Rozproszone systemy obiektowe

Grzegorz Jabłoński
Mariusz Orlikowski

Program wykładu

- Wprowadzenie
- Sockets
- ~~ONC (Sun) RPC~~
- ~~CORBA~~Ice
- DCOM

Co to jest system rozproszony ?

- Poznajemy go po tym, że pracę uniemożliwia nam awaria komputera, o którym nigdy wcześniej nie słyszeliśmy – Leslie Lamport, 1987
- Zespół (być może) heterogenicznych węzłów połączonych w sieć, która zapewnia dostęp do dzielonych zasobów lub usług
- Zespół niezależnych komputerów, który widziany jest przez użytkownika jako jeden spójny system

Główne cechy systemu rozproszonego

- Wiele komputerów
  - Mogą być homo- lub heterogeniczne
- Połączonych w sieć
  - Typowo sieć ogólnego przeznaczenia, nie skonstruowaną celowo na potrzeby systemu rozproszonego
- Współpracujących w celu dzielenia zasobów lub usług
  - Program jest wykonywany na więcej niż jednym komputerze

Trendy technologiczne

- Przetwarzanie równoległe – podział obliczeń między wiele komputerów
- Dwa sposoby budowy takich systemów
  - Systemy wieloprocesorowe – ściśle powiązane, procesory współdzielą pamięć i zegar, komunikacja odbywa się przez pamięć dzieloną, wydzielona wewnętrzna sieć komunikacyjna
  - Systemy rozproszone – luźno powiązane, procesory nie dzielą pamięci i zegara, komunikacja odbywa się poprzez zewnętrzną sieć ogólnego przeznaczenia
- Tanie, wydajne komputery osobiste
- Wydajne sieci

Motywacja dla systemów rozproszonych

- Dzielenie zasobów
- Przyspieszenie obliczeń
  - Części składowe mogą działać współbieżnie
  - Równoważenie obciążenia
- Niezawodność
  - Przy odpowiedniej nadmiarowości częściowo uszkodzony system może pracować nadal
- Komunikacja
  - Wsparcie dla aplikacji do pracy grupowej

Terminologia

- Z punktu widzenia każdego procesora w systemie rozproszonym reszta procesorów i ich zasoby są zdalne, jego własne są lokalne
- Jednostki obliczeniowe w systemie rozproszonym bywają różnie nazywane
  - Hosty, węzły, komputery, maszyny, ...

Systemy operacyjne

- Sieciowe systemy operacyjne
  - Każdy komputer ma własny SO
  - SO zawiera usługi sieciowe umozliwiające dostęp do zdalnych zasobów
  - Użytkownik jest świadomy istnienia wielu komputerów i musi się tym jawnie zajmować
- Rozproszone systemy operacyjne
  - SO wielu komputerów współpracują ze sobą tworząc wrażenie pojedynczego systemu
  - Użytkownik jest nieświadomy istnienia wielu komputerów
- Middleware
  - Warstwa oprogramowania między SO i aplikacjami wspierającymi przetwarzanie rozproszone

Sieciowe systemy operacyjne

- Typowym przykładem są r-polecenia w systemie UNIX
- Remote login (rlogin)
  - Przezroczyste, dwukierunkowe połączenie
- Remote file transfer (rcp)
  - Lokalizacja plików nie jest przezroczysta dla użytkownika
  - Brak prawdziwego dzielenia plików
- Rsh, rexec etc.
- Usługi są implementowane jako procesy na zdalnym komputerze czekające na połączenia

Rozproszone systemy operacyjne

- Wbudowane wsparcie dla
  - Migracji danych
    - Całego pliku z miejsca A do miejsca B lub jedynie jego aktualnie niezbędnych części
  - Migracji obliczeń
    - Przeniesienie obliczeń w inne miejsce może być bardziej efektywne niż przeniesienie danych
  - Migracji procesów
    - Proces nie jest zawsze wykonywany na komputerze na którym został uruchomiony (równoważenie obciążeń lub niezawodność)

Cele projektowe systemów rozproszonych

- Przezroczystość
  - Z punktu widzenia użytkownika system rozproszony powinien wyglądać jak konwencjonalny, scentralizowany system
  - Dlaczego trudno to osiągnąć
    - Inne typy uszkodzeń
    - Wydajność zależy od podziału obliczeń
  - Innym aspektem przezroczystości jest mobilność użytkowników
    - Użytkownik siadający przy dowolnym komputerze w systemie powinien widzieć to samo
  - Przezroczystość nie zawsze jest korzystna
    - “Dobre udogodnienie to takie, które można wyłączyć”
    - Potrzeba wiedzy o rozproszonej naturze w celu optymalizacji aplikacji

Inne cele: niezawodność

- Odporność na uszkodzenia
  - Dodanie komputerów do systemu zwiększa prawdopodobieństwo, że któryś z nich ulegnie uszkodzeniu
  - Tak jak w przypadku macierzy dyskowych RAID, chcemy z wady uczynić zaletę: stosujemy replikację stanu obliczeń, dzięki czemu system może pracować (być może mniej wydajnie) nawet przy występowaniu uszkodzeń
  - Powinien wspierać automatyczną reintegrację komputerów w systemie rozproszonym

