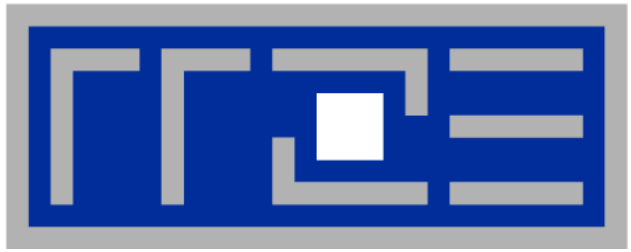
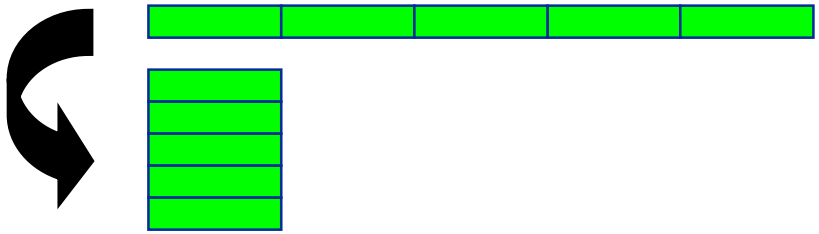


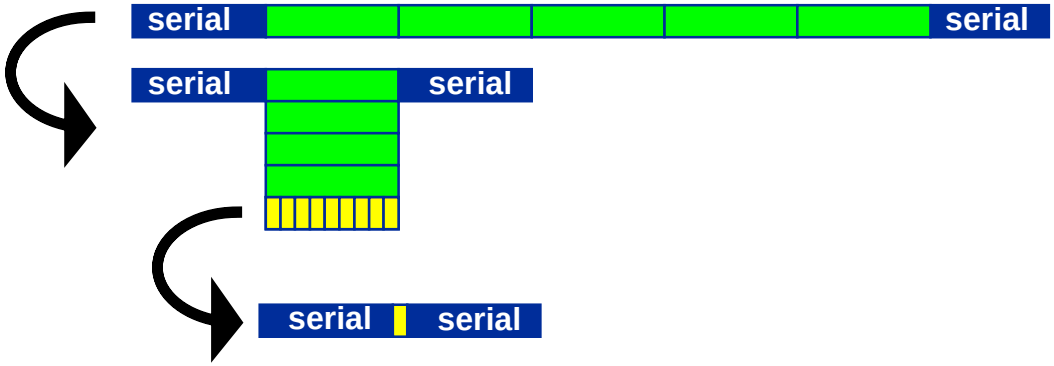
The costs of parallelization

- Amdahl‘s law
- Gustafson’s law
- OpenMP overhead

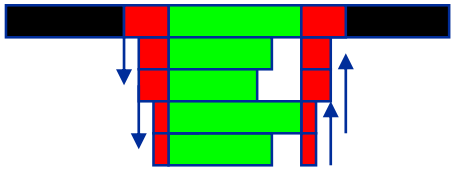
Limitations of Parallel Computing: Amdahl's Law



