CS 423
Computer Architecture
Spring 2012

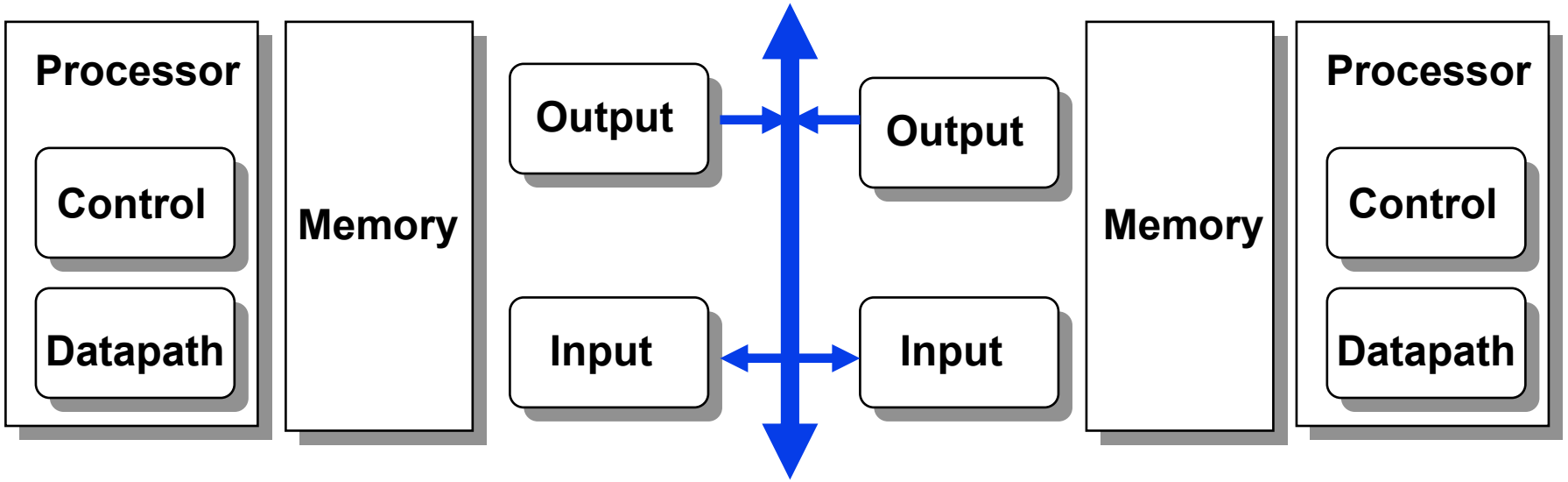
Lecture 08:
Intro to Multiprocessors

Ozcan Ozturk

http://www.cs.bilkent.edu.tr/~ozturk/cs423/

[Adapted from *Computer Organization and Design*,
Patterson & Hennessy, © 2005, UCB]

The Big Picture: Where are We Now?



- Multiprocessor – multiple processors with a single shared address space
- Cluster – multiple computers (each with their own address space) connected over a local area network (LAN) functioning as a single system

Applications Needing "Supercomputing"

- Energy (plasma physics (simulating fusion reactions), geophysical (petroleum) exploration)
- DoE stockpile stewardship (to ensure the safety and reliability of the nation's stockpile of nuclear weapons)
- Earth and climate (climate and weather prediction, earthquake, tsunami prediction and mitigation of risks)
- Transportation (improving vehicles' airflow dynamics, fuel consumption, crashworthiness, noise reduction)
- Bioinformatics and computational biology (genomics, protein folding, designer drugs)
- Societal health and safety (pollution reduction, disaster planning, terrorist action detection)

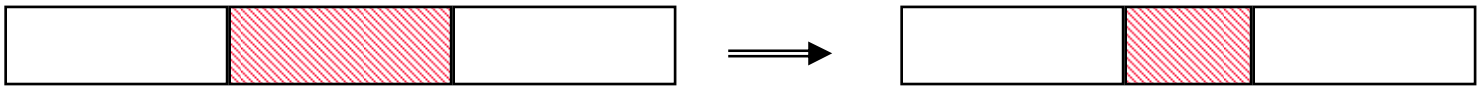
http://www.nap.edu/books/0309095026/html/

Encountering Amdahl's Law

- Speedup due to enhancement E is

$$\text{Speedup w/ E} = \frac{\text{Exec time w/o E}}{\text{Exec time w/ E}}$$

- Suppose that enhancement E accelerates a fraction F (F <1) of the task by a factor S (S>1) and the remainder of the task is unaffected



$$\text{ExTime w/ E} = \text{ExTime w/o E} \times$$

$$\text{Speedup w/ E} =$$

Examples: Amdahl's Law

Speedup w/ E =

- Consider an enhancement which runs 20 times faster but which is only usable 25% of the time.

  Speedup w/ E =

- What if its usable only 15% of the time?

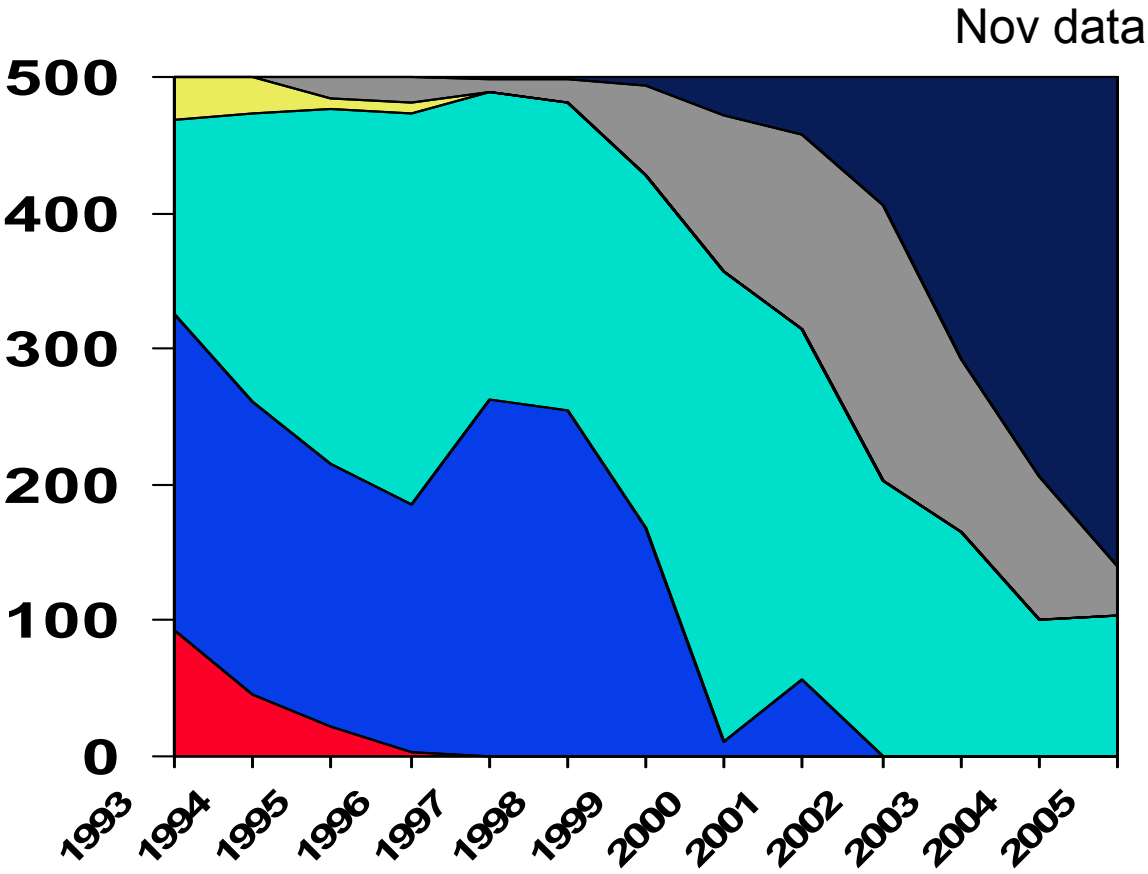
  Speedup w/ E =

- Amdahl's Law tells us that to achieve linear speedup with 100 processors, none of the original computation can be scalar!

