1. Distributed databases are the union of database management systems and computer networking
   a. Authors (OV) introductory thesis of this book
   b. They propose that these approaches appear to be diametrically opposed. Do you agree of disagree? They base this claim/idea on:
      i. Database systems are centralized management systems in which each application defines and manages its own data
      ii. Computer networking allows many computers to communicate and cooperate, decentralizing control and data
   c. Why is this claim of opposition false?
      i. The fundamental concept in database systems is data independence, totally independent of storage organization or centralization

2. Stop – what is data independence? Review from the last class
   a. Data independence renders applications programs (e.g. SQL scripts) immune to changes in the logical and physical organization of data in the system
      i. logical organization – changes in schema, e.g. adding a column or tuples does not stop queries from working
      ii. physical organization – changes in indices, file organization, etc.

3. How do we merge the concepts of DBMS and computer networks
   a. Authors propose that the ultimate goal of databases is integration not centralization
   b. I prefer to think at a higher level, data independence drives this whole process and leads us from centralization to centrally organized (with distributed elements) to fully distributed all the while preserving data independence

4. Distributed computing – definitions
   a. Authors (OV) definition – “a number of autonomous processing elements (not necessarily homogeneous) that are interconnected by a computer network and that cooperate in performing their assigned tasks.”
   b. This definition does not consider what is being distributed which can include
      i. Processing logic or processing elements (portions of the “same” program or multiple programs that cooperate to produce unified of integrated results
      ii. Function – implies heterogeneity, perform difference parts of a program at different locales. The simplest and first form of distributed computing is breaking function at a single computer into sub-parts and using specialized hardware (e.g. floating point, math and dsp coprocessors) or (network display etc.) hardware
      iii. Data – may be distributed throughout a network and the data what processing is executed at which site.
      iv. Control – distribute the management of the execution of the distributed tasks.
   c. A simple way to consider distributed computing is systems that either move data to processing elements, move processing to data, or some combination. This include control, which is just another element of
   d. Need to describe

5. What are the goals of distributed computing?
   a. Match processing organization to computer networks
   b. Deploy processing power and storage to support inherently distributed tasks (Web-serving, Internet commerce)
   c. (I like a practical, economic driven view) Perform very large tasks on commodity hardware. Cheaper and more scalable than building large systems. This is why one might use distributed computing for applications that can be performed on monolithic hardware and obviates definition (a) which is strangely self-referential.
i. This argument applies to most of the advantages of distributed computing including availability, reliability, redundancy, performance, etc. It is not required to distribute computing to achieve safe computing environments. In fact, sometimes it is much more challenging because distribution introduces complexity.

ii. *E.g.* distributed systems are much more susceptible to attacks and faults like?
   1. denial of service
   2. man-in-the-middle
   3. network partitions

6. Database architectures
   a. Networking allows tasks to be conducted on many computers
   b. Parallel processing allows execution to be sped up
   c. Distributed of data protects sites from failures

7. Centralized systems
   d. Single system hosts whole DBMS and data
   e. Single user systems – no concurrency control, this is really a relic, because the single user OS is now defunct, any DB must be able to act as a transaction server or support multiple users
   f. Mutli-user systems

8. Client-server systems
   g. Multiple types of servers
      i. Transaction servers – clients function ship transactions to a central location through an interface like ODBC. This means that the server is effectively a multi-user database with an ODBC server
      ii. Data-server systems – clients access pages or files This is effectively a single system database over shared storage. However, storage services can be enhanced to split database function across the machines. *E.g.* storage that supports tuples, data typing.
      iii. *Given that client/server systems devolve into multi-user systems, what are the key advantages of this approach?* Resource deployment. And storage capabilities (caching, locking). Also, move non-db application off of server, which are increasingly computer intensive.
      iv. Client-server is really about resource deployment rather than software architecture as far as we are concerned in this class, because we are looking at low-level DBs..

