① Basic abstraction

1. Process and Link
2. Properties: Livliness and safety
3. Process failures (5 crash-fault, omissions, crash-recovery, eavesdropping, Byzantine arbitary faults)
4. Crypto abstraction: 3 hash, MAC, Digital signature.
5. Abstraction communication: 4 fully-connected mesh, ring, broadcast medium, mesh of links intercorn, by bridges and route
6. link abstraction: (All are point-to-point link).
   1. Fair-loss links (FLL1, FLL2, FLL3)

Crash-stop
   2. Stubborn Links (SL1, SL2), — Algo: Retransmit Forever (every step, just send again and again) + no dublication
   3. Perfect Links (PL1, PL2, PL3) — Algo: Eliminate Dublicates (deliver only unique, receive everything)

Crash-Recovery
   4. Logged Perfect Links (LPL 1-3 + Indication (stable Storage)) — Algo: Log Delivered (+ safe not delivered)

Byzantine
   5. Authenticated Perfect Links (AL1-3+Indication) — Algo: Authenticate and Filter (MAC) Authenticity (instead duno creation, check if sender correct)

reliable delivery (under never crashes) + indication (Epe, Deliver | delivered)

1. **FLL1:** *Fair-loss:* If a correct process p infinitely often sends a message m to a correct process q, then q delivers m an infinite number of times.

   FLL2: *Finite duplication:* If a correct process p sends a message m a finite number of times to process q, then m cannot be delivered an infinite number of times by q.

   FLL3: *No creation:* If some process q delivers a message m with sender p, then m was previously sent to q by process p.

2. **SL1:** *Stubborn delivery:* If a correct process p sends a message m once to a correct process q, then q delivers m an infinite number of times.

   SL2: *No creation:* If some process q delivers a message m with sender p, then m was previously sent to q by process p.

3. **PL1:** *Reliable delivery:* If a correct process p sends a message m to a correct process q, then q eventually delivers m.

   PL2: *No duplication:* No message is delivered by a process more than once.

   PL3: *No creation:* If some process q delivers a message m with sender p, then m was previously sent to q by process p.

4. **LPL1:** *Reliable delivery:* If a process that never crashes sends a message m to a correct process q, then q eventually log-delivers m.

   LPL2: *No duplication:* No message is log-delivered by a process more than once.

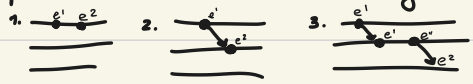
   LPL3: *No creation:* If some process q log-delivers a message m with sender p, then m was previously sent to q by process p.

5. **AL1:** *Reliable delivery:* If a correct process sends a message m to a correct process q, then q eventually delivers m.

   AL2: *No duplication:* No message is delivered by a correct process more than once.

   AL3: *Authenticity:* If some correct process q delivers a message m with sender p and process p is correct, then m was previously sent to q by p.

7. Time Assumptions:
   1. Synchronous System
      - We know: processing delay + m transmission delays + local physical clock deviation
      - $2 \cdot \Delta t_s + \Delta tp \leq 2 \cdot t_{link} + \Delta tq$
   2. Asynchronous System
      - Logical time (clock)
      - Algo to construct Logical clock
      - Happen-before: (Eeach process generates events in a seq)



1. Each process p keeps an integer called *logical clock* l_p, initially 0.
2. Whenever an event occurs at process p, the logical clock l_p is incremented by one unit.
3. When a process sends a message, it adds a timestamp to the message with the value of its logical clock at the moment the message is sent. The timestamp of an event e is denoted by $t(e)$.

Opposite is not true ...

- Casual history

8. Failure detector

Synchronous system →
1. Perfect Failure Detection (PFD1, PFD2)
   - Algo: Exclude on Timeout
   - Synchronous system, so timeout + heart-beat
2. Leader Election (LE1, LE2)
   - Algo: Monarchical leader Election
3. Eventually-Perfect Failure Detector
   - Algo: Increasing Timeout (wrongly suspected → restore)
4. Eventual Leader Detection (ELD1, ELD2)
   - Algo: 1. Monarchical Eventual leader Detection + 2. Elect lower Epoch (remember who how many times fails)

PFD1: *Strong completeness:* Eventually, every process that crashes is permanently detected by every correct process.

