Distributed Computing and Storage Systems

From better theory to better practice

Gregory Chockler and Dahlia Malkhi

Hebrew University

SAN potential

- Cost-effective bandwidth scaling achieved by allowing the data to be transferred directly from network attached disks to clients
- Supporting heterogeneous disk controllers within a single network
- Centralized data management and truly incremental growth

Realizing SAN potential

- Advanced application services
  - Object store, file systems, virtualization layers, etc.
- Imposes a lot of challenges to research community
  - Concurrency control, security, availability and reliability, etc…

SAN and Distributed Computing

- SAN is a distributed system
- The models and methods developed for distributed computing apply
- The goal: Use theory to gain insights into power and limitations of SAN based systems

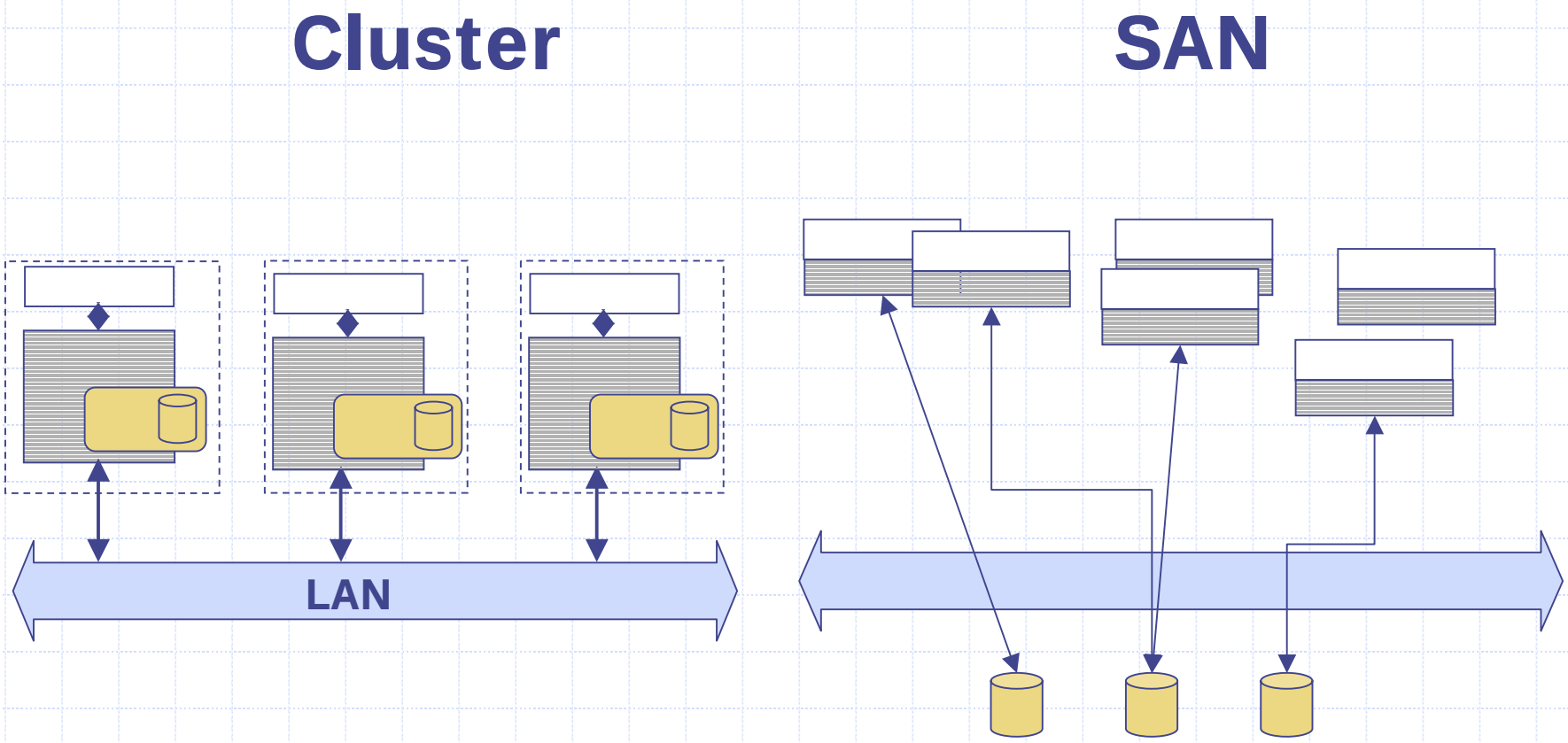
The problems considered

- Reliability/availability/fault-tolerance
- Scalability with respect to the number of clients served by the SAN
- Mutual exclusion and leader election

