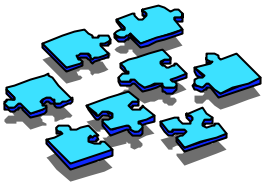
Tivoli Storage Manager Server Internals

Dave Cannon
Tivoli Storage Management Development
Oxford University TSM Symposium
September 2001



Agenda

- Overview
- Session management
- Database applications
- Transaction management
- Inventory management
- Bitfiles
- Storage service
- Data-transfer examples
- Database and recovery log
  - Structure
  - Referential integrity
  - Recoverability/reorganization

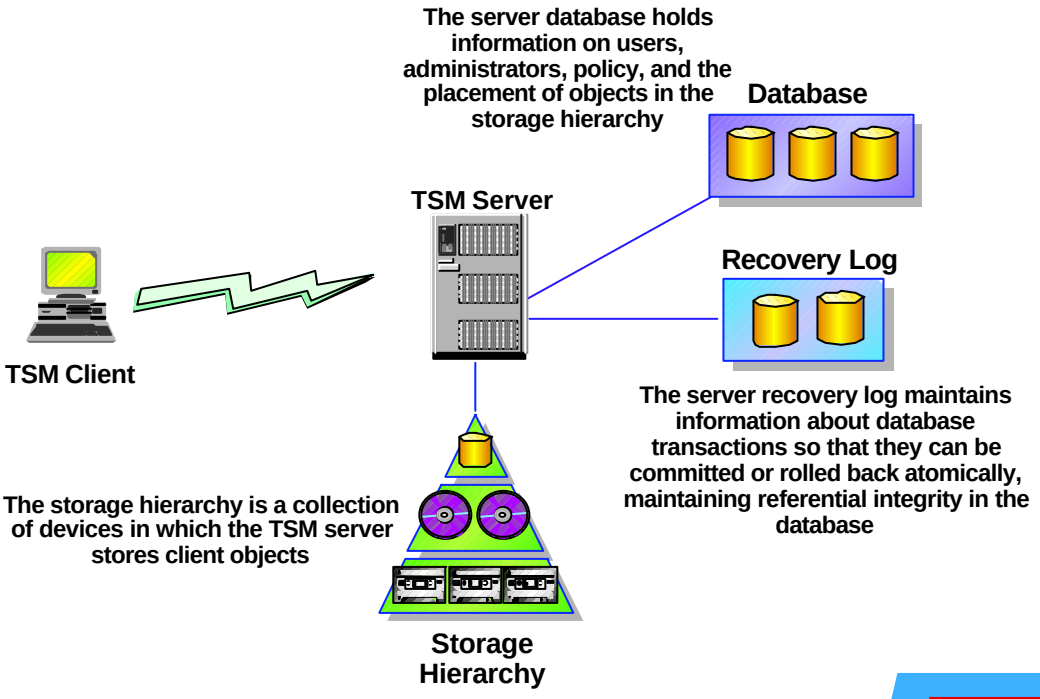
Tivoli

Disclaimer

- For illustrative purposes, this presentation contains the results of various show commands
- Show commands are intended for diagnostic use only under the direction of IBM Service or Development personnel
- If you should choose to use show commands on your own, please understand that
  - These commands are neither documented nor supported
  - Results may not match those obtained from Query or Select commands
  - Problem management records should not be opened regarding show commands

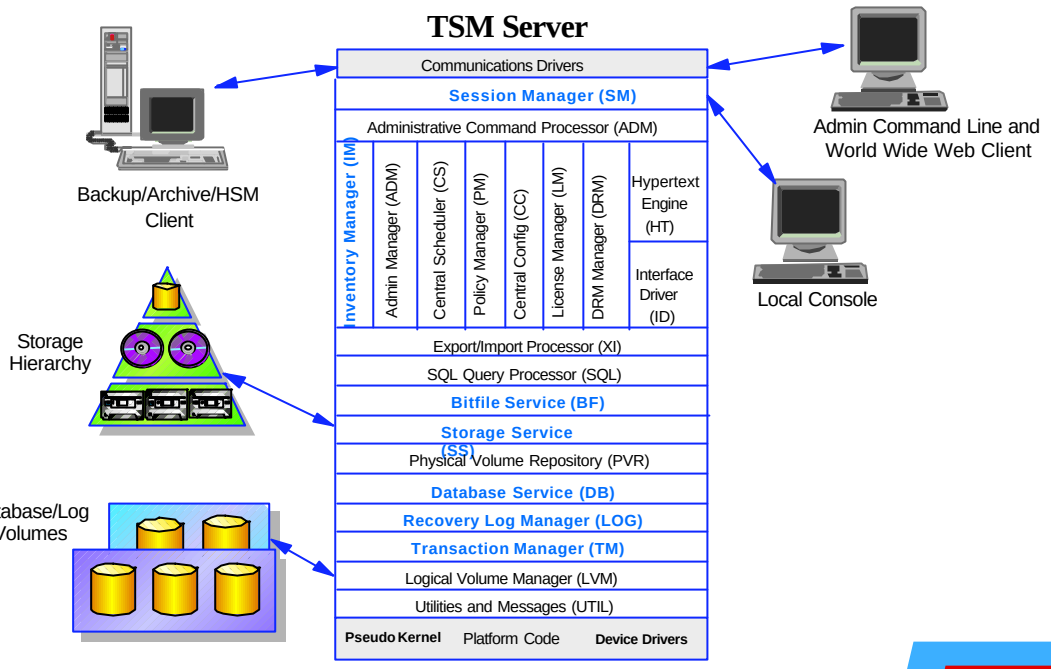
Tivoli

Topology



Tivoli

TSM Architecture Overview



Session Management

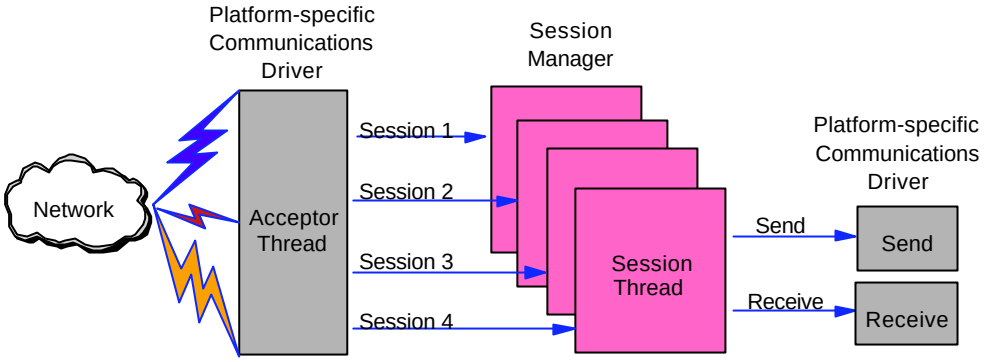
What Session Manager Does

- Handles local console input/output
- Manages remote sessions
  - Independent of communication method
  - Verifies functional capabilities of both parties
  - Performs authentication
  - Executes session protocol
  - Invokes services of other server components to satisfy requests made during the session

Session Types

- Client node
- Client scheduler
- Virtual volumes
- Administrative client
- Mount console
- Remote console
- Export/import
- Event logging
- Enterprise configuration
- Command routing
- HTTP
- Library sharing
- Storage agent

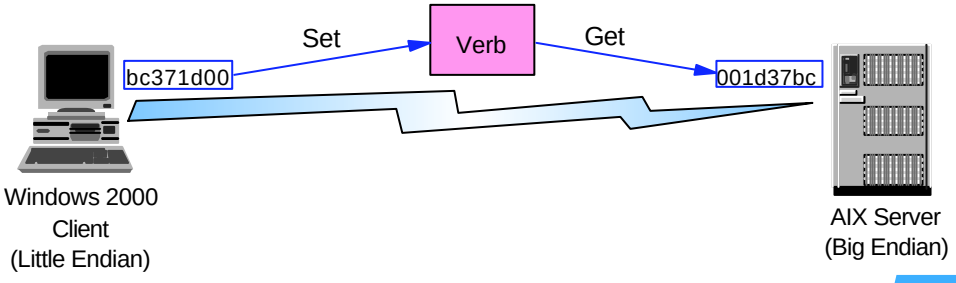
Session Communications



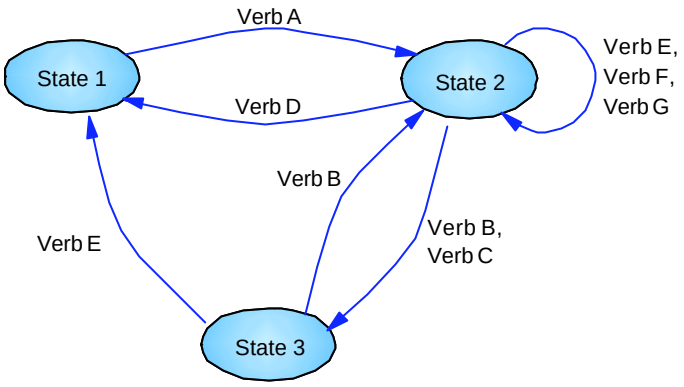
- Inbound connections are accepted by the acceptor thread
- Acceptor thread starts a session thread and assigns inbound session to the new thread
- Sessions are half-duplex

Verbs

- For most session types, information is transferred in structured messages called "verbs"
- Verbs represent data in "network format"
  - Integers can be exchanged between big-endian and little-endian machines
  - Dates
  - Variable-length strings

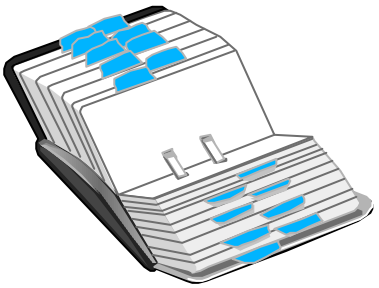


Session Protocols

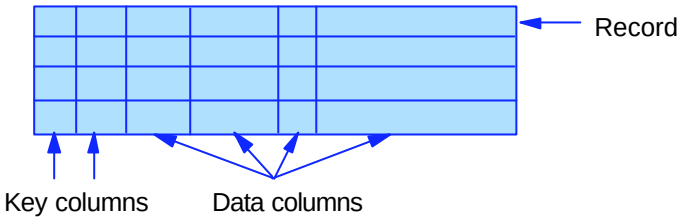


- Automaton (state machine) defines expected protocol for each session type
- State transitions occur when an expected verb is received
- Protocol error occurs if the automaton receives an unexpected verb

Database Applications



Database Table Objects



- Most components maintain tables to support that component's function
- Table schema describes key and data columns for table
- Table schema and contents are never directly accessed by other components
- Owning component usually provides interface to allow other components to query or modify selected table contents

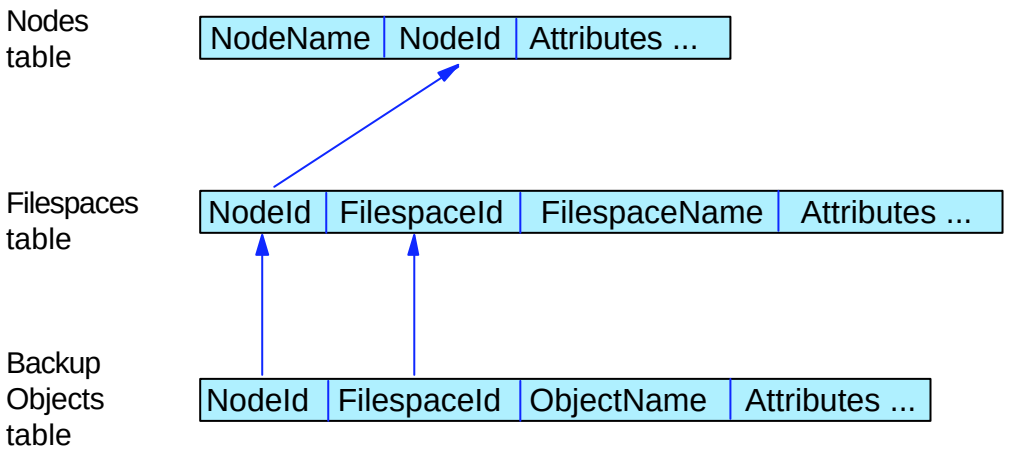
Database Table Functions

- Create
- Destroy
- Open
- GetAttr
- Extend
- Insert
- Update
- Delete
- Fetch
- SearchBounds
- FetchNext
- FetchPrev

Surrogate Keys

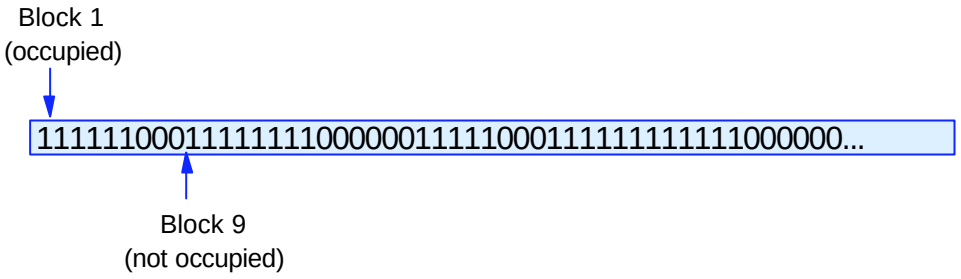
- Numeric identifiers are used to represent certain entities within the database
  - Nodes
  - Filespaces
  - Storage pools
  - Volumes
  - Objects
- Benefits
  - Faster compares
  - Efficient use of space
  - Ability to rename certain entities

Surrogate Keys (cont.)



