High-Performance Computing

Introduction

Me

Thomas Fogal

- “Tom”, please
- thomas.fogal@uni-due.de

Course Goals

Common language

Software engineering, practicals

HPC: theory *&& practice*

- Distributed memory systems, MPI
- Memory wall. NUMA
- Shared memory, threading; OpenMP
- Filesystems, I/O
- Load balancing
- Profiling and scalability

'Research' / final project

Course Organization

1 part 'C', 1 part N-Body, 1 part Final Project

Clusters, MPI, OpenMP

- CUDA/OpenACC/OpenCL off-topic
  - But we have another (master's) course for that!

Language choice: C, C++, Fortran 90+. GNU.

Why take this course?

You will come out a **much** better programmer.

Parallelism is **not** going away.

Specialized skill that's in demand, and growing

Popularity at parties++!

You won't find a more interested teacher!

- “Tom's erreichbarkeit war super!”
- “Die Kommunikation zwischen Lehrstuhl und Stundenten war hervorragend.”
- “... by far my favorite course this semester. Learned a lot!”
- “... dies eines der besten Kurse war die ich bisher (???) durfte.”

It's **fun!**

Grading

N-body is individual

- Points will be on moodle
- Points will increase

Final project is group-based

- Proposals in January (concurrent with N-body!)
- Due end of February
- You choose the number of points
- Final demo / writeup

Grading is simple: points earned vs. points available ratio

No quizzes / tests / etc.

HiWis? Thesis? Research?

- Our groups' focus
- Ace the class first.

“Plan”

1. Intro, Linux essentials
2. C: pointers, basic IO
3. N body
4. Distributed memory
5. Shared memory
6. Memory access, t/s consistency
7. Scalability, profiling
8. Distributed Filesystems
9. Future clusters

Assignments

6 or 7 total

- Generally a couple of weeks per
- Build on each other
  - Live with your code!

No sharing code!

No sharing code!

- … but please discuss :-)

Expectations

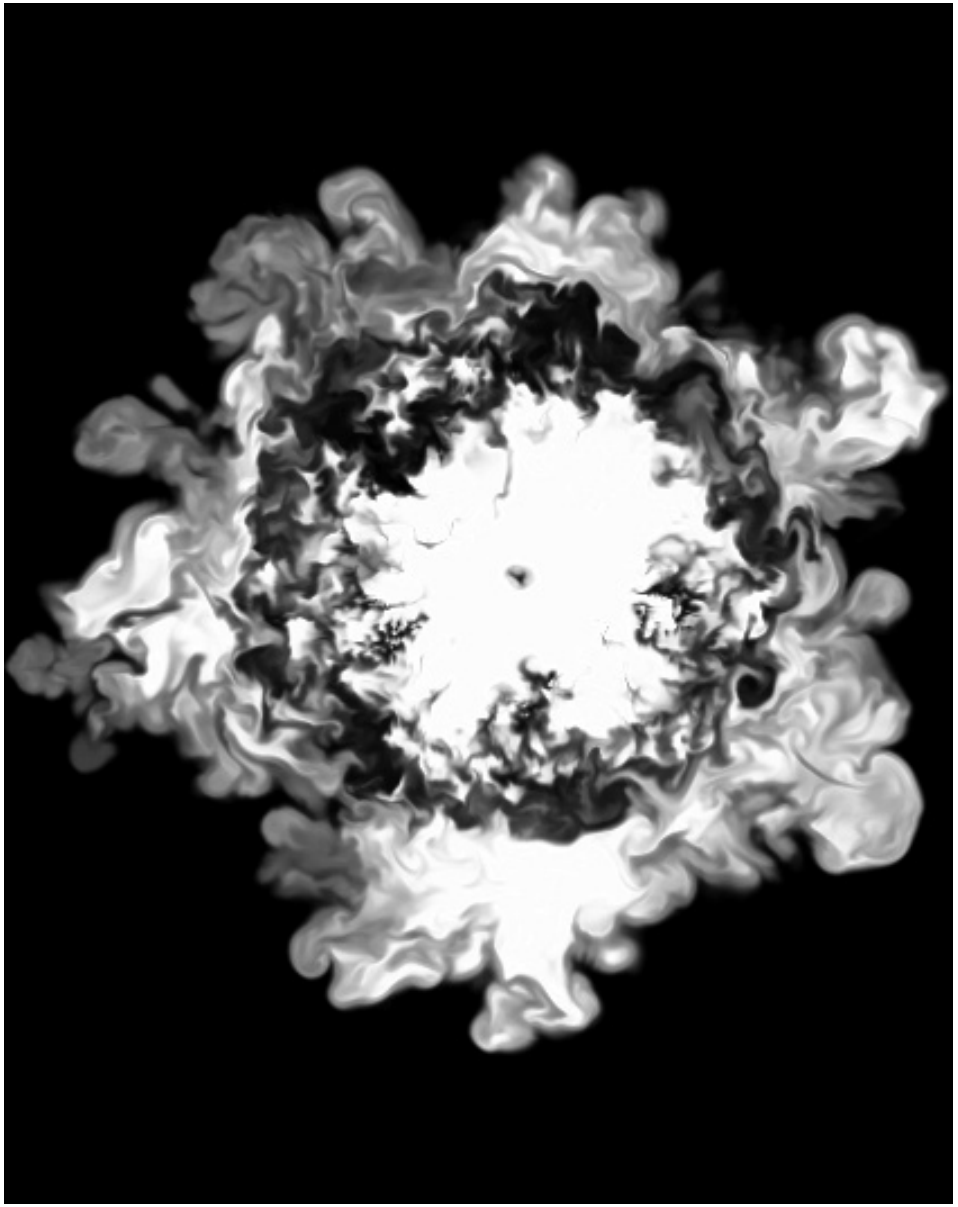
- Know an imperative language
- Do your homework
- ***Ask Questions***
- *Issues with me?*
  - Talk to me.
  - Meet with Prof. Krüger (jens.krueger@uni-due.de)
  - DUE admin

