# SINCRONIZAÇÃO DE RELÓGIOS

- Em uma arquitetura distribuída não há relógio físico comum
  - Cada nó possui um relógio físico separado
  - Diferenças mínimas nas frequências dos cristais que controlam os relógios levam a **perda de sincronia**
    - Cada relógio indica valor diferente
    - Esta diferença é denominada de **defasagem de relógio**.

- A defasagem de relógio pode causar problemas
- Exemplo:
    - Programa **make** determina se um arquivo será recompilado comparando a hora do arquivo fonte com a do arquivo objeto

**Questão**: Como manter os relógios sincronizados?

# MEDIDAS DE TEMPO

- No início, o tempo tem sido medido por métodos astronômicos
  - **Trânsito solar**: Passagem do sol pelo seu ponto aparente mais alto do céu
  - **Dia solar**: intervalo entre dois trânsitos consecutivos
  - **Segundo solar**: Dia solar divido por 86.400 (24x3600)

- O dia solar muda durante o ano
  - **Segundo solar médio**: Calcula-se a média de diversos dias solares e divide-se o resultado por 86.400

- Para uniformizar a medida de tempo, foi proposto o uso de relógio de césio

- **Relógio de césio**:
  - Conta transições do átomo de césio 133
  - Para 1 segundo, há 9.192.631.770 transições

- Existem diversos relógios de césio no mundo

- Cada um deles informa periodicamente  o BIH (Bureau International de l'Heure)
  - O BIH calcula a média destes valores e produz a **hora atômica internacional** (TAI)

- TAI é o número médio de ciclos dos relógios de césio 133 desde a meia-noite de 01/01/1958 dividido por 9.192.631.770

- **Problema com a TAI**
  - O dia solar médio está ficando mais longo

  - No decorrer dos anos, o meia dia (12hs) ficaria cada vez mais cedo, chegando a madrugada

- **Solução**:
  - Acrescentar segundos extras sempre a discrepância entre o dia solar médio e a TAI chegar a 800ms
  - Isto resulta em um sistema de medição de tempo de segundos constantes, em fase com o movimento aparente do sol
  - Este sistema é denominado de **hora coordenada universal** (UTC)

- Para fornecer UTC:
  - Existem rádios de ondas curtas, denominadas WWV, que transmite um pulso curto no início de cada segundo UTC
    - A precisão interna é de 1ms e externa de 10ms, devido às variações atmosféricas
  - Existem satélites oferecendo o serviço UTC
  - Para usar rádio ou satélite é preciso receptores específicos e conhecer a exata posição de transmissor e receptor, para compensar as distâncias

# RELÓGIOS FÍSICOS DE COMPUTADORES

- Os relógios de computadores baseiam-se em osciladores controlados por cristais de quartzo
  - O sinal deste oscilador é dividido, para gerar pulso a intervalos determinados
  - Este intervalo pode ser ajustado (em geral 1ms)
  - Este pulso gera uma interrupção
  - Esta interrupção ajusta o relógio interno do computador

- **Problema**: O oscilador apresenta variações nos ciclos da ordem de $10^{-5}$
  - Assim, se ele gera pulsos a cada 1ms
  - Temos 3600000 pulsos por hora
  - Variação de $10^{-5}$ : 36 pulsos
  - Em uma hora, temos de 3.599.964 à 3.600.036 pulsos

- Se quisermos que dois computadores **não tenham defasagem** maior do que $\delta$ segundos:
  - Relógios devem ser **sincronizados** a cada $\delta/\nu$, sendo $\nu$ a variação máxima do relógio por segundo
  -
- Esta sincronização deve levar em conta o tempo de propagação pela rede
  - Deve ser feito por **algoritmos de sincronização de relógios**.

