Computer Architecture:
Fundamentals, Tradeoffs, Challenges

Chapter 13: Multi-thread Parallelism

Yale Patt
The University of Texas at Austin

Austin, Texas
Spring, 2023

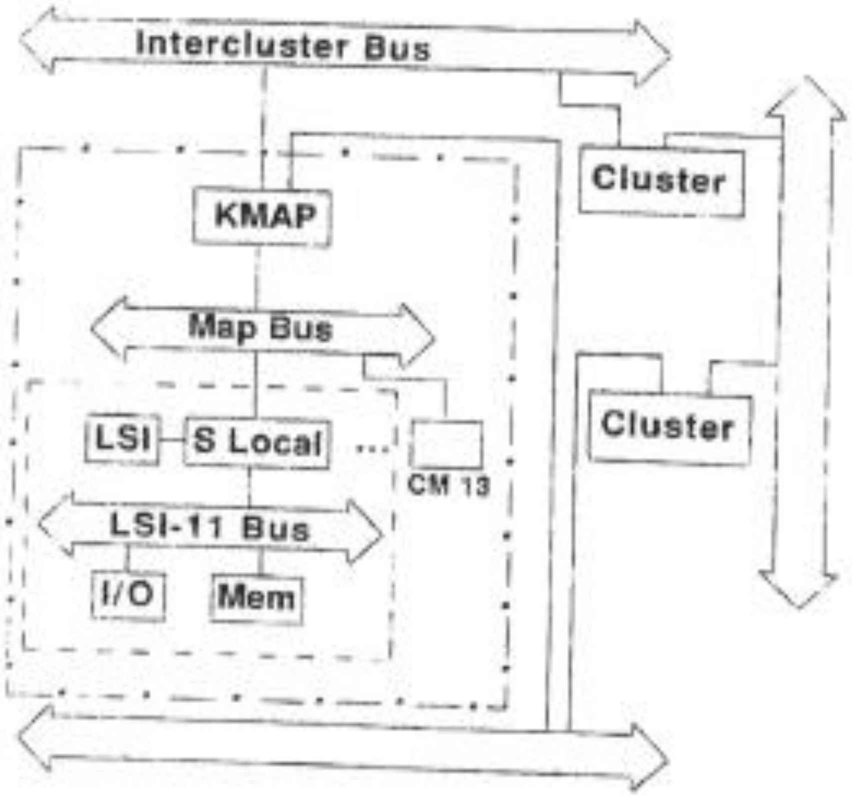
Outline

- ***A few examples***
- ***Some fundamentals***
  - ***Amdahl's Law***
  - ***Metrics (speedup, efficiency, redundancy, utilization)***
- ***Trade-offs***
  - ***Tightly coupled vs. Loosely coupled***
    - ***Shared Distributed vs Shared Centralized***
  - ***CMP vs SMT, and while we are at it: SSMT (aka helper threads)***
  - ***Heterogeneous vs. Homogeneous***
  - ***Interconnection networks***
- ***Cache Coherence***
- ***Memory Consistency***

A few examples

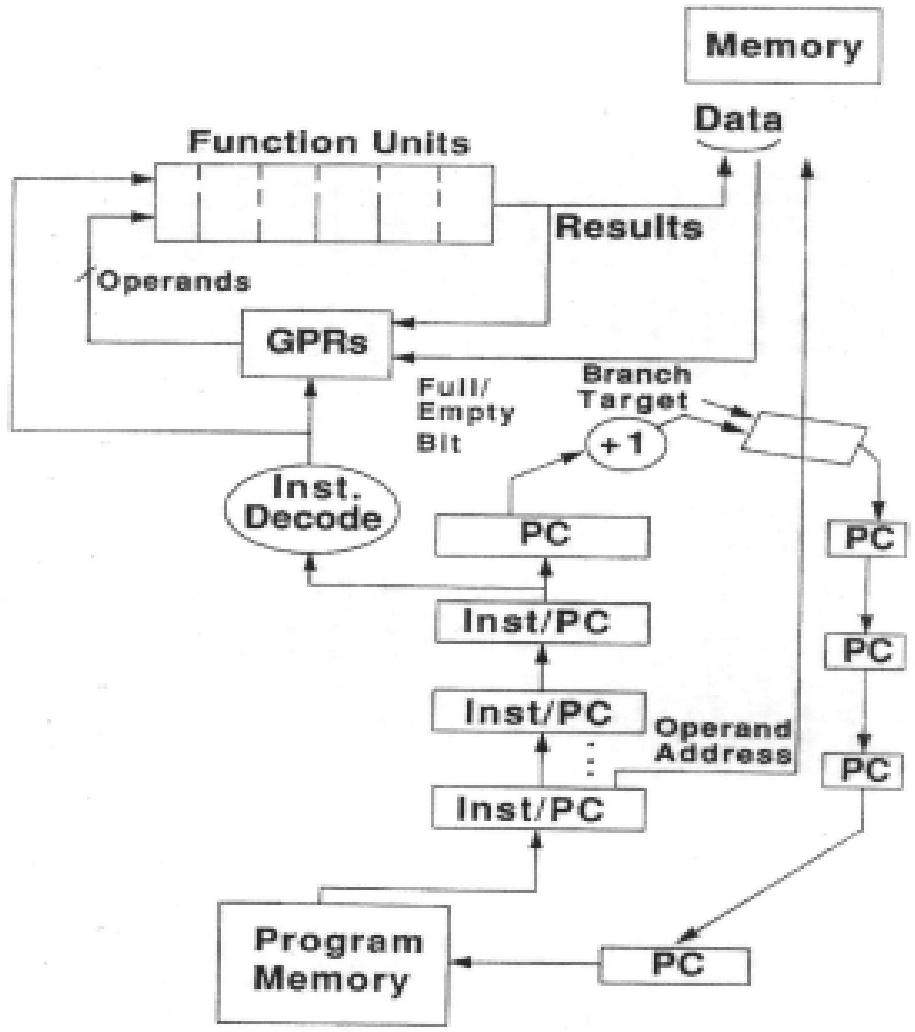
- ***Cm* (Carnegie Mellon, late 1970s***
- ***HEP (Burton Smith, late 1970s)***
- ***Cosmic Cube (Geoffrey Fox)***
- ***Classical GPU (Nvidia, for example)***

