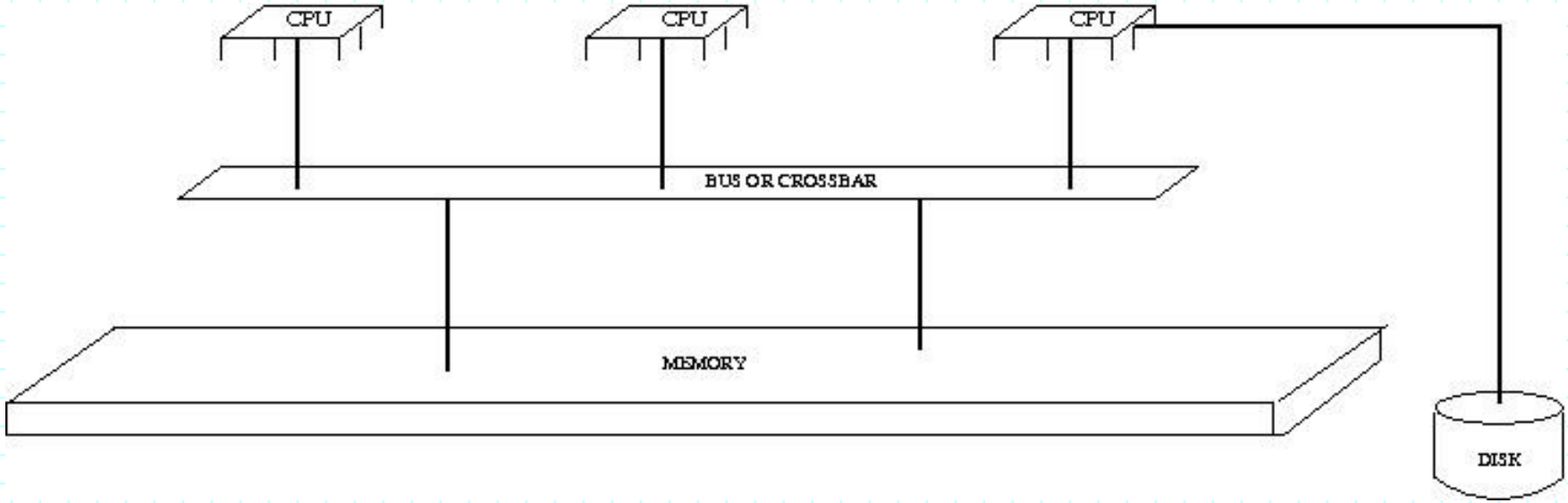
High Performance Computing Systems

- Shared Memory

Doug Shook

Shared Memory



- Bottlenecks
  - Trips to memory
  - Cache coherence

Why Multicore?

- Shared memory systems used to be purely the domain of HPC....
- What happened?

