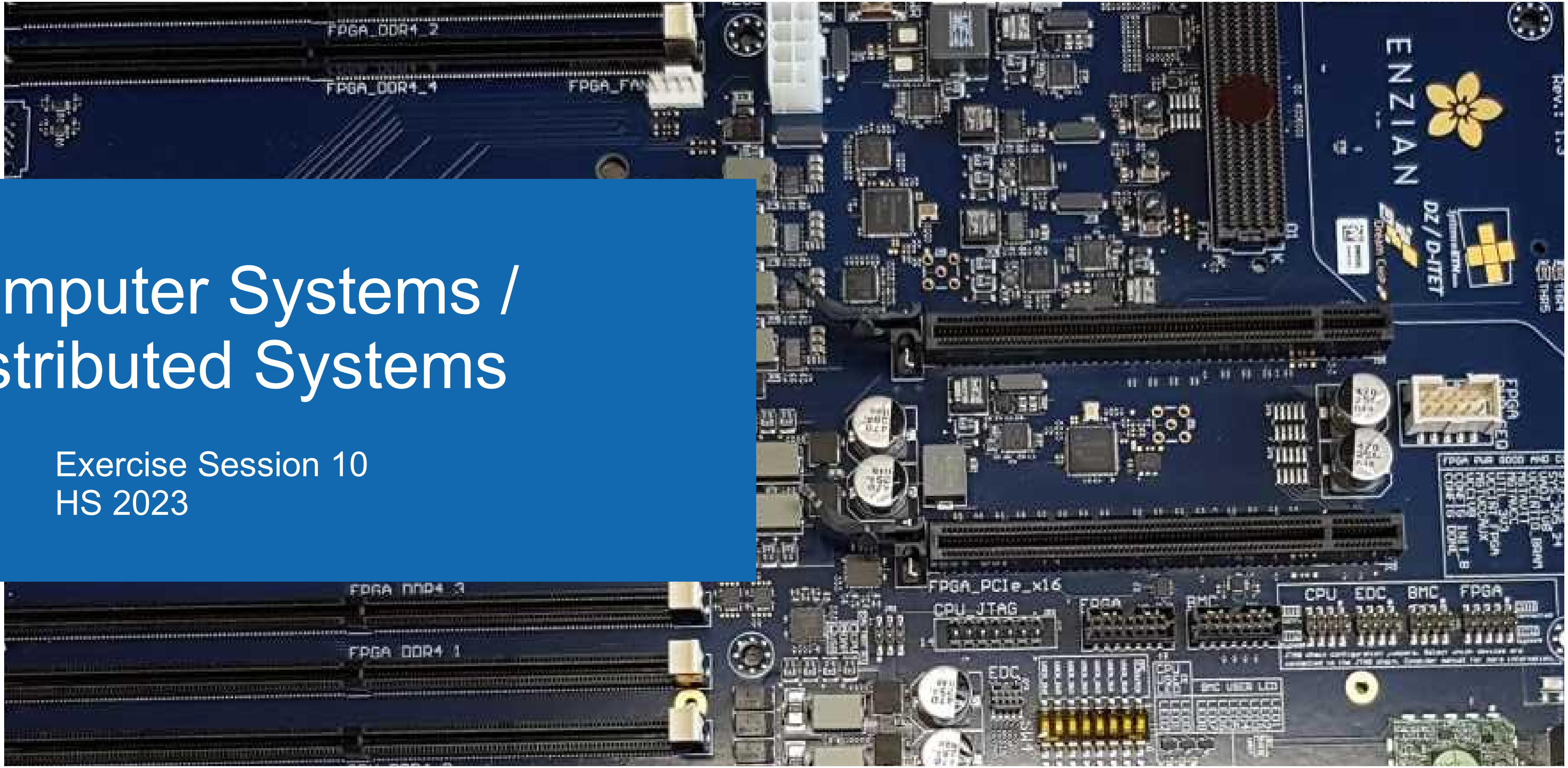
Slides last updated:
28.11.2023

Computer Systems / Distributed Systems

Exercise Session 10
HS 2023



Quorum Systems

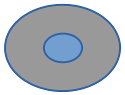
Quorum Systems

High-level functionality:

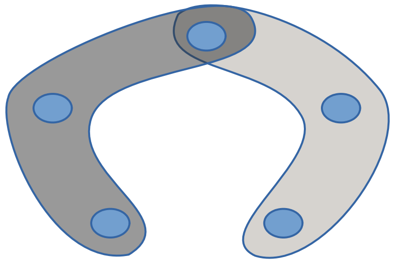
1. Client selects a free quorum
2. Locks all nodes of the quorum
3. Client releases all locks

ETH zürich Systems@ETH zürich Distributed Computing

Singleton and Majority Quorum Systems



Singleton quorum system



Majority quorum system
(all sets of n / 2 + 1 nodes)

Load and Work

An access strategy Z defines the probability $P_Z(Q)$ of accessing a quorum Q ∈ S such that:

$$\sum_{Q \in S} P_Z(Q) = 1$$

Load and Work

- **Load** of access strategy Z on a node v_i
- **Load** induced by Z on quorum system S
- **Load** of quorum system S

$$L_Z(v_i) = \sum_{Q \in S; v_i \in Q} P_Z(Q)$$

$$L_Z(S) = \max_{v_i \in S} L_Z(v_i)$$

$$L(S) = \min_Z L_Z(S)$$

- **Work** of quorum Q
- **Work** induced by Z on quorum system S
- **Work** of quorum system S

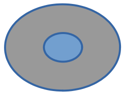
$$W(Q) = |Q|$$

$$W_Z(S) = \sum_{Q \in S} P_Z(Q) \cdot W(Q)$$

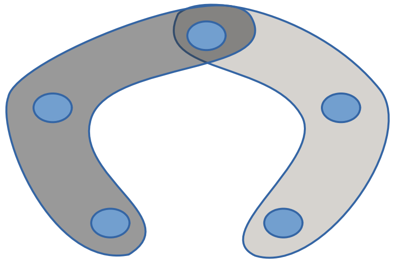
$$W(S) = \min_Z W_Z(S)$$

ETH zürich Systems@ETH zürich Distributed Computing

Load and Work



Singleton quorum system

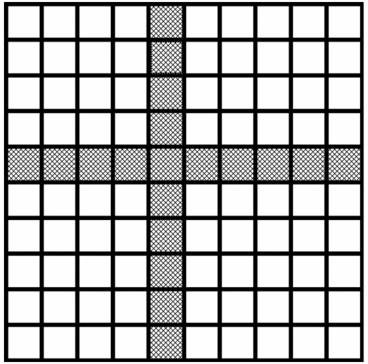


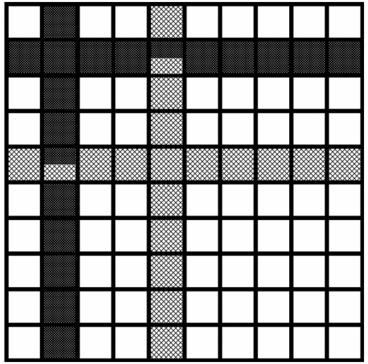
Majority quorum system
(all sets of n / 2 + 1 nodes)

	Singleton	**Majority**
How many servers need to be contacted? **(Work)**	1	> n/2
What's the load of the busiest server? **(Load)**	100%	≈ 50%
How many server failures can be tolerated? **(Resilience)**	0	< n/2

Basic Grid Quorum System

- Nodes arranged in a square matrix
- Each quorum i contains the union of row i and column i

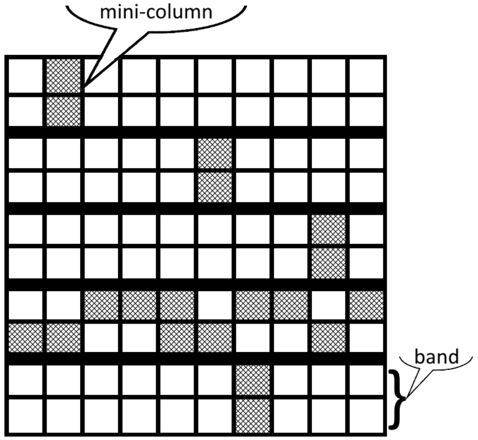




ETH zürich | Systems@ETH zürich | Distributed Computing

B-Grid Quorum System

- Nodes arranged in rectangular grid with h • r rows
- Group of r rows is a band
- Group of r elements in the same column and band is a mini-column
- **Quorums** consists of one mini-column in every band and one element from each mini-column of one band



Quiz

1. Does a quorum system exist which can tolerate that all nodes of a specific quorum fail?

2. Consider the **nearly all** quorum system, which is made up of n different quorums, each containing n - 1 servers. What is the resilience?

