

Direct summation N-body codes for astrophysical simulations: from GRAPE to GPU

Summer School

HIGH PERFORMANCE SCIENTIFIC COMPUTING

**Strategic Research Project AACSE - Algorithms and Architectures
for Computational Science and Engineering**

September 16-19, 2013

OUTLINE:

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 4th 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 $O(N \log N)$**

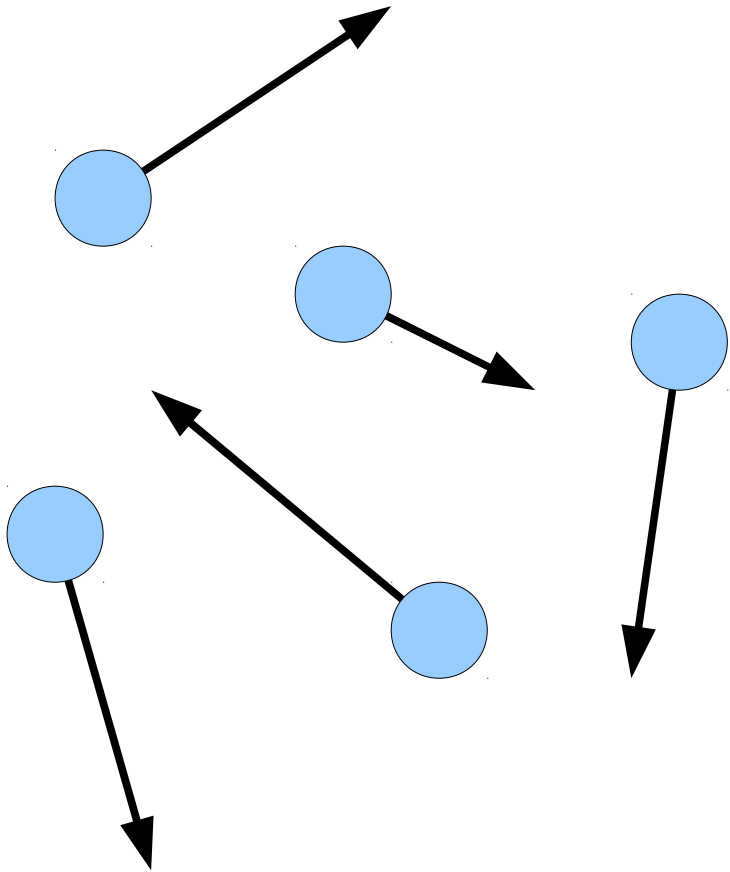
– see Carlo Giocoli's lecture

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

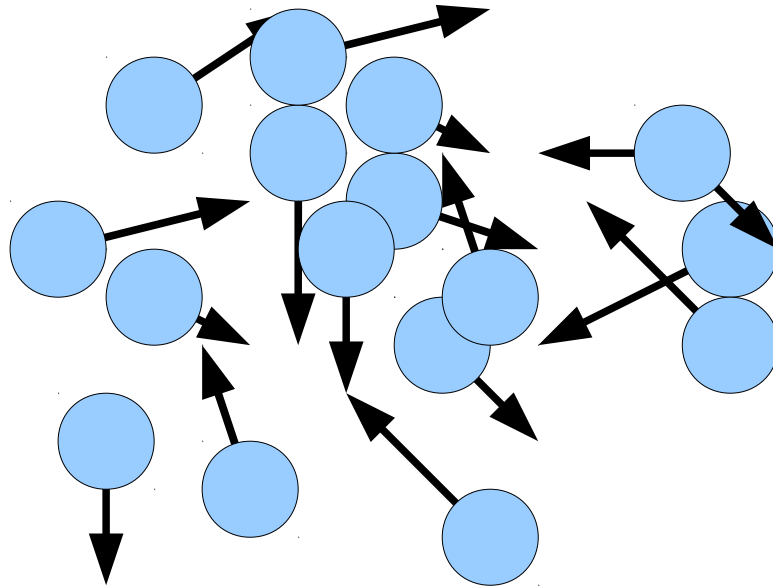
→ they can be treated as a FLUID in the phase space

e.g. **GALAXIES** are **COLLISIONLESS SYSTEMS**



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

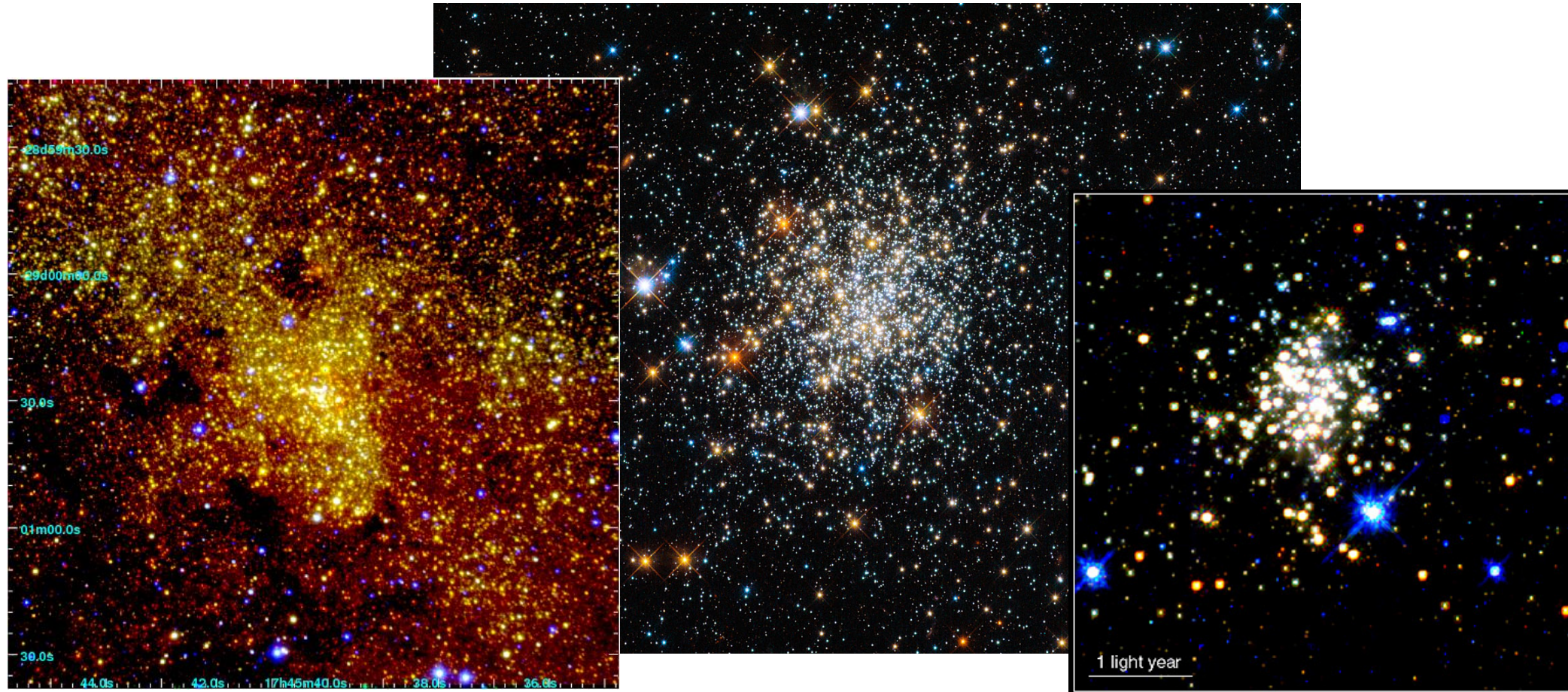
density / vel^3

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)

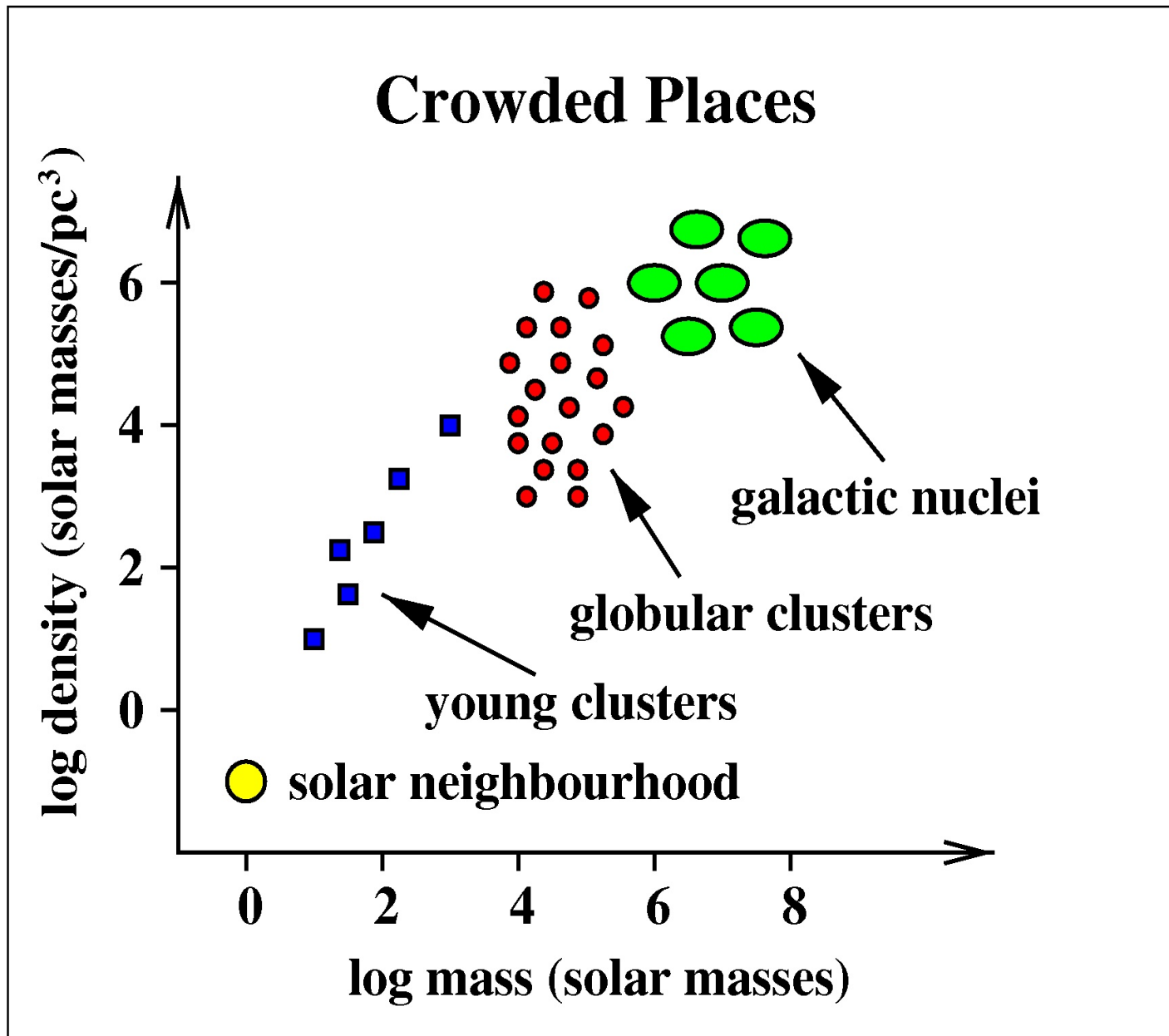
So that we need to **resolve each single star and each interaction it undergoes** → **We cannot use approximations!!!**

THE DENSEST STELLAR SYSTEMS: STAR CLUSTERS and GALACTIC NUCLEI



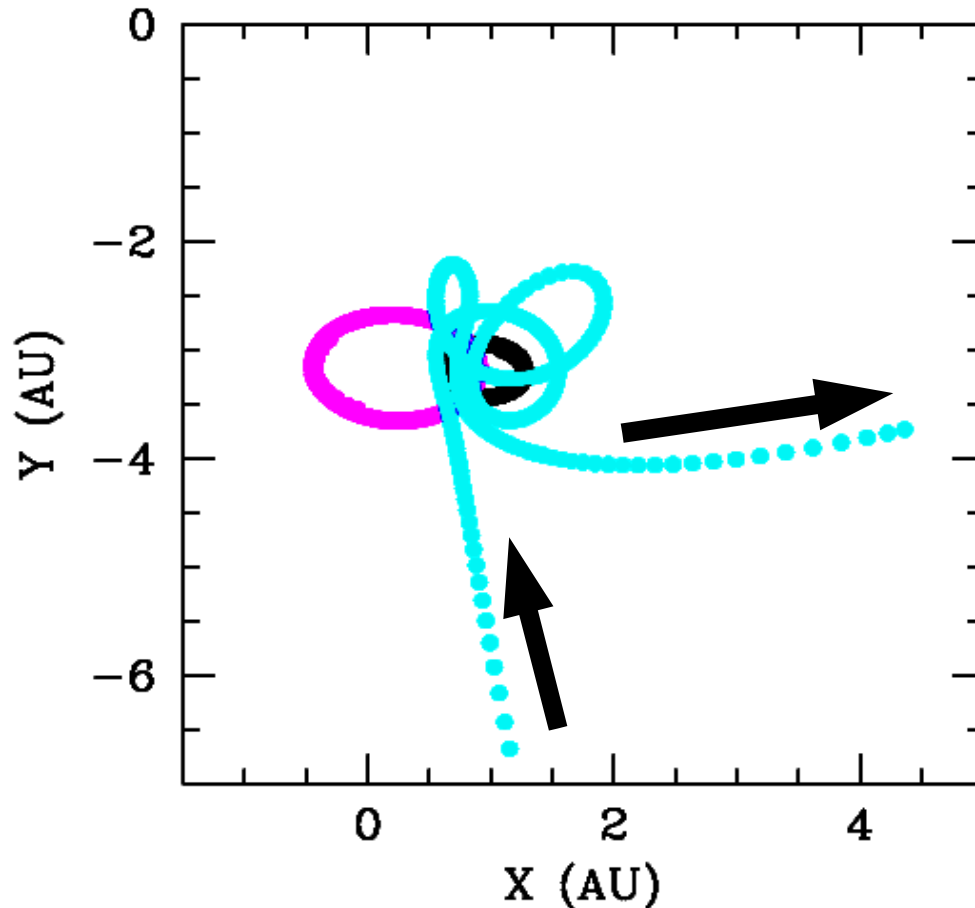
2. WHY/WHEN do we use direct N-body codes?

MAP of the DENSEST PLACES in the Universe



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

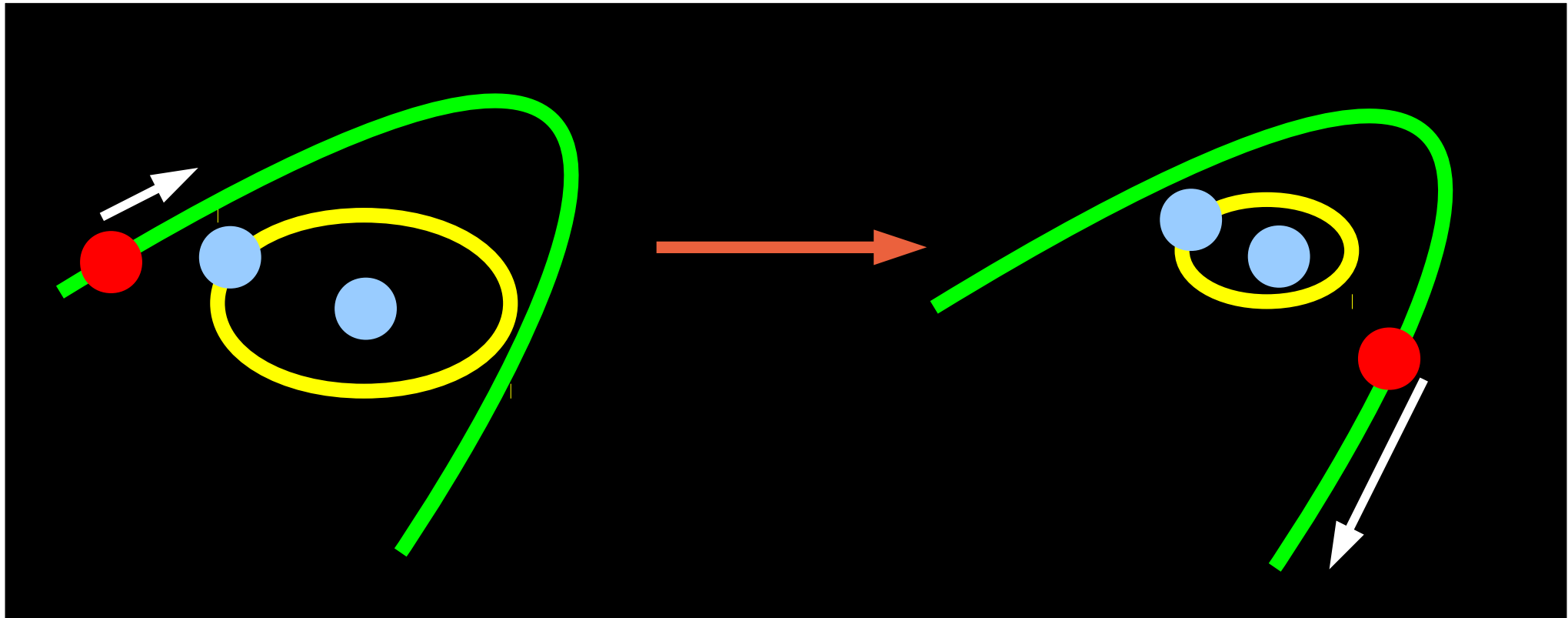


* Similar to scattering experiments in (sub)atomic physics but involving stars/binary stars and ONLY GRAVITATIONAL FORCE

* 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?

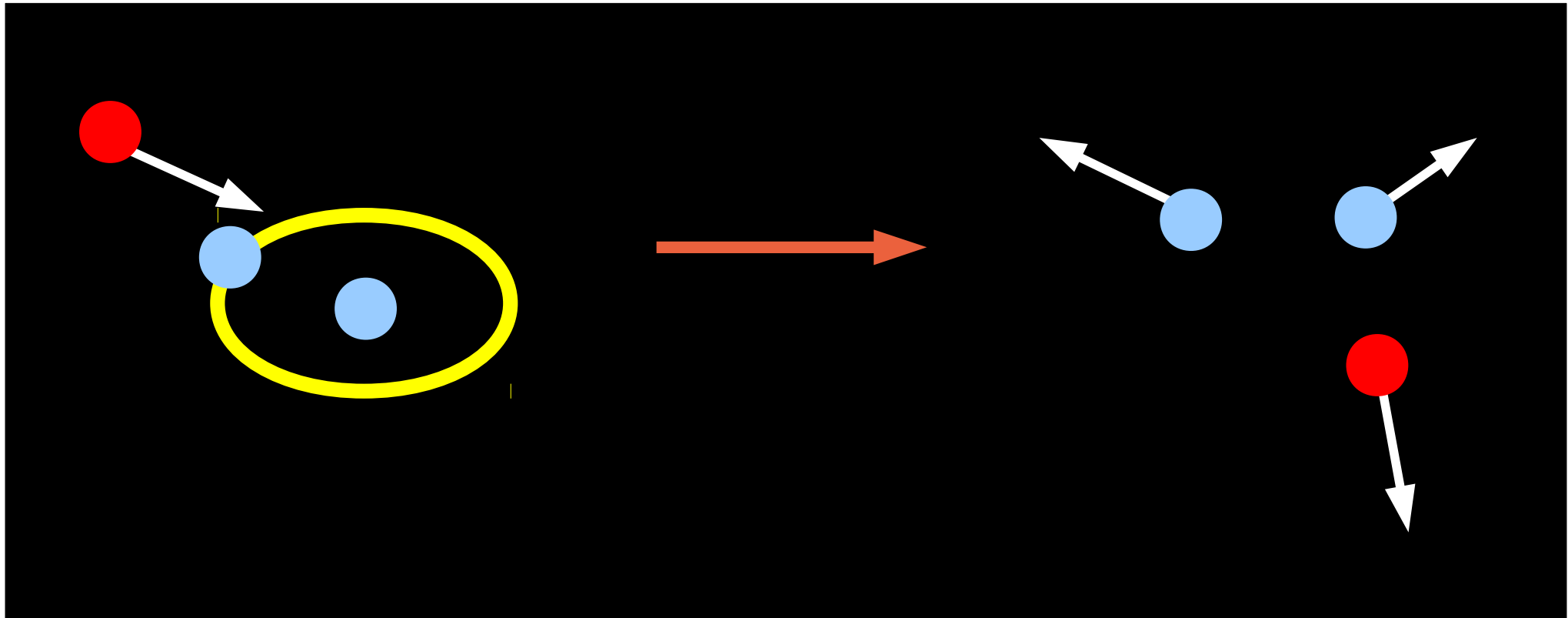
EXAMPLES of 3-BODY ENCOUNTERS



FLYBY: ORBITS CHANGE

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

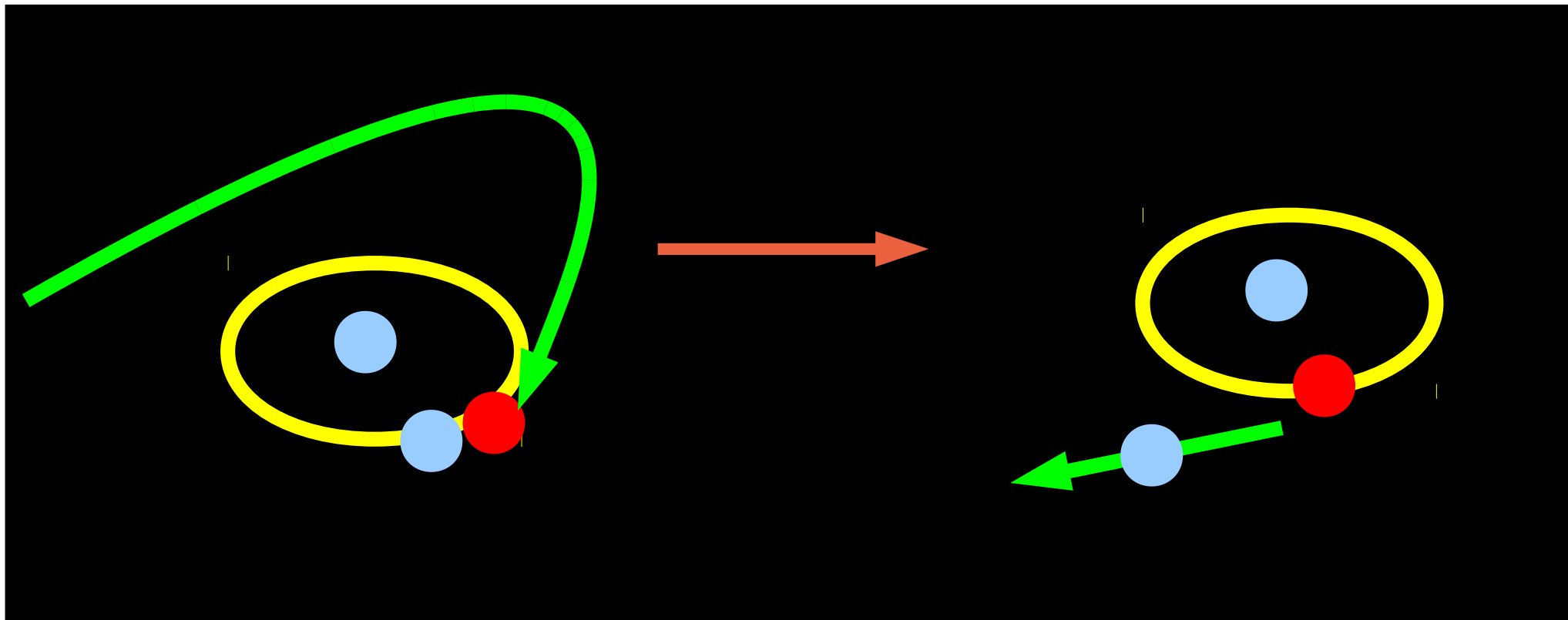
EXAMPLES of 3-BODY ENCOUNTERS



IONIZATION: binary is destroyed (analogy with atoms)

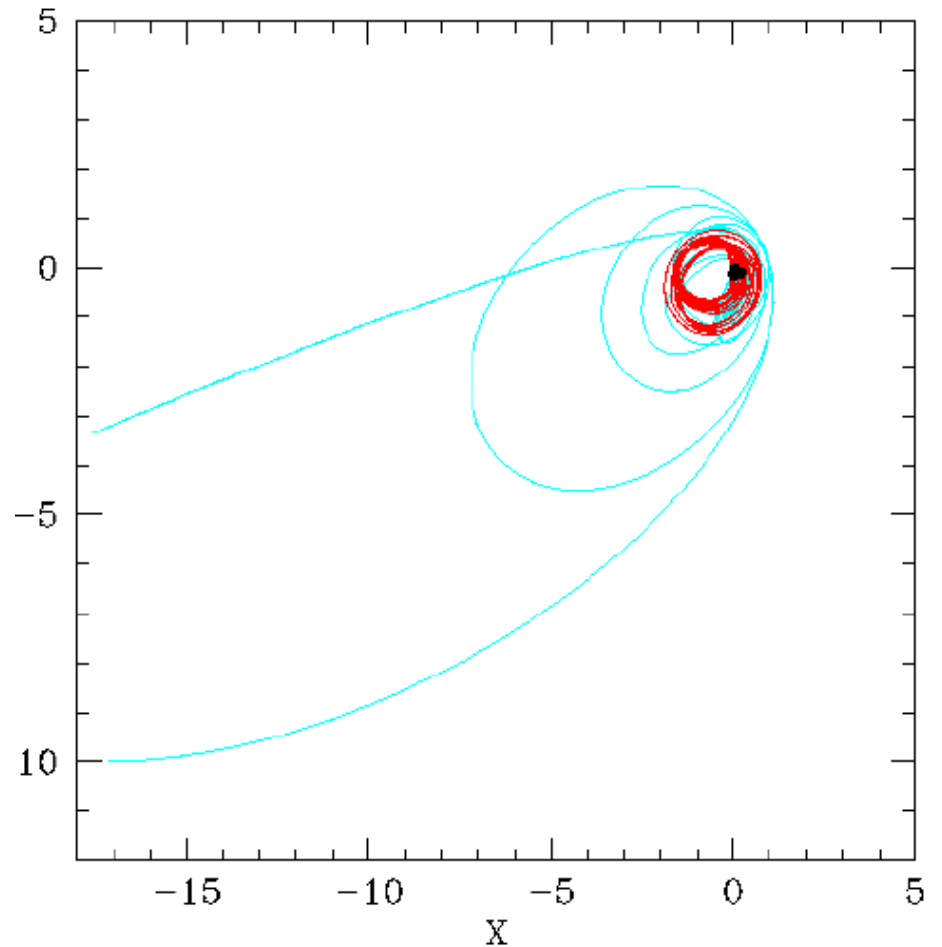
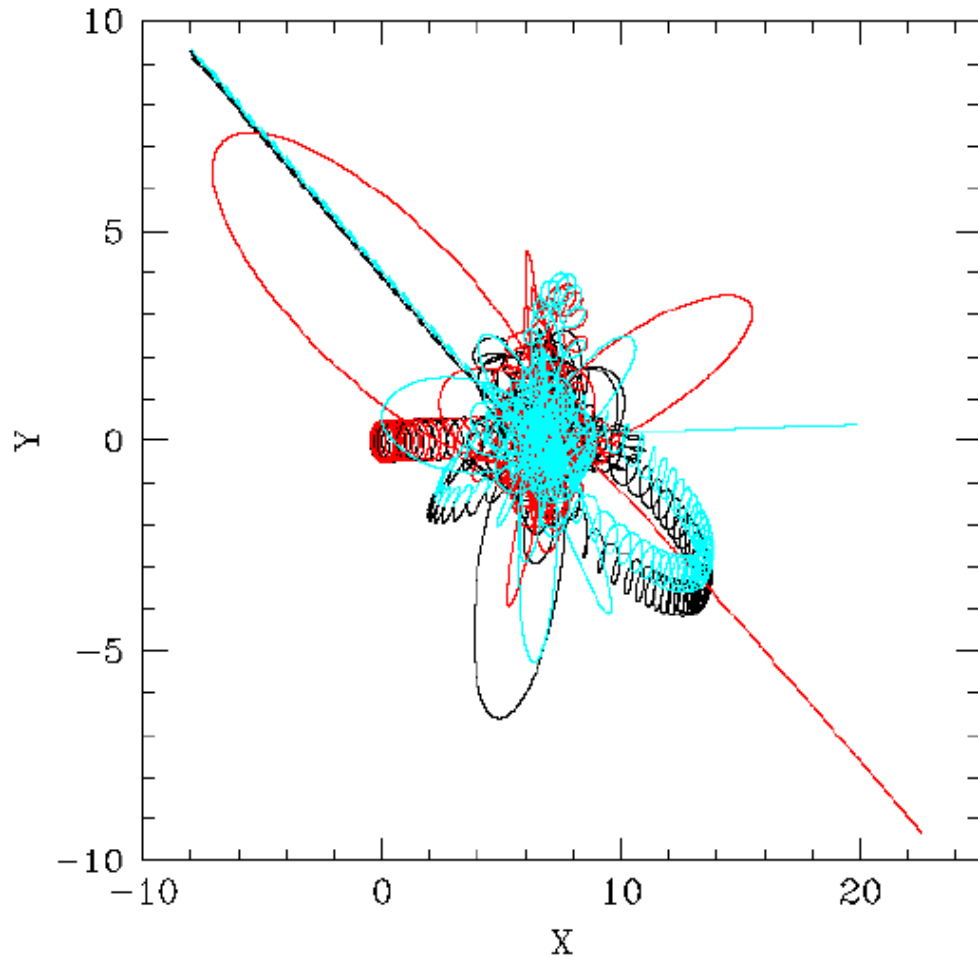
2. WHY/WHEN do we use direct N-body codes?

