ソフトウェアアーキテクチャ まとめ

環境情報学部
萩野　達也

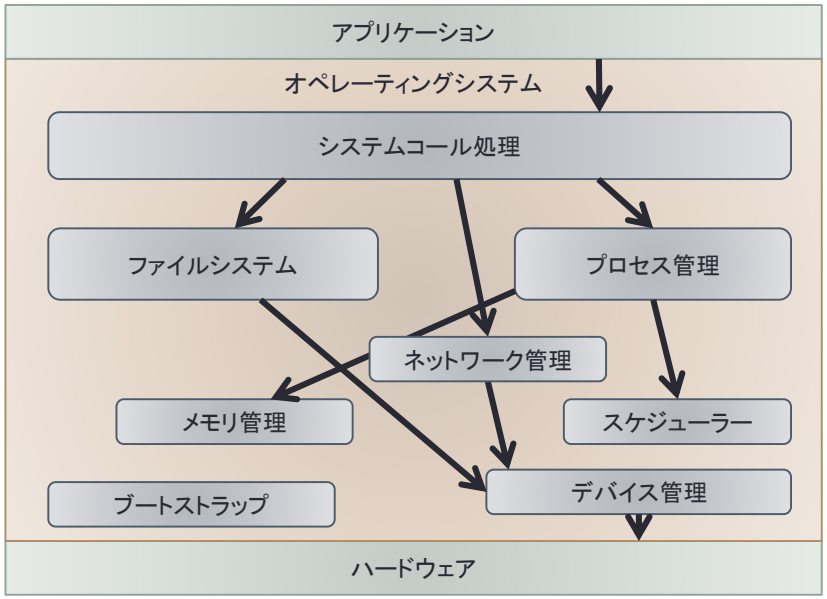
lecture URL

https://vu5.sfc.keio.ac.jp/slide/

第1回　オペレーティングシステム

- オペレーティングシステムとは
  - 基本ソフトウェア
- よく使われているオペレーティングシステム
  - Windows
  - Mac OS X
  - Unix, Linux
- オペレーティングシステムの役割
  - ハードウェアの制御調停
  - プログラムの分離
  - マルチプログラミング
  - メモリ管理
  - ファイルシステムの提供
  - ネットワークシステムの提供
  - プログラム間の通信の提供

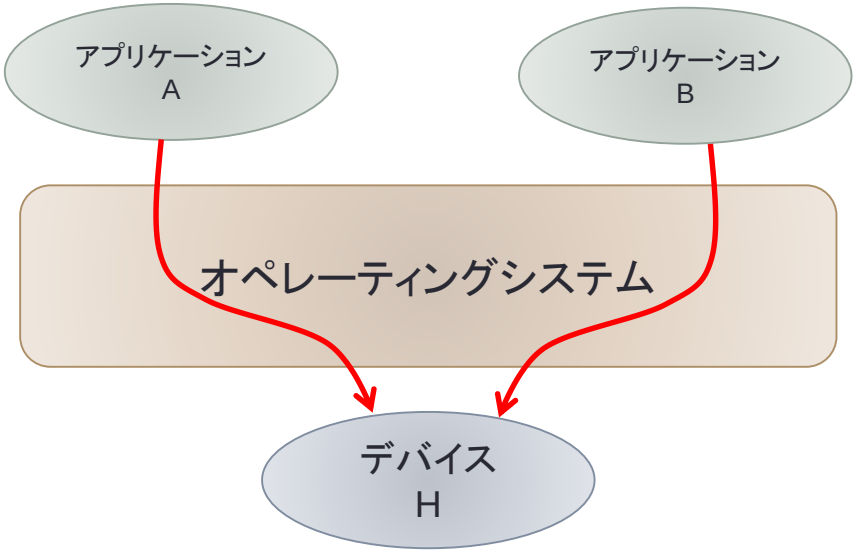
オペレーティングシステムの構成要素



デバイス制御の調停

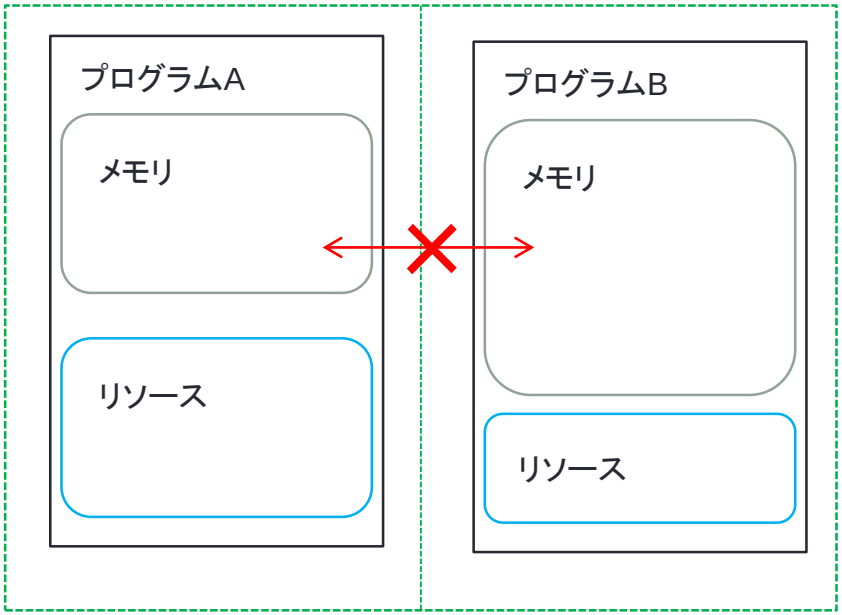
- 2つのアプリケーションが同時に同じデバイスを利用してはいけない
  - デバイス＝コンピュータに接続しているハードウェア
  - キーボードは1つ，マウスも1つ，プリンタも1台，など

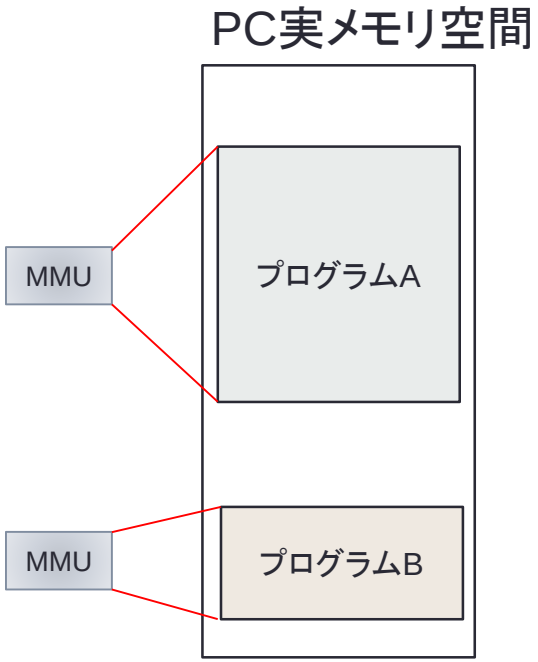
- OSによる調停
  - アプリケーションは直接デバイスを利用しない
  - OSを介してデバイスを利用する



プログラム実行環境の分離

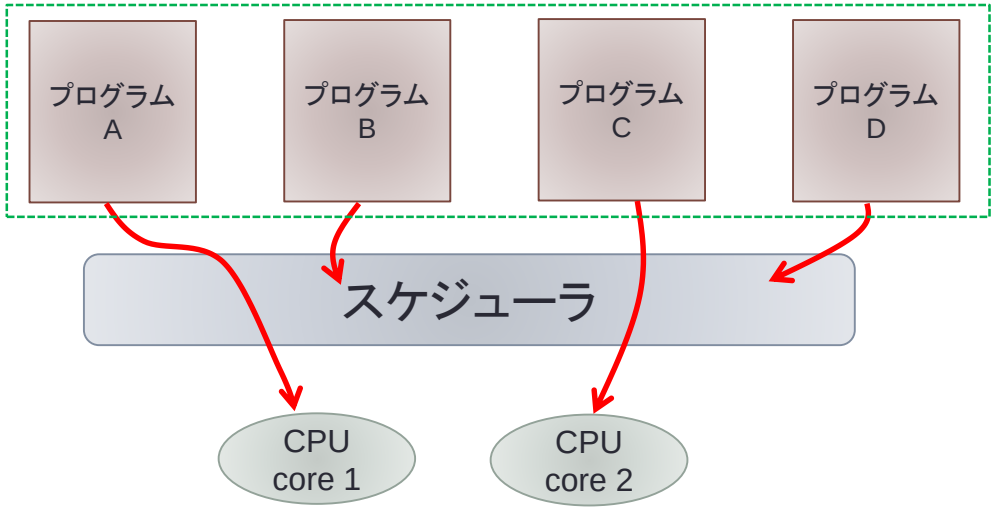
- プログラムは独立して動作している.
  - 他のプログラムの動作には影響されない.
- それぞれのプログラムは，別々のメモリ空間を使う.
- それぞれのプログラムのメモリ空間は保護されている.
  - 他のプログラムから参照や変更ができない.



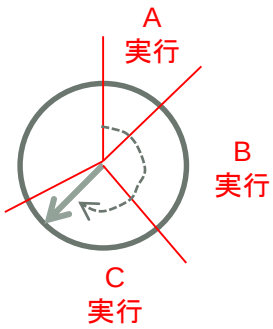


マルチプログラミング

- 複数のプログラムを同時に実行することができる.
- 同時に実行することができるプログラムの数は，CPUのcore数に制限されない.
- それぞれのプログラムにCPUが割り当てられる.
  - スケジューリング
  - 優先制御

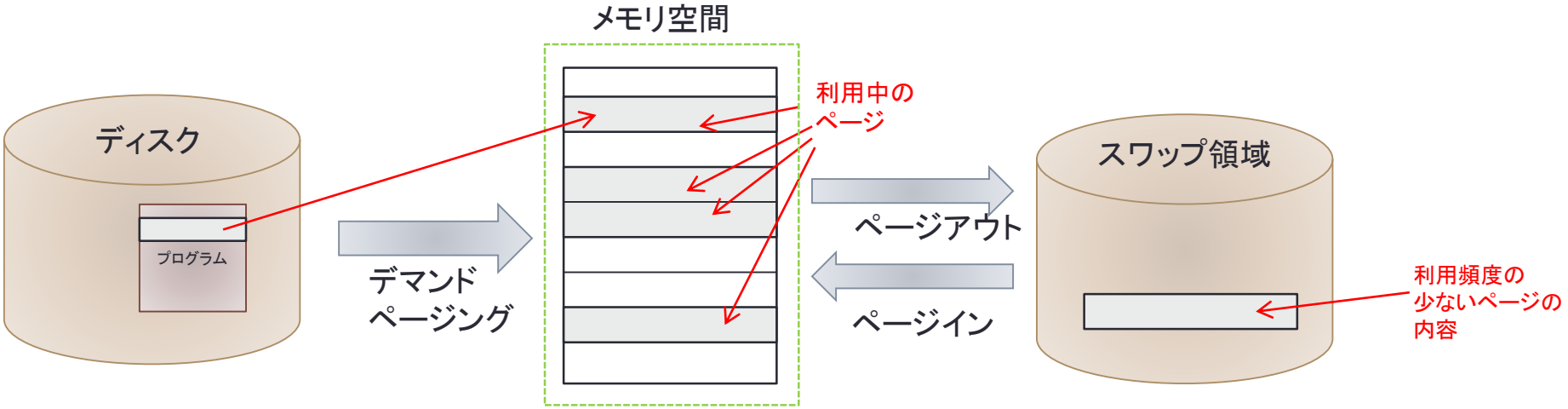


時分割



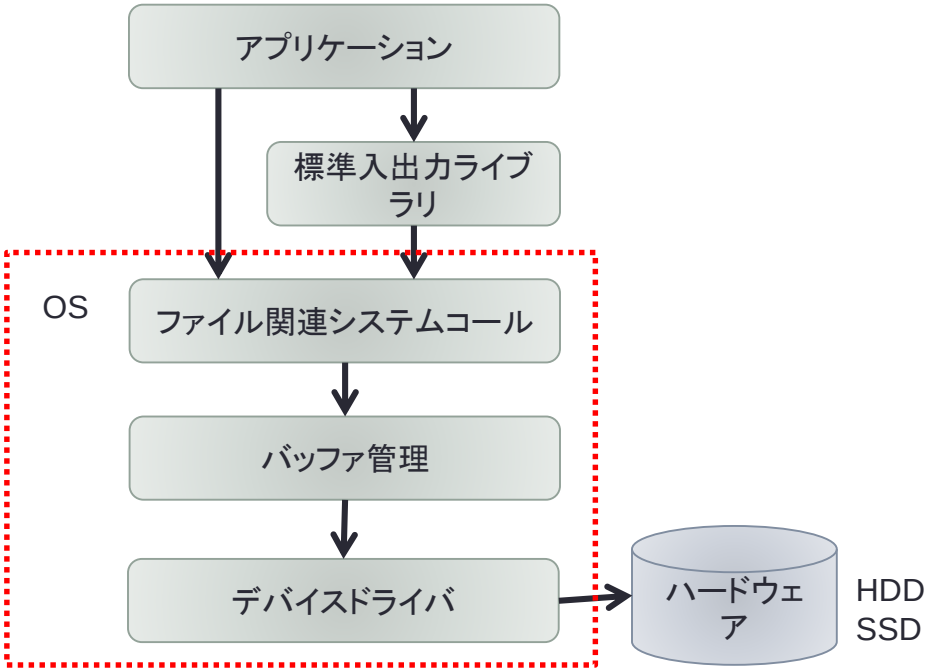
メモリ管理

- プログラムが必要とするメモリを管理
  - メモリはページ単位（例：1ページ＝4KB）に分けられている．
- 不要なメモリを回収し，必要とするプログラムに割り当てる．
- 仮想記憶の提供
  - ページアウト＝利用頻度の少ないメモリ（ページ）は外部記憶（スワップ領域）に追い出し，空きメモリを増やす．
  - ページイン＝必要となった時に，データをメモリ内に読み込みなおす．
  - 実際のメモリの量をプログラムは気にする必要はない．
    - 仮想記憶ではメモリ管理はOSにお任せ
    - 小型ゲーム機なのでは，プログラムオーバーレイを使い，自分でメモリを管理する．
- デマンドページング
  - プログラムやデータは必要となって初めてメモリ内に読み込まれる．
  - プログラム実行開始時はメモリ内は空の状態．

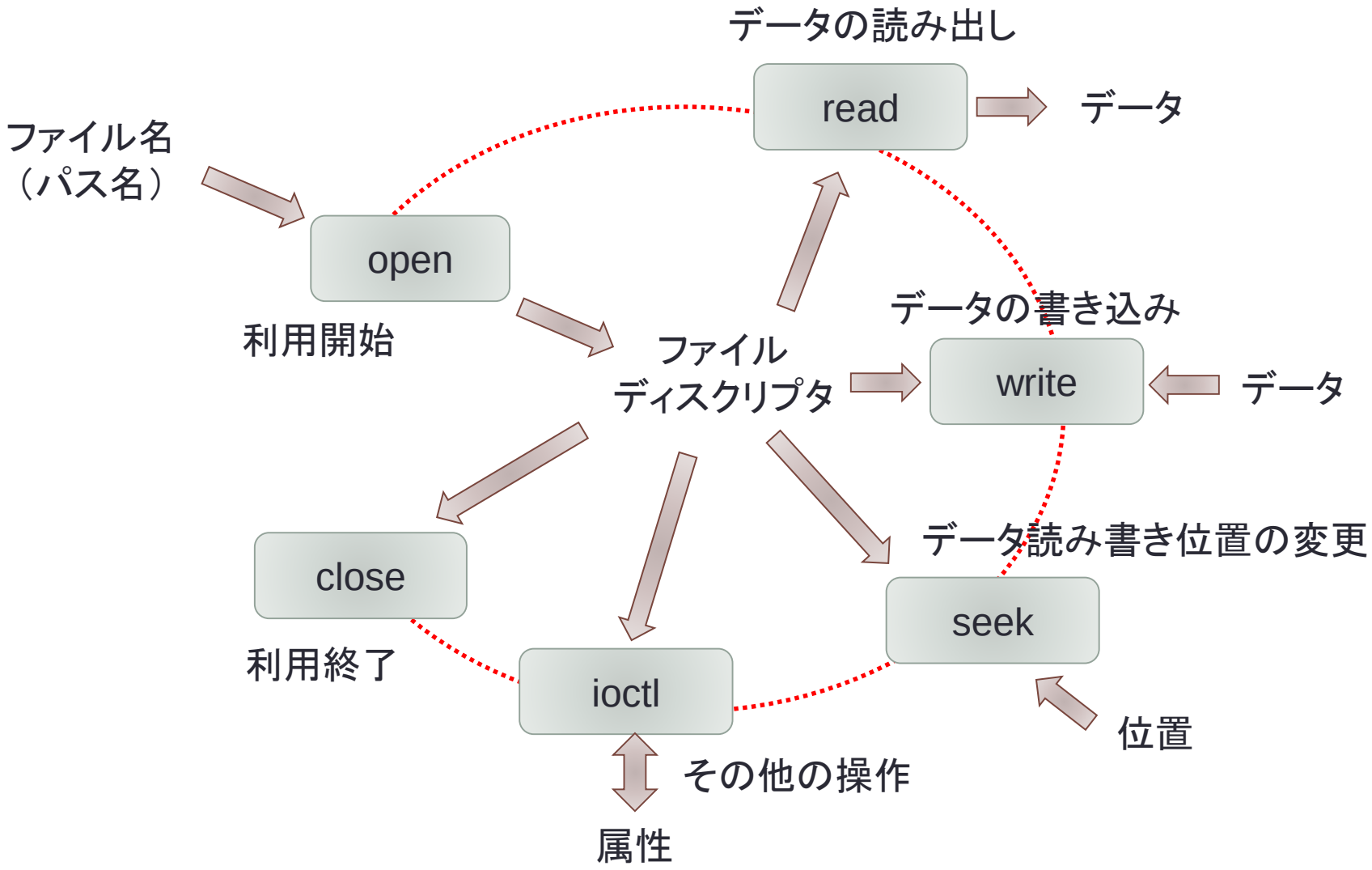


第2回　ファイルシステム

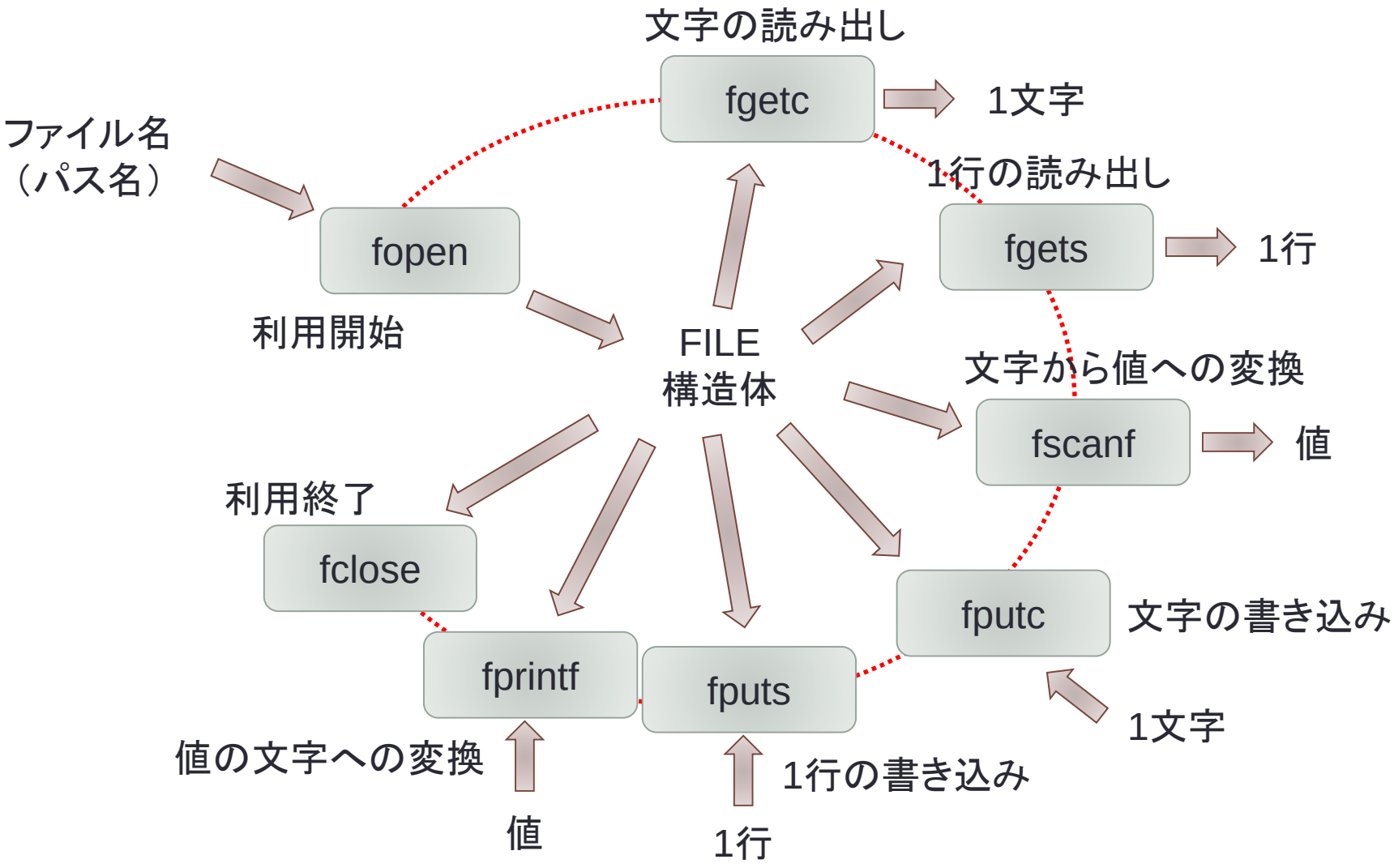
- ファイルとは
  - ファイル名
  - ファイル構造
  - ファイルの種類
  - アクセス方法
- 階層型ファイルシステム
  - パス名（絶対・相対）
- ファイルの読み書き
  - システムコール
    - ファイルディスクリプタ
  - 標準入出力ライブラリ
- ファイルシステムの実装
  - inode