3. Can you think of a quorum system that contains as many quorums as possible? Note: does not have to be minimal.

Quiz Solution

1. Does a quorum system exist which can tolerate that all nodes of a specific quorum fail?

2. Consider the **nearly all** quorum system, which is made up of n different quorums, each containing n - 1 servers. What is the resilience?

3. Can you think of a quorum system that contains as many quorums as possible? Note: does not have to be minimal.

Quiz Solution

1. Does a quorum system exist which can tolerate that all nodes of a specific quorum fail?

   A: no, as any two quorums intersect!

2. Consider the **nearly all** quorum system, which is made up of n different quorums, each containing n - 1 servers. What is the resilience?

3. Can you think of a quorum system that contains as many quorums as possible? Note: does not have to be minimal.

Quiz Solution

1. Does a quorum system exist which can tolerate that all nodes of a specific quorum fail?

   A: no, as any two quorums intersect!

2. Consider the **nearly all** quorum system, which is made up of n different quorums, each containing n - 1 servers. What is the resilience?

   A: one, as two nodes failing fails all quorums!

3. Can you think of a quorum system that contains as many quorums as possible? Note: does not have to be minimal.

Quiz Solution

1. Does a quorum system exist which can tolerate that all nodes of a specific quorum fail?

   A: no, as any two quorums intersect!

2. Consider the **nearly all** quorum system, which is made up of n different quorums, each containing n - 1 servers. What is the resilience?

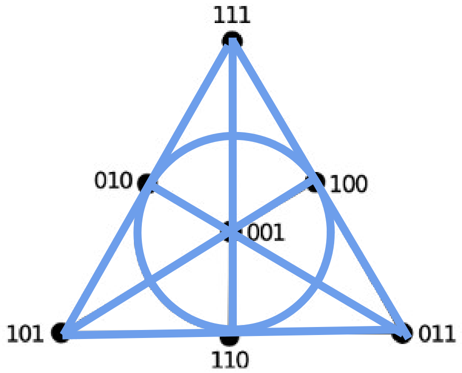
   A: one, as two nodes failing fails all quorums!

3. Can you think of a quorum system that contains as many quorums as possible? Note: does not have to be minimal.

   A: pick a node and take all quorums containing it. Maximality: between any quorum and its complement at most one can be in the system.

A Quorum System

Consider a quorum system with 7 nodes numbered from 001 to 111, in which each three nodes fulfilling $x \oplus y = z$ constitute a quorum. In the following picture this quorum system is represented: All nodes on a line (such as 111, 010, 101) and the nodes on the circle (010, 100, 110) form a quorum.

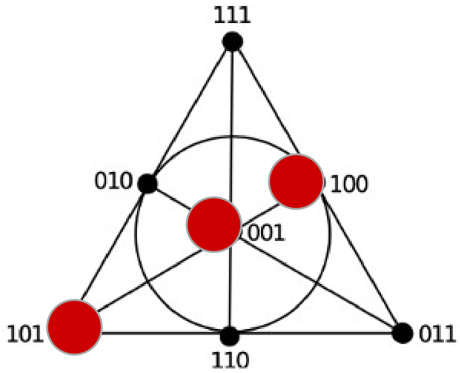


- Quorums: 7
- Work: 3
- Load: 3/7

a) Of how many different quorums does this system consist and what are its work and its load?

A Quorum System

Consider a quorum system with 7 nodes numbered from 001 to 111, in which each three nodes fulfilling $x \oplus y = z$ constitute a quorum. In the following picture this quorum system is represented: All nodes on a line (such as 111, 010, 101) and the nodes on the circle (010, 100, 110) form a quorum.



Resilience: 2

Every node is in 3 quorums
=> any two nodes can be contained in at most 2*3 quorums

b) Calculate its resilience f. Give an example where this quorum system does not work anymore with $f + 1$ faulty nodes.