Inne cele: skalowalność

- Zdolnośc systemu do adaptacji do zwiększonego obciążenia
  - Zmniejszenie wydajności i zajęte zasoby
- Konieczność zapewnienia rezerwowych zasobów w celu zapewnienia niezawodności i poprawnej pracy przy zwiększonym obciążeniu (np. wytrzymanie “slashdot effect” w przypadku serwerów www)
- Zasada: zapotrzebowanie na usługi pojedynczego komponentu systemu powinno być ograniczone z góry przez pewną stałą, niezależnie od liczny węzłów w systemie

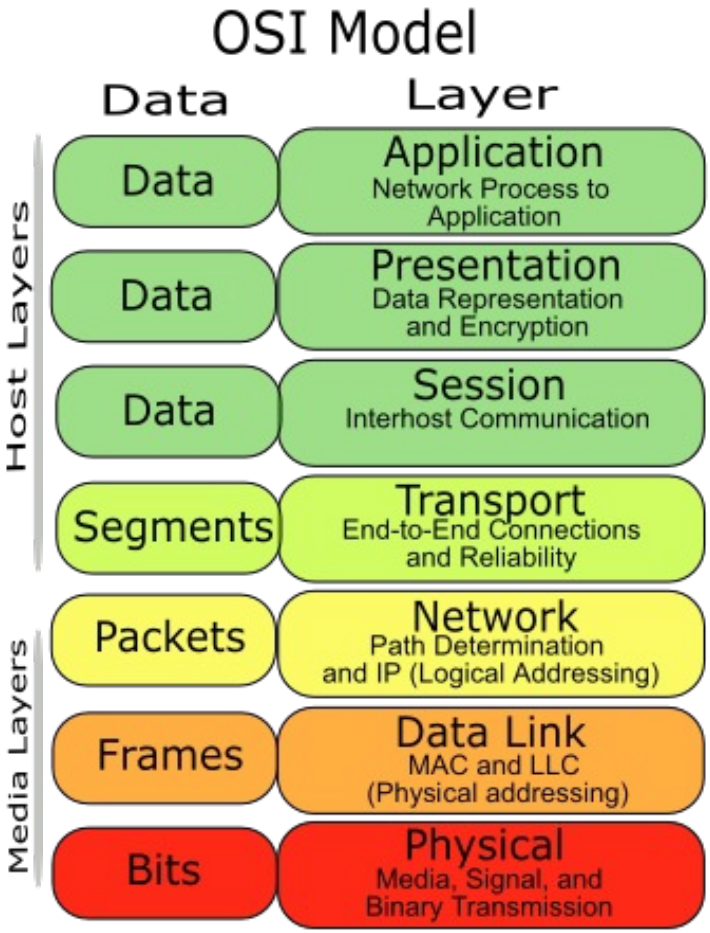
Architektura klient-serwer

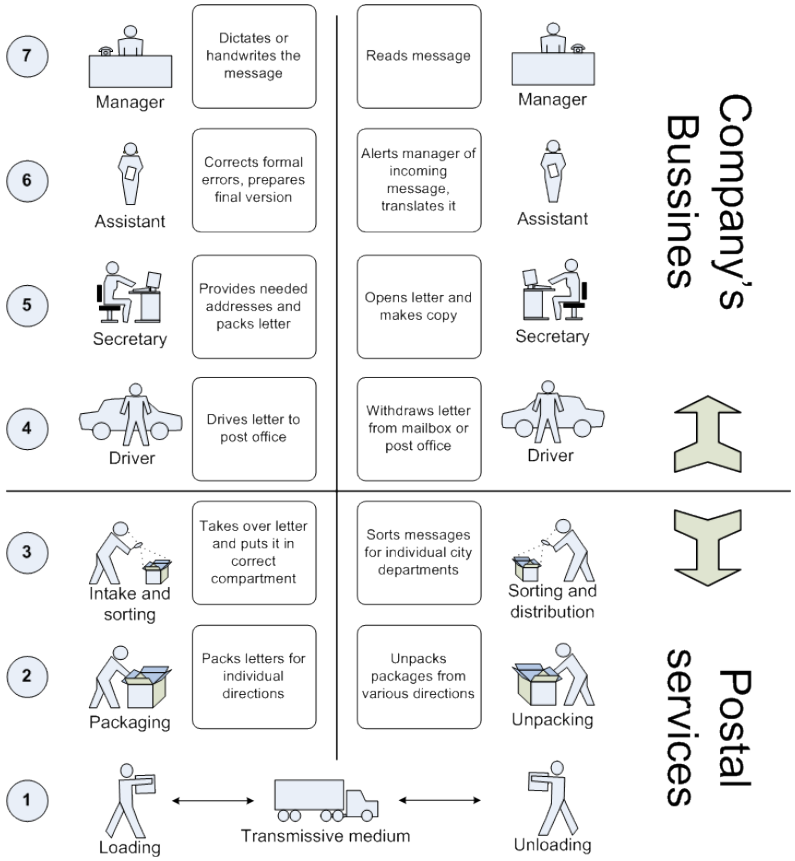
- Zadania aplikacji są podzielone między komputery klienckie i serwery
  - Typowo serwery posiadają jakieś szczególne cechy, niemożliwe do bezpośredniego uzyskania w dużej liczbie klientów
    - Więcej pamięci, więcej lub szybsze procesory, więcej pamięci, licencje na kosztowne oprogramowanie etc.
  - Wybór sposobu podziału aplikacji zależy od chrakterystyki klienta, serwera, aplikacji, sieci komunikacyjnej i spodziewanego obciążenia systemu
    - Thin client vs. thick client