cm*

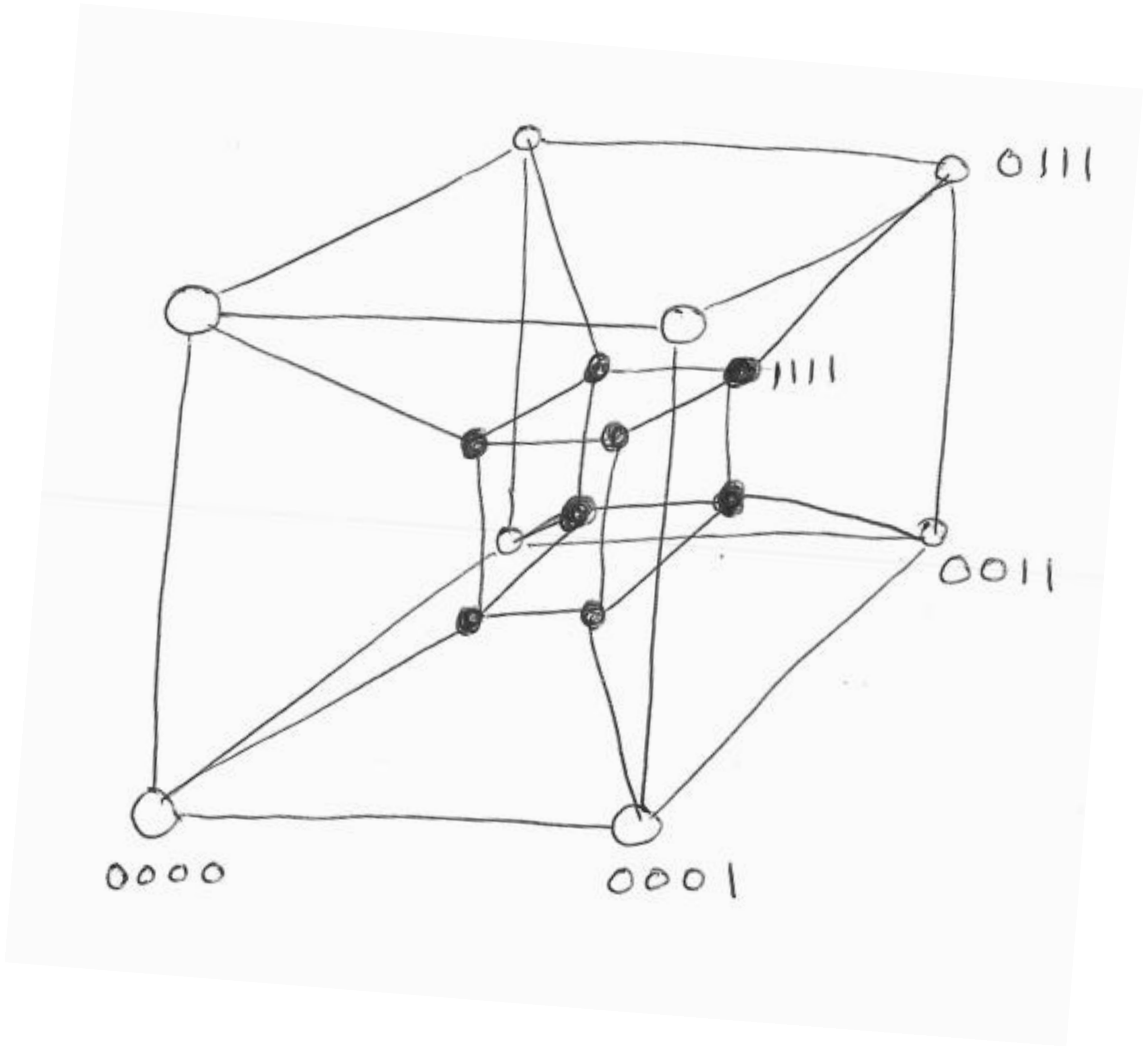


<u>Note</u>: ***A well-meaning student told me to get rid of this slide. cm* is old. People will think you are an old man, and not take you seriously.***

The HEP


Memory
Data
Function Units
Results
Operands
GPRs
Full/
Empty
Bit
Branch
Target
+1
Inst.
Decode
PC
Inst/PC
Inst/PC
Operand
Address
Inst/PC
PC
PC
PC
Program
Memory
PC

Cosmic Cube



Classical GPU

- ***Ten thousand threads not uncommon***
  - ***32 stream processors***
  - ***Each runs 32-way SIMD (with predication)***
  - ***10-stages, as in the HEP***
- ***Great! …unless***
  - ***Memory collisions (coalescing)***
  - ***Branch divergence***

Amdahl's Law

- ***Speed-up of an application running on many processors***
  - ***Depends on how many processors (p)***
  - ***Depends on how parallel the application is (alpha)***

- ***The Speed-up Equation***

$$S_p = \frac{T_1}{T_p} = \frac{T_1}{\frac{\alpha T_1}{p} + \frac{(1-\alpha) T_1}{1}} = \frac{1}{\frac{\alpha}{p} + (1-\alpha)}$$

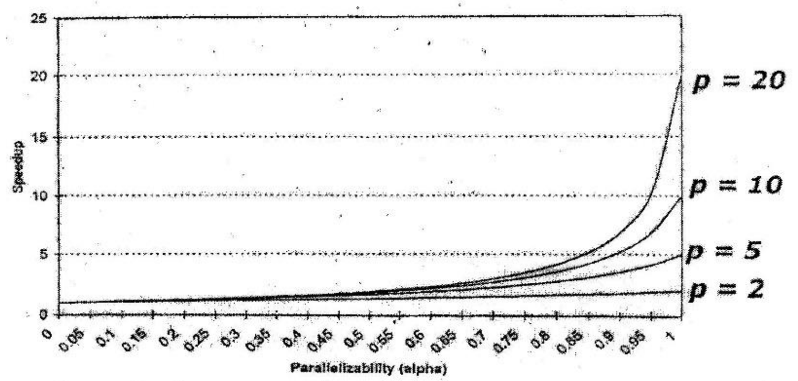
- ***The Serial Bottleneck always limits performance***
  - ***If alpha is .5, an infinite no. of processors only gives Speed-up of 2***
  - ***If alpha is .9, an infinite no. of processors only gives Speed-up of 10***