- To get a speedup of 99 from 100 processors, the percentage of the original program that could be scalar would have to be 0.01% or less

Supercomputer Style Migration (Top500)

http://www.top500.org/list/2009/06/100



Cluster – whole computers interconnected using their I/O bus

Constellation – a cluster that uses an SMP multiprocessor as the building block

- In the last 8 years uniprocessor and SIMDs disappeared while Clusters and Constellations grew from 3% to 80%

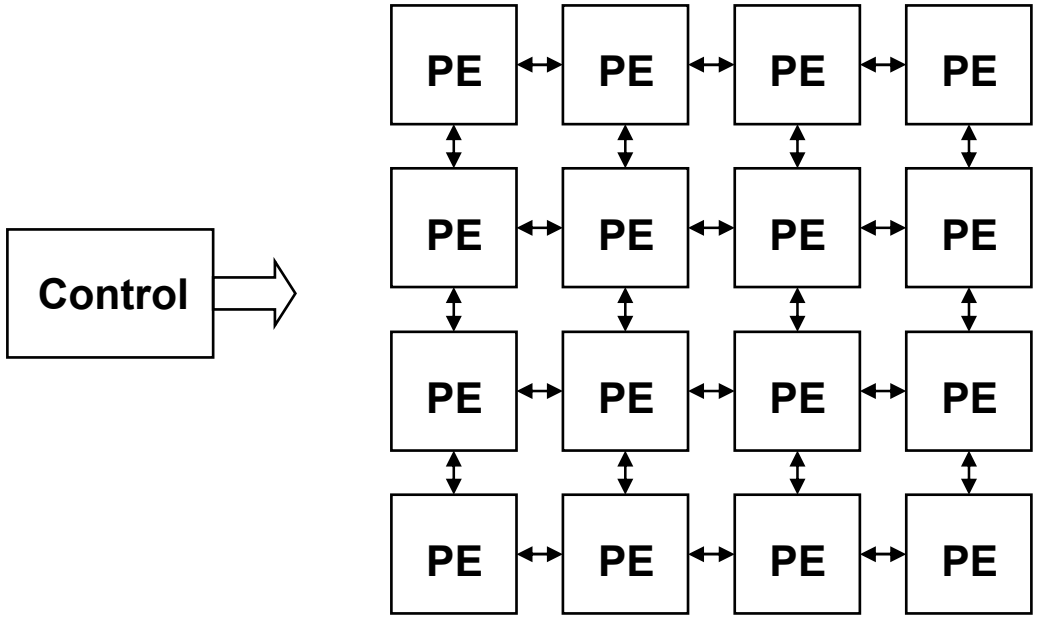
Multiprocessor/Clusters Key Questions

- Q1 – How do they share data?
- Q2 – How do they coordinate?
- Q3 – How scalable is the architecture? How many processors can be supported?

Flynn's Classification Scheme

- SISD – single instruction, single data stream
  - aka uniprocessor - what we have been talking about all semester
- SIMD – single instruction, multiple data streams
  - single control unit broadcasting operations to multiple datapaths
- MISD – multiple instruction, single data
  - no such machine (although some people put vector machines in this category)
- MIMD – multiple instructions, multiple data streams
  - aka multiprocessors (SMPs, MPPs, clusters, NOWs)
- Now obsolete except for . . .

SIMD Processors



- Single control unit
- Multiple datapaths (processing elements – PEs) running in parallel
  - Q1 – PEs are interconnected (usually via a mesh or torus) and exchange/share data as directed by the control unit
  - Q2 – Each PE performs the same operation on its own local data

Example SIMD Machines

	Maker	Year	# PEs	# b/ PE	Max memory (MB)	PE clock (MHz)	System BW (MB/s)
Illiac IV	UIUC	1972	64	64	1	13	2,560
DAP	ICL	1980	4,096	1	2	5	2,560
MPP	Goodyear	1982	16,384	1	2	10	20,480
CM-2	Thinking Machines	1987	65,536	1	512	7	16,384
MP-1216	MasPar	1989	16,384	4	1024	25	23,000

Multiprocessor Basic Organizations

- Processors connected by a single bus
- Processors connected by a network

			# of Proc
Communication model	Message passing		8 to 2048
	Shared address	NUMA	8 to 256
		UMA	2 to 64
Physical connection	Network		8 to 256
	Bus		2 to 36

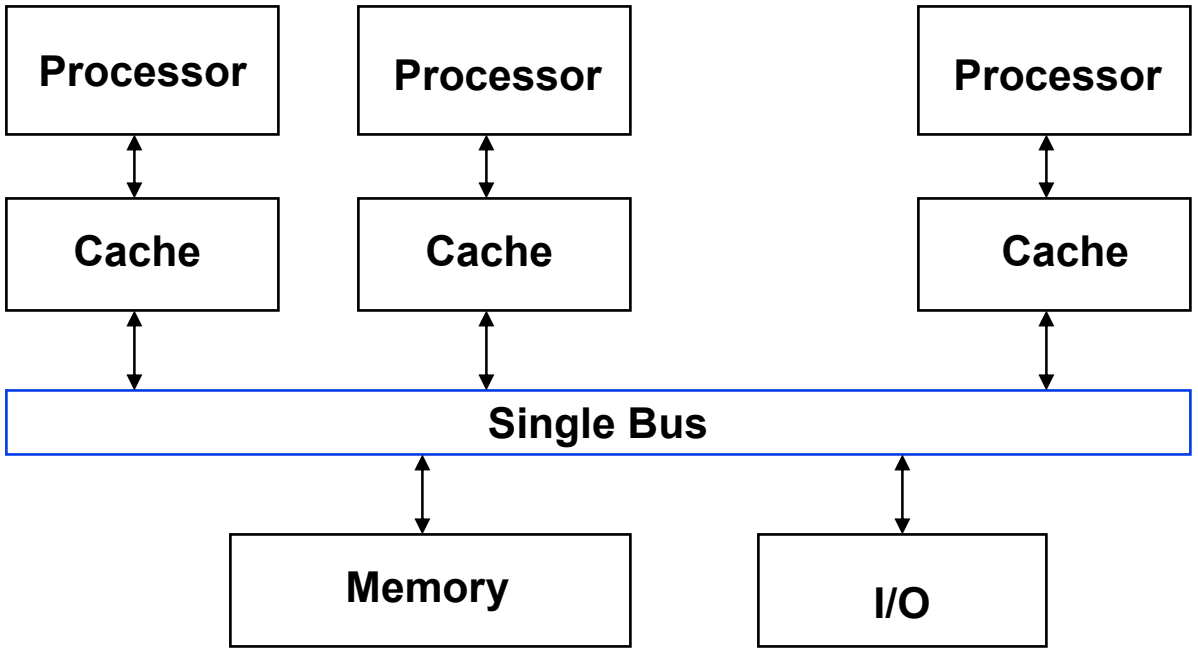
Shared Address (Shared Memory) Multi's

- Q1 – Single address space shared by all the processors
- Q2 – Processors coordinate/communicate through shared variables in memory (via loads and stores)
  - Use of shared data must be coordinated via synchronization primitives (locks)
- UMAs (uniform memory access) – aka SMP (symmetric multiprocessors)
  - all accesses to main memory take the same amount of time no matter which processor makes the request or which location is requested
- NUMAs (nonuniform memory access)
  - some main memory accesses are faster than others depending on the processor making the request and which location is requested
  - can scale to larger sizes than UMAs so are potentially higher performance

