

# Designing Dependable Storage Solutions for Shared Application Environments

Shravan Gaonkar, *Kimberly Keeton, Arif Merchant, William H. Sanders,*  
[DSN 2006]

Presenter: Shahin Kamali

[s3kamali@cs.uwaterloo.ca](mailto:s3kamali@cs.uwaterloo.ca)

David Cheriton School of Computer Science

University of Waterloo

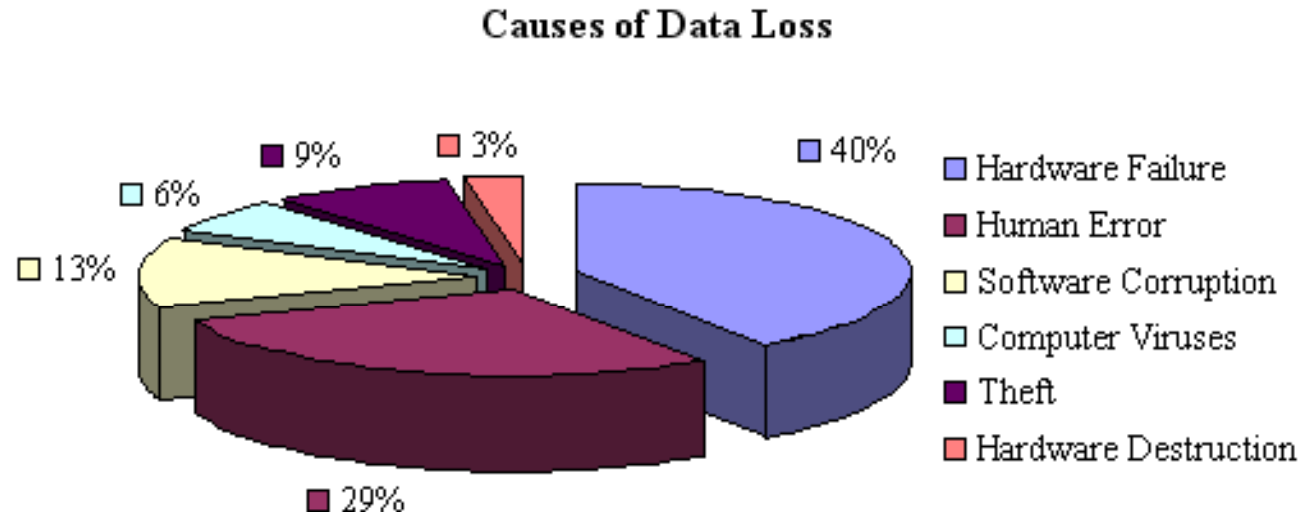


# Road Map

- Background
  - Failure causes and remedies
  - Dependable storage systems
- Problem statement
  - Shared application environments
- Solution technique
  - Heuristic approach
- Experimental results
- Conclusion and feature works

# Data Loss

- Data can be lost... It is bad !



Author's estimates based on data from Safeware, The Insurance Agency, Inc., "2000 Safeware Loss Study," 2001; and ONTRACK Data International, Inc., "Understanding Data Loss," 2003.

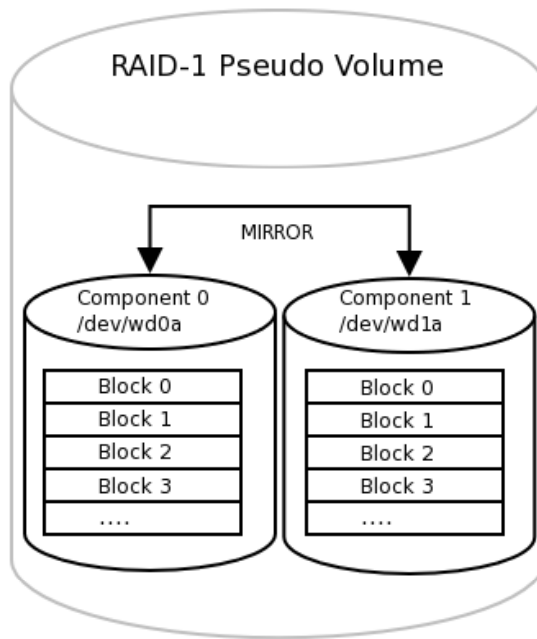
From <http://gbr.pepperdine.edu/033/dataloss.html>

# Data Protection Techniques

- RAID technology
- Point in time copying
- Periodic backup
- Remote mirroring

# RAID: Redundant Array of Inexpensive Disks

- replicate data among multiple hard disk drives



<http://www.netbsd.org/docs/guide/en/chap-ri.html>



[http://www.orbitmicro.com/support/resources/raid\\_tutorials.html](http://www.orbitmicro.com/support/resources/raid_tutorials.html)

- Protects data from internal hardware failures

# Point in Time Copies

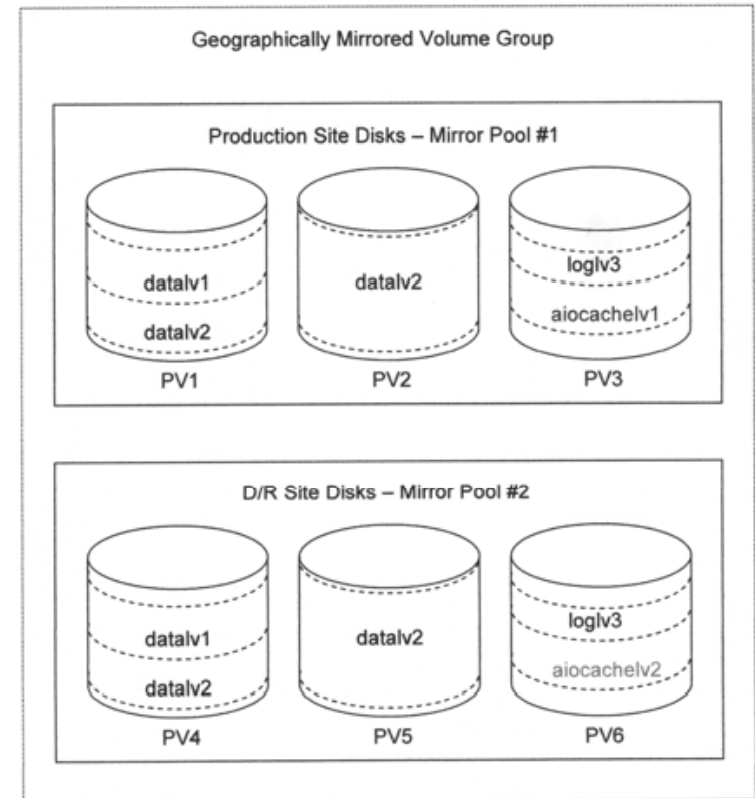
- Ex: Snapshots
- Consistent data in the case of buggy software or viruses
- Space efficient
- Internal → fast recovery

