Solaris Dynamic File System

Sun Microsystems, Inc.

Sun microsystems

The Perfect Filesystem

- Write my data
- Read it back
- Keep it simple
- Keep it safe
- Do it fast

Existing Filesystems

- Difficult and complex to manage
- No data integrity checks
- No defense against silent data corruption
- No data security: spying, tampering, theft
- Many limits: size, number of files, etc.
- Performance and scaling problems

Solaris Dynamic File System

Unlimited Scalabilty

- Write my data
  - Immense capacity (128-bit)
    - Moore's Law: need 65th bit in 12 years
    - Zettabyte = 70-bit (a billion TB)
    - Dynamic File System capacity: 256 quadrillion ZB
    - Quantum limit of Earth-based storage
  - Dynamic metadata
    - No limits on files, directory entries, etc.
    - No strange knobs (e.g. Inodes/cg)

Solaris Dynamic File System

Reduced Administrative Overhead

- Keep it simple
  - Pooled storage – no more volumes
  - Filesystems are cheap and easy to create
  - Grow and shrink are automatic
  - No raw device names to remember
  - No more fsck(1M)
  - No more editing /etc/vfstab
  - Unlimited snapshots and user undo
  - All administration online

Solaris Dynamic File System

Data Integrity

- Keep it safe
  - Provable data integrity model
    - Complete end-to-end verification
    - 99.9999999999999% certainty of error detection
    - Detects bit rot, phantom writes, misdirections, common administrative errors
  - Self-healing data
  - Disk scrubbing
  - Real-time remote replication
  - Data authentication and encryption

Solaris Dynamic File System

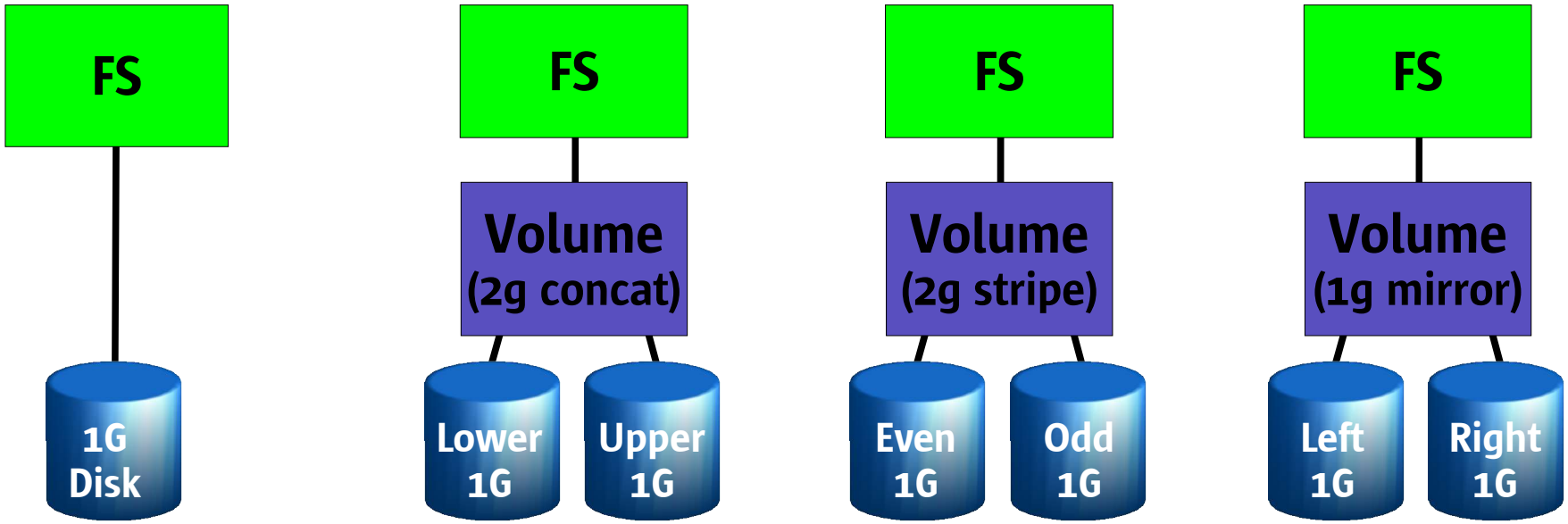
High Performance

- Do it fast
  - Write sequentialization
  - Dynamic striping across all disks
  - Multiple block sizes
  - Constant-time snapshots
  - Concurrent, constant-time directory ops
  - Byte-range locking for concurrent writes

Background: Why Volumes Exist

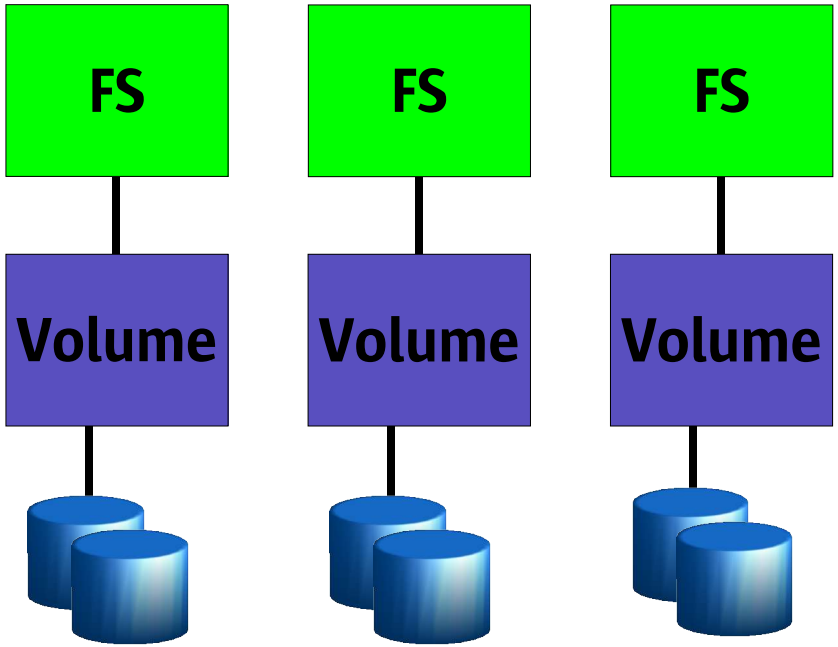
In the beginning, each filesystem managed a single disk

