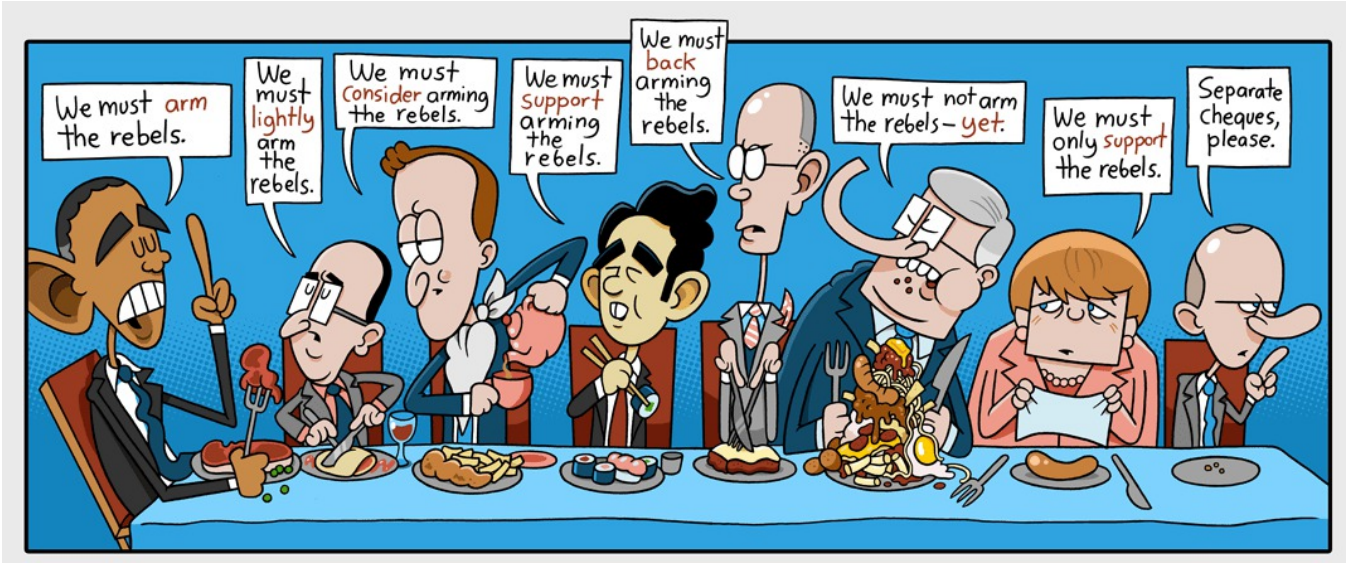
Consensus = agreeing on a result, any… not important which!

Binary consensus: Initially, all processes randomly select 0 or 1.

Eventually, all correct processes must uniformly decide 0 or 1.



Consensus underlies many important problems in distributed computing: termination detection, mutual exclusion, leader election, ...

Fault tolerance

A process may (1) *crash*, i.e., execute no further events, or even (2) be *Byzantine*, meaning that it can perform arbitrary events.

Assumption: The network is *complete*, i.e., there is an undirected channel between each pair of different processes.

So failing processes never make the remaining network disconnected.

Assumption: Crashing of processes can't be observed.

Fault tolerance

A process may (1) *crash*, i.e., execute no further events, or even (2) be *Byzantine*, meaning that it can perform arbitrary events.

Assumption: The network is *complete*, i.e., there is an undirected channel between each pair of different processes.

So failing processes never make the remaining network disconnected.

Assumption: Crashing of processes can't be observed.

Consensus - Assumptions

k-crash consensus: At most k processes may crash.

Validity: If all processes randomly select the same initial value b, then all correct processes decide b.

This excludes trivial solutions where e.g. processes always decide 0.

By validity, each k-crash consensus algorithm with $k \geq 1$ has a bivalent initial configuration that can reach terminal configurations with a decision 0 as well as with a decision 1.

Consensus - Assumptions

k-crash consensus: At most k processes may crash.

Validity: If all processes randomly select the same initial value b, then all correct processes decide b.

This excludes trivial solutions where e.g. processes always decide 0.

