Outline

1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → **Flynn's taxonomy**
   - → Basic limitations of par. computing
     - → Amdahl's law
     - → Gustafson's law
   - → Multithreaded processors (SMT)
   - → Multicore processors
   - → Shared-memory computers
     - → UMA
     - → ccNUMA
   - → Distributed-memory computers
   - → Networks
     - → Basic performance characteristics
     - → Network topologies
       - → Buses
       - → Ring & fully connected networks
       - → Switched & fat-tree networks
       - → Mesh networks
     - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
9. Distributed-memory programming with MPI
10. Shared-memory programming with OpenMP
11. Hybrid programming (MPI + OpenMP)
12. Heterogeneous architectures (GPUs, Xeon Phis)
13. Energy efficiency

Parallel computing

- **As all modern supercomputer architectures heavily depend on parallelism, this trend will substantially increase in the near future**

- **Parallel computing has already entered the everyday life**
  - → Nearly every desktop pc or notebook sold today includes multicore chips (dual, quad, …)

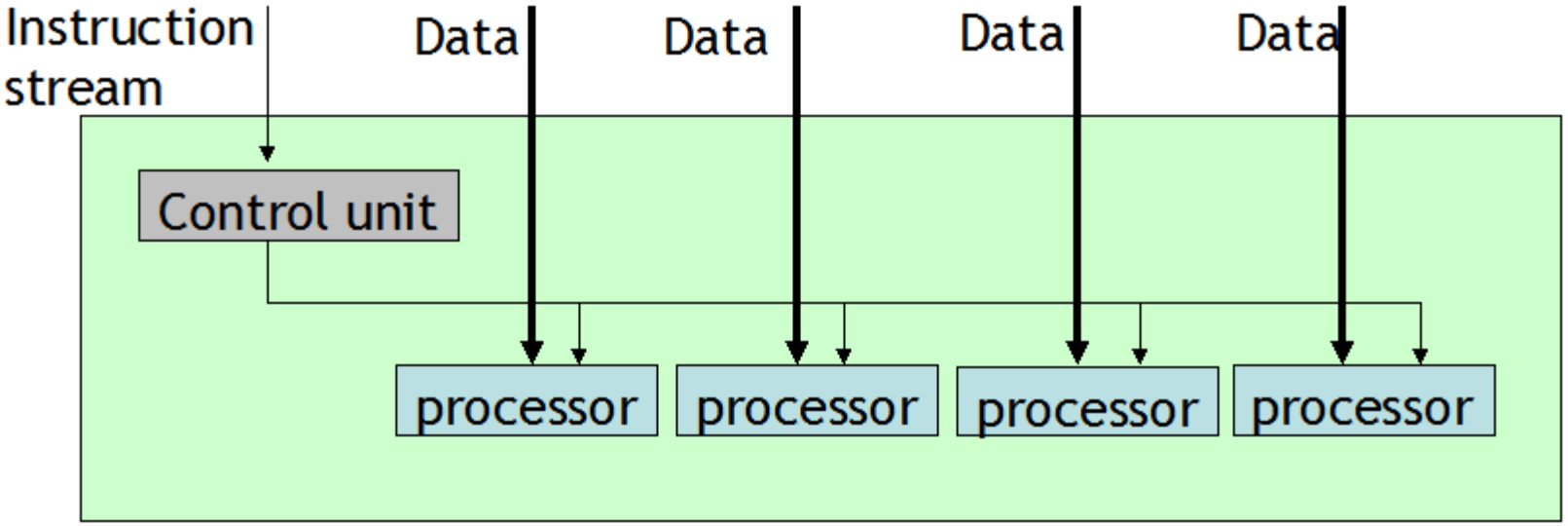
- **Basic classification of parallel computers**
  - → Shared-memory multiprocessors systems: Multiple processors can execute code in parallel but they all share one virtual address space (shared memory)
  - → Distributed-memory systems: Multiple processors/compute nodes are connected over a network. The processors each have their own virtual address space (memory).

Flynn's Taxonomy

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

- **SISD – Single instruction, single data**
  - → An example is the classical von Neumann architecture
- **SIMD – Single instruction, multiple data**
  - → Vector-processors, SIMD capabilities of modern superscalar microprocessors, GPU's
- **MISD – Multiple instructions, single data**
  - → Not existant: Is commonly not regarded as practical relevant
- **MIMD – Multiple instructions, multiple data**
  - → Shared memory, distributed memory architectures
- → **Only two still considered meaningful are SIMD and MIMD**

Single instruction multiple data (SIMD)



- **E.g. Vector units like MMX, SSE and AVX also belong to this category**

Multiple instruction multiple data (MIMD)

- **Most general model: Each processor works on its own data with its own instruction stream**

- **Very general case:**
  - → Every scenario can be mapped to MIMD

- **Further breakdown of MIMD is usually based on the organization of memory**
  - → Shared-memory systems
  - → Distributed-memory systems

Multiple instruction multiple data (MIMD)

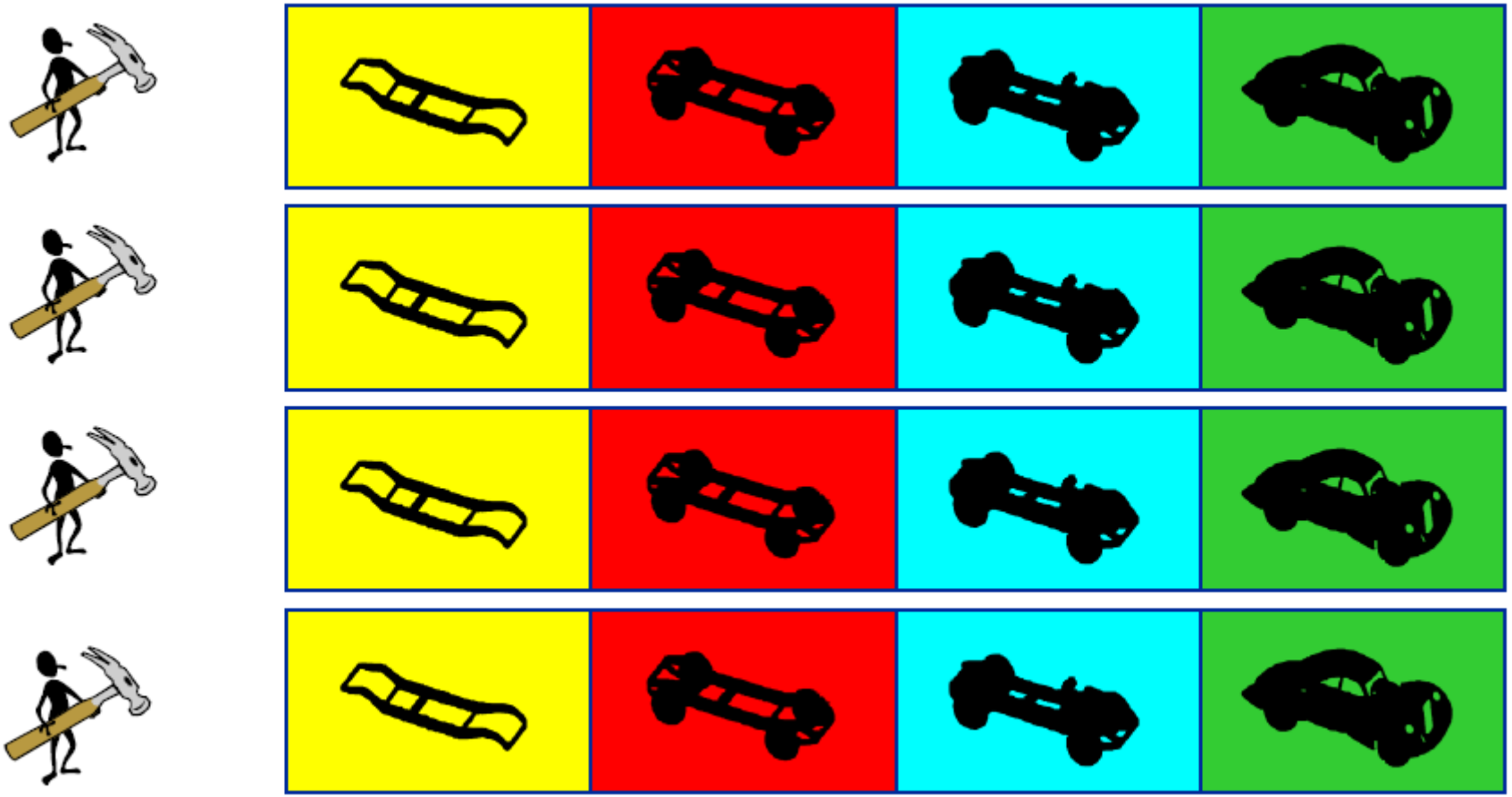
- **In practice: Single Program Multiple data**
  - → All processors execute the same code stream
  - → Just not the same instruction at the same time
  - → Control flow is relatively independent and can be completely different
  - → Amount of data to process may vary: Load balancing becomes important

