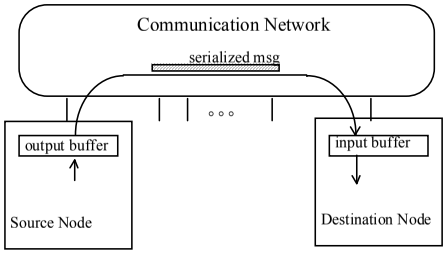
CS 258
Parallel Computer Architecture
Lecture 8

Network Interface Design

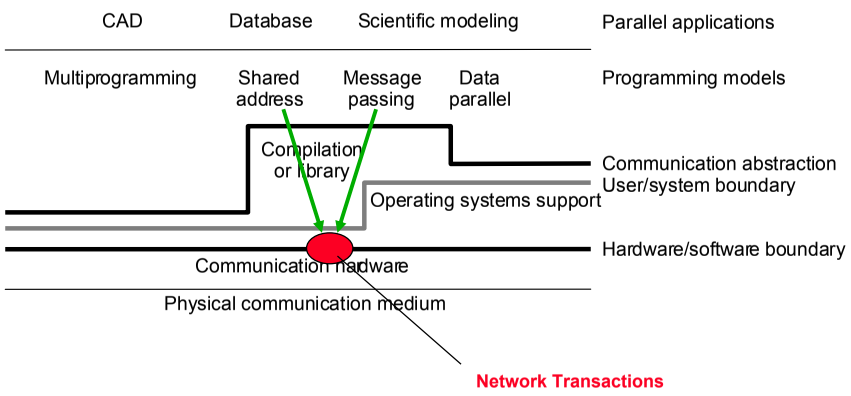
February 20, 2008
Prof John D. Kubiatowicz
http://www.cs.berkeley.edu/~kubitron/cs258

Network Transaction Primitive

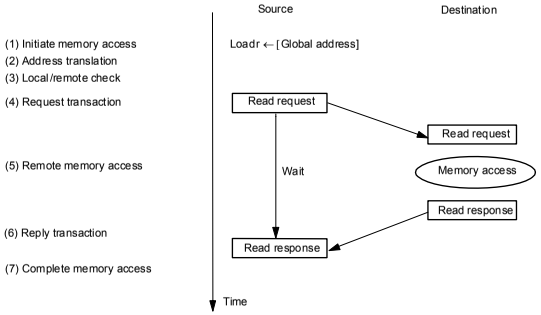


- one-way transfer of information from a source output buffer to a dest. input buffer
  - causes some action at the destination
  - occurrence is not directly visible at source
- deposit data, state change, reply

Programming Models Realized by Protocols

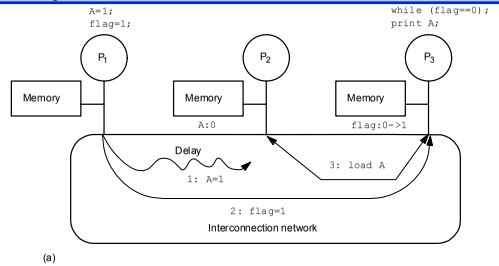


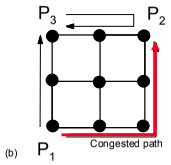
Shared Address Space Abstraction



- 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?

Consistency





- write-atomicity violated without caching
  - No way to enforce serialization
- Solution? Acknowledge write of A before writing Flag…

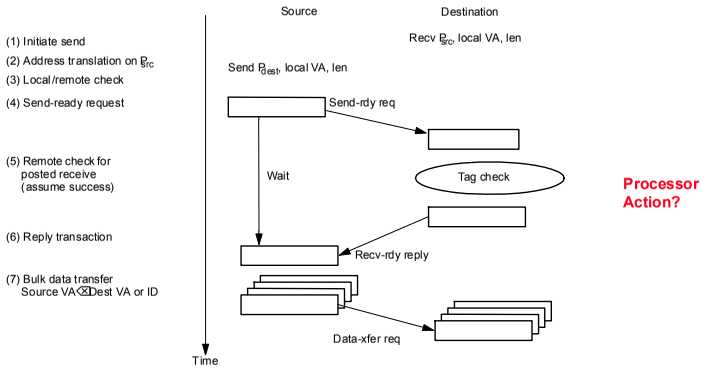
Properties of Shared Address Abstraction