- ***BUT Heterogeneous cores can minimize the effect***

Amdahl's Law

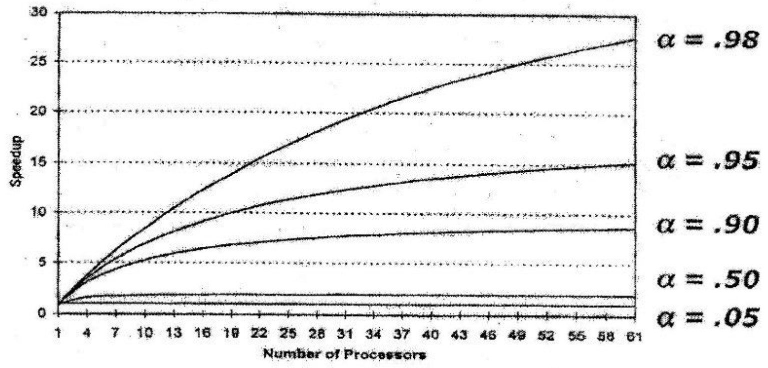
- Speed-up as a function of the parallelizability (α) of the application

Speedup vs. Parallelizability for a given number of processors (p)



- Speed-up of an application as we add more and more processors (p)

Speedup vs. Number of Processors (p) for a given alpha



An important observation from Bill Buzbee

- ***First, he recognized that p processors adds complexity***
  - ***Every system requires management activity, and its cost***
- ***Buzbee's Amdahl's Law equation:***

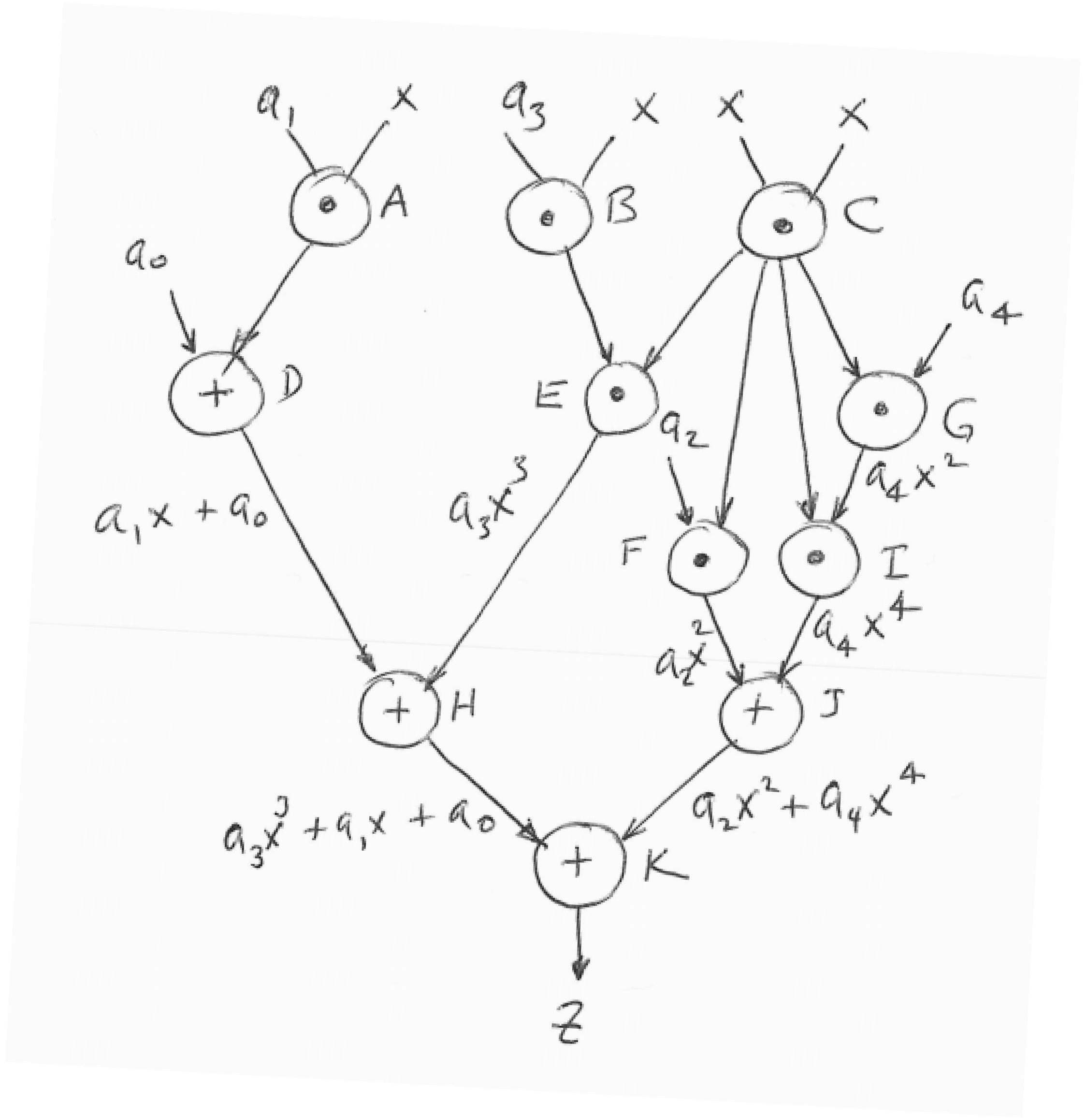
$$S_p = \frac{1}{\frac{\alpha}{p} + (1-\alpha) + \sigma(p)}$$