N/UMA Remote Memory Access Times (RMAT)

	Year	Type	Max Proc	Interconnection Network	RMAT (ns)
Sun Starfire	1996	SMP	64	Address buses, data switch	500
Cray 3TE	1996	NUMA	2048	2-way 3D torus	300
HP V	1998	SMP	32	8 x 8 crossbar	1000
SGI Origin 3000	1999	NUMA	512	Fat tree	500
Compaq AlphaServer GS	1999	SMP	32	Switched bus	400
Sun V880	2002	SMP	8	Switched bus	240
HP Superdome 9000	2003	SMP	64	Switched bus	275
NASA Columbia	2004	NUMA	10240	Fat tree	???

Single Bus (Shared Address UMA) Multi's



- Caches are used to reduce latency *and* to lower bus traffic
- Must provide hardware to ensure that caches and memory are consistent (cache coherency)
- Must provide a hardware mechanism to support process synchronization

Summing 100,000 Numbers on 100 Processors

- How would you add 100,000 Numbers on 100 processors?

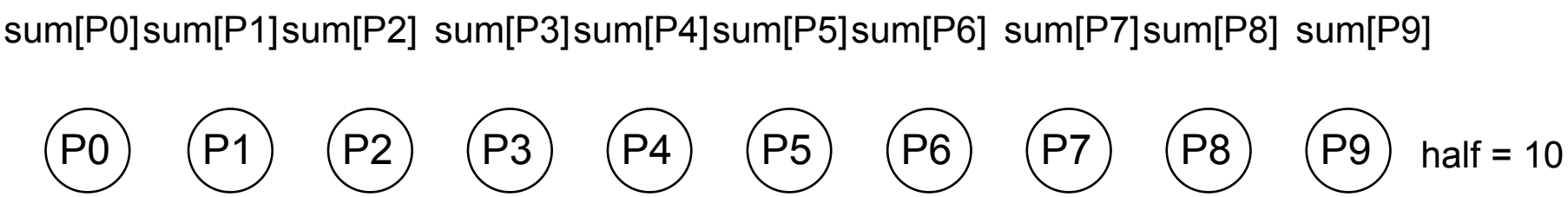
Summing 100,000 Numbers on 100 Processors

```
sum[Pn] = 0;
for (i = 1000*Pn; i< 1000*(Pn+1); i = i + 1)
  sum[Pn] = sum[Pn] + A[i];
```

- Processors start by running a loop that sums their subset of vector A numbers (vectors A and sum are shared variables, Pn is the processor's number, i is a private variable)
- The processors then coordinate in adding together the partial sums (half is a private variable initialized to 100 (the number of processors))

```
repeat
  synch();                  /*synchronize first
  if (half%2 != 0 && Pn == 0)
      sum[0] = sum[0] + sum[half-1];
  half = half/2
  if (Pn<half) sum[Pn] = sum[Pn] + sum[Pn+half]
until (half == 1);          /*final sum in sum[0]
```

An Example with 10 Processors

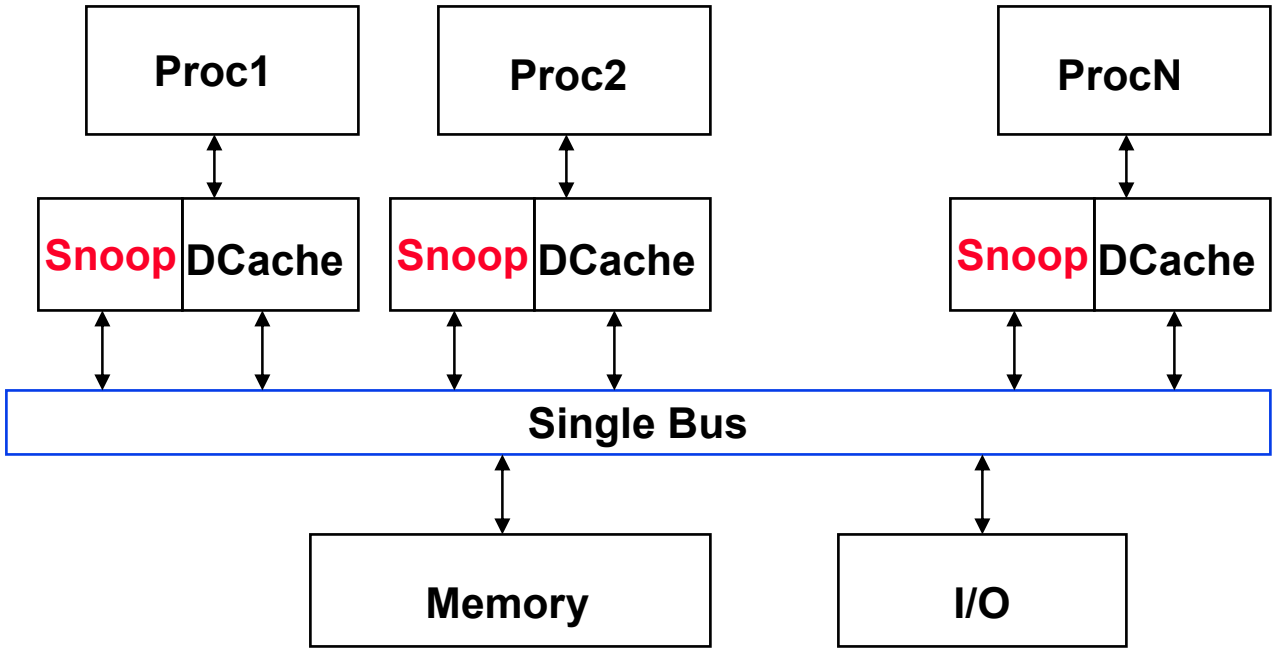


Message Passing Multiprocessors

- Each processor has its own private address space
- Q1 – Processors share data by explicitly sending and receiving information (messages)
- Q2 – Coordination is built into message passing primitives (send and receive)

Multiprocessor Cache Coherency

- Cache coherency protocols
  - Bus snooping – cache controllers monitor shared bus traffic with duplicate address tag hardware (so they don't interfere with processor's access to the cache)



Bus Snooping Protocols

- Multiple copies are not a problem when reading
- Processor must have exclusive access to write a word
  - What happens if two processors try to write to the same shared data word in the same clock cycle? The bus arbiter decides which processor gets the bus first (and this will be the processor with the *first* exclusive access). Then the second processor will get exclusive access. Thus, bus arbitration forces sequential behavior.
  - This sequential consistency is the most conservative of the memory consistency models. With it, the result of any execution is the same as if the accesses of each processor were kept in order and the accesses among different processors were interleaved.
- All other processors sharing that data must be informed of writes

