並列分散コンピューティング
(8)時計の同期

大瀧保広

今日の内容

- 分散アルゴリズム
- 時間と時刻と時計
  - 時計の正しさとは？
- 時計の同期（Clock Syncronization）
  - Christian アルゴリズム
  - Berkeley アルゴリズム
  - NTP（Network Time Protocol）

分散アルゴリズム

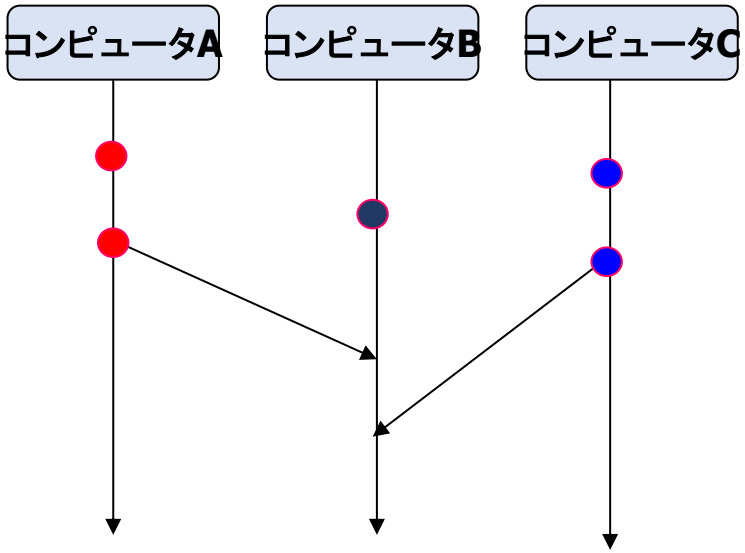
分散システムと分散アルゴリズム

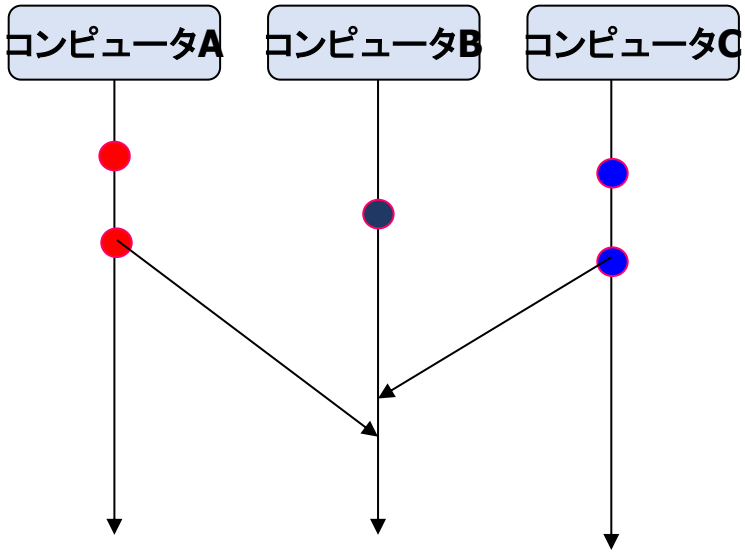
- 分散システムでは、複数のコンピュータ上で動くプロセス同士が協調して処理を行う。
  - 同期したメモリアクセスによる制御ができないため、協調して処理を行うためには**ネットワークを介した通信が必要**となる。

- 分散システムで動作する処理手順（**分散アルゴリズム**）では、通信に関する事象が無視できない。
  - 通信の**遅延**（到達順序に影響）
  - 通信の**不達**

通信の影響により非決定的になる

■各コンピュータで動作しているプログラムが
決定的（deterministic）であったとしても、
分散システム全体としては非決定的となる。

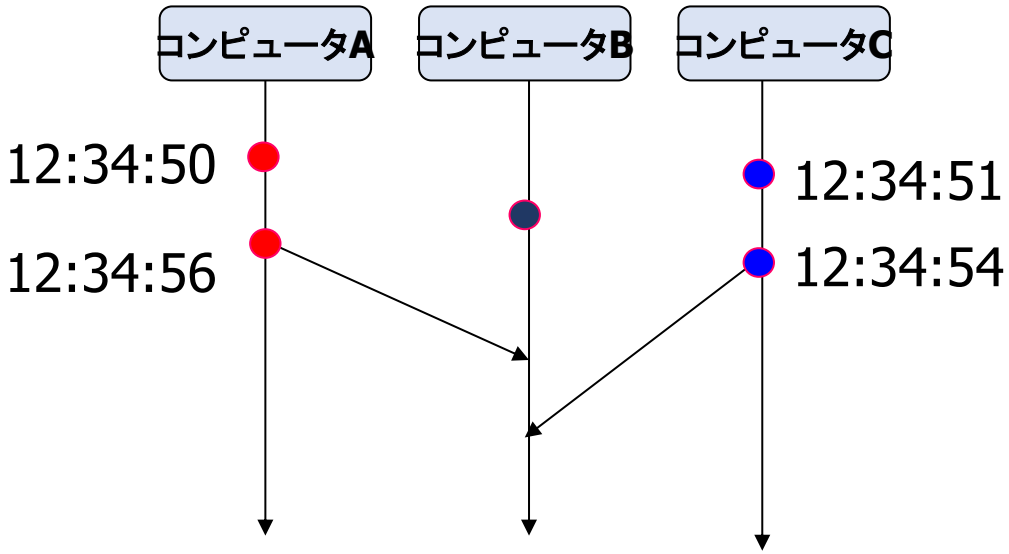




コンピュータB での処理が通信遅延の状況で変わる

決定的に動作するには？

- コンピュータBにおいて、コンピュータAとコンピュータCのどちらが先に通信を開始したか、が判断できれば、到着順序が多少前後しても非決定性を減少させることが可能かもしれない。
- これを判断するには、3台のコンピュータの時計が揃っていなければならない。



時間と時刻と時計

時間、時刻、時計

- 時間（物理量）：
  我々が生活する時空を構成する次元のうち、「空間」ではないもの。
  - 「時間」は「過去」から「未来」に向かう方向に進む
  - 過去と未来を区分する点を「現在」と呼ぶ。
- 時刻
  - 時間の経過を定量的に示すために、人間が時間軸上に定義した目盛の値
  - どのように目盛をつけるかは文化によって異なる
- 時計
  - 「時刻」を示すことを目的として構成される装置
  - 通常は「現在」の時刻を指す。
    時間の流れに合わせて示す値が変化する「仕掛け」

脱線：時刻の基準（「秒」の基準）

- かつて時計は天文学に基づいて測定されていた。
  - 太陽の「移動」（すなわち地球の自転）にもとづく時刻
    - 問題点：地球の自転がだんだん遅くなっている
      - 1日はだんだん長くなっている。
      - 3億年前、1年は400日あったらしい
- 周期的な物理現象で「刻む」ように変化
  - 振り子時計
  - 水晶発振器（クォーツ）
- 現在の時間の測り方の基準は原子時計や光格子時計
  - セシウム133原子

セシウム133 原子の基底状態の2つの超微細準位間の遷移に対応する放射の9,192,631,770周期の継続時間を1秒とする

時計の「正確さ」はどのように考えるか

以下の考え方はどうだろう？

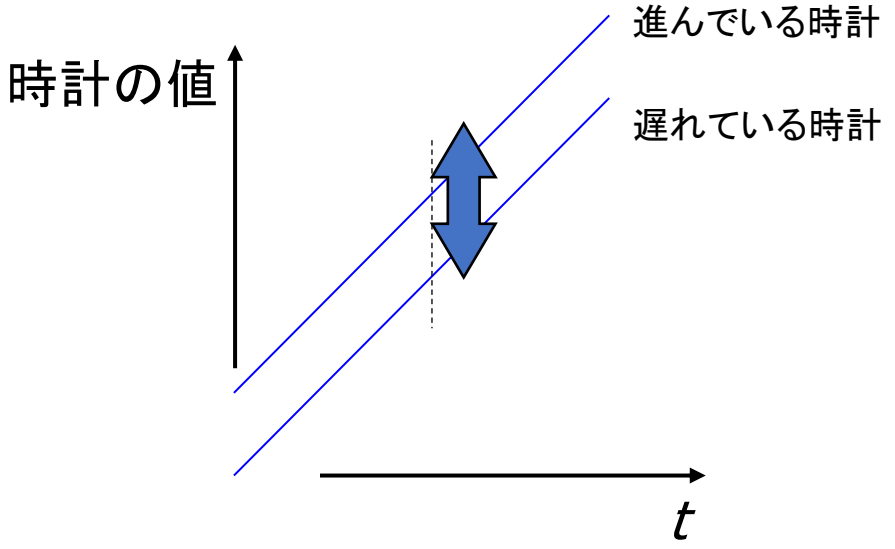
- 考え方1：正しい時刻（誤差0）を示す回数が多い方が正確
- 考え方2：正しい時刻との誤差が小さい方が正確