Outline

1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → **Basic limitations of par. computing**
     - → **Amdahl's law**
     - → Gustafson's law
   - → Multithreaded processors (SMT)
   - → Multicore processors
   - → Shared-memory computers
     - → UMA
     - → ccNUMA
   - → Distributed-memory computers
   - → Networks
     - → Basic performance characteristics
     - → Network topologies
       - → Buses
       - → Ring & fully connected networks
       - → Switched & fat-tree networks
       - → Mesh networks
     - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
9. Distributed-memory programming with MPI
10. Shared-memory programming with OpenMP
11. Hybrid programming (MPI + OpenMP)
12. Heterogeneous architectures (GPUs, Xeon Phis)
13. Energy efficiency

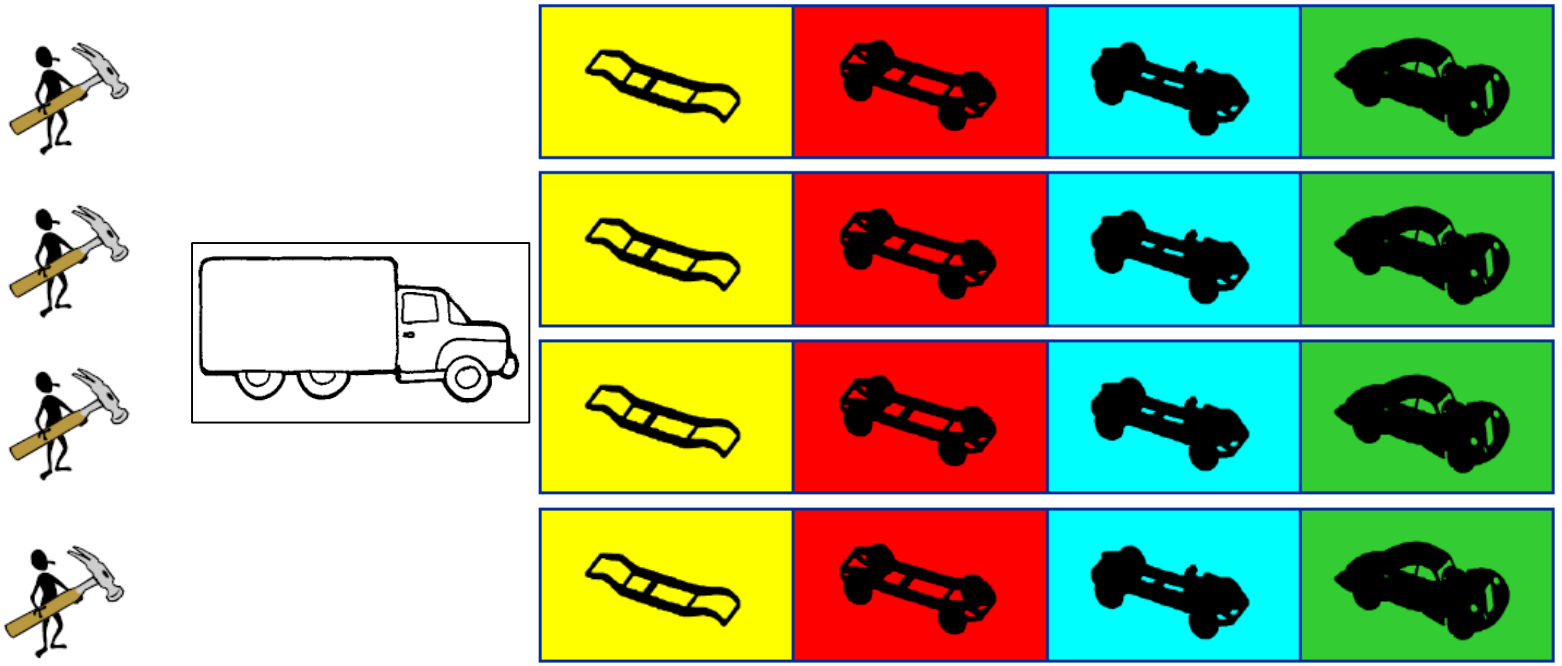
Example of parallel work

- **Example: 4 cars are produced in parallel**



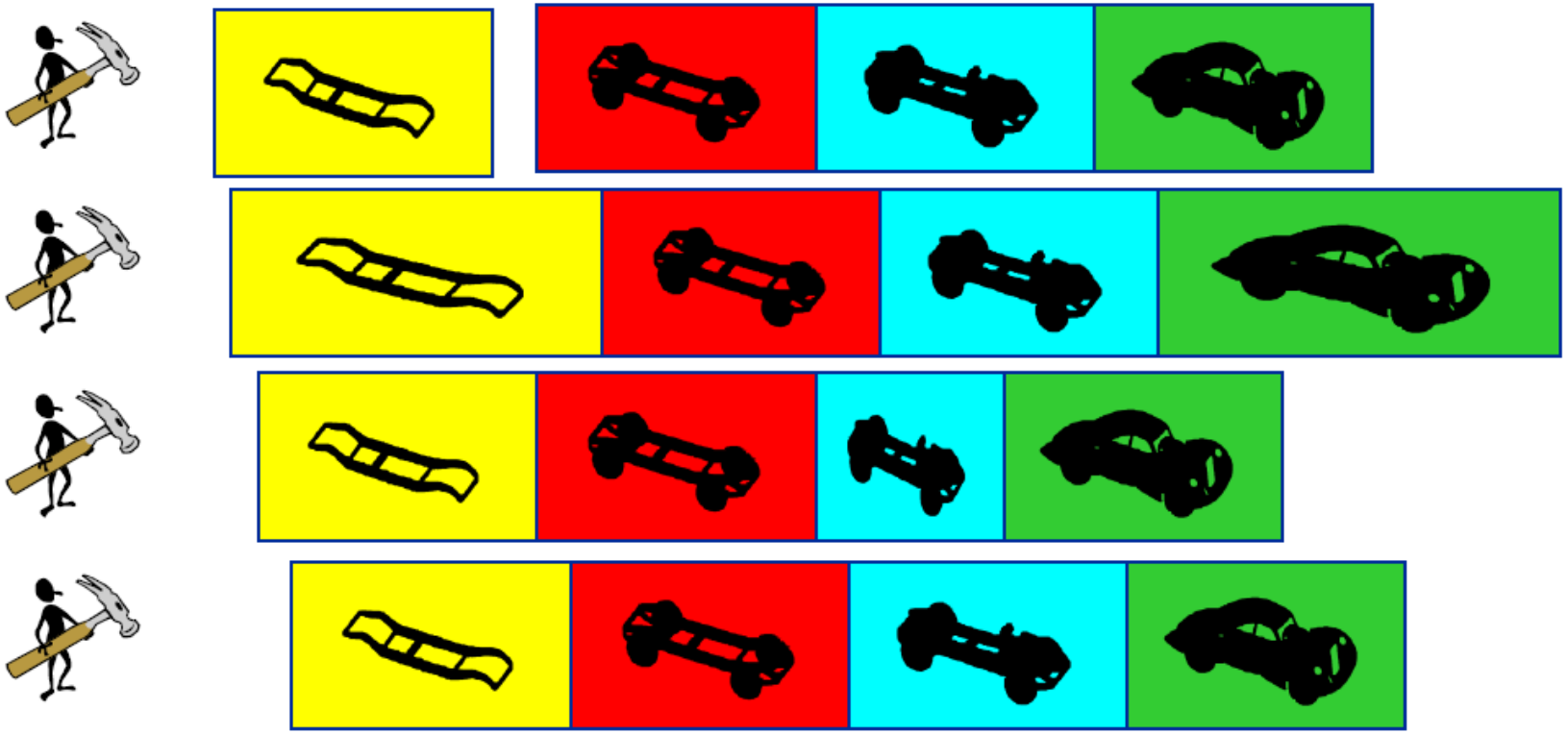
