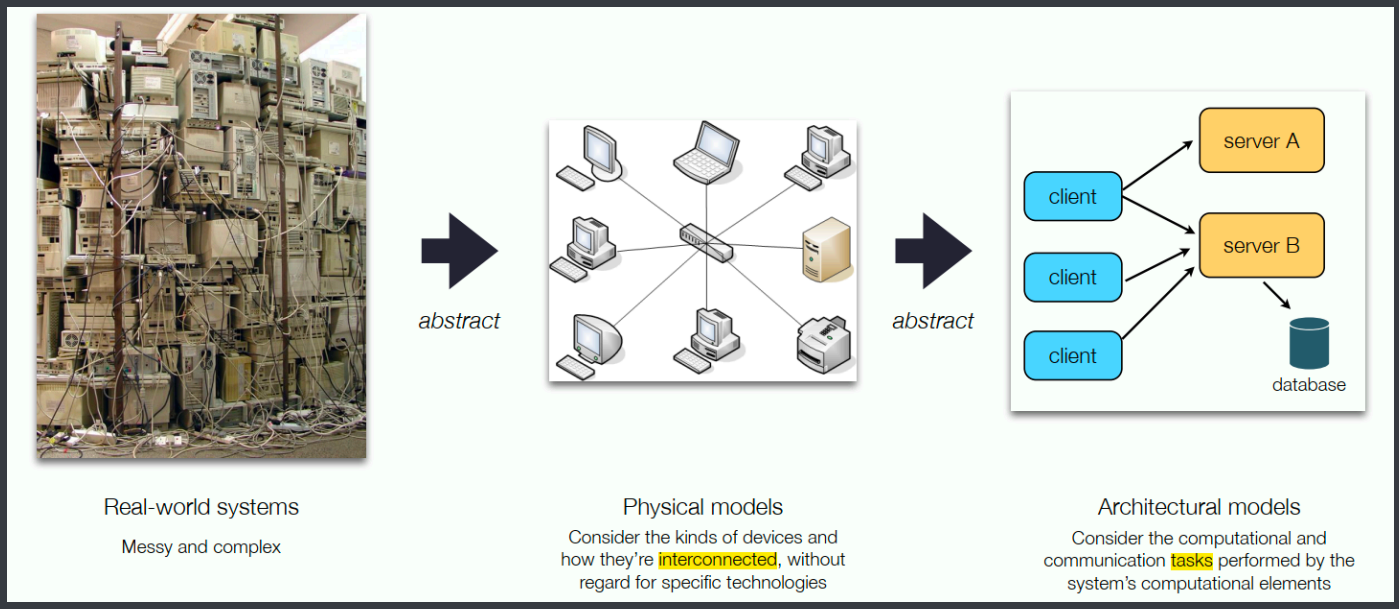
architectures



computational entities

phase	context (what are we programming)	entities
80's, 90's	system	nodes & processes
90's ->	application	objects & components
00's ->	composition & virtualization	services & containers

communication paradigm

1. inter-process communication

   OS network programming APIs, e.g. UNIX sockets
2. remote invocation

   remote procedure call / remote method invocation
3. indirect communication

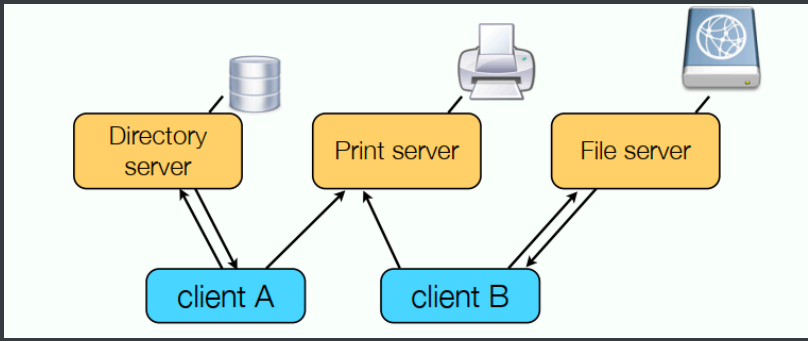
   by a middleman (a "broker")

roles & responsibilities

2 main paradigms: client-server / peer-to-peer

client-server

a server can be the client of another server



peer-to-peer

a **peer** acts both as a client and as a server,

roughly equivalent capabilities for each peer

decentralize: no single point of failure & better scaling and utilization

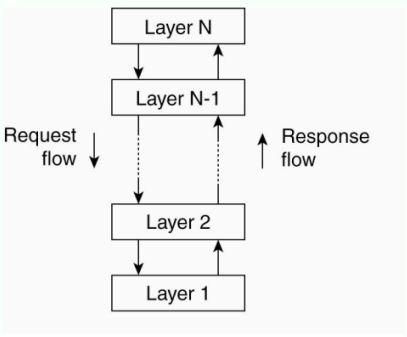
placement

no universal solution, typical strategies:

map services to multiple servers, caching, mobile code, mobile agents etc.

