时间、时钟与事件排序

赵来平
天津大学软件学院

一则“侦探故事”

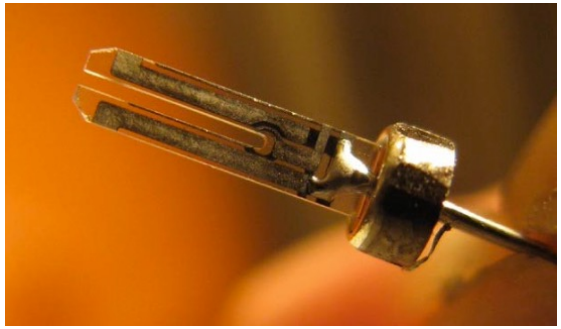
- 在 2012 年 6 月 30 日到 7 月 1 日的夜里（英国时间），全球许多在线服务和系统几乎在同一时间发生故障。
- 服务器死锁卡死，停止响应。
- 一些航空公司在数小时内无法处理任何机票预订或登机手续。
- 到底发生了什么？

分布式系统中的时间和时钟

- **分布式系统中经常需要“计时”，例如：**
  - 调度器、超时机制、故障检测器、重试定时器
  - 性能测量、统计分析、性能剖析（profiling）
  - 日志文件和数据库：记录事件发生的时间
  - 具有时间有效期的数据（例如缓存条目）
  - 在多个节点之间确定事件发生的先后顺序
- **我们区分两类“时钟”：**
  - **物理时钟（physical clocks）**：计算已经过去的秒数
  - **逻辑时钟（logical clocks）**：计算事件的数量，例如发送的消息数
- **注意：数字电路中的“时钟”（振荡器产生的时钟信号）**
  /= 分布式系统中的“时钟”（用于产生时间戳的时间来源）

石英钟

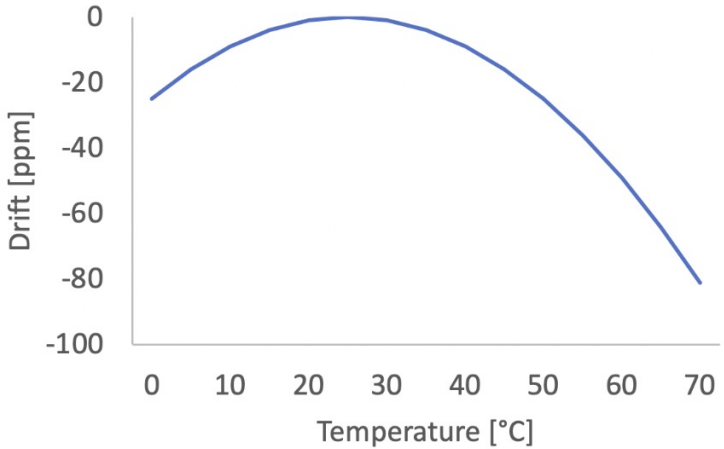
- 石英晶体通过激光微调，使其在特定频率下产生机械谐振
- 压电效应：机械力 ⇔ 电场
- 振荡电路在谐振频率下产生信号
- 通过计数振荡周期的数量来测量经过的时间



石英钟走时误差：漂移

- 一台时钟会稍微快一点，另一台会稍微慢一点
- 漂移用“百万分率”（parts per million，ppm）来度量
- 1 ppm = 每秒快/慢 1 微秒 = 每天约 86 毫秒 = 每年约 32 秒
- 大多数计算机时钟的校准精度在约 50 ppm 之内

温度会显著影响漂移



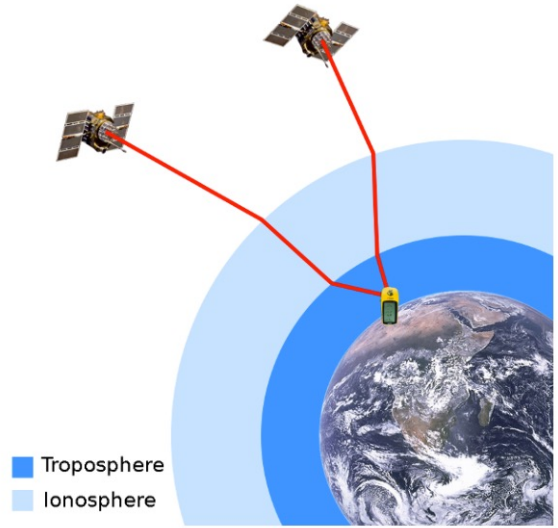
原子钟

- 铯-133（Caesium-133）在约 9 GHz 处具有一个共振频率（“超精细能级跃迁”）
- 可以把电子振荡器调谐到这个共振频率上
- 定义：1 秒 = 该信号的 9,192,631,770 个周期
- 精度约为 1×10^{-14}（约 300 万年才误差 1 秒）
- 价格约 £20,000（？）
  - （也可以购买更便宜的铷原子钟，约 £1,000）

https://www.microsemi.com/product-directory/cesium-frequency-references/4115-5071a-cesium-primary-frequency-standard

基于GPS的时间源

- 共有 31 颗卫星，每颗卫星都携带一台原子钟
- 卫星会广播当前的时间和自身位置
- 接收端根据卫星到接收器之间的光速传播延迟来计算位置
- 需要对大气层影响、相对论效应等进行修正
- 在数据中心中，需要在屋顶安装天线才能接收信号



https://commons.wikimedia.org/wiki/File:Gps-atmospheric-efects.png

协调世界时UTC

- 格林尼治平时（GMT，太阳时）：从格林尼治子午线观测，当太阳位于南方时即为正午。
- 国际原子时（TAI）：1 天被定义为铯-133 共振频率的 24 × 60 × 60 × 9,192,631,770 个周期。

问题：地球自转的速度并不是恒定不变的

折中方案：UTC 在国际原子时（TAI）的基础上，加入对地球自转的修正。

时区和夏令时都是在 UTC 基础上的偏移量。

闰秒

- 每年在 6 月 30 日和 12 月 31 日的 23:59:59（UTC），时钟会发生以下三种情况之一：
  - 时钟立刻跳到 00:00:00，跳过一秒（负闰秒）
  - 时钟在一秒之后照常变为 00:00:00
  - 时钟在一秒之后变为 23:59:60，再过一秒变为 00:00:00（正闰秒）
- 这一调整会在实施前的数个月提前公布。

http://leapsecond.com/notes/leap-watch.htm

计算机如何表示时间戳

- **最常见的两种表示方式**：
  - Unix 时间：自 1970 年 1 月 1 日 00:00:00 UTC（“纪元”）以来所经过的秒数，不计入闰秒
  - ISO 8601：用年份、月份、日期、小时、分钟、秒以及相对于 UTC 的时区偏移来表示

