ネットワーク通信と RPC (遠隔手続き呼び出し)

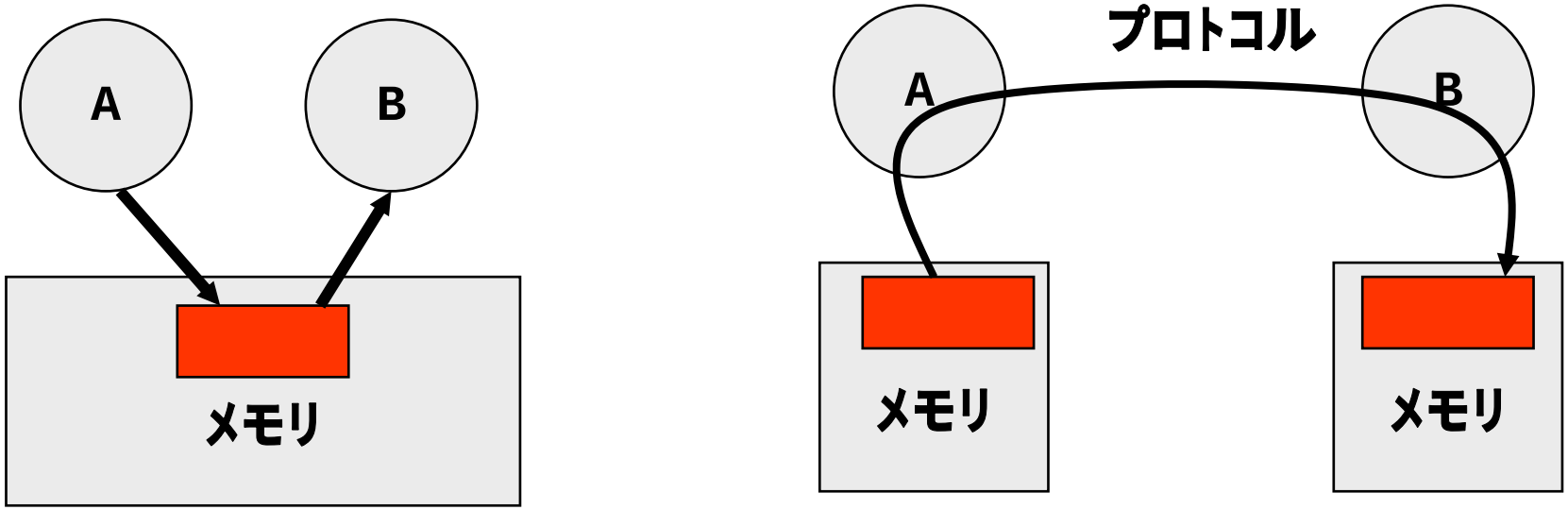
佐藤

分散システムの通信

- 通信レイヤとプロトコル
- Remote Procedure Call (RPC: 遠隔手続き呼び出し)
- Message-Oriented Middleware (MOM)
- data-streaming

異なるコンピュータのプロセス間のデータの交換

- 共有メモリがない場合には、データを明示的に送らなくてはならない
- 取り決め=プロトコル(protocol)



Connection

- protocol = 通信のための取り決め
- connection oriented protocol
  - 通信を始めるために、明示的に通信路を確保する手続きをするプロトコル
  - TCP (Transmission Control Protocol)
- connectionless protocol
  - データを送るときに相手を指定するプロトコル。事前の手続きは必要ない
  - UDP (Universal Datagram Protocol)

OSIの7層モデル

- OSI model, ISO OSI
  - International Standards Organization (ISO)が決めたモデルで、Open System Interconnection (OSI) Reference Modelと呼ばれる

- 7つのレイヤ(層）がある
  - Application protocol
  - Presentation protocol
  - Session protocol
  - Transport protocol
  - Network protocol
  - Data Link protocol
  - Physical protocol

