

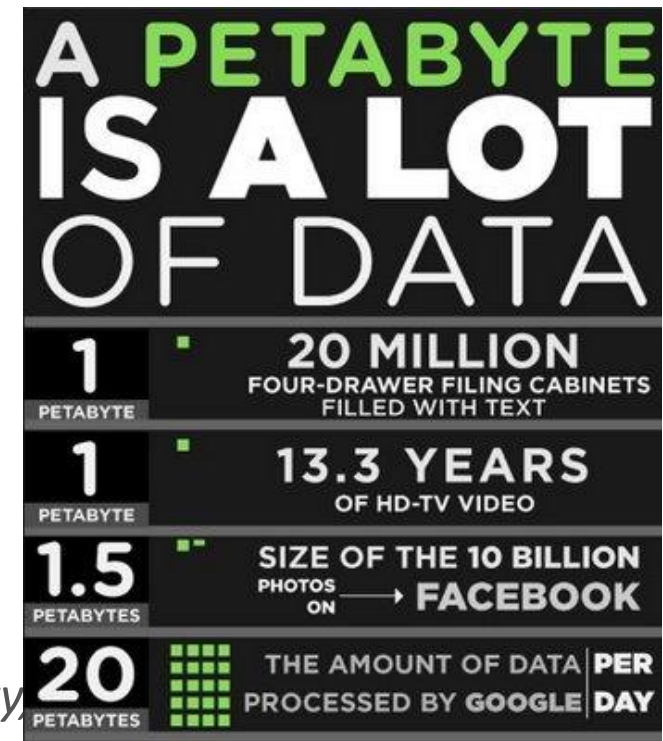


# 1 Introduction



*The Web is a huge source of information: search engines (Google, Yahoo!) collect and store billions of documents*

- 20 PB processed every day at Google (2008)
- eBay has 6.5 PB of user data + 50 TB/day (2009)
- Structured data, text, images, video
  - 35 hr of video uploaded to YouTube every min
- World of Warcraft utilizes 1.3 PB of storage space
- Valve Steam delivers 20 PB of content monthly
- Data is partitioned, computation is distributed
  - Reading 20PB would take 12 years at 50MB/s



*[S. Melnik: The Frontiers of Data Programmability]*

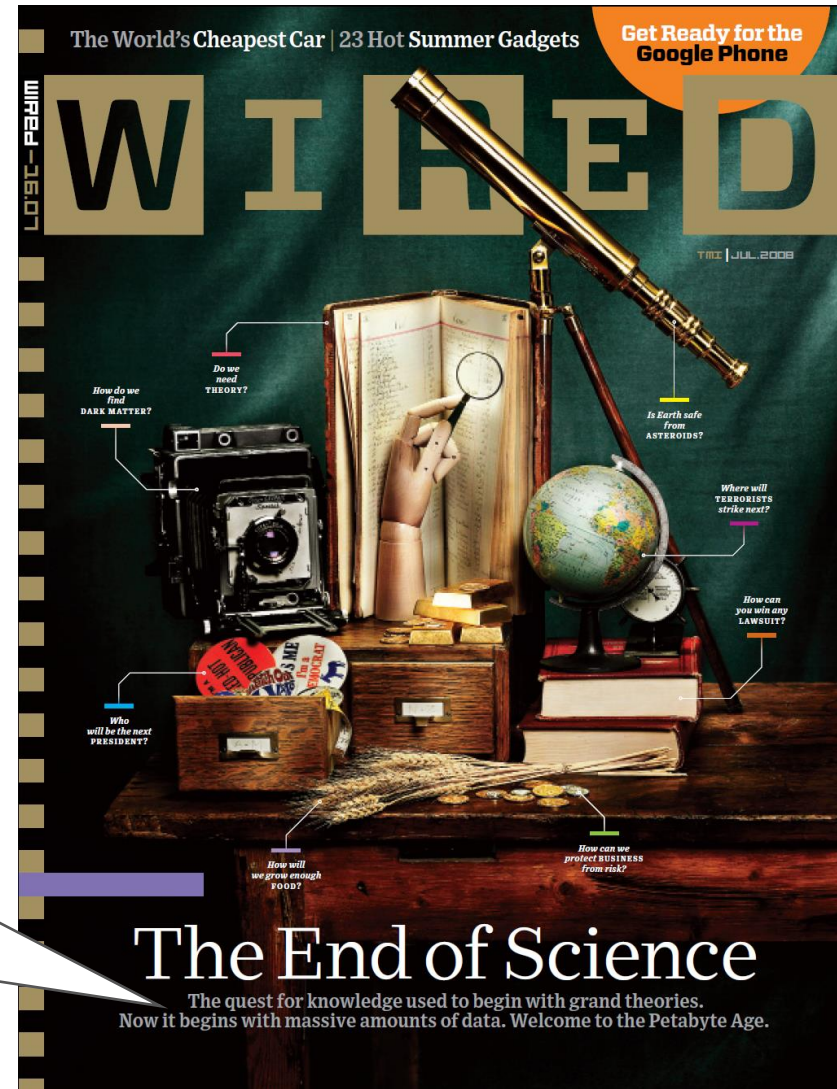


### *New Realities*

- TB disks < \$100
- Everything is data
- Rise of data-driven culture
  - CERN's LHC generates 15 PB a year
  - Sloan Digital Sky Survey (200 GB/night)
  - Microsoft Research Maps (formerly TerraServer)

The quest for knowledge used  
to begin with grand theories.  
Now it begins with massive  
amounts of data.

Welcome to the Petabyte Age.





## Data Workloads

### OLTP

OnLine Transaction Processing

- Random access to a few records
- CRUD
- \$ per latency

Read-heavy

Write-heavy

### OLAP

OnLine Analytical Processing

- Scan access to a large number of records
- Focus on sequential disk I/O
- \$ per CPU cycle