ファイル関連のシステムコール



標準入出力ライブラリの使い方



ファイルの実装方法

- ファイルはディスク上の複数のブロックから構成
  - ファイル毎に利用しているブロック番号を記録する必要がある
- UNIX では inode（index node）により管理
  - FATファイルシステムではFATが管理

inode structure

```
struct inode {
  short mode;
  int uid
  int gid;
  long size;
  ...
  int atime;
  int mtime;
  int ctime;
  ...
  int direct[12];
  int indirect[3];
};
```

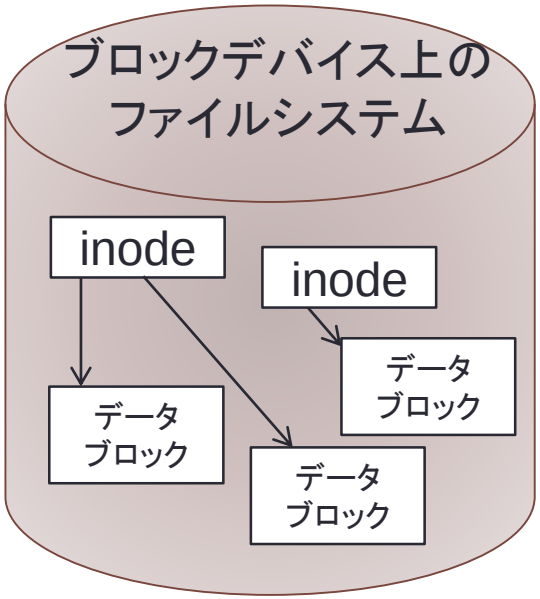
所有者

サイズ

変更日付

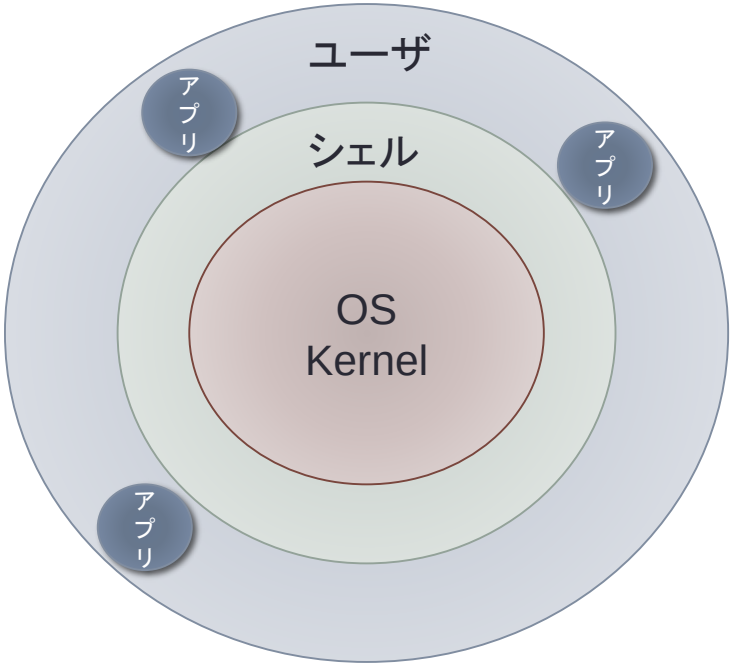
直接参照ブロックの番号（12個）

間接参照ブロック番号（3個）



第3回　シェル

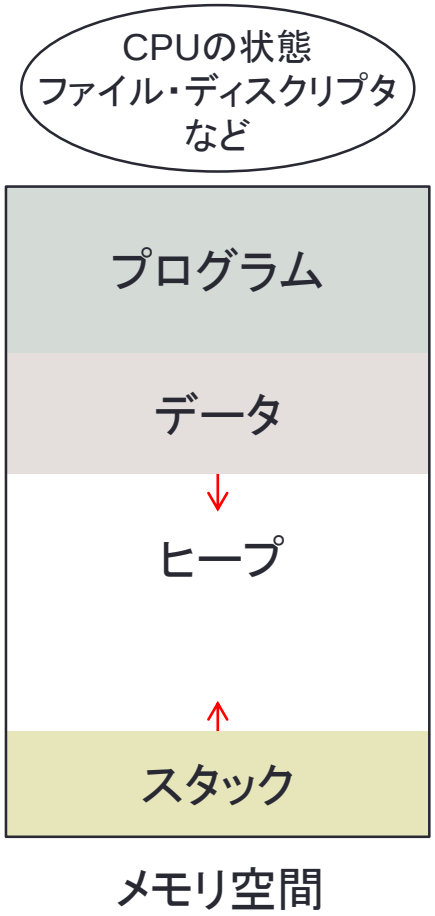
- シェルの機能
  - プログラムの起動・制御
  - 実行環境の設定
  - 便利な機能
  - スクリプトの実行
- プロセス関係のシステムコール
  - fork, exec, wait, exit
- シェルの処理
  - フォアグラウンドとバックグラウンド
  - ジョブ制御
  - リダイレクション
  - シェル変数・環境変数
  - ワイルドカード



プロセス

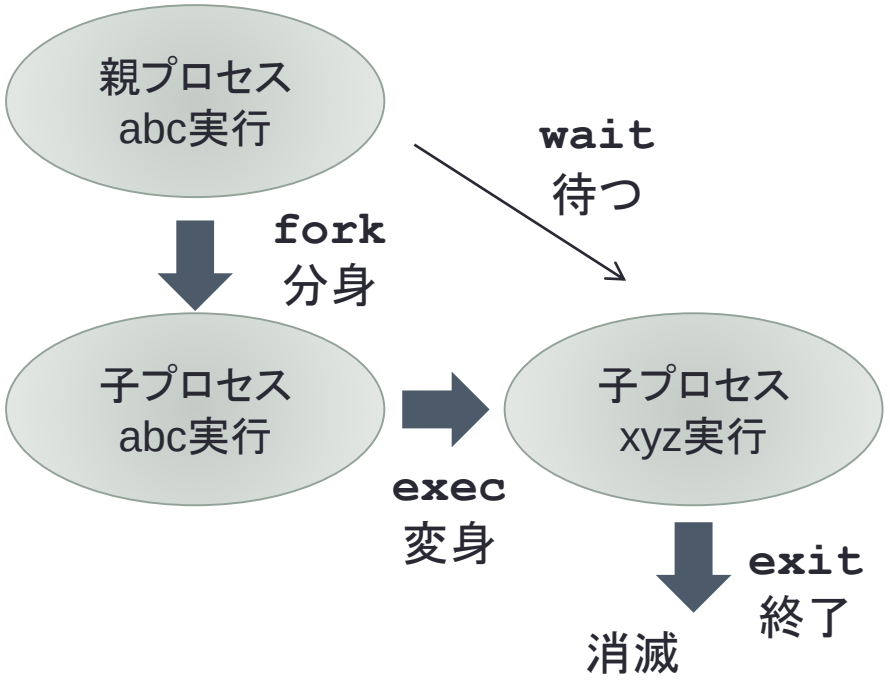
- 実行状態のプログラム
  - 同じプログラムを実行している複数のプロセスが存在する.

- プロセスは以下のもので構成されている:
  - プログラム
  - CPUの状態（レジスタ，PC，SP）
  - データ
  - メモリ空間
  - ファイル・ディスクリプタ
  - 現在の作業ディレクトリ
  - ルートディレクトリ
  - その他のプロセス関係の状態



プロセス関係のシステムコール

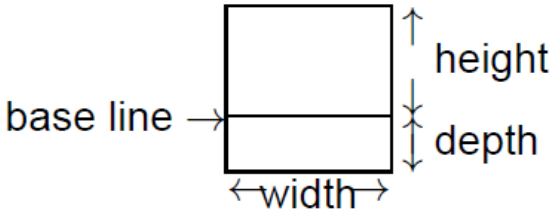
- **fork**
  - 子プロセスを作る
  - 中身は親プロセスと同じ
  - 分身
- **exec**
  - 実行するプログラムを指定する
  - 現在のプログラムを捨てる
  - 変身
- **wait**
  - 子プロセスが停止するまで待つ
- **exit**
  - プログラムを停止する
- その他
  - **signal**
    - 割り込み処理の指定
  - **kill**
    - 処理の割り込み

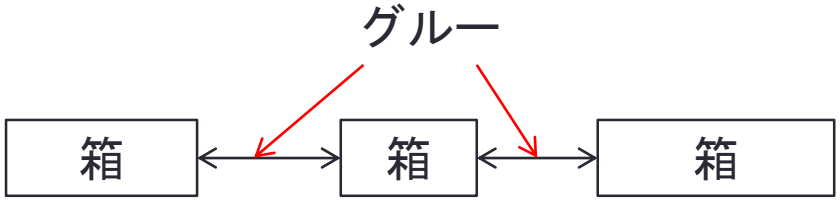


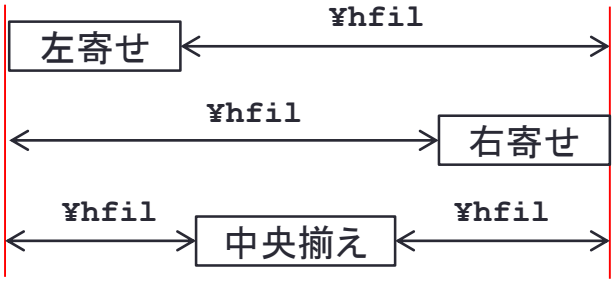
プロセスには親子関係がある

第4回　文書清書システム

- TeXの特徴
  - 数式の清書
  - マクロ機能
  - WEBによる記述
- TeXによる清書
  - hboxとvbox
  - 箱と箱をつなげるグルー
  - 無限に伸びるグルー
  - 最適化による行分割
- その他
  - LaTeX
  - Postscript
  - PDF

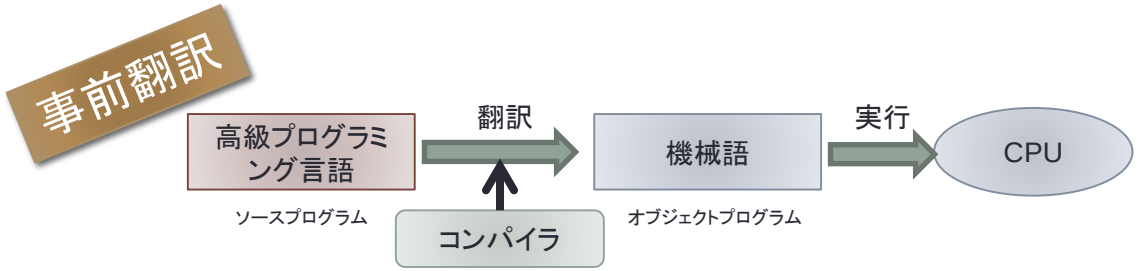


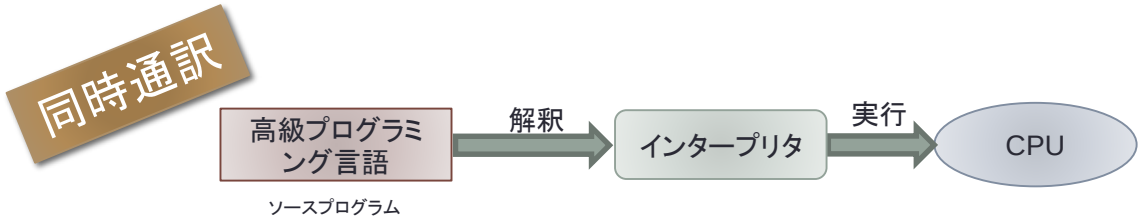




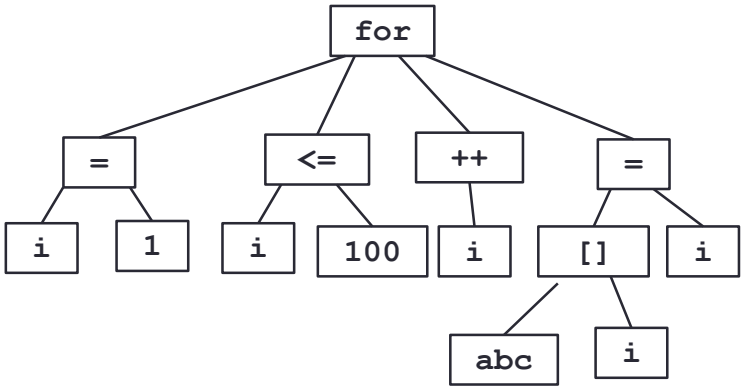
第5回　コンパイラ

- プログラミング言語
  - 高級・低級
  - コンパイラ・インタープリタ
- コンパイラの構成
  - 前処理
  - 字句解析
  - 構文解析
  - 意味処理
  - 最適化
  - コード生成
- 構文解析
  - 構文木
  - 文脈自由文法・BNF
- 最適化
  - Peephole最適化
  - 繰り返しの最適化

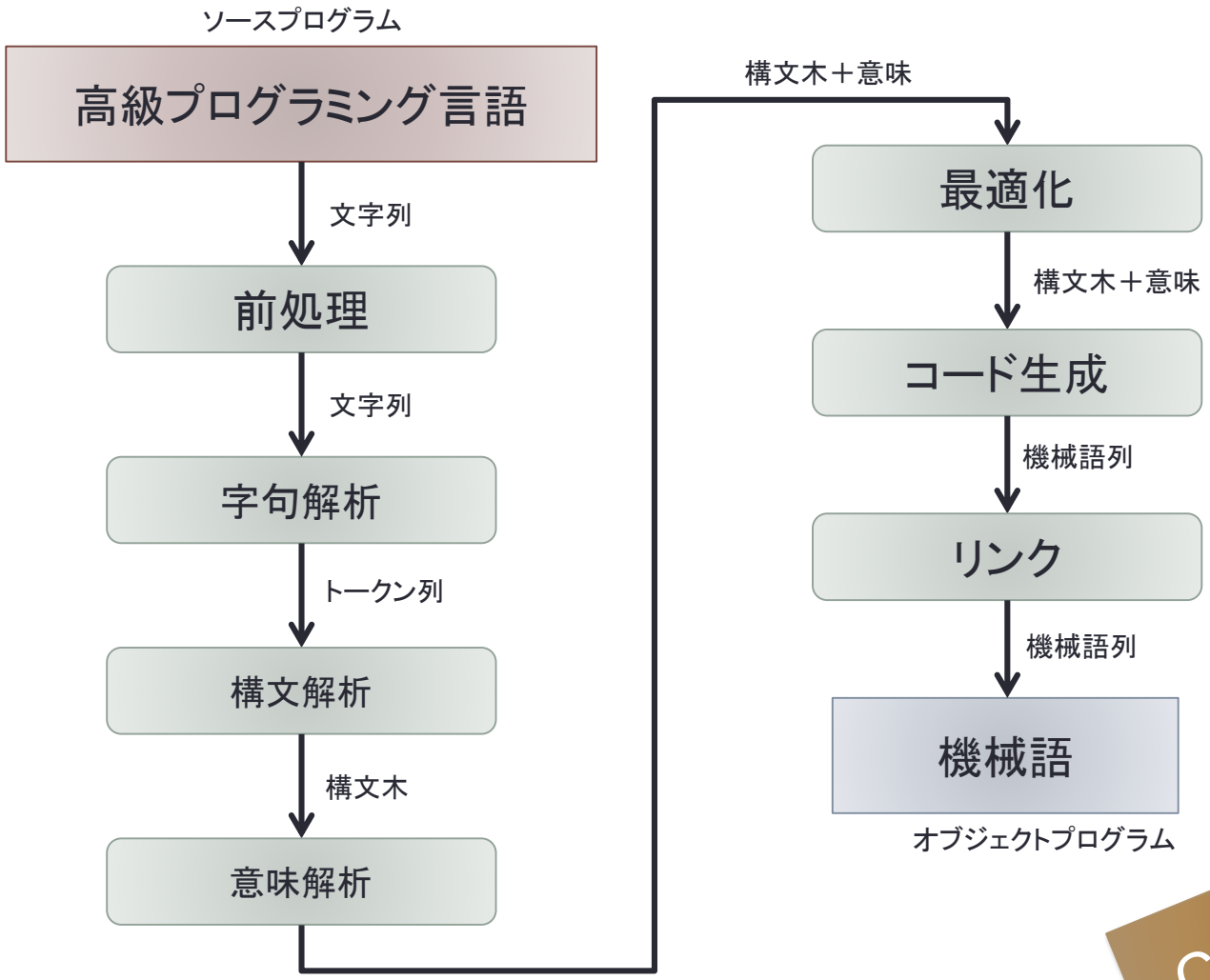




構文木
(parse tree)



コンパイラの構成



C言語を例とする

構文規則

- for文の構文規則

- 構文規則は文脈自由文法として与えられることが多い
  - BNF（Bacus Naur Form）による定義

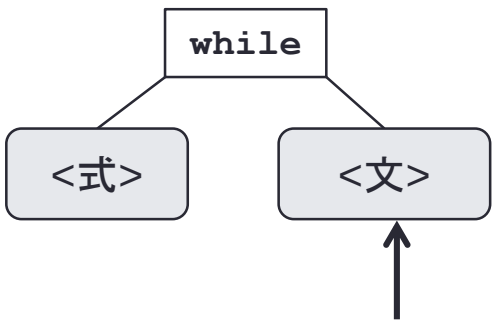
```
<for> ::= 'for' '(' <expr> ';' <expr> ';' <expr> ')' <statement>
```

