

Portals 3.0: A High-performance Message Passing Layer Supporting Application-Bypass

Arthur B.

Computer Science Department
The University of New Mexico

Ma

Joint Work

Ron Brightwell

Tramm Hudson (now at TurboLinux)

Rolf Riesen

Goal

Make MPI go fast.

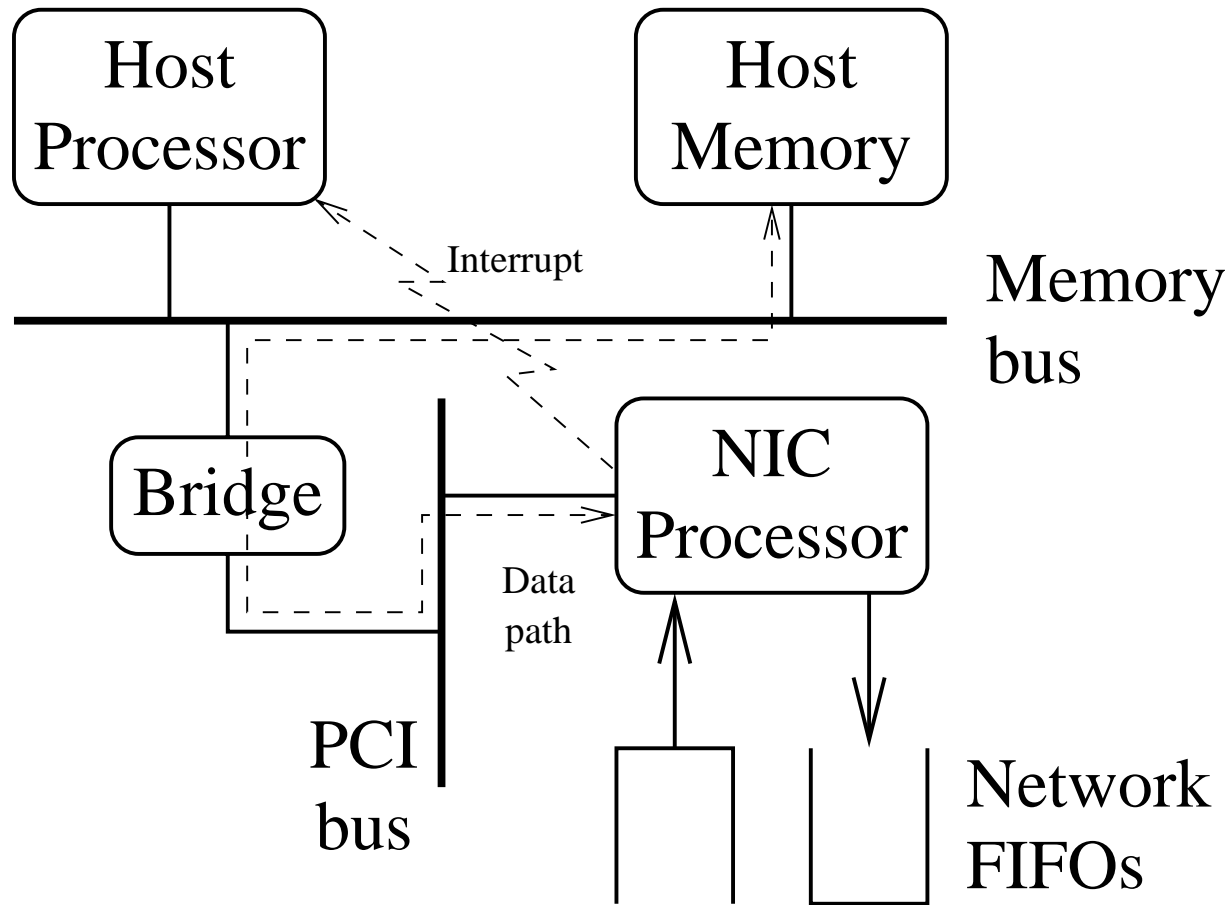
Oh yeah, also make it possible to run our systems software

Try to do something to make parallel I/O possible

TCP/IP? Why?

Portals is an API, not a protocol

Interrupt Path



Which Factors Impact Point-to-point MPI Performance?