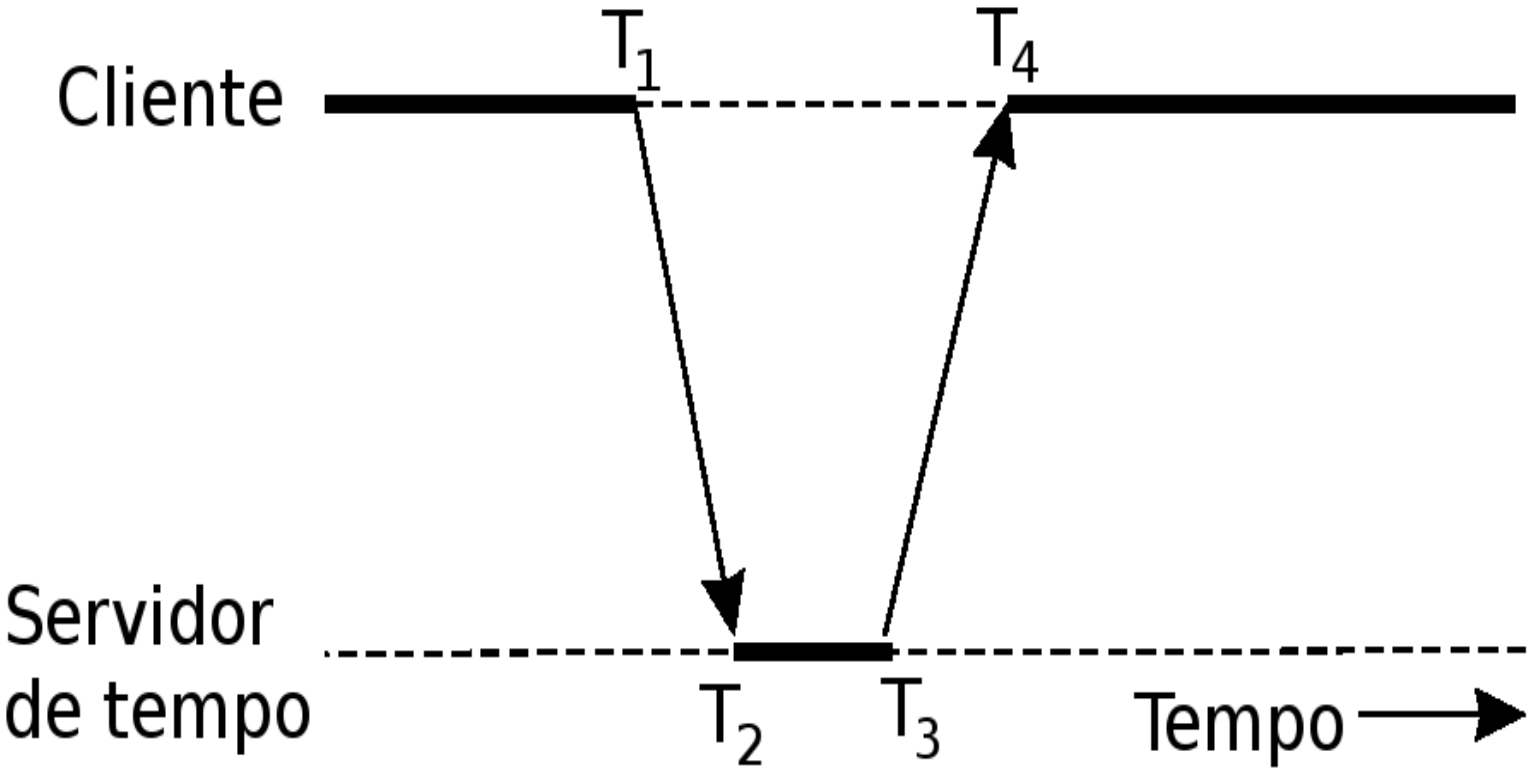
# PROTOCOLO DE TEMPO DE REDE – NTP

- Proposta:
  - Usar um **servidor de tempo**
  - Este servidor fornece a hora corrente exata, sincronizada por WWV, por exemplo
  - Clientes que necessitam atualizar o relógio, consultam o servidor

- Problema:
  - Precisa levar em conta o **tempo de atraso** das mensagens

- Considere que um cliente envia uma requisição ao servidor de tempo
  - Esta requisição contém uma marca, dado pelo valor do tempo no relógio do cliente $T_1$
  - Ao receber a requisição, o servidor marca a mensagem com o horário $T_2$ , **no seu relógio**, em que recebeu a mesma
  - Ao enviar a resposta, o servidor marca com o horário de envio $T_3$
  - Ao receber a resposta, o cliente marca o horário do recebimento $T_4$

