Michela Mapelli

N-body techniques for astrophysics: Lecture 3 – DIRECT N-BODY codes

PhD School in Astrophysics,
University of Padova
November 19-30, 2018

SCALING of a NUMERICAL PROBLEM:

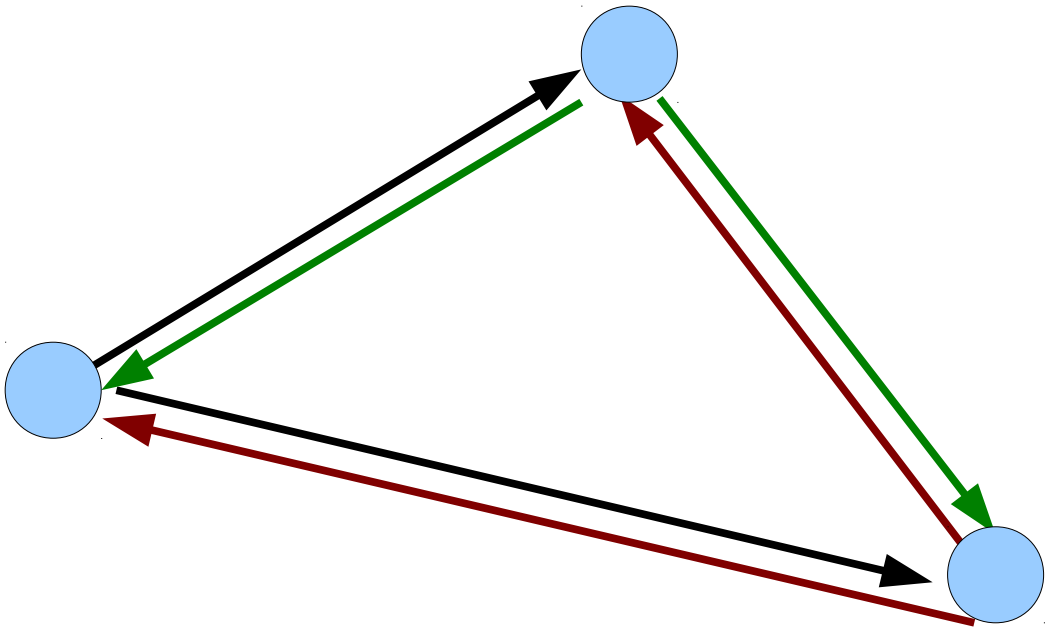
Numerical complexity:

How many calculations I have to do for N-particles?

SCALING of a NUMERICAL PROBLEM:

Numerical complexity:

How many calculations I have to do for N-particles?

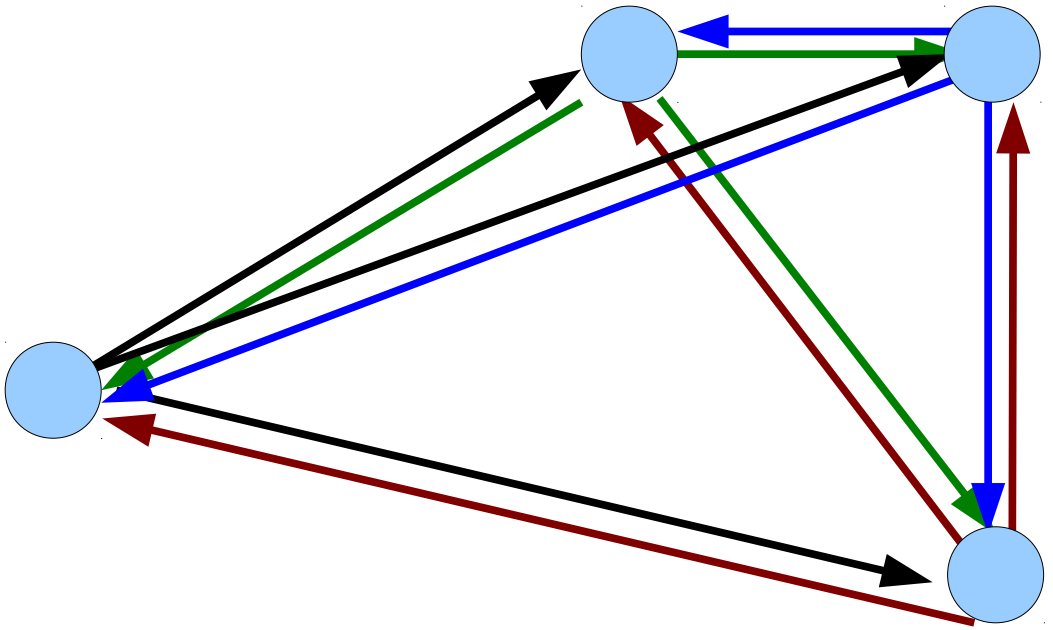


3 particles
6 forces
N (N - 1)

SCALING of a NUMERICAL PROBLEM:

Numerical complexity:

How many calculations I have to do for N-particles?

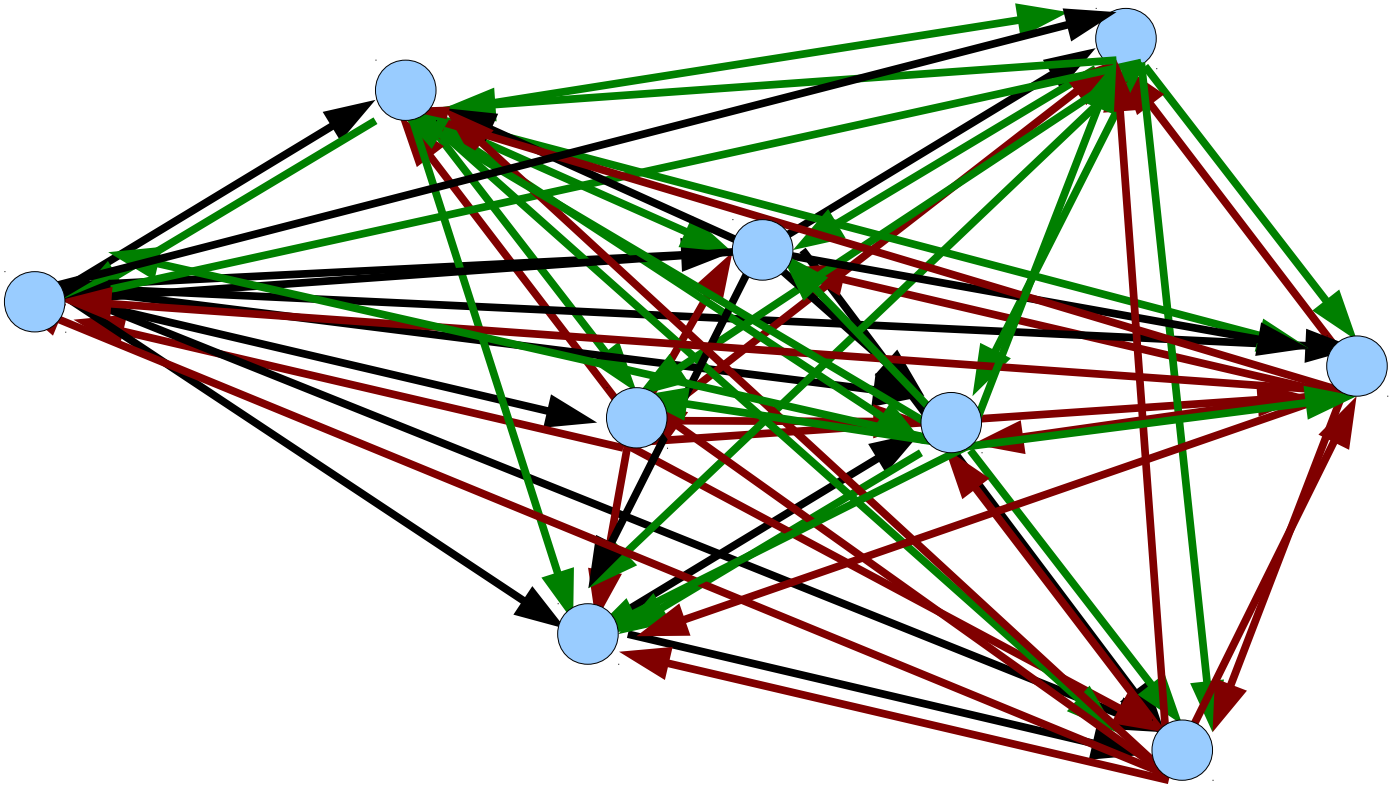


4 particles
12 forces
N (N - 1)

SCALING of a NUMERICAL PROBLEM:

Numerical complexity:

How many calculations I have to do for N-particles?



9 particles
72 forces
N (N – 1)

COMPLEXITY GROWS as N^2 - VERY FAST !!!!

SCALING of a NUMERICAL PROBLEM:

NUMERICAL COMPLEXITY GROWS as N^2 - VERY FAST !!!!

- CAN I REDUCE COMPLEXITY?

YES, BUT IT IS NOT ALWAYS THE RIGHT CHOICE
→ See direct vs indirect N-body in this lecture

- HOW CAN I REDUCE COMPLEXITY?
E.G. with BARNES-HUT TREE METHOD
and/or with MULTIPOLE EXPANSION
→ See LECTURE 4

OUTLINE of LECTURE 3:

BASIC NOTIONS:

1. **WHAT?** DEFINITION of DIRECT N-BODY

2. **WHY/WHEN** DO WE NEED DIRECT N-BODY CODES?

3. **HOW** ARE DIRECT N-BODY CODES IMPLEMENTED?
   - 3.1 EXAMPLE OF INTEGRATOR: Hermite 4^{th} order
   - 3.2 EXAMPLE OF TIME STEP CHOICE: block time step
   - 3.3 EXAMPLE of REGULARIZATION: KS

4. **WHERE?** HARDWARE: 4.1 GRAPE → 4.2 GPU

EXTRA:

5. MPI?

6. coupling with more physics: stellar evolution

7. EXAMPLES

1. DEFINITION

- ONLY force that matters is GRAVITY

- Newton's EQUATIONS of MOTION:

$$\ddot{\vec{r}}_i = -G \sum_{j \neq i} m_j \frac{\vec{r}_i - \vec{r}_j}{|\vec{r}_i - \vec{r}_j|^3}$$

DIRECT N-Body codes calculate all N^2 inter-particle forces
→ SCALE as $O(N^2)$

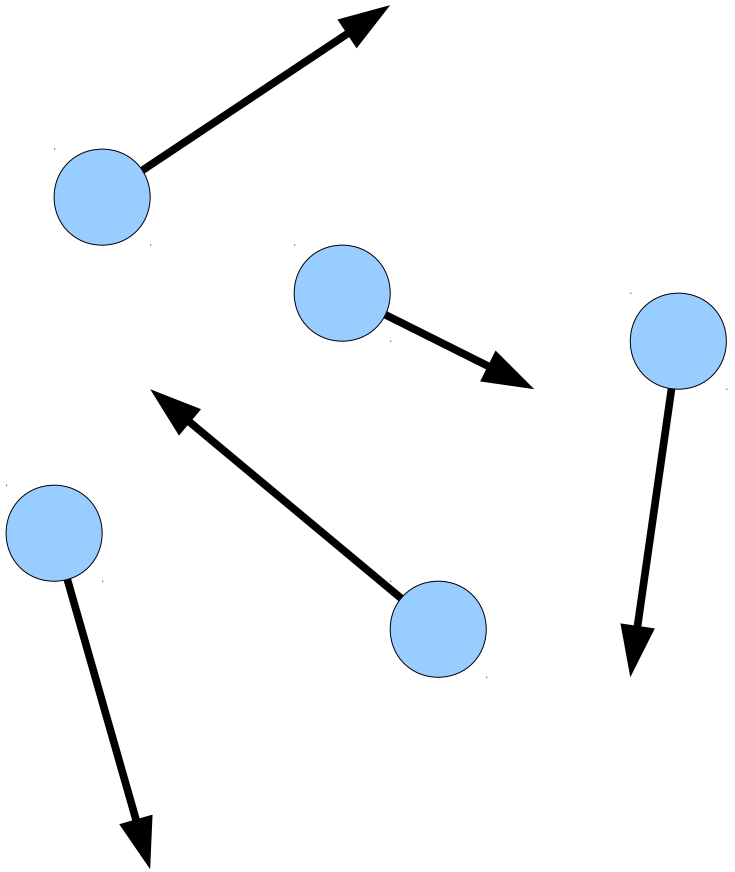
N-body codes that use different techniques
(e.g. MULTIPOLE EXPANSION of FORCES for sufficiently distant particles)
induce **LARGER ERRORS on ENERGY BUT scale as O(N logN)**
– see **NEXT LECTURE**

→ Why do we use expensive direct N-body codes that scale as $O(N^2)$ if we can do similar things with O(N logN) codes?

2. WHY/WHEN do we use direct N-body codes?

We **DO NOT NEED** direct N-body codes for **COLLISIONLESS systems**: astrophysical systems where the **stellar density is low**

→ gravitational interactions between stars are weak and rare, and do not affect the evolution of the system



Interaction
Rate
scales as

density / vel^3

2. WHY/WHEN do we use direct N-body codes?

We **DO NOT NEED** direct N-body codes for **COLLISIONLESS systems**: astrophysical systems where the **stellar density is low**

→ gravitational interactions between stars are weak and rare, and do not affect the evolution of the system

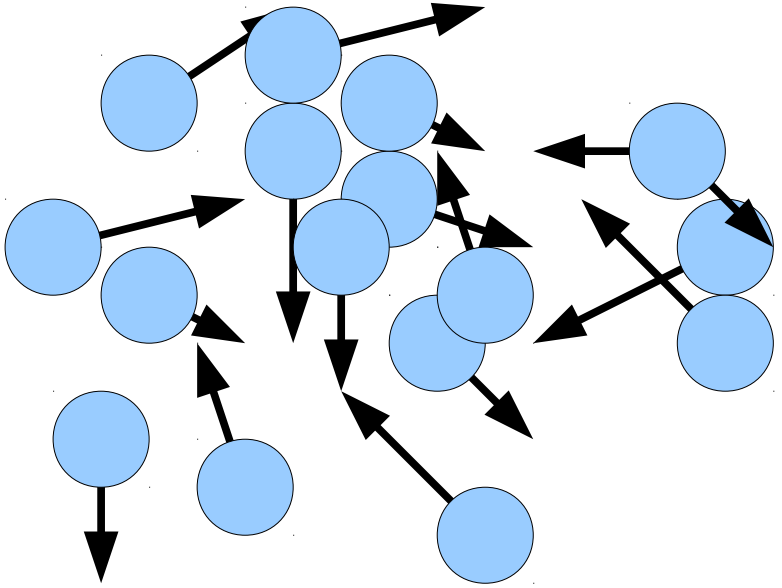
The collisionless systems evolve **SMOOTHLY** in time

→ they can be treated as a FLUID in the phase space

2. WHY/WHEN do we use direct N-body codes?

We NEED DIRECT N-BODY CODES for the COLLISIONAL SYSTEMS:

SYSTEMS WHERE the stellar **DENSITY is so high that single gravitational interactions between particles are frequent**, strong and affect the overall evolution of system (concept of GRANULARITY)



Interaction
Rate
scales as

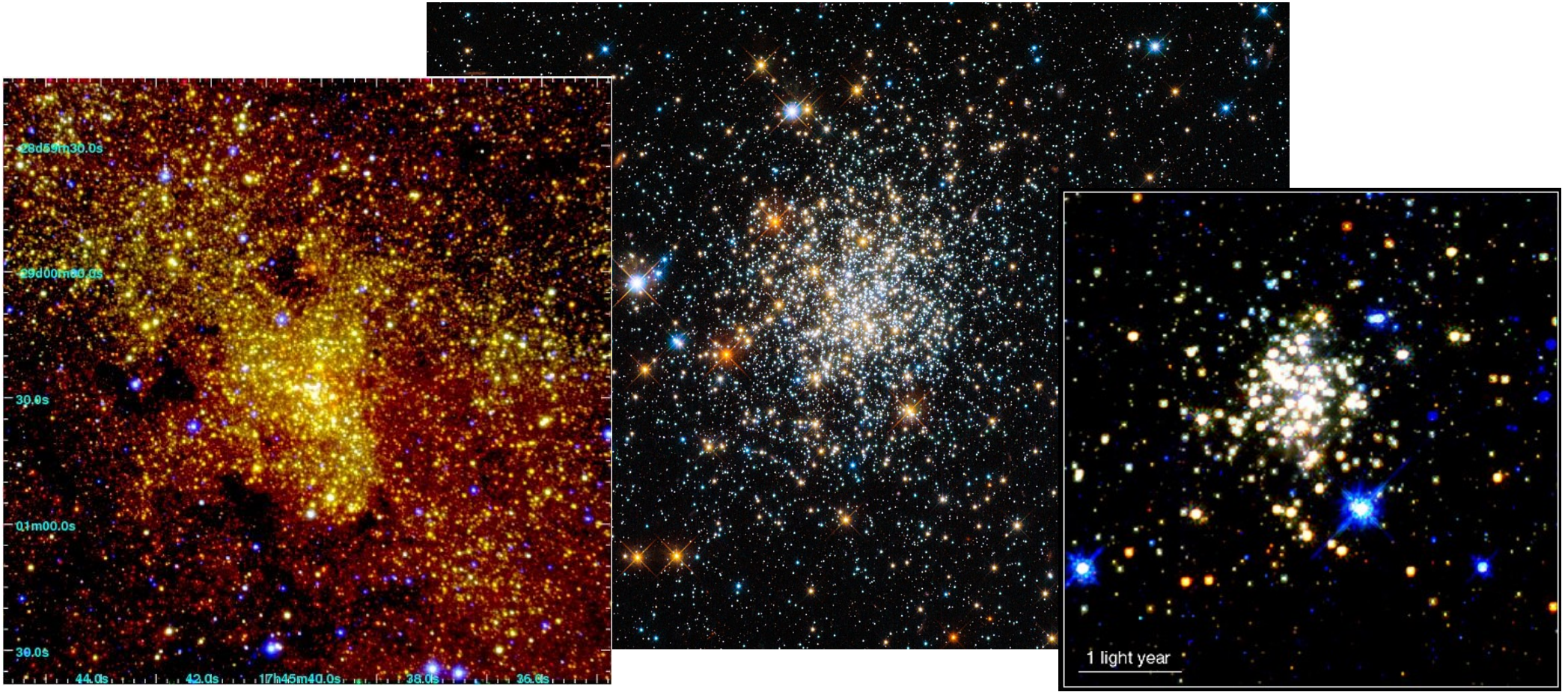
density / vel^3

2. WHY/WHEN do we use direct N-body codes?

We NEED DIRECT N-BODY CODES for the COLLISIONAL SYSTEMS:
SYSTEMS WHERE the stellar **DENSITY is so high that single gravitational interactions between particles are frequent**, strong and affect the overall evolution of system (concept of GRANULARITY)

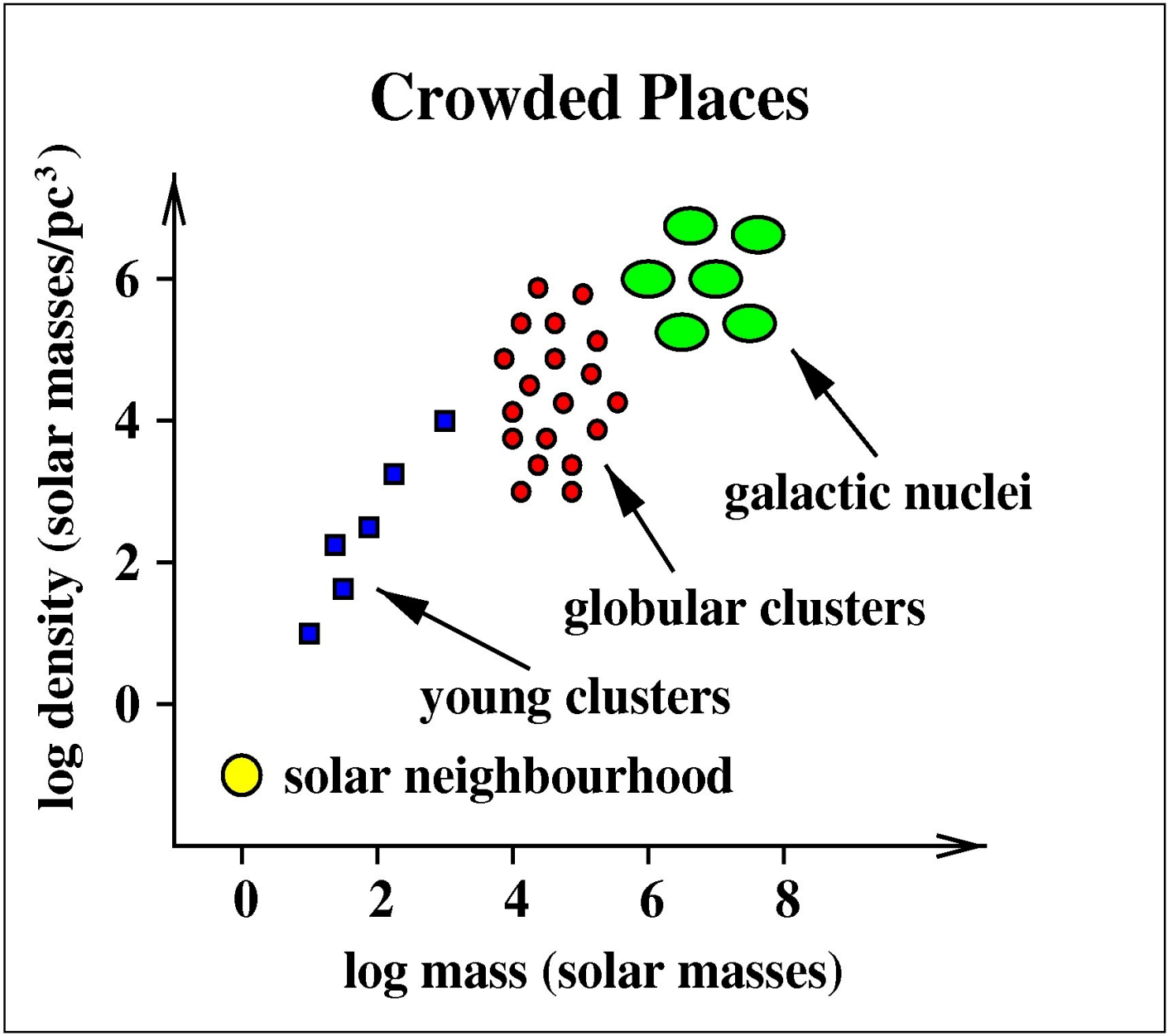
So that we need to **resolve each single star and each interaction it undergoes → We cannot use approximations!!!**

THE Ddensest STELLAR SYSTEMS: STAR CLUSTERS and GALACTIC NUCLEI



2. WHY/WHEN do we use direct N-body codes?

MAP of the DENSEST PLACES in the Universe

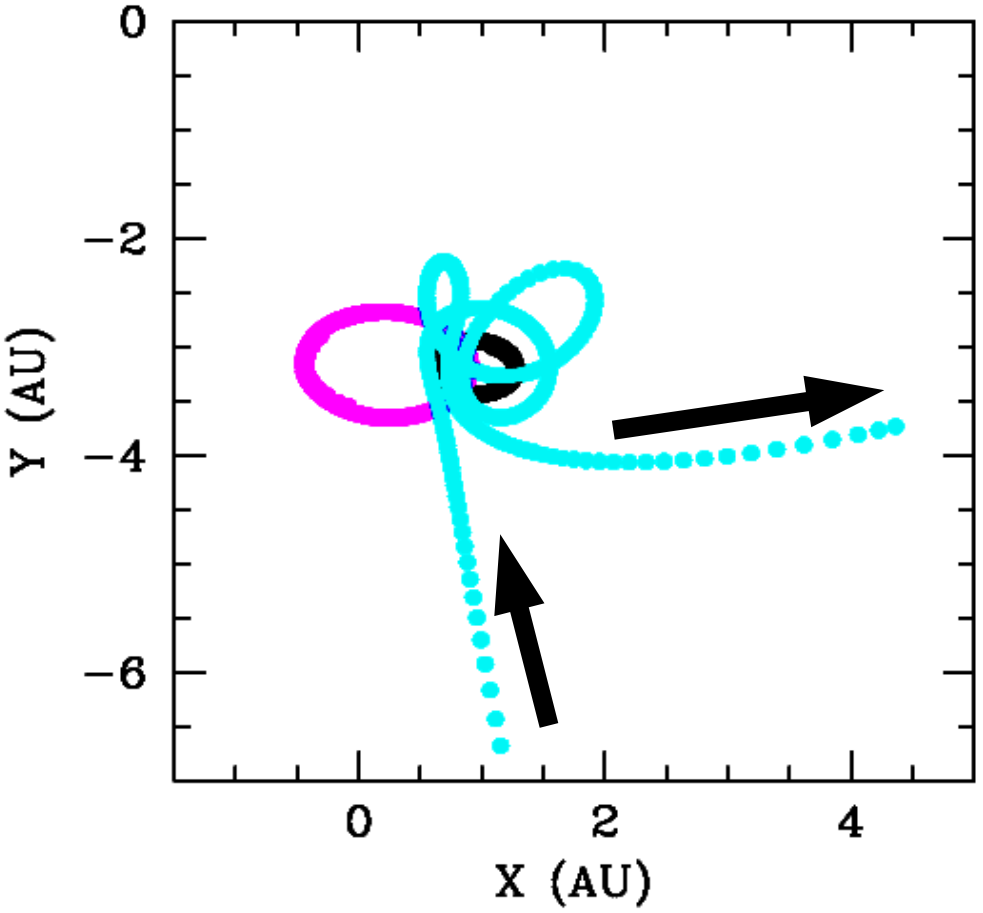


From M. B. Davies 2002

2. WHY/WHEN do we use direct N-body codes?

An important ingredient of COLLISIONAL SYSTEMS are BINARY STARS and

3-BODY ENCOUNTERS := KEPLER BINARIES INTERACT CLOSELY WITH SINGLE STARS AND EXCHANGE ENERGY WITH THEM

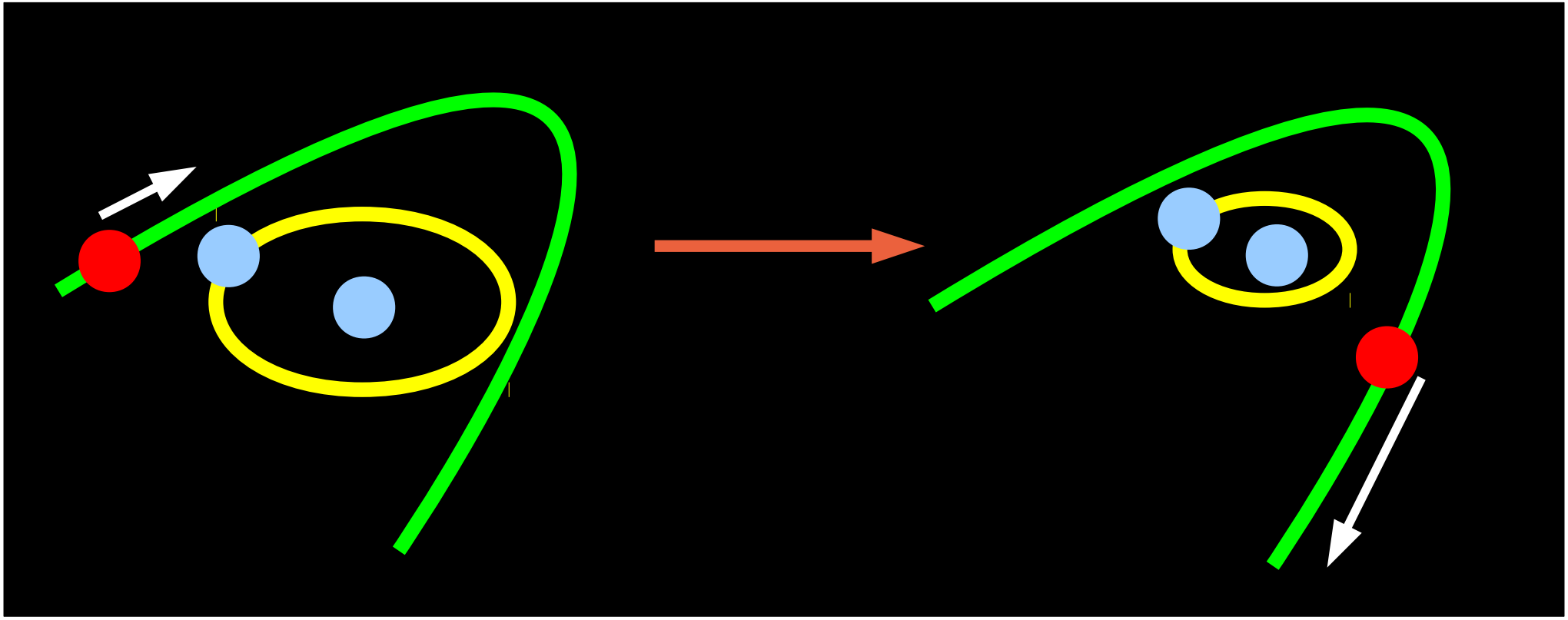


*** Similar to scattering experiments in (sub)atomic physics but involving stars/binary stars and ONLY GRAVITATIONAL FORCE**

