Lecture 5 - Parallel Computer Architectures

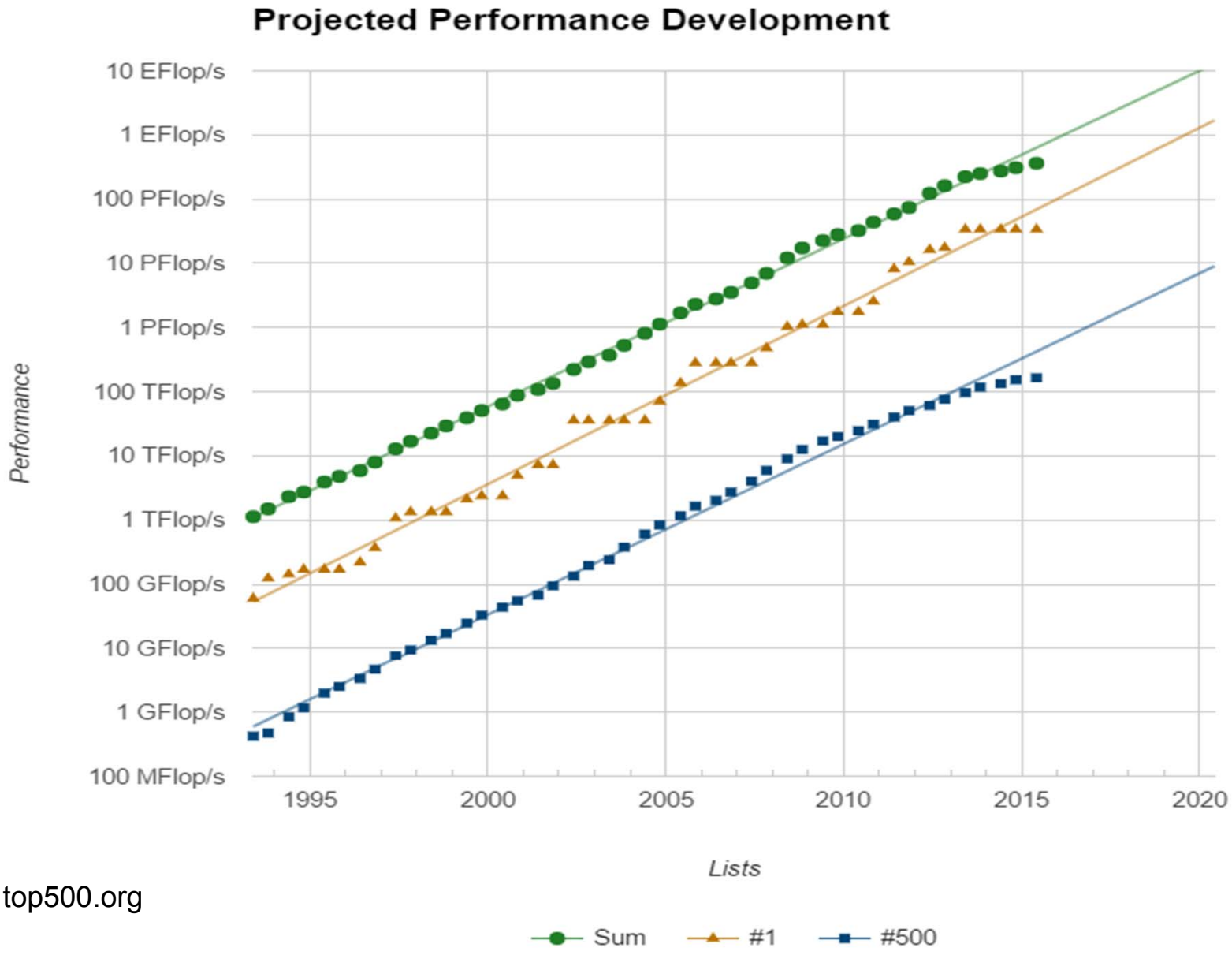
- **Computer program performance relies on**
  - the processor speed and
  - the ability of the memory system to feed data to the processor
- **A memory system takes in a request for a word of memory and returns a bock of data of size "b" containing the requested word after "l" nanoseconds**
  - "l" is referred to as the ***latency*** of the memory (time-delay to fetch data)
  - The rate at which the data can be sent from memory to the processor determines the ***bandwidth*** of the memory system

Parallel Computer Architectures

- **Some Units:**

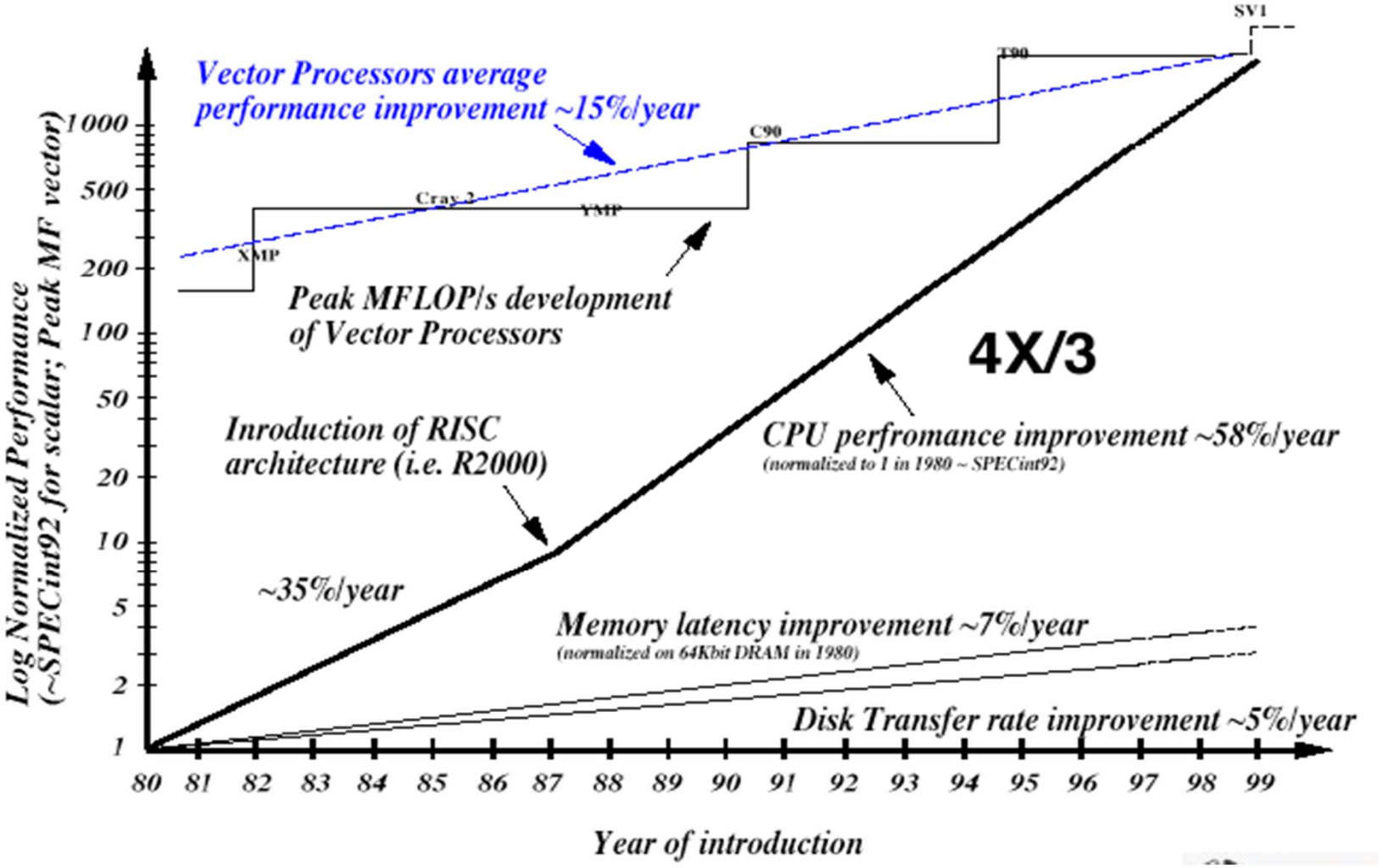
1Mflop= 10^6 flops/sec	**1Mbyte=10^6 bytes**
1Gflop= 10^9 flops/sec	**1Gbyte=10^9 bytes**
1Tflop= 10^{12} flops/sec	**1Tbyte=10^{12} bytes**
1Pflop= 10^{15} flops/sec	**1Pbyte=10^{15} bytes**

Current/Projected Supercomputer Performance

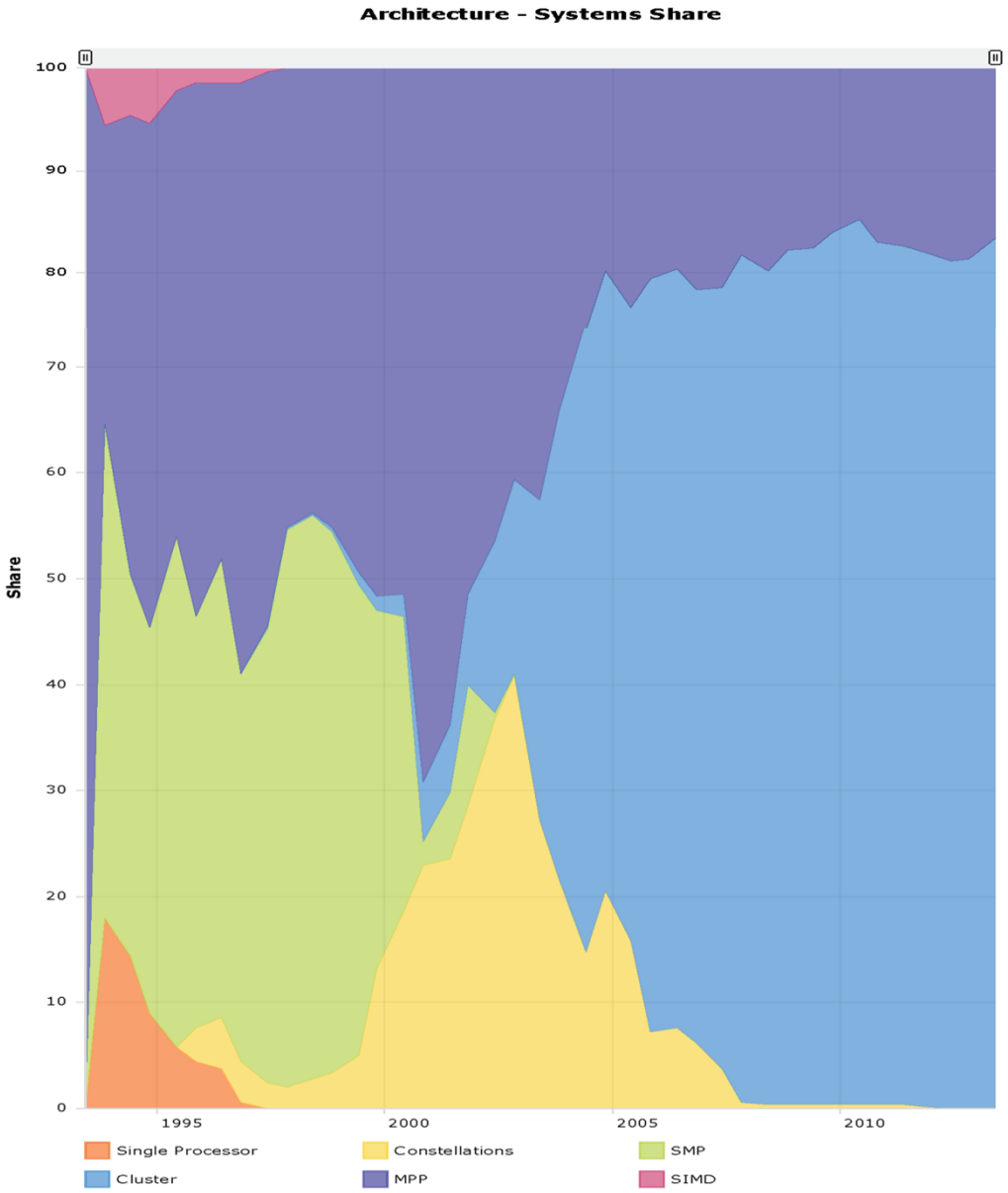


www.top500.org

Past Microprocessor Performance



Computer System



www.top500.org

Parallel Computers

- **A parallel computer is a collection of CPUs that cooperate to solve a problem.**

- **It can solve a problem faster or just solve a bigger problem.**
  - How large is the collection?
  - How is the memory organized?
  - How do they communicate and transfer information?

