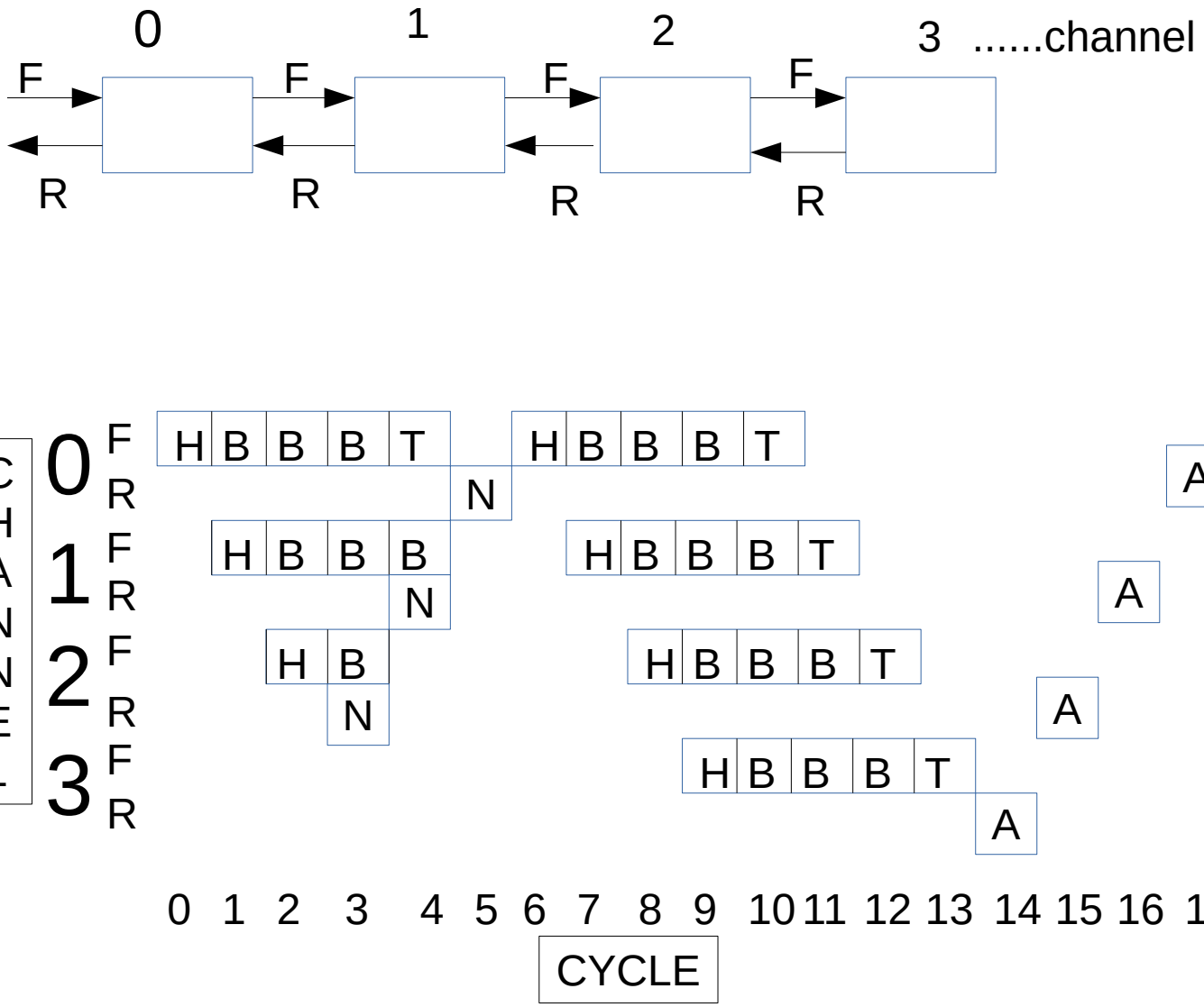
Dropping Flow Control with explicit negative acknowledgement



H	head flit
T	tail flit
B	body flit
N	nack
A	acknowledgement

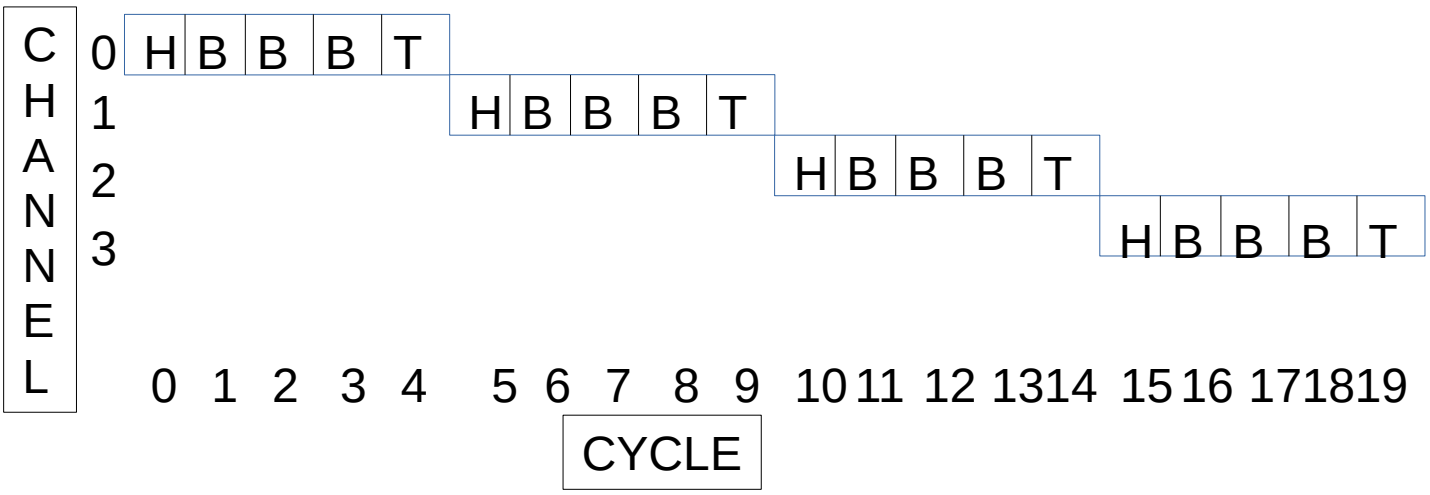
A five-flit packet is sent along a four-hop route. The first transmission of the packet is unable to allocate channel 3 and is dropped. A nack (negative acknowledgement) triggers a retransmission of the packet which succeeds.