Ideal world:
All work is perfectly parallelizable

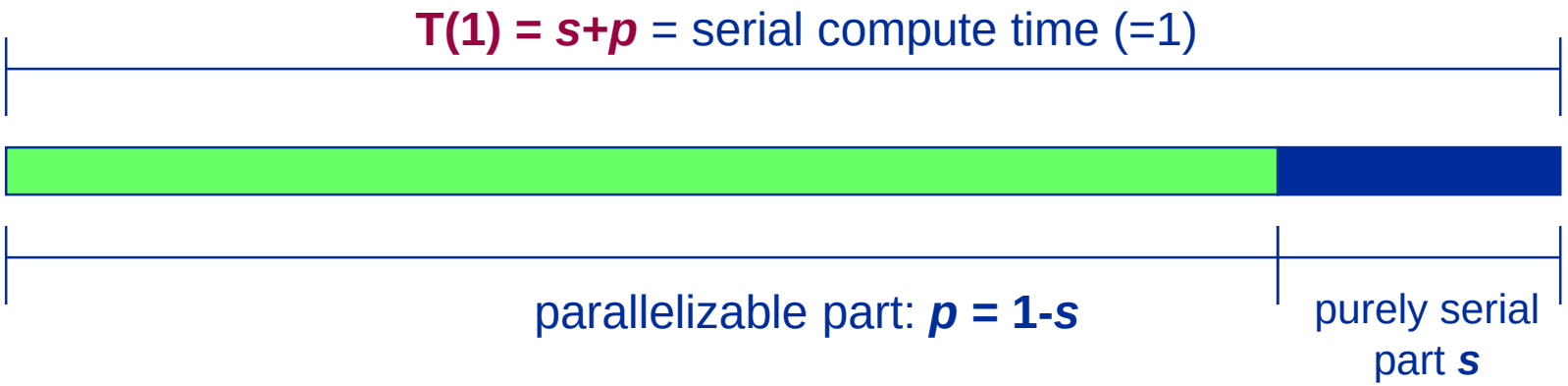


Closer to reality:
Purely serial parts
limit maximum speedup

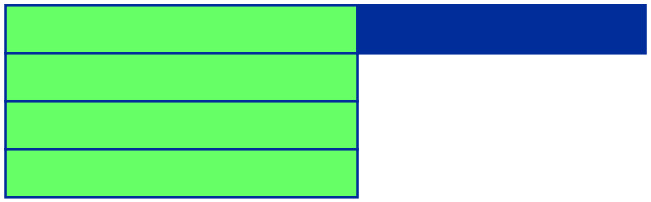


Reality is even worse:
Communication and synchronization
impede scalability even further

Limitations of Parallel Computing: Calculating Speedup in a Simple Model ("strong scaling")



Parallel execution time: T(*N*) = *s*+*p*/*N*

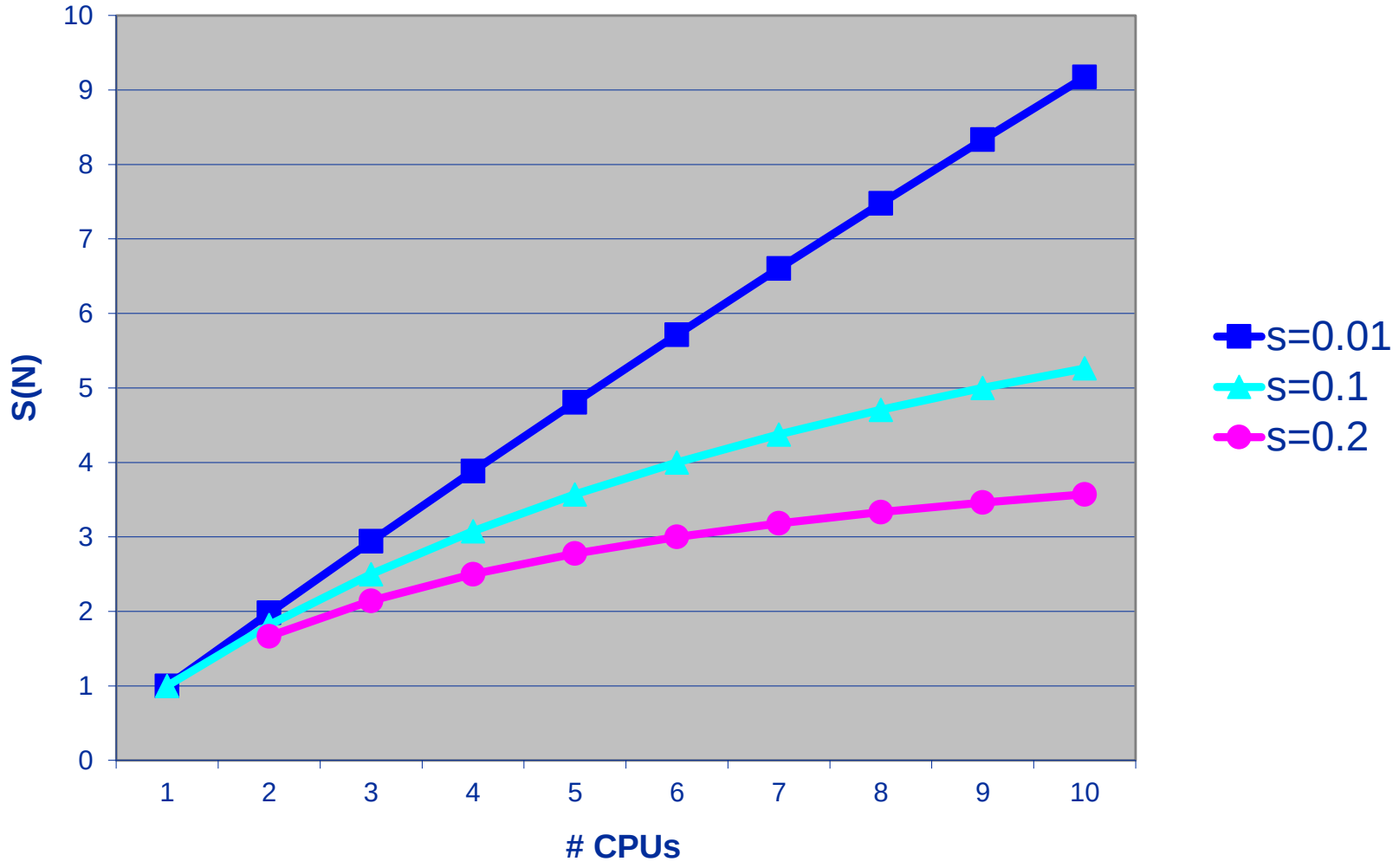


General formula for speedup: Amdahl's Law

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

This holds for constant problem size ("strong scaling„)

