Challenges to Obtaining Good Parallel Processing Performance

- **Outline:**
  - **Coverage** or extent of parallelism in algorithm
    - Amdahl's Law
  - **Granularity** of partitioning among processors
    - Communication cost and load balancing
  - **Locality** of computation and communication
    - Communication between processors or between processors and their memories

- **Coverage: The Parallel Processing Challenge of Finding Enough Parallelism**

  Amdahl's Law:

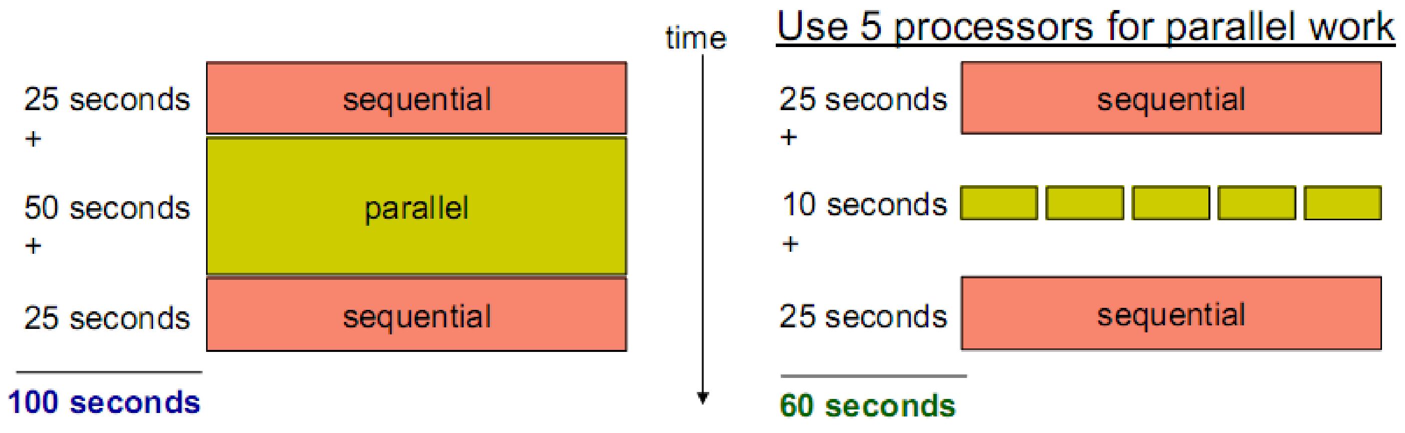
  - **The parallel speedup of any program is limited by the time needed for any sequential portions of the program to be completed.**
  - **For example, if a program runs for ten hours on a single core and has a sequential (nonparallelizable) portion that takes one hour, then no matter how many cores are devoted to the program, it will never go faster than one hour.**

- Amdahl's law: If s is the execution time for inherently sequential computations, the speedup is limited by

$$speedup(p) = \frac{time(1)}{time(p)} = \frac{time(1)}{s + (parallel\_time(p))} \leq \frac{time(1)}{s}$$

- If 20% of the sequential execution time is in sequential regions, the speedup is limited to 5 independent of the number of processors.

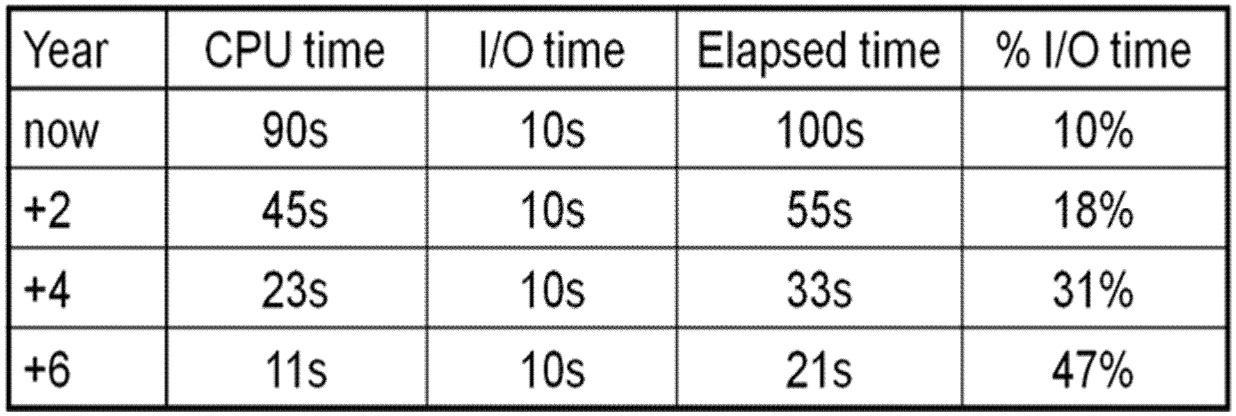
➢ **Even if parallel part speeds up perfectly, performance is limited by sequential part**