BUFFERED FLOW CONTROL

Buffered flow control is more efficient than bufferless flow control. This is because a buffer decouples the allocation of adjacent channels. Without a buffer, the two channels must be allocated to a packet (or flit) during consecutive cycles, or the packet must be dropped or misrouted. There is nowhere else for the packet to go. Adding a buffer gives us a place to store the packet (or flit) while waiting for the second channel, allowing the allocation of the second channel to be delayed without complition. For example, a flit can be transferred over the input channel on cycle i and stored in a buffer for a number of cycles j until the output channel is successfully allocated on cycle i+j. We can approach 100% channel utilization with buffered flow control.

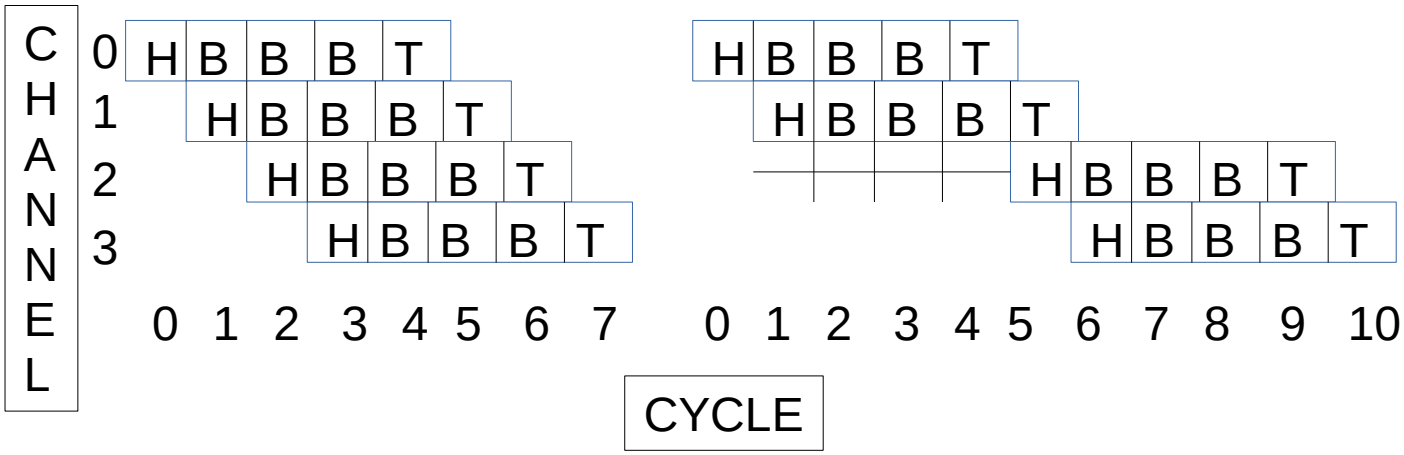
PACKET-BUFFER FLOW CONTROL

method-1 : STORE-AND-FORWARD FLOW CONTROL (HIGH LATENCY)



Time-space diagram showing store-and-forward flow-control used to send a 5-flit packet over 4 channels. With store-and-forward flow control each node along a route waits until a packet has been completely received (stored) and then forwards the packet to the next node. The packet must be allocated two resources, before it can be forwarded. A packet-sized buffer is the far side of the channel and exclusive use of the channel. Once the entire packet has arrived at a node and these two resources are acquired, the packet is forwarded to the next node. While waiting to acquire resources, if they are not immediately available, no channels are being held idle and only a single packet buffer on the current node is occupied.

Method-2 : CUT-THROUGH FLOW CONTROL



(a) Packet proceeds without contention

(b) Packet encounters contention for three cycles before it is able to allocate Channel 2

Time-space diagram showing cut-through flow control sending a 5-flit packet over 4 channels.

CUT-THROUGH : forwards a packet as soon as the header is received and resources (buffer and channel) are acquired, without waiting for the entire packet to be received.

Shortcomings of packet-buffer flow control

- Buffers are allocated in units of packets. This results in a very inefficient use of buffer storage.
- Contention latency is increased. A high priority packet colliding with a low-priority packet must wait for the entire low-priority packet to be transmitted before it can acquire the channel.

