Database Systems
WS 07/08

Prof. Dr. Jens Dittrich

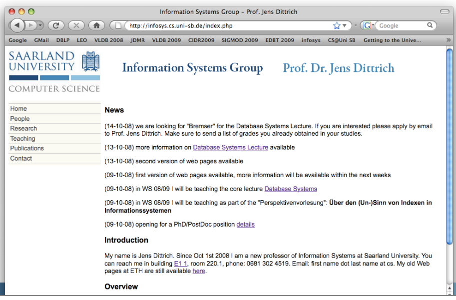
Chair of Information Systems Group
http://infosys.cs.uni-saarland.de

About me.

- born 1972
- 1993 - 99: Marburg University, Prof. Seeger: Geographic Information Systems (GIS)
- 1999 - 2002: PhD on efficient database algorithms, join processing, XXL Software-library
- 2003 - 2004: SAP AG, data warehousing and OLAP databases, distributed main-memory column-store
- 2004 - 09/2008: ETH Zurich, senior researcher in information system, Systems Group, Prof. Kossmann
- Since Oct 1st 2008: new professor at the CS department

Information Systems Group

- where: E 1 1, 2nd floor
- my office: 220.1
- Web:



What we do.

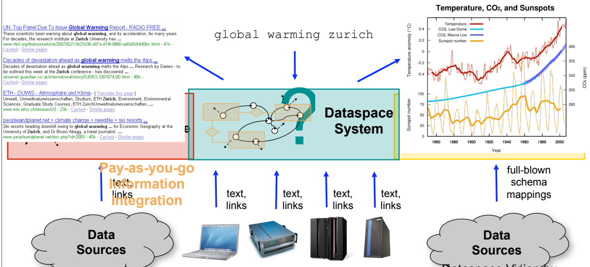
New system architectures for data management

- idea of a Database Management Systems (DBMS) was developed more than 30 years ago
- Meanwhile many new data managing problems have been identified.
- Several new data models were invented.
- Hardware has changed dramatically.
- We doubt that the abstraction of a “Database Management System“ is always the best abstraction.
- Therefore we are interested in coming up with better system abstraction that cover a **wider** range of information management problems.

Dataspaces

- dataspaces are a new abstraction for information management
- A Dataspace system is a new kind of information integration system.
- It incorporates features of search engines (Google et.al.) as well as information integration systems (EII/OLAP/DWH).
- Think of it as a **search&store++**
- We have built one of the first dataspace management systems: iMeMex.
- Extremely hot topic in research.
- We will go into detail later in this lecture.

Research Challenge: Is There an Integration Solution in-between These Two Extremes?



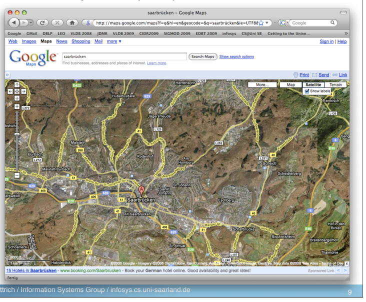
Dataspace Vision by Franklin, Halevy, and Maier [SIGMOD Record 05]

Indexing

- indexing is key to providing efficient query processing in almost any information system.
- We are interested in problems that may not be solved with existing techniques...
- ...or require a fresh look at available techniques.
- Examples
  - indexing of semi-structured data (including graphs)
  - spatial indexing
  - moving object indexing
  - update-efficient indexing
  - adaptive indexing

Spatial/geographical data

- Spatial data is used in systems like Google Maps or any other Geographic Information System (GIS).
- In order to organize large collections of spatial data, efficient indexing and query processing techniques are required.



Data warehousing and OLAP

- Online Analytical Processing (OLAP) is at the heart of any medium to large business.
- A data warehouse can be considered a “database of databases“.
- huge data volumes involved
- very interesting algorithmic and design challenges
- Building these systems is far more challenging compared to 'standard' OLAP systems.
- OLAP systems (should) provide instantaneous answers to complex queries, but how?

Indexing support under high update rates

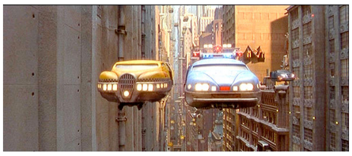
- Most indexing techniques do not work well if the data organized by the index is frequently updated (millions of updates per second).
- Oxymoron: an index is either query **or** update-efficient, but not both
- We develop techniques to overcome this barrier.

Main memory databases

- Technology of commercial DBMS was developed having disk-based architectures in mind
- However
  - main memory is getting extremely cheap
  - flash is getting cheaper
- => Most databases today already fit into main memory or some combination with flash memory
- In addition, new kind of storage media is coming up, e.g., persistable RAM. This new hardware will affect how information systems are built.
- multi-core architectures
- Challenge: what are the right techniques?

Moving objects and vehicle tracking including cars and aircraft

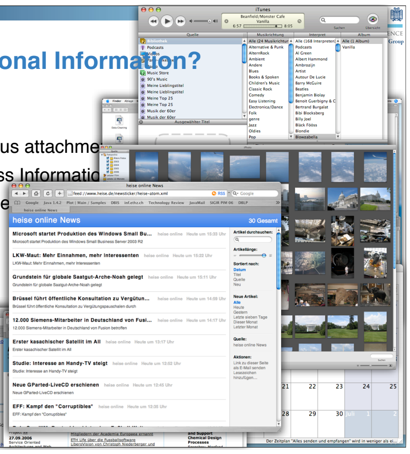
- Remember the movie “The Fifth Element“?



- security surveillance (distance to other cars)
- tolling
- automatic speeding control (yes, politically not realistic... ;-))
- we develop new scalable techniques to solve this

What is Personal Information?

- All your files
- All you emails plus attachme
- Calender/Address Informatio
- Visited Web page
- Music
- Pictures
- Video



Personal Information: Where is it?

