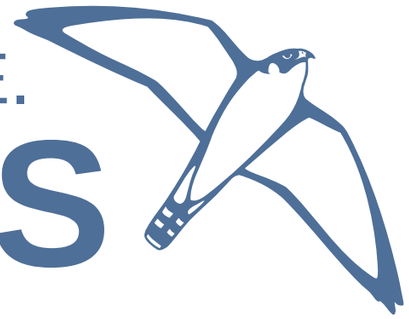
FAST. FLEXIBLE. FREE.

GROMACS

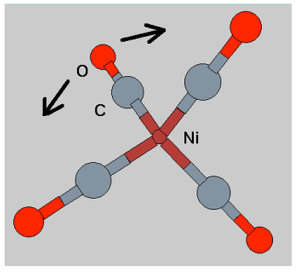


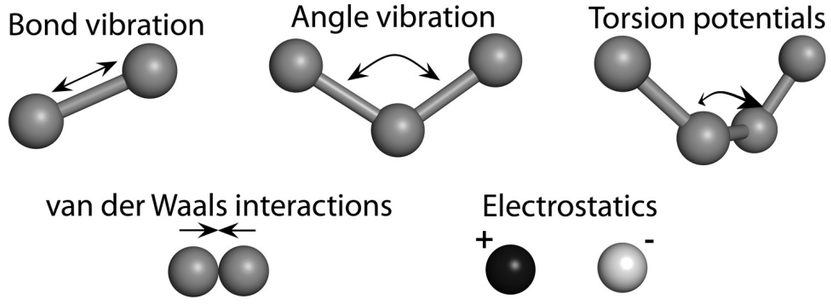
Parallel Algorithms for Heterogeneous Architectures

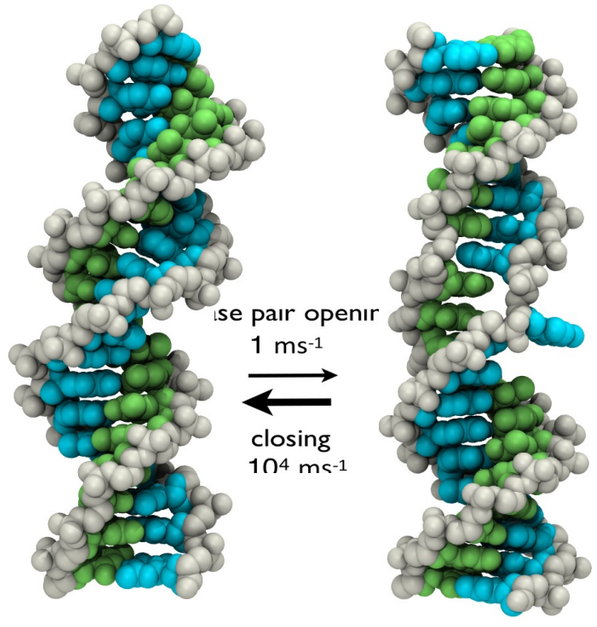
Szilárd Páll
pszilard@kth.se

GROMACS on LUMI Workshop
January 24, 2024

Why parallelize?
MD timescale challenge





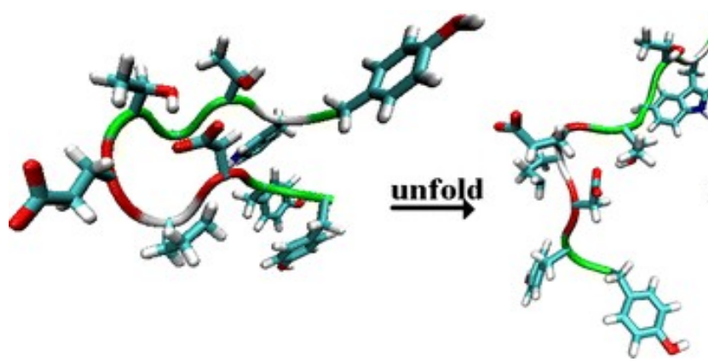


Physics | Chemistr | Biology

10^{-15} s | 10^{-12} s | 10^{-9} s | 10^{-6} s | 10^{-3} s | 10^{0} s | 10^{3} s

Simulations:

- high spatial/temporal detail
- sampling bottleneck
- model quality?

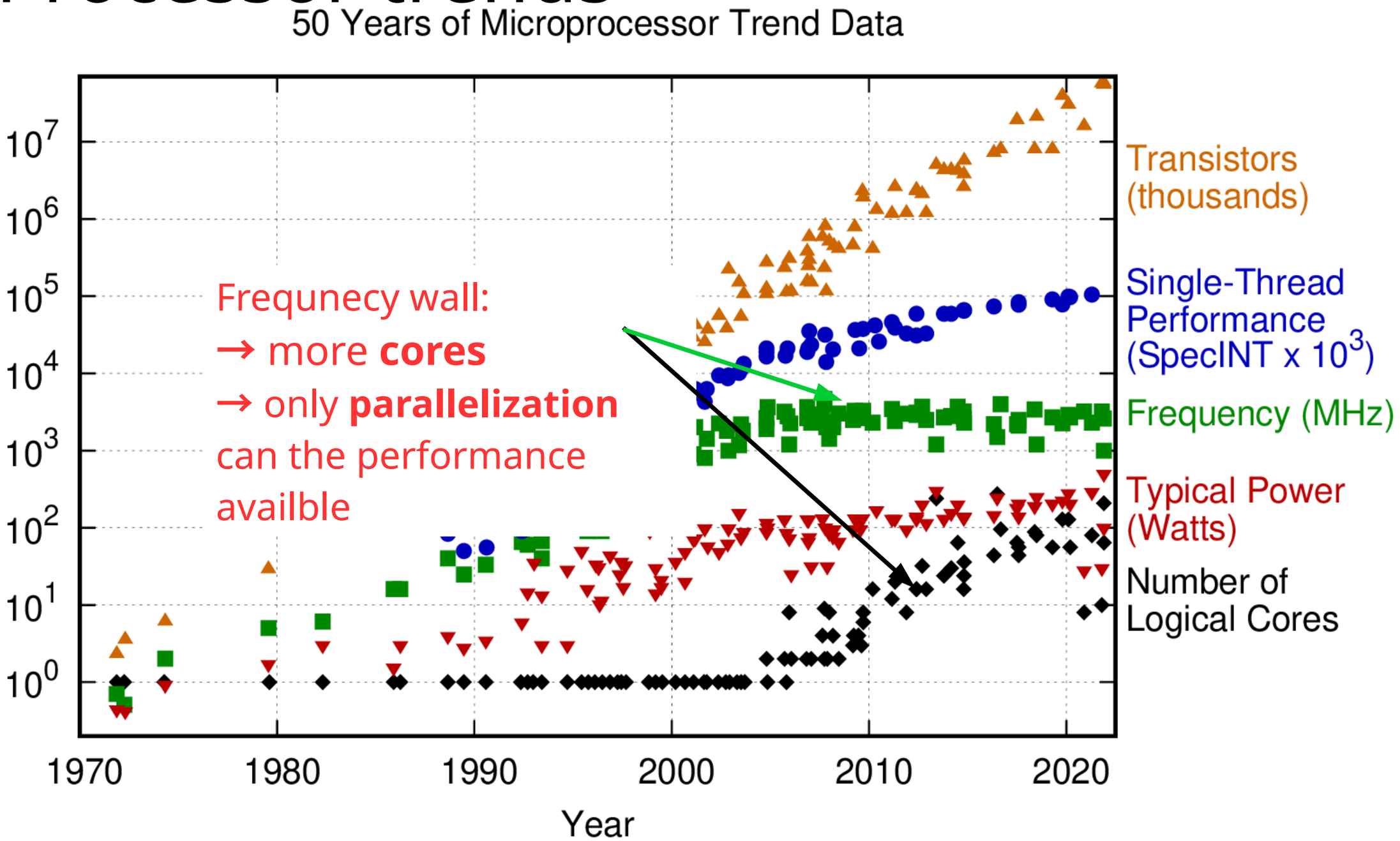


Laboratory experiments:

- lower detail
- higher efficiency
- high degree of averaging

Why parallelize?

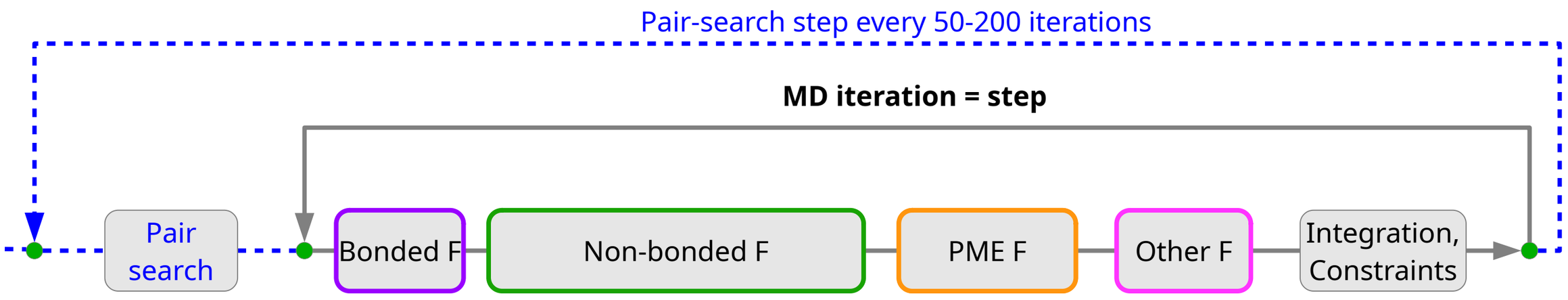
Processor trends



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shach
New plot and data collected for 2010-2021 by K. Rupp

Slowing of Moore scaling,
new approaches needed:
architectural specialization &
component integration.

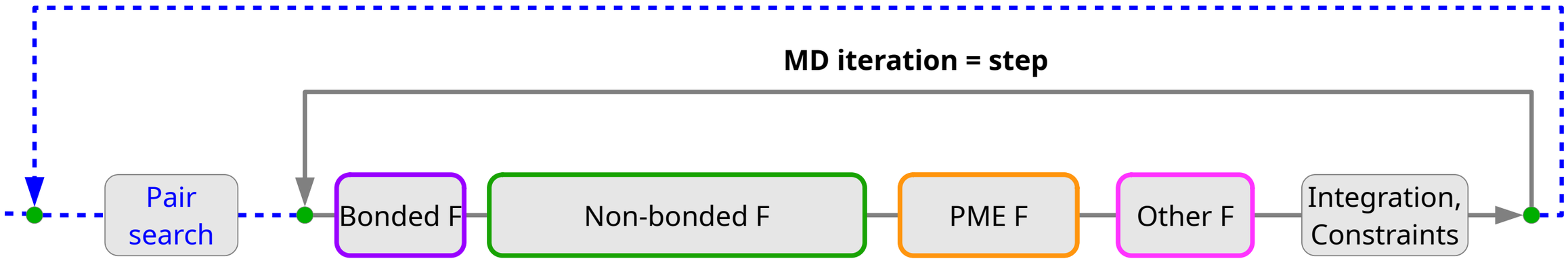
Molecular dynamics step

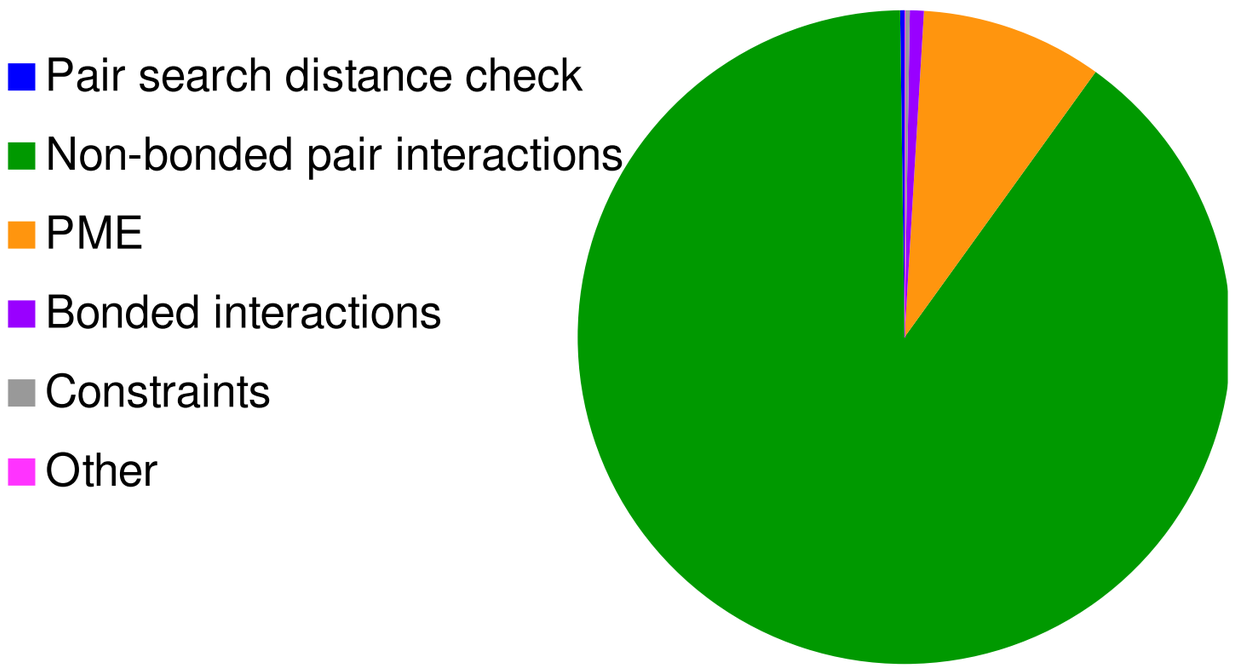


~ millisecond or less

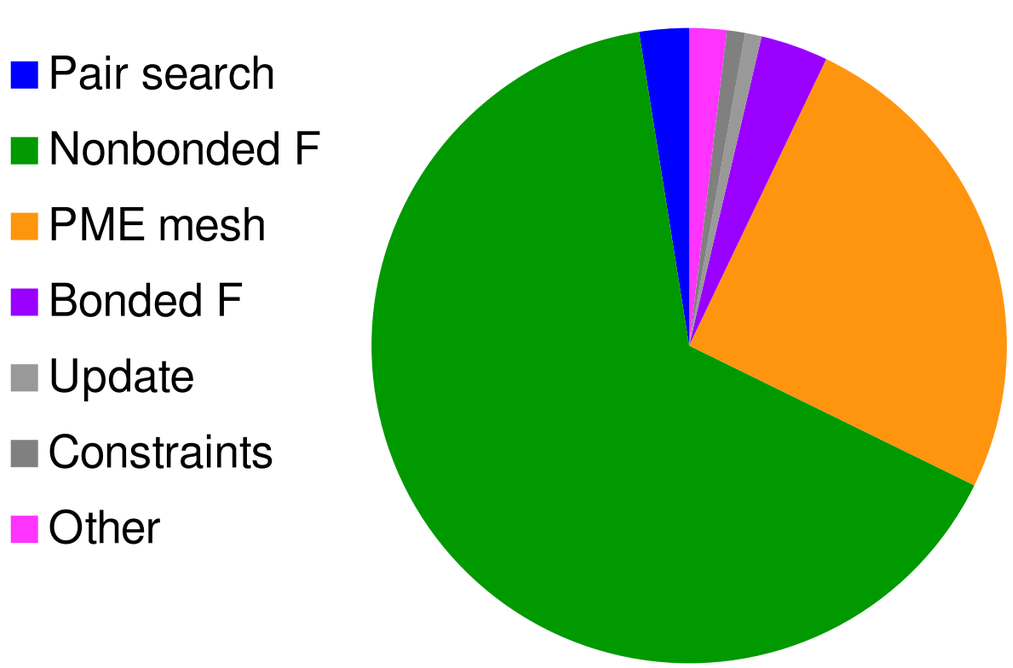
Goal: do it as fast as possible!

Computational costs



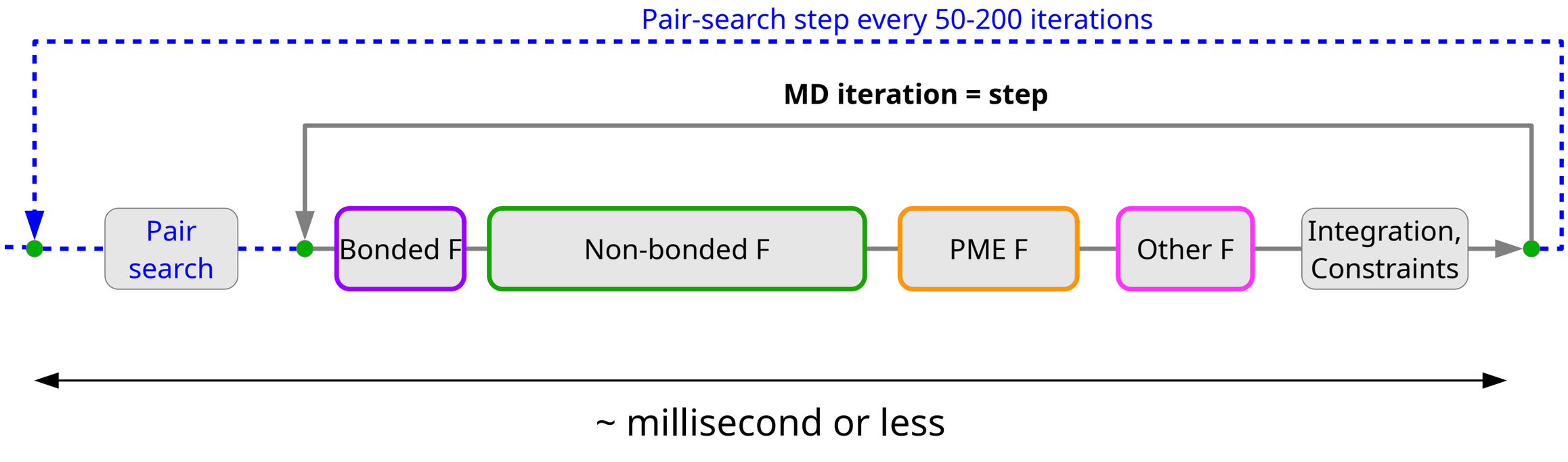


FLOPs in a typical simulation



Wall-time breakdown

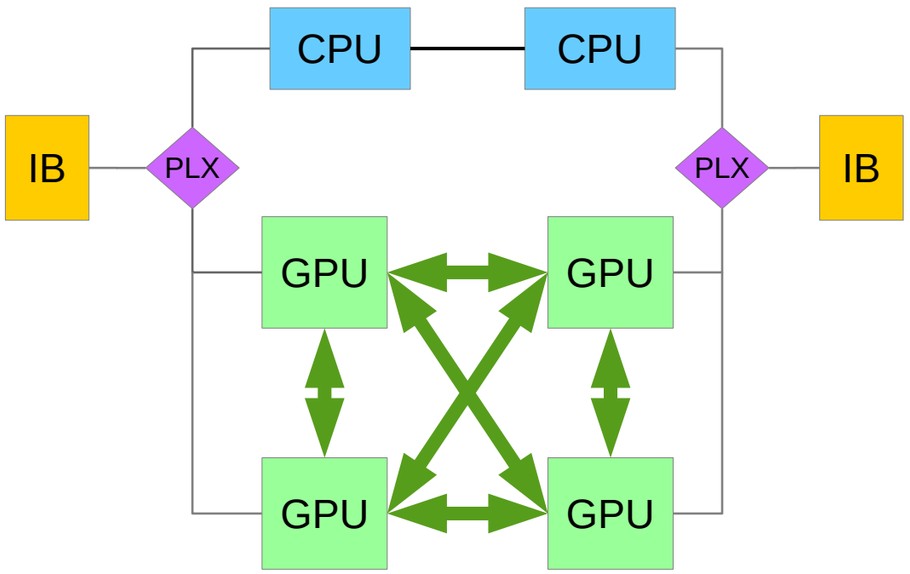
MD: computational challenge



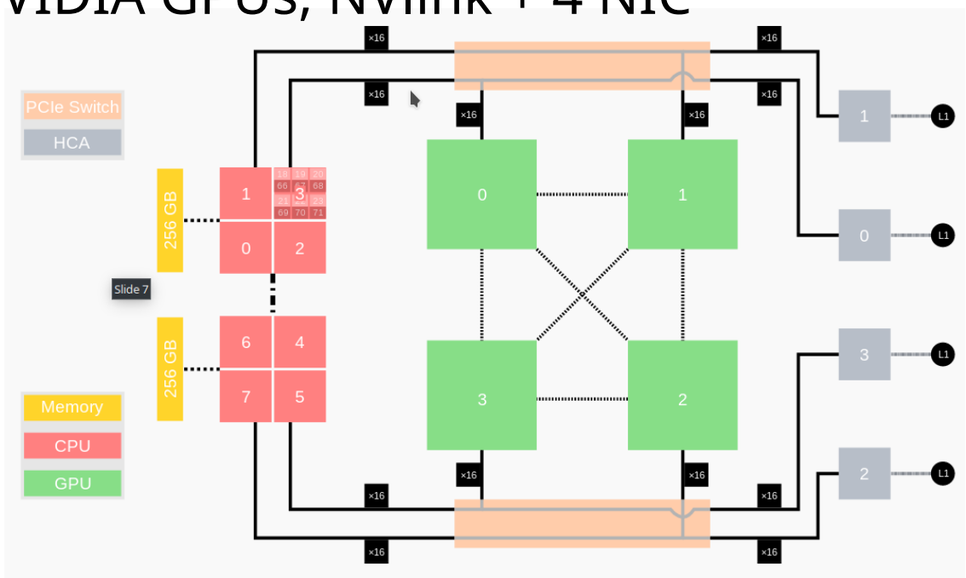
- Simluation vs real-world **time-scale gap**
  - Every simulation: 10^8 –10^{15} steps
  - Every step: 10^6 – 10^9 FLOPs
- (Often) need **strong scaling**
- MD codes at peak: ~**100 µs / stepc**
  - <100 atoms/core at peak
  - <10000 atoms / GPU

Heterogeneous HPC: changing landscape

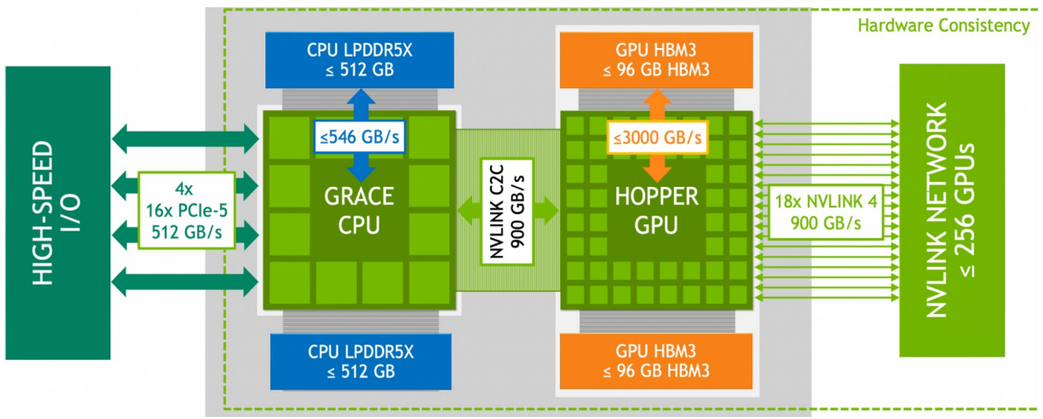
CSC Puhti: 2 Intel CPU + 4 NVIDIA GPU+ NVlink, 2 NIC



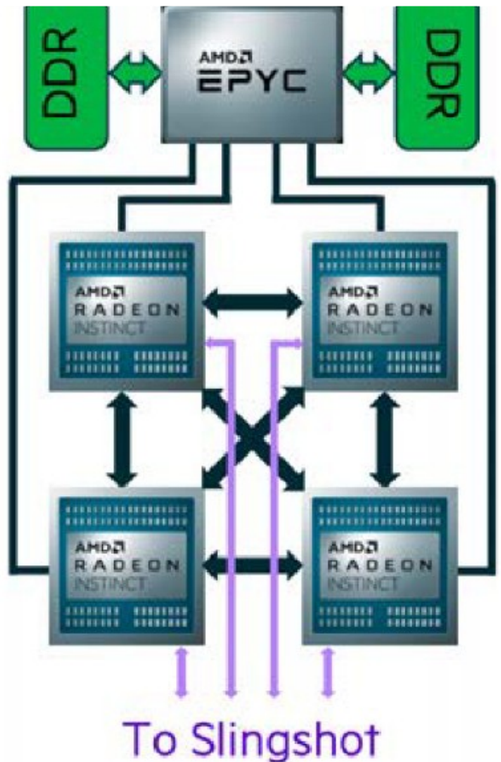
JUWELS-Booster: 2 AMD CPUs, 4 NVIDIA GPUs, NVlink + 4 NIC



