Rechnernetze Grundlage für VS, Webtechnologien Teilgebiet der VS

Warum?

Zuverlässigkeit, Verfügbarkeit, Redundanz

Nachteile:

Kosten (Geld, Latenz, Schwankung), Erhöhte Komplexität (Debugging, Timeouts)

Verteilungstransparenz -> System wirkt so als wäre alles an einem Ort

**Nebenläufigkeit:**

Quasiparallelität auf Einprozessorsystemen

Echte Parallelität auf Mehrkernprozessoren

Nebenläufigkeit in VS:  
Client: Asynchrones warten auf Serverantwort

Server: Parallele Abarbeitung von Anfragen

**Client:**

Synchron -> blockingReceive()

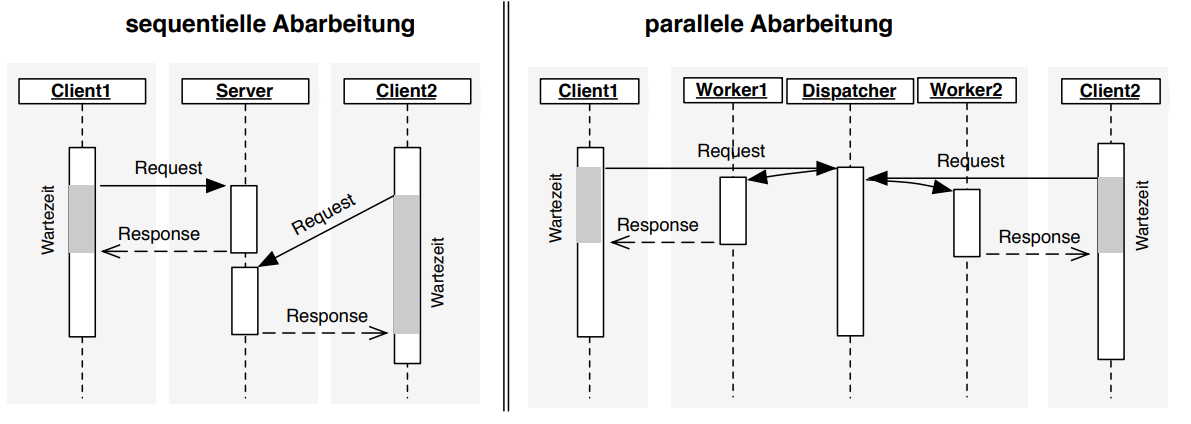
Asynchron, polling -> nonBlockingReceive() in Loop, break loop when getting response

Asynchron, nebenläufig -> Send Request, start Thread2, continue. Thread2 -> wait for response

**Server:**

Seriell: Wait for Request, answer it, wait for next Request

Parallel: Dispatcher waits for Request, give it to worker to answer it, wait for next Request



Threadpool:  
Statt pro request einen Thread zu starten und diesen anschließend zu verwerfen wird zu Beginn ein Pool erstellt. Alle Threads dieses Pools „bedienen“ sich aus einer Queue in der der Dispatcher die requests ablegt.

Dynamisch wachsender Threadpool:

Wie Threadpool, aber wenn z.B. die Queue zu lang wird werden neue Threads gestartet

POOL\_SIZE = Kerne / CPU-Intensität (der Tasks)

* Wenn die Bearbeitung der Requests zu 50% aus File-IO besteht können mehr Threads erstellt werden um die CPU voll auszulasten
* Beenden des Servers:

Flag -> while(!done) {request = blockingReceive()}

Nachteil: Wenn nach setzen der Flag keine Requests mehr kommen wartet der server ewig im blockingReceive()

Poison Pill:

Shutdown Aufforderung als Request

Nachteil (bösartiger) Client kann Poison Pill versehentlich (absichtlich) verschicken