- Speedup = old running time / new running time
  = 100 seconds / 60 seconds
  = 1.67
  (parallel version is 1.67 times faster)

- **Granularity: The Parallel Processing Challenge of Overhead caused by Parallelism**
    - **Given enough parallel work, this is the biggest barrier to getting desired speedup**
    - **Parallelism overheads include:**
        - **Cost of starting a thread or process**
        - **Cost of communicating shared data**
        - **Cost of synchronizing**
    - **Each of these can cost several milliseconds (=millions of flops) on some systems**
    - **Tradeoff: Algorithm needs sufficiently large units of work to run fast in parallel (i.e. large granularity), but not so large that there is not enough parallel work.**

- **I/O Time vs. CPU Time**
    - **Input/Output Time includes both the Memory System and Bus/Network System**
    - **The rate of improvement of I/O is much slower than that of the CPU**



Year	CPU time	I/O time	Elapsed time	% I/O time
now	90s	10s	100s	10%
+2	45s	10s	55s	18%
+4	23s	10s	33s	31%
+6	11s	10s	21s	47%

- **Exponentially growing gaps are occurring between:**
  - **Floating point time (CPU processing speed) and**
  - **Memory BandWidth (Transmission Speed of Memory) and**
  - **Memory Latency (Startup Time of Memory Transmission)**

Floating Point Time << 1/Memory Bandwidth << Memory Latency Time

	Annual increase	Typical value in 2006
Single-chip floating-point performance	59%	4 GFLOP/s
Memory bus bandwidth	23%	1 GWord/s = 0.25 word/flop
Memory latency	5.5%	70 ns = 280 FP ops = 70 loads

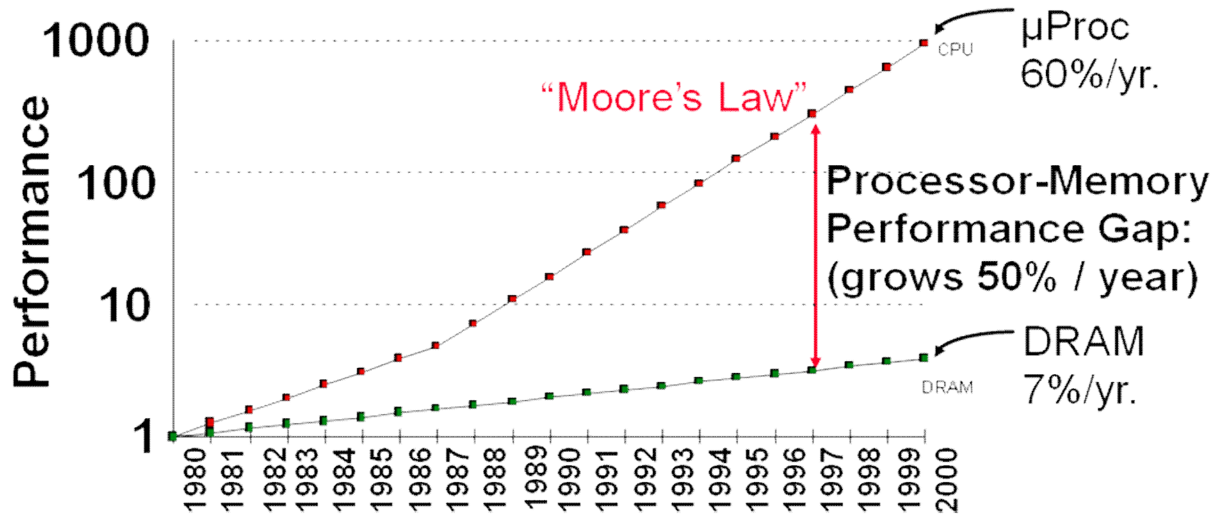
- **Exponentially growing gaps are also occurring between:**
  - **Floating point time (CPU processing speed) and**
  - **Network BandWidth (Transmission Speed of Network) and**
  - **Network Latency (Startup Time of Network Transmission)**

