# Policy Based Storage System in Heterogeneous Environment

Dai Qin, Ashvin Goel, Angela Demke Brown
University of Toronto
{mike,ashvin}@eecg.toronto.edu demke@cs.toronto.edu

## Motivation

Storage environments are highly heterogeneous

**Workloads and Applications**
- Databases
- File systems
- NFS servers
- Scientific clusters

**Storage and Hardware**
- SSDs (PCIE/SATA)
- Disks (SAS/SATA)
- Remote storage
- Tapes

Management Challenges

**Current solutions are**:
Applications manage underlying hardware
Examples:
File systems: ZFS/btrfs
- ZFS as volume manager
- Integrated: metadata replication, RAID-Z,...

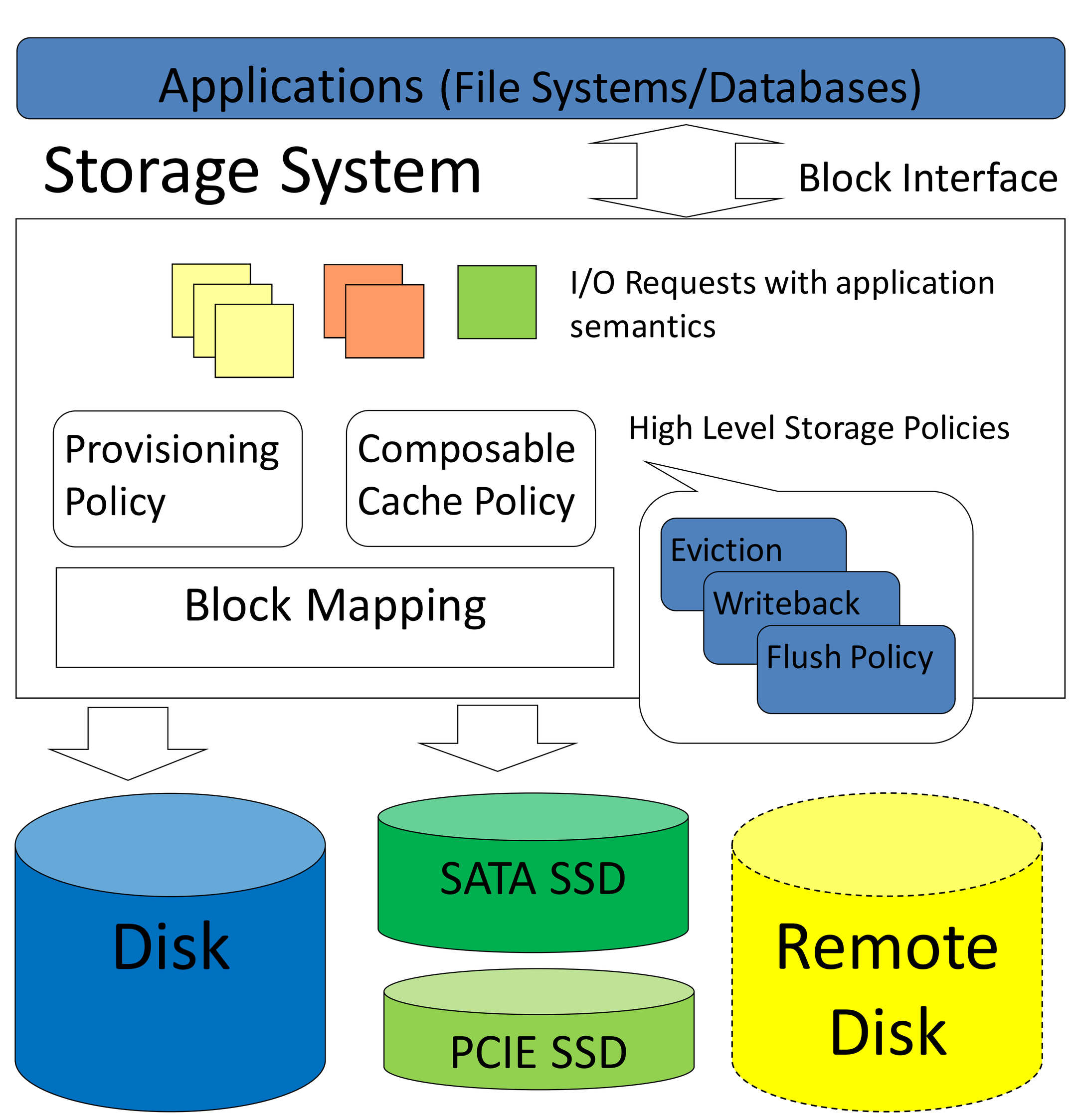
NFS Servers: NetApp Applience
- Even integrate special hardware

