分散システム

第4回 通信

双見 京介（FUTAMI Kyosuke）
高田秀志（TAKADA Hideyuki）
2025年10月

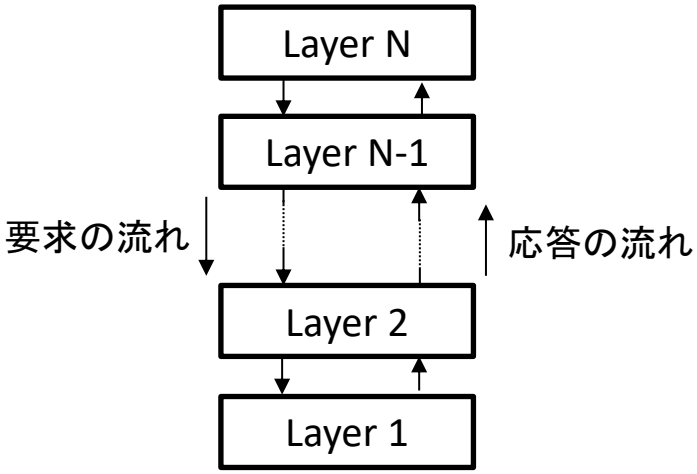
通信技術の発展

- 有線通信から無線通信へ
  - 固定電話から携帯電話へ
  - イーサネットから無線LANへ
  - ICタグの登場（鉄道乗車用パスなど）
- さらなる高速化
  - 10メガイーサネットから10ギガイーサネットへ
  - 無線LANはIEEE 802.11b（11Mbps）からIEEE 802.11ax（9.6Gbps）へ
  - 携帯電話網は5G（第5世代移動通信システム）へ
- 異機種間相互接続
  - 様々な種類の機器を相互に接続（インターネットプロトコル）
  - クラウドコンピューティング，エッジコンピューティング，センサネットワーク

分散システムにおける通信技術の捉え方

- コンピュータは「データを処理し，伝達する機械」
- 分散システムは，このような機械が要素（コンポーネント）として複数動作しているもの
- 通信技術により，コンポーネント間のデータの伝達を実現
- 様々な通信技術により実現されるデータの伝達の仕組みを抽象化して扱う
- 抽象化された仕組みは，様々な用途に利用可能

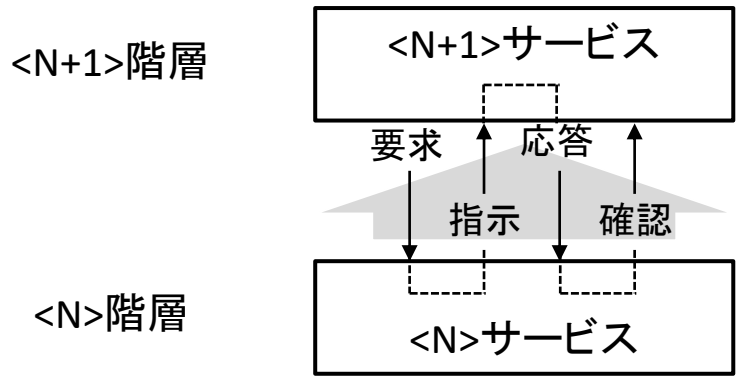
通信機能の階層化



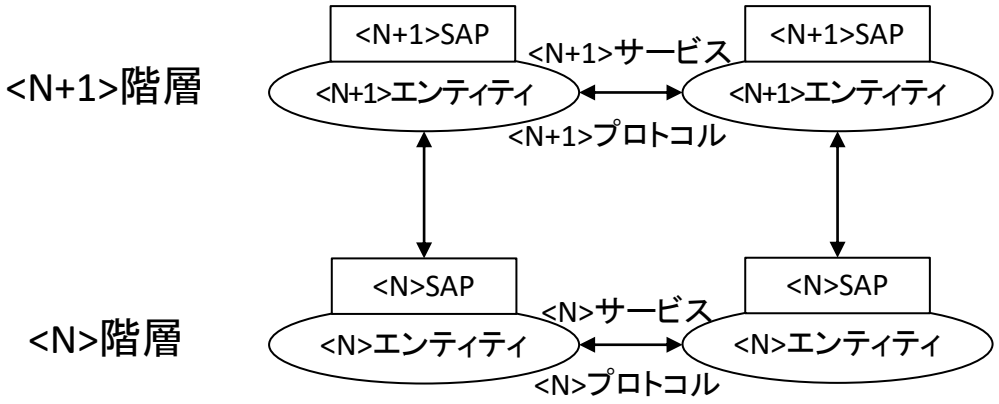
階層化アーキテクチャ
（Layered architecture）

- レイヤ L_i のコンポーネントはレイヤ $L_{i\text{-}1}$ のコンポーネントを利用する
- レイヤ L_i はレイヤ $L_{i\text{-}1}$ の機能を利用して要求を発行し，応答を受け取る
- レイヤ L_i はレイヤ $L_{i\text{-}1}$ より下のレイヤについては関知しない
- 例えば，Webブラウザは，HTTPというレイヤのみに依存し，実際の接続が無線LANなのか携帯電話網なのかなどについて関知しない