Categorization of Parallel Architectures

- **Control mechanism:**
  - Instruction (program) stream and data stream
- **Process granularity**
  - Decomposition (low, medium, fine-grain)
- **Address space organization**
- **Interconnection network**
  - Static (routing path of messages between processors is fixed)
  - Dynamic (routing path is flexible depending on congestion)

Control Mechanism (Flynn's Taxonomy)

- **SISD: Single Instruction stream Single Data stream**
  - Traditional single processor computers
- **SIMD: Single Instruction stream Multiple Data stream**
  - Massively parallel computers
- **MIMD: Multiple Instruction stream Multiple Data stream**
  - Most multi-processor computers where processors can work separately or together
- **MISD: Multiple Instruction stream Single Data stream**
  - Never commercially successful

SIMD

- **Multiple processing elements are under the supervision of a control unit**
  - Thinking Machine CM-2, MasPar MP-2, Quadrics, GPUs

- **SIMD extensions are now present in commercial microprocessors (MMX or Katmai in Intel x86, 3DNow in AMD K6 and Athlon, Altivec in Motorola G4)**

- **Example:**

```
DO I = 1,1000
   C(I) = A(I) + B(I)
END DO
```

  - In this example, the various iterations are independent of each other so they can be executed independently. Thus the same instruction (add) can be broadcast with the appropriate data to 1000 CPU's and performed in parallel.

MIMD

- **Each processing element is capable of executing a different program independent of the other processors**

- **Most multiprocessor can be classified in this category)**

MIMD

- **MIMD computers can be "shared-memory" or "distributed-memory"**

- **Example:**

```
DO I = 1,1000
   C(I) = A(I) + B(I)
END DO
```

  - For shared-memory computers, this loop could be internally broken up into n sub-loops, each performed on a different processor. All processors have access to the arrays A, B, and C.
  - For distributed-memory computers, this loop could be broken up by the programmer with the sub-sets of A and B sent to each processor using messages, and then receiving the sum back in another message. The sum of the sub-sets of C are then accumulated.

Process Granularity

- **Coarse grain: Cray C90, Fujitsu, Intel Phi**
  - Decomposition is minimal. Small number of blocks with large amounts of data are processed in parallel.

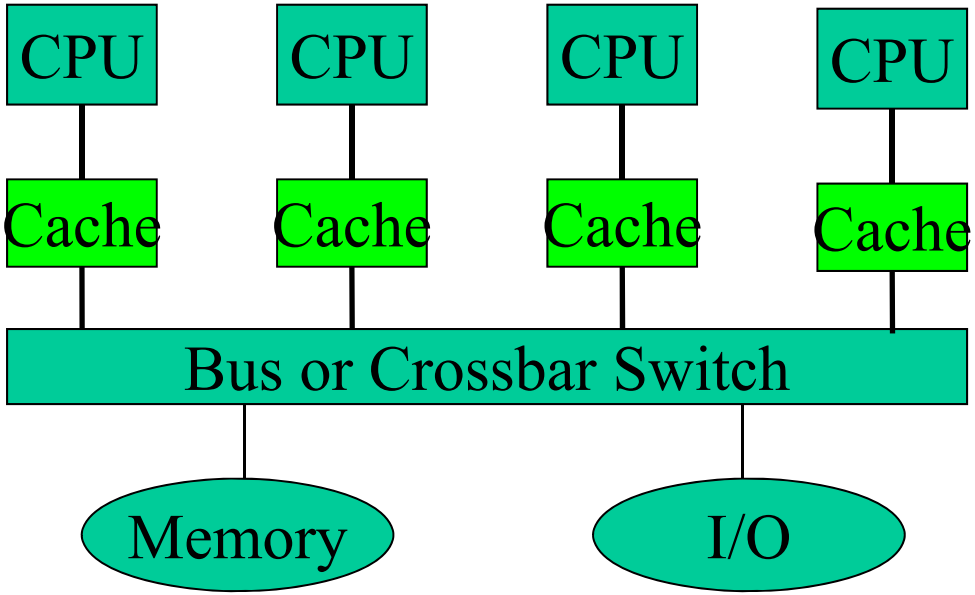
- **Medium grain: IBM SP2/3, CM-5**
  - Decomposition is medium to large. Medium to large number of blocks with medium amounts of data are processed in parallel.

- **Fine grain: CM-2, Quadrics, GPUs**
  - Decomposition is very large (perhaps on a by-point or by-cell basis). Very large number of blocks (or units) with very small amounts of data are processed in parallel.

Types of Computer Platforms

- **Single (shared or common) address space**
  - Uniform Memory Address: SMP (Symmetric Muliti-Processors or Shared-Memory processors, UMA)
    - The time by a processor to access any word in memory (local or global) is identical.
  - Non Uniform Memory Address (NUMA)
    - The time by a processor to access certain words in memory are longer than others. Processors that have high-speed local memory in addition to cache fall into this category.
- **Distributed address space - Message passing**
  - Memory is distributed so that each processor has its own exclusive address space. Processing nodes can consist of single processors or shared-address-space multi-processors.

SMP-UMA Architecture

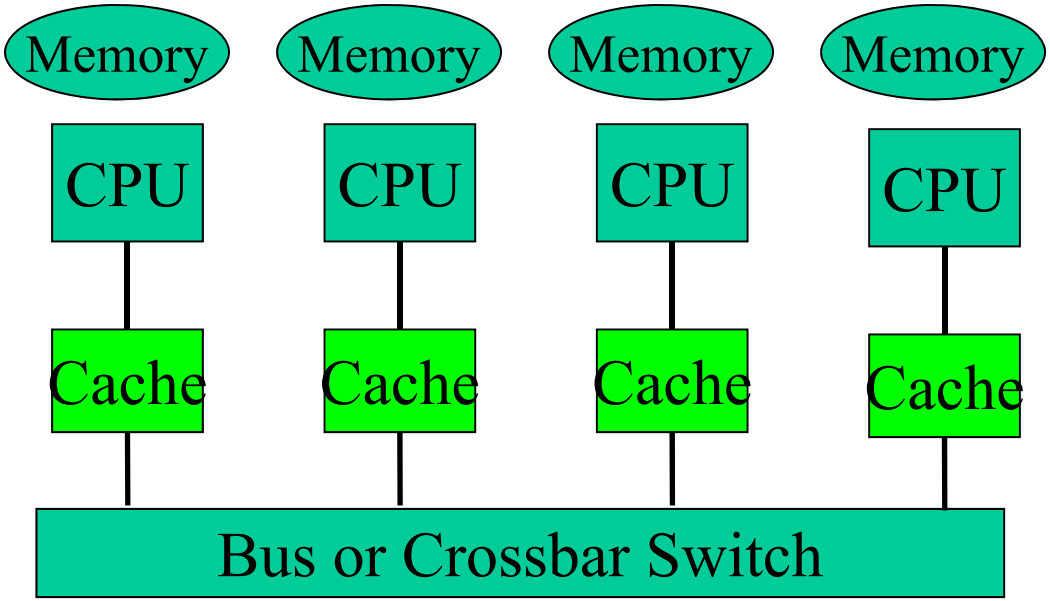


Cache => low-latency high-bandwidth storage may be included

Note that processors do not have local memory

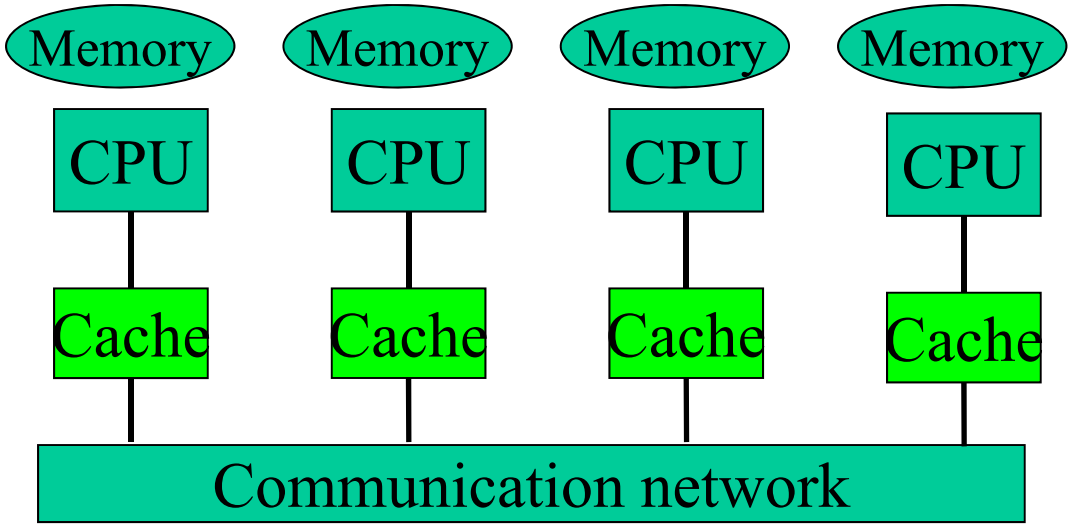
- **SMP uses shared system resources (memory, I/O) that can be accessed equally from all the processors**
- **Cache coherence is maintained.**
  - The presence of cache or local memory leads to multiple copies of a memory word being manipulated by 2 or more processors at the same time.
  - *Invalidate* and *update* protocols can be used to maintain consistency of data across all memory systems

SMP-NUMA Architecture



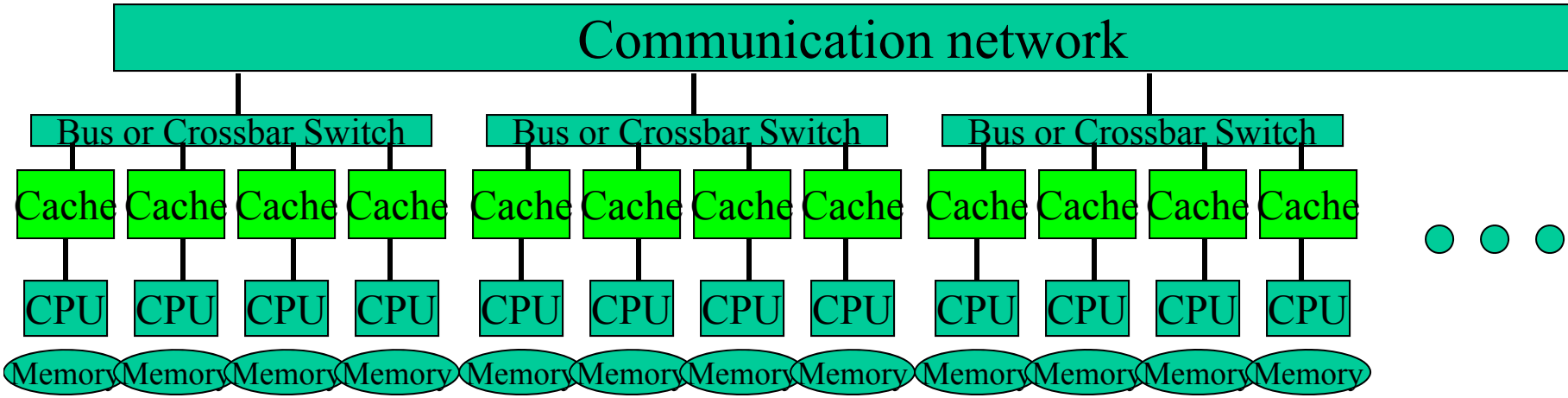
- **Shared address space with non-uniform access. Central memory in addition to local memory.**
- **Memory latency varies whether you access local or remote memory**
- **Cache coherence is maintained using a hardware or software protocol**
- **SGI Origin 2000 and Sun Ultra HPC are examples**

Message-Passing Distributed Memory (MDM)



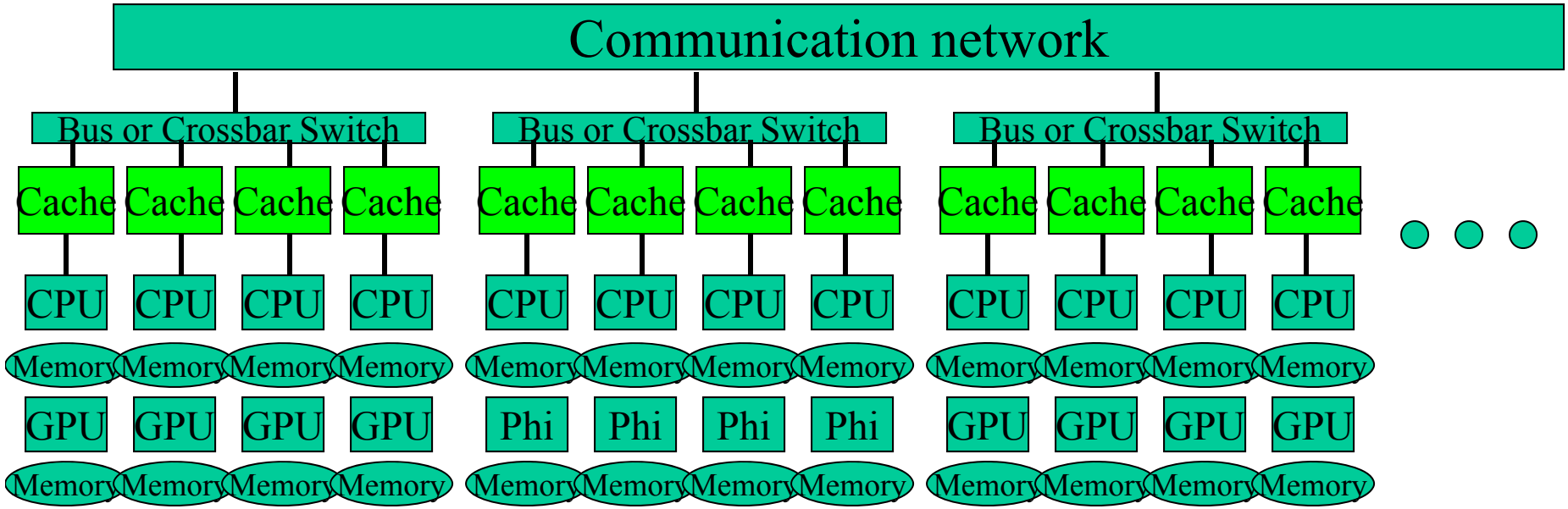
- **Local address space (distributed memory).**
- **No cache coherence. Each processor responsible for its own cache coherence.**
- **Bus or cross-bar switch of SMP is replaced with a communication network (switch or Ethernet)**
- **Beowulf cluster is an example where each "CPU" might represent multiple "cores"**