- Users wanted more space, bandwidth, reliability
  - Rewrite filesystems to handle many disks: hard
  - Insert a “volume” to tie disks together: easy
- An industry grew up around the FS/volume model
  - Filesystems, volume managers sold as separate products
  - Inherent problems in FS/volume interface can't be fixed

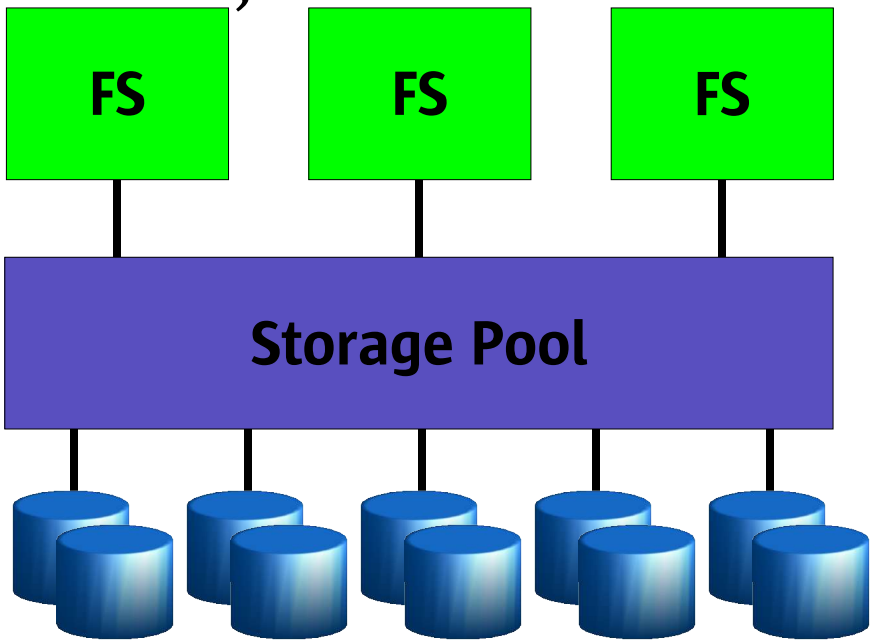


Volumes vs. Pooled Storage

- Traditional volumes
  - Partition per filesystem; painful to manage
  - Block-based FS/Volume interface slow, brittle



- Pooled Storage
  - Filesystems share space
  - Easy to manage
  - Transactional interface fast, robust



Storage Administration

- The Task:
  - Given two disks, create mirrored filesystems for three users - Ann, Bob, and Sue
  - Later, add more space

Solaris 8 Administration

```
# format
... (long interactive session omitted)

# metadb -a -f disk1:slice0 disk2:slice0

# metainit d10 1 1 disk1:slice1
d10: Concat/Stripe is setup
# metainit d11 1 1 disk2:slice1
d11: Concat/Stripe is setup
# metainit d20 -m d10
d20: Mirror is setup
# metattach d20 d11
d20: submirror d11 is attached

# metainit d12 1 1 disk1:slice2
d12: Concat/Stripe is setup
# metainit d13 1 1 disk2:slice2
d13: Concat/Stripe is setup
# metainit d21 -m d12
d21: Mirror is setup
# metattach d21 d13
d21: submirror d13 is attached

# metainit d14 1 1 disk1:slice3
d14: Concat/Stripe is setup
# metainit d15 1 1 disk2:slice3
d15: Concat/Stripe is setup
# metainit d22 -m d14
d22: Mirror is setup
# metattach d22 d15
d22: submirror d15 is attached
```

```
# newfs /dev/md/rdsk/d20
newfs: construct a new file system /dev/md/rdsk/d20: (y/n)? y
... (many pages of 'superblock backup' output omitted)
# mount /dev/md/dsk/d20 /export/home/ann
# vi /etc/vfstab  ... while in 'vi', type this exactly:
/dev/md/dsk/d20 /dev/md/rdsk/d20 /export/home/ann ufs 2 yes -

# newfs /dev/md/rdsk/d21
newfs: construct a new file system /dev/md/rdsk/d21: (y/n)? y
... (many pages of 'superblock backup' output omitted)
# mount /dev/md/dsk/d21 /export/home/ann
# vi /etc/vfstab  ... while in 'vi', type this exactly:
/dev/md/dsk/d21 /dev/md/rdsk/d21 /export/home/bob ufs 2 yes -

# newfs /dev/md/rdsk/d22
newfs: construct a new file system /dev/md/rdsk/d22: (y/n)? y
... (many pages of 'superblock backup' output omitted)
# mount /dev/md/dsk/d22 /export/home/sue
# vi /etc/vfstab  ... while in 'vi', type this exactly:
/dev/md/dsk/d22 /dev/md/rdsk/d22 /export/home/sue ufs 2 yes -

# format
... (long interactive session omitted)
# metattach d12 disk3:slice1
d12: component is attached
# metattach d13 disk4:slice1
d13: component is attached
# metattach d21
# growfs -M /export/home/bob /dev/md/rdsk/d21
/dev/md/rdsk/d21:
... (many pages of 'superblock backup' output omitted)
```

Dynamic File System Administration

- Create a storage pool named “home”

```
# zpool create "home" mirror(disk1,disk2)
```

- Create filesystems “ann”, “bob”, “sue”

```
# zfs mount -c home/ann /export/home/ann
# zfs mount -c home/bob /export/home/bob
# zfs mount -c home/sue /export/home/sue
```

