# Reliable Distributed Storage

Presented by: James Larkby-Lahet

Including material from ‘Outshining Mirrors: MTTDL of Fixed-Order SSPiRAL Layouts’ by Ahmed Amer Jehan-François Paˆris, Thomas Schwarz, Vincent Ciotola and James Larkby-Lahet

# Why We Want Reliability

- Most modern systems are comprised of many components (Supercomputers - nodes, Virtualized Storage - disks)
- Without redundancy, the System likelihood of failure is the *sum* of the component's likelihoods of failure
- As systems grow, they become *more* unreliable

# What Needs to be Reliable?

- Storage is persistent ‘state’
- without statefulness, all systems are trivially ‘fault-tolerant’
  - webservers can drop out and the client will retry a request
- So in some sense, storage fault-tolerance is a redundant phrase

# Redundancy codes

- Create redundant information about relationships between data: Parities
- Used for Communication and for data Storage
- Error codes for noisy channels
- Erasure codes for stop-fault models

# Optimal Codes

- N of M
- Hard to compute - multiplying big matrices
- data may be ‘scrambled’ in with parity, requiring a decode step even without a fault

# RAID - Special Optimal Codes

- RAID 4,5 - XOR based parity
- RAID 6 - additional parity, using Reed-Solomon code
- RAID DP
- Triple Redundant Parity -- The free lunch limit, I think

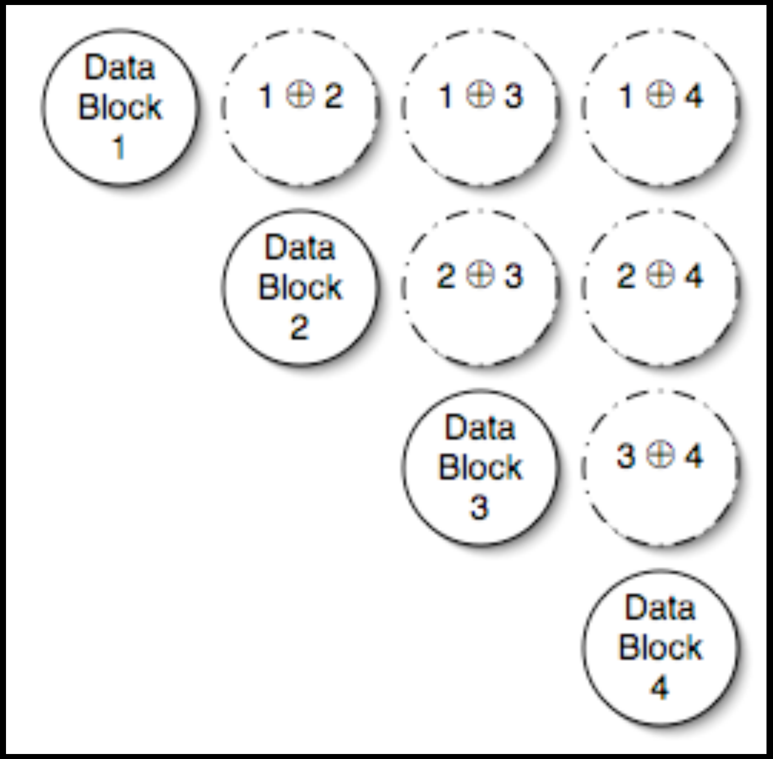
# What about Mirroring?