Uniform Quorum Systems

Definitions:

s-Uniform: A quorum system $\mathcal{S}$ is *s-uniform* if every quorum in $\mathcal{S}$ has exactly s elements.

Balanced access strategy: An access strategy Z for a quorum system $\mathcal{S}$ is *balanced* if it satisfies $L_Z(v_i) = L$ for all $v_i \in V$ for some value L.

Claim: An s-uniform quorum system $\mathcal{S}$ reaches an optimal load with a balanced access strategy, if such a strategy exists.

a) Describe in your own words why this claim is true.

ETH zürich Systems@ETH zürich Distributed Computing

Uniform Quorum Systems

Definitions:

s-Uniform: A quorum system $\mathcal{S}$ is *s-uniform* if every quorum in $\mathcal{S}$ has exactly s elements.

Balanced access strategy: An access strategy Z for a quorum system $\mathcal{S}$ is *balanced* if it satisfies $L_Z(v_i) = L$ for all $v_i \in V$ for some value L.

Claim: An s-uniform quorum system $\mathcal{S}$ reaches an optimal load with a balanced access strategy, if such a strategy exists.

a) Describe in your own words why this claim is true.

Idea: No matter which quorum gets accessed, **exactly s nodes have to work**.

=> the sum of all loads should be to s

To minimize the maximum element of a sum, set all elements to the average (balanced access strategy).

ETH zürich Systems@ETH zürich Distributed Computing

Uniform Quorum Systems

Definitions:

s-Uniform: A quorum system $\mathcal{S}$ is s-*uniform* if every quorum in $\mathcal{S}$ has exactly s elements.

Balanced access strategy: An access strategy Z for a quorum system $\mathcal{S}$ is *balanced* if it satisfies $L_Z(v_i) = L$ for all $v_i \in V$ for some value L.

Claim: An s-uniform quorum system $\mathcal{S}$ reaches an optimal load with a balanced access strategy, if such a strategy exists.

b) Prove the optimality of a balanced access strategy on an s-uniform quorum system.

Uniform Quorum Systems

b) Let $V = \{v_1, v_2, ..., v_n\}$ be the set of servers and $\mathcal{S} = \{Q_1, Q_2, ..., Q_m\}$ an s-uniform quorum system on V. Let Z be an access strategy, thus it holds that: $\sum_{Q\in\mathcal{S}} P_Z(Q) = 1$. Furthermore let $L_Z(v_i) = \sum_{Q\in\mathcal{S}; v_i\in Q} P_Z(Q)$ be the load of server v_i induced by Z.

Then it holds that:

$$\sum_{v_i\in V} L_Z(v_i) = \sum_{v_i\in V} \sum_{Q\in\mathcal{S}; v_i\in Q} P_Z(Q) = \sum_{Q\in\mathcal{S}} \sum_{v_i\in Q} P_Z(Q)$$

$$= \sum_{Q\in\mathcal{S}} P_Z(Q) \sum_{v_i\in Q} 1 \stackrel{*}{=} \sum_{Q\in\mathcal{S}} P_Z(Q)\cdot s = s\cdot \sum_{Q\in\mathcal{S}} P_Z(Q) = s$$

The transformation marked with an asterisk uses the uniformity of the quorum system.

To minimize the maximal load on any server, the optimal strategy is to evenly distribute this load on all servers. Thus if a balanced access strategy exists, this leads to a system load of s/n.

Approximate Agreement

Approximate Agreement

It enables nodes to obtain values that are:

1. within the range of correct inputs (**correct-range validity**)
2. ε-close for some predefined $\varepsilon > 0$ (**ε-agreement**)

3. $n > 3f$ must hold
4. synchronous algorithm for $f < n/3$ byzantine nodes
5. asynchronous algorithm for $f < n/3$ byzantine nodes

Algorithm outline

$\mathbf{I}$ = a sufficient number of iterations
$\mathbf{x_0}$ = initial value

for $\mathbf{i} = 1 \ldots \mathbf{I}$:

- Distribute your value $\mathbf{x_{i-1}}$.
- $\mathbf{R}$ = multiset containing the values received.
- $\mathbf{T}$ = multiset containing all but the lowest f and the highest f values in $\mathbf{R}$.
- $\mathbf{x_i} = (\min \mathbf{T} + \max \mathbf{T}) / 2$

Output $\mathbf{x_I}$

Insights

1. The multisets **R** contain at most f corrupted values
   => the multisets **T** are included in the range of correct values.