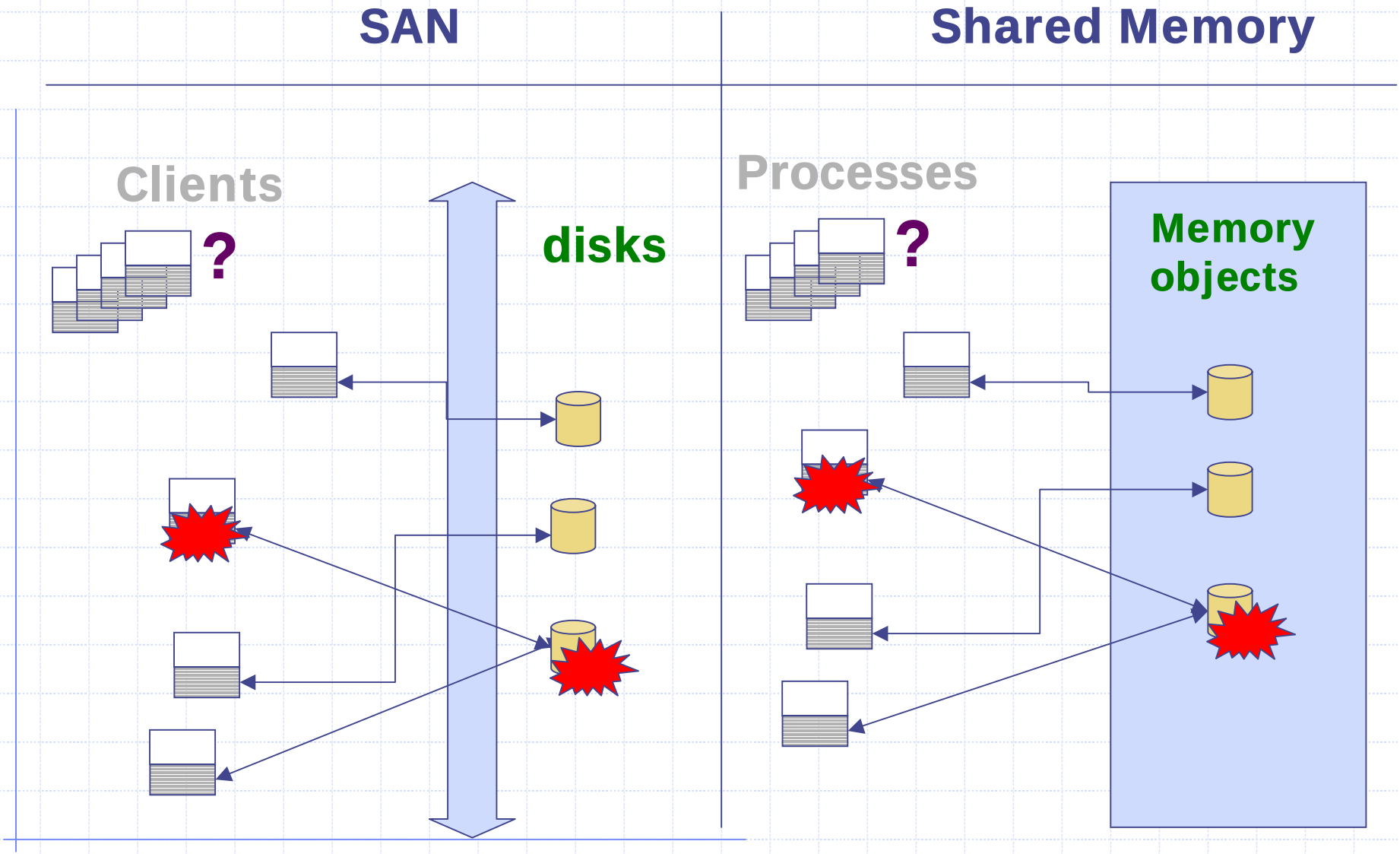
Fault-tolerance in SAN

- Disk is a separate entity which can fail independently from hosts
  - Disks become 1st class citizens not just a non-volatile extension of the main memory
- Communication is asymmetric: clients to disks
  - Not disk-to-disk
  - Client-to-client may be problematic
    - No broadcast support, limited scalability, need to deploy a separate cluster