…where sigma is a function of the number of processors

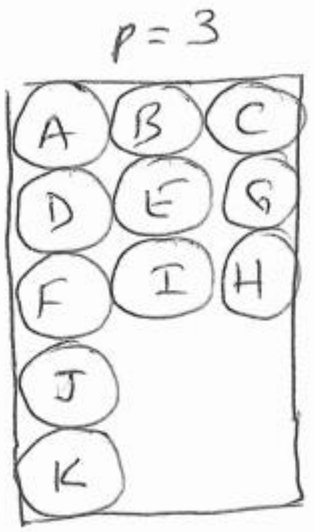
Metrics

- ***Speed-up: How much faster with p processors?***
- ***Efficiency: How many extra processors do you tie up?***
- ***Utilization: How much of it is actually used?***
- ***Redundancy: How much extra processing do you do?***

*An example: z = a4*x^4 + a3*x^3 + a2*x^2 + a1*x + a0*



Calculation of Speed-up



$T_3 = 5$

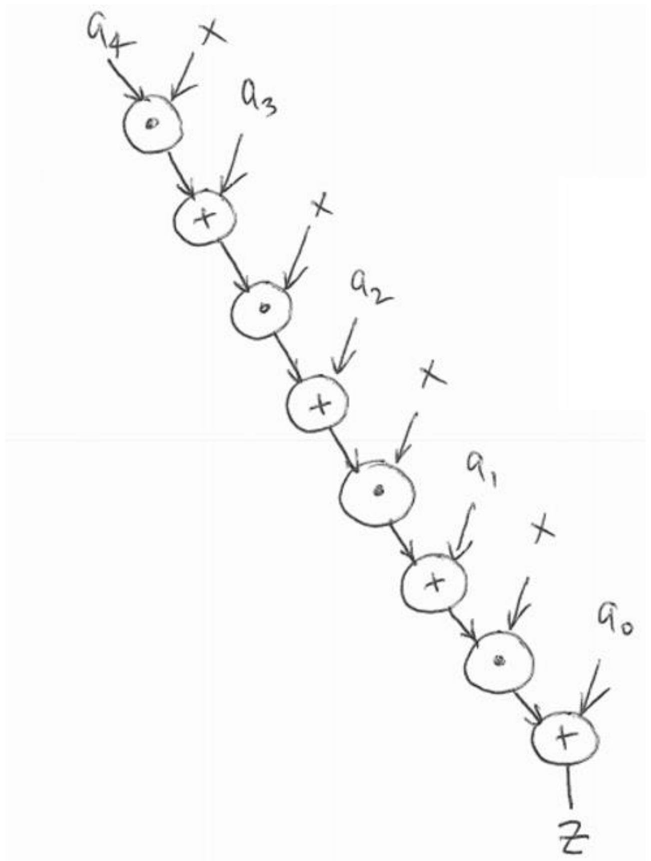
$$S_3 = \frac{T_1}{T_3} = \frac{11}{5} = \underline{2.2}$$

- ***Correct?*** ***NO!***

Why Not?

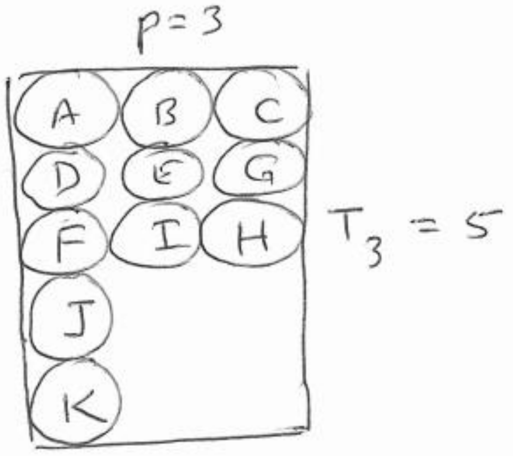
Definition of Speedup

- ***Speed-up with p processors = T1 divided by Tp,***
  - ***where T1 is the best one processor solution!***



- ***With Horner's Rule, T1 = 8, not 11***
  - ***Therefore, speedup = 8/5, or 1.6, not 11/5, or 2.2.***

Calculation of Efficiency, Utilization, Redundancy



$$\text{Efficiency} = \frac{1 \cdot T_1}{p \cdot T_p} = \frac{8}{3 \cdot 5} = \frac{8}{15}$$

$$\text{Utilization} = \frac{O_p}{p\, T_p} = \frac{11}{3 \cdot 5} = \frac{11}{15}$$

$$\text{Redundancy} = \frac{O_p}{O_1} = \frac{11}{8}$$

Tradeoffs

- ***Tightly coupled vs. loosely coupled***
- ***Distributed Shared memory vs. Centralized Shared***
- ***CMP (aka multicore) vs. SMT (and SSMT)***
- ***One Supercomputer vs. many light-weight cores***
- ***Homogeneous vs Heterogeneous***
- ***Interconnection networks***

Tightly-coupled vs Loosely-coupled

- ***Tightly coupled (i.e., Multiprocessor)***
  - ***Shared global memory (centralized or distributed)***
    (an early example of shared distributed: cm*)
  - ***Each processor capable of doing work on its own***
    - ***8086 and 8087 coprocessor is NOT a multiprocessor***
  - ***Easier for the software***
  - ***Hardware has to worry about cache coherency, memory contention***

- ***Loosely-coupled (i.e., Multicomputer Network)***
  - ***Message passing***
  - ***Easier for the hardware***
  - ***Programmer's job is tougher***

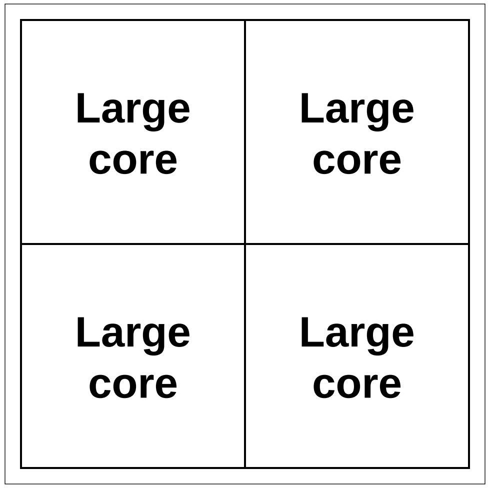
CMP vs SMT

- ***CMP (chip multiprocessor)***
  - ***aka Multi-core***
- ***SMT (simultaneous multithreading)***
  - ***One core – a PC for each thread, in a pipelined fashion***
  - ***Multithreading introduced by Burton Smith, HEP ~1975***
  - ***"Simultaneous" first proposed by H.Hirata et.al, ISCA 1992***
  - ***Carried forward by Nemirovsky, HICSS 1994***
  - ***Popularized by Tullsen, Eggers, ISCA 1995***
- ***One can do both on same chip (IBM POWER ~2004)***