# Backups

- Create copies on disk or tape
- Cheap !
  - Store more data (older versions)
- Can be geographically distributed
  - Data vaults
  - Protect against theft or natural disaster
- Not fast
  - Data is not fresh

# Remote Mirroring

- Mirror an exact copy of data
  - Synchronous vs. Asynchronous
- Failover vs. Reconstruct
- Geographically distributed
- Expensive layout !

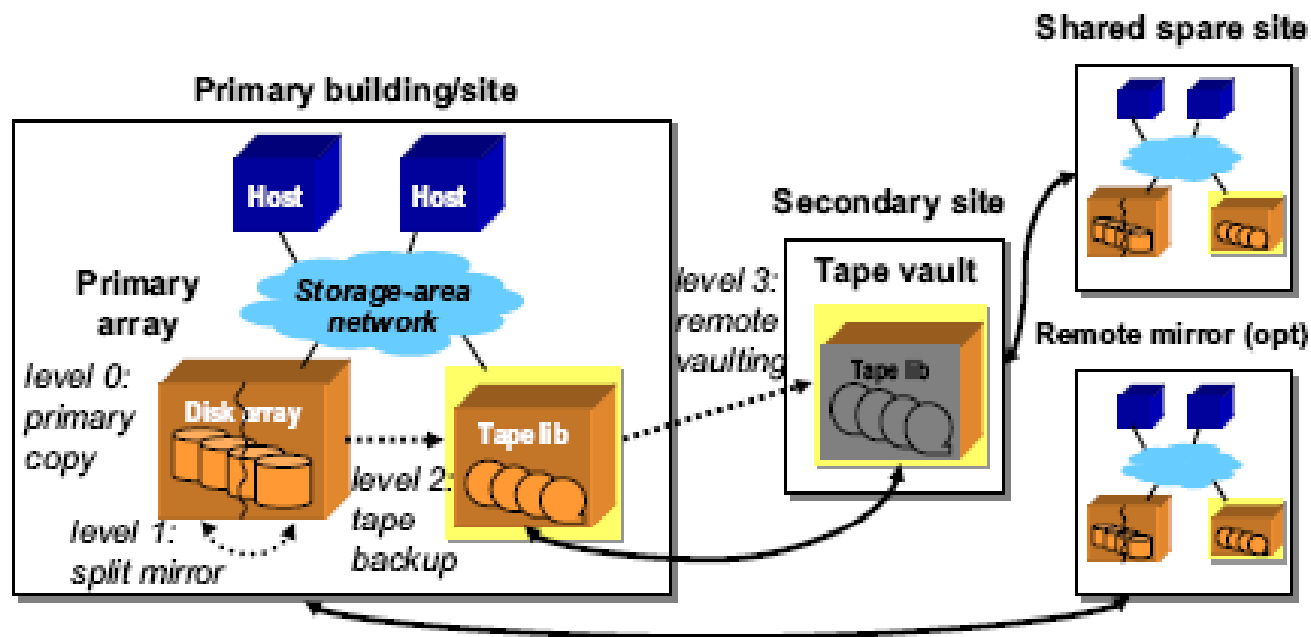


# Data Protection Techniques

- RAID → ex: internal software design
- Point in time copies → ex: buggy software
- Backups → ex: natural disaster [cheap]
- Mirroring → ex: natural disaster [fast recovery]
- Key Idea:
  - Use a combination based on application requirements

# Combination of Techniques

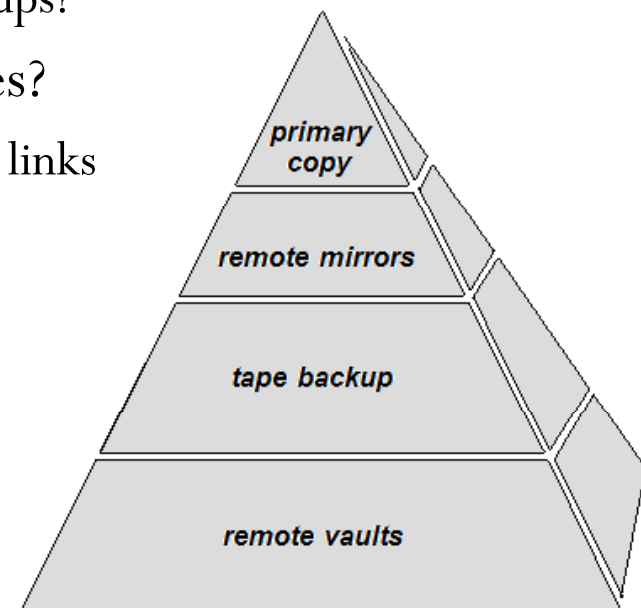
- A hierarchy
- Ex:



K. Keeton and A. Merchant. A framework for evaluating storage system dependability. In *Proc. 2004 Intl. Conf. on Dependable Systems and Networks (DSN)*, pages 877–886.