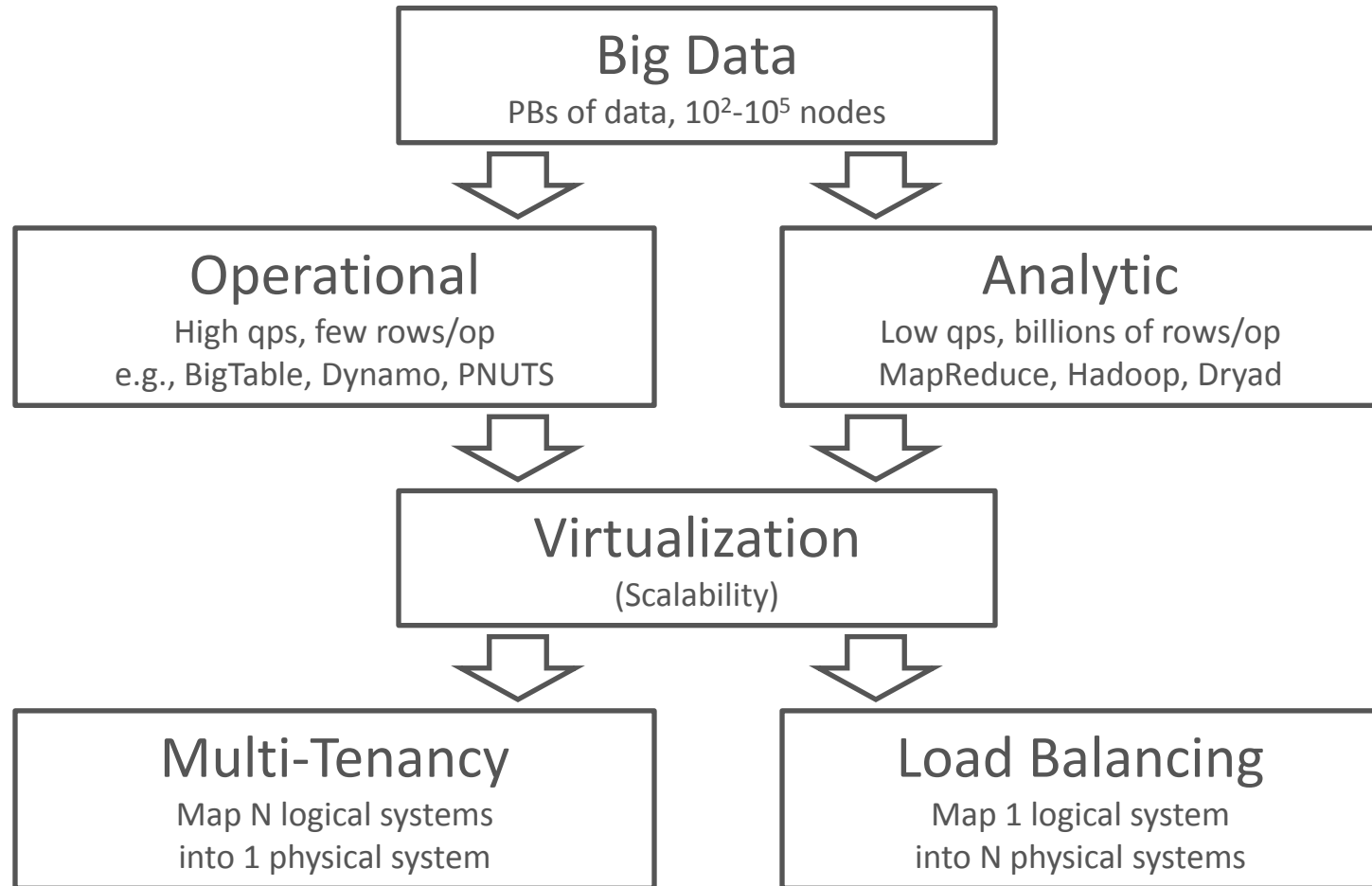
By rows

By columns

Unstructured

### Combined

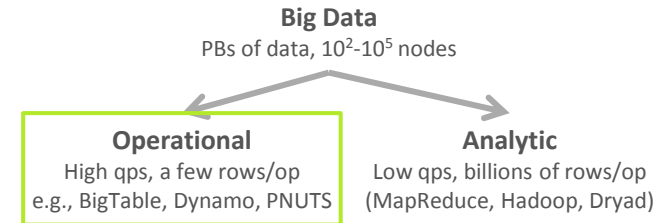
(Some OLTP and OLAP tasks)





## *Managing data for Web companies and providers*

- Data for Google Earth (roads, borders, geographic Web), Google Finance, ...
- Metadata and data for Flickr, del.icio.us, ...



## *Operational data for Internet Services*

- Session data for Yahoo! (100+ Mill. users), Microsoft Windows Live Messenger (250 million unique users/month), Hotmail, ...
- Amazon.com: bestseller lists, shopping carts, preferences, product catalogs, ...
- Social applications like Facebook (30.000 MySQL databases @1800 DB servers)

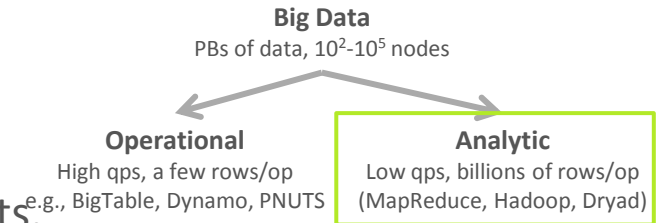
## *Hosted Database Services (DB Outsourcing)*

- Amazon S3, SaaS applications like Salesforce.com



*Analysis queries become more and more complex*

- Discovery of statistical patterns is compute intensive,
- e.g. Netflix recommendations, Google search results,
- Facebook social network analysis
- May require multiple passes over the data



*In addition:*

- Tasks/queries become increasingly hard to predict.
- Makes conventional database approaches (e.g., indexes) hard to apply.



## *Given*

- 100M movie ratings (500k customers, 20k movies)

## *Goal*

- Predict 3M unknown ratings
- 10% less RMSE than Netflix's Cinematch

## *Cracked in September '09*

- But data is still available
- The winner is a blend of multiple teams:
  - Bob Bell, Martin Chabbert, Michael Jahrer,
  - Yehuda Koren, Andreas Toscher, Chris Volinsky
- Used multiple models: Matrix factorization, neighborhood models, restricted Boltzmann machines, ...



## *Competition dataset was small*

- 100,000 DVD titles and more than 10 million subscriber
- 10s of billions of ratings in practice
- TBs of data













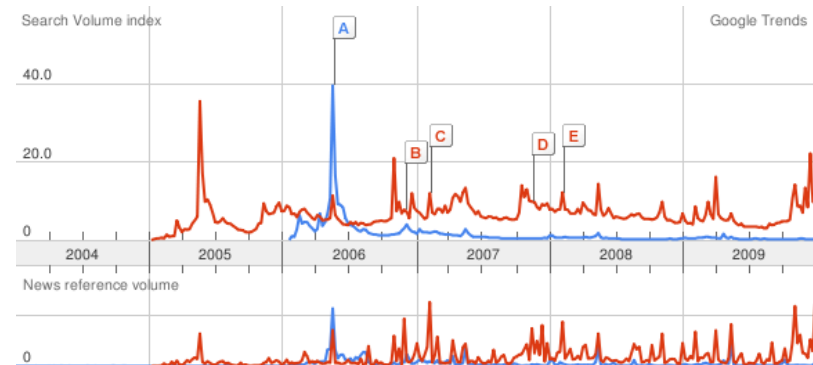
- Estimated 1 Mio. low-cost commodity servers in 2009
- Over 200 GFS clusters at Google
- One cluster: 1000 - 5000 machines
- Pools of tens of thousands of machines retrieve data from GFS clusters that run as large as 5 peta-bytes of storage
- Aggregate read/write throughput can be as high as 40 gigabytes/second across the cluster
- 6000 MapReduce applications so far

[Web](#) [Images](#) [Videos](#) [Maps](#) [News](#) [Shopping](#) [Mail](#) [more](#) ▾