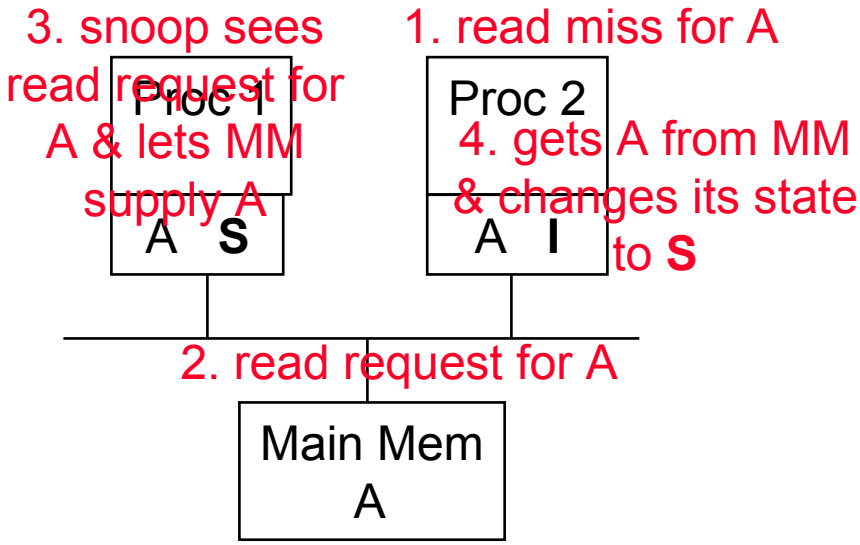
Handling Writes

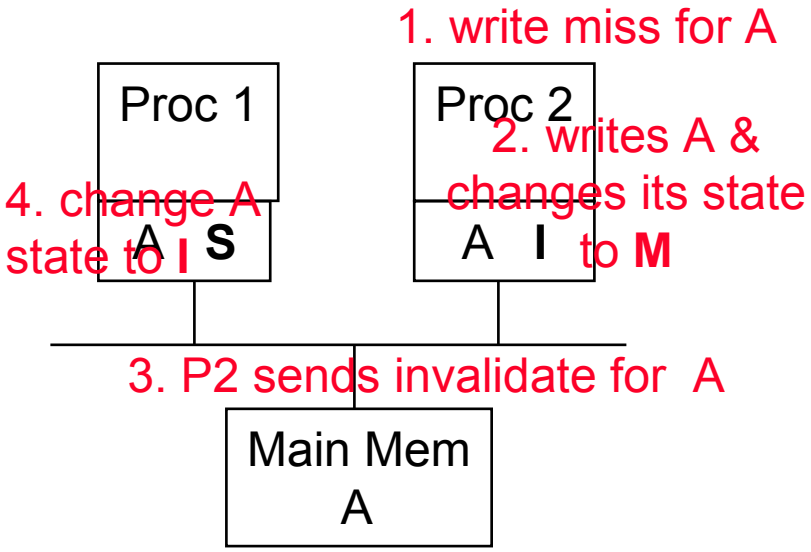
Ensuring that all other processors sharing data are informed of writes can be handled two ways:

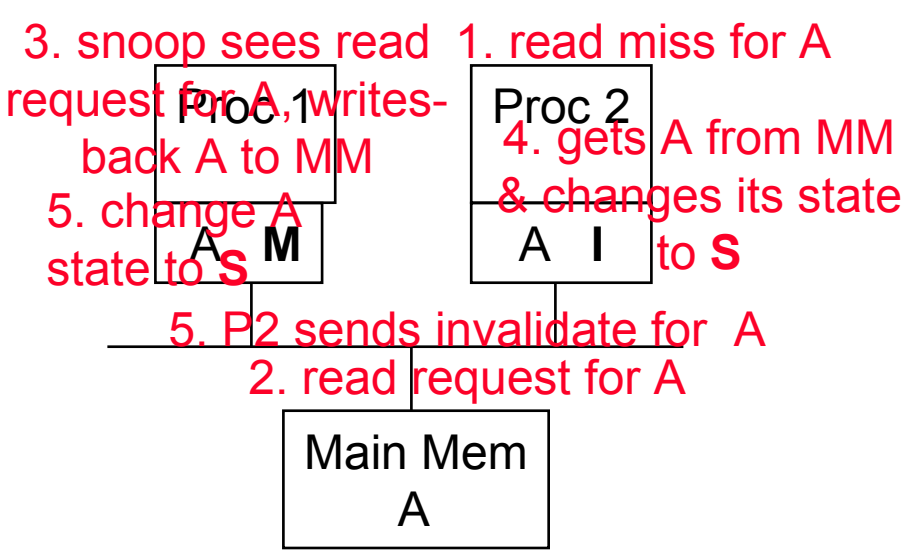
1. Write-update (write-broadcast) – writing processor broadcasts new data over the bus, all copies are updated
   - All writes go to the bus → higher bus traffic
   - Since new values appear in caches sooner, can reduce latency
2. Write-invalidate – writing processor issues invalidation signal on bus, cache snoops check to see if they have a copy of the data, if so they invalidate their cache block containing the word (this allows multiple readers but only one writer)
   - Uses the bus only on the first write → lower bus traffic, so better use of bus bandwidth

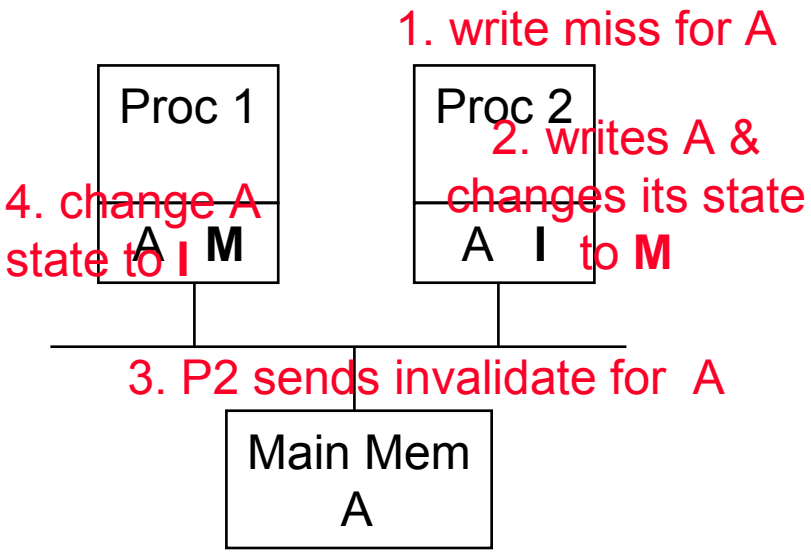
Write-Invalidate CC Examples

- I = invalid (many), S = shared (many), M = modified (only one)









Other Coherence Protocols

- There are many variations on cache coherence protocols

- Another write-invalidate protocol used in the Pentium 4 (and many other micro's) is MESI with four states:
  - Modified – same
  - Exclusive – only one copy of the shared data is allowed to be cached; memory has an up-to-date copy
    - Since there is only one copy of the block, write hits don't need to send invalidate signal
  - Shared – multiple copies of the shared data may be cached (i.e., data permitted to be cached with more than one processor); memory has an up-to-date copy
  - Invalid – same

Process Synchronization

- Need to be able to coordinate processes working on a common task
- Lock variables (semaphores) are used to coordinate or synchronize processes
- Need an architecture-supported arbitration mechanism to decide which processor gets access to the lock variable
  - Single bus provides arbitration mechanism, since the bus is the only path to memory – the processor that gets the bus wins

