



IBM Zurich Research Lab

Broadcast and Agreement with Byzantine Faults

HariGovind V. Ramasamy (simply known as “Hari”)

Security and Fault Tolerance in Distributed Systems

Dec 14, 2006

Byzantine Faults

- **More severe than crash faults**
- **Example causes**
 - software errors/bugs
 - operator errors
 - attacks, Trojan horses, viruses
- **Example effects**
 - sending badly formatted messages
 - not taking actions when supposed to
 - taking actions when supposed to, but the wrong action(s)
- **Arbitrary deviations from specified behavior**

Fault Classification by Objective

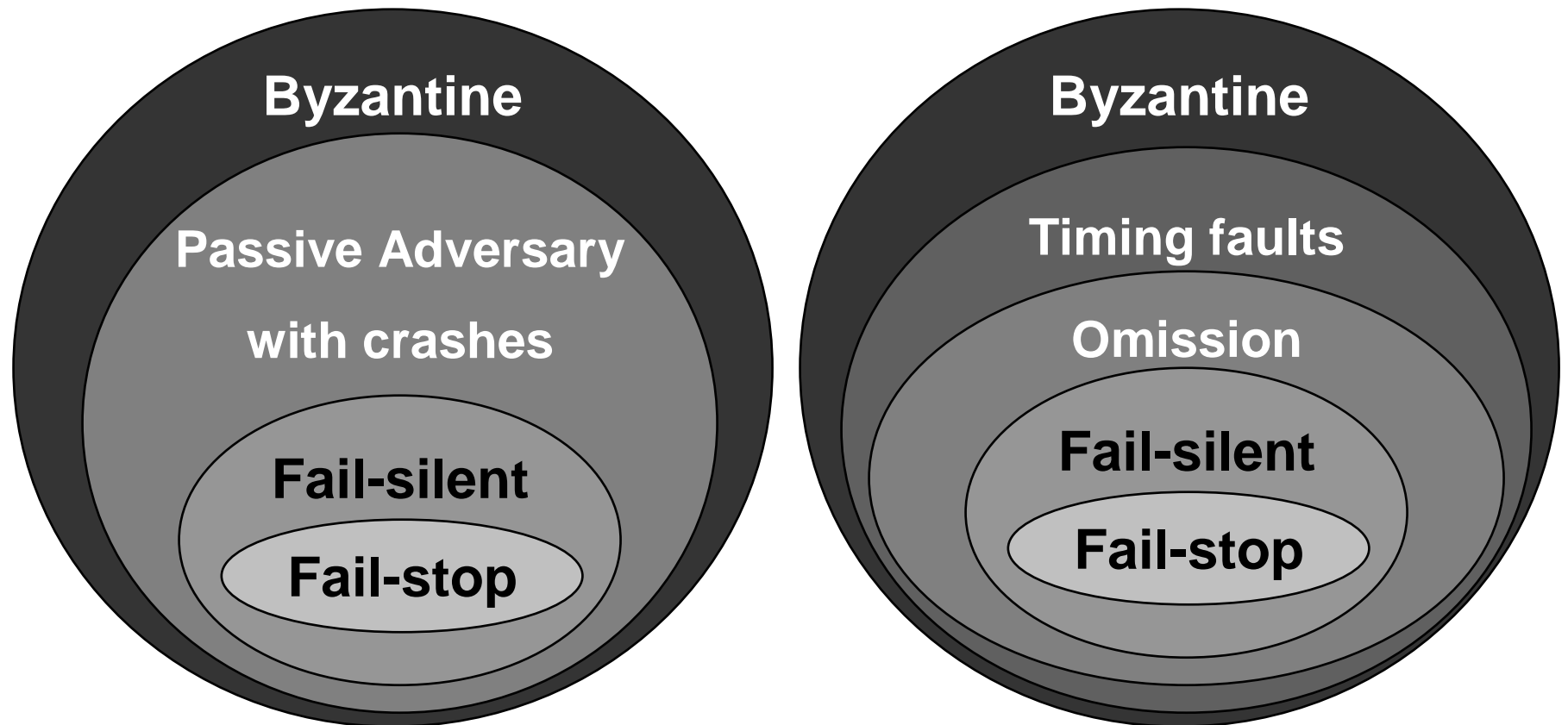
- **malicious or adversarial fault**
 - introduced by a human with objective of harming system
- **non-malicious or non-adversarial fault**
 - introduced without a malicious object
 - sometimes called “benign” or accidental faults

Fault Classification by Objective and Severity

Severity \ Objective	Objective	
	Non-Malicious	
Low	Fail-stop	
▪ ▪ ▪	Fail-silent or crash	Crash-recovery
	Omission faults	
	Timing faults	
High	Benign value faults (e.g., bit flip)	

Severity \ Objective	Objective	
	Malicious	
Low	Passive, e.g., honest-but-curious, semi-honest	
	Passive with crash	
High	Active or Fully Byzantine	

Inclusion Relationships among Fault Classes



System Model for the Rest of the Discussion

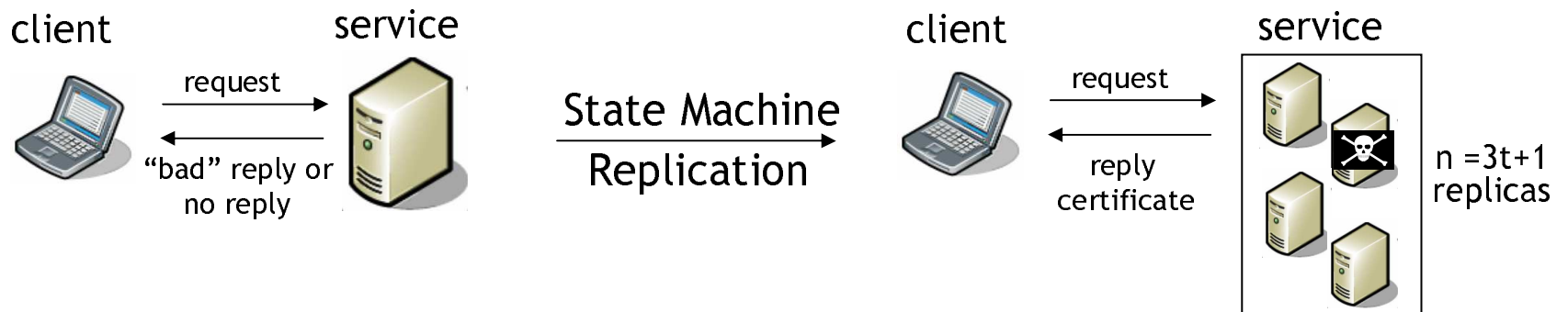
- **$n \geq 3t + 1$ asynchronous servers, tolerant to t Byzantine faults**
 - Faulty servers, under control of adversary, referred to as *corrupted*
 - Non-faulty servers referred to as *correct* or *honest*
 - Asynchronous: no bounds on relative processor speeds
- **Computationally-bounded adversary**
 - Can't break cryptography
- **Cryptographic mechanisms**
 - Digital signatures & Threshold signatures based on public key crypto
 - Secret sharing
- **Trusted initialization and key distribution**
 - Trusted dealer initially distributes keys and key shares
- **Reliable asynchronous network**
 - Asynchronous: No bounds on message delays
 - Reliable: message exchanges between correct parties
 - secure, authenticated channels for each pair of nodes