Practicals / Recommendations

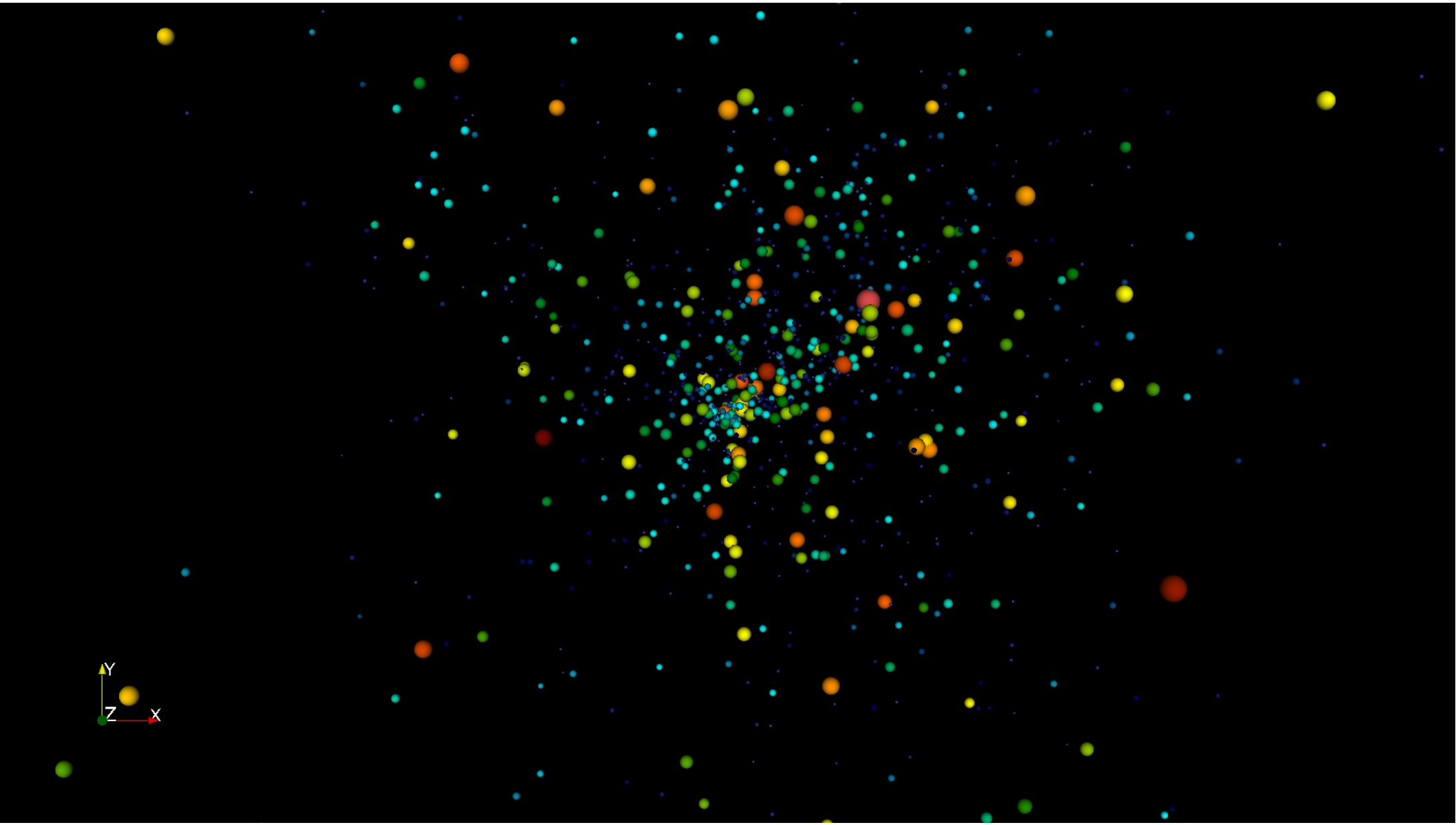
- Use C.
- Code locally, test on Cray
  - Don't waste CPU hours.
  - VM if you need it
  - ssh to student.uni-due.de