- 下の層を使って、上の層が定義されている

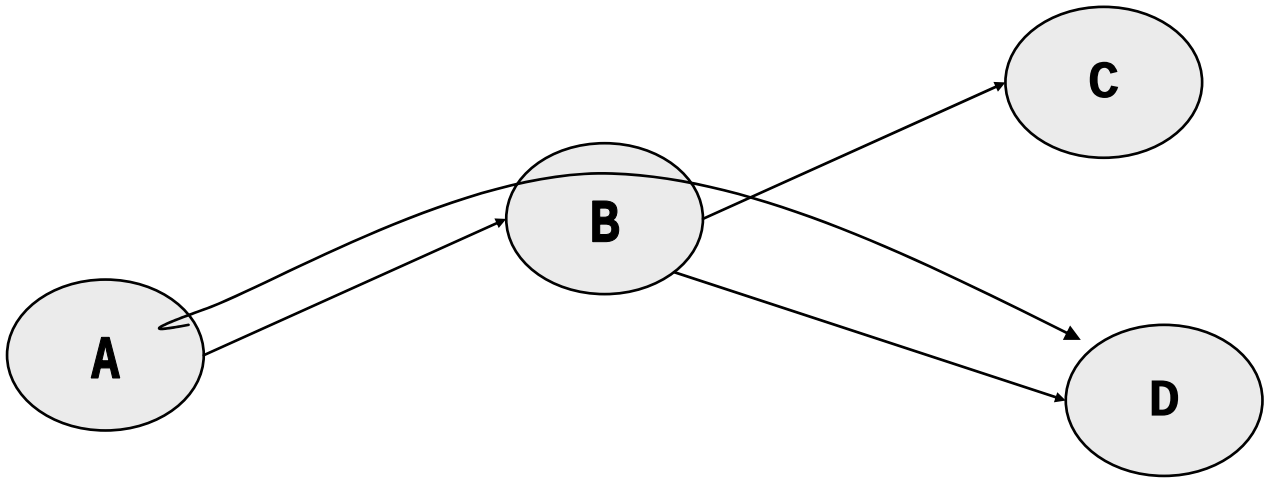
Application 層
Presentation 層
Session 層
Transport 層
Network 層
Data Link 層
物理層

Lower-Level Protocol

- 物理層（Physical Layer)
  - どのように、0と1の信号を伝えるか
  - イーサネットethernetの物理層
  - RS-232C、シリアルインターフェース
- 送信されるデータのかたまりをframeを呼ぶ
- データリンク層（Data Link Layer)
  - ノードからノードへ、(直接）frameを伝える仕組み
  - checksum - エラーのチェックの仕組み
  - ethernetのhub, PPP(point-to-point protocol), …
  - MACアドレス：Media Access Control Address, データリンク層の”アドレス”

Lower-Level Protocol

- Network Layer
  - データフレームを配送する仕組みを提供する
  - routing
  - IP: Internet protocolが有名
    - IP addressで、routingを行う



Transport Protocol

- データ転送の信頼性を確保するための方式を定めたもの。
  - ネットワーク層を通して送られてきたデータの整序や誤り訂正、および再送要求などをおこなう。
- TCP (Transmission Control Protocol)
  - connection-oriented な通信で、信頼性を確保したstream通信を提供する
  - TCP/IPが現在の主要なプロトコル
- UDP (Universal Datagram Protocol)
  - connection-lessプロトコルで、データグラム単位の通信を提供、信頼性は保障しない。
- RTP (Real-time Transport Protocol)

Higher-level Protocol

- 4層(Transport layer)以上の定義は、明確ではない

- FTP: File Transfer Protocol
  - ファイルを転送するためのプロトコル
- HTTP: Hyper Text Transfer Protocol
  - webのページの転送のためのプロトコル

