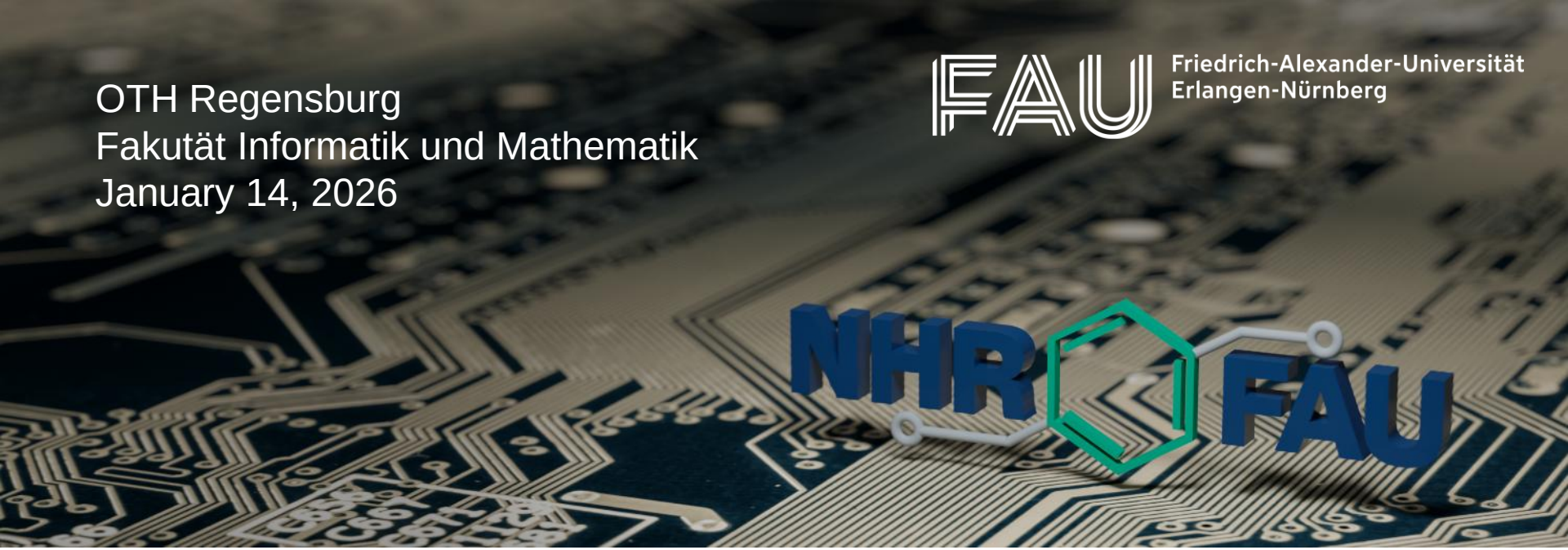


OTH Regensburg
Fakutät Informatik und Mathematik
January 14, 2026

Parallel Computing: From CPU Core to Supercomputer

Georg Hager and Alireza Ghasemi

Erlangen National High Performance Computing Center (NHR@FAU)

The plan

- An old idea that stuck:
  The stored-program computer
- Plowing the fields:
  CPU cores, chips, and nodes
- Brute-forcing the game: GPUs
- Shooting for the stars:
  High-performance networks and clusters
- Doing “work” in parallel – easy as π?
- The rule of threads: OpenMP
- We need to talk: MPI
- Some “cool” science

Image: NHR@FAU

A word about the NHR Alliance and NHR@FAU

- Funding 2021-2030: 62.5 Mio. € p.a.
- ~ 7 Mio. € per center (state+federal)
  - hardware / infrastructure
  - operational costs
  - advanced user support & training

NHR FAU

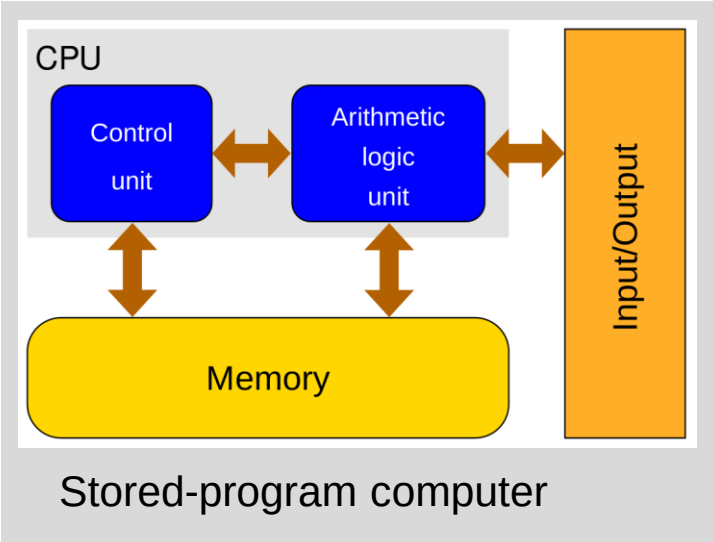
- Powerful HPC infrastructure
- Expert user support and user training
- NHR@FAU fields of expertise within NHR
  - Atomistic Simulations
  - Performance Engineering & Tools
  - AI Research

NHR FAU

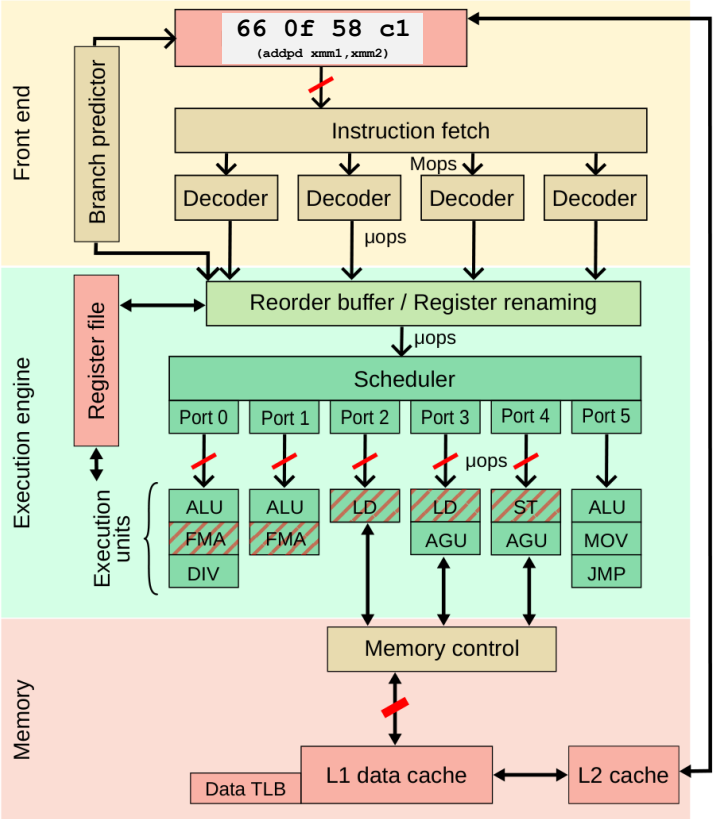
FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

