Multiprocessor Interconnection Networks

Todd C. Mowry

CS 740

November 19, 1998

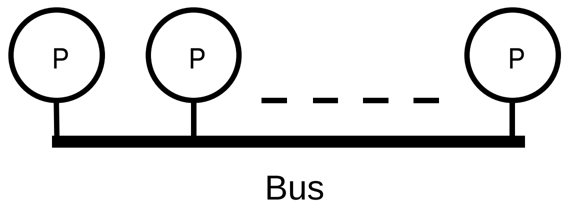
- Topics
  - Network design space
  - Contention
  - Active messages

Networks

- **Design Options:**
  - **Topology**
  - **Routing**
  - **Direct vs. Indirect**
  - **Physical implementation**

- **Evaluation Criteria:**
  - **Latency**
  - **Bisection Bandwidth**
  - **Contention and hot-spot behavior**
  - **Partitionability**
  - **Cost and scalability**
  - **Fault tolerance**

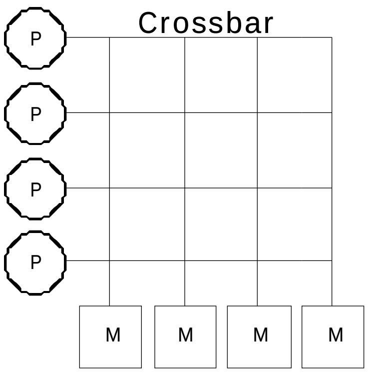
Buses



• **Simple and cost-effective for small-scale multiprocessors**

• **Not scalable (limited bandwidth; electrical complications)**

Crossbars



• Each port has link to every other port

+ Low latency and high throughput

- Cost grows as O(N^2) so not very scalable.

- Difficult to arbitrate and to get all data lines into and out of a centralized crossbar.

• Used in small-scale MPs (e.g., C.mmp) and as building block for other networks (e.g., Omega).

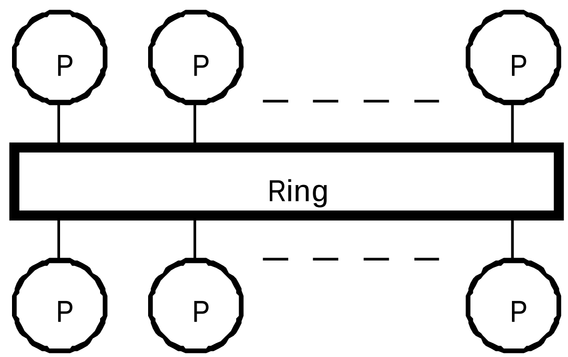
Rings

- Cheap: Cost is O(N).

- Point-to-point wires and pipelining can be used to make them very fast.

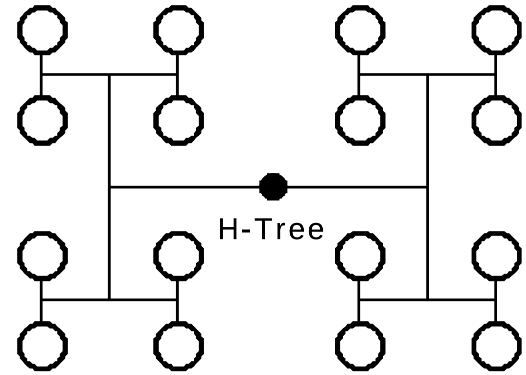
+ High overall bandwidth

- High latency O(N)