    例如：2021-11-09T09:50:17+00:00
- **在两种表示方式之间转换需要**：
  - 公历规则：一年有 365 天，闰年的例外情况是：

    year % 4 == 0 && (year % 100 != 0 || year % 400 == 0)
  - 对过去和未来所有闰秒的完整了解……？！

大多数软件如何处理闰秒

- 不管它

https://www.flickr.com/photos/ru_boff/37915499055/

大多数软件如何处理闰秒

- 不管它

然而，操作系统和分布式系统经常需要亚秒级精度的计时。

2012 年 6 月 30 日：Linux 内核中的一个缺陷在插入闰秒时导致了活锁，从而使许多互联网服务宕机。

务实做法：把这一闰秒均匀分摊到一天的时间里

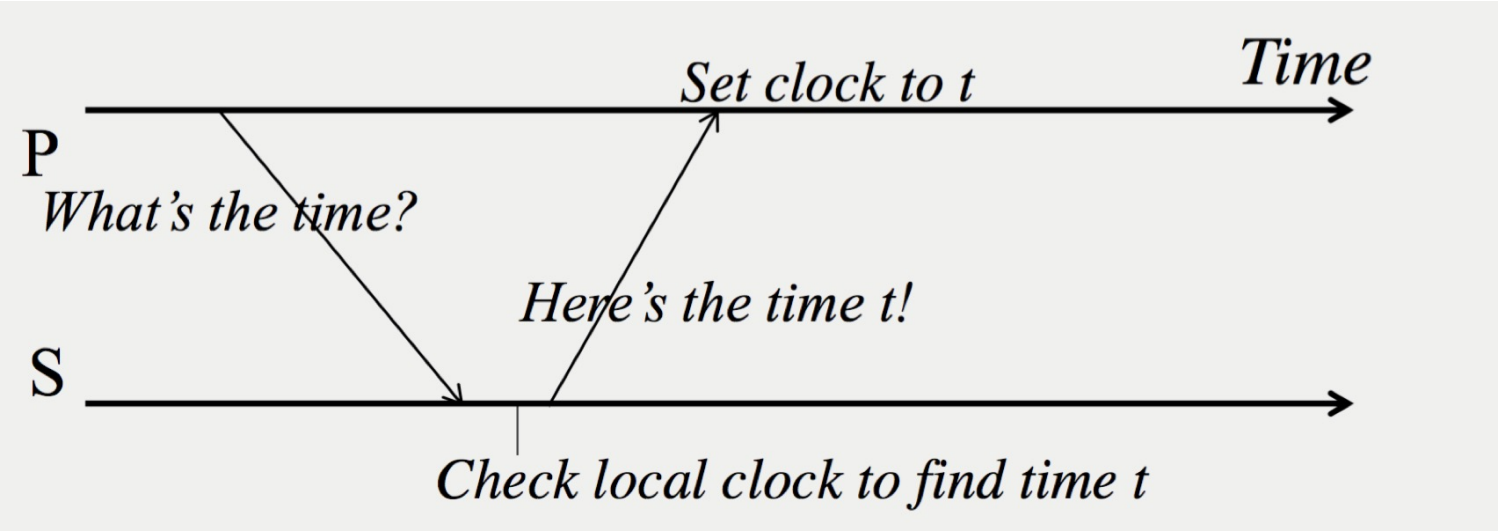
https://www.flickr.com/photos/ru_boff/37915499055/

时钟同步

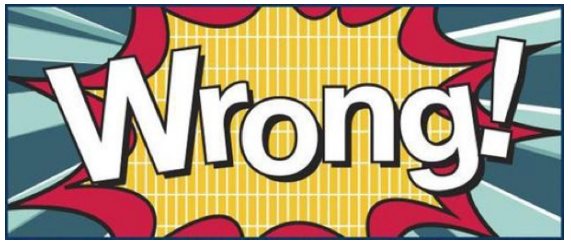
- 计算机使用石英钟来跟踪物理时间 / UTC（带电池，即使断电也会继续运行）
- 由于时钟漂移（clock drift），时钟误差会随时间逐渐增大
- 时钟偏差（clock skew）：在某一时刻，两台时钟之间的时间差
- 解决办法：
  - 定期向拥有更精确时间源（如原子钟或 GPS 接收机）的服务器获取当前时间，并进行校准
- 常用时间同步协议：
  - 网络时间协议 NTP (Network Time Protocol)
  - 精确时间协议 PTP (Precision Time Protocol)

Cristian算法

- **使用外同步：**
  - 所有处理节点P 与一外部时间服务器S同步。



这么简单？



- P在收到S的回复消息时，已经又过了一段时间。
- P的时钟更新为t是不正确的。
- 误差取决于消息的传输延迟。
- 异步系统中，消息的传输延迟是不受限的，因为误差可大可小。

网络时间协议 NTP

- 许多操作系统厂商都会运行自己的 NTP 服务器，并将操作系统默认配置为使用这些服务器进行时间同步。

网络时间协议 NTP

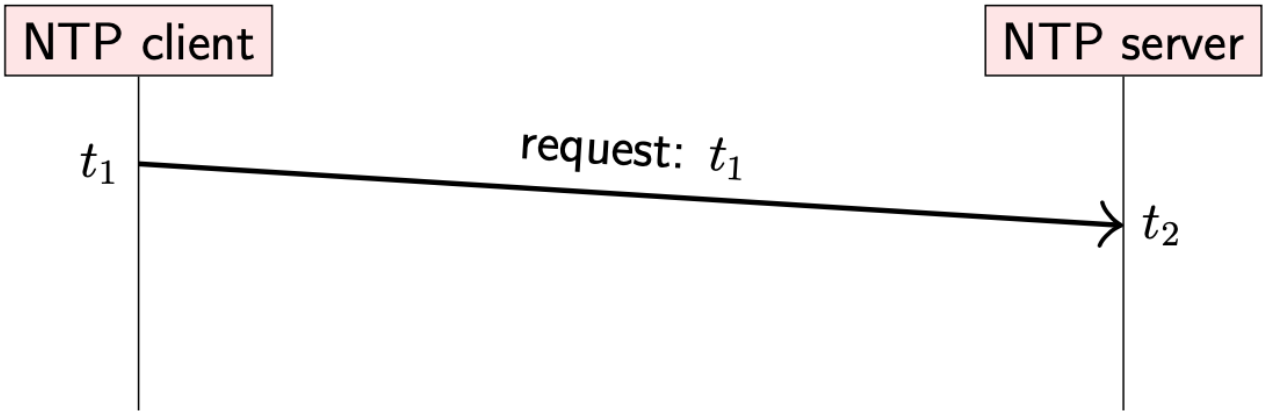
- **许多操作系统厂商都会运行自己的 NTP 服务器，并将操作系统默认配置为使用这些服务器进行时间同步。**
- **时钟服务器按照分层结构（strata）组织：**
  - Stratum 0：原子钟或 GPS 接收机等最精确的时间源
  - Stratum 1：直接与 stratum 0 设备同步的服务器
  - Stratum 2：与 stratum 1 服务器同步的服务器，依此类推