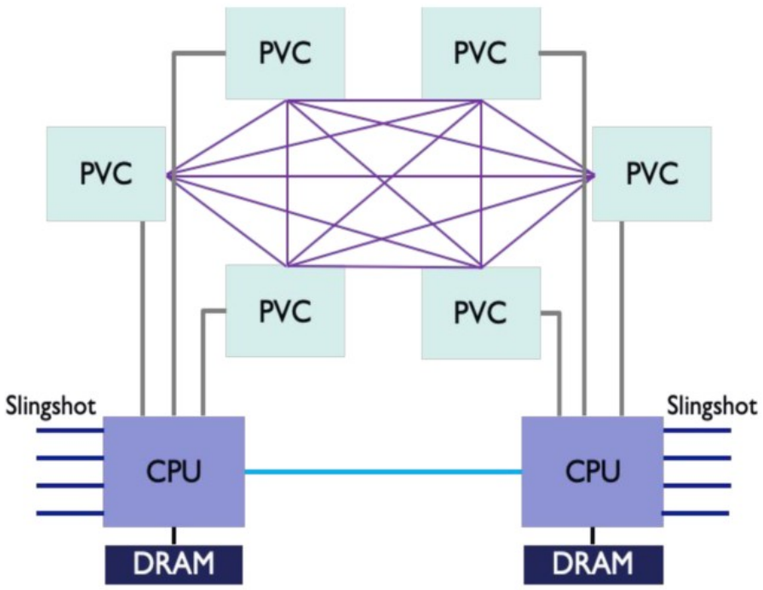
JSC Jupiter 4x NVIDIA Grace-Hopper + Nvlink + 4 NIC



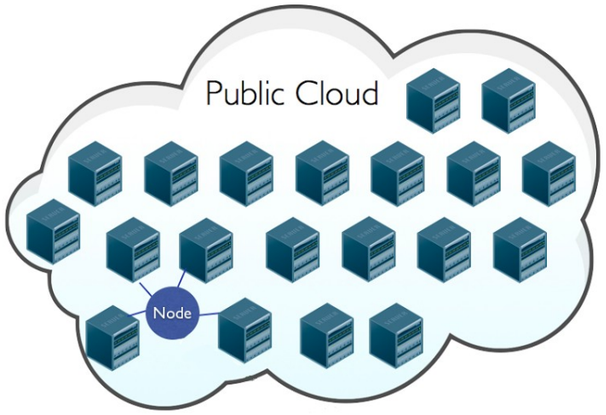
AMD CPU+GPU Exascale architecture: LUMI, Frontier



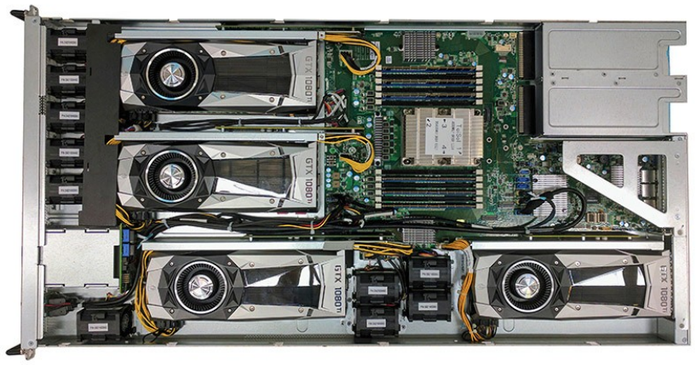
Intel CPU+GPU Exascale architecture: Aurora

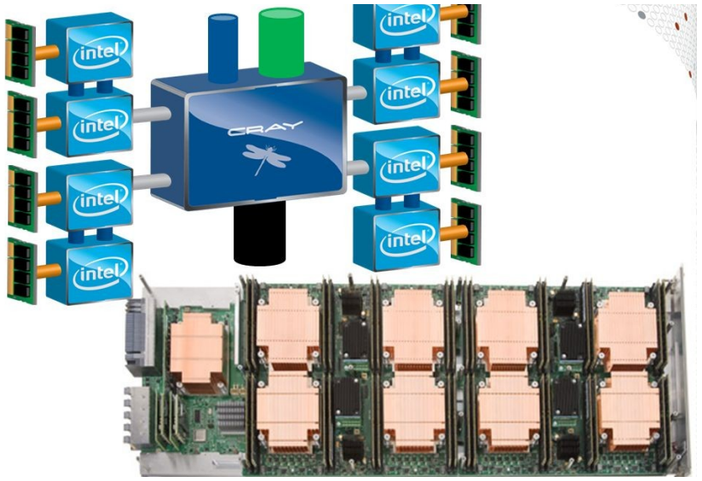


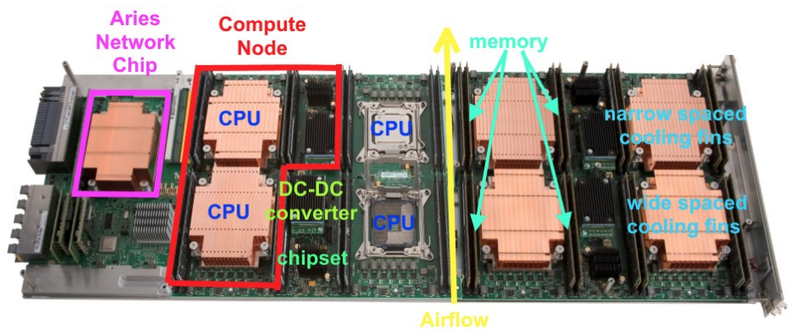
Multiple levels of hardware parallelism



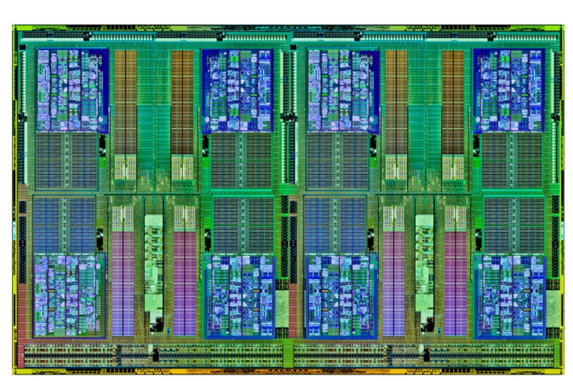
Compute cluster or cloud
Networked computers:
topology, bandwidth, latency

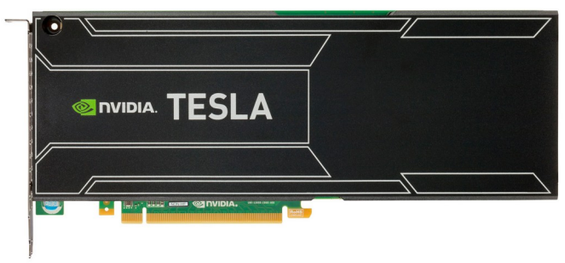






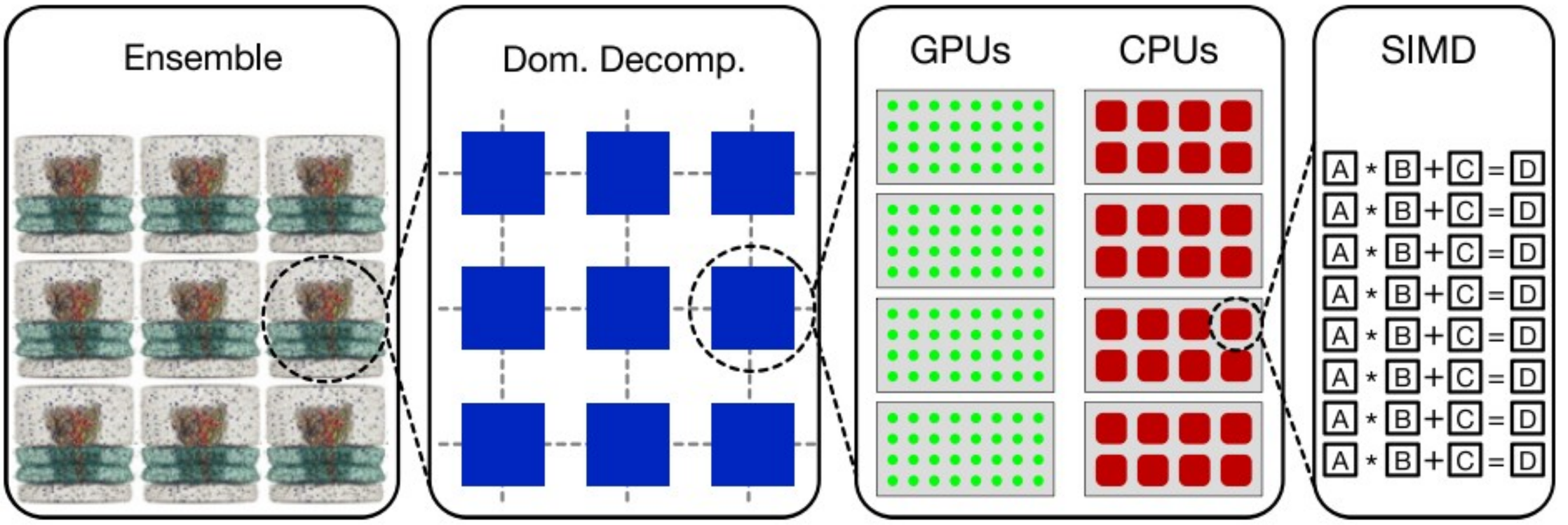
Compute node / workstation
NUMA topology, PCIe

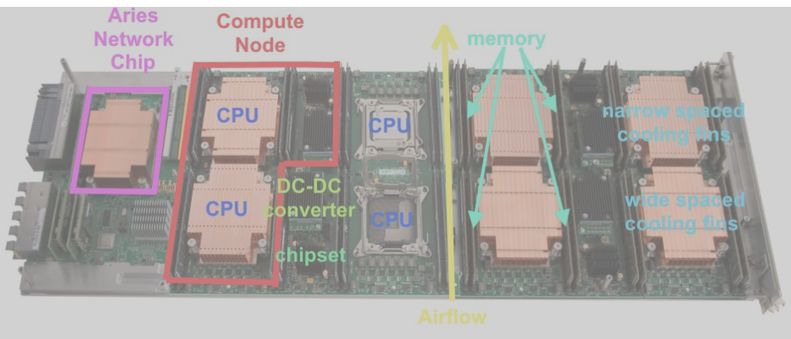


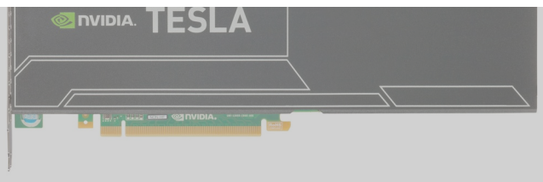


Multicore CPU & manycore GPU
caches, interconnects

Multiple levels of **hardware parallelism**
Multiple levels of **parallelization**





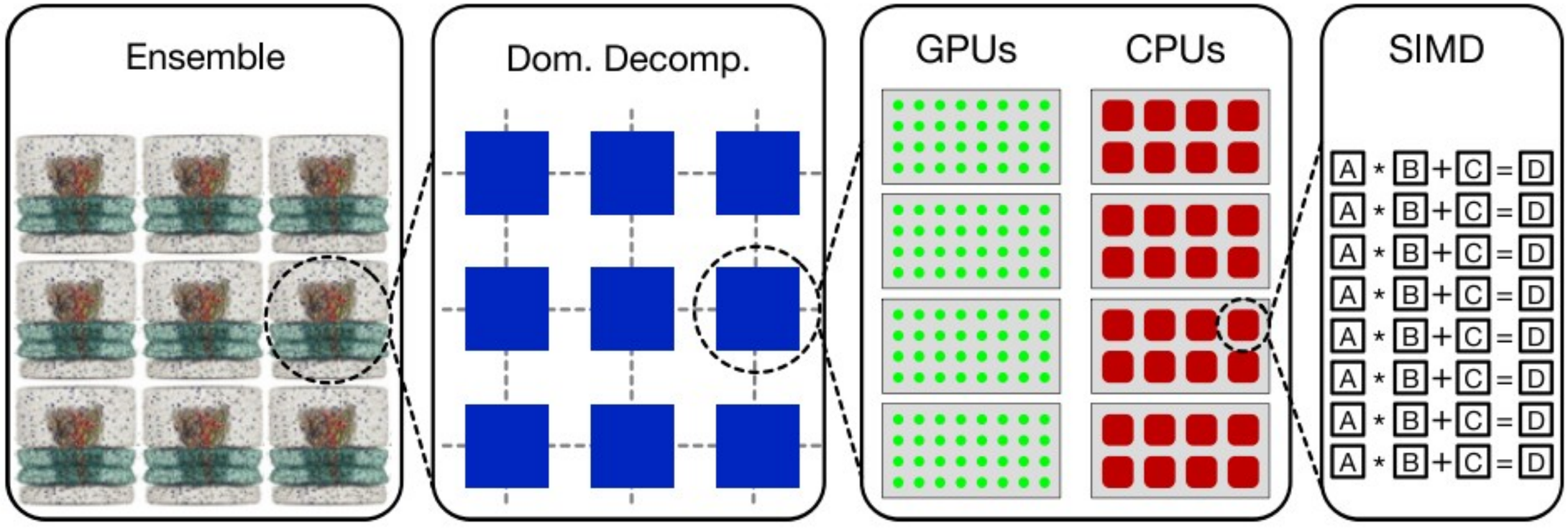


Compute cluster or cloud
Networked computers:
topology, bandwidth, latency

Compute node / workstation
NUMA topology, PCIe

Multicore CPU & manycore GPU
caches, interconnects

Multi-level parallelism



- The goal of parallelization:

  mapping the computational problem to the hardware

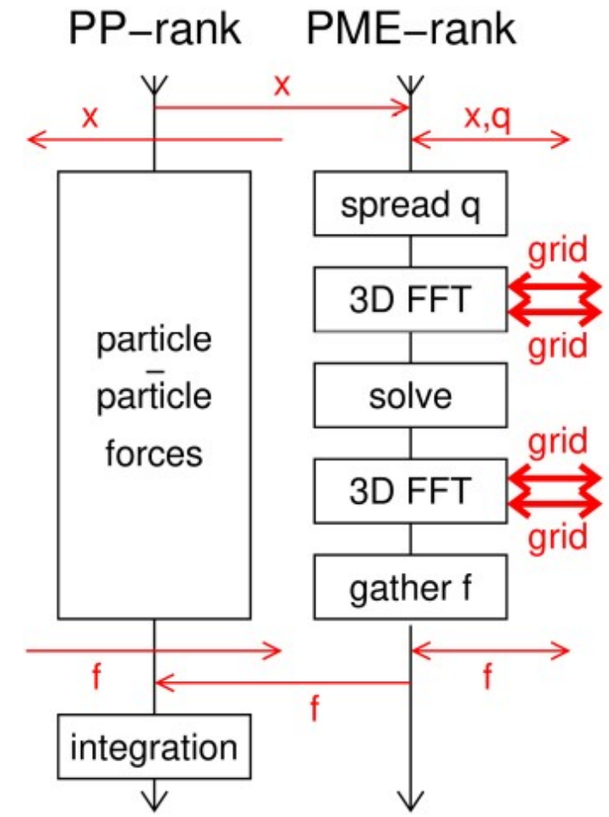
  - expose parallelism: algorithms
  - express parallelism: parallel implementation
- Need the right tool (programming language/model/API) for the problem!

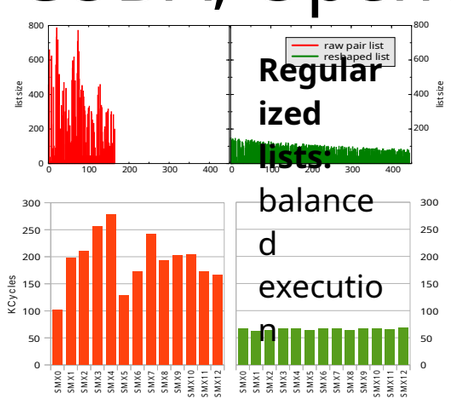
GROMACS parallelization overview

Multi-level parallelism:

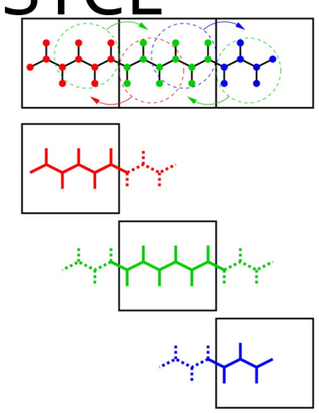
- SIMD / threading / NUMA / async offload / MPI

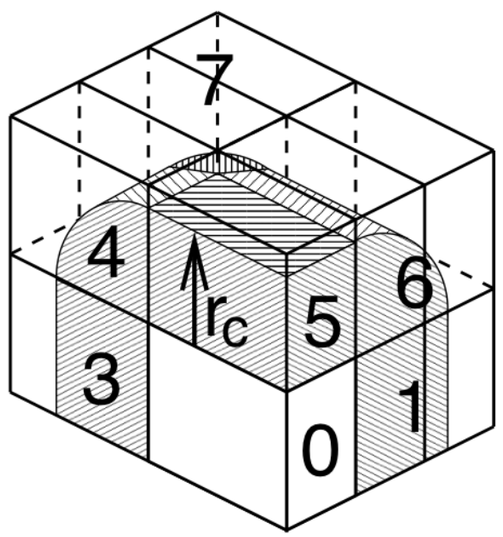
- **Hierarchical** parallelization**:** target **each level of hardware parallelism**
  - MPI: SPMD / MPMD; thread-MPI
  - OpenMP
  - SIMD: 14 flavors (SIMD library)
  - CUDA, OpenCL, SYCL

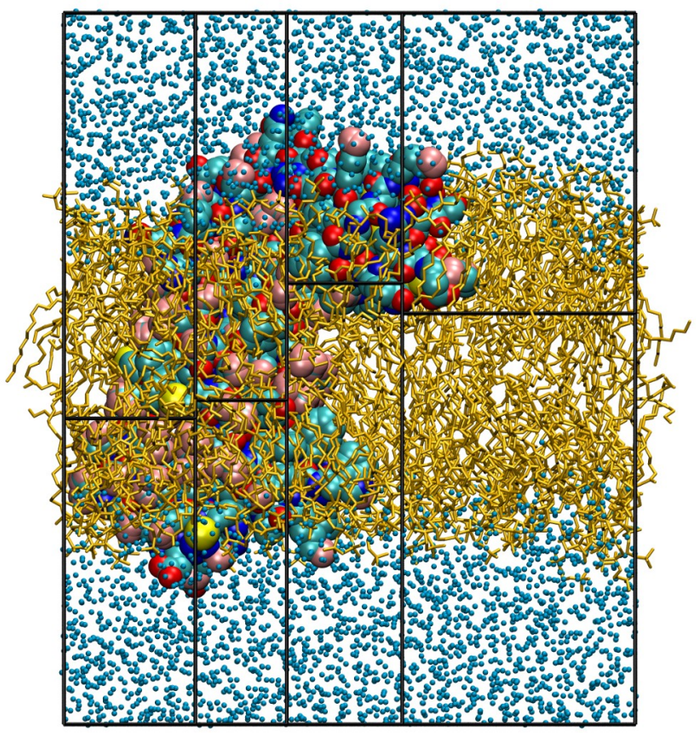




Regularized lists: balanced execution

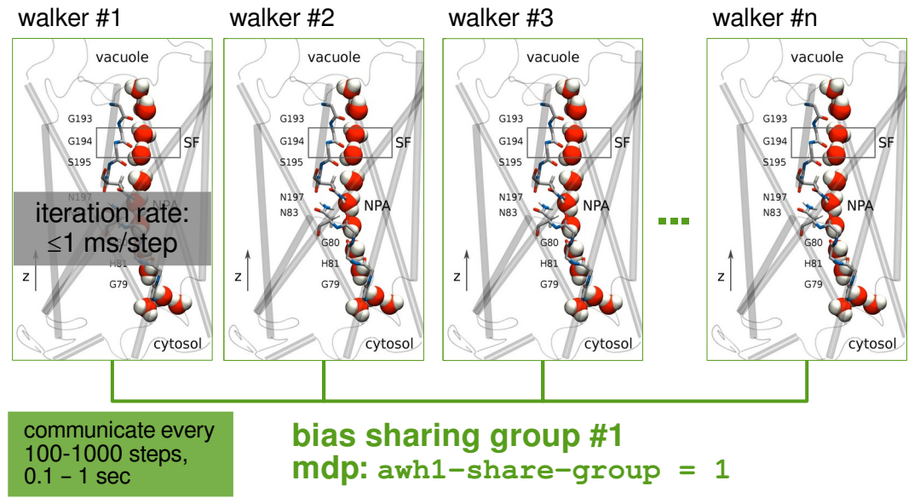


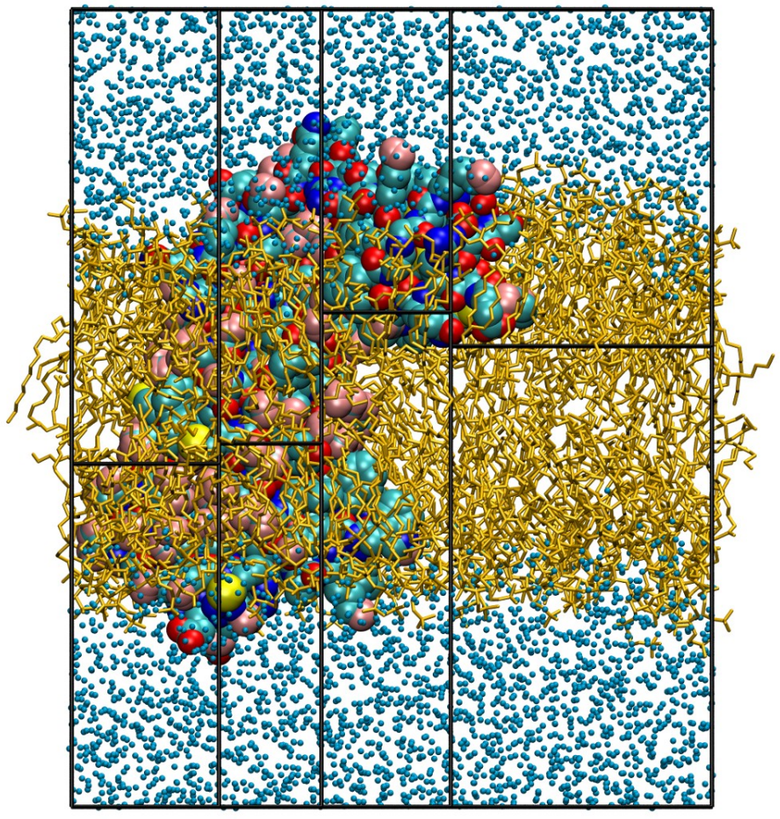




Decomposition approaches

- Problem decomposition approaches:
  - single-trajectory
  - multi-trajectory: ensemble / workflows





Performance & productivity

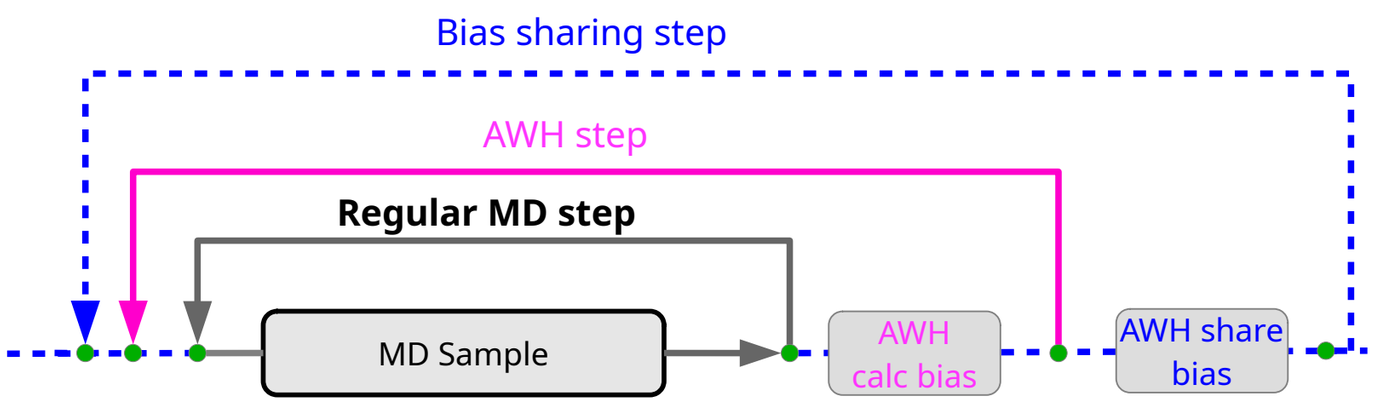
- Performance: rate of generating trajectories
  - ns/day, time/step
  - ensemble: aggregate

