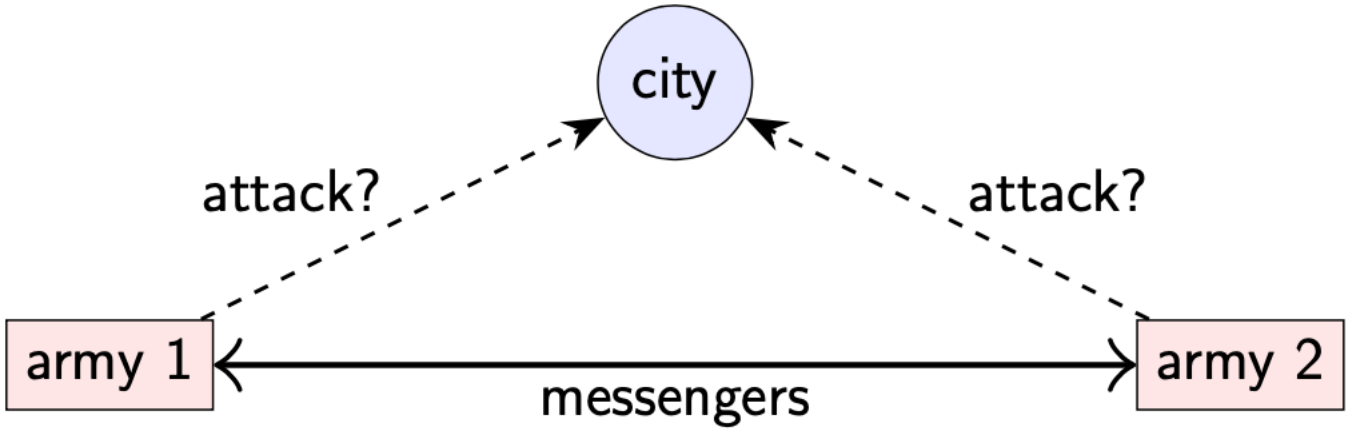
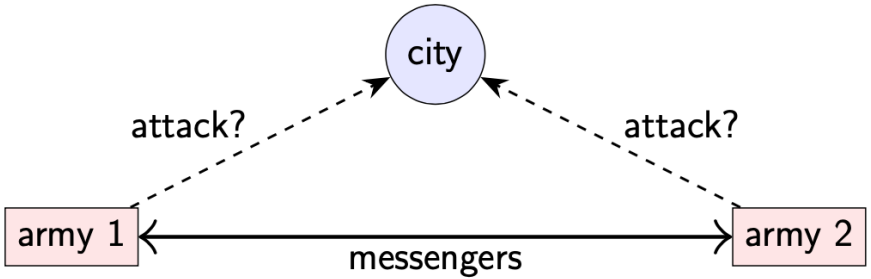
分布式系统模型