An old idea that stuck: The stored-program computer

The stored-program computer



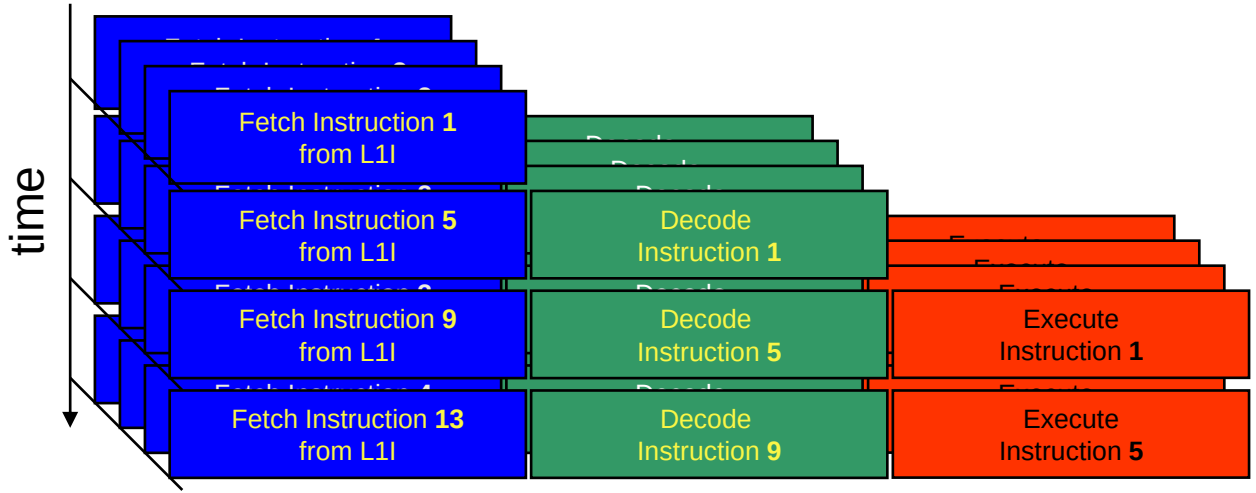
Stored-program computer



Modern CPU core

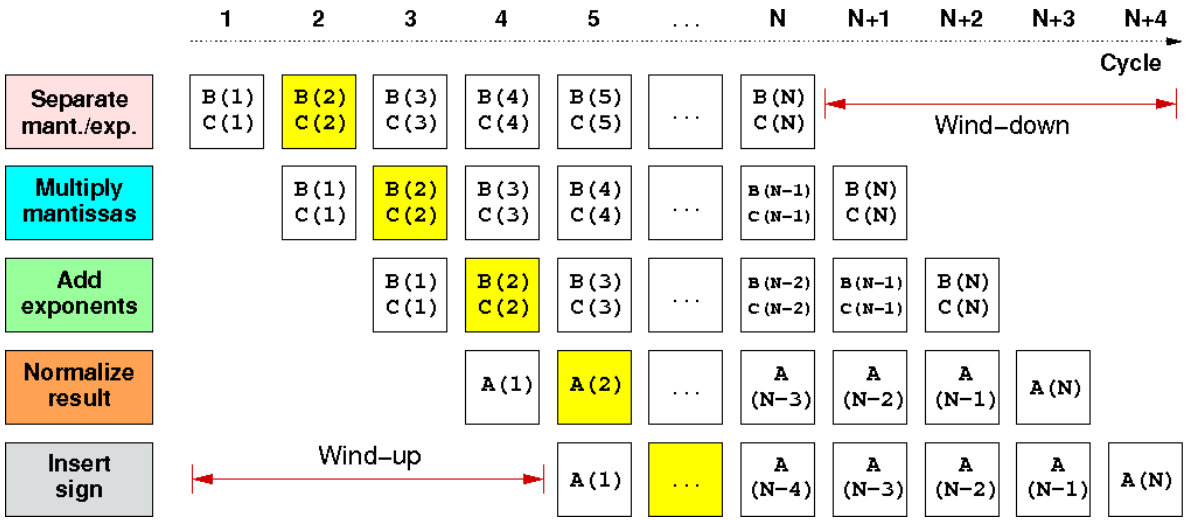
Fancy features for faster execution

Superscalarity: Multiple instructions per cycle



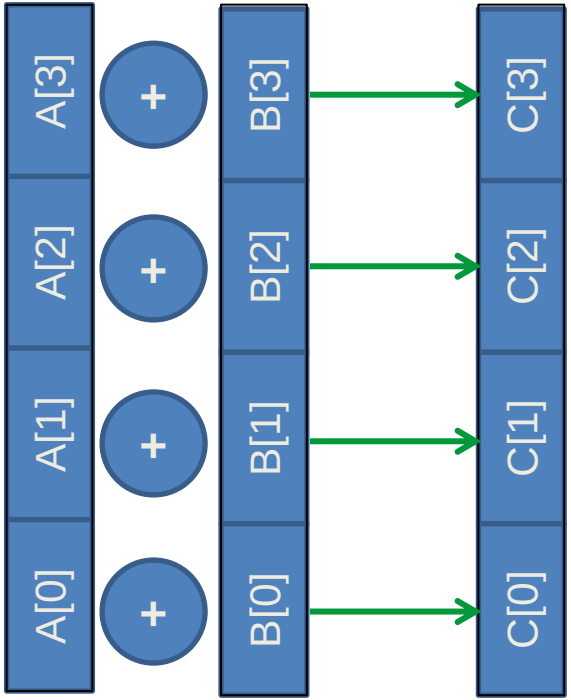
Fancy features for faster execution

Pipelining: Instruction execution in multiple steps



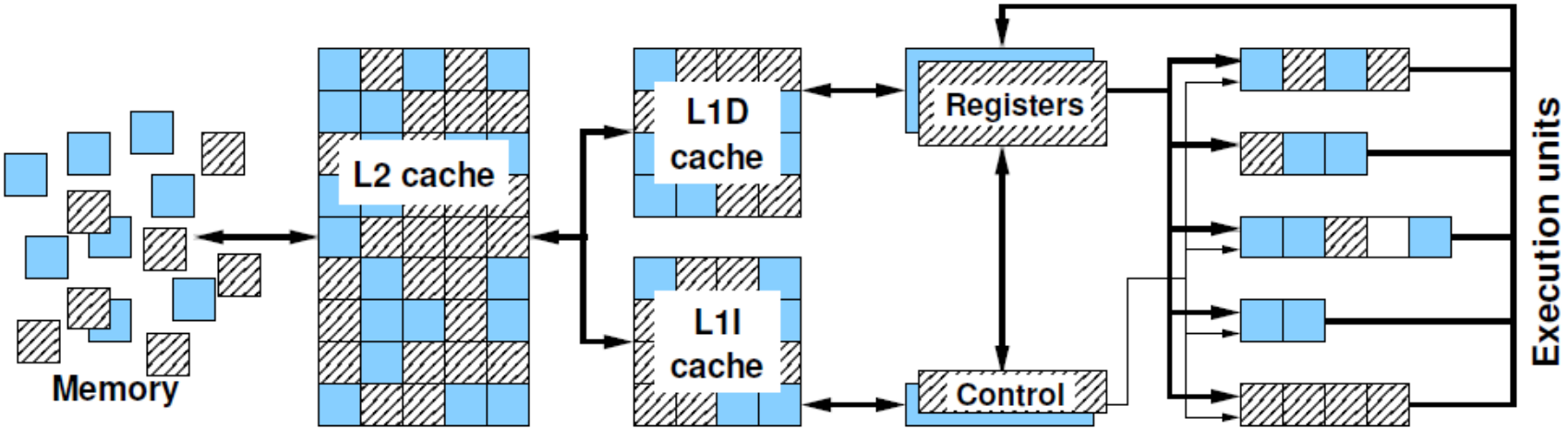
Fancy features for faster execution

Single Instruction Multiple Data: Multiple operations per instruction



Fancy features for faster execution

Simultaneous Multi-Threading:
Multiple instruction sequences in parallel



NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Plowing the fields: CPU cores, chips, and nodes

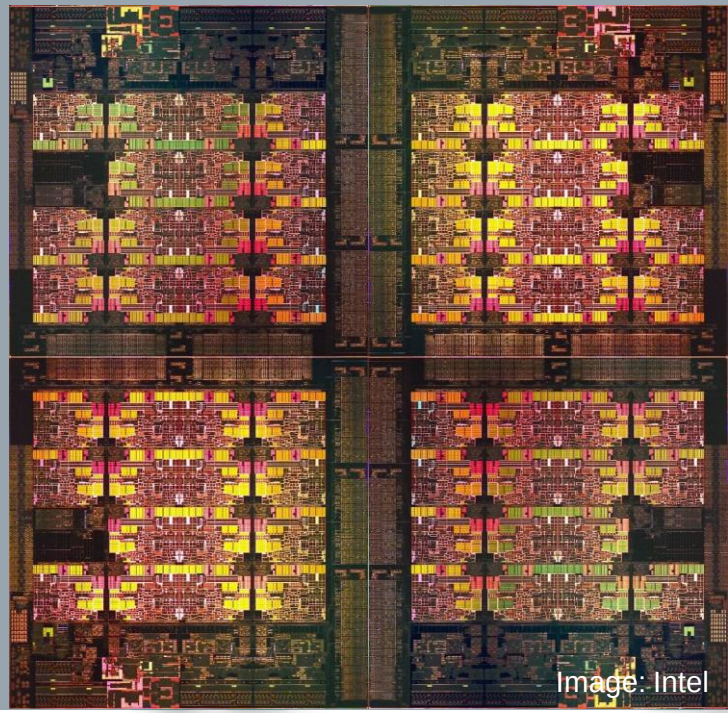
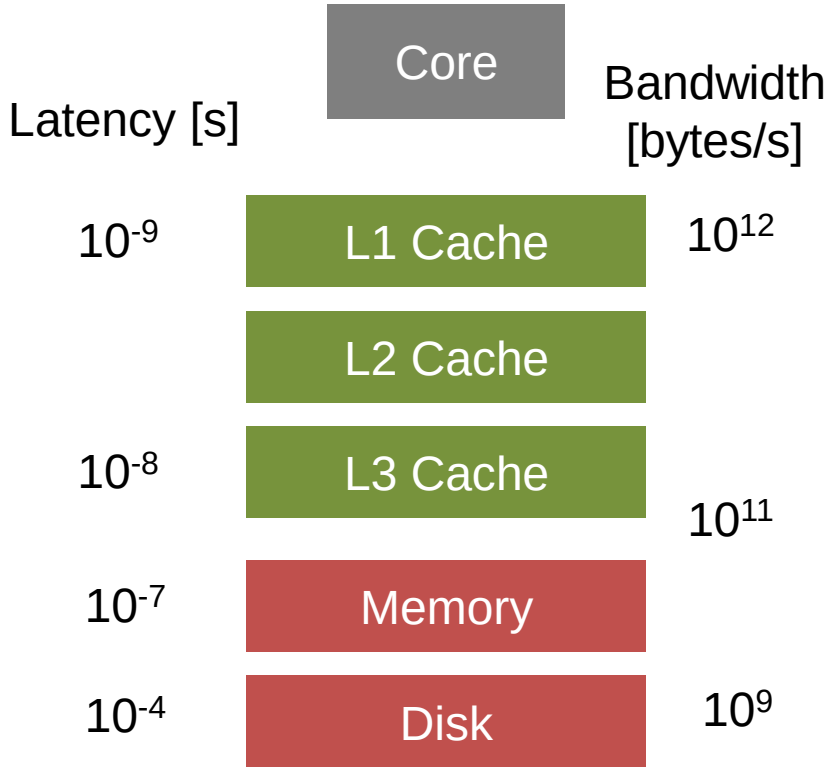


