

Parallelism

Understanding parallelism

Basic limitations of parallel computing

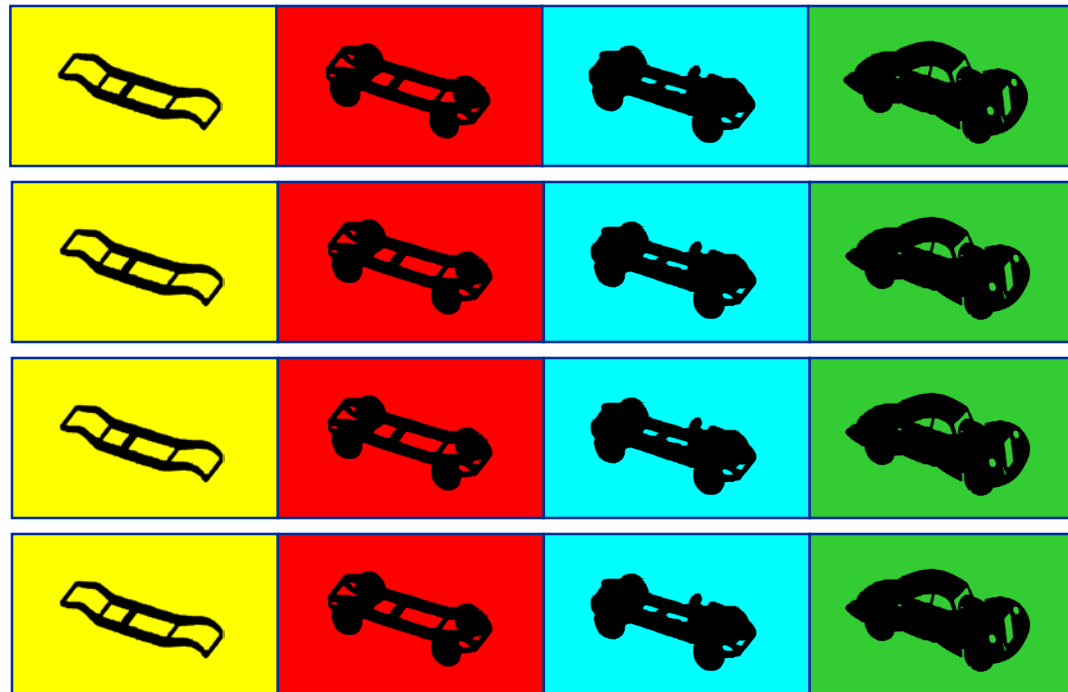
Prof. Dr. G. Wellein^(a,b) , Dr. G. Hager^(a), J. Hammer^(b), C.L. Alappat^(b)

^(a)HPC Services – Regionales Rechenzentrum Erlangen

^(b)Department für Informatik

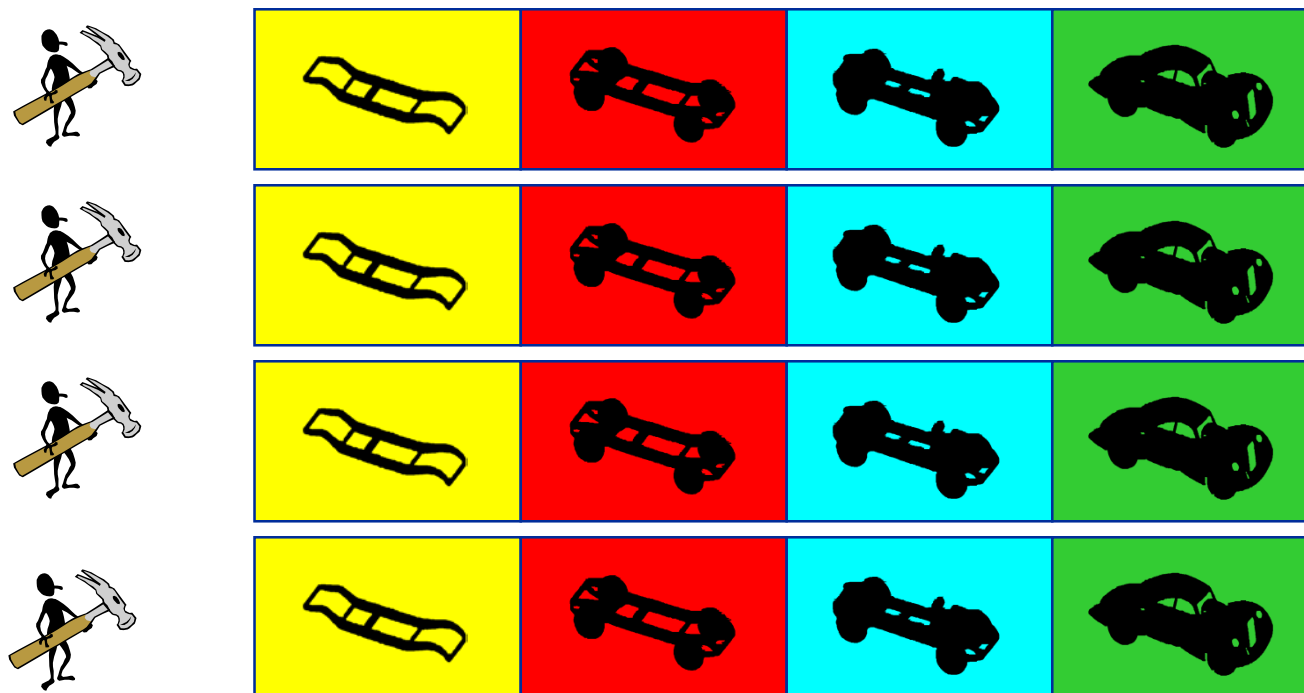
University Erlangen-Nürnberg, Sommersemester 2019