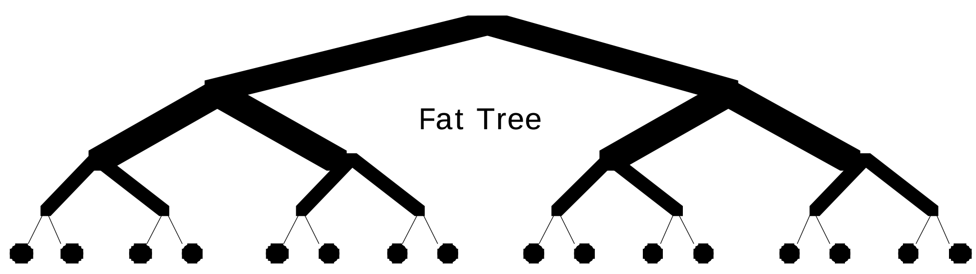


- Examples: KSR machine, Hector

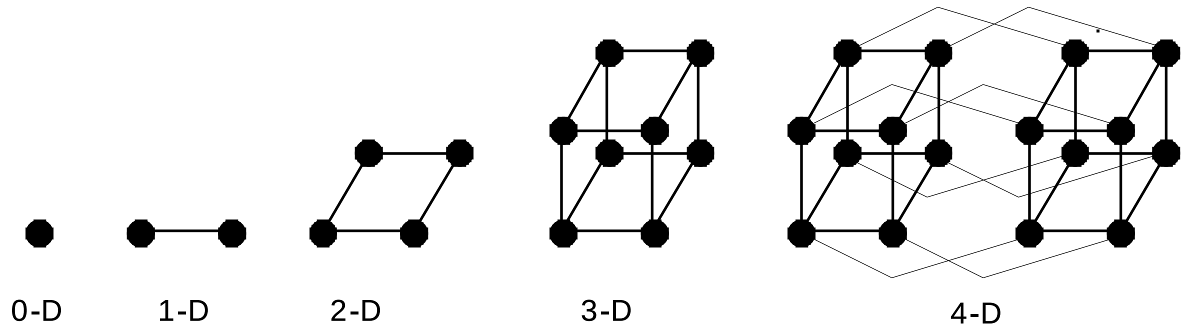
Trees



- Cheap: Cost is O(N).
- Latency is O(logN).
- Easy to layout as planar graphs (e.g., H-Trees).
- For random permutations, root can become bottleneck.
- To avoid root being bottleneck, notion of Fat-Trees (used in CM-5)
    - channels are wider as you move towards root.



Hypercubes

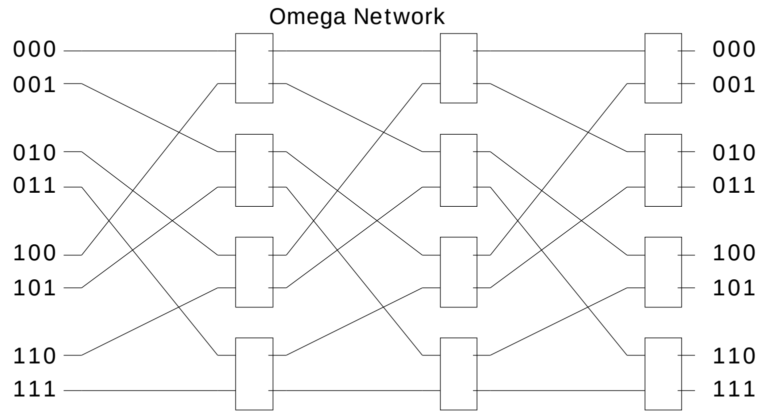


- Also called binary n-cubes.   # of nodes = N = 2^n.
- Latency is O(logN); Out degree of PE is O(logN)
- Minimizes hops; good bisection BW; but tough to layout in 3-space
- Popular in early message-passing computers (e.g., intel iPSC, NCUBE)
- Used as direct network ==> emphasizes locality

Multistage Logarithmic Networks

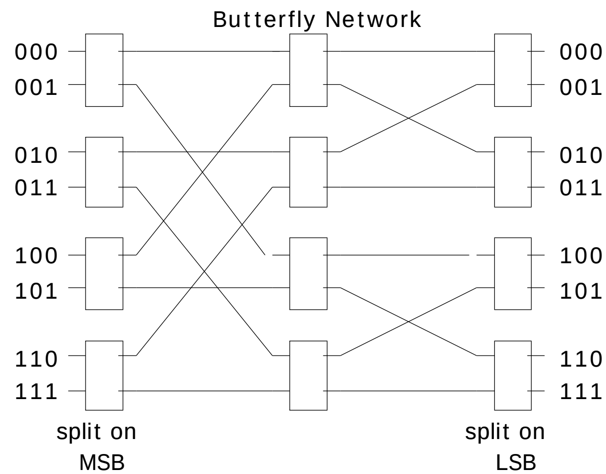
- Cost is O(NlogN); latency is O(logN); throughput is O(N).
- Generally indirect networks.
- Many variations exist (Omega, Butterfly, Benes, ...).
- Used in many machines: BBN Butterfly, IBM RP3, ...

Omega Network



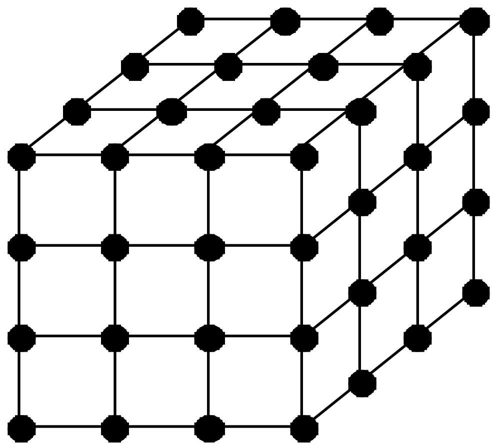
- All stages are same, so can use recirculating network.
- Single path from source to destination.
- Can add extra stages and pathways to minimize collisions and increase fault tolerance.
- Can support combining. Used in IBM RP3.

Butterfly Network



- **Equivalent to Omega network. Easy to see routing of messages.**
- **Also very similar to hypercubes (direct vs. indirect though).**
- **Clearly see that bisection of network is (N / 2) channels.**
- **Can use higher-degree switches to reduce depth. Used in BBN machines.**

k-ary n-cubes



4-ary 3-cube

- Generalization of hypercubes (k-nodes in a string)
- Total # of nodes = N = k^n.
- k > 2 reduces # of channels at bisection, thus allowing for wider channels but more hops.

