libm3l and lsipdx

Data Protocol and Synchronization and Inter-Process Data Exchange Utilities

Adam Jirásek, Arthur Rizzi

The Royal Institute of Technology, KTH

Stockholm, Sweden

Introduction

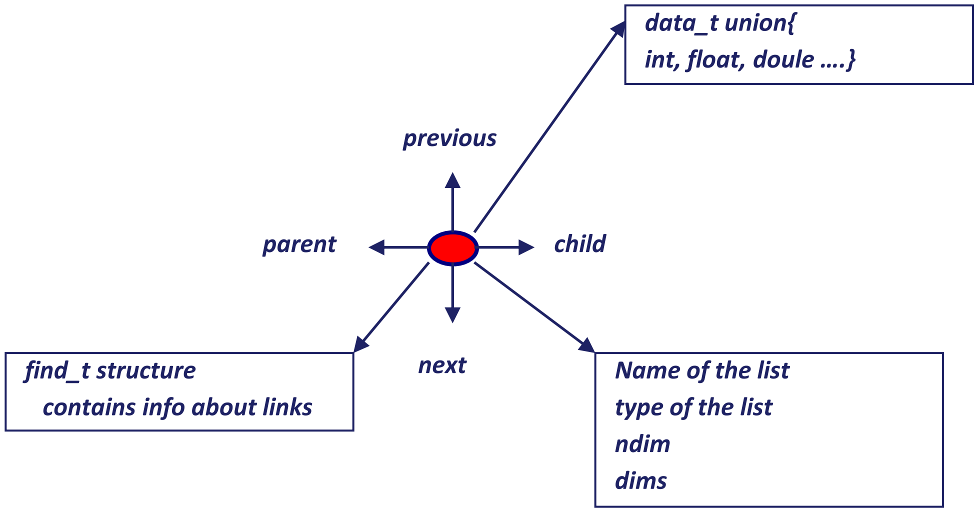
- Primary goal to make a utility which enables easy implementation of the Inter-Process Communication, IPC
  - Increased demand using multi-disciplinary computer modeling
  - Modular approach to software
    - Enable integration of newly designed and legacy software
    - Easy way to replace older modules with newer ones
    - All processes/solvers dedicated to their specific task, communication through common interface

Overview

- Two OSS libraries
  - **libm3l**
    - Multi-Level Linked List Library
    - The main purpose – enable easy basic data storage and their transfer over the TCP/IP sockets
  - **lsipdx**
    - Synchronization and Inter-Process Data eXchange
    - The main purpose is data flow control and synchronization

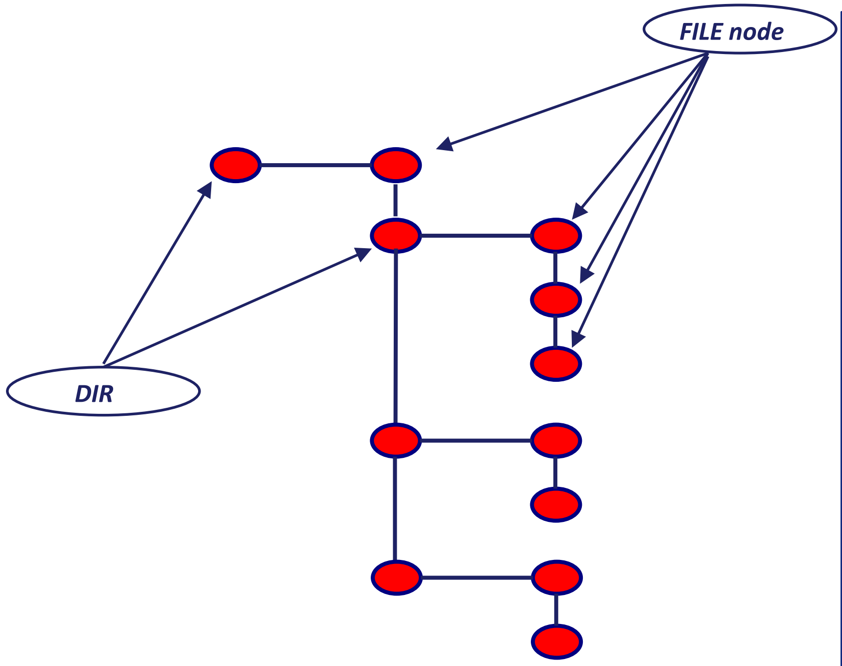
libm3l - Overview

- Basic element
  - Node (type *node_t **)
    - Name of the node
    - Type of the node
      - **FILE**
      - **DIR**
      - **LINK**
    - Number of array dimensions
      - In case of **DIR**, number of items in **DIR** node
      - In case of **FILE**, number of dimensions of array in **FILE**
    - In case of **FILE** - array dimensions
    - Basic data stored in **FILE** node



libm3l - Overview

- Example of linked list
  - 3 DIR nodes
  - Char arrays
  - 2D integer array of 2x2 dimensions



```
grid1 DIR 4
        name_of_grid C 1 9 ‘
                CSM_grid‘
        boundary DIR 3
                name C 1 5
                        ‘Wing‘
                type C 1 5
                        ‘Wall‘
                Index_FieldI 2 2 2
                        8001 8002 8003 8004
        boundary DIR 2
                name1 C 1 9
                        ‘Fuselage‘
                type C 1 5
                        ‘Wall‘
        boundary DIR 2
                name2 C 1 9
                        ‘External‘
                type2 C 1 11
                        ‘freestream‘
```

libm3l - Overview

- Locating node
  - Linked list can contain several nodes of the same name
  - Location determined by path and additional specification

m3l_Locate(List1, “/grid1/boundary”, “/*/SV_type=Wall”, (char *)NULL)

- Gives all boundaries in grid1 which are type ***Wall***

```
grid1 DIR 4
        name_of_grid C 1 9 ‘
                CSM_grid‘
        boundary DIR 3
                name C 1 5
                        ‘Wing‘
                type C 1 5
                        ‘Wall‘
                Index_FieldI 2 2 2
                        8001 8002 8003 8004
        boundary DIR 2
                name1 C 1 9
                        ‘Fuselage‘
                type C 1 5
                        ‘Wall‘
        boundary DIR 2
                name2 C 1 9
                        ‘External‘
                type2 C 1 11
                        ‘freestream‘
```

libm3l – Create Node

- Creating a FILE type and allocate memory

```
int *a;

dim[0]=10;
Node = m3l_Mklist("IntNum", "I", 1, dim, &MainNode, "/Data", "./", (char *)NULL);
a = (int*)m3l_get_data_pointer(TmpNode);
```

  - operations with the array

```
a[i] = …. ;
```

  - Function *m3l_Mklist* invokes malloc
    - *malloc(sizeof(int) * 10)*

libm3l – Create Node as Reference

- Creating a FILE type without allocating memory

```
int MyVar[10];
int *a;

dim[0]=10;
Node = m3l_Mklist("IntNum", "I", 1, dim, &MainNode, "/Data", "./", "—no_malloc",
        (char *)NULL);
a = (int *)m3l_get_data_pointer(TmpNode);
a = MyVar;
```