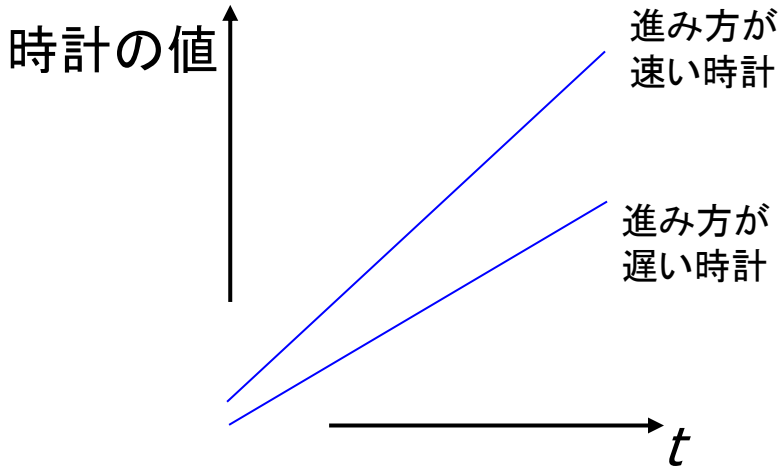
時計が「正しくない」とは

時計が同じ動きをしていない要因は2つに分解できる。

- Skew（スキュー）：
  2つの時計が示す時刻の差（ズレ）
- Drift（ドリフト）：
  2つの時計の進む速度の違い
  - 原因：水晶発振器の個体差
    +温度、湿度、電圧の差など

- ドリフトによる差が累積すると
  大きなスキューの原因となる。



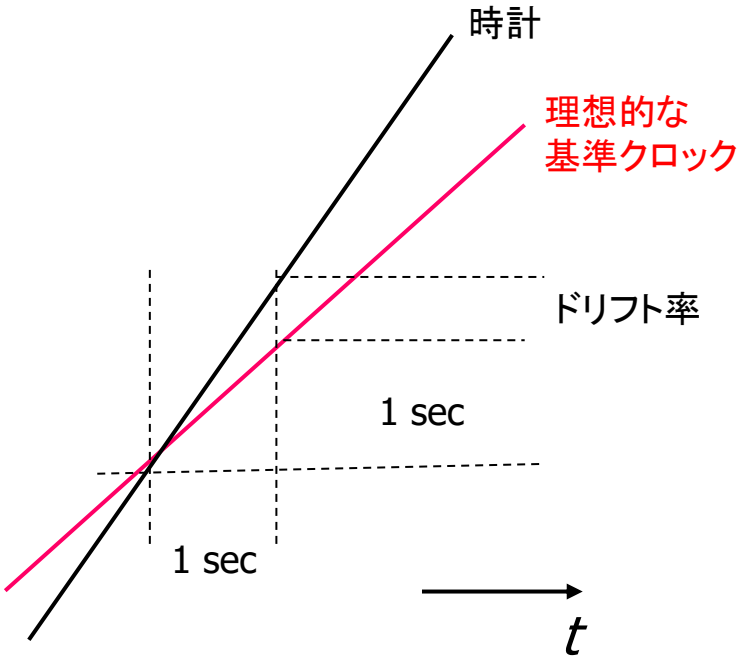


ドリフト率

‘率’は無次元量
(単位がない)

- ドリフト率：理想的な基準クロック と 時計の精度の差 を表す
  - 一般に 10^{-6} sec/sec 程度
  - 高精度でも10^{-7} ～10^{-8} 程度

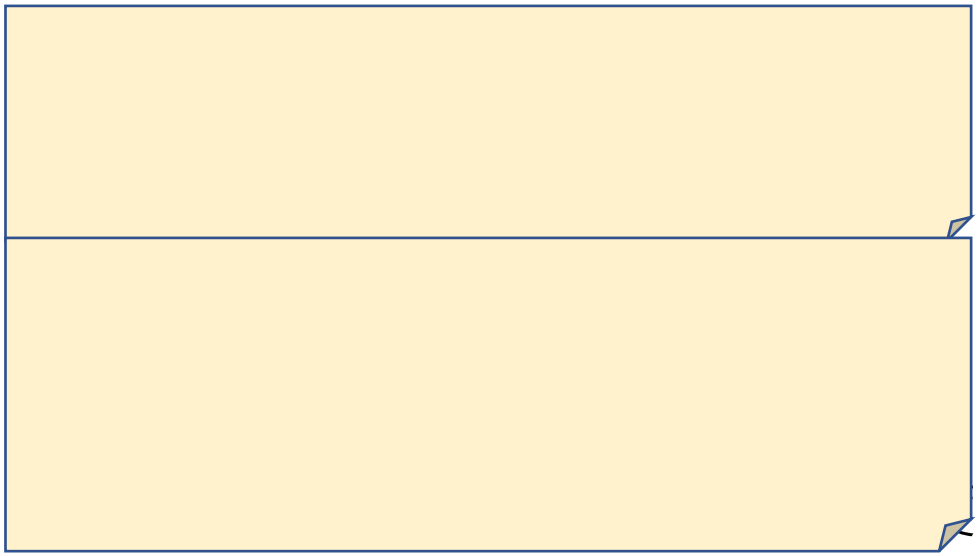
- ドリフト率 10^{-6} とすると
  約17分で1msec のズレ
  約11.6日で 1sec のズレ



予習：このクオーツ式腕時計のドリフト率は？

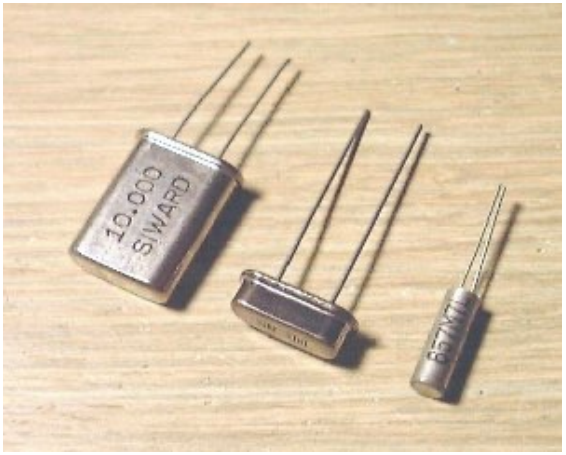
仕　様

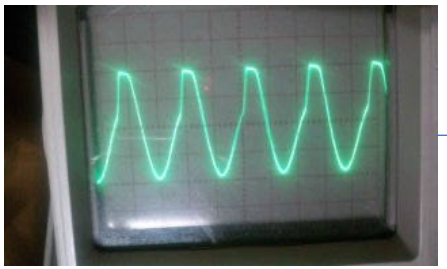
機種：E820
型式：アナログソーラーパワーウオッチ
時間精度：平均月差± 15 秒
常温（＋ 5℃～ 35℃）携帯時
作動温度範囲：－ 10℃～＋ 60℃
表示機能
・時刻：24 時間、時、分、秒
・カレンダー：パーペチュアルカレンダー（2100 年 2 月 28 日まで）
日表示
秒針による月表示
機能針による年表示（うるう年からの経過年…修正時のみ）
・クロノグラフ：60 分計、1/20 秒単位、自動停止機能付き
・ローカルタイム：時差修正（1 時間単位）
・アラーム：24 時間制