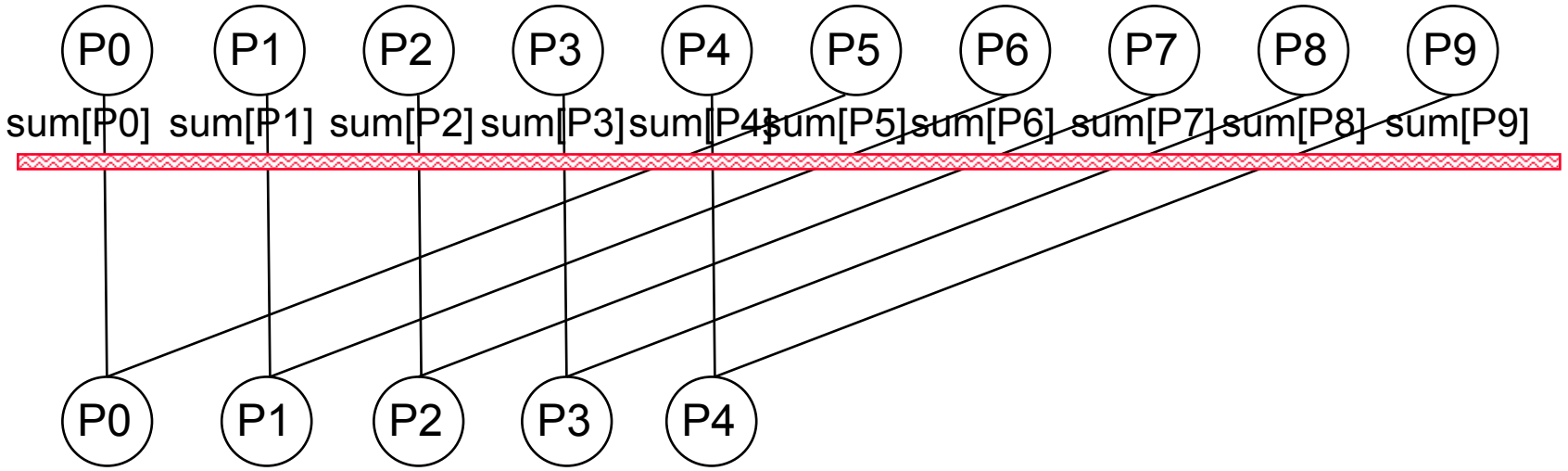
Review: Summing Numbers on a SMP

- Pn is the processor's number, vectors A and sum are shared variables, i is a private variable, half is a private variable initialized to the number of processors

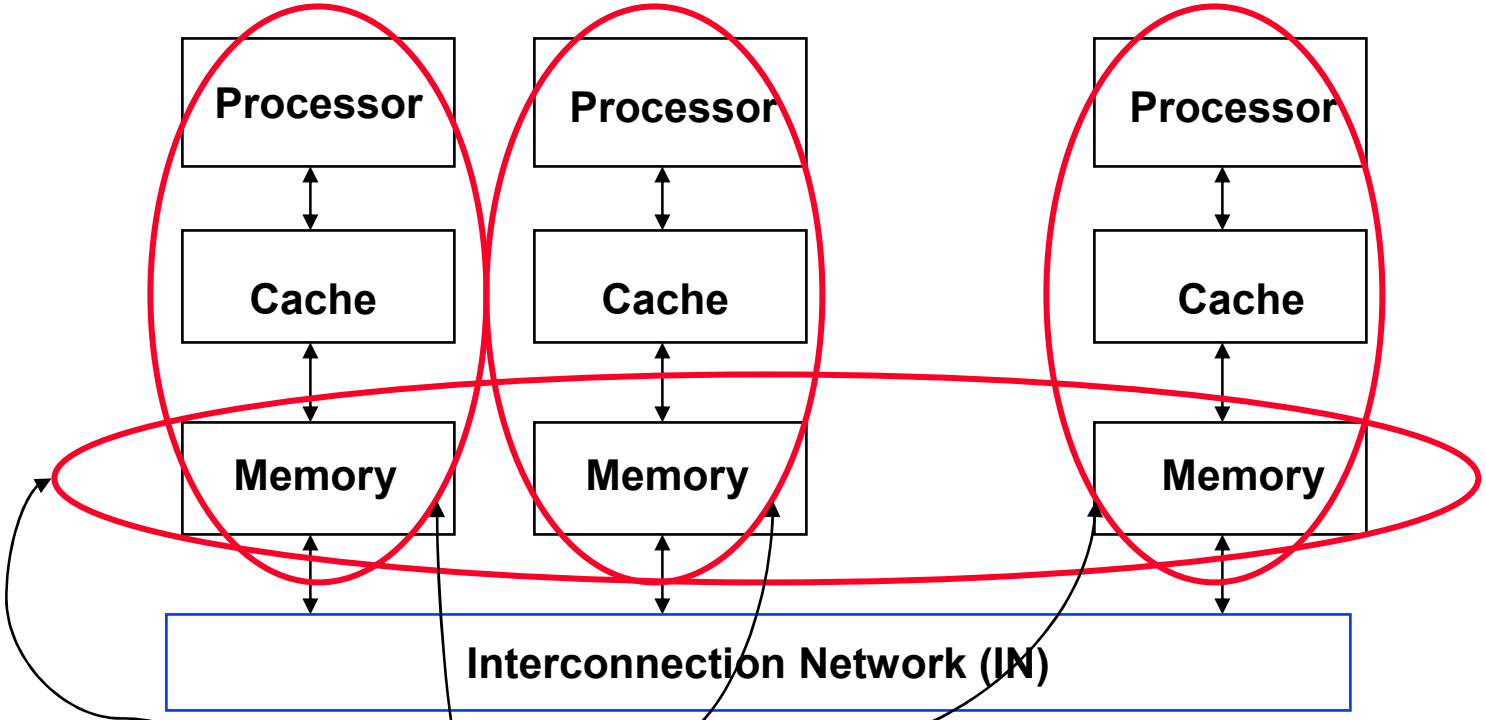
```
 sum[Pn] = 0;
 for (i = 1000*Pn; i< 1000*(Pn+1); i = i + 1)
   sum[Pn] = sum[Pn] + A[i];
                          /* each processor sums its
                          /* subset of vector A

repeat                    /* adding together the
                          /* partial sums
   synch();               /*synchronize first
   if (half%2 != 0 && Pn == 0)
       sum[0] = sum[0] + sum[half-1];
   half = half/2
   if (Pn<half) sum[Pn] = sum[Pn] + sum[Pn+half];
until (half == 1);        /*final sum in sum[0]
```

An Example with 10 Processors

- synch( ): Processors must synchronize before the "consumer" processor tries to read the results from the memory location written by the "producer" processor
  - Barrier synchronization – a synchronization scheme where processors wait at the barrier, not proceeding until every processor has reached it



Network Connected Multiprocessors



- Either a single address space (NUMA and ccNUMA) with implicit processor communication via loads and stores *or* multiple private memories with message passing communication with sends and receives
  - Interconnection network supports interprocessor communication

Summing 100,000 Numbers on 100 Processors

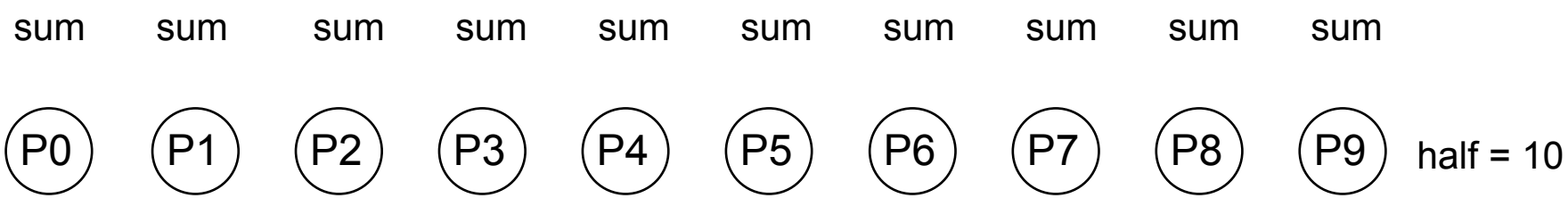
- Start by distributing 1000 elements of vector A to each of the local memories and summing each subset in parallel

```
sum = 0;
for (i = 0; i<1000; i = i + 1)
  sum = sum + Al[i];    /* sum local array subset
```