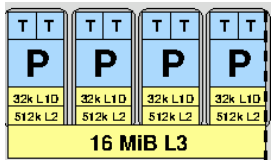
Image: Intel

Beyond the core: The cache hierarchy

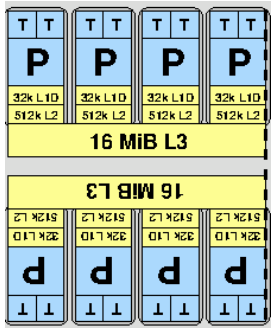
You can either build a
small and fast memory
or a
large and slow memory



Beyond a single core: multicore and multisocket



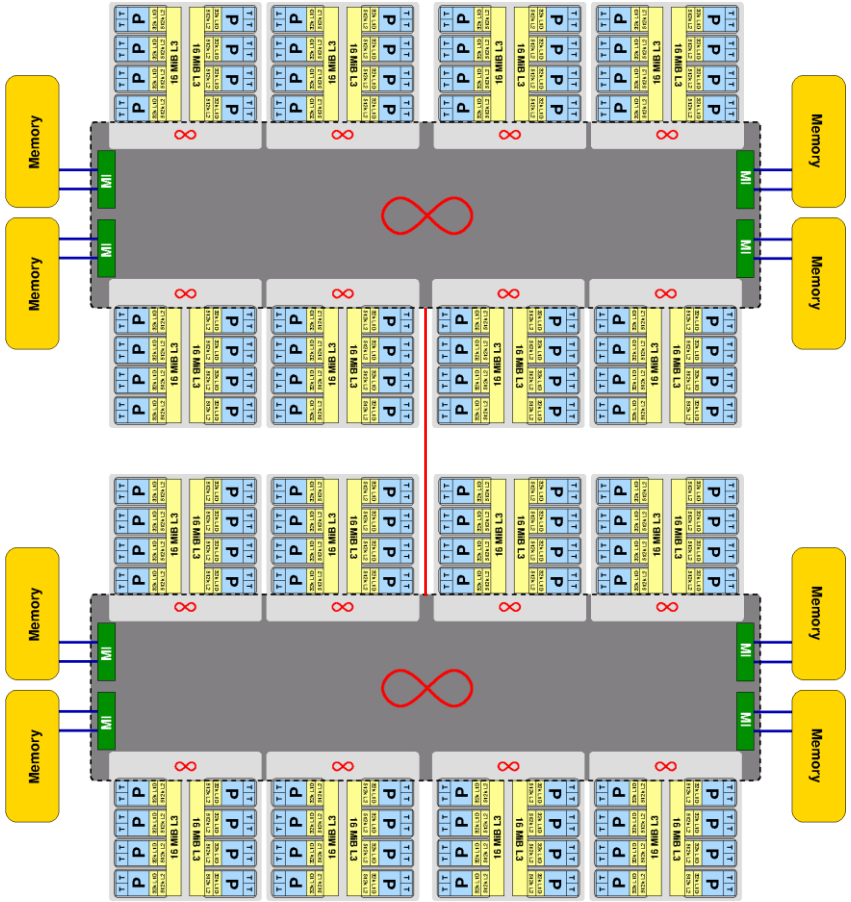
L3 group



Chiplet

CPU

Compute node



Brute-forcing the game: GPUs

Image: NVIDIA

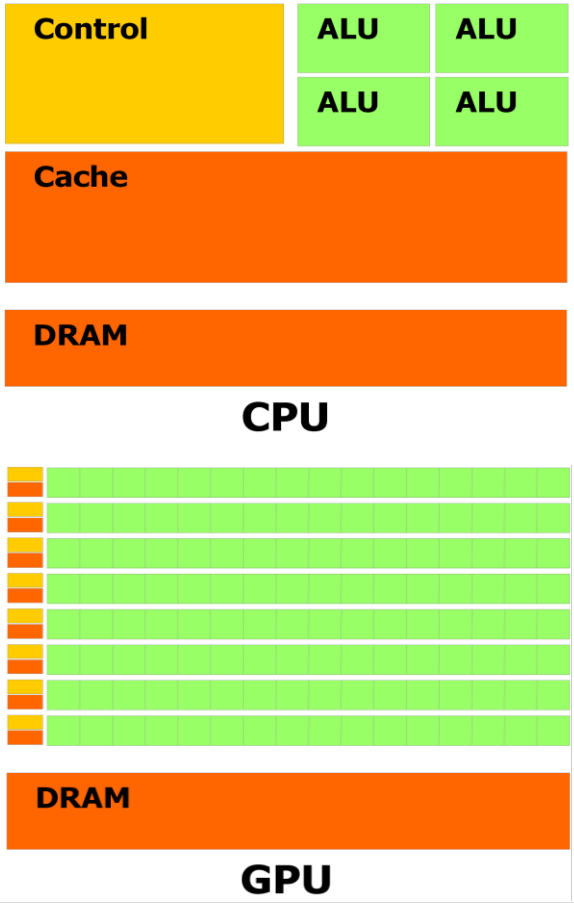
Why GPUs?

CPUs spend die area on

- … fancy core features to make serial code fast
- … lots of cache to mitigate memory delays

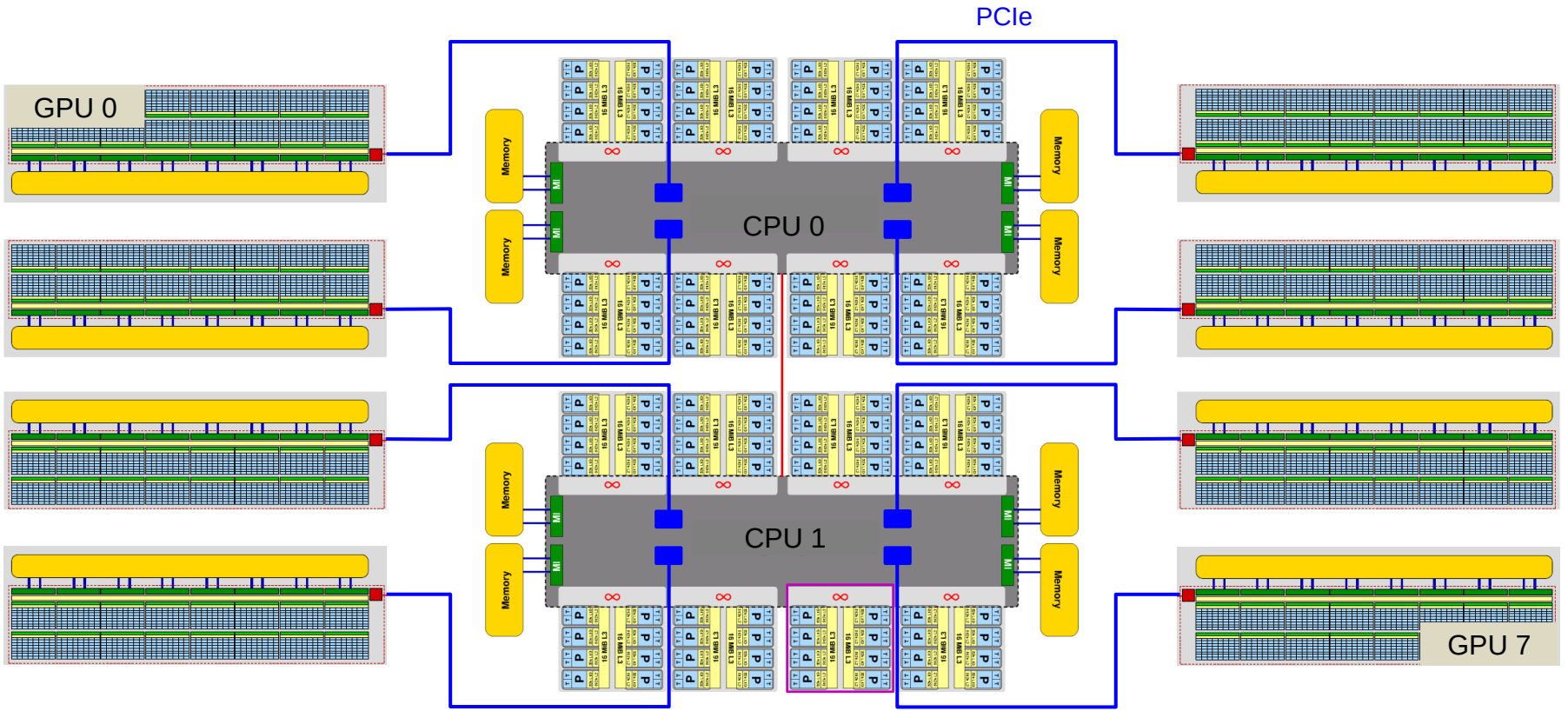
GPUs spend die area on

- … execution resources to boost computational performance



NVIDIA H100 SXM5 (80 billion transistors)