网络时间协议 NTP

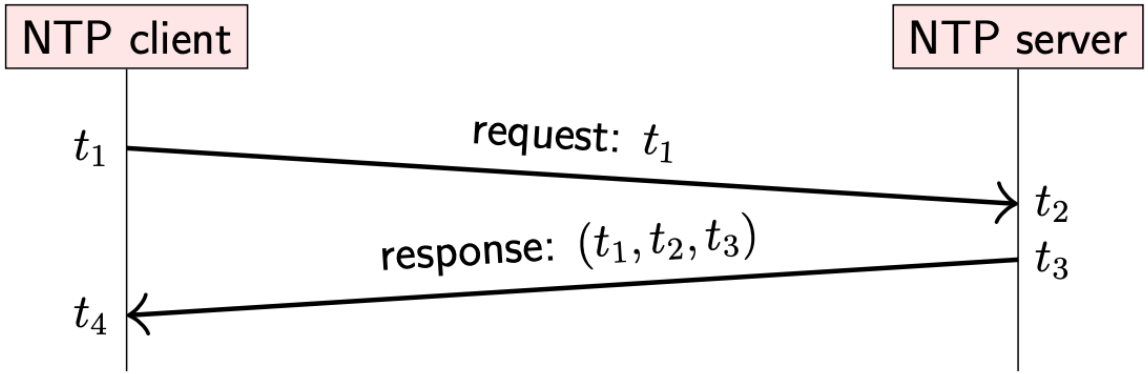
- **许多操作系统厂商都会运行自己的 NTP 服务器，并将操作系统默认配置为使用这些服务器进行时间同步。**
- **时钟服务器按照分层结构（strata）组织：**
  - Stratum 0：原子钟或 GPS 接收机等最精确的时间源
  - Stratum 1：直接与 stratum 0 设备同步的服务器
  - Stratum 2：与 stratum 1 服务器同步的服务器，依此类推
- **客户端通常会联系多个时间服务器，丢弃明显异常的结果（离群值），对剩余结果取平均。**
- **同时，它会对同一台服务器发出多次请求，利用统计方法减少由于网络延迟波动带来的随机误差。**
- **在网络状况良好的情况下，这种方法可以把时钟偏差（clock skew）控制在几毫秒以内，但在不良网络条件下，误差可能会大得多！**

在网络环境下估计时间

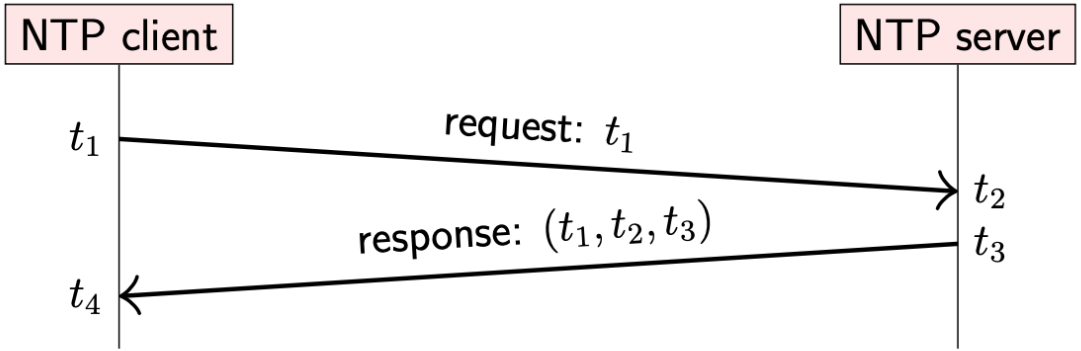
在网络环境下估计时间



在网络环境下估计时间

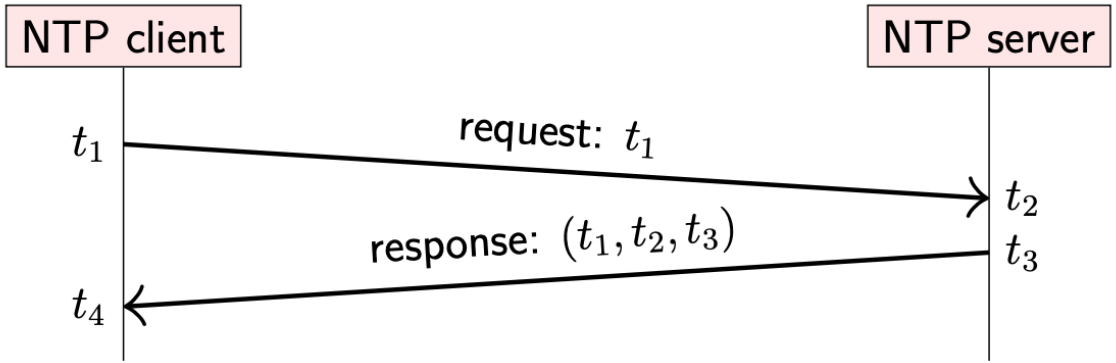


在网络环境下估计时间



Round-trip network delay: $\delta = (t_4 - t_1) - (t_3 - t_2)$

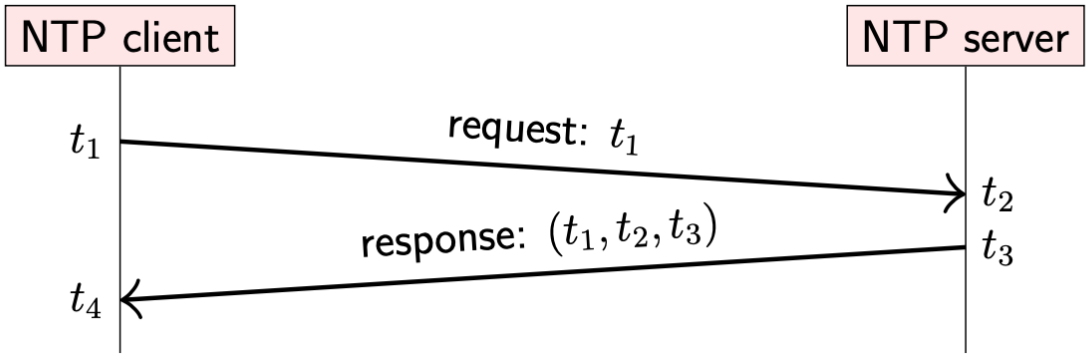
在网络环境下估计时间



Round-trip network delay: $\delta = (t_4 - t_1) - (t_3 - t_2)$

Estimated server time when client receives response: $t_3 + \frac{\delta}{2}$

在网络环境下估计时间



Round-trip network delay: $\delta = (t_4 - t_1) - (t_3 - t_2)$

Estimated server time when client receives response: $t_3 + \frac{\delta}{2}$

Estimated clock skew: $\theta = t_3 + \frac{\delta}{2} - t_4 = \frac{t_2 - t_1 + t_3 - t_4}{2}$