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)

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

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

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:

$$\left\{ \begin{array}{l} x_1 = x_0 + v_0 \Delta t + \frac{1}{2} a_0 \Delta t^2 + \frac{1}{6} \dot{j}_0 \Delta t^3 + \frac{1}{24} \ddot{j}_0 \Delta t^4 \quad (1) \\ v_1 = v_0 + a_0 \Delta t + \frac{1}{2} \dot{j}_0 \Delta t^2 + \frac{1}{6} \ddot{j}_0 \Delta t^3 + \frac{1}{24} \overset{\cdot\cdot}{j}_0 \Delta t^4 \quad (2) \\ a_1 = a_0 + \dot{j}_0 \Delta t + \frac{1}{2} \ddot{j}_0 \Delta t^2 + \frac{1}{6} \overset{\cdot\cdot}{j}_0 \Delta t^3 \quad (3) \\ \dot{j}_1 = \dot{j}_0 + \ddot{j}_0 \Delta t + \frac{1}{2} \overset{\cdot\cdot}{j}_0 \Delta t^2 \quad (4) \end{array} \right.$$

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

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

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:

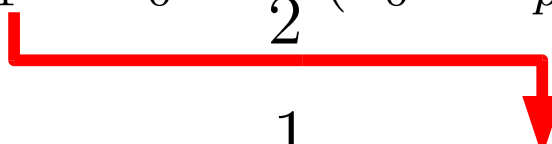
we then substitute $a_{p,1}$ and $j_{p,1}$ into equations (5) and (6):

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

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

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


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


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

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
 $\eta = 0.01 - 0.02$ is good choice

$$\Delta t_i = \eta \frac{a_i}{j_i}$$

2. system time is set as $t := t_i + \min(\Delta t_i)$
All particles with time-step = $\min(\Delta t_i)$ 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:

$$\Delta t_i = \eta \frac{a_i}{j_i}$$

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 $\Delta t_{min} = 2^{-23}$

NOTES on Hermite and time steps:

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

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

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

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

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):**

$$\Delta t_i = \sqrt{\eta \frac{|a_{i,1}| |a_{i,1}^{(2)}| + |j_{i,1}|^2}{|j_{i,1}| |a_{i,1}^{(3)}| + |a_{i,1}^{(2)}|^2}} \quad \text{where } \eta = 0.01 - 0.02 \text{ is good choice}$$

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

$$\eta_{\text{STARLAB}} = \text{sqrt}(\eta) \rightarrow \eta_{\text{STARLAB}} = 0.1 \text{ is good choice (Anders+2012)}$$

***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's 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:

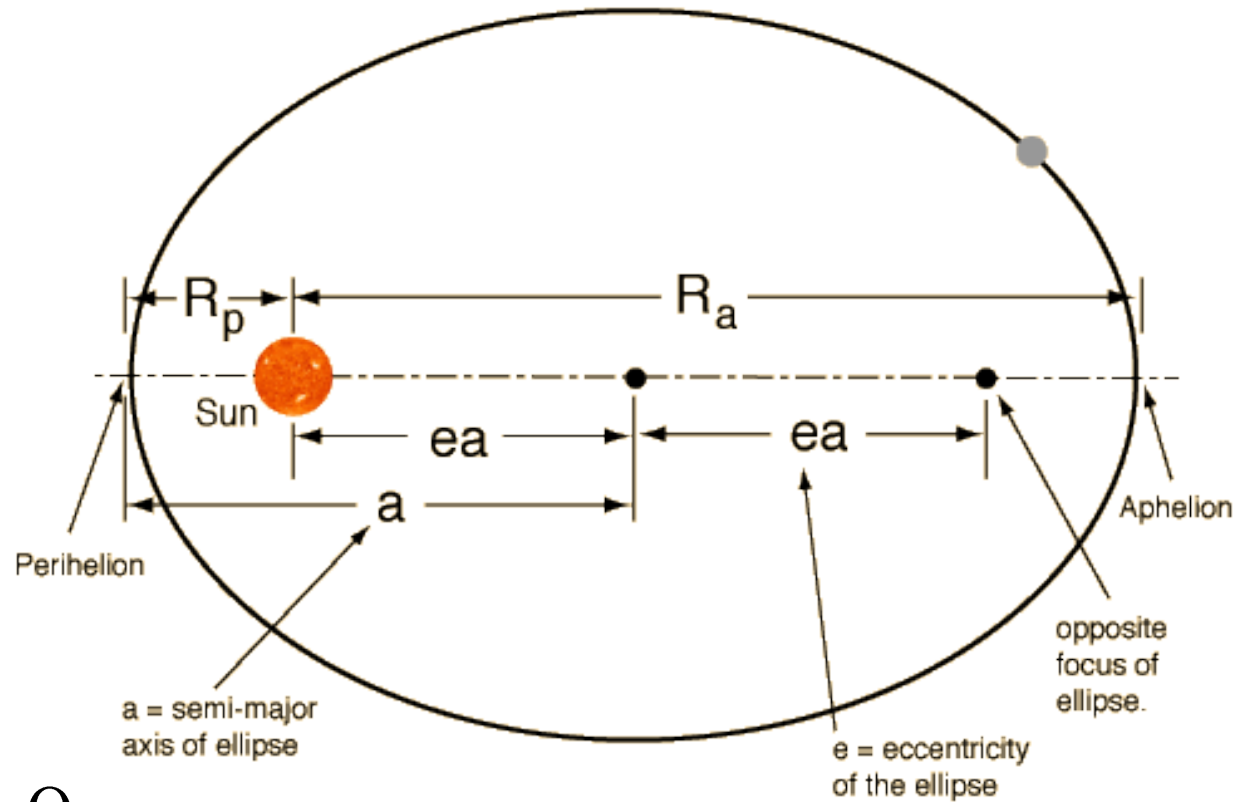
Mtot = M1+M2

Reduced mass:

$\mu = M1 M2/(M1+M2)$

equation of Kepler motion for reduced Mass:

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



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

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:

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

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):

$$dt = \frac{r}{\xi} d\tau \quad \text{WHERE} \quad \xi = \sqrt{\frac{G \mu}{a}}$$

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

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

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

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

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

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

2 becomes

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

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:

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

↓ IF $\frac{d\xi}{d\tau} = 0$

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

**EQUATION OF HARMONIC
OSCILLATOR
(NO SINGULARITY)!**

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

**CASE of UNPERTURBED BINARY:
ENERGY DOES NOT CHANGE**

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

**CASE of PERTURBED BINARY:
3-BODY ENCOUNTER**

BUT

3.3 REGULARIZATION

Regularisation for multi-body systems:

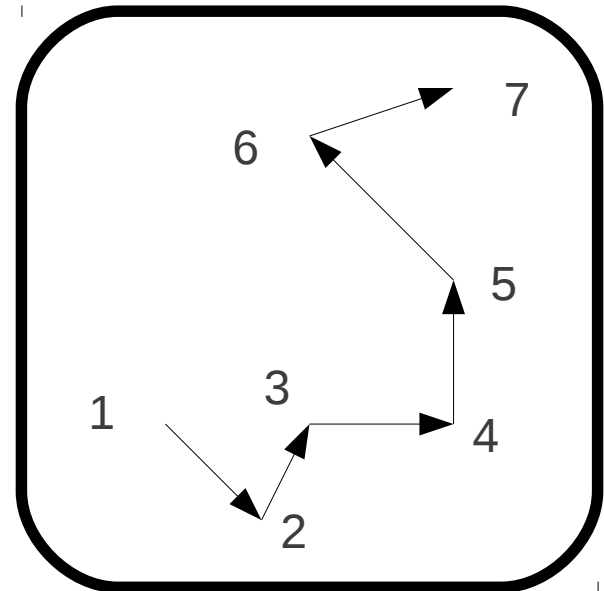
CHAIN regularisation by Aarseth

(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

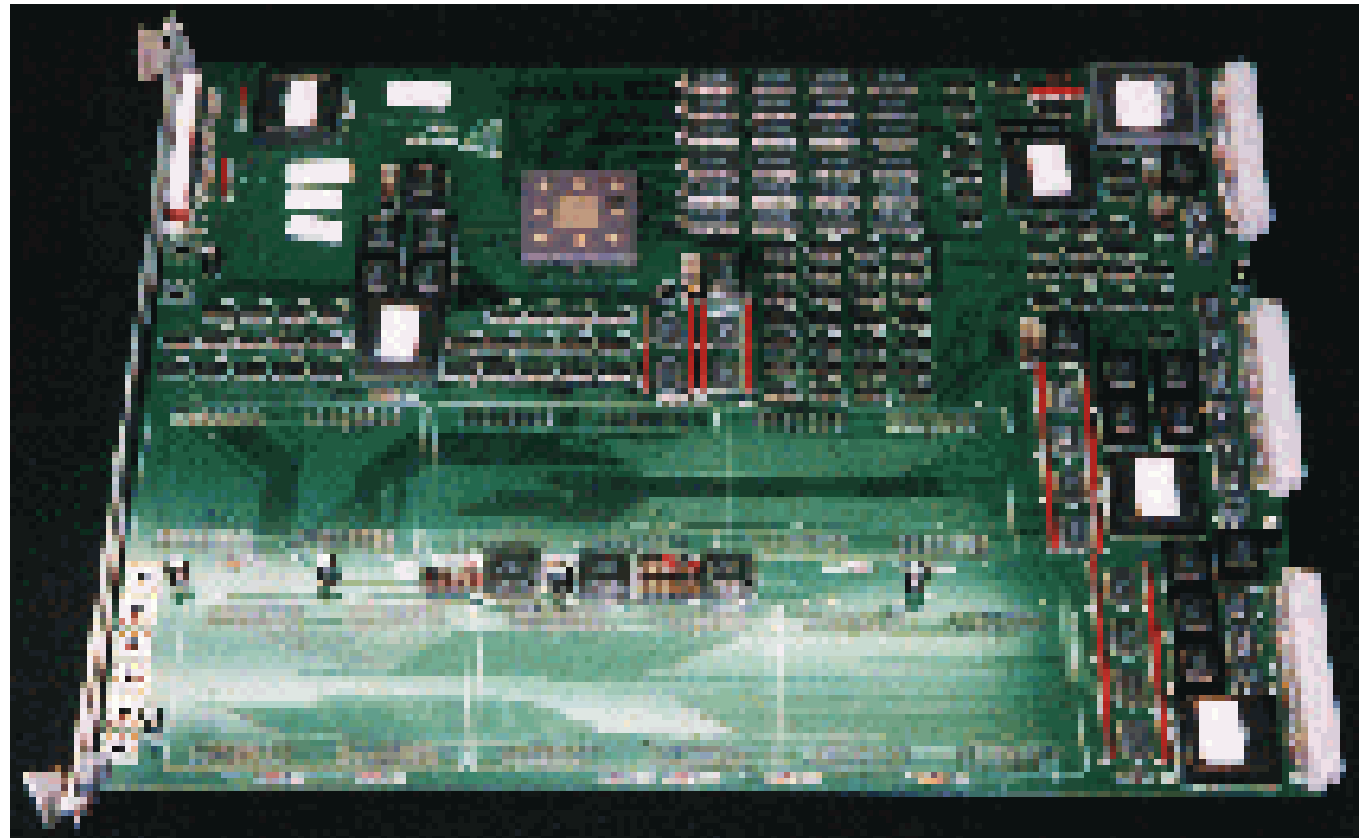
SORT of GRAVITY
ACCELERATOR

as a

GRAPHICS CARD is
a GRAPHICS
ACCELERATOR

Predictor/corrector
on PC

Acceleration and
jerk calculation on
GRAPE



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

History:

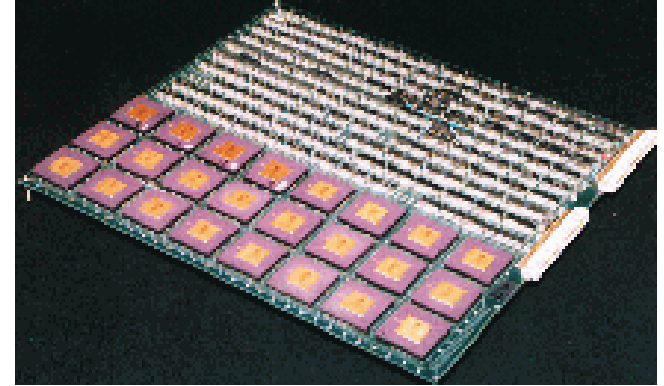
1989: GRAPE project starts at Tokyo university (Daiichiro Sugimoto and then Junichiro Makino)

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.

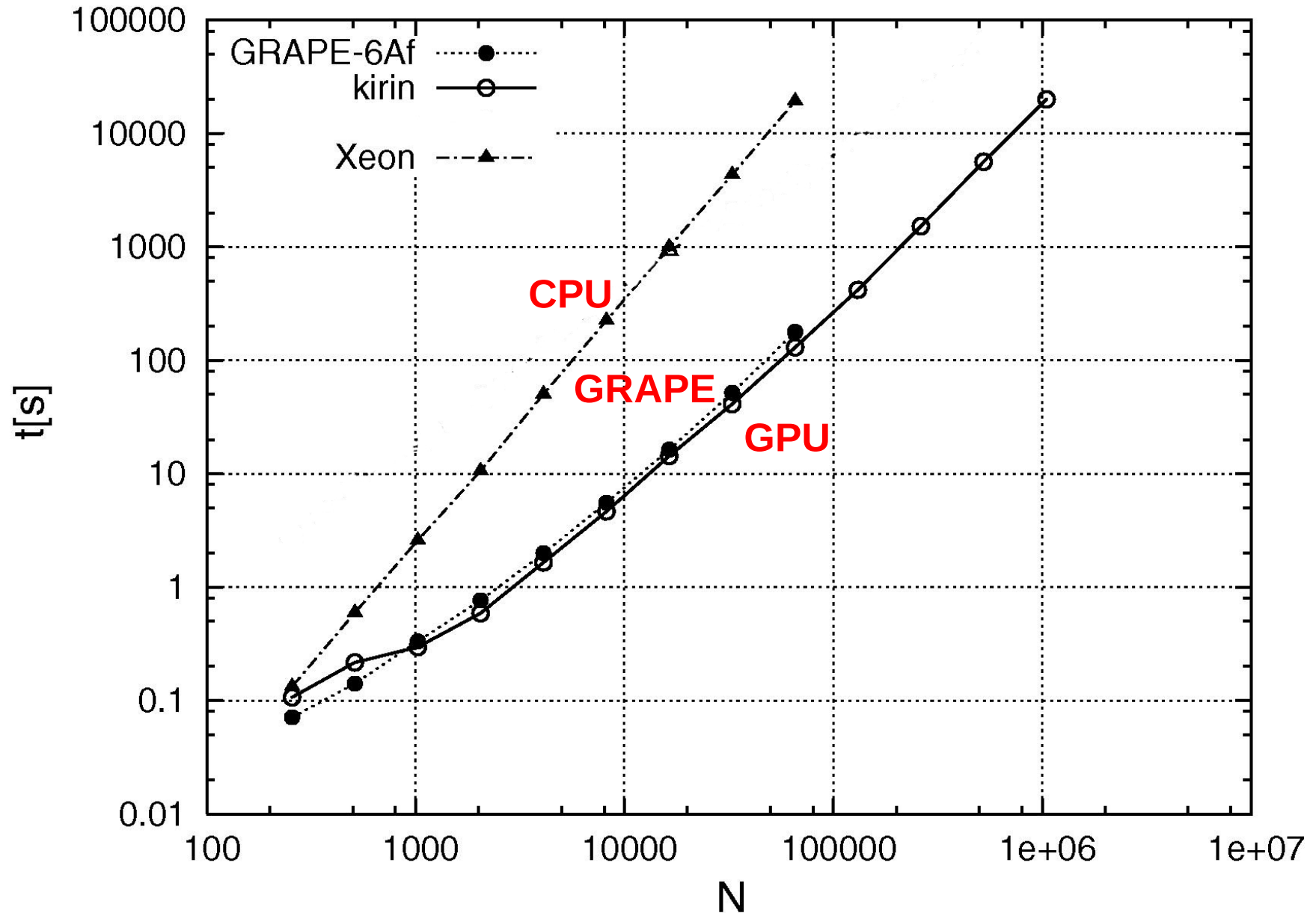
A single GRAPE-6 boards runs at 1 Tflop
A 4-cabinet (with 8 GRAPE-6 boards each)
at 32 Tflop

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

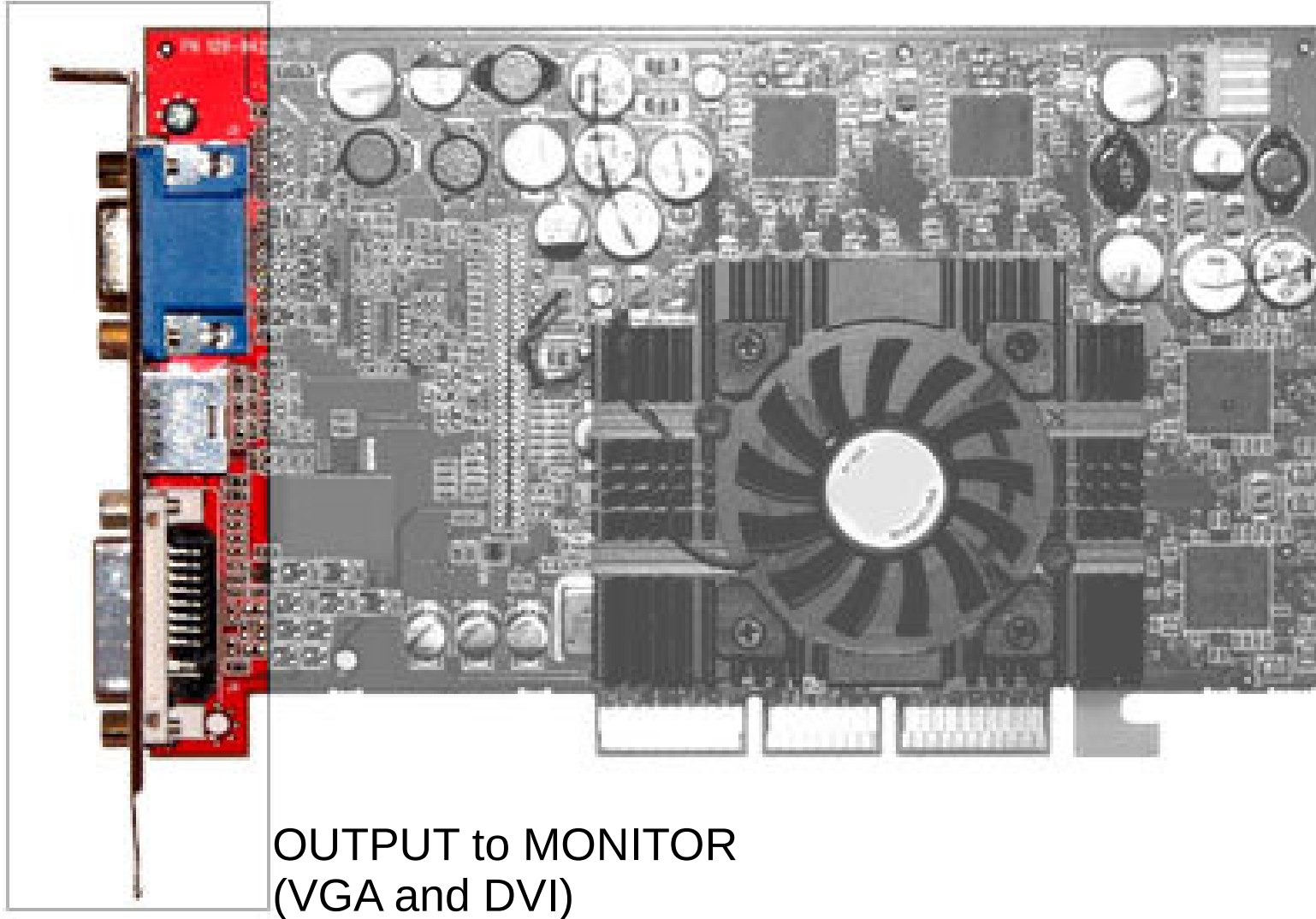
Mostly graphics
accelerator of the
VIDEO CARD,
but in some PC
are in the
MOTHERBOARD



**VIDEO CARDS
WITH GPUS**

4.2 GRAPHICS PROCESSING UNITS (GPUs)

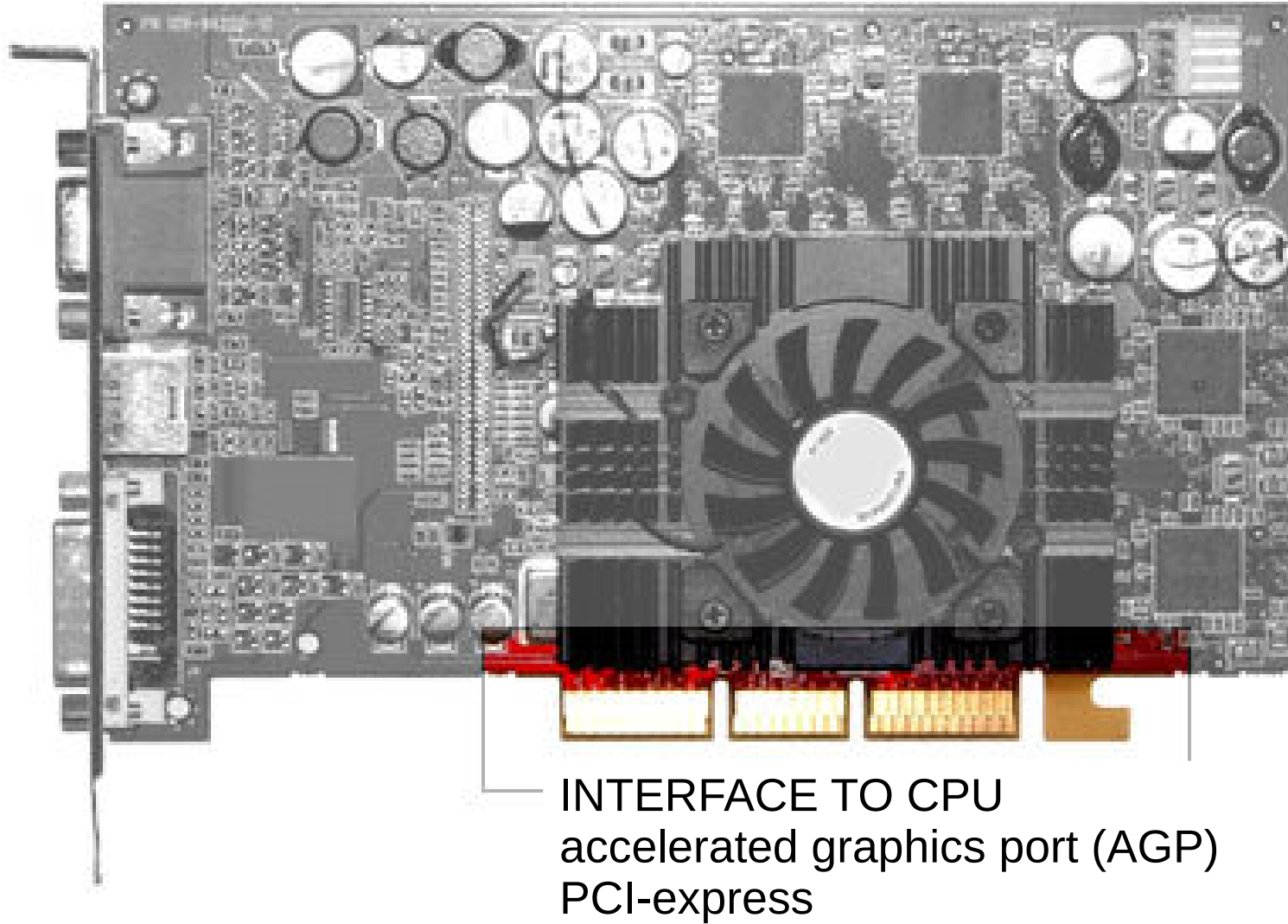
COMPONENTS of a VIDEO CARD



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

4.2 GRAPHICS PROCESSING UNITS (GPUs)

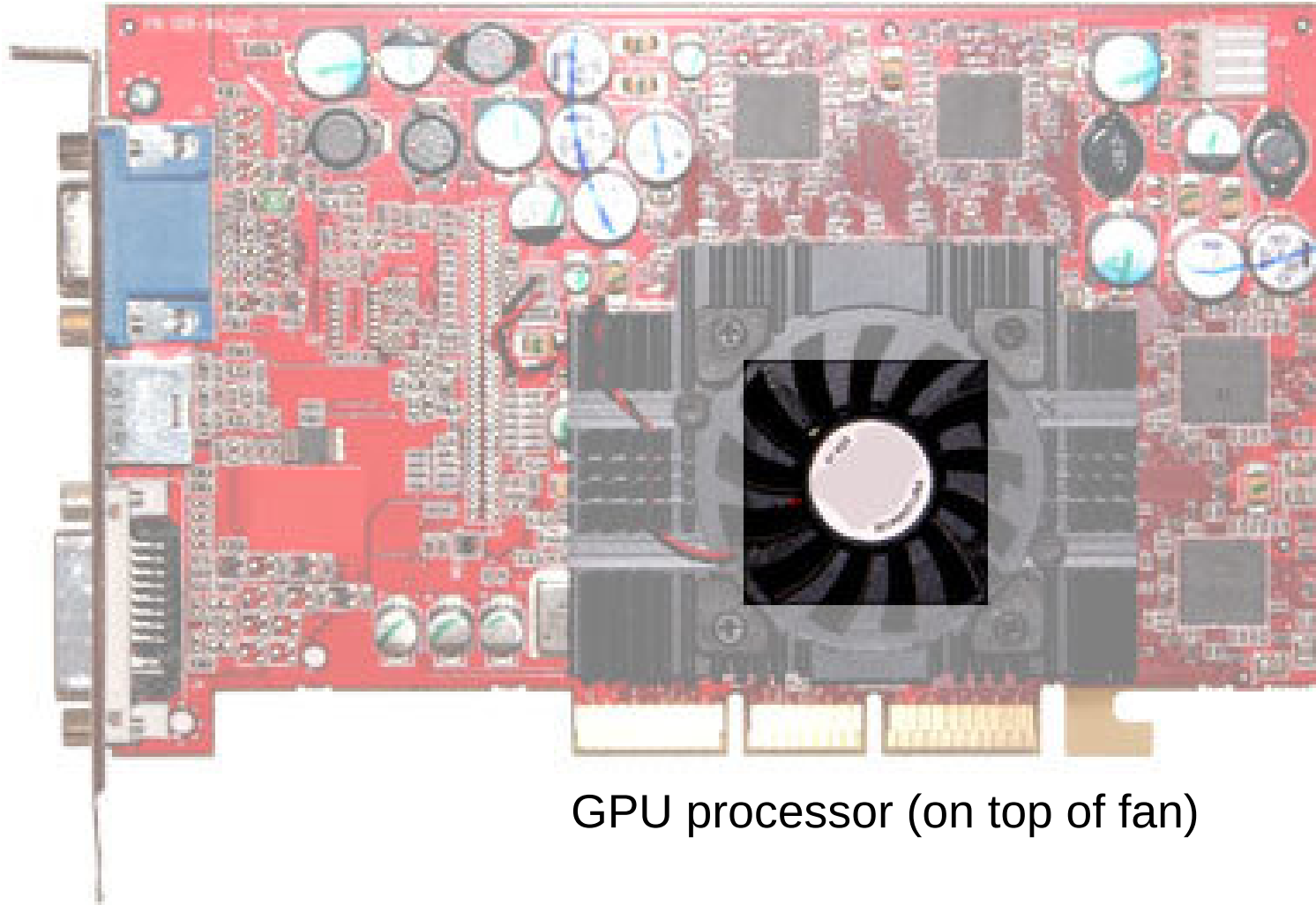
COMPONENTS of a VIDEO CARD



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

4.2 GRAPHICS PROCESSING UNITS (GPUs)

COMPONENTS of a VIDEO CARD

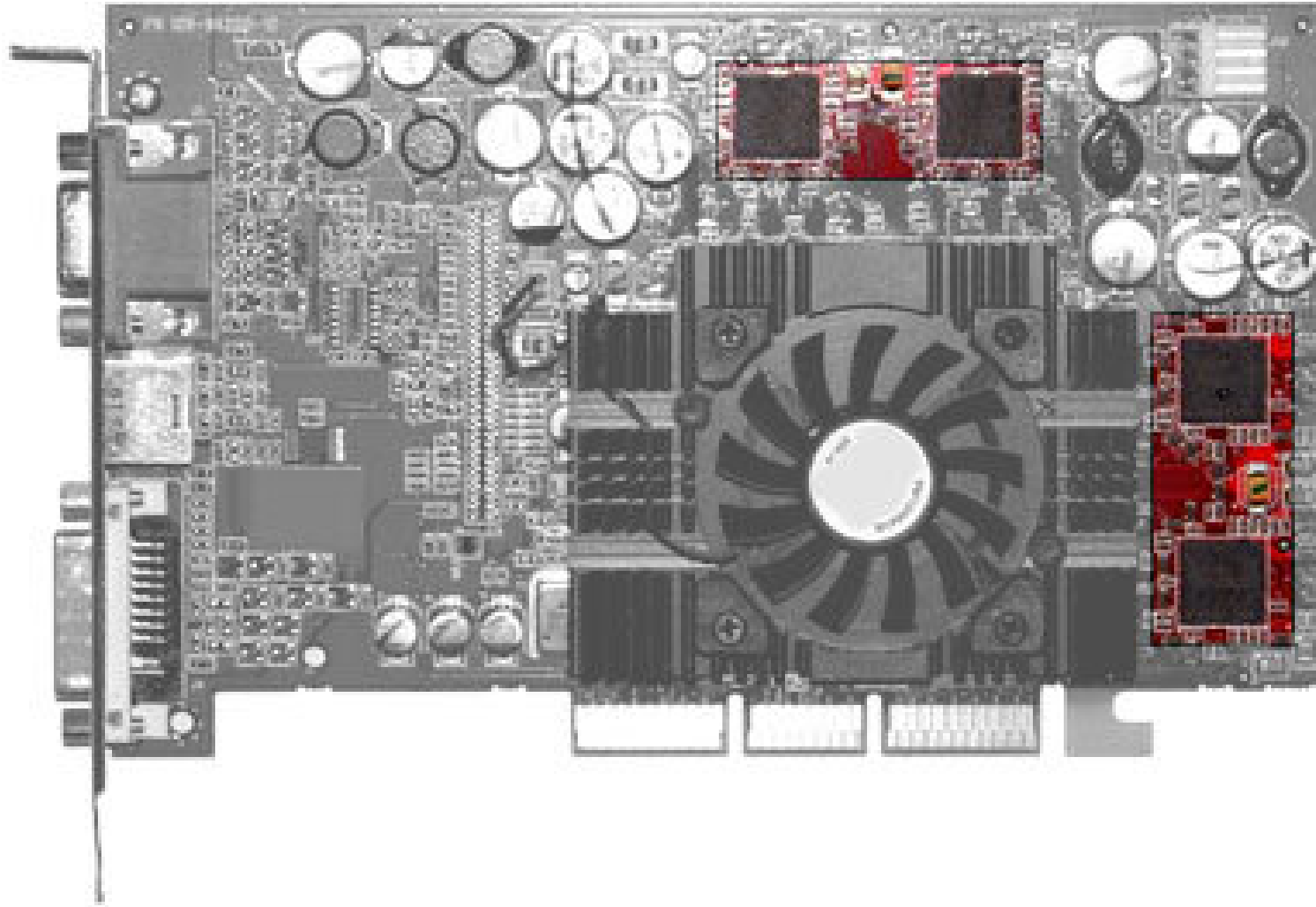


GPU processor (on top of fan)