- C言語の構文規則は大きく5つに分けられる
  - 式
  - 文
  - 関数
  - 変数宣言
  - 型宣言

英語の5文型
S V
S V C
S V O
S V O O
S V O C

while文のコード生成

<while文> ::= 'while' '(' <式> ')' <文>



<式>が0になるまで実行

コード生成

```
label1:
```

<式>に対するコード　　<式>を計算

```
cmp 0,%eax
je label2
```

<式>の値が0なら
label2へジャンプして終了

<文>に対するコード　　<文>を実行

```
jmp label1
label2:
```

もう一度<式>をチェック

より良い
コード生成

```
  jmp label2
label1:
```

<文>に対するコード

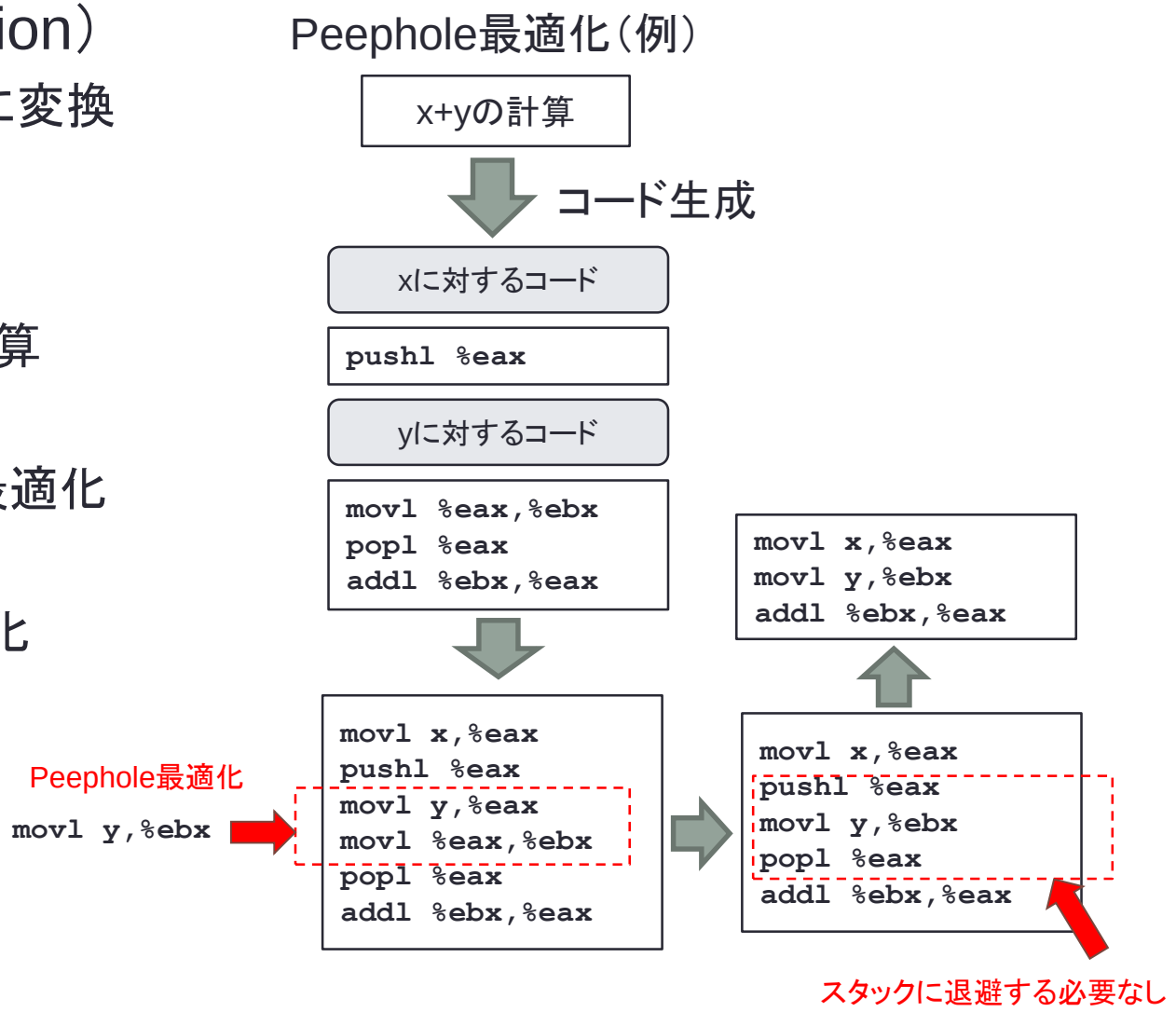
```
label2:
```

<式>に対するコード

```
cmp 0,%eax
jne label1
```

最適化

- 最適化（optimization）
  - 効率の良いコードに変換
- 最適化の手法
  - 定数だけの式の計算
  - Peephole最適化
  - 繰り返しにおける最適化
  - レジスタの利用
  - グローバルな最適化



繰り返しの最適化

- 繰り返しで変化しないものは外に出す
- 掛け算は足し算に置き換える

```
for (i = 0; i < 1000; i++) {
  b = x * x + 3;
  a = a + b + i * 4;
}
```

変化しないものは外に出す

```
b = x * x + 3;
for (i = 0; i < 1000; i++) {
  a = a + b + i * 4;
}
```

掛け算を
足し算に

```
b = x * x + 3;
j = 0;
for (i = 0; i < 1000; i++) {
  a = a + b + j;
  j = j + 4;
}
```

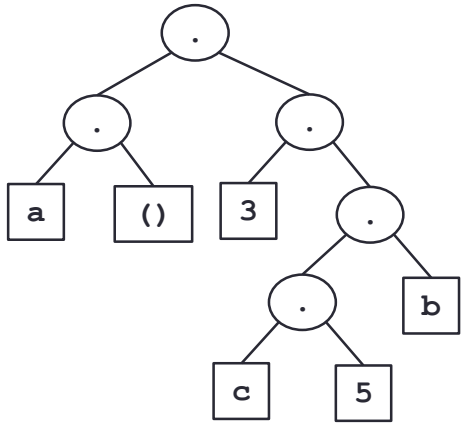
計算コスト

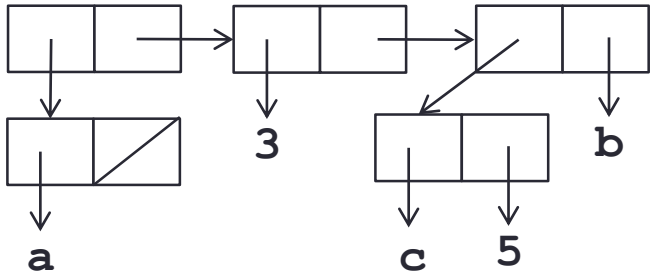
足し算
引き算 < 掛け算 < 割り算

第6回　LISP処理系

- LISP（LIST Processing）の特徴
  - ラムダ計算を実装した言語
  - 記号処理向き（FORTRANは数値計算向き）
  - 人工知能的な分野で多く利用
  - 関数型プログラミング言語
  - 再帰呼び出しによる繰り返し
- LISPオブジェクト
  - 単一データ型
  - プログラムもデータ
  - アトムとリスト
- 基本関数
  - carとcdr
  - cons
  - cond
- ガーベジコレクション

```
((a . ()) . (3 . ((c . 5) . b)))
```



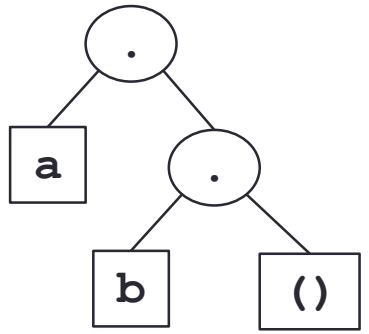


リスト

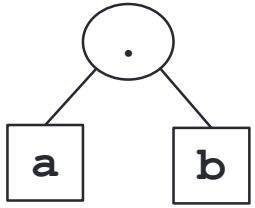
- リスト（あるいはS 式）

```
<リスト> ::= '()' | <アトム> | '(' <リスト> '.' <リスト> ')'
```

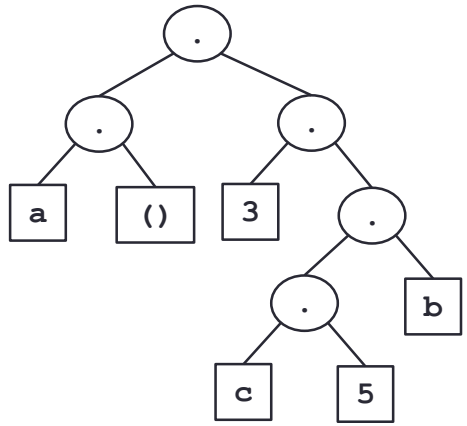
- () は空のリスト，nil とも書く



- リストはアトムを葉にもつ2分木

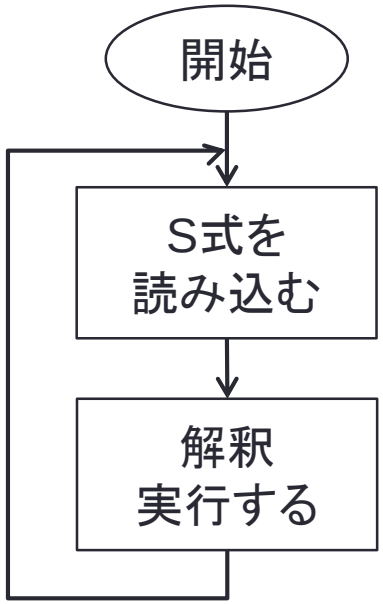


- 例
  - (a . b)
  - (a . (b . ()))
  - ((a . ()) . (3 . ((c . 5) . b)))



LISPインタープリタ

- インタープリタはプログラムを直接解釈実行する

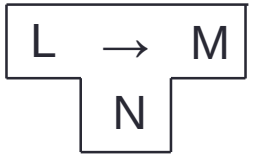


S式を読み込みリストにする

S式を簡略化・関数の適用を繰り返す

LISP in LISP

- T図式

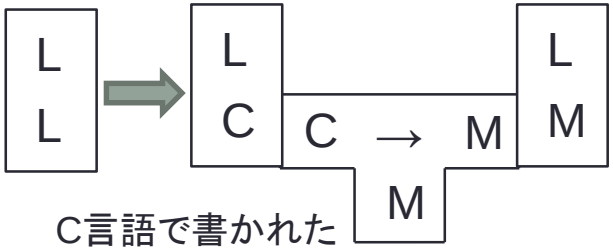


N言語で書かれた
言語Lから言語M
へのコンパイラ

N言語で書かれた
言語Lのインタープリタ

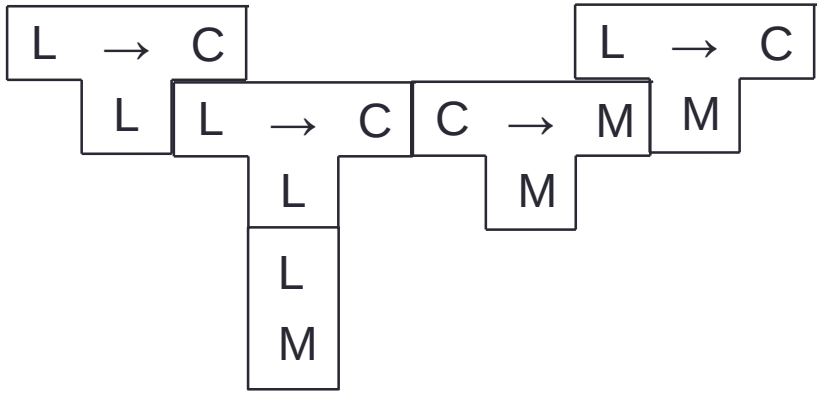
- LISPの仕組みをLISPで説明
  - **eval**が評価する関数
  - **(eval e a)**: 状態**a**(変数へのバインド)で式**a**を評価

LISPで書かれた
LISPインタープリタ



C言語で書かれた
LISPインタープリタ

LISPインタープリタを拡張



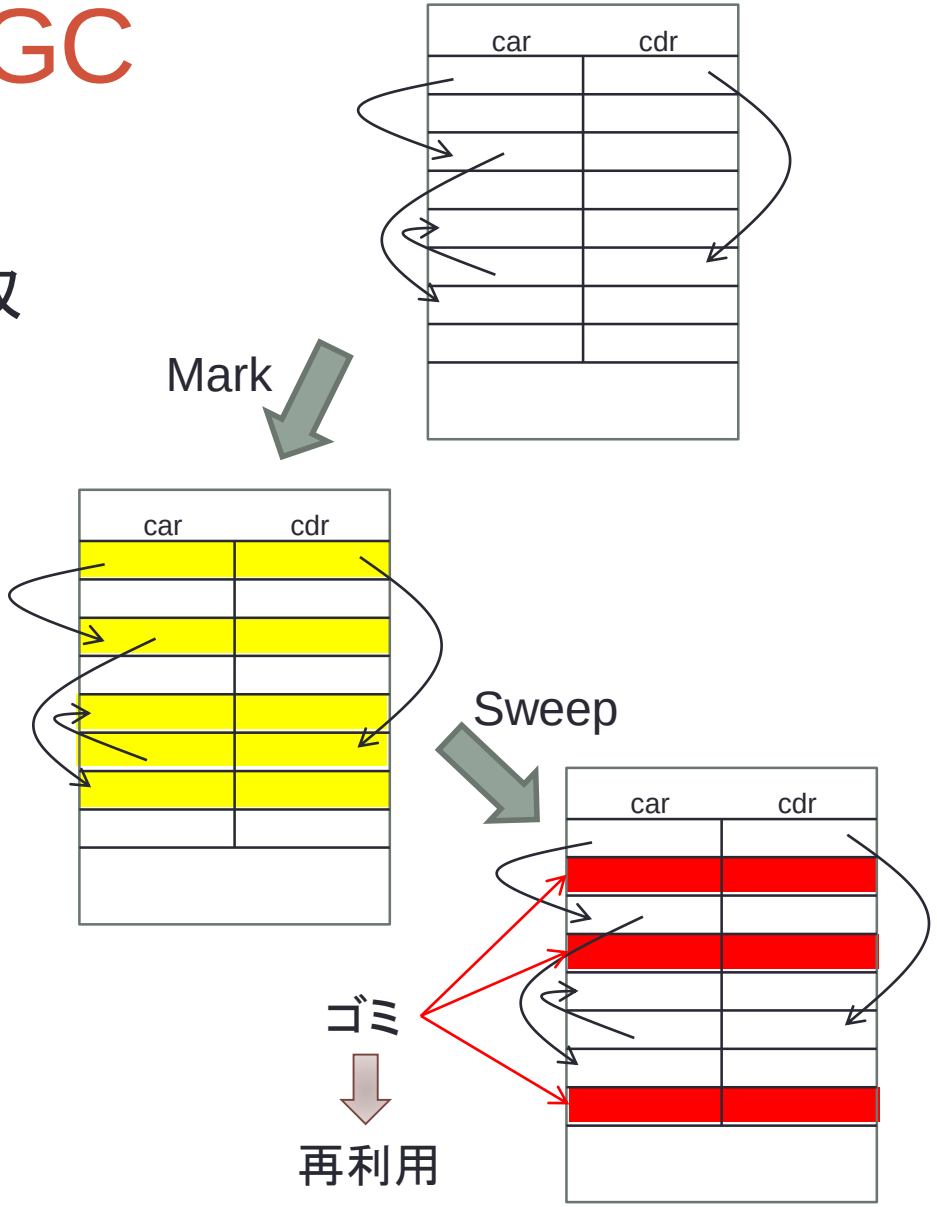
LISPで書かれた
LISPからCへのコンパイラ

Mark and Sweep GC

- 使っているセルに印を付ける
- 印の付いていないセルを回収

```
struct cons {
   struct cons *car;
   struct cons *cdr;
   int marked;
} cell[N], *free;

mark(struct cons *x) {
   if (x->marked) return;
   x->marked = 1;
   mark(x->car);
   mark(x->cdr);
}
sweek() {
   free = 0;
   for (i = 0; i < N; i++) {
      if (!cell[i].marked) {
         cell[i].car = free;
         free = &cell[i];
      }
   }
}
```

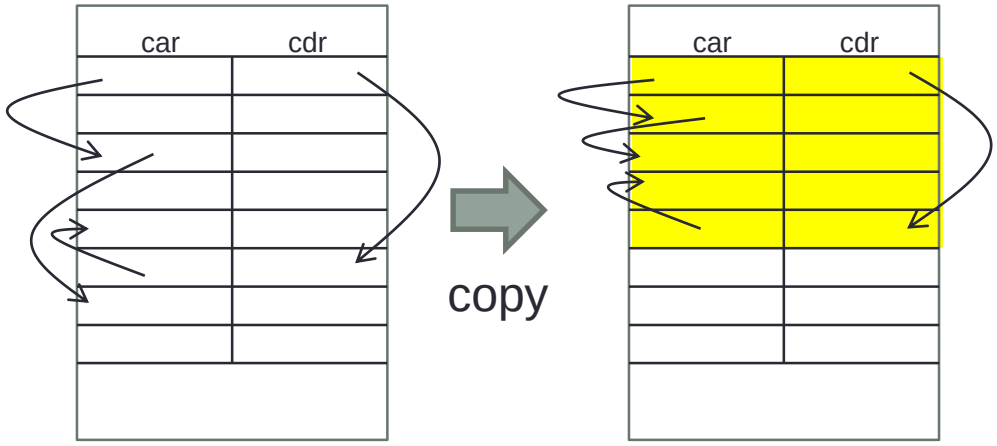


Copying GC

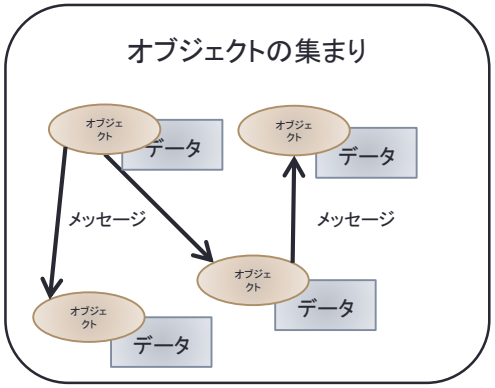
- 使っているセルを別のところにコピーする
- 領域が2倍必要

```
struct cons {
   struct cons *car;
   struct cons *cdr;
   int copied;
};

copy(struct cons *x) {
   struct cons *y, *car, *cdr;
   if (x->copied) return x->car;
   car = x->car;
   cdr = x->cdr;
   x->car = y = new();
   x->copied = 1;
   y->car = copy(car);
   y->cdr = copy(cdr);
   return y;
}
```

