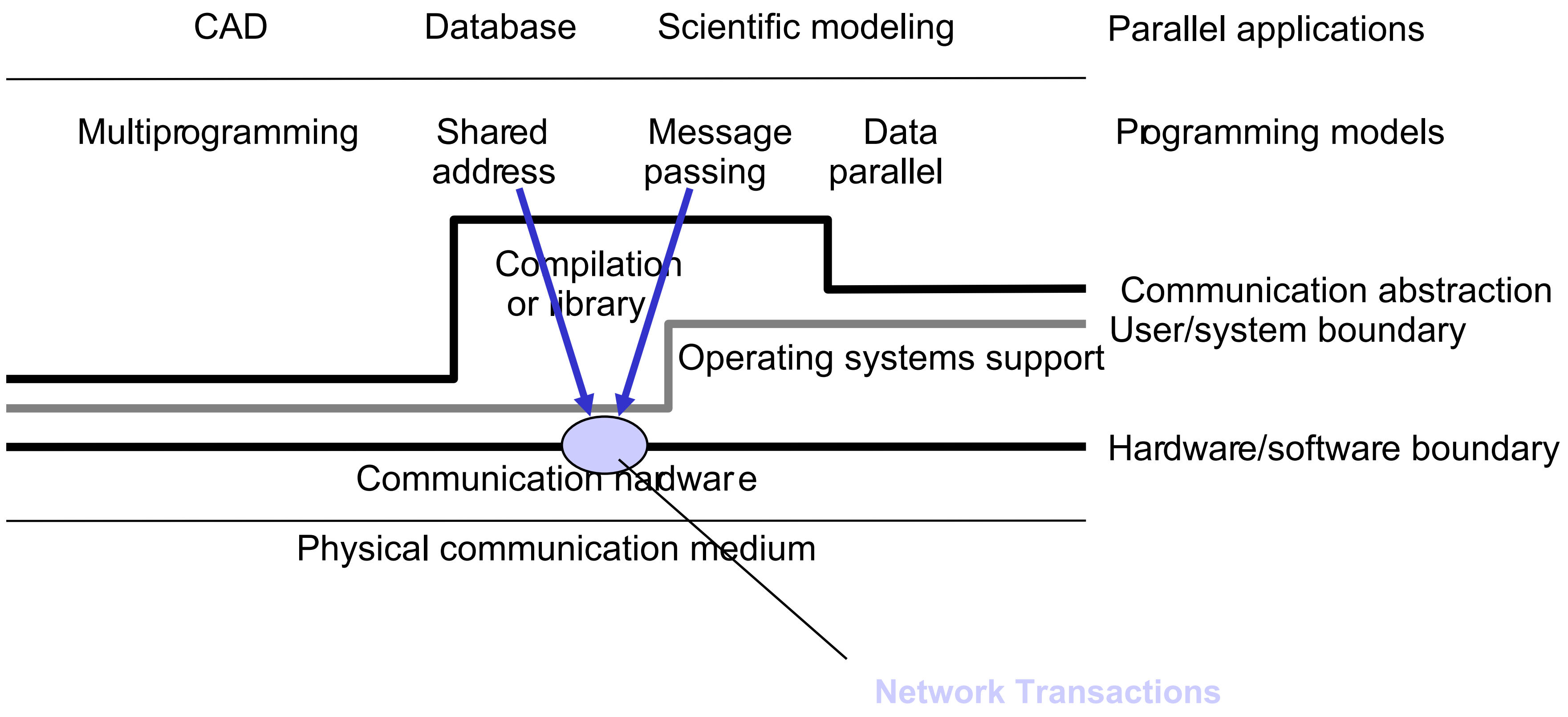
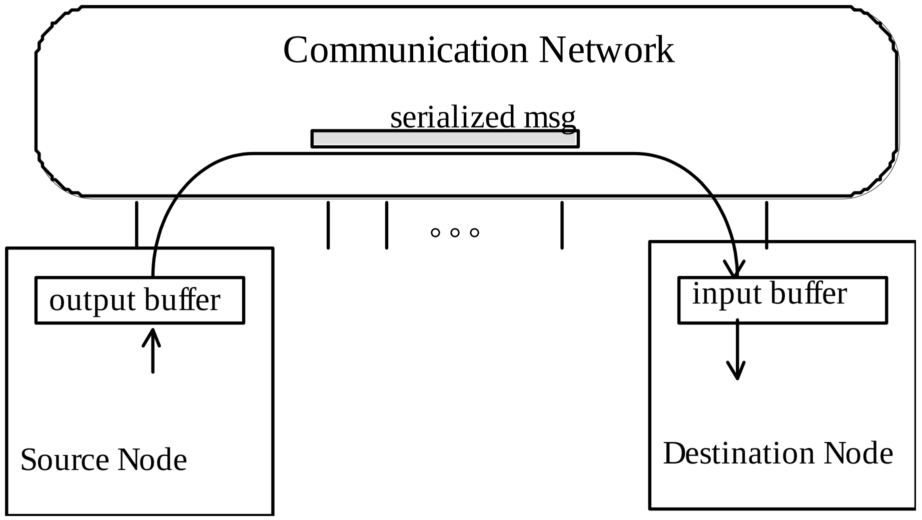
Message Passing Multiprocessors

Programming Models Realized by Protocols



Network Transaction Primitive

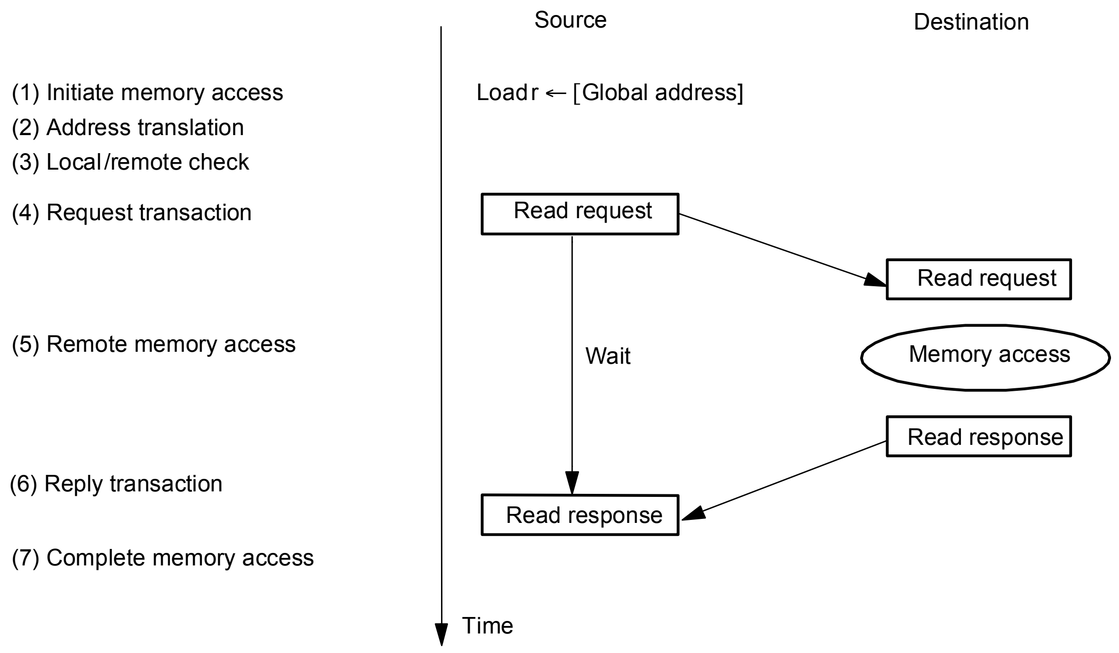


one-way transfer of information from a source output buffer to a dest. input buffer

- causes some action at the destination
- occurrence is not directly visible at source

deposit data, state change, reply

Shared Address Space Abstraction



Fundamentally a two-way request/response protocol
- writes have an acknowledgement

Issues
- fixed or variable length (bulk) transfers
- remote virtual or physical address, where is action performed?
- deadlock avoidance and input buffer full

coherent? consistent?