Komunikacja w systemach rozproszonych

- Brak fizycznej pamięci dzielonej, konieczność przesyłania komunikatów
- Podstawowe operacje to Send() i Receive()
  - Client wysyła zapytanie i czeka na odbiór odpowiedzi. Serwer odbiera zapytanie i wysyła odpowiedź
  - Dane muszą być w jakiś sposób zakodowane w ciele komunikatu
- Komunikaty mogą być zawodne lub niezawodne (w przypadku zawodnych, wyższa warstwa musi zapewnić niezawodność transmisji)
- Operacje mogą być blokujące i nieblokujące
- Może być oparta na specjalizowanym protokole lub protokole ogólnego przeznaczenia (np. TCP/IP)

OSI Reference Model





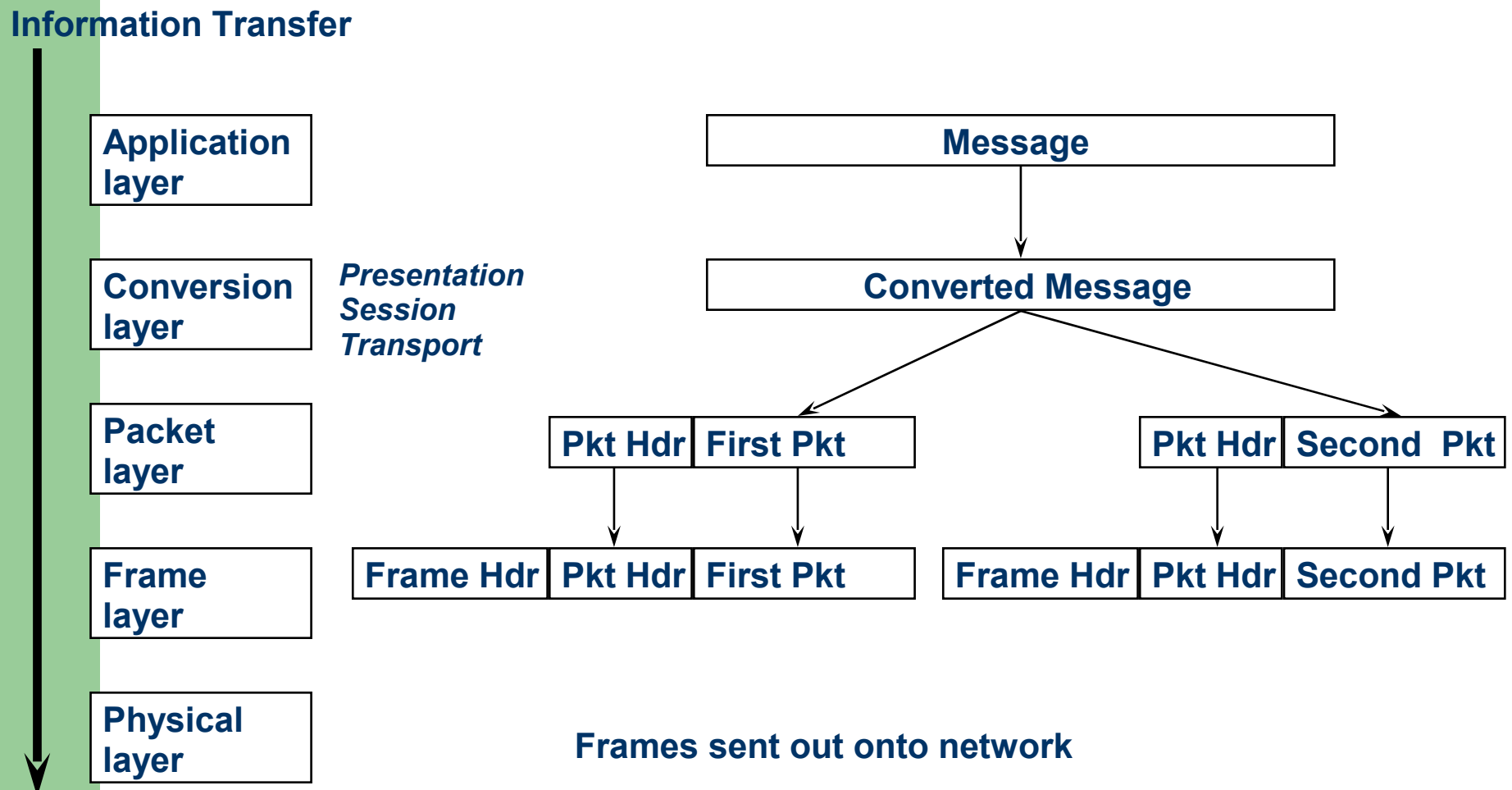
RM – OSI and letter communication parallel