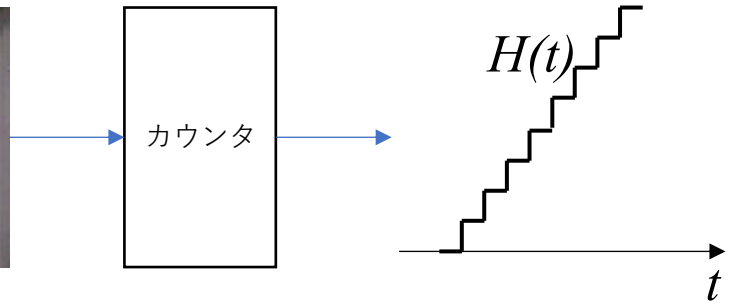


コンピュータの時計の仕組み（ハード）

- コンピュータの中の物理的な時計は、
  水晶発振器（クォーツ）とカウンタを用いて
  ハードウェアとして実現されている。
  - プログラムからはカウンターレジスタ として参照可能
    （以後この値を $H(t)$とする）
  - ハードウェア割り込みを発生させるタイミングのもと



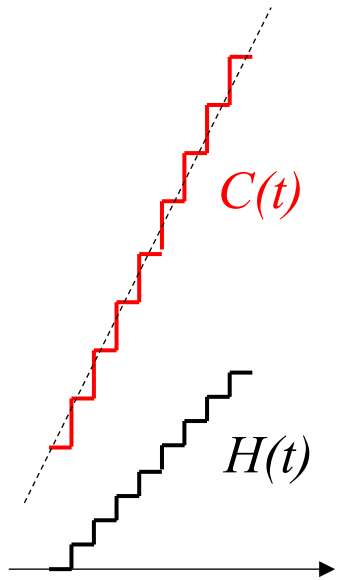




コンピュータの時計の仕組み（ソフト）

■ハードゥエア時計 H(t) を、
物理時間 t に近似するようにスケーリングすることで、
ソフトウェア時計$C(t)$ が生成される。
$C(t) = \alpha H(t) + \beta$

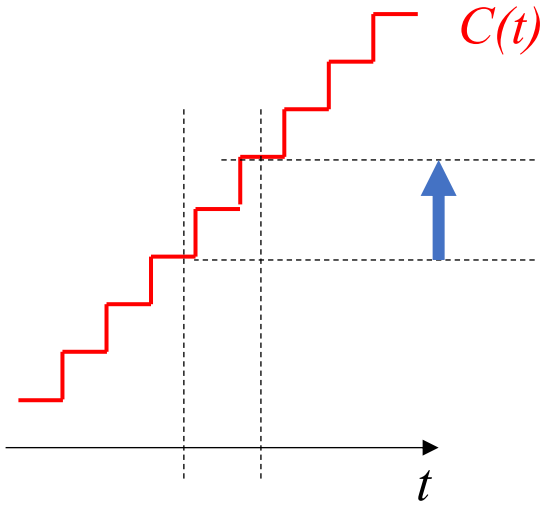
■$C(t)$ は、ある特定のイベントからの相対時間を
計測するように調整される。
例：リブートしてからのナノ秒単位でのカウントを
64bitで保持している、など。

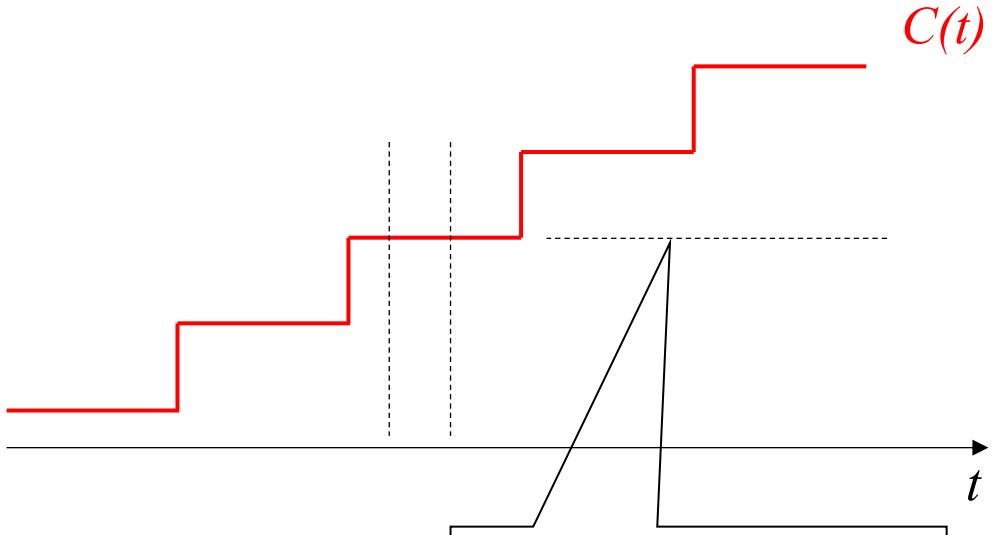


■C(t) は実時間の近似値である。
■理想的には C(t) = t であるが、実現されることは決してない。

コンピュータの時計（ソフト）

- 連続した2回の「時刻の取得」で得られる値は、
  クロックの分解能がプロセッサ・サイクル時間よりも
  十分に小さい場合にのみ、異なった値として観測される。





コンピュータシステムにおける「時間」

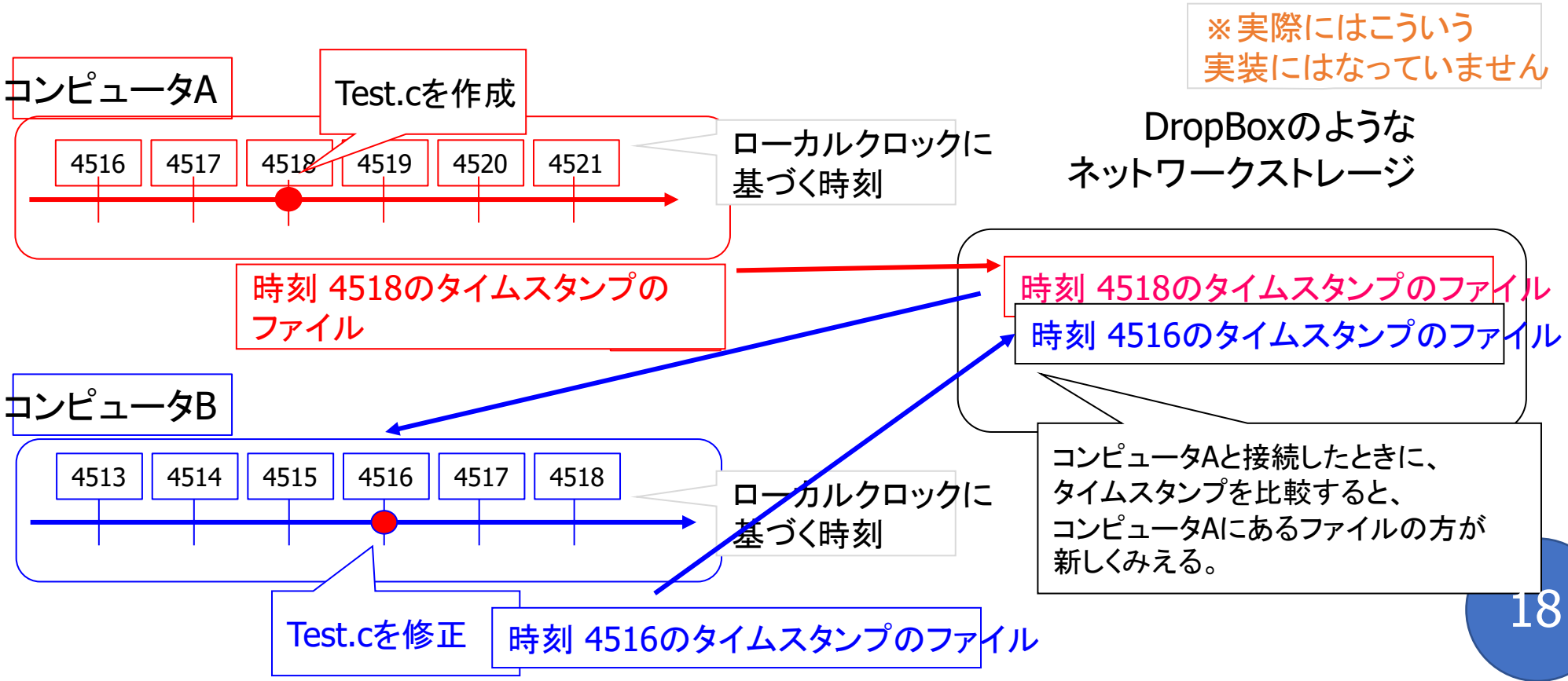
- コンピュータシステムにおいて、時間絡みで欲しい機能は様々。
  - 現在の時刻を知りたい
  - 経過時間(インターバル)を知りたい
  - どちらが先か（順序）を知りたい