9. Often C/S and multi-user systems have multiple processors, but these processors support only coarse grained parallelism, split work be query or whatever the OS can break out as a thread. *Is it a worthwhile goal to parallelize single tasks? For which applications? For which is it not reasonable?*

10. Parallel systems – improve I/O speeds and distribute work among many processors at a fine granularity
    h. Increase throughput by processing many small transactions in parallel
    i. Increase response time by breaking out each task
    j. In fact it is not quite this simple, throughput and response time are closely related. MVA tells us that response time $R = C/T$, $C=$number of completed tasks, $T=$observation period. So, on average, for systems with jobs waiting (non-zero queue length) the response time varies as a function of the throughput.

k. Speedup and Scaleup
   i. Speedup, how much faster does the system goes as you add n processors, *diagram SKS 16.4.* Run tasks in less time by increasing parallelism.
      1. Linear speedup
      2. Sub-linear speedup (with a decreasing factor)
      3. The definition in SKS is confusing, but correct. $Ts/Tl$ where $Ts$ is the speed on a smaller machine and $Tl$ is the speed on a larger (parallel) machine. I find it easier to think in terms of rates $Rl/Rs \equiv Ts/Tl$
      4. *What factors lead to $<n$ speedup? Does anyone have a good example? Matrix multiplication from (CLR)?* (lookup matrix mult.) Speedup is related to the degree of parallelism inherent in the operation and architectural issues.

   ii. Scaleup, ability to solve much larger tasks in the same amount of time. Run larger tasks in same time by increasing parallelism. *Diagram SKS 16.5*
      1. Measure of speedup as a function of problem size on the same resources.
2. Q is a problem and Qn is an n-times larger problem
   a. Q takes time Ts on machine Ms
   b. Qn takes time Tl on machine Ml, which is M times larger than Ms
   c. The scaleup is the ratio of Ts/Tl (equiv Rl /Rs) as a function of Qn

iii. What is the relationship between speedup and scaleup.

1. SpU is ScU normalized by factor Q.
   a. SP = Ts/Tl(n) = Rl(n)/Rs
   b. SC = Ts/Tl(Qn) = Ts/Q*Tl(n) = Rl(n)/Q*Rs

2. Noticing that Tl(Qn) = Q*Tl(n), it takes Q times longer to
   complete a job Q larger on any machine (Is this reasonable?) We
   show scaleup to be a normalized form of speedup.

1. What factors work against efficient parallelism?
   i. Startup costs – *examples* coordinated jobs among all nodes, distributing data
   ii. Interference – resource blocking, synchronization
   iii. Skew – break down job into different size pieces that take different amounts of time.
   Skew can result from problem (not all tasks can be divided evenly), from execution
   (maybe some processors hit a high latency I/O or have a pipeline stall), or from
   hardware heterogeneity.

11. Distributed database systems – definitions and concepts
    a. Author’s (OV) – a collection of multiple, logically interrelated databases distributed over a
       computer network
       i. Not just a collection of files, files must have structure and relationship. In contrast to
        the Web. However, there is lots of intermediate ground, which are sometimes called
         ad-hoc databases, e.g. things on the Web that can be queried in forms.
    b. Again, very general and, therefore vacuous, it is not necessarily reasonable to generalize into
       a single concept, as we see when we generate a taxonomy for databases next week.
    c. Distributed DBMS – software that make DDBS transparent to the users

12. Develop figure 1.8 by steps showing fragmentation and replication
    a. What are the goals of fragmentation? Load balance and load distribution.
    b. What are the types of fragmentation?
       i. Horizontal, place tuples from a relation at multiple locations. This ia analogous to
        what relational algebra operator
       ii. Vertical fragmentation, place columns (attributes) from a relation at multiple
        locations.

