Terminating Reliable Broadcast

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

Termination Every correct process eventually delivers some message

TRB for benign failures

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

Process p in round    k, 1 ≤ k ≤ f+1
1:  if delivered m in round k-1 then
2:    if p ≠ sender then
3:      send m to all
4:    halt
5:  receive round k messages
6:  if received m then
7:    deliver(m)
8:    if  k = f+1 then halt
9:  else if k = f+1
10:   deliver(SF)
11:   halt
```

TRB for benign failures

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

Process p in round    k, 1 ≤ k ≤ f+1
1:  if delivered m in round k-1 then
2:    if p ≠ sender then
3:      send m to all
4:    halt
5:  receive round k messages
6:  if received m then
7:    deliver(m)
8:    if  k = f+1 then halt
9:  else if k = f+1
10:   deliver(SF)
11:   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

$faulty(p,k) \equiv$ set of processes that failed to send a message to p in some round

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

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

2: send value to all
3: if delivered in round $k-1$ then halt
4: receive round k values from all
5: $faulty(p,k) := faulty(p,k-1) \cup \{q \mid p$ received no value from q in round $k\}$
6: if received value $v \neq ?$ then
7: value := v
8: deliver value
9: if p = sender then value := ?
10: else if $k = f+1$ or $|faulty(p,k)| < k$ then
11: value := SF
12: deliver value
13: 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 $1, \ldots, k$

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

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

2: send value to all
3: if delivered in round $k-1$ then halt
4: receive round k values from all
5: $faulty(p,k) := faulty(p,k-1) \cup \{q \mid p$ received no value from q in round $k\}$
6: if received value $v \neq ?$ then
7: value := v
8: deliver value
9: if p = sender then value := ?
10: else if $k = f+1$ or $|faulty(p,k)| < k$ then
11: value := SF
12: deliver value
13: 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 $1, \ldots, k$

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

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

2: send value to all
3: if delivered in round $k-1$ then halt
4: receive round k values from all
5: $faulty(p,k) := faulty(p,k-1) \cup \{q \mid p$ received no value from q in round $k\}$
6: if received value $v \neq ?$ then
7: value := v
8: deliver value
9: if p = sender then value := ?
10: else if $k = f+1$ or $|faulty(p,k)| < k$ then
11: value := SF
12: deliver value
13: 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 $1, \ldots, k$

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

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

2: send value to all
3: if delivered in round $k-1$ then halt
4: receive round k values from all
5: $faulty(p,k) := faulty(p,k-1) \cup \{q \mid p$ received no value from q in round $k\}$
6: if received value $v \neq ?$ then
7: value := v
8: deliver value
9: if p = sender then value := ?
10: else if $k = f+1$ or $|faulty(p,k)| < k$ then
11: value := SF
12: deliver value
13: 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 $1,\ldots,k$

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

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

2: send value to all
3: if delivered in round $k-1$ then halt
4: receive round k values from all
5: $faulty(p,k) := faulty(p,k-1) \cup \{q \mid p$ received no value from q in round $k\}$
6: if received value $v \neq ?$ then
7: value := v
8: deliver value
9: if p = sender then value := ?
10: else if $k = f+1$ or $|faulty(p,k)| < k$ then
11: value := SF
12: deliver value
13: 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 $1,\ldots,k$

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

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

2: send value to all
3: if delivered in round $k-1$ then halt
4: receive round k values from all
5: $faulty(p,k) := faulty(p,k-1) \cup \{q \mid p$ received no value from q in round $k\}$
6: if received value $v \neq ?$ then
7: value := v
8: deliver value
9: if p = sender then value := ?
10: else if $k = f+1$ or $|faulty(p,k)| < k$ then
11: value := SF
12: deliver value
13: if $k = f+1$ then halt

Lemma 1:

For any $r \ge 1$, if a process p delivers $m \neq$ 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 $k, 1 \le k \le r$, 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 $r \ge 1$, if a process p sets value to SF in round r, then there exist some $j \le r$ 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 $k, j+1 \le k \le r$, 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 $1,\ldots,k$

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

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

2: send value to all
3: if delivered in round $k-1$ then halt
4: receive round k values from all
5: $faulty(p,k) := faulty(p,k-1) \cup \{q \mid p$ received no value from q in round $k\}$
6: if received value $v \neq ?$ then
7: value := v
8: deliver value
9: if p = sender then value := ?
10: else if $k = f+1$ or $|faulty(p,k)| < k$ then
11: value := SF
12: deliver value
13: 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 $1,\ldots,k$

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

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

2: send value to all
3: if delivered in round $k-1$ then halt
4: receive round k values from all
5: $faulty(p,k) := faulty(p,k-1) \cup \{q \mid p$ received no value from q in round $k\}$
6: if received value $v \neq ?$ then
7: value := v
8: deliver value
9: if p = sender then value := ?