- Later, add space to the “home” pool

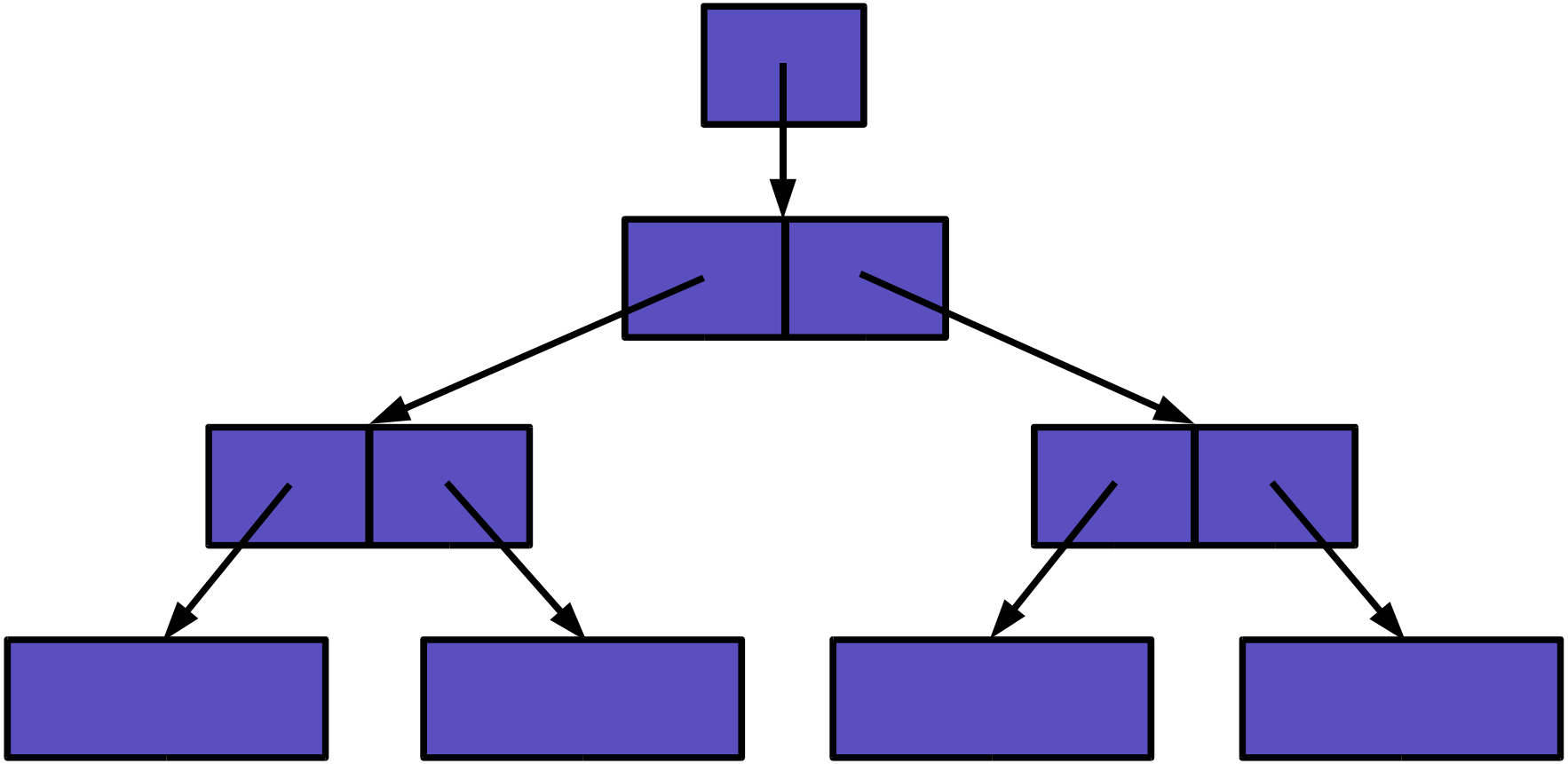
```
# zpool add "home" mirror(disk3,disk4)
```

Provable Data Integrity Model

- Three Big Rules
  - All operations are copy-on-write
    - Never overwrite live data
    - On-disk state always valid
    - No need for fsck(1M)
  - All operations are transactional
    - Related changes succeed or fail as a whole
    - No need for journaling
  - All data is checksummed
    - No silent data corruption
    - No panics on bad metadata