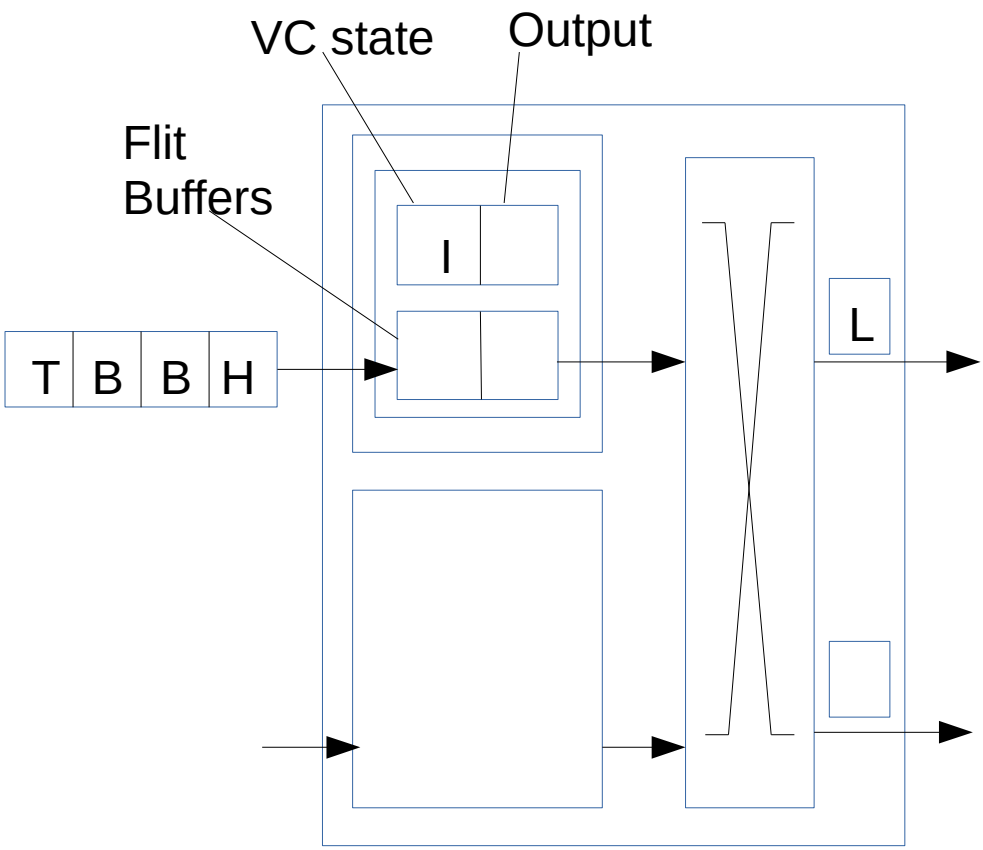
FLIT-BUFFER FLOW-CONTROL

(1) WORMHOLE Flow Control : Wormhole flow control operates like cut-thorugh, but with channel and buffers allocated to FLITS rather than packets. When the head flit of a packet arrives at a node, it must acquire three resources before it can be forwarded to the next node along a route:

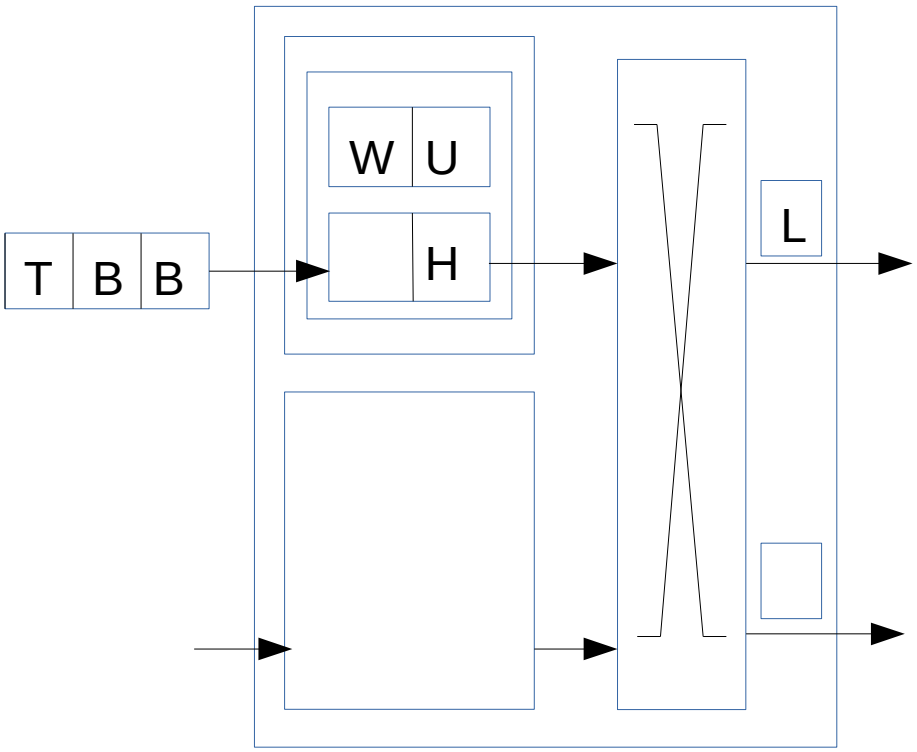
- a virtual channel (channel state) for the packet.
- one flit buffer.
- one flit of channel bandwidth.

Body flits of a packet use the virtual channel acquired by the head flit and hence need only acquire a flit buffer and a flit of channel bandwidth to advance. The tail flit of a packet is handled like a body flit, but also releases the virtual channel as it passes.