- POP: Post-office Protocol
- SMTP: Simple Mail Transfer protocol

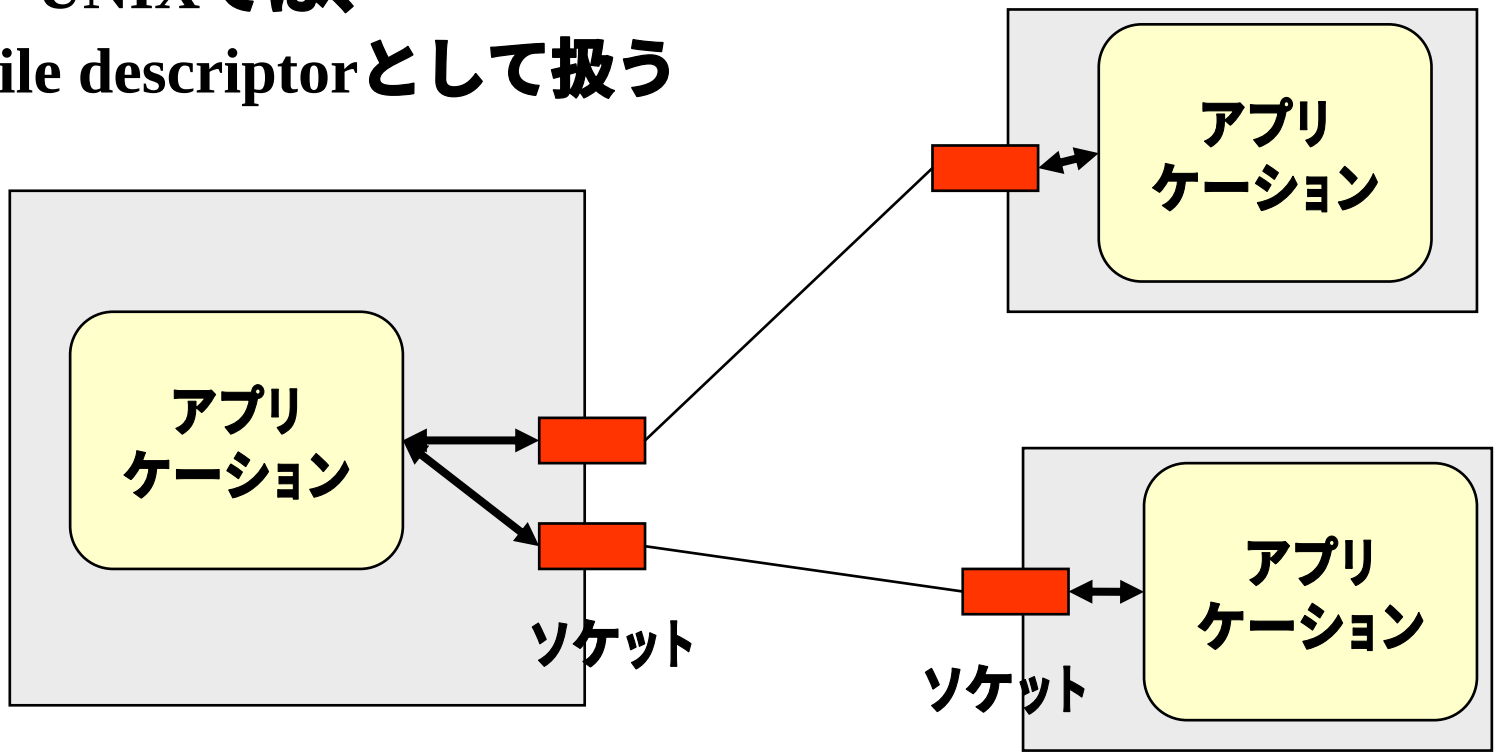
通信の分類

- persistent communication
  - メッセージが通信中にどこかで保持されること
- transient communication
  - メッセージが通信中に消えてしまう可能性がある通信

- asynchronous communication （非同期通信）
  - 送り手が、受け取りを待たずに、動作を始めること
- synchronous communication （同期通信）
  - 送り手が、相手が受け取るまでblockingして行う通信

ネットワークプログラミング

- socket　ソケット
  - TCPやUDPを行うためのend-point
  - UNIXでは、
    file descriptorとして扱う



ネットワークプログラミング：ＴＣＰ

- サーバー側

```
s = socket(); /* socketを作る*/
bind(s,address); /* 名前を与える */
listen(s,backlog); /* backlogの指定 */
ss = accept(s); /* connectionが発生したら
          新しいfile descriptorを返す */
close(s); /* 必要なければ、もとのsはclose */
recv(ss,…); /* read 開始 */
```

- クライアント側

```
s = socket(); /* socketを作る*/
connect(s,address); /* connectionする*/
send(s,…); /* send開始 */
```

Server側(1)

```
// 省略
int my_fd;
struct sockaddr_in my_sin;
static int _setup_server_socket(struct sockaddr_in *sinp,
                                int port, int backlog);
int main(int argc,char *argv[])
{
    int sinlen;
    struct sockaddr_in client_sin;
    char buf[128];
    int r,s;
    int port;
    if(argc != 2){
        fprintf(stderr,"%s #port¥n",argv[0]);
        exit(1);
    }
    port = atoi(argv[1]);
    printf("server test program ... wait on port %d¥n",port);
   my_fd = _setup_server_socket(&my_sin,port,1);
    sinlen = sizeof(struct sockaddr_in);
    s = accept(my_fd,(struct sockaddr *)&client_sin,&sinlen);
    if(s < 0){
        perror("accept failed");
        exit(1);
    }
    while((r = read(s,buf,128)) >= 0){
        write(1,buf,r);
    }
    printf("terminated ...¥n");
    close(s);
    close(my_fd);
    exit(0);
```

