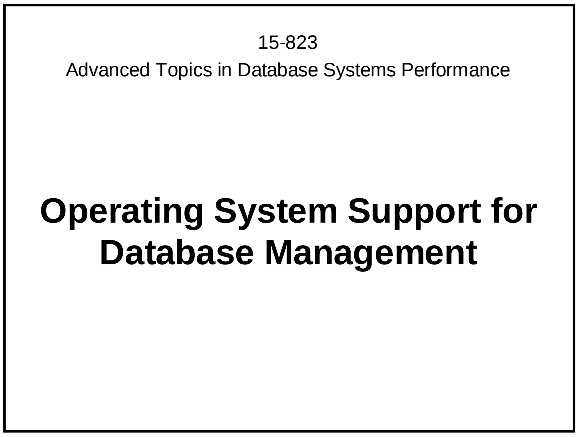
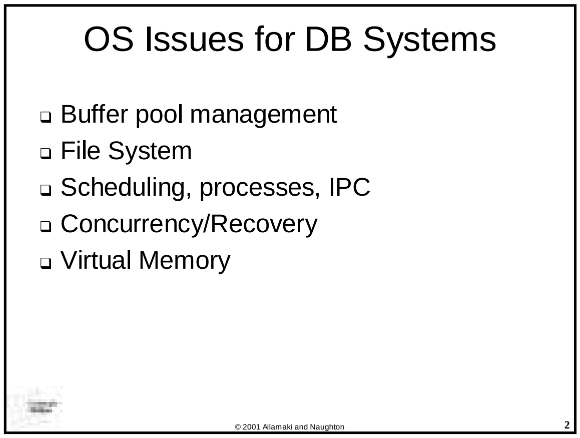


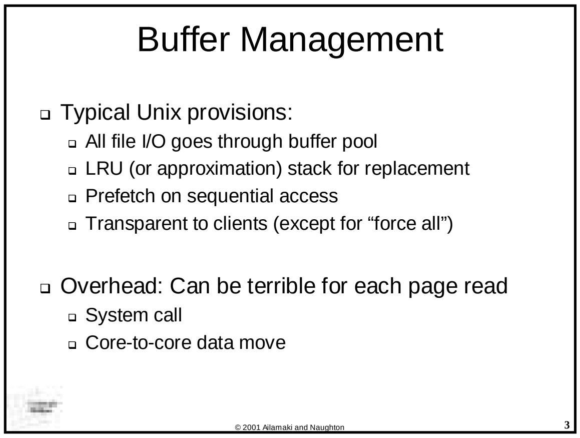
15-823
Advanced Topics in Database Systems Performance

Operating System Support for Database Management



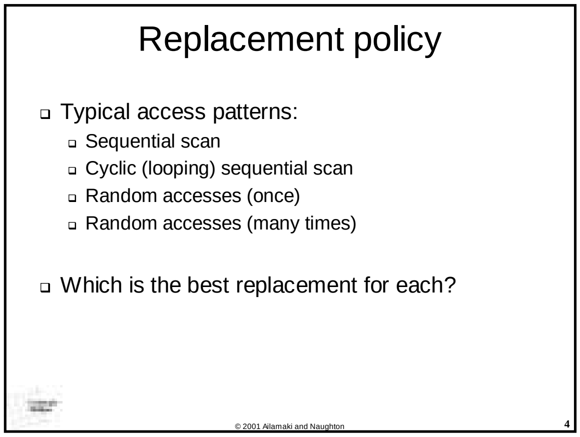
OS Issues for DB Systems

- Buffer pool management
- File System
- Scheduling, processes, IPC
- Concurrency/Recovery
- Virtual Memory



Buffer Management

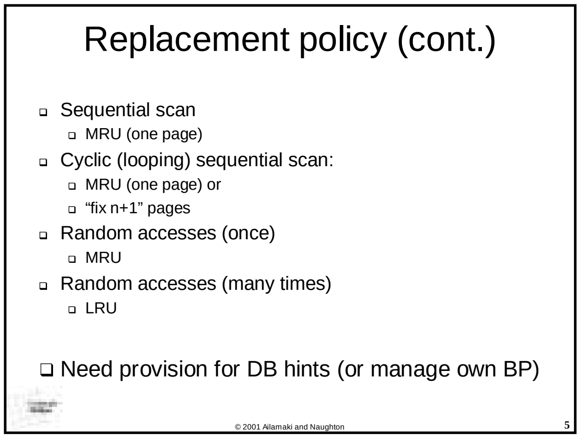
- Typical Unix provisions:
  - All file I/O goes through buffer pool
  - LRU (or approximation) stack for replacement
  - Prefetch on sequential access
  - Transparent to clients (except for “force all”)
- Overhead: Can be terrible for each page read
  - System call
  - Core-to-core data move