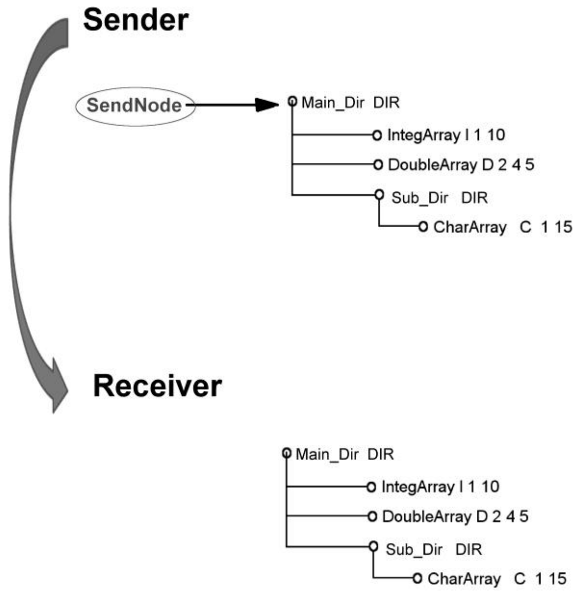
  – the variable can be access and manipulated through linked list too

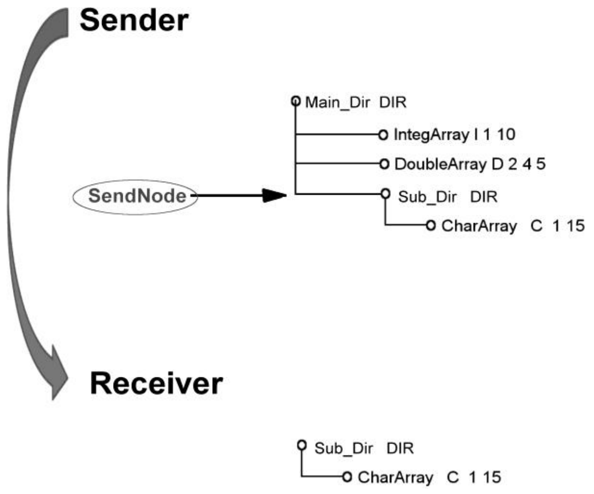
```
a[i] = …. ;
MyVar[i] … ;
```

- libm3l list can be used as a reference to already existing data
  - Compatibility with already existing data structure in a code

libm3l – Sending Data Over TCP/IP

- **Send/receive functions**
  - **m3l_Send_to_tcpipsocket**(node_t *Send_node, hostname, IP)
  - **(node_t *) m3l_Receive_tcpipsocket**(hostname, IP)





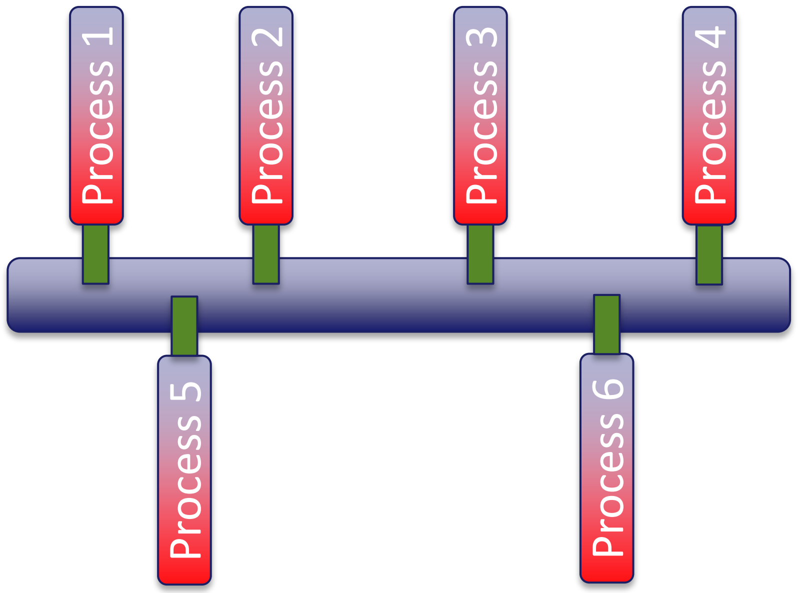
Invoking libm3l Function

- Each function has API
  - Either called with parameters through caller
    - Node specified using path and location
    - Using short and long options
    - getopt_long() used to parse arguments
  - Or directly with filled structure with options
    - Node usually have to be located before
    - Options specified through structure

```
m3l_Mv(List1, “/list1/data/grid1”, “/*/*/*”, List2, “/list2”, “./”, “--ignore”, (char *)NULL)

m3l_mv_list (List1, List2, Opts)
```

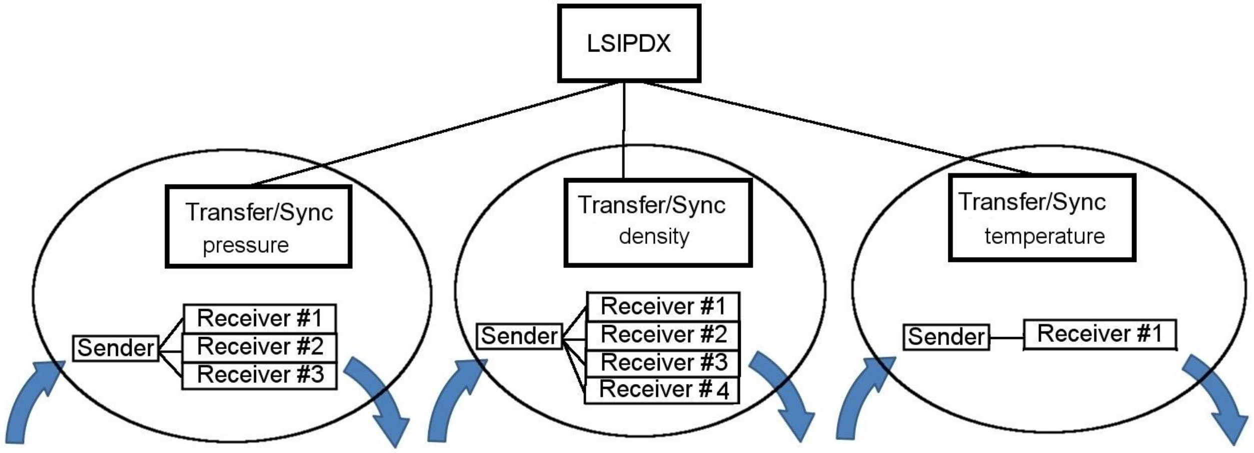
Inter-Process Communication and Synchronization

- Inter-process communication and synchronization
  - Dedicated utility for the layer which enables data flow
  - Processes are modules
    - Keep them out of data transfer and synchronization business
      - Examples – CREATE, GMSEC, SALOME



lsipdx - Inter-Process Communication and Synchronization

- **lsipdx** – **S**ynchronization and **I**nter-**P**rocess **D**ata e**X**change utility
  - Opens communication channels with given number of hubs for each channel
  - Receives requests to send/receive data and directs them to the specified hub
  - Directs and synchronizes transfer between processes connected to channel
    - Sender – Receiver(s)



Isipdx - Inter-Process Communication and Synchronization

- **Server**
  - Opens/binds/listen port
  - Input file
    - Number of connections/channels
    - Number of connecting processes for each channel
    - Other info (KA and ATDT mode)