Understanding Parallelism: Sequential work



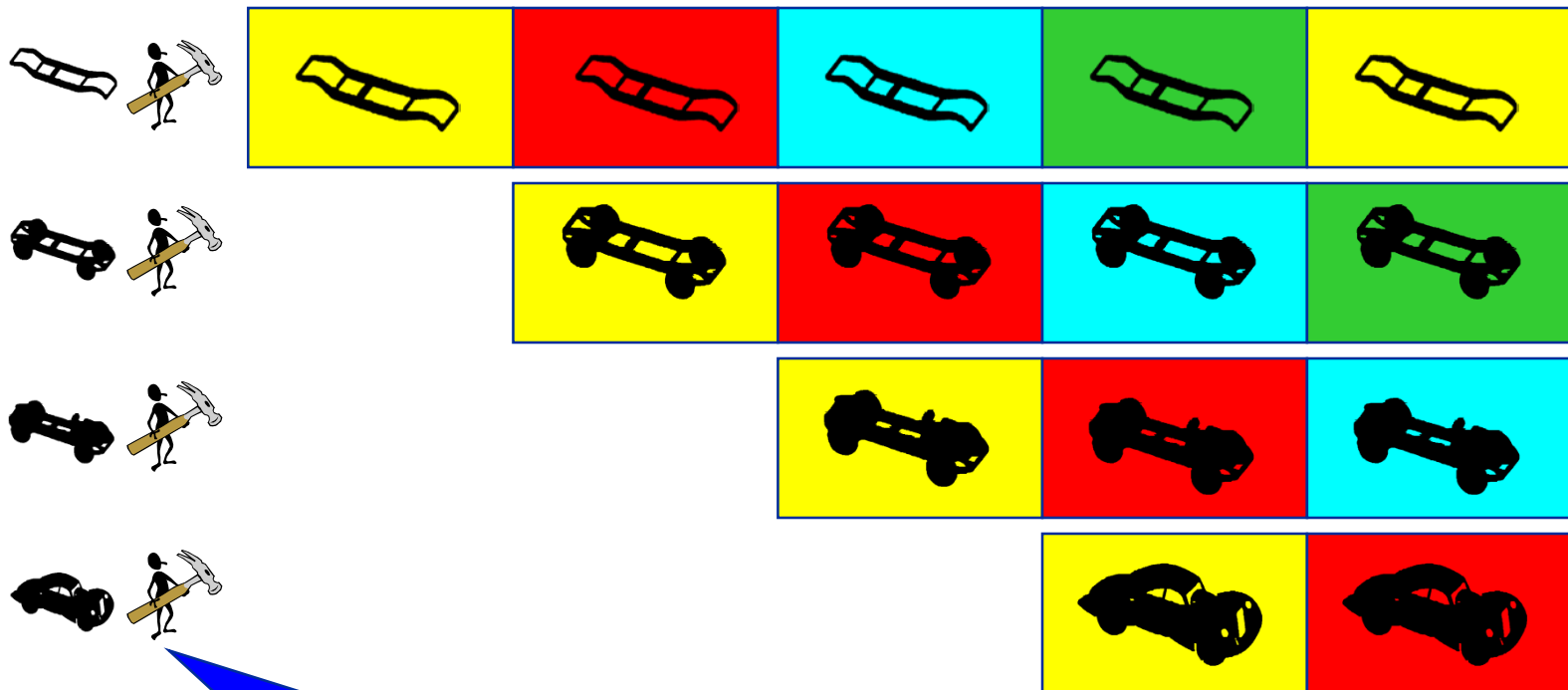
After 16 time steps: 4 cars

Understanding Parallelism: Parallel work: A straightforward approach



After 4 time steps: 4 cars

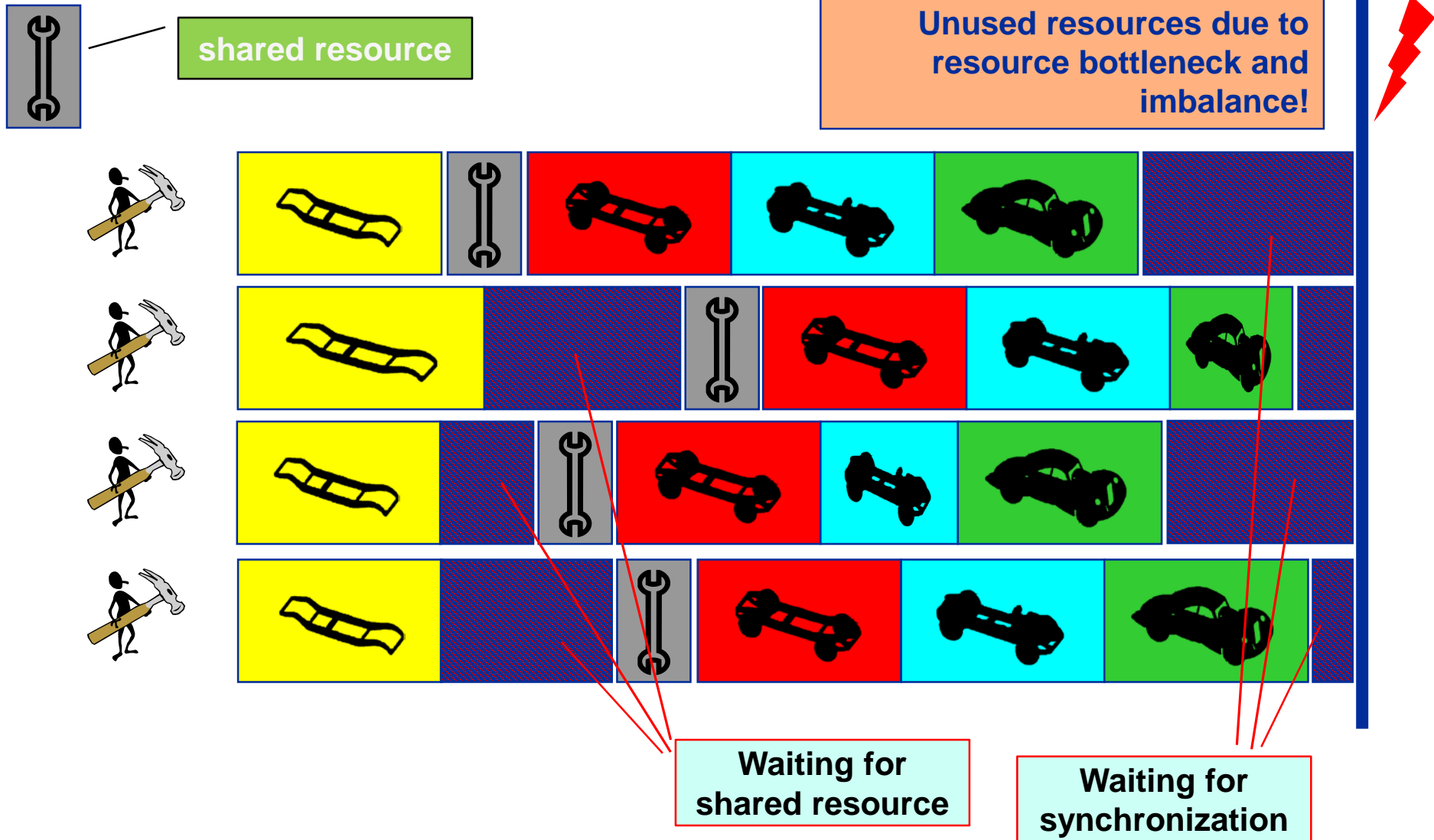
Understanding Parallelism: Parallel work: Pipelining

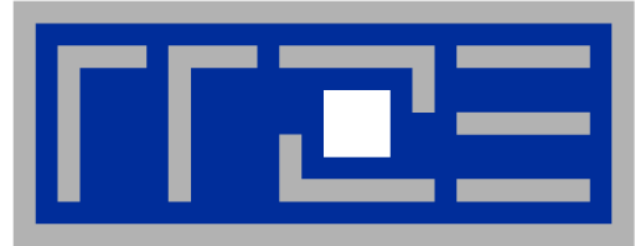


specialized workers

After wind-up phase: 1 car per time step at lower cost!