Message passing

Bulk transfers

Complex synchronization semantics

- more complex protocols
- More complex action

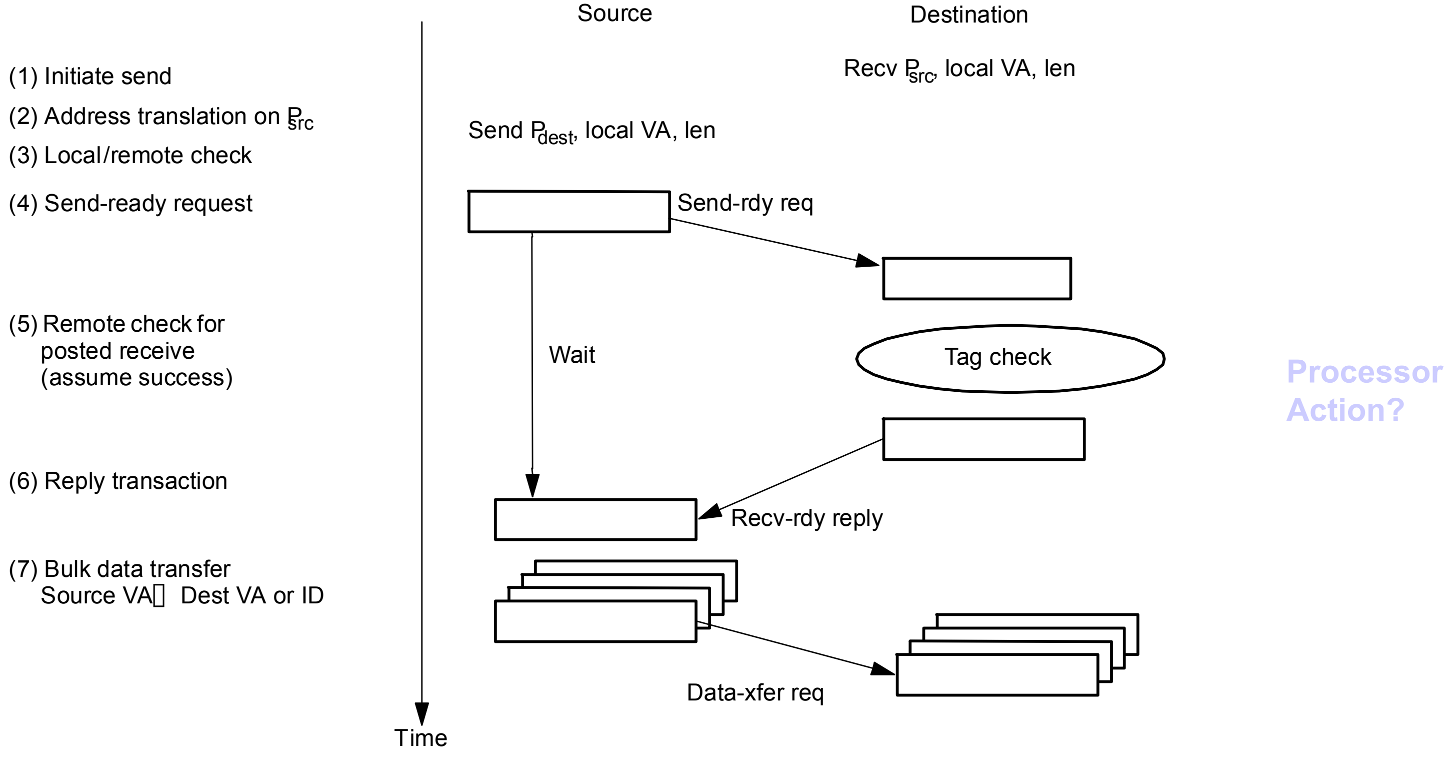
Synchronous

- Send completes after matching recv and source data sent
- Receive completes after data transfer complete from matching send

Asynchronous

- Send completes after send buffer may be reused

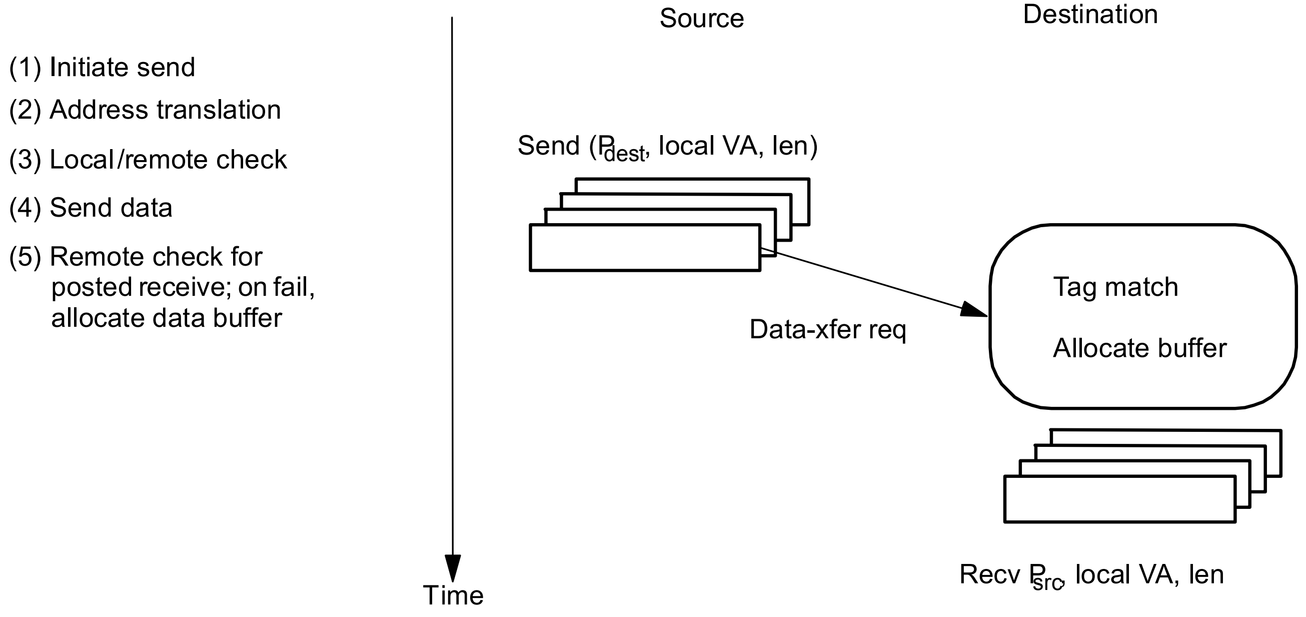
Synchronous Message Passing


Source
Destination
(1) Initiate send
(2) Address translation on P_src
(3) Local/remote check
(4) Send-ready request
(5) Remote check for posted receive (assume success)
(6) Reply transaction
(7) Bulk data transfer Source VA → Dest VA or ID
Recv P_src, local VA, len
Send P_dest, local VA, len
Send-rdy req
Wait
Tag check
Processor Action?
Recv-rdy reply
Data-xfer req
Time

Constrained programming model.

Deterministic! What happens when threads added?

Asynch. Message Passing: Optimistic

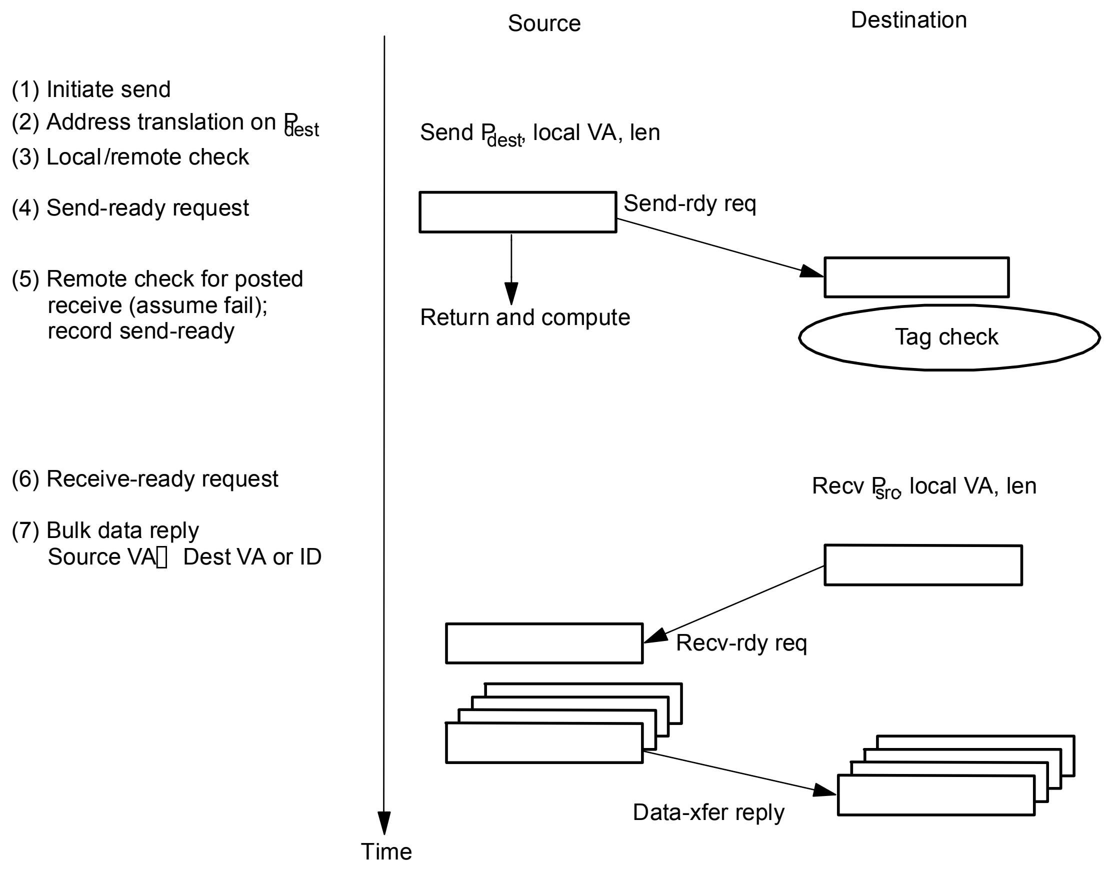


More powerful programming model

Wildcard receive => non-deterministic

Storage required within msg layer?

Asynch. Msg Passing: Conservative


Source
Destination
(1) Initiate send
(2) Address translation on P_{dest}
(3) Local/remote check
(4) Send-ready request
(5) Remote check for posted receive (assume fail); record send-ready
(6) Receive-ready request
(7) Bulk data reply Source VA → Dest VA or ID
Send P_{dest}, local VA, len
Send-rdy req
Return and compute
Tag check
Recv P_{src}, local VA, len
Recv-rdy req
Data-xfer reply
Time

Where is the buffering?

Contention control? Receiver initiated protocol?