Google **squared** labs

russian submarines					
Item Name	Image	Description	Complement	Beam	Length
<input checked="" type="checkbox"/> Alfa class submarine		Two <b>Alfa class submarines</b> (the fictional V. K. Konovalov and E. S. Polkovsky) are featured in Tom Clancy's novel, <i>The Hunt for Red October</i> ...	4 possible values	9.5 metres	81.4 metres
<input checked="" type="checkbox"/> Bars class submarine		The <b>Bars class</b> were a group of <b>submarines</b> built for the Imperial Russian Navy during World War I. A total of 24 boats were built between 1914 and 1917 ...	33	4.5 m	68 m
<input checked="" type="checkbox"/> Karp class submarine		The <b>Karp Class</b> were a group of <b>submarines</b> built by Krupp GermaniaWerft for the Imperial Russian Navy. The boats were ordered in the 1904 emergency programme ...	28	2.7 m	39.6 m
<input checked="" type="checkbox"/> Osetr class submarine		Most Russian (and Soviet) submarines had no "personal" name, but were only known ... <b>Kasatka class submarine</b> - <b>Osetr class submarine</b> - <b>Som class submarine</b> ...	24	3.6 m	22 m
<input checked="" type="checkbox"/> Narval class submarine		The <b>Narval class</b> (sous-marins d'escadre, "fleet submarines") were <b>patrol submarines</b> built for the French Navy in the 1950s. ...	47	7.8 m	78.4 m
<input checked="" type="checkbox"/> Kaiman class submarine		The <b>Kaiman class</b> were a group of <b>submarines</b> built for the Imperial Russian Navy before World War One. They were designed by Simon Lake and built by the ...	34	4.3 m	40.2 m
<input checked="" type="checkbox"/> Kasatka class submarine		The <b>Kasatka class</b> were a group of <b>submarines</b> built for the Imperial Russian Navy. The six boats were built between 1904 and 1905. They were designed by I.G. ...	24	3.5 m	33.5 m
<input checked="" type="checkbox"/> Russian submarine K-141 Kursk		<b>K-141 Kursk</b> was an Oscar-II class nuclear cruise missile submarine of the <b>Russian Navy</b> . lost with all hands when it sank in the Barents Sea on August 12 ...	44 officers, 68 enlisted	18.2 m	4 possible values

Google Squared



Google Trends



- Uses Hadoop Distributed File System (HDFS) to store copies of internal log and dimension data sources
- Uses Hadoop Map/Reduce engine as a source for reporting/analytics and machine learning
- Two major clusters
  - A 1100-machine cluster with 8800 cores and about 12 PB raw storage.
  - A 300-machine cluster with 2400 cores and about 3 PB raw storage.
- Each (commodity) node has 8 cores and 12 TB of storage.
- Built a higher level data warehousing framework using these features called Hive (see the <http://hadoop.apache.org/hive/>)



### *Two basic techniques*

- Scale up – Use a small number of large servers
- Scale out – Use a large number of small server
- Scale in – Multiple apps, tenants, virtual machines on a single machine

Scale Up: Salesforce.com	Scale Out: RightNow
<ul style="list-style-type: none"><li>▪ 1.75 billion rows</li><li>▪ 139,000 tenants (35,000 customers)</li><li>▪ 8 Oracle RAC databases (17,000 tenants/instance)</li><li>▪ 170 million transactions per day</li></ul>	<ul style="list-style-type: none"><li>▪ 50 TB of data in DB+NFS</li><li>▪ 3000 tenants (1800 customers)</li><li>▪ 200 MySQL servers 1-100s tenants/instance)</li><li>▪ 17 million transactions per day</li></ul>

CRM Case Studies (October 2007)



### *Lack of Hardware Reliability for Scale Up*

- Reliable Hardware is really expensive (...and yet it might fail)
- Use cheap, unreliable hardware and be prepared to deal with failures.

### *Effects*

- Failure handling becomes really simple. If in doubt, just replace the machine.
- (At some point) scales almost trivially: If you know how to deal with failures in 1,000 nodes, you also know how to deal with failures when you have 5,000 nodes.

### *Elasticity*

- Enables incremental adjustments of capacity

### *Lifecycle Management*

- Enables incremental rolling upgrades
  - Gain production experience with a small set of users
  - Eliminate down-time by using side-by-side systems
  - Immediately roll back if problems arise



### *Scale out to 1000+ of machines*

- Data, request load or both are too large for single machine
- Many services are deployed in multiple locations

### *Highly available even on unreliable (commodity) hardware*

- Typical first year for a new cluster (J. Dean, LADIS 2009)
  - ~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
  - ~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours)
  - ~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hrs)
  - ~1 network rewiring (rolling ~5% of machines down over 2-day span)
  - ~5 racks go wonky (40-80 machines see 50% packetloss)
  - ~1000 individual machine failures
  - ~thousands of hard drive failures slow disks, bad memory, misconfigured machines ...

→ *Slow disks, bad memory, misconfigured machines, etc.*



*Frequent load balancing and capacity adjustments are required to achieve good utilization*

- Otherwise over-provisioning is required to handle temporary load peaks
- Must not shuffle around large amounts of data
- Must be operationally simple

*Large data sets get distributed across multiple servers*

- Read-queries can be efficiently processed only if the data is distributed so as to minimize inter-server communication
- Write-queries require distributed transactions (not necessarily 2PC)

*The CAP Theorem*

- Consistency of data, Availability of data, Partition-tolerance (ability to scale out)



### *Reminder*

- Capital Expenditures – cost of acquiring or upgrading physical assets such as equipment, property, or buildings
- Operational Expenditures – costs for the day-to-day running of a business, including salaries, rent, and utilities

### *Reduce expenditures: Multi-Tenancy*

- Consolidate multiple tenants into the same process
- Worth the effort only if enough tenants fit on the given hardware
- Reduces CapEx because resource utilization is increased
- Reduces OpEx because there are fewer processes to manage



### *Cloud computing*

- Why buy machines when you can rent cycles?
- Use of rented computing resources like servers, storage, services, or applications
- Wikipedia: “In general, cloud computing customers do not own the physical infrastructure, instead avoiding capital expenditure by renting usage from a third-party provider. They consume resources as a service and pay only for resources that they use. [...] Sharing "perishable and intangible" computing power among multiple tenants can improve utilization rates, as servers are not unnecessarily left idle (which can reduce costs significantly while increasing the speed of application development). A side-effect of this approach is that overall computer usage rises dramatically, as customers do not have to engineer for peak load limits.”
- Key characteristics: Agility, Cost (capital expenditure -> operational expenditure), Device and location independence, Multi-tenancy, Reliability, Scalability, Security(?), Sustainability





"All that matters is results —  
I don't care how it is done"

*Acquisition Model*  
**Service**

"I don't want to own assets — I want  
to pay for elastic usage, like a utility"

*Business Model*  
**Pay for usage**

"I want accessibility from anywhere from  
any device"

*Access Model*  
**Internet**

"It's about economies of scale, with  
effective and dynamic sharing"

*Technical Model*  
**Scalable, elastic,  
shareable**

### Cloud Computing

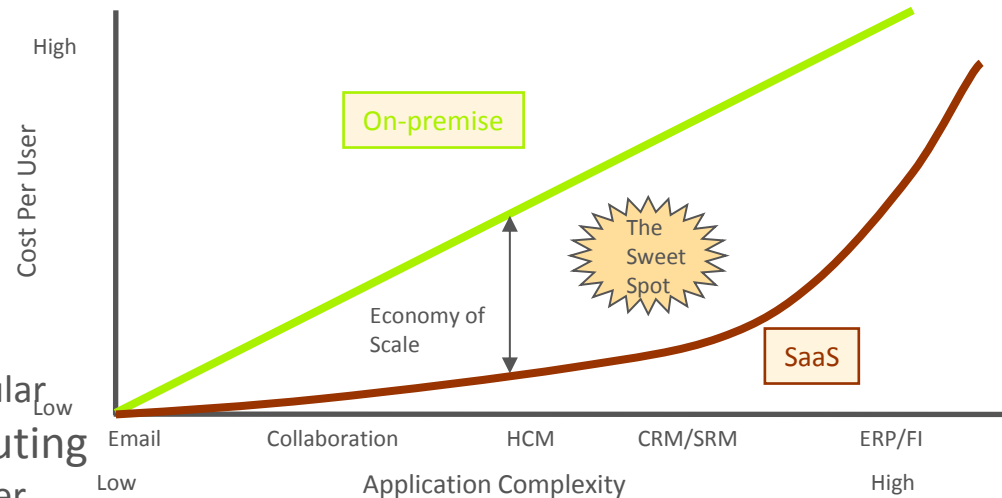
A style of computing where massively scalable, IT-enabled capabilities are provided "as a service" across the Internet to multiple external customers.