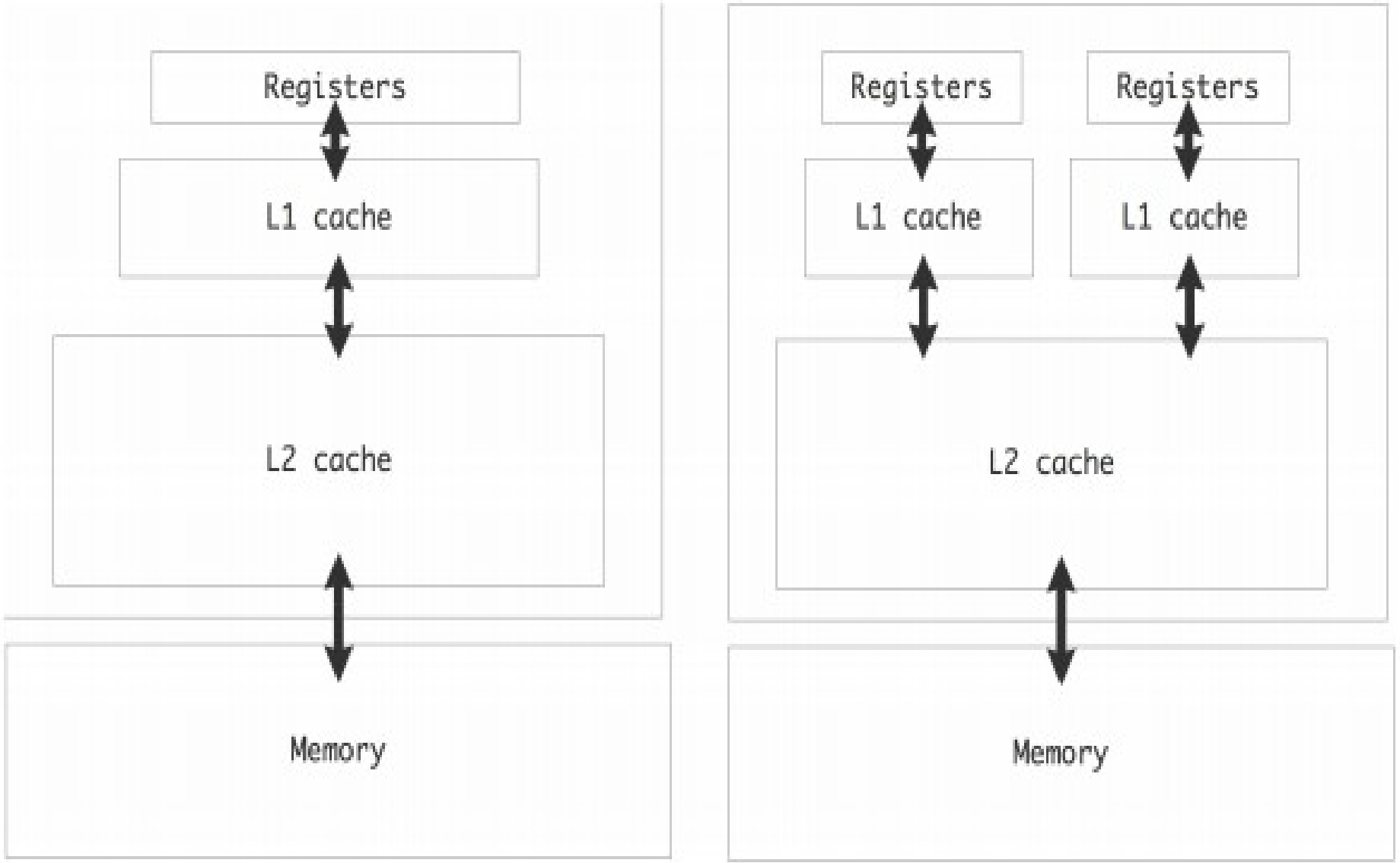
Architecture Types

- Four primary architectures (Flynn's taxonomy)
  - SISD
  - SIMD
  - MISD
  - MIMD
- Based on these descriptions, what do today's machines fall under?

Cache Coherency

- The "shared" in "shared memory" refers to main memory
- What about caches?

Cache Coherency



Cache Coherency

- What kinds of problems can result in this new architecture development?
  - Think about cache replacement policies...
  - Think about two cores using the same data set...

Cache Coherency

- Each line in cache has a state associated with it:
  - Modified
  - Shared
  - Invalid

Cache Coherency

- Consider a two core machine, each with private caches. Each cache has a copy of x.
  - Three states
  - Two potential operations: read and write
- Let's design an FSA that models how this works
- How do we pass state information through the CPU?