- 分散システムで重要なこと：**時間的な順序関係**
  メッセージが送られたタイミング、
  プロセス実行の順序、
  資源確保の順序など

- 重要なこと:「時間」とは本来 一方向にしか進まないはず。
  システム内で これが崩れると混乱が起きる。

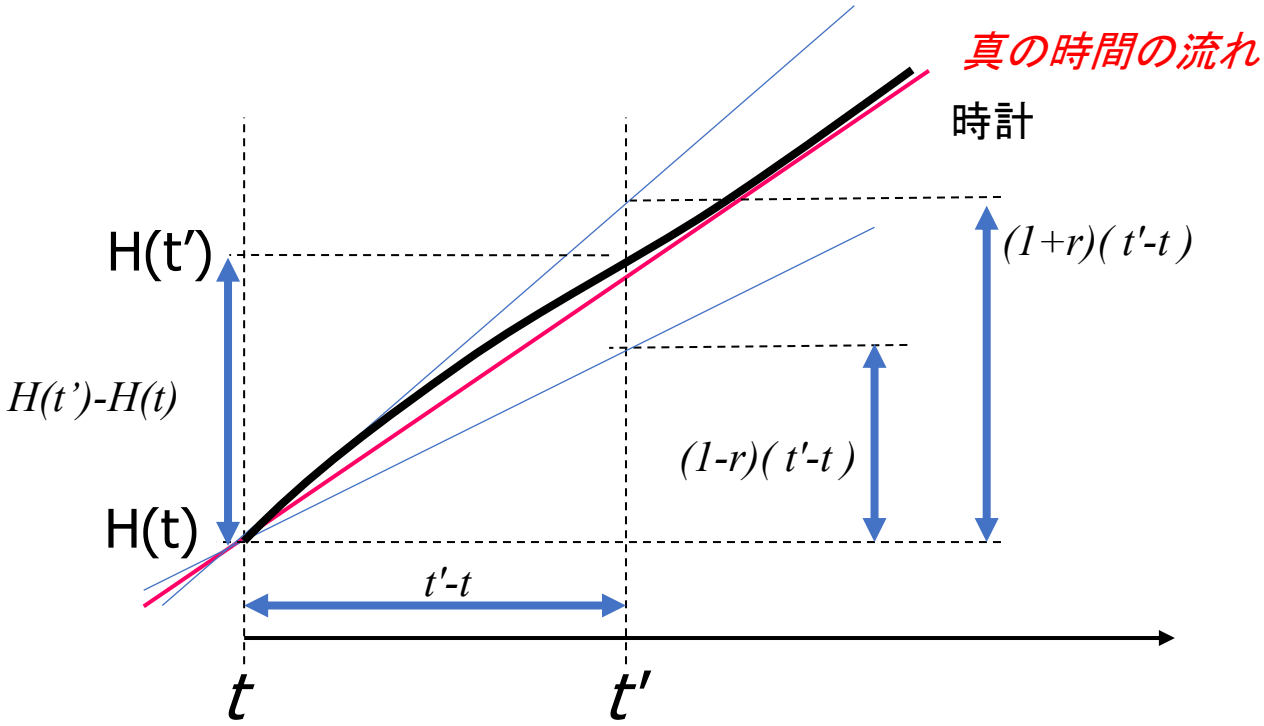
時計の同期の必要性

■コンピュータはそれぞれに時計を持っているので、
実際には後で起きた事象に対して、
より早い時刻が割り当てられることがある。



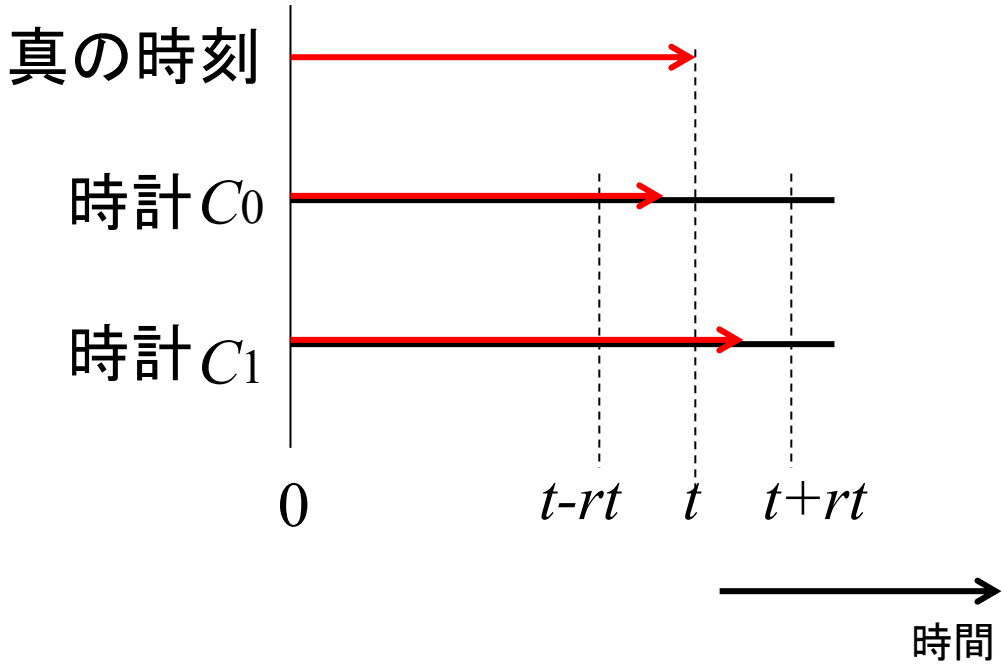
ドリフト率と時計の値の関係

- ドリフト率が r 以内(ただし r >0)であるならば、
  任意の時刻 t, t’ （ t’ > t ）における時計の値H(t), H(t’)を
  計測するとき、以下の関係が成り立つ。
  (1-r)(t’-t) ≤ H(t’) - H(t) ≤ (1+r)(t’-t)



時計の同期問題

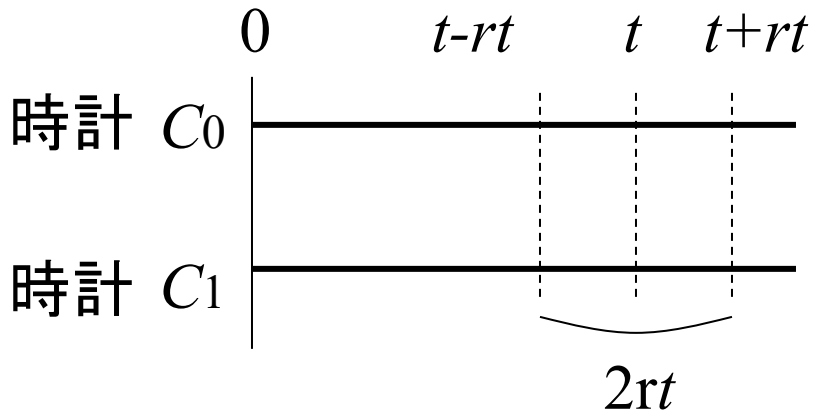
- 仮定：ドリフト率 r の時計では
  t 秒の間に最大で $\pm rt$ 秒の誤差が生じる。
- 目標：2つの時計のズレを、常に δ秒以下に収まるように
  同期させること。（δ：デルタ）