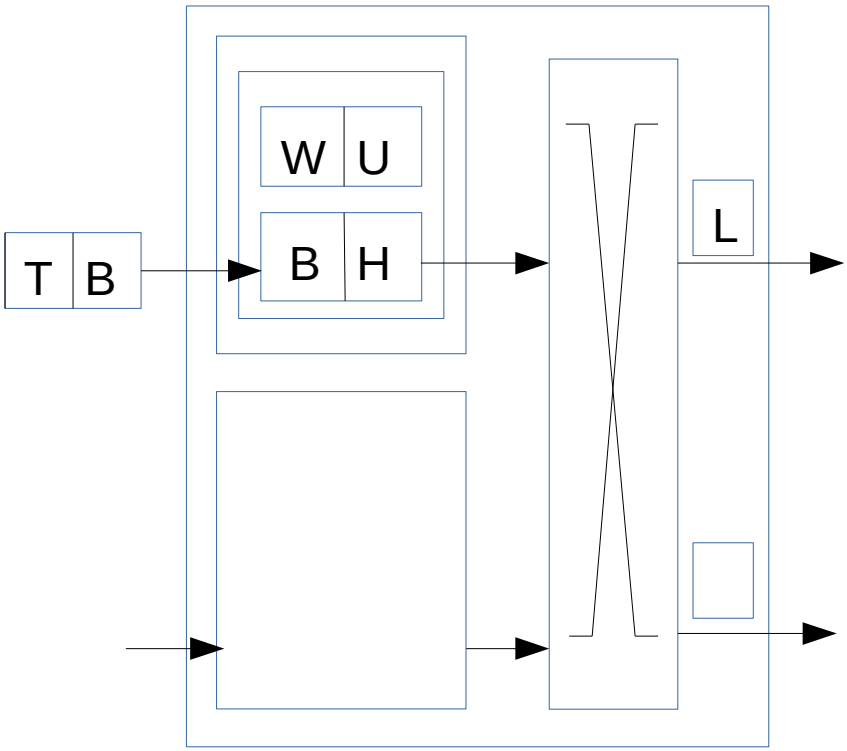
A virtual channel holds the state needed to coordinate the handling of the flits of a packet over the channel (output channel, state etc.)



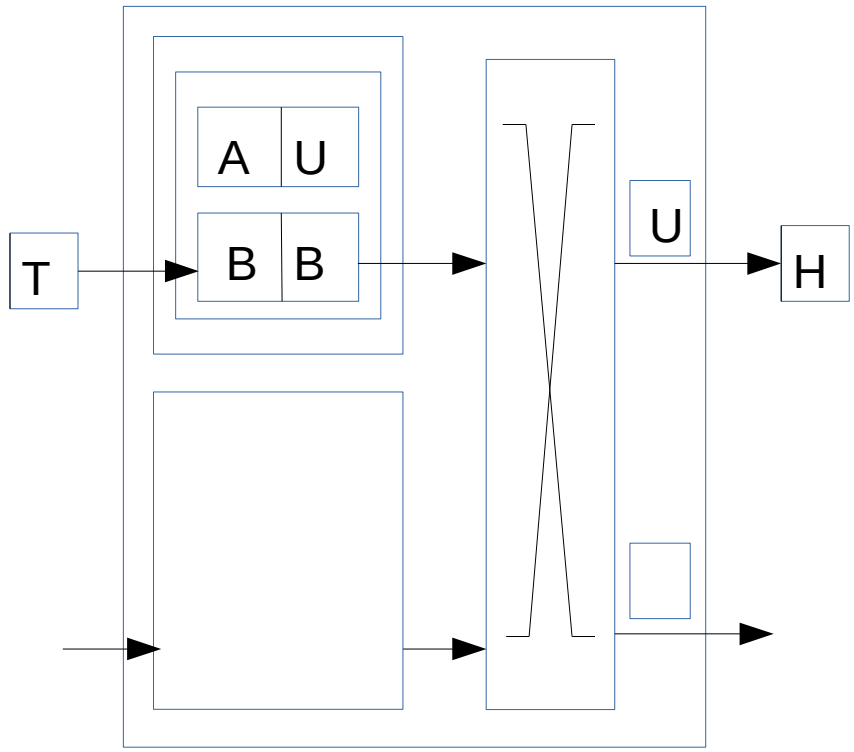
(a) Header arrives. Virtual channel is idle (I). Desired upper output channel is busy-allocated to the lower (L) input.



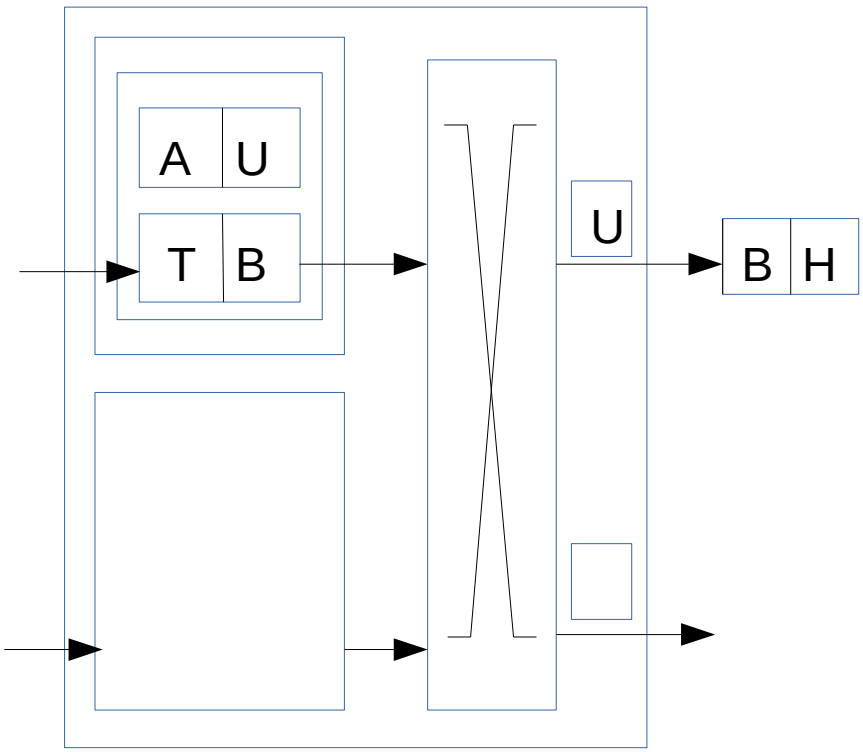
(b) Hedaer is buffered. Virtual channel is in waiting (W) state. First body flit arrives.