13. Reviewing the concept of transparency
    a. The “separation of higher-level semantics of a system from lower-level implementation
       issues.” “hides” implementation details
    b. “Fully transparent access means that the users can pose queries without paying any attention
       to the fragmentation, location, or replication of data, and let the system worry about these
       issues.”
    c. *Why is this transparent and not opaque?*
       i. Example of transparency, 802.11b the wireless link is transparent to the applications
          that use Ethernet.
       ii. Opaque (black box) cannot see inside.
       iii. Really a subtle point about whether the abstracted software component is a pass
        through or an interface.

14. The transparencies in a DDBMS shield the user from the operational and implementation details of a
    system, pushing the complexity of managing distributed data into the DDBMS, shielding the
    user/application.
    a. Types of transparency in a DDBS
       i. Network transparency (aka distribution transparency) hide operational details of the
          network. Which includes transparency location transparency (the task/command is
          independent from the system on which the task/command is executed) and naming
          transparency (a unique name must be provided for each object in the database).
       ii. Replication transparency – users need not be aware of the existence or management
          of multiple copies of data. Tasks and commands are independent of replicas.
iii. Fragmentation transparency – user does not know or care how data are placed and distributed. Compiling a global query into fragment queries is part of implementing fragmentation transparency.

b. A data driven view of the “transparency” properties of DDBS – I contend that network, replication, and fragmentation transparency fall out of providing data independence and that data independence is a general concept that dictates all of the design properties.

15. “The level of transparency is a compromise between ease of use and the difficulty and overhead cost of providing high levels of performance.”

a. Discuss this concept in class.

b. E.g. fragmentation transparency hides the performance implications of performing a distributed query and seemingly simple queries can take a long time to run.

16. Where can transparency be implemented and give examples:

a. Access layer – language, e.g. compiling global queries into local queries

b. Operating system – use native OS services to provide guarantees for DB. E.g. use copy services of storage/file system to provide DDBMS transparent replication. Frequent approach for banks, called remote copy. Also RPC is a technique for network transparency.

c. Within the DBMS

i. Benefit – more information, the database is the locale where application semantics and physical details meet and therefore better decisions can often be made.

1. OSes do not have knowledge of high level application semantics (like the concept of relations of joins).

2. Compilers do not know where data are necessarily stored and the costs of accessing data.

ii. Downside – complexity, creates a monolithic role for the DB and makes the system more complex

17. Reliability through distributed transactions. DDBMSs have replicated components (not only data but things like network routing too) and replication eliminates single points of failure. Distributed transactions are the system tool to manage data that are replicated and fragmented.

18. Transactions convert a database from a consistent (w.r.t what? – integrity constraints) state into another consistent state. This includes:

a. Concurrency transparency – users/applications do need to know or care about multiple transactions against the same data at the same time. This is a local DB concept

b. Failure Atomicity – preserve transaction semantics when distributed components fail. Why is this or is it not a type of transparency? Because users/applications can see the outcomes of distributed transactions through data unavailability and transaction aborts. It is a semantic guarantee, not a transparency.

19. The purported benefits on DDBMSs are the same as that of distributed computing. DDBMSs are just a special class of application.

a. Better availability and reliability.

b. Higher performance through parallelism

c. scalability

Discussion question – Some people assert the “with the widespread use of high-speed, high-capacity networks, distributed data and data management functions no longer make sense and it may be much simpler to store data at a central site?”

What is wrong with this argument

• Does not consider the difference between bandwidth and latency. Many if not most distributed applications are latency constrained rather than bandwidth.

• Does not consider the ability to decluster work in a switch-network of computers. Achieving on overall aggregate bandwidth much greater than that of any single high-capacity link.

Discussion question: “Distributed systems require additional hardware (communication mechanisms), thus have increased hardware costs”
This statement indicates that the authors revise the second edition enough between 1992 and 1999. The mechanisms for distributed computing are common enough on today's computers, because networks are ubiquitous. A more important cost factor is commodity parts, which are leveraged heavily to build distributed systems much more cheaply than monolithic systems.