分散システム 第15回
― まとめと復習 ―

大連理工大学・立命館大学 国際情報ソフトウェア学部

大森 隆行

※このスライドで復習すれば100点が取れるわけではありません

授業の目的

- 分散システム(distributed system)に関する基礎知識の習得
  - 分散システムを構築するために必要な概念やアルゴリズムについて説明できる
  - また、それらを分散システムの構築にどのように適用できるかについて説明できる
  - 分散システムの実例を取り上げ、その機能や構成について説明できる

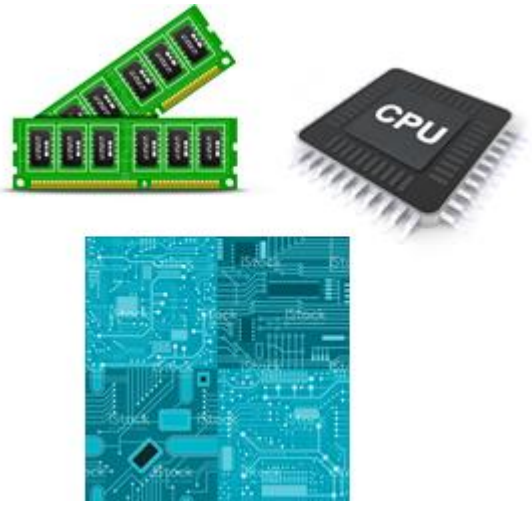
分散システム

- 分散システムはそのユーザに対して
  単一の首尾一貫した（coherent）システム
  として見える
  独立したコンピュータの集合である
- A distributed system is
  a collection of independent computers
  that appears to its users
  as a single coherent system.

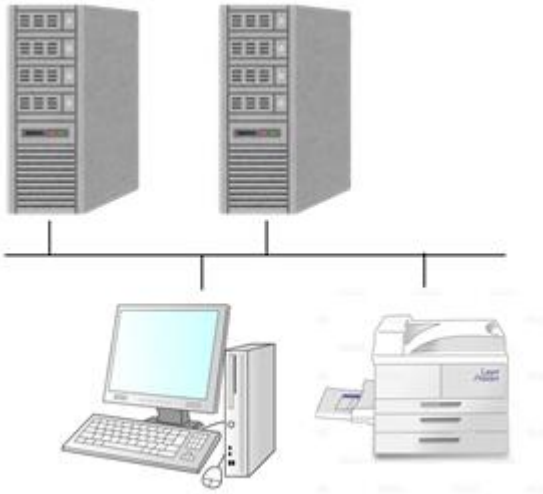
* A.S. Tanenbaum et al.
"Distributed Systems - Principles and Paradigms -"

分散システム

- 独立したコンピュータの集合
  → 単体でも自律している(autonomous)
  コンピュータの集合



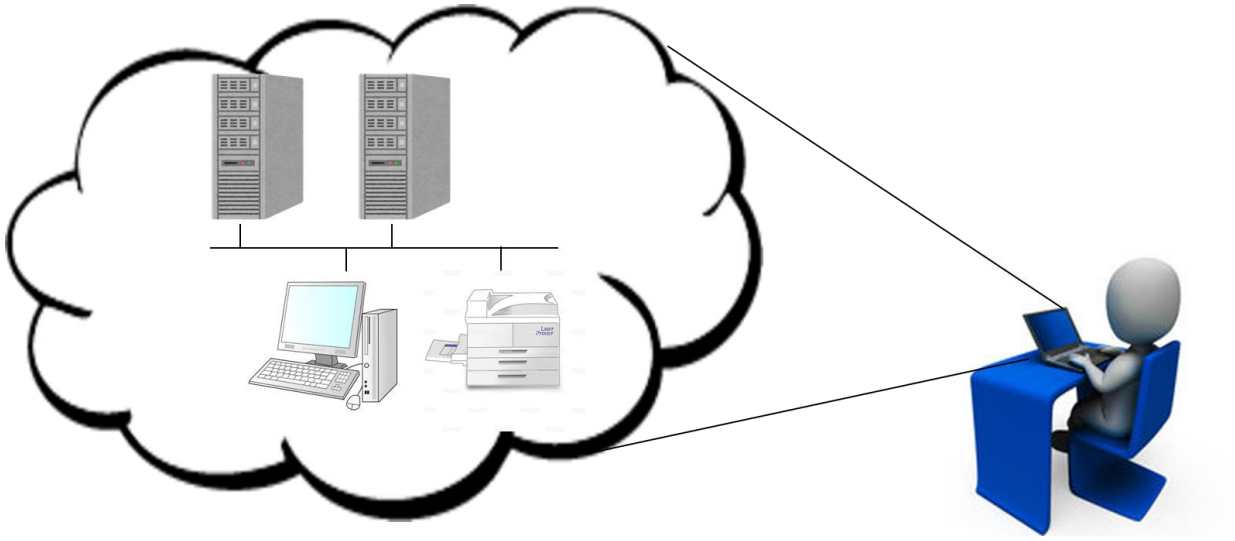
自律していないものの
集まり（＝部品）



自律したものの集まり

分散システム

- 単一の首尾一貫した（coherent）システム



- 複数のコンピュータで構成されていても
  利用者はその詳細は知らない(知る必要がない)
  → 個々のコンピュータは協調しなければならない

分散システムの種類

- 分散コンピューティングシステム
- 分散情報システム
- 分散パーベイシブシステム

分散コンピューティングシステム

- 目的：
  高性能計算、信頼性の高いデータ保管等
- クラスタコンピューティング cluster computing
  - 複数のPCやワークステーションで構成
  - 高速LANで結合
- グリッドコンピューティング grid computing
  - ネットワーク上に分散した
    複数のサブグループにより構成
  - 複数の管理組織にまたがる

分散情報システム

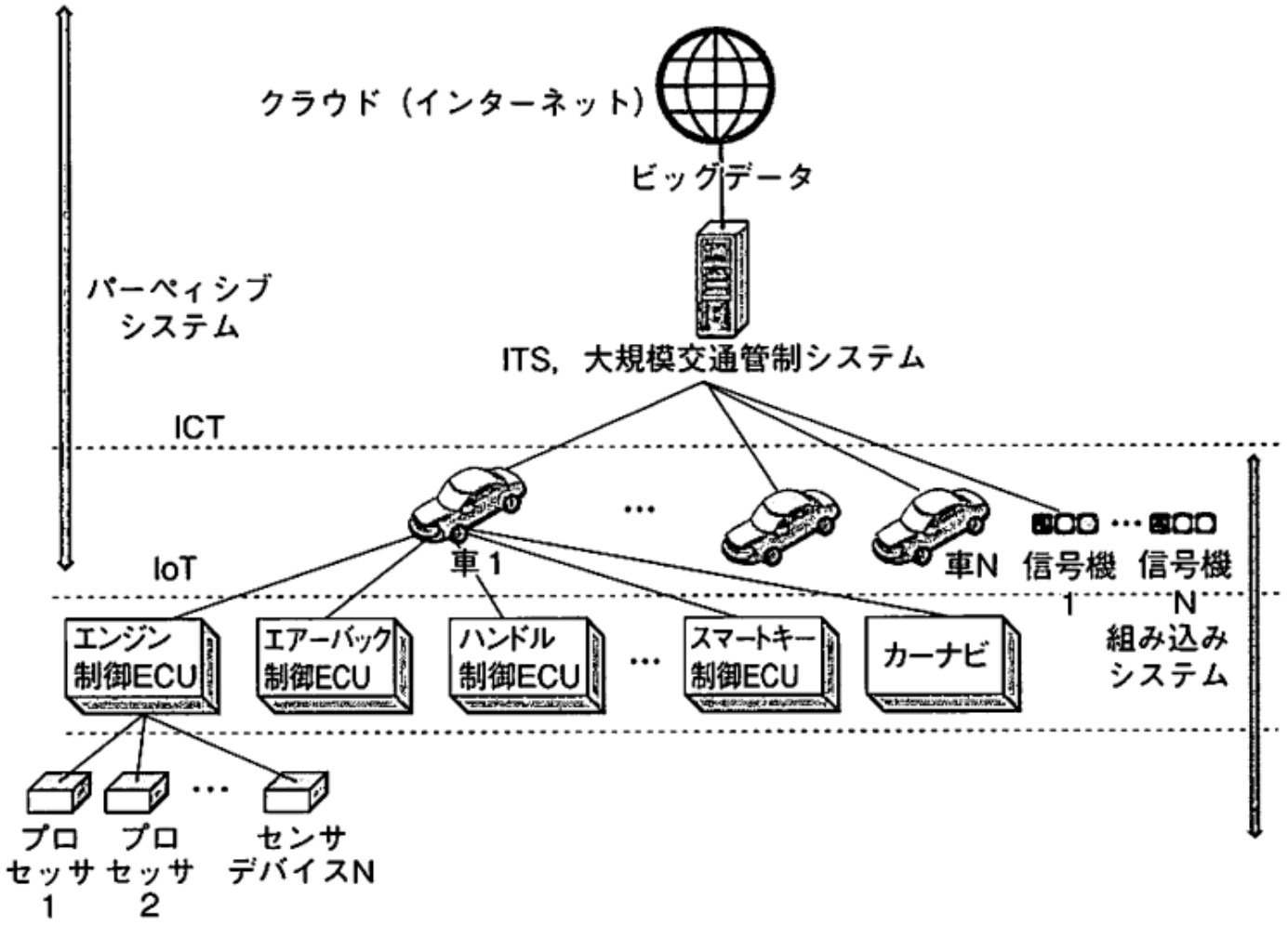
- 大学・企業等による
  ネットワークアプリケーションの提供
- トランザクション処理システム
  - 遠隔からのデータベース操作
  - 操作はトランザクション単位で実行
- エンタプライズアプリケーション統合
  (EAI: enterprise application integration)
  - 企業内で使用される複数の分散アプリケーションを統合
  - 遠隔手続き呼び出し(RPC)や
    遠隔メソッド呼び出し(RMI)等を利用

分散パーベイシブシステム

pervasive: 遍在する

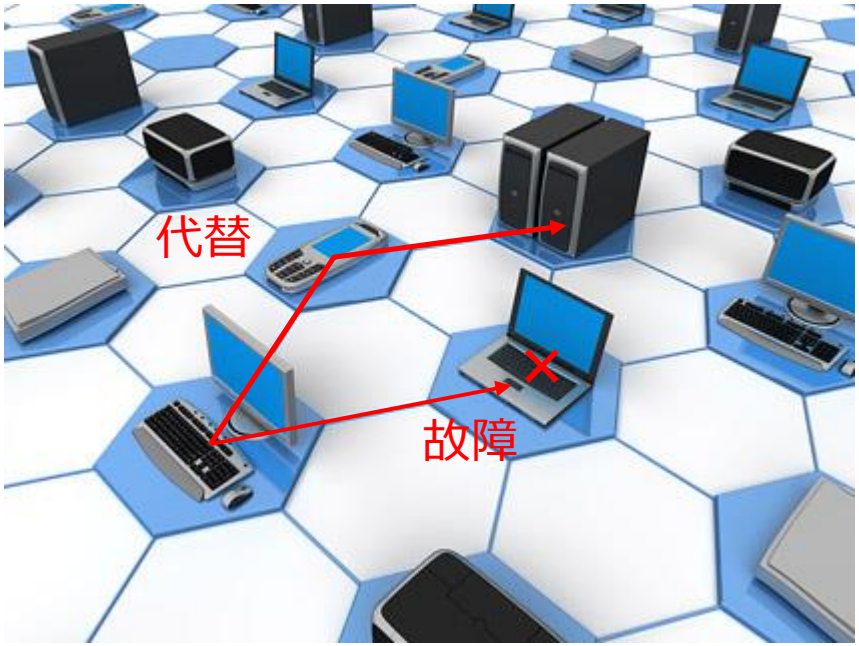
- ユビキタスシステム(ubiquitous system)ともいう
- 小型のセンサやその他の機器を
  互いに連結
- 要素技術
  - 無線アクセス (無線LAN、Bluetooth等)
  - RFID(radio frequency identification)
    - 無線認証を用いて物体の認証を行う
  - センサネットワーク
  - 位置特定技術 (GPS等)

自動車と交通システムによる分散パーベイシブシステムの例



分散システムの目的 - 可用性

- システムのサービスを利用したいときに利用できる
  - 分散システムでは、1台が故障してもサービスの継続が可能



分散システムの目的 - アクセス可能性向上

- ユーザ(またはアプリケーション)が遠隔地にある資源(リソース)へのアクセスをすることを可能にする
  - リソースの例：プリンタ、コンピュータ、ストレージ、装置、データ、ファイル、Webページ、ネットワーク
- 経済性：個別にリソースを保持するより安価
- 情報交換のしやすさ
  - グループウェア(groupware)による共同作業等

分散システムの目的 - 局所性

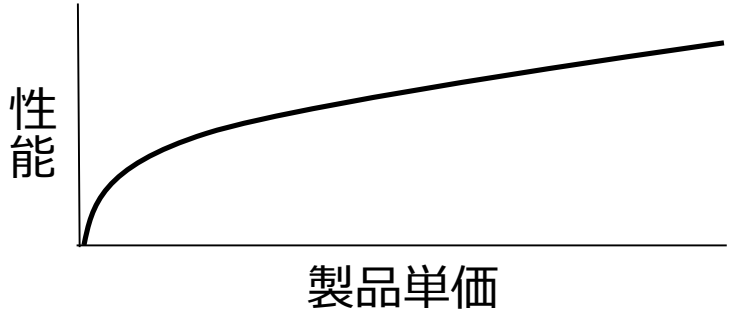
- 保護すべき情報を局所的に保存

- 例：電子メール
  - 学内宛のメールは学内サーバで対応
  - 学外宛のメールは学外のサーバへ転送
  - メール処理をするサーバが1箇所に集中していると、宛先に関わらずそのサーバへ送ることになる

分散システムの目的 - スケーラビリティ

- コンピュータの数を増やすことで
  変化に柔軟に対応

- 集中システムの問題
  - ユーザ数が10倍に増えたら？



10倍の性能を得るためのコストは
10倍では済まない

  - 稼働中に規模を拡大したくなったら？
    - 設計段階で規模拡大を想定しておく必要

分散システムの設計目標

- 分散透過性 (distribution transparency)
- 開放性 (openness)

位置に関する透過性

- 位置透過性 (location transparency)
  - システム内のどこにリソースが物理的に存在しているかを隠蔽する
  - (例)http://www.ritsumei.ac.jp/では，立命館のサーバがどこにあるのかを意識させない
- 移動透過性 (migration transparency)
  - リソースを移動しても同じアクセス方法を使用可能
- 再配置透過性 (relocation transparency)
  - 利用中の移動が可能
  - (例)無線パソコンをネットワークに接続したまま移動できる

システム構成に関する透過性

- 複製透過性 (replication transparency)
  - 可用性や性能を向上させるために配置する
    コピーを意識させない
- 並行透過性 (concurrency transparency)
  - 共有リソースへの同時アクセスに対して
    一貫性を保証する
- 障害透過性 (failure transparency)
  - リソースの障害と回復を意識させない

透過性の度合い

- 完全な透過性を提供することは技術的に不可能
  - (例) San Franciscoのプロセスと Amsterdamのプロセスの間で35ms以下で通信したい → 不可能
- 完全な透過性を提供すれば良いとも限らない
  - (例) 一部のサーバで障害
    → 隠蔽しようとすると全体のサービスの速度が低下

分散システムの設計目標 - 開放性

- プロトコル(protocol)が開かれていること
  - プロトコルに従った機器を誰でも開発可能
  - プロトコルに従った機器は相互運用可能(interoperable)
  - プロトコルに従った機器を別のシステムで利用可能(portable)
- プロトコルが開かれていることにより、機器の置き換えや拡張が容易になる

