

Cap7

Scalable Distributed Memory Multiprocessors

Outline

- Scalability (centenas → milhares de processadores)
 - physical, bandwidth, latency and cost
 - level of integration
- Realizing Programming Models
 - network transactions
 - protocols
 - safety
 - input buffer problem: N-1
 - fetch deadlock
- Communication Architecture Design Space
 - how much hardware interpretation of the network transaction?

Limited Scaling of a Bus

<u>Characteristic</u>	<u>Bus</u>
Physical Length	~ 1 ft
Number of Connections	fixed
Maximum Bandwidth	fixed
Interface to Comm. medium	extension to memory interface
Global Order	bus order (arbitration)
Protection	Virt -> physical addr translation
Trust	total
OS	single
comm. abstraction	HW
if any processor fails	reboot

- Bus: each level of the system design is grounded in the scaling limits at the layers below and assumptions of close coupling between components
- adicional: N. limitado de operações pendentes

Workstations in a LAN?

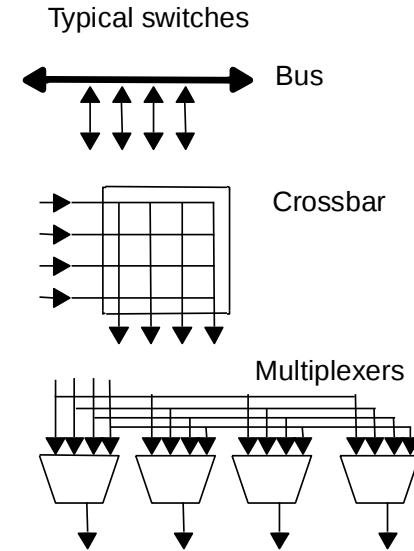
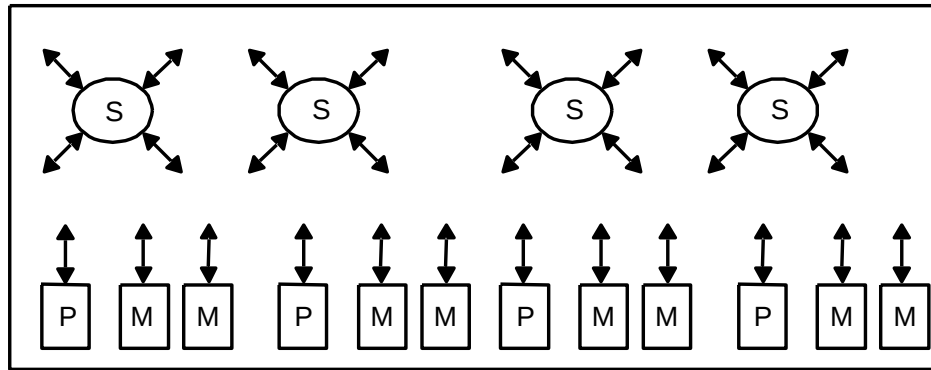
<u>Characteristic</u>	<u>Bus</u>	<u>LAN</u>
Physical Length	~ 1 ft	KM
Number of Connections	fixed	many
Maximum Bandwidth	fixed	???
Interface to Comm. medium	memory inf	peripheral
Global Order	arbitration	???
Protection	Virt -> physical	OS
Trust	total	none
OS	single	independent
comm. abstraction	HW	SW
reliability	1 proc fails → reboot	WS restart independently

- No clear limit to physical scaling, little trust, no global order, consensus difficult to achieve.
- Adicional: alta latência, BW baixa, N. desconhecido de outstanding operations, desconfiança → dificuldade de trabalhar coordenadamente em um problema

Scalable Machines

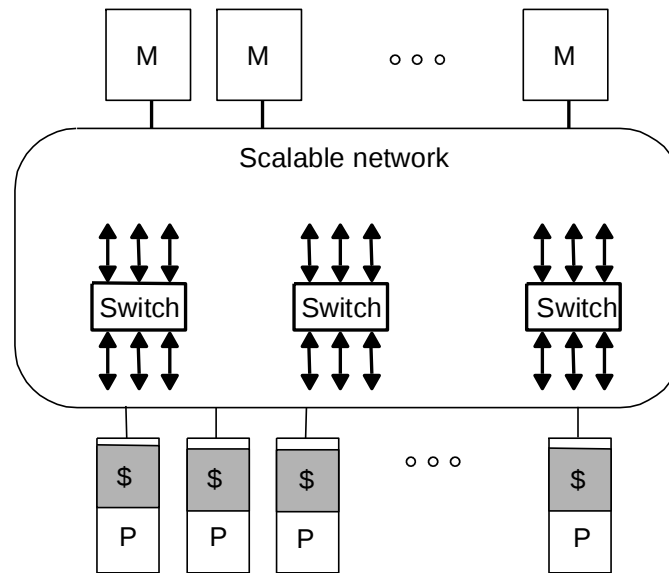
- What are the design trade-offs for the spectrum of machines between?
 - specialize or commodity nodes?
 - capability of node-to-network interface
 - supporting programming models?
- 7.1 What does scalability mean?
 - avoids inherent design limits on resources
 - bandwidth increases with P (ideal = linear)
 - latency does not (ideal = constante)
 - cost increases slowly with P
- Sistemas baseados em barramento único (cap 6) → do not scale well (segundo quatro aspectos: BW/throughput, latência, custo, packaging)

7.1.1 Bandwidth Scalability



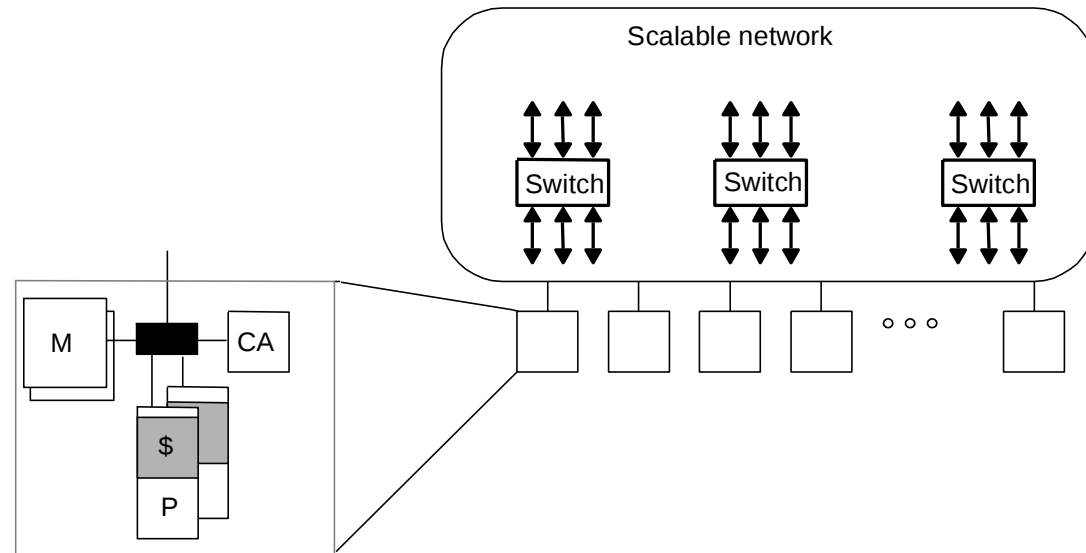
- What fundamentally limits bandwidth?
 - single set of wires
- Must have many independent wires
- Connect modules through switches (degree = N. de entradas ou saídas)
- Bus vs Network Switch?
- Bus switch: endereço é broadcasted para todos os conectados
- Network switch: informação na entrada é suficiente para o switch saber a quais outras switches ela deve se conectar (sem consultar todos os nós)

Dancehall MP Organization



- Network bandwidth?
- Bandwidth demand?
 - independent processes? (mesmo assim cresce linearmente com p)
 - communicating processes?
- Latency?

Generic Distributed Memory Org.



- Network bandwidth?
- Bandwidth demand?
 - independent processes?
 - communicating processes?
- Latency?

Key Property

- Abandonar hipóteses usadas em barramento único:
- N. limitado de transações simultâneas, globalmente ordenadas e arbitradas
- Large number of independent communication paths between nodes
- => allow a large number of concurrent transactions using different wires
- initiated independently
- no global arbitration
- sem ordem global
- effect of a transaction only visible to the nodes involved
 - effects propagated through additional transactions

7.1.2 Latency Scaling

$T(n) = \text{Overhead} + \text{Channel Time} + \text{Routing Delay}$

- overhead: tempo para iniciar ou completar a transação
- channel time: n/B do segmento crítico no caminho (gargalo)
- routing delay: $f(H,n) \rightarrow$ função do N. de hops (e possivelmente de n)

Overhead?

- normalmente fixo

Channel Time(n) = n/B --- BW at bottleneck

RoutingDelay(h,n) = (delay fixo por hop; mas no caso de store&forward depende de n)

- Store & Forward é impraticável em sistemas de grande porte

Switches com degree fixo $\rightarrow$ routing distance entre nós não escala muito bem (pode crescer demais)

Typical example (expl 7.1)

- Fixed degree $\rightarrow$ max distance: $\log n$; number of switches: $a n \log n$
- se overhead = 1 μ s/msg, BW = 64 MB/s, 200 ns per hop
- Calcular: aumento do tempo para transferir 128Bytes, se $n=64 \rightarrow n=1024$

escalamento = 16 x

64 nós $\rightarrow \log n = 6$;

1024 $\rightarrow \log 1024 = 10$

128 Bytes / (64 MB/s) = 2.0 μ s

Pipelined

$$T_{64}(128) = 1.0 \mu\text{s} + 2.0 \mu\text{s} + 6 \text{ hops} * 0.2 \mu\text{s/hop} = 4.2 \mu\text{s}$$

$$T_{1024}(128) = 1.0 \mu\text{s} + 2.0 \mu\text{s} + 10 \text{ hops} * 0.2 \mu\text{s/hop} = 5.0 \mu\text{s}; \rightarrow \text{aumento } 20\%$$

Store and Forward

$$T_{64}^{\text{sf}}(128) = 1.0 \mu\text{s} + 6 \text{ hops} * (2.0 + 0.2) \mu\text{s/hop} = 14.2 \mu\text{s}$$

$$T_{1024}^{\text{sf}}(128) = 1.0 \mu\text{s} + 10 \text{ hops} * (2.0 + 0.2) \mu\text{s/hop} = 23 \mu\text{s} \rightarrow \text{aumento } 62\% !!$$

7.1.3 Cost Scaling

$\text{cost}(p,m) = \text{fixed cost} + \text{incremental cost}(p,m)$ /*p=proc; m=mem

Bus Based SMP? (config. popular vendas; $p = P_{\max} / 2$!!)

- fixed = gabinete+fonte alim + bus;
- incremental = menor que 1 proc (stand-alone)

Custo relativo de: {processors , memory , network , I/O} ?

(ver exemplo 7.2, pag 462)

$\text{Parallel efficiency}(p) = \text{Speedup}(P) / P$

$\text{Costup}(p) = \text{Cost}(p) / \text{Cost}(1)$

Cost-effective:

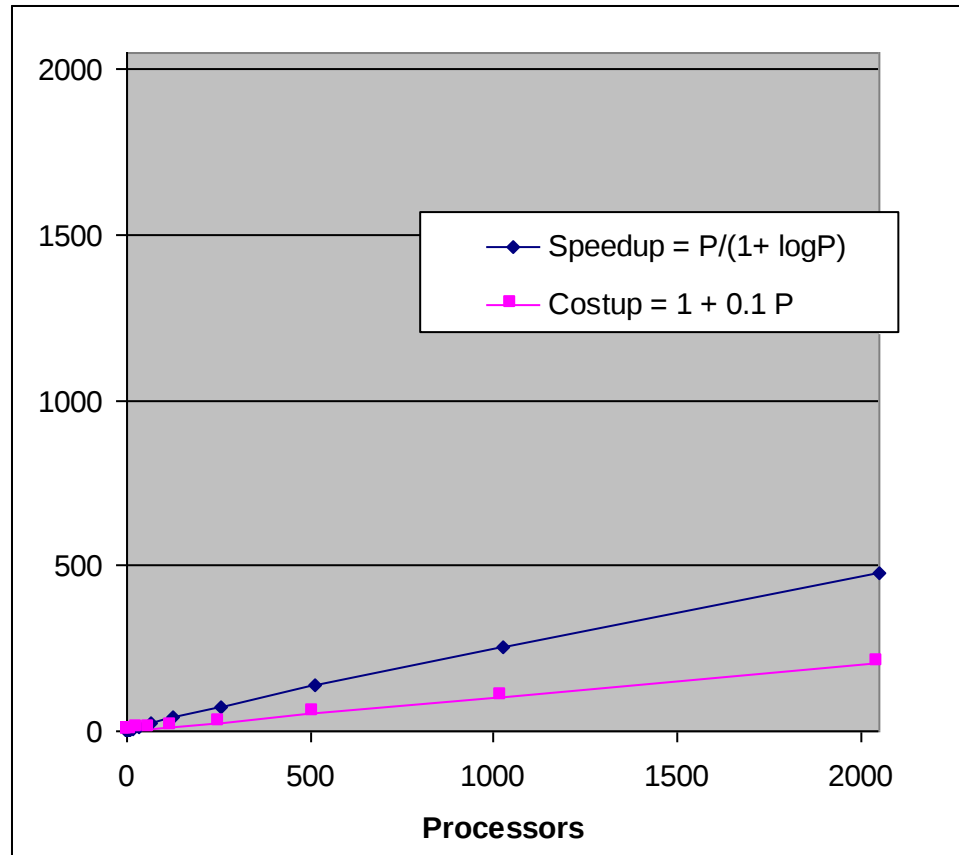
se

$\text{speedup}(p) > \text{costup}(p)$

então

o speedup é super-linear

Cost Effective?