Database Bit-Vector Objects

Linear array of bits representing allocation of 4KB blocks on disk storage pool volume



Tivoli

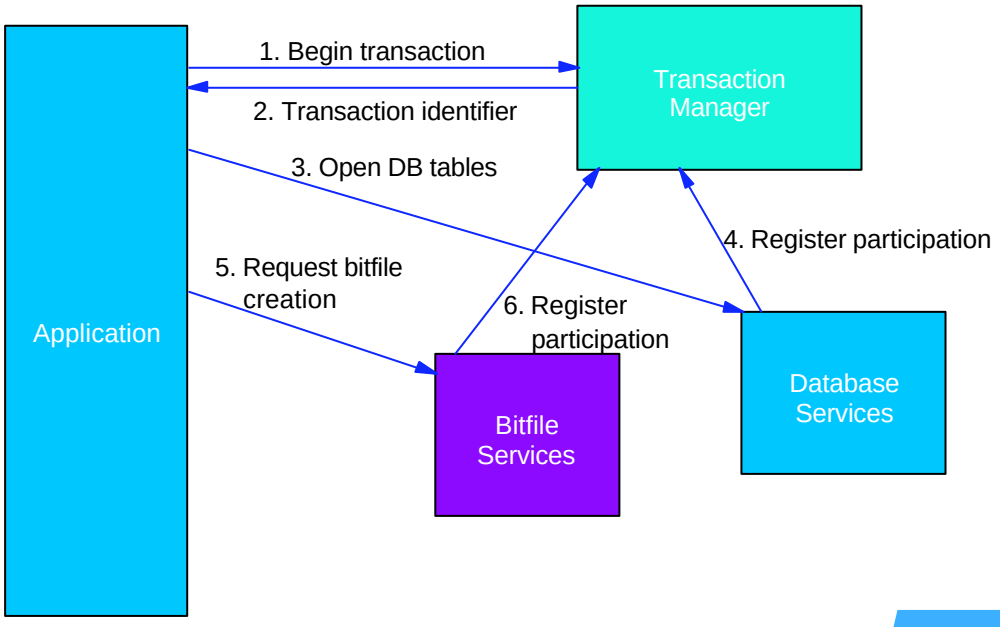
Transaction Management

Tivoli

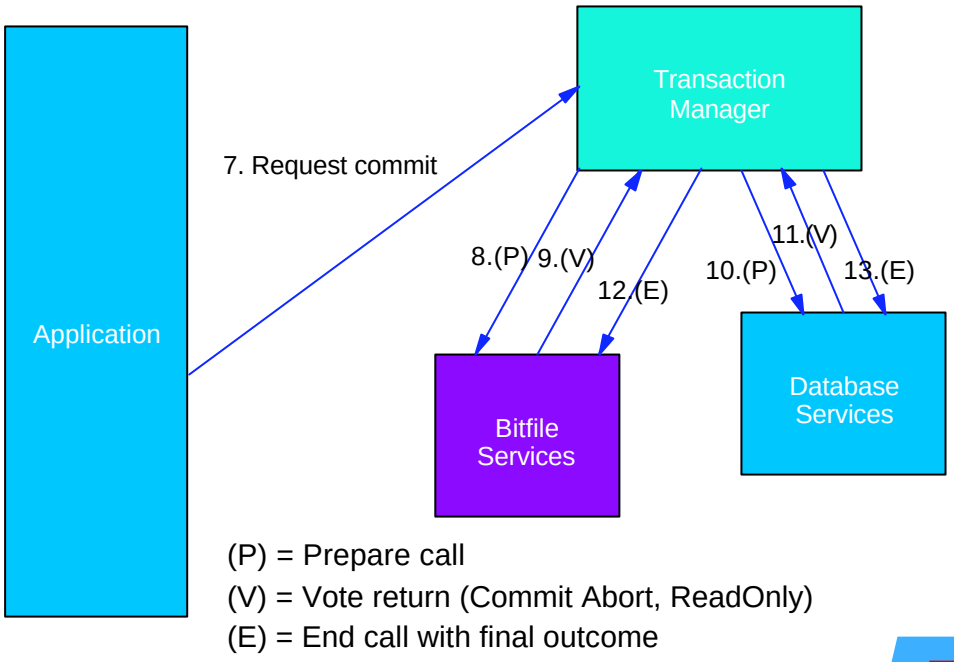
What Transaction Manager Does

- Transaction Manager provides mechanisms to coordinate concurrent database activity and thereby preserve referential integrity
- Transactions allow atomic database changes
  - Changes are performed in their entirety (commit) or not at all (rollback)
  - Uses two-phase commit protocol (prepare and end)
- Generalized locking semantics
  - Used primarily to synchronize concurrent transactions
  - Provides concurrency control for database

Two-Phase Commit



Two-Phase Commit (cont.)

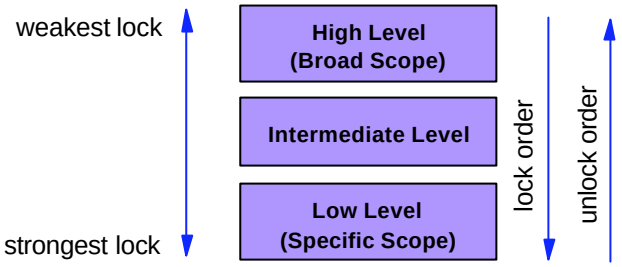


Vote Collection

- Votes are collected from all *volatile* participants before *nonvolatile* participants
  - Database is the only nonvolatile component
  - Volatile participants may make database updates in the prepare phase
  - Database component votes last, after all database updates have been made
- End-phase actions must be guaranteed to succeed (e.g. memory update)

Locks

- Transaction manager supports locking of logical entities, rather than real tables or rows
- Each component supports its own locking hierarchy
  - Scope is broad at highest level and specific at lower levels
  - Allows locks of different strengths at each level
- All operations must follow correct lock/unlock order
- Locks are associated with transactions and are released at commit/abort



Lock Compatibility Table

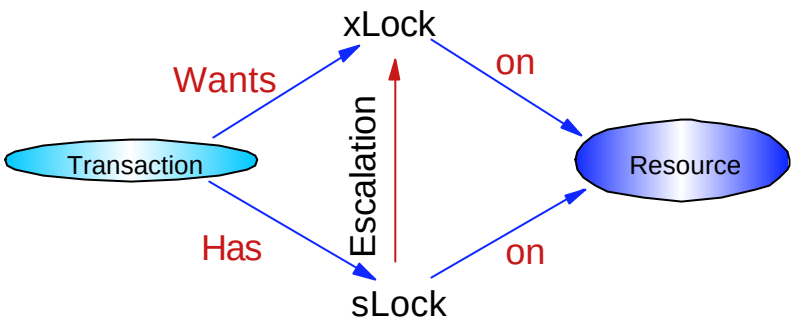
Increasing strength

Increasing strength

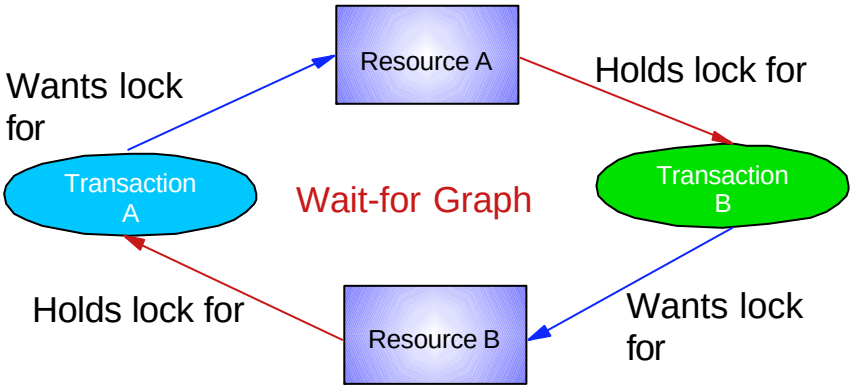
	isLock	ixLock	sLock	xixLock	sixLock	xLock
isLock	Yes	Yes	Yes	Yes	Yes	No
ixLock	Yes	Yes	No	Yes	No	No
sLock	Yes	No	Yes	No	No	No
xixLock	Yes	Yes	No	No	No	No
sixLock	Yes	No	No	No	No	No
xLock	No	No	No	No	No	No

Lock Escalation

- Occurs when a transaction requests a stronger lock on a resource on which the transaction already has a (weaker) lock
- If the requested lock mode is not compatible with another lock holder, the acquisition will fail with lock conflict error

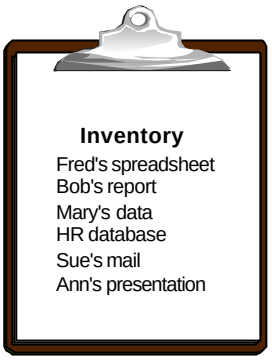


Deadlock



- Deadlock detector looks for and resolves transaction deadlocks by failing the oldest transaction
- Does not detect deadlocks involving other resources (e.g., mount points)

Inventory Management



Tivoli

What Inventory Manager Does

- Maintains a database catalog of "end-user-visible" attributes (node, filespace, object name, policy) for client objects managed by the server
- Maps each object to a unique identifier (object id) that is used for operations on that object
- Via expiration, administers policy rules for retention and versioning of client objects
- Maintains and administers filespace information, including authorization rules

Inventory Objects

- Inventory Manager tracks objects managed by the server
  - Directories
  - Files
  - File system images
  - Delta objects (subfiles)
- Each distinct object is assigned an object identifier
  - Unique 64-bit integer (unsigned)
  - Usually represented as x.y (where x and y are 32-bit integers that represent high- and low-order parts of id)
  - Does not change as long as object is managed by this server (new object ids are assigned during import)
  - Required for most operations on that specific object (e.g., restore/retrieve, deletion)

Object Information

- Node Id
- Filespace Id
- Object Type (directory, file, etc.)
- High-level name (path)
- Low-level name (object name)
- Management class id
- Object Id
- Insertion date/time (when object was stored on server)
- Deactivation date/time (inactive backup objects)
- Expiration base date (archive and inactive backup)
- Client-specific object information (e.g., permissions)
- Owner name
- Description (archive objects)
- Is object stored in storage pool?
- Delta-base and other group relationships

Versioning of Backup Objects

- During backup
  - If no corresponding object on the server, new object becomes the active version
  - If corresponding object already exists on the server, new object becomes active version and the existing active version is deactivated
  - Extraneous versions are marked for expiration
- The number of allowed versions of a backup object is determined by the object's management class and copy group

Backup Object Versioning Example

Operation	Result on Server
Client backs up file report.txt.	File report.txt is inserted in server at 12:00:00 01/01/2000 (objId=0.1).
End user updates report.txt.	None
Client does an incremental backup and report.txt is backed up again.	File report.txt is inserted in server at 14:00:00 01/01/2000 (objId=0.2). Object 0.1 is deactivated.
End user updates report.txt.	None
Client does another incremental backup and report.txt is backed up again.	File report.txt is inserted in the server at 08:00:00 01/02/2000 (objId=0.3) and object 0.2 is deactivated.

