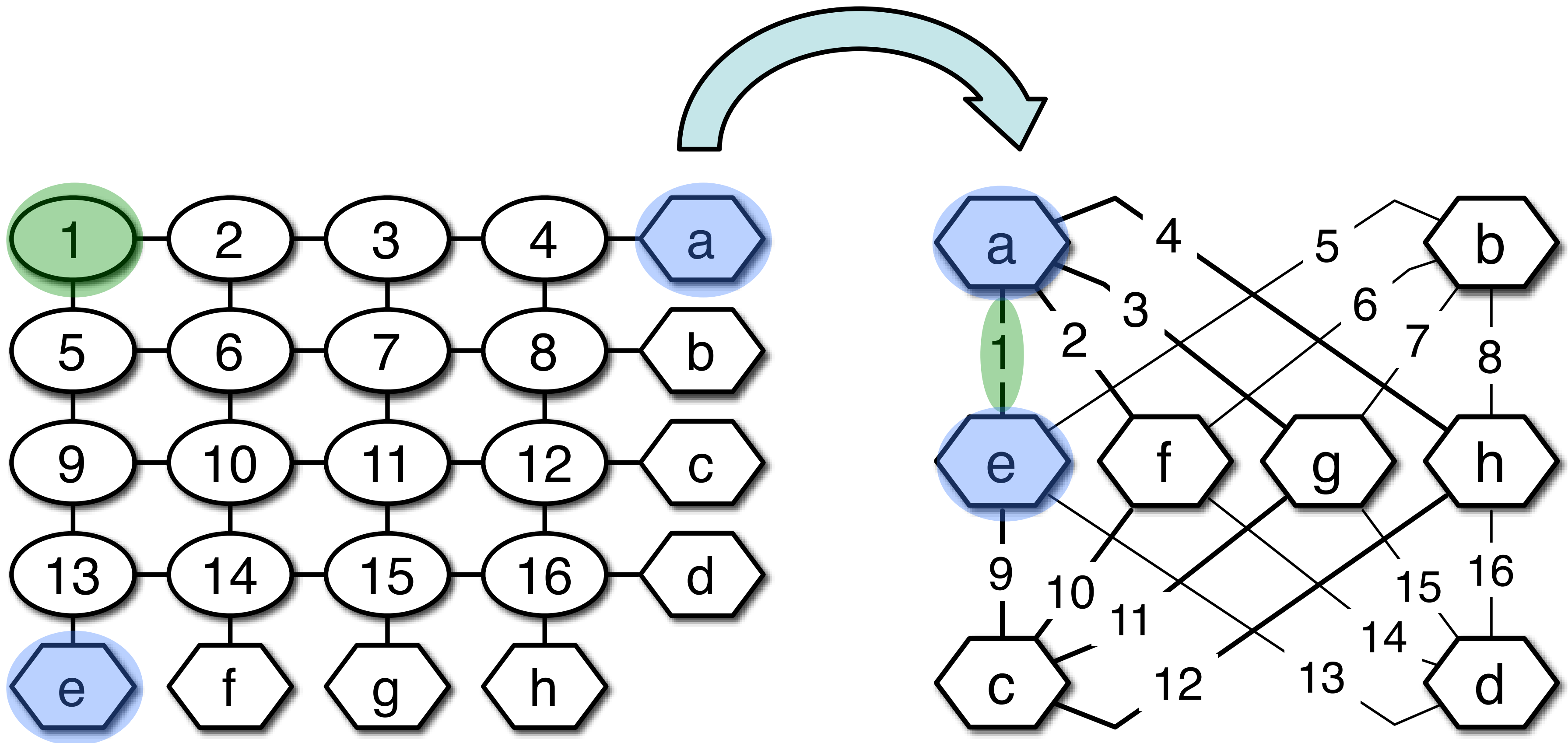
Two-dimensional arrays



Key Observation

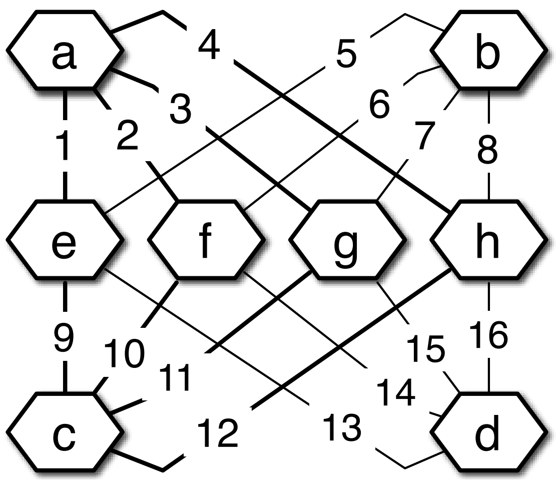
- A RAID array can be viewed as a graph
- The graph is slightly unusual in that:
  - Data (disklets) are the edges
  - Parity (disklets) are the vertices
- In fact, *any* RAID array can be viewed as a graph
  - But not every graph corresponds to a RAID array

Two-dimensional array to Graph

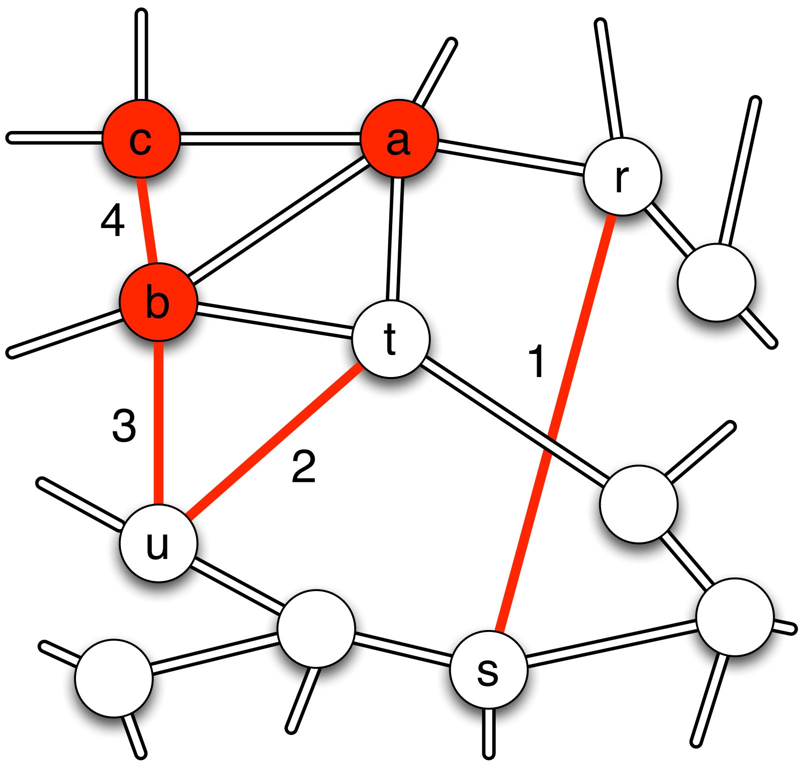


Graph Representation

- This frees us from the rigid structure
- *Any* graph corresponds to a disklet layout
  - Data disklets are edges
  - Parity disklets are vertices
  - A reliability stripe is a vertex and all edges adjacent to the vertex

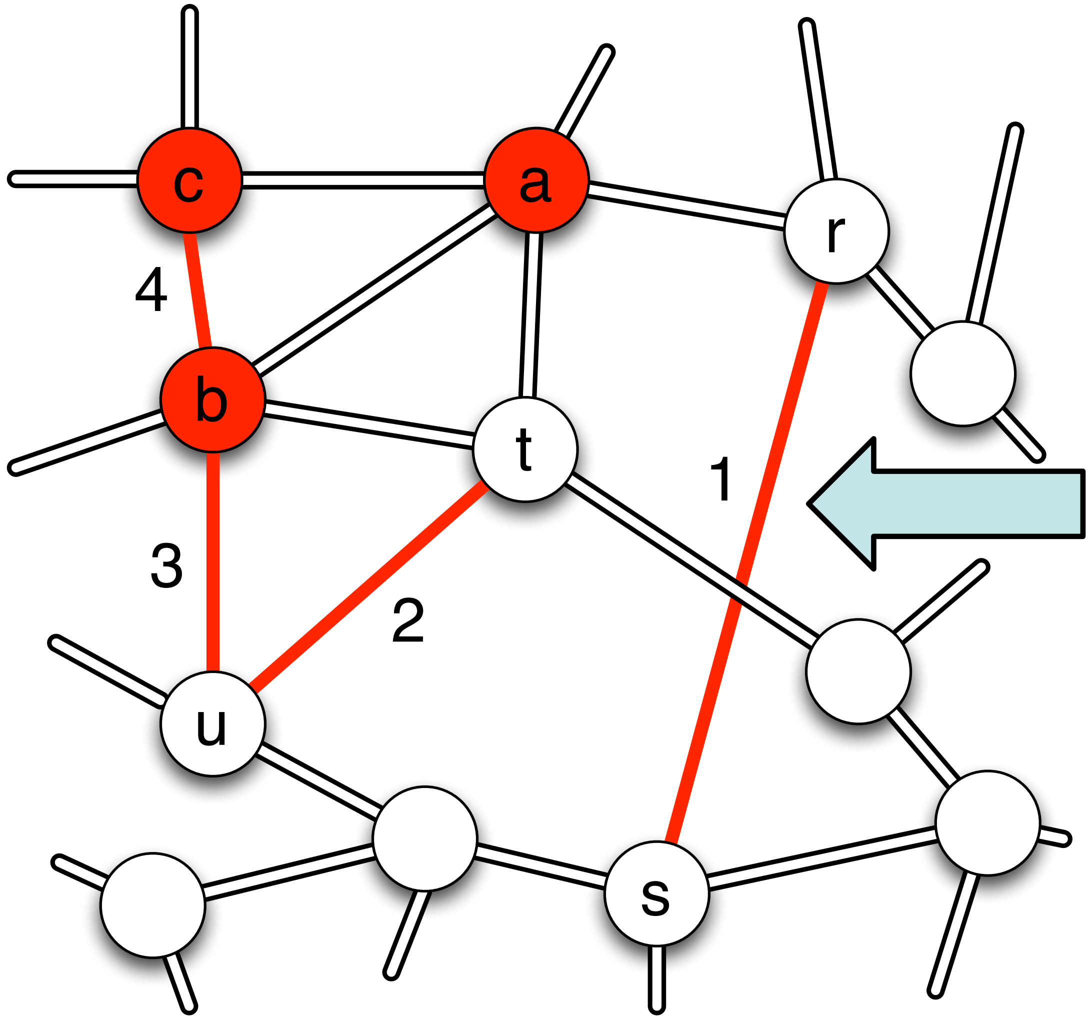


What do failures look like?



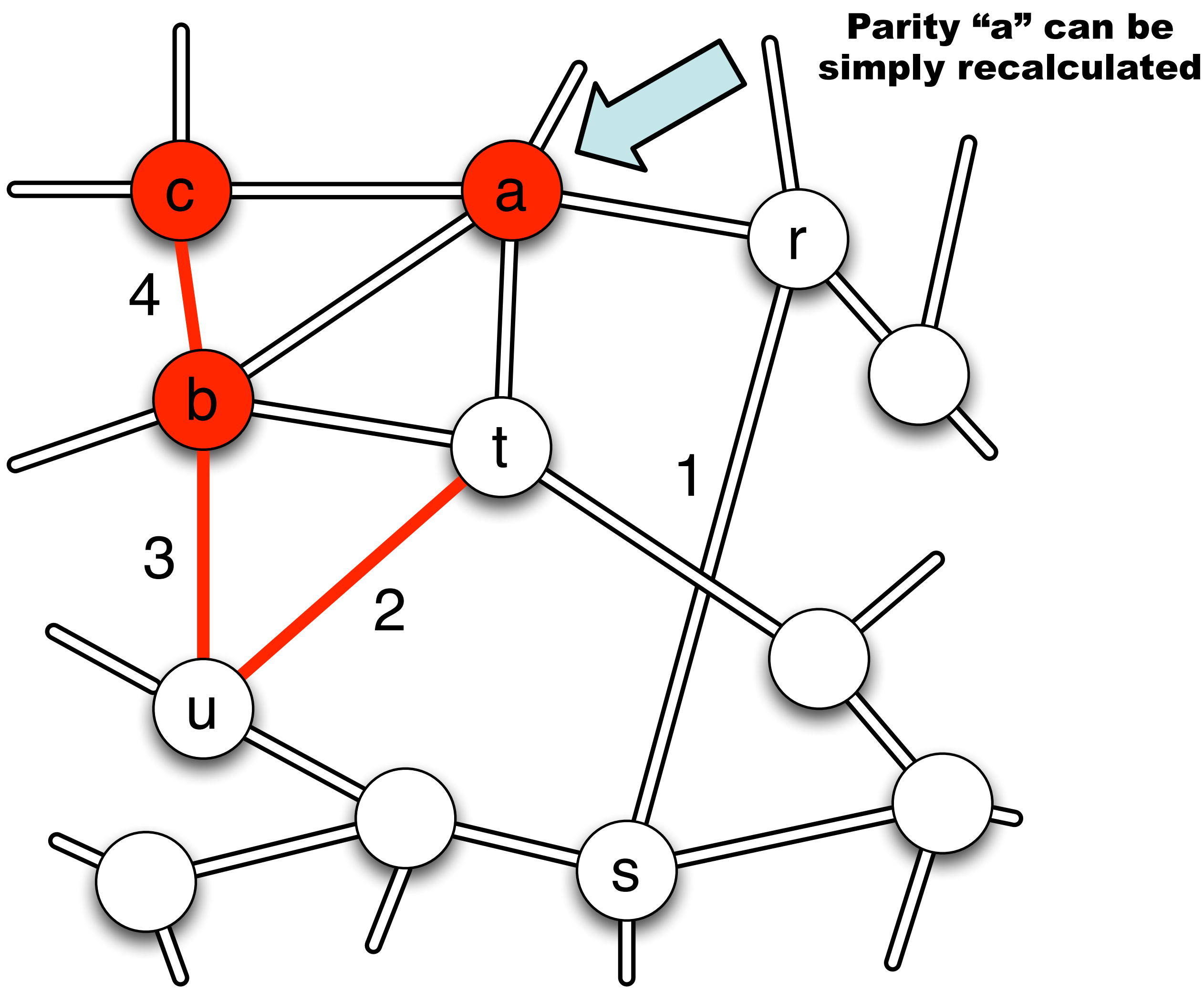
- Failed parity are solid red vertices
- Failed data are bold red lines
- Recovery must be done based on topological sort of the failed subgraph

