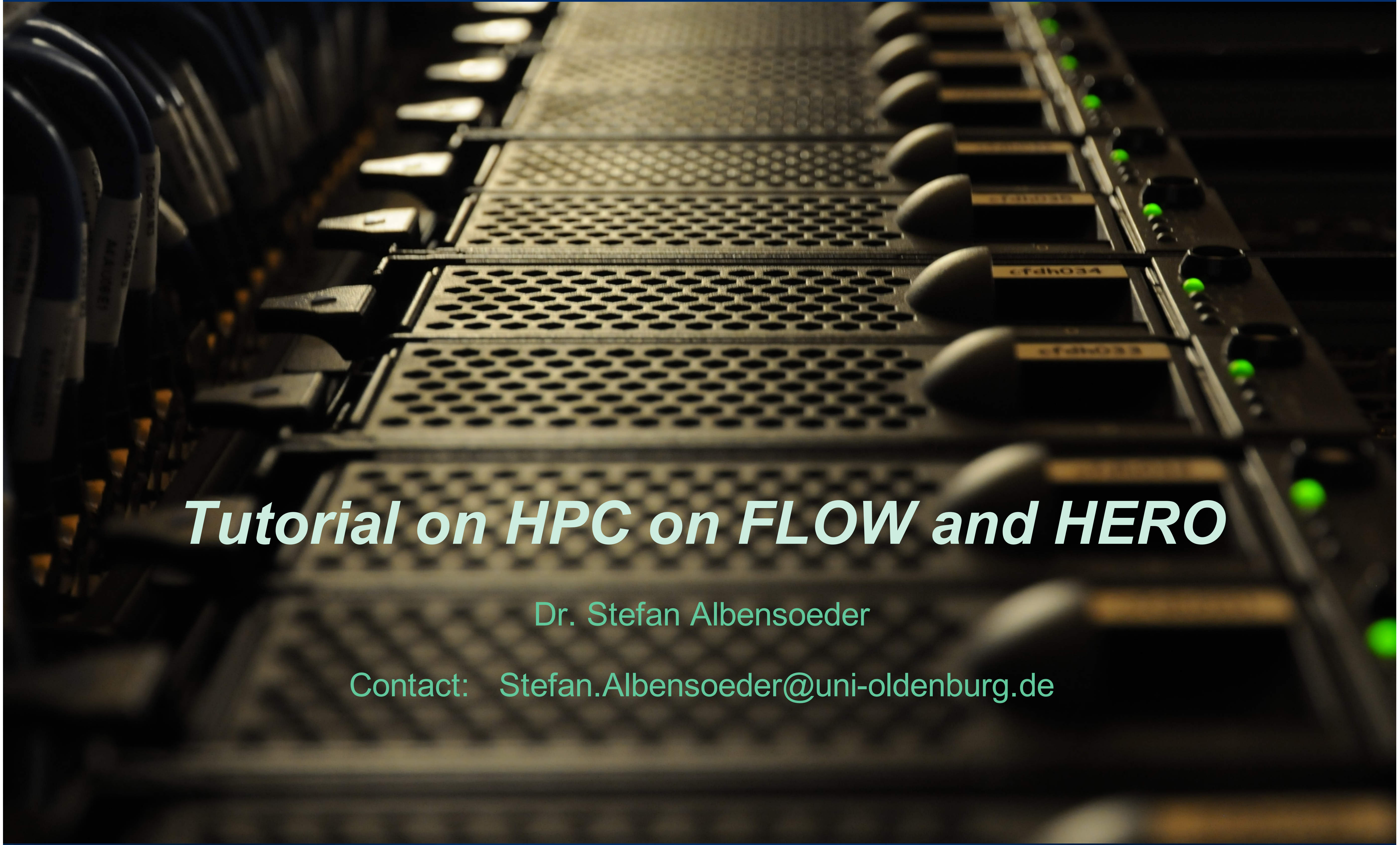
ForWind
Center for Wind Energy Research



Tutorial on HPC on FLOW and HERO

Dr. Stefan Albensoeder

Contact: Stefan.Albensoeder@uni-oldenburg.de

Leibniz Universität Hannover

Carl von Ossietzky Universität Oldenburg

Universität Bremen

Outline Day 1

- Introduction to HPC
  - Motivation
  - Architectures
  - HPC at the University of Oldenburg
  - Parallel models
- HPC cluster usage
  - Login
  - Modification of user environments via "module,,
  - Available development tools
  - Brief hints on performance

Outline Day 2

- Introduction to the usage of the job scheduler SGE
  - Introduction to SGE
  - General Job submission (specifying job requirements)
  - Single Slot jobs (how to compile submit and monitor status)
  - Parallel Jobs (openMPI, impi, smp)
  - Monitoring and Controlling jobs (qstat, qrsh, qacct)
- Debugging
  - Compiling programs for debugging
  - Tracking memory issues
  - Profiling

Outline Day 2

- Misc.
  - Logging in from outside the university
  - Mounting the HPC home directory
  - Parallel environment memory issue
  - Importance of allocating proper resources

Outline Day 3

- Exercises (Computer-Lab)
  - Try out the examples given in part II
  - Estimate pi using Monte Carlo simulation

Introduction to HPC

MOTIVATION

Motivation

- HPC - *High-Performance-Computing*
  - Large amount of computational resources which are somehow connected
    - Close to each other
      → Computing-Cluster (e.g. FLOW, HERO)
    - Distributed resources (e.g. Laptops, PC's)
      → Grid-Computing
- Why HPC?
  - Enable to solve large problems by parallelization
    (e.g. large linear equation systems, simulation)
    - Processors work together
    - Decrease memory usage per core
    - Reduce simulation wall clock times
  - Computing of small problems in a huge parameter space
    - Processes are independent