A compute node with GPUs



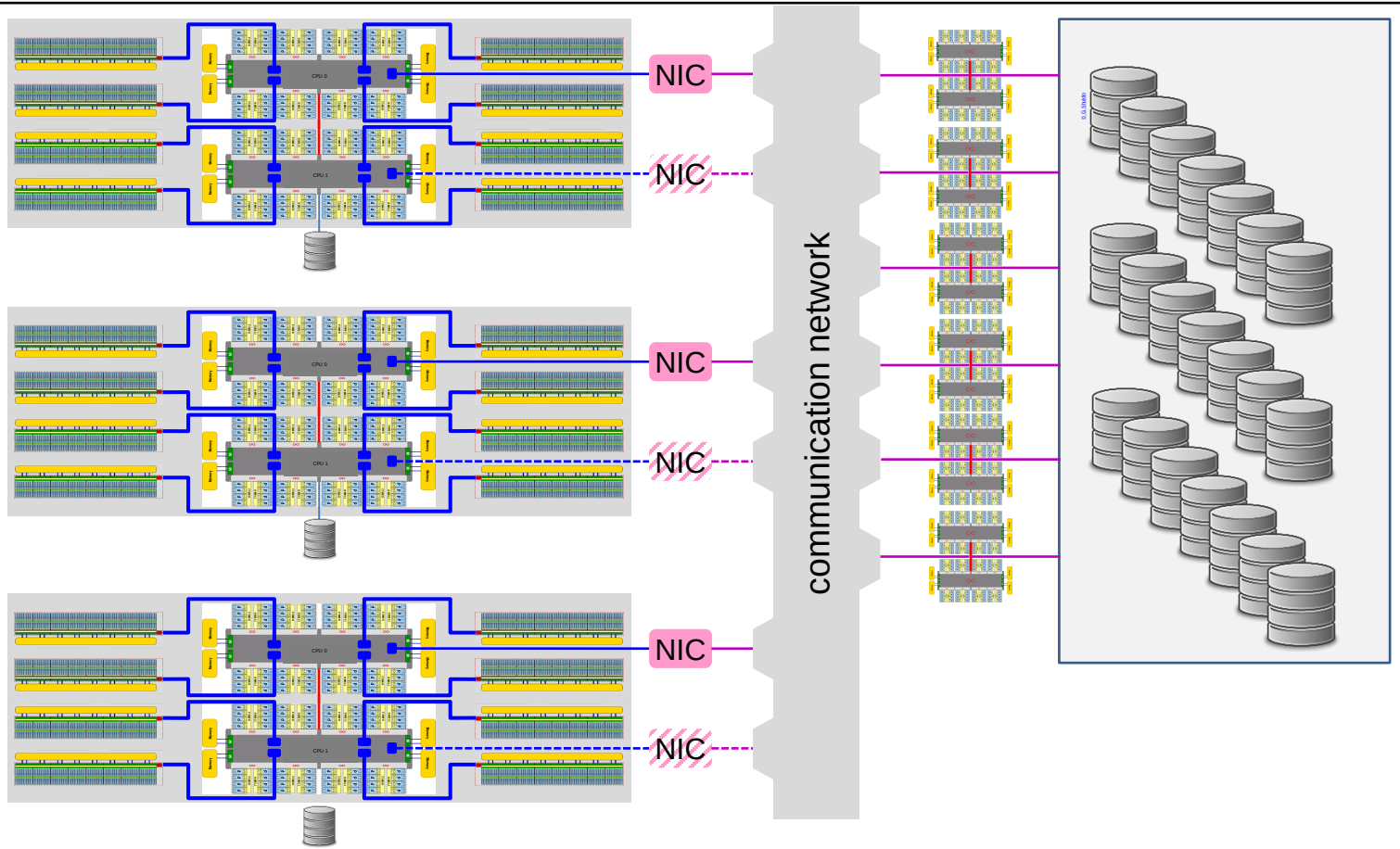
NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Shooting for the stars: High-performance networks and clusters

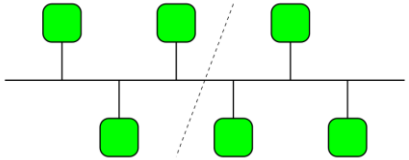
Image: NHR@FAU

Overall cluster structure

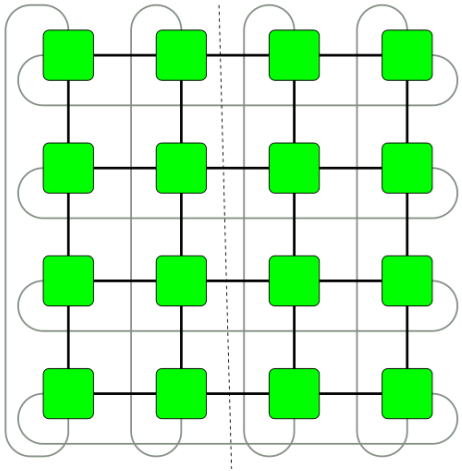


Network structures

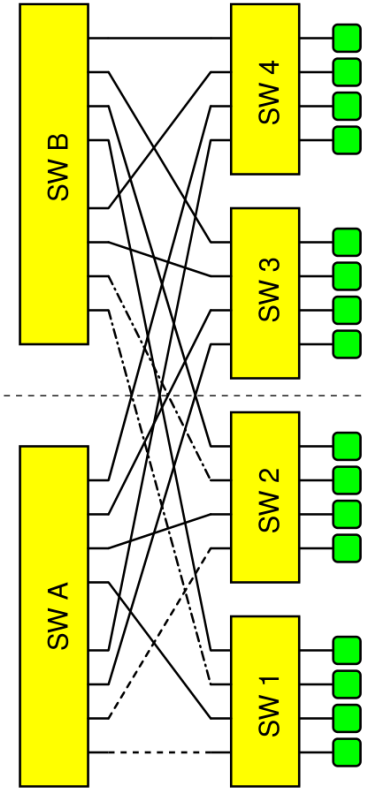
Bus



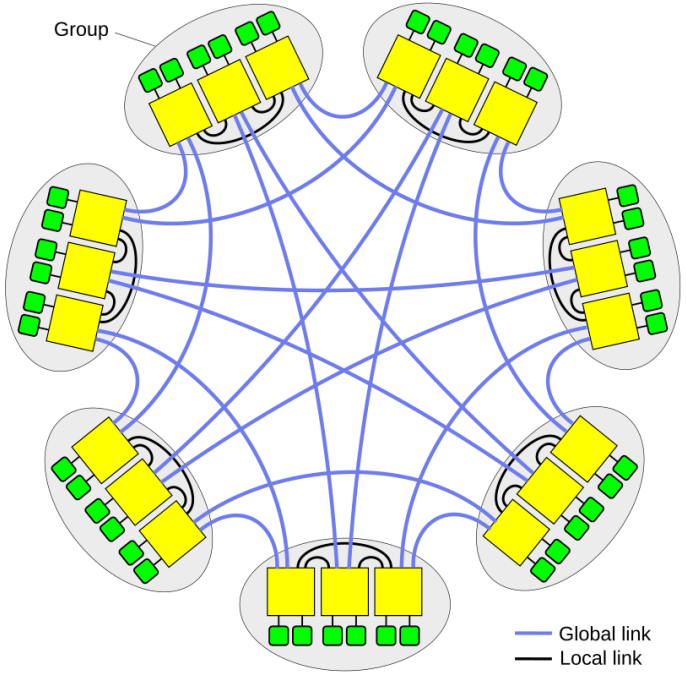
2D torus



Fat tree



Dragonfly



NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Doing “work” in parallel – easy as π?

What is “performance”?

Performance metric:

$$P = \frac{\text{Work}}{\text{Time}}$$

- „**flops**“ (+ - * /)
- lattice points
- “runs”
- “Solving the problem”
- ...

Speedup with n workers:

$$S(n) = \frac{P(n)}{P(1)}$$

Theoretical peak performance of a supercomputer

"Fugaku" – #1 in the world from 2020-2022
Still #7 today (top500.org/)

10^9 clock cycles per second (GHz)