サービスとプロトコル



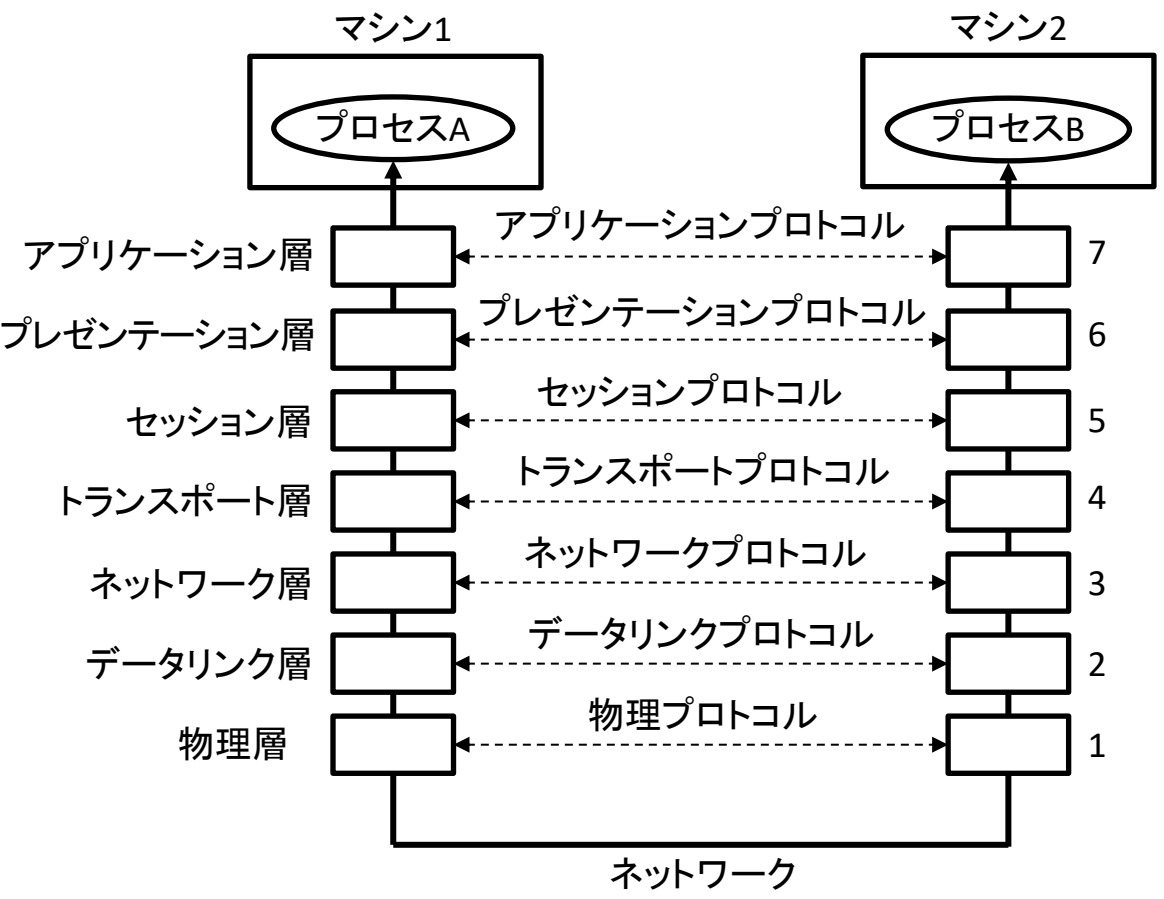
- <N+1>階層は，直下の<N>サービスを利用して<N+1>サービスが提供する機能を実現
- 階層間ではサービスプリミティブ（要求・指示・応答・確認）により相互に作用



- <N>階層の目的とする機能を実現するための通信規約を<N>プロトコルと呼ぶ
- <N>プロトコルの処理を行う主体を<N>エンティティと呼ぶ
- <N+1>エンティティは<N>SAPを利用して<N>エンティティにアクセスする

SAP: Service Access Point

OSI参照モデル
(Open System Interconnection Basic Reference Model)