Simulation Overview

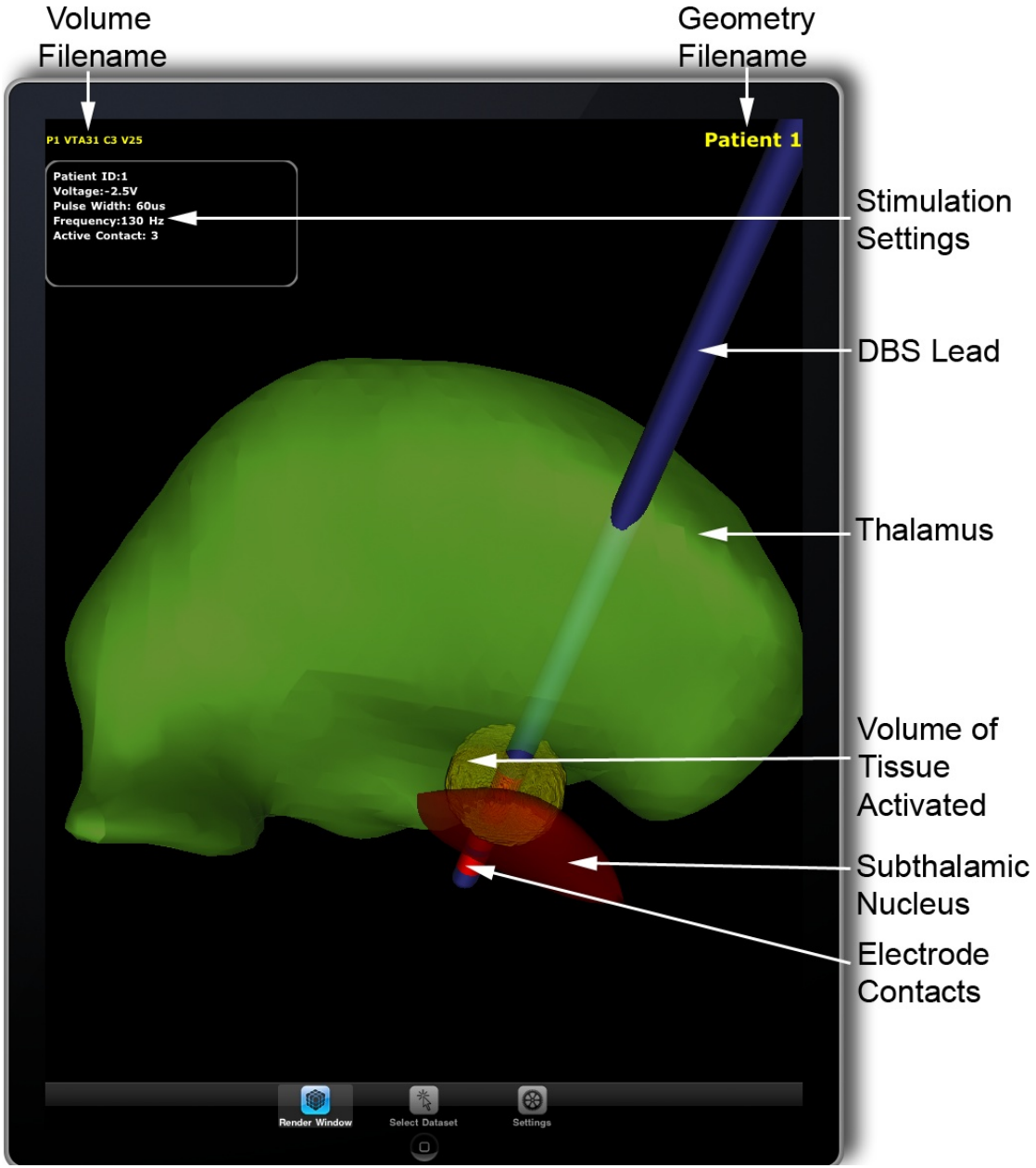
Simulation Scenario: Combustion



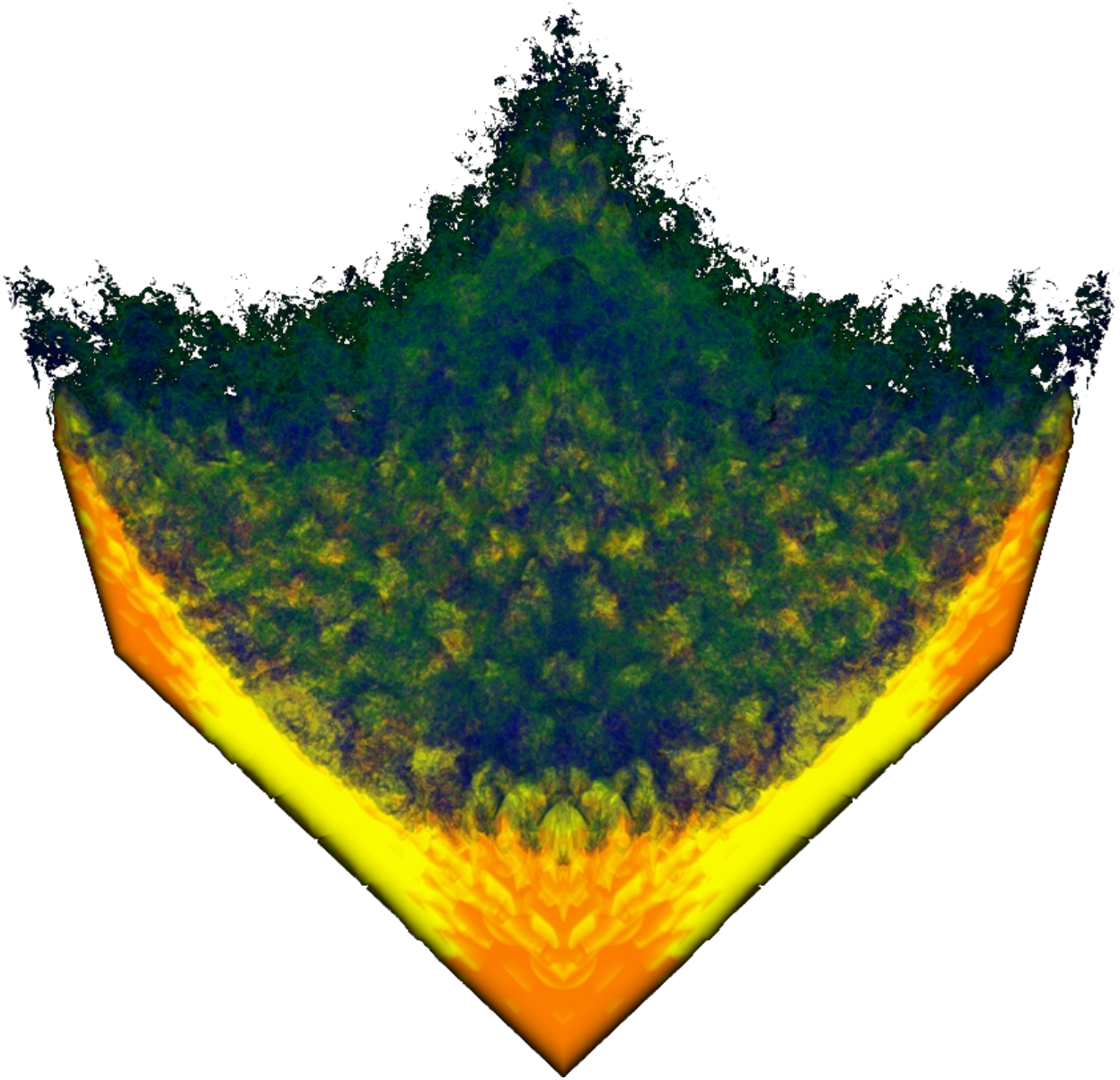
Simulation Scenario: NBody



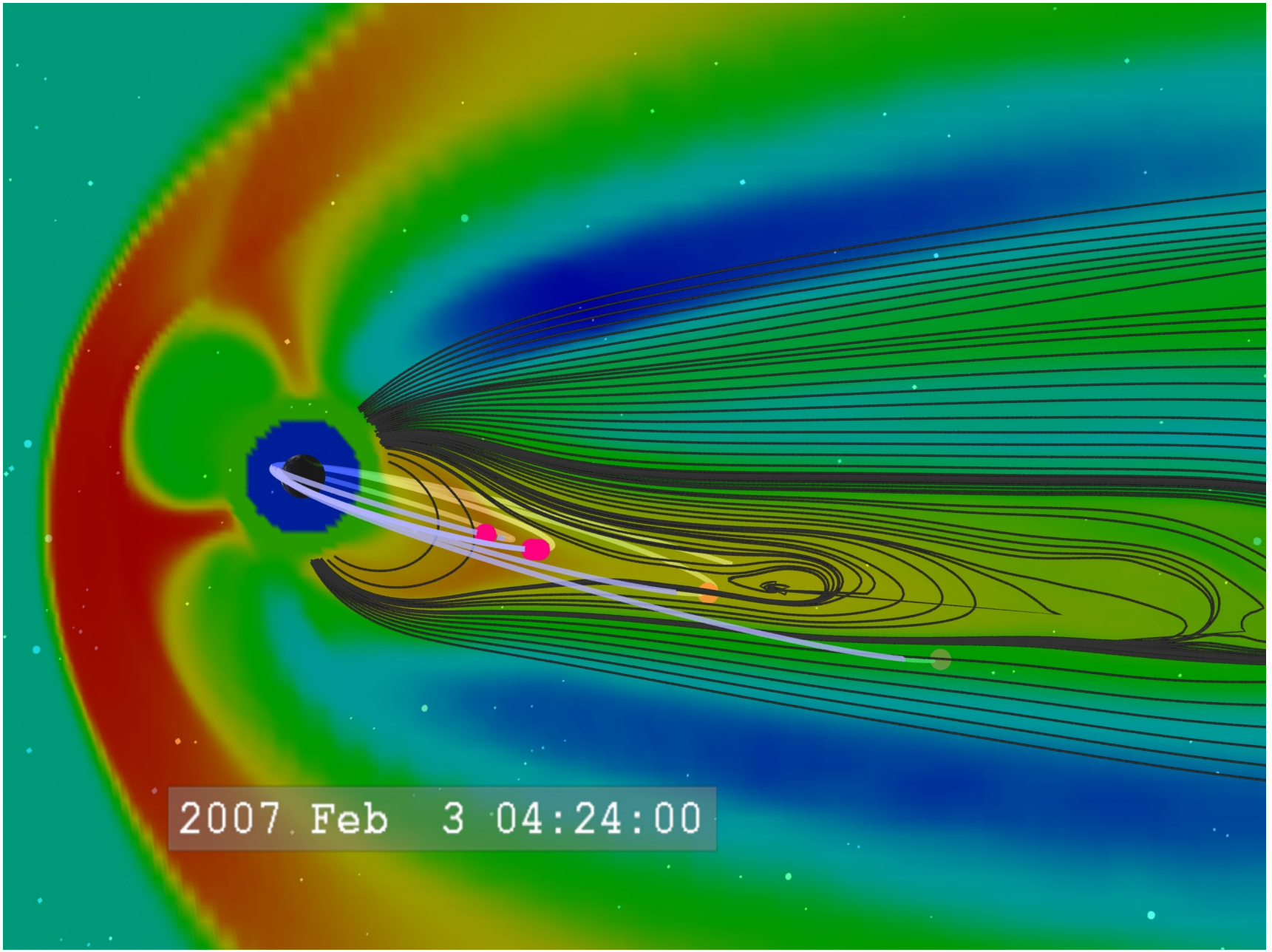
Simulation Scenario: DBS



Simulation Scenario: Fluid mixing



Simulation Scenario: MHD



Simulation Cycle

1. idea/theory/model
2. discretize domain
3. encode math into calculation
4. run simulation
5. verify result / explore data
    1. See our SciVis course :-)
6. GOTO 1

Parallel Simulation

1. Reduce time to solution
2. More nodes → more memory

Parallelization

Hard.