时间误差校准

- **一旦客户端估计出了时钟偏差 θ，就需要用这个校正值来调整自己的时钟。**
  - 如果 |θ| < 125 ms，就通过“微调”方式来校准时钟：
    把时钟频率稍微调快或调慢，最多调整到 500 ppm（百万分之五百），这样可以在大约 5 分钟内让各个时钟同步。

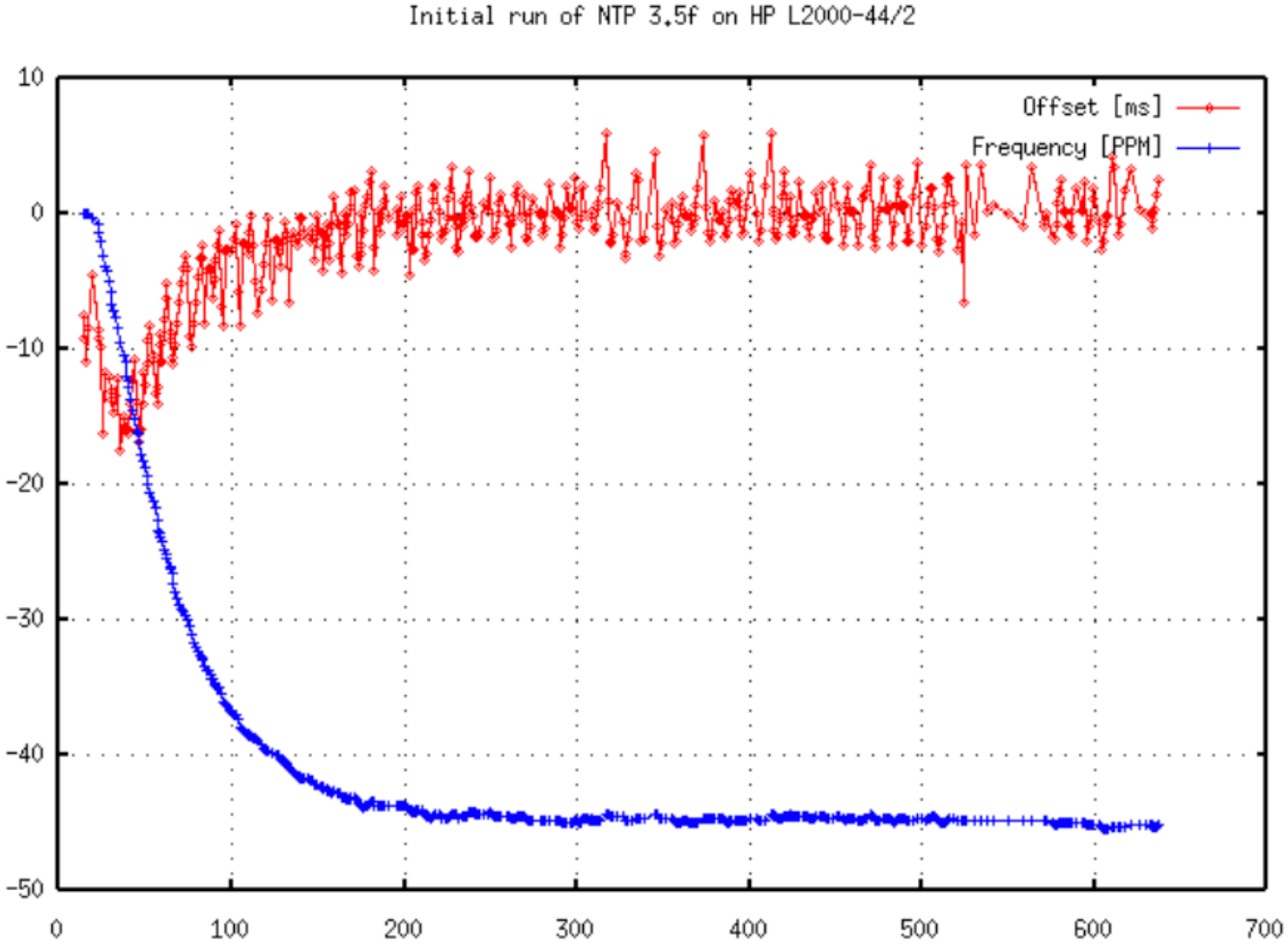
时间误差校准

- **一旦客户端估计出了时钟偏差 θ，就需要用这个校正值来调整自己的时钟。**
  - 如果 |θ| < 125 ms，就通过“微调”方式来校准时钟：
    把时钟频率稍微调快或调慢，最多调整到 500 ppm（百万分之五百），这样可以在大约 5 分钟内让各个时钟同步。
  - 如果 125 ms ≤ |θ| < 1,000 s，就采用“跳变”方式校正时钟：
    立刻把客户端的时钟重置为估计得到的服务器时间戳。

时间误差校准

- **一旦客户端估计出了时钟偏差 θ，就需要用这个校正值来调整自己的时钟。**
  - 如果 |θ| < 125 ms，就通过“微调”方式来校准时钟：
    把时钟频率稍微调快或调慢，最多调整到 500 ppm（百万分之五百），这样可以在大约 5 分钟内让各个时钟同步。
  - 如果 125 ms ≤ |θ| < 1,000 s，就采用“跳变”方式校正时钟：
    立刻把客户端的时钟重置为估计得到的服务器时间戳。
  - 如果 |θ| ≥ 1,000 s，就进入“恐慌”状态并且不做任何校正
    （把问题留给人工运维人员来处理）。