PFD2: *Strong accuracy:* If a process p is detected by any process, then p has crashed.

LE1: *Eventual detection:* Either there is no correct process, or some correct process is eventually elected as the leader.

LE2: *Accuracy:* If a process is leader, then all previously elected leaders have crashed.

EPFD1: *Strong completeness:* Eventually, every process that crashes is permanently suspected by every correct process.

EPFD2: *Eventual strong accuracy:* Eventually, no correct process is suspected by any correct process.

ELD1: *Eventual accuracy:* There is a time after which every correct process trusts some correct process.

ELD2: *Eventual agreement:* There is a time after which no two correct processes trust different correct processes.

9. Distributed Systems Models

	P	L	FD	
fail-stop	c-s	perfect	perfect	← synchronous
fail-noisy	c-s	perfect	eventually perfect	← partially synch.
fail-silent	c-s	perfect	none	← asynchronous
fail-recovery	c-r	stubborn	eventual. perfect	
fail-silent-arbitary	Byzantine	authentic. perfect	none	
fail-noisy-arbitary	Byzantine	auth. perfect	Byzantine eventual leader detector.	

(2) Reliable Broadcast

1. Strongest assumptions (fail-stop: c-s, perfect, _____)

Best-Effort Broadcast

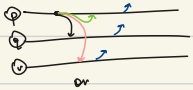
- Assumptions: all process know Π, p-t-p link between all Π, messages are unique,

Algorithm: O(N) Steps

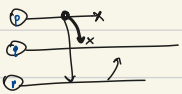
1. Basic Broadcast.

- **BEB1:** *Validity:* If a correct process broadcasts a message m, then every correct process eventually delivers m.
  BEB2: *No duplication:* No message is delivered more than once.
  BEB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

- Sender Correct ∨ some p. correct



or



- We need some property if sender is faulty, to make sure that all process will have the same reality. => inconsistencies

2. (Regular) Reliable Broadcast

Properties:

RB1: *Validity:* If a correct process p broadcasts a message m, then p eventually delivers m.

RB2: *No duplication:* No message is delivered more than once.

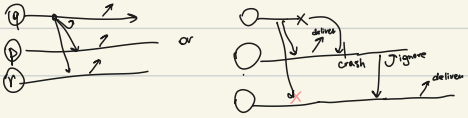
RB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

RB4: *Agreement:* If a message m is delivered by some correct process, then m is eventually delivered by every correct process.

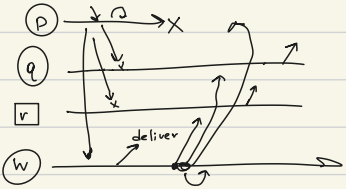
Algorithm: (Two ways to hold „Agreement" property: 1. relies on Perfect failure detector 2. Take care of everything but increase complexity)

1. Fail-stop: Lazy Reliable Broadcast (Synchronous system) $O(N) + O(N^2)$ messages (worst case)

- Retransmit a message only if the original sender has been detected to have crashed.



2. Fail-silent: Eager Reliable Broadcast - $O(n^2)$



3. Uniform Broadcast

(differ from reliable broadcast : inconsistent, side-effects

)

Properties:

RB1: *Validity:* If a correct process p broadcasts a message m, then p eventually delivers m.

RB2: *No duplication:* No message is delivered more than once.

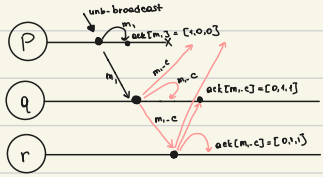
RB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

URB4: *Uniform agreement:* If a message m is delivered by some process (whether correct or faulty), then m is eventually delivered by every correct process.