Short message optimizations

Key Features of Msg Passing Abstraction

Source knows send data address, dest. knows receive data address

- after handshake they both know both

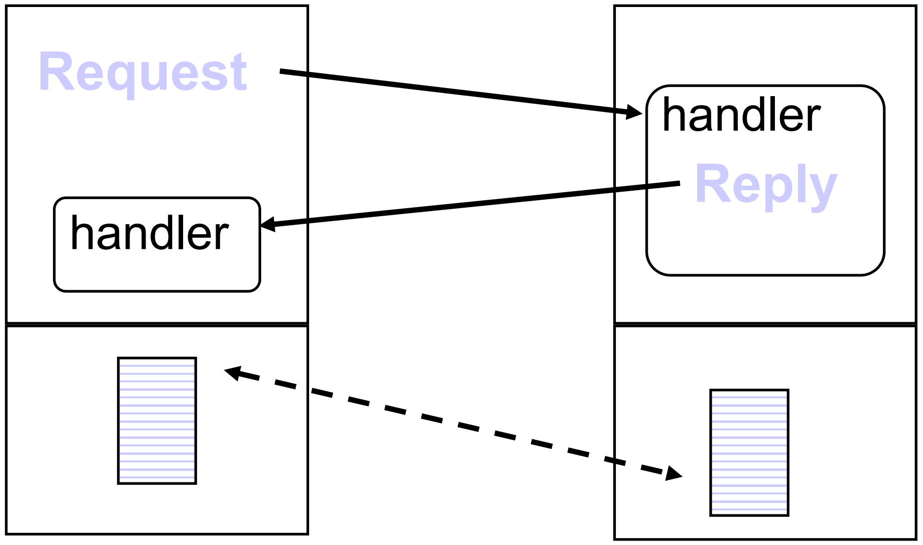
Arbitrary storage “outside the local address spaces”

- may post many sends before any receives
- non-blocking asynchronous sends reduces the requirement to an arbitrary number of descriptors
  - fine print says these are limited too

Fundamentally a 3-phase transaction

- includes a request / response
- can use optimisitic 1-phase in limited “Safe” cases
  - credit scheme

Active Messages



User-level analog of network transaction

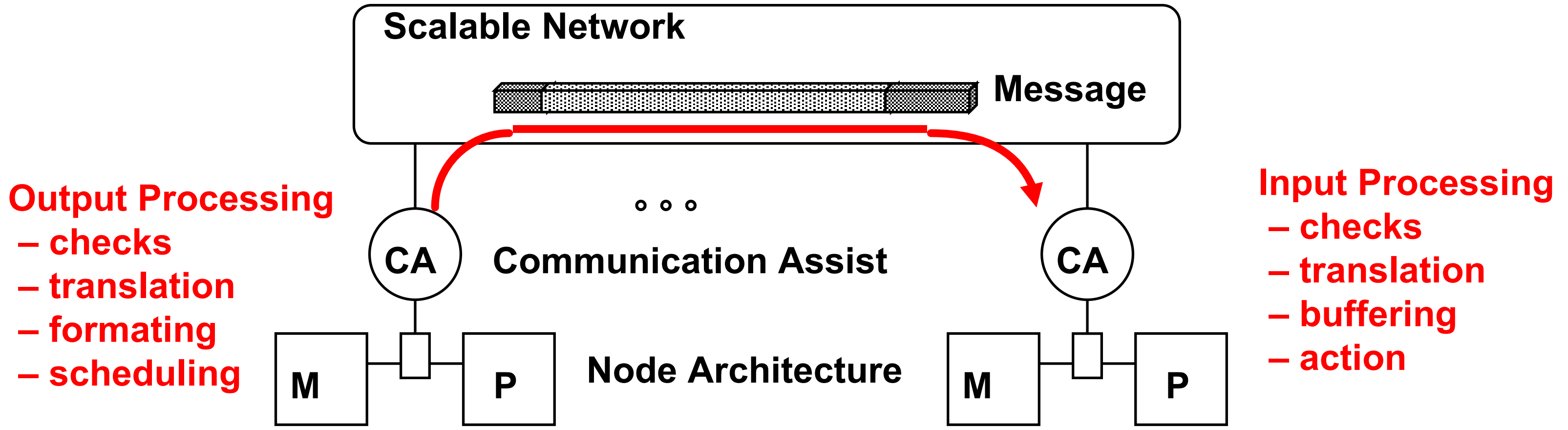
- transfer data packet and invoke handler to extract it from the network and integrate with on-going computation

Request/Reply

Event notification: interrupts, polling, events?

May also perform memory-to-memory transfer

Network Transaction Processing



Key Design Issue:

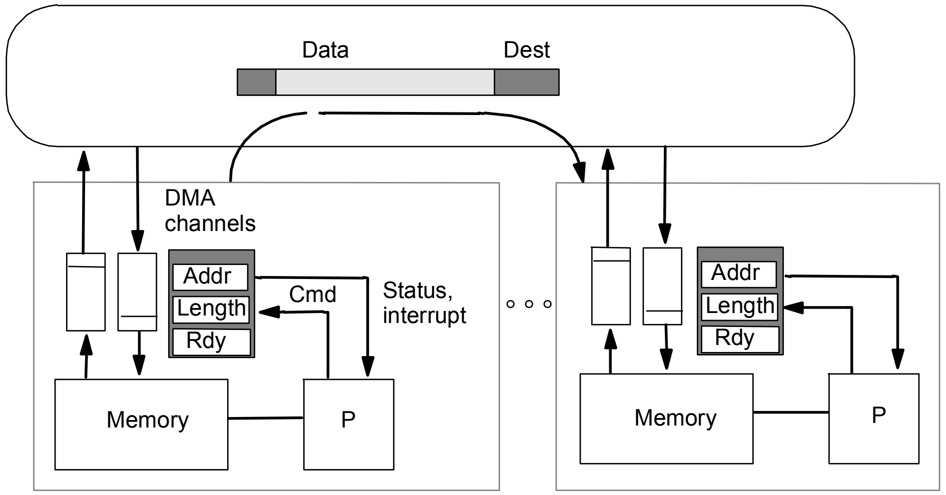
How much interpretation of the message?

How much dedicated processing in the Comm. Assist?

Hardware Capabilities of Network Interface

- Just DMA Support
  - E.g. Normal LAN interfaces, nCUBE/2, …
- Direct CPU Access, with fine-grained communication
  - CM-5, Monsoon, J-machine, …
- Dedicated Hardware
  - E.g. Intel Paragon, Myrinet, DEC PCI Memory Channel, …

Net Transactions: Physical DMA



DMA controlled by regs, generates interrupts

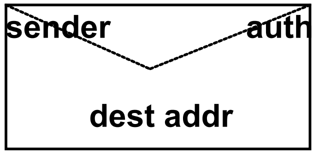
Physical => OS initiates transfers

Send-side

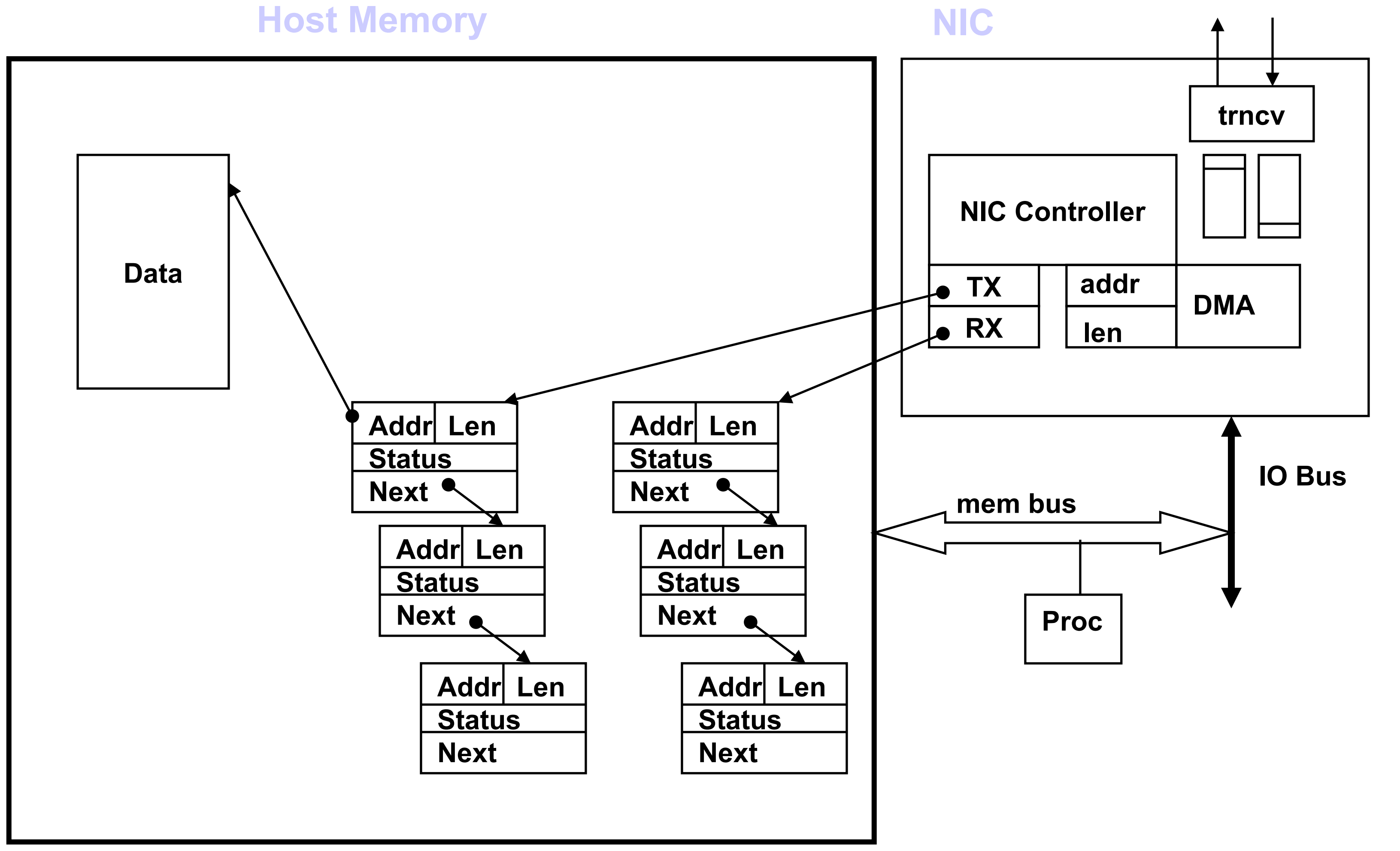
- construct system “envelope” around user data in kernel area