陥りやすい誤解

- ネットワークは信頼性がある
- 遅延は存在しない
- 帯域幅は無限である
- ネットワークはセキュアである
- トポロジーは変化しない
- 存在する管理者は一人である
- 転送コストはゼロである
- ネットワークは均一である

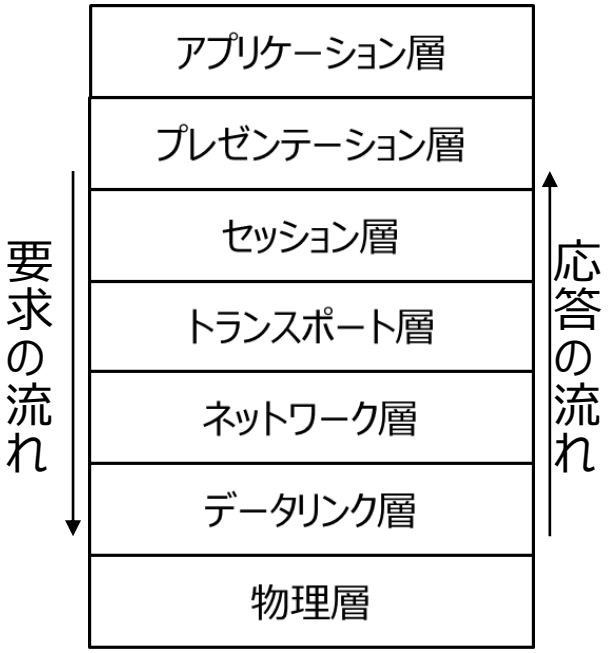
分散システムとアーキテクチャ

- 分散システムは複雑になりやすいため、適切に設計することが重要
  - 様々なソフトウェア部品から構成される
  - それらが様々なコンピュータ上に配置される

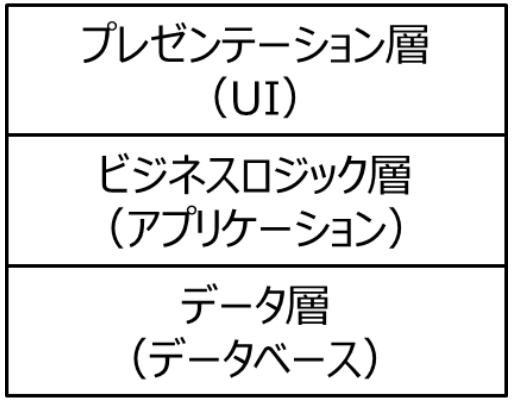
- アーキテクチャが重要
  - ソフトウェアアーキテクチャ
  - システムアーキテクチャ

階層型アーキテクチャ

- i番目の層は、直下の層（i-1番目の層）に対して、サービスの要求を行う



ISO 7層モデル

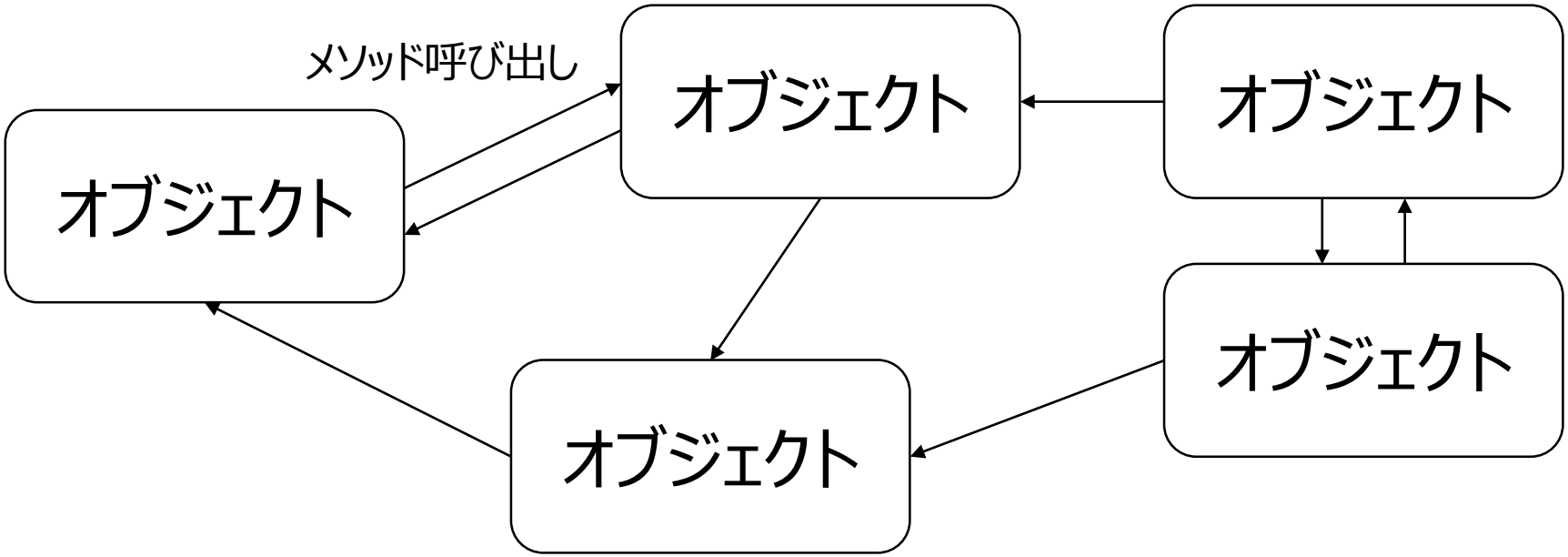


Webアプリケーション
3層モデル

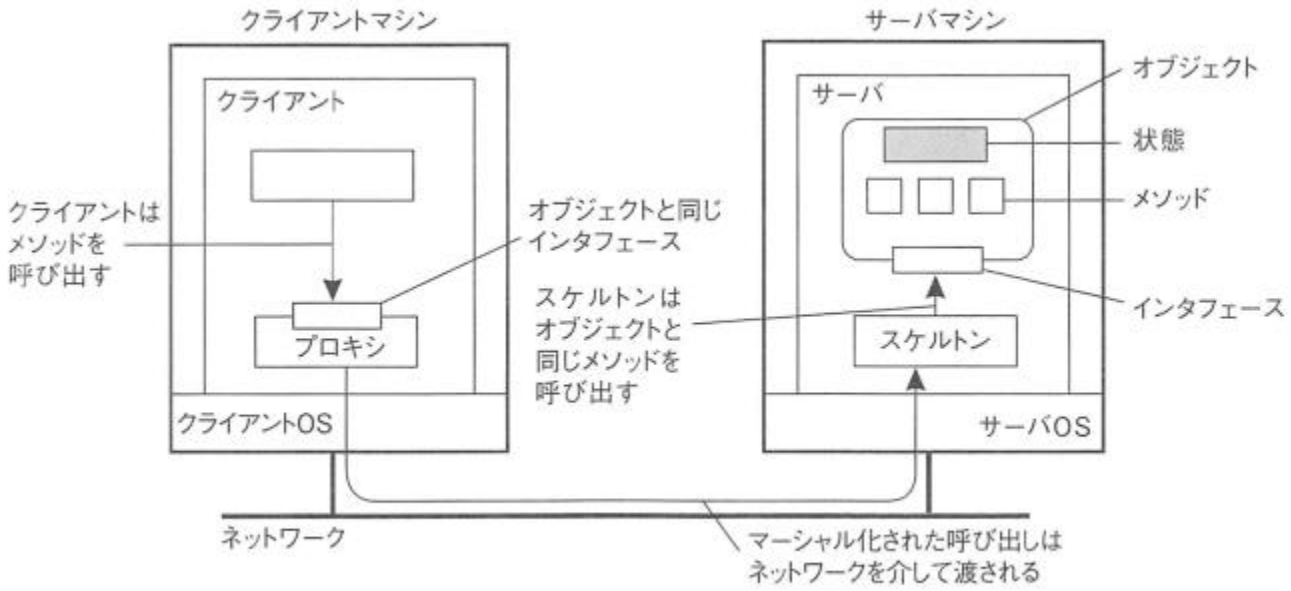
仮想マシンモデル

オブジェクトベースアーキテクチャ

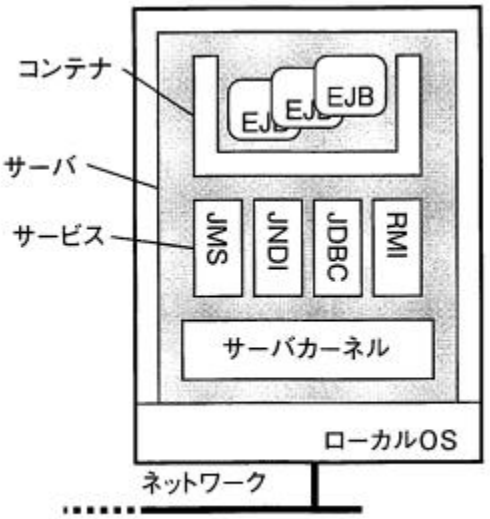
- オブジェクトをコンポーネントとする
  - 上下関係はない
- クライアントサーバともよくマッチする



分散オブジェクトの利用



クライアント側にプロキシを持つ遠隔オブジェクトの構成例

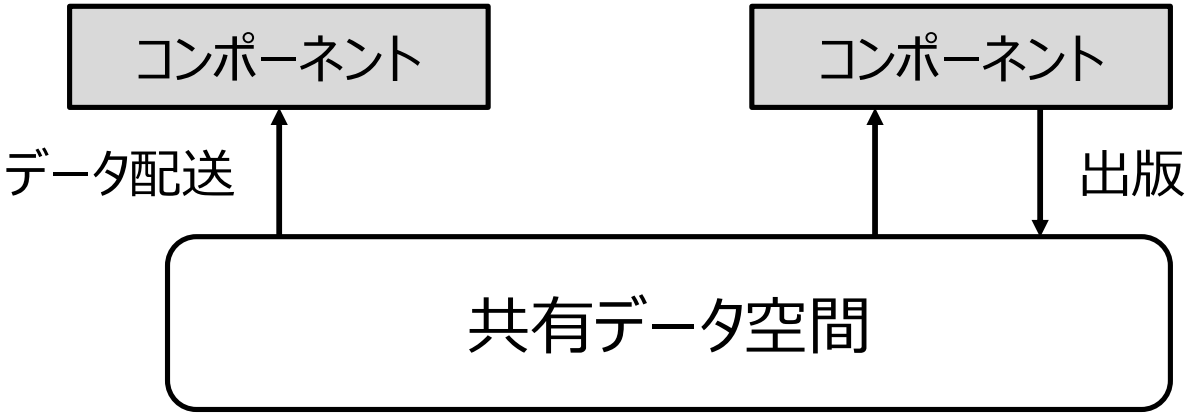


EJBサーバの一般的なアーキテクチャ

EJB Enterprise Java Beans
RMI Remote Method Invocation
JDBC Java DataBase Connectivity
JNDI Java Naming and Directory Interface
JMS Java Message Service

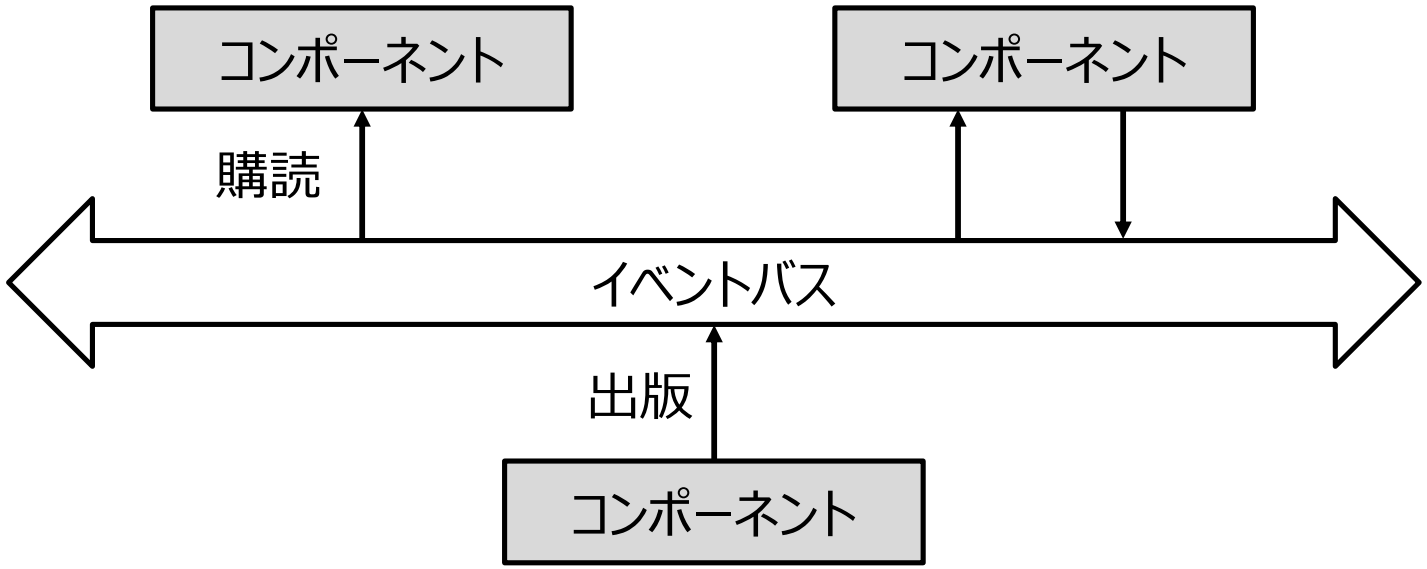
データ中心型アーキテクチャ

- 共有リポジトリ(データベース)への書き込みや参照によって通信を行う
- 一貫性を持ってデータを管理するシステムとよく適合する



イベントベースアーキテクチャ

- コンポーネント同士がイベントを介して通信
  - 互いを明示的に参照する必要がない (疎結合)



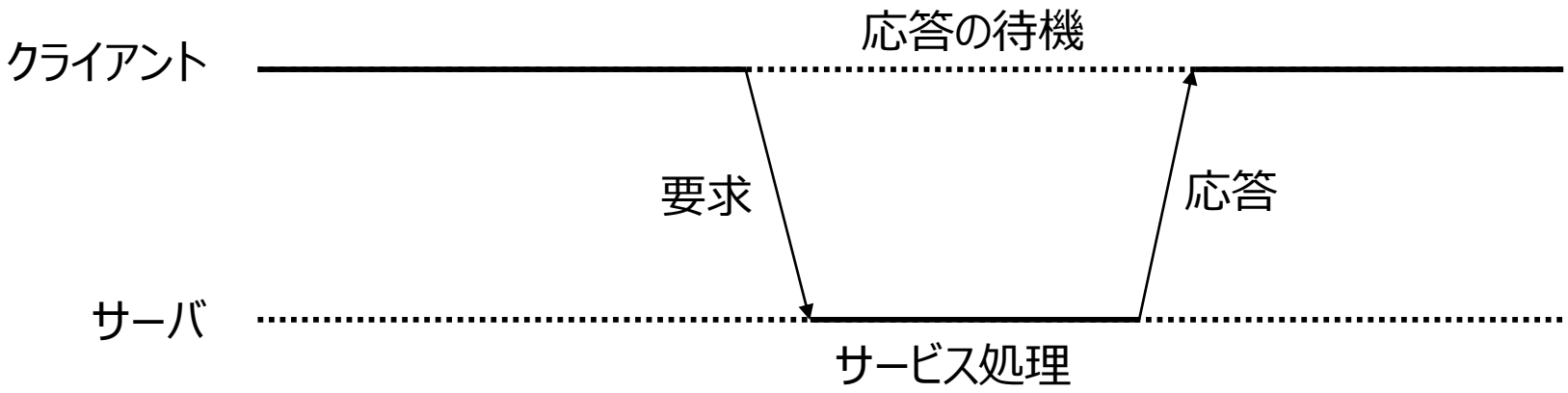
システムアーキテクチャの例

- 集中型(centralized)
  - クライアントサーバ
    - 役割の異なるグループ(クライアント・サーバ)により構成
- 非集中型(decentralized)
  - ピアツーピア(peer-to-peer)
    - 原則として各ノードが対等