*** It is a very important process, because it dominates the energy budget of collisional systems**

2. WHY/WHEN do we use direct N-body codes?

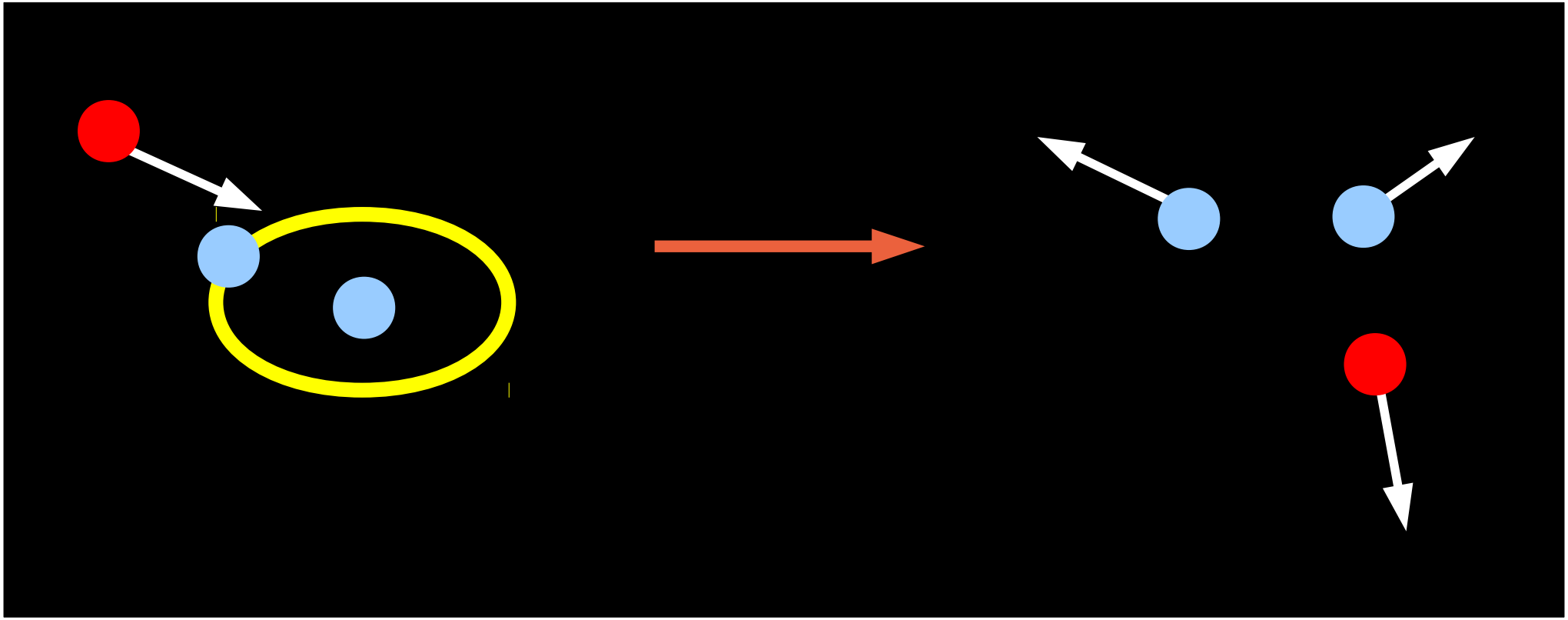
EXAMPLES of 3-BODY ENCOUNTERS



FLYBY: ORBITS CHANGE

2. WHY/WHEN do we use direct N-body codes?

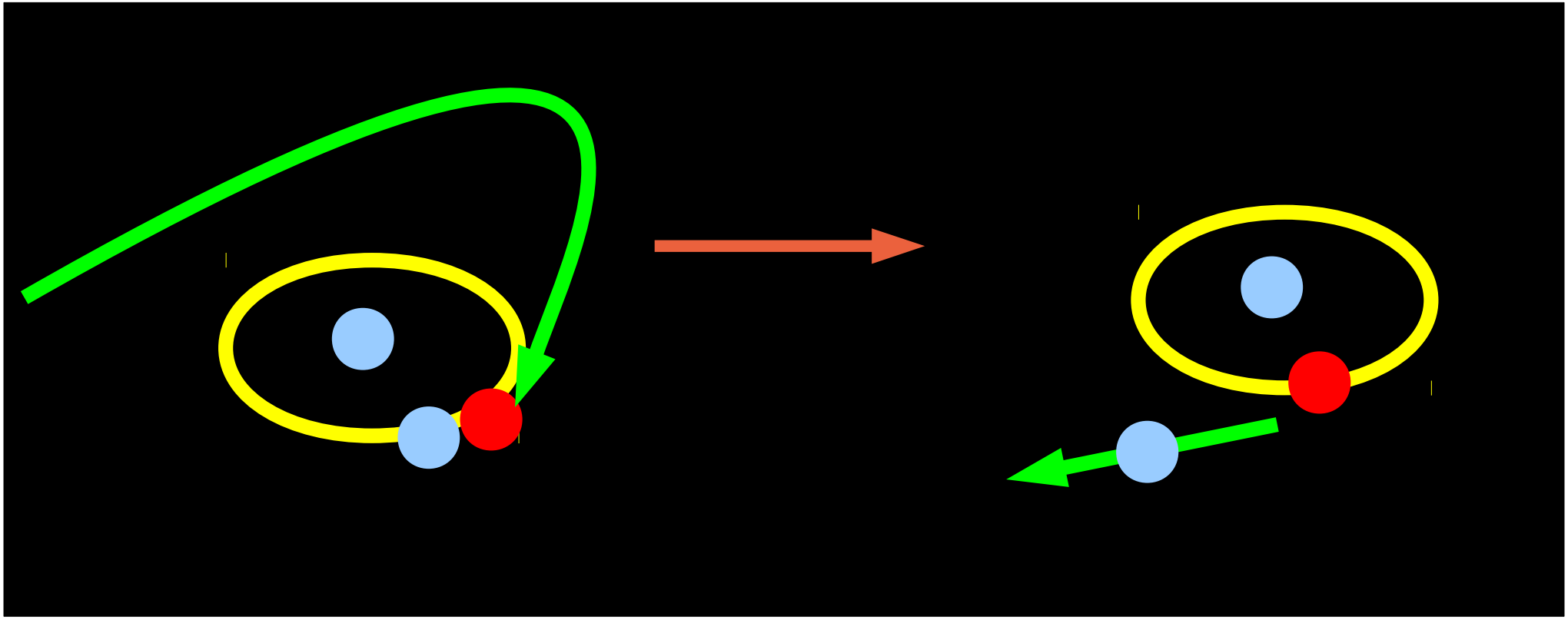
EXAMPLES of 3-BODY ENCOUNTERS



IONIZATION: binary is destroyed (analogy with atoms)

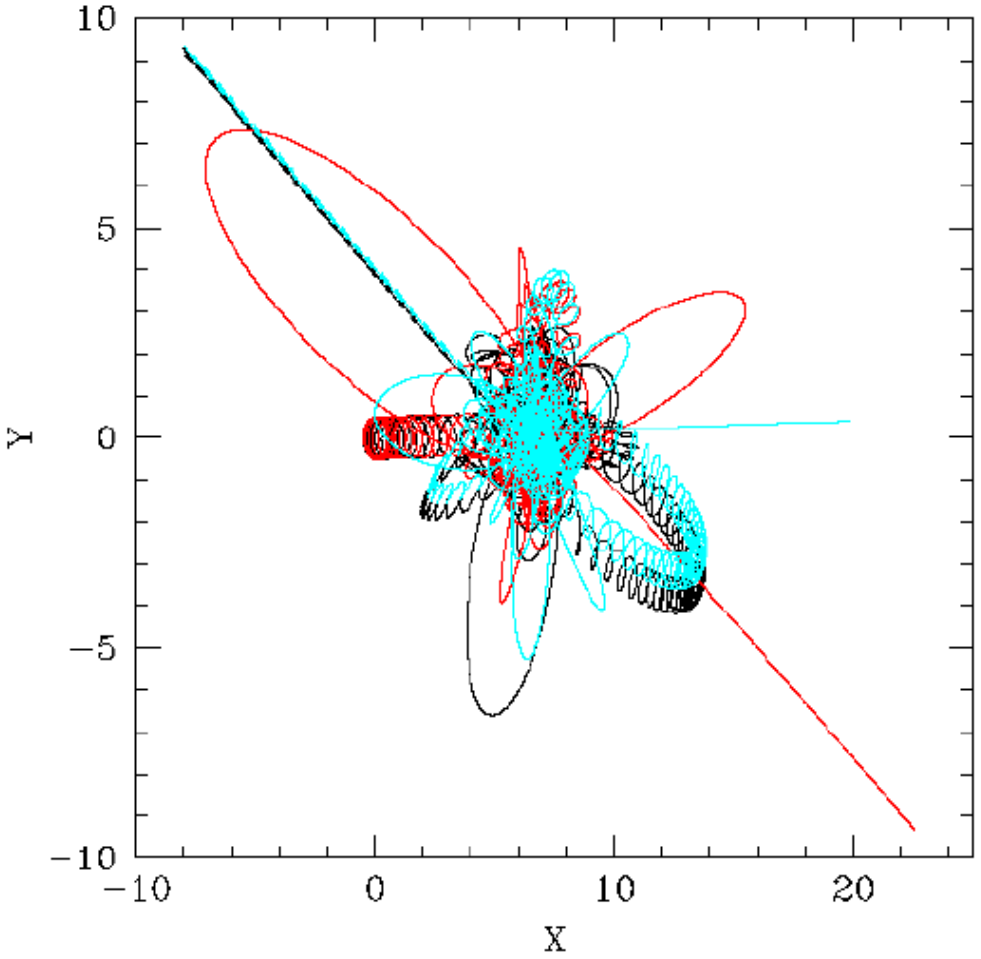
2. WHY/WHEN do we use direct N-body codes?

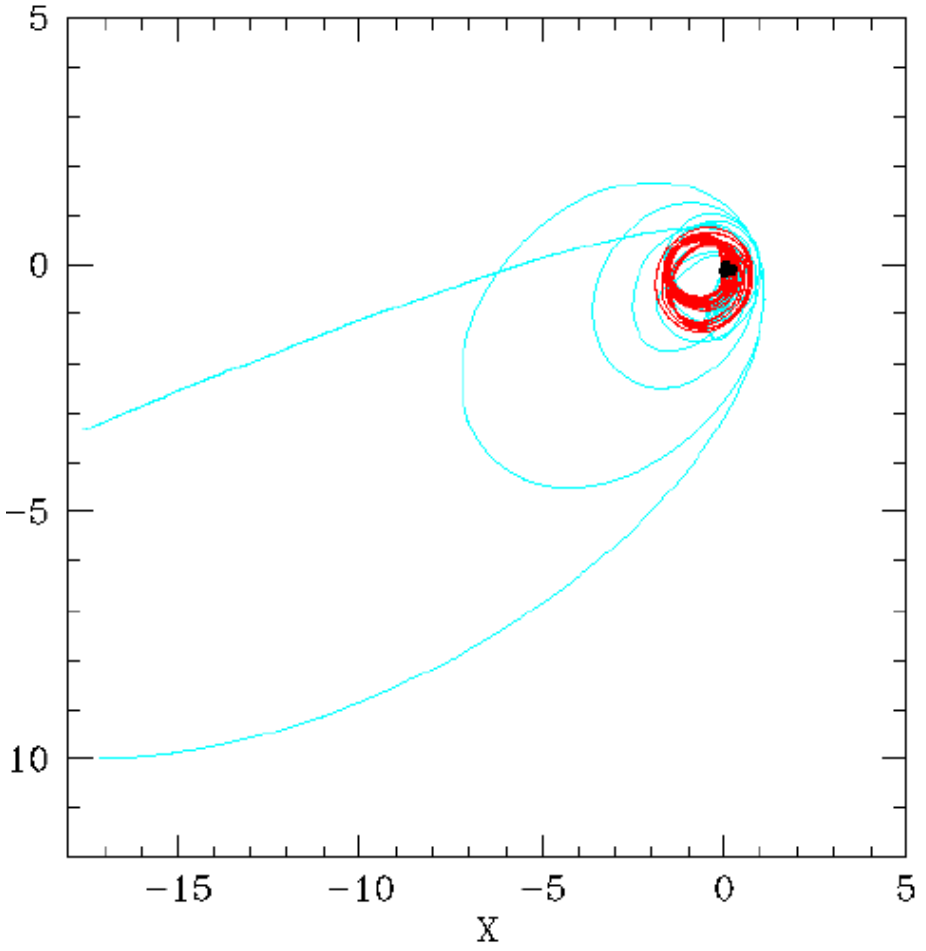
EXAMPLES of 3-BODY ENCOUNTERS



EXCHANGE: binary member is replaced by single star

2. WHY/WHEN do we use direct N-body codes?





→ **TO INTEGRATE CLOSE 3-BODY ENCOUNTERS CORRECTLY IS ONE OF THE MOST CHALLENGING TASKS of DIRECT N-BODY CODES:**
IT REQUIRES
i) VERY SMALL TIMESTEPS (~ a FEW YEARS) AND
ii) HIGH-ORDER INTEGRATION SCHEMES
TO CONSERVE ENERGY and ANG. MOMENTUM DURING THE 3-BODY!

3. HOW are direct N-body codes implemented?

3.1 INTEGRATION SCHEME

If interactions (and especially close interactions) between stars are important

→ integrator must be HIGH ACCURACY even over SHORT TIMES
(integrate perturbations in < 1 orbit)

→ AT LEAST FOURTH-ORDER ACCURACY

4th ORDER PREDICTOR-CORRECTOR HERMITE SCHEME

Based on **JERK** (time derivative of acceleration)

$$\vec{a}_i = G \sum_{j \neq i} \frac{M_j}{r_{ji}^3} \vec{r_{ij}}$$

$$\frac{d\vec{a}_i}{dt} = \dot{\vec{j}}_i = G \sum_{j \neq i} M_j \left[\frac{\vec{v}_{ji}}{r_{ji}^3} - 3 \frac{(\vec{r}_{ji} \cdot \vec{v}_{ji})\ \vec{r}_{ji}}{r_{ji}^5} \right]$$

3. HOW are direct N-body codes implemented?

3.1 INTEGRATION SCHEME

4th ORDER PREDICTOR-CORRECTOR HERMITE SCHEME

Based on **JERK** (time derivative of acceleration)

BETTER ADD A SOFTENING
(often is the PHYSICAL RADIUS OF STARS)

$$\vec{a_i} = G \sum_{j \neq i} \frac{M_j \, \vec{r_{ij}}}{\left(r_{ji}^2 + \epsilon^2 \right)^{3/2}}$$

$$\frac{d\vec{a_i}}{dt} = \vec{j_i} = G \sum_{j \neq i} M_j \left[\frac{\vec{v_{ij}}}{\left(r_{ji}^2 + \epsilon^2 \right)^{3/2}} + \frac{3(\vec{v_{ij}} \cdot \vec{r_{ij}}) \, \vec{r_{ij}}}{\left(r_{ji}^2 + \epsilon^2 \right)^{5/2}} \right]$$

3.1 INTEGRATION SCHEME

Let us start from 4th order derivative of Taylor expansion:

$$\begin{cases} x_1 = x_0 + v_0\,\Delta t + \frac{1}{2}\,a_0\,\Delta t^2 + \frac{1}{6}j_0\,\Delta t^3 + \frac{1}{24}\dot{j}_0\,\Delta t^4 & (1) \\ v_1 = v_0 + a_0\,\Delta t + \frac{1}{2}j_0\,\Delta t^2 + \frac{1}{6}\dot{j}_0\,\Delta t^3 + \frac{1}{24}\ddot{j}_0\,\Delta t^4 & (2) \\ a_1 = a_0 + j_0\,\Delta t + \frac{1}{2}\dot{j}_0\,\Delta t^2 + \frac{1}{6}\ddot{j}_0\,\Delta t^3 & (3) \\ j_1 = j_0 + \dot{j}_0\,\Delta t + \frac{1}{2}\ddot{j}_0\,\Delta t^2 & (4) \end{cases}$$

We use equations (3) and (4) to eliminate the 1st and 2nd derivative of jerk in equations (1) and (2). We obtain

$$x_1 = x_0 + \frac{1}{2}\,(v_0 + v_1)\,\Delta t + \frac{1}{12}\,(a_0 - a_1)\,\Delta t^2 + O(\Delta t^5) \quad (5)$$

$$v_1 = v_0 + \frac{1}{2}\,(a_0 + a_1)\,\Delta t + \frac{1}{12}\,(j_0 - j_1)\,\Delta t^2 + O(\Delta t^5) \quad (6)$$

WHICH ARE 4th order accuracy:
ALL TERMS in dj/dt *(snap)* and d^2j/dt^2 *(crackle)* disappear: it is 4th order accuracy with only 2nd order terms!!!

But IMPLICIT for a_1, v_1 and j_1 → we need something to predict them

3.1 INTEGRATION SCHEME

DOUBLE TRICK!

1) PREDICTION: we use the 3rd order Taylor expansion to PREDICT x_1 and v_1

$$x_{p,1} = x_0 + v_0\,\Delta t + \frac{1}{2}\,a_0\,\Delta t^2 + \frac{1}{6}\,j_0\,\Delta t^3 \qquad v_{p,1} = v_0 + a_0\,\Delta t + \frac{1}{2}\,j_0\,\Delta t^2$$

2) FORCE EVALUATION:

we use these PREDICTIONS to evaluate PREDICTED acceleration and jerk ($a_{p,1}$ and $j_{p,1}$), from Newton's formula.

3) CORRECTION:

we then substitute $a_{p,1}$ and $j_{p,1}$ into equations (5) and (6):

$$x_1 = x_0 + \frac{1}{2}\,(v_0 + v_{p,1})\,\Delta t + \frac{1}{12}\,(a_0 - a_{p,1})\,\Delta t^2$$

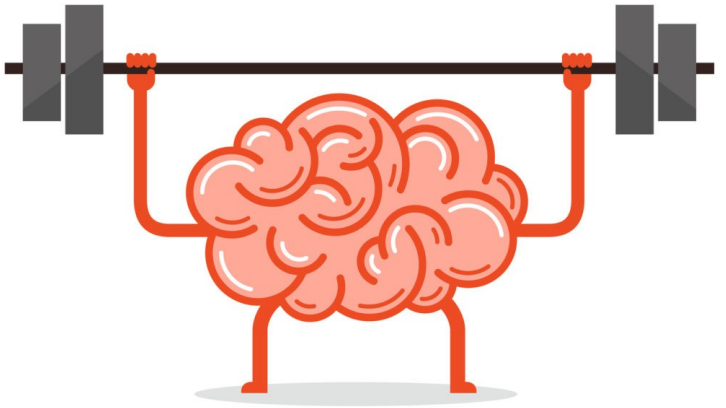
$$v_1 = v_0 + \frac{1}{2}\,(a_0 + a_{p,1})\,\Delta t + \frac{1}{12}\,(j_0 - j_{p,1})\,\Delta t^2$$

This result is only 3rd order in positions! But there is a dirty trick to make it 4th order: we calculate v_1 first and then use the result into x_1

$$v_1 = v_0 + \frac{1}{2}\,(a_0 + a_{p,1})\,\Delta t + \frac{1}{12}\,(j_0 - j_{p,1})\,\Delta t^2$$

$$x_1 = x_0 + \frac{1}{2}\,(v_0 + v_1)\,\Delta t + \frac{1}{12}\,(a_0 - a_{p,1})\,\Delta t^2$$

Exercise # 9:

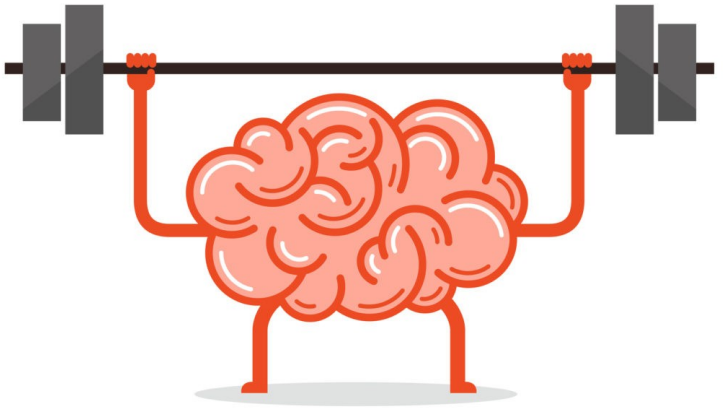


Write your own
4th order Hermite code

*** Use it to simulate the binary of exercise #1**

*** Use it to simulate the Plummer initial conditions of exercise #8**

Exercise # 9:



Write your own
4th order Hermite code

*** Use it to simulate the binary of exercise #1**

*** Use it to simulate the Plummer initial conditions of exercise #8**

3. HOW are direct N-body codes implemented?

3.2 TIME STEP

We can always choose the SAME TIMESTEP for all PARTICLES

BUT: highly expensive because a few particles undergo close encounters → force changes much more rapidly than for other particles

→ we want different timesteps:
longer for 'unperturbed' particles
shorter for particles that undergo close encounter

A frequently used choice:

BLOCK TIME STEPS (Aarseth 1985)

3.2 TIME STEP:

IDEAL CHOICE of TIMESTEP

1. Initial time-step calculated as
   for a particle *i*
   η = 0.01 – 0.02 is good choice

$$\Delta t_i = \eta \frac{a_i}{\dot{j}_i}$$

2. system time is set as $t := t_i + \min(\Delta t_i)$
   All particles with time-step = min (Δt_i) are called
   ACTIVE PARTICLES
   At time *t* the predictor-corrector is done only for active particles

3. Positions and velocities are PREDICTED for ALL PARTICLES

4. Acceleration and jerk are calculated ONLY for ACTIVE PARTICLES

5. Positions and velocities are CORRECTED ONLY for active particles
   (for the other particles predicted values are fine)

After force calculation, new timesteps evaluated as 1. and everything is repeated

BUT a different t_i for each particles is VERY EXPENSIVE and system loses coherence

3.2 TIME STEP:

$$\Delta t_i = \eta \frac{a_i}{\dot{j}_i}$$

A different Δt_i for each particles is VERY EXPENSIVE and the system loses coherence

→ BLOCK TIME STEP SCHEME consists in grouping particles by replacing their individual time steps Δt_i with a

BLOCK TIME STEP $\Delta t_{i,b} = (1/2)^n$

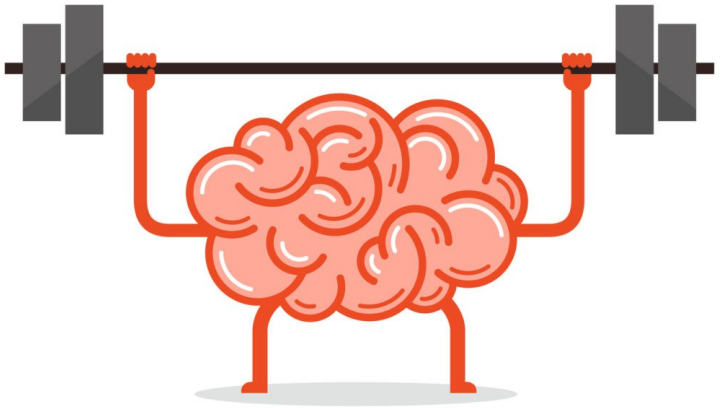
where *n* is chosen according to

$$\left(\frac{1}{2}\right)^n \leq \Delta t_i < \left(\frac{1}{2}\right)^{n-1}$$

This imposes that $t/\Delta t_{i,b}$ be an integer → good for synchronizing the particles at some time

Often it is set a minimum $\Delta t_{min} = 2^{\wedge -23}$

Exercise # 10:



Add block time steps
to your 4th order Hermite code
(the one you developed in exercise #9)

Use it to simulate the Plummer initial conditions of exercise #8

NOTES on Hermite and time steps:

*** MOST CODES USE slightly more accurate equations for the CORRECTOR:**

$$x_1 = x_{p,1} + \frac{\Delta t^4}{24}\, a_0^{(2)} + \frac{\Delta t^5}{120}\, a_0^{(3)} \qquad v_1 = v_{p,1} + \frac{\Delta t^3}{6}\, a_0^{(2)} + \frac{\Delta t^4}{24}\, a_0^{(3)}$$

where

$$a_0^{(2)} = \frac{-6\,(a_0 - a_1) - \Delta t\,(4\,j_0 + 2\,j_1)}{\Delta t^2}$$

$$a_0^{(3)} = \frac{12\,(a_0 - a_1) + 6\,\Delta t\,(j_0 + j_1)}{\Delta t^3}$$

see eg. phiGRAPE (Harfst et al. 2007), STARLAB (Portegies Zwart et al. 2001)

*** Then, the choice of time steps is done with the formula (Aarseth 1985):**

$$\Delta t_i = \sqrt{\eta\, \frac{|a_{i,1}|\,|a_{i,1}^{(2)}| + |j_{i,1}|^2}{|j_{i,1}|\,|a_{i,1}^{(3)}| + |a_{i,1}^{(2)}|^2}}$$

where η = 0.01 – 0.02 is good choice

NOTE: definition of η for some codes (eg STARLAB) is different

$\eta_{STARLAB}$=sqrt(η) → $\eta_{STARLAB}$= 0.1 is good choice (Anders+2012)

***Some codes even use the 6th order Hermite scheme**

eg. **HiGPUs code,** http://astrowww.phys.uniroma1.it/dolcetta/HPCcodes/HiGPUs.html
Capuzzo Dolcetta, Spera & Punzo, 2013, Journal of Computational Physics, 236, 580

3. HOW are direct N-body codes implemented?

3.3 REGULARIZATION

Definition:

mathematical trick to remove the singularity in the Newtonian law of gravitation for two particles which approach each other arbitrarily close.