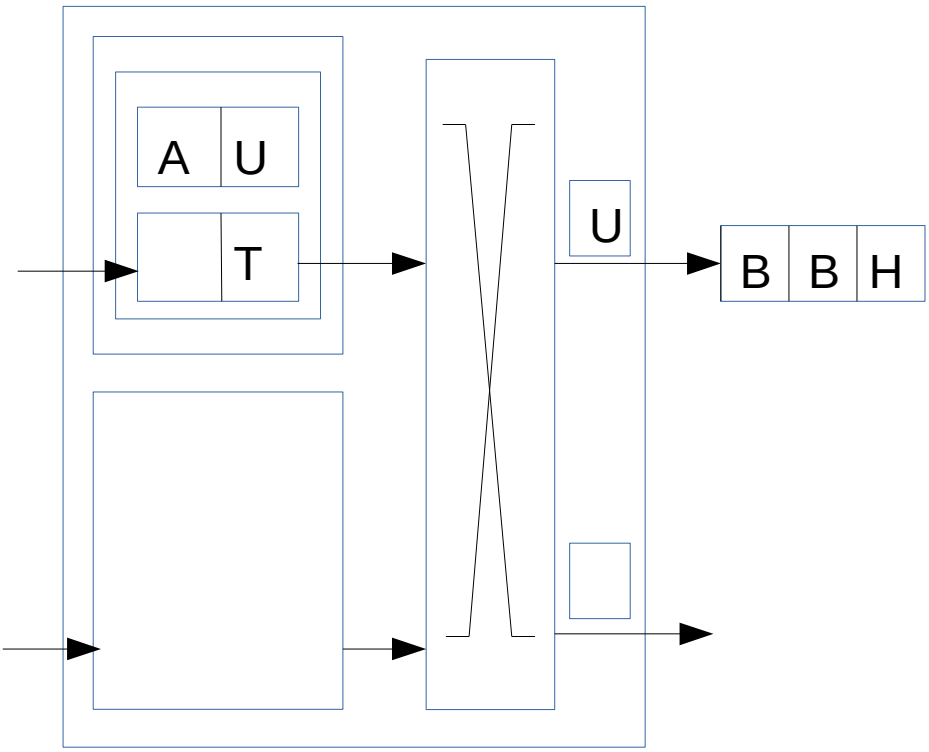
(c) Header and first body flit are buffered. Virtual channel is still in waiting state (persists for two cycles). Input channel is blocked. Second body flit can’t be transmitted since it can’t acquire a flit buffer.



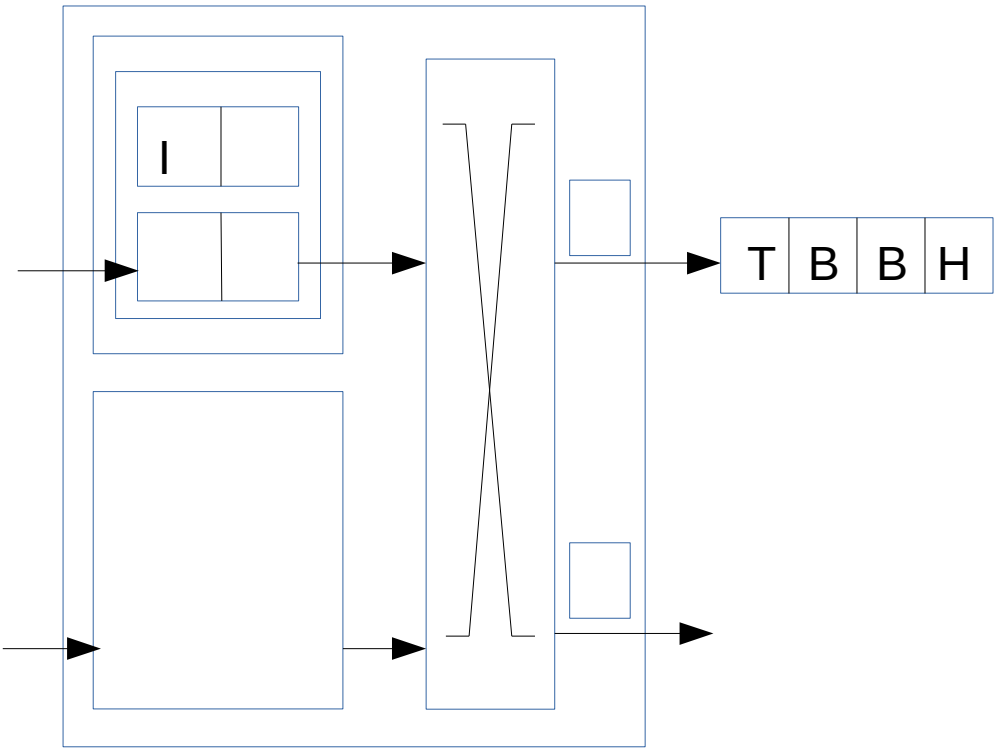
(d) Output virtual channel becomes available and allocated to this packet. The state moves to active (A) and the head is transmitted to the next node.



(e) Body flit moves.

