

How to make routing protocols tolerant of Byzantine failures

Robbert van Renesse (Cornell)

joint work with

Chi Ho (Cornell)

Mark Bickford (ATC-NY)

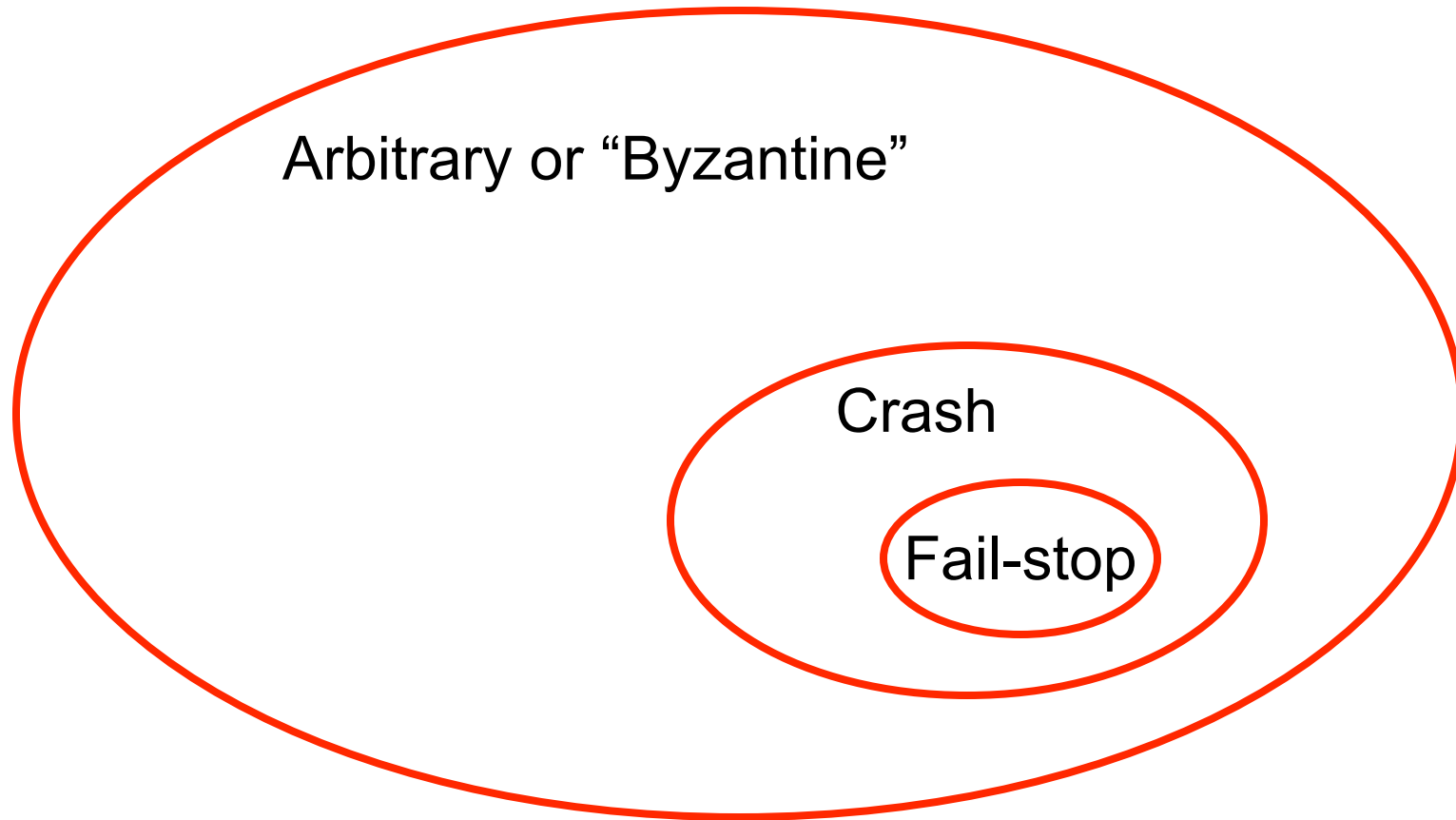
Danny Dolev (HUJI)



Goal: *Masking host failures in large distributed systems*

- Requires *Redundancy* and *Diversity*
 - Deterministic or “logical” failures can’t be masked (Need debugging, safe languages, access control, ...)
- Types of non-deterministic failures:
 - Heisenbugs, bit errors, misconfiguration
 - Malicious behavior
- Crash failures are relatively rare (< 20%)

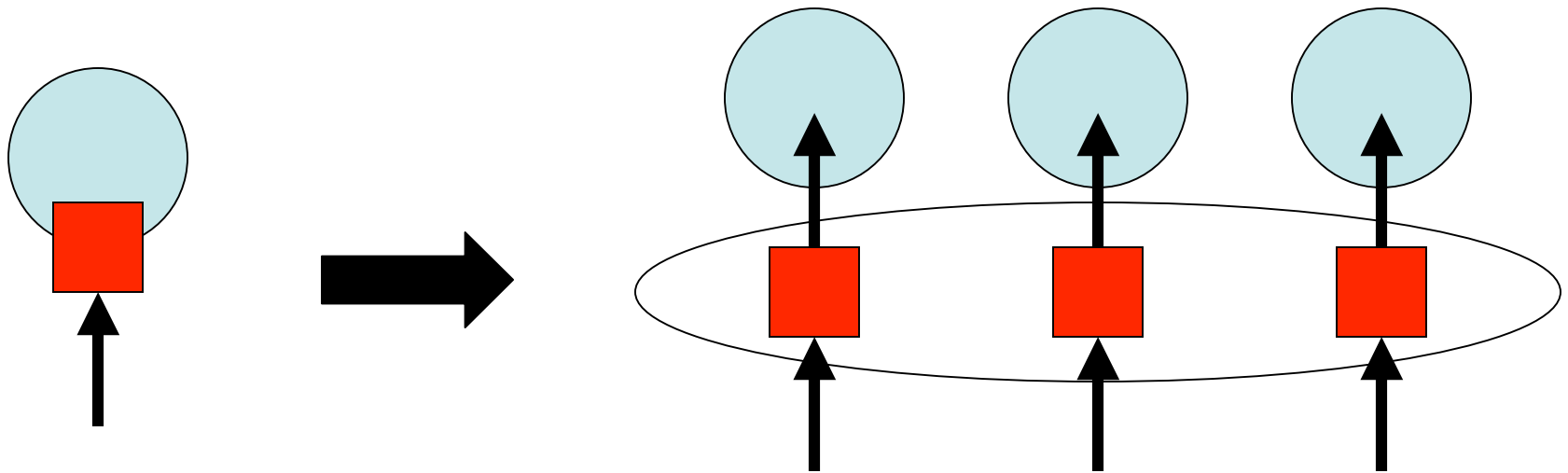
Assumptions about failures in distributed systems



State of the Art

- We know how to deal with crashes pretty well
 - Detect failures and reconfigure
- We understand how to build storage services that tolerate arbitrary failures efficiently
 - PBFT [CL99], Q/U [AGG+05], Zyzzyva [KAD+07], etc.
- *But how can systems like DNS, BGP, etc. mask arbitrary failures?*

Replicated State Machine Approach (Lamport'78)