- The processors then coordinate in adding together the sub sums (Pn is the number of processors, `send(x,y)` sends value y to processor x, and `receive()` receives a value)

```
half = 100;
limit = 100;
repeat
  half = (half+1)/2;    /*dividing line
  if (Pn>= half && Pn<limit) send(Pn-half,sum);
  if (Pn<(limit/2)) sum = sum + receive();
  limit = half;
until (half == 1);      /*final sum in P0's sum
```

An Example with 10 Processors



Communication in Network Connected Multi's

- Implicit communication via loads and stores
  - hardware designers have to provide coherent caches and process synchronization primitive
  - lower communication overhead
  - harder to overlap computation with communication
  - more efficient to use an address to remote data when *demanded* rather than to send for it in case it *might* be used (such a machine has distributed shared memory (DSM))
- Explicit communication via sends and receives
  - simplest solution for hardware designers
  - higher communication overhead
  - easier to overlap computation with communication
  - easier for the programmer to optimize communication

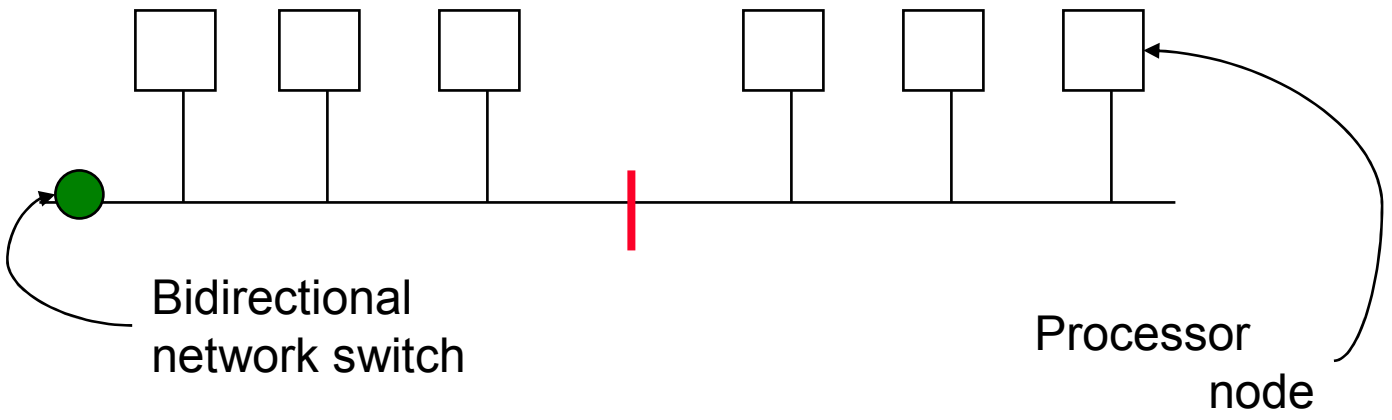
Cache Coherency in NUMAs

- For performance reasons we want to allow the shared data to be stored in caches
- Once again have multiple copies of the same data with the same address in different processors
  - bus snooping won't work, since there is no single bus on which all memory references are broadcast
- Directory-base protocols
  - keep a directory that is a repository for the state of every block in main memory (which caches have copies, whether it is dirty, etc.)
  - directory entries can be distributed (sharing status of a block always in a single known location) to reduce contention
  - directory controller sends explicit commands over the IN to each processor that has a copy of the data

IN Performance Metrics

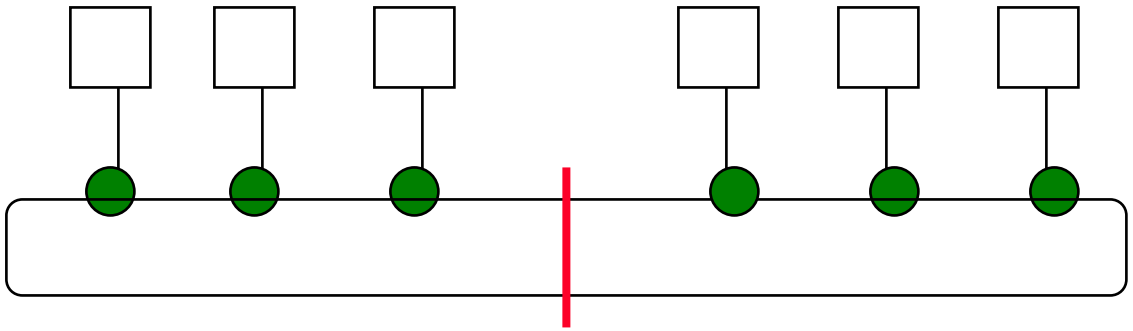
- Network cost
  - number of switches
  - number of (bidirectional) links on a switch to connect to the network (plus one link to connect to the processor)
  - width in bits per link, length of link
- Network bandwidth (NB) – represents the best case
  - bandwidth of each link * number of links
- Bisection bandwidth (BB) – represents the worst case
  - divide the machine in two parts, each with half the nodes and sum the bandwidth of the links that cross the dividing line
- Other IN performance issues
  - latency on an unloaded network to send and receive messages
  - throughput – maximum # of messages transmitted per unit time
  - # routing hops worst case, congestion control and delay

Bus IN



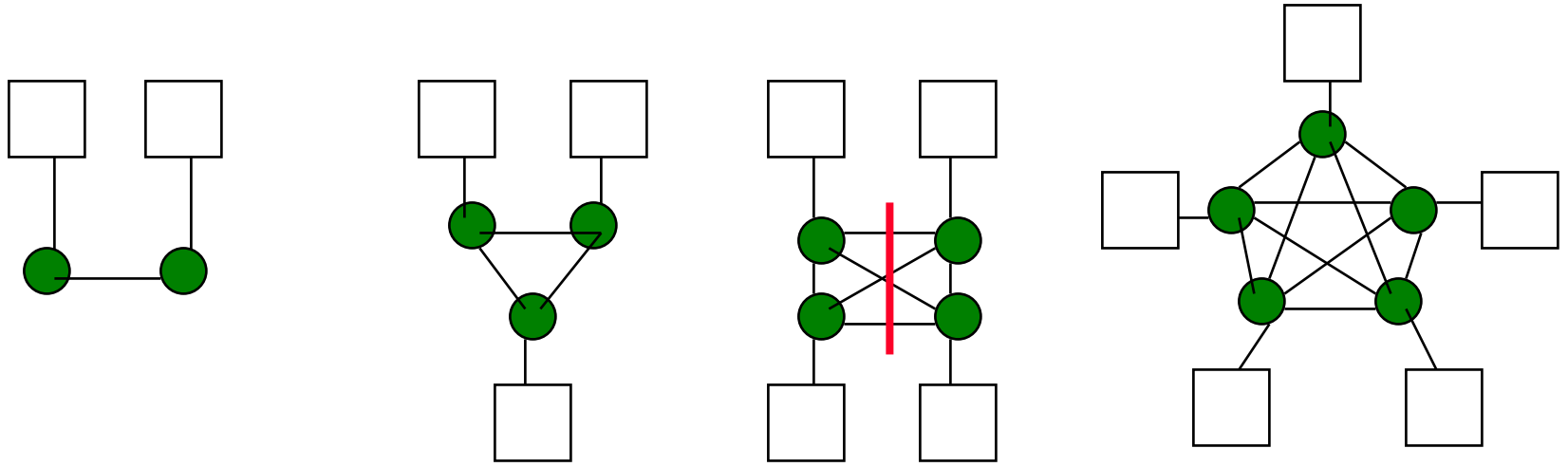
- N processors, 1 switch (●), 1 link (the bus)
- Only 1 simultaneous transfer at a time
  - NB = link (bus) bandwidth * 1
  - BB = link (bus) bandwidth * 1