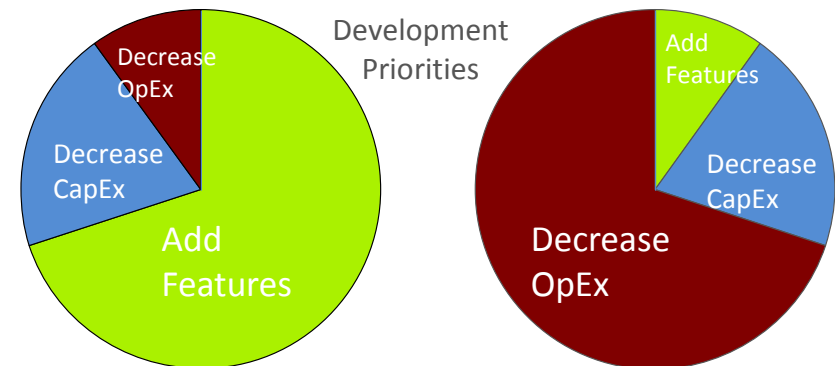
## Cost

- „Pay as you go“ for HW and SW
  - No upfront cost / investment: CapEx vs. OpEx
- Utilization
  - statistical allocation of resources
  - Scale down if service becomes less popular
- Out-source and commoditize computing
  - HW automatically gets cheaper and faster
  - economy of scale for admin: patches, backups, etc.
- Failures: cost of preventing and having failures



## Time to market

- Avoid unnecessary steps, e.g. HW provisioning, purchasing, test



On-Premises Software

Software as a Service



### *Cloud Software as a Service (SaaS)*

- Use provider's applications over a network



### *Cloud Platform as a Service (PaaS)*

- Deploy customer-created applications to a cloud



### *Cloud Infrastructure as a Service (IaaS)*

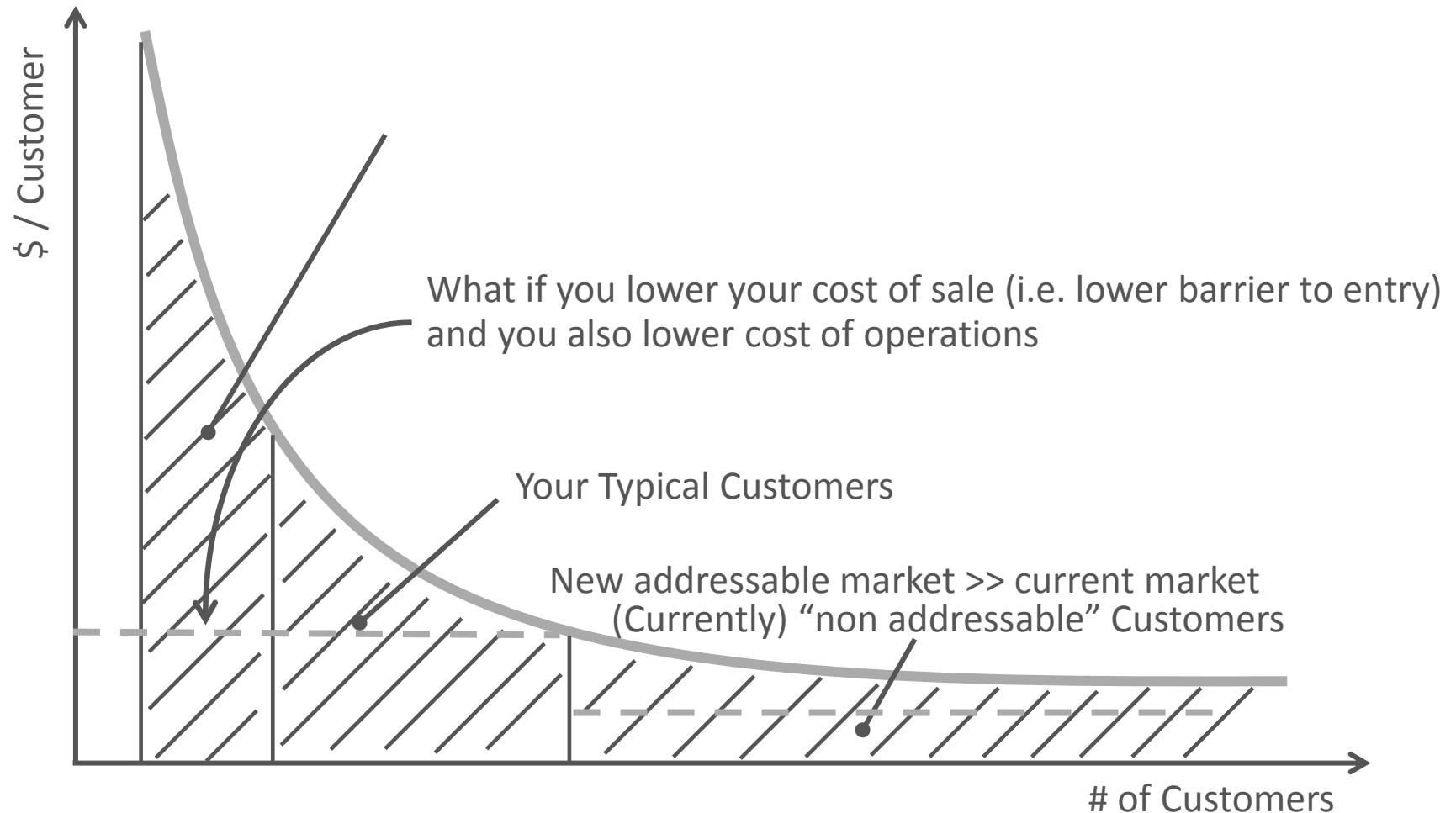
- Rent processing, storage, network capacity, and other fundamental computing resources



## > The Long Tail



*Your Large Customers*





### *Common, Location-independent, Online Utility on Demand [1]*

- Common implies multi-tenancy, not single or isolated tenancy
- Utility implies pay-for-use pricing
- on Demand implies ~infinite, ~immediate, ~invisible scalability