Limitations of parallel computing: Amdahl´s Law



- **Benefit of parallelization may be strongly limited**
  - **Limited scalability leads to inefficient use of resources**
  - **Metric: Parallel Efficiency**
    (*what percentage of the workers/processors is efficiently used*):

$$\varepsilon_p(N) = \frac{S_p(N)}{N}$$

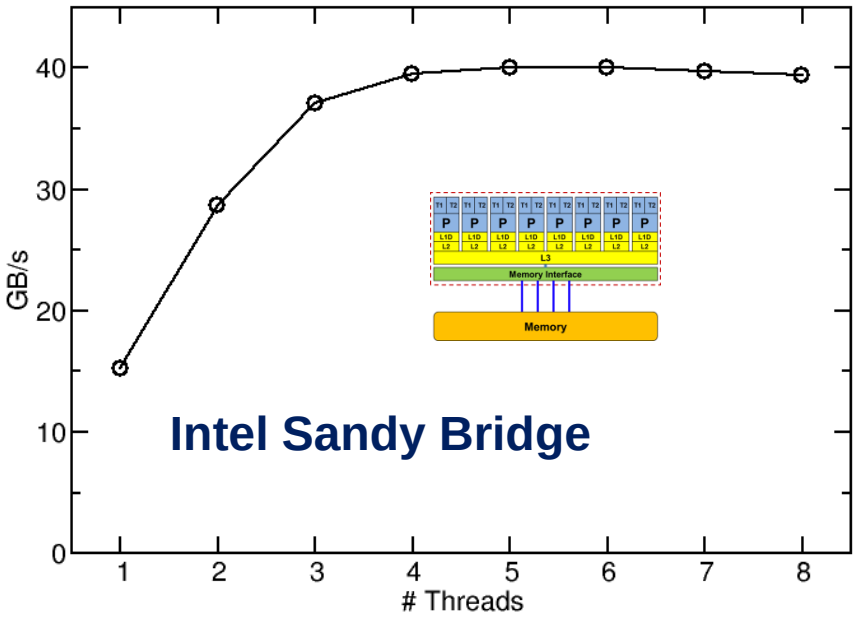
  - **Amdahl case:**

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

Limitations of Parallel Computing: Amdahl's Law ("strong scaling")

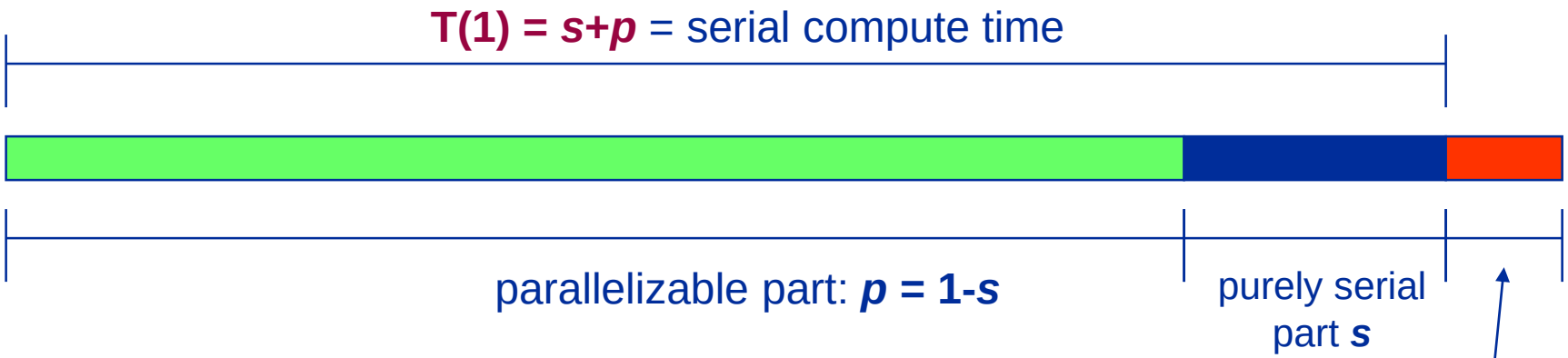
- Many important effects are not covered
  - Saturating hardware resources

    (e.g. main memory bandwidth limits scalability for vector triads; s=0)