Limits of scalability

- **Parts of the manufacturing process can not be parallelized**
  - → Example: Delivery of components (all workers have to wait)



Limits of scalability (cont.)

- **Individual steps may take more or less time**
  - → Load imbalances lead to unused resources



- **Definition of Speedup (According to Amdahl)**
  - → Ratio between serial and parallel Execution of a Program
  - → Indicatior for relative performance improvement

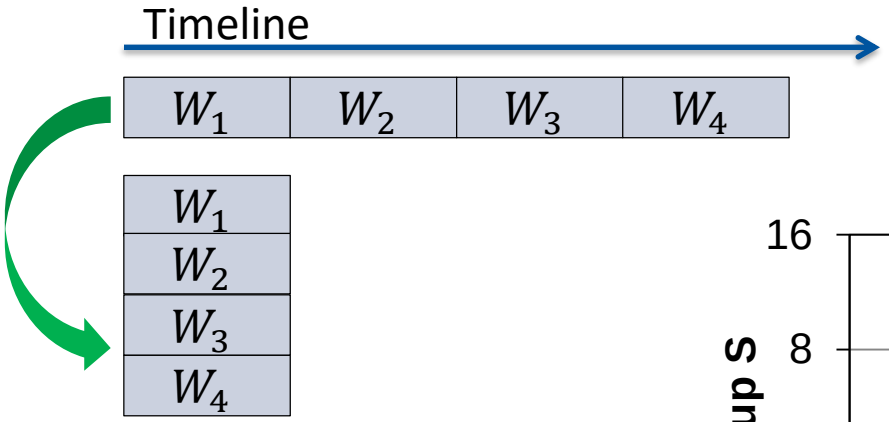
$$\textbf{Speedup } S_p(N) = \frac{T(1)}{T(N)}$$

  - → With T(N): runtime of a (parallel) program with N Processors
- **Efficiency:**

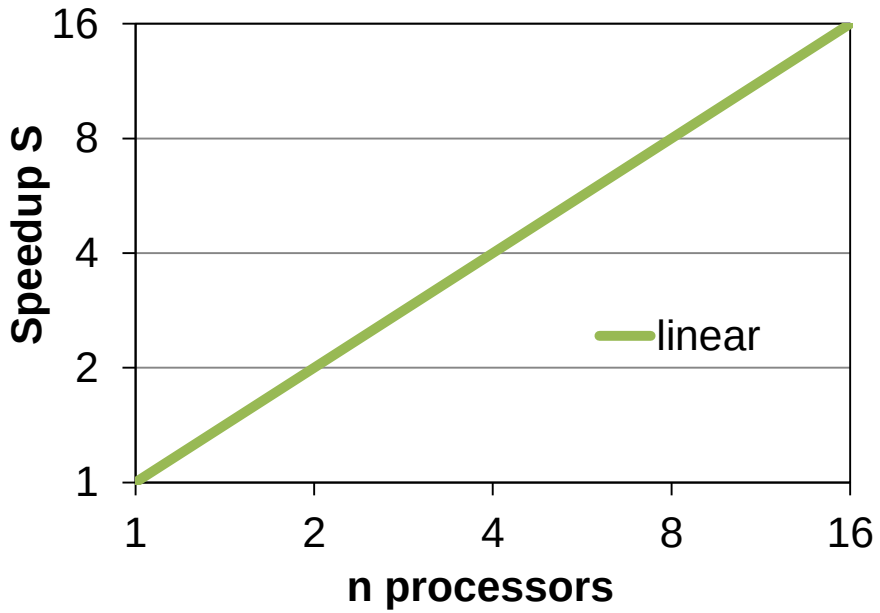
$$\textbf{Efficiency } E_p(N) = \frac{S_p(N)}{N} = \frac{T(1)}{N \cdot T(N)}$$

Linear Speedup

- **Ideal situation: All work is perfectly parallelizable: linear Speedup**
  - → In general: upper bound for parallel execution of programs