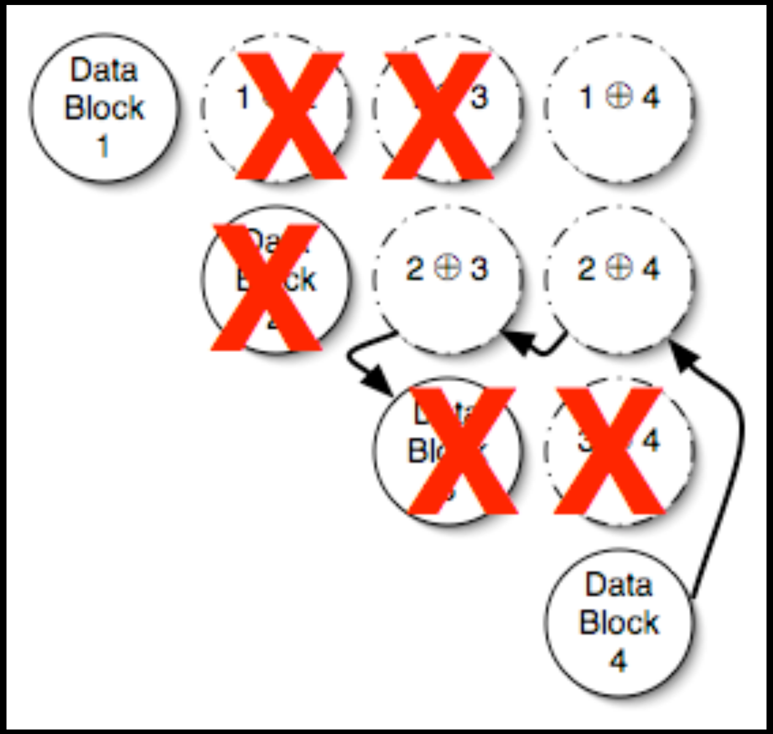
- How Reliable are 3 disks with 3 mirrors?

# SSPiRAL

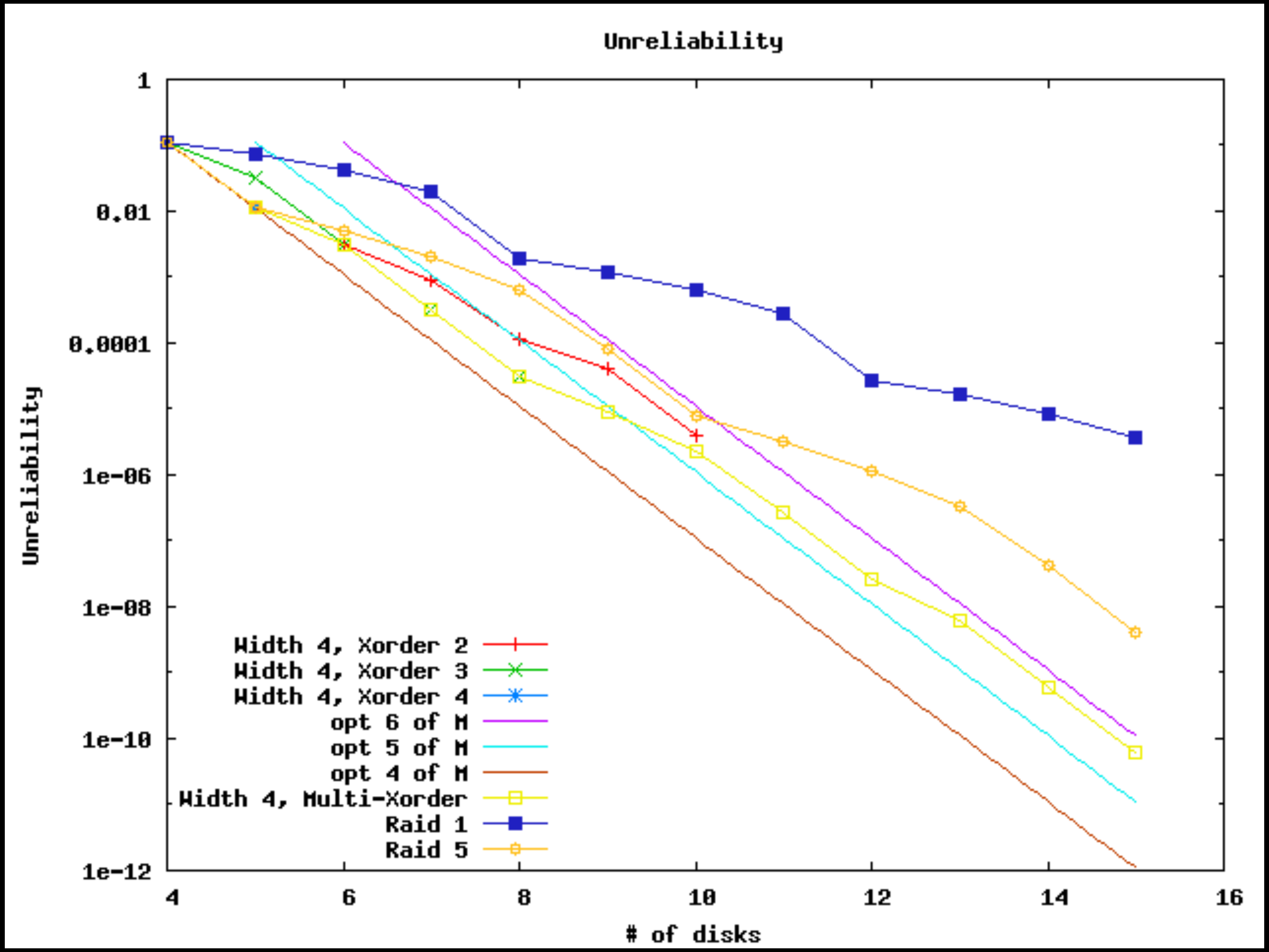
- Survivable Storage using Parity in Redundant Array Layout
- Mirroring is not the best way to provide arbitrary reliability
- Xor Based Parities
- two or more data blocks combined in each Parity block

