Heterogeneous parallelization in GROMACS

lessons learned from a accelerating molecular dynamics on CPUs & GPUs

Szilárd Páll
pszilard@kth.se

June 9, 2020

Acknowledgments

GROMACS

Berk Hess

Artem Zhmurov

Erik Lindahl

Paul Bauer

Mark Abraham

Magnus Lundborg

Roland Schulz

Aleksei Yupinov

Alan Gray (NVIDIA)

Gaurav Garg (NVIDIA)

STREAM
High Performance Computing

Hardware

AMD

intel

NVIDIA

CSCS
Swiss National Supercomputing Centre

Funding

Swedish Foundation for Strategic Research

SeRC
Swedish e-Science Research Centre

Vetenskapsrådet

erc European Research Council

Molecular dynamics

- Newton's equation of motion

$$m_i \frac{d^2 \mathrm{r}_i}{dt^2} = \mathrm{F}_i = -\nabla_i U(\mathrm{r}),\ \ i = 1, ..., N$$

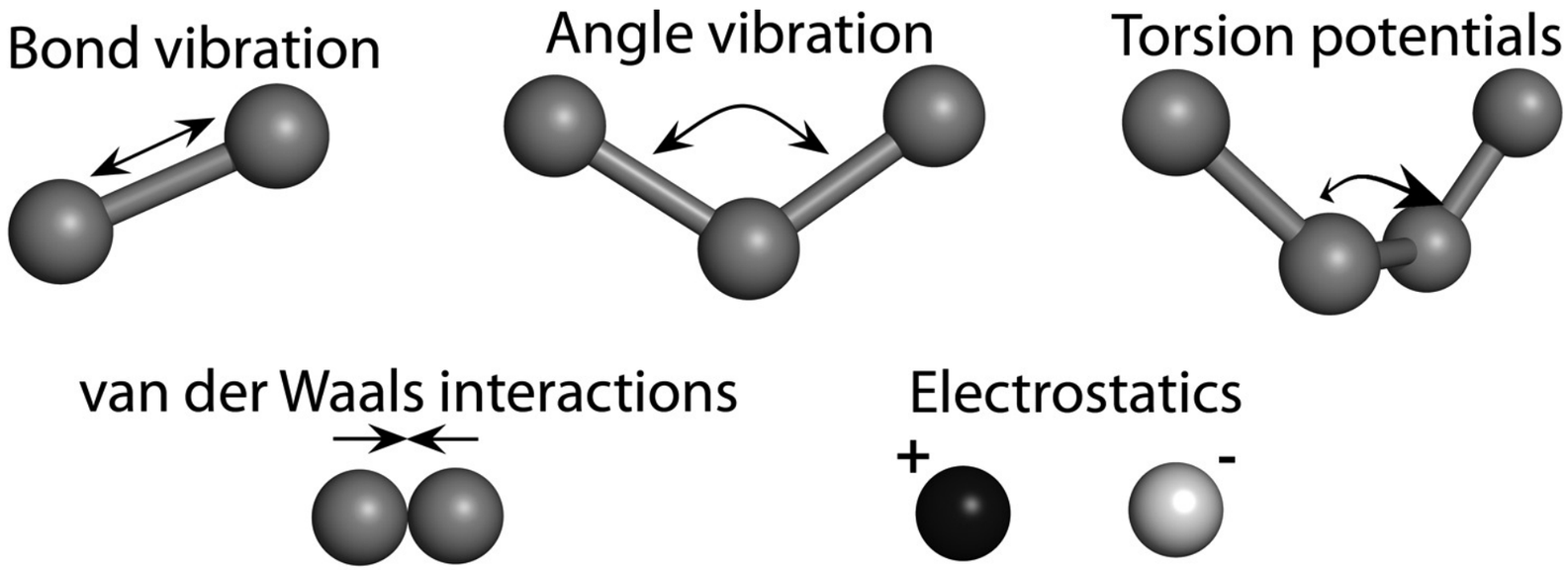
- Integrate e.g. leap-frog

$$v_i(t + \frac{\Delta t}{2}) = v_i(t - \frac{\Delta t}{2}) + a_i(t)\Delta t$$

$$r_i(t + \Delta t) = r_i(t) + v_i(t + \frac{\Delta t}{2})\Delta t$$

- Fundamental challenge:
  - **sequential** problem
  - **time-step limited** to 1-5 fs (by fastest atomic motions)

Main cost: computing forces



$$U(\mathbf{r}) = \sum_{bonds} U_{bond}(\mathbf{r}) + \sum_{angles} U_{angle}(\mathbf{r}) + \sum_{dihs} U_{dih}(\mathbf{r})$$ Bonded

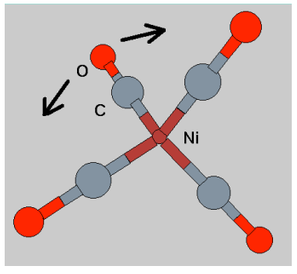
$$+ \sum_{i} \sum_{j>i} \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}} + \frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}}$$ Non-bonded

Over all atom-pairs!

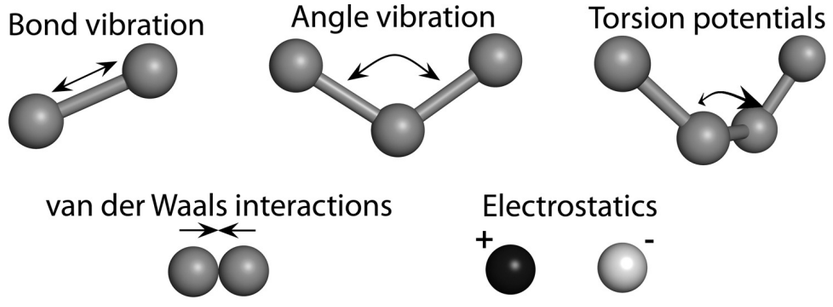
Compute force on each particle

$$\mathrm{F}_i = -\frac{dU}{d\mathrm{r}_\mathrm{i}}$$

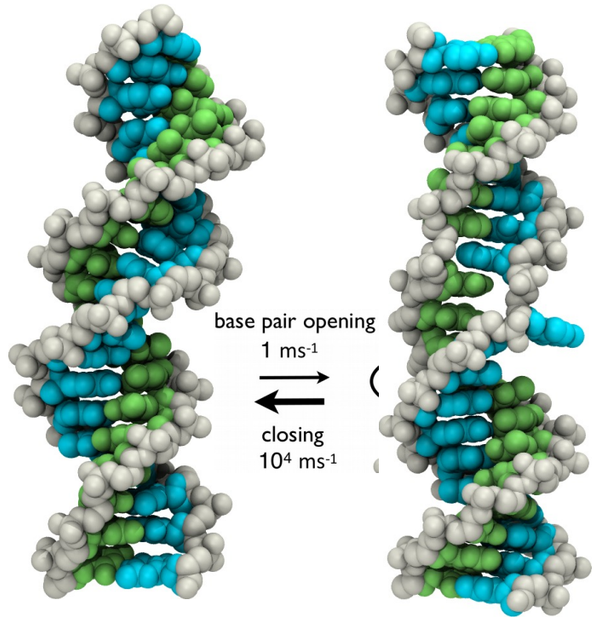
Timescale challenge



Physics



Chemistry

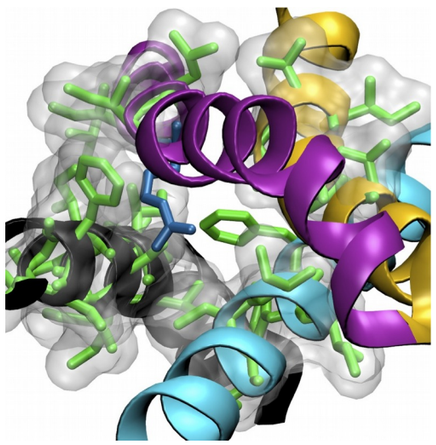


Biology

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
- model quality?



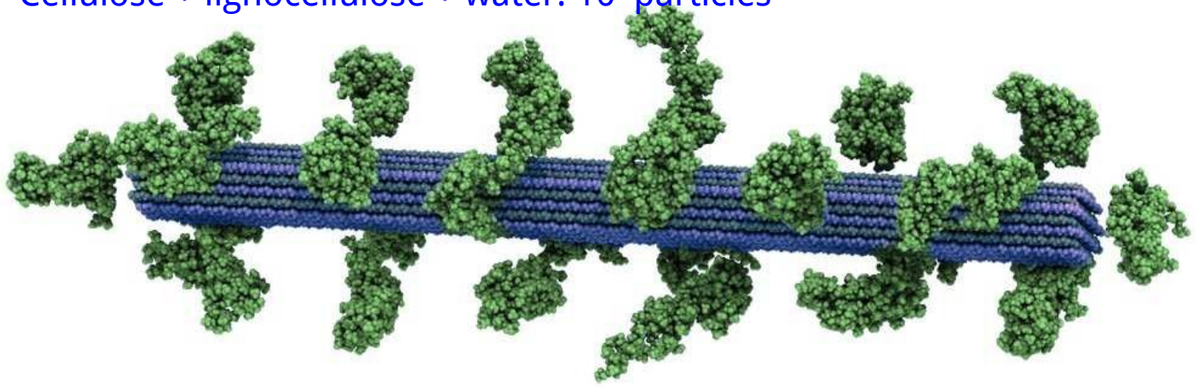
Experiments:

- lower detail
- higher efficiency
- high degree of averaging

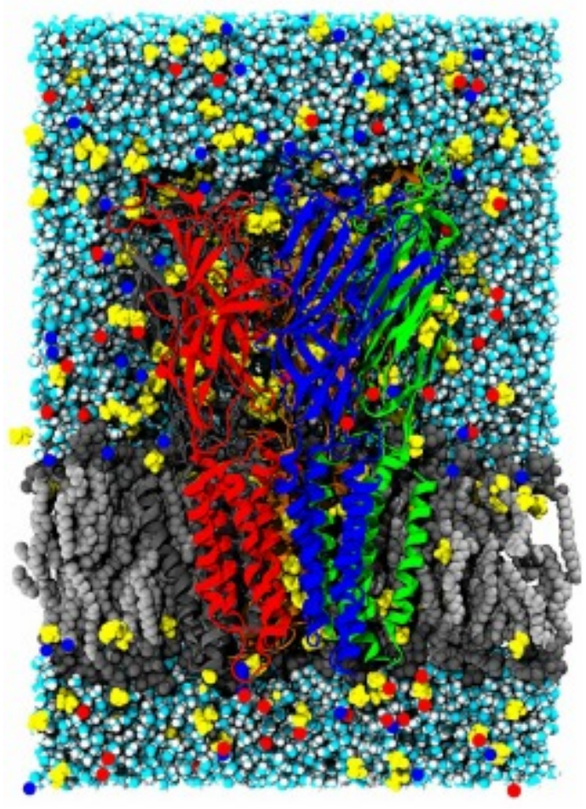
Molecular simulation: use-cases

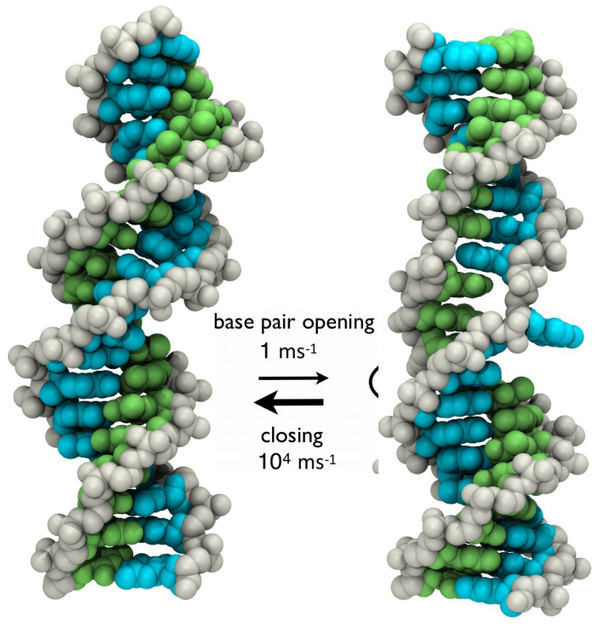
Biomolecular MD

Cellulose + lignocellulose + water: 10^7 particles



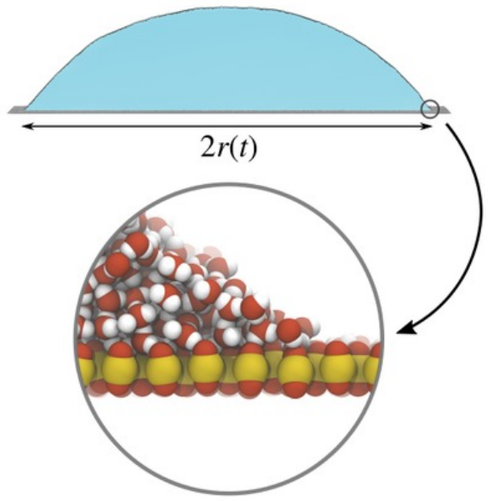
Membrane protein: 10^5 particles



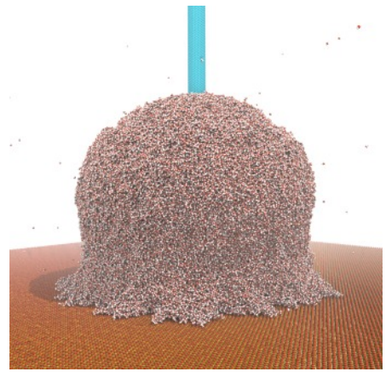


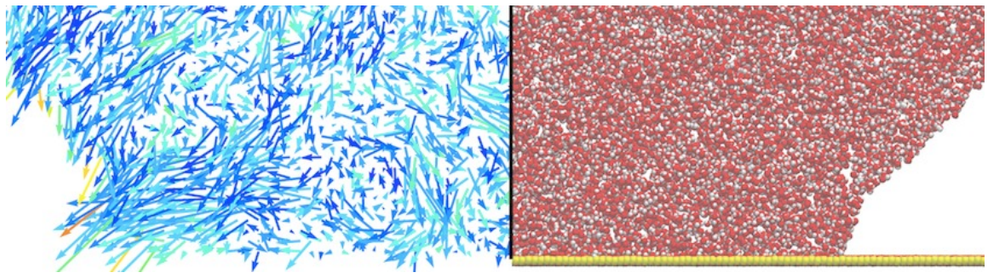
DNA base-pair opening: 10^4 particles

Materials MD

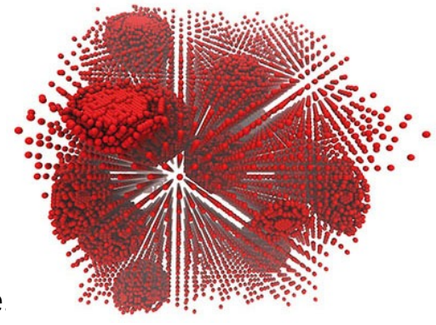


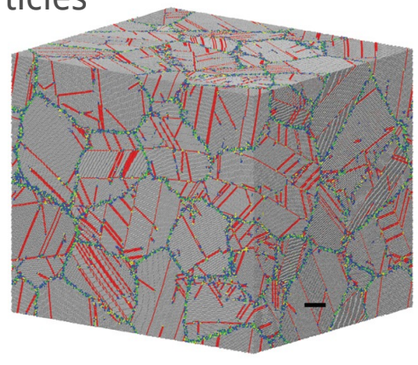
Contact line friction &
wetting dynamics
10^7 -10^9 particles





Nucleation in nano-crystals:
10^{10} -10^{12} particles





Molecular simulation: use-cases

Biomolecular MD

time-scale challenge

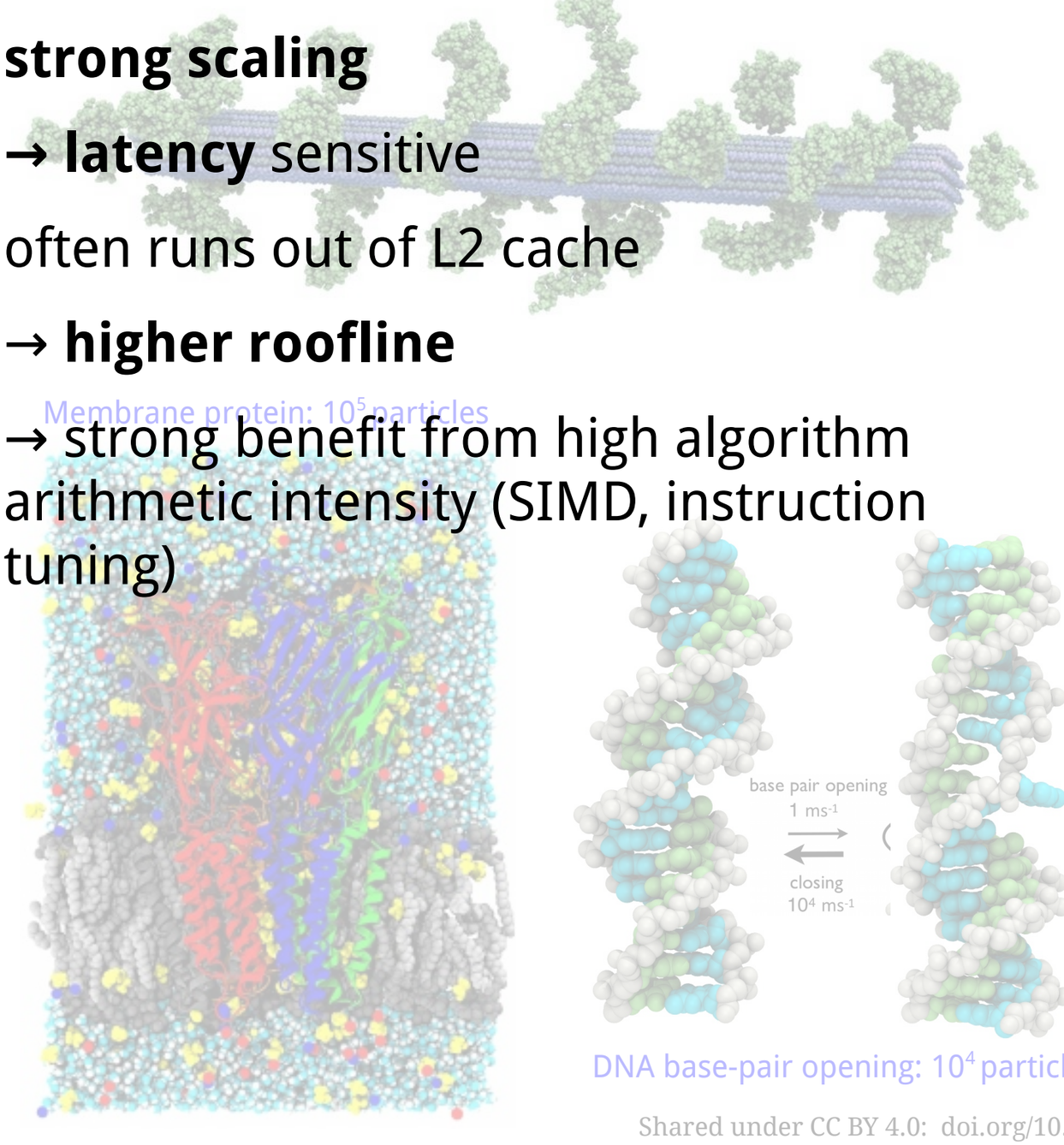
strong scaling

→ **latency** sensitive

often runs out of L2 cache

→ **higher roofline**

→ strong benefit from high algorithm arithmetic intensity (SIMD, instruction tuning)