依赖时钟同步的系统必须持续监控时钟偏差！



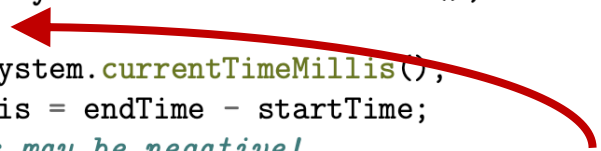
http://www.ntp.org/ntpfaq/NTP-s-algo.htm

单调时钟与日历时钟

```
// BAD:
long startTime = System.currentTimeMillis();
doSomething();
long endTime = System.currentTimeMillis();
long elapsedMillis = endTime - startTime;
// elapsedMillis may be negative!
```

单调时钟与日历时钟

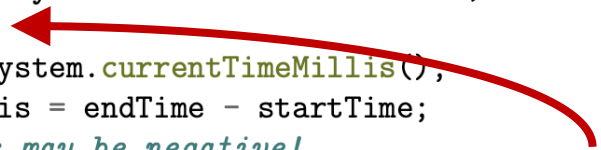
```
// BAD:
long startTime = System.currentTimeMillis();
doSomething();
long endTime = System.currentTimeMillis();
long elapsedMillis = endTime - startTime;
// elapsedMillis may be negative!
```



NTP 客户端会在此期间采用“跳变”的方式来调整时钟。

单调时钟与日历时钟

```
// BAD:
long startTime = System.currentTimeMillis();
doSomething();
long endTime = System.currentTimeMillis();
long elapsedMillis = endTime - startTime;
// elapsedMillis may be negative!
```



NTP 客户端会在此期间采用“跳变”的方式来调整时钟。

```
// GOOD:
long startTime = System.nanoTime();
doSomething();
long endTime = System.nanoTime();
long elapsedNanos = endTime - startTime;
// elapsedNanos is always >= 0
```

单调时钟与日历时钟

- **日历时钟（当前时间时钟）：**
  - 从某个固定日期开始计算经过的时间（例如自 1970 年 1 月 1 日纪元起算）

- **单调时钟：**
  - 自某个任意参考时刻起经过的时间（例如从机器启动时算起）

单调时钟与日历时钟

- **日历时钟（当前时间时钟）：**
  - 从某个固定日期开始计算经过的时间（例如自 1970 年 1 月 1 日纪元起算）
  - 可能会突然向前或向后跳变（由于 NTP 的跳变校时），并且会受到闰秒调整的影响。

- **单调时钟：**
  - 自某个任意参考时刻起经过的时间（例如从机器启动时算起）
  - 始终以近似恒定的速率向前推进。

单调时钟与日历时钟

- **日历时钟（当前时间时钟）：**
  - 从某个固定日期开始计算经过的时间（例如自 1970 年 1 月 1 日纪元起算）
  - 可能会突然向前或向后跳变（由于 NTP 的跳变校时），并且会受到闰秒调整的影响。
  - 时间戳可以在不同节点之间进行比较（前提是它们已完成同步）。

- **单调时钟：**
  - 自某个任意参考时刻起经过的时间（例如从机器启动时算起）
  - 始终以近似恒定的速率向前推进
  - 适合在单个节点上测量经过时间。

单调时钟与日历时钟

- **日历时钟（当前时间时钟）：**
  - 从某个固定日期开始计算经过的时间（例如自 1970 年 1 月 1 日纪元起算）
  - 可能会突然向前或向后跳变（由于 NTP 的跳变校时），并且会受到闰秒调整的影响。
  - 时间戳可以在不同节点之间进行比较（前提是它们已完成同步）。

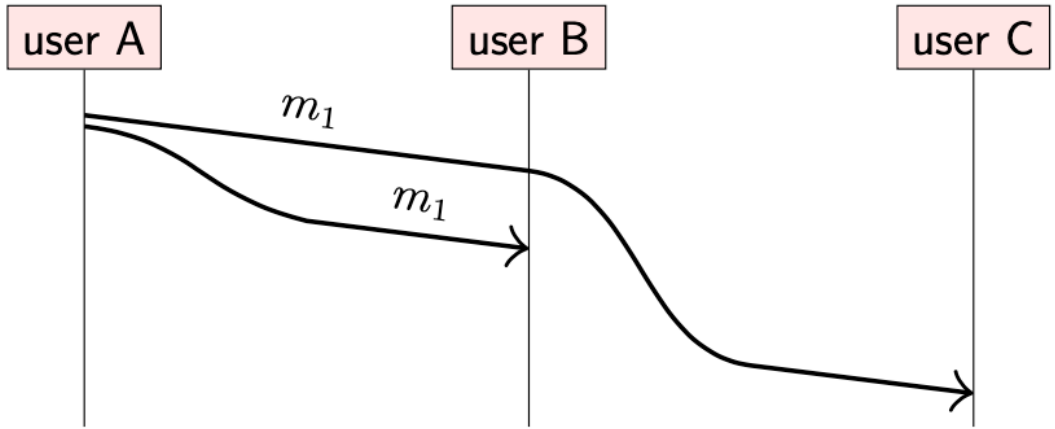
    Java: `System.currentTimeMillis()`

    Linux: `clock_gettime(CLOCK_REALTIME)`
- **单调时钟：**
  - 自某个任意参考时刻起经过的时间（例如从机器启动时算起）
  - 始终以近似恒定的速率向前推进
  - 适合在单个节点上测量经过时间。

    Java: `System.nanoTime()`

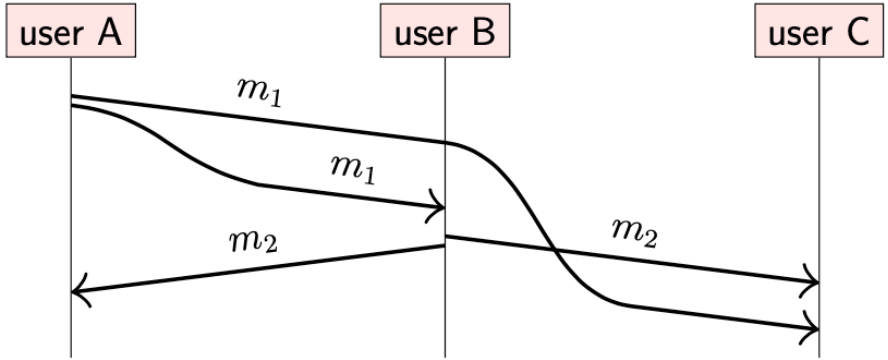
    Linux: `clock_gettime(CLOCK_MONOTONIC)`

消息的排序



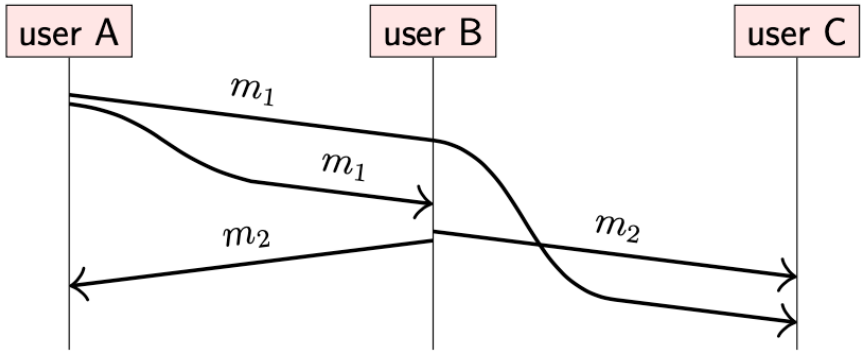
m_1 = "A says: The moon is made of cheese!"