Floating Point Time << 1/Network Bandwidth << Network Latency Time

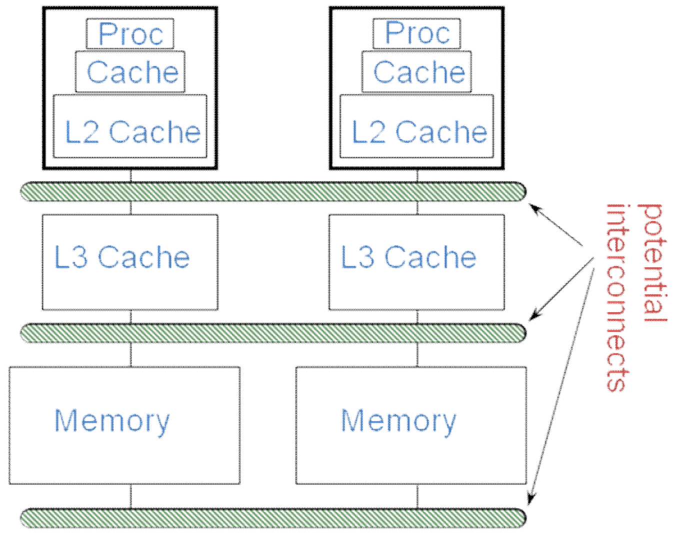
	Annual increase	Typical value in 2006
Network Bandwidth	26%	65 MWord/s = 0.03 word/flop
Network latency	15%	5 ms = 20K FP ops

- **Note that for both Memory and Network, Latency (not bandwidth) is the weaker link**
  - ➢ **This means that it is better to use Larger Chunk Sizes (Larger Granularity)**
  - ➢ **Better to Retrieve (from Memory) or Transmit (over the Network) a small number of large blocks, rather than a large number of small blocks.**

- **However, there is a Tradeoff between using larger Granularity and Locality**
  - **CPU Performance improves much faster than RAM Memory Performance**



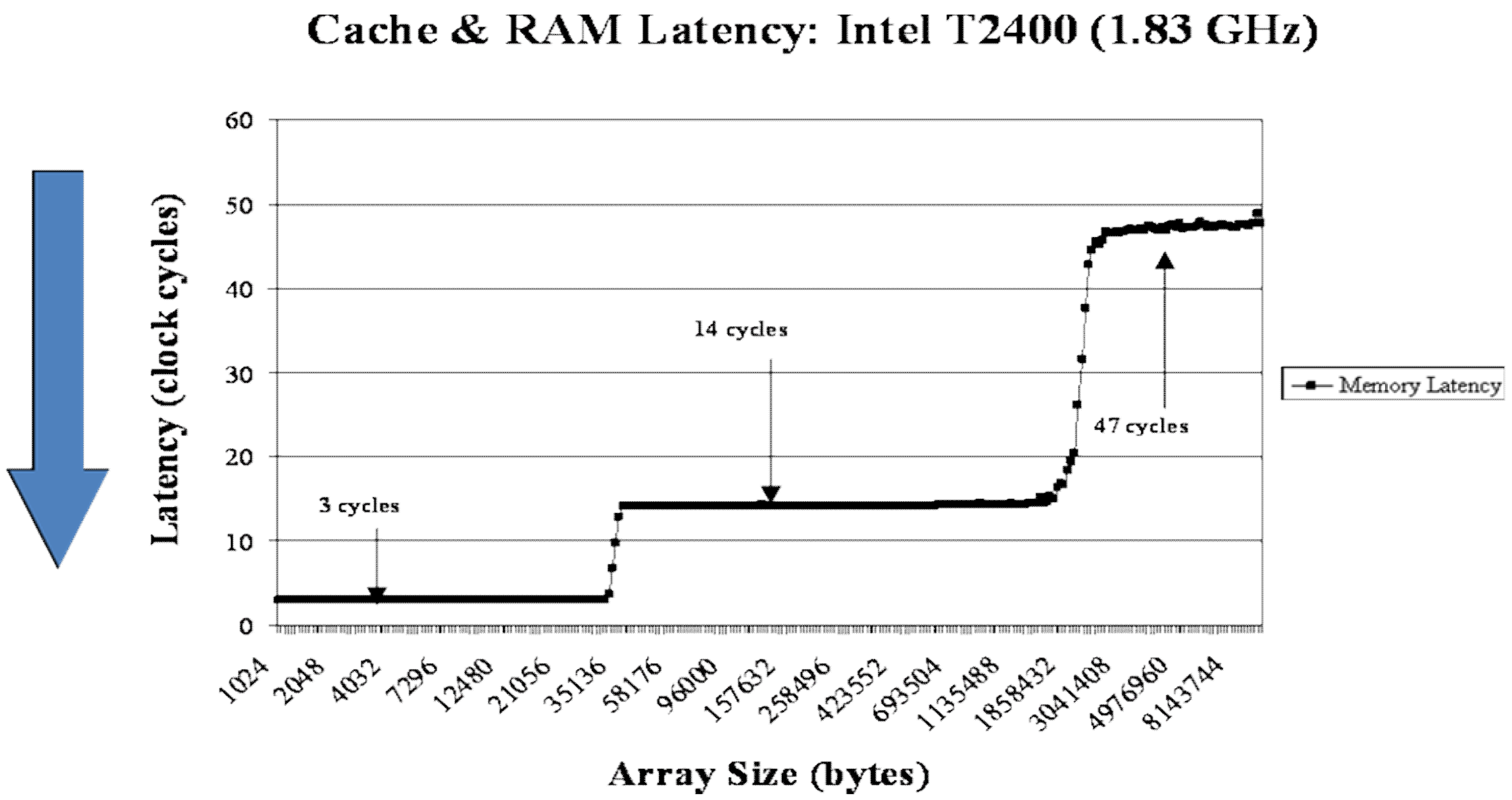
  - **So Memory Hierarchies are Used to Provide Cost-Performance Effectiveness**
    - **Small Memories are Fast, but Expensive; Large Memories are Cheap, but Slow**



