Extreme Computing Revision
Emphasis on solving problems

Design a system to store YouTube videos.
You’re Twitter. Make a low-latency feed system.
Show the most frequently visited pages. Live.
Reason About a Problem

- What’s the bottleneck? Compute, storage, bandwidth, latency, ...
- Which data is big? Which is small? → Join strategies
- Is it balanced?
- What if that machine fails?
Reason About a Problem

- What’s the bottleneck? Compute, storage, bandwidth, latency, ...?
- Which data is big? Which is small? → Join strategies
- Is it balanced?
- What if that machine fails?

More fun: how will this system fail?
Bag of Tricks

- Sharding aka partitioning: divide the work across machines
- Replication for speed and fault tolerance
- “Cold” large read-only store, “hot” small mutable store
- Approximations: Bloom filters, streaming counts, reservoir sampling
MapReduce  Parallel batch processing

BitTorrent  File sharing

HDFS  File storage

Chord  Divide responsibility as machines join and leave

SSTable  Large read-only key-value store
BigTable  Large mutable key-value store
Systems

MapReduce Parallel batch processing

BitTorrent File sharing

HDFS File storage

Chord Divide responsibility as machines join and leave

SSTable Large read-only key-value store
BigTable Large mutable key-value store

Given a problem, name a system and apply it.
Take inspiration from the design of these systems.
Linearisable versus Sequential

Alice and Bob write checks to each other for the same amount.

<table>
<thead>
<tr>
<th>Alice’s Statement</th>
<th>Bob’s Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10 Check Alice → Bob</td>
<td>-10 Check Bob → Alice</td>
</tr>
<tr>
<td>0 Check Bob → Alice</td>
<td>0 Check Alice → Bob</td>
</tr>
</tbody>
</table>

Both overdraft.

✅ Sequential: each client sees a consistent order
❌ Linearizable: no globally linear story