Distributed Shared Memory (DSM)



- **Shared address space (shared memory at the individual processor level).**
- **Local address space (distributed memory at the system level)**
- **Cache coherence at the individual processor level is maintained using a hardware or software protocol**
- **No cache coherence at system level. Each processor responsible for its own cache coherence.**
- **Multi-node Beowulf cluster is an example where each node contains multiple cores (CPU)**

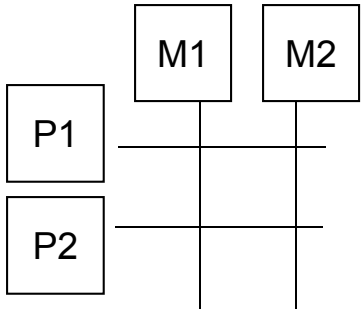
Multi-Processor Distributed Shared Memory (MPDSM)



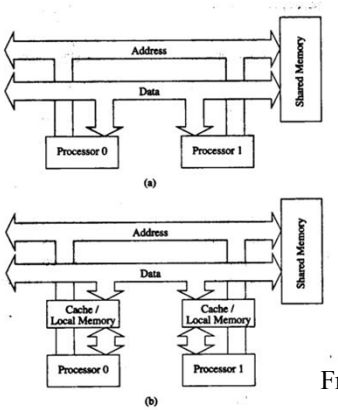
- **Shared address space (shared memory at the individual processor level).**
- **Local address space (distributed memory at the system level)**
- **Cache coherence at the individual processor level is maintained using a hardware or software protocol**
- **No cache coherence at system level. Each processor responsible for its own cache coherence.**
- **Multi-core CPU/GPU or CPU/Phi Beowulf cluster or cluster of NVIDIA Tesla or Intel Phi computers is an example**

Dynamic Interconnections

- **Crossbar Switching :**
  - Most expensive and extensive interconnection.

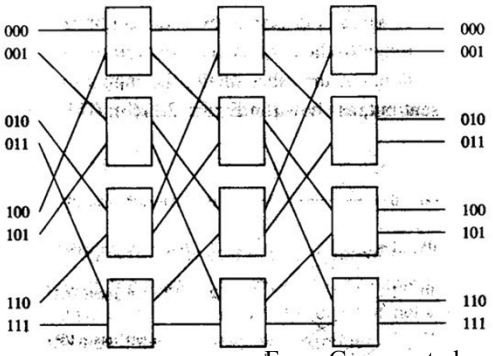


- **Bus connected :**
  - Processors are connected to memory through a common datapath

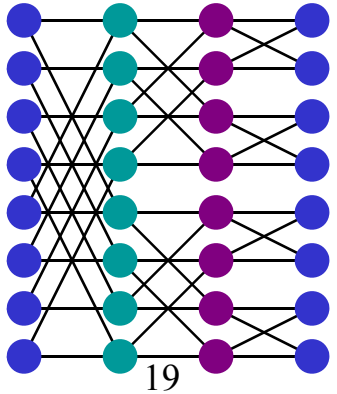


From Grama, et al

- **Multistage interconnection:**
  - Butterfly,Omega network, perfect shuffle, etc



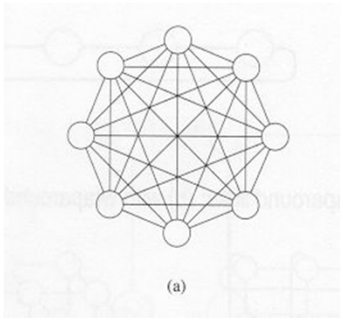
From Grama, et al

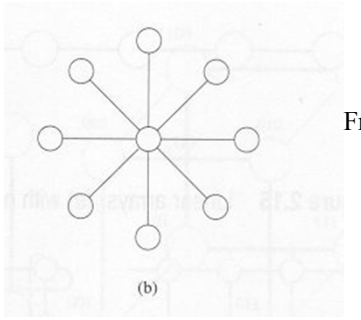


Butterfly

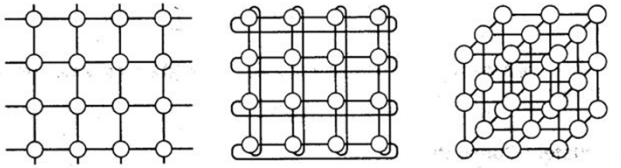
Static Interconnection Network

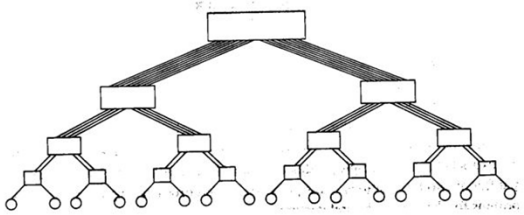
- **Complete interconnection**
- **Star interconnection**
- **Linear array**
- **Mesh: 2D/3D mesh, 2D/3D torus**
- **Tree and fat tree network**
- **Hypercube network**



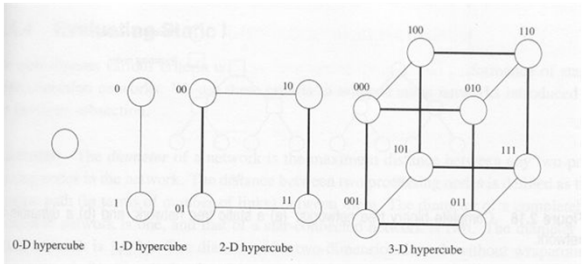


From Grama, et al





From Grama, et al



From Grama, et al

Characteristic of Static Network

- **Diameter: maximum distance between any two processors in the network**

D=1	complete connection
D=N-1	linear array
D=N/2	ring
D=2(√N -1)	2D mesh
D=2 (√(N/2))	2D torus
D=log N	hypercube

N is the number of processors in network

- **Connectivity: measure of the multiplicity of paths between any two processors in the network**
  - High connectivity is desirable because it lowers contention for communication resources

Characteristic of Static Network

- **Bisection width: minimum number of communications links that have to be removed to partition the network in half.**
- **Channel rate: peak rate at which a single wire can deliver bits**
- **Channel bandwidth: product of channel rate and channel width**
- **Bisection bandwidth: product of bisection width and channel bandwidth.**

Network Characteristics

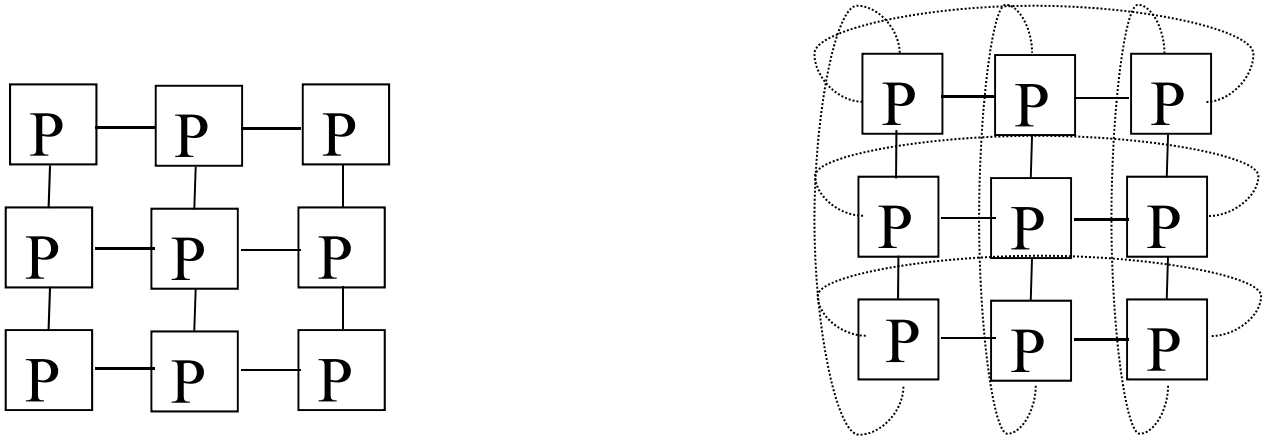
- **Some characteristics of each network:**

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Completely-connected	1	$p^2/4$	$p-1$	$p(p-1)/2$
Star	2	1	1	$p-1$
Complete binary tree	$2\log((p+1)/2)$	1	1	$p-1$
Linear array	$p-1$	1	1	$p-1$
2-D mesh, no wraparound	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2(p-\sqrt{p})$
2-D wraparound mesh	$2\lfloor\sqrt{p}/2\rfloor$	$2\sqrt{p}$	4	$2p$
Hypercube	$\log p$	$p/2$	$\log p$	$(p\log p)/2$
Wraparound k-ary d-cube	$d\lfloor k/2\rfloor$	$2k^{d-1}$	$2d$	dp

From Grama, et al

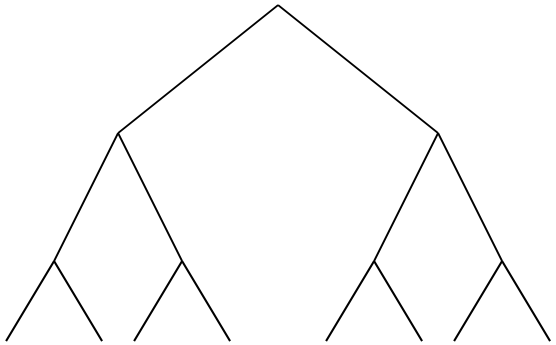
Linear Array, Ring, Mesh, Torus

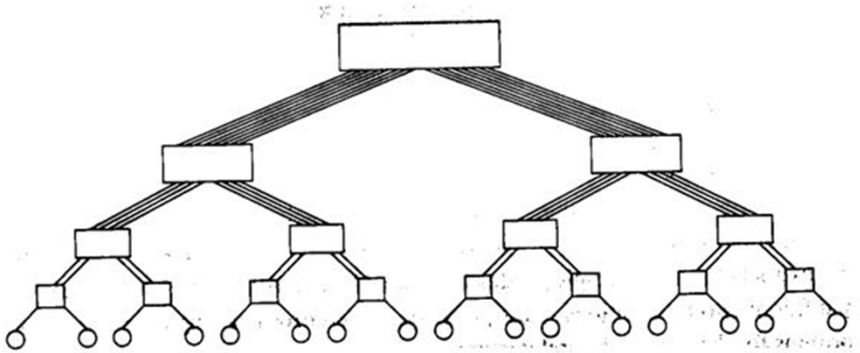
Processors are arranged as a d-dimensional grid or torus