- On devices I own	**Example: myself**
- laptop	(an old Mac and a new Mac)
- desktop	(no desktop)
- cellular	(got one)
- memory stick	(half a dozen)
- private desktop	(yes, yet another Mac)
- digital cam	(got a very nice one)
- iPod	(don't have)
- On devices I do not own	
- network share	(group's SMB share)
- On remote services	
- email server	(SB IMAP plus private account)
- subversion service	(everything project-related goes here)
- web	(web pages, news services, RSS, CMS, etc.)
- backup services	(I do not use this, only backup of network shares)
- On my cupboard	
- archived information on DVDs	(regularly)

Personal Information Management

- Personal information is everywhere
- Currently people use a zoo of techniques to manage and query this information.
- We look at ways of coming up with a unified system for personal information management allowing you to handle your data with a single system.
- see http://iMeMex.org

Indexing Social Networks

- facebook, linkedin, orkut, et.al.
- We are currently investigating indexing challenges in the area of social networks.
- sorry: stealth mode

About this Lecture.

"Database Systems"

- the term "database systems" is somewhat narrow
- "database systems" are assumed to
  - **store** the data, take **full control** on the data
  - support some sort of **declarative language** like SQL
  - have a **pull-based** query-model
  - support **transactions**: **ACID**
  - be **generic** to work well for all sorts of scenarios
- however, there are many more data managing scenarios out there that do not require these assumptions
- in addition, some scenarios require different assumptions
- DBMSs are **ill-equipped** to handle these scenarios
- therefore I prefer the more general term "Information System"

Examples of Information Systems

- database (management) systems: relational, OO, XML, ...
- data warehouse systems, OLAP
- search engines
- text mining engines
- mediators, information integrators
- file systems
- publish-subscribe systems
- streaming engines
- graph databases
- geographic information systems
- <you name it>

Idea of this Lecture

- teach fundamentals of data/information managing technology
- teach fundamental principles used to build **any** information systems (not only database systems)
- make you understand
  - algorithms
  - data managing patterns
  - architectures
  - principles
  - best practices

Why would that be useful?

- helps you to find/define the right type of information system for a given scenario (choose the right weapon)
- helps you to make good choices when planning and designing an information management scenario
- helps you to understand and fix performance issues in information management scenarios

Topics (1/6)

- storage media
  - disk
  - flash
  - main memory
- storage management
  - principles
  - page/block mapping and replacement
- data layout
  - vertical
  - horizontal
  - PAX
  - fractured mirrors
  - data model specific

Topics (2/6)

- indexing
  - one- and multidimensional
  - tree-structured
  - hash-indexes
  - partition-based indexing
  - bulk-loading
  - differential indexing
  - read-optimized indexing
  - write-optimized indexing
  - data warehouse indexing
  - text indexing: inverted files
  - main-memory indexing
  - (flash-indexing)

Topics (3/6)

- operator models
  - push-model
  - pull-model
- operator implementations
  - general idea
  - join algorithms for relational and multidimensional data
  - other operators
- query processing
  - scanning & “naive plans“
  - canonical plan computation

Topics (4/6)

- query optimization
  - query rewrite
  - cost-based
- data recovery
  - single versus multiple instance
  - ARIES
- parallelization of data and queries
  - horizontal partitioning
  - vertical partitioning
  - replication
  - map-reduce
  - multi-cores

Topics (5/6)

- read-optimized system concepts
  - search engines
  - data warehouses and OLAP
- write-optimized system concepts
  - OLTP
  - publish/subscribe
  - streaming
  - moving objects
- management of geographical data
  - basic concepts
  - GIS, google maps

Topics (6/6)

- dataspace systems
  - big vision
  - pay-as-you-go information integration
  - system examples (iMeMex)
- ...

Exercises.

Exercises: “Bremser“ wanted

- “Bremser“ (teaching assistants) wanted
  - will teach exercise groups
  - will supervise practical projects
  - will be paid
- as this lecture is new you may, exceptionally for this year, at the same time
  - attend the lecture, i.e., participate in the exams
  - and be a teaching assistant
- interested?
  - talk to me after the lecture
  - or apply by email: jens dot dittrich @ cs
  - please send grades of lectures you attended

Exercises: What?

- 40% paper work
  - small paper exercises, Q&A, etc.
  - no hand-in of exercises
  - you should present at least one solution per year in the exercise groups
  - all voluntary
  - however, in order to pass the exams
    - you will need to attend exercises
    - will have to be able to solve paper exercises by yourself
- 60% practical project
  - solve a small data managing task
  - based on small teams

Project description

- link

Exercise groups

- First group in week 3.11-7.11.
- please enter your availabilities in the Doodle
- http://www.doodle.com/iy3bhafw2pcmtqmv
- **deadline for entering availabilities: Oct 27**

Exams.

2+1 Exams

- one mid-term in December 2008
- one end-term: Feb 12, 2009
- one repetition exam: April 16, 2009
- you need to pass two exams
- grade will be computed on your two best exam grades
- exam grade determines 70% of your overall grade for this course
- again: make sure to attend the exercise groups, solve paper exercises and actively participate in the project...

Office hours Prof. Jens Dittrich

- Wednesdays 2 to 3pm
- or ask for an appointment by email:

  Office: E1 1, room 220.1
  EMail: jens.dittrich @ cs
  Web: infosys.cs.uni-saarland.de

- please do not come at other times without appointment

Literature

- Books
  - Raghu Ramakrishnan:
    Database Management Systems
    Mc Graw-Hill
  - Alfons Kemper, André Eickler:
    Datenbanksysteme: Eine Einführung.
    Oldenbourg Verlag
  - Dennis Shasha, Philippe Bonet:
    Database Tuning.
    Morgan Kaufmann.
  - Theo Härder, Erhard Rahm:
    Konzepte und Techniken der Implementierung von Datenbanksystemen.
    Springer Verlag.