File Name	Insertion Date	Object Id	State
report.txt	12:00:00 01/01/2000	0.1	Inactive
report.txt	14:00:00 01/01/2000	0.2	Inactive
report.txt	08:00:00 01/02/2000	0.3	Active

Expiration

- Deletes qualifying backup or archive objects based on copy group attributes
- Database entries for expired objects are removed from the inventory and from storage-related tables in BF/SS

Versioning vs. Retention

- During backup, extraneous versions are marked for expiration (base date set to minus infinite)
  - VEREXISTS (backup object that exists on client system)
  - VERDELETED (backup object deleted from client system)
- During expiration, object is deleted if retention period for the object has been exceeded
  - RETEXTRA (inactive backup object with other versions)
  - RETONLY (inactive backup object with no other versions)
  - RETVER (archive object)
- Object becomes eligible for expiration as soon as either version or retention criterion is satisfied

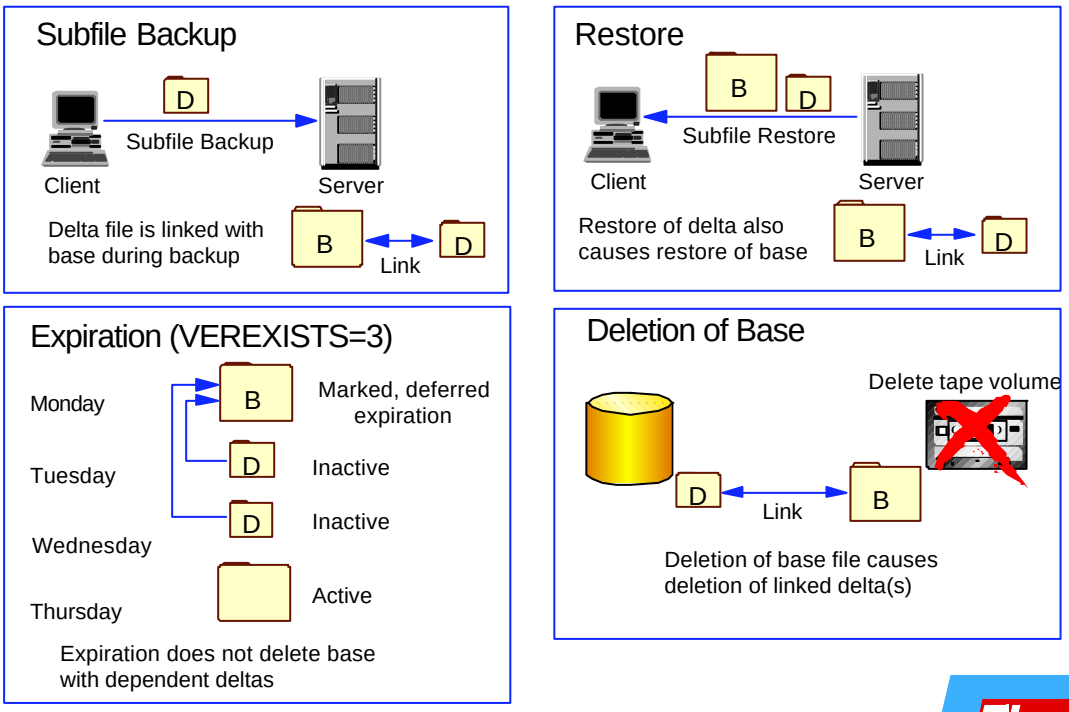
Example of Versioning and Retention

- VEREXISTS: 2
- RETEXTRA: 30

Insertion Date	Object Id	State	Expiration Base Date
12:00:00 01/01/2000	0.1	Inactive	Minus infinite
14:00:00 01/01/2000	0.2	Inactive	08:00:00 01/02/2000
08:00:00 01/02/2000	0.3	Active	None

- Object 0.1 is an extraneous version and would be expired during the next expiration operation
- Object 0.2 would be eligible for expiration on 08:00:00 02/01/2000 (30 days after the base date)

Delta-Base Object Relationships



Show Versions

```
tsm: SERVER1>show versions client1 \\cannonhome\d$ \TSM\TSM421\BF\
TESTFILE.C

\\cannonhome\d$ : \TSM\TSM421\BF\ TESTFILE.C (MC: DEFAULT)
Active, Inserted 08/25/2001 14:50:36
ObjId: 0.1037

\\cannonhome\d$ : \TSM\TSM421\BF\ TESTFILE.C (MC: DEFAULT)
Inactive, Inserted 08/25/2001 14:47:49, Deactivated ** EXPIRE IMMEDIATELY
**
ObjId: 0.1035  GroupType: Delta-base  Group leader

\\cannonhome\d$ : \TSM\TSM421\BF\ TESTFILE.C (MC: DEFAULT)
Inactive, Inserted 08/25/2001 14:48:49, Deactivated 08/25/2001 14:50:36
ObjId: 0.1036  GroupType: Delta-base  GroupId: 0.1035
```

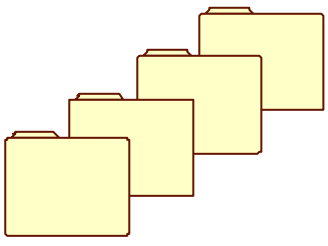

Show Invobject

```
tsm: SERVER1>show invo 0 1036

OBJECT: 0.1036 (Backup):
  Node: CLIENT1 Filespace: \\cannonhome\d$.
  \TSM\TSM421\BF\ TESTFILE.C
  Type: 2 (File)  CG: 1  Size: 0.69632  HeaderSize: 398

BACKUP OBJECTS ENTRY:
  State: 2  Type: 2  MC: 1  CG: 1
\\cannonhome\d$ : \TSM\TSM421\BF\ TESTFILE.C (MC: DEFAULT)
Inactive, Inserted 08/25/2001 14:48:49, Deactivated 08/25/2001 14:50:36
GroupType: Delta-base(1), GroupId: 0.1035

EXPIRING OBJECTS ENTRY:
  Copy Type: Backup
  MC: 1  CG: 1
  BaseDate: 08/25/2001 14:50:36
```

Show Groupmembers

```
tsm: SERVER1>show groupm 0 1035
Group 0.1035 contains the following members
0.1036
```


Bitfiles



-

What Bitfile Service Does

- Manages bitfiles (client objects in storage pools)
- Supports aggregation of logical files
- Maintains metadata and summary information relating to bitfiles
- Tracks and exploits bitfile copies in multiple storage pools (cached copies and backups in copy storage pools)
- Performs operations on a single bitfile
- Controls bitfile-transfer operations

General Terminology

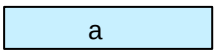
- **Bitfile:** Client object (or group of objects) in a storage pool. Some objects have an inventory entry, but no corresponding bitfile.
- **Cluster:** Data belonging to a specific combination of NodeId (cluster key 1) and FilespaceId (cluster key 2). Bitfiles may be transferred by cluster, especially if the target storage pool is collocated.

Description of Aggregation

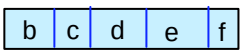
- TSM server groups client objects into aggregates during backup or archive
- Information about individual client objects is maintained and used for certain operations (e.g., deletion, retrieval)
- For many operations, entire aggregate can be processed as a single entity
- Aggregate size and number of client objects per aggregate can be controlled

Aggregate Terminology

- **Logical file:** Client object stored in storage pool. Each logical file has an identifier, corresponding to the object identifier.
- **Physical file:** Server bitfile consisting of either
  - Single, non-aggregated logical file
  - Multiple logical files aggregated together
- **Aggregate:** Physical file composed of multiple logical files



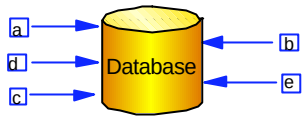
Physical file (non-aggregate) with logical file a



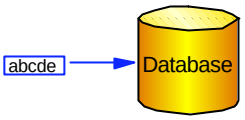
Physical file (aggregate) with logical files b - f

Performance Benefits of Aggregation

- Reduced overhead for database updates (storage information not updated for each logical file during move/copy operations)

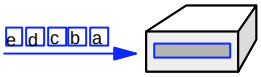


Without aggregation

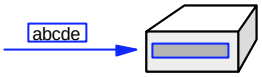


With aggregation

- Improved storage device performance because data is transferred in larger units



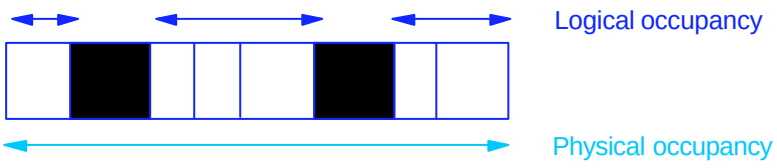
Without aggregation



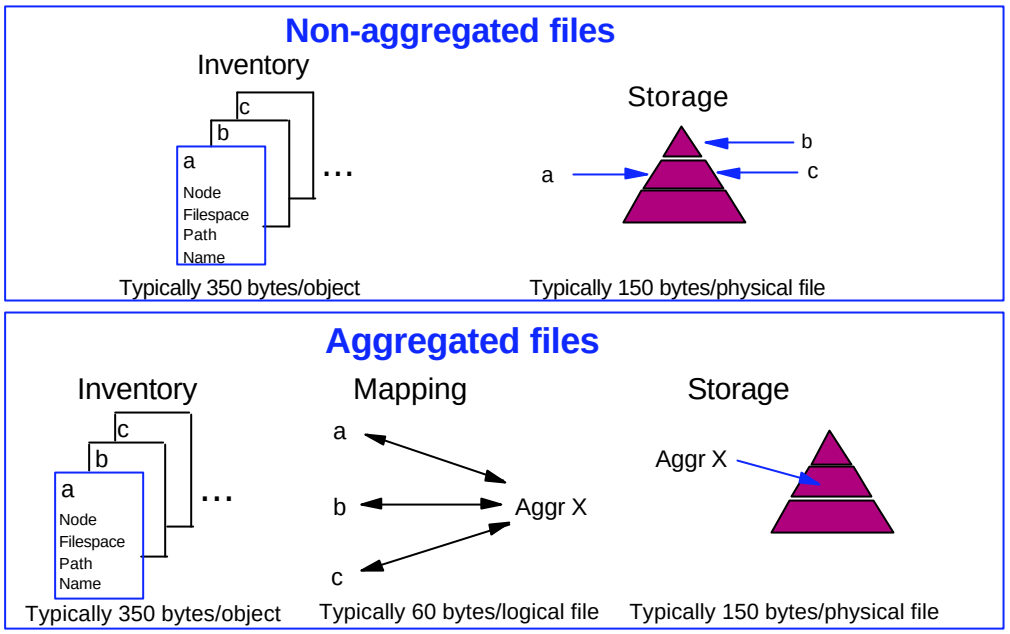
With aggregation

Physical vs. Logical Occupancy

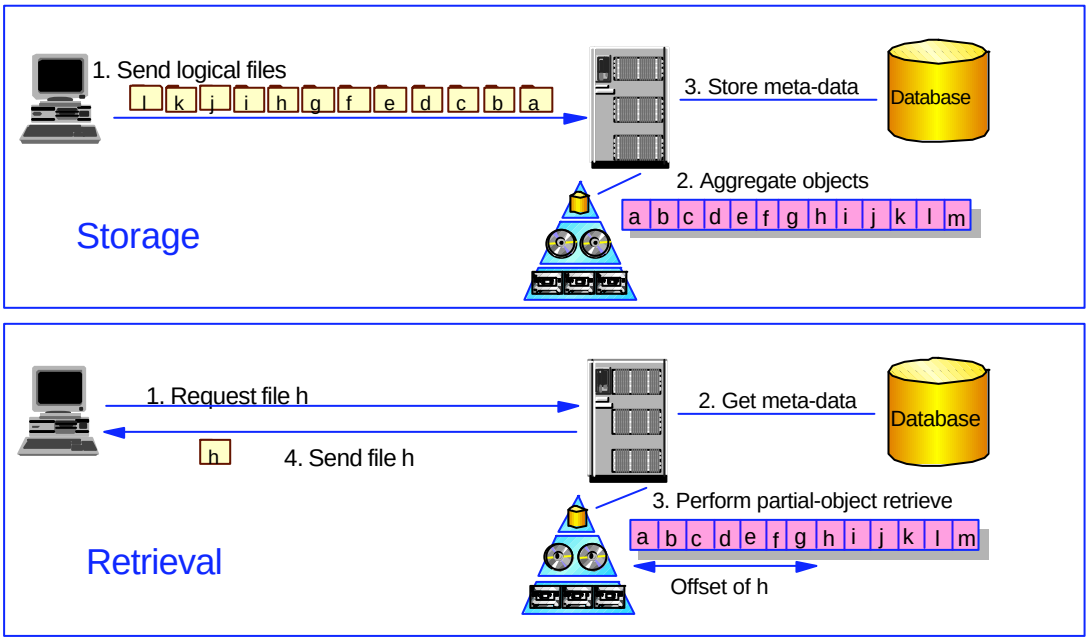
- **Physical occupancy:** Space required for storage of physical files on the server
- **Logical occupancy:** Space required for storage of logical files on server (does not include vacant space within aggregates)



Database Information for Stored Files



Logical File Storage and Retrieval

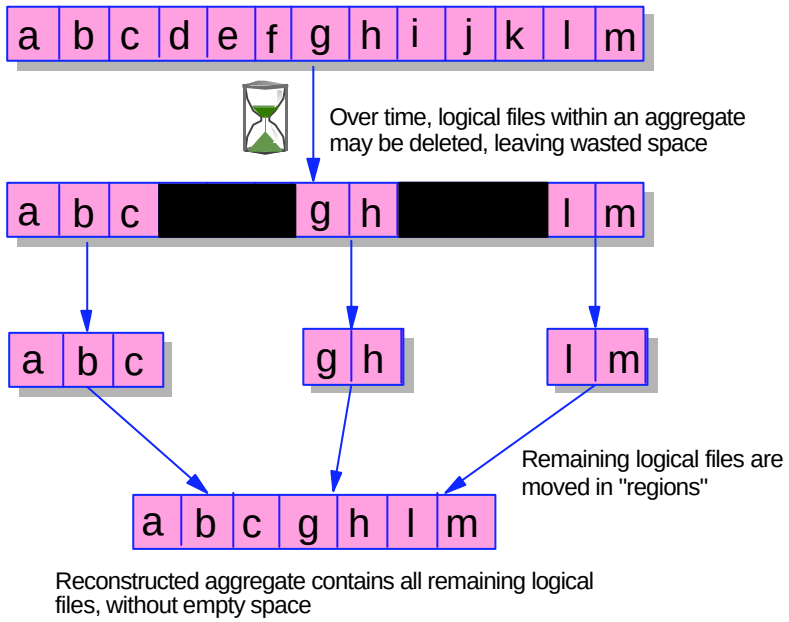


Determining Aggregate Size

- Based on transaction boundaries
  - Client control via TXNBYTELIMIT option
  - Server control
    - TXNGROUPMAX option
    - 25 MB maximum size
- Aggregates are homogenous with respect to
  - Node
  - Filespace
  - Copytype (backup, archive, space managed)
  - Destination storage pool