- Packets?
 - probably not
- Memory copy?
 - maybe
- Protocol implementation?
 - yes
- PCI Bus
 - yes (wait for PCI-X or InfiniBand, your choice)
- OS Interrupt latency
 - definitely

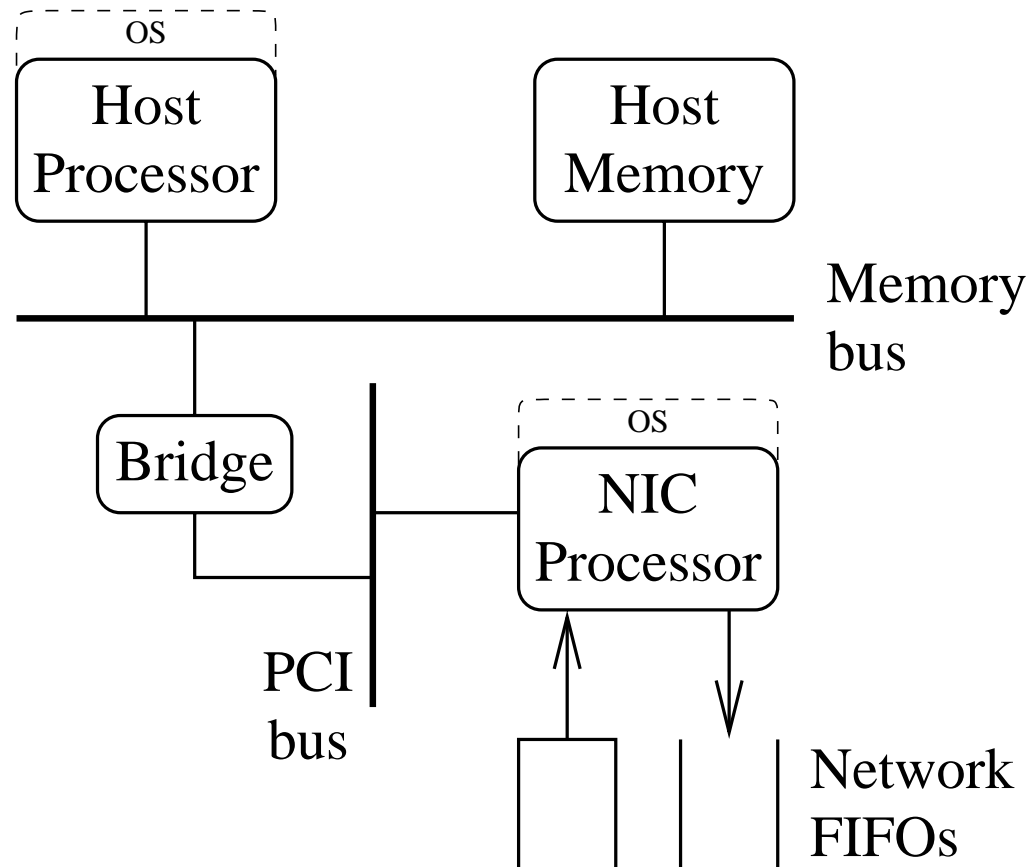
Barney's Favorite Calculation

- Assume 1500 byte packet and constant stream of packets
- Time between packets is
 - 1200 usec @ 10 Mb/s
 - 120 usec @ 100 Mb/s
 - 12 usec @ 1 Gb/s
 - 1.2 usec @ 10 Gb/s
- OS Interrupt latencies are 5-10 usecs!

How to Cope?

- Jumbo frames (packetization is the problem)
 - Doesn't address existing hardware
 - Good reasons for small packets
 - ◇ network fairness
 - ◇ reliability
 - Learn to live with small packets!
 - ◇ at least it's not ATM
- Coalescing interrupts
 - Amortize interrupt overhead over many packets
 - An easy fix for many applications
 - Still has overhead implications