赵来平
天津大学软件学院

两个将军问题



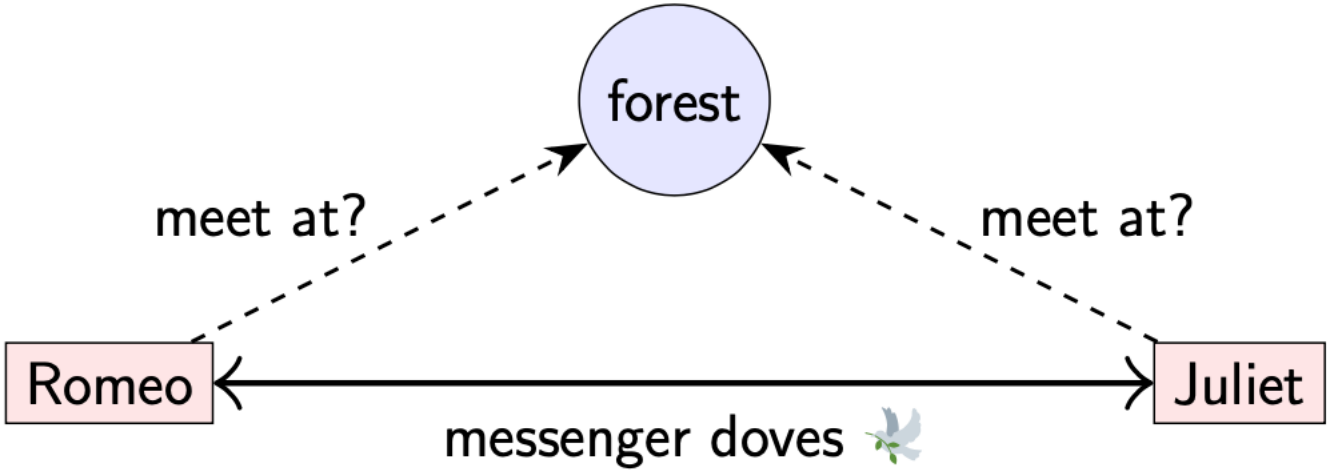
两个将军问题



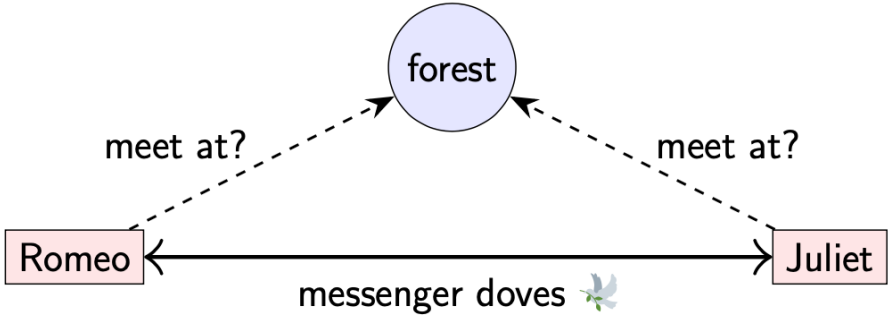
army 1	army 2	outcome
does not attack	does not attack	nothing happens
attacks	does not attack	army 1 defeated
does not attack	attacks	army 2 defeated
attacks	attacks	city captured