Outline of Rest of Discussion

- **Consistent Broadcast**
- **Reliable Broadcast or Byzantine Generals Problem**
- **Byzantine Agreement**
- **Atomic Broadcast**
- **Atomic Broadcast \equiv Consensus / Byzantine Agreement**

Discussion: Applications (“Why should anyone care?”)

Service replication for tolerating faults



Consistent Broadcast

Input: *c-broadcast* (only once, only at **designated sender**)

Output: *c-delivery*

Main Idea: **Same content *c-delivered* at any two correct servers**

Termination: Correct server *c-broadcasts* some payload m

⇒ all correct servers eventually *c-deliver* m

But, with faulty sender, no termination guarantee

⇒ **some may *c-deliver*, some may not *c-deliver***

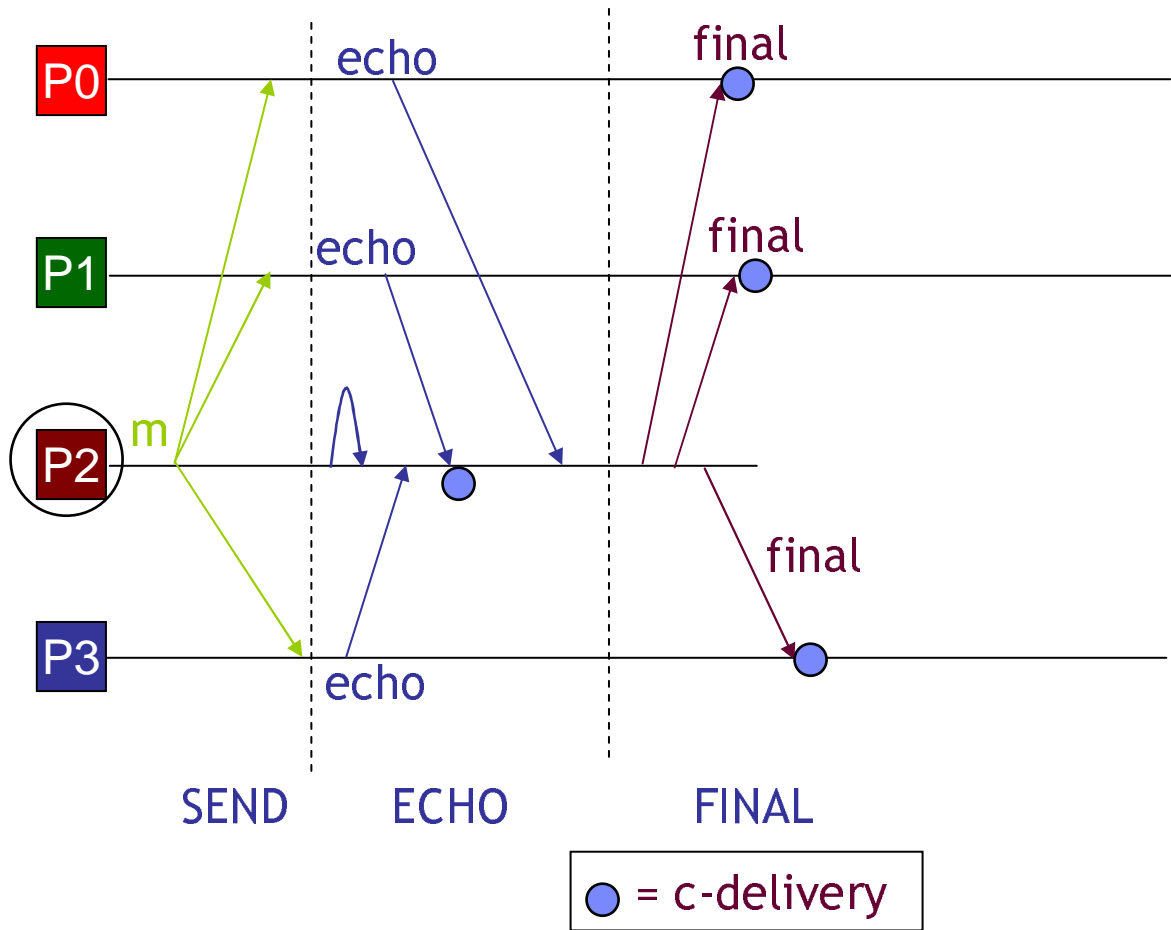
Consistency: Two correct servers *c-delivered* m and $m' \Rightarrow m = m'$

Integrity:

1. Correct server *c-delivers* at most one payload m
2. Designated sender correct ⇒ m was *c-broadcast* by sender

A Protocol for Consistent Broadcast

P2 *c*-broadcasts payload m , by obtaining **signed echoes** from $\lceil \frac{n+t+1}{2} \rceil$ witnesses



Format of echo message = (echo, m, σ) ,
 where $\sigma = \text{sign on } (\text{echo}, 2, m)$
 and 2 indicates the sender, P2

Format of final message = $(\text{final}, m, \Sigma)$,
 where $\Sigma = \lceil \frac{n+t+1}{2} \rceil$ echoes

Message complexity = $O(n)$

Bit complexity = $O(n^2(k + |m|))$ or $O(n(k + |m|))$
 depending on digital or threshold signatures