Routing Strategies and Latency

- Store-and-Forward routing:
  - Tsf = Tc • (D • L / W)
  - L = msg length, D = # of hops,

    W = width, Tc = hop delay
- Wormhole routing:
  - Twh = Tc • (D + L / W)
  - # of hops is an <u>additive</u> rather than <u>multiplicative</u> factor
- Virtual Cut-Through routing:
  - Older and similar to wormhole. When blockage occurs, however, message is removed from network and buffered.
- Deadlock are avoided through use of <u>virtual channels</u> and by using a routing strategy that does not allow channel-dependency <u>cycles</u>.

Advantages of Low-Dimensional Nets

- **What can be built in VLSI is often wire-limited**
- **LDNs are easier to layout:**
  - **more uniform wiring density (easier to embed in 2-D or 3-D space)**
  - **mostly local connections (e.g., grids)**
- **Compared with HDNs (e.g., hypercubes), LDNs have:**
  - **shorter wires (reduces hop latency)**
  - **fewer wires (increases bandwidth given constant bisection width)**
    - » **increased channel width is the major reason why LDNs win!**
- **Factors that limit end-to-end latency:**
  - **LDNs: number of hops**
  - **HDNs: length of message going across very narrow channels**
- **LDNs have better hot-spot throughput**
  - **more pins per node than HDNs**

Performance Under Contention

Types of Hot Spots

- Module Hot Spots:
  - Lots of PEs accessing the same PE's memory at the same time.
  - Possible solutions:
    - suitable distribution or replication of data
    - high BW memory system design
- Location Hot Spots:
  - Lots of PEs accessing the same memory location at the same time
  - Possible solutions:
    - caches for read-only data, updates for R-W data
    - software or hardware combining

NYU Ultracomputer/ IBM RP3

- Focus on scalable bandwidth and synchronization in presence of hot-spots.

- Machine model: Paracomputer (or WRAM model of Borodin)
  - Autonomous PEs sharing a central memory
  - Simultaneous reads and writes to the same location can all be handled in a single cycle.
  - Semantics given by the serialization principle:
    - ... as if all operations occurred in some (unspecified) serial order.

- Obviously the above is a very desirable model.
  - Question is how well can it be realized in practise?
  - To achieve scalable synchronization, further extended read (write) operations with atomic read-modify-write (fetch-&-op) primitives.

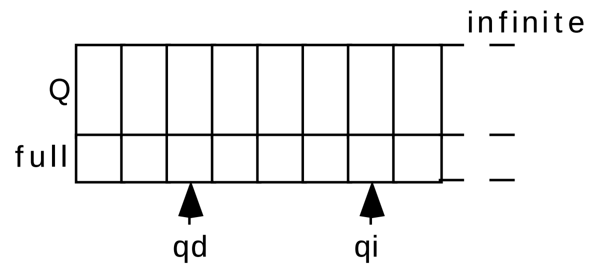
The Fetch-&-Add Primitive

- F&A(V,e) returns old value of V and atomically sets V = V + e;
- If V = k, and   X = F&A(V, a)   and Y = F&A(V, b)   done at same time
  - One possible result: X = k, Y = k+a,   and   V = k+a+b.
  - Another possible result: Y = k, X = k+b,   and   V = k+a+b.

- Example use: Implementation of task queues.

Insert: myI = F&A(qi, 1);
Q[myI] = data;
full[myI] = 1;

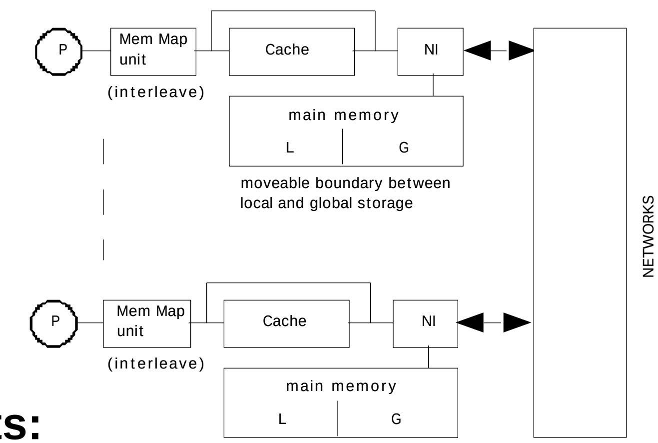
Delete: myI = F&A(qd, 1);
while (!full[myI]) ;
data = Q[myI];
full[myI] = 0;



The IBM RP3 (1985)

- Design Plan:
  - 512 RISC processors (IBM 801s)
  - Distributed main memory with software cache coherence
  - Two networks: Low latency Banyan and a combining Omega