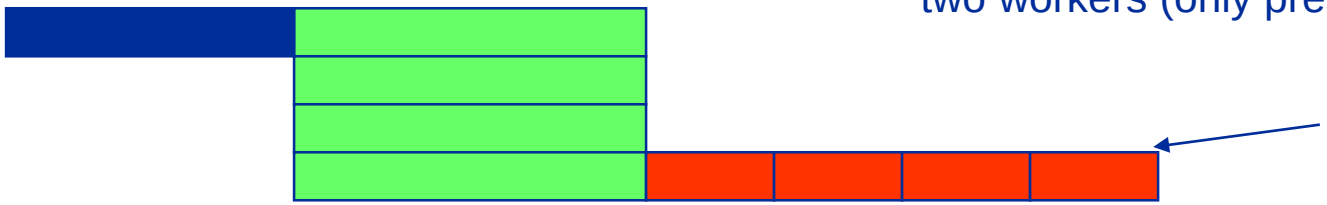


  - Communication overhead (but that can be modeled separately – see next slides)

Limitations of parallel computing: Strong Scaling with a Simple Communication Model



parallel: T(*N*) = *s*+*p*/*N*+*Nk*



fraction ***k*** for "communication" between each two workers (only present in parallel case)

Example:
$c(N) = kN$

General formula for speedup (worst case):

(*k*=0 → Amdahl's Law)

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

- **Large N limits**
  - Amdahl's Law predicts (c(N)=0)

$$\lim_{N\to\infty} S_p^0(N) = \frac{1}{s}$$

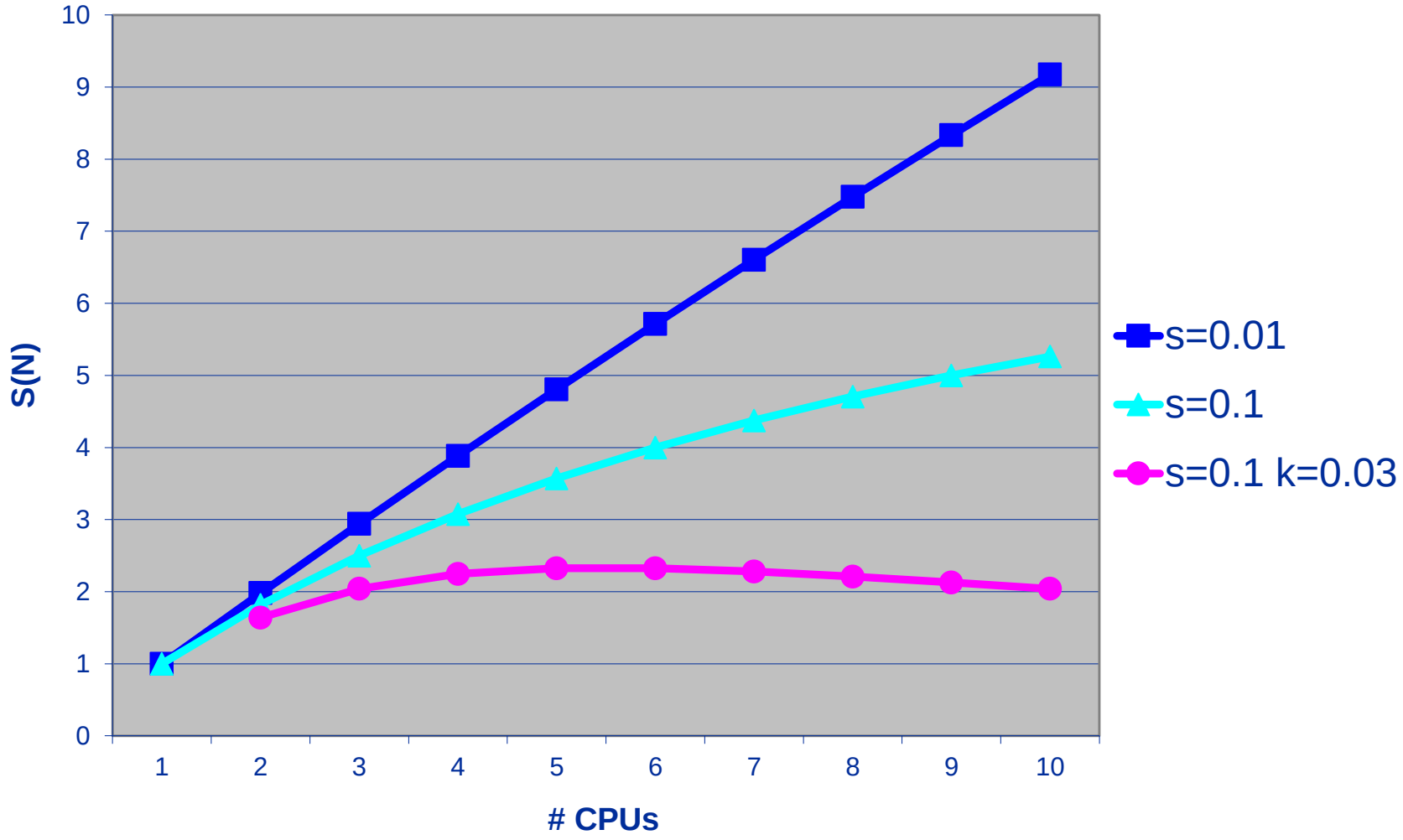
(independent of *N*)