10: else if $k = f+1$ or $|faulty(p,k)| < k$ then
11: value := SF
12: deliver value
13: 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} $1 \le k \le j$ in round k

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

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

CONTRADICTION

Agreement - 3

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

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

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

2: send value to all
3: if delivered in round $k-1$ then halt
4: receive round k values from all
5: $faulty(p,k) := faulty(p,k-1) \cup \{q \mid p$ received no value from q in round $k\}$
6: if received value $v \neq$? then
7: value := v
8: deliver value
9: if p = sender then value := ?
10: else if $k = f+1$ or $|faulty(p,k)| < k$ then
11: value := SF
12: deliver value
13: 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 $1,\ldots,k$

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

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

2: send value to all
3: if delivered in round $k-1$ then halt
4: receive round k values from all
5: $faulty(p,k) := faulty(p,k-1) \cup \{q \mid p$ received no value from q in round $k\}$
6: if received value $v \neq$? then
7: value := v
8: deliver value
9: if p = sender then value := ?
10: else if $k = f+1$ or $|faulty(p,k)| < k$ then
11: value := SF
12: deliver value
13: 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

$r \le f$

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

$r = f+1$

- 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 $1,\ldots,k$

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

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

2: send value to all
3: if delivered in round $k-1$ then halt
4: receive round k values from all
5: $faulty(p,k) := faulty(p,k-1) \cup \{q \mid p$ received no value from q in round $k\}$
6: if received value $v \neq$? then
7: value := v
8: deliver value
9: if p = sender then value := ?
10: else if $k = f+1$ or $|faulty(p,k)| < k$ then
11: value := SF
12: deliver value
13: 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 $1,\ldots,k$

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

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

2: send value to all
3: if delivered in round $k-1$ then halt
4: receive round k values from all
5: $faulty(p,k) := faulty(p,k-1) \cup \{q \mid p$ received no value from q in round $k\}$
6: if received value $v \neq$? then
7: value := v
8: deliver value
9: if p = sender then value := ?
10: else if $k = f+1$ or $|faulty(p,k)| < k$ then
11: value := SF
12: deliver value
13: 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 $m \neq$ SF, only if m was broadcast
  - From Lemma 1 in the proof of Agreement

A Lower Bound

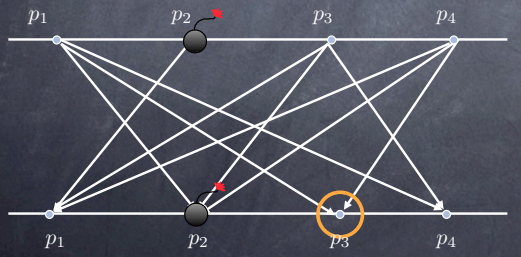
Theorem

There is no algorithm that solves the consensus problem in fewer 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 the proof technique

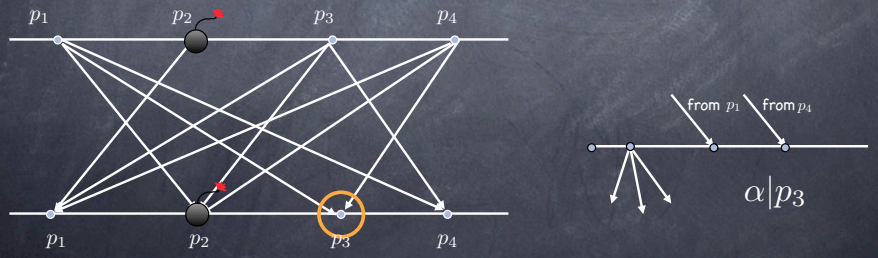
Views

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



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.
α_1 is similar to 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.
α_1 is similar to 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.
α_1 is similar to 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.
α_1 is similar to 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.
α_1 is similar to 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 consensus in fewer 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.

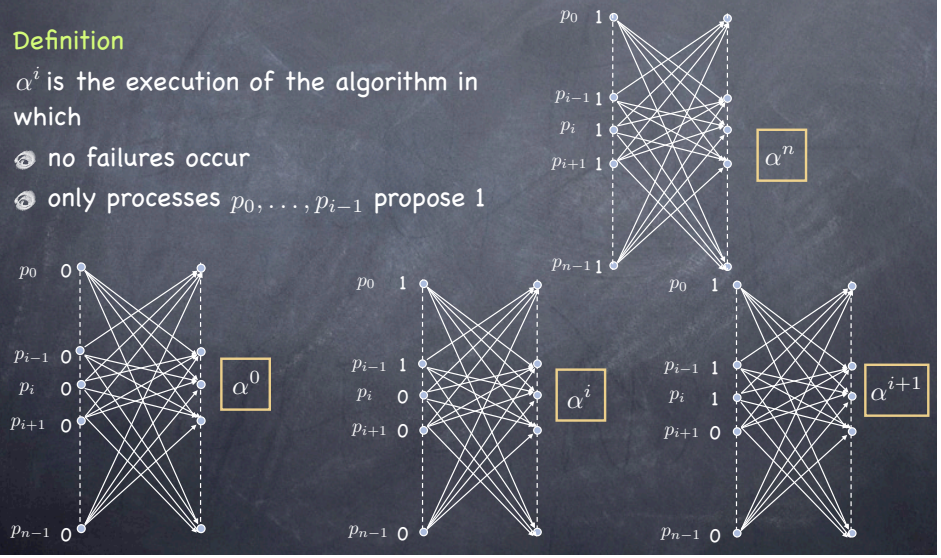
So what?