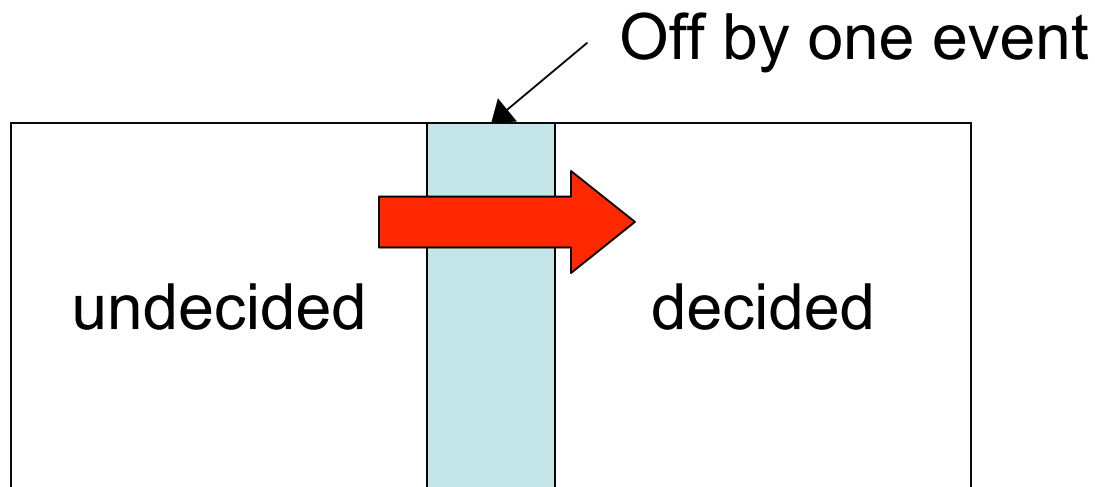


Input: Requires solving Consensus: Agreement + Order
Output: simple voting is sufficient

Solving consensus is hard...

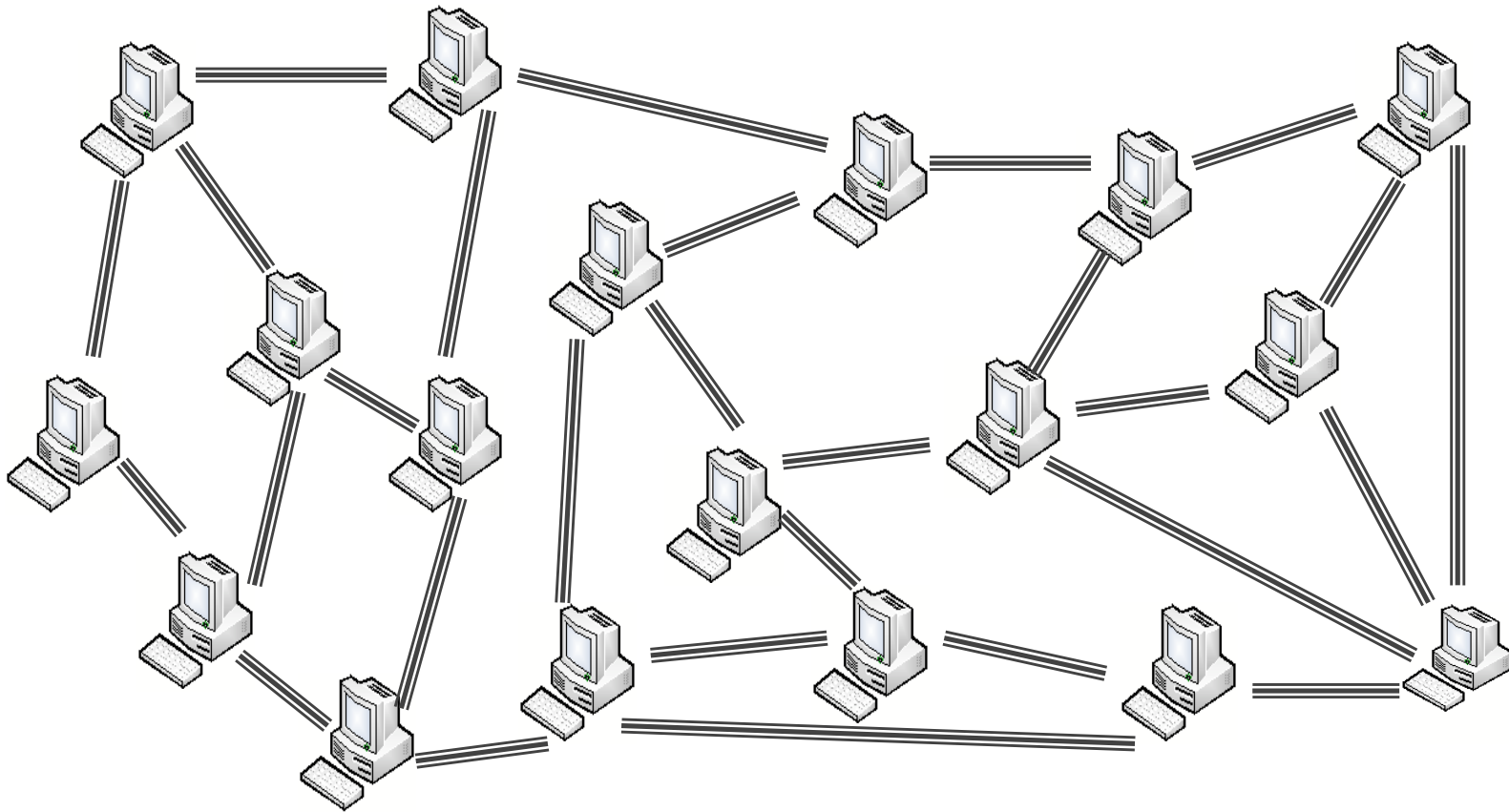


Crash failures + no assumptions about timing \Rightarrow *solving consensus is impossible* (FLP'83, FLP'85)