- Race conditions
- Coordination
- Performance!

How?

- Automatic parallelization?
- Threads?
- MPI
  - System assigns procIDs → processors!

Threads

- Task-based parallelism
- data parallelism?

Message Passing Interface

Single program, different data

- SPMD

Each process assigned an ID

Explicit synchronization

Explicit memory transfer

Input

Disk → memory (restart)

Configuration

- Derived from visualizing the data :-)

Analysis / statistics

Output

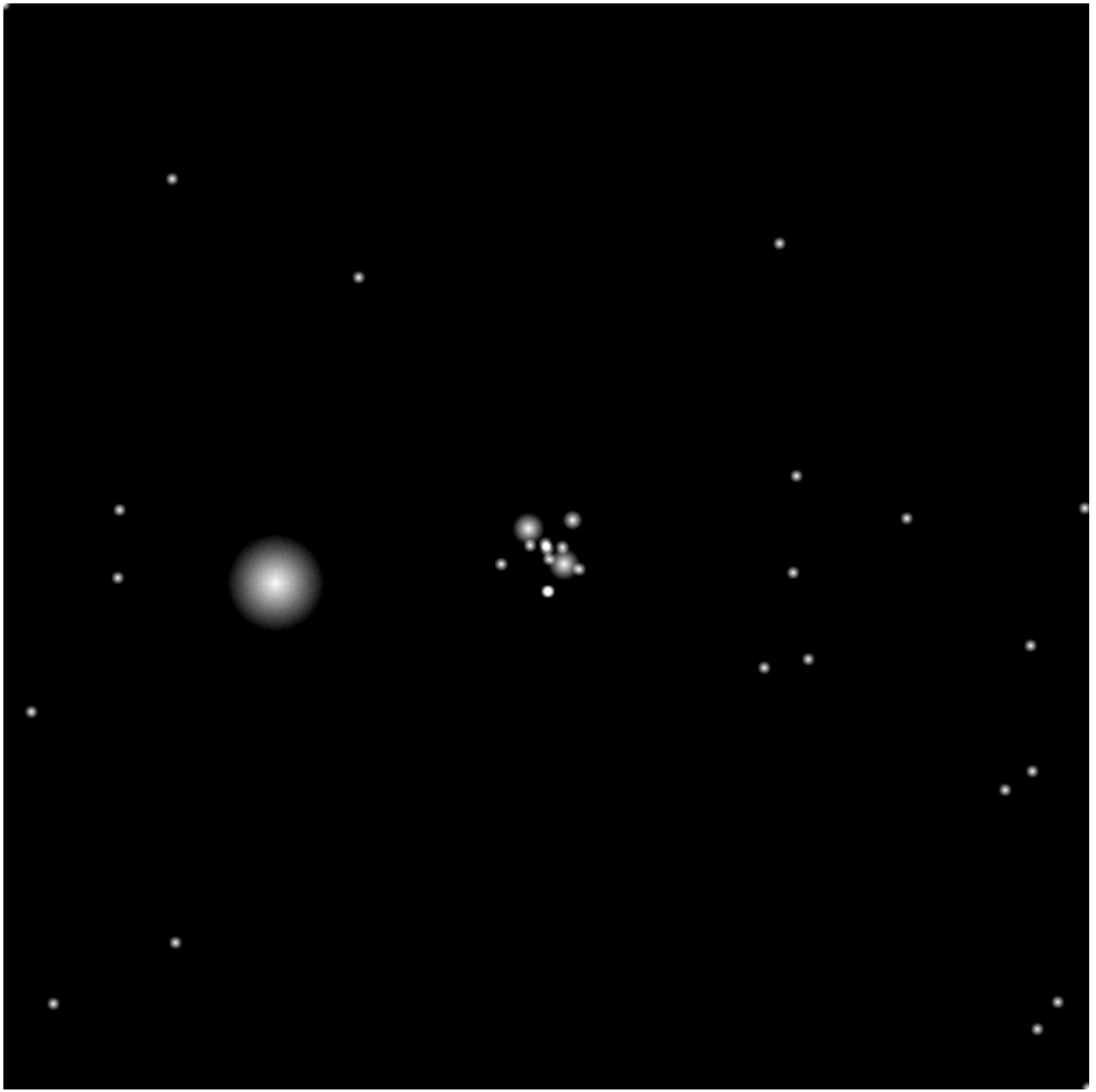
Distributed file systems

- GFS, GoogleFS (GFS..), Hadoop FS, Lustre, (NFS?)

Usage patterns

- Dump memory to disk (checkpoint)
- Data arrays
- Appends (log files)