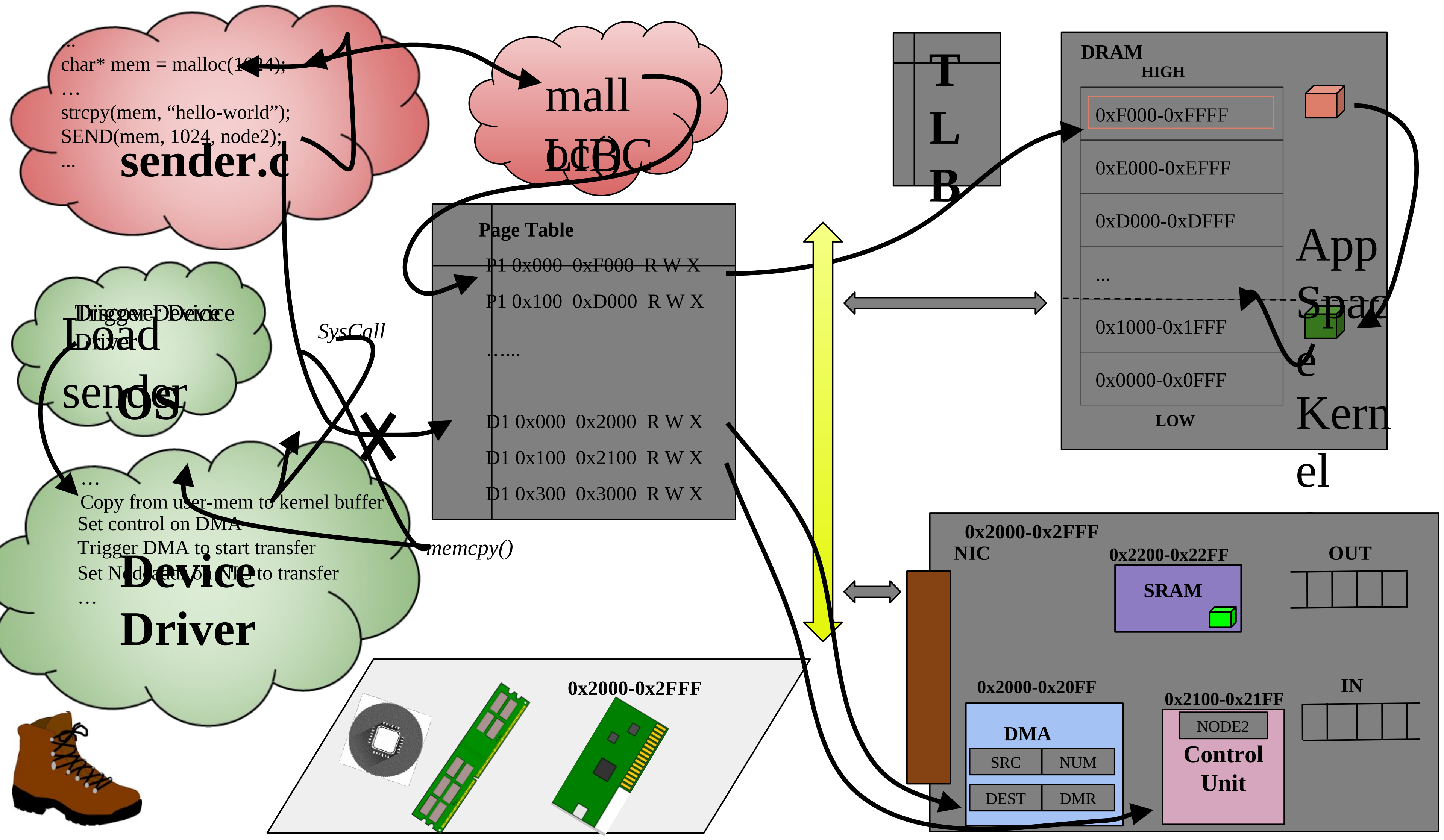
Receive

- must receive into system buffer, since no interpretation inCA



Conventional LAN Network Interface





```
…
char* mem = malloc(1024);
…
strcpy(mem, “hello-world”);
SEND(mem, 1024, node2);
…
```

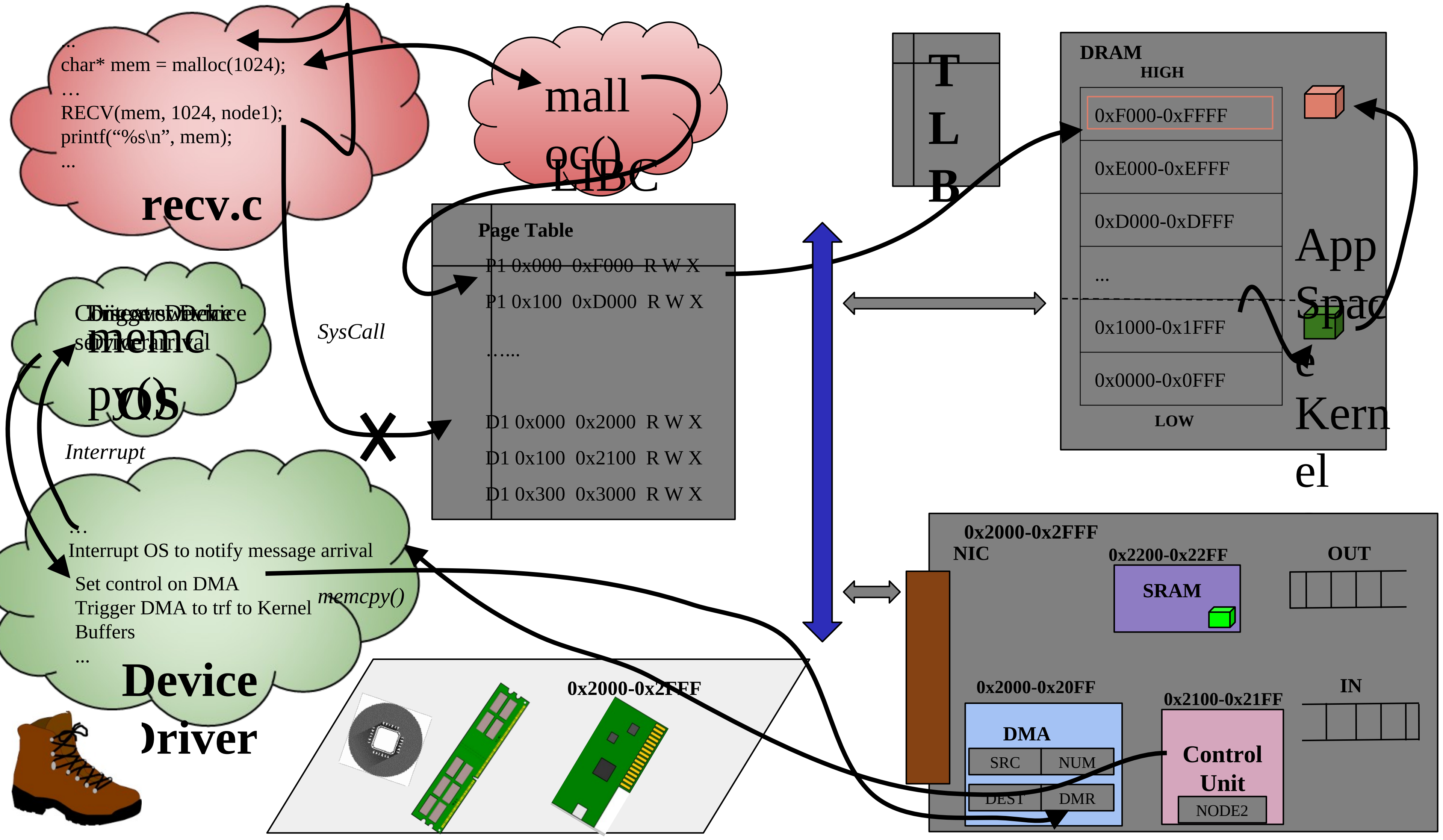
…
Copy from user-mem to kernel buffer
Set control on DMA
Trigger DMA to start transfer
Set Node addr on NIC to transfer
…

Page Table

P1 0x000 0xF000 R W X
P1 0x100 0xD000 R W X
……
D1 0x000 0x2000 R W X
D1 0x100 0x2100 R W X
D1 0x300 0x3000 R W X

DRAM

0xF000-0xFFFF
0xE000-0xEFFF
0xD000-0xDFFF
…
0x1000-0x1FFF
0x0000-0x0FFF



```
…
char* mem = malloc(1024);
…
RECV(mem, 1024, node1);
printf("%s\n", mem);
…
```

…
Interrupt OS to notify message arrival
Set control on DMA
Trigger DMA to trf to Kernel Buffers
…

