The Part-Time Parliament

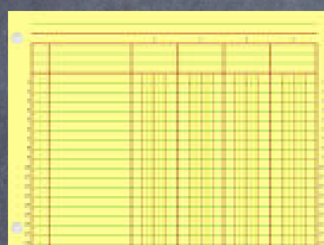
- Parliament determines laws by passing sequence of numbered decrees
- Legislators can leave and enter the chamber at arbitrary times
- No centralized record of approved decrees–instead, each legislator carries a ledger

Government 101

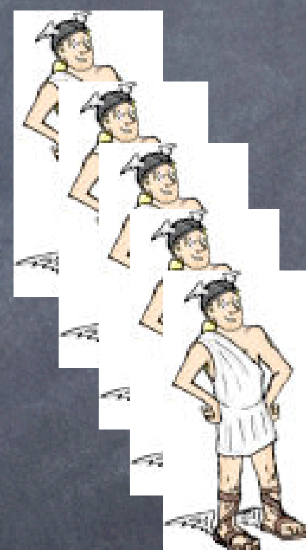
- No two ledgers contain contradictory information
- If a majority of legislators were in the Chamber and no one entered or left the Chamber for a sufficiently long time, then
  - any decree proposed by a legislator would eventually be passed
  - any passed decree would appear on the ledger of every legislator

Supplies

Each legislator receives



ledger



lots of messengers

scratch paper

pen with indelible ink

hourglass

Back to the future

- A set of processes that can propose values
- Processes can crash and recover
- Processes have access to stable storage
- Asynchronous communication via messages
- Messages can be lost and duplicated, but not corrupted

The Game: Consensus

SAFETY

- Only a value that has been proposed can be chosen
- Only a single value is chosen
- A process never learns that a value has been chosen unless it has been

LIVENESS

- Some proposed value is eventually chosen
- If a value is chosen, a process eventually learns it

The Players

- Proposers
- Acceptors
- Learners

Choosing a value

Have a single acceptor

Choosing a value

majority

Have a ~~single~~ acceptor

of

Using a majority set guarantees that at most one value is chosen

Accepting a value

- Suppose only one proposer proposes a single value
- assume no failures
- that value should be accepted!

Accepting a value

- Suppose only one proposer proposes a single value
- assume no failures
- that value should be accepted!

P1: Acceptors must accept
first received proposal

Accepting a value

P1: Acceptors must accept
first received proposal

- Choosing a value requires a majority of acceptors to accept that value
- What if we have multiple proposers, each proposing a different value?
- Acceptors must accept multiple proposals (each identified by pair (n, value))

?

Guaranteeing uniqueness

P2. If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v

How do we implement P2?

What about: If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v

- It satisfies P1 and P2, but it not implementable in an asynchronous system!

Another take on P2

- If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v

Another take on P2

- If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v
- If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v

Implementing P2

- If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v

How would we enforce this? Use as inspiration a possible proof!

- Assume some (m, v) has been chosen by a set C of acceptors
- Assume, by induction, that all proposal issued with numbers in the range $m..n$-1 proposed v
- Then, any acceptor that accepts a proposal with number $m..n$-1 has value v
- The proposal with number n has value v if the following invariant holds:
- Let S be a majority set. of acceptors When a proposer issues a value v

Implementing P2

If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v

Achieved by enforcing the following invariant

For any v and n, if a proposal with value v and pid n is issued, then there is a majority-set S of acceptors such that one of the following holds:

- no acceptor in S has accepted any proposal numbered less than n
- v is the value of the highest-numbered proposal among all proposal numbered less than n accepted by the acceptors in S

The proposer's protocol

1. A proposer chooses a new *n* and sends <*prepare,n*> to each member of some set of acceptors, asking it to respond with:
   a. A promise never again to accept a proposal numbered less than *n*, and
   b. The accepted proposal with highest number less than *n* if any.