Tivoli

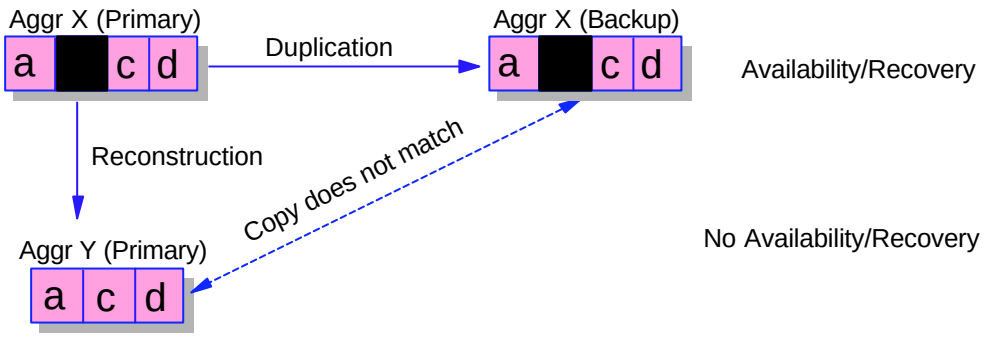
Aggregate Reconstruction



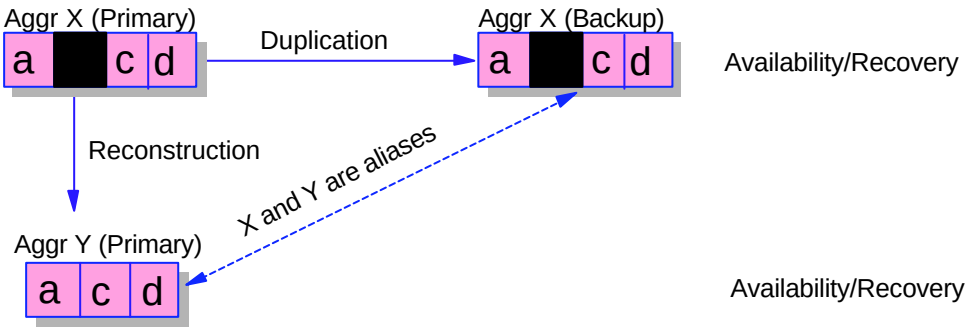
Bitfiles

Tivoli

Problem: Invalidation of Copies

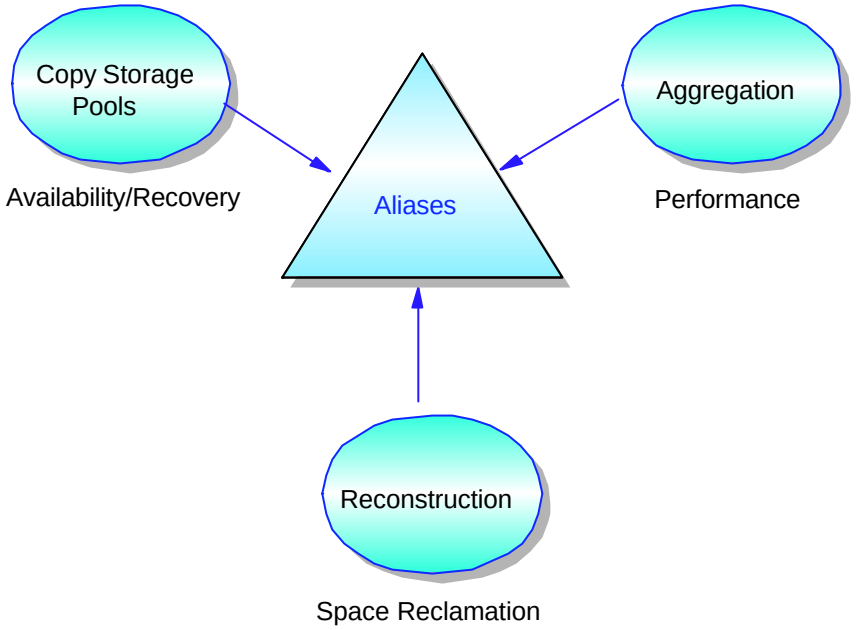


Solution: Aggregate Aliases



- New backup not required
- Files in X can be used if Y unavailable
- Y can be restored from X

Benefits of Aggregate Aliases



Show Bfobject

```
adsm> show bfo 0 19463

Bitfile Object: 0.19463
**Super-bitfile 0.19463 contains following aggregated bitfiles
  0.19463
  0.19465
  0.19466
  0.19467
  0.19468
  0.19469
  0.19470
  0.19472
  0.19473
  0.19474

**Sub-bitfile 0.19463 is stored in the following aggregate(s)
  Super-bitfile: 0.19463, Offset: 0.0, Length 0.47004

**Archival Bitfile Entry
  Bitfile Type: PRIMARY  Storage Format: 9
  Bitfile Size: 0.801383  Number of Segments: 1
  Storage Pool ID: 4  Volume ID: 10  Volume Name:
D:\TSM\TSM370\BUILD\0000000A.BFS
**Archival Bitfile Entry
  Bitfile Type: COPY  Storage Format: 9
  Bitfile Size: 0.801383  Number of Segments: 1
  Storage Pool ID: -1  Volume ID: 11  Volume Name:
  D:\TSM\TSM370\BUILD\0000000B.BFS
```

Show Bfobject (cont.)

```
adsm> show bfo 0 19474

Bitfile Object: 0.19474
**Sub-bitfile 0.19474 is stored in the following aggregate(s)
  Super-bitfile: 0.19463, Offset: 0.734860, Length 0.66363
```


Show Aggregate

```
adsm> show aggr 0 19463

AggrId: 0.19463, AggrSize: 0.801223, LogSize: 0.602009, NumFiles: 10.
Aliases: None.
```


Storage Service

Tivoli

What Storage Service Does

- Manages device classes, storage pools and storage-pool volumes
- Allocates space in disk storage pools using database bit vectors
- Selects sequential-access volumes
- Stores, retrieves, and transfers bitfile segments

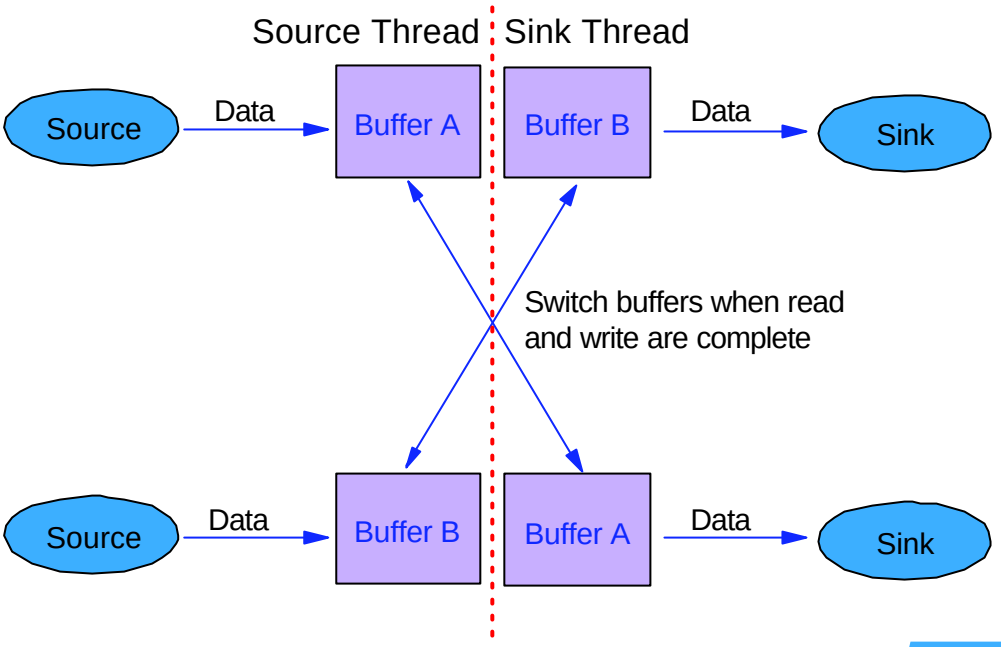
Terminology

- **Segment:** Space occupied on a volume by a physical bitfile
- **Segment Group:** One or more segments comprising a physical bitfile
- **Partial-Object Retrieve:** Provides the ability to index into a segment group and retrieve a portion of that group

Data-Transfer Operations

- Store (backup, archive, space management, import, backupset generation)
- Retrieve (restore, retrieve, recall, export)
- Copy (migration, reclamation, storage pool backup/restore)
- Reconstruct (removal of empty space in aggregates)
- Clone (backup of space-managed files)

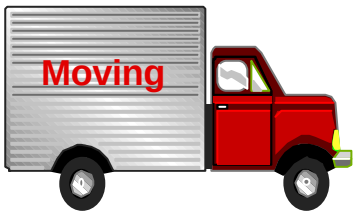
Data-Transfer Threads and Buffers



Storage Service

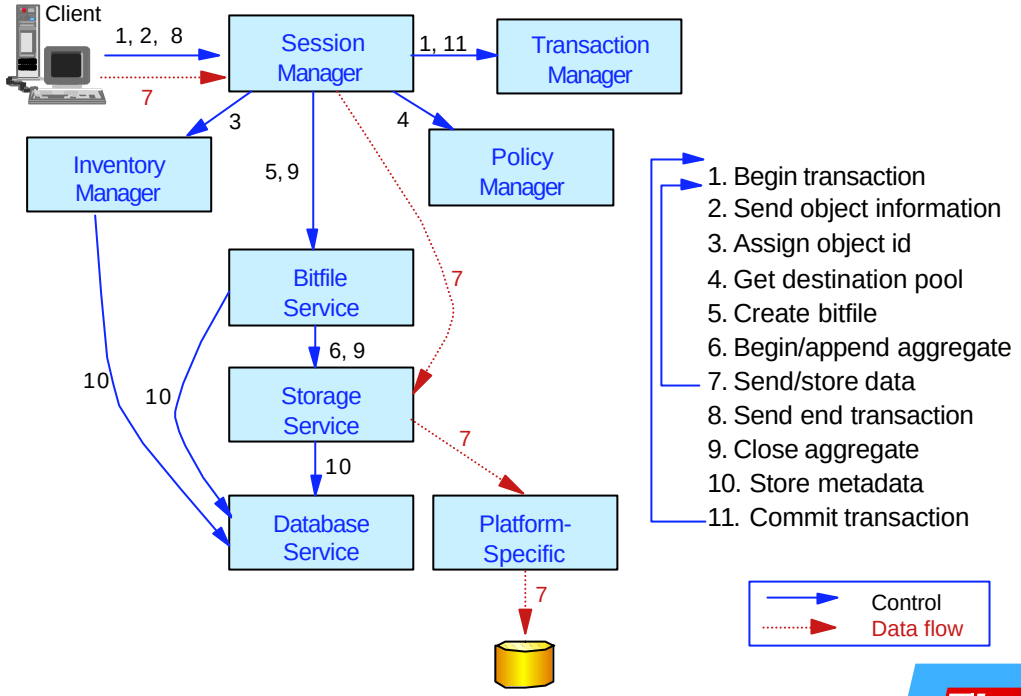
Tivoli

Data-Transfer Examples

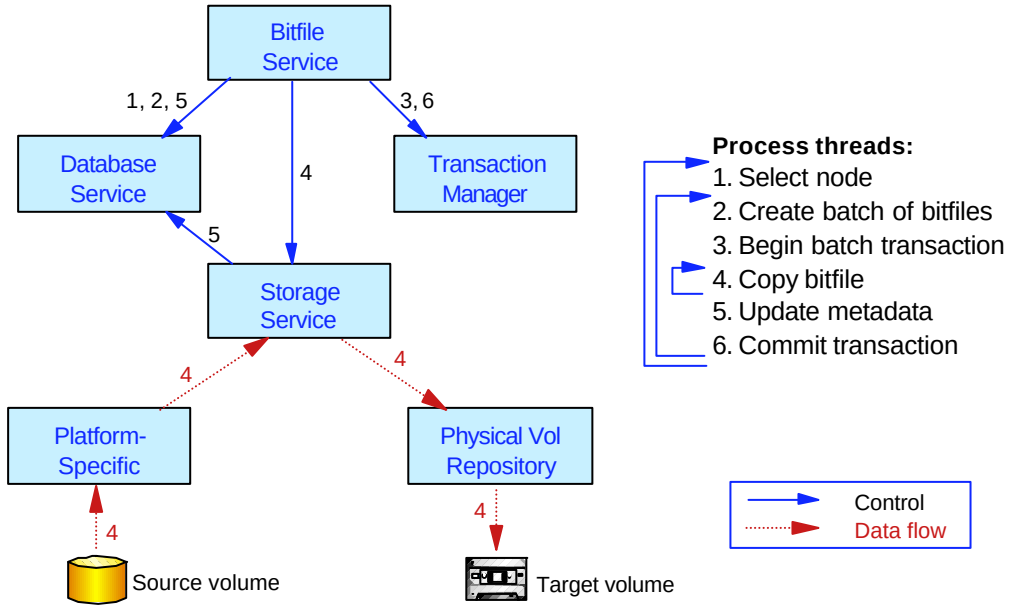


Tivoli

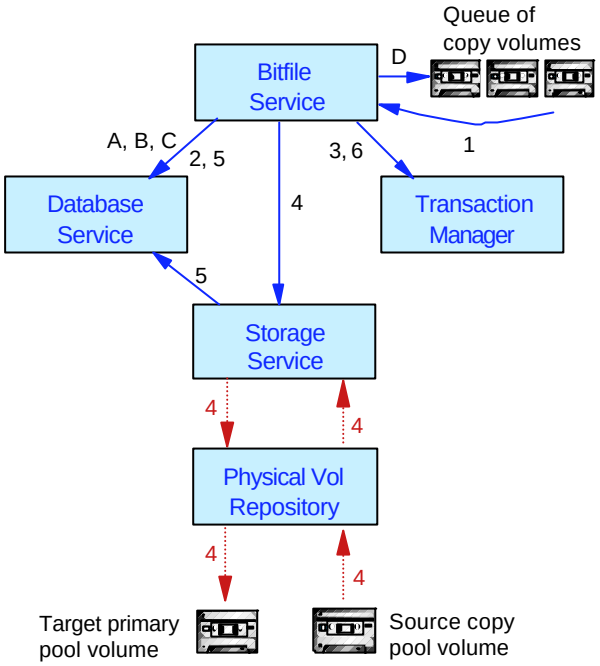
Client Backup to Disk



Migration/Backup from Disk Pool



Restore of Sequential Pool/Volume



Queue thread
A. Check primary volume
B. Scan bitfile entry on volume
C. Find best copy volume
D. Add copy volume to queue