Recovery

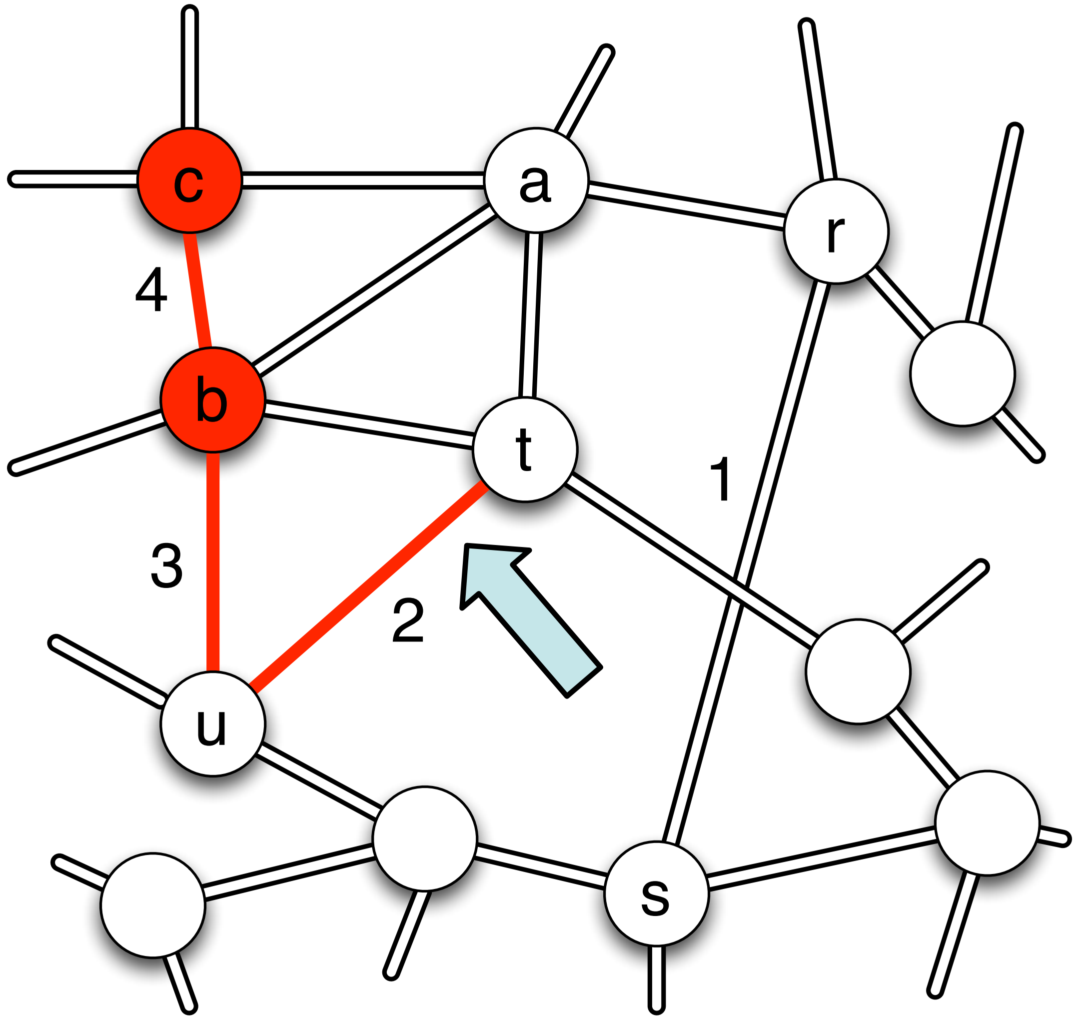


First fix data "1"
We can use groups "r" or "s"

Recovery

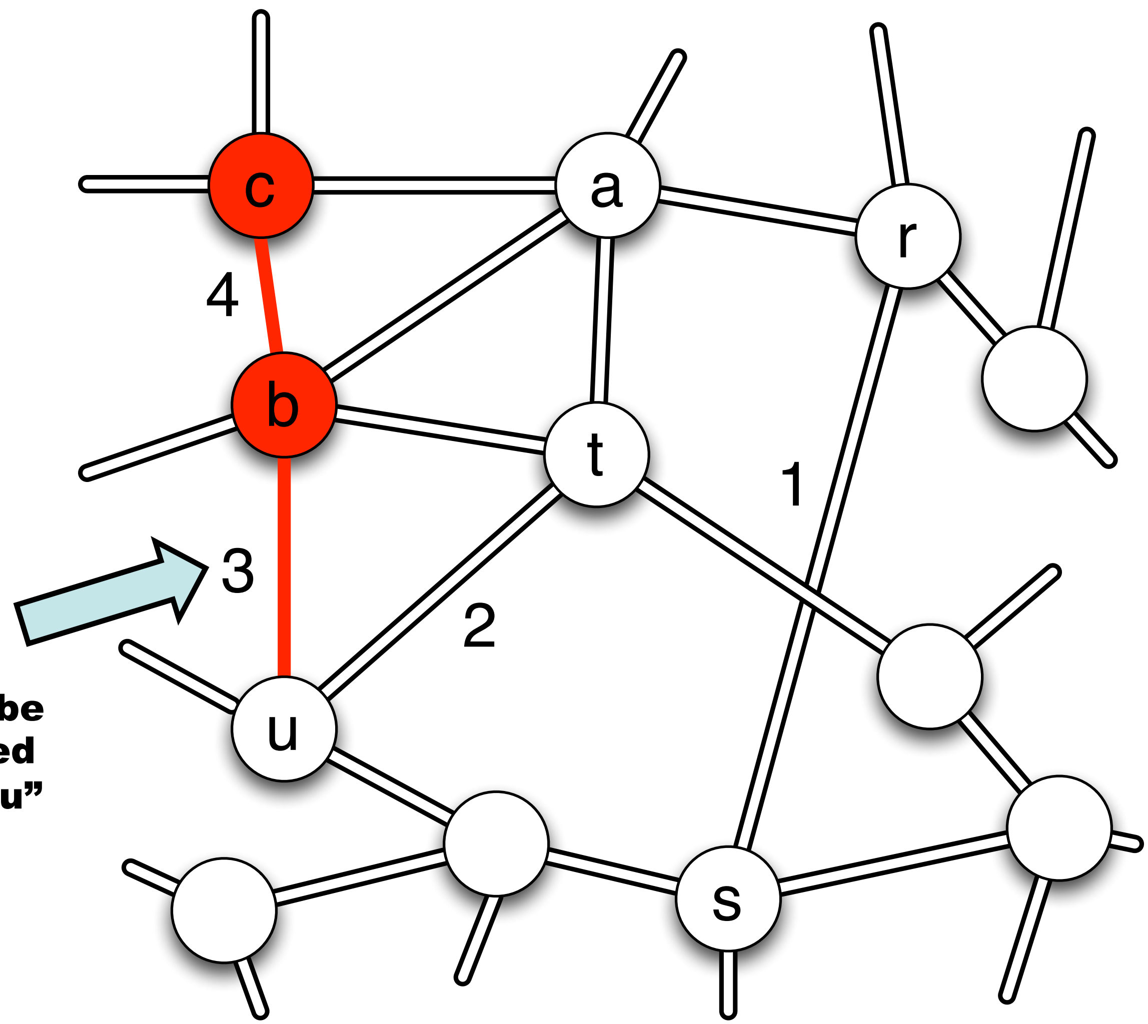


Recovery



To recover data "2" we can only use group "t"

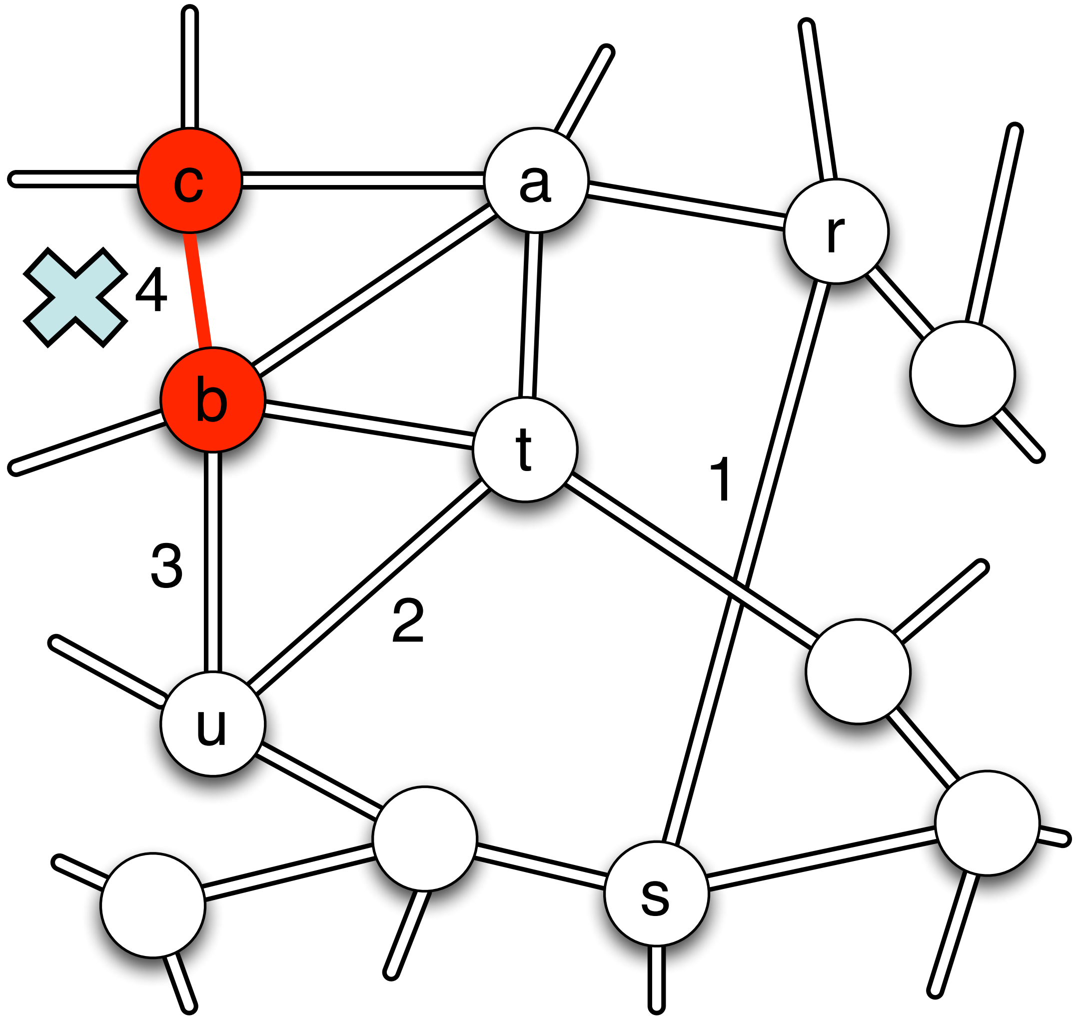
Recovery



Data "3" can be now recovered using group "u"