# Example SSPiRAL arrays





# Unreliability



# Which Nodes should we Use?

- There are 2^N-1 possible nodes
- Not all nodes improve reliability equally
- We need a way to evaluate different node’s contributions, which changes relative to the other nodes in the system

# Simulation 1

- Brute Force!
  - generate all possible array layouts (GBs in size for N=12)
  - for each, recursively kill nodes until data loss occurs, in all possible combinations
  - also need a fast way to test for ‘liveness’
- naively (2^N-1)! steps

# Simulation II

- In the previous approach, killing a node involves solving for the reliability of a sublayout
- We can work from the bottom up (all layouts of size N) and solve for the reliability of all layouts (for a given N) simultaneously

# Simulation III

- Symmetries that can be eliminated
- What is the difference between 1,2,4,1^2 and 1,2,4,1^4
    - Just names for the same thing
    - still have to permute name to find
- Open problem: are there other, more complex symmetries?

# An Aside: Liveness Testing

- Binary Decision Diagrams allow efficent storage and worst-case linear-time testing of a boolean function
- we can write a boolean function for data liveness with 2(N-1) boolean variables
- space-time tradeoff, much faster than attempting to recover data by brute force

# Modeling Reliability Mathematically

- Markov Chains - a collection of states and transitions between them
- lambda - likelihood of a disk failure
- mu - probability of disk repair