Materials MD

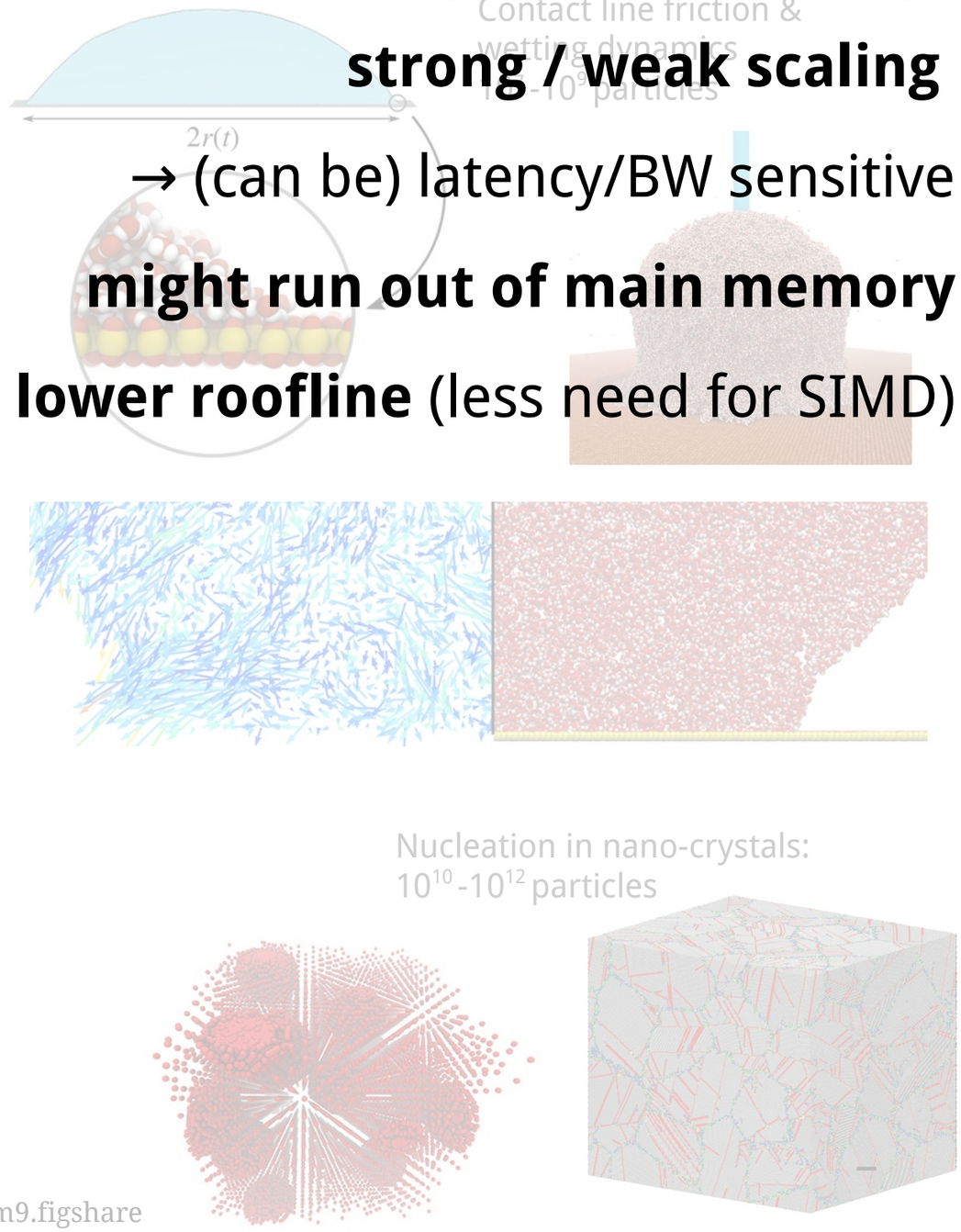
time- & length-scale challenge

strong / weak scaling

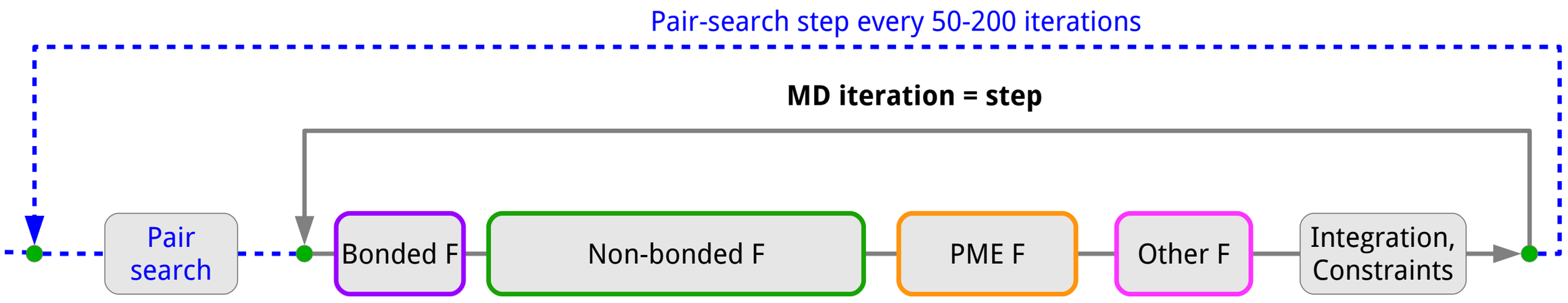
→ (can be) latency/BW sensitive

might run out of main memory

→ **lower roofline** (less need for SIMD)



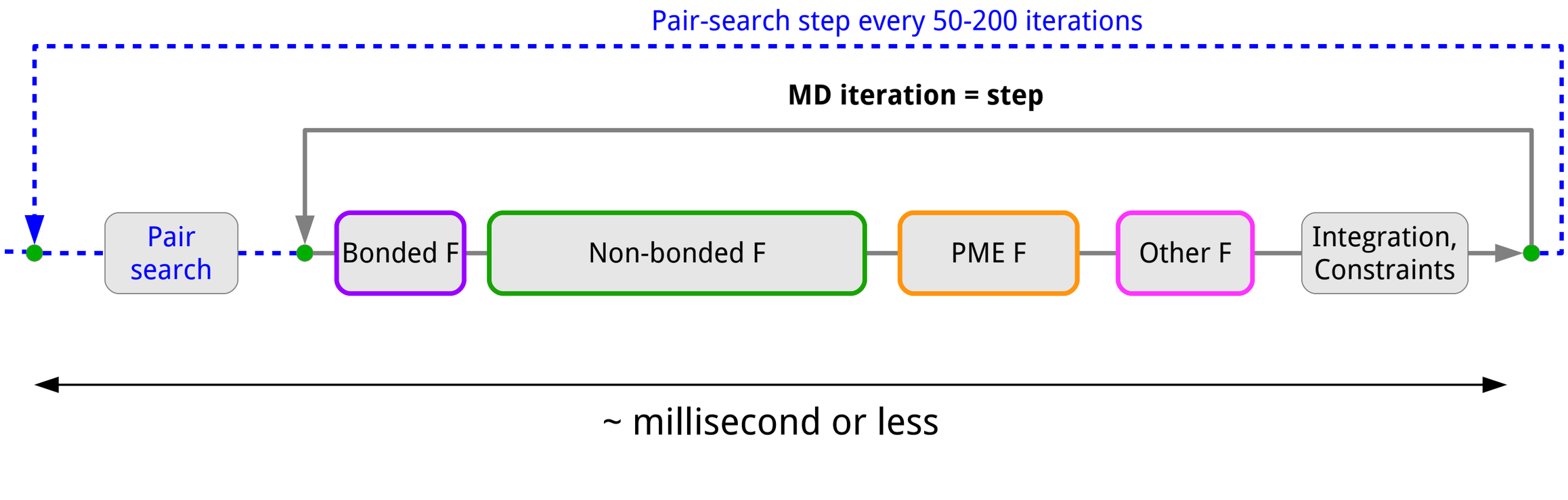
Molecular dynamics step



~ millisecond or less

Goal: do it as fast as possible!

MD: strong scaling challenge



- Simluation vs real-world **time-scale gap**
  - Every simulation: 10^8 –10^{15} steps
  - Every step: 10^6 – 10^9 FLOPs

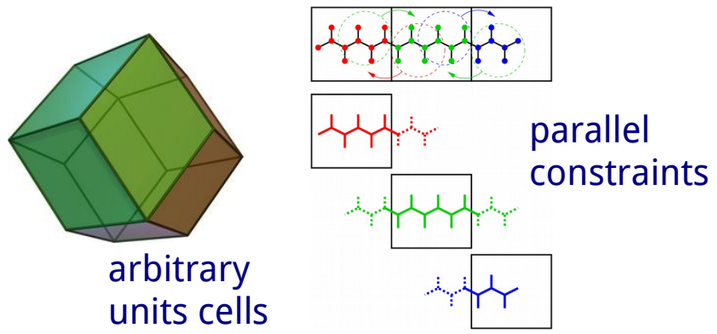
- MD codes at peak: ~**100 µs / step** (single node)
  - <100 atoms/core at peak (running in L1)
  - <10000 atoms / GPU

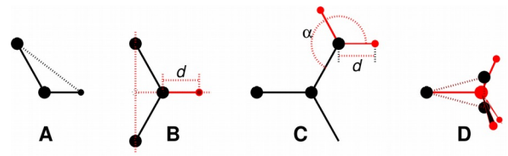
GROMACS

FAST. FLEXIBLE. FREE.

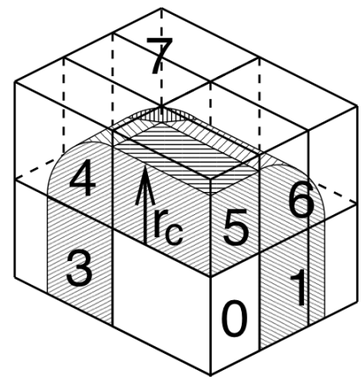
- **Classical MD** code
  - supports all major force-fields
  - broad algorithm support
- **Development**:

  Stockholm Sweden & partners worldwide
- **Large user base**:
  - 10k's academic & industry
  - deployed on most HPC resources

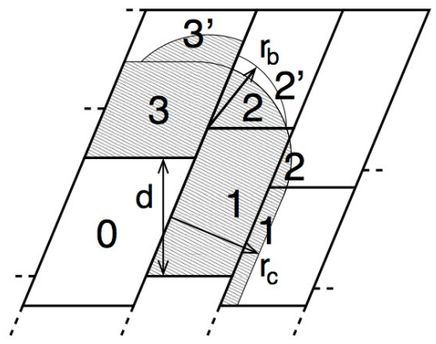




virtual interaction sites



Eighth shell
domain
decomposition



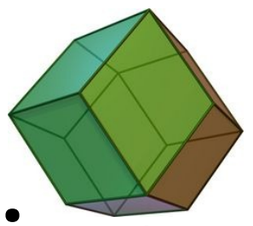
Triclinic unit cell with
load balancing and
staggered cell boundaries

- **Open source**: LGPLv2
- **Open development:**
  - code review & bug-tracker:https://gitlab.com/gromacs

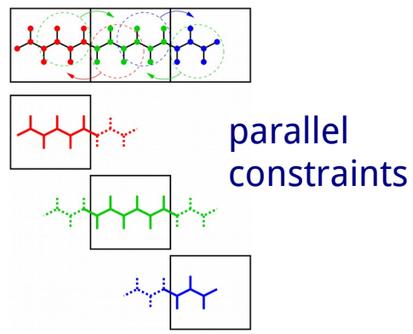
GROMACS
FAST. FLEXIBLE. FREE.

- **Focus on high performance:**

  efficient algorithms & highly-tuned parallel code

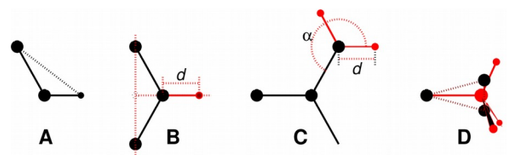


arbitrary
units cells



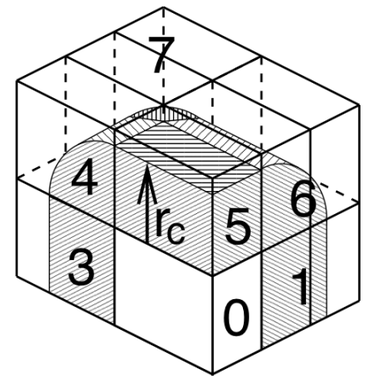
parallel
constraints

- **Bottom-up performance oriented design:**
  - absolute performance over “just scaling”

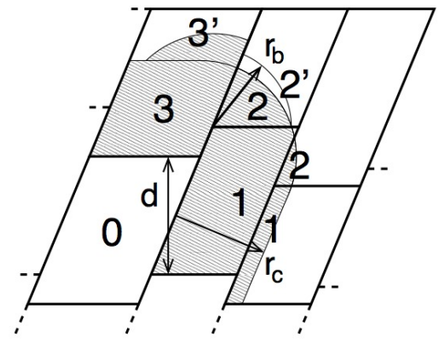


virtual interaction sites

- **Portability**
  - broad CI testing, Linux distro integration
  - regular testing on all HPC arch



Eighth shell
domain
decomposition



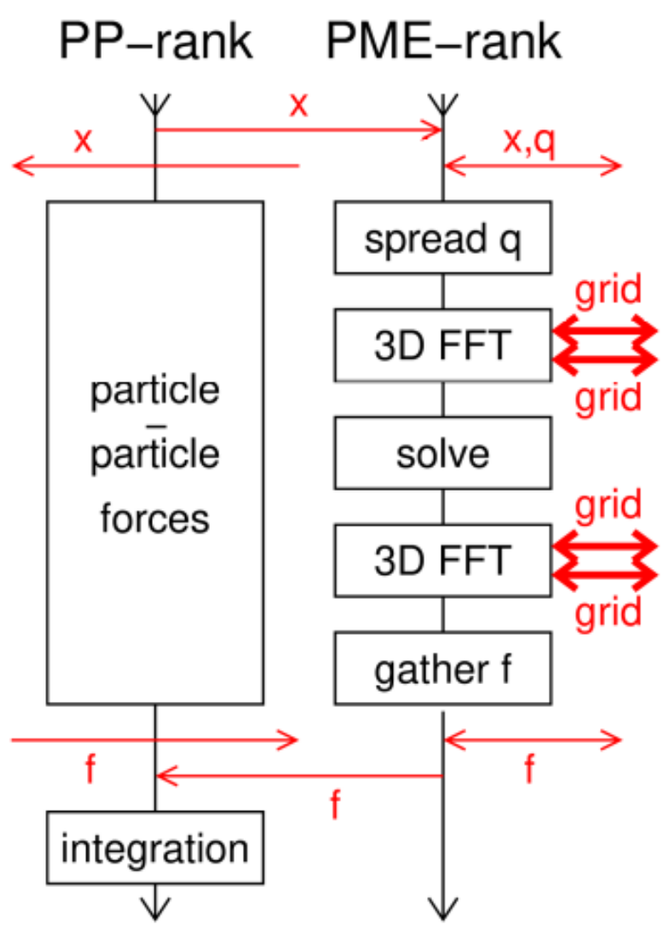
Triclinic unit cell with
load balancing and
staggered cell boundaries

- **Code-base:** C++17, >1M LOC

GROMACS parallelization

Parallelism exploited on **multiple levels**:

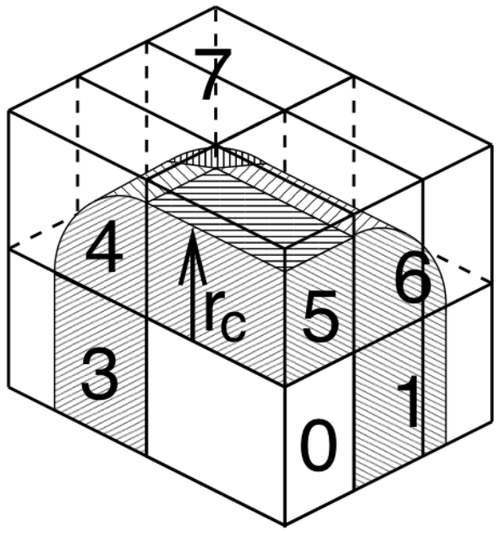
SIMD / threading / NUMA / async offload / MPI

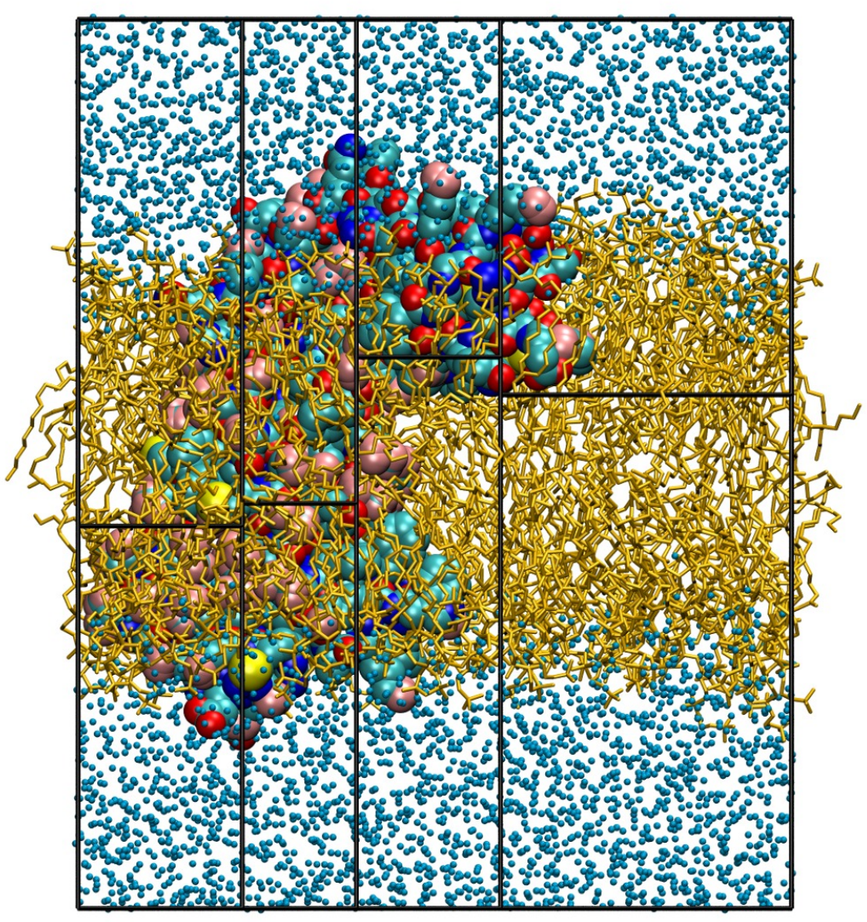


- **Hierarchical** parallelization**:**

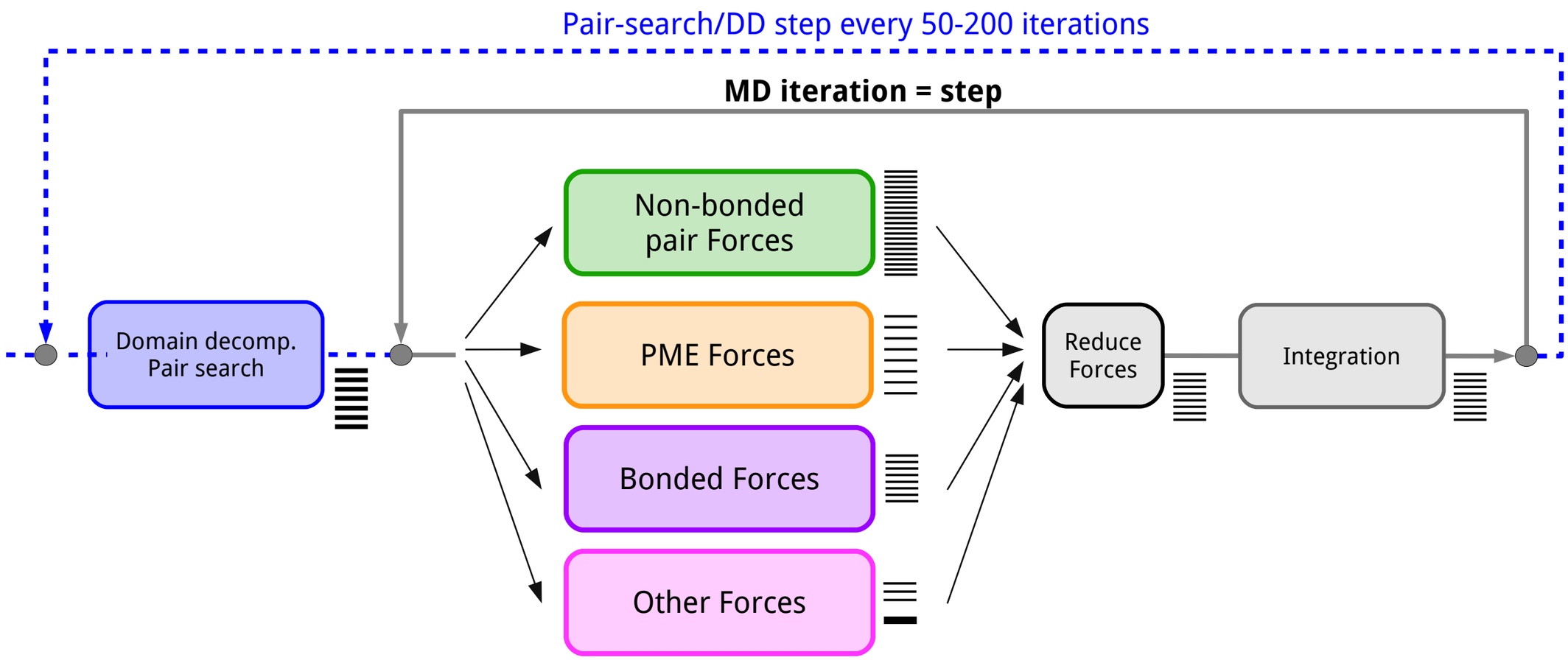
  target each level of hw parallelism

  - MPI: SPMD / MPMD; thread-MPI
  - OpenMP
  - SIMD: 14 flavors (SIMD library abstraction)
  - CUDA, OpenCL