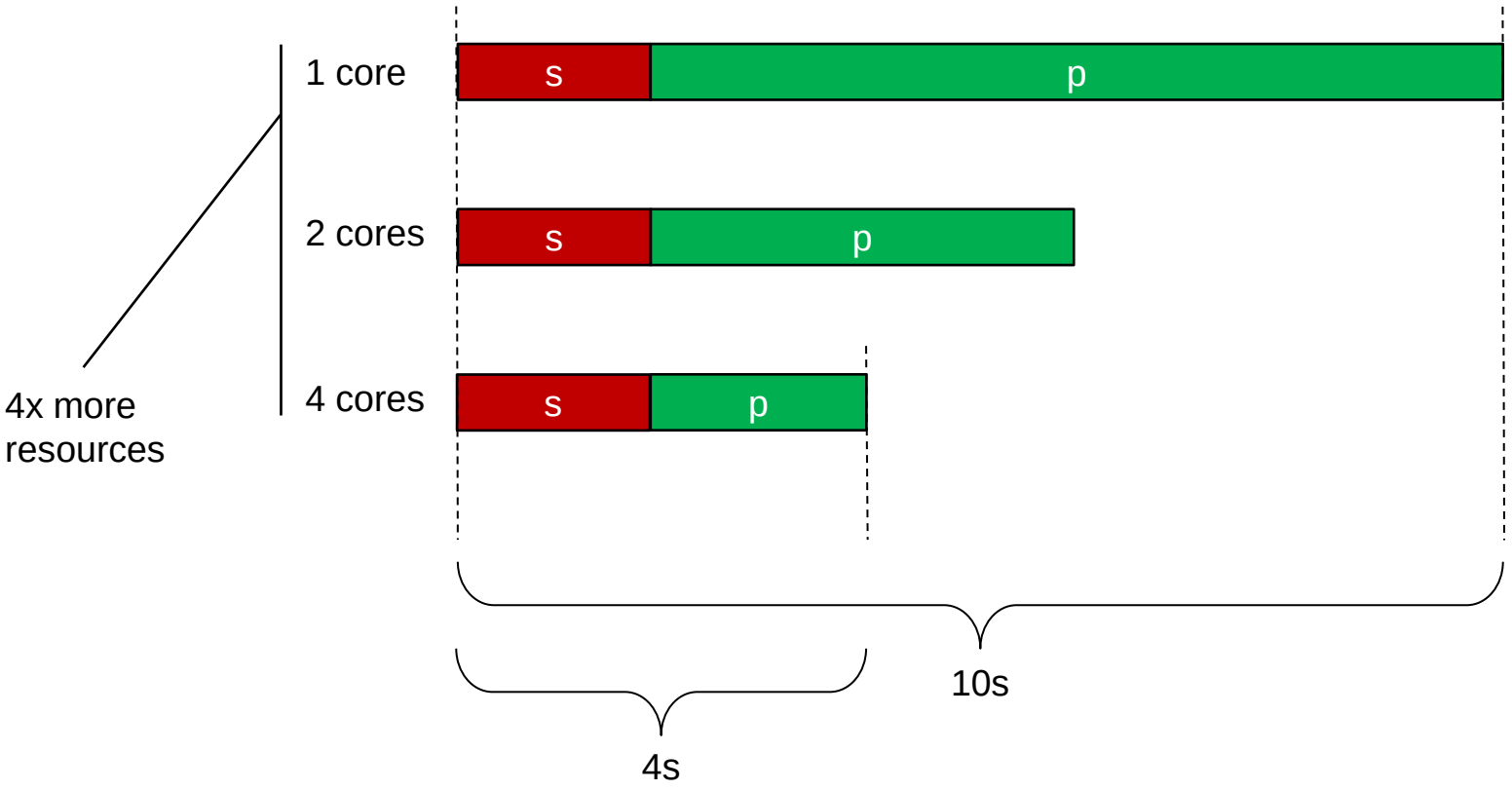
FP64 operations per cycle and core

Cores per CPU

CPUs (nodes)

$$P = 158976 \times 48 \times 2{,}2 \times 32 \frac{\text{GFlops}}{\text{s}} \approx 537 \frac{\text{PFlops}}{\text{s}}$$

Can everything be parallelized?

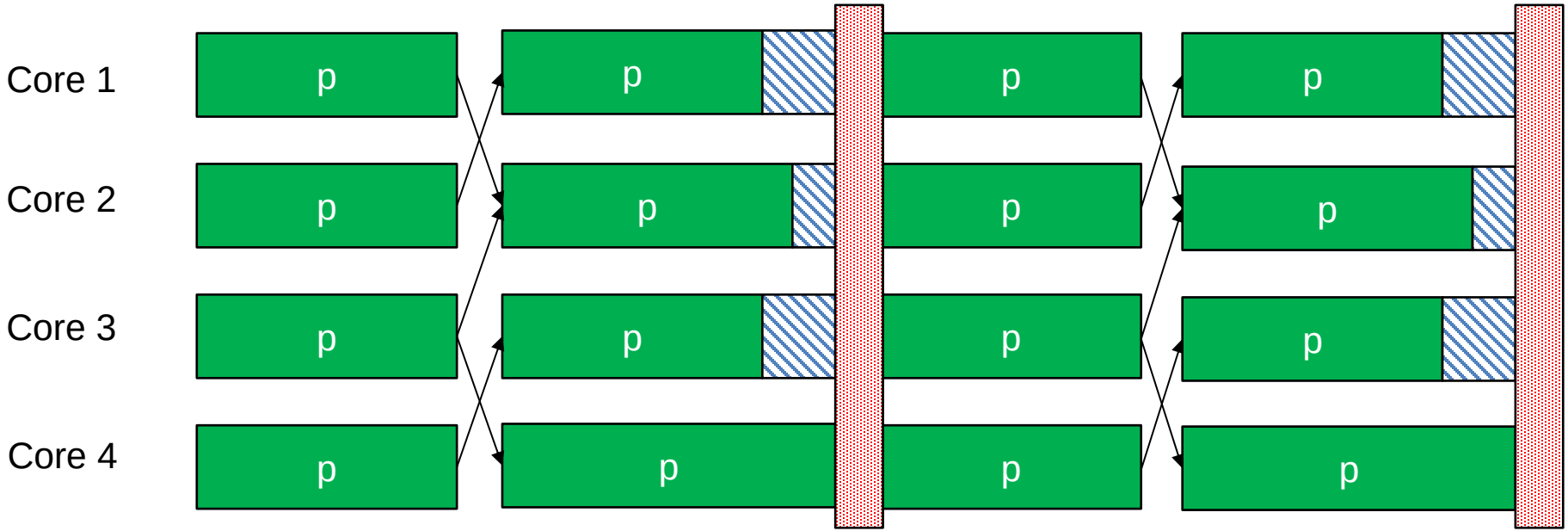


Gene Amdahl

Computer History Museum

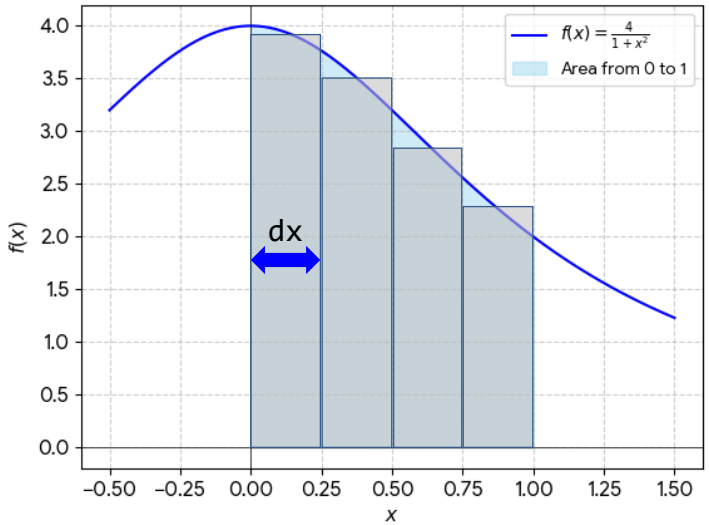
Other impediments

Communication, synchronization, load imbalance



Getting down to it – calculating π with multiple threads

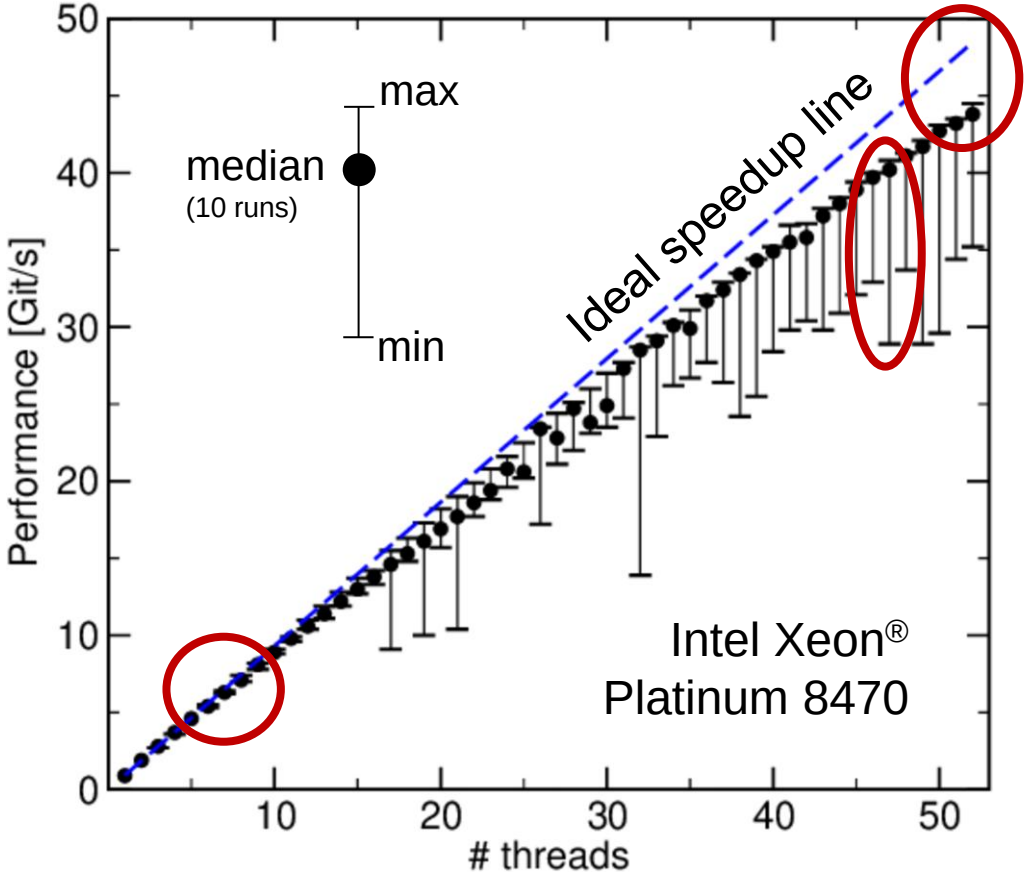
$$\pi = \int_0^1 \frac{4}{1+x^2}\,\mathrm{d}x$$




```
double pi,dx,sum;
int N=10000000;
pi = 0.; dx = 1.0/N; sum = 0.;
#pragma omp parallel
{
  #pragma omp for reduction(+:sum)
for(int i=0; i<N; ++i) {
  double x = dx*(i+0.5);
  sum += 4./(1. + x*x);
}
}
pi = sum * dx;
```