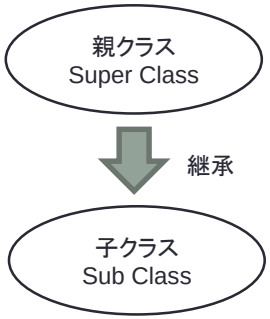


第7回　Java仮想機械



- Javaとは
  - オブジェクト指向言語
  - 構文はC言語に類似

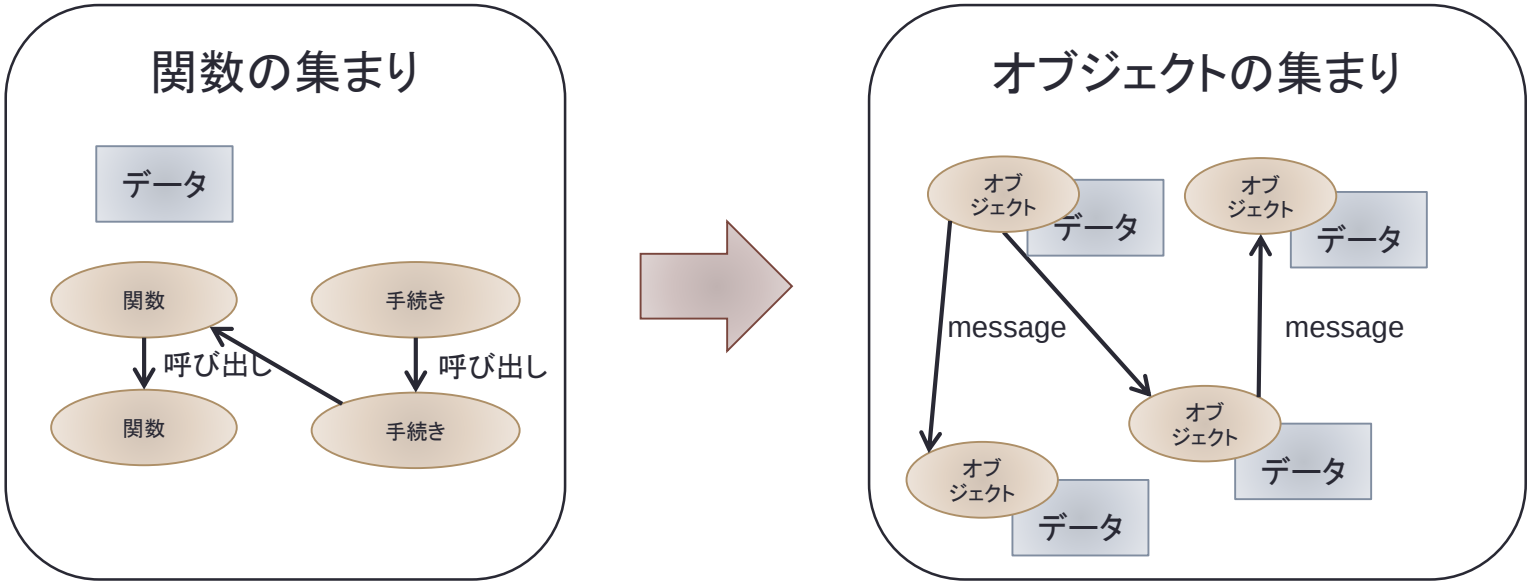
- オブジェクト指向プログラミング言語
  - オブジェクトがメッセージを行うことでプログラムを構成する
  - カプセル化
  - クラスと継承



- Java仮想機械
  - スピードとポータビリティの両立
  - JITコンパイラ

オブジェクト指向プログラミング言語とは

- オブジェクトの集まりでソフトウェアを構成する
  - オブジェクトは互いにメッセージを送りあう

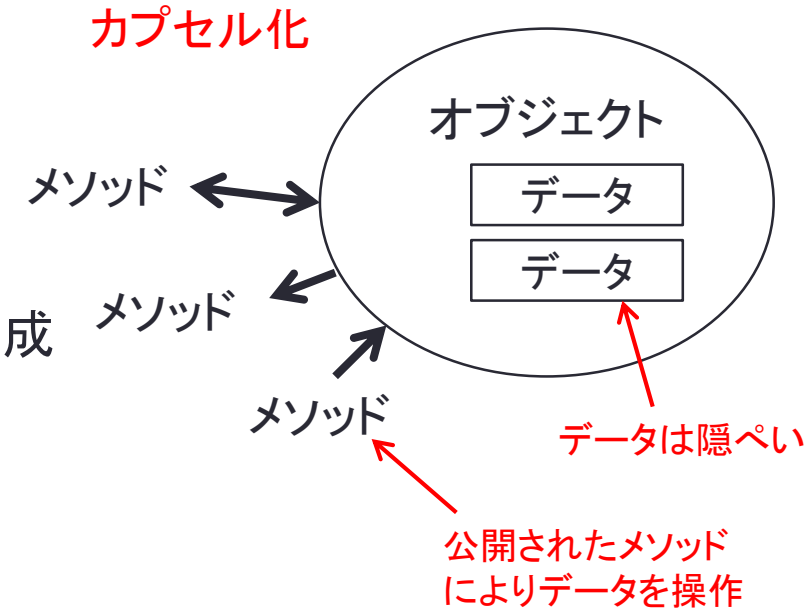


非オブジェクト指向
- 関数とデータは独立
- 複雑な処理を別関数にお願いする
- 関数・手続き中心の考え

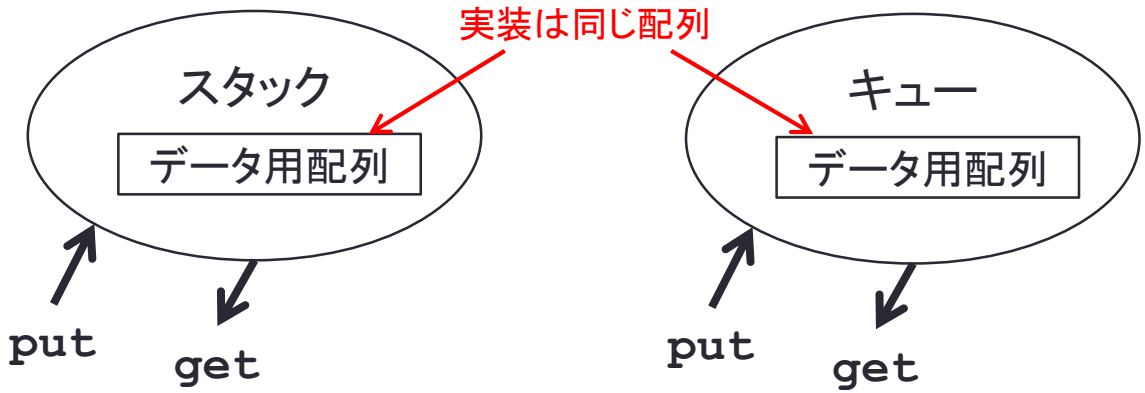
オブジェクト指向
- オブジェクトがデータを持つ
- オブジェクトのデータの処理はメッセージで行う
- データ中心の考え

オブジェクト指向の特徴(1)

- カプセル化
  - データをオブジェクト内にカプセル化
  - メソッドによりデータにアクセス
  - オブジェクトはデータとメソッドから構成される
  - 抽象データ型に由来



抽象データ型

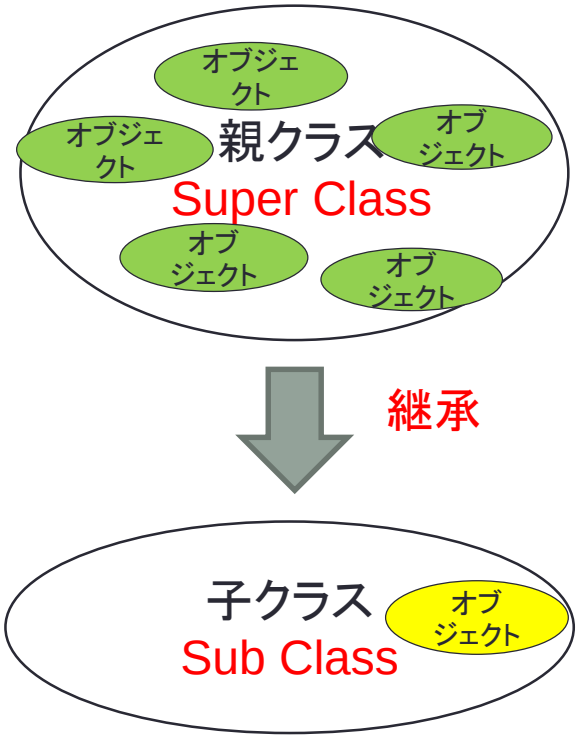


最後にputしたものがgetできる
LIFO = Last In First Out

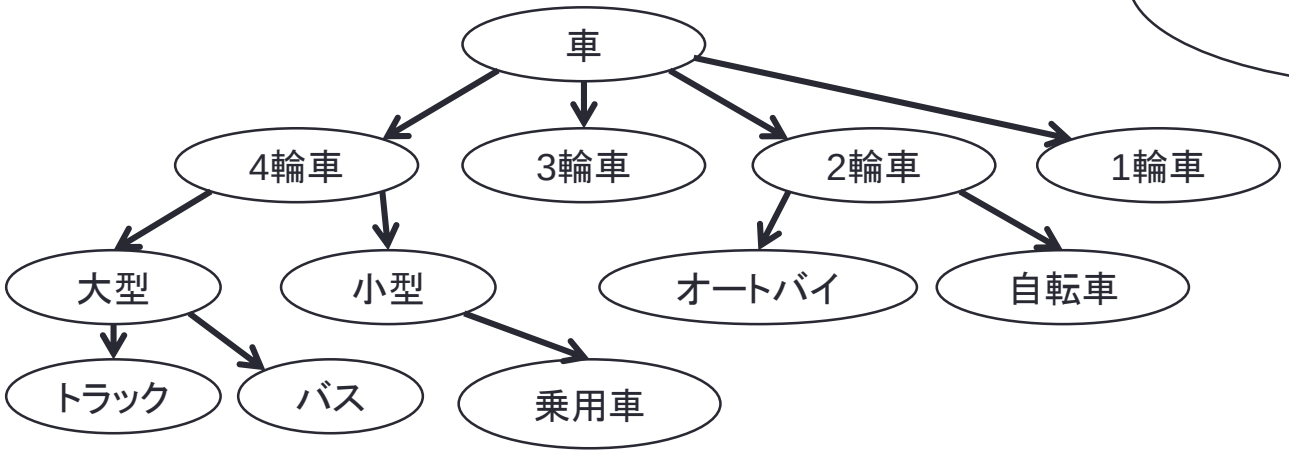
最初にputしたものがgetできる
FIFO = First In First Out

オブジェクト指向の特徴(2)

- クラスと継承
  - オブジェクトはクラスに属する
  - オブジェクトはクラスのインスタンス
  - クラスには親子関係があり，子は親を継承
    - 親のメソッドは子のメソッド
    - 親のデータは子のデータ



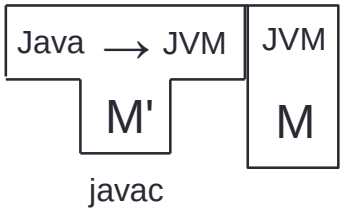
- クラス階層
  - クラスの親子関係で階層ができる



Java仮想機械

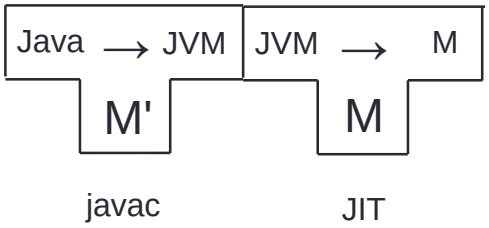
- スピードとポータビリティの両立
  - コンパイラ　＋　インタープリタ
  - Java仮想機械（Java Virtual Machine）の機械語に翻訳
  - 仮想機械をいろいろなアーキテクチャ上に作成

- Java仮想機械（Java Virtual Machine）
  - 仮想（理想）的に考えられたCPU
  - Javaがコンパイルしやすいように設計
  - Javaのオブジェクト指向の実装
  - メモリ管理をガーベジコレクションにより実装

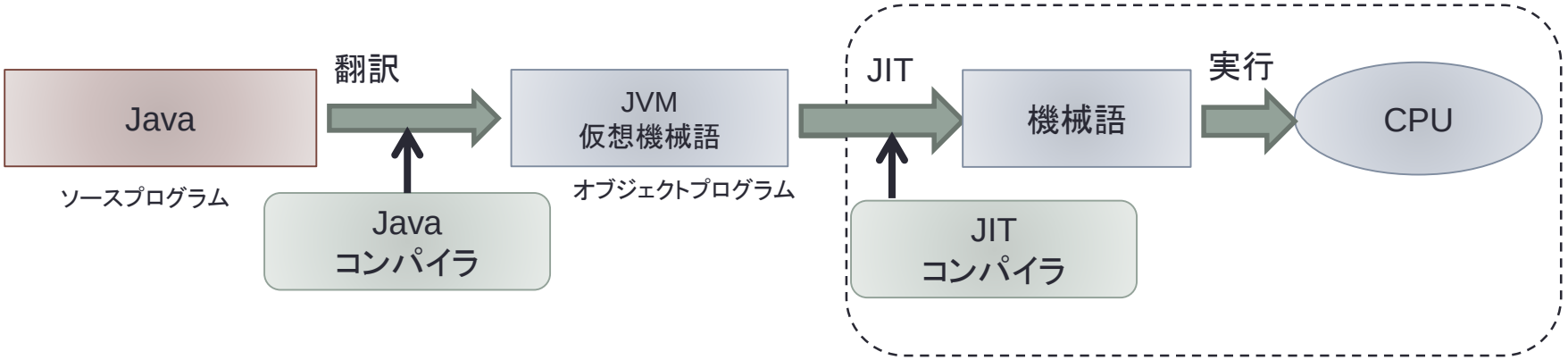


JITコンパイラ

- 仮想機械を使う場合の利点と欠点
  - ポータビリティが高くなる
  - 実行速度がネイティブ機械語より遅くなる

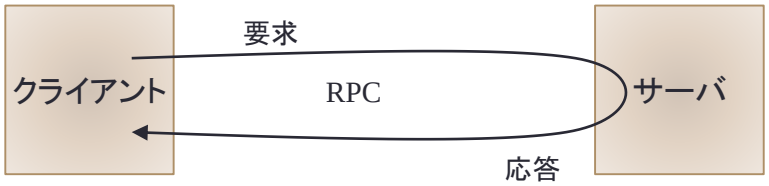


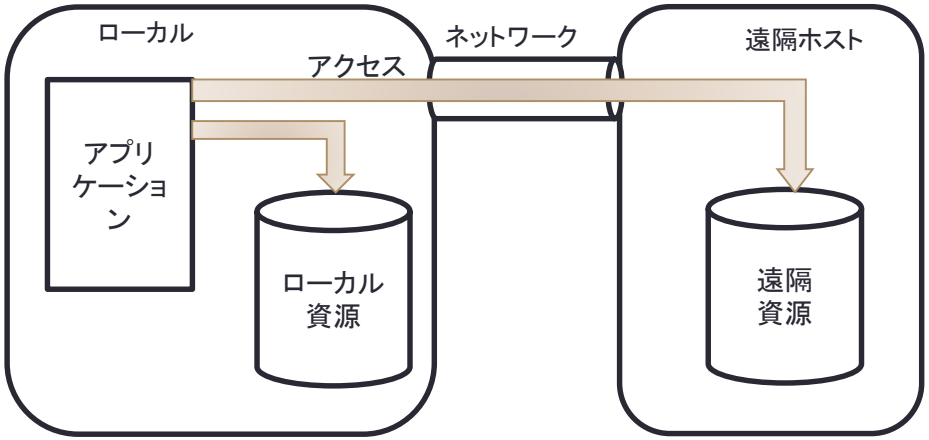
- JITコンパイラ
  - Just In Time
  - 仮想機械語を実行時にネイティブ機械語に変換する
  - 最初の変換に時間がかかるが，その後はネイティブと同じ速度

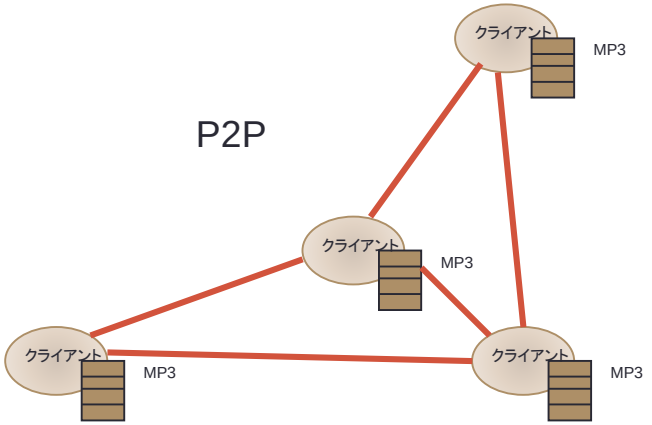


第8回　ネットワークシステム

- 分散システム
- 透過性
  - アクセス透過性
  - 位置透過性
  - 並行透過性
  - 複製透過性
  - 故障透過性
  - 移動透過性
  - 性能透過性
  - 規模透過性
- 通信モデル
  - クライアントサーバモデル
  - RPC
  - 関数移送（function shipping）
  - グループマルチキャスト
  - P2P