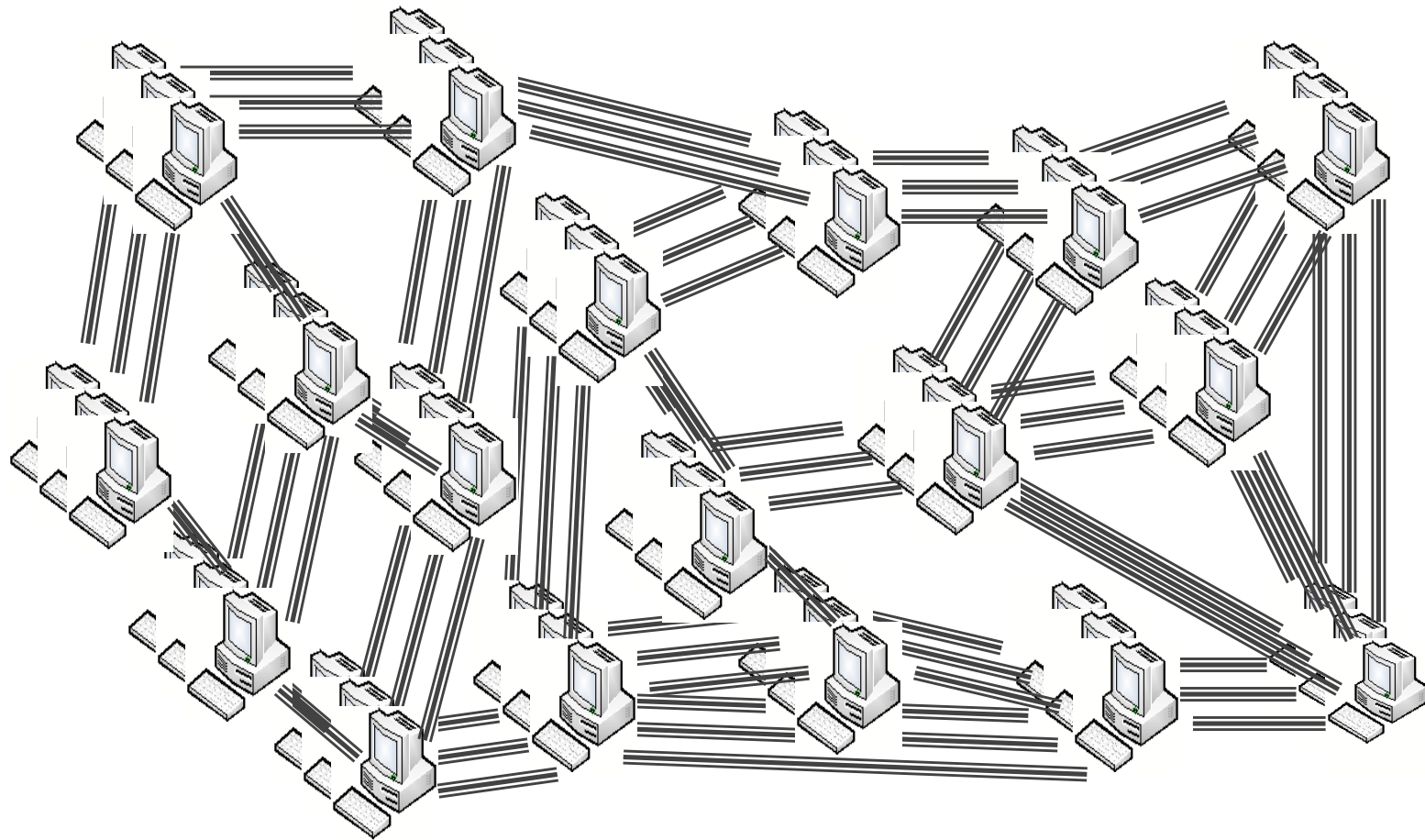


...and expensive

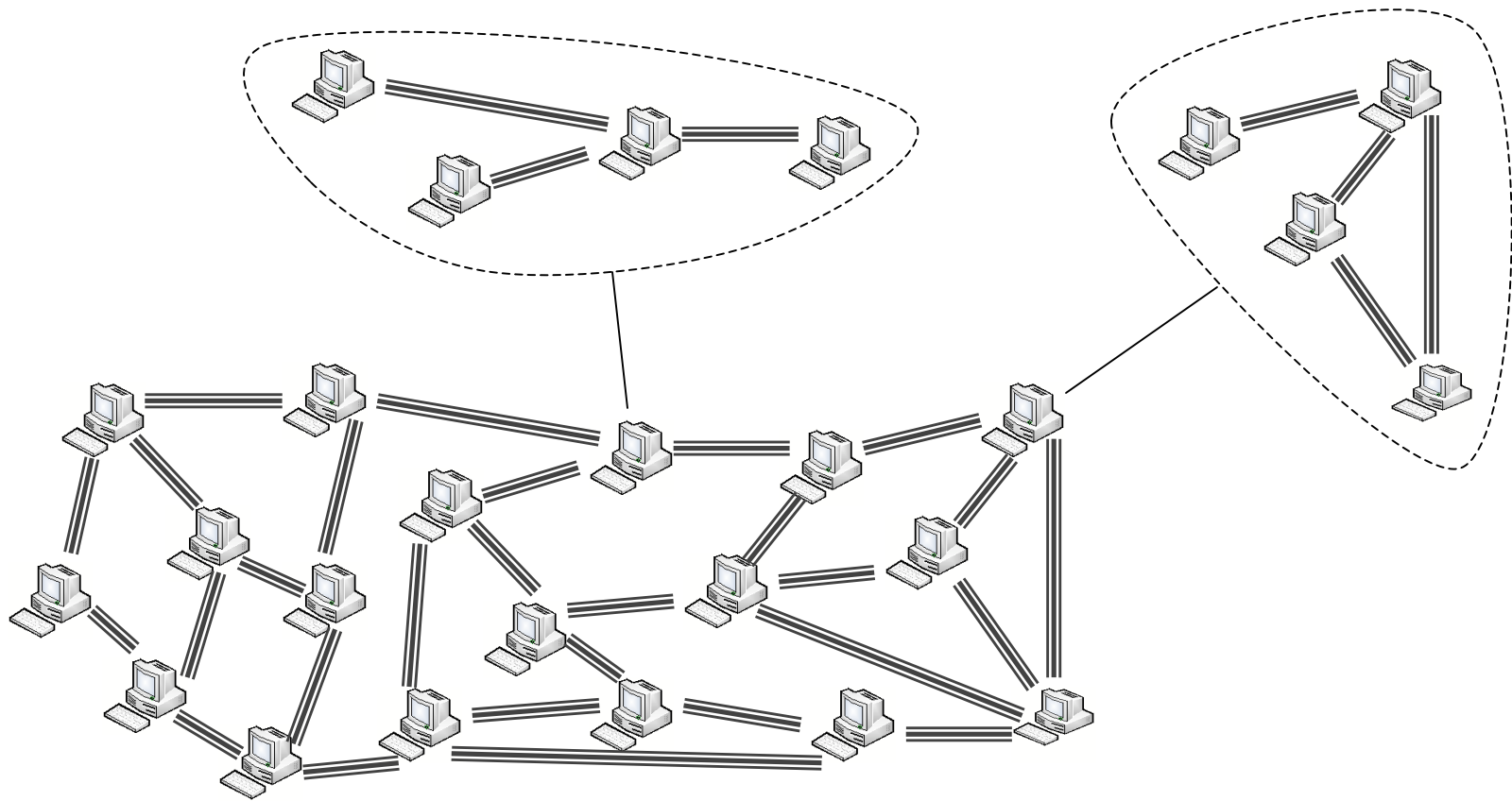
How to apply to a network?



Here's one way



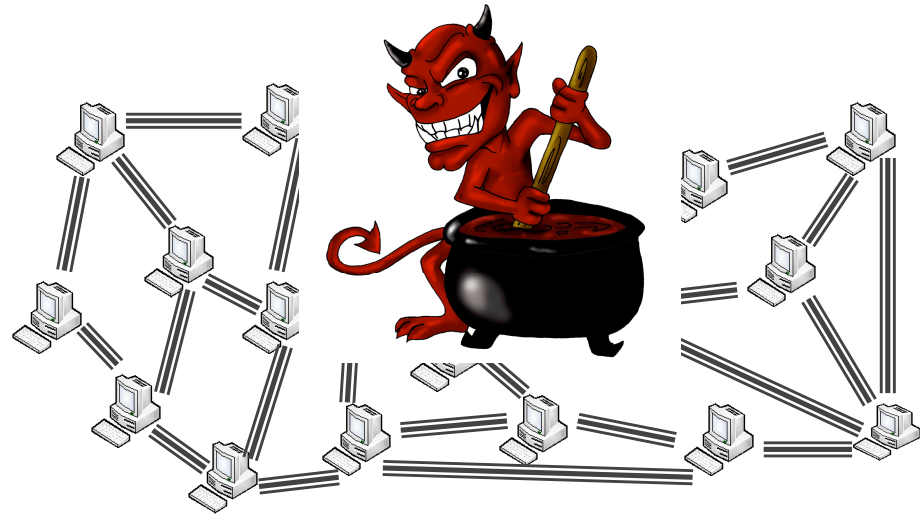
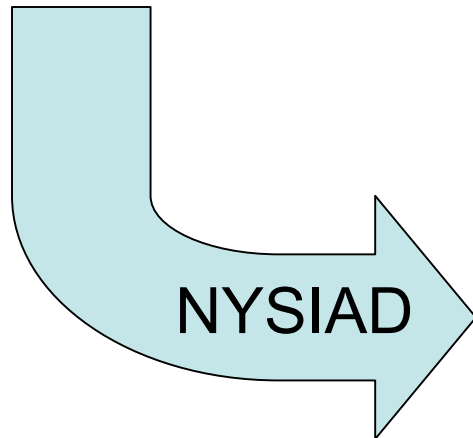
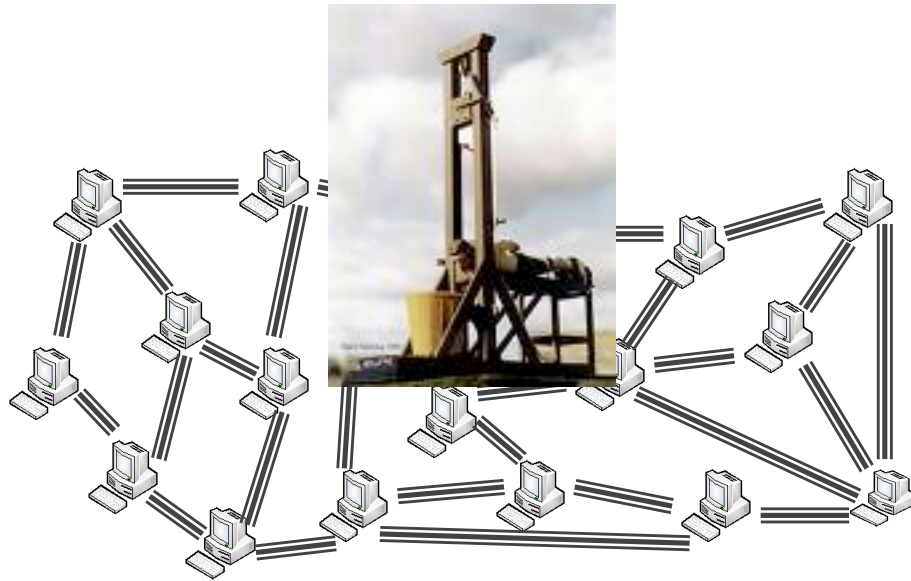
Here's a cheaper way:



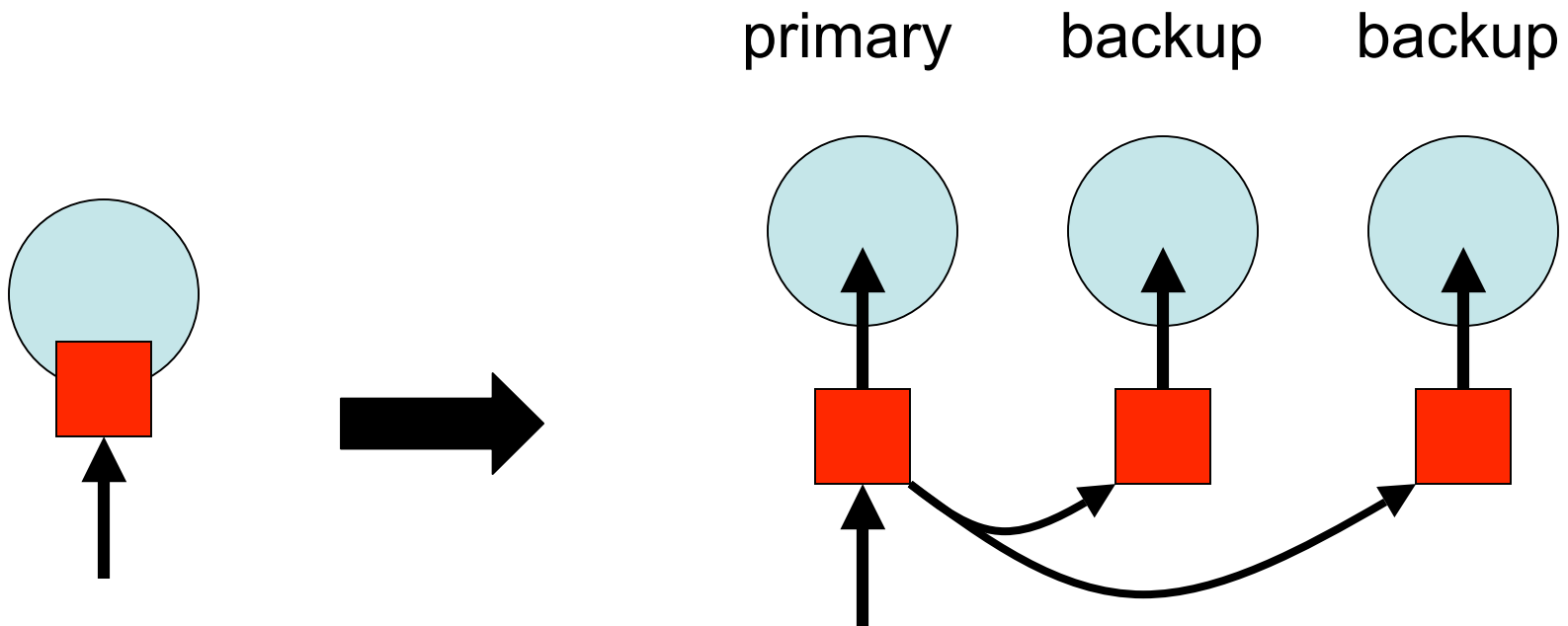
Both are overkill

- RSM transforms the network from one in which arbitrary failures might happen into one in which no failures happen
- But the network was perfectly capable of dealing with crash failures!

Our approach



Primary-Backup Replication



Input: FIFO broadcast from primary to backups

Output: voting is sufficient

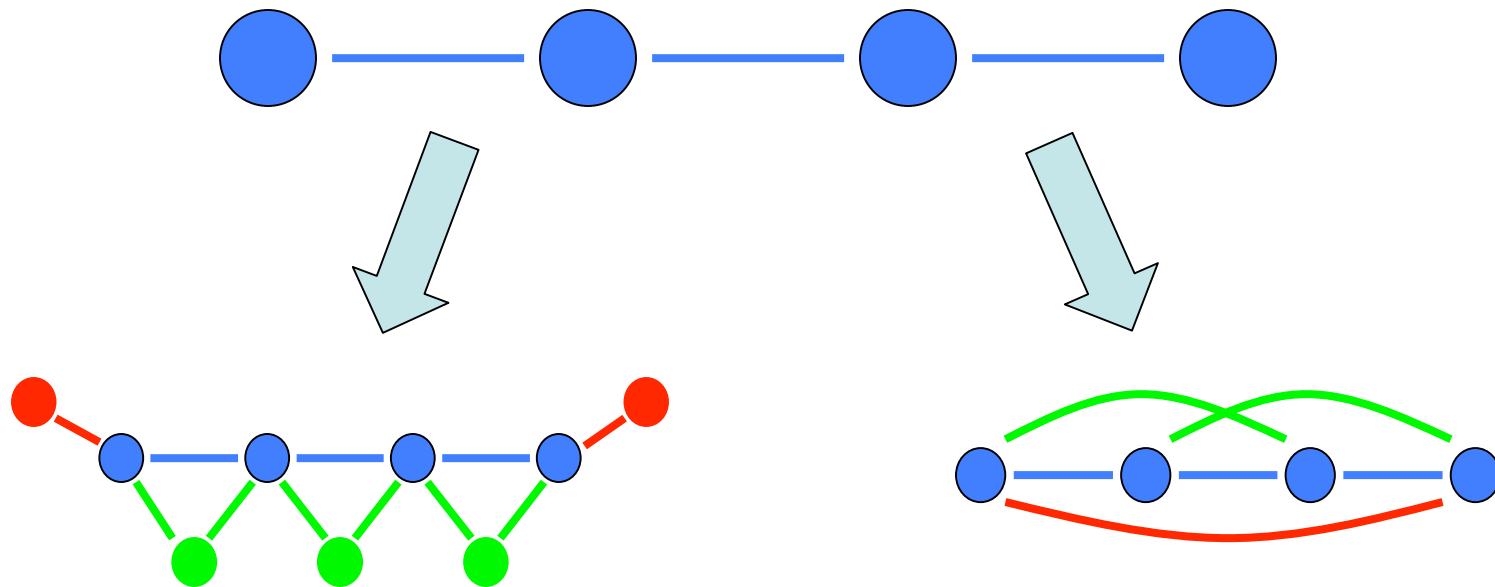
If primary fails, no re-election!