Server側 (2)

```
static int _setup_server_socket(struct sockaddr_in *sinp,int port,
       int backlog)
{
    int sinlen,r;
    struct sockaddr_in sin;
    char hostname[MAXHOSTNAMELEN];
    struct hostent *hp;
    int fd;

    fd = socket(AF_INET, SOCK_STREAM, 0);
    if(fd < 0){
        perror("socket failed");
        exit(1);
    }
   r = gethostname(hostname,MAXHOSTNAMELEN);
    if(r < 0){
        perror("hostname");
        exit(1);
    }
    printf("hostname=%s¥n",hostname);

    hp = gethostbyname(hostname);
    if(hp == NULL){
        perror("gethostbyname");
        exit(1);
    }
     memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_port = htons(port);
    bcopy(hp->h_addr,&sin.sin_addr.s_addr,hp->h_length);
```

Server側 (3)

```
    sinlen = sizeof(sin);

    r = bind(fd, (struct sockaddr *) & sin, sizeof(sin));
    if (r < 0){
        perror("bind");
        exit(1);
    }

    r = listen(fd,backlog);  /* set backlog */
    if (r < 0){
        perror("listen");
        exit(1);
    }

    r = getsockname(fd,(struct sockaddr *)sinp, &sinlen);
    if(r < 0){
        perror("getsockname");
        exit(1);
    }

    return fd;
}
```

Client側(1)

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>

#ifndef MAXHOSTNAMELEN
#define MAXHOSTNAMELEN 256
#endif

int main(int argc,char *argv[])
{
    int r;
    struct sockaddr_in sin;
    char hostname[MAXHOSTNAMELEN];
    struct hostent *hp;
    int fd,port;
    char buf[128];
     if(argc != 3){
        fprintf(stderr,"%s: hostname port¥n");
        exit(1);
    }
    strcpy(hostname,argv[1]);
    port = atoi(argv[2]);
    printf("client test ... connect to %s:%d¥n",hostname,port);
    hp = gethostbyname(hostname);
    if(hp == NULL){
        perror("gethostbyname");
        exit(1);
    }
```

Client側(2)

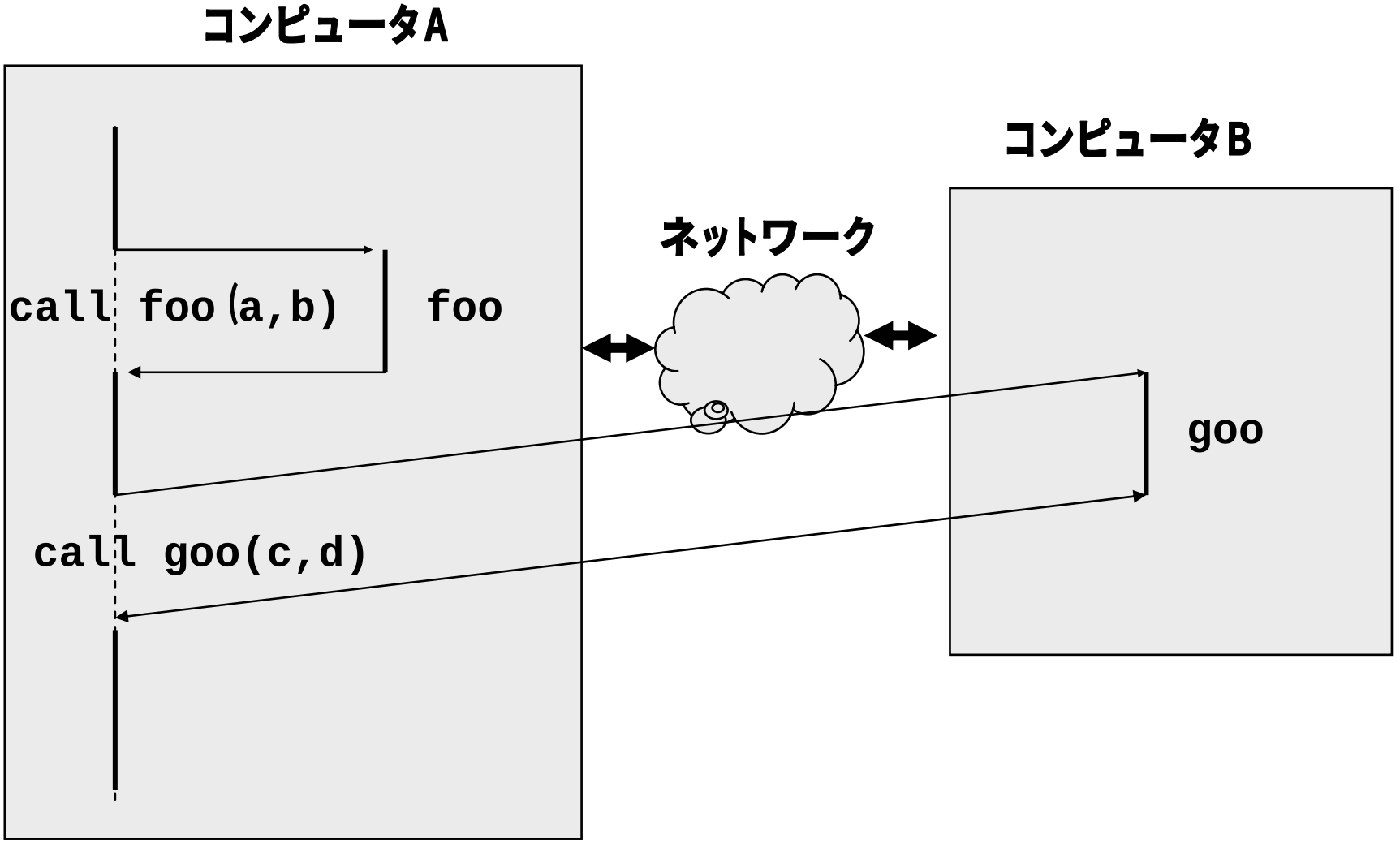
```
    memset(&sin, 0, sizeof(sin));
     sin.sin_family = AF_INET;
     bcopy(hp->h_addr,&sin.sin_addr.s_addr,hp->h_length);
     sin.sin_port = port;