Background: (n,k,t) Threshold Signature Scheme

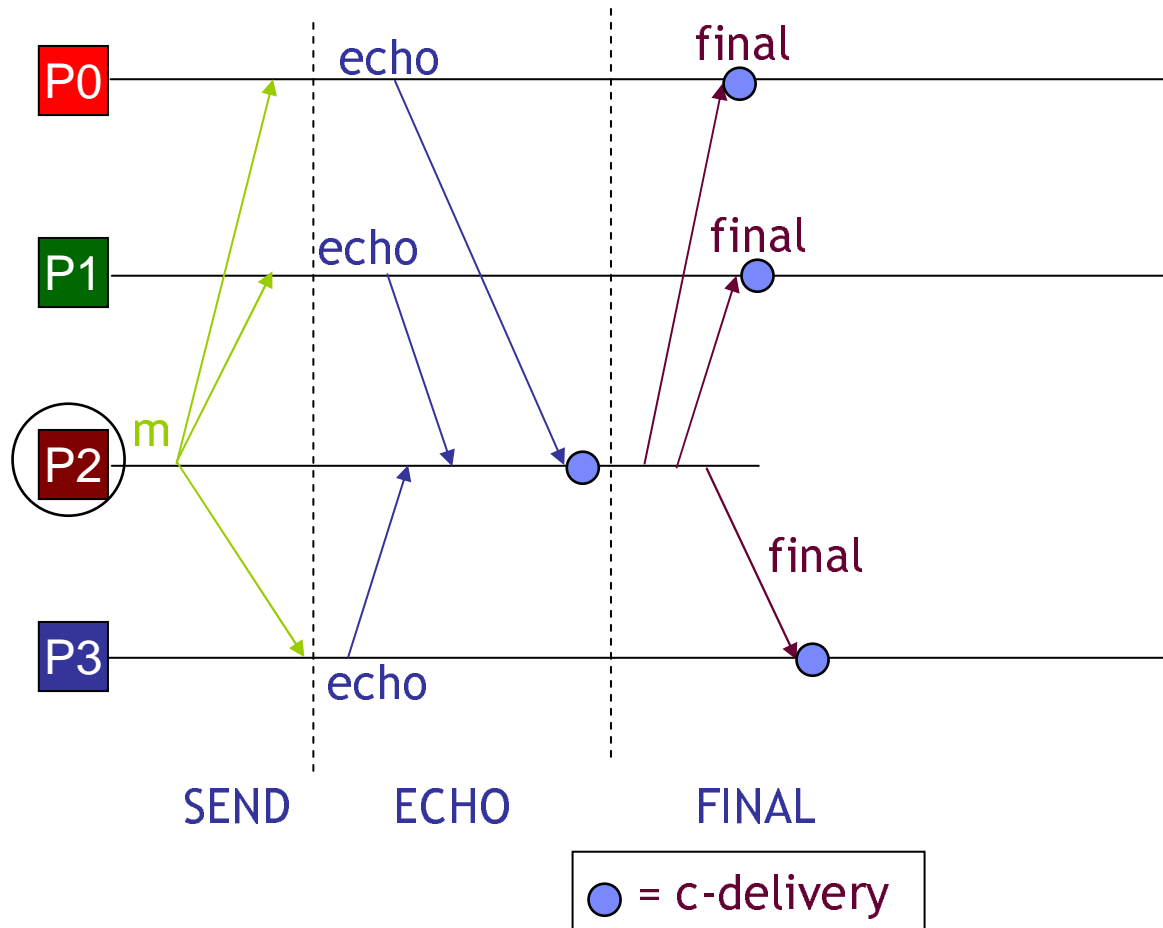
- Will be covered in detail in next lecture
- Just like a digital signature, there is a public key and a private key
- But instead of one public key-private key for each server, there is **one** public key-private key **for all the n** servers
- Each server has not a private key, but a private key **share**
- So, there is one secret (*the* private key), shared by n servers
- To produce a single signature on a single message, **$t < k \leq n-t$** servers have to produce signatures shares on this message using their private key shares