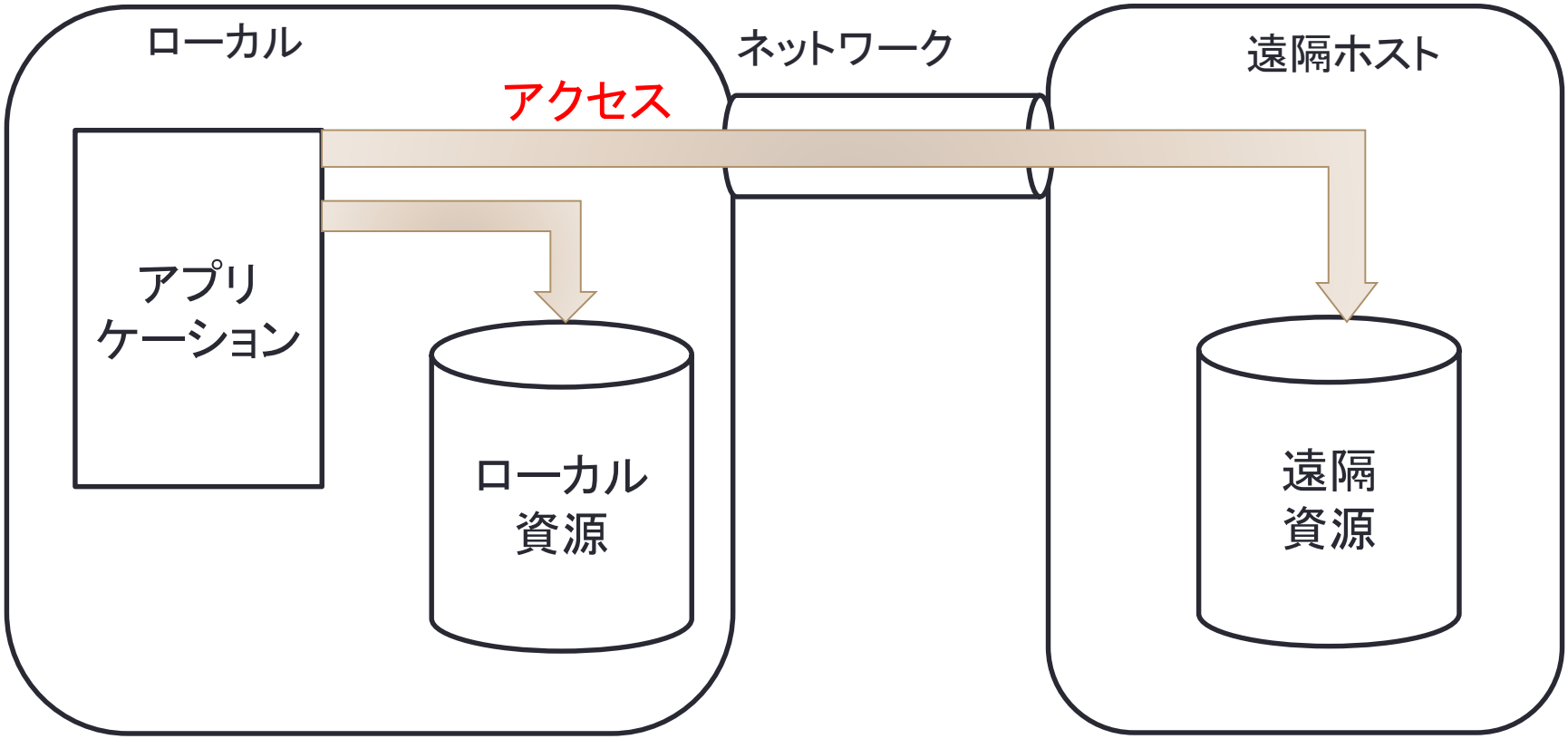






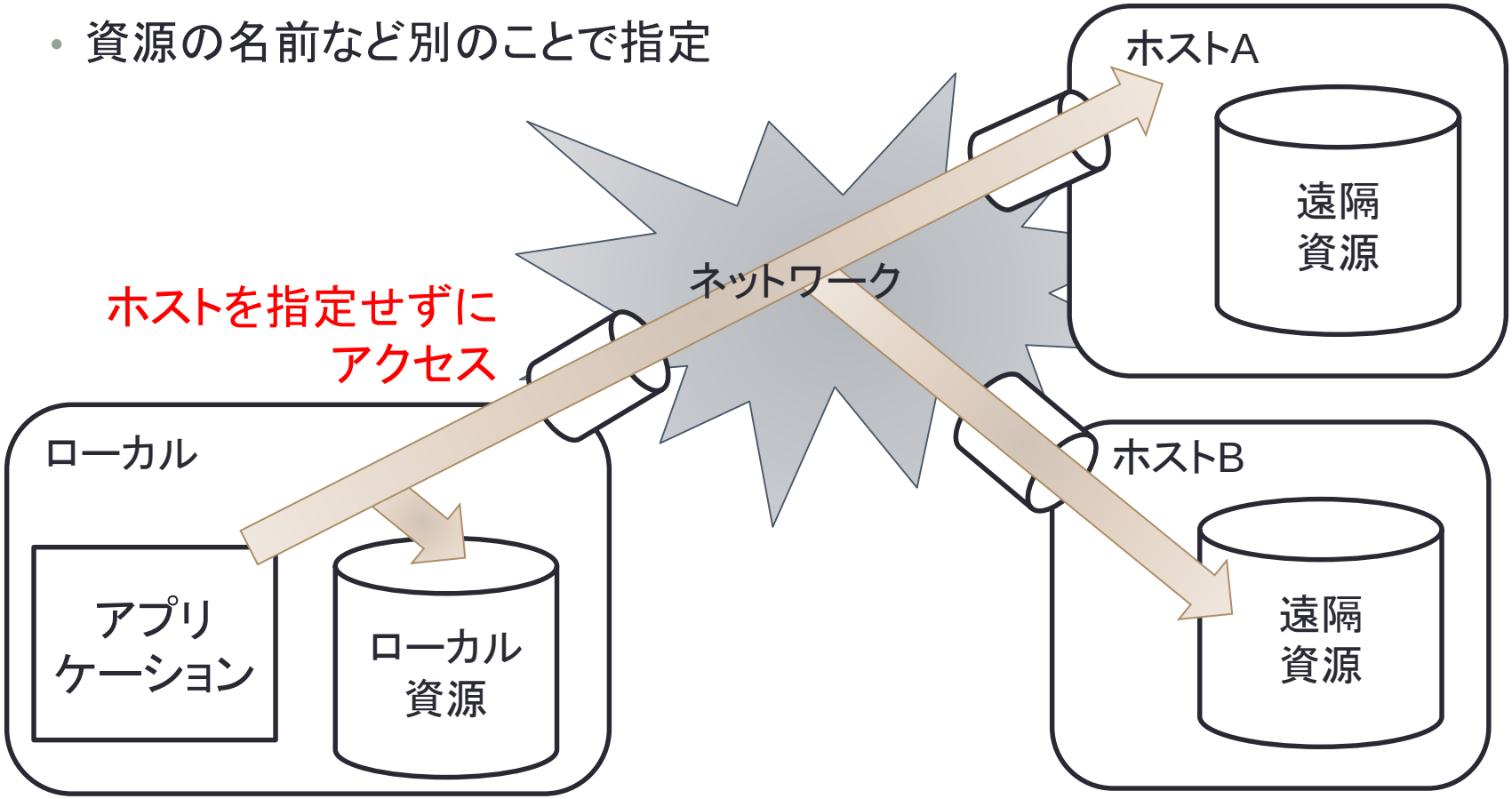
アクセス透過性

- ローカルと遠隔の資源を同じようにアクセス可能
  - 遠隔資源のアクセスに特別なことする必要はない



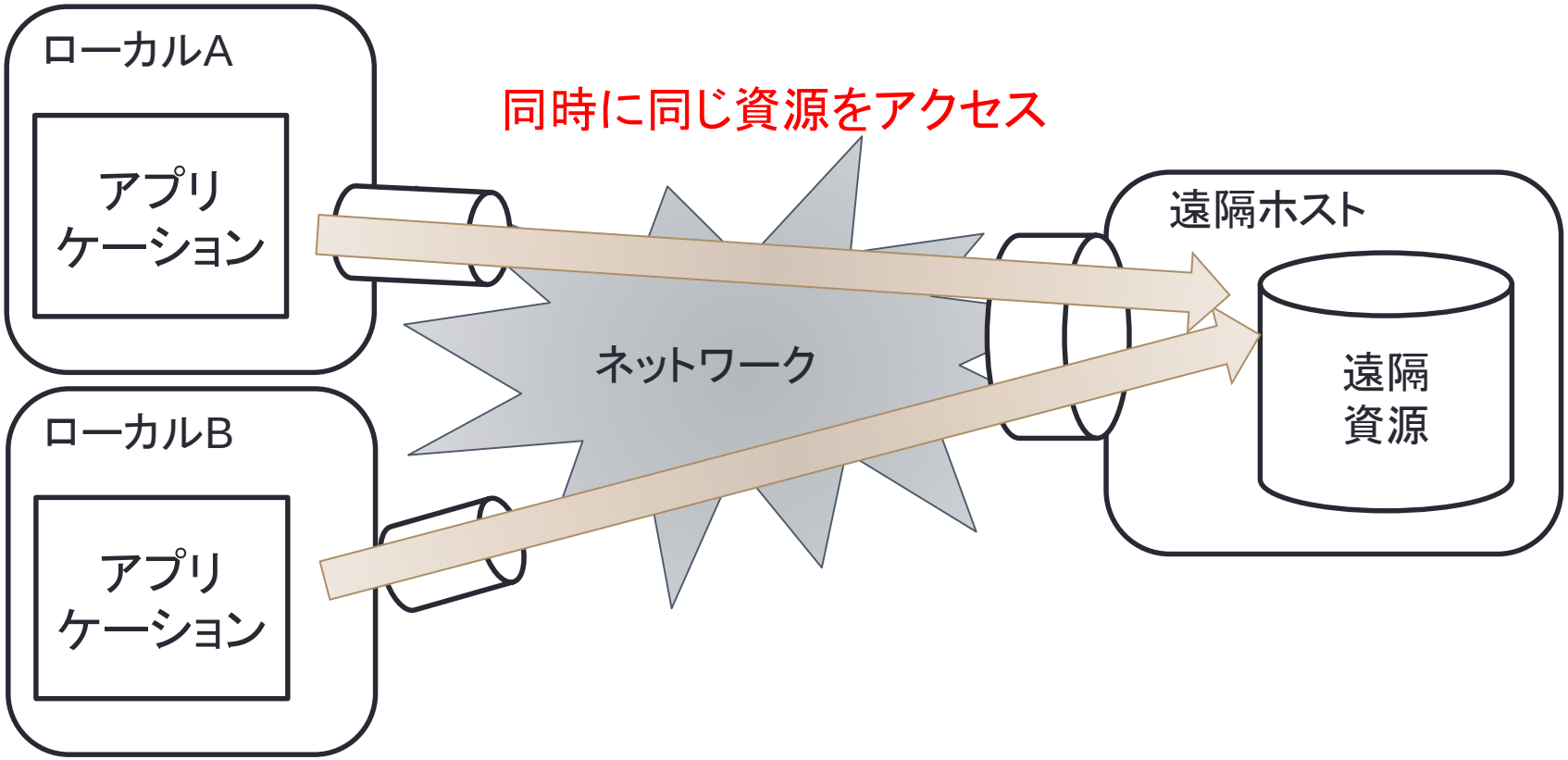
位置透過性

- 資源の位置に関する知識なしにアクセス可能
  - どこにあるかは知らなくても良い
  - 資源の名前など別のことで指定



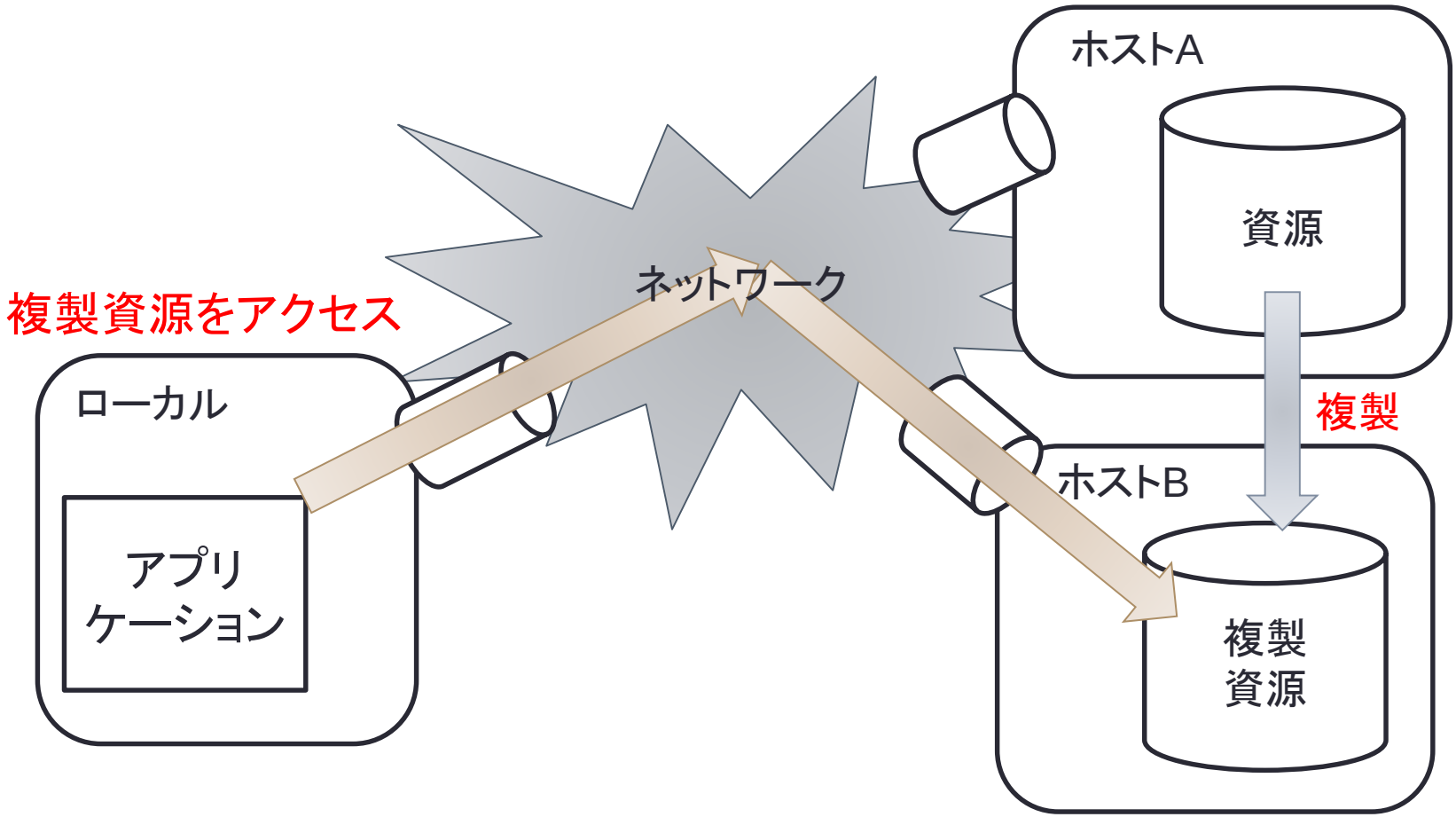
並行透過性

- 複数から同時に並行して操作が可能
  - だれかが利用中は使えないなどない



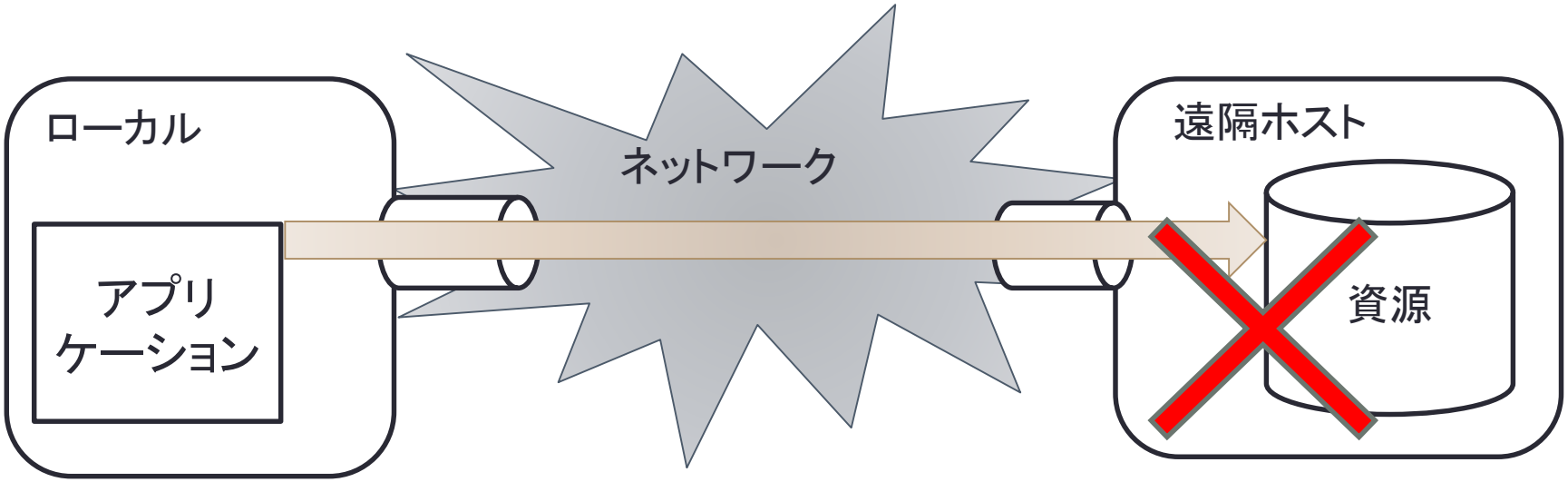
複製透過性

- 信頼性と性能を向上するために複製して処理可能



故障透過性

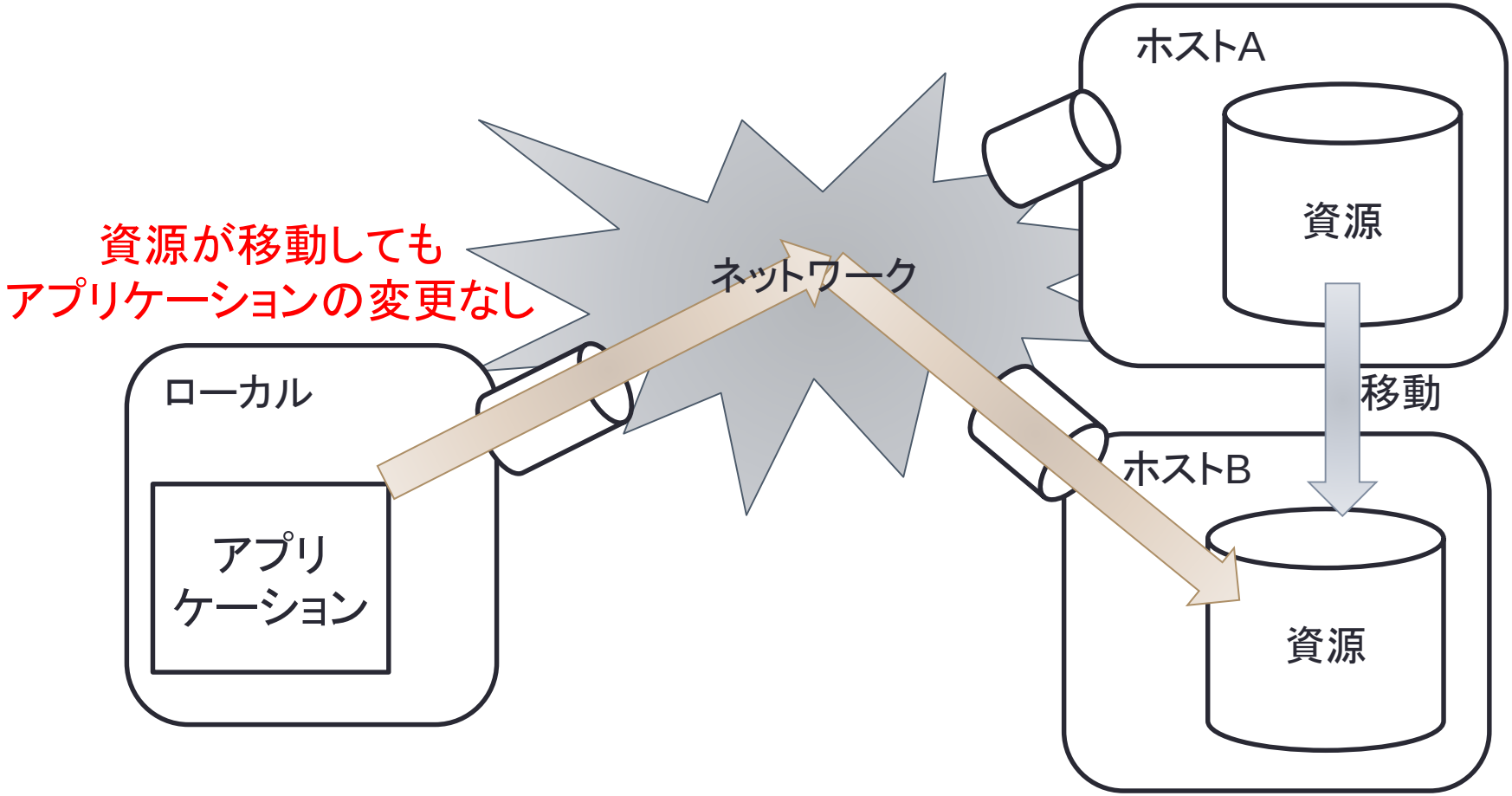
- ソフトウェアおよびハードウェアの故障にもかかわらず，故障を隠蔽して操作可能



ハード・ソフトの故障でも操作可能

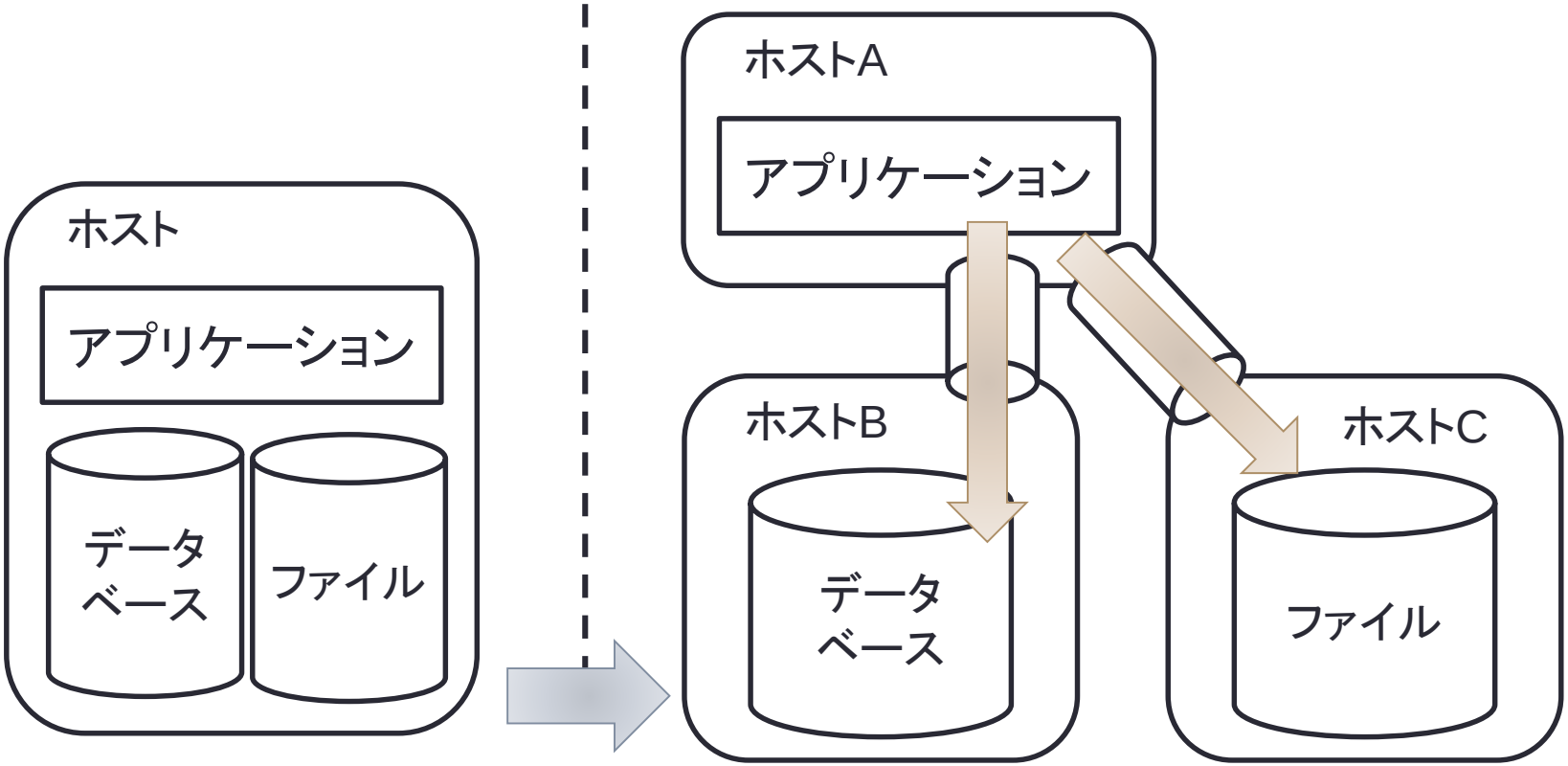
移動透過性

- 資源を移動させてもユーザおよびアプリケーションには影響を与えない



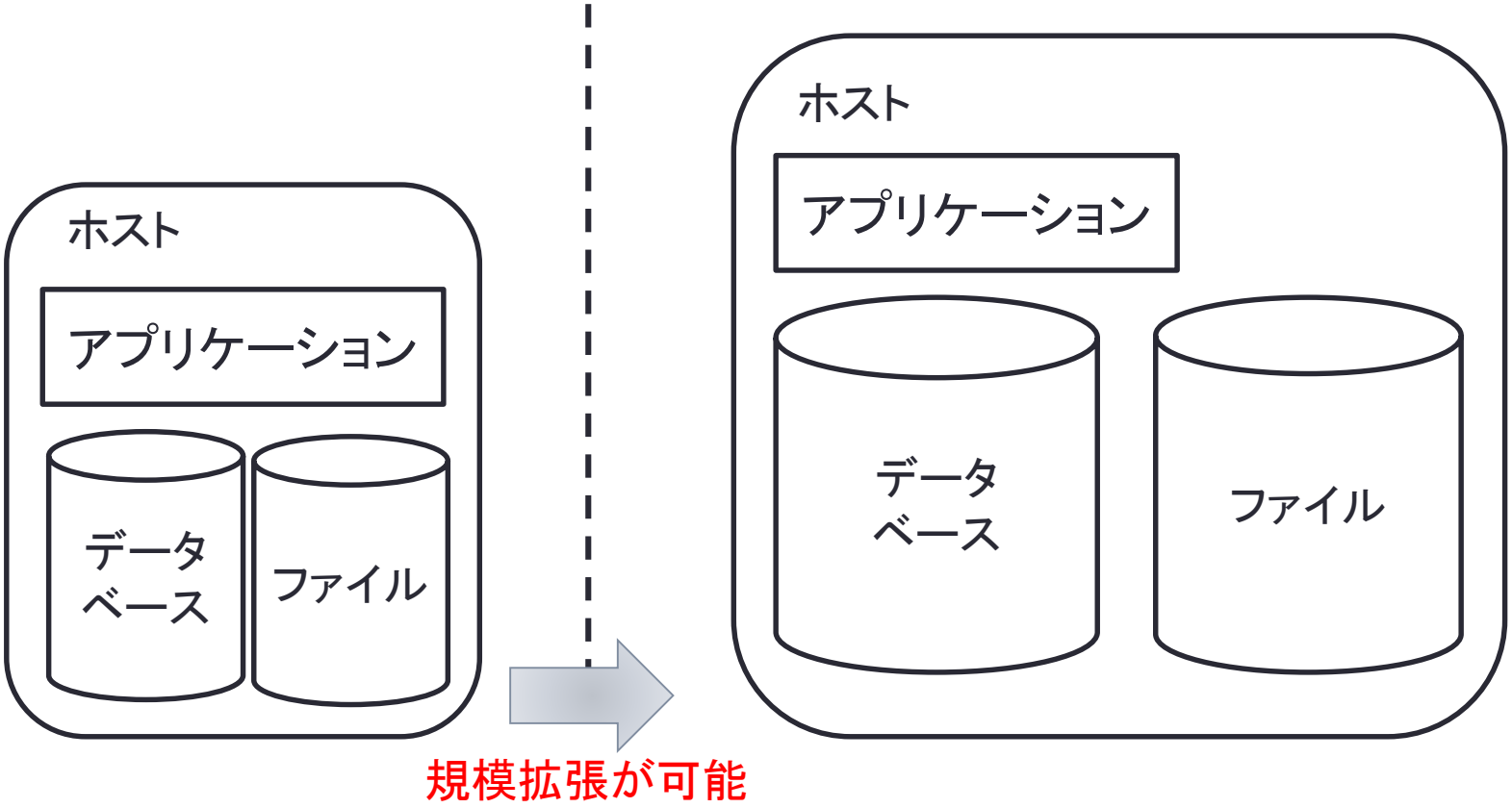
性能透過性

- システムの負荷が変化するに従い，性能を改善するための再構成が可能



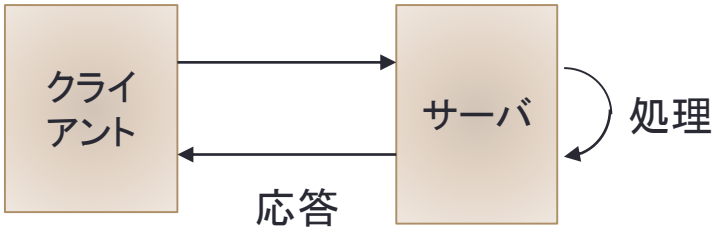
規模透過性

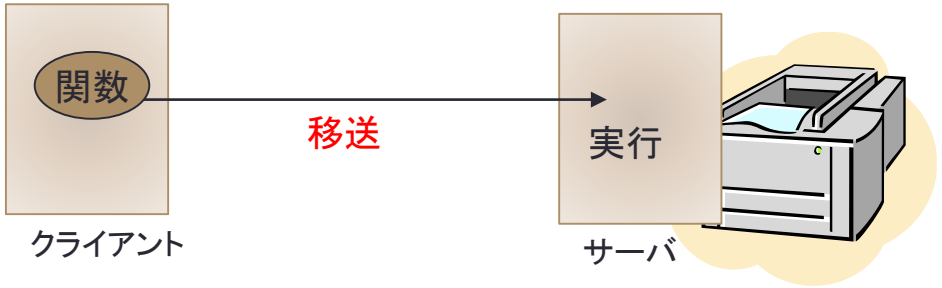
- システム，アプリケーションの構造を変更することなく規模の拡張が可能

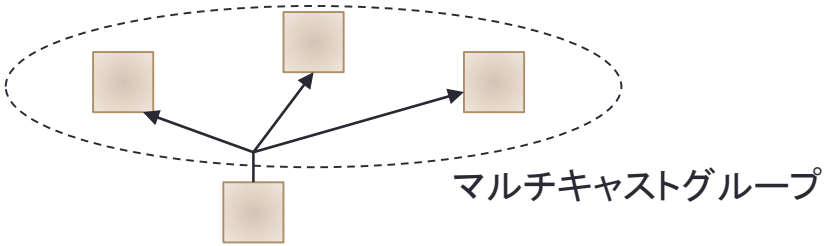


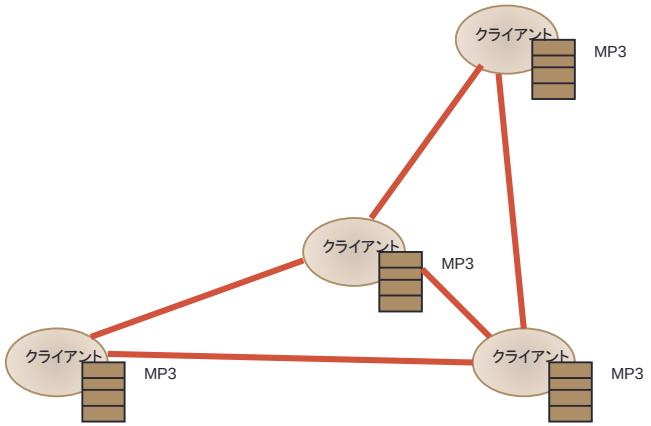
分散システムにおける通信

- 分散システムは分散しているため通信が必要
- 代表的な通信形態
  - クライアントサーバモデル
  - RPC
  - 関数移送（function shipping）
  - グループマルチキャスト
  - P2P









RPCとローカル手続きの違い

- 処理途中での障害を考慮しなくてはならない
  - 要求メッセージを再送するのか？
    - 要求がサーバに届いていないかもしれない.
  - 要求メッセージの重複を削除するのか？
    - 要求が再送される場合，重複を削除しないと，二重に実行されるかもしれない.
  - 応答を再送するのか？
    - 応答がクライアントに帰っていないかもしれない.