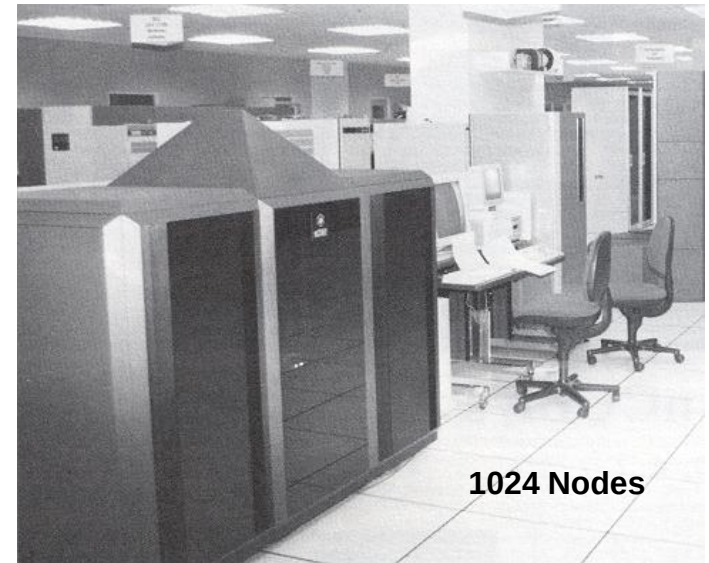
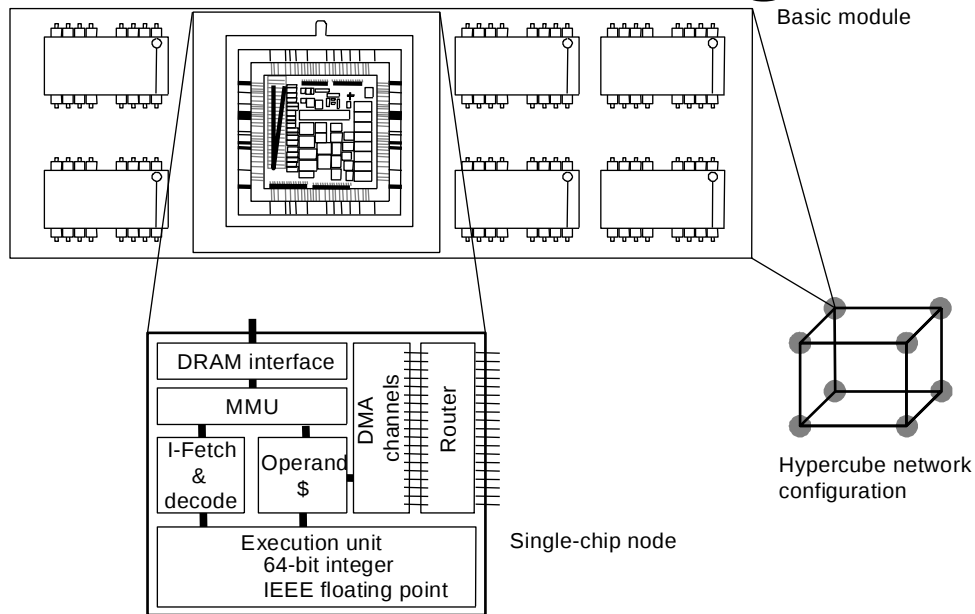


- 2048 processors: 475 fold speedup at 206x cost

7.1.4 Physical Scaling

- Chip-level integration
 - Board-level
 - System level
-
- fatores: distância, atraso, custo, escalabilidade, uso de partes comerciais (quanto menos integrado mais fácil), time-to-mkt, escalabilidade da tecnologia (acompanhar evolução é mais fácil se usar partes comerciais)
 - gama de soluções de projeto é muito grande.

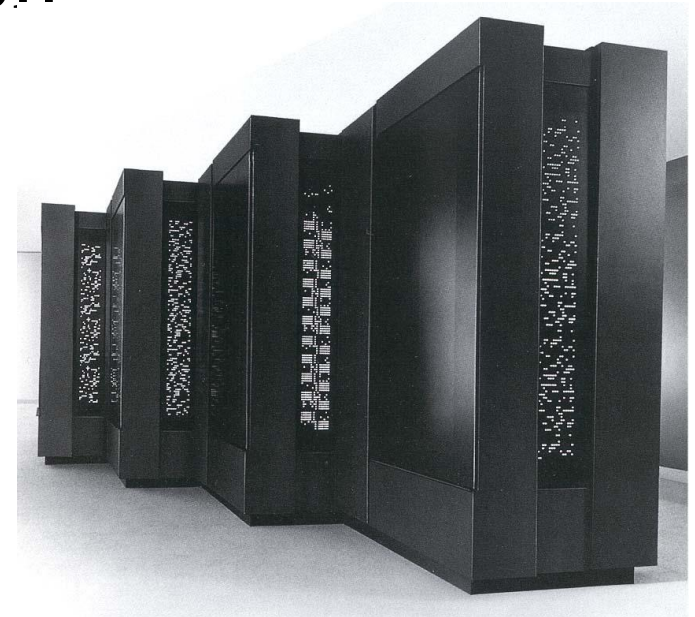
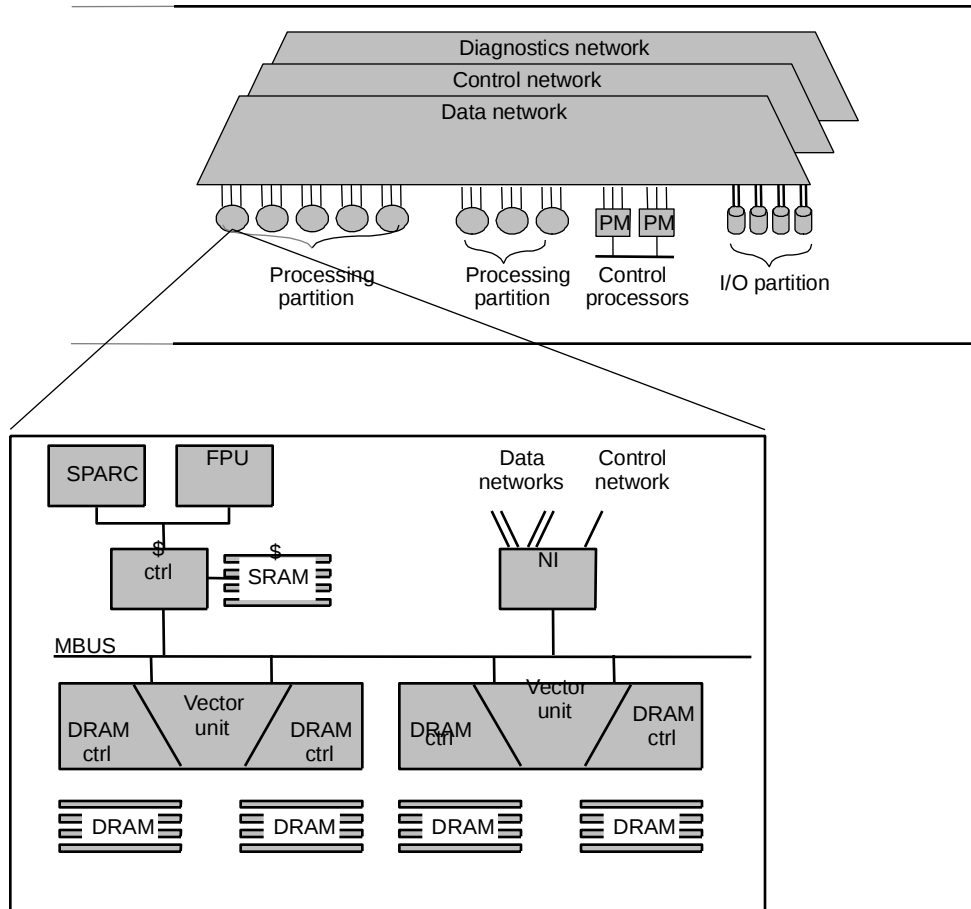
Chip-level integration: nCUBE/2 Machine Organization



1024 Nodes

- Cada nó = 1 chip: processador, controlador de memória, interface de rede, roteador
- Processador: VAX simplificado, 20 MHz, ALU inteira e fp de 64 bits, 7.5 MIPS e 2.4 MFLOPS
- Latência: roteamento (44 ciclos = 2.2 μ s), transfer (36 ciclos / word)
- Entire machine synchronous at 40 MHz
- Config max: 8k nodes ($\log 8096 = 13$; 13 vizinhos por nó); bit serial

Board-level integration: CM-5 Machine Organization



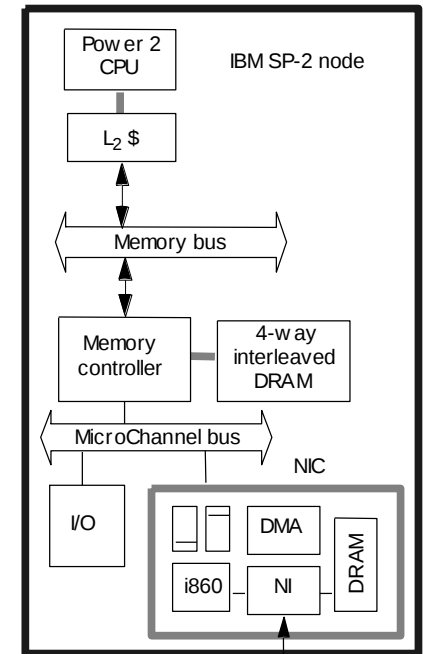
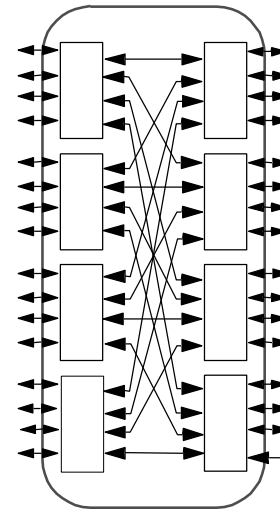
- Nó: SPARC chispset, 33MHz, FPU externa, Network interface (outro ASIC)
- Rede: árvore grau 4 c/ processadores nas folhas (fat-tree)
- Placa: 4 processadores

IBM scalable SP-2: System Level Integration

- Nó = quase sistema completo (RS6000)
- Abordagem criou tecnologias que migraram para LANs:
 - ATM, FDDI, FiberChannel, HPPI, Ethernet 100 Mbs,
 - System Area Network (SAN)
 - Semelhante a clusters



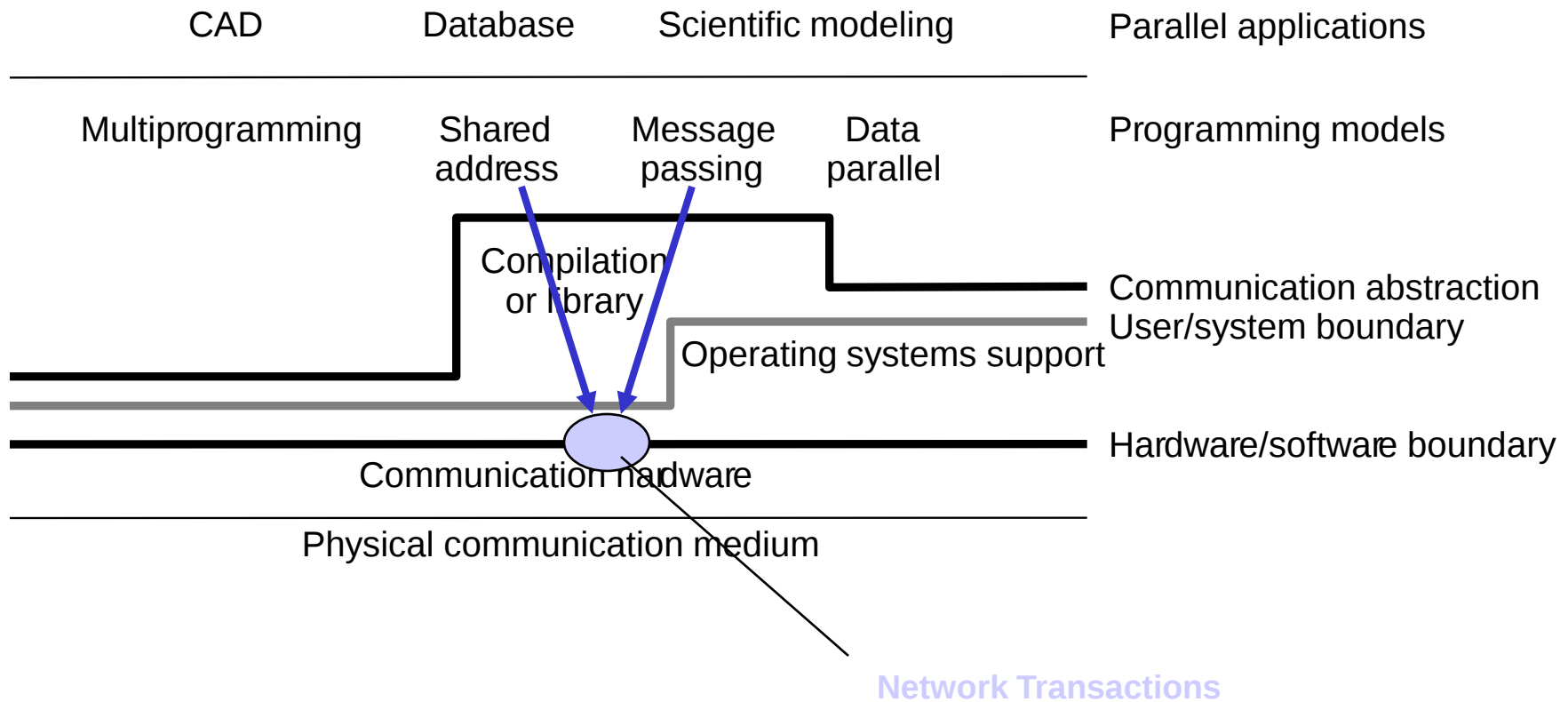
General inter connection network formed from 8-port switches



Outline

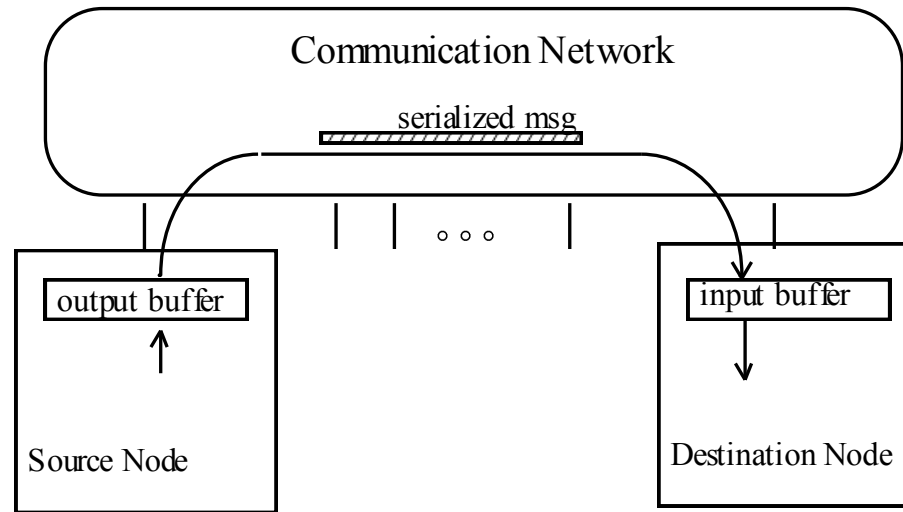
- 7.1 Scalability
 - physical, bandwidth, latency and cost
 - level of integration
 - (ler 7.1.5)
- 7.2 Realizing Programming Models
 - network transactions
 - protocols
 - safety
 - input buffer problem: $N-1$
 - fetch deadlock
- 7.2.6 Communication Architecture Design Space
 - how much hardware interpretation of the network transaction?