2. If any two correct nodes obtain multisets **R** that intersect in n - f values, the range of correct values is **halved** in each iteration
   1. **synchronous** model: simply sending your value to everyone is enough.
   2. **asynchronous** model: *witness technique*.

Single Value Reliable-Broadcast

- **asynchronous** network with f < n/3 byzantine nodes

- Properties:
  - If the sender is correct, all correct nodes accept its value eventually.
  - **If a correct node accepts x, no correct node accepts y != x.**
  - If a correct node accepts x, all correct nodes accept x eventually.

Witness Technique

Key idea:

Once a node accepts values from $n - f$ nodes via Single-Value Reliable Broadcast, it tries to convince all nodes to wait ***a bit longer***: so that they receive these nodes' values as well.

=> nodes obtain multisets **R** that pair-wise intersect in $n - f$ values.

Quiz

In the lecture, you have seen a Single-Value Reliable Broadcast algorithm (Algorithm 20.11). Sometimes, ideas used in the asynchronous model also lead to cute properties in the synchronous model. Let us analyze the algorithm below in a **synchronous** network where $f < n/3$ of the nodes are byzantine.

Algorithm 1 Single-Valued Reliable Broadcast, But in a Synchronous Network

---

Code for sender v_S with input x_S:
Round 1: Send $\texttt{msg}(x_S)$ to everyone.

Code for node v:
Round 2:
    If you received a message $\texttt{msg}(x)$ from the sender:
    Send $\texttt{echo}(x)$ to everyone.

Round 3 or later:
    Upon receiving $\texttt{echo}(x)$ from $n - f$ distinct nodes or $\texttt{ready}(x)$ from $f + 1$ distinct nodes:
        Send $\texttt{ready}(x)$ to everyone.

Round 4 or later:
    Upon receiving $\texttt{ready}(x)$ from $2f + 1$ distinct nodes:
    Accept $\texttt{msg}(x)$.

---

a) What strategy should the byzantine nodes use so that two correct nodes accept different values?

Algorithm 1 Single-Valued Reliable Broadcast, But in a Synchronous Network

Code for sender v_S with input x_S:
Round 1: Send `msg`(x_S) to everyone.

Code for node v:
Round 2:
    If you received a message `msg`(x) from the sender:
    Send `echo`(x) to everyone.

Round 3 or later:
    Upon receiving `echo`(x) from $n - f$ distinct nodes or `ready`(x) from $f + 1$ distinct nodes:
        Send `ready`(x) to everyone.

Round 4 or later:
    Upon receiving `ready`(x) from $2f + 1$ distinct nodes:
    Accept `msg`(x).

a) What strategy should the byzantine nodes use so that two correct nodes accept different values? Lemma 20.16 from the lecture notes ensures that there is no such strategy.

b) Assume that a correct node v has accepted $\texttt{msg}(x)$. Explain why every correct node accepts $\texttt{msg}(x)$ within two additional communication rounds.

Algorithm 1 Single-Valued Reliable Broadcast, But in a Synchronous Network

Code for sender v_S with input x_S:
Round 1: Send $\texttt{msg}(x_S)$ to everyone.

Code for node v:
Round 2:
    If you received a message $\texttt{msg}(x)$ from the sender:
    Send $\texttt{echo}(x)$ to everyone.

Round 3 or later:
    Upon receiving $\texttt{echo}(x)$ from $n - f$ distinct nodes or $\texttt{ready}(x)$ from $f + 1$ distinct nodes:
        Send $\texttt{ready}(x)$ to everyone.

Round 4 or later:
    Upon receiving $\texttt{ready}(x)$ from $2f + 1$ distinct nodes:
    Accept $\texttt{msg}(x)$.

a) What strategy should the byzantine nodes use so that two correct nodes accept different values? Lemma 20.16 from the lecture notes ensures that there is no such strategy.

b) Assume that a correct node v has accepted $\texttt{msg}(x)$. Explain why every correct node accepts $\texttt{msg}(x)$ within two additional communication rounds.