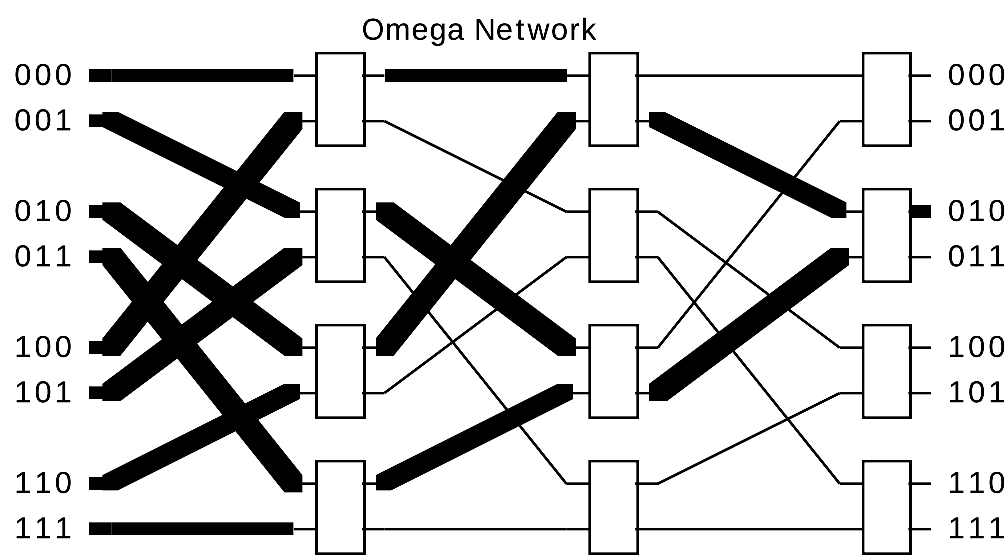
  ==> Goal was to build the NYU Ultracomputer model



- Interesting aspects:
  - Data distribution scheme to address locality and module hot spots
  - Combining network design to address synchronization bottlenecks

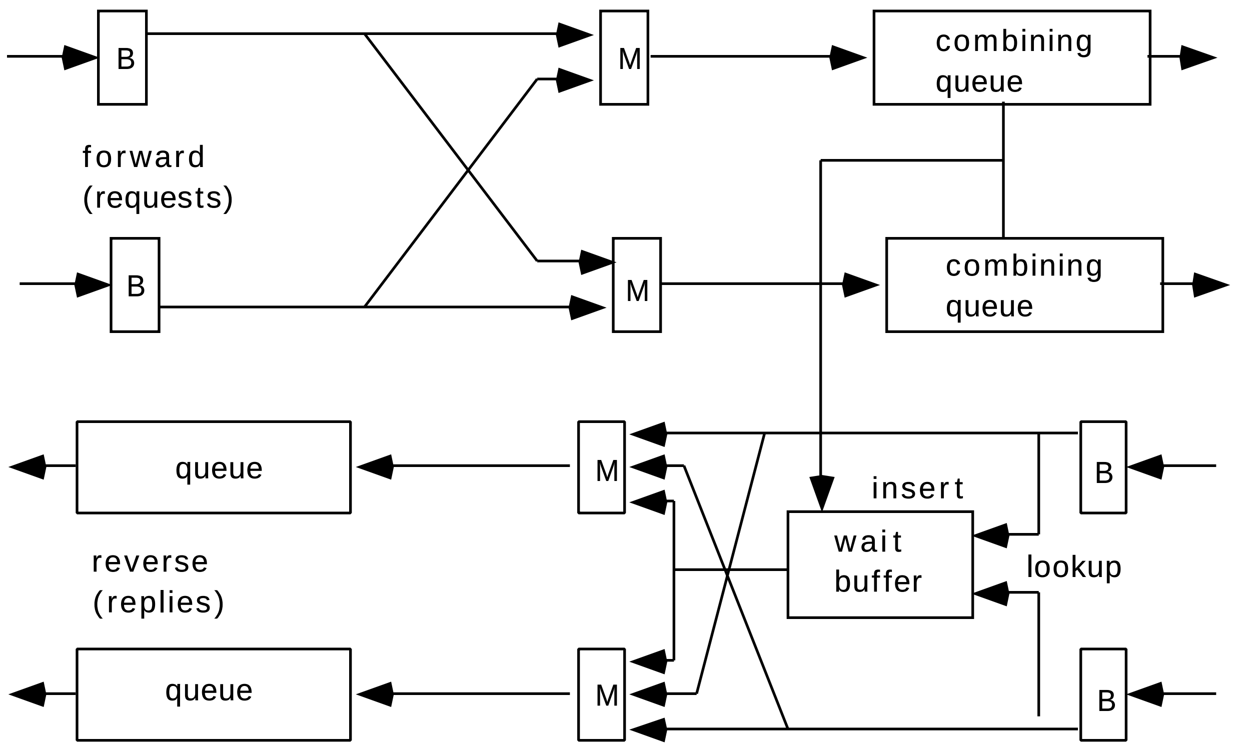
Combining Network

- Omega topology; 64-port network resulting from 6-levels of 2x2 switches.
- Request and response networks are integrated together.
- Key observation: To any destination module, the paths from all sources form a tree.