Programming Models Realized by Protocols



- em small scale SAS: communication abstraction → bus + HW
- em large scale: network xaction (not bus xaction)

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 (precisa xaction adicional)
- Tipos: deposit data, FSM state change, reply

7.2.1 Bus Transactions vs Net Transactions

Issues:

- protection check
- format
- output buffering
- media arbitration
- destination naming and routing
- input buffering
- action
- completion detection
- Ligação direta
- Transaction ordering
- Deadlock avoidance
- Delivery guarantees

Bus

V->P

wires (par.)

reg, FIFO

global

implícita

limited

fio

bus

Net

?? (low trust)

flexible (serial)

??

local

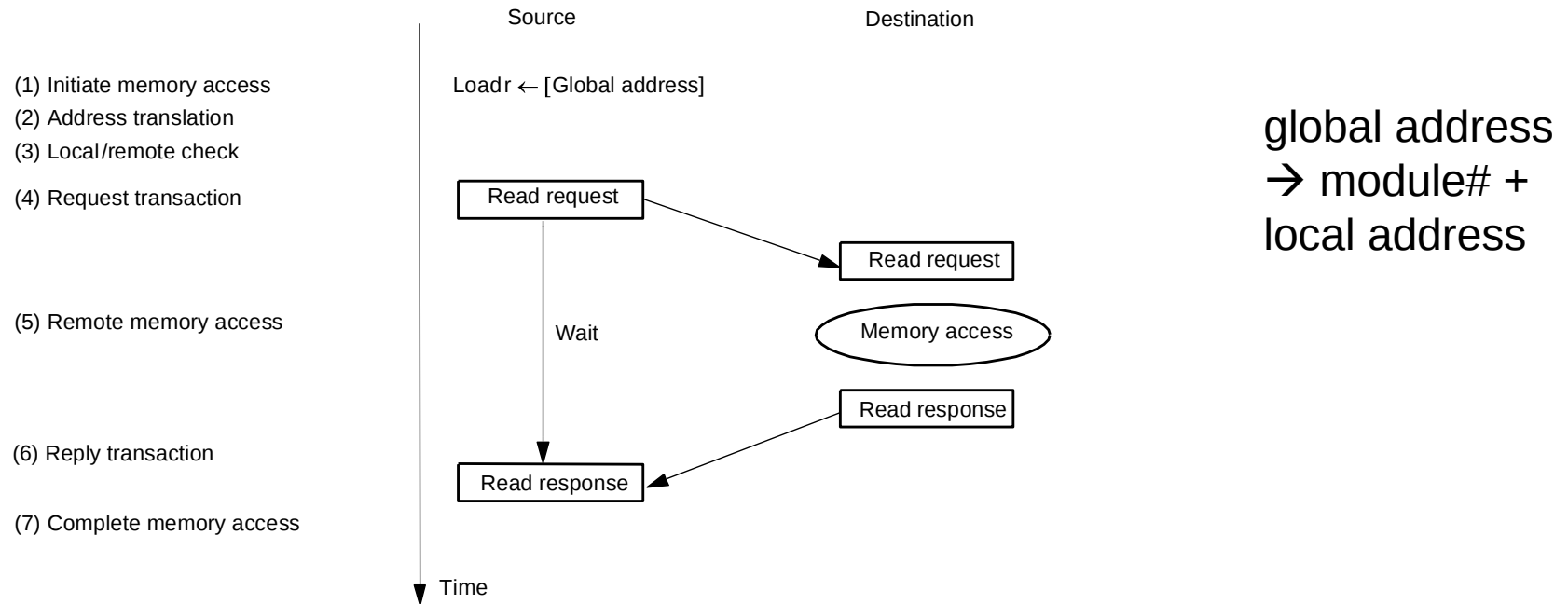
many source

s/ ligação direta estável

weak

dest. Buffer full??

7.2.2 Shared Address Space Abstraction

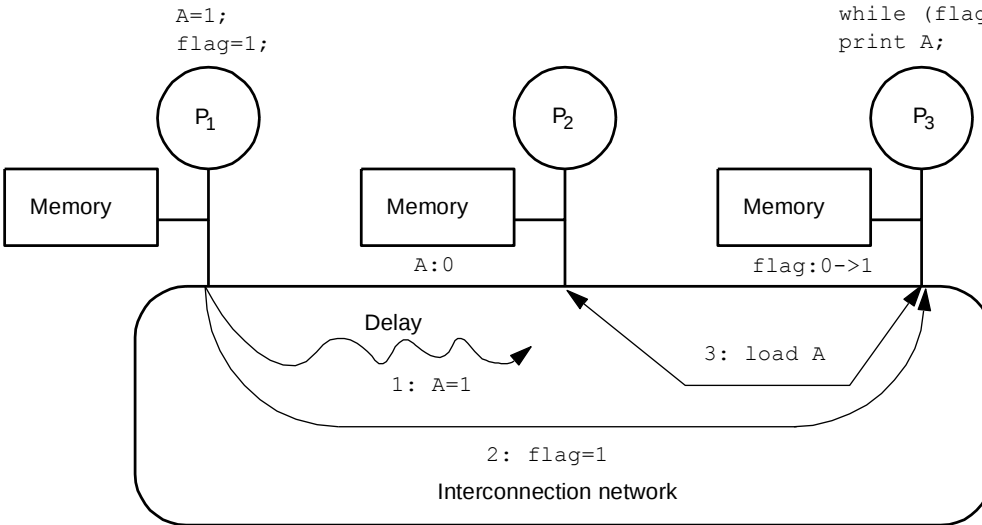


- Fundamentally a two-way request/response protocol (writes have response=ack)
- Issues
 - fixed or variable length (bulk) transfers
 - remote virtual or physical address, where is action performed?
 - deadlock avoidance and input buffer full (vistos mais adiante)
- coherent? (neste capítulo sem replicação em caches → sem problemas; replicação → cap 8 e 9)
- consistent? transparência adiante

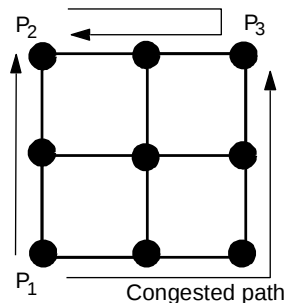
The Fetch Deadlock Problem

- Muitos nós podem emitir requests simultâneos (pode ser necessária capacidade de armazenamento $>$ buffers)
- Even if a node cannot issue a request (devido a buffer full), it must sink network transactions (para evitar o fetch deadlock)
- Incoming transaction may be a request, which will generate a response.
- Closed system (finite buffering)

Consistency



(a)



(b)

- processador emite um write e segue adiante (outro WR)
- variável A alocada na memória de P₂
- variável flag alocada na memória de P₃
- se há congestionamento, P₃ pode ler flag=1 de P₃ e ler A = 0 em P₂

- problemas se mensagens são roteadas primeiro na direção X e depois Y

- write-atomicity violated (even) without caching
- para corrigir: aguardar o ACK da primeira escrita antes de emitir a próxima

Key Properties of SAS

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 application address space(s)”
 - 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

7.2.3 Message passing

- Bulk transfers (mensagens longas de tamanho variável)
- 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

MPI (= Message Passing Interface)

Synchronous Message Passing

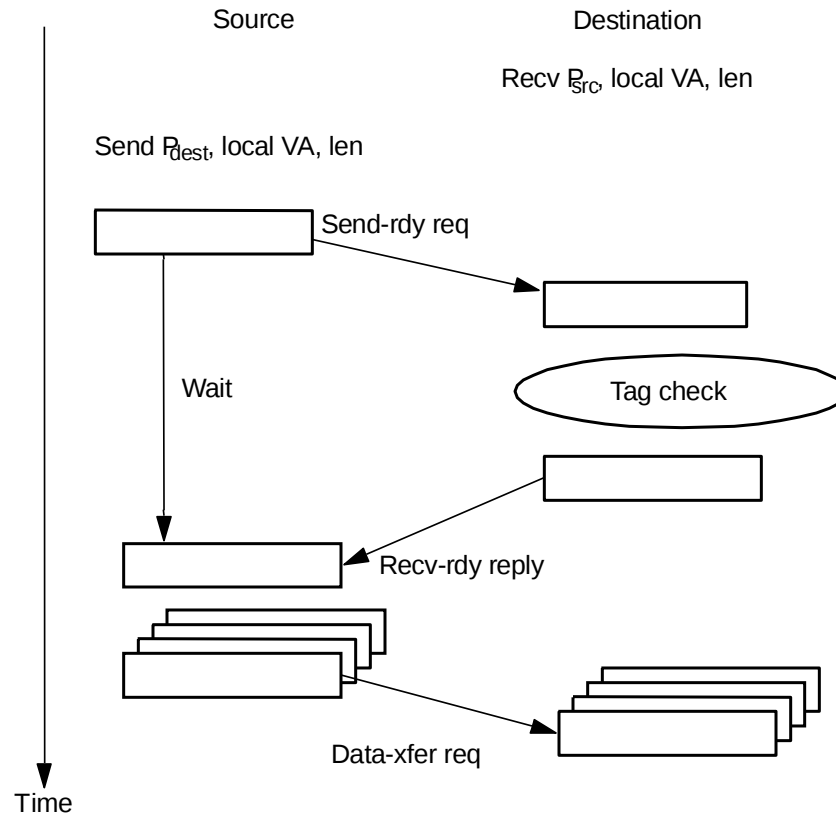
3 fases; send initiated

- (1) Initiate send
- (2) Address translation on P_{rc}
- (3) Local/remote check
- (4) Send-ready request

- (5) Remote check for posted receive (assume success)

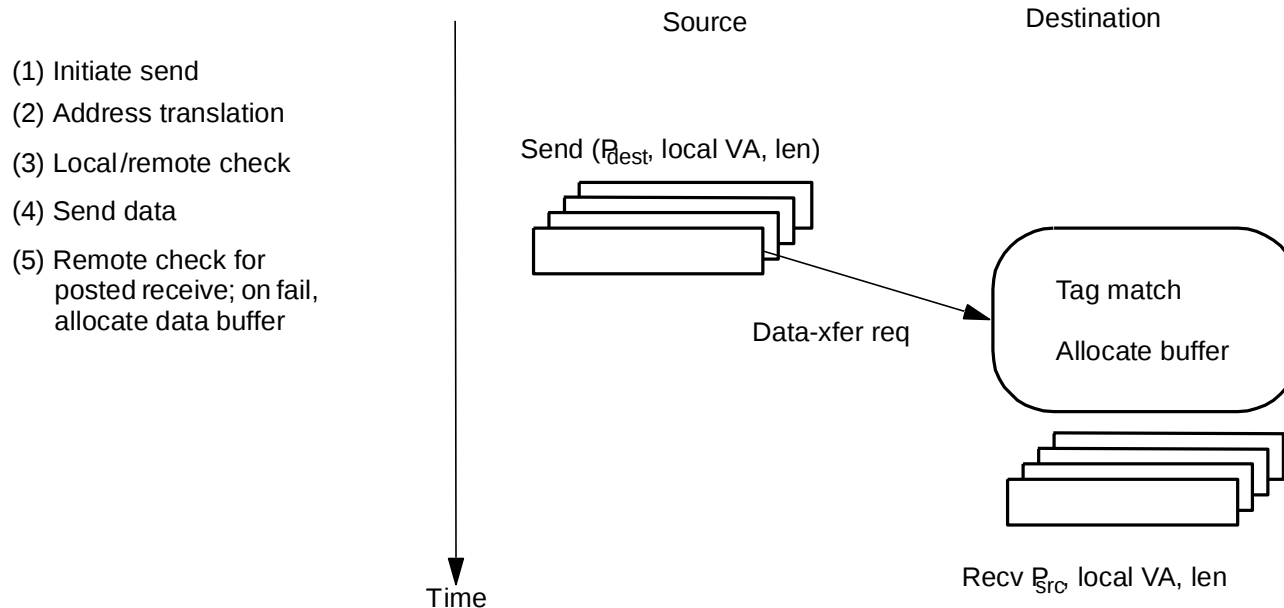
- (6) Reply transaction

- (7) Bulk data transfer
Source VA $\rightarrow$ Dest VA or ID



- Sender $\rightarrow$ rdy-to-send (contém src process + tag); sender espera rdy-to-rcv; receiver verifica se existe em tabela local um matching receive; se não, grava na tabela e aguarda o matching rcv; se sim, receiver envia rdy-to-rcv; sender envia dados; assumir operação completada (assume-se rede confiável) ao final do envio

