# 概念术语：

## lockstep:步调一致，即同步

参考：

<https://en.wikipedia.org/wiki/Lockstep_(computing)>

Lockstep systems are fault-tolerant computer systems that run the same set of operations at the same time in parallel.[1] The redundancy (duplication) allows error detection and error correction: the output from lockstep operations can be compared to determine if there has been a fault if there are at least two systems (dual modular redundancy), and the error can be automatically corrected if there are at least three systems (triple modular redundancy), via majority vote. The term "lockstep" originates in the army usage, where it refers to the synchronized walking, in which the marchers walk as closely together as physically practical.

To run in lockstep, each system is set up to progress from one well-defined state to the next well-defined state. When a new set of inputs reaches the system, it processes them, generates new outputs and updates its state. This set of changes (new inputs, new outputs, new state) is considered to define that step, and must be treated as an atomic transaction; in other words, either all of it happens, or none of it happens, but not something in between. Sometimes a timeshift (delay) is set between systems, which increases the detection probability of errors induced by external influences

# Agreement protocol

## 参考：

1)《Advanced Concepts in Operating System》

2) <http://www.cs.fsu.edu/~xyuan/cop5611/lecture11.html>

## 定义：

1）所有节点必须就结果达成共识（agree on a value）：0或1；

（例如：decision to commit a DB transaction（要不要提交一个数据库事务）

2）仅仅通过投票的方法是不够的（因为某些节点可能向其他节点发送的不一致的投票信息）

### 系统模型：

#### 假设前提：

1. n个节点中可能有m个故障节点；
2. 系统中节点是两两连接的（fully connected , pairwise）；
3. 接收者能识别发送者的ID；
4. 通信信道是可靠的（即通信消息不会丢弃，被篡改等）
5. 节点间工作模式必须是同步的；

因为在异步方式下，agreement problem是不可解的；

### 故障模式（failure modes）：

1. crash faults（崩溃故障）；
2. omission fault(遗漏故障)
3. malicious（Byzantine） fault(恶意故障)：

节点间同步工作模式下可以检测到前两种故障。拜占庭故障可能是由于软硬故障也可能是由于恶意攻击；

### 其他问题(other issues)：

1. 节点间消息的鉴别；
2. Metrics(度量)：time, message traffic, storage overhead；

## 问题分类(taxonomy of problems)：

所有的正常节点必须对来自一个正常节点的信息达成一致（agree on value(s) from a non-faulty node）；

~~这里一致暗含两个意思：1）一致；2）有效；即有效的一致，这里有效指的是所有正常节点对单独某个正常节点的信息达成一致的必须是该正常节点的initial value；~~

这里的达成一致分成如下三种情况：

1、byzantine agreement：

三个条件：

1）一个源节点向所有其他节点广播它的初始值（initial value）（*这里“initial”或许还有原始的，真实的意思，就是说每个节点发送并且被其他节点接受到的是源节点真实的信息，而不是虚假的，被篡改过的信息*）；

2）Agreement：所有的正常节点达成一致的是同样的值（agree on the same value）；

3）Validity：指的是如果源节点是正常的，那么所有其他正常节点对给源节点所发送值达成一致的结果就是该源节点发送的Inittial value；

*综上所述，byzantine agreement要求的是所有正常节点都对某个源节点发送的消息达成一致，而且如果该源节点是正常的，还要求达成一致的结果必须是该正常源节点的initail value。*

2、consensus:

1）每个节点都向其他节点广播它的initial value；

2）Agreement：所有的正常节点达成一致的是同样的值（agree on the same value）；

3）Validity：如果每个正常源节点的initial value都是v，那么所有正常节点最终达成一致的结果必须是v；

3、interactive consistency:

1)每个节点向其他节点广播他的Initial value；

2）Agreement：所有的正常节点达成一致的是同样的值向量（agree on the same vector，即v1,v2…vn）；

3）Validity：如果第i个节点是正常节点，它的initail value记为vi，那么其他所有正常节点所达成一致的第i个值必须是vi；

以上三种问题中，byzantine agreement是最基本的一个，虽然也有直接解决其他两个问题的算法，但是一般都可以从byzantin agreement问题的算法中推导出来；

## 问题分类之--Byzantine agreement

### Impossibility results

* Byzantine agreement is impossible if *m* > (*n*1)/3 
  + e.g., (31)/3  = 0(故障节点个数超过1/3就不可解)
* Byzantine agreement is impossible with < (*m*+1) message exchanges

We will see some algorithms for solving the Byzantine agreement problem that fall within these bounds. However, we will also see that the algorithms are fairly complex. This should naturally lead one to think twice when designing a system, to see if there is a way to avoid creating situations that require agreement.