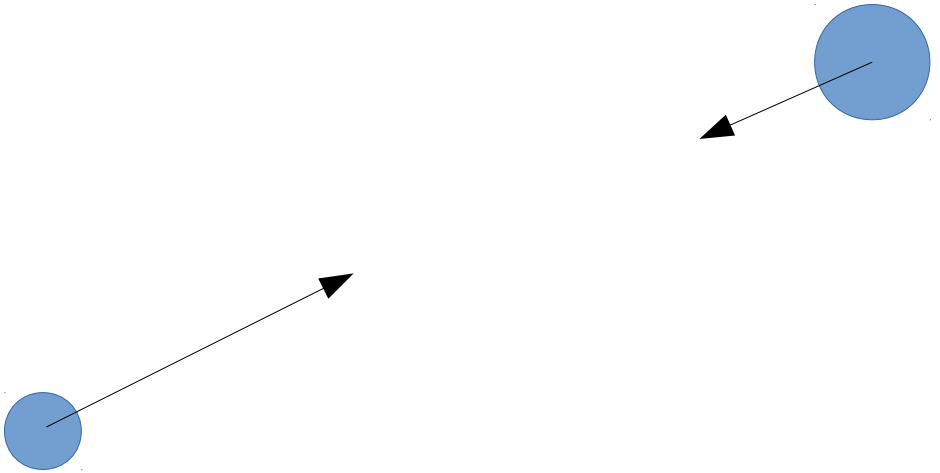
N-Body Problem



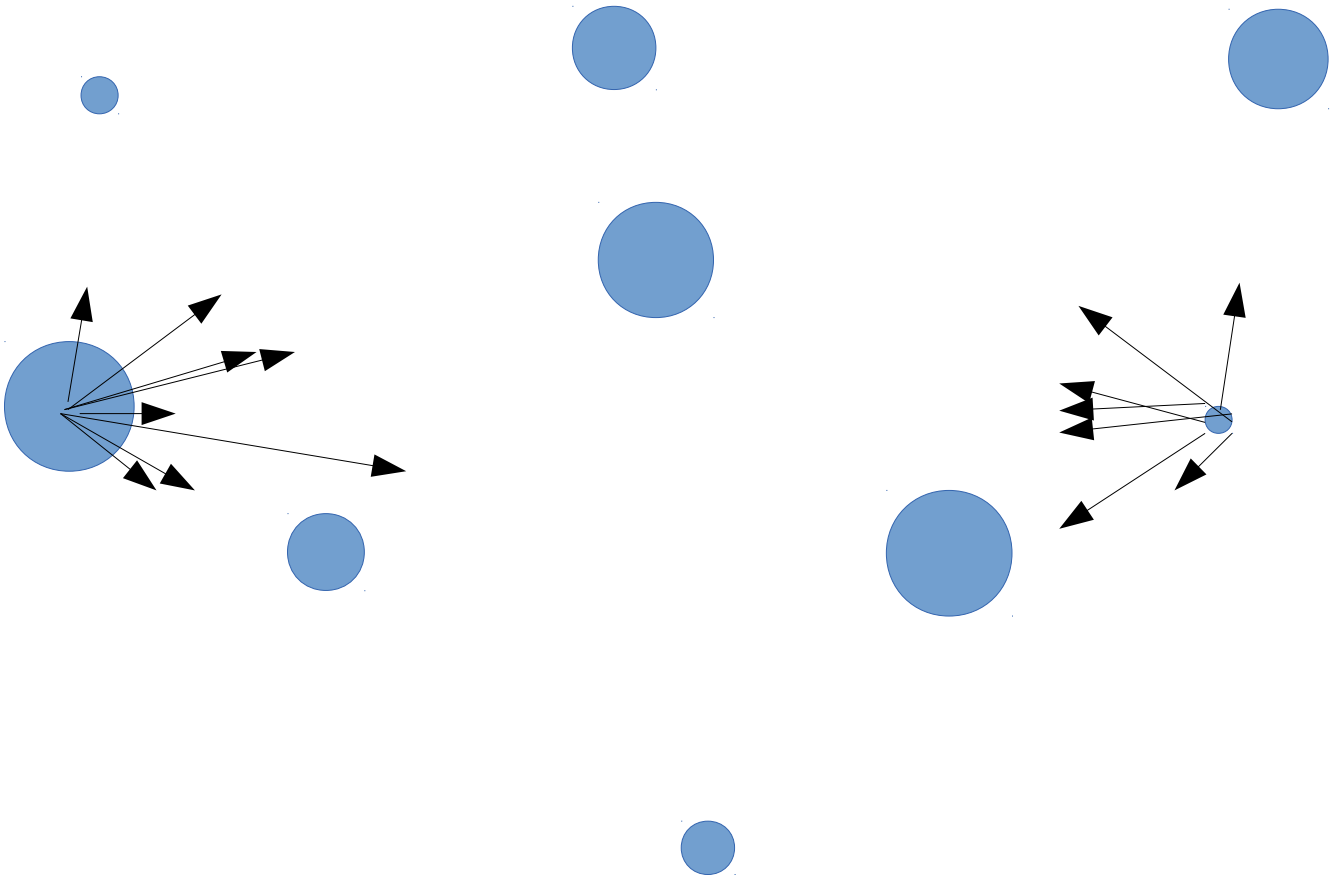
Questions

Newtonian Gravity

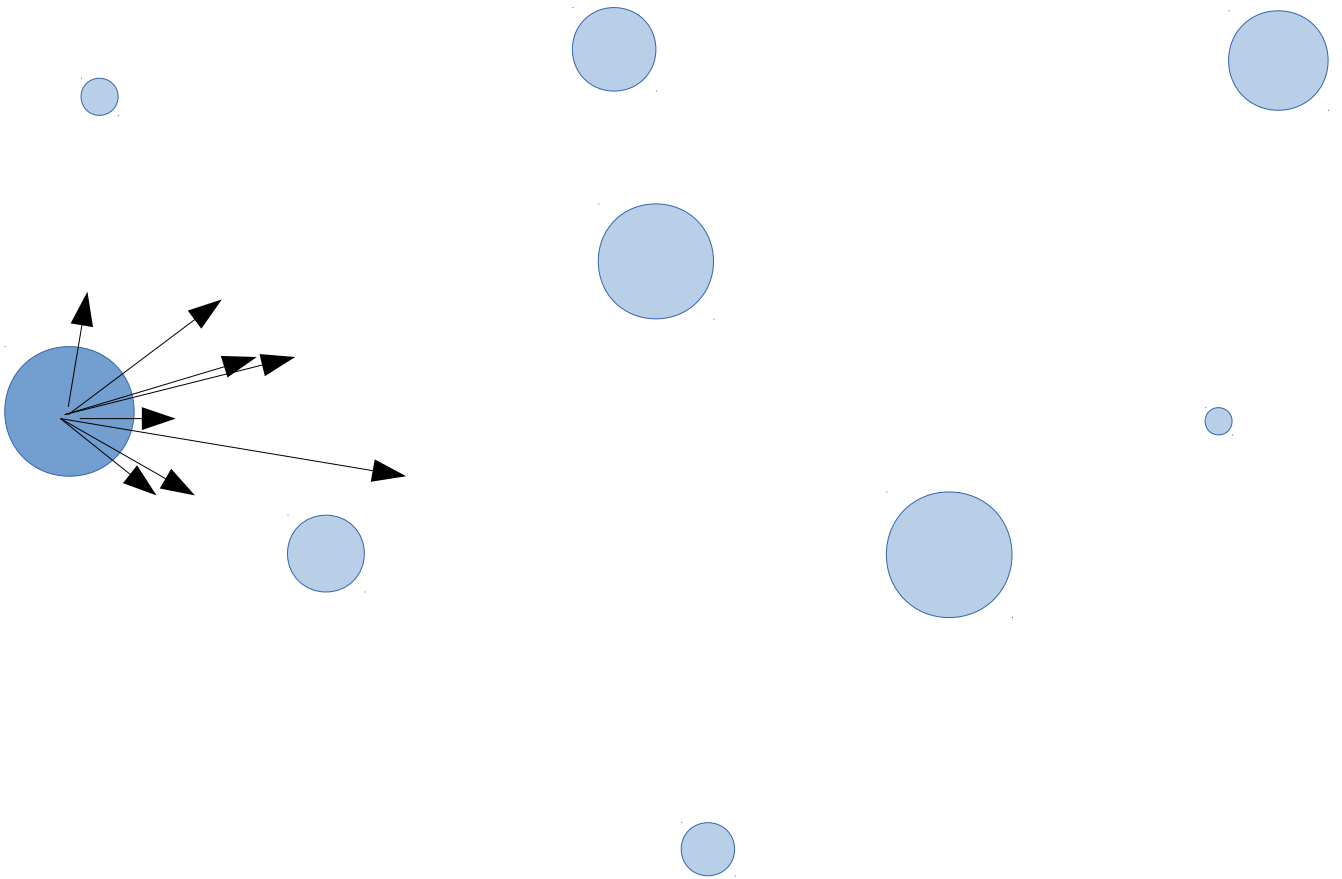
$$F_1 = F_2 = G\,(m_1 m_2)/r^2$$



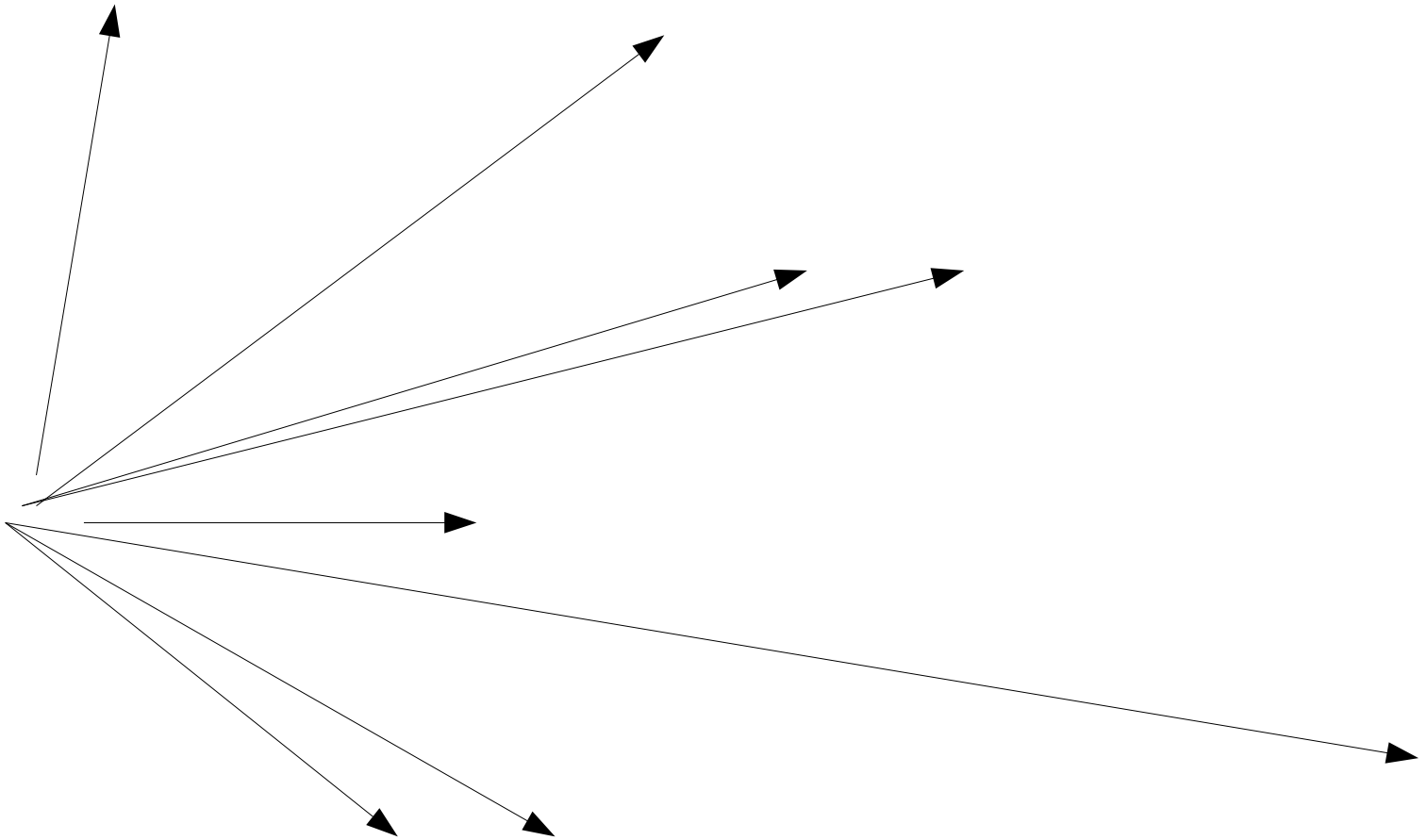
Many Particles



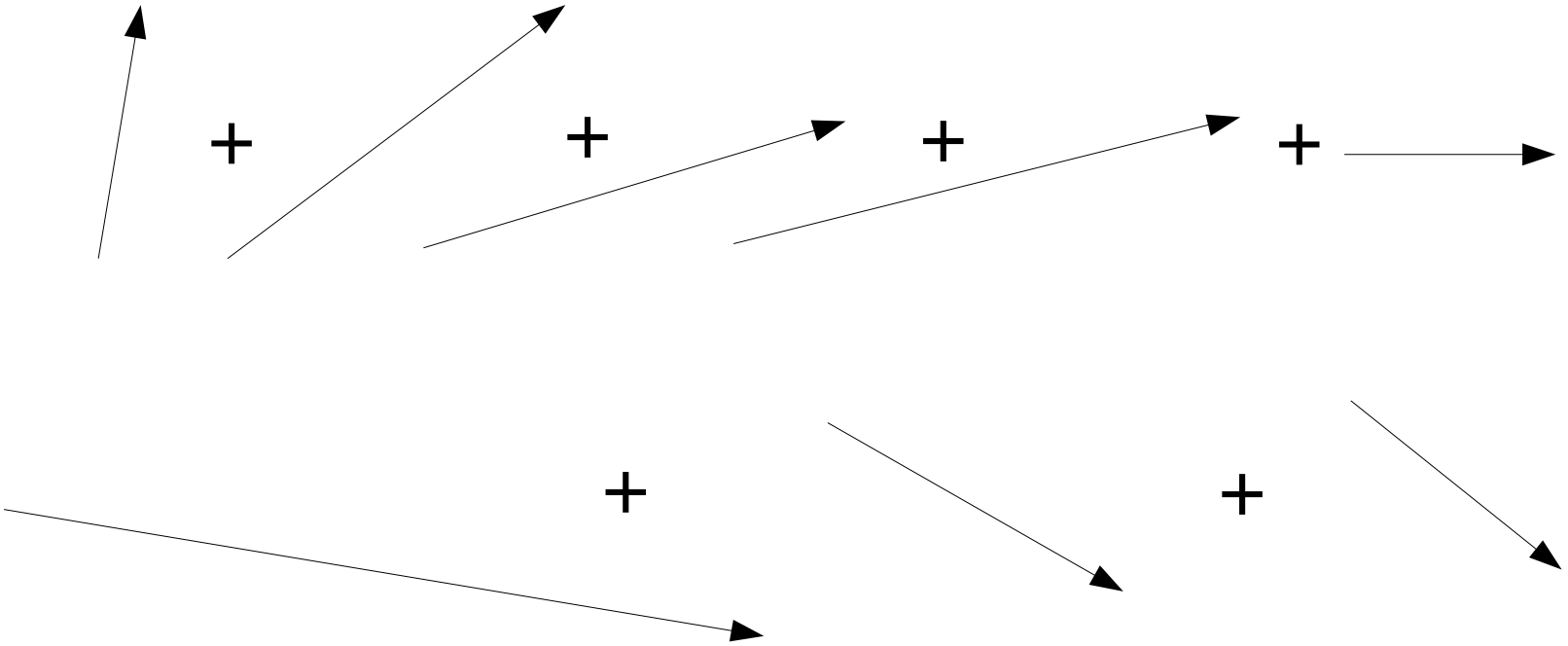
Summation of Forces



Vector Addition



Vector Addition



Particle Force Summation

$$m_i p_i = G \sum_{j=1}^{N-1} \left(m_j m_i \left(p_j - p_i \right) \right) / \left| p_j - p_i \right|^3$$

$$G = 6.67 \times 10^{-11} N \left(\frac{m}{kg} \right)^2$$

Practicals

How large is T?

Linux Essentials

Terminal

```
$ _

$ cmd1
$ cmd2
```

Output

```
$ cmd
cmd's output
more output
$
```

Canceling Commands

```
$ ./a.out^C
$
```

Navigation

```
$ cd directory
$ cd ..
$ ls
$ pwd
```

Compiling

$ gcc -Wall -Werror -ggdb3 file.c

-Wall: turn on **all w**arnings

-Werror: **w**arnings *are* **errors**

-ggdb3: debug symbols on

-O3: heavy **O**ptimization

Debugging

```
$ gdb -q ./a.out
(gdb) run
...
^C
(gdb) bt
(gdb) list
```

Valgrind

```
$ valgrind --leak-check=full --track-origins=yes --leak-resolution=high --show-reachable=yes ./a.out ...
```