Ring IN



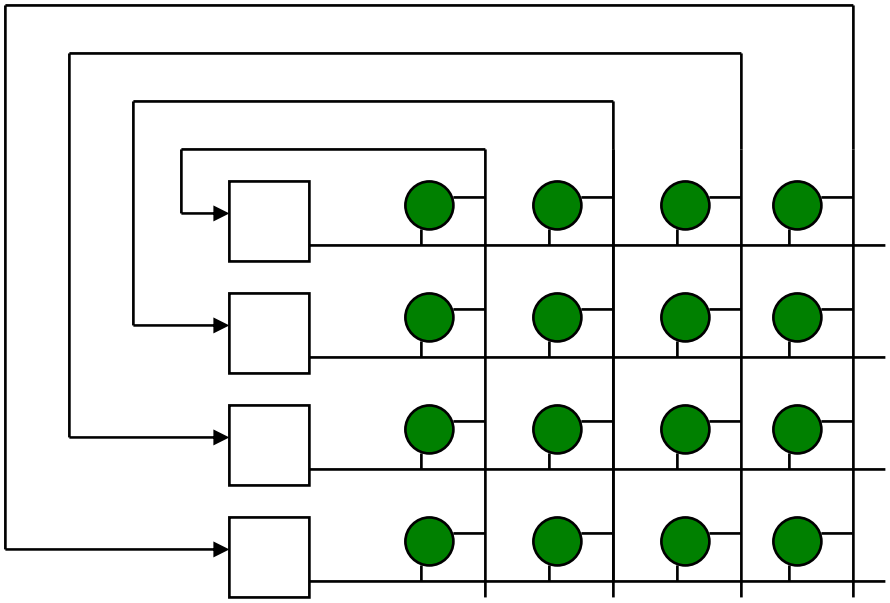
- N processors, N switches, 2 links/switch, N links
- N simultaneous transfers
  - NB = link bandwidth * N
  - BB = link bandwidth * 2
- If a link is as fast as a bus, the ring is only twice as fast as a bus in the worst case, but is N times faster in the best case

Fully Connected IN



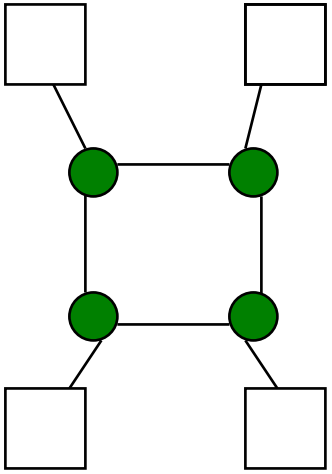
- N processors, N switches, N-1 links/switch, (N*(N-1))/2 links
- N simultaneous transfers
  - NB = link bandwidth * (N*(N-1))/2
  - BB = link bandwidth * $(N/2)^2$

Crossbar (Xbar) Connected IN

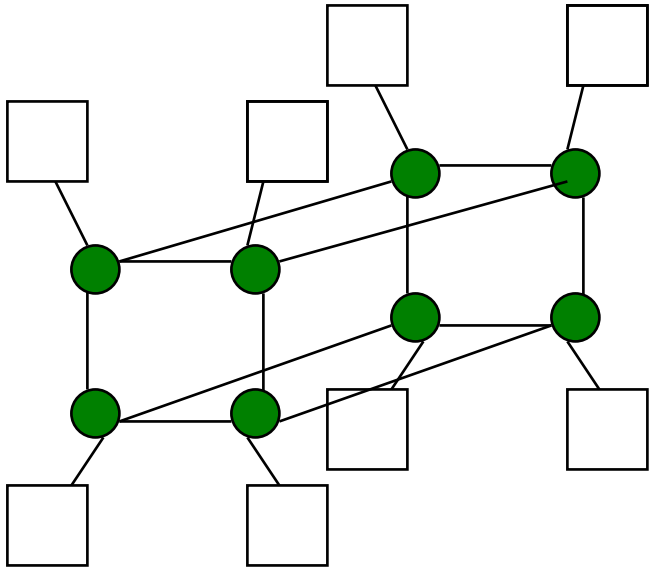


- N processors, N^2 switches (unidirectional),2 links/switch, N^2 links
- N simultaneous transfers
  - NB = link bandwidth * N
  - BB = link bandwidth * N/2

