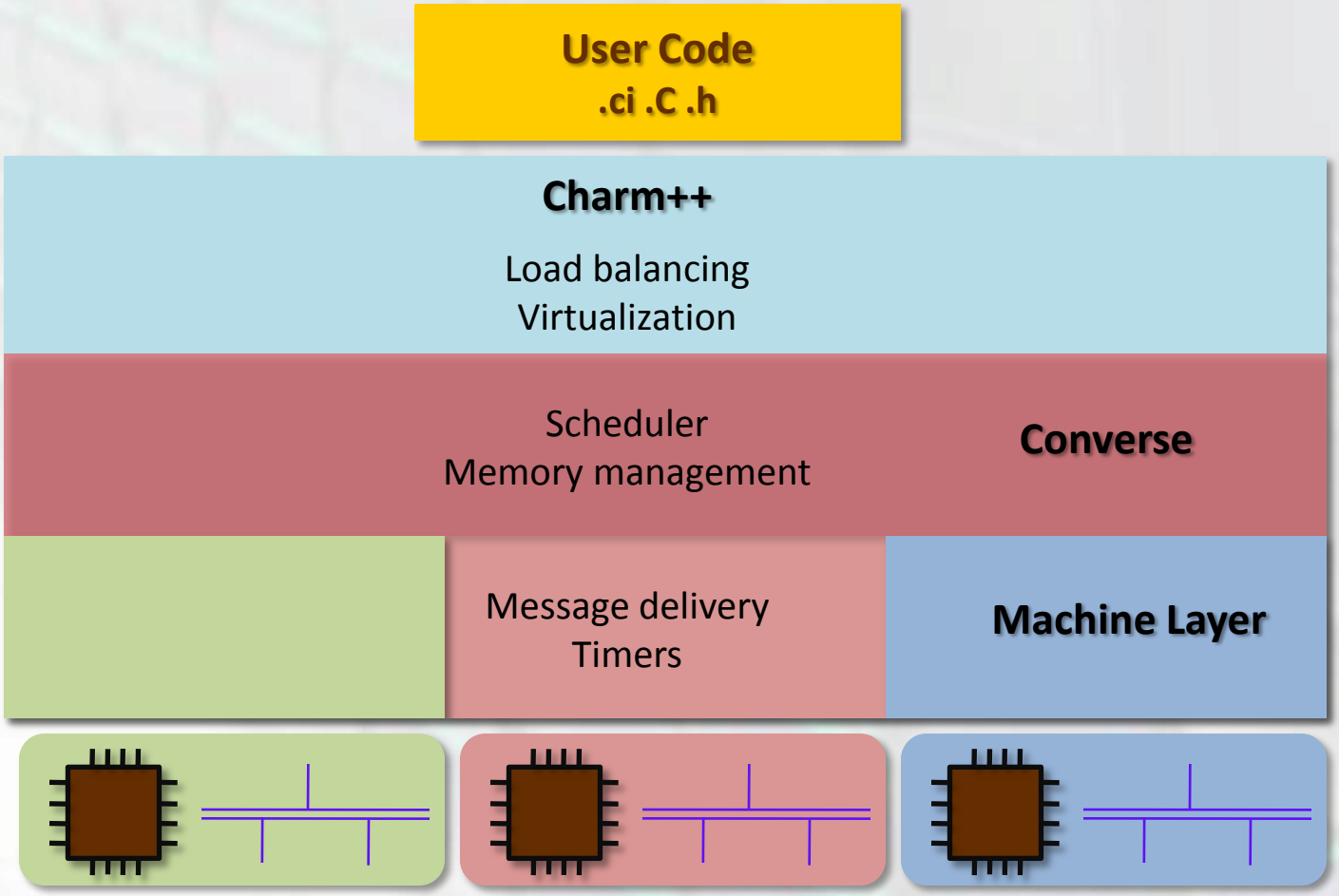
Porting Charm++ to a New System

Writing a Machine Layer

Sayantan Chakravorty

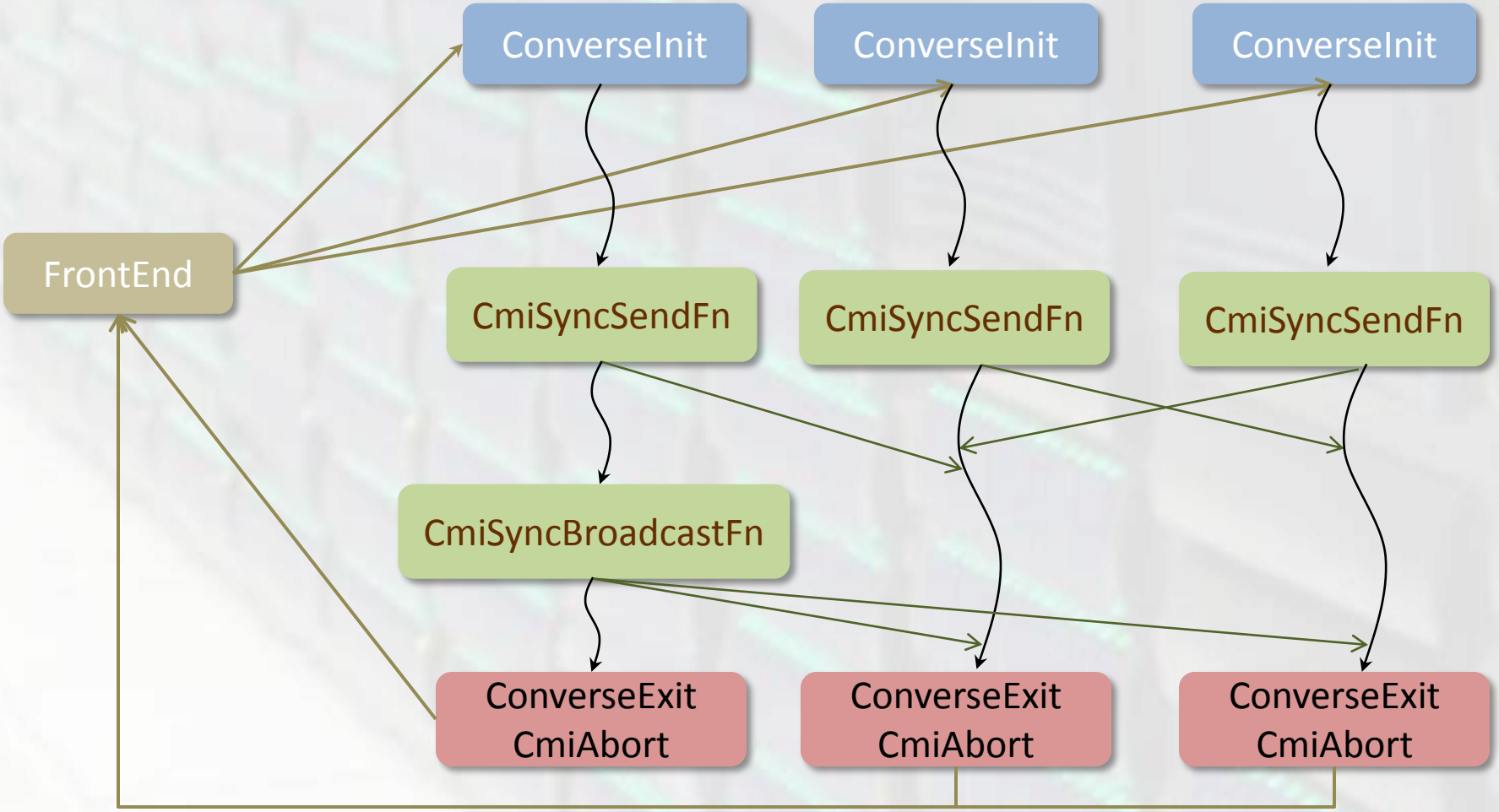
Why have a Machine Layer ?



Where is the Machine Layer ?

- Code exists in charm/src/arch/<Layer Name>
- Files needed for a machine layer
  - **machine.c** : Contains C code
  - **conv-mach.sh** : Defines environment variables
  - **conv-mach.h :** Defines macros to choose version of machine.c
  - Can produce many variants based on the same **machine.c** by varying conv-mach-<option>.*
    - **132** versions based on only **18** machine.c files

What all does a Machine Layer do?