Bit complexity of consistent broadcast = $O(n^2(k + |m|))$ or $O(n(k + |m|))$

depending on digital or threshold signatures

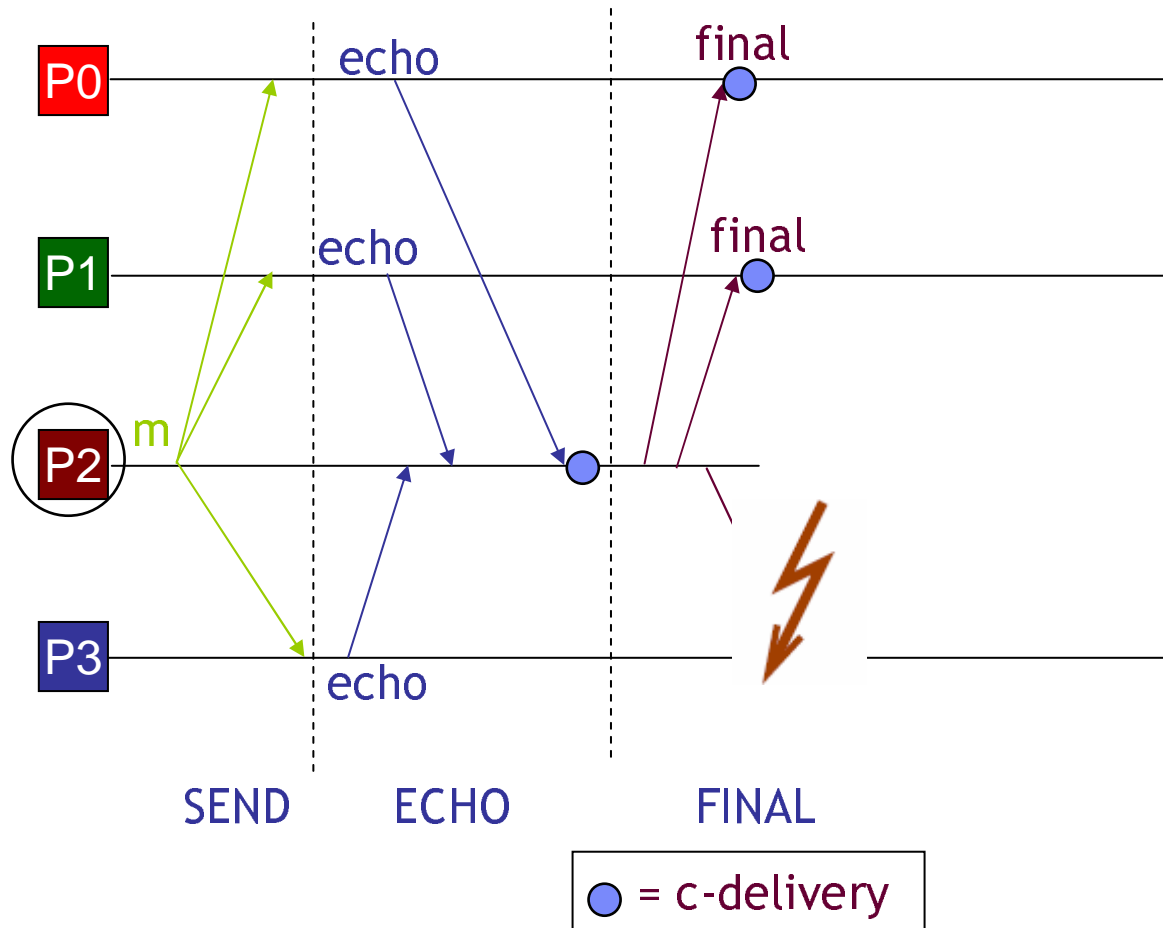
A Protocol for Consistent Broadcast

Why does this work?



A Protocol for Consistent Broadcast

What happens if P2 is faulty?



Reliable Broadcast

Input: *r-broadcast* (only once, only at **designated sender**)

Output: *r-delivery*

Reliable Broadcast = Consistent Broadcast + **Totality**

If any correct server *r-delivers* a message, then ALL correct servers eventually *r-deliver* same message

Either all correct servers *r-deliver* **or** no correct server *r-delivers*

Termination: Correct server *r-broadcasts* some payload m

\Rightarrow all correct parties eventually *r-deliver* m

Consistency: Two correct parties *c-delivered* m and $m' \Rightarrow m = m'$

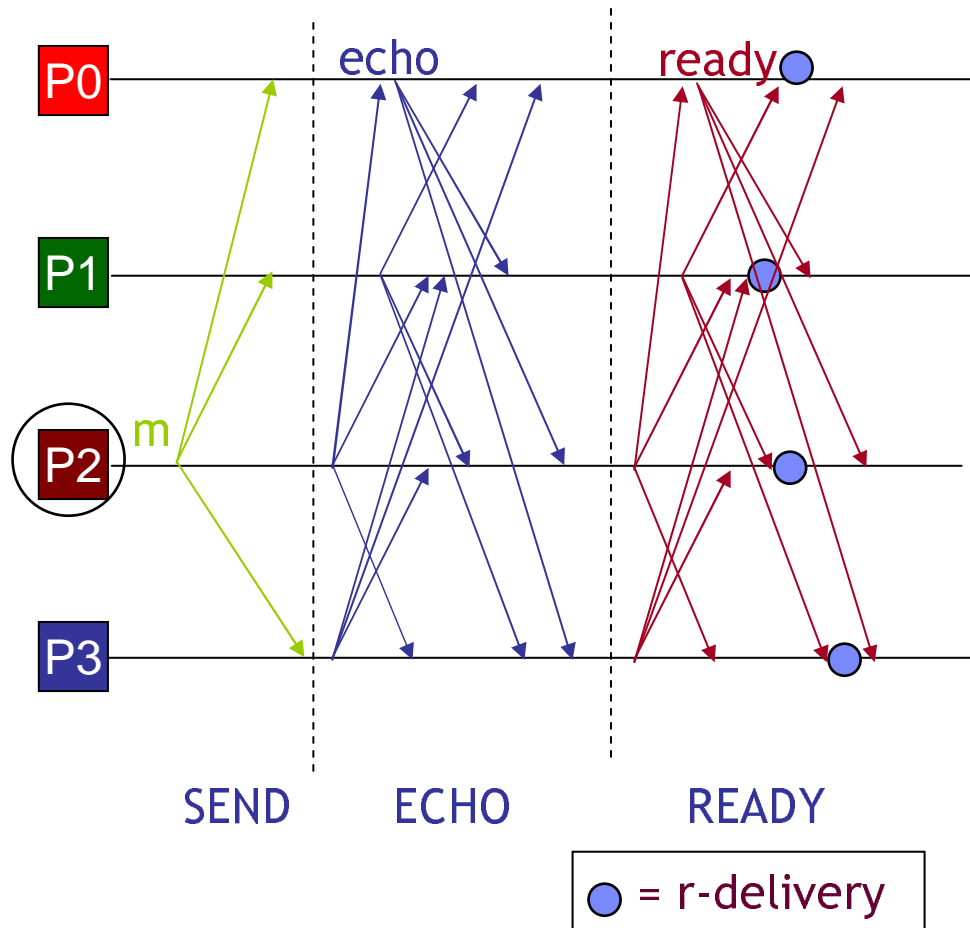
Integrity:

1. Correct server *c-delivers* at most one payload m
2. Designated sender correct $\Rightarrow m$ was *c-broadcast* by sender

A Protocol for Reliable Broadcast

Many-to-many communication pattern

No signatures



Format of echo message =
(echo, m)

Format of ready message =
(ready, m)

Send (ready, m) if:

Not sent (ready, m), AND

a) received $\lceil \frac{n+t+1}{2} \rceil$ echoes

OR

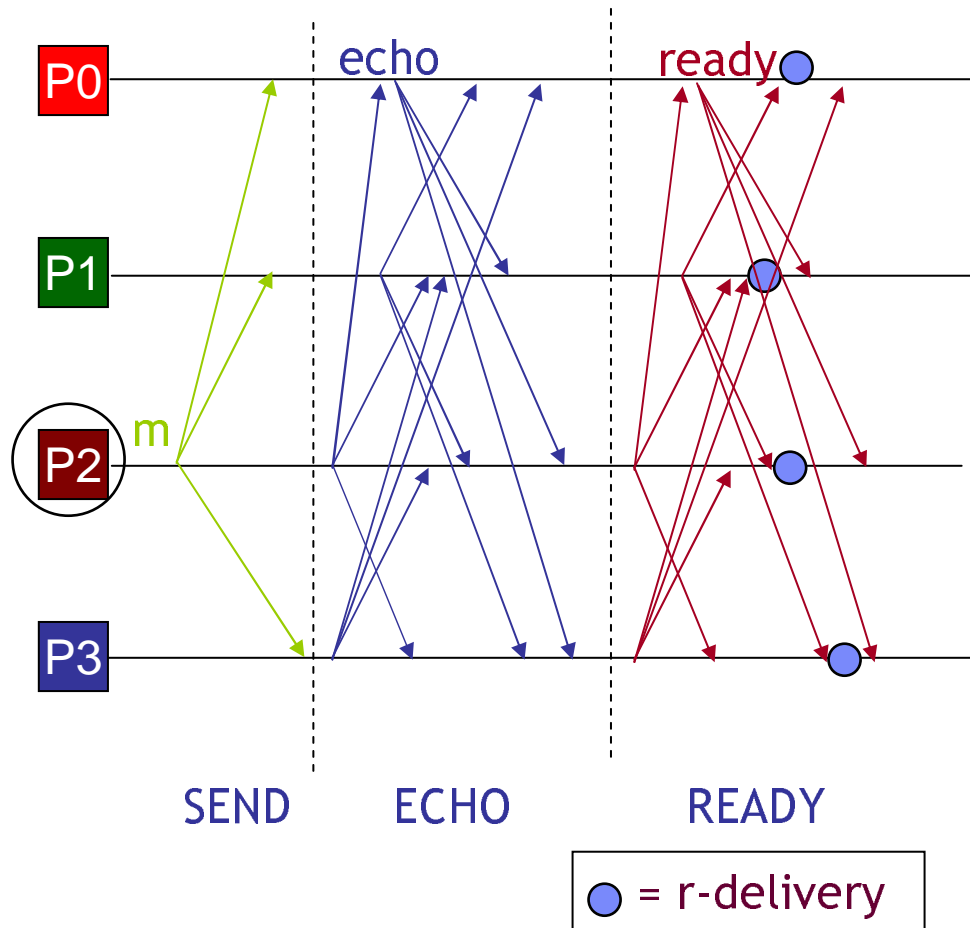
b) received $t + 1$ ready

Message complexity = $O(n^2)$

A Protocol for Reliable Broadcast

How is totality provided?

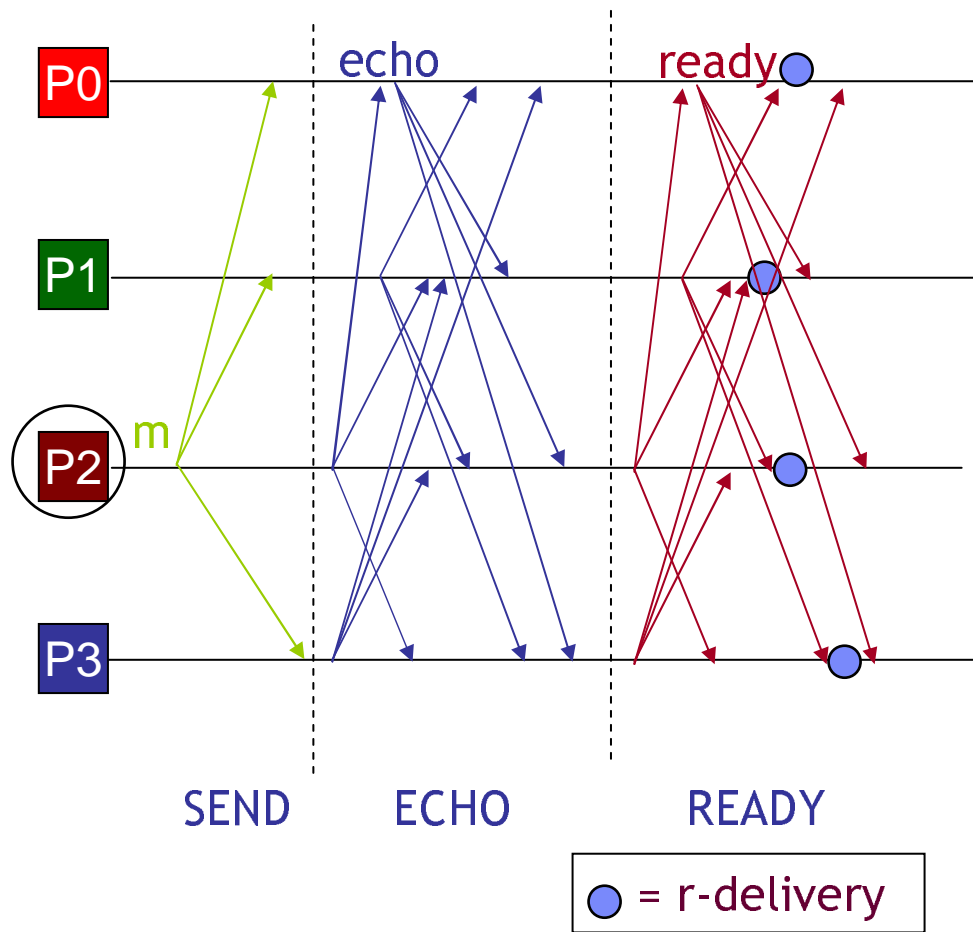
“Amplification” of ready messages from $t + 1$ to $2t + 1$



A Protocol for Reliable Broadcast

What happens if P2 is faulty?

Still Totality holds



Binary Byzantine Agreement

Input: *propose* (b), where b is a bit (0 or 1)

Output: *decide* (b), where b is a bit

All correct servers eventually decide on the same value

Termination: Every correct server eventually *decides*

Validity: All correct servers *propose* $v \Rightarrow$ some correct server eventually *decides* v

Agreement: Two correct servers *decide* v and $v' \Rightarrow v = v'$

FLP Impossibility Result

- **Fischer, Lynch, Patterson 1983**
- **Impossibility of agreement in a fault-prone distributed system**
- **An asynchronous agreement protocol can't ALWAYS guarantee termination, even with a single crash fault**
- **Impossibility based on difficulty of distinguishing between a slow process and crashed process in an asynchronous system**

Shortcuts to circumvent FLP impossibility result

- **Failure detector oracle [Chandra-Toueg]**
- **Randomization**
 - weaken termination property
 - not required to hold always, but only with very high probability
- **Stronger synchrony models**
 - partial synchrony [Dwork, Lynch, Stockmeyer]
 - timed asynchronous [Fetzer-Cristian]

