摘要

Basic Paxos 是分布式系统解决分布式协商问题的基本算法。很多人会忽视这个算法，认为它并不复杂且已经得到了很好的解决，但事实上算法本身在理解和解释上是存在一定困难的。 本次讲座的内容按照Heidi Howard的博士论文<<Distributed consensus revised>>的内容，意图用作者自己的理解尽可能讲述Basic Paxos算法的过程，内在思想和一些安全性上的证明。

1 单值分布式协商问题

Single valued distributed consensus is the problem of deciding a single value $v \in V$ between a finite set of n participants, Non-triviality. The decided (commit) value must have been proposed by a participant.

Q1.1 为什么我们不能规定所有acceptor都选择同一个节点呢

Q1.2 为什么不能规定所有acceptor都按照同一顺序选择proposer的提案

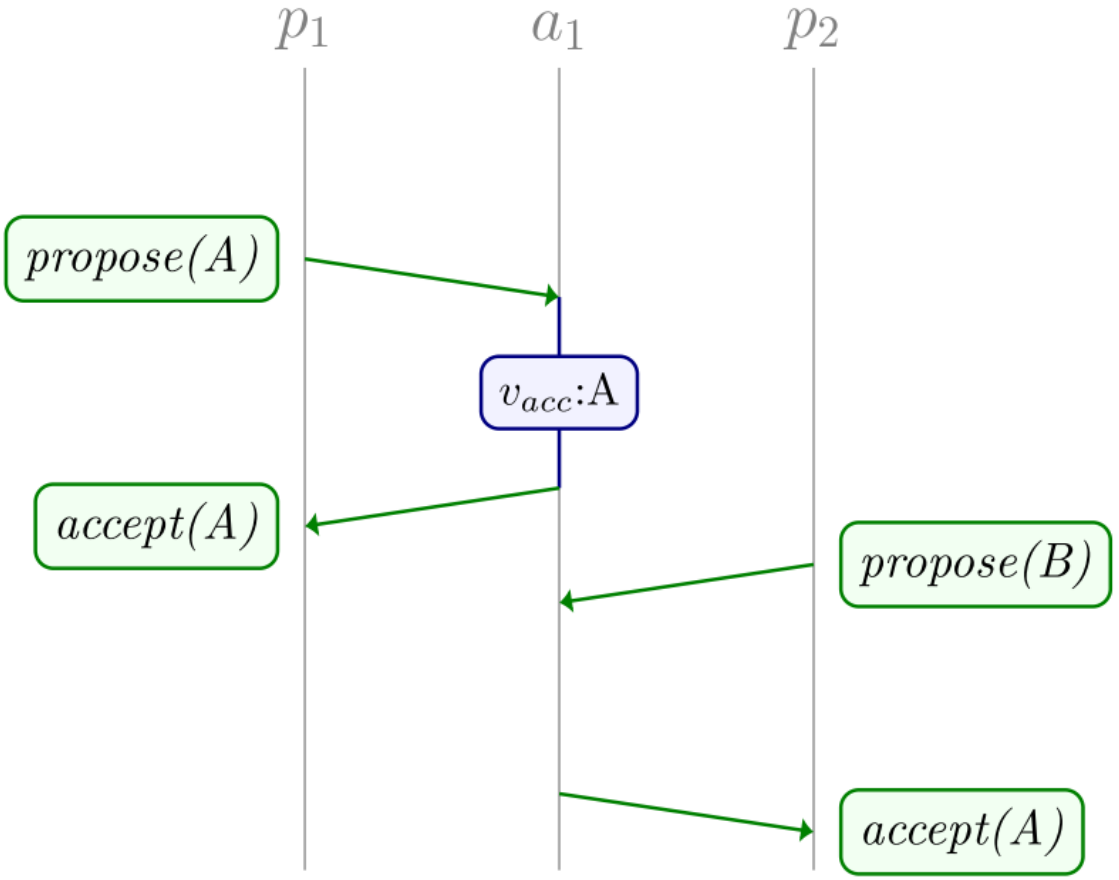
Safety. If a value has been decided, no other value will be decided.

Liveness. under certain liveness condition, it will eventually decide a value .