# Problem statement

- Given an application
  - Which combination of techniques?
  - What configuration parameters?
    - Ex: how often take snapshots and backups?
  - How to provision physical resources?
    - Ex: disk arrays, tape libraries, network links
- Goal: Minimize “overall cost”



# Cost Definition

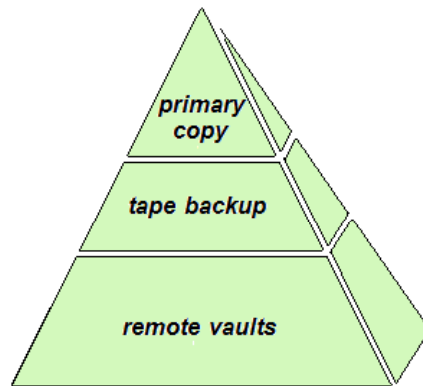
- When a failure happens:
  - Data is not available for a while → Outage penalty rate (\$/hr)
    - Ex: Company web service: 5 M \$/hr
    - Consumer banking: 5 K \$/hr
  - Recovered data is not fresh → Recent loss penalty rate
    - Ex: Company web service: 5 K \$/hr
    - Consumer banking: 5 M \$/hr
- Outlay cost
  - Devices have different bandwidth and capacity

# Previous Work

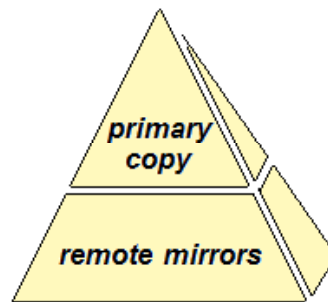
- Automatically design a system which uses a single technique for a **single** application
- Multiple application environment
  - Naïve heuristics:
    - Categorize applications (Ex: gold, silver, bronze)
    - Chose between a fixed set of techniques
  - Either too expensive or under-provision

# Current Work

- Automatically design a system for **Multiple** application
- Challenges:
  - Very large design space
  - Contention for the resources
  - Different design decisions

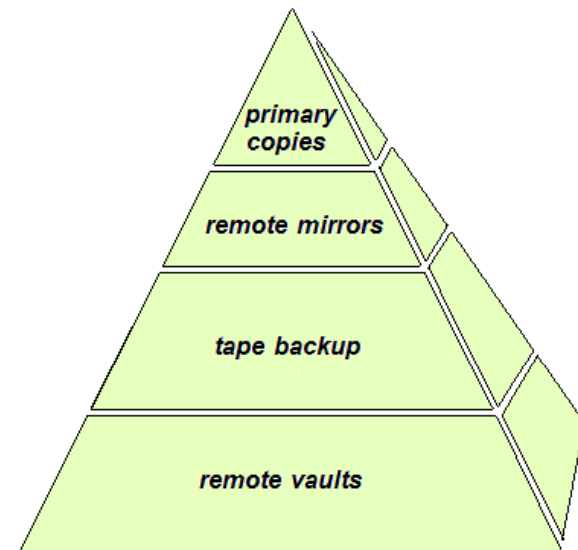


App1



App2

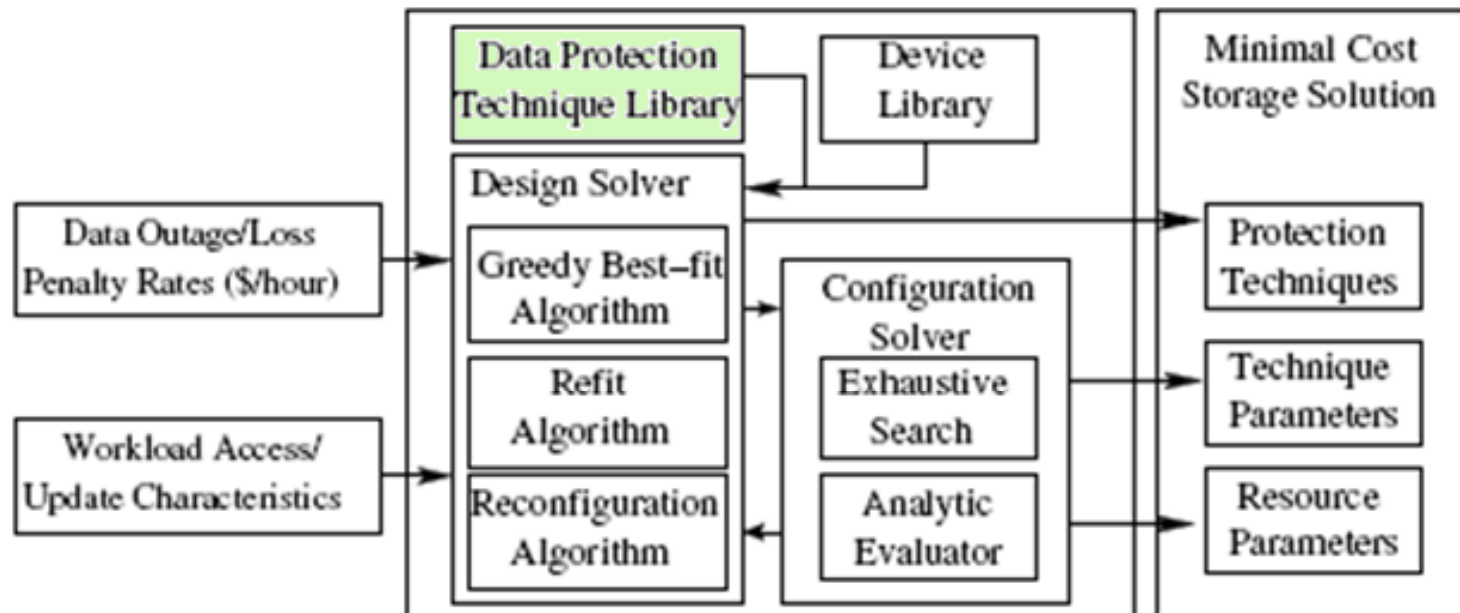
$\geq$  (cost)



Shared App

# Solution Technique

- Design solver: heuristic for finding a ‘good’ design
- Configuration solver: set configuration parameters and derives layout