### *Alternatively, a “Zero-One-Infinity” definition [2]*

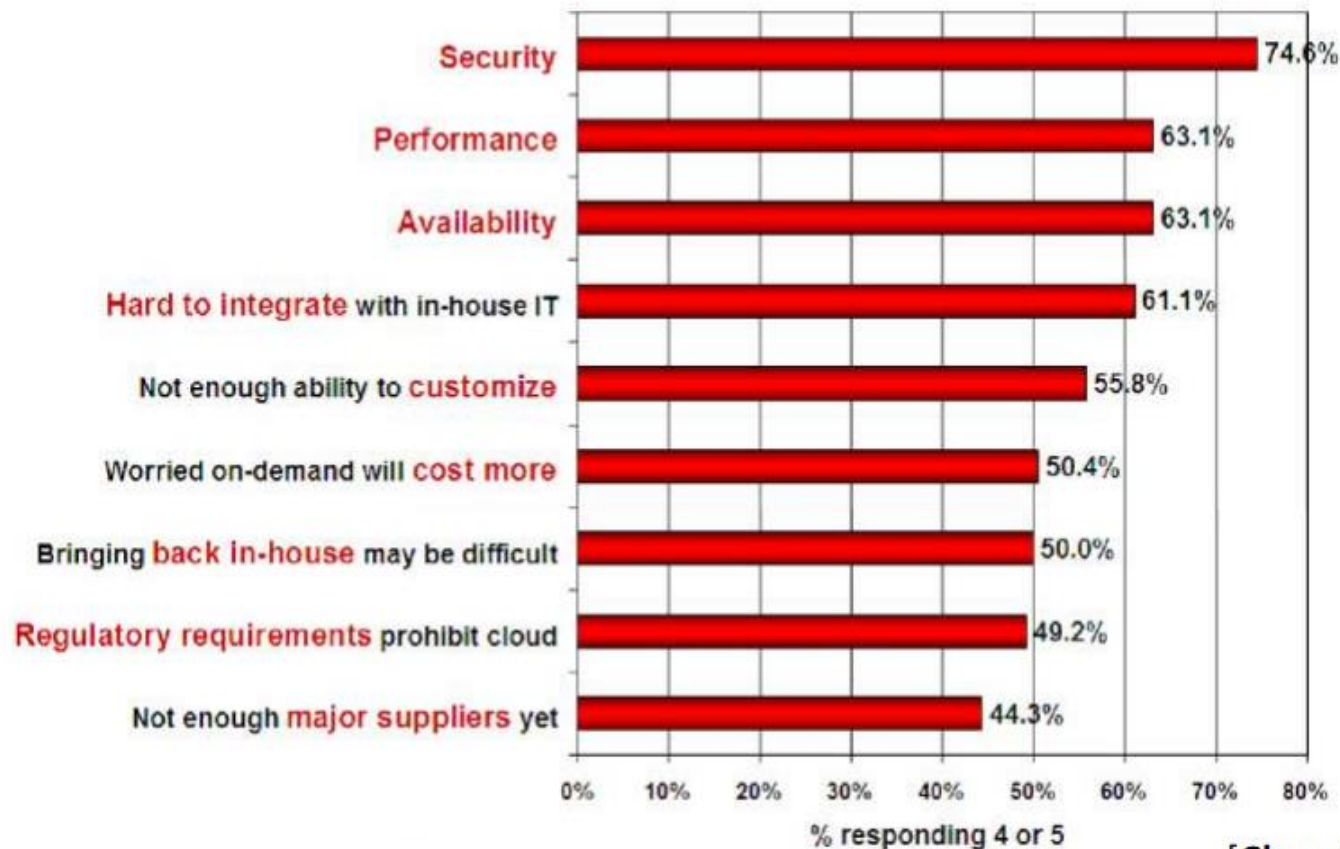
- 0 On-premise infrastructure, acquisition cost, adoption cost, support cost
- 1 Coherent and resilient environment – not a brittle “software stack”
- Scalability in response to changing need, Integratability/
- Interoperability with legacy assets and other services
- Customizability/Programmability from data, through logic, up into the user interface without compromising robust
- multi-tenancy

[1] Joe Weinman, Vice President of Solutions Sales, AT&T, 3 Nov. 2008

[2] From The Jargon File: “Allow none of foo, one of foo, or any number of foo”



Q: Rate the **challenges/issues** ascribed to the 'cloud'/on-demand model  
(1=not significant, 5=very significant)

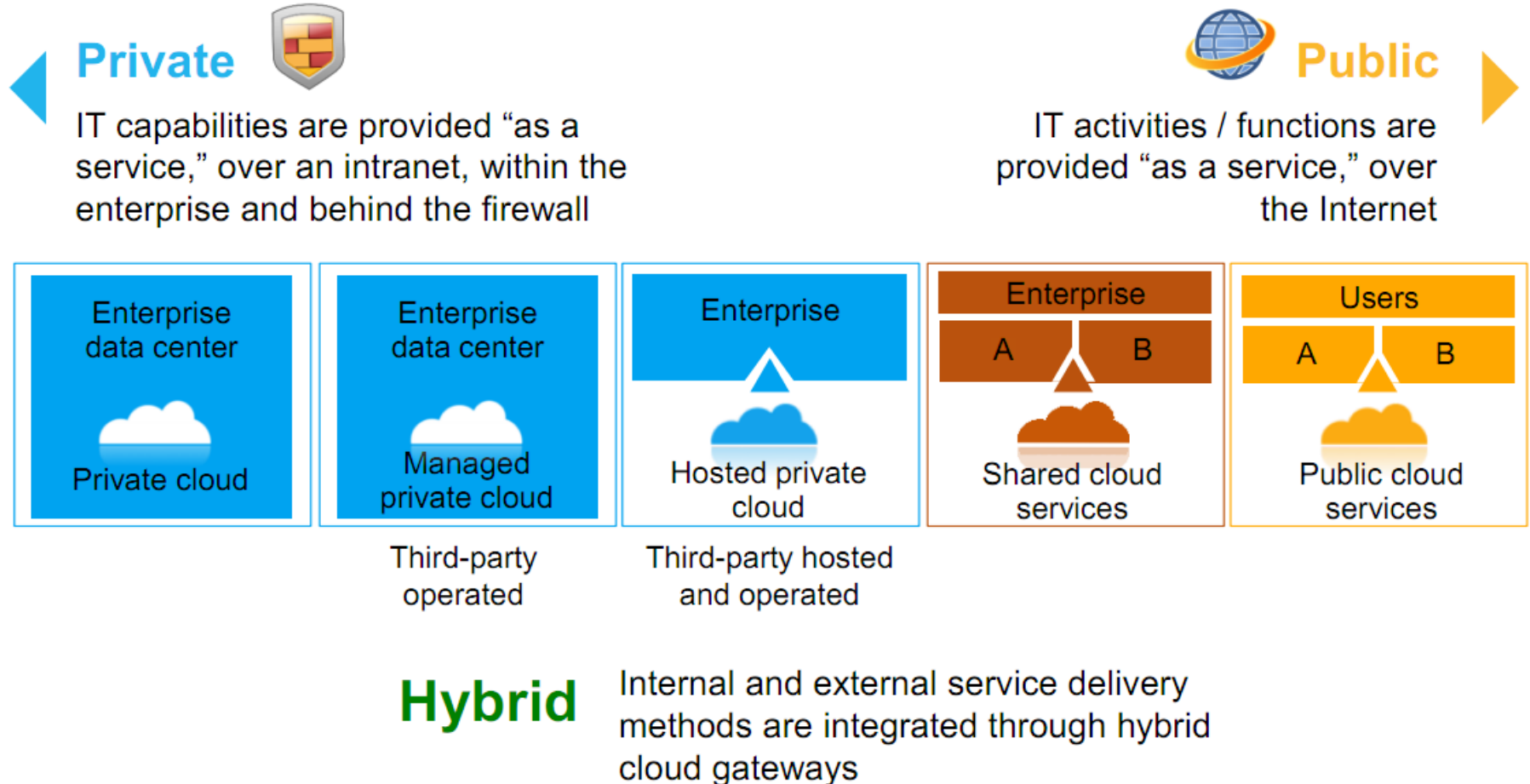


Source: IDC Enterprise Panel, August 2008 n=244

[Chow09ccsw]



## *Spectrum of Deployment Options for Cloud Computing*





## *Traditional DB*

- strict consistency
- internal data format (data lock-in)
- sophisticated access method
- defined schema (semantics known to the system/optimizer)
- semantics of the operators known to the system (closed set of operators)  
plus some black-box UDFs