  - For c(N) =k*N (k≠0), our simplified model of communication overhead yields a beavior of

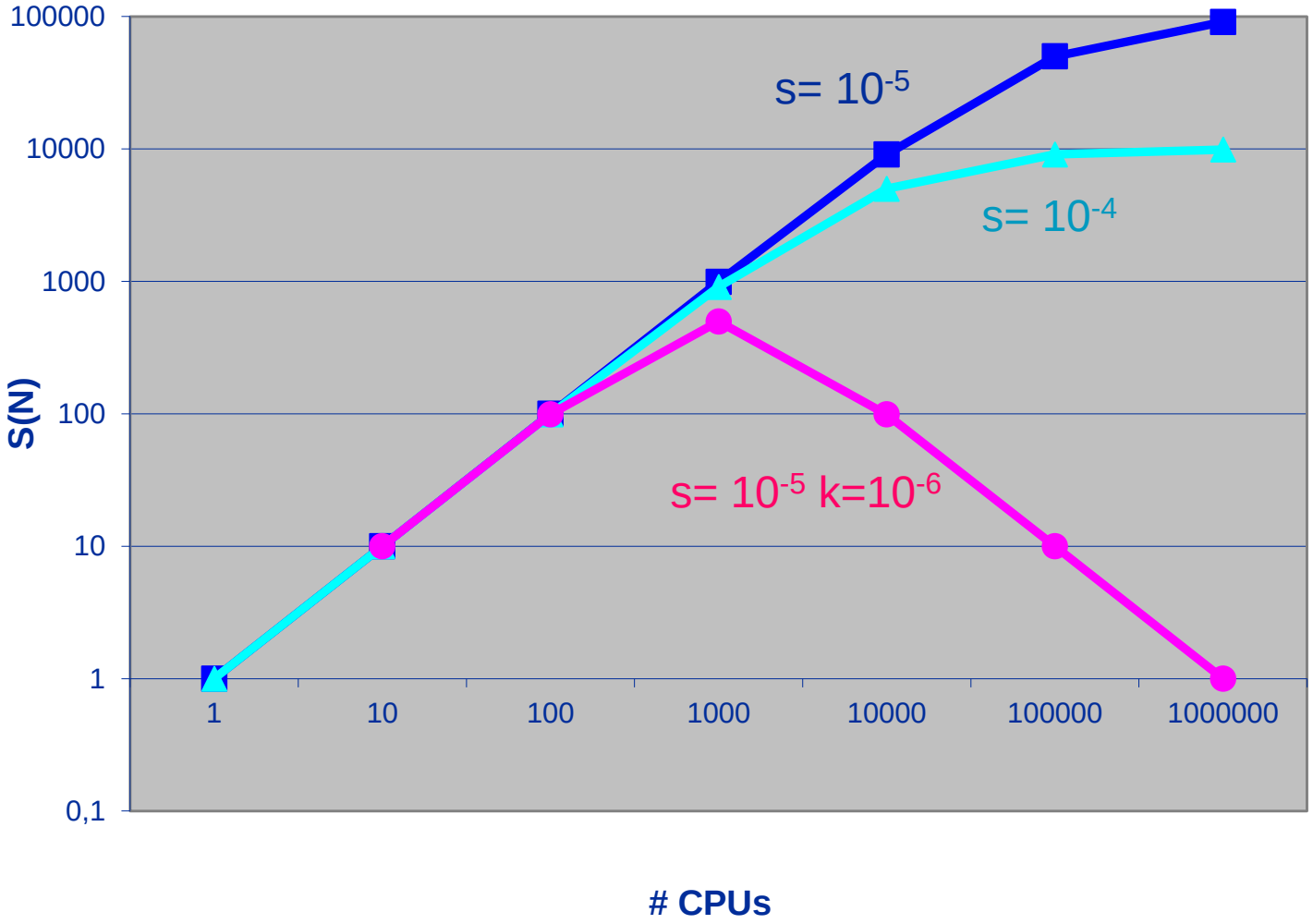
$$S_p^k(N) \xrightarrow{N>>1} \frac{1}{Nk}$$

- **In reality the situation gets even worse**
  - **Load imbalance**
  - **Pipeline startups**
  - **Task dependecies**