Different kinds of Machine Layers

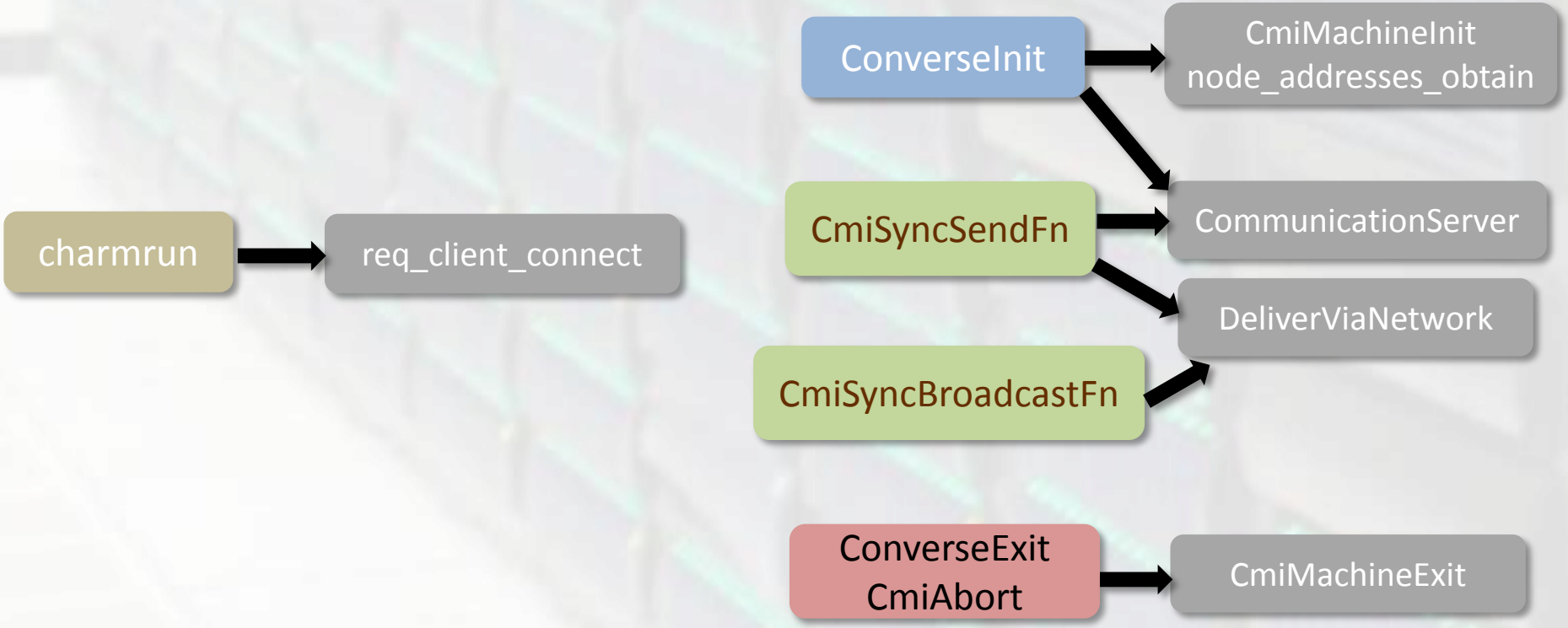
- Differentiate by **Startup** method
  - Uses lower level library/ run time
    - MPI: **mpirun** is the frontend
      - cray, sol, bluegenep
    - VMI: **vmirun** is the frontend
      - amd64, ia64
    - ELAN: **prun** is the frontend
      - axp, ia64
  - Charm run time does startup
    - Network based (net) : **charmrun** is the frontend
      - amd64, ia64,ppc
      - Infiniband, Ethernet, Myrinet

Net Layer: Why ?