# Data Protection Technique Library

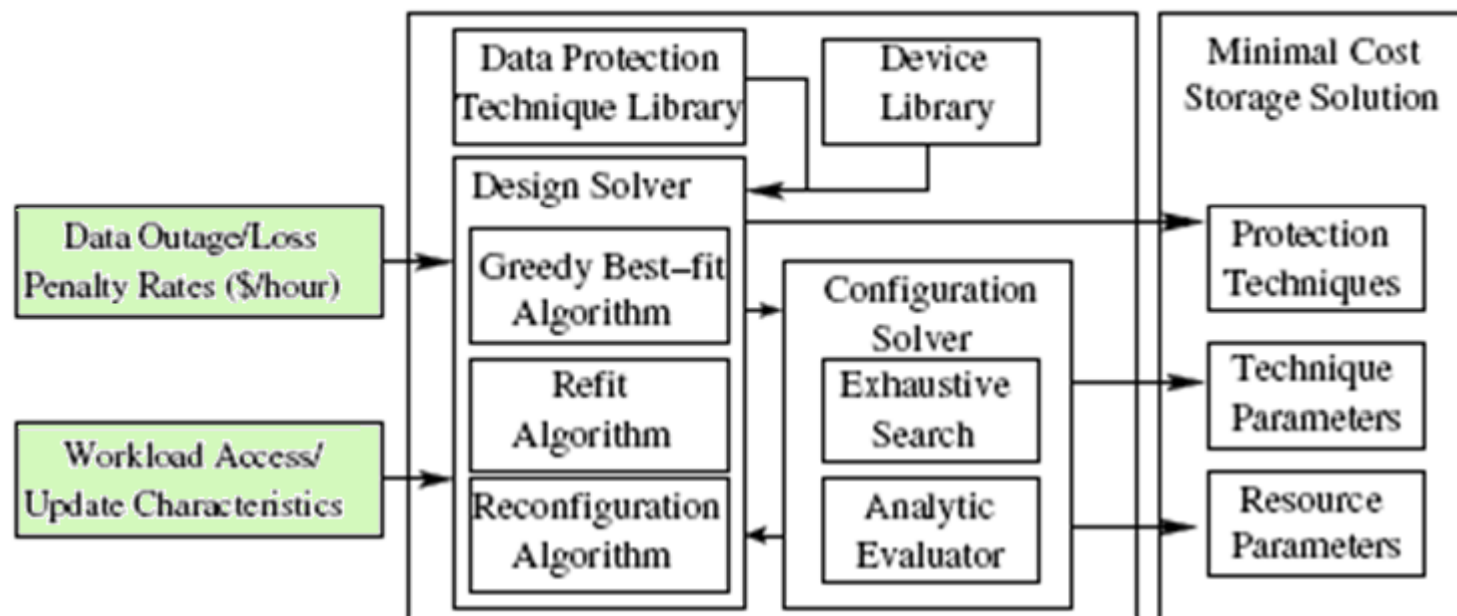
- Output of design solver is among these techniques

Data protection technique type	Reconstruct (R) or Failover (F)	Category	Level 1 snapshot (S) or mirror (M)		Level 2 tape library in days		Level 3 vault in days		
			accWin	propWin	accWin	propWin	accWin	propWin	
Synchronous mirror with backup	Failover	Gold	M S	0.5 min 12 hr	n/w tape	7 days	tape	28 days	1 day
Synchronous mirror with backup	Reconstruct	Silver	M S	0.5 min 12 hr	n/w tape	7 days	tape	28 days	1 day
Asynchronous mirror with backup	Failover	Gold	M S	10 min 12 hr	n/w tape	7 days	tape	28 days	1 day
Asynchronous mirror with backup	Reconstruct	Silver	M S	10 min 12 hr	n/w tape	7 days	tape	28 days	1 day
Synchronous mirror	Failover	Gold	M	0.5 min	n/w				
Synchronous mirror	Reconstruct	Silver	M	0.5 min	n/w				
Asynchronous mirror	Failover	Gold	M	10 min	n/w				
Asynchronous mirror	Reconstruct	Silver	M	10 min	n/w				
Tape backup	Reconstruct	Bronze	S	12 hr	tape	7 days	tape	28 days	1 day

note: "n/w" and "tape" indicate that the propagation delay depends on the available network or tape library bandwidth

# System Input

- Penalty rates
- Workload characteristics



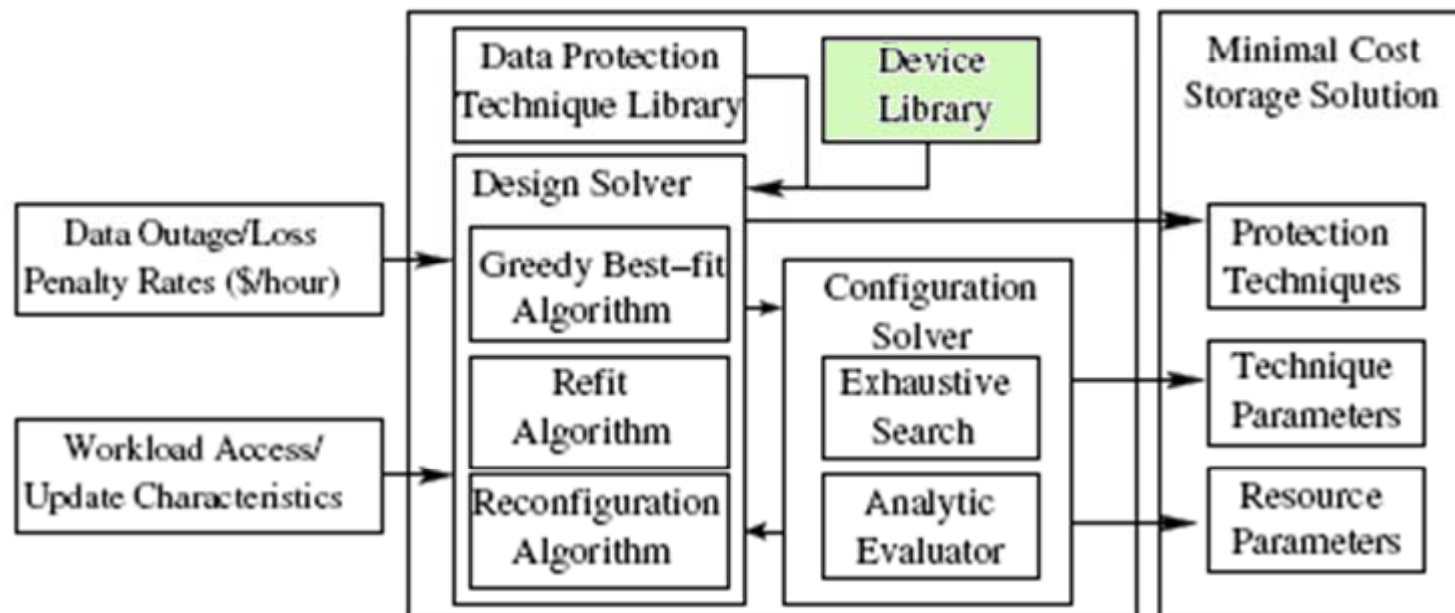
# System Input

- Recall:
  - Outage penalty (data not available)
  - Recent data loss penalty (fresh data lost)