### 解决：

#### Lamport-Shostak-Pease Algorithm—no failures

##### 算法特点：

1、适用性：solves Byzantine agreement for *n*  3*m*+1 processors in the presence of *m* faulty processors。即仅适用于正常节点是故障节点个数的3倍+1的场景。

2、递归定义：recursively defined, as *OM*(*m*), *m*  0

这个算法被称为“oral message”算法，因为这个算法达成最终的解决条件的方法是通过两两节点之间发送口头消息来实现的。（This is called the ``Oral Message'' algorithm, because the conditions correspond to what we would expect if messages are delivered orally, in person, by pairwise conversations between the parties involved in the consensus.）

##### 术语定义：

1. 每个节点称为一个将军；
2. 发起agreement protocol（即发起协商的）称为commander:
3. Commander的发出的消息（value）称为命令（order）’
4. 其他接收commder发出的order的将军称为lieutenants；
5. 故障节点称为叛徒（traitors）;
6. 非故障节点称为为忠臣（loyal）;

##### 算法分析：

###### 没有traitor场景---OM(0,S)

If there are no traitors, achieving agreement is easy:

The commander i sends the proposed value v to every lieutenant j in S - {i}

Each lieutenant j accepts the value v from i

###### m个traitors场景---OM(m,S) for m > 0

*S* is the set of generals for which we want agreement.

1. The commander *i* sends a value *v* directly to every lieutenant *j*  *S*  {*i*}.
2. For each lieutenant *j*  *S*  {*i*}, let *vj* be the value lieutenant *j* receives from the commander *i*, or else be RETREAT of he receives no value. Lieutenant *j* initiates *OM*(*m*1, *S*  {*i*}) (recursively) with value *vj*, acting as commander.

*The notation vj here helps us to remember that j received the value vj from i in the previous round, and j is asking the other generals to agree on this fact. At the end of each of these recursive executions, all every loyal lieutenants j  S  {i} has agreed on a set of pairs (k,vk), one for each k  S{i}.*

1. When Step 2 has been completed by all lieutenants, each lieutenant *j* tabulates the pairs it received in Step 2 (its own pair containing the original value from its commander and the other pairs containing the values returned by its own lieutenants by the recursive invocation of *OM*(*m*,*S*{*i*})) and agrees on the value *v* = *majority* ({(*k*,*vk*) | *k*  *S* {*i*}}) that is in the majority of those pairs, to be the result of *OM*(*m*, *S*).

One feature of this algorithm that some people have found confusing is the way in which the results of the recursive algorithms are combined. That is, the values must be retained and then combined, by taking the majority, after the entire round has completed.

Another feature that some people have found confusing is that there must be an arbitrary rule, such as choosing the lower value, is to break ties. Since traitors may not send messages, there also must be a default value, such as 0, that is used for all generals from which no pair is received. Likewise, if there is no majority, a default value must be used for the result of *OM*(*m*,*S*). So long as all loyal generals agree on the tie-breaking rule and the default value, there will still be consensus among the loyal generals.

To understand this algorithm, it helps to start with the case that the commander *i* is loyal. In that case, each lieutenant *j* will receive the same value *v* from *i*. The loyal ones can simply accept the value *v* and it will not matter what the traitors do.

However, since there is no way for a lieutenant *j* to tell whether the commander *i* is traitor, one must assume that he may be a traitor. To protect against the commander sending different values to the different lieutenants, the lieutenants must hold a ballot to reach consensus on what message the commander sent to each one of them. The rest of the algorithm is the procedure for that ballot.