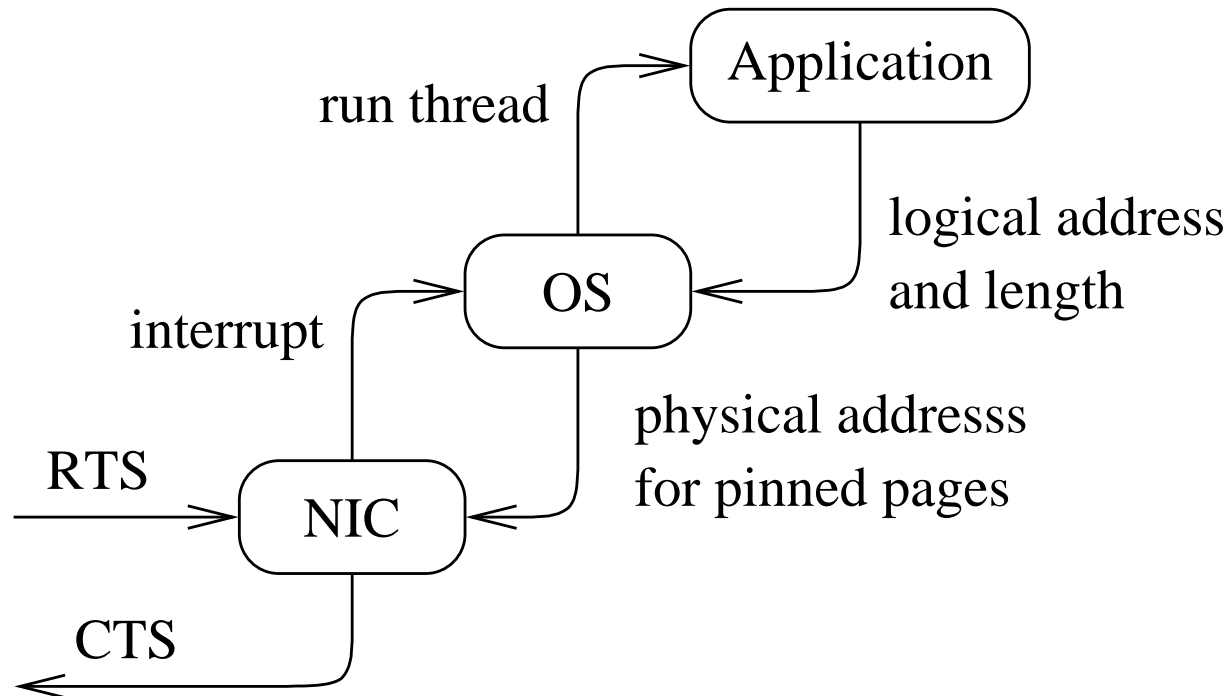
OS Bypass



- Really "host-processor" bypass
- OS policies on the NIC
 - address mapping and protection
- Messages go directly to application memory