By validity, each k-crash consensus algorithm with $k \geq 1$ has a bivalent initial configuration that can reach terminal configurations with a decision 0 as well as with a decision 1.

Consensus - Assumptions

k-crash consensus: At most k processes may crash.

Validity: If all processes randomly select the same initial value b, then all correct processes decide b.

This excludes trivial solutions where e.g. processes always decide 0.

By validity, each k-crash consensus algorithm with $k \geq 1$ has a bivalent initial configuration that can reach terminal configurations with a decision 0 as well as with a decision 1.

(no predetermined outcome from initial configurations)

Impossibility of 1-crash consensus

Theorem: No algorithm for 1-crash consensus always terminates.

Idea: A decision is determined by an event e at a process p.

Since p may crash, after e the other processes must be able to decide without input from p.

Impossibility of 1-crash consensus

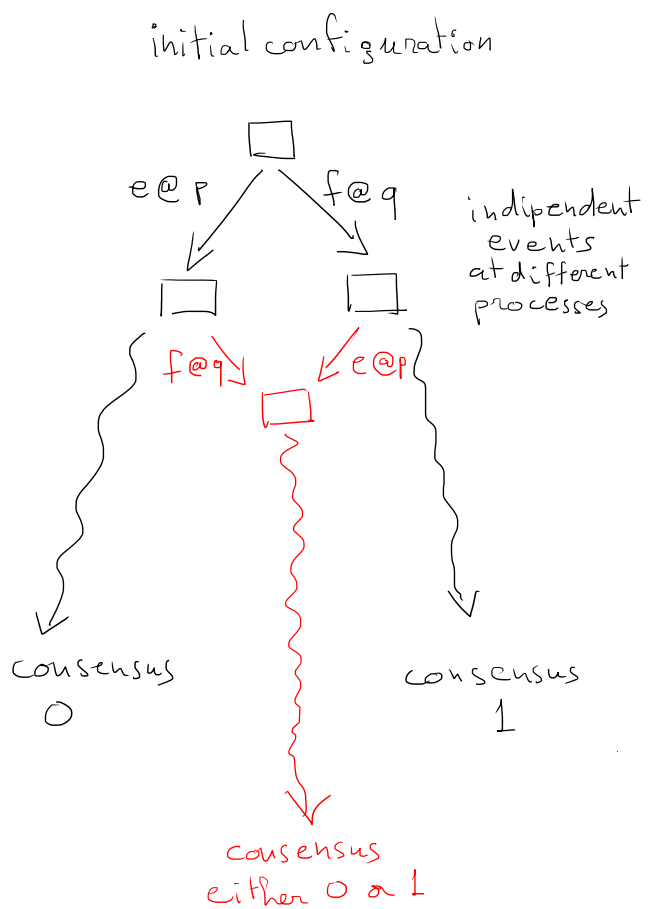
Theorem: No algorithm for 1-crash consensus always terminates.