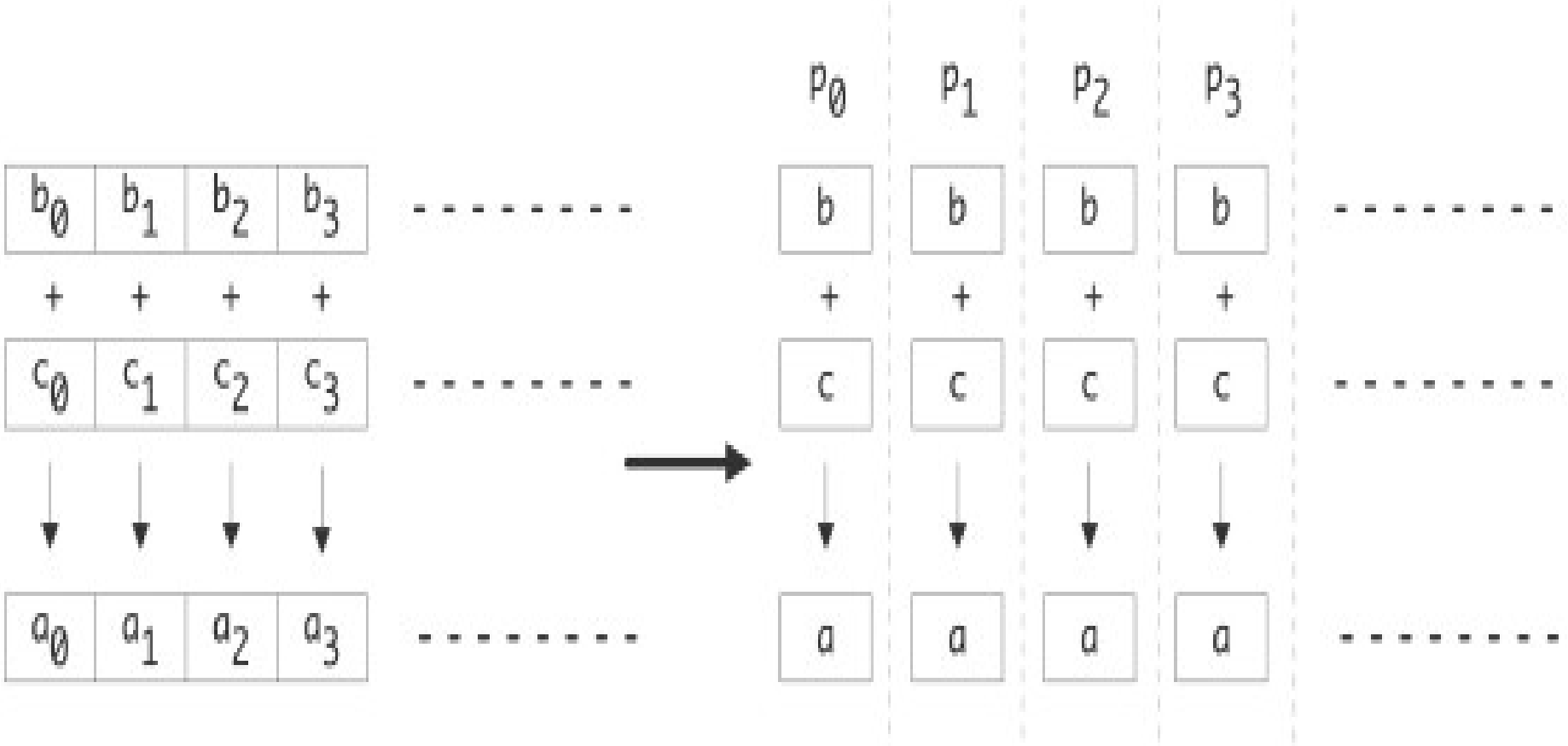
Writing Parallel Code

- Examine the following:

```
for (i = 0; i < n; i++)
    a[i] = b[i] + c[i]
```

- Can we run this code on multiple cores?
  - Why or why not?
  - If so, how many could we use?

Writing Parallel Code



- If the original algorithm took time n and we have p processors, how fast would we expect the parallel code to be?

Writing Parallel Code

- Now take a look at this code:

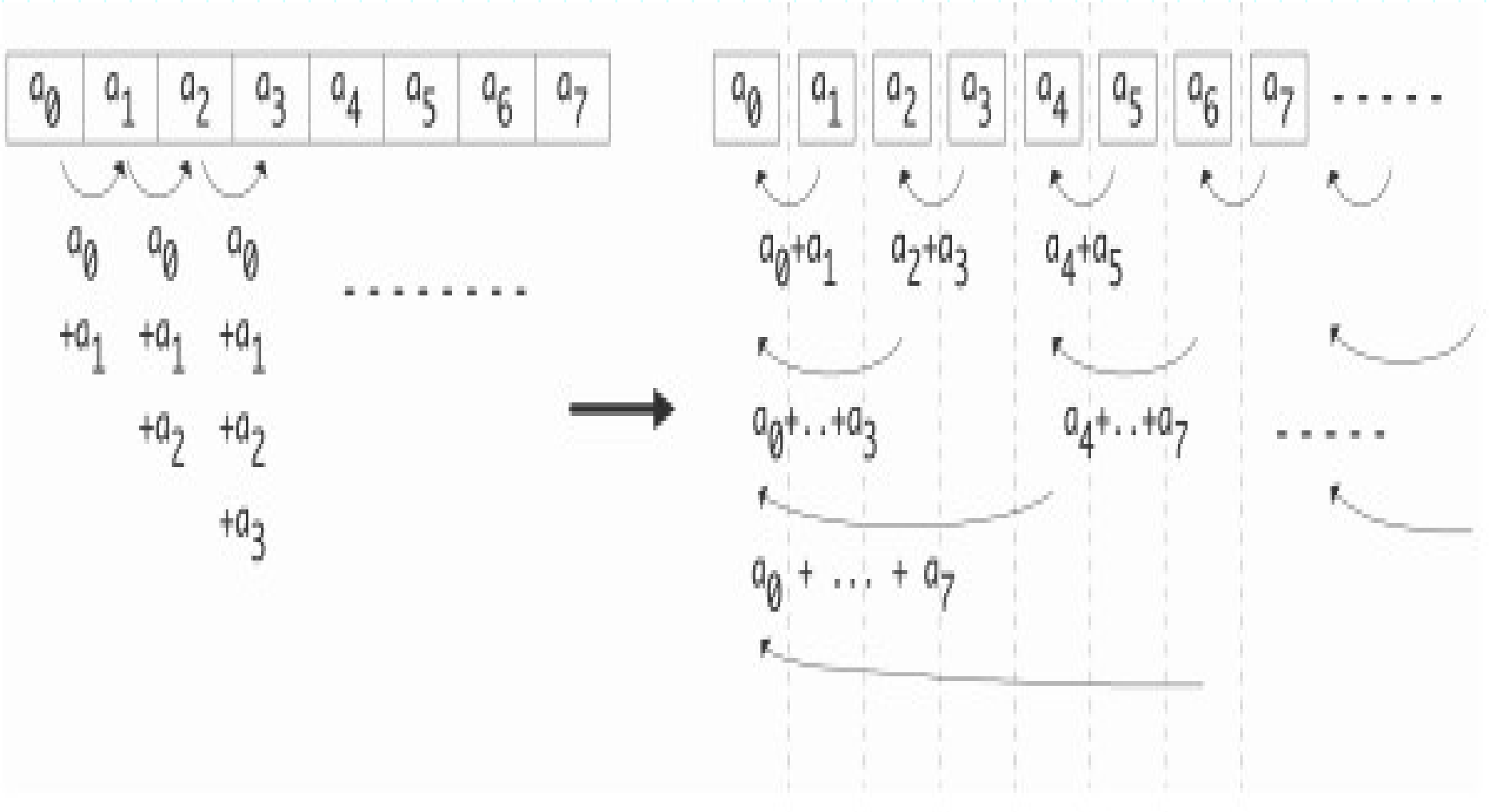
```
s = 0;
for (i = 0; i < n; i++)
    s += x[i]
```

- Can we run this code on multiple cores?
  - Why or why not?
  - If so, how many could we use?