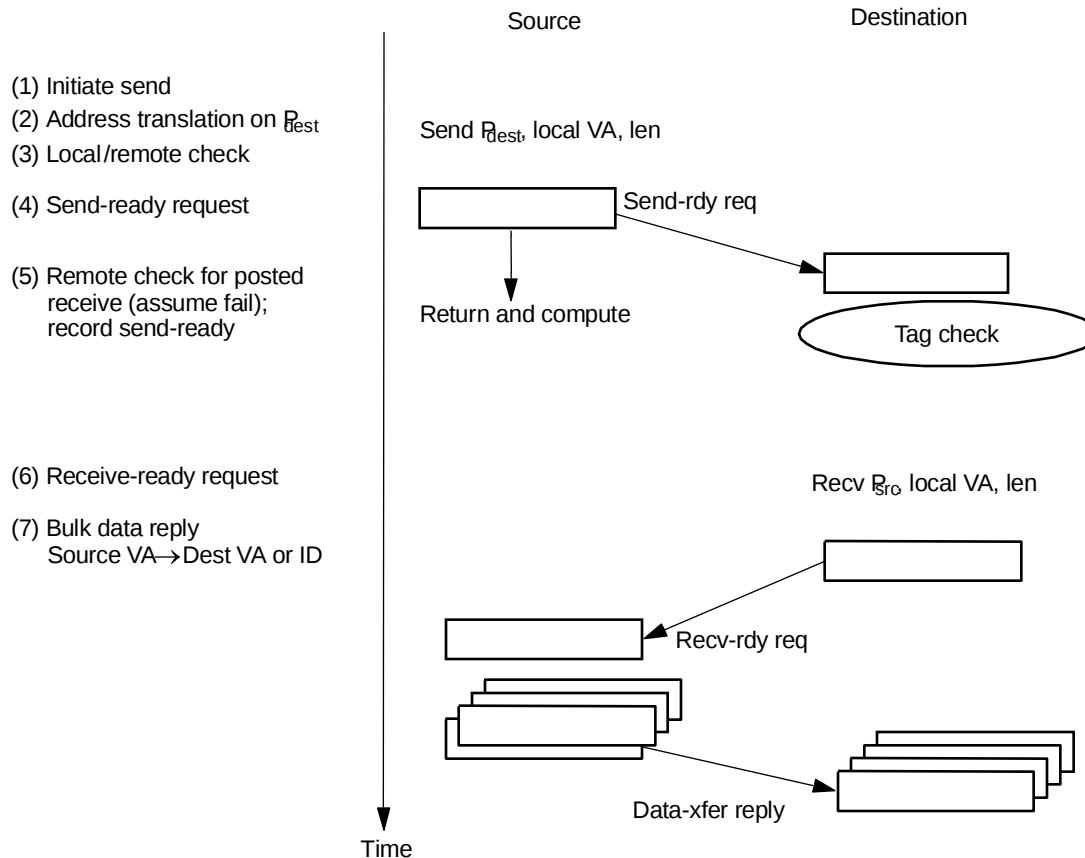
Asynch. Message Passing: Optimistic



- Tag match: se matching rcv existe ok; caso contrário receiver armazena em buffer temporário
- Problemas: (1) endereço só pode ser determinado após examinar process+tag (envelope), o que é muito custoso; (2) não há rdy-to-rcv → fácilimo estourar o buffer (mensagem grande + muitos senders)

Asynch. Msg Passing: Conservative

3 fases; send initiated

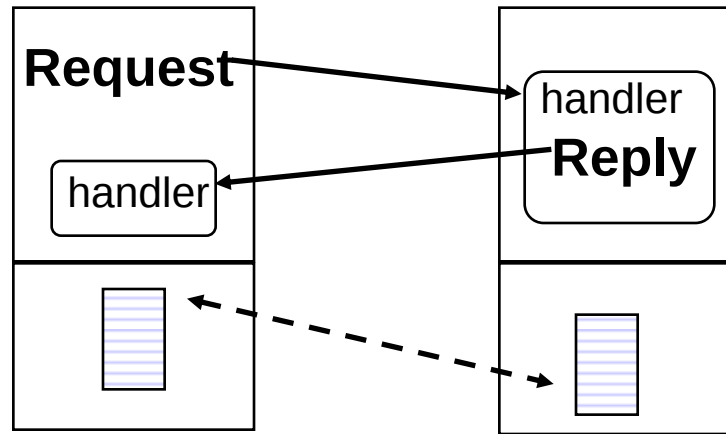


- Sender $\rightarrow$ rdy-to-send com envelope (contém process + tag) e volta a trabalhar em outra tarefa; receiver envia rdy-to-rcv se existe em tabela local um matching receive ou se há espaço no buffer; endereço nas duas pontas é conhecido antes da transmissão;

Key Features of Msg Passing Abstraction

- Conhecimento dos endereços nas duas pontas; virtual ou físico? Mapeamento V-P pode ser feito pelo communication assist. (páginas devem estar residentes durante a transferência)
- Source knows send data address, dest. knows receive data address
 - after handshake they both know both
- Fundamentally a 3-phase transaction
 - includes a request / response
 - can use optimisitic 1-phase in limited “Safe” cases
 - credit scheme (para pequenas mensagens; cada processo tem um crédito de espaço reservado que é decrementado a cada uso)

7.2.4 Active Messages



- User-level analog of network transaction (alternativa a SAS ou MsgPassing)
 - transfer data packet and invoke handler to extract it from the network and integrate with on-going computation (ponteiro de código a ser executado no nó remoto)
- Request/Reply: request (endereço do processador-destino, ID para o handler no destino, alguns dados passados como argumento); semelhante à “remote procedure call” (RPC)
- Event notification (incoming messages = invoking handlers): interrupts, polling (null message para examinar a rede, servir pedidos), events?
- May also perform memory-to-memory transfer

7.2.5 Common Challenges

- Muitas transações simultâneas. Origem e destino têm pouca informação sobre status. Várias origens podem sobrecarregar um destino.
- **Input buffer overflow**
 - N-1 queue over-commitment => must slow sources
 - reserve space per source (credit)
 - when available for reuse (como saber que espaço foi liberado)?
 - Ack or Higher level (ex: protocolo nível superior; reply indica liberação)
 - Refuse input when full
 - backpressure in reliable network; nó não consegue entregar sua mensagem, enche buffer e não consegue receber mais mensagens
 - tree saturation (crescimento no sentido das fontes das mensagens)
 - deadlock free (as mensagens são removidas/processadas e não haverá deadlock) mas, a latência pode explodir
 - Reserve ack back channel (no sentido inverso; mensagem buffer-full)
 - drop packets (NACK)
 - ou timeout; não recebeu ack, entende como falha e tenta novamente

Challenges (contd)

- **Fetch Deadlock**
 - ***For network to remain deadlock free, nodes must continue accepting messages, even when cannot source msgs***
 - what if incoming transaction is a request? (como saber?)
 - Each may generate a response, which cannot be sent! (→ mais problema)
 - What happens when internal buffering is full?
- Solução comum: logically independent request/reply networks
 - Alt1: physical networks (fisicamente separadas)
 - Alt2: virtual channels with separate input/output queues
- Outra solução: bound (limitar) requests and reserve input buffer space
 - Reserva de espaço p/ proc (se cada proc puder emitir k requests e se todos mandarem para o mesmo destino): $K(P-1)$ requests + K responses per node
 - service discipline to avoid fetch deadlock?
- Terceira solução: NACK on input buffer full

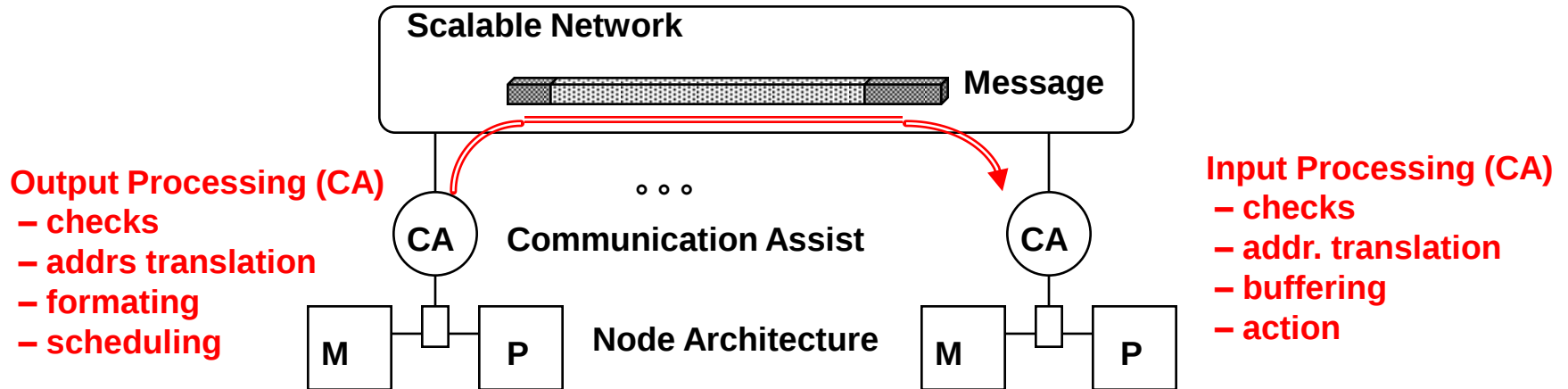
Challenges in Realizing Prog. Models in the Large

- One-way transfer of information
- No global knowledge, nor global control
 - barriers, scans, reduce, global-OR give fuzzy global state
- Very large number of concurrent transactions
- Management of input buffer resources
 - many sources can issue a request and over-commit destination before any see the effect
- Latency is large enough that you are tempted to “take risks”
(múltiplas escritas incompletas)
 - optimistic protocols
 - large transfers
 - dynamic allocation
- Many many more degrees of freedom in design and engineering of these system

Summary

- Scalability
 - physical, bandwidth, latency and cost
 - level of integration
- Realizing Programming Models
 - network transactions
 - protocols
 - safety
 - input buffer problem: N-1
 - fetch deadlock
- 7.2.6 Communication Architecture Design Space
 - how much hardware interpretation of the network transaction? (quanto é feito pelo communication assist sem o envolvimento do nó processador??)

Network Transaction Processing

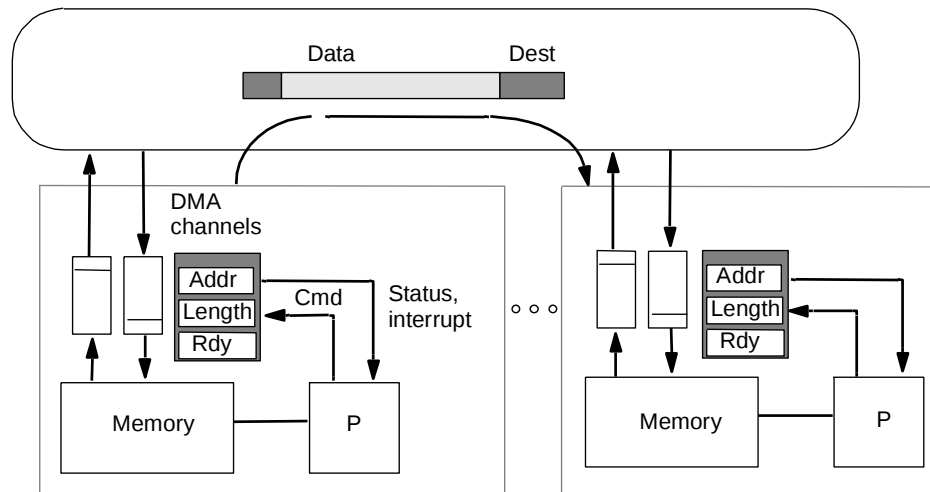


- Key Design Issue:
- How much interpretation of the message?
- How much dedicated processing in the Comm. Assist (sem envolver P)?
- Papel dos dois CAs (source & destination) tem implicações:
 - como fazer o buffering, proteção, tradução de endereço, formatação
 - interpretação dos dados: mínimo (nada, aceitar dados brutos), variável (significado das mensagens, trabalhar com endereço virtual, etc)