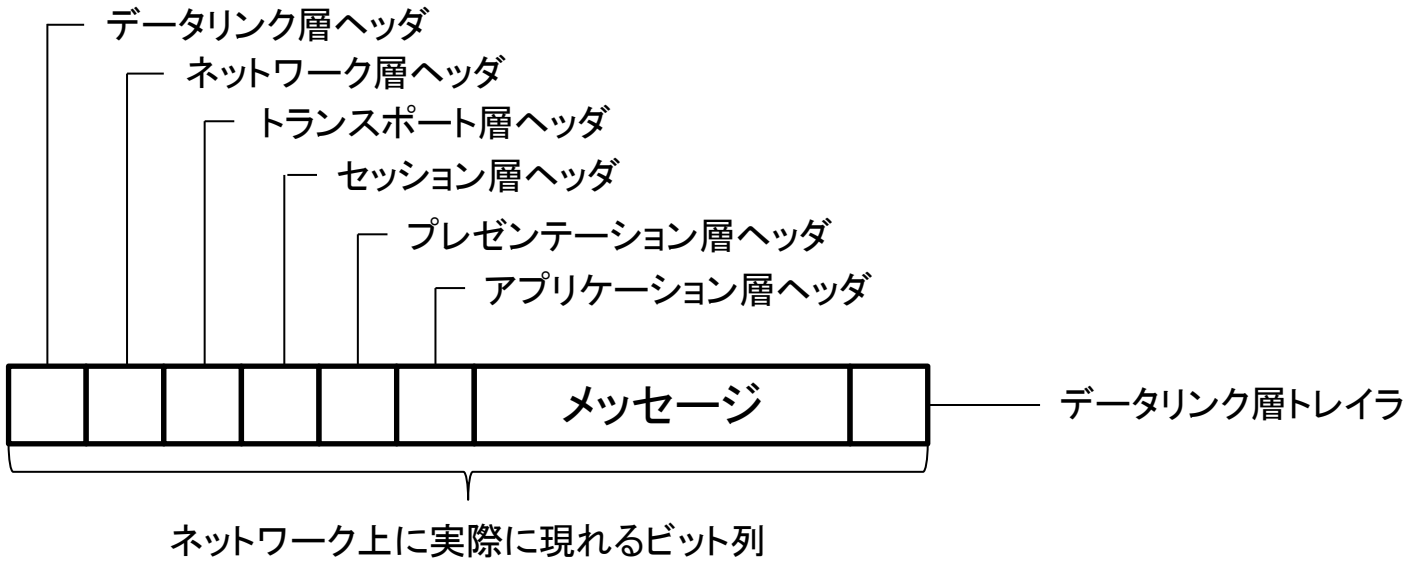


- マシン1のプロセスAとマシン2のプロセスBは送信されるビット列の意味について合意する必要がある
- OSIモデルでは，通信が7階層に分割される
- 各階層は，その上の階層にインタフェースを提供する
- インタフェースは操作の集合から構成され，ユーザに対してサービスを提供する

OSI参照モデルにおける各階層の役割

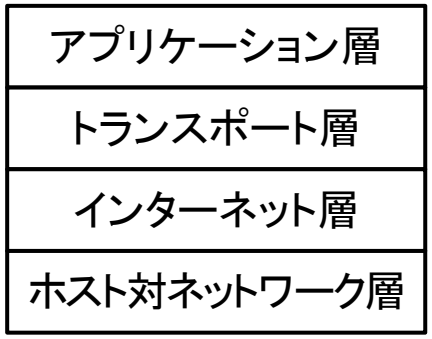
1. 物理層
   - ビット列の伝送を行うための機械的・電気的規約
   - 送受信タイミング，片方向/半二重/全二重伝送などの取り決め
2. データリンク層
   - ビット列のフレーム化
   - フレームの順序制御，誤り制御
3. ネットワーク層
   - ネットワークアドレス処理
   - 中継処理
4. トランスポート層
   - エンドシステム間の転送機能
   - パケットの紛失や順序誤りに対する制御
5. セッション層
   - アプリケーション間の情報の流れの同期を制御
6. プレゼンテーション層
   - アプリケーション間で相互に理解可能なデータ表現の提供
   - データ型変換，暗号化，圧縮
7. アプリケーション層
   - エンドユーザにサービスを提供
   - 電子メール，ファイル転送等

階層化ネットワークにおけるメッセージ



- マシン1のプロセスAはメッセージを構成し，アプリケーション層に渡す
- アプリケーション層はメッセージの最初に「ヘッダ」を追加し，それをプレゼンテーション層に渡す
- メッセージがマシン2に到着すると，それぞれの階層でヘッダを外しながら，上の階層へ渡していく
- 最終的に，送られたメッセージがマシン2のプロセスBに届く

TCP/IP参照モデル



- 特定のベンダに依存しないオープンなプロトコル
- 各階層の役割
  1. ホスト対ネットワーク層
     - コンピュータとネットワークのインタフェース
     - イーサネット，各種広域ネットワークなど
  2. インターネット層
     - IPアドレスの規定
     - パケット転送サービスの提供
  3. トランスポート層
     - プロセス間の通信を規定
     - コネクション型のTCP，コネクションレス型のUDP
  4. アプリケーション層
     - ユーザに対するサービスに関わる規定
     - ファイル転送（FTP），メール配信（SMTP），Web（HTTP）など

プロトコルの例

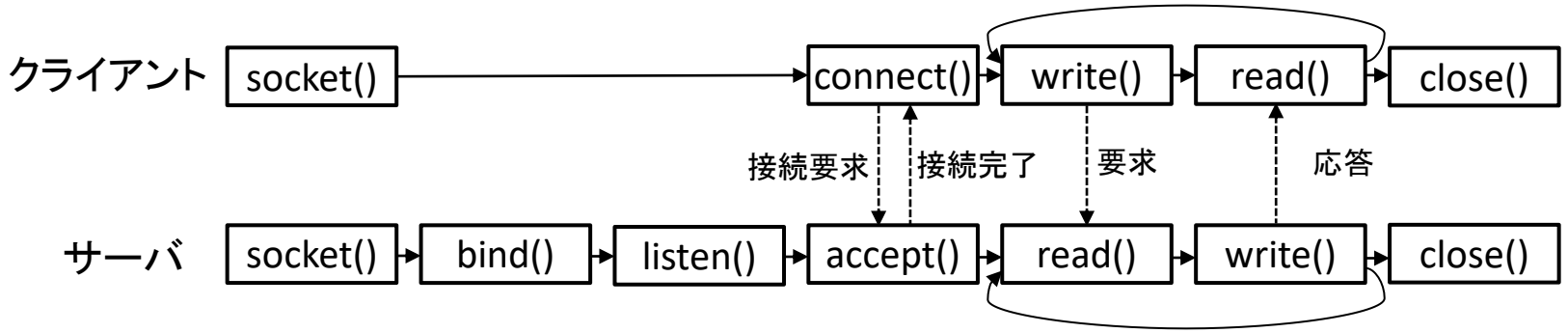
- 下位層
  - Ethernet, Wi-fi, 5G
  - IP (Internet Protocol)
- トランスポート層プロトコル
  - TCP (Transmission Control Protocol)
  - UDP (Universal Datagram Protocol)
  - RTP (Real-time Transport Protocol)
- 上位層
  - HTTP (HyperText Transfer Protocol)
  - FTP (File Transfer Protocol)

