2-phase commit is good if different nodes are doing different things but in general you have to wait for all sites and TC to be up you have to know if each site voted yes or no and the TC must be up to decide not very fault-tolerant: has to wait for repair can we get work done even if some nodes can't be contacted? yes: in the special case of replication

state machine replication works for any kind of replicated service: storage or lock server or whatever every replica must see same operations in same order if deterministic, replicas will end up with same state

how to ensure all replicas see operations in the same order? primary + backup(s) clients send all operations to current primary primary chooses order, sends to backups, replies to client

what if the primary fails? need to worry about that last operation, possibly not complete need to pick a new primary can't afford to have two primaries! suppose lowest-numbered live server is the primary so after failure, everyone pings everyone then everyone knows who new primary is? well, maybe not: pings may be lost => two primaries pings may be delayed => two primaries partition => two primaries

idea: a majority of nodes must agree on the primary at most one network partition can have a majority if two potential primaries, their majorities must overlap

technique: "view change" algorithm system goes through a sequence of views view: view# and set of participants ensure agreement on unique successor of each view the participant set allows everyone to agree on new primary

view change requires "fault-tolerant agreement" at most a single value is chosen agree despite lost messages and crashed nodes can't really guarantee to agree but we can guarantee to *not* "agree" on different values!

Paxos fault-tolerant agreement protocol eventually succeeds if a majority of participants are reachable best known algorithm
one (or more) nodes decide to be the leader
leader chooses a proposed value to agree on
(view# and participant set)
leader contacts participants, tries to assemble a majority
participants are all the nodes in the old view (including unreachable)
or a fixed set of configuration master nodes
if a majority respond, we're done

why agreement is hard
what if two nodes decide to be the leader?
what if network partition leads to two leaders?
what if the leader crashes after persuading only some of the nodes?
what if leader got a majority, then failed, without announcing result?
or announced result to only a few nodes?
new leader might choose a different value, even though we agreed

Paxos
has three phases
may have to start over if failure/timeouts

state (per view)
\( n_a, v_a \): highest value and \( n \) which node has accepted
\( n_h \): highest \( n \) seen in a Q1
done: leader says agreement was reached, we can start new view

Paxos Phase 1
a node (maybe more than one...) decides to be leader
picks a proposal number \( n \)
must be unique, good if it's higher than any known #
how about last known proposal number, plus one, append node ID
sends Q1(n) to every node (including itself)
if node gets Q1(n) and \( n > n_h \):
\( n_h = n \)
return R1(n_a, v_a)

Paxos Phase 2
if leader gets R1 from majority of nodes (including self):
  if any R1(n,v) had a value, \( v = \) value of highest \( n \)
  else leader gets to choose a value
    old view# + 1, set of pingable nodes
    send Q2(n, v) to all responders
  if node gets Q2(n, v) and \( n >= n_h \)
    \( n_a = n \)
    \( v_a = v \)
  return R2()

Paxos Phase 3
if leader gets a majority of R2():
  send Q3() to all
if node gets Q3():
  done = true
  primary is lowest-numbered node in \( v_a \)

if at any time any node gets bored (times out)
it declares itself a leader and starts a new Phase 1
if nothing goes wrong, Paxos clearly reaches agreement

how do we ensure good probability that there is only one leader?
   every node has to be prepared to be leader, to cope w/ failure
   so delay a random amount of time after you realize a new view is
   required
   or delay your ID times some constant

key danger:
   nodes w/ different v_a receive Q3
   goal: if Q3 *could* have been sent, future Q3s guaranteed to have
   same v_a

what if more than one leader?
   due to timeout or partition or lost packets
   the two leaders used different n, say 10 and 11
   if 10 didn't get a majority to R2
      it never will, since no-one will R2 10 after seeing 11's Q1
      or perhaps 10 is in a network partition
   if 10 did get a majority to R2
      i.e. might have sent Q3
         10's majority saw 10's Q2 before 11's Q1
         otherwise they would have ignored 10's Q2, so no majority
         so 11 will get a R1 from at least one node that saw 10's Q2
         so 11 will be aware of 10's value
         so 11 will use 10's value, rather than making a new one
         so we agreed on a v after all

what if leader fails before sending Q2s?
   some node will time out and become a leader
   old leader didn't send any Q3, so we don't care what he did
   it's good, but not necessary, that new leader chooses higher n
   if it doesn't, timeout and some other leader will try
   eventually we'll get a leader that knew old n and will use a higher
   n

what if leader fails after sending a minority of Q2s?
   same as two leaders...

what if leader fails after sending a majority of Q2s?
   i.e. potentially after reaching agreement!
   same as two leaders...

what if a node fails after receiving Q2?
   if it doesn't restart, possible timeout in Phase 3, new leader
   if it does restart, it must remember v_a/n_a! (on disk)
   leader might have failed after sending a few Q3s
   new leader must choose same value
   our node might be the intersecting node of the two majorities

what if a node reboots after sending R1?
   does it have to remember n_h on disk?
   it uses n_h to reject Q1/Q2 with smaller n
   scenario:
      leader1 sends Q1(n=10), a bare majority sends R1
      so node X's n_h = 10
leader2 sends Q1(n=11), a majority intersecting only at node X
sends R1
    node X's n_h = 11
    leader2 got no R1 with a value, so it chooses v=200
node X crashes and reboots, loses n_h
leader1 sends Q2(n=10, v=100), its bare majority gets it
    including node X (which should have rejected it...)
    so we have agreement w/ v=100
leader2 sends Q2(n=11, v=200)
    its bare majority all accept the message
    including node X, since 11 > n_h
    so we have agreement w/ v=200. oops.
so: each node must remember n_h on disk

conclusion
what have we achieved?
remember the original goal was replicated state machines
    and we want to continue even if some nodes are not available
after each failure we can perform view change using Paxos agreement
that is, we can agree on exactly which nodes are in the new view
so, for example, everyone can agree on a single new primary
but we haven't talked at all about how to manage the data