Some Background: Secret Sharing

- **Will be covered in detail next lecture**
- **A dealer generates shares of a secret s , gives one share per server**
- **To recover secret s , at least $t+1$ servers must reveal their shares**
- **For the Byzantine Agreement protocol we are going to discuss next,**
 - Dealer is trusted
 - A sequence of secrets s_0, s_1, \dots
 - Secret s_r used in round r of the protocol
 - Each secret is a random bit or “coin”
 - source of randomization which is required to circumvent FLP
 - $recover(s_r)$ is the interface to the recovery protocol
 - recovers secret s_r

A Randomized Binary Agreement Protocol (Part I)

[Toeug 1984]

upon *propose*(v):

$r \leftarrow 0$

loop

send the signed message $(1\text{-vote}, r, v)$ to all

receive properly signed $(1\text{-vote}, r, v')$ messages from $n - t$ distinct servers

$\Pi \leftarrow$ set of received 1-vote messages including the signatures

$v \leftarrow$ value v' that is contained most often in Π

PHASE 1

r-broadcast the message $(2\text{-vote}, r, v, \Pi)$

wait for *r-delivery* of $(2\text{-vote}, r, v', \Pi)$ messages with valid proofs Π from $n - t$ senders

$v'' \leftarrow$ value v' that is contained most often among the *r-delivered* 2-vote messages

$c \leftarrow$ number of *r-delivered* 2-vote messages with $v' = v''$

recover(s_r) \longrightarrow assemble shared secret

if $c = n - t$ **then**

$v \leftarrow v''$

else

$v \leftarrow s_r$

if $v'' = s_r$ **then**

send the message (decide, v) to all

// note that $v = s_r = v''$

PHASE 2

$r \leftarrow r + 1$

A Randomized Binary Agreement Protocol (Part II)

upon *receiving* $t + 1$ *messages* `(decide, b)`:
send the message `(decide, b)` to all
decide(b)

A Randomized Binary Agreement Protocol

Lemma:

- all correct servers start a given round r with same vote $v = v_0$**
- \Rightarrow all correct servers terminate round r with vote $v = v_0$**

Proof:

- all correct servers start round r with same vote v_0**
- \Rightarrow all correct servers send $(1\text{-vote}, r, v_0)$**
- \Rightarrow at least $n-2t > t$ of 1-votes received at any server have vote v_0**
- \Rightarrow impossible for any server to generate a valid 2-vote with vote $\neq v_0$**
- \Rightarrow All the $n-t$ valid 2-votes received in PHASE 2 have vote = v_0**
- $\Rightarrow c = n-t$ holds with $v'' = v_0$**
- \Rightarrow any correct server terminates round r with vote v_0**

A Randomized Binary Agreement Protocol

Lemma:

correct server P_i sends a `decide` message for v_0 at end of round r

\Rightarrow all correct servers terminate round r with vote v_0

Proof:

P_i sends a `decide` message for v_0 at end of round r

$\Rightarrow v'' = s_r$ was satisfied at P_i at end of round r

Also, $v'' = s_r = v_0$

\Rightarrow majority of 2-votes received at P_i in round r have vote = v_0

\Rightarrow impossible for any server to obtain $n-t$ valid 2-votes with vote $\neq v_0$

\Rightarrow **Either $\{c = n-t \text{ with } v'' = v_0\}$ **or** $\{c < n-t\}$ at any correct server**

\Rightarrow **Either $\{v \leftarrow v'' = v_0\}$ **or** $\{v \leftarrow s_r = v_0\}$ at any correct server**

\Rightarrow any correct server terminates round r with vote v_0

A Randomized Binary Agreement Protocol

Lemma:

Prob. [all correct servers terminating round r with same vote v_0] $\geq \frac{1}{2}$

Proof:

Adversary has control only over message scheduling, faulty servers

\Rightarrow She can affect only value of v'' and c

Observe that, value of coin s_r is revealed only after v'' and c have been assigned at least one correct server

$\Rightarrow v_0$ and s_r are independent

$\Rightarrow \text{Prob. } [v_0 = s_r] = \frac{1}{2}$

A Randomized Binary Agreement Protocol

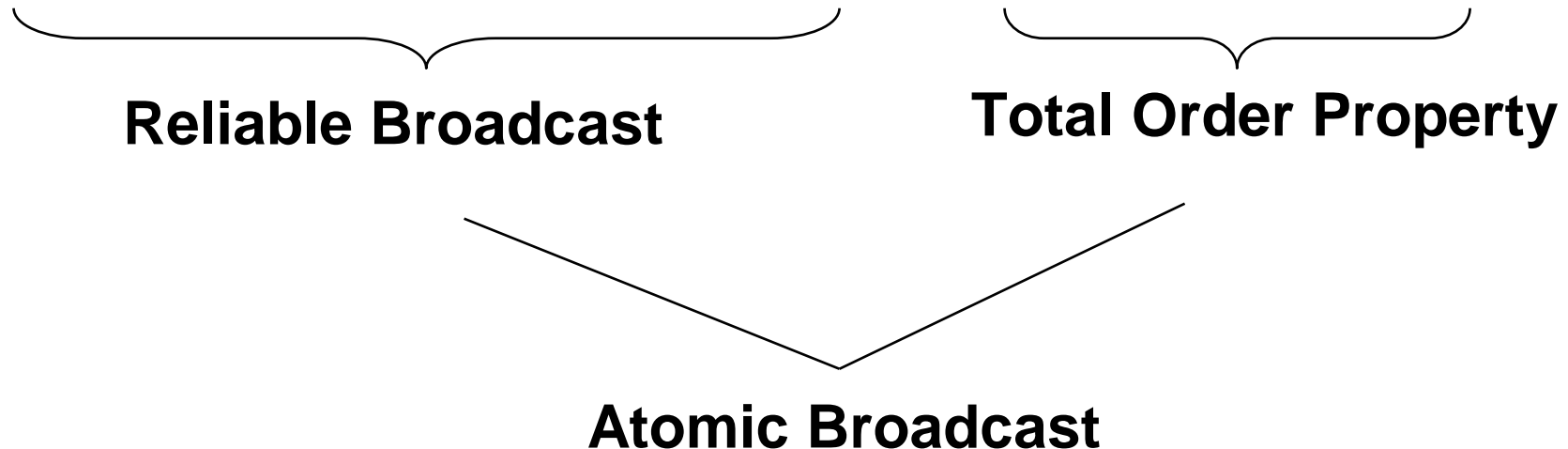
Why does it work?