Introduction to HPC

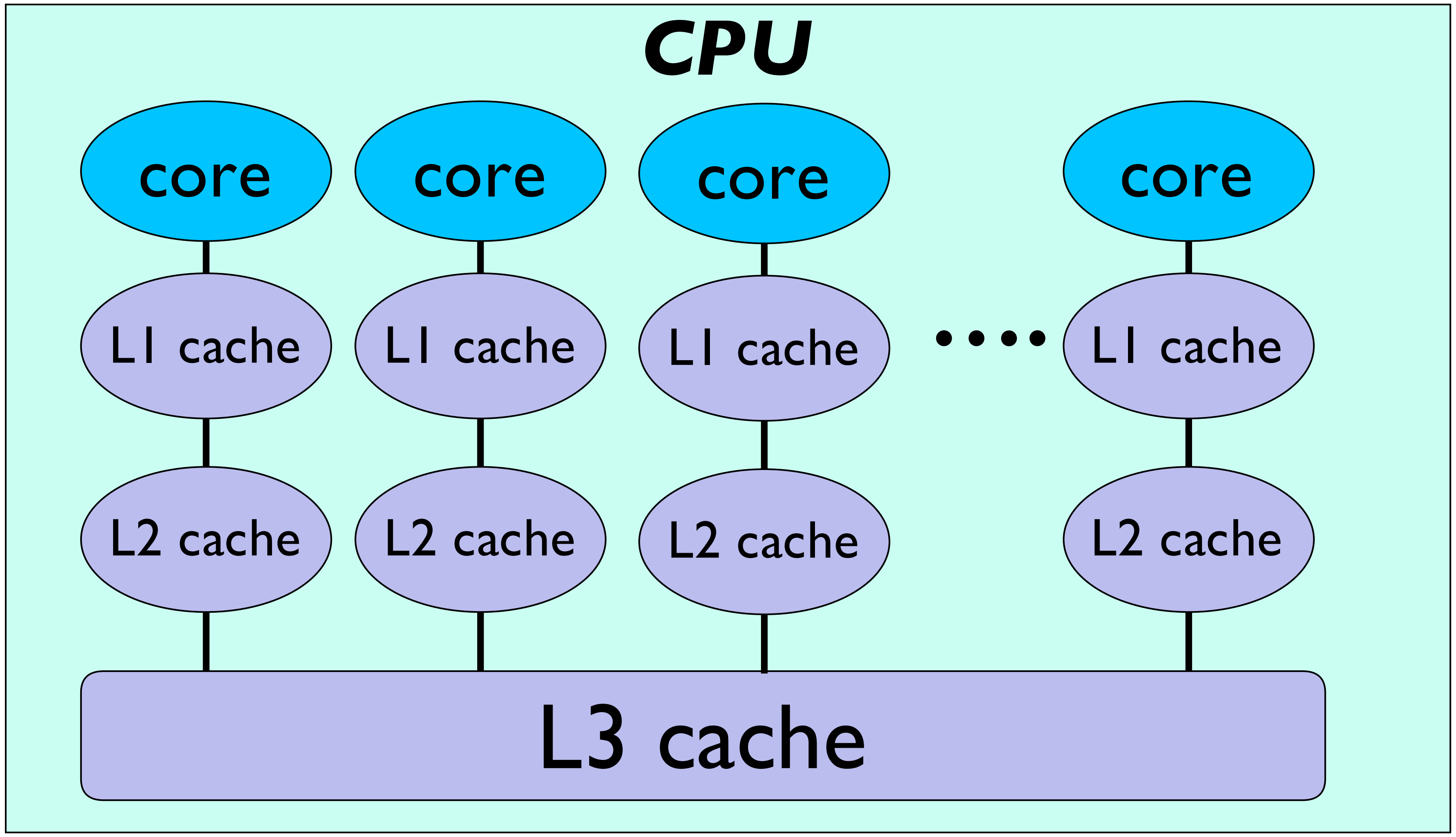
ARCHITECTURES

Architectures

- Why is knowledge about hardware architecture important?
  - To write efficient code:
    - Algorithm should fit to the architecture
      → Increase performance
  - To know the limits

Central Processing Unit - CPU

- CPU contains several cores
- Cores connected to caches for fast memory access, low latency
  → O(10) faster than direct memory access
- Cache coherence