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 (2nd 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 & Itaka 2007), kirin (Belleman et al. 2008), Yebisu (Nitadori & Makino 2008; Nitadori 2009), Sapporo (Gaburov et al. 2009)

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

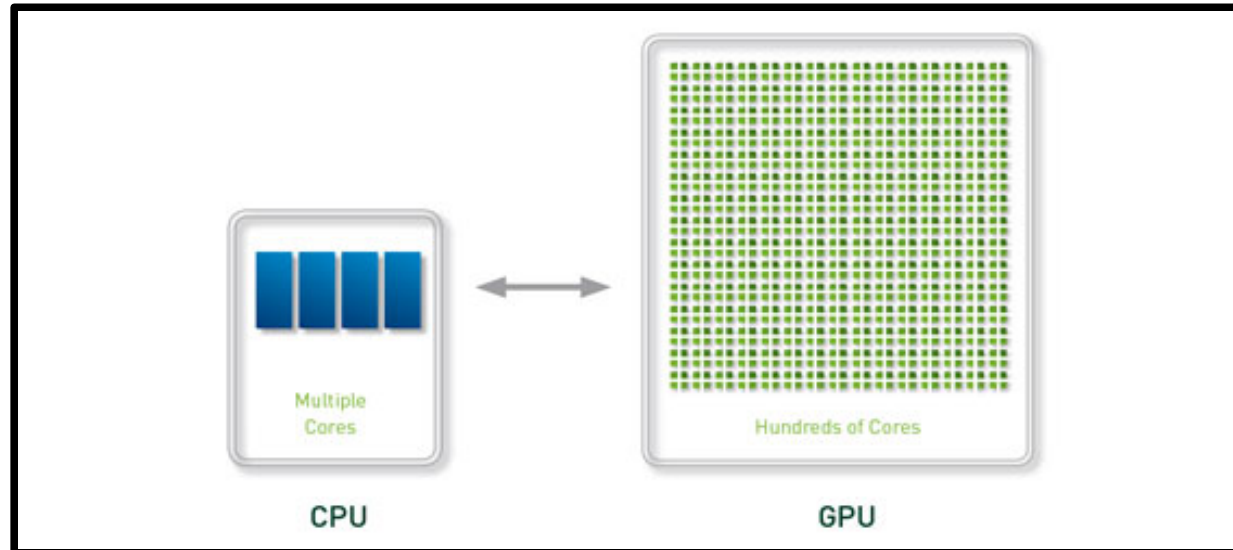
GPUS are composed of many small threads, each able to perform a small instruction (**kernel**), which is the same for all threads but applied on different data

- 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

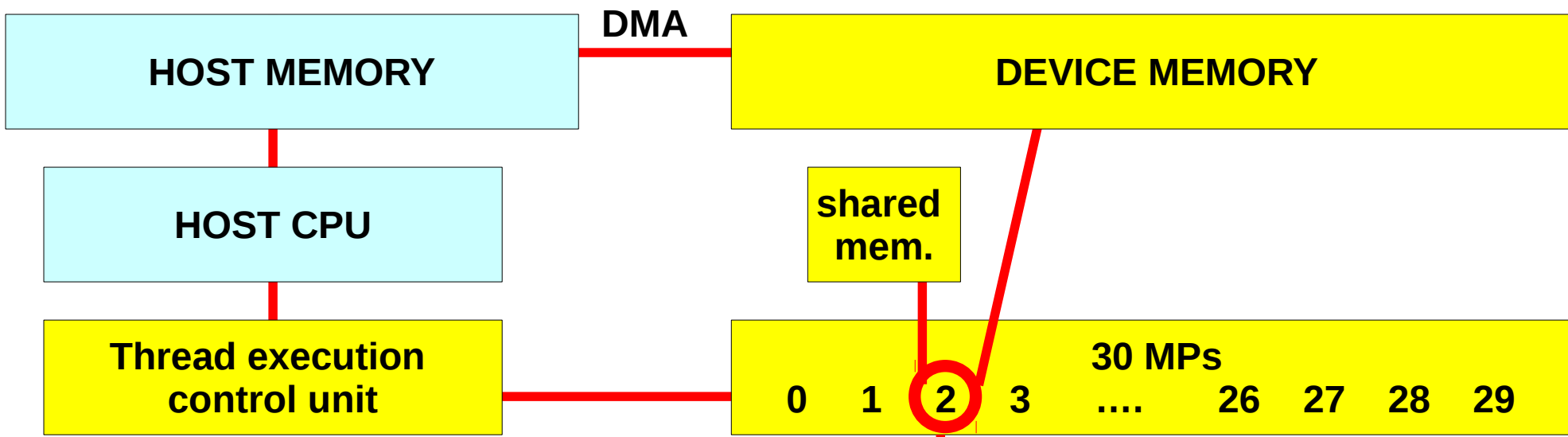
MULTIPLE DATA: N(N-1)/2 ~ N² FORCES

4.2 GRAPHICS PROCESSING UNITS (GPUs)

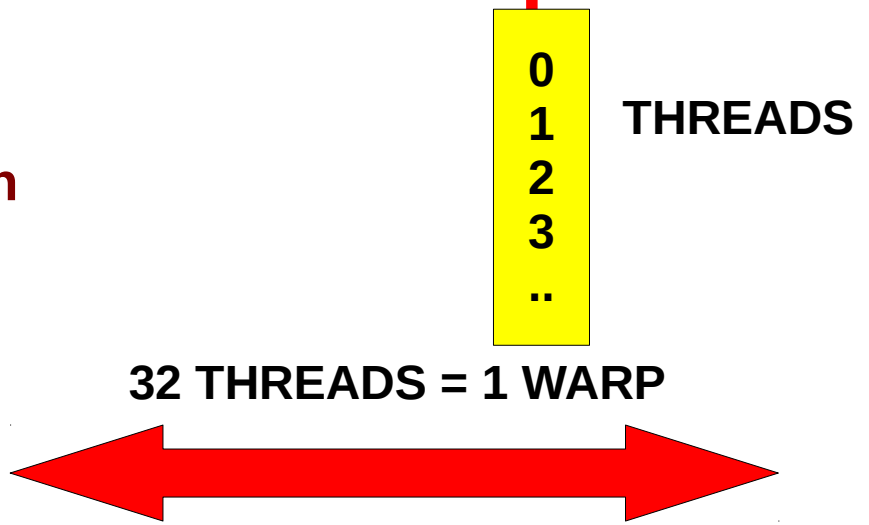
HOW ARE DIRECT N-BODY CODES ADAPTED TO GPUS?

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

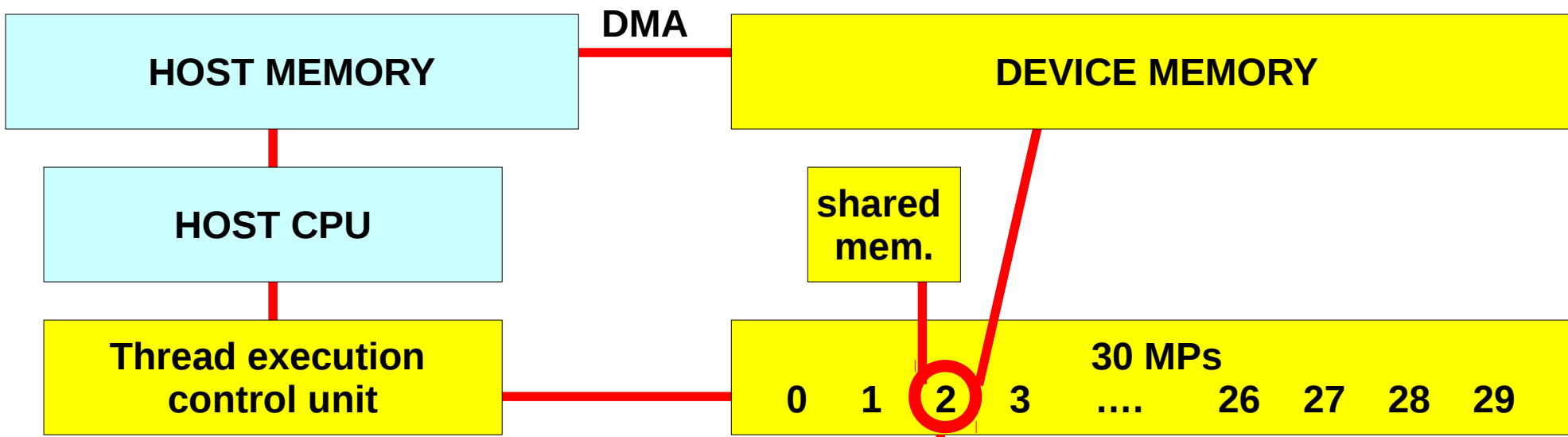
1. inside the GPU – Host := CPU, Device := GPU – EXAMPLE: Tesla C1060



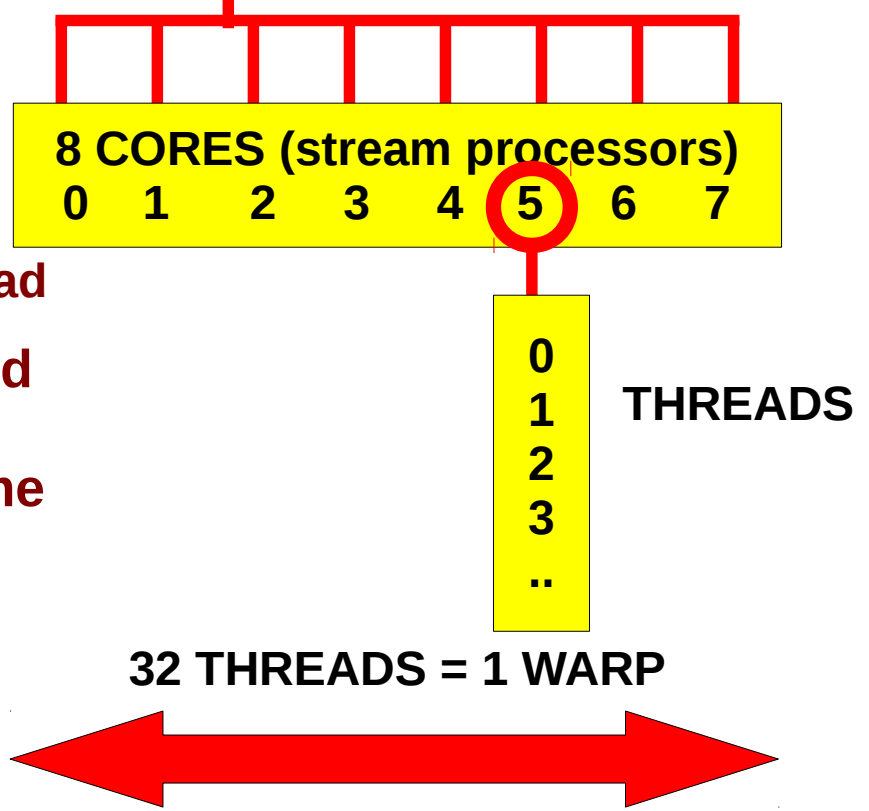
- 30 multiprocessors (MPs)**
- + 1 shared memory per MP**
(16 KB low latency – register-speed - data cache)
- + a single DEVICE MEMORY** for the entire GPU (several GB), slower than shared memory (>100 cycles)
- + Device memory talks with host memory through direct memory access (DMA): even slower (direct access to host memory?)**



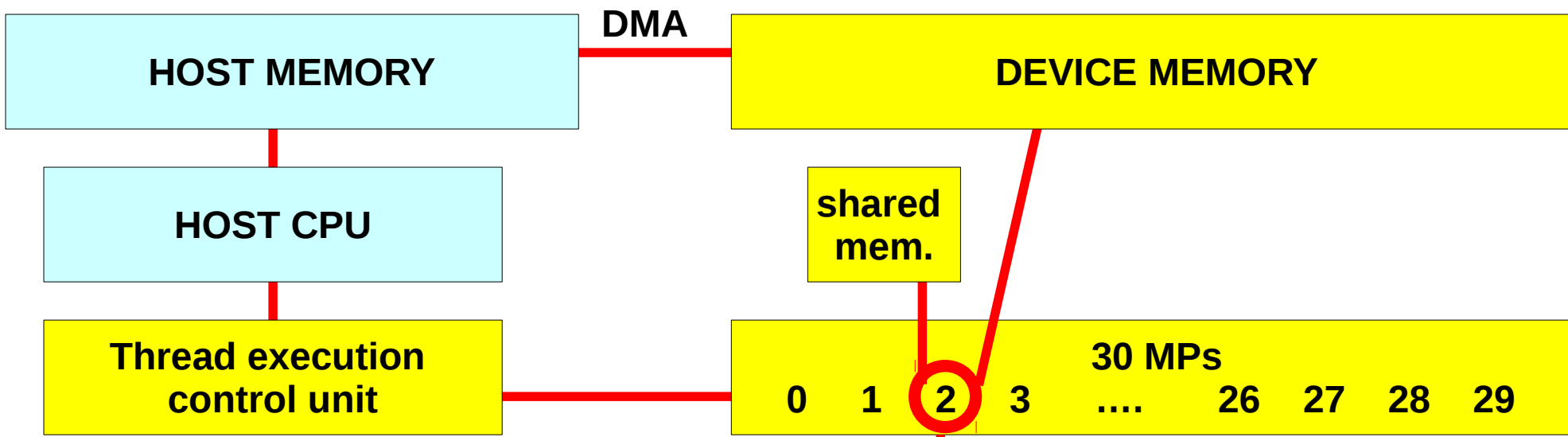
1. inside the GPU – Host := CPU, Device := GPU – EXAMPLE: Tesla C1060



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



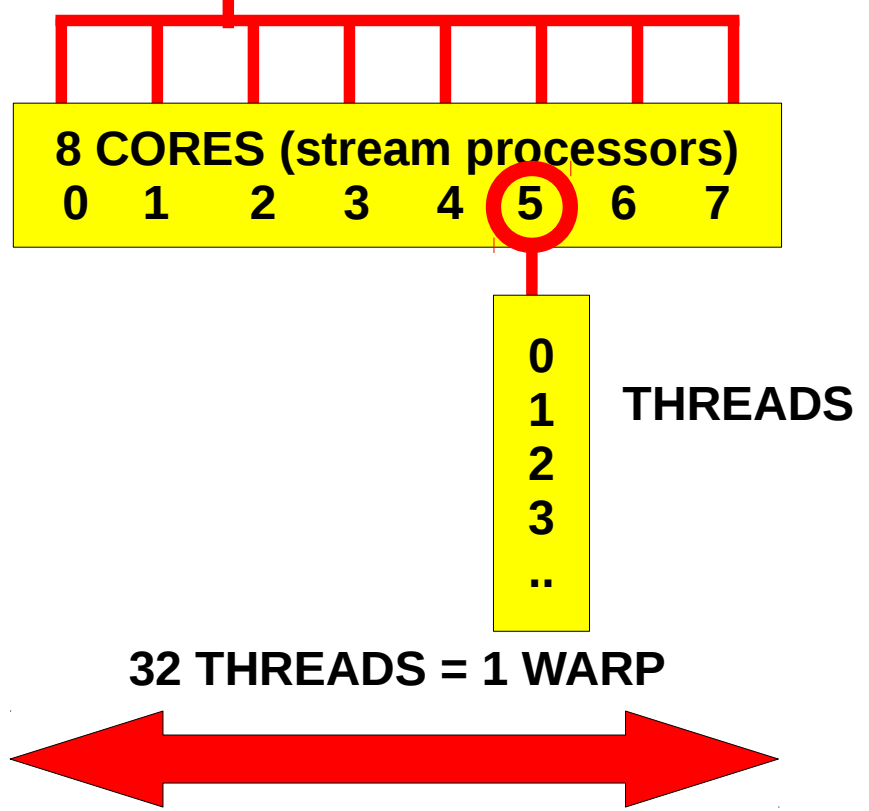
each GROUP of 32 threads connected to the same MP is a WARP

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

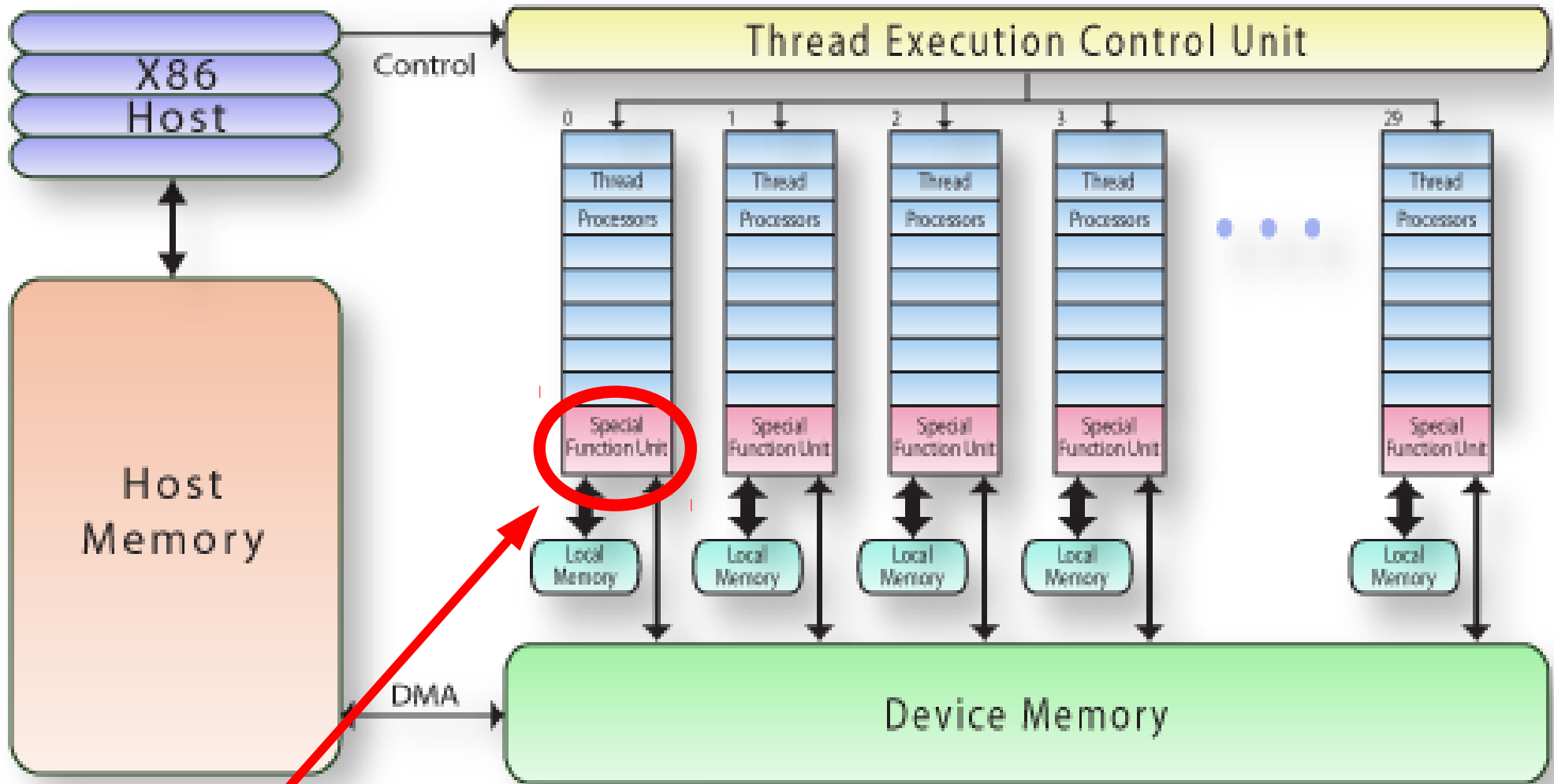
of threads per block is always multiple of # threads per warp

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

3. application to the Hermite scheme

**EXAMPLE: Sapporo library for N-body (Gaburov et al. 2009,
<http://arxiv.org/abs/0902.4463>)**

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=phigrabe>

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

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

1. predictor step: predicts positions and velocities at 3rd 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:

$m \leq n$ because ONLY ACTIVE PARTICLES ARE CORRECTED in the HERMITE PREDICTOR-CORRECTOR !!!

Even $m \ll n$ 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) $\Sigma j = n$
i- particles: sinks of gravitational forces (those on which the force is exerted) $\Sigma i = m$
 $m < n$ because ONLY ACTIVE PARTICLES ARE CORRECTED

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

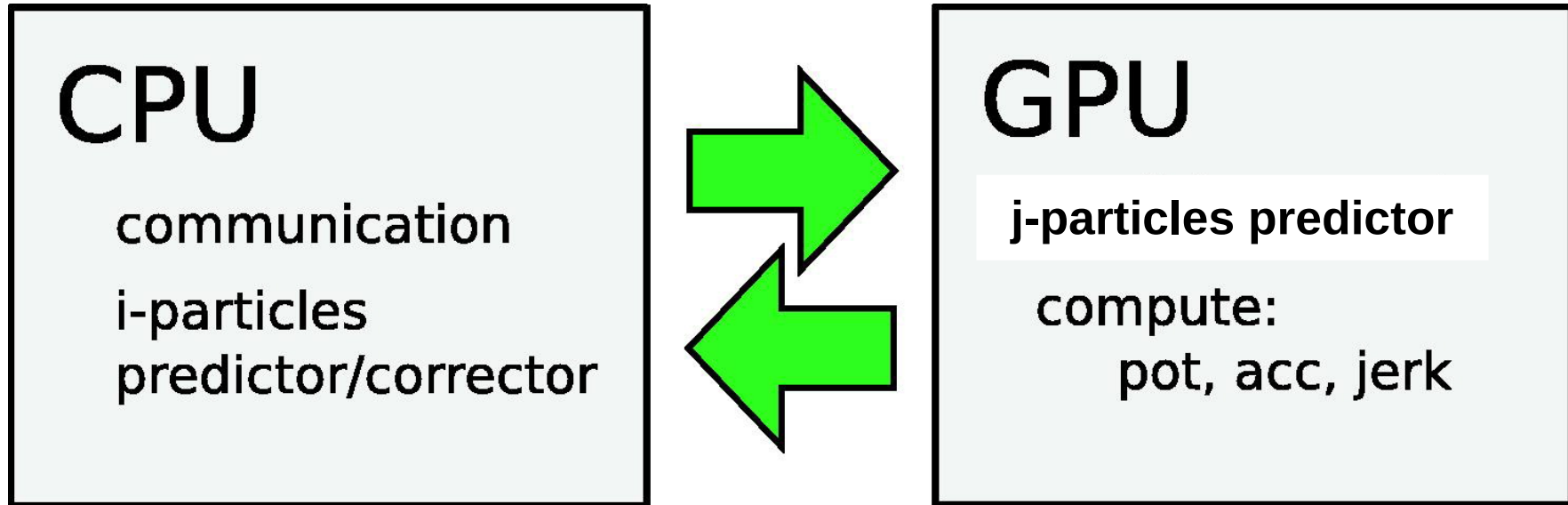
1. predictor step: predicts positions and velocities of the j-particles and i-particles at 3rd order

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

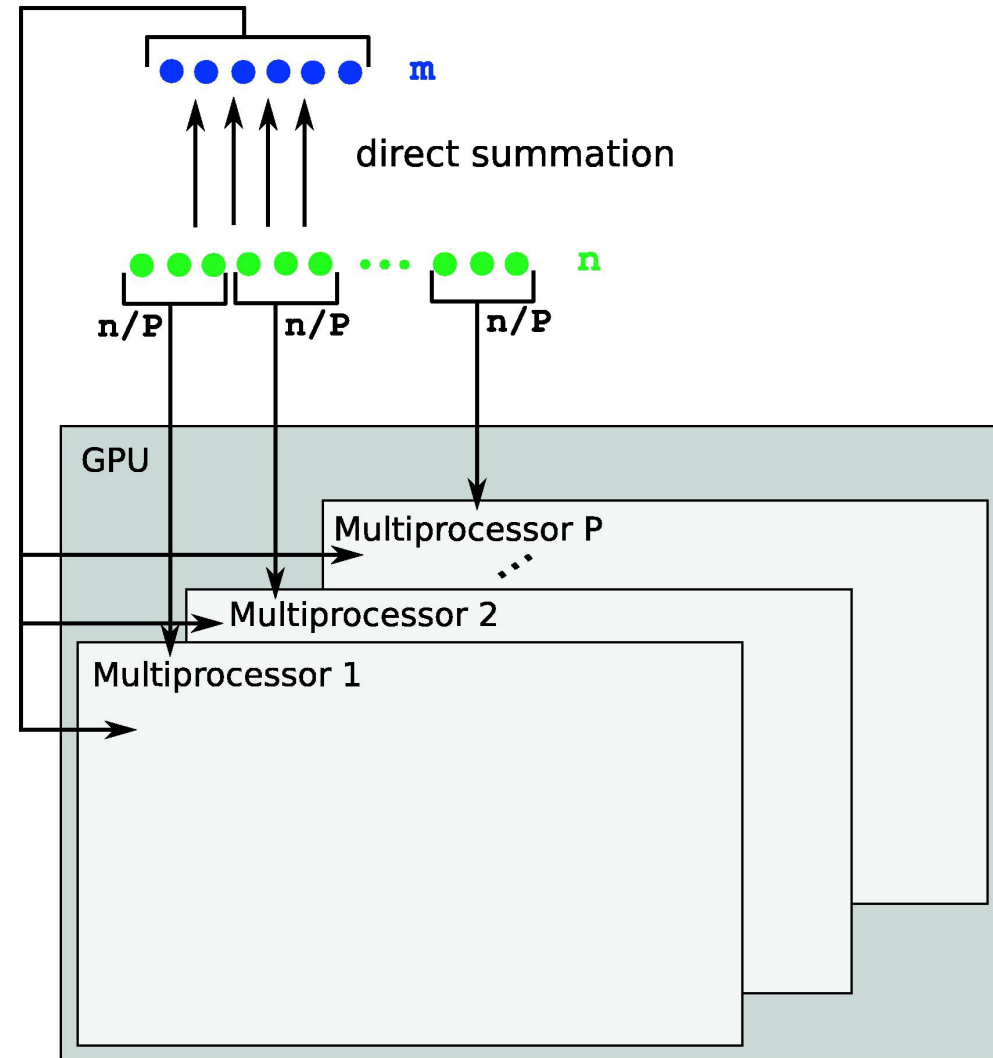
Parallelization:

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.