Is the same as softening????

NO, it is a CHANGE OF VARIABLES, that removes singularity without affecting the physics

Most used regularizations in direct N-body codes:

-Kustaanheimo-Stiefel (KS) regularisation
a regularization for binaries and 3-body encounters

-Aarseth / Mikkola CHAIN regularization
a regularization for small N-body problems

3.3 REGULARIZATION

Regularisation for binaries and 3-body encounters:

Kustaanheimo-Stiefel (KS) regularisation

Levi-Civita (1956): regularize Kepler orbit of a binary in 2 dimensions

KS (1965): extension to 3 dimensions of Levi-Civita regularization
see Funato et al. (1996, astro-ph/9604025) for improvement
see Waldvogel lecture at Scottish University Summer School in Physics (2007)

www.sam.math.ethz.ch/~joergw/Papers/scotpaper.pdf

BASIC IDEAS:

*Change from coordinates to offset coordinates: CM and relative particle

$$x_{CM} = \frac{m_1\, x_1 \;+\; m_2\, x_2}{m_1 + m_2} \qquad x_{rel} = x_1 - x_2$$

* a Kepler orbit is transformed into a **harmonic oscillator** and the number of steps needed for the integration of an orbit is reduced significantly & round-off errors reduce too

3.3 REGULARIZATION

Regularisation for binaries and 3-body encounters:

Kustaanheimo-Stiefel (KS) regularisation
AKA PERTURBED KEPLER PROBLEM

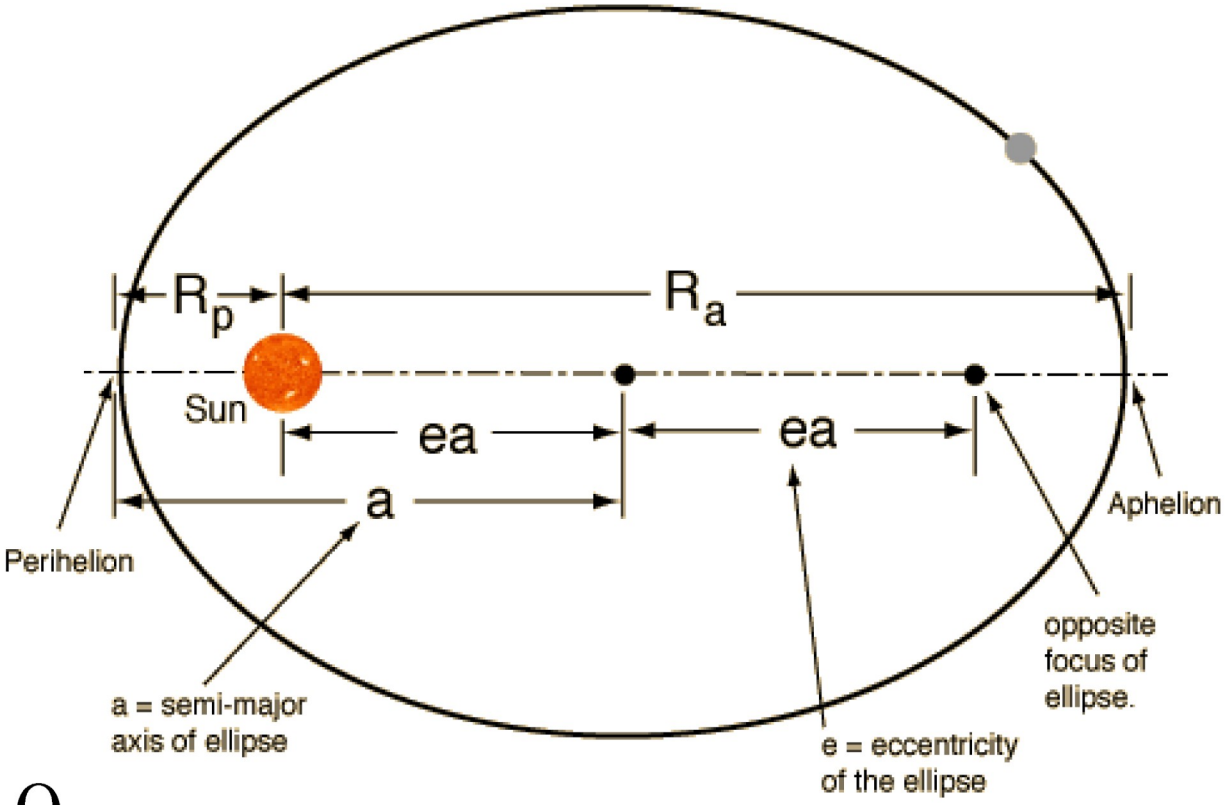
Let us consider a Kepler binary (eg Sun+planet)
M1 = Sun mass
M2 = planet mass

Total mass:
Mtot = M1+M2

Reduced mass:
μ = M1 M2/(M1+M2)

equation of Kepler motion for reduced Mass:

$$\frac{d^2\vec{r}}{dt^2} + G\,\mu\frac{\vec{r}}{r^3} = 0$$



$R_a = a(1+e)$ $R_p = a(1- e)$

3.3 REGULARIZATION

Regularisation for binaries and 3-body encounters:

Kustaanheimo-Stiefel (KS) regularisation
AKA PERTURBED KEPLER PROBLEM

CALCULATIONS (for Levi-Civita in 2D – KS is the same in 3D):

1- equation of Kepler motion for reduced mass

$$\frac{d^2\vec{r}}{dt^2} + G\,\mu\frac{\vec{r}}{r^3} = 0$$

reduced mass

2- total energy of binary:

$$\frac{1}{2}\left|\frac{d\vec{r}}{dt}\right|^2 - \frac{G\,\mu}{r} = -h \text{ , where } h = \frac{G\,\mu}{2\,a}$$

Binding energy

semi-major axis

3.3 REGULARIZATION

CALCULATIONS (for Levi-Civita in 2D – KS is the same in 3D):

3- change time coordinate (for infinitesimally small steps):

$$dt = \frac{r}{\xi} d\tau$$

WHERE $\xi = \sqrt{\frac{G\,\mu}{a}}$

THEN $$\frac{d^2}{dt^2} = \xi^2 \left(r^{-2} \frac{d^2}{d\tau^2} + \left(\frac{d\xi}{d\tau} \frac{r}{\xi} - \frac{dr}{d\tau} \right) r^{-3} \frac{d}{d\tau} \right)$$

4- represent the physical coordinates $\vec{r}$ as the square $\vec{u}^2$ of a complex variable

$\vec{u} = u_1 + i\, u_2$

$$\vec{r} = \vec{u}^2 \qquad r = |\vec{u}|^2 = \vec{u}\vec{u}$$

3.3 REGULARIZATION

CALCULATIONS (for Levi-Civita in 2D – KS is the same in 3D):

5- substituting 3 and 4 in 1 (Kepler equation) and 2 (binary energy), and using properties of complex numbers:

1 becomes

(*) $$2\,r\,\frac{d^2\vec{u}}{d\tau^2} + 2\,\frac{d\xi}{d\tau}\,\frac{r}{\xi}\frac{d\vec{u}}{d\tau} + \left(\frac{G\,\mu}{\xi^2} - 2\left|\frac{d\vec{u}}{d\tau}\right|^2\right)\,\vec{u} = 0$$

2 becomes

(**) $$2\,\xi^2\left|\frac{d\vec{u}}{d\tau}\right|^2 = G\,\mu - r\,h$$

WE CAN USE THE (**) TO REMOVE THE $\left|\frac{d\vec{u}}{d\tau}\right|^2$ TERM IN (*)

3.3 REGULARIZATION

CALCULATIONS (for Levi-Civita in 2D – KS is the same in 3D):

6- The Kepler equation becomes:

$$2\,\xi^2\,\frac{d^2\vec{u}}{d\tau^2} + 2\,\xi\,\frac{d\xi}{d\tau}\frac{d\vec{u}}{d\tau} + h\,\vec{u} = 0$$

$$\downarrow \text{IF } \frac{d\xi}{d\tau} = 0$$

$$\boxed{2\,\xi^2\,\frac{d^2\vec{u}}{d\tau^2} + h\,\vec{u} = 0}$$

EQUATION OF HARMONIC OSCILLATOR (NO SINGULARITY)!

$$\frac{d\xi}{d\tau} \equiv \frac{d\sqrt{2\,h}}{d\tau} = 0$$

CASE of UNPERTURBED BINARY: ENERGY DOES NOT CHANGE

BUT $$\frac{d\xi}{d\tau} \equiv \frac{d\sqrt{2\,h}}{d\tau} \neq 0$$

CASE of PERTURBED BINARY: 3-BODY ENCOUNTER

3.3 REGULARIZATION

Regularisation for multi-body systems:

CHAIN regularisation by Aarseth

(e.g. Mikkola & Aarseth 1993, Celestial Mechanics and Dynamical Astronomy, 57, 439)

USEFUL for PLANETARY SYSTEMS and for the surrounding of SUPER-MASSIVE BLACK HOLES (where multiple interactions with a dominant body are frequent)

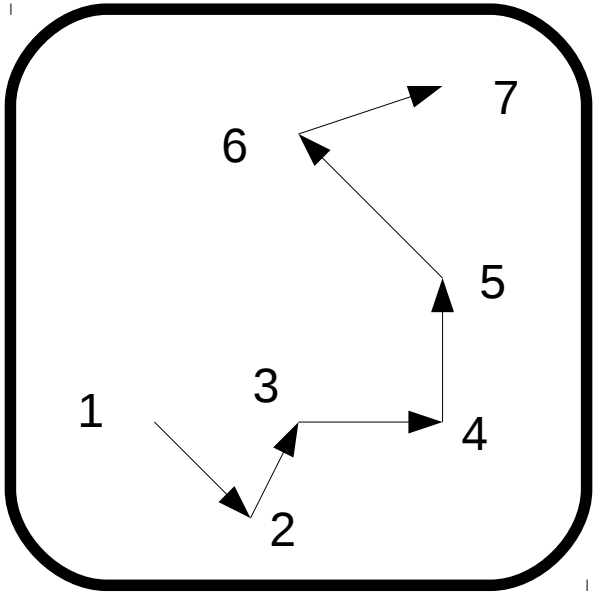
BASIC IDEAS:

- calculate distances between an active object (e.g. binary) and the closest neighbours

- find vectors that minimize the distances

- use these vectors ("**chain coordinates**") to change coordinates and make SUITABLE CHANGE OF TIME COORDINATE

- calculate forces with new coordinates



4. WHERE? THE HARDWARE – from GRAPE to GPUs

4.1 GRAPE (see http://www.ids.ias.edu/~piet/act/comp/hardware/index.html)

GRAvity PipE: a hardware implementation of Newtonian pair-wise force calculations between particles in a self-gravitating N-body system

HIGHLY SPECIALIZED HARDWARE, FASTER than LIBRARY CALL TO GRAVITY CALCULATION ROUTINE

SORT of GRAVITY ACCELERATOR

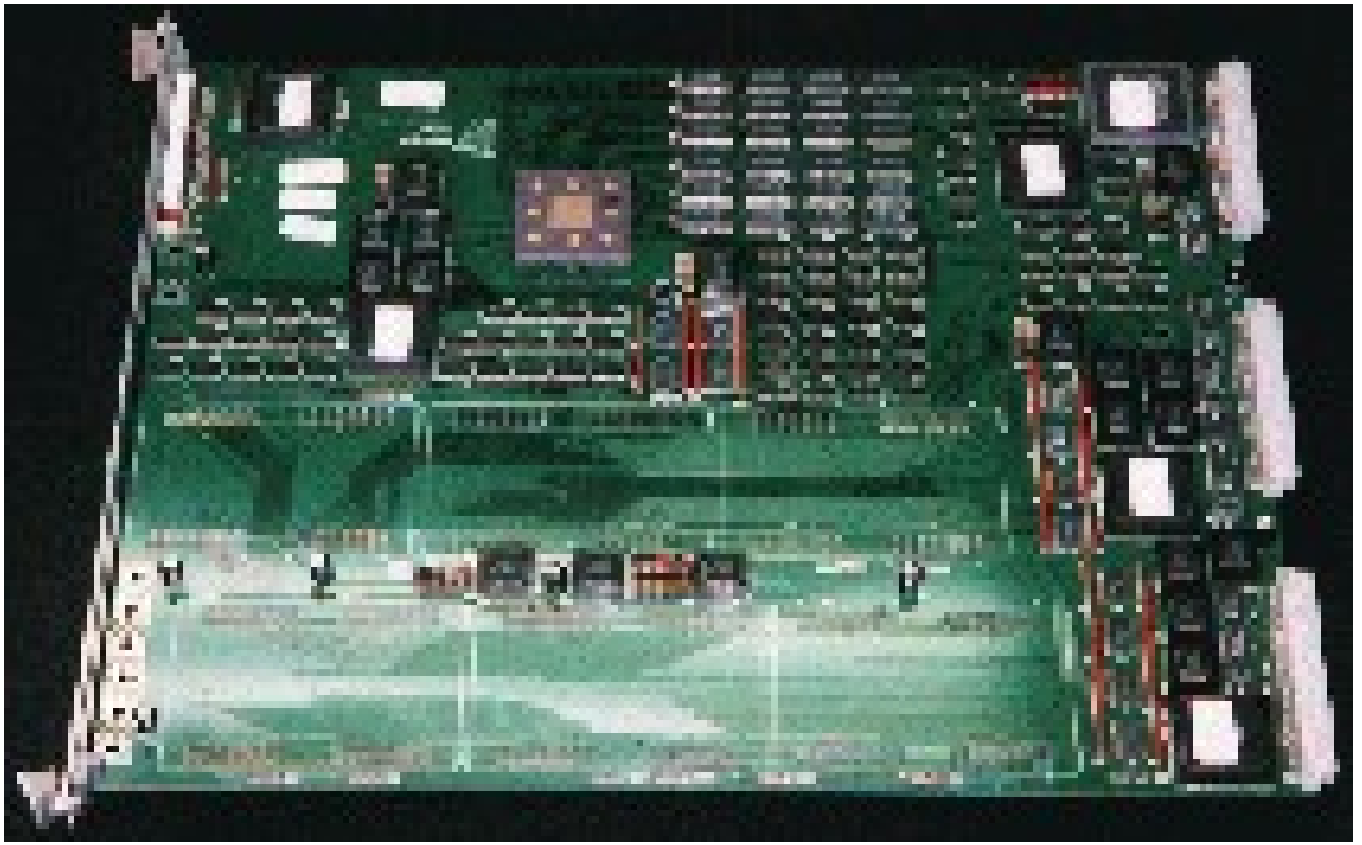
as a

GRAPHICS CARD is a GRAPHICS ACCELERATOR

– – – – – – – – – – – – –

Predictor/corrector on PC

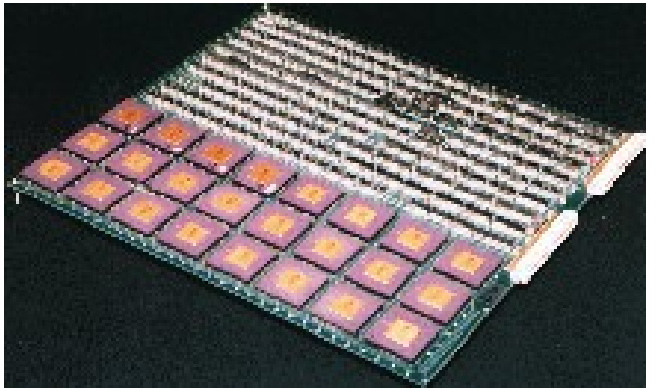
Acceleration and



4.1 GRAPE (see http://www.ids.ias.edu/~piet/act/comp/hardware/index.html)

History:

1989: GRAPE project starts at Tokyo university (Daiichiro Sugimoto and then Junichiro Makino)



GRAPE-1 at 240 Mflops at single precision

1990: GRAPE-2 at 40 Mflops at double pr.

1991: GRAPE-3 at 15 Gflops at single pr.

(first one with specialized gravity chips rather than commercial chips)

1995: GRAPE-4 at double pr.
4-cabinet GRAPE-4 computer reaches 1Tflop
!!! 1st computer who reached 1Tflop !!!

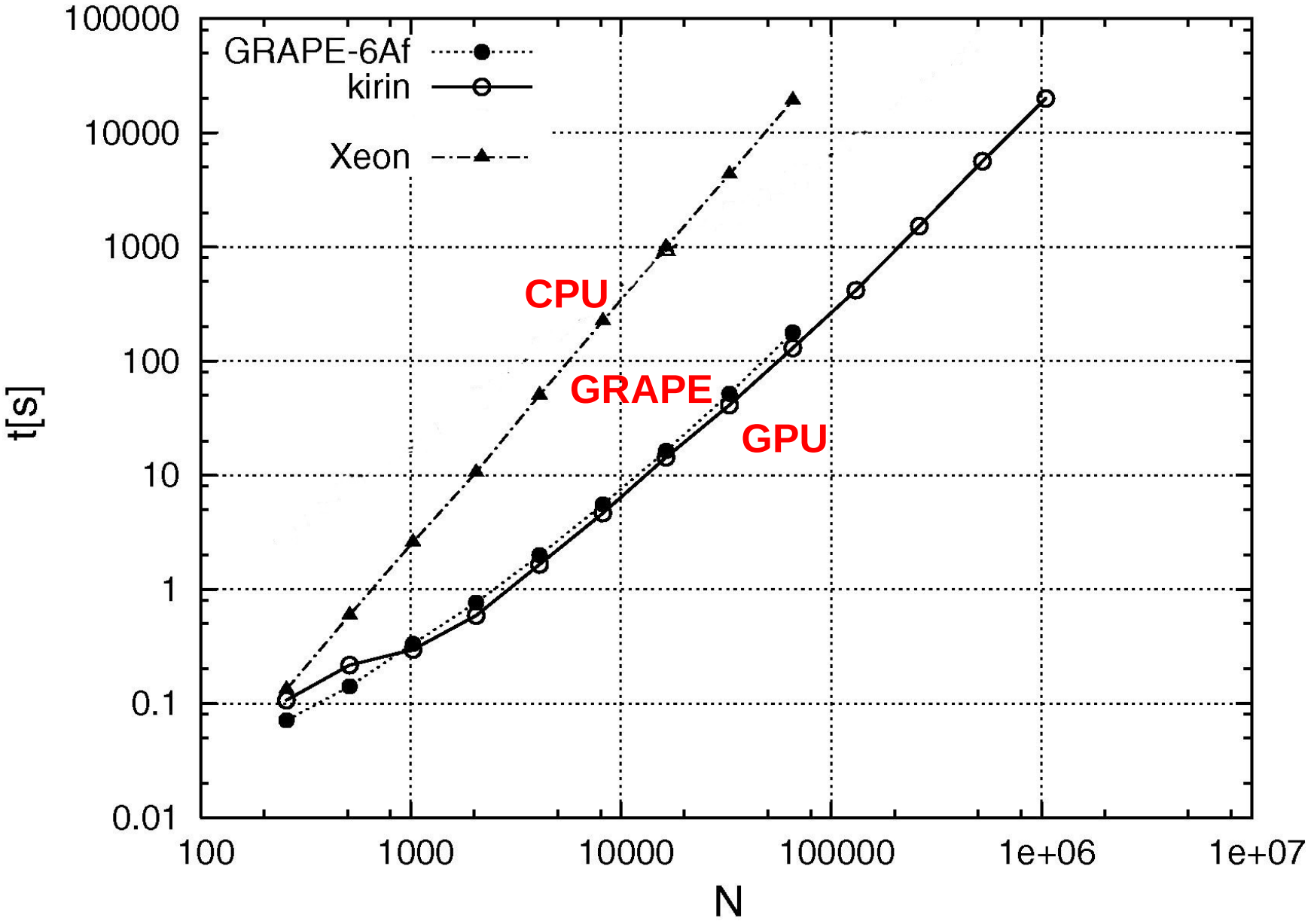
2001: GRAPE-6 at double pr.

A single GRAPE-6 boards runs at 1 Tflop
A 4-cabinet (with 8 GRAPE-6 boards each) at 32 Tflop

GRAPE-8 was in project but.....

4.2 GRAPHICS PROCESSING UNITS (GPUs)

In 2004-2008, researchers found that GPUs are at least as fast as GRAPES for direct N-body codes (Portegies Zwart et al. 2007; **Belleman et al. 2008**; Gaburov et al. 2009)



4.2 GRAPHICS PROCESSING UNITS (GPUs)

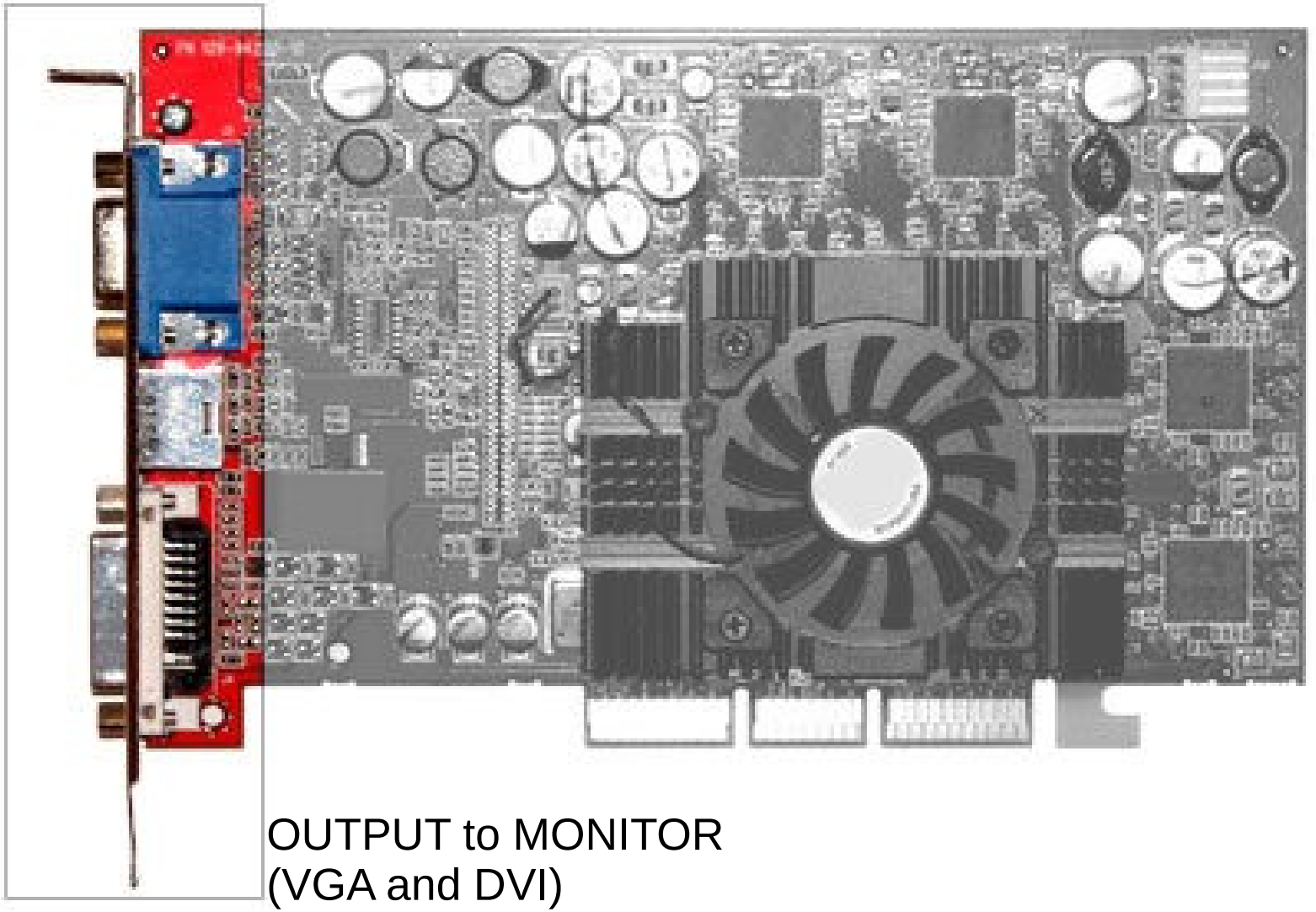
Wikipedia's definition: specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display

Mostly graphics accelerator of the VIDEO CARD, but in some PC are in the MOTHERBOARD