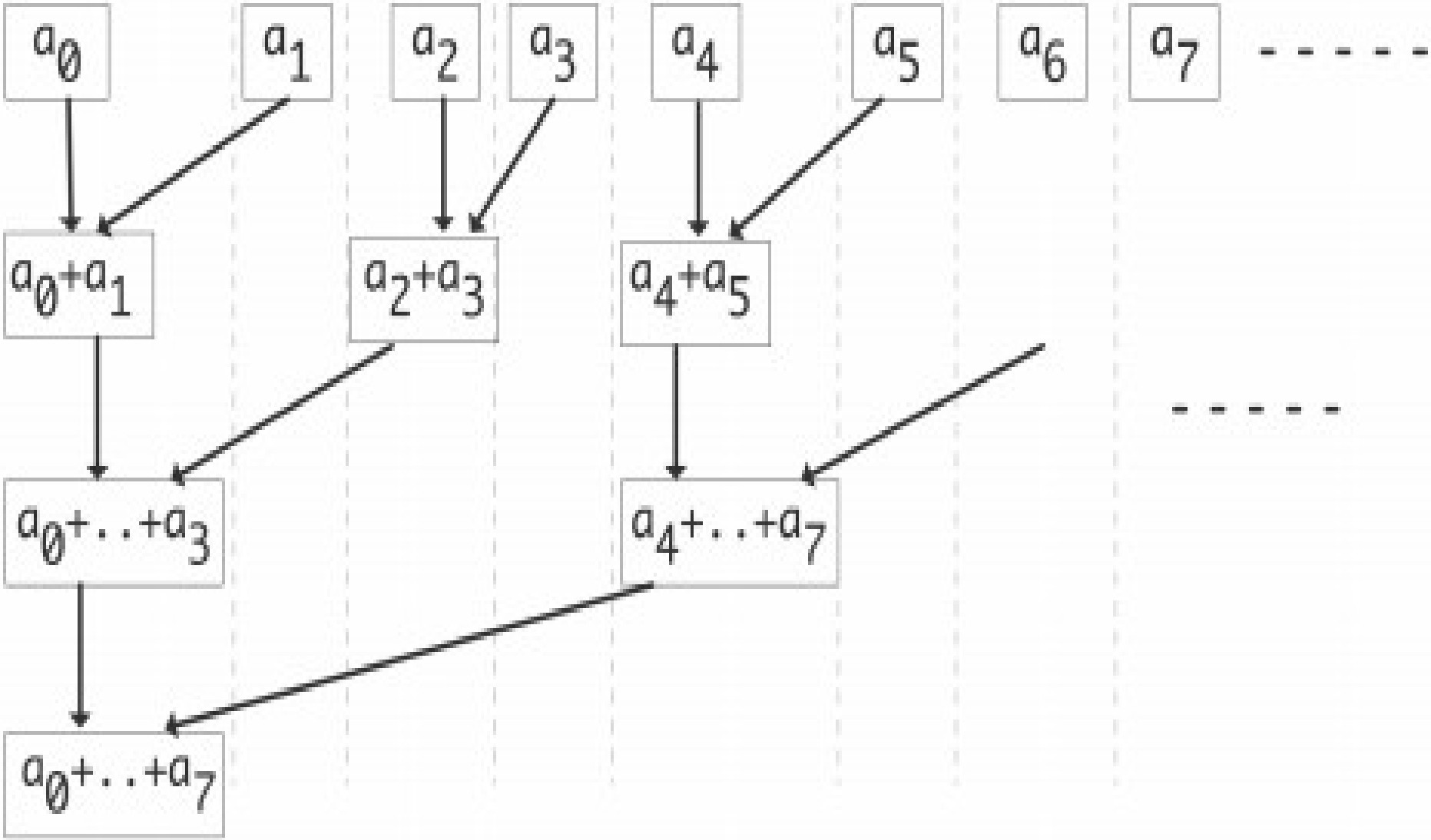
Writing Parallel Code

- Okay so the last example didn't work....
  - Could we rewrite it somehow?

Writing Parallel Code



Writing Parallel Code



Writing Parallel Code

- Consider that in the previous example each node started with exactly one element.
  - How does the communication change if we have 8 elements with 4 processors?
    - How should the elements be distributed in the beginning?
  - What about 16 elements with 4 processors?