     fd = socket(AF_INET, SOCK_STREAM, 0);
     if(fd < 0){
         perror("socket failed");
         exit(1);
     }

     r = connect(fd,(struct sockaddr *)&sin,sizeof(sin));
     if(r < 0){
         perror("connect failed");
         exit(1);
     }

    sprintf(buf,"hello world...¥n");
     write(fd,buf,strlen(buf)+1);

     close(fd);
     exit(0);
}
```

Remote Procedure Call: 遠隔手続き呼び出し

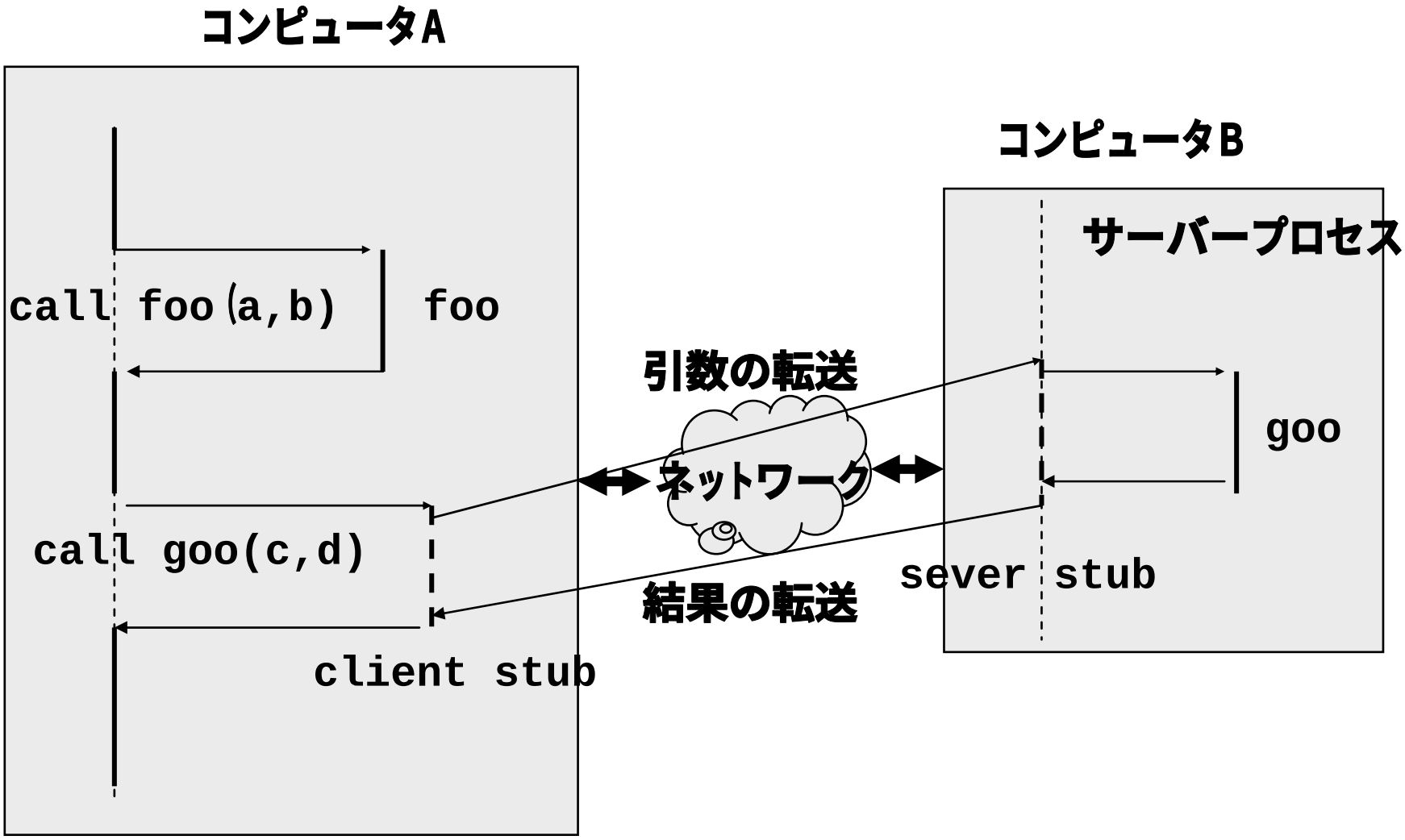
- 基本的には分散システムをプログラミングするためにはTCP/IPやUDPなど低レベルの通信レイヤを使つかう。しかし、いちいち、機能ごとにプロトコルを設計して、通信しなくてはならない。

- このプロトコルを関数呼び出しに抽象化したのが、RPC(remote procedure call）
  - SUN　RPC
  - CORBA (Java、C++)
  - Web Service
  - RMI, Jini….
  - JAX RPC
  - …

「普通」の手続き呼び出しとRPC



もう少し詳しく見ると


コンピュータA
call foo(a,b)
foo
call goo(c,d)
client stub
引数の転送
ネットワーク
結果の転送
コンピュータB
サーバープロセス
goo
sever stub

RPCの手順

① クライアントは通常の手続き呼び出しで、対応するclient stubを呼び出す
② client stubは、引数の情報を含んだメッセージを作り、OSに転送を依頼