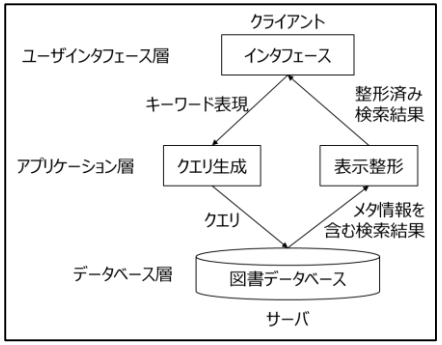
クライアントサーバモデル

- システムを2つの異なる役割を持つグループ
  クライアントとサーバに分ける

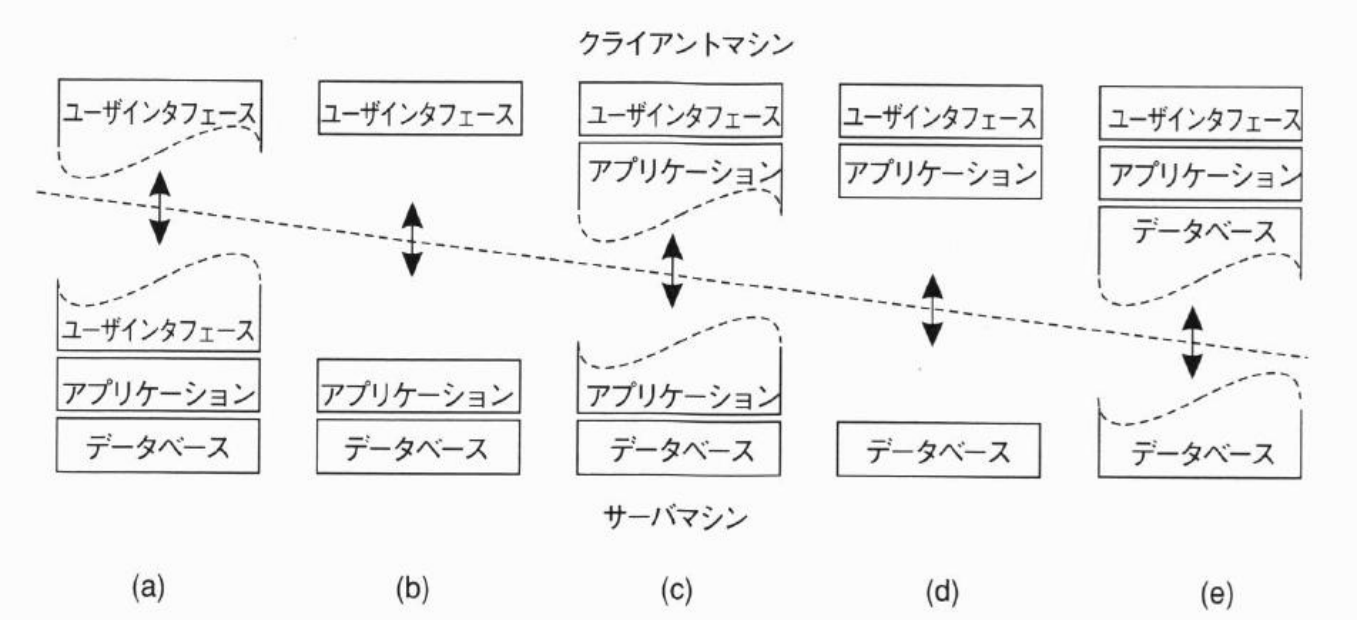
- サーバ(server)
  - サービスを提供する端末やアプリケーション
    - (例) ファイルサーバ、印刷サーバ、Webサーバ…
- クライアント(client)
  - サービスを要求する端末やアプリケーション



クライアントサーバモデル



- クライアント・サーバの分け方も様々



シンクライアント
(thin client)

- クライアント性能が低いとき有効
- 通信量増大

ファットクライアント
(fat client)

- クライアント性能が高いとき有効
- 通信量減少
- データの一貫性やセキュリティの問題

垂直分散・水平分散

- 垂直分散 (vertical distribution)
  - 役割が異なる構成要素に分割
  - クライアントサーバモデルが該当
- 水平分散 (horizontal distribution)
  - 対等な構成要素に分割
  - ピアツーピアモデルが該当
  - (例) 大規模Webシステム
    同一のコンテンツを持ったWebサーバを複数立ち上げ、負荷が集中しないように接続要求を振り分ける

サーバの分類

- 反復サーバ (iterative server)
  1つのクライアントからの要求を処理し終えると次の要求を受け付け可能となる
  - 短時間の接続で済む場合には有効
- 並行サーバ (concurrent server)
  複数のクライアントからの要求を同時に(並行して)受け付け可能
  - 複数のプロセス(orスレッド)で対応する
  - 無限に受け付け可能なわけではない

サーバの分類

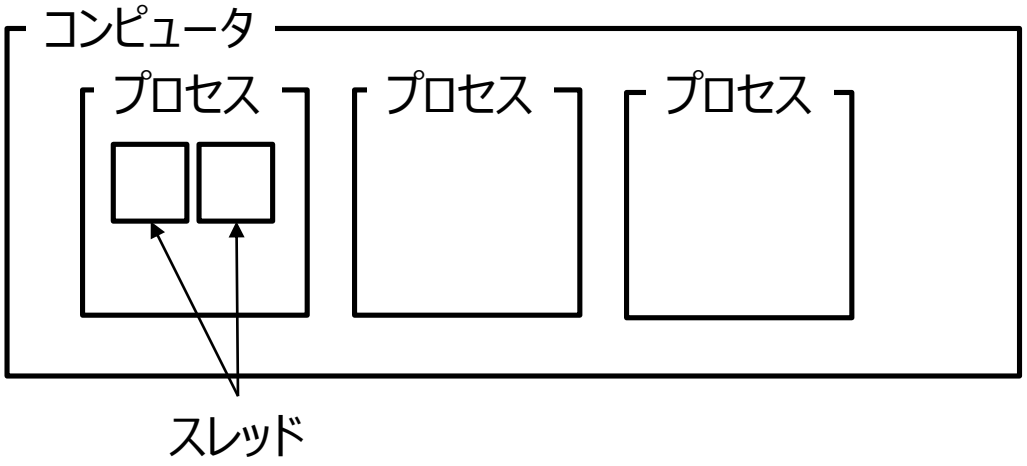
- ステートフル (stateful)
  - サーバが接続に関する状態管理を行う
  - 状態に関する情報のやり取りを減らすことができるが、クライアント増加の際の負荷増大が大きい(スケーラビリティで劣る)
- ステートレス (stateless)
  - サーバが接続に関する状態管理を行わない
  - (例) HTTPサーバ
    - Cookieを使えば状態管理可能

プロセスとプログラム

- プログラム：
  - 処理の手順をプログラミング言語で表したもの (＝ソースコード)
  - 実行可能なアプリケーション、実行ファイル
- プロセス：
  - プログラムに従ってプロセッサが実行する一連の動作・処理
    - プログラムのインスタンス

マルチスレッド

- 1つのプロセスが複数のスレッドにより実行される
  - スレッド(thread)：
    プロセス内部の実行の筋(線)
  - 各スレッドは並列的に実行される



プロセスの実行

- OSはプロセステーブルでプロセスを管理
  - レジスタの値、メモリ割り当ての状況、オープンされたファイル 等々
- 複数のプロセスが同一リソースを共有する場合もある → OSが管理
- プロセス生成は高コスト
  - 1プロセスに独立したメモリ空間1つ
  - 複数のプロセスでのCPUの切り替え

スレッドの実行

- スレッドの実行
  - スレッドもプロセスと同様、それぞれ独立して実行
  - 高度な並行透過性は実現しない
    - (例) 複数スレッドから同一データにアクセスする場合、支障がないようにプログラムの開発者が注意する必要
  - ユーザ空間でスレッドを実装
    - OSはプロセス内部に複数のスレッドがあることを知らないので、複数のプロセッサは割り当てられない
  - カーネル空間でスレッドを実装
    - カーネル(kernel)：OSの中核部分
    - スレッドごとにプロセッサを割り当てることも可能だが、OSの設計・実装時にマルチスレッドに対応しておく必要

通信プロトコル

- 通信には標準化されたプロトコルが必要
  - 異なるプロトコルで通信すると…？
    - e.g., 異なる文字コード、各社固有のプロトコル
- プロトコルは様々なレベルで必要
  - e.g., ビット列の伝送を行うための電気的規格、電子メールをやり取りするときの手順

→階層化プロトコルが必要

プロトコルの種類

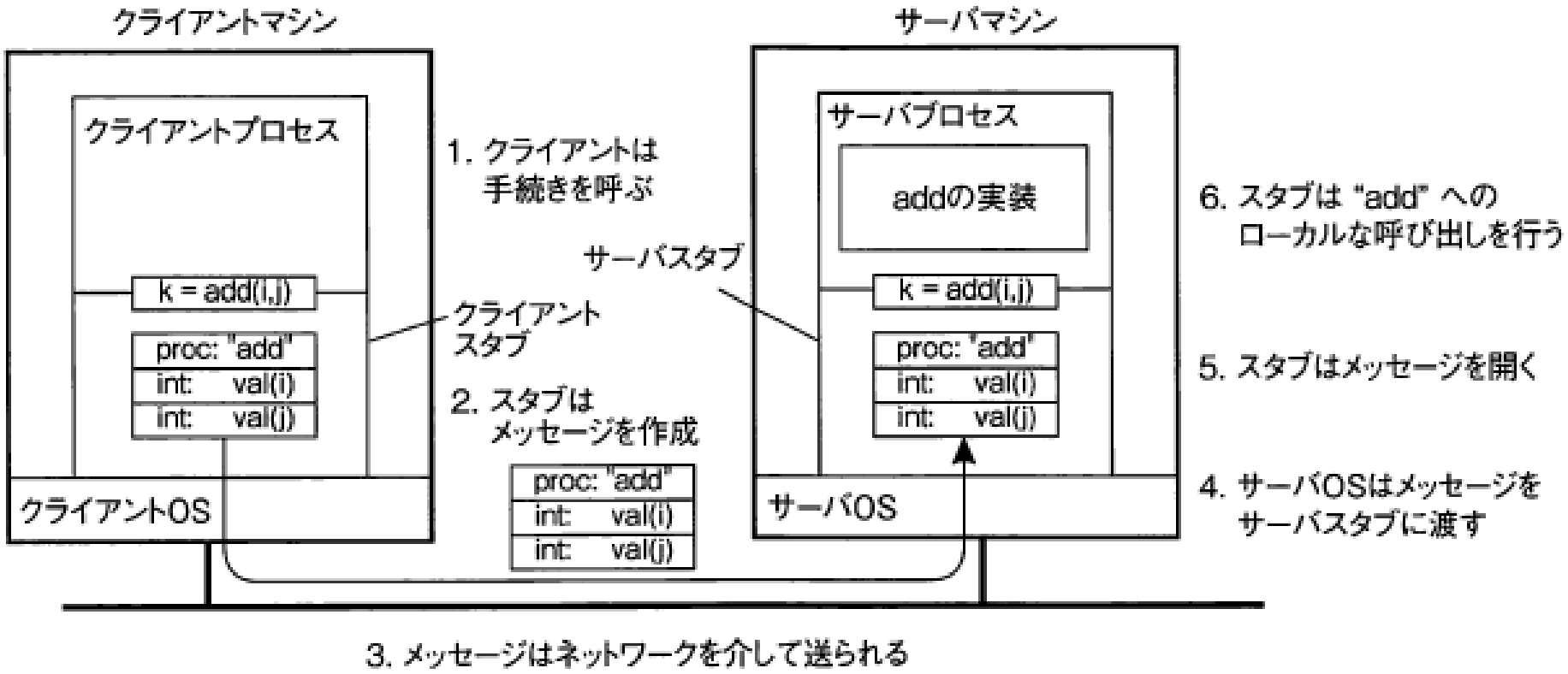
一般的なプロトコルの区別として、以下の2つがある

- コネクション型
  - データ交換前に明示的にコネクションを確立
  - 同期的な通信が可能
  - (例) TCP
- コネクションレス型
  - コネクション確立は不要
  - 送信者の準備ができれば送信
  - 非同期通信に使用
  - (例) UDP

遠隔手続き呼び出し(RPC)

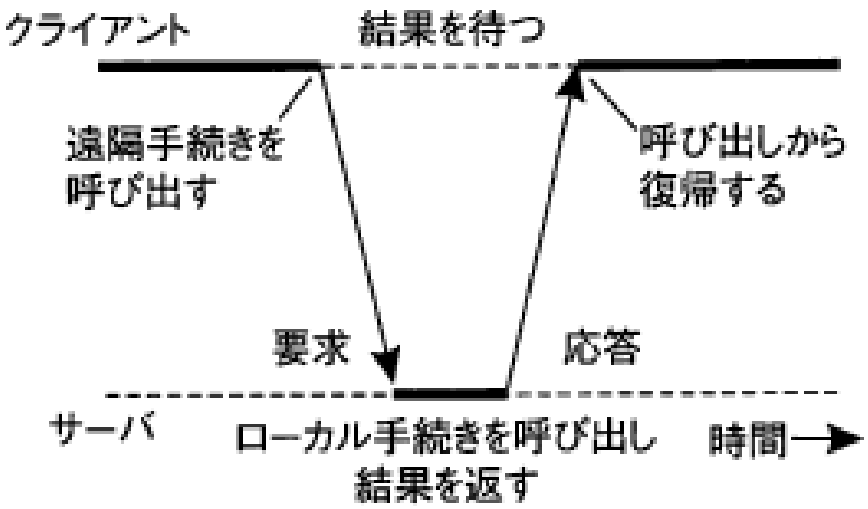
- RPC: Remote Procedure Call
  - ネットワークを介して他のマシンの手続きを呼び出す仕組み
  - プログラマの立場から見るとメッセージ交換が隠蔽される
    - 通常の関数呼び出しと同じように見える

RPCによる遠隔通信

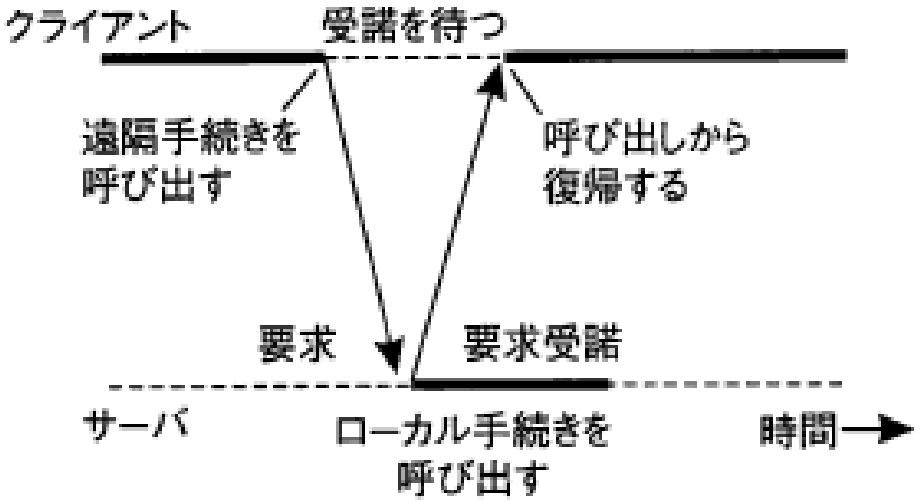


- スタブ：上位モジュールに呼ばれるモジュールのこと
(RPCでは通信を隠蔽する役割を担う)
- パラメータマーシャリング(parameter marshaling)：
パラメータを1つのメッセージに包み込むこと
  - 引数の解釈方法はサーバ、クライアントで同じでなければならない

