

# L22-CS8421-Parallel

## Intro Parallel Architectures

### CS8421

### Computing Systems

### Dr. Ken Hoganson

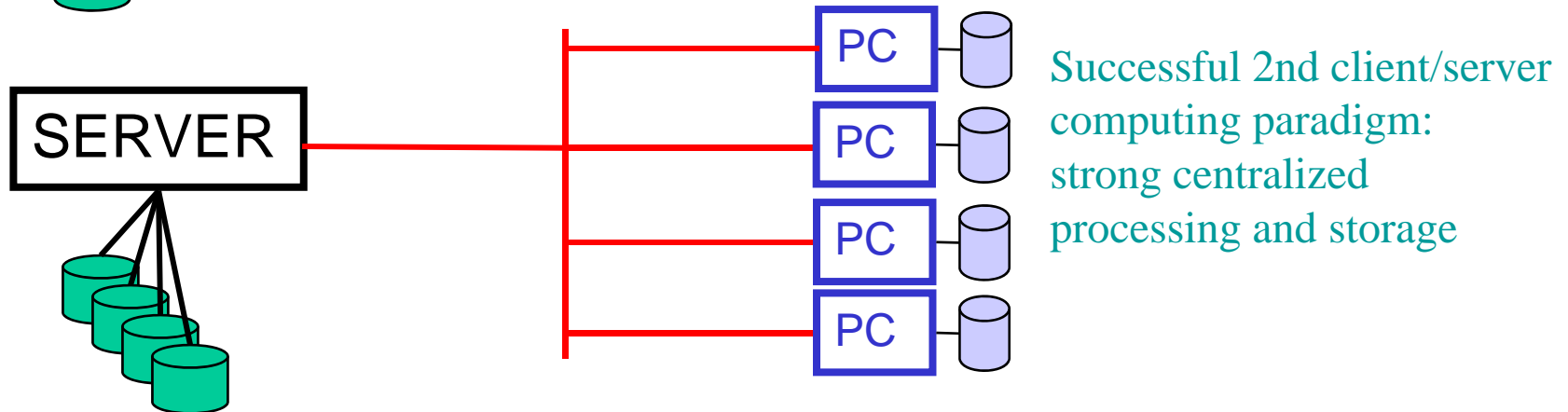
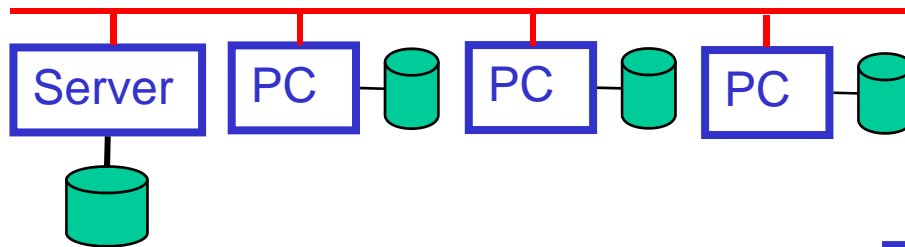
*Class*