2. If proposer receives a response from a majority of acceptors, then it can issue <*accept*(*n,v*)> where *v* is the value of the highest numbered proposal among the responses, or is any value selected by the proposer if responders returned no proposals

The acceptor's protocol

1. Can ignore any request without violating safety
2. Can always respond to *prepare* messages
3. Can respond to <*accept*(*n,v*)> iff it has not promised not to–i.e. it has not responded to <*prepare,n'*> with *n'* > *n*

Acceptor must remember

- highest numbered proposal ever accepted
- highest numbered prepare request to which it responded

Learning chosen values

Once a value is chosen, it is forwarded to the learners. Many strategies are possible:

i. Each acceptor informs each learner

ii. Acceptors inform a distinguished learner, who informs the other learners

iii. Something in between

Liveness

Progress is not guaranteed:

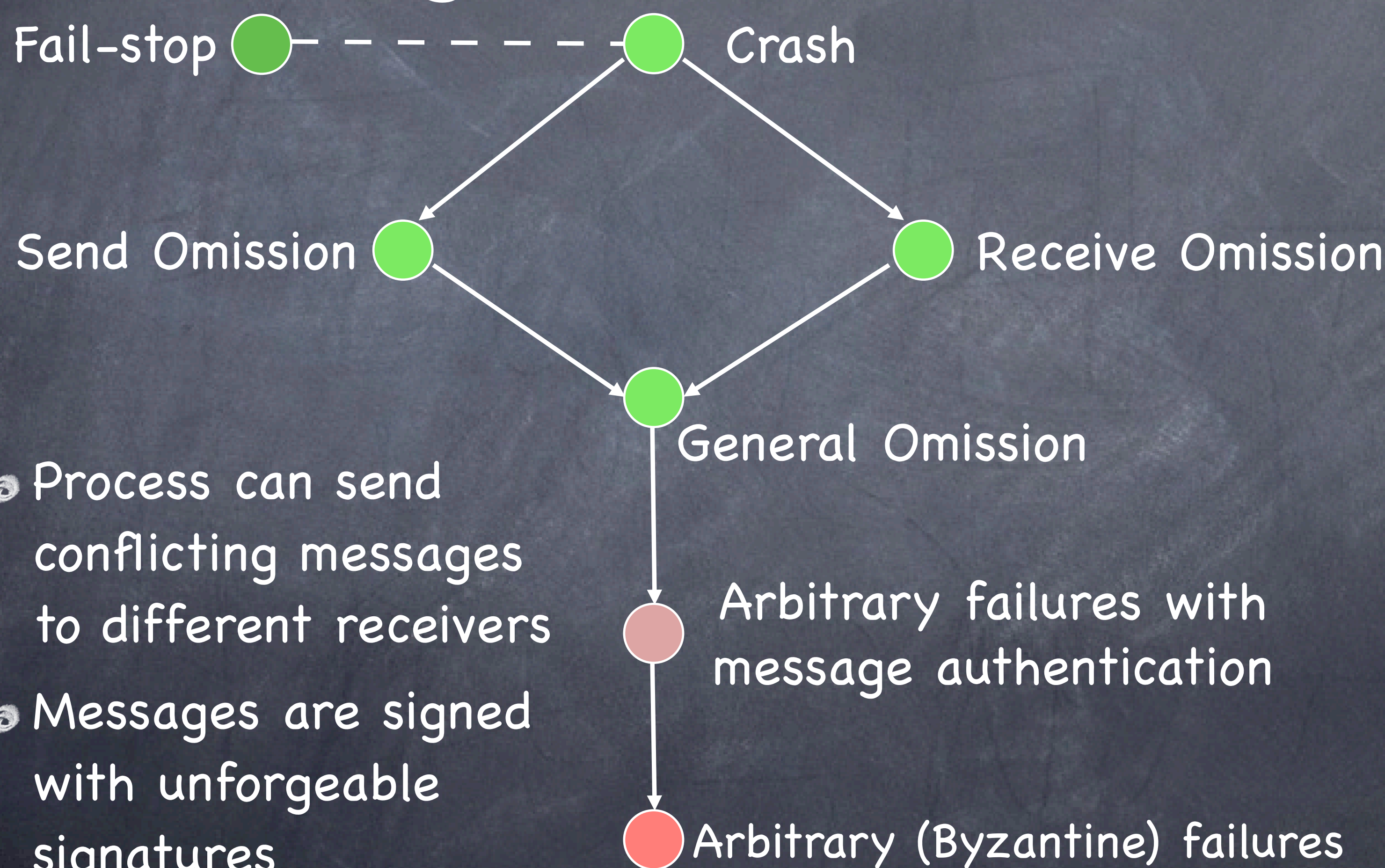
$n_1 < n_2 < n_3 < n_4 < \ldots$

p_1	Time ↓	p_2
<*propose*,n_1>		
		<*propose*,n_2>
<*accept*(n_1,v_1)>		
		<*accept*(n_2,v_2)>
<*propose*,n_3>		
		<*propose*,n_4>

All proposers are equal, but some more so than others

- Elect a distinguished proposer
- Can't be done reliably in asynchronous systems, so...
  - real time
  - randomization

Arbitrary failures with message authentication



- Process can send conflicting messages to different receivers
- Messages are signed with unforgeable signatures

Valid messages

A valid message m has the following form:

in round 1:

$< m : s >$ (m is signed by the sender)

in round r > 1, if received by p from q:

$< m : p_1 : p_2 : \ldots : p_r >$ where

- p_1 = sender; $p_r = q$
- $p_1, \ldots, p_r$ are distinct from each other and from p
- message has not been tampered with

AFMA: The Idea

- A correct process p discard all non-valid messages it receives
- If a message is valid,
  - it “extracts” the value from the message
  - it relays the message, with its own signature appended
- At round f + 1:
  - if it extracted exactly one message, p delivers it
  - otherwise, delivers SF