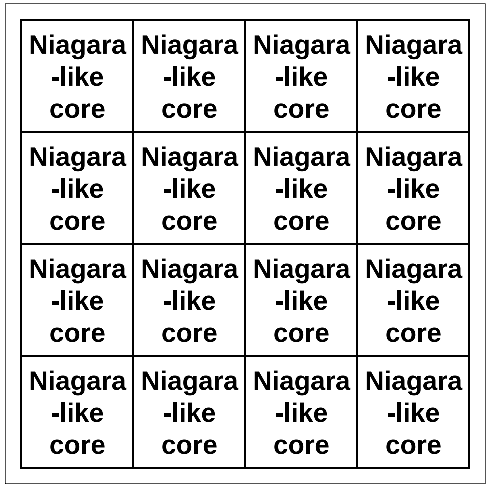
HPC with one heavyweight or many lightweights

- ***A Choice: (1 times 2^n), (2^k times 2^n-k), (2^n times 1)***
  - ***Not really a choice anymore***
- ***Many reasons for championing HPC***
  - ***It is like a better microscope or better telescope***
- ***Scalability***
  - ***SIMD is easy (The General barks an order to his troops)***
  - ***MIMD is very hard***
  - ***Massive SIMD has been around since the 1980s***
    - ***Cm1 from Thinking Machines (2^16 cores)***
    - ***Non-Von from D.E.Shaw (2^20 cores)***
    - ***Boolean Vector Machine (2^30 cores)***