非同期型RPC



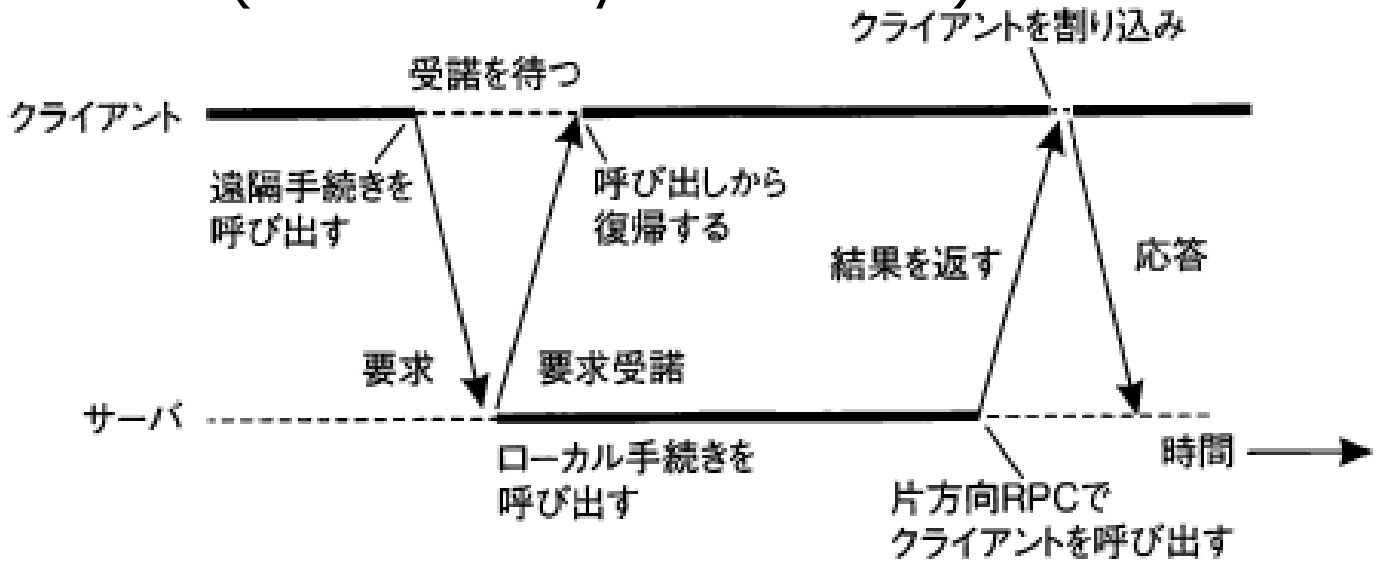
同期型RPC
(Synchronous RPC)



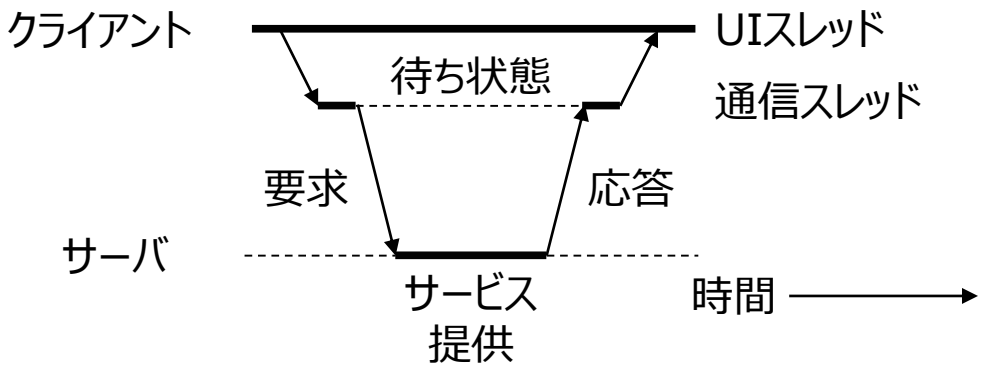
非同期型RPC
(Asynchronous RPC)

非同期型RPC

- 保留同期型 (Deferred Synchronous) RPC



- (参考) Ajax (Asynchronous Javascript and XML)



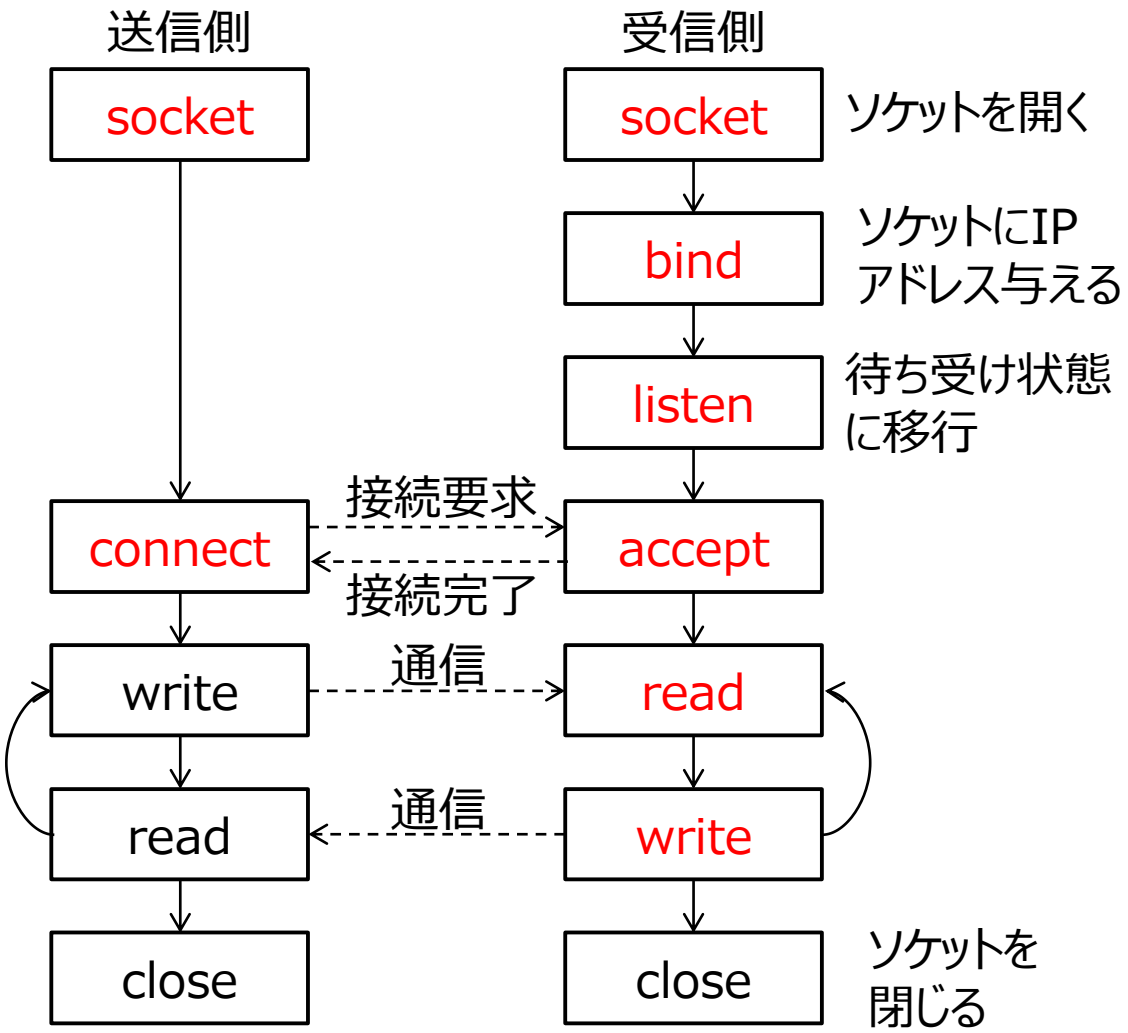
メッセージ型一時通信

- Berkeleyソケット
  - トランスポート層のインタフェースを標準化
  - 1970年代 Berkeley UNIXで導入
  - ネットワークを介するプロセス間通信を抽象化
- ソケット
  - 下位のネットワークを通じて送信されるデータへの書き込み、受信されるデータの読み込みができる通信のエンドポイント(endpoint)

確認問題

- 右図はソケット通信の過程を図示したものである。
空欄にあてはまる適切なものを選択肢から選んで答えよ。

選択肢：
accept, bind,
connect, listen,
socket,
read, write

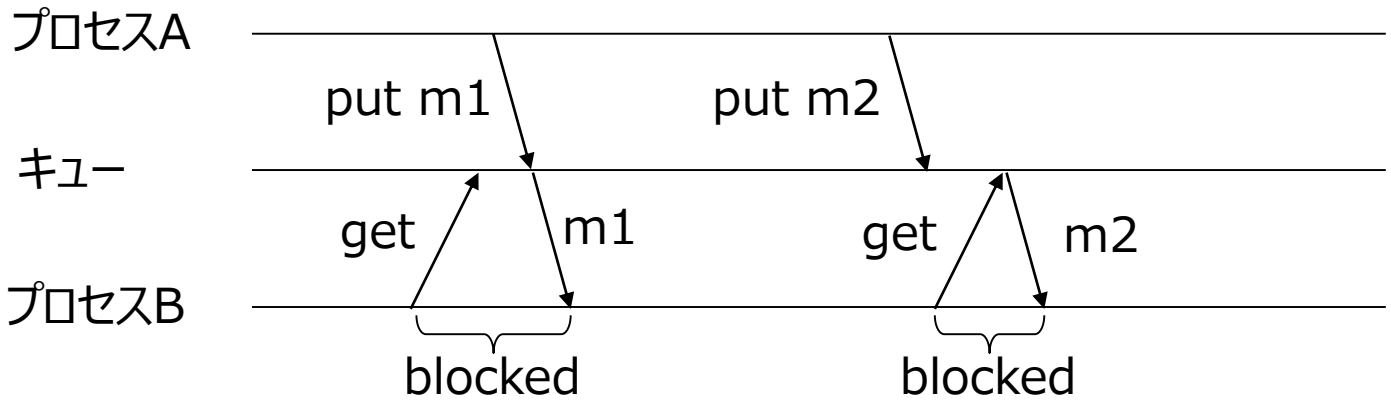


通信の種類

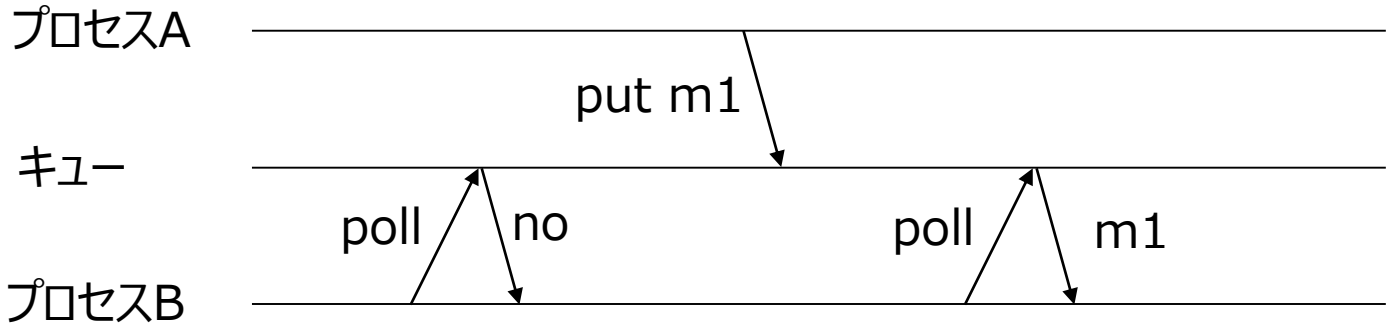
- 一時通信 (transient communication)
  - 受信側はメッセージ投入時に実行されていなければならない
  - (例) 電話
  - (例) RPC、ソケット通信
- 永続通信 (persistent communication)
  - 受信側はメッセージ投入時に実行されている必要がない
  - メッセージは”ある場所”に蓄積される
  - (例) Eメール

メッセージキューイングシステム

キューの操作(put & get)

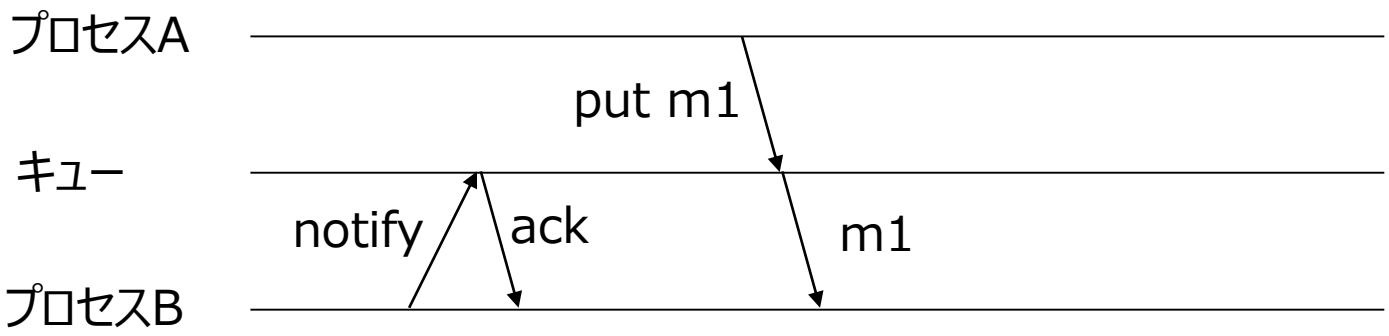


キューの操作(poll)



メッセージキューイングシステム

キューの操作(notify)



ストリーム型通信

- (例) オーディオストリーム、ビデオストリーム
- 原音が44100Hzでサンプリング
  → 1/44100秒で演奏しなければ不正確な再生になる
- 時間依存の情報を交換するための分散システムは
  どのような機能を持たなければならないか？

- 連続表現メディア(continuous representation media) :
  データアイテム間の時間的な関係がデータの正確な
  解釈のために必須
  - (例) 動画：順番に画像を表示しなければならない
- 離散表現メディア(discrete representation media) ：
  データアイテム間の時間的な関係は必須ではない
  - (例) テキスト情報、静止画像