Literature: Conferences

- Papers in conference proceedings (annual)
  in the area of DBMS far more important than journals!
  - **ACM SIGMOD/PODS:**
    ACM Special Interest Group on Management of Data/Principles of Database Systems
    2008: http://www.sigmod08.org
  - **VLDB**: International Conference on Very Large Data Bases
    2008: http://www.vldb2008.auckland.ac.nz
  - **PVLDB**: Proceedings of VLDB
    http://www.jdmr.org/
  - **IEEE ICDE**: International Conference on Data Engineering
    2009: http://www.icde2008.org

Literature: Journals (quarterly)

- ACM TODS: Transactions on Database Systems
  http://www.acm.org/tods/
- VLDB Journal
  http://link.springer.de/link/service/journals/00778/index.htm
- IEEE TKDE: Transactions on Knowledge and Data Engineering
  http://www.computer.org/tkde/
- ACM Computing Surveys
  http://www.acm.org/pubs/surveys/
- ACM Sigmod Record
  http://www.sigmod.org/record/
- IEEE TKDE: Transactions on Knowledge and Data Engineering
  http://www.computer.org/tkde/

How to Obtain a Particular Paper?

- Web-search for papers
  - **DBLP**: **D**ata **B**ases and **L**ogic **P**rogramming
    http://www.informatik.uni-trier.de/~ley/db/
  - Google
    use quotes: "<paper title>" (will find almost everything)
  - ACM Portal
    http://portal.acm.org
  - Citeseer
    http://citeseer.csail.mit.edu/
- Did not find a paper using these methods?: ask us.

Introduction.

Requirements of an Information System.

Classic Requirements for a DBMS

- store the data
  - take full control on the data
  - systems **owns** data
  - all data in one system
- support some sort of declarative language like SQL
- pull-based query-model
- support transactions: ACID
- concurrency
- crash recovery
  - at different levels (one system, multiple systems data centers)
- be generic to work well for all sorts of scenarios
- good query/update/insert performance

Additional Requirements (1/4)

- extensibility
- should be parallelizable
- support for other data models: Geo, OO, RDF, XML, graphs
- support for other query languages (e.g., XQuery)
- subscriptions and triggers
- time travel and versioning
  - how did the data look like two days ago?
- support for special hardware
  - embedded
  - mobile
  - flash
- data accessible from everywhere

Additional Requirements (2/4)

- self-tuning
  - decides on best plans
  - learns from past queries
  - decides on "right" indexes
- can be scaled dynamically at peak times
  - redistribution to other machines completely hidden
  - scales across thousands of computers and not just a few dozen
  - optimal: scales with a single code base from mobile phone to thousands of server machines
- zero-admin
  - (almost) no DBA interaction required

Additional Requirements (3/4)

- Interactive interfaces
  - User may interact with the computation of a query result
    e.g., online aggregation
    e.g., progressive algorithms
  - Incremental refinement of queries
    e.g., progressive algorithms with error guarantees
  - data navigation/browsing
- probabilistic
  - uncertainty on data and queries
    e.g. in IR, a query is never precise but just a "hint"
  - best-effort answers

Even more Requirements.... (4/4)

- read-optimized (e.g. search engines)
- write-optimized (e.g. vehicle monitoring)
- push-based processing (streams, pub/sub)
- support for textual data (Google et.al.)
- support for BLOBS (as in file systems)
- ...

Wait!
Can we support all of this within a **single** system?

Fundamental System Concepts.

Information System Architectures

- DBMSs started with the most basic assumptions
- DBMSs were then extended over time to support other features
- however, today several different information system architectures exist
  - **OLTP, classic "Database Systems"** (see summer term lecture): read/write-optimized, row-store, transactions, recovery, concurrency, up-to-date results
  - **OLAP, data warehouse system**: read-optimized, column-store, no transactions, stale results
  - **search engines**: Google et.al. read-optimized, text and attributes, no transactions, (stale results)
  - **tracking engines**: publish/subscribe, streams, mobile write-optimized, (stale results), (transactions)

Classical Store/Retrieve Architecture

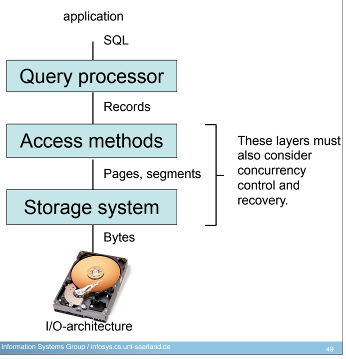
Tasks

Creation and optimization of query plans

Management of records and access paths

DB-buffer and management of external memory

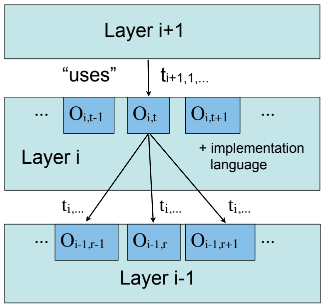
Storage Media



Virtual Machine Concept

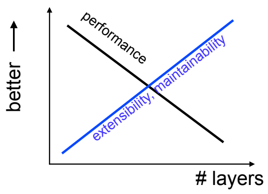
Operators O

Data objects t



Virtual Machine Concept

- Each layer may be considered a virtual machine.
- Programs running on layer i realize implementations of operators $O_{i,...}$ (using functionality of layer i-1)
- trade-off:



Advantages

- Higher layers are easier to realize as they may make use of lower layers.
- Changes on higher levels do not have an impact on lower layers.
- Higher layers may be removed; lower layers still remain functional.
- Lower layers may be tested independent from the fact whether higher layers are working.

- OK this is textbook stuff.
- but then there is reality...

Real Systems

- it is very difficult to consider all aspects of a system design in the first place
- typically systems start with an initial design
- then they evolve over time (think in years and decades)
- over time new requirements come in
  - change of interfaces may be needed
  - may require new partitioning of components, modules, block
  - smaller changes may be done without changing the architecture
  - eventually, deep architectural changes required in order to keep a clean system design
  - in practice this is often "avoided" by introducing **hacks** (aka patches, workarounds, quick-fixes, hot-patches, etc.)
  - these hacks may violate the layered system architecture

Impact of a Hack (1/2)

- if a hack violates the layered architecture, the system becomes a little bit more monolithic
- over time system converges against a monolithic system
- drawbacks of a hack
  - system complexity rises
  - system much harder to understand
  - system harder to test (single component tests?)
  - component dependencies may be unclear
  - extensions become more and more difficult over time due to global effects
  - maintainability more difficult ("What the heck is this flag for?")
  - less people will really understand the system

Impact of a Hack (2/2)

- there are also advantages of a hack
  - problem quickly solved
  - no or few interactions with other developers
  - “it works, so what?“
  - customer and boss happy

Impact of a System Re-Design

- drawbacks
  - much more work
  - need to come up with right interface for the extension
  - your change may trigger other changes generating even more work...
  - difficult if many people involved
- advantages
  - clean design
  - system easier to understand
  - system easier to test
  - system easier to maintain
  - and I claim: less error-prone
  - saves future work

Hacks versus Re-Designs

- or: monolithic system aspects versus a clean layered design
- Hacks:
  - quick solution
  - save time now, postpone work to the future
  - easy
- ReDesigns:
  - solution takes time
  - do work now, save time in the future
  - may be painful

What is the “right“ Solution?