- **Client**
  - Establish connection with server

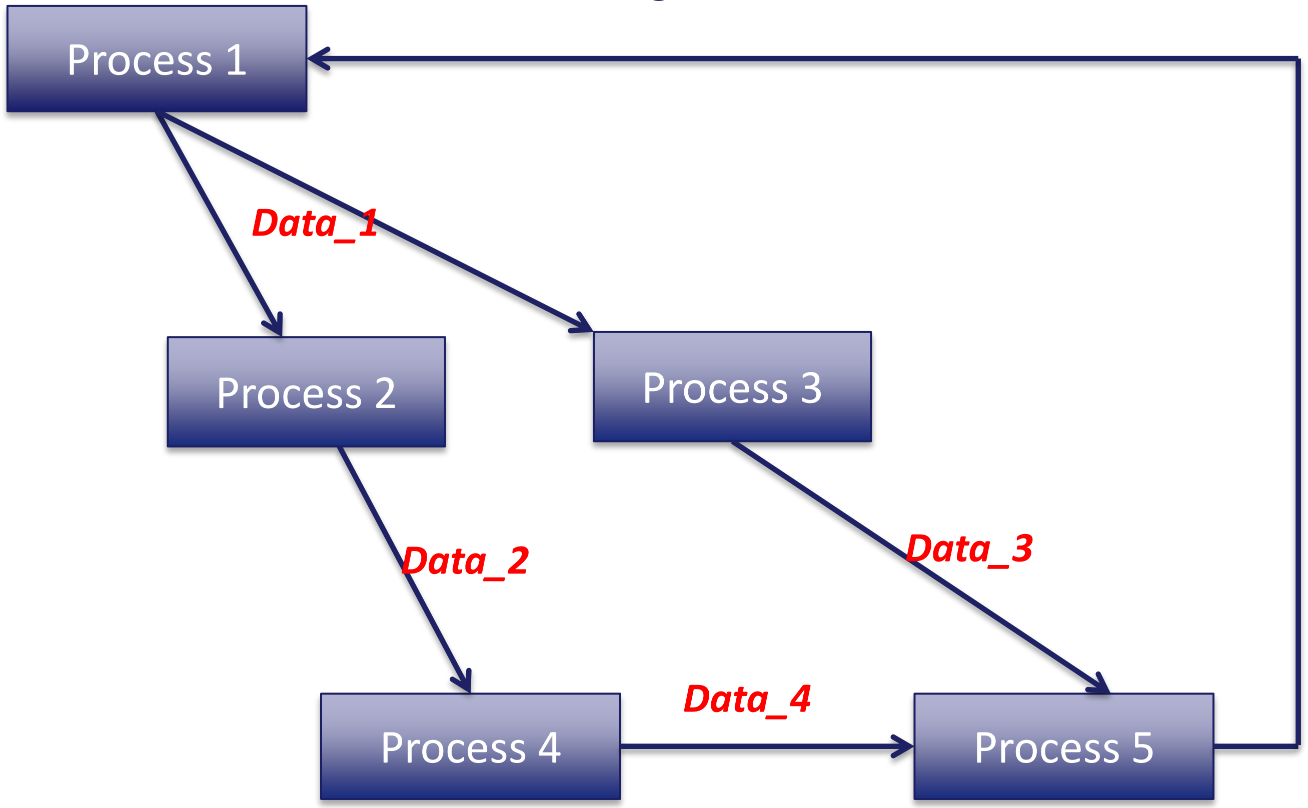
    open socket(hostname, port #, name of channel, “S/R,”..)
  - Transfer data

    send/receive libm3l data set
  - Close connection to server

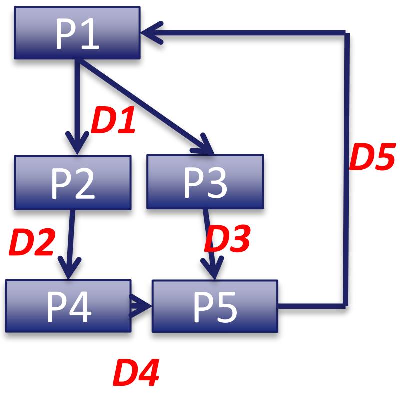
```
COMM_DEF  DIR 1
      Connections  DIR 3
            Connection  DIR 4
                  Name_of_Connection C 1 9
                        `Pressure`
                  Sending_Process I 1 1
                        1
                  Receiving_Processes ST 1 1
                        3
                  ........
            Connection  DIR 4
                  Name_of_Connection C 1 8
                        `Density`
                  Sending_Process I 1 1
                        1
                  Receiving_Processes ST 1 1
                        4
                  .......
            Connection  DIR 4
                  Name_of_Connection C 1 12
                        `Temperature`
                  Sending_Process I 1 1
                        1
                  Receiving_Processes ST 1 1
                        1
                  ..............
```

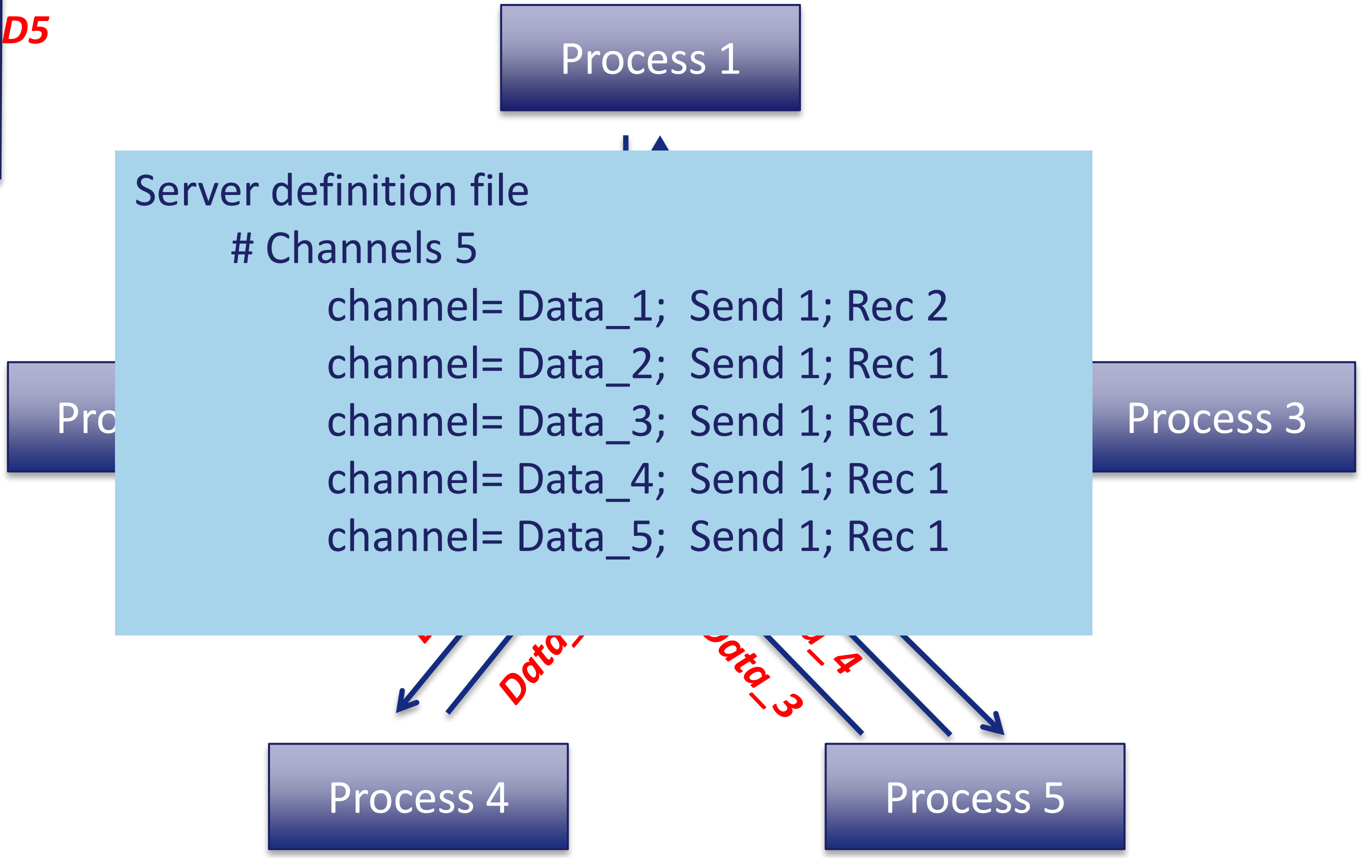
Isipdx - Inter-Process Communication and Synchronization – Example