Process threads:
1. Process queued volume
2. Create batch of bitfiles
3. Begin batch transaction
4. Copy bitfile
5. Update metadata
6. Commit transaction

Control
Data flow

Show Bfstats

```
adsm> show bfstat client*

Bitfile storage statistics from last backup/archive session.
Node CLIENT1:
  Total physical files stored: 83
  Average logical files per physical file: 32.4
  Average physical file size: 1870.2 KB
Node CLIENT2:
  Total physical files stored: 33
  Average logical files per physical file: 12.0
  Average physical file size: 2415.6 KB
```

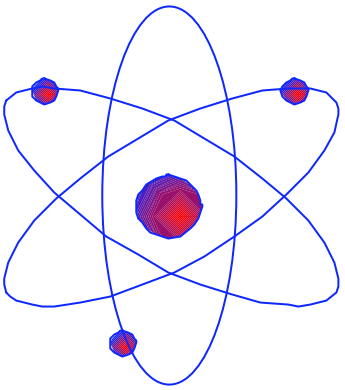
Show Transferstats

```
adsm> show transferstat backuppool
Statistics for last migration from pool BACKUPPOOL
   Start date/time: 07/08/2000 16:44:58
   Elapsed time: 164 seconds
   Total wait time: 1 seconds
   Number of participating processes: 2
   Total duration of all processes: 233 seconds
   Total physical files: 112
   Total logical files: 2970
   Total bytes: 199086080
   Average logical files per physical file: 26.5
   Average physical file size: 1735.9 KB
   Number of batch/file transactions ended: 6
   Number of batch transactions aborted: 0
   Number of file transactions started: 0
   Number of file transactions aborted: 0
Statistics for last storage pool backup of BACKUPPOOL
   Target copy storage pool: COPYPOOL
   Start date/time: 07/08/2000 16:38:19
   Elapsed time: 155 seconds
   Total wait time: 1 seconds
   Number of participating processes: 2
   Total duration of all processes: 220 seconds
   Total physical files: 112
   Total logical files: 2970
   Total bytes: 199086080
   Average logical files per physical file: 26.5
   Average physical file size: 1735.9 KB
   Number of batch/file transactions ended: 6
   Number of batch transactions aborted: 0
```


Show Transferstats (cont.)

```
adsm> show transferstat filepool

No migration information available for pool FILEPOOL.

No storage pool backup information available for FILEPOOL.

Statistics for last reclamation from pool FILEPOOL
   Volume reclaimed: D:\TSM\TSM370\BUILD\0000000E.BFS
   Start date/time: 07/08/2000 16:53:27
   Elapsed time: 26 seconds
   Total wait time: 0 seconds
   Total physical files: 32
      Copied physical files: 2
      Reconstructed physical files: 30
   Total logical files: 205
      Copied logical files: 2
      Reconstructed logical files: 203
   Total bytes (copied and pre-reconstruction): 65979755
   Average logical files per physical file: 6.4
   Average size of copied and pre-reconstruction physical files: 2013.5 KB
   Total reconstruction regions: 149
   Average reconstruction regions per physical file: 5.0
   Number of batch/file transactions ended: 2
   Number of batch transactions aborted: 0
   Number of file transactions started: 0
   Number of file transactions aborted: 0
```

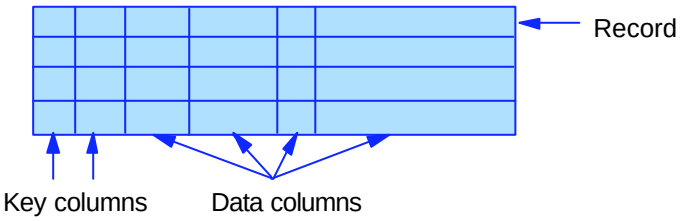
Database and Recovery Log: Structure



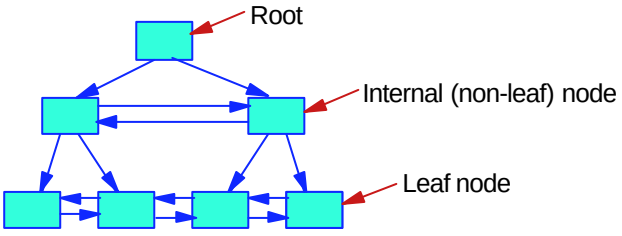
Tivoli

Database Table Objects

From an application perspective, table schema describes key and data columns for table



Within the database, table structure is a balanced tree (B^+-tree), whose nodes are stored as database pages



Database and Log: Structure

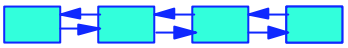
Tivoli

Database Bit-Vector Objects

From an application perspective, a bit-vector is a linear array of bits which is used to represent allocation of blocks on a disk storage pool volume

11111100011111111000000111110001111111111111000000...

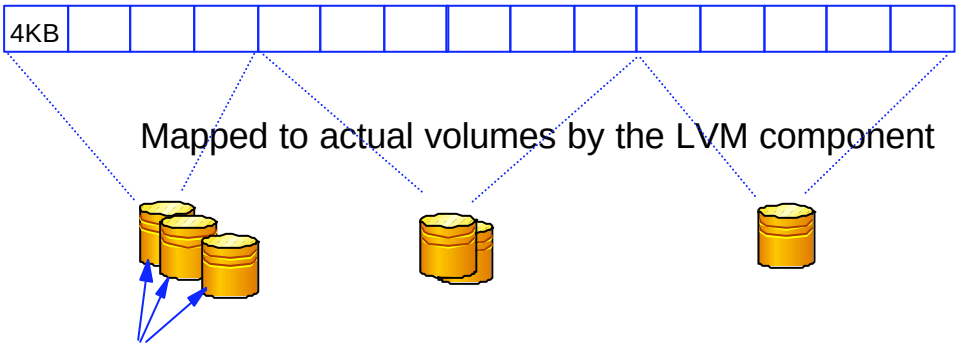
Within the database, a bit-vector is stored as linked pages



Tivoli

Overall DB/Log Structure

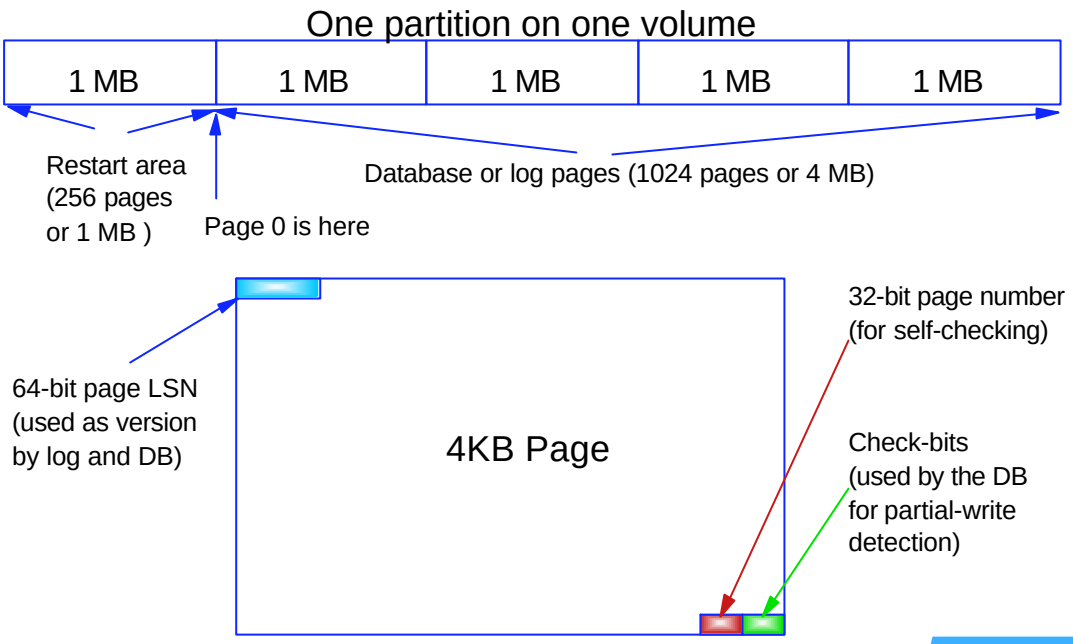
DB/Log: Linear address spaces of 4KB pages



- Can be mirrored (up to 3 ways)
- Allocated in 4MB partitions
- First 1 MB on each volume is used by LVM to record names of all volumes and mirrors
- Volume with two partitions requires 9MB

Tivoli

Overall DB/Log Structure (cont.)



Database Structure

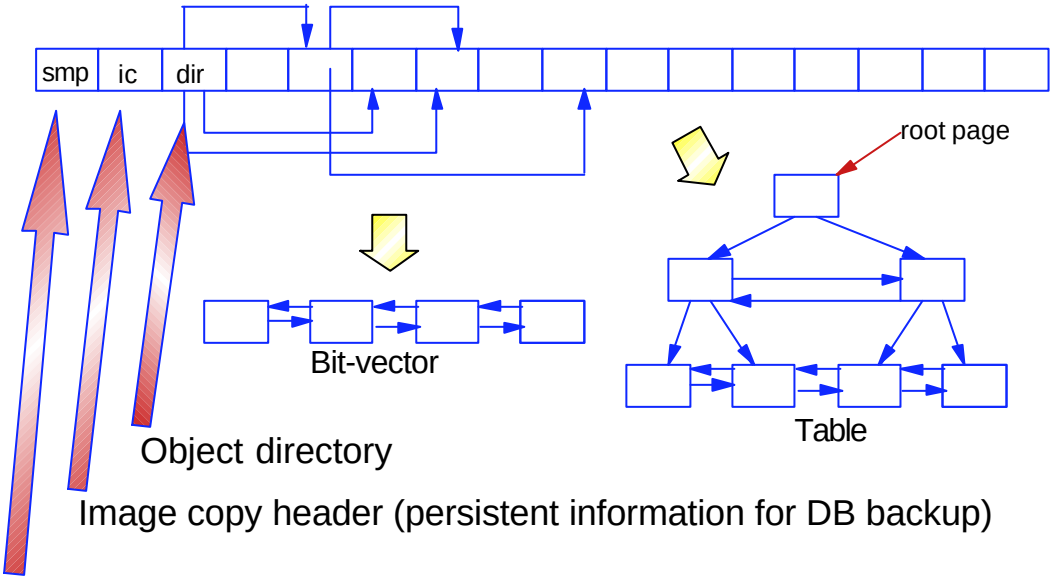
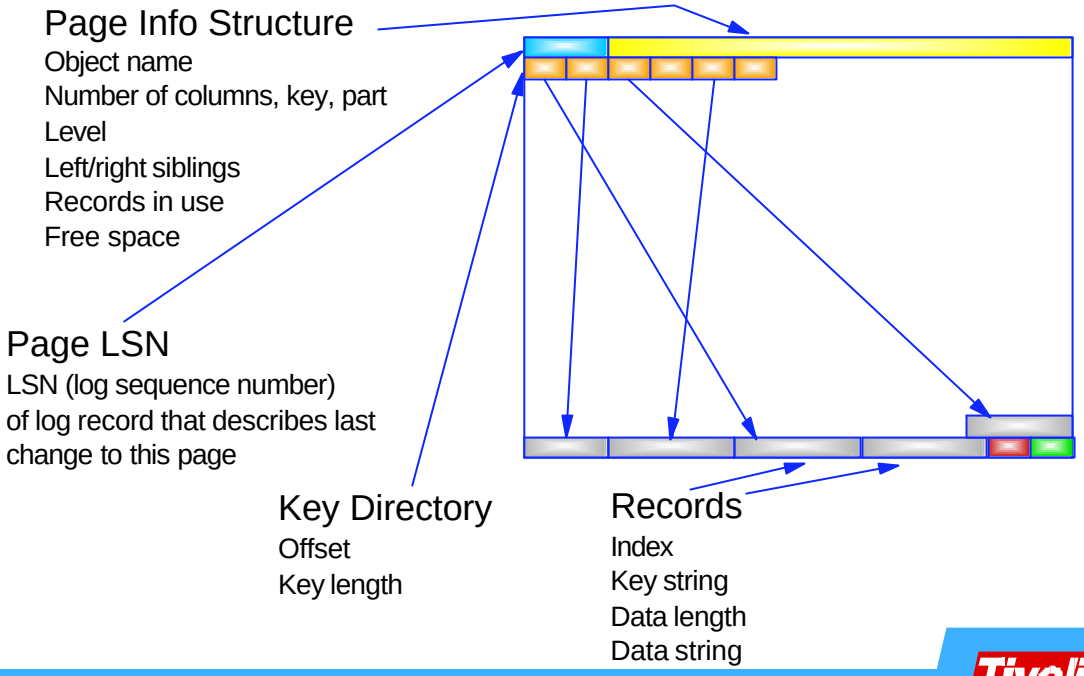