ちなみにUTCの精度
電波：0.1～10msec
GPS：1μsec

時計の同期問題（続き）

- ドリフト率rの2つのコンピュータの時計の差は、t秒の間に最大 2rt 秒になる。

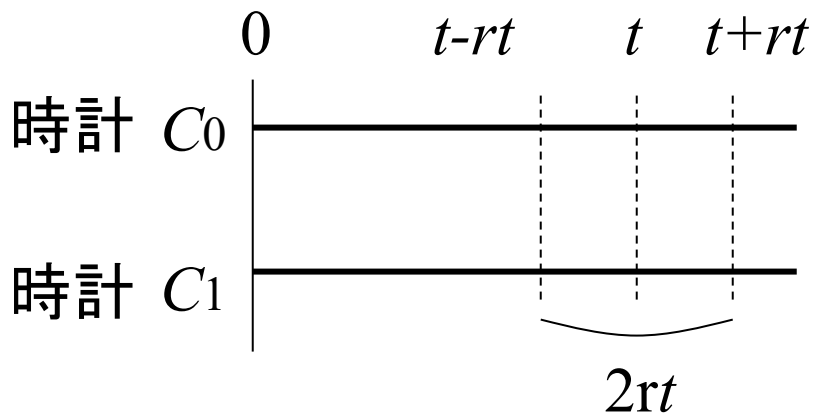


- 2つの時計の差を δ 秒以下に抑えるということは、2rt≦ δ すなわち　t ≦ δ / 2r
- 遅くとも δ/2r 秒ごとに同期（時刻合わせ）すればよい。
  - 外部同期の場合、 δ/2r秒ごとに基準サーバに同期する。

予習：時計の同期

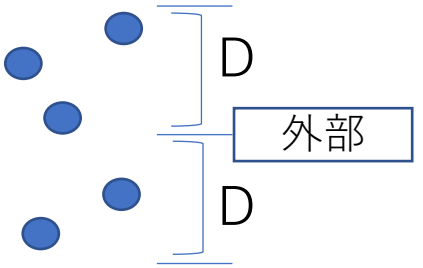
ドリフト率が 10^{-6} である2つの時計C_0とC_1がある。

- この2つの時計のズレ（スキュー）はt秒間で最大どのくらいであるか
- この2つの時計のズレを常に10^{-4}秒以下に抑えるためには、何秒ごとに同期処理を行えばよいか。

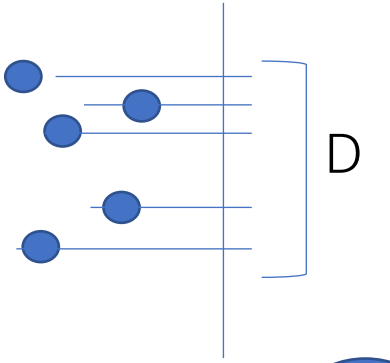


分散システムにおける時計の同期

- 外部同期
  - 分散システム内の各プロセスの時計 $C_i(t)$ を、
    外部の基準となる時計 S(t) に対して
    常に 許容誤差 D>0 以内になるように維持する。
  - $|S(t) - C_i(t)| < D$ for all i, t



- 内部同期
  - 分散システム内の相互の時計の間を
    許容誤差 D>0 以内になるように維持する。
  - $|C_i(t) - C_j(t)| < D$ for all i, j, t
  - 外部同期を利用する必要は必ずしもない。



- 外部時計に許容誤差Dで同期するシステムは、
  明らかに、内部同期としては許容誤差 2D以内に収まる。

時刻同期アルゴリズム

(物理)時計の同期

- コンピュータの時計を（タイムサーバの時計に）同期したい。
- 使用する技術
  - リアルタイムクロックのタイムスタンプ
  - メッセージ送信
  - ラウンド-トリップ タイム(RTT)

- Cristianのアルゴリズム
- Berkeleyアルゴリズム
- Network Time Protocol (Internet)

Christianのアルゴリズム

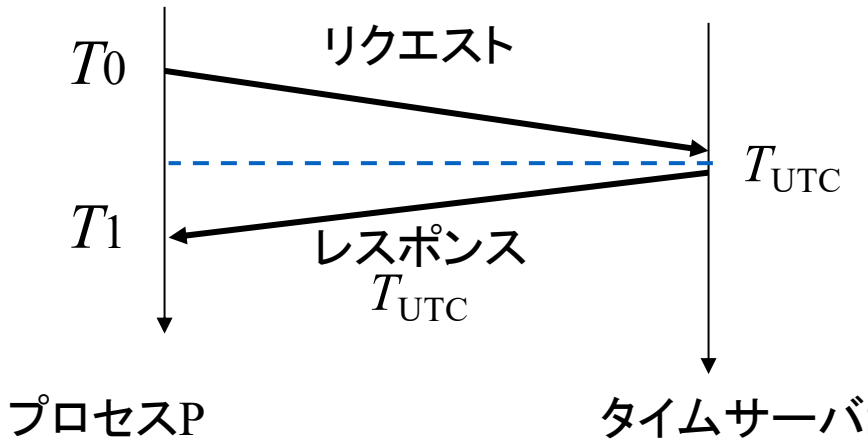
- プロセス間通信における RTTは、現実には十分に短時間である。
  しかし、理論上は無限にもなりうる（返事がこない）。

- 原理
  - 基準となるタイムサーバSを利用する。
  - プロセス P は S にリクエストを送信する。
  - RTTの値 T_{round}を計測する。
  - 時計の時刻を、レスポンスで得た時刻$T_{UTC} + T_{round}/2$ に合わせる。

- 要求される精度に対して RTTが十分に小さければ、実用的な時刻推定ができる。
  - LANの場合 T_{round} の値は 1～10msec程度。
  - ドリフト率を 10^{-6}とすれば、T_{round} の誤差は最大でも 10^{-8}秒。
    十分に高精度。

Christianのアルゴリズム（基本形）

- T_0：プロセスPが時刻の問合せを送信した時刻
- T_1：プロセスPが時刻(T_{UTC}) を受信した時刻
- 時点 T_1における現在時刻を、サーバから得られた時刻 T_{UTC} に片道の通信遅延時間 $(T_1-T_0)/2$ を加えた時刻とする。
- 前提
  - リクエストの遅延時間 と レスポンスの遅延時間は同じ。
  - タイムサーバが T_{UTC}を取り出すのに要する時間は無視できる。

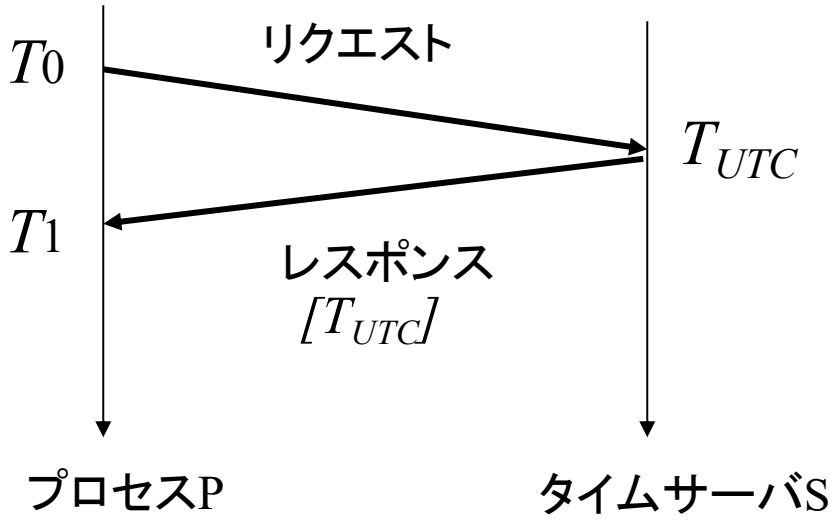


演習：Christianのアルゴリズム

- クライアントの時刻T0= 12:34:06 に時刻同期リクエストを送信したところ、クライアントの時刻T1= 12:34:10 にレスポンス [12:34:11]を受信した。