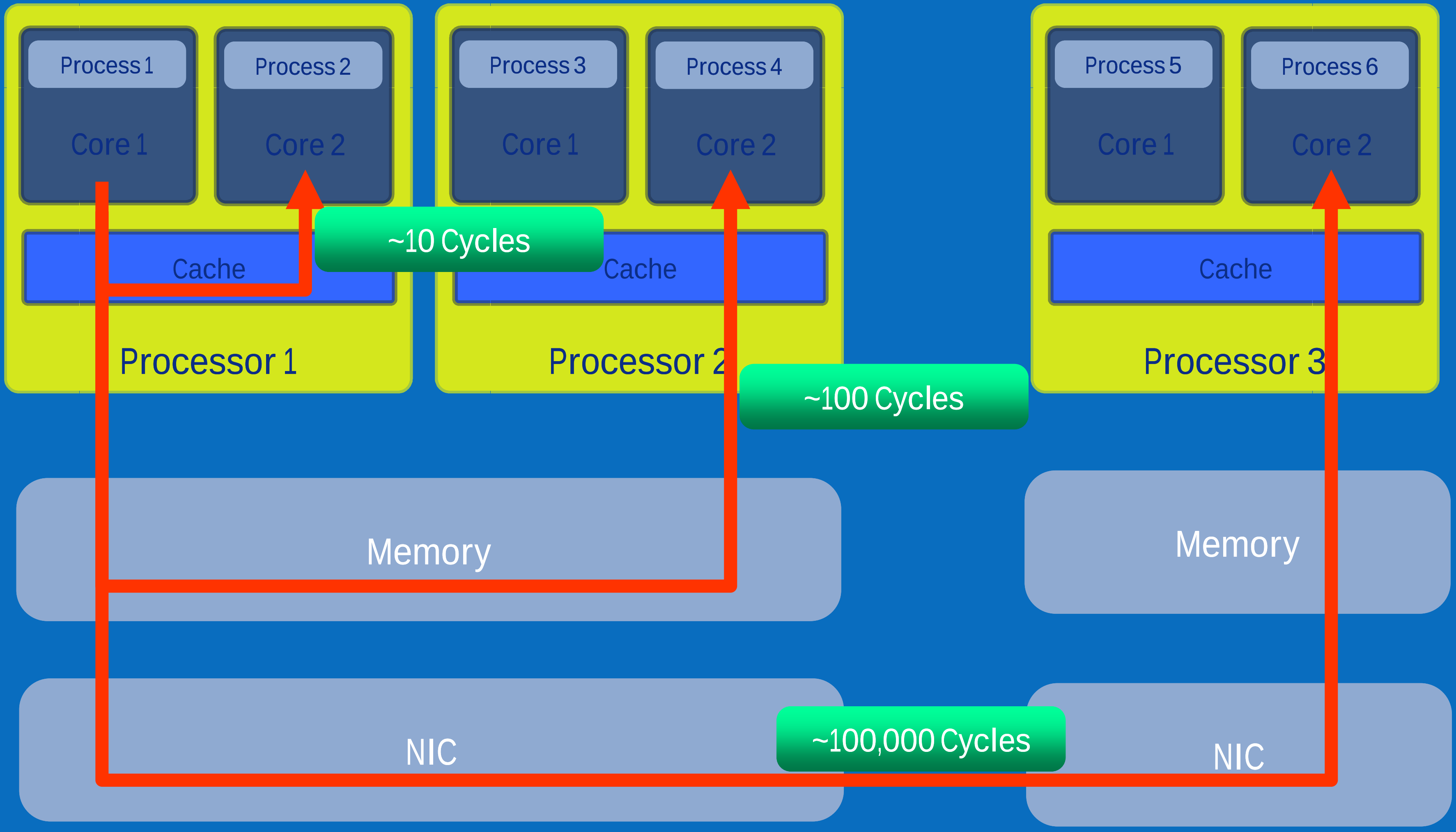
- **Locality (location of the data in the Mem Hierarchy) Substantially Impacts Performance**
    - **Keeping active Working Set in upper levels improves performance**
        - **But this means we need to use finer granularity (many smaller blocks)**