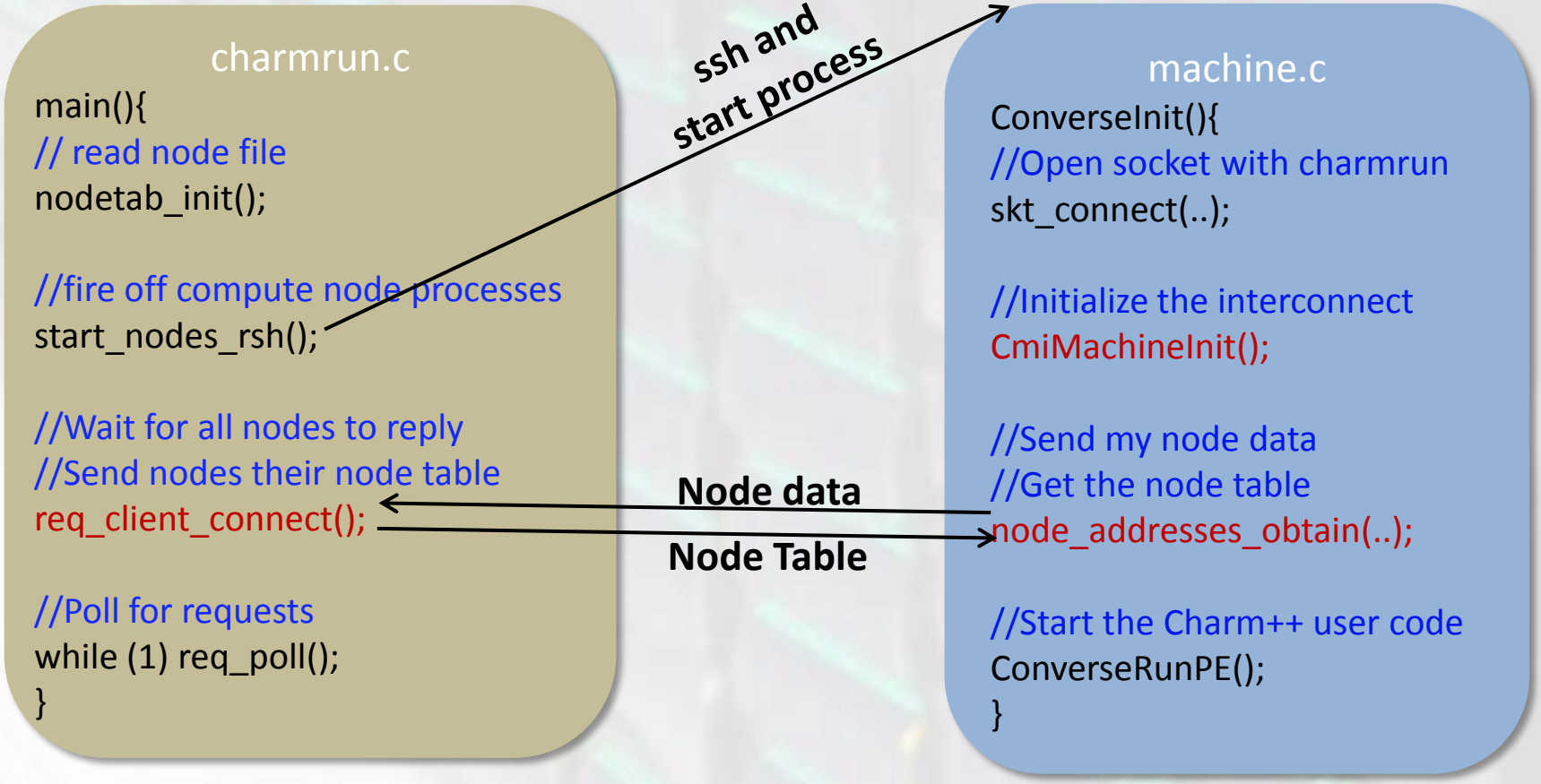
- Why do we need a startup in Charm RTS ?
  - Using a low level interconnect API, no startup provided
    - Why use low level API ?
      - Faster
        - » Why faster
          - Lower overheads
          - We can design for a message driven system
      - More flexible
        - » Why more flexible ?
          - Can implement functionality with exact semantics needed

Net Layer: What ?

- Code base for implementing a machine layer on low level interconnect API



Net Layer: Startup



Net Layer: Sending messages

```
CmiSyncSendFn(int proc,int size,char *msg){
  //common function for send
  CmiGeneralSend(proc,size,`S',msg);
}

CmiGeneralSend(int proc,int size,    int  freemode,
               char *data){
   OutgoingMsg ogm =   PrepareOutgoing(cs,pe,
                                size,freemode,data);
   DeliverOutgoingMessage(ogm);

   //Check for incoming messages and completed
   //sends
   CommunicationServer();
}

DeliverOutgoingMessage(OutgoingMsg ogm){
   //Send the message on the interconnect
   DeliverViaNetwork(ogm,..);
}
```

Net Layer: Exit

```
ConverseExit(){
   //Shutdown the interconnect cleanly
   CmiMachineExit();

   //Shutdown Converse
   ConverseCommonExit();

   //Inform charmrun this process is done
   ctrl_sendone_locking("ending",NULL,0,
         NULL,0);
}
```

Net Layer: Receiving Messages

- No mention of receiving messages
- Result of message driven paradigm
  - No explicit Receive calls
- Receive starts in CommunicationServer
  - Interconnect specific code collects received message
  - Calls CmiPushPE to handover message