Type	Outage penalty rate (\$/hr)	Recent loss penalty rate (\$/hr)	Data size (GB)	Avg update rate (MB/sec)	Peak update rate (MB/sec)	Average access rate (MB/sec)	Category
Central banking (B): critical, expects zero data loss and data outage loss							
B	\$5M	\$5M	1300	5	50	50	Gold
Company web service (W): high transaction volume, modest recent data loss, zero outages							
W	\$5M	\$5K	4300	2	20	20	Silver
Consumer banking (C): high transaction volume, expects zero recent data loss, modest outages							
C	\$5K	\$5M	4300	1	10	10	Silver
Student accounts (S): student accounts, tolerant to data loss and vulnerability							
S	\$5K	\$5K	500	0.5	5	5	Bronze

# Device Library

- A list of devices embedded in the system



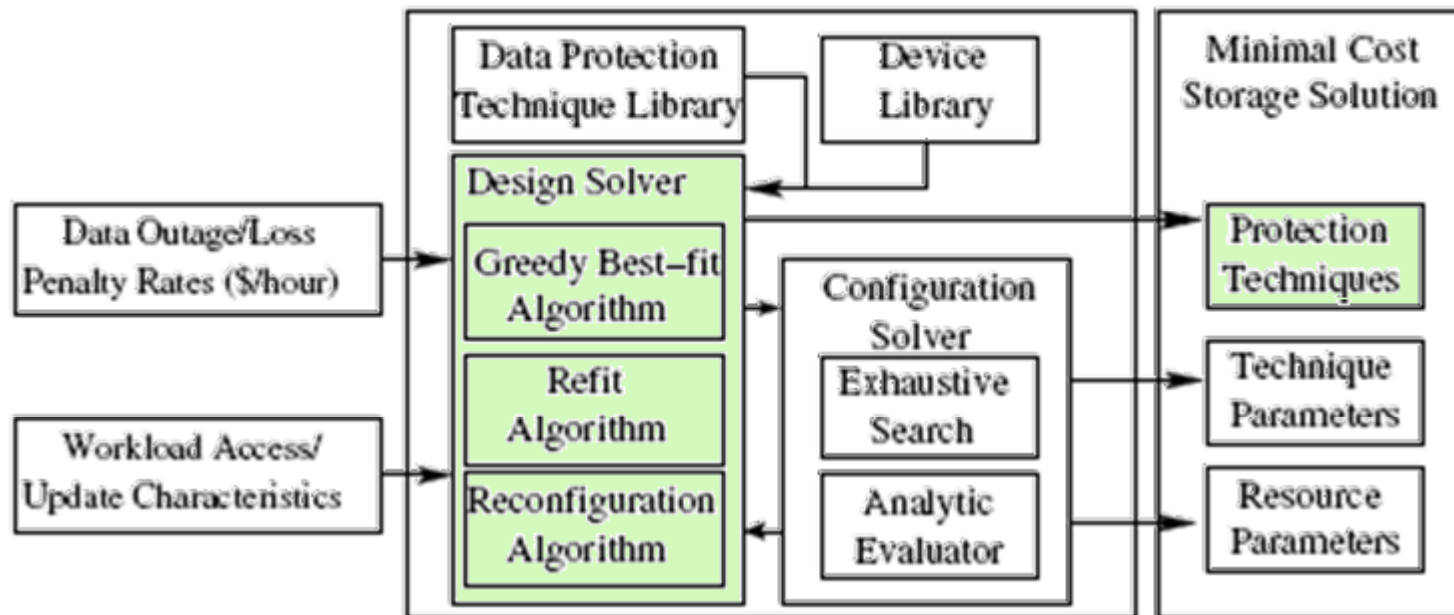
# Device Library

- Available devices to implement design with
- Different cost, bandwidth, and capacity

Resource type	Class	Fixed		Incremental cost (\$)		Total number of		Capacity per unit (GB)	BW per unit (MB/s)
		cost (\$)	BW (MB/s)	per unit capacity	per unit BW	capacity (units)	BW (units)		
Disk array (XP1200)	High	375,000	512	8723		1024		143	25
Disk array (EVA800)	Med	123,000	256	3720		512		143	10
Disk array (MSA1500)	Low	123,000	128	3720		128		143	8
Tape Library	High	141,000	2400	18,400		720	24	60	120
Tape Library	Med	76,000	400	10,400		120	4	60	120
Network	High		640		500,000		32		20
Network	Med		160		200,000		16		10
Compute Site	High	1,000,000	125,000						