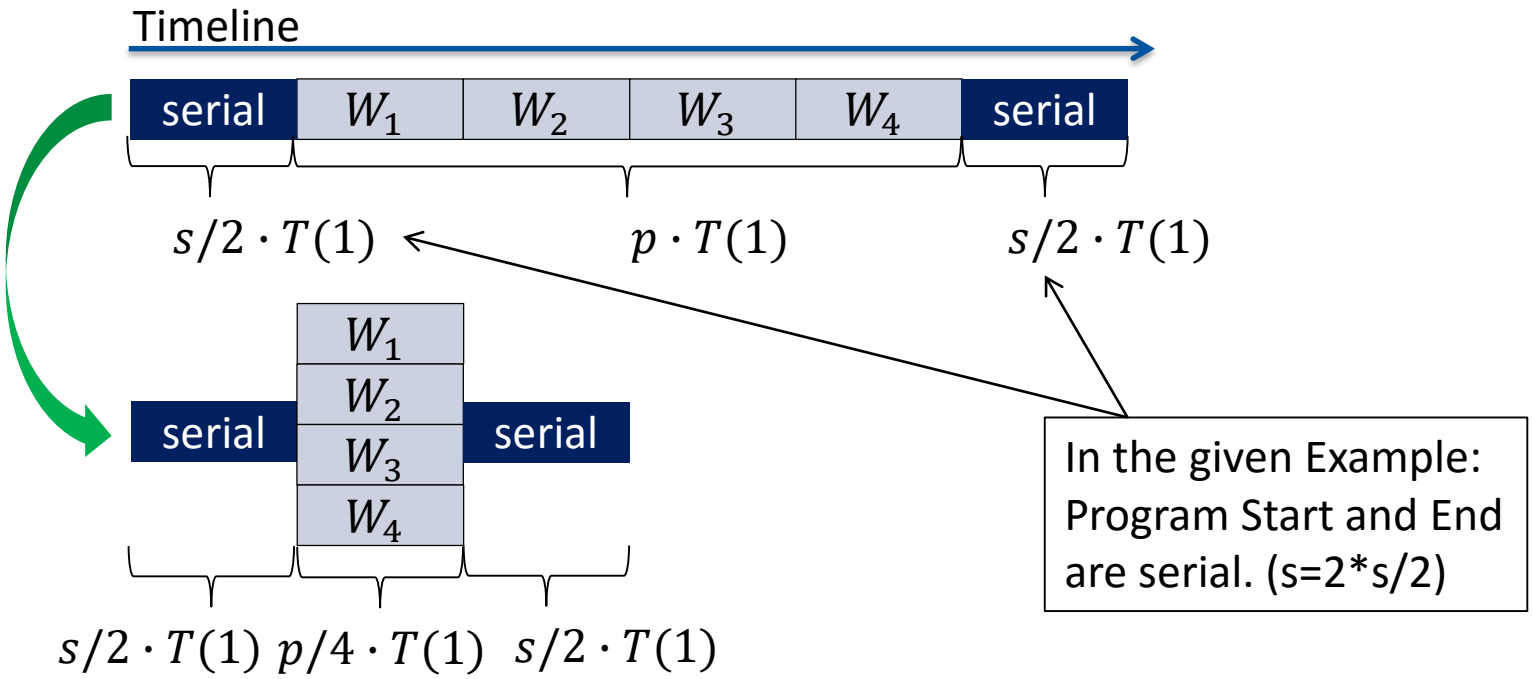


$$S(N) = \frac{T(1)}{T(N)} = N$$



Limitations of scalability

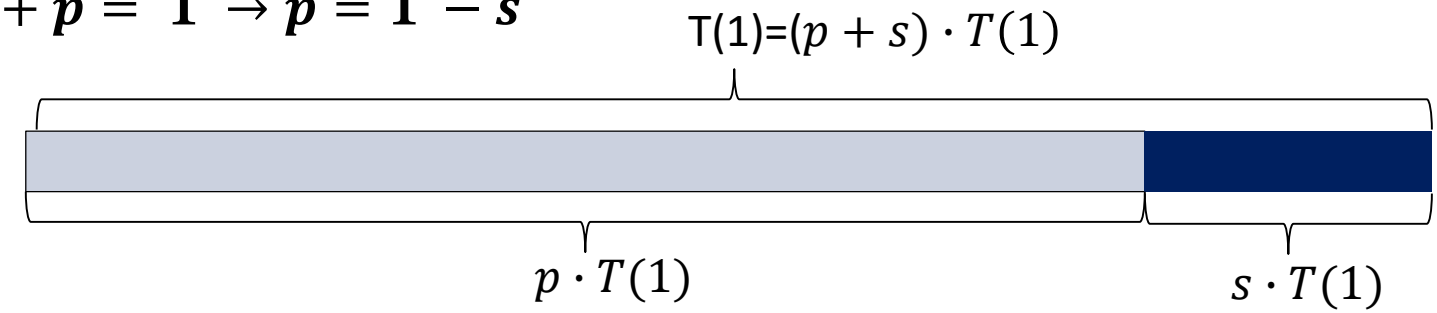
- **A model more close to reality: There are serial parts which limit the maximum speedup**



- **Amdahl's law assumes the program is dividable into an ideal parallelizable fraction *p* and a serial fraction *s* (non-parallelizable)**

Amdahl's Law

- $s + p = 1 \rightarrow p = 1 - s$



- **The parallelized program's execution time is then assumed to be (with N processors):**

  → $\mathbf{T(N) = (s + \frac{p}{N}) \cdot T(1)}$

- **The speedup thus resembles to:**

$$\boldsymbol{S_p(N) = \frac{T(1)}{T(N)} = \frac{1}{s + \frac{1 - s}{N}}}$$

Amdahl's Law (1967)
or "strong scaling"

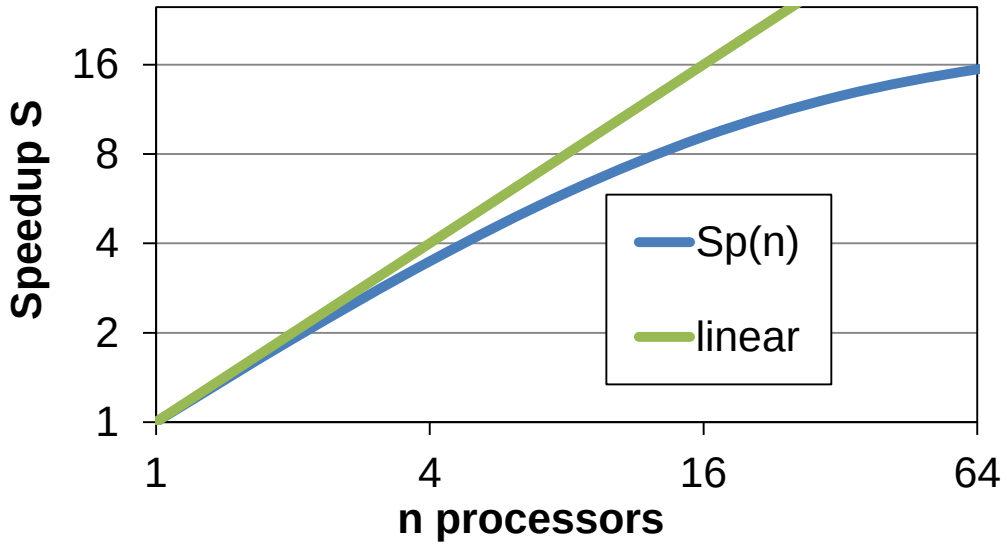
Further reading: Gene Amdahl: Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. In: AFIPS Conference Proceedings. 30, 1967, S. 483–485

Amdahl's Law (cont.)

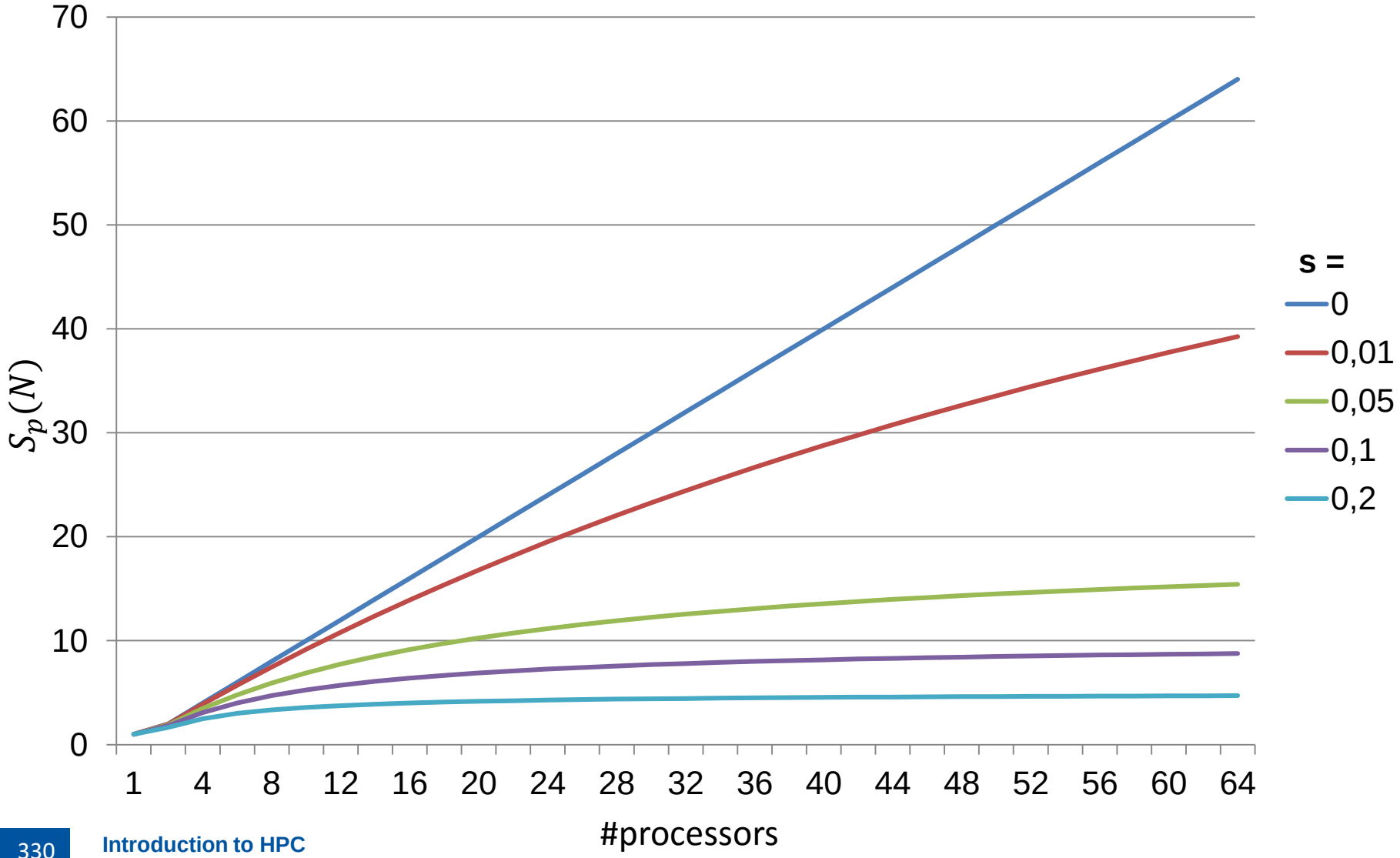
- **Example: Program with 5% serial and 95% parallel fraction**