Performance and scalability: food for thought

- Serial performance ≈ 0.93 Git/s
- Good speedup ($S(n) \approx n$) at low thread counts
- Some loss of scalability at high thread counts
- Significant statistical variations at larger thread counts



NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

The rule of threads: OpenMP

OpenMP

https://www.openmp.org/

OpenMP fork-join model

Used mainly via compiler directives…

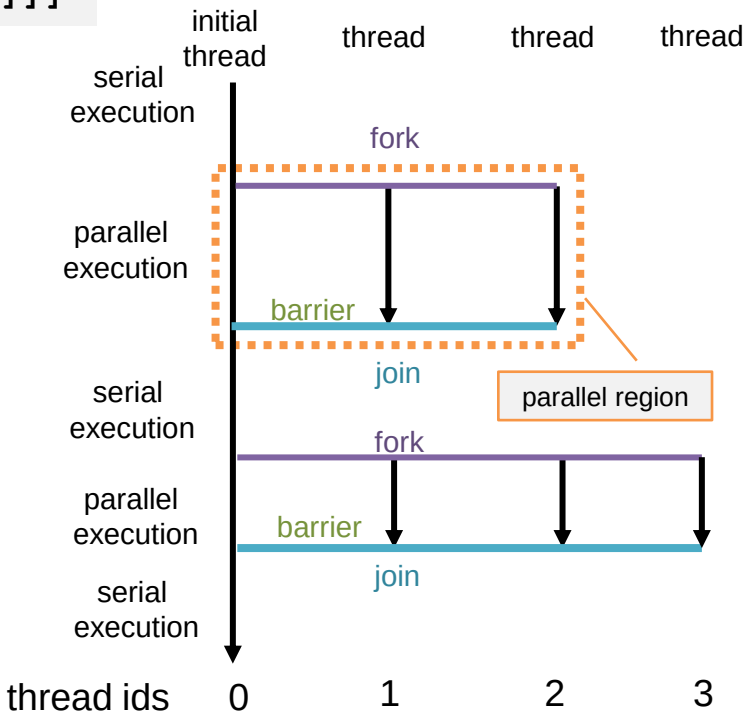
```
#pragma omp <directive> [<clause>[, <clause>[...]]]
```

… and some API functions …

```
int thread_id = omp_get_thread_num();
```

… and some environment variables:

```
$ gcc -fopenmp code.c
$ OMP_NUM_THREADS=4 ./a.out
```



OpenMP “Hello World”

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char *argv[])
{
  printf("Sequential part\n");

  #pragma omp parallel
  {
    printf("Thread %d of %d\n",
           omp_get_thread_num(),
           omp_get_num_threads());
  }
  printf("Sequential part again\n");
  return 0;
}
```

```
$ gcc -fopenmp code.c
$ OMP_NUM_THREADS=4 ./a.out
Sequential part
Thread 1 of 4
Thread 3 of 4
Thread 2 of 4
Thread 0 of 4
Sequential part again
$ OMP_NUM_THREADS=6 ./a.out
Sequential part
Thread 0 of 6
Thread 1 of 6
Thread 5 of 6
Thread 4 of 6
Thread 2 of 6
Thread 3 of 6
Sequential part again
```

Sharing the work among threads

```
#pragma omp parallel
{
  #pragma omp for
  for (int i = 0; i < n; ++i) {
    a[i] = b[i];
  }
}
```

Work is spread across threads in team

Support for reductions:

```
double sum=0.;
// ...
#pragma omp parallel reduction(+:sum)
{
  #pragma omp for
  for (int i = 0; i < n; ++i) {
    sum += a[i];
  }
}
```

Critical regions

```
#pragma omp parallel
{
  #pragma omp for
  for (int i = 0; i < n; ++i) {
    #pragma omp critical
    sum += device_read();
  }
}
```

Advanced OpenMP: tasking

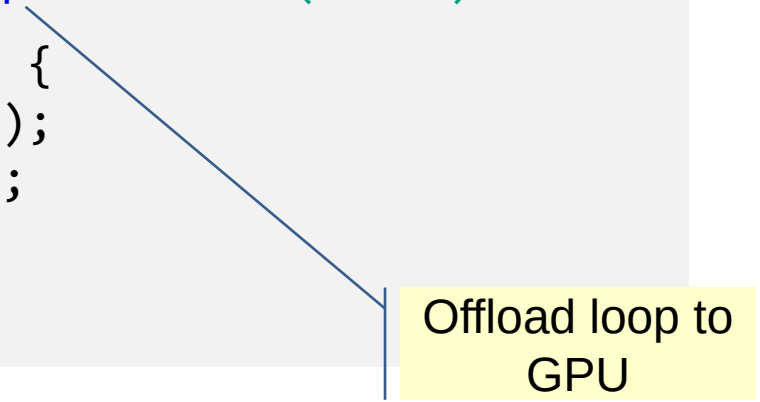
```
E *anchor = ... // linked list
#pragma omp parallel
{
  #pragma omp single
  {
    for(E *ptr=anchor; ptr!=NULL; ptr=ptr->next) {
      #pragma omp task
      do_stuff_with_data(ptr->payload);
    }
  } // <-- implicit barrier here
}
```

One thread fills "queue of tasks"

Other threads waiting and working on tasks in queue

Advanced OpenMP: device offloading

```
double pi,dx,sum;
int N=10000000;
pi = 0.; dx = 1.0/N; sum = 0.;
#pragma omp target loop reduction(+:sum)
for(int i=0; i<N; ++i) {
  double x = dx*(i+0.5);
  sum += 4./(1. + x*x);
}
pi = sum * dx;
```



OpenMP: What we have left out

- Data scoping (thread-private/shared data structures)
- Loop scheduling (which thread does what)
- Advanced synchronization (atomics, locks)
- Advanced tasking (taskloops, dependencies)
- SIMD support (vectorizing loops and functions)
- Advanced offloading (asynchronous execution, tasks, dependencies)
- Execution modalities (thread affinity, memory affinity)
- ...

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Message Passing Interface (MPI)

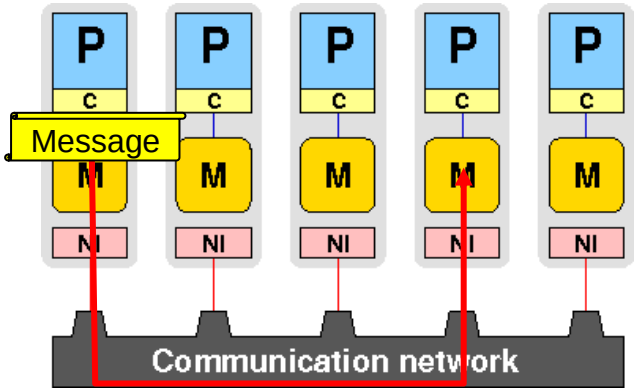
The message passing paradigm

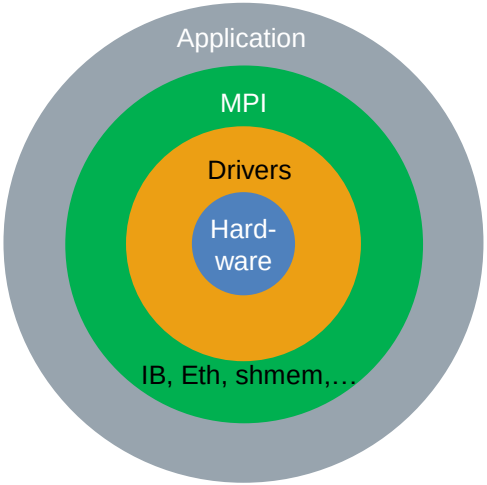
Distributed-memory architecture:

- Each process(or) can only access its dedicated address space.
- No global shared address space
- Data exchange and communication between processes is done by explicitly passing messages through a communication network

Message passing library:

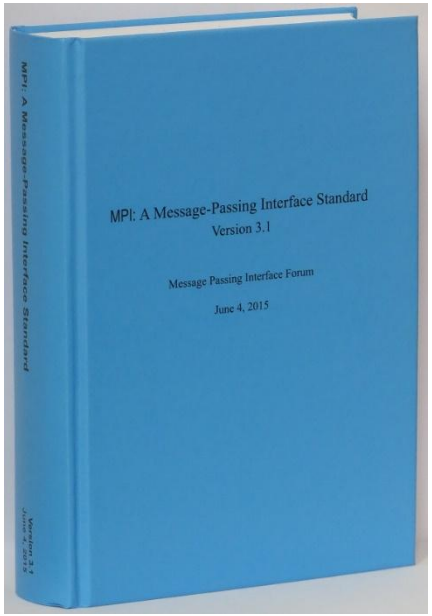
- Should be flexible, efficient and portable
- Hide communication, hardware and software layers from application developer



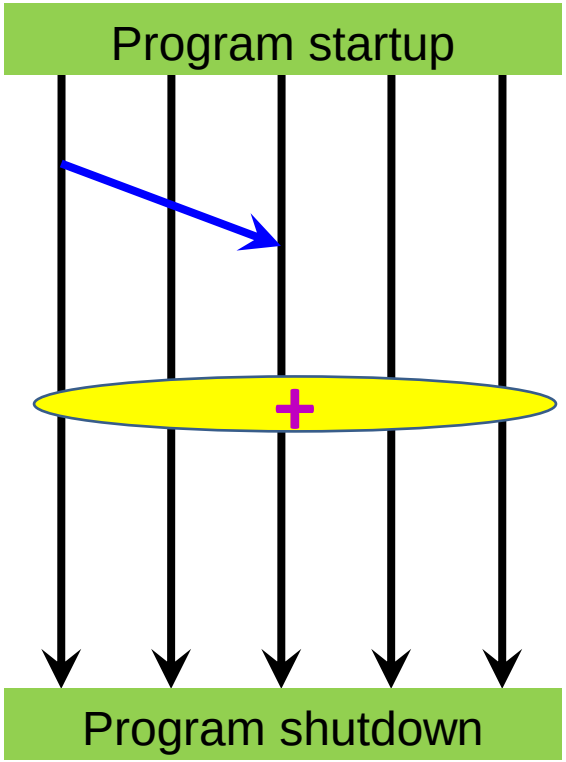


The message passing paradigm

- Message Passing Interface (MPI): Widely accepted standard in HPC
- The program is written in a sequential language (Fortran/C[++]), but …
- Data exchange between processes:
  - Send/receive messages via MPI library calls
  - No automatic workload distribution
- The MPI standard
  - MPI forum – defines MPI standard/library components
  - **http://www.mpi-forum.org/**
- Latest version: MPI 5.0, Release 05.06.2025 (libraries ?)
- Libraries: MPICH, mvapich, OpenMPI, … and Intel, Cray, HP,…



Parallel execution in MPI



- Processes run throughout program execution
- MPI Point-to-point communication:
  - between pairs of tasks/processes
- MPI Collective communication:
  - between all processes or a subgroup
  - barrier, reductions, scatter/gather
- Clean shutdown by MPI

Point-to-Point Communication

It is a communication between two processes where a sender (source process) sends message to a receiver (destination process).

- Procedure (C/C++ binding, Fortran binding, Fortran 2008 binding)
- Message data
  - Buffer (address)
  - Datatype (basic or derived?)
  - Count (number of elements, not bytes)
- Message envelope
  - Source
  - Destination
  - Tag

Basic Datatypes (C/C++)

MPI datatype	C datatype
MPI_INT	int
MPI_UNSIGNED	unsigned int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_C_COMPLEX	float _Complex
MPI_C_DOUBLE_COMPLEX	double _Complex
MPI_C_BOOL	_Bool
MPI_CHAR	char
and many more -> https://www.mpi-forum.org/docs/	

MPI_SEND and MPI_RECV

- MPI_Send C/C++ binding:

```
#include <mpi.h>
int MPI_Send(const void *buf, int count, MPI_Datatype datatype,
int dest,int tag, MPI_Comm comm)
```

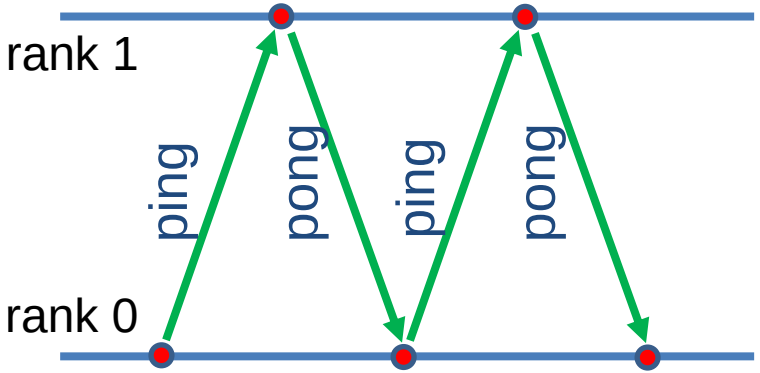
- MPI_Recv C/C++ binding:

```
#include <mpi.h>
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
             int source,int tag, MPI_Comm comm,
             MPI_Status *status)
```

- buf: address of the first entry of the buffer to be sent
- count: number of elements to be sent (note that it is not bytes!)

Single-round ping-pong in C

```c
#include <mpi.h>
#include <stdio.h>
int main(int argc, char **argv) {
      int ierr, irank, nrank;
      MPI_Status status;
      double d=0.0;
      ierr=MPI_Init(&argc,&argv);
      ierr=MPI_Comm_rank(MPI_COMM_WORLD,&irank);
      ierr=MPI_Comm_size(MPI_COMM_WORLD,&nrank);
      if(irank==0) d=100.0;
      if(irank==1) d=200.0;
      printf("BEFORE: nrank,irank,d = %5d%5d%8.1f\n",nrank,irank,d);
      if(irank==0) {
          MPI_Send(&d,1,MPI_DOUBLE,1,11,MPI_COMM_WORLD);
          MPI_Recv(&d,1,MPI_DOUBLE,1,22,MPI_COMM_WORLD,&status);
      }
      else if(irank==1) {
          MPI_Send(&d,1,MPI_DOUBLE,0,22,MPI_COMM_WORLD);
          MPI_Recv(&d,1,MPI_DOUBLE,0,11,MPI_COMM_WORLD,&status);
      }
      printf("AFTER: nrank,irank,d = %5d%5d%8.1f\n",nrank,irank,d);
      ierr=MPI_Finalize();
}
```



A deadlock is a scenario in which a process is trying to exchange data to another process but there is no match, e.g. it is ready to send a data but the other process is not and will not be prepared to accept or the opposite case, i.e. the process is waiting to receive but the other is not sending and will not send a matching message.

How to compile and run an MPI program

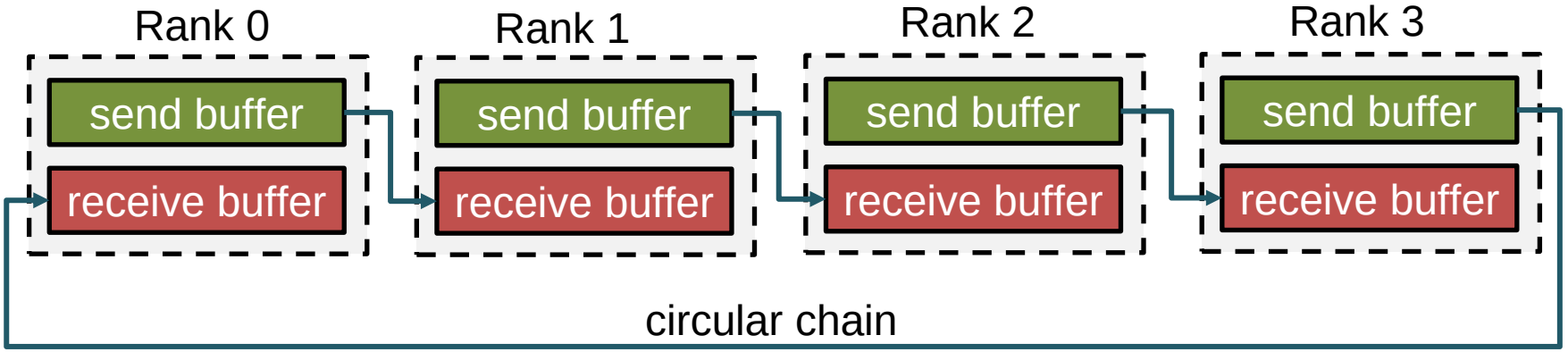
- MPI implementations provide wrappers to the compilers, e.g. **mpicc**

```
$ mpicc pingpong.c -o pingpong
```

- To execute the program, one should use startup wrappers such as **mpirun**, **mpiexec**, … or a job scheduler wrapper like **srun**

```
$ mpirun -np 2 ./pingpong
BEFORE: nrank,irank,d =     2    1   200.0
BEFORE: nrank,irank,d =     2    0   100.0
AFTER:  nrank,irank,d =     2    0   100.0
AFTER:  nrank,irank,d =     2    1   100.0
```

Example: Shift operation across a chain of processes

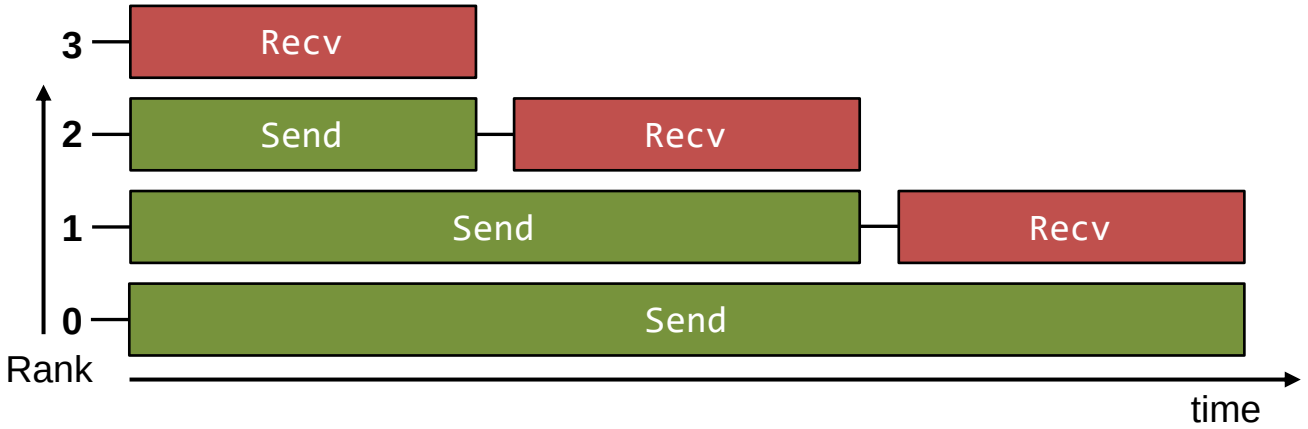


- Simplistic send/recv
  - pairing is not reliable

```
//my left neighbor
left=(rank-1)%size;
//my right neighbor
right=(rank+1)%size;
MPI_Send(sendbuf,n,type,right,tag,comm);
MPI_Recv(recvbuf,n,type,left,tag,comm,status);
```

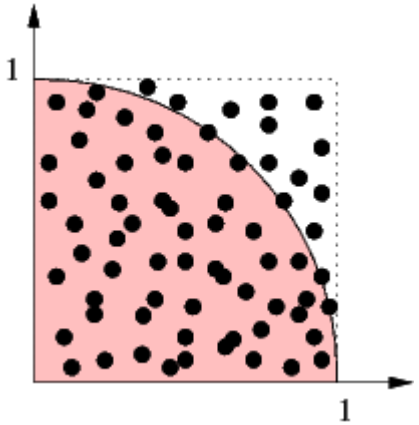
Serialization: Loss of efficiency

- Ring shift communication pattern: non-circular shifts
  - No concern over deadlock
  - Serialization
    - MPI_Send with rendezvous protocol
    - MPI_Ssend



Calculating PI with Monte Carlo

- The quarter circle in the first quadrant with origin at (0,0) and radius 1 has an area of π/4. We look at the random number pairs in [0, 1] × [0, 1]. The probability that such a point lies inside the quarter circle is π/4, so given enough statistics we are able to calculate π using this “Monte Carlo” method.