Software Stack

Sending

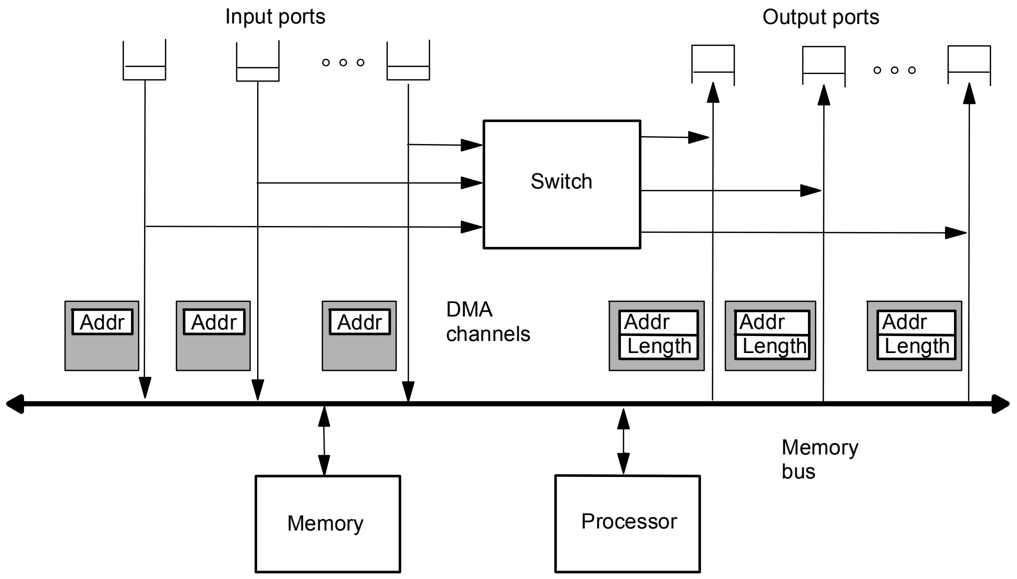
- Trap to OS - Why????
- Copy data to Pinned Memory - Why???
- Wait for DMA to become available
- Program DMA (addr, length, ….)

Receiving

- If you don’t know where to put it, interrupt host CPU
- OS programs DMA to copy to pinned memory - Why???
- On subsequent application RECEIVE call, copy from system to user buffer

Overhead can run to dozens/hundreds of Microseconds

nCUBE Network Interface



independent DMA channel per link direction

- leave input buffers always open
- segmented messages

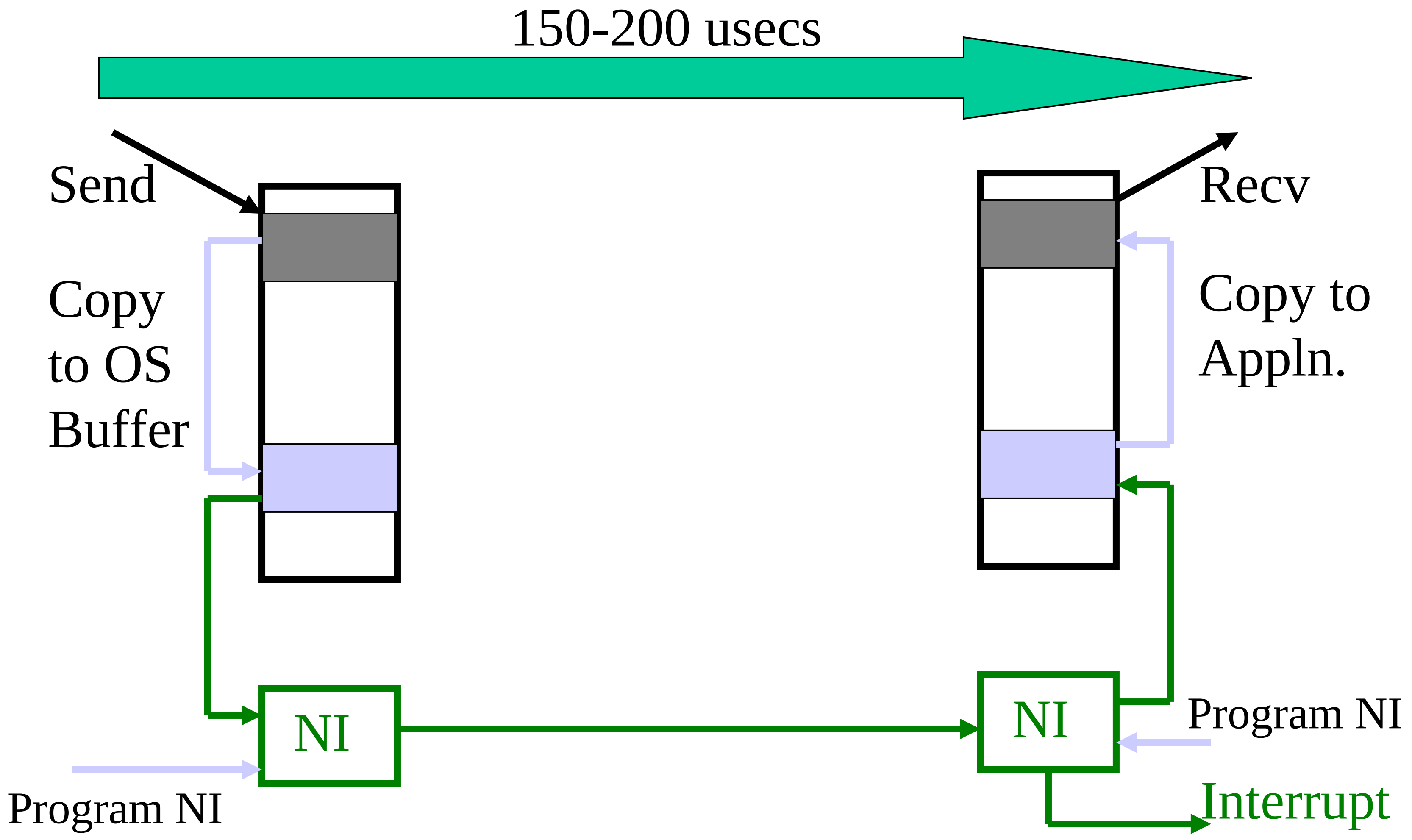
routing interprets envelope

- dimension-order routing on hypercube
- bit-serial with 36 bit cut-through

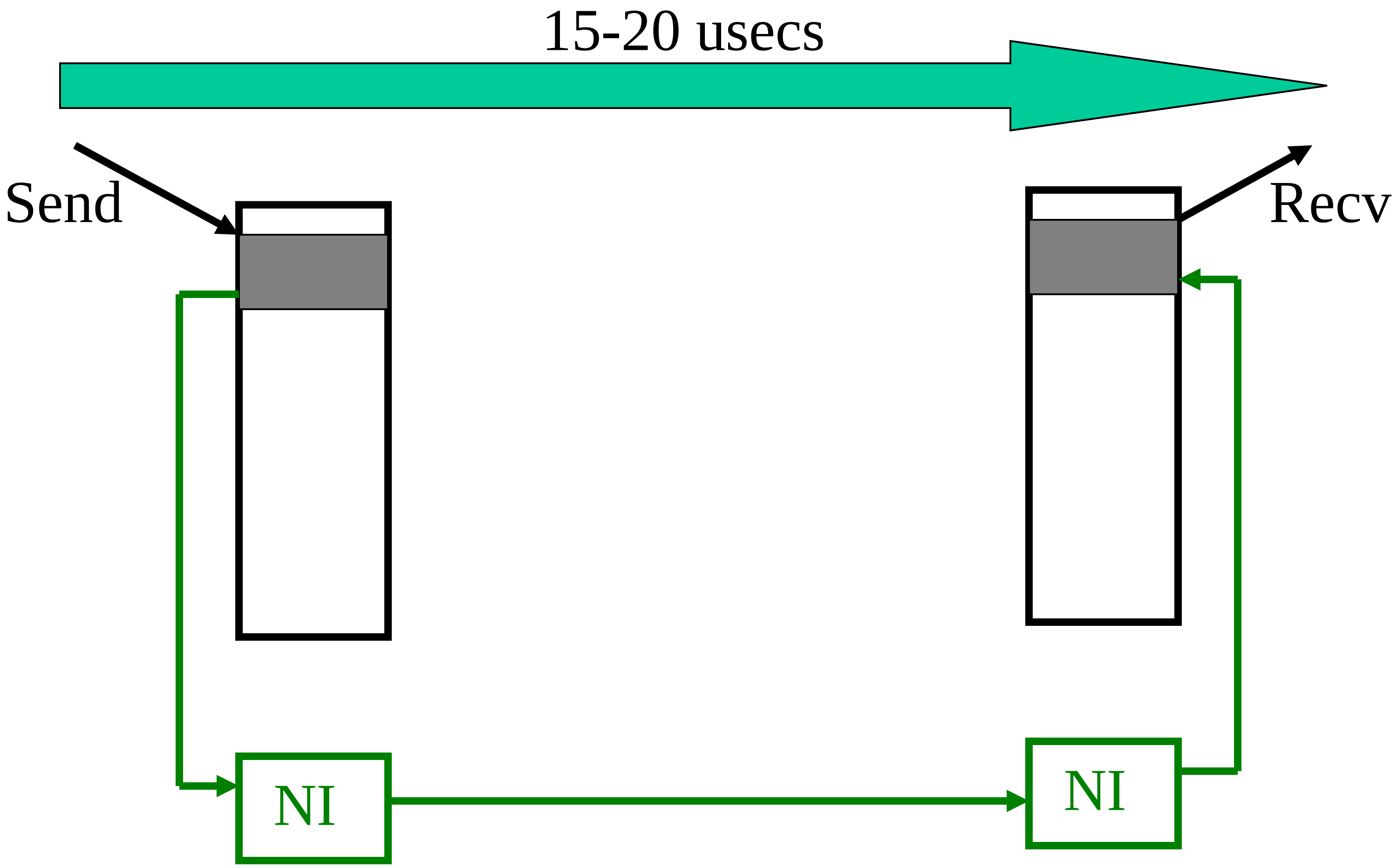
Os	**16 ins** **13 us**	**260 cy**
Or	**18** **15 us**	**200 cy**