Replacement policy

- Typical access patterns:
  - Sequential scan
  - Cyclic (looping) sequential scan
  - Random accesses (once)
  - Random accesses (many times)

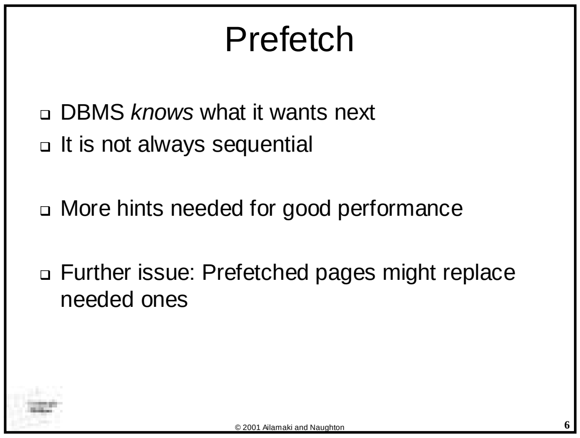
- Which is the best replacement for each?



Replacement policy (cont.)

- Sequential scan
  - MRU (one page)
- Cyclic (looping) sequential scan:
  - MRU (one page) or
  - “fix n+1” pages
- Random accesses (once)
  - MRU
- Random accesses (many times)
  - LRU

- Need provision for DB hints (or manage own BP)

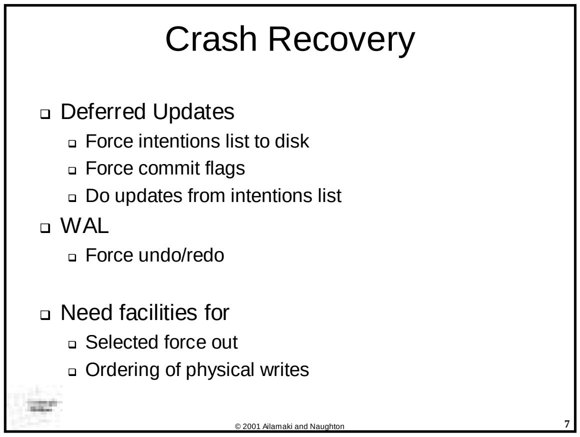


Prefetch

- DBMS *knows* what it wants next
- It is not always sequential

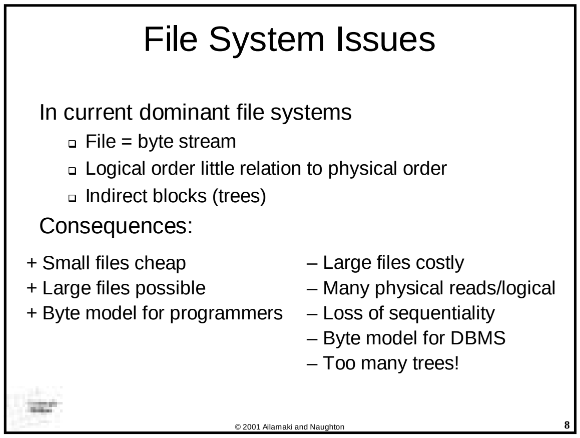
- More hints needed for good performance

- Further issue: Prefetched pages might replace needed ones



Crash Recovery

- Deferred Updates
  - Force intentions list to disk
  - Force commit flags
  - Do updates from intentions list
- WAL
  - Force undo/redo
- Need facilities for
  - Selected force out
  - Ordering of physical writes



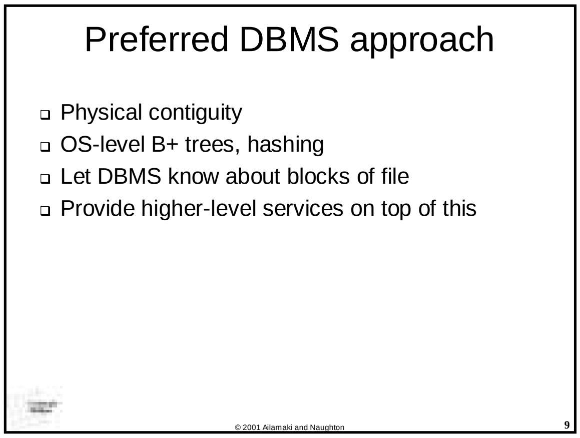
File System Issues

In current dominant file systems

- File = byte stream
- Logical order little relation to physical order
- Indirect blocks (trees)

Consequences:

+ Small files cheap	– Large files costly
+ Large files possible	– Many physical reads/logical
+ Byte model for programmers	– Loss of sequentiality
	– Byte model for DBMS
	– Too many trees!