- Example of implementation
  - 5 communicating processes
  - 5 transferred data sets through five connections/channels


$$\begin{tikzcd}
\text{Process 1} \arrow[d, "Data\_1"] \arrow[dr] & & \\
\text{Process 2} \arrow[d, "Data\_2"] & \text{Process 3} \arrow[dr, "Data\_3"] & \\
\text{Process 4} \arrow[rr, "Data\_4"] & & \text{Process 5} \arrow[uull, bend right, "Data\_5"']
\end{tikzcd}$$

Isipdx - Inter-Process Communication and Synchronization – Example

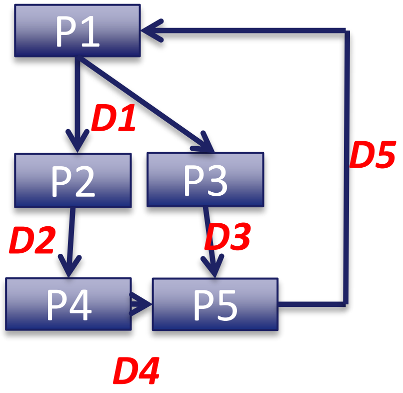


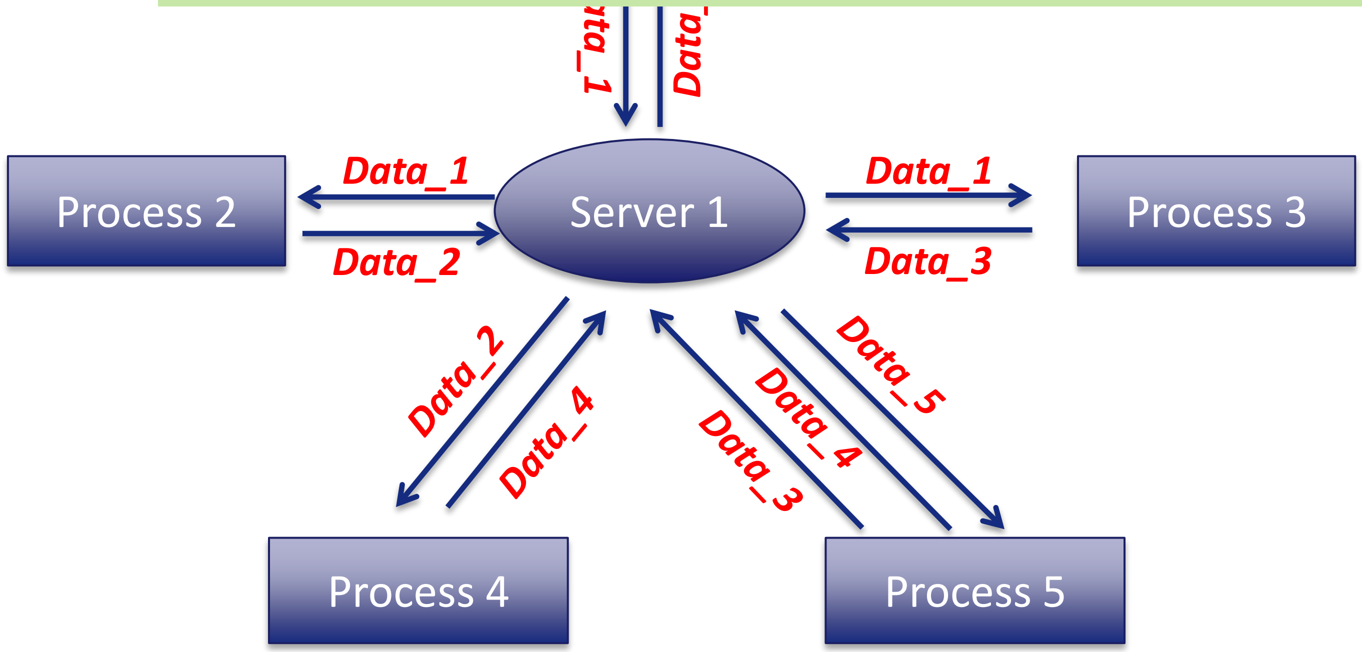


```
Server definition file
        # Channels 5
                channel= Data_1;  Send 1; Rec 2
                channel= Data_2;  Send 1; Rec 1
                channel= Data_3;  Send 1; Rec 1
                channel= Data_4;  Send 1; Rec 1
                channel= Data_5;  Send 1; Rec 1
```

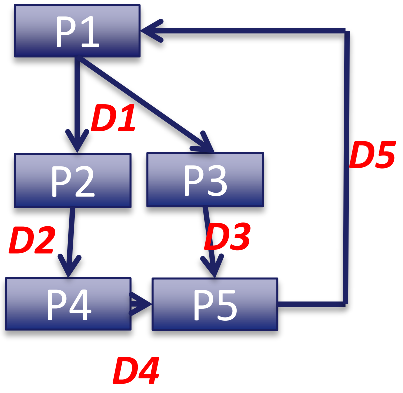
Isipdx - Ir
Sy

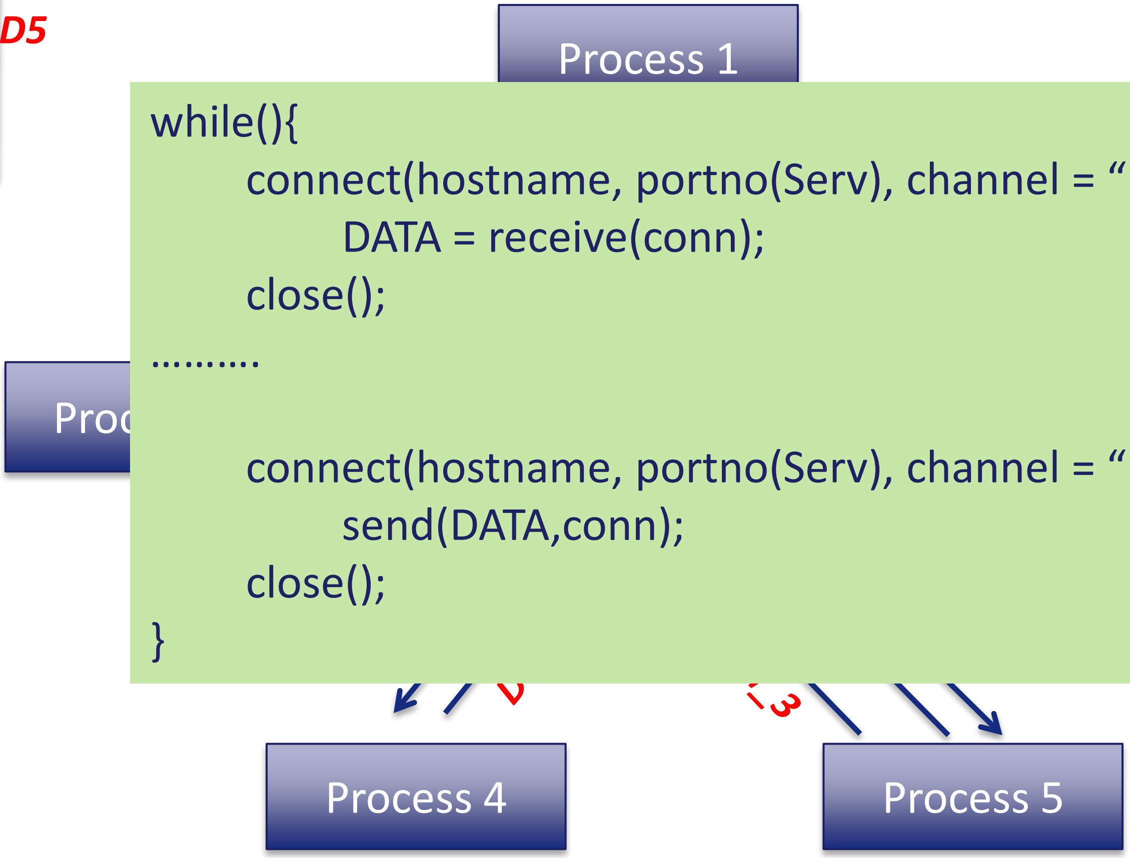
```
for(i=1; i< …. ; i++){
        connect(hostname, portno(Serv), channel = “Data_1”,’S’, ...)
                send(DATA, conn );
        close();
……….

        connect(hostname, portno(Serv), channel=“Data_5”,’R’,….)
                DATA = receive(conn);
        close();
}
```