データストリーム

- データストリーム：データの流れ
  - 連続メディアをサポートするためにタイミングの扱いが必須

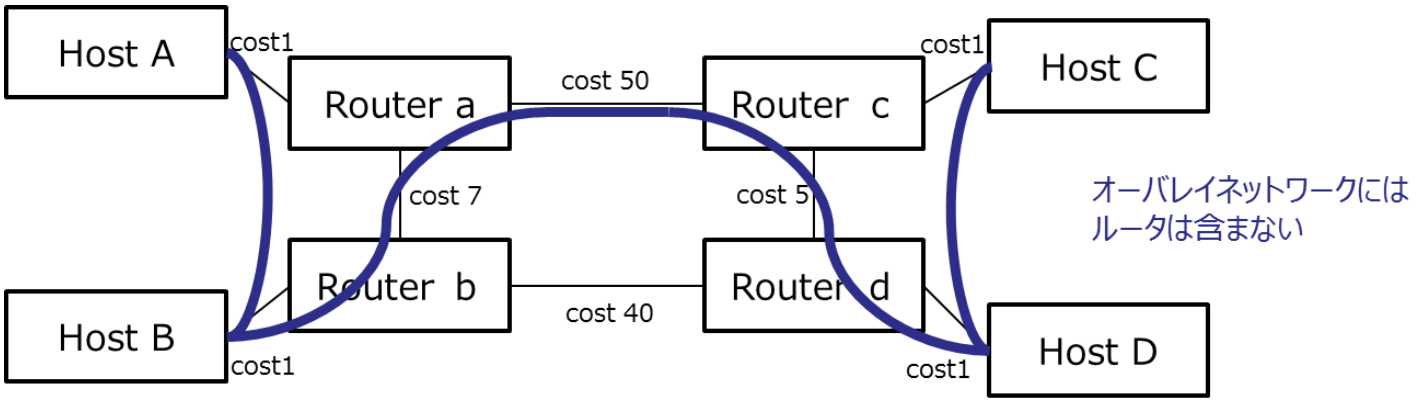
- 非同期型転送モード(asynchronous transmission mode)：アイテムの送信は順々に行われるが、いつ送信されるべきかについての制約はない
- 同期型転送モード(synchronous transmission mode)：端末間で最大遅延が定められる(データがそれよりも早く到着してもOK)
- 等時性転送モード(isochronous transmission mode)：時間通りにデータを転送しなければならない(遅延時間の幅が定められる)
  - オーディオ、ビデオの配信において特に重要

ストリーム間の同期

- 離散データストリームと連続データストリーム間
  - それほど厳しい要件ではない
  - (例) スライドショーに音声を付ける
- 連続データストリームどうし
  - ステレオオーディオストリーム
    - サンプリング周期44100Hz
      → 1/44100 = 23マイクロ秒ごとに同期
  - ビデオとオーディオ(リップシンクロ)
    - 1フレームごとにオーディオ情報をグループ化することができる(カラーテレビの規格30Hzだと、33ミリ秒単位)
- 実際にはストリームごとに遅延の大きさが異なる
  → 送信側でストリームをマージ(多重化)

マルチキャスト通信

- 同時に複数のマシンと通信
  - ブロードキャストとは違って、決められたマシンに対して送信
- アプリケーションレベルのマルチキャスト
  - 各ノード間にオーバレイネットワークを構成
    → その構成員に情報を転送
  - オーバレイネットワーク(overlay network)：実際のネットワーク上に作る仮想的なネットワーク



名前・アドレス・識別子

- (分散システムにおける)名前：エンティティ(実体)を識別するために使われるビット列または文字列
  - エンティティ：ユーザ、端末、Webサーバ、メッセージ、プロセス等
  - エンティティは分散システム内で通信相手や処理の対象になりうる → 参照できなければならない
    → そのために名前が必要
  - 名前とエンティティを対応づけることを名前解決(name resolution)という
- アドレス(address)：エンティティにアクセスするためのアクセスポイントの名前
  - (例) 友達に手紙を送るときに使う住所、
    友達に電話をかけるときに使う電話番号

名前・アドレス・識別子

- 識別子(identifier)：
  エンティティを一意に識別可能な名前
  - 一つの識別子は複数のエンティティを参照することはない
  - 各エンティティはたかだか一つの識別子により参照される
  - ある識別子は常に同じエンティティを参照し、再利用されない
    - 名前、アドレスは再利用される可能性がある

たかだか： at most

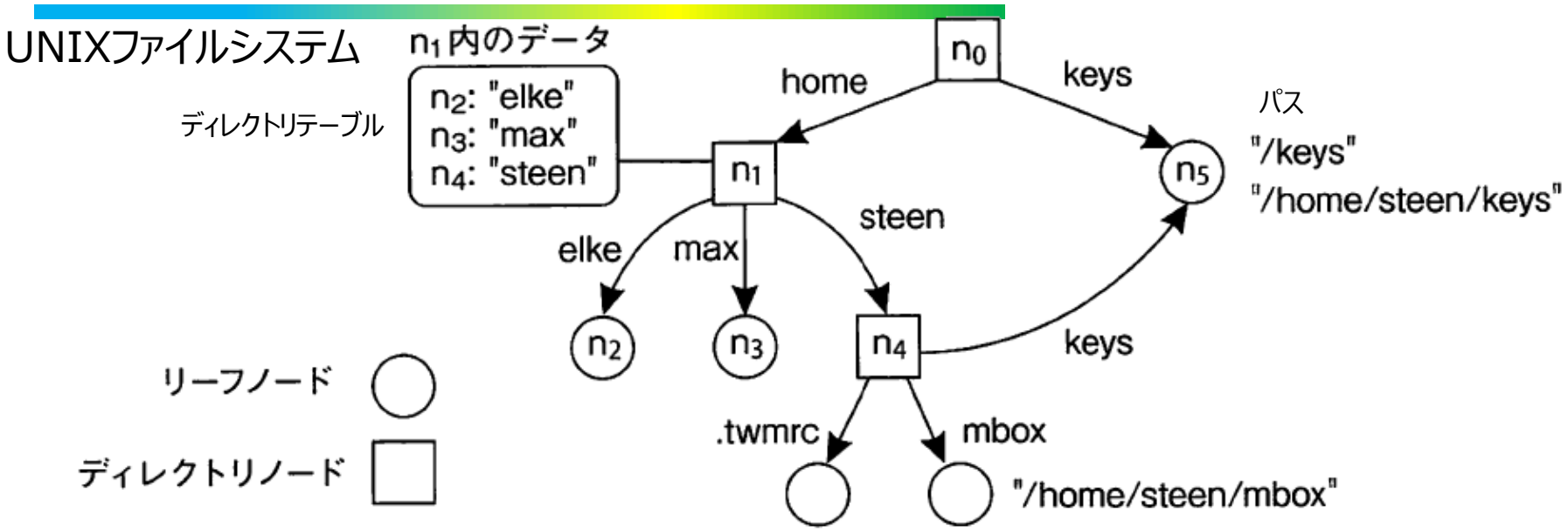
フラットな名前付け

- フラット(flat): 平らな＝構造化されていない
  - 名前自体に名前解決に役立つ情報は含まれない
  - (例) MACアドレス

通信機器に固有の番号

- 名前解決
  - ブロードキャスト
  - マルチキャスト
  - 転送ポインタ
  - ホームベースアプローチ

構造化された名前付け



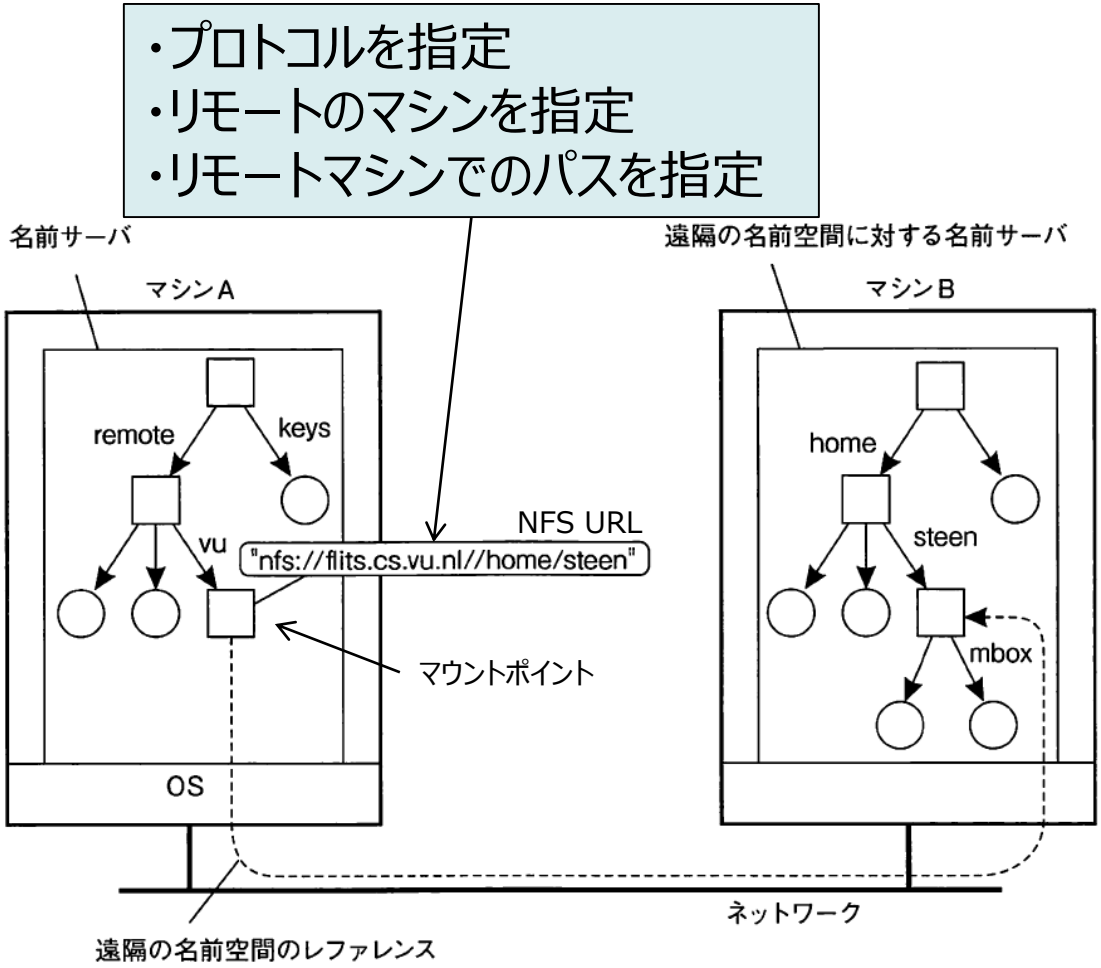
- 2種類のノードからなるラベル付き有向グラフ
  - リーフノード(葉ノード)：エンティティを示す
  - ディレクトリノード：出力辺の
    (辺のラベル、行き先ノードの識別子) の組を表形式で保持
    - この表をディレクトリテーブル(directory table)と呼ぶ
- 各ノードはパスにより識別できる
  (1ノードに複数のパスが対応づけられることもある)
  - 絶対パス：ルート(根)を起点とするパス
  - 相対パス：ルート以外を起点とするパス

マウント(mount)

- 複数の名前空間を併合することもできる
- マウント：OSからファイルシステムを使えるようにするための手続き
  - 既存のファイル階層に、他のファイル階層の一部を併合することも可能
  - 併合地点のことをマウントポイント(mount point)と呼ぶ

マウント(mount)

(例) NFS (Network File System):
ネットワークの先にある他のマシンのファイルシステムをマウントできる



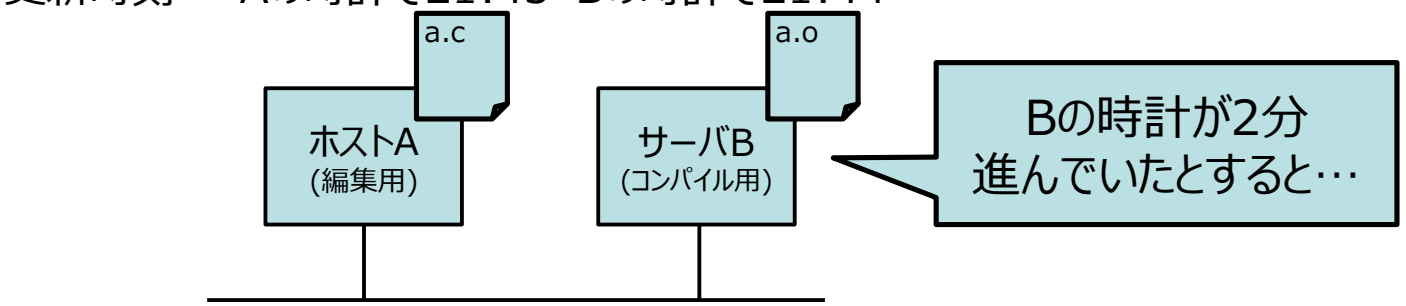
名前空間の実装

- 名前サーバ(name server)：名前付けサービスを提供
  - 分散システムがLANに限定される場合、1つの名前サーバで対応可能
  - 地理的に分散し、複数の組織にまたがる分散システムでは、複数の名前サーバで分散管理
- 名前管理分散の論理的3層
  - グローバル層：
    - ルートノードやその近傍。内容がほぼ不変
  - 部門管理層：
    - 1つの組織の中で管理されるディレクトリノード
  - マネージャ層：
    - 1つのLANに属するホストやファイル。内容変更が頻繁

同期 (synchronization)

- 分散システム内のプロセスをいかに同期させるか
  - 同期：タイミングを合わせること
- 同期の必要性
  - 共有リソースへのアクセス
    - 排他制御
    - 矛盾した書き込み・呼び出しの防止
  - イベント、メッセージの順序の保持

(例) make (タイムスタンプに基づくソースコードのコンパイル)



標準時間

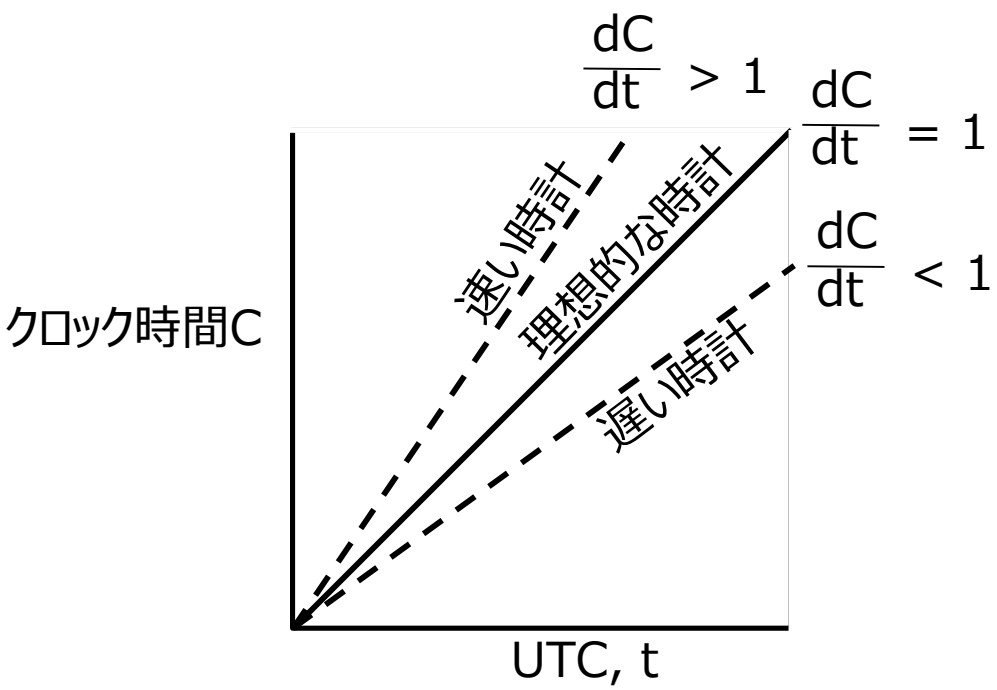
- セシウム時計
  - Cs133を使って発生させたマイクロ波(約91億Hz)を使用 → 1秒の基準とする
- 国際原子時間 (TAI：International Atomic Time)
  - パリの国際時間局で1958年1月1日0時からのセシウム時計のティック数に基づく
  - 平均太陽日(太陽の南中時刻に基づく測定)より約3ミリ秒短い
    → 太陽時間との差が800ミリ秒より大きくなったときに閏秒を導入(1秒ずらす)
  - ⇨ 協定世界時 (UTC: Universal Coordinated Time)