architectural styles

1. layered (message passing)



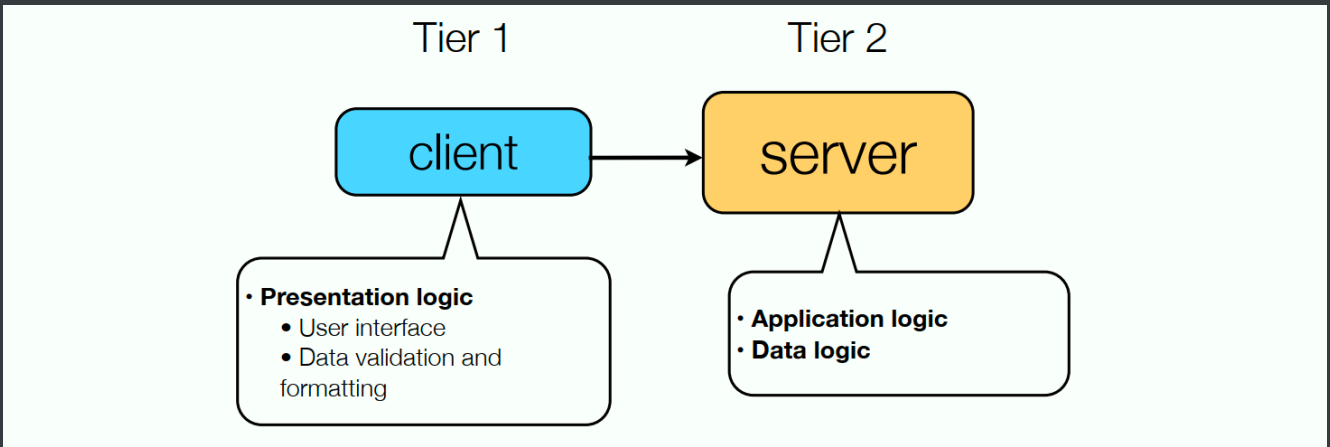
Layered (message passing)

middleware layer

end-to-end argument 端到端原则：

某些通信功能需要application的参与，因为只有在endpoint拥有完整信息时才能被正确实现；

2. two-tier architecture



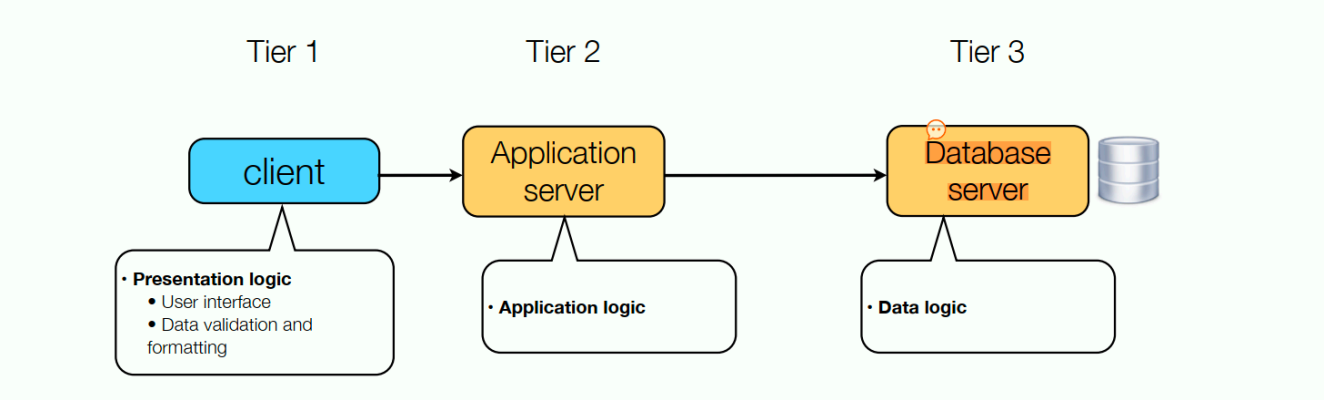
e.g. modern single-page web applications

client = webpage with embedded Javascript code

server = API provider & serves static web content

AJAX: Asynchronous JavaScript And XML (page updated without reloading)

3. three-tier architecture



4. multi-tier architecture

real-world distributed applications often multi-tier

5. proxies

between client & server,

can: cache, load balancer, filter, authentivation, hide servers ...

6. wrappers / adaptors
7. interceptors

a client stub (that offers the same interface as a remote object)

not a proxy!! (proxy is a server)