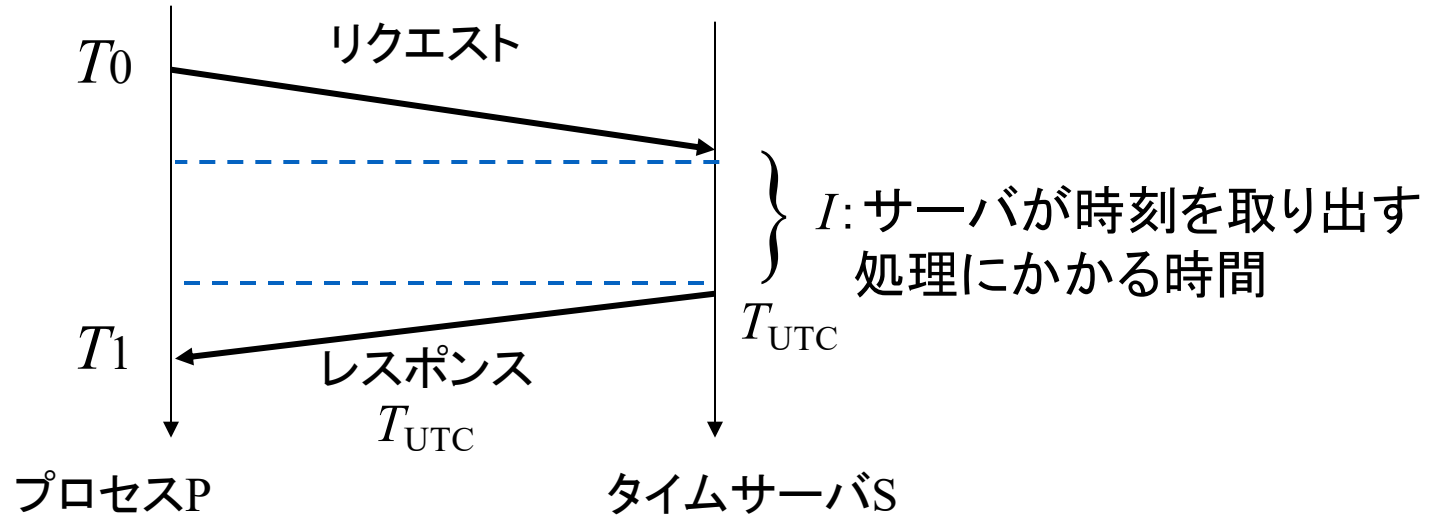
- クライアント-サーバ間の通信遅延の時間はどの程度と推定されるか

- クライアントはT1における時刻をどのような値に補正したら良いか。



Christianのアルゴリズム

- T_0：プロセスPが時刻の問合せを送信した時刻
- T_1：プロセスPが時刻(T_{UTC}) を受信した時刻
- T_1-T_0 に含まれている時間の要因：
  - RTT
  - タイムサーバが時刻を取り出す処理にかかる時間 I



Christianのアルゴリズムの精度の解析

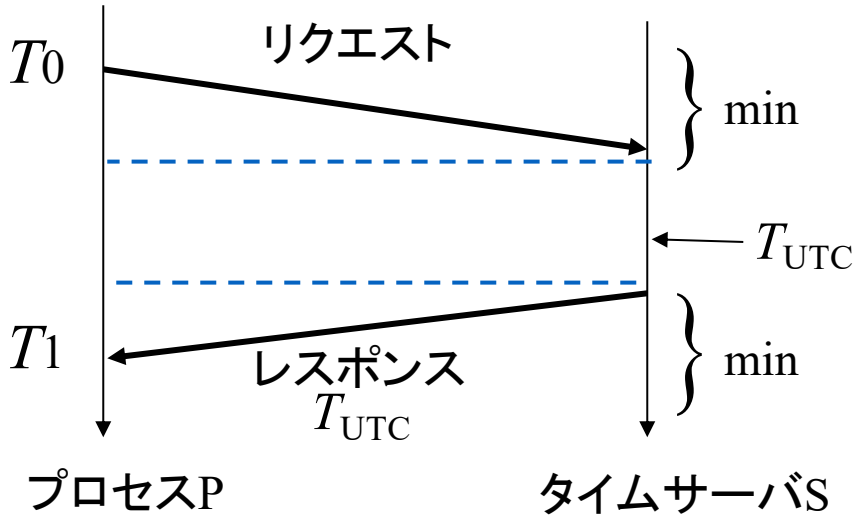
- 片道の通信遅延時間を min とする。
- $T_{round} = T_1 - T_0$ (i.e. $T_1 = T_0 + T_{round}$)
- タイムサーバSがT_{UTC}を計測している地点は、
  - 最も早いときで T_0+min
  - 最も遅い時で $T_1 - min = T_0 + T_{round} - min$

- T_{UTC} の振れ幅は

  $(T_0 + T_{round} - min) - (T_0 + min)$

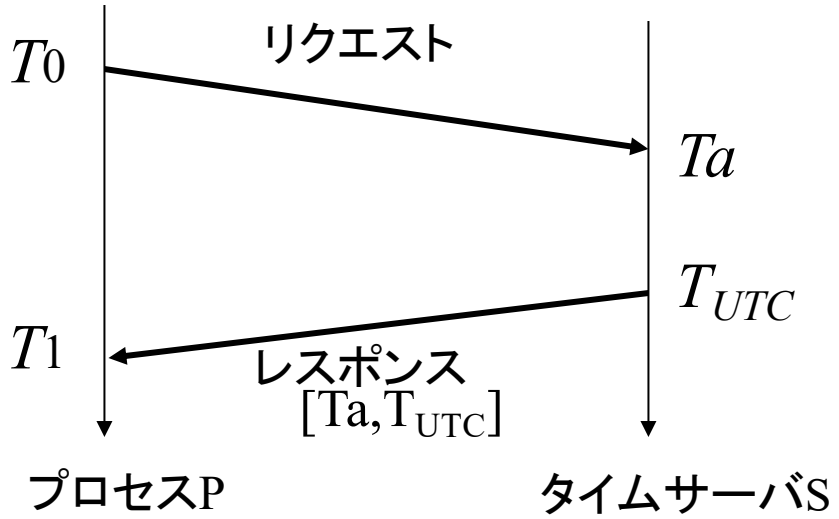
  $= T_{round} - 2 * min$

- したがって精度は $\pm (T_{round}/2 - min)$



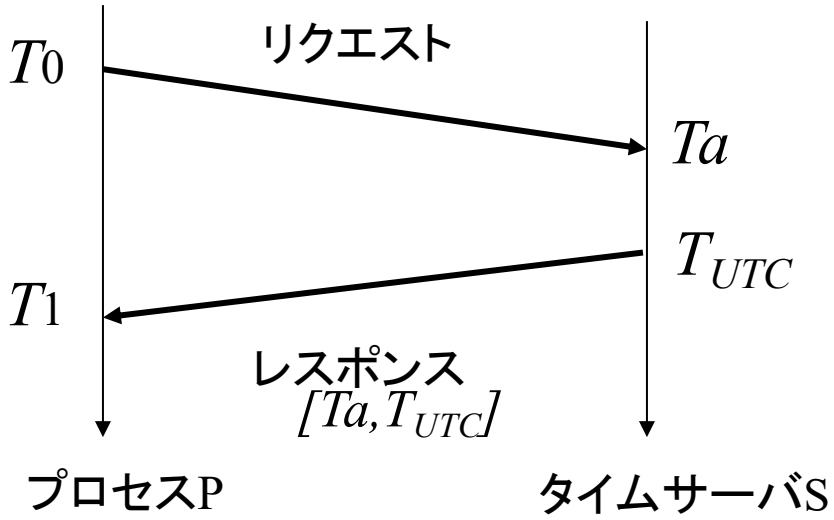
Christianのアルゴリズム

- タイムサーバSは、リクエストを受信した時刻 Taと
  レスポンスを送信する時刻T_{UTC}をレスポンスとして返す。
- $I = T_{UTC} - T_a$
- $RTT = (T_1 - T_0) - (T_{UTC} - T_a)$
- 地点T1における時刻を $T_{UTC} + RTT/2$ にセットする。