- Source and destination data addresses are specified by the source of the request
  - a degree of logical coupling and trust
- no storage logically "outside the address space"
  - » may employ temporary buffers for transport
- Operations are fundamentally request response
- Remote operation can be performed on remote memory
  - logically does not require intervention of the remote processor

Message passing

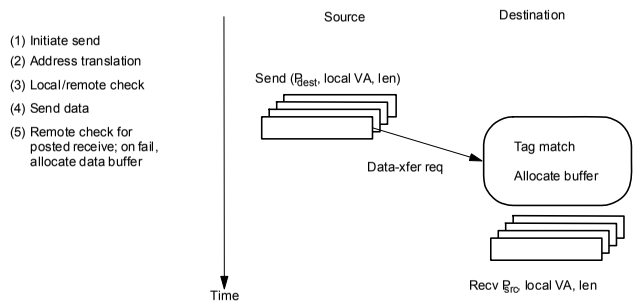
- Bulk transfers
- Complex synchronization semantics
  - more complex protocols
  - More complex action
- Synchronous
  - Send completes after matching recv and source data sent
  - Receive completes after data transfer complete from matching send
- Asynchronous
  - Send completes after send buffer may be reused

Synchronous Message Passing



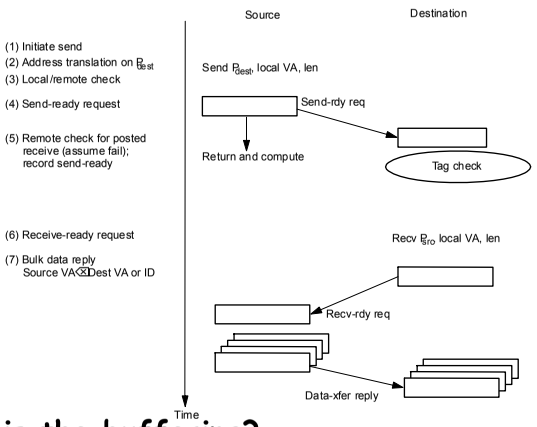
- Constrained programming model.
- Deterministic! What happens when threads added?
- Destination contention very limited.
- User/System boundary?

Asynch. Message Passing: Optimistic



- More powerful programming model
- Wildcard receive => non-deterministic
- Storage required within msg layer?

Asynch. Msg Passing: Conservative

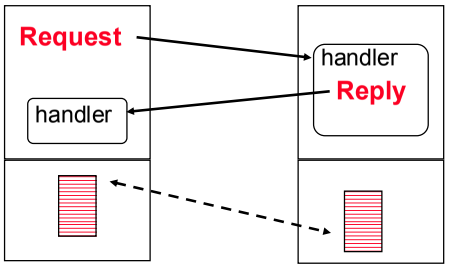


- Where is the buffering?
- Contention control? Receiver initiated protocol?
- Short message optimizations

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
- Optimistically, can be 1-phase transaction
  - Compare to 2-phase for shared address space
  - Need some sort of flow control
    - » Credit scheme?
- More conservative: 3-phase transaction
  - includes a request / response
- Essential point: combined synchronization and communication in a single package!

Active Messages



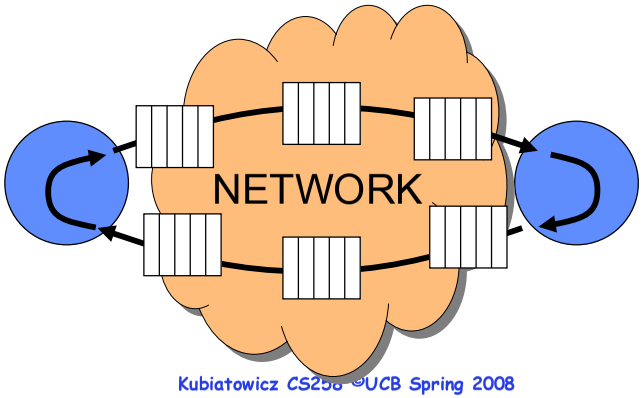
- User-level analog of network transaction
  - transfer data packet and invoke handler to extract it from the network and integrate with on-going computation
- Request/Reply
- Event notification: interrupts, polling, events?
- May also perform memory-to-memory transfer