α^i's

Definition

α^i is the execution of the algorithm in which

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

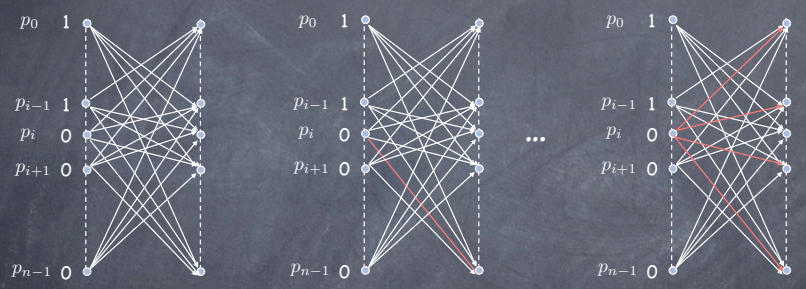


Adjacent α^is are similar!

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

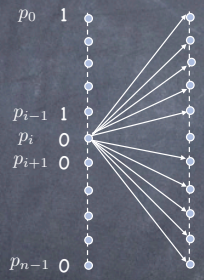
α^i_j is obtained from α^i after removing the messages that p_i sends to the j-th highest numbered processors (excluding itself)

The executions



α_0^i α_1^i α_{n-1}^i

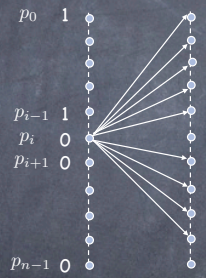
Indistinguishability



α^i

α_0^i

Indistinguishability

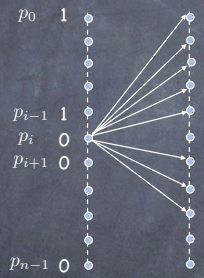


α^i

$\approx$

α_1^i

Indistinguishability

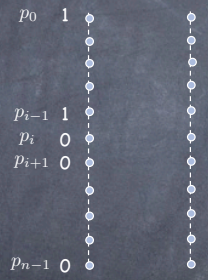


α^i

$\approx$

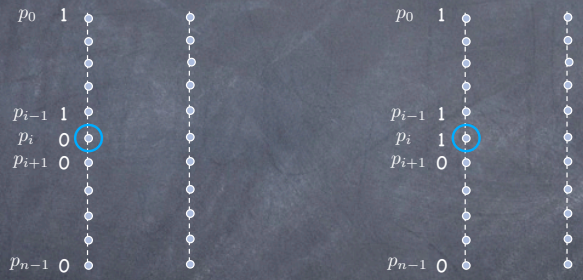
α_2^i

Indistinguishability



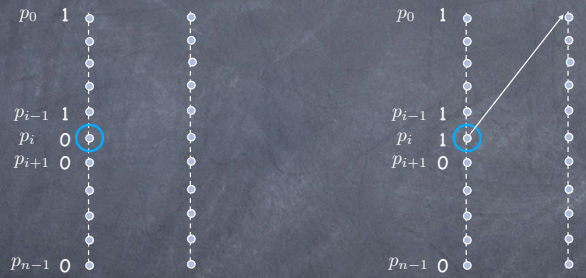
α^i
$\wr\!\wr$
α^i_{n-1}

Indistinguishability



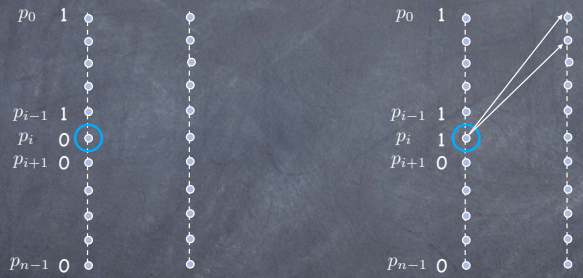
α^i
$\wr\!\wr$
α^i_{n-1} $\approx$ β^i_{n-1}

Indistinguishability



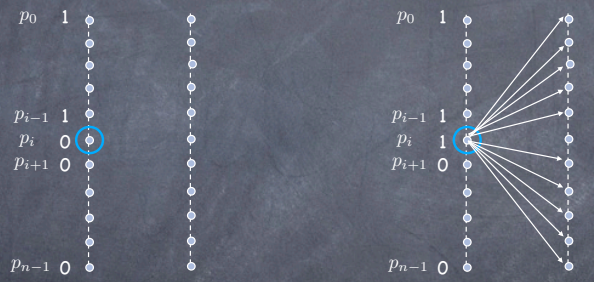
α^i
$\wr\!\wr$
α^i_{n-1} $\approx$ β^i_{n-2}

Indistinguishability



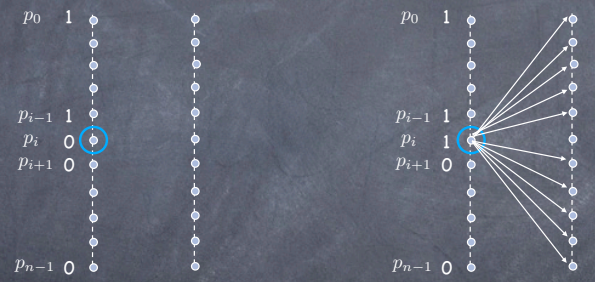
α^i
$\wr\!\wr$
α^i_{n-1} $\approx$ β^i_{n-3}

Indistinguishability



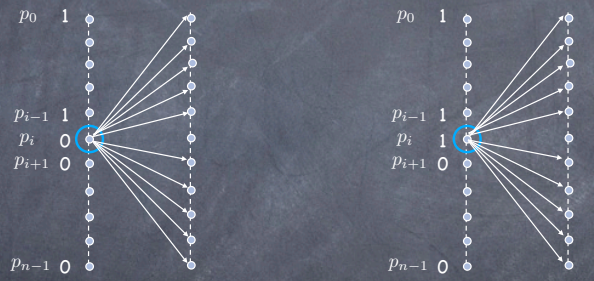
$\alpha^i \rightsquigarrow \alpha^i_{n-1} \approx \beta^i_0$

Indistinguishability



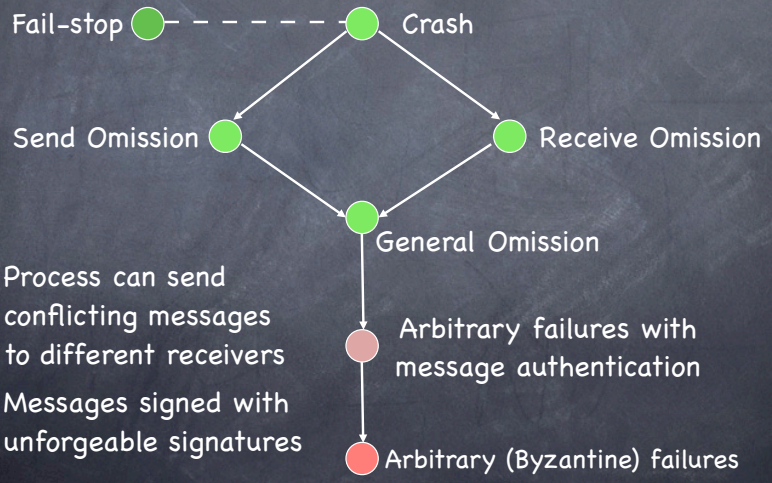
$\alpha^i \rightsquigarrow \alpha^i_{n-1} \approx \alpha^{i+1} \rightsquigarrow \beta^i_0$

Indistinguishability



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

Arbitrary failures with message authentication



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

Valid messages

A valid message m has the following form:

in round 1:

$m : s_{id}$ (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 discards 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, p delivers SF