2 只有一个接受者的算法

idea: the acceptor determines the commit value as the first of all the proposals receieved.

example 2.1



algorithm 2.1

Algorithm 1: Proposer algorithm for SAA

state:

- γ: candidate value (configured, persistent)

1 **send** *propose*(γ) *to acceptor*
2 **case** *accept*(v) *received from acceptor*
`/* proposer learns that` v `was decided so return` v `*/`
3 **return** v

Algorithm 2: Acceptor algorithm for SAA

state:

- v_{acc}: accepted value (persistent)

1 **while** *true* **do**
2 **case** *propose*(v) *received from proposer*
3 **if** $v_{acc} = nil$ **then**
4 $v_{acc} \leftarrow v$
5 **send** *accept*(v_{acc}) *to proposer*

what are the difficulties if we have more than one acceptors ?

1. acceptors recieve proposals in different sequence?

=> Q 2.1 why proposal order is requried?

2. acceptors are not aware of the proposals identified by other acceptors

=> Q 2.2 why 2 phase algorithm is required?

3 Basic Paxos的过程

Phase 1： a proposer tries to learn the state of the system

Phase 2: a proposer tries to get a value to be accepted.

decided: if majoraty have accepted the value.

3. Q 3.1 what is the role of proposal id ?

after the value is decided?

before the value is decided?

algorithm 3.1

Classic Paxos Phase 1

1. A proposer chooses a unique epoch e and sends *prepare(e)* to the acceptors.

2. Each acceptor stores the last promised epoch and last accepted proposal. When an acceptor receives *prepare(e)*, if e is the first epoch promised or if e is equal to or greater than the last epoch promised, then e is written to storage and the acceptor replies with *promise(e,f,v)*. *(f,v)* is the last accepted proposal (if present) where f is the epoch and v is the corresponding proposed value.

3. Once the proposer receives *promise(e,_,_)* from the majority of acceptors, it proceeds to phase two. Promises may include a last accepted proposal which will be used by the next phase.

4. Otherwise if the proposer times out, it will retry with a greater epoch.

Classic Paxos Phase 2

1. The proposer must now select a value v using the following *value selection rules*:

   i If no proposals were returned with promises in phase one, then the proposer will choose its candidate value γ.

   ii If one proposal was returned, then its value is chosen.

   iii If more than one proposal was returned then the proposer must choose the value associated with the greatest epoch.