**Problems with current solutions: What if …?**
- Management features are not built in?
- Storage virtualization/deeply stacked storage?
- Users run other applications?

### Monolithic solutions are inflexible

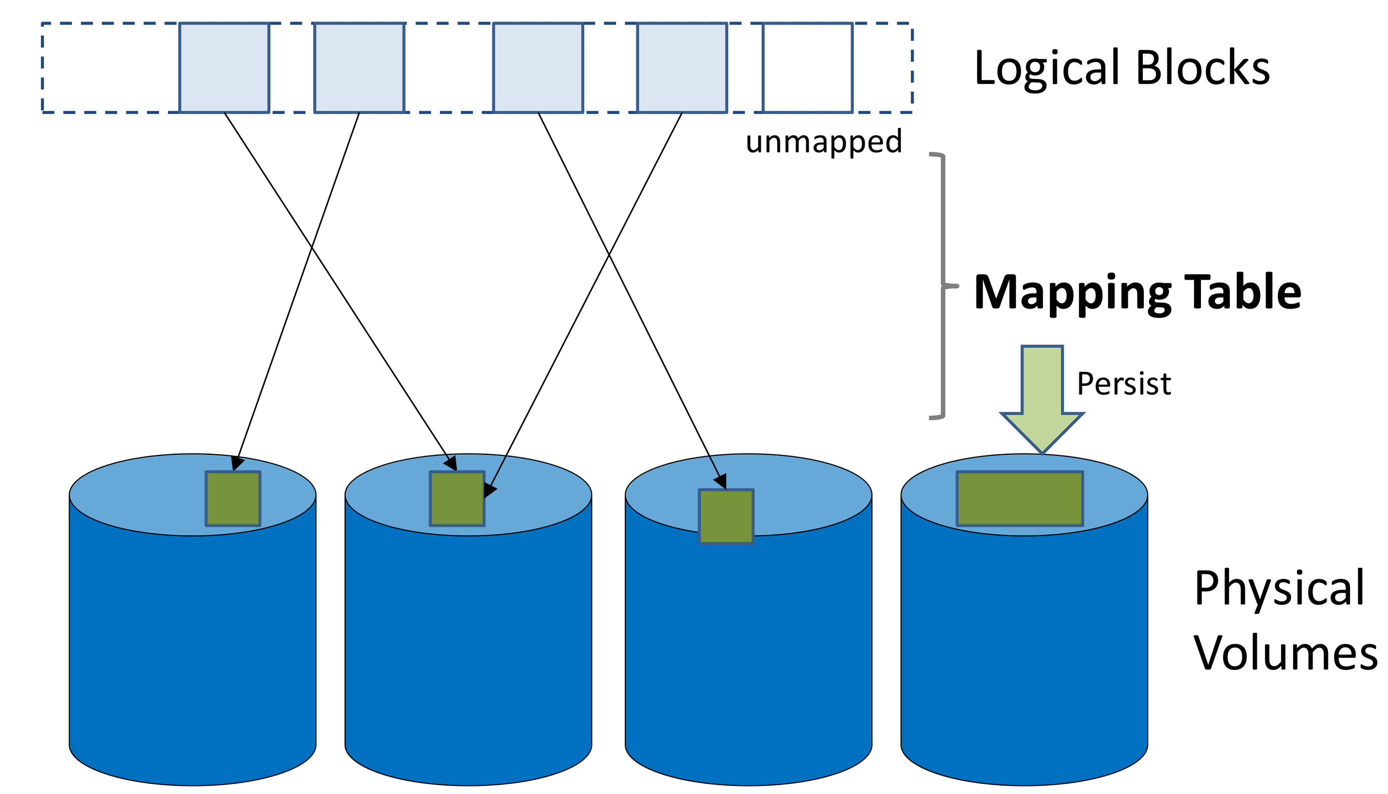
## Policy Based Management

- Storage hardware is associated with properties (e.g. performance, reliability)
- Storage system is aware of application semantics
  - Metadata vs data