Spectrum of Designs

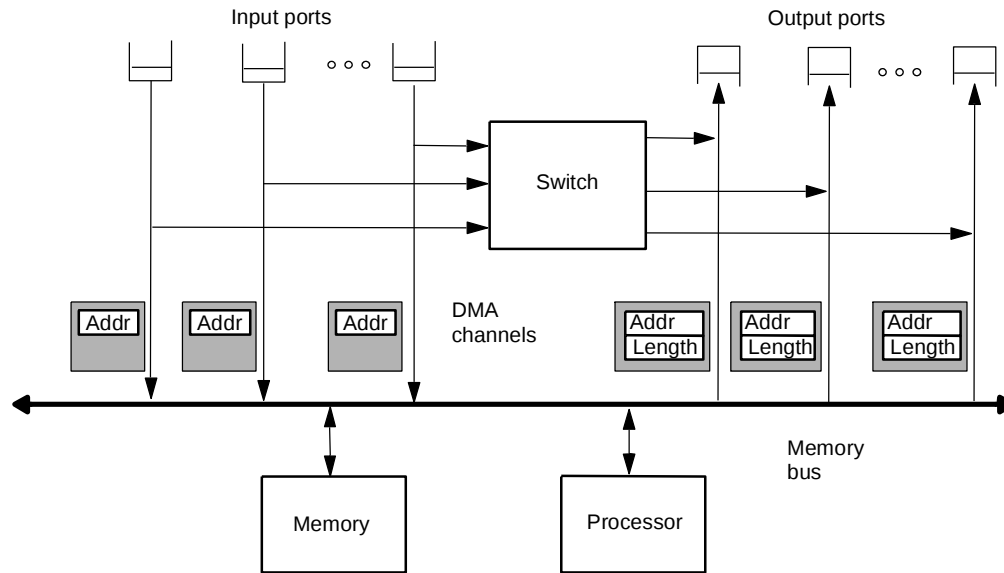
- None: Physical bit stream
 - blind, physical DMAnCUBE, iPSC, . . . (ver 7.3)
- User/System
 - User-level port
 - User-level handlerCM-5, *T
J-Machine, Monsoon, . . .
- Remote virtual address
 - Processing, translationParagon, Meiko CS-2
- Global physical address
 - Proc + Memory controllerRP3, BBN, T3D
- Cache-to-cache
 - Cache controllerDash, KSR, Flash

7.3.1 Net Transactions: Physical DMA



- Nenhuma interpretação é feita pela Network Transaction. Dados são simplesmente depositados no destino. Abordagem popular nos primeiros sistemas de MP
- DMA controlled by regs, generates interrupts (Addr., length, status)
- Physical address => 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
 - end-of-message: interrupt

7.3.3 nCUBE Network Interface



Active Message (impl HW)

primeira palavra na mens. endereço da rotina p/ tratamento

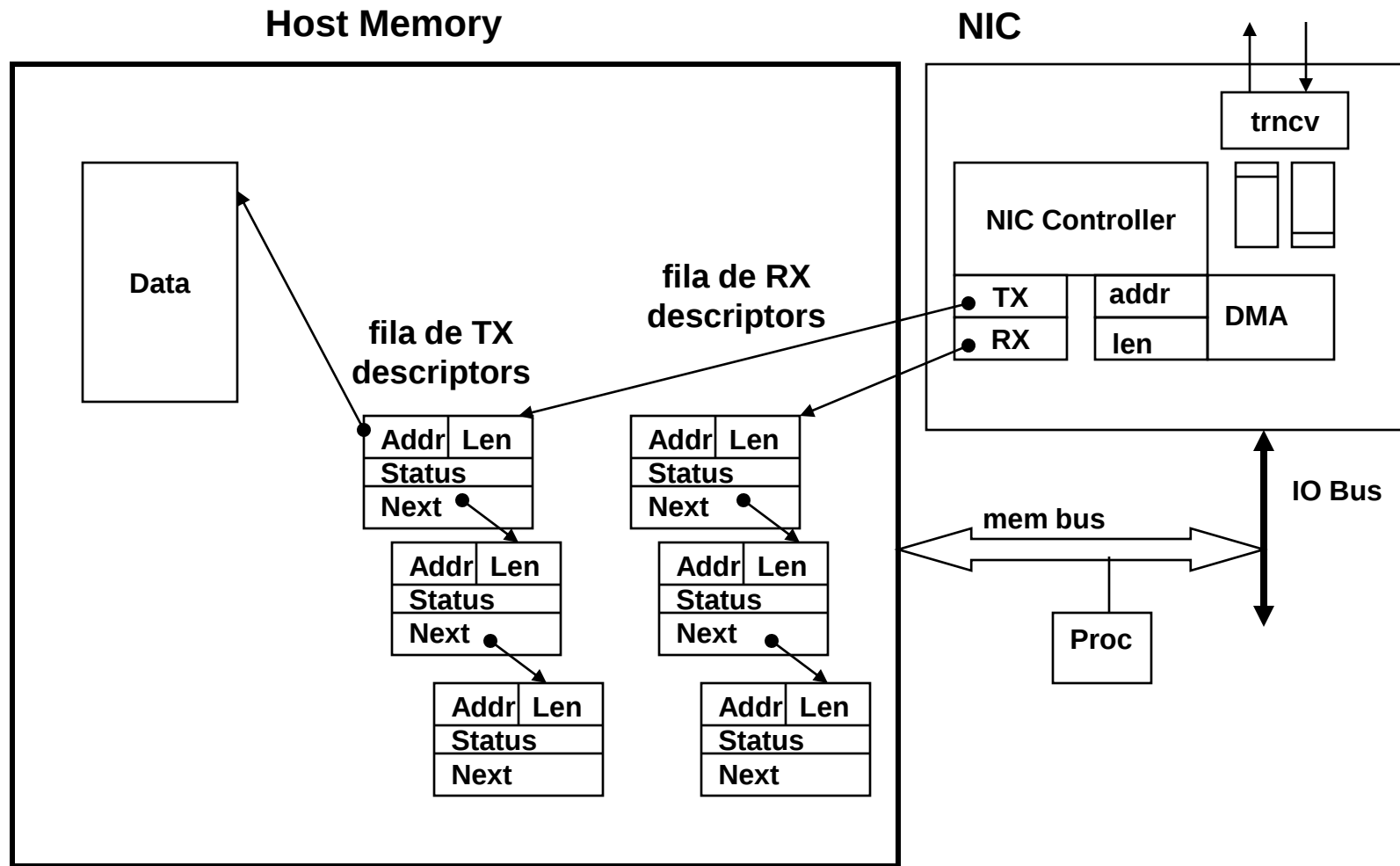
in/out	ins	ciclos	tempo
out	16	260	13 us
in	18	300	15 us

- includes interrupt

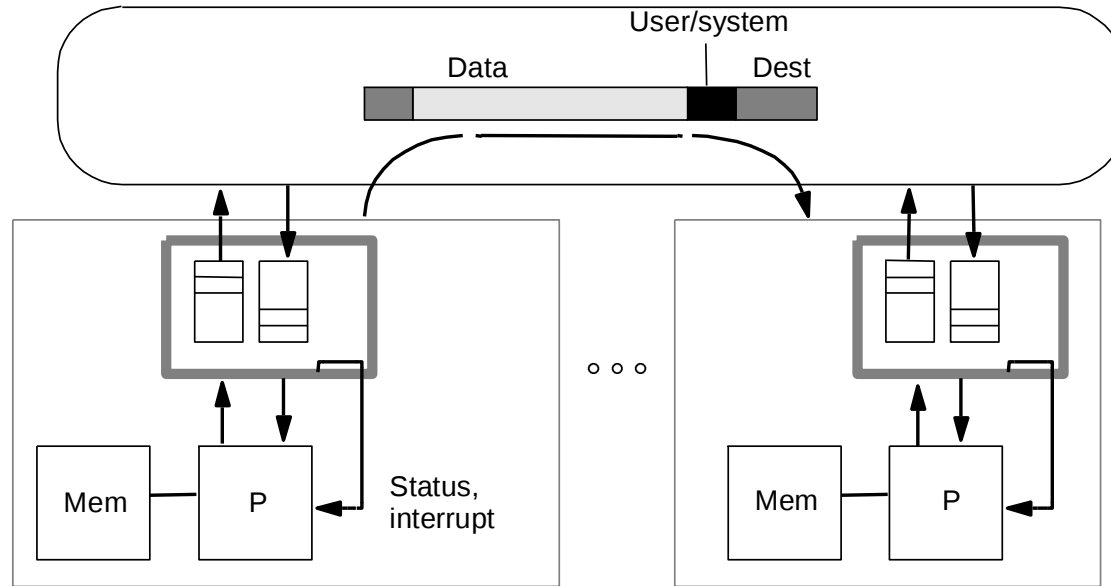
Implem. vendor message passing library: 150 us para start-up

- independent DMA channel per link direction (Direct Network)
 - leave input buffers always open
 - segmented messages (enviadas em sequencia, interpretadas seg por seg)
- routing interprets envelope (se addr ok → ativar DMA; else, rotar p/ output)
 - dimension-order routing on hypercube (n-cube = 3 neste caso)
 - link-by-link flow control
 - bit-serial with 36 bit cut-through (S&F vs cut-through = pipeline)

Conventional LAN Network Interface

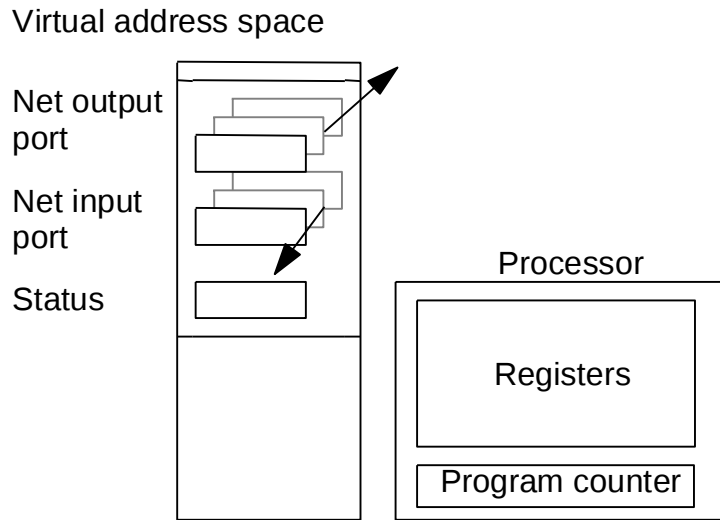


7.4 User Level Ports



- Distinguir transações de sistema e de usuário
- initiate transaction at user level
- deliver to user without OS intervention
- network port in user space
- User/system flag in envelope
 - protection check, translation, routing, media access in src CA
 - user/sys check in dest CA, interrupt on system (se for sys)

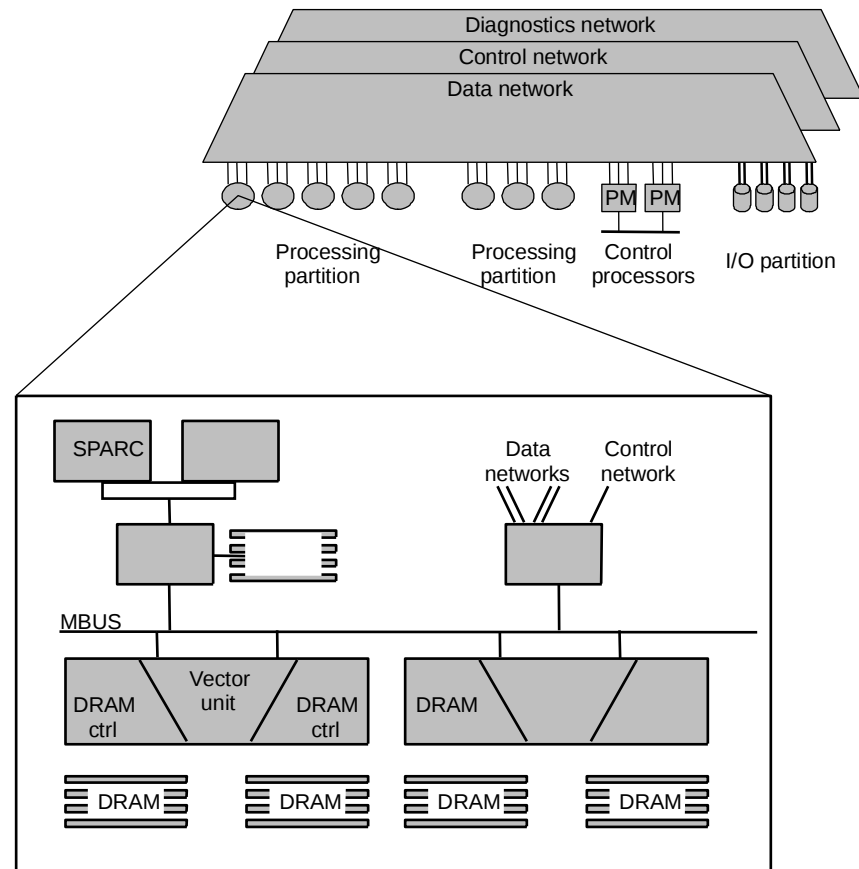
7.4.1 User Level Network ports



- Região do espaço virtual mapeada I/O ports e status register
- Processador inicia net/xaction → WR destination node # e dado no out/port
- src CA: protection check, translation ($V \rightarrow P$ ou rota), arbitra pelo acesso ao meio, insere msg type
- dest CA: sys msg → interrupção; usr msg → fila de espera pela leitura (pop queue)

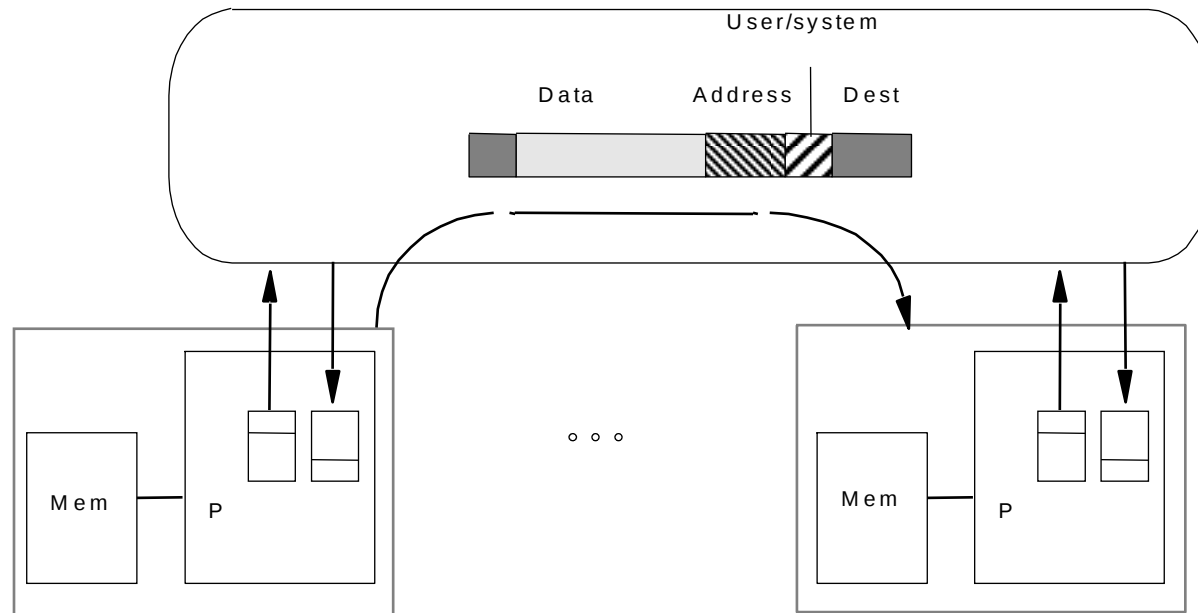
7.4.2 CM-5 (1st example of user-level network)