# Design Solver

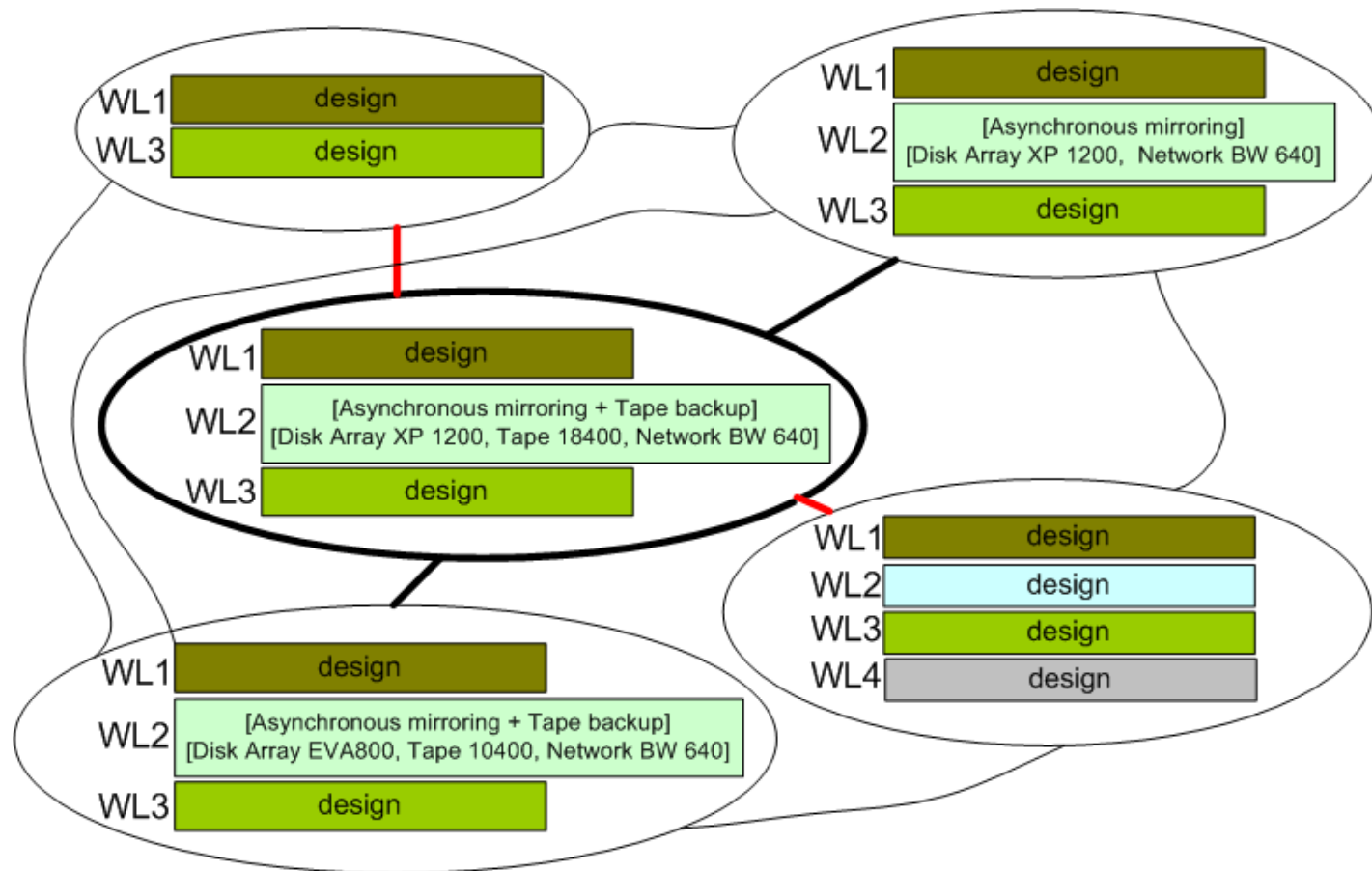
- Output: a candidate design composed of
  - Data protection techniques
  - Resource design



# Space Solution Graph

- Each node of a graph is a partial candidate solution
  - A design for a subset of applications
- Two nodes are connected if they differ by design of one workload
  - Adding a workload (red edges)
  - Changing the protection technique or resource assignment for one workload (black edges)