    - 再送するためには応答を残しておく必要がある

RPCのセマンティックス

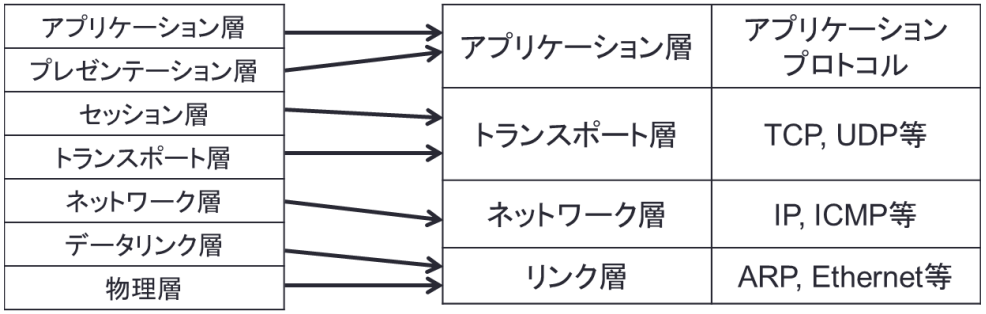
- 不確実呼び出し (maybe call)
  - 要求の再送は行わない
    - 要求の再送はない → 重複の削除は必要ない
  - 応答の再送は行わない

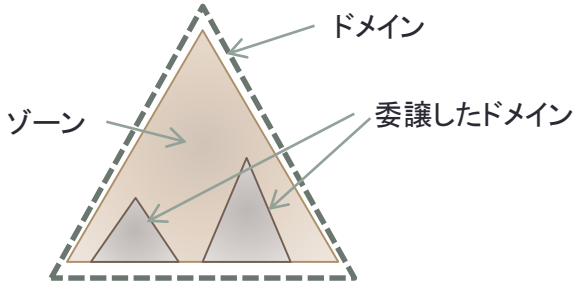
- 最低一度呼び出し(at-least-once call)
  - 要求メッセージの再送を行う
  - サーバ側で重複はチェックしない
  - クライアントは応答が返るまで要求の再送を行う
  - サーバは処理を少なくとも一度は行う
  - idempotentな処理に向く

- 最大一度呼び出し(at-most-once call)
  - 要求メッセージの再送を行う
  - サーバ側で重複のチェックを行う
  - サーバは処理を多くとも一度しか行わない
  - トランザクション処理などはこのタイプ

第9回　名前解決

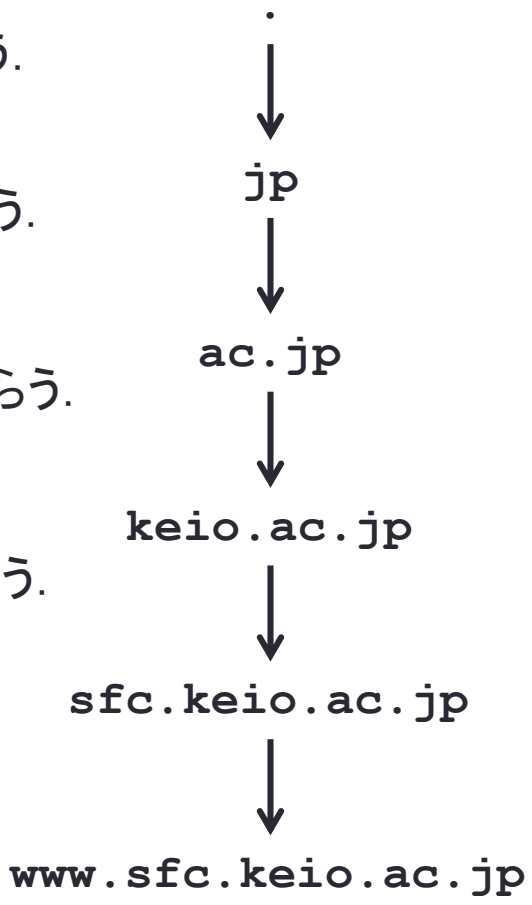
- インターネットプロトコル
  - TCP/IP
  - IPv4とIPv6
- アプリケーションプロトコル
- 階層的な名前管理
  - DNS
  - ルートサーバ
    - 13個
    - anycastの利用
  - キャッシュの活用
  - メールアドレスの解決





名前の解決

- **www.sfc.keio.ac.jp**のアドレスを問い合わせる
  1. ルートネームサーバに問い合わせる.
     - jpネームサーバに問い合わせるように返事をもらう.
  2. jpネームサーバに問い合わせる.
     - acネームサーバに問い合わせるように返事をもらう.
  3. acネームサーバに問い合わせる.
     - keioネームサーバに問い合わせるように返事をもらう.
  4. keioネームサーバ問い合わせる.
     - sfcネームサーバに問い合わせるように返事をもらう.
  5. sfcネームサーバに問い合わせる,
     - wwwのアドレスを教えてもらう.



キャッシュ

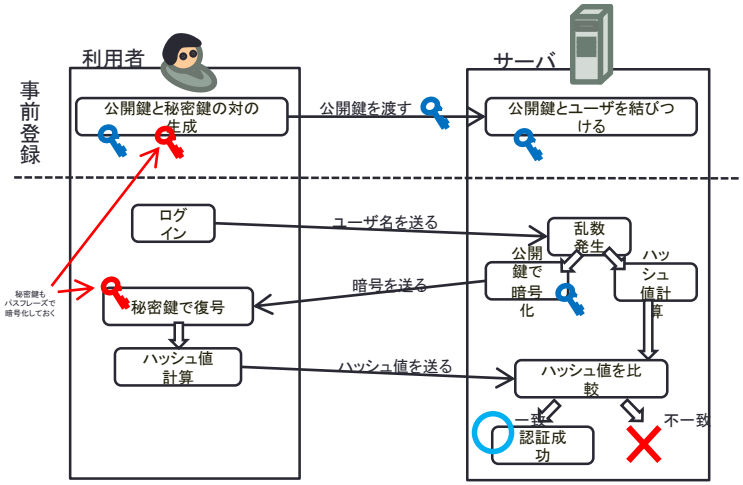
- ネームサーバに対する負荷を軽減させなくてはいけない.
  - 問い合わせた結果はキャッシュしておく
  - キャッシュされたデータにはTTL（Time To Live）が設定されていて，その時間が過ぎると無効になる.
  - 存在しない名前に対する返答（存在しないこと）もキャッシュしておかないと，DOSの標的となる.
- TTLは通常1日程度に設定されている.
  - IPアドレスの変更は頻繁に行うものではない.
  - 長く設定しすぎると，IPアドレスの変更が反映に時間がかかる.