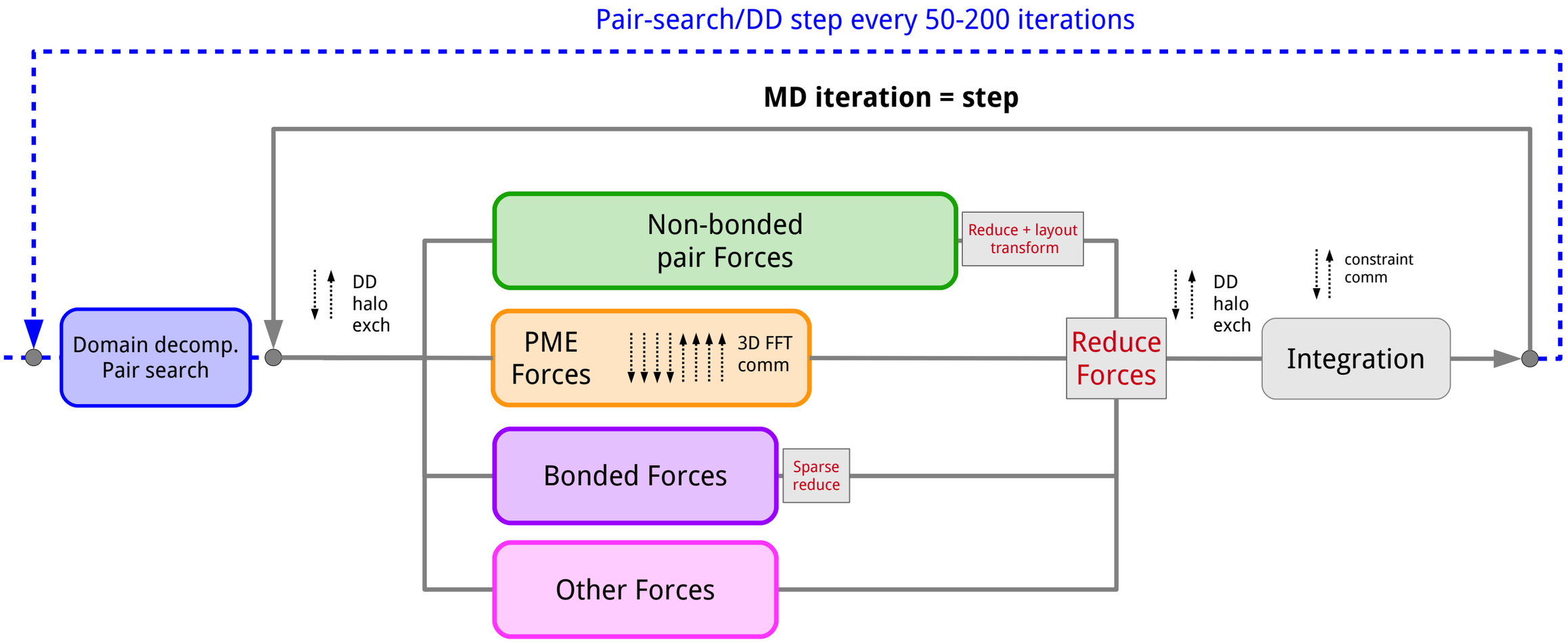




Concurrency in the MD step



Parallelization of the MD step



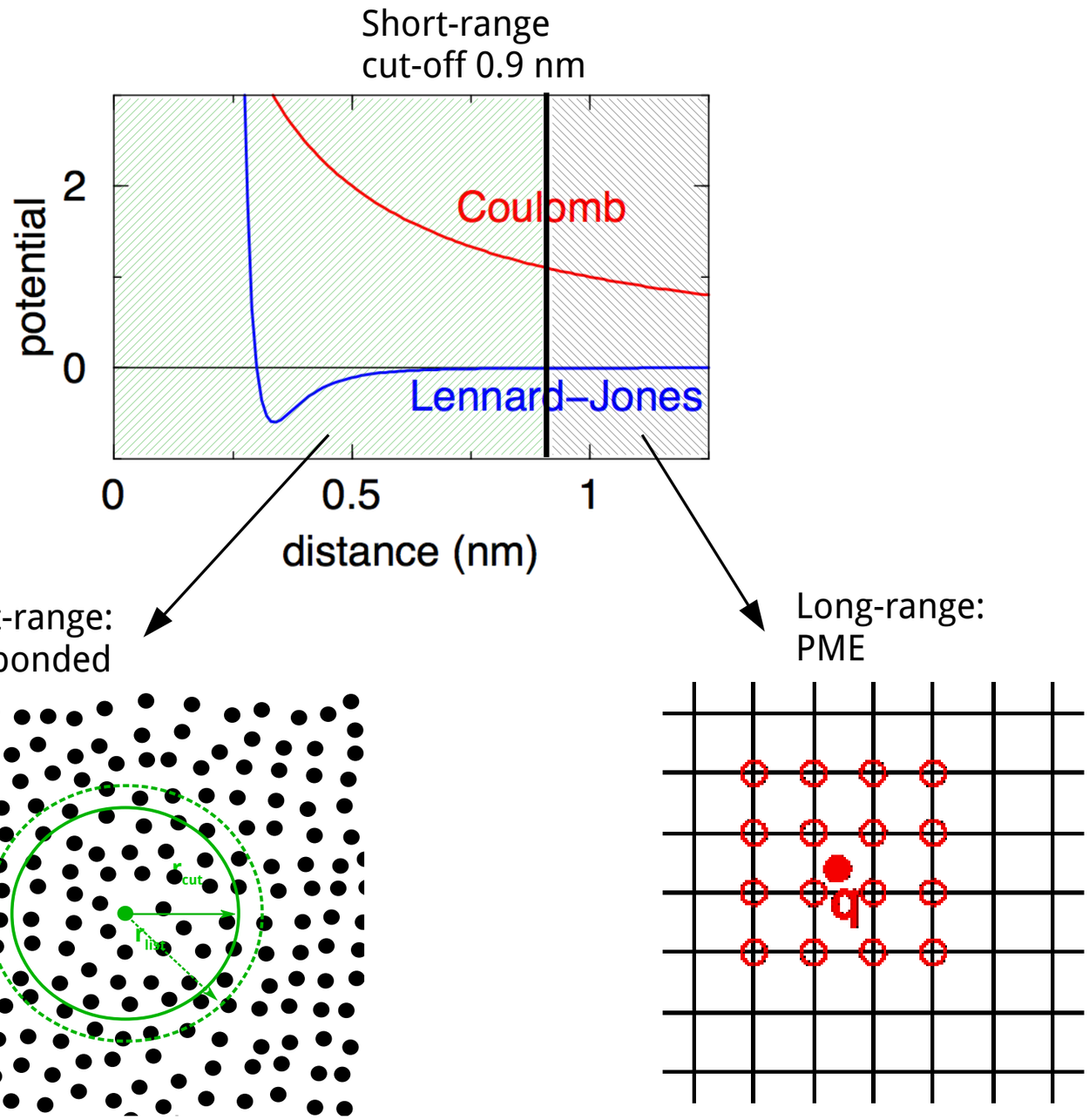
Non-bonded interactions

- Lennard-Jones:
  - decays fast, use cutoff
- Coulomb
  - decays slowly, need long range
- Short-range:

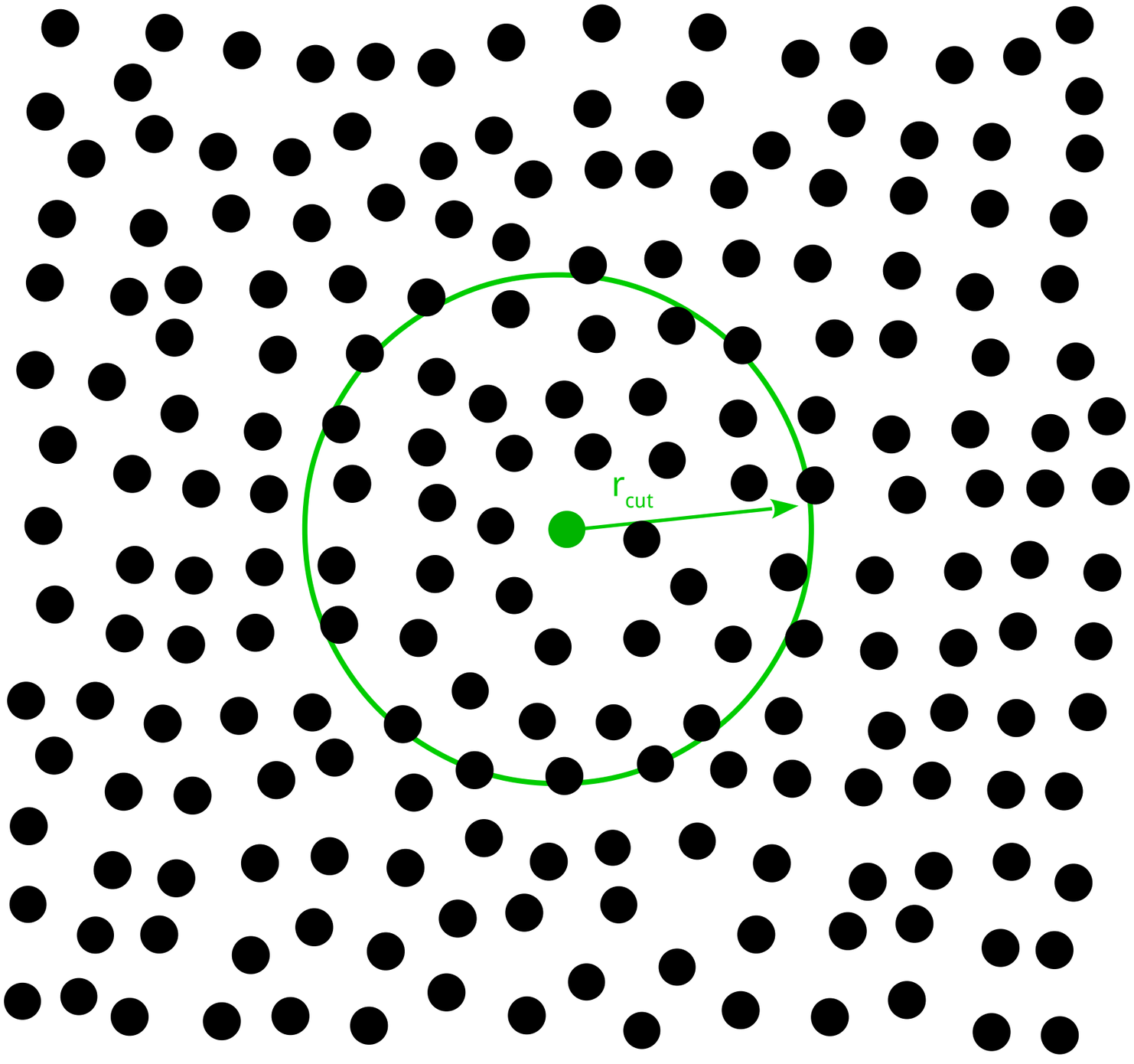
  calculate all interactions

  within a spherical cutoff
- Long range:

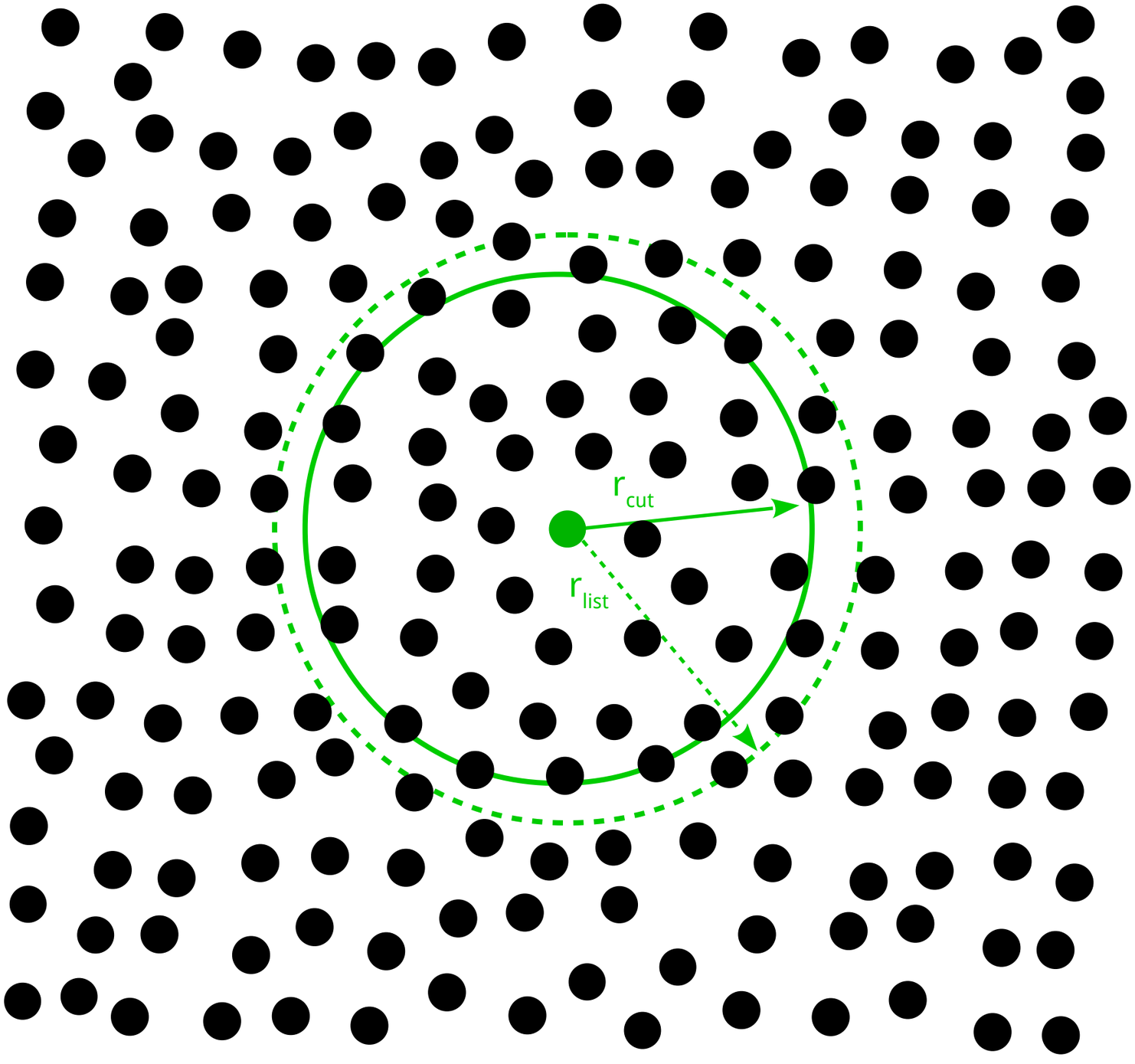
  PME grid-based solver using FFT



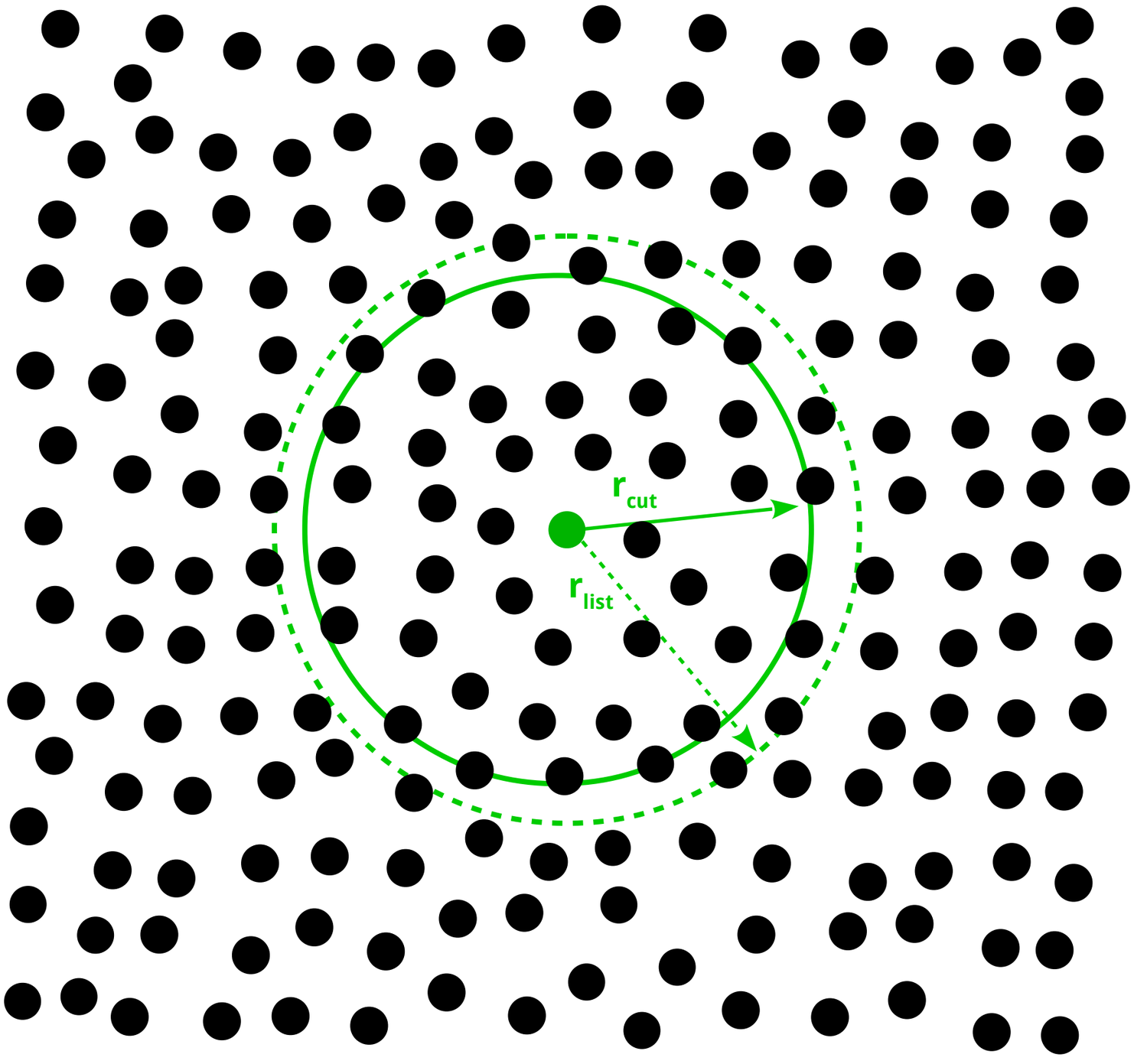
Pair interactions



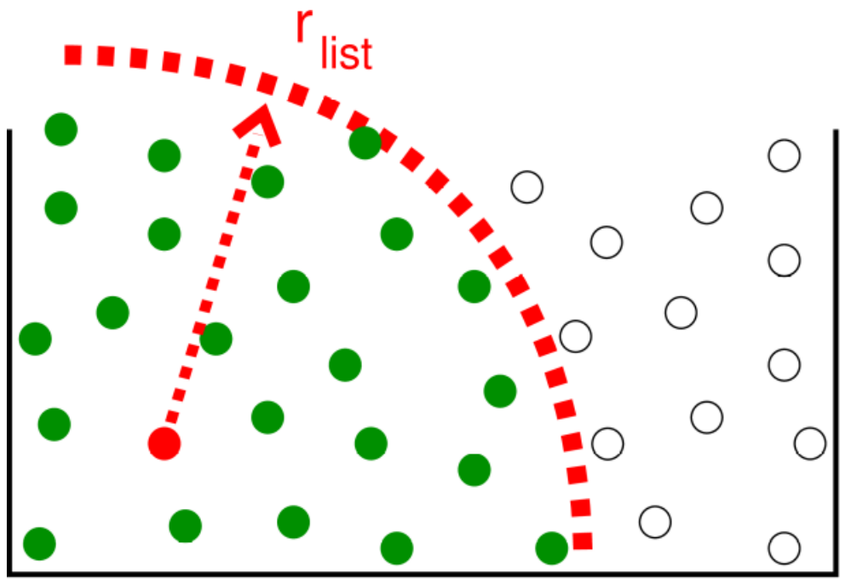
Pair interactions: Verlet algorithm

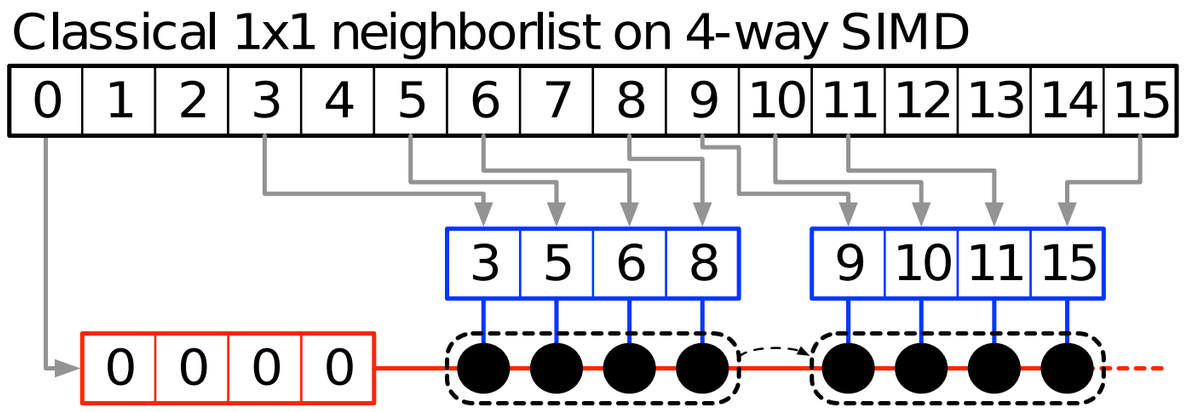


Pair interactions: Verlet algorithm



Pair force calculation: SIMD-parallel traditional algorithm





Traditional algorithm:
cache pressure, ill data reuse, register shuffle bound

Neighbor list = particle pair list:

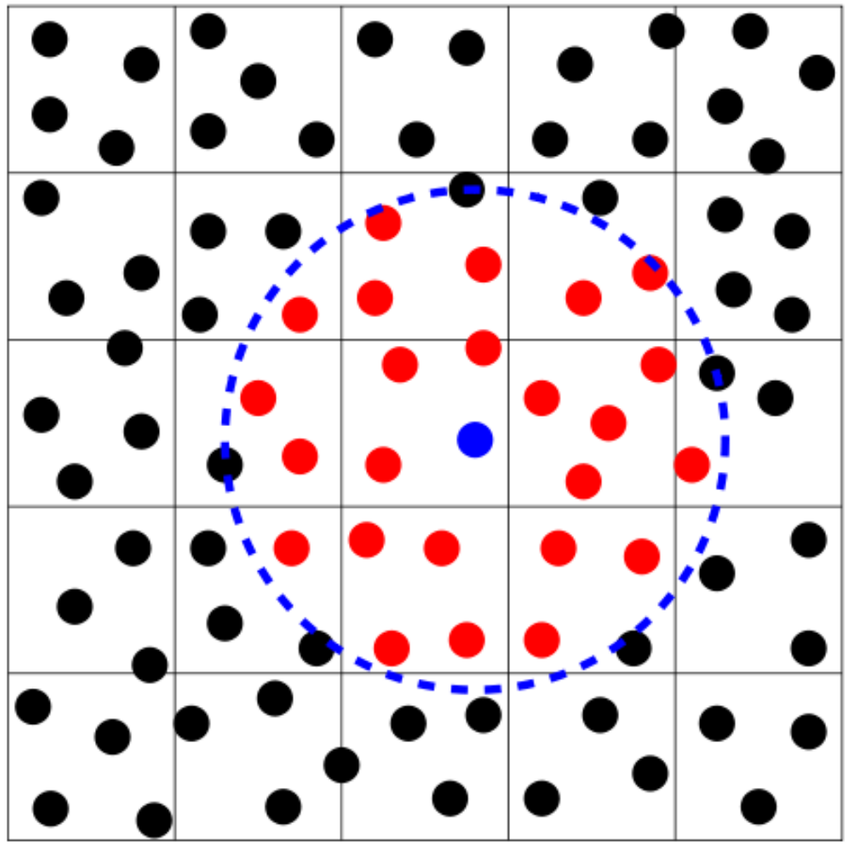
i-atom j-atoms in range = neighbors

- **irregular data:** non-contiguous layout
- **need to shuffle particles** in registers
- **low data reuse**:
  for each loadeded i-atom compute only once

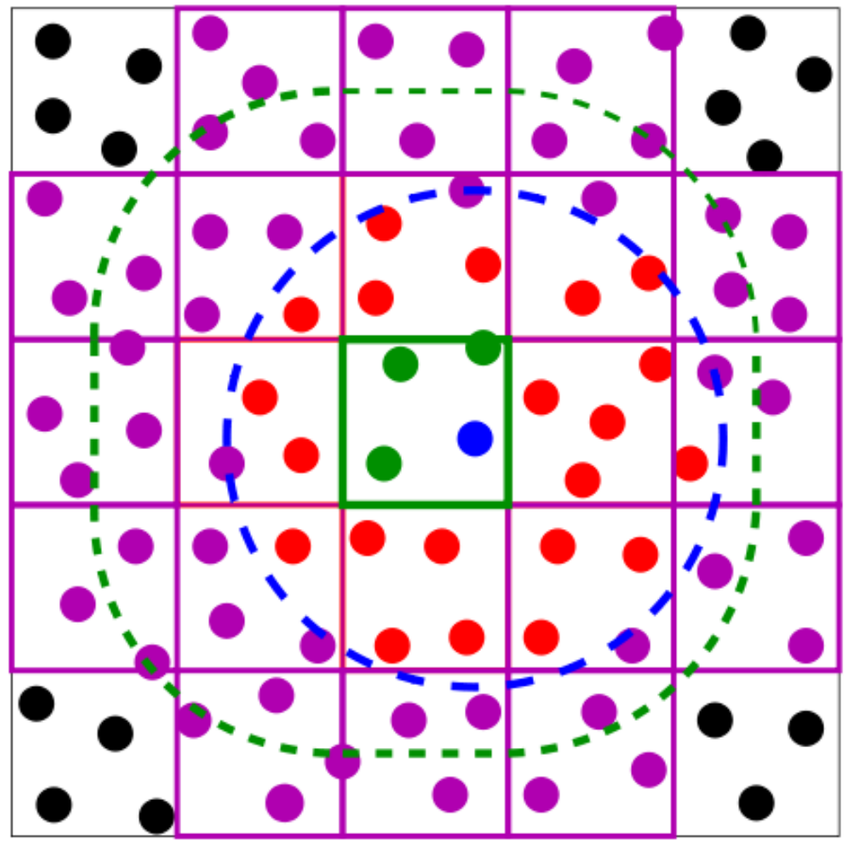
Bad for SIMD!

Calculating zeros: serial vs SIMD

serial algoritm: no 0s



SIMD-style algoritm: **lots of 0s**



Source of zeros:

- need many squares/cubes to cover a sphere
- extra particles considered: cells crossed by the cut-off sphere

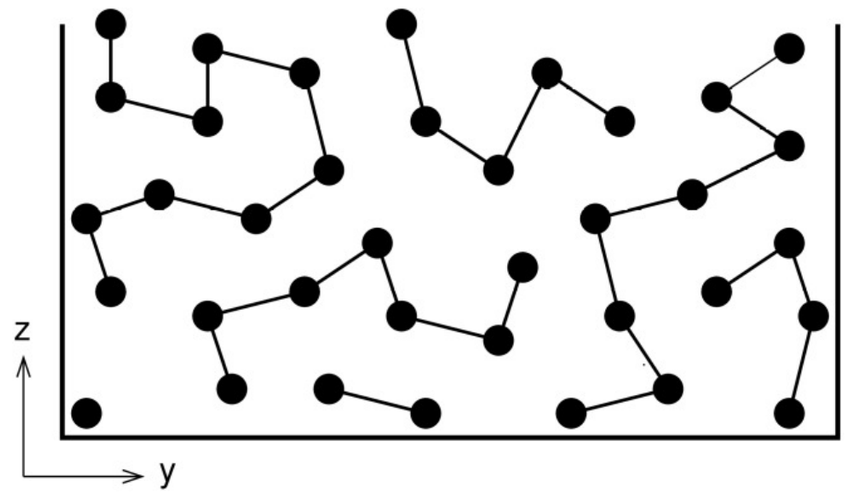
A better pair-interaction algorithm

design objectves

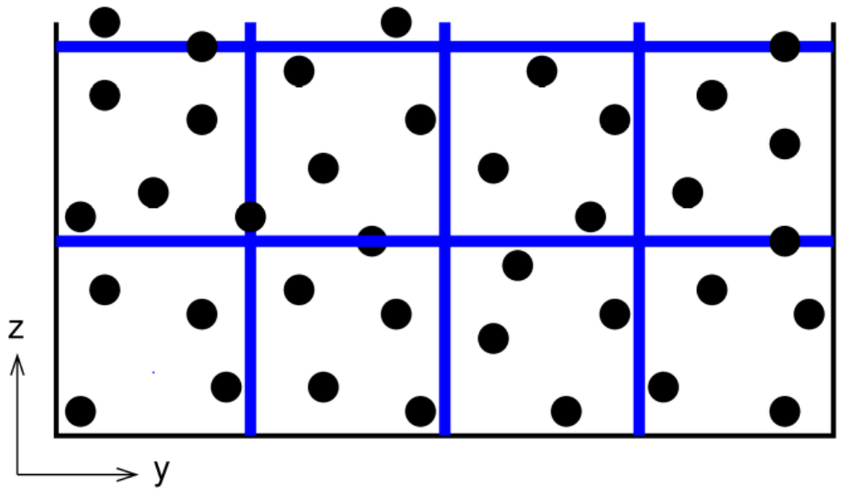
- **Need:**
  - **Data reuse**: inner loop should calculate multiple interactions without having to load
  - Calculate **as few 0s** as possible
  - Fixed pair list update interval
- **Nice to have:**
  - **future-proof** (can we predict the future?)
  - **one algorithm, many architectures**: narrow & wide SIMD, accelerators, FPGAs,...
- **Keep in mind:**
  - Flops are “free”
  - Data movement is expensive

Regularize the problem:
cluster algorithm

grid cell **uniform in particle count** rather than dimensions!

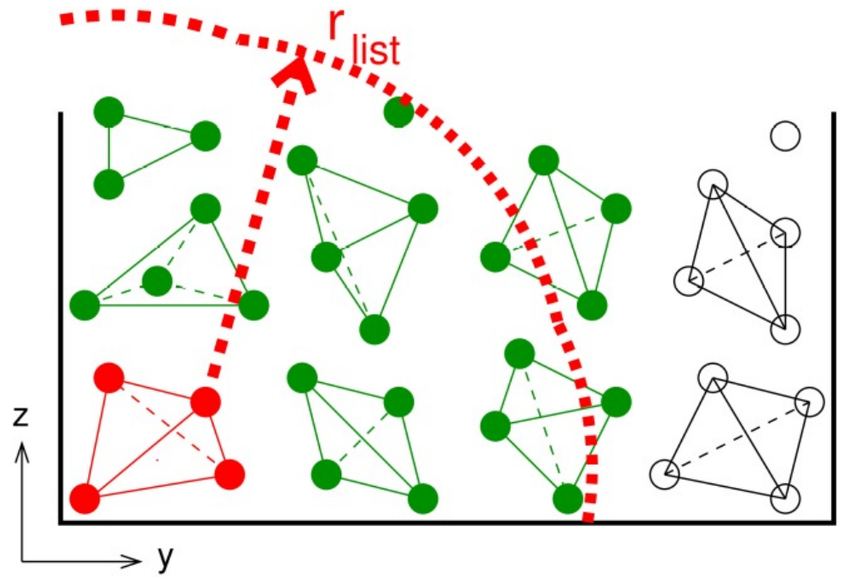


x,y,z gridding

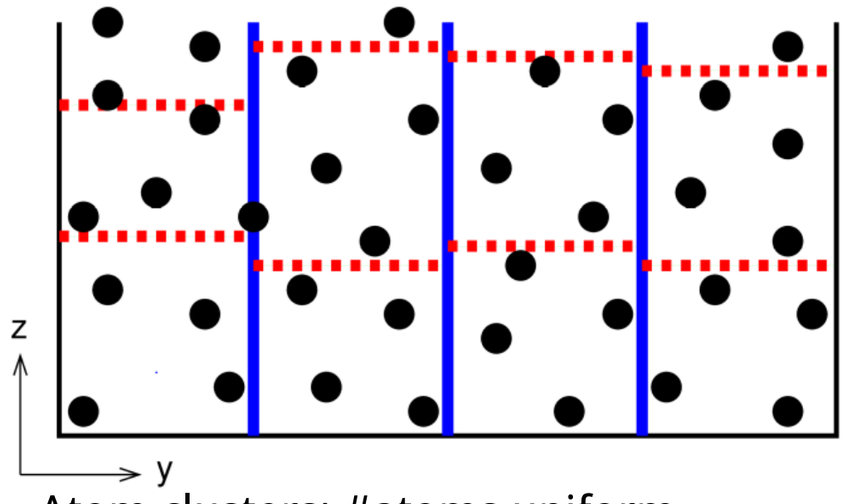


Standard cell grid: spatially uniform

x, y gridding
z sorting
z binning

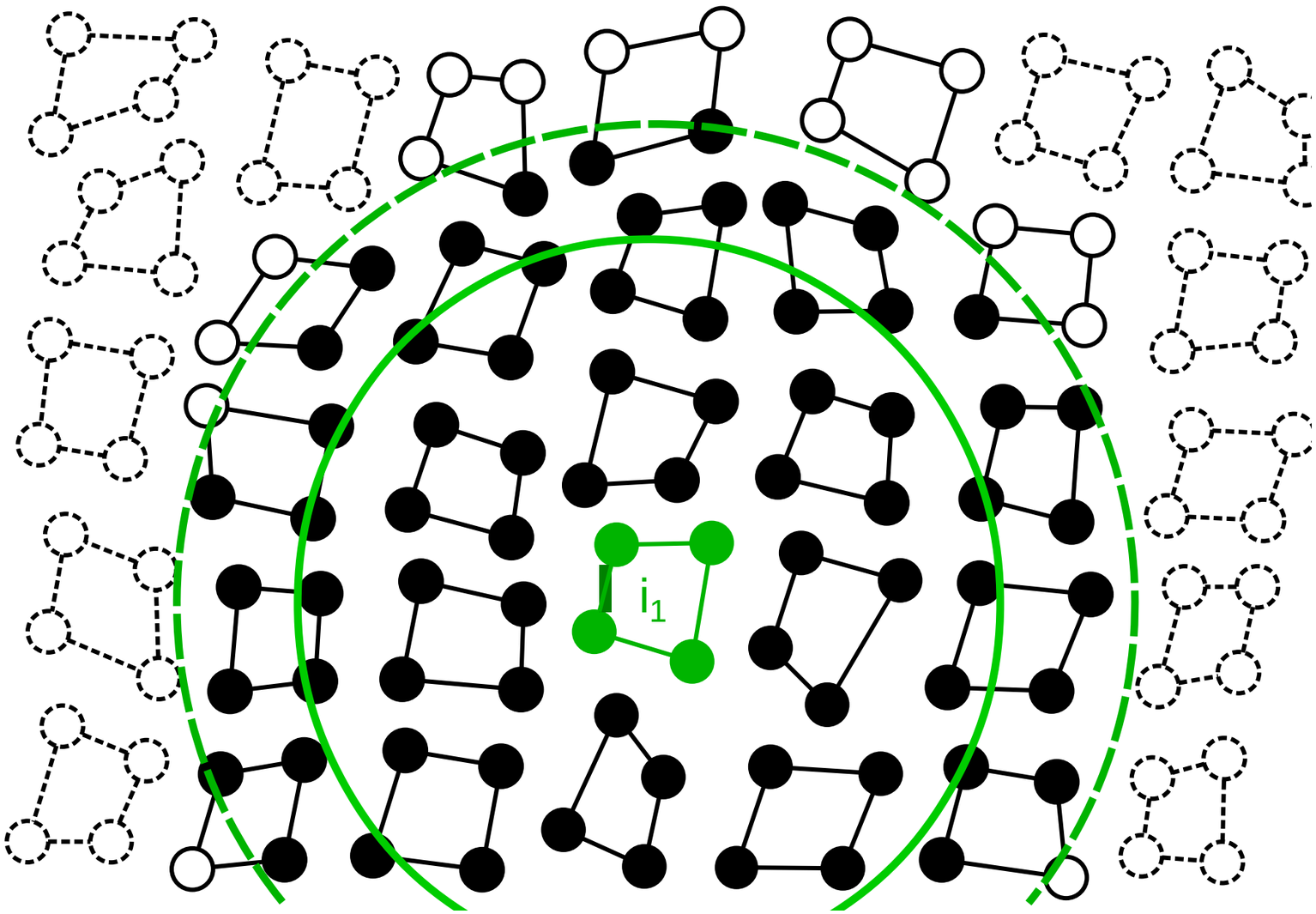


make
cluster pair-list

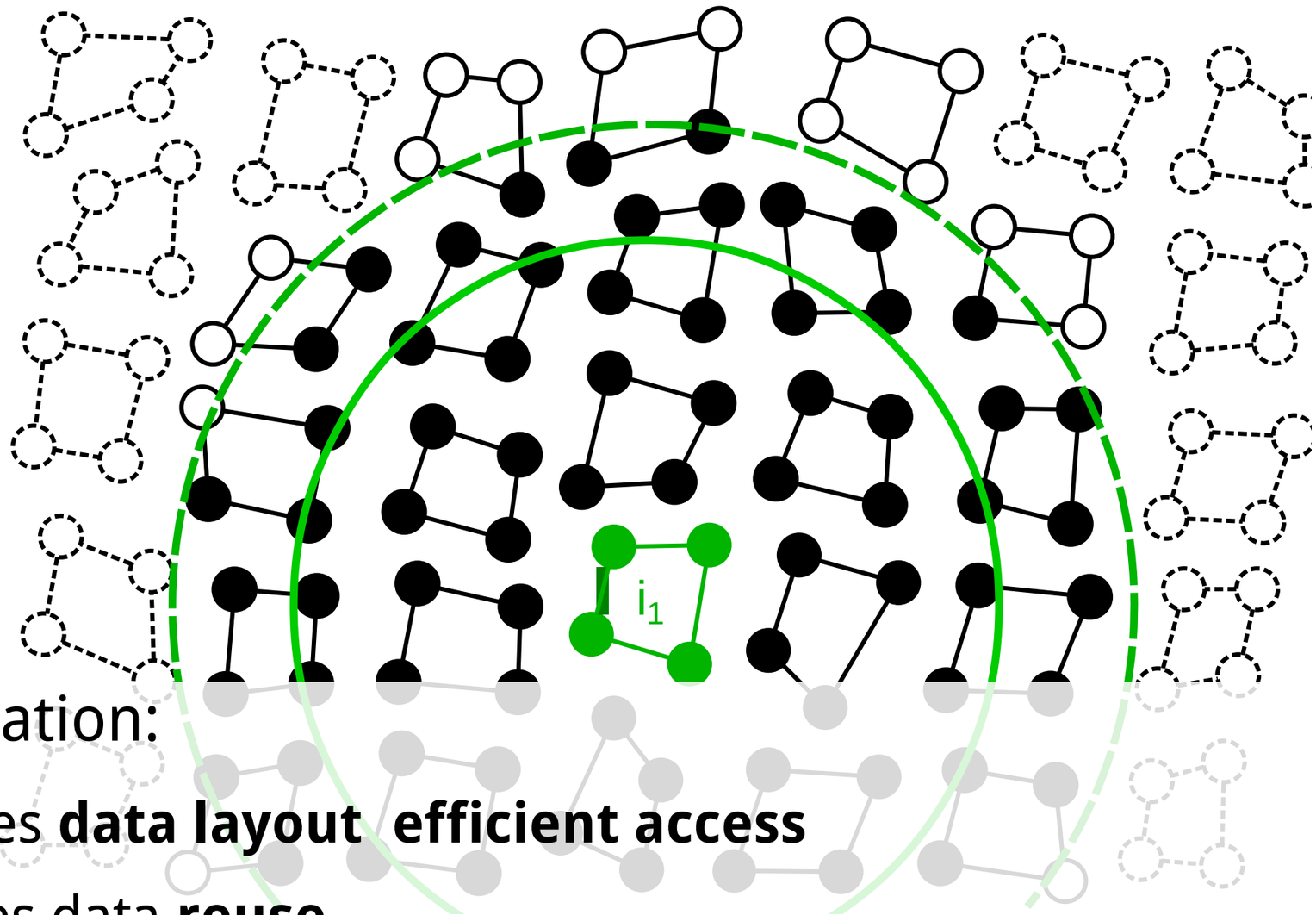


Atom clusters: #atoms uniform

Cluster algorithm



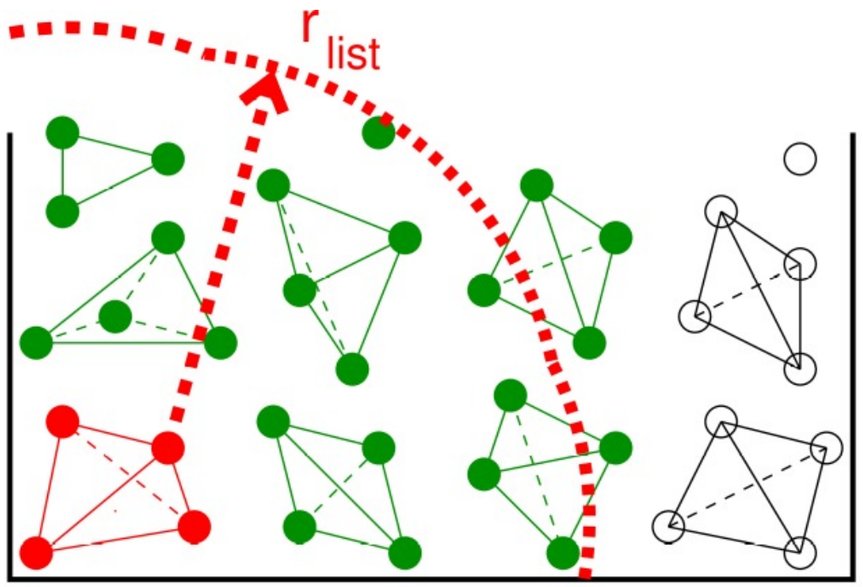
Cluster algorithm

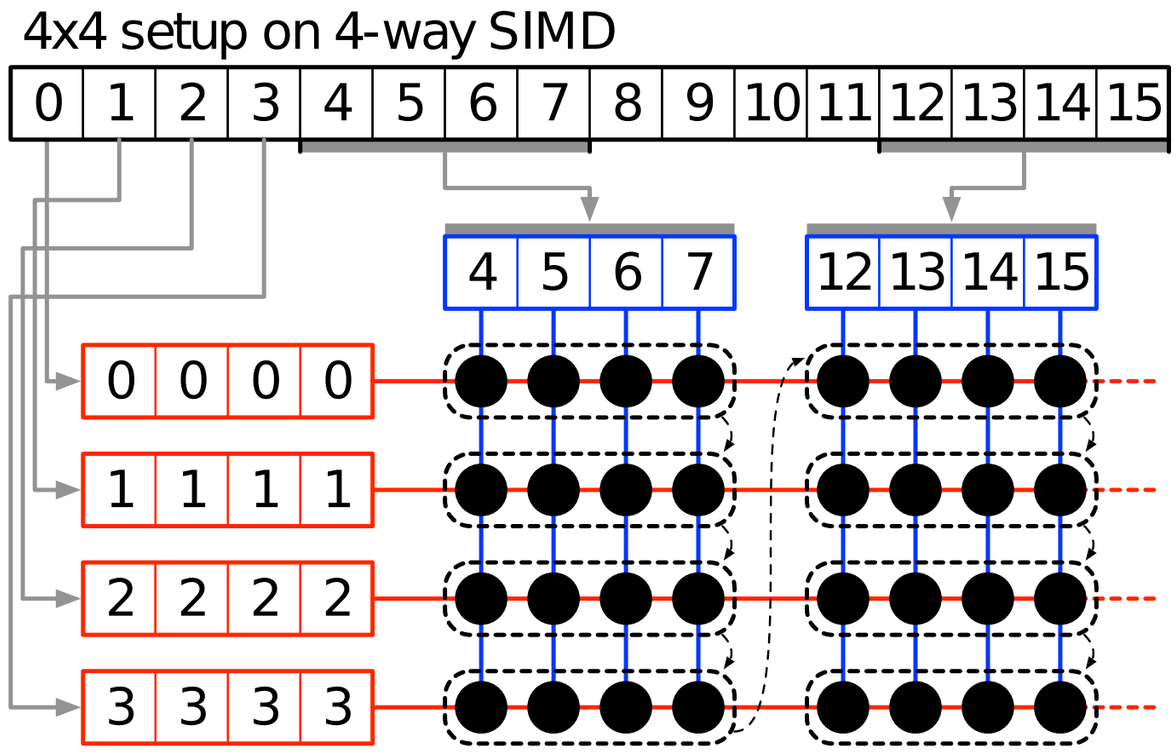


- Regularization:
  - optimizes **data layout efficient access**
  - increases data **reuse**
- Flexible & adaptable:

  SIMD width & reuse factor algorithmic parameters

Pair force calculation: SIMD-parallel cluster algorithm





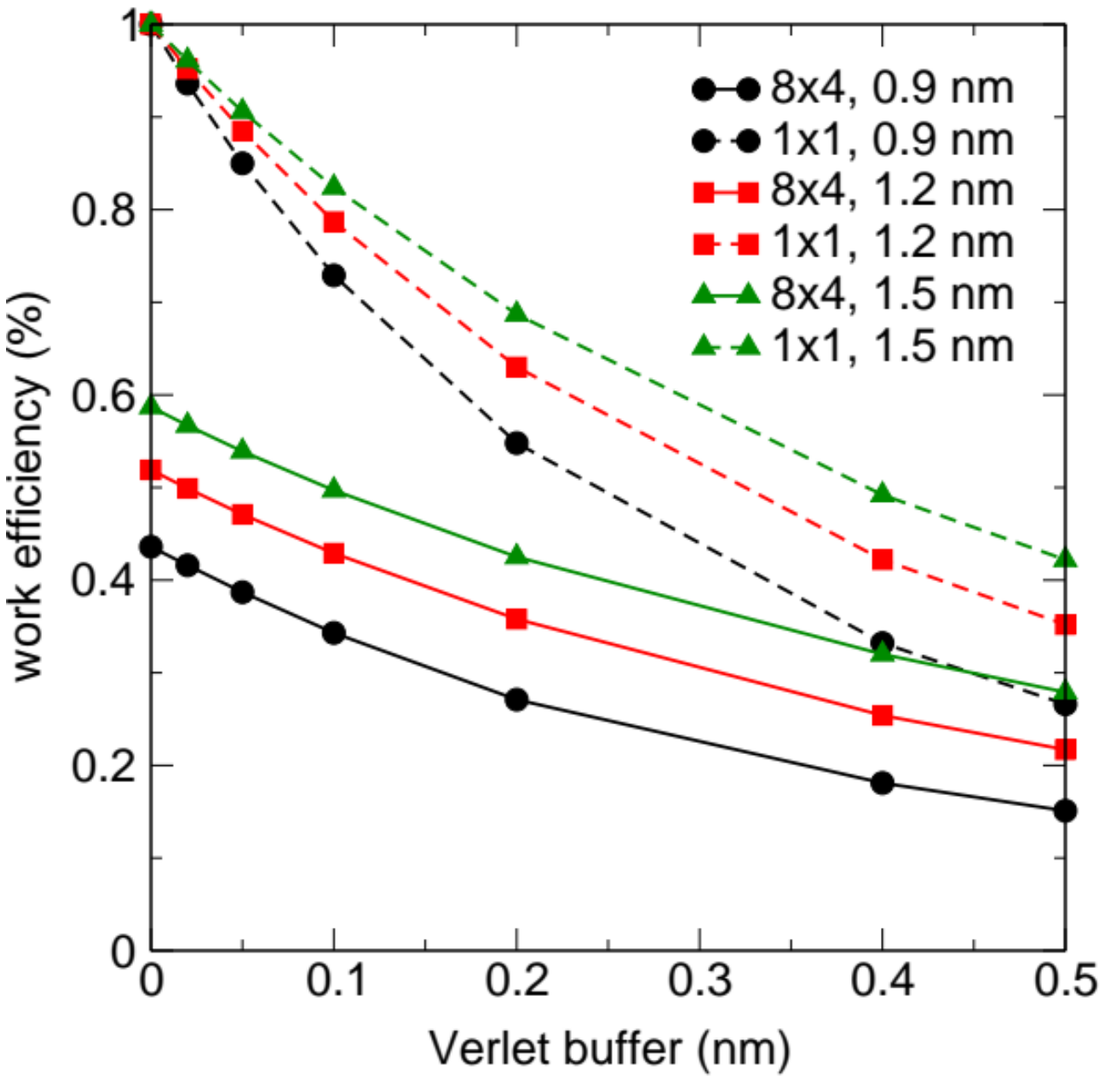
Cluster algorithm on 128-bit SIMD SP (SSE4, VMX, 128-bit AVX): cache friendly, 4-way j-reuse

Cluster sizes are the “knobs” to adjust for a specific arch:

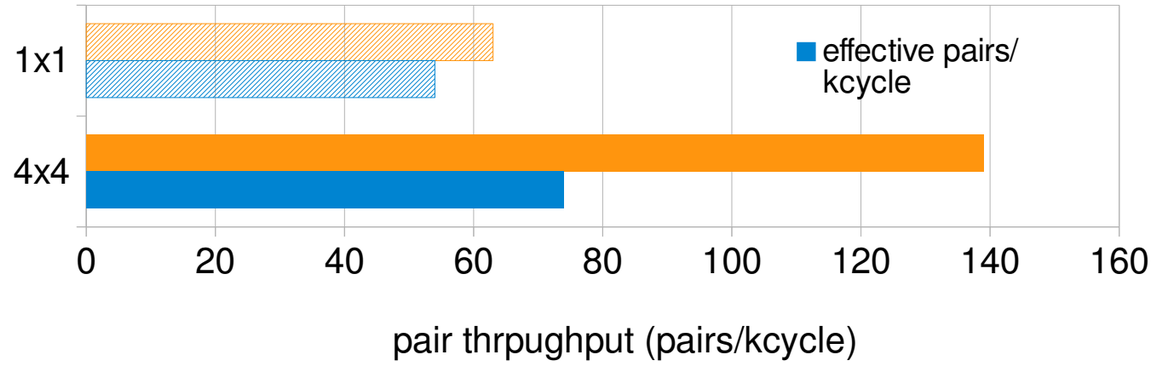
- j-size: adjust to SIMD width
- i-reuse: data reuse & cache pressure

  => allows restoring the algorithm’s arithmetic intensity

Parallel work-efficinency

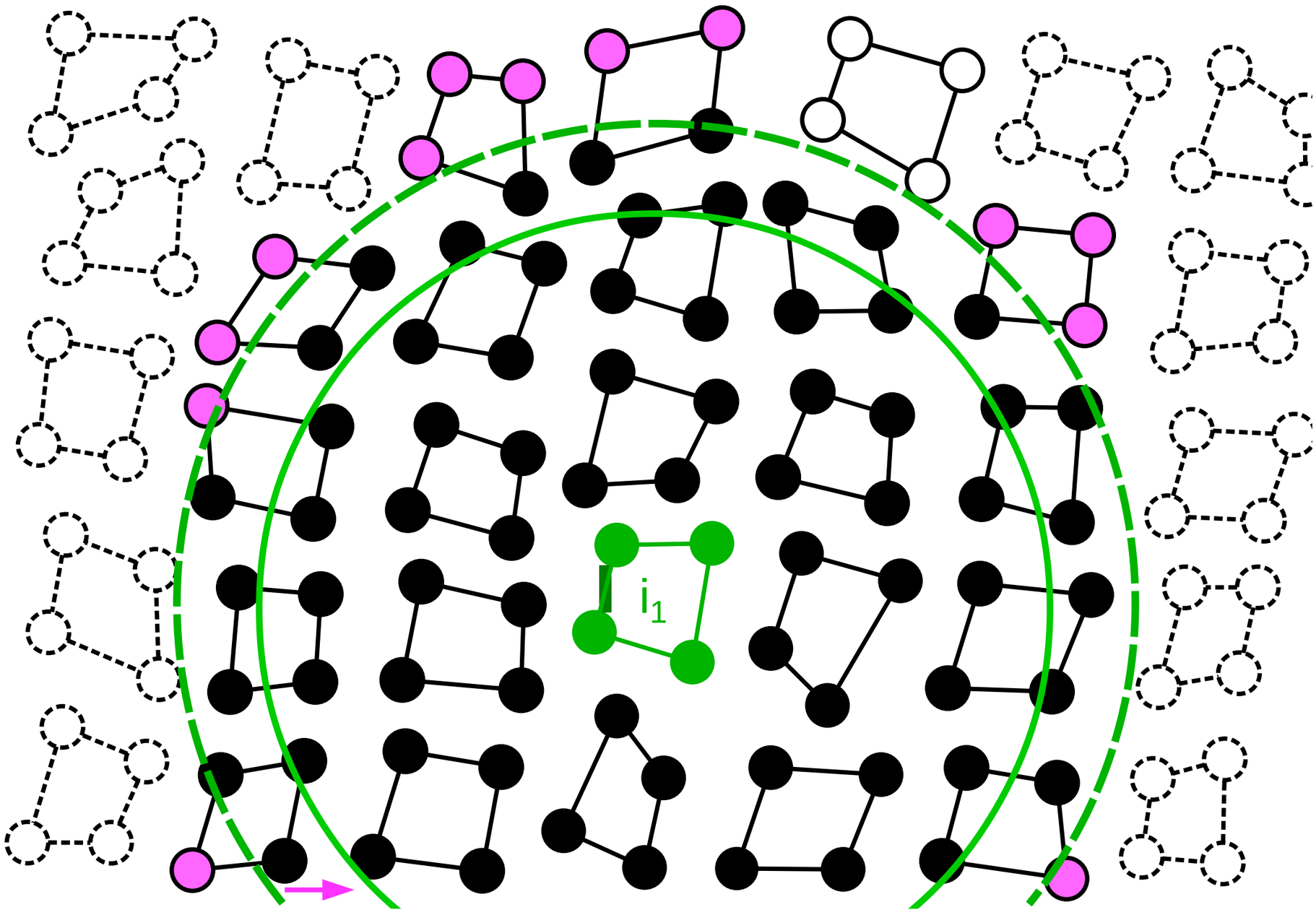


- always >= 1 for a SIMD-optmimzed implementation
- benefits outweigh the cost



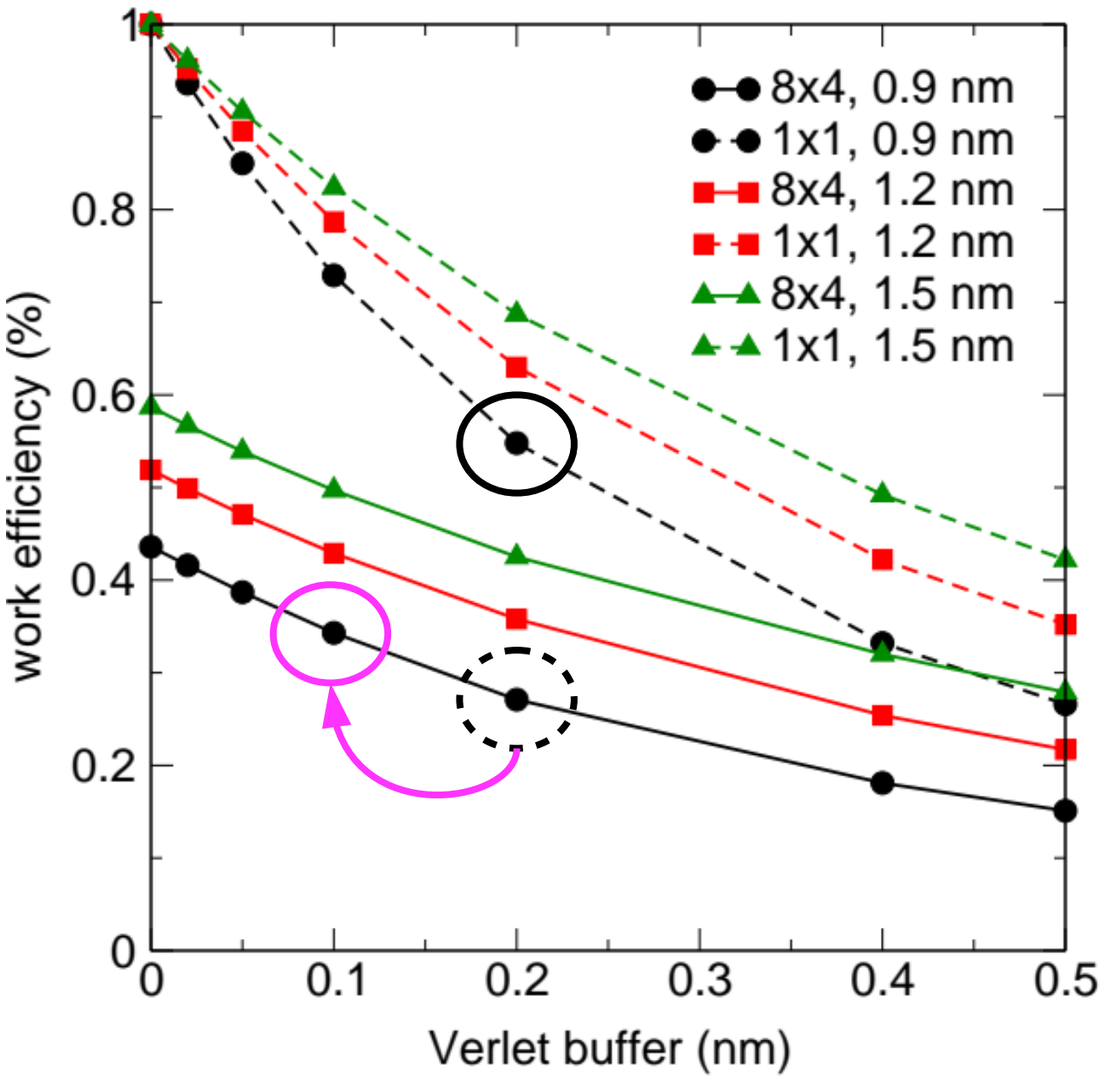
Cluster algorithm: implicit interaction buffer

Extra particles "sticking out" of cut-off: **implicit interaction buffer**



Explicit interaction buffer: can be reduced

Parallel work-efficinency revisited



- Making use of the implicit buffer

  gives a parallel efficiency gain:

  => **shorter explicit buffer** with the cluster

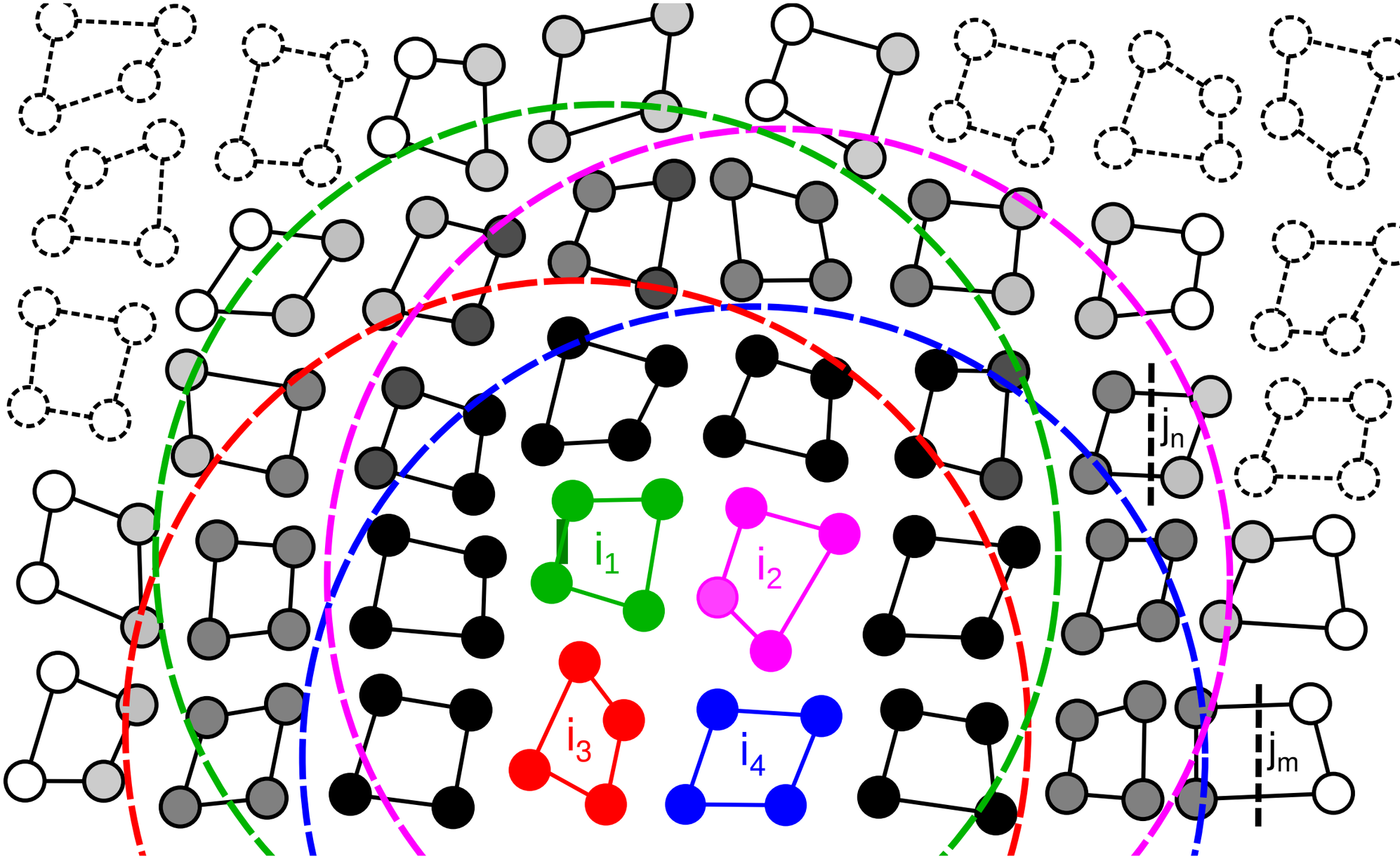
Example:

4x8 clusters allow

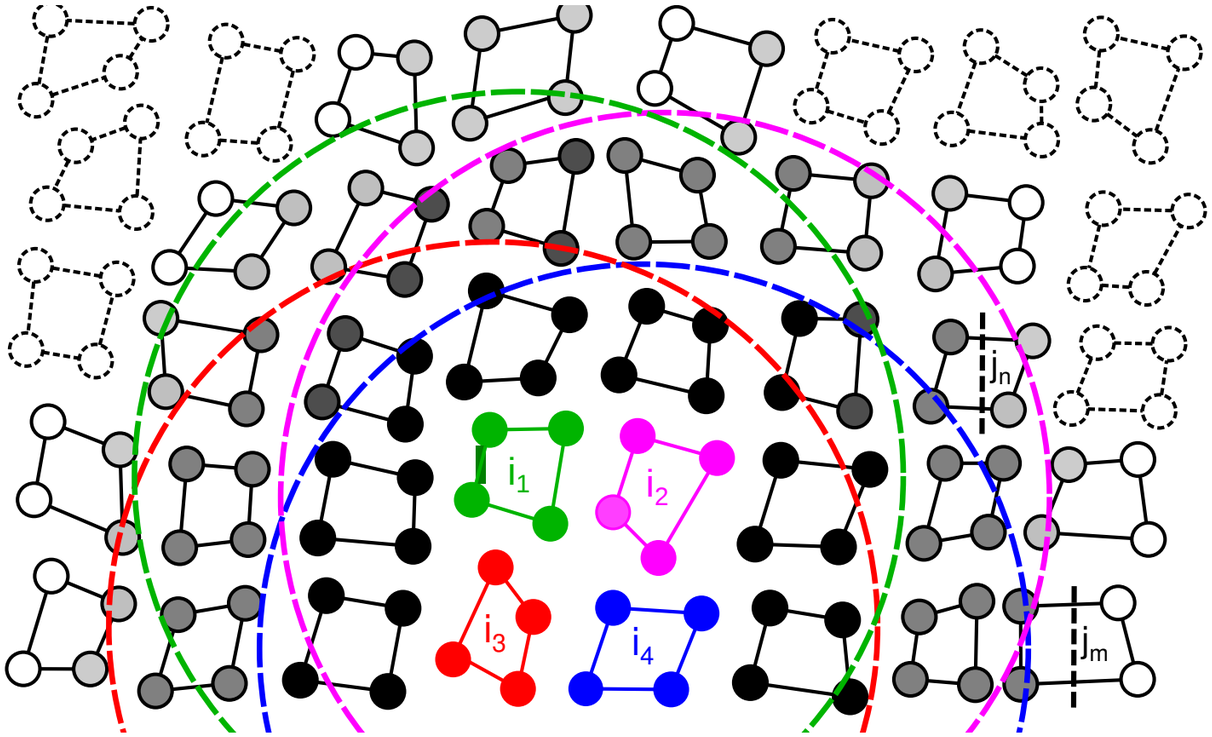
0.1 nm instead of 0.2 nm

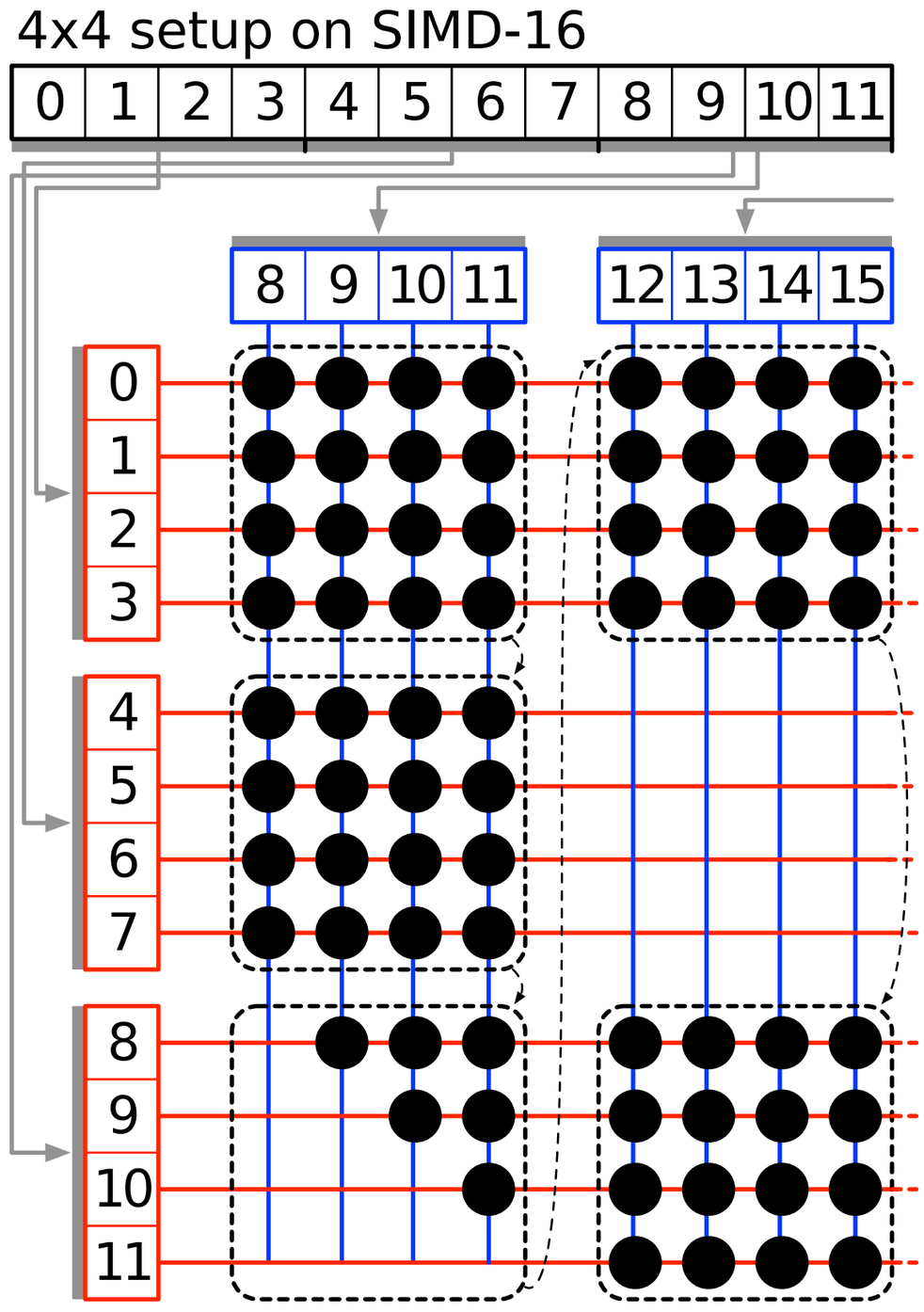
(standard water-box with PME and 0.9 nm cutoff 40 steps list rebuild)

GPU super-cluster setup: avoid large tiling



Super-cluster setup for GPUs





Wider execution: use **hierarchical pair list to avoid work-efficiency cliff**

- joint list for multiple i-clusters
- swap i-j loop order

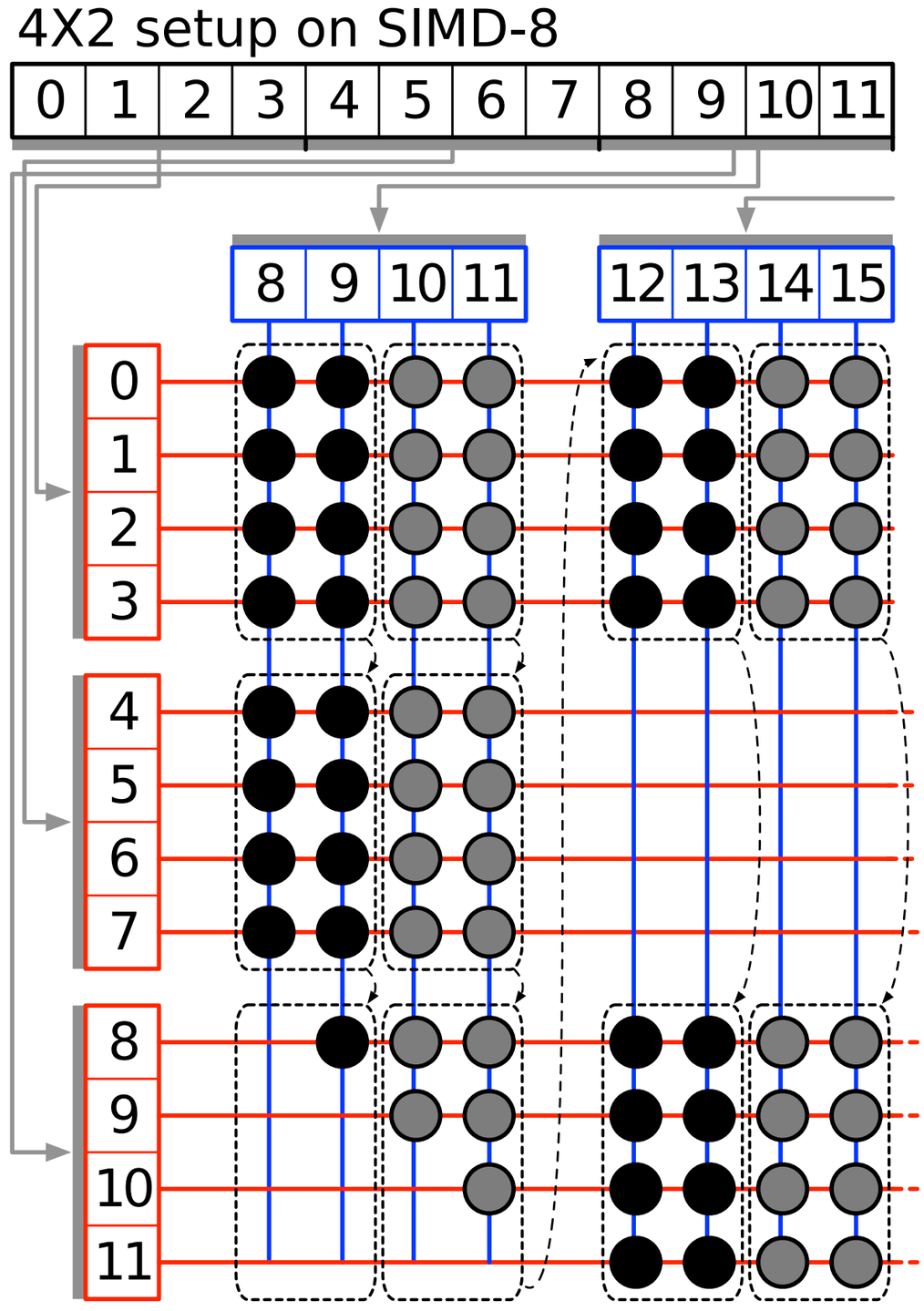
Cluster algorithm adaptability to hardware

- Flavors:
  - NVIDIA 32-wide 8x4
  - AMD 64-wide 8x8
  - Intel 8-wide 4x2

- Cluster-setup:

  search 4x4, pruned to 4x2 for force

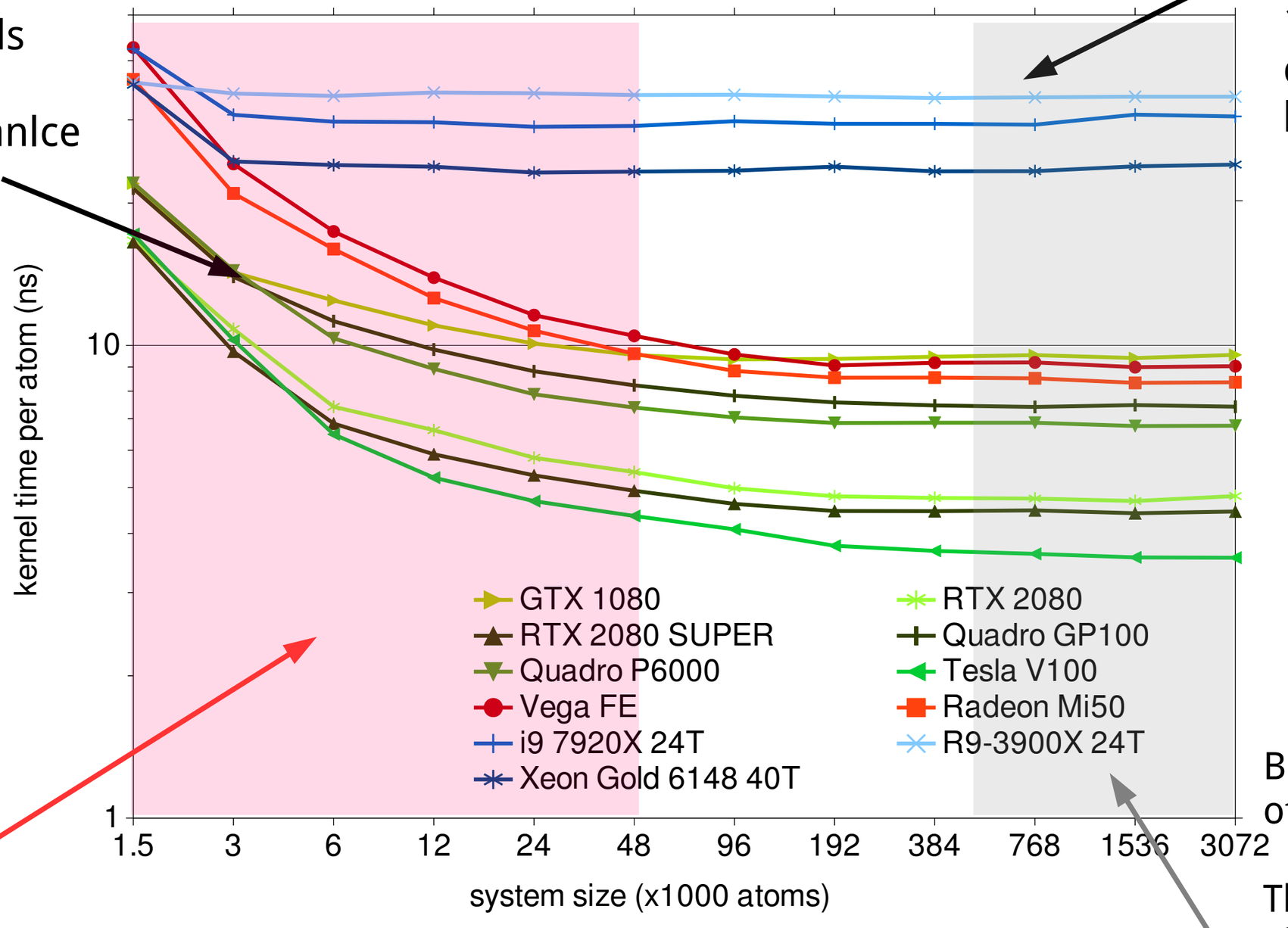
  (parallel work efficiency advantage)



Pair interaction kernel throughput

GPUs very sensitive to input size: fixed overheads kernel startup SM load imabanlce

CPUs insensitive to input size to 100s atoms/core cache effects at large inputs

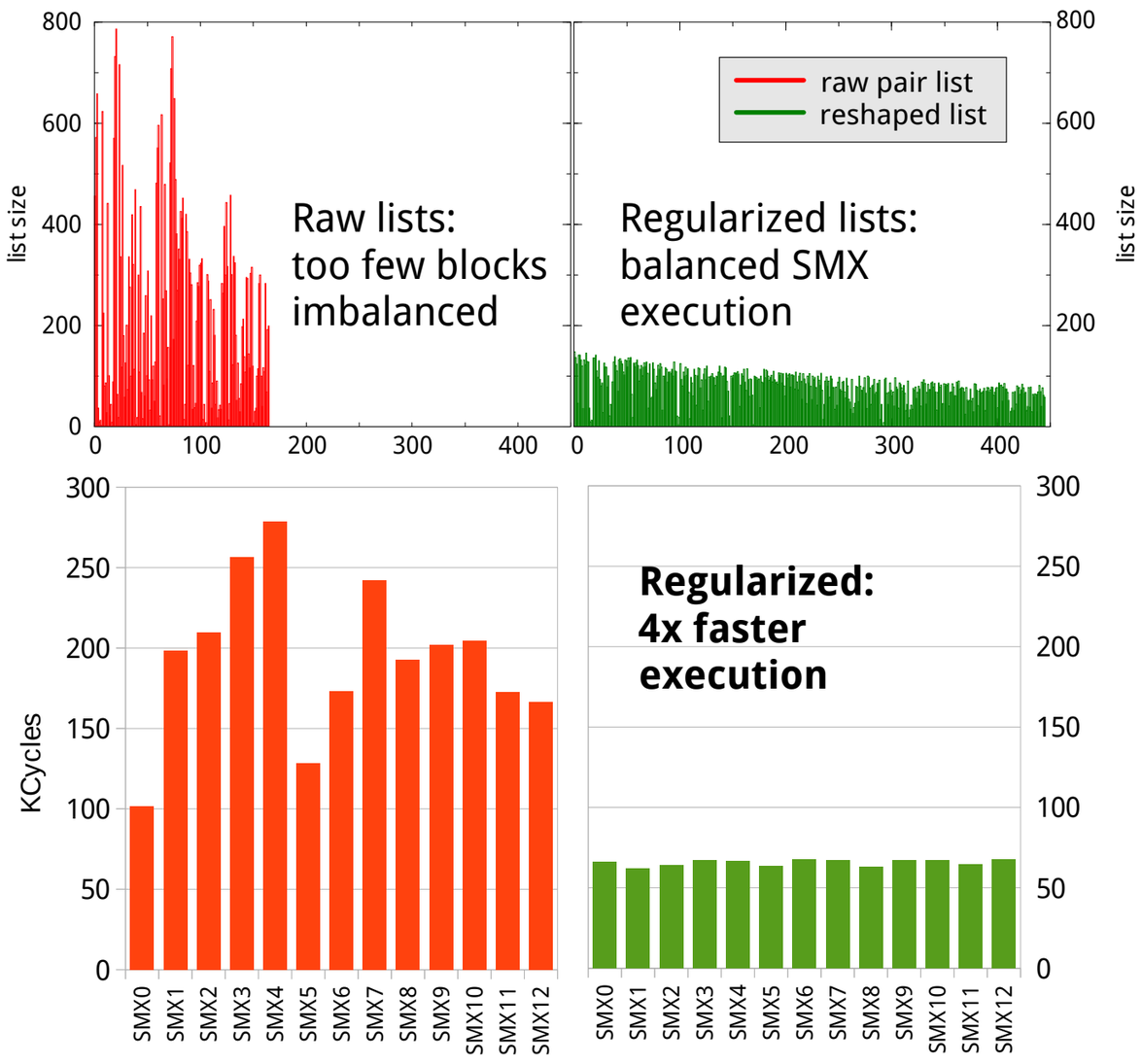


Strong scaling regime: **where most of our efforts go!**

Benchmark "show-off" regime:

This is where the **"free lunch" from new hardware** comes in full effect

Intra-GPU load balance



Workload per SMX: Tesla K20c, 1500 atoms

- GPU scheduling black-box
- Reshape irregular workload:
  - sort and split lists
  - improve load balance
  - avoid kernel tail effects

Heterogeneous Parallelization

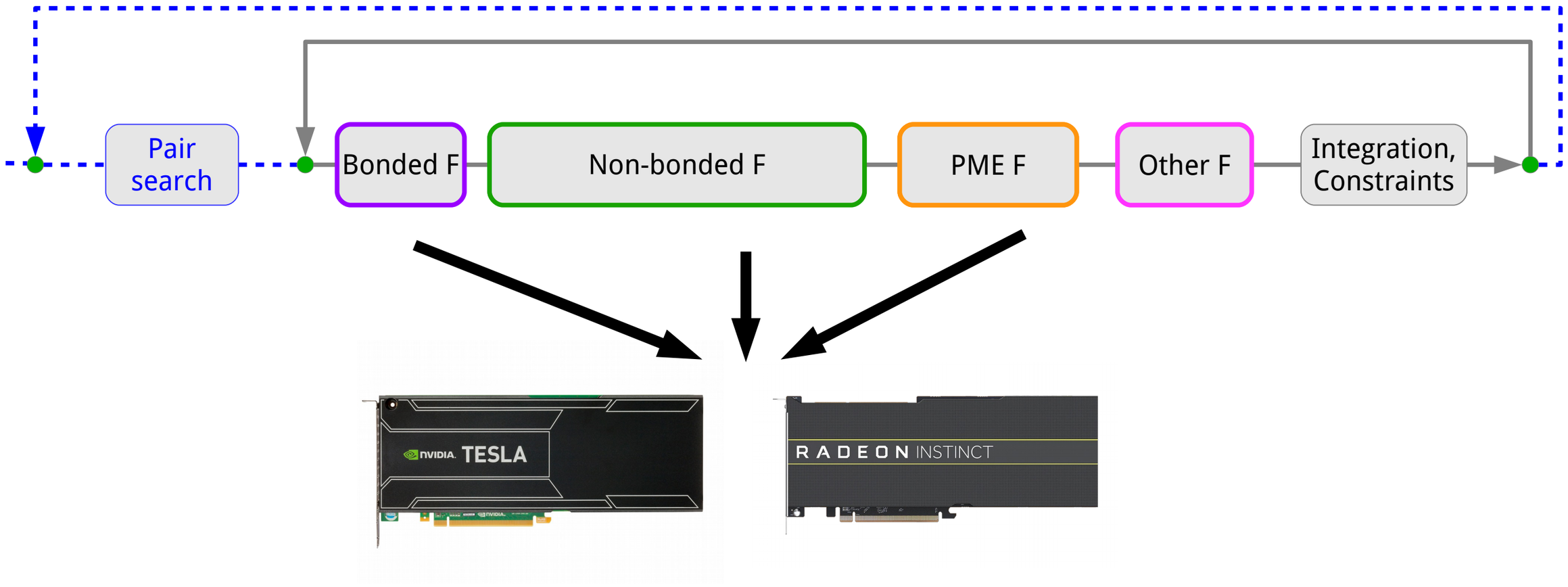
Heterogeneous offload design

- **Heterogeneous** vs homegeneous GPU offload
  - full port to multiople toolkits/APIs not an option for a large codebase