1 April 1990 - RFC 1149

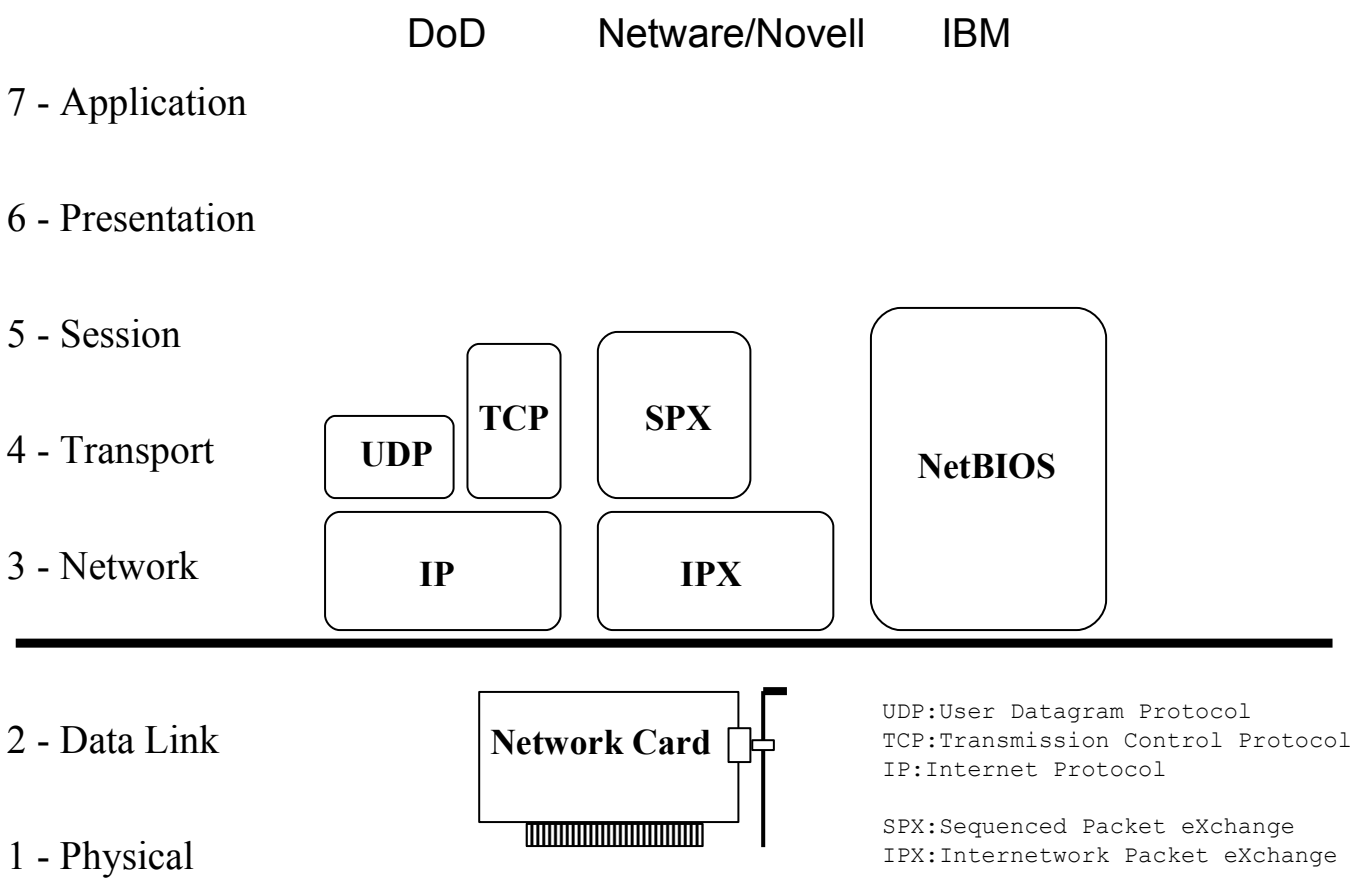
Standard for the transmission of IP datagrams on avian carriers
Implementation: http://www.blug.linux.no/rfc1149/

```
Script started on Sat Apr 28 11:24:09 2001
vegard@gyversalen:~$ /sbin/ifconfig tun0
tun0   Link encap:Point-to-Point Protocol
       inet addr:10.0.3.2  P-t-P:10.0.3.1  Mask:255.255.255.255
       UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:150  Metric:1
       RX packets:1 errors:0 dropped:0 overruns:0 frame:0
       TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0
       RX bytes:88 (88.0 b)  TX bytes:168 (168.0 b)

vegard@gyversalen:~$ ping -i 900 10.0.3.1
PING 10.0.3.1 (10.0.3.1): 56 data bytes
64 bytes from 10.0.3.1: icmp_seq=0 ttl=255 time=6165731.1 ms
64 bytes from 10.0.3.1: icmp_seq=4 ttl=255 time=3211900.8 ms
64 bytes from 10.0.3.1: icmp_seq=2 ttl=255 time=5124922.8 ms
64 bytes from 10.0.3.1: icmp_seq=1 ttl=255 time=6388671.9 ms

--- 10.0.3.1 ping statistics ---
9 packets transmitted, 4 packets received, 55% packet loss
round-trip min/avg/max = 3211900.8/5222806.6/6388671.9 ms
vegard@gyversalen:~$ exit

Script done on Sat Apr 28 14:14:28 2001
```

Implementacja warstw



Kilka ważniejszych protokołów



Implementowany protokół

- Zaimplementować serwer pierwiastkujący liczby rzeczywiste i podający bieżący czas na serwerze
- Oparty na protokole TCP
- W celu zapewnienia przenośności między różnymi procesorami wszystkie liczby przesyłamy w standardzie Big Endian
- RQ ID pozwala na rozróżnienie różnych zapytań

Pytanie o pierwiastek:

0	0	0	1	RQ ID	Liczba (IEEE double)

Odpowiedź:

1	0	0	1	RQ ID	Pierwiastek (IEEE double)