南中：太陽が真南に来ること

クロック同期

- クロック時間C
- 時間(UTC) t
- 最大ドリフト率ρ: クロックスキューがどの程度許容されるか

$$1-\rho \leqq \frac{dC}{dt} \leqq 1+\rho$$

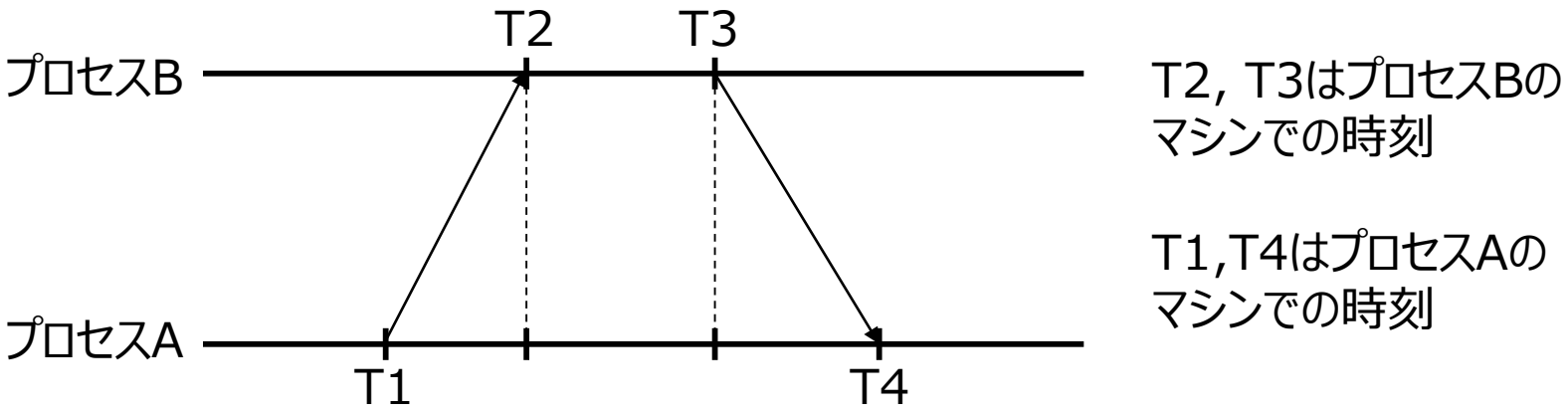


最大ドリフト率ρの
2台のマシンの間では
t秒経過後には
最大2 ρ tだけ時計がずれる

時計のズレをδ以内に
抑えたい場合は、少なくとも
δ/2 ρごとに再同期が必要

Network Time Protocol (NTP)

- 時間サーバ(time server)との交信のための
  プロトコル



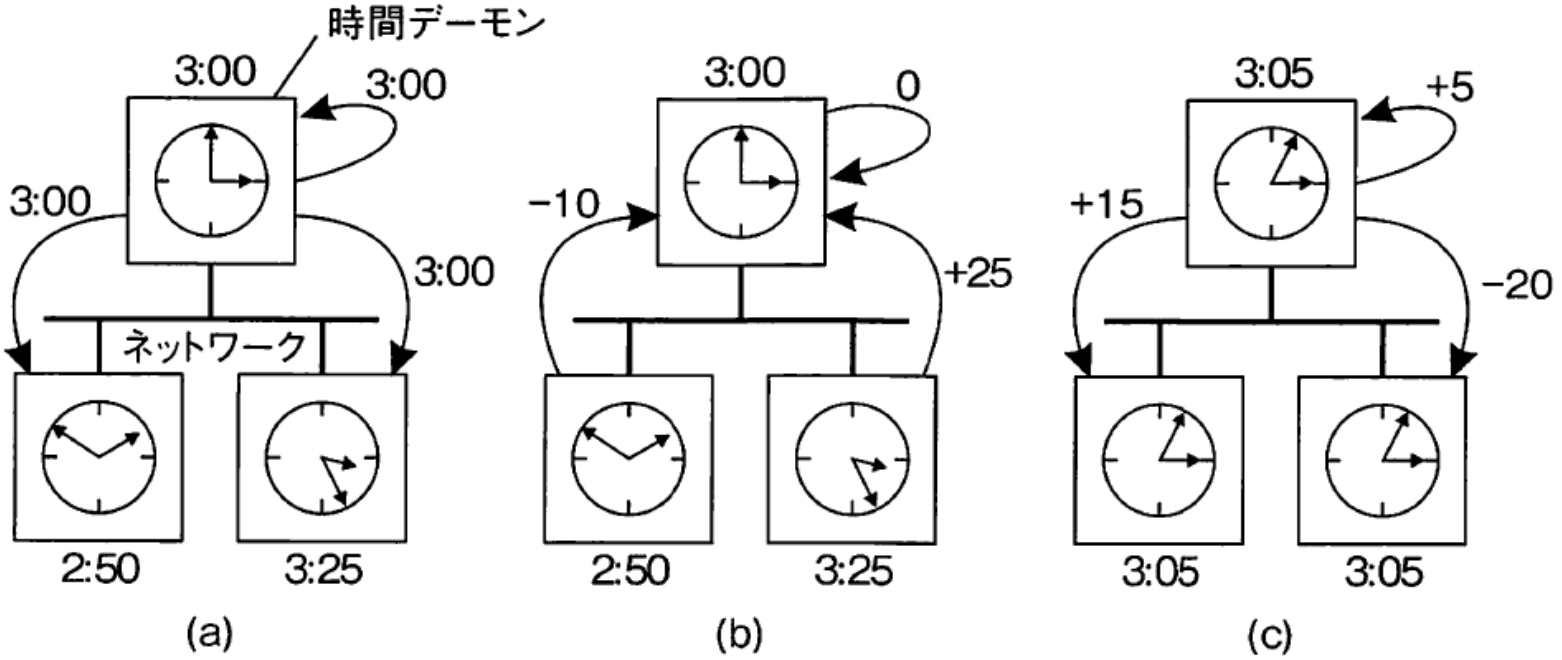
- 伝達遅れ T2-T1 と T4-T3 がほぼ同じと仮定
- Aの時計が遅れていた場合 → Bの時刻に合わせる
- Aの時計が進んでいた場合 → 徐々に時刻調整

Network Time Protocol (NTP)

- NTPのサーバ階層
  - 標準時計(reference clock)を持つサーバ
    ＝ 階層-1サーバ
  - 階層-kサーバ の時刻をもらうサーバ
    ＝ 階層-(k+1)サーバ
    (階層が高いサーバが相手サーバに合わせる)
  - 同じ階層のサーバ同士は相互に時刻を
    問い合わせる (AがBに問い合わせると同時に
    BもAに問い合わせ)

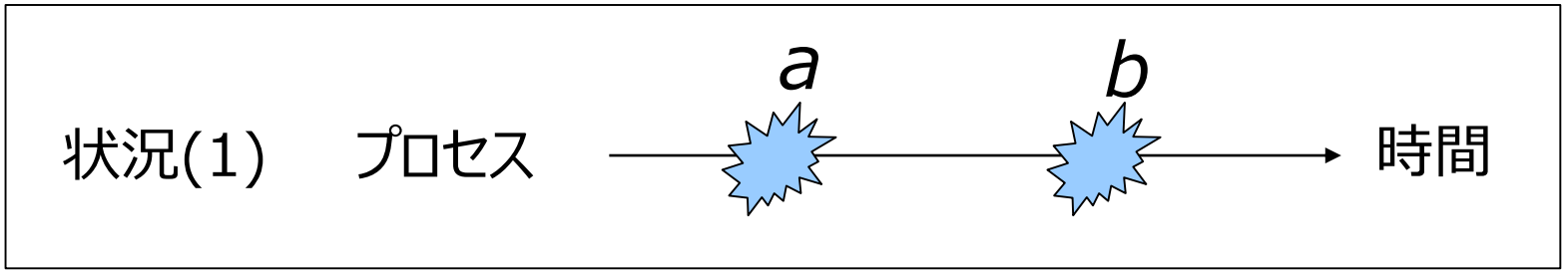
Berkeleyアルゴリズム

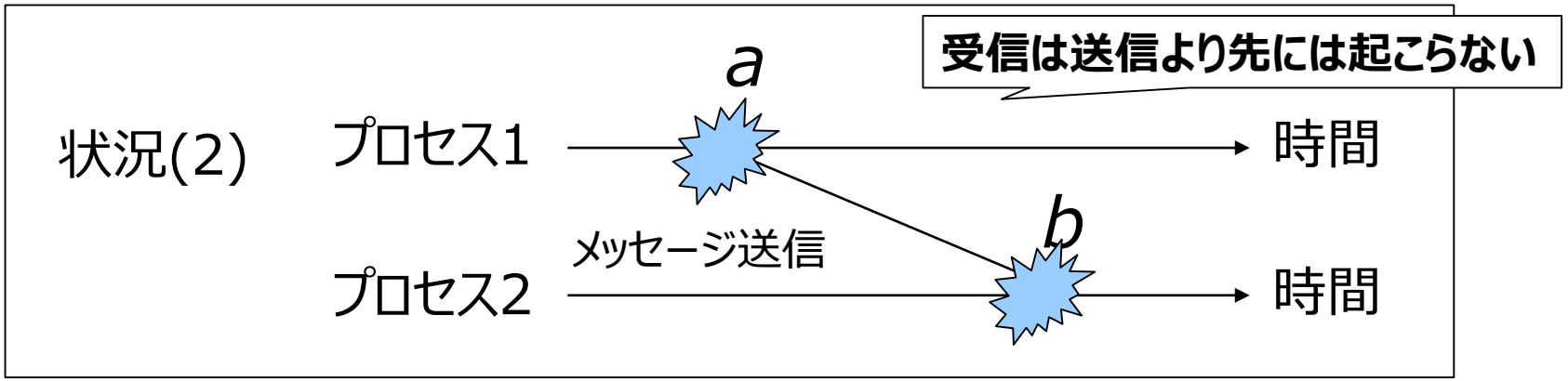
- 時間デーモン(time daemon)が能動的に各コンピュータに問い合わせ、平均時刻を計算



Lamportの論理クロック

- 事前発生(happens-before)の関係
  - 「まずaが発生し、次にbが発生する」ということにすべてのプロセスが同意
  - a→b と表記



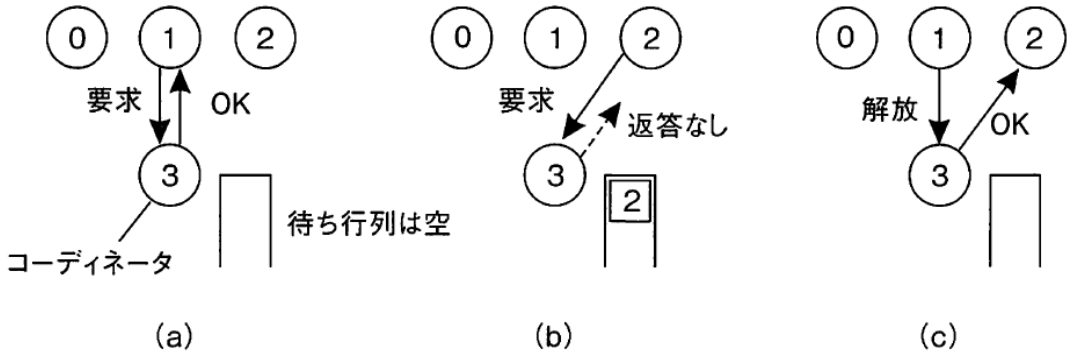


排他制御

- 排他制御は分散システムにおいて生来的な問題
  - 複数プロセスの並行と協調
  - 同じ資源への同時アクセス
- 排他制御の際に避けるべきこと
  - 飢餓(starvation)
    - プロセスが必要なリソースにアクセスできない状態
  - デッドロック(dead lock)
    - 複数のプロセスが互いの終了を待ち、
      いずれも先に進めなくなる状態
- 排他制御の方式
  - トークンベース(token-based)方式
    - トークンを取得したプロセスがリソースを利用可能
  - 許可ベース(permission-based)方式
    - 許可を得たプロセスがリソースを利用可能

集中アルゴリズム

- 一つのプロセスをコーディネータとして専任
- リソースにアクセスするときはコーディネータの許可を得る
- コーディネータが待ち行列を管理
  - リソース使用中なら待ち行列に入れる



長所
- 先着順→公平
- どのプロセスも永遠に待つことはない
- 実装が容易

短所
- 故障単一箇所性 (Single point of failure)
  コーディネータがダウンすると全体が止まる

複製

- 分散システム上のデータを複製(replication)
- 複製の必要性
  - 信頼性向上
    - 同じデータを複製することで、一つのレプリカ(複製したもの)が使用不可能になっても、他のレプリカに切り替えることができる
  - 性能向上
    - データを近くに配置することで、アクセス時間を短縮
      → システムの規模拡大にもつながる
- 複製の代償
  - 一貫性(consistency)の保持
    - 大規模ネットワークでは一貫性保持のコストが大きい
    - 現実的には一貫性を諦めざるを得ないこともある

データ中心一貫性モデル
(data-centric consistency model)

- プロセスが特定の条件を満たす場合、
  データストアが正常に動作することを保証
  - データストア(data store)：データの保管場所。
    分散している可能性あり

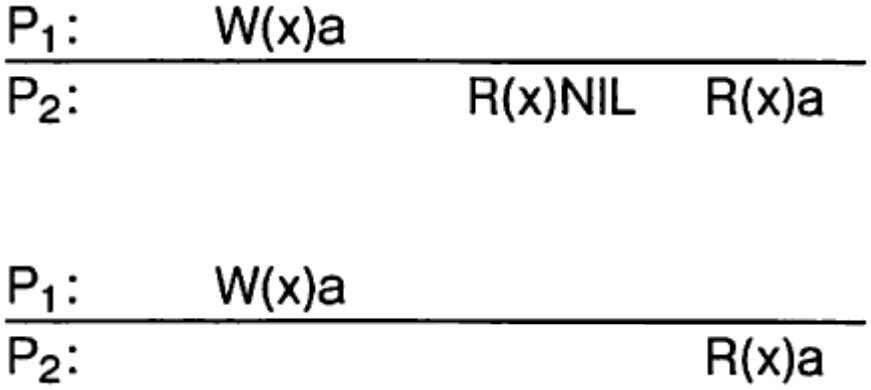
- 「正常」とは？
  - 完璧に一貫性を保証することは困難
    → どのように不一貫性を許容可能かはアプリケーションに依存
  - 連続的一貫性の3つの判断基準：
    - 数値の相違 (例：値の相違が0.5以内、値の相違が1%以内)
    - 陳腐化度の相違
      (例：最新版との相違が1時間以内、更新1回以内)
    - 更新操作の順序の相違

データ中心一貫性モデル

- 操作の順序に関する一貫性
  - 厳密一貫性：
    すべての操作を絶対時間により順序づける
  - 順序一貫性：
    すべてのプロセスによるデータストアへの
    すべての読み取り、書き込みの操作の結果が、
    それらの操作が一定の順序で実行された結果と同じになる
  - 因果一貫性：
    前の操作の影響を受けている可能性のある書き込みは、
    すべてのプロセスによって、同じ順序で
    観測されなければならない