演習：Christianのアルゴリズム

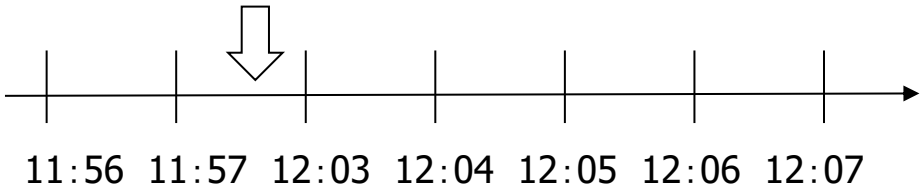
- クライアントの時刻T0= 12:34:06 に時刻同期リクエストを送信したところ、クライアントの時刻T1= 12:34:10 にレスポンス [12:34:05, 12:34:07]を受信した。
- クライアント-サーバ間の通信遅延の時間はどの程度と推定されるか
- クライアントはT1における時刻をどのような値に補正したら良いか



同期ポイントでの処理

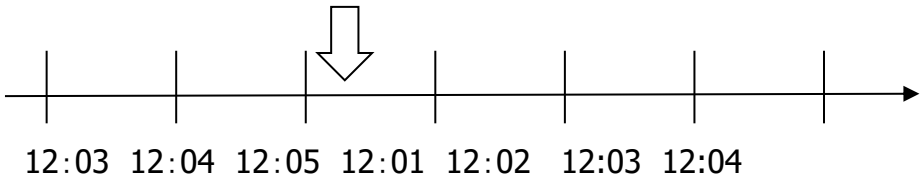
■時計の同期を行う時に
ローカルの時計の時刻を目的とする値に書き換えてしまうと、
システムによっては問題が生じる恐れがある。

1. 連続性が失われる。（時間がジャンプする）



12:00にスケジュールされていた処理が動かない？

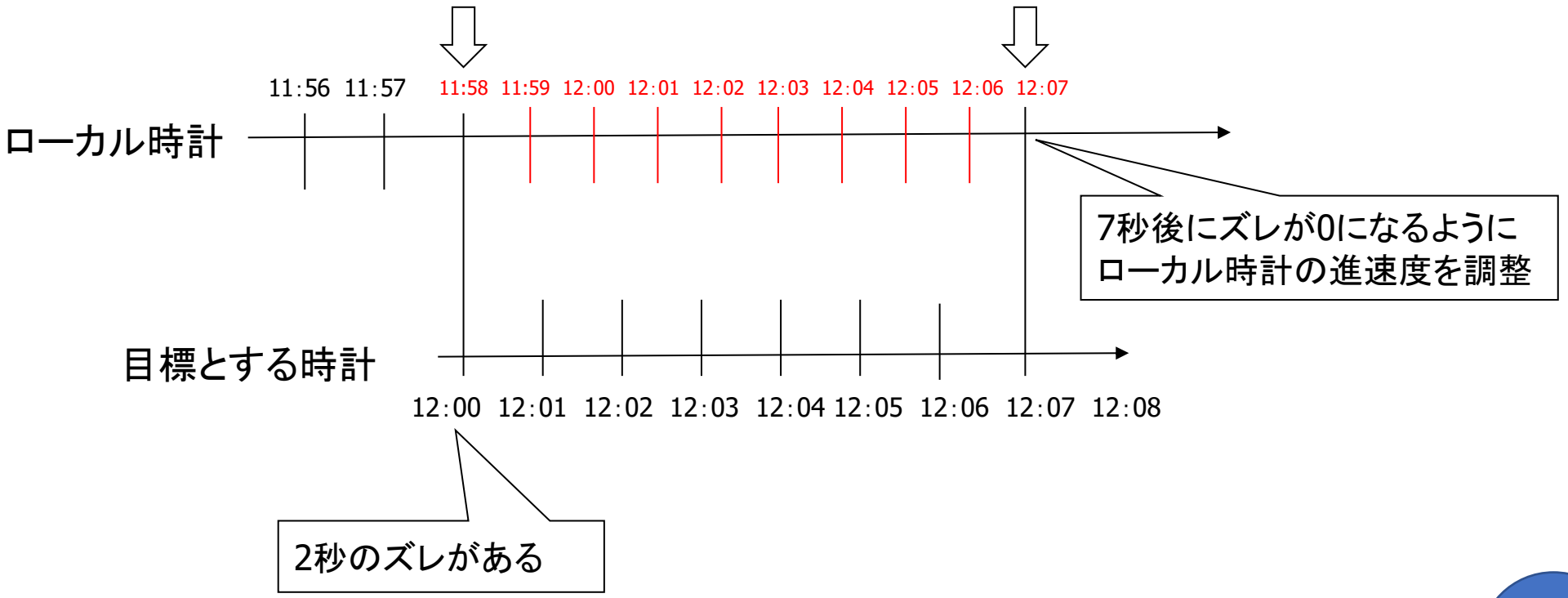
2. 逆行する。



後の処理の時刻の方が早い？
同じ時刻が2回？

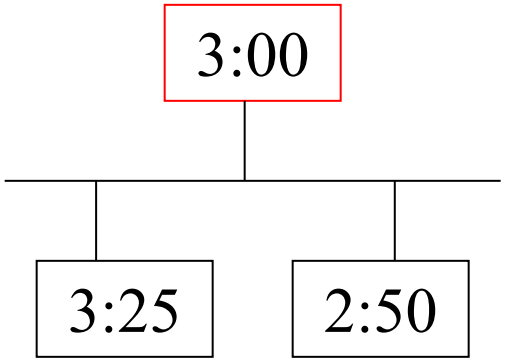
同期ポイントでの処理

- 処理の前後関係に矛盾がないように、ローカルの時計の進み方を速めたり緩めたりして、徐々に合わせる。



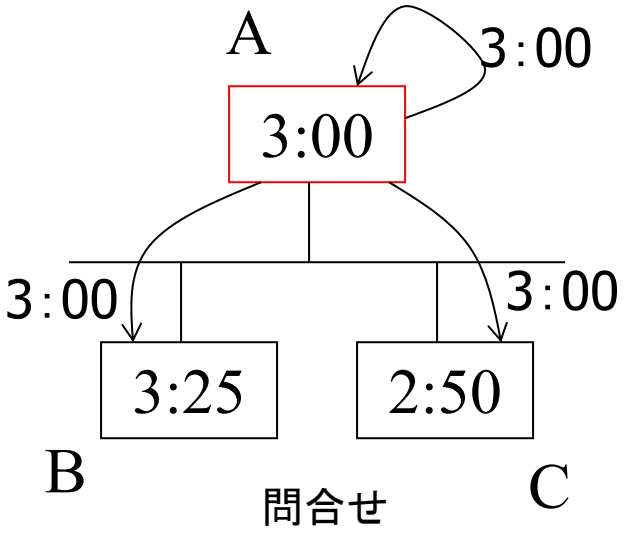
Berkeleyアルゴリズム

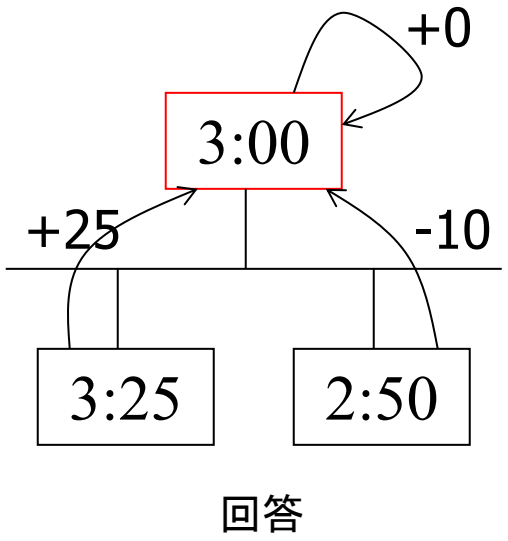
- 現在ほどInternetを超えた通信がなかった時代には、
  時刻同期は比較的狭い範囲でのサーバ同士での問題であった。
- Berkeley Unixで動作するシステムにおいて
  ネットワークで相互に接続されたサーバ同士の時計を
  自動的に合わせる方法が開発された(1989年)。
- 各サーバで動作する Clock Daemon が使用する内部時刻の同期手法
  - 外部の専用のタイムサーバはない。
  - マスターとなるサーバを1台決める。
- Christian Algorithmがベース