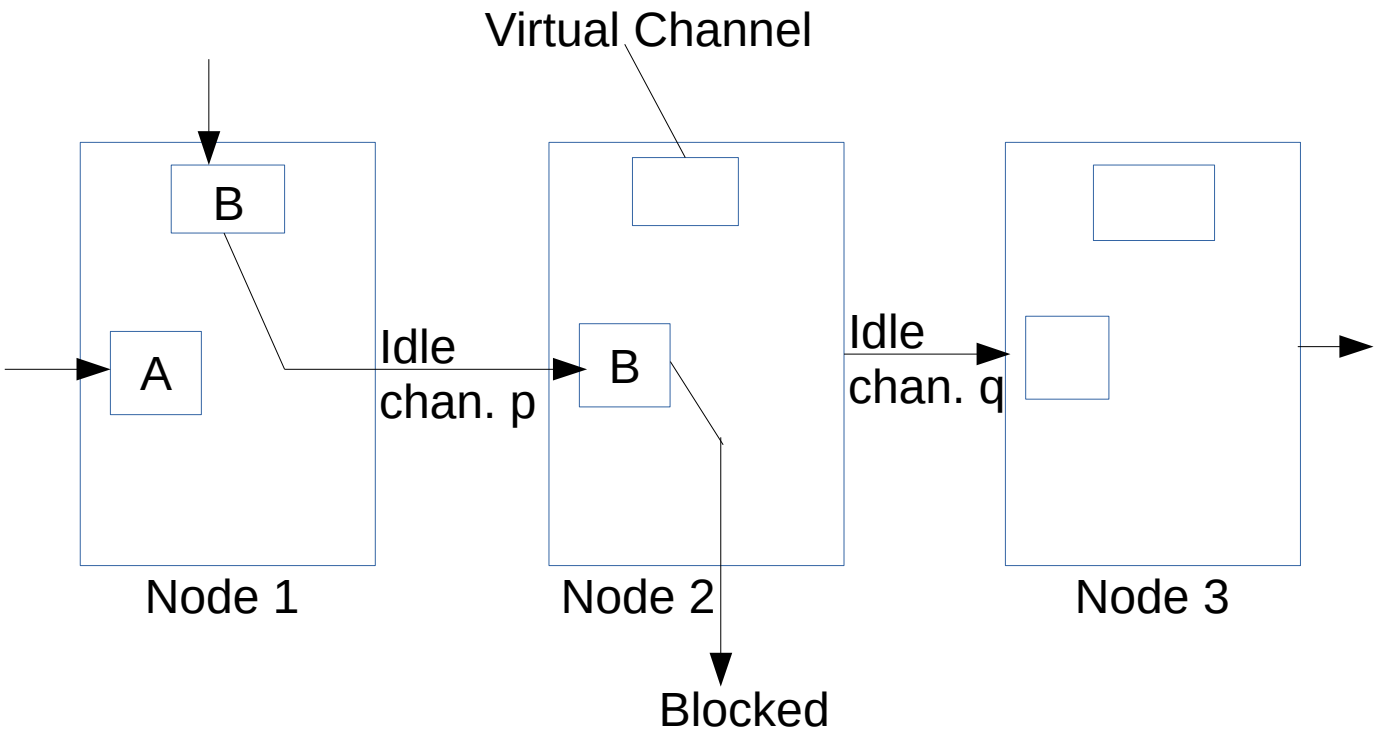


(f) Body flit moves.

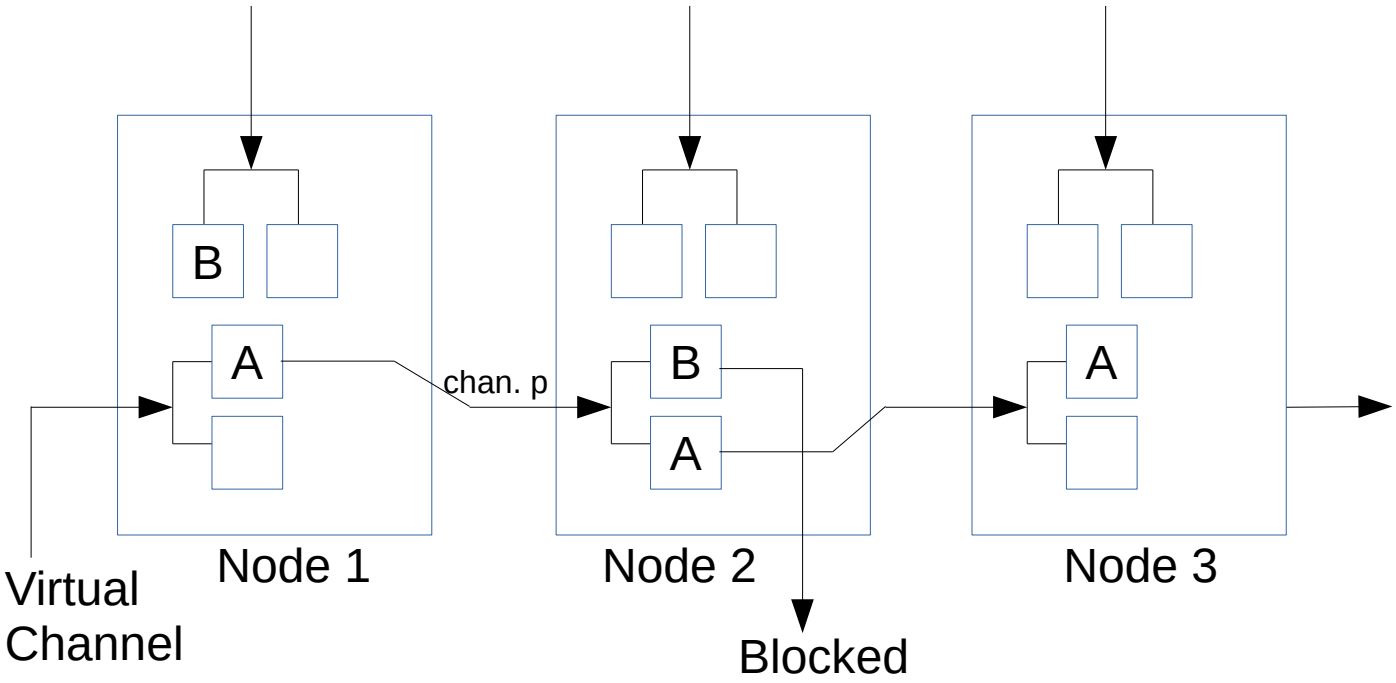


(g) Tail flit is transmitted and frees the virtual channel.