# Markov Chains

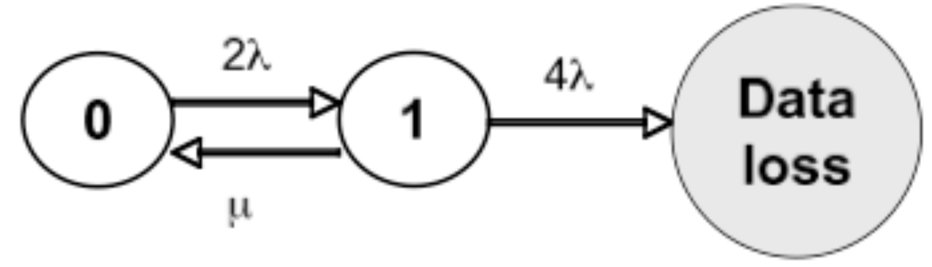


Figure 5: *Single pair of mirrored disks.*

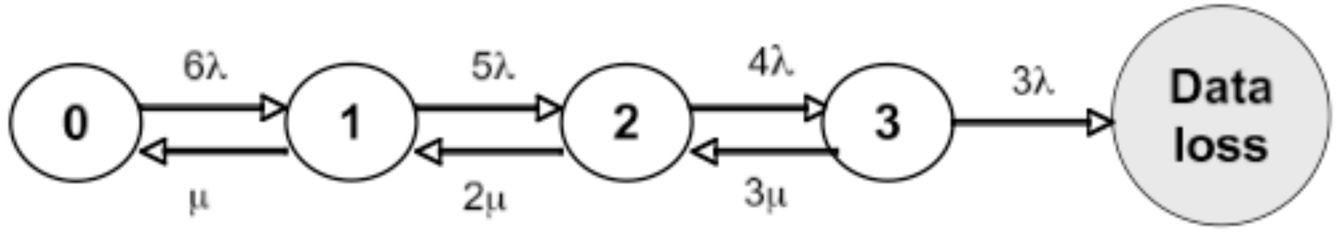


Figure 4: *3-out-of-6 array.*