Hypercube (Binary N-cube) Connected IN



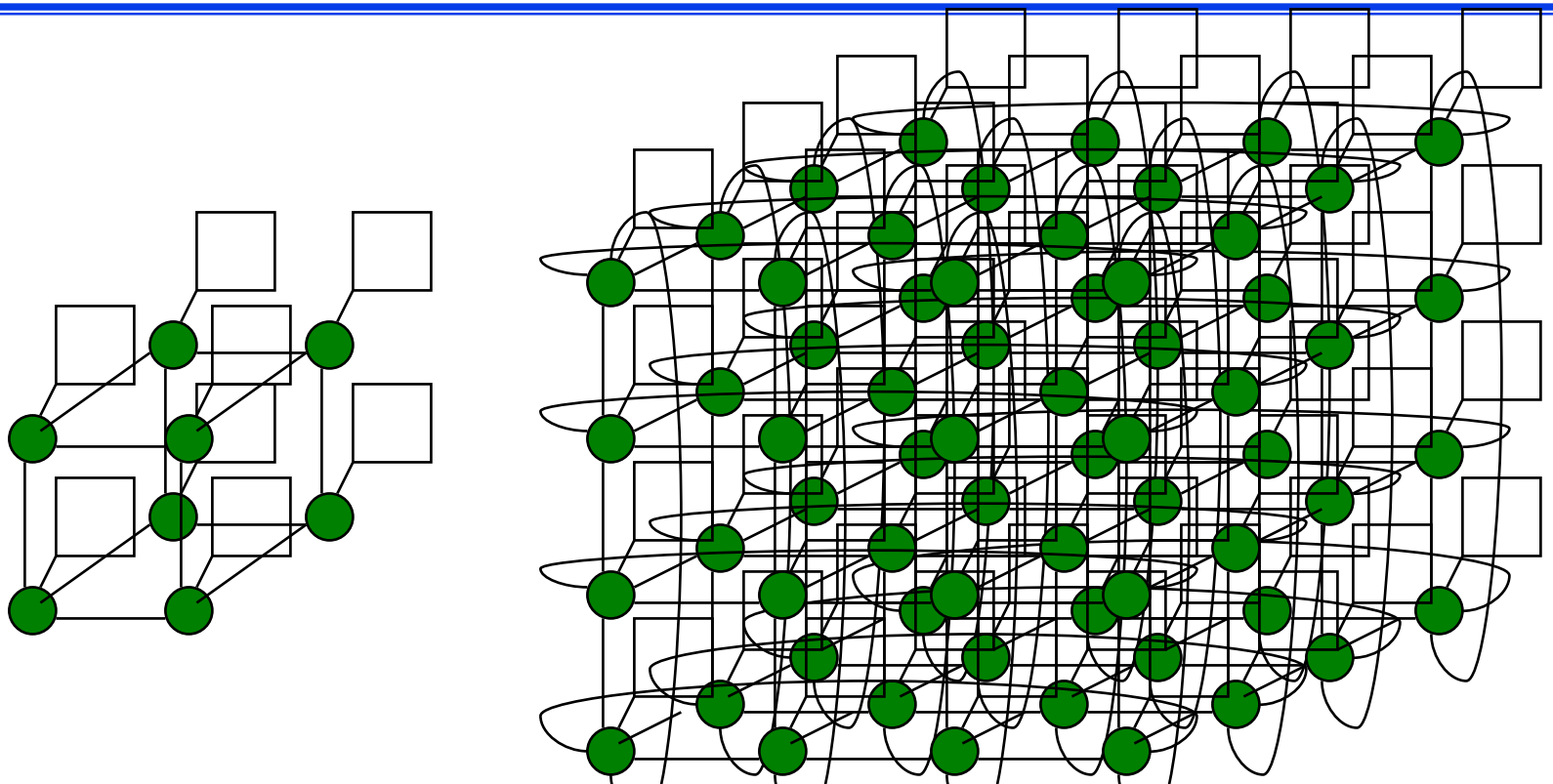
2-cube



3-cube

- N processors, N switches, logN links/switch, (NlogN)/2 links
- N simultaneous transfers
  - NB = link bandwidth * (NlogN)/2
  - BB = link bandwidth * N/2

2D and 3D Mesh/Torus Connected IN



- N processors, N switches, 2, 3, 4 (2D torus) or 6 (3D torus) links/switch, 4N/2 links or 6N/2 links
- N simultaneous transfers
  - NB = link bandwidth * 4N or link bandwidth * 6N
  - BB = link bandwidth * 2 $N^{1/2}$ or link bandwidth * 2 $N^{2/3}$

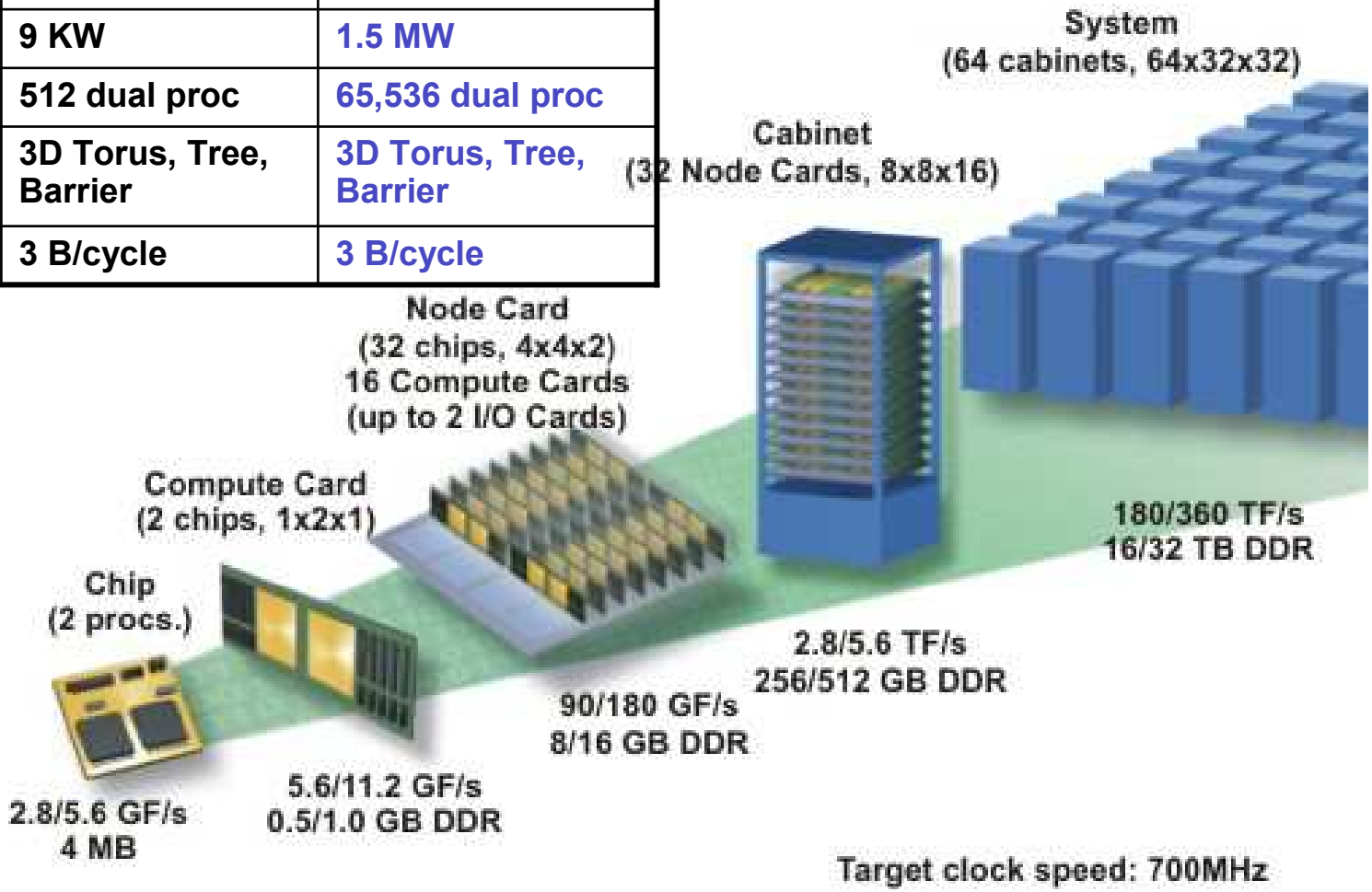
IN Comparison

- For a 64 processor system

	Bus	Ring	2D Torus	6-cube	Fully connected
Network bandwidth	1	64	256	192	2016
Bisection bandwidth	1	2	16	32	1024
Total # of switches	1	64	64	64	64
Links per switch		2+1	4+1	6+7	63+1
Total # of links (bidi)	1	64+64	128+64	192+64	2016+64

IBM BlueGene

	512-node proto	BlueGene/L
Peak Perf	1.0 / 2.0 TFlops/s	180 / 360 TFlops/s
Memory Size	128 GByte	16 / 32 TByte
Foot Print	9 sq feet	2500 sq feet
Total Power	9 KW	1.5 MW
# Processors	512 dual proc	65,536 dual proc
Networks	3D Torus, Tree, Barrier	3D Torus, Tree, Barrier
Torus BW	3 B/cycle	3 B/cycle



Summary

- Flynn's classification of processors - SISD, SIMD, MIMD
  - Q1 – How do processors share data?
  - Q2 – How do processors coordinate their activity?
  - Q3 – How scalable is the architecture (what is the maximum number of processors)?
- Shared address multis – UMAs and NUMAs
  - Scalability of bus connected UMAs limited (< ~ 36 processors)
  - Network connected NUMAs more scalable
  - Interconnection Networks (INs)
    - fully connected, xbar
    - ring
    - mesh
    - n-cube, fat tree
- Message passing multis
- Cluster connected (NOWs) multis