Part 1 Basic Concept

1. The Asynchronous Systems Model

No bound-on time to deliver a message.

No bound-on time to compute.

Clocks are not synchronized.

时钟不同步

(Consensus cannot be solved in asynchronous system if node crashes can happen)

2. The Synchronous Systems Model

Known bound-on time to deliver a message.

Known bound-on time to compute.

Known lower and upper bounds in physical clock drift rate.

时钟知道最大和最小偏差

(Consensus can be solved in synchronous system with up to N-1 crashes)

We need an accurate crash detection: every node sends a message to every node. If no msg from a node within bound, node has crashed.

3. Partially Synchronous Systems Model

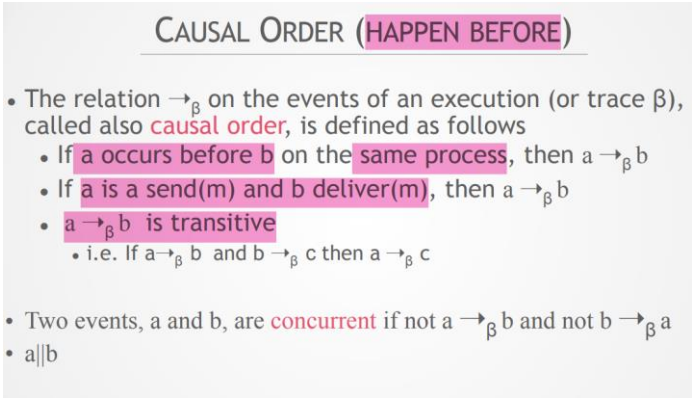
Initially system is asynchronous.

Eventually the system becomes synchronous.

(Consensus can be solved in synchronous system with up to N/2 crashes)

4. Causality

No timing assumption and reasoning model based on which events may cause other event.

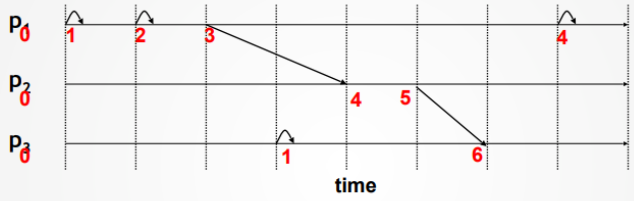


5. Computation Theorem

- **Computation Theorem**:
  - Let E be an execution (c_0,e_1,c_1,e_2,c_2,...), and V the trace of events (e_1,e_2,e_3,...)
  - Let P be a permutation of V, preserving causal order
    - P=(f_1, f_2, f_3...) preserves the causal order of V when for every pair of events $f_i \rightarrow_V f_j$ implies f_i is before f_j in P
  - Then E is similar to the execution starting in c_0 with trace P

If two executions F and E have the same collection of events, and their causal order is preserved, F and E are said to be similar executions, written F~E.

6. Lamport Clocks



Lamport logical clocks guarantee that:

If $a \rightarrow_\beta b$, then $\mathbf{t}(a) < \mathbf{t}(b)$,

if $\mathbf{t}(a) \geq \mathbf{t}(b)$, then not $(a \rightarrow_\beta b)$

Lamport logical clocks guarantee causal order

7. Vector Clocks

- $v_p \leq v_q$ iff
  - $v_p[i] \leq v_q[i]$ for all i
- $v_p < v_q$ iff
  - $v_p \leq v_q$ and for some i, $v_p[i] < v_q[i]$
- v_p and v_q are concurrent ($v_p \,||\, v_q$) iff
  - not $v_p < v_q$, and not $v_q < v_p$

[3,0,0] ≤ [3,1,0]

[3,0,0] < [3,1,0]

[3,1,0] <> [4,0,0]

- Vector clocks guarantee
  - If v(a) < v(b) then a → b, and
  - If a → b, then v(a) < v(b)
    - where v(a) is the vector clock of event a

(Strengthen the Lamport clocks for guarantee 2)

8. Failure Detectors

A failure detector can substitute timing assumption.