- includes interrupt

Kernel-based Communication



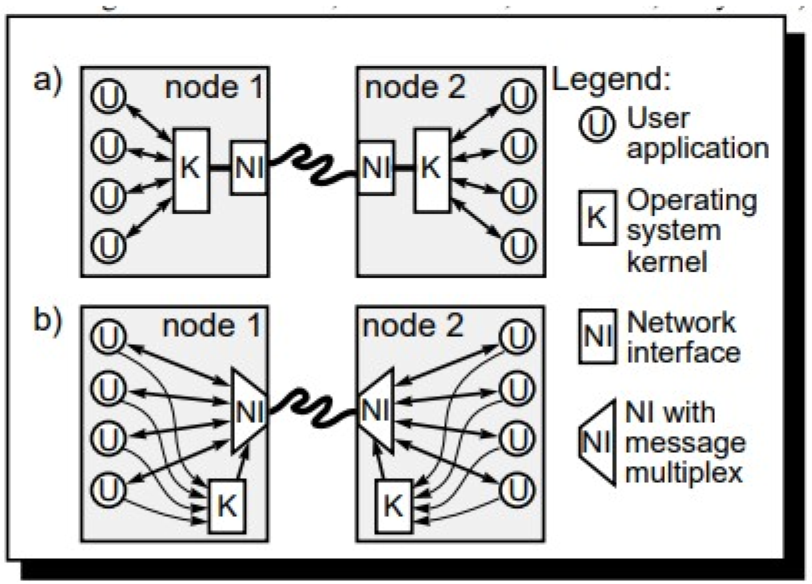
User-level Communication



How do we achieve User-Level Access?

Smart Scheduling (CM-5)

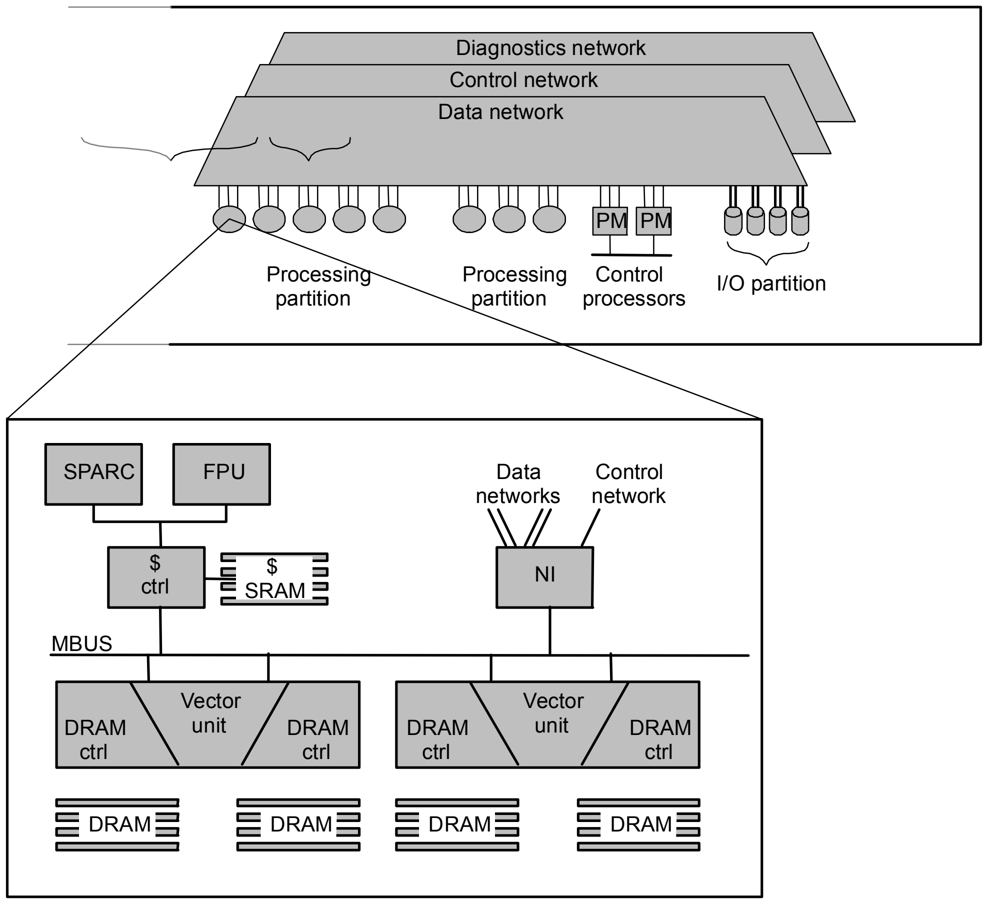
Additional Hardware (Paragon, Myrinet, DEC Memory Channel, …)



Example: CM-5

- NI sits on Memory Bus
- Input and output FIFO for each network
- NI registers are mapped into user-address space
- Processor can load/store into these registers
- NI can also be made to interrupts
- 2 data networks

Other Examples: *T integrated NI on chip, J-Machine, iWARP



Os	50 cy	1.5 us
Or	53 cy	1.6 us
interrupt		10us

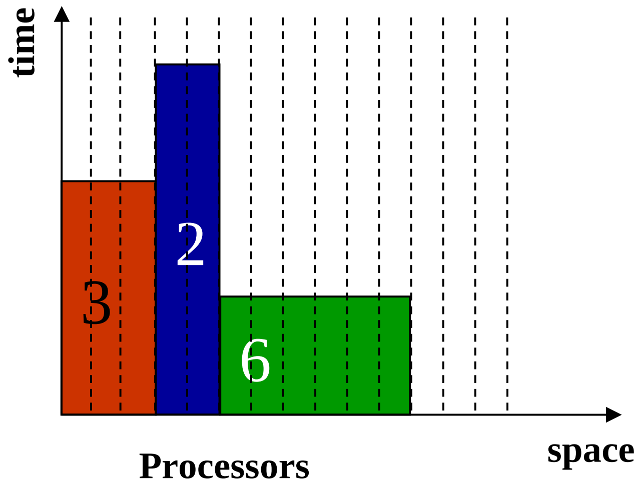
Coordinated Scheduling (Co-Scheduling)

We need to ensure you can only send to or receive from whoever you are authorized to do so.

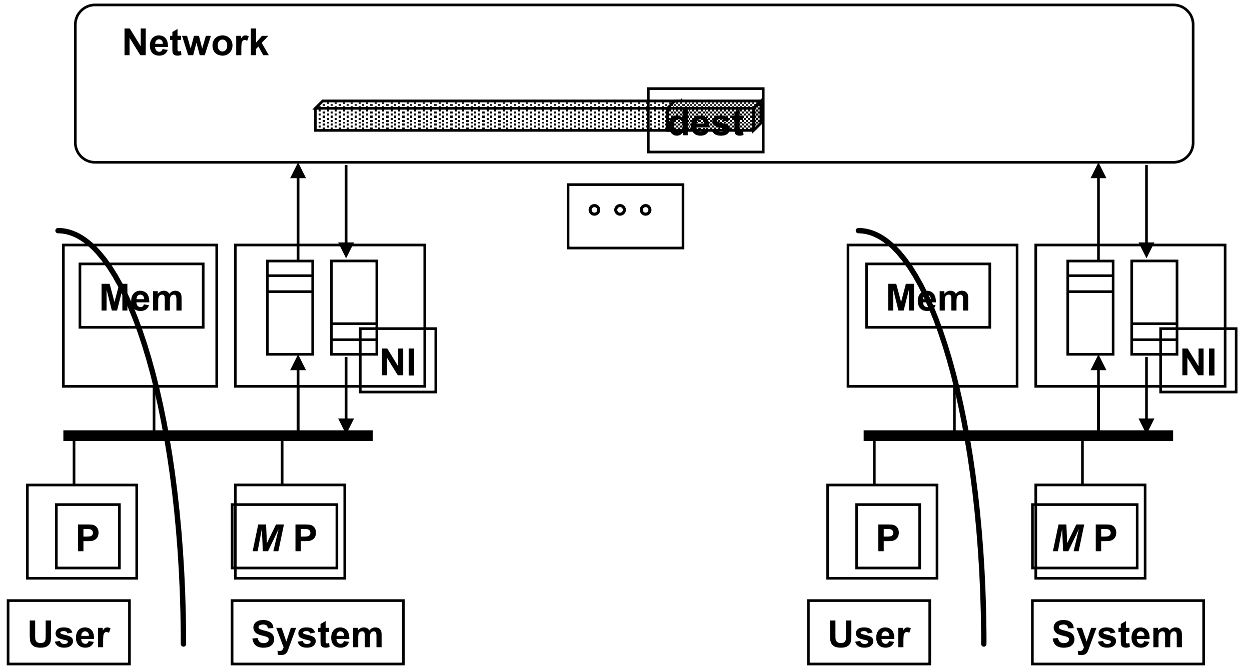
Source and Destination should have counterparts of same application scheduled in their respective nodes. i.e. coordinated context-switching

But that does not suffice!!!