Granularity of Parallelism

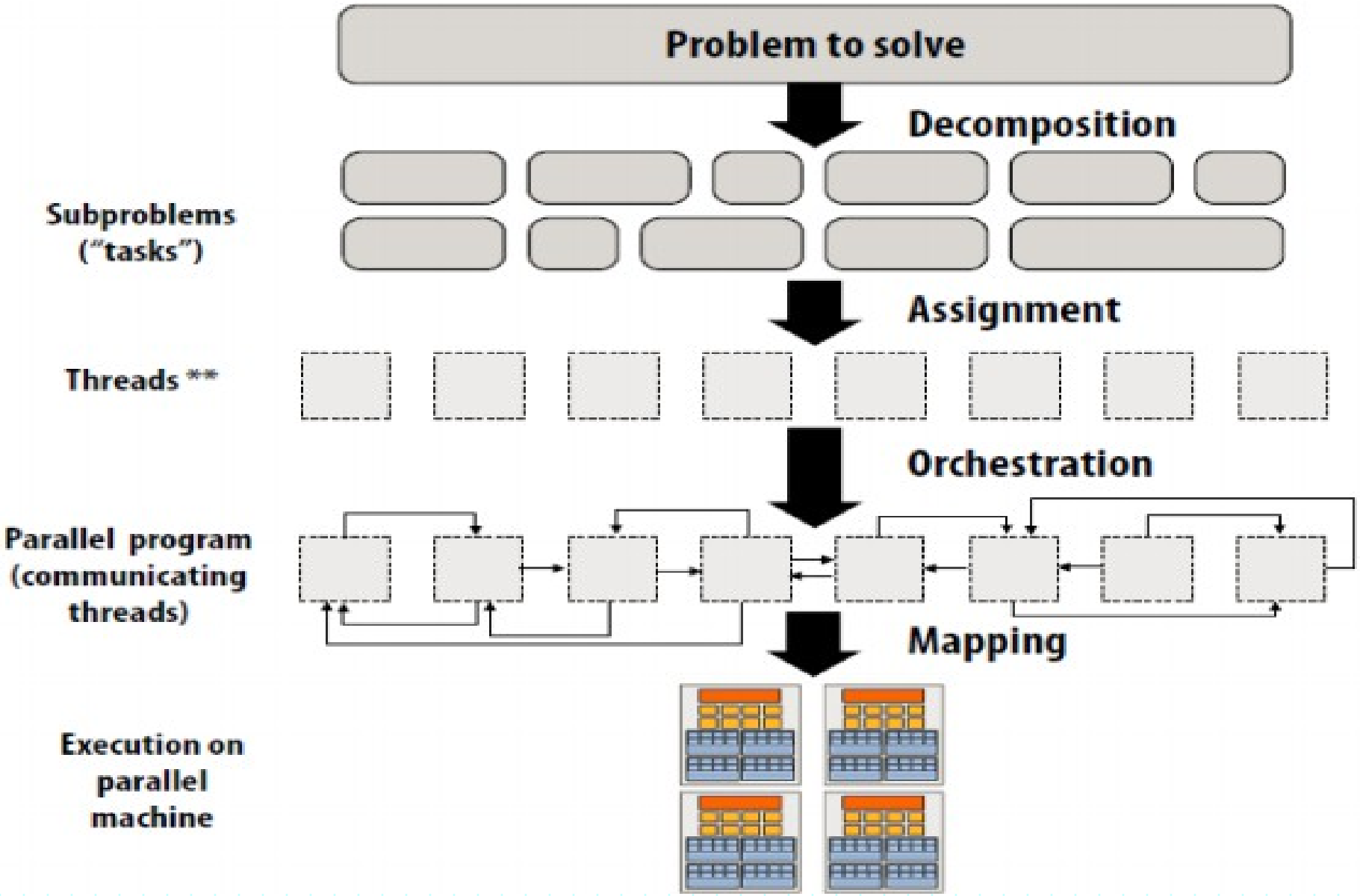
- What had to be true in the previous examples in order to parallelize our code?

- Other types of parallelism exist
  - Instruction level
  - Task-level

Task Parallelism

- Entire subprograms that can be executed simultaneously
  - Classic example: tree search
    - Two potential approaches

Parallel Program Design



Efficiency

- How much performance gain should we expect?
  - Can we predict this ahead of time?
  - What factors go into efficiency of parallel programs?

Speedup

- Simply compare the time it takes to run on one processor to the time it takes to run on p processors:

$$S_p = T_1 / T_p$$

- Ideal case?
  - Should we expect the ideal case?
- Superlinear speedup – real or myth?

Efficiency

- Used to measure how far we are from ideal speedup:

$$E_p = S_1 / p$$

Amdahl's Law

- Let's suppose that only part of the code is parallelizable:

  sequential part + parallel part = 1

- How does this effect speedup?
  - What does our equation become?
  - What if we have an infinite number of processors?

- Limit of speedup?
  - Efficiency as a function of processors?

Amdahl's Law

- Questions:
  - Assume code with 1 second sequential execution and 1000 seconds of parallelizable execution. What is the speedup and efficiency with 100 processors? 500 processors?
  - If the number of processors increases, how much does the parallel fraction of code have to increase to maintain the same efficiency?

Amdahl's Law

- This is still a bit optimistic...
  - What are we missing?
- How can we adjust the equation to reflect this?
  - Effect on speedup?

Gustafson's Law

- One major flaw with Amdahl's Law
  - What assumption does it make about problem size?
- Enter Gustafson!
- Implications of these two laws?