Westmere X5650

6 cores, 2,66GHz

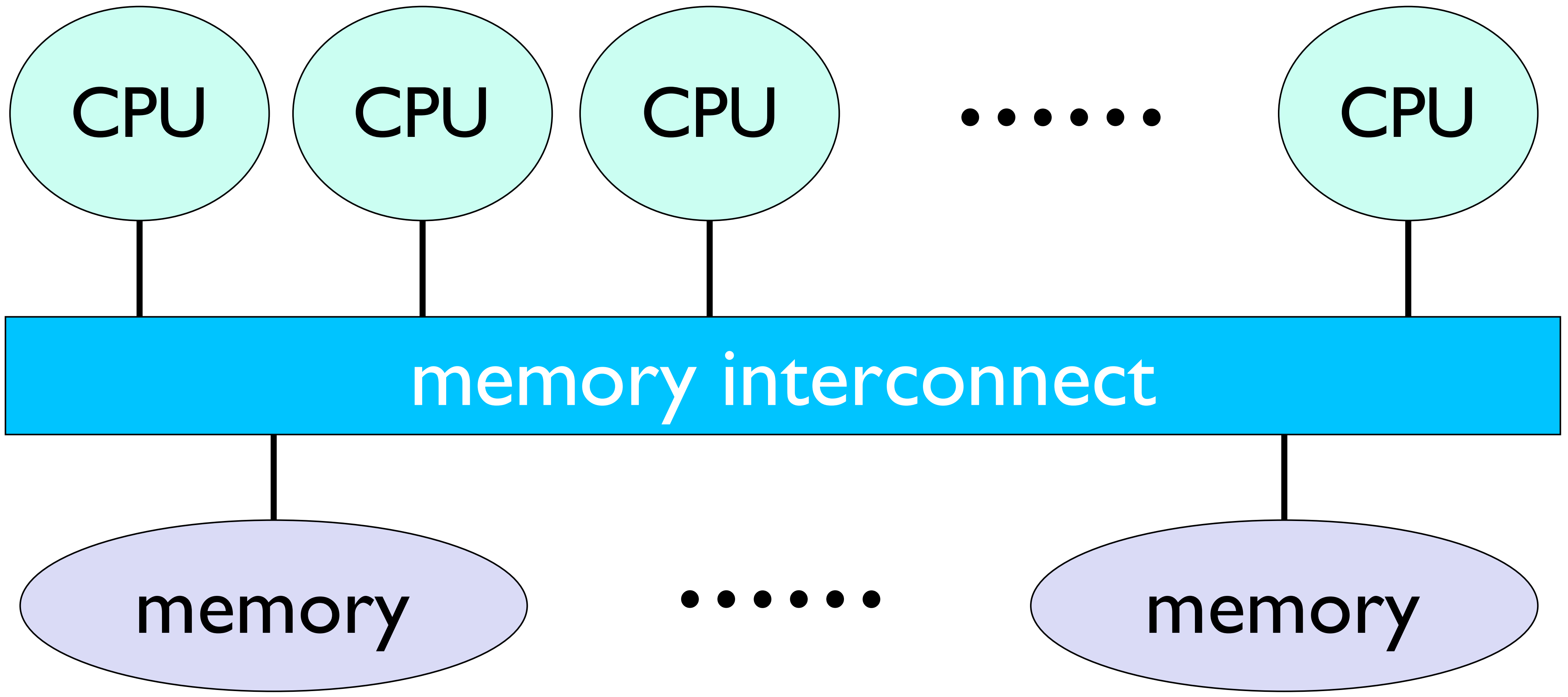
64Kb

256Kb

12Mb

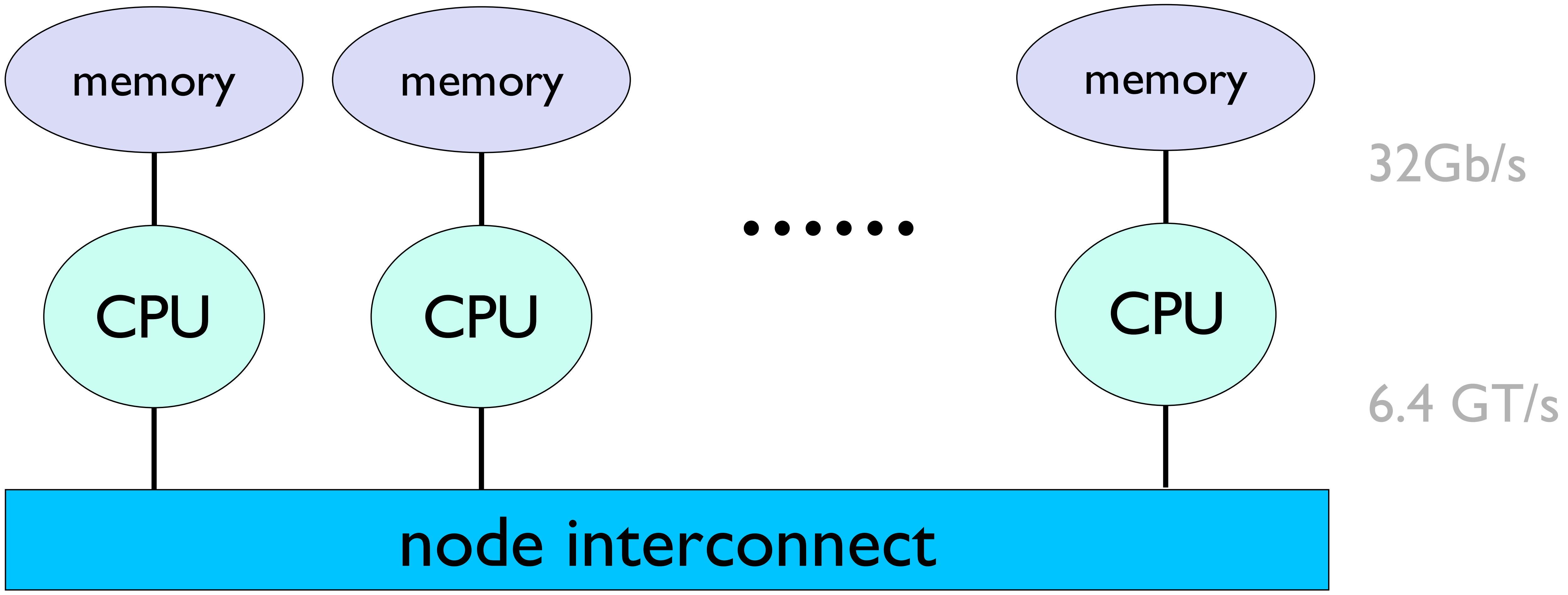
Shared memory system

- Symmetric multi-processing (smp)
- Uniform memory access (uma)
  - Same access time from every CPU



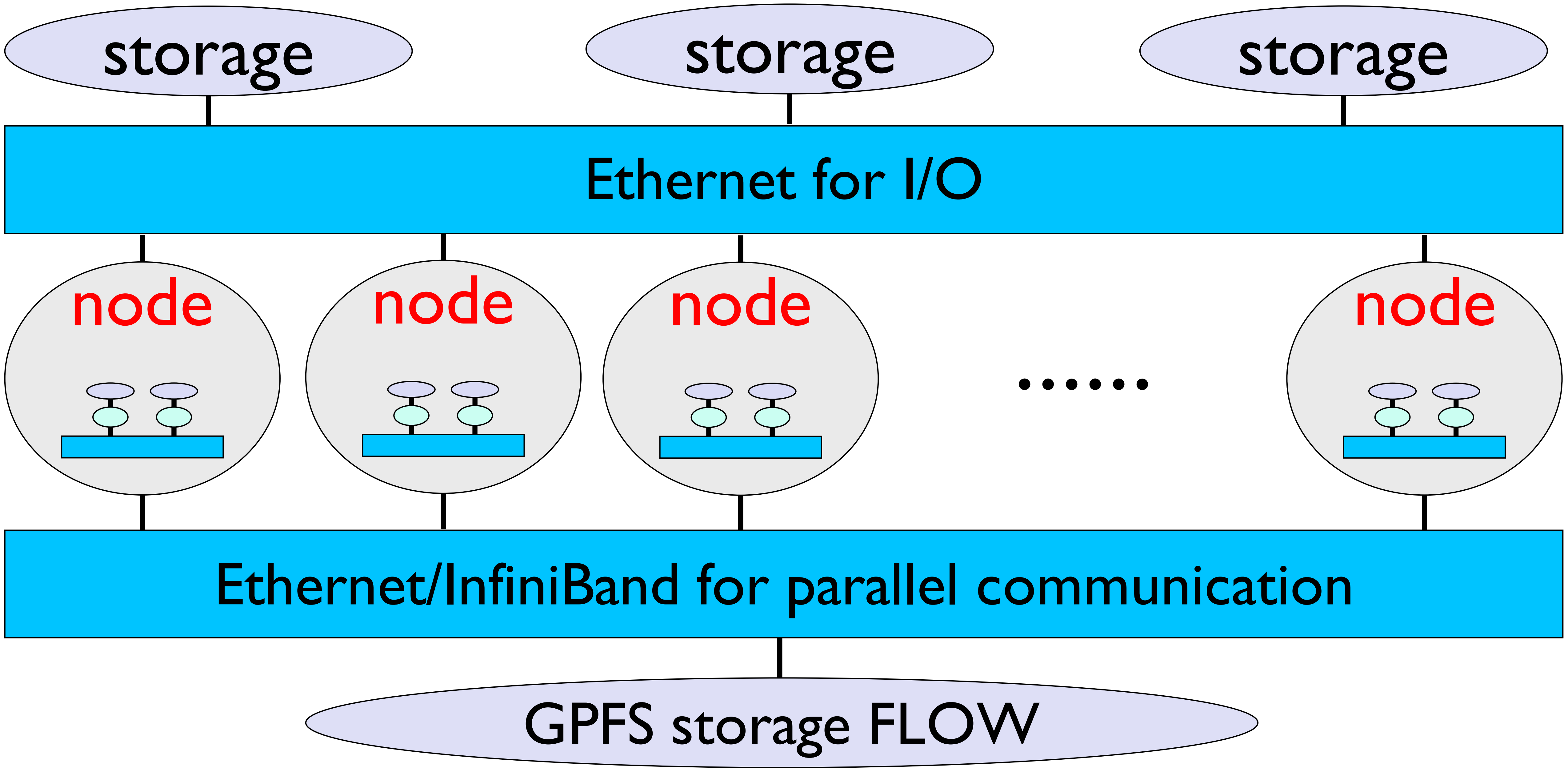
Node architecture

- Non-uniform memory access (NUMA)
  - Fast access to own memory
  - Slow access to other memory
- Cache coherence → ccNUMA