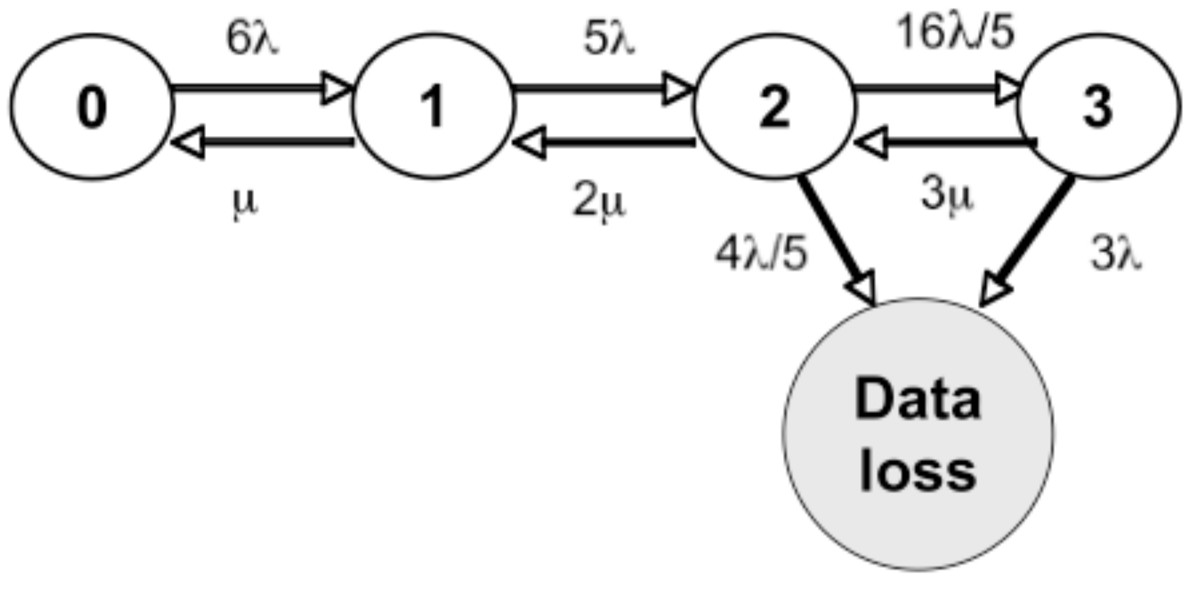


Figure 3: *3+3 disk SSPiRAL array.*

# Solving for MTTDL

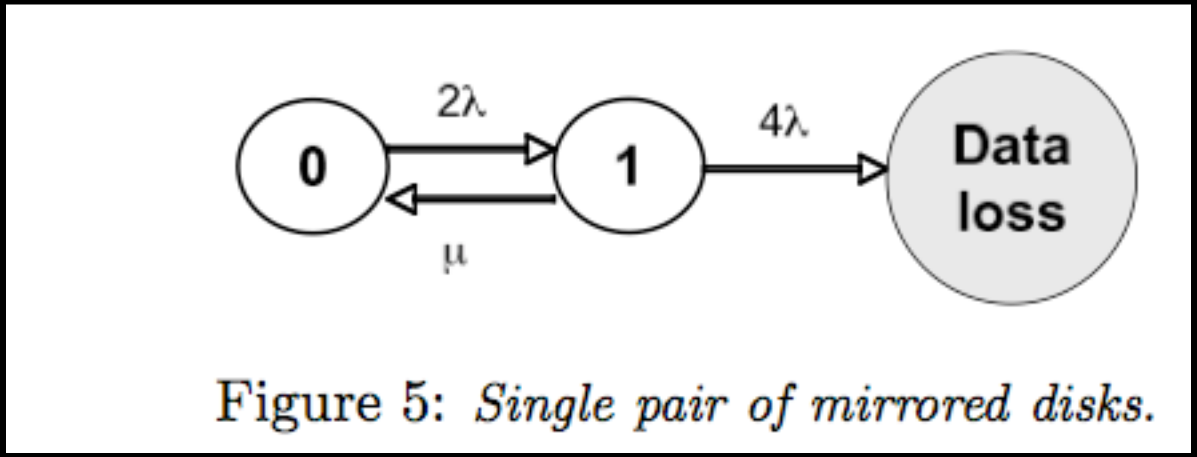
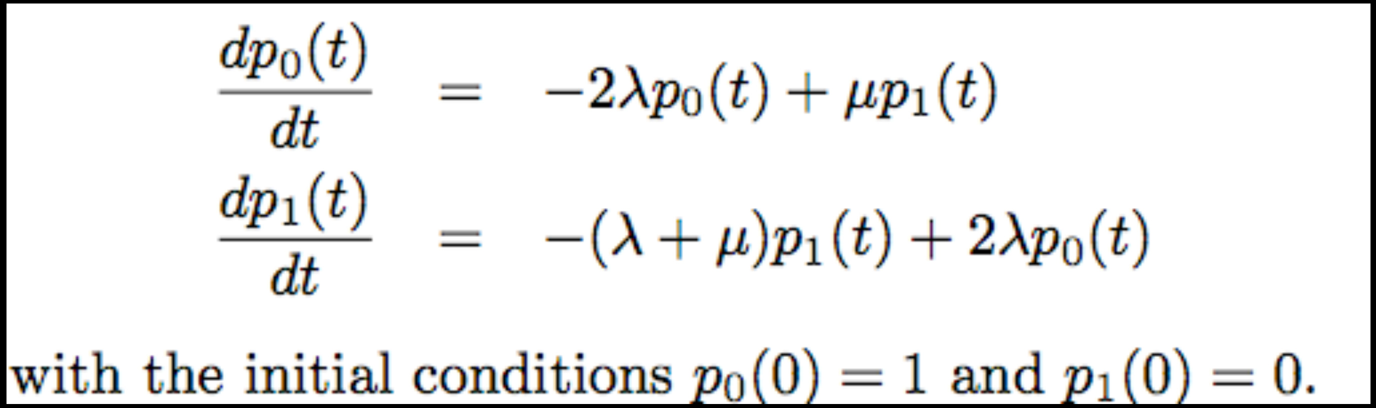