Common Challenges

- Input buffer overflow
  - N-1 queue over-commitment => must slow sources
- Options:
  - reserve space per source (credit)
    - » when available for reuse?
      - Ack or Higher level
  - Refuse input when full
    - » backpressure in reliable network
    - » tree saturation
    - » deadlock free
    - » what happens to traffic not bound for congested dest?
  - Reserve ack back channel
  - drop packets
  - Utilize higher-level semantics of programming model

The Fetch Deadlock Problem

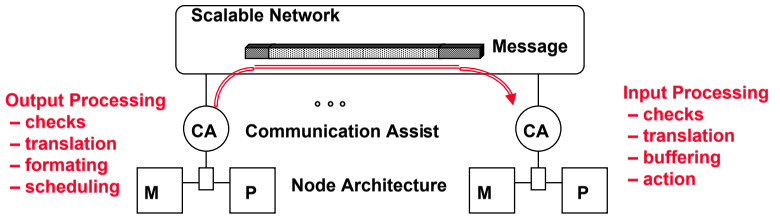
- Even if a node cannot issue a request, it must sink network transactions!
  - Incoming transaction may be request ⇒ generate a response.
  - Closed system (finite buffering)
- Deadlock occurs even if network deadlock free!



Solutions to Fetch Deadlock?

- logically independent request/reply networks
  - physical networks
  - virtual channels with separate input/output queues
- bound requests and reserve input buffer space
  - K(P-1) requests + K responses per node
  - service discipline to avoid fetch deadlock?
- NACK on input buffer full
  - NACK delivery?
- Alewife Solution:
  - Dynamically increase buffer space to memory when necessary
  - Argument: this is an uncommon case, so use software to fix

Network Transaction Processing



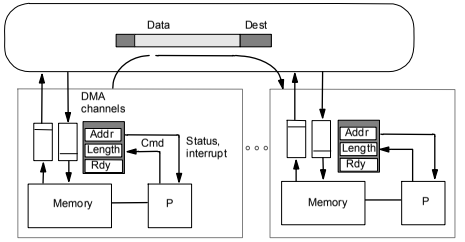
- Key Design Issue:
- How much interpretation of the message?
- How much dedicated processing in the Comm. Assist?

Spectrum of Designs

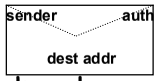
- None: Physical bit stream
  - blind, physical DMA — nCUBE, iPSC, . . .
- User/System
  - User-level port — CM-5, *T, Alewife
  - User-level handler — J-Machine, Monsoon, . . .
- Remote virtual address
  - Processing, translation — Paragon, Meiko CS-2
- Global physical address
  - Proc + Memory controller — RP3, BBN, T3D
- Cache-to-cache
  - Cache controller — Dash, Alewife, KSR, Flash

Increasing HW Support, Specialization, Intrusiveness, Performance (???)