Cluster vs. SAN



Modeling SAN

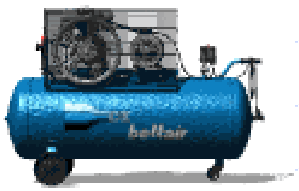


Modeling SAN

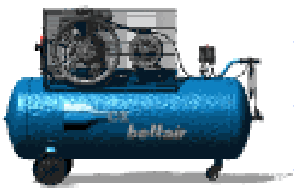
- SAN is best modeled as an asynchronous shared memory with
  - Process faults
  - Non-responsive memory faults
  - Participation: Known or unknown

Consensus

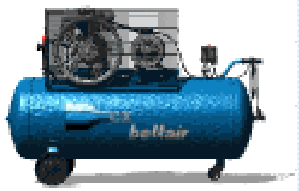
- Each participant starts from a value
- Outcome: common decision value
- Core technique for supporting fault-tolerance
  - Enables state-machine replication



a
b



a



a
b
c

Consensus solvability

- [JCT98]: Consensus is unsolvable with process and memory faults
  - Even for two processes
  - Regardless of the memory object types

- Consensus is unsolvable with faulty disks and faulty clients regardless of the disk functionality

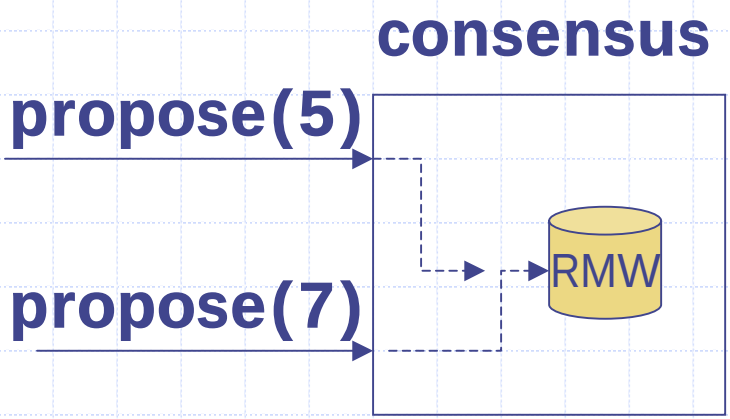
Solving Consensus

- Approach: compromise termination but never decide different values
- Termination is ensured once everything stabilizes (timeliness, no failures, etc…)
  - Can elect a consistent leader

Solving Consensus

- The question: what should the disk functionality be to enable Consensus?
  - What should the shared memory object satisfy to enable Consensus?
- Can it be made independent of actual participants?
  - Both their numbers and identities