- CA implementado no NI
- Input and output FIFO for each network
- 2 data networks (uma para requests, outra p/ response → resolve fetch deadlock)
- Tempo para in/out mensagem é dominado pelo MBUS



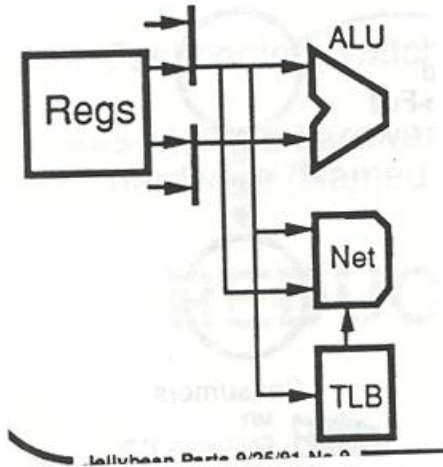
out	50 cy	1.5 us
in	53 cy	1.6 us
latência = 3-5 us p/ 1024 nós		
interrupt		10us

7.4.3 User Level Handlers

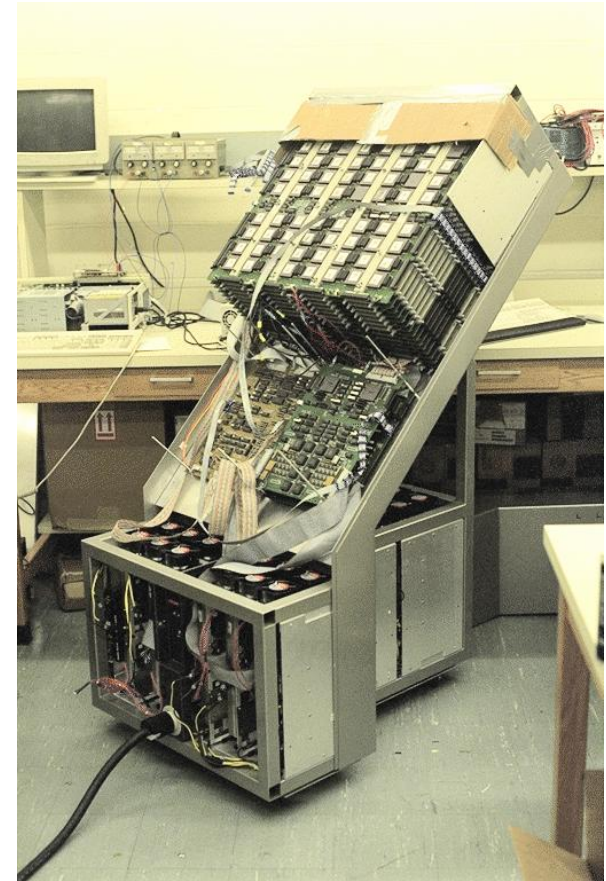


- Máquinas experimentais: maior integração do usr-level-port com o processador: neste exemplo, msg ports são registradores do processador, em vez de regiões na memória → CA é parte do processador → latência interna diminuiu, demanda de BW diminuiu, mas processador é envolvido nas transações, consumindo ciclos do proc e poluindo a cache
- CA determina que a net/xaction é destinada para um usr process → disponibiliza para o processo (cada processo tem um conjunto de FIFOs dedicadas ou as FIFOs são compartilhadas)

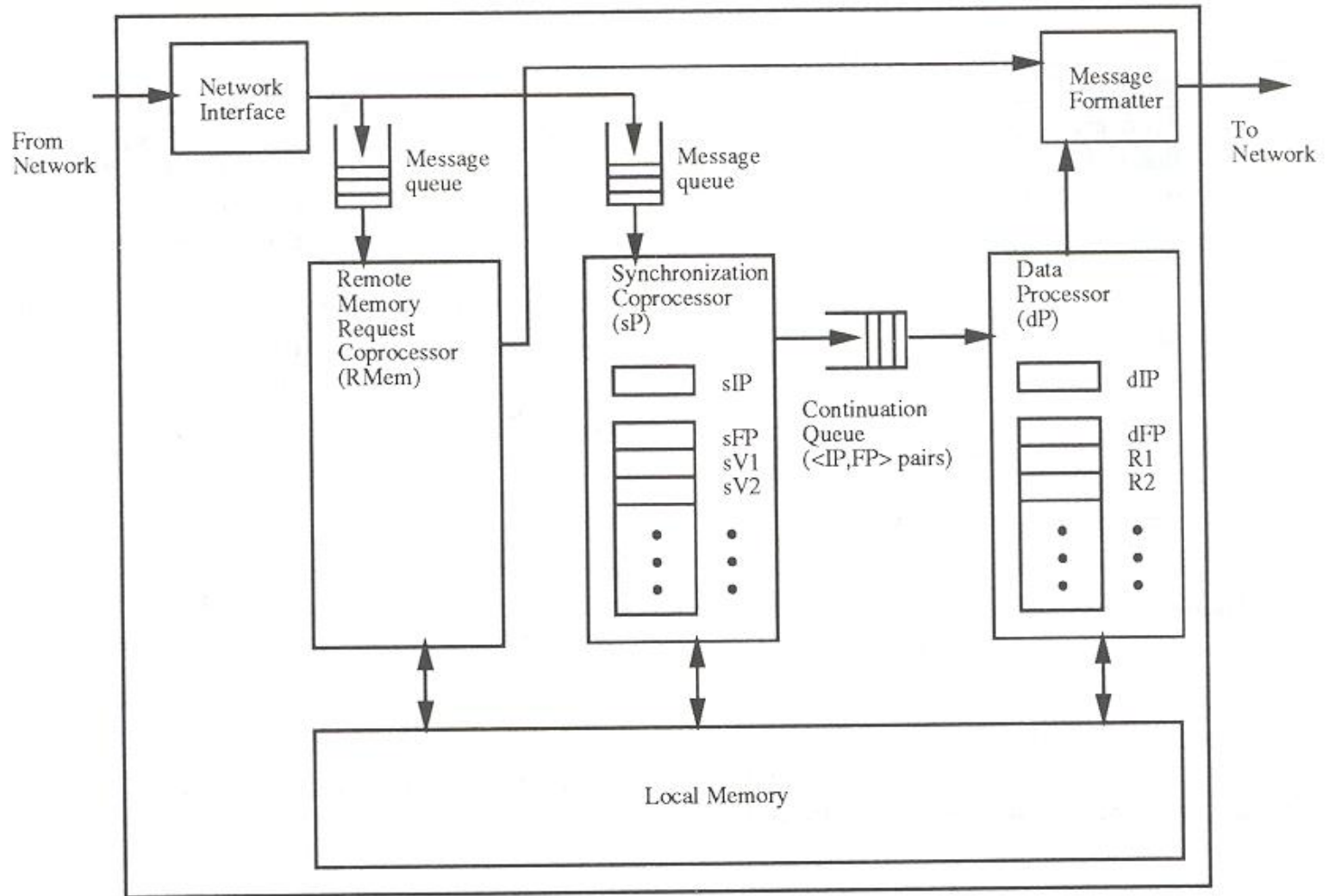
J-Machine



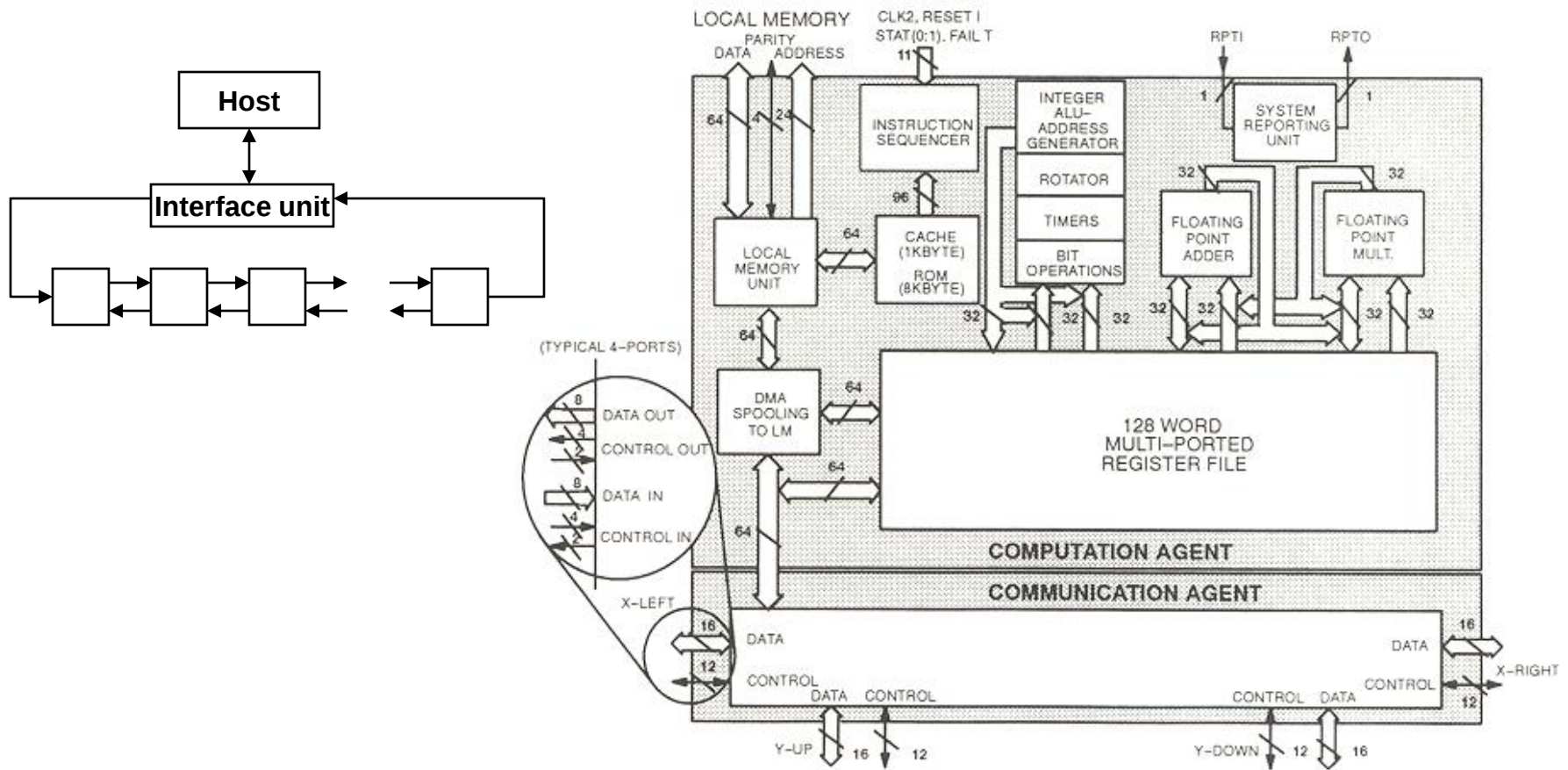
- Each node a small msg driven processor
- HW support to queue msgs and dispatch to msg handler task



*T

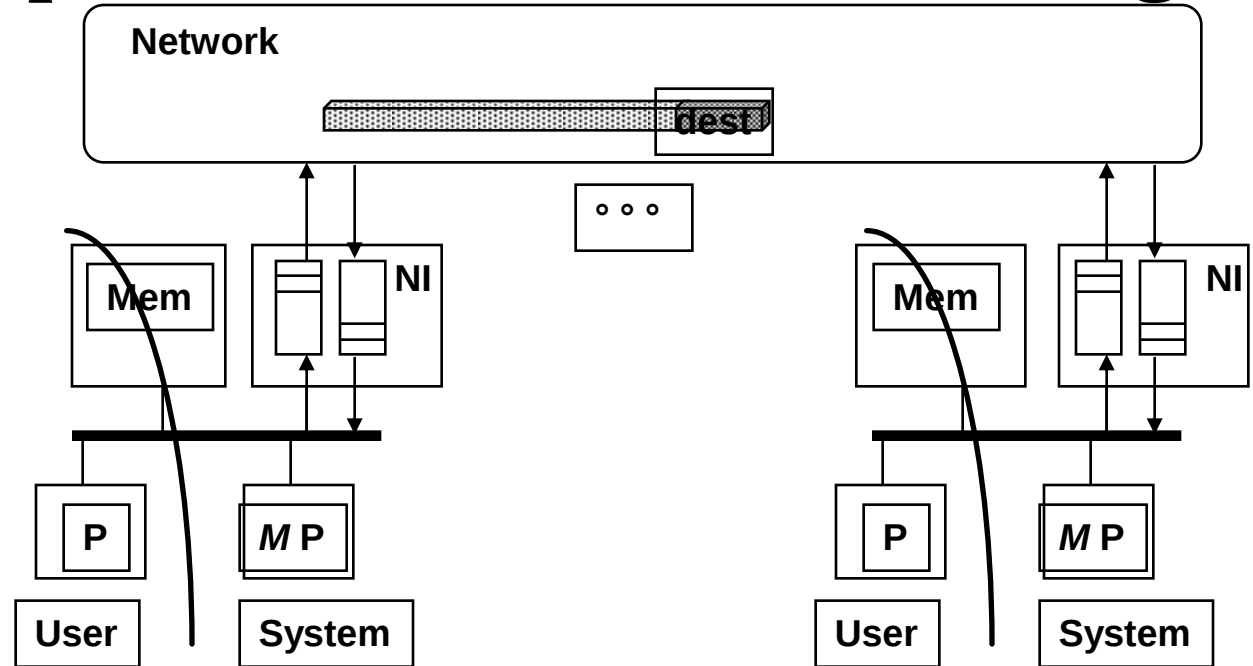


iWARP (CMU, Intel)