13個のルートネームサーバ

頭文字	IPv4アドレス	IPv6アドレス	管理者	サーバ所在地
A	198.41.0.4	2001:503:ba3e::2:30	Verisign (US)	anycast
B	170.247.170.2	2801:1b8:10::b	UCS-ISI (US)	Marina Del Rey, California, USA
C	192.33.4.12	2001:500:2::c	Cognet Communications (US)	anycast
D	128.8.10.90	2001:500:2d:d	University of Maryland (US)	College Park, Maryland, USA
E	192.203.230.10	2001:500:a8::e	NASA (US)	Mountain View, California, USA
F	192.5.5.241	2001:500:2f::f	ISC (US)	anycast
G	192.112.36.4	2001:500:12::d0d	U.S. DoD NIC	anycast
H	198.97.190.53	2001:500:1::53	U.S. Army Research Lab	Aberdeen, Proving Ground, Maryland, USA
I	192.36.148.17	2001:7fe::53	Netnod (Sweden)	anycast
J	192.58.128.30	2001:503:c27::2:30	Verisign (US)	anycast
K	193.0.14.129	2001:7fd::1	RIPE NCC (Holand)	anycast
L	199.7.83.42	2001:500:3::42	ICANN (US)	anycast
M	202.12.27.33	2001:dc3::35	WIDE Project (Japan)	anycast

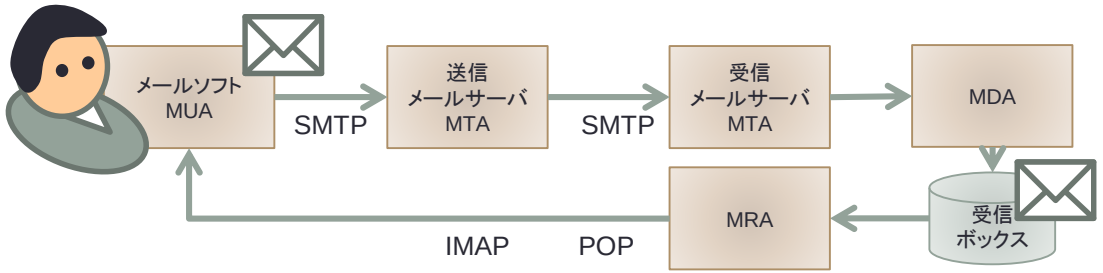
第10回　遠隔利用と電子メール

- Telnet
  - もっとも古いTCPプロトコルの一つ
  - 単純に仮想端末をインターネット上で実装したもの
  - セキュリティの問題あり

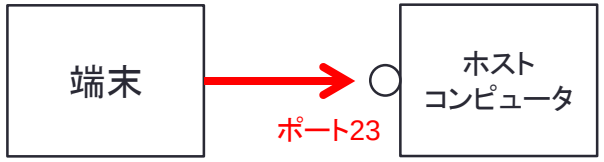


- SSH
  - 暗号化および認証をもつ端末プロトコル
  - 複数のユーザ認証をサポート
  - 1つの接続で複数のチャネルをサポート

- 電子メール
  - もっともよく使われているインターネットプロトコルの一つ
    - 複数のRFCで規定
  - メールクライアント（MUA）の設定
    - 電子メールの送信と受信ではプロトコルが異なる
    - SMTP
    - POP/IMAP
  - セキュリティ上の問題も多い
    - SPAM



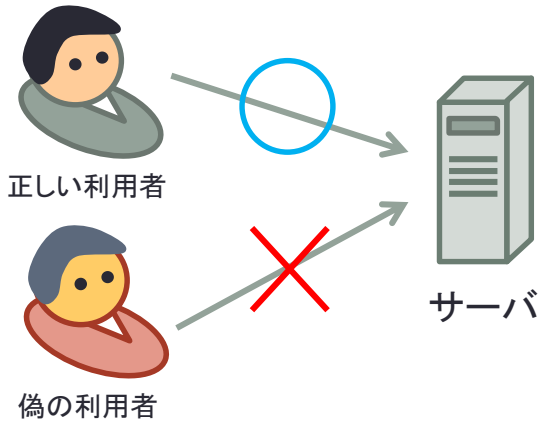
Telnetプロトコル



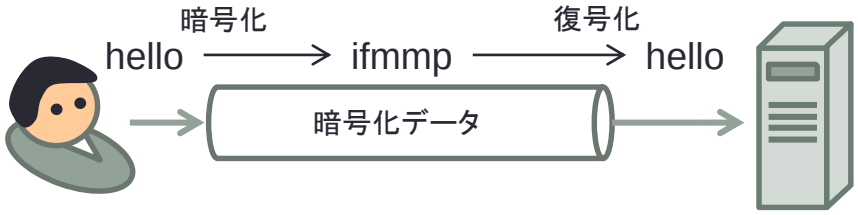
- インターネットを使った仮想端末の実現
  - シリアル回線などの代わりにインターネットを通信に使う
  - TCPコネクション（セッション，信頼性あり）
    - ポート23
- 非常に単純なプロトコル
  - 端末から入力された文字を遠隔のコンピュータに送る
  - 遠隔のホストコンピュータが出力した文字を端末に表示する
  - シリアル回線の代わりを行うだけで，認証などについてこれまでの仕組みを用いる
- 端末とホストの通信におけるいくつかのオプションの交渉も可能
  - DO と DON'T
  - WILL と WON'T

ユーザ認証と暗号化とハッシュ

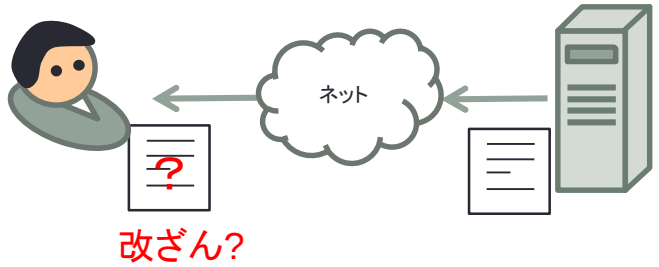
- ユーザ認証
  - 利用者が正しい利用者であることを確認する
  - パスワード認証
  - ワンタイムパスワード
  - チャレンジ／レスポンス認証
  - RSA（Rivest Shamir Adleman）認証
  - DSA（Digital Signature Algorithm）認証



- 暗号化
  - 通信の中身が他人に分からないようにする
  - 共通鍵暗号
    - 鍵を他人に知られてはいけない
  - 公開鍵暗号
    - 公開鍵と秘密鍵のペアを用いる



- ハッシュ
  - データが改ざんされていないことを確認する
  - チェックサム
  - 暗号学的ハッシュ関数(MD5，SHA-1, SHA-2)



SMTP

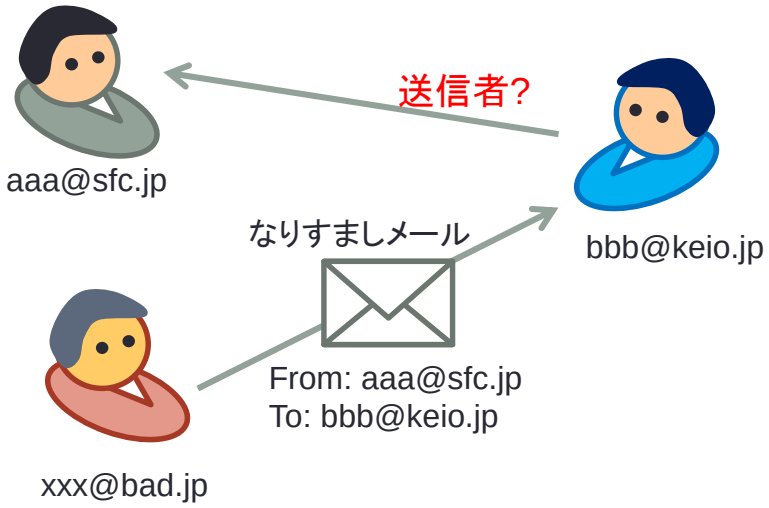
- Simple Mail Transfer Protocol
  - MUAがMTAにメールを送るときに利用
  - MTAが他のMTAにメールを転送するときに利用

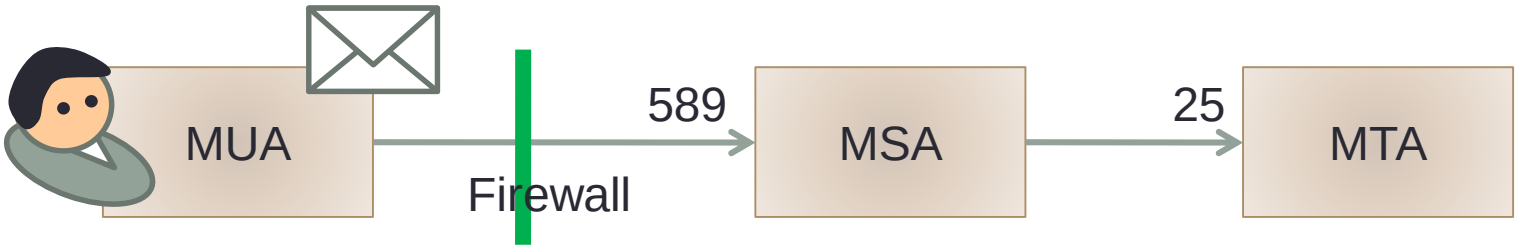
- 仕様
  - RFC821（1982年）が最初
  - 各種拡張機能が追加
  - ESMTP (Extended SMTP)

- TCPコネクション
  - ポート25

SMTPのセキュリティ

- 誰でもがメールを送ることができる
  - なりすましメールが可能
  - Spamができる
- MUAを限定する
  - POP before SMTP
  - SMTP AUTH
- 外部MTAの利用を制限
  - 外部の25番ポートへの通信を制限
  - MSA (Message Submission Agent)
- メール内容を暗号化
  - SMTP over SSL

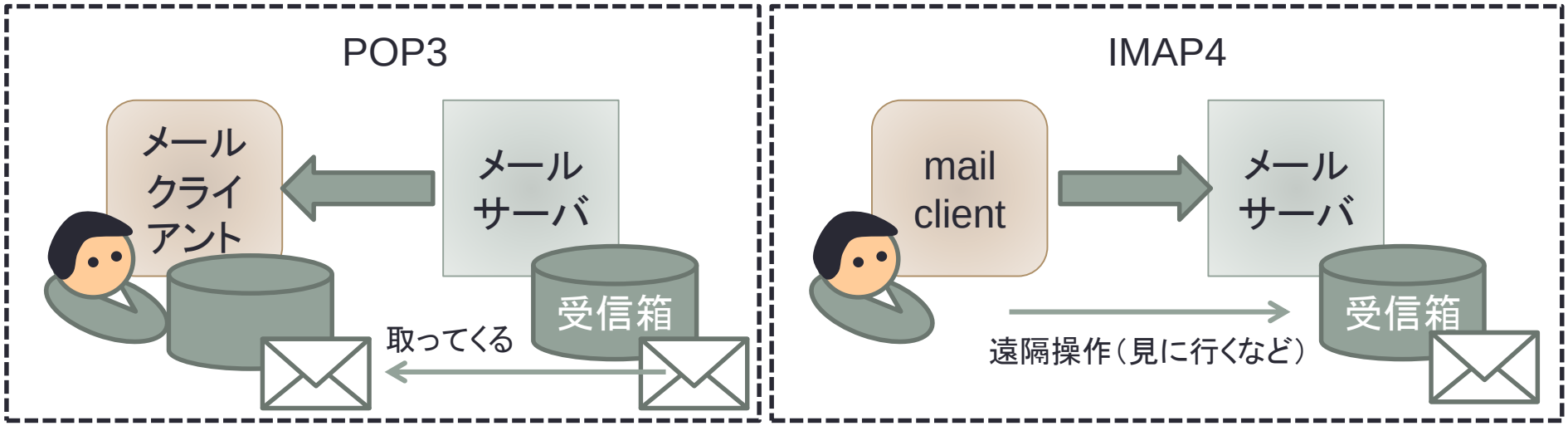




電子メールの受信

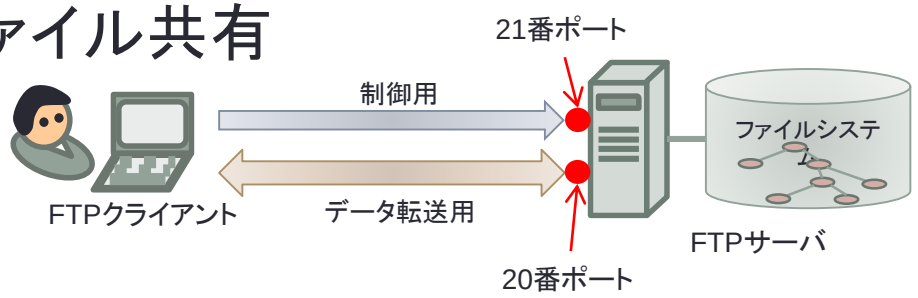
- メールボックスを直接見る
- リモートなメールボックスからメールを取ってきて見る

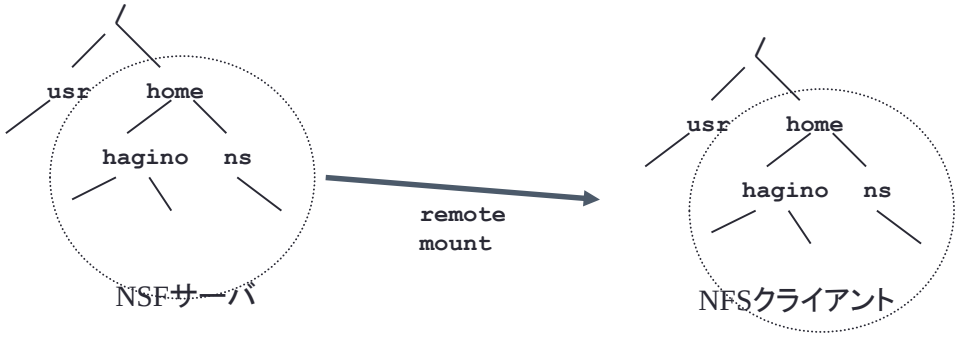
- POP3
  - Post Office Protocol
  - メールをMUAに取り込むプロトコル
  - クライアントがメールを持つ
  - メールをサーバに残しておき他のMUAと共有することも可能
- IMAP4
  - Internet Message Access Protocol
  - メールボックスを操作するプロトコル
  - MUAはメールのキャッシュを持つだけ
  - 複数のMUAと共有可能
  - クライアントはキャッシュを持つだけ



第11回　分散ファイルシステム

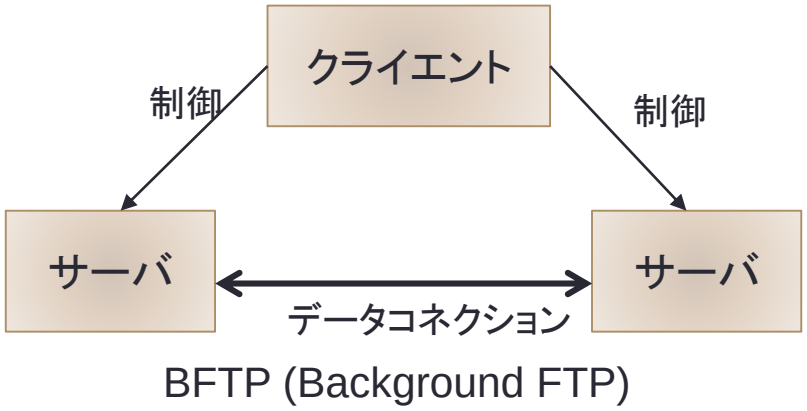
- オンラインストレージによるファイル共有
  - Webインターフェイス
  - Dropbox, Google Drive, Sky Drive, iCloudなど
  - 自動同期アプリケーションあり
- アプリケーションによるファイル共有
  - FTP
- OSによるファイル共有
  - 分散ファイルシステム
  - NFS
  - AFS



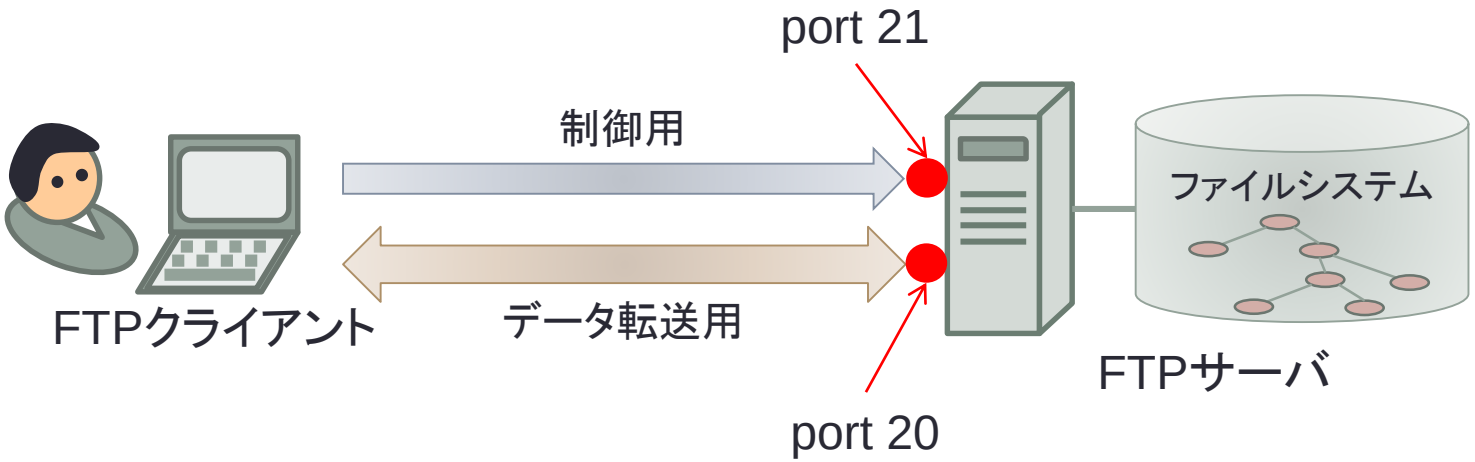


FTP

- File Transfer Protocol
  - インターネットの初期から存在する
  - TCPポート20と21
- クライアントサーバモデル
- 2本のコネクションを使用
  - FTPコマンドと応答用 （制御コネクション）
  - データ転送用 （データコネクション）

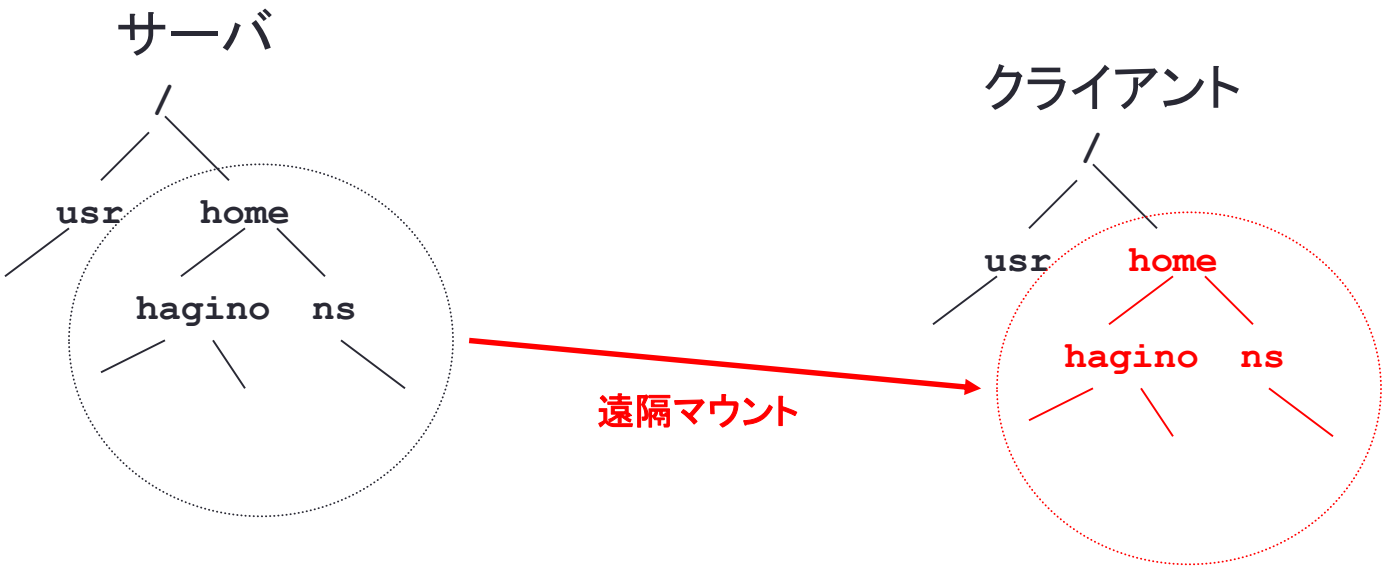


BFTP (Background FTP)



NFS (Network File System)

- UNIXにおける分散ファイル共有のプロトコル
  - Sun Microsystemsが開発
  - AT&Tが開発したRFS（Remote File Sharing）と一時期張り合うもNFSが生き残る
- 遠隔サーバのファイル木をローカルディスクのようにマウントし利用する
  - サーバはファイル木をexportする
- もともとはUDP上のプロトコル，現在はTCP上でも利用可能
  - ポート2049