Image copy header (persistent information for DB backup)

Space map page every 4MB (tracks allocated pages)

Database Page Layout



Tivoli

Database Navigation

- SHOW OBJDIR displays the object directory (page 2)
- SHOW NODEHEADER displays node/page header information
- SHOW NODE displays node records
- SHOW BV displays bit-vector information
- SHOW SMP displays space map page

Tivoli

Show Objdir

```
adsm> show objdir
Defined Database Object Names (Home Address):
ADM.SSL.Keyring(60), AF.Backup.Optimization(36), AF.Bitfiles(29),
AF.Clusters(30), AF.Damaged(35), AF.Segments(32), AF.Vol.Clusters(31),
AF.Vol.Partial.Bitfiles(34), AF.Vol.Segments(33), AS.Segments(14),
AS.Volume.Assignment(16), AS.Volume.Status(15), Activity.Log(49),
Activity.Summary(50), Administrative.Attributes(37), Administrators(40),
Analyst.Administrators(43), Arch.Objs.ByDescription(74),
Archive.Descriptions(73), Archive.Objects(72), Authorization.Rules(81),
BF.Aggregate.Attributes(20), BF.Aggregated.Bitfiles(18), BF.Attributes(17),
BF.Super.Bitfiles(19), Backup.Objects(71), Central.Config.Attributes(123),
Client.Administrators(46), Client.Eventrules(53), Client.Optionset(54),
Clientopt.Members(55), Copy.Group.Ids(69), Copy.Groups(67), DF.Bitfiles(21),
DF.CachedBitfiles(25), DF.Clusters(22), DF.Damaged(28), DF.MigrBitfiles(24),
DF.MigrCandidates(23), DF.Segments(26), DF.Vol.Contents(27),
DRM.Attributes(107), DRM.Mach.Char(113), DRM.Mach.Node(110),
DRM.MachName.MachId(109), DRM.Machine.Definition(108), DRM.Rec.Inst(114),
DRM.Recov.Media(111), DRM.Rm.Mach.Association(112), DS.Overflow(11),
DS.Segments(10), DS.Shared.Blocks(12), DS.Volume.Status(13),
DSKV0000000002(130), DSKV0000000003(160), DSKV0000000005(270),
DSKV0000000008(170), Expiring.Objects(78), Extended.Attrs(39), Filespaces(79),
```


Tivoli

Show Node

```
adsm> show node 71

B-Tree ROOT node: PageAddr=71, Lsn=20.183.118, ObjName=Backup.Objects
LeftSib=-1, RightSib=-1, Continuation=-1
NumRecs=3, Free=973, NextSlot=35, DirOff=0, PartKeyLen=0
Level=2, NumCols=19, KeyCols=8, PartCols=4, NodeAttr=80
MaxCapacity=1004, Capacity=1004, Occupancy=31, LowOccupancy=334
  Record 0 (KeyLen=40, DataLen=4):
    Key:  ->(01000000)(06000000)(01)(02)(\TEST\SMALLFILES\)(PSTAPE.H)(01000000)
(000000004F360000)<-
    Subtree=<317>
  Record 1 (KeyLen=38, DataLen=4):
    Key:  ->(01000000)(07000000)(01)(02)(\GEMACS\LISP\)(GOMOKU.ELC)(01000000)(-
00000000C9400000)<-
    Subtree=<284>
  Record 2 (KeyLen=8, DataLen=4):
    Key:  ->(+Infinite)(+Infinite)(+Infinite)(+Infinite)(+Infinite)(+Infinite)(
+Infinite)(+Infinite)<-
    Subtree=<267>
```

Database and Log: Structure

Tivoli

Show Node (cont.)

```
adsm> show node 267
B-Tree NONLEAF node: PageAddr=267, Lsn=21.126.661, ObjName=Backup.Objects
LeftSib=284, RightSib=-1, Continuation=-1
NumRecs=55, Free=196, NextSlot=250, DirOff=0, PartKeyLen=0
Level=1, NumCols=19, KeyCols=8, PartCols=4, NodeAttr=80
MaxCapacity=1004, Capacity=1004, Occupancy=808, LowOccupancy=334
  Record 0 (KeyLen=39, DataLen=4):
    Key:  ->(01000000)(07000000)(01)(02)(\GEMACS\LISP\)(HIDESHOW.EL)(01000000)(
00000000DC400000)<-
    Subtree=<286>
  Record 1 (KeyLen=40, DataLen=4):
    Key:  ->(01000000)(07000000)(01)(02)(\GEMACS\LISP\)(INF-LISP.ELC)(01000000)
(00000000EF400000)<-
    Subtree=<289>
  Record 2 (KeyLen=41, DataLen=4):
    Key:  ->(01000000)(07000000)(01)(02)(\GEMACS\LISP\)(ISO-TRANSL.EL)(01000000
)(0000000002410000)<-
    Subtree=<333>
  Record 3 (KeyLen=36, DataLen=4):
    Key:  ->(01000000)(07000000)(01)(02)(\GEMACS\LISP\)(LIFE.ELC)(01000000)(00-
00000015410000)<-
    Subtree=<334>
```


Show Node (cont.)

```
adsm> show node 289

B-Tree LEAF node: PageAddr=289, Lsn=20.205.2866, ObjName=Backup.Objects
LeftSib=286, RightSib=333, Continuation=-1
NumRecs=19, Free=35, NextSlot=53, DirOff=0, PartKeyLen=8
Level=0, NumCols=19, KeyCols=8, PartCols=4, NodeAttr=80
MaxCapacity=1004, Capacity=1002, Occupancy=967, LowOccupancy=334
Partition Key ->(01000000)(07000000)(01)(02)<-
  Record 0 (KeyLen=32, DataLen=162):
    Key:  ->(\GEMACS\LISP\)(HIDESHOW.ELC)(01000000)(00000000DD400000)<-
    Data: ->(d...7.)(Null)(....w........,..I..................... ....8-....
...9Q........_......!.....................................................
..........)(<empty string>)(01000000)(00000000)(Null)(Null)(Null)(Null)(Null)-
<-
  Record 1 (KeyLen=30, DataLen=162):
    Key:  ->(\GEMACS\LISP\)(HILIT19.EL)(01000000)(00000000DE400000)<-
    Data: ->(d...7.)(Null)(....w............I..................... ....8-....
...9Q........^......!.....................................................
..........)(<empty string>)(01000000)(00000000)(Null)(Null)(Null)(Null)(Null)-
<-
  Record 2 (KeyLen=31, DataLen=162):
    Key:  ->(\GEMACS\LISP\)(HILIT19.ELC)(01000000)(00000000DF400000)<-
    Data: ->(d...7.)(Null)(....w.........x$.I..................... ... F.....
...9Q........_......!.....................................................
..........)(<empty string>)(01000000)(00000000)(Null)(Null)(Null)(Null)(Null)-
<-
```

Database and Log: Structure

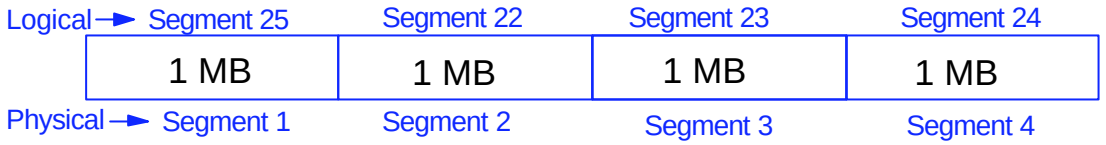
Show Bv

```
adsm> show bv 130

Dump of Bit Vector data page 130, Lsn=21.242.459:
PrevAddr=-1, NextAddr=-1
Capacity=5119, Low=0, Bits=5119
  Allocation information:
    Allocated 0 for 967
    Free      967 for 1
    Allocated 968 for 481
    Free      1449 for 1986
    Allocated 3435 for 1684
```


Show Smp

```
show smp 0 x
ANR2017I Administrator SERVER_CONSOLE issued command: show smp 0 x
pageLsn:      1.198.515
dataPageInfo: objType=01
              pageType=01
              objName=Db_Space_Map_Page
smpInfo:      smpNum=0
              zeroBits=706
              lowZero=318
alloc: ->FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF-
FFFFFFFFFFFC0000000000000000000000000000000000000000000000000000000000000000000000-
0000000000000000000000000000000000000000000000000000000000000000000000000000000000-
00000000000000000000000000000000<-
   ic: ->0000000000000000000000000000000000000000000000000000000000000000000000000-
0000000000000000000000000000000000000000000000000000000000000000000000000000000000-
0000000000000000000000000000000000000000000000000000000000000000000000000000000000-
00000000000000000000000000000000<-

lvmPageTrl:   pageAddr=0
              saveBits1=00000001, saveBits2=00000001

-> Next SMP is at page address 1024.
```

Recovery Log Structure



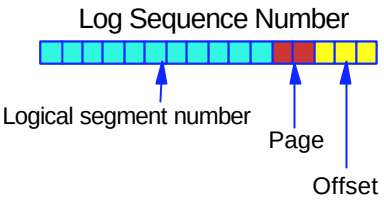
Segments:

Log is composed of segments, each containing 256 4KB pages
Logical segment numbers always increase
Log will wrap

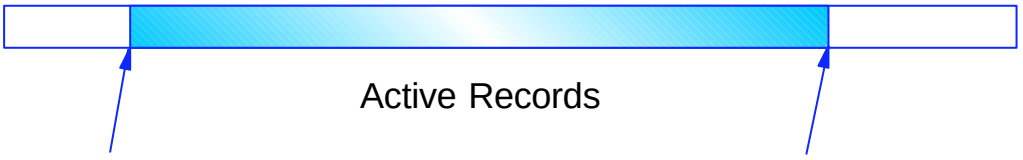
Log Sequence Number (LSN):

64 Bits
- Logical segment number : 44 bits
- Page : 8 bits
- Offset: 12 bits

Displayed as 3 dotted numbers: 23141.153.45



Recovery Log Tail and Head

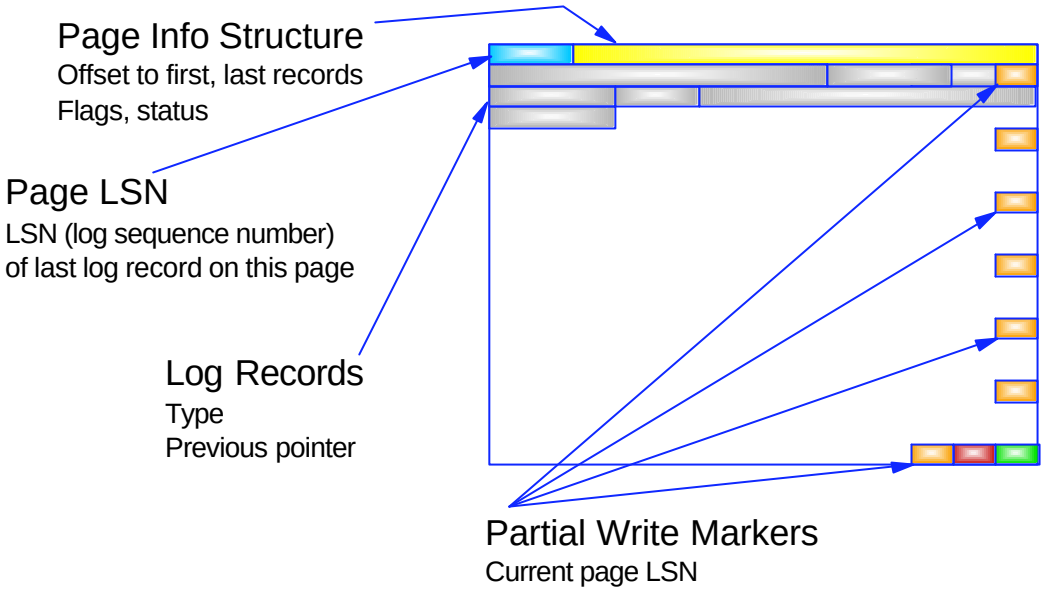


Log tail or truncation point
(oldest active log record)