メッセージ指向通信

- 一時（Transient）通信
  - **ソケット通信**
  - アプリケーションは，ソケットと呼ばれるエンドポイントに対してデータの読み書きを行うことにより，ネットワーク通信を行うことができる
  - 送信者と受信者が同時に動作していることが必要
- 永続（Persistent）通信
  - **メッセージキューイングシステム**
  - 送信者と受信者の中間に記憶領域を配置することにより，両者が同時に動作していることを必要としない

ソケット通信

- Berkeley UNIXで導入されたプロセス間通信の枠組み
- ファイルの読み出し（read）と書き込み（write）の概念を用いて，プロセス間通信を抽象化
  - ファイルが通信路のイメージ
  - 片方のプロセスがwriteしたデータを，もう一方のプロセスがreadする
- ソケットを利用するためのシステムコールを用いてプロセス間通信を実現



Pythonによるソケット通信

クライアント

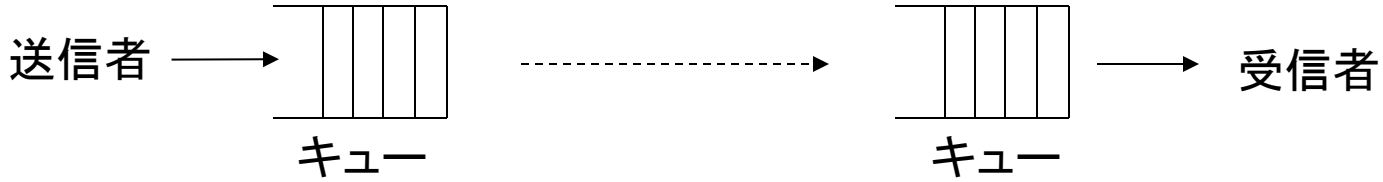
```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM);
s.connect(('192.168.1.1', 3000))
s.sendall(b'Hello')
data = s.recv(1024)
print(data)
s.close()
```

サーバ

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM);
s.bind(('192.168.1.1', 3000))
s.listen(5)

conn, addr = s.accept()
data = conn.recv(1024)
print('data: {}, addr: {}'.format(data,addr))

conn.sendall(b'Received: ' + data)