Guards (backups)

- Assign to each host a collection of *guards*
- The guards are responsible for
 - Tracking state of the primary host
 - Attesting the validity of primary's output
 - Making sure primary processes all its input, or none of it
- Not responsible for masking crash failure

Communication Graph Requirements

- Each (original) host needs at least $3t+1$ **guards**
 - t is a bound on the number of faults in a neighborhood
- Each two “**neighbors**” need at least $2t+1$ common “**monitors**”

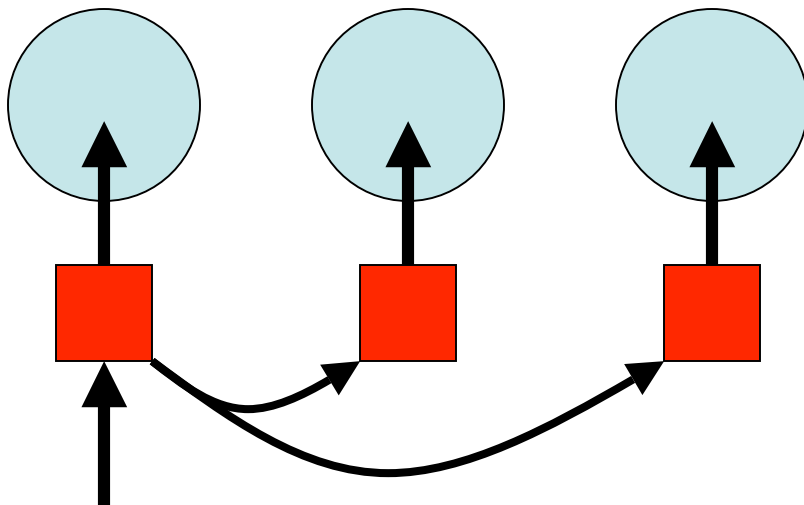


$t = 1$

Tracking state with OARCAST

Ordered Authenticated Reliable Broadcast

Primary Guard 1 Guard 2

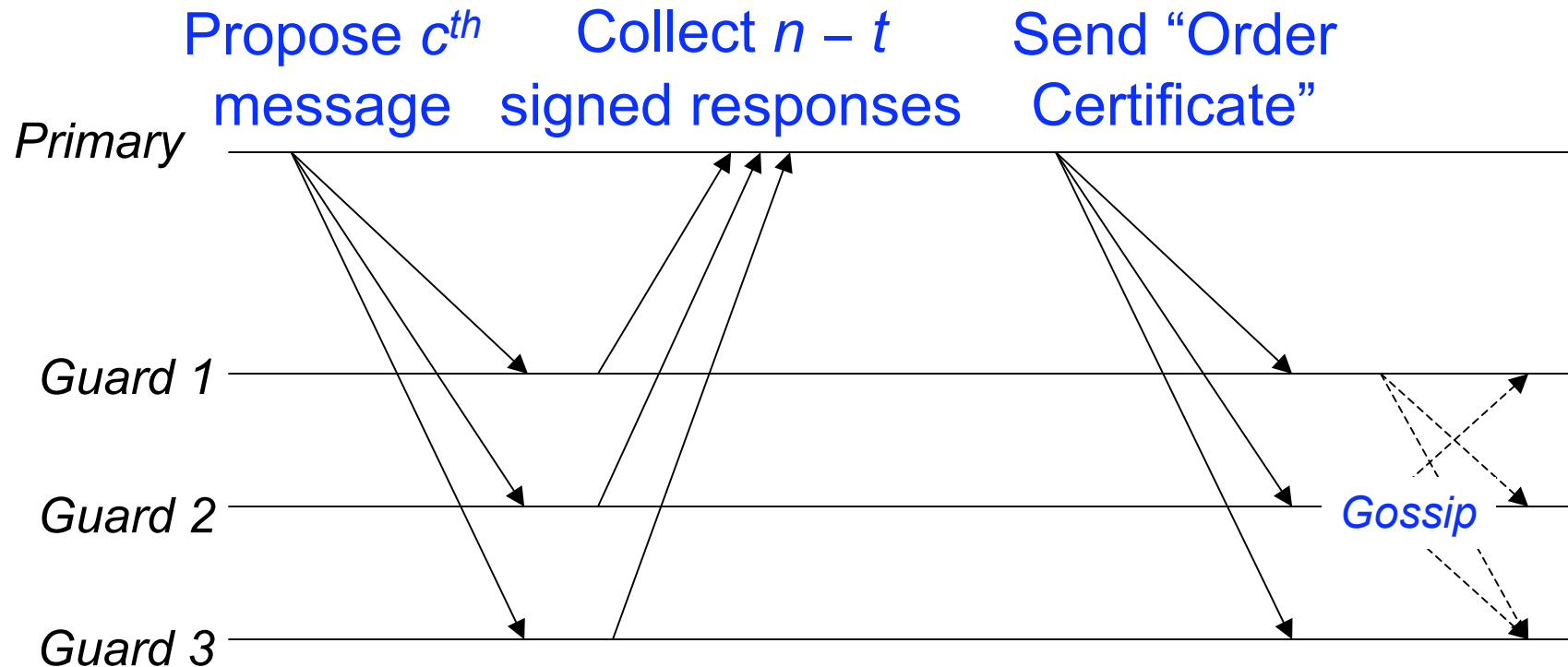


- FIFO
- Ordering
- Persistence
- Relay
- Authenticity

Weaker than consensus!

- Only one “proposer”
- May “block” if proposer faulty

OARCAST protocol



With $3t$ guards, primary cannot generate two order certificates for the same c but different messages

(Re-)configuration

- Tacitly assumed guard graph is static and known
- But communication graph is unknown initially and changes over time
 - Guard graph has to be updated accordingly
- Each host goes through a sequence of *epochs*

Epoch certificates

An epoch certificate for a host contains

1. The host identifier
2. The epoch number
3. The set of identifiers of the guards
4. A hash of the final state of the host in its previous epoch

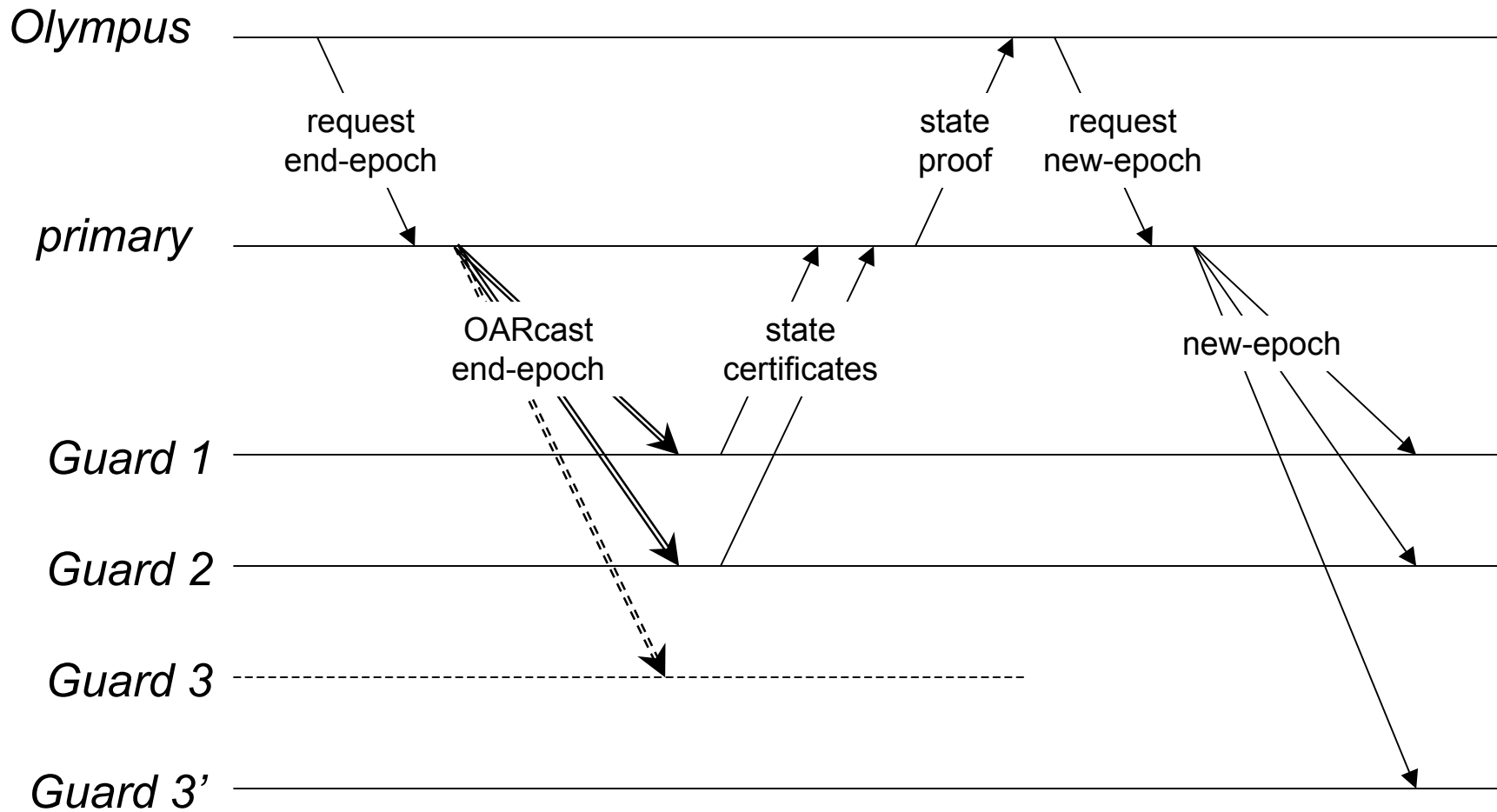
Created and signed by a Certification Authority called the *“Olympus”*