- My personal recipe:

1. always start with a throw-away prototype
2. take everything you learned and try to come up with a good initial system design
3. stick to that design as much as you can
4. if not possible: use hacks and document them well
5. if amount of hacks becomes too big, redesign, go to Step 3

- Note: you will learn a lot from re-designs
  (not only about design but also algorithms)

Storage Media.

Storage Media.

Introduction.

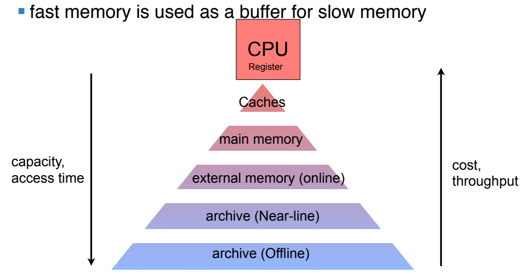
Properties of Ideal Memory

- unlimited capacity
- fast access for random access
- high bandwidth for sequential access
- cheap
- persistent

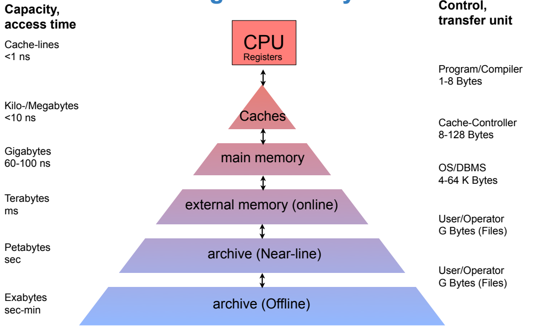
Why not store all data inside the CPU cache?

Storage Hierarchy

- small, expensive but fast memory close to the CPU
- big, cheap but slow memory a the periphery
- fast memory is used as a buffer for slow memory



Storage Hierarchy



Tasks of Storage Hierarchy Layers

- localization of data objects
- allocation of free space
- caching of date from lower-level layer
- replacement strategies
- write strategies for modified data
  - Write-back (write if data gets evicted)
  - Write-through (immediate write through to the underlying layer)
- if needed: transformation to the right transfer size

Note: These tasks are similar on all layers!

Tape.

Tape

- has been used for 50 years
- now based on cartridges
- 2008: up to 1 TB per cartridge
- slow access time due to winding: approx 100 sec
- high bandwidth: up to 100 MB/sec
- good for archival/backup storage

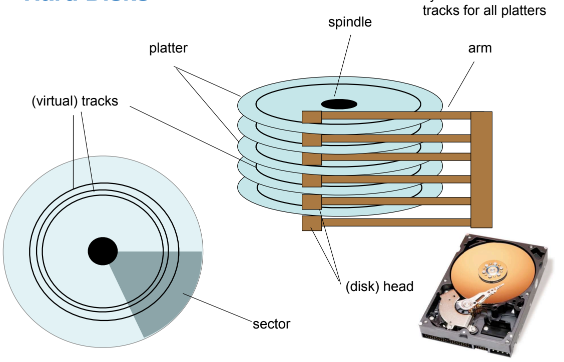
Tape Libraries/Tape Jukebox

- cartridge library
- plus loading robot
- may hold several 100,000 cartridges
- up to 100 Petabyte storage (uncompressed)
- slow access
  - fetch cartridge (several seconds)
  - winding
- good for infrequent access to archive

Hard Disks.

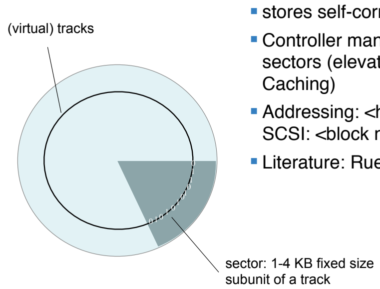
Hard Disks

cylinder = set of similar tracks for all platters



Sector

- Unit for read/write-access
- stores self-correcting error codes
- Controller manages access to sectors (elevator strategy and Caching)
- Addressing: <head,track,sector>, for SCSI: <block number>
- Literature: Ruemmler&Wilkes 1994



Hard Disk Details

- **Zoning**
  - Tracks at the periphery are longer and therefore contain more sectors.
  - Adjacent cylinders having the same number of sectors are grouped into zones.
  - Zones near the outer edge have more sectors per track than zones on the inside
- **Track Skewing**
  - move sector 0 of each track such that sequential scans are supported (jump from one track to an adjacent track without having to pay for rotational delay)
- **Sparing**
  - erroneous sectors are detected by the controller and mapped to other places
  - sectors may be detected during the production of the hard disk, or when the disk is in use

Hard Disk Controller

- **Read caching**
  - Request is served by the hard disk cache
  - Tries to read the entire track into the cache (read ahead)
- **Write caching**
  - good for asynchronous write
  - problem: volatile vs. non-volatile disk cache
  - special parameter allows applications to force data to disk
  - important for logging (we will come back to this)
- **Elevator strategy**
  - If multiple requests have to be served, take the one that needs the smalls arm movement.
  - Fairness?