AFMA: The Protocol

Initialization for process p :
  if p = sender and p wishes to broadcast m then
    extracted := relay := $\{m\}$

Process p in round $k, 1 \leq k \leq f+1$
  for each $s \in$ relay
    send $s : p$ to all
  receive round k messages from all processes
  relay := $\emptyset$
  for each valid message received $s = m : p_1 : p_2 : \ldots : p_k$
    if $m \notin$ extracted then
      extracted := extracted $\cup \{m\}$
      relay := relay $\cup \{s\}$

At the end of round $f+1$
  if $\exists m$ such that extracted = $\{m\}$ then
    deliver m
  else deliver SF

Termination

Initialization for process p :
  if p = sender and p wishes to broadcast m then
    extracted := relay := $\{m\}$

Process p in round $k, 1 \leq k \leq f+1$
  for each $s \in$ relay
    send $s : p$ to all
  receive round k messages from all processes
  relay := $\emptyset$
  for each valid message received $s = m : p_1 : p_2 : \ldots : p_k$
    if $m \notin$ extracted then
      extracted := extracted $\cup \{m\}$
      relay := relay $\cup \{s\}$

At the end of round $f+1$
  if $\exists m$ such that extracted = $\{m\}$ then
    deliver m
  else deliver SF

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

Agreement

Initialization for process p:
  if p = sender and p wishes to broadcast m then
    extracted := relay := $\{m\}$

Process p in round k, $1 \le k \le f+1$
  for each $s \in$ relay
    send $s : p$ to all
  receive round k messages from all processes
  relay := $\emptyset$
  for each valid message received $s = m : p_1 : p_2 : \ldots : p_k$
    if $m \notin$ extracted then
      extracted := extracted $\cup \{m\}$
      relay := relay $\cup \{s\}$

At the end of round $f+1$
    if $\exists m$ such that extracted = $\{m\}$ then
      deliver m
    else deliver SF

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.