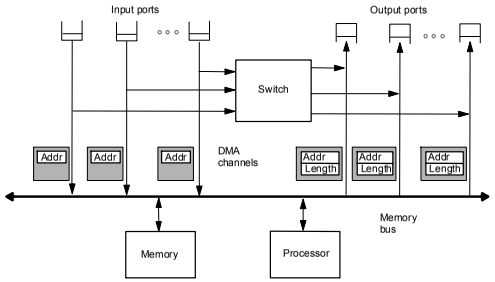
Net Transactions: Physical DMA



- 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 in CA



nCUBE Network Interface

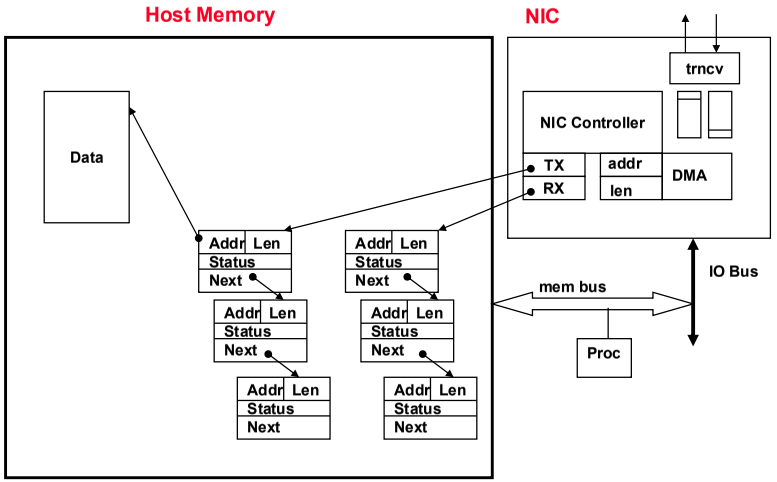


- independent DMA channel per link direction
  - leave input buffers always open
  - segmented messages
- routing interprets envelope
  - dimension-order routing on hypercube
  - bit-serial with 36 bit cut-through

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

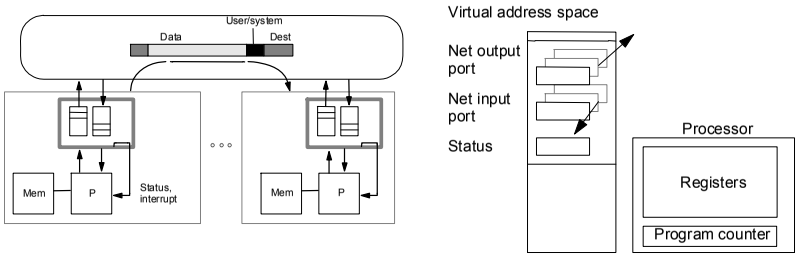
- includes interrupt

Conventional LAN NI



- Costs: Marshalling, OS calls, interrupts

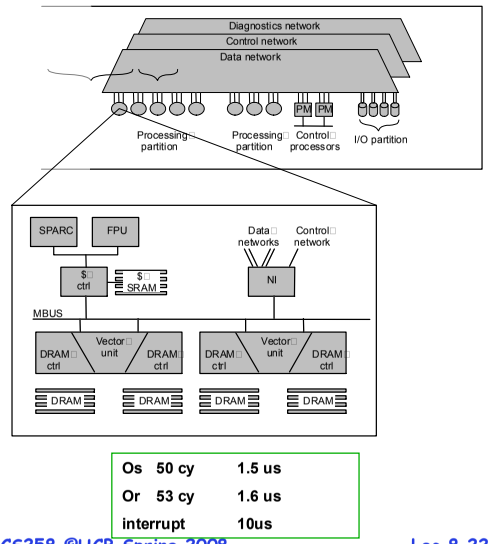
User Level Ports



- initiate transaction at user level
- deliver to user without OS intervention
- network port in user space
  - May use virtual memory to map physical I/O to user mode
- User/system flag in envelope
  - protection check, translation, routing, media access in src CA
  - user/sys check in dest CA, interrupt on system

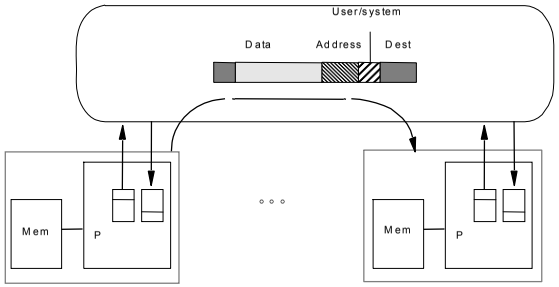
Example: CM-5

- Input and output FIFO for each network
- 2 data networks
- tag per message
  - index NI mapping table
- context switching?
- Alewife integrated NI on chip
- *T and iWARP also



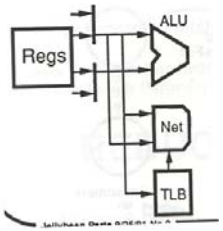
Os	50 cy	1.5 us
Or	53 cy	1.6 us
interrupt		10us

User Level Handlers



- Hardware support to vector to address specified in message
  - On arrival, hardware fetches handler address and starts execution
- Active Messages: two options
  - Computation in background threaads
    - » Handler never blocks: it integrates message into computation
  - Computation in handlers (Message Driven Processing)
    - » Handler does work, may need to send messages or block