消息的排序



$m_1 =$ "A says: The moon is made of cheese!"
$m_2 =$ "B says: Oh no it isn't!"

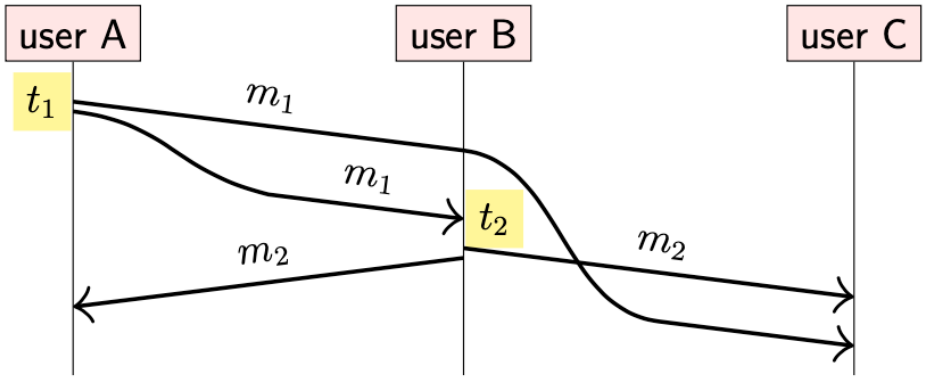
消息的排序



$m_1 =$ "A says: The moon is made of cheese!"
$m_2 =$ "B says: Oh no it isn't!"

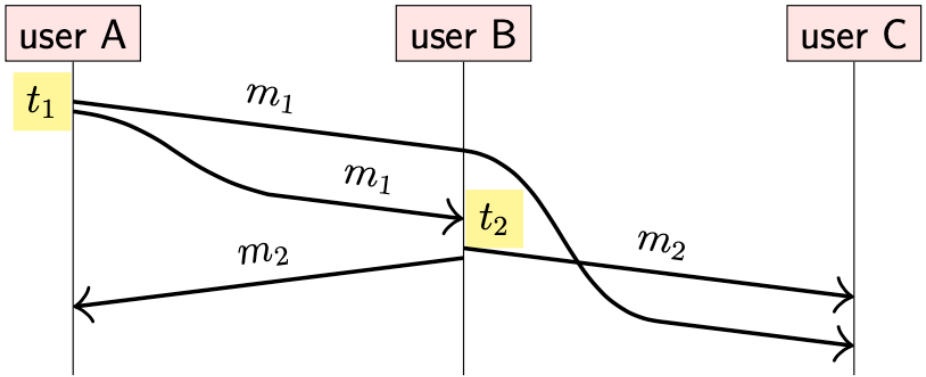
- C 先看到消息 m2，后看到消息 m1，
  尽管从逻辑上讲，m1 发生在 m2 之前。

使用时间戳进行消息排序



$m_1 = (t_1,$ "A says: The moon is made of cheese!")
$m_2 = (t_2,$ "B says: Oh no it isn't!")

使用时间戳进行消息排序



$m_1 = (t_1,$ "A says: The moon is made of cheese!")
$m_2 = (t_2,$ "B says: Oh no it isn't!")

- 问题：即使时钟已经同步，仍然有可能出现 t2 < t1 的情况。
  时间戳的顺序和我们期望的事件顺序并不一致！

“happens-before 关系”

- 一个事件是在某个节点上发生的一件事情（发送或接收一条消息，或者一次本地执行步骤）。
  我们称事件 a 先于事件 b 发生（记作 a → b），当且仅当：

“happens-before 关系”

- 事件：某个节点上发生的一件事情（发送或接收一条消息，或者一次本地执行步骤）。我们称事件 a 先于事件 b 发生（记作 a → b），当且仅当：
  - a 和 b 发生在同一个节点上，并且 a 在该节点的本地执行顺序中先于 b 发生；或者

“happens-before 关系”

- 事件：某个节点上发生的一件事情（发送或接收一条消息，或者一次本地执行步骤）。我们称事件 a 先于事件 b 发生（记作 a → b），当且仅当：
  - a 和 b 发生在同一个节点上，并且 a 在该节点的本地执行顺序中先于 b 发生；或者
  - 事件 a 是某条消息 m 的发送，事件 b 是同一条消息 m 的接收（假设发送的消息是唯一可区分的）；或者

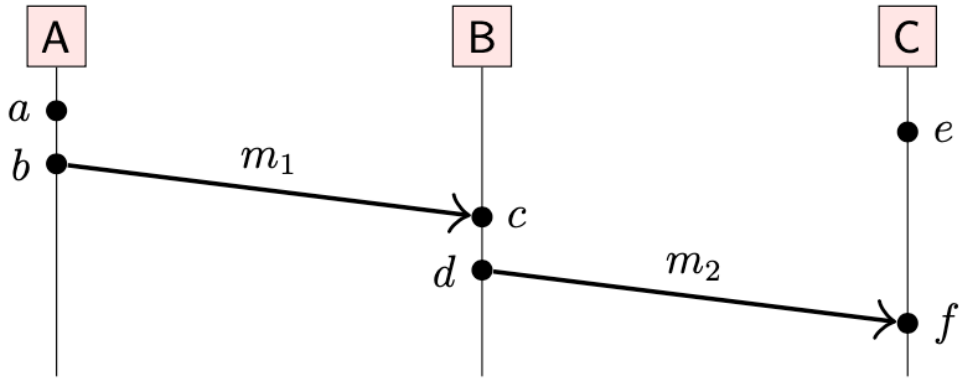
“happens-before 关系”