“Changing of the guards”

Protocol to change the epoch of a host

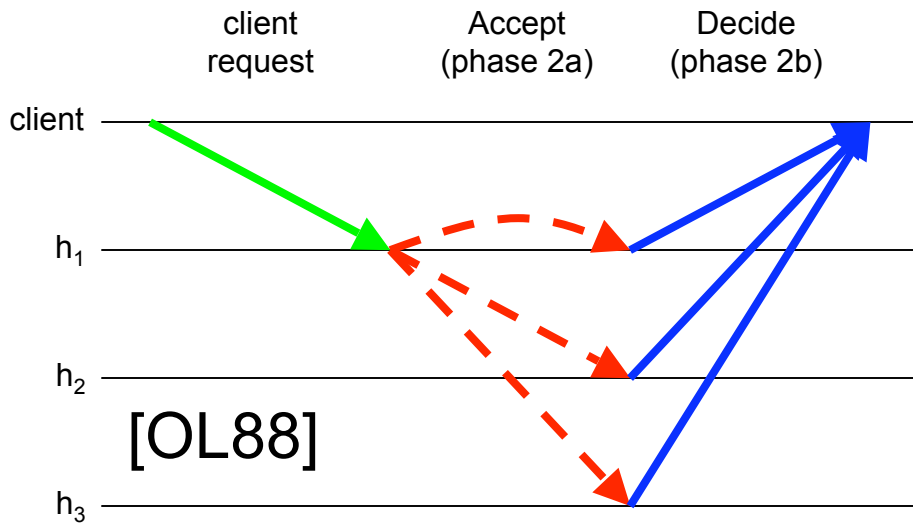
1. Guards send hash of state to Olympus
2. Olympus collects $n - t$ identical hashes
disables OARcasts \Rightarrow fixes state
3. Olympus assigns new guards and generates a new epoch certificate

Example Reconfiguration (when guard 3 is suspected)

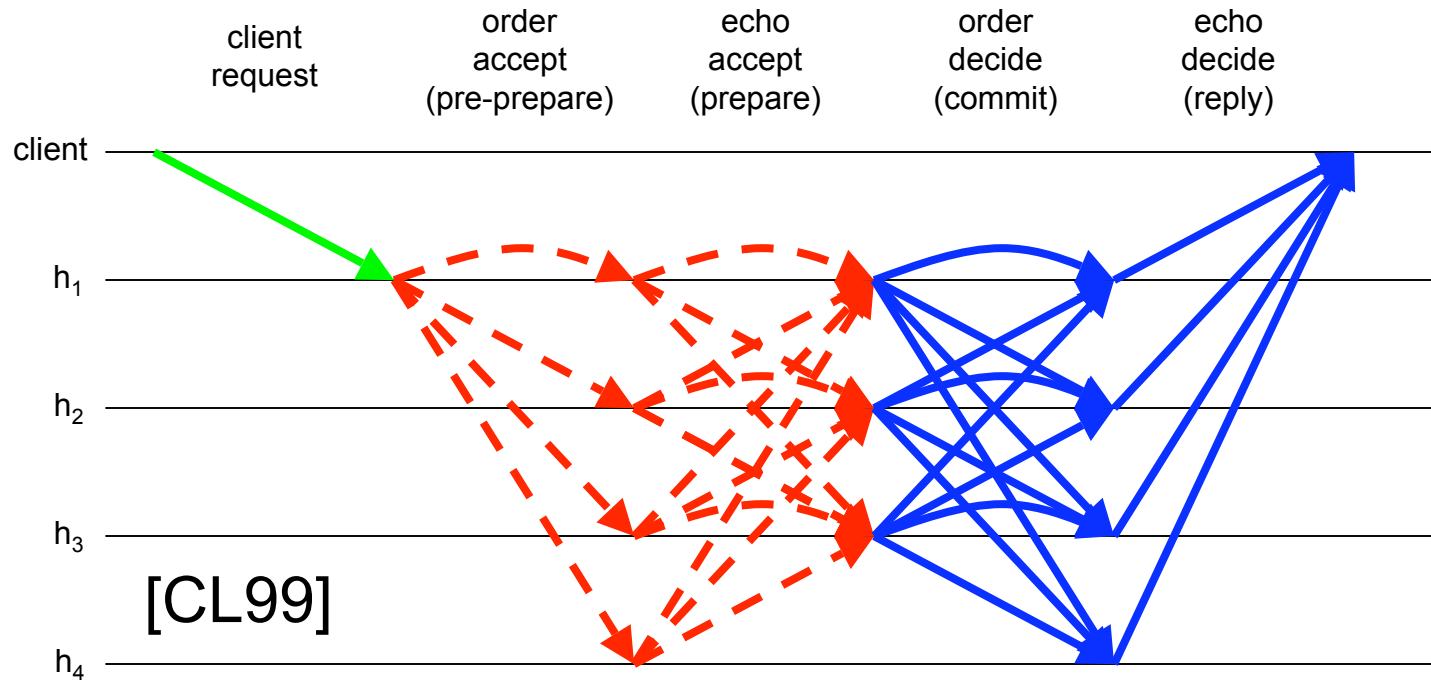
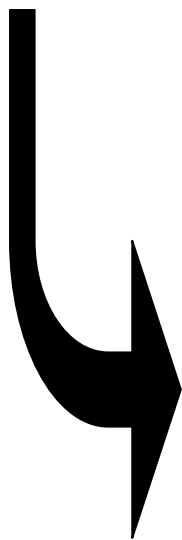


How expensive is Nysiad?