OS Bypass Isn't Enough

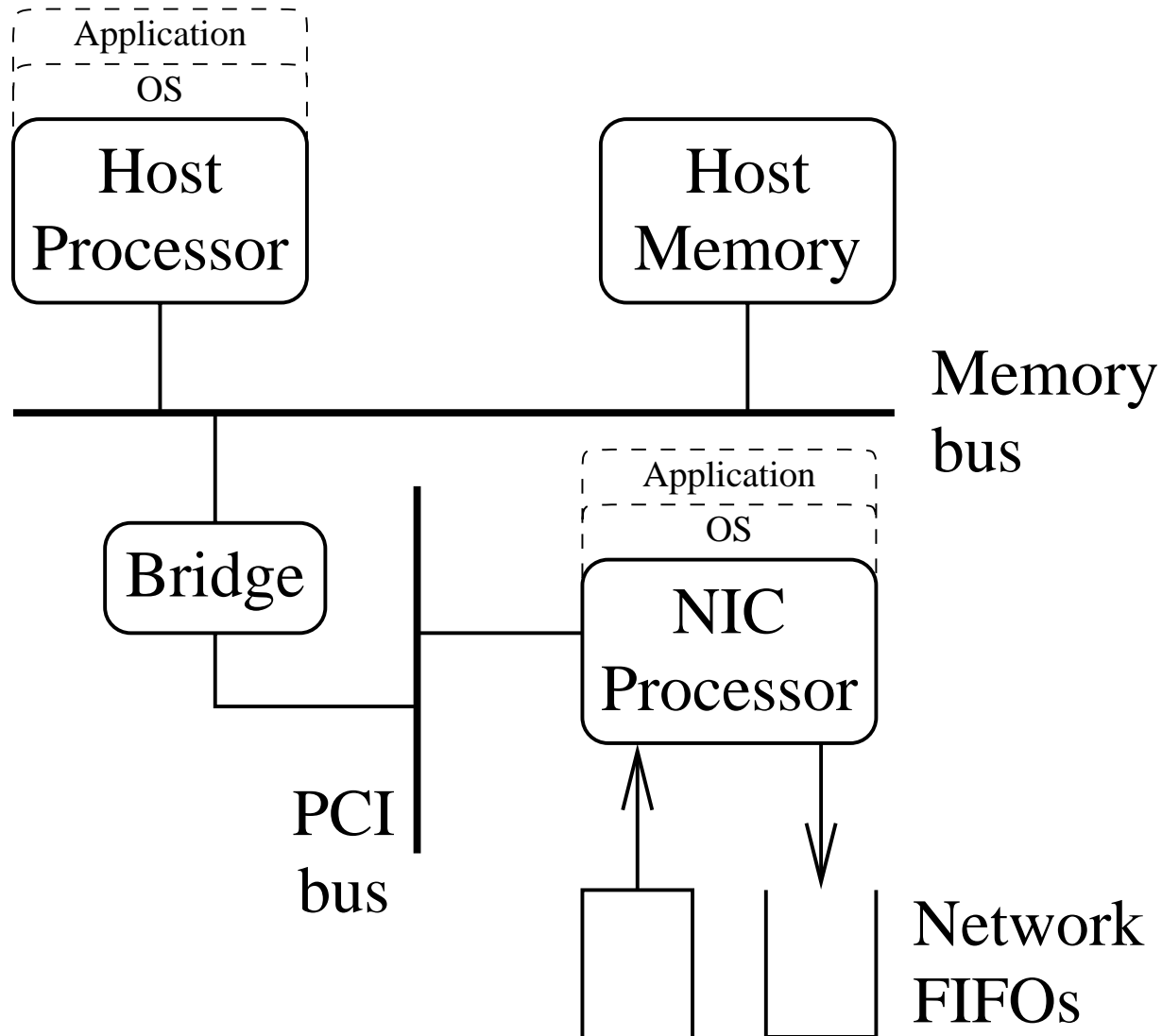
Matching RTS to posted receive requires application knowledge