Let’s write a Net based Machine Layer

A Simple Interconnect

- Let's make up an interconnect
  - Simple
    - Each node has a port
    - Other Nodes send it messages on that port
    - A node reads its port for incoming messages
    - Messages are received atomically
  - Reliable
  - Does Flow control itself

The Simple Interconnect AMPI

- Initialization
  - void si_init()
  - int si_open()
  - NodeID si_getid()
- Send a message
  - int si_write(NodeID node, int port, int size, char *msg)
- Receive a message
  - int si_read(int port, int size, char *buf)
- Exit
  - int si_close(int port)
  - void si_done()

Let's start

- Net layer based implementation for SI

conv-mach-si.h

```
#undef    CMK_USE_SI
#define   CMK_USE_SI    1

//Polling based net layer
#undef CMK_NETPOLL
#define CMK_NETPOLL   1
```

conv-mach-si.sh

```
CMK_INCDIR="-I/opt/si/include"
CMK_LIBDIR="-I/opt/si/lib"
CMK_LIB="$CMK_LIBS  –lsi"
```

Net based SI Layer

machine-si.c

#include “si.h”

CmiMachineInit

DeliverViaNetwork

CommunicationServer

CmiMachineExit

machine-dgram.c

#if CMK_USE_GM
#include "machine-gm.c“
#elif CMK_USE_SI
#include “machine-si.c”
#elif …

machine.c

//Message delivery
#include “machine-dgram.c”

Initialization

machine-si.c

```
NodeID si_nodeID;
int si_port;

CmiMachineInit(){
 si_init();  si_port = si_open();
 si_nodeID = si_getid();
}
```

charmrun.c

```
void req_client_connect(){
//collect all node data
for(i=0;i<nClients;i++){
 ChMessage_recv(req_clients[i],&msg);
 ChSingleNodeInfo *m=msg->data;
 #ifdef CMK_USE_SI
nodetab[m.PE].nodeID = m.info.nodeID
nodetab[m.PE].port = m.info.port
#endif
}
//send node data to all
for(i=0;i<nClients;i++){
  //send nodetab on req_clients[i]
}
```

ne.c

```
es;
ain(){

  me.info.nodeID = si_nodeID;
  me.info.port = si_port;
#endif
//send node data to chamrun
ctrl_sendone_nolock("initnode",&me,
                sizeof(me),NULL,0);
//receive and store node table
ChMessage_recv(charmrun_fd, &tab);
for(i=0;i<Cmi_num_nodes;i++){
   nodes[i].nodeID = tab->data[i].nodeID;
   nodes[i].port = tab->data[i].port;
}
```

Messaging: Design

- Small header with every message
  - contains the size of the message
  - Source NodeID (not strictly necessary)
- Read the header
  - Allocate a buffer for incoming message
  - Read message into buffer
  - Send it up to Converse

Messaging: Code

machine-si.c

```
typedef  struct{
  unsigned int  size;
  NodeID nodeID;
} si_header;

void DeliverViaNetwork(OutgoingMsg
     ogm, int dest,…) {
  DgramHeaderMake(ogm->data,…);

  si_header hdr;
  hdr.nodeID =  si_nodeID;
  hdr.size = ogm->size;

  OtherNode n = nodes[dest];
   if(!si_write(n.nodeID, n.port,sizeof(hdr),
       &hdr) ){}
  if(!si_write(n.nodeID, n.port, hdr.size,
       ogm->data) ){}
}
```

machine-si.c

```
void CommunicationServer(){
 si_header hdr;
 while(si_read(si_port,sizeof(hdr),&hdr)!= 0)
 {
   void *buf = CmiAlloc(hdr.size);
   int readSize,readTotal=0;
   while(readTotal < hdr.siez){
     if((readSize= si_read(si_port,hdr.size,buf)
        ) <0){}
     readTotal += readSize;
   }
   //handover to Converse
  }
}
```

Exit

machine-si.c

```
NodeID si_nodeID;
int si_port;

CmiMachineExit (){
 si_close(si_port);
 si_done();
}
```

More complex Layers

- Receive buffers need to be posted
  - Packetization
- Unreliable interconnect
  - Error and Drop detection
  - Packetization
  - Retransmission
- Interconnect requires memory to be registered
  - CmiAlloc implementation

Thank You