May not be able to hide behind overhead of disk I/O



*Automatic translation
not necessarily worse
than hand-crafted
Byzantine protocols!*



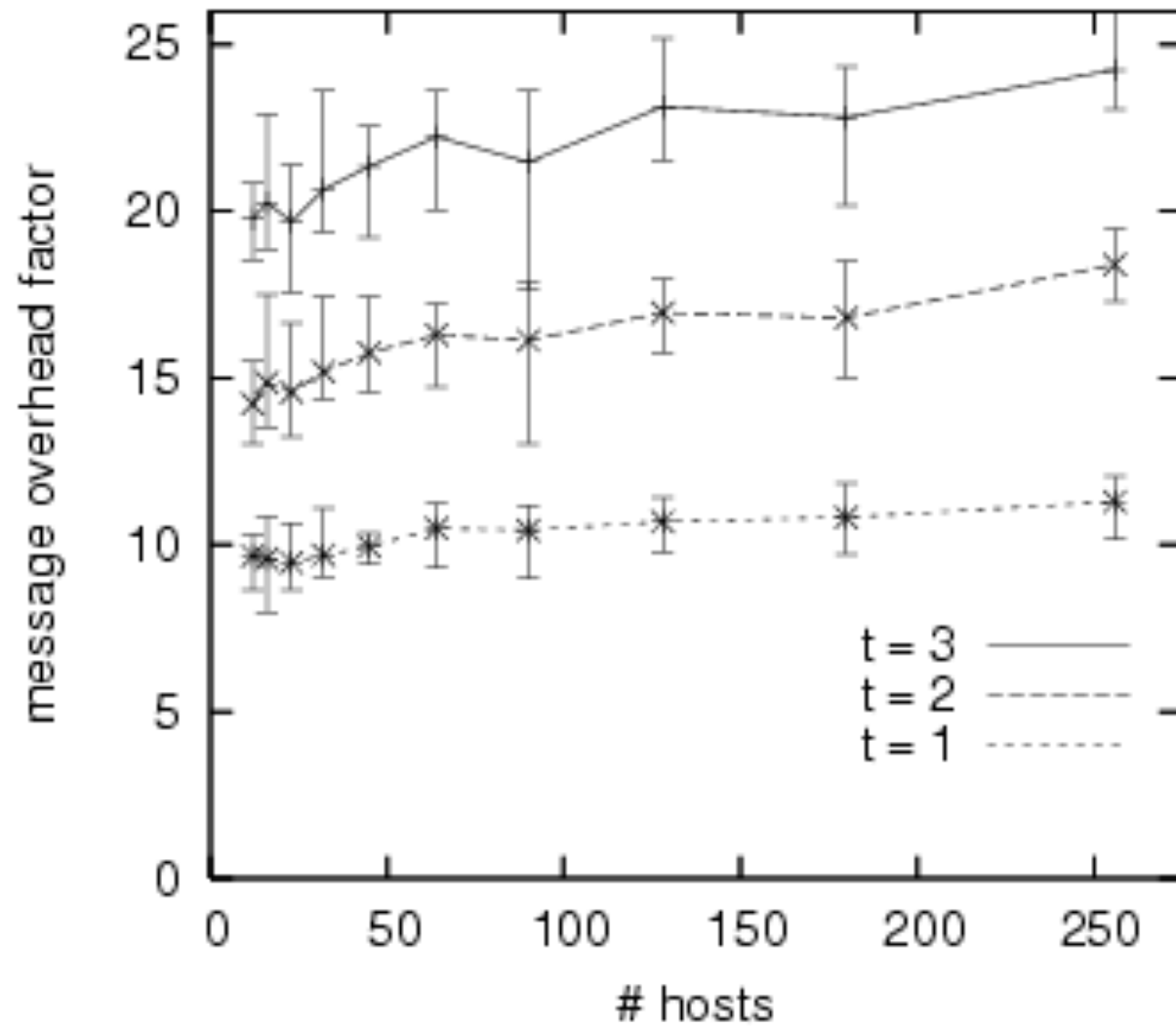
Simulations

- Two applications
 - SSR: scalable source routing [Fuhrmann'05]
 - MCAST: multicast tree
- Two topologies
 - RANDOM: connect to k closest hosts
 - TREE balanced binary tree