## *Map/Reduce Programming Model*

- only read access, focus on scalability
- use (CSV) files as data container
- scan and shuffle methods
- schema defined during query time (schema on the fly)
- 2nd order functions; semantics of the operator is totally unknown to the system, only a contract exists between operators and infrastructure







## *Limited Scalability*

- preserving consistency in a distributed environment is costly...
- ensure serializability even if the application can ensure no conflicting writes

## *Data Lock-in*

- necessity to put data completely into control of the system (effort to load data into a database system, perform runstats)
- no native support for regular CSV files
- time to query is critical (no time to load and reorg data)

## *Static in terms of Schema Definition at design-time*

- need to follow the „data comes first, schema comes second“- principle

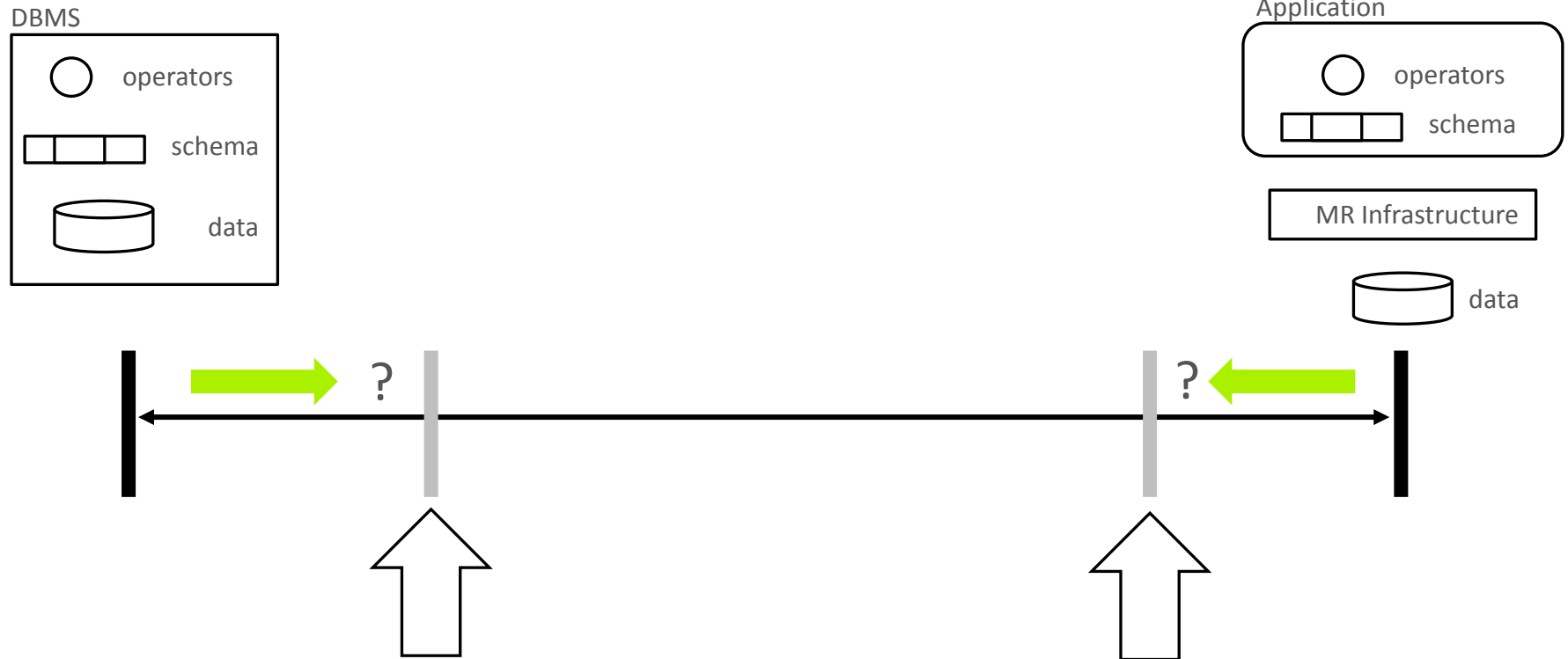
## *Robustness within Query Execution*

- R1 recovery for queries/statements, no easy compensation when losing a node

## *NOT the main problem*

- no problems with the data model – the tabular model is still very popular (with flexibility)
- no problems with the query model – SQL is just fine (everybody knows SQL)
  - NoSQL systems are hard to program, e.g. Cassandra 1.0 did not ensure consistency within a row!
  - responsibility is left to the application programmer (e.g. store redundant hash codes to compare versions at the application level)

## > Current Trend: platforms & new systems



scalable, high performance  
read-only relational  
query execution engine

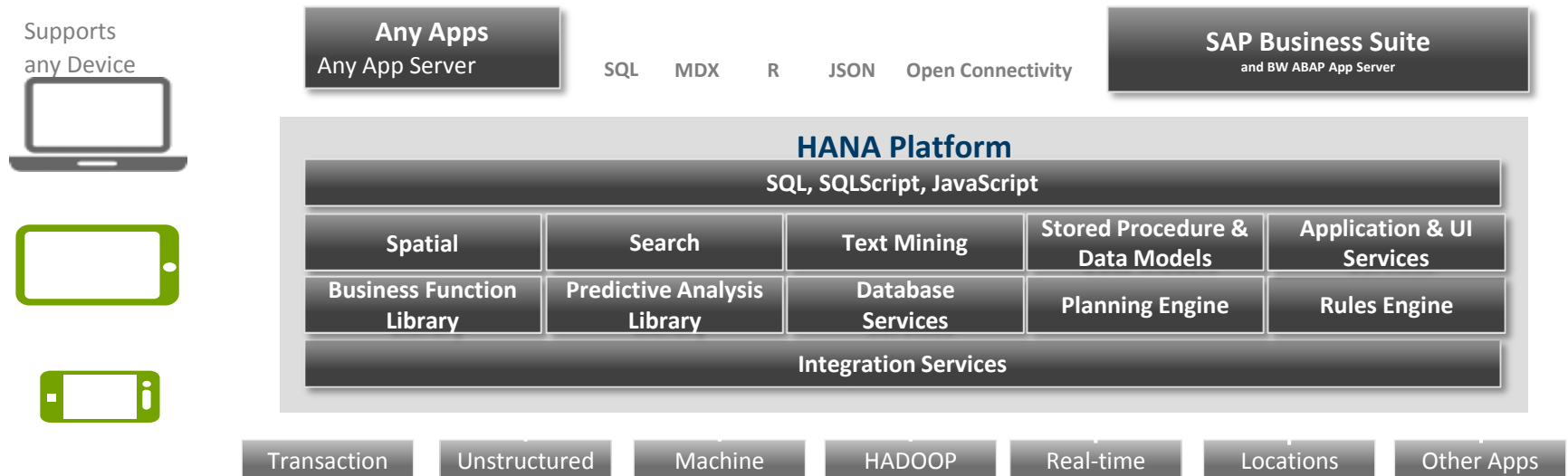
e.g. Presto, ...

scalable, read-only MapReduce  
engine with SQL interface and  
tight coupling to a relational engine



*...from Database System to Data Management Platform*

- Comprising different different DM service within a single eco-system
- Tight integration with compute- as well as data-intensive applications



SAP HANA Platform converges Database, Data Processing and Application Platform capabilities & provides libraries for Predictive, Planning, Text, Spatial, and Business Analytics to enable business to operate in real-time