s.close()
```

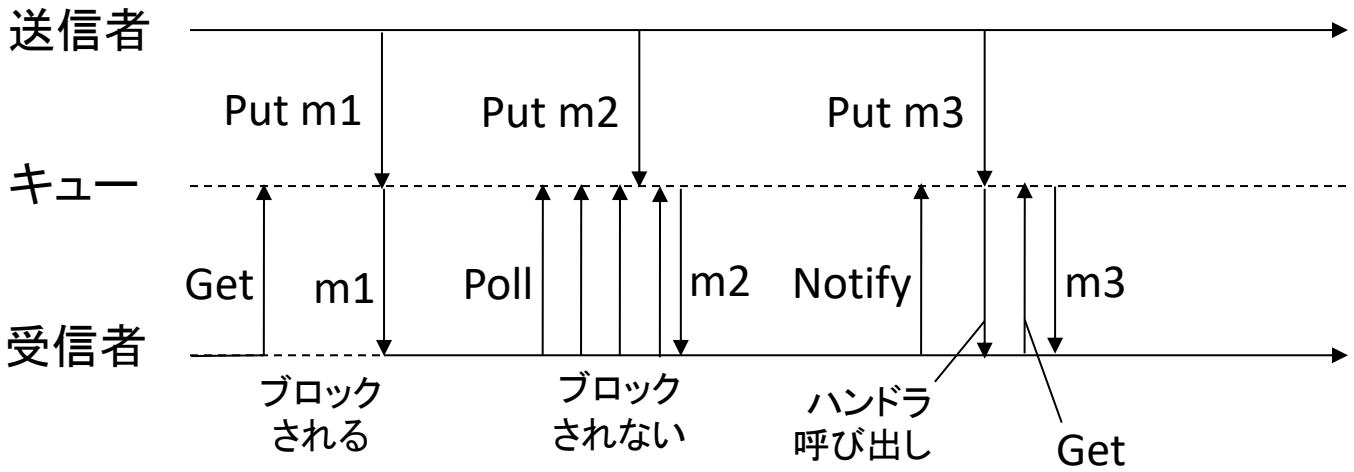
- 単純化のため，本来必要な例外処理は省いている
- C言語によるソケット通信プログラムは教科書を参照のこと

メッセージキューイングシステム



キューに対する基本インタフェース

Put: キューにメッセージを追加
Get: キューにメッセージが入るまでブロック，最初のメッセージを取り出して削除
Poll: キューをチェックし，メッセージがあれば取り出して削除，ブロックしない
Notify: キューにメッセージが入ったときに呼び出されるハンドラを設定

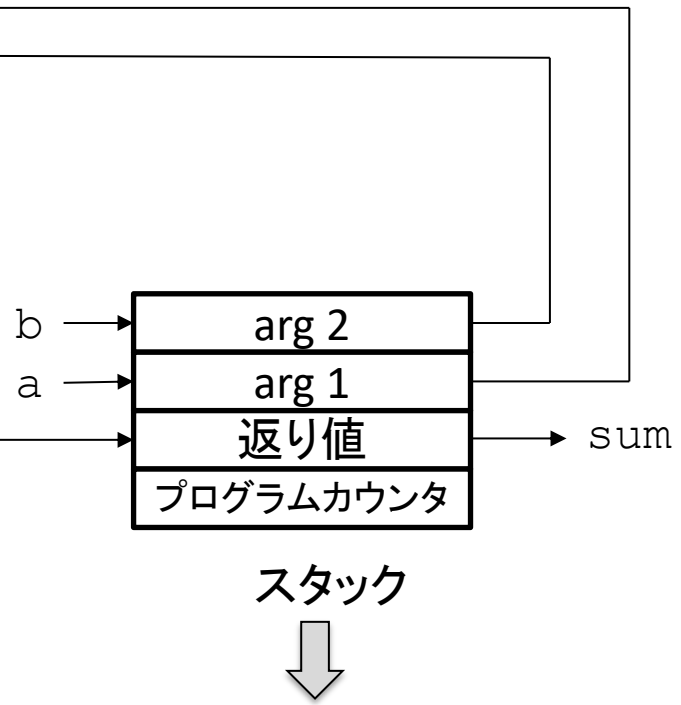


遠隔手続き呼び出し（Remote Procedure Call, RPC）

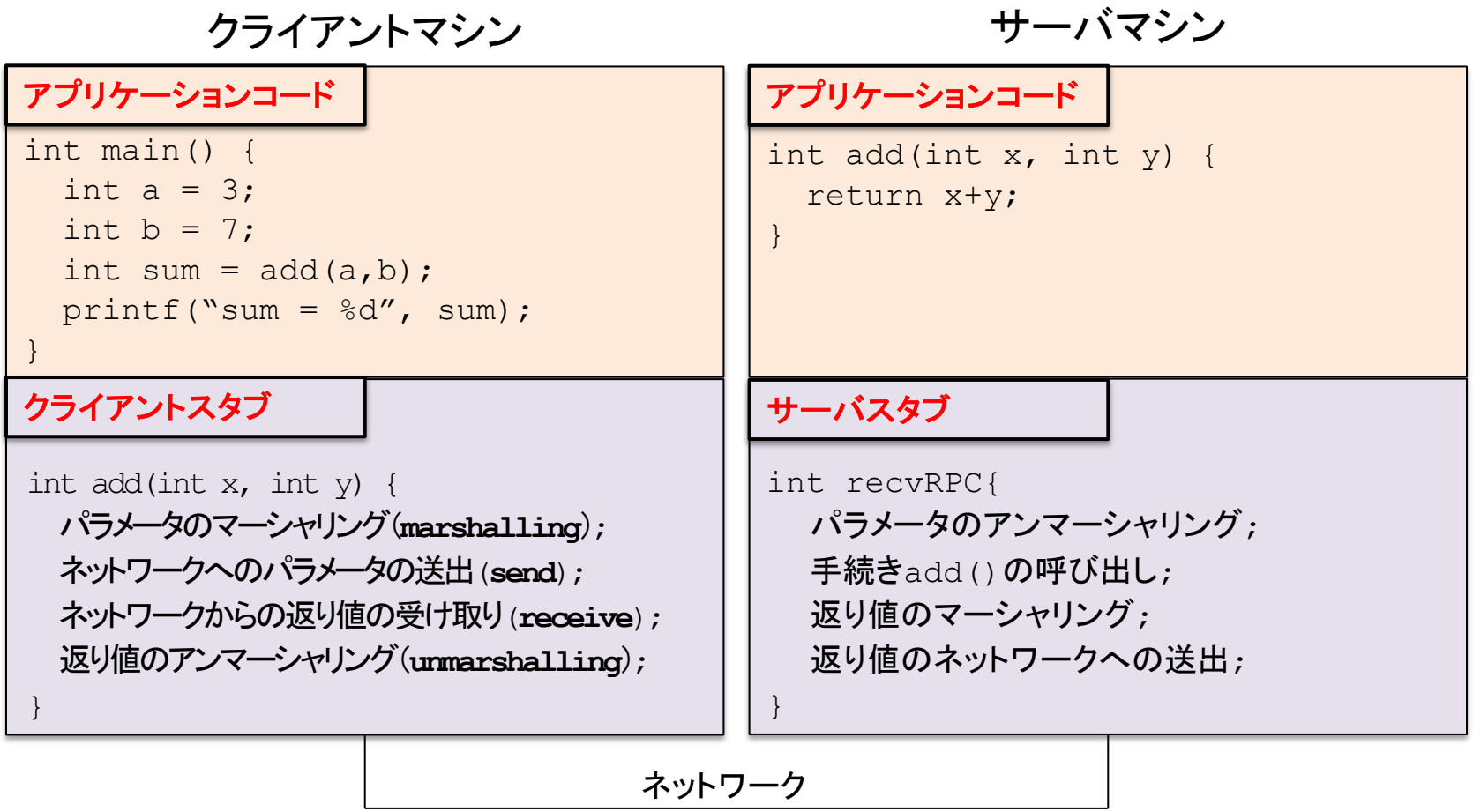
- プロセス間のメッセージ交換のための「Send手続き」と「Receive手続き」では透過性を提供しない
- RPCは他のマシンに存在する手続き（C言語で言う関数）を呼び出すことを可能にする
  - マシンAのプロセスがマシンBの手続きを呼び出すと，マシンA上の呼び出し側のプロセスが一旦停止し，呼び出された手続きがマシンB上で実行される
  - 呼び出した側と呼び出された側がパラメータや返り値により情報を交換する
- メッセージのやりとりはプログラマには見えない

通常の手続き呼び出し