- A figura a seguir ilustra estes momentos:



- Assumindo que o tempo de envio da requisição é aproximadamente o mesmo do envio da resposta:
  - Consideramos o tempo de propagação $T_p$ igual a média destes tempos, ou seja:

$$T_p = \frac{(T_2 - T_1) + (T_4 - T_3)}{2}$$

  - Este tempo de propagação **independe** do sincronismo dos relógios, pois a soma do tempo de envio $T_e$ e de resposta $T_r$ é :

$$T_e + T_r = (T_4 - T_1) - (T_3 - T_2)$$

- Rescrevendo a equação:
  - $T_e + T_r = (T_4 - T_3) + (T_2 - T_1)$
- Onde obtemos a mesma equação de $T_p$

- Assim, o instante que o cliente recebe a resposta corresponde, no relógio do servidor, ao tempo dado por:

$$T_3 + T_p$$

- Logo, a defasagem entre o relógio do cliente e do servidor é:

$$\delta = T_4 - T_3 - T_p$$

- ou ainda:

$$\delta = \frac{(T_4 - T_3) - (T_2 - T_1)}{2}$$

- Se $\delta < 0$ o relógio do cliente está atrasado
  - Pode ser corrigido acrescentando $\delta$

- Se $\delta > 0$ o relógio do cliente está adiantado
  - Não deve ser corrigido atrasando o relógio
    - Causa problemas com eventos sequenciais

  - Deve ser somado menos tempo por interrupção, até que o relógio esteja correto

- Para uma melhor correção:
  - Oito valores de $\delta$ são gerados
  - É considerado o menor deles.

- NTP é uma aplicação simétrica
  - Teoricamente, o servidor pode ajustar o relógio com o do cliente

  - Na prática, o serviço de NTP é dividido em **estratos**, numerados de 0 à 16:

    - **Estrato 0**:
      - Relógio de referência. Não é um servidor.

- **Estrato 1**:
  - Formado por servidores com receptor WWV ou relógio atômico

- **Estrato 16**:
  - Formado por servidores inoperantes

- Uma máquina só ajusta o seu relógio se o seu nível de **estrato for maior** do que o da outra máquina

- Em geral, servidores com número de estrato menor são mais precisos:
    - Na prática, as diferenças não são expressivas (exceto para estrato 16)
    - É preciso levar em conta a carga que os servidores podem estar submetidos

# RELÓGIOS LÓGICOS

- Lamport (1978) mostrou que embora seja possível a sincronização de relógios, ela não precisa ser absoluta:
  - Se dois processos não interagem
    - Relógios não precisam ser sincronizados
    - A falta de sincronismo não causará problemas e, portanto, não é observável
  - Também não importa que dois processos concordem com a hora exata
    - Importa apenas a **ordem** que os eventos ocorrem

**Causalidade**:

- Quando um evento *b* pode ocorrer apenas após a ocorrência do evento *a*, dizemos que dizemos que existe uma relação de **causalidade** entre eles
  - Exemplo:
    - O recebimento de uma mensagem só pode ocorrer após o envio da mesma

- Quando não for possível obter uma relação de causalidade entre os eventos, eles são ditos **concorrentes**

- A relação de causalidade pode ser indicada pelo operador "acontece antes", na forma: $a \rightarrow b$ , ou seja, *a* acontece antes de *b*.