Log head
(location for the next record to be written)

- Gaps exist in active record range
- Recovery log records can be compressed (moved or remapped)
- Location of log tail depends upon log mode

Recovery Log Page



Show Commands for the Log

- SHOW LOGSEG displays all log segments
- SHOW LOGVARS displays global settings for the log
- SHOW LSN displays log record for a specified LSN

Show Logseg

```
show logseg
ANR2017I Administrator SERVER_CONSOLE issued command:
show logseg

Log Segment Table:

  00: INUSE  BaseLsn=1.0.0  <-- Head (Page 209, Lsn 1.209.407)
  01: Empty  BaseLsn=0.0.0
  02: Empty  BaseLsn=0.0.0
  03: Empty  BaseLsn=0.0.0
  04: Empty  BaseLsn=0.0.0
  05: Empty  BaseLsn=0.0.0
  06: Empty  BaseLsn=0.0.0
  07: Empty  BaseLsn=0.0.0
```


Tivoli

Show Logv

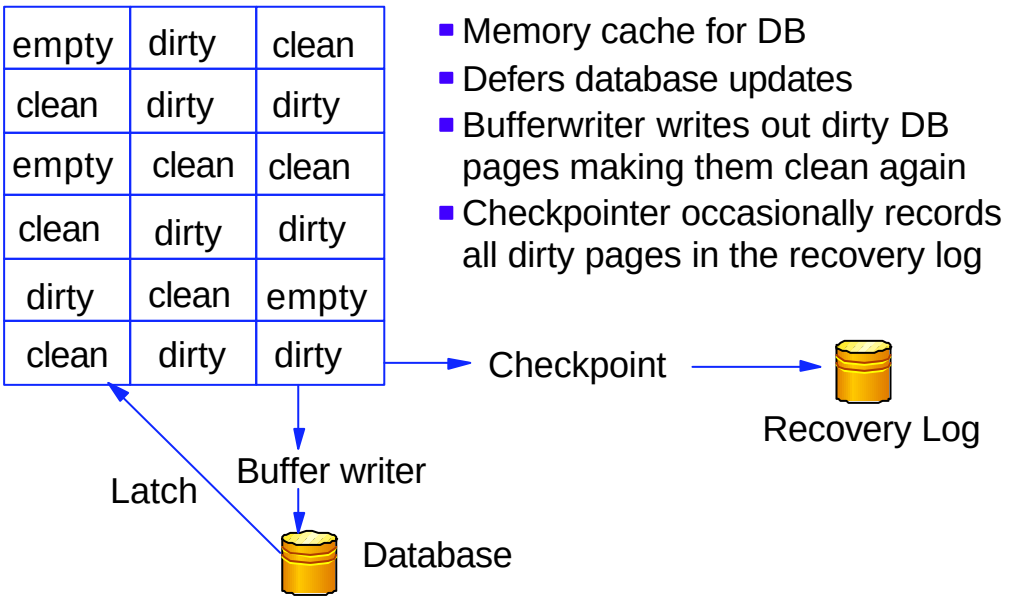
```
show logv
Log Global Variables:
SegTableSize=8, ResizeLocked=False, EmptySegs=7, PagesReserved=33, MaxPagesUse-
d=244, PagesInUse=243, LastReportedFull=88
  HeadLsn:      1.209.407  (segment 0, page 209)
  StableLsn:    1.209.407  (segment 0, page 209)
  TruncLsn:     1.207.16  (segment 0, page 207)
  ScanLsn:      1.207.16
  ScanEndLsn:   1.207.16

Log Restart Record:
VersId=1, RecLen=88, InstallId=39299459, FmtCapacity=8, Capacity=8, NewCapacit-
y=8 FmtDate=959026265, ResizeDate=0, MovePhase=0, SourceSeg=-1, TargetSeg=-1,
consumedLsn=1.0.0, consumedReset=959026265

  TruncBaseLsn: 1.0.0  (segment 0, page 0)
```


Tivoli

Show Lsn

```
show lsn 1.207.16
Lsn=1.207.16, Owner=DB, Length=4, NextLsn=1.207.36
Type=BeginCkpt, Flags=00

show lsn 1.207.36
Lsn=1.207.36, Owner=DB, Length=4, NextLsn=1.207.56
Type=EndCkpt, Flags=00

show lsn 1.207.56
Lsn=1.207.56, Owner=DB, Length=164, NextLsn=1.207.236
Type=Update, Flags=C2, Action=ExtDelete, Page=37, Tsn=0:6148, PrevLsn=0.0.0,
UndoNextLsn=0.0.0, UpdtLsn=1.206.1734 ===> ObjName=Administrative.Attributes,
Index=0, RootAddr=37, PartKeyLen=0, NonPartKeyLen=8, DataLen=56
```


Database Buffer Pool



- Memory cache for DB
- Defers database updates
- Bufferwriter writes out dirty DB pages making them clean again
- Checkpointer occasionally records all dirty pages in the recovery log

Buffer Pool (cont.)

- Buffers (database pages) are fetched into the pool from the database automatically when locked (latched)
- Buffers remain in the pool until space is needed for new page requests and the buffer has been made clean by the buffer writer
- Pages may be pre-fetched into the buffer pool by a scan function
- Pages in the buffer pool can only live through so many checkpoint calls before they MUST be written back to the database

Log Modes and Location of Log Tail

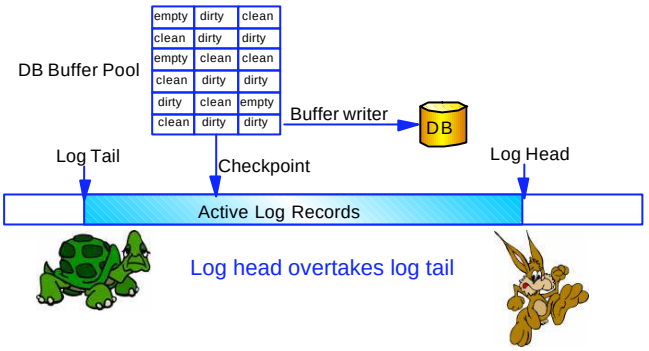
Log tail

Log head

For normal log mode, tail is oldest of these

a) Oldest active transaction
b) LSN for oldest dirty page in buffer pool
c) Last checkpoint record (there MUST be a checkpoint in the active record area)
d) Start of currently running DB backup (log tail is pinned during backup)
-or-
e) Oldest transaction since last DB backup if the log is in rollforward mode

Full Recovery Log

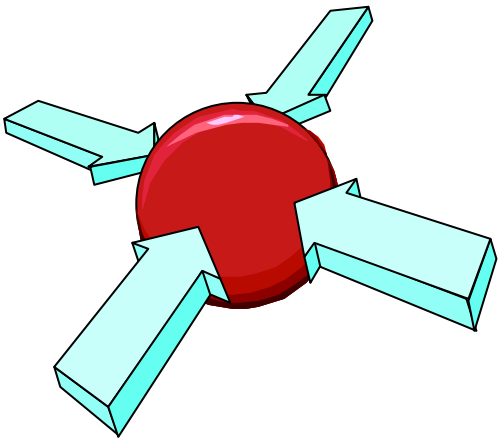


Diagnostic commands
Show Logvars
Show Lsn
Show Dbvars
Show Bufvars
Query Process
Query Status
Query Session F=D
Show Sessions
Show Txntable
Show Locks
Show Dbtxntable
Show Threads
Show Deadlocks

Steps to consider
- Increase recovery log size
- More frequent database backups (roll-forward mode)
- Cancel stalled sessions (Throughputdatathreshold / Throughputtimethreshold options)
- Avoid long-running transactions during periods of high transaction activity

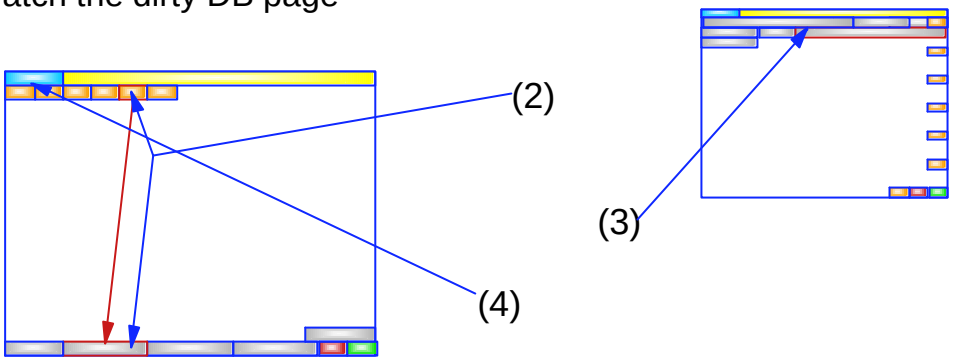
Possible product enhancements
- Improved diagnostic tools
- Reduce checkpoint frequency
- Avoid accumulation of checkpoint records
- Improve performance of buffer write

Database and Recovery Log: Referential Integrity



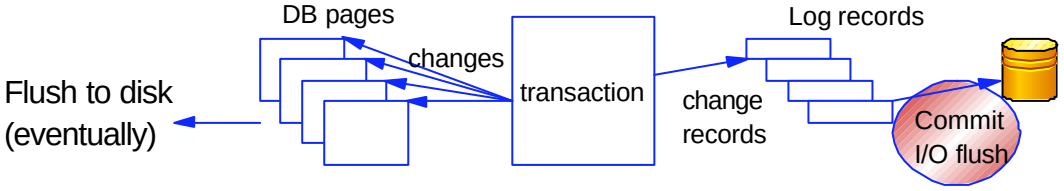
Database/Log Update Sequence

1. Latch the database page to be updated
2. Update the database page
3. Write a log record that describes the page change, returning the LSN for the log record
4. Update the DB page LSN with the LSN of the log record
5. Unlatch the dirty DB page



Transaction Commit

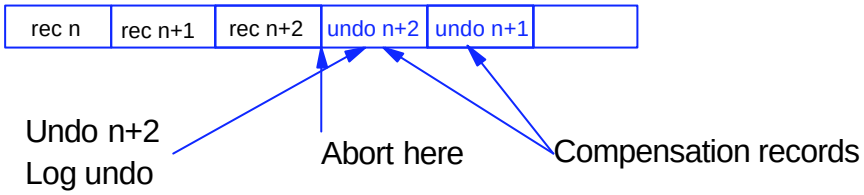
- Until a transaction commits, it is "in flight"
- Transaction commit only requires that recovery log pages describing the database changes be flushed to stable media (disk)
- During recovery/restart, the DB changes will be redone to make sure that the transaction always commits



Transaction Abort

- Also known as rollback
- The "previous-pointer" is used to link backwards in log records and undo each page change
- An UNDO action exists for every DO action
- All UNDO actions are also logged using "compensation" records

Log Records for transaction rollback:



Restart/Recovery of the TSM server

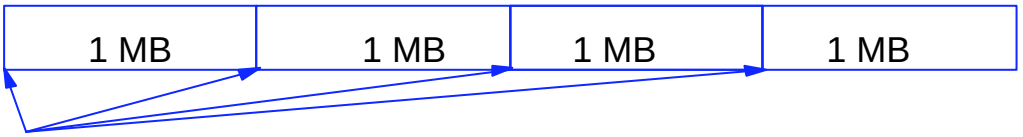
Steps:

1. Determine the end of log location (log mount)
2. Get transaction state from last checkpoint (analysis)
3. Redo all transactions that were active at the time (redo)
4. Undo all transactions that were active when server ended, but had not committed (undo)
5. Write new checkpoint record

Example Restart/Recovery

```
Version 4, Release 2, Level 1.0

Licensed Materials - Property of IBM

5698-TSM (C) Copyright IBM Corporation 1999,2001. All rights reserved.
U.S. Government Users Restricted Rights - Use, duplication or disclosure
restricted by GSA ADP Schedule Contract with IBM Corporation.

ANR0990I Server restart-recovery in progress.
ANR0200I Recovery log assigned capacity is 8 megabytes.
ANR0201I Database assigned capacity is 8 megabytes.
ANR0306I Recovery log volume mount in progress.
ANR0353I Recovery log analysis pass in progress.
ANR0354I Recovery log redo pass in progress.
ANR0355I Recovery log undo pass in progress.
ANR0352I Transaction recovery complete.
```


Determine End of Log (Mount)