Example: Intel Pentium4

- L1 cache: 3 cycles = 1.64 ns for a 1.83 GHz CPU = 12 calculations
- L2 cache: 14 cycles = 7.65 ns for a 1.83 GHz CPU = 56 calculations
- RAM: 48 cycles = 26.2 ns for a 1.83 GHz CPU = 192 calculations

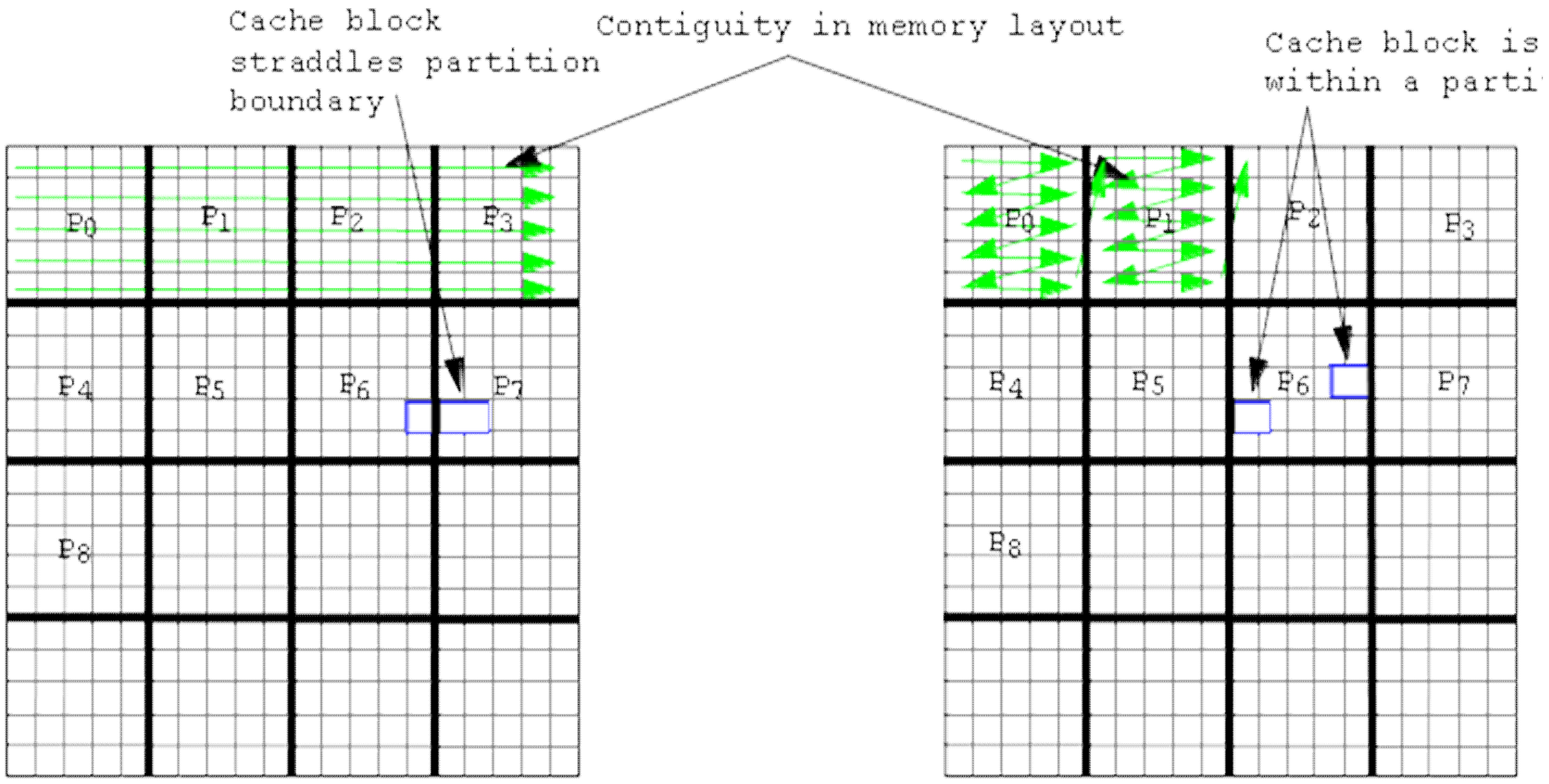


Communication in Parallel Applications



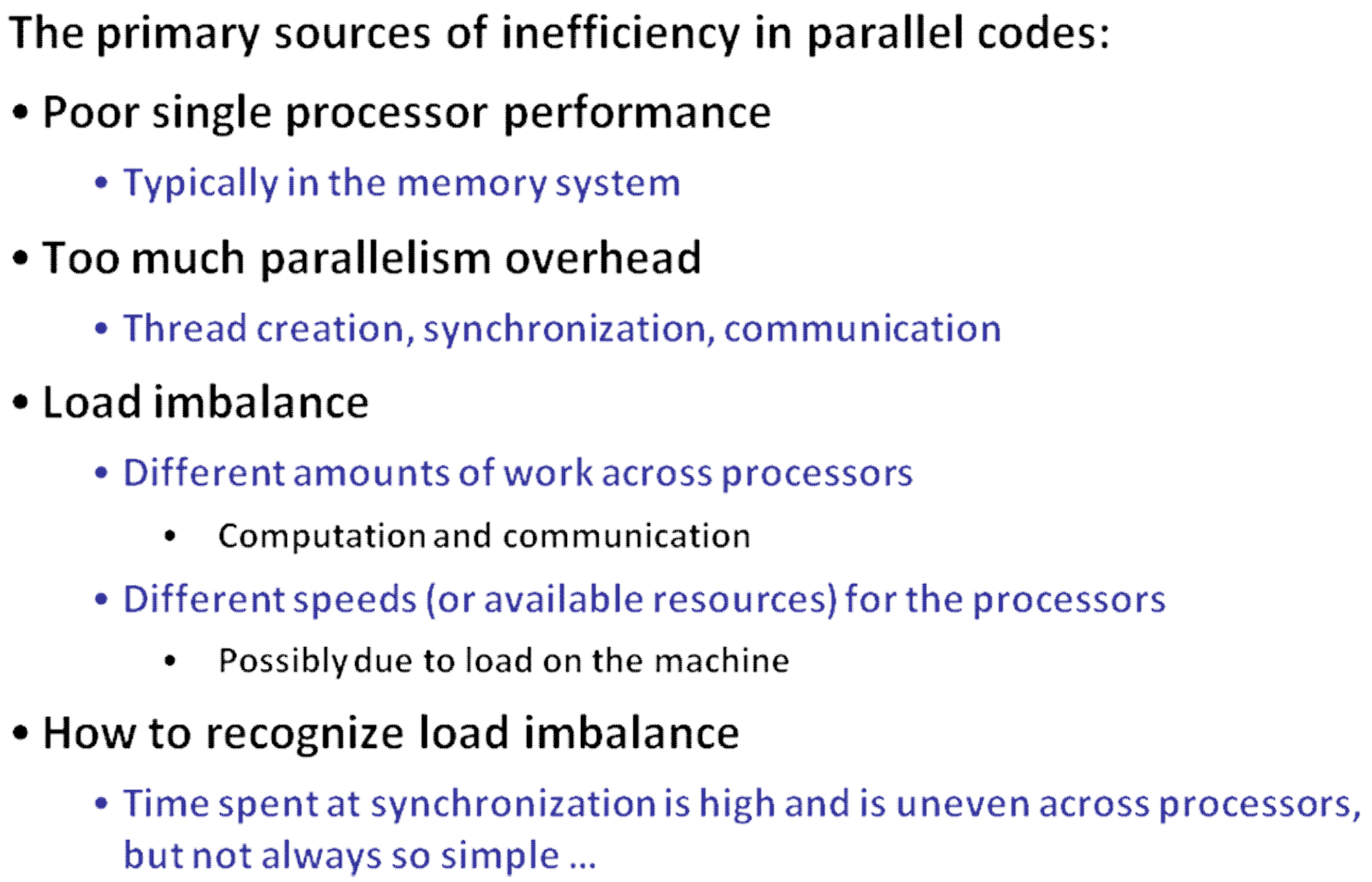
In parallel programming, communication considerations have the same importance as single core optimizations!