-Implementation idea

1. Periodically exchange heartbeat messages.
2. Timeout based on worst case message round trip
   -If timeout, then suspect process
   -If received message from suspected node, revise suspicion and increase time-out

一些构建算法时的性质！虽然大概率不会被问但还是看一下吧！

Completeness and accuracy

- **Strong Completeness**
  - Every crashed process is eventually detected by all correct processes

每个 crashed 进程最终都会被所有 correct 进程检测到

- **Weak Completeness**
  - Every crashed process is eventually detected by some correct process

at least one

每个 crashed 进程最终都会被一些（至少一个）correct 进程检测到

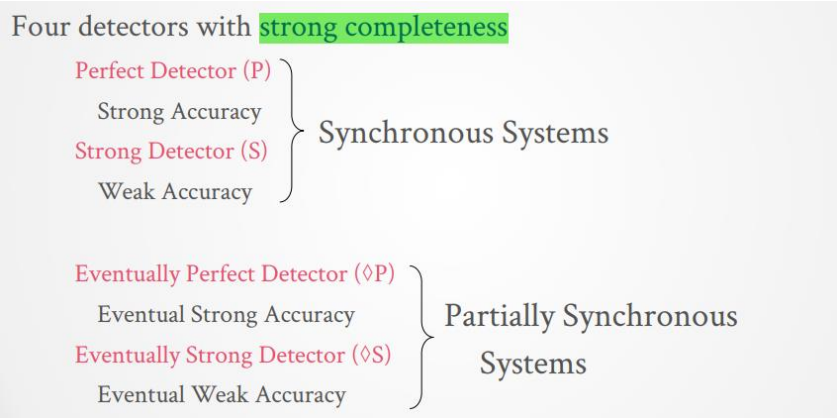
- **Strong Accuracy**
  - No correct process is ever suspected

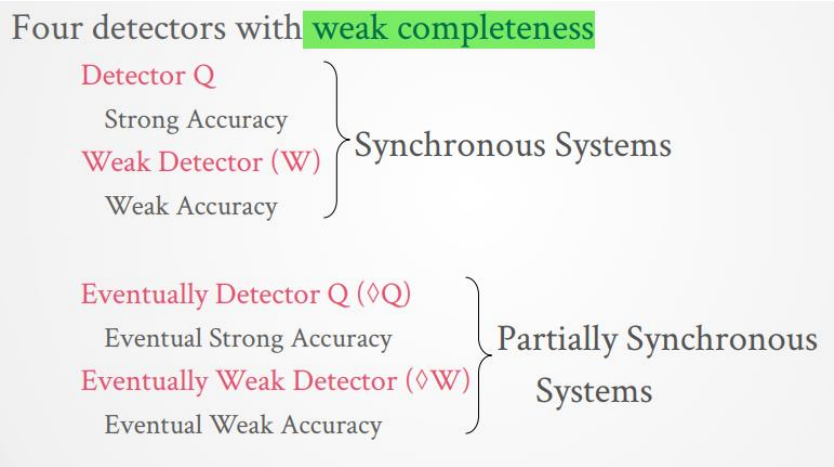
没有 correct 进程被 suspect

- **Weak Accuracy**
  - There exists a correct process which is never suspected by any process