Limitations of parallel computing: Amdahl´s Law

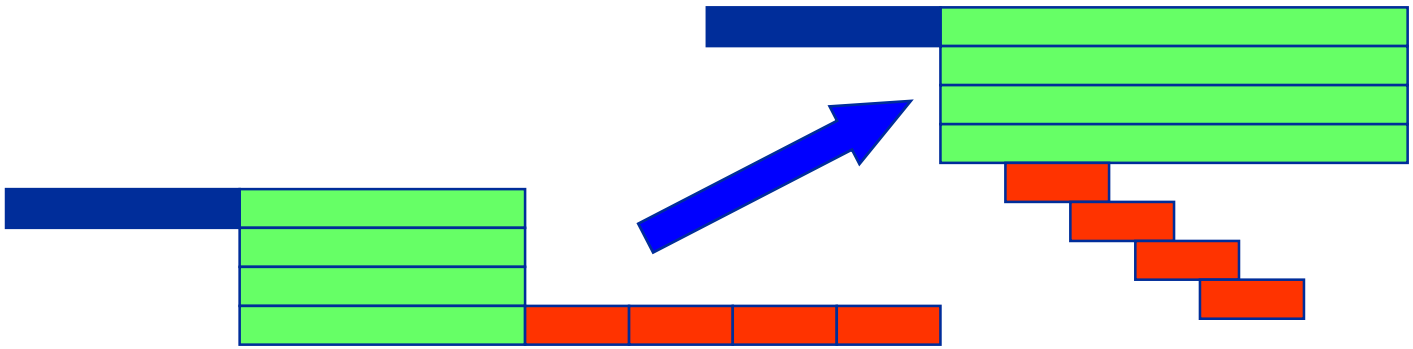


Limitations of parallel computing: Amdahl´s Law at scale

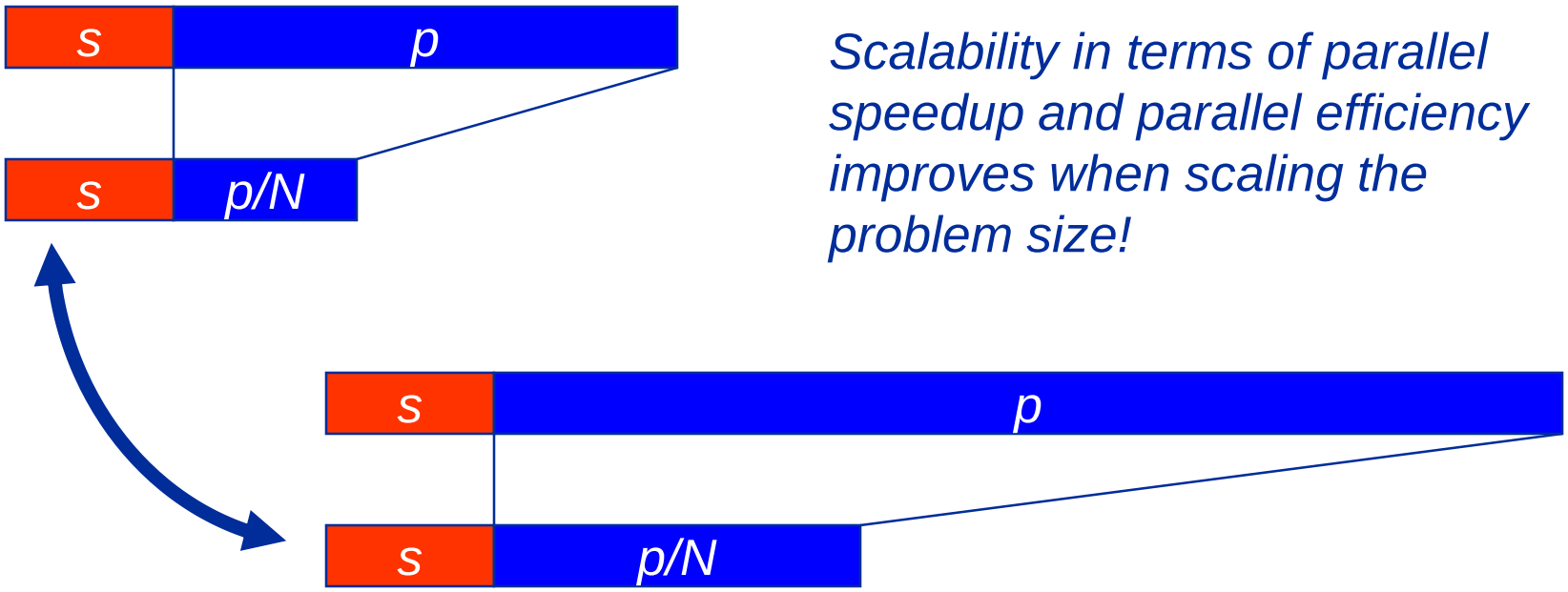


Limitations of parallel computing: Impact of communication is not always as bad…

- **Communication is not necessarily so hazardous**
  - Non-blocking networks can transfer many messages concurrently: $Nk \rightarrow k$
  - Some problems have more favorable communication overheads, e.g., $c(N) = kN^{-2/3}$ (cubic domain decomposition)
  - Sometimes we want to increase the problem size instead of reducing the runtime
  - Communication may be overlapped with useful work :