Implementowany protokół

- Datę i czas przesyłamy w postaci tekstowej, bez kończącego zera
- Długość podajemy w kolejności Big Endian
- W jednym połączeniu można przesłać kilka zapytań
- Kolejność odpowiedzi może być inna, niż kolejność zapytań

Pytanie o czas:

0	0	0	2	RQ ID

Odpowiedź:

1	0	0	2	RQ ID	Długość (BE)	Data i czas

Inne paradygmat projektowania

- Projektowanie zorientowane na komunikację
  - Zaprojektuj protokół
  - Napisz program zgodny z protokołem
- Projektowanie zorientowane na aplikację
  - Napisz program
  - Podziel program i dodaj protokół komunikacyjny

Sockety

RPC

RPC – Remote Procedure Call

- Wywołanie procedury (podprogramu) działającej na innym komputerze
- Problemy
  - Identyfikacja i dostęp do zdalnej procedury
  - Parametry
  - Wartości zwracane

Zdalna procedura

Client

```
blah, blah, blah

bar = foo(a,b);

blah, blah, blah
```

protocol

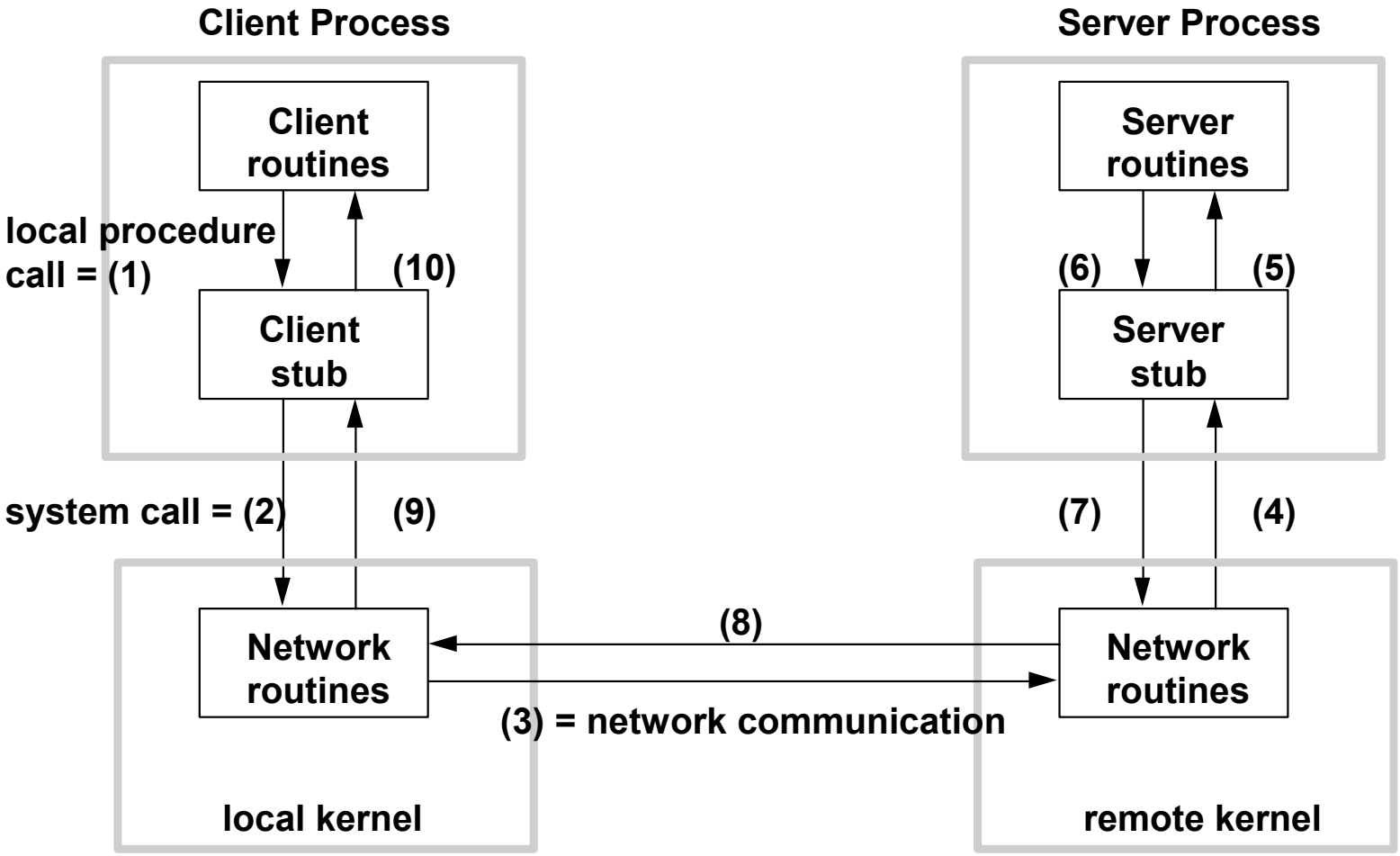
Server

```
int foo(int x, int y ) {
  if (x>100)
       return(y-2);
  else if (x>10)
       return(y-x);
  else
       return(x+y);
}
```