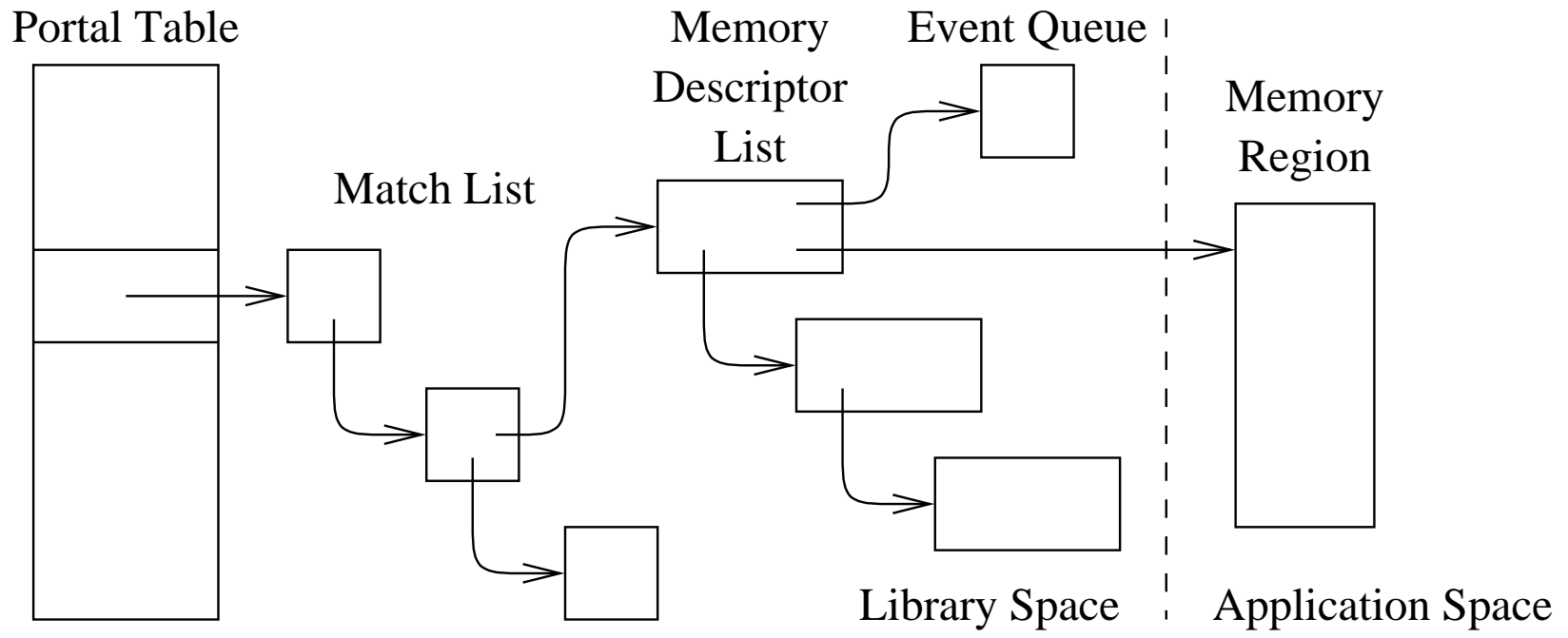
Issues

- Good things
 - Overhead associated with interrupt is amortized over length of message
 - ◇ not nearly as bad as an interrupt per packet
 - Including initial data packet in RTS good for small messages
- Problems
 - Increased complexity in runtime environment
 - ◇ needs a thread to generate response to RTS messages
 - Added latency cannot be hidden when receiver is waiting

Application Bypass



Matching in Portals 3.0



Portal Tables

- Multiple processes and/or network interfaces
 - Each process has its own portal table on every network interface it uses
- Multiple upper level protocols
 - A portal table is an array of match lists
 - Each upper level protocol uses a separate portal table index
- Provides access control

Match Lists

- A collection of match entries
 - Each match entry has a stack of memory descriptors
- Sequential search for match with non-rejecting memory descriptor
- Optional unlink when memory descriptor stack becomes empty
- Matching criteria includes:
 - Id of initiator (can use a wildcard)
 - Tag bits
 - ◇ 64-bit value supplied by the initiator
 - ◇ Match entry specifies don't care bits and must match bits

Memory Descriptors

Used for sending requests and for responding to incoming requests

- A logically contiguous region of the application's address space (no scatter/gather)
- When an application registers a memory descriptor, the OS can:
 - "pin" the pages associated with the memory descriptor
 - make the appropriate part of the page table available to the NIC
- Common to all OS-Bypass protocols

Event Queues

- Record information about "successful" operations
 - Incoming operations that match are successful, even if they truncate the data
 - Records pertinent information about the operation
- Event queues are associated with memory descriptors
 - Several memory descriptors may share an event queue
- Event queues have finite length
 - Applications can query for dropped events