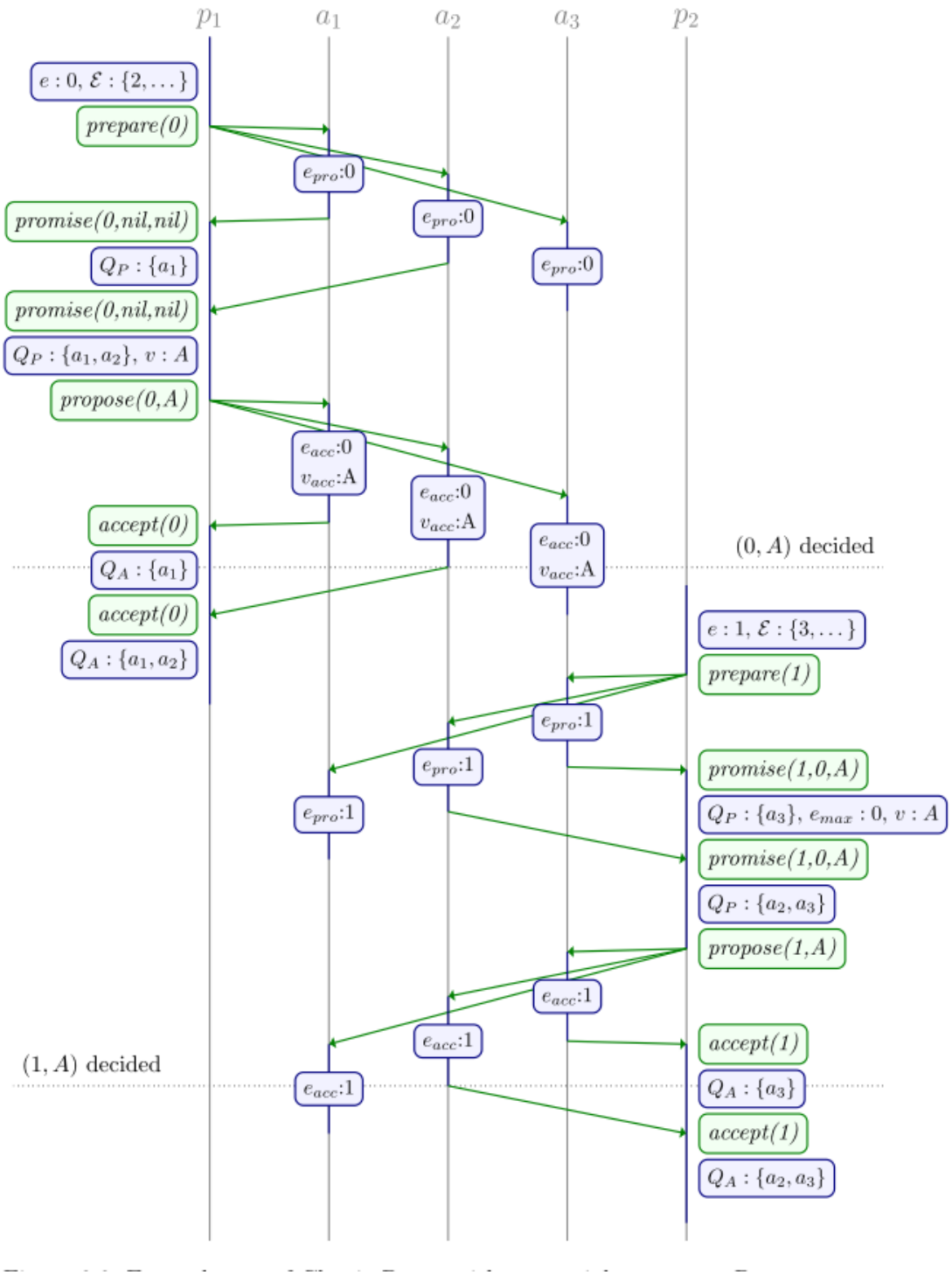
   The proposer then sends *propose(e,v)* to the acceptors.

2. Each acceptor receives a *propose(e,v)*. If e is the first epoch promised or if e is equal to or greater than the last promised epoch, then the promised epoch and accepted proposal is updated and the acceptor replies with *accept(e)*.

