1. Distribution Transparency

In the context of this course, the concept of transparency means hiding the true details of an implementation and making it seem like a single system.

Access - Hide differences in data representation and how a resource is accessed

The system should work the same way, regardless of the kind of machine it's used from. The world wide web is a good example of this; a web-page is accessed the same way regardless of what kind of a computer or operating system the user has. A disadvantage could be if a standard followed some convention that would be painfully difficult to implement on some systems. Also some platform-specific advanced features would break this transparency on other systems.

Location - Hide where a resource is located

The user does not see a difference whether the data is stored in a file, a database, another computer or generated on the fly, and the user doesn't need to know anything about it, the system can take care of that. This also helps implementing other forms of transparency.

Migration - Hide that a resource may move to another location

A system can point some name to some resource, and update it to point to the new location, should the resource move somewhere else. This allows others accessing the resource to keep referencing it the same way as they used to, but allows the system itself to be changed t.e. to provide better performance or reliability.

Relocation -Hide that a resource may be moved to another location while in use

A tighter form of migration. Might require the system to tolerate network topology changes and need a fast way to renegotiate with servers to tell about the new location or some sort of temporary multi-homing scheme. This kind of hiding might be complex and costly to implement and may not be worth it in systems where topology rarely changes and the occasional service break can be tolerated.

Replication - Hide that a resource is replicated

Replication allows for added performance (the system can serve a bigger audience) and fault tolerance (when one server fails, requests will be redirected to working ones). Hiding the replication allows the system to still look like one system, but a somewhat more robust and stress-tolerant one. Also, resources can be replicated to servers on different sites (co-located), improving access times to the resource and the system's fault tolerance around the world.

Transaction - Hide that multiple competing users perform concurrent actions on the resource

Users should not be able to break anything by accessing a resource at the same time with others. Eg. user ordering books from an online store should not have their order mixed up with another customer's only because they happened to be ordering at the same time. Mutual exclusion through locks or database transactions (temporarily locking the related tables or rows to make inserting the customer's order atomically) should be in used.

Failure - Hide the failure and recovery of a resource

The user should not need to know if a system has crashed. Hiding this usually shows up as a some delay in how the system responds, which is a usual effect from many other situations as well. Replication can sometimes be used to go around a failed part of the system while it recovers.

Persistence - Hide whether a (software) resource is in memory or on disk

Persistence hiding is important in big software systems. A good implementation of this would make it seem like everything about the program is in memory at the same time, but in the background, reads things from disk to memory when they are needed and removes unused resources from memory when more memory is needed. This enables software to start up faster, to have a much smaller memory footprint and to be run on commodity hardware.

2. The Domain Name Service

a) Availability

The DNS service distributed horizontally keeps the name service available even if some name server goes down. If a primary name server fails or is rebooted, a secondary name server is used instead. As a disadvantage, having to replicate servers costs more than running a single server.

Vertically distributed, a single name server does not need to know all the addresses in the internet. It can check if it knows about the name it was asked for, and if not, pass the problem along to another name server that might. Not having all the entries replicated on all name servers doesn't hurt availability much, since the name servers can cache responses, and usually the often used resources are available in the cache. (non-authoritative answers)

b) Reliability

If reliability would mean how easily a name request would fail because of a failed server, distributing purely horizontally, the bigger the amount of servers the more reliable the system. Distributed purely vertically, the bigger the amount of servers, the higher the possibility that a server in the used "pipeline" fails.

c) Performance

Distributing horizontally allows more requests to be processed by the system at the same time, making answering a bunch of questions quicker, but not speeding up any single request.

Distributing vertically would allow each server to search from a smaller database, allowing for quicker decisions on whether the problem needs to be passed along or not. With caching involved, the searches would be done from a relatively small address table with the often used addresses already in there, and other addresses would be answered through a series of small searches on different servers.

d) Scalability

The answer can somewhat be found in answer c. Horizontal scalability allows for more users to have their requests in at the same time. Vertical scalability gets each request to spend a shorter time on a single server. The best would be to distribute both ways, effectively resulting in a tree.

3. Shared file-access

File sharing as a distributed service has made the shared resources much better available. In systems like BitTorrent, the original source of the resource doesn't even need to be present anymore if all the parts of the resource can be found somewhere. Also the bandwidth or transfer limits of the original source are no longer a limiting factor.

NFS (Network File System), where each client connects to a central server to use files that are located there is an example of a centralised file sharing system.
Where the centralised systems rely quite directly on the client-server-model, the distributed systems often use a special superset, where each node acts as both a client and a server. This has effects on things like how you have to configure your firewall.

4. The Internet Relay Chat, network on a single server

a) Scalability and performance

The server would need a lot of network bandwidth and a lot of hardware resources. The only way to scale up would be to update the hardware or redesign the software. Performance would be likely to drop quite quickly as the number of users and channels grows. Also, in a single server setting, most of the world would be pretty far away from the server, so for distant users, it might have unacceptably long response times.

b) Availability and reliability

If the one server goes down, the whole network goes down. Since the server would be under a lot of stress, it would most likely to go down sooner than later. On the other hand, there would never be net splits; it would be all in or all out.

c) Openness, security and user acceptance

The service would be more open, in the sense it would be easier to make changes that would not be possible in a multi-server setting administered by different parties. This allows some more room to affect security too, the administrator could configure security features more freely, like enforce use of TLS. On the other hand a single server is more vulnerable to dos-attacks. With the big irc-networks already in place, I don't see a single random server gaining much popularity.

5. Eight common false assumptions

The network is reliable.

The network can be congested, misroute packets or simply go down. You lose data, if you fail to take this into account.

The network is secure.

A distributed system can store it's users' passwords in a very secure way, but it's of little use if the clients send their passwords through the internet in plain text. Also many kinds of DOS or penetration attacks can originate from the network, from any side.

The network is homogenous

The network can contain anything, from 10gbps ethernet on a server to 9.6kbps circuit-switched network on a mobile client. MTU:s might vary from network to network. Your datacenter jumbo-packets are probably not usable on the internet, and fragmentation would occur.

The topology does not change

If this assumption is hardcoded into your application, you might be in trouble. The system should usually be able to advertise outside that it has moved, or have some other means to be found.

Latency is zero

What works on a LAN might not work on the internet where latencies can be anything. If it takes too long to access a resource, she system will feel slow.

Bandwidth is infinite

A distributed system that accesses resources from many locations might perform poorly, since even though the resources are all available, they may not arrive as fast as expected. Also the resources might be too large to be transmitted at once. Importing pay check-information for a company with 30 000 personnel into the banks database in one big transaction might prevent other customers from using the service for a while.

Transport cost is zero

The bigger the system, the more it costs. If your service becomes very popular, you might need to scale it up, and that costs money. Also buying and managing network hardware is not cheap. Also, there is a computational cost to building, sending and processing packets, then again computations cost money too. I'm not quite sure which kind of a cost exactly this meant.

There is one administrator

The different administering parties might enforce conflicting policies. Some of your security decisions might be circumventable just by coming through a different party's point of entrance.