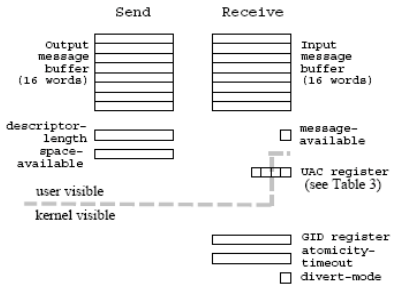
J-Machine

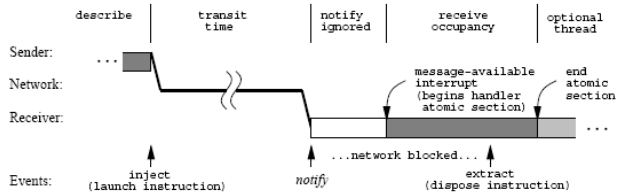


- Each node a small mdg driven processor
- HW support to queue msgs and dispatch to msg handler task

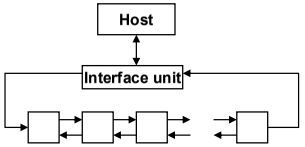
Alewife Messaging

- Send message
  - write words to special network interface registers
  - Execute atomic launch instruction
- Receive
  - Generate interrupt/launch user-level thread context
  - Examine message by reading from special network interface registers
  - Execute dispose message
  - Exit atomic section

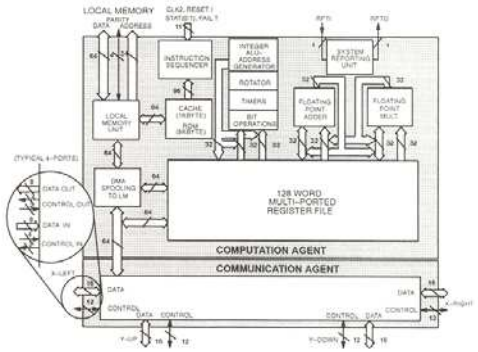




iWARP



- Nodes integrate communication with computation on systolic basis
- Msg data direct to register of neighbor
- Stream into memory

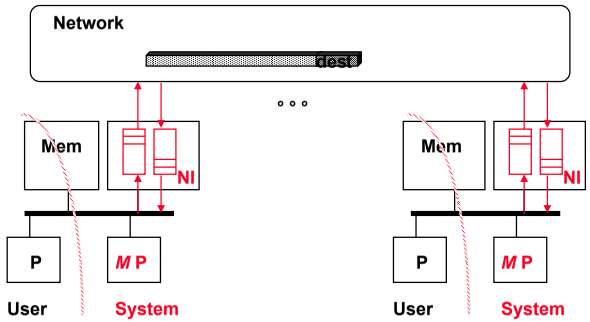


Sharing of Network Interface

- What if user in middle of constructing message and must context switch???
  - Need *Atomic Send operation!*
    - » Message either completely in network or not at all
    - » Can save/restore user's work if necessary (think about single set of network interface registers
  - J-Machine mistake: after start sending message must let sender finish
    - » Flits start entering network with first *SEND* instruction
    - » Only a SENDE instruction constructs tail of message
- Receive Atomicity
  - If want to allow user-level interrupts or polling, must give user control over network reception
    - » Closer user is to network, easier it is for him/her to screw it up: Refuse to empty network, etc
    - » However, must allow *atomicity:* way for good user to select when their message handlers get interrupted
  - Polling: ultimate receive atomicity – never interrupted
    - » Fine as long as user keeps absorbing messages

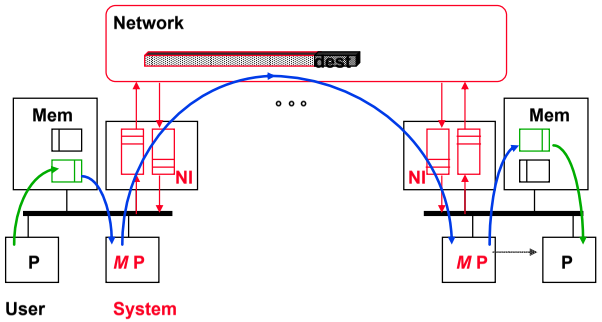
Dedicated processing without dedicated hardware design