Understanding parallelism: Limits of scalability: shared resources + imbalance





Basic limitations of parallel computing

Amdahl's law (“strong scaling”)

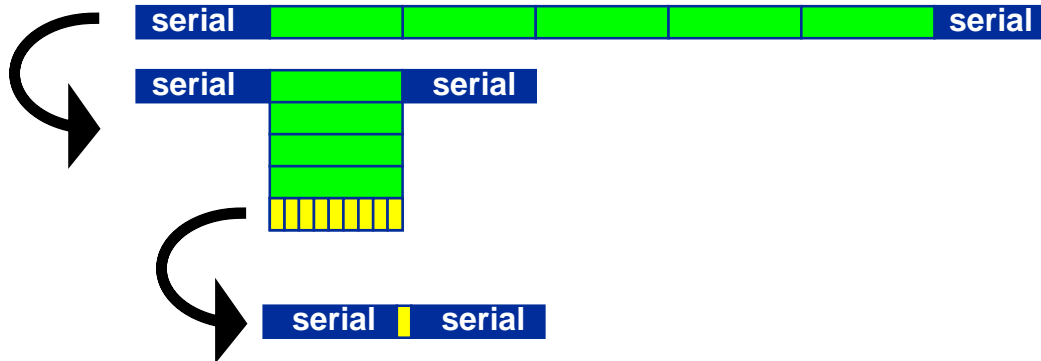
Gustafson's law (“weak scaling”)

Applying Amdahl's law

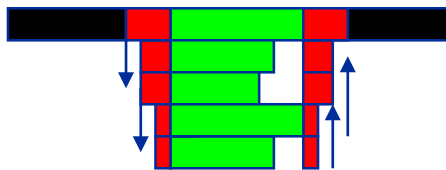
Limitations beyond Amdahl/Gustafson



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: Metrics to quantify the efficiency of parallel computing



- Assume $T(N)$ is the time to execute „some workload“ with N workers
- How much faster do I execute the given workload on N workers?

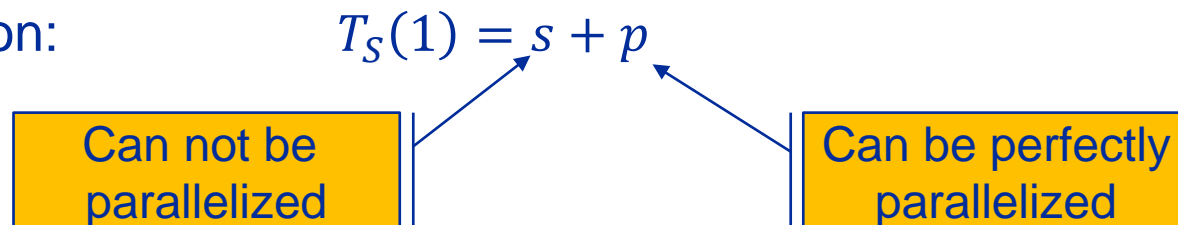
→ Parallel Speed-Up: $S_P(N) = \frac{T(1)}{T(N)}$

- How efficient do I use the workers in average?

→ Parallel Efficiency: $\varepsilon_P(N) = \frac{S_P(N)}{N}$

- Time to execute the serial ($N = 1$) workload on one worker: $T_S(1) = 1$

- Basic assumption:

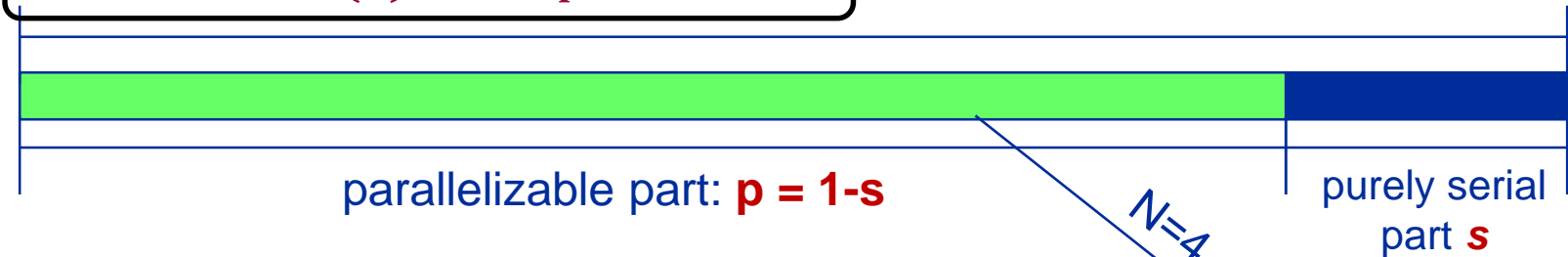


Limitations of Parallel Computing: Calculating Speedup in a Simple Model (“strong scaling”)