存在一个 correct 进程从未被任何进程 suspect

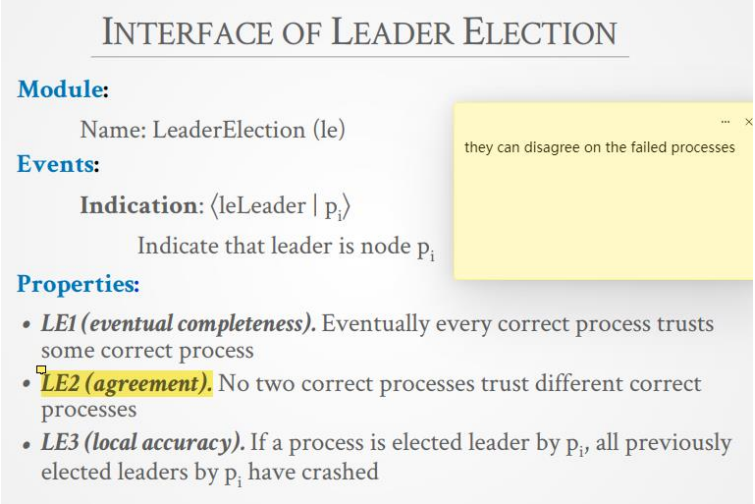
Class of failure detectors





Leader elections

LE with P (perfect failure detector) Synchronous Systems



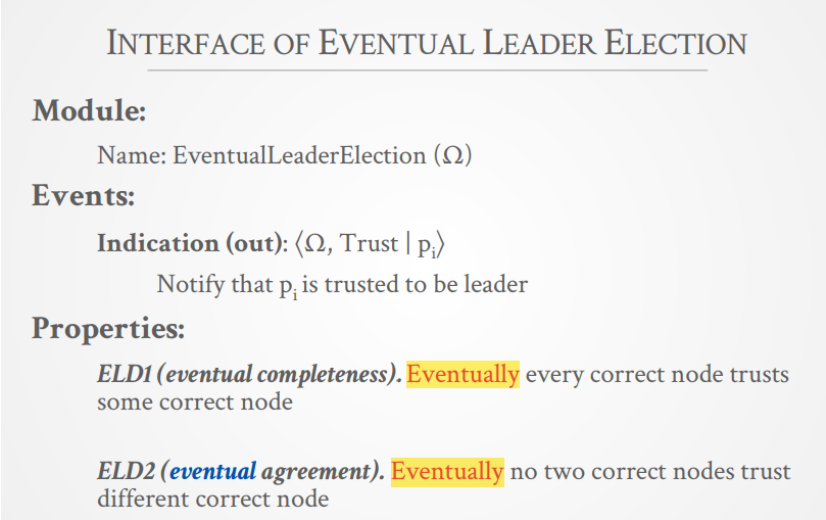
LE1：eventual completeness 最终所有正确的进程相信一些正确的进程

LE2：agreement 没有两个正确的进程相信不同的正确进程（这两个条件已经可以选出唯一节点了）

LE3：local accuracy 如果 pi 选出进程 a 作为 leader， 那么所有之前由 pi 选出的 leader 就已经 crash 了，因为 Perfect Detector 有 Strong Accuracy 所以之前选出的节点如果没有 crash 是不会被怀疑的！

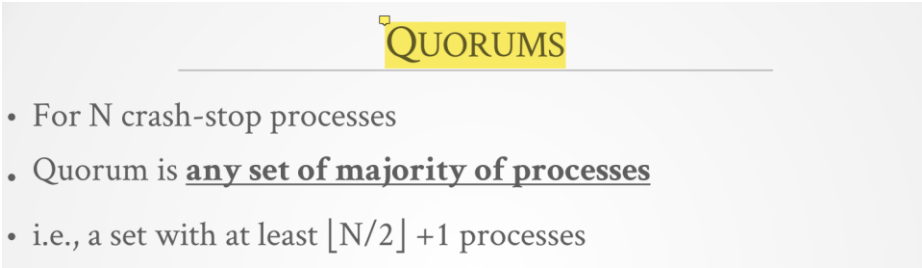
（之前考试笔记的注释，现在忘了为什么这么注释了： “说法更加 focus on 找哪些是 correct,以及对 correct 一致的 agreement”）

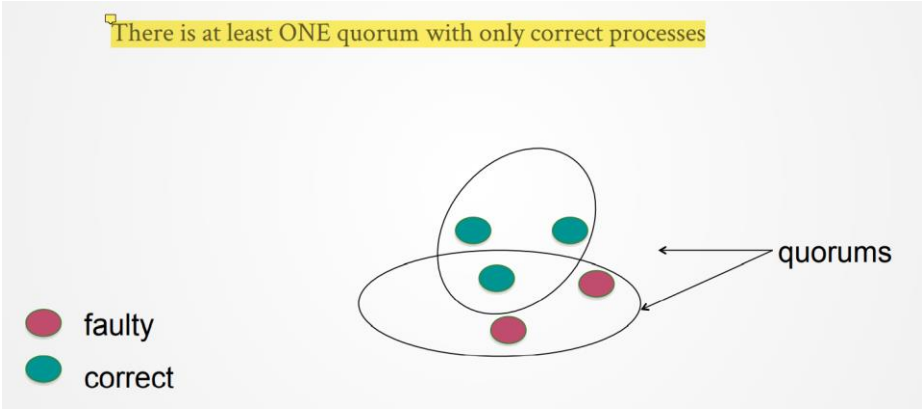
Eventual LE Ω with ◊ P (eventual perfect failure detector) Partially Synchronous Systems (Paxos 用的这个哦！)