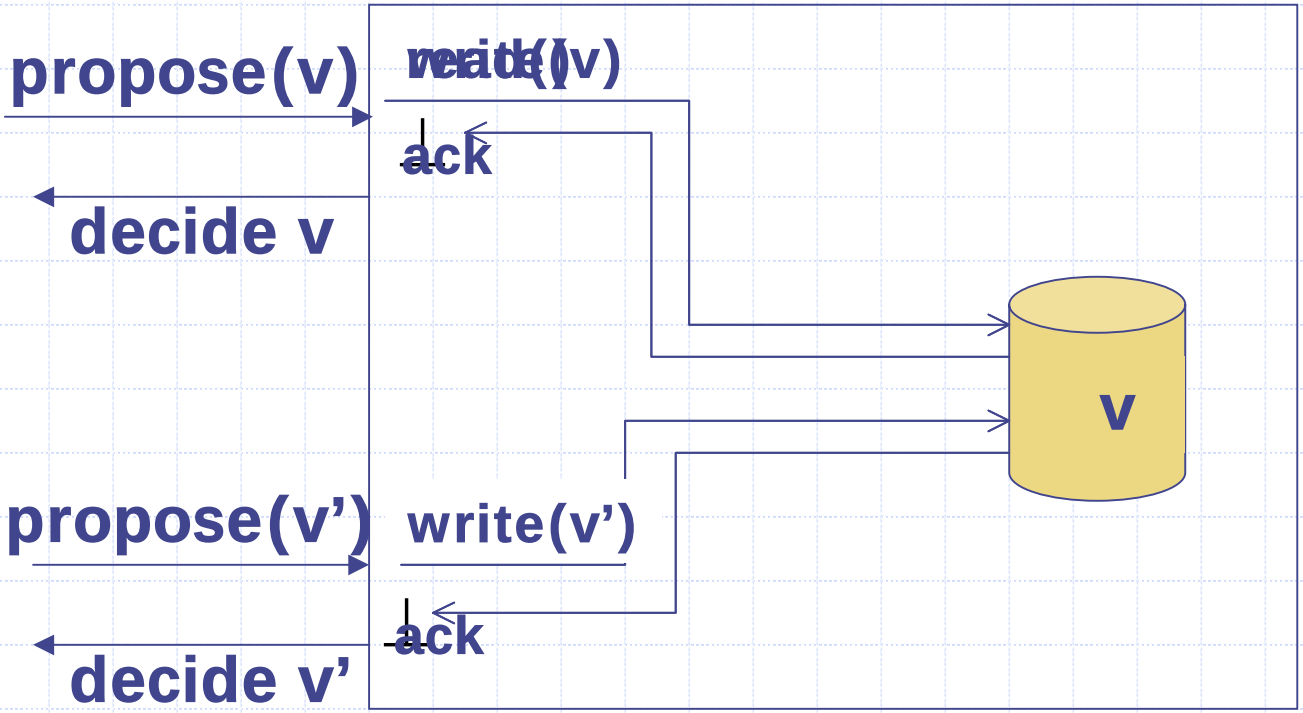
Identifying disk functionality



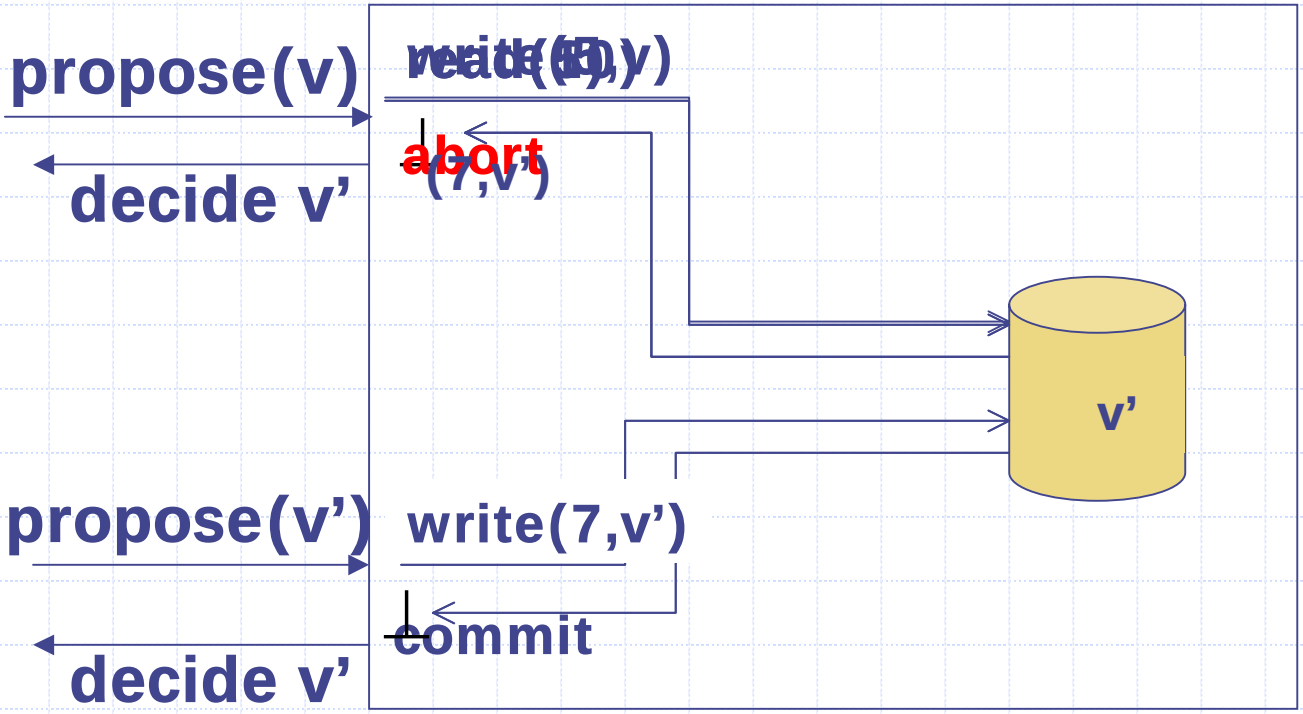

```
Propose(V):
  Begin RMW:
      if (val= ⊥ )
            val:=V;
      return val;
  End RMW
```

- Agreement is trivial with a single RMW
- RMW cannot be emulated out of faulty memory objects of any type [JCT98]