- Heterogeneity: both CPU and GPU
  - maintain the **versatility and feature** set
  - **performance:**
    - use the fastest compute unit
    - allow flexibility to balance load on various hardware

- Challenges:
  - short time/step
  - fast CPU code

Force offload



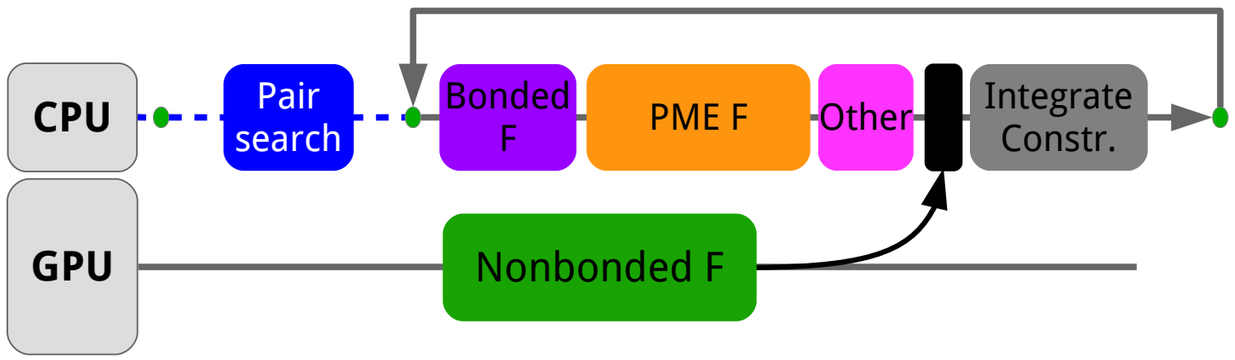
GPU:

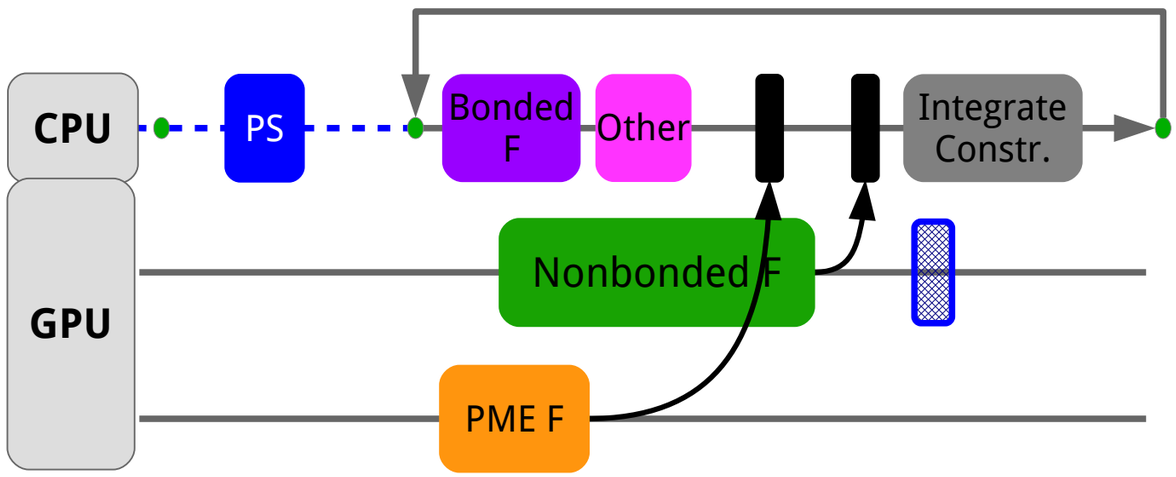
- compute forces where most acceleration is to be had

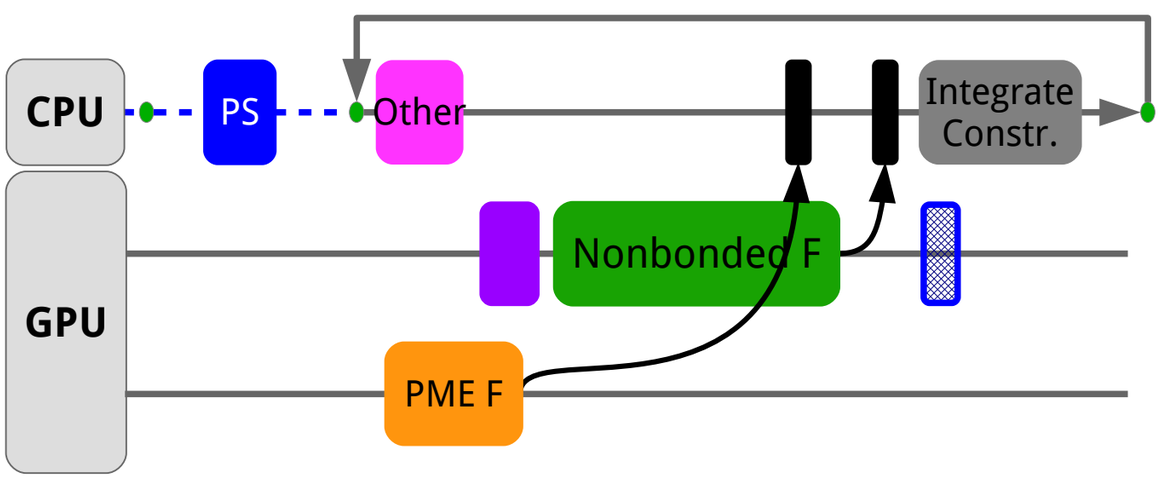
CPU:

- pair search
- other F / special algorithms
- integration

Force offload schemes







- Offloading different force components allows adjusting to hardware balance

- Seach / DD:

  complex code

  kept on the CPU

  → use algorithmic optimization to improve CPU—GPU overlap

Dual pair list

- Introduce two buffers

  outer & inner

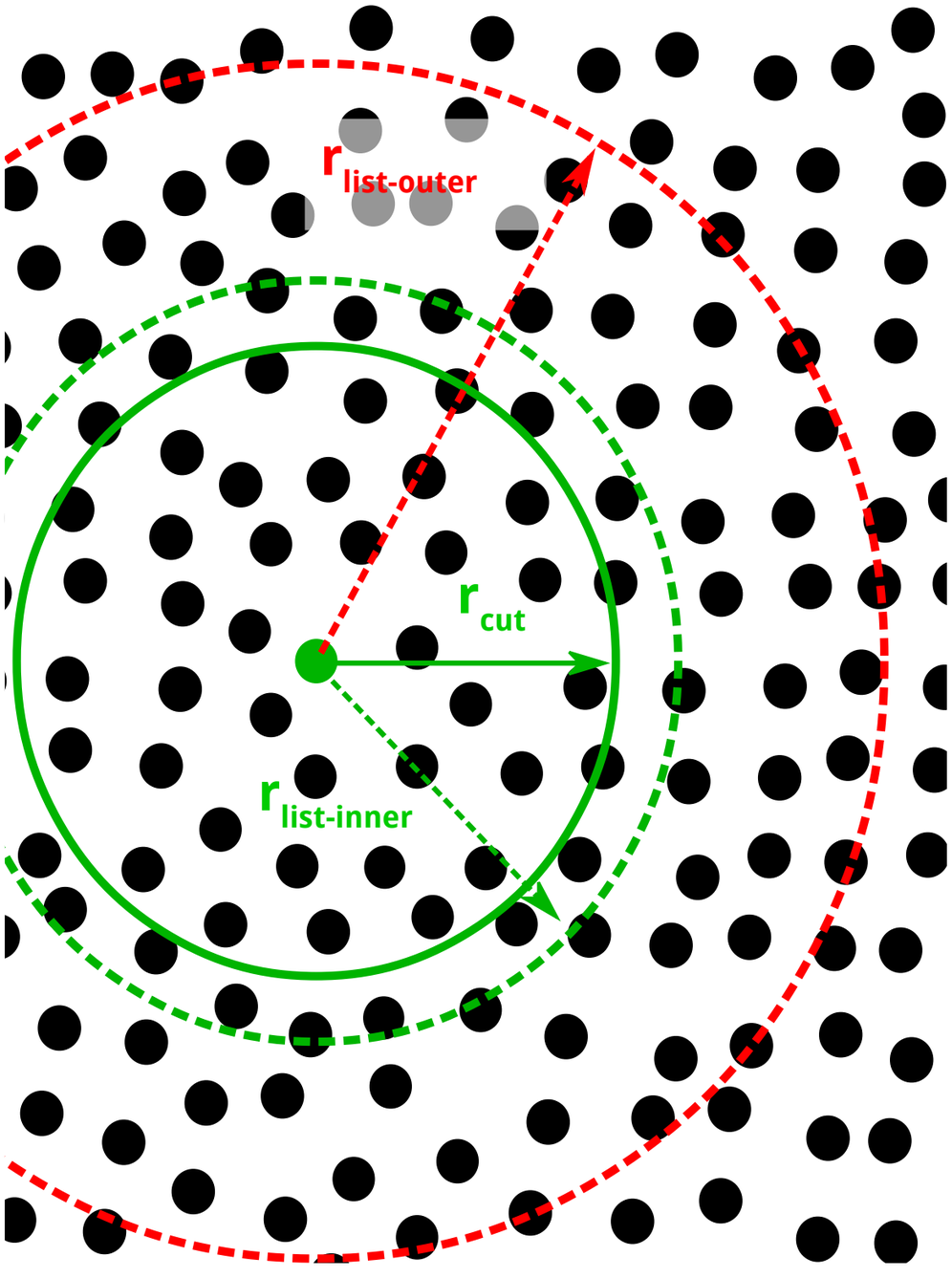
- Periodically re-prune

  outer → inner

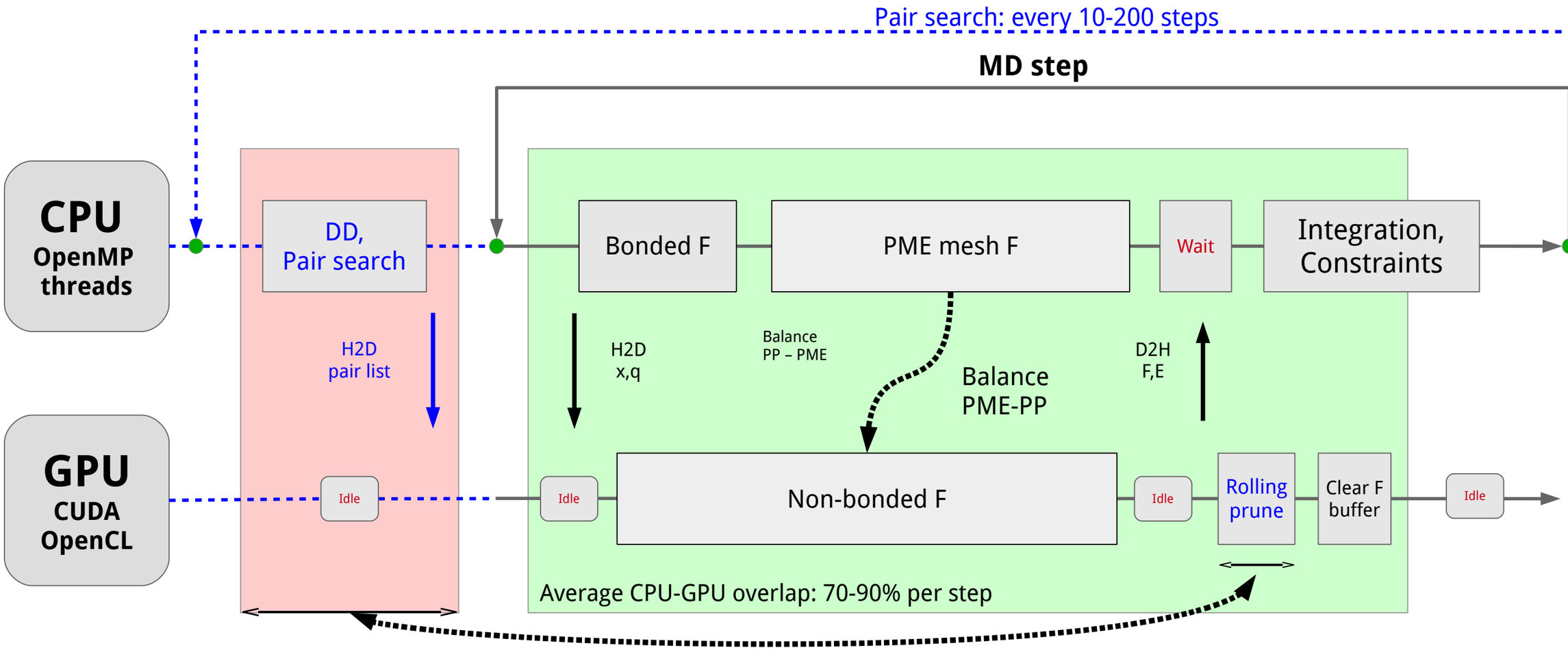
- Build:
  - outer list less frequently
  - inner list more frequently

- Reduces search space to obtaining the inner list used for force compute

`

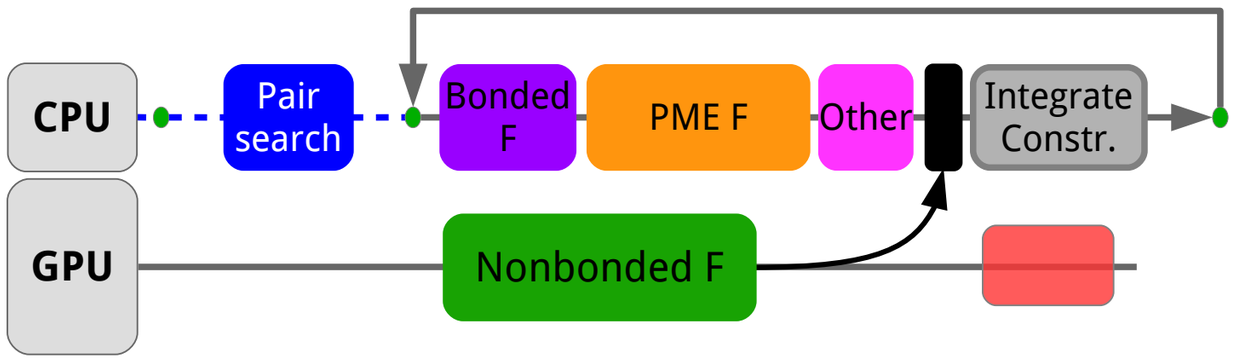


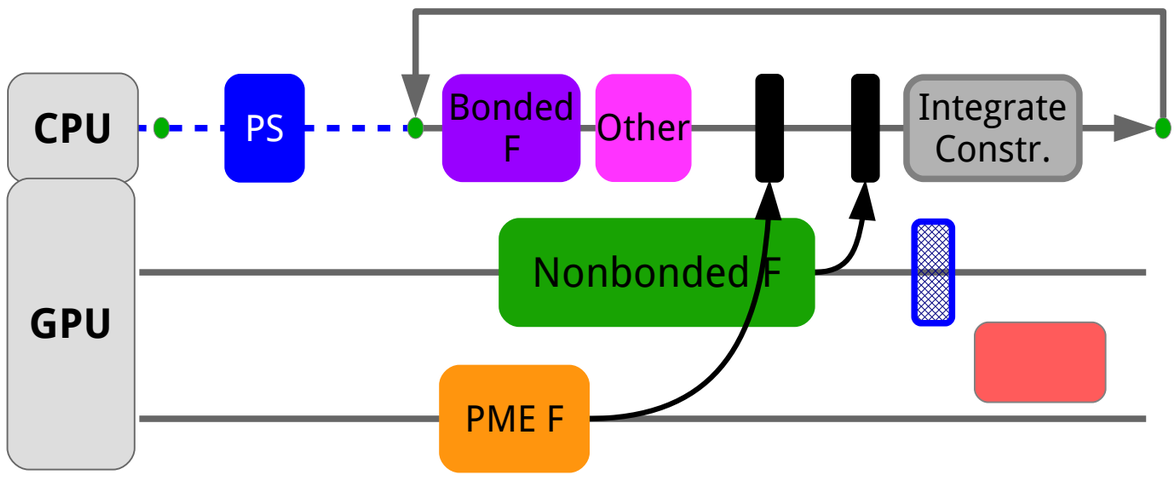
Heterogeneous scheme load balancing

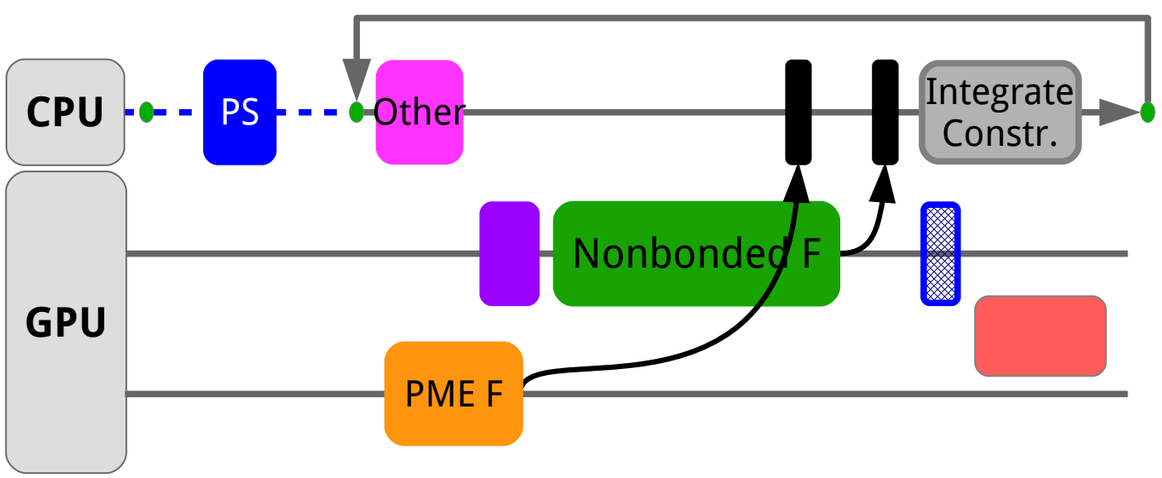


Balance DD+ search & pruning cost

Force offload schemes







Integration on the CPU

=>

CPU – GPU data movement needed

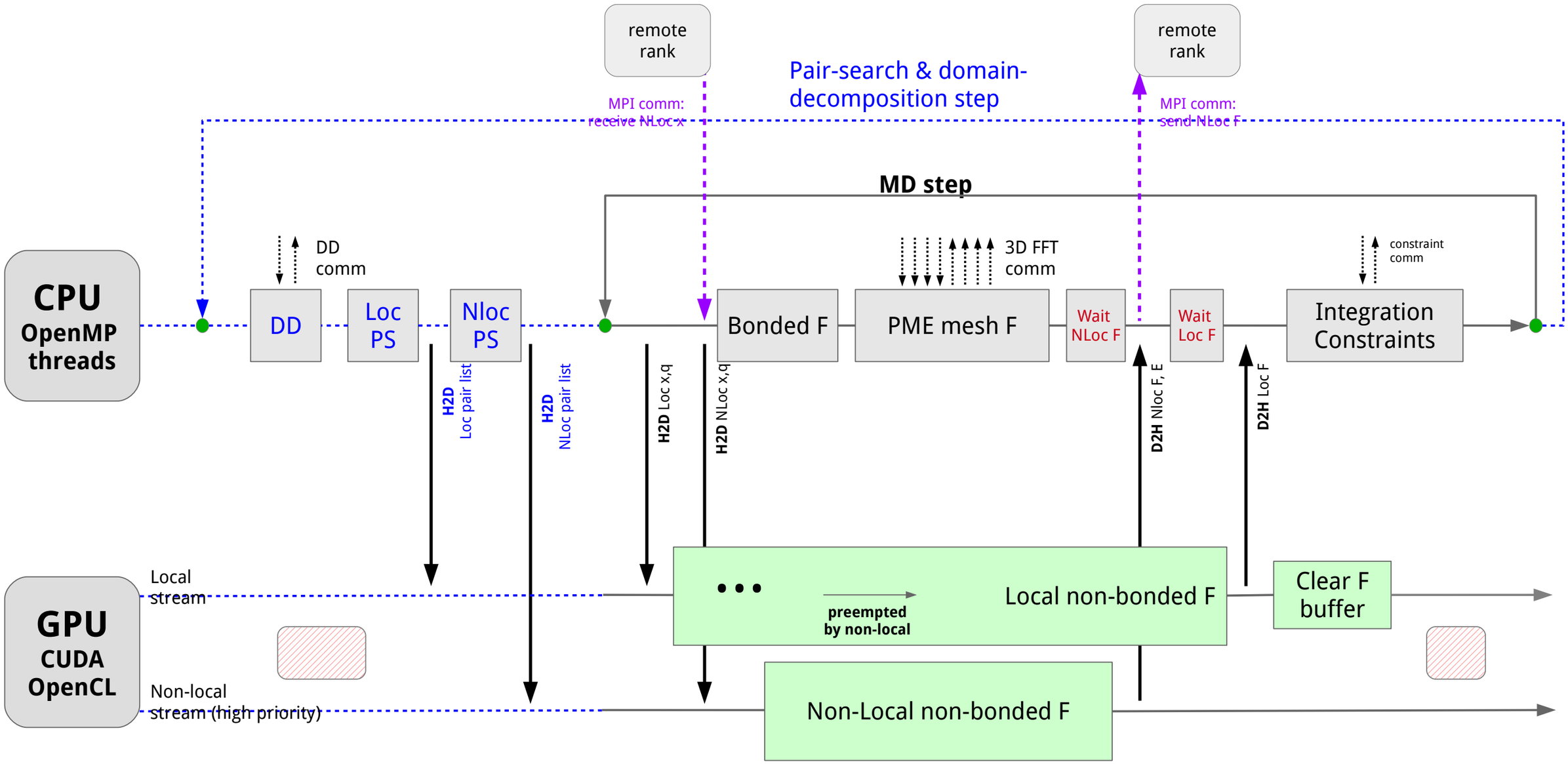
Amdahl:

as GPUs get faster,

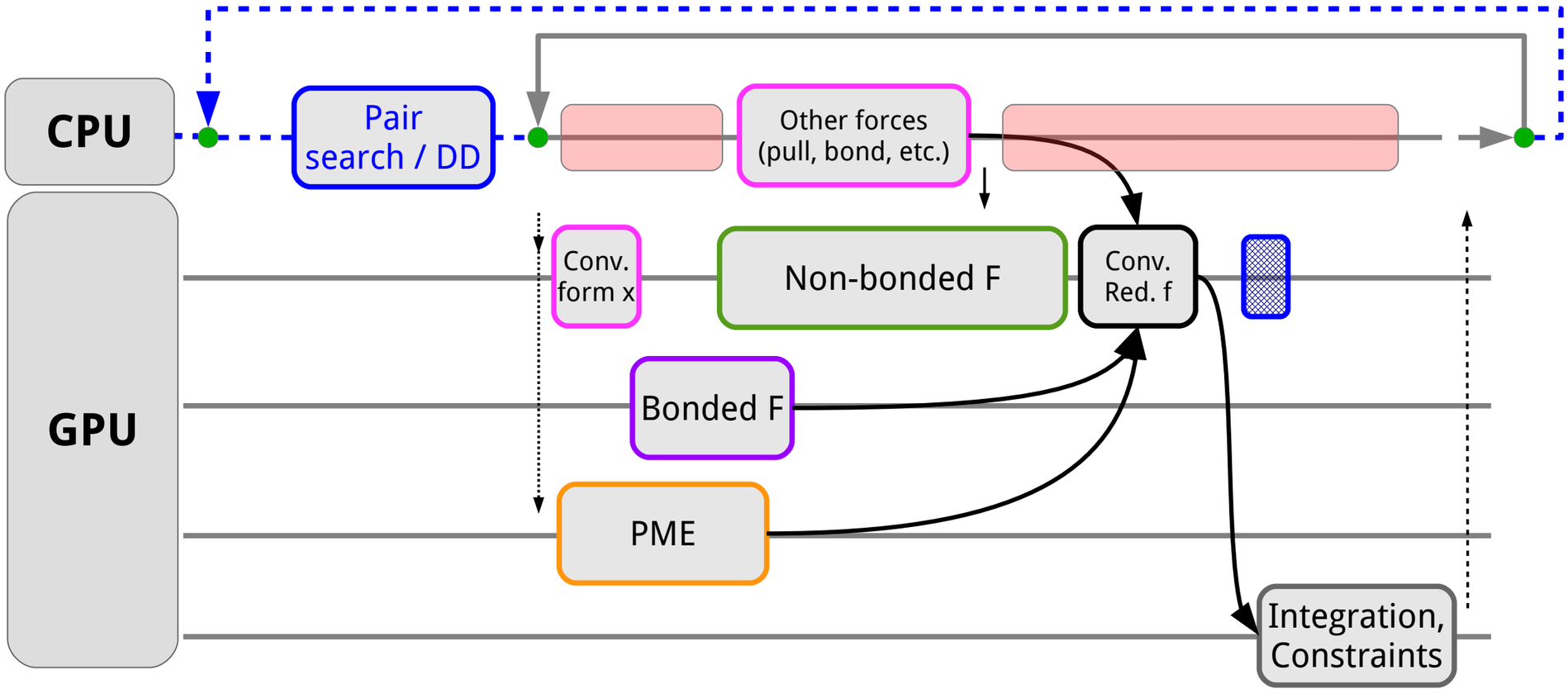
CPU integration time increases

- Solutions:
  - use force decomp & pipeline update (PCIe bottleneck!)
  - offload integration

Multi-node force offload

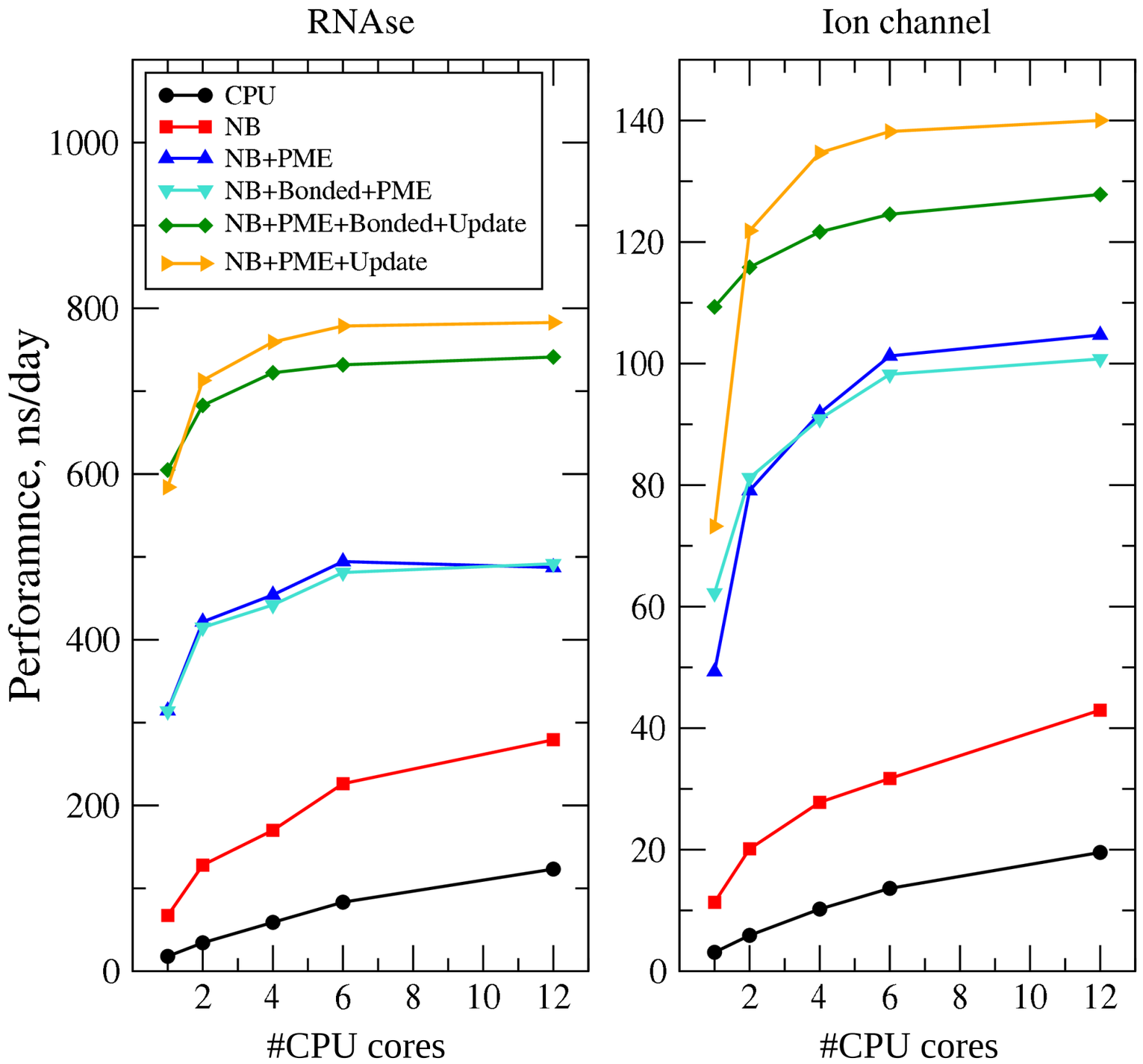


Full step offload



- **Trade GPU idling for CPU idling**: ideal for GPU dense architectures
- CPU supporting role (“back-offload”):
  - non-offloaded per-step algorithms
  - infrequent tasks (search, DD)
- Major benefits with direct communication

Heterogeneous offload performance



- Hardware:

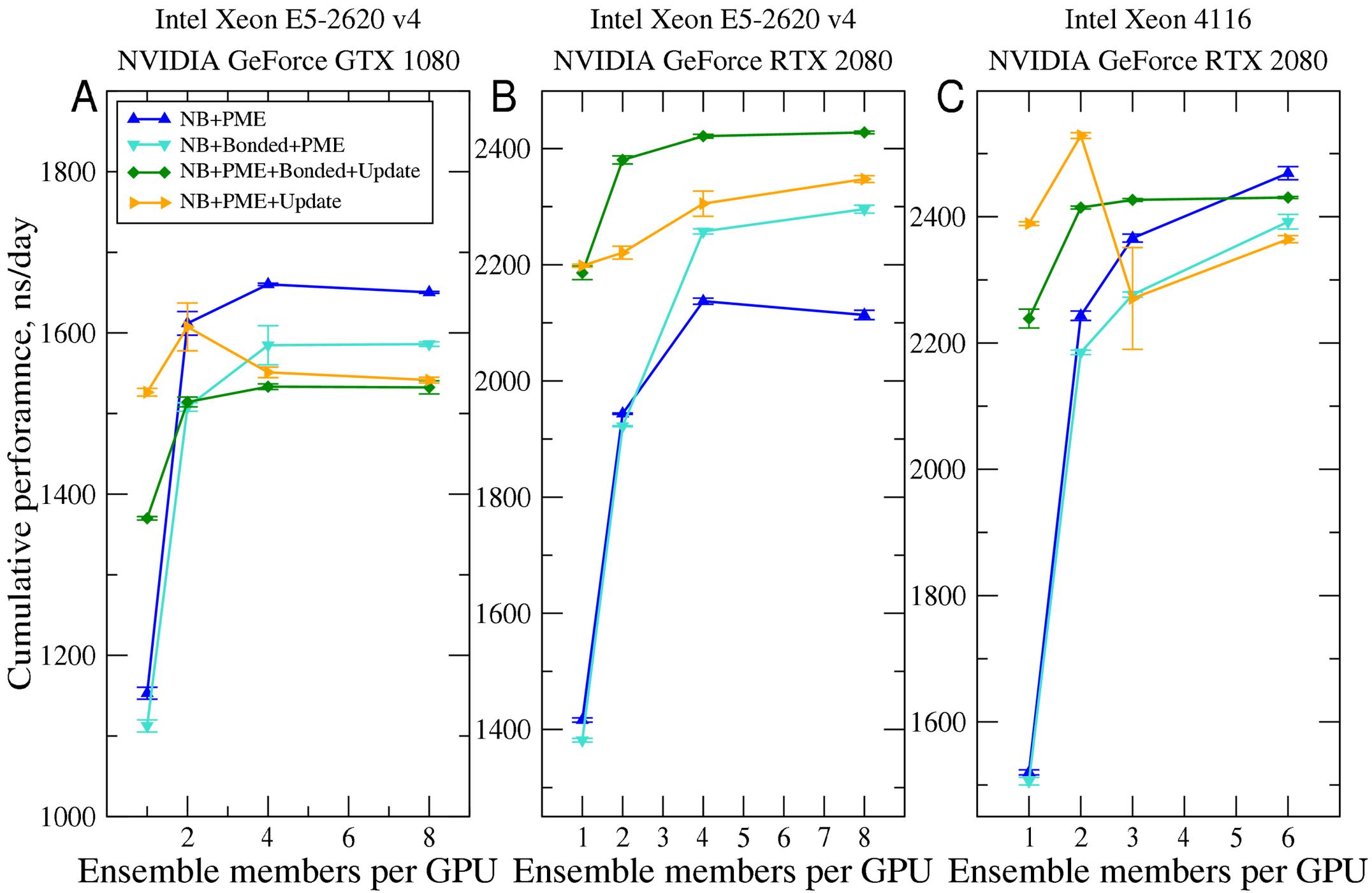
  AMD R 3900X

  NVIDIA 2080 SUPER

- Inputs:
  - RNAse: 24k atoms
  - GluCL ion channel: 144k atoms

  (AMBER FF)

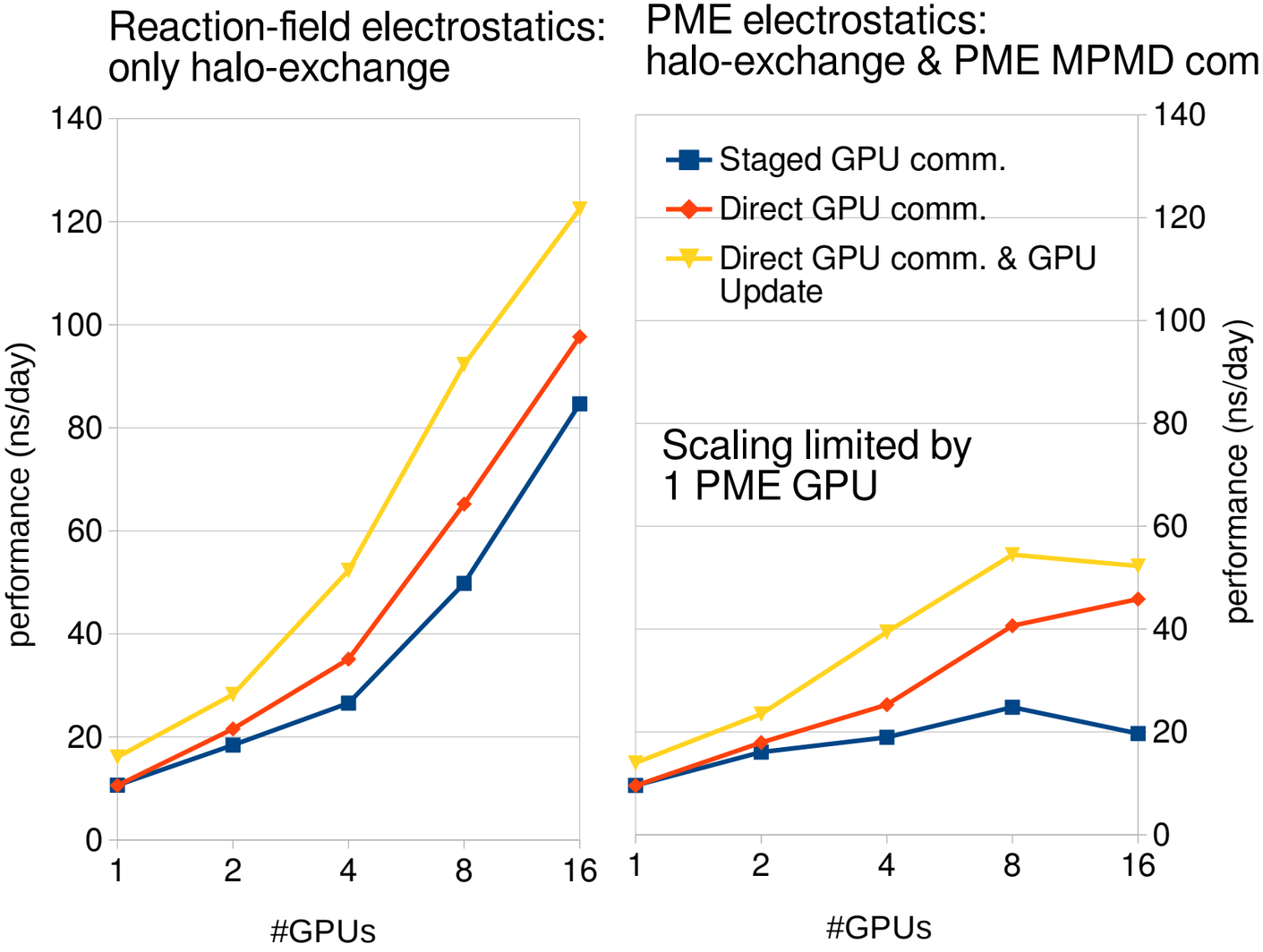
Ensemble performance: small system



- Input: RNAse (24k atom AMBER99sb), uncoupled ensemble run

GPU direct communication

- CUDA-aware MPI: requires CPU sync
- thread-MPI: exchange CUDA events, fully offload communication
- NVIDIA co-design



System:

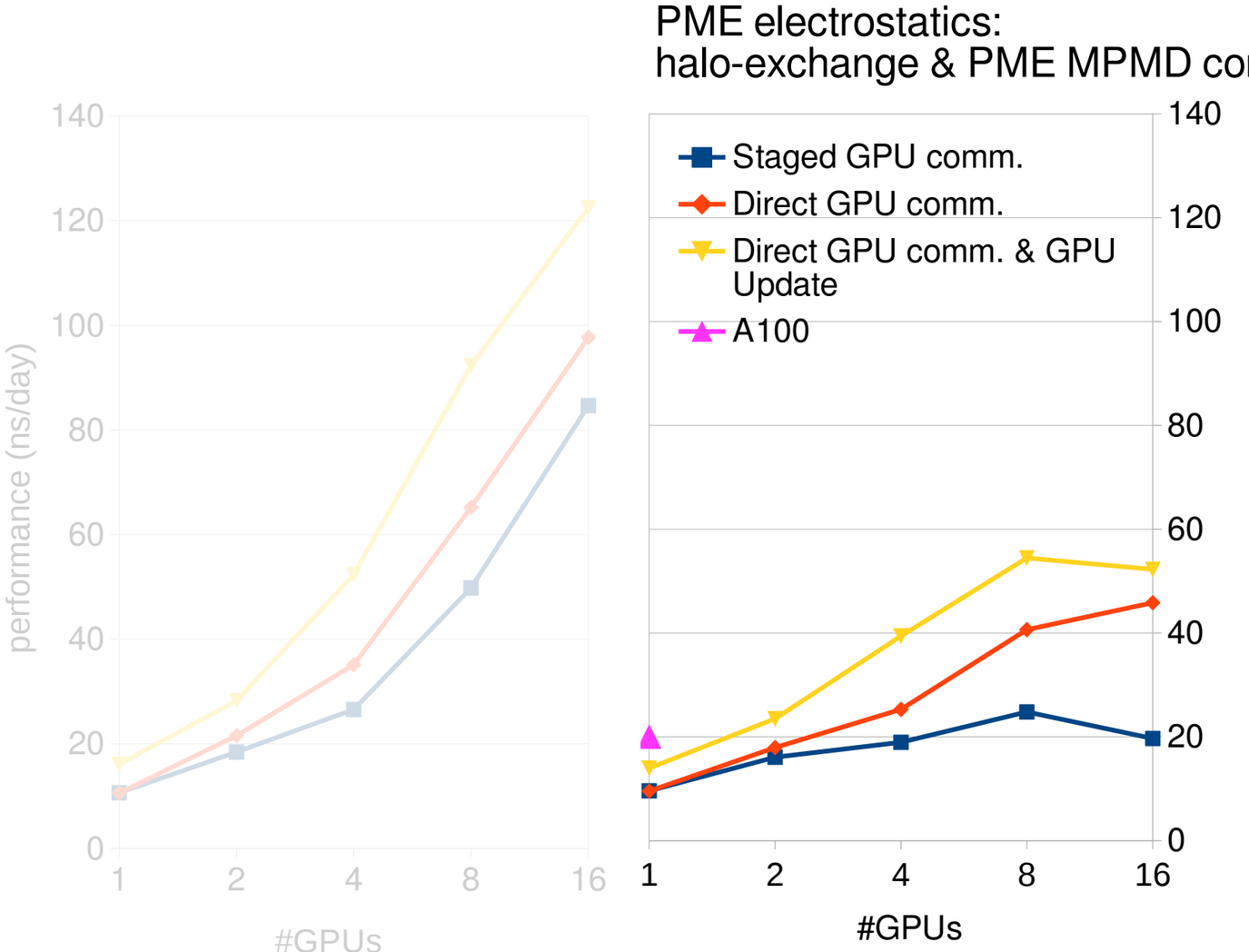
STMV 1M atoms

Hardware:

DGX-1V

A100 is coming

- Promising hardware and CUDA sw features
- Impressive performane at start: ~1.5x



System:

STMV 1M atoms

Hardware:

NVIDIA V100 vs A100*

*Disclaimer: the performance was measured on pre-production hardware

Summary

- Heterogeneity is here to stay
- Strong scaling is hard, fast code is hard to accelerate
- Start with the algorithms
- Target each (most) levels of parallelism
- Open standards-based tools vs propritery tech

Q&A