• Distributed transactions
• Multi-site atomicity
• Two-phase commit
**goal:** build reliable systems from unreliable components

the abstraction that makes that easier is

**transactions**, which provide **atomicity** and **isolation**, while not hindering **performance**

**atomicity** → **shadow copies** (simple, poor performance) or **logs** (better performance, a bit more complex)

**isolation** → **two-phase locking**

eventually, we also want transaction-based systems to be **distributed**: to run across multiple machines
problem: one server committed, the other did not
goal: develop a protocol that can provide multi-site atomicity in the face of all sorts of failures (message loss, message reordering, worker failure, coordinator failure)

message failures solved with reliable transport protocol (sequence numbers + ACKs)
two-phase commit: nodes agree that they’re ready to commit before committing.
failure: lost prepare
client

coordinator

A-M server

N-Z server

failure: lost ACK for prepare
client

---

coordinator

---

A-M server

---

N-Z server

---

**failure:** worker failure during prepare
failure: lost commit message
failure: lost ACK for commit message
failure: worker failure during commit
if workers fail after the commit point, we cannot abort the transaction. workers must be able to recover into a prepared state

workers write \texttt{PREPARE} records once prepared. the recovery process — reading through the log — will indicate which transactions are prepared but not committed
failure: worker failure during commit
failure: worker failure during commit
failure: coordinator failure during prepare
failure: coordinator failure during prepare
failure: coordinator failure during commit
failure: coordinator failure during commit
problem: in our example, when workers fail, some of the data (e.g., accounts A-M) is completely unavailable
solution: replicate data

but! how will we keep multiple copies of the data consistent? what type of consistency do we want?
• **Two-phase commit** allows us to achieve **multi-site atomicity**: transactions remain atomic even when they require communication with multiple machines.

• In two-phase commit, failures prior to the commit point can be aborted. If workers (or the coordinator) fail after the commit point, they **recover into the prepared state**, and complete the transaction.

• Our remaining issue deals with availability and replication: we will replicate data across sites to improve availability, but must deal with keeping multiple copies of the data **consistent**.