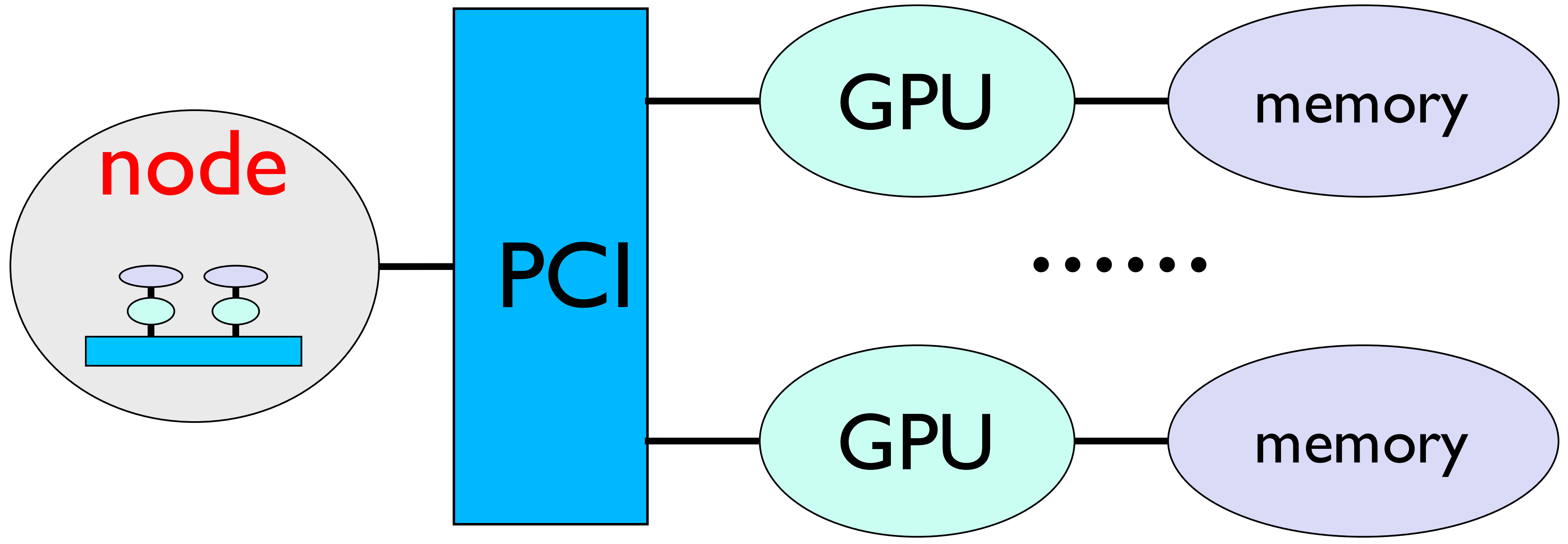
HPC-Cluster FLOW/HERO

- O(100) NUMA nodes connected by fast interconnect



Accelerators (many cores)

- Todays typical accelerators
  - General-Purpose computing on Graphics Processing Units (GPGPU), e.g. Graphic card chips with O(1000) cores
  - Intel Phi (~60 Pentium cores)



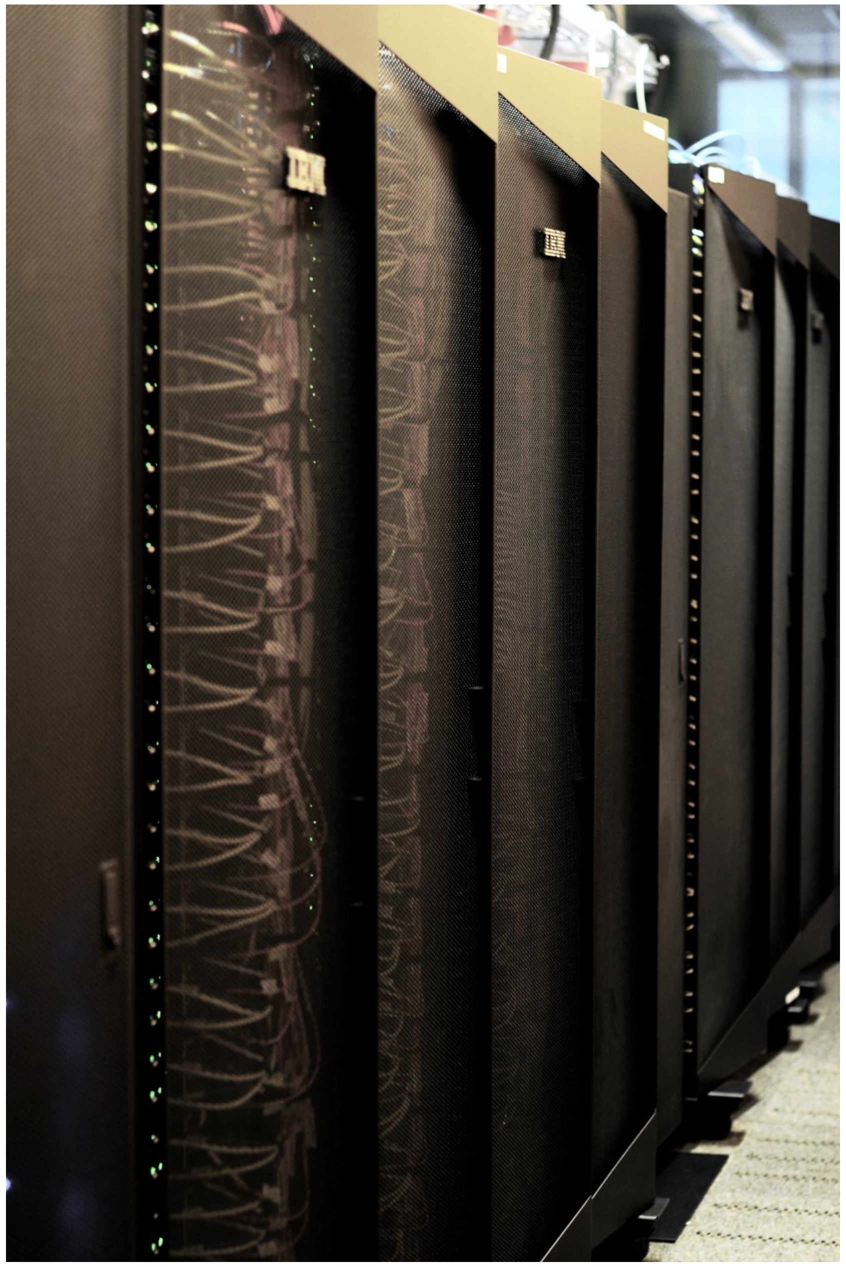
- Other
  - Field-Programmable Gate Arrays (FPGA)