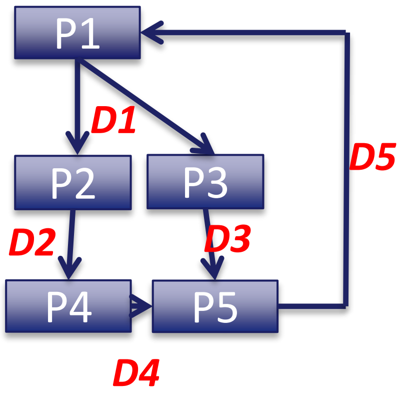
Isipdx - Inter-Process Communication and Synchronization – Example

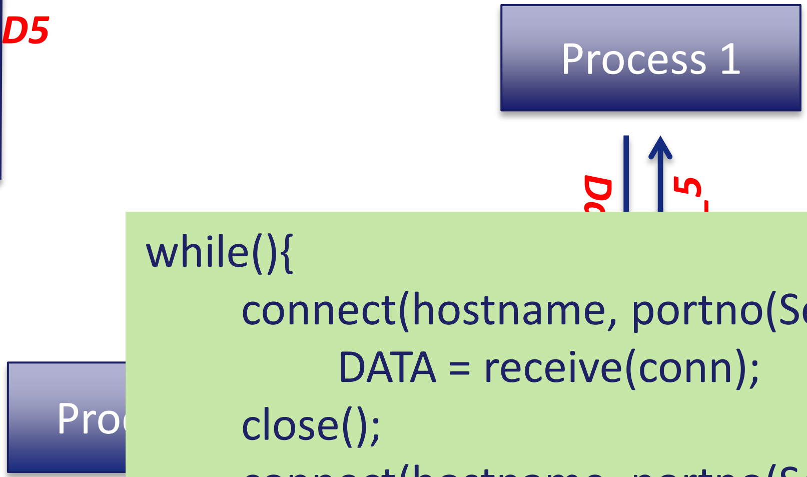




```
while(){
        connect(hostname, portno(Serv), channel = “Data_1”,’R’,….)
                DATA = receive(conn);
        close();
……….

        connect(hostname, portno(Serv), channel = “Data_2”, ‘S’….)
                send(DATA,conn);
        close();
}
```

Isipdx - Inter-Process Communication and Synchronization – Example

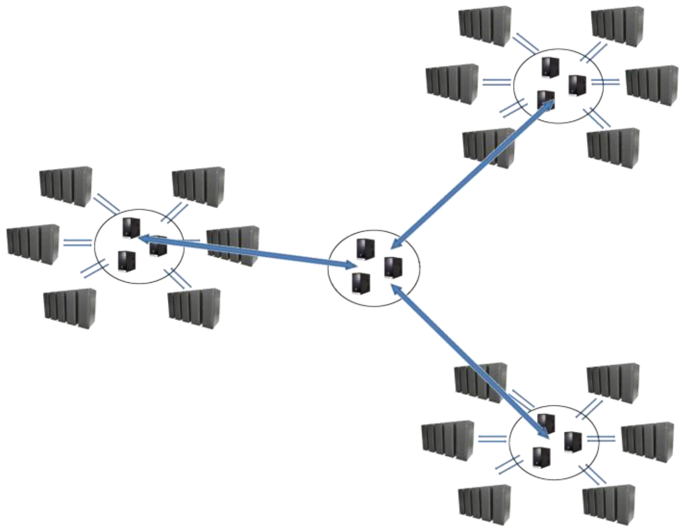




```
while(){
        connect(hostname, portno(Serv),  channel = “Data_3”, ‘R’,….)
                DATA = receive(conn);
        close();
        connect(hostname, portno(Serv),  channel = “Data_4”, ’R’, ….)
                DATA = receive(conn);
        close();


……….


        connect(hostname, portno(Serv),  channel = “Data_5”, ‘S’, ….)
                send(DATA,conn);
        close();
}
```

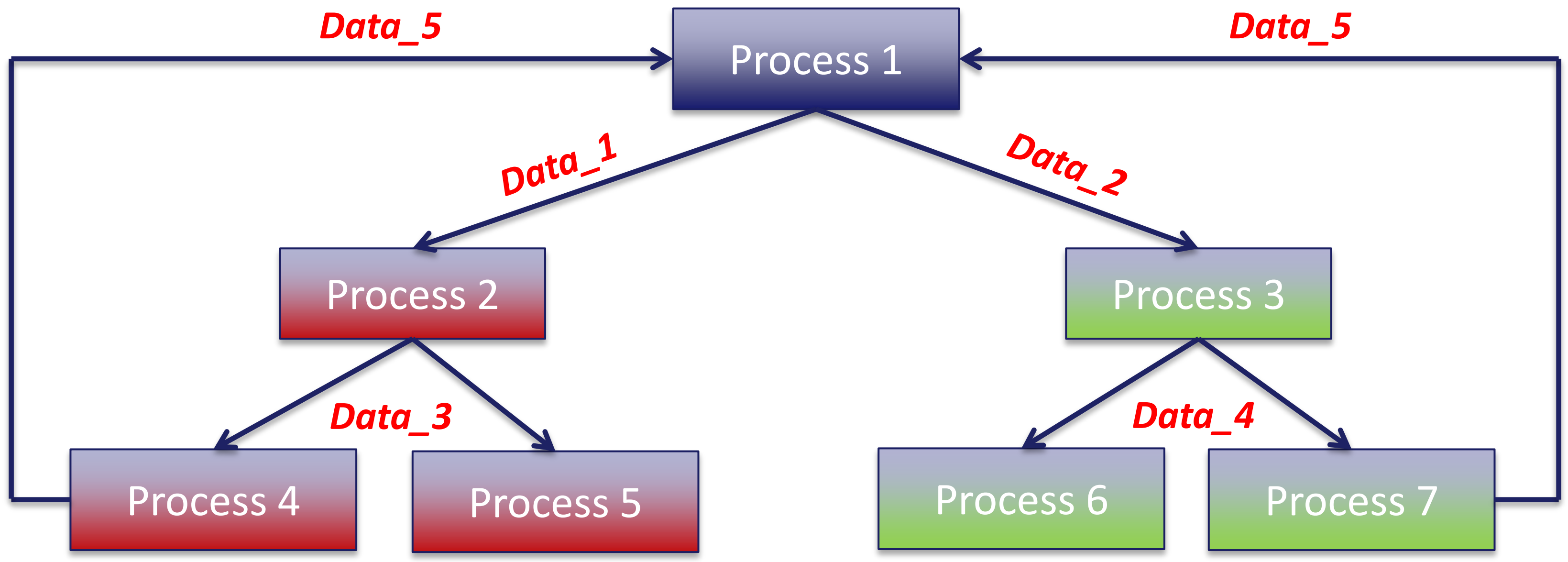
Isipdx - Inter-Process Communication and Synchronization

- Modularity on data transfer and synchronization level
  - Group processes communicating more often with each other then with other processes
  - Communication and syncing faster on the same physical unit
  - Do not need to send “unnecessary data” to other syncing centers
  - Decisions
  - Redundancy
  - etc.