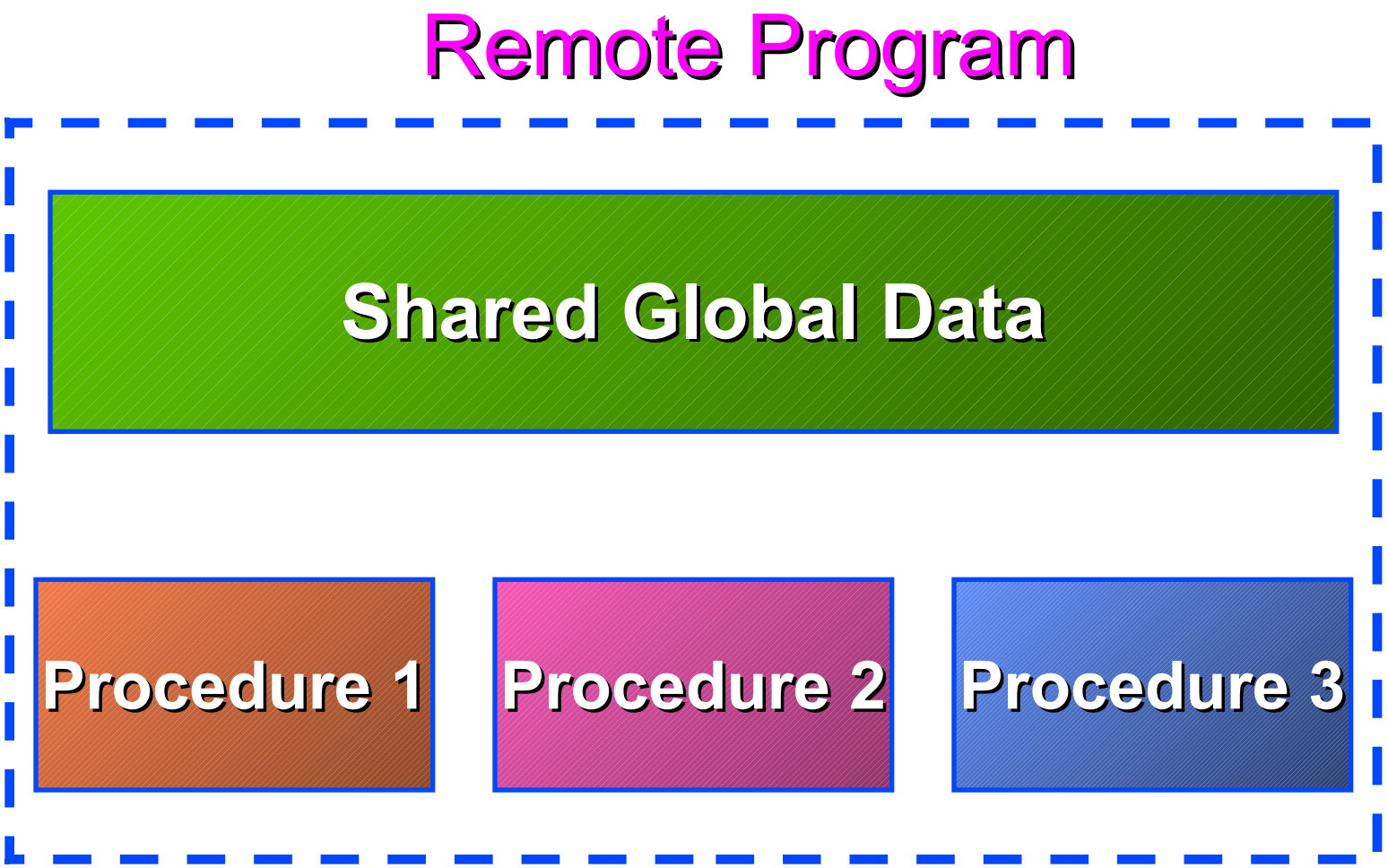
Sun RPC

- Jest kilka różnych specyfikacji RPC
- Jedną z powszechnie stosowanych jest Sun RPC (ONC RPC)
- NFS (Network File System) jest oparty na RPC
- Bogaty zestaw narzędzi pomocniczych

10 kroków wywołania RPC



Organizacja Sun RPC



Argumenty procedur

- W Sun RPC mozna przekazac do procedury jedynie pojedynczy argument
- Zazwyczaj jest to struktura, zawierająca kilka wartości

RPCGEN

- Istnieje narzędzie ułatwiające tworzenie klientów i serwerów RPC
- Program *rpcgen* wykonuje większość pracy programisty
- Wejściem do programu *rpcgen* jest definicja protokołu w formie listy zdalnych procedur i typów parametrów

Plik definicji protokołu

- Opis interfejsu zdalnych procedur
  - Prawie prototypy funkcji
- Definicje wszystkich struktur danych używanych w wywołaniach (typy argumentów i wartości zwracanych)
- Może również zawierać współdzielony kod (współdzielony przez klienta i serwer)

Przykładowy plik definicji protokołu

```
struct twonums {
   int a;
   int b;
};
program UIDPROG {
   version UIDVERS {
      int  RGETUID(string<20>) = 1;
      string RGETLOGIN( int ) = 2;
      int  RADD(twonums)  = 3;
   } = 1;
} = 0x20000001;
```

Język XDR

- Typy danych XDR to nie to samo, co typy języka C
  - Istnieje odwzorowanie (ang. *mapping*) między typami danych używanymi przez XDR i C – tym głównie zajmuje się **`rpcgen`**
- Składnia XDR jest w większości zbliżona do C
  - Różnica przy łańcuchach i tablicach

Tablice w XDR

- Tablice o stałej długości wyglądają tak samo, jak w C

```
int foo[100]
```

- Tablice o zmiennej długości wyglądają następująco
- `int foo<>` lub `int foo<MAXSIZE>`
  - Domyślna długość maksymalna to 2^{32}-1