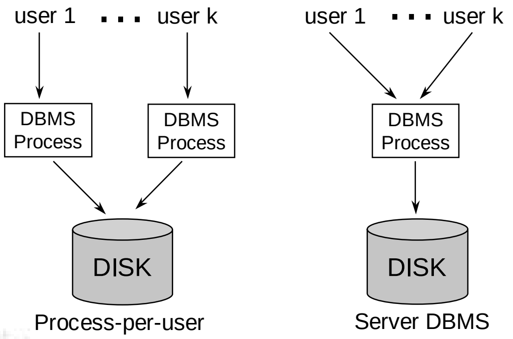
Preferred DBMS approach

- Physical contiguity
- OS-level B+ trees, hashing
- Let DBMS know about blocks of file
- Provide higher-level services on top of this

Scheduling, Processing, IPC

- DBMS needs
  - Shared buffer pool
  - Shared lock table
  - Critical sections

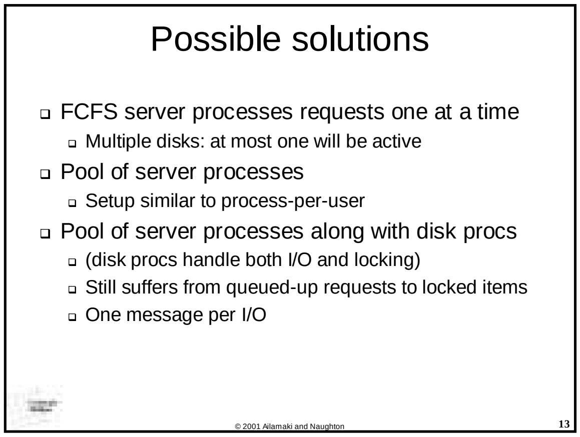
Structure Alternatives



Evaluation

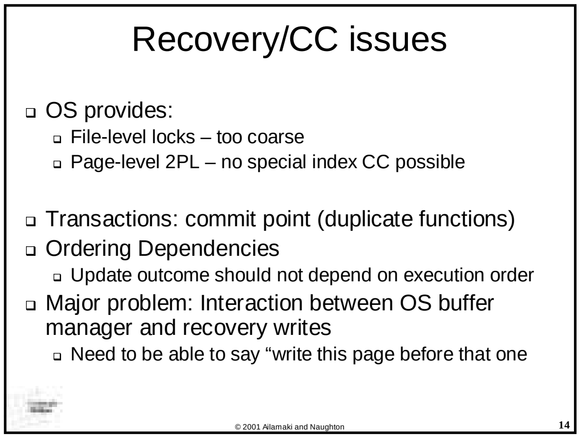
- Process-per-user structure
  - Expensive context-switching
  - Preemption at “bad places” (DBMS’s critical sections)
- DBMS server
  - Duplication of OS services (must do own multi-tasking)
  - Cost of messages is several thousand instructions

- DBMS would like
  - Reduced message/task overheads
  - No-preemption scheduling
  - “fast-path” for context-switching among DBMS procs



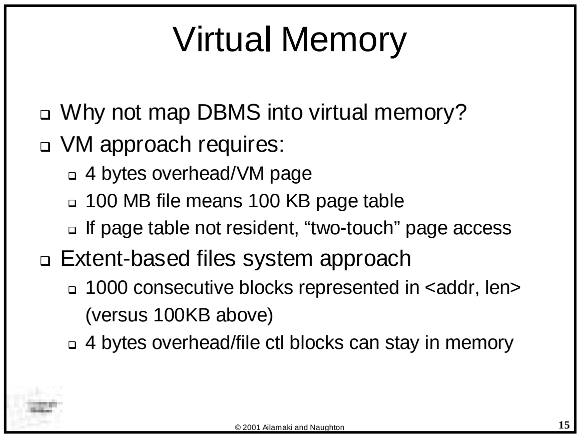
Possible solutions

- FCFS server processes requests one at a time
  - Multiple disks: at most one will be active
- Pool of server processes
  - Setup similar to process-per-user
- Pool of server processes along with disk procs
  - (disk procs handle both I/O and locking)
  - Still suffers from queued-up requests to locked items
  - One message per I/O



