There is a distinction between data-intensive and compute-intensive applications

- Raw CPU power is rarely the limiting factor -- the bigger problem is the amount of data, the complexity of data, and the speed at which it changes

A data intseive application is typically built from standard building blocks

- store data so you can find it again - database
- remember the result of an expensive operation to speed up reads - cache
- allow users to search data by keyword/filter - search indexes
- send a message to another process to be handled asyncly - stream processiong
- periodically crunch a large amount of accumulated data - batch processing

There are various approaches to each of these steps, and evaluating which one to pick for your problem is a challenge.

- For example, although a database and a message queue both store data for some time, they have very different access patterns, which means very different performance characteristics, and thus very different implementations

New technologies blur the lines between these tools

- Datastores that are message queues - Redis
- Message queues with database-like durability guarantees - Apache Kafka

Three concerns of prime importance
1. **Reliability:** the system should work correctly (perform the correct function at the desired level of performance) even in the face of adversity (hardware or software faults, even human error)
2. **Scalability:** As the system grows (in data volume, traffic volume, or complexity) there should be reasonable ways of dealing with that growth
3. **Maintainability:** Manye different people should be able to work on the system productively

Reliability
- 'prevent faults from causing failures'
- faults are things that can go wrong -- a failure is when the entire system comes down
- generally preferable to tolerate faults than to prevent faults, but for some kinds of faults (security) there is no cure, so better prevent!

Hardware faults

- hard disks have Mean Time To Failure MTTF of about 10-50 years
- typically address with redundancy
  - RAID configuration
  - dual power supplies to servers and hot-swappabale CPUs

    - datacentres have generators
- however, as data volumes have increased and applications' computing demands have correspondingly increased, systems are using more machines, which increases the number of hardware faults
- so there is a move towards software fault-tolerance techniques
    - rolling upgrades?

Software faults
- process isolation
- careful design about assumptions and interactions
- alerting

Scalability
Describing load
- load parameters capture and describe the nature of the load on a system
- these params depend on the architecture of the system
    - requests per second to a web server
    - ratio of reads to writes to a db
    - number of simultaneous active users in a chat room
    - hit rate on a cache

Describing performance
- two ways to investigate performance
    - when you increase a load parameter and keep the system resources -- CPU, memory, network bandwidth, etc -- the same, how is the performance of the system affected?
    - when you increase a load parameter, how much do you need to increase system resources to keep performance unchanged?
- in a batch processing system like Hadoop, you care about throughput
    - the number of records processed per second, or the total time taken to run a job on a dataset of a particular size
- in online systems, typically care more about response time
    - the time between a client sending a request and receiving a response
    - response time is what the client sees = service time + network delays + queueing delays
    - latency is the duration that a request is waiting to be handled
- There is likely to always be variation in the response time for a given packet
    - random additional latency can stem from context switching to a background process, the loss of a network packet and TCP retransmission, garbage collection, a page fault forcing a disk read, mechanical vibrations in the server rack...

            * queueing delays: a server can only process as many requests at the same time as it has CPU cores -- so it takes only a small number of slow requests to hold up processing of subsequent requests

- percentile metrics
    - hdrHistogram, forward decay -

Coping with load

- scaling out/horizontal scaling: distributing load across multiple smaller machines
- scaling up/vertical scaling: moving to a more powerful machine
- distributing load across multiple machines = shared nothing architecture