The Algorithm

Code for process p_i :

```
Initially V={v_i}
To execute propose(v_i)
  round k, 1 ≤ k ≤ f+1
  send {v in V : p_i has not already sent v} to all
  for all j, 0 ≤ j ≤ n-1, j ≠ i do
      receive S_j from p_j
      V:= V U S_j
decide(x) occurs as follows:
  if k = f+1  then
      decide min(V)
```

Termination and Integrity


```
Initially V={v_i}
To execute propose(v_i)
  round k, 1 ≤ k ≤ f+1
  send {v in V : p_i has not already sent v} to all
  for all j, 0 ≤ j ≤ n-1, j ≠ i do
      receive S_j from p_j
      V:= V U S_j
decide(x) occurs as follows:
  if k = f+1  then
      decide min(V)
```


Termination

Every correct process

- reaches round f + 1
- Decides on min(V) --- which is well defined

Integrity

At most one value:

- one decide, and min(V) is unique

Only if it was proposed:

- To be decided upon, must be in V at round f+1
- if value = v_i, then it is proposed in round 1
- else, suppose received in round k. By induction:
- k = 1:
  - by Uniform Integrity of underlying send and receive, it must have been sent in round 1
  - by the protocol and because only crash failures, it must have been proposed
- Induction Hypothesis: all values received up to round k = j have been proposed
- k = j+1
  - sent in round j+1 (Uniform Integrity of send and synchronous model)
  - must have been part of V of sender at end of round j
  - by protocol, must have been received by sender by end of round j
  - by induction hypothesis, must have been proposed

Validity


```
Initially V={v_i}
To execute propose(v_i)
  round k, 1 ≤ k ≤ f+1
  send {v in V : p_i has not already sent v} to all
  for all j, 0 ≤ j ≤ n-1, j ≠ i do
      receive S_j from p_j
      V:= V U S_j
decide(x) occurs as follows:
  if k = f+1  then
      decide min(V)
```


Validity


```
Initially V={v_i}
To execute propose(v_i)
  round k, 1 ≤ k ≤ f+1
  send {v in V : p_i has not already sent v} to all
  for all j, 0 ≤ j ≤ n-1, j ≠ i do
      receive S_j from p_j
      V:= V U S_j
decide(x) occurs as follows:
  if k = f+1  then
      decide min(V)
```


- Suppose every process proposes v^*
- Since only crash model, only v^* can be sent
- By Uniform Integrity of send and receive, only v^* can be received
- By protocol, V={ v^* }
- min(V) = v^*
- decide(v^*)

Agreement

```
Initially V={v_i}

To execute propose(v_i)
  round k, 1 ≤ k ≤ f+1
    send {v in V : p_i has not already sent v} to all
    for all j, 0 ≤ j ≤ n-1, j ≠ i do
      receive S_j from p_j
      V:= V ∪ S_j

decide(x) occurs as follows:
  if k = f+1 then
    decide min(V)
```

Lemma 1

For any r ≥ 1, if a process p receives a value v in round r, then there exists a sequence of processes $p_0, p_1, \dots, p_r$ such that p_0 = v's proponent, $p_r = p$ and in each round $k, 1 \leq k \leq r$, p_{k-1} sends v and p_k receives it. Furthermore, all processes in the sequence are distinct.

Proof

By induction on the length of the sequence

Agreement

```
Initially V={v_i}

To execute propose(v_i)
  round k, 1 ≤ k ≤ f+1
    send {v in V : p_i has not already sent v} to all
    for all j, 0 ≤ j ≤ n-1, j ≠ i do
      receive S_j from p_j
      V:= V ∪ S_j

decide(x) occurs as follows:
  if k = f+1 then
    decide min(V)
```

Agreement

```
Initially V={v_i}

To execute propose(v_i)
  round k, 1 ≤ k ≤ f+1
    send {v in V : p_i has not already sent v} to all
    for all j, 0 ≤ j ≤ n-1, j ≠ i do
      receive S_j from p_j
      V:= V ∪ S_j

decide(x) occurs as follows:
  if k = f+1 then
    decide min(V)
```