Combining Network Details

- **Requests must come together <u>locationally</u> (to same location), <u>spatially</u> (in queue of same switch), and <u>temporally</u> (within a small time window) for combining to happen.**



Contention for the Network

- **<u>Location Hot Spot</u>: Higher accesses to a single location imposed on a uniform background traffic.**
  - **May arise due to synch accesses or other heavily shared data**
  - **Not only are accesses to hot-spot location delayed, they found <u>all</u> other accesses were delayed too. (<u>Tree Saturation</u> effect.)**

Saturation Model

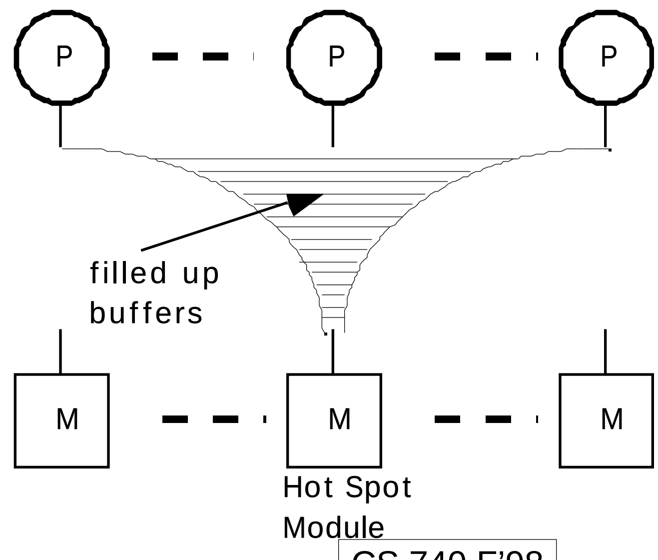
- **Parameters:**
  - **p = # of PEs; r = # of refs / PE / cycle; h = % refs from PE to hot spot**

- **Total traffic to hot-spot memory module = rhp + r(1-h)**
  - **"rhp" is hot-spot refs and "r(1-h)" is due to uniform traffic**

- **Latencies for all refs rise suddenly when [rhp + r(1-h)] = 1, assuming memory handles one request per cycle.**

- **<u>Tree Saturation Effect</u>: Buffers at all switches in the shaded area fill up, and even non-hot-spot requests have to pass through there.**

- **They found that combining helped in handling such location hot spots.**



Bandwidth Issues: Summary

- Network Bandwidth

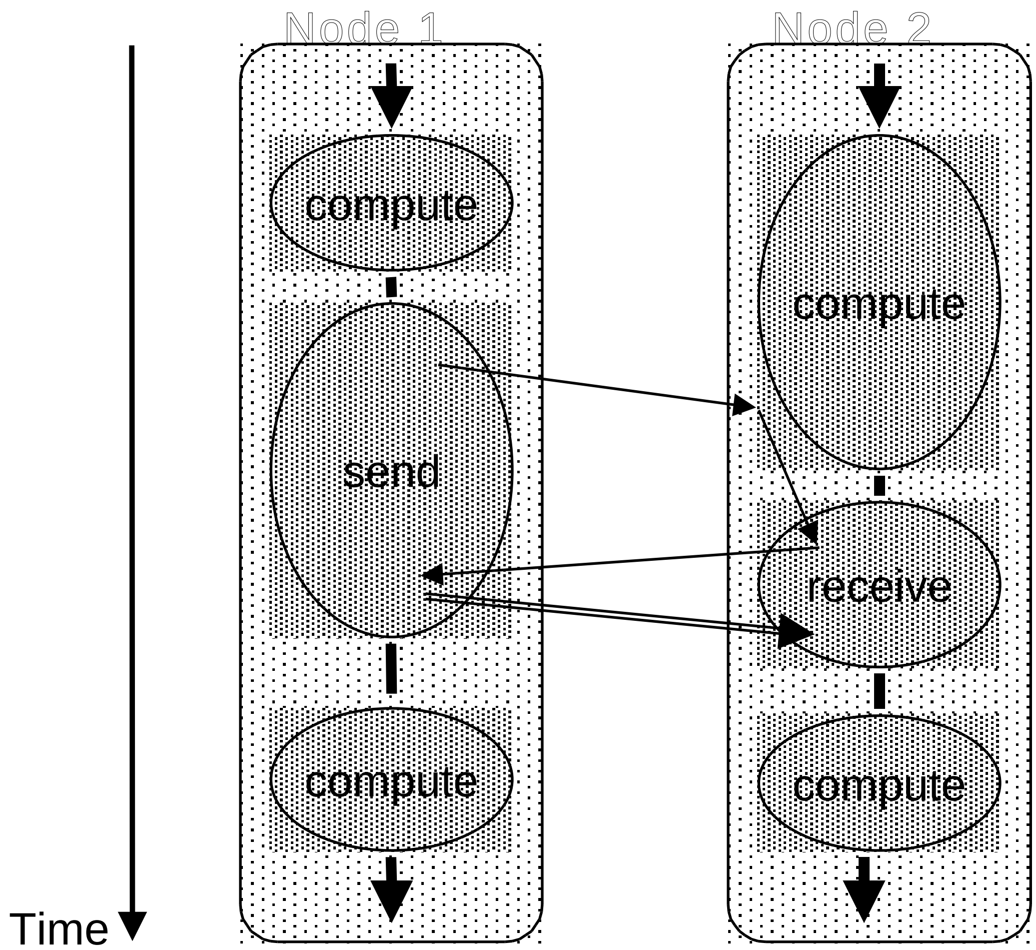
- Memory Bandwidth
  - local bandwidth
  - global bandwidth