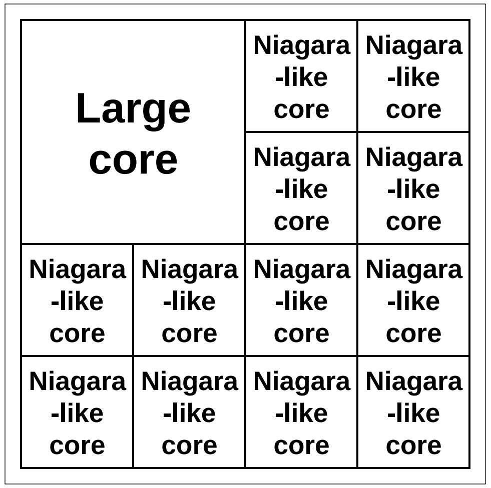
Heterogeneous vs Homogeneous



“Tile-Large” Approach

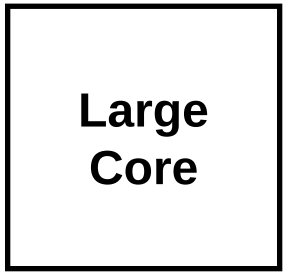


“Niagara” Approach



ACMP Approach

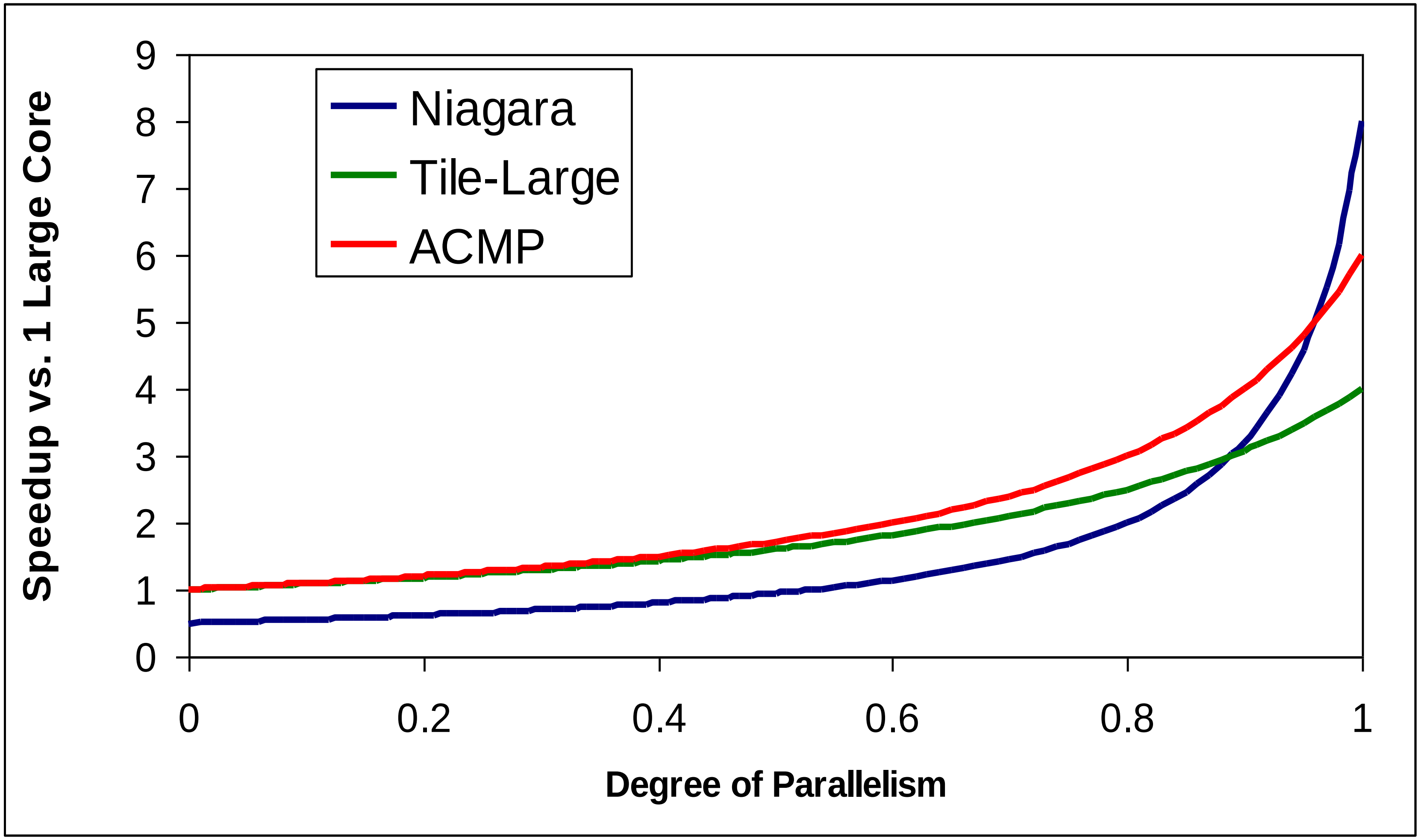
Large core vs. Small Core



- ***Out-of-order***
- ***Wide fetch e.g. 4-wide***
- ***Deeper pipeline***
- ***Aggressive branch predictor (e.g. hybrid)***
- ***Many functional units***
- ***Trace cache***
- ***Memory dependence speculation***

- ***In-order***
- ***Narrow Fetch e.g. 2-wide***
- ***Shallow pipeline***
- ***Simple branch predictor (e.g. Gshare)***
- ***Few functional units***

Throughput vs. Serial Performance



Interconnection Networks

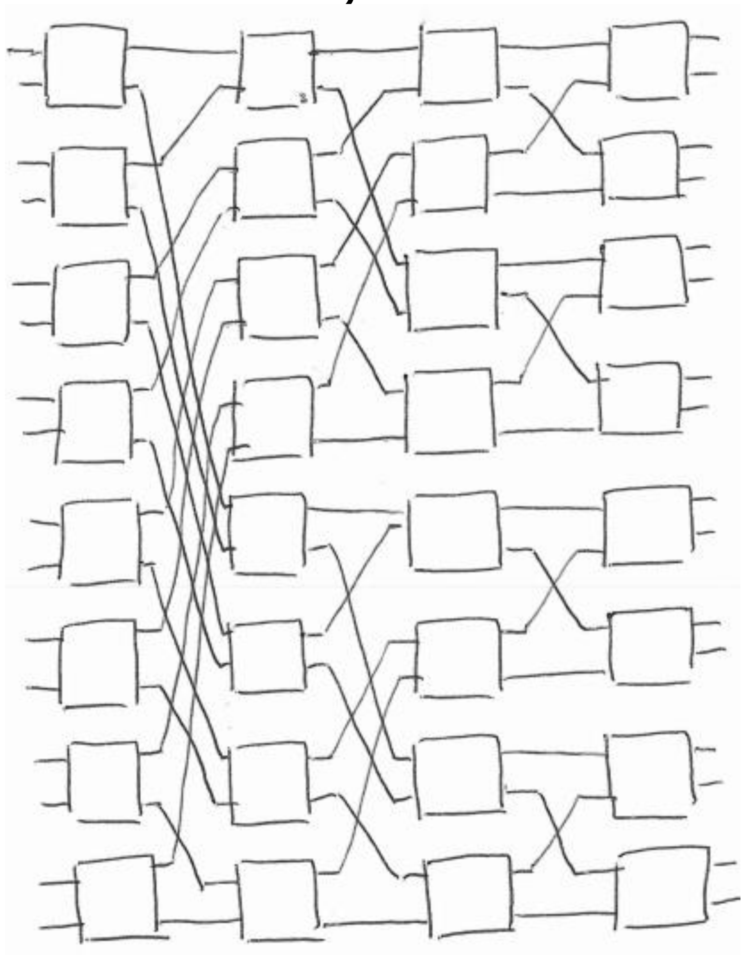
- ***Parameters: Latency, Cost, Contention***

- ***Basic Topologies***

Basic Topologies	Latency	Cost	Contention
Bus	1	O (n)	Worst
Crossbar	1	O (n^2)	Best
Omega Network	Logk(n)	O (nklogk(n))	
Hypercube	Log2(n)	O (nlogn)	
Tree	Log2(n)	O (n)	
Mesh	sq.root(n)	O (n)	
Ring	n/2	O (n)	

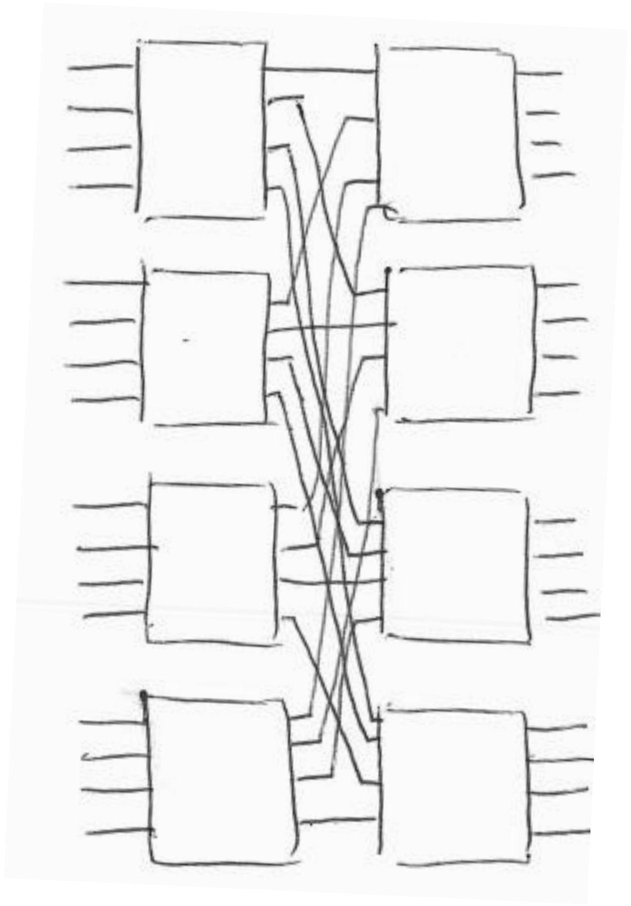
Example Omega Networks (constructed from k by k crossbar switches)

