**Database:**

Working size exceeds available RAM

Long & short running queries

Write-write conflicts

Large joins taking up memory

**Virtualisation**:

Sharing a HDD, disk seek death

Network I/O fluctuations in the cloud

**Programming**:

Threads: deadlocks, heavyweight as compared to events, debugging, non-linear scalability, etc...

Event driven programming: callback complexity, how-to-store-state-in-function-calls, etc...

Lack of profiling, lack of tracing, lack of logging

One piece can't scale, SPOF, non horizontally scalable, etc...

Stateful apps

Bad design : The developers create an app which runs fine on their computer. The app goes into production, and runs fine, with a couple of users. Months/Years later, the application can't run with thousands of users and needs to be totally re-architectured and rewritten.

Algorithm complexity

Dependent services like DNS lookups and whatever else you may block on.

Stack space

**Disk**:

Local disk access

Random disk I/O -> disk seeks

Disk fragmentation

SSDs performance drop once  data written is greater than SSD size

**OS:**

Fsync flushing, linux buffer cache filling up

TCP buffers too small

File descriptor limits

Power budget

**Caching:**

Not using memcached (database pummeling)

In HTTP: headers, etags, not gzipping, etc..

Not utilising the browser's cache enough

Byte code caches (e.g. PHP)

L1/L2 caches. This is a huge bottleneck. Keep important hot/data in L1/L2. This spans so much: snappy for network I/O, column DBs run algorithms directly on compressed data, etc. Then there are techniques to not destroy your TLB. The most important idea is to have a firm grasp on computer architecture in terms of CPUs multi-core, L1/L2, shared L3, NUMA RAM, data transfer bandwidth/latency from DRAM to chip, DRAM caches DiskPages, DirtyPages, TCP packets travel thru CPU<->DRAM<->NIC.

**CPU:**

CPU overload

Context switches -> too many threads on a core, bad luck w/ the linux scheduler, too many system calls, etc...

IO waits -> all CPUs wait at the same speed

CPU Caches: Caching data is a fine grained process (In Java think volatile for instance), in order to find the right balance between having multiple instances with different values for data and heavy synchronization to keep the cached data consistent.

Backplane throughput

**Network:**

NIC maxed out, IRQ saturation, soft interrupts taking up 100% CPU

DNS lookups

Dropped packets

Unexpected routes with in the network

Network disk access

Shared SANs

Server failure -> no answer anymore from the server

**Process:**

Testing time

Development time

Team size

Budget

Code debt

**Memory:**

Out of memory -> kills process, go into swap & grind to a halt

Out of memory causing Disk Thrashing (related to swap)

Memory library overhead

Memory fragmentation

In Java requires GC pauses

In C, malloc's start taking forever