VIDEO CARDS WITH GPUS

4.2 GRAPHICS PROCESSING UNITS (GPUs)

COMPONENTS of a VIDEO CARD

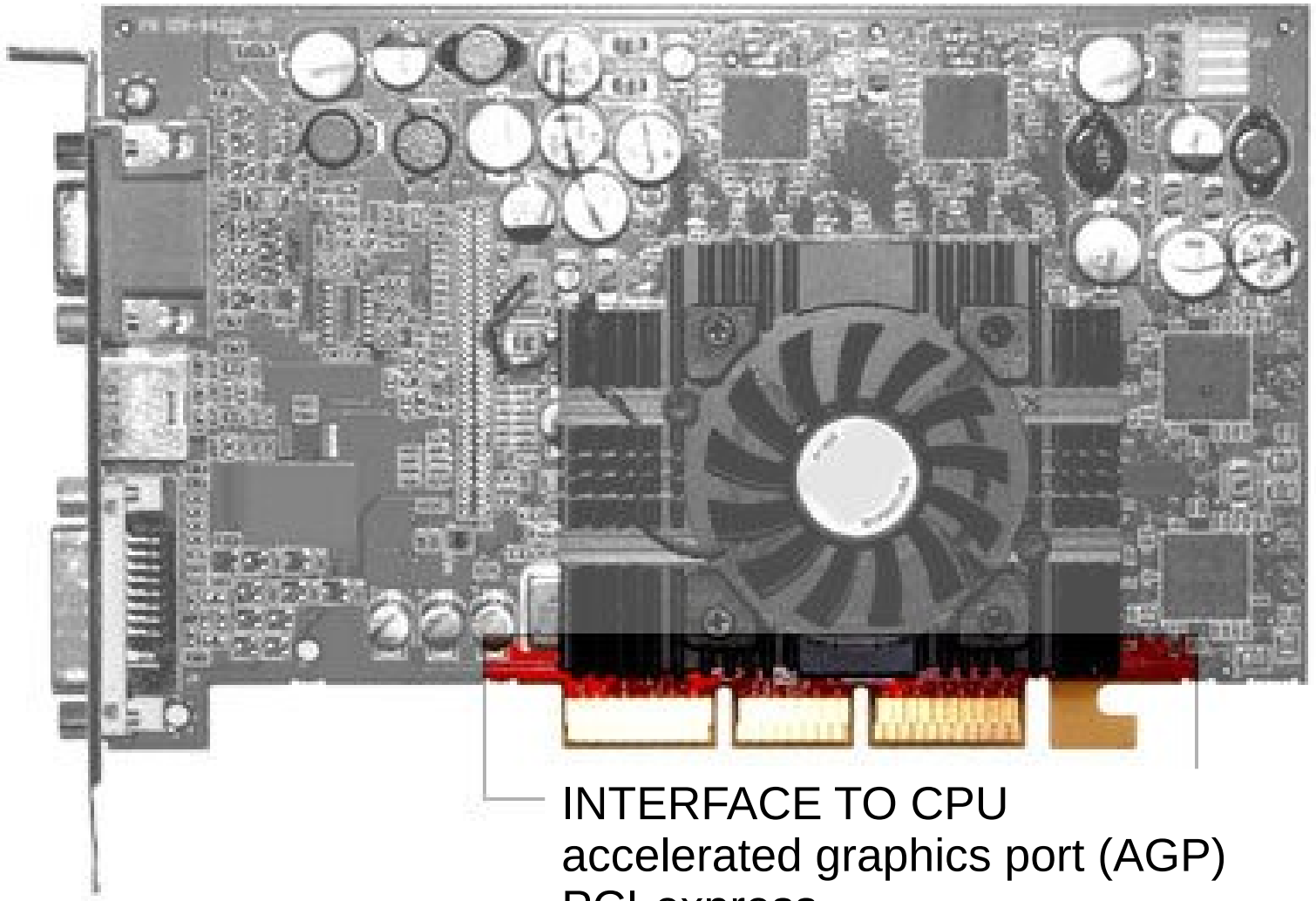


OUTPUT to MONITOR
(VGA and DVI)

From http://www.tomshardware.com/reviews/graphics-beginners,1288.html
By Don Woligroski

4.2 GRAPHICS PROCESSING UNITS (GPUs)

COMPONENTS of a VIDEO CARD

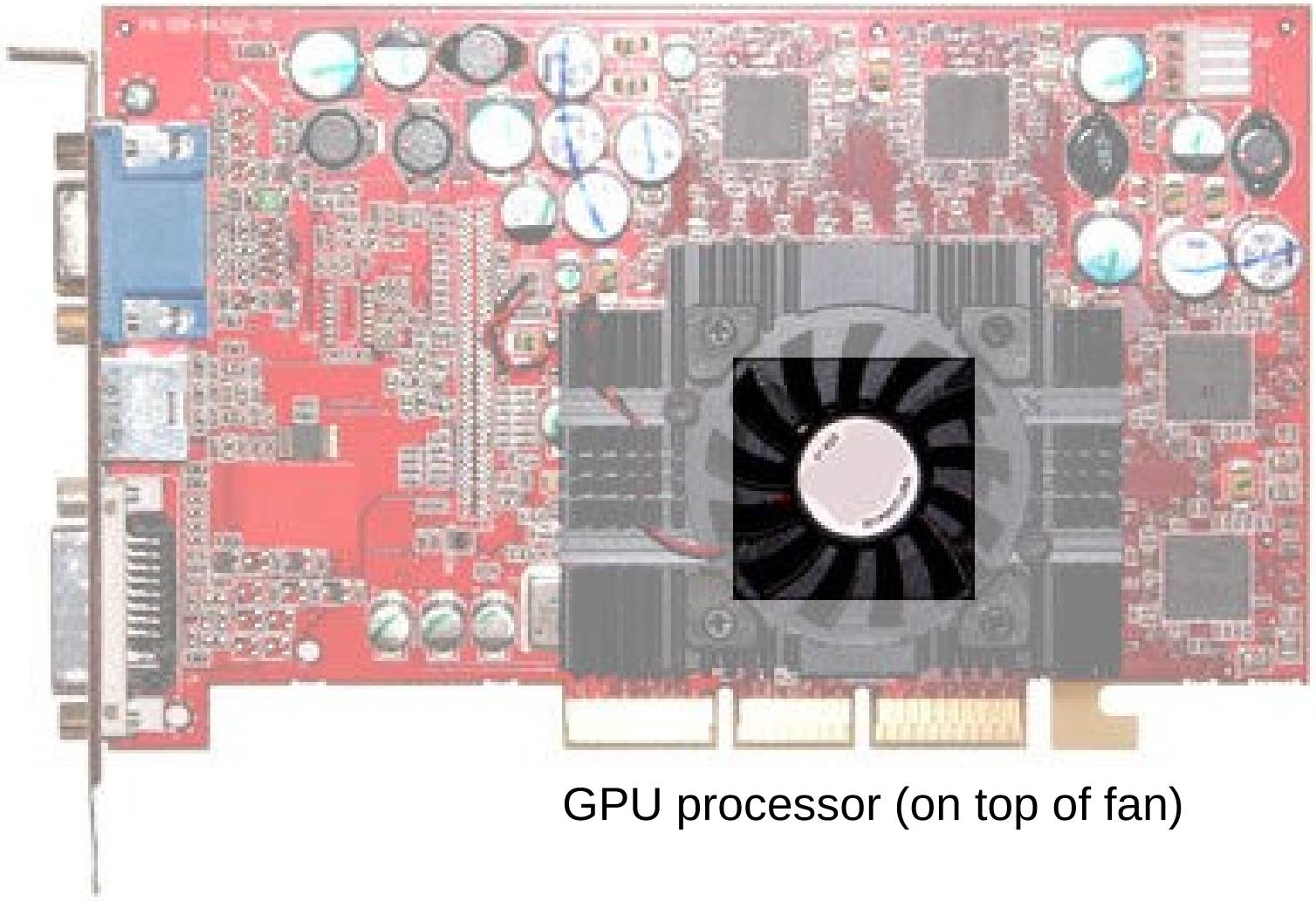


INTERFACE TO CPU
accelerated graphics port (AGP)
PCI-express

From http://www.tomshardware.com/reviews/graphics-beginners,1288.html
By Don Woligroski

4.2 GRAPHICS PROCESSING UNITS (GPUs)

COMPONENTS of a VIDEO CARD

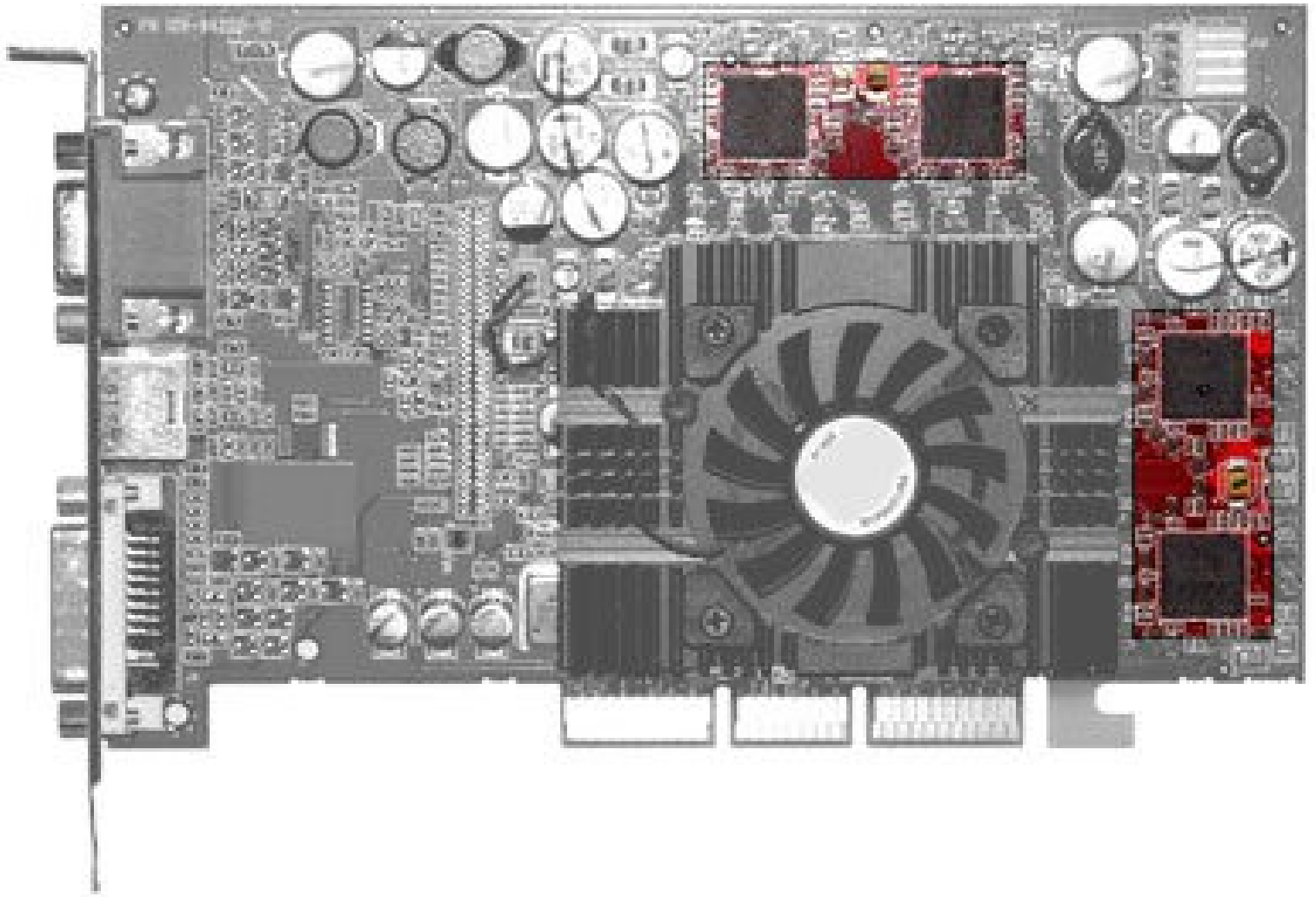


GPU processor (on top of fan)

From http://www.tomshardware.com/reviews/graphics-beginners,1288.html
By Don Woligroski

4.2 GRAPHICS PROCESSING UNITS (GPUs)

COMPONENTS of a VIDEO CARD



VIDEO

MEMORY

From http://www.tomshardware.com/reviews/graphics-beginners,1288.html
By Don Woligroski

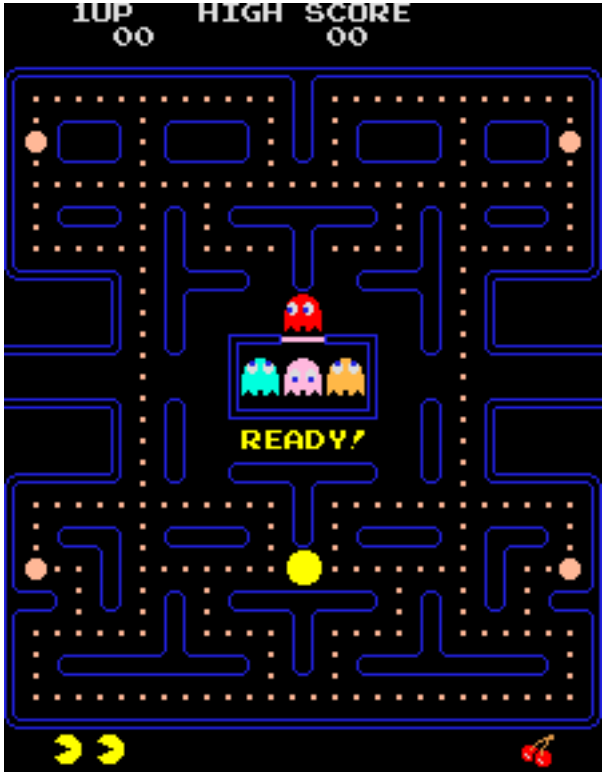
4.2 GRAPHICS PROCESSING UNITS (GPUs)

Wikipedia's definition: specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display

Mostly graphics accelerator of the VIDEO CARD, but in some PC are in the MOTHERBOARD

Born for applications that need FAST and HEAVY GRAPHICS: VIDEO GAMES

BEFORE GPU



AFTER GPU

4.2 GRAPHICS PROCESSING UNITS (GPUs)

In ~2004 GPUS WERE FOUND TO BE USEFUL FOR CALCULATIONS:

- first N-body simulations (2^{nd} order) by Nyland et al. (2004)

- first GPU implementation of Hermite scheme by Portegies Zwart et al. (2007)

- molecular dynamics on GPU (Anderson et al. 2008; van Meel et al. 2008)

- Kepler's equation (Ford 2009)

- many more N-body: Cunbody (Hamada & Iitaka 2007), kirin (Belleman et al. 2008), Yebisu (Nitadori & Makino 2008; Nitadori 2009), Sapporo (Gaburov et al. 2009, Bedorf et al. 2015)

WHY?

4.2 GRAPHICS PROCESSING UNITS (GPUs)

SIMPLE IDEA:

coloured pixel represented by 4 numbers (R, G, B and transparency)

each pixel does not need information about other pixels (near or far)

→ **when an image must be changed each single pixel can be updated INDEPENDENTLY of the others and SIMULTANEOUSLY to the others**

→ **GPUs are optimized to perform MANY SMALL OPERATIONS (change a single pixel) SIMULTANEOUSLY i.e. MASSIVELY PARALLEL**

THIS IS THE CONCEPT OF **SIMD** TECHNIQUE:

SINGLE INSTRUCTION MULTIPLE DATA

GPUS are composed of many small threads, each able to perform a small instruction (**kerne**l), which is the same for all threads but applied on different data

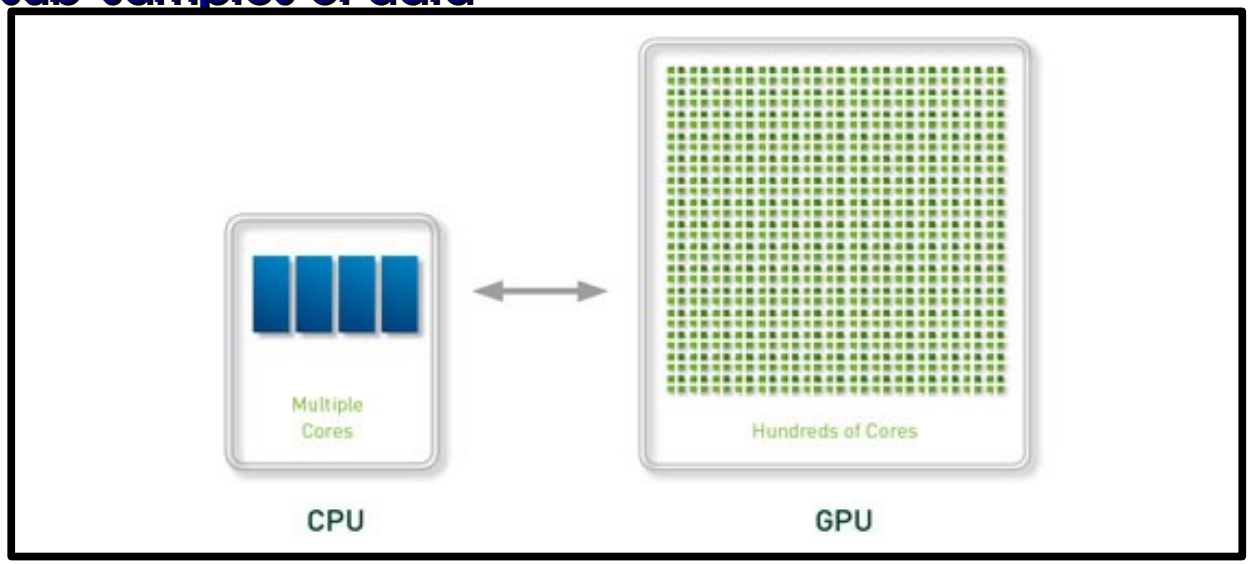
→ NVIDIA calls it **SIMT**= single instruction multiple **THREAD**

4.2 GRAPHICS PROCESSING UNITS (GPUs)

SIMD/SIMT TECHNIQUE: SINGLE INSTRUCTION MULTIPLE DATA/THREADS

many processing units perform the same series of operations

on different sub-samples of data



Even current CPUs are multiple CORES (i.e. can be multi-threading)

but the number of independent cores in GPUs is ~100 times larger!

1M $ QUESTION: WHY IS THIS PARTICULARLY GOOD FOR DIRECT N-BODY CODES?

4.2 GRAPHICS PROCESSING UNITS (GPUs)

SIMD TECHNIQUE: SINGLE INSTRUCTION MULTIPLE DATA

WHY IS THIS PARTICULARLY GOOD FOR DIRECT N-BODY CODES?

BECAUSE THEY DO A SINGLE OPERATION

(acceleration and jerk calculation)

on MANY PAIRS of PARTICLES

$$\vec{a}_i = G \sum_{j \neq i} \frac{M_j}{r_{ji}^3} \vec{r_{ij}}$$

EACH INTERPARTICLE FORCE BETWEEN A PAIR IS INDEPENDENT OF THE OTHER PAIRS!!

SINGLE INSTRUCTION: ACCELERATION CALCULATION

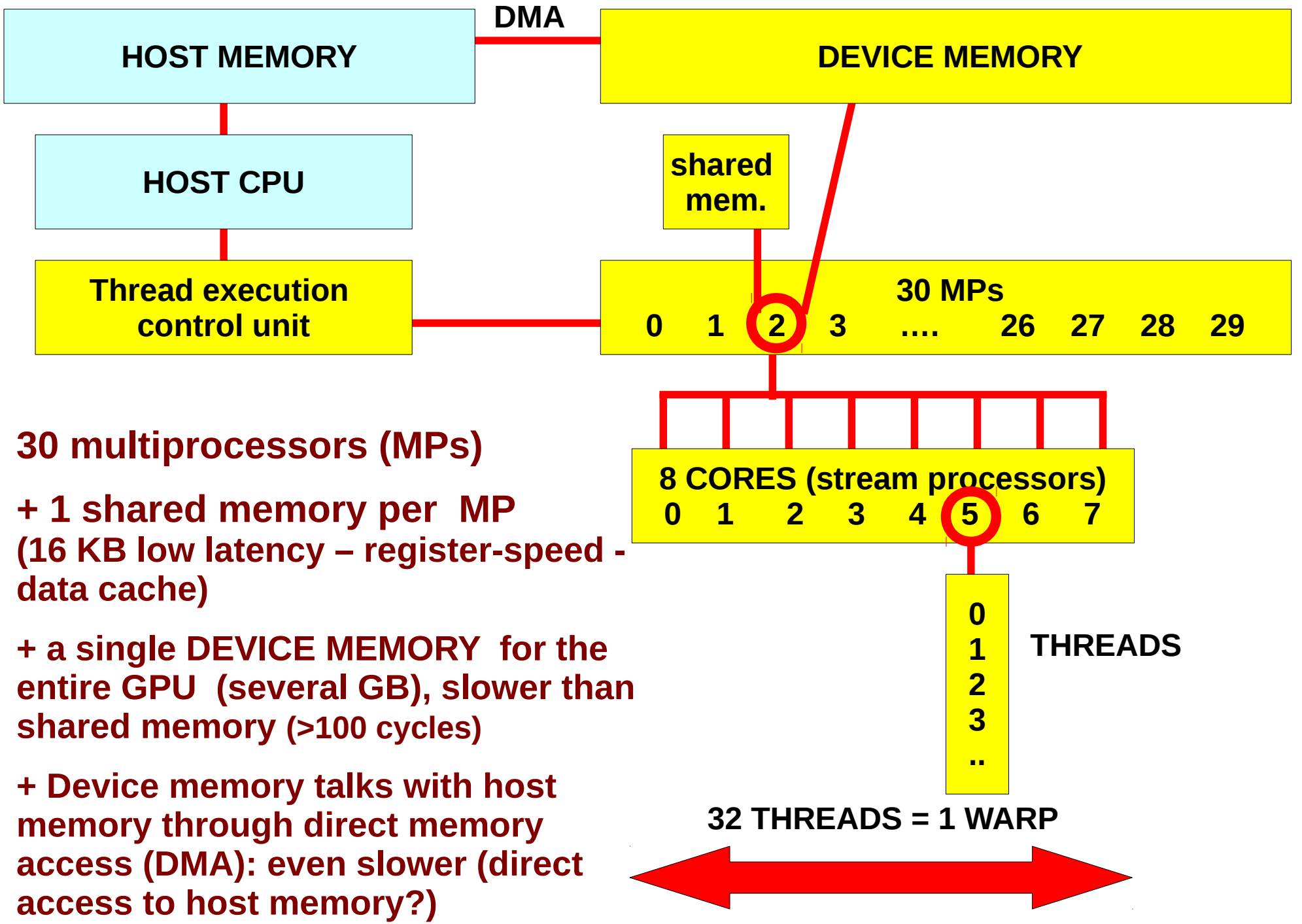
MULTIPLE DATA: N (N-1)/2 ~ N^2 FORCES

4.2 GRAPHICS PROCESSING UNITS (GPUs)

HOW ARE DIRECT N-BODY CODES ADAPTED TO GPUs?

1. inside the GPU
2. languages for GPU computing
3. application to the Hermite scheme

1. inside the GPU – Host := CPU, Device := GPU – EXAMPLE: Tesla C1060



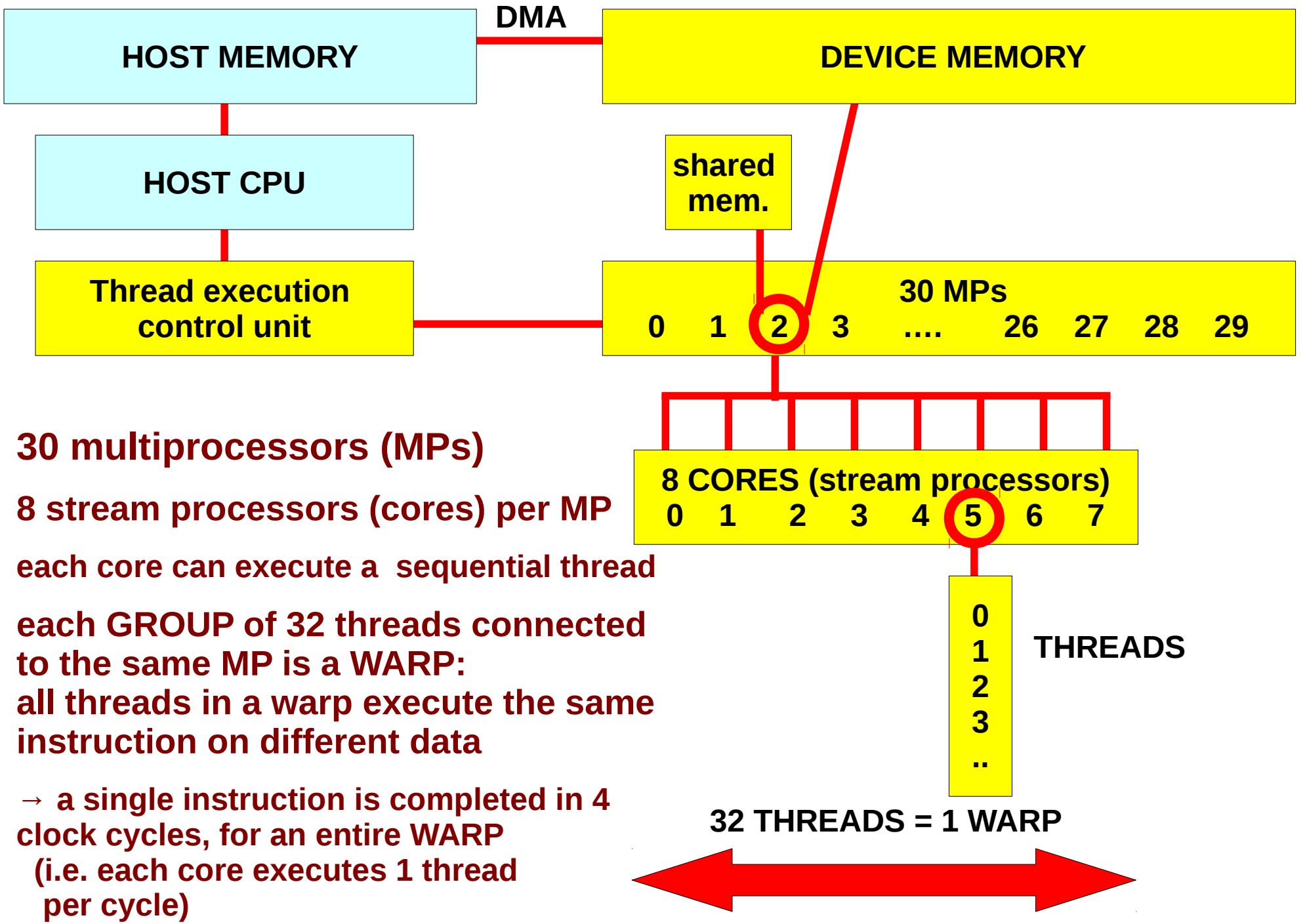
30 multiprocessors (MPs)

+ 1 shared memory per MP (16 KB low latency – register-speed - data cache)

+ a single DEVICE MEMORY for the entire GPU (several GB), slower than shared memory (>100 cycles)

+ Device memory talks with host memory through direct memory access (DMA): even slower (direct access to host memory?)