AFMA: The Protocol

sender s in round 0:

extract m

sender in round 1:

send $< m:s >$ to all

Process p in round k, $1 \leq k \leq f+1$

if p extracted m from a valid message $< m:p_1: \ldots :p_{k-1}>$ in round $k - 1$ and $p \neq$ sender then

send $< m:p_1: \ldots :p_{k-1}:p>$ to all

receive round k messages from all processes

for each valid round k message $< m:p_1: \ldots :p_{k-1}:p_k>$ received by p

if p has not previously extracted m then

extract m

if $k = f+1$ then

if in the entire execution p has extracted exactly one m then

deliver(m)

else deliver(SF)

halt

Termination

```
sender s in round 0:
    extract m
sender in round 1:
    send < m:s > to all
Process p in round   k, 1 ≤ k ≤ f+1
    if p extracted m from a valid message <m:p1: … :pk-1>
    in round k - 1 and p ≠ sender then
        send <m:p1: … :pk-1:p> to all
    receive round k messages from all processes
    for each valid round k message < m:p1: … :pk-1:pk>
    received by p
        if p has not previously extracted m then
            extract m
    if k = f+1 then
        if in the entire execution p has extracted exactly
                one m then
            deliver(m)
        else deliver(SF)
        halt
```

In round $f+1$, every correct process delivers either m or SF and then halts

Agreement

```
sender s in round 0:
  extract m
sender in round 1:
  send < m:s > to all
Process p in round    k, 1 ≤ k ≤ f+1
  if p extracted m from a valid message <m:p1: … :pk-1>
    in round k - 1 and p ≠ sender then
      send <m:p1: … :pk-1:p> to all
  receive round k messages from all processes
  for each valid round k message < m:p1: … :pk-1:pk>
    received by p
      if p has not previously extracted m then
          extract m
  if k = f+1 then
      if in the entire execution p has extracted exactly
                  one m then
          deliver(m)
      else deliver(SF)
      halt
```

Lemma If a correct process extracts m, then every correct process eventually extracts m

Proof

Let r be the earliest round in which some correct process extracts m. Let that process be p.