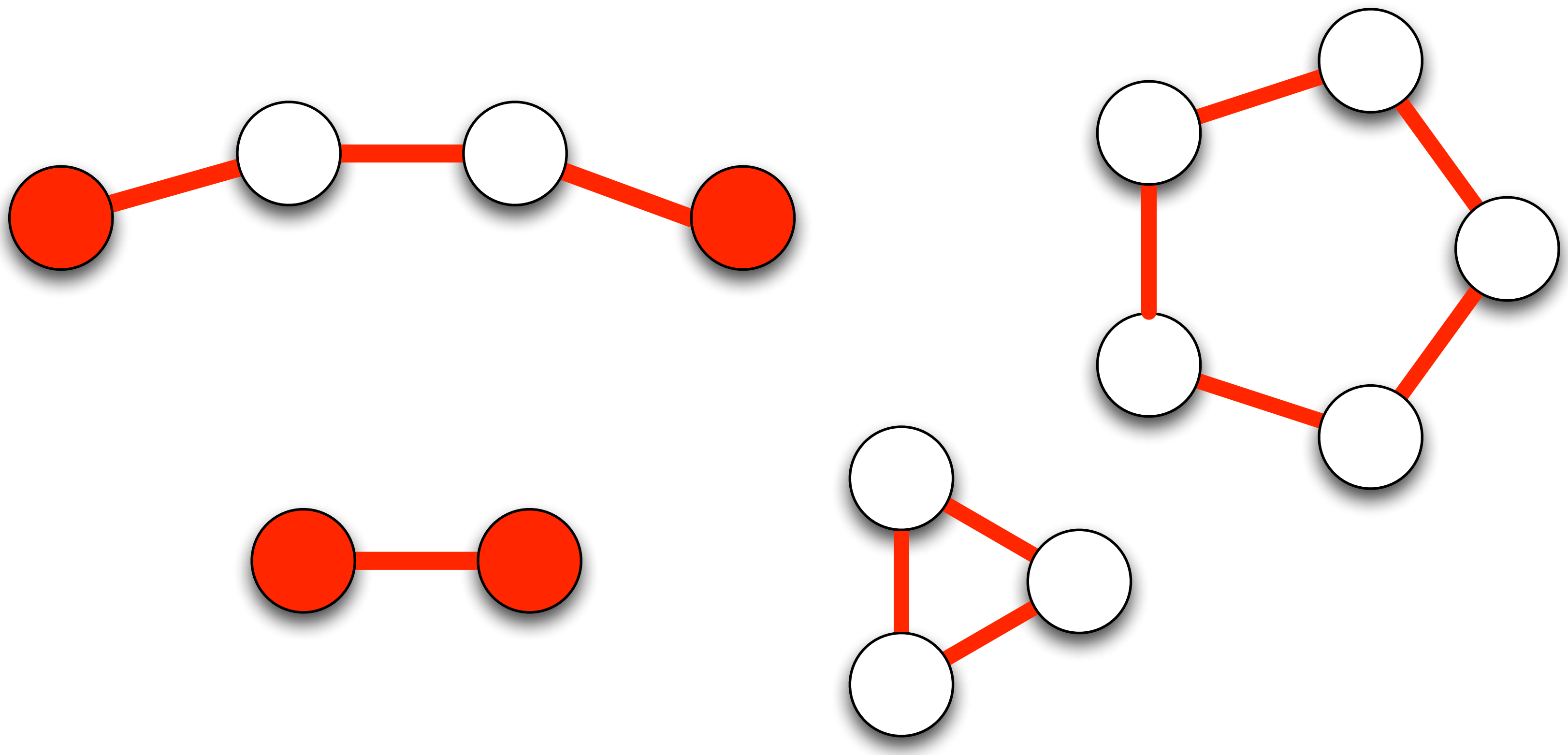
Recovery

Data “4” is not recoverable



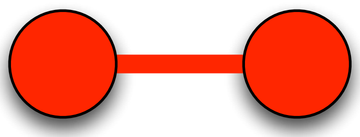
Irreducible failure patterns

- These patterns represent data loss

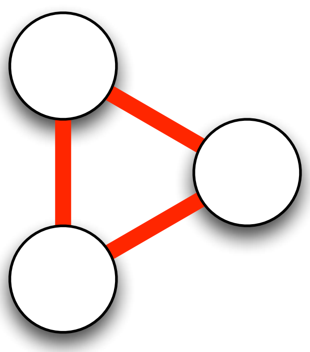


Failure patterns

- Not all layouts (graphs) are equal
  - We cannot avoid the barbell



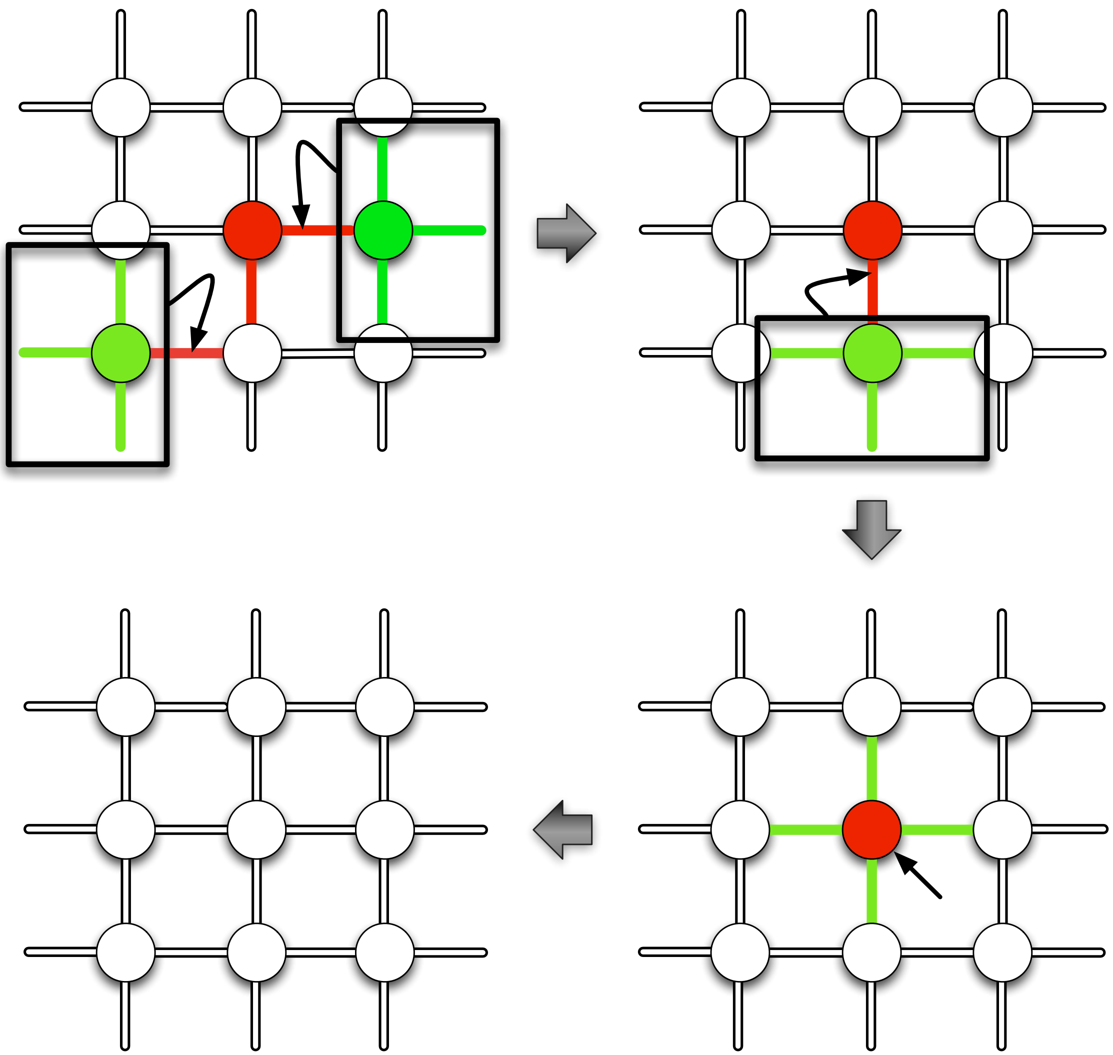
  - But we can avoid triangles



Simultaneous Recovery

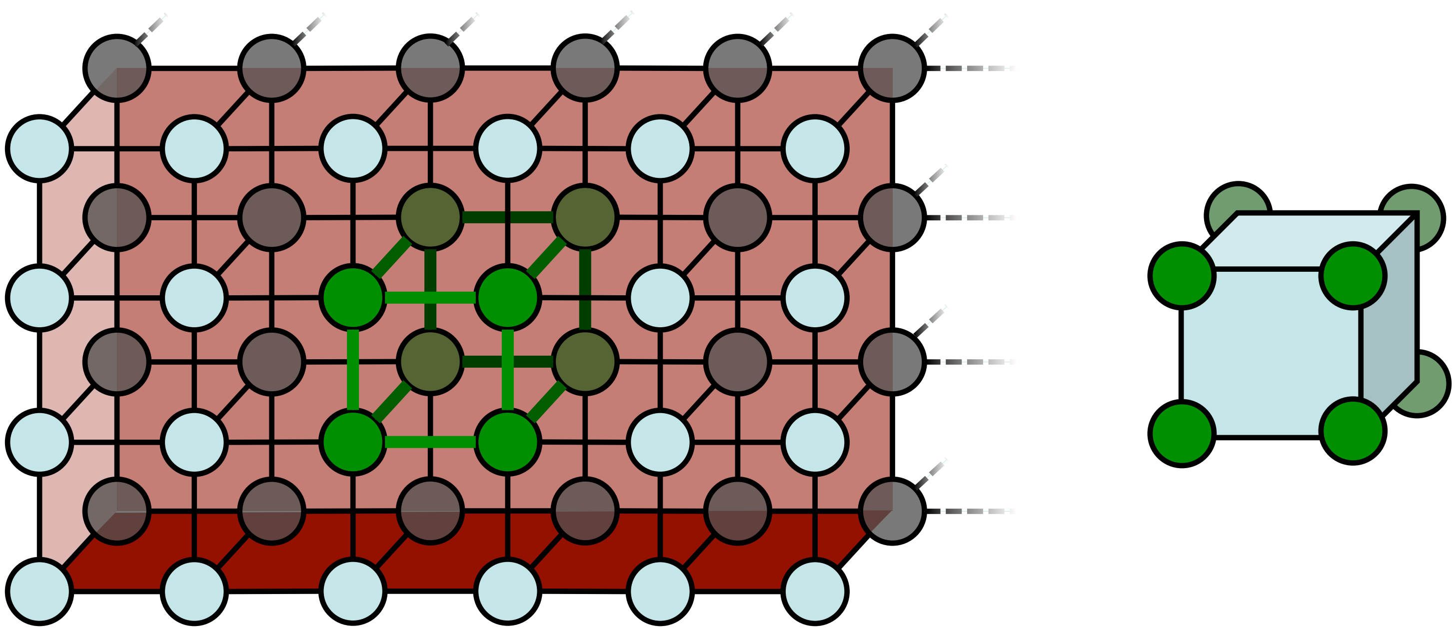
Most of the previous recoveries can happen at the same time.

Any group with a single failure can engage in recovery at the same time.



Proposed layout

- Graph based on an n-dimensional grid
  - Triangle free
  - Vertex degree = 2n



Making it work

- Disklet to disk assignment
  - On which disk do we put a given disklet?
- Incorporating new disks
  - What happens when I buy a new rack of disks?
- Load distribution
  - What's the cost of recovery?
  - What happens when "hot" data from different disklets ends up on the same disk?

Disklets to disk assignment

- *Requirement*:
  - Simultaneous failure of *two* disks *must not* lead to data loss
- *Solution*:
  - Graph coloring with added restrictions
- *Restriction*:
  - Two elements (edge, or vertex) with the same color must be at least at a *walking distance* of *two* from each other
  - This prevents single or double failure from generating irreducible failure patterns

Coloring Algorithm

- For each disklet on the graph
  - 1. Select randomly a disk from the non-full disks pool
  - 2. Check coloring constraints
  - 3a. If valid then
    - 3.1. Assign disk color to disklet
    - 3.2 If disk cannot have more disklets then remove from pool
  - 3b. Else go back to 1

- Random selection limited to 10 tries, after that the pool is permuted.
  - This *never* happens.
- Each disk is a different color, and provides a homes for a certain number of disklets.

Hierarchical coloring

- Drive failures are not always independent, sometimes a whole server goes down taking with it 20 drives, or a tsunami takes out your entire data center.
- You can sustain double failures of disks, servers, racks, rows, rooms, floors or locations.
  - Provided that you have enough elements of that type.
- You can apply this algorithm to disklets all the way up to data ceters.
  - Use the servers, racks, etc. as colors and applying the coloring algorithm.

Adding new disks

- When you buy a new rack you need to assure the reliability of the data you are going to place there

- Simplistic way: make a new isolated graph
  - Drawback: Correlated failures or “infant mortality” will cause you to lose data

- A more elaborate solution:
  - Expand the perimeter of the graph then run coloring algorithm on the new structure to swaps colors between the new perimeter and the core.

  ✓Prevents data losses due to correlated failures!

Load Balancing

- What happens if multiple "hot" disklets end up on the same disk?
- How can we adjust the layout to better balance the disks load based on disklets load?

- Heat maps on the graph can identify stressed groups

- Taking "cold" disklets and swapping them with some of the "hot" disklets on a disk can reduce the disk load

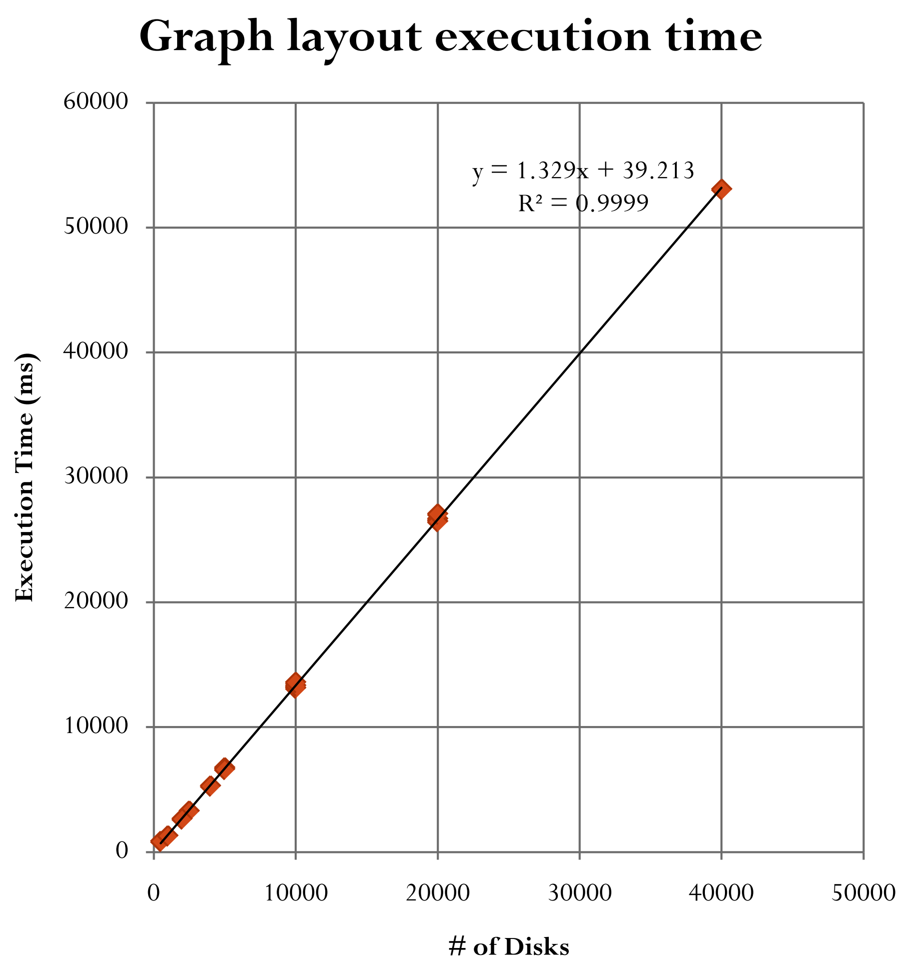
Energy Saving

- Color frozen disklets with same color and shut down the disk

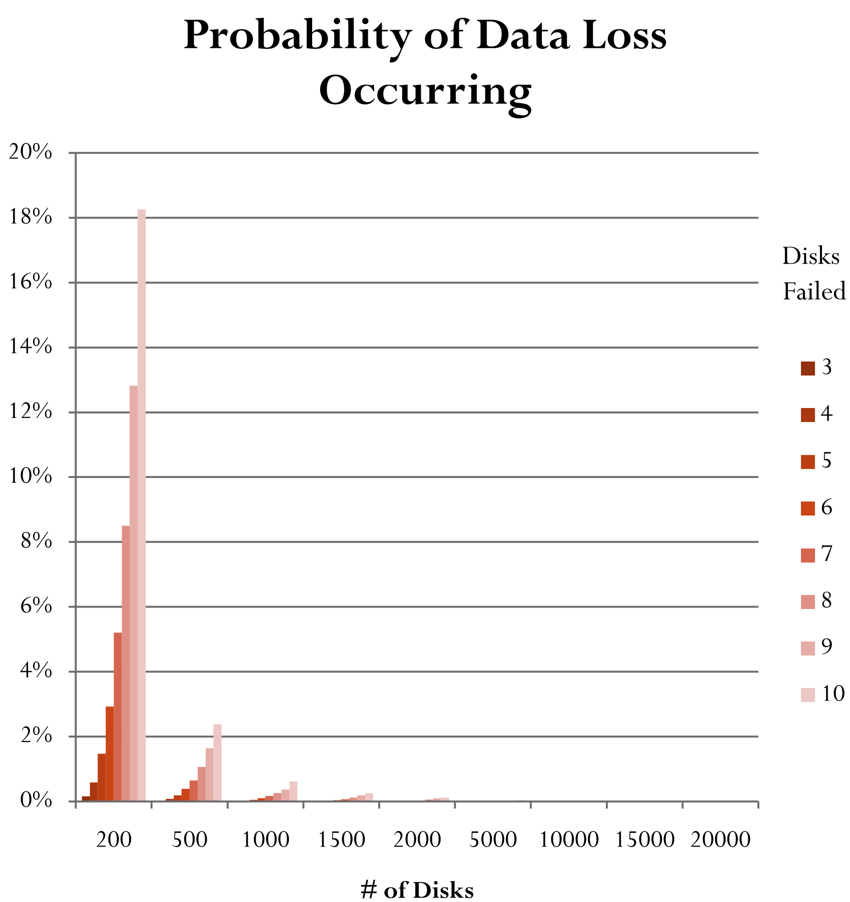
- Tradeoffs between load balancing and energy saving can be adjusted for the specific deployment.

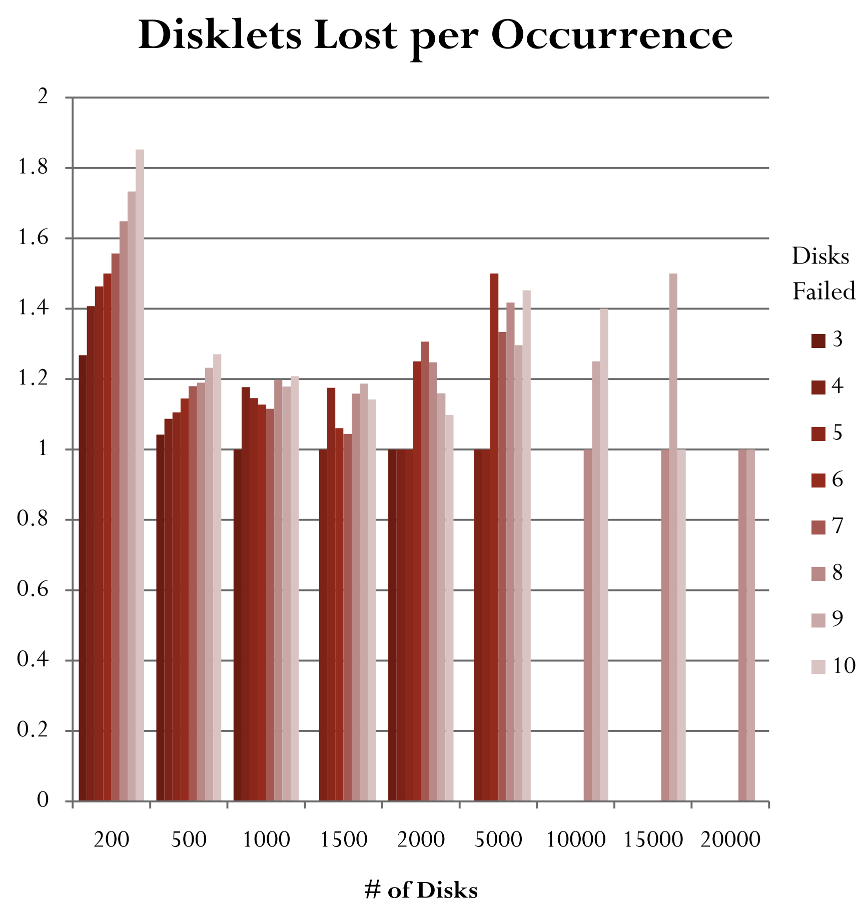
Layout: Execution Time

- Graph layout is linear on the number of disks
- Execution time is roughly 1.329ms per disk
- **Very** fast layout