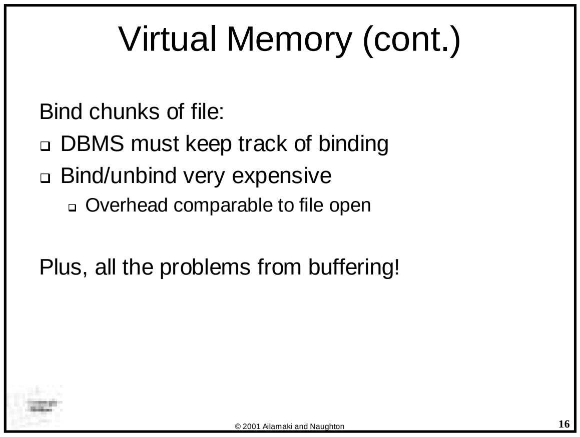
Recovery/CC issues

- OS provides:
  - File-level locks – too coarse
  - Page-level 2PL – no special index CC possible
- Transactions: commit point (duplicate functions)
- Ordering Dependencies
  - Update outcome should not depend on execution order
- Major problem: Interaction between OS buffer manager and recovery writes
  - Need to be able to say “write this page before that one



Virtual Memory

- Why not map DBMS into virtual memory?
- VM approach requires:
  - 4 bytes overhead/VM page
  - 100 MB file means 100 KB page table
  - If page table not resident, “two-touch” page access
- Extent-based files system approach
  - 1000 consecutive blocks represented in <addr, len> (versus 100KB above)
  - 4 bytes overhead/file ctl blocks can stay in memory

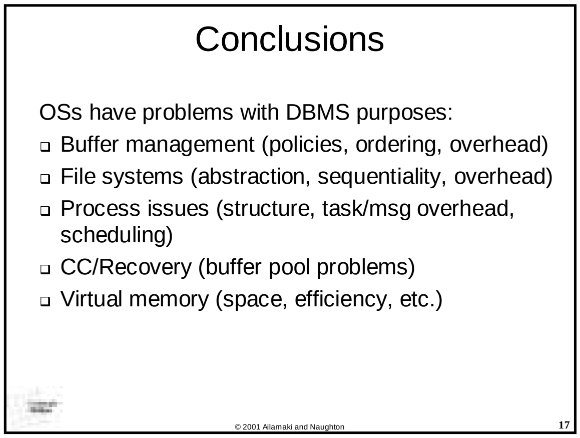


Virtual Memory (cont.)

Bind chunks of file:

- ❑ DBMS must keep track of binding
- ❑ Bind/unbind very expensive
  - ❑ Overhead comparable to file open

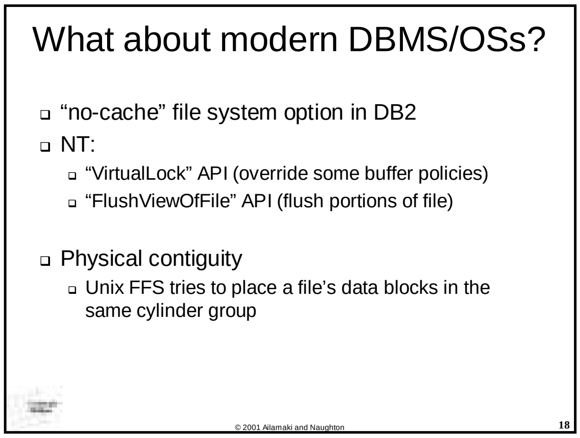
Plus, all the problems from buffering!



Conclusions

OSs have problems with DBMS purposes:

- ❑ Buffer management (policies, ordering, overhead)
- ❑ File systems (abstraction, sequentiality, overhead)
- ❑ Process issues (structure, task/msg overhead, scheduling)
- ❑ CC/Recovery (buffer pool problems)
- ❑ Virtual memory (space, efficiency, etc.)



What about modern DBMS/OSs?

- ❑ “no-cache” file system option in DB2
- ❑ NT:
  - ❑ “VirtualLock” API (override some buffer policies)
  - ❑ “FlushViewOfFile” API (flush portions of file)
- ❑ Physical contiguity
  - ❑ Unix FFS tries to place a file’s data blocks in the same cylinder group