N=16, k=2



32 2x2 crossbars

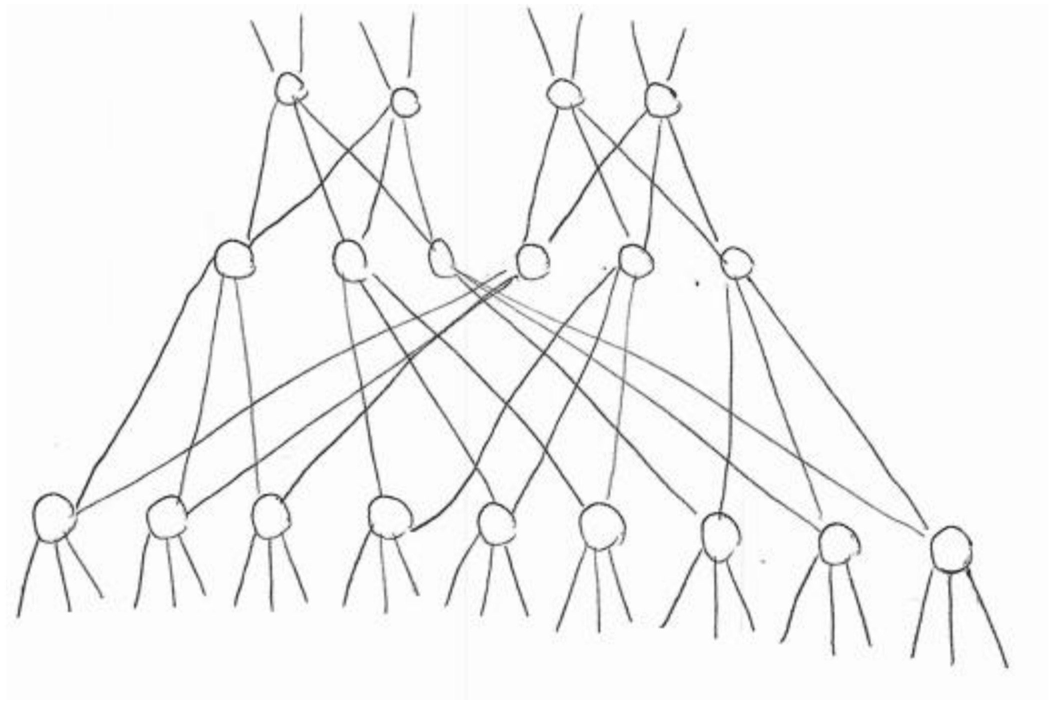
N=16, k=4



8 4x4 crossbars

Banyan Trees

- ***A variation on Omega Networks***
  - ***Here at UT (Prof Jack Lipovski, ECE; Prof Jim Browne, CS)***
  - ***Constructed from m levels of j by k crossbar switches***
  - ***Like Omega networks, a unique path for each A to each B***
- ***Example:***
  - ***j: 2***
  - ***k: 3***
  - ***m: 3***
  - ***J^m Processors***
  - ***K^m Memories***