Dedicated Message Processor



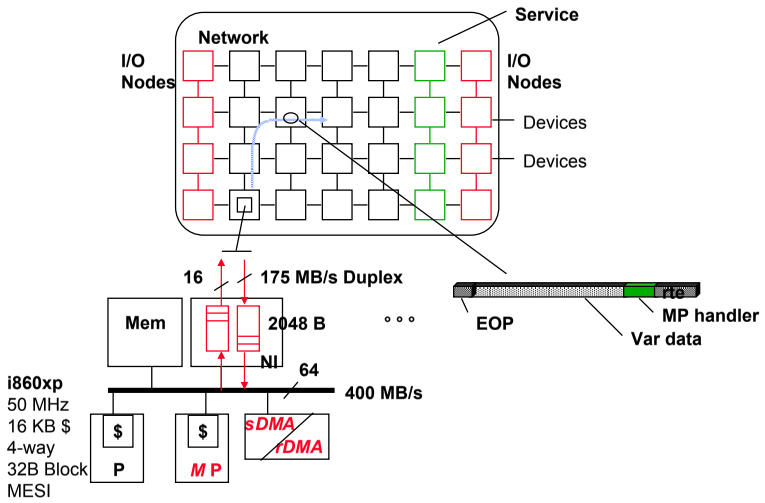
- General Purpose processor performs arbitrary output processing (at system level)
- General Purpose processor interprets incoming network transactions (at system level)
- User Processor <-> Msg Processor share memory
- Msg Processor <-> Msg Processor via system network transaction

Levels of Network Transaction

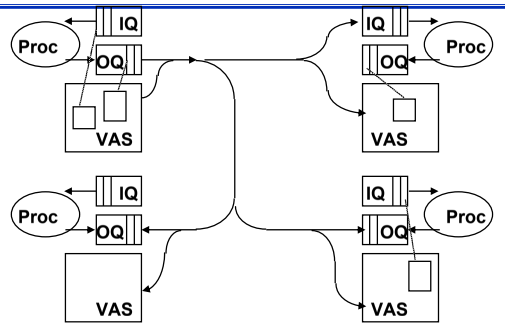


- User Processor stores cmd / msg / data into shared output queue
  - must still check for output queue full (or make elastic)
- Communication assists make transaction happen
  - checking, translation, scheduling, transport, interpretation
- Effect observed on destination address space and/or events
- Protocol divided between two layers

Example: Intel Paragon

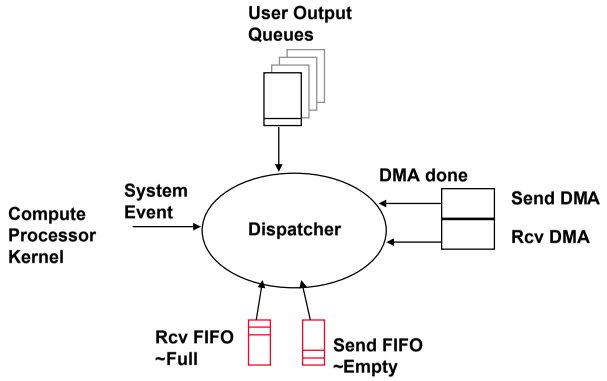


User Level Abstraction (Lok Liu)

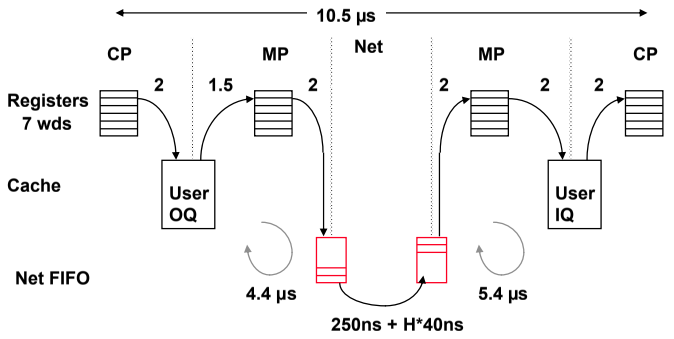


- Any user process can post a transaction for any other in protection domain
  - communication layer moves OQ_{src} -> IQ_{dest}
  - may involve indirection: VAS_{src} -> VAS_{dest}
- See, for instance:
  - "Remote Queues: Exposing Message Queues for Optimization and Atomicity," Eric A. Brewer, Fredrik T. Chong, Lok T. Liu, Shamik D. Sharma, John D. Kubiatowicz, SPAA 1995