- p has received in round r a message
  $$m : p_1 : p_2 : \ldots : p_r$$
- If $r \le f$, p will send a valid message
  $$m : p_1 : p_2 : \ldots : p_r : p$$
  in round $r+1 \le f+1$ and every correct process will extract it in round $r+1 \le f+1$
- What if $r = f+1$?

- Claim: $p_1, p_2, \ldots, p_{f+1}$ are all faulty
  - true for $p_1 = s$
  - Suppose $p_j, 1 < j \le f+1$, were correct
    - p_j signed and relayed message in round j
    - p_j extracted message in round $j-1 < f+1$
    - $f+1$ was supposed to be earliest round where a correct process extracted m

    CONTRADICTION
  - At most f faulty processes
  - CONTRADICTiON

Validity

Initialization for process p:
  if p = sender and p wishes to broadcast m then
    extracted := relay := $\{m\}$

Process p in round k, $1 \le k \le f+1$
  for each $s \in$ relay
    send $s : p$ to all
  receive round k messages from all processes
  relay := $\emptyset$
  for each valid message received $s = m : p_1 : p_2 : \ldots : p_k$
    if $m \notin$ extracted then
      extracted := extracted $\cup \{m\}$
      relay := relay $\cup \{s\}$

At the end of round $f+1$
    if $\exists m$ such that extracted = $\{m\}$ then
      deliver m
    else deliver SF

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

Around FLP in 80 Slides

How can one get around FLP?

Weaken the problem

- Weaken termination
  - use randomization to terminate with arbitrarily high probability
  - guarantee termination only during periods of synchrony
- Weaken agreement
  - ε - agreement
    - real-valued inputs and outputs
    - agreement within real-valued small positive tolerance ε
  - k-set agreement
    - Agreement: In any execution, there is a subset W of the set of input values, |W| =k, s.t. all decision values are in W
    - Validity: In any execution, any decision value for any process is the input value of some process

How can one get around FLP?

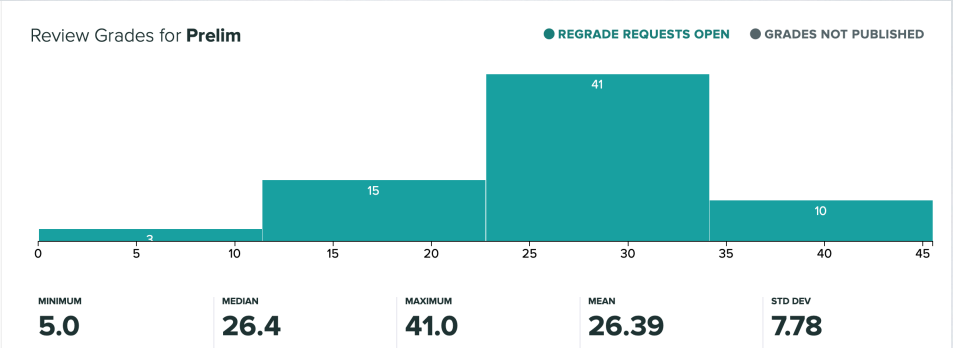
Constrain input values

- Characterize the set of input values for which agreement is possible

Strengthen the system model

- Introduce failure detectors to distinguish between crashed processes and very slow processes

THE MIDTERM



If your score is less than 20:
examine your conscience

If your score is less than 15:
set up an appointment with me

Paxos

It's (Almost) Everywhere!

Production use of Paxos [edit]

- The Petal project from DEC SRC was likely the first system to use Paxos, in this case for widely replicated global information (e.g., which machines are in the system).[20]
- Google uses the Paxos algorithm in their Chubby distributed lock service in order to keep replicas consistent in case of failure. Chubby is used by BigTable which is now in production in Google Analytics and other products.
- The Infinit peer-to-peer file system relies on Paxos to maintain consistency among replicas while allowing for quorums to evolve in size.
- Google Spanner and Megastore use the Paxos algorithm internally.
- The OpenReplica replication service uses Paxos to maintain replicas for an open access system that enables users to create fault-tolerant objects. It provides high performance through concurrent rounds and flexibility through dynamic membership changes.
- IBM supposedly uses the Paxos algorithm in their IBM SAN Volume Controller product to implement a general purpose fault-tolerant virtual machine used to run the configuration and control components of the storage virtualization services offered by the cluster.[citation needed]
- Microsoft uses Paxos in the Autopilot cluster management service from Bing.
- WANdisco have implemented Paxos within their DConE active-active replication technology.[21]
- XtreemFS uses a Paxos-based lease negotiation algorithm for fault-tolerant and consistent replication of file data and metadata.[22]
- Heroku uses Doozerd which implements Paxos for its consistent distributed data store.
- Ceph uses Paxos as part of the monitor processes to agree which OSDs are up and in the cluster.
- The Clustrix distributed SQL database uses Paxos for distributed transaction resolution.
- Neo4j HA graph database implements Paxos, replacing Apache ZooKeeper from v1.9
- VMware NSX Controller uses Paxos-based algorithm within NSX Controller cluster.
- Amazon Web Services uses the Paxos algorithm extensively to power its platform.[23]
- Nutanix implements the Paxos algorithm in Cassandra for metadata.