Tree, Fat-Tree

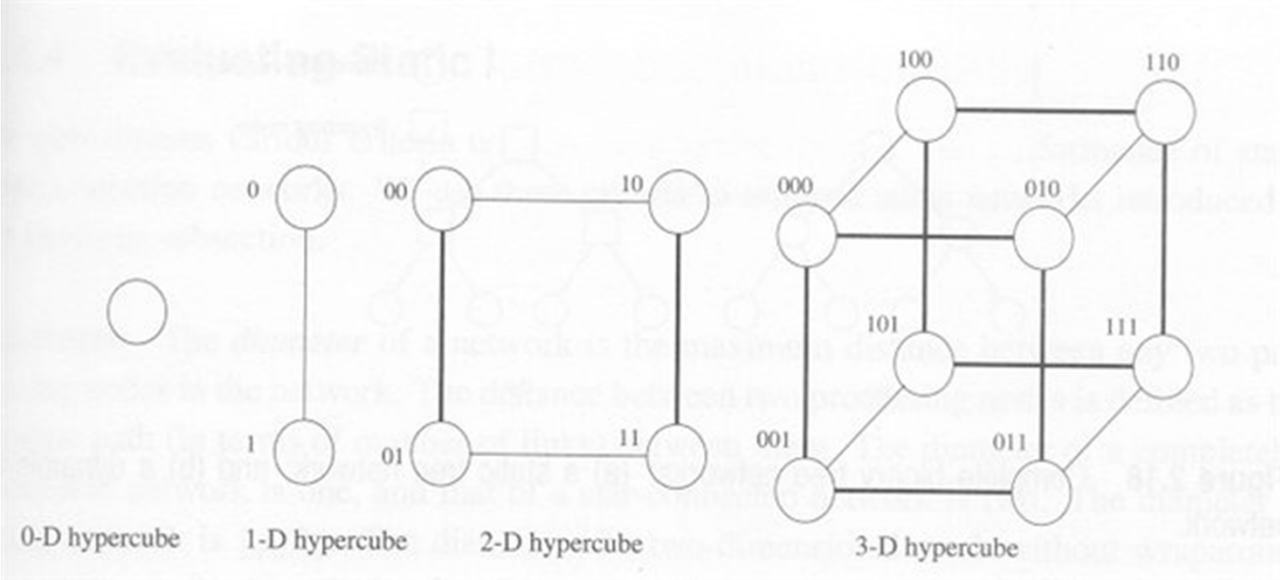
- **Tree network: there is only one path between any pair of processors.**
- **Fat tree network: increase the number of communication links close to the root**



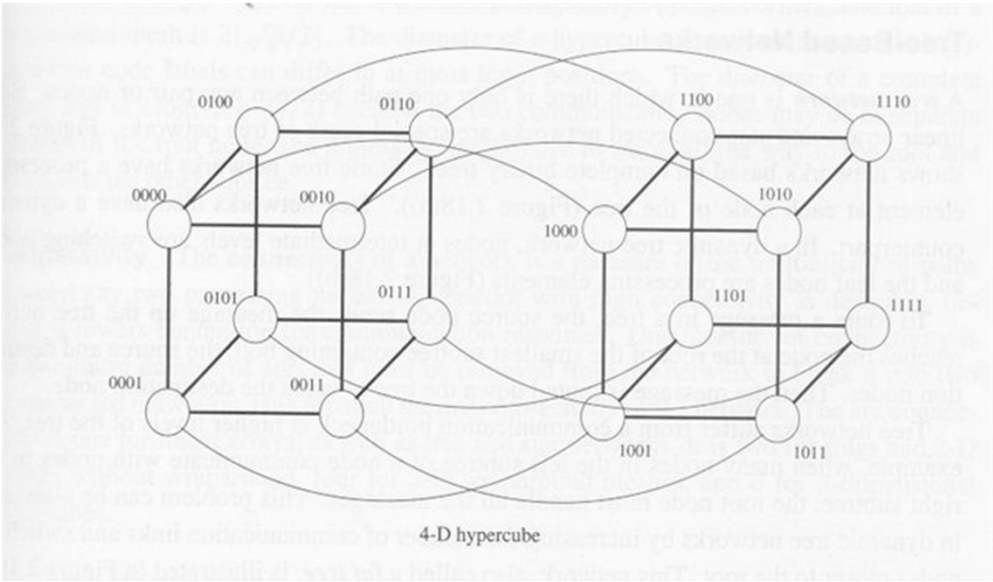


From Grama, et al

Hypercubes



From Grama, et al



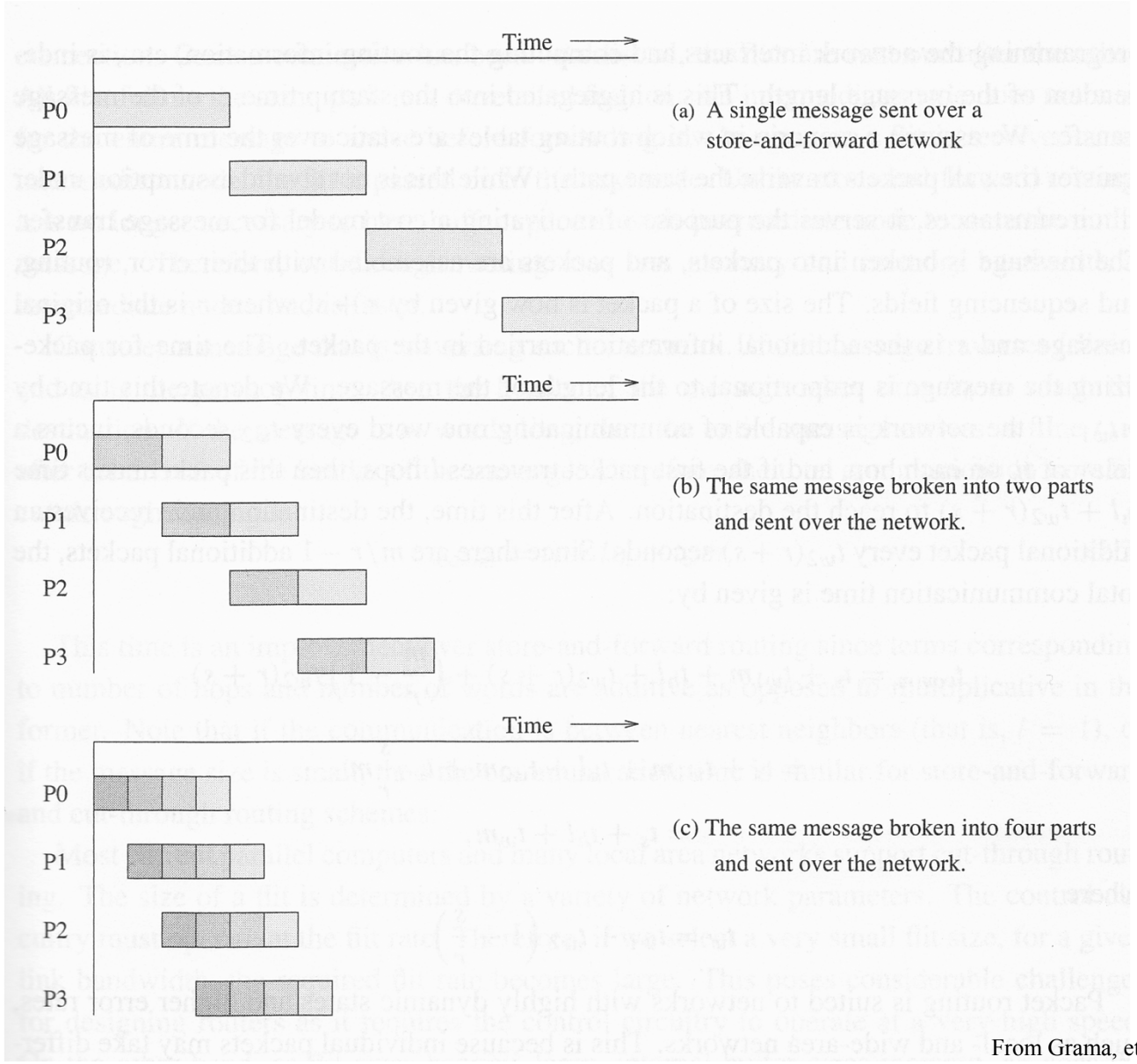
Communication Costs

- **Communication of information between processing elements is one of the major overheads related to parallel computing on distributed-memory systems**
- **Cost factors include:**
  - Startup time, t_s, is the time required to handle a message at the sending and receiving nodes
  - Per-hop time, t_h, is the time it takes the header of a message to travel between two directly-connected nodes. This is also known as node latency
  - Per-word transfer time, t_w, is the average amount of time that a message of size m takes to traverse l links (or hops)

$$time_{communication} = t_s + (mt_w + t_h)l \quad \text{for store - and - forward routing}$$
$$= t_s + lt_h + mt_w \quad \text{for cut - through routing}$$

t_h is generally considered to be small compared to t_s and t_w

Routing Strategies



(a) A single message sent over a store-and-forward network

(b) The same message broken into two parts and sent over the network.

(c) The same message broken into four parts and sent over the network.

From Grama, et al

Communication Costs

- **So in order to optimize the cost of message transfers, we need to**
  - Communicate in bulk: aggregate a number of small messages into a single large message to reduce the effect of t_s
  - Minimize the volume of data: reduce the amount of data that is being passed
  - Minimize the distance of data transfer: minimize the number of hops, l, that a message must traverse
- **The first and second of these involves programming strategy and techniques**
- **The third involves the inter-connection of the processing elements**

Routing

- **Efficient algorithms to route messages to processors are critical to achieve good parallel performance**
- **Routing mechanisms:**
  - Minimal: always select the shortest path. Provides the minimum t_h but can lead to congestion
  - Non-minimal: can route messages along longer (than the shortest) paths to avoid congestion
  - Deterministic: a unique path based upon the source and destination
  - Adaptive: a path based upon the current status of the network and selects a path that avoids congestion

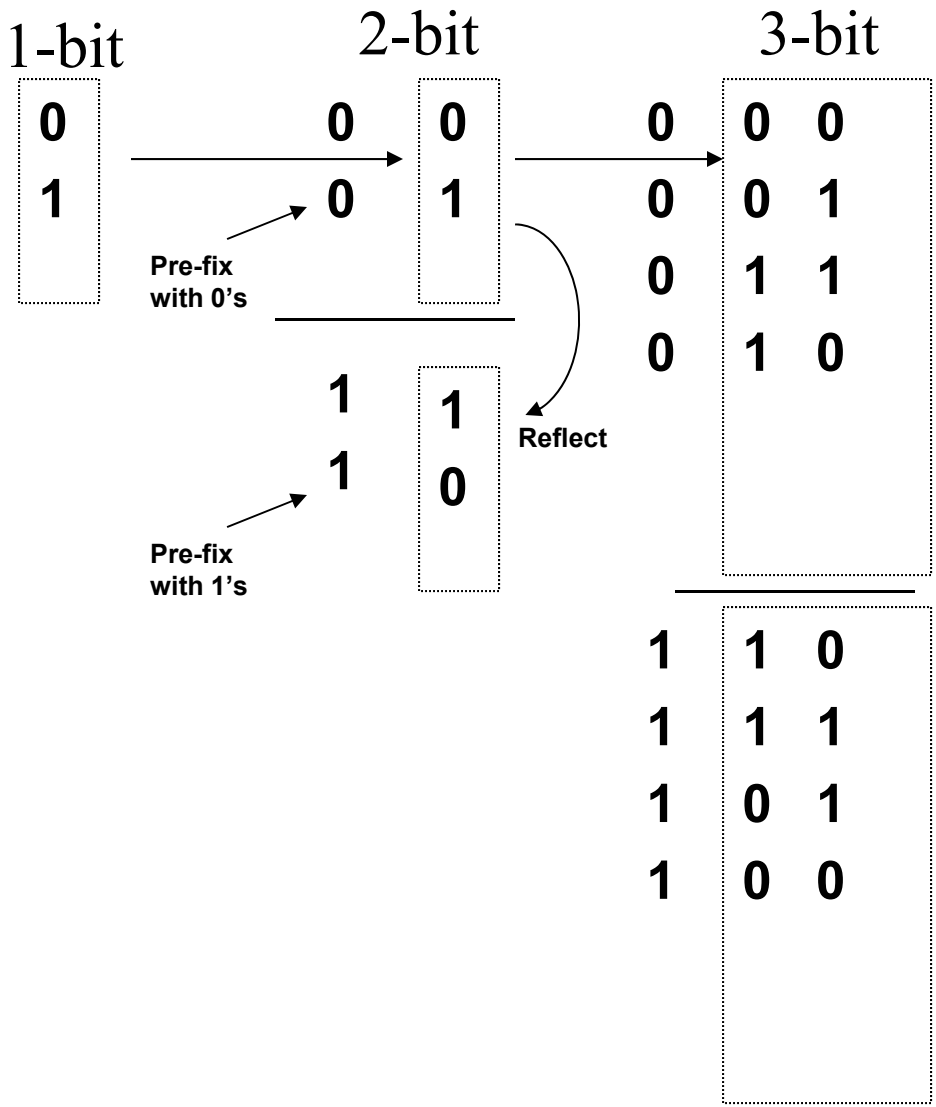
Mapping Techniques to Determine Inter-Connections

- **Mapping techniques are used to determine optimal processor inter-connections and predict the efficiency of networks**
- **Binary Reflected Gray Code (BRG): G(i,d) denotes the i-th entry in a sequence of Gray codes of d bits. G(i,d+1) is derived from G(i,d) by reflecting the table and prefixing the reflected entry with 1 and the original entry with 0.**

$$G(0,1) = 0$$
$$G(1,1) = 1$$
$$G(i,x+1) = \begin{cases} G(i,x), & i < 2^x \\ 2^x + G\left(2^{x+1} - 1 - i, x\right), & i \geq 2^x \end{cases}$$