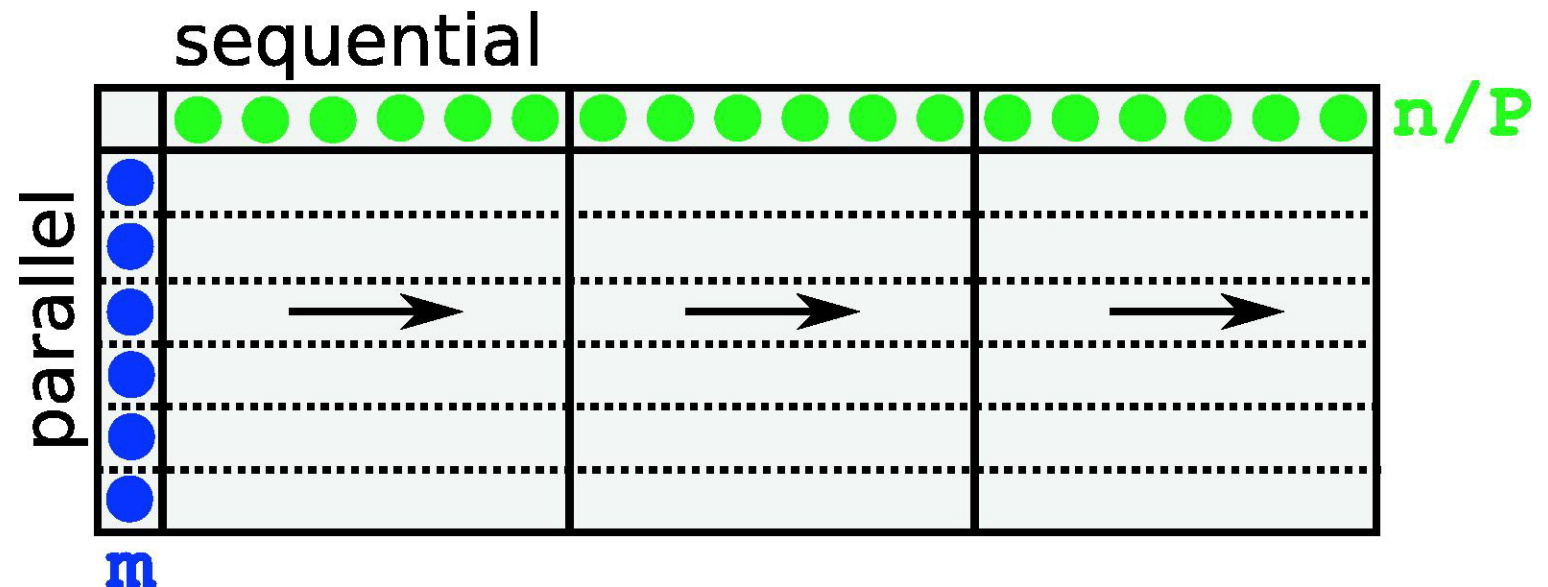


3. application to the Hermite scheme: Sapporo library for N-body

STEP 2 (continues):

if the number of threads in a block is $n_{\text{thread}} \geq m$ each i -particle is assigned to a single thread of each block

if $n_{\text{thread}} < m$, 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 $=n_{\text{thread}}$)
- * 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

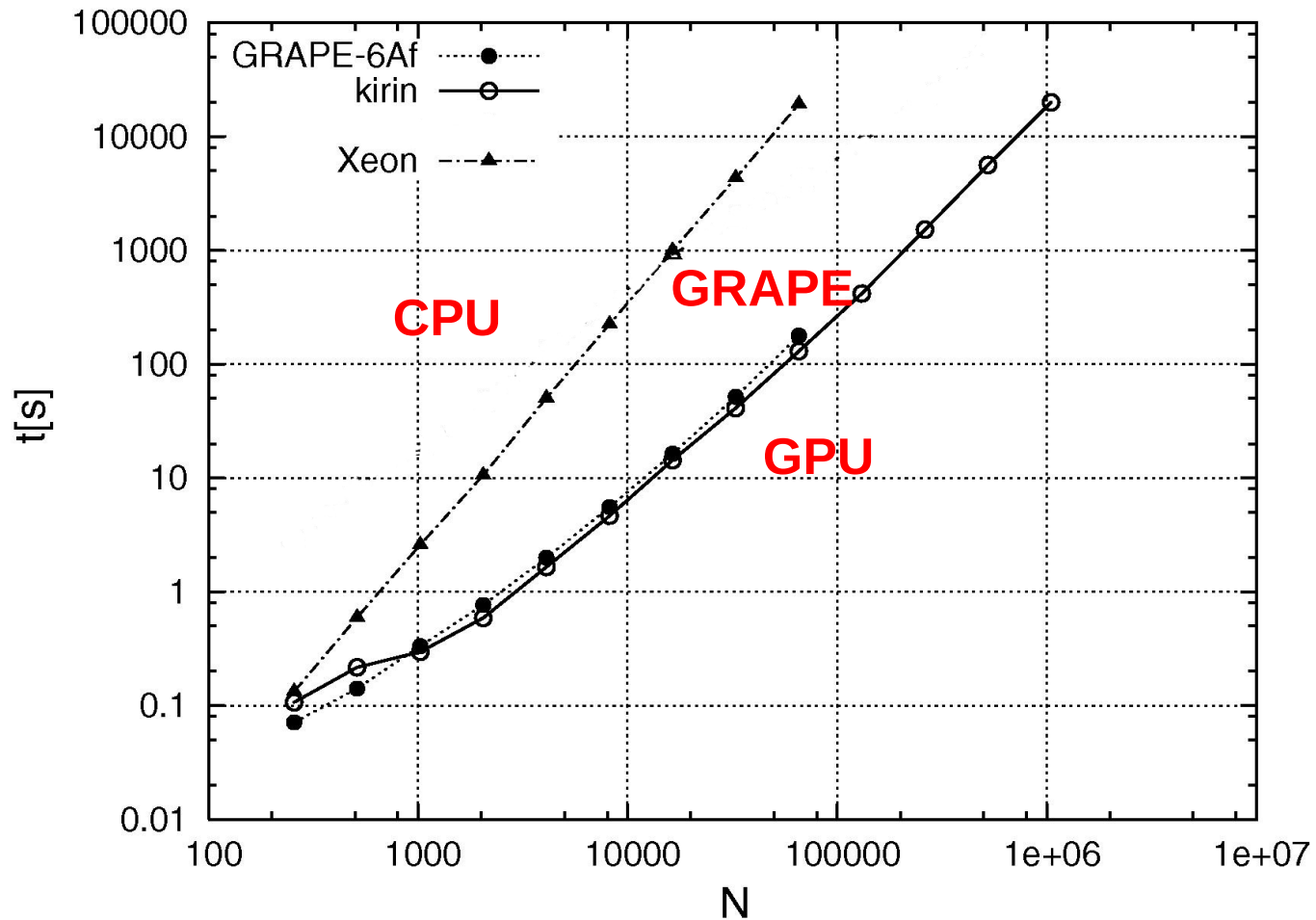
In the host (=CPU) the positions and velocity of the active m particles (the i -particles) are corrected according to:

$$v_1 = v_0 + \frac{1}{2} (a_0 + a_{p,1}) \Delta t + \frac{1}{12} (j_0 - j_{p,1}) \Delta t^2$$
$$x_1 = x_0 + \frac{1}{2} (v_0 + v_1) \Delta t + \frac{1}{12} (a_0 - a_{p,1}) \Delta t^2$$

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:



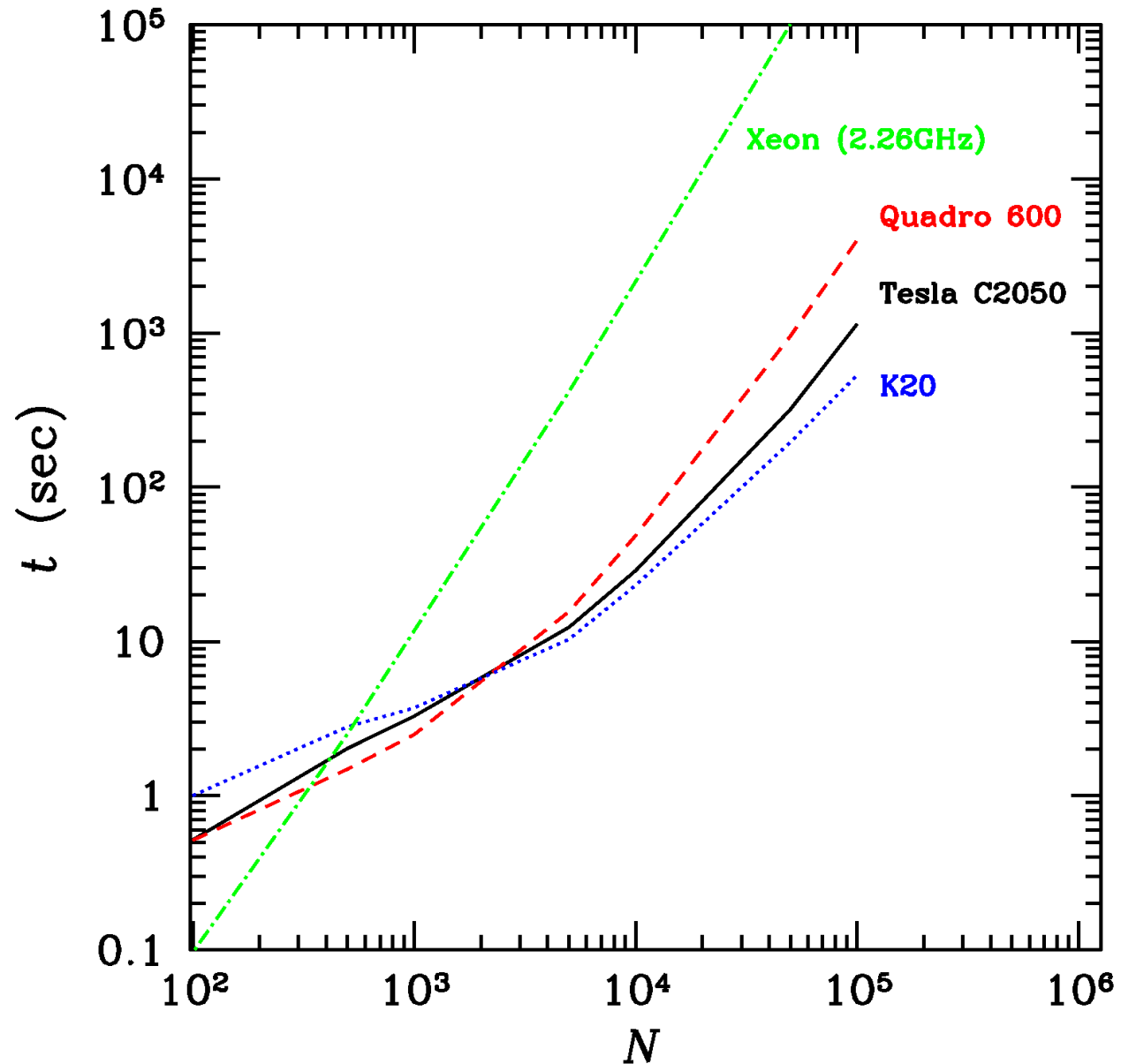
NOTE: SAPPORO WORKS IN PARALLEL ON ALL THE GPU DEVICES CONNECTED TO THE SAME HOST thanks to the GPUWorker library, which is part of the **HOOMD molecular dynamics GPU code (Anderson et al. 2008)**
→ If each node has 2 or 4 GPUs, you can use all the 2 or 4 GPUs

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

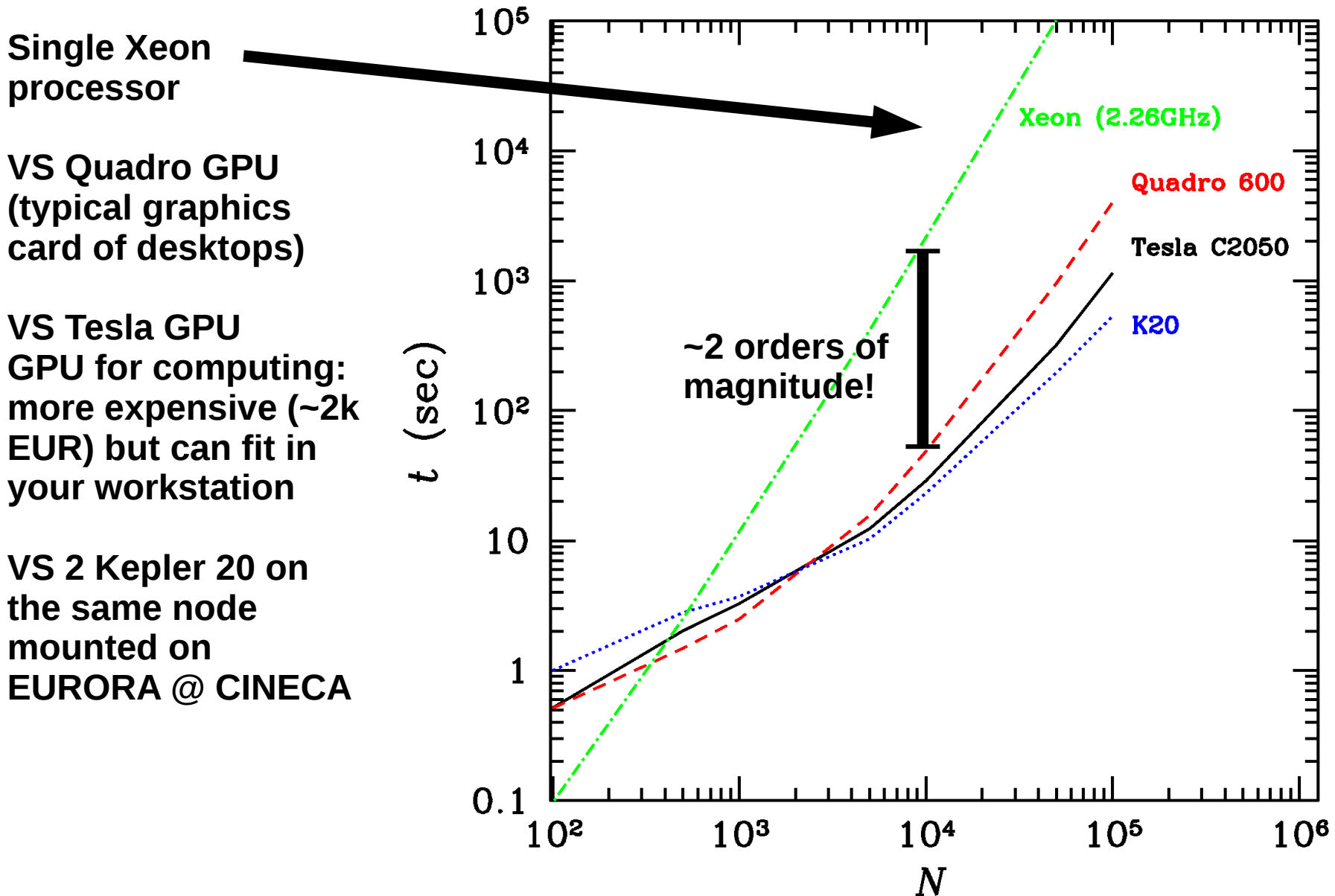
for a star cluster
with N particles



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)

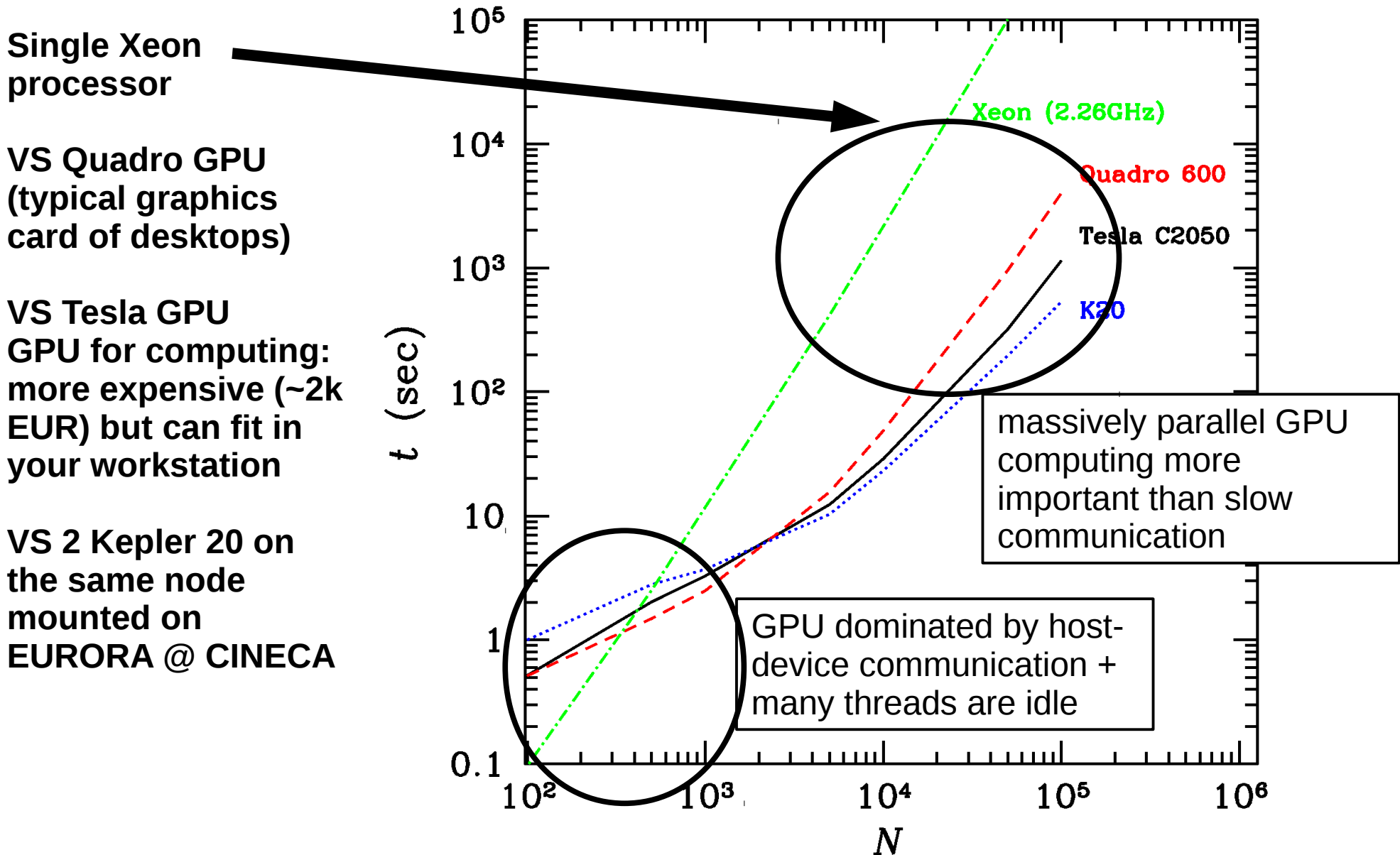
VS Tesla GPU
GPU for computing: more expensive (~2k EUR) but can fit in your workstation

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

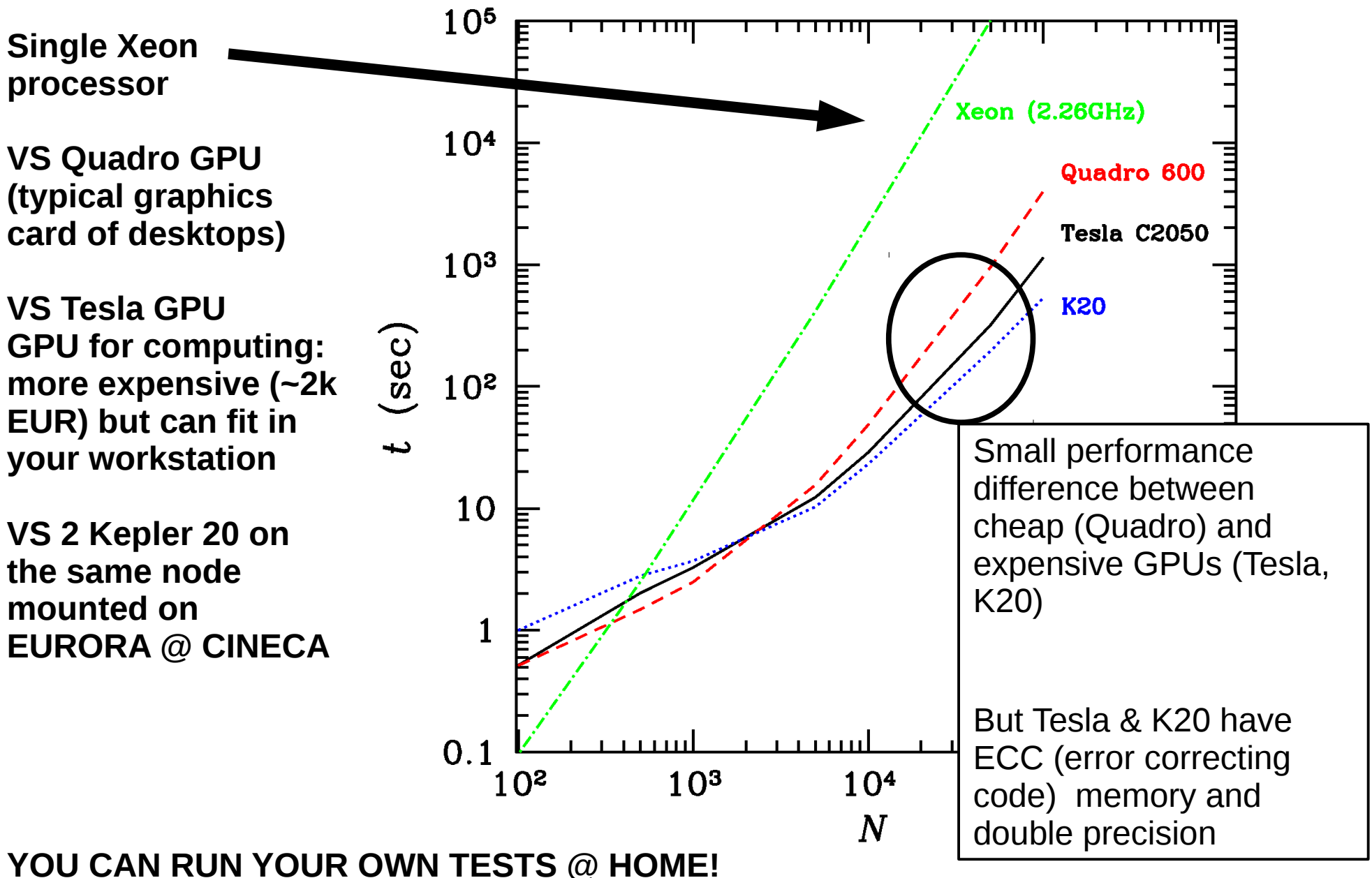
This implementation of Hermite with Sapporo allows to reach the performance I showed before:



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:



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

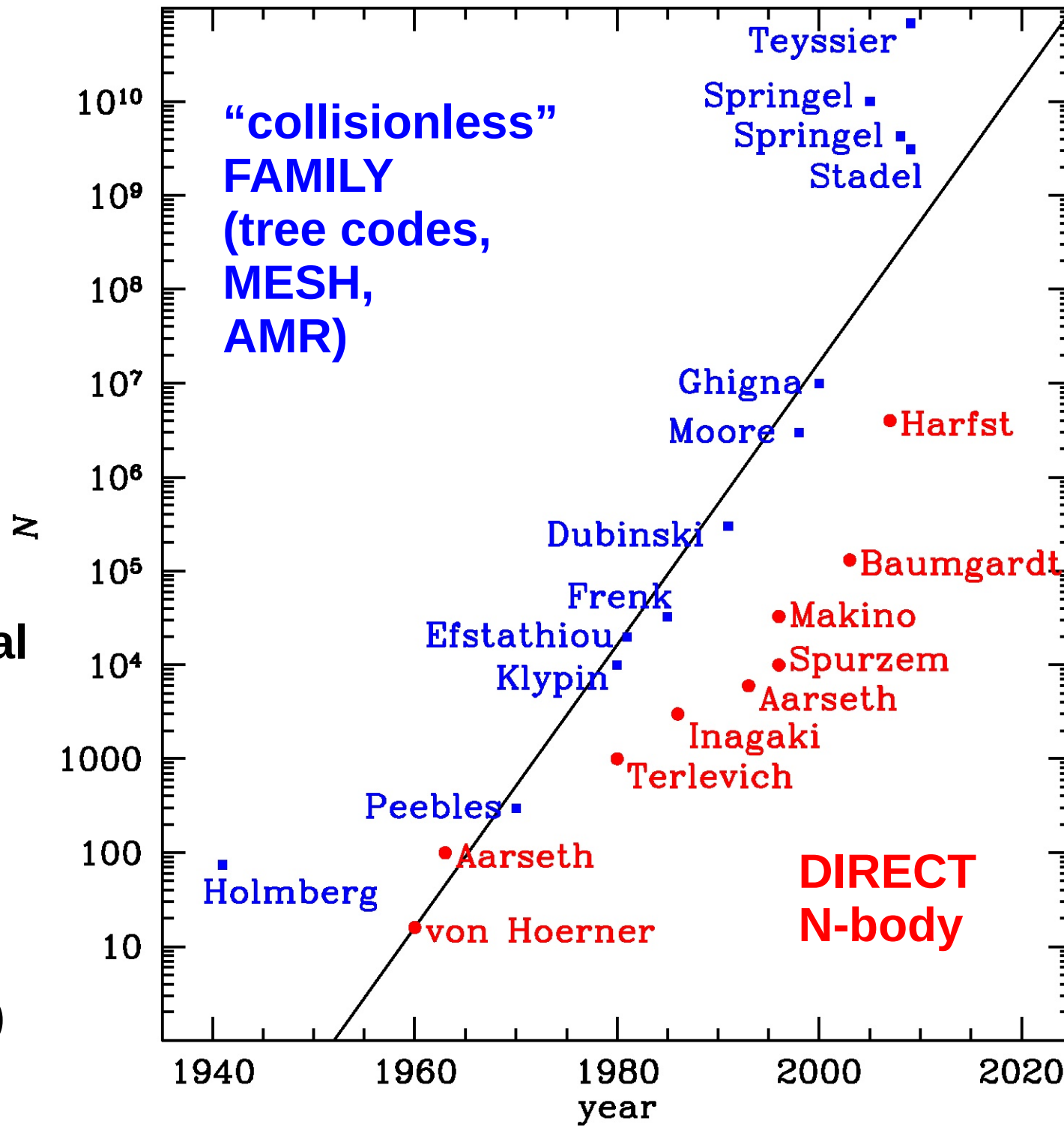
2 Xeon E5-2687W
3.10 GHz per node

2 NVIDIA K20 per
node (64 cards now)

4.2 GRAPHICS PROCESSING UNITS (GPUs)

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. $p = 4, n = 24 \rightarrow q = 6$

p0

| | | | | | |
|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |

p1

| | | | | | |
|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |

p2

| | | | | | |
|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |

p3

| | | | | | |
|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |

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 \rightarrow the particles with shorter Δt are **ACTIVE** and forces must be updated

p0

| | | | | | |
|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |

p1

| | | | | | |
|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |

p2

| | | | | | |
|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |

p3

| | | | | | |
|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |

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

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

p_0

| | | | | | |
|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |

calculates forces by $n-1$ particles on

$i=2$

p_1

| | | | | | |
|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |

calculates forces by $n-1$ particles on

$i=8, 10$

p_2

| | | | | | |
|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |

calculates forces by $n-1$ particles on

$i=17$

p_3

| | | | | | |
|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |

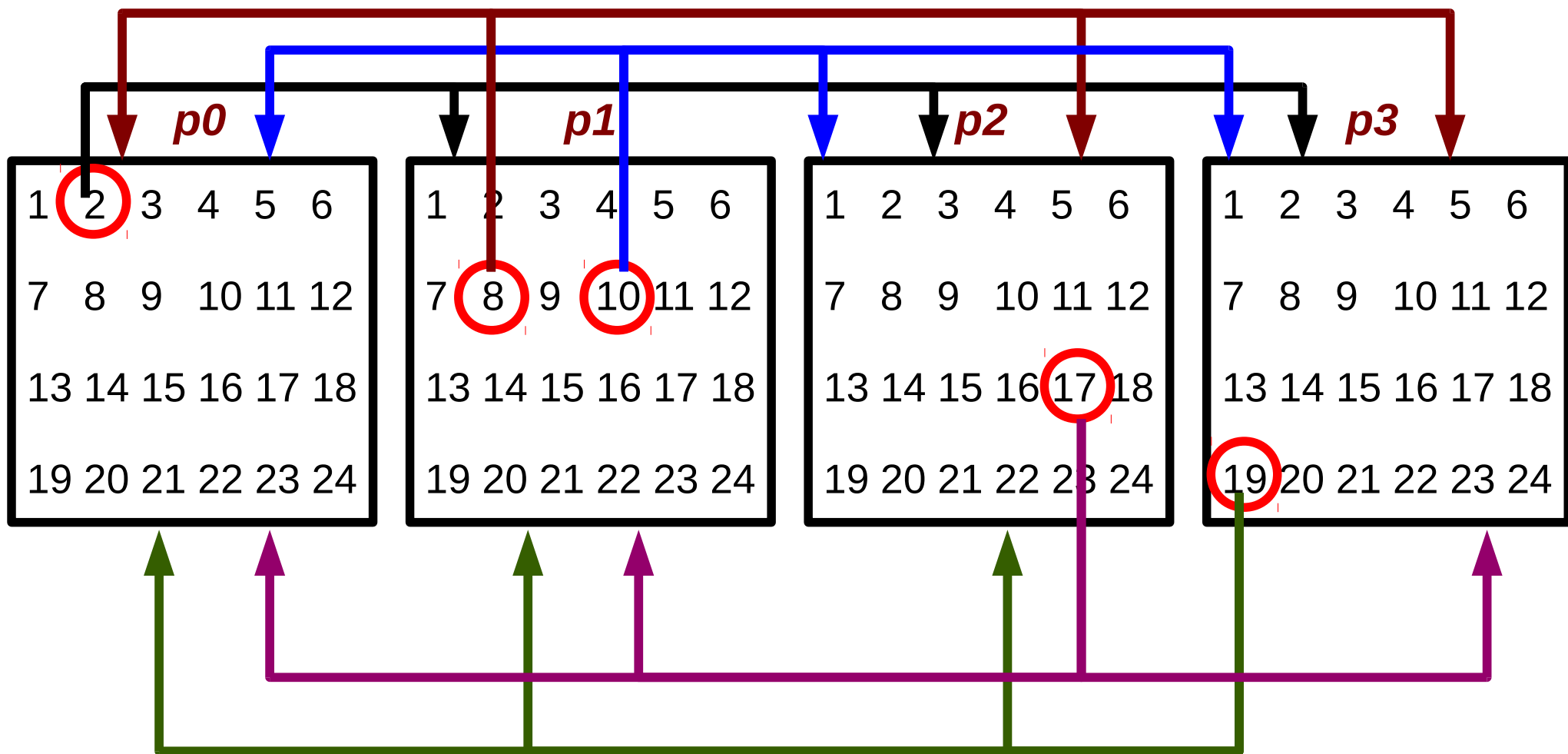
calculates forces by $n-1$ particles on

$i=19$

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?