```
int add(int x, int y) {
  return x+y;
}

int main() {
  int a = 3;
  int b = 7;
  int sum = add(a,b);
  printf("sum = %d", sum);
}
```



RPCの原理



- sendやreceiveの呼び出しはアプリケーションコードには見えず，通常の関数呼び出しに見える
- クライアントスタブおよびサーバスタブのソースコードは，開発環境により自動生成される

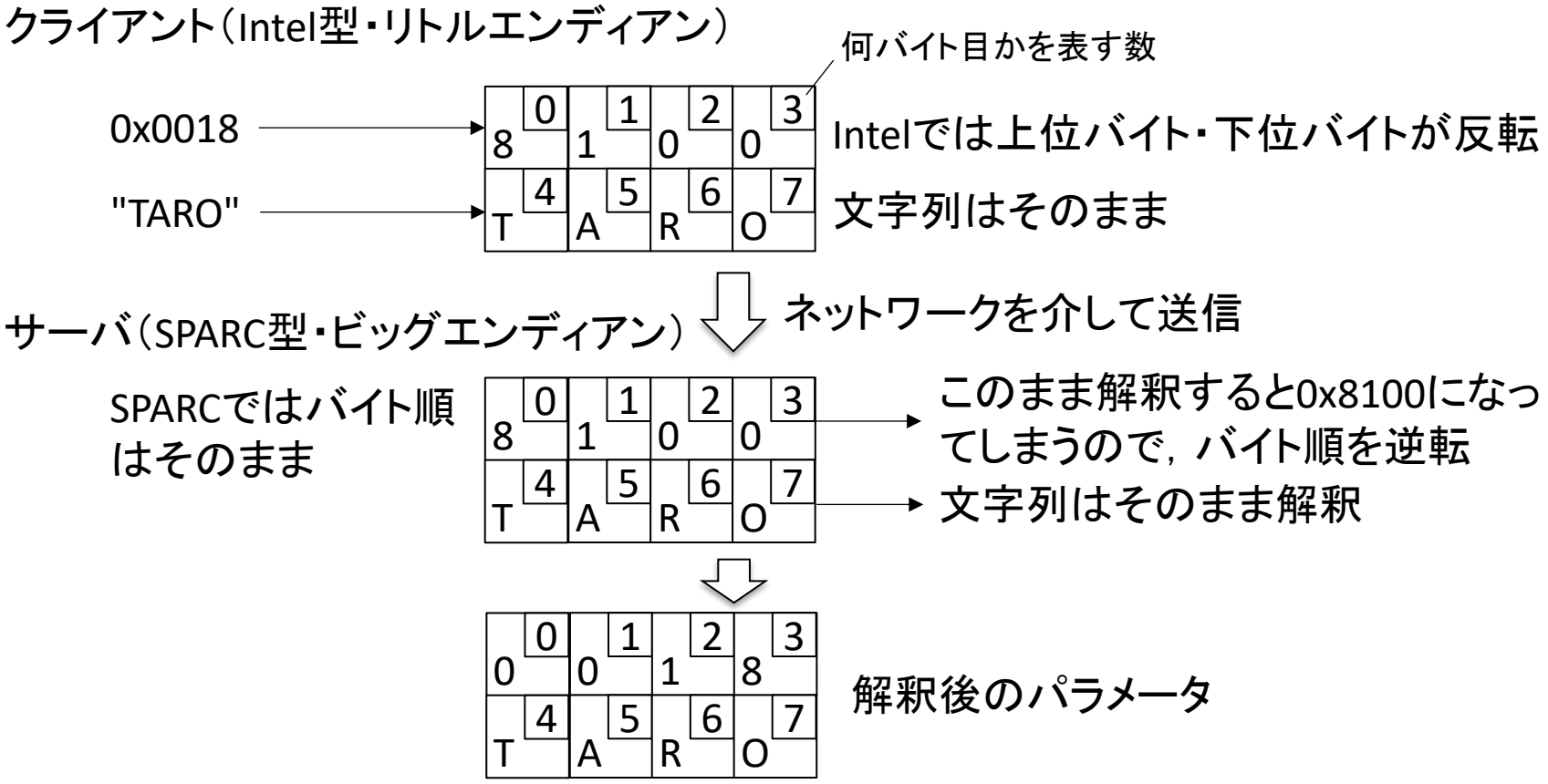
透過性の提供

RPCの実行手順

1. クライアント側の手続きは，クライアントスタブを通常と同じように呼び出す
2. クライアントスタブはメッセージを構築し（マーシャリングし），OSのsend機能を呼び出す
3. クライアント側のOSはサーバ側のOSへメッセージを送信する
4. サーバ側のOSはメッセージをサーバスタブへ渡す
5. サーバスタブはメッセージを取り出し（アンマーシャリングし），サーバ側の手続きを呼び出す
6. サーバ側の手続きは処理を実行し，結果をサーバスタブに返す
7. サーバスタブは返り値をメッセージの中に格納し，OSのsend機能を呼び出す