1. inside the GPU – Host := CPU, Device := GPU – EXAMPLE: Tesla C1060



30 multiprocessors (MPs)

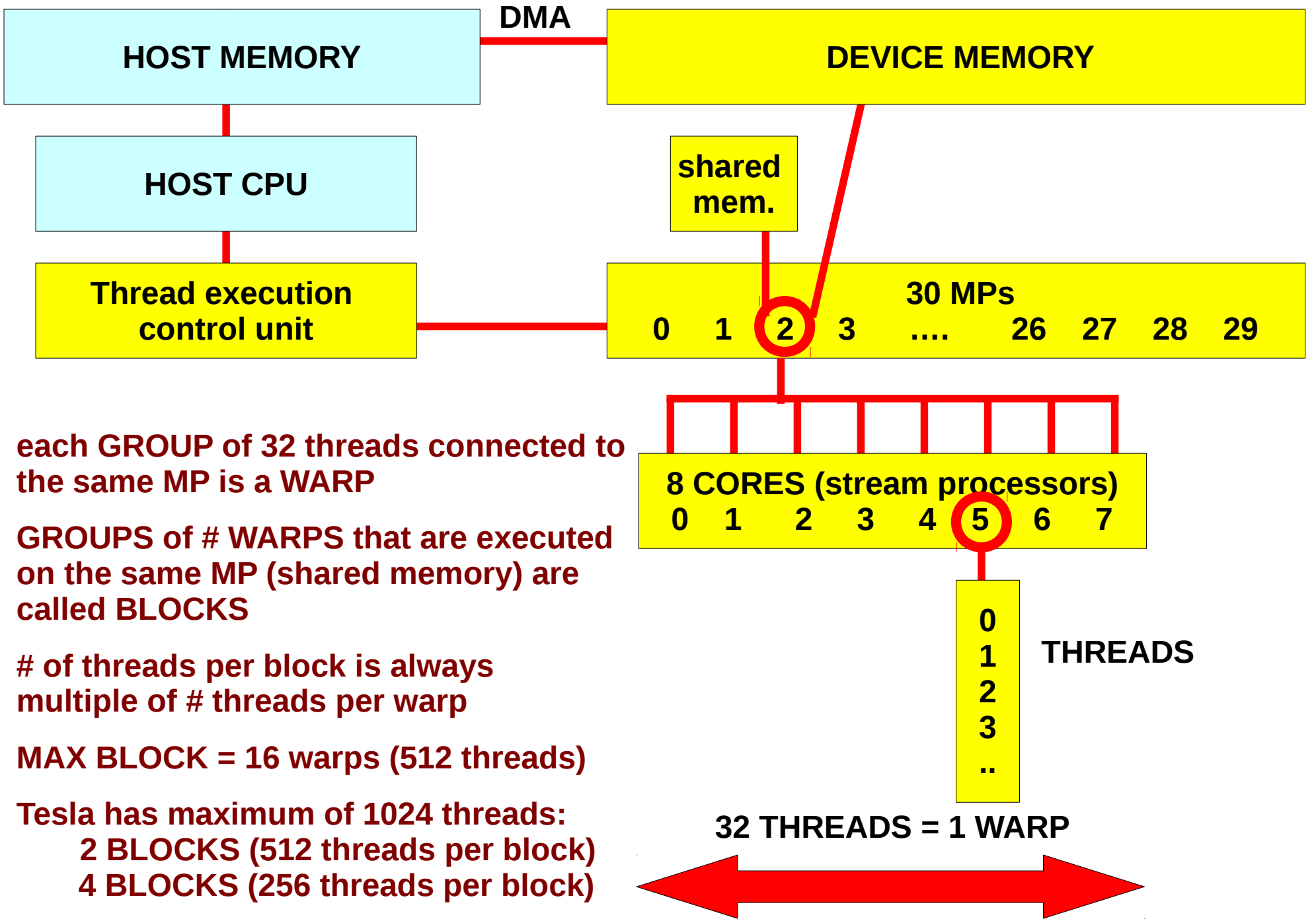
8 stream processors (cores) per MP

each core can execute a sequential thread

each GROUP of 32 threads connected to the same MP is a WARP: all threads in a warp execute the same instruction on different data

→ a single instruction is completed in 4 clock cycles, for an entire WARP (i.e. each core executes 1 thread per cycle)

1. inside the GPU – Host := CPU, Device := GPU – EXAMPLE: Tesla C1060



each GROUP of 32 threads connected to the same MP is a WARP

GROUPS of # WARPS that are executed on the same MP (shared memory) are called BLOCKS

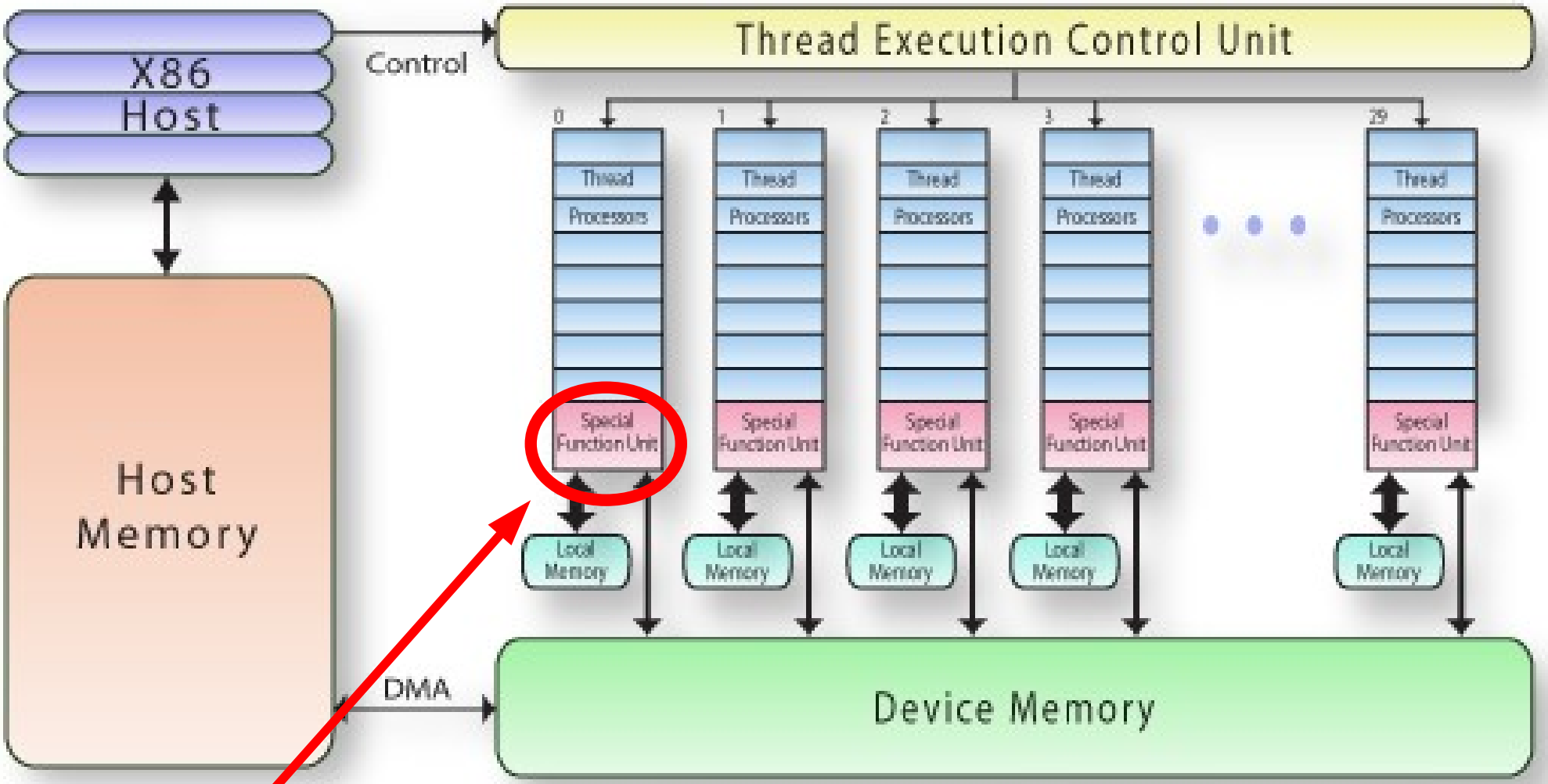
# of threads per block is always multiple of # threads per warp

MAX BLOCK = 16 warps (512 threads)

Tesla has maximum of 1024 threads:

- 2 BLOCKS (512 threads per block)
- 4 BLOCKS (256 threads per block)
- ....

1. inside the GPU – Host := CPU, Device := GPU – EXAMPLE: Tesla C1060



GPUs were born single precision. In some recent GPUs (eg TESLA) each MP has a 'special function unit' to mimic double precision → important for science calculation

2. languages for GPU computing

- Cg = C for graphics computer language (Fernando & Kilgard 2003)
  for use with open graphics library (Open GL)

  eg the kirin (Belleman et al. 2008) N-body library is in Cg
  https://developer.nvidia.com/cg-toolkit

- CUDA= Compute Unified Device Architecture (Fernando 2004)
  for use with NVIDIA proprietary drivers

  Also similar to C/C++

  eg the Sapporo library for N-body (Gaburov et al. 2009)
https://developer.nvidia.com/get-started-cuda-cc

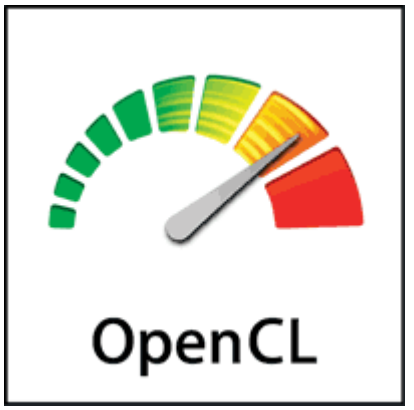
Both Cg and CUDA are developed by NVIDIA

- Open CL = born 2009, for use with open graphics library (Open GL)
  similar to C

  OPEN SOURCE AND

  NO LIMITS ON DEVICE (even intel phi)

  Developed by Apple, AMD, Intel, IBM...



3. application to the Hermite scheme

EXAMPLE: Sapporo library for N-body
(Gaburov et al. 2009, http://arxiv.org/abs/0902.4463,
Bedorf et al. 2015, http://arxiv.org/abs/1510.04068)

Public software – download:
http://home.strw.leidenuniv.nl/~spz/MODESTA/Software/src/sapporo.html

BASIC IDEA: allows a code that uses Hermite scheme optimized for GRAPE

to run on multiple GPUS through CUDA architecture

e.g. works with

phiGRAPE (Harfst et al. 2007, New Astronomy, 12, 357)

http://www-astro.physik.tu-berlin.de/~harfst/index.php?id=phigrape

STARLAB (Portegies Zwart et al. 2001, MNRAS, 321, 199)

http://www.sns.ias.edu/~starlab/

3. application to the Hermite scheme: Sapporo library for N-body

Let us repeat the basic concepts..

acceleration $$\vec{a}_i = G \sum_{j \neq i} \frac{M_j}{r_{ji}^3} \vec{r_{ij}}$$

jerk $$\frac{d\vec{a}_i}{dt} = \vec{j}_i = G \sum_{j \neq i} M_j \left[\frac{\vec{v}_{ji}}{r_{ji}^3} - 3 \frac{(\vec{r}_{ji} \cdot \vec{v}_{ji})\ \vec{r}_{ji}}{r_{ji}^5} \right]$$

4^{th} order Hermite predictor-corrector scheme is 3 step:

1. **predictor step: predicts positions and velocities at 3^{rd} order**
2. **calculation step: calculates acceleration and jerk for the predicted positions and velocities**
3. **corrector step: corrects positions and velocities using the acceleration and jerk calculated in 2**

3. application to the Hermite scheme: Sapporo library for N-body

IF BLOCK TIME STEP OR SIMILAR IS USED:

j- particles: sources of gravitational forces (those that exert the force)

$$\Sigma j = n$$

i- particles: sinks of gravitational forces (those on which the force is exerted)

$$\Sigma i = m$$

IMPORTANT:

m<=n because ONLY ACTIVE PARTICLES ARE CORRECTED in the HERMITE PREDICTOR-CORRECTOR !!!

Even m<<n is possible

3. application to the Hermite scheme: Sapporo library for N-body

Let us repeat the basic concepts..

acceleration $$\vec{a}_i = G \sum_{j \neq i} \frac{M_j}{r_{ji}^3} \vec{r_{ij}}$$

jerk $$\frac{d\vec{a}_i}{dt} = \vec{j}_i = G \sum_{j \neq i} M_j \left[\frac{\vec{v}_{ji}}{r_{ji}^3} - 3 \frac{(\vec{r}_{ji} \cdot \vec{v}_{ji})\ \vec{r}_{ji}}{r_{ji}^5} \right]$$

IF BLOCK TIME STEP OR SIMILAR IS USED:

j- particles: sources of gravitational forces (those that exert the force) Σj = n

i- particles: sinks of gravitational forces (those on which the force is exerted) Σi = m

m<n because ONLY ACTIVE PARTICLES ARE CORRECTED

4th order Hermite predictor-corrector scheme is 3 step:

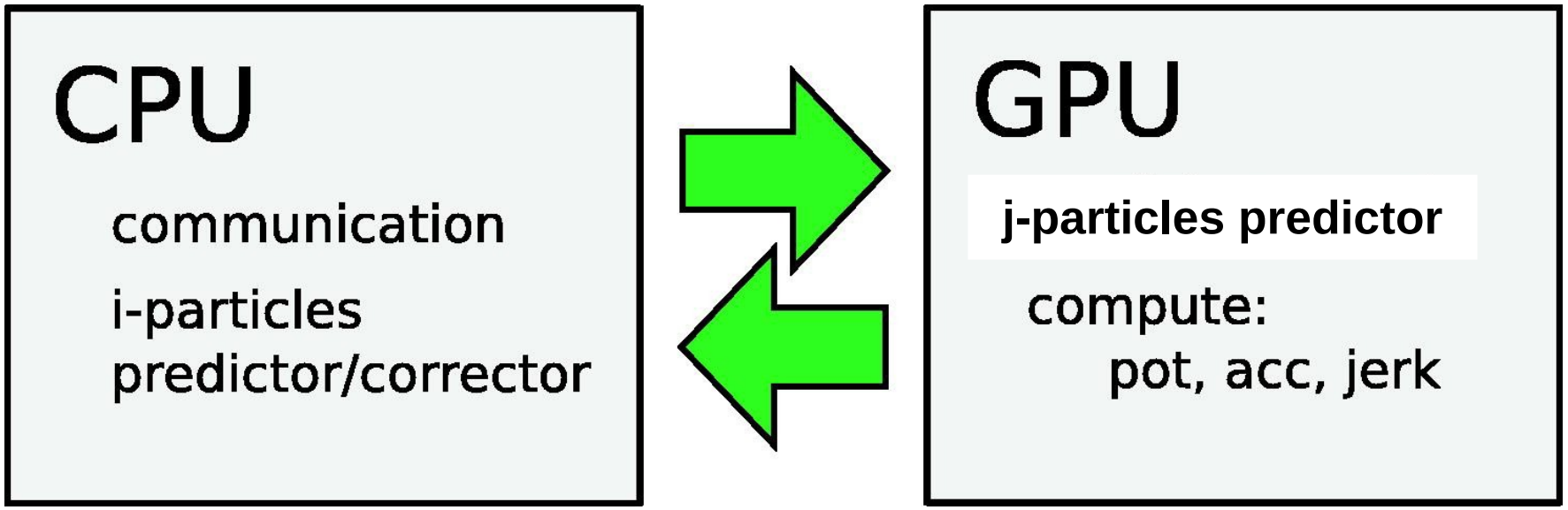
1. predictor step: predicts positions and velocities of the j-particles and i-particles at 3rd order

2. calculation step: calculates acceleration and jerk exerted by j-particles on the i-particles, for the predicted positions and velocities of the i-particles

3. corrector step: corrects positions and velocities of the i-particles using the acceleration and jerk calculated in 2

3. application to the Hermite scheme: Sapporo library for N-body

Implementation of Hermite scheme by Sapporo:



1. predictor step : j-particle predic. on GPU / i-particle predic. on CPU

2. calculation step : ENTIRELY ON GPU

3. corrector step : ENTIRELY ON CPU

WHY?

STEP 1 for the j scales as O(n) / for the i scales as O(m) with n>m

It is important that STEP 2 is on GPU because $O(n \cdot m)$

While STEP 3 is O(m) : less heavy step!

3. application to the Hermite scheme: Sapporo library for N-body

STEP 1 (predictor of j and i):

On GPU

each j-particle is read by a single thread on the GPU
position, velocity, acceleration, jerk and Δt from time 0 are read from global device memory to the local shared memory

Then prediction is done:

$$x_{p,1} = x_0 + v_0\,\Delta t + \frac{1}{2}\,a_0\,\Delta t^2 + \frac{1}{6}\,j_0\,\Delta t^3$$

$$v_{p,1} = v_0 + a_0\,\Delta t + \frac{1}{2}\,j_0\,\Delta t^2$$

Comment: positions must be in double precision (DP). This was impossible in old GPUs and is expensive in new GPUs.

Then in new GPUs only the position (and the sum to predict position) must be in DP, while v, a and j are stored in single precision (SP). The DP in GPUs is **emulated by double single (DS) technique**: a double is stored as two single p. (containing the most significant digits and the least significant ones).

On CPU

The same for i-particles

3. application to the Hermite scheme: Sapporo library for N-body

STEP 2 (calculation of acceleration and jerk onto i-particles):

On GPU

Remember: Only threads on the same MP have the same shared memory
Threads executed by different MPs share only global memory
A block is a number of threads executed by the same MP

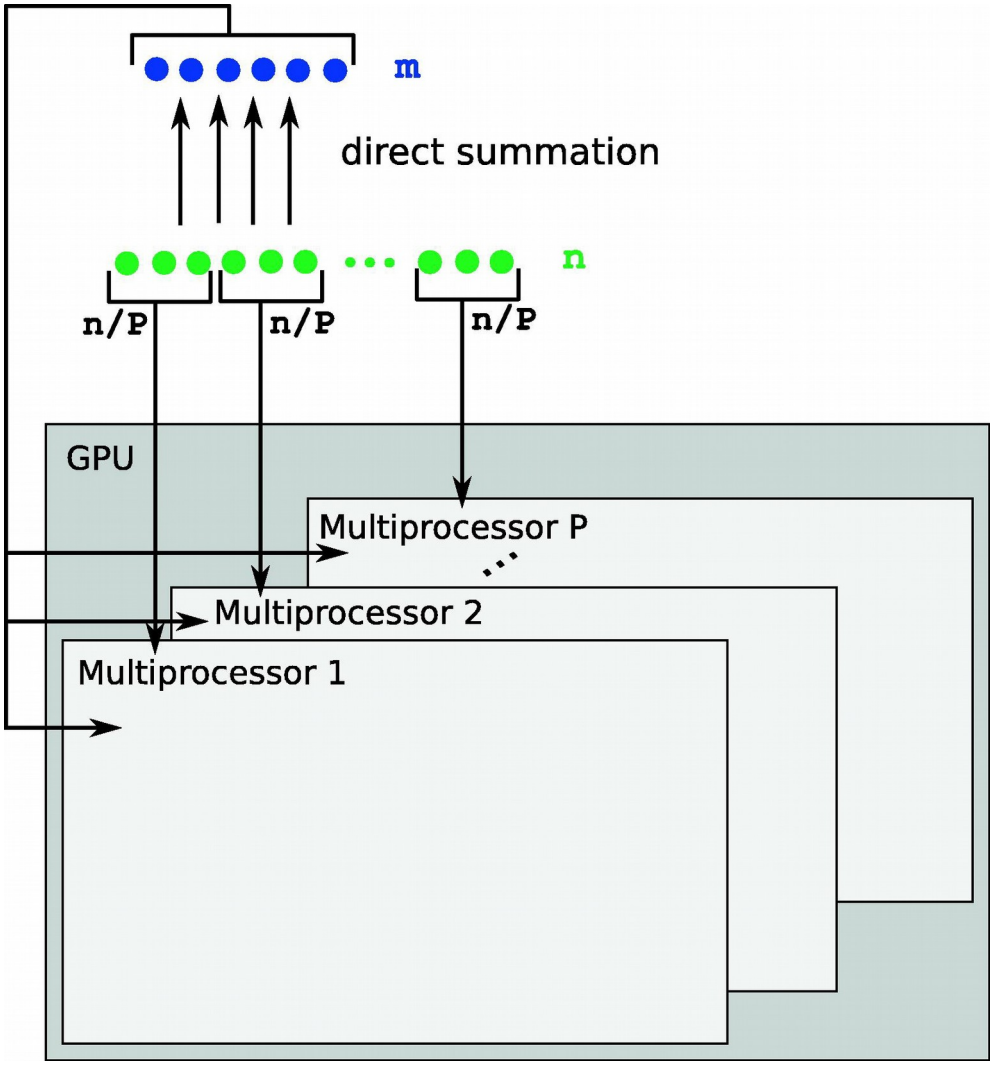
Parallelization:

the calculation is split in P blocks, where P is the # of available MPs

The j particles are distributed evenly among the P blocks (n/P per each block)

The i particles are visible to all blocks (i.e. a copy of the i-particles is sent to all MPs)

Each of the MPs computes the partial forces exerted by the n/P j-particles assigned to that MP, on all the i-particles in parallel.

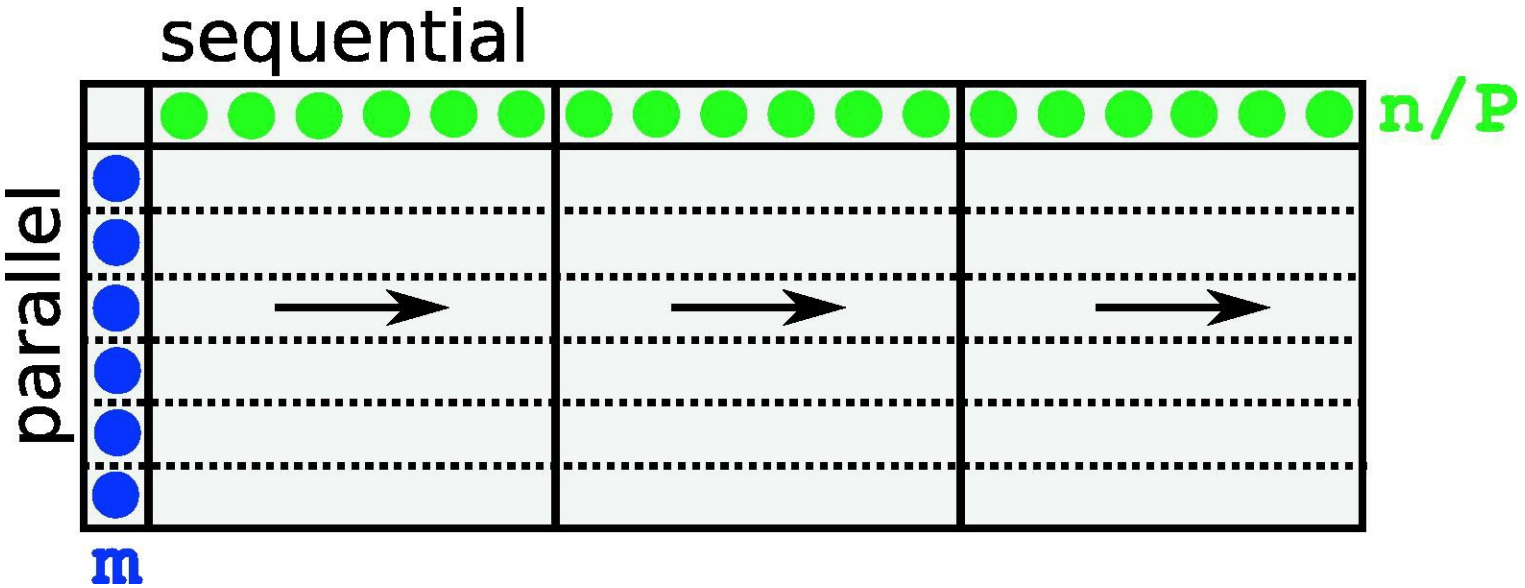


3. application to the Hermite scheme: Sapporo library for N-body

STEP 2 (continues):

if the number of threads in a block is nthread>=m each i-particle is assigned to a single thread of each block

if nthread<m, the i-particles must be split in more segments



IN PRACTICE:

*** Each thread in the same MP loads one of the i- particles from the global to the local memory (so that the total numbers of particles in the shared memory is =nthread)**

*** Each thread SEQUENTIALLY calculates and sums the partial forces exerted by the n/P j-particles stored in the block onto its associated i-particle**

*** The final step is to sum the partial forces exerted on each i-particle by each block of n/P j-particles (very last step as DIFFERENT BLOCKs communicate only through the slow GLOBAL memory)**

*** Sums are done in DS to emulate DP**

3. application to the Hermite scheme: Sapporo library for N-body

STEP 3 (correction of x and v for the i-particles):

On CPU

The total acceleration and jerks calculated on GPU are then copied from the global device memory to the host memory

In the host (=CPU) the positions and velocity of the active m particles (the i-particles) are corrected according to:

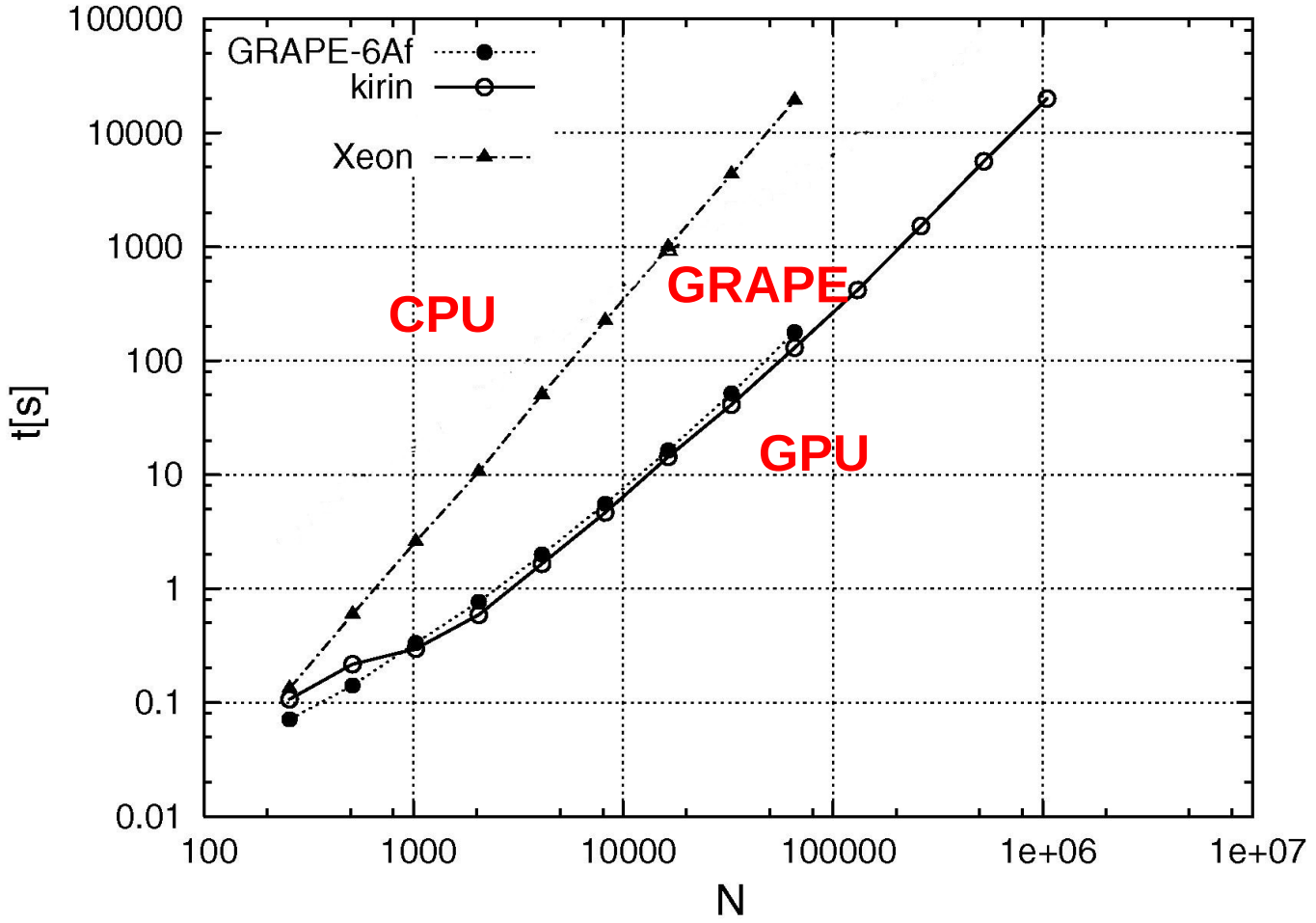
$$v_1 = v_0 + \frac{1}{2}\left(a_0 + a_{p,1}\right)\Delta t + \frac{1}{12}\left(j_0 - j_{p,1}\right)\Delta t^2$$

$$x_1 = x_0 + \frac{1}{2}\left(v_0 + v_1\right)\Delta t + \frac{1}{12}\left(a_0 - a_{p,1}\right)\Delta t^2$$

Then a new block time step Δt is calculated..etcetc

3. application to the Hermite scheme: Sapporo library for N-body

This implementation of Hermite with Sapporo allows to reach the performance I showed before:



NOTE: SAPPORO WORKS IN PARALLEL ON ALL THE GPU DEVICES CONNECTED TO THE SAME HOST thanks to the GPUWorker library, which is part of the HOOMD molecular dynamics GPU code (Anderson et al. 2008)

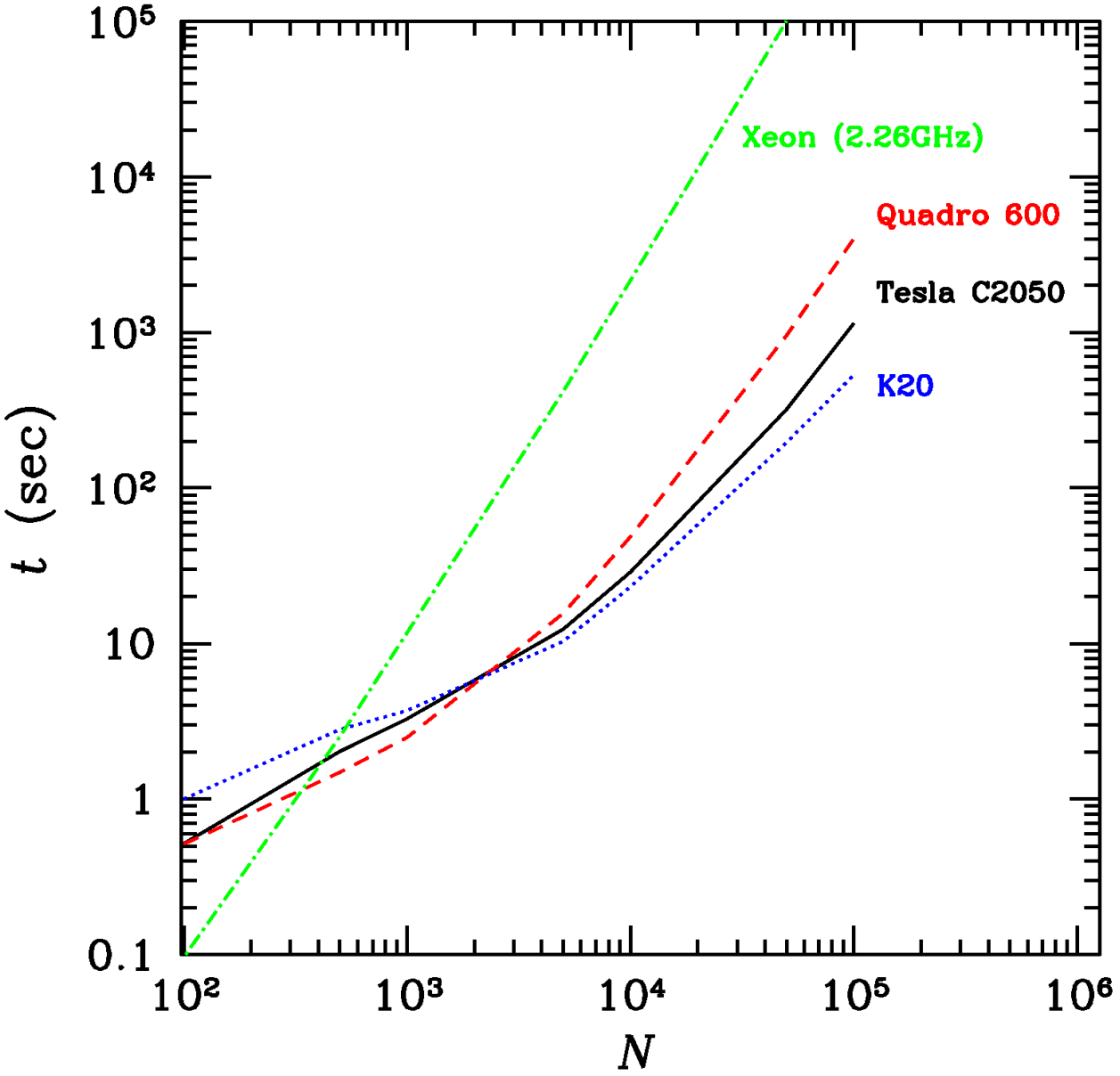
→ If each node has 2 or 4 GPUs, you can use all the 2 or 4 GPUs

3. application to the Hermite scheme: Sapporo library for N-body

This implementation of Hermite with Sapporo allows to reach the performance I showed before:

SIMPLE PERFORMANCE TEST

for a star cluster with N particles



YOU CAN RUN YOUR OWN TESTS @ HOME!

3. application to the Hermite scheme: Sapporo library for N-body

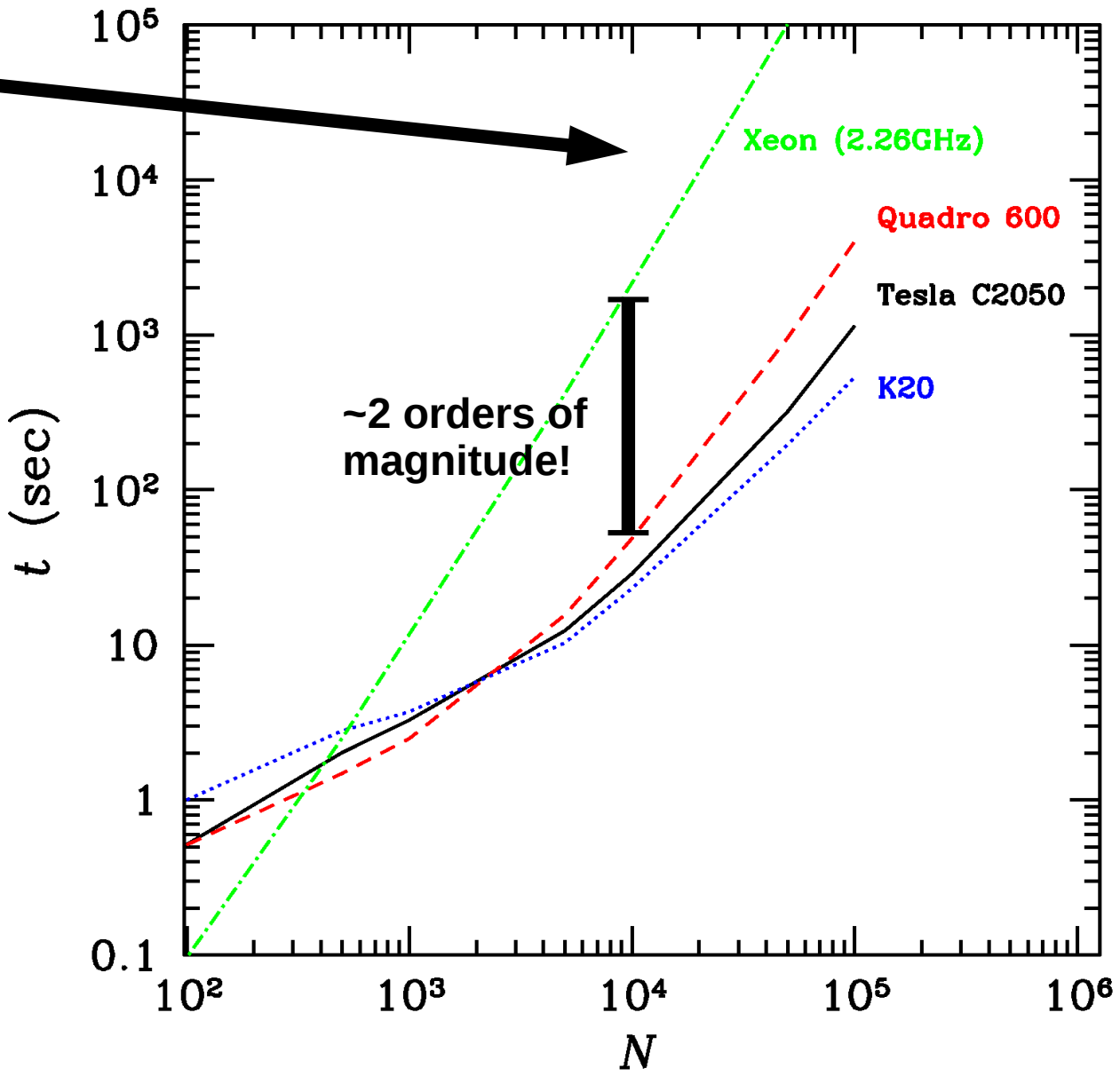
This implementation of Hermite with Sapporo allows to reach the performance I showed before:

Single Xeon processor

VS Quadro GPU (typical graphics card of desktops)

VS Tesla GPU GPU for computing: more expensive (~2k EUR) but can fit in your workstation

VS 2 Kepler 20 on the same node mounted on EURORA @ CINECA



YOU CAN RUN YOUR OWN TESTS @ HOME!

3. application to the Hermite scheme: Sapporo library for N-body

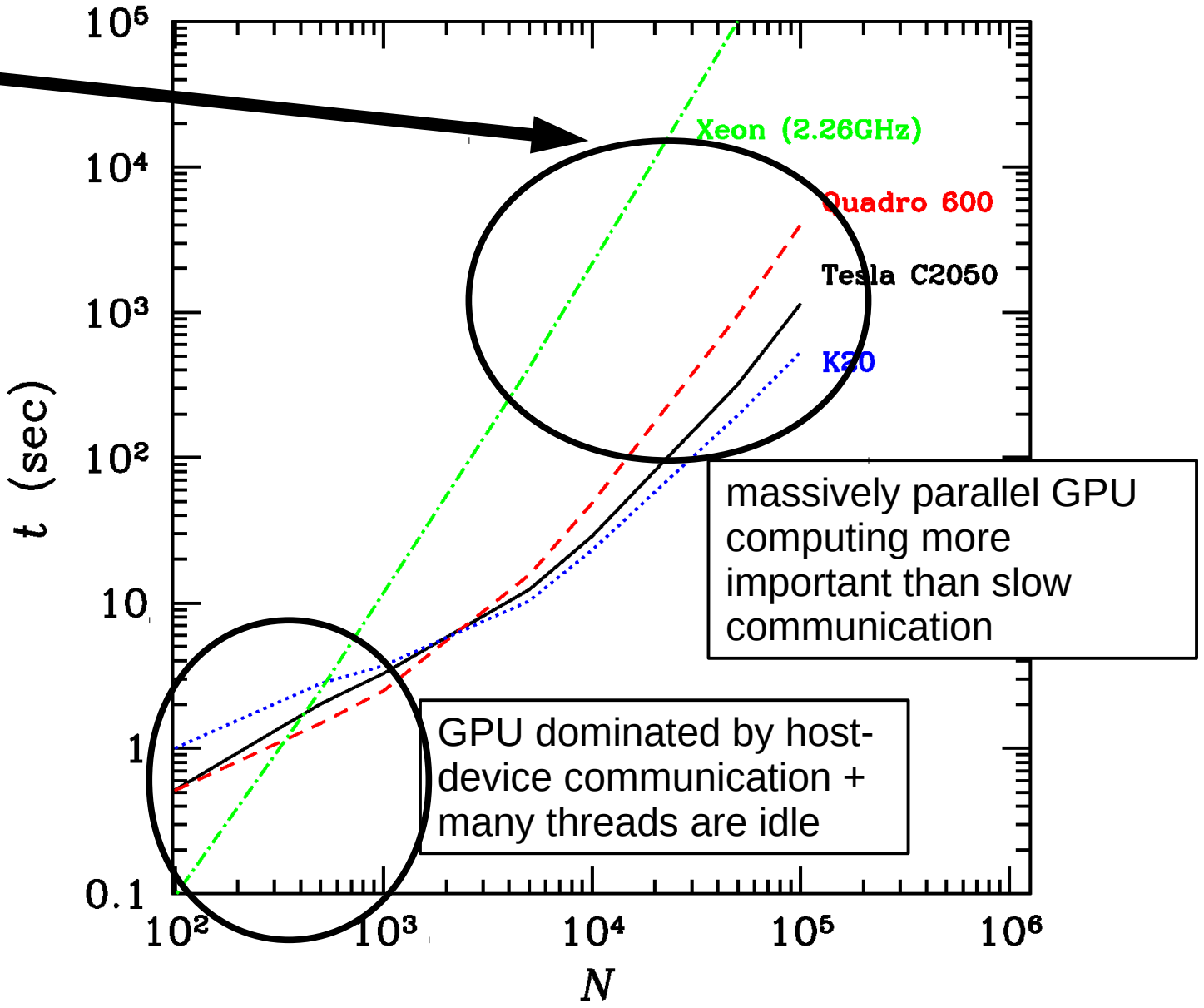
This implementation of Hermite with Sapporo allows to reach the performance I showed before:

Single Xeon processor

VS Quadro GPU (typical graphics card of desktops)

VS Tesla GPU GPU for computing: more expensive (~2k EUR) but can fit in your workstation

VS 2 Kepler 20 on the same node mounted on EURORA @ CINECA



YOU CAN RUN YOUR OWN TESTS @ HOME!

3. application to the Hermite scheme: Sapporo library for N-body

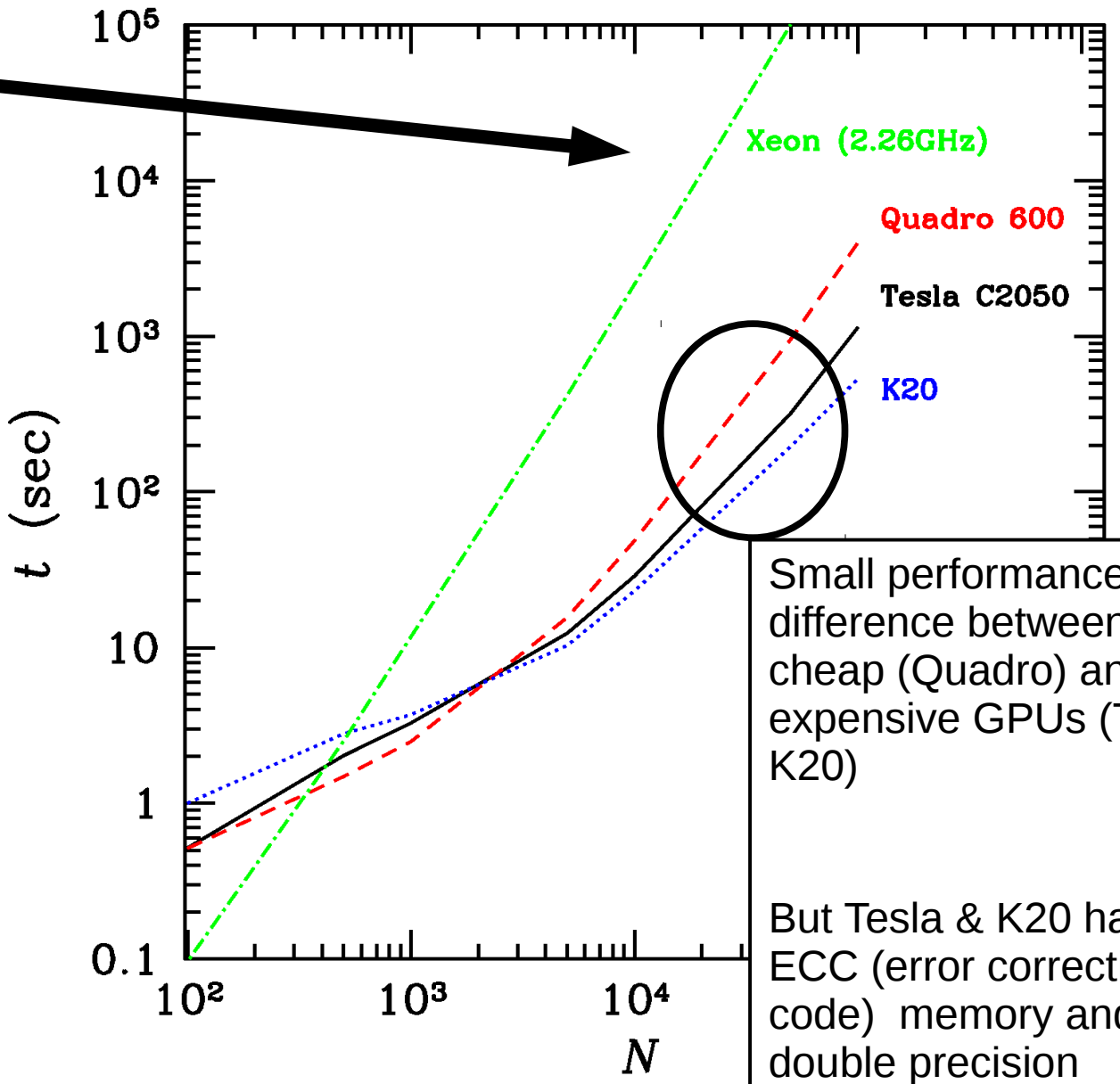
This implementation of Hermite with Sapporo allows to reach the performance I showed before:

Single Xeon processor

VS Quadro GPU (typical graphics card of desktops)

VS Tesla GPU GPU for computing: more expensive (~2k EUR) but can fit in your workstation

VS 2 Kepler 20 on the same node mounted on EURORA @ CINECA



Small performance difference between cheap (Quadro) and expensive GPUs (Tesla, K20)

But Tesla & K20 have ECC (error correcting code) memory and double precision

YOU CAN RUN YOUR OWN TESTS @ HOME!

4.2 GRAPHICS PROCESSING UNITS (GPUs)

FACILITIES with GPUs @ CINECA:

IBM PLX:

six-cores Intel Westmere 2.40 GHz per node (548 processors, 3288 cores in total)

2 NVIDIA Tesla M2070 per node (for 264 nodes) + 2 NVIDIA Tesla M2070Q per node (for 10 nodes) for a total of 548 GPUs

EURORA:

64 nodes

2 Xeon E5-2687W 3.10 GHz per node

2 NVIDIA K20 per node (64 cards now)

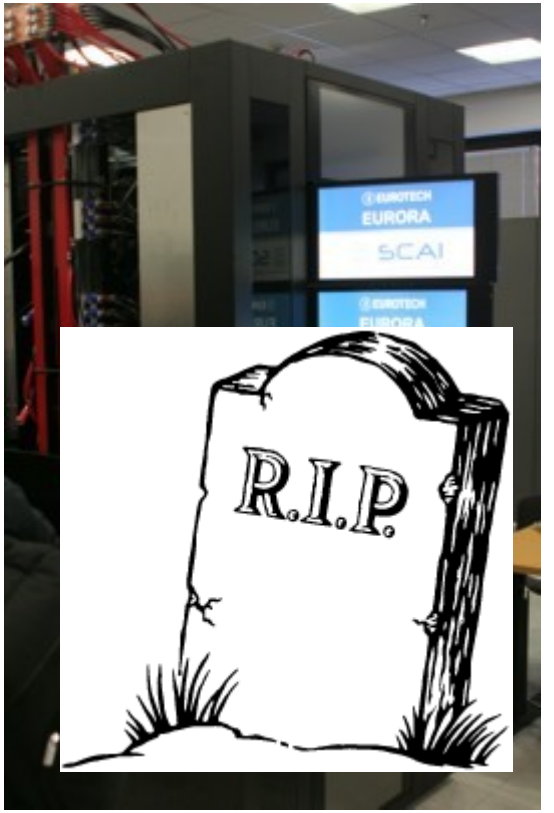
4.2 GRAPHICS PROCESSING UNITS (GPUs)

FACILITIES with GPUs @ CINECA:

IBM PLX:

six-cores Intel Westmere 2.40 GHz per node (548 processors, 3288 cores in total)

2 NVIDIA Tesla M2070 per node (for 264 nodes) + 2 NVIDIA Tesla M2070Q per node (for 10 nodes) for a total of 548 GPUs



EURORA:

64 nodes

2 Xeon E5-2687W 3.10 GHz per node

2 NVIDIA K20 per node (64 cards now)

4.2 GRAPHICS PROCESSING UNITS (GPUs)

FACILITIES with GPUs @ CINECA:

IBM GALILEO:

Model: IBM NeXtScale

Architecture: Linux Infiniband Cluster

Nodes: 516

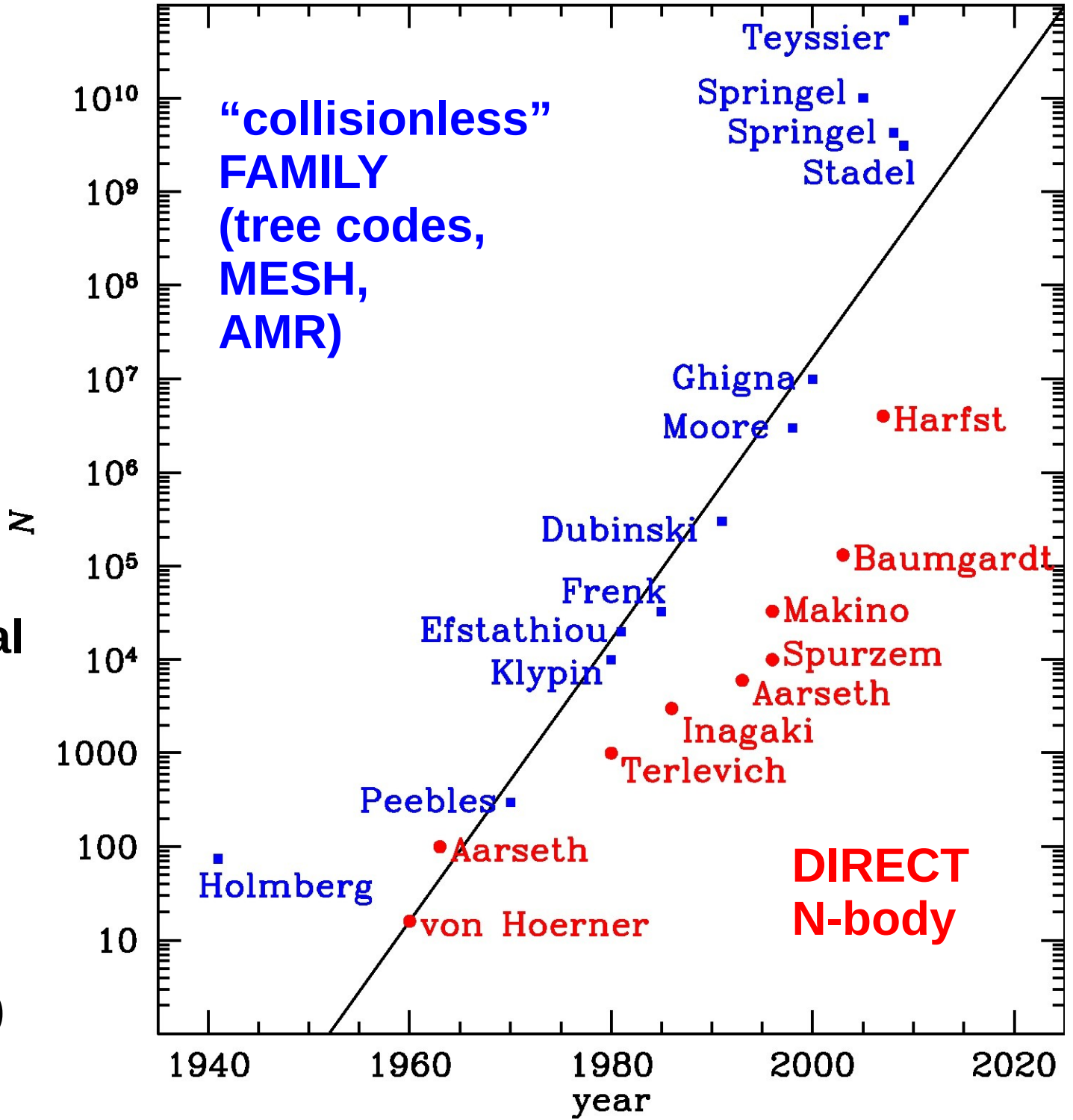
Processors: 2 8-cores Intel Haswell 2.40 GHz per node