ELD1(eventual completeness) 最终所有正确的进程相信一些正确的进程
ELD2(eventual agreement) 最终没有两个正确的进程相信不同的正确进程

Reliable Broadcast





Quorums used in Fail-Silent and Fail-Noisy algorithms; a process never waits for messages from more than [N/2] + 1 different processes!

Best-effort broadcast (Paxos 也是用的这个)

1. best-effort-validity

If p1 and p2 are correct, then any broadcast by p1 is eventually delivered by p2

2. no duplication

No message delivered more than once

3. no creation

No message delivered unless broadcast

Implementing BEB

-The bundles (fail-stop, fail-silent and fail-noisy) have to access perfect channel.

-We can just send m to all processes by perfect channel.

-The channel guarantees that if both sides are correct then the message is going to be delivered on the other side).

除了 BEB 以外还有

Reliable broadcast： **if sender crashes, ensure all or none of the correct nodes get msg.**

Uniform reliable broadcast： **if a failed node delivers, everyone must deliver, at least correct nodes.**

Paxos

Single value uniform consensus properties

- **Validity(safety):**

Only proposed values may be decided.

- **Uniform Agreement(safety):**

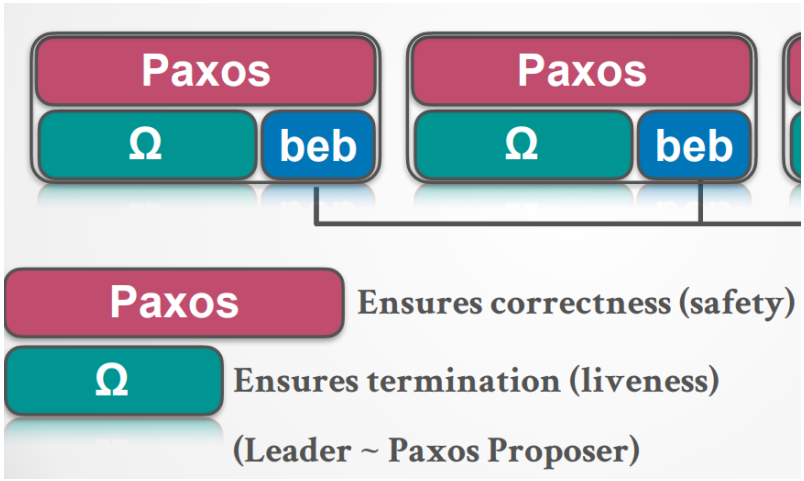
No two nodes decide differently.

- **Termination(liveness):**

Every correct node eventually decides.

- **Integrity(safety):**

Each process can decide a value at most once.



P1. An acceptor accepts first proposal it received.

(This can provide the obstruction-free progress – if a single proposer executes without interference, it makes progress; ensure obstruction-free progress and validity. And the validity).

If competition exists, we should enable restarting by distinguish proposals with unique sequence number, ballot number, this number increases monotonically.

P2. If proposal (n, v) is chosen, every higher proposal chosen has value v.

(This can ensure agreement but how to implement it?)

P2a. If v is chosen, every higher proposal accepted has value v.

(Because the acceptor cannot use the knowledge they don't have, they cannot know what has chosen. We make it stricter - we accept only things that have been chosen in the past.)

一个提案可能被没有接受过任何提案的 acceptor 可能会违反这一 condition，因为它 can not use the knowledge they don't have.

P2b. If v is chosen, every higher proposal issued has value v.

(We cannot prevent an acceptor from accepting higher value proposal, because the acceptor cannot use the knowledge they don't have as same before, so we make it stricter – only issue that have been chosen.)