- 1. all correct servers start a given round with same vote**
⇒ **all correct servers terminate that round with that vote**
- 2. correct server sends a `decide` message for v_0 at end of round r**
⇒ **all correct servers terminate round r with vote v_0**
- 3. Prob. [all correct servers terminating round r with same vote] $\geq \frac{1}{2}$**
- 4. Prob. [reaching agreement in each round] $\geq \frac{1}{2}$**

Prob. distribution of the number of rounds before termination is a Geometric distribution with parameter $\frac{1}{2}$ and mean 2

Atomic Broadcast

- **FIFO order, Causal order are partial orders**
 - different servers can still deliver messages in different orders
- **Desirable to have all servers deliver same set of messages and in the same order**



Atomic Broadcast Specification

Input: *a-broadcasts*

Output: *a-deliveries*

Ensure that all correct parties deliver the same sequence of payloads

Validity

Correct server *a-broadcasts* payload $m \Rightarrow$ some correct server eventually *a-delivers* m

Agreement

Some correct server has *a-delivered* $m \Rightarrow$ all correct parties eventually *a-deliver* m

Total Order

Two correct parties both *a-delivered* distinct payloads m_1 and m_2
 \Rightarrow they have *a-delivered* them in the same order

Integrity

- For any payload m , a correct server *a-delivers* m at most once
- All parties correct $\Rightarrow m$ was previously *a-broadcast* by some server

Atomic Broadcast \equiv Consensus (Crash fault model)

- **What does equivalence mean?**

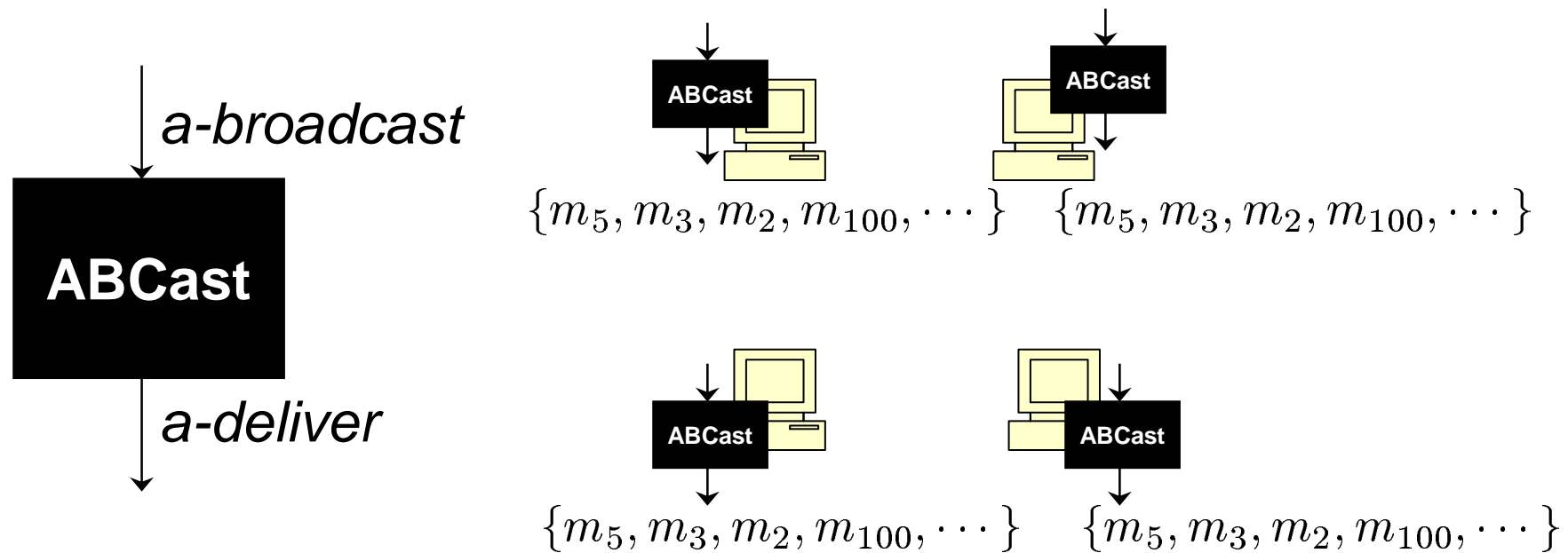
- given an algorithm for consensus, we can solve atomic broadcast
- given an algorithm for atomic broadcast, we can solve consensus

Atomic Broadcast \equiv Byzantine Agreement (Byzantine fault model)

- **What does equivalence mean?**
 - given an algorithm for BA, we can solve atomic broadcast
 - given an algorithm for atomic broadcast, we can solve BA