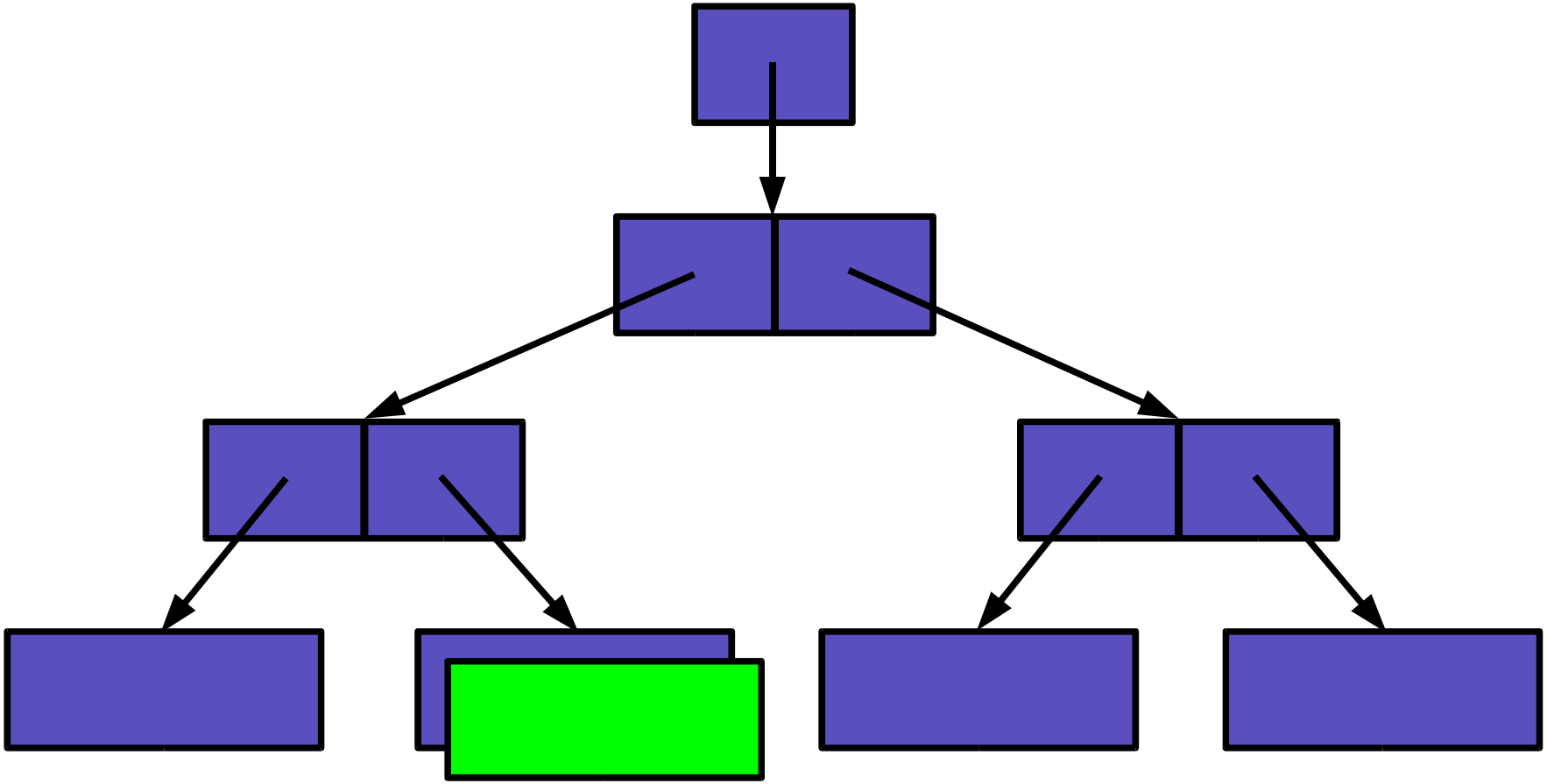
Copy-On-Write Transaction Model

- Initial block tree



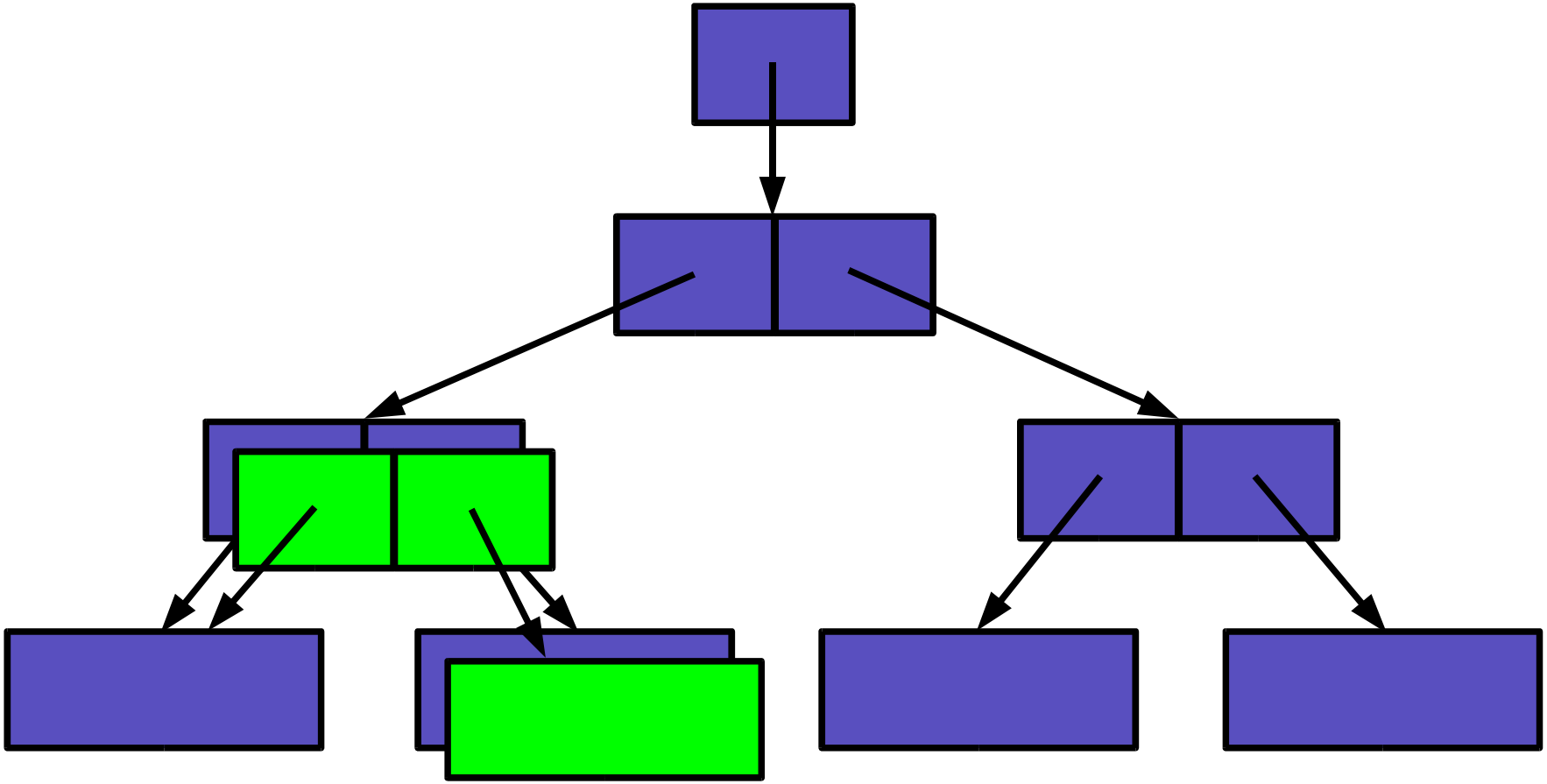
Copy-On-Write Transaction Model

- Write: Copy-on-write a data block