- **Increasing problem size often mainly/only enlarges the parallel fraction *p***
  - Assume *p* scales with problem size while s stays constant
  - → Fraction of *s* relative to overall runtime decreases



Scalability in terms of parallel speedup and parallel efficiency improves when scaling the problem size!

Limitation of parallel computing: Increasing Parallel Efficiency ("*weak scaling*")

- **When scaling a problem to more workers, the amount of work will often be scaled as well**
  - Enables finer "resolution" or larger "problem size"
  - **Let *s* and *p* be the serial and parallel fractions so that *s*+*p*=1**
  - **Perfect situation: runtime stays constant while *N* increases**
  - "Performance Speedup" **=**

$$\frac{\text{work/time for problem size } N \text{ with } N \text{ workers}}{\text{work/time for problem size 1 with 1 worker}}$$

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

Gustafsson's Law ("weak scaling")

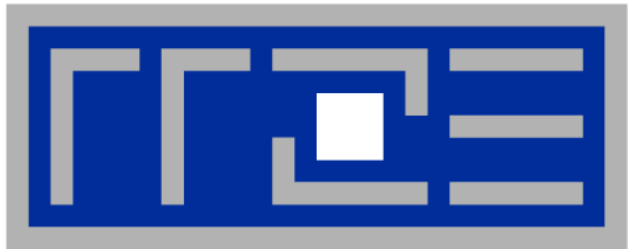
- Linear in *N* – but closely observe the meaning of the word "work"!

Limitations of parallel computing: Serial & Parallel fraction

Serial fraction *s* may depend on

- **Program / algorithm**
  - Non-parallelizable part, e.g. recursive data setup
  - Non-parallelizable IO, e.g. reading input data
  - Communication structure
  - Load balancing (assumed so far: perfect balanced)
  - …
- **Computer hardware**
  - Processor: Cache effects & memory bandwidth effects
  - Parallel Library; Network capabilities; Parallel IO
  - …

Determine *s* "experimentally":
Measure speedup and fit data to performance model