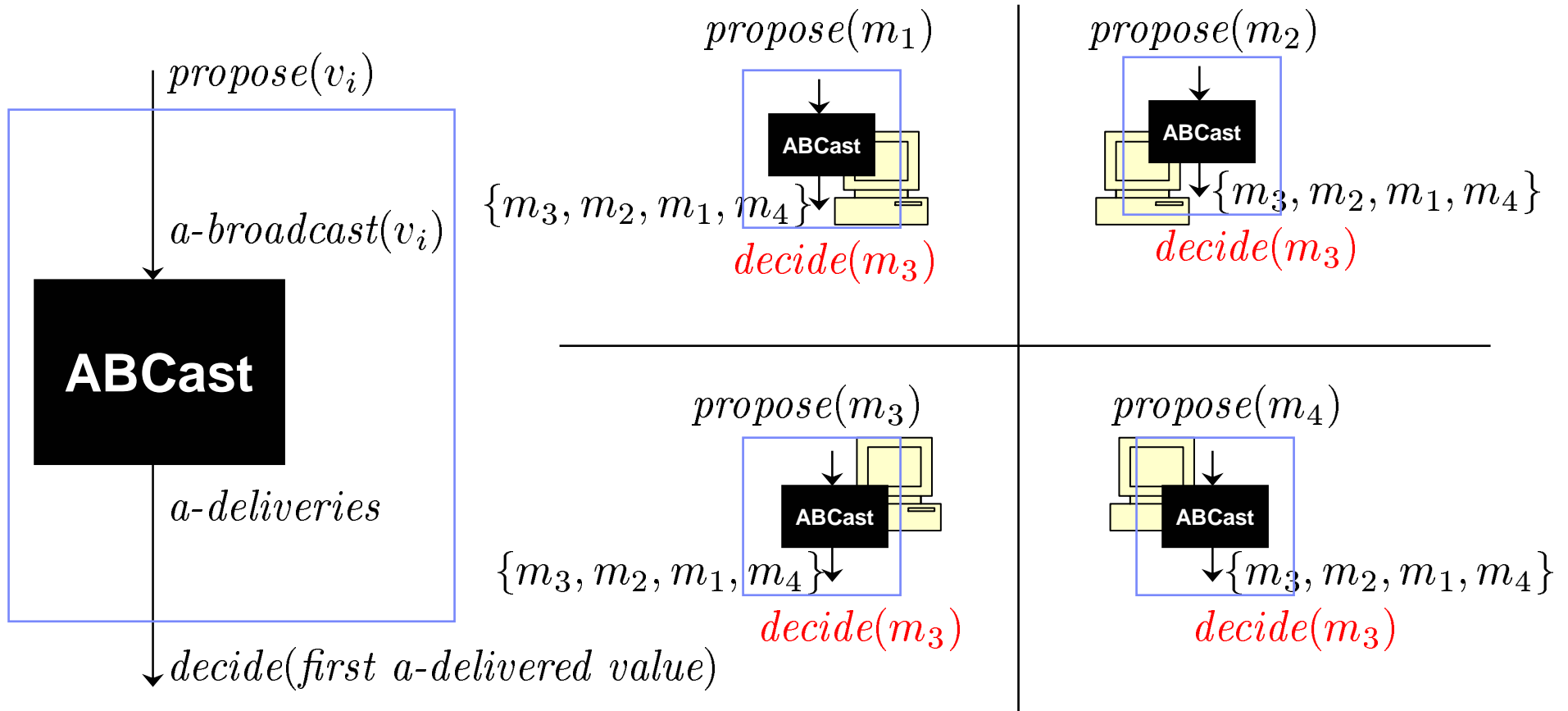
If Atomic Broadcast, then Consensus?

Given a black box that can do atomic broadcast....



Sequence of *a-delivered* messages
same at all correct servers

If Atomic Broadcast, then Consensus



If Consensus, then Atomic Broadcast

initialization:

$\mathcal{R} \leftarrow \emptyset$ // set of *r-delivered* messages
 $\mathcal{A} \leftarrow \emptyset$ // set of *a-delivered* messages
 $r \leftarrow 0$ // round number

upon *a-broadcast*(*m*):

r-broadcast(*m*)

upon *r-deliver*(*m*):

$\mathcal{R} \leftarrow \mathcal{R} \cup \{m\}$

repeat forever: // concurrently with the above statements

if $\mathcal{R} \setminus \mathcal{A} \neq \emptyset$ then

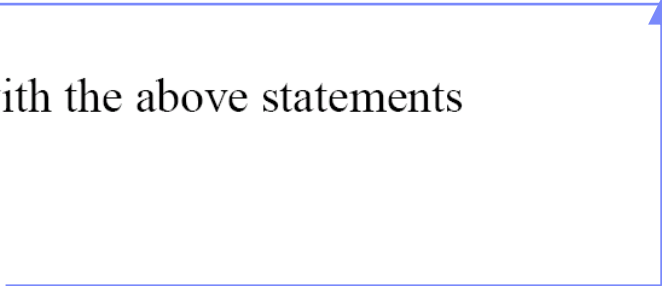
propose($\mathcal{R} \setminus \mathcal{A}$) in consensus *r*

wait for *decide*(\mathcal{S}) of consensus *r*

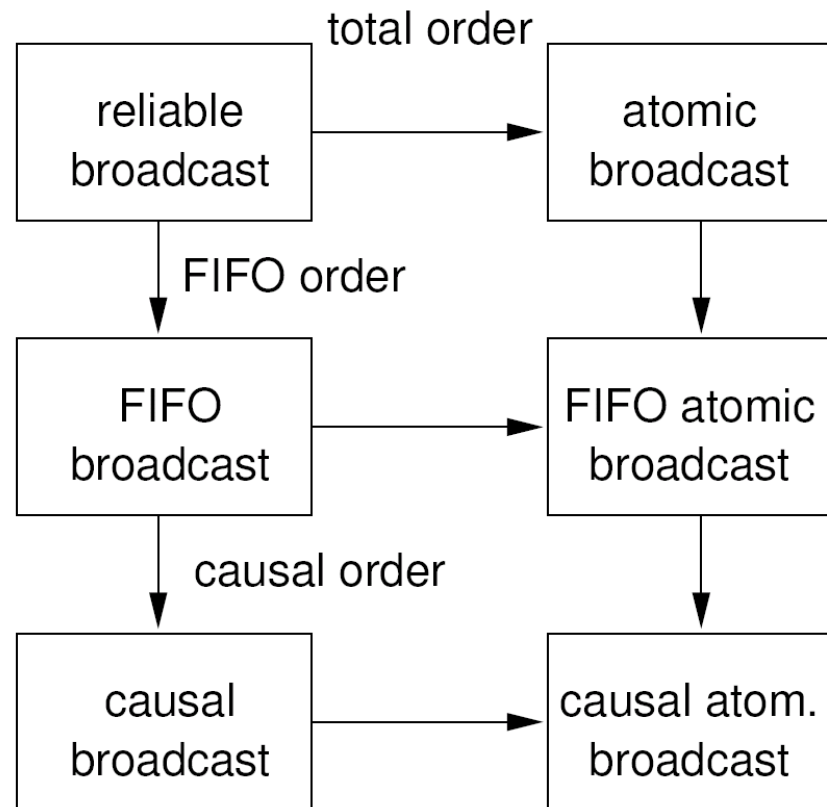
a-deliver all messages in $\mathcal{S} \setminus \mathcal{A}$ in some deterministic order

$\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{S}$

$r \leftarrow r + 1$

1. All correct servers will eventually decide for consensus *r*
 2. Decision set \mathcal{S} for consensus *r* is same at all correct servers
- 

Relations Among the Broadcast Primitives [HT93]



Questions



How to reach me?

HariGovind Ramasamy <hvr@zurich.ibm.com>

<http://www.zurich.ibm.com/~hvr>

Phone +41 44 724-8314