Desired: army 1 attacks *if and only if* army 2 attacks

罗密欧与朱丽叶的问题



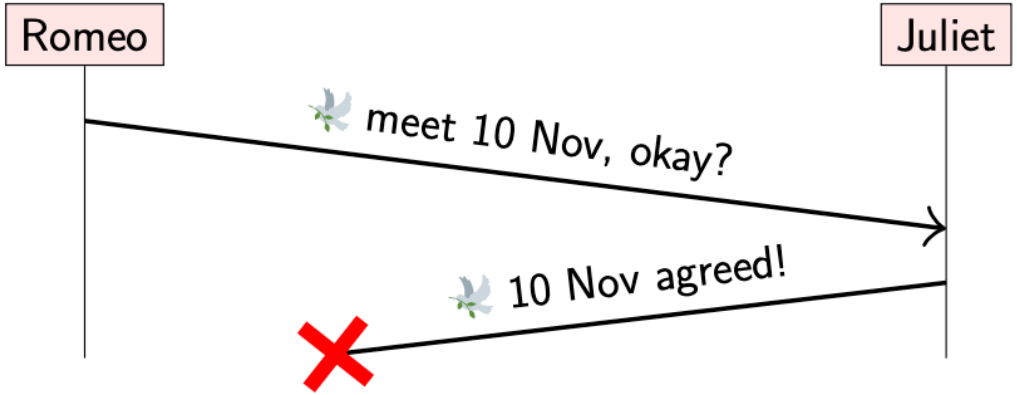
罗密欧与朱丽叶的问题



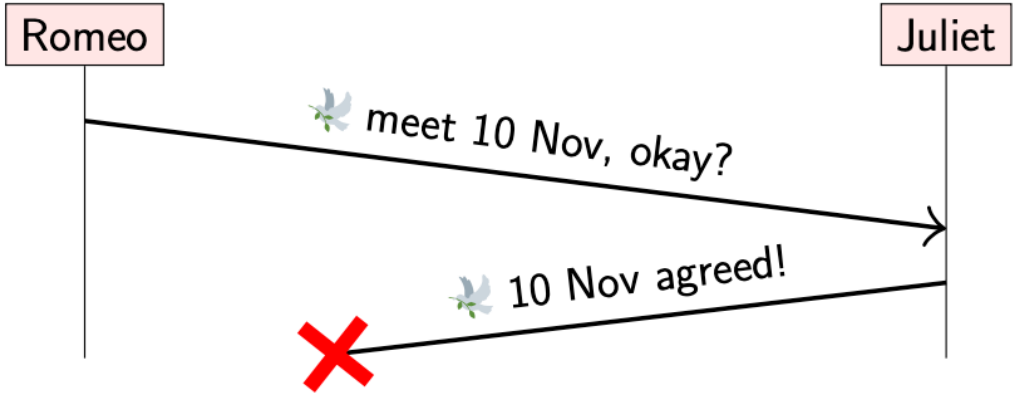
Romeo	Juliet	outcome
does not go	does not go	nothing happens
goes	does not go	Romeo gets desperate
does not go	goes	Juliet gets desperate
goes	goes	happy ever after

Desired: Romeo goes to the forest *if and only if* Juliet goes

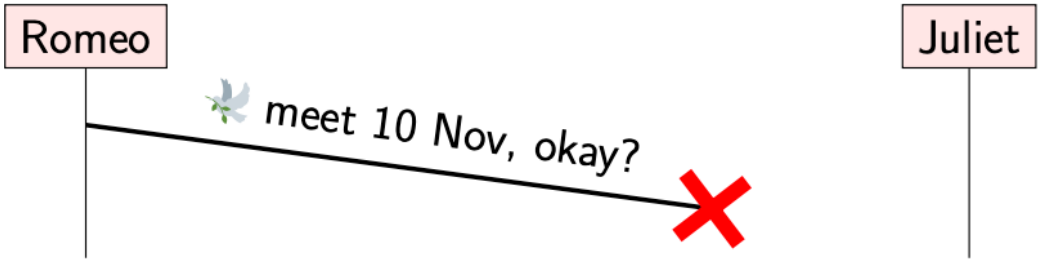
如何在消息可能丢失时仍达成一致？



如何在消息可能丢失时仍达成一致？



- 在罗密欧眼中，上面和下面的情形没有任何区别！



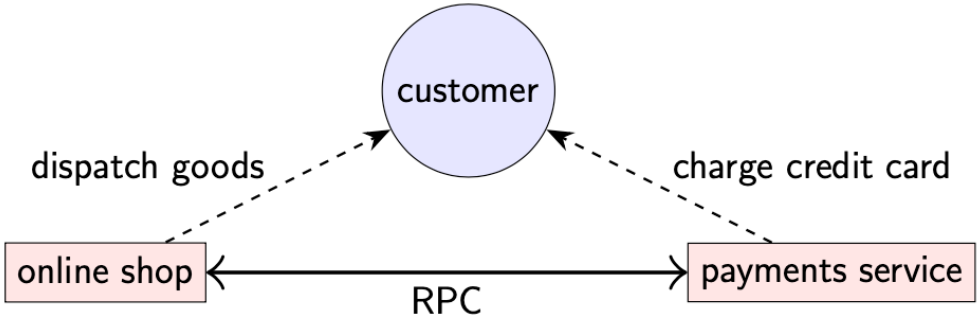
如何在消息可能丢失时仍达成一致？

- **不管有没有收到回复，罗密欧都去森林？**
  - 多次重复发消息以提高消息传递成功的概率
  - 如果消息全部丢失，朱丽叶将得不到消息，罗密欧孤独寂寞冷
- **只有收到朱丽叶的确认回复，罗密欧才去森林？**
  - 罗密欧将不再孤单
  - 但是，在朱丽叶的心中，她需要判断她的确认回复是否被罗密欧收到
  - 如果是这样，朱丽叶将面临上面罗密欧相同的问题