  - Allocated vs unallocated
- Flexible policies map application requests to hardware capability
- Composable policies make it easier to create new solutions.




## Challenges

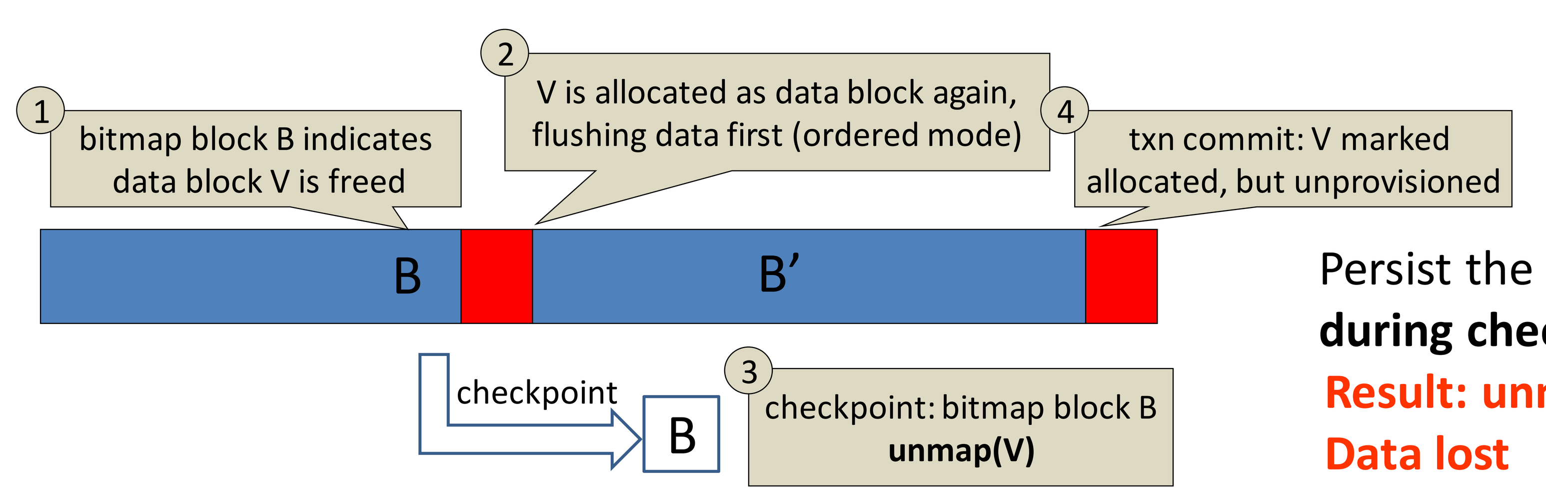
- Policies may need to persist their own metadata
  - Key-Value pair like mapping




- Mapping needs to be **consistent** with application state
- Mapping shouldn't violate **durability expectations**