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

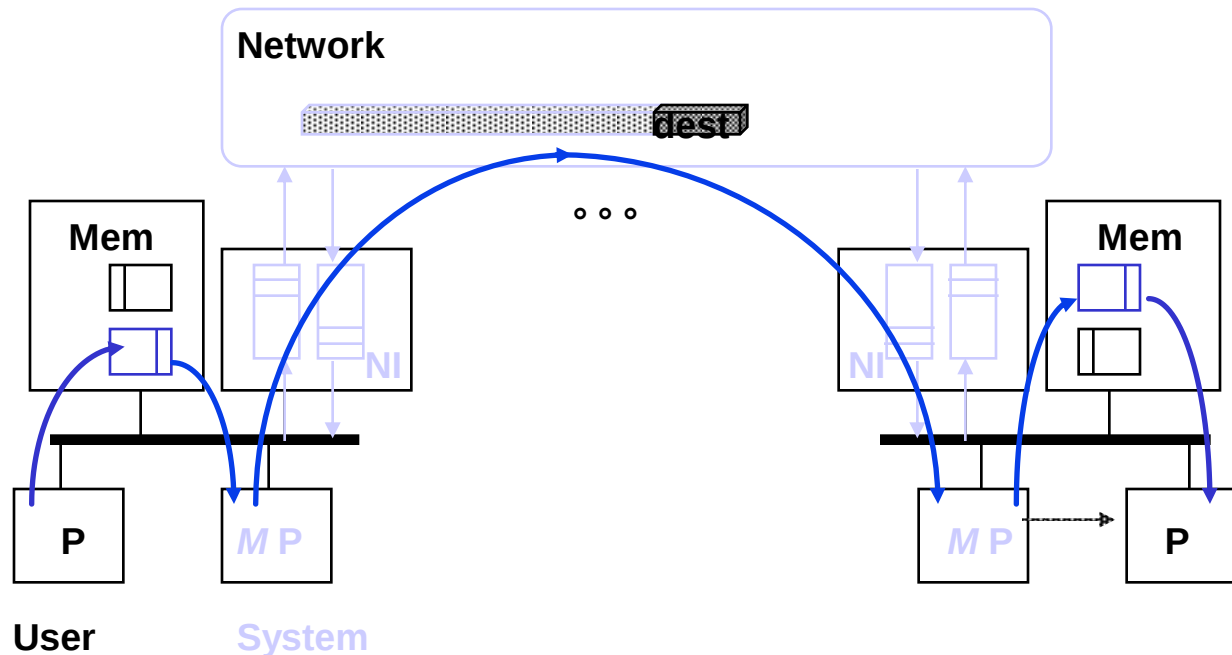
7.5 Dedicated Message Processing Without Specialized Hardware Design



- P + MP:
semelhante a
SMP com SAS
- poderia ser
MP+NI (mostrar
fig 20)
- detalhes da rede
escondidos de P
por MP

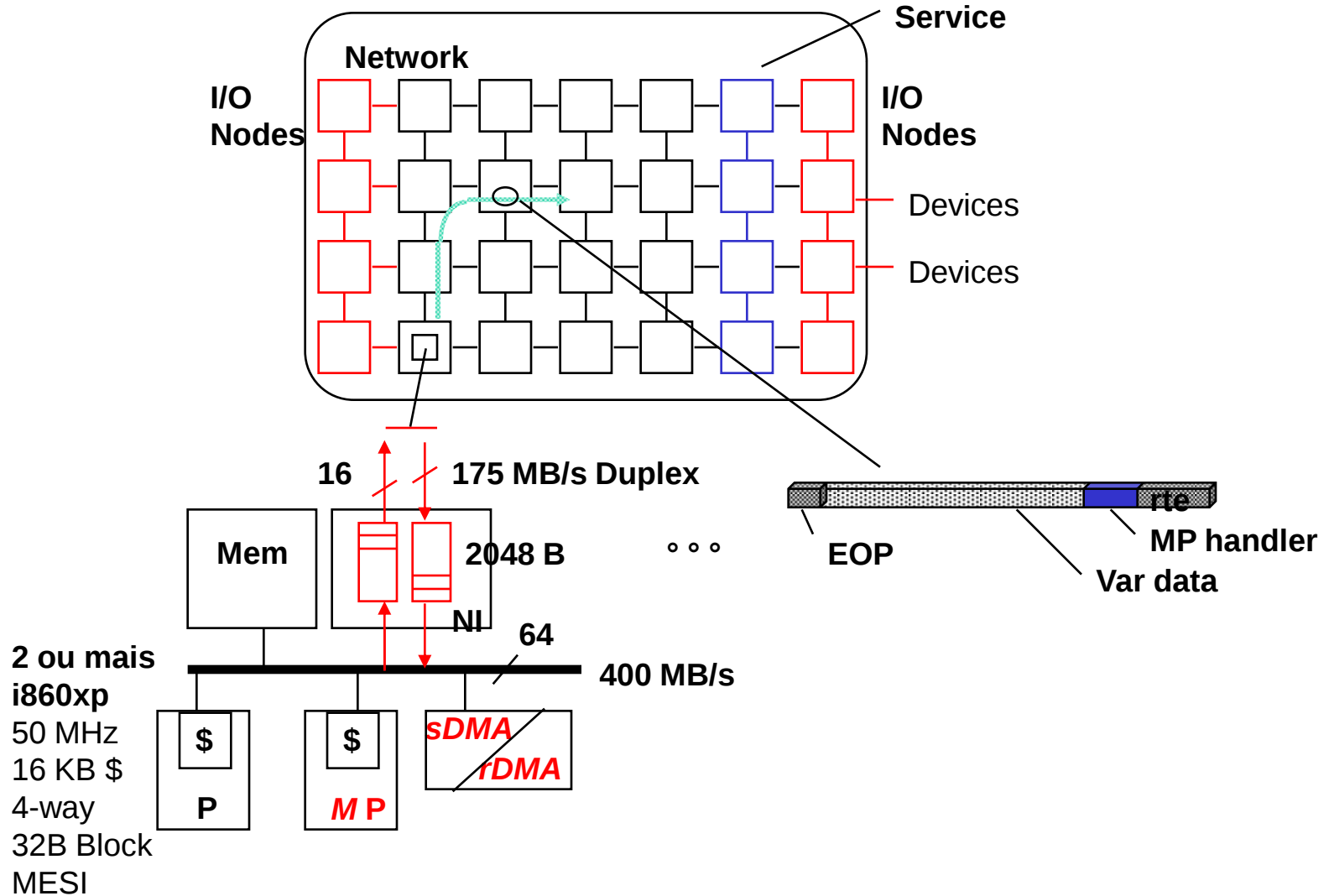
- General Purpose processor (MP) performs arbitrary output processing and interprets incoming network transactions (at system level); buffering, matching, copying/moving data, ACK, synchronization (flag)
- User Processor $\leftrightarrow$ Msg Processor share memory
- Msg Processor $\leftrightarrow$ Msg Processor via system network transaction
- Observação: MP = CP (no livro texto)

Levels of Network Transaction

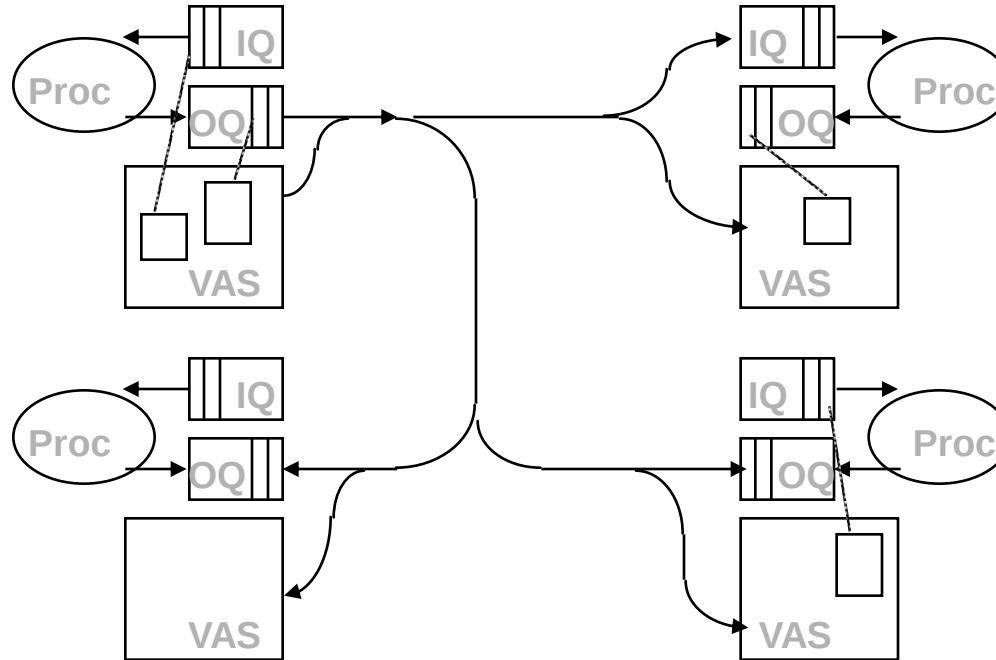


- User Processor stores cmd / msg / data into shared output queue
 - must still check for output queue full (or make elastic)
- Communication assists make transaction happen
 - checking, translation, scheduling, transport, interpretation
- Effect observed on destination address space and/or events
- Protocol divided between two layers
- Ver problemas MP vs P no texto: produtor consumidor, coerência de cache, alta concorrência no MP, troca de flags entre MP e P