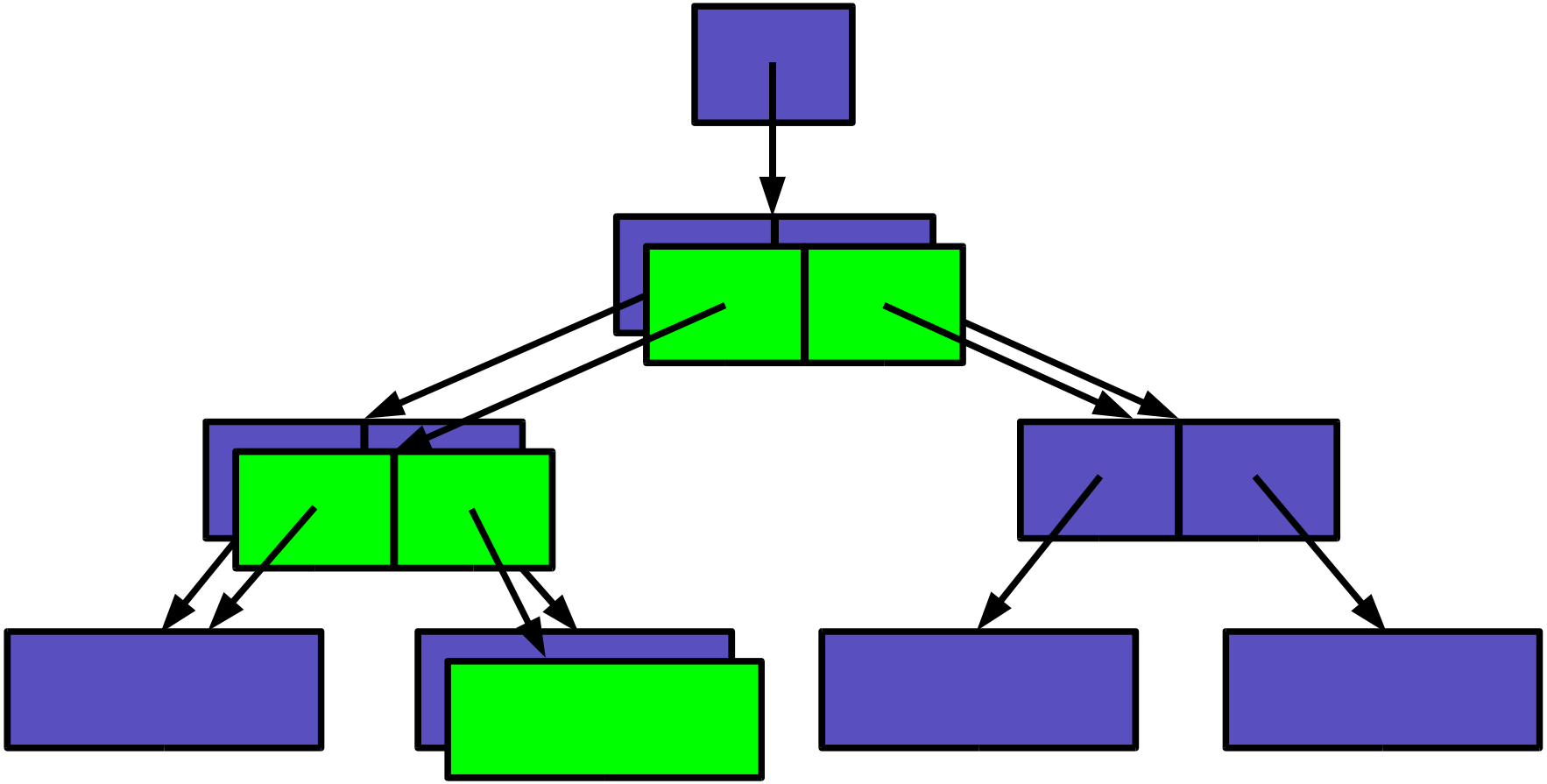
Copy-On-Write Transaction Model

- Copy-on-write its level-1 indirect block



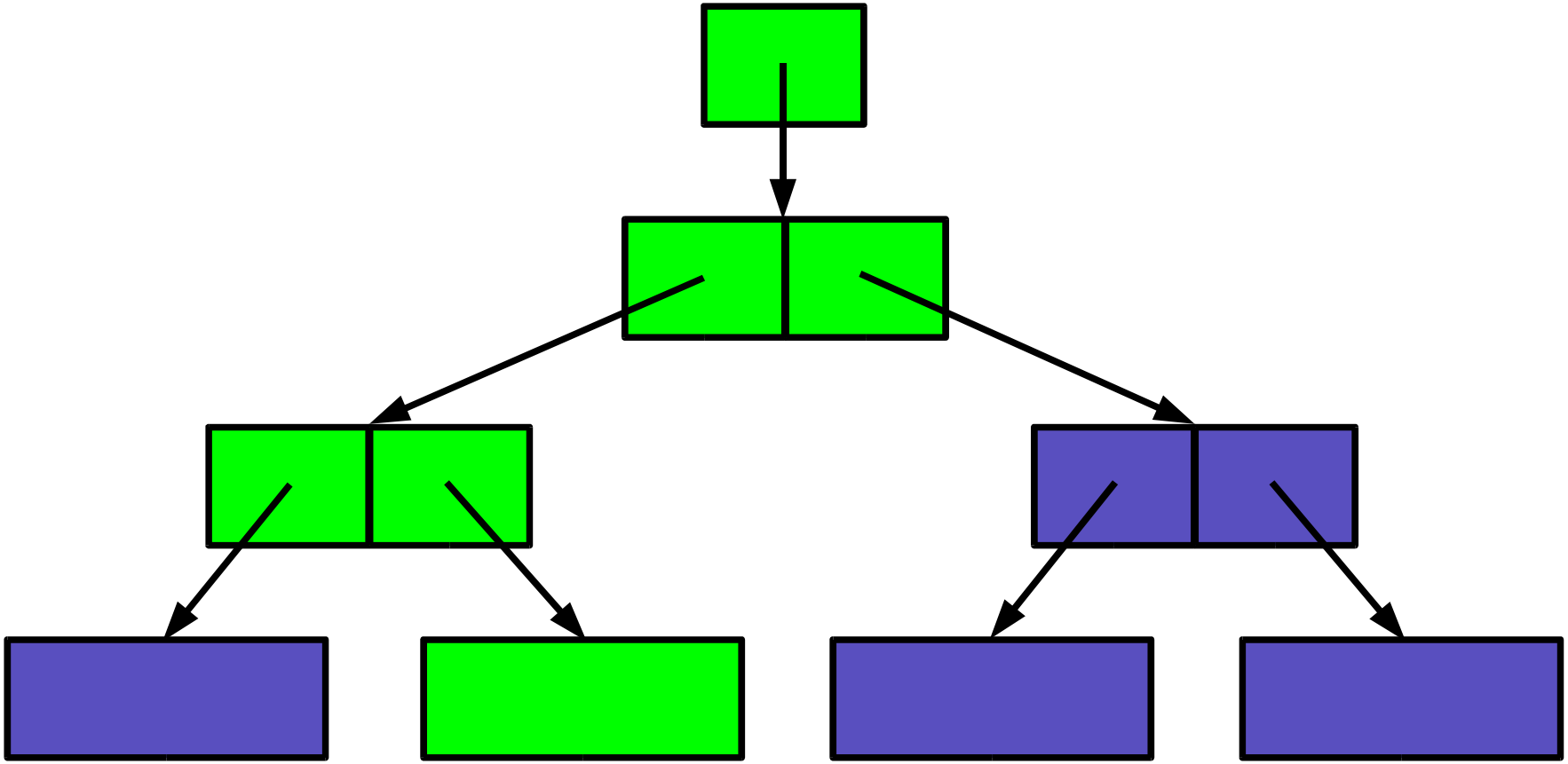
Copy-On-Write Transaction Model