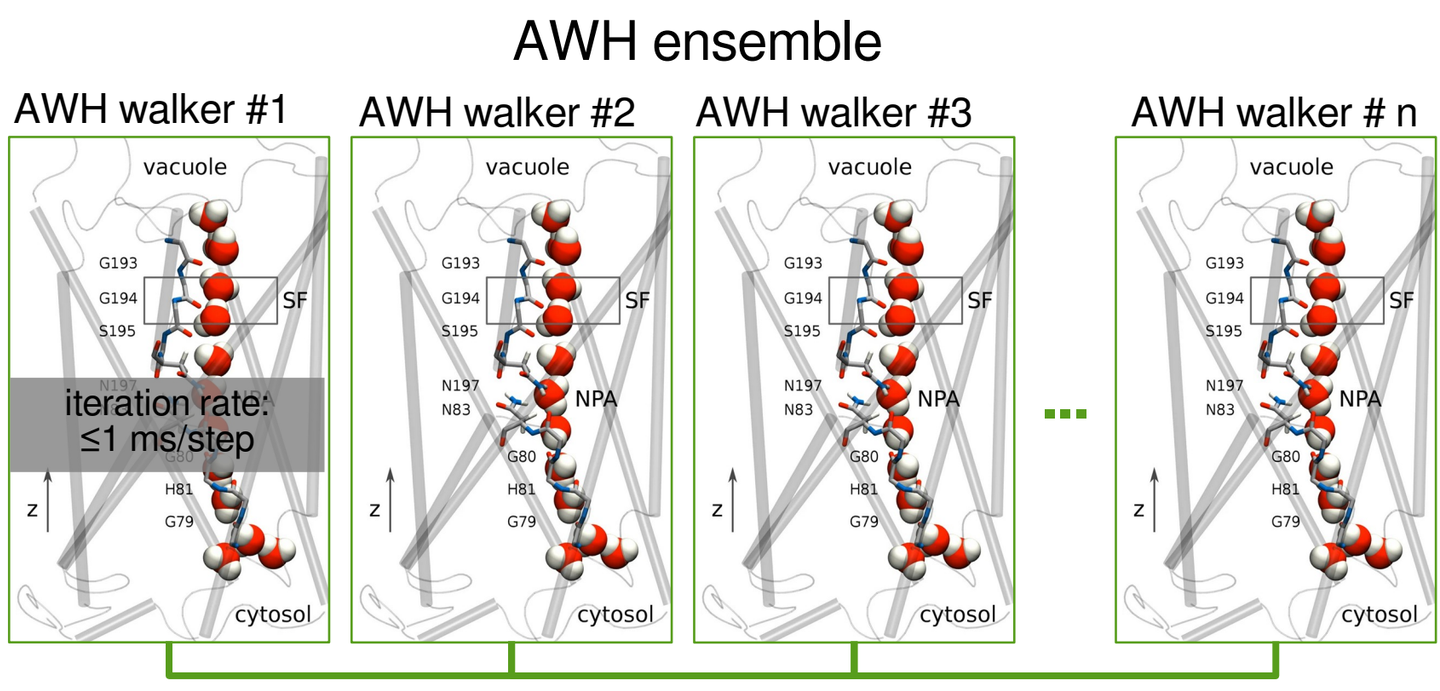
- Efficiency:
  - parallelization comes at a cost (strong vs weak scaling)
  - heterogeneous parallelization tradeoffs: load balance, utilization

- Productivity: time-to-solution
  - Single-trajectory:
    - requires making trajectory-generation as fast as possible
    - Efficiency tradeoff: parallel efficiency, hardware utilization
  - Ensemble:
    - Can balance efficiency with time-to-solution

Ensemble parallelization

- Ensemble algorithms:
  - multiple simulations/trajectories: **ensemble members**
  - express more independent work → more parallelism
- **Uncoupled ensemble**
  - no comm/data exchange during simulation (post-run e.g. file-system)
  - e.g. FEP, MSM
- **Coupled ensembles**
  - communicating **during** simulation (typically at a fixed step)
  - strongly/ weakly coupled: frequency of data exchange (10 vs 10^5 steps)
  - e.g. AWH, replica exchange, REST
  - coupling → performance sensitive



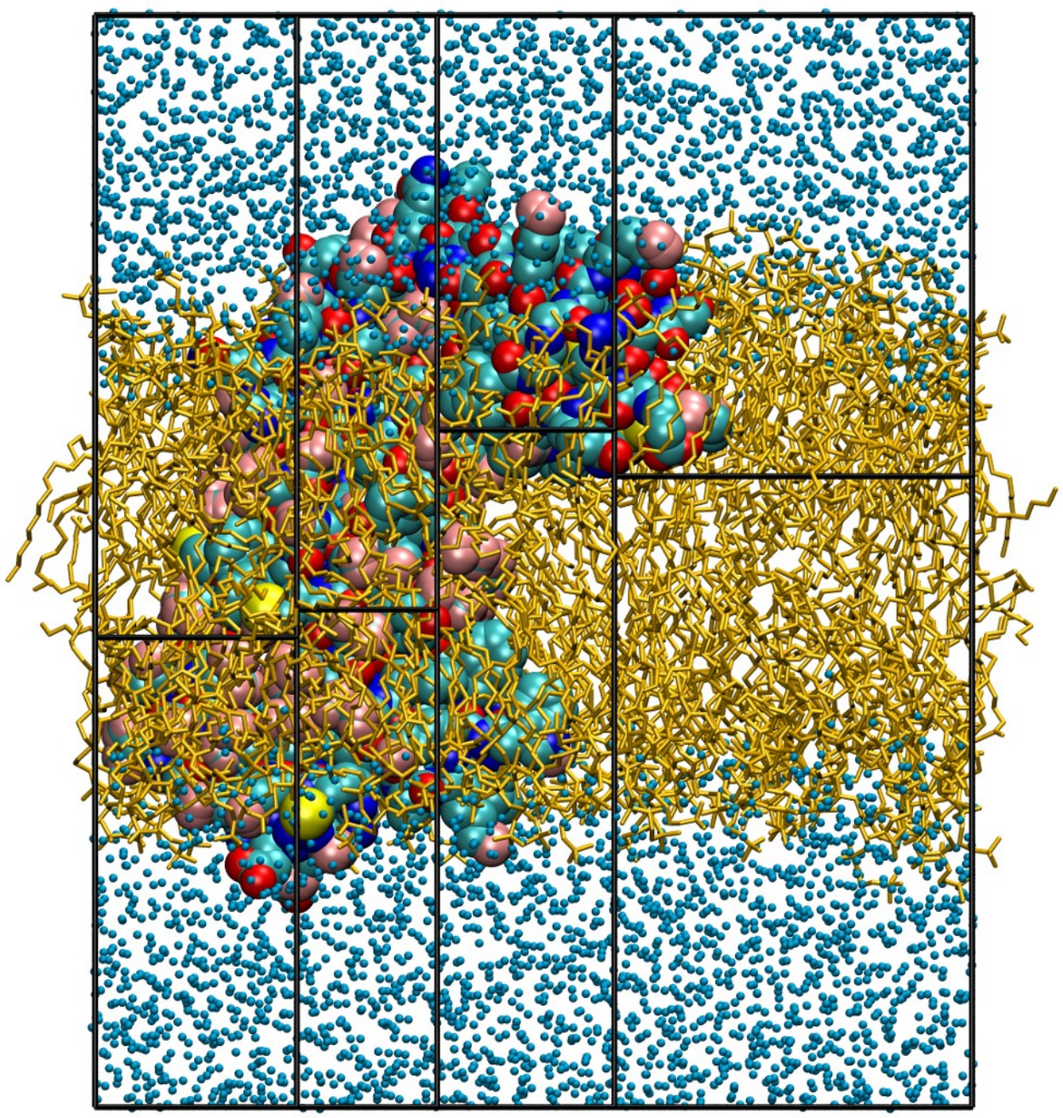


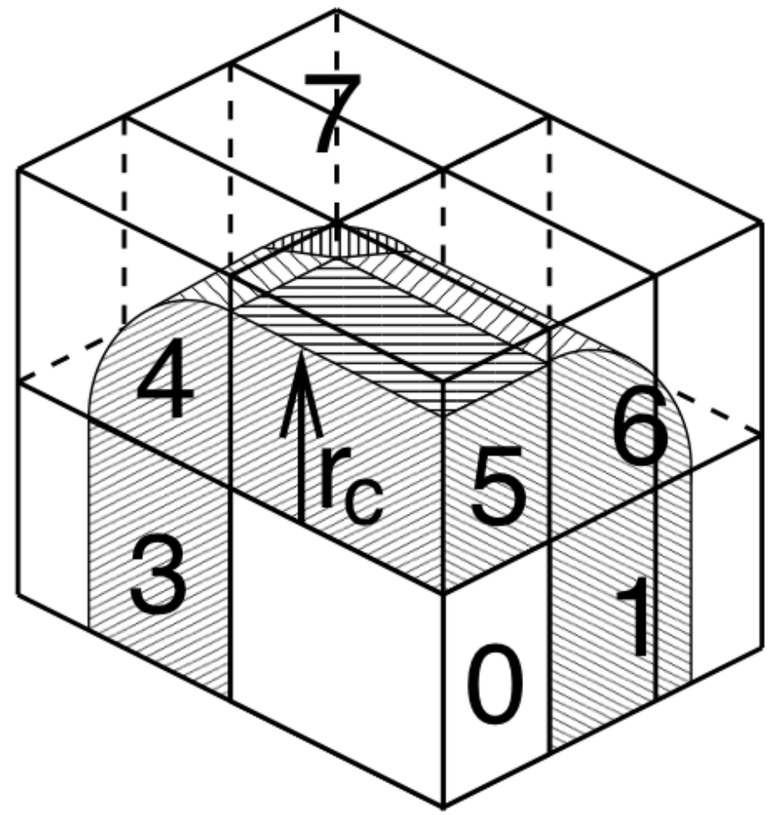
communication every 100-1000 steps, 0.1 – 1 sec

bias sharing across a multi-simulation

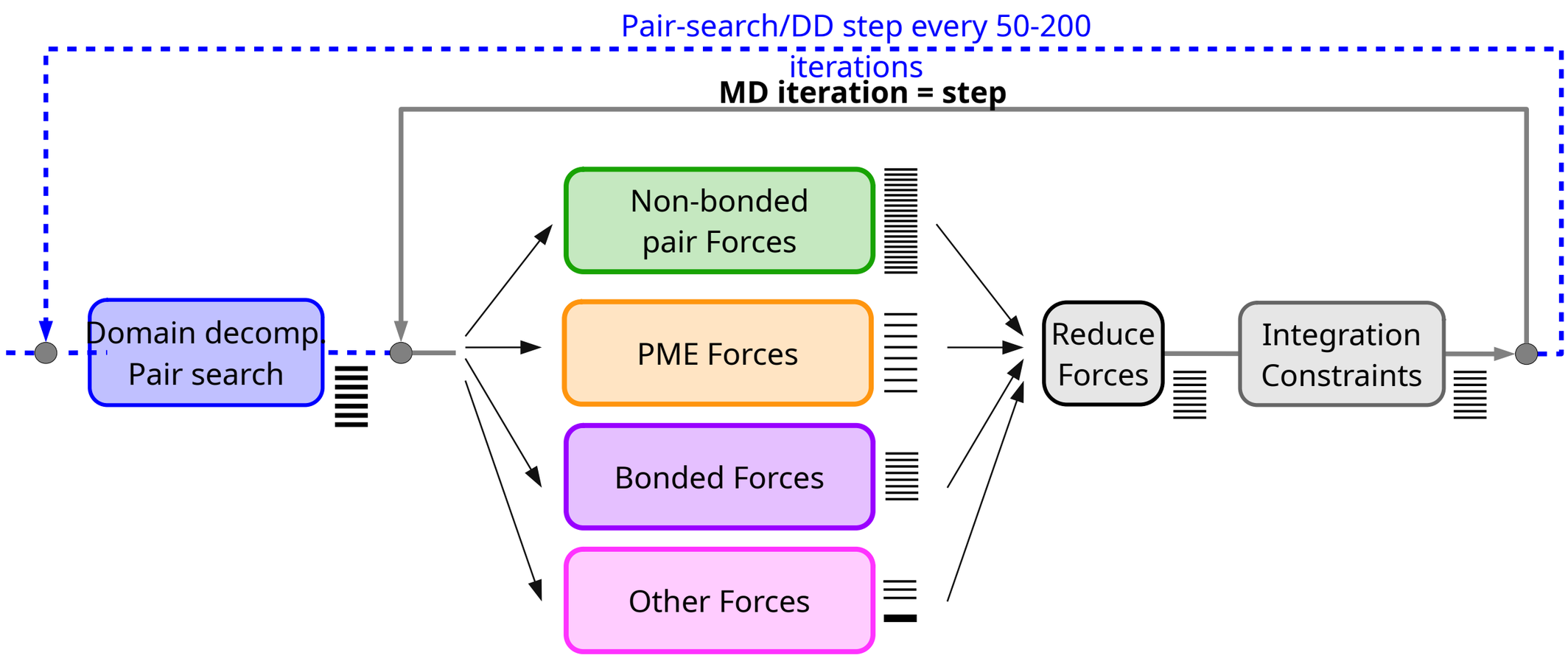
Domain-decomposition

- Neutral-territory domain-decomposition:
  - volume decomposition to create independent work
  - eighth-shell
  - triclinic unit cell with staggered cell boundaries supported



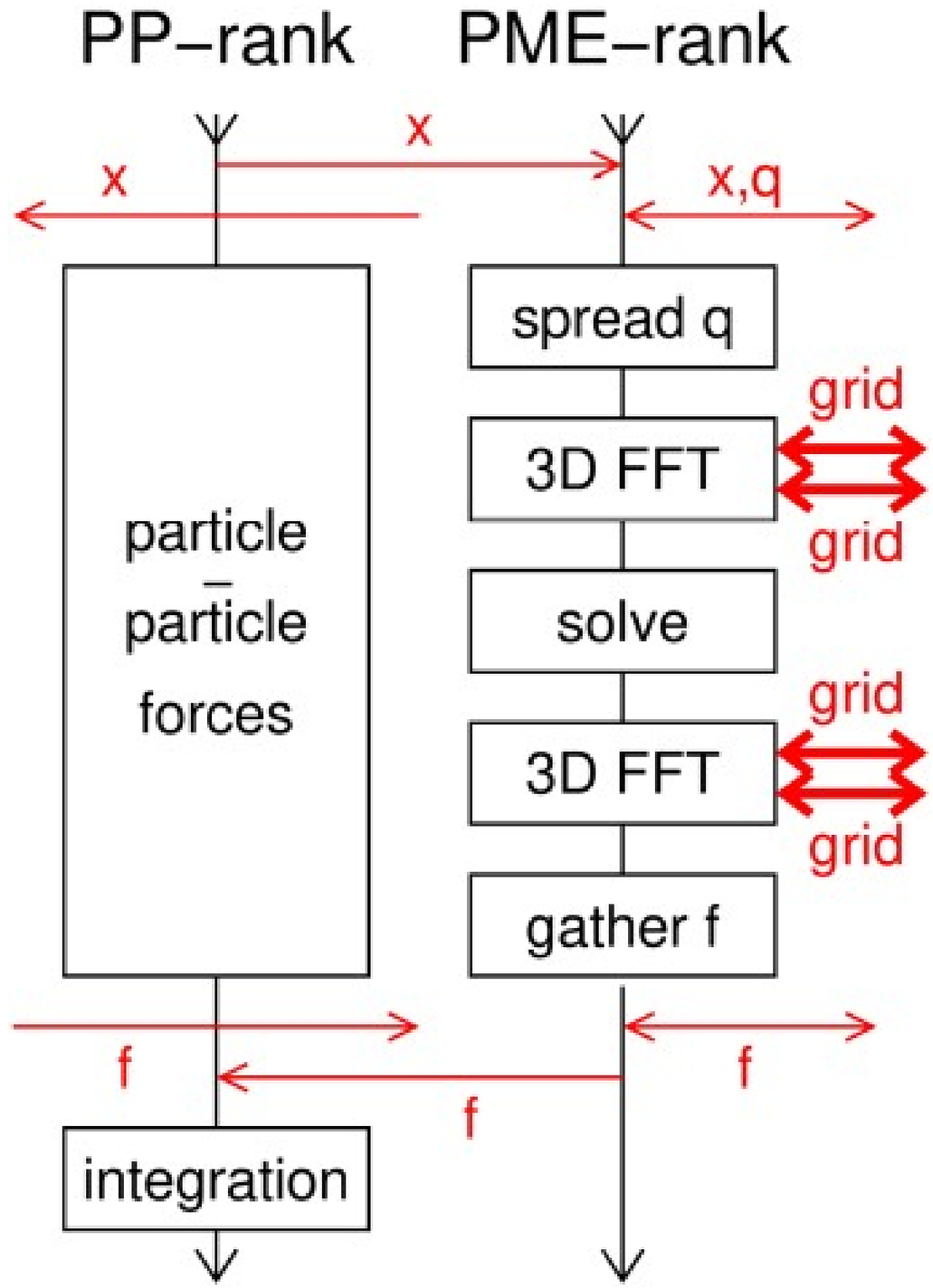


Concurrency within an the MD step



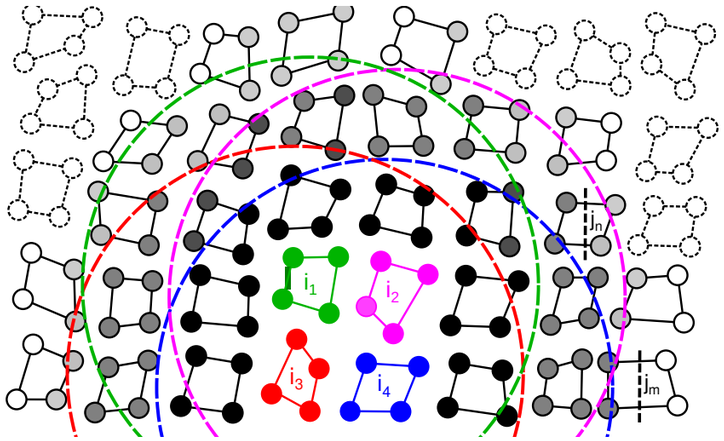
Separate PME ranks

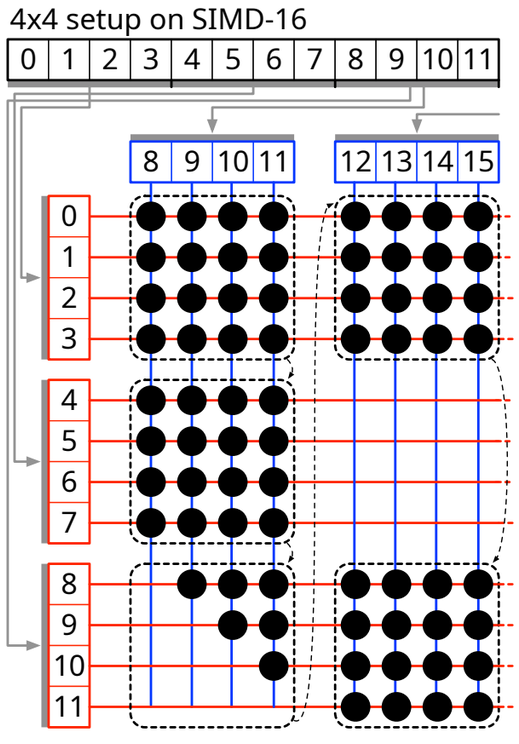
- Task decomposition to improve PME scaling
- MPMD: multiple program multiple data
- Dedicate a **subset of resources** to computing PME
  - fewer ranks => reduces FFT communication cost



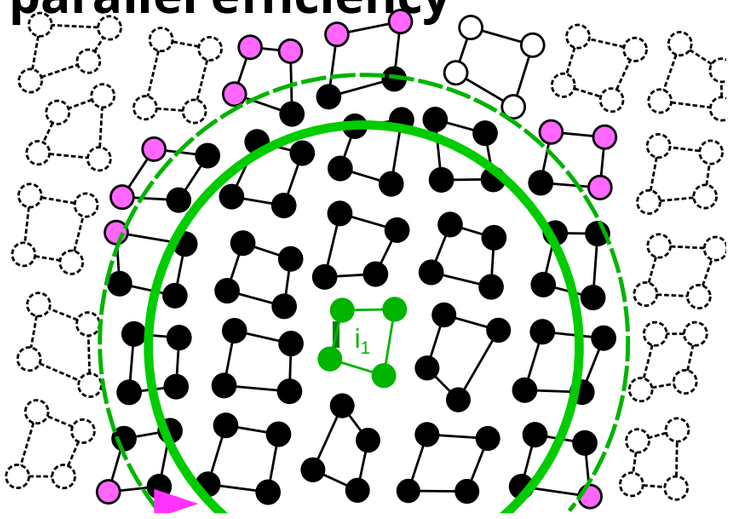
Fundamental MD algortihms redesigned for modern architectures

Cluster pair-interaction algorithm for SIMD/SIMT

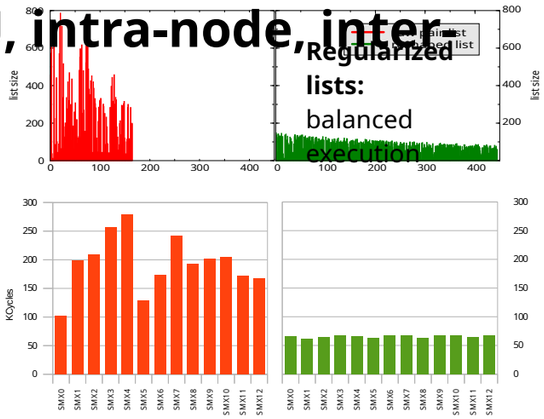


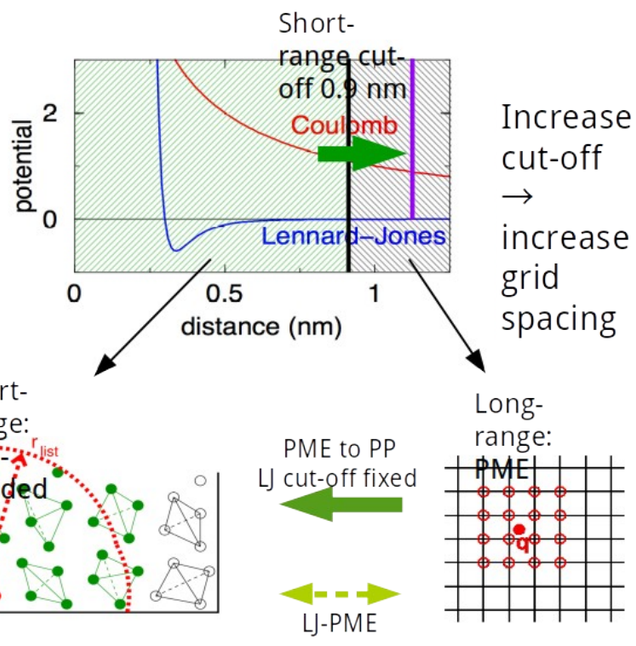


Accuracy-based automated list buffer improves SIMD algorithm parallel efficiency

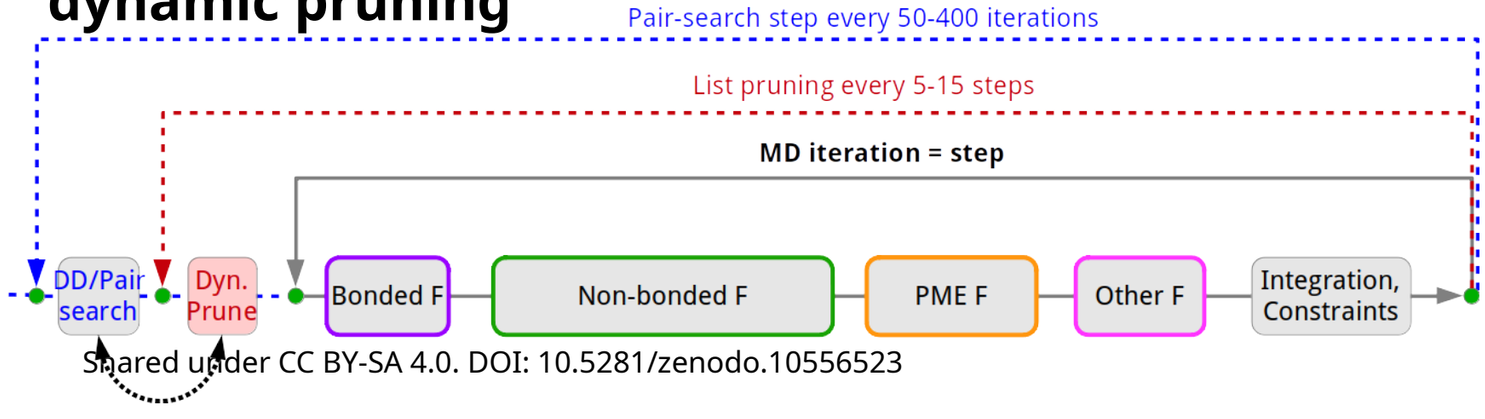


Multi-level heterogeneous data and task load-balancing: intra-GPU, intra-node, inter-node