Lemma 2:

In every execution, at the end of round f + 1, $V_i = V_j$ for every correct processes p_i and p_j

Agreement follows from Lemma 2, since min is a deterministic function

Agreement

```
Initially V={v_i}

To execute propose(v_i)
  round k, 1 ≤ k ≤ f+1
    send {v in V : p_i has not already sent v} to all
    for all j, 0 ≤ j ≤ n-1, j ≠ i do
      receive S_j from p_j
      V:= V ∪ S_j

decide(x) occurs as follows:
  if k = f+1 then
    decide min(V)
```

Lemma 2:

In every execution, at the end of round f + 1, $V_i = V_j$ for every correct processes p_i and p_j

Agreement follows from Lemma 2, since min is a deterministic function

Proof:

- Show that if a correct process has x in its V at the end of round f + 1, then every correct process has x in its V at the end of round f + 1
- Let r be earliest round x is added to the V of a correct process. Let that process be p
- If r ≤ f, then p sends x in round r + 1 ≤ f + 1; every correct process receives x and adds x to its V in round r + 1
- What if r = f + 1?
- By Lemma 1, there exists a sequence $p_0, \dots, p_{f+1} = p$ of distinct processes
- Consider processes $p_0, \dots, p_f$
- f + 1 processes; only f faulty
- one of $p_0, \dots, p_f$ is correct, and adds x to its V before p does it in round r

CONTRADICTION!

A Lower Bound

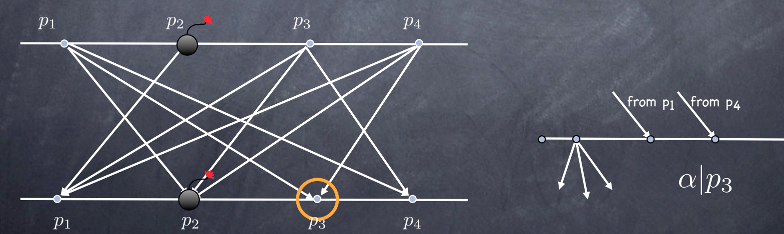
Theorem

There is no algorithm that solves the consensus problem in less than $f+1$ rounds in the presence of f crash failures, if $n \geq f+2$

We consider a special case $(f=1)$ to study proof technique

Views

Let α be an execution. The view of process p_i in α, denoted by $\alpha|p_i$, is the subsequence of computation and message receive events that occur in p_i together with the state of p_i in the initial configuration of α



Similarity

Definition Let α_1 and α_2 be two executions of consensus and let p_i be a correct process in both α_1 and α_2. Execution α_1 is similar to execution α_2 with respect to p_i, denoted $\alpha_1 \sim_{p_i} \alpha_2$ if $\alpha_1|p_i = \alpha_2|p_i$

Similarity

Definition Let α_1 and α_2 be two executions of consensus and let p_i be a correct process in both α_1 and α_2. Execution α_1 is similar to execution α_2 with respect to p_i, denoted $\alpha_1 \sim_{p_i} \alpha_2$ if $\alpha_1|p_i = \alpha_2|p_i$

Note If $\alpha_1 \sim_{p_i} \alpha_2$ then p_i decides the same value in both executions

Similarity

Definition Let α_1 and α_2 be two executions of consensus and let p_i be a correct process in both α_1 and α_2. Execution α_1 is similar to execution α_2 with respect to p_i, denoted $\alpha_1 \sim_{p_i} \alpha_2$ if $\alpha_1|p_i = \alpha_2|p_i$

Note If $\alpha_1 \sim_{p_i} \alpha_2$ then p_i decides the same value in both executions

Lemma If $\alpha_1 \sim_{p_i} \alpha_2$ and p_i is correct, then dec(α_1) = dec(α_2)