Cores: 16 cores/node, 8256 cores in total

Accelerators: 2 Intel Phi 7120p per node on 384 nodes (768 in total); 2 NVIDIA K80 per node on 40 nodes (80 in total, 20 available for scientific research)

4.2 GRAPHICS PROCESSING UNITS (GPUs)

Moore's Law for advances in computational Astrophysics



(from Dehnen & Read 2011, arXiv:1105.1082)

5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

Done with at least 2 algorithms:

- **copy algorithm: all processors have the entire list of particles**
- **ring algorithm: particles are split between processors**

Definition: p = number of processors, n = number of particles, m = number of active particle (sinks of gravity)

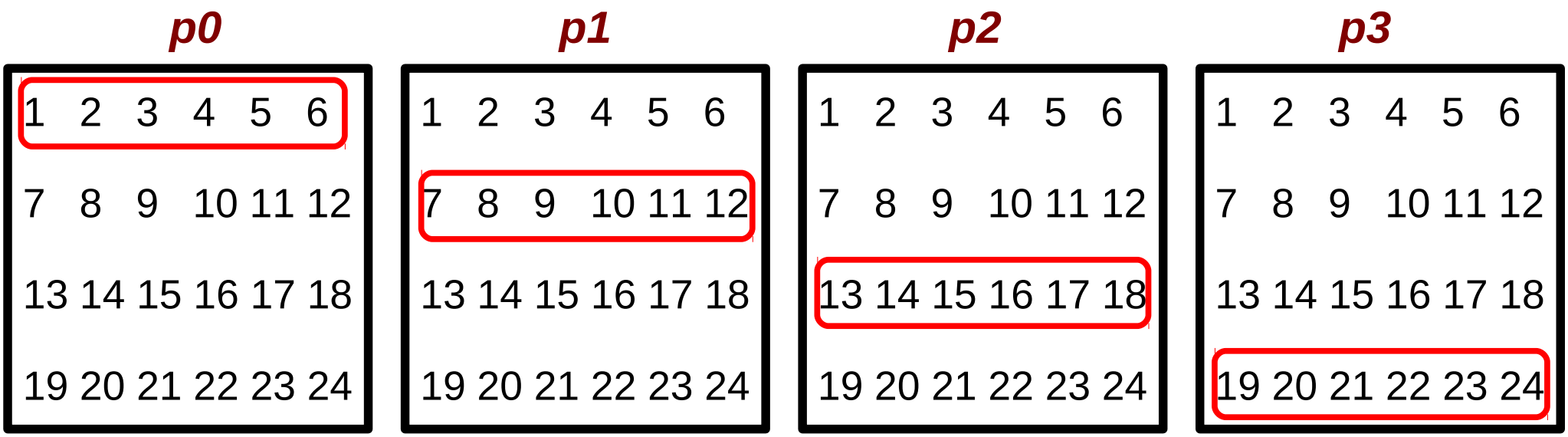
Time complexity:

- **$O(n\ p)$ for communication**
- **$O(n^2/p)$ for calculation [or rather $O(nm/p)$]**

5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

COPY ALGORITHM or REPLICATED DATA ALGORITHM:
all *p* have the entire list of particles (id., pos. & vel.)

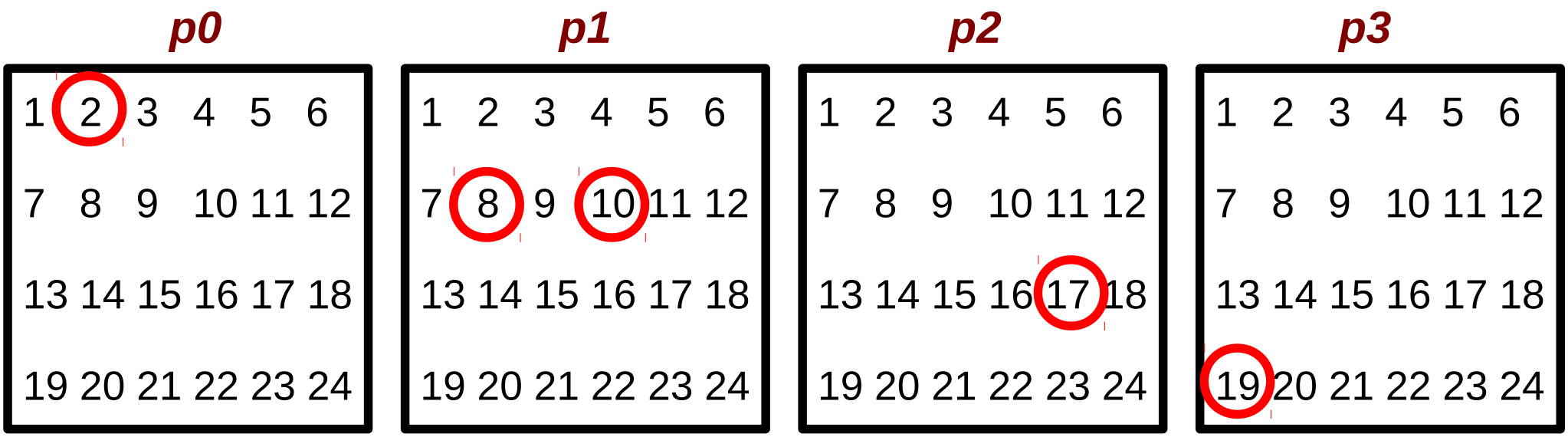
Step 1: each *p* receives a list of all the *n* particles (but will calculate the Δt of a subsample *q* of particles)
e.g. *p* = 4, *n* = 24 → *q* = 6



5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

COPY ALGORITHM or REPLICATED DATA ALGORITHM:
all *p* have the entire list of particles (id., pos. & vel.)

Step 2: Δt is calculated for the q particles → the particles with shorter Δt are ACTIVE and forces must be updated

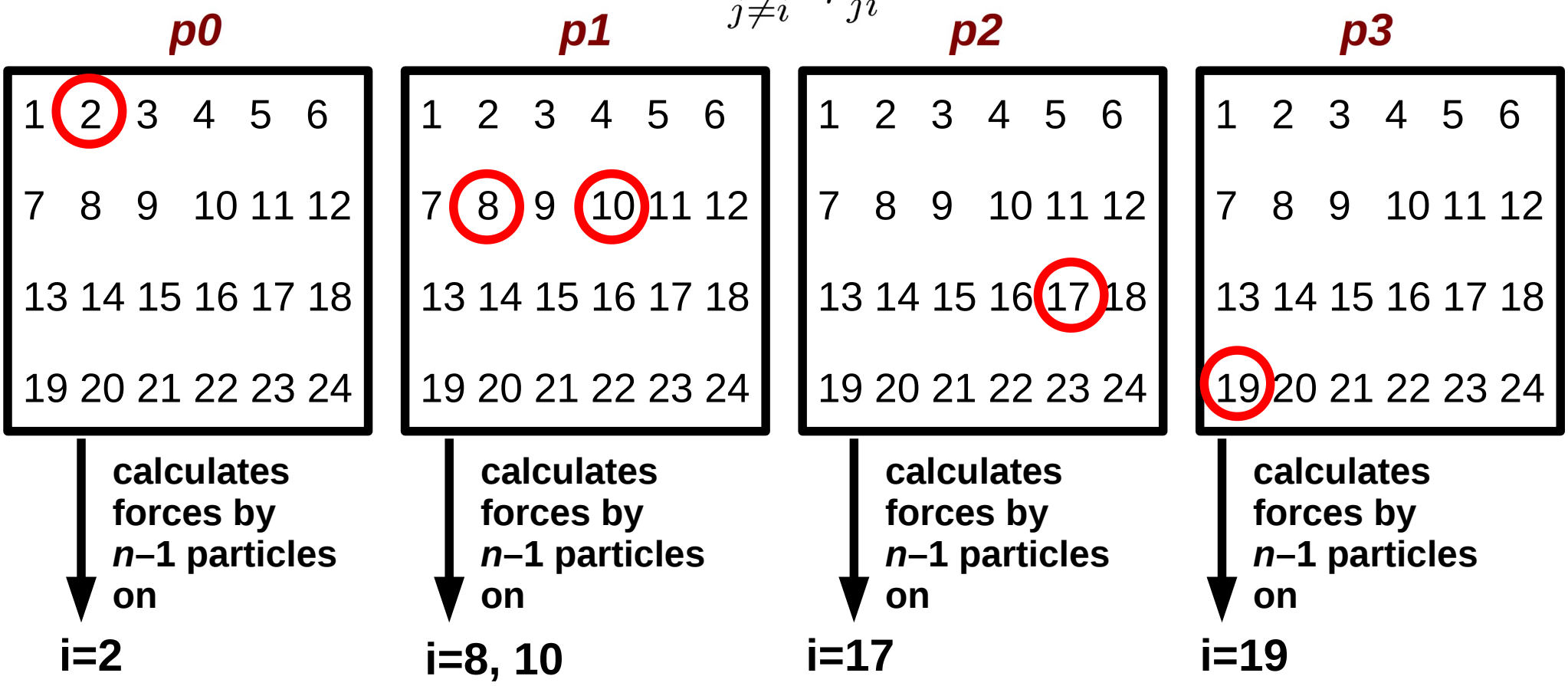


5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

COPY ALGORITHM or REPLICATED DATA ALGORITHM:
all *p* have the entire list of particles (id., pos. & vel.)

Step 3: each *p* calculates forces on the active particles in its list exerted by all the other particles

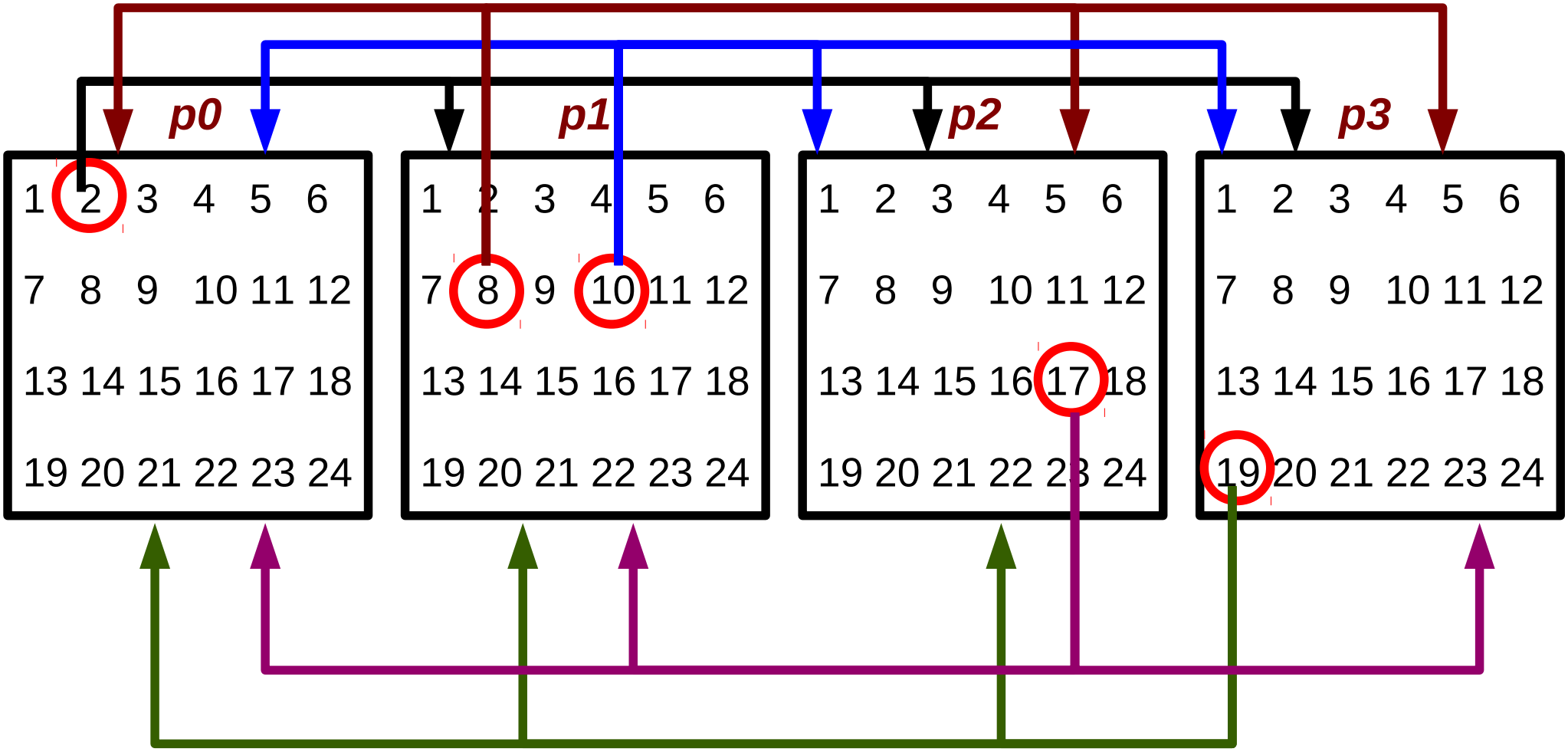
$$\vec{a}_i = G \sum_{j \neq i} \frac{M_j}{r_{ji}^3} \vec{r_{ij}}$$



5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

COPY ALGORITHM or REPLICATED DATA ALGORITHM:
all *p* have the entire list of particles (id., pos. & vel.)

Step 4: the updated forces/positions/velocities for the active particles are broadcasted to all *p*

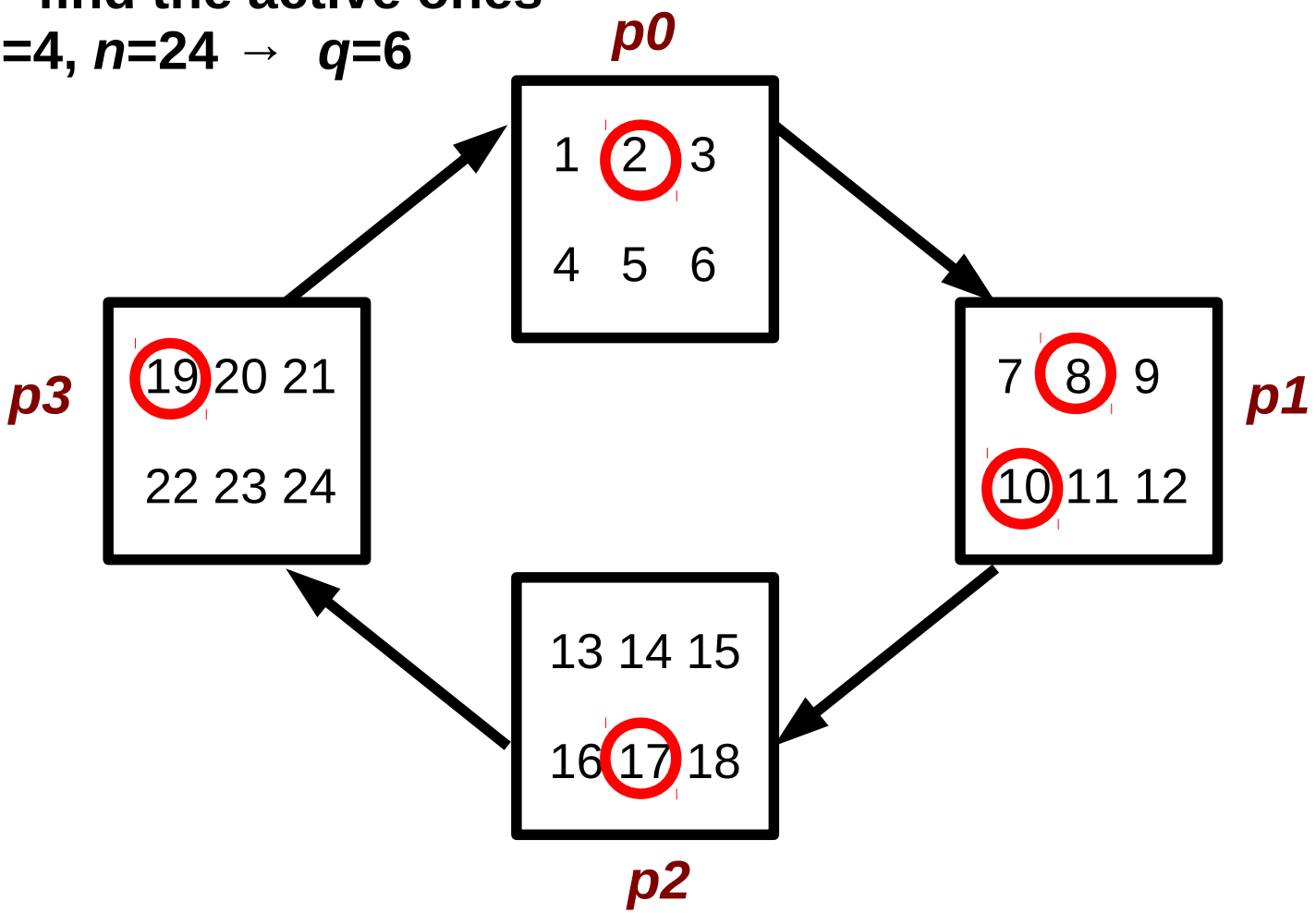


5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

RING ALGORITHM or SYSTOLIC ALGORITHM:
Each *p* has only a partial list of particles (*q* particles)
The processors *p* are connected in a ring topology

Step 0: each *p* receives a list of *q* particles and calculates Δt to find the active ones

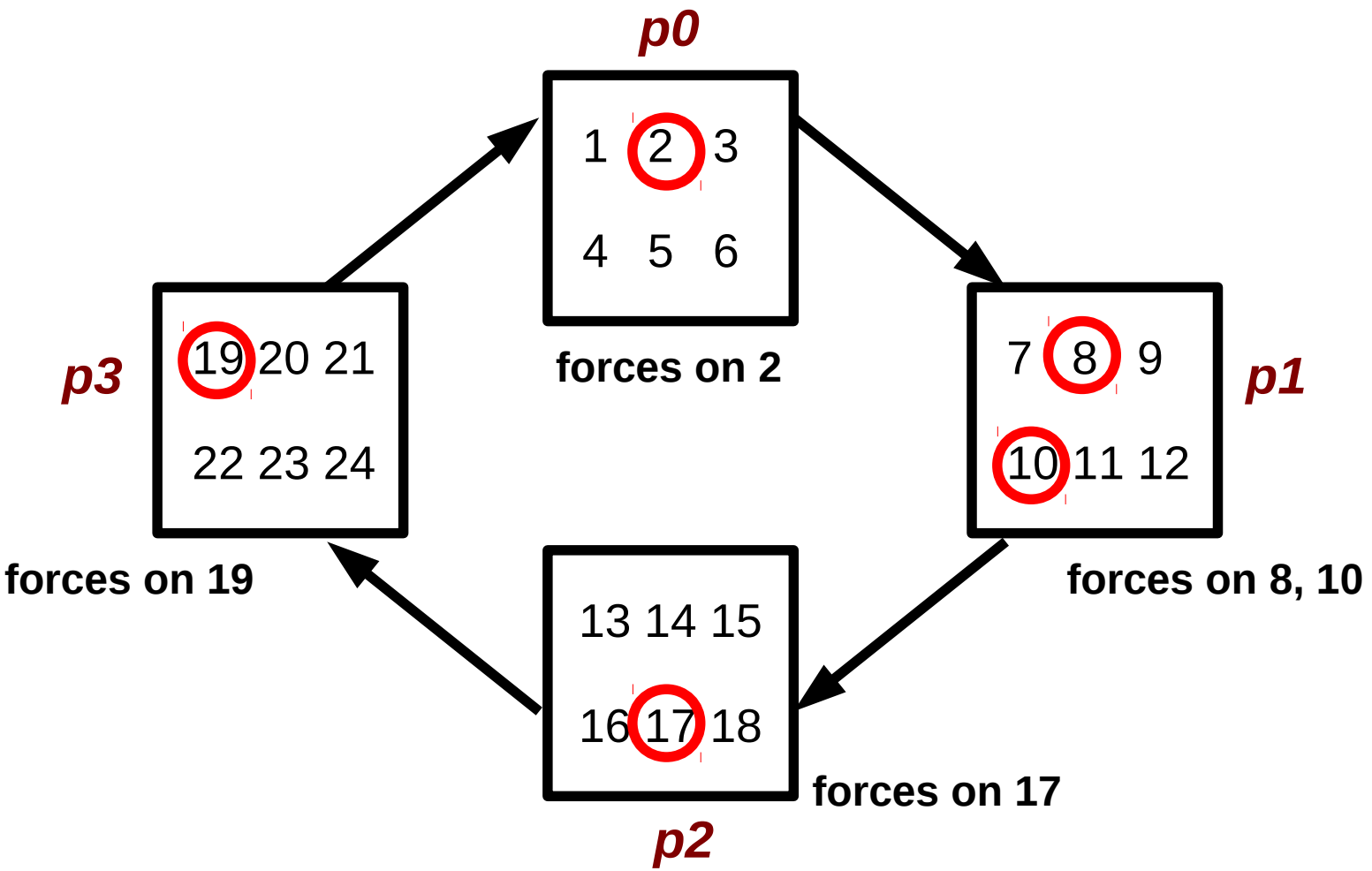
e.g. *p*=4, *n*=24 → *q*=6



5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

RING ALGORITHM or SYSTOLIC ALGORITHM:
Each *p* has only a partial list of particles (*q* particles)
The processors *p* are connected in a ring topology

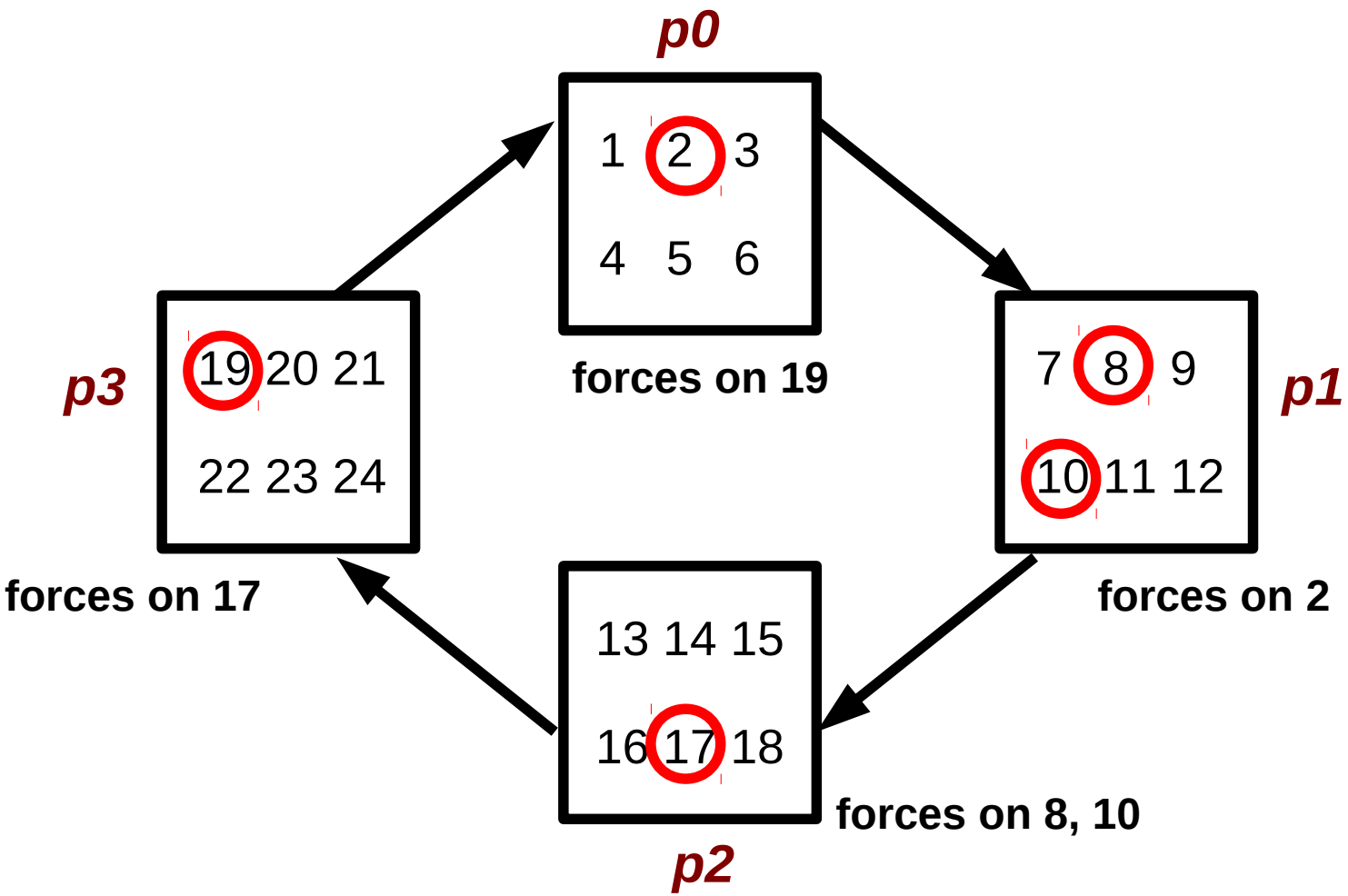
Step 1: each *p* calculates forces on ITS active particles