Cost of a Random Access

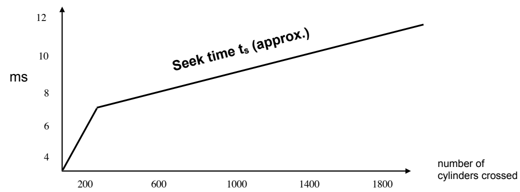
1. send request to operating system
2. send call to hard disk controller
3. controller decodes request
4. seek-time $\mathbf{t_s}$ (arm movement)
5. activate disk head
6. rotational delay $\mathbf{t_r}$ (wait for the right sector)
7. transfer time $\mathbf{t_{tr}}$ (transfer the actual data)
8. check-time
9. transfer to operating system

Simplification: $t = t_s + t_r + t_{tr}$

= "seek + rotation + transfer"

Details of a Seek

- speed-up: arm acceleration
- coast: arm movement with maximal velocity
- slowdown: slow-down arm
- settle: fine-tune arm position to the right position



Sequential vs. Random Access

- **Experiment**
  Read 1000 blocks of size 8 KB

  (u: transfer rate in MB, k: number of cylinders crossed)

  - sequential read:
    $t_{seq} = avg(t_s) + t_r/2 + k*min(t_s) + 1000 * 8$ KB/u
  - each block randomly read:
    $t_{random} = 1000 * (avg(t_s) + t_r/2 + t_{tr})$

Sequential vs. Random Access

- **Experiment**
  Read 1000 blocks of size 8 KB

	1970	2007	improvement
random	48 275 ms	6 000 ms	8,0
sequential	10 315 ms	70 ms	147,4
ratio	4,7	85,7	

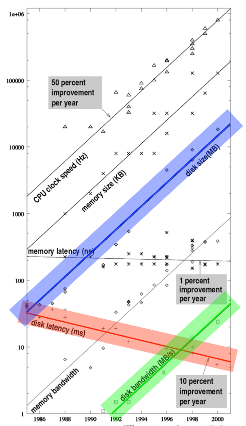
Consequences:

- If more than 1/85,7= **1,1%** of the blocks have to be accessed, it pays off to read the **entire file!!!**
- In 1970 this value was much higher: **21.3%**.

⇨ Important design criteria for index structures!

Evolution of Hard Disks

- random access time (disk latency) is improved by only approx. 10% every year.
- But:
  throughput (disk bandwidth) for sequential access is improved by 50% every year!
- In addition:
  Hard disk capacity (disk size) grows by 50% every year.



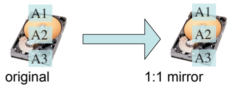
Source: P. Boncz

Disk Arrays

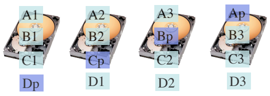
- **Motivation**
  - single hard disk has a high throughput but is slow on random access
- **Idea**
  - many small hard disks are grouped to form a virtual hard disk
  - I/O-parallelism can be exploited
  - Parallel accesses (**intra**-query parallelism)
    - Goal: speed-up of single tasks
    - Method: horizontal partitioning
    - partition data such that a single query can be split into subqueries
  - Parallel tasks (**inter**-query parallelism)
    - Goal: scale to handle multiple tasks in parallel
    - Method: de-clustering
    - partition data such that different queries do not interfere with each other

Disk Array Details

- more hard disks => probability of an error increases
- RAID (Redundant Array of Inexpensive Disks)
- RAID 1:
  - mirroring
  - halve the capacity
  - no I/O-parallelism

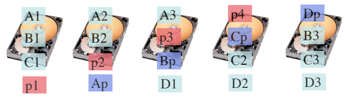


- RAID 5:
  - parity codes distributed to several disks
  - at least 3 hard disks needed
  - survives failure of one hard disk



Disk Array Details

- RAID 6:
  - several independent parity codes
  - survives failure of multiple hard disk
  - large number of disks required

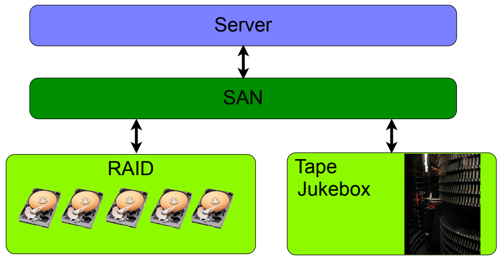


Network-Based Storage

- network not a bottleneck anymore
  - 10 Gb Ethernet has 1.25 GB/sec bandwidth
  - Infiniband QDR (Quad Data Rate) 12X has 12 GB/sec bandwidth
- may be used to **hide** storage devices including
  - disk raids
  - tape
  - tape libraries
- abstract from underlying storage devices

Storage-Area Networks (SAN)

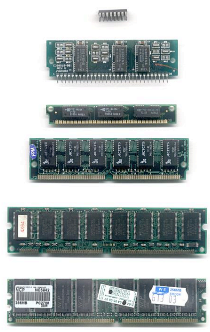
- provides block access to logical disks
- in contrast to NAS no file system but block-level access
- “give me block 42 of disk 11“



Main Memory.

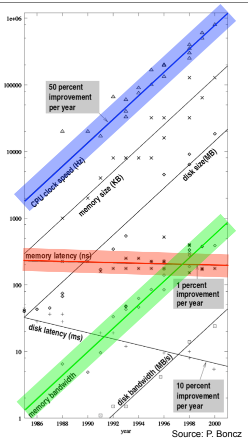
Main Memory

- fast access: 60-100ns
- volatile
- getting cheaper and cheaper
- depending on brand/features prices approx 30€/GB
- dozens of Gigabytes even on a small server
- information/database systems used to be disk-oriented
- these times are over...



Evolution of CPU and RAM

- access time (memory latency) is improved by approx. 1% every year.
- But: throughput (memory bandwidth) improved by approx. 50% every year.
- In addition: CPU clock rate (CPU clock speed) is improved by approx. 50% every year.
- in principal the same problem as with hard disks: minor improvements on random access, huge improvements on throughput



Source: P. Boncz

Flash.