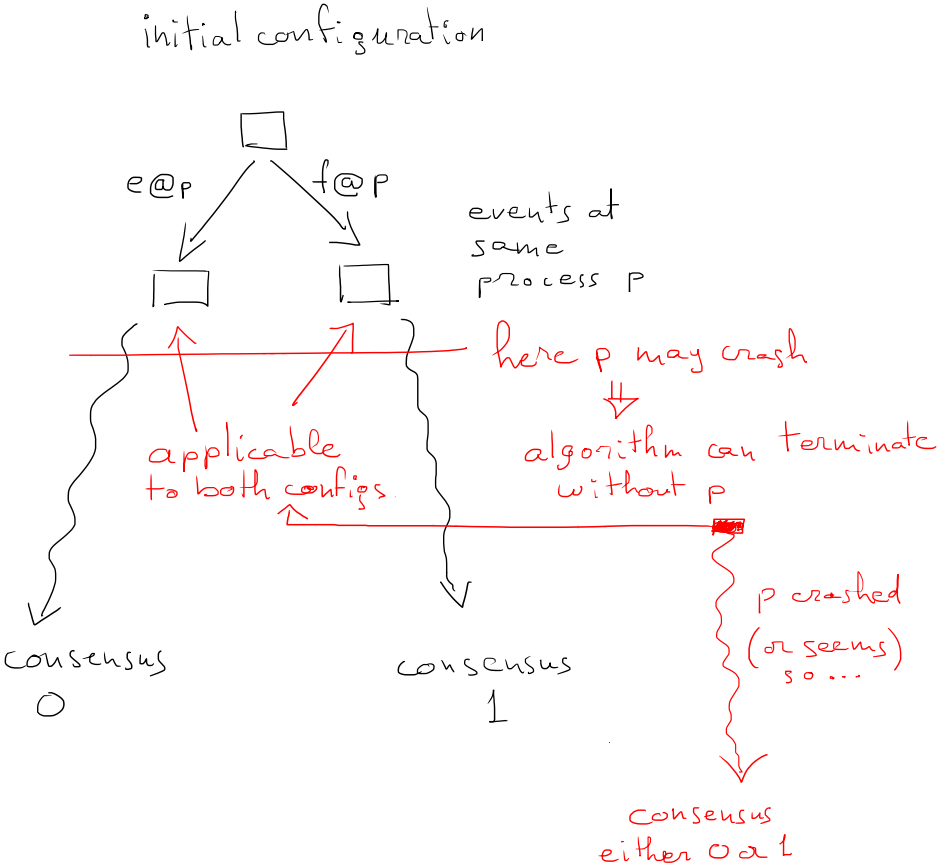
Idea: A decision is determined by an event e at a process p.

Since p may crash, after e the other processes must be able to decide without input from p.

Important underlying assumption: crashes cannot be detected

(e.g. a message can be delayed in a channel for arbitrarily long)





$\Rightarrow \exists$ infinite execution

Las Vegas and Monte Carlo algorithms

A probabilistic algorithm is Las Vegas if:

- the probability that it terminates is greater than zero, and
- all terminal configurations are correct.

It is Monte Carlo if:

- it always terminates, and
- the probability that a terminal configuration is correct is greater than zero.

Question

Give a Monte Carlo algorithm for k-crash consensus for any k.

Question

Give a Monte Carlo algorithm for k-crash consensus for any k.

Answer: Let any process decide for its initial (random) value.

With a (very small) positive probability all correct processes decide for the same value.

Impossibility of $\lceil \frac{N}{2} \rceil$-crash consensus

Theorem: Let $k \geq \frac{N}{2}$. There is no Las Vegas algorithm for k-crash consensus.

Proof: Suppose, toward a contradiction, there is such an algorithm.

Divide the set of processes in S and T, with $|S| = \lfloor \frac{N}{2} \rfloor$ and $|T| = \lceil \frac{N}{2} \rceil$.

Suppose all processes in S select 0 and all processes in T select 1.

Suppose that messages between processes in S and in T are very slow.

Since $k \geq \frac{N}{2}$, at some point the processes in S must assume the processes in T all crashed, and decide 0.

Likewise, at some point the processes in T must assume the processes in S all crashed, and decide 1.

Impossibility of $\lceil \frac{N}{2} \rceil$-crash consensus

Theorem: Let $k \geq \frac{N}{2}$. There is no Las Vegas algorithm for k-crash consensus.

Proof: Suppose, toward a contradiction, there is such an algorithm.

Divide the set of processes in S and T, with $|S| = \lfloor \frac{N}{2} \rfloor$ and $|T| = \lceil \frac{N}{2} \rceil$.

Suppose all processes in S select 0 and all processes in T select 1.

Suppose that messages between processes in S and in T are very slow.

Since $k \geq \frac{N}{2}$, at some point the processes in S must assume the processes in T all crashed, and decide 0.

Likewise, at some point the processes in T must assume the processes in S all crashed, and decide 1.

Impossibility of $\lceil \frac{N}{2} \rceil$-crash consensus

Theorem: Let $k \geq \frac{N}{2}$. There is no Las Vegas algorithm for k-crash consensus.

Proof: Suppose, toward a contradiction, there is such an algorithm.