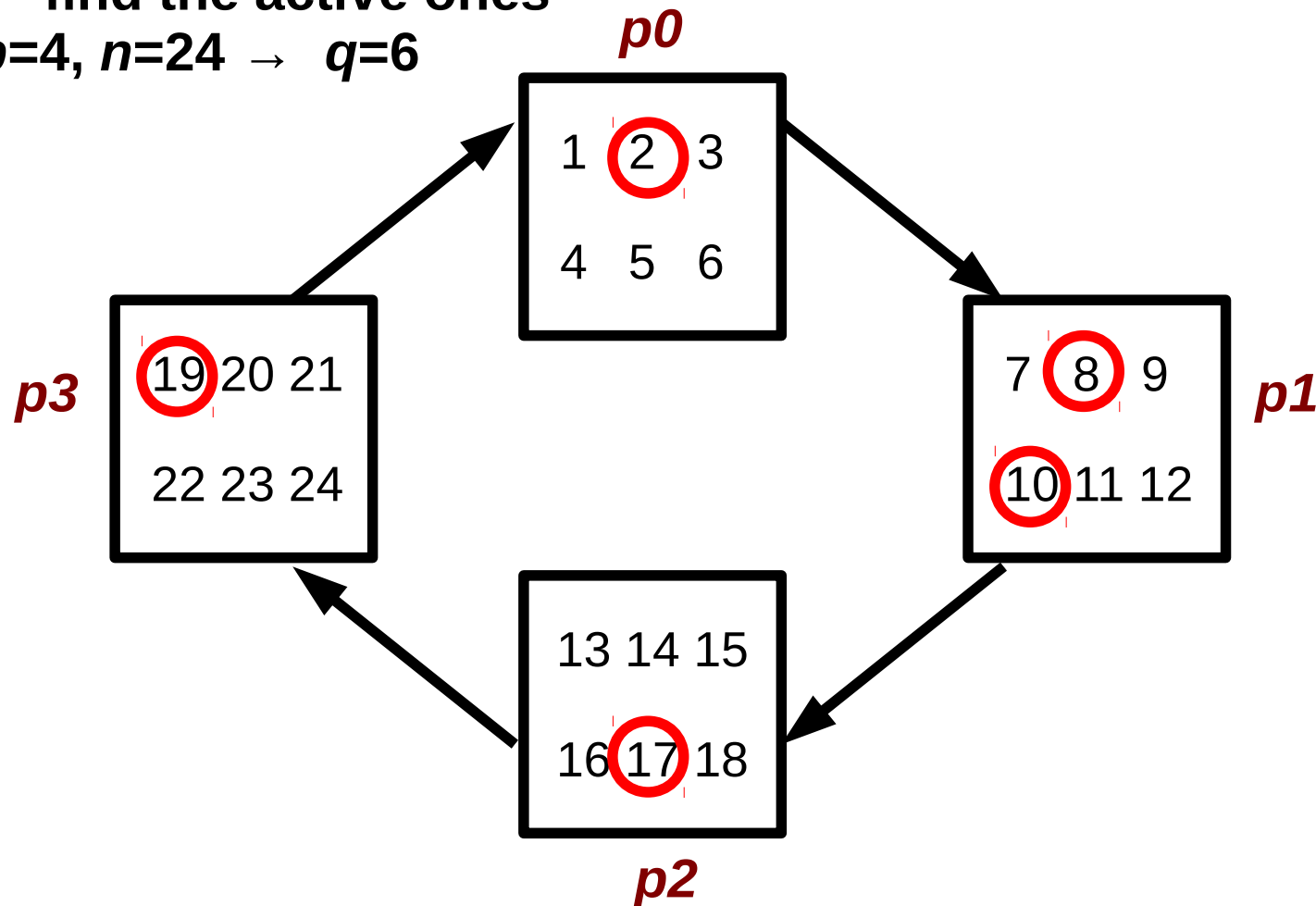
RING ALGORITHM or SYSTOLIC ALGORITHM:

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. $p=4, n=24 \rightarrow q=6$



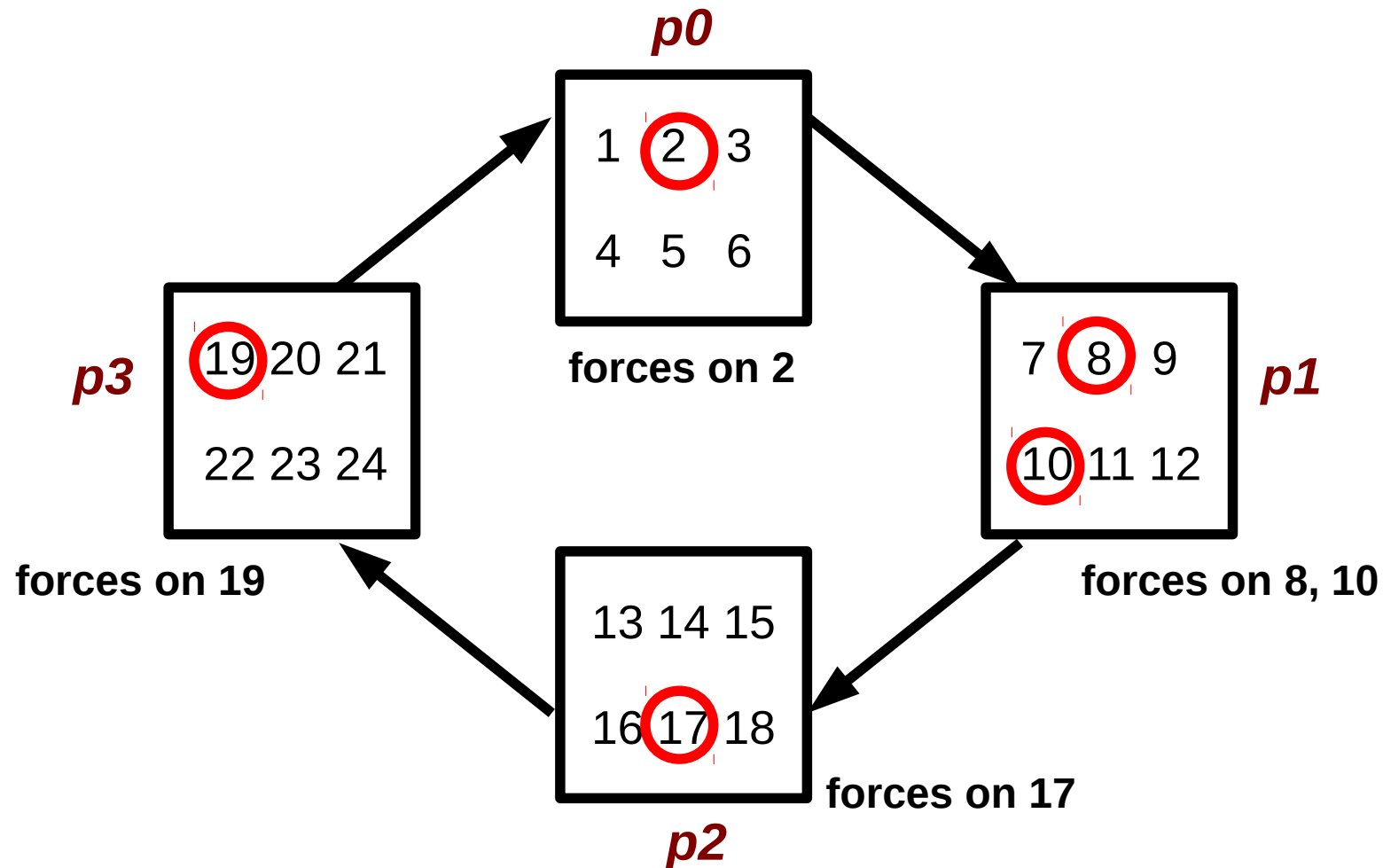
5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

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



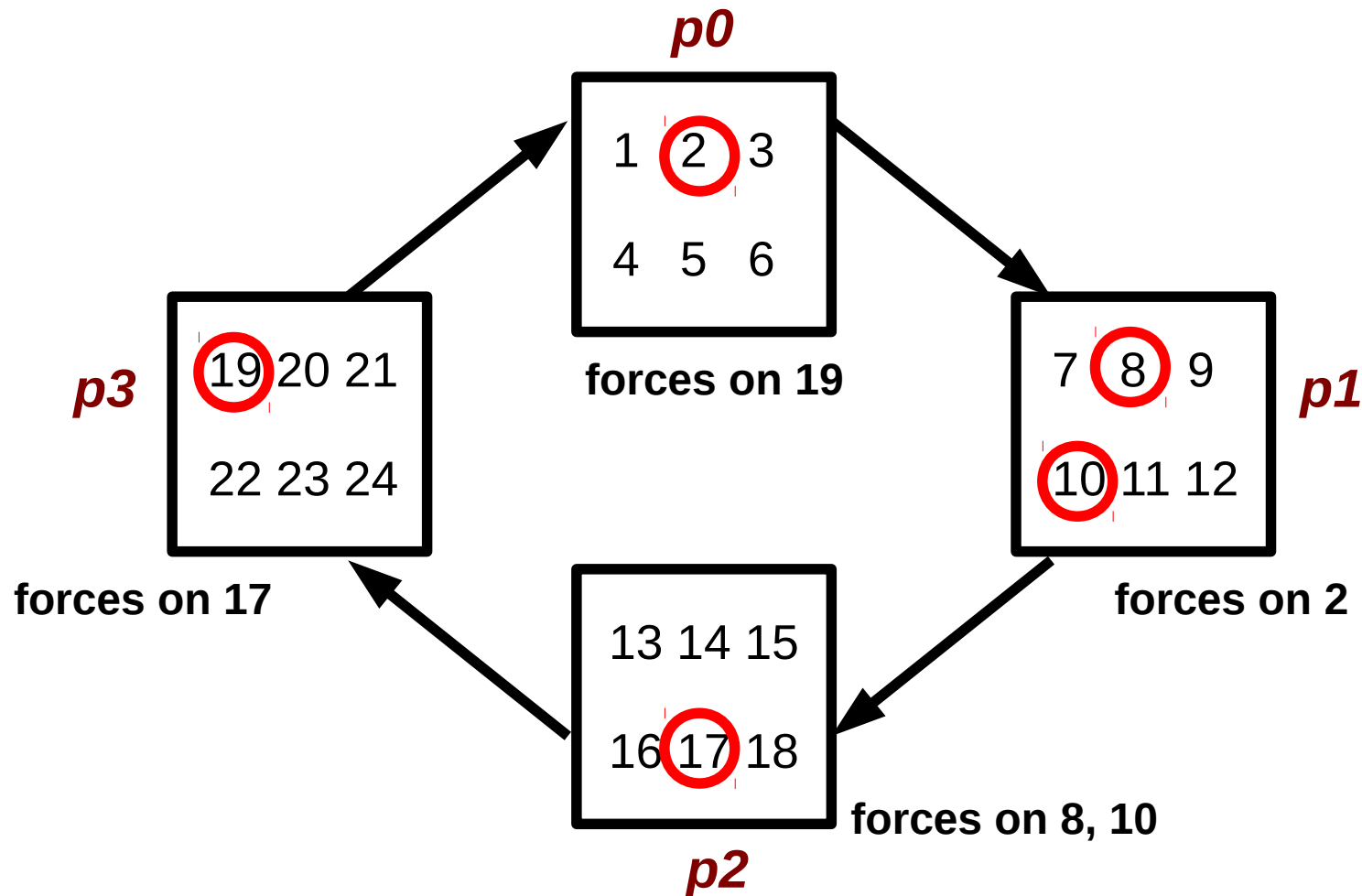
5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

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)



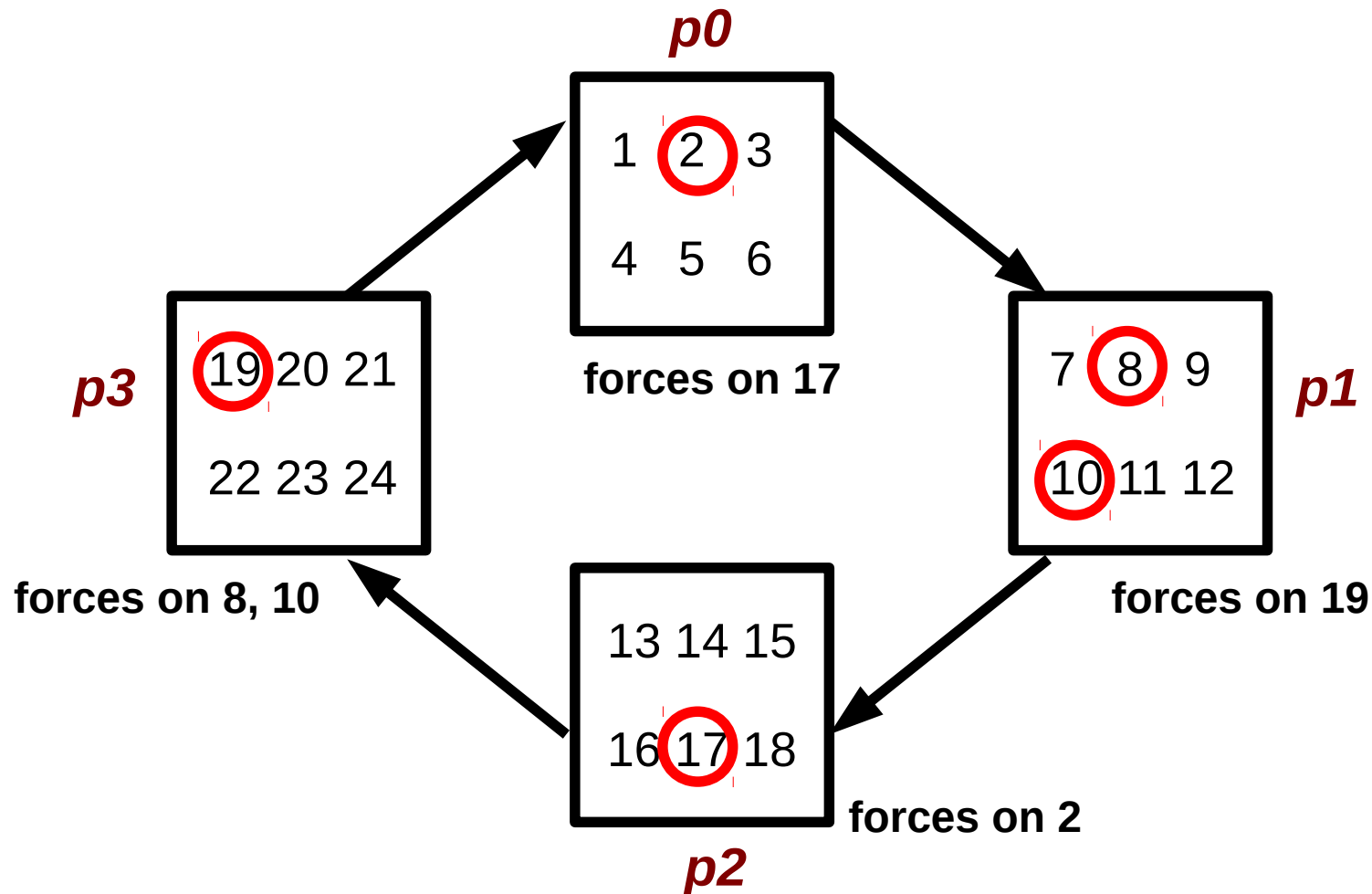
5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

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)



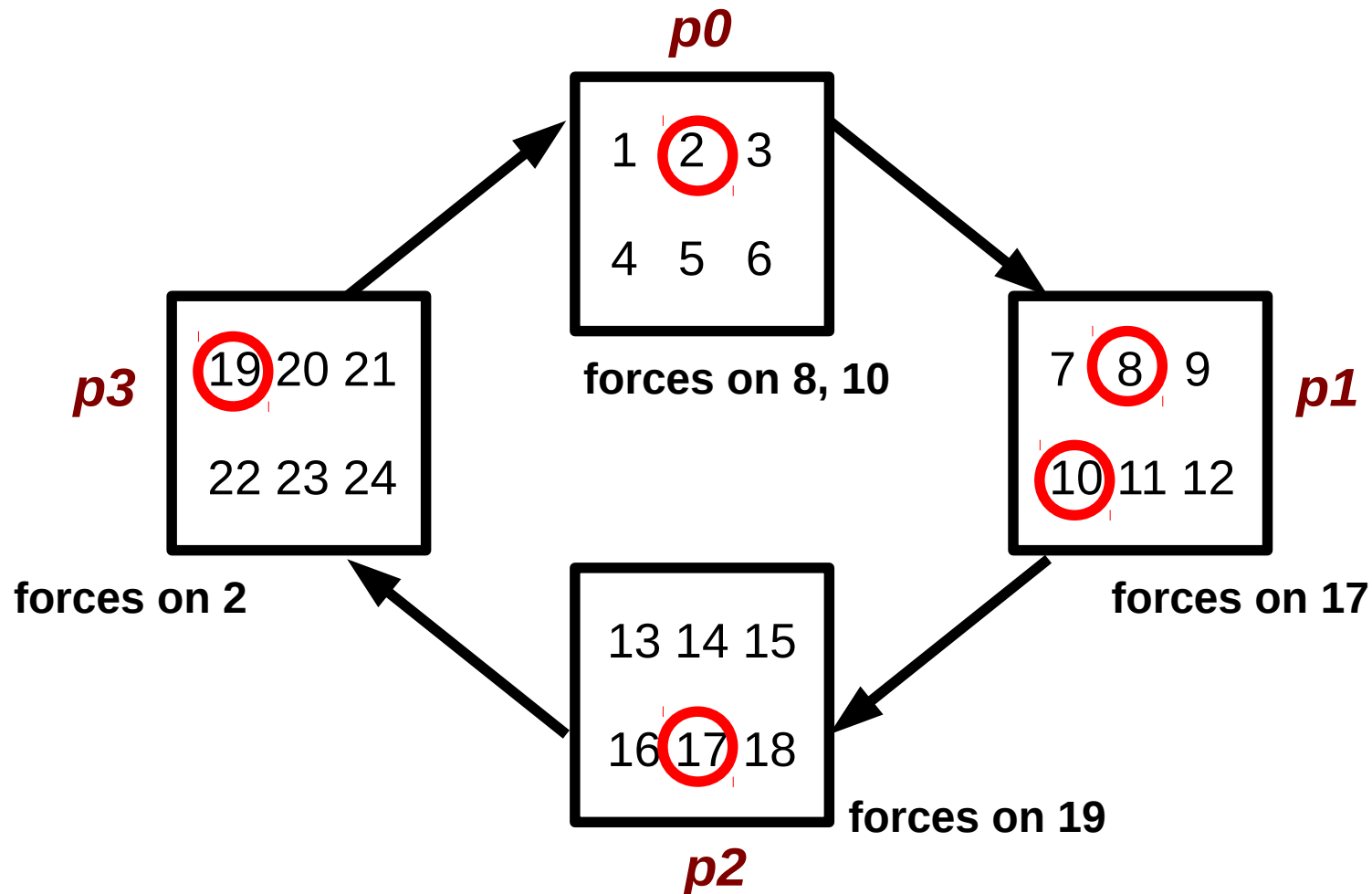
5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

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)



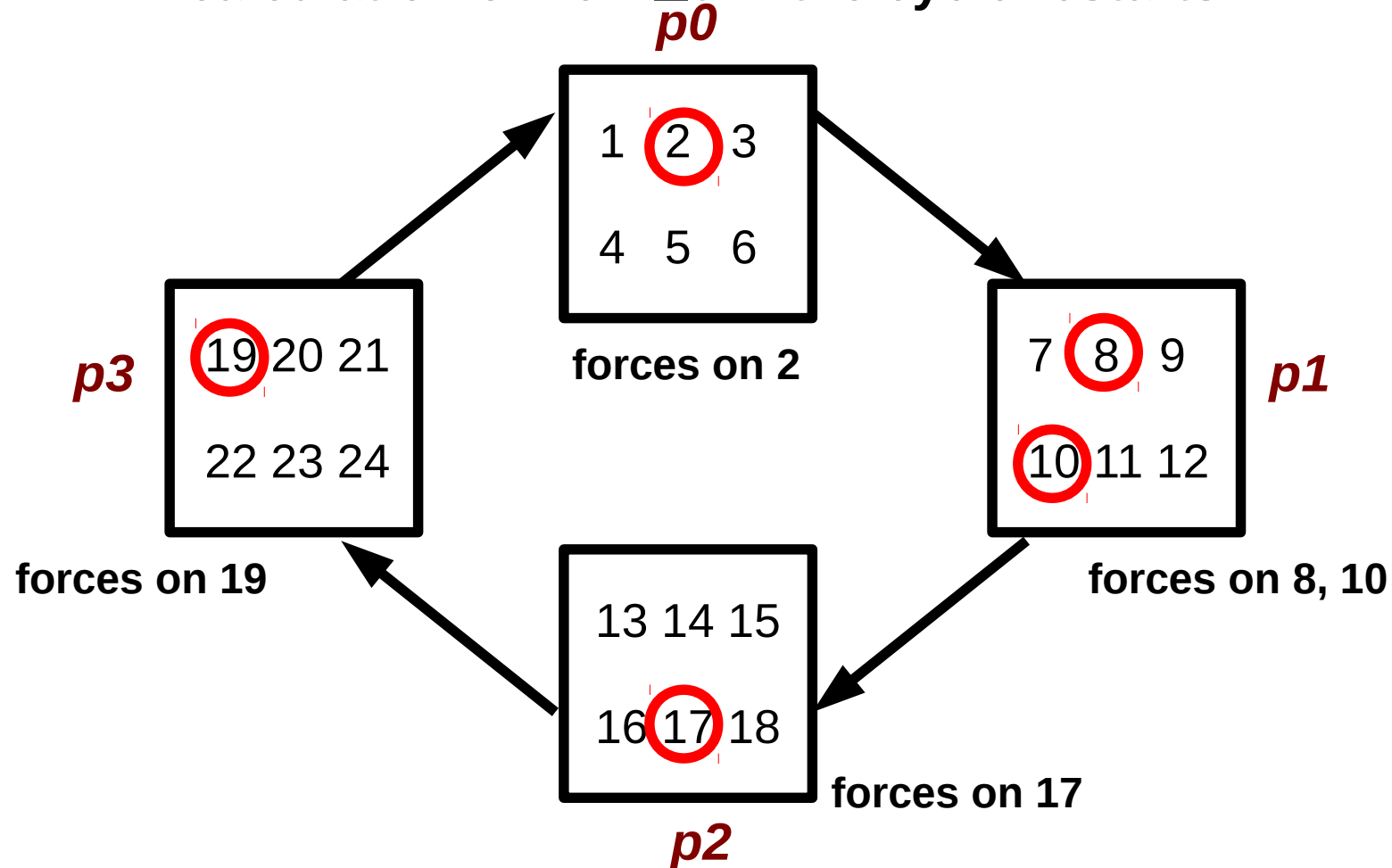
5. MPI? WHAT ABOUT PARALLEL direct summation N-body codes on CPU clusters?

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 $\Delta t \rightarrow$ 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)

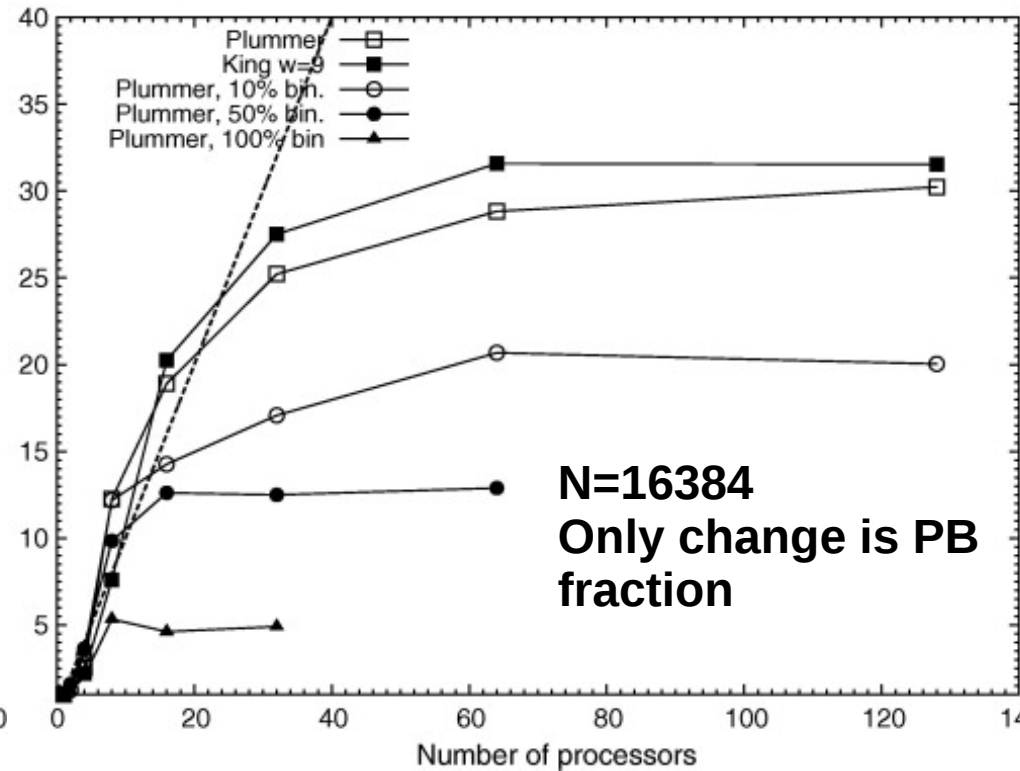
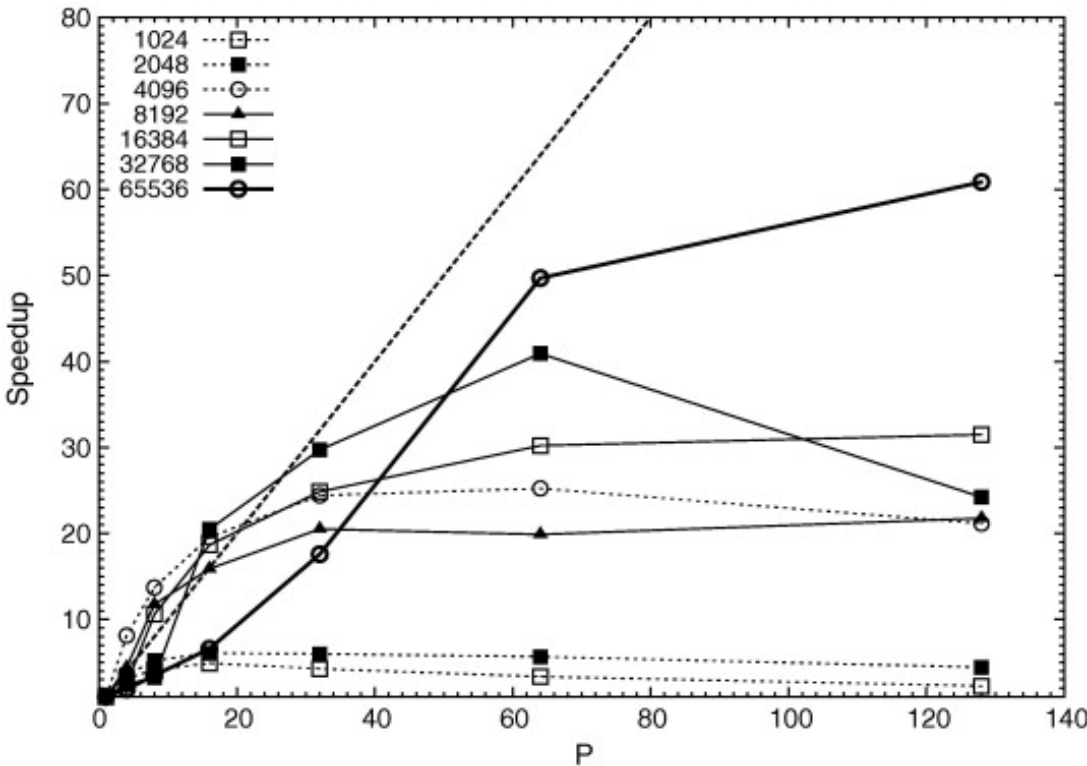
**With GPUs the list of perturbers is in the device memory!
(still bottleneck but not so serious)**

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

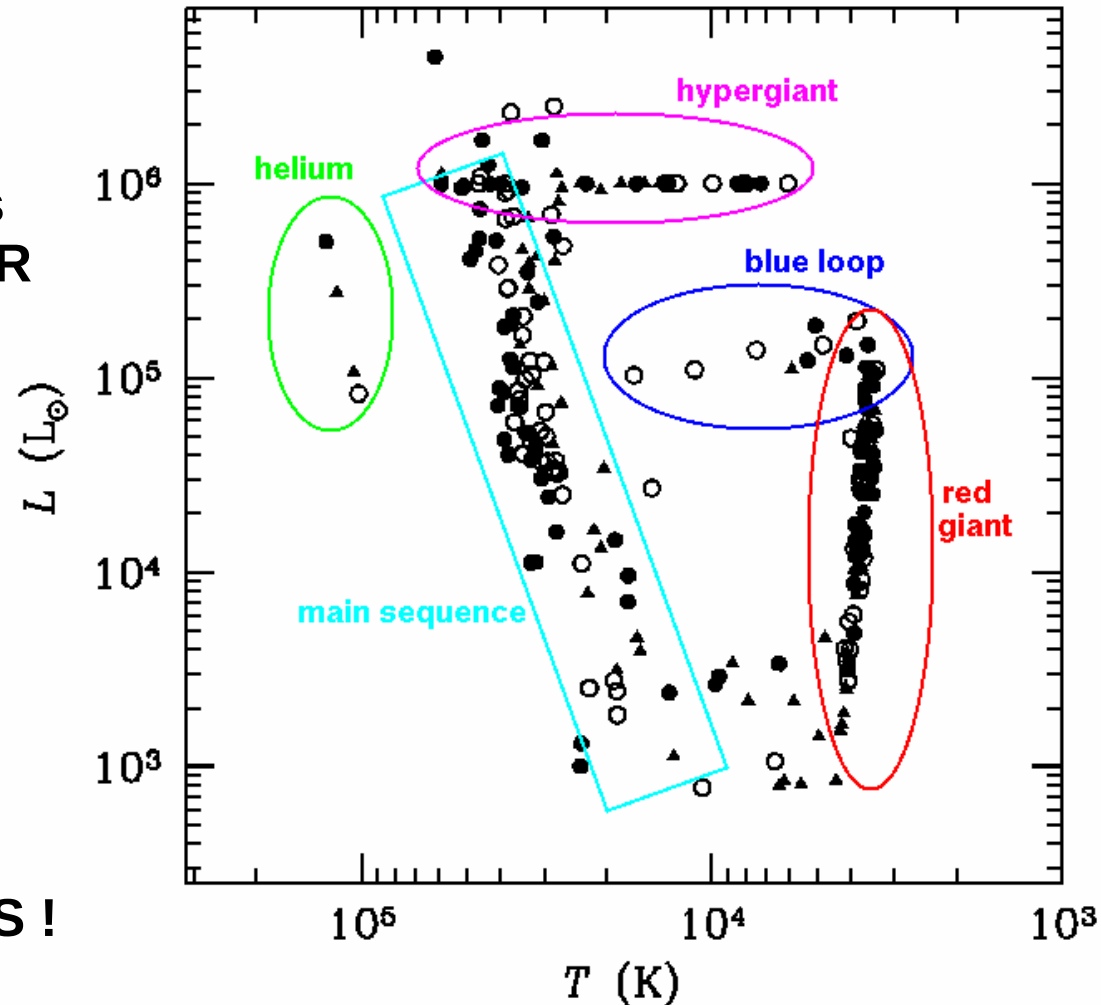
UNPHYSICAL RADIUS: softening,
to avoid spurious relax.

In simulations of collisional systems
(star clusters) each particle is a STAR

→ mass ~ 0.1 - $150 M_{\text{sun}}$
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 ($m > 25 M_{\text{sun}}$) 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 ($m < 2 M_{\text{sun}}$).
- * black hole (19) Star with radius smaller than the event horizon. The result of evolution of massive ($m > 25 M_{\text{sun}}$) 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 ($\sim 1e5$ 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%.