- 事件：某个节点上发生的一件事情（发送或接收一条消息，或者一次本地执行步骤）。我们称事件 a 先于事件 b 发生（记作 a → b），当且仅当：
  - a 和 b 发生在同一个节点上，并且 a 在该节点的本地执行顺序中先于 b 发生；或者
  - 事件 a 是某条消息 m 的发送，事件 b 是同一条消息 m 的接收（假设发送的消息是唯一可区分的）；或者
  - 存在某个事件 c，使得 a → c 且 c → b。

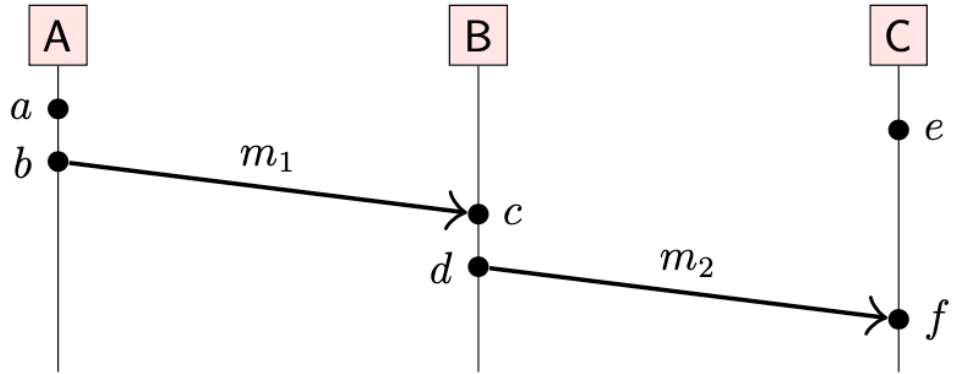
“happens-before 关系”

- 事件：某个节点上发生的一件事情（发送或接收一条消息，或者一次本地执行步骤）。
  我们称事件 a 先于事件 b 发生（记作 a → b），当且仅当：
  - a 和 b 发生在同一个节点上，并且 a 在该节点的本地执行顺序中先于 b 发生；或者
  - 事件 a 是某条消息 m 的发送，事件 b 是同一条消息 m 的接收（假设发送的消息是唯一可区分的）；或者
  - 存在某个事件 c，使得 a → c 且 c → b。
- happens-before 关系是一个偏序关系：有可能既不是 a→b，也不是 b→a。
  在这种情况下，a 和 b 是并发的（记作 a∥b）。

“happens-before 关系”举例

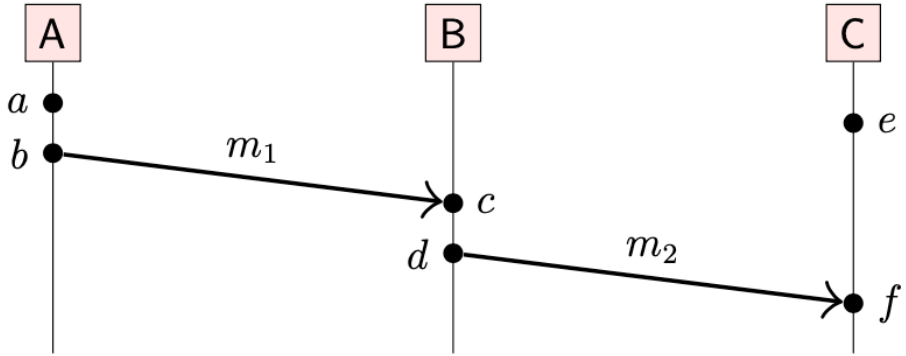


“happens-before 关系”举例



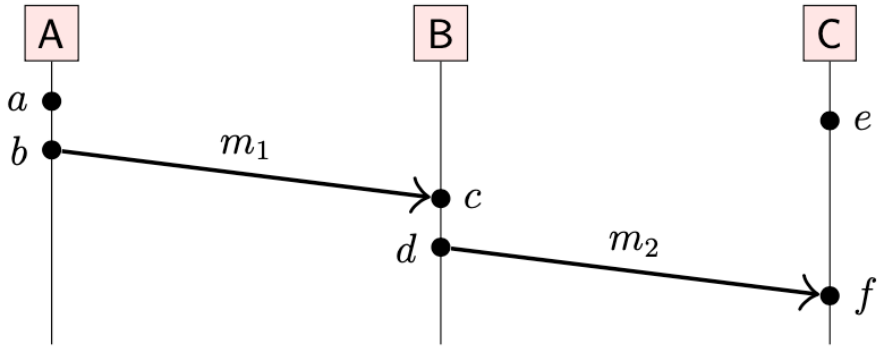
- $a \to b$, $c \to d$, and $e \to f$ due to node execution order

“happens-before 关系”举例



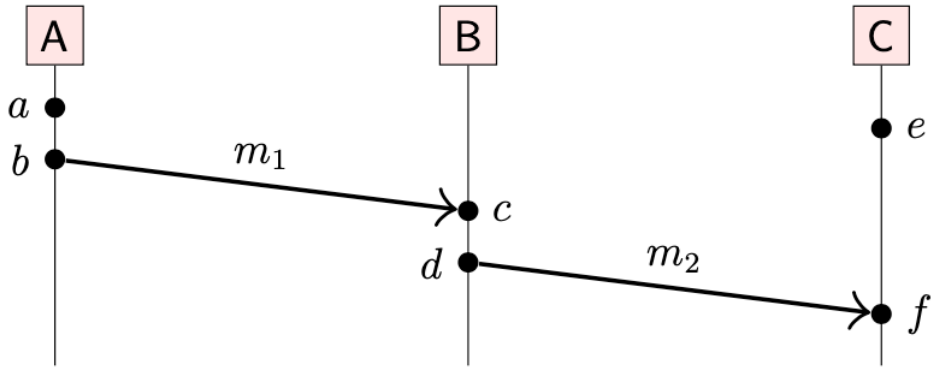
- $a \to b$, $c \to d$, and $e \to f$ due to node execution order
- $b \to c$ and $d \to f$ due to messages m_1 and m_2

“happens-before 关系”举例



- $a \to b$, $c \to d$, and $e \to f$ due to node execution order
- $b \to c$ and $d \to f$ due to messages m_1 and m_2
- $a \to c$, $a \to d$, $a \to f$, $b \to d$, $b \to f$, and $c \to f$ due to transitivity