In all tests: low-level latency cost is x3

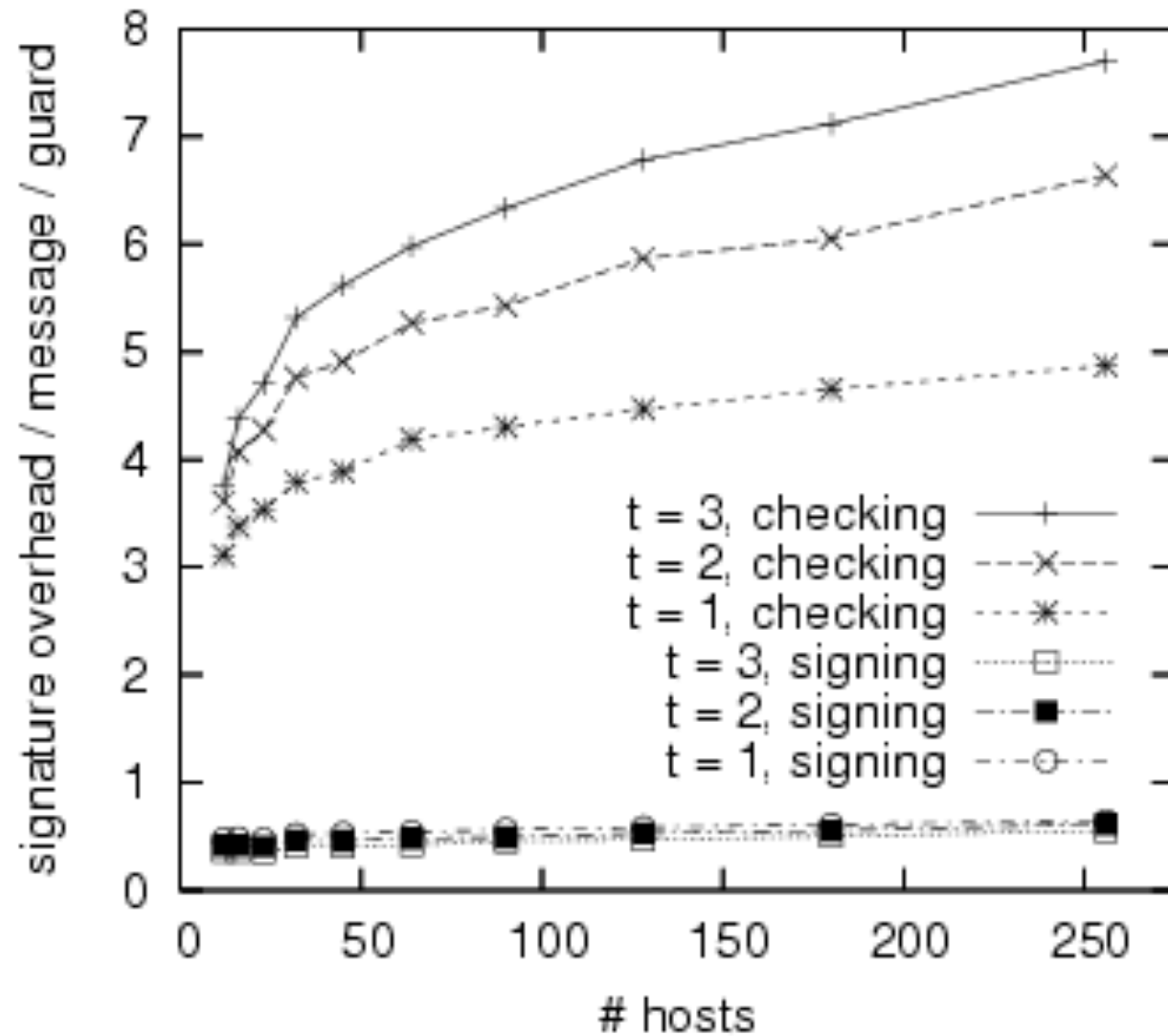
Message Overhead

- SSR / RANDOM
- $k = 3$



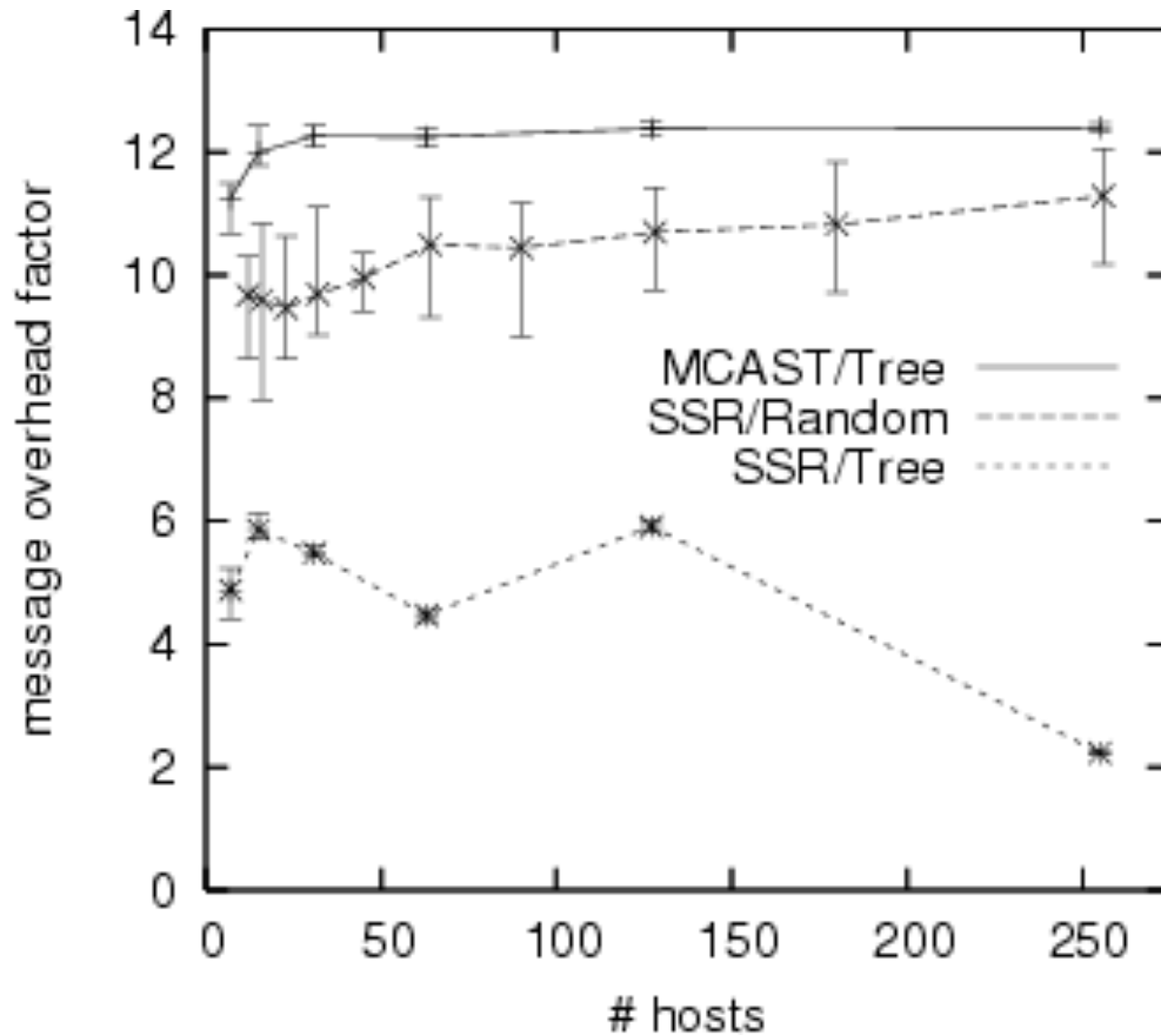
Signing/checking overhead

- SSR / RANDOM
- $k = 3$



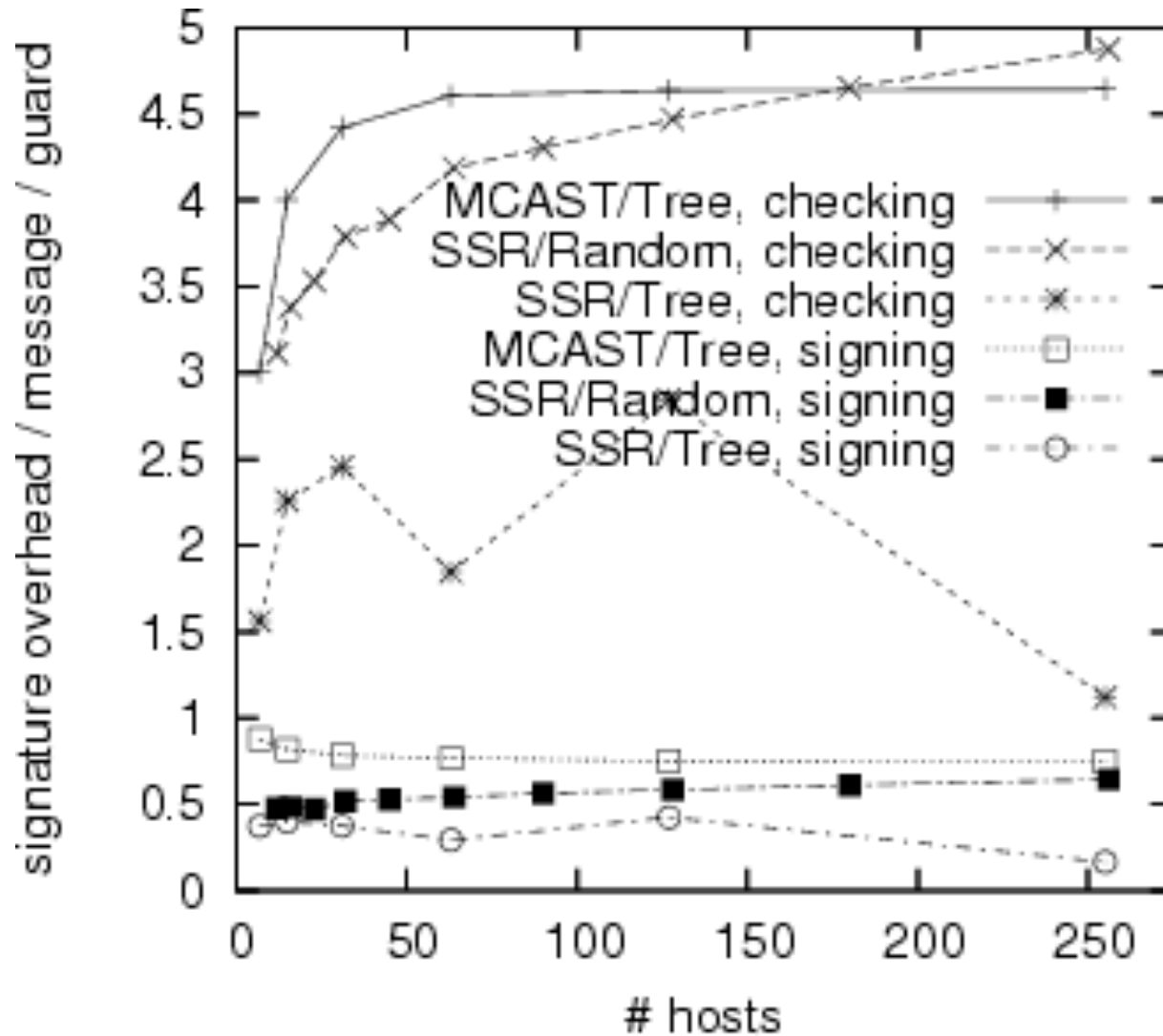
Message overhead

- $k = 2$
- $t = 1$



Signing/checking Overhead

- $k = 2$
- $t = 1$



Performance Discussion

- Hardware multicast would help performance
- Public key crypto use can be reduced
- Combine with Intrusion Detection and Accountability techniques
 - E.g., PeerReview [HKD07]
- Application-dependent optimizations
 - E.g., factor out payload from control traffic

In summary

- Byzantine failures more common than crash failures
- Byzantine fault tolerance possible without solving consensus
 - Disentangles *Byzantine* and *Consensus*
- Generic solution to dynamic configuration in a Byzantine environment
- Overhead substantial, but scales

Questions?