Cache Coherence

- ***Definition:***

  If a cache line is present in more than one cache,

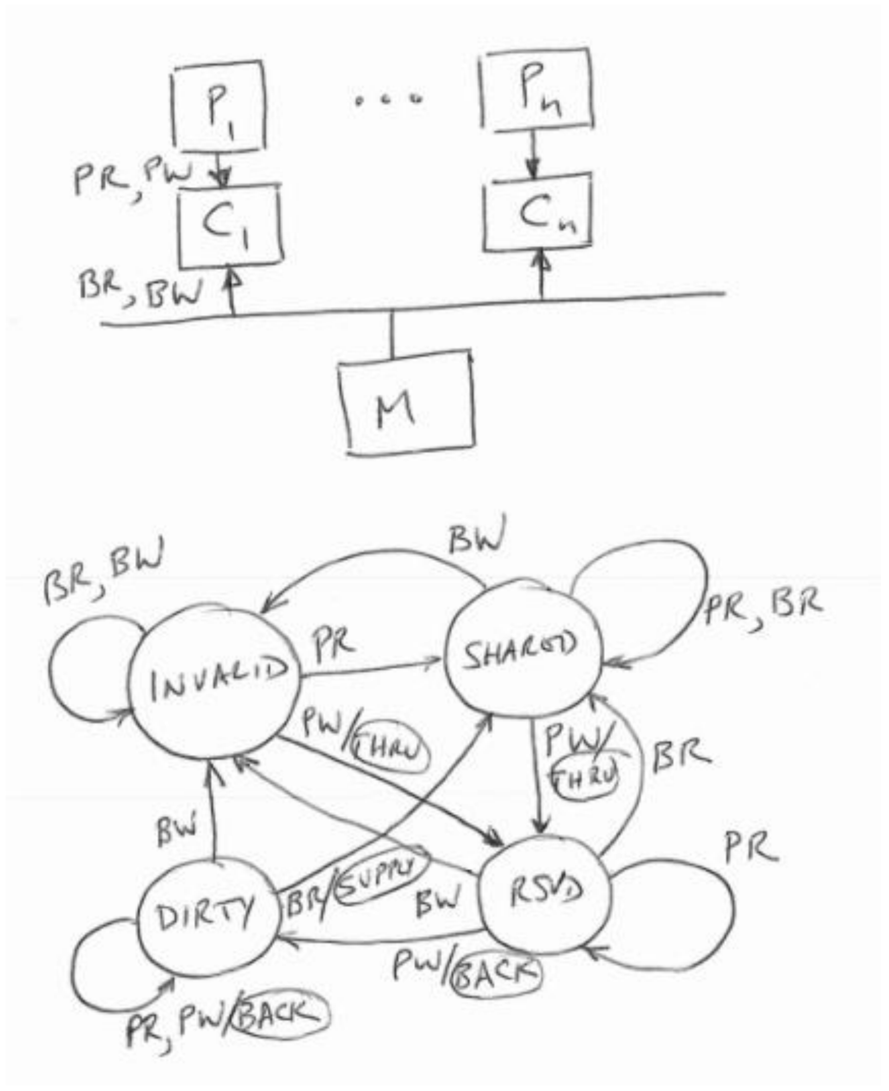
  it has the same values in all caches

- ***Why is it important?***

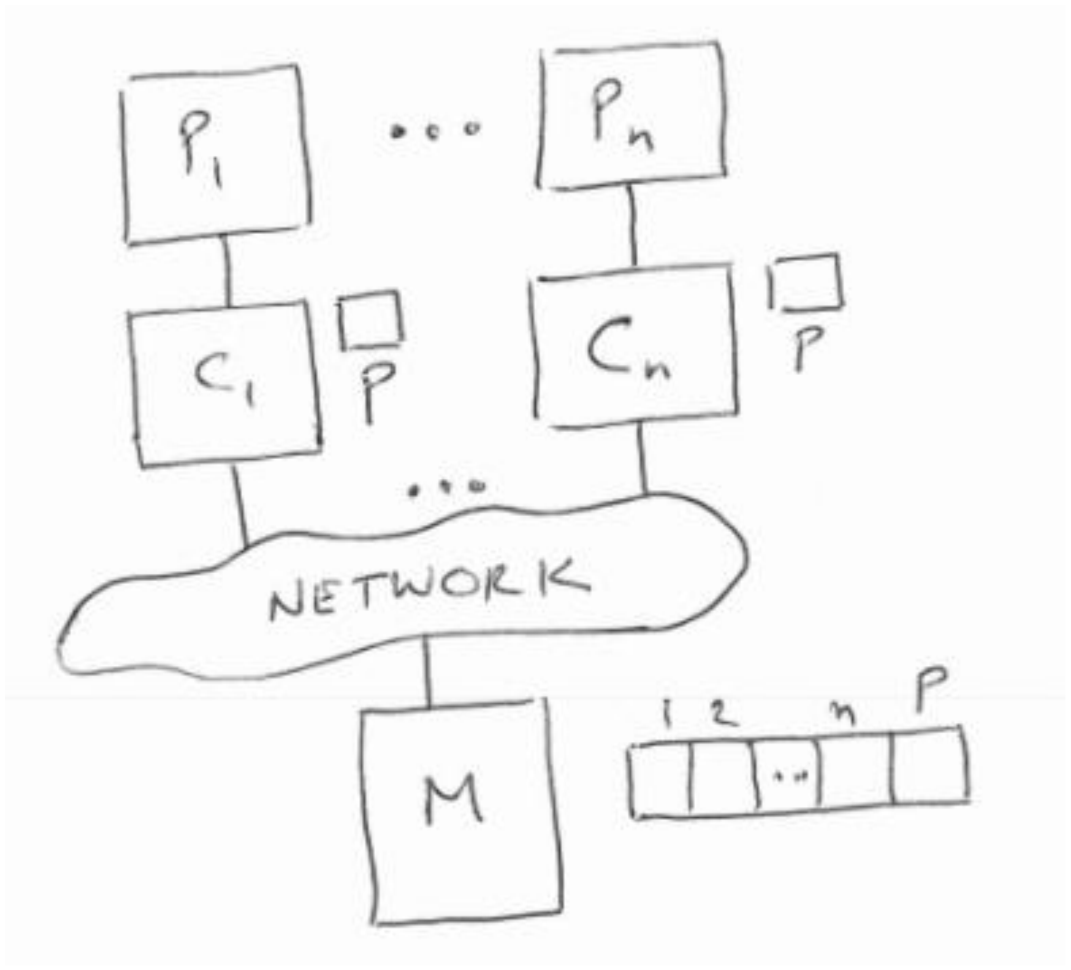
- ***Snoopy schemes***
  - ***All caches snoop the common bus***

- ***Directory schemes***
  - ***Each cache line has a directory entry in memory***

A Snoopy Scheme



Directory Scheme (p=n)



Sequential Consistency

- ***Definition: Memory sees loads/stores in program order.***
- ***Why it is important: It guarantees mutual exclusion.***

Two Processors and a critical section

```
Processor 1                         Processor 2
    L1=0                                L2=0

A:  L1 = 1                          X:  L2=1
B:  If L2 =0                        Y:  If L1=0
      {critical section}                  {critical section}
C:  L1=0                            Z:  L2=0
```

What can happen?

Order of A,B,C,X,Y,Z obeying seq. consistency (Shown are the ten starting with A, There are also ten starting with X)

A	A	A	A	A	A	A	A	A	A
B	B	B	B	X	X	X	X	X	X
C	X	X	X	B	B	B	Y	Y	Y
X	C	Y	Y	C	Y	Y	B	B	Z
Y	Y	C	Z	Y	C	Z	C	Z	B
Z	Z	Z	C	Z	Z	C	Z	C	C

Which processors get exclusive access to the critical section under each order?

1,2   1,2   1   1   2   None   None   None   None   1

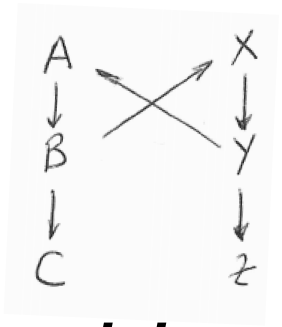
Note: 1,2 means first 1 gets exclusive access, and when done, 2 gets access.

Critical Sections require Mutual Exclusion

- ***A critical section: a region of memory containing data wherein only one thread can be allowed access at a time.***
- ***Why is this important?***
  - ***P1 program wants to execute LD R1,A followed by ADD A,A,R1***
  - ***Between the two instructions, suppose P2 executes ST R2,A***
  - ***The bad result: A contains A + R2 instead of A + A.***
- ***Mutual Exclusion Property***
  - ***Only one processor at a time can access the critical section***
- ***What happens if B occurs before X, and Y occurs before A?***
  - ***If B occurs before X, Process 1 can access the memory***
  - ***If Y occurs before A, Process 2 can access the memory***
  - ***No mutual exclusion!***

Sequential Consistency Guarantees Mutual Exclusion

- ***Sequential consistency means all memory accesses reach the memory system in program order. That is, A occurs before B, B before C, and X before Y, Y before Z.***
- ***Both processors accessing shared memory concurrently means B must occur before X and Y must occur before A.***
- ***That is, sequential consistency AND concurrent access only if A before B before X before Y before A. Impossible!***



- ***Therefore, having sequential consistency guarantees no concurrent access, i.e., mutual exclusion.***

Bedankt!