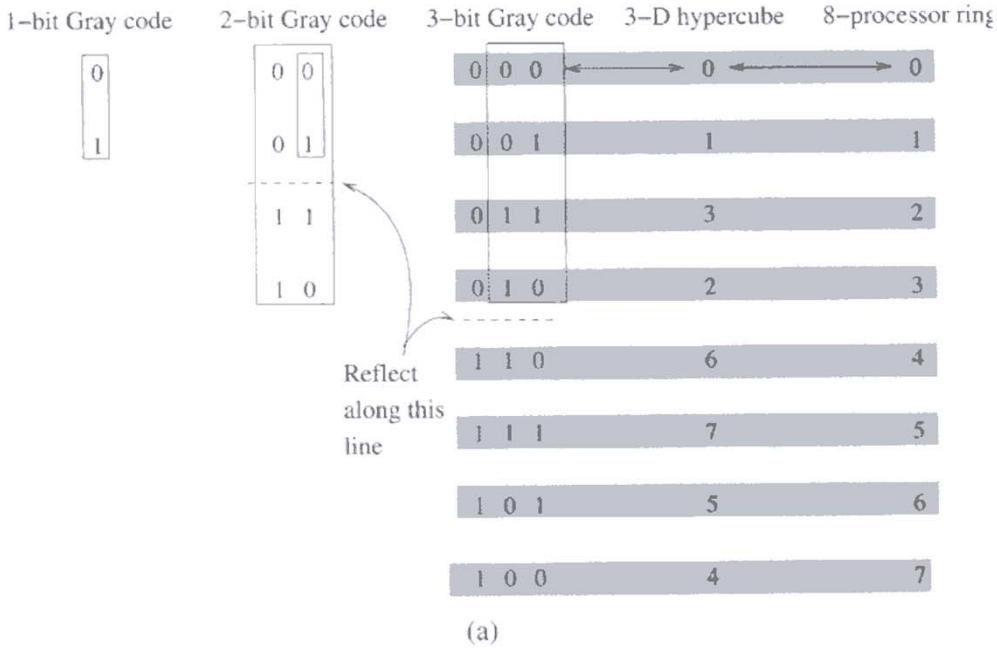
A linear array composed of 2^d nodes can be embedded into a d-dimensional hypercube using this mapping

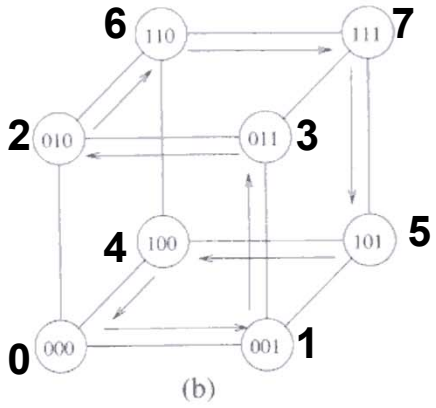
Example of BRG Code: 8p Ring → 8p hypercube



8p ring	8p hyper
0	0
1	1
2	3
3	2
4	6
5	7
6	5
7	4

8p ring → 8p hypercube





From Grama, et al

Embedding Other Networks on Hypercubes:

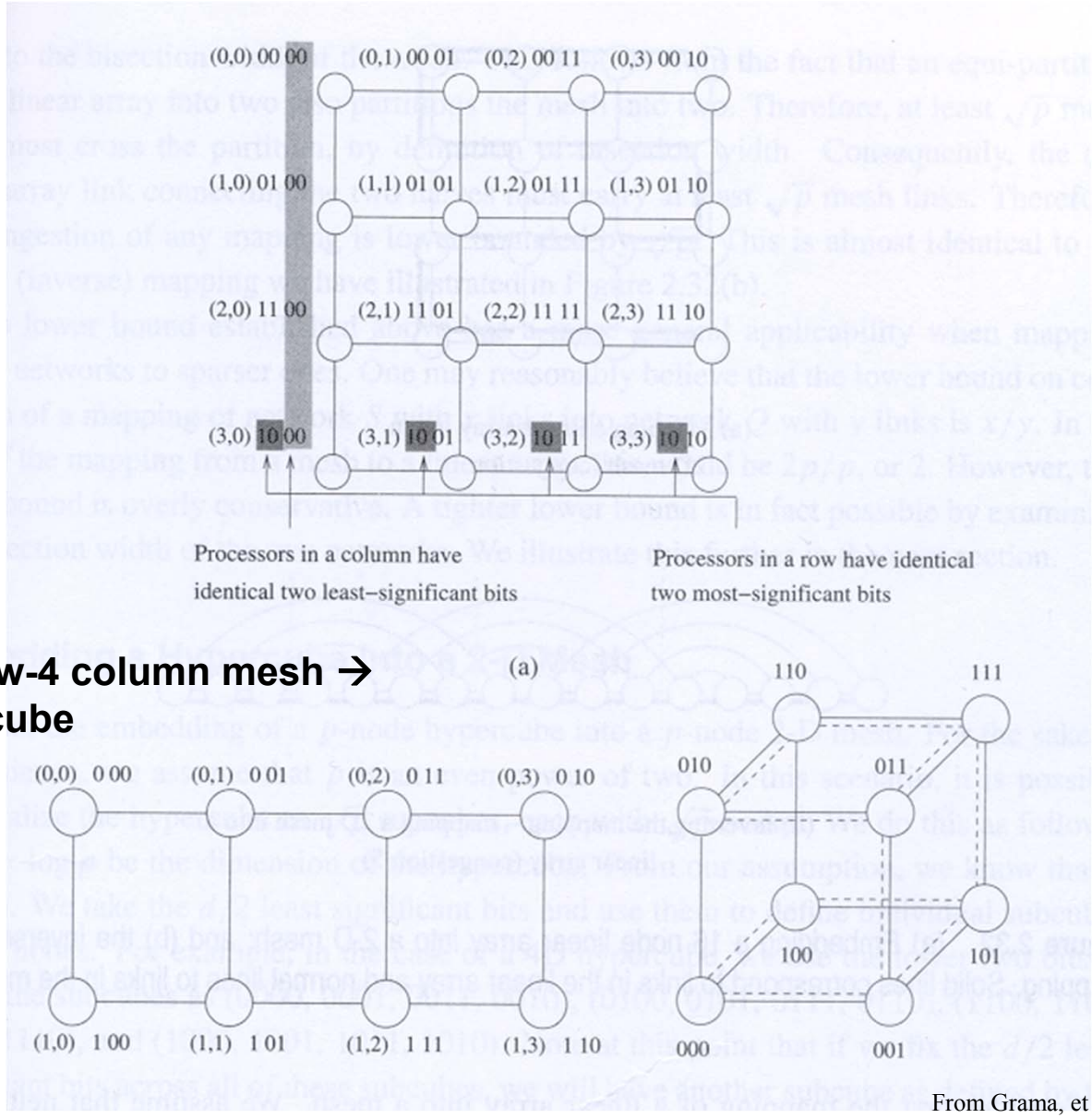
- **Hypercube is a rich topology, many other networks can be "easily" mapped onto it.**

- **Mapping a linear array into a hypercube:**
  - A linear array (or ring) of 2^d processors can be embedded into a d-dimensional hypercube by mapping processor i onto processor G(i,d) of the hypercube

- **Mapping a 2^r x 2^s mesh on a hypercube:**

  processor(i,j)---> G(i,r)||G(j,s) (|| denote concatenation)

Mapping Meshes → Hypercubes



Example: 2 row-4 column mesh → 8-node hypercube

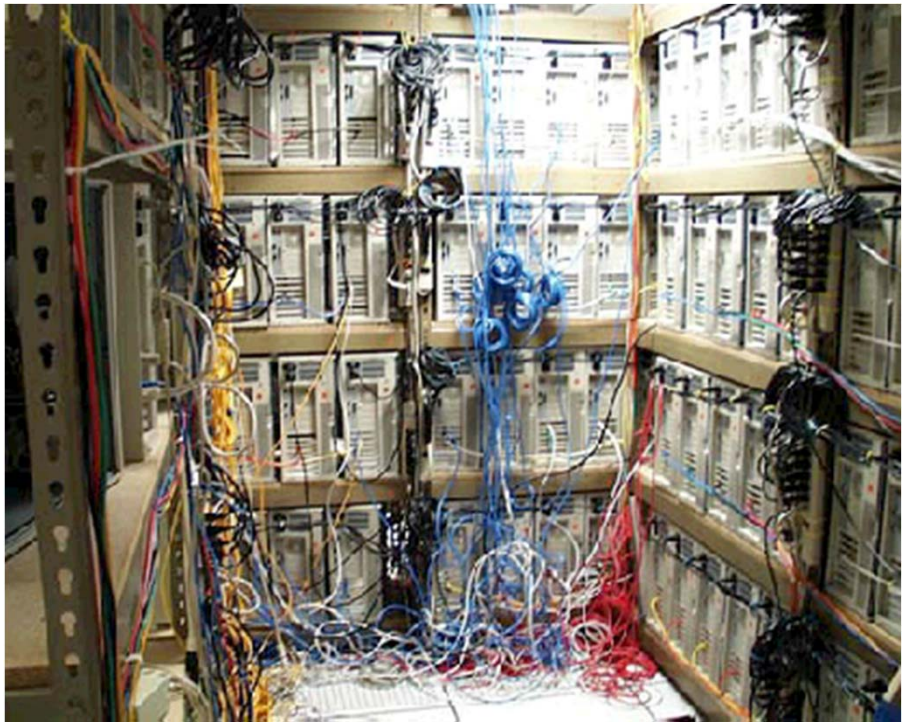
From Grama, et al

Trade-Off Among Different Networks

Network	Minimum latency	Maximum Bw per Proc	Wires	Switches	Example
Completely connected	Constant	Constant	O(p*p)	-	-
Crossbar	Constant	Constant	O(p)	O(p*p)	Cray
Bus	Constant	O(1/p)	O(p)	O(p)	SGI Challenge
Mesh	O(sqrt p)	Constant	O(p)	-	Intel ASCI Red
Hypercube	O(log p)	Constant	O(p log p)	-	Sgi Origin
Switched	O(log p)	Constant	O(p log p)	O(p log p)	IBM SP-2

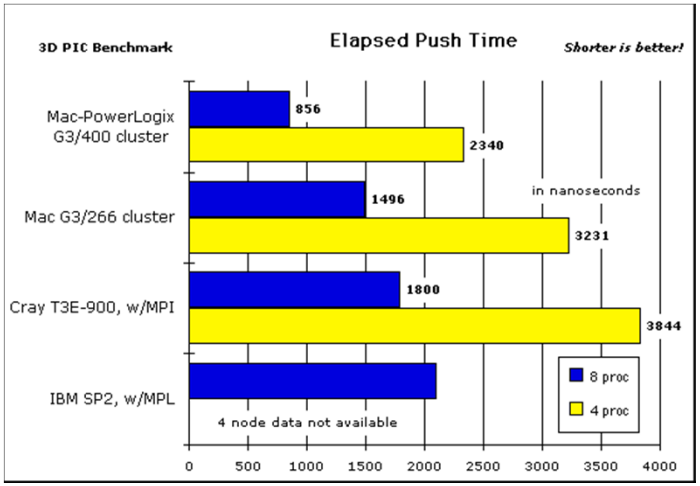
Beowulf

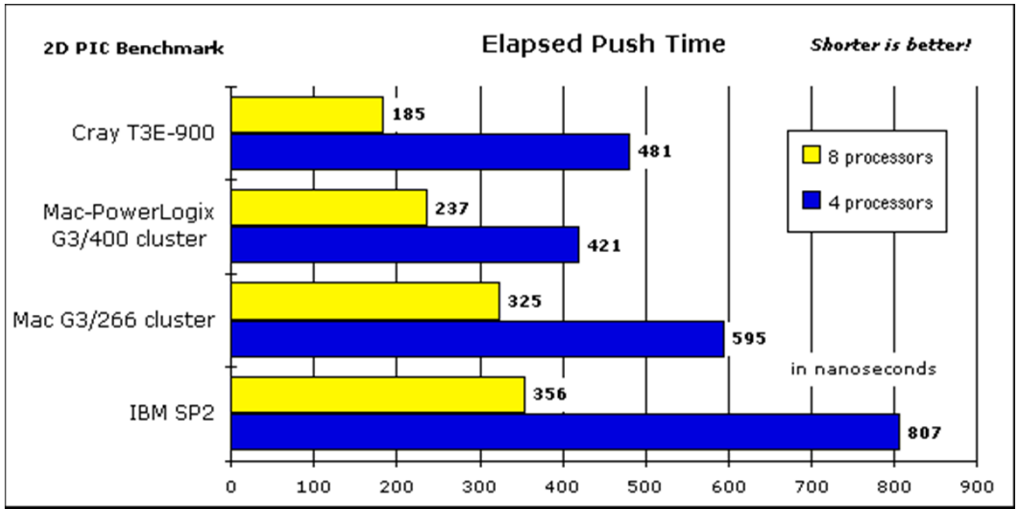
- **Cluster built with commodity hardware components**
  - PC hardware (x86,Alpha,PowerPC)
  - Commercial high-speed interconnection (100Base-T, Gigabit Ethernet, Myrinet,SCI)
  - Linux, Free-BSD operating system