Introduction to HPC

HPC AT THE UNIVERSITY OF OLDENBURG

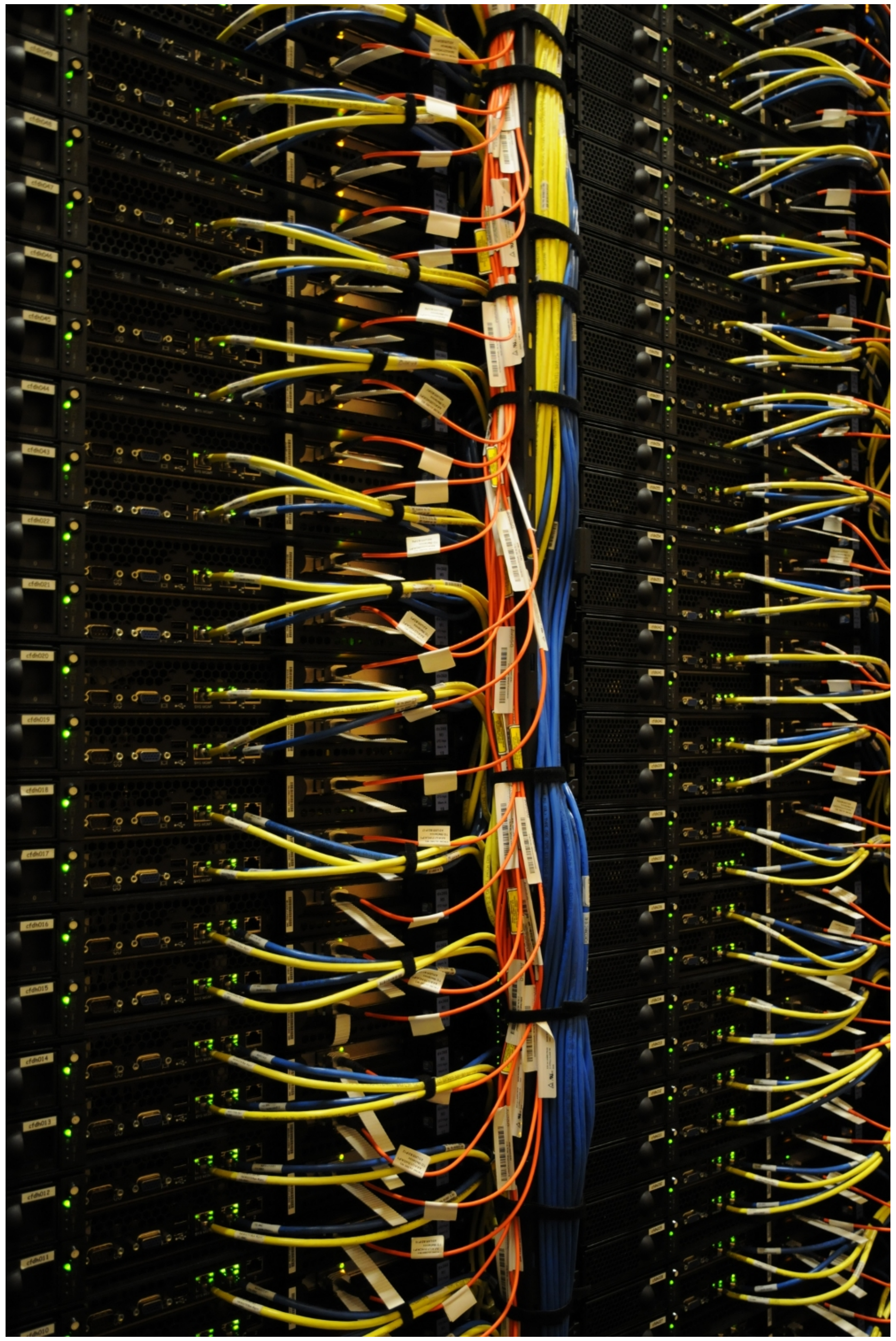
FLOW –
Facility for Large-Scale Computations in Wind Energy Research

- 122 "low-memory" compute nodes: 2x6 cores per node, 24Gb, diskless (host names cfdl001..cfdl122)
- 64 “high-memory" compute nodes: 2x6 cores per node, 48Gb, diskless (host names cfdh001..cfdh064)
- 7 compute nodes: 2x4 cores per node, 32Gb, diskless (host names cfdx001..cfdx007)
- QDR InfiniBand interconnect (fully non-blocking)
- Gigabit Ethernet for File-I/O etc.
- High-performance IBM GPFS storage system, 130TB connected by InfiniBand
- 160 TB NAS storage shared with HERO
- Theoretical peak performance: 24 TFlop/s (Flop/s – Floating Point Operations per second)



HERO - High-End Computing Resource Oldenburg

- 130 "standard" nodes: 2x6 cores, 24 GB, 1 TB disk
  (host names mpcs001..mpcs130)
- 20 "big" nodes: 2x6 cores, 48 GB, RAID 8 x 300 GB SAS
  (host names mpcb001..mpcb020)
- Gigabit Ethernet II for communication of parallel jobs
- Second, independent Gigabit Ethernet for File-I/O
- SGI Altix UV 100 shared-memory system: 10x6 cores,
  640 GB, RAID 20 x 600 GB SAS (host uv100)
- 160 TB NAS storage shared with FLOW
- Theoretical peak performance: 19.2 TFlop/s
  (Flop/s – Floating Point Operations per second)

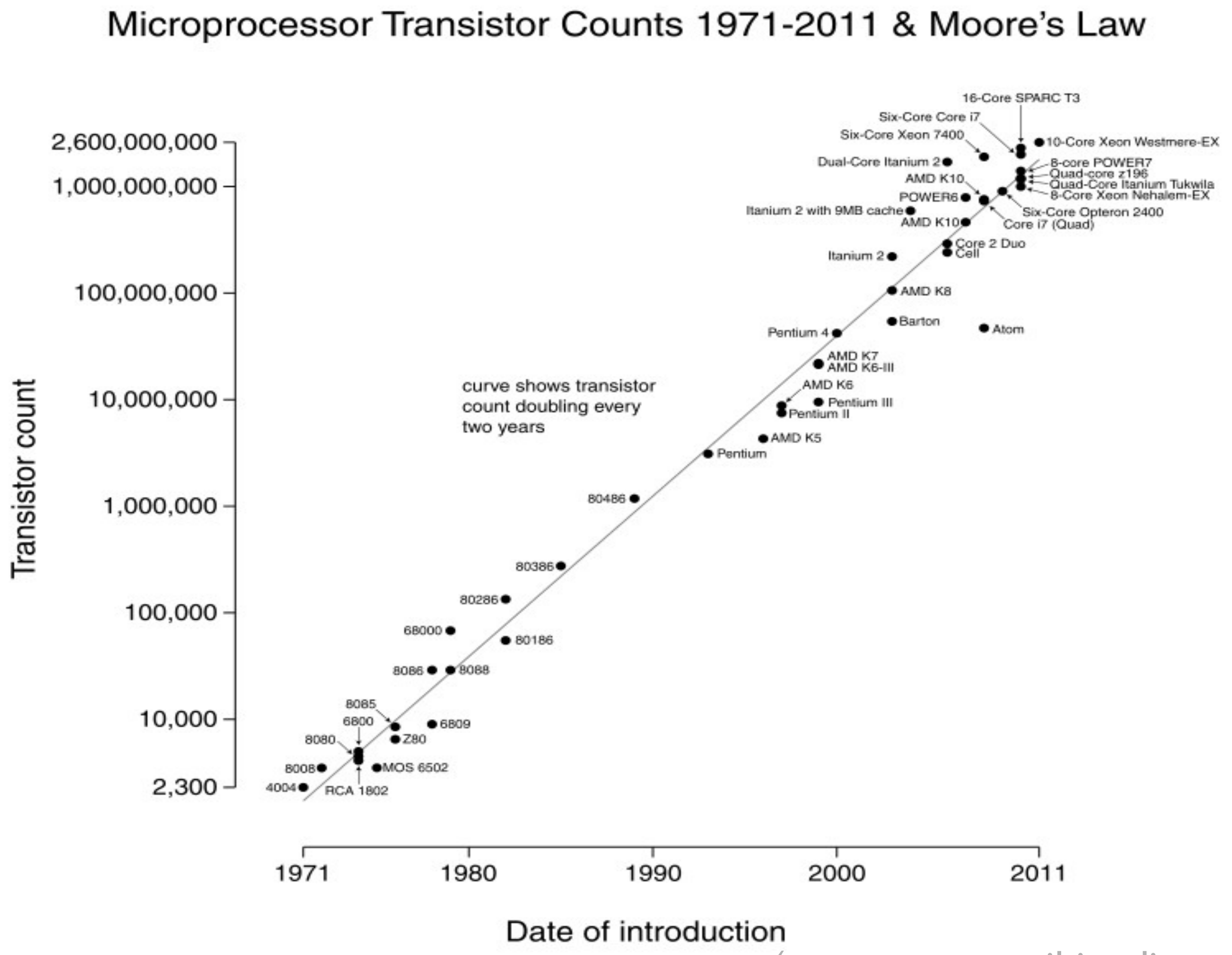


Introduction to HPC

PARALLEL MODELS

Why parallelization?