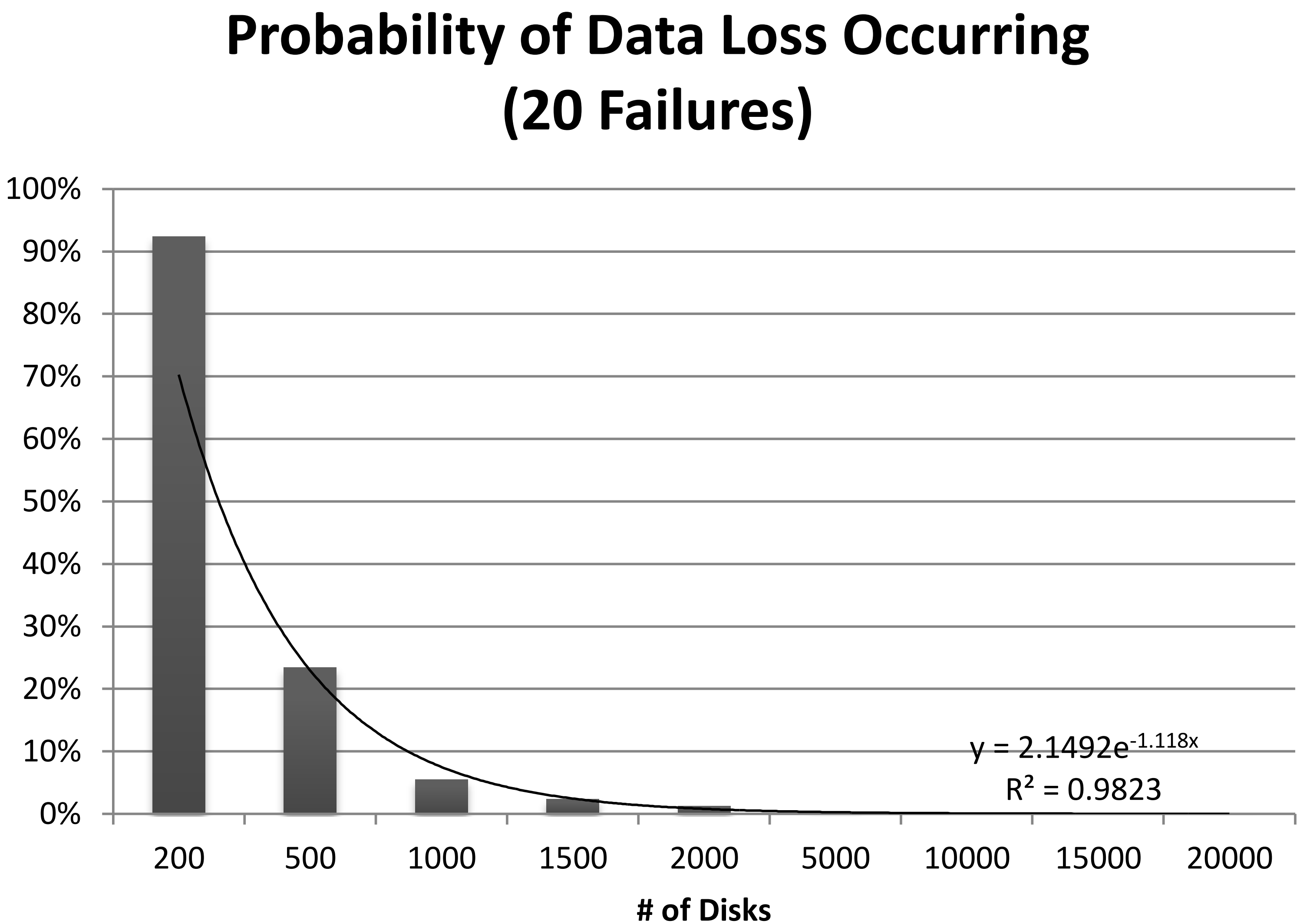


Failure Tolerance

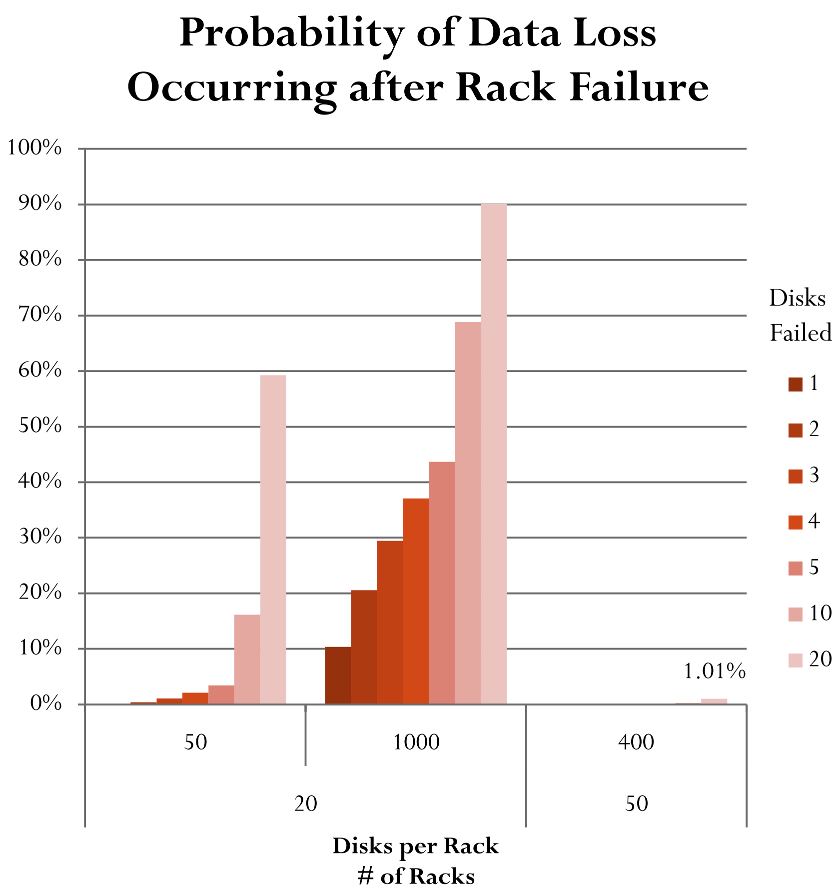


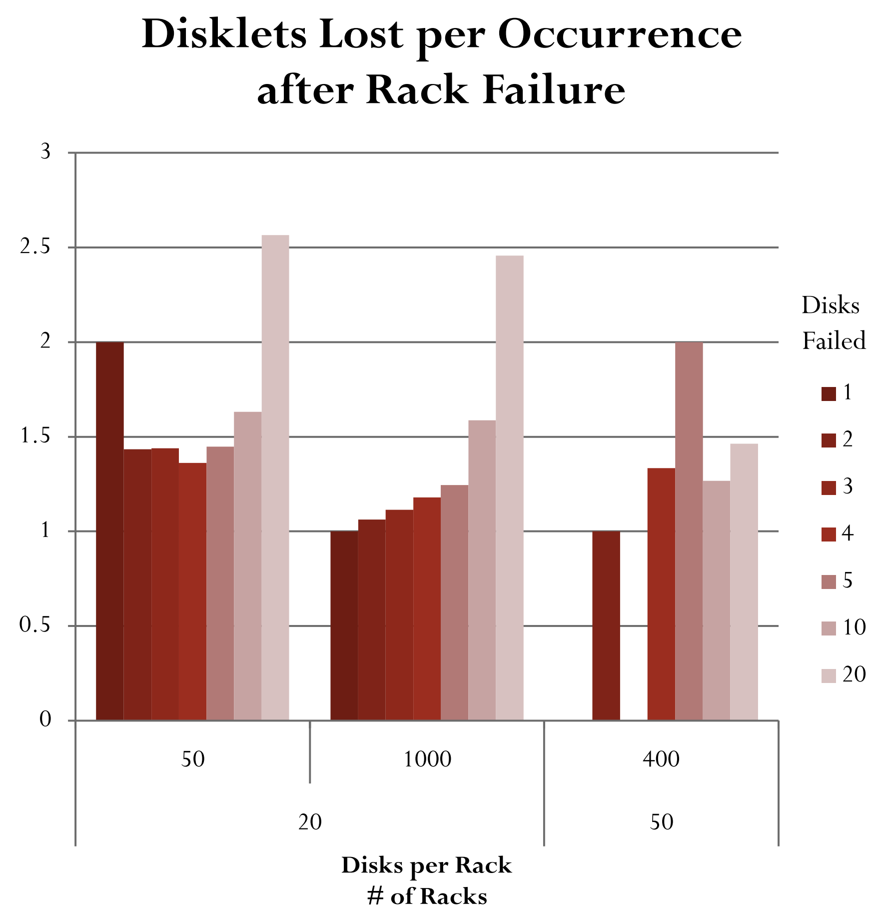


Failure Tolerance

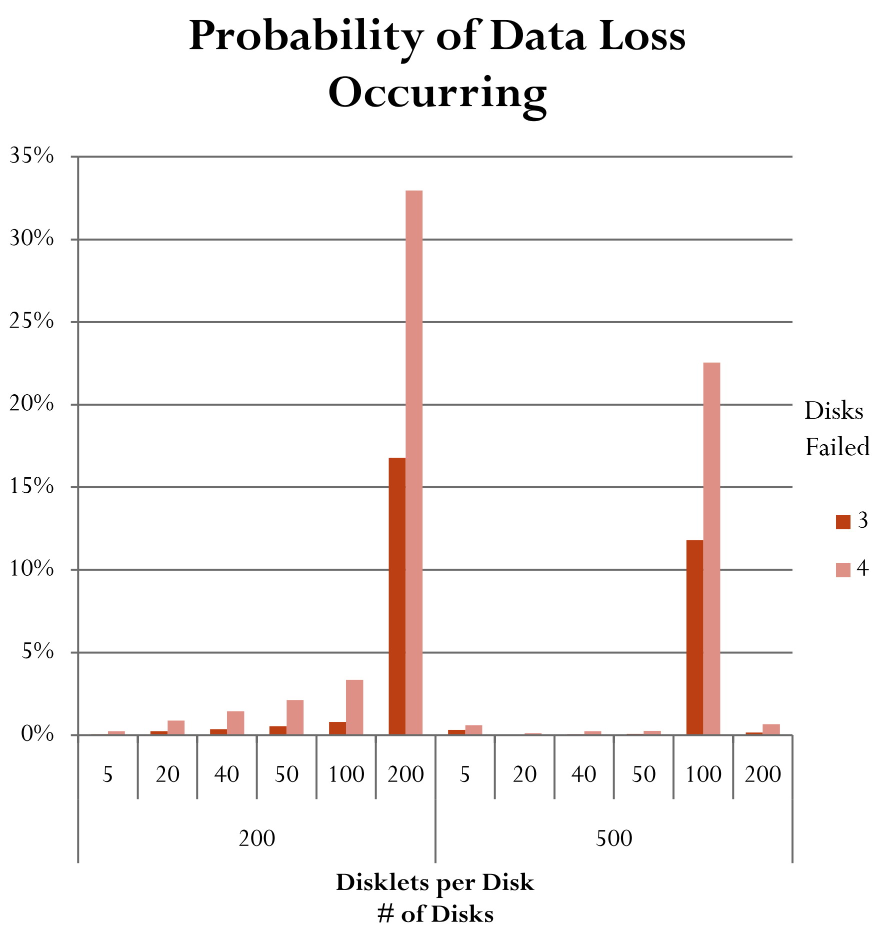


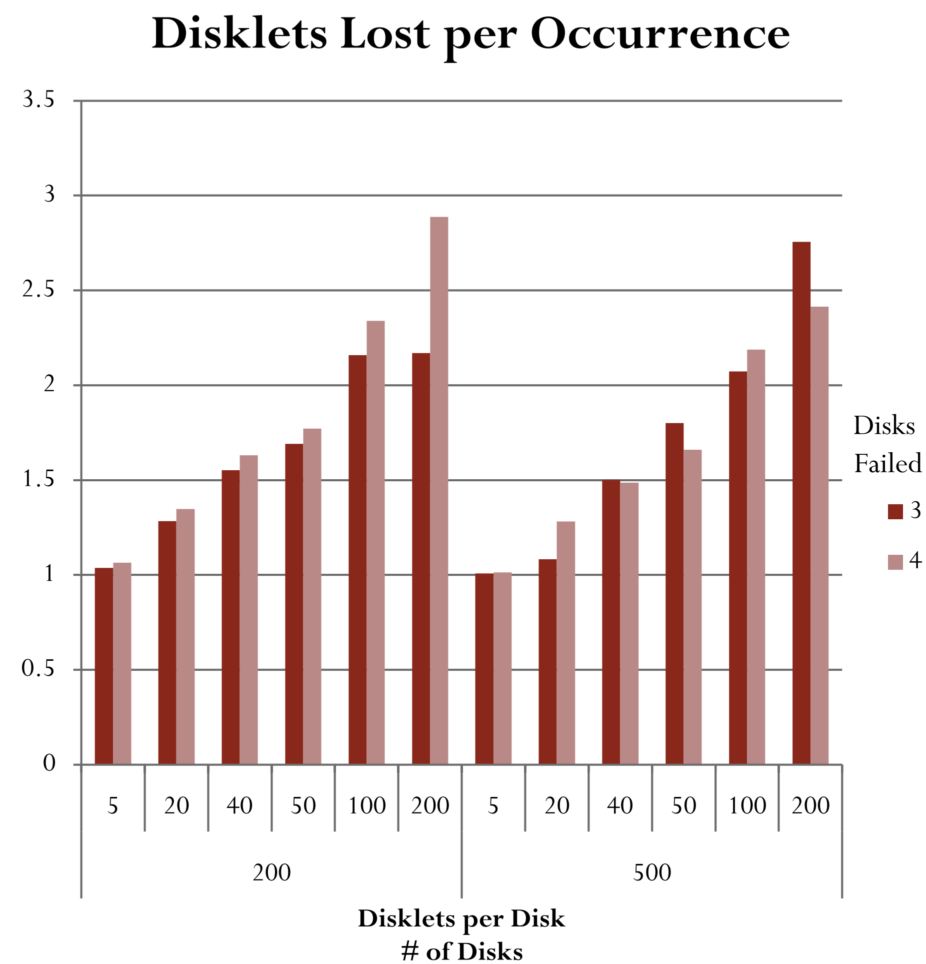
Correlated failures





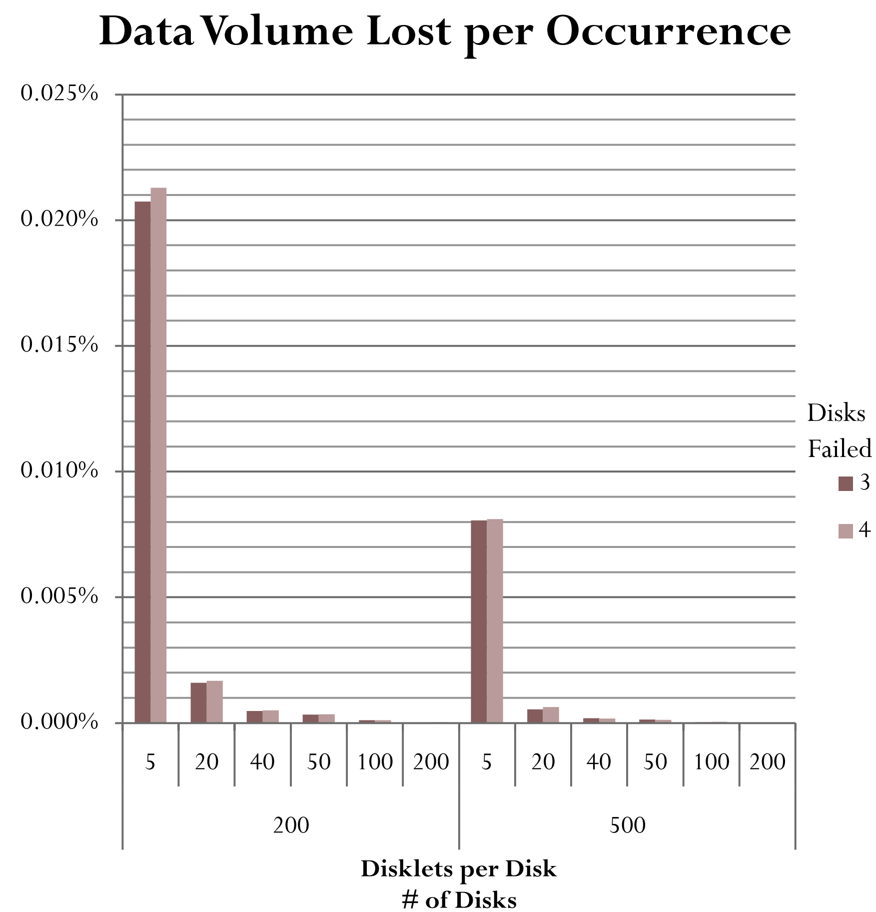
How do the disklets per disk affect reliability?





How do the disklets per disk affect reliability?