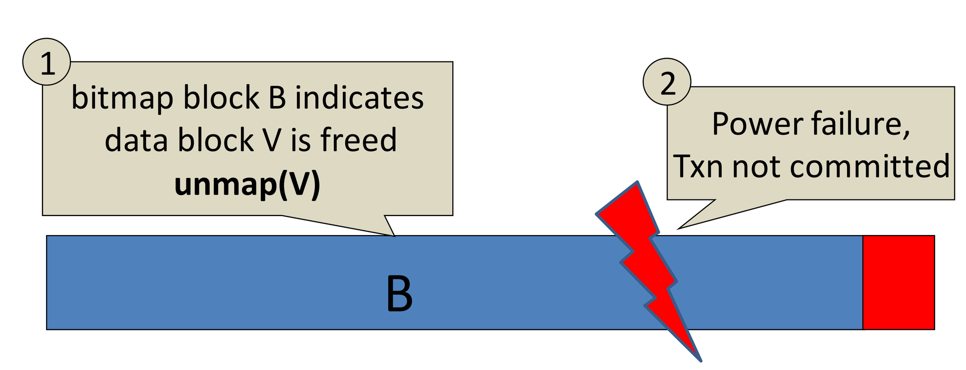
## Challenge: Consistency

- Example Policy: Thin Provisioning in a journaled file system
  - When bitmap deallocate a block, reclaim it



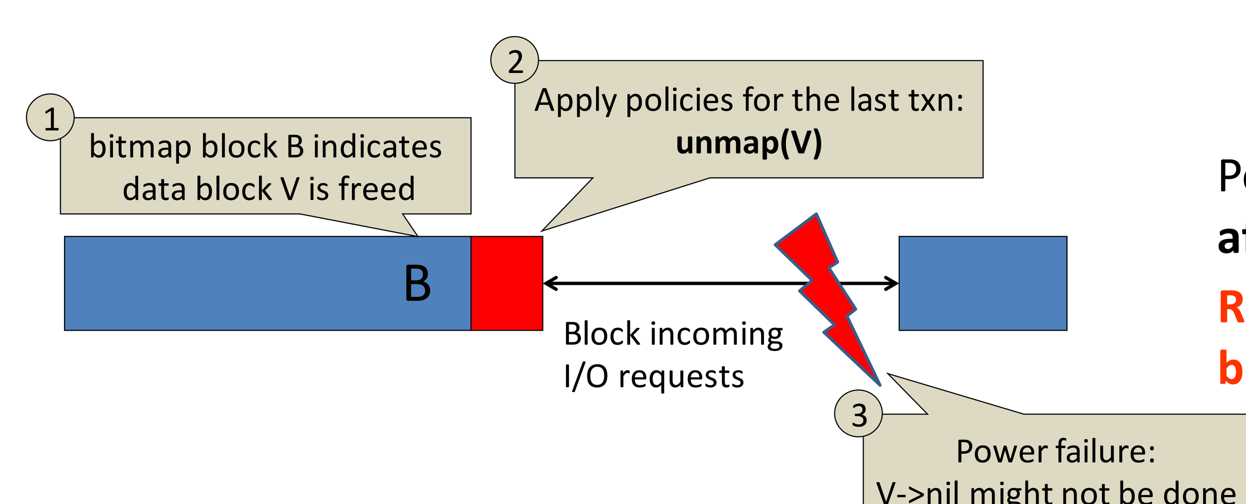

Persist the mapping **during checkpoint**
**Result: unmap(V) Data lost**

**Lessons**: ordering must be preserved




Persist the mapping **before txn commit**
**Result: unmap(V) Data lost**

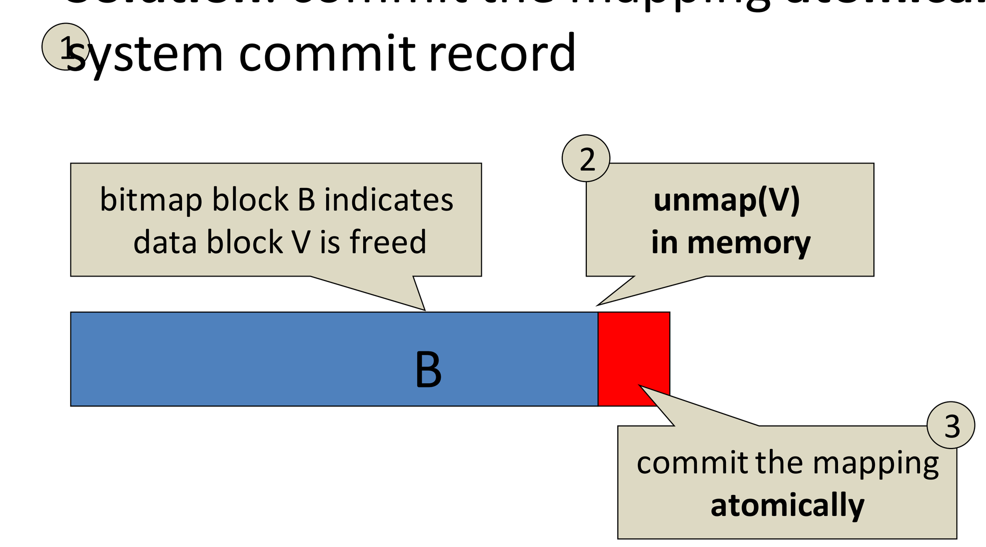
**Lessons**: before txn commit would fail