7. AN EXAMPLE of DIRECT N-BODY code: starlab

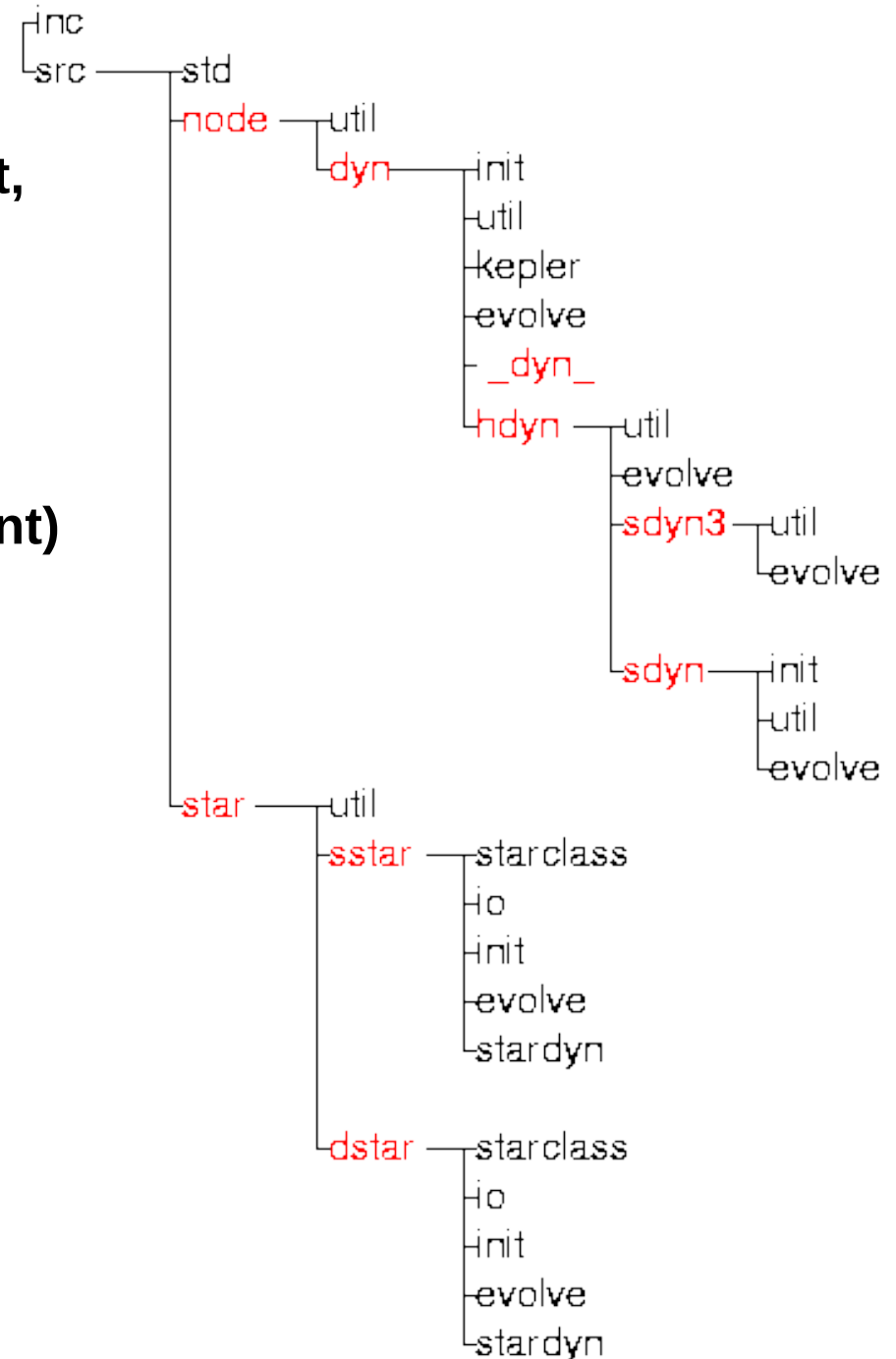
<http://www.sns.ias.edu/~starlab/overview/>

<http://www.sns.ias.edu/~starlab/structure/>

* not a code but a software environment,
a collection of modular software tools:
generate ICs (plummer, king),
dynamics, stellar evolution,
binary evolution,
plot tools (better not use),
analysis tools (statistics..some important)

*c++, something in fortran (DON'T USE)
→ CLASSES!!!

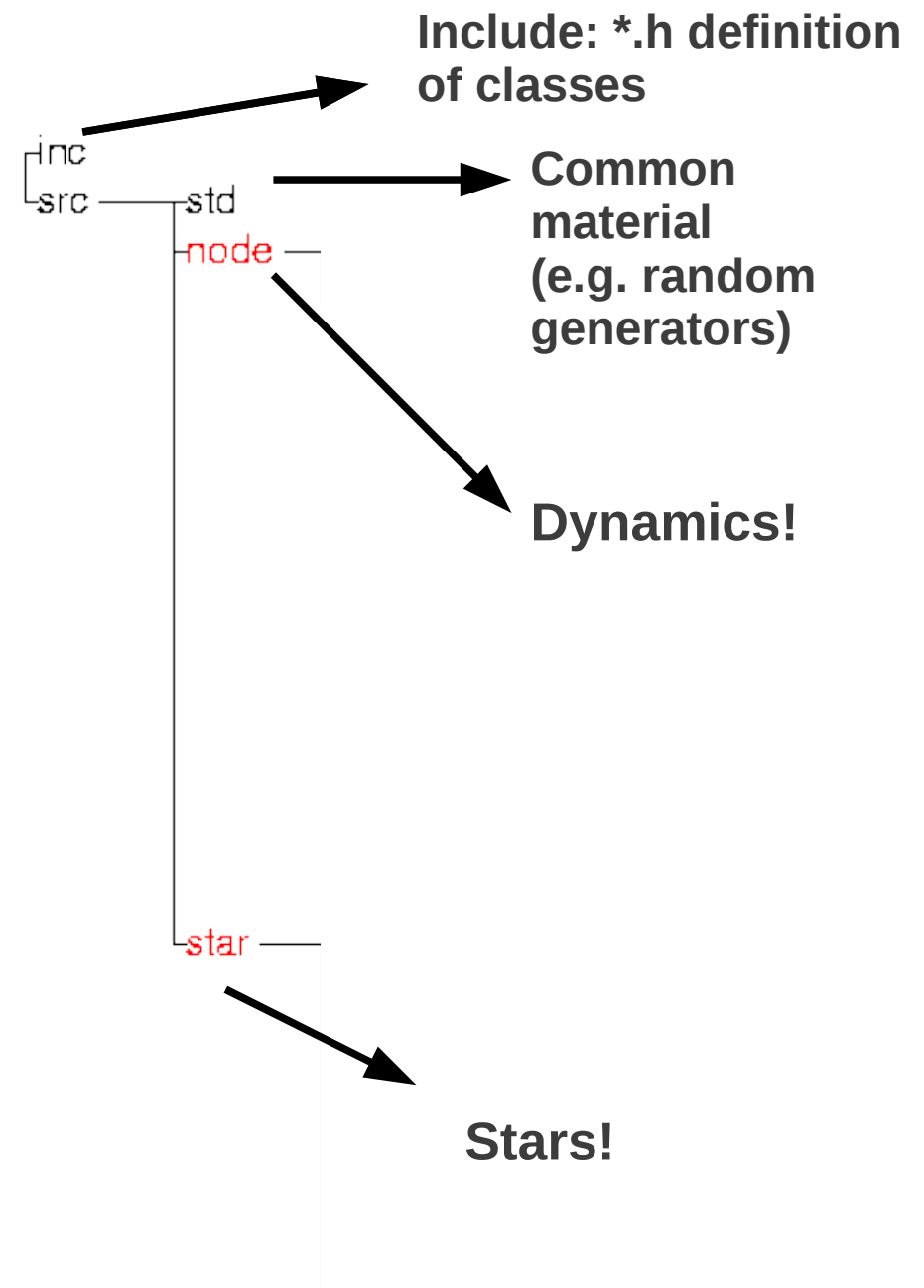
*complex, directory structure:



7. AN EXAMPLE: starlab

<http://www.sns.ias.edu/~starlab/overview/>
<http://www.sns.ias.edu/~starlab/structure/>

*complex, directory structure:



7. AN EXAMPLE: starlab

<http://www.sns.ias.edu/~starlab/overview/>

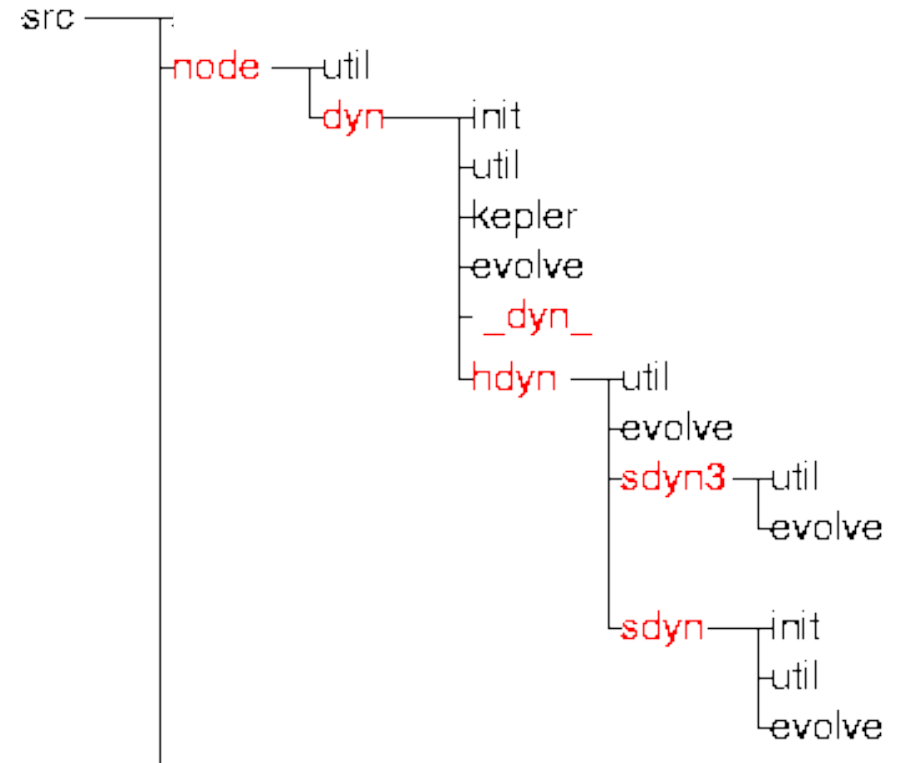
<http://www.sns.ias.edu/~starlab/structure/>

* dynamics:

init: contain tool for initialization

util: data analysis or plot

evolve: evolve dynamics in time



7. AN EXAMPLE: starlab

<http://www.sns.ias.edu/~starlab/overview/>
<http://www.sns.ias.edu/~starlab/structure/>

* dynamics:

init: contain tool for initialization
(src/node/dyn/init/makeking.C)

util: data analysis or plot

evolve: evolve dynamics in time

Kepler: only 2-body Keplerian

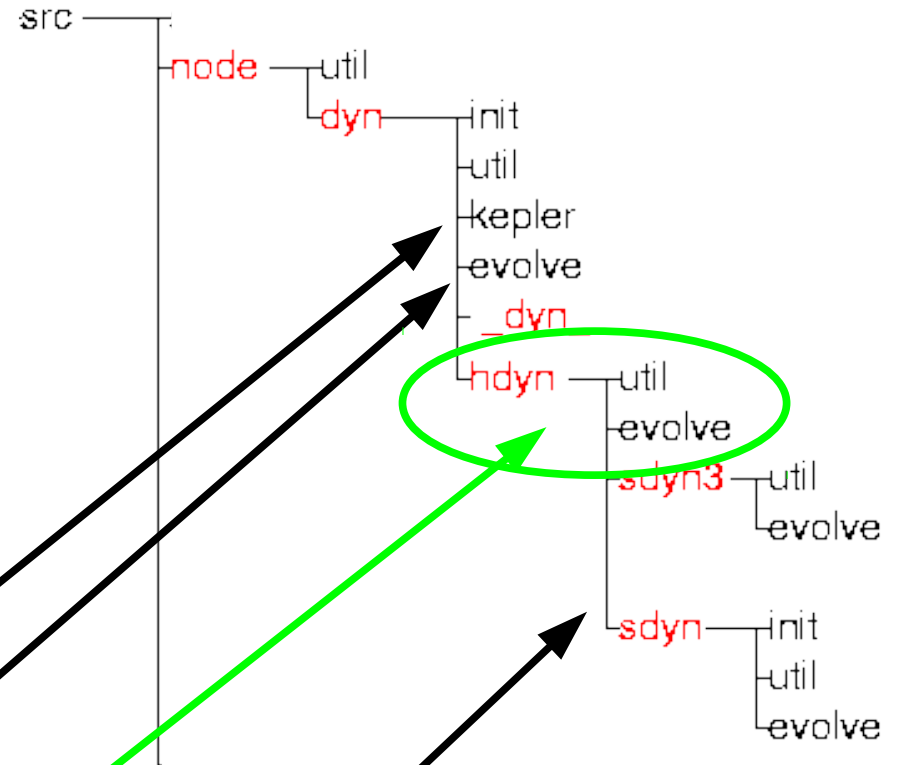
Only leapfrog

HDYN: high-res dynamics

KIRA INTEGRATOR

`./src/node/dyn/hdyn/evolve/kira.C`

only 3-body scattering



7. AN EXAMPLE: starlab

<http://www.sns.ias.edu/~starlab/overview/>

<http://www.sns.ias.edu/~starlab/structure/>

* stars:

init: contain tool for initialization

util: data analysis or plot

evolve: evolve in time star or binary

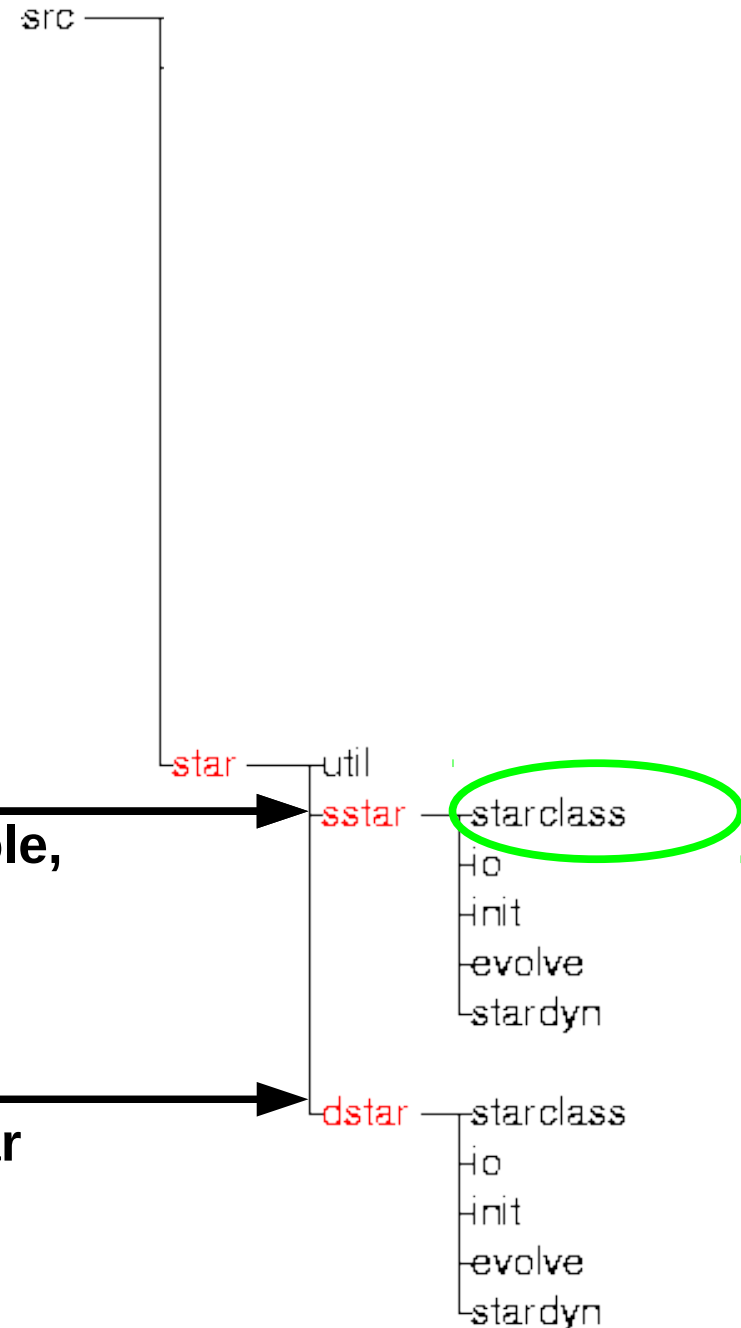
io: input output of star data

sstar: single stars

class: single star,
derived class: MS star, black hole,
hyper-giant, etcetc
In starclass/

dstar: double star

starclass: only class double star



7. AN EXAMPLE: starlab

Kira: the gravity integrator

<http://www.sns.ias.edu/~starlab/kira/>

based on 4th order Hermite with corrector/predictor

STEPS:

- 1. determines which stars need to be updated**
- 2. checks for: reinitialization, log output, escaper removal, termination, snapshot output**
- 3. perform low-order prediction (grape)**
- 4. calculates acceleration/jerk and correct position/velocities (grape)**
- 5. checks for all unperturbed motion**
- 6. checks for collisions and mergers**
- 7. checks tree reorganization**
- 8. checks for stellar/binary evolution**

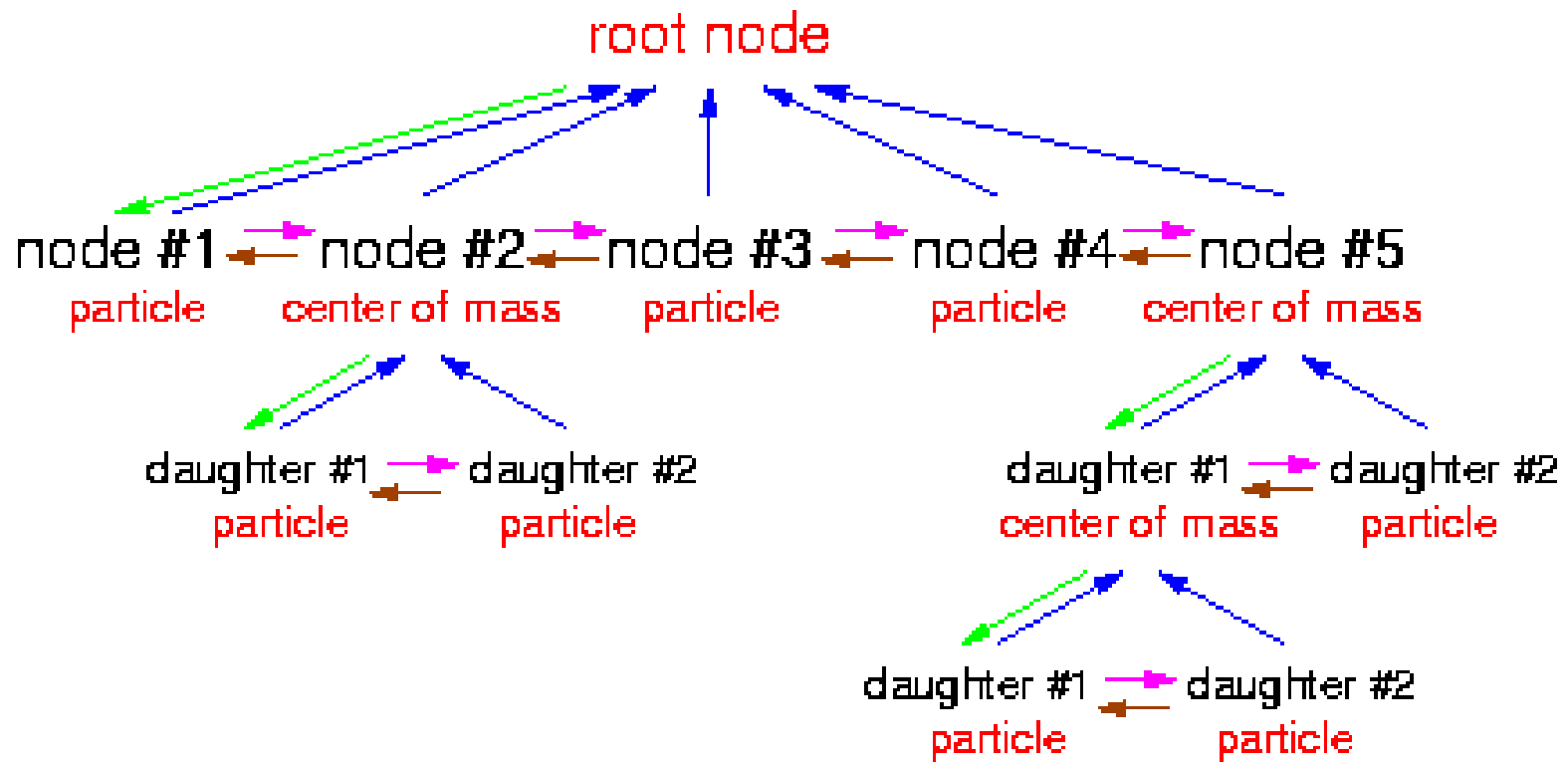
7. AN EXAMPLE: starlab

kira

<http://www.sns.ias.edu/~starlab/kira/>

based on 4th order Hermite with corrector/predictor

TREE simpler than tree code: leaves are single stars, parents can be binaries or multiples, no more



7. AN EXAMPLE: starlab

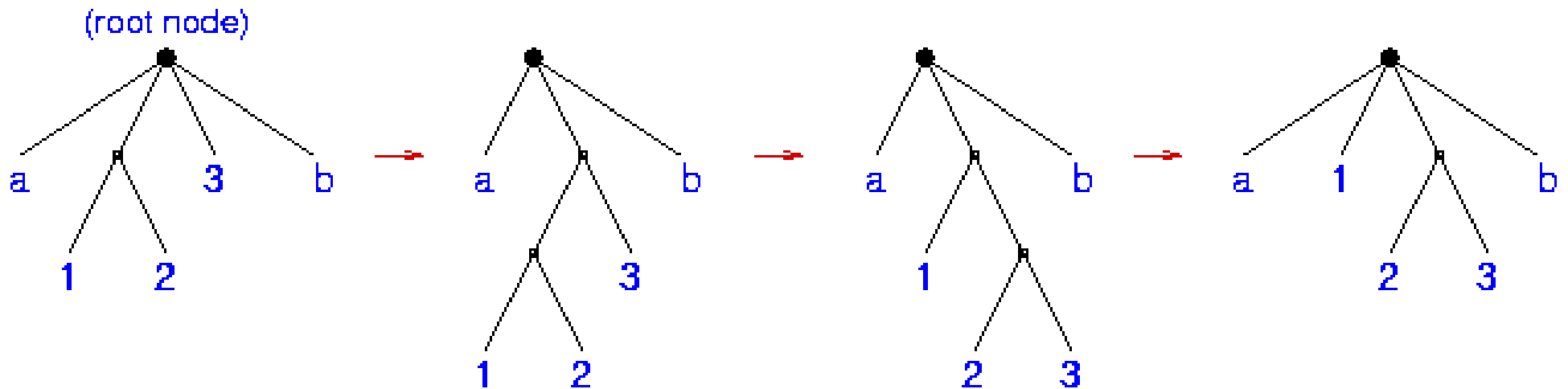
kira

<http://www.sns.ias.edu/~starlab/kira/>

based on 4th order Hermite with corrector/predictor

TREE simpler than tree code: leaves are single stars, parents can be binaries or multiples, no more (FLAT tree)

Example of a 3-body encounter



PERTURBED binaries (3-body) are split into components

UNPERTURBED binaries are evolved ANALYTICALLY

Critical point: how to decide perturber list!!!

7. AN EXAMPLE: starlab

No Aarseth chain and no KS regularization → thanks to the tree and to the continuous usage of CM/relative coordinates, 3-body encounters are integrated with accuracy (Portegies Zwart+ 2008)

Motion of a binary component described by (1) influence of companion, (2) influence of perturbers

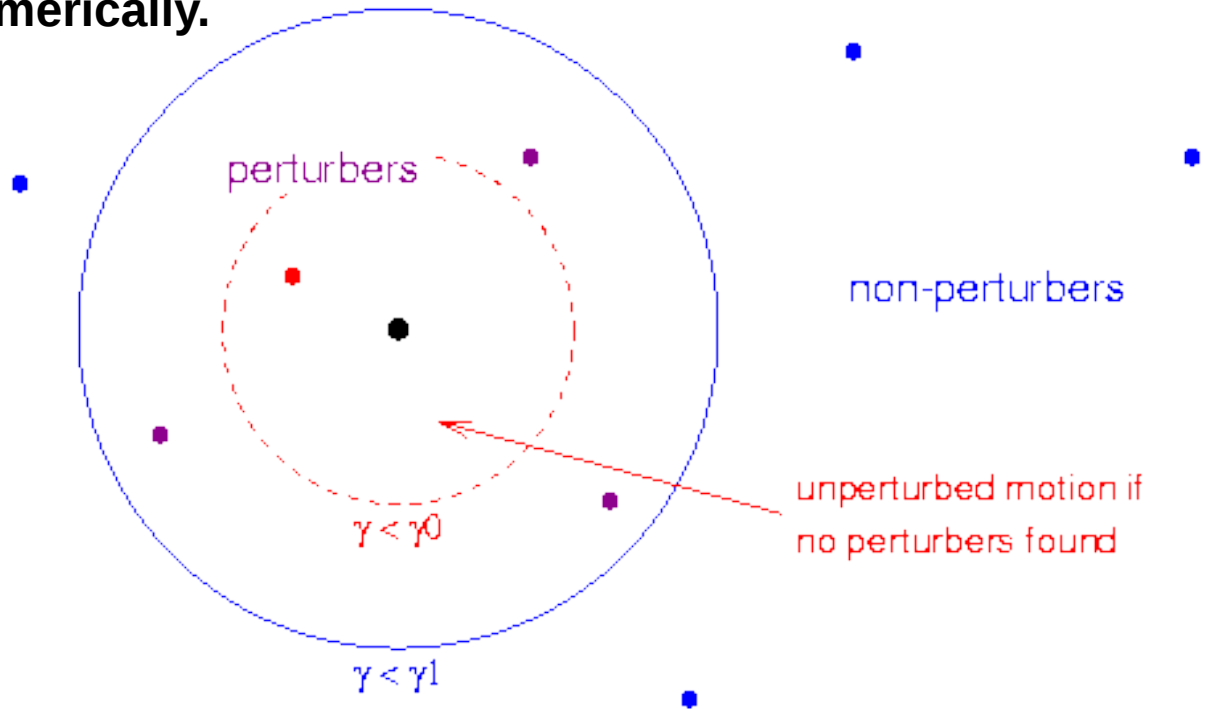
A perturber list is done and regularly updated for each binary

If perturbations < threshold → binary is assumed **UNPERTURBED** and **EVOLVED ANALYTICALLY (KEPLER MOTION)**
Only CM motion is integrated numerically.

PERTURBED binaries (3-body) are split into components

UNPERTURBED binaries are evolved ANALYTICALLY

Critical point: how to decide perturber list!!!



7. AN EXAMPLE: starlab

the stellar evolution: SEBA <http://www.sns.ias.edu/~starlab/seba/>

Portegies Zwart & Verbunt 1996

proto star (0) Non hydrogen burning stars on the Hyashi 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 ($m > 25M_{\text{sun}}$) 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 ($m < 2M_{\text{sun}}$).

black hole (19) Star with radius smaller than the event horizon. The result of evolution of massive ($m > 25M_{\text{sun}}$) star or collapsed neutron star.

disintegrated (20) Result of Carbon detonation to Type Ia supernova.