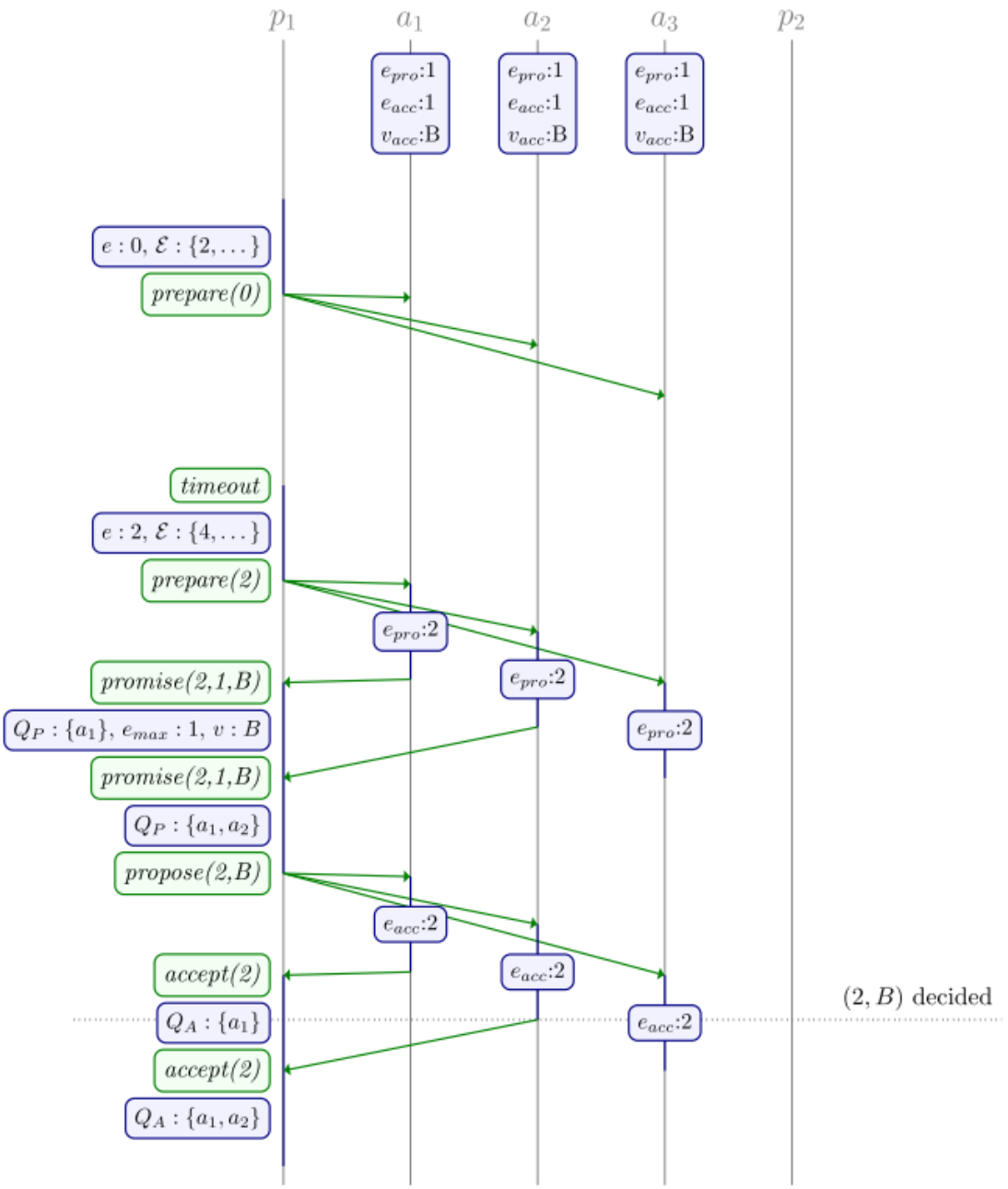
3. Once the proposer receives *accept(e)* from the majority of acceptors, it learns that the value v is decided.