# *Layering of Data Management Systems*



## Why is Data Management and Transaction Processing so difficult?

- Answer: the “-ities” ....
  - Reliability system should rarely fail
  - Availability system must be up all the time
  - Response time within 1-2 seconds
  - Throughput thousands of transactions/second
  - Scalability start small, ramp up to Internet-scale
  - Security for confidentiality and high finance
  - Configurability for above requirements + low cost
  - Atomicity no partial results
  - Durability a transaction is a legal contract
  - Distribution of users and data

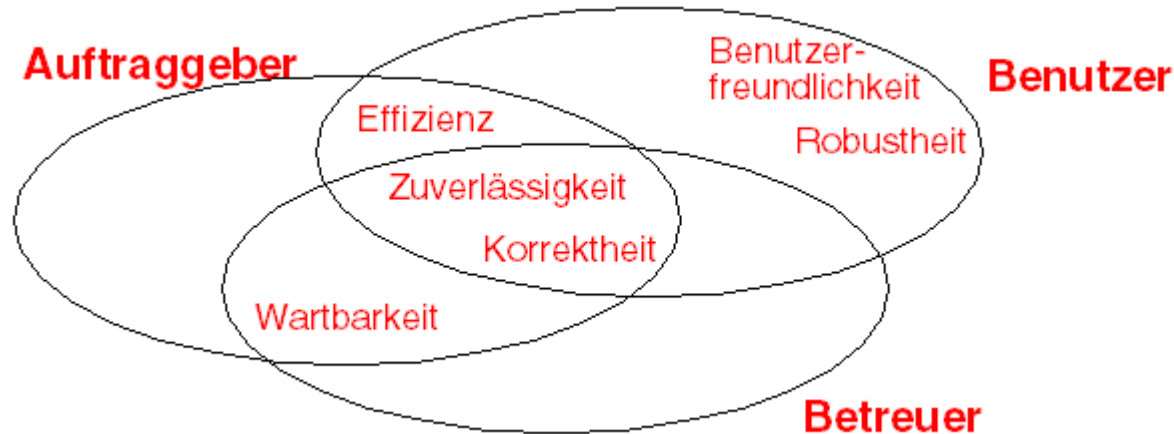
## Why is Data Management and Transaction Processing so important?

- Backbone of almost all larger software systems
  - „Relational Database Systems are the foundation of western civilization“
- Big Business (e.g. Lowell Report)

# > DBS als Beispiel eines generischen Softwaresystems



## *Innere und äußere Qualitätskriterien*



## *Bekannte Konzepte der Implementierung*

- Geheimnisprinzip (Information Hiding)
- Hierarchische Strukturierung durch Schichten
- eine Schicht realisiert einen bestimmten Dienst, den sie an der Schnittstelle “nach oben” höheren Schichten zur Verfügung stellt
- eine Schicht nimmt Dienste unterliegender Schichten in Anspruch

# > Allgemeine Betrachtungen zur Schichtenbildung



## *Beobachtung*

- Schichten entstehen sukzessive durch Strukturierung oder Erweiterung des Anwendungsprogrammes
- Schichtenbildung ist eine Möglichkeit, um die bei der Software-Entwicklung geforderten Ziele (Wartbarkeit, Erweiterbarkeit, etc.) zu erreichen

## *“Veredelungshierarchie”*

- entlang der Schichten wird nach oben hin “abstrahiert”
- “Eine Hauptaufgabe der Informatik ist systematische Abstraktion”

H. Wedekind

## *Problem: Anzahl der Schichten in einem Softwaresystem*

- $n = 1$ : monolithisches System -> keine Vorteile der Schichtenbildung
- $n$  sehr groß: -> hoher Koordinierungsaufwand
- Daumenregel:  $n$  typischerweise zwischen 3 und 10



### *Vorteile als Konsequenzen der Nutzung hierarchischer Strukturen*

- Höhere Ebenen (Systemkomponenten) werden einfacher, weil sie tiefere Ebenen (Systemkomponenten) benutzen können.
- Änderungen auf höheren Ebenen haben keinen Einfluss auf tiefere Ebenen.
- Höhere Ebenen können abgetrennt werden, tiefere Ebenen bleiben trotzdem funktionsfähig.
- Tiefere Ebenen können getestet werden, bevor die höheren Ebenen lauffähig sind.
- 

### *Jede Hierarchieebene kann als abstrakte oder virtuelle Maschine aufgefasst werden*

- Programme der Schicht  $i$  benutzen als abstrakte Maschine die Programme der Schicht  $i-1$ , die als Basismaschine dienen.
- Abstrakte Maschine der Schicht  $i$  dient wiederum als Basismaschine für die Implementierung der abstrakten Maschine der Schicht  $i+1$

### *Eine abstrakte Maschine entsteht aus der Basismaschine durch Abstraktion*

- Einige Eigenschaften der Basismaschine werden verborgen.
- Zusätzliche Fähigkeiten werden durch Implementierung höherer Operationen für die abstrakte Maschine bereitgestellt.





## *Layered Architecture*

- Modularisation, Maintainability, ...
- Implementation of generic software components

## *Generic Software components*

- NOT built for a specific application (e.g. management of a DVD collection)
- BUT built for a class of applications (e.g. management systems in general)

## *NOTE*

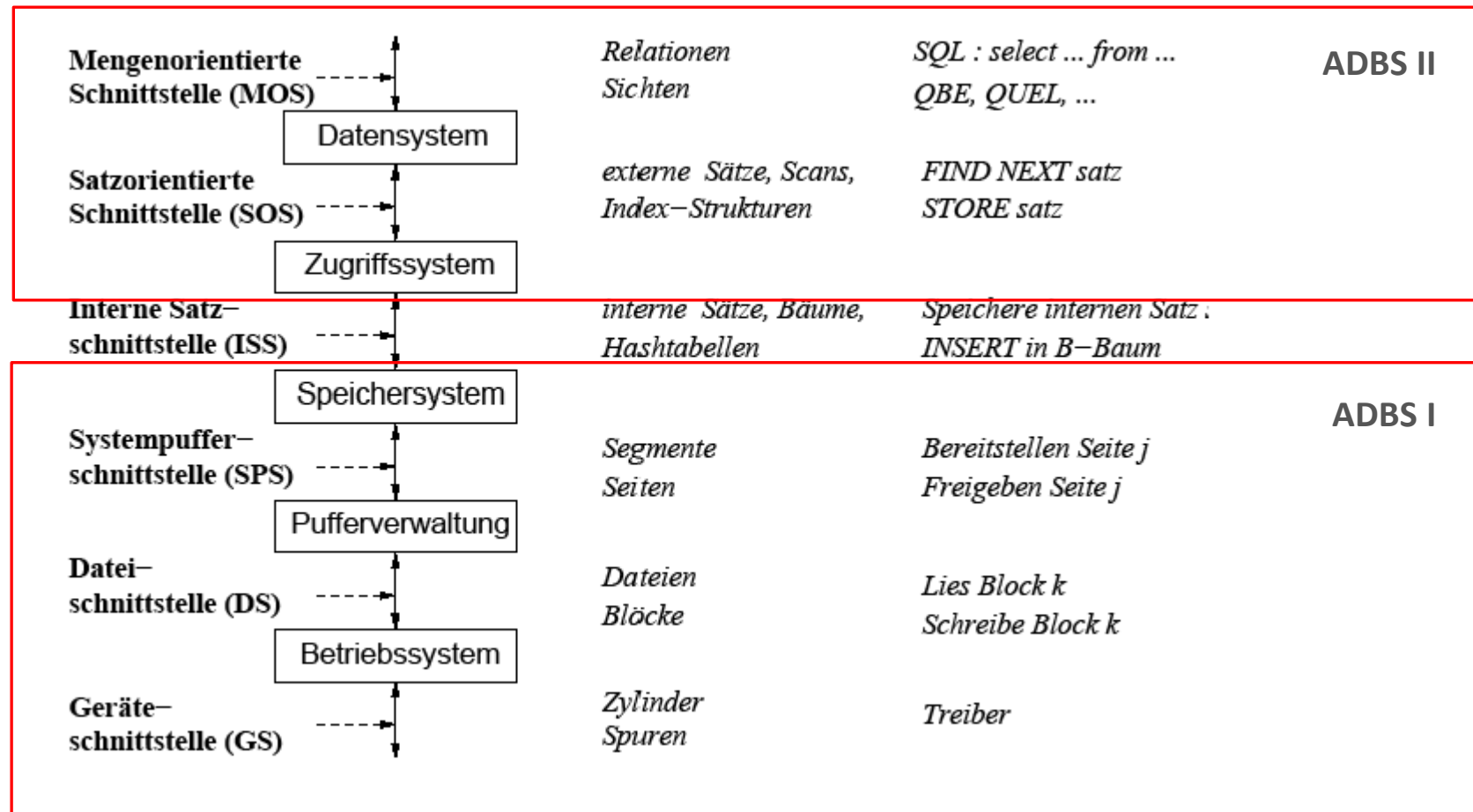
- Database systems generic software systems independent of a specific application