REFERENCES:

- **The Art of Computational Science, by P. Hut & J. Makino,**
<http://www.artcompsci.org/>
- direct N-body code description:
 - Starlab → Portegies Zwart et al. 2001, MNRAS, 321, 199
<http://www.sns.ias.edu/~starlab/>
 - PhiGRAPE → Harfst et al. 2007, New Astronomy, 12, 357
<http://www-astro.physik.tu-berlin.de/~harfst/index.php?id=phigrape>
 - N-body6 → Nitadori & Aarseth 2012, MNRAS, 424, 545
<http://www.ast.cam.ac.uk/~sverre/web/pages/nbody.htm>
 - HiGPUs → Capuzzo Dolcetta et al. 2013, Journal of Computational Physics, 236, 580
<http://astrowww.phys.uniroma1.it/dolcetta/HPCcodes/HiGPUs.html>
- GPU as hardware: <http://www.tomshardware.com/reviews/graphics-beginners,1288.html>
- GPU for computing:
 - Sapporo → Gaburov et al. 2009, New Astronomy, 14, 630
 - Nvidia Webinars → <https://developer.nvidia.com/get-started-cuda-cc>
- MPI in direct N-body codes:
 - Gualandris et al. 2007, Parallel Computing, 33, 159
 - Portegies Zwart et al. 2008, New Astronomy, 13, 285
- Stellar evolution:
 - Portegies Zwart & Verbunt 1996, A&A, 309, 179
 - Hurley et al. 2000, MNRAS, 315, 543
 - Mapelli et al. 2013, MNRAS, 429, 2298



THANKS

CUDA C



Standard C Code

```
void saxpy_serial(int n,
                 float a,
                 float *x,
                 float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}

// Perform SAXPY on 1M elements
saxpy_serial(4096*256, 2.0, x, y);
```

Parallel C Code

```
__global__
void saxpy_parallel(int n,
                   float a,
                   float *x,
                   float *y)
{
    int i = blockIdx.x*blockDim.x +
            threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}

// Perform SAXPY on 1M elements
saxpy_parallel<<<4096, 256>>>(n, 2.0, x, y);
```

SOFTENING:

numerical trick used in N-body techniques to prevent numerical divergences when a particle comes too close to another (and the force goes to infinity). This is obtained by modifying the gravitational potential of each particle as

$$\Phi = \frac{1}{\sqrt{r^2 + \epsilon^2}}$$

Dehnen & Read 2011, arXiv:1105.1082

2) definition and structure of starlab

<http://www.sns.ias.edu/~starlab/structure/>

- **CLASS**: description of structure+ its functions
- an **OBJECT** belongs to a class if it is **DEFINED** as member of the class → `star a;`

EACH PARTICLE + root belongs to the **node class (include/node.h)**

```
class node {
    static node* root; // Global address of the root node.
    long int node_flag; // Indicator of valid node (for internal
                        // bookkeeping purposes only)
    int index; // Nodes can be numbered,
    char * name; // or they can receive individual names.
    real mass;
    node * oldest_daughter; // Define the node's place in
    node * elder_sister; // the tree.
    node * younger_sister;
    story * log_story; // Log story is a generalized scratchpad.
    story * dyn_story; // The dyn story is a placeholder for
                        // dynamical information not recognized by
                        // a program -- this allows the information
                        // to be preserved and passed down a pipe.
    hydrobase * hbase; // hydrobase is the class underlying all
                        // classes that handle hydrodynamics.
    starbase * sbase; // starbase is the class underlying all
                       // classes that handle stellar evolution.
}
```


2) definition and structure of starlab

<http://www.sns.ias.edu/~starlab/structure/>

EACH PARTICLE + root belongs to the **node** class

If dynamics is defined, the **dyn** class is derived from node (include/dyn.h)

HEREDITARIETY

```
class dyn : public node {
    static real system_time;
    static bool use_sstar; // Single star evolution if true.
    vector pos;           // Position (3-D Cartesian vector).
    vector vel;           // Velocity: (d/dt) pos.
    vector acc;           // Acceleration: (d/dt) vel.
    kepler * kep;         // Pointer to a kepler orbit object.
}
```

NB: mass belongs to node, pos, vel, acc only to dyn

2) definition and structure of starlab

<http://www.sns.ias.edu/~starlab/structure/>

EACH PARTICLE + root belongs to the **node** class

If dynamics is defined, the **dyn** class is derived from node

If high-res **hdyn** class is derived from **_dyn_** which is derived from **dyn**
(include/hdyn.h, include/_dyn_.h)

```
class _dyn_ : public dyn {
    real time;           // Individual particle time
    real timestep;      // and time step.
    real pot;           // Potential.
    vector jerk;        // (d/dt) acc
    vector pred_pos;    // Predicted variables for use in the
    vector pred_vel;    // standard predictor-corrector scheme.
    real t_pred;        // Time of prediction.
    real radius;        // Effective (or actual) radius.
}
```

2) hdyn

Tidal field

Binary evolution

Time dynamical
Integration
(e.g. softening)

Removal
of escapers

Infos on
perturbers
(see kira)

```
class hdyn :public _dyn_ {
//-----
// Global variables:
// Tidal field:
static int tidal_type; // none, point-mass, halo, disk
static real alpha1; // tidal field is conventionally taken
static real alpha3; // to be (-alpha1*x, 0, -alpha3*z)
static real omega; // system angular speed
// Binary evolution:
static bool use_dstar; // binary evolution if true
// Stellar encounters and mergers:
static real stellar_encounter_criterion_sq;
static real stellar_merger_criterion_sq;
static real stellar_capture_criterion_sq;
// Run-time integration parameters:
static real eta; // time step parameter
static real eps; // softening length
static real d_min_sq; // scale term governing tree adjustment
static real lag_factor; // squared hysteresis factor
static real mbar; // mass scale
static real gamma2; // squared threshold for unperturbed motion
static real gamma23; // gamma^{-2/3}
static real initial_step_limit; // limit on first time step
static real step_limit; // limit on all time steps
// Escaper removal:
static real scaled_stripping_radius; // stripping radius for unit mass
//-----
// Variables for unperturbed motion:
real perturbation_squared; // Relative perturbation squared.
real unperturbed_timestep; // Time step for unpert. motion.
bool fully_unperturbed; // True if orbit is fully
// unperturbed.
// Perturber information:
int n_perturbers; // Number of perturbers.
hdyn** perturber_list; // Pointer to perturber array.
bool valid_perturbers; // True if any particle is
// within the perturbation
// radius and the perturber
// list has not overflowed.
// Other neighbor information:
hdyn* nn; // Pointer to nearest neighbor.
real d_nn_sq; // Distance squared to nn.
hdyn* coll; // Pointer to neighbor whose
// surface is closest to this node.
real d_coll_sq; // Distance squared to coll.
// HARP-3 variables:
int harp3_index; // HARP-3 address of this particle.
real harp3_rnb_sq; // HARP-3 neighbor sphere radius.
}
```

2) definition and structure of starlab

<http://www.sns.ias.edu/~starlab/structure/>

Basic class for stars is **starbase** (include/starbase.h, for root):

```
class starbase {
    node * the_node;           // pointer to associated node
    story * star_story;       // pointer to star story

    static real m_conv_star_to_dyn; // mass conversion factor
    static real r_conv_star_to_dyn; // length conversion factor
    static real t_conv_star_to_dyn; // time conversion factor
    static bool use_hdyn;        // true iff binary evolution
                                // is enabled

    /*mmapelli add on December 30 2012*/
    static real starmetal; /* default is solar metallicity*/
    /*mmapelli add on December 30 2012*/
}
```

2) definition and structure of starlab

<http://www.sns.ias.edu/~starlab/structure/>

Basic class for stars is **starbase** (for root)

For each particle, **star** class is derived from starbase (include/star/star.h)

```
class star : public starbase {
    // No private or
protected data...
    public:
        .
        .
        .
}
```

2) definition and structure of starlab

<http://www.sns.ias.edu/~starlab/structure/>

Basic class for stars is **starbase** (for root)

For each particle, **star** class is derived from starbase

If star evolution, **single_star** class is derived from star (include/star/single_star.h)

```
class single_star : public star {
    int identity;
    stellar_type star_type;
    // main sequence,

    // red giant, etc.
    stellar_type_spec spec_type[no_of_spec_type];
    // spectral type
    real current_time;
    real relative_age;
    real last_update_age;
    real next_update_age;
    real relative_mass;
    real envelope_mass;
    real core_mass;
    real radius;
    real core_radius;
    real effective_radius;
    real luminosity;
}
```

2) definition and structure of starlab

<http://www.sns.ias.edu/~starlab/structure/>

Basic class for stars is **starbase** (for root)

For each particle, **star** class is derived from starbase

If star evolution, **single_star** class is derived from star

Each stellar type derives from single_star

E.g. **main_sequence** (in include/star/main_sequence.h)

```
class main_sequence : public
single_star {

        real main_sequence_core_mass();
        real
main_sequence_core_radius();
        void adjust_donor_age(const
real mdot);
    }
```

2) definition and structure of starlab

<http://www.sns.ias.edu/~starlab/structure/>

Basic class for stars is **starbase** (for root)

For each particle, **star** class is derived from starbase

If star evolution, **single_star** class is derived from star

If binary evolution, **double_star** class is derived from star
(include/star/double_star.h)

```
class double_star : public star {
    real semi;
    real eccentricity;
    binary_type bin_type;
    int identity;
    real binary_age;
    real minimal_timestep;
    int donor_identity;
    stellar_type donor_type;
    real donor_timescale;
    mass_transfer_type
current_mass_transfer_type;
}
```

NB: single_star is associated with leaves, double_star with parent (kira!)

3) the outputs

<http://www.sns.ias.edu/~starlab/internals/>
comes naturally from the class structure

PARTICLE:
each single
node

```
(Particle
  i = 4
  N = 1
  (Log
    Close encounter with black hole #7 at time 10 Myr
  )Log
  (Dynamics
    m = 0.5
    r = -0.1  0.2  0.5
    v = 0.3  -0.4  -0.3
  )Dynamics
  (Hydro
  )Hydro
  (Star
    Type = main_sequence
    T_cur = 0
    M_rel = 1
    M_env = 0.99
    M_core = 0.01
    T_eff = 6000
    L_eff = 1
  )Star
)Particle
```

LOG: log
story of the
node

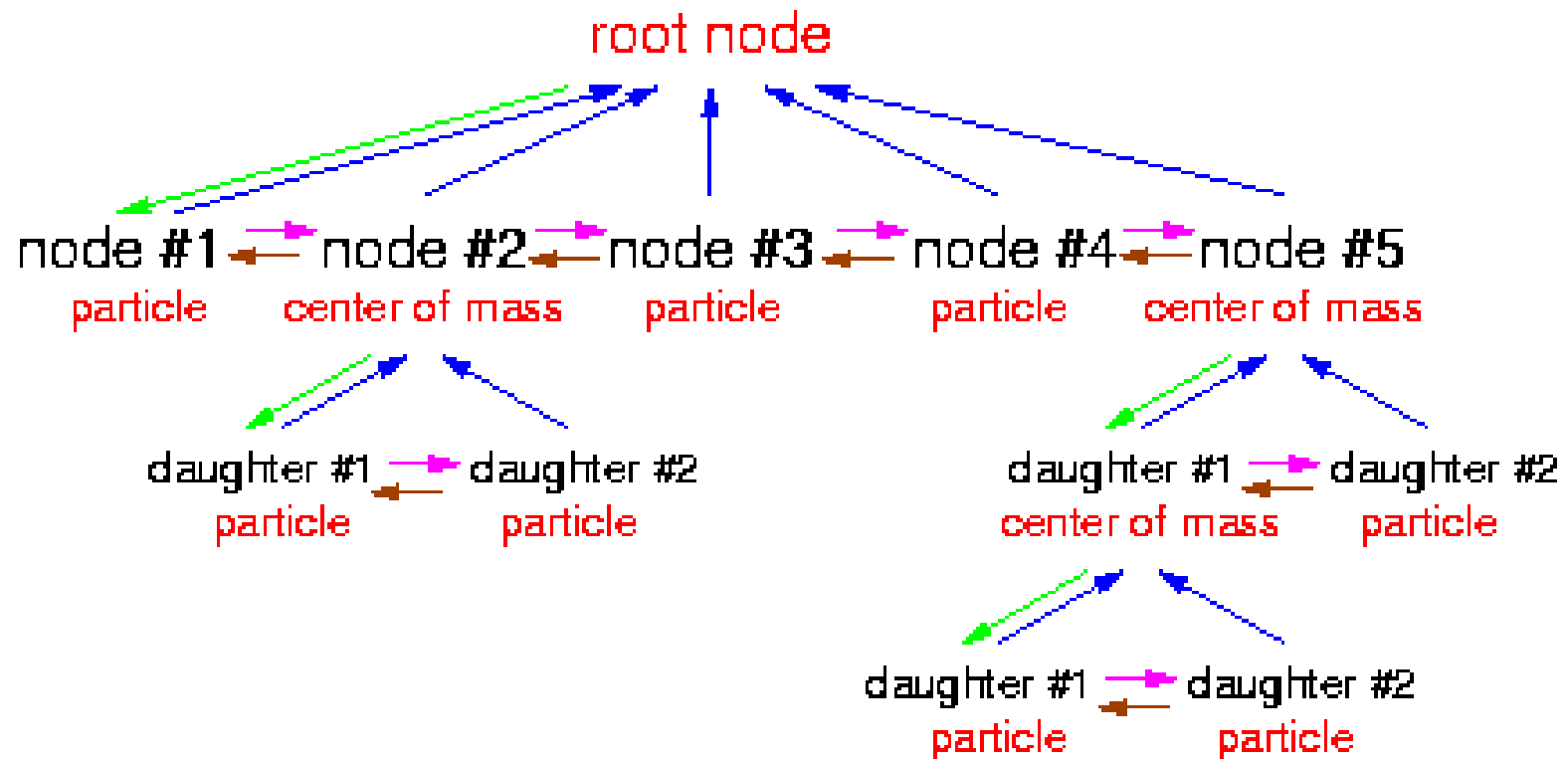
DYN story of
the node

Hydro story
of the node

STAR class
story

3) the outputs

<http://www.sns.ias.edu/~starlab/internals/>
comes naturally from the class structure



3) the outputs

<http://www.sns.ias.edu/~starlab/internals/>
comes naturally from the class structure

PARTICLE: can be single (N = 1), or a binary (N=2) with 2 daughter particles, or a root (name=root), or more complicated dependence

```
(Particle <-----  
  N = 1  
(Log  
)Log  
(Dynamics  
  m = 1  
)Dynamics  
)Particle <-----  
(Particle <-----  
  N = 1  
(Log  
)Log  
(Dynamics  
  m = 1  
)Dynamics  
)Particle <-----  
(Particle <-----  
  N = 1  
(Log  
)Log  
(Dynamics  
  m = 1  
)Dynamics  
)Particle <-----
```

```
(Particle <-----  
  N = 2  
(Log  
)Log  
(Dynamics  
  m = 1  
)Dynamics  
(Particle <-----  
  N = 1  
(Log  
)Log  
(Dynamics  
  m = 0.5  
)Dynamics  
)Particle <-----  
(Particle <-----  
  N = 1  
(Log  
)Log  
(Dynamics  
  m = 0.5  
)Dynamics  
)Particle <-----  
)Particle <-----
```

4) kira

<http://www.sns.ias.edu/~starlab/kira/>

based on 4th order Hermite with corrector/predictor

STEPS:

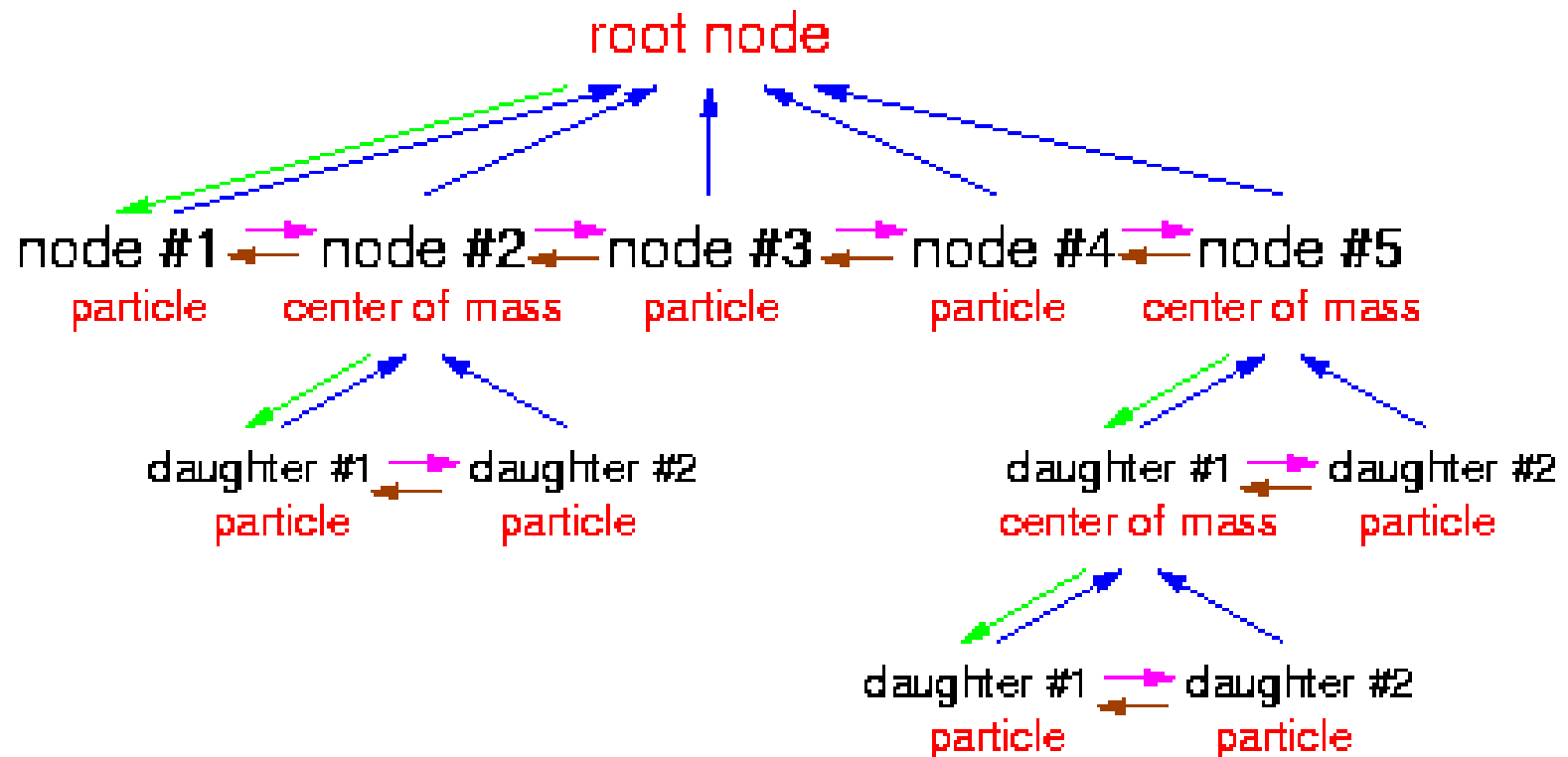
- 1. determines which stars need to be updated**
- 2. checks for: reinitialization, log output, escaper removal, termination, snapshot output**
- 3. perform low-order prediction (grape)**
- 4. calculates acceleration/jerk and correct position/velocities (grape)**
- 5. checks for all unperturbed motion**
- 6. checks for collisions and mergers**
- 7. checks tree reorganization**
- 8. checks for stellar/binary evolution**

4) kira

<http://www.sns.ias.edu/~starlab/kira/>

based on 4th order Hermite with corrector/predictor

TREE simpler than tree code: leaves are single stars, parents can be binaries or multiples, no more



Forces are computed using direct summation over all other particles in the system; no tree or neighbor-list constructs are used!!!

NO $O(N \log N)$

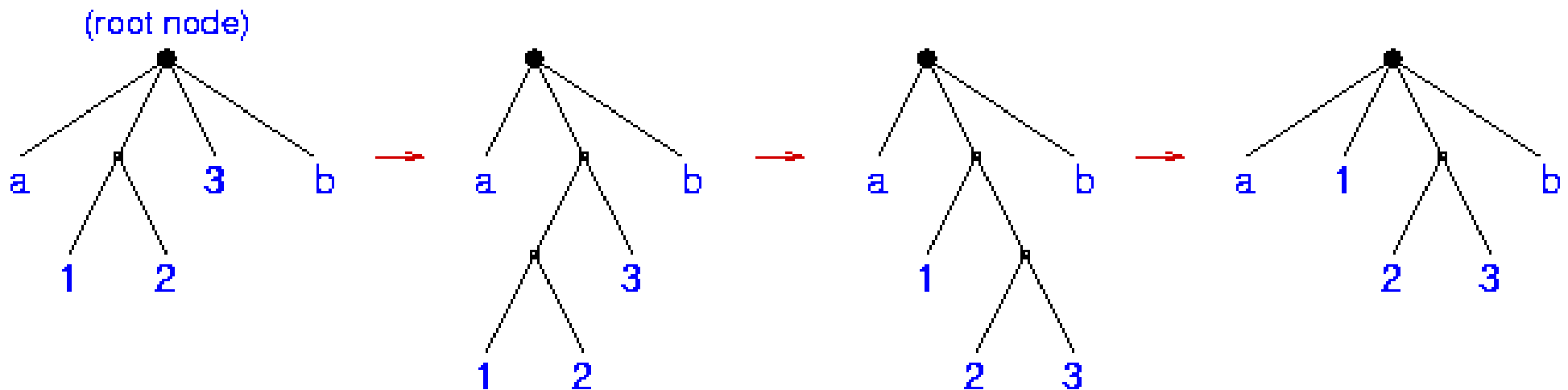
4) kira

<http://www.sns.ias.edu/~starlab/kira/>

based on 4th order Hermite with corrector/predictor

TREE simpler than tree code: leaves are single stars, parents can be binaries or multiples, no more

Example of a 3-body encounter



PERTURBED binaries (3-body) are splitted into components

UNPERTURBED binaries are evolved ANALYTICALLY

Critical point: how to decide perturber list!!!

4) kira

<http://www.sns.ias.edu/~starlab/kira/>

based on 4th order Hermite with corrector/predictor

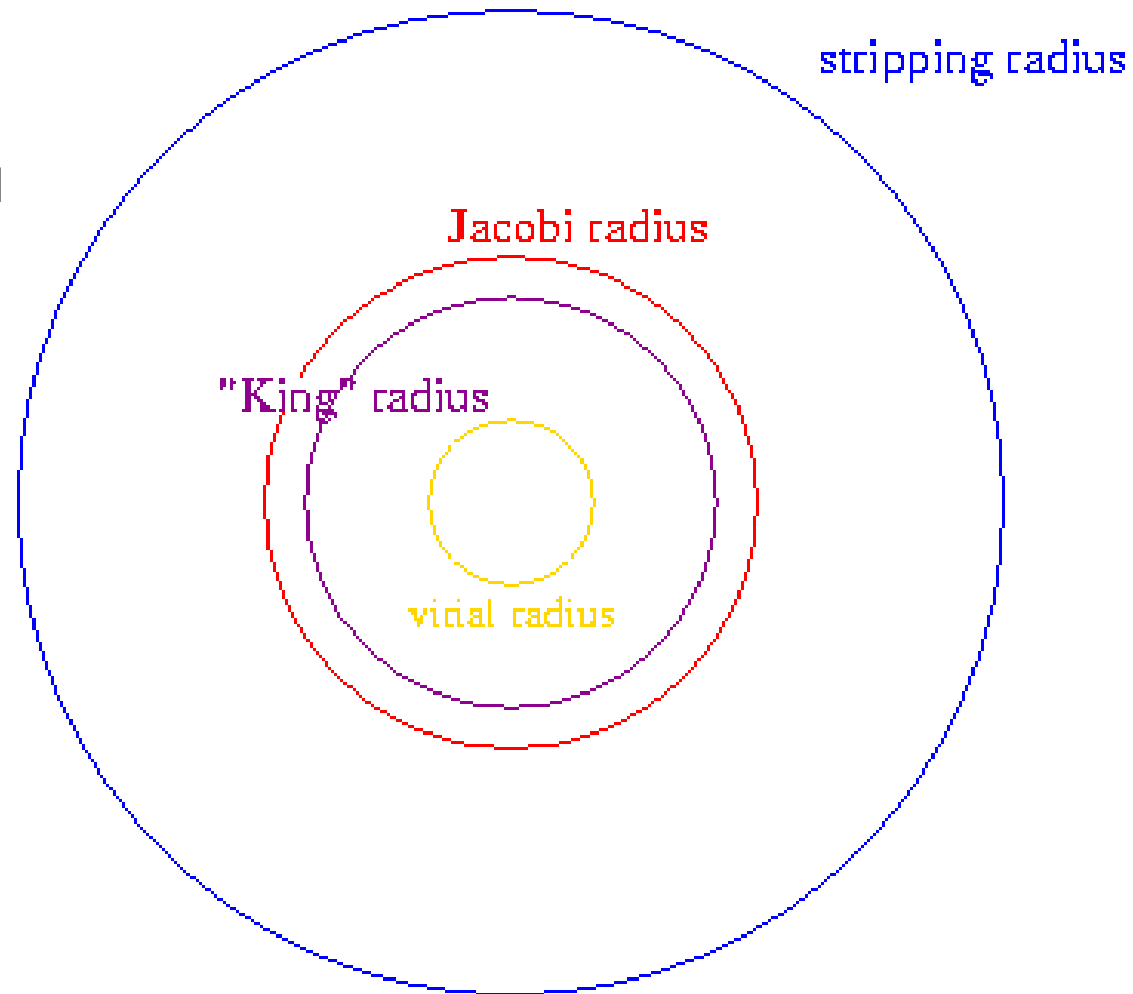
ESCAPER REMOVAL (not in current simulations)

* Virial radius

* King radius:
cutoff of King model

* Jacobi radius:
tidal radius

*stripping radius:
radius for escaper
removal
(eg 2 Jacobi)



4) kira

<http://www.sns.ias.edu/~starlab/kira/>

RUNNING KIRA

```
./kira -t 150 -d 1 -D 1 -b 1 -f 0 -n 10 -e 0.000 -B -s 31107
```

-t number of timesteps

-d log output interval

-D snapshot interval

-b specify frequency of full binary output

-f add analytic formula for internal dynamical friction (should already be accounted for by integrator). 0 means no friction, 1 means friction. It works only with Plummer & Power-law.

If you use King, you can avoid specifying -f.

WARNING: in old versions of kira -f indicates the minimum energy to form a binary (-f 0.3 means that only binaries with $|E| > 0.3 kT$ can form).

-n minimum number of particles (below n terminate simulation, *i.e. if $>(N-n)$ stars escape and are removed, terminate the simulation*)

-e softening

-B with binary evolution (and also star, otherwise -S)

-s random seed (default internal clock)

5) the stellar evolution: SEBA

<http://www.sns.ias.edu/~starlab/seba/>

Portegies Zwart & Verbunt 1996

proto star (0) Non hydrogen burning stars on the Hyashi 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 ($m > 25 M_{\text{sun}}$) 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 ($m < 2 M_{\text{sun}}$).

black hole (19) Star with radius smaller than the event horizon. The result of evolution of massive ($m > 25 M_{\text{sun}}$) star or collapsed neutron star.

disintegrated (20) Result of Carbon detonation to Type Ia supernova.

5) the stellar evolution: SEBA

<http://www.sns.ias.edu/~starlab/seba/>

Portegies Zwart & Verbunt 1996

WHAT I CHANGED:

- include/starbase.h → add starmetal
- src/star/sstar/starclass/hertzsprung_gap.C → remove spurious wind (8-20Msun)
- star/sstar/starclass/main_sequence.C → Hurley+ 2000 metal dependent radii
Vink+2001 winds for MS
- star/sstar/starclass/horizontal_branch.C → remove spurious wind (8-20Msun)
- star/sstar/starclass/single_star.C → winds for MS, WR and LBV
- star/sstar/starclass/sub_giant.C → remove spurious wind (8-20Msun)
- star/sstar/starclass/helium_giant.C → remove Disintegrated stars
change winds to adapt to WR
- star/sstar/starclass/black_hole.C → insert direct collapse (failed supernova)

SEE YOURSELF with `grep -R mmapelli *`

6) compilation & installation

```
tar xvfz starlabapr19_2013.tgz
```

```
cd starlabapr19_2013/
```

```
make clean
```

```
./configure
```

```
make
```

```
make install
```



Copy executables on /usr/bin

6) compilation & installation, ADVANCED

- * if you have grape or GPU+CUDA, put **grape.sh** (optimized for grape or GPU) in **local/** → configure will find it and configure starlab for grape or GPU
otherwise configure will optimize for serial CPU
- * if you have the file local/grape.sh but you DO NOT WANT TO COMPILE FOR GRAPE or GPU, change name to grape.sh or configure with

./configure --without-grape

- * the only important for us is GPU+CUDA → NEEDS:
 1. configure for GPU (no grape) → my file
 2. NVIDIA GPU
 3. CUDA (<https://developer.nvidia.com/cuda-downloads>)
 4. SAPPORO LIBRARY (Gaburov et al. 2009)
(<http://castle.strw.leidenuniv.nl/software/sapporo.html>)

- * if you have fortran, please make configure not to use it:

./configure --without-f77

6) compilation & installation, ADVANCED

* on PLX @ cineca use setup_starlab_mm2.sh

```
#!/bin/bash
#PBS -N test1
#PBS -A PROJECTNAME
#PBS -q debug
#PBS -l walltime=0:20:00
#PBS -l select=1:ncpus=1:ngpus=2

module load gnu/4.1.2
module load profile/advanced
module load boost/1.41.0--intel--11.1--binary
#module load boost/1.41.0--gnu--4.1.2
module load cuda/4.0

LD_LIBRARY_PATH=/cineca/prod/compilers/cuda/4.0/none/lib64:/cineca/prod/compilers/cuda/4.0/none/lib:/cineca/prod/lib
raries/boost/1.41.0/intel--11.1--binary/lib:/cineca/prod/compilers/intel/11.1/binary/lib/intel64

export LD_LIBRARY_PATH

cd /plx/userexternal/mmapelli/starlabapr19_2013/
make clean
./configure --without-f77
make
make install
```

* NB to run setup_starlab_mm2.sh you need to be on the computing nodes:

qsub setup_starlab_mm2.sh

PROJECTNAME found with saldo -b

6) compilation & installation, ADVANCED

* on PLX @ cineca you can also compile interactively (e.g. if you debug):

*Submit an interactive job as **qsub -l***

qsub -l walltime=0:10:00 -l select=1:ncpus=1 -q debug -A projectname

and then type in the shell

```
module load gnu/WHICH_VERSION
module load profile/advanced
module load boost/WHICH_VERSION
module load cuda/WHICH_VERSION
cd /plx/userexternal/USER/starlabYOUR_VERSION/
make clean
./configure --without-f77
make
make install
```

7) writing initial conditions

<http://www.sns.ias.edu/~starlab/examples/>

Use a sh script (easier and keep memory)

```
./makeking -n 5000 -w 5 -i -u \  
| ./makemass -f 8 -l 0.1 -u 150 \  
| ./makesecondary -f 0.1 -q -l 0.1 \  
| ./add_star -R 1 -Z 0.01 \  
| ./scale -R 1 -M 1\  
| ./makebinary -f 2 -o 1 -l 1 -u 107836.09 \  
> cineca110_bin_N5000_frac01_W5_Z001_IC.txt
```

* makeking: generates a king profile with
-n number of centres of mass
-w dimensionless central potential
-i number the particles sequentially
-u leave final N-body system unscaled
src/node/dyn/init/makeking.C

Useful alternative: makeplummer (**src/node/dyn/init/makeplummer.C**)

7) writing initial conditions

<http://www.sns.ias.edu/~starlab/examples/>

Use a sh script (easier and keep memory)

```
./makeking -n 5000 -w 5 -i -u \  
| ./makemass -f 8 -l 0.1 -u 150 \  
| ./makesecondary -f 0.1 -q -l 0.1 \  
| ./add_star -R 1 -Z 0.01 \  
| ./scale -R 1 -M 1\  
| ./makebinary -f 2 -o 1 -l 1 -u 107836.09 \  
> cineca110_bin_N5000_frac01_W5_Z001_IC.txt
```

- * makemass: generates mass of primary & single stars from IMF
 - f 1-8: kind of IMF (1 Power-law, 2 Miller & Scalo, 3 Scalo, 4 old Kroupa, 5 DeMarchi, 6 old Kroupa+ 1991, 7 two power law, 8 Kroupa 2001)
 - l minimum star mass (units of Msun)
 - u maximum star mass (units of Msun)

src/node/util/makemass.C

7) writing initial conditions

<http://www.sns.ias.edu/~starlab/examples/>

Use a sh script (easier and keep memory)

```
./makeking -n 5000 -w 5 -i -u \  
| ./makemass -f 8 -l 0.1 -u 150 \  
| ./makesecondary -f 0.1 -q -l 0.1 \  
| ./add_star -R 1 -Z 0.01 \  
| ./scale -R 1 -M 1\  
| ./makebinary -f 2 -o 1 -l 1 -u 107836.09 \  
> cineca110_bin_N5000_frac01_W5_Z001_IC.txt
```

* makesecondary: generates mass of secondary from flat distribution

-f binary fraction

-q if present, secondary mass ratio is chosen uniformly
on [lower_limit, upper_limit]

-l lower limit secondary mass (if -q in fraction of primary mass)

-u upper limit secondary mass (if -q in fraction of primary mass)

If not specified =1

src/node/util/makesecondary.C

7) writing initial conditions

<http://www.sns.ias.edu/~starlab/examples/>

Use a sh script (easier and keep memory)

```
./makeking -n 5000 -w 5 -i -u \  
  | ./makemass -f 8 -l 0.1 -u 150 \  
  | ./makesecondary -f 0.1 -q -l 0.1 \  
  | ./add_star -R 1 -Z 0.01 \  
  | ./scale -R 1 -M 1\  
  | ./makebinary -f 2 -o 1 -l 1 -u 107836.09 \  
  > cineca110_bin_N5000_frac01_W5_Z001_IC.txt
```

* `add_star`: generates physical properties of stars (radius)

-M `m-scale` - mass scale for stars. If not set uses `Mtot` → better!

-R `l-scale` - dynamical size scaling (in parsecs)

Error if you do not put anything. May be virial radius of cluster or other scale. Suggestion: put 1 (1 parsec=44370956 sun radii), otherwise you lose control on units.