Assumption: Constant workload („strong scaling“)
→ $T(1) = T_s(1)$

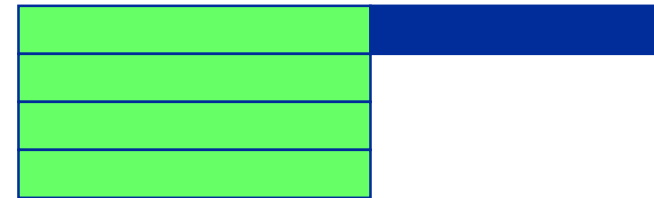
One worker: $T(1) = s + p = 1$



N workers: $T(N) = s + p/N$

Purely Serial

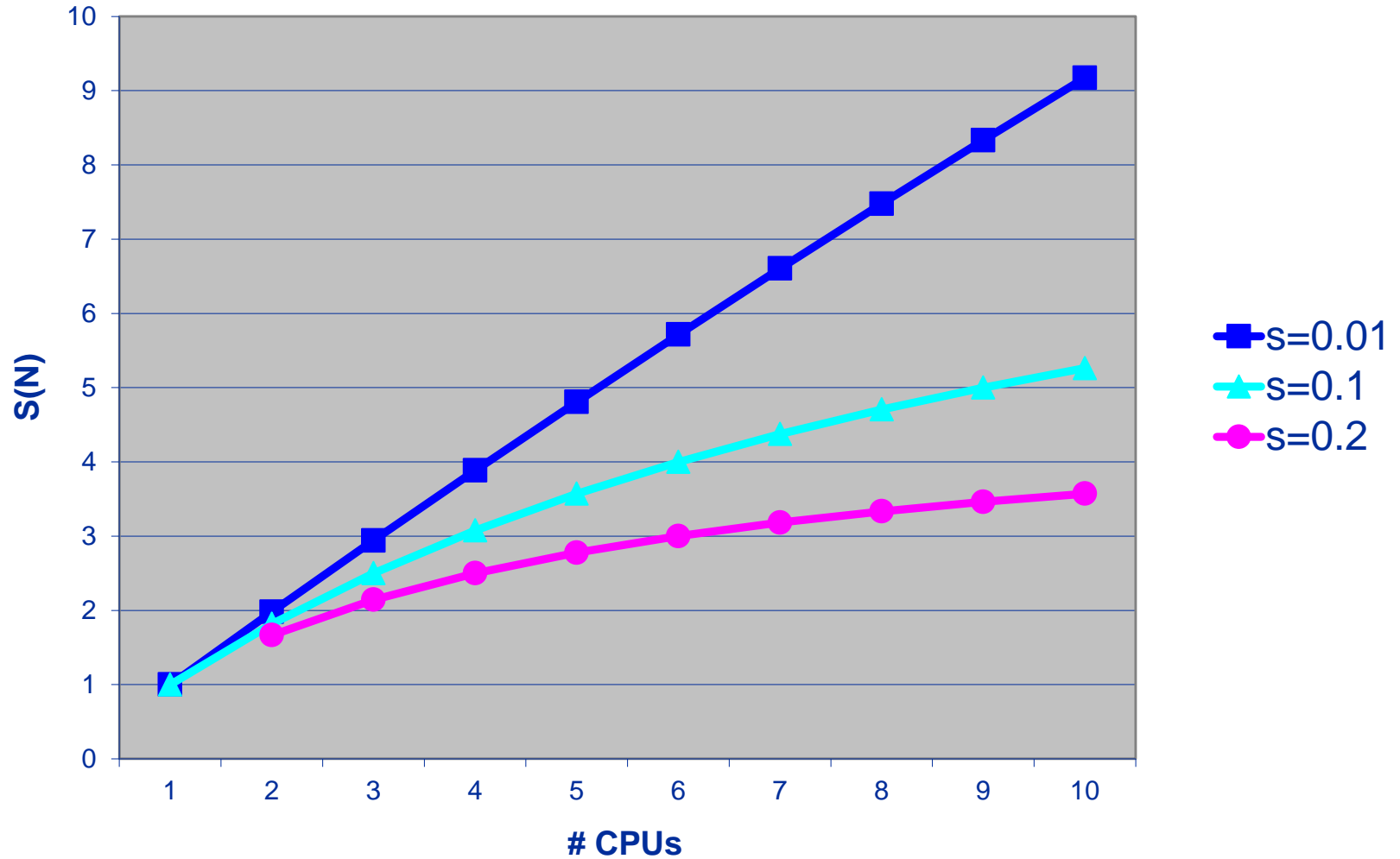
Perfectly Parallelizable



→ Parallel speedup:
Amdahl's Law

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

Limitations of parallel computing: Amdahl's Law ("strong scaling")





- Benefit of parallelization is strongly limited by serial part (s)
 - Maximum Speed-Up which can be attained: $\lim_{N \rightarrow \infty} S_P(N) = \frac{1}{s}$
- Parallel Efficiency: $\varepsilon_p = \frac{1}{s(N-1) + 1}$
 - For large number of workers $\lim_{N \rightarrow \infty} \varepsilon_p(N) = 0$
- Reality: No task is perfectly parallelizable
 - Shared resources have to be used serially
 - Task interdependencies must be accounted for
 - **Communication overhead (but that can be modeled separately)**



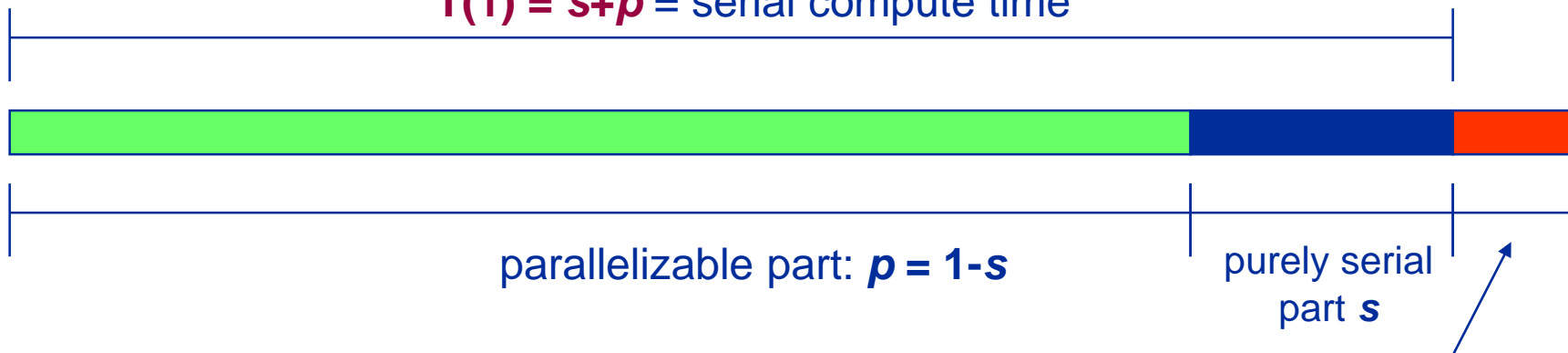
- Assume that $c(N)$ is the communication time when using N processors with $c(1) = 0$
→ $T(N) = s + \frac{p}{N} + c(N)$
- Communication time may depend on many factors:
 - Network topology
 - Communication pattern
 - Message sizes
 - ...
- Typical scaling of communication times:
 - Global communication, e.g. barrier: $c(N) = k \log N$
 - Every process sending message over bus based network $c(N) = k N$
(see next slide)

Limitations of parallel computing:

Amdahl with (simple) communication Model: Extended Amdahl



$T(1) = s+p$ = serial compute time

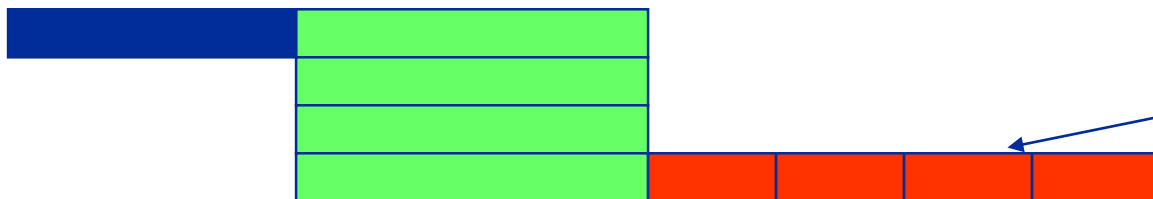


parallelizable part: $p = 1-s$

purely serial part s

Communication model: Constant fraction k for each “communication” between each two workers

parallel: $T(4) = s+p/4+4k$



Specific choice of communication model (ring shift – bus network)

Extended Amdahl model
(for specific communication model):
($k=0 \rightarrow$ Amdahl's Law)

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



- **Large N limits**

- Amdahl's Law predicts ($k=0$)