http://www.beowulf.org

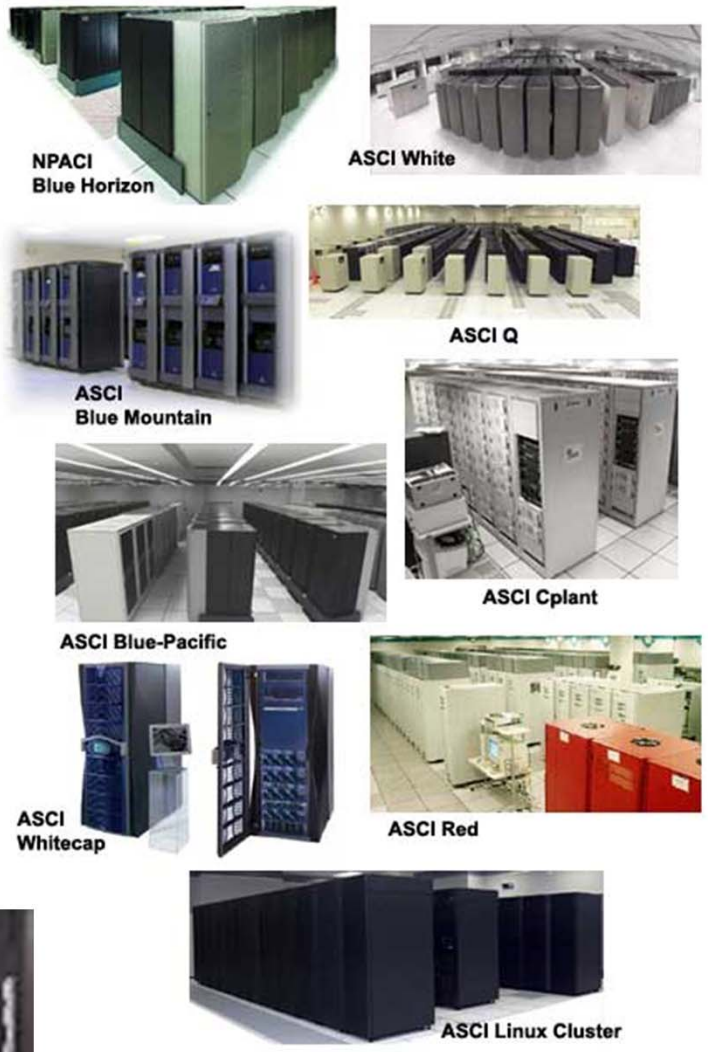
Apple: PowerPC cluster





Clusters of SMP

- **The next generation of supercomputers will have thousand of SMP nodes connected.**
  - Increase the computational power of the single node
  - Keep the number of nodes “low”
  - New programming approach needed, MPI+Threads (OpenMp,Pthreads,….)
  - See www.top500.org




Jaguar


Sequoia

Tianhe-2 (1st)

- China's National University of Defense Technology
- 54.9 Petaflops at peak
- 1,024 Terabytes of memory
- 3,120,000 Xeon cores
- Sustained performance of up to 33.8 Petaflops

- http://en.wikipedia.org/wiki/Tianhe-2

Titan (2nd)

- DoE Oak Ridge National Lab
- 27.1 Petaflops at peak
- 560,640 core processors (70,080 8-core nodes) Opteron with NVIDIA K20x
- 710 Terabytes of memory
- Sustained performance of up to 17.6 Petaflops

- http://en.wikipedia.org/wiki/Titan_(supercomputer)

Sequoia – BlueGene/Q (3rd)

- DoE LLNL
- 20.1 Petaflops at peak
- 1,573 Terabytes of memory
- 1,572,864 cores IBM Power BQC
- Sustained performance of up to 17.2 Petaflops

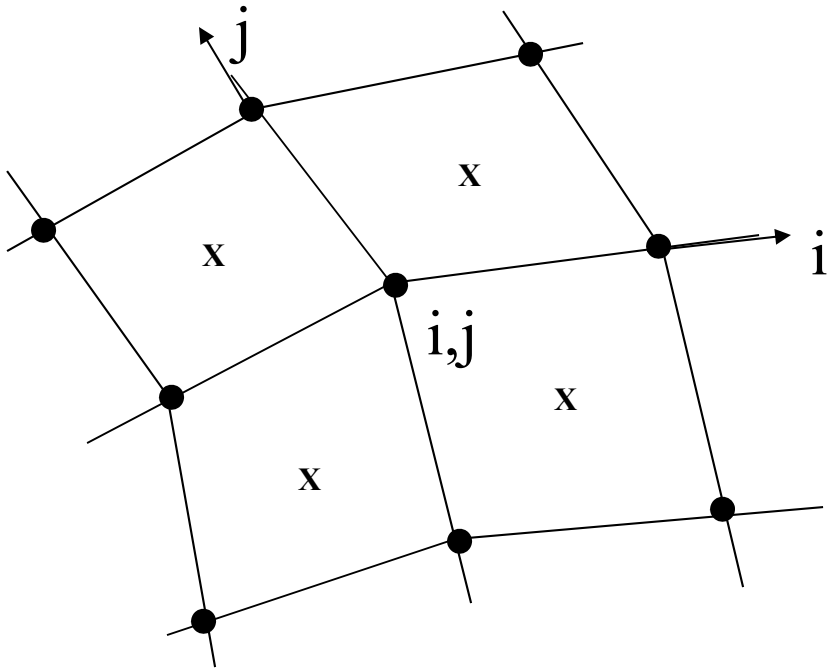
- http://en.wikipedia.org/wiki/IBM_Sequoia

Finite Volume Numerical Techniques

- There are several tricks used in finite-volume schemes that make the creation of first and second difference and averaging operators more straightforward
- These finite-volume operations stay within the confines of a given block and do not require obtaining information outside of the block
- All of the operators “accumulate” cell contributions at the nodes

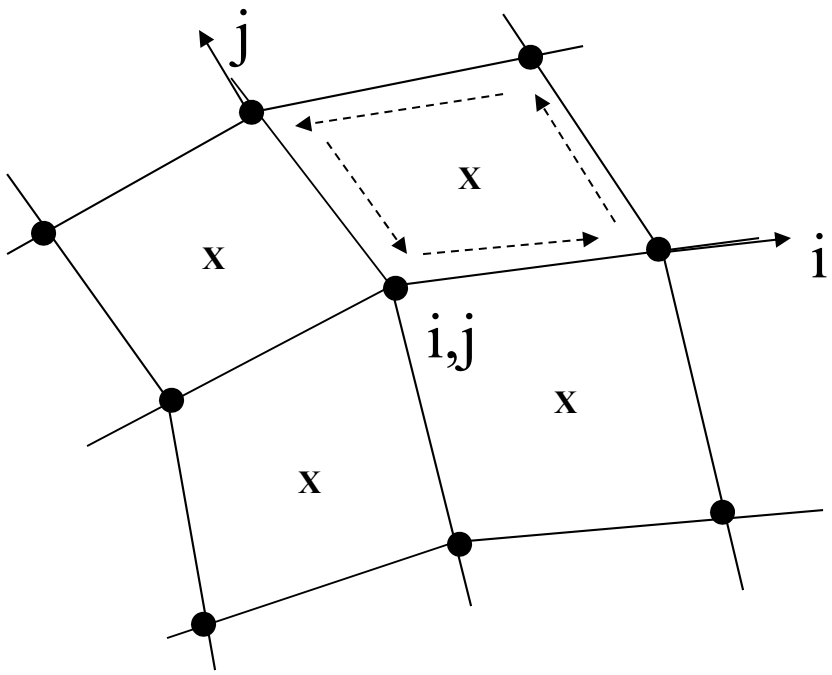
First-Derivative Operators

- **Finite-difference derivative operators have influence from the 5 nodes along the i- and j-directions (corner node influence is omitted)**
- **Finite-volume derivative operators will have influence from all 9 nodes surrounding any given node (in 2D)**



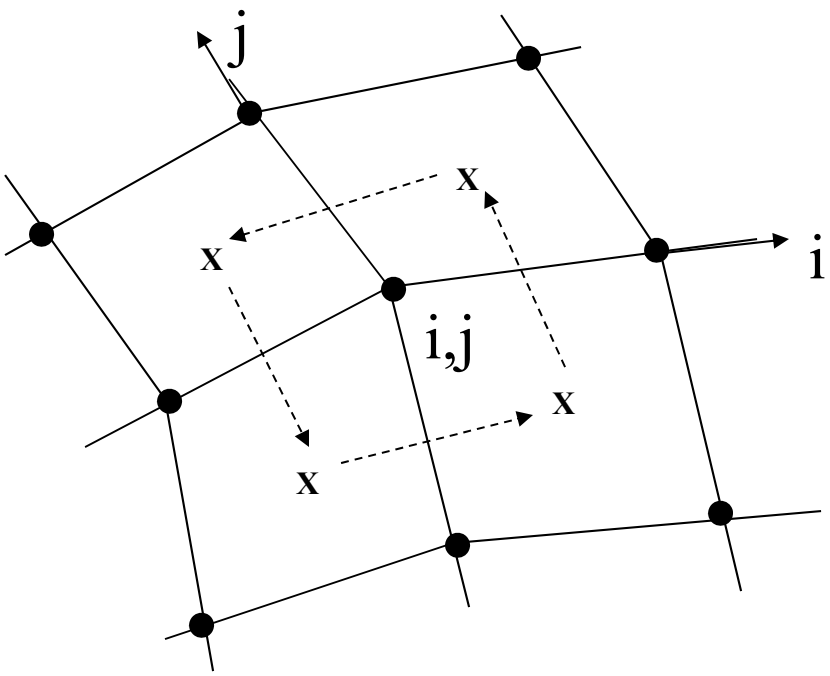
Cell-Centered Scheme First-Derivative Operator

- **Cell-centered schemes integrate around the "primary" control volumes made up from the nodes**
- **Physical variables are stored at the cell centers whereas coordinates are stored at the nodes**
- **First-derivatives at cell-centers are found using Green's integrations around the primary control volumes**



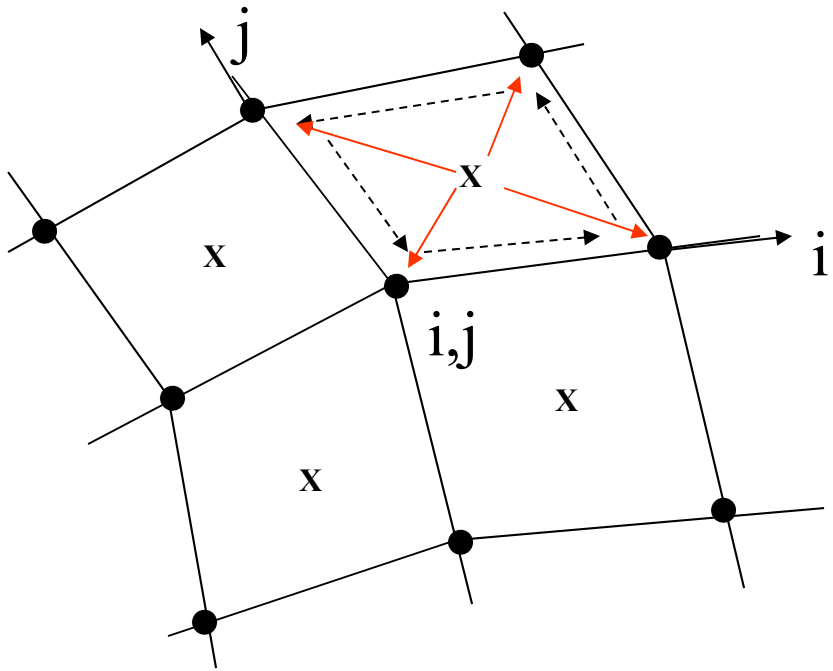
Node-Centered Scheme First-Derivative Operator

- **Node-centered schemes could integrate around the "secondary" control volumes made up from the nodes**
- **Physical variables and coordinates are both stored at the nodes**
- **First-derivatives at nodes are found using Green's integrations around the secondary control volumes**



Alternate Node-Centered Scheme First-Derivative Operator

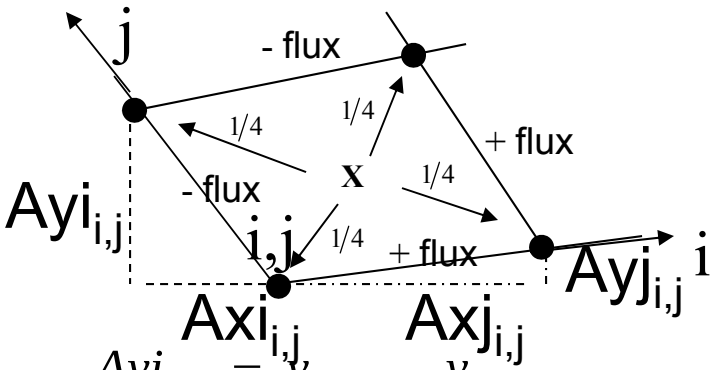
- **Node-centered schemes could also be found from integrations around the "primary" control volumes made up from the nodes followed by subsequent distributions to the nodes**
- **The equivalent of the node-centered integration around the node requires that ¼ * of the cell-centered first derivative value be distributed to the nodes making up the primary control volume**