```
one_over_rand_max=1.0/(double)RAND_MAX;
count=0.0;
for(i=0; i<nn; ++i) {
    x=rand_r(&seed)*one_over_rand_max;
    y=rand_r(&seed)*one_over_rand_max;
    if(x*x+y*y <1.0) ++count;
}
pi=4.0*count/(double)nn;
```

Calculating PI with Monte Carlo

```
nn=pow(10,9);
n_local=nn/nrank;
if(irank<nn%nrank) n_local+=1; //doing one more point if nrank is not a divisor (factor) of nn
one_over_rand_max=1.0/(double)RAND_MAX;
seed=2+irank;
count=0.0;
for(i=0; i<n_local; ++i) {
    x=rand_r(&seed)*one_over_rand_max;
    y=rand_r(&seed)*one_over_rand_max;
    if(x*x+y*y <1.0) ++count;
}
if(0==irank) {
    //This is rank=0 and it receives `count' from all other ranks
    for(i=1; i<nrank; ++i) {
        MPI_Recv(&val,1,MPI_DOUBLE,i,0,MPI_COMM_WORLD,&status);
        count+=val;
    }
    pi=4.0*count/(double)nn;
    printf(accuracy: %14.5E\n",fabs(M_PI-pi)/M_PI);
}
else {
    //Every rank except rank=0 should send its `count' to rank=0
    MPI_Send(&count,1,MPI_DOUBLE,0,0,MPI_COMM_WORLD);
}
```

What we have left out

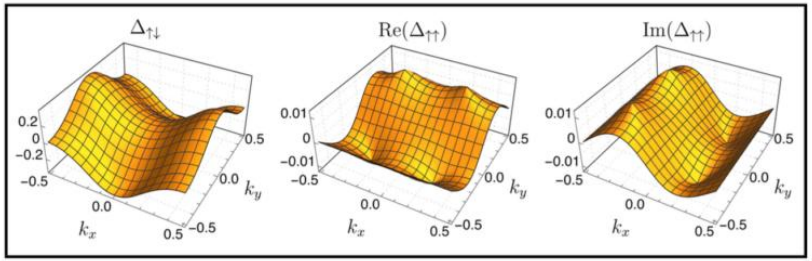
- Point-to-Point communication
  - Communication modes
  - Non-blocking
- Synchronization
- Collective communication
- Subcommunicators
- Derived types
- Parallel I/O
- Topology
- Shared memory

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

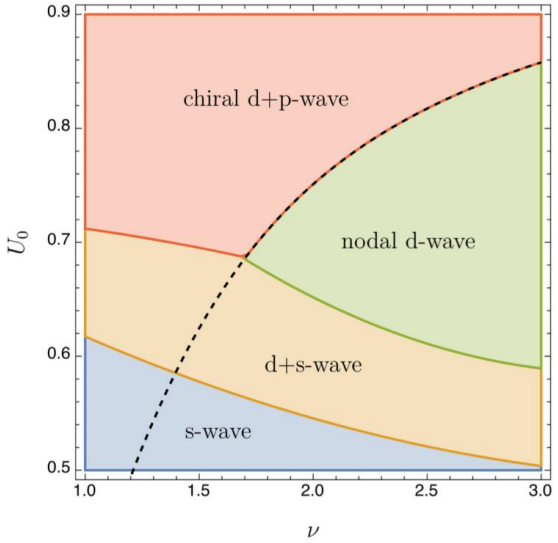
Cool Science

Unconventional Superconductors (A. Buchheit, U Saarland)

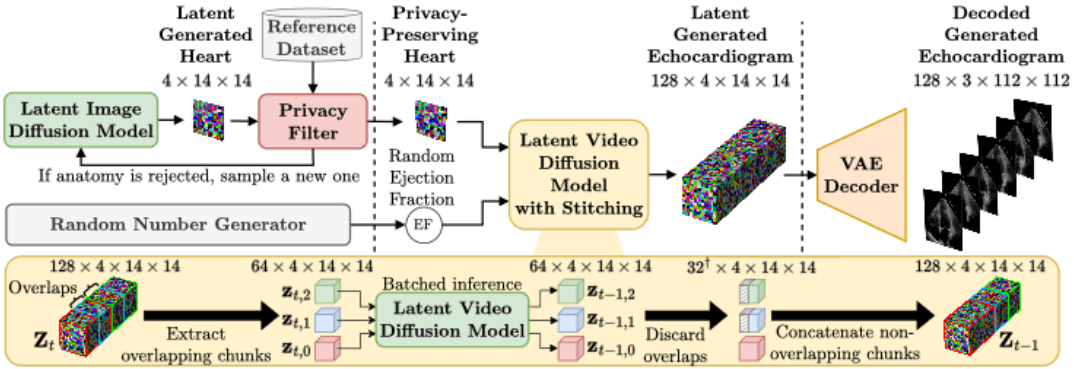


$$\Delta(k) = \int_{\mathrm{BZ}} \Big(C_0 + U_0 Z_{\Lambda,\nu}(k-q) \Big) \frac{\Delta(q)}{2\sqrt{\xi^2(q)+|\Delta(q)|^2}} \, \mathrm{d}q$$

https://youtu.be/D03loILYeTA

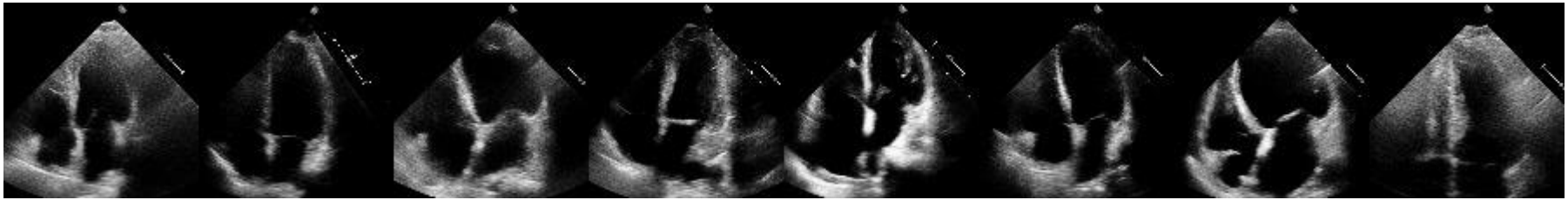


Generative Video Models for Privacy-Preserving Medical AI (B. Kainz, FAU)



Synthetic data sets for AI model training

https://github.com/HReynaud/EchoNet-Synthetic?tab=readme-ov-file



NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

“I need this – how do I get access?”

The application process for NHR projects at NHR@FAU

PI at German university or U.A.S.

NHR FAU

Researchers

Application

Technical Review

grant access

External Review

Steering Commitee

Use of HPC resources

final allocation

Decision

End of Project

Report

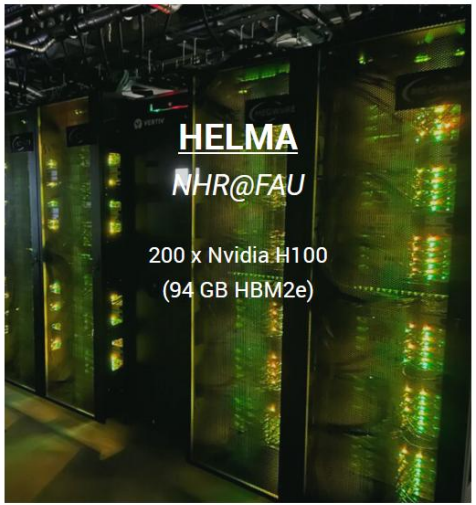
https://nhr.fau.de/

AI projects in Bavaria

Researchers at state-funded Bavarian universities and universities of applied sciences (HAW, TH, and FH) have **free of charge** access to these current hardware resources via **BayernKI Wissenschaft**:

BAYERN KI WISSENSCHAFT

https://www.ki-in-bayern.de/



HIGHTECH Agenda Bayern

Focus is on AI research (training, methodology), not inference

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Thank you.

https://nhr.fau.de/

Bundesministerium für Bildung und Forschung

DFG Deutsche Forschungsgemeinschaft

HIGHTECH Agenda Bayern