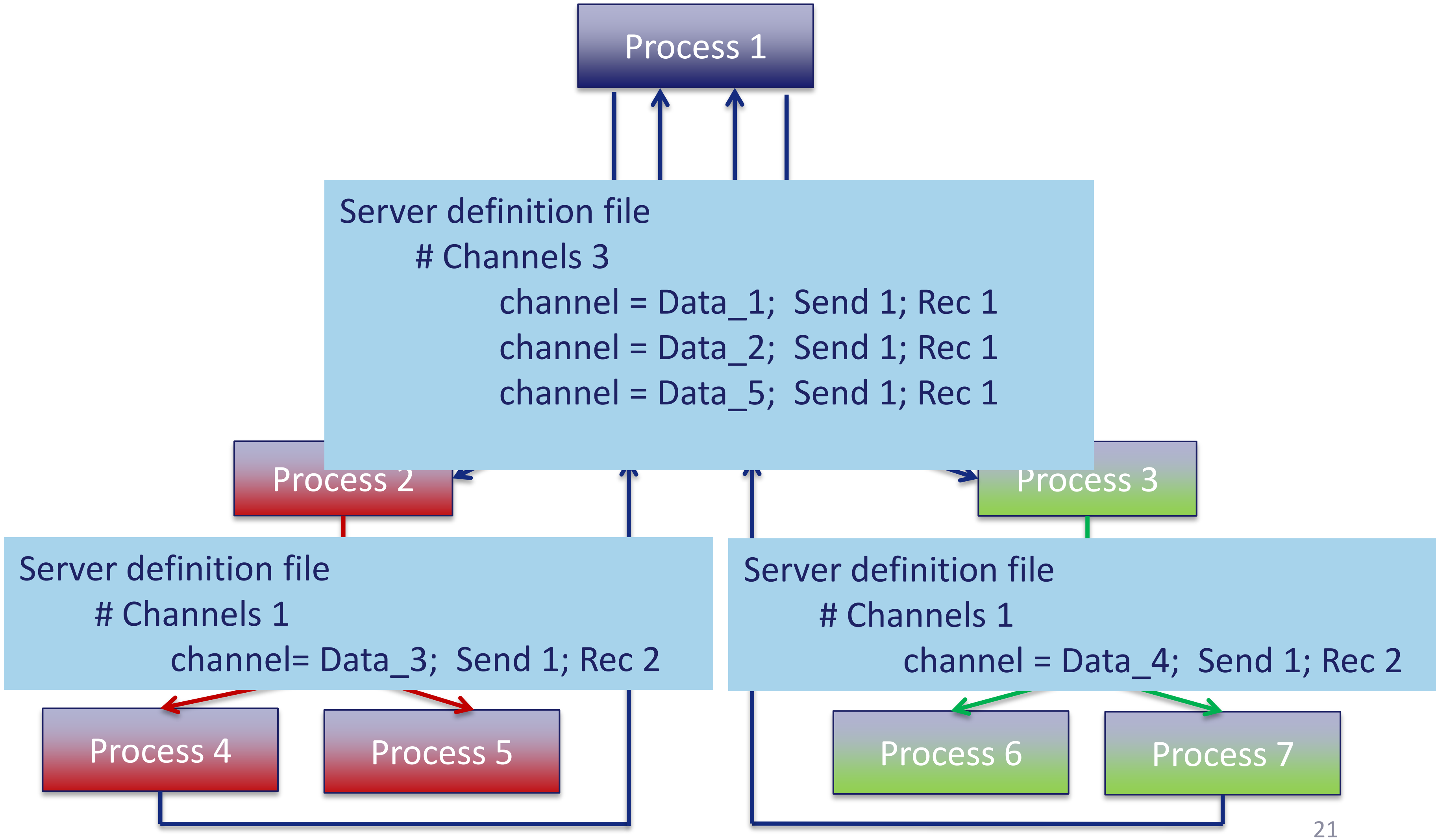


Isipdx - Inter-Process Communication and Synchronization – Example

- Two branches
  - Depending on result of Process 1, execute either “left-red” or “right-green” branch