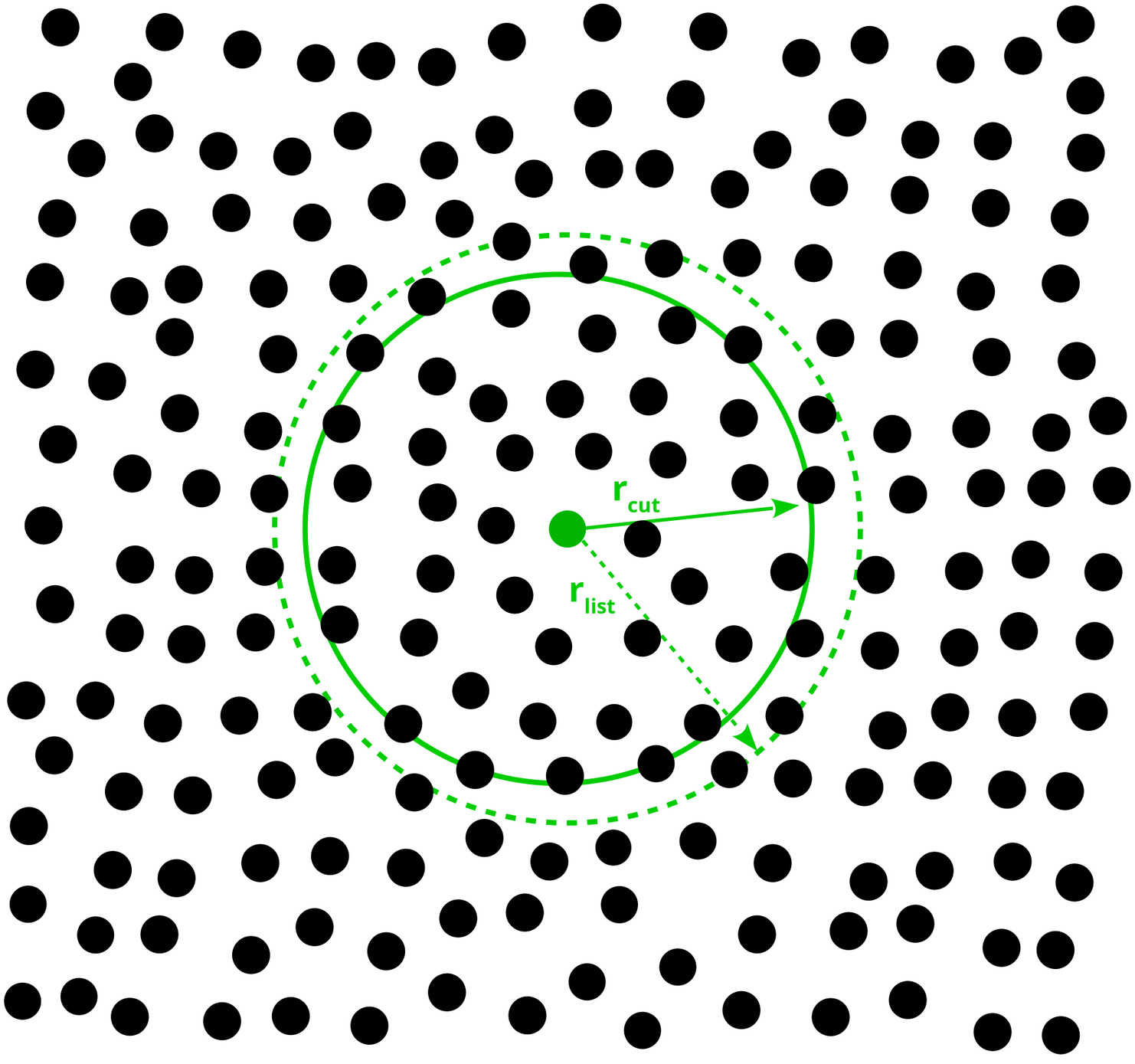




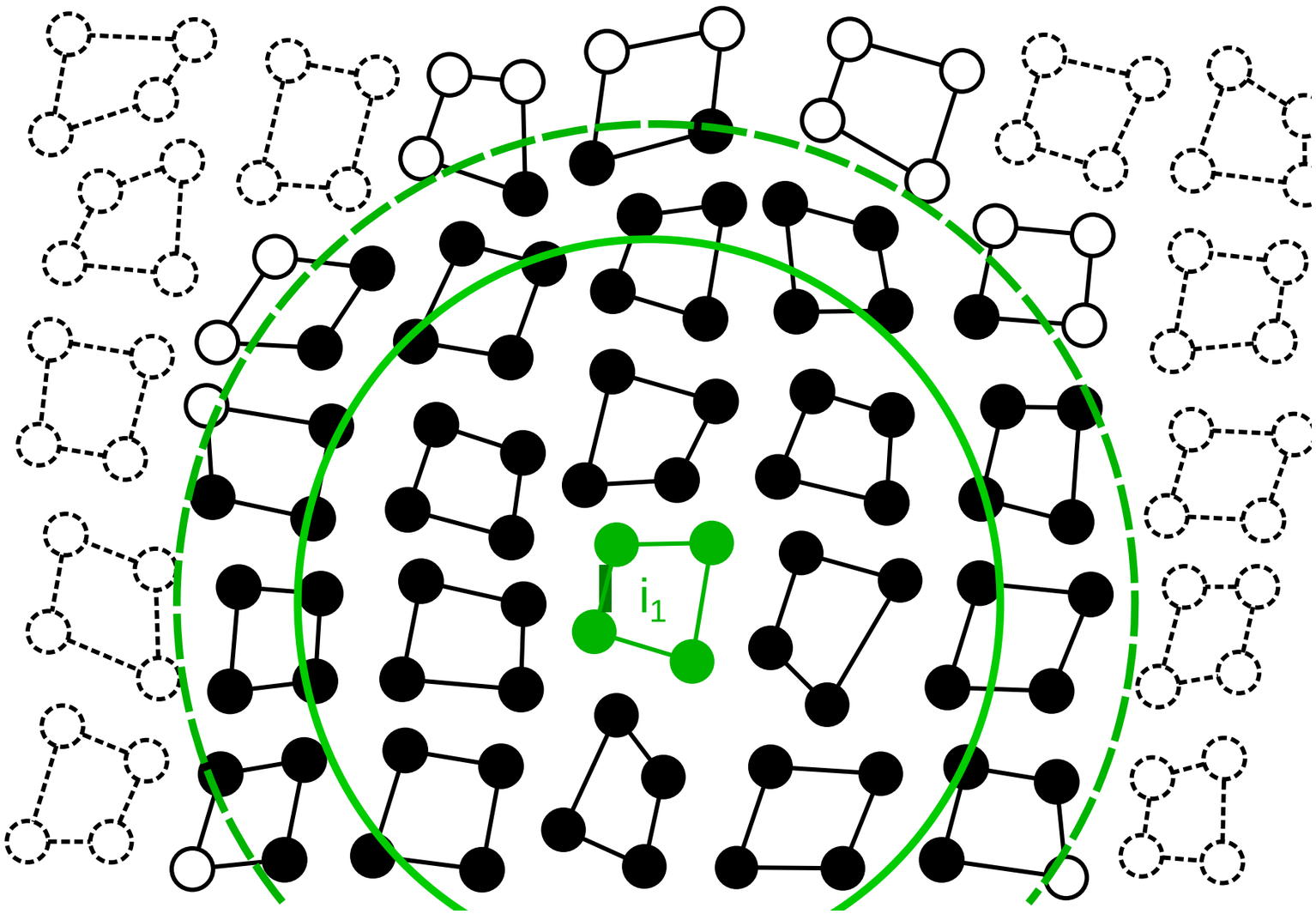
Dual pair list with dynamic pruning



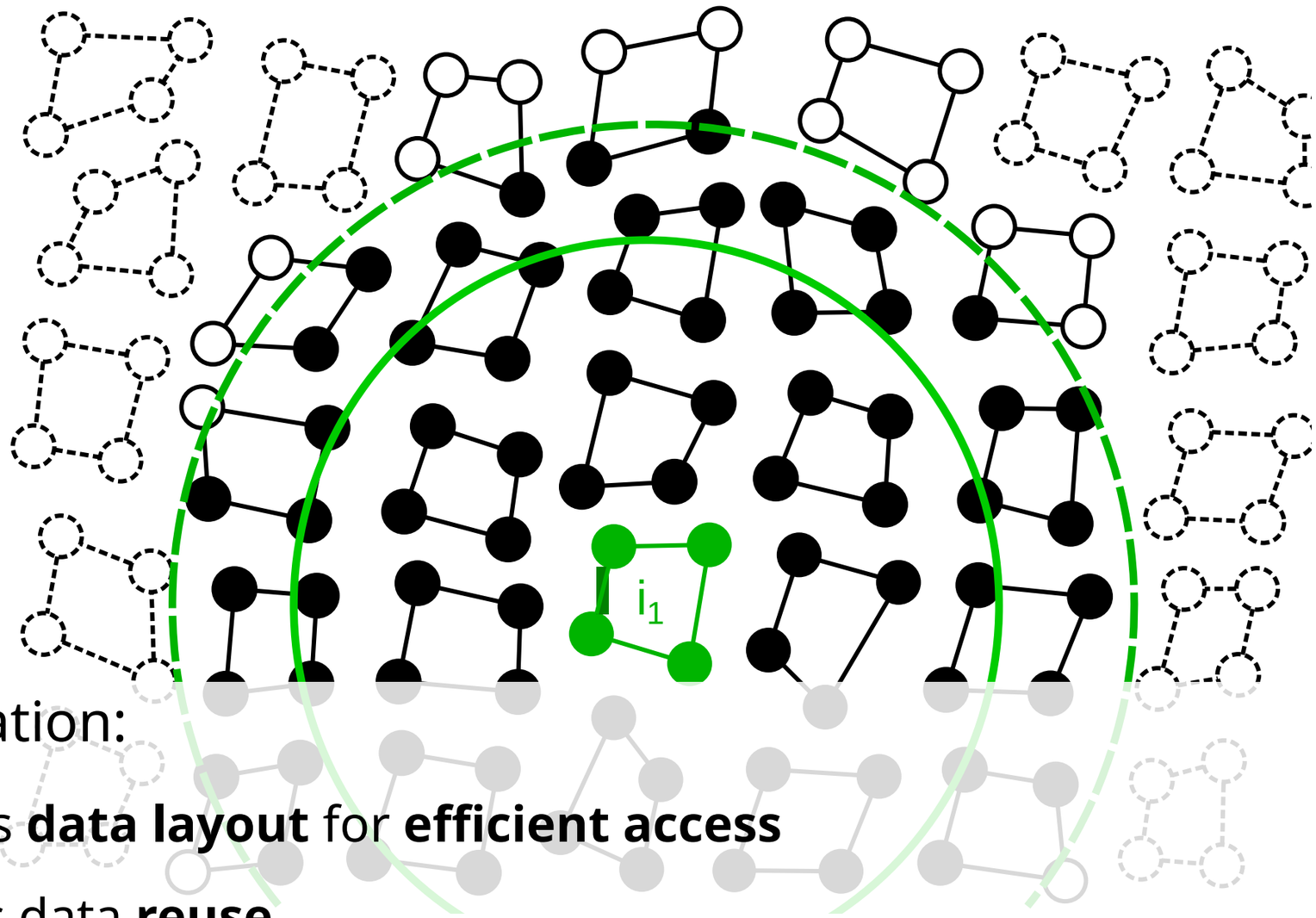
Pair interactions: Verlet algorithm



Cluster algorithm



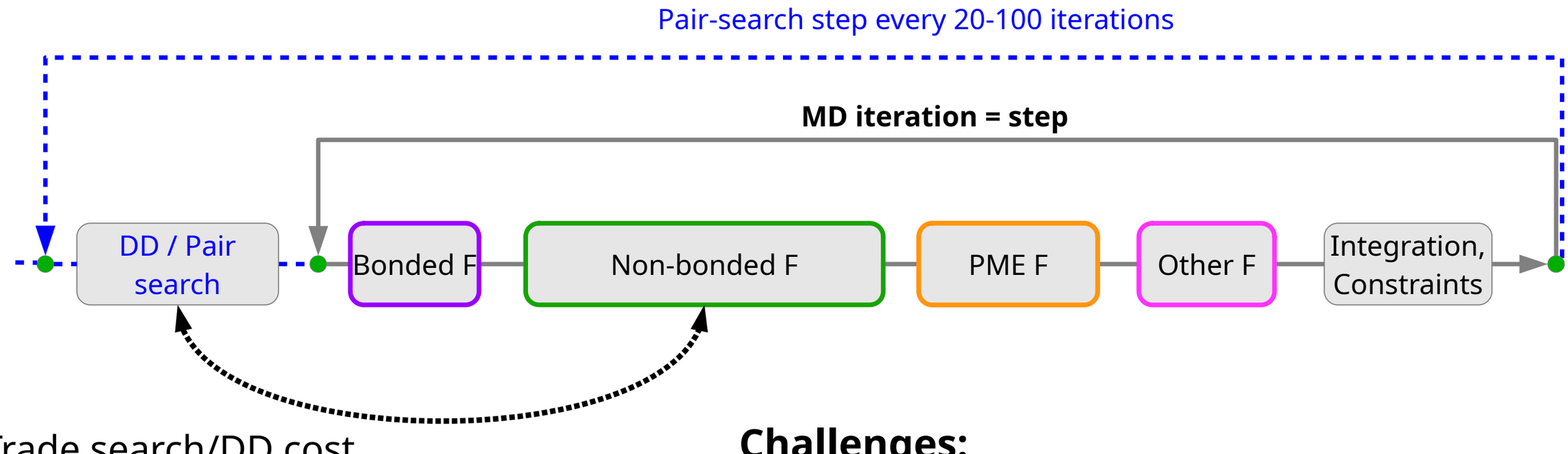
Cluster algorithm



- Regularization:
  - optimizes **data layout** for **efficient access**
  - increases data **reuse**
- Flexible & adaptable:

  SIMD width & reuse factor algorithmic parameters

Accuracy-based cost balancing: reducing decomposition & search cost



Trade search/DD cost
→
pair interaction cost

Challenges:
- Buffer size can increase quickly: trade-off in non-bonded computation
- On CPUs there is less freedom

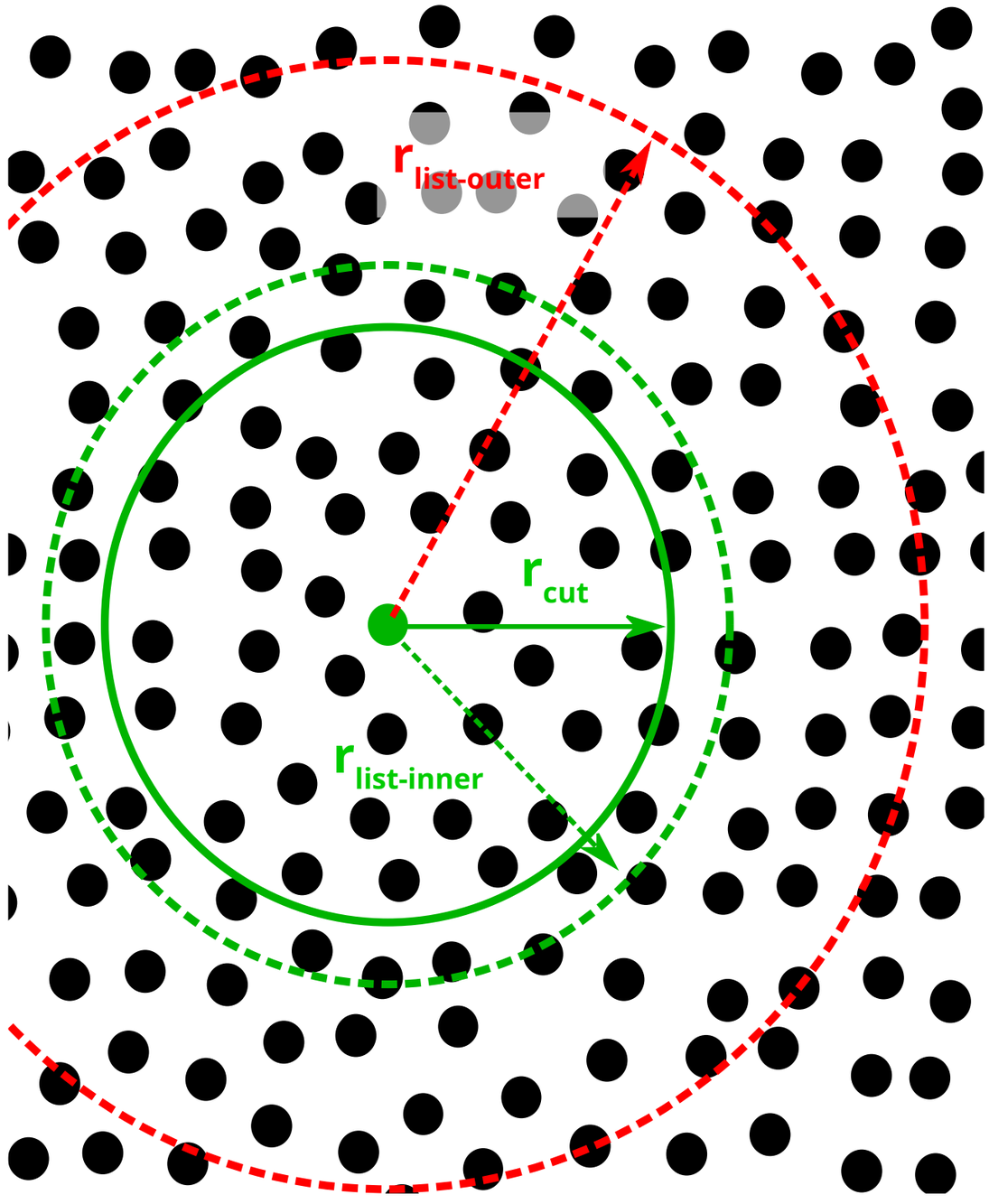
- Pair search frequency no longer based on a **rule of thumb!**
  - free parameter
  - `nstlist` (mdp parameter) can be tuned

Dual pair list

- Trading costly data regularization for force computation not ideal!
- Instead: keep regularized particle data longer, **shift the cost tradeoff**
- Use two buffers and lists:

  outer / inner
- Periodically re-prune

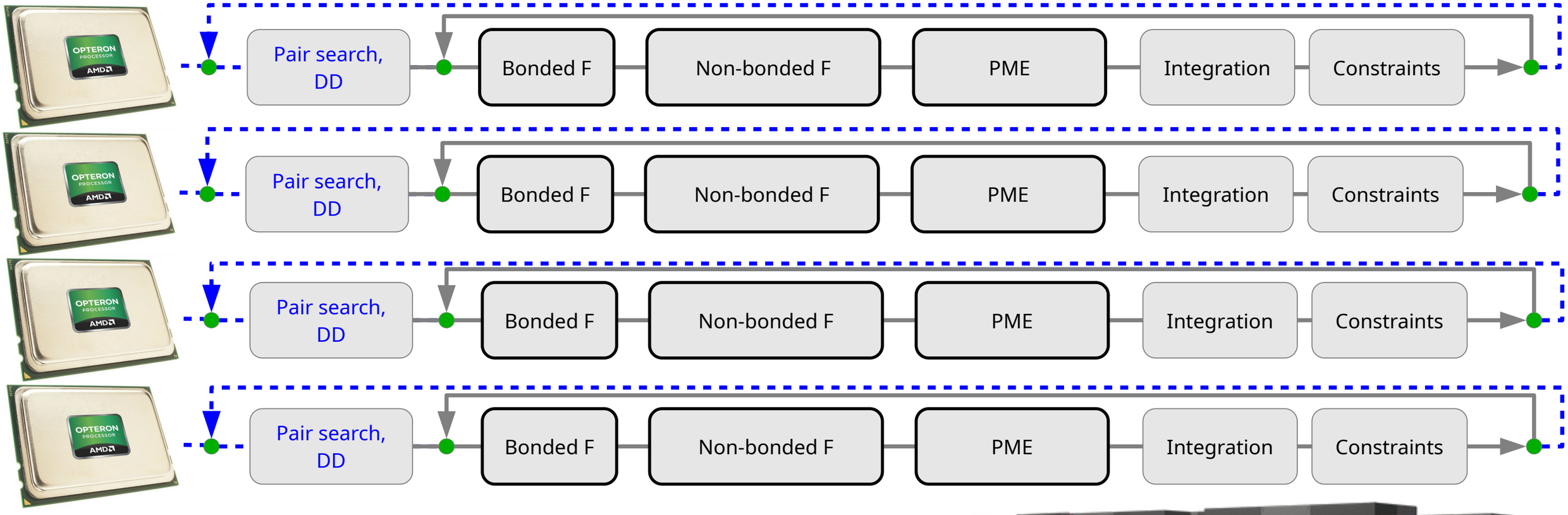
  outer → inner
- List lifetime / search frequency:
  - outer list less frequently (costly)
  - inner list more frequently (cheap)



Homogeneous vs heterogeneous parallelization

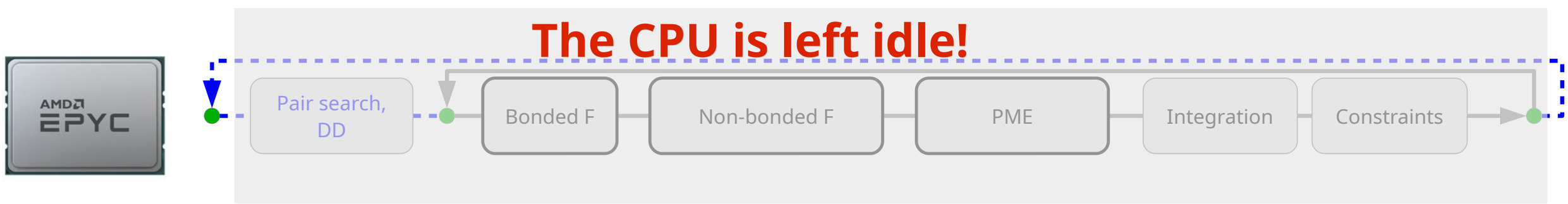
- **Homogeneous**: use a single type of compute unit

  e.g. a CPU cluster

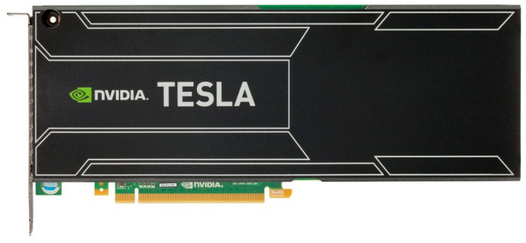


Homogeneous & heterogeneous parallelization

- **Homogeneous**: use a single type of compute unit
  - use only the GPU for computation, ignore the CPU



Offload everything



GROMACS heterogeneous engine: flexibility, extensibility, performance

- **Maintain the versatility** of the GROMACS software
  - (nearly) all features can be supported
  - “full port” to multiple toolkits/APIs not an option for a large codebase (& small team)
  - extensibility: implement new methods on CPUs only witout giving up GPU acceleration

- **Performance**
  - use CPU & GPU for the tasks they are best at
  - flexibility for performance: adapt to CPU/GPU hw balance

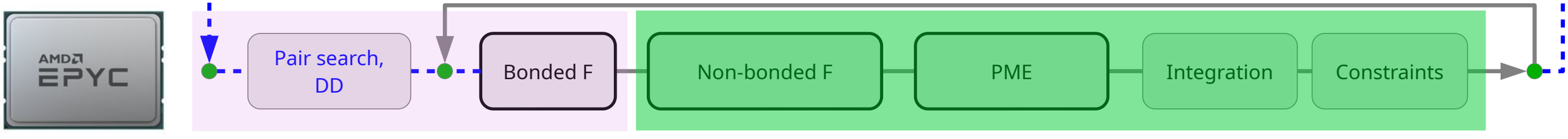
- **Portability and hardware support**
  - based on GPU abstraction layer with multiple backends: CUDA, OpenCL, SYCL
  - NVIDIA, AMD, Intel hardware support

- **Challenges**:
  - flexibility vs complexity
  - fast CPU code often worth using vs data locality
  - load balancing

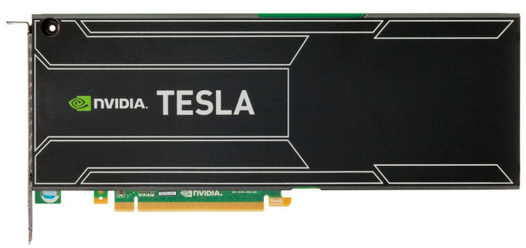
Homogeneous & **heterogeneous** parallelization

- **Heterogeneous**: use multiple types of compute units

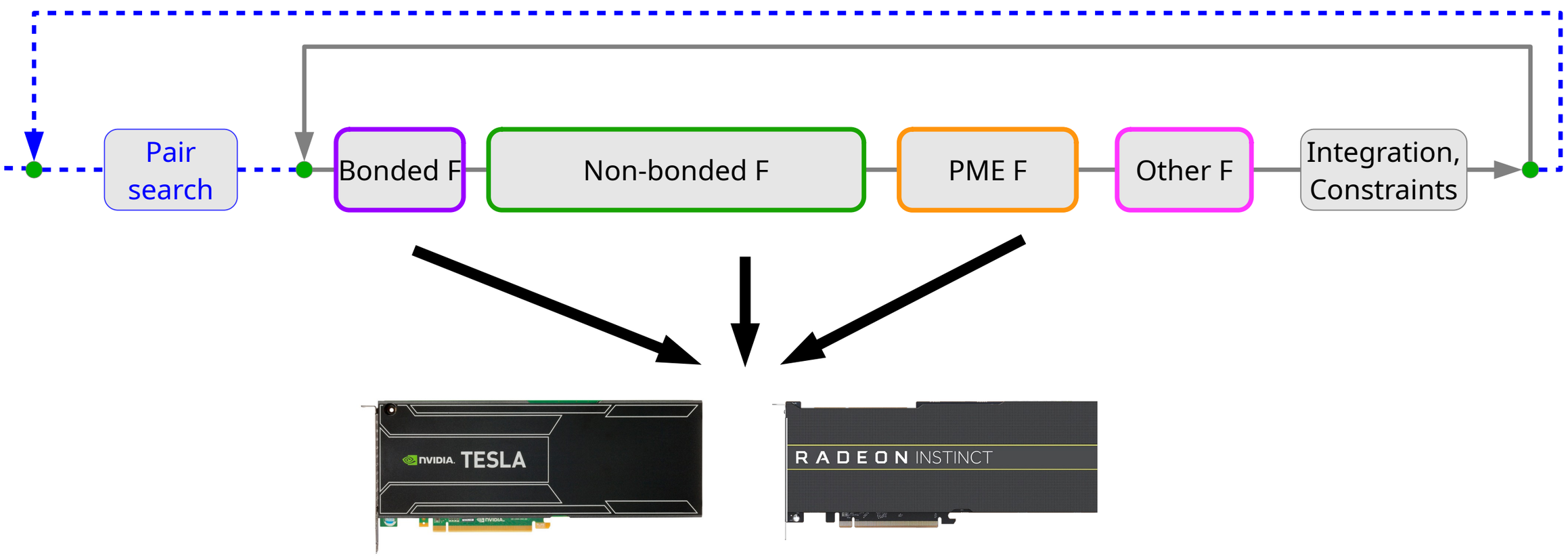
The CPU is used for some computatiton!