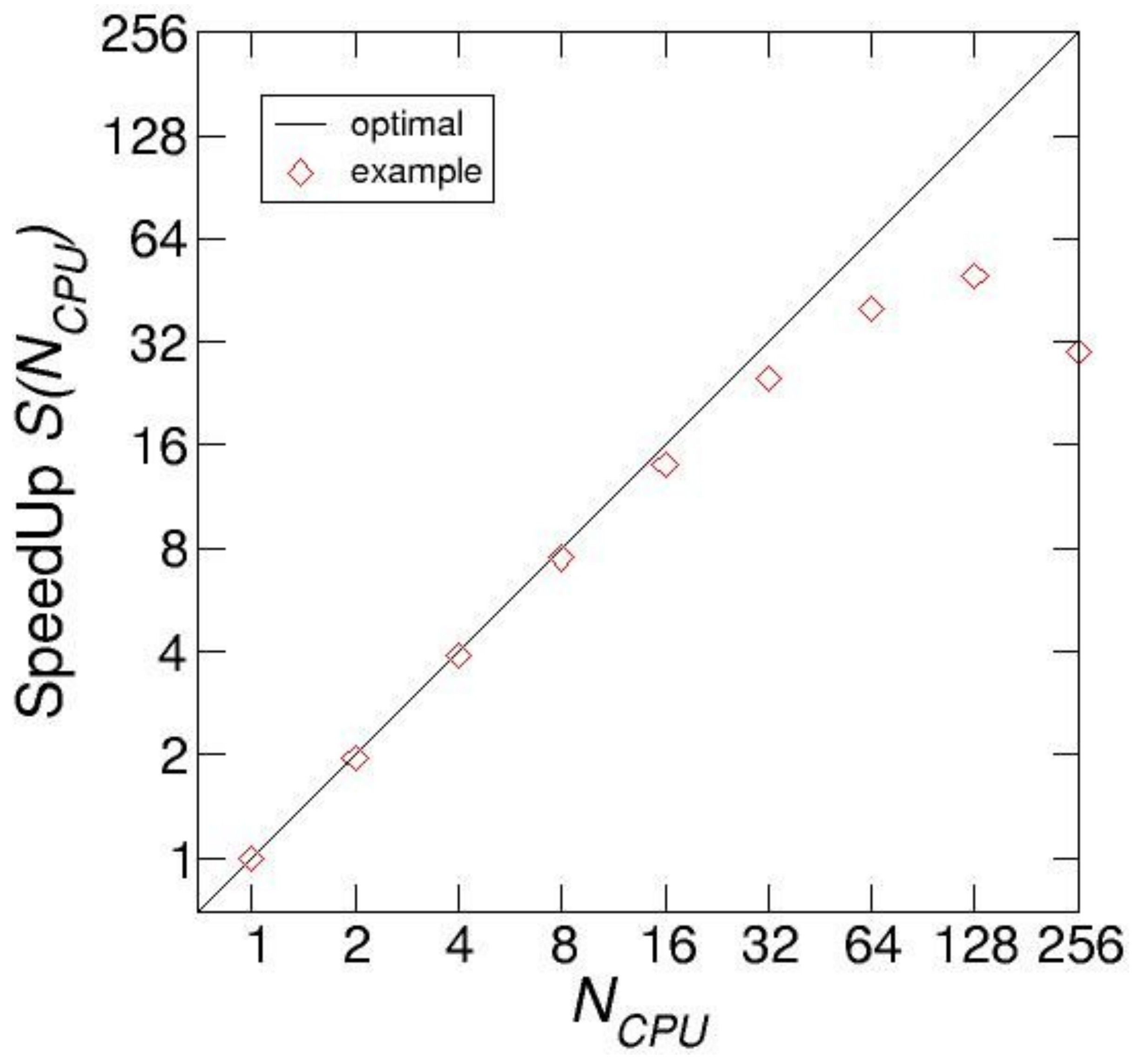
- Moore‘s law
  - The number of transistors on a chip will double every ~18 month
  - Frequency of todays CPUs
    → constant or decreasing
    (→ power consumption)
  - Increase only
    → more cores per CPU
- Problem too complex
  → time consuming
- Problem size
  → high memory needs



(source www.wikipedia.org)