4. Otherwise if the proposer times out, it will retry phase 1 with a greater epoch.

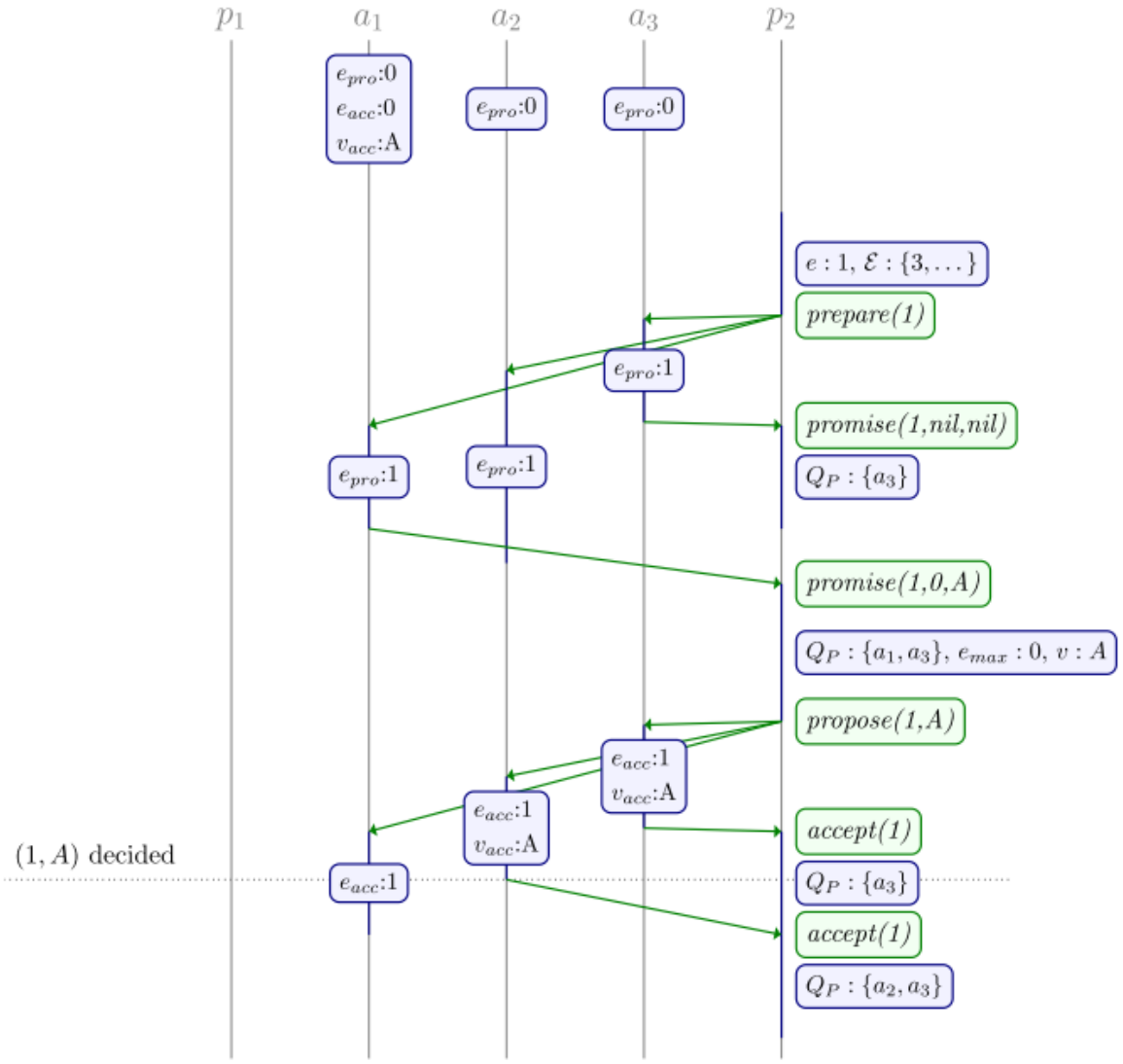
example 3.1


p_1
a_1
a_2
a_3
p_2
$e : 0, \mathcal{E} : \{2, \dots\}$
prepare(0)
e_{pro}:0
promise(0,nil,nil)
$Q_P : \{a_1\}$
promise(0,nil,nil)
$Q_P : \{a_1, a_2\}, v : A$
propose(0,A)
e_{acc}:0
v_{acc}:A
accept(0)
$Q_A : \{a_1\}$
accept(0)
$Q_A : \{a_1, a_2\}$
$(0, A)$ decided
$e : 1, \mathcal{E} : \{3, \dots\}$
prepare(1)
e_{pro}:1
promise(1,0,A)
$Q_P : \{a_3\}, e_{max} : 0, v : A$
promise(1,0,A)
$Q_P : \{a_2, a_3\}$
propose(1,A)
e_{acc}:1
accept(1)
$(1, A)$ decided
$Q_A : \{a_3\}$
accept(1)
$Q_A : \{a_2, a_3\}$

example 3.2


p_1
a_1
a_2
a_3
p_2
e_{pro}:1
e_{acc}:1
v_{acc}:B
e_{pro}:1
e_{acc}:1
v_{acc}:B
e_{pro}:1
e_{acc}:1
v_{acc}:B
$e : 0, \mathcal{E} : \{2, \dots\}$
prepare(0)
timeout
$e : 2, \mathcal{E} : \{4, \dots\}$
prepare(2)
e_{pro}:2
promise(2,1,B)
e_{pro}:2
$Q_P : \{a_1\}, e_{max} : 1, v : B$
e_{pro}:2
promise(2,1,B)
$Q_P : \{a_1, a_2\}$
propose(2,B)
e_{acc}:2
accept(2)
e_{acc}:2
$(2, B)$ decided
$Q_A : \{a_1\}$
e_{acc}:2
accept(2)
$Q_A : \{a_1, a_2\}$