- **Tiling can be used to Partition Task such that Memory Hierarchy is better Leveraged**


Cache block straddles partition boundary
Contiguity in memory layout
Cache block is within a parti
P0
P1
P2
P3
P4
P5
P6
P7
P8
P0
P1
P2
P3
P4
P5
P6
P7
P8

- **Challenge: Tradeoff in Granularity Size**
  - **From a BandWidth vs. Latency Point of View with Memory and Network:**
    - **➔ Want Larger Blocks because Latency is Slower than Bandwidth**
  - **From a Memory Locality Point of View:**
    - **➔ Want Smaller Blocks that will fit into Fastest (Smallest) Memory in Hierarchy Reduces Mem Access Times & Can make possible SuperLinear Speedup**

- **Partitioning Should also Strive to Load Balance Tasks onto the Processors**



The primary sources of inefficiency in parallel codes:

- Poor single processor performance
  - Typically in the memory system
- Too much parallelism overhead
  - Thread creation, synchronization, communication
- Load imbalance
  - Different amounts of work across processors
    - Computation and communication
  - Different speeds (or available resources) for the processors
    - Possibly due to load on the machine
- How to recognize load imbalance
  - Time spent at synchronization is high and is uneven across processors, but not always so simple …

- **Load Imbalance is the Time that some processors in the system are idle due to:**
  - **Insufficient Parallelism**
  - **Unequal Size Tasks**

- **Load Imbalance Exacerbates Synchronization Overhead**
  - **Slowest (Longest) Task or Processor holds up all other Tasks or Processors**

Improving Real Performance

Peak Performance grows exponentially, a la Moore's Law

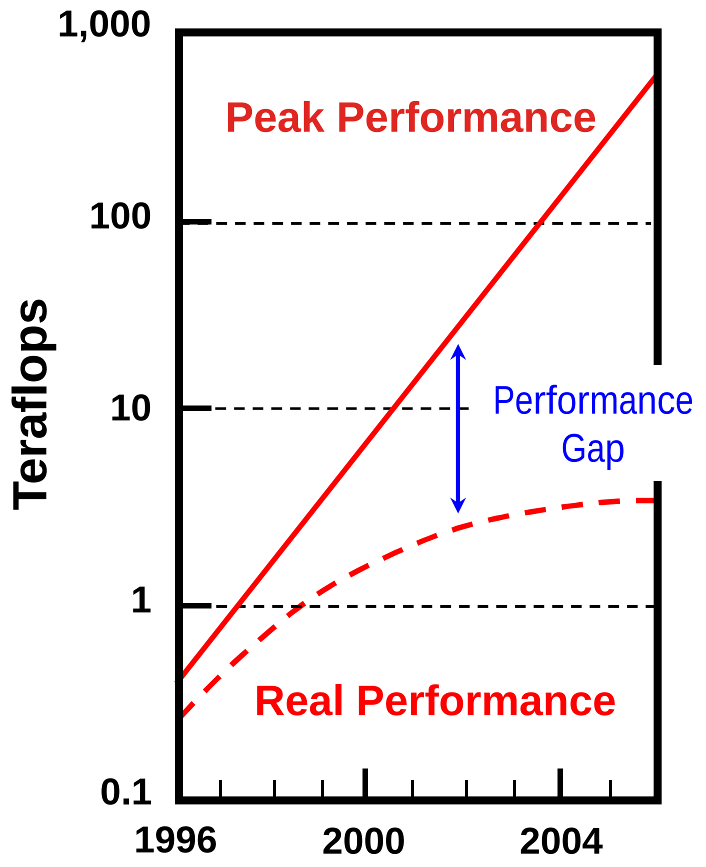
- In 1990's, peak performance increased 100x; in 2000's, it will increase 1000x