Offload some of the computation



Force offload



GPU:

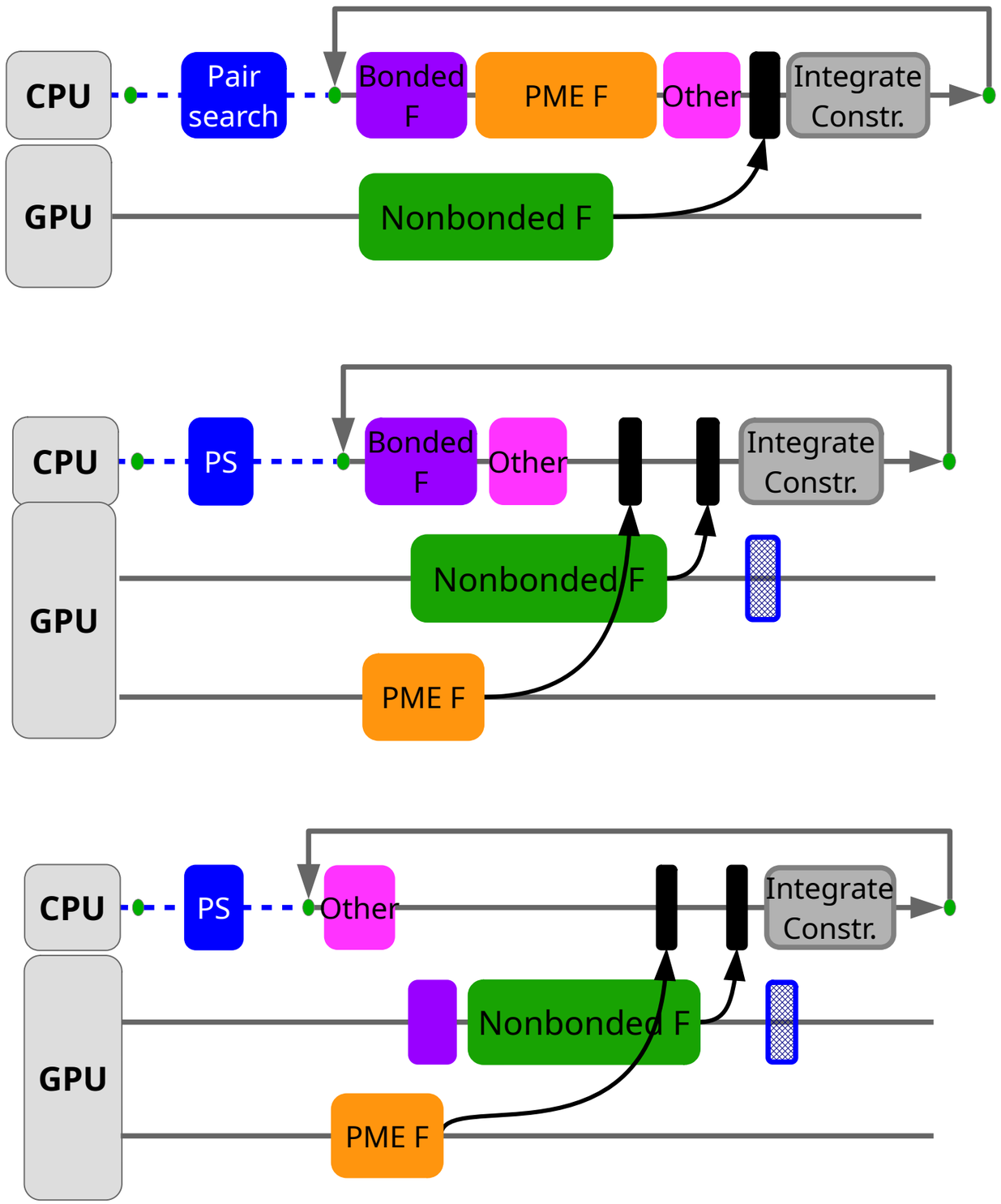
- offload force computation which benefits most from GPU acceleration:

  bonded, non-bonded pair, PME

CPU:

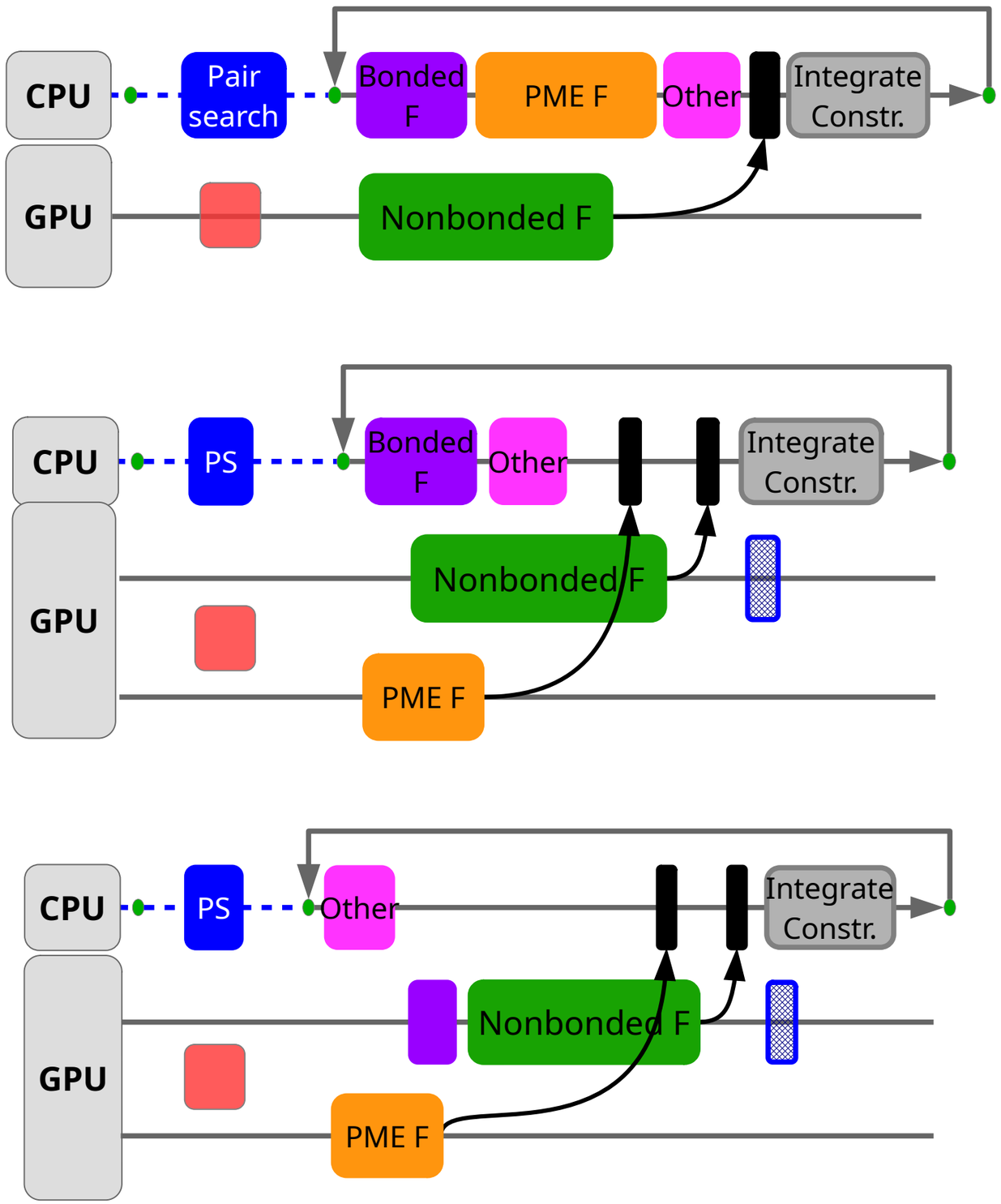
- pair search / domain decomp
- other F / special algorithms
- integration

Force offload schemes



- Offloading different force components allows adjusting to hardware balance

Force offload schemes



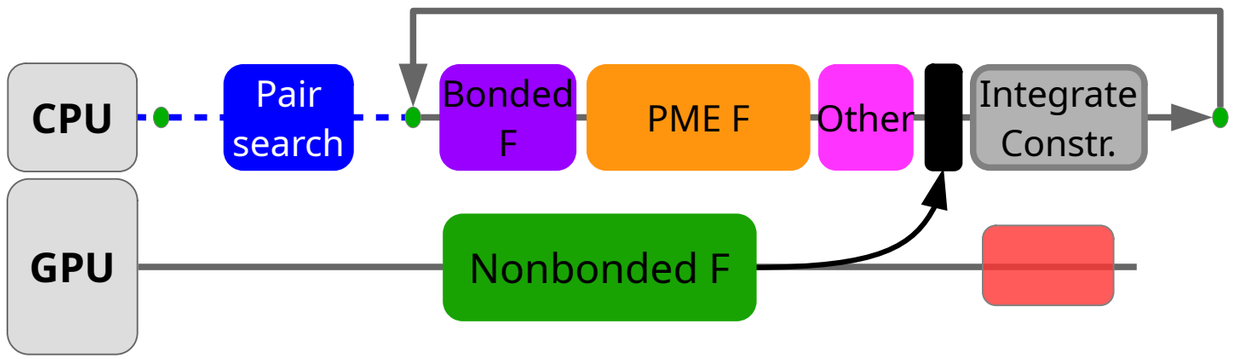
- Offloading different force components allows adjusting to hardware balance

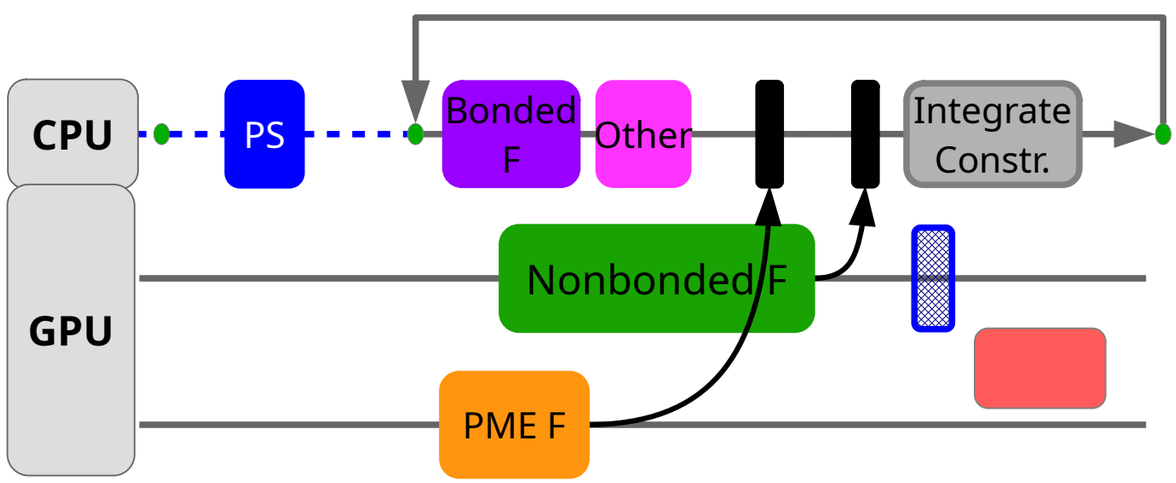
- Search / DD: complex code, kept on the CPU

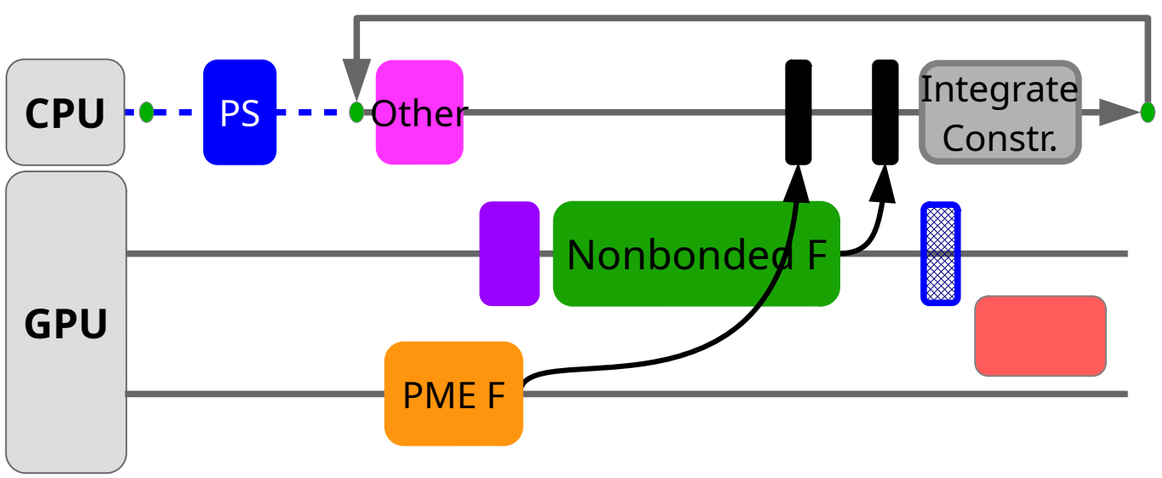
  → algorithm and code optimization to improve CPU—GPU overlap

  (dual pair-list)

Force offload schemes







Integration on the CPU

==>

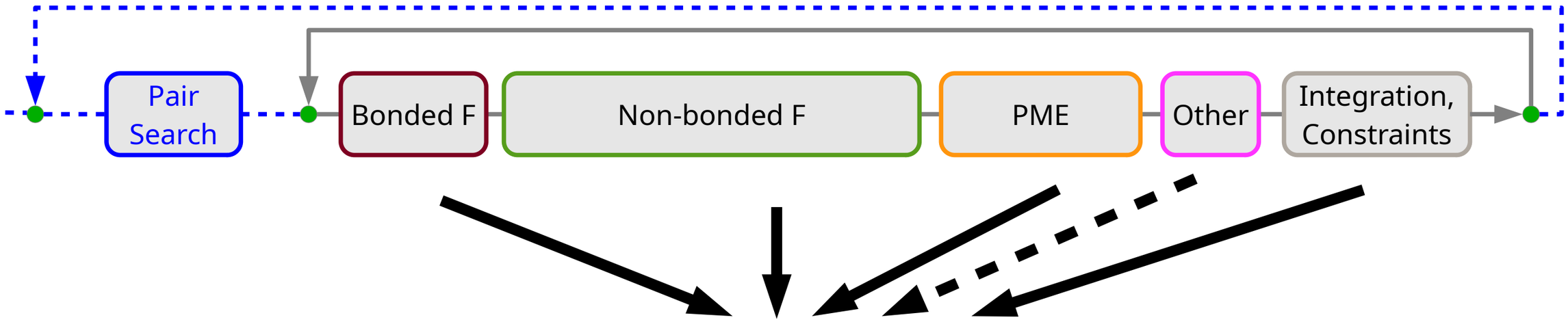
per-step CPU – GPU data copy needed

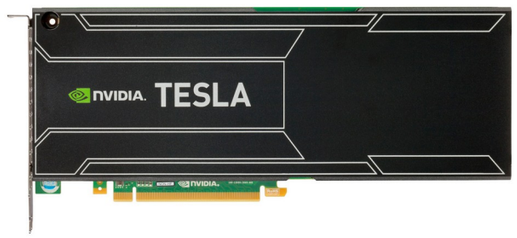
Amdahl's law!

as GPUs get faster,

fraction of time spent in CPU integration increases

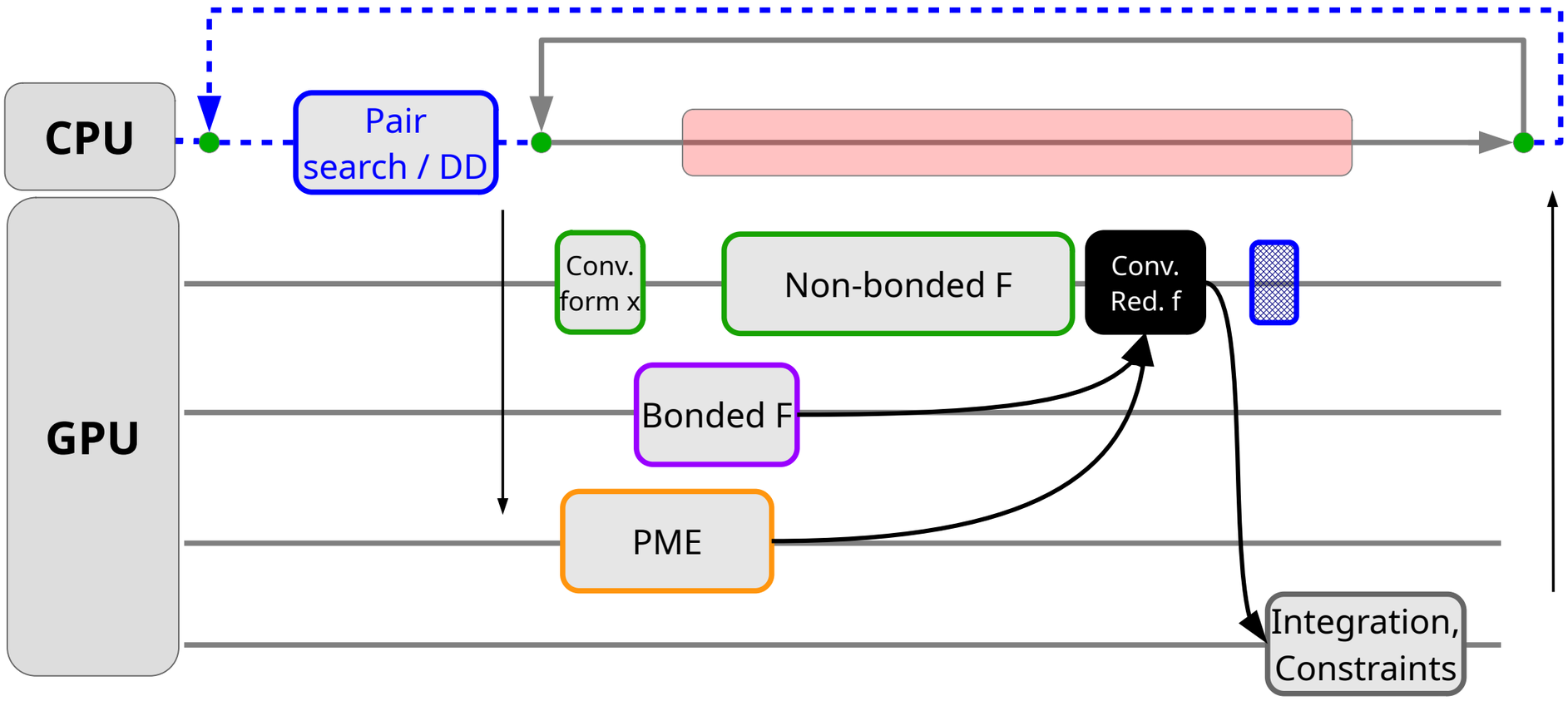
Full step offload: GPU resident step





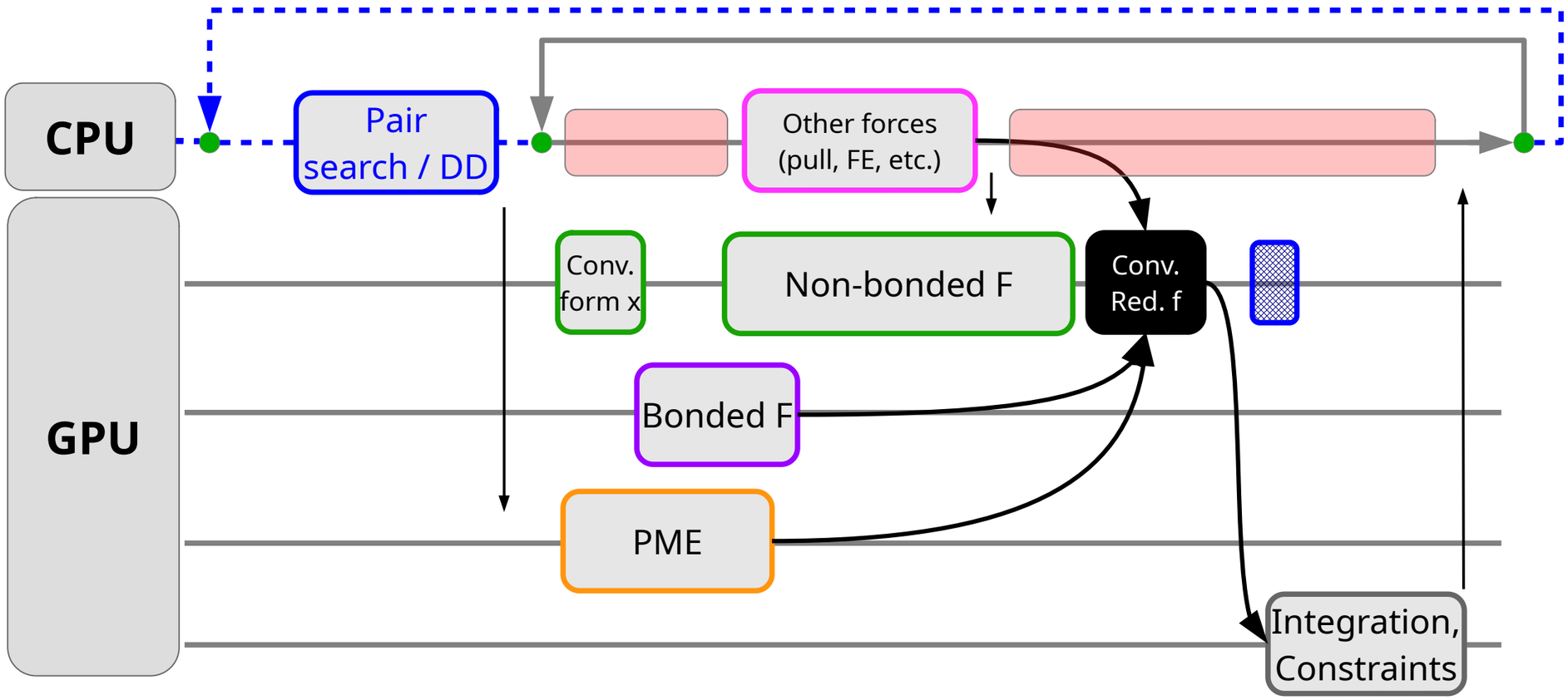
- Offload integration & constraints:
  - allows keeping x/f on the GPU for several iterations
  - avoid the CPU—GPU interconnect bottleneck
  - trade GPU idle time for CPU idle time