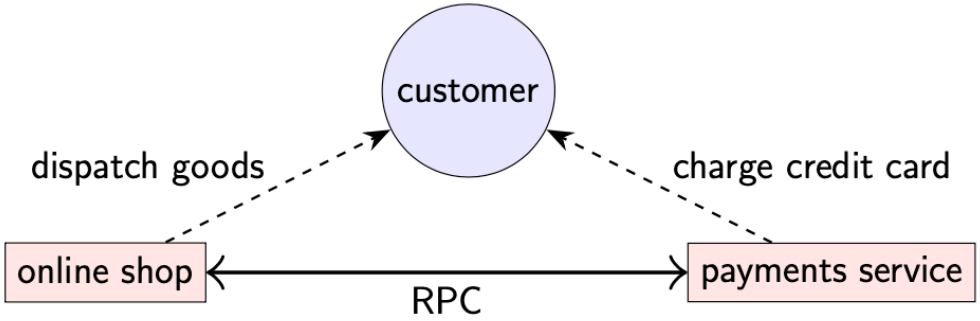
如何在消息可能丢失时仍达成一致？

- **不管有没有收到回复，罗密欧都去森林？**
  - 多次重复发消息以提高消息传递成功的概率
  - 如果消息全部丢失，朱丽叶将得不到消息，罗密欧孤独寂寞冷
- **只有收到朱丽叶的确认回复，罗密欧才去森林？**
  - 罗密欧将不再孤单
  - 但是，在朱丽叶的心中，她需要判断她的确认回复是否被罗密欧收到
  - 如果是这样，朱丽叶将面临上面罗密欧相同的问题
- **如果不能心有灵犀，那就只能加强沟通交流！**

两个将军问题的具体应用



两个将军问题的具体应用



online shop	payments service	outcome
does not dispatch	does not charge	nothing happens
dispatches	does not charge	shop loses money
does not dispatch	charges	customer complaint
dispatches	charges	everyone happy

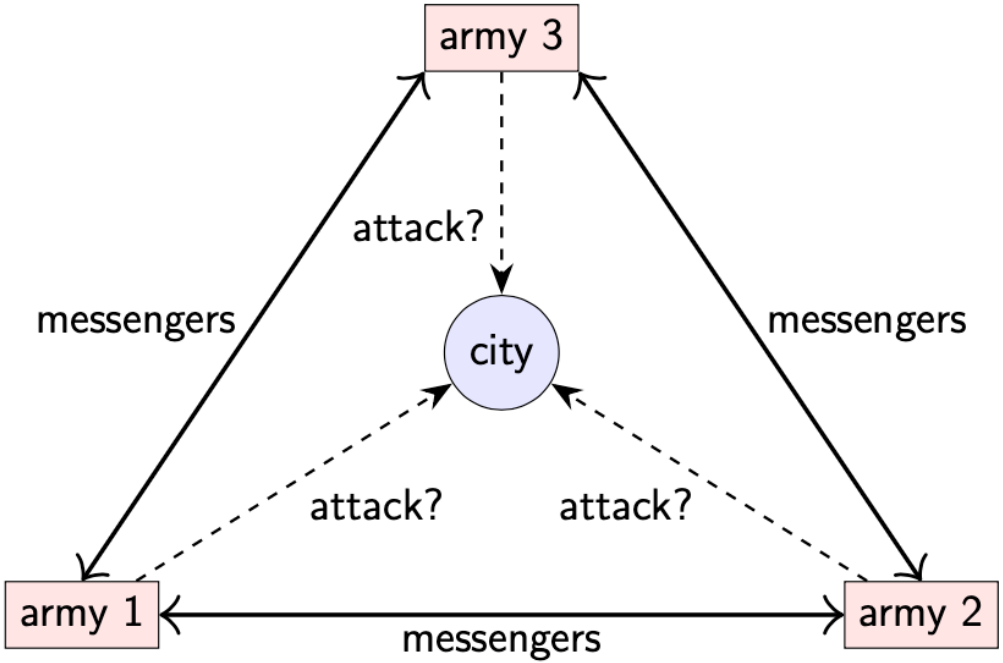
- **在线商店只在客户付完款之后才会发货！**

两个将军 ≠ 在线购物

- **更仔细地分析，我们发现网上购物并非如此就像两位将军一样。**
- **网上购物可以使用以下协议：**
  - 1. 尝试信用卡扣款
  - 2. 如果扣款成功，去发货
  - 3. 如果发货失败（比如：缺货），则退款