Figure 5: *Single pair of mirrored disks.*



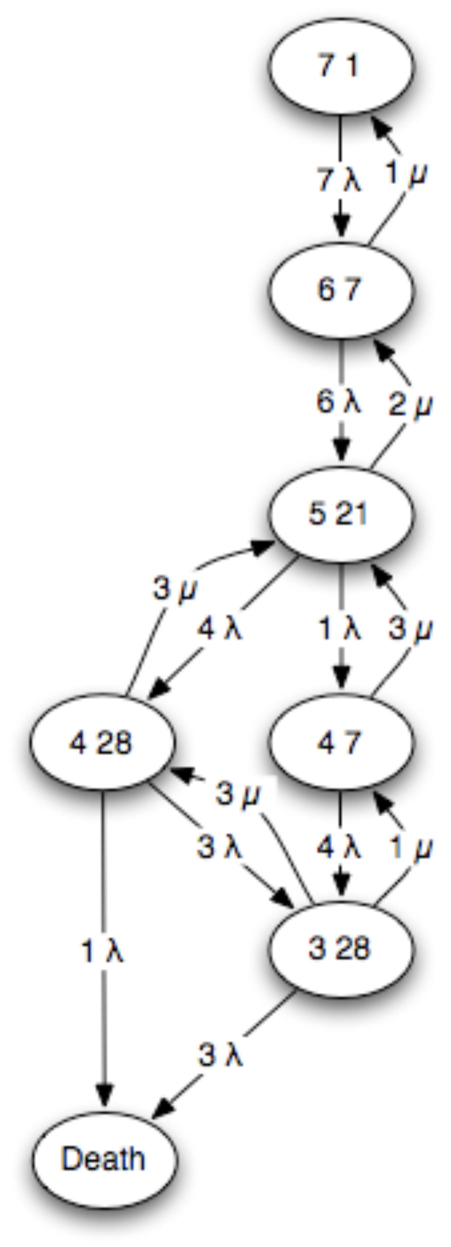
$$
\begin{aligned}
\frac{dp_0(t)}{dt} &= -2\lambda p_0(t) + \mu p_1(t) \\
\frac{dp_1(t)}{dt} &= -(\lambda + \mu) p_1(t) + 2\lambda p_0(t)
\end{aligned}
$$

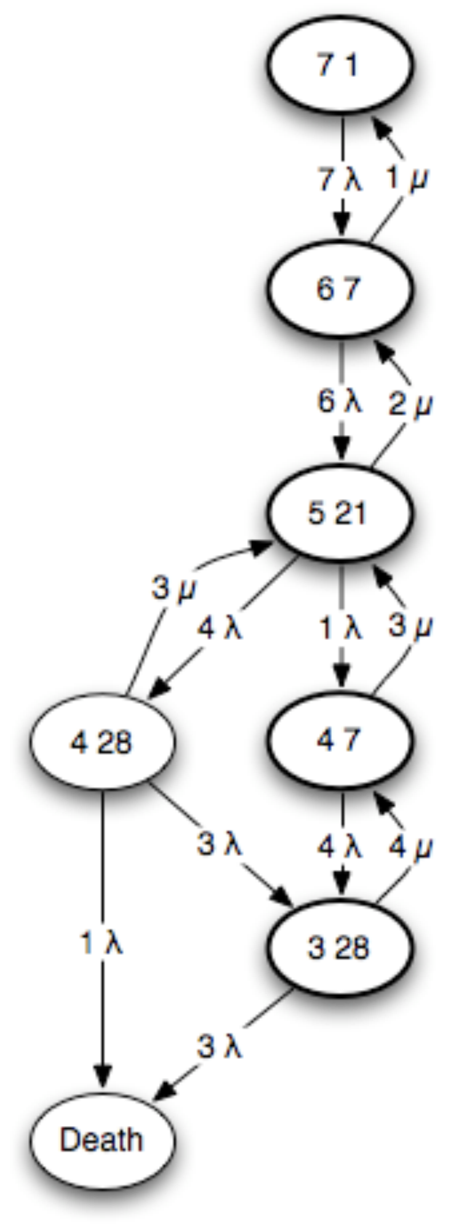
with the initial conditions $p_0(0) = 1$ and $p_1(0) = 0$.