7.5.1 Example: Intel Paragon ('92)

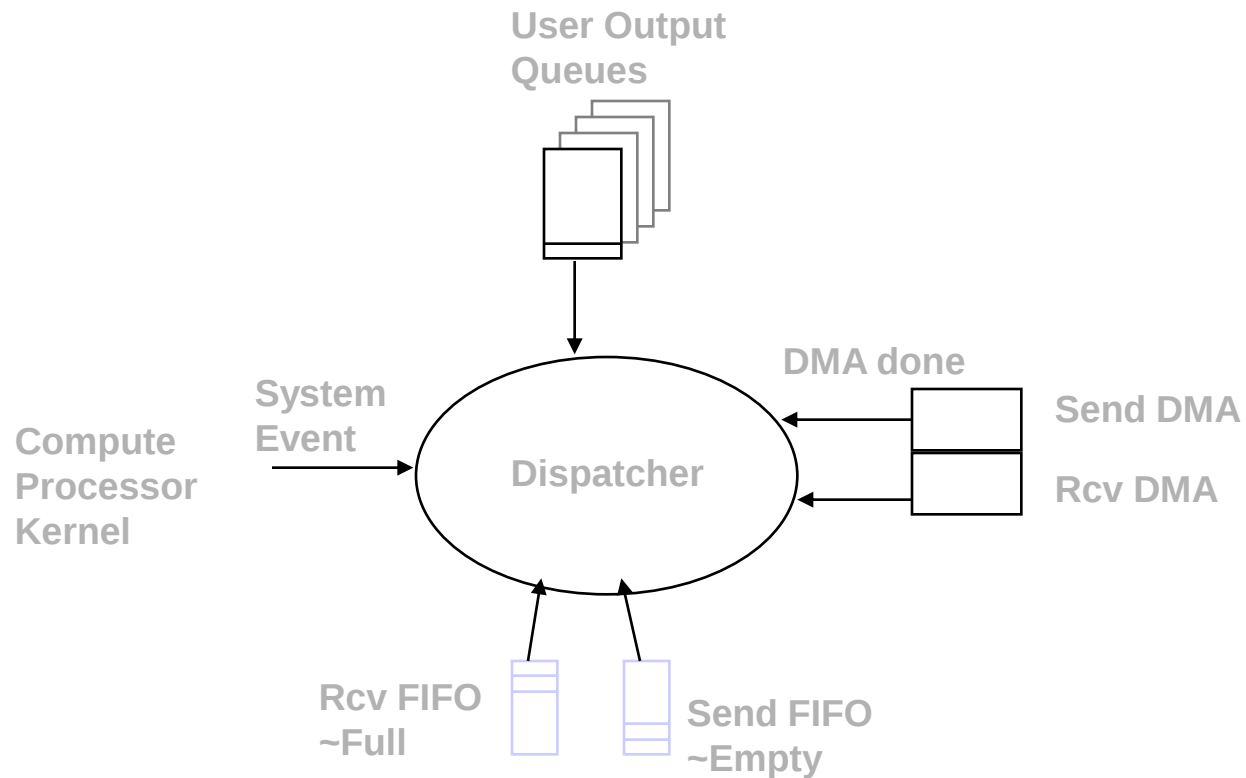


User Level Abstraction

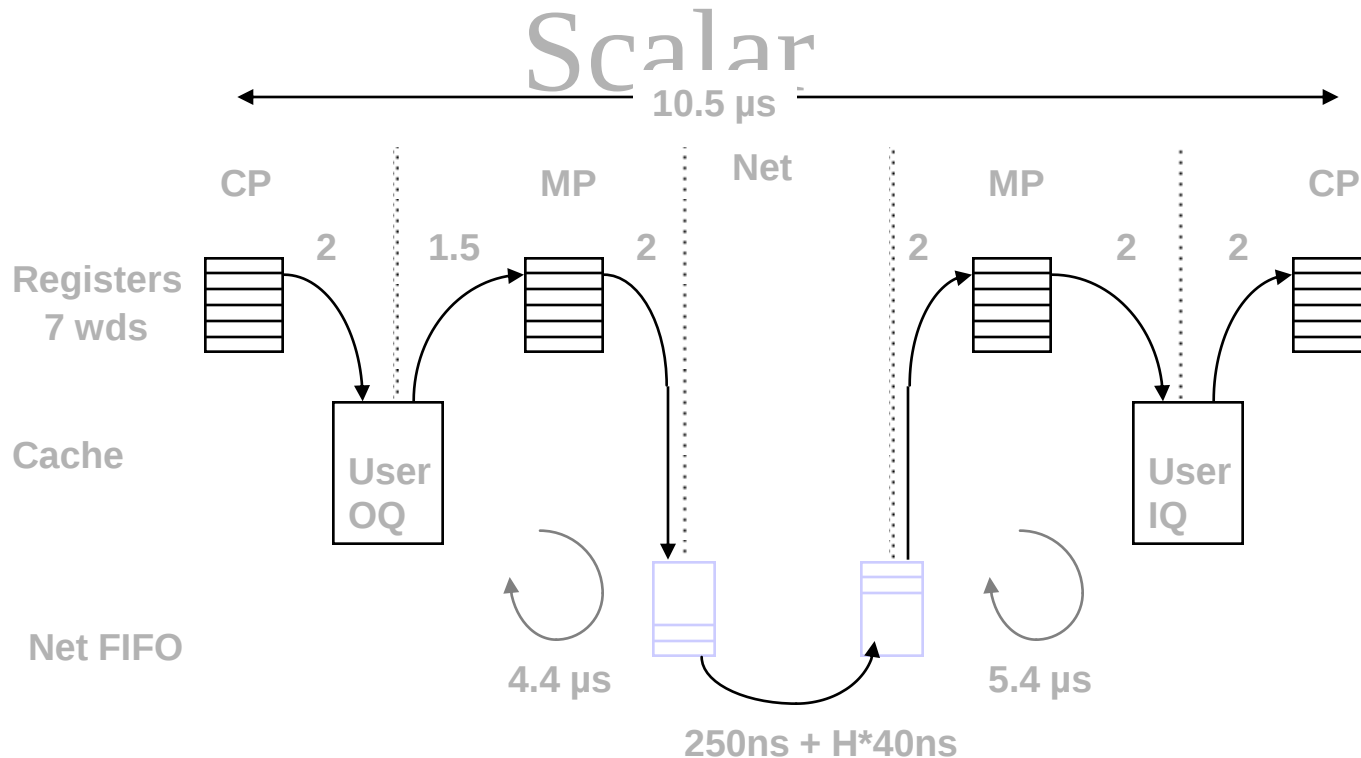


- Any user process can post a transaction for any other in protection domain
 - communication layer moves $OQ_{src} \rightarrow IQ_{dest}$
 - may involve indirection: $VAS_{src} \rightarrow VAS_{dest}$

Msg Processor Events

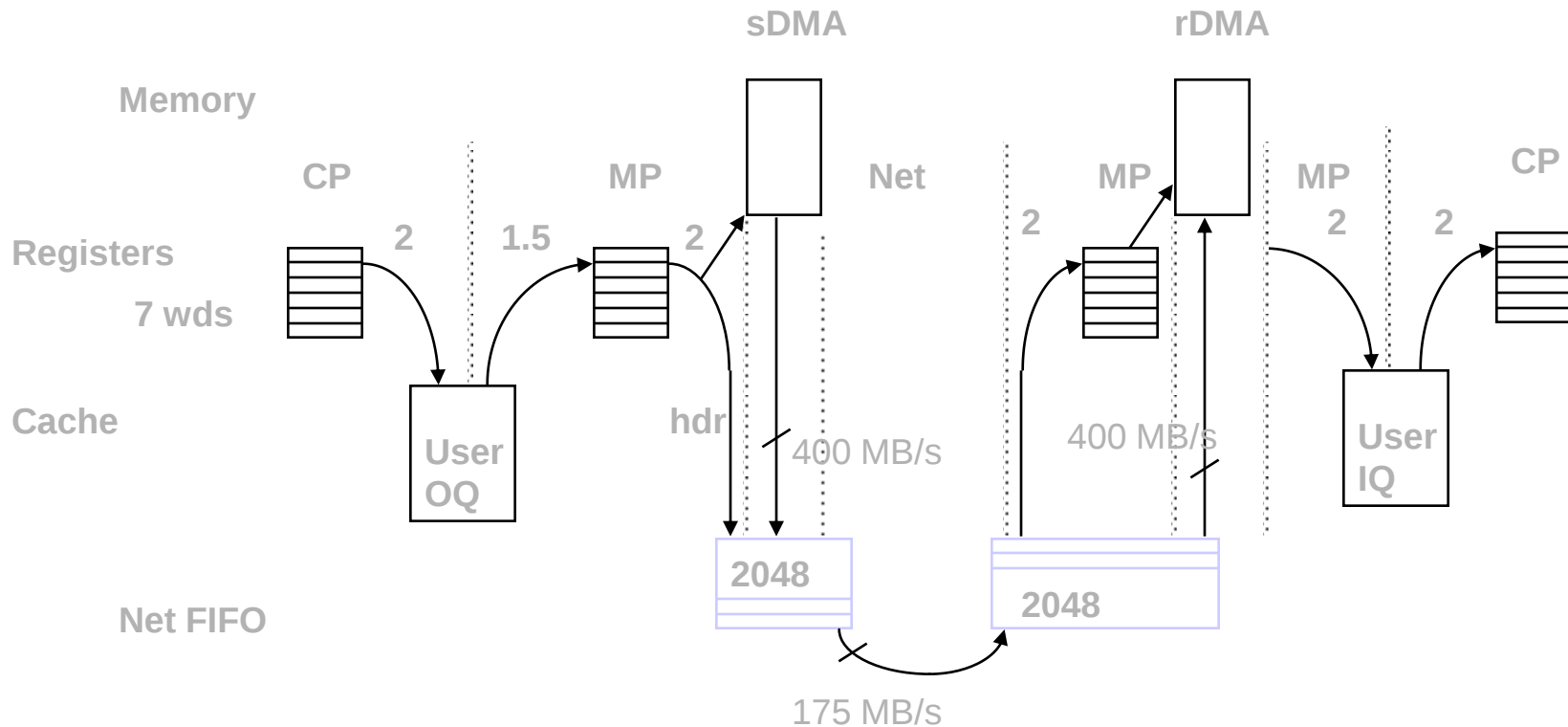


Basic Implementation Costs:



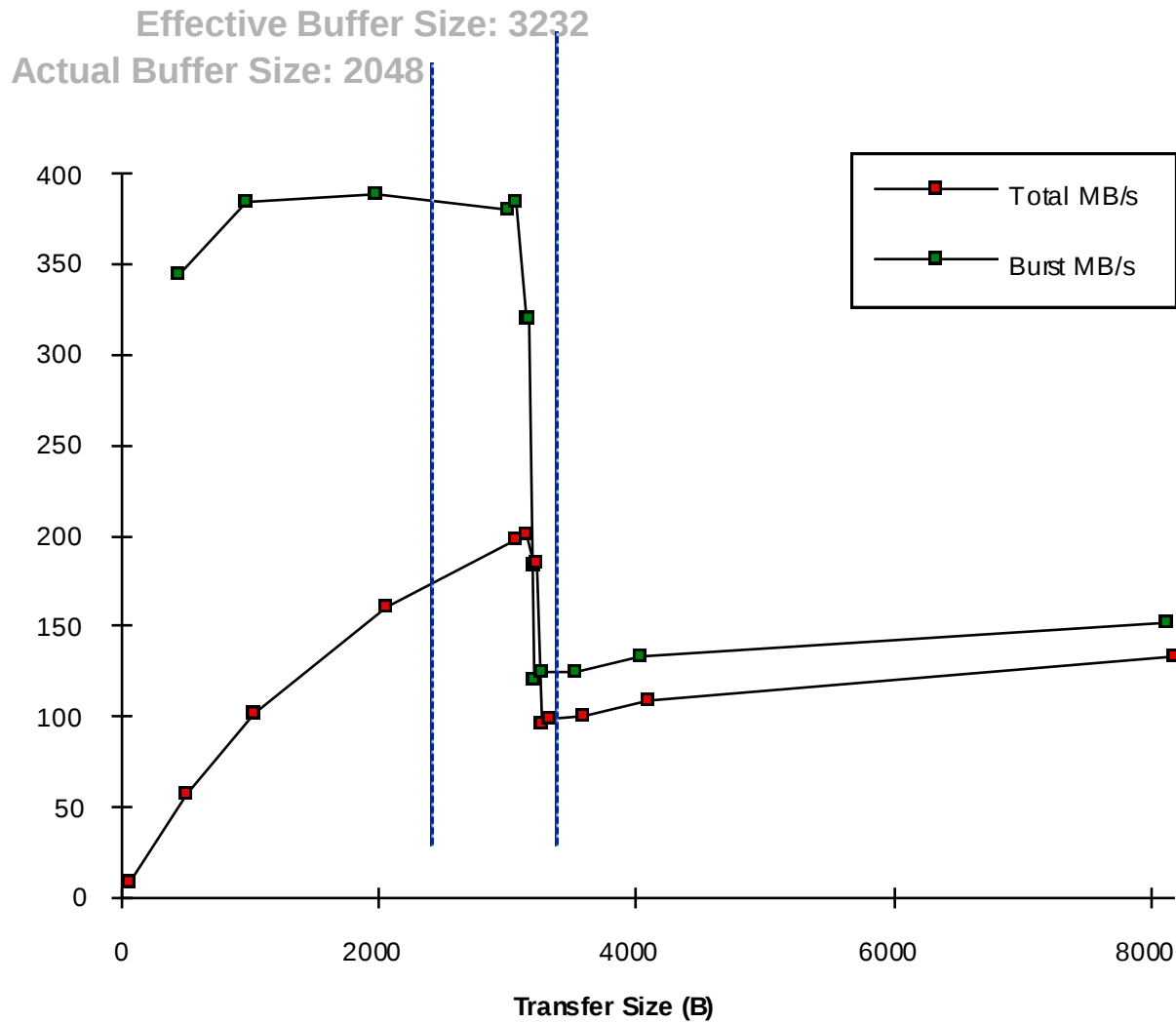
- Cache-to-cache transfer (two 32B lines, quad word ops)
 - producer: `read(miss,S), chk, write(S,WT), write(I,WT),write(S,WT)`
 - consumer: `read(miss,S), chk, read(H), read(miss,S), read(H),write(S,WT)`
- to NI FIFO: read status, chk, write, . . .
- from NI FIFO: read status, chk, dispatch, read, read,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

Virtual DMA -> Virtual DMA

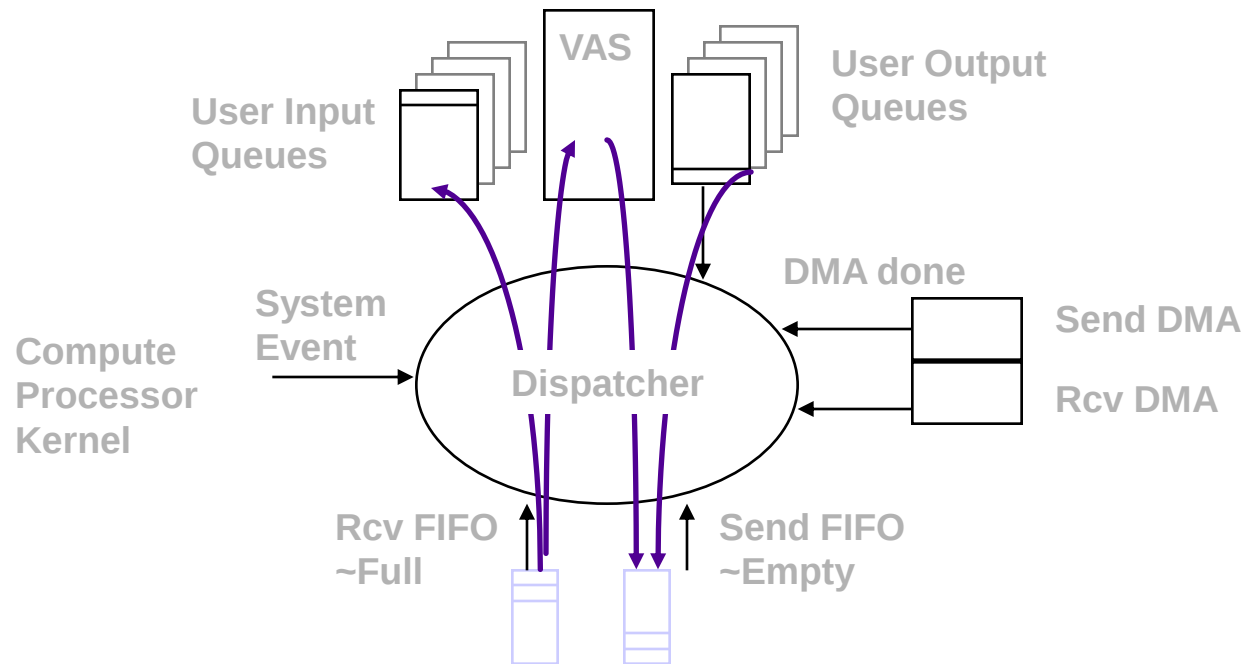


- Send MP segments into 8K pages and does VA -> PA
- Recv MP reassembles, does dispatch and VA -> PA per page

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