- We also need coordinated Network context switching
- CM-5 does network context switching through "All Fall Down"



User-level Messaging with Additional H/W (But NOT specialized H/W)



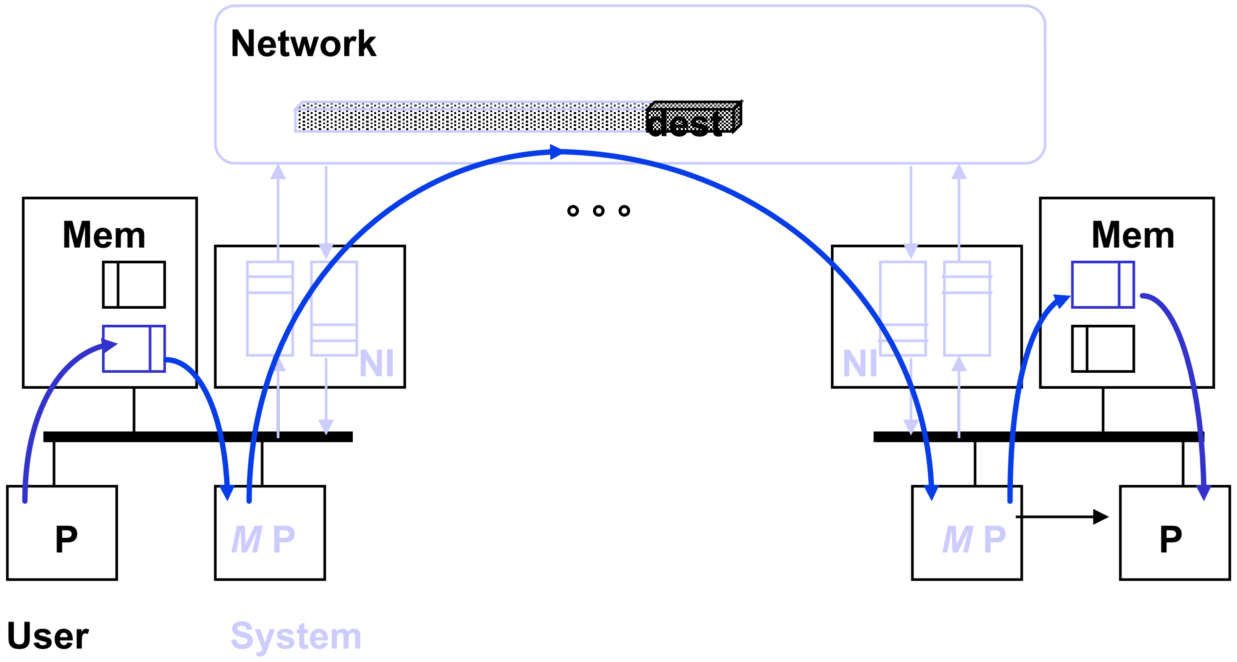
General Purpose processor performs arbitrary output processing (at system level)

General Purpose processor interprets incoming network transactions (at system level)

User Processor <–> Msg Processor share memory

Msg Processor <–> Msg Processor via system network transaction

Levels of Network Transaction



User Processor stores cmd / msg / data into shared output queue

- must still check for output queue full (or make elastic)

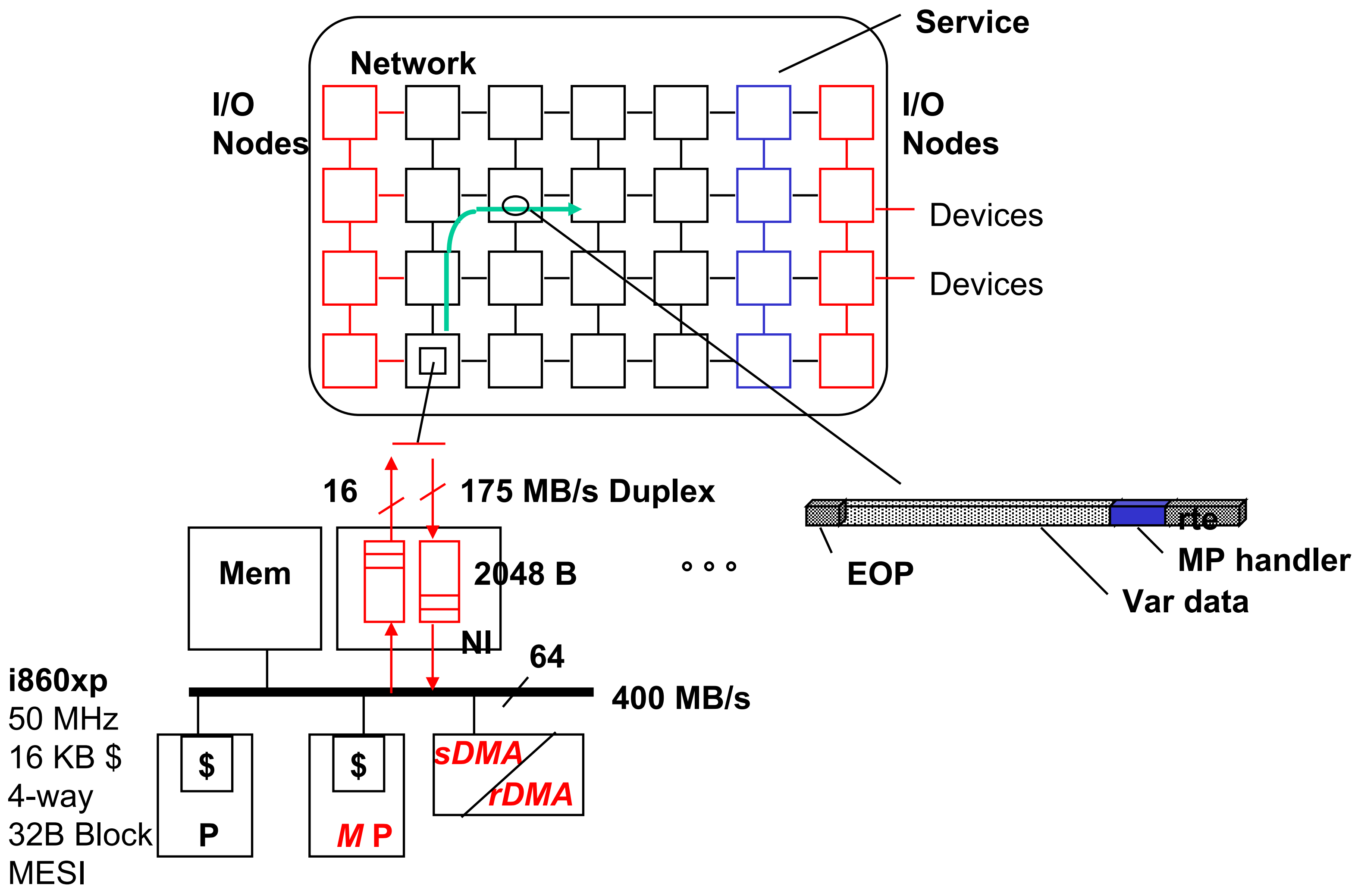
Communication assists make transaction happen

- checking, translation, scheduling, transport, interpretation

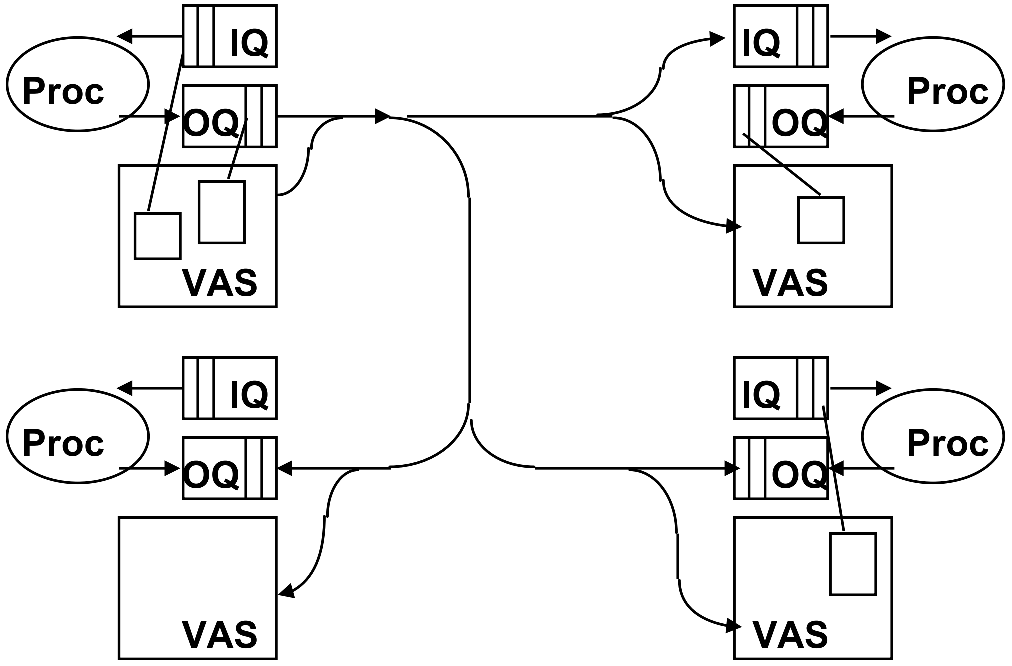
Effect observed on destination address space and/or events