Similarity

Definition Let α_1 and α_2 be two executions of consensus and let p_i be a correct process in both α_1 and α_2. Execution α_1 is similar to execution α_2 with respect to p_i, denoted $\alpha_1 \sim_{p_i} \alpha_2$ if $\alpha_1|p_i = \alpha_2|p_i$

Note If $\alpha_1 \sim_{p_i} \alpha_2$ then p_i decides the same value in both executions

Lemma If $\alpha_1 \sim_{p_i} \alpha_2$ and p_i is correct, then dec(α_1) = dec(α_2)

The transitive closure of $\alpha_1 \sim_{p_i} \alpha_2$ is denoted $\alpha_1 \approx \alpha_2$.
We say that $\alpha_1 \approx \alpha_2$ if there exist executions $\beta_1, \beta_2, \ldots, \beta_{k+1}$ such that

$$\alpha_1 = \beta_1 \sim_{p_{i_1}} \beta_2 \sim_{p_{i_2}} \ldots, \sim_{p_{i_k}} \beta_{k+1} = \alpha_2$$

Similarity

Definition Let α_1 and α_2 be two executions of consensus and let p_i be a correct process in both α_1 and α_2. Execution α_1 is similar to execution α_2 with respect to p_i, denoted $\alpha_1 \sim_{p_i} \alpha_2$ if $\alpha_1|p_i = \alpha_2|p_i$

Note If $\alpha_1 \sim_{p_i} \alpha_2$ then p_i decides the same value in both executions

Lemma If $\alpha_1 \sim_{p_i} \alpha_2$ and p_i is correct, then dec(α_1) = dec(α_2)

The transitive closure of $\alpha_1 \sim_{p_i} \alpha_2$ is denoted $\alpha_1 \approx \alpha_2$.

We say that $\alpha_1 \approx \alpha_2$ if there exist executions $\beta_1, \beta_2, \ldots, \beta_{k+1}$ such that

$$\alpha_1 = \beta_1 \sim_{p_{i_1}} \beta_2 \sim_{p_{i_2}} \ldots, \sim_{p_{i_k}} \beta_{k+1} = \alpha_2$$

Lemma If $\alpha_1 \approx \alpha_2$ then dec(α_1) = dec(α_2)

Single-Failure Case

There is no algorithm that solves the consensus problem in less than two rounds in the presence of one crash failure, if $n \geq 3$

The Idea

By contradiction

- Consider a one-round execution in which each process proposes 0. What is the decision value?
- Consider another one-round execution in which each process proposes 1. What is the decision value?
- Show that there is a chain of similar executions that relate the two executions.

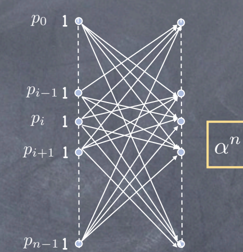
So what?

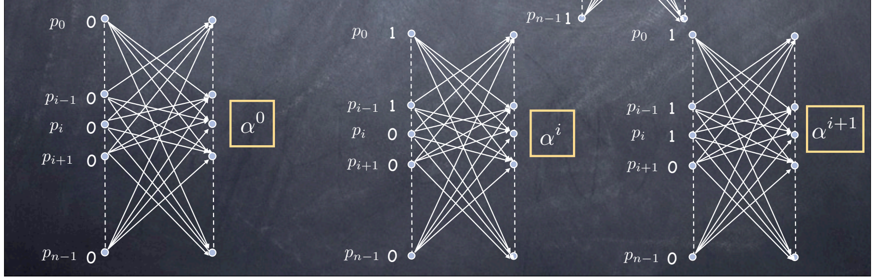
α^is

Definition

α^i is the execution of the algorithm in which

- no failures occur
- processes $p_0, \ldots, p_{i-1}$ propose 1



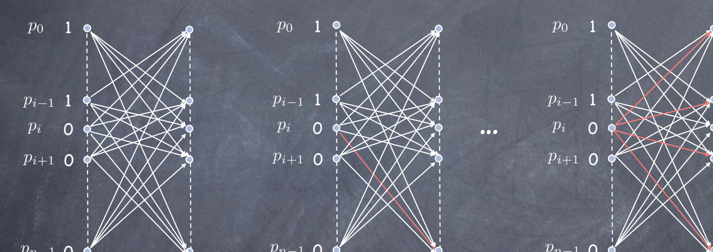


Adjacent α^is are similar!