- if p is the sender, then in round 1 p sends a valid message to all. All correct processes extract message in round 1
- otherwise, p has received in round r a message

  $\langle m:p_1:p_2: \dots :p_r \rangle$

- Claim: $p_1, p_2, \dots, p_r$ are all faulty
  - true for $p_1 = s$
  - Suppose p_j, $1 \le j \le r$, were correct
    - p_j signed and relayed message in round j
    - p_j extracted message in round $j - 1$

  CONTRADICTION
- If $r \le f$, p will send a valid message

  $\langle m:p_1:p_2: \dots :p_r:p \rangle$

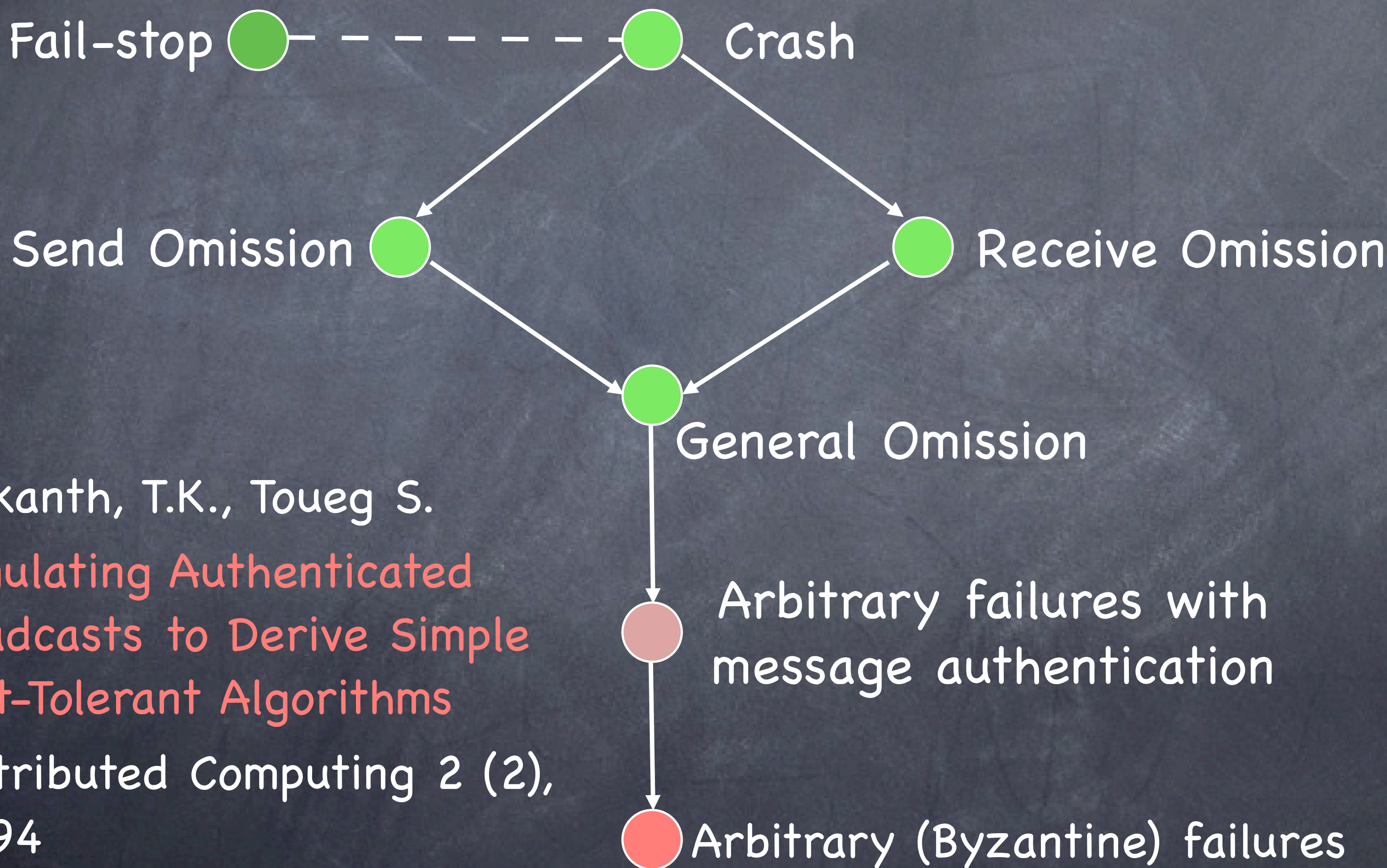
  in round $r + 1 \le f + 1$ and every correct process will extract it in round $r + 1 \le f + 1$
- If $r = f + 1$, by Claim above, $p_1, p_2, \dots, p_{f+1}$ faulty
  - At most f faulty processes
  - CONTRADICTION

