Consensus

FAULT TOLERANCE
• Processes
  ○ the number of the processes is $n$
  ○ Processes have unique (known) identities

• Communication
  ○ Shared memory (read/write)
  ○ Message passing (send/receive)
Failure

- **Link failure:**
  - Loss, duplication, modification...

- **Process failure:**
  - Crash failure
  - Omission (send/receive)
  - Byzantine failure

Correct/ incorrect process
Synchrony

- **Synchronous system:**
  - There is a bound $\delta$ such that for all processes $p$ and $q$: if a message is sent by process $p$ to process $q$ at time $\tau$ then $q$ receives this message by time $\delta + \tau$.
  - There is a bound $\delta$ such that for all processes $p$: in $\delta$ consecutive steps of any process $p$ there is at least one step of any process.

- **Asynchronous system:** no bound

- **Partially synchronous system:**
  - The bound exists but is unknown.
  - There is a time after which the bound holds.
  - ….
A problem $P$, a system $S$ of $n$ processes:

- There does not exist an algorithm that solves $P$ in system $S$ assuming at most $t$ failures.

- There exists an algorithm that solves $P$ in system $S$ assuming at most $t$ failures (in some time, number of messages ...).
Round model

- At each round:
  - All (live) processes send a message to all processes.
  - Wait (following some condition) in order to receive the messages of the round
  - Make some computation
Synchronous system:

- There is a bound $\delta$ such that for all processes $p$ and $q$: if a message is sent by process $p$ to process $q$ at time $\tau$ then $q$ receives this message by time $\delta + \tau$.
- There is a bound delta such that for all processes $p$: in $\delta$ consecutive steps of any process $p$ there is a step of any process.

Wait for $\delta$ time

All processes receive all the messages sent by correct processes.
Asynchronous system: no bound.

If $t$ is the number of processes that may crash

Wait for $n-t$ messages

It is possible that a correct process does not wait for a message sent by another correct process
Consensus

- At the heart of fault tolerance systems:
  - At least the correct processes have to agree on something!

- Ex: active replication
  - Each process executes the same code
  - At each step they agree on the same inputs
  - They output the same sequence

- agree on the same inputs \implies Consensus
Processes propose some values, they have to agree (decide) on one of them.

- If all processes propose the same value $v$ then all correct processes decide $v$.
- If processes propose different values then all correct processes have to decide the same value and this value has to be one of the proposed values.
Specifications

- **Agreement**: if two processes decide they decide the same value
- **Validity**: if a process decides it decides a value that has been proposed
- **Termination**: all correct processes decide
<table>
<thead>
<tr>
<th></th>
<th>synchronous</th>
<th>asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>No failure</td>
<td>Algo (1 round)</td>
<td>Algo (1 round)</td>
</tr>
<tr>
<td>Crash failure</td>
<td>Algo (t+1 rounds)</td>
<td>Impossible (FLP85)(t=1)</td>
</tr>
<tr>
<td>Byzantine failure</td>
<td>Algo if $n&gt;3t$ (t+1 rounds)</td>
<td>Impossible if $n=3t$</td>
</tr>
</tbody>
</table>
If number of 1 > number of 0 then decide 0 else decide 1
If number of 1 > number of 0 then decide 0 else decide 1
If number of 1 > number of 0 then decide 0 else decide 1

D=?
In asynchronous systems if you design an algorithm that can tolerate $t$ crashes you can only wait for $n-t$ processes.
<table>
<thead>
<tr>
<th>Failure Type</th>
<th>Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash failure</td>
<td>Algo (O(1)) expected rounds if (n&gt;2t)</td>
<td>Algo if (n&gt;2t) Impossible if (n\leq2t)</td>
</tr>
<tr>
<td>Byzantine failure</td>
<td>No better result</td>
<td>Algo if (n&gt;3t) Impossible if (n\leq3t)</td>
</tr>
</tbody>
</table>
Ben-Or’s Algorithm

- Assume a majority of correct processes \((n>2t)\)
- Every process can toss a coin that gives value 0 or 1 with probability \(c>0\)
- Binary consensus

\[x:=\text{value proposed by } p\]
- For \(k:=1, 2, \ldots\)
  - send\((R, k, x)\) to all processes
  - wait for messages of the form \((R,k,*)) from \(n-t\) processes
  - If received \((R,k,v)\) with the same \(v\) then send\((P, k, v)\) to all processes
    else send\((P, k, ?)\) to all processes
wait for messages of the form \((R,k,*)\) from n-t processes
Property of majority