Tablice w XDR – co jest wysyłane

```
int x[n]
```

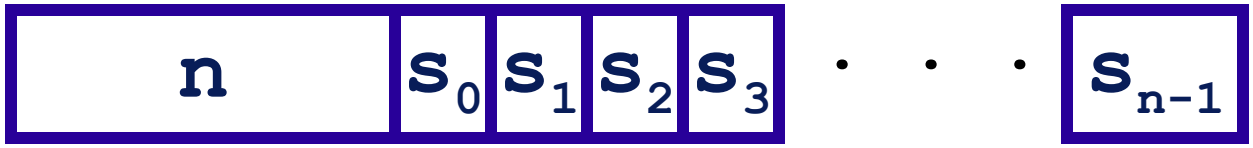

```
int y<m>
```

`k` jest rzeczywistym rozmiarem tablicy

$k \leq m$

Łańcuchy w XDR

- Deklarowane podobnie, jak tablice o zmiennej długości
- **`string s<100>`**
  - Wysyłana jest długość łańcucha, a następnie sekwencja znaków ASCII



`n` jest rzeczywistą długością łańcucha (wysyłaną jako **`int`**)

Identyfikacja procedur

- Każda procedura jest identyfikowana przez
  - Nazwa komputera (Adres IP)
  - Identyfikator programu (32-bitowa liczba całkowita)
  - Identyfikator procedury (32-bitowa liczba całkowita)
  - Numer wersji programu
    - Dla celów testowych i uproszczenia migracji

Identyfikator programu

- Każdy zdalny program ma unikalny identyfikator
- Zakresy identyfikatorów
  - `0x00000000 - 0x1fffffff Defined by Sun`
  - `0x20000000 - 0x3fffffff Defined by user`
  - `0x40000000 - 0x5fffffff Transient`
  - `0x60000000 - 0xffffffff Reserved`

Identyfikator procedury i numer wersji programu

- Identyfikatory procedur zwykle zaczynają się od 1 i przyjmują wartości kolejnych liczb całkowitych
- Numery wersji zwykle zaczynają się od 1 i przyjmują wartości kolejnych liczb całkowitych

Serwer iteracyjny

- W Sun RPC najwyżej jedna zdalna procedura może być wykonywana w danym momencie
- Jeżeli zostanie wywołana kolejna, wywołanie jest blokowane do momentu zakończenia pierwszego
- W niektórych sytuacjach (np. operacje na bazie danych) może to być korzystne

Semantyka wywołania procedury

- Co oznacza wywołanie lokalnej procedury
  - Jest ona uruchomiona dokładnie raz
- Co oznacza wywołanie zdalnej procedury
  - Może nie być uruchomiona dokładnie raz, jeżeli stosuje się domyslny protokół UDP
- Semantyka “co najmniej raz”
  - Jeżeli procedura zwraca wartość
- Semantyka “zero lub więcej razy”
  - Jeżeli procedura nie zwraca wartości

Zdalna procedura `deposit()`

`deposit(GWJAccount,$100)`

- Zawsze należy pamiętać, że nie wiadomo, ile razy procedura została wywołana
  - Sieć może powielić wywołanie (UDP)

Dynamiczne mapowanie portów

- Serwery RPC zwykle nie używają dobrze znanych portów
- Klient zna Program ID i adres IP serwera
- RPC zawiera mechanizm umożliwiający znalezienie numeru portu zdalnego programu
- Usługa wyszukiwania portów musi być uruchomiona na każdym komputerze, który zawiera serwery RPC
- Serwer RPC rejestruje się w tym serwisie
  - “Jestem programem 35 i nasłuchuję na porcie 12343”

Dynamiczne mapowanie portów

- Program implementujący wyszukiwanie portów nosi nazwę portmapper
- Portmapper to również serwer RPC
- Portmapper nasłuchuje na porcie 111

Programowanie z użyciem Sun RPC

- Biblioteka RPC to zestaw narzędzi służących do automatyzacji tworzenia klientów i serwerów RPC
- Klienci RPC to procesy wywołujące zdalne procedury
- Serwery RPC to procesy zawierające procedury, które mogą być wywoływane przez klientów

Programowanie z użyciem Sun RPC

- Biblioteka RPC
  - Procedury XDR
  - Biblioteka RPC
    - Wywołanie usługi RPC
    - Rejestracja w portmapperze
    - Przekazanie nadchodzących wywołań do właściwych procedur
  - Generator programów

Biblioteka RPC

- Funkcje niskiego i wysokiego poziomu które mogą być używane w serwerach i klientach
- Wysokopoziomowe funkcje zapewniają prosty dostęp do usług RPC

Wysokopoziomowa biblioteka RPC

```
int callrpc( char *host,
             u_long prognum,
             u_long versnum,
             u_long procnum,
             xdrproc_t inproc,
             char *in,
             xdrproc_t outproc,
             char *out);
```

Wysokopoziomowa biblioteka RPC

```
int registerrpc(
            u_long prognum,
            u_long versnum,
            u_long procnum,
            char *(*procname)(),
            xdrproc_t inproc,
            xdrproc_t outproc);
```

Wysokopoziomowa biblioteka RPC

```
void svc_run();
```

- `svc_run()` to dyspozytor (ang. *dispatcher*)
- Dyspozytor czeka na połączenia przychodzące i wywołuje odpowiednią funkcję w celu obsłużenia danego zgłoszenia