Parallelization: Potential overheads and costs

There is no free lunch…
OpenMP overhead

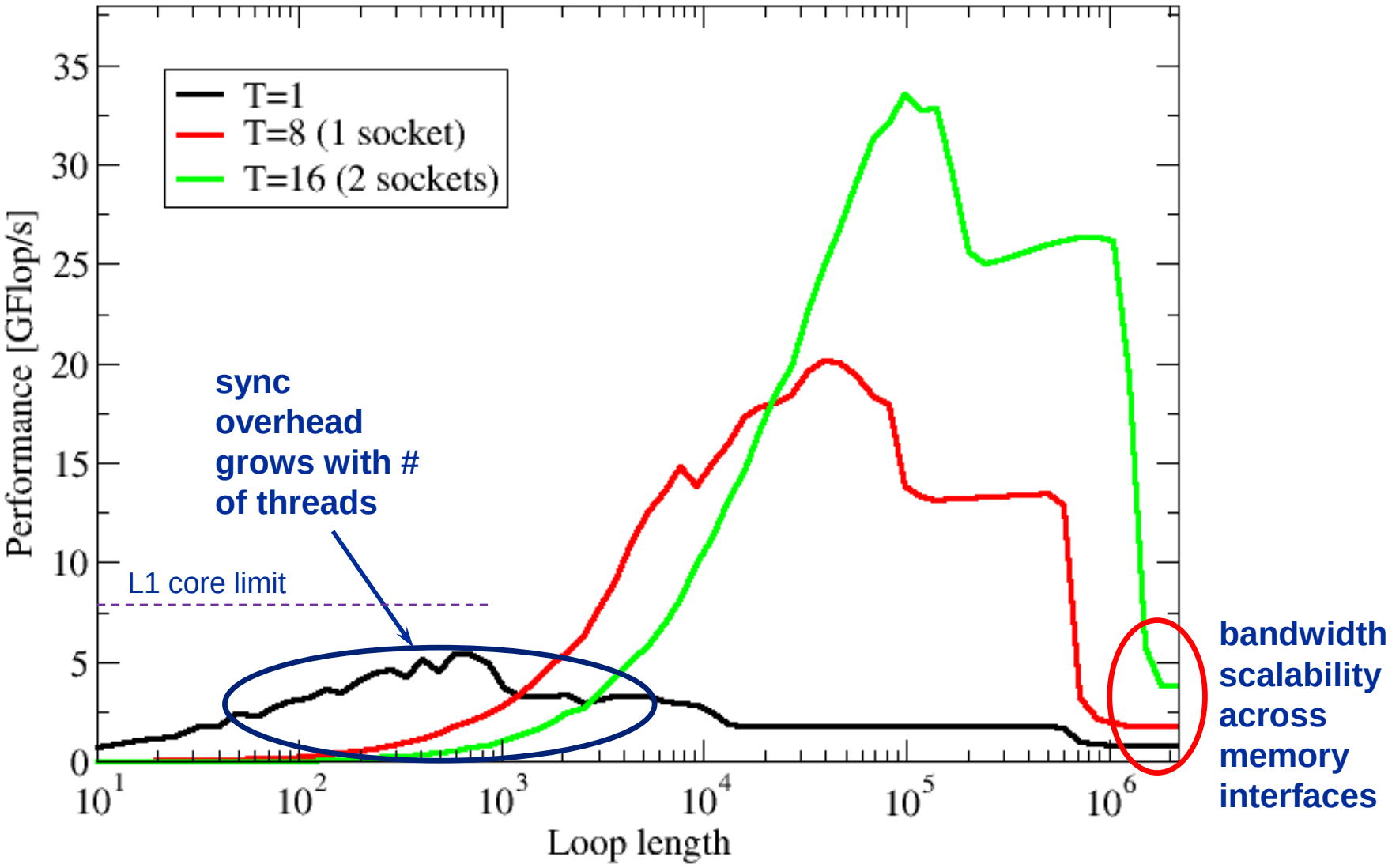
OpenMP work sharing in the benchmark loop

```
double precision, dimension(:), allocatable :: A,B,C,D

allocate(A(1:N),B(1:N),C(1:N),D(1:N))
A=1.d0; B=A; C=A; D=A
!$OMP PARALLEL private(i,j)
do j=1,NITER
!$OMP DO
  do i=1,N
    A(i) = B(i) + C(i) * D(i)
  enddo
!$OMP END DO
  if(.something.that.is.never.true.) then
    call dummy(A,B,C,D)
  endif
enddo
!$OMP END PARALLEL
```

Implicit barrier

OpenMP vector triad on Sandy Bridge socket (3 GHz)



```
!$OMP PARALLEL …
…
!$OMP BARRIER
!$OMP DO
…
!$OMP ENDDO
!$OMP END PARALLEL
```

Threads are synchronized at **explicit** AND **implicit** barriers. These are a main source of overhead in OpenMP progams.

Determine costs via modified OpenMP Microbenchmarks testcase (epcc)

On x86 systems there is no hardware support for synchronization!

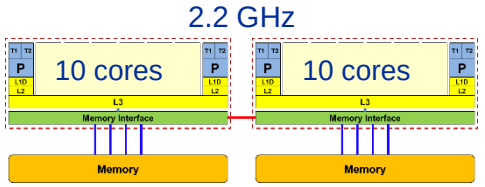
- **Next slide: Test OpenMP Barrier performance…**
- **for different compilers**
- **and different topologies:**
  - **shared cache**
  - **shared socket**
  - **between sockets**
- **and different thread counts**
  - **2 threads**
  - **full domain (chip, socket, node)**

Thread synchronization overhead on IvyBridge-EP

Barrier overhead in CPU cycles

2 Threads	Intel 16.0	GCC 5.3.0
Shared L3	**599**	**425**
SMT threads	**612**	**423**
Other socket	**1486**	**1067**

Strong topology dependence!



Full domain	Intel 16.0	GCC 5.3.0
Socket (10 cores)	1934	1301
Node (20 cores)	**4999**	**7783**
Node +SMT	5981	9897

Overhead grows with thread count

- **Strong dependence on compiler, CPU and system environment!**
- `OMP_WAIT_POLICY=ACTIVE` **can make a big difference**

Thread synchronization overhead on Xeon Phi 7210 (64-core)

Barrier overhead in CPU cycles (Intel C compiler 16.03)

2 threads on distinct cores: 730

	SMT1	SMT2	SMT3	SMT4
One core	n/a	**963**	1580	2240
Full chip	5720	8100	9900	**11400**

Still the pain may be much larger, as more work can be done in one cycle on Phi compared to a full Ivy Bridge node

3.2x cores (20 vs 64) on Phi

4x more operations per cycle per core on Phi

→ 4 · 3.2 = 12.8x more work done on Xeon Phi per cycle

1.9x more barrier penalty (cycles) on Phi (11400 vs. 6000)

→ One barrier causes 1.9 · 12.8 ≈ 24x more pain ☺.