Berkeleyアルゴリズム

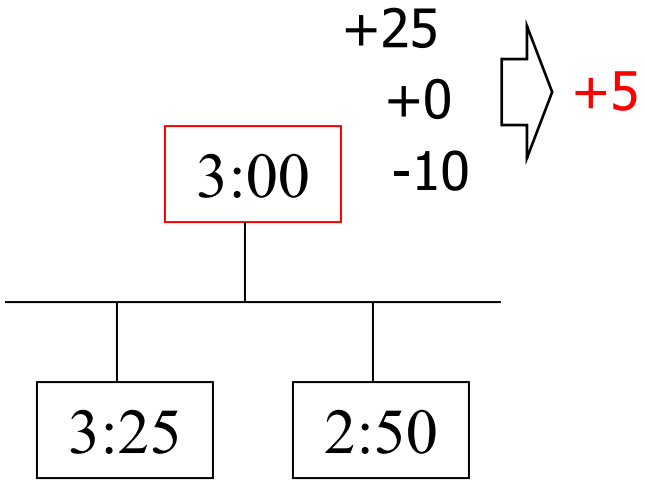
- 時刻を同期するサーバ群のMaster Clock Daemonが、定期的にSlaveサーバに対して時刻を問い合わせる。
- Slaveサーバは受信した時刻と自分の時計の時刻とのズレを回答する。

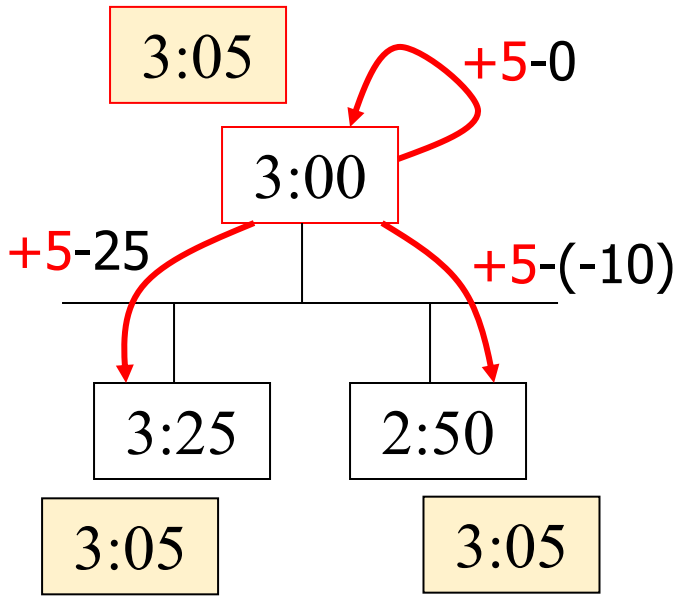




Berkeleyアルゴリズム

- Masterサーバは、回答に基づいてズレの平均を計算。
  - Christianアルゴリズムと同様に遅延を考慮する。
  - 極端に外れているものがいたら除外する。
- Masterサーバは、Slaveサーバに対して時計の修正方法を通知する。





Berkeleyアルゴリズムの短所

- Master Clock Daemon に負荷がかかる
- Master Clock Daemon が停止すると時刻同期ができない。
  →新たに別のサーバをMasterに指定することで対応する。
  （後の講義で取り上げるリーダー選出アルゴリズムを使用）

- ChristianのアルゴリズムもBerkelyアルゴリズムも
  インターネット規模の分散システムには不向き。

Berkeleyアルゴリズム

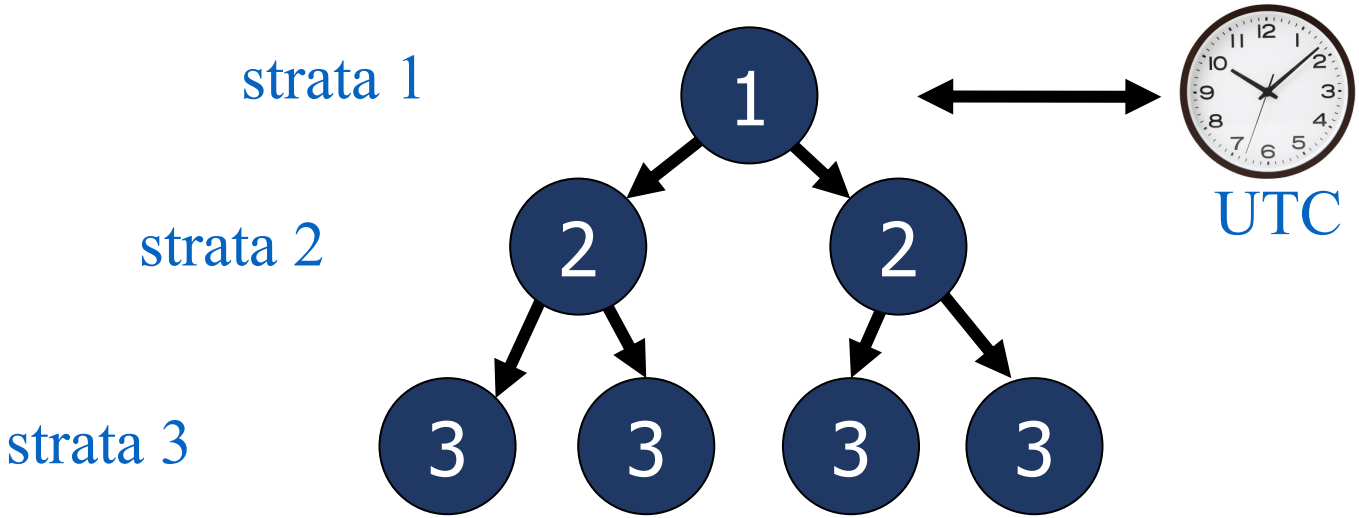
- 実例：15台のサーバの内部時刻同期
  - 個々の時計のドリフト率は $2x10^{-5}$ 程度
  - RTTの最大値は 10 ms
  - 同期にかかる時間 20～25ms

Network Time Protocolによる時刻同期

- 目的
  - クライアントの時刻をインターネット経由で
    外部サーバのUTC時刻に同期させること。
  - 通信が長期間 切れていても信頼性のあるサービスを提供する。
  - 典型的なハードウェアによるドリフト率を補正するのに十分な頻度で何度も同期できること
  - 妨害に対する防御を有すること

Network Time Protocolによる時刻同期

- UDPベースのメッセージパッシングによる、
  多層化されたクライアントサーバアーキテクチャ
- クライアントのstrata番号が大きくなるほど、
  時刻の精度は悪くなる。
  これはstrata 1サーバからの遅延が増えるからである。
- 堅牢性：strata 1 サーバがこけると、再起動後、
  他のstrata 1サーバに同期することで strata 2サーバとなる。



NTP Modes

- Multicast:
  - ひとつのコンピュータがネットワーク上の他のコンピュータに定期的に時刻をマルチキャストする。
  - 通信点が非常に小さいことが前提
  - 高速なLANに向いている。
- Procedure-call:
  - Christian's のプロトコルに似た手法
  - サーバはクライアントからのリクエストに対応して回答する。
  - 高い精度が必要な場面、マルチキャストが利用できない場面で使用される。
- Symmetric:
  - とても高い精度が必要な時に使用される

今日のまとめ

- 正しい時間は、分散システムが正しく動作するためにとても重要

- 時間と時刻と時計
  - 時計の正しさ（スキュー、ドリフト）
- 時刻同期問題
  - Berkeley アルゴリズム
  - Christian アルゴリズム
  - NTP