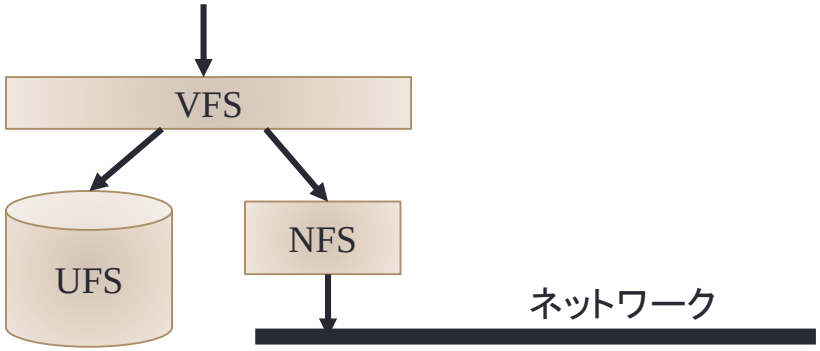


NFSの特徴と問題点

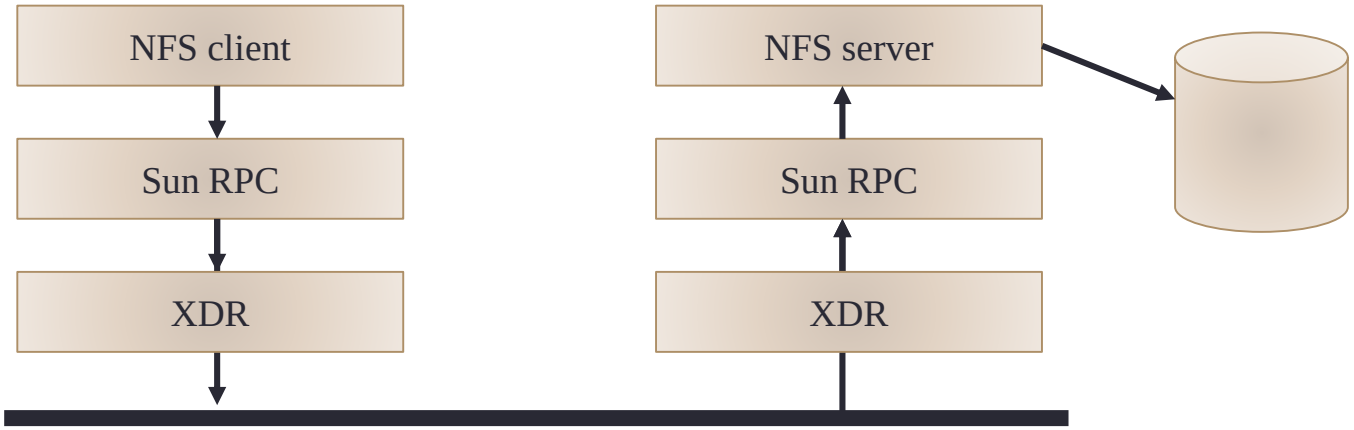
- アクセス透過性を持つ
  - 分散ファイルシステムだから当然
- 位置透過性を持つ
  - 名前空間はローカル
  - 任意の場所にマウント
- 障害透過性を持つ
  - 状態を保持しない
  - ほとんどの処理は再試行可能
- 性能透過性の確保
  - クライアントはデータをキャッシュする
- 複製透過性がない
  - サーバの複製は作ることができない
- 並行透過性がない
  - ファイルをロックすることができない
- 規模透過性はあまり考えない
  - 組織内での共有が主
  - ユーザを統一する必要がある
  - ユーザ名の共有はNIS(YP)などを利用

NFSの実装

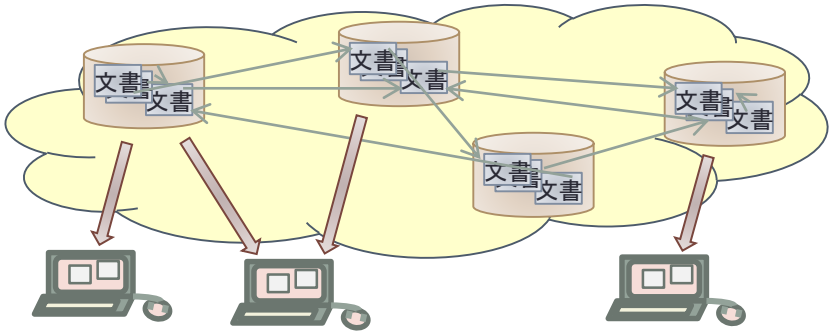
- 仮想ファイルシステム(VFS)によりローカルと遠隔を切り分ける



- Sun RPCを利用
  - XDR (External Data Representation)を使いメッセージを表現

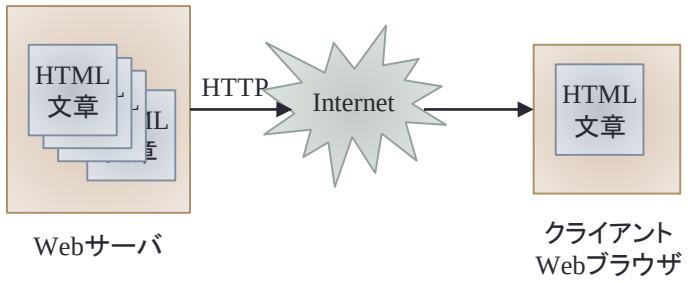


第12回　World Wide Web

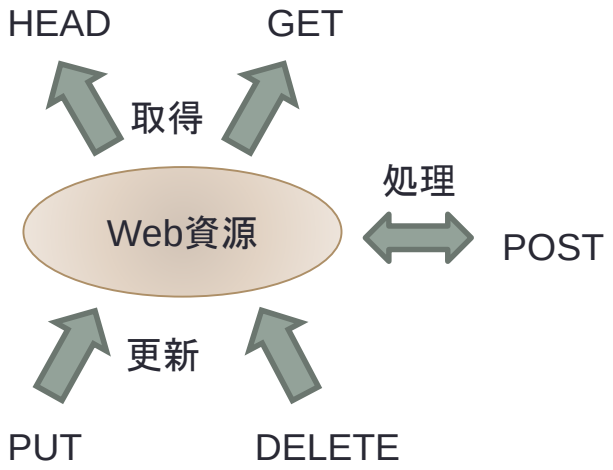


- World Wide Webとは
  - インターネット+ハイパーテキスト

- WWWの3つの発明
  - 文章をハイパーテキストとしてHTMLで記述
  - 文章の場所をURLで指定
  - HTTPによりサーバからブラウザに転送



- HTTP
  - Anonymous FTPを単純化したもの
  - マルチメディアに対応
  - 言語ネゴシエーション
  - GET vs POST



URIのシンタックス

http://www.sfc.keio.ac.jp/teacher/hagino.html?title=web#lecture

スキーマ　オーソリティ　パス　問い合わせ　フラグメント

- スキーマ(Schema)
  - URIの種類
  - プロトコル
- オーソリティ(Authority)
  - ホスト名
  - サーバ名
- パス(Path)
  - オーソリティ内の位置
  - ファイル名
- 問い合わせ(Query)
  - 検索などの検索語
- フラグメント(Fragment)
  - 文書内の位置

httpでは問い合わせまでWebサーバ，フラグメントはブラウザが処理

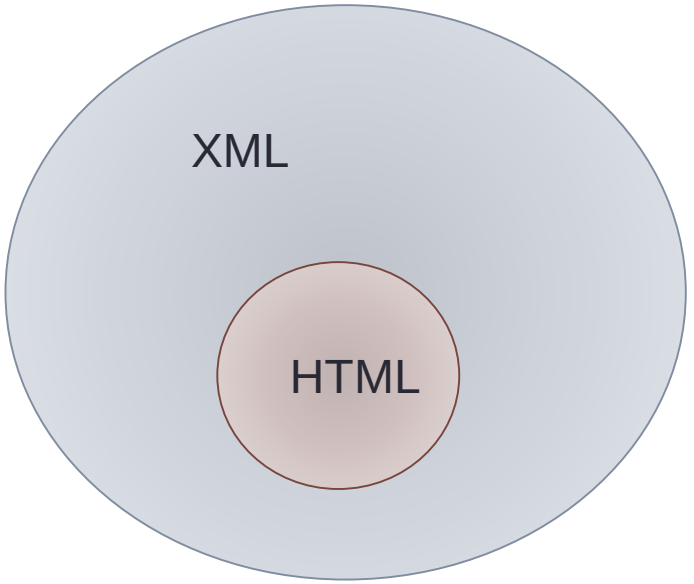
URIの公理

- 普遍性(Universality)
  - すべてのWeb資源はURIを持つ
- 大域性（Global Scope)
  - URIはどこでも同じ意味を持つ
  - 一意性
- 同一性（Sameness)
  - URIは常に同じものを意味する
  - 意味が同じであり，内容は異なることもある
- 不透明性（Opacity)
  - URIだけから資源の種類を知ることはできない
  - 資源の表現を調べないと種類等はわからない

HTML

- HTML
  - XML(SGML)アプリケーション
  - ハイパーテキスト文書

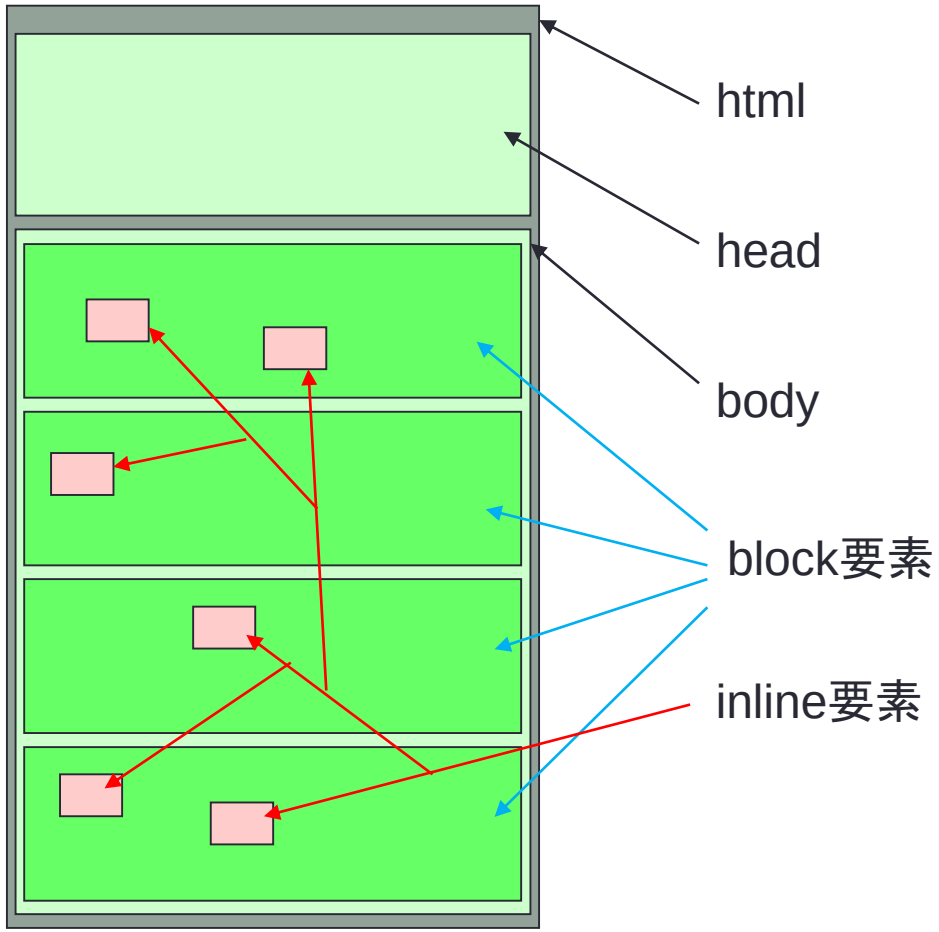
- HTMLの特徴
  - 内容と表現の分離
  - スタイルはCSSで指定
  - 直交する技術を用いる
    - 内容: HTML
    - スタイル: CSS
    - プログラミング: Javascript



直交技術　ビデオ
- **映像**
- **音声**
- **サブタイトル**

HTMLの要素の分類

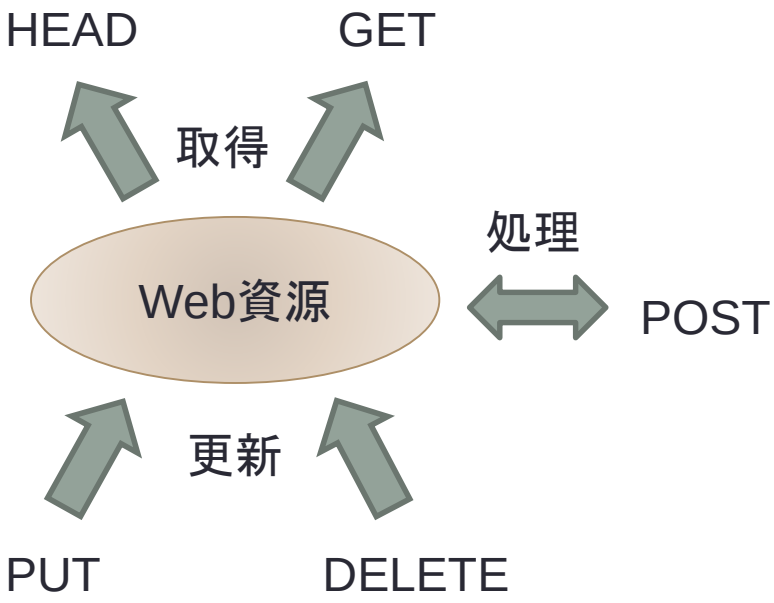
- 文書全体を構成する要素
  - html, head, bodyなど
  - section, articleなど
- 段落を構成する要素
  - block要素
  - h1, h2, ul, ol, tableなど
- 文の中で用いられる要素
  - inline要素(text要素)
  - i, b, em, strongなど



HTTP (Hypertext Transfer Protocol)

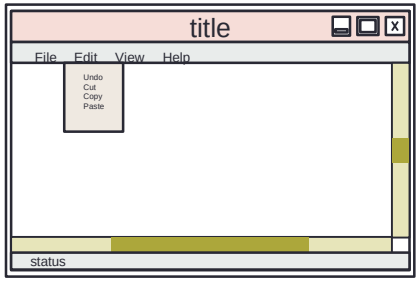
- Web資源を操作するプロトコル

- 主なメソッド
  - HEAD
    - 資源の情報を取得する
  - GET
    - 資源の表現を取得する
  - PUT
    - 資源の作成または更新
  - DELETE
    - 資源の削除
  - POST
    - データを資源に送り処理する

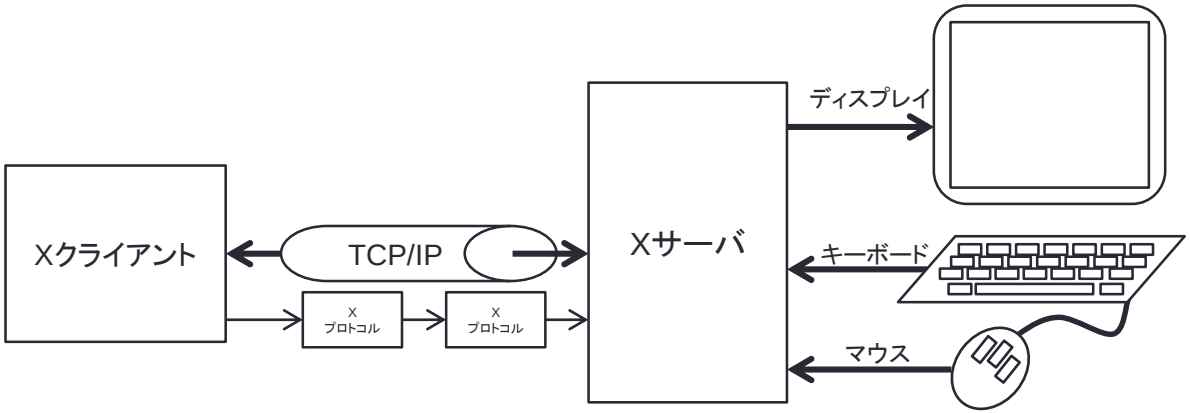


第13回　ウインドウシステム

- ウインドウシステム
  - X Window System
  - Windows
  - Quartz Composer

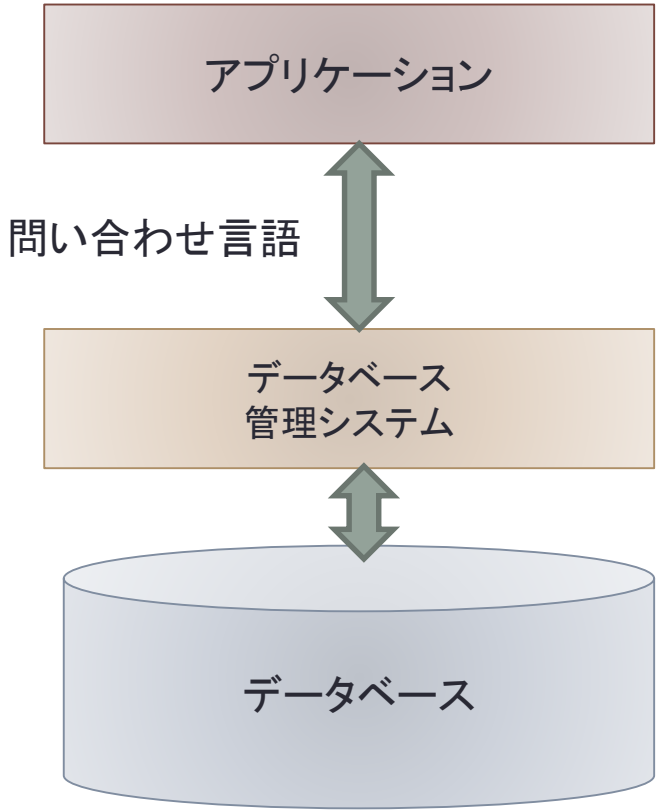


- Xウインドウシステム
  - サーバクライアント
  - TCP/IP対応



第14回　データベースシステム

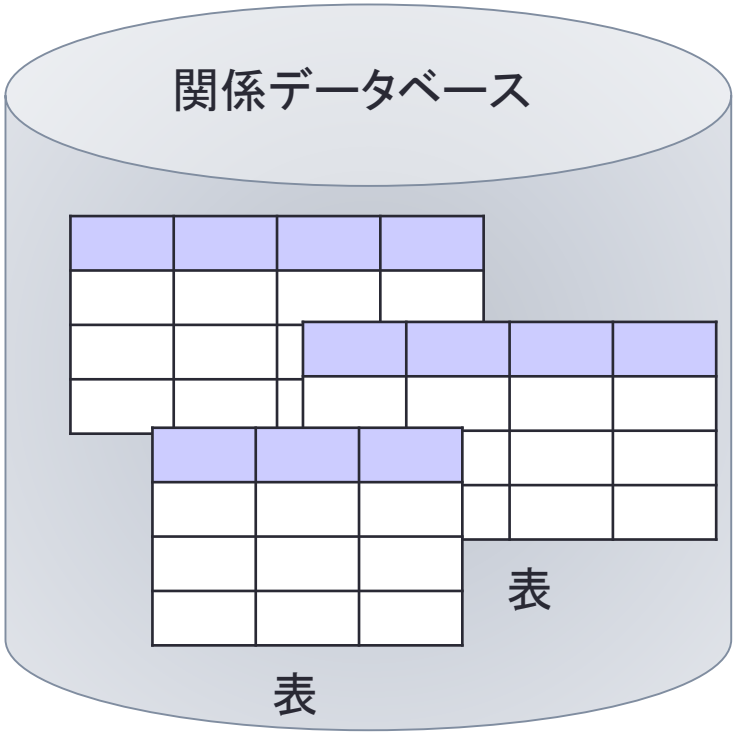
- データベース
  - データベース管理システム
- 関係データベース
  - 関係演算
  - 基本演算（制限，射影，結合）
  - SQL
- ロック
  - 並行処理
  - デッドロック



関係データベース

データベースモデル
- 階層モデル
- ネットワークモデル
- 関係モデル

- 関係モデルに基づくデータベース
  - 関係 = 表
  - RDB = 表の集まり
- 関係演算により関係（表）を操作
  - 和（union）
  - 差（difference）
  - 交わり（intersection）
  - 制限（restriction, selection）
  - 射影（projection）
  - 結合（join）
- SQL
  - 関係演算に基づくRDBMS問い合わせ言語
    - INSERT
    - UPDATE
    - DELETE
    - SELECT



デッドロック

- 複数人が互いにロックすることにより先に進めなくなる
  - データベース以外でも並行処理では同じことが起こる

- デッドロックの検出
  - 依存関係を調べてデッドロックしていることを見つける
  - ロールバックさせて，再度実行する
- デッドロックしないためには
  - 同じ順でテーブル（資源）をロックする