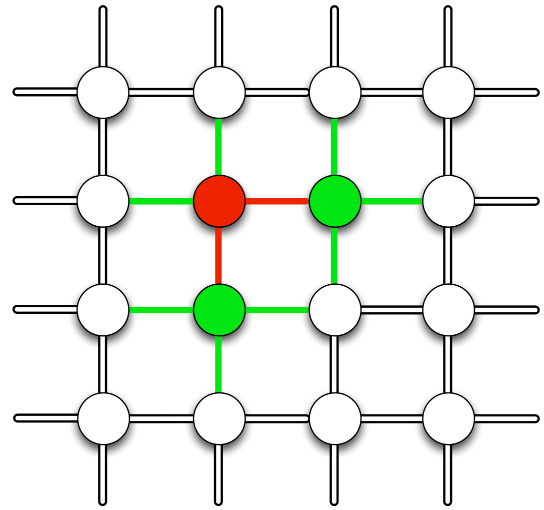
- Units lost increases with disklets per disk
- The % of actual data lost actually decreases



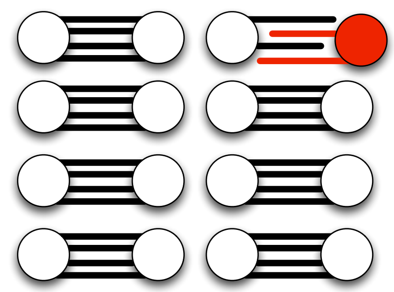
Distributed RAID 6 and Replication

Comparison on Probability of Data Loss

- With **20%** storage overhead RESAR is **15** times more resilient than RAID 6 (RESAR vs. 8+2 codes)



- At the **same storage capacity** RESAR is almost **14** times more resilient than triplication.

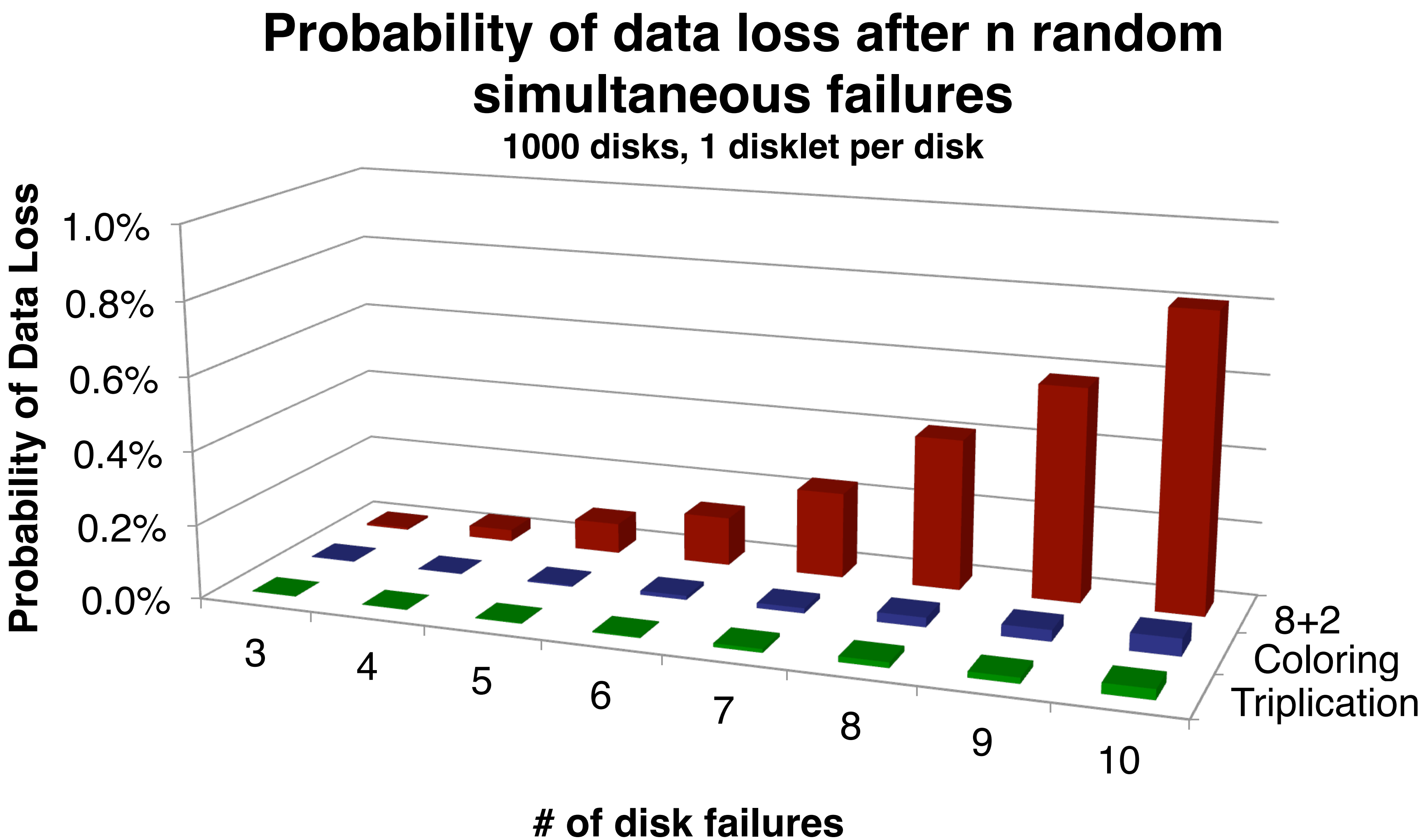


Comparisons

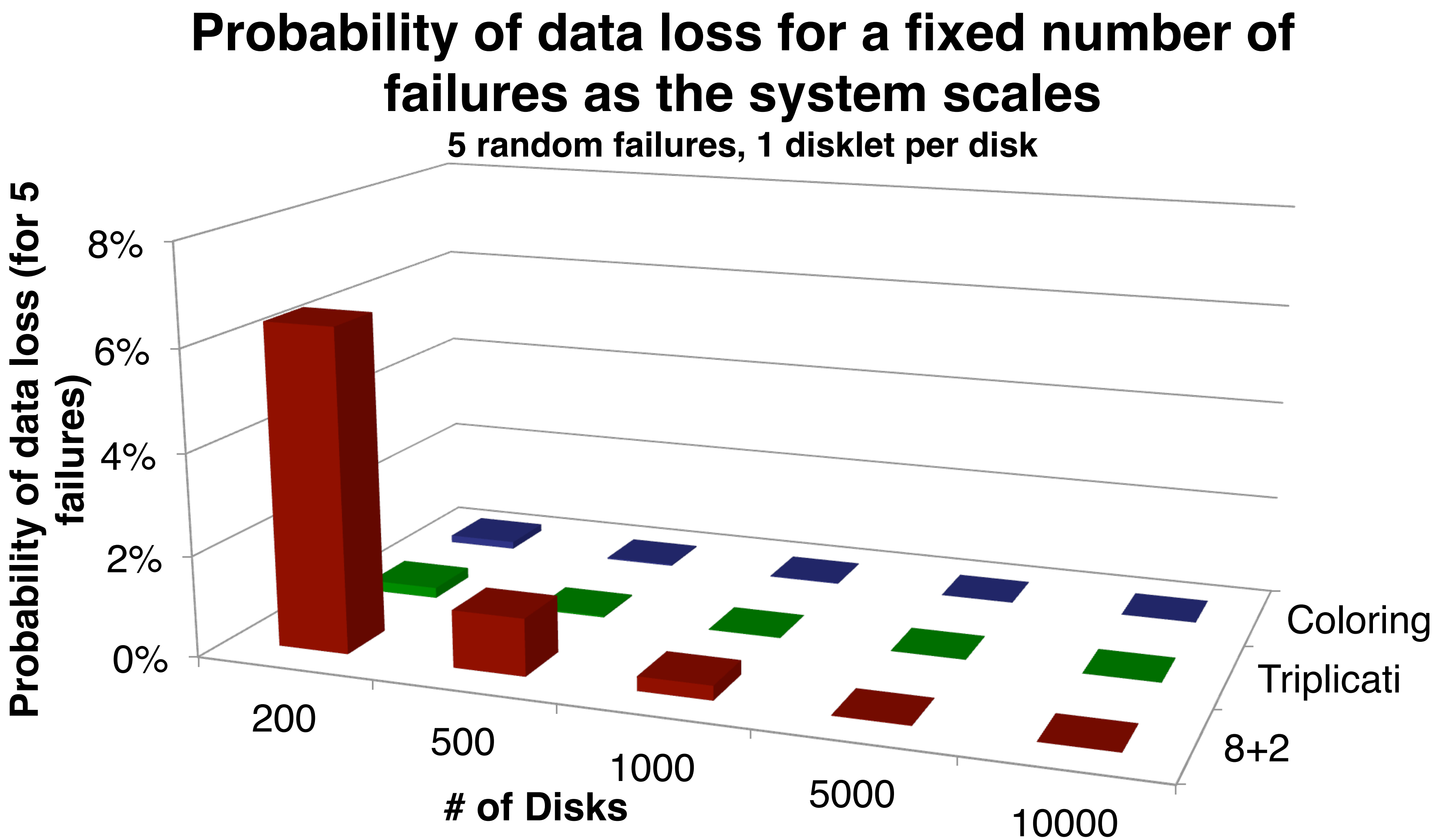
- RAID 6
  - At 8+2 offers same storage overhead (80% of storage capacity is usable for data) and same guarantees

- Triplication
  - Offers same guarantees at the cost of an extra 200% of storage (only 33% of storage capacity is usable for data)

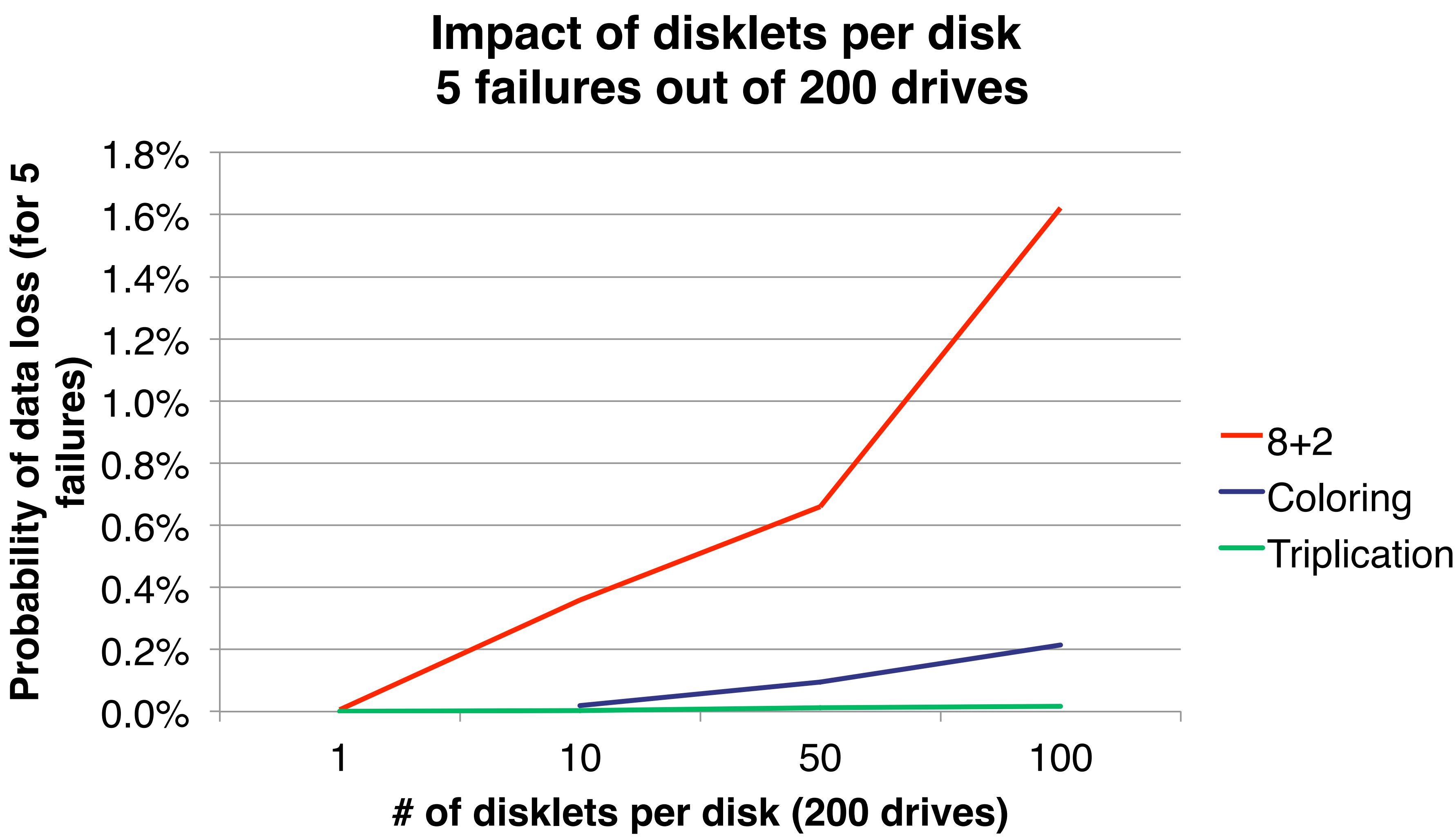
Random Failures



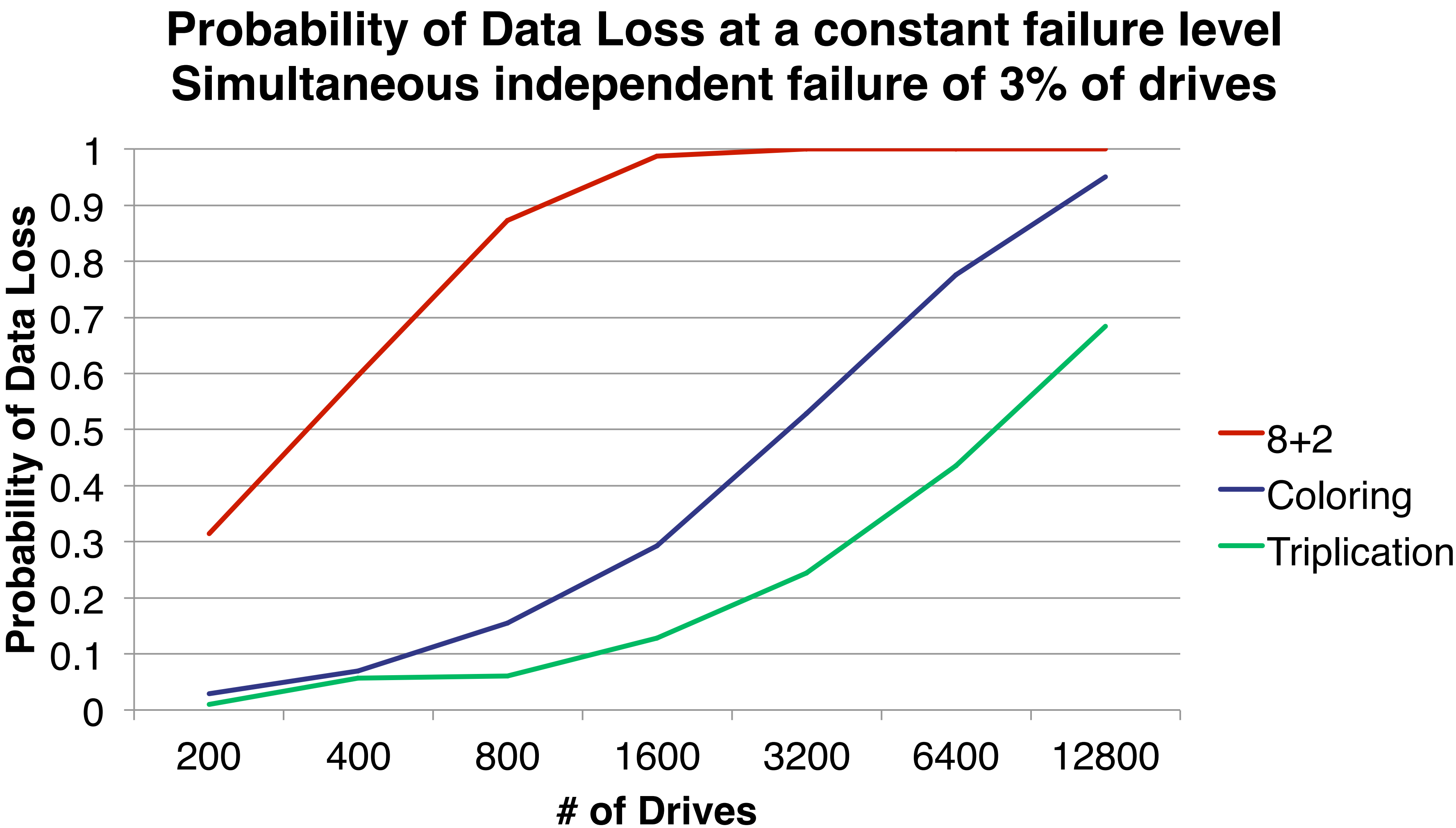
Random Failures



Disklets per disk



Constant failure level



History

- In 2010 we proposed the idea on PDSW'10
- In 2011 we evaluated and compared it with triplication and erasure codes