- Apache Mesos uses Paxos algorithm for its replicated log coordination.
- Windows Fabric used by many of the Azure services make use of the paxos algorithm for replication between nodes in a cluster
- Oracle NoSQL Database leverages Paxos-based automated fail-over election process in the event of a master replica node failure to minimize downtime.[24]

The Part-Time Parliament

- Parliament determines laws by passing sequence of numbered decrees
- Direct democracy: Citizens/ Legislators leave and enter the chamber at arbitrary times
- No centralized records: each legislator carries a ledger recording the approved decrees

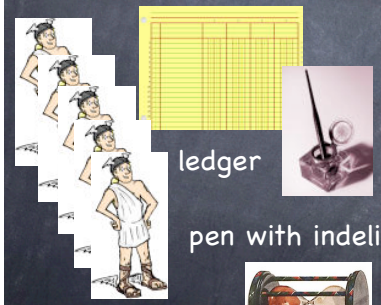
Government 101

- No two ledgers contain contradictory information
- If a majority of legislators are in the Chamber and no one enters or leaves the Chamber for a sufficiently long time, then
  - any decree proposed by a legislator is eventually passed
  - any passed decree appears on the ledger of every legislator

"In a world..."

Political intrigue!

Funky equipment!



ledger

pen with indelible ink

hourglass

messengers!

Λαμπσων

Δἴκστρα

Λισκωφ

Mysterious characters

Σθνειδερ

Σκεεν

Πνυελι

Δωλεφ