-Z star cluster metallicity (in units of solar=0.019)

added by MMapelli on December 31 2012

`src/star/sstar/init/add_star.C`

7) writing initial conditions

<http://www.sns.ias.edu/~starlab/examples/>

Use a sh script (easier and keep memory)

```
./makeking -n 5000 -w 5 -i -u \  
| ./makemass -f 8 -l 0.1 -u 150 \  
| ./makesecondary -f 0.1 -q -l 0.1 \  
| ./add_star -R 1 -Z 0.01 \  
| ./scale -R 1 -M 1\  
| ./makebinary -f 2 -o 1 -l 1 -u 107836.09 \  
> cineca110_bin_N5000_frac01_W5_Z001_IC.txt
```

* add_star: produces in output

```
(Star  
mass_scale    = 0.000299852183843106945  
size_scale    = 2.25500000000000000001e-08  
time_scale    = 3.88903717906355428  
metallicity   = 1  
)Star
```

1/Mtot in Msun
1/Rsun in pc NB!
BUG!!!
1/tscale in Myr
in Zsun

7) writing initial conditions

<http://www.sns.ias.edu/~starlab/examples/>

Use a sh script (easier and keep memory)

```
./makeking -n 5000 -w 5 -i -u \  
| ./makemass -f 8 -l 0.1 -u 150 \  
| ./makesecondary -f 0.1 -q -l 0.1 \  
| ./add_star -R 1 -Z 0.01 \  
| ./scale -R 1 -M 1\  
| ./makebinary -f 2 -o 1 -l 1 -u 107836.09 \  
> cineca110_bin_N5000_frac01_W5_Z001_IC.txt
```

* kira + add_star: produces in stderr

scale factors taken from input snapshot

[m]: 3335.0 M_sun

[R]: 1 pc

[T]: 0.257133 Myr

Mscale = Mtot/Msun

lscale in pc

t scale in Myr

7) writing initial conditions

<http://www.sns.ias.edu/~starlab/examples/>

Use a sh script (easier and keep memory)

```
./makeking -n 5000 -w 5 -i -u \  
| ./makemass -f 8 -l 0.1 -u 150 \  
| ./makesecondary -f 0.1 -q -l 0.1 \  
| ./add_star -R 1 -Z 0.01 \  
| ./scale -R 1 -M 1\  
| ./makebinary -f 2 -o 1 -l 1 -u 107836.09 \  
> cineca110_bin_N5000_frac01_W5_Z001_IC.txt
```

* scale: generates physical scales for final SC

-R specify virial radius

in parsecs, if add_star -R 1 →

(1) ./add_star -R 1 | ./scale -R 5 means $r_{\text{vir}}=5$ in units of 1 pc → $r_{\text{vir}}=5$ pc!!

in units of add_star, if add_star -R != 1 →

(2) ./add_star -R 5 | ./scale -R 1 means $r_{\text{vir}}=1$ in units of 5 pc → $r_{\text{vir}}=5$ pc!!

Almost equivalent, (1) easier, (2) gives more physical meaning to timescale

-M specify star cluster mass

in units of M_{tot} , if add_star has no -M option →

-M 1 means that mass units in the output file are $/M_{\text{tot}}$

IMPORTANT THAT SCALE BE AFTER ADD_STAR IF STAR EVOL
src/node/dyn/util/scale.C

7) writing initial conditions

<http://www.sns.ias.edu/~starlab/examples/>

Use a sh script (easier and keep memory)

```
./makeking -n 5000 -w 5 -i -u \  
| ./makemass -f 8 -l 0.1 -u 150 \  
| ./makesecondary -f 0.1 -q -l 0.1 \  
| ./add_star -R 1 -Z 0.01 \  
| ./scale -R 1 -M 1\  
| ./makebinary -f 2 -o 1 -l 1 -u 107836.09 \  
> cineca110_bin_N5000_frac01_W5_Z001_IC.txt
```

* makebinary: generates orbital properties of primordial binaries

-f function select option

1: angular momentum per unit reduced mass

($L^2 = am[1-e^2]$), solar units

2: semi-major axis or peri/apo, solar units

3: energy

-o specify interpretation of limits - With -f 2 -o 1: semi-major axis,

-l lower limit on selected binary parameter (sma in R_{sun})

-u upper limit on selected binary parameter (sma in R_{sun})

src/node/dyn/init/makebinary.C

7) writing initial conditions

<http://www.sns.ias.edu/~starlab/examples/>

Last note on units

-units of stdout:

In (Dynamics ..)Dynamics units scaled to Mscale, Iscale, tscale (note *)

In (Star ..)Star units scaled to Msun, Rsun=6.95e10 cm, Myr

-units in stderr: units scaled to Msun, Rsun=6.95e10 cm, Myr

Note * = Iscale is that in stderr ([R]: .. pc)

or $2.255e-8/(\text{value in stdout})$

where $2.255e-8=R_{\text{sun}}$ in pc

8) running with PBS

SEE launch_starlab.sh

```
#!/bin/bash
#PBS -N bigZ1N9
#PBS -A IscrC_GClife2
#PBS -q longpar
#PBS -l walltime=24:00:00
#PBS -l select=1:ncpus=1:ngpus=2
module load gnu
module load profile/advanced
module load boost
module load cuda
LD_LIBRARY_PATH=/cineca/prod/compilers/cuda/4.0/none/lib64:/cineca/pr
od/compilers/cuda/4.0/none/lib:/cineca/prod/libraries/boost/1.41.0/intel--
11.1--binary/lib:/cineca/prod/compilers/intel/11.1/binary/lib/intel64
export LD_LIBRARY_PATH
sh /plx/userexternal/mmapelli/Z001/big_Z1_9.sh
```

Shell

Job name

Project name

Queue type

Time

1 node, 1 cpu, 2 gpu

Load modules

New library path

Runs big_Z1_9.sh

8) running with PBS

launch_starlab.sh calls big_Z1_9.sh:

```
echo $PWD
echo $LD_LIBRARY_PATH
/plx/userexternal/mmapelli/Z001/kira -t 500 -d 1 -D 1 -b 1 -f 0.3 \
    -n 10 -e 0.000 -B -s 1361557926 \
    < $CINECA_SCRATCH/Z1bb/ICs/npppp9_645 \
    > $CINECA_SCRATCH/Z1bb/new_cineca9_bin_N50000_frac00_W5_Z1.txt5 \
    2> $CINECA_SCRATCH/Z1bb/ew_cineca9_bin_N50000_frac00_W5_Z1.txt5
```

To submit launch_starlab.sh
qsub launch_starlab.sh

To see if running (R) or queued (Q)
qstat -u username

To delete if wrong
qdel job_id

9) CREDITS for STARLAB:

- * Thank the authors in the acknowledgments (Portegies Zwart, McMillan, Makino, Hut,...)
- * Cite Portegies Zwart+ 2001MNRAS.321..199
Portegies Zwart & Verbunt 1996A&A...309..179P
- * If use GPU, thank the authors of Sapporo: Gaburov, Harfst, Portegies Zwart and cite Gaburov+ 2009NewA...14..630G
- * If use my metallicity-dep. Version
cite Mapelli+ 2013MNRAS.429.2298M

10) Online material:

http://www.science.uva.nl/sites/modesta/wiki/index.php/Starlab_tools

and of course

<http://www.sns.ias.edu/~starlab/index.html>

Download my version and templates

The **HARDWARE**: Graphics Processing Units

MULTIPLE PROCESSING CORES in one chip:
all the cores have FAST SHARED MEMORY

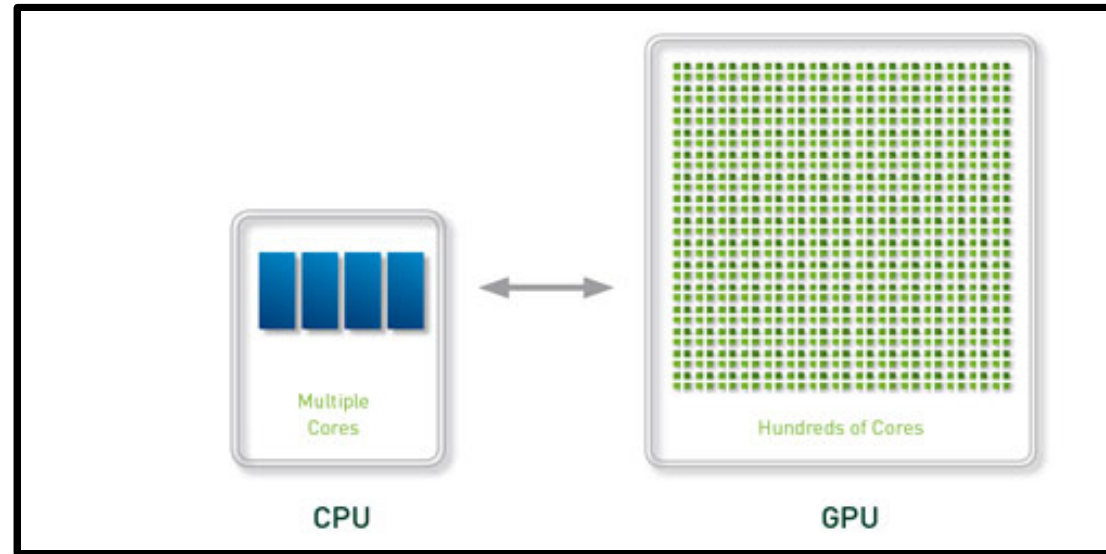
DOUBLING of performance over ~9 months
Instead of ~18 months for GPU (NVIDIA 2007)

LOW COST respect to GRAPE

Born for primitive graphical operations (computer games)
NOW programmable

→ various programming languages
e.g. Cg (Mark et al. 2003),
Compute Unified Device Architecture (CUDA, by NVIDIA)

Single Instruction Multiple Data (SIMD) technique:
many processing units perform the same series of operations on different sub-samples of data



Graphics Processing Units:

CUDA basic principle: applications consist of 2 parts

- 1) KERNEL:** executes operations on the GPU (C based)
- 2) executes on CPU (mostly transfer of data CPU ↔ GPU)**

Particularly effective for GPUs because:

- each GPU contains many **THREADS** with **SMALL** but **VERY FAST SHARED MEMORY**
- a Kernel can be run at the same time by different threads
- MEMORY** of the **HOST COMPUTER** is **LARGE** but **CPU ↔ GPU TRANSFER** is **SLOW**:
the less the communication the better the performance

Graphics Processing Units:

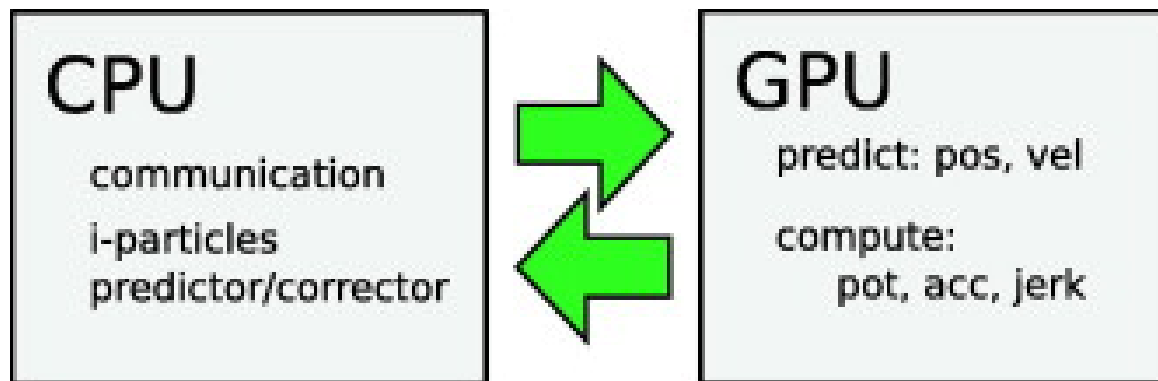
N-body example:

With Hermite scheme (p and v predicted by acceleration and jerk)

$$\mathbf{x}_{\text{pred}} = \mathbf{x}_0 + \mathbf{v}_0 dt + \mathbf{a}_0 dt^2 / 2 + \mathbf{j}_0 dt^3 / 6$$

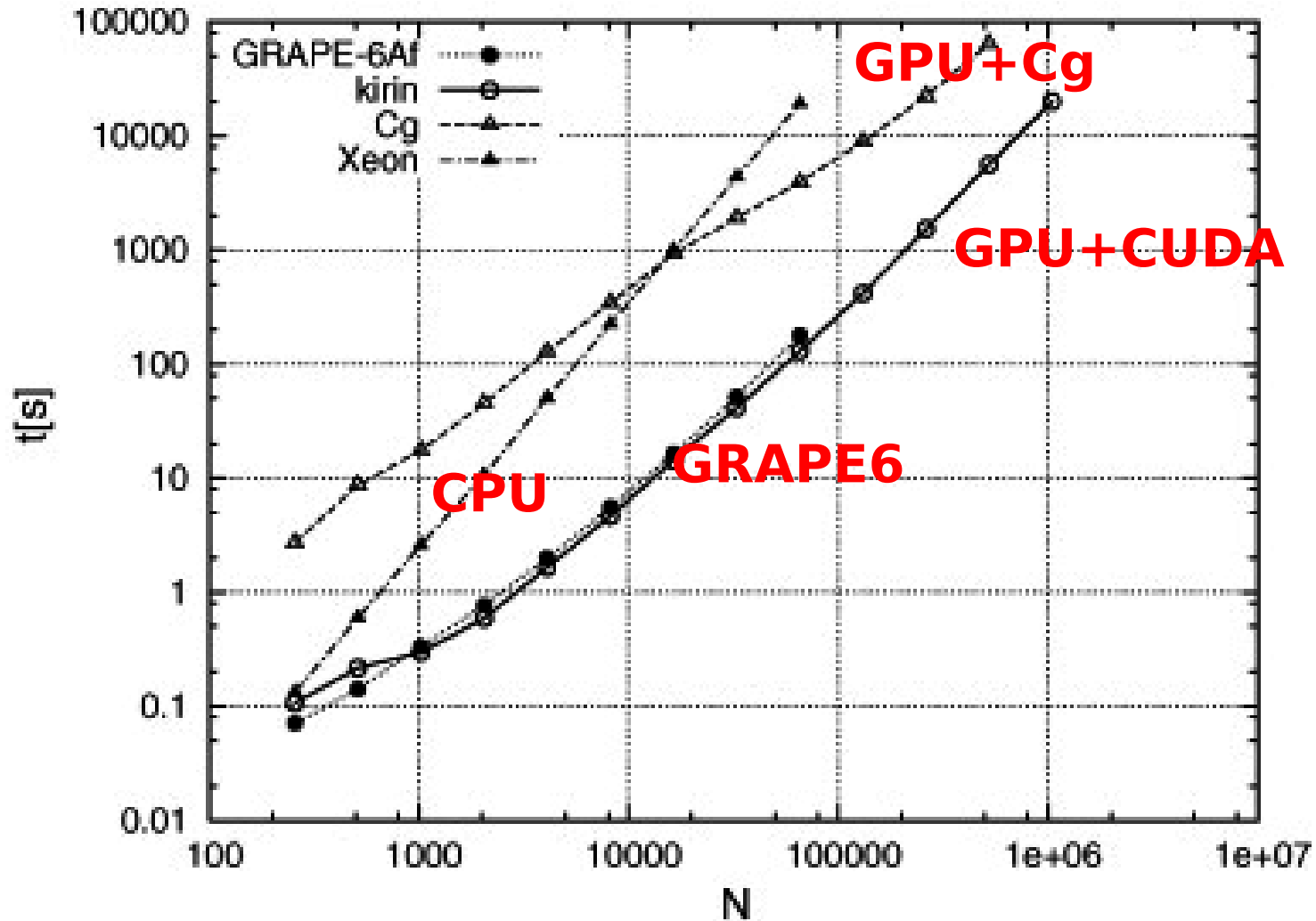
$$\mathbf{v}_{\text{pred}} = \mathbf{v}_0 + \mathbf{a}_0 dt + \mathbf{j}_0 dt^2 / 2$$

- CPU sends to GPU mass, position and vel of particles
- each thread operates over 1 particle put in shared memory with other threads in the same bundle (e.g. 128 threads per bundle)
- each thread calculates acceleration and jerk for 'its' particle (partial summation in shared memory for each bundle)
- results saved in the global memory



Graphics Processing Units:

PERFORMANCE (Belleman+2008)



GPU+CUDA BETTER than GRAPE for $N > 512$

References:

- * Mapelli M. et al. 2006, MNRAS, 373, 361
- * Dehnen & Read 2011, arXiv:1105.1082
- * Hurley, Pols & Tout 2000, MNRAS, 315, 543
- * Sippel et al. 2012, arXiv:1208.4851
- * Sigurdsson et al. 2003, Science, 301, 193
- * Agnor & Hamilton 2006, Nature, 441, 192

1. DEFINITION

Newton's EQUATIONS of MOTION:

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

Or

$$\begin{cases} \dot{\vec{r}}_i = \vec{v}_i \\ \dot{\vec{v}}_i = -G \sum_{j \neq i} m_j \frac{\vec{r}_i - \vec{r}_j}{|\vec{r}_i - \vec{r}_j|^3} \end{cases}$$

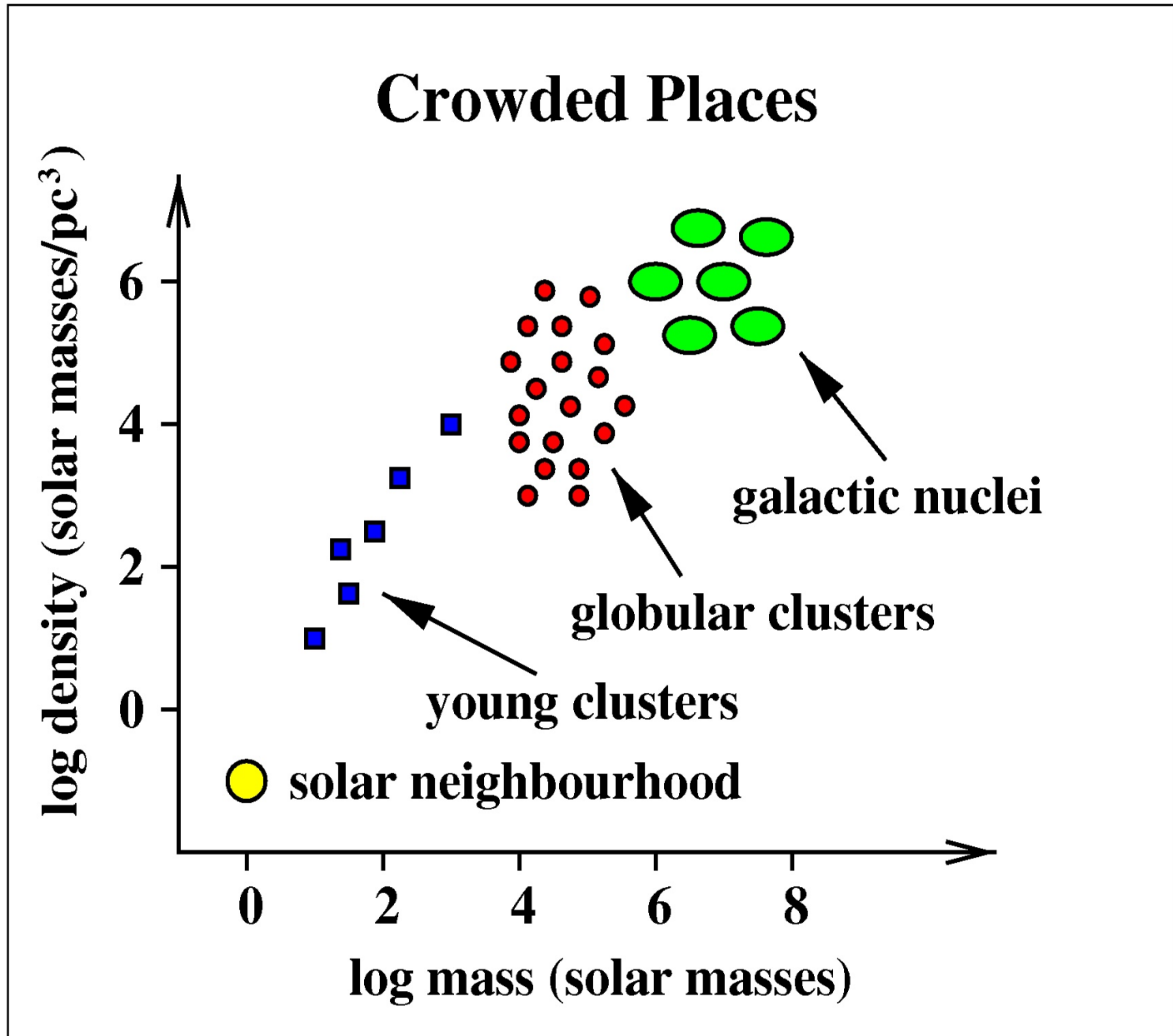
DIRECT N-Body codes calculate all N^2 inter-particle forces
→ SCALES as $O(N^2)$

CODES that USE MULTIPOLE EXPANSION of FORCES for sufficiently distant particles scale as $O(N \log N)$ – see C. Giocoli's lecture

→ Are we crazy? Why do we use direct N-body then?

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

A CARTOON of COLLISIONAL/COLLISIONLESS SYSTEMS



2. Simulating collisional systems

You must resolve SINGLE STARS (softening based codes cannot be used)

Solving (i) equations of motion

→ DIRECT N-BODY CODES

- 1* Forces on binaries are stronger and change more frequently
 - binaries need to be updated more frequently than single stars
 - we need a criterion for different timesteps

Timesteps for BINARIES and THREE-BODY ENCOUNTERS
<< timesteps for other bodies!

- 2* Solve Newton's equations for EACH star directly → scale as N^2
+ relaxation time scales as N

→ time complexity $t_{CPU} \propto N^3$

(cfr with tree codes and Monte Carlo $\propto N \ln N$)

1) Simulating collisional systems

INTEGRATION SCHEME:

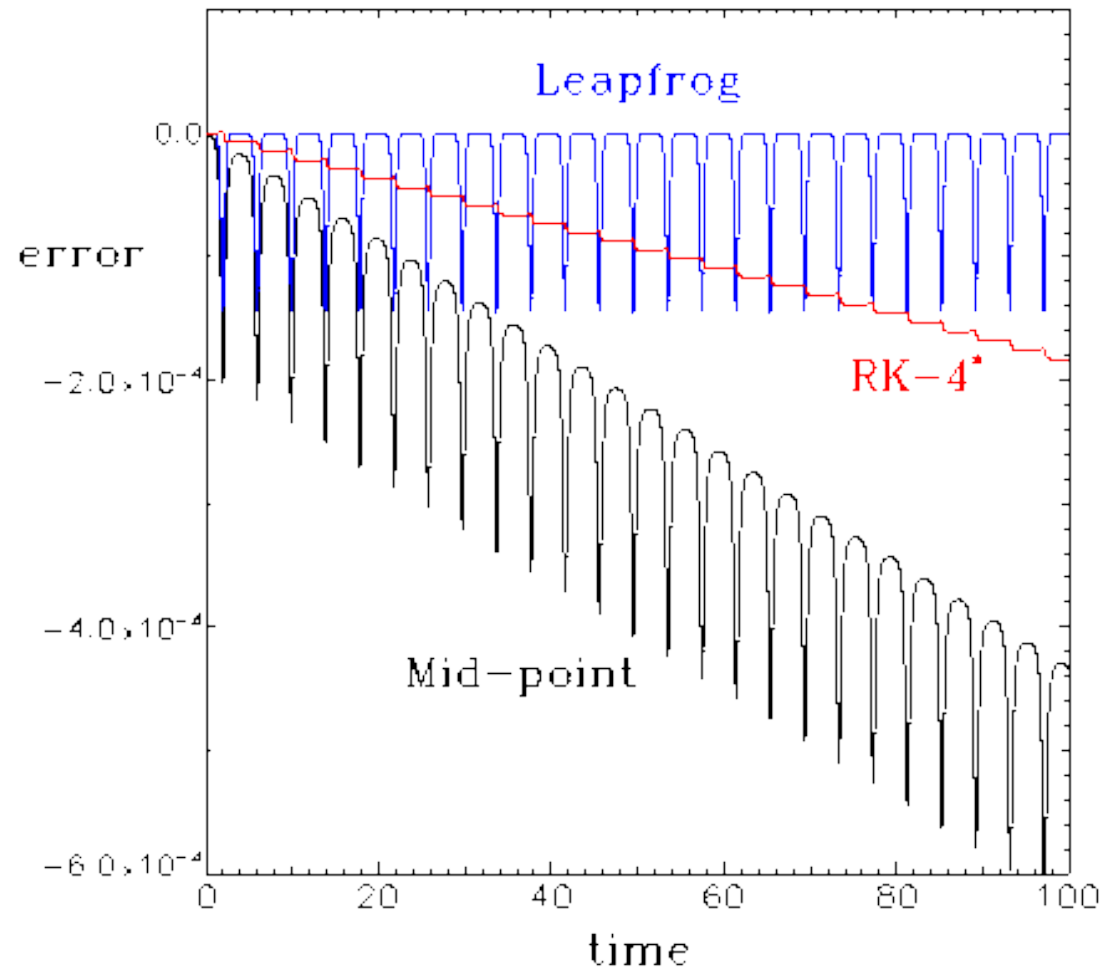
Usage of a high-order integrator ($> \sim 4$) often with a predictor-corrector scheme, because needs HIGH PRECISION on a SMALL TIME DURATION

EXAMPLES: better Hermite than Leapfrog

Leapfrog is SYMPLECTIC: solves correctly a Hamiltonian
(even if an APPROXIMATED Hamiltonian)

i.e. is TIME REVERSIBLE → good solution on a long time-scale (for conservation of angular momentum, energy, etc), but with large errors on the single timesteps

Hermite is NOT symplectic,
but is 4th order on each
timestep: much more accurate
on single timesteps



1) Simulating collisional systems

2nd order **LEAPFROG** kick-drift-kick (KDK)
or drift-kick-drift (DKD).

Drift=operation that changes only position

Kick= operation that changes only velocity

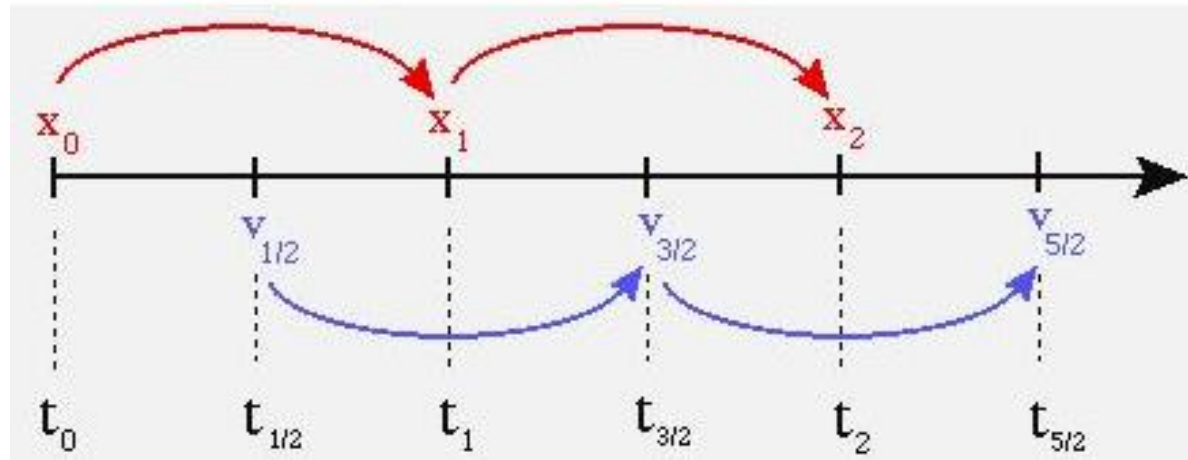
NEEDS an intermediate (auxiliary) quantity (velocity in the KDK,
position in the DKD) that will be corrected with a second operation:

1) Simulating collisional systems

D $x_{n+\frac{1}{2}} = x_n + v_n \frac{\Delta t}{2}$

K $v_{n+1} = v_n + a(x_{n+\frac{1}{2}}) \Delta t$

D $x_{n+1} = x_{n+\frac{1}{2}} + v_{n+1} \frac{\Delta t}{2}$



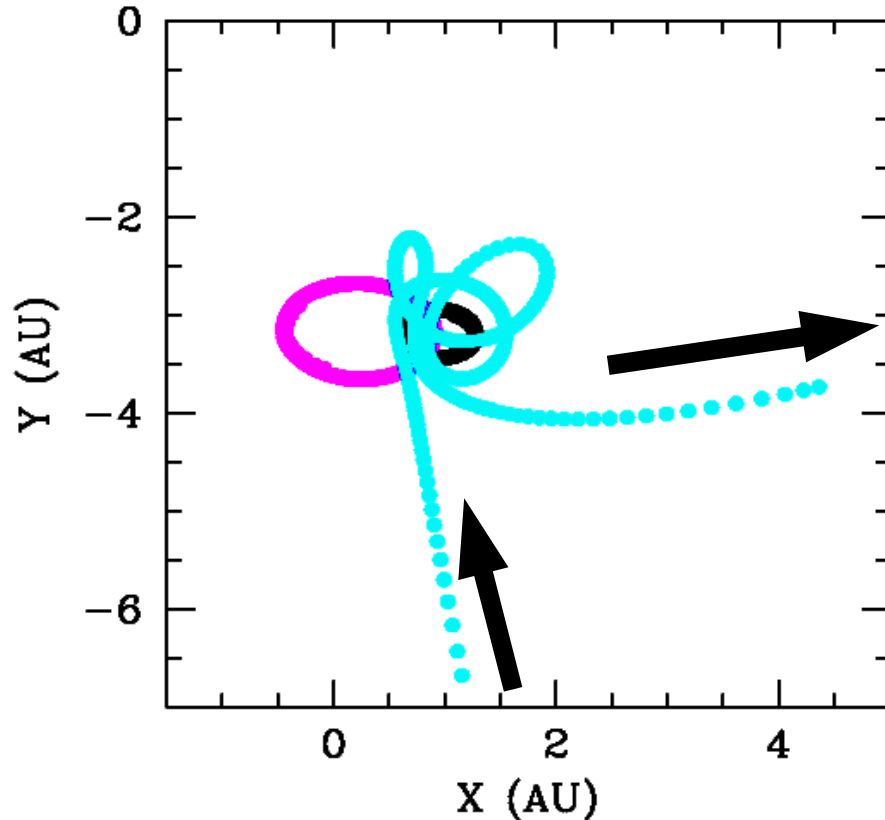
K $v_{n+\frac{1}{2}} = v_n + a(x_n) \frac{\Delta t}{2}$

D $x_{n+1} = x_n + v_{n+\frac{1}{2}} \Delta t$

K $v_{n+1} = v_{n+\frac{1}{2}} + a(x_{n+1}) \frac{\Delta t}{2}$

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

→ 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!