VIRTUAL-CHANNEL FLOW CONTROL



(a) WORMHOLE : When a packet B blocks while holding the sole virtual channel associated with channel p, channel p and q are idled even though packet A requires use of these idle channels.



(b) VIRTUAL-CHANNEL : Packet A is able to proceed over channels p and q using a second virtual channel associated with channel p on node 2.

Virtual-channel flow-control, which associates several virtual channels (channel state and flit buffers) with a single physical channel, overcomes the blocking problems of wormhole flow control by allowing other packets to use the channel bandwidth that would otherwise be left idle when a packet blocks.

Multiprocessors ⟶ **problem of LOW SCALABILITY**

Solutions

- Use a high throughput low-latency interconnection network.
- Use of cache memories.
- The logically shared memory can be physically implemented as a collection of local memories. This new architecture type is called a VIRTUAL SHARED MEMORY or DISTRIBUTED SHARED MEMORY architecture.

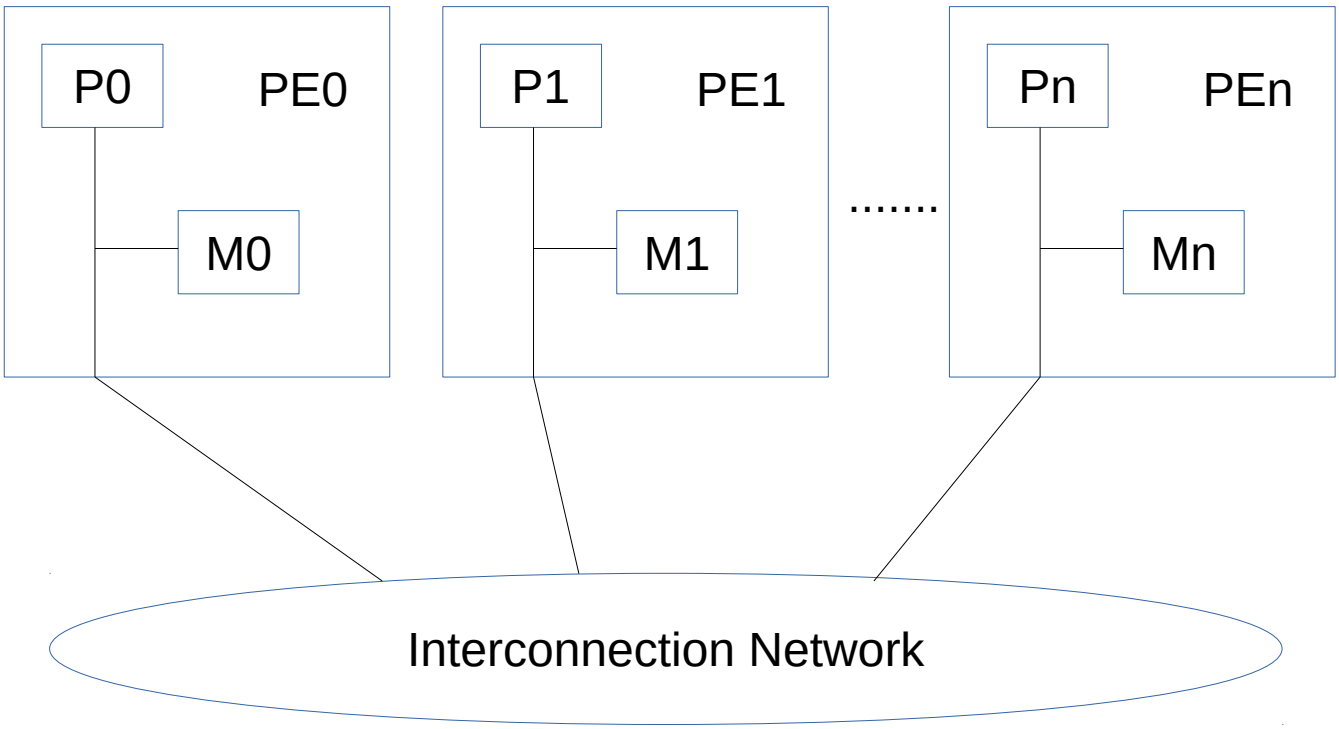
In distributed shared memory systems, the local meories are components of a global address space and any processor can access the local memories of any other.

In distributed memory systems, the local memories have separate address spaces and direct access to the local memory of a remote processor is prohibited.