Scalability

- The way a problem is divided can make it difficult to talk about speedup

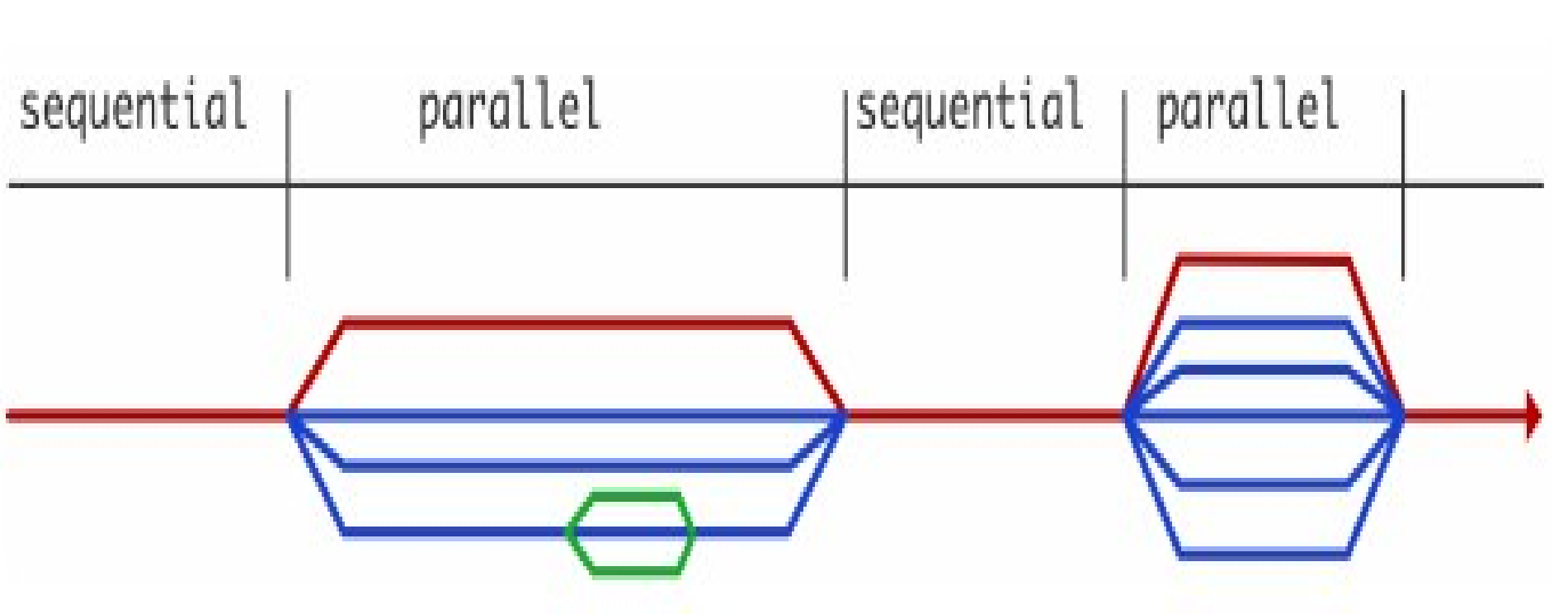
- Use scalability instead:
  - Strong scalability
  - Weak scalability

Load Balancing

- Which is better?
  - Out of p processors, one finishes early
  - Out of p processors, one finishes late
- Let's prove it!
- Which parts of a parallel program affect this?

Threads

- Process vs. Thread
- Fork / join



Context

- There are actually two types of data at work here
  - Can you determine which is which?
- All of the data that a thread can access defines its context
- What has to happen when a new thread is scheduled on a processor?

Atomic Operations

- Let's say we have a variable of interest, sum. One thread wants to increase sum by 2, another thread wants to increase it by 3.

- What potential problems could arise?

Atomic Operations

scenario 1.		scenario 2.		scenario 3.	
I = 0					
read I = 0	read I = 0	read I = 0	read I = 0	read I = 0	
compute I = 2	compute I = 3	compute I = 2	compute I = 3	compute I = 2	
write I = 2			write I = 3	write I = 2	
	write I = 3	write I = 2			read I = 2
					compute I = 5
					write I = 5
I = 3		I = 2		I = 5	

Atomic Operations

- Here's a more realistic example:

```
for (i=0; i<1000; i++)
    sum = sum+a[i]*b[i];
```

- What's the problem?
  - Two possible solutions

Affinity

- Putting the execution where the data is:

```
for (i=0; i<ndata; i++) // this loop will be done by threads
  x[i] = ....
for (i=0; i<ndata; i++) // as will this one
  ... = .... x[i] ...
```

```
for (i=0; i<ndata; i++) // produces loop
  x[i] = ....
for (i=0; i<ndata; i+=2) // use even indices
  ... = ... x[i] ...
for (i=1; i<ndata; i+=2) // use odd indices
  ... = ... x[i] ...
```

Hyperthreading