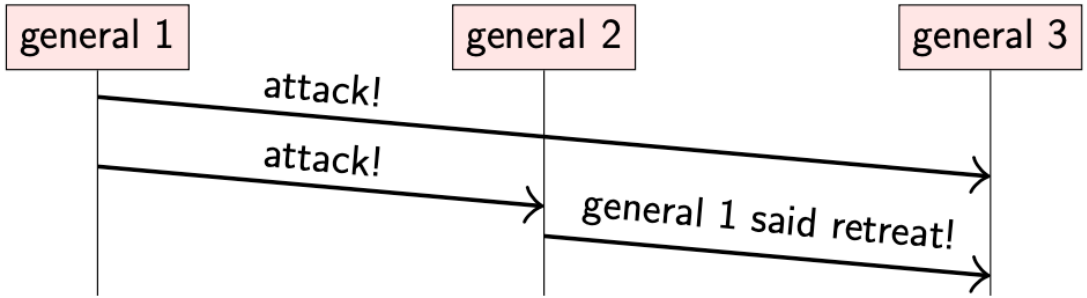
- 这里由于可执行退款操作，使得问题可解决
- 但是军队发起进攻是不可撤销的
- 商品发货的操作也是不可撤销的

拜占庭将军问题

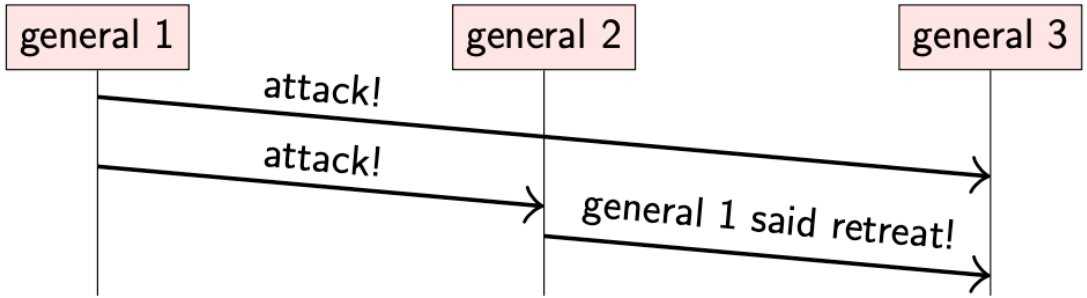


- 难点：三个将军之中，可能有内奸（编造谎言）

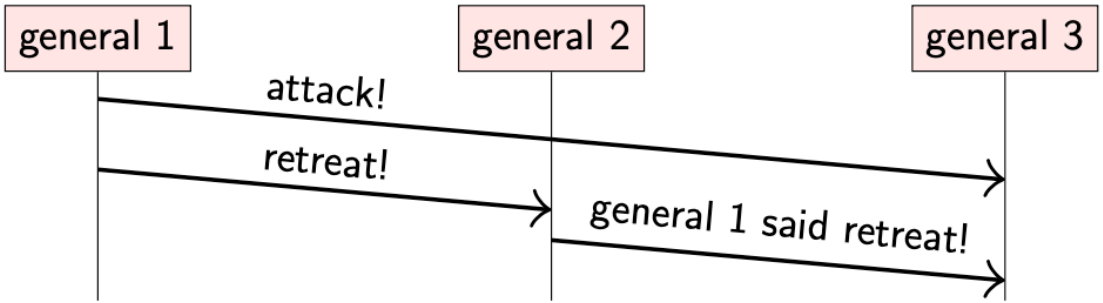
将军可能撒谎


general 1
general 2
general 3
attack!
attack!
general 1 said retreat!