GPU resident steps



- Full step offloaded:
  - launch N iteration in the inner loop (asynchronously)
  - transfer & wait only when CPU dependencies require it (IO, search/DD)
- CPU is left mostly idle: tradeoff to keep GPU busy
  - opportunity: performance & support other forces

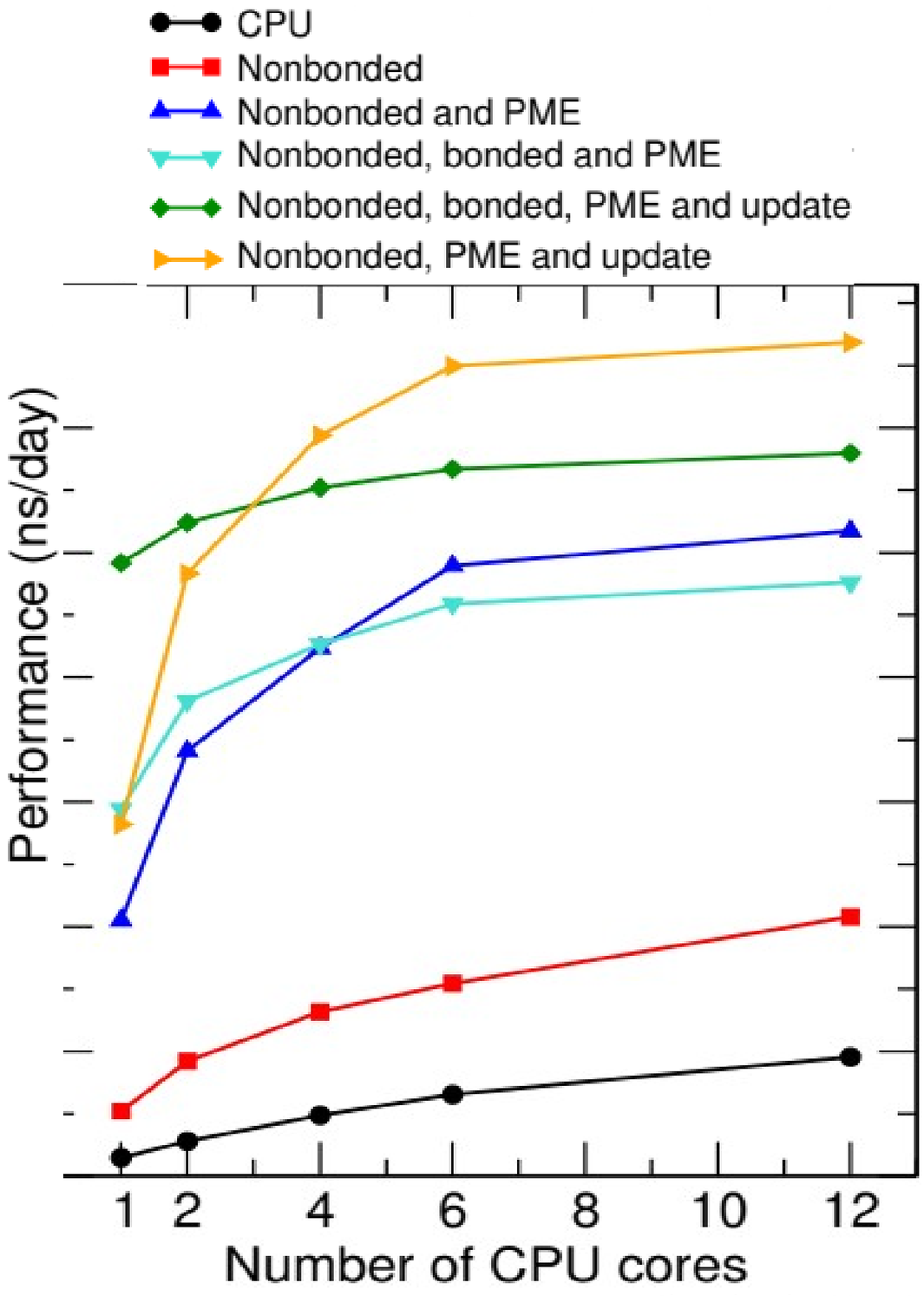
GPU resident steps



- CPU in supporting role (“back-offload”):
  - any algorithm not ported to the GPU
  - infrequent tasks, complex code (search, DD)

→ opportunities to exploit data residency for direct GPU communication

GROMACS heterogeneous schemes: hardware balance



- Best performance:
  - with few cores/GPU: offload everything
  - from 3-4 cores/GPU: bondeds on CPU

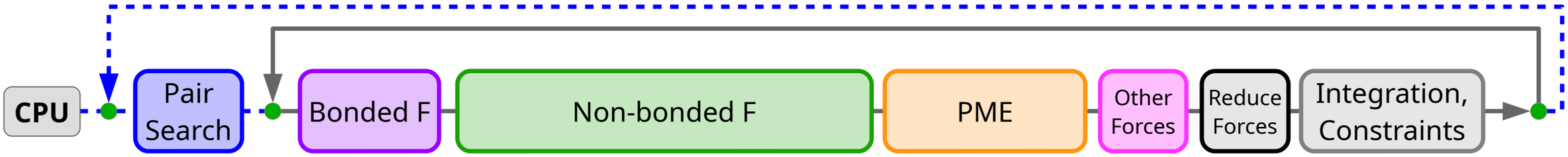
Benchmark system: GluCL ion channel (144k atoms)

Hardware:

- AMD R3900X CPU
- NVIDIA 2080 SUPER GPU
- PCIe 3.0 interconnect (slow)

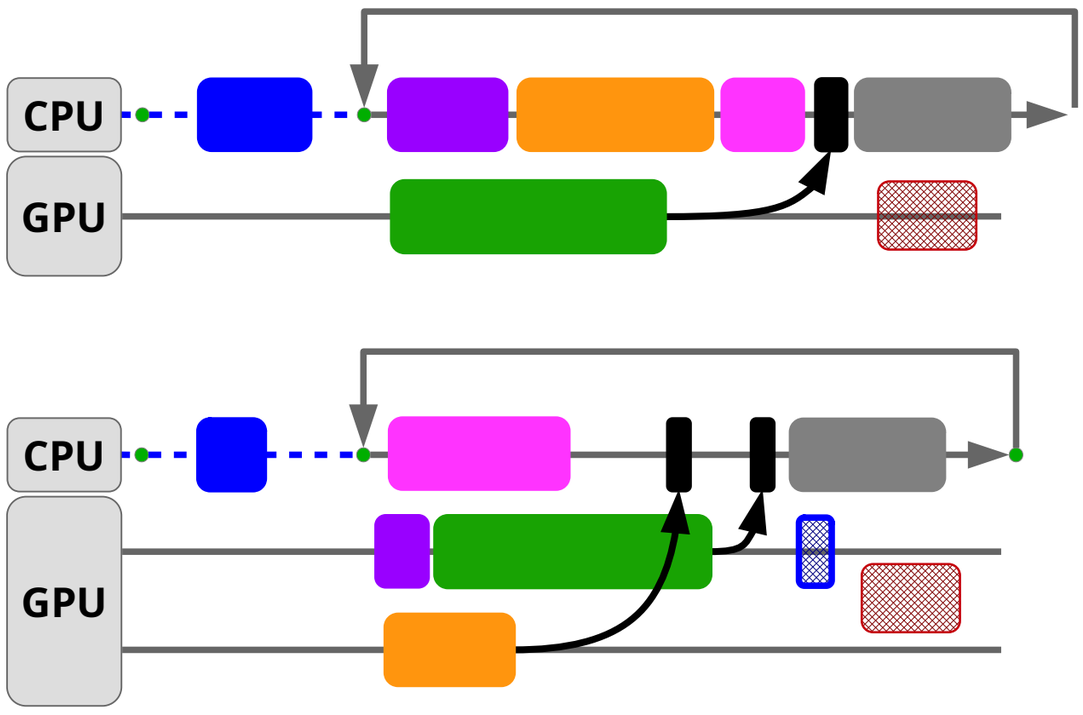
GROMACS GPU heterogeneous schemes [recap]

Homogeneous scheme

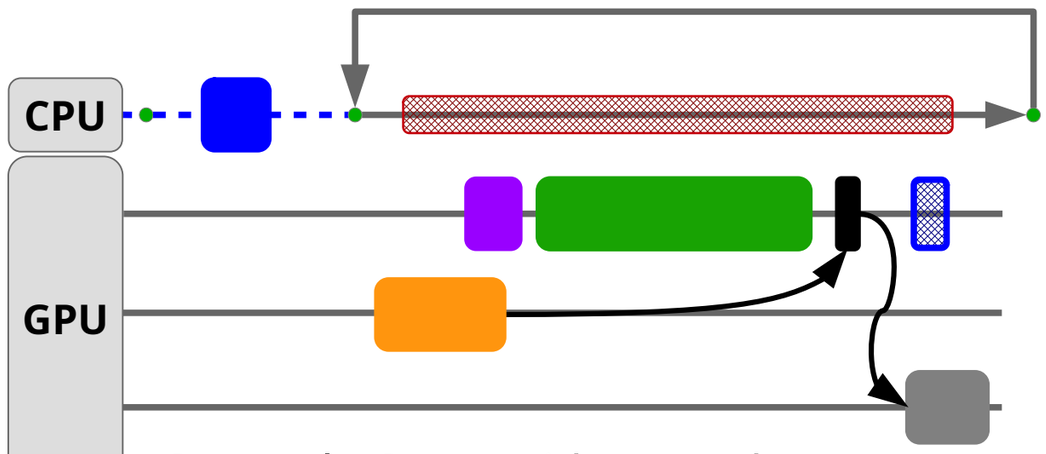


Heterogeneous schemes

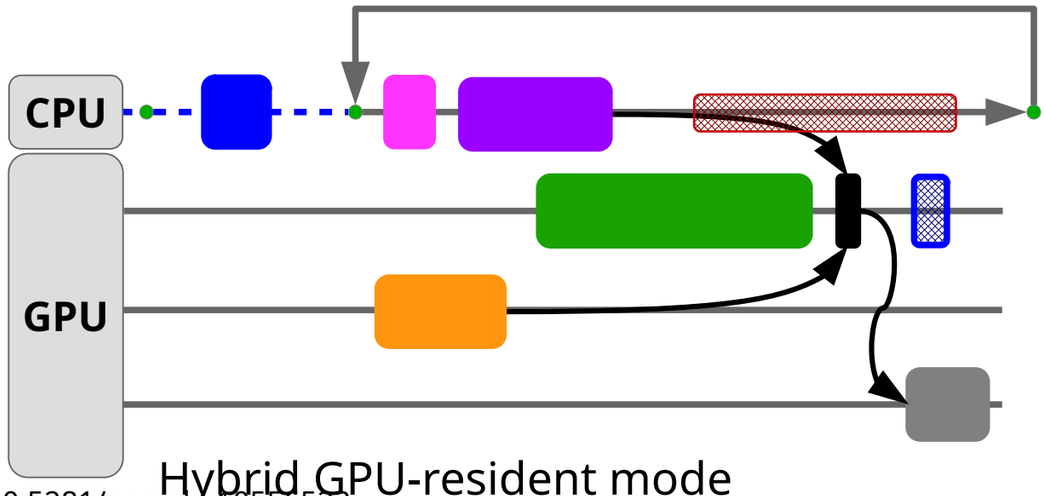
Force offload parallelization



GPU-resident parallelization

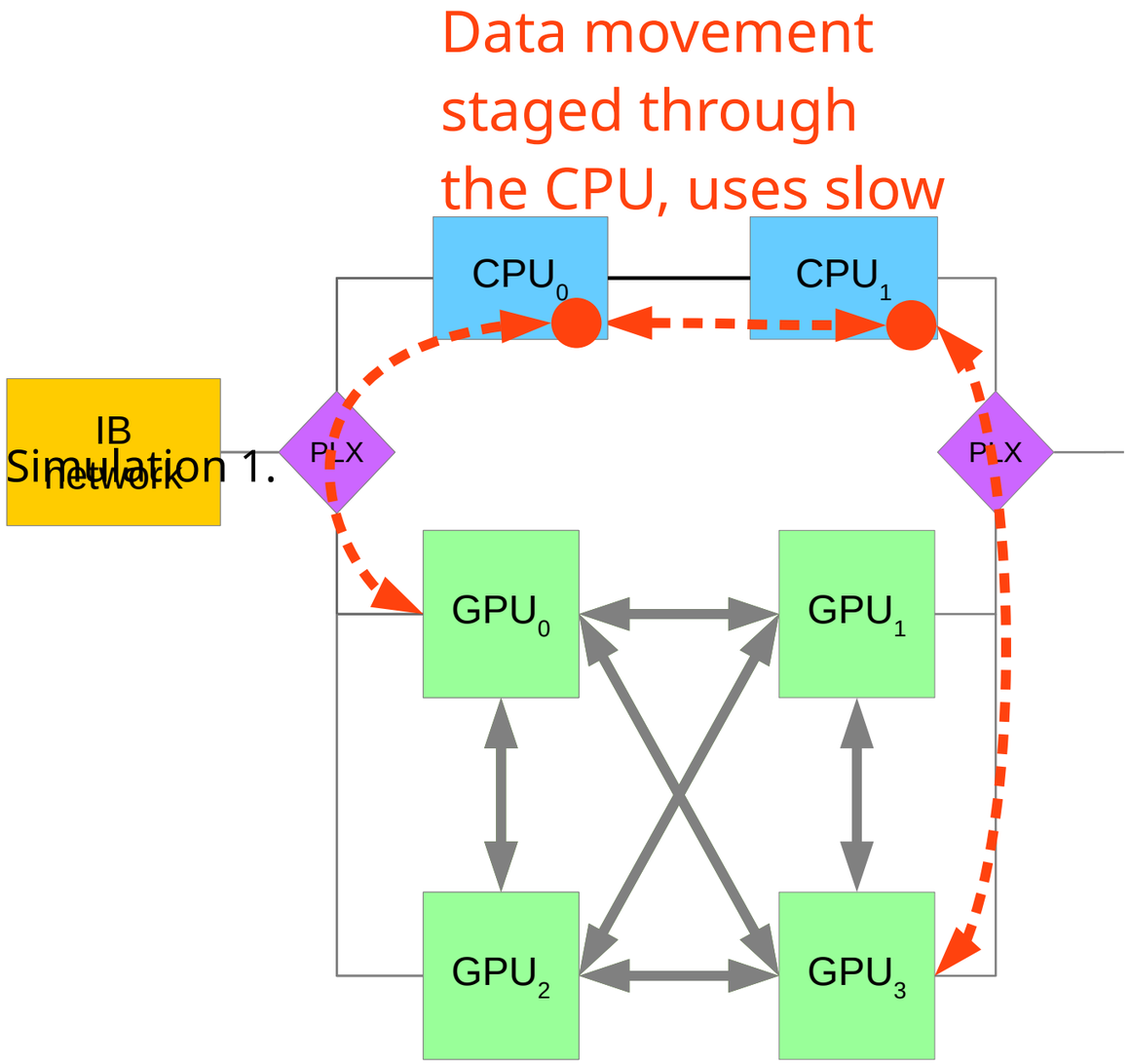


GPU-only GPU-resident mode



Hybrid GPU-resident mode

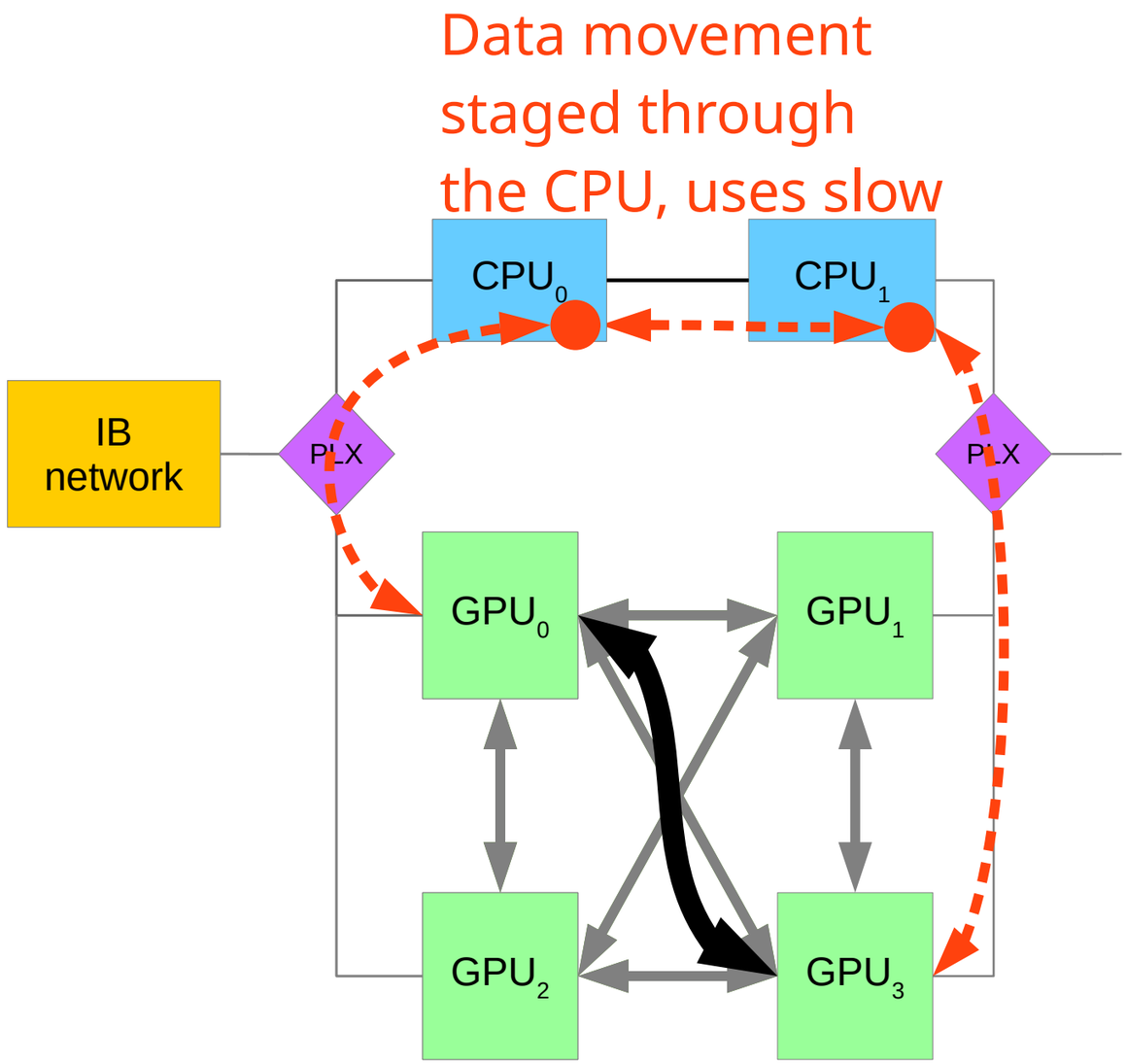
Staged GPU communication



Simulation 1.

- Staging through the CPU: “the straightforward” option:
  - GPU→ CPU → [CPU →] GPU
  - multiple hops add latency
  - does not allow making use of fast interconnects
  - blocks the CPU while waiting for data

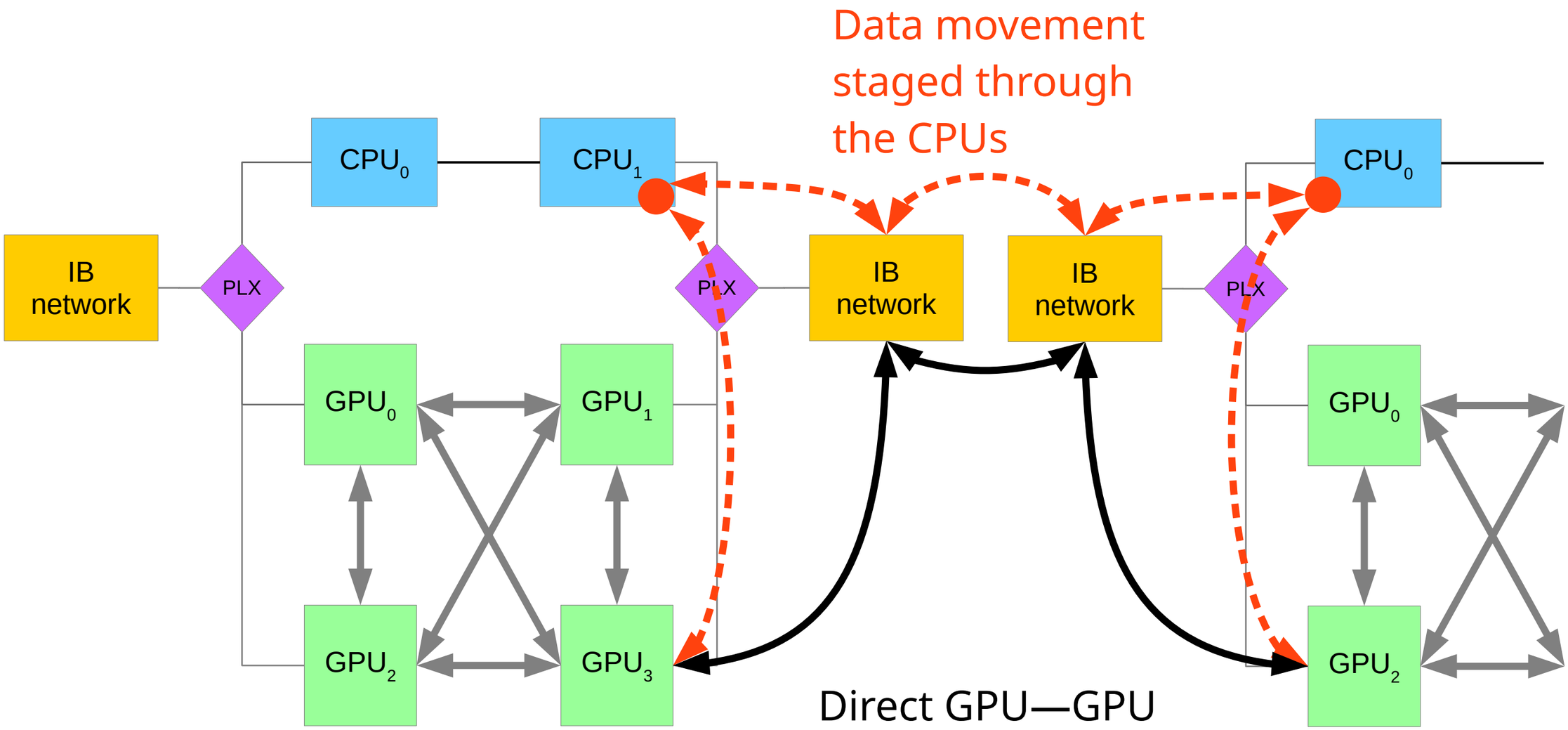
GPU-aware / direct GPU communication



Data movement direct: skips CPU & make **use of fast GPU—GPU path**

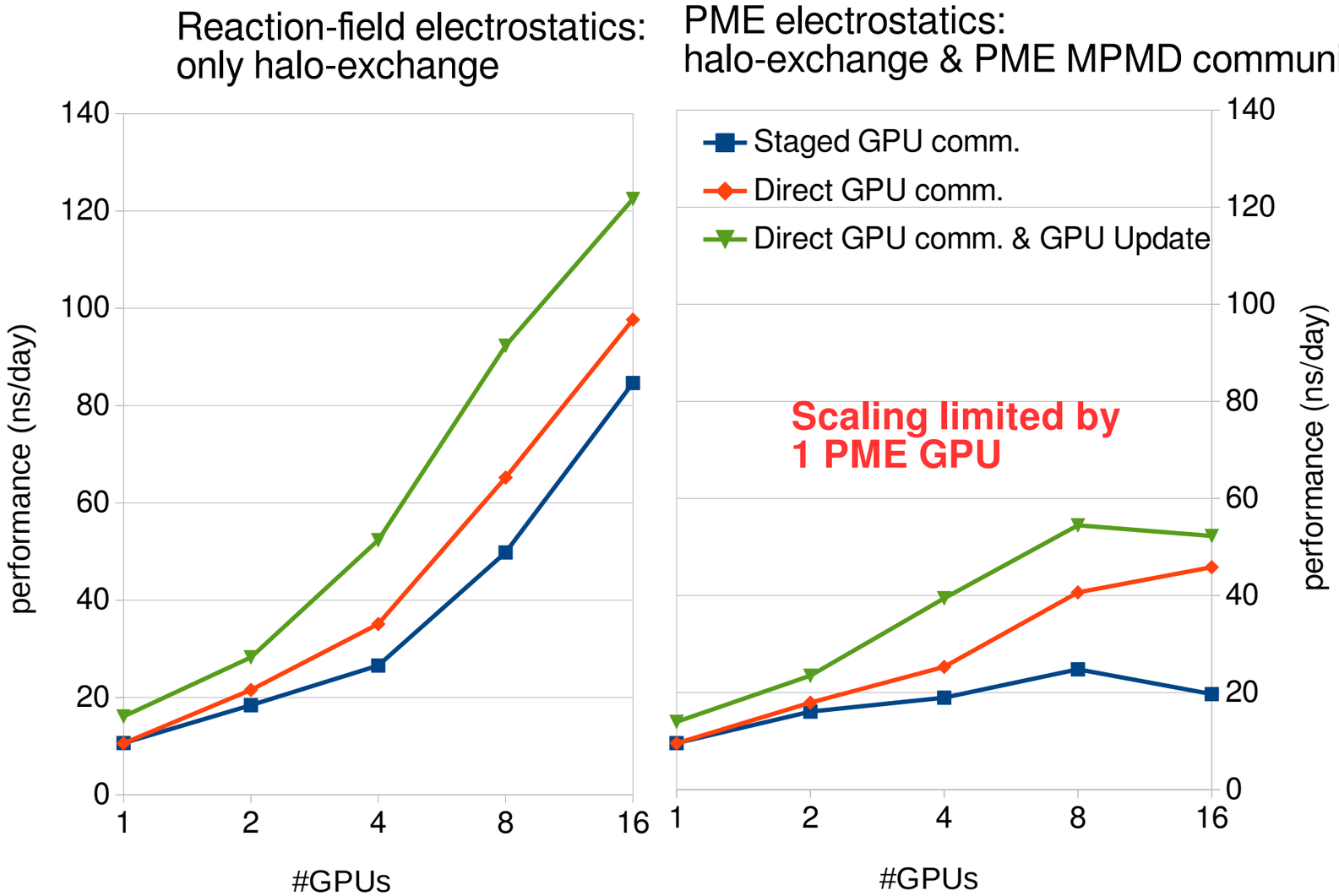
- GPU-aware communication:
  - move data without the (explicit) staging by application code
  - GPU-aware MPI
  - NVIDIA nccl, AMD rccl
  - NVSHMEM