Validity

```
sender s in round 0:
1:  extract m
sender in round 1:
2:  send < m:s > to all
Process p in round    k, 1 ≤ k ≤ f+1
3:  if p extracted m from a valid message <m:p1: … :pk-1>
    in round k - 1 and p ≠ sender then
4:      send <m:p1: … :pk-1:p> to all
5:  receive round k messages from all processes
6:  for each valid round k message < m:p1: … :pk-1:pk>
    received by p
7:      if p has not previously extracted m then
8:          extract m
9:  if k = f+1 then
10:     if in the entire execution p has extracted exactly
                one m then
11:         deliver(m)
12:     else deliver(SF)
13:     halt
```

From Agreement and the observation that the sender, if correct, delivers its own message.

TRB for arbitrary failures



Srikanth, T.K., Toueg S.

Simulating Authenticated Broadcasts to Derive Simple Fault-Tolerant Algorithms

Distributed Computing 2 (2), 80-94

AF: The Idea

- Identify the essential properties of message authentication that made AFMA work
- Implement these properties without using message authentication

AF: The Approach

- Introduce two primitives

  broadcast(p,m,i) (executed by p in round i)
  accept(p,m,i) (executed by q in round $j \geq i$)

- Give axiomatic definitions of broadcast and accept
- Derive an algorithm that solves TRB for AF using these primitives
- Show an implementation of these primitives that does not use message authentication

Properties of broadcast and accept

- Correctness If a correct process p executes broadcast(p,m,i) in round i, then all correct processes will execute accept(p,m,i) in round i
- Unforgeability If a correct process q executes accept(p,m,i) in round j ≥ i, and p is correct, then p did in fact execute broadcast(p,m,i) in round i
- Relay If a correct process q executes accept(p,m,i) in round j ≥ i, then all correct processes will execute accept(p,m,i) by round j + 1

AF: The Protocol - 1

sender s in round 0:

0: extract m

sender s in round 1:

1: broadcast (s,m,1)

Process p in round k, $1 \leq k \leq f + 1$

2: if p extracted m in round k - 1 and p $\neq$ sender then

4: broadcast (p,m,k)

5: if p has executed at least k accept(q_i,m,j_i) $1 \leq i \leq k$ in rounds 1 through k (where (i) q_i distinct from each other and from p, (ii) one q_i is s, and (iii) $1 \leq j_i \leq k$) and p has not previously extracted m then

6: extract m

7: if k = f+1 then

8: if in the entire execution p has extracted exactly one m then

9: deliver(m)

10: else deliver(SF)

11: halt

Termination

```
sender s in round 0:
0:  extract m
sender s in round 1:
1:  broadcast (s,m,1)

Process p in round  k, 1 ≤ k ≤ f+1
2:  if p extracted m  in round k - 1 and p ≠ sender then
4:       broadcast (p,m,k)
5:  if p has executed at least k accept(q_i,m,j_i) 1 ≤ i ≤ k  in
    rounds 1 through k
            (where  (i) q_i distinct from each other and from
            p, (ii) one q_i is s, and (iii) 1 ≤ j_i ≤ k )
      and p has not previously extracted m then
6:          extract m
7:  if k = f+1 then
8:       if in the entire execution p has extracted exactly
                 one m then
9:          deliver(m)
10:      else deliver(SF)
11:      halt
```

In round $f+1$, every correct process delivers either m or SF and then halts

Agreement - 1

```
sender s in round 0:
extract m
sender s in round 1:
broadcast (s,m,1)

Process p in round    k, 1 ≤ k ≤ f+1
if p extracted m  in round k - 1 and p ≠ sender then
    broadcast (p,m,k)
if p has executed at least k accept(q_i,m,j_i) 1 ≤ i ≤ k  in
rounds 1 through k
        (where  (i) q_i distinct from each other and from
        p, (ii) one q_i is s, and (iii) 1 ≤ j_i ≤ k )
   and p has not previously extracted m then
        extract m
if k = f+1 then
    if in the entire execution p has extracted exactly
            one m then
        deliver(m)
    else deliver(SF)
    halt
```

Lemma

If a correct process extracts m, then every correct process eventually extracts m

Proof

Let r be the earliest round in which some correct process extracts m. Let that process be p.