- Hot-Spot Issues
  - module hot spots
  - location hot spots

Active Messages

(slide content courtesy of David Culler)

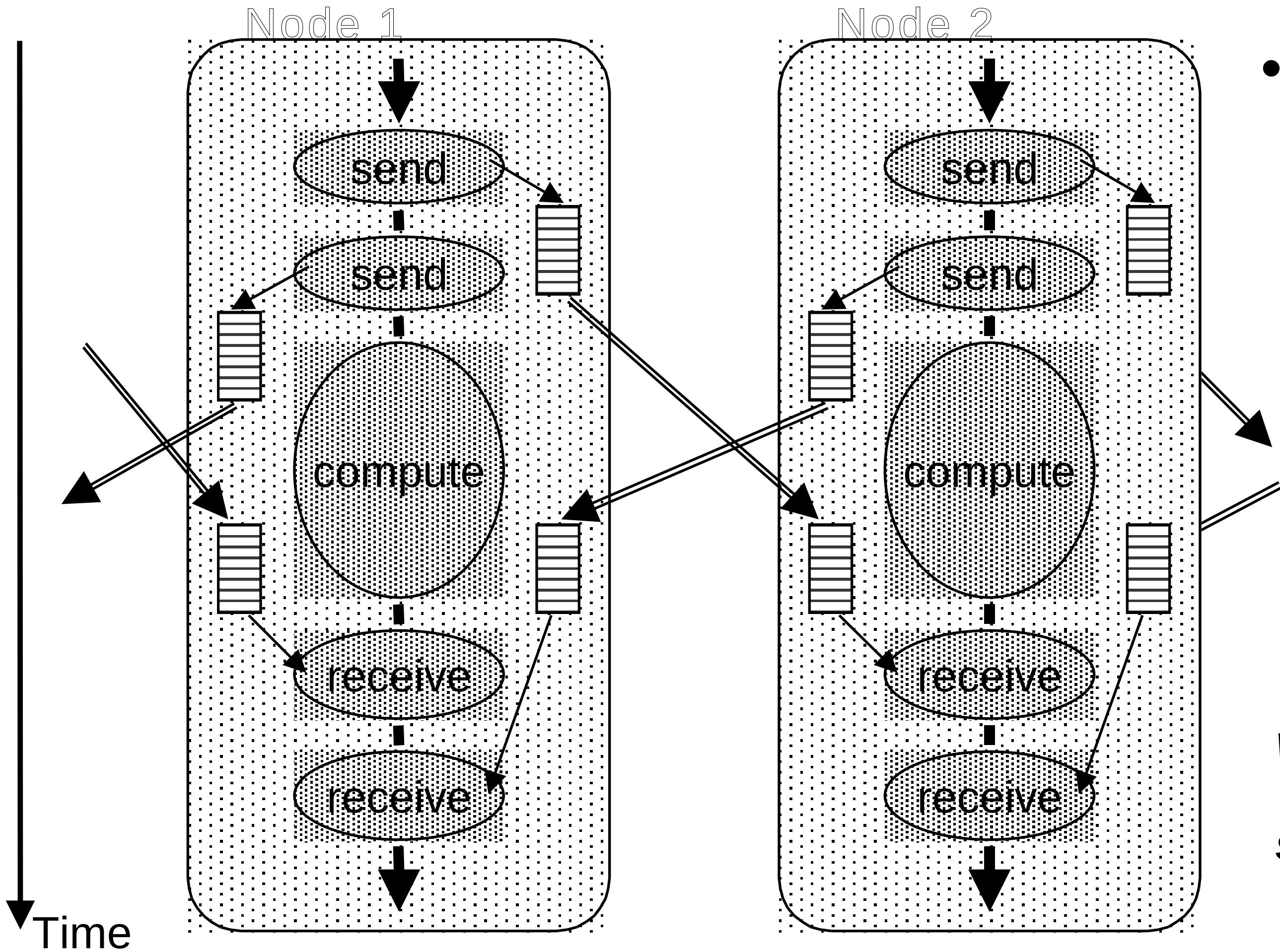
Problems with Blocking Send/Receive



- 3-way latency

Remember: back-to-back DMA hardware...