- Copy-on-write its level-2 indirect block



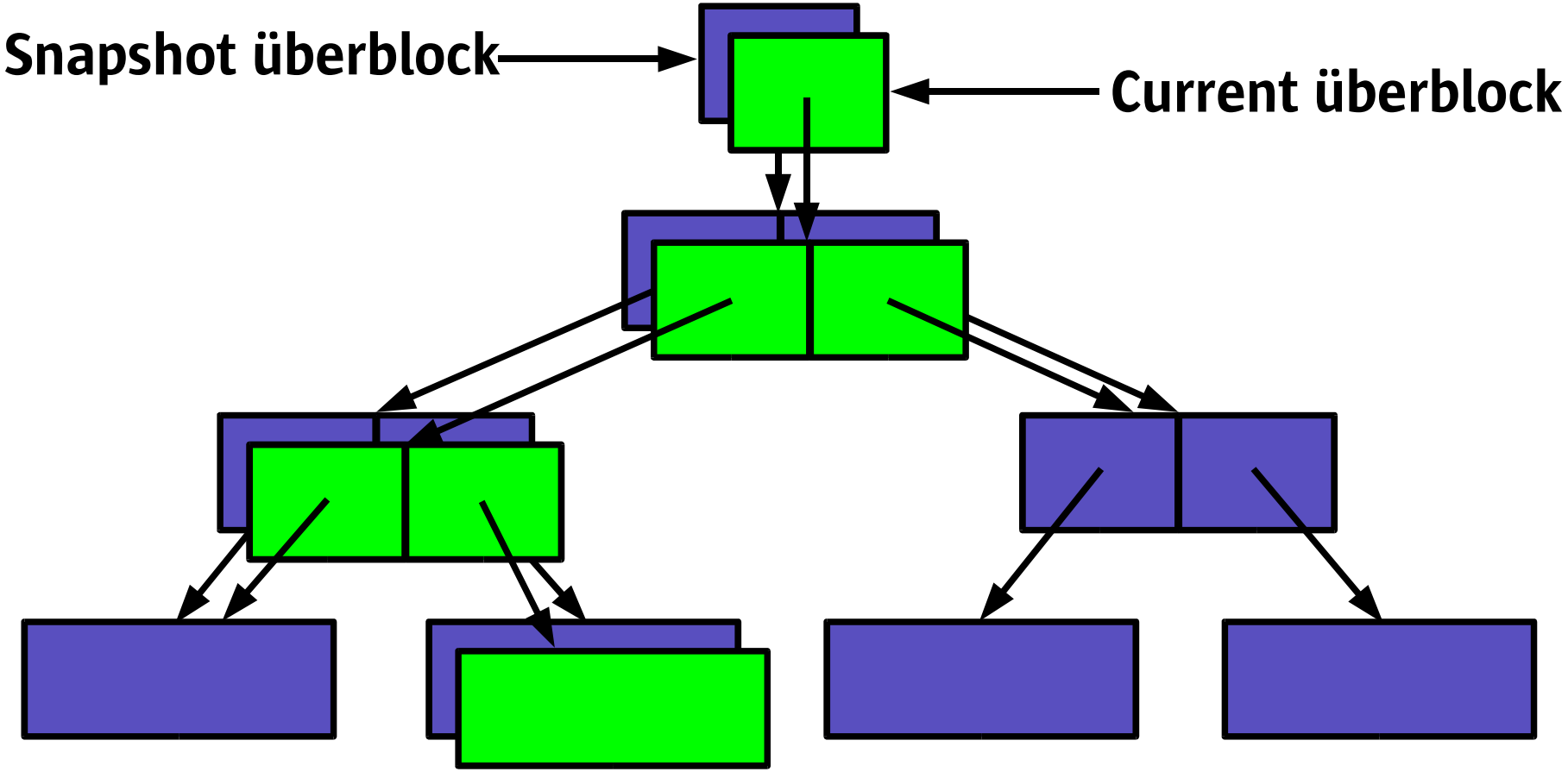
Copy-On-Write Transaction Model

- Rewrite the überblock (atomic)



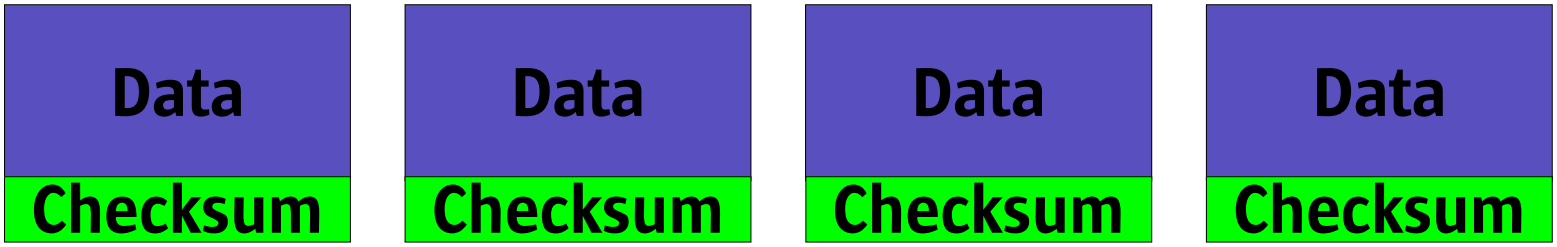
Snapshots are Free!