8. サーバ側のOSはクライアント側のOSへメッセージを送信する
9. クライアント側のOSはメッセージをクライアントスタブに渡す
10. クライアントスタブは返り値を取り出し，クライアント側の手続きに返す

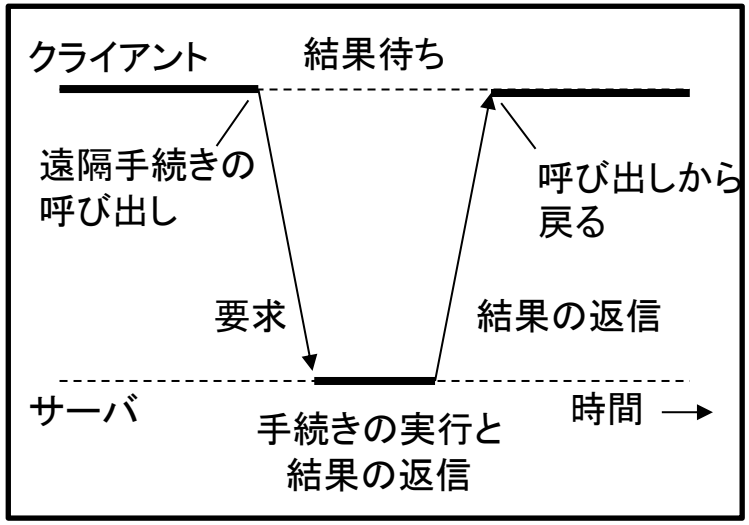
パラメータ渡し

0x0018という整数値と"TARO"という文字列をパラメータとして
クライアントからサーバへ送る場合



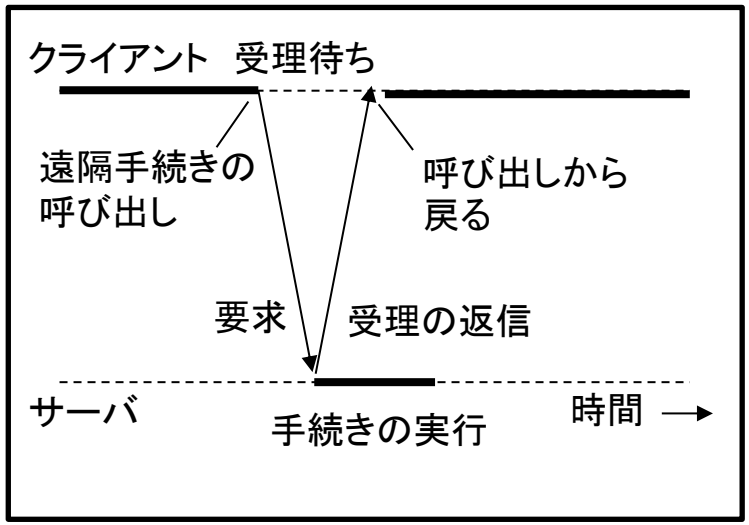
RPCの種別（1）

同期型（Synchronous）RPC



- クライアントプロセスは，遠隔手続きの呼び出し後，待ち状態へ入る（ブロックされる）
- サーバプロセスは，手続きの実行が終了後，結果をクライアントプロセスに返す
- 「サーバが手続きを実行中，クライアントは動作を停止している」という意味で「同期型」

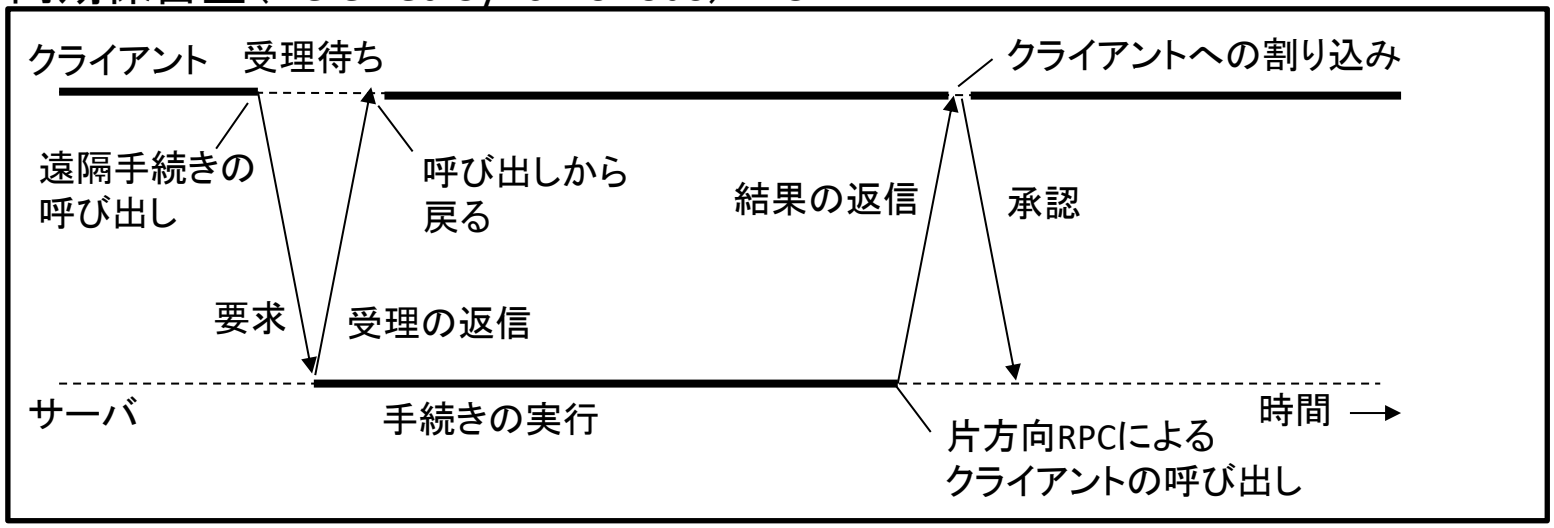
非同期型（Asynchronous）RPC



- サーバプロセスは，要求を受理したことを手続きの実行前にクライアントプロセスに返す（同期型に比べてクライアントがブロックされる時間は短くなる）