Msg Processor Events

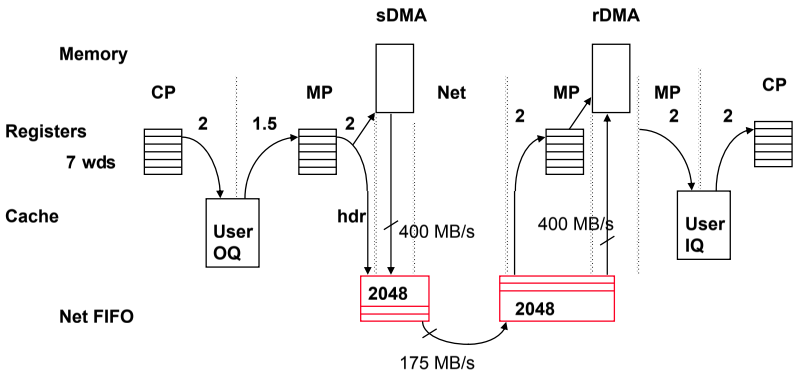


Basic Implementation Costs: Scalar



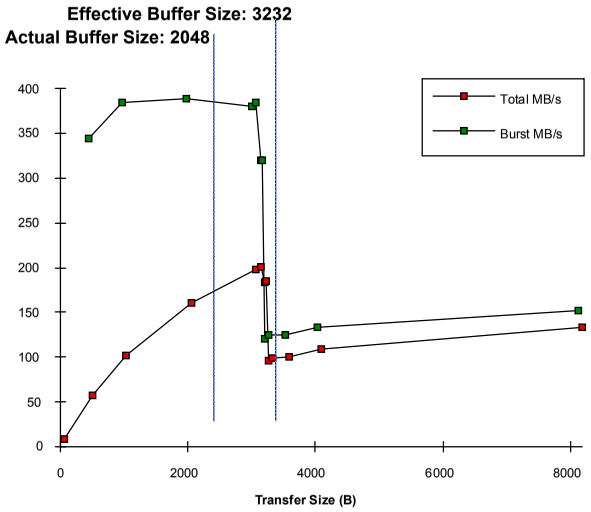
- Cache-to-cache transfer (two 32B lines, quad word ops)
  - **producer**: read(miss,S), chk, write(S,WT), write(I,WT),write(S,WT)
  - **consumer**: read(miss,S), chk, read(H), read(miss,S), read(H),write(S,WT)
- to NI FIFO: read status, chk, write, . . .
- from NI FIFO: read status, chk, dispatch, read, read, . . .

Virtual DMA -> Virtual DMA

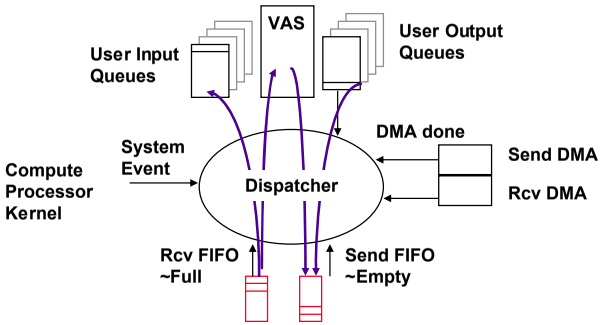


- Send MP segments into 8K pages and does VA -> PA
- Recv MP reassembles, does dispatch and VA -> PA per page

Single Page Transfer Rate



Msg Processor Assessment



- Concurrency Intensive
  - Need to keep inbound flows moving while outbound flows stalled
  - Large transfers segmented
- Reduces overhead but adds latency

Conclusion

- Shared Address Space
  - Request/Response Protocol
  - Global names for memory locations specify nodes
- Many different Message-Passing styles
  - Global Address space: 2-way
  - Optimistic message passing: 1-way
  - Conservative transfer: 3-way
- "Fetch Deadlock"
  - Request⇒Response introduces cycle through network
  - Fix with:
    - » 2 networks
    - » dynamic increase in buffer space
- Network Interfaces
  - User-level access
  - DMA
  - Atomicity