1. **Cache Coherence**

   Two processors can have two different values for the same memory location.

   Memory system is coherent if any read of a data item returns the most recently written value of data item.

   Coherence defines values returned by a read; consistency determines when a written value will be returned by a read.

   *Basic schemes for enforcing coherence are—*

   - Migration and Replication

2. **Cache coherence protocols**

   Protocol to maintain coherence for multiple processors. Key to implementing a cache coherence protocol is tracking the state of any sharing of a data block.

   Two different techniques are—

   - **Spying protocol** — No centralized directory; designed for bus connected system. Two types—

     1. **Write-Invalidate** —

        The processor that is writing data causes copies in the cache of all other processors in the system to be rendered invalid before it changes its local copy.

   2. **Write-Update** (Write-Broadcast) —

        The processor that is writing the data broadcast the new data over the bus. All caches that contained copies of the data are then updated.
Before Write-Through Write-Back

Problem with Snoopy bus protocol:
1. Cannot be used with for a multistage network.
2. System loss is not available for snooping.
3. Snoopy bus protocols at a remote node increase delays there.
4. This increases latency and reduces memory bandwidth.

Directory based Protocols:
Sharing status of a block of physical memory is kept in just one location, called the directory. Applied to network-connected system. Three types:
1. Full-map directories:
   Each directory entry can identify all processors with cached copies of data.

2. Limited directories:
   Each entry has a fixed number of process identifiers, regardless of the system size.
(3) **Chained directories**

Emulate full-map directories by distributing entries among the caches.

```
  Shrew Memory
  X: [C1, C2, ...]
  Cache 
  X: [Data, CT]
  Cache 
  X: [Data, P1]
  Cache 
  X: [Data, P2]
```

> **Limitations of directory-based protocols**

1. **Limited capacity for replication**
2. **Cost of complex design implementation when using hardware control**
3. **Limitations on physical address space to map the information.**

(3) **Message Routing schemes in multicompute networks**

→ **Message Formats**

```
Message
```

```
Packet
```

(Flow control digit) Flit

| D | D | D | D | D | D | S | R |

(4) **Store and Forward Routing**

```
Source node
```

```
Packet Buffer
```

```
Intermediate node
```

```
Destination node
```

Advantages → simple, suitable for interactive traffic, bandwidth demand

Disadvantages → Buffer for every packet, potential long latency, potential deadlock.
(2) Fli& and wormhole routing -

Divides a packet into smaller fixed sized pieces called flits.

Source Node

<table>
<thead>
<tr>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Intermediate node

|        |
|        |
|        |
|        |
|        |
|        |
|        |

Destination Node

Advantages - Good for long messages, required need for buffering, reduced effect of path length.

Disadvantages - Possibility for deadlock, inability to support backtracking.

4] Deadlock and virtual channels -

→ Virtual channels

A principle introduced to allow the design of deadlock-free routing algorithms. It is inexpensive method to increase the number of logical channels without adding more wires.

```
X ——— A ——— B ——— Z
|   |   |   |
|   |   |   |
Y ——— W
```

Virtual channels -

X - A - B - Z
Y - A - B - W

→ Deadlock -

Deadlock can occur if it is impossible for any messages to move (without discarding one). Buffer deadlock occurs when all buffers are full in a tree and forced network.

Channel deadlock occurs if all channels around a circular path in a wormhole-based network are busy.

5] Vector processing principles -

→ Vector instruction types -

1] Vector-Vector instructions - One or two vector operands are fetched from the respective vector or register.
Vector instruction:

- Vector-scalar instruction:
  - Obtain one operand from a vector register and one from a scalar register.

- Vector-memory instruction:
  - Transmit data between memory and a vector register.

- Vector-reduction instructions:
  - Finding maximum, minimum, sum, and mean value of elements in a vector.

- Gather and scatter instructions:
  - Gather:
    - Fetches from memory the non-zero elements of a sparse vector using indices that themselves are indexed.
  - Scatter:
    - Stores into memory a vector in a sparse vectors where non-zero entries are understood.

- Masking instruction:
  - The mask vector is used to compress or to expand a vector to a shorter or longer index vector.

- Vector address memory schemes:
  - To access a vector in a memory, one must specify its base, stride, and length.

Diagram:

S-Access Memory Organization:

- File cycle
- Access cycle
- Module 0
- Module 1
- Module M-1
- High order address bus
- RO/WR

S-Access organization for an M-way interleaved memory.
Vector supercomputer architecture -

- Most supercomputer are clusters of MIMD multiprocessors, each processor of which is SIND.
- A SIND processor executes the same instruction on more than one set of data at the same time.
- MIMD is employed to achieve parallelism, by using a number of processors that function asynchronously and independently.

Features:
- (1) More than one CPU
- (2) Large storage capacity
- (3) Very fast I/O capability
- (4) Beryllium fluids are used for cooling
- (5) Unix/Linux operating system used
- (6) FORTRAN language is preferred

SIND Organization:
- Distributed memory model - Example: Illiac IV
- Interconnection Network
  - Pros: cost effective, way to scale memory bandwidth, reduces latency
  - Cons: Complex, communicating data, Must change software

- Shared memory model - Example: BSI (Burrell's Scientific Instruments)
  - Pros: Global address space, fast data sharing
  - Cons: lack of scalability, responsibility for synchronization, Enforces
3. **Principle of Multithreading**

   **Software Multithreading** — Software that is aware of more than one processor/core and can use these to be able to simultaneously complete multiple tasks.

   **Hardware Multithreading** — Allows multiple to share the functional unit of a single processor in an overlapping fashion.

4. **Multithreading Issues and Solutions**

   → **Problem of Asynchrony**

   Triggered fundamental latency problems on remote loads and synchronization loads

   Solution of remote loads — cost of thread switching should be much smaller than that of the latency of the remote load.

   Solution of synchronization load — distributed caching

   A large continuation space is provided to name an adequate number of threads waiting for remote responses.

5. **Multiple Context Processors**

   Multithreaded systems are constructed with multiple context processors.

   ![Diagram of Multithreaded System](image)

   - Efficiency = \( \frac{\text{busy time}}{\text{total time}} \)