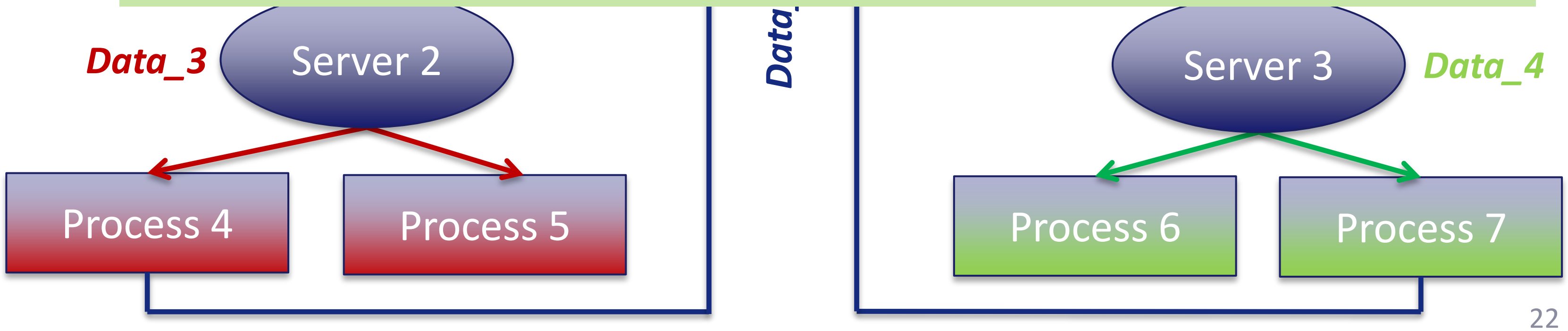


Isipdx - Inter-Process Communication and Synchronization – Example



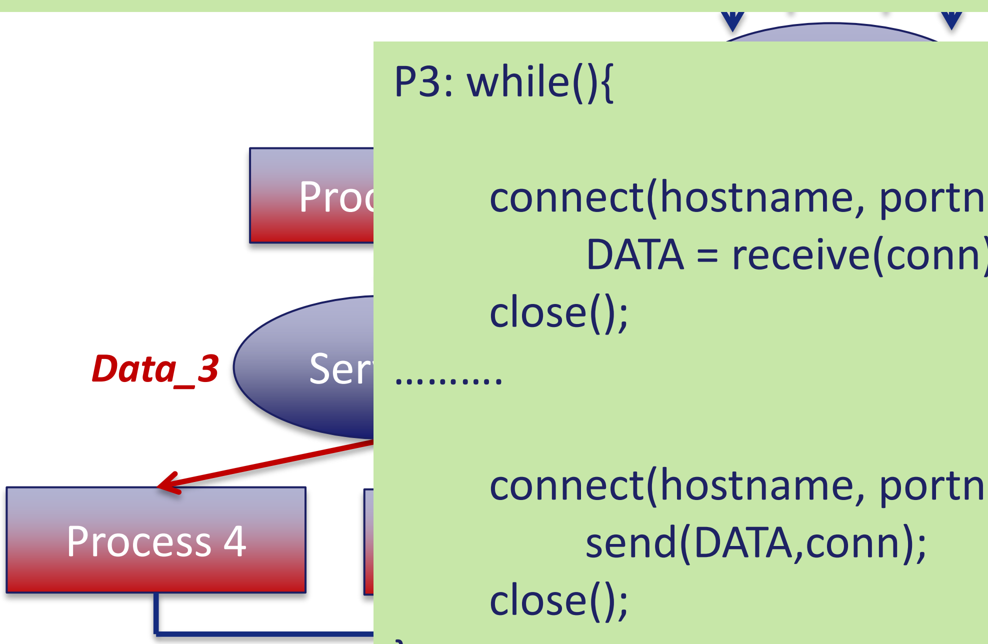
Isip

```
P1: for(i=1; i< … ; i++){

        if(condition){
                connect(hostname, portno(Serv1),  channel = “Data_1”, ‘S’,….)
                        send(DATA,conn);
                close(conn);}
        else{
                connect(hostname, portno(Serv1),  channel = “Data_2”, ‘S’,….)
                        send(DATA,conn);
                close(conn);}
……….

        connect(hostname, portno(Serv1), channel = “Data_5”, ‘R’, ….)
                DATA = receive(conn);
        close();
}
```



Jaindx – Inter Process Communication and
ple

```
P2: while(){

        connect(hostname, portno(Serv1) channel = “Data_1”, ‘R’, ….)
                DATA = receive(conn);
        close();
……….
        connect(hostname, portno(Serv2), channel = “Data_3”, ‘S’, ...)
                send(DATA,conn);
        close();
}
```

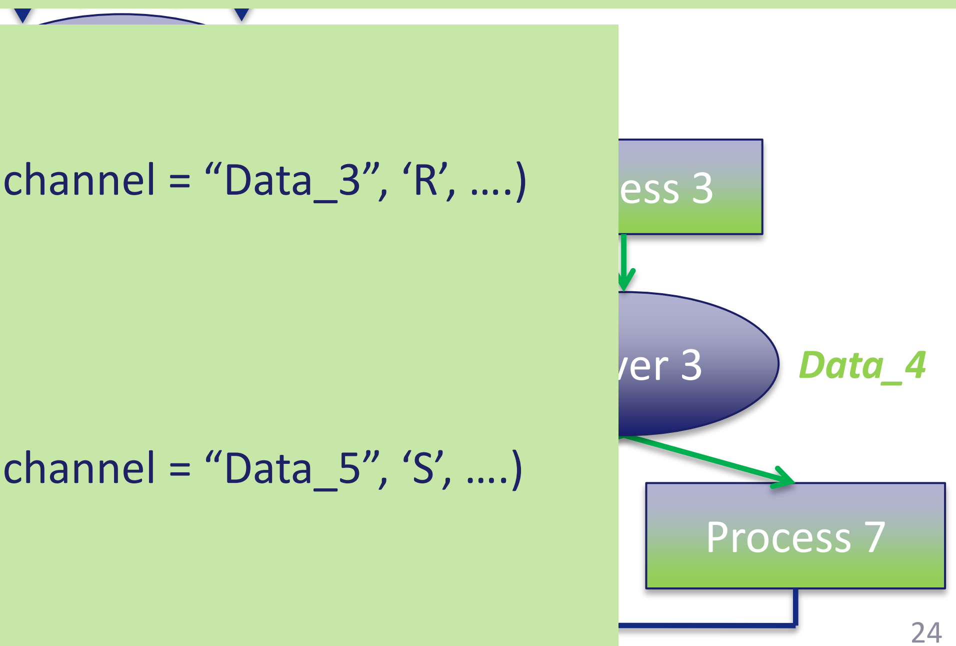


```
P3: while(){

        connect(hostname, portno(Serv1), channel = “Data_2”, ‘R’, ….)
                DATA = receive(conn);
        close();
……….

        connect(hostname, portno(Serv3), channel = “Data_4”, ‘S’, ….)
                send(DATA,conn);
        close();
}
```

Isipdx - In
Sy

```
P7: while(){

        connect(hostname, portno(Serv3), channel = “Data_4”, ’R’ ...)
                DATA = receive(conn);
        close();
……….

        connect(hostname, portno(Serv1) , channel = “Data_5”, ‘S’, ...)
                send(DATA, conn);
        close();
}
```

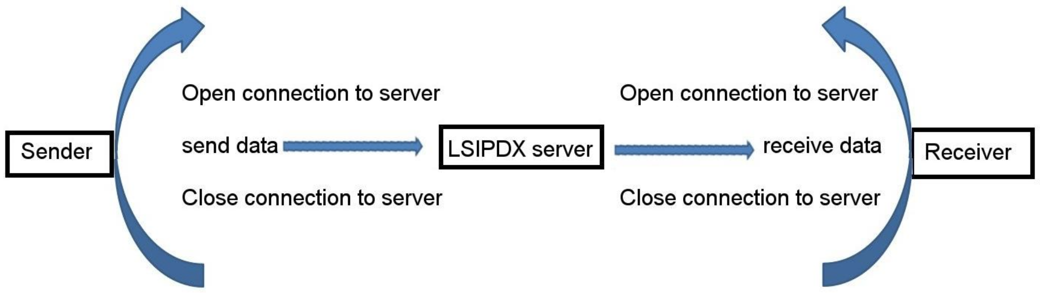
```
P4: while(){

        connect(hostname, portno(Serv2), channel = “Data_3”, ‘R’, ….)
                DATA = receive(conn);
        close();
……….

        connect(hostname, portno(Serv1), channel = “Data_5”, ‘S’, ….)
                send(DATA,conn);
        close();
}
```



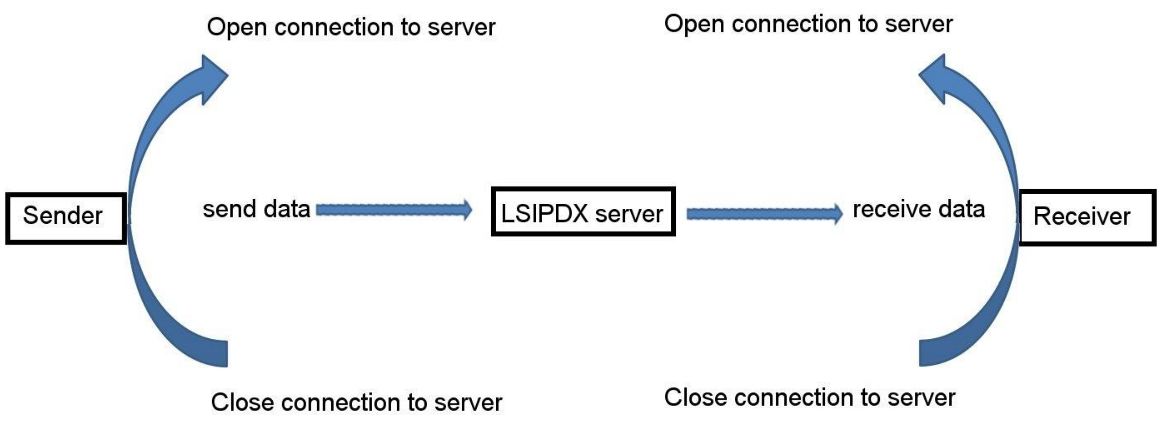
Lsipdx – KA mode

- KeepAlive mode (*KA*)
  - Specifies where to open and close connection

KA = N


Open connection to server
send data
Close connection to server
Sender
LSIPDX server
Open connection to server
receive data
Close connection to server
Receiver