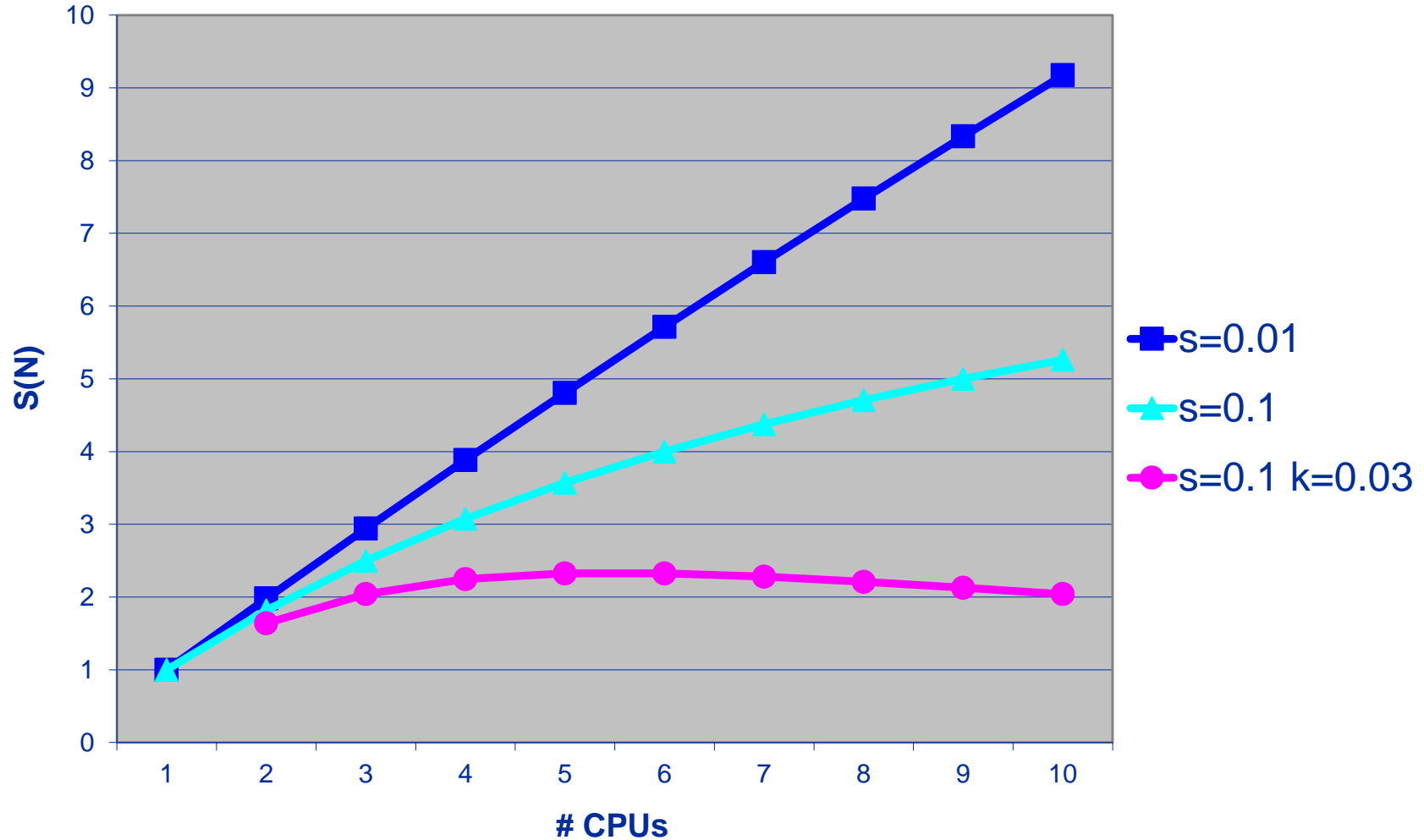
$$\lim_{N \rightarrow \infty} S_p^0(N) = \frac{1}{s}$$

(independent of N)

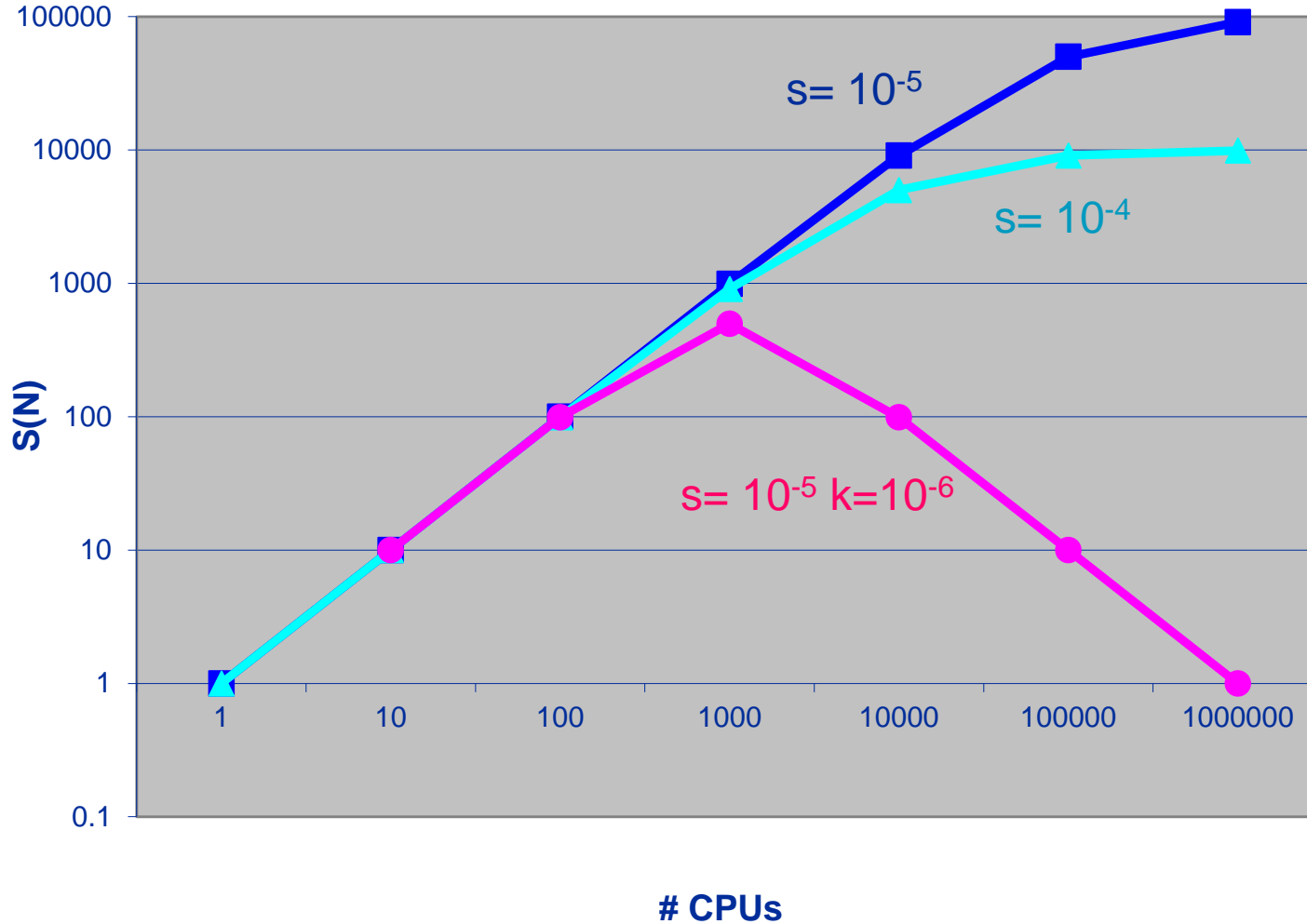
- At $k \neq 0$, our simplified model of communication overhead yields a behaviour of