Divide the set of processes in S and T, with $|S| = \lfloor \frac{N}{2} \rfloor$ and $|T| = \lceil \frac{N}{2} \rceil$.

Suppose all processes in S select 0 and all processes in T select 1.

Suppose that messages between processes in S and in T are very slow.

Since $k \geq \frac{N}{2}$, at some point the processes in S must assume the processes in T all crashed, and decide 0.

Likewise, at some point the processes in T must assume the processes in S all crashed, and decide 1.

Bracha-Toueg crash consensus algorithm

Let $k < \frac{N}{2}$. Initially, each process randomly selects 0 or 1, with weight 1. In round n, at each , undecided p:

- p sends $\langle n, value_p, weight_p \rangle$ to all processes (including itself).
- p waits until $N - k$ messages $\langle n, b, w \rangle$ have arrived.

  (p dismisses/stores messages from earlier/future rounds.)

  If $w > \frac{N}{2}$ for an $\langle n, b, w \rangle$, then $value_p \leftarrow b$. *(This b is unique.)*

  Else, $value_p \leftarrow 0$ if most messages voted 0, $value_p \leftarrow 1$ otherwise.

  $weight_p \leftarrow$ the number of incoming votes for $value_p$ in round n.
- If $w > \frac{N}{2}$ for $> k$ incoming messages $\langle n, b, w \rangle$, then p decides b.

  (Note that $k < N - k$.)

If p decides b, it broadcasts $\langle n + 1, b, N - k \rangle$ and $\langle n + 2, b, N - k \rangle$, and terminates.

Bracha-Toueg crash consensus algorithm

Let $k < \frac{N}{2}$. Initially, each process randomly selects 0 or 1, with weight 1. In round n, at each , undecided p:

- p sends $\langle n, value_p, weight_p \rangle$ to all processes (including itself).
- p waits until $N - k$ messages $\langle n, b, w \rangle$ have arrived.
  (p dismisses/stores messages from earlier/future rounds.)
  If $w > \frac{N}{2}$ for an $\langle n, b, w \rangle$, then $value_p \leftarrow b$. *(This b is unique.)*
  Else, $value_p \leftarrow 0$ if most messages voted 0, $value_p \leftarrow 1$ otherwise.
  $weight_p \leftarrow$ the number of incoming votes for $value_p$ in round n.
- If $w > \frac{N}{2}$ for $> k$ incoming messages $\langle n, b, w \rangle$, then p decides b.
  (Note that $k < N - k$.)

If p decides b, it broadcasts $\langle n + 1, b, N - k \rangle$ and $\langle n + 2, b, N - k \rangle$, and terminates.

Bracha-Toueg crash consensus algorithm

Let $k < \frac{N}{2}$. Initially, each process randomly selects 0 or 1, with weight 1. In round n, at each , undecided p:

- p sends $\langle n, value_p, weight_p \rangle$ to all processes (including itself).

(if p waits for more messages, deadlock may occur)

- p waits until $N - k$ messages $\langle n, b, w \rangle$ have arrived.

  (p dismisses/stores messages from earlier/future rounds.)

  If $w > \frac{N}{2}$ for an $\langle n, b, w \rangle$, then $value_p \leftarrow b$. *(This b is unique.)*

  Else, $value_p \leftarrow 0$ if most messages voted 0, $value_p \leftarrow 1$ otherwise.

  $weight_p \leftarrow$ the number of incoming votes for $value_p$ in round n.

- If $w > \frac{N}{2}$ for $> k$ incoming messages $\langle n, b, w \rangle$, then p decides b.

  (Note that $k < N - k$.)

If p decides b, it broadcasts $\langle n + 1, b, N - k \rangle$ and $\langle n + 2, b, N - k \rangle$, and terminates.

Bracha-Toueg crash consensus algorithm

Let $k < \frac{N}{2}$. Initially, each process randomly selects 0 or 1, with weight 1. In round n, at each , undecided p:

- p sends $\langle n, value_p, weight_p \rangle$ to all processes (including itself).

(if p waits for more messages, deadlock may occur)

- p waits until $N - k$ messages $\langle n, b, w \rangle$ have arrived.