But efficiency (the performance relative to the hardware peak) has declined

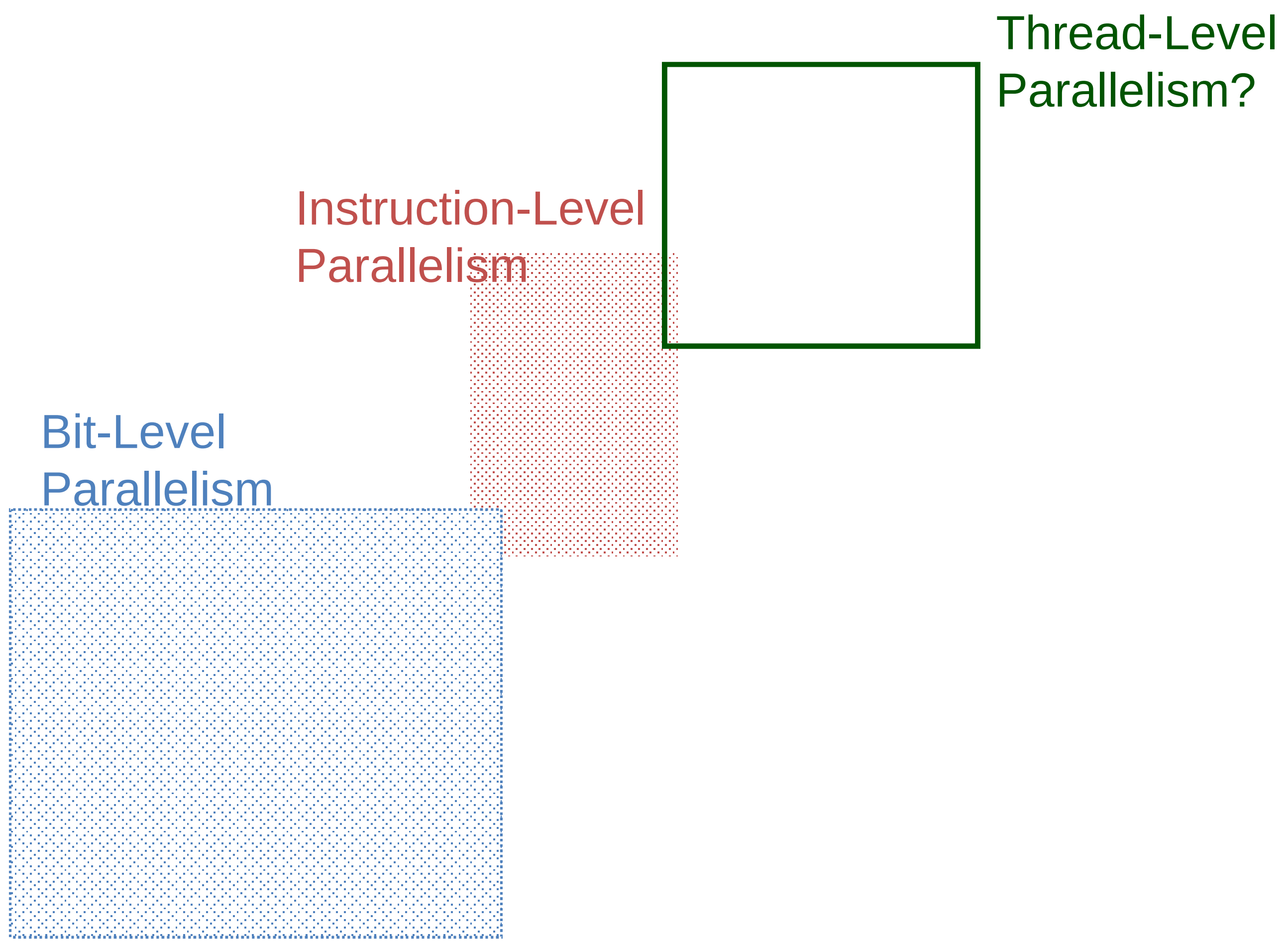
- was 40-50% on the vector supercomputers of 1990s
- now as little as 5-10% on parallel supercomputers of today

Close the gap through ...

- Mathematical methods and algorithms that achieve high performance on a single processor and scale to thousands of processors
- More efficient programming models and tools for massively parallel supercomputers



Much of the Performance is from Parallelism


Thread-Level
Parallelism?
Instruction-Level
Parallelism
Bit-Level
Parallelism