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

Limitations of parallel computing: Amdahl's Law



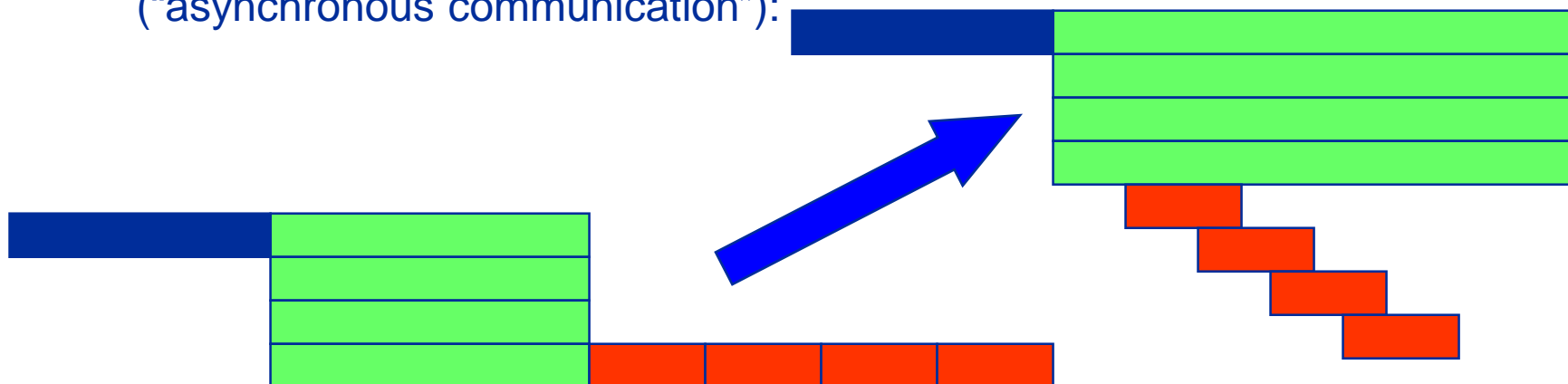
Limitations of parallel computing: Amdahl's Law at scale





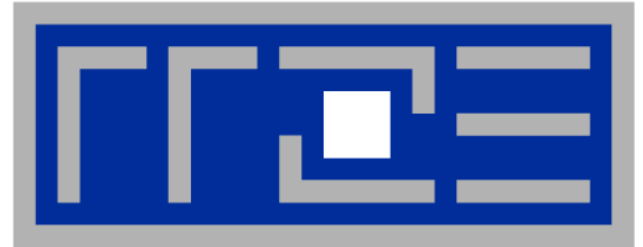
- **Communication is not necessarily purely serial**

- **Non-blocking** networks can transfer many messages concurrently – factor Nk in denominator becomes k , which can be added to s (technical measure)
- Sometimes, **communication can be overlapped** with useful work (“asynchronous communication”):



- **But never forget**

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



Basic limitations of parallel computing

Amdahl's law (“strong scaling”)

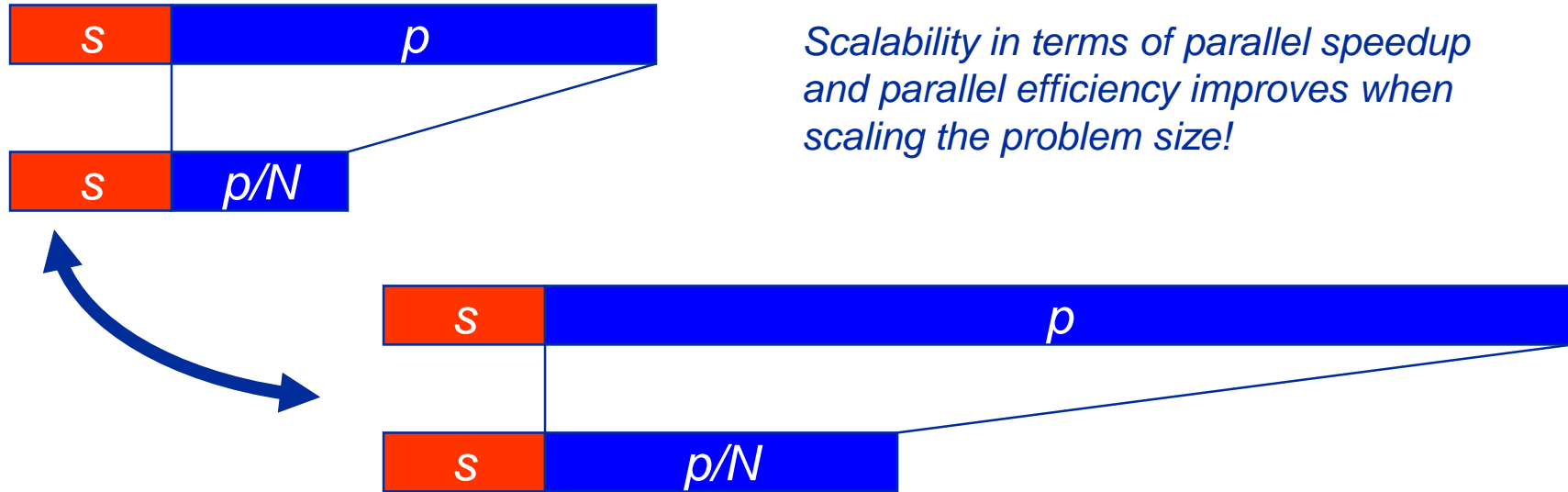
Gustafson's law (“weak scaling”)

Applying Amdahl's law

Limitations beyond Amdahl/Gustafson



- Increasing problem size often mainly enlarges „parallel“ workload p
 - Then Speed-up increases with problem size



- For some application fields: Solve problems as big as possible
 - Increase (parallel) workload with available workers / processors
 - This is called „weak scaling“

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

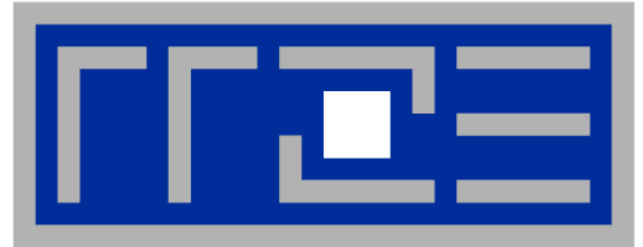


- Assume simple and optimistic scenario: Parallel Workload increases linearly with N , i.e. $p \rightarrow N p$
 - $\rightarrow T(N) = s + \frac{N p}{N} = s + p$
 - \rightarrow Runtime stays constant if workload is increased linearly with N
 - \rightarrow Performance increases linearly with N
- How long does it take to solve the workload of N processors on 1 processor
 - $\rightarrow T_N(1) = s + N p$

$$\rightarrow S(N) = \frac{T_N(1)}{T(N)} = \frac{s + N p}{s + p} = \frac{s + N p}{T_S(1)} = s + (1 - s)N$$