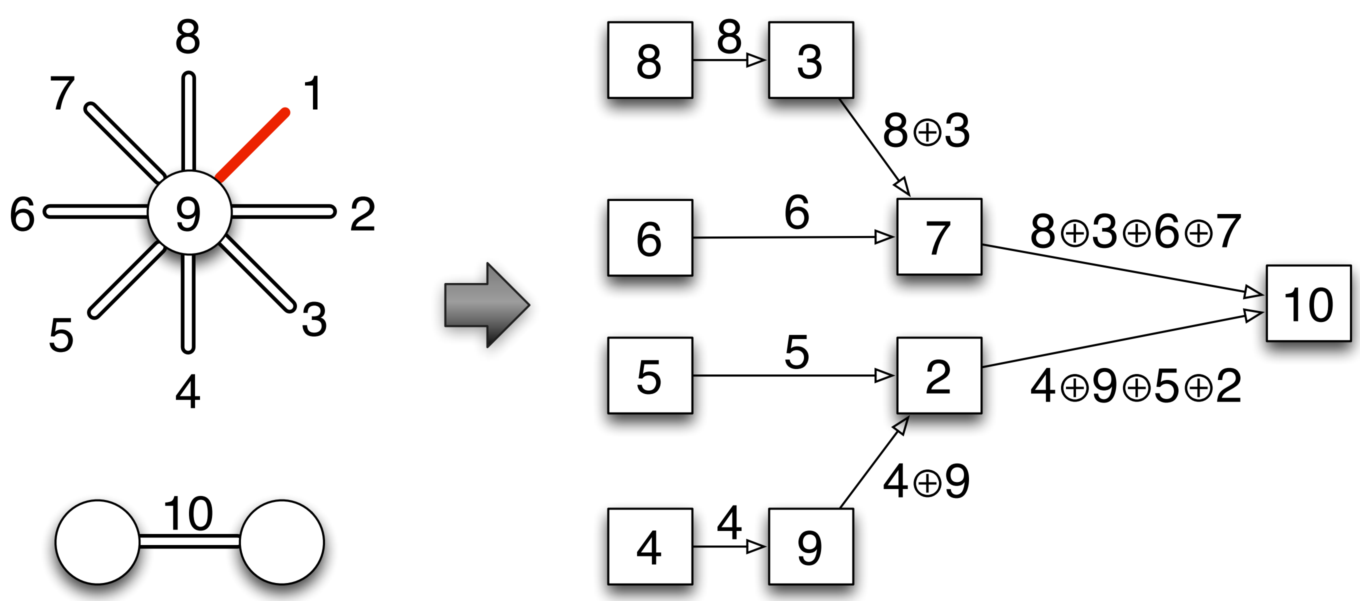
Can we build a system based on RESAR that scales to millions of drives, targeting both HPC and cloud systems?

Summer 2012
First implementation

- Goal: 1 Million Drives
- Megatux (Sandia National Labs)
  - Lightweight virtualization platform developed by Sandia
  - Virtualized Infrastructure with 20,000 servers
  - Each server emulated 50 hard drives
- Recovery Times < 4 minutes
- Years of operation emulated with *zero* data loss

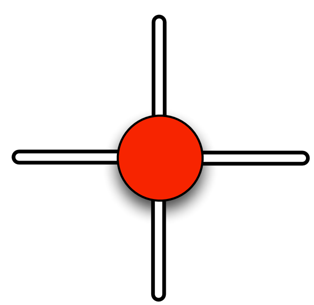
Disklet Recovery Process

- Massively distributed recovery
- 100% decentralized
- Recovery pipeline constructed based on utilization

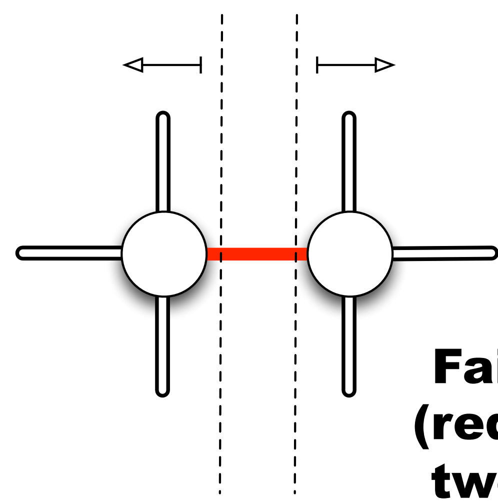


Servicing requests during recovery

- RESAR has *no downtime* during failure recovery
- Data protected by two groups
- One group can recover while the other can service requests



Failed parity (red) and disklets involved in recalculation (white)



Failed data (red) and it's two groups (white)

Choosing the right size

- Disklet size impacts:
  - Recovery time – takes longer to read
  - Recovery bandwidth requirements – more disklets = more traffic
  - # of resources involved in recovery – more disklets = more disks
- 5 GB disklets on 4 TB drives
  - Recovery = 40 seconds
  - Disks used = 6,552
  - On 1 million drives not an issue, for 10,000 a bit too much

Simulation Clock

- Running the system in real-time would take too long
- Global emulation clock sped up
  - This adds some positive noise because of the 50 Virtual Machines running on each PC.
  - With a clock multiplier of 600x a few extra hundred milliseconds add up to minutes.

Drives

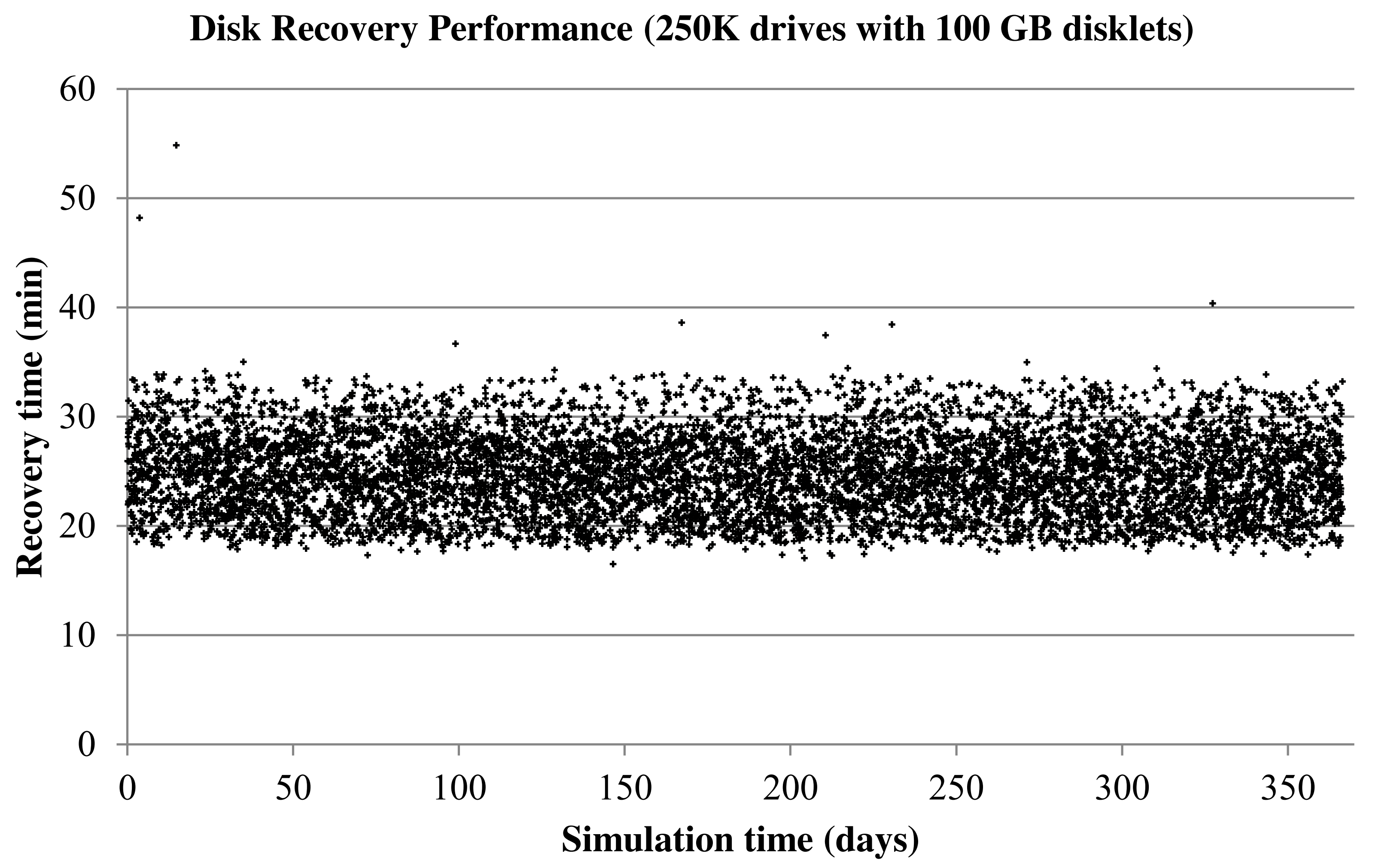
- Hard drives had 1 TB and a bandwidth of 128 MB/s
- Annual failure rate of 4%
  - Failure distribution follows a Poisson process

Experiments

- Reliability analysis as system scales
  - 250,000 drives, 500,000 drives and 1,000,000 drives
- Recovery Performance
  - We run the experiment at multiple disklets sizes
- Reliability with high failure rates
- Simulation noise

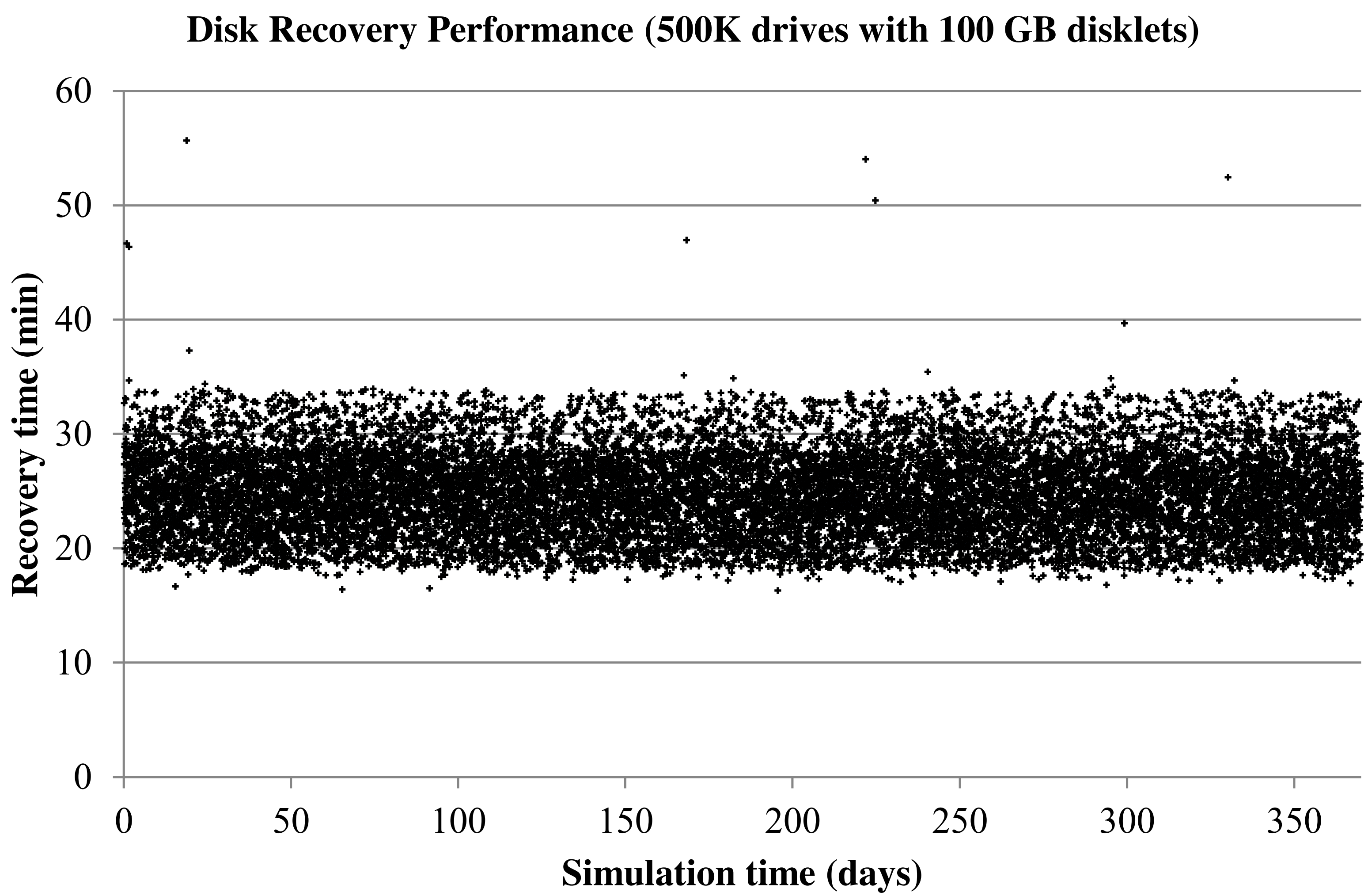
IMPACT OF SCALE ON RECOVERY

250,000 drives



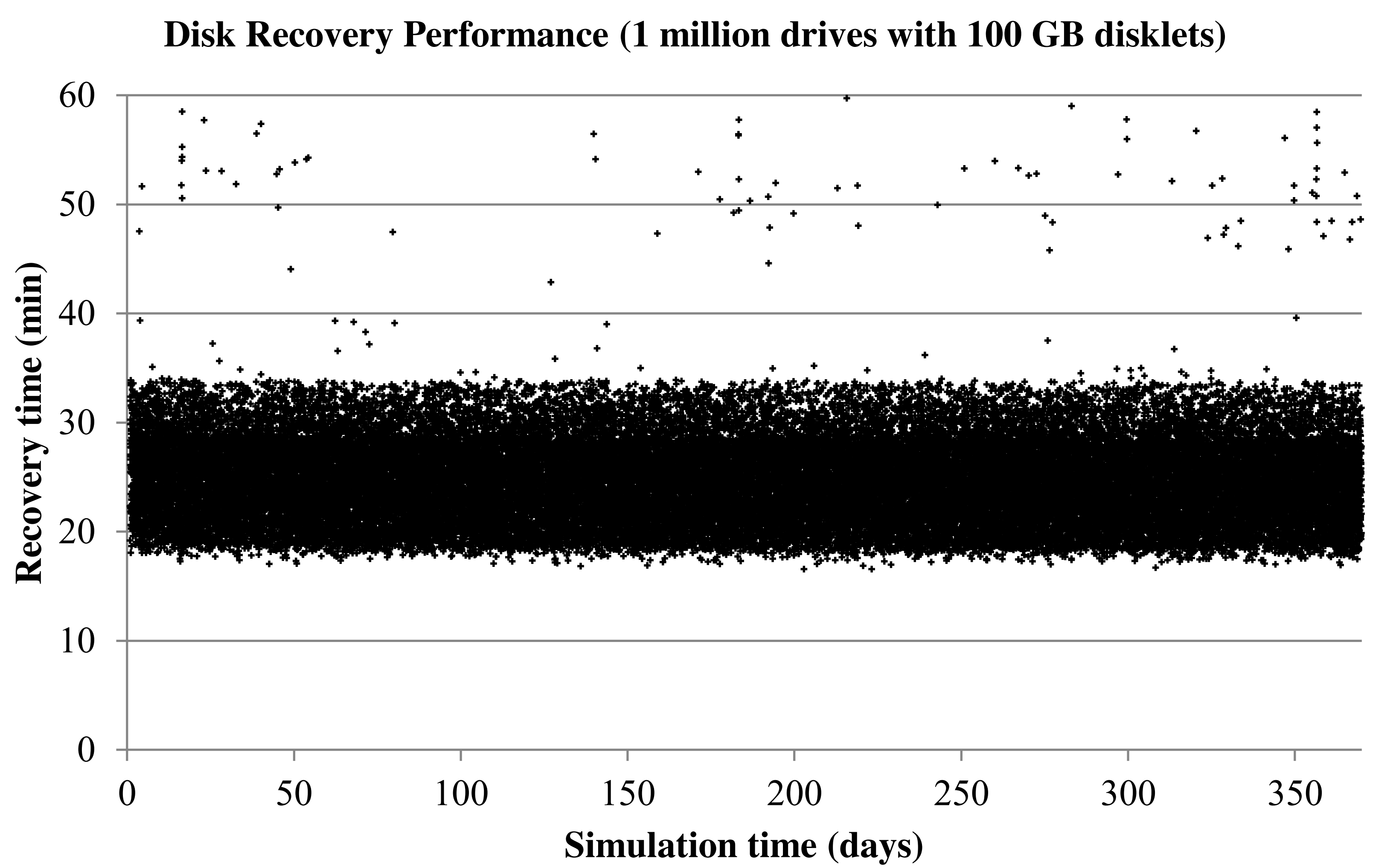
* 4% AFR running at 1 second = 10 minutes