# Making SSPiRAL ‘optimal’

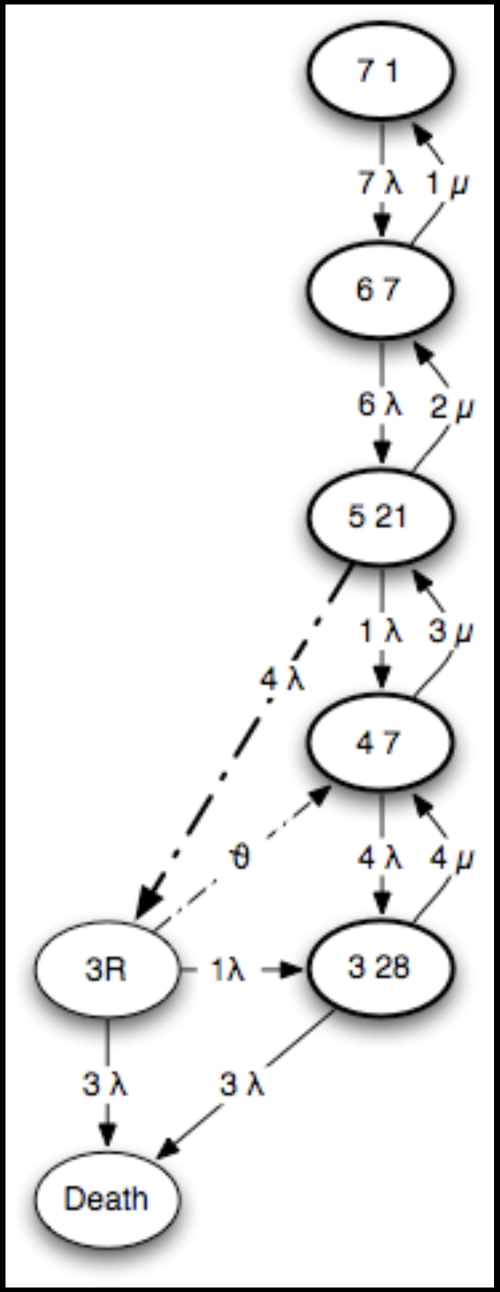
- Critical nodes are ones that can lead to data loss
- can we avoid critical nodes by reconfiguring?