Starting from α^i, we build a set of executions α^i_j where $0 \le j \le n-1$ as follows:

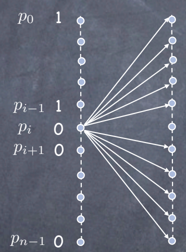
α^i_j is obtained from α^i after removing the messages that p_i sends to the j-th highest numbered processors (excluding itself)

The executions



α^i_0 α^i_1 α^i_{n-1}

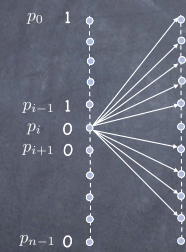
Indistinguishability



α^i

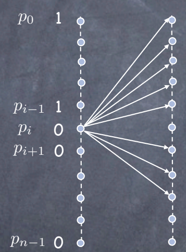
α_0^i

Indistinguishability



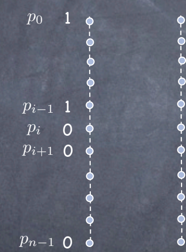
α^i
$\approx$
α_1^i

Indistinguishability



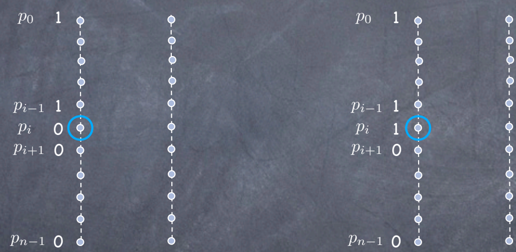
α^i
$\approx$
α_2^i

Indistinguishability



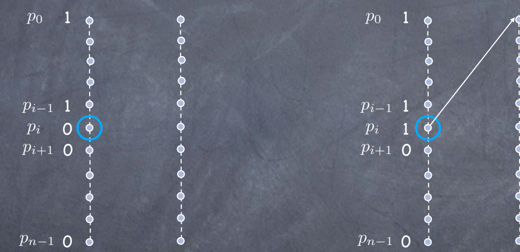
α^i
$\approx$
α_{n-1}^i

Indistinguishability



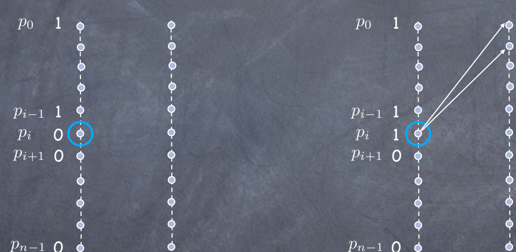
$$\alpha^i \approx \alpha^i_{n-1} \approx \beta^i_{n-1}$$

Indistinguishability



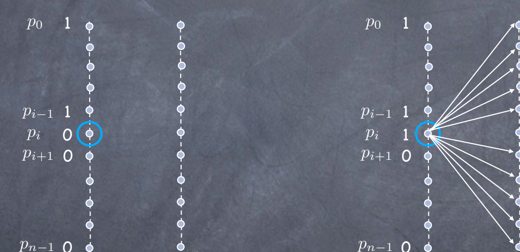
$$\alpha^i \approx \alpha^i_{n-1} \approx \beta^i_{n-2}$$

Indistinguishability



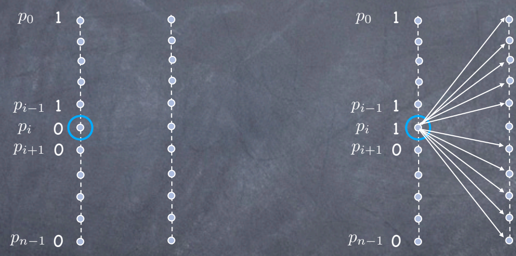
$$\alpha^i \approx \alpha^i_{n-1} \approx \beta^i_{n-3}$$