- **Note that this operator stays within the block of cells and never requires information outside of it**
- **Essentially an averaging operator**

* some error for stretched grids

Node-Centered Scheme First-Derivative Operator



- **Trapezoidal counter-clockwise integration to get first x-derivative at cell-center using Gauss's theorem:** $\frac{\partial \phi}{\partial x} = \frac{1}{Vol}\oint_{CV} \phi\, dy \qquad \frac{\partial \phi}{\partial y} = -\frac{1}{Vol}\oint_{CV} \phi\, dx$

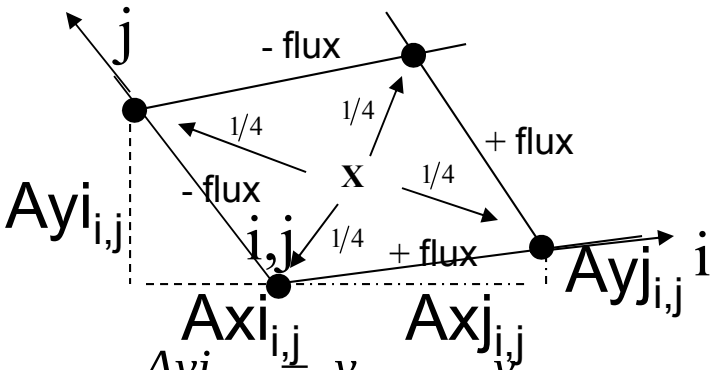
$$Ayi_{i,j} = y_{i,j+1} - y_{i,j}$$
$$Axi_{i,j} = x_{i,j+1} - x_{i,j}$$
$$Ayj_{i,j} = y_{i+1,j} - y_{i,j}$$
$$Axj_{i,j} = x_{i+1,j} - x_{i,j}$$

$$\frac{\partial T}{\partial x} = \frac{1}{2Vol_{i+\frac{1}{2},j+\frac{1}{2}}}\left[\sum_{\substack{faces \\ of\ face}} T_{average} \Delta y_{\substack{axross\ face\ in \\ counter-clockwise \\ direction}}\right]$$

$$\left.\frac{\partial T}{\partial x}\right|_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{2Vol_{i+\frac{1}{2},j+\frac{1}{2}}}\begin{bmatrix} \left(T_{i+1,j} + T_{i+1,j+1}\right)Ayi_{i+1,j} - \left(T_{i,j} + T_{i,j+1}\right)Ayi_{i,j} \\ -\left(T_{i,j+1} + T_{i+1,j+1}\right)Ayj_{i,j+1} + \left(T_{i,j} + T_{i+1,j}\right)Ayj_{i,j} \end{bmatrix}$$

$$= \frac{1}{2Vol_{i+\frac{1}{2},j+\frac{1}{2}}}\begin{bmatrix} axialfluxi_{i+1} - axialfluxi_{i} \\ -axialfluxj_{j+1} + axialfluxj_{j} \end{bmatrix}$$

Node-Centered Scheme First-Derivative Operator



- **Likewise, trapezoidal counter-clockwise integration to get first y-derivative at cell-center using Gauss's theorem:**

$$Ayi_{i,j} = y_{i,j+1} - y_{i,j}$$
$$Axi_{i,j} = x_{i,j+1} - x_{i,j}$$
$$Ayj_{i,j} = y_{i+1,j} - y_{i,j}$$
$$Axj_{i,j} = x_{i+1,j} - x_{i,j}$$

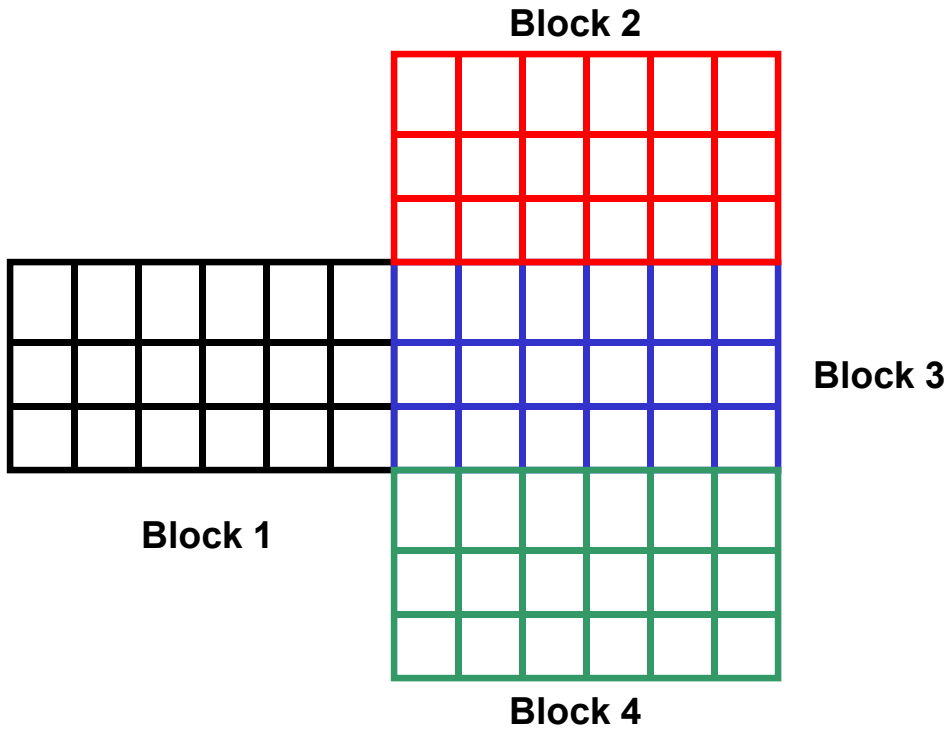
$$\frac{\partial T}{\partial y} = -\frac{1}{2Vol_{i+\frac{1}{2},j+\frac{1}{2}}}\left[\sum_{faces} T_{\substack{average\\ of\ face}} \Delta x_{\substack{axross\ face\ in\\ counter-clockwise\\ direction}}\right]$$

$$\left.\frac{\partial T}{\partial y}\right|_{i+\frac{1}{2},j+\frac{1}{2}} = -\frac{1}{2Vol_{i+\frac{1}{2},j+\frac{1}{2}}}\begin{bmatrix}\left(T_{i+1,j}+T_{i+1,j+1}\right)Axi_{i+1,j} - \left(T_{i,j}+T_{i,j+1}\right)Axi_{i,j}\\ -\left(T_{i,j+1}+T_{i+1,j+1}\right)Axj_{i,j+1} + \left(T_{i,j}+T_{i+1,j}\right)Axj_{i,j}\end{bmatrix}$$

$$= -\frac{1}{2Vol_{i+\frac{1}{2},j+\frac{1}{2}}}\begin{bmatrix}tangentialfluxi_{i+1} - tangentialfluxi_{i}\\ -tangentialfluxj_{j+1} + tangentialfluxj_{j}\end{bmatrix}$$

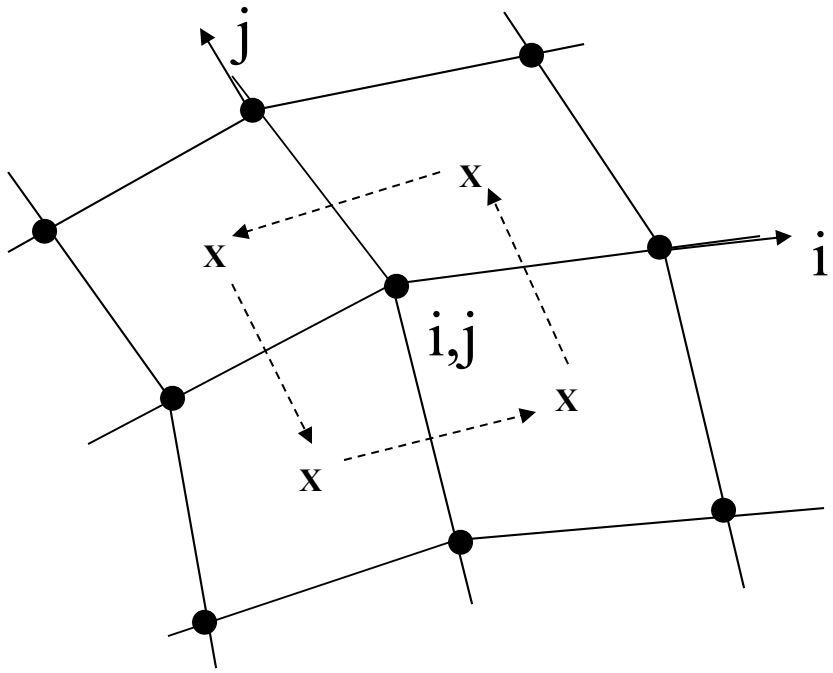
Node-Centered Scheme First-Derivative Operator

- **IF additional blocks are point-matched along the edges, then the first derivatives can be obtained by accumulating the contributions from the adjacent blocks (gather-add operation)**
- **At physical boundaries, Dirichlet or Neumann boundary conditions often preclude the need for derivatives to be determined using finite volume operators**



Node-Centered Scheme Second-Derivative Operator

- **Node centered second derivative operators can be found more directly by using Green's theorem and integrating the first-derivatives around the secondary control volume**
- **This gives the second derivatives directly at the nodes ***
- **Note, however that this could require information outside of a block when performing this operation along the block edges**

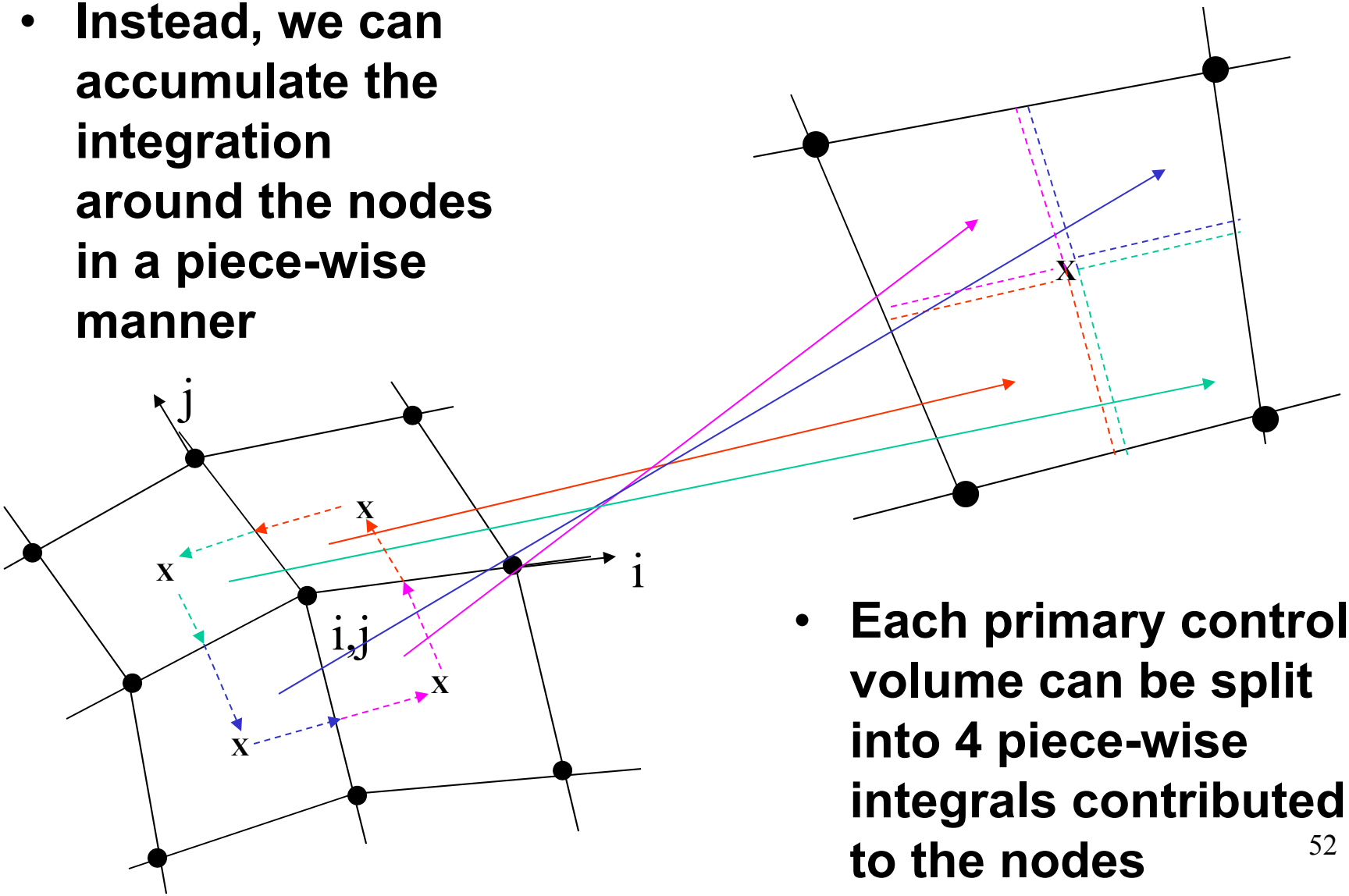


- **We can reformulate this operator to avoid this problem**

* some error for stretched grids

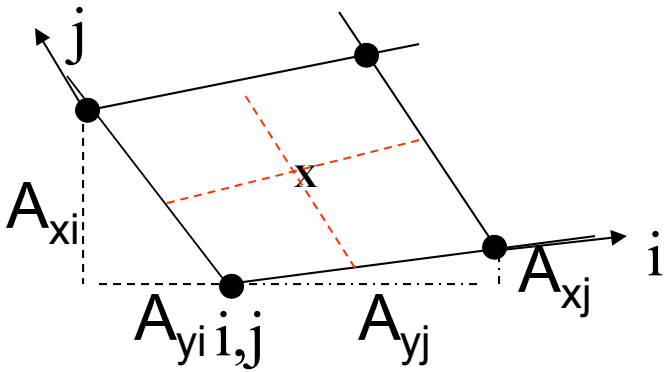
Alternate Distributive Scheme Second-Derivative Operator