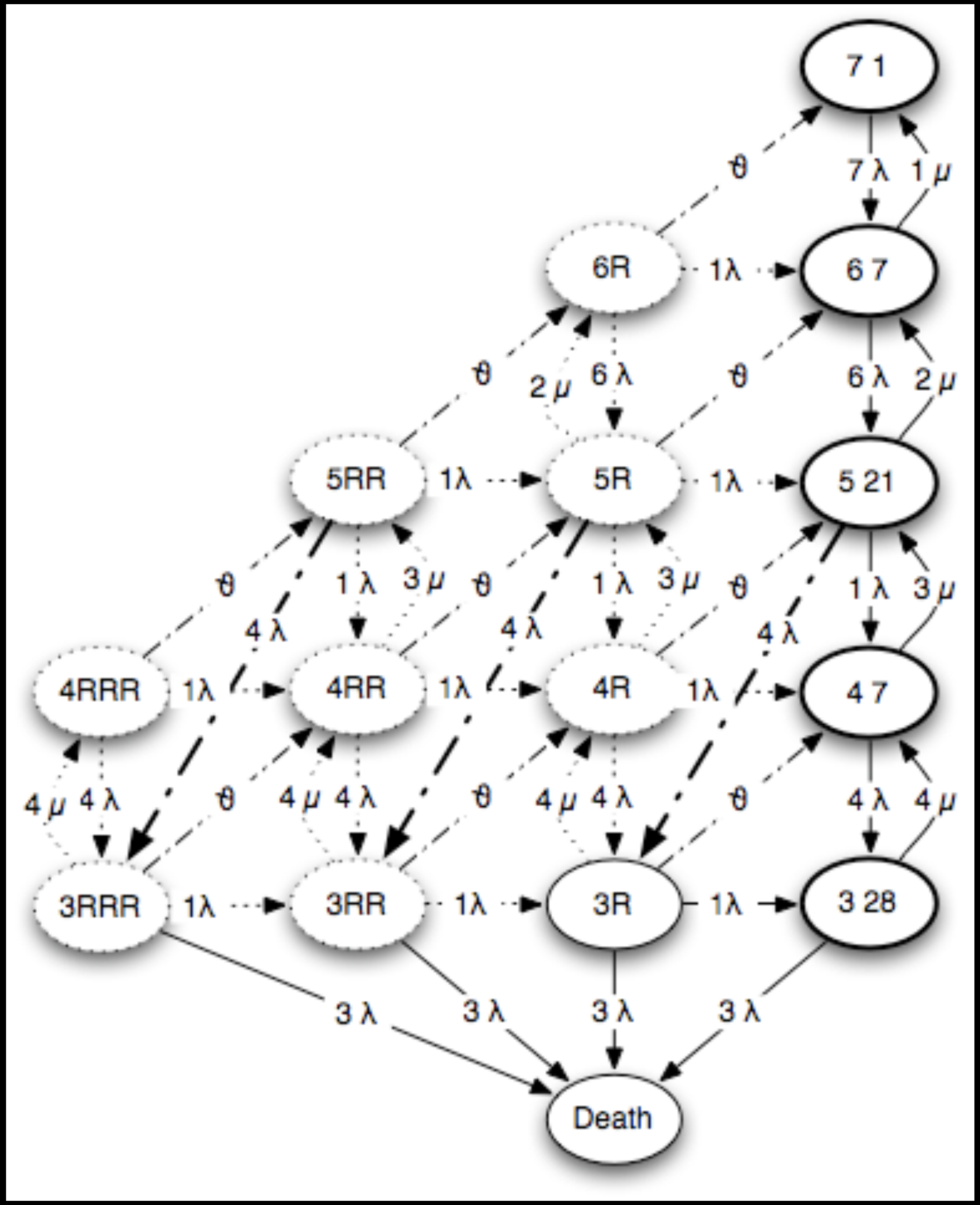
# Best Recovery





# Rebuild States





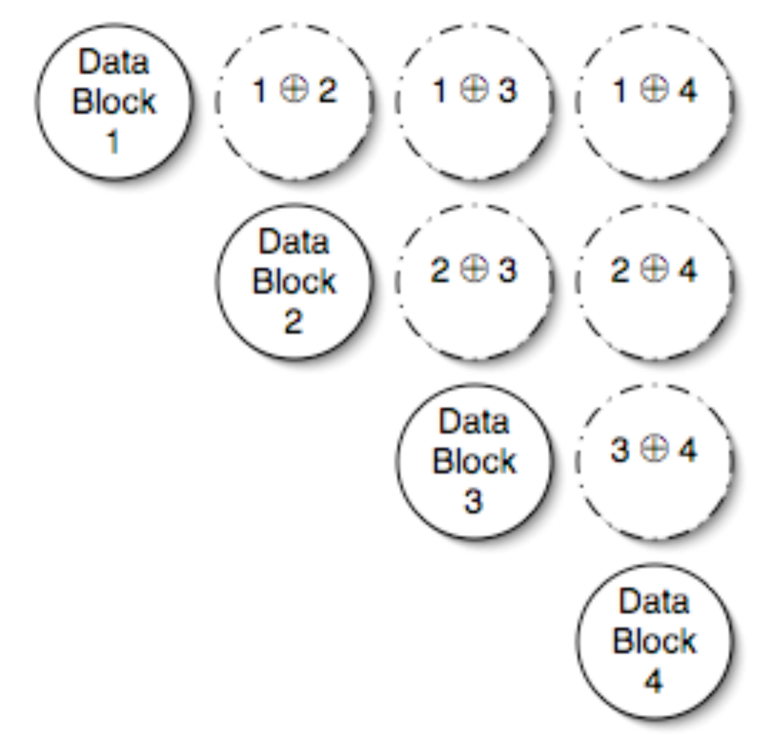
# Problems

- Most Reliable states are not always parents of each other
- Simulation is required to discover layout reliabilities and equivalences
- O(2^N-1) :(
- can we find symmetries, reasons some layouts have the same reliability?

# Using Asymmetry

- Some Nodes are more valuable than others
- Some hardware is more reliable than others
- Should we map more important nodes to more reliable hardware?

# load balancing


Data Block 1
1 ⊕ 2
1 ⊕ 3
1 ⊕ 4
Data Block 2
2 ⊕ 3
2 ⊕ 4
Data Block 3
3 ⊕ 4
Data Block 4