Indistinguishability



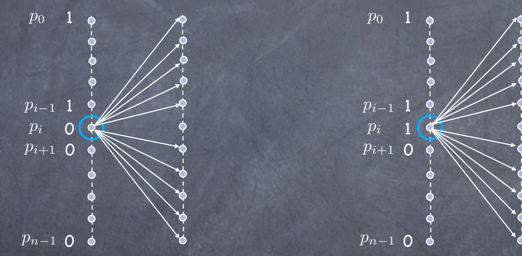
$$\alpha^i \approx \alpha^i_{n-1} \approx \beta^i_0$$

Indistinguishability



α^i $\wr\wr$ α^i_{n-1} $\approx$ α^{i+1} $\wr\wr$ β^i_0

Indistinguishability



$\alpha^i \approx \alpha^{i+1}$

Terminating Reliable Broadcast

Termination Every correct process eventually delivers some message

Validity If the sender is correct and broadcasts a message m, then all correct processes eventually deliver m

Agreement If a correct process delivers a message m, then all correct processes eventually deliver m

Integrity Every correct process delivers at most one message, and if it delivers $m \neq$ SF, then some process must have broadcast m

TRB for benign failures

```
Sender in round 1:
    send m to all

Process p in round   k, 1 ≤ k ≤ f+1
    if delivered m in round k-1 and p ≠ sender then
        send m to all
        halt
    receive round k messages
    if received m then
        deliver(m)
        if  k = f+1 then halt
    else if k = f+1
        deliver(SF)
        halt
```

Terminates in $f + 1$ rounds

How can we do better?

- find a protocol whose round complexity is proportional to t –the number of failures that actually occurred–rather than to f –the max number of failures that may occur

Early stopping: the idea

- Suppose processes can detect the set of processes that have failed by the end of round i
- Call that set $faulty(p, i)$
- If $|faulty(p, i)| < i$ there can be no active dangerous chains, and p can safely deliver SF

Early Stopping: The Protocol

Let $|faulty(p, k)|$ be the set of processes that have failed to send a message to p in any round 1...k

```
if p = sender then value := m else value:= ?
```

Process p in round k, 1 ≤ k ≤ f+1

```
send value to all
if value ≠ ? then halt
receive round k values from all
|faulty(p,k)| := |faulty(p,k − 1)| ∪ {q | p received no value from q in round k}
if received value v ≠ ? then
    value := v
    deliver(value)
else if k = f+1 or |faulty(p,k)| < k then
    value := SF
deliver(value)
    if k = f+1 then halt
```

Termination

Let |faulty(p,k)| be the set of processes that have failed to send a message to p in any round 1...k

```
if p = sender then value := m else value:= ?
```

Process p in round k, 1 ≤ k ≤ f+1

```
send value to all
if value ≠ ? then halt
receive round k values from all
|faulty(p,k)| := |faulty(p,k - 1)|U {q | p
 received no value from q in round k}
if received value v ≠ ? then
    value := v
    deliver(value)
else if k = f+1 or |faulty(p,k)| < k then
    value := SF
    deliver(value)
    if k = f+1 then halt
```

Termination

Let |faulty(p,k)| be the set of processes that have failed to send a message to p in any round 1...k

```
if p = sender then value := m else value:= ?
```

Process p in round k, 1 ≤ k ≤ f+1

```
send value to all
if value ≠ ? then halt
receive round k values from all
|faulty(p,k)| := |faulty(p,k - 1)|U {q | p
 received no value from q in round k}
if received value v ≠ ? then
    value := v
    deliver(value)
else if k = f+1 or |faulty(p,k)| < k then
    value := SF
    deliver(value)
    if k = f+1 then halt
```

- If in any round a process receives a value, then it delivers the value in that round
- If a process has received only "?" for $f+1$ rounds, then it delivers SF in round $f+1$