Limit of parallelization - Scaling

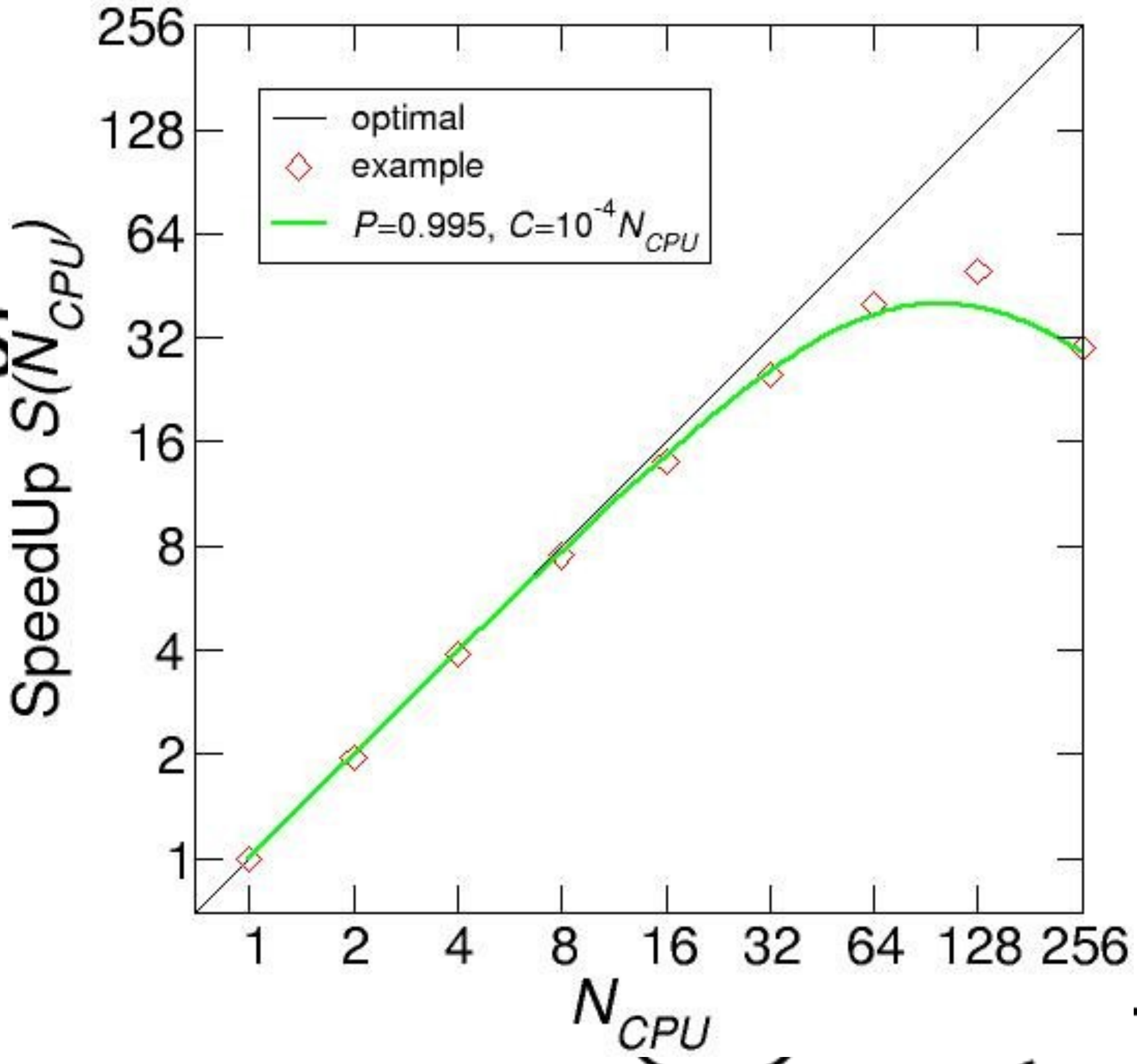
- Speedup of the program when increasing number of processes / used cores
- Dependencies
  - Fraction of parallelization code P
  - Amount of communication / synchronization $C \sim O(N)$
  - Size / complexity of problem (e.g. number of unknown of an equation system)
  - Load balance (distribution of load on processors)
  - Hardware (latency/speed of network/memory access, cache coherence,…)



ForWind
Center for Wind Energy Research

Limit of parallelization - Scaling

- *Strong scaling*
  - Problem size cons
  - Speedup *S(N)* in
  - Amdahl's law:



$(1 -$

Weak scaling

Leibniz Universität Hannover
Carl von Ossietzky universität Oldenburg
Universität Bremen

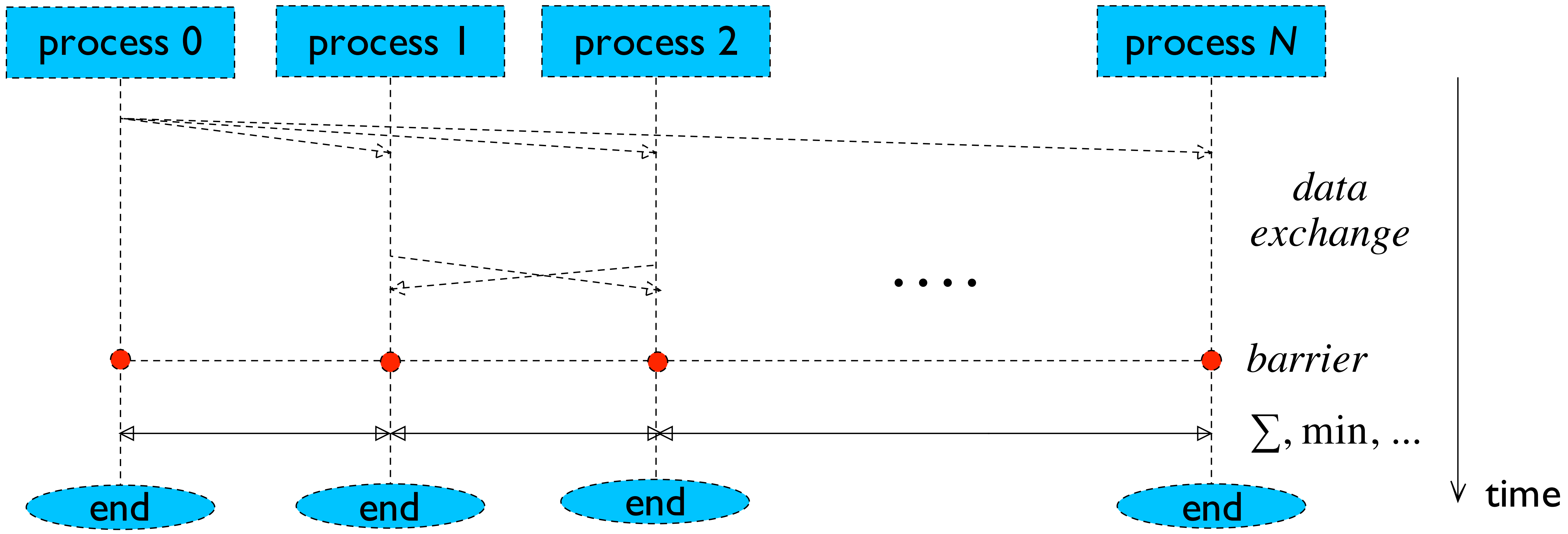
Parallel programming models - Overview

- Distributed memory
  - MPI (Message Passing Interface)
  - PVM (Parallel Virtual Machine)
- Distributed shared memory
  - PGAS (Partitioned Global Address Space)
- Shared memory
  - PThreads (POSIX Threads)
  - OpenMP (Open Multi-Processing)
- Accelerator device
  - Nvidia’s CUDA (Compute Unified Device Architecture)
  - OpenCL (Open Computing Language)