  → Speedup according to Amdahl:

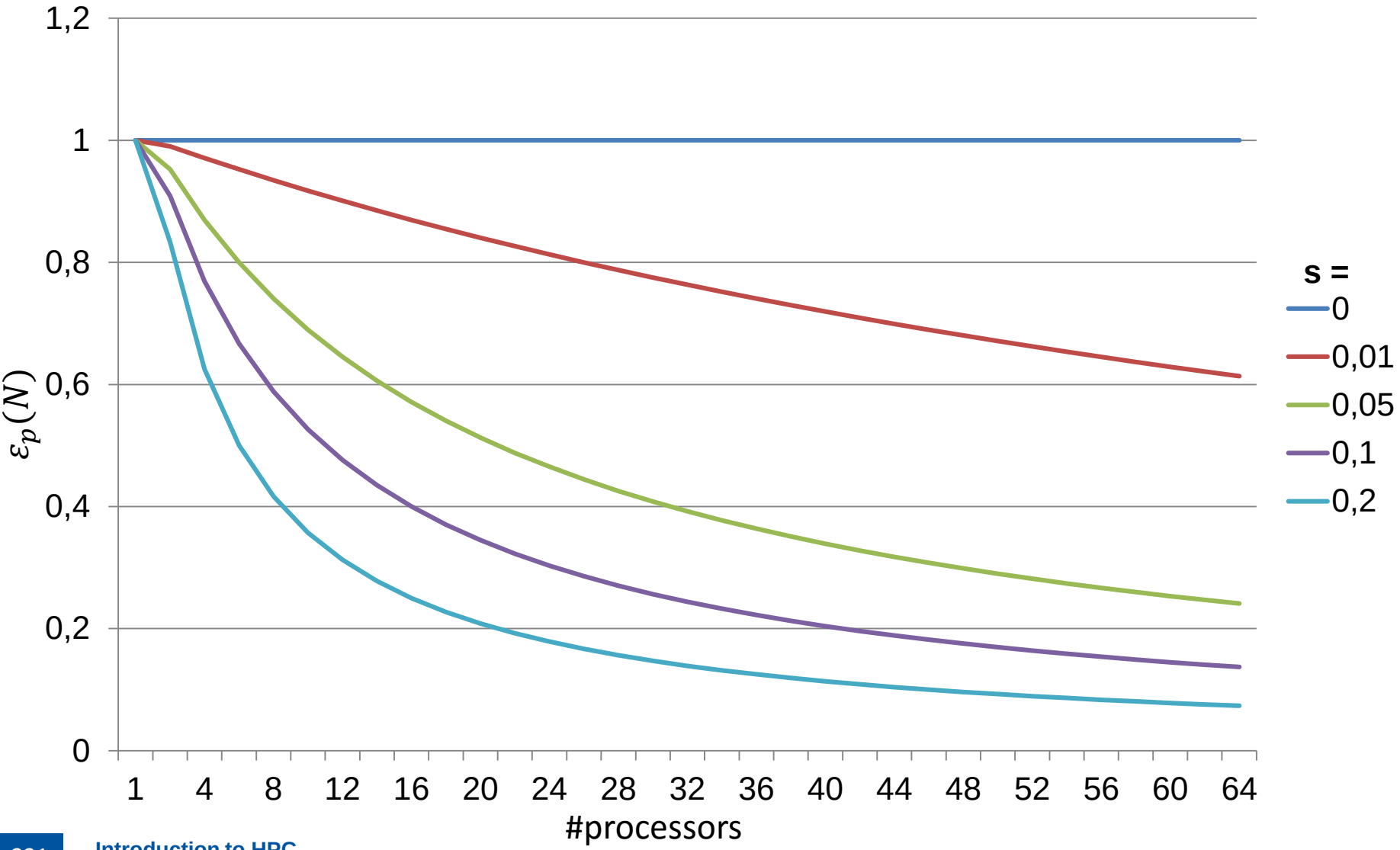
$$S_{0.95}(N) = \frac{T(1)}{T(N)} = \frac{1}{0.05 + \frac{1 - 0.05}{N}}$$



Speedup examples with varying *s*



Efficiency examples with varying *s*

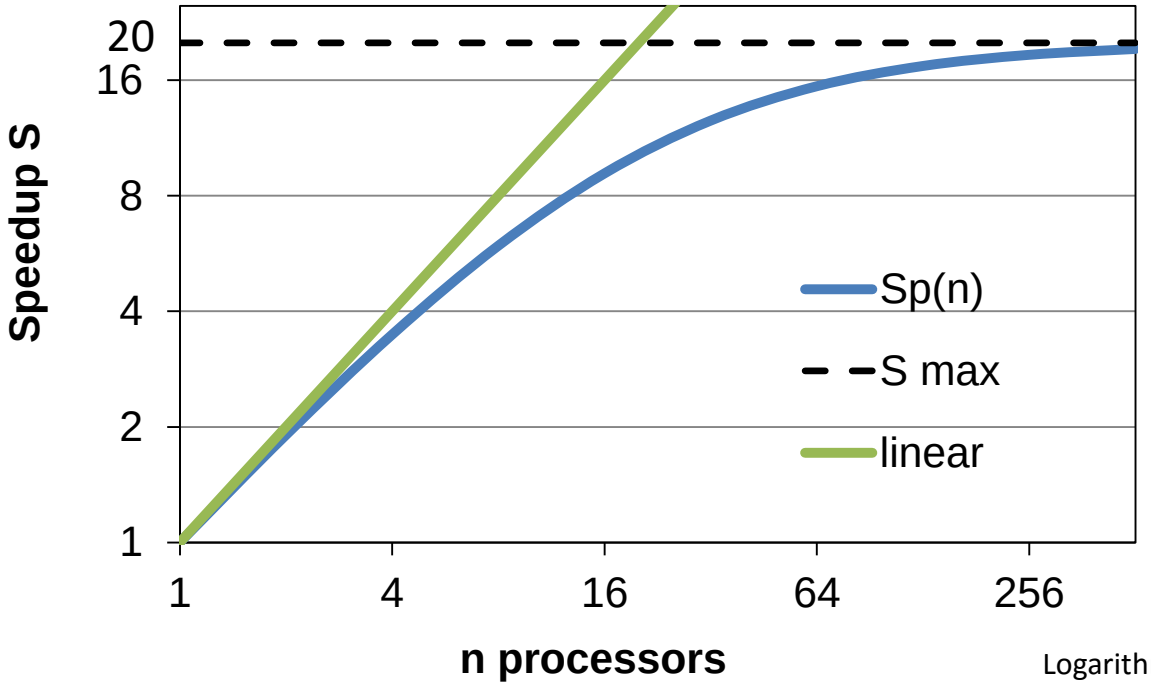


Limit value consideration of S_p

$$S_p(N) = \frac{T(1)}{T(N)} = \frac{1}{s + \frac{1-s}{N}}$$

$$S_{max} = \lim_{N\to\infty} S_p(N) = \lim_{N\to\infty} \frac{1}{s + \frac{1-s}{N}} = \frac{1}{s}$$