Flash Chips

- market driven by cell phones, digital cameras, iPods, ...
- persistent storage
- yet no mechanical (moving) parts
- small form factor
- also replacement for disks and tapes
- USB drives

Features of Flash

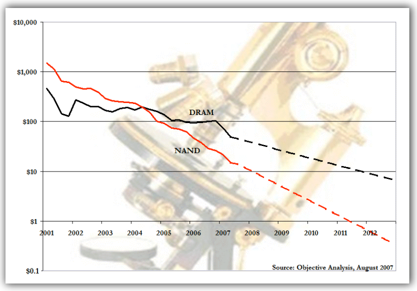
- Chip Read ~ 20 MB/s
  Write ~ 10 MB/s
- N chips have N x bandwidth --> Flash-RAIDS!
- persistent
- Access time ~ 25 µs to start read,
  ~ 100 µs to read a single "2K page"

  ~ 2,000 µs to delete
  ~ 200 µs to write a "2K page"
- beware:
  - you cannot simply overwrite a block
  - you first have to delete the block which is expensive
- power supply ~ 1W for 8 chips and controller

Source: Jim Gray, CIDR 2007 extended

Flash Chips

- cost of a GB of flash versus DRAM



Example: SSD - Solid State Disks

- MTRON MSD-P Series with ATA 7 Standard Interface
  - Burst Read/Write: 133 MB/sec
  - Sustained Read: 100 MB/sec
  - Sustained Write: 80 MB/sec
  - IOPS:
    - (Sequential/Random): 76,000/**16,000**
    - Access Time: less than 0.1 msec

approx. by a factor 100 (!) better than hard disks

- Drawback: expensive
  - about X times more expensive than hard disks
  - But: price is expected to drop due to mass-market (factor 10 in 2012?)

Source: http://www.mtron.net/eng/sub_eb11.asp (extended)

Solution: Hybrid Approach

- combination of hard disk and SSD
- flash used as a disk cache
- in contrast to volatile disk cache flash is persistent
- Hybrid Storage Alliance
  - Fujitsu
  - Samsung
  - Seagate
  - Toshiba
  - Western Digital
  - Hitachi
- see http://www.hybridstorage.org

Flash as DRAM Replacement

- problem: access to flash still limited by maximal disk interface bandwidth
- how to fix this?
- idea:
  - replace one of the CPUs by an additional memory controller
  - use one of the DRAM banks to put in **flash banks** (sic!)
- effect:
  - reads as fast as DRAM
  - writes as slow as flash
  - persistent
  - much bigger readable memory available
  - hundreds of Gigabytes DRAM-fast memory on a single node!!!
- good for read-intensive work-loads
- a startup company in the US is doing this

Storage Management.

Tasks of the Storage System

- Manage external memory devices
- Hide properties of these devices
- Map physical blocks to files
- Control of data transfer from/to DB-buffer
- if necessary:
  - block encoding
  - realize multi-level storage hierarchy
  - software fault tolerance (if RAID is not used)

Control of I/O-Operations

- **Scenario**: hard or flash-disk crashes
- Question: Was block 42 written to the hard disk?
  - Yes
  - No
  - partially (Ouch!)
- How do we detect whether the block was correctly written or not?

Control of I/O-Operations

- If a block resides completely inside a single sector, the hard disk may provide the information on the state of the block. (blocks may be larger than a sector)
- In case of a hard-disk crash this may not work...
- Additional methods:
  - **Parity-bits:**
    - Use parity-bit at the beginning and at the end of each block.
    - initial write of the block: set both bits to false
    - for each write operation on this block: invert both bits
    - if bits differ => block was partially written
  - **Logged write:**
    1. copy old block to a save (new) place
    2. overwrite old block with modified block

Note: every DBMS should log data and operations (we will come back to this)

Storage Management.

A Data-oriented File System.

A Data-oriented File System: Motivation

- selective activation of files
- temporary files
- mixing of different storage media
- short addresses
- Data-Storage = set of files

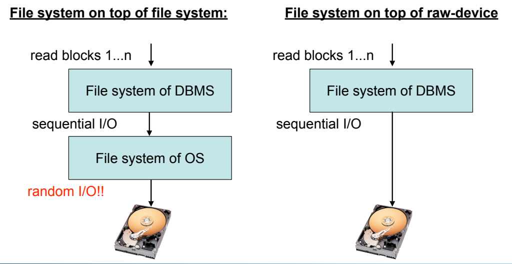
Realization of a File System

- File system contains a catalogue of all files
- file descriptor: name owner, access control list (ACL), size, time stamp, etc.
- management of free external memory (bit lists)
- Access granularity: block (x times 1KB)
  - fixed block size for each file
  - support for sequential I/O
- in the following we will stick to hard disks
- however problem similar for other devices

Note: all of this is already implemented by the operating system!

Raw-devices

- operating system already implements a file system
- if the DBMS implements an additional file system on top of this, ugly things may happen:

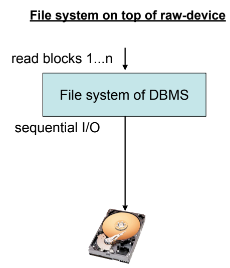


Why not simply use the File System of the Operating System?

- access patterns not optimized for DBMS access patterns
- lack of support for recovery
- would have to be tweaked on all OS-platforms for specific DBMS-requirements
- Several OS file system implementations are proprietary

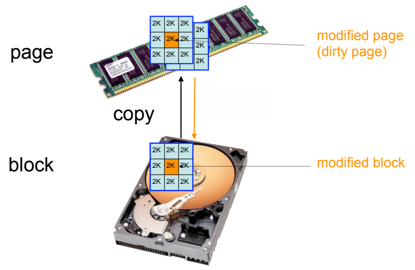
Alternative: Raw-devices

- DBMS-vendor has full-control on file system
- one implementation for all platforms
- better control: sequential versus random I/O (however, hard disk controller may still map blocks to other places...)
- additional advantages (see slides on DB-buffer)



Pages and Blocks

- Page = main memory representation of a block



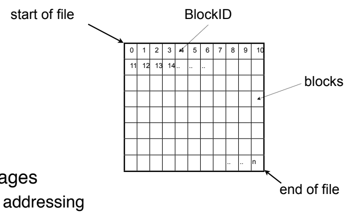
Block Assignment for Hard Disks

- realizes the mapping

  <BlockID> ⟶ <file,offset>

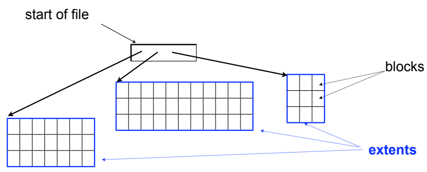
- has **huge** impact on I/O-performance
- huge impact on flexibility of the file system concept (enlargement of files, management of empty blocks)
- three principal approaches
  - static file assignment
  - dynamic extent-assignment
  - dynamic block-assignment
- another practical approach: Unix file system