500,000 drives



* 4% AFR running at 1 second = 10 minutes

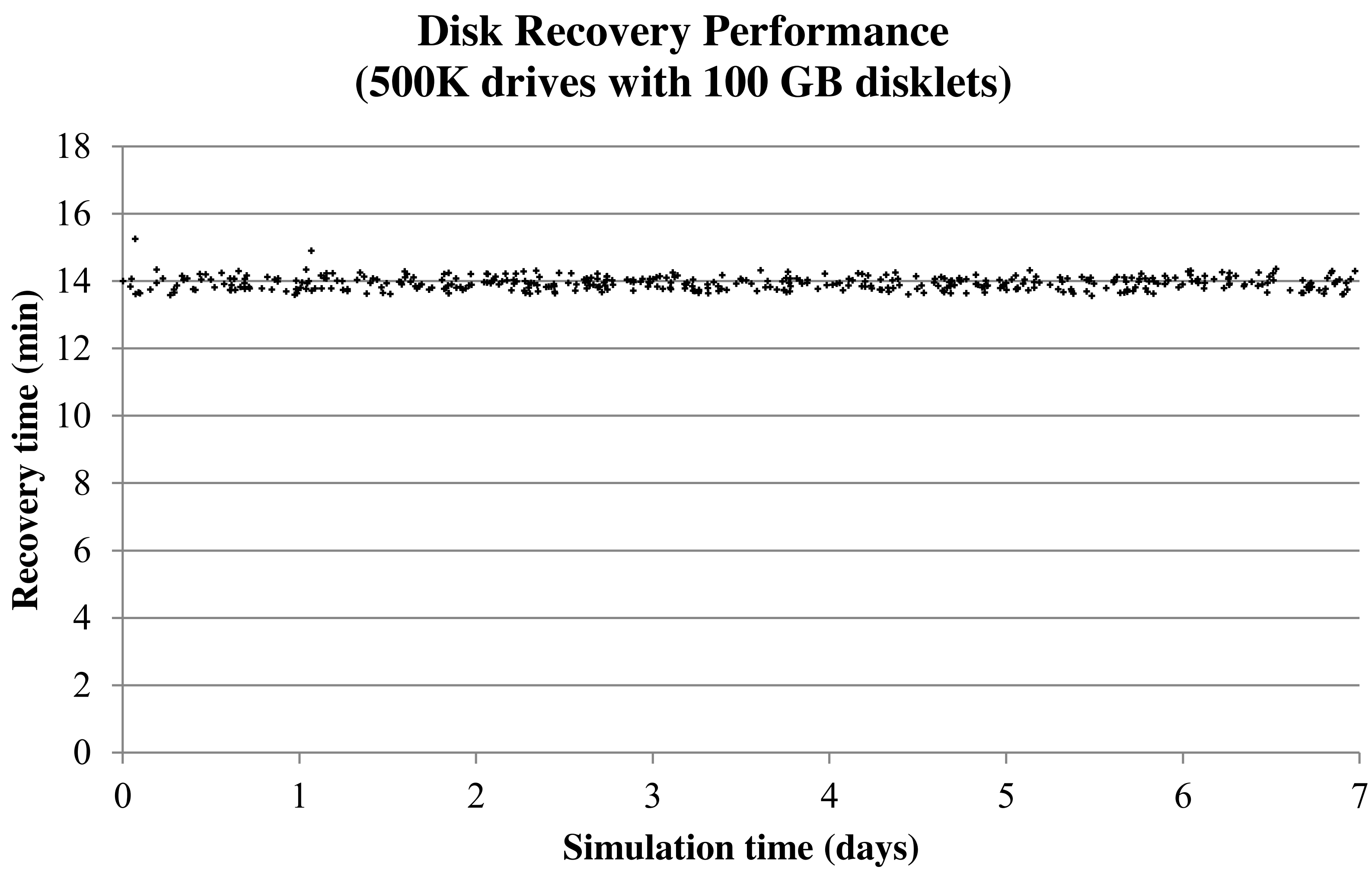
1 Million drives



* 4% AFR running at 1 second = 10 minutes

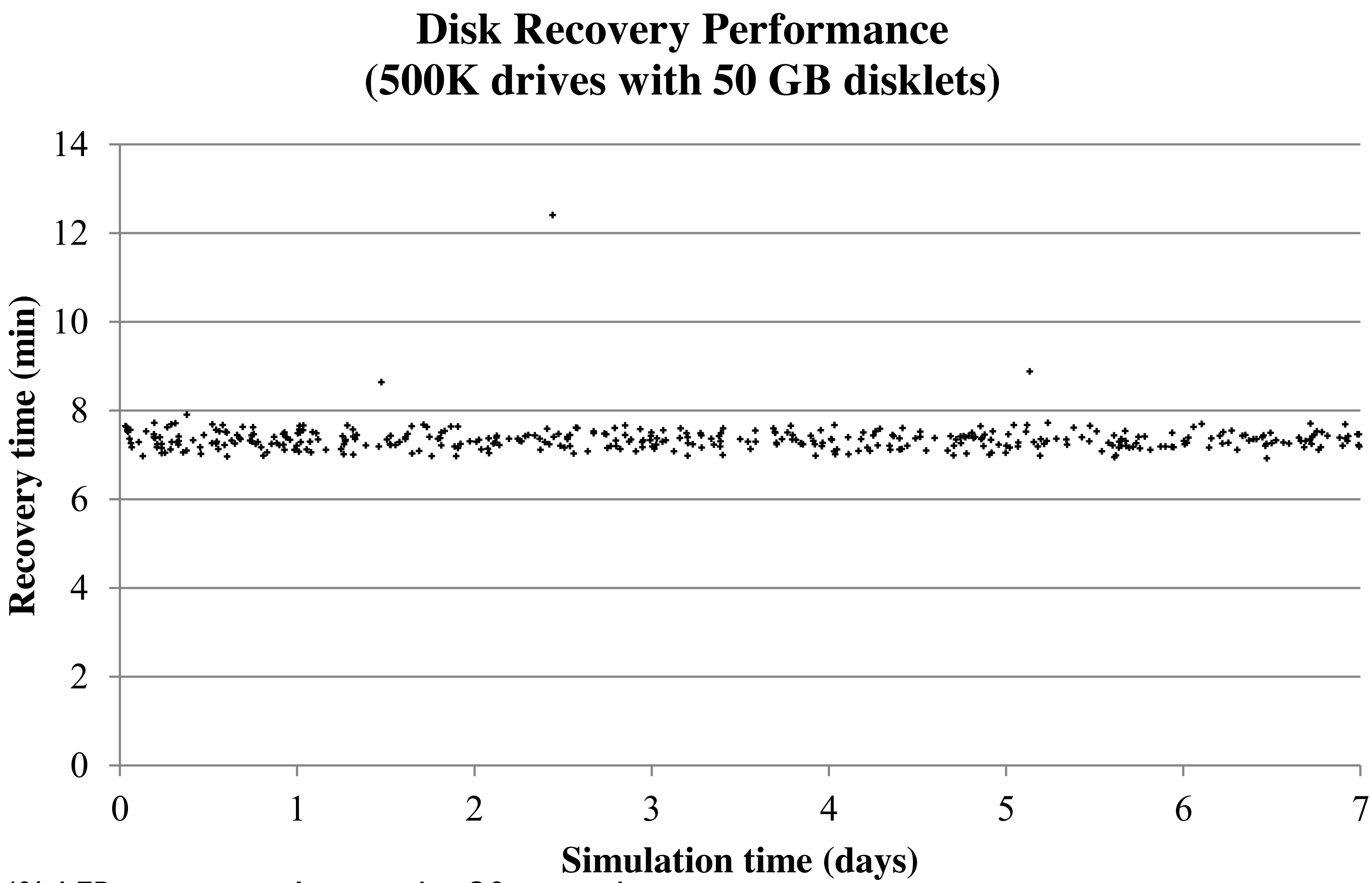
IMPACT OF DISKLET SIZE ON RECOVERY

500k drives with 100 GB disklets



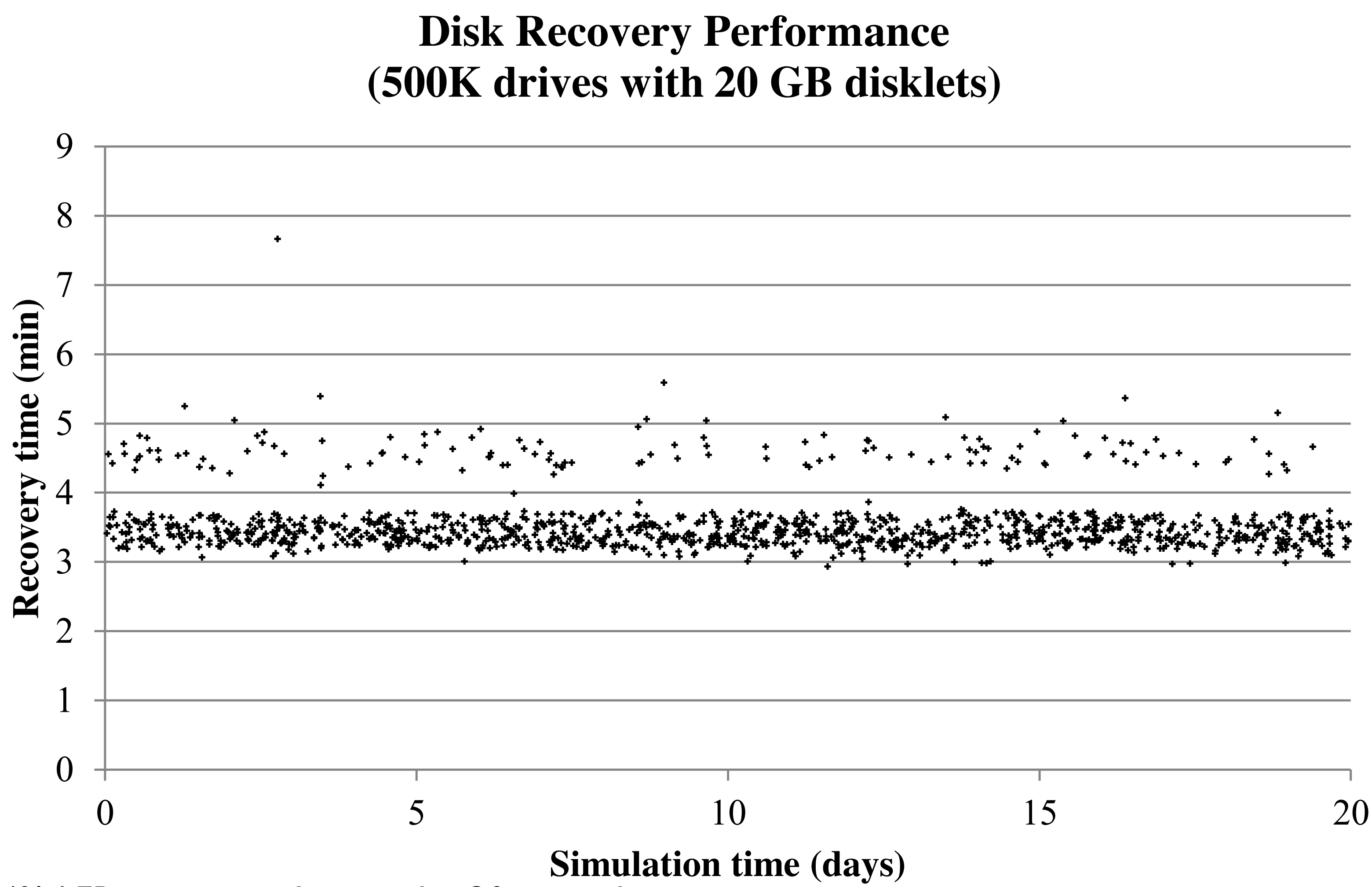
* 4% AFR running at 1 second = 30 seconds

500k drives with 50 GB disklets



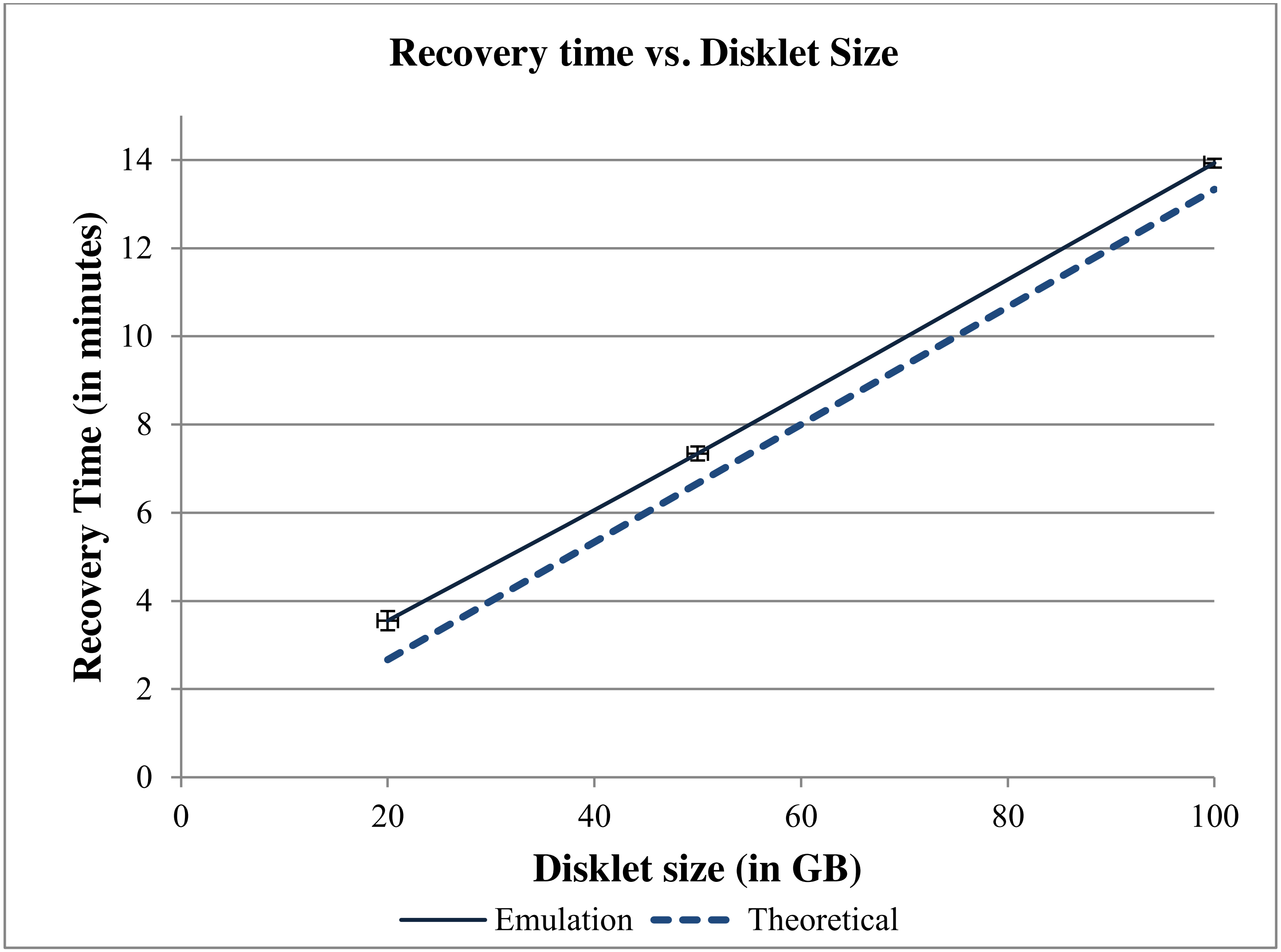
* 4% AFR running at 1 second = 30 seconds