If a correct node v accepts $\texttt{msg}(x)$ at some point in time τ, then $f+1$ correct nodes have sent $\texttt{ready}(x)$ by time τ. Therefore, since the network is synchronous, all correct nodes receive these $f+1$ messages $\texttt{ready}(x)$ within one additional communication round, and therefore send $\texttt{ready}(x)$. These messages are afterwards received within one more communication round, hence within two communication rounds after time τ.

c) Assume that a correct node v has not accepted a value by the end of round 4. What does that tell v about the sender v_S?

Algorithm 1 Single-Valued Reliable Broadcast, But in a Synchronous Network

---

Code for sender v_S with input x_S:

Round 1: Send `msg`(x_S) to everyone.

Code for node v:

Round 2:

  If you received a message `msg`(x) from the sender:

  Send `echo`(x) to everyone.

Round 3 or later:

  Upon receiving `echo`(x) from $n - f$ distinct nodes or `ready`(x) from $f + 1$ distinct nodes:

    Send `ready`(x) to everyone.

Round 4 or later:

  Upon receiving `ready`(x) from $2f + 1$ distinct nodes:

  Accept `msg`(x).

---

a) What strategy should the byzantine nodes use so that two correct nodes accept different values? Lemma 20.16 from the lecture notes ensures that there is no such strategy.

b) Assume that a correct node v has accepted $\texttt{msg}(x)$. Explain why every correct node accepts $\texttt{msg}(x)$ within two additional communication rounds.

If a correct node v accepts $\texttt{msg}(x)$ at some point in time τ, then $f+1$ correct nodes have sent $\texttt{ready}(x)$ by time τ. Therefore, since the network is synchronous, all correct nodes receive these $f+1$ messages $\texttt{ready}(x)$ within one additional communication round, and therefore send $\texttt{ready}(x)$. These messages are afterwards received within one more communication round, hence within two communication rounds after time τ.

c) Assume that a correct node v has not accepted a value by the end of round 4. What does that tell v about the sender v_S?

Note that, if the sender v_S is correct, every correct node accepts $\texttt{msg}(x_S)$ in round 4. This is because all nodes receive the sender's value by the beginning of round 2. In round 2, all correct nodes send $\texttt{echo}(x_S)$, and these $n-f$ messages get delivered by the third round. Then, round 3, all correct nodes send $\texttt{ready}(x_S)$, and therefore all correct nodes accept the sender's value in round 4.

Then, if a node did not receive the sender's value by the end of round 4, then the sender must be a byzantine node.

2.2 From Approximate Agreement to Byzantine Agreement

We want to design an **asynchronous** byzantine agreement algorithm (where nodes' inputs are bits) that relies on Algorithm 20.22 from the lecture nodes. Recall that Algorithm 20.22 achieves asynchronous approximate agreement even when $f < n/3$ of the nodes are byzantine.

Nodes proceed as follows: every node joins Algorithm 20.22 with its input bit as initial value. Once a node obtains a value x from Algorithm 20.22, it outputs 0 if $x < 0.5$ and 1 otherwise.

a) Does all-same validity hold?

b) What about agreement?

c) Assume an ideal shared coin that enables the nodes to agree on a uniformly distributed random value in $(0, 1)$. Once $f + 1$ nodes query this shared coin, the random value is sampled and all nodes learn it eventually.

How can we use this coin to achieve agreement except with probability 10^{-2023}?

2.2 From Approximate Agreement to Byzantine Agreement

a) Yes. If all correct nodes have the same input bit b, correct-range validity ensures that all correct nodes obtain value $x = b$.

b) No. If the correct nodes have distinct input bits, then they can obtain any ε-close values in $[0, 1]$. It is possible that a correct node obtains $x = 0.5 - \varepsilon/3$, and therefore its final output is 0, while another correct node obtains $x = 0.5 + \varepsilon/3$ and therefore its final output is 1.

c) We set $\varepsilon = 1/2 \cdot 10^{-2023}$. The nodes join the approximate agreement algorithm with their input bits as initial values. When a node obtains a value x, it queries the shared coin. Once $f + 1$ nodes (hence at least one correct node v) query the coin, the random value r is decided and all nodes learn it eventually. When a node has obtained both the random value r and a value x via approximate agreement, if outputs 0 if $x < r$ and 1 otherwise.