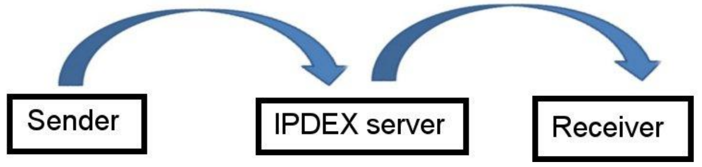
KA = Y


Open connection to server
Sender
send data
Close connection to server
LSIPDX server
Open connection to server
receive data
Close connection to server
Receiver

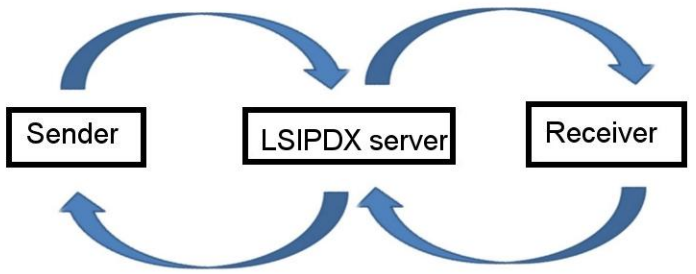
Lsipdx – ATDT mode

- Alternating Transfer / Direct Transfer mode (*ATDT*)
  - Specifies direction for data transfer
    - Only for two processes, one Sender, one Receiver

ATDT=D



ATDT=A



Performance

- Testing platform TP1
  - Cluster
  - Intel(R) Xeon(R) CPU E5-2670 (2.60GHz, 20M Cache, 2.60 GHz, 8.00 GT/s Intel® QPI)
  - CentOS release 6.4 (Final)

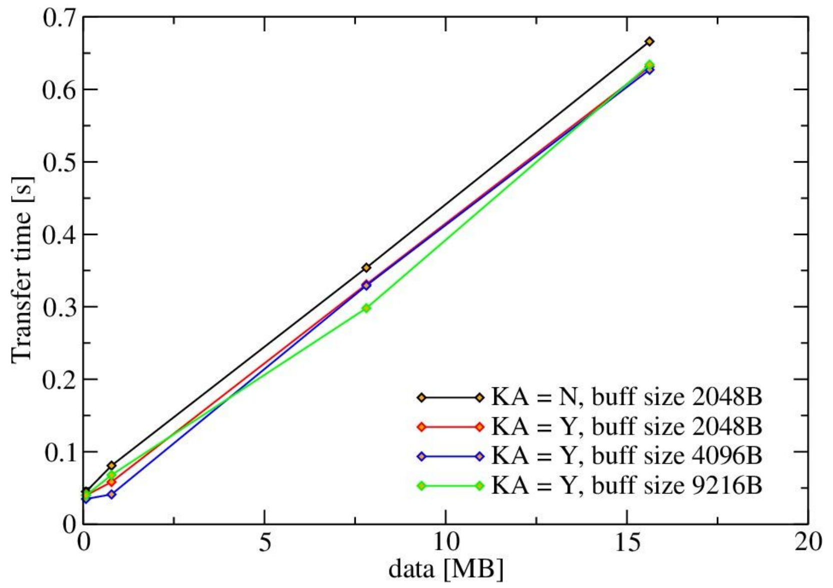
- Testing platform TP2
  - PC
  - Intel(R) Core(TM)2 Quad CPU   Q9550 @ 2.83GHz
  - openSUSE 11.1 (x86_64)

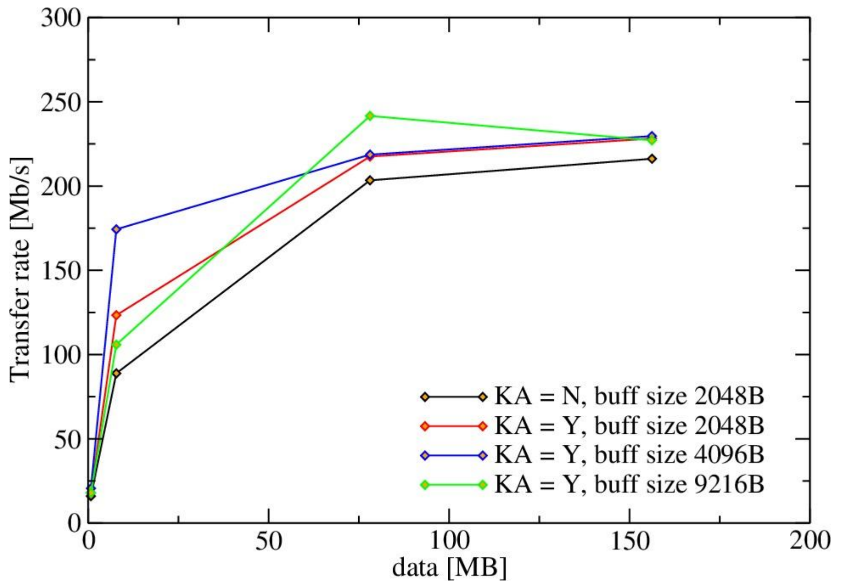
- All test use loopback (127.0.0.1) IP

- Transfer of data set containing
  - TP1:
  - TP2:

Performance

- Transfer of
  - 10,000 100,000 1,000,000 and 2,000,000 double numbers





Performance

- 100,000 cycles of sending data from a process to a process (KA=Y, ATDT=N)
  - Data set

```
H-DIR   Client_Data          2
 -C                 Name         1     18
                           Text from Client1
 -D                 numbers            1     10
                           0.0000000000000000 1.1000000000000001 2.2000000000000002 3.3000000000000003
4.4000000000000004 5.5000000000000000 6.6000000000000005 7.7000000000000011 8.8000000000000007
9.9000000000000004
```

- "useful" data
  - Text from Client1
  - 10 double numbers

- TP1: # of exchanges ~16,000/s
- TP2: # of exchanges ~16,000 – 20,000/s

Conclusion

- Two OSS tools for Inter-Process Communication (IPC) and Synchronization
  - under development
  - **libm3l** - Data Protocol for data transfer over TCP/IP sockets
  - **lsipdx** - Inter-process data transfer control and synchronization
  - **Requirements**
    - **Flexible data protocol**
      - manipulation and transfer through socket
    - **Control and synchronize data transfer**
      - Arbitrary number of processes and communication lines
      - Data transfer is independent of data set content
    - **Modularity and flexibility**
      - Possibility to have more synchronization centers/layer
      - Minimal or no changes in data flow if additional data required
    - **Easy to embed and use with client processes**
      - Minimal changes in clients source code
      - Does not pose any special requirement on execution of client processes
  - **Software platform**
    - Primarily Unix/Linux (POSIX compliant)
    - ANSI-C programming language (C99)
  - **Tested**
    - Gentoo, openSUSE 11.1 (x86_64), CentOS release 5.5 (Final)) and 6.4 (Final)
      - One specific application required libm3l @ Windows XP and Widows7 (not part of distribution)
    - Laptop, PC, Cluster
    - Valgrind memory check
  - **Available** @ **www.github.com/libm3l** under **LGPL** license

Thank you!