将军可能撒谎



- 从将军3来看，上面和下面其实一样



拜占庭将军问题

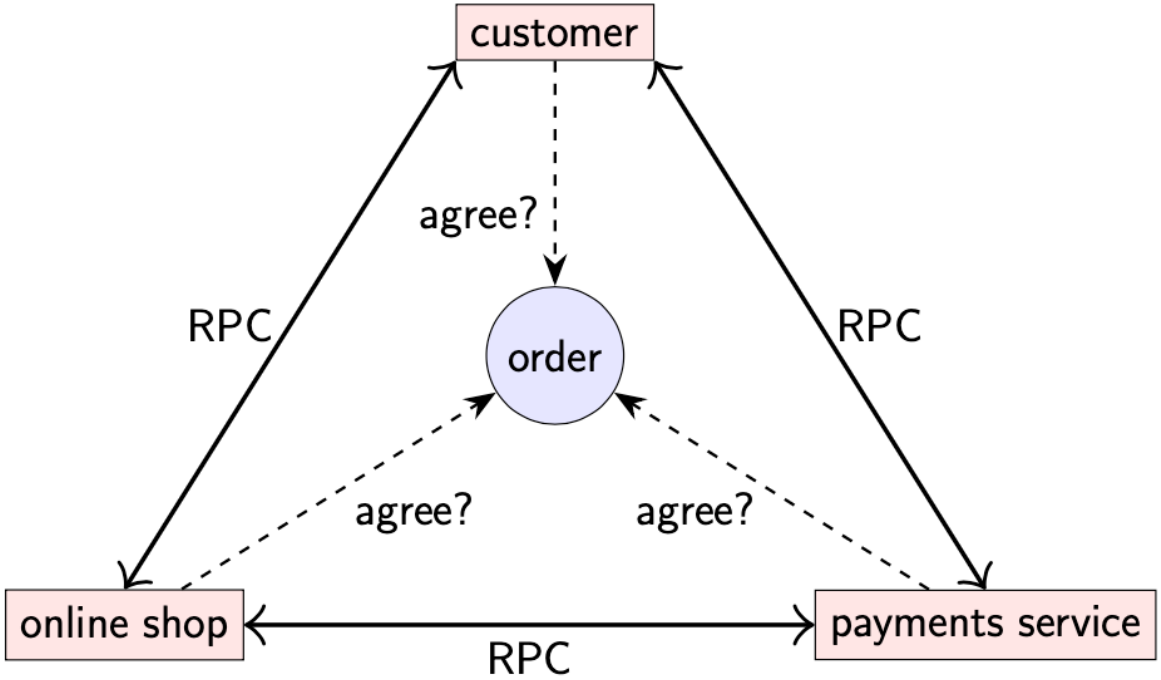
- 每个将军都可能是坏人
- 最多有 f 个将军是坏人
- 诚实的将军并不知道谁是坏人
- 坏人与坏人可能会串通一气
- 然而，诚实的将军必须同意计划

拜占庭将军问题

- 每个将军都可能是坏人
- 最多有 f 个将军是坏人
- 诚实的将军并不知道谁是坏人
- 坏人与坏人可能会串通一气
- 然而，诚实的将军必须同意计划

- 定理：如果有 f 个坏蛋，至少需要3f+1个将军参与决策才有可能不会作出错误决策（坏人占比<1/3）
- 需要用到密码学保证信息不会被篡改，但是问题仍然存在！

信任关系与恶意行为



- 谁是可信任的？

拜占庭帝国（公元650年）

Source: https://commons.wikimedia.org/wiki/File:Byzantiumby650AD.svg

- “拜占庭”一词长期以来被用来表示“过于复杂， 官僚的，狡猾的”（例如“拜占庭式的税法”）

系统模型

- **上面2个思想实验的启示**
  - 两个将军问题：网络模型
  - 拜占庭将军问题：节点行为模型
- **在真实系统中，节点和网络都可能出错！**

系统模型

- **上面2个思想实验的启示**
  - 两个将军问题：网络模型
  - 拜占庭将军问题：节点行为模型
- **在真实系统中，节点和网络都可能出错！**
- **对系统模型的假设需要非常明确**
  - 网络行为（比如：假设可丢失数据包）
  - 节点行为（比如：假设可能故障）
  - 时间行为（比如：对延迟有约束假设）
- **需要对每个环节都明确**

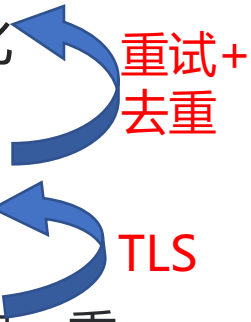
网络并不可靠

- 海底光缆被鲨鱼咬断！
- 陆地光缆被牛踩断

https://www.theguardian.com/technology/2014/aug/14/google-undersea-fibre-optic-cables-shark-attacks