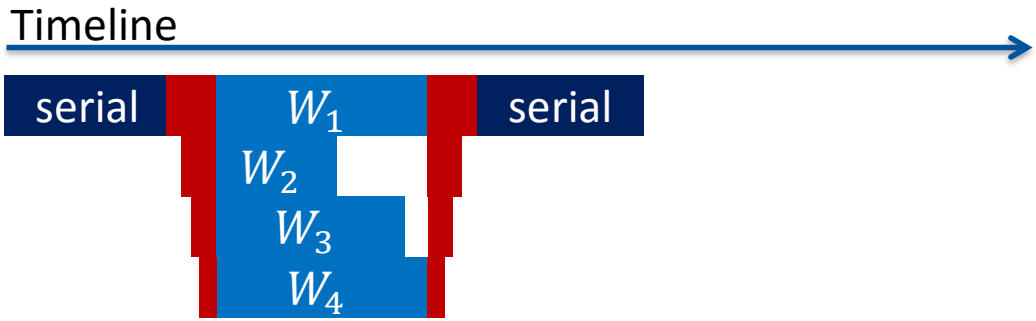
Example: s=0.05



Logarithmic scale!

Limitation of Amdahl's Law

- **Amdahl's law assumption**
  - → Tasks exist that are perfectly parallelizable
- **In reality, no task is perfectly parallelizable**
  - → Resources have to be shared. These mostly have to be used serially
  - → Dependencies between tasks exists which have to be communicated
  - → Generally tasks have to communicate information to each other



Limit value consideration of E_p

- **the benefit of parallelization may be limited**
  - → Limited scalability leads to inefficient utilization of available resources
  - → Parallel efficiency describes the utilization of available computing resources through parallelization:

$$\lim_{N\to\infty} \varepsilon_p(N) = \lim_{N\to\infty} \frac{\mathbf{1}}{\boldsymbol{s}(\boldsymbol{N}-\mathbf{1})+\mathbf{1}} = 0$$

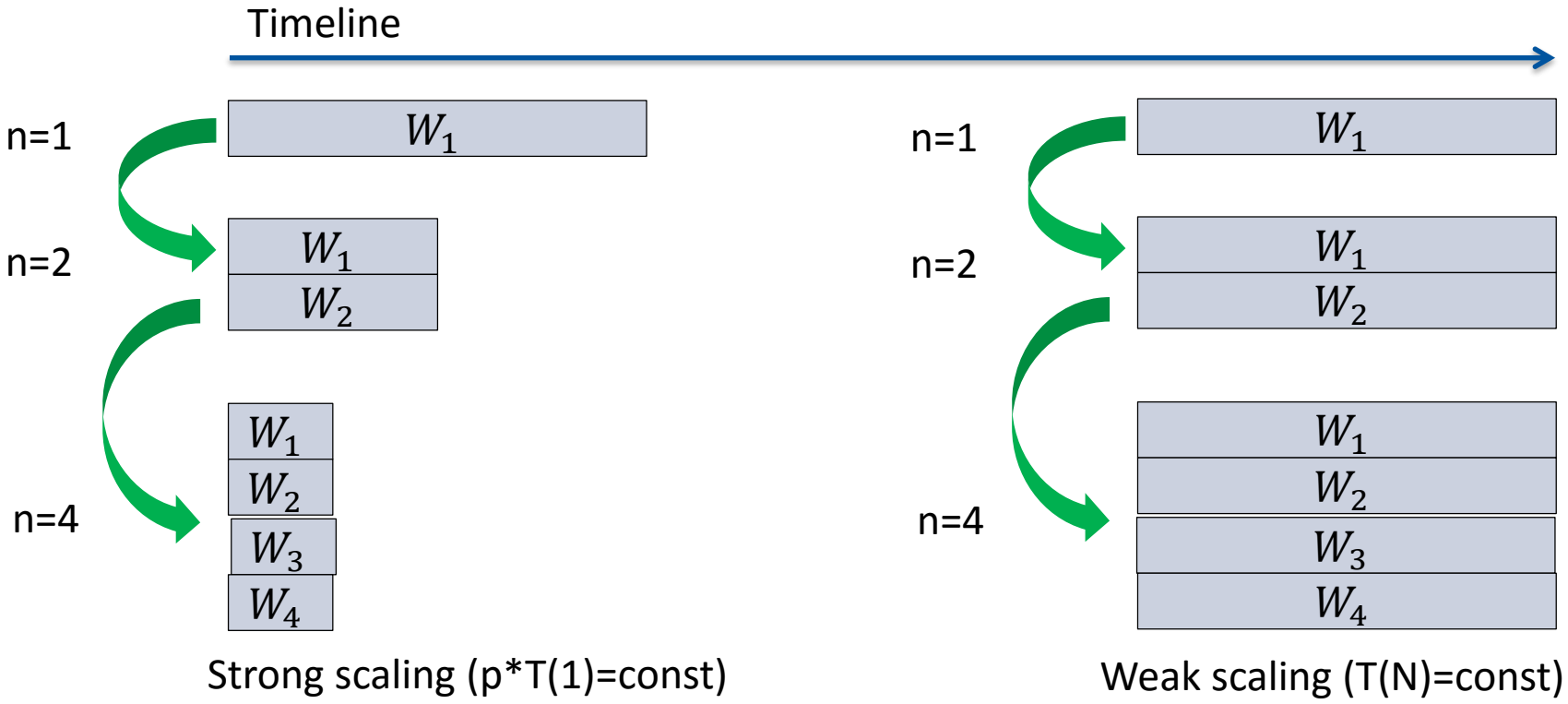
- **Problems of real world applications**
  - → Load imbalance
  - → Communication overhead
  - → Problem size usually varies with increasing number of processors

Outline

1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → **Basic limitations of par. computing**
     - → Amdahl's law
     - → **Gustafson's law**
   - → Multithreaded processors (SMT)
   - → Multicore processors
   - → Shared-memory computers
     - → Cache coherence
     - → UMA
     - → ccNUMA
   - → Distributed-memory computers
   - → Networks
     - → Basic performance characteristics
     - → Network topologies
       - → Buses
       - → Ring & fully connected networks
       - → Switched & fat-tree networks
       - → Mesh networks
     - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
9. Distributed-memory programming with MPI
10. Shared-memory programming with OpenMP
11. Hybrid programming (MPI + OpenMP)
12. Heterogeneous architectures (GPUs, Xeon Phis)
13. Energy efficiency

Weak scaling vs. strong scaling

- **Gustafson's law**
  - → Addresses the assumption of a fixed Data set, which Amdahl's law is based on
  - → Problem size changes for weak scaling, fixed runtime (sketched below)



Gustafson's Law

- **Gustafson's law**
  - → Computations involving an arbitrarily large data set can be efficiently parallelized
  - → Counterpoint to Amdahl's law, which puts a limit on the speedup of computations involving a fixed-size data set
- **Assumption: execution time of a program can be decomposed into**

$$s + p = (1 - p) + p = 1$$

1 - p:	serial execution time
p:	parallel time for any of N processors

  - → The key assumption here is that the work that needs to be done varies linearly with the number of involved processors

→ **Time that a single processor needs:** $(1 - p) + Np = p(N - 1) + 1$

Gustafson's Law

- **Speedup with Gustafson's law:**

$$S_p(N) = \frac{T(1)}{T(N)} = \frac{(1-p)+Np}{(1-p)+p} = Np + s$$

"weak scaling"