Persist the mapping **after txn commit**
**Result: V might still be provisioned**

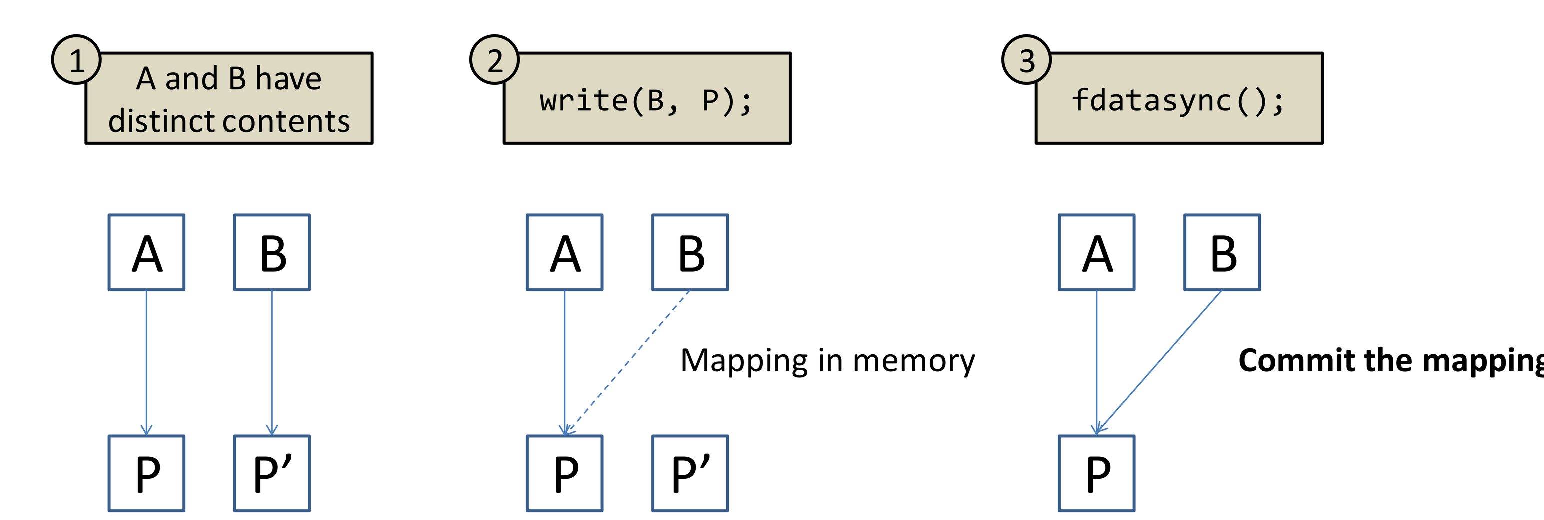
**Lessons**: after txn commit would fail

**Solution**: commit the mapping **atomically** with file system commit record




## Challenge: Durability

- Application have durability expectations
  - After disk barrier applications expect that file data are durable
- Example Policy: Inline Deduplication
  - Blocks with same content mapped to same location
  - Need to persist the mapping on `fdatasync()`




## Summary of Solutions

| Guarantee | Durability | Consistency |
|---|---|---|
| When to create mapping | On data update | After txn finish Before commit |
| When to commit mapping | On disk barrier | Commit atomically wit txn |