  (p dismisses/stores messages from earlier/future rounds.)

  If $w > \frac{N}{2}$ for an $\langle n, b, w \rangle$, then $value_p \leftarrow b$. *(This b is unique.)*

  Else, $value_p \leftarrow 0$ if most messages voted 0, $value_p \leftarrow 1$ otherwise.

  $weight_p \leftarrow$ the number of incoming votes for $value_p$ in round n.

- If $w > \frac{N}{2}$ for $> k$ incoming messages $\langle n, b, w \rangle$, then p decides b.

  (Note that $k < N - k$.)

If p decides b, it broadcasts $\langle n + 1, b, N - k \rangle$ and $\langle n + 2, b, N - k \rangle$, and terminates.

Bracha-Toueg crash consensus algorithm

Let $k < \frac{N}{2}$. Initially, each process randomly selects 0 or 1, with weight 1. In round n, at each , undecided p:

- p sends $\langle n, value_p, weight_p \rangle$ to all processes (including itself).

(if p waits for more messages, deadlock may occur)

- p waits until $N - k$ messages $\langle n, b, w \rangle$ have arrived.

  (p dismisses/stores messages from earlier/future rounds.)

  If $w > \frac{N}{2}$ for an $\langle n, b, w \rangle$, then $value_p \leftarrow b$. *(This b is unique.)*

  Else, $value_p \leftarrow 0$ if most messages voted 0, $value_p \leftarrow 1$ otherwise.

  $weight_p \leftarrow$ the number of incoming votes for $value_p$ in round n.

- If $w > \frac{N}{2}$ for $> k$ incoming messages $\langle n, b, w \rangle$, then p decides b.

  (Note that $k < N - k$.)

If p decides b, it broadcasts $\langle n + 1, b, N - k \rangle$ and $\langle n + 2, b, N - k \rangle$, and terminates.

Bracha-Toueg crash consensus algorithm

Let $k < \frac{N}{2}$. Initially, each process randomly selects 0 or 1, with weight 1. In round n, at each undecided p:

- p sends $\langle n, value_p, weight_p \rangle$ to all processes (including itself).

(if p waits for more messages, deadlock may occur)

- p waits until $N - k$ messages $\langle n, b, w \rangle$ have arrived.

  (p dismisses/stores messages from earlier/future rounds.)

  If $w > \frac{N}{2}$ for an $\langle n, b, w \rangle$, then $value_p \leftarrow b$. *(This b is unique.)*

  Else, $value_p \leftarrow 0$ if most messages voted 0, $value_p \leftarrow 1$ otherwise.

  $weight_p \leftarrow$ the number of incoming votes for $value_p$ in round n.

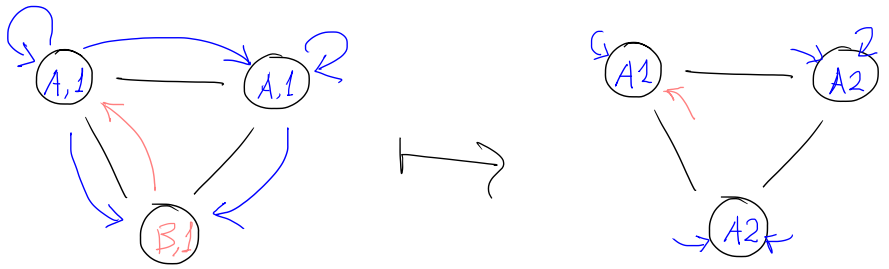
- If $w > \frac{N}{2}$ for $> k$ incoming messages $\langle n, b, w \rangle$, then p decides b.

  (Note that $k < N - k$.)