5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

RING ALGORITHM or SYSTOLIC ALGORITHM:
Each *p* has only a partial list of particles (*q* particles)
The processors *p* are connected in a ring topology

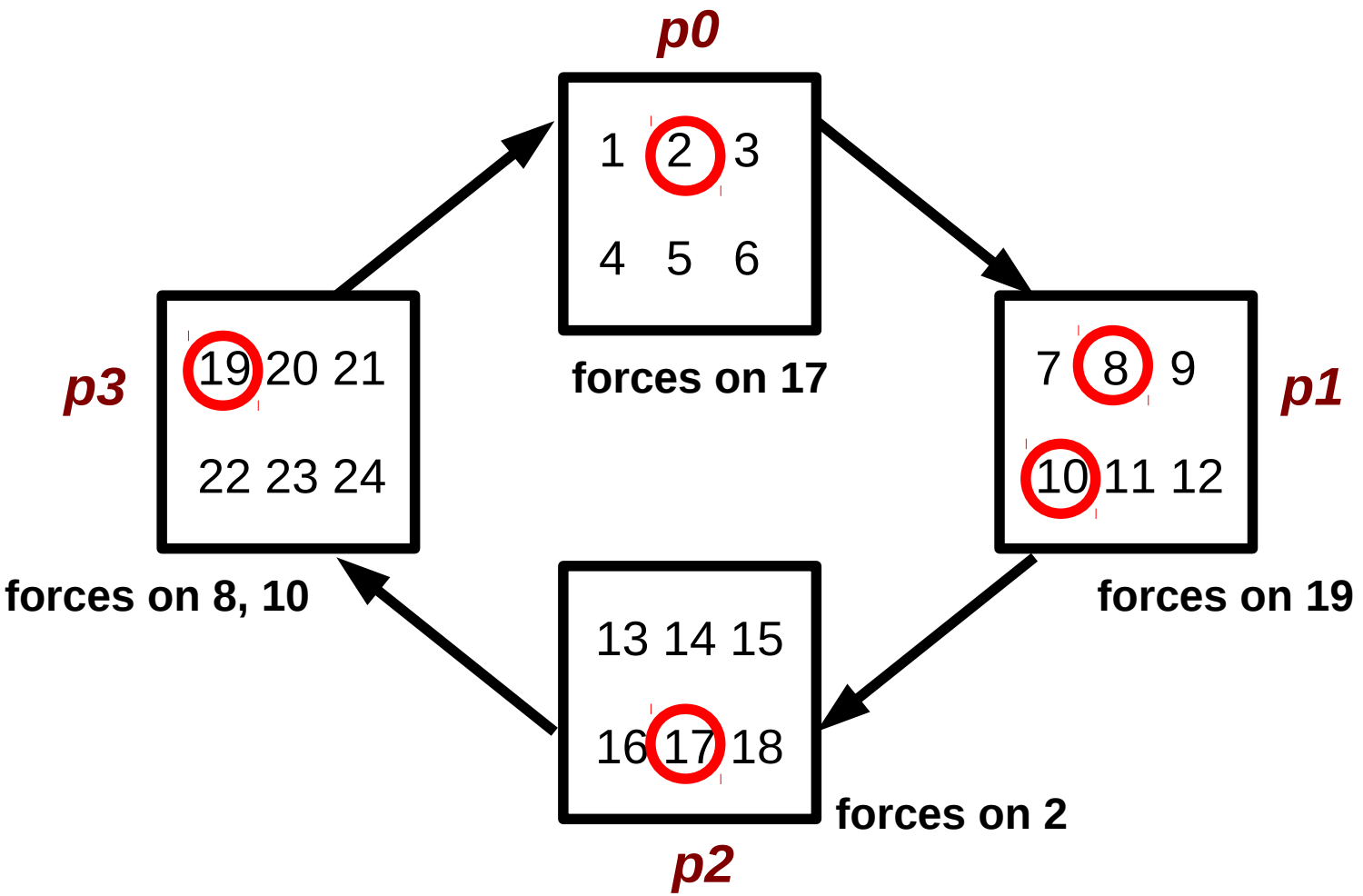
Step 2: each *p* calculates forces on next *p* (clockwise)



5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

RING ALGORITHM or SYSTOLIC ALGORITHM:
Each *p* has only a partial list of particles (*q* particles)
The processors *p* are connected in a ring topology

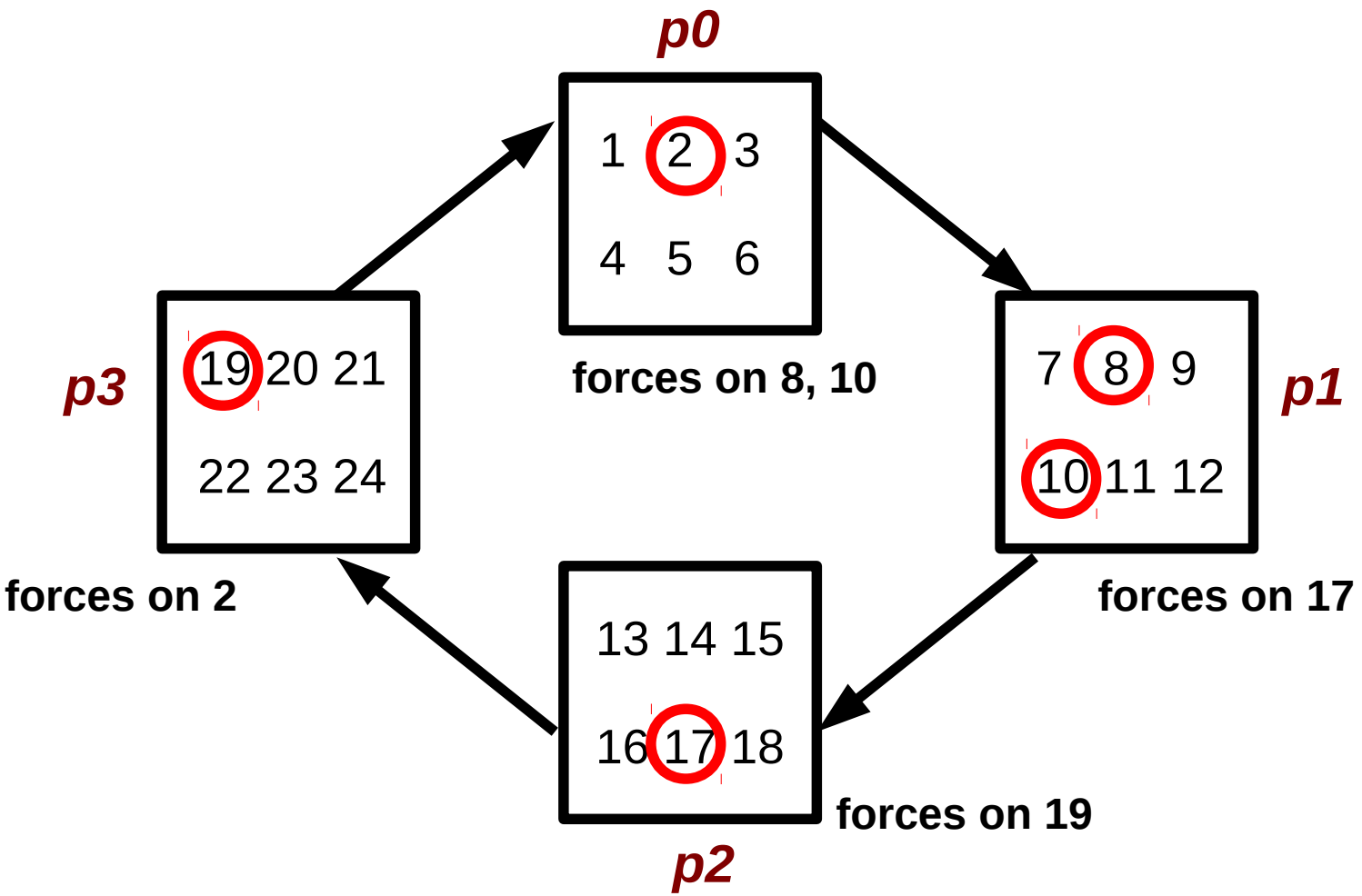
Step 3: each *p* calculates forces on bis-next *p* (clockwise)



5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

RING ALGORITHM or SYSTOLIC ALGORITHM:
Each *p* has only a partial list of particles (*q* particles)
The processors *p* are connected in a ring topology

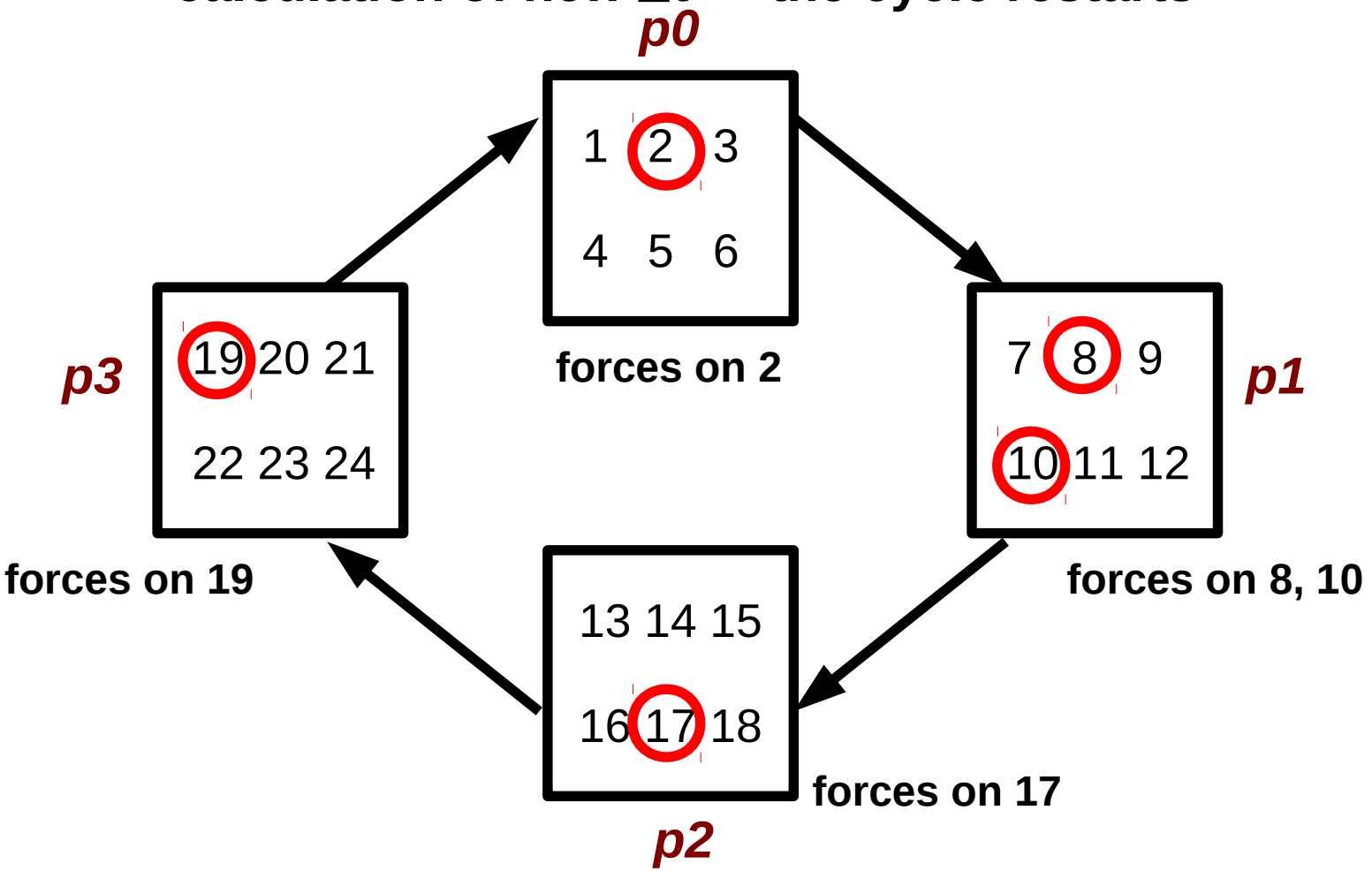
Step 4: each *p* calculates forces on bis-next *p* (clockwise)



5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

RING ALGORITHM or SYSTOLIC ALGORITHM:
Each *p* has only a partial list of particles (*q* particles)
The processors *p* are connected in a ring topology

Step 4+1: communication of new positions/velocities and calculation of new Δt → the cycle restarts



5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

RING vs COPY ALGORITHM?

Copy a. performs better if COMMUNICATION is SLOW and # of particles small (<1e5)

Ring a. performs better if COMMUNICATION is FAST and # of particles large

COMPARISON WITH Sapporo:

Parallelization on Sapporo is different:

-no copy because each *p* knows only *n/p* particles

-no systolic because the gravity sink particles are known to all multiprocessors

5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

PROBLEMS of MPI version:

difficult to treat BINARY SYSTEMS →
Binary/multiple systems continuously form/destroy during the simulation
New binary systems must be in the same processor, because of regularization → slow algorithms to change the distribution of particles between processors (there is no real tree)
→ less efficient than GPUs

The SYSTOLIC ALGORITHM DOES NOT WORK, BECAUSE A LIST OF ALL PARTICLES IN THE ENTIRE SYSTEM MUST BE KNOWN BY ALL PROCESSORS,
OTHERWISE LIST OF PERTURBERS OF BINARIES REMAINS INCOMPLETE!!!!
(Portegies Zwart et al. 2008 for this caveat)

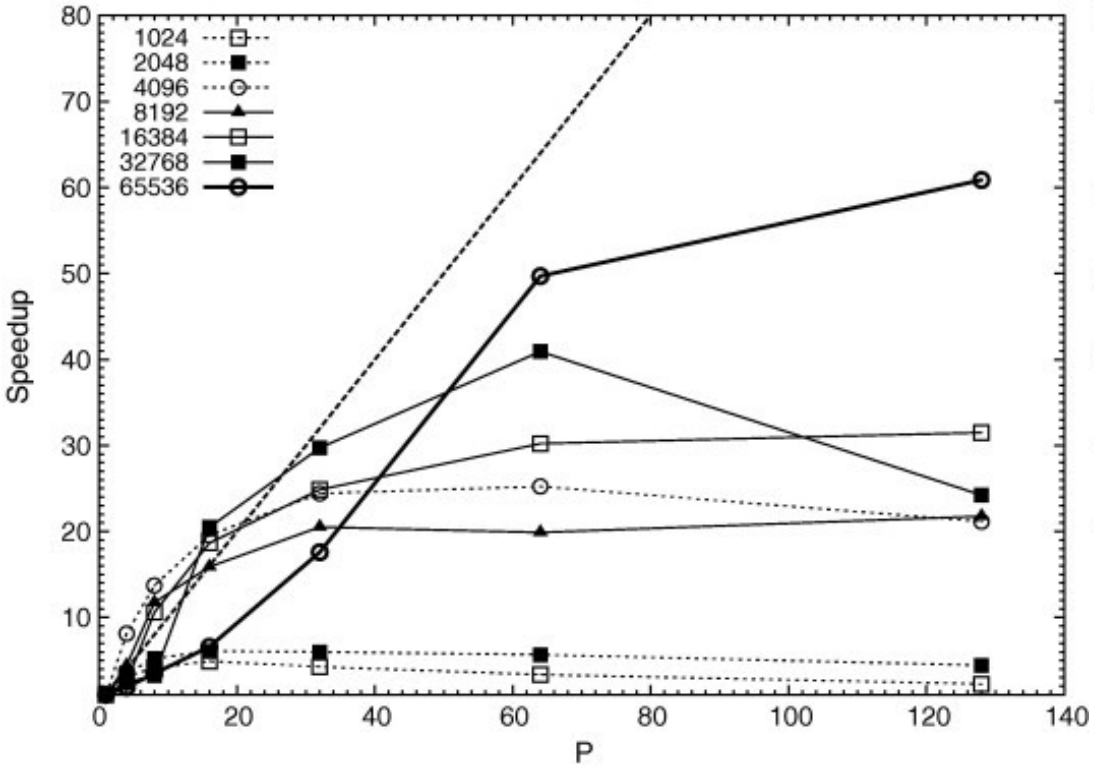
With GPUs the list of perturbers is in the device memory!
(still bottleneck but not so serious)

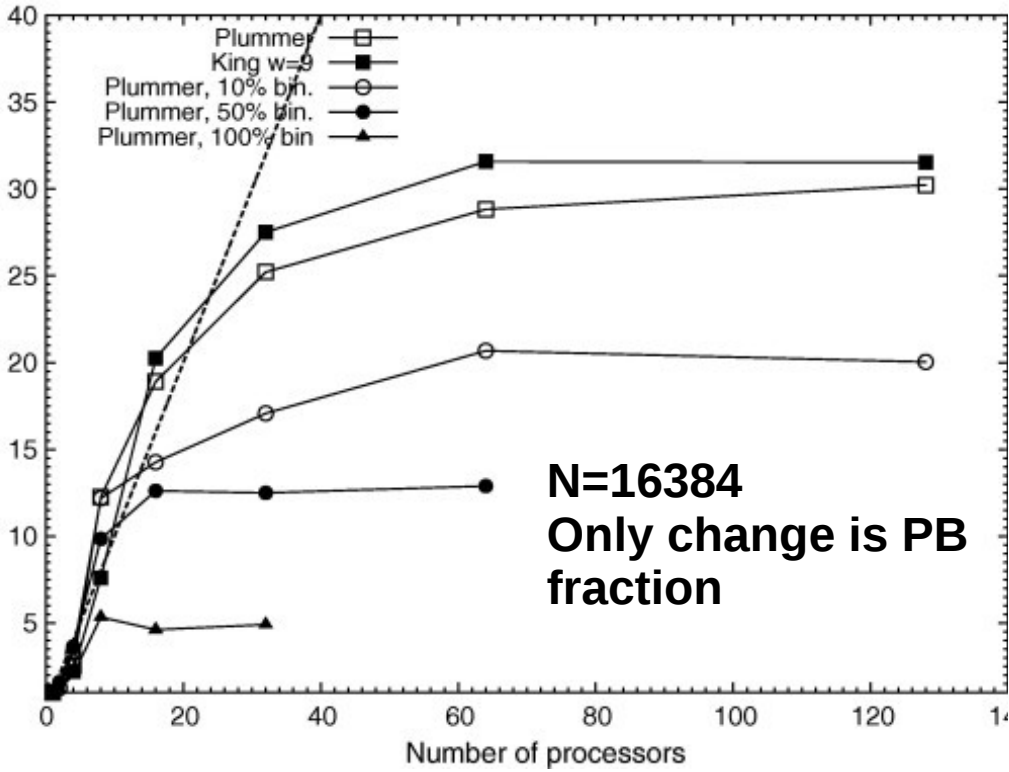
5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

PROBLEMS of MPI version:

Speed up without primordial binaries (reasonable)

Speed up WITH primordial binaries (awful)





From Portegies Zwart et al. 2008

6. STELLAR EVOLUTION

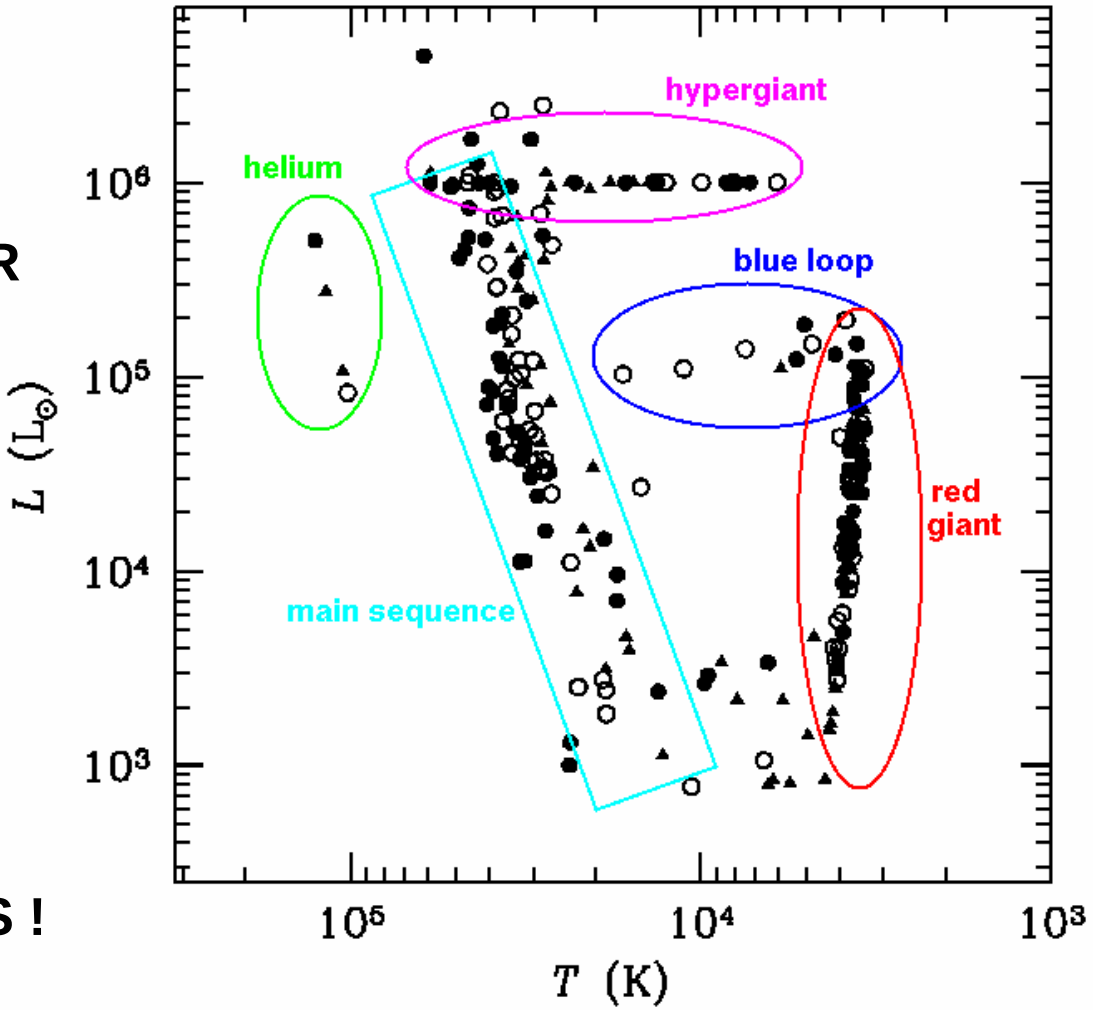
EACH PARTICLE IS A SINGLE STAR!

In simulations of galaxies and large scale structures (see Carlo Giocoli's lecture) each particle is a 'super-star':
Mass equal to ~1000 or more stars
UNPHYSICAL RADIUS: softening,
to avoid spurious relax.

In simulations of collisional systems (star clusters) each particle is a STAR
→ mass~0.1-150 Msun
and physical radius!

→ POSSIBLE ADD RECIPES FOR LUMINOSITY, TEMPERATURE, METALLICITY and LET THEM CHANGE WITH TIME!

! RESOLVED (not sub-grid) PHYSICS !



6. STELLAR EVOLUTION

Example of stellar evolution implementation:

SEBA (Portegies Zwart & McMillan 1996)

Stars are evolved via the time dependent mass-radius relations for solar metallicities given by Eggleton et al. (1989) with corrections by Eggleton et al. (1990) and Tout et al. (1997). These equations give the radius of a star as a function of time and the star's initial mass (on the zero-age main-sequence).

In MM+ 2013 the equations were upgraded to include metallicity dependence of stellar properties (with recipes in Hurley et al. 2000) and mass loss via stellar winds (Vink et al. 2001; Belczynski et al. 2010).

In the code the following stellar types are identified and tagged as different C++ CLASSES:

* proto star (0) Non hydrogen burning stars on the Hayashi track
* planet (1) Various types, such as gas giants, etc.; also includes moons.
* brown dwarf (2) Star with mass below the hydrogen-burning limit.
* main sequence (3) Core hydrogen burning star.
* Hypergiant (4) Massive (m>25Msun) post main sequence star with enormous mass-loss rate in a stage of evolution prior to becoming a Wolf-Rayet star.
* Hertzsprung gap (5) Rapid evolution from the Terminal-age main sequence to the point when the hydrogen-depleted core exceeds the Schonberg-Chandrasekhar limit.
* sub giant (6) Hydrogen shell burning star.
* horizontal branch (7) Helium core burning star.
* supergiant (8) Double shell burning star.
* helium star (9-11) Helium core of a stripped giant, the result of mass transfer in a binary. Subdivided into carbon core (9), helium dwarf (10) and helium giant (11).
* white dwarf (12-14) Subdivided into carbon dwarf (12) , helium dwarf (13) and oxygen dwarf (13).
* Thorne-Zytkow (15) Shell burning hydrogen envelope with neutron star core.
* neutron star (16-18) Subdivided into X-ray pulsar (16), radio pulsar (17) and inert neutron (18) star (m<2Msun).
* black hole (19) Star with radius smaller than the event horizon. The result of evolution of massive (m>25Msun) star or collapsed neutron star.
* disintegrated (20) Result of Carbon detonation to Type Ia supernova.

6. STELLAR EVOLUTION

Example of stellar evolution implementation:

SEBA (Portegies Zwart & McMillan 1996)

Interface with dynamics integrator:

Difficult to solve for the evolution of dynamics and stellar evolution in a completely self-consistent way!
trajectories of stars ← block timestep scheme (~1e5 yr)
stellar and binary evolution ← updated at fixed intervals
(every 1/64 of a crossing time, typically a few thousand years).

→ feedback between st. ev. and dynamics may experience a delay of at most one timestep.

After each 1/64 of a crossing time, all stars and binaries are checked to determine if evolutionary updates are required. Single stars are updated every 1/100 of an evolution timestep or when the mass of the star has changed by more than 1% since the last update. A stellar evolution timestep is the time taken for the star to evolve from the start of one evolutionary stage to the next.

After each stellar evolution step the dynamics is notified of changes in stellar radii, but changes in mass are, for reasons of efficiency, not passed back immediately (mass changes generally entail recomputing the accelerations of all stars in the system). Instead, the ``dynamical'' masses are modified only when the mass of any star has changed by more than 1%, or if the orbital parameters, semi-major axis, eccentricity, total mass or mass ratio of any binary has changed by more than 0.1%.