The Abdus Salam International Centre for Theoretical Physics

United Nations Educational, Scientific and Cultural Organization

IAEA International Atomic Energy Agency

Search

Search in Conferences:

- Overview
- Programme
- Speakers
- Apply here
- Practical info

4th Workshop on Advanced Techniques for Scientific Programming and Management of Open source Software Packages | (smr 2875)

Modern scientific research work involves using computers for simulation, modeling, data processing and visualization. The growing complexity of those calculations favors using software packages that provide an abstract interface to computations through scripting language interfaces, modular software design, and code reuse. Such design patterns also facilitate transparent optimizations for modern multi- core architectures or accelerators, and specialization of contributors to subsets of a package program. Software package developers therefore need to learn how to work effectively in a collaborative environment.

This Hands-on Workshop focuses on disseminating best practices and building fundamental skills in creating, extending and collaborating on modular and reusable software frameworks with a scripting language interface. The curriculum also covers using modern collaborative software management tools, testing frameworks, and embedding structured documentation into software packages.

*****

Organizers

Ivan Girotto, David Grellscheid, Luca Heltai,
ICTP Local Organizer: I. Girotto

Co-sponsors

ma per seguir virtute e canoscenza
SISSA

Mission - An institute run by scientists for scientists

- Foster the growth of advanced studies and research in physical and mathematical sciences, especially in support