The outputs x obtained via approximate agreement are ε-close, and agreement only fails if r is between the lowest and the highest values x obtained by correct nodes via approximate agreement. Then, if the first correct node that queries the coin has obtained value x, all correct nodes obtain outputs in the interval $[x - \varepsilon, x + \varepsilon]$. This means that only values $r \in [x - \varepsilon, x + \varepsilon]$ may lead to disagreement. Such a value r is obtained with probability at most $2 \cdot \varepsilon = 10^{-2023}$.

2.3 Unbounded Input Space: Quick Fix

The approximate agreement algorithms presented in the lecture rely on a publicly known **max_range** that the input space should satisfy. This allows us to (overestimate) a sufficient number of iterations. To drop this assumption **in the synchronous model** (Algorithm 20.10), we will build a mechanism that enables each node to (over)estimate a **max_range** based on the nodes' inputs. Hence, if X denotes the multiset of correct inputs, we will ask each node to estimate $\max X - \min X$.

a) How would obtaining agreement on $\max X - \min X$ help?

b) Describe in your own words why correct nodes cannot agree on $\max X - \min X$.

Instead, each node will try to *estimate* the initial range X. This can be done using one round of communication preceding the for loop of Algorithm 20.10.

c) Write an algorithm that uses one round of communication and allows each correct node v to obtain an estimation $\texttt{max\_range}_v \geq \max X - \min X$.

d) How can the algorithm from Task **c)** be used to replace the hard-coded value I in Algorithm 20.10? Keep in mind that nodes do not obtain the same value $\texttt{max\_range}_v$.

e) Can you provide an upper bound on the number of iterations in your solution in Task **d)**?

2.3 Unbounded Input Space: Quick Fix

a) Nodes could simply define the number of iterations $I = \lceil \log_2((\max X - \min X)/\varepsilon) \rceil$.

b) Similarly to *correct-input validity*, one cannot distinguish between a correct node and a byzantine node that follows the algorithm correctly, but with an input of its own choice.

c) The algorithm proceeds as follows: every node sends its value to all nodes. Node v computes its estimation $\texttt{max\_range}_v$ as the difference between the highest value received (which is at least $\max X$, since all correct values were received), and the lowest value received (which is at most $\min X$, also because all correct values were received).

Note: removing the lowest f and the highest f values might discard some correct values and make the algorithm stop too early.

Algorithm 1 Synchronous Approximate Agreement: Unbounded Input Space

Code for node v with input x.
Send v to all nodes.
Add every received value to X.
$\texttt{max\_range}_v = \max X - \min X$.
$I_v = \lceil \log_2(\texttt{max\_range}_v / \varepsilon) \rceil$.
$x_0 = x$.
for i in $1 \ldots I_v$ **do**
    Send x_{i-1} to all nodes.
    Add every received value to R_i.
    If node u sent $(\texttt{halt}, x_{I_u})$ (now or in some previous iteration), add x_{I_u} to R_i.
    $T_i =$ the multiset obtained by removing the lowest f and the highest f values in R_i.
    $x_i = (\min T_i + \max T_i)/2$.
end for
Send $(\texttt{halt}, x_{I_u})$ to all the nodes.
Output x_{I_u}.

Algorithm 1 Synchronous Approximate Agreement: Unbounded Input Space

Code for node v with input x.
Send v to all nodes.
Add every received value to X.
$\texttt{max\_range}_v = \max X - \min X$.
$I_v = \lceil \log_2(\texttt{max\_range}_v / \varepsilon) \rceil$.
$x_0 = x$.
for i in $1 \ldots I_v$ **do**
    Send x_{i-1} to all nodes.
    Add every received value to R_i.
    If node u sent $(\texttt{halt}, x_{I_u})$ (now or in some previous iteration), add x_{I_u} to R_i.
    T_i = the multiset obtained by removing the lowest f and the highest f values in R_i.
    $x_i = (\min T_i + \max T_i)/2$.
end for
Send $(\texttt{halt}, x_{I_u})$ to all the nodes.
Output x_{I_u}.

e) With the mechanism above, not really. The values $\texttt{max\_range}_v$ are essentially chosen by the byzantine nodes.