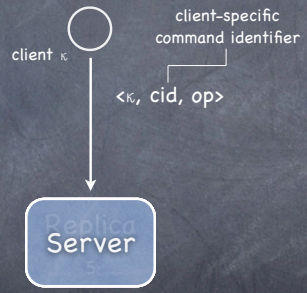
Δφωρκ

and more!

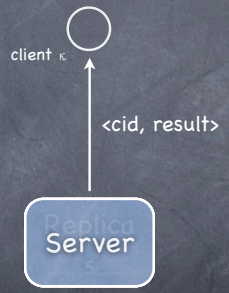
Back to the future

- A protocol for state machine replication in an asynchronous environment that admits crash failures (key ideas already present in earlier work on Viewstamped Replication by Oki and Liskov)
- Messages:
  - between correct endpoints are eventually received
  - can be lost and duplicated, but not corrupted

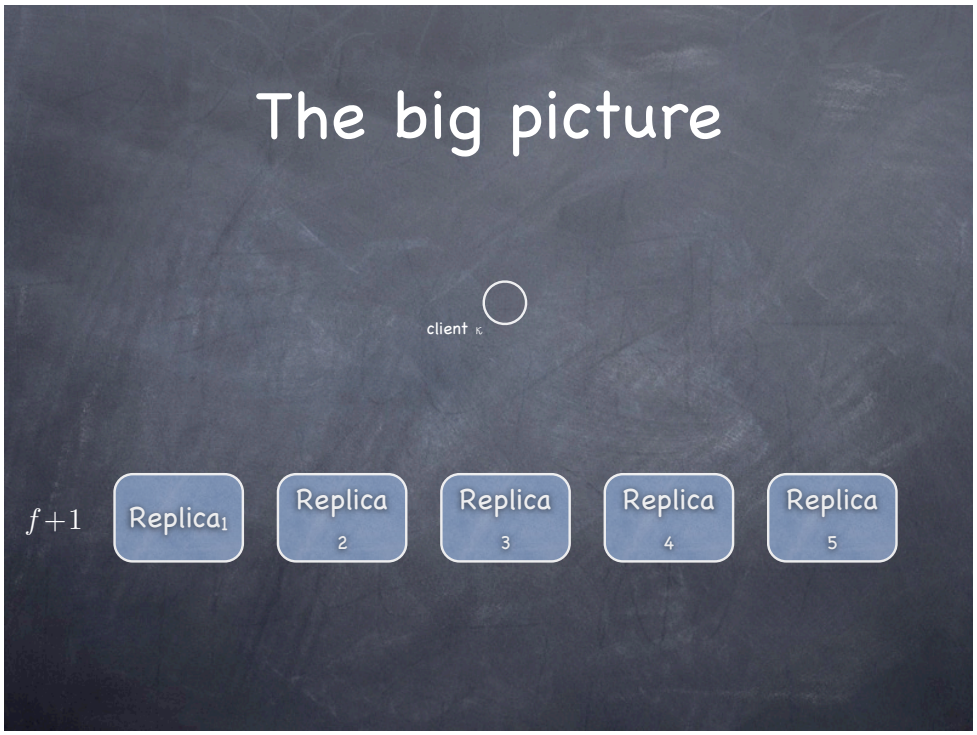
The big picture



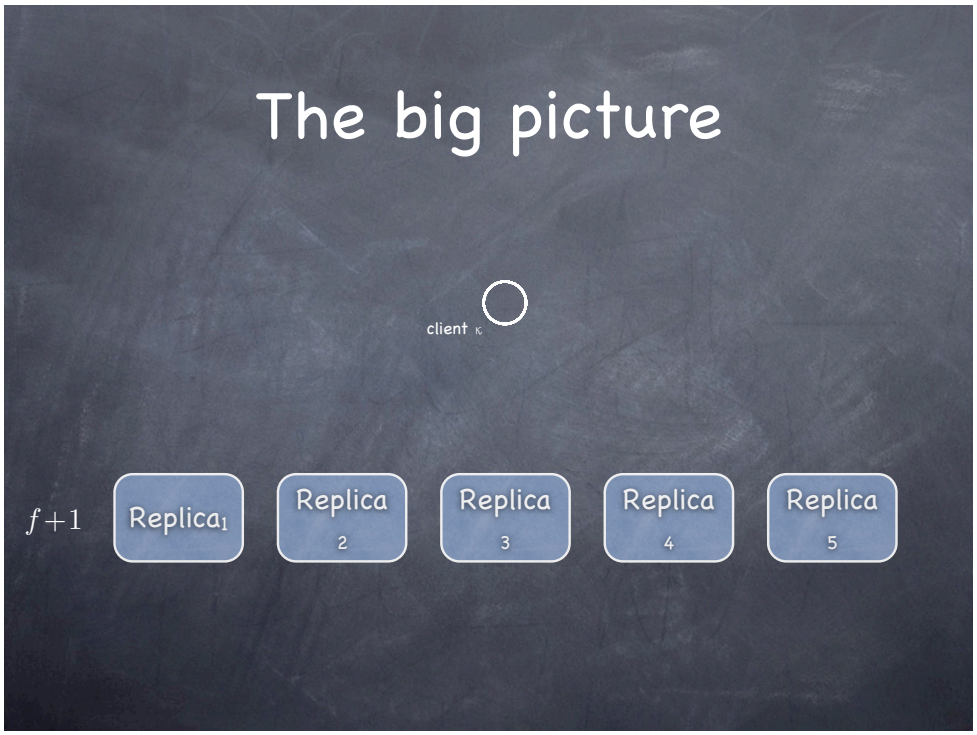
The big picture



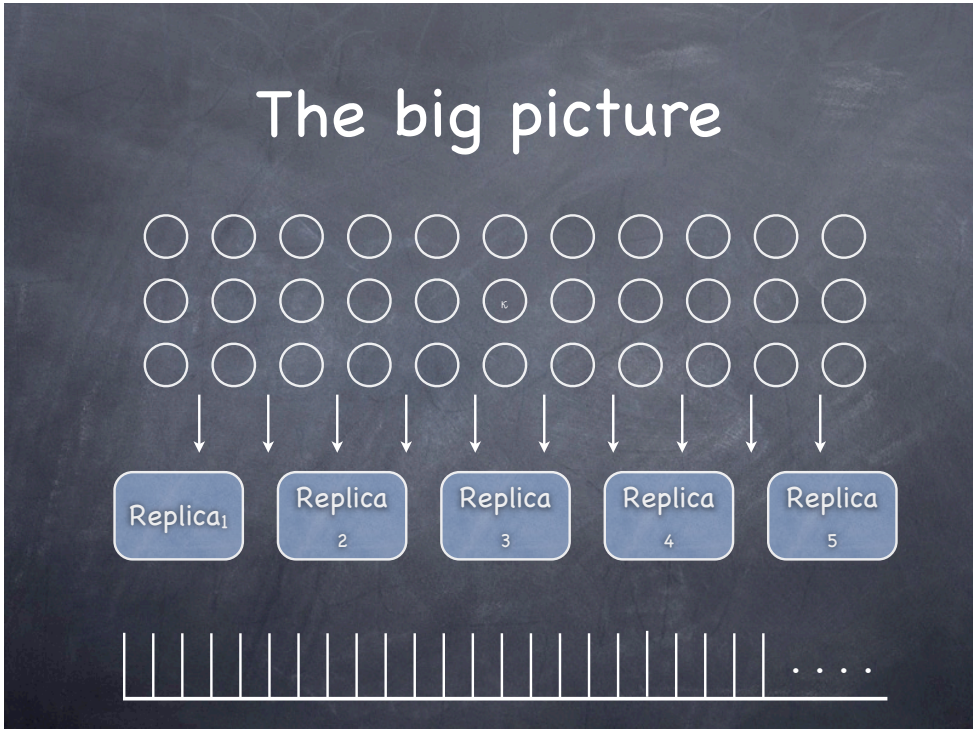
The big picture



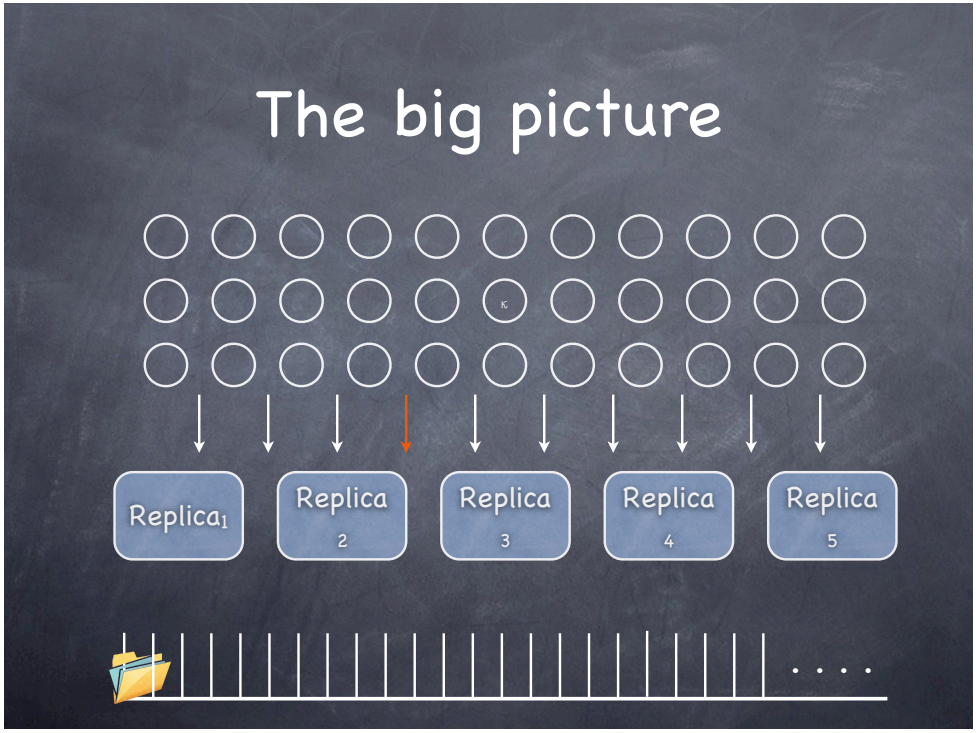
The big picture



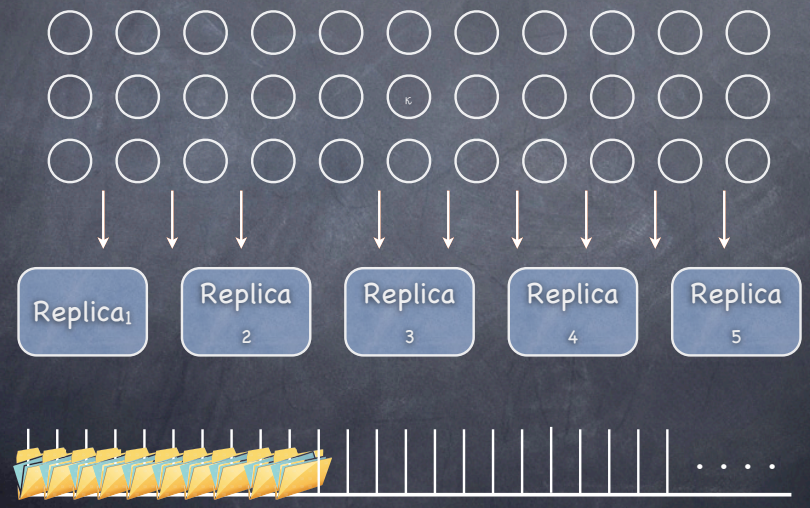
The big picture



The big picture



The big picture



Replicas

- Receive client requests
- Propose command for lowest unused slot to leaders
- Upon decision, execute commands in slot order
- Return result to clients
- Not necessarily identical at any time!

- Each replica ρ maintains four variables:
  - $\rho.state$: the application state
  - $\rho.slot\_num$: next slot for which ρ does not know a decision
  - $\rho.proposals$: set of <slot number, command> pairs for past proposals