- At end of transaction, do not free old blocks



Traditional Checksums

- Checksums stored with data blocks

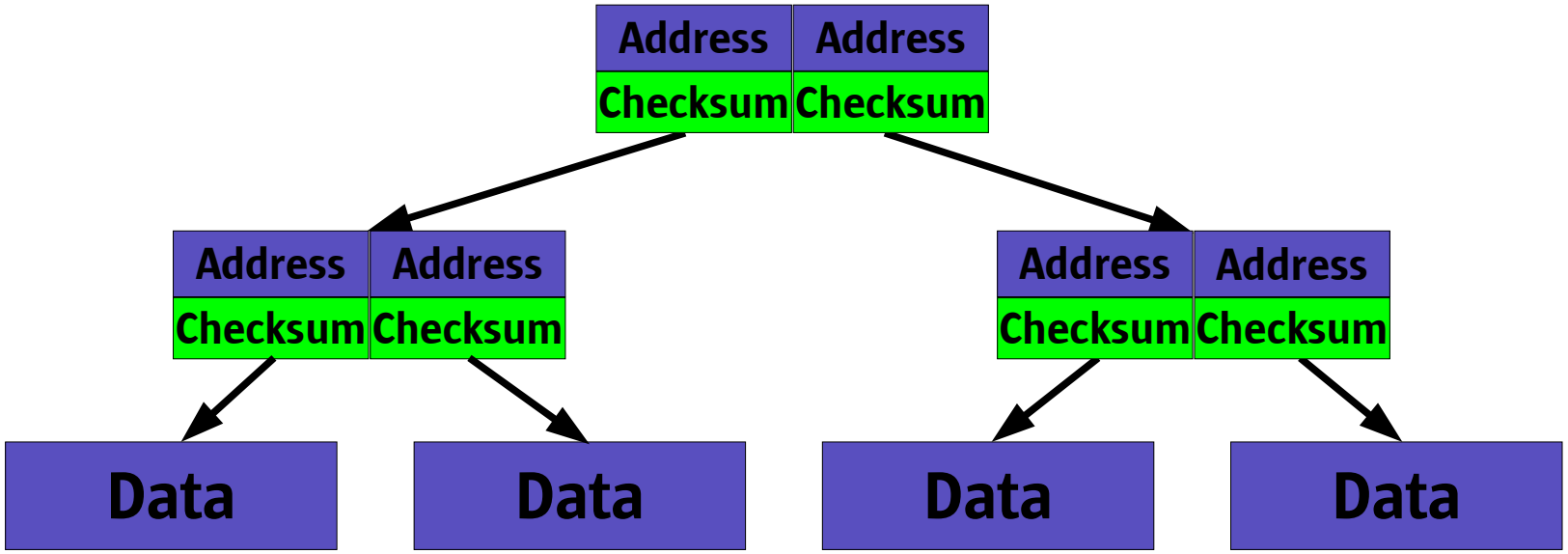


Fine for detecting bit rot, but:

- Cannot detect phantom writes, misdirections
- Cannot validate the checksum itself
- Cannot authenticate the data
- Cannot detect common administrative errors

Dynamic File System Checksums

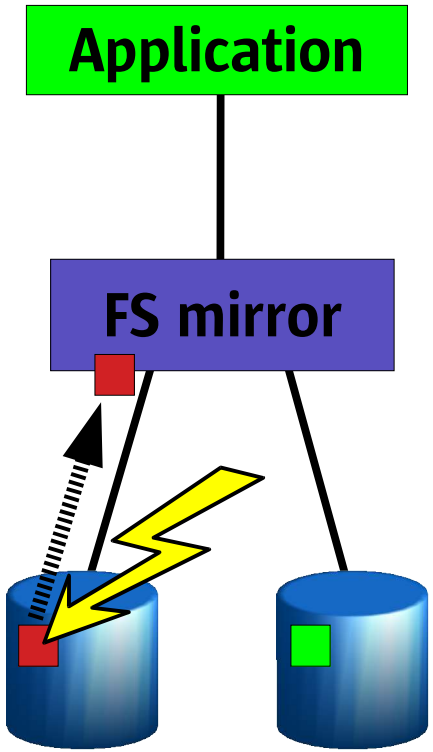
- Checksums stored with indirect blocks



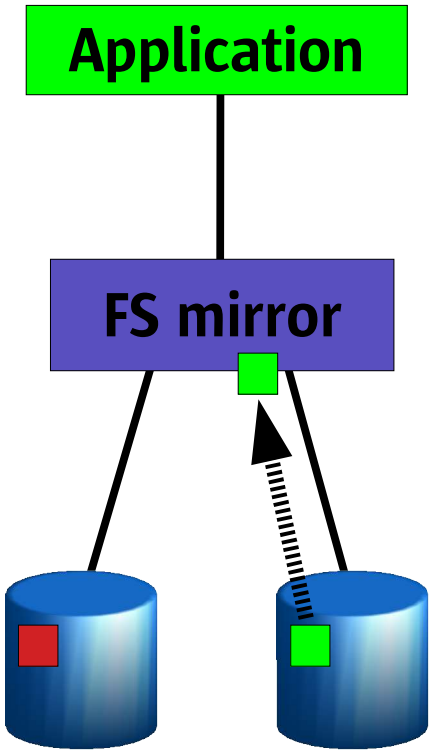
  - Self-validating, self-authenticating checksum tree
  - Detects phantom writes, misdirections, common administrative errors

Self-Healing Data

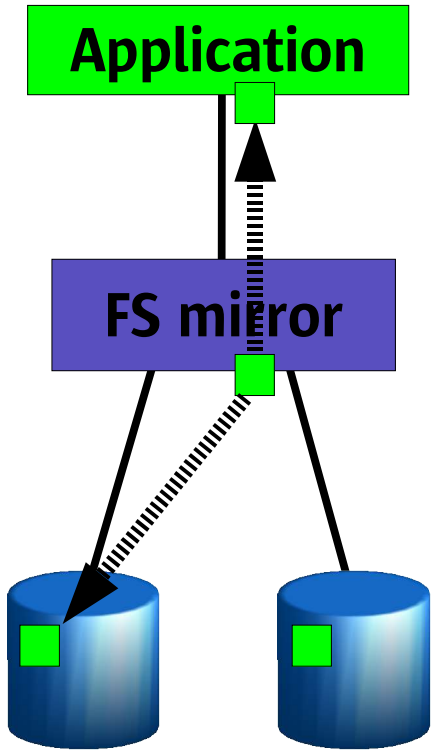
1. Issue a read.
Try the first disk.
Checksum reveals that the block is corrupt on disk.