Algorithms:

1. All-Ack URB (Perfect Failure Detector) [2 communication step, N^2 messages]

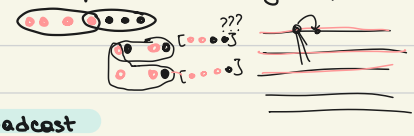
- Every process relay the message once, after they have seen it. Each process keeps a record of processes from which they received a message. The algo uses an array ack[m] gather the set of processes which seen m.



2. Majority-Ack URB (Fail-silent) [2 communication step, N^2 messages]

- Assumes a majority of correct processes i.e. $N > 2f$. So don't wait responce from all process, just majority.

def Quorum- subset of processes such that any two quorums intersect on at least one correct process.



4. Probability broadcast

Properties:

PB1: *Probabilistic validity:* There is a positive value ε such that when a correct process broadcasts a message m, the probability that every correct process eventually delivers m is at least $1 - \varepsilon$.

PB2: *No duplication:* No message is delivered more than once.

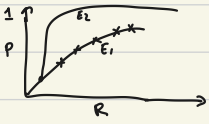
PB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

Algorithm: Eager Probabilistic Broadcast ( )

- Probabilistic validity

$$E_1 = 1 - \left(1 - \frac{k}{N}\right)^R$$

$$E_2 = 1 - \left(1 - \frac{k}{N}\right)^{\sum_{r=0}^{R-1} I_r}$$



5. FIFO

Properties:

FRB1–FRB4: Same as properties RB1–RB4 in (regular) reliable broadcast (Module 3.2).

FRB5: *FIFO delivery:* If some process broadcasts message m_1 before it broadcasts message m_2, then no correct process delivers m_2 unless it has already delivered m_1.

6. Casual Broadcast

Properties:

CRB1–CRB4: Same as properties RB1–RB4 in (regular) reliable broadcast (Module 3.2).

CRB5: *Causal delivery:* For any message m_1 that potentially caused a message m_2, i.e., $m_1 \rightarrow m_2$, no process delivers m_2 unless it has already delivered m_1.

(3) Byzantine Broadcast

(1) Byzantine Quorums – tolerate up to f Byzantine processes. (N process overall)

Set of more $\frac{N+f}{2}$ processes.

Q – number of nodes in a Byzantine Quorum

C correct nodes

CQ correct nodes in quorum Q

$\#C = N-f$ $\quad \#Q > \frac{N+f}{2}$ by def; $\#CQ > \frac{N+f}{2} - f = \frac{N-f}{2}$ ← minimum for consistency

- Two BQ overlap in at least **one** correct process. consistency
  - Only N-f correct in the whole system.

    $\#CQ_1 + \#CQ_2 > \frac{N-f}{2} + \frac{N-f}{2} = N-f$, so they intersect in 1 correct process.
- There is at least ONE BQ with all correct. (what is f in relation to N) (Liveness)

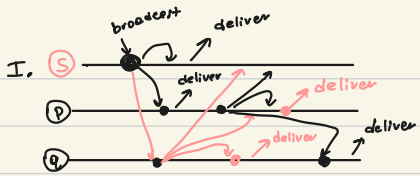
  $\#C = N-f$. $\exists Q_1$ with correct p. $\#CQ_1 > \frac{N+f}{2} - 0$,

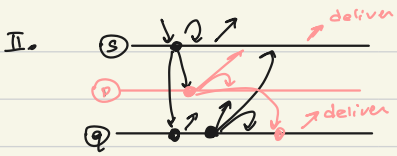
  $\#C \geq \#CQ_1 > \frac{N+f}{2}$

  $N-f > \frac{N+f}{2}$, $2N-2f > N+f$, $\boxed{N > 3f}$
- Majority in a BQ has to be correct p.

(2) Byzantine consistent Broadcast (Fair-arbidary model)

- Remark: No exact violation of dublication and agreement, but we have smth looks like a dublication:





- This was dubication
- Agreement: different delivery orderings.

Properties:

BCB1: *Validity:* If a correct process p broadcasts a message m, then every correct process eventually delivers m.