example 3.3



4. 安全性的理解

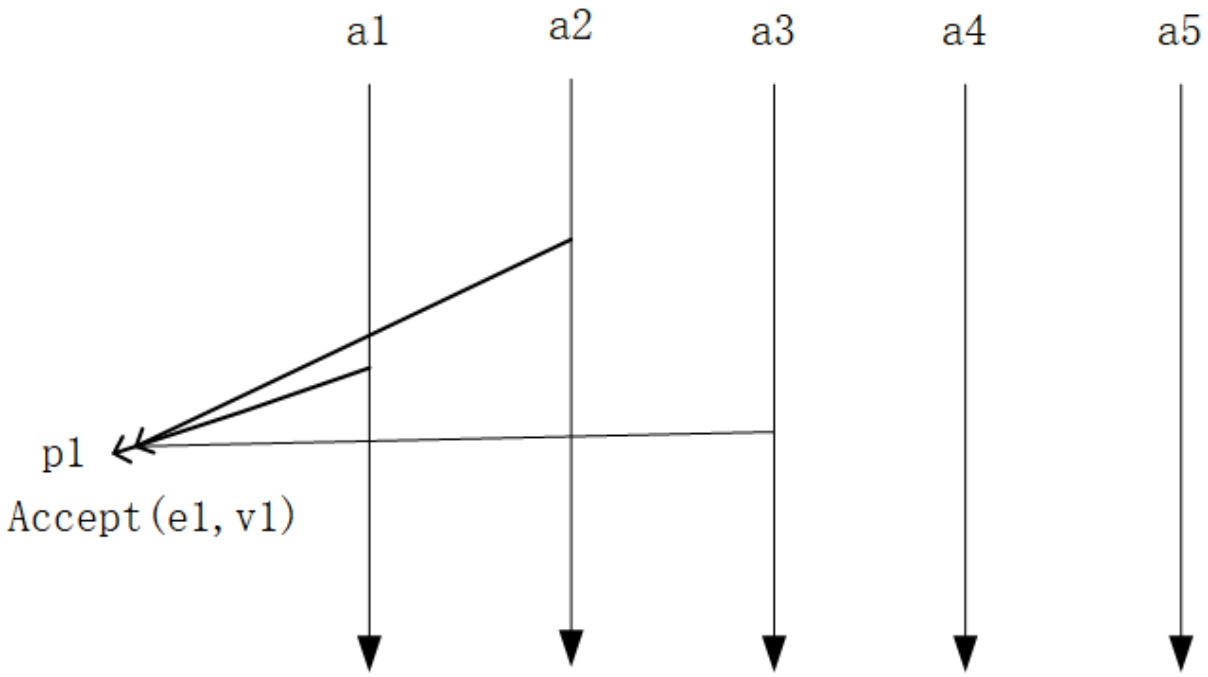
Q 4.1 how the algorithm guarantees that "If a value has been decided, no other value will be decided" ?

under the premise of:

至少一个存活的proposer 提出提案;

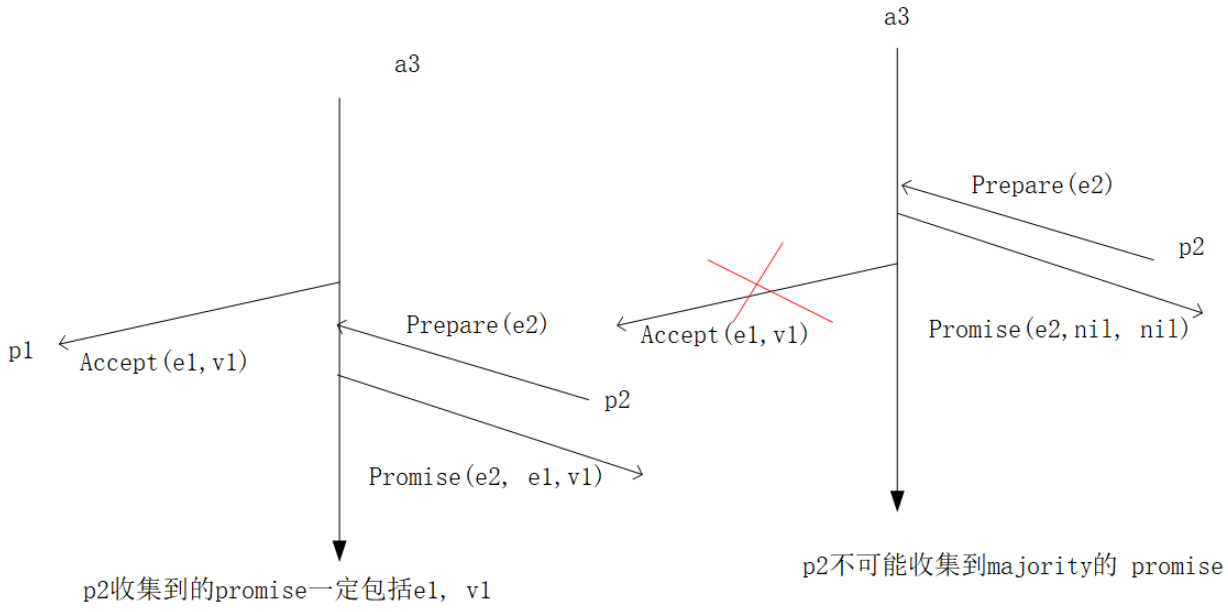
至少一半以上(majoirity) acceptor 存活;