  - $\rho.decisions$: set of <slot number, command> pairs for decided slots

Replica

```
process Replica(leaders, initial_state)
  var state := initial_state, slot_num := 1, proposals := ∅; decisions := ∅
  for ever
    switch receive()
      case <request, p> :
        propose(p);
      case <decision, s, p> :
        decisions := decisions ∪ {(s, p)}
        while ∃p' : ⟨slot_num, p'⟩ ∈ decisions do
          if ∃p'' : ⟨slot_num, p''⟩ ∈ proposals ∧ p'' ≠ p' then
            propose(p'');
          end if
          perform(p');
        end while
    end switch
  end for
end process
```

```
function perform(⟨κ, cid, op⟩)
  if ∃s : s < slot_num ∧
        ⟨s, ⟨κ, cid, op⟩⟩ ∈ decisions then
    slot_num := slot_num + 1
  else
    ⟨next, result⟩ := op(state);
    atomic
      state := next;
      slot_num := slot_num + 1
    end atomic
    send(κ, ⟨response, cid, result⟩);
  end if
end function
```

```
function propose(p)
  if ∄s : ⟨s, p⟩ ∈ decisions then
    s' := min{s | s ∈ ℕ⁺ ∧ ∄p' : ⟨s, p'⟩ ∈ proposal ∪ decisions};
    proposals := proposals ∪ {(s', p)};
    ∀λ ∈ leaders : send(λ, ⟨propose, s', p⟩);
  end if
end function
```

R4. For each ρ, the variable $\rho.slot\_num$ never decreases