500k drives with 20 GB disklets



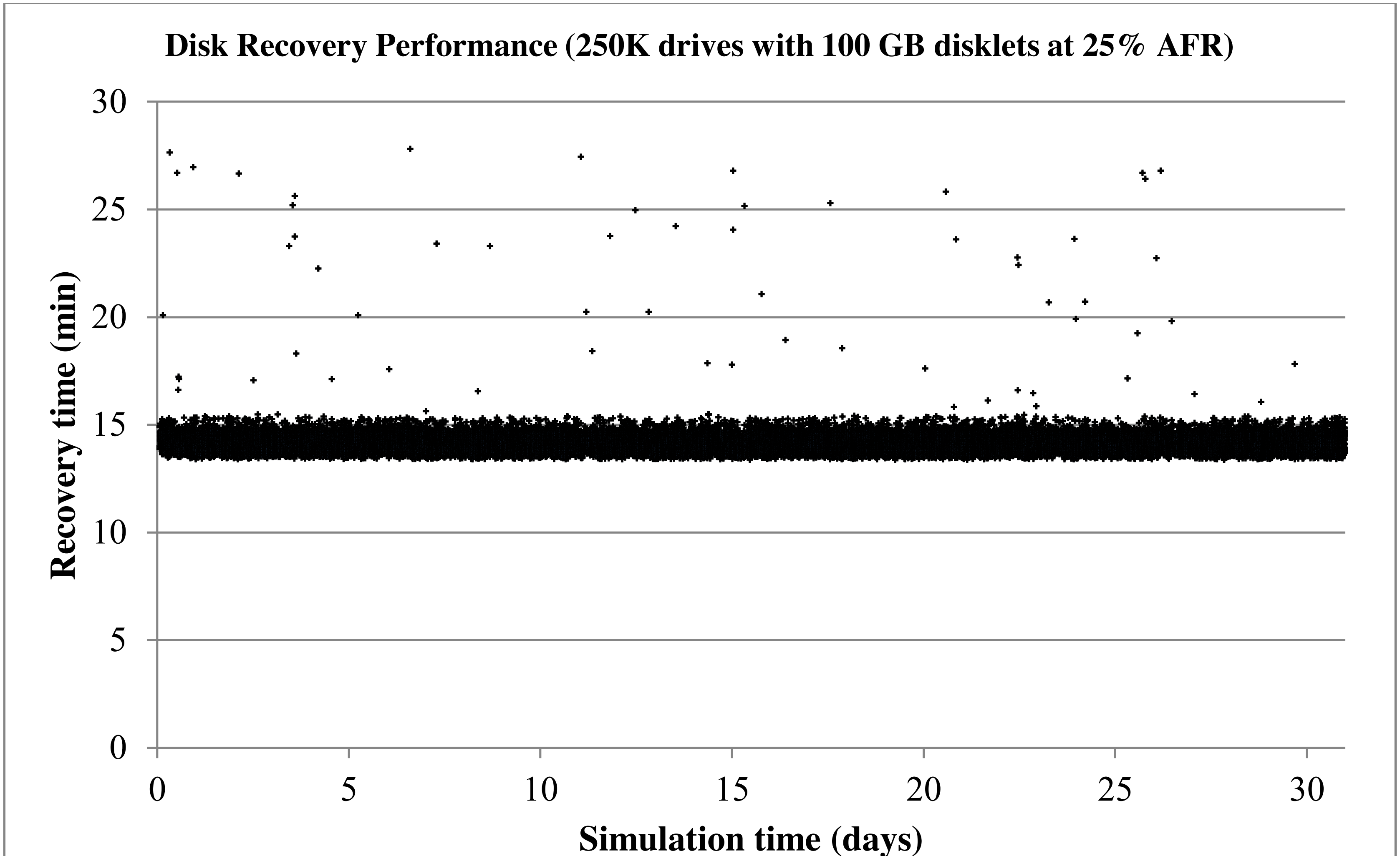
* 4% AFR running at 1 second = 30 seconds

Recovery Profile



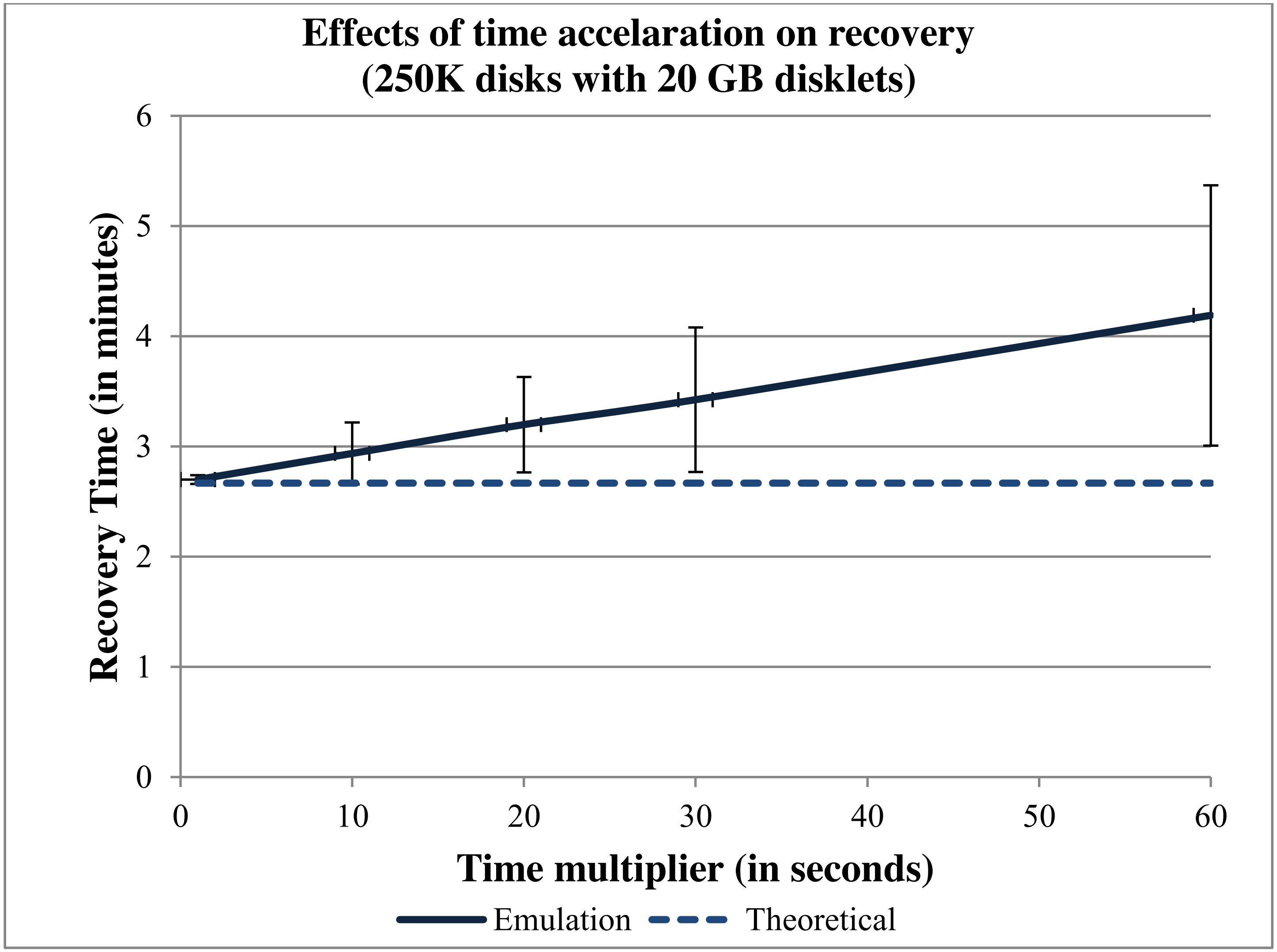
EFFECTS OF OTHER SYSTEM PARAMETERS

High failure rates



Running at 1 second = 1 minute

Simulation Noise

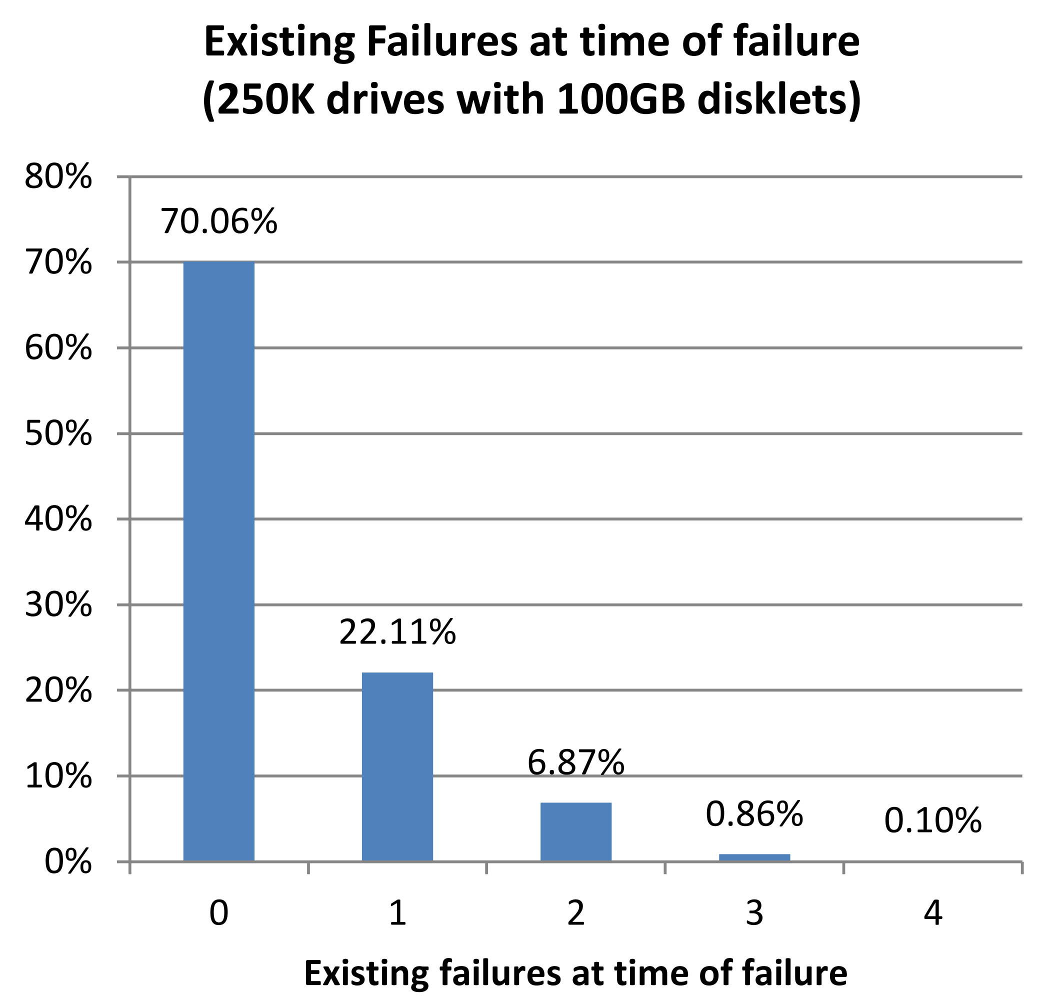


Conclusions

- Two failure tolerance based on XOR
  - **Fast** algorithms
  - Suboptimal but good enough
- **Greater reliability** than 8+2 erasure codes.
- **Greater reliability** than Triplication **without the storage overhead**.
- **Scales** to over **1 million drives**
- **Can sustain high failure rates**

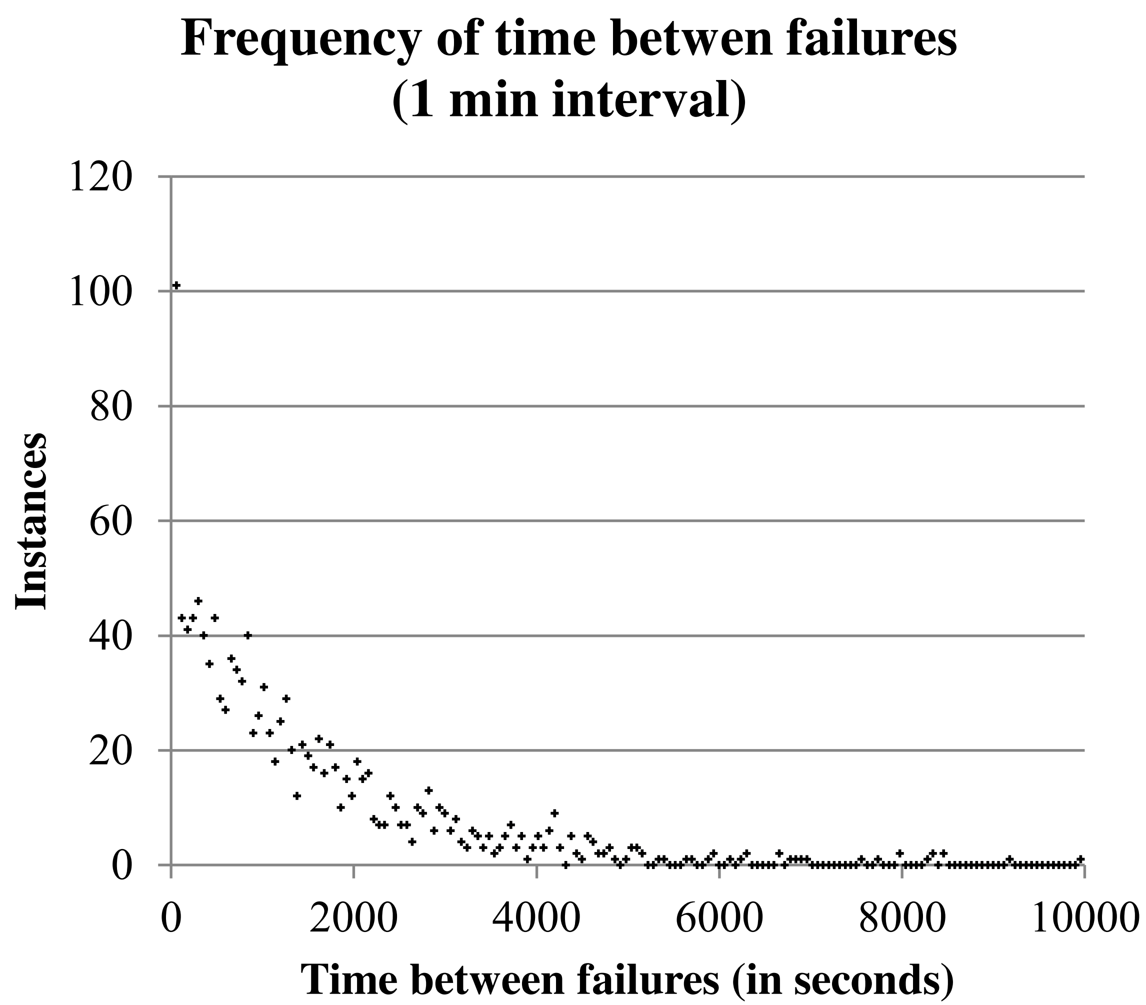
QUESTIONS?

Existing Failures – 250K drives



* 4% AFR running at 1 second = 10 minutes

Distribution of time between failures



* 4% AFR running at 1 second = 30 seconds