2. Try the second disk.
Checksum indicates that the block is good.



3. Return good data to the application.
Repair the damaged block.



Self-Healing Data in Action

```
# dd if=/dev/zero of=/dev/dsk/c2d9d0s0 bs=128k ... count=12
# ... read the affected file ... no problem!
# zpool iostat home
                              capacity     operations    bandwidth
vdev   description           used  avail   read  write   read  write   err
   1   mirror(2,3)            305M  136G    167      0 21.0M      0   0/0
   2     /dev/dsk/c2t8d0s0   -----  ----     88      0 11.0M      0   0/0
   3     /dev/dsk/c3t8d0s0   -----  ----     79      0  9.9M      0   0/0
   4   mirror(5,6)            256M  136G    168      0 21.0M      0  12/12
   5     /dev/dsk/c2t9d0s0   -----  ----     86      0 10.8M      0  12/0
   6     /dev/dsk/c3t9d0s0   -----  ----     81      0 10.2M      0   0/0
   7   mirror(8,9)            258M  136G    169      0 21.2M      0   0/0
   8     /dev/dsk/c2t10d0s0  -----  ----     93      0 11.7M      0   0/0
   9     /dev/dsk/c3t10d0s0  -----  ----     76      0 9.45M      0   0/0
  10   mirror(11,12)          257M  136G    176      0 22.1M      0   0/0
  11     /dev/dsk/c2t11d0s0  -----  ----     85      0 10.7M      0   0/0
  12     /dev/dsk/c3t11d0s0  -----  ----     91      0 11.3M      0   0/0
```

Where Are We Now?

- Initial Work complete
  - Complete POSIX-compliant filesystem
  - Full builds of Solaris
  - Key features:

    pooled storage

    dynamic striping

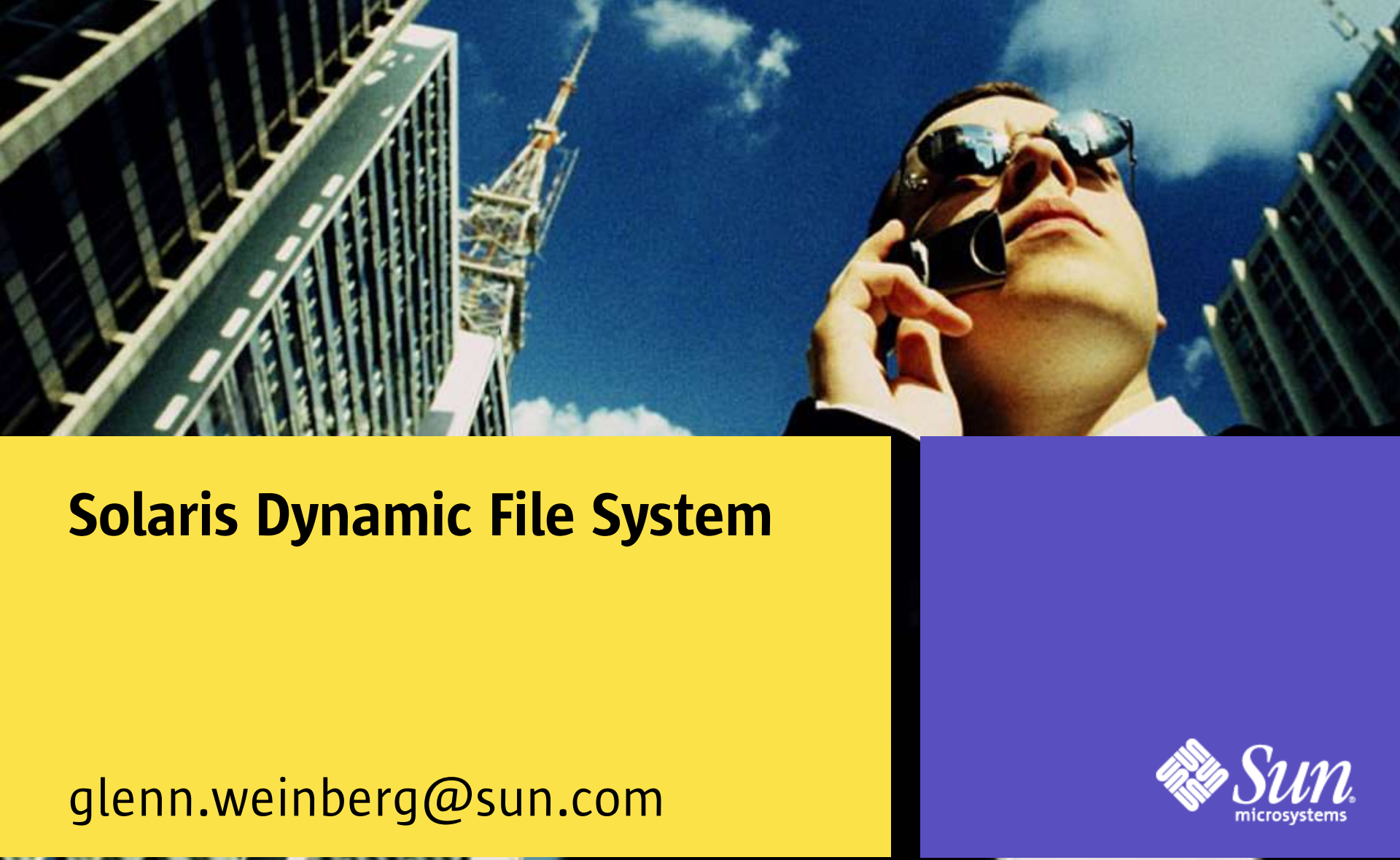
    self-healing data: even under sustained, abusive fault injection

    crash resilience: over 1,000,000 forced, violent crashes, never lost data integrity
- Dynamic File System – Coming soon in Solaris Express

Solaris Dynamic File System

Simple, Reliable, and Infinitely Scalable

- Breakthrough data management approach
  - Efficient resource allocation via storage pools
  - Automates administrative tasks
- Perpetual data integrity, availability
  - Pervasive data fault detection and correction
  - Defends against common administrative errors
  - Extensible: add features such as encryption
- Virtually unlimited capacity
  - 16 *billion billion* times greater than today
- Reduced costs
  - Higher terabyte-to-administrator ratio
  - Lower cost of acquisition, testing, maintenance



Solaris Dynamic File System

glenn.weinberg@sun.com

Sun microsystems