GPU-aware / direct GPU communication



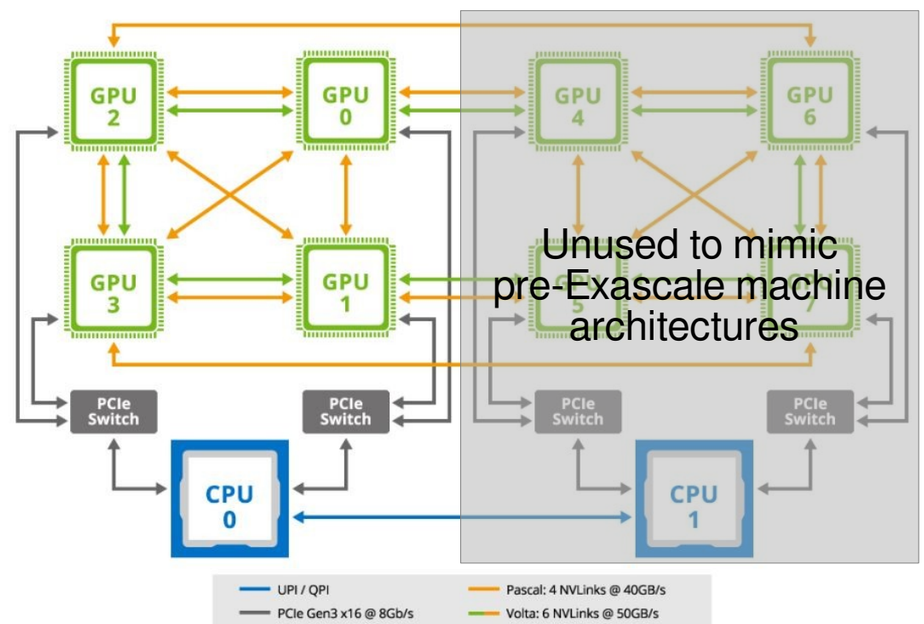
Direct GPU communication performance

- Major benefit on fast interconnects with GPU-resident steps
- Improvements on low-end interconnects are modest

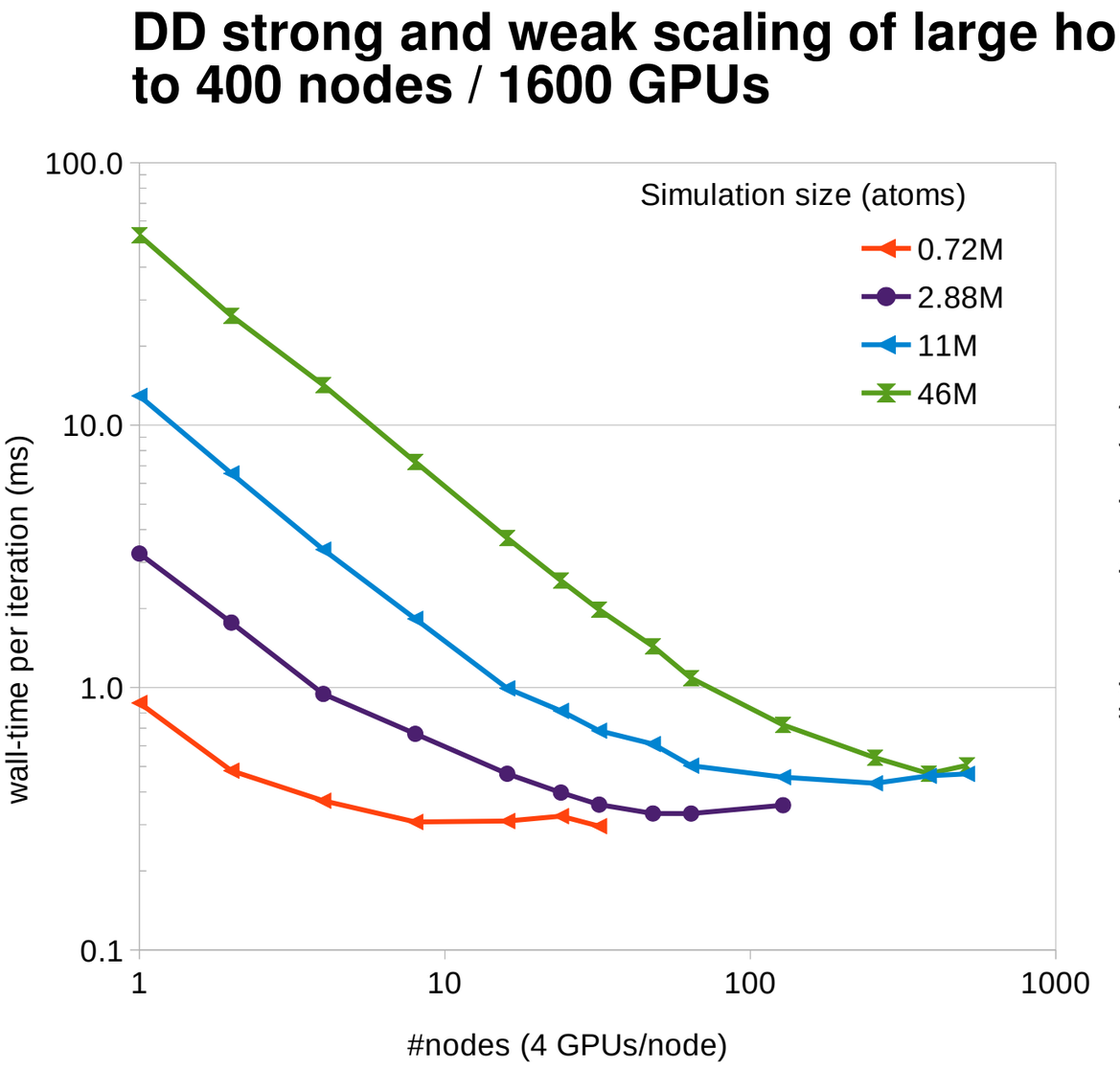


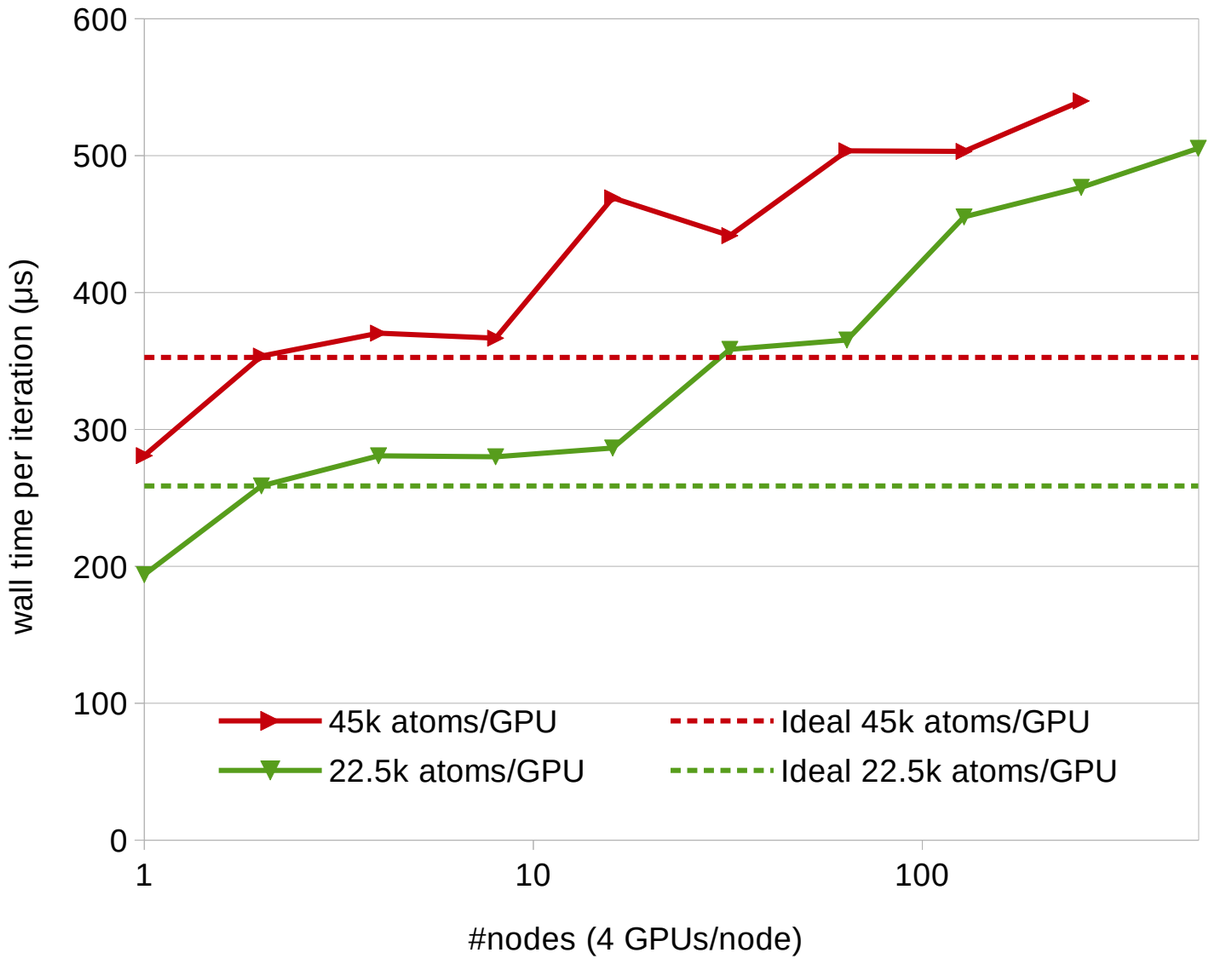
System: STMV 1M atoms

Hardware: DGX-1V



GPU DD halo exchange: great strong scaling

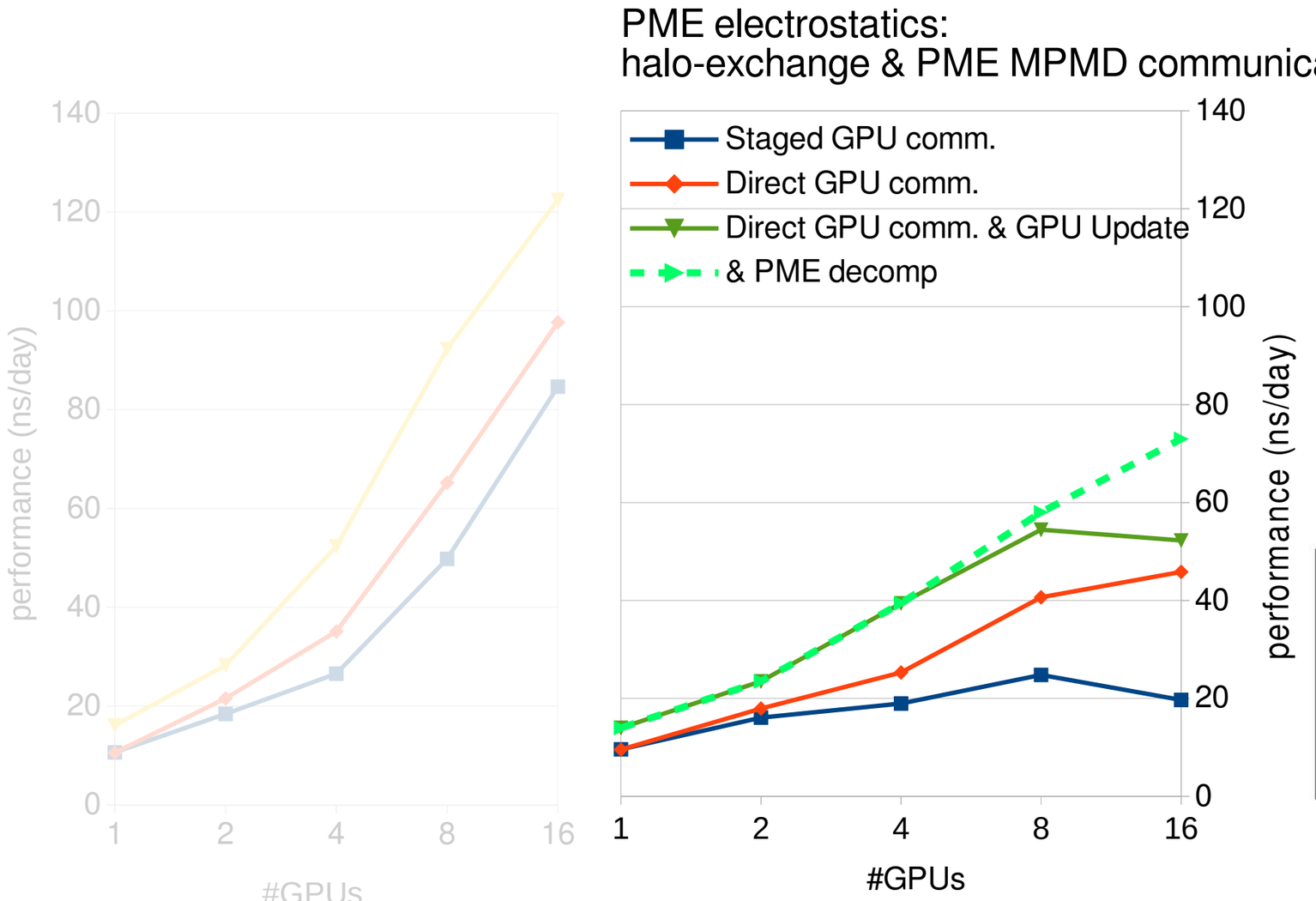




- Hardware: JUWELS-booster, 2x24-core AMD EPYC Rome, 4xA100, 4xNIC
- Scaling to ~10000 atoms/GPU on up to 1600 GPUs
- Parallel efficiency ~40-50% with 50000 atoms/GPU

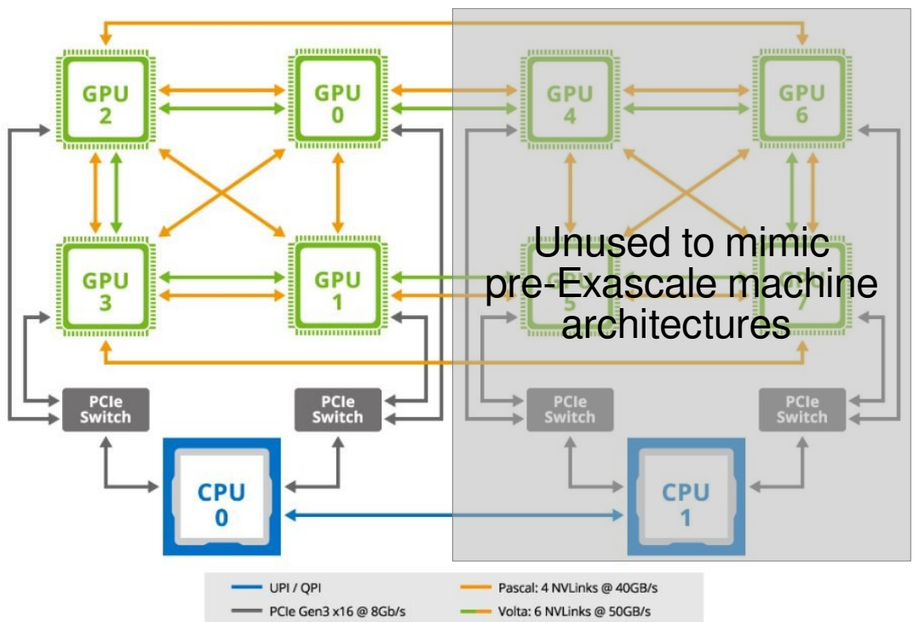
PME decomposition

- Allow running PME across multiple GPUs
- 3D FFTs strong-scaling challenges: typical size 32^3-256^3, very hard to scale
- Distributed GPU FFT libraries needed (for GPU-resident mode)
  - cuFFTmp (NVIDIA)
  - HeFFT (portable, AMD, Intel, NVIDIA)