Speed-Up increases linearly with N

Gustafson's Law
("weak scaling" – performance scaling)



Basic limitations of parallel computing

Amdahl's law (“strong scaling”)

Gustafson's law (“weak scaling”)

Applying Amdahl's law

Limitations beyond Amdahl/Gustafson



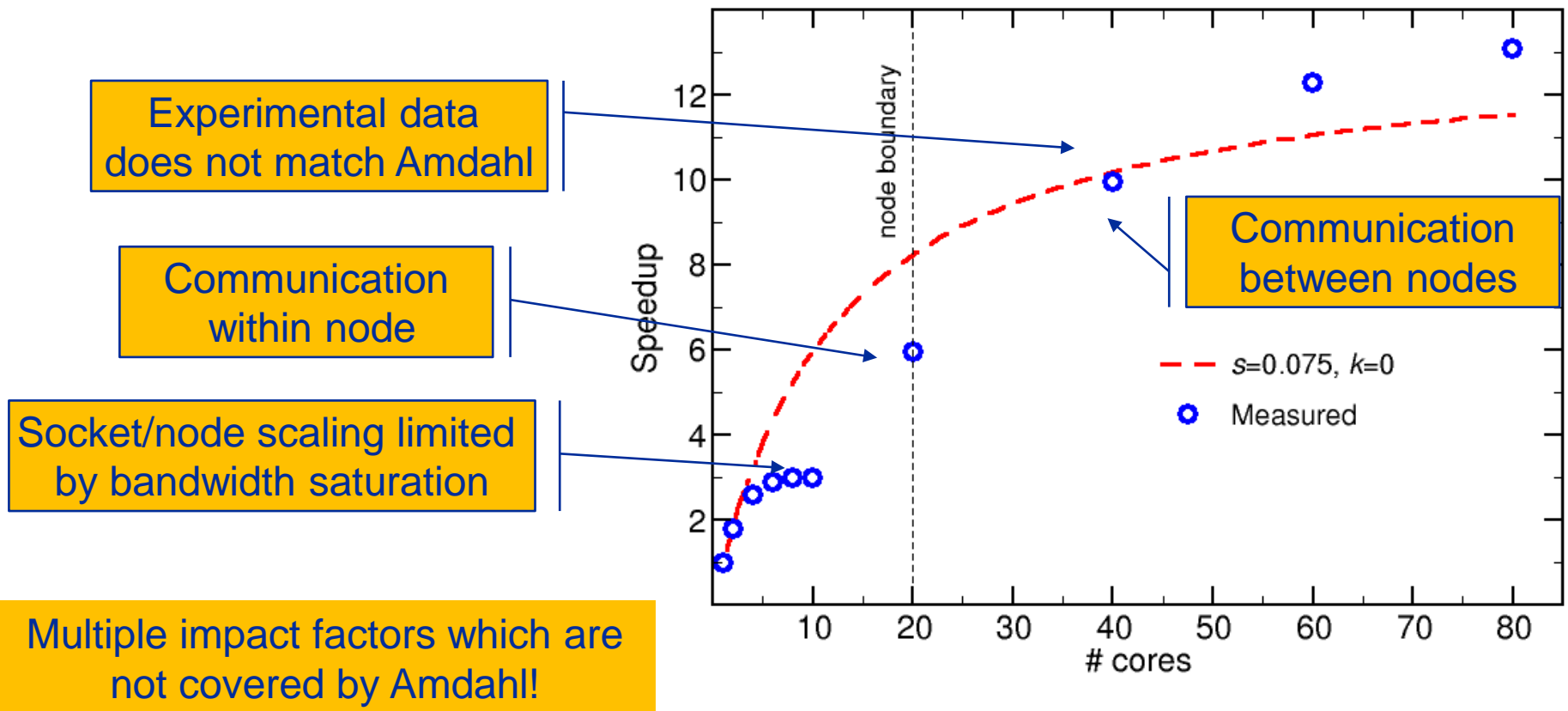
Always remember **model assumptions**:

- **Workload consists of**
 - purely serial (s) and
 - perfectly parallelizable ($p \rightarrow \frac{p}{N}$) parts
- **Scalability is limited by**
 - serial fraction or
 - communication overhead (extended Amdahl).
- **Impact of shared/saturating hardware resources is not modeled**
- **How to determine model parameters (s, p)?**
 - First principles: Complete knowledge of application and hardware parameters required – too complex for most applications/kernels
 - **Fit model parameters to speedup measurements**

Limitations of parallel computing: Applying Amdahl: Serial & Parallel fraction



- **Naïve approach:** Measure performance as a function of cores and fit (extended) Amdahl's law (cf. slide 8/11)
- Hypothetical study on Emmy (i.e. 2-sockets 10 core each per node) – extended Amdahl



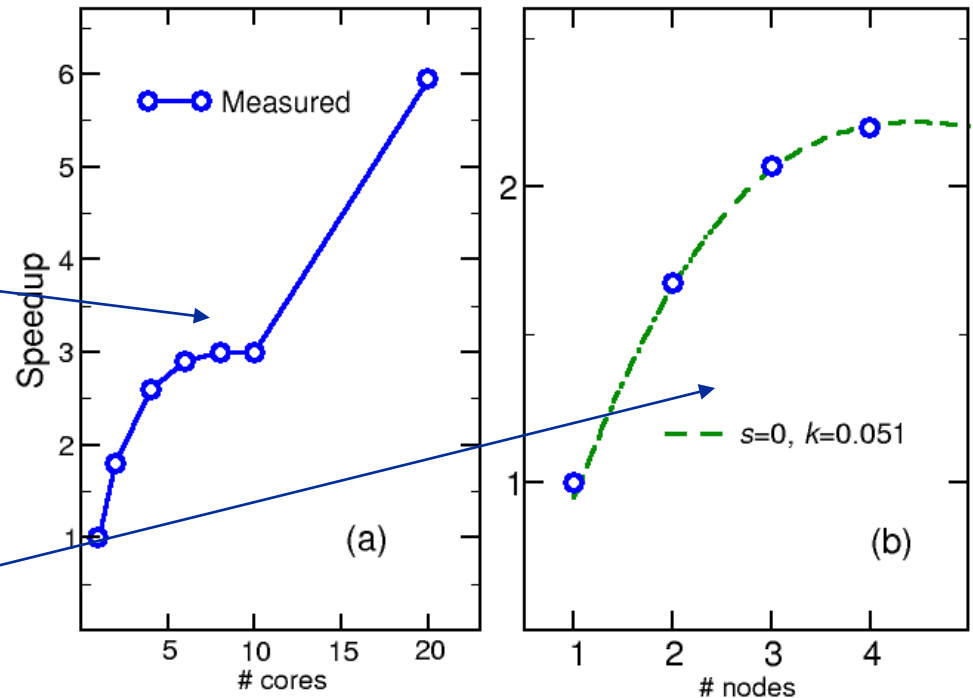
Limitations of parallel computing: Applying Amdahl: Serial & Parallel fraction