Static File Assignment



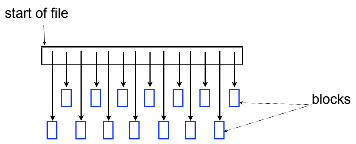
- Advantages
  - direct addressing
  - blocks contiguous on disk
  - excellent performance for sequential I/O
- Disadvantages
  - file has to be allocated entirely
  - no dynamic enlargement

Dynamic Extent-Assignment



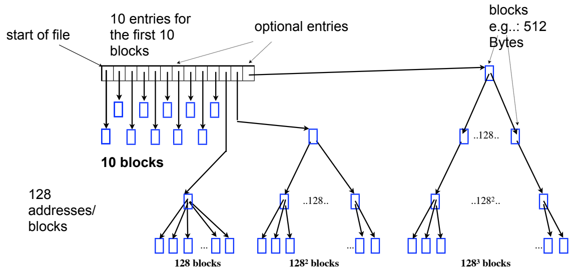
- Advantages
  - Allows allocation of extents on different storage media
  - acceptable for sequential I/O (in particular inside an extent)
- Disadvantages
  - clipping (allocation of unused space)
  - fragmentation (extents not contiguous on disk => random I/O)

Dynamic Block-Assignment



- Advantages
  - highest flexibility
  - no clipping
- Disadvantages
  - fragmentation (blocks not contiguous on disk => random I/O)
  - bad performance for sequential I/O

How does Unix do it?



10 + 128 + 128² + 128³ = 2,113,674 data blocks

Storage Management.

Write-operations.

Replacement Strategies for Write-Operations

- How is a modified page written back to a block on hard disk?
  - **Direct write**
    - page will be written to the position of the assigned old block on hard disk (=overwrite of old block)
    - what happens in case the hard disk crashes during this overwrite operation?
    - Answer: ...
  - **Indirect write**
    - keep old version of the block until the modifying transaction has committed
    - simplifies rollback of transaction
    - Three different approaches:
      Twin Block, shadow storage, differential files

Twin-Block-Algorithm

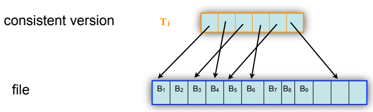
- Goal: atomic insert for a set modifications
- Idea: keep two versions for each block:
  - old, consistent version
  - new, inconsistent version
- In case anything goes wrong, the old version of the block will still be available.
- switch among two versions is done using a global (materialized) bit
- Disadvantage: storage requirements doubled

B_{1A}	B_{1B}	B_{2A}	B_{2B}	B_{3A}	B_{3B}	B_{4A}	B_{4B}	...

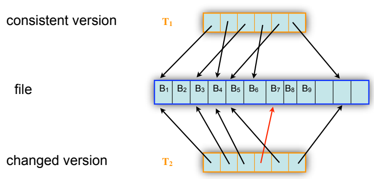
Shadow Storage

- Goal: atomic insert of a set of modifications
- Idea: keep two versions of each **modified** block:
  - old, consistent version
  - new, inconsistent version
- old, consistent state is stored in so-called “shadow pages“

Shadow Storage



Shadow Storage: Insert&Update

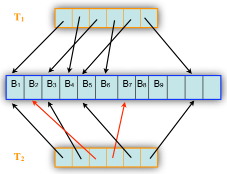


Shadow Storage: Insert&Update

consistent version

file

changed version



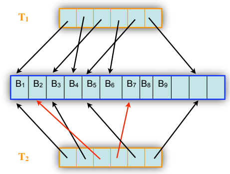
Modified pages are written to free blocks in the file,
i.e., only modified pages will trigger copies of blocks.

Shadow Storage: Crash

consistent version

file

changed version



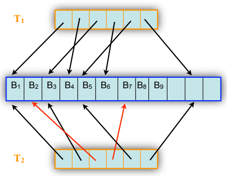
1. throw away T_2, old T_1 becomes current version again
2. free modified blocks in the file

Shadow Storage: Persisting Changes

consistent version

file

changed version



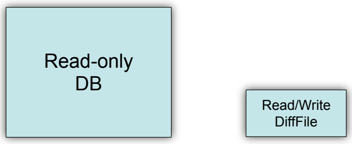
1. write all modified blocks
2. write T_2
3. perform atomic switch to T_2

Discussion

- Advantages
  - Doubles storage requirements only for changed blocks
  - undo of changes easy
  - catastrophic error: compared to direct write strategy likelihood is higher that the DBMS is in a consistent state
- Disadvantages
  - helper data structures may become big (> 1 block)
  - helper data structures may have to be kept on disk
  - sequential access to data is destroyed over time
  - not suitable for big databases

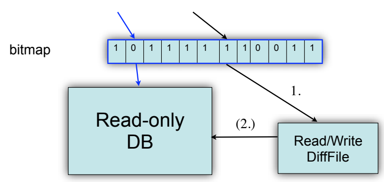
Differential Database File

- Idea
  - keep 2 versions of the DB
    - Read-only DB: consistent old state
    - Read/write "differential file" (DiffFile): modifications (inserts/updates)
  - merge modifications into read-only part at fixed points in time
  - Analogy: publishing house collects corrections to create a new edition of a book



Differential File

- Which version of the DB contains the most recent block?
- Algorithm (Bloom-Filter)
  - hash-function h(B) => Bit
  - Bit = 0: block in read-only DB
  - Bit = 1: block **maybe** in differential file



Differential File: Discussion

- Advantages
  - easy to implement
  - easy to extend for transactions
  - differential file corresponds to incremental backup
  - merge of two versions may be exploited to defragment block order on disk
- Disadvantages
  - Locking during the merge (can be fixed easily)
- We will come back to this idea int the context of indexing....