- 手続きの実行はクライアントへの返信後に行われるので，結果をクライアントへ返すことはできない
- 「サーバが手続きを実行中でも，クライアントは別の処理を行える」という意味で「非同期型」

RPCの種別（2）

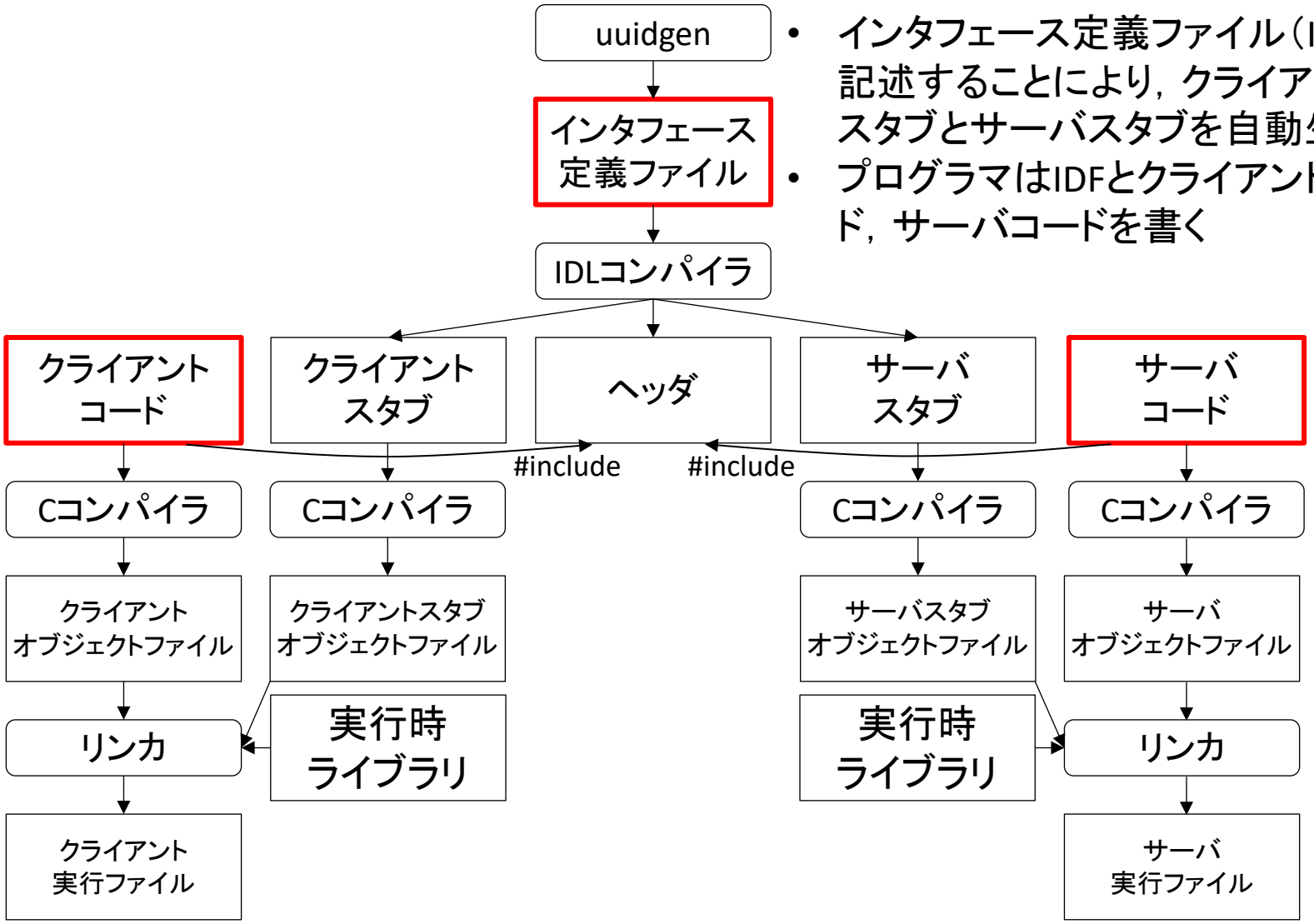
同期保留型（Deferred Synchronous）RPC



- サーバプロセスは，要求を受理したことを手続きの実行前にクライアントプロセスに返すという点では非同期型と同じ
- 手続きの実行後，サーバプロセスはクライアントへの割り込み処理によって手続きの実行結果を返す（非同期型のクライアントとサーバを逆転させたものと同じ）
- 「サーバからクライアントへ手続き実行の結果を返す」という意味では「同期型」であるが，一旦受理通知だけを返信し，結果の返信は手続き実行が終了するまで「保留」する

OSF: Open Software Foundation
DCE: Distributed Computing Environment

OSF DCEによるRPCの実装



- インタフェース定義ファイル（IDF）を記述することにより，クライアントスタブとサーバスタブを自動生成
- プログラマはIDFとクライアントコード，サーバコードを書く