At the end of the round:

- All live processes send \((P, k, 0)\) or \((P, k,?)\)
- All live processes send \((P, k, 1)\) or \((P, k,?)\)
- **Impossible**: a process sends \((P, k, 1)\) and another \((P, k, 0)\)
Ben-Or’s Algorithm

- wait for messages of the form (P,k,*) from n-t processes
- If received (R,k,v) from t+1 with v ≠?
  - then decide v
  - else if received at least one (R,k,v) with v≠?
    - then x:=v
    - else x:=COIN_TOSS()
Agreement

- If a process decides \( \nu \), all processes have \( x=\nu \)
Decide 0---------0000000000000

0?? x=0
If all processes begin the phase $k$ with the same value $v$
- All processes send($P,k,v$)
- All processes send($R,k,v$)
- All processes decide $v$
Validity

- If all processes begin with $v$ all processes decide $v$ at the end of the first phase.
- If not all processes begin with $v$ then it is possible to decide either 0 or 1
Termination with proba 1

- At the end of a phase
  - either processes have the same value $v$
  - or some processes have $v$ and some other toss coin

If the result of the coin toss is $v$, at the next phase all processes begin with the same value and decide.
Termination with proba 1

- Either all processes have the same value $v$ (≠’?)
- or no process has a value (≠’?) and all processes toss coin the same value
- or some processes have $v$ and some others toss coin $v$
Definition: A coin is called *weakly global* if there exists an absolute constant $c > 0$, such that for all $v$ in $\{0, l\}$ the probability that at least $\min(n/2 + t + 1, n)$ processes all see outcome $v$ is at least $c$. 
Algorithm

- Idea: a leader randomly volunteers, and this leader tosses a coin.
- The procedure LEADER produces a local biased bit where the probability of a 1 (“I volunteer”) is equal to $1/n$;
- the procedure RANDOM_BIT produces a local unbiased bit.
Code for processor P:
Function COIN_TOSS
• leader:=LEADER()
• value:= RANDOM_BIT()
• send (leader, value) to all
• receive all (l, v) messages
• if all messages received with l = 1 have the same v then COIN_TOSS:=v
  else COIN_TOSS:=RANDOM_BIT()
Theorem: The function COIN_TOSS, produces a weakly global coin, where the constant probability for either common outcome is at least $1/e$ if $2t < n$. 
Quantum?
Quantum

Can be used to obtain a weak global coin

<table>
<thead>
<tr>
<th></th>
<th>Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash failure</td>
<td>No better results</td>
<td>No better result</td>
</tr>
<tr>
<td>Byzantine failure</td>
<td>Algo $O(1)$ expected rounds if $n&gt;3t$</td>
<td>Algo if $n&gt;3t$</td>
</tr>
<tr>
<td></td>
<td>Impossible if $n&lt;3t$</td>
<td>Impossible if $n&lt;3t$</td>
</tr>
</tbody>
</table>
Quantum weak global coin

Function COIN_TOSS()
Code for process \( p_i \)

Round 1:
- Generate the state \( |C_{\text{Coin}}_i\rangle = \frac{1}{\sqrt{2}} |0,0,...0\rangle + \frac{1}{\sqrt{2}} |1,1,...1\rangle \) on \( n \) qubits and send the \( k \)-th qubit to the \( k \)-th player (keeping one part to yourself)
- Generate the state
  \[ |\text{Leader}_i\rangle = \frac{1}{n^3} \sum_{a=1}^{n^3} |a,a,...a\rangle \]
on \( n \) qubits, an equal superposition of the numbers between 1 and \( n^3 \). Distribute the \( n \) qubits among all processes
- Receive the quantum messages
Round 2:

- Measure (in the standard base) all Leader\textsubscript{j} qubits received in round 1. Select the process with the highest Leader value as the “leader” of the round.
- Measure the leader ‘s coin in the standard base return (measurement outcome of the leader’s coin)
Theorem: The function COIN_TOSS, produces a weakly global coin, where the constant probability for either common outcome is at least $\frac{1}{3}$ if $3t < n$. 
Can we do better with quantum computing?


• Michael Ben-Or, Avinatan Hassidim: Fast quantum byzantine agreement. *STOC* 2005: 481-485