③ OSは、メッセージをremoteのOSにネットワークで転送
④ remote OSは、そのメッセージをserver stubに転送
⑤ sever stubは、そのメッセージから引数の情報を取り出す。
⑥ server stubは、引数をもって指定された手続きを通常の手続き呼び出しで呼び出して、結果を得る
⑦ server stubは、その結果を含むメッセージを作り、ローカルなＯＳに転送を依頼
⑧ サーバのＯＳは、クライアントのOSにメッセージをネットワークで転送
⑨ クライアントのOSはそのメッセージを受け取り、client stubに転送
⑩ client stubは、結果を取り出し、クライアントに返す。

RPCミドルウエアとIDL

- RPCの手順をいちいちプログラミングするのは、面倒
- RPCミドルウエアは、関数のインタフェースの記述(IDL: Interface Description Language)から、自動的に、client stubやserver stubを作ってくれるソフトウエア
  - SUN RPC
  - Java RMI
- これによって、関数のローカル呼び出しとほとんど変わらず、RPCが利用可能になる。

その他

- 非同期RPC
  - RPCの終了を待たずに、他のローカルの動作をする方法
  - RPCの呼び出しはそれなりに時間がかかる
  - RPCの呼び出された手続きの実行は他のコンピュータで行われているので、ローカルでは待機状態のはず。

- namingサービス
  - 実際にRPCを使ってシステムを構築するには、どこにどの手続きがあるかを管理する必要がある。
  - そのためのデータベース（これ自身も分散処理されているプログラム）

まとめ

- ISOの７層モデル
  - ４層までは、覚えておくこと
- 基本的なネットワークプログラミング
  - socketなど
- RPCの基本
  - 有効な通信の抽象化、広く用いられている