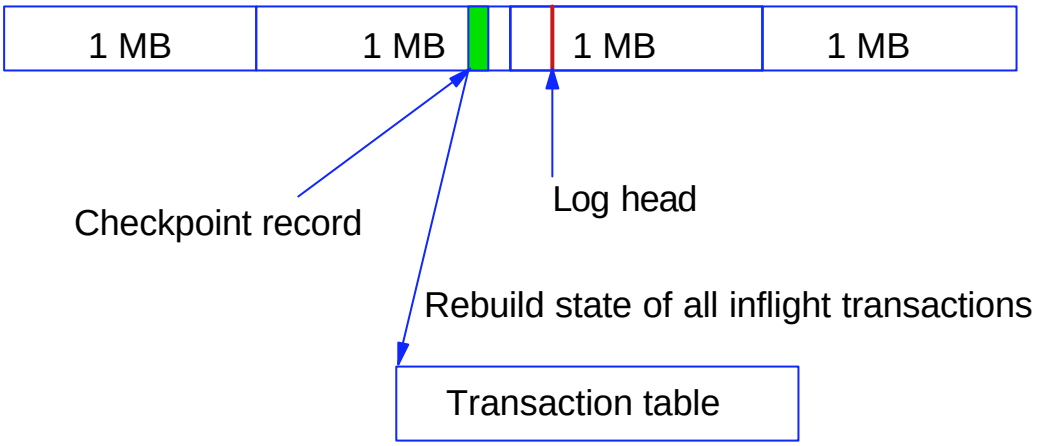


Read first page of every segment

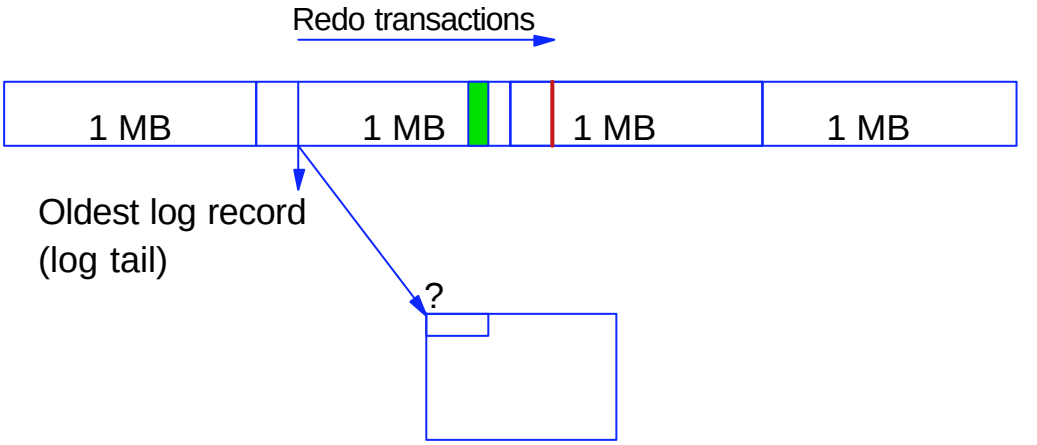
Binary search in the pages in the highest segment

Check -16 and +16 pages around the highest page found (to look for partial writes)

Rebuild Transaction Table (Analysis)

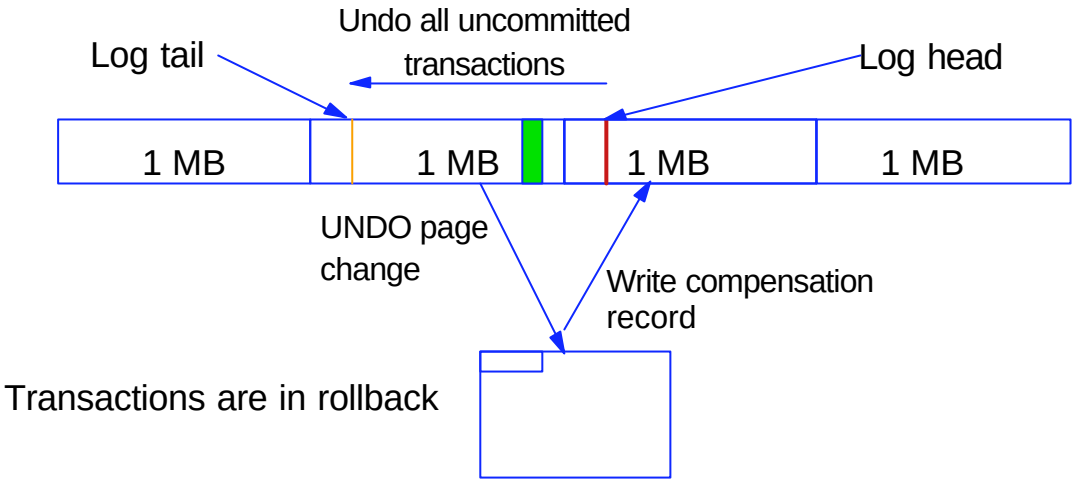


Redo Transactions



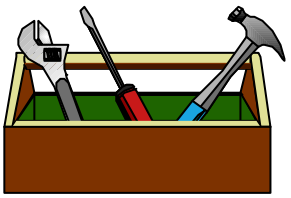
For each log record:
If DB page LSN < log LSN
redo change to the page

Undo Transactions



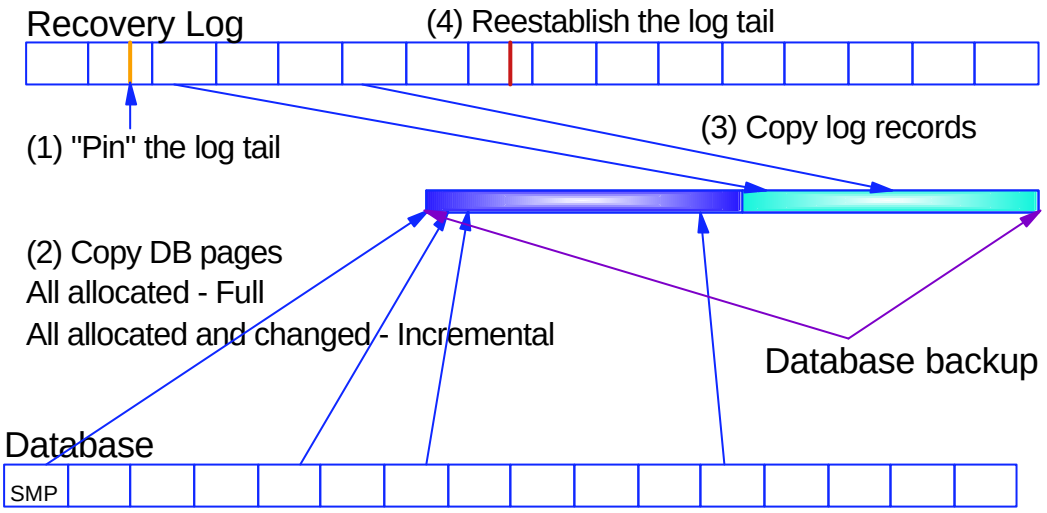
Tivoli

Database and Recovery Log : Recoverability/Reorganization



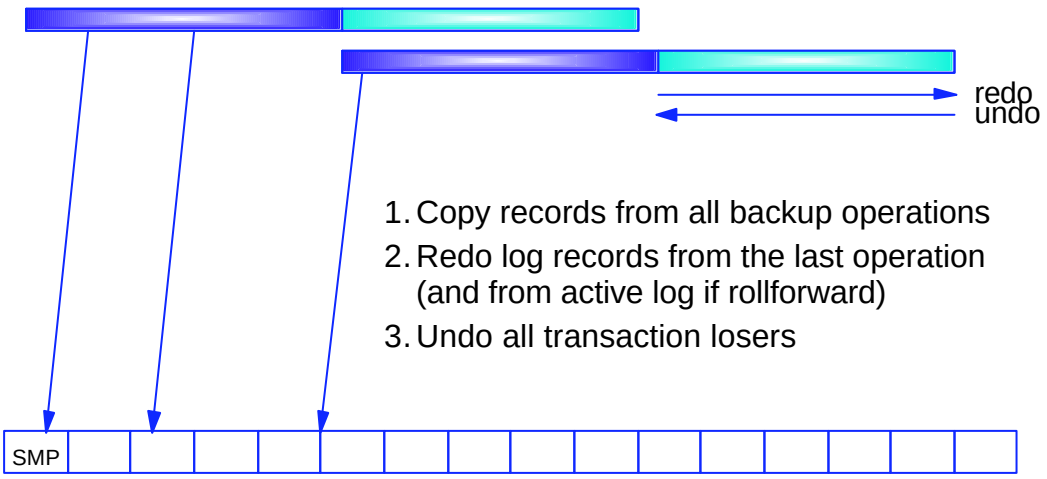
Tivoli

Database Backup



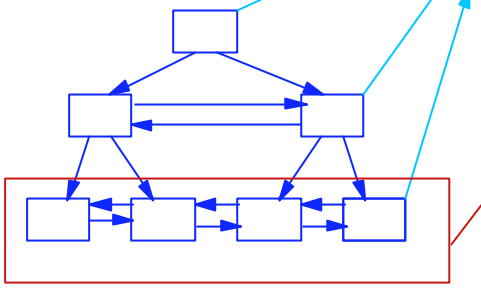
Backup uses transaction to test and reset bits in the SMP pages, which allows backup to roll back

Database Restore



1. Copy records from all backup operations
2. Redo log records from the last operation (and from active log if rollforward)
3. Undo all transaction losers

Backup vs. Dump/Unload

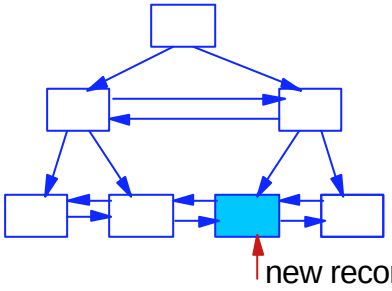


Backup
- Copies all pages (literal images)
- Does not use transaction for restore
- Guaranteed consistent

Dump/Unload
- Only copies entries from leaf pages
- Reinserts all records during load (using transaction)
- Not guaranteed consistent for off-line dump/unload operations

Tivoli

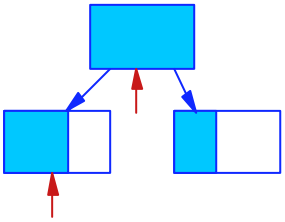
Packing Pages



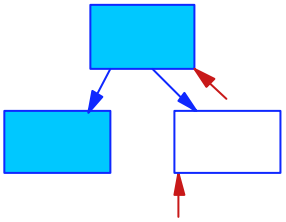
Consider processing that must take place when a new record is inserted in a page that is full

new record added to full page

Causes a "page split" to occur - the full page is split into 2 pages so new record will fit



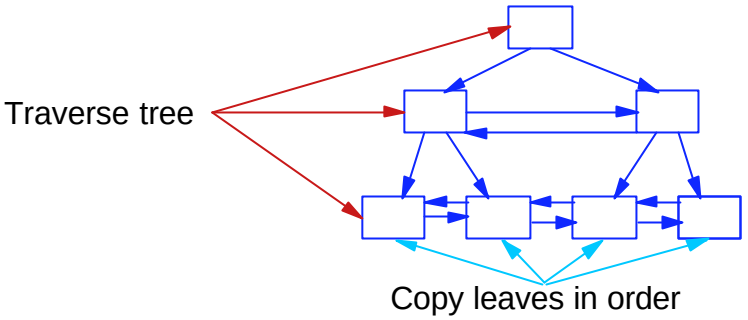
If new record fits in middle, page is split into 2 half-full pages



If new record fits at end, new page is created for new record

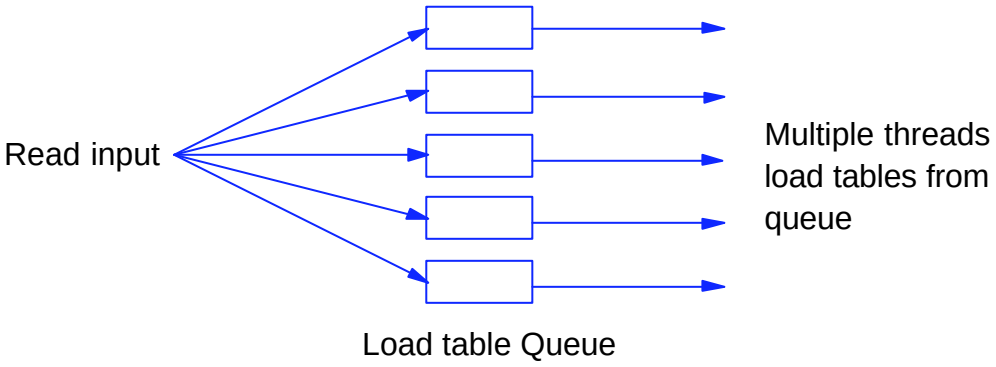
Tivoli

Unload Processing



- Unload processing copies the records for a table in ascending key order
- Requires a database whose internal B-tree tables are intact (not for salvage)
- Guarantees that database tables will be loaded in key order (packed)
- Multiple threads are used - performance tuned dynamically by throughput
- dsmserv unloaddb - stand-alone processing

Loaddb Processing



- Performed in key order if UNLOAD was used
- One thread maximum per table
- Throughput determines number of threads
- No recovery logging, buffer pool must be flushed at end

Summary

- The Tivoli Storage Manager server is complex and cannot be completely described in the allotted time
- This presentation has covered selected features of key components
- Topics have included
  - Architecture
  - Session management
  - Database applications
  - Transaction management
  - Inventory management
  - Bitfiles
  - Storage service
  - Data transfer
  - Database and recovery log

Tivoli