→ *most common: MPI, PThreads and OpenMP*

Distributed memory model - Message Passing Interface

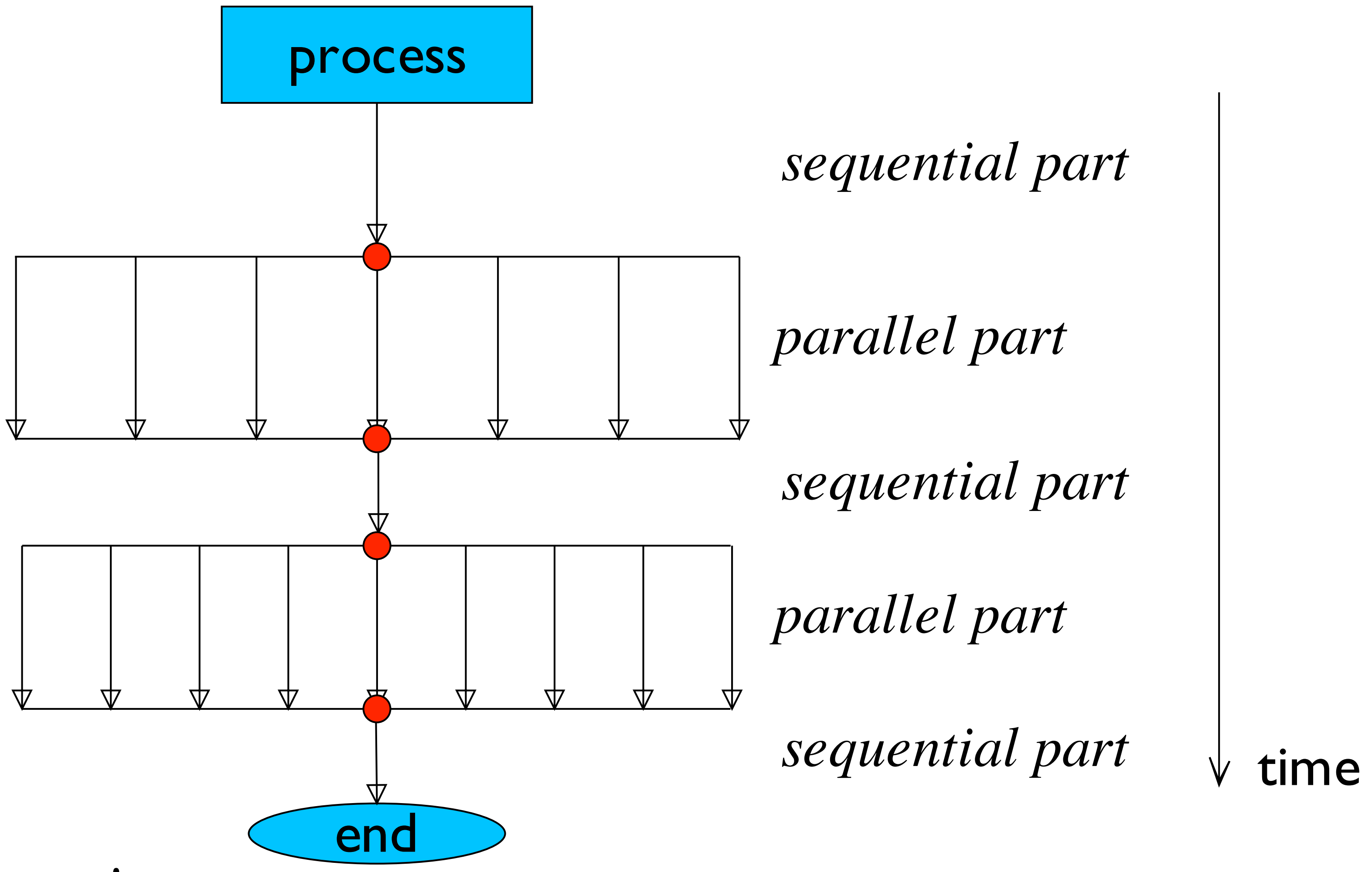
- *N* distinct processes with own memory segment
- Synchronization / data exchange / collective operations over messages



- Parallel I/O
- Usable for shared memory architectures

Shared memory models - PThreads and OpenMP

- One process with multiple threads
- Use same memory segment → variables are accessible from every thread
- OpenMP
  - Pragma compiler statements
    → Minimal change of code
- PThreads
  - Library interface
  - Mutexes to avoid data collisions
  - Code has to be changed



- Thread creation can be time consuming
- Only usable on shared memory architectures!

HPC CLUSTER USAGE

Login

- Login nodes of FLOW and HERO
  - `flow.hpc.uni-oldenburg.de` (to access `flow01` or `flow02`)
  - `hero.hpc.uni-oldenburg.de` (to access `hero01` or `hero02`)
- Login from terminal by ssh, e.g.

```
ssh -XY abcd1234@flow.hpc.uni-oldenburg.de
```

- Terminal programs
  - Windows
    - Putty (`http://www.putty.org/` , no X-Window support)
    - MobaXterm (`http://mobaxterm.mobatek.net/`, including X-Window support)
  - Linux/Mac: Terminal included
- Copy of data sftp/scp, e.g. from your host

```
scp FILENAME abcd1234@flow.hpc.uni-oldenburg.de:TARGET_DIR
```

- Note: Direct login on computational nodes is forbidden

User environment

- Most of the programs (e.g. compilers, MPI,…) not available per default
- Use command `module` to load the needed environment
  - Show available modules
    ```
    module av [NAME]
    ```
    (e.g. `module av intel` to see all Intel products)
  - Module name convention: lower case letters
  - More information about a module
    ```
    module help MODULE_NAME
    ```
  - Load module
    ```
    module load MODULE_NAME
    ```
    (e.g. `module load intel/ics/2013.5.192/64` to load Intel Cluster Studio 2013.5)
  - Unload module
    ```
    module unload MODULE_NAME
    ```

Available development tools

Compiler (C, C++,Fortran)
- Intel Cluster Studio (2013.5.192)
- GNU compiler (4.7.1)
- PGI Accelerator Suite (13.7)
- Open64

- MPI implementation
  - OpenMPI
  - MVAPICH
  - MPICH
  - Intel MPI (4.1.1.036)

- Libraries
  - BLAS/LAPACK/MKL
  - LEDA
  - FFTW
  - NetCDF, HDF5
  - …
- Other languages
  - Python
  - R
  - Matlab, Octave

and many more….

(grey numbers: preferable releases)

HPC cluster usage

BRIEF HINTS ON PERFORMANCE

Why spend time in performance optimization?

- Performance
  → Fast solving of a problem
  → Optimal usage of computational resources consumption
- The optimization of code can be time consuming
  - For best performance may need deep knowledge of hardware/programming
- BUT already simple things can speedup your code significantly!
  (e.g. the choice of the compiler can give a speedup of 1.5-2)
- Benefits
  - less use of computational resources
    - more resources for all users left
    - less energy consumption
  - more results in the same time
  - enabling of computing larger problems

→ an optimization of a frequently used code pays

Brief hints on performance optimization

- Selection of the programming language:
  - Script languages (python, R) typically slow
    - Subroutines (written in C, C++,…) could be fast
  - C, Fortran give more performance
  - C++ could simply lead to inefficient code if you not know what you are doing
- Selection of compiler and compiler flags
  - Simplest way to increase the performance
  - Intel compiler for C and Fortran give usually the best performance (speedup up to a factor 1.5-2)
  - Example for compiler flags:
    - Intel compiler: `-fast` or `-ipo -O3 -xHost` (`-vec-report 5` gives hints why not optimized)
    - Gcc 4.7.1: `-Ofast -IPA -mtune=native -march=native`
    - Results should be rechecked!

Brief hints on performance optimization

- Speedup could be archived by
  - Using optimized libraries, e.g. Intel Math Kernel Library (MKL) (contains BLAS, LAPACK and FFTW routines)
- Parallel computing
  - MPI
    - Selection of MPI implementation (OpenMPI, Intel MPI)
    - Usage of the fastest interconnect , e.g. on FLOW InfiniBand (see HPC-Wiki)
    - Overlapping communication
    - Non-blocking communications
  - OpenMP/PThreads
    - Keep number of thread creation/destruction small
  - Avoid barriers/synchronization
  - Check scaling of parallelization (depends on problem size!)

Thanks!

Please visit our HPC-Wiki

`http://wiki.hpcuser.uni-oldenburg.de`