Tablespaces (Oracle)

- 1:n relationship to files
- administrator manages tablespaces and determines which tables resides on which device
- Creating a tablespace

```
CREATE TABLESPACE my_tablespace DATAFILE
file1.dat SIZE 40 MB, file2.dat SIZE 30 MB
```

- Extending a tablespace (not necessarily automatically)

```
ALTER TABLESPACE my_tablespace ADD DATAFILE
file3.dat SIZE 20 MB
```

- Assigning relations to tablespaces

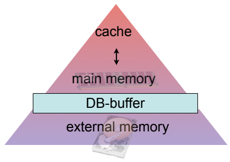
```
CREATE TABLE customer(...) TABLESPACE
my_tablespace
```


Storage Management.

Buffer Management.

Tasks of the DB-Buffer

- DB-buffer resides in-between external storage and main memory
- keeps a set of buffer frames (slots of size 1 page)
- Goal: decrease I/O-accesses by exploiting temporal access locality



- only a limited set of pages may be held in the DB-buffer

Methods of the DB-Buffer

- GET P_x
  - returns a reference to P_x
- FIX P_x
  - page P_x may not be evicted anymore
- UNFIX P_x
  - page P_x may be evicted
- PAGE_IN_BUFFER P_x
  - returns true, if buffer contains page P_x
    implementation: hash-table
- CHOOSE_PAGE
  - chooses a page to evict
  - returns a reference to this page
    implementation: page replacement strategy (following slides)

Implementation of GET

	Cost
GET P_x	
• If PAGE_IN_BUFFER(P_x):	
- FIX P_x	cheap
- return reference to P_x	
• Else	
- if no empty slot available in buffer:	
• P_i = CHOOSE_PAGE	
• If page P_i was modified (dirty):	cheap
◦ write page P_i to external memory	
- load block B_x from external memory into buffer slot	**expensive**
- FIX P_x	**expensive**
- return reference to P_x	cheap

Write Strategies

- **FORCE** (write-through)
  - writes modified pages no later than transaction commit
  - drawback: high I/O-cost
  - bad response times for modifying transactions
  - simple recovery
- **NO FORCE** (write-back)
  - writes modified pages when pages get evicted from the buffer
  - Note: page may be evicted only after a long period of time
    - What happens if a page that was modified very often is not evicted at all?
    - What is the state of the page's corresponding block on hard disk?

⇨ this only works when combined with logging!

Read Strategies

- **Preplanning**
  - analyze applications
  - try to determine set of pages that will be read by the application
  - Drawbacks:
    - imprecise supersets
    - not always possible
- **Prefetching**
  - exploit clustering of data
  - overlap CPU- and I/O-operations (double buffering)
  - Drawbacks:
    - wrong decisions
    - hard disk is already performing read-ahead

Read Strategies

- **Demand Fetching**
  - get page only when it is requested by the application
  - Advantage: no additional effort
  - Drawbacks:
    - bad response times
    - applications have to wait for data

Page Reference Patterns

- Reference patterns = sequence of accesses to pages in the buffer
- **Sequential**
  - Example: table scan of l pages
    P_1, P_2, P_3, P_4,... , P_l
- **Hierarchical path**
  - Example: index scan
    P_1, P_{11}, P_{42}, P_{77}, P_{34}
- **Random access**
  - Example: RID-scan
    P_{456}, P_{1124}, P_{422}, P_7, P_{343}
- **Cyclic path**
  - Example: nested-loops join, inner loop of size n
    P_{1001}, P_1,..., P_n, P_{1002}, P_1,..., P_n, P_{1003}, P_1,..., P_n

Page Replacement Strategies (Implementation of CHOOSE_PAGE)

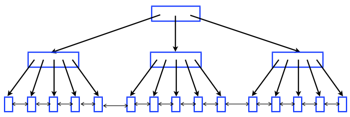
- least-recently used (LRU) vs. most-recently used (MRU)
- first-in first out (FIFO)
- least-frequently used (LFU)
- clock, second-chance, gclock
- least-reference density

LRU-k

- **Idea**
  - Consider k-latest accesses not only the latest
  - for page eviction consider the k-latest accesses to a page
  - page with the oldest k-latest accesses gets evicted
- LRU-1 = LRU
- Drawback: history of a page needs to be stored (even for pages that were already evicted from the buffer!)

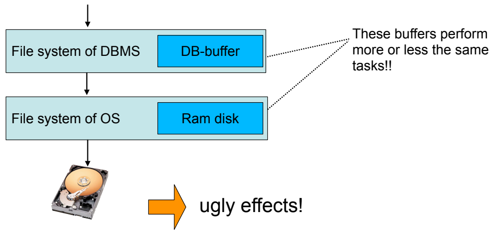
LRU-k

- **Idea**
  page with the oldest k-latest access gets evicted
- Example
  B+-tree, 5 leaves per node, access pattern: path access + 5 leaves
  buffer size: 6 (resp. 8) pages
  LRU-1 vs. LRU-k?



DB-Buffer and Virtual Memory

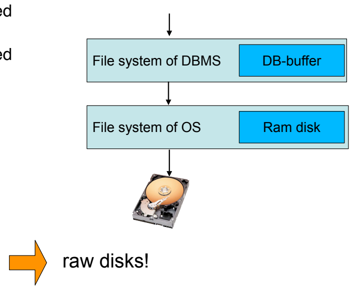
File system on top of file system:



DB-Buffer and Virtual Memory

- Double page fault
  - page P is not contained in DB-buffer
  - page P is not contained in ram disk
  - page has to be read from swap or file
- Why two buffers?
- How do we know that the operating system has written a block to hard disk?

File system on top of file system:



raw disks!

Page- versus Record-Oriented Buffering

- Page-oriented
  - \+ easy to manage
  - \+ easy to implement
  - \- overhead due to oversized pages (dead space)
- Record-oriented (Why manage an entire page if only one record is accessed?)
  - \- management per record
  - \- harder to implement
  - \- extra copying effort to isolate records
  - \- memory fragmentation
  - \+ effective memory usage

Next Topic: Mapping Data Items to Pages.