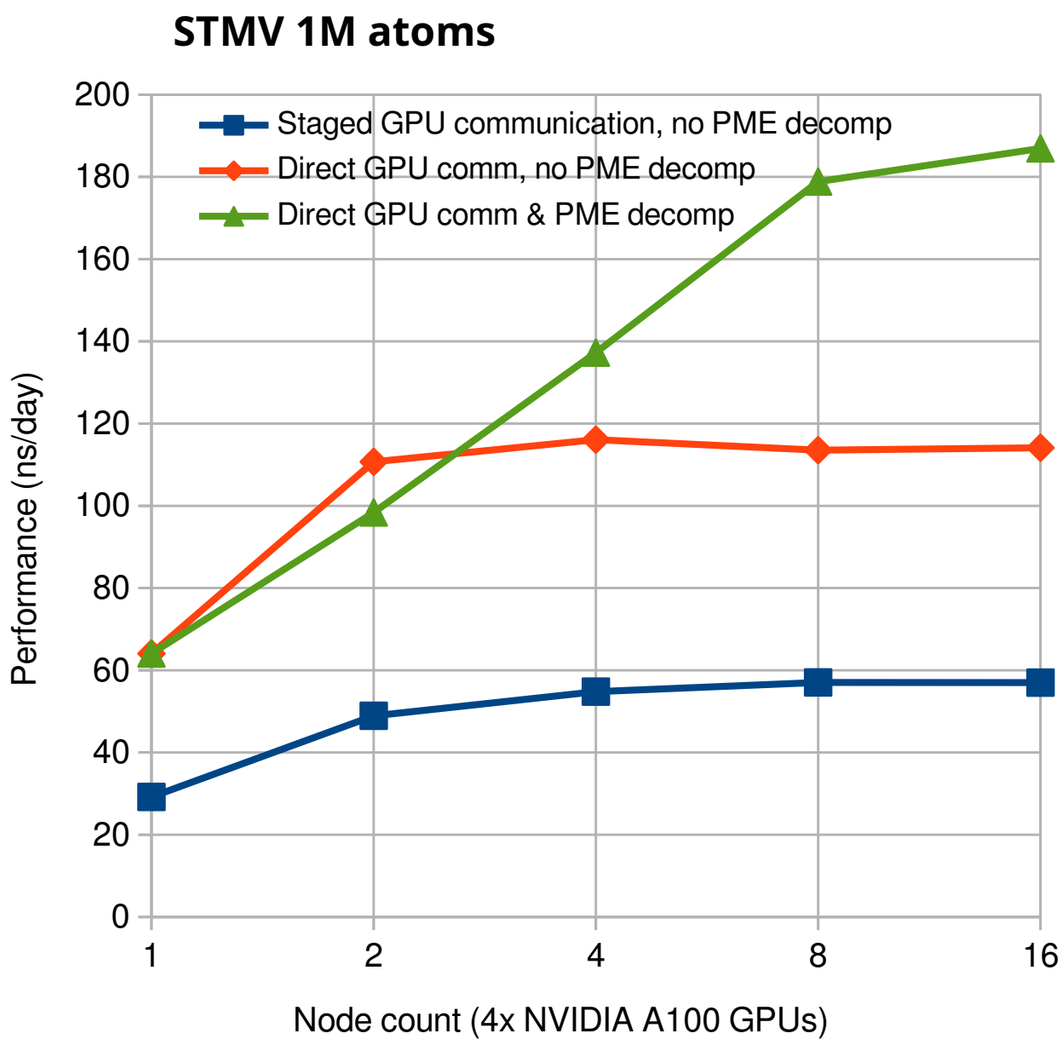


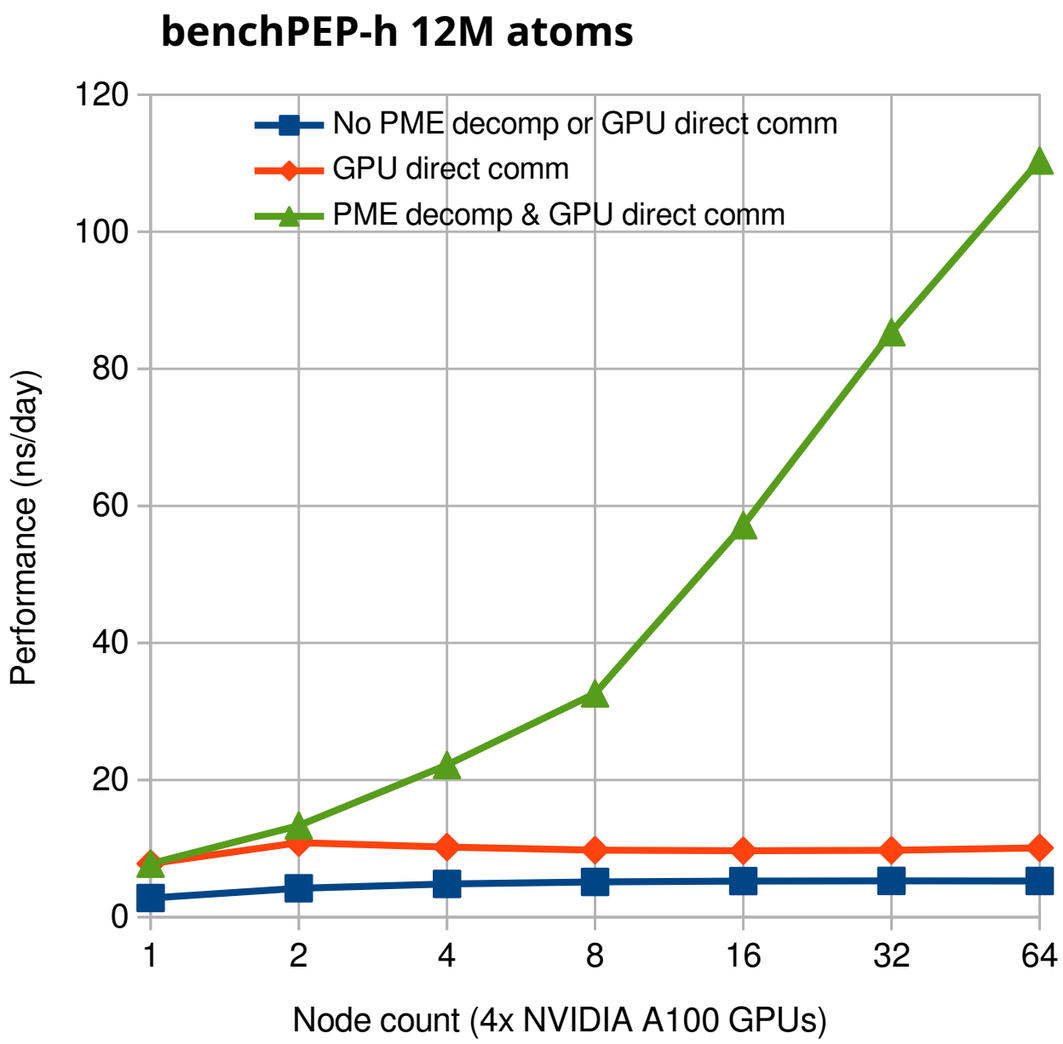
System: STMV 1M atoms

Hardware: DGX-1V



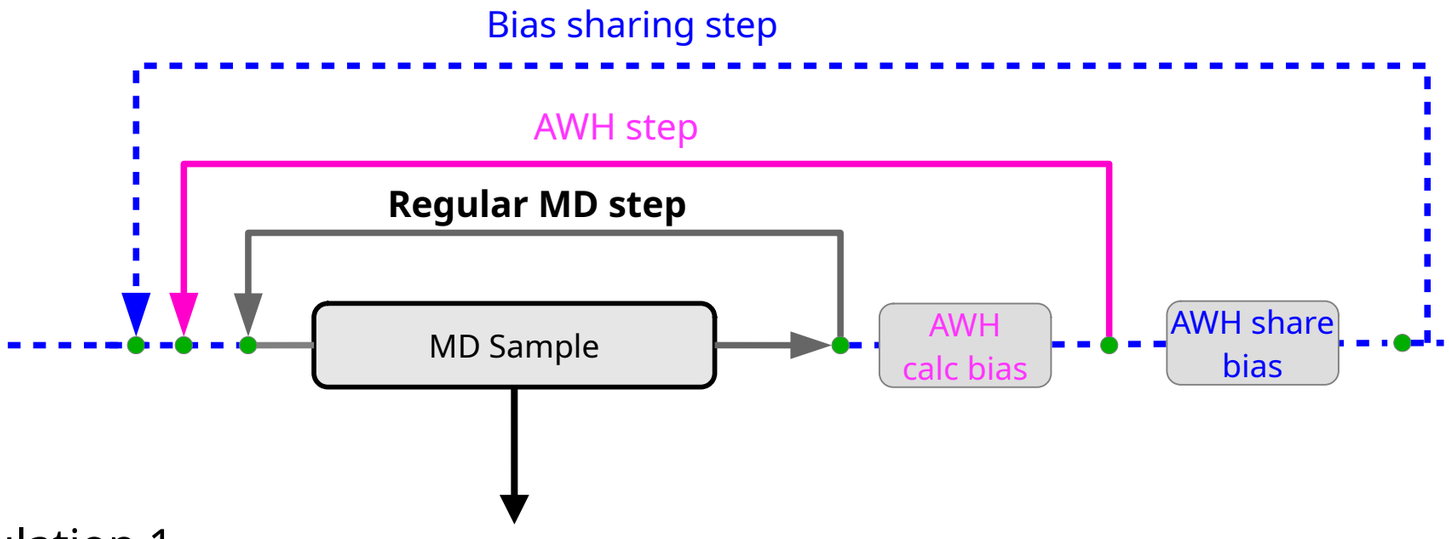
Direct GPU communication & PME decomposition





- Hardware: NVIDIA Selene (half node) 1x64-core AMD EPYC Rome, 4xA100, 4xNIC

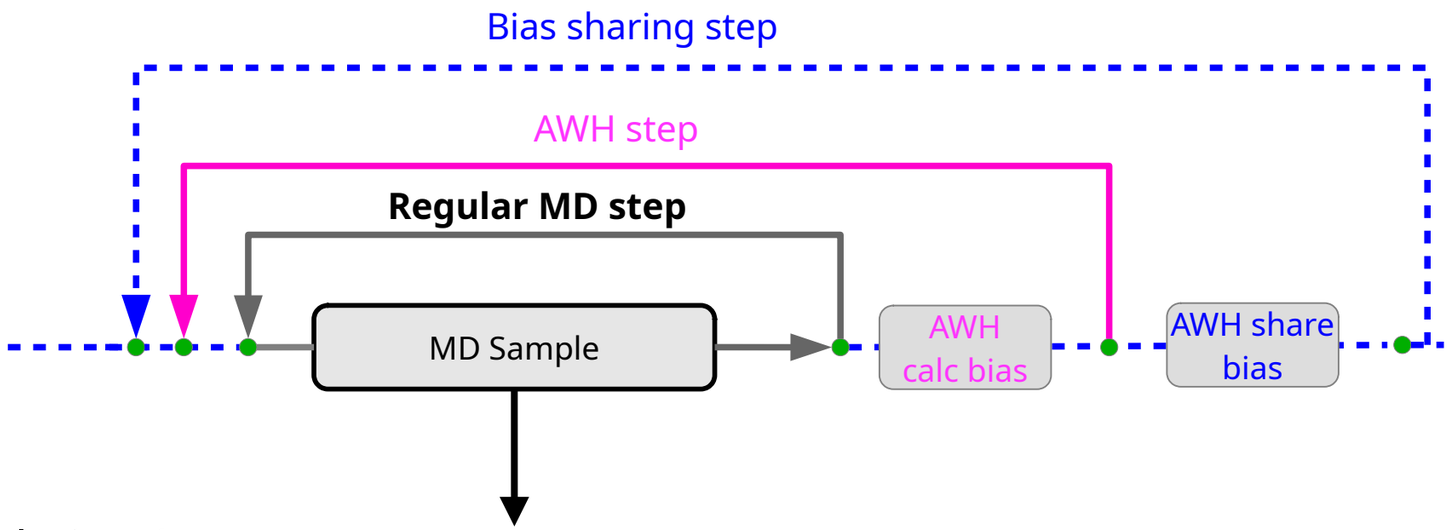
Ensembles on heterogeneous hardware



Simulation 1.

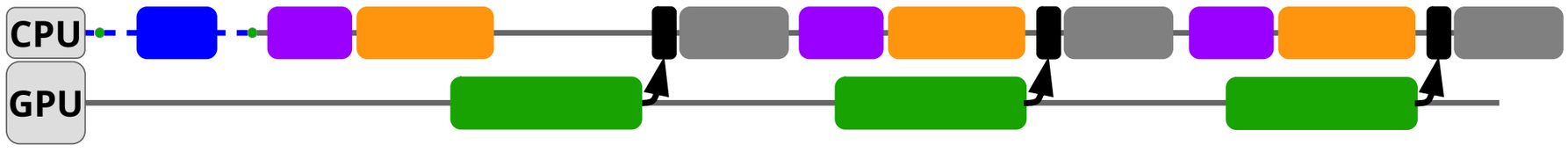
Simulation 2.

Ensembles on heterogeneous hardware



Simulation 1.

Simulation 2.

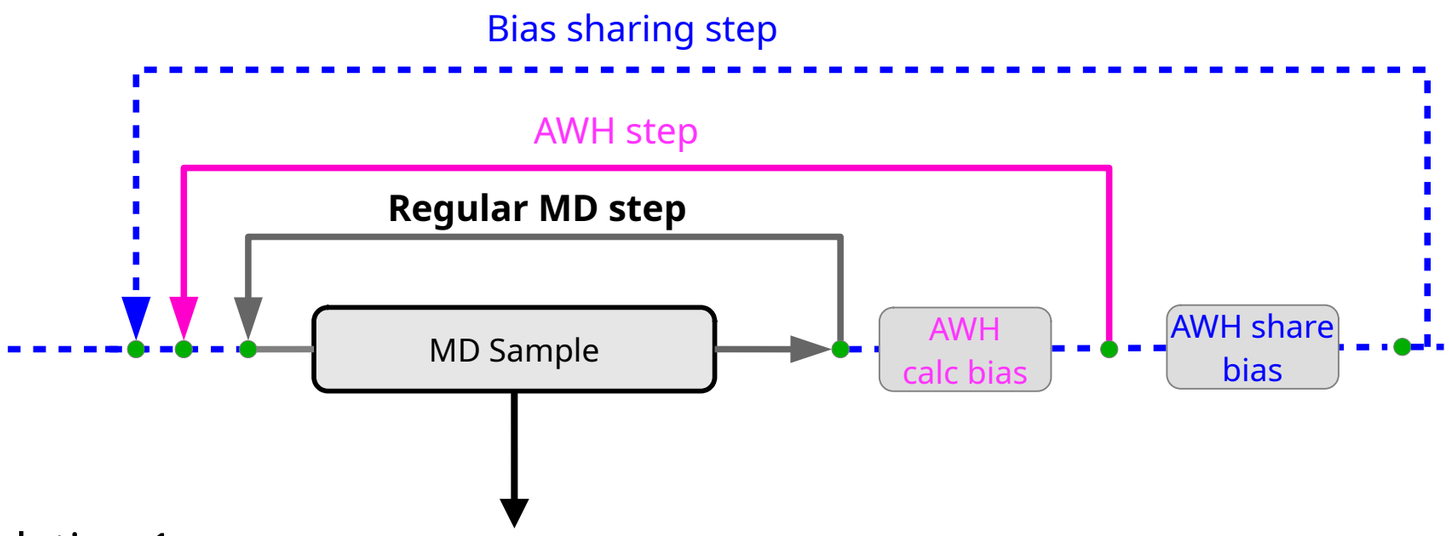


Between data exchanges **simulations can go out of sync**

→

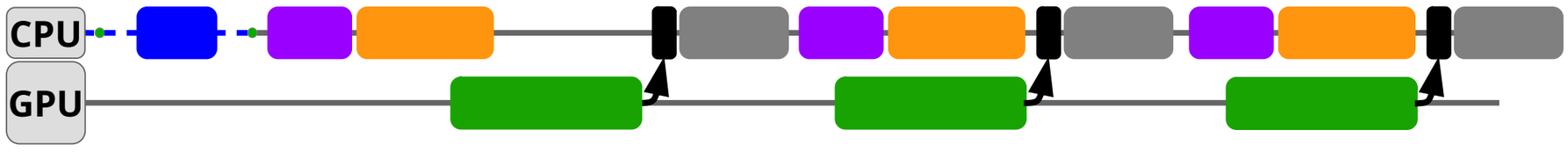
With a shared GPU: **increase effective GPU utilization**

Ensembles on heterogeneous hardware



Simulation 1.

Simulation 2.



Between data exchanges **simulations can go out of sync**

→

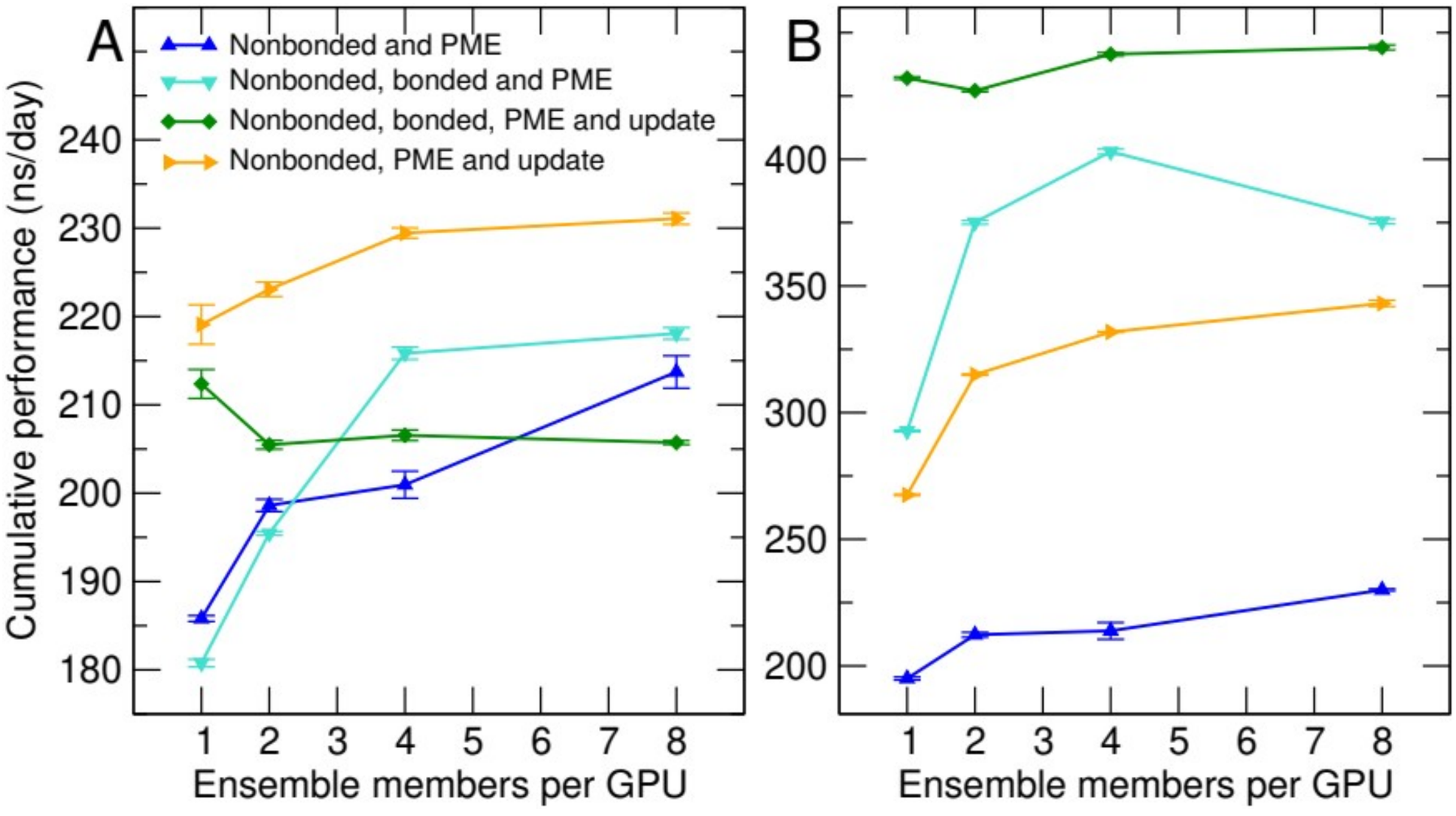
With a shared GPU: **increase effective GPU utilization**

Simulation 1 + 2 GPU timelines overlayed: **most gaps are gone!**

Ensemble performance & hardware balance

CPU: 2 x Xeon E5-2620 v4
GPU: 4 x GeForce **GTX 1080 GPU**

CPU: 2 x Xeon E5-2620 v4
GPU: 4 x GeForce **GTX 2080 GPU**



Benchmark:
Aquaporin (110k atom CHARMM FF)

coupled AWH ensemble run

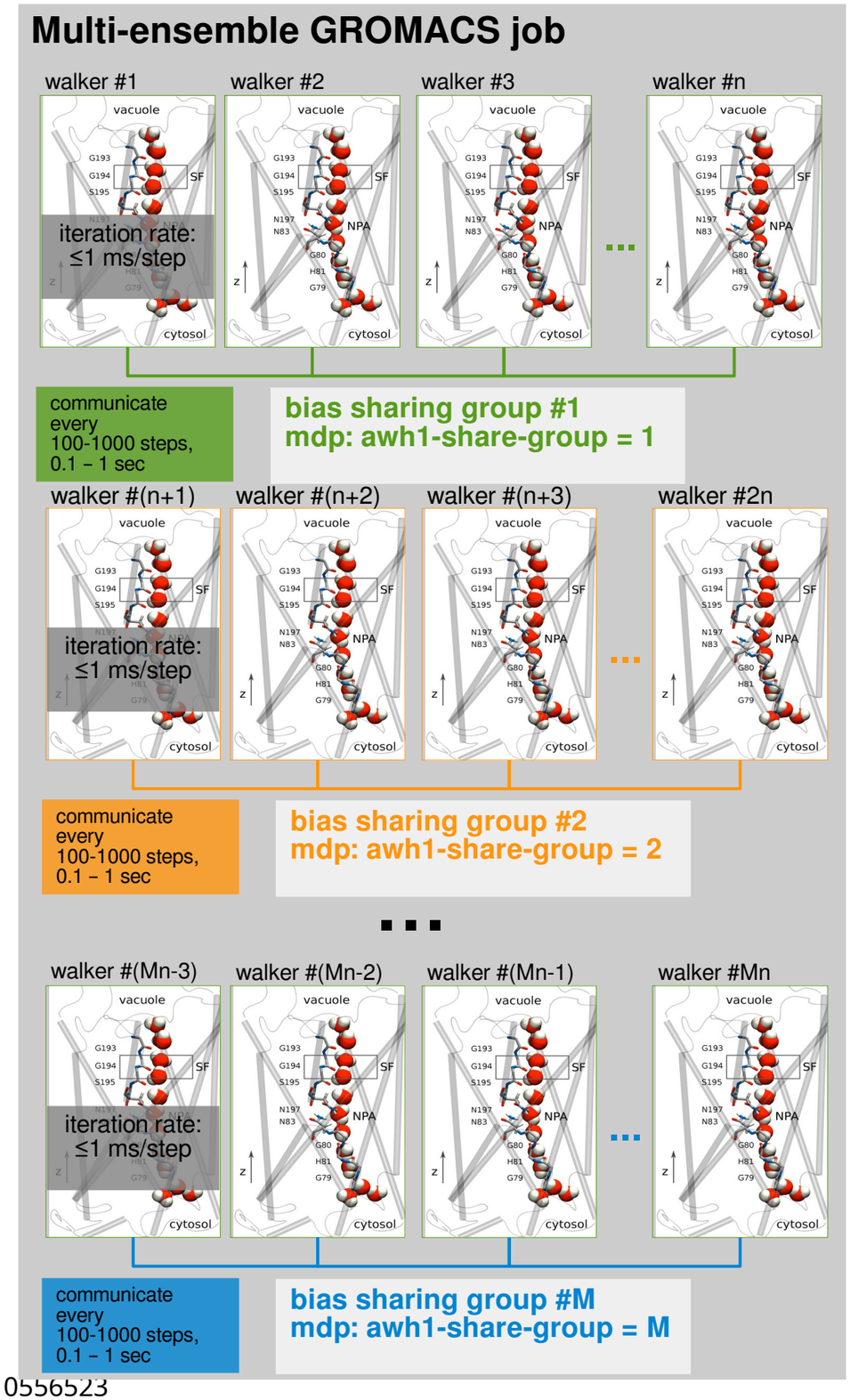
Performance saturates around 3-4 runs/GPU

CPU-GPU relative performance determines whether to use CPU for some forces

Running large-scale ensembles

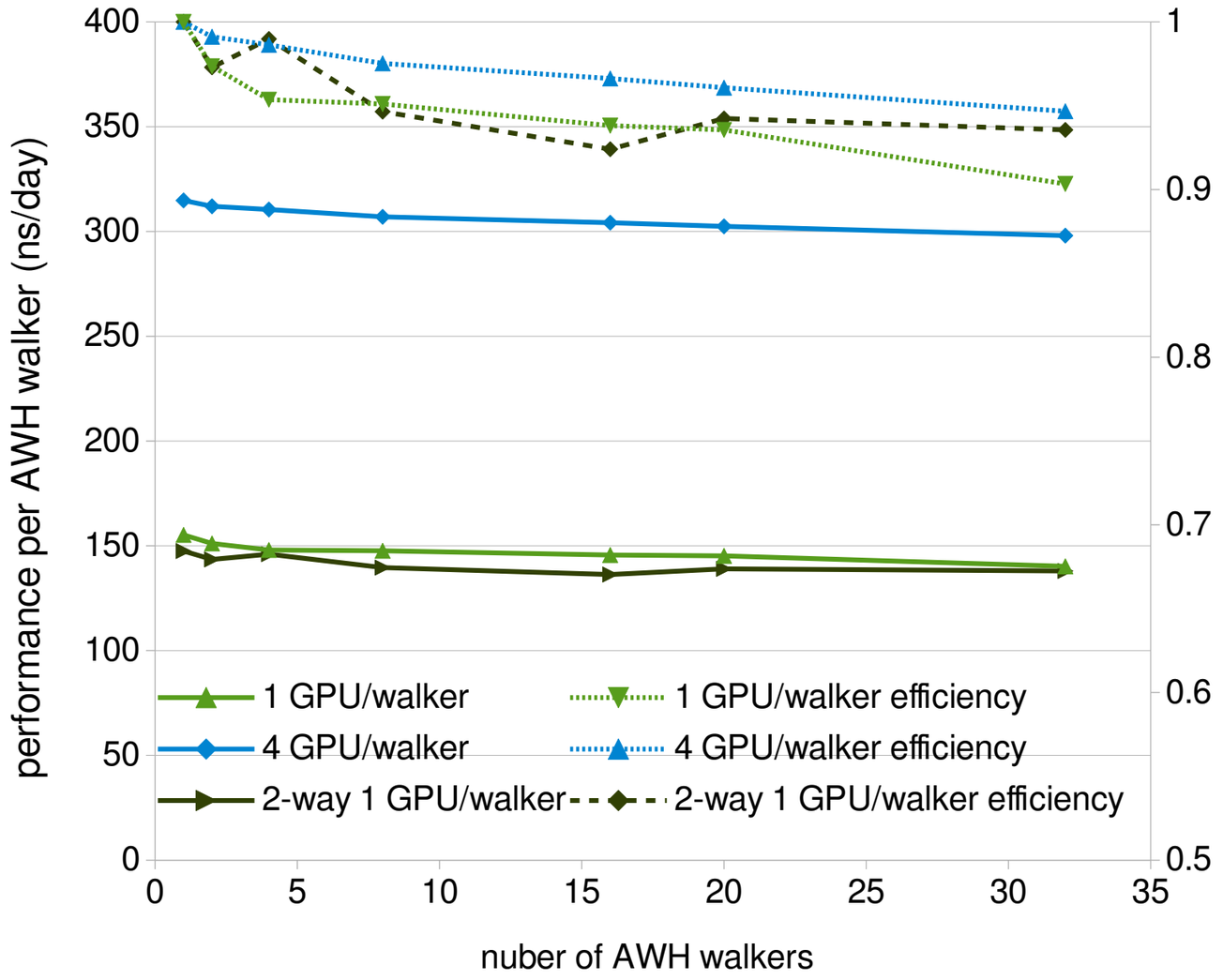
Scheduling challenges

- mapping parallelization hierarchy to the machine
- placement of tightly vs loosely coupled parts of the job
- How to map to hardware to optimize?
  - time-to-solution / cost-to-solution
  - energy-to-solution
- inter-simulation load imbalance
- efficiency vs throughput



AWH ensemble scaling

- Good parallel efficiency
  - Caveats/needs:
    - good node placement
    - low noise (avoid divergent tuning
- flexible sharing with low overhead
- machine topology a major challenge



- Heterogeneous coupled jobs can span across exascale machines:

  e.g. ~32 nodes/member x ~32-way ensemble = 1024 nodes (x N-way flex sharing)

Acknowledgments

GROMACS

Andrey Alekseenko

Artem Zhmurov

Berk Hess

Erik Lindahl

Magnus Lundborg

Paul Bauer

Mark Abraham (Intel)

Roland Schulz (Intel)

Alan Gray (NVIDIA)

Gaurav Garg (NVIDIA)

Mahesh Doijade (NVIDIA)

Ania Brown (NVIDIA)

HW / code contrib

Funding

Links and resources

- Further reading:
  - S. Páll, A. Zhmurov, P. Bauer, M. Abraham, M. Lundborg, A. Gray, B. Hess, & E. Lindahl (2020). Heterogeneous Parallelization and Acceleration of Molecular Dynamics Simulations in GROMACS. J. Chem. Phys. 153, 134110 (2020); https://doi.org/10.1063/5.0018516
  - Maximizing GROMACS Throughput with Multiple Simulations per GPU Using MPS and MIG https://developer.nvidia.com/blog/maximizing-gromacs-throughput-with-multiple-simulations-per-gpu-using-mps-and-mig

- GROMACS documentation:

  https://manual.gromacs.org/documentation/current

- Post your questions on the GROMACS users' forum:

  https://gromacs.bioexcel.eu