“happens-before 关系”举例



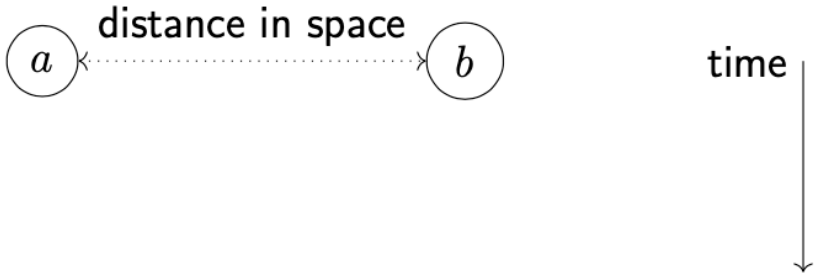
- $a \to b$, $c \to d$, and $e \to f$ due to node execution order
- $b \to c$ and $d \to f$ due to messages m_1 and m_2
- $a \to c$, $a \to d$, $a \to f$, $b \to d$, $b \to f$, and $c \to f$ due to transitivity
- $a \parallel e$, $b \parallel e$, $c \parallel e$, and $d \parallel e$

因果关系

- 借用自物理学（相对论）：
  - 当 a → b 时，a 可能是 b 的原因；
  - 当 a ∥ b 时，我们知道 a 不可能导致 b。
- happens-before 关系编码了“潜在的因果关系”。

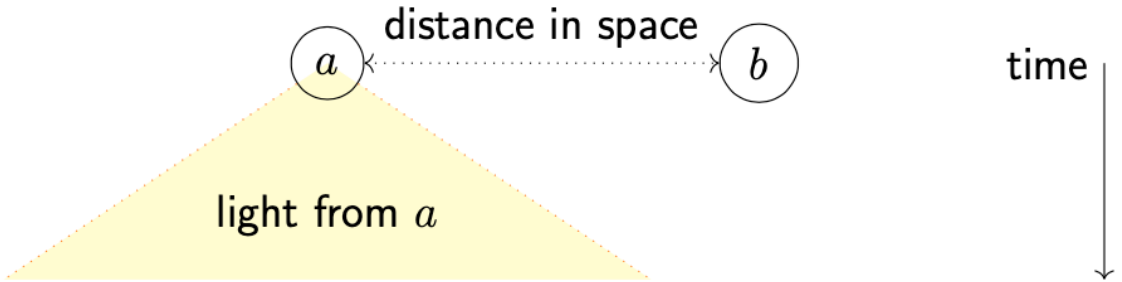
因果关系

- 借用自物理学（相对论）：
  - 当 a → b 时，a 可能是 b 的原因；
  - 当 a ∥ b 时，我们知道 a 不可能导致 b。
- happens-before 关系编码了“潜在的因果关系”。



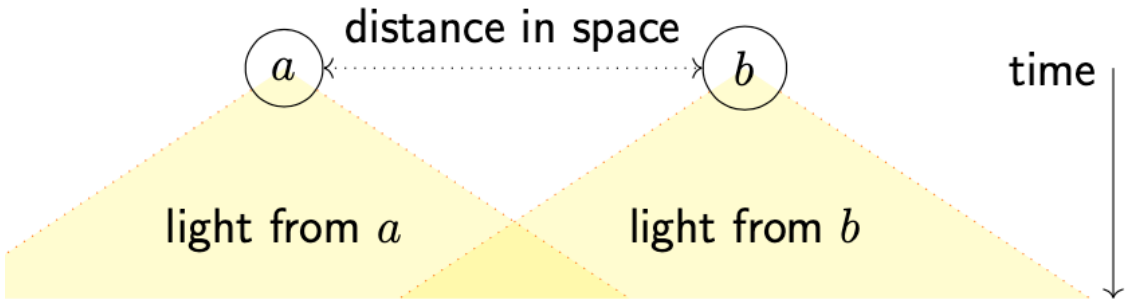
因果关系

- 借用自物理学（相对论）：
  - 当 a → b 时，a 可能是 b 的原因；
  - 当 a ∥ b 时，我们知道 a 不可能导致 b。
- happens-before 关系编码了“潜在的因果关系”。



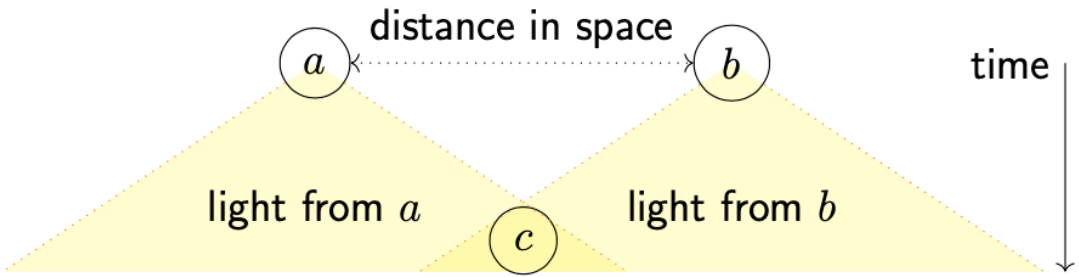
因果关系

- 借用自物理学（相对论）：
  - 当 a → b 时，a 可能是 b 的原因；
  - 当 a ∥ b 时，我们知道 a 不可能导致 b。
- happens-before 关系编码了“潜在的因果关系”。



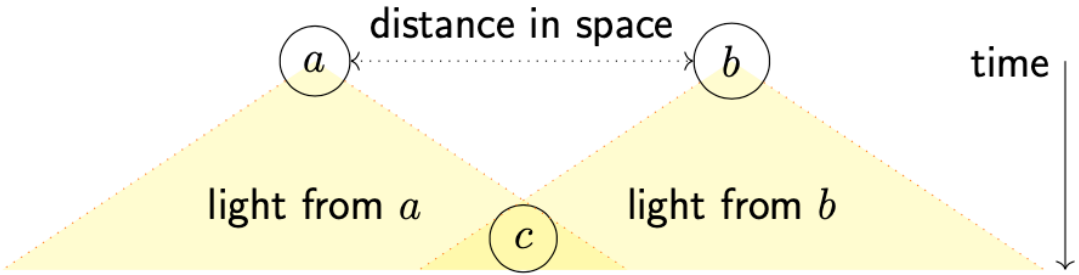
因果关系

- 借用自物理学（相对论）：
  - 当 a → b 时，a 可能是 b 的原因；
  - 当 a ∥ b 时，我们知道 a 不可能导致 b。
- happens-before 关系编码了“潜在的因果关系”。



因果关系

- 借用自物理学（相对论）：
  - 当 a → b 时，a 可能是 b 的原因；
  - 当 a ∥ b 时，我们知道 a 不可能导致 b。
- happens-before 关系编码了“潜在的因果关系”。



- 设 ≺ 是事件上的一个严格全序关系。
  如果 (a → b) ⇒ (a ≺ b)，那么称 ≺ 为一个因果有序（causal order）
  （或者说：≺ “与因果关系一致”（consistent with causality））。

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