- **Better approach:** Separation of concerns! Use well-defined basic building blocks as “workers”, which
 - are perfectly scalable (no shared resource in between)
 - restrict measured effects to model assumptions, e.g. use full nodes only (one communication path, serial fraction still visible)

Socket/node saturation to be modeled separately by e.g. ECM model

Amdahl's modelling serial fraction and one communication speed





- **Amdahl's law can also be interpreted as follows**

- A fraction p of a given code/workload can be “accelerated” by a factor N through some “acceleration technique”
- The remainder part s cannot be accelerated, i.e. $s + p = 1$
- “Normalized” runtime of baseline code $T_{base} = 1$ (slide 8: $T(1)$)
- “Normalized” runtime of accelerated code $T_{acc}(N, s) = s + p/N$ (slide 8: $T(N)$)

- **The speed-up of the acceleration technique is**

$$S_p(N) = \frac{T_{base}}{T_{acc}(N, s)} = \frac{1}{s + \frac{1-s}{N}}$$

- **Potential “Acceleration factors”**

- Parallel processing with N processes assuming perfect speed-up on fraction p
- Using an accelerator (e.g. GPGPU) which executes the fraction p of a code N times faster
- Implementing a code transformation which speeds up a fraction p of a code by N times

Limitations on parallel computing: Applying Amdahl: A more general view



Application: GPGPU accelerated code

- Execution time of original code on host:

$$T_{base}$$

- "Accelerated execution" (offload)

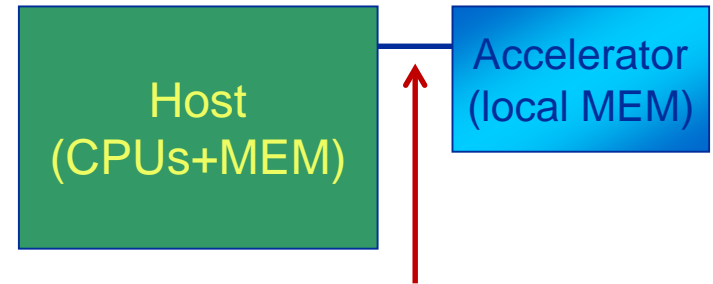
- A fraction p of the original code can be executed on GPGPU N times faster than CPU
- The remaining part s is executed on host

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

- Consider data transfer between host and accelerator: Extended Amdahl's law

$$\rightarrow S_p(N, k) = \frac{1}{s + \frac{1-s}{N} + k}$$

where $k = (\text{total data transfer time})/T_{base}$



Data exchange (e.g. via PCIe)

Example:

- $T_{base} = 150$ s
- 75% of that is put on GPGPU $\rightarrow p = 0.75$
- GPGPU runs 15x faster than host $\rightarrow N = 15$
- $\rightarrow S_{0.75}(15) = 3,33$
- If total data transfer is 15 s
- $\rightarrow k = \frac{15}{150} = 0.1$
- $\rightarrow S_{0.75}(15, 0.1) = 2,5$



Basic limitations of parallel computing

Amdahl's law (“strong scaling”)

Gustafson's law (“weak scaling”)

Applying Amdahl's law

Limitations beyond Amdahl/Gustafson

Limitations of parallel computing – beyond Amdahl/G.

Shared/saturated hardware resources



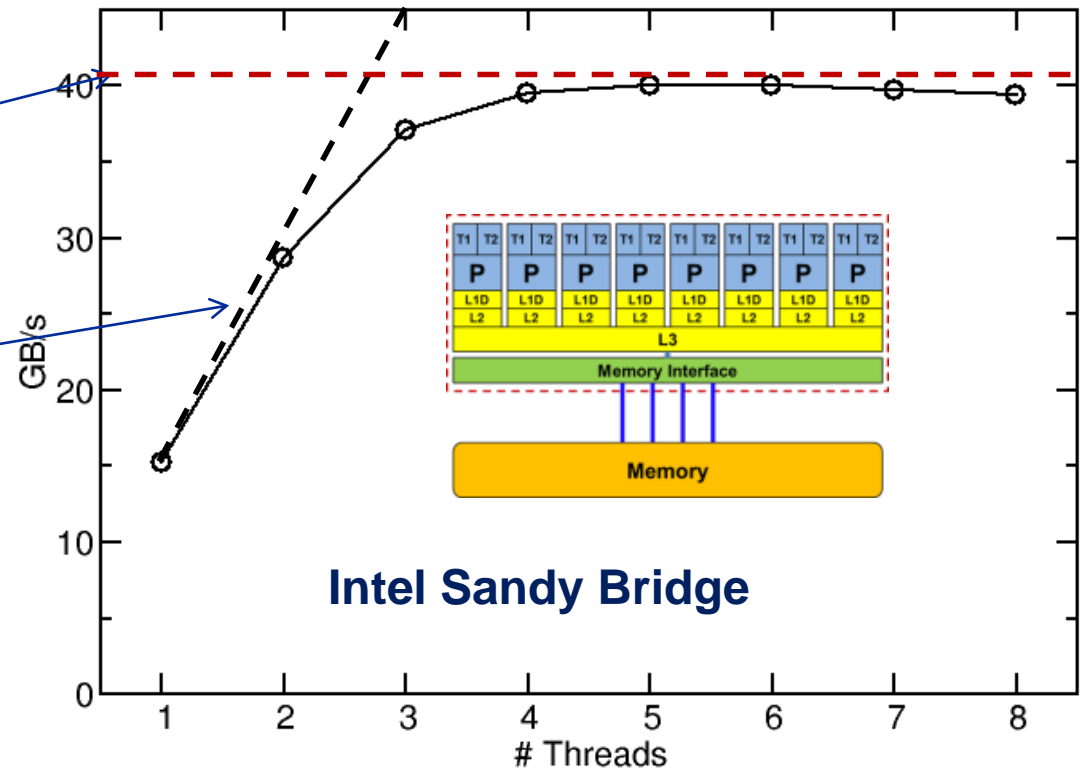
- Saturations of shared hardware resources set limits to scalability not covered by Amdahl's / Gustafson's law

```
do i= 1 , 10 000 000
  a(i) = b(i) + s * c(i)
enddo
```

Assume perfect parallelization: $s=0$

- Technical limit imposed by hardware (40 GB/s)

- Parallel performance assuming perfect scalability (p/M)
 → Parallel scalability limited by saturated hardware resource



- Other potential HW bottlenecks: QPI, PCIe, networks (see next lecture)

Limitations of parallel computing – beyond Amdahl/G.

Synchronization points and **load imbalance**



- **Load imbalance** between “workers”
→ p/N assumption no longer valid (in general)

- Hard to model in a general way, but there are important special cases:
 - A few “**lagers**” waste lots of resources
 - A single (consistent) lagger could be modeled by increased serial fraction

- A few “**speeders**” may be harmless

→ turning some “lagers” into “speeders” may boost performance a lot!