Handling Incoming Requests

Applications must anticipate incoming requests

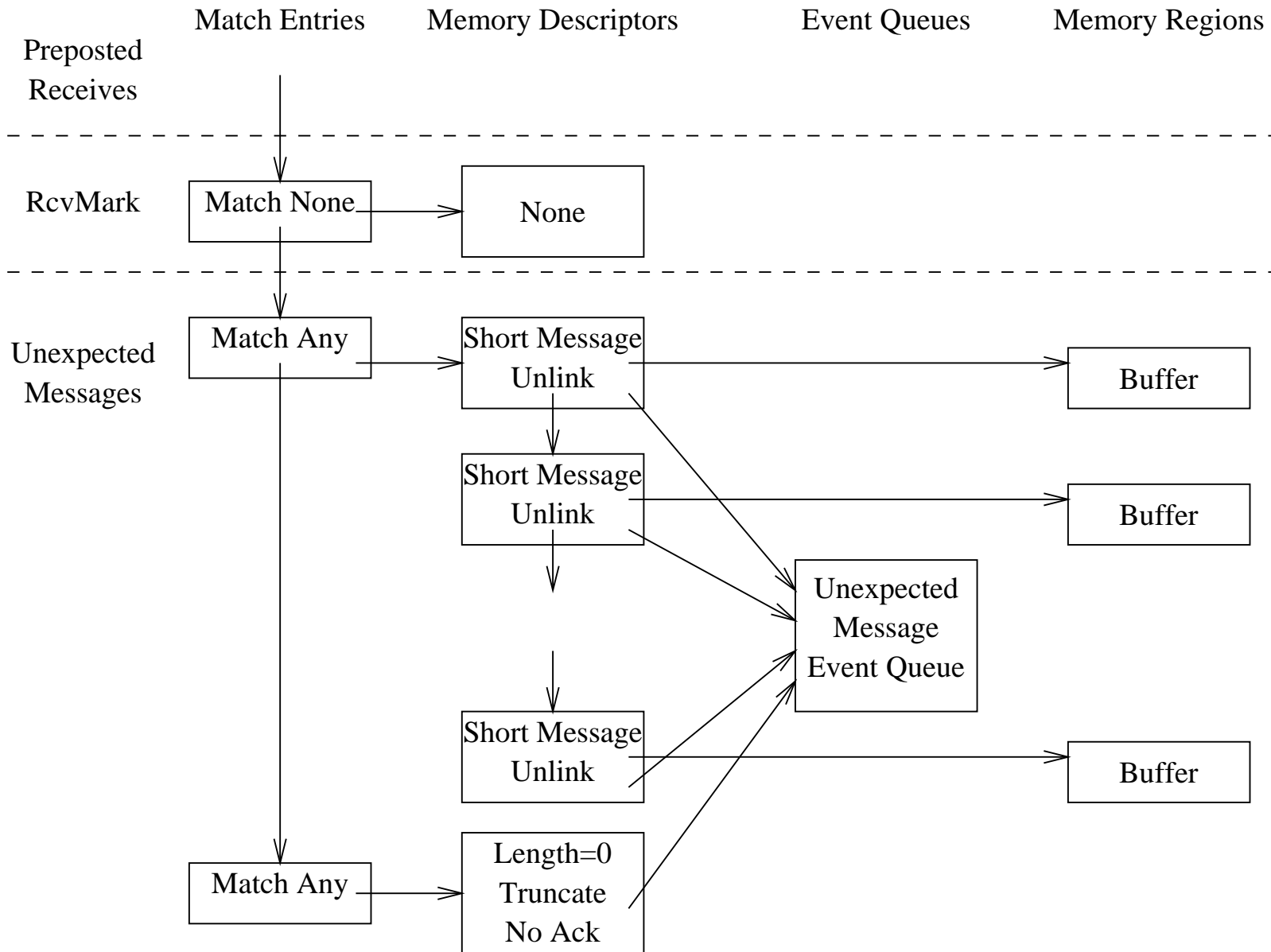
- Unhandled requests are (quietly) discarded
- Not as bad as it may seem: it's easy to "handle"
unexpected requests
 - discard any data associated with the request and save the event information

Rejecting Requests

Memory descriptors used to handle incoming requests may reject requests based on:

- Threshold
 - Number of operations allowed (must be >0)
- Size of the data in incoming request
 - Descriptor can allow truncation
- Operation
 - Put only, Get only, Put or Get

Implementing MPI



The MPI Implementation Protocol

- Short messages
 - just send the message
 - receiver has preposted received for short messages
- Long messages
 - post a "get" for the message body
 - send the message (in case the receiver has posted a matching receive)
 - unexpected long messages will be discarded, but the header will be kept
 - preposted get is cleared by an ACK or by a later get operation

Implementing the Implementation Protocol

- Issue: unexpected long messages waste network bandwidth
- Solution: define the lower layer correctly
- Implementing Portal Put:
 - send RTS
 - resulting CTS is based on disposition of the message body
 - lower level send does not try to send the whole message body, only the portion enabled by the CTS

Relation to One-sided Operations

- Portals 3.0 is based on put and get operations
- In Portals, remote address is a triple:
 - Portal index
 - ◇ Different upper level protocols use different portal indexes
 - Tag/matching bits
 - ◇ Needed to implement many two-sided operations
 - Offset
- Matching is based on
 - Time of posting (order)
 - Identity of sending process
 - Tag bits in the message

Further Work

- Implementation
 - NAL implementation (on IP!)
 - Myrinet using a linux kernel module
 - Tnet implementation underway
 - Alteon implementation underway
- Extending application-bypass
 - collective operations?
 - slow NIC processors
- Looking at commodity protocols
 - IP?