DISTRIBUTED SHARED MEMORY SYSTEMS : classification

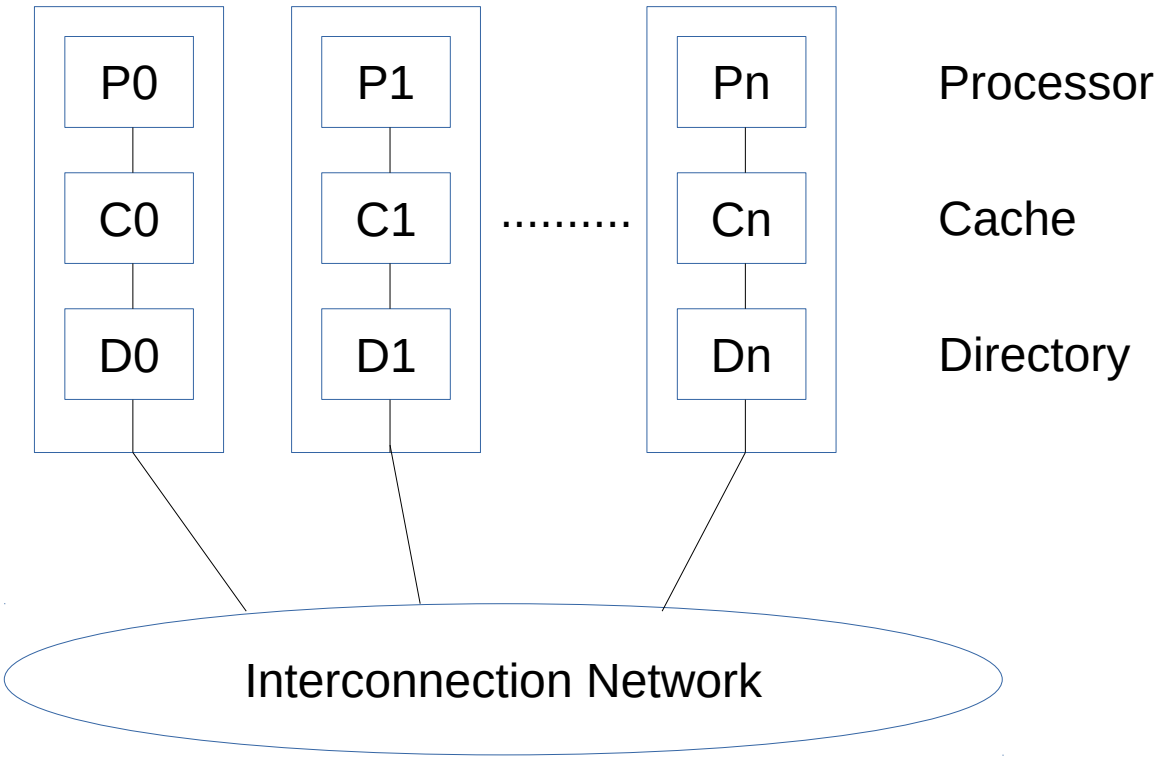
1. Non-Uniform Memory Access (NUMA) machines
2. Cache-Coherent Non-Uniform Memory Access (CC-NUMA) machines
3. Cache-Only Memory Access (COMA) machines

NUMA machines



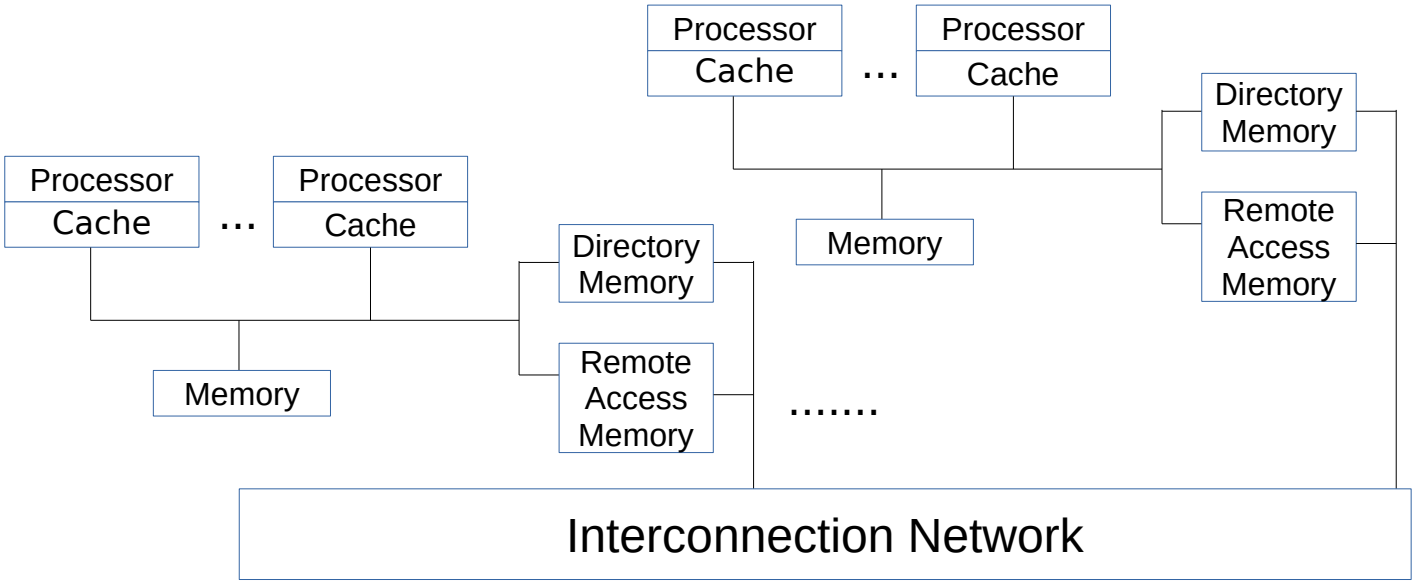
In NUMA machines, the shared memory is divided into as many blocks as there are processors in the system and each memory block is attached to a processor as a local memory with direct bus connection. As a result, whenever a processsor addresses the port of the shared memory that is connected as local memory, access to that block is much faster than access to remote ones.

COMA machines



The COMA model is a special case of a NUMA machine, in which the distributed main memories are converted to caches. There is no memory hierarchy at each processor node. All the caches form a global space. Remote cache access is assisted by the distributed cache directories.

CC-NUMA machines



Stanford Dash [A CC-NUMA architecture]

CC-NUMA machines represent a compromise between the NUMA and COMA machines. Like the NUMA machines, the shared memory is constructed as a set of local memory blocks. However, in order to reduce the traffic on the interconnection network, each processor node is supplied with a large cache memory block.

The dash architecture is a large-scale CC-NUMA system. It consists of multiple microprocessor clusters connected through a scalable, low-latency interconnection network. Physical memory is distributed among the processing nodes in various clusters. The distributed memory forms a global address space.

For each memory block, the directory keeps track of remote nodes caching it. Directory memory and remote access cache facilitate prefetching and the directory-based coherence protocol.