Protocol divided between two layers

Example: Intel Paragon



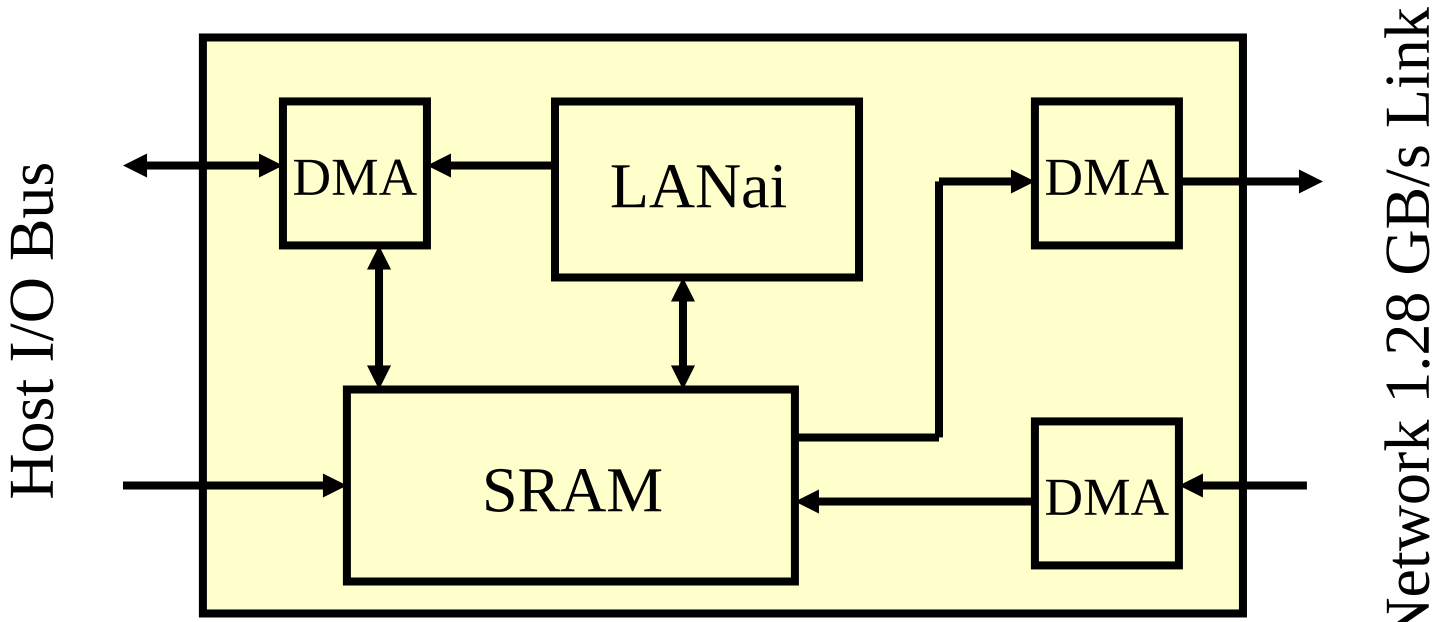
User Level Abstraction



Any user process can post a transaction for any other in protection domain

- communication layer moves OQ_{src} –> IQ_{dest}
- may involve indirection: VAS_{src} –> VAS_{dest}

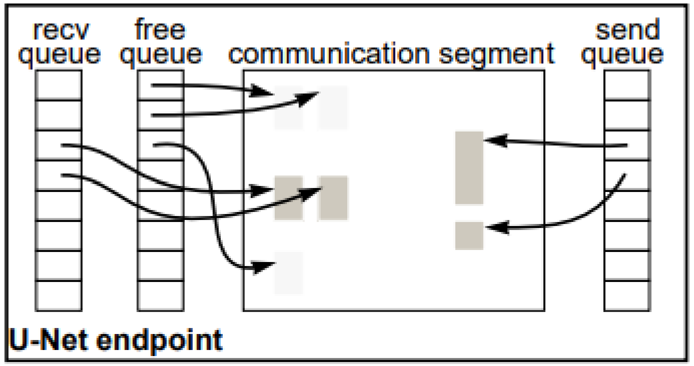
Specialized NICs (Myrinet)



Data Structures (End Point)

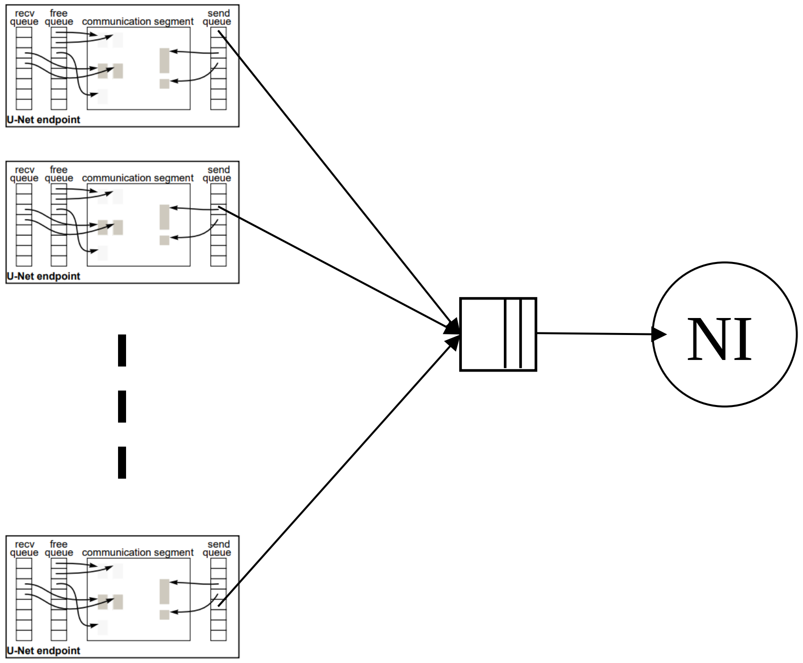
Where should each of the queues be maintained?

- Send – On the NI
- Receive – On the Host
- Free – On the Host



Need to use memory locations for synchronization

Scalability with # of endpoints (Doorbell mechanism + Hardware Combining Queue)



DEC Memory Channel

“Push-Only” Memory-based message passing

Map part of the Virtual Address Space to the NI

Open as “Outgoing” or “Incoming” channels

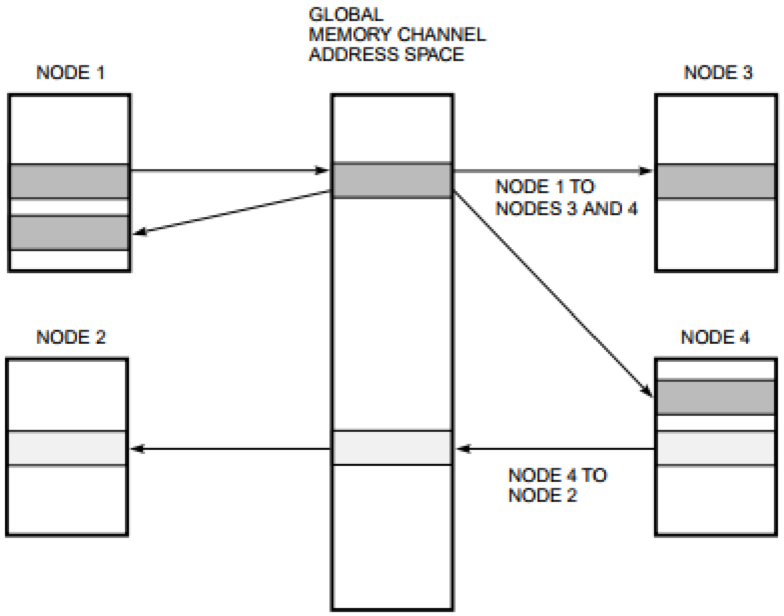
Incoming channels allocate a physical frame that is pinned

No frames allocated for outgoing channels.

One-to-Many channels possible

Send is simply a sequence of stores and Receives are a sequence of loads.

Completely avoids additional copies!



DEC Memory Channel

Send:

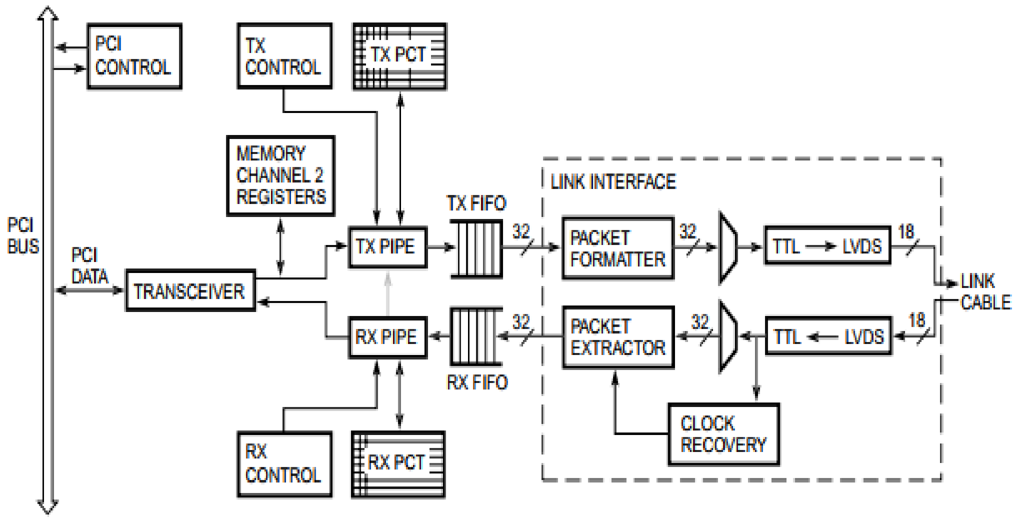
CPU Stores words into outgoing channel (page in VA)

Write-Buffer in CPU coalesces words (resulting in a Cache block write)

NIC uses Physical Frame # to Index PCT

Get Descriptor for Destination (route, control bits, Frame # at receiver)

Compose Packet and Send



Receive:

Extract Receive Frame #

Use that to index PCT

Verify validity

DMA to Physical Frame

Receive Frames are cacheable (cache coherent transfer)