- **And efficiency:**

$$\varepsilon_p(N) = \frac{S_p(N)}{N} = \frac{(1-p)}{N} + p$$

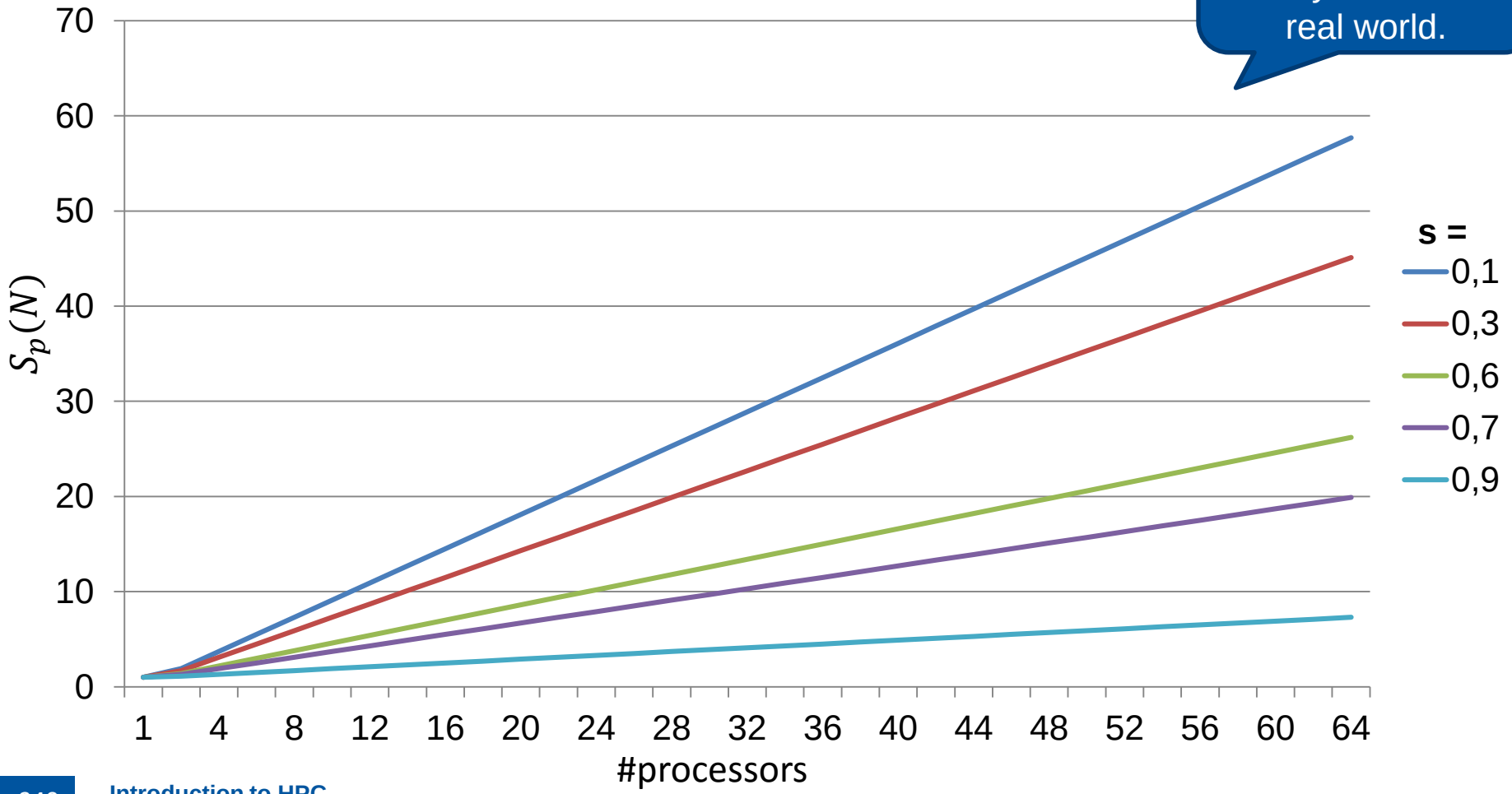
- **Implications**
  - → p (=fraction of time that the program executes in parallel) is held fixed
  - → (while) the number of processors N is varied
  - → With increasing number of processors, the speedup grows linearly
  - → The serial part becomes more and more unimportant with more processors involved.

Further reading: John L. Gustafson: Reevaluating Amdahl's law. In: Commun. ACM. 31, Nr. 5, 1988, S. 532–533, doi:10.1145/42411.42415

Speedup examples with varying *s*

$$\lim_{N\to\infty} S_p(N) = \lim_{N\to\infty} (\boldsymbol{Np} + \boldsymbol{s}) = \infty$$

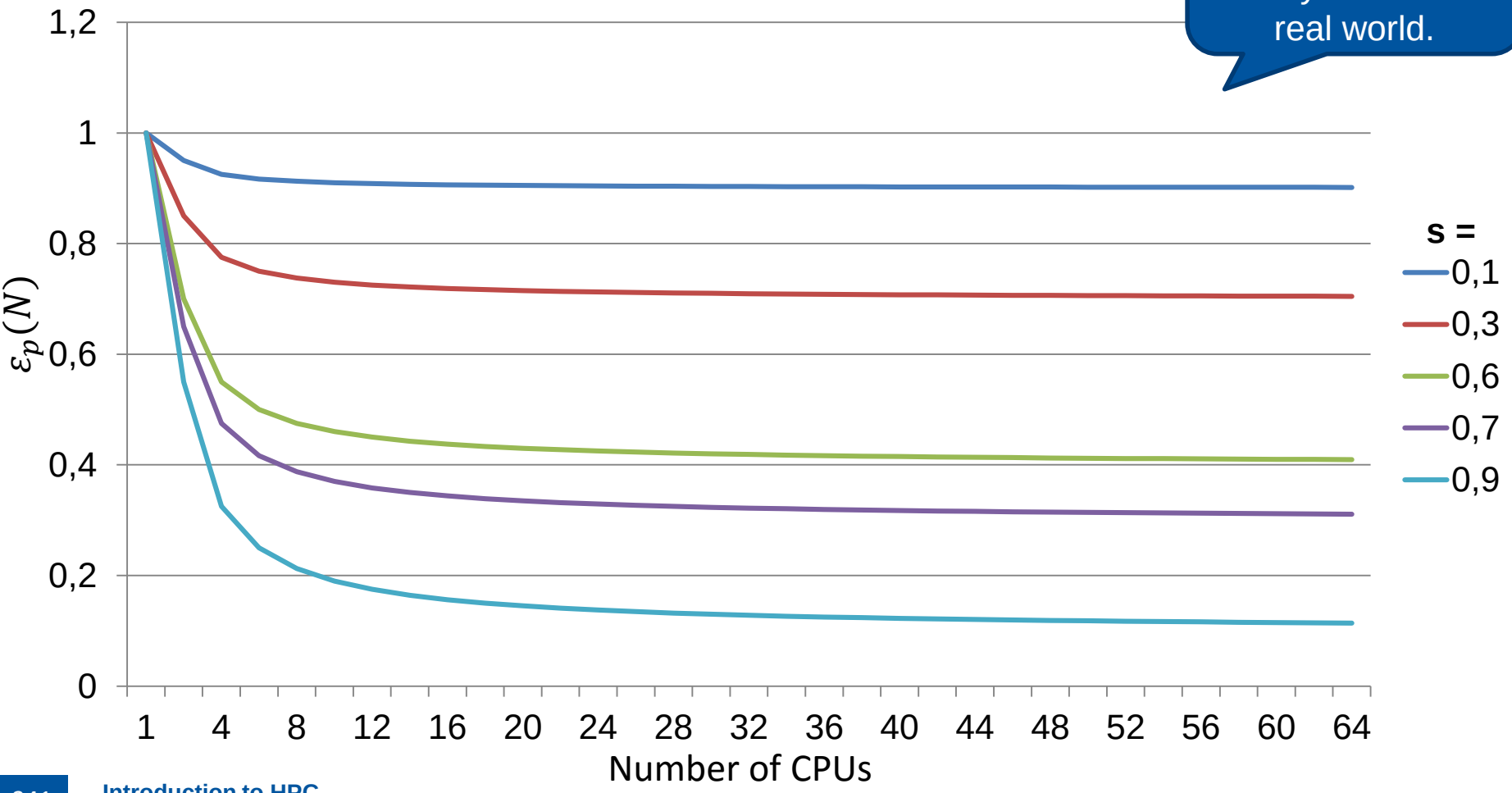
These speedup measurements are usually not done in real world.