Validity

Let |faulty(p,k)| be the set of processes that have failed to send a message to p in any round 1...k

```
if p = sender then value := m  else value:= ?

Process p in round  k, 1 ≤ k ≤ f+1

send value to all
if value ≠ ?  then halt
receive round k values from all
|faulty(p,k)| := |faulty(p,k – 1)|U {q | p
 received no value from q in round k}
if received value v ≠ ?  then
    value := v
    deliver(value)
else if k = f+1 or |faulty(p,k)| < k then
    value := SF
    deliver(value)
    if k = f+1 then halt
```


Validity

Let |faulty(p,k)| be the set of processes that have failed to send a message to p in any round 1...k

```
if p = sender then value := m  else value:= ?

Process p in round  k, 1 ≤ k ≤ f+1

send value to all
if value ≠ ?  then halt
receive round k values from all
|faulty(p,k)| := |faulty(p,k – 1)|U {q | p
 received no value from q in round k}
if received value v ≠ ?  then
    value := v
    deliver(value)
else if k = f+1 or |faulty(p,k)| < k then
    value := SF
    deliver(value)
    if k = f+1 then halt
```


- If the sender is correct then it sends m to all in round 1
- By Validity of the underlying send and receive, every correct process will receive m by the end of round 1
- By the protocol, every correct process will deliver m by the end of round 1

Agreement – 1

Let |faulty(p,k)| be the set of processes that have failed to send a message to p in any round 1...k

```
if p = sender then value := m  else value:= ?

Process p in round  k, 1 ≤ k ≤ f+1

send value to all
if value ≠ ?  then halt
receive round k values from all
|faulty(p,k)| := |faulty(p,k – 1)|U {q | p
 received no value from q in round k}
if received value v ≠ ?  then
    value := v
    deliver(value)
else if k = f+1 or |faulty(p,k)| < k then
    value := SF
    deliver(value)
    if k = f+1 then halt
```


Lemma 1:

For any r ≥ 1, if a process p delivers m ≠ SF in round r, then there exists a sequence of processes $p_0, p_1, \ldots, p_r$ such that p_0 = sender, $p_r = p$, and in each round k, 1 ≤ k ≤ r, p_{k-1} sent m and p_k received it. Furthermore, all processes in the sequence are distinct, unless r = 1 and $p_0 = p_1 =$ sender

Lemma 2:

For any r ≥ 1, if a process p sets value to SF in round r, then there exist some j ≤ r and a sequence of distinct processes $q_j, q_{j+1}, \ldots, q_r = p$ such that q_j only receives "?" in rounds 1 to j, $|faulty(q_j, j)| < j$, and in each round k, j + 1 ≤ k ≤ r, q_{k-1} sends SF to q_k and q_k receives SF

Agreement – 2

Let |faulty(p,k)| be the set of processes that have failed to send a message to p in any round 1...k

```
if p = sender then value := m  else value:= ?

Process p in round  k, 1 ≤ k ≤ f+1

send value to all
if value ≠ ?  then halt
receive round k values from all
|faulty(p,k)| := |faulty(p,k – 1)|U {q | p
 received no value from q in round k}
if received value v ≠ ?  then
    value := v
    deliver(value)
else if k = f+1 or |faulty(p,k)| < k then
    value := SF
    deliver(value)
    if k = f+1 then halt
```


Lemma 3:

It is impossible for p and q, not necessarily correct or distinct, to set value in the same round r to m and SF, respectively

Agreement - 2

Let |faulty(p,k)| be the set of processes that have failed to send a message to p in any round 1...k

```
if p = sender then value := m else value:= ?

Process p in round k, 1 ≤ k ≤ f+1

send value to all
if value ≠ ? then halt
receive round k values from all
|faulty(p,k)| := |faulty(p,k - 1)|U {q | p
 received no value from q in round k}
if received value v ≠ ? then
    value := v
    deliver(value)
else if k = f+1 or |faulty(p,k)| < k then
    value := SF
    deliver(value)
    if k = f+1 then halt
```