Problems w/ Non-blocking Send/Rec



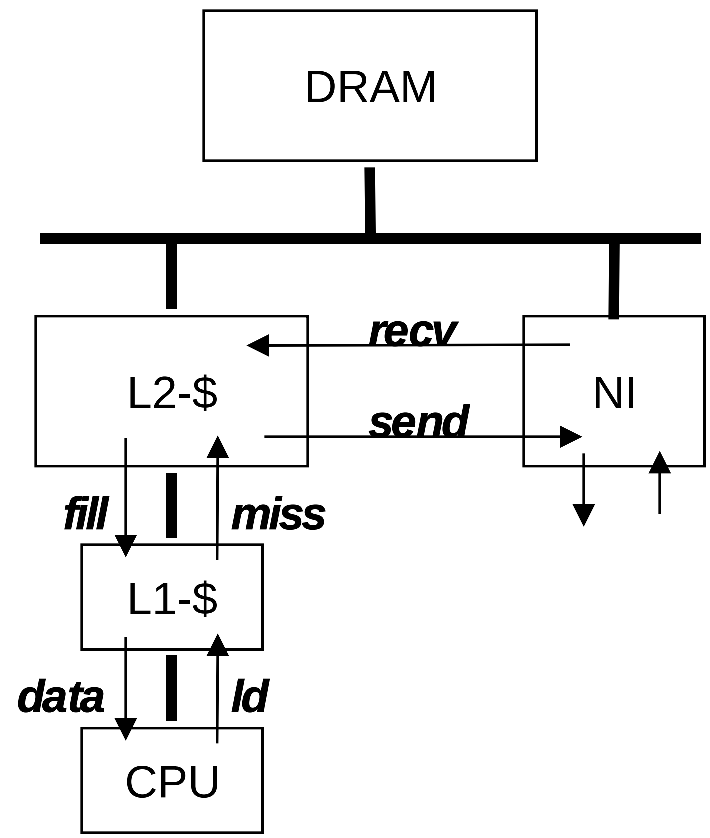
- expensive buffering

With receive hints: ``waiting-matching store´´ problem

Problems with Shared Memory

Local storage hierarchy:

- access several levels before communication starts (DASH: 30cycles)
- resources reserved by outstanding requests
- difficulty in suspending threads



Inappropriate semantics in some cases:

- only read/write cache lines
- signals turn into consistency issues

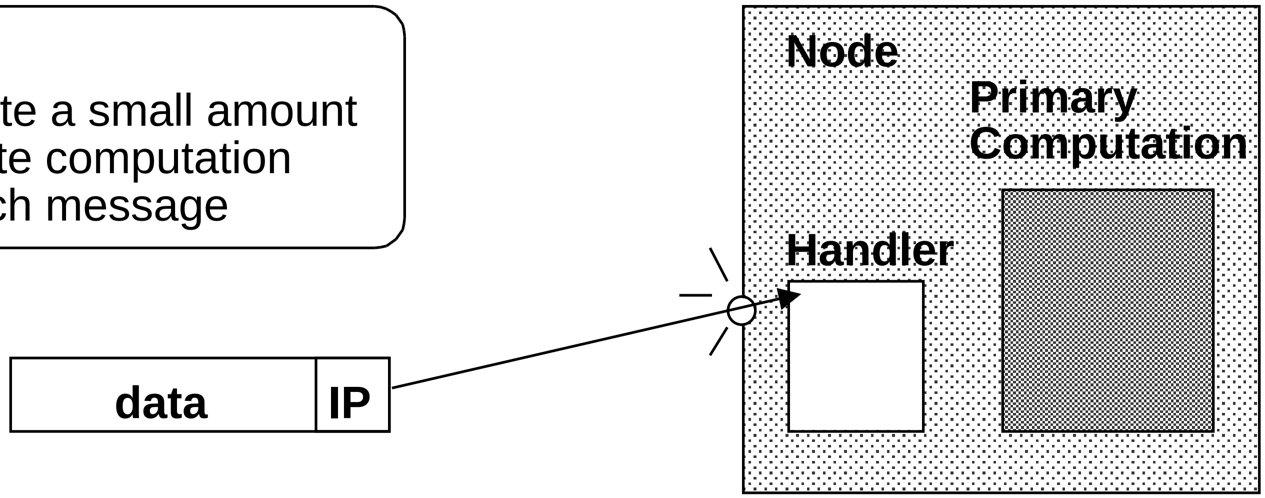
Example: broadcast tree

```
while(!me->flag);
left->data = me->data;
left->flag = 1;
right->data = me->data;
right->flag = 1;
```

Active Messages

Idea:

Associate a small amount of remote computation with each message



Head of the message is the address of its handler

Handler executes immediately upon arrival

- extracts msg from network and integrates it with computation, possibly replies
- handler does not ``compute´´

Note: user-level handler

No buffering beyond transport

- data stored in pre-allocated storage
- quick service and reply, e.g., remote-fetch

Active Message Example: Fetch&Add

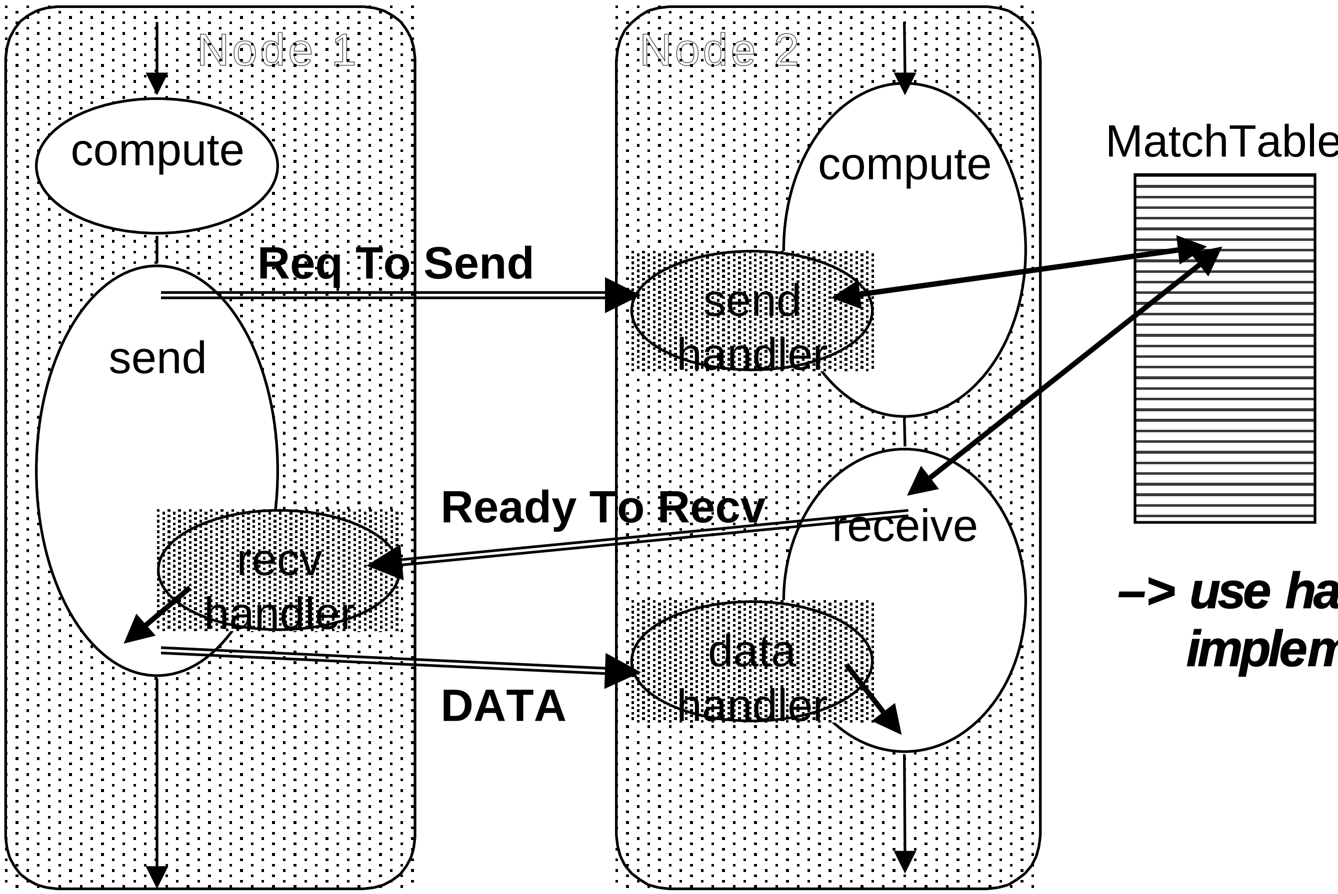
```
static volatile int value;
static volatile int flag;

int FetchNAdd(int proc,
        int *addr, int inc)
{
    flag = 0;
    AM(proc, FetchNAdd_h,
        addr, inc, MYPROC);
    while(!flag) ;
    return value;
}


void FetchNAdd_rh(int data)
{
    value = data;
    flag++;
}
```

```
void FetchNAdd_h(int *addr,
        int inc, int retproc)
{
    int v = inc + *addr;
    *addr = v;
    AM_reply(retproc,
        FetchNAdd_rh, v);
}
```

Send/Receive Using Active Messages



–> use handlers to implement protocol

Reduces send+recv overhead from 95 μsec to 3 μsec on CM-5.