假设如下图所示， 设p1的proposal (e1, v1) 是第一个decided的proposal.



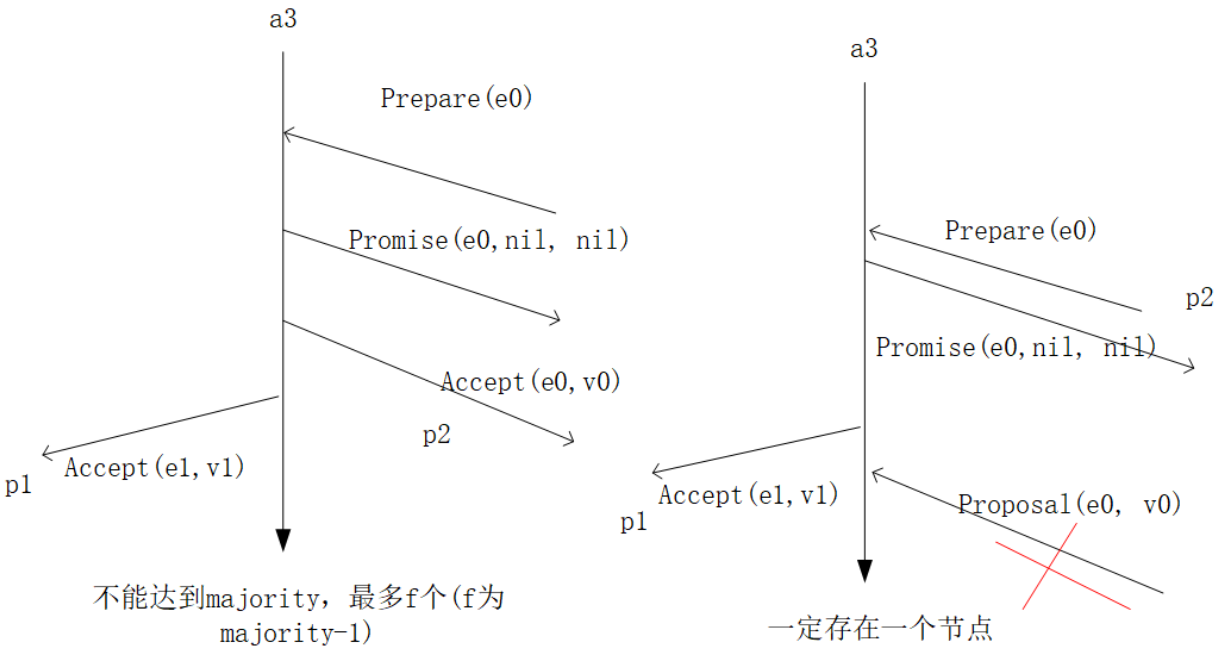
（1） 那么p1的下一个proposal p2，假设epoch 为e2, .

prepare(e2)的quorum和a1, a2, a3 必有一个交集, 假设为a3



所以比e1大的proposal不可能decide除v1以外的一个value。

（2）有没有可能比e1小的proposal（比如e0) 达成一致?



=Q4.1 再理解下epoch的作用?

如果（e1, v1)是第一个达成一致的proposal，

（1） 那么比e1小的proposal不可能达成一致，epoch的作用用来否决比e1小的proposal；

（2） 比e1大的proposal 要么不可能达成一致，要么达成一致的值为v1。

5 Basix Paxos的一些优化

1. Distinguish Proposer.

参考文献

Heidi Howard. Distributed consensus revised