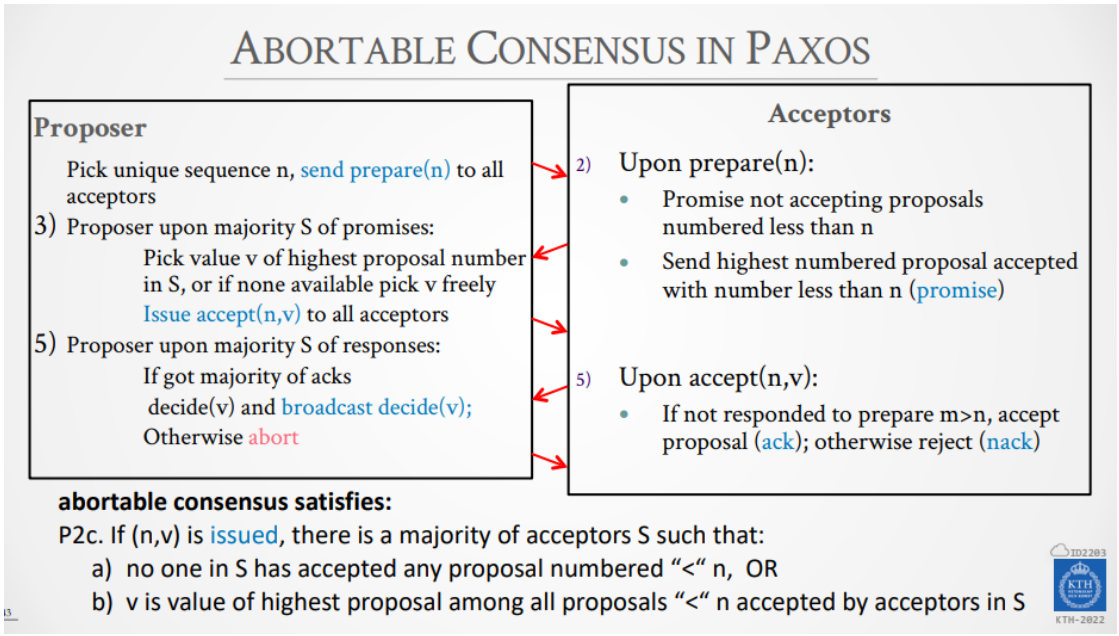
P2c. if any proposal (n, v) is issued, there is a majority set S of acceptors such that either

(a) no one in S has accepted any proposal numbered less than n. (nothing is chosen so far so we can pick any value we want)

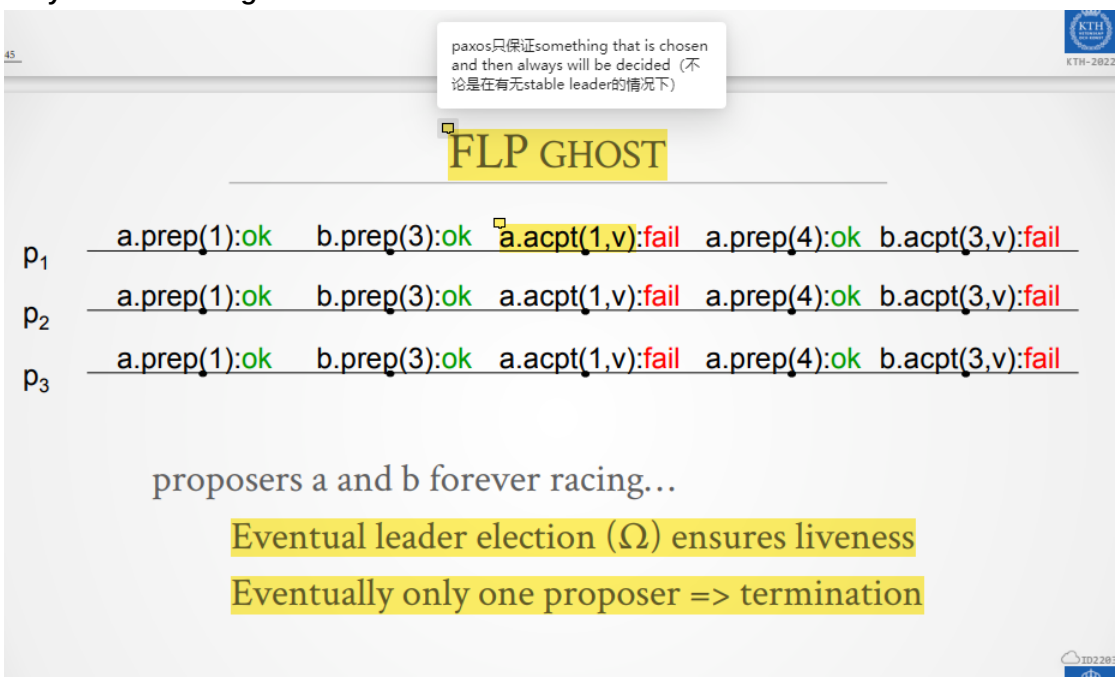
(b) v is the value of the highest proposal among all proposals less than n accepted by acceptors in S. (something is probably chosen, and I use that as my value)这里是可能决定，因为在 majority 中存在的 highest proposal 可能是被决定的，也可能并不是被决定的。