Since the messages are transmitted "orally" (not broadcast), the lieutenants must all exchange information about what they received in the previous round, before they can hold the ballot. The ballot would still be easy if we could trust every processor to report accurately what it received. However, we must allow fo the possibility that some lieutenants are traitors, and so will report different things to different other lieutenants. That is why we need to do a Byzantine agreement on each of the messages that was sent to a lieutenant in the previous round.

When we get to the recursive invocation of *OM*(*m*1,*S*{*i*}), it is not obvious that we have reduced the problem sufficiently to satisfy the preconditions for *OM*(*m*1,*S*{*i*}). There are two possibilities:

1. The commander *i* is a traitor. In this case, it is clear that the recursion should work. Only *m* of the lieutenants are traitors. We are assuming | *S* |  3*m*+1, so | *S* {*i*} |  3*m* > 3(*m*1)+1. It follows that *OM*(*m*1, *S*{*i*}) can achieve Byzantine agreement on the message "*i* sent *j* the value *v*" among the loyal lieutenants in *S*{*i*}.
2. Processor *i* is a traitor. In this case, it is not so clear that the recursion should work. We have reduced the number of processors in the consensus by one, but there may remain *m* processors in *S*{*i*}. If *m* is the number of traitors, how can we get away with using *OM*(*m*1,...)?

The second case is dealt with by the Validity Lemma, which is stated and proven below. This lemma guarantees that if the commander is loyaal, *O*(*m*,*S*) can tolerate up to *k* traitors if | *S* |  2*k*+*m*. We will explain this lemma in more detail below, using the original theorems and proofs of Lamport, Shostak, and Pease.

###### Byzantine Agreement Conditions

1. Agreement: All loyal generals agree on the same value.
2. Validity: If the commander is loyal, then the common agreed upon value for all loyal lieutenants is the initial one given by the commander.

###### 有效性引理（validity lemma）

**Lemma:** For any *m* and *k*, *OM*(*m*,*S*) satisfies the Validity Condition if there are more than 2*k*+*m* processors and at most *k* of them are traitors.

**Proof:**

The proof is by induction on *m*. As a basis for the induction, we consider the case of *OM*(0). The Validity Condition only specifies what must happen if the commander is loyal. It is easy to see that if the commander is loyal *OM*(0) satisfies the Validity Condition, since all the processes get the same value *v* and agree upon that. We therefore can assume the theorem is true for *OM*(*m*1) and prove that is tis true for *OM*(*m*), *m* > 0.

For the induction step, we have *m*  1. In Step 1, the loyal commander *i* sends a value *v* to all the other processors. At Step 2, each loyal lieutenant *j* applies *O*(*M*1,*S*{*i*}). Since we are assuming that | *S* | > 2*k* + *m*, we have | *S* {*i*} | > 2*k* + (*m*1), so we can apply the induction hypothesis to conclude that every loyal lieutenant agrees on the value *vj*=*v* for each invocation of *OM*(*m*1,*S*{*i*}) by a loyal commander *j*. Since there are atmost *k* traitors, and | *S* {*i*} | > 2*k* + (*m*1) > 2*k*, a majority of the lieutenants in *S* {*i*} are loyal. Hence, when each lieutenant gets to Step 3 it will find a majority of the other lieutenants support the value *v*, and so it will agree to the value *v*. This confirms the Validity Condition.

###### Agreement 定理(Theorem)

**Theorem:** For any *m*, *OM*(*m*,*S*) satisfies the Validity and Agreement Conditions if there are more than 3*m* generals and at most *m* of them are traitors.

**Proof:**

The proof is by induction on *m*, similar to that of the Validity Lemma. As a basis for the induction, we consider the case of *OM*(0). If there are no traitors, it is easy to see that *OM*(0) satsfies the Validity and Agreement Conditions. We therefore can assume the theorem is true for *OM*(*m*1) and prove that it is true for *OM*(*m*), *m* > 0.

For the induction step, have *m*  1. We consider two cases, depending on whether the commander is a traitor.