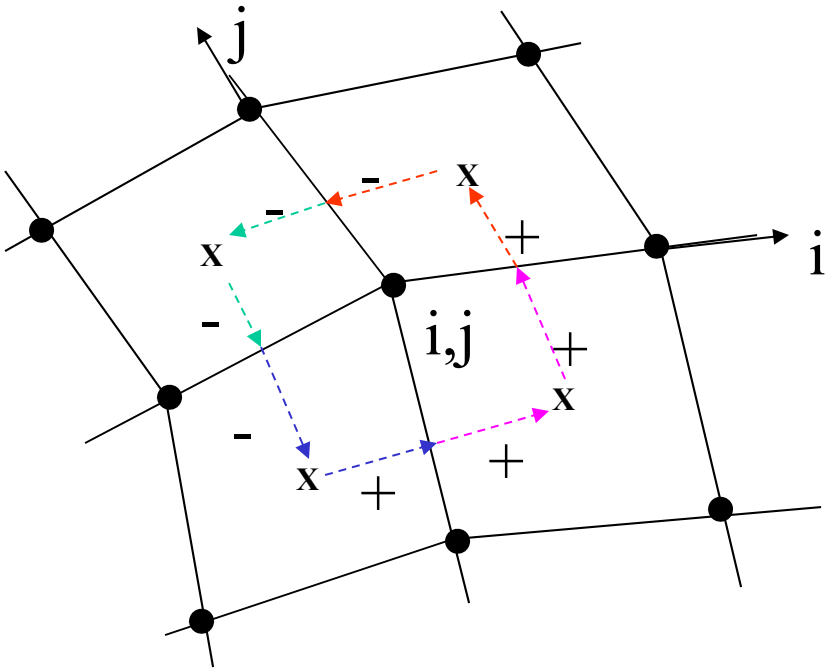
- **Instead, we can accumulate the integration around the nodes in a piece-wise manner**



- **Each primary control volume can be split into 4 piece-wise integrals contributed to the nodes**

Alternate Distributive Scheme Second-Derivative Operator





- **Areas used in calculation of fluxes:**

$$Ayi_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{Ayi_{i+1,j} + Ayi_{i,j}}{4}$$

$$Axi_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{Axi_{i+1,j} + Axi_{i,j}}{4}$$

$$Ayj_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{Ayj_{i,j} + Ayj_{i,j+1}}{4}$$

$$Axj_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{Axj_{i,j} + Axj_{i,j+1}}{4}$$

Alternate Distributive Scheme Second-Derivative Operator

$$\frac{\partial^2 T}{\partial x^2}_{i,j+1} = \frac{\partial^2 T}{\partial x^2}_{i,j+1}$$

$$+ \text{axialfluxi of } \frac{\partial T}{\partial x}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$+ \text{axialfluxj of } \frac{\partial T}{\partial x}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$\frac{\partial^2 T}{\partial x^2}_{i+1,j+1} = \frac{\partial^2 T}{\partial x^2}_{i+1,j+1}$$

$$- \text{axialfluxi of } \frac{\partial T}{\partial x}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$+ \text{axialfluxj of } \frac{\partial T}{\partial x}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$\frac{\partial^2 T}{\partial x^2}_{i+1,j} = \frac{\partial^2 T}{\partial x^2}_{i+1,j}$$

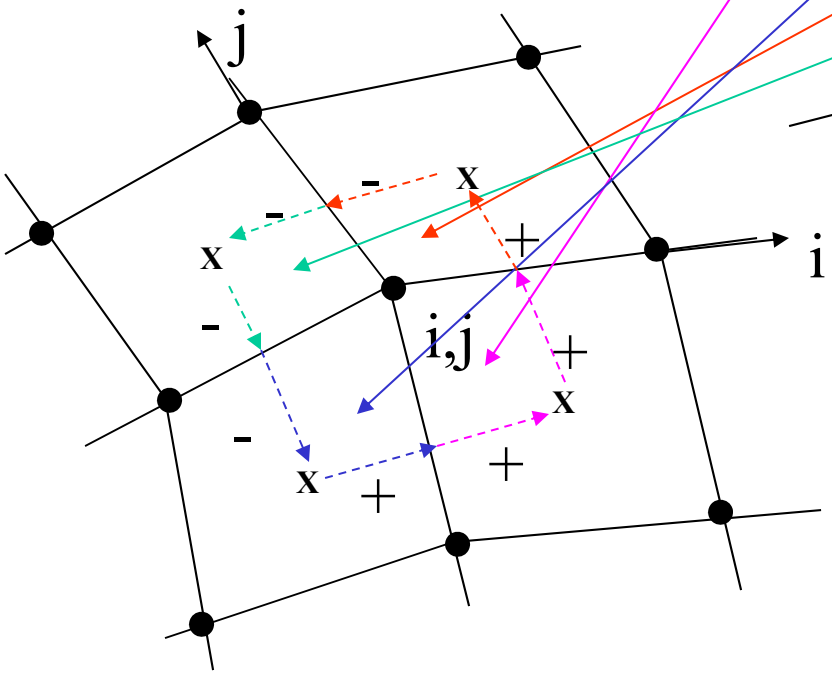
$$- \text{axialfluxi of } \frac{\partial T}{\partial x}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$- \text{axialfluxj of } \frac{\partial T}{\partial x}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$\frac{\partial^2 T}{\partial x^2}_{i,j} = \frac{\partial^2 T}{\partial x^2}_{i,j}$$

$$+ \text{axialfluxi of } \frac{\partial T}{\partial x}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$- \text{axialfluxj of } \frac{\partial T}{\partial x}_{i+\frac{1}{2},j+\frac{1}{2}}$$



Alternate Distributive Scheme Second-Derivative Operator

$$\frac{\partial^2 T}{\partial y^2}_{i,j+1} = \frac{\partial^2 T}{\partial y^2}_{i,j+1}$$

$$- tangentialfluxi\ of\ \frac{\partial T}{\partial y}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$- tangentialfluxj\ of\ \frac{\partial T}{\partial y}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$\frac{\partial^2 T}{\partial y^2}_{i+1,j+1} = \frac{\partial^2 T}{\partial y^2}_{i+1,j+1}$$

$$+ tangentialfluxi\ of\ \frac{\partial T}{\partial y}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$- tangentialfluxj\ of\ \frac{\partial T}{\partial y}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$\frac{\partial^2 T}{\partial y^2}_{i+1,j} = \frac{\partial^2 T}{\partial y^2}_{i+1,j}$$

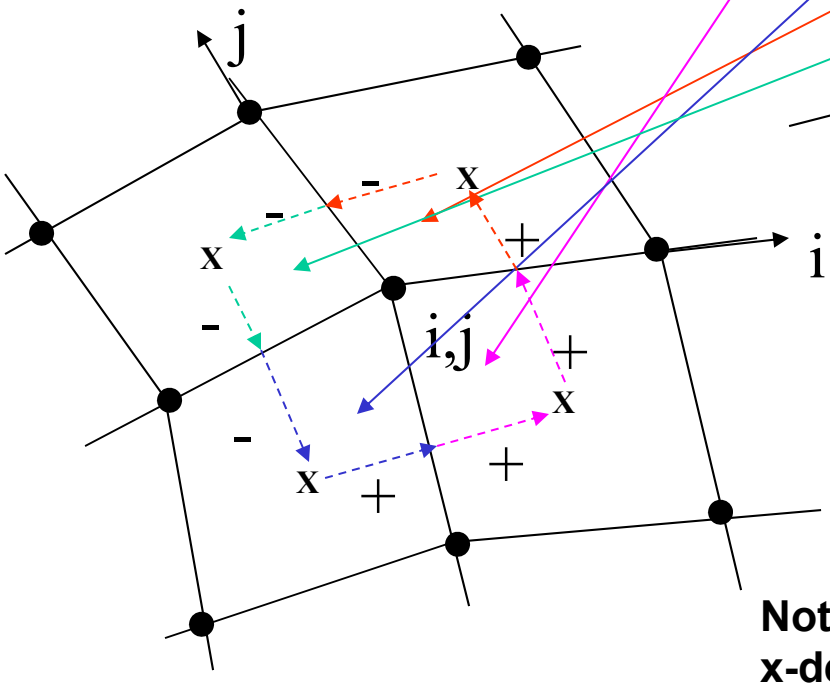
$$+ tangentialfluxi\ of\ \frac{\partial T}{\partial y}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$+ tangentialfluxj\ of\ \frac{\partial T}{\partial y}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$\frac{\partial^2 T}{\partial y^2}_{i,j} = \frac{\partial^2 T}{\partial y^2}_{i,j}$$

$$- tangentialfluxi\ of\ \frac{\partial T}{\partial y}_{i+\frac{1}{2},j+\frac{1}{2}}$$

$$+ tangentialfluxj\ of\ \frac{\partial T}{\partial y}_{i+\frac{1}{2},j+\frac{1}{2}}$$



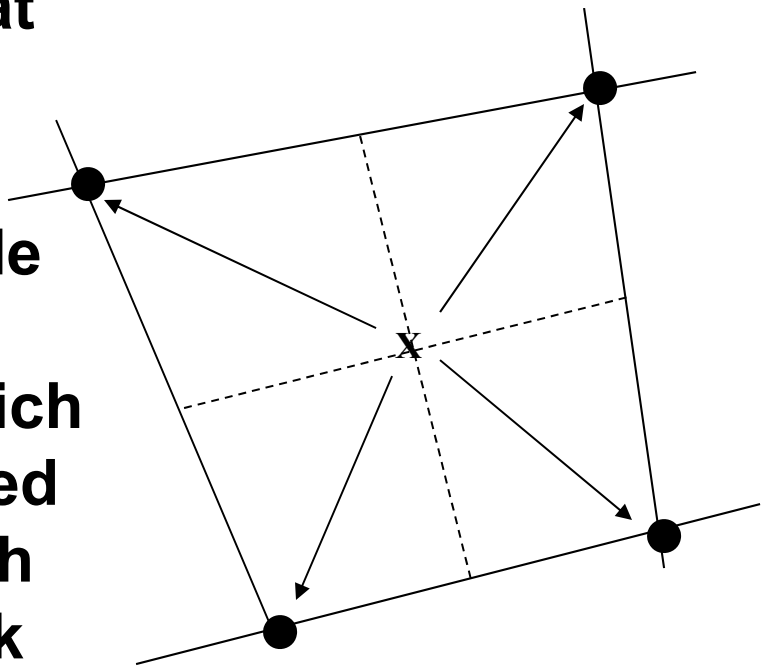
Note sign change in fluxes compared to x-derivative due to Gauss theorem

Node-Centered Scheme Second-Derivative Operators

- **The advantages of this accumulation operator are**
  - Both the first and second derivatives at the interior nodes can be found in one loop across the primary control volume cells
  - the second-derivatives at the edges where adjoining blocks exist can be again found with a simple gather-add operation

Averaging Operator

- We often need to find the average value of quantities at the nodes from the cell-centered values
- We can find the average node values using a similar distribution technique in which we distribute the cell-centered value to the nodes along with a "hit" index that keeps track of the number of contributions
- In a second loop, we then divide the total accumulated contribution at the node by the number of "hits" to obtain the average



- The advantage of this approach is it allows for any number of cells around an edge or corner node