厳密一貫性

- すべての操作を絶対時間により順序づける
  - 書き込み後には、その書き込みが反映されていなければならない
    → 実システムでは遅延があるため実現不可能



厳密一貫性がない

プロセスP1によってxにaが書き込まれた後、
プロセスP2がxを読み込んだらNIL(値未設定)を得た。
その後プロセスP2がxを読み込んだらaを得た。

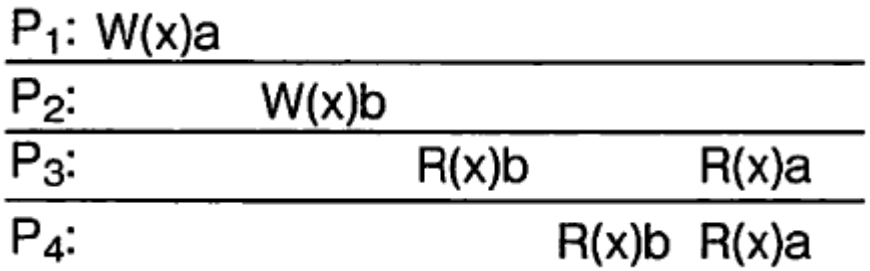
厳密一貫性がある

プロセスP1によってxにaが書き込まれた後、
プロセスP2がxを読み込んだらaを得た。

※記号の意味
W(x)a 「xにaを書き込む」
R(x)a 「xを読み込んだ結果aを得る」
Piはプロセス。横軸は時間。

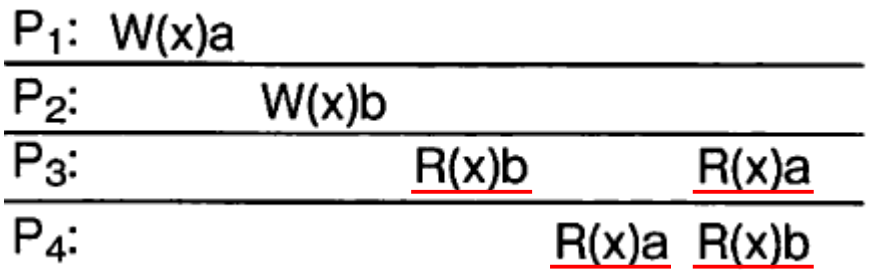
順序一貫性

- すべてのプロセスによるデータストアへのすべての読み取り、書き込みの操作の結果が、それらの操作が一定の順序で実行された結果と同じになる



順序一貫性がある

※実際の書き込み順と逆になっていてもOK



順序一貫性がない

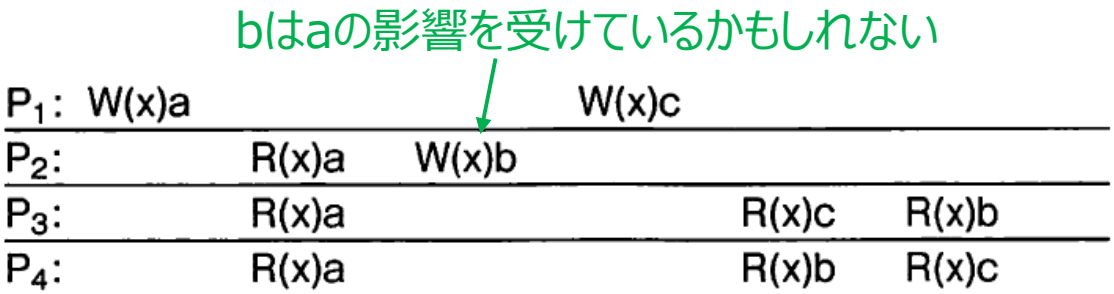
←先にbが書き込まれたことになっている

←先にaが書き込まれたことになっている

※記号の意味
W(x)a 「xにaを書き込む」
R(x)a 「xを読み込んだ結果aを得る」
Piはプロセス。横軸は時間。

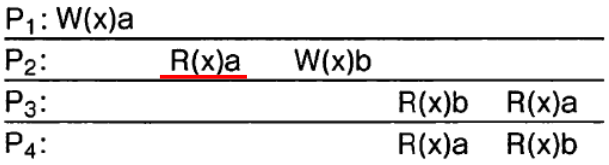
因果一貫性

- 前の操作の影響を受けている可能性のある書き込みは、すべてのプロセスによって、同じ順序で観測されなければならない
  - 前の操作の影響を受けている可能性がある
    ＝ 因果的に関連する

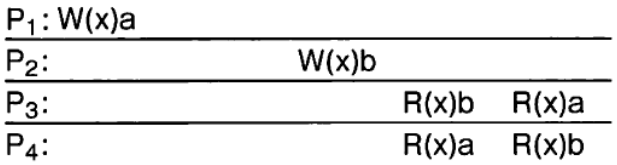


因果一貫性があるが、
順序一貫性はない

※bとcは因果的に関連がない



因果一貫性がない

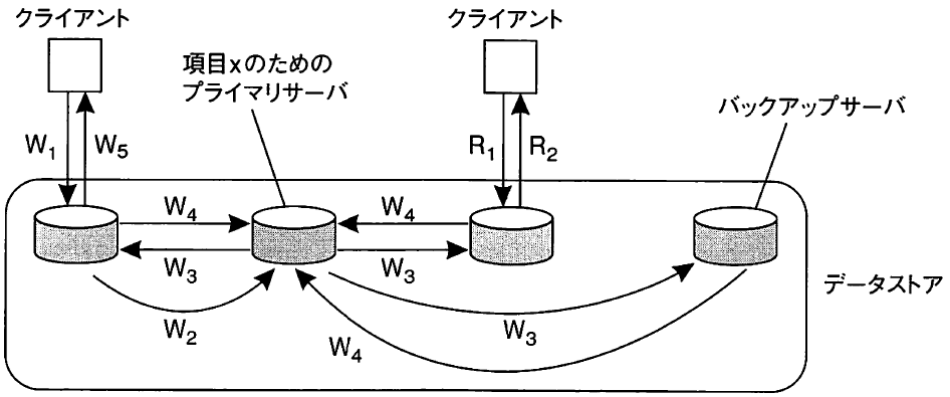


因果一貫性がある

※記号の意味
W(x)a 「xにaを書き込む」
R(x)a 「xを読み込んだ結果aを得る」
Piはプロセス。横軸は時間。

遠隔書き込みプロトコル

- すべての書き込み操作を
  一つのサーバ(プライマリサーバ)に送る
- サーバは複製を更新後、書き込みを要求した
  プロセスに通知
- 長所：順序一貫性が保証される
- 短所：書き込み後の処理再開が遅くなる

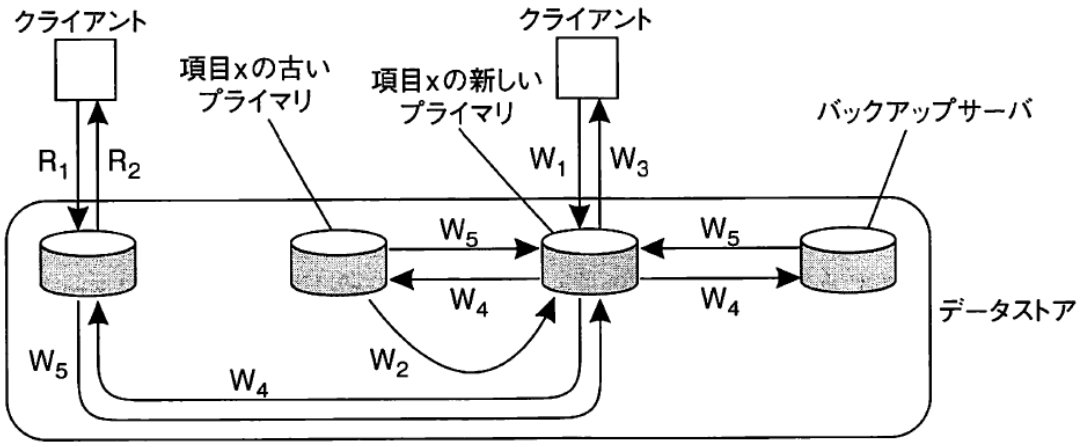


W_1. 書き込み要求
W_2. プライマリへ要求を転送する
W_3. バックアップに更新を告げる
W_4. 更新の確認通知
W_5. 書き込み完了の確認通知

R_1. 読み取り要求
R_2. 読み取りへの応答

ローカル書き込みプロトコル

- 書き込み要求が発生すると、プライマリを
  ローカルサーバに移動して書き込みを行う



W1. 書き込み要求
W2. 項目xを新しいプライマリに移動する
W3. 書き込み完了の確認通知を行う
W4. バックアップに更新を告げる
W5. 更新の確認通知を行う

R1. 読み取り要求
R2. 読み取りへの応答

- 複数の後続の読み書きがローカルで行える
- 非接続モードでローカルのみ更新しておく場合に利用可能

フォールトトレラント性

- [JIS-X0014:1999]によるフォールトトレランス(fault tolerance)の定義
  - 障害または誤りが存在しても、要求された機能を遂行し続けることのできる、機能単位の能力
  - フォールトトレランスを達成するための性質
    → フォールトトレラント性、耐故障性
- 分散システムでは部分的に障害が発生する
  - 単一システムの障害は全体をダウンさせることが多い
  - いかにシステムを動かしたまま障害から回復させるか(→障害透過性)
- 故障に強いシステム構築のために
  - プロセスの回復力(resilience) - プロセスの多重化
  - 高信頼クライアントサーバ間通信
  - 分散コミット
  - エラーからの回復

フォールトトレラント性

- フォールトトレラント性を支える性質
  - 可用性(availability)
    - 利用しようとした瞬間に利用できるか
  - 信頼性(reliability)
    - 障害を起こすことなく実行し続けられるか
  - 安全性(safety)
    - システムが正常に稼働しなくても重大な問題が生じないか
      - 原発やロケットの制御システム等では特に高い安全性が必要
  - 保守性(maintainability)
    - 故障からの回復が容易か
      - 可用性にも影響

フォールトトレラント性

- 可用性と信頼性の指標
  - MTTR(Mean Time To Recovery)
    - 平均修復時間(修復に要する時間の平均)
    - MTTRが小さいほど可用性が高い
  - MTBF(Mean Time Between Failures)
    - 平均故障間隔
    - MTBFが大きいほど信頼性が高い
- (例1) 1時間に1回1ミリ秒ダウン
  - 可用性: 1-(0.001/3600)=99.9999%以上
  - 信頼性(MTBF): 約1時間
- (例2) 1年のうち1回2週間ダウン
  - 可用性: 1-(14/365)=約96%
  - 信頼性(MTBF): 351日

MTTR=故障時間/故障回数

MTBF=稼働時間/故障回数

可用性(%)=稼働時間/総時間

障害(fault)

障害の種類	**説明**
クラッシュ障害	サーバの停止。直前まで正常に動作
欠落障害 　受信欠落 　送信欠落	サーバが要求の応答に失敗 サーバがメッセージの受信に失敗 サーバがメッセージの送信に失敗
タイミング障害	サーバが指定時間内に応答できない
応答障害 　値障害 　状態遷移障害	サーバの応答が不正確 応答の値が不正確 サーバが正しい制御の流れから逸脱
任意障害 (ビザンチン障害)	サーバが任意の時間に任意の応答を生成

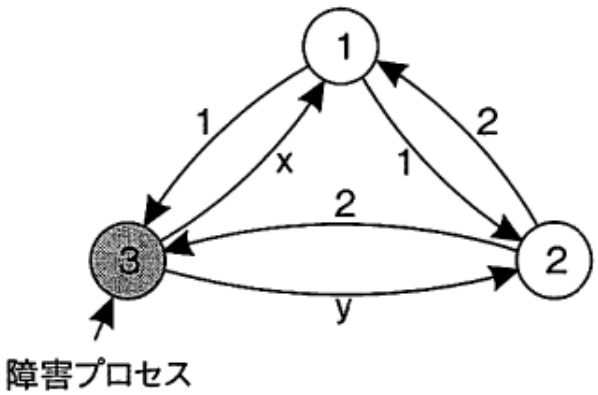
障害システムにおける合意
- ビザンチン将軍問題 -

- 前提
  - 通信は完全であるが，プロセスは完全でない
  - 同盟軍はn人の将軍が軍を率いており、
    このうち、m人は裏切り者である
  - 将軍は自分の軍の人数を知っており、
    他の軍の将軍に伝えることができる
  - 裏切り者の将軍は嘘を言うかも知れない

→裏切り者以外の将軍は他の軍の人数を合意できるか？

障害システムにおける合意
- ビザンチン将軍問題 -

- m個の障害プロセスを持つシステムにおいて、
  合意には2m+1個の正しく機能するプロセスが必要
  （Lamport, 1982）



STEP2
プロセス1は(1,2,x)を獲得
プロセス2は(1,2,y)を獲得
プロセス3は(1,2,3)を獲得

STEP3
プロセス1　プロセス2
(1,2,y)　(1,2,x)
(a,b,c)　(d,e,f)

合意できない

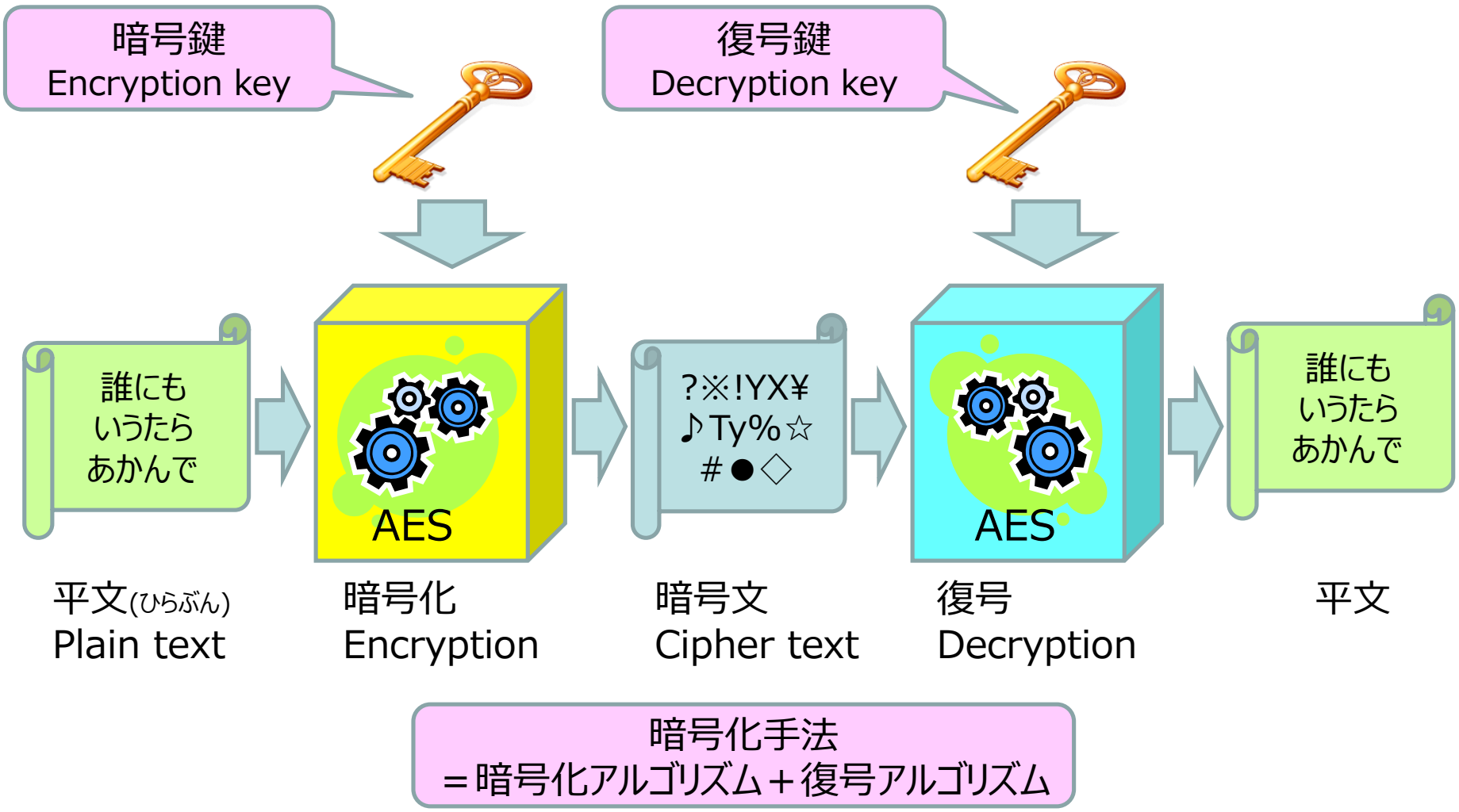
情報セキュリティ

- セキュリティ特性 (セキュリティ3要素)
  - 機密性(Confidentiality)
    - システムが認可されたユーザに対してのみ情報を開示する
  - 完全性(Integrity)
    - システムの資産への改変が認可された方法でのみ行われる
    - 不当な改変は検知・修正する必要がある
  - 可用性(Availability)
    - 利用したいときに利用できるか
      (cf. フォールトトレラント性)
      → e.g., プロセス多重化、安定ストレージ