系统模型：网络行为假设

- **两个节点间的双向点到点通信的三种行为：**
  - 可靠链路：消息只要正常发出，就会被收到，顺序可能变化
  - 丢包链路：消息可能会丢失，复制或者被改变顺序，如果重复发送的话，最终会被收到
  - 恶意链路：坏蛋可能会篡改消息（窃听，修改，删除，欺骗，重播）
- **网络分区：**部分链路长时间丢弃或延迟所有消息



系统模型：节点行为假设

- **每个节点执行指定算法，并假设以下情况之一：**
  - 崩溃停止（Crash-stop/fail-stop）：节点若崩溃（无论何时）即视为故障，崩溃后永久停止执行。
  - 崩溃恢复（Crash-recovery/fail-recovery）：节点可能随时崩溃并丢失内存状态，但后续可能恢复执行，磁盘数据在崩溃后仍保留。
  - 拜占庭（Byzantine/fail-arbitrary）：节点若偏离算法即视为故障，故障节点可能执行任意行为（包括崩溃或恶意操作）。

非故障节点被称为"正确"节点

系统模型：对时间假设

假设网络和节点满足以下三种情况之一：

- **同步**（Synchronous）：消息延迟存在一个已知的上界，不会超过该上界。各节点以已知的速度执行算法。
- **部分同步**（Partially synchronous）：系统在某些有限但未知的时间段内是异步的，在其他时间段内是同步的。
- **异步**（Asynchronous）：消息可能被任意长时间延迟。节点的执行可能被任意长时间暂停。完全没有任何时间方面的保证。

在实践中违反同步

- **网络的延迟通常是相当可预测的，但有时会升高，例如：**
  - 消息丢失，需要重传
  - 拥塞/争用导致排队等待
  - 网络/路由重新配置
- **节点通常以相对可预测的速度执行代码，但也会出现偶发性的暂停，例如：**
  - 操作系统调度问题，如优先级反转
  - “停止世界”（stop-the-world）式的垃圾回收暂停
  - 缺页异常、交换（swap）、抖动（thrashing）
- **实时操作系统（RTOS）可以提供调度方面的时间保证，但大多数分布式系统并不使用 RTOS。**

系统模型总结

- 对于下面三个方面中的每一个，都需要做出一种假设（从各自的选项中选一个）：
- 网络（Network）：
  reliable（可靠）、fair-loss（丢包）、或 arbitrary（恶意行为）
- 节点（Nodes）：
  crash-stop（崩溃停止）、crash-recovery（崩溃恢复）、或 Byzantine（拜占庭式故障）
- 时间（Timing）：
  synchronous（同步）、partially synchronous（部分同步）、或 asynchronous（异步）

- 这是设计任何分布式算法时的基础。
  如果你的这些假设是错的，那后面的所有结论都不再可靠

故障检测

- 故障检测器（Failure detector）：
  用于检测其他节点是否发生故障的一种算法。
- 完美故障检测器（Perfect failure detector）：
  当且仅当某个节点已经崩溃时，才将该节点标记为故障节点。
- 典型的 crash-stop / crash-recovery 场景下的实现方式：
  发送一条消息，等待对方响应；
  如果在设定的超时时间内没有收到回复，就把该节点标记为已崩溃。
- 问题在于：
  故障检测器无法区分以下情况：节点确实已经崩溃；节点只是暂时无响应；消息在传输过程中丢失；消息只是被延迟了

故障检测和部分同步

- **基于超时的完美故障检测器只存在于这样一种系统中：同步的、crash-stop 节点模型，并且通信链路可靠。**
- **最终完美故障检测器（Eventually perfect failure detector）：**
  - 可能会在某段时间内把一个实际上正常的节点误标记为“已崩溃”；
  - 也可能会在某段时间内把一个实际上已经崩溃的节点误标记为“正常”；
  - 但最终，它会做到：当且仅当节点真的已经崩溃时，才将其标记为“已崩溃”。
- **这反映了一个事实：故障检测并不是瞬时完成的，而且我们可能会遇到一些“虚假超时”（spurious timeouts）。**

- **PPT部分内容来自于剑桥大学**
  cst.cam.ac.uk/teaching/2526/ConcDisSys/