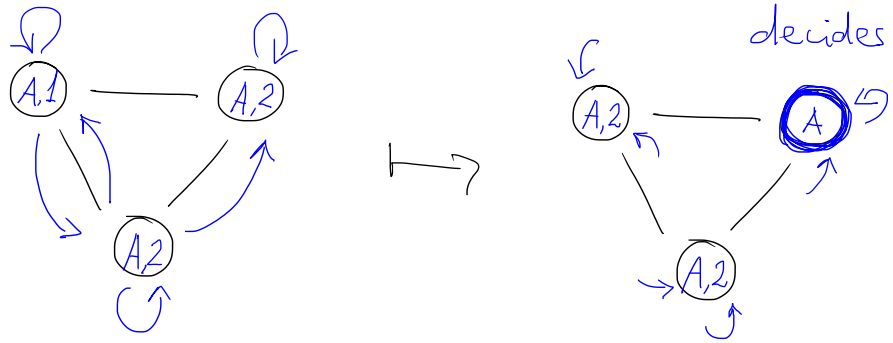
If p decides b, it broadcasts $\langle n + 1, b, N - k \rangle$ and $\langle n + 2, b, N - k \rangle$, and terminates.

N=3 k=1 values = A, B

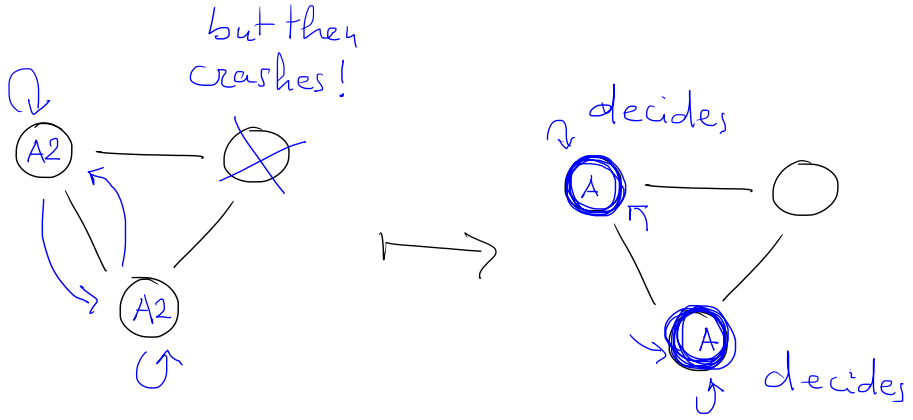
ROUND 1



ROUND 2



ROUND 3



Bracha-Toueg crash consensus algorithm - Correctness

Theorem: Let $k < \frac{N}{2}$. The Bracha-Toueg k-crash consensus algorithm is a Las Vegas algorithm that terminates with probability 1.

Proof (part I): Suppose a process decides b in round n.

Then in round n, $value_q = b$ and $weight_q > \frac{N}{2}$ for $> k$ processes q.

So in round n, each correct process receives a $\langle q, b, w \rangle$ with $w > \frac{N}{2}$.

So in round $n+1$, all correct processes vote b.

So in round $n+2$, all correct processes vote b with weight $N-k$.

Hence, after round $n+2$, all correct processes have decided b.

Concluding, all correct processes decide for the same value.

Bracha-Toueg crash consensus algorithm - Correctness

Theorem: Let $k < \frac{N}{2}$. The Bracha-Toueg k-crash consensus algorithm is a Las Vegas algorithm that terminates with probability 1.

Proof (part I): Suppose a process decides b in round n.

Then in round n, $value_q = b$ and $weight_q > \frac{N}{2}$ for $> k$ processes q.

So in round n, each correct process receives a $\langle q, b, w \rangle$ with $w > \frac{N}{2}$.

So in round $n+1$, all correct processes vote b.

So in round $n+2$, all correct processes vote b with weight $N - k$.

Hence, after round $n+2$, all correct processes have decided b.

Concluding, all correct processes decide for the same value.

Bracha-Toueg crash consensus algorithm - Correctness

Proof (part II): *Assumption*: Scheduling of messages is fair.

Due to fair scheduling, there is a chance $\rho > 0$ that in a round n all processes receive the first $N - k$ messages from the same processes.

After round n, all correct processes have the same value b.

After round $n + 1$, all correct processes have value b with weight $N - k$.

After round $n + 2$, all correct processes have decided b.

Concluding, the algorithm terminates with probability 1.

Exercise: give an example of an infinite computation of Bracha-Toueg