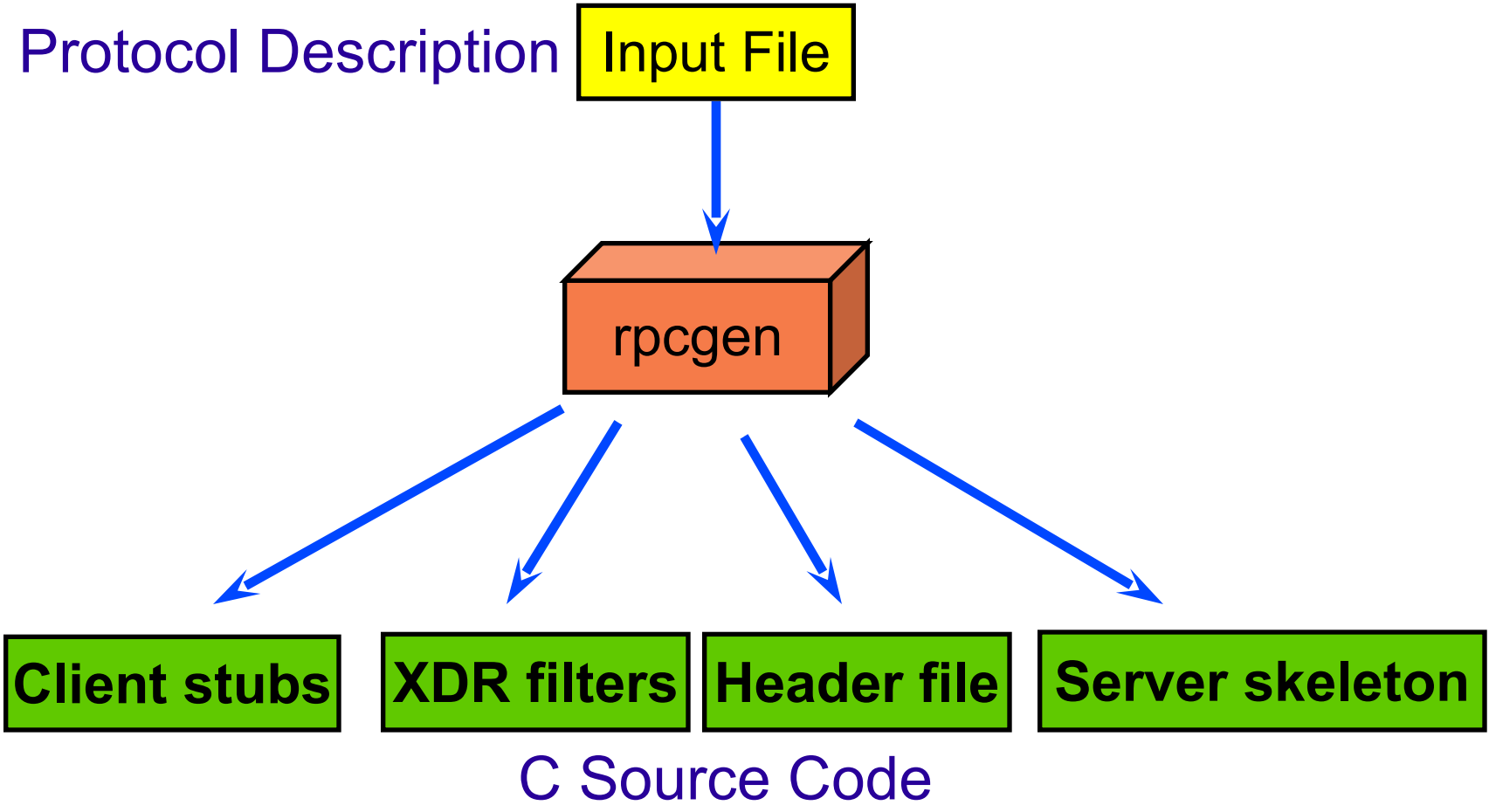
Ograniczenia biblioteki wysokopoziomowej

- Wysokopoziomowa biblioteka RPC wspiera jedynie UDP (nie TCP)
- W celu użycia TCP należy używać biblioteki niskopoziomowej
- Biblioteka wysokopoziomowa nie wspiera uwierzytelniania

Niskopoziomowa biblioteka RPC

- Pełna kontrola nad wszystkimi opcjami RPC
  - TCP & UDP
  - Wartości timeout
- Broadcast

RPCGEN


Protocol Description
Input File
rpcgen
Client stubs
XDR filters
Header file
Server skeleton
C Source Code

Pliki wyjściowe `rpcgen`

```
> rpcgen foo.x
```

`foo_clnt.c`	(client stubs)
`foo_svc.c`	(server main)
`foo_xdr.c`	(xdr filters)
`foo.h`	(shared header file)

Kompilacja klienta

```
> gcc -o fooclient foomain.c foo_clnt.c foo_xdr.c
```

- **`foomain.c`** zawiera funkcję **`main()`** klienta (i być może inne funkcje) która wywołuje usługi rpc za pośrednictwem funkcji pośredniczących (ang. *stub functions*) w **`foo_clnt.c`**
- Funkcje pośredniczące używają filtrów xdr

Kompilacja serwera

```
> gcc -o fooserver fooservices.c foo_svc.c foo_xdr.c
```

- **`fooservices.c`** zawiera definicje zdalnych procedur