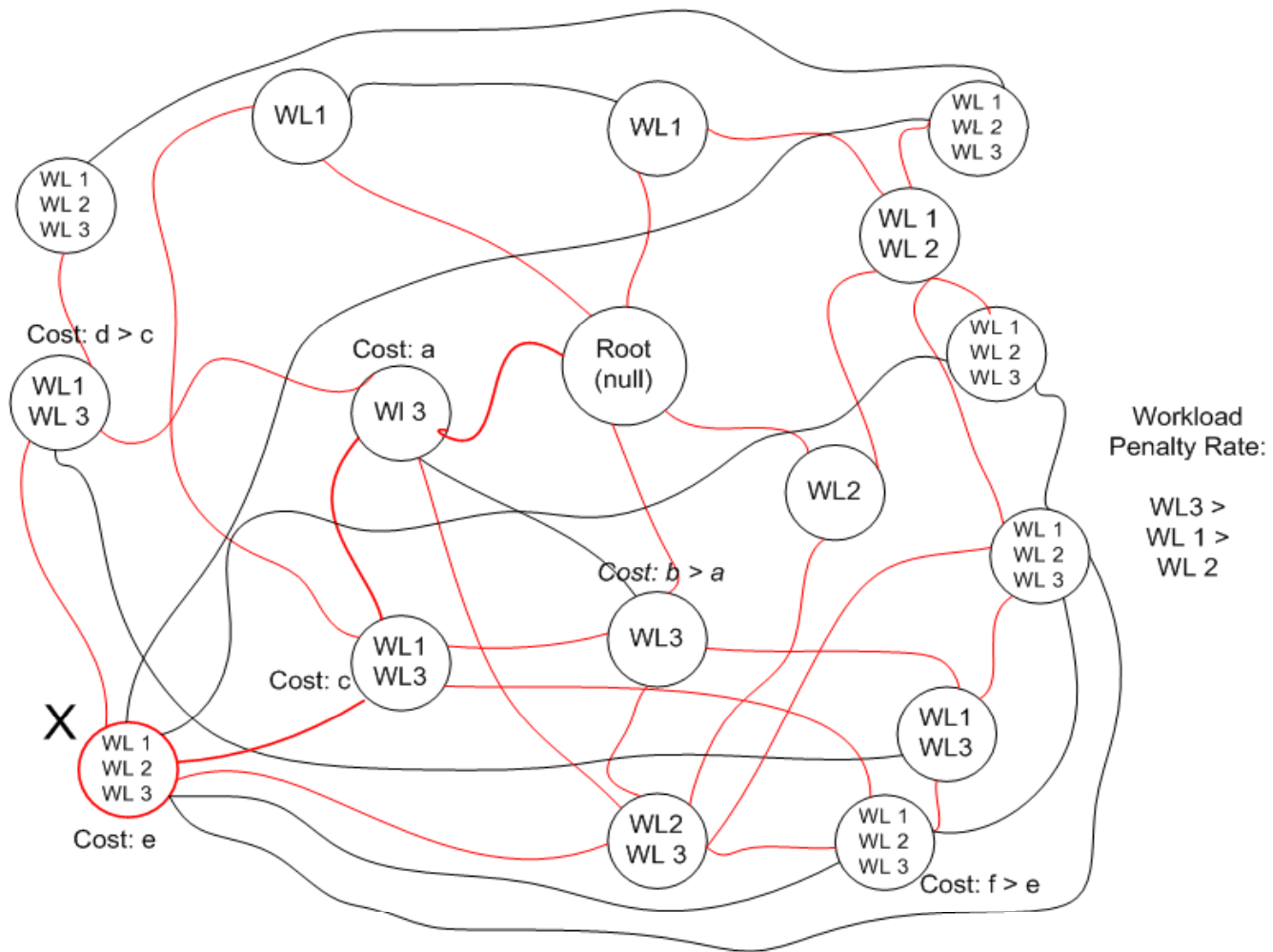
# Space Solution Graph



# Space Solution Graph

- Each node has a cost
  - Find the one with lowest cost [containing all workloads]
- A huge, dense graph !
- There is a unique node which contains no workload (root)
- Start with root and walk on red edges (add workloads) in a greedy manner !

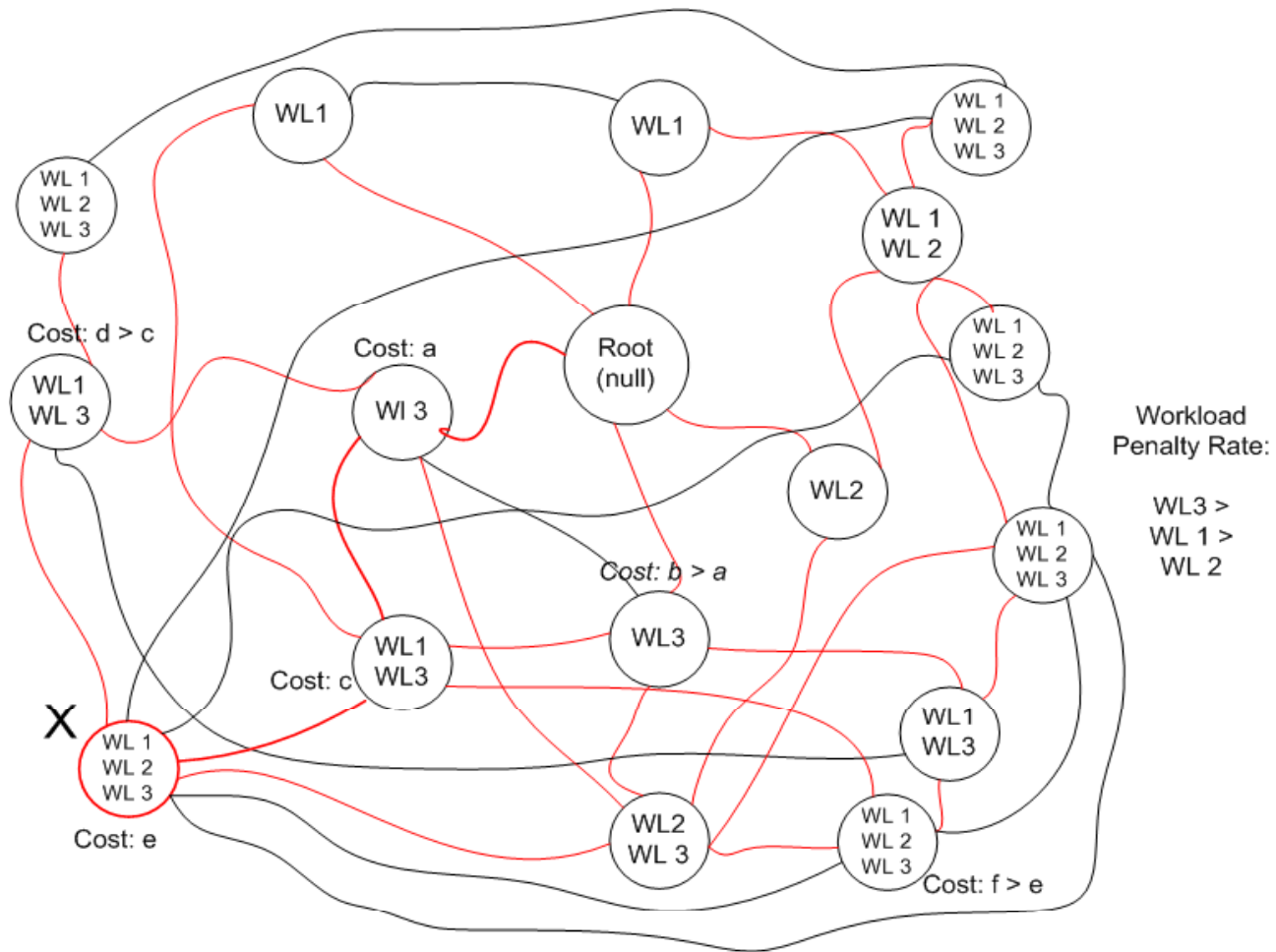
# Greedy best fit algorithm



# Refit Algorithm

- Ignore red edges
- Randomly select 3 neighbours of candidate solution  $x$
- Perform a DFS of depth 5
- Recurs if an better node (design) comparing to  $X$  found