Efficiency examples with varying *s*

$$\lim_{N\to\infty} \varepsilon_p(N) = \lim_{N\to\infty} \left(\frac{(\mathbf{1}-\boldsymbol{p})}{\boldsymbol{N}} + \boldsymbol{p}\right) = p$$

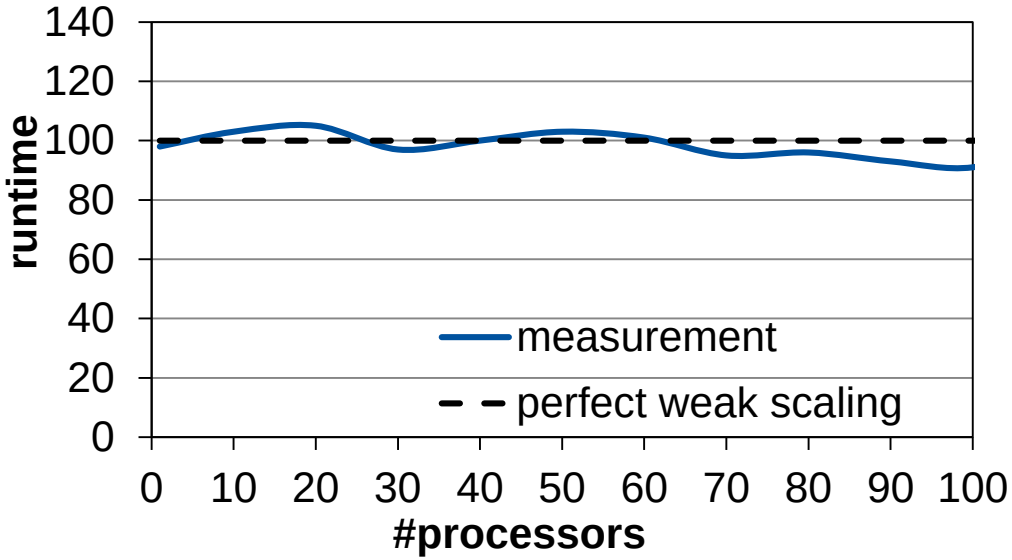
These efficiency measurements are usually not done in real world.



Real-world example

- **In real world, you usually measure runtimes**
  - → Increase data set with increasing number of processors

- **Perfect weak scaling: constant runtime among varying #processors**



General considerations

- **The serial fraction of execution time depends on the algorithm**

- **In practice, "s" is determined empirically**
  - → Non parallelizable parts of the application
  - → I/O (e.g. when reading input data)
  - → Communication pattern of the application
  - → Worksharing (load balance) – while both Amdahl's and Gustafson's Law assume perfect load balance
  - → Processor details: cache, memory bandwidth, etc..
  - → Implementation of the parallel runtime (OpenMP + MPI)

Quiz: What you have learnt

Question 1: Determine the code balance B_c of the loop nest of the following code. Assume that a store goes directly to the main memory (no temporal stores).

```
double a[N][N];
double b[N];
double c;

for (int i=0; i<N; ++i) {
   for (int j=0; j<N; ++j) {
      a[i][j] = b[j] * b[j] * c;
   }
}
```

a) $B_c = \frac{3 \text{ words}}{3 \text{ Flops}}$

b) $B_c = \frac{3 \text{ words}}{2 \text{ Flops}}$

c) $B_c = \frac{2 \text{ words}}{2 \text{ Flops}}$

d) $B_c = \frac{4 \text{ words}}{2 \text{ Flops}}$

Quiz: What you have learnt

Question 1: Determine the code balance B_c of the loop nest of the following code. Assume that a store goes directly to the main memory (no temporal stores).

```
double a[N][N];
double b[N];
double c;

for (int i=0; i<N; ++i) {
   for (int j=0; j<N; ++j) {
      a[i][j] = b[j] * b[j] * c;
   }
}
```

1 store 1 load kept in a register

c) $B_c = \frac{2 \text{ words}}{2 \text{ Flops}} = 1 \frac{\text{word}}{\text{Flop}}$

Quiz: What you have learnt

Question 2: Determine the theoretical double precision peak performance (Flop/s) of the following exemplary architecture:

- → 10 cores, 3.0 GHz clock frequency
- → Fused-Multiply add (FMA): 1 ADD & 1 MULT per cycle
- → width of vector registers: 256 bit
- → One double precision value takes 64 bit

a) $10 \cdot 3.0\,\text{GHz} \cdot 2 = 60\,\text{GFlop/s}$

b) $10 \cdot 3.0\,\text{GHz} \cdot 4 = 120\,\text{GFlop/s}$

c) $10 \cdot 3.0\,\text{GHz} \cdot 8 = 240\,\text{GFlop/s}$

d) $10 \cdot 3.0\,\text{GHz} \cdot 512 = 15.36\,\text{TFlop/s}$

Quiz: What you have learnt

Question 2: Determine the theoretical double precision peak performance (Flop/s) of the following exemplary architecture:

- → 10 cores, 3.0 GHz clock frequency
- → Fused-Multiply add (FMA): 1 ADD & 1 MULT per cycle
- → width of vector registers: 256 bit
- → One double precision value takes 64 bit

c) $\#\text{cores} \cdot \text{frequency} \cdot \frac{\text{operations}}{\text{cycle}}$

$= 10 \cdot 3.0\ \text{GHz} \cdot 2 \cdot 4 = 240\ \text{GFlop/s}$

Quiz: What you have learnt

Question 3: From an experiment we know that for a given problem an application runs for 16 minutes with one thread and 4 minutes with 8 threads. What speedup and parallel efficiency does the parallel execution reach?

a) Speedup $= 1$, parallel efficiency $= 1$
b) Speedup $= 4$, parallel efficiency $= 0.5$
c) Speedup $= 4$, parallel efficiency $= 2$
d) Speedup $= 0.25$, parallel efficiency $= 0.5$

Quiz: What you have learnt

Question 3: From an experiment we know that for a given problem an application runs for 16 minutes with one thread and 4 minutes with 8 threads. What speedup and parallel efficiency does the parallel execution reach?

b) Speedup $S_p = \frac{T_s}{T_p} = \frac{16\ \text{min}}{4\ \text{min}} = 4$

Parallel efficiency = $\frac{S_p}{N} = \frac{4}{8} = 0.5$

Quiz: What you have learnt

Question 4: Given is an application with two parts A and B. The serial execution time of part A is 10 seconds and of part B 40 seconds. Only part B can be sped up by utilizing multiple threads. How many threads are required to achieve a speedup of 2 according to Amdahl's law?

a) 2 threads
b) 3 threads
c) 4 threads
d) 8 threads

Quiz: What you have learnt

Question 4: Given is an application with two parts A and B. The serial execution time of part A is 10 seconds and of part B 40 seconds. Only part B can be sped up by utilizing multiple threads. How many threads are required to achieve a speedup of 2 according to Amdahl's law?

b) $S_p(N) = \frac{1}{s+\frac{1-s}{N}} \rightarrow N = \frac{1-s}{\frac{1}{S_p(N)}-s} = \frac{1-0.2}{\frac{1}{2}-0.2} = 2.\overline{6}$

→ minimum 3 threads