How to implement P2c?

1. **The value of the highest round number.**
2. **A promise that the state of S does not change until round n. (key of paxos)**

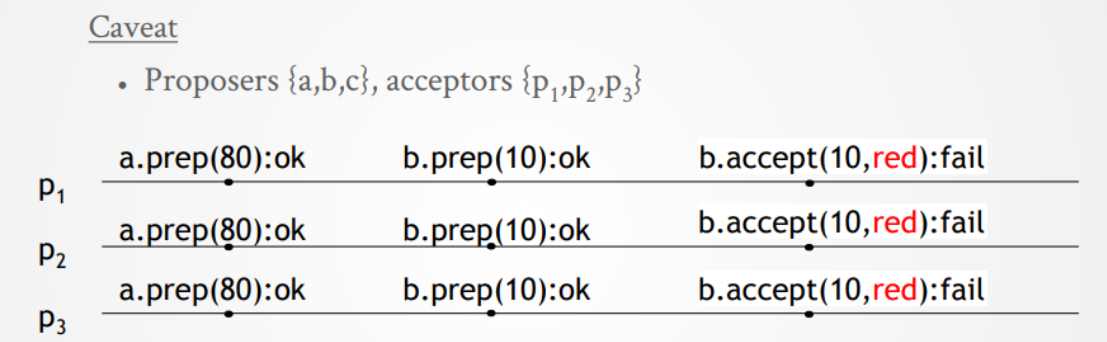


Why we need omega

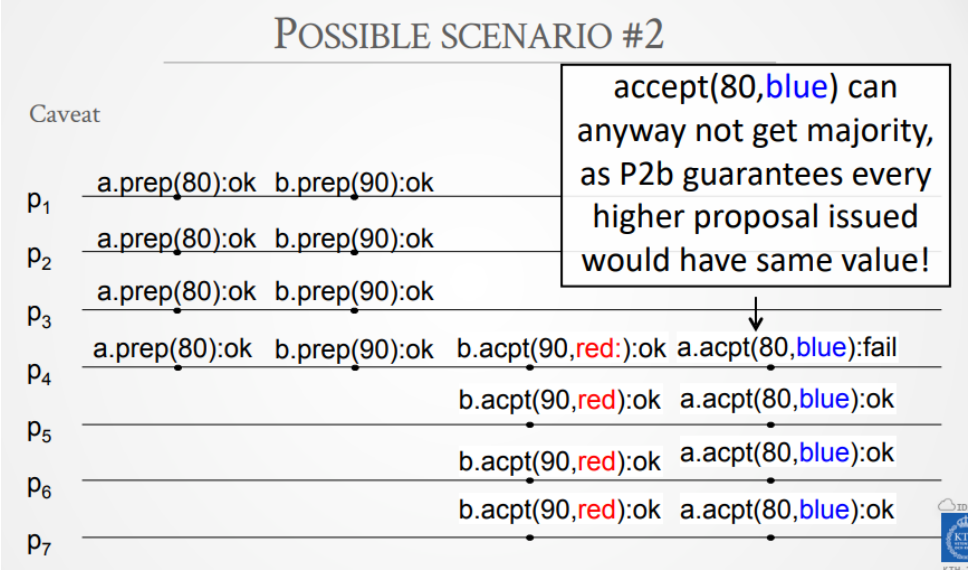


The original Paxos will have following scenario:

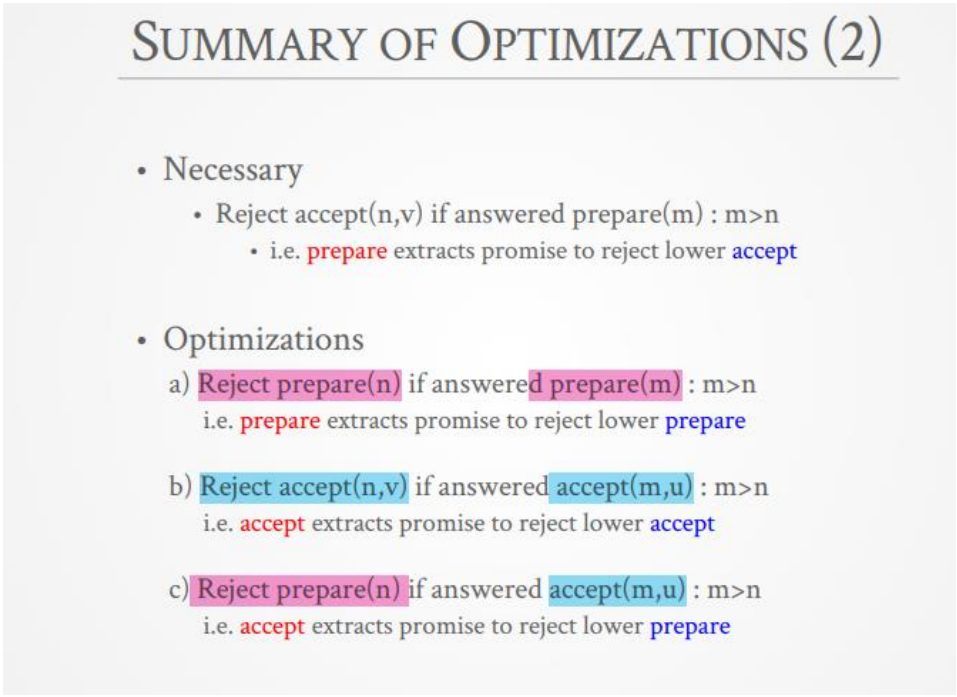
1. 海王情况：accept every prepare.接受所有 prepare

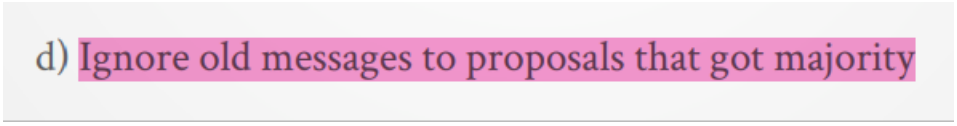


2. 渣男情况：因为没有 prepare,所以在已经 accpet 的情况下又接受 lower accept



Optimization:





Each acceptor remembers

- Highest proposal (n, v) accepted
- Highest prepare it has promised

Paxos Made Simple

比较好的翻译 https://www.cnblogs.com/snake-fly/p/12070819.html

结合着我之前的课堂 ppt 总结看!原文和翻译都看看吧!

Raft

比较好的翻译 https://www.jianshu.com/p/2a2ba021f721

看完发现忘记 Paxos 了，要哭了

Baxos： 2204.10934.pdf (arxiv.org)

-Abstract

1. Replace leader election in Multi-Paxos with random exponential backoff(REB).
2. Baxos offers more robustness to liveness and performance downgrade attacks than leader-based consensus protocols.

-Introduction

1. Most consensus protocols use leader election to handle contention and lock-free termination. However, when the network is volatile, leader-based approaches fail to deliver good performance due to leader timeouts and subsequent leader election mechanisms.
2. Previously proposed consensus algorithm without using a leader node include multi-leader-based protocols. They are also vulnerable to DDoS attacks
3. Fully asynchronous algorithm are generally complex, and do not perform well as Multi-Paxos in practice.