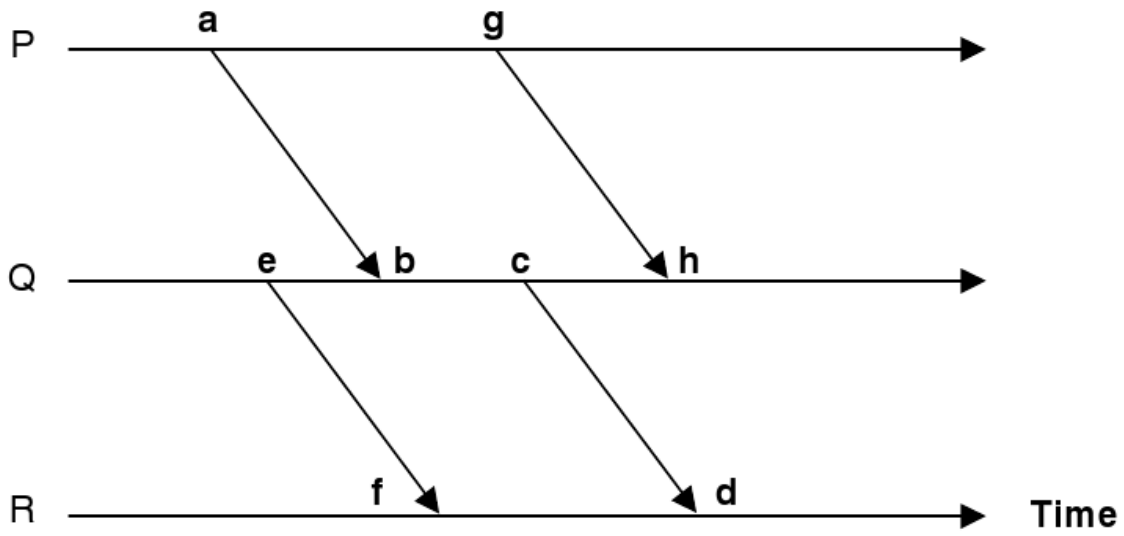
- Em um sistema distribuído serão consideradas 3 regras:
- **Regra 1**:
  - Cada processo tem um relógio que é um valor monótono crescente
  - Se *a* e *b* são dois eventos em um mesmo processo P e o evento *a* ocorreu antes do evento *b*, então $a \rightarrow b$ .

- **Regra 2**:
  - Se o evento *a* é o envio de uma mensagem pelo processo P e *b* é o recebimento da mensagem pelo processo Q, então $a \rightarrow b$

- **Regra 3**: (**Transitiva**) Se $a \rightarrow b$ e $b \rightarrow c$ então $a \rightarrow c$

**Exemplo:** Considere um sistema com 3 processos P, Q e R, cada um com o seu relógio, desconhecido dos demais.

- A figura abaixo ilustra a ocorrência dos eventos *a, b, c, d, e, f, g* e *h* nas retas de tempo de cada processo




- Podemos concluir que $a \rightarrow d$ e $e \rightarrow b$, por exemplo, mas não podemos dizer nada sobre *a* e *e* e *b* e *f*

## **Relógio Lógico**:

- É um contador de eventos que respeita a ordem estabelecida pela relação de causalidade.

- Cada processo tem um contador **LC** inicializado com zero e que segue três regras básicas:

  **LC1**: Cada vez que um evento local ou interno ocorre, incremente o valor de **LC**

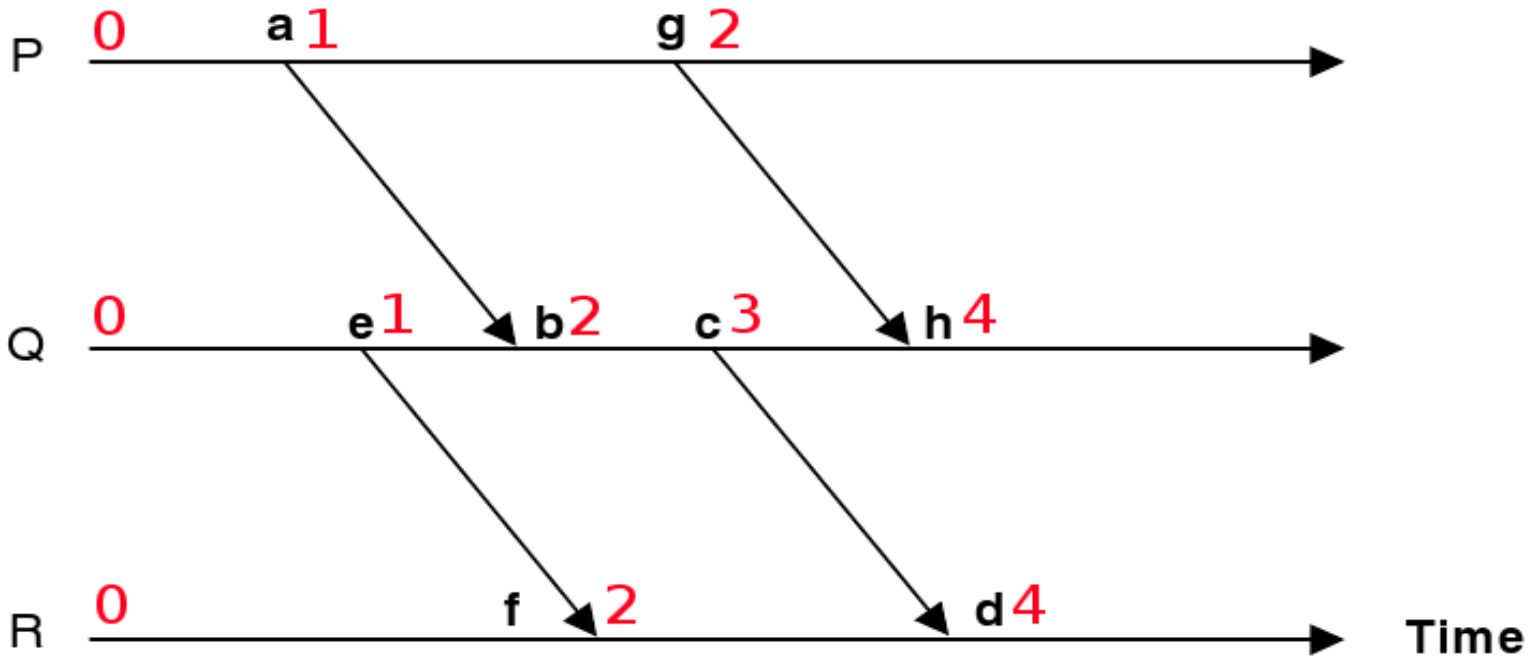
  **LC2**: Quando enviar uma mensagem, incorpore a mensagem o valor de **LC**

**LC3**: Quando receber uma mensagem, atribua a LC 1 mais o maior valor entre o LC local e o que veio na mensagem: LC = 1 + máx { LC, LC_Mens}

- Se $a \rightarrow b$ então $LC(a) < LC(b)$

- Não vale a volta: Se $LC(a) < LC(b)$, não podemos garantir que $a \rightarrow b$.

Exemplo: Para os processos e eventos mostrados anteriormente, temos:




Observe que LC($g$) < LC($c$) mas $c$ e $g$ **são concorrentes**.

# RELÓGIOS VETORIAIS

- Relógios lógicos em um sistema distribuído preservam a ordenação
- Entretanto, não permitem determinar a casualidade dos eventos a partir do valor do relógio lógico
- **Relógios vetoriais** se propõem a detectar a casualidade

- Em um sistema com *N* processos:
  - Definimos um vetor de relógio VC com *N* posições, para cada processo, como sendo um vetor de inteiros, inicialmente todo nulo

- A implementação do relógio é baseada em 3 regras:

  **VC1**: Cada evento local do processo *i* incrementa a posição *i* do seu próprio vetor de relógio VC em uma unidade

  **VC2**: O processo remetente faz uma cópia do seu vetor de relógio na mensagem que enviar.

  **VC3**: Quando um processo *j* recebe uma mensagem com uma cópia *T* de um vetor de relógio de outro processo, deve efetuar as seguintes operações:
  - Incrementa a posição *j* do seu vetor de relógio, ou seja: VC[j] := VC[j] + 1

- Atualiza o seu vetor de relógio, usando:

  VC[k] := máx { T[k], VC[k] }

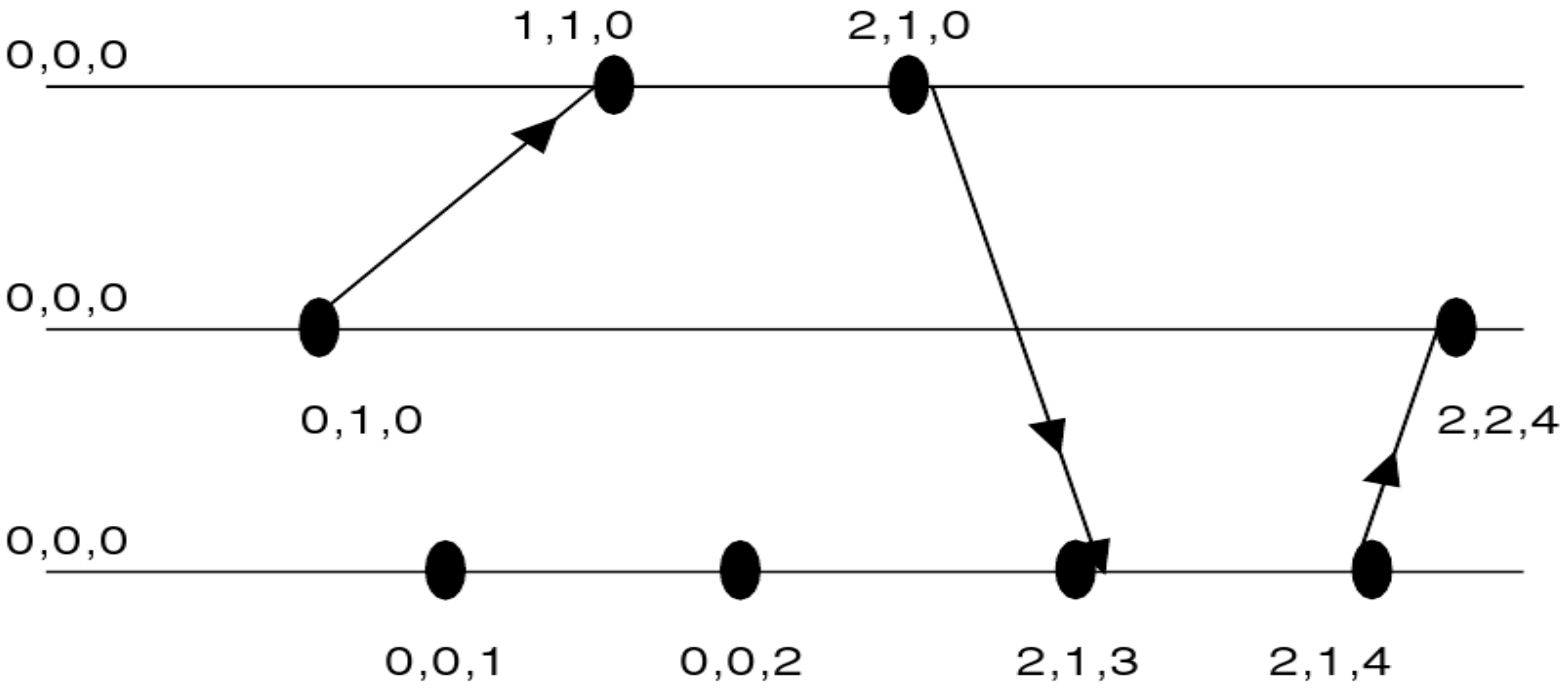
- Definindo VC($a$) ao vetor de relógio do processo na ocorrência do evento $a$, temos que:

$$a \rightarrow b \Leftrightarrow VC(a) < VC(b)$$

sendo que $VC(a) < VC(b)$ vale se, e somente se, as seguintes condições são verdadeiras:

1. $VC(a)[k] \leqslant VC(b)[k]$ para todo $k = 0, 1, 2, \ldots, N\text{-}1$
2. $VC(a)[i] < VC(b)[i]$ para algum $i$

# Exemplo: Relógio vetorial com 3 processos



Obs.: Relógios vetoriais apresentam dois problemas:

- Dificuldades na escalabilidade
  - Aumentar o número de processos requer reorganizar todo o sistema, pois o tamanho do vetor aumenta

- Custo do tráfego do vetor pela rede