# Refit Algorithm

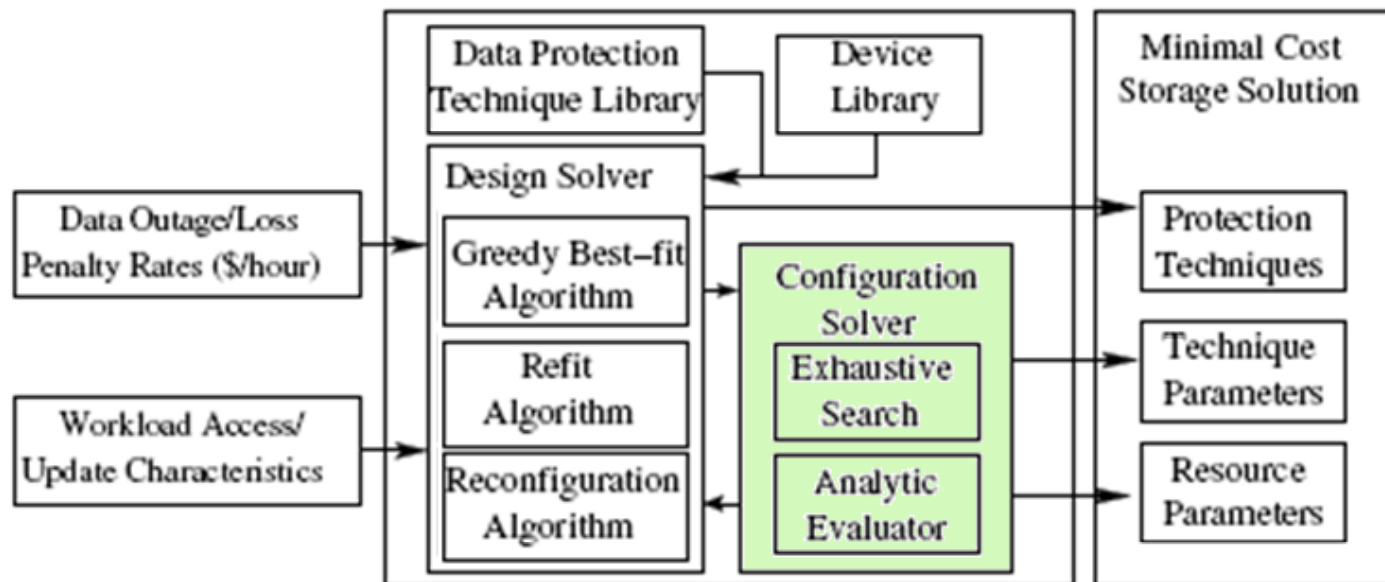


# Reconfiguration

- Used to apply some system requirements
- Move on black edges to find a better node
  - Pick the workload which impose more cost to the node !
- Randomly pick a neighbour
  - Biased to the neighbours with inexpensive techniques
    - Ex: mirroring + tape backup > mirroring
  - Biased to then neighbours with more balanced resource use

# Configuration Solver

- Set the parameters of the resulted design (node)
  - Ex: how often take backups?
- Use exhaustive search
  - Discretized range of values



# Pseudo-code of the Algorithm

## Algorithm – Design Solver

```
1: Let
    $N$            = number of applications
    $A_i$          =  $i^{th}$  application with its parameters,  $1 \leq i \leq N$ 
    $unC$         = unassigned set of applications, i.e.  $\bigcup_{i=0}^N A_i$ 
    $curC$        =  $\emptyset$ , current partial candidate solution
    $newC$       = new partial candidate solution
    $bestC$      = minimum candidate solution seen so far
    $d$          = level of depth of the search of a sibling tree
    $b$          = breadth of search of sub-tree
    $stack[b * d]$  = stack
    $tos$        = top of stack
    $rfgCnt$     = reconfiguration iteration count

2:  $rfgCnt = 0$ 
   {STAGE 1: greedy best-fit algorithm}
3: repeat
4:   choose  $A_i$  such that sum of recovery time and data loss penalty rate
   is maximum from the set of applications in  $unC$ .
5:   reconfiguration( $curC, A_i$ )
6:    $newC = \text{configuration\_solver}(curC)$ 
7:    $curC += A_i, unC -= A_i$ 
8: until ( $unC = \emptyset$ )
   {STAGE 2: re-fit algorithm}
9:  $tos = 0$ 
10:  $bestC = curC$ 
11: if  $rfgCnt > threshold$  then
12:   terminate solver, return  $bestC$  to User.
13: end if

14: repeat
15:    $stack[tos ++] = curC$ 
16:   for  $i = 1$  to  $b$  do
17:      $curC = \text{reconfiguration}(curC)$ 
18:      $curC = \text{configuration\_solver}(curC)$ 
19:      $stack[tos ++] = curC$ 
20:      $j = 0$ 
21:     while ( $j \leq d$ ) do
22:        $popCnt = 0$ 
23:       for  $k = 1$  to  $b$  do
24:          $newC = \text{reconfiguration}(curC)$ 
25:          $newC = \text{configuration\_solver}(newC)$ 
26:         if ( $cost(newC) < cost(curC)$ ) then
27:            $stack[tos ++] = curC$ 
28:            $popCnt = popCnt + 1$ 
29:         end if
30:       end for
31:        $curC = \text{find\_min}(stack, popCnt)$ 
       {find minimum cost solution for the current level}
32:        $tos = tos - popCnt$ 
33:        $stack[tos ++] = curC$ 
34:        $j = j + 1$ 
35:     end while
36:      $curC = stack[0]$ 
       {restart search for the next sibling of the initial node}
37:   end for
38:    $bestC = \text{find\_min}(stack, tos)$ 
39:    $tos = 0$ 
40:    $curC = stack[tos ++] = bestC$ 
41:    $rfgCnt = rfgCnt + 1$ 
   {if sufficient progress check fails, go back to best-fit}
42: until ( $rfgCnt > max$ ) || (user-defined termination condition)
43: return  $bestC$ 
```

# Experimental Results

- Compare the proposed method with a human architect
  - A new heuristic called “human heuristic”
- Our solutions are 2 to 3 times better (Hoorah !)
- The algorithm is scalable
- Millions of dollars can be saved !!!

# Conclusion and future work

- A new method is presented for automatically generating near optimal storage design in shared-application environment
  - A heuristic for exploring the huge search space
- The heuristic works better than a worst heuristic presenting human architect !
- Future work: none !
- A lot of problems and black holes in the approach !

Thank you

By the way... I rejected this paper !