## *Implication*

- the specific data structures or properties of an application have to be customized/created for a specific application
  - by specification (schema design)
  - not by programming

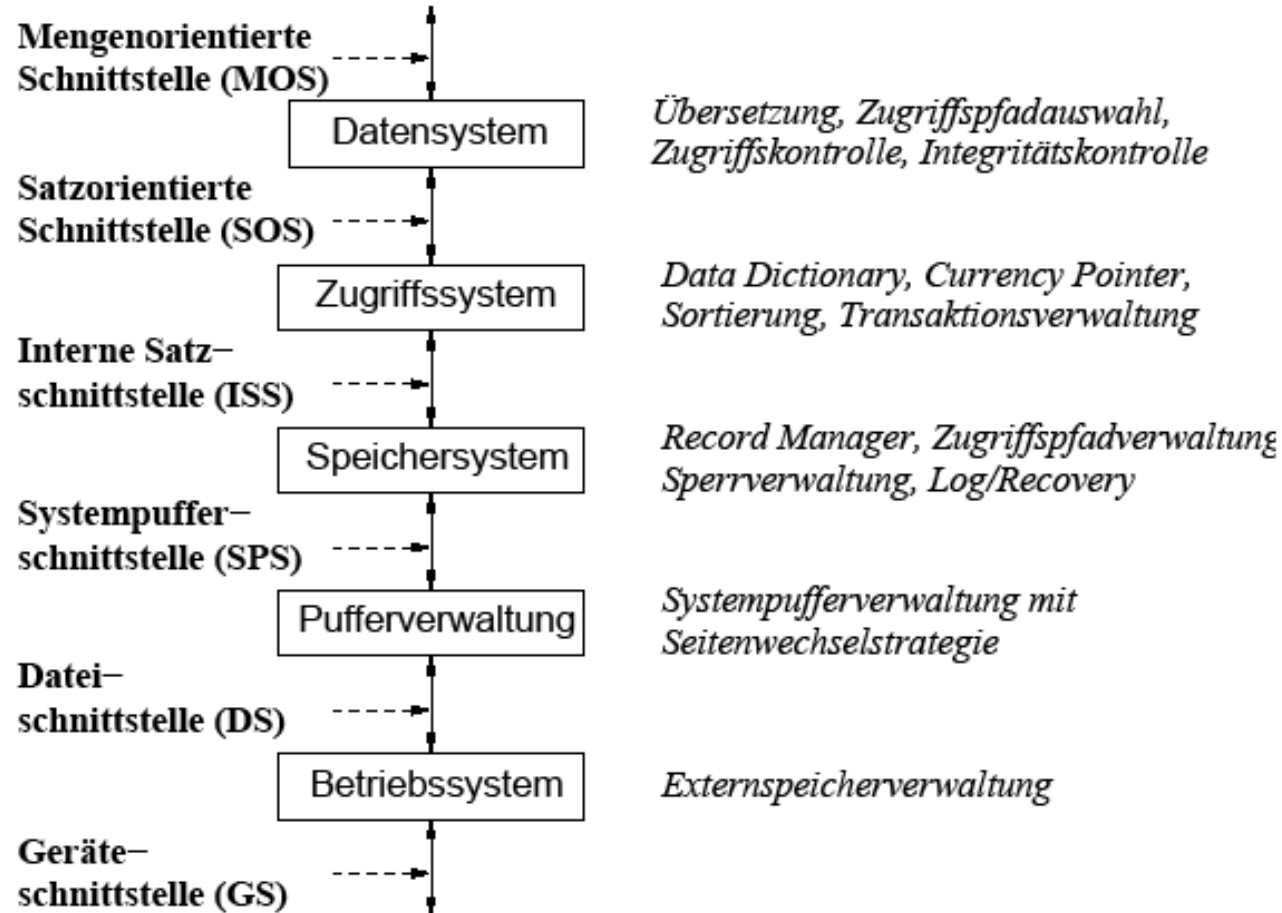


## Übersicht über Datenobjekte und Objektstrukturen





## Übersicht über Funktionen





### *MOS: mengenorientierte Schnittstelle*

- deklarative Datenmanipulationssprache auf Tabellen und Sichten (etwa SQL)

### *SOS: satzorientierte Schnittstelle*

- navigierender Zugriff auf interner Darstellung der Relationen
- manipulierte Objekte: typisierte Datensätze und interne Relationen sowie logische Zugriffspfade (Indexe)
- Aufgaben des Datensystems: Übersetzung und Optimierung von SQL-Anfragen

### *ISS: interne Satzschnittstelle*

- interne Tupel einheitlich verwalten, ohne Typisierung
- Speicherstrukturen der Zugriffspfade (konkrete Operationen auf B-Bäumen und Hash-Tabellen), Mehrbenutzerbetrieb mit Transaktionen

### *SPS: Systempufferschnittstelle*

- Umsetzung auf interne Seiten eines virtuellen linearen Adressraums
- Typische Operationen: Freigeben und Bereitstellen von Seiten, Seitenwechselstrategien, Sperrverwaltung, Schreiben des Protokolls

### *DS: Dateischnittstelle -> Umsetzung auf Geräteschnittstelle*



*Data Management is the core of modern software applications*

- Relevance as well as big business
- Trend towards data management platform – well beyond single systems

*Wide spectrum of data management requirements*

- traditional transaction processing (robust, reliable, ...)
- big volume data crunching (advanced analytical algorithms)
- data stream processing (high velocity, one look at the data)

*What are common architectural principles*

- The layered architecture!!!

*ADBS – Part I*

- From HW (compute, memory) to Key/Value interface
- Q: How to provide an efficient Key-Value interface (indexing, etc.)

*ADBS – Part II*

- From Key-Value interface to user/application-interaction
- Q: How to efficiently exploit an existing Key-Value interface (query optimization, etc.)

## > Structure of the ADBS class