セキュリティ脅威

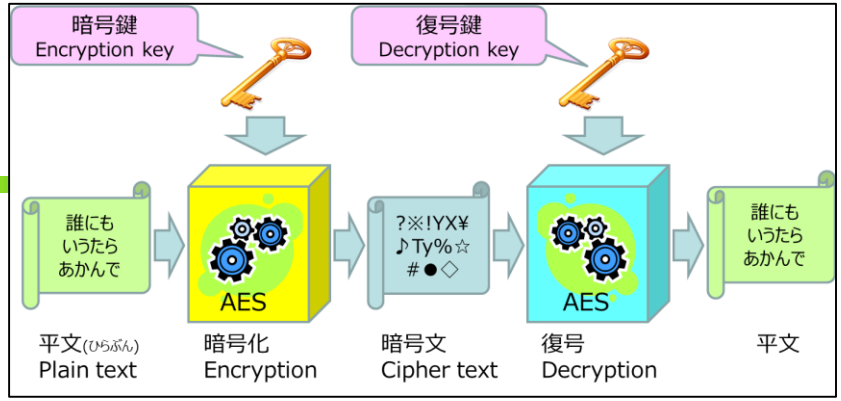
- 横取り(interception)：不正なデータアクセスが可能な状態
  - (例) ファイルシステムの個人ディレクトリが破られる
- 中断(interruption)：サービスやデータが利用不可能や無効になったり、破壊されたりする状態
  - (例) DoS攻撃(Denial of service)
- 改変(modification)：データの認められない変更、サービスの改竄
  - (例) データベース項目の改竄
- 合成(fabrication)：通常は存在しない付加的なデータやアクティビティが作られる状態
  - (例) 不正なアカウントの追加、不正なデータの追加

暗号化



暗号化手法
＝暗号化アルゴリズム＋復号アルゴリズム

暗号化



- 共通鍵暗号
  - 暗号鍵と復号鍵が同じ
  - 安全なルートで鍵を共有しなければならない
  - DES, 3DES, AES等のアルゴリズム
- 公開鍵暗号
  - 暗号鍵は公開されている。復号鍵は非公開(秘密)
    (暗号鍵と復号鍵は異なる)
  - 公開鍵から秘密鍵を推測するのは不可能 or
    莫大な計算量を要する
  - 公開鍵で生成された暗号文は、対になる
    秘密鍵でしか復号できない
  - 暗号文から公開鍵や秘密鍵が推測できない
  - RSA, DSA等のアルゴリズム

認証と署名

- 認証：適切なユーザであることを保証すること
- 送信元・送信先の正当性(＝認証) と
  メッセージの完全性(＝署名) 両方が必要
  - AがBにmを送るとき、
    送信元は確かにAか？
    送信先は確かにBか？
    メッセージは本当にmか？

メッセージの機密性・完全性

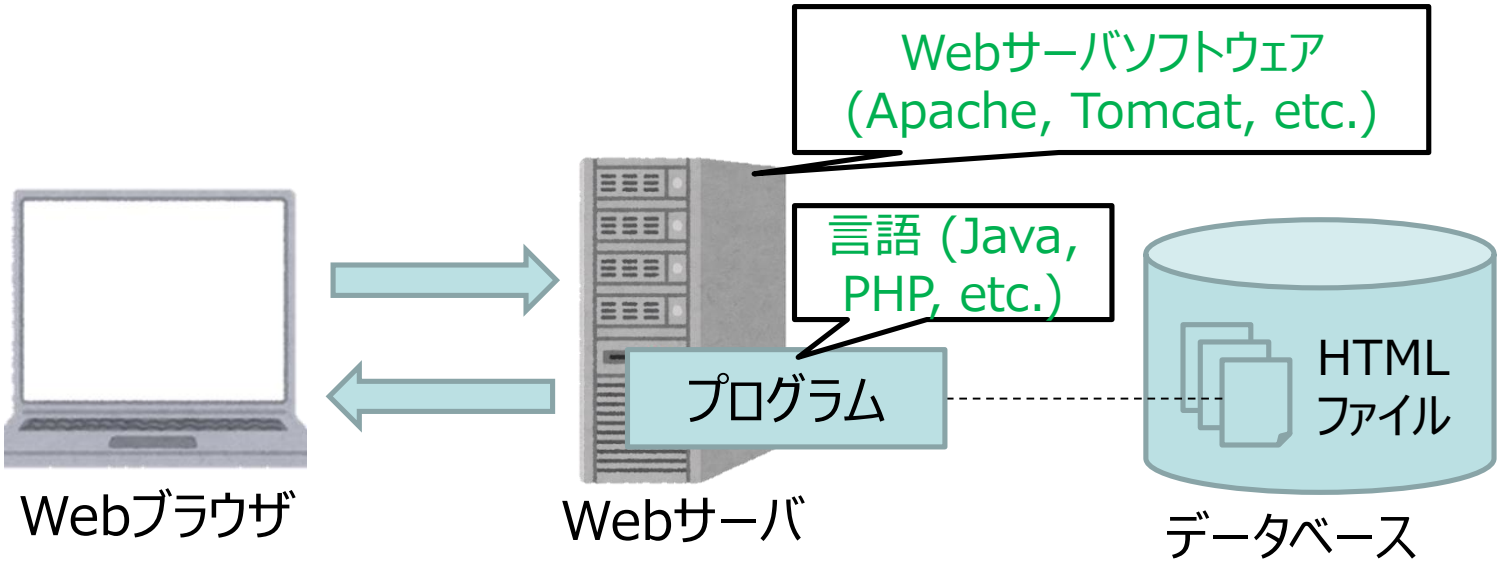
(再掲)

- 機密性(Confidentiality)
  - システムが情報を認可されたユーザに対してのみ開示する
- 完全性(Integrity)
  - システムの資産への改変が認可された方法でのみ行われる
    → 不当な改変は検知・修正する必要がある

- 暗号化すれば機密性は保たれる
- しかし、暗号化したものを改竄される可能性はある

- 電子署名
  - 断りなく内容を変更することを防止

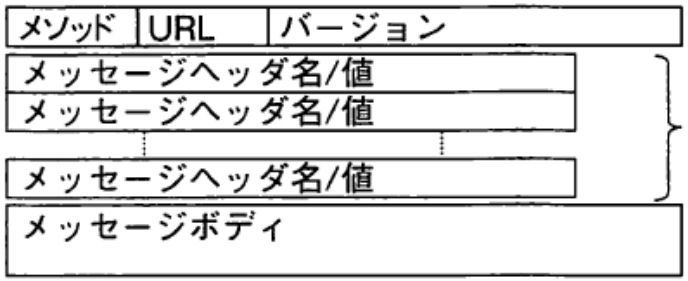
Webアプリケーション



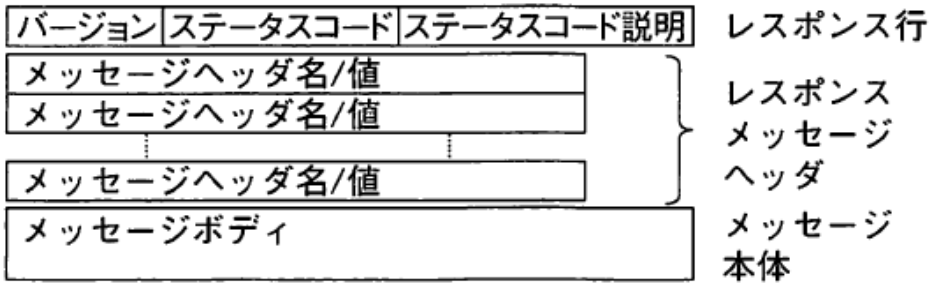
- 静的なWebページ：常に同じ内容
- 動的なWebページ：その都度異なる内容
  - プログラム(Webアプリケーション)により生成
  - サーバ側、ブラウザ側両方で生成可

HTTP

- WebブラウザとWebサーバの間のメッセージ交換プロトコル
  - 様々なデータを扱う可能性
  - 用途ごとに基本的なメソッドが存在
- 基本的なメソッド
  - head: ドキュメントのヘッダ(日付やサイズ)を要求
  - get: サーバからドキュメントを取得
  - put: サーバにドキュメントを送信 (ファイル送信等)
  - post: サーバにドキュメントを送信 (パスワード送信等の使用が多い)
  - delete: サーバ上のドキュメント削除を要求



HTTP要求メッセージのフォーマット



HTTP応答メッセージのフォーマット

メッセージヘッダの例：
If-Modified-Since(指定の日付以降の情報を要求), If-None-Match(ドキュメントの内容が一致しない場合のみ実行)

パスワードクラッキング

- ユーザのパスワードを不正に取得しようとする試み
- 代表的な手法
  - 辞書攻撃：よく使われる単語を単体あるいは組み合わせて、次々にログインを試す
  - ブルートフォースアタック：パスワードとして可能な文字の組み合わせをすべて試す
- 対策
  - 簡単なパスワードを設定できないようにしておく

DoS攻撃(Denial of Service)

- 短時間にサーバが処理しきれないような大量のアクセスをすることでサービス停止に陥らせる
- 代表的な手法
  - SYN Flood：TCP SYNパケットを大量に送りつける
  - F5攻撃：Webページの再読み込みを何度も繰り返す
- 対策
  - 不自然なアクセスを早期に検知し、当該IPからのアクセスを遮断する

セッションハイジャック

- Cookieの中身やセッションIDを取得し、正規のセッションを乗っ取る
- 代表的な手法
  - 盗聴
  - Webアプリケーションの脆弱性を突く
- 対策
  - 通信の暗号化
  - 急に異なるIPアドレスからアクセスした場合に強制ログアウトさせる

クロスサイトスクリプティング(XSS)

- 実際には、攻撃対象者を罠サイトに誘導
  → 取得したCookie値を利用して正規サイトへのセッションを乗っ取る

- 対策
  - メタ文字(HTMLの文法上特別な意味を持つ記号)をエスケープ(escape)する
    - "<"を"<"に置換する等

```
keyword=&lt;script&gt;alert(document.cookie)&lt;/script&gt;
```

タグだとみなされないので無害

SQLインジェクション

injection[N]投入、注入

- XSSと同様にサーバにコードを送る。送る物が「SQL文」
  → データベース内の情報を不正に取得

```
$sql = "SELECT * FROM books WHERE author='$author'";
```

クエリ文字列が ' で開始

```
http://example.jp/ex.php?author='+AND+EXTRACTVALUE(0,(SELECT+
CONCAT('$',id,':',pwd)+FROM+users+LIMIT+0,1)+%23
```

表users 0行目ユーザのidとpwdを取得

- 対策
  - データベースの権限設定
  - SQL文の文字列リテラルの扱いに注意

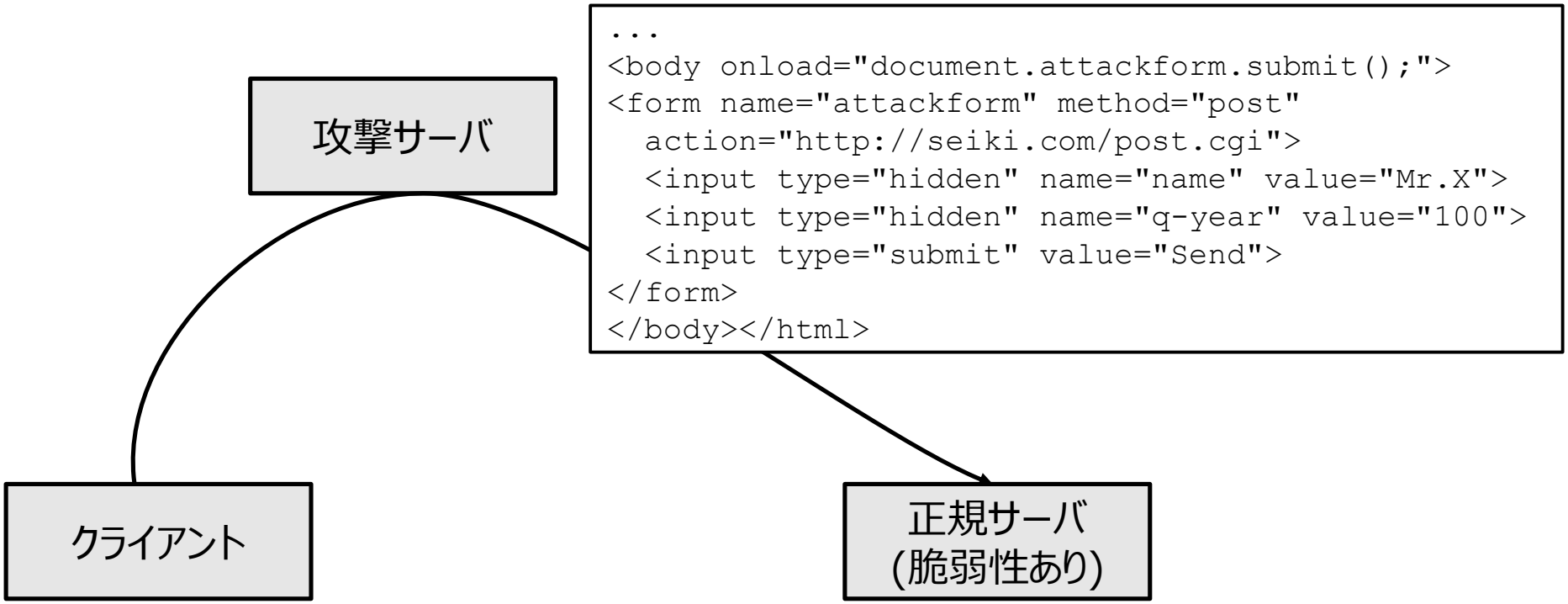
```
$sql = "SELECT * FROM books WHERE author=?";
...
(bindValueメソッドによる ? と $author の対応付け)
```

プレースホルダ (?)を利用

クロスサイトリクエストフォージェリ (CSRF)

forgery [N] 偽造

- 意図しないHTTPリクエストの偽造



- 対策
  - 正規サーバにおけるセッション確認等