- if r = 0, then p = s and p will execute broadcast(s,m,1) in round 1. By CORRECTNESS, all correct processes will execute **accept** (s,m,1) in round 1 and extract m
- if r > 0, the sender is faulty. Since p has extracted m in round r, p has accepted at least r triples with properties (i), (ii), and (iii) by round r
  - $r \leq f$ By RELAY, all correct processes will have accepted those r triples by round r + 1
  - p will execute broadcast(p,m,r + 1) in round r + 1
  - By CORRECTNESS, any correct process other than p, $q_1, q_2, \ldots, q_r$ will have accepted r + 1 triples (q_k, m, j_k), $1 \leq j_k \leq r + 1$, by round r + 1
  - $q_1, q_2, \ldots, q_r, p$ are all distinct
  - every correct process other than $q_1, q_2, \ldots, q_r, p$ will extract m
  - p has already extracted m; what about $q_1, q_2, \ldots, q_r$?

Agreement - 2

```
sender s in round 0:
    extract m
sender s in round 1:
    broadcast (s,m,1)

Process p in round   k, 1 ≤ k ≤ f+1
    if p extracted m  in round k - 1 and p ≠ sender then
        broadcast (p,m,k)
    if p has executed at least k accept(qi,m,ji) 1 ≤ i ≤ k  in
    rounds 1 through k
            (where  (i) qi distinct from each other and from
            p, (ii) one qi is s, and (iii) 1 ≤ ji ≤ k )
       and p has not previously extracted m then
            extract m
    if k = f+1 then
        if in the entire execution p has extracted exactly
                one m then
            deliver(m)
        else deliver(SF)
        halt
```

Claim: $q_1, q_2, \ldots, q_r$ are all faulty

- Suppose q_k were correct
- p has accepted (q_k, m, j_k) in round $j_k \leq r$
- By UNFORGEABILITY, q_k executed broadcast (q_k, m, j_k) in round j_k
- q_k extracted m in round $j_{k-1} < r$

CONTRADICTION

- Case 2: r = f + 1
  - Since there are at most f faulty processes, some process q_l in $q_1, q_2, \ldots, q_{f+1}$ is correct
  - By UNFORGEABILITY, q_l executed broadcast (q_l, m, j_l) in round $j_l \leq r$
  - q_l has extracted m in round $j_{l-1} < f + 1$

CONTRADICTION

Validity

```
sender s in round 0:
    extract m
sender s in round 1:
    broadcast (s,m,1)

Process p in round    k, 1 ≤ k ≤ f+1
    if p extracted m  in round k - 1 and p ≠ sender then
        broadcast (p,m,k)
    if p has executed at least k accept(q_i,m,j_i) 1 ≤ i ≤ k  in
    rounds 1 through k
            (where  (i) q_i distinct from each other and from
            p, (ii) one q_i is s, and (iii) 1 ≤ j_i ≤ k )
       and p has not previously extracted m then
            extract m
    if k = f+1 then
        if in the entire execution p has extracted exactly
                one m then
            deliver(m)
        else deliver(SF)
        halt
```

- A correct sender executes broadcast$(s, m, 1)$ in round 1
- By CORRECTNESS, all correct processes execute accept$(s, m, 1)$ in round 1 and extract m
- In order to extract a different message m', a process must execute accept$(s, m', 1)$ in some round $i \leq f + 1$
- By UNFORGEABILITY, and because s is correct, no correct process can extract $m' \neq m$
- All correct processes will deliver m