1. Suppose the commander *i* is loyal. By taking *k* equal to *m* in the Validity Lemma, we see that *OM*(*m*) satisfies the Validity Condition. Moreover, since we are assuming the commander is loyal the Agreement condition is also satisfied.
2. Suppose the commander is a traitor. At most *m*1 of the lieutenants can be traitors. Since there are more than 3*m* processes, there are more than 3*m*1 > 3(*m*1) processes in *S*{*i*}. We may therefore apply the induction hypothesis to conclude that *OM*(*m*1) satisfies the Agreement and Validity Conditions. Hence, any two loyal lieutenants get the same vector of values *v*1,,*vn*1, and therefore obtain the same value *majority*(*v*1,,*vn*1) in Step 3, proving the Agreement Condition.

###### Message Complexity

* *T*(0,*n*) = *n*1
* *T*(*m*,*n*) = (*n*1)*T*(*m*1,*n*1), for *m* > 0
* *T*(*m*,*n*) = (*n*1)(*n*2)(*n*3)(*n**m*1)       *O*(*nm*)

# 论文：拜占庭将军问题（the Byzantine Generals Problem）

作者:Leslie lamport, Robert shostak, and marshall pease

sri internation

#### 后记：

##### 本论文主要内容的介绍：

参考：

<https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/>

The main reason for writing this paper was to assign the new name to the problem. But a new paper needed new results as well. I came up with **a simpler way to describe the general 3n+1-processor algorithm**. (Shostak’s 4-processor algorithm was subtle but easy to understand; Pease’s generalization was a remarkable tour de force.) **We also added a generalization to networks that were not completely connected**. (I don’t remember whose work that was.) **I also added some discussion of practical implementation details.**

1. 对general 3n+1处理器算法的更简单的描述；
2. 增加了在非完全连接的网络环境中的该算法的通用描述；
3. 以及算法实现上的一些细节的讨论;

#### 摘要：

计算机系统的故障中有一种是：存在故障部件，该部件的故障表现是其输出给系统其它部件的信息是不一致的（要求是一致的）。而一个可靠的计算机系统必须能处理这种情况并正常对外提供服务（即CAP中的P的一种表现）。

这种故障可以抽象的描述为拜占庭将军问题：

#### 概述：

# 附录：

## Byzantine general problems总结：

#### 前提：

1、将军之间通信的信道可靠可信；

2、叛徒将军个数没有多到问题不可解；

#### 问题：

1、判断将军的个数，以便确定问题是否可解？

#### 目标：

在忠诚将军之间达成**正确的**进攻与否的**一致**意见。

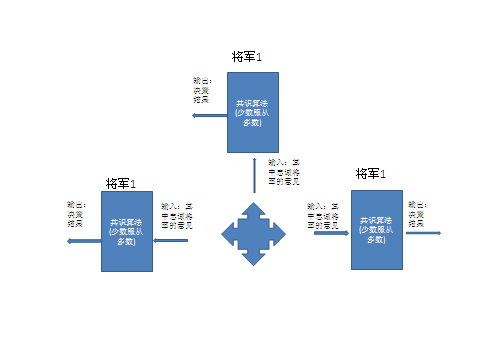
两个目标：

“一致”：这是基本目标；

“正确”：指的最终达成的一致结果应该是最大化忠诚将军利益的。（条件有利时，应该达成进攻的意见，条件不利时，应该达成撤退的意见）

如何达成“正确”：

要达成“正确”的目标，关键是每个忠诚将军能表达自己的真实意见，而不会受到叛徒意见的误导。而忠诚将军表达的真实意见是基于现有战场环境数据做出的研判，而战场环境数据里面同时包含了真伪数据，需要识别伪数据；



共识达成方法：少数服从多数；

1、每个忠诚将军表达真实意见（不会受环境伪数据）

2、所有忠诚将军接受的其他将军的指令集合（准确来说是忠诚将军的指令集合）相同；

然后根据共识算法才能产生相同的指令输出；

3、每个忠诚将军必须能采纳其他忠诚将军的指令，而不会误认为是叛徒；

归根结底，每个忠诚将军需要从输入的指令集合中识别出叛徒的指令；