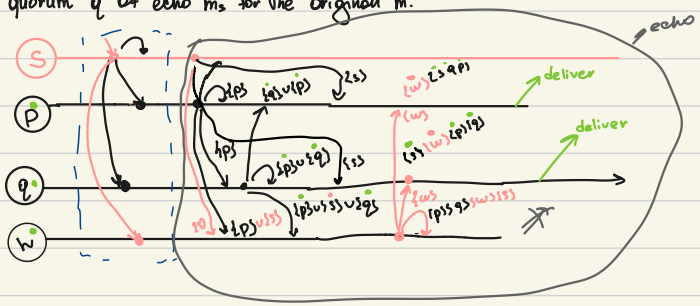
BCB2: *No duplication:* Every correct process delivers at most one message.

BCB3: *Integrity:* If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p.

BCB4: *Consistency:* If some correct process delivers a message m and another correct process delivers a message m', then $m = m'$.

Algorithm:

1. Authenticated Echo Broadcast (some process might NOT deliver anything) (Authenticated link + N>3f ← for consistency) [$O(N^2)$ messag] (2 step)
   - Two rounds of messages exchanges. No correct process delivers until they have received a Byzantine quorum q of echo ms for the original m.



- using digital signs we can reduce messages

③ Byzantine Reliable Broadcast (one message)

Properties:

BCB1: *Validity:* If a correct process p broadcasts a message m, then every correct process eventually delivers m.

BCB2: *No duplication:* Every correct process delivers at most one message.

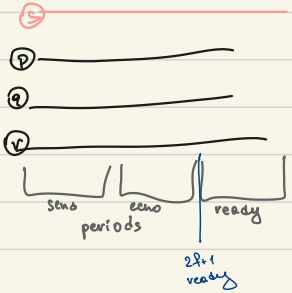
BCB3: *Integrity:* If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p.

BCB4: *Consistency:* If some correct process delivers a message m and another correct process delivers a message m', then $m = m'$.

BRB5: *Totality:* If some message is delivered by any correct process, every correct process eventually delivers a message.

- Algorithm: Authenticated Double-Echo Broadcast (3 round: 2 echo.) [$O(N^2)$ + 3 com. steps]

  - $\frac{N+f}{2}$ echo → ready [$\exists CQ$]
  - $f+1$ ready for you → send ready (so byzantine can't cause amplification)
  - $2f$ ready → deliver



(4) Byzantine Broadcast Channels (every process can broadcast m, just add name → many messages)

1 Byzantine consist channel

Properties:

BCCH1: *Validity:* If a correct process p broadcasts a message m, then every correct process eventually delivers m.

BCCH2: *No duplication:* For every process p and label ℓ, every correct process delivers at most one message with label ℓ and sender p.

BCCH3: *Integrity:* If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p.

BCCH4: *Consistency:* If some correct process delivers a message m with label ℓ and sender s, and another correct process delivers a message m' with label ℓ and sender s, then $m = m'$.

2. Byzantine reliable channel

Properties:

BRCH1–BRCH3: Same as properties BCCH1–BCCH3 of Byzantine consistent channel (Module 3.13).

BRCH4: *Agreement:* If some correct process delivers a message m with label ℓ and sender s, then every correct process eventually delivers message m with label ℓ and sender s. [Combine Consistency and totality]

Algorithm: Byzantine Consistent Channel

- One broadcasting per instance (seq number for every sender + seq. of instances)

④ Consensus (crash-stop only)

- Agree on common value they initially propose.
- To require consensus, a problem must lead to negative conseq. if any of) are violated.

1. Regular Consensus

Properties:

C1: *Termination:* Every correct process eventually decides some value.

C2: *Validity:* If a process decides v, then v was proposed by some process.

C3: *Integrity:* No process decides twice.

C4: *Agreement:* No two correct processes decide differently.

Algorithm:

- Oblivious (Ignor proposals)
- Eager Best-Effort (Best-effort proposals, then decide on first proposal received)
- Waiting Best-Effort (Wait and then decide, but without failure Detection no...)
- Impossibility Result →
  - Impossible to reach consensus in an asynchronous system even when only one process may crash-stop.
  - Can't have all $\frac{4}{5}$ properties.

- Flooding Consensus (fail-stop)
  - Monotonic Rounds