Lemma 3:

It is impossible for p and q, not necessarily correct or distinct, to set value in the same round r to m and SF, respectively

Proof

By contradiction

Suppose p sets value = m and q sets value = SF

By Lemmas 1 and 2 there exist

$$p_0, \ldots, p_r$$

$$q_j, \ldots, q_r$$

with the appropriate characteristics

Since q_j did not receive m from process p_{k-1} 1 ≤ k ≤ j in round k

q_j must conclude that $p_0, \ldots, p_{j-1}$ are all faulty processes

But then, $|faulty(q_j, j)| \geq j$

CONTRADICTION

Agreement - 3

Let |faulty(p,k)| be the set of processes that have failed to send a message to p in any round 1...k

```
if p = sender then value := m else value:= ?

Process p in round k, 1 ≤ k ≤ f+1

send value to all
if value ≠ ? then halt
receive round k values from all
|faulty(p,k)| := |faulty(p,k - 1)|U {q | p
 received no value from q in round k}
if received value v ≠ ? then
    value := v
    deliver(value)
else if k = f+1 or |faulty(p,k)| < k then
    value := SF
    deliver(value)
    if k = f+1 then halt
```


Agreement - 3

Let |faulty(p,k)| be the set of processes that have failed to send a message to p in any round 1...k

```
if p = sender then value := m else value:= ?

Process p in round k, 1 ≤ k ≤ f+1

send value to all
if value ≠ ? then halt
receive round k values from all
|faulty(p,k)| := |faulty(p,k - 1)|U {q | p
 received no value from q in round k}
if received value v ≠ ? then
    value := v
    deliver(value)
else if k = f+1 or |faulty(p,k)| < k then
    value := SF
    deliver(value)
    if k = f+1 then halt
```


Proof

If no correct process ever receives m, then every correct process delivers SF in round f + 1

Let r be the earliest round in which a correct process delivers value ≠ SF

r ≤ f

- By Lemma 3, no (correct) process can set value differently in round r
- In round r + 1 ≤ f + 1, that correct process sends its value to all
- Every correct process receives and delivers the value in round r + 1 ≤ f + 1

r = f + 1

- By Lemma 1, there exists a sequence $p_0, \ldots, p_{f+1} = p_r$ of distinct processes
- Consider processes $p_0, \ldots, p_f$
    - f + 1 processes; only f faulty
    - one of $p_0, \ldots, p_f$ is correct-- let it be p_c
    - To send v in round c + 1, p_c must have set its value to v and delivered v in round c < r

CONTRADICTION

Integrity

Let |faulty(p,k)| be the set of processes that have failed to send a message to p in any round 1...k

```
if p = sender then value := m else value:= ?

Process p in round k, 1 ≤ k ≤ f+1

send value to all
if value ≠ ? then halt
receive round k values from all
|faulty(p,k)| := |faulty(p,k - 1)|U {q | p
 received no value from q in round k}
if received value v ≠ ? then
    value := v
    deliver(value)
else if k = f+1 or |faulty(p,k)| < k then
    value := SF
    deliver(value)
    if k = f+1 then halt
```

Integrity

Let |faulty(p,k)| be the set of processes that have failed to send a message to p in any round 1...k

```
if p = sender then value := m  else value:= ?

Process p in round  k, 1 ≤ k ≤ f+1

send value to all
if value ≠ ?  then halt
receive round k values from all
|faulty(p,k)| := |faulty(p,k - 1)|U {q | p
 received no value from q in round k}
if received value v ≠ ?  then
    value := v
    deliver(value)
else if k = f+1 or |faulty(p,k)| < k then
    value := SF
    deliver(value)
    if k = f+1 then halt
```

- At most one m
  - Failures are benign, and a process executes at most one deliver event before halting
- If m ≠ SF, only if m was broadcast
  - From Lemma 1 in the proof of Agreement