Decoupling Read and Write



Adding Ranks



Ranked Register

- If WRITE(R1) commits, then READ(R2), R2>R1 must “see” it
- Implementation:
  - Tolerates up to minority disk failures
  - Any number of process failures
- Trades participation obliviousness for the disk controller complexity
  - R/W registers: known participation
  - RMW registers: oblivious to participation

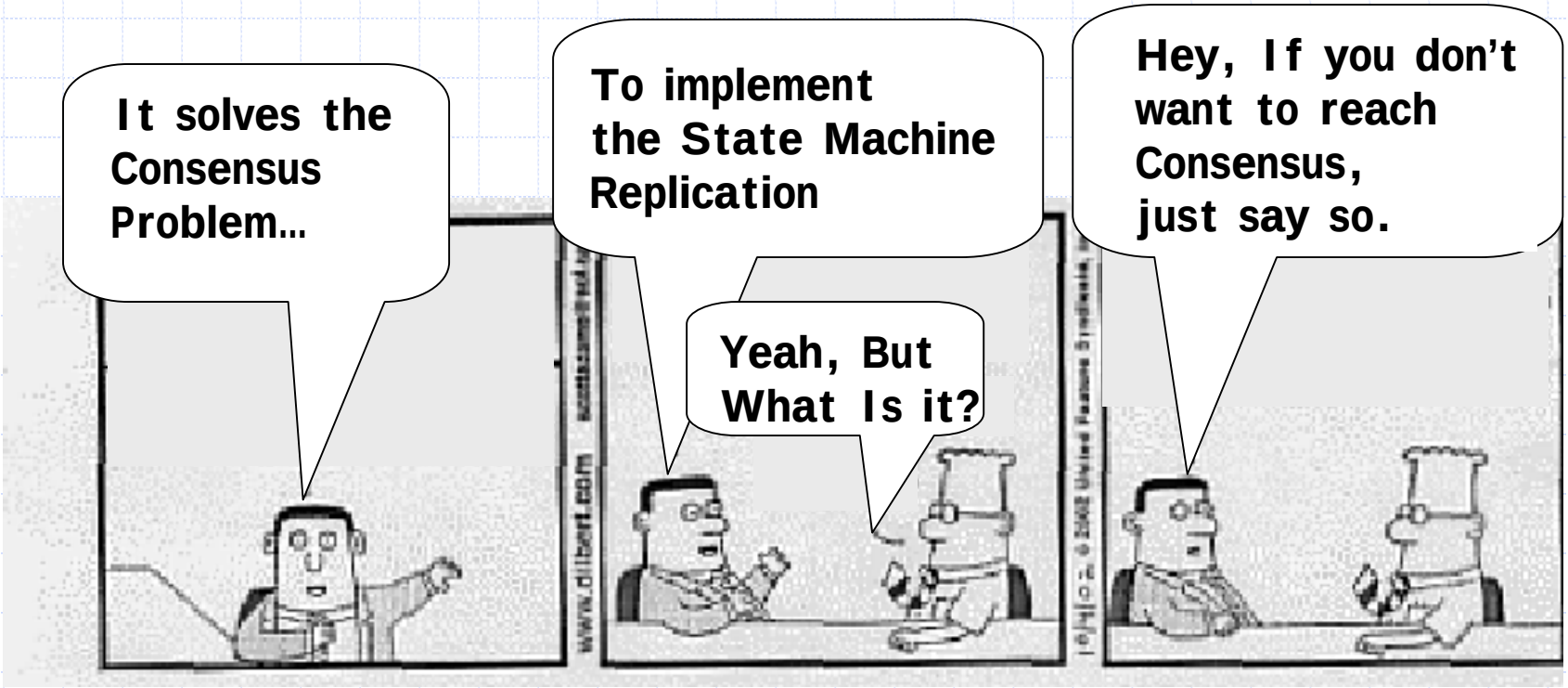
Leader election

- Leader election is the liveness side of the "Consensus coin"
- Assume everything is synchronous but participation is unknown
- Probabilistic LE protocol estimates the number of participants on-the-fly
  - Exponential backoff

Future work

- Assume the disks can lie. Can we be a “software RAID”?
  - Preliminary results: Yes, we can but the cost is prohibitively high
- What is the minimal requirements from the leader election module?
  - Can we come up with a (efficient) deterministic solution?

To conclude


It solves the Consensus Problem…
To implement the State Machine Replication
Yeah, But What Is it?
Hey, If you don't want to reach Consensus, just say so.