*Will*

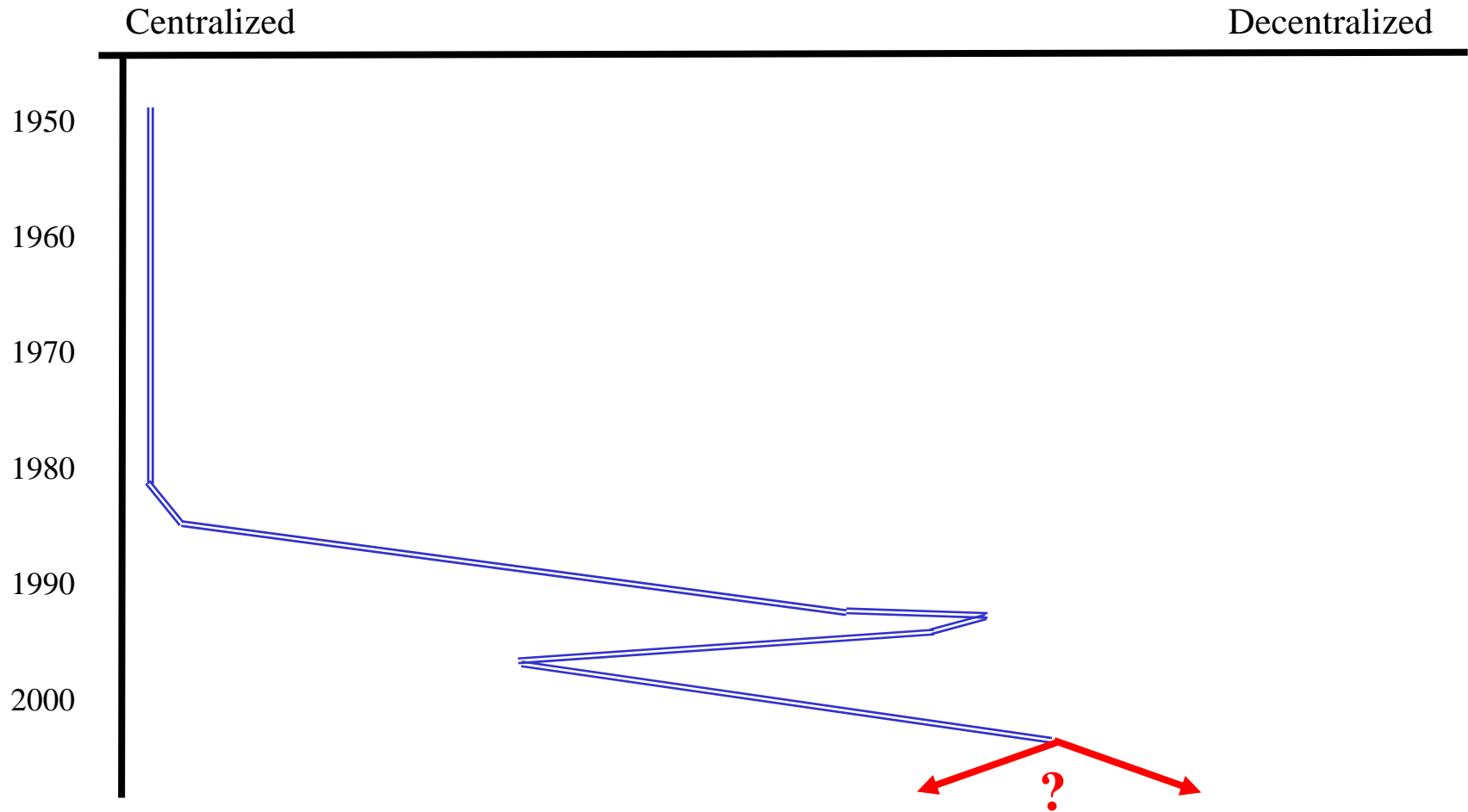
*Start*

*Momentarily...*

- Mission-critical
  - High reliability
  - redundancy
- Massive storage (disk)
  - RAID for redundancy
- High performance through replication of components
  - Multiple processors
  - Multiple buses
  - Multiple hard drives
  - Multiple network interfaces



## Processing and Storage



- Client/server architecture was originally predicted to bring about the demise of the mainframe.
- Critical corporate data must reside on a highly reliable high performance machine
- Early PC networks did not have the needed performance or reliability
  - NOW (Network Of Workstations)
  - LAN (Local Area Network)
- Some firms, after experience with client/server problems, returned to the mainframe for critical corporate data and functions
- Modern computing paradigm combines
  - powerful servers (including mainframes when needed) where critical corporate data and information resides
  - With decentralized processing and non-critical storage on PCs
  - Interconnected with a network

- Multiprocessor servers offer high performance at much lower cost than a traditional mainframe
- Uses inexpensive, “off-the-shelf” components
- Combine multiple PCs or workstations in one box
- Processors cooperate to complete the work
- Processors share resources and memory
- One of the implementations of Parallel Processing

## 5 levels of parallelism have been identified

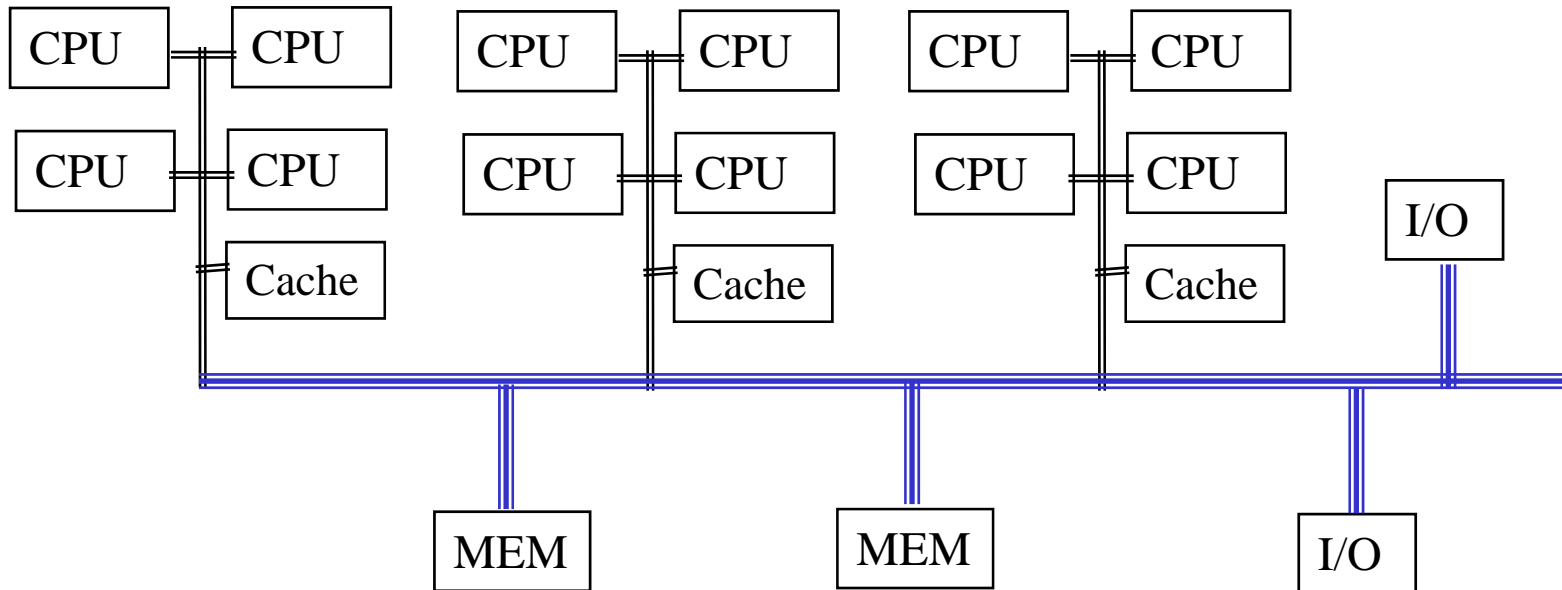
Each level has both a software level parallelism, and a hardware implementation that accommodates or implements the software parallelism

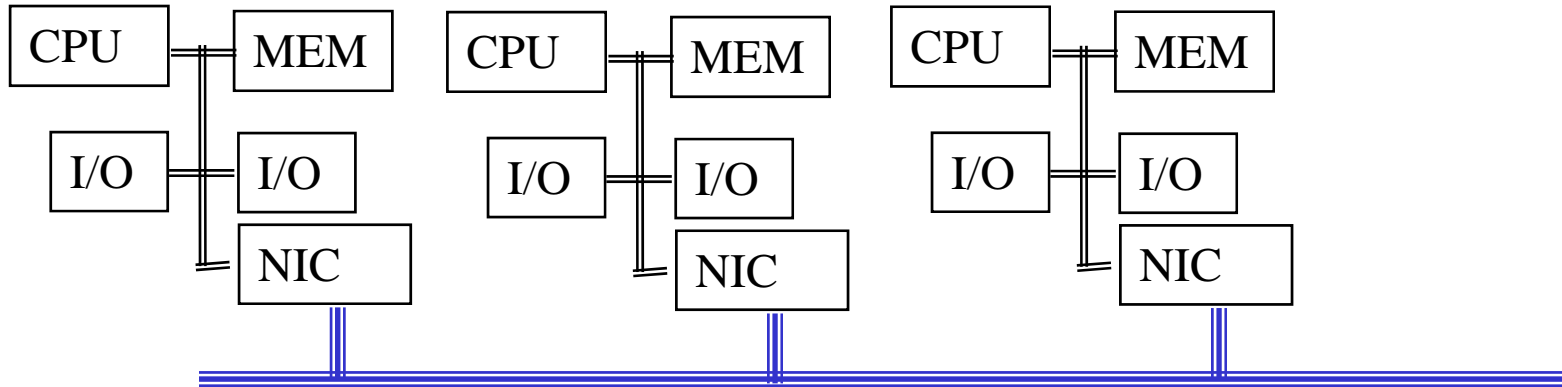
Sources:

- The Unified Parallel Speedup Model and Simulator, K. Hoganson, SE-ACM 2001, March 2001
- Alternative Mechanisms to Achieve Parallel Speedup, K. Hoganson, First IEEE Online Symposium for Electronics Engineers, IEEE Society, November 2000.
- Workload Execution Strategies and Parallel Speedup on Clustered Computers, K. Hoganson, IEEE Transactions on Computers, Vol. 48, No. 11, November 1999.

	<b>Software</b>	<b>Hardware Implementation</b>
1	Intra-Instruction	Pipeline
2	Inter-Instruction	Super-Scalar, multiple pipelines
3	Algorithm/Thread/Object	MultiProcessor
4	Multi-Process	Clustered-Multiprocessor
5	Distributed/N-Tier CS	Multicomputer/Internet/Web

- **Thread** - a lightweight process, easy (efficient) to multi-task between.
- **Multiprocessor** - a computer system with multiple processors combined in a single system (in a single box or frame). Usually share memory and other resources between processors.
- **Multicomputer** - multiple discrete computers with separate memory and etc. Interconnected with a network.
- **Clustered computer** - a multiprocessor OR multicomputer that builds two levels of interconnection between processors
  - **Intra-Cluster** connection (within cluster)
  - **Inter-Cluster** connection (between clusters)
- **Distributed Computer** - a loosely coupled multicomputer
  - a n-Tiered Client/Server computing system is an example of distributed computing





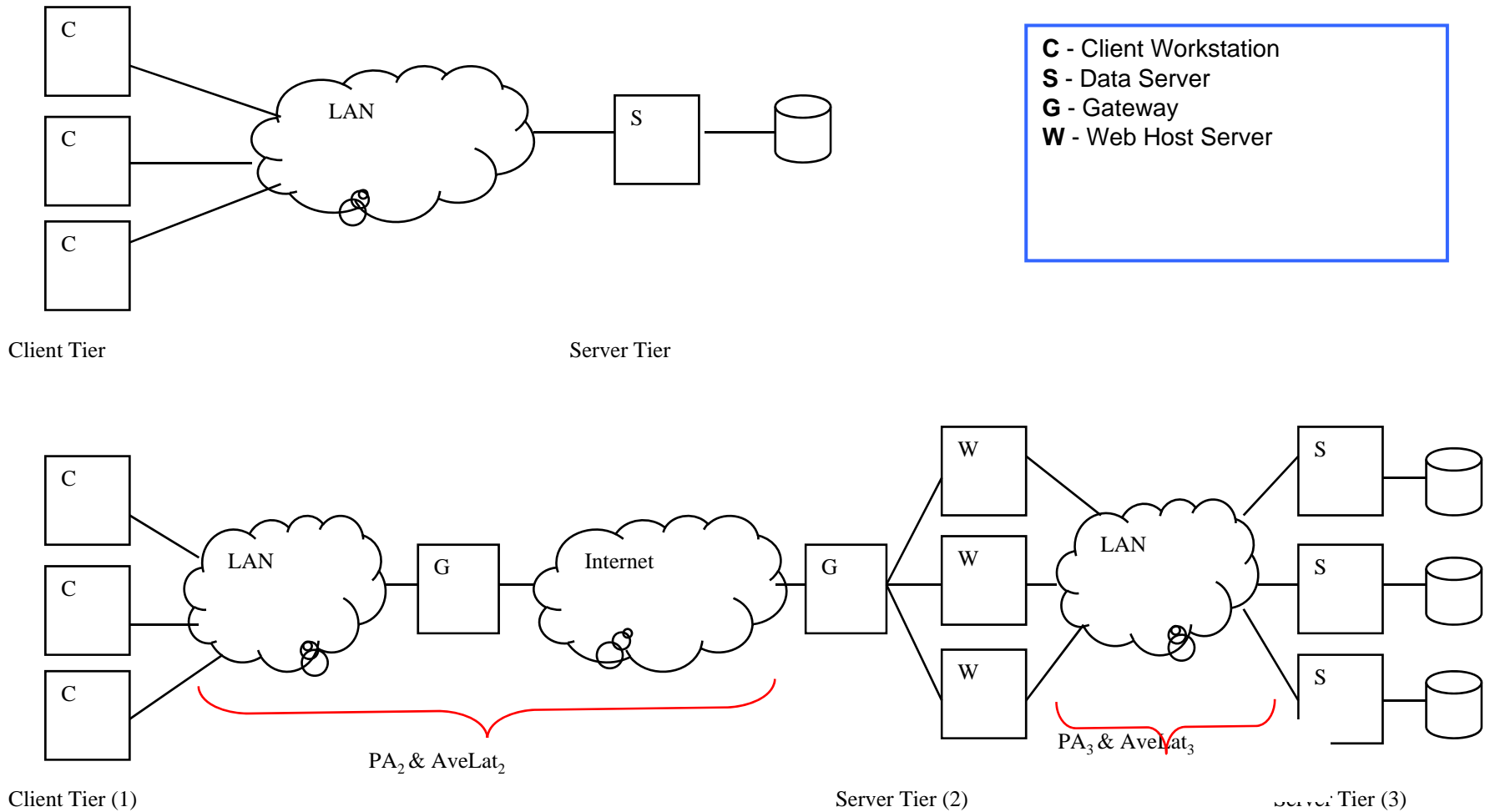


Figure 2. N-Tier Architectures

- Old idea, still useful.
- Examines parallelism from the point of view of what is the parallel scope of an instruction
- SISD - Single Instruction, Single Data: Each instruction operates on a single data item
- SIMD - Single Instruction, Multiple Data: Each instruction operates on multiple data items simultaneously (classic supercomputing)
- MIMD - Multiple Instruction, Multiple Data: Separate Instruction/Data streams. Super-scalar, multiprocessors, multicomputers.
- MISD - Multiple Instruction Single Data: No known examples

- Asymmetric Multiprocessing:
  - multiple unique processors, each dedicated to a special function
  - PC is an example
- Symmetric Multiprocessing:
  - multiple identical processors able to work together on parallel problems
- Homogenous system: a symmetric multiprocessor
- Heterogenous system: different “makes” of processors combined in a system. Example: distributed system with different types of PCs with different processors

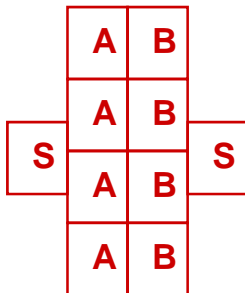
An example parallel process of time 10:

- S** - Serial or non-parallel portion
  - A** - All A parts can be executed concurrently
  - B** - All B parts can be executed concurrently
- All **A** parts must be completed prior to executing the **B** parts

Executed on a single processor:



Executed in parallel on 4 processors:



- Multiple Processors available (4)
- A Process can be divided into serial and parallel portions
- The parallel parts are executed concurrently
- **Serial Time: 10 time units**
- **Parallel Time: 4 time units**

$$Speedup = \frac{SerialTime}{ParallelTime} = \frac{10}{4} = 2.5$$

$$Efficiency = \frac{Speedup}{processors} = \frac{2.5}{4} = 62.5\%$$

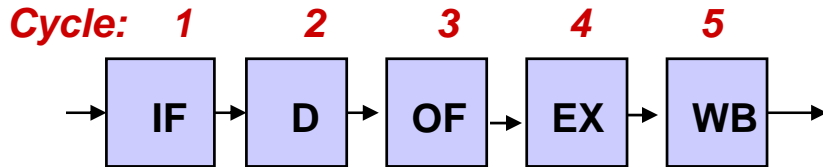
$$\text{Speedup} = \frac{\text{SerialTime}}{\text{ParallelTime}} =$$

$$= \frac{1}{(1 - \alpha) + \frac{\alpha}{n}}, \quad \text{where}$$

$\alpha$  = fraction of work that can be done in parallel

$n$  = number of processors

- Analytical model of parallel speedup from 1960s
- Parallel fraction ( $\alpha$ ) is run over  $n$  processors taking  $\alpha/n$  time
- The part that must be executed in serial ( $1 - \alpha$ ) gets no speedup
- Overall performance is limited by the fraction of the work that cannot be done in parallel ( $1 - \alpha$ )
- diminishing returns with increasing processors ( $n$ )



- F** - Instruction Fetch
- D** - Instruction Decode
- OF** - Operand Fetch
- EX** - Execute
- WB** - Write Back or Result Store

Processor clock/cycle is divided into sub-cycles, each stage takes one sub-cycle

- **Single Processor enhanced with discrete stages**
- **Instructions “flow” through pipeline stages**
- **Parallel Speedup with multiple instructions being executed (by parts) simultaneously**
- **Realized speedup is partly determined by the number of stages: 5 stages=at most 5 times faster**

- Speedup is serial time ( $nS$ ) over parallel time
- Performance is limited by the number of pipeline flushes ( $n$ ) due to jumps
- speculative execution and branch prediction can minimize pipeline flushes
- Performance is also reduced by pipeline stalls ( $s$ ), due to conflicts with bus access, data not ready delays, and other sources

$S$  = Number of pipeline STAGES

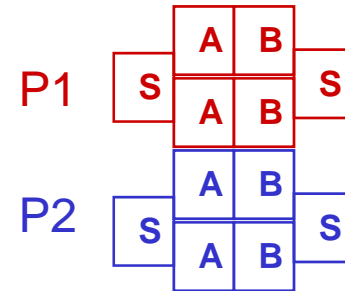
$n$  = Average number of instructions between pipeline flushes

$s$  = frequency of pipeline stalls (%)

$$Speedup = \frac{nS}{S + n - 1 + ns}$$

- Concurrent Execution of Multiple sets of instructions
- Example: Simultaneous execution of instructions though an integer pipeline while processing instructions through a floating point pipeline
- Compiler: identifies and specifies separate instruction sets for concurrent execution through different pipes

- Parallel “threads of execution”
  - could be a separate process
  - could be a multi-thread process
- Each thread of execution obeys Amdahl’s parallel speedup model
- Multiple concurrently executing processes resulting in:
- Multiple serial components executing concurrently - another level of parallelism



Observe that the serial parts of Program 1 and Program 2 are now running in parallel with each other.

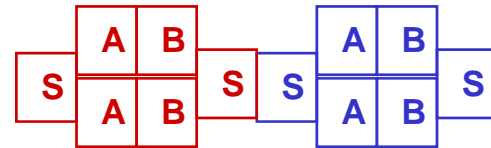
Each program would take 6 time units on a uniprocessor, or a total workload serial time of 12. Each has a speedup of 1.5.

The total speedup is  $12/4 = 3$ , which is also the sum of the program speedups.

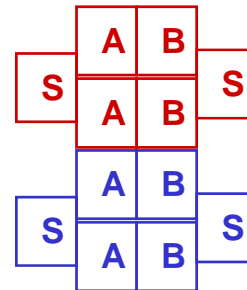
- Concurrent Execution of Multiple Processes
- Each process is limited by Amdahl's parallel speedup
- Multiple concurrently executing processes resulting in:
- Multiple serial components executing concurrently - another level of parallelism
- Avoid Degree of Parallelism (DOP) speedup limitations
- Linear scaling up to machine limits of processors and memory:  $n \times$  single process speedup



No speedup - uniprocessor **12 t**



Single Process **8 t**, Speedup = 1.5



Multi-Process **4 t**, Speedup = 3

## Multi-Process/Thread Speedup

$\alpha$  = fraction of work that can be done in parallel

$n$  = number of processors

$N$  = number concurrent (assumed similar) processes or threads

$$Speedup = N \times \left( \frac{1}{(1-\alpha) + \frac{\alpha}{n}} \right)$$

## Multi-Process/Thread Speedup

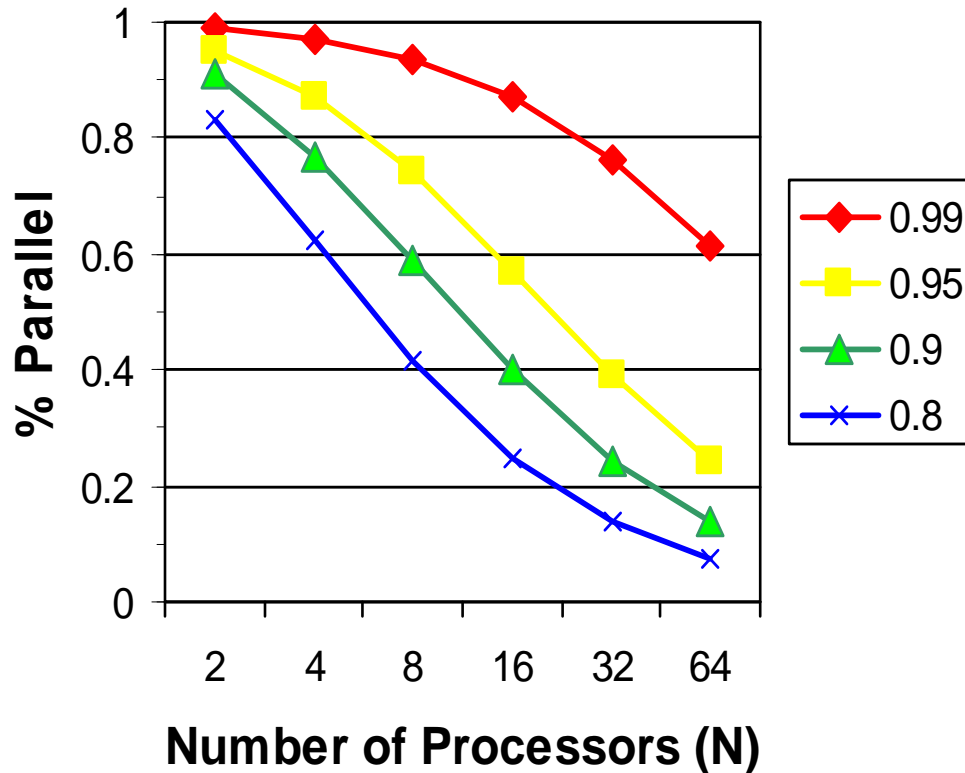
$\alpha$  = fraction of work that can be done in parallel

$n$  = number of processors

$N$  = number concurrent (assumed dissimilar) processes or threads

$$Speedup = \sum_{i=1}^N \frac{1}{(1-\alpha) + \frac{\alpha}{n}}$$

## Efficiency: Speedup/N



- Most parallelism suffers from diminishing returns - resulting in limited scalability.
- Allocating hardware resources to capture multiple levels of parallelism - operate at efficient end of speedup curves.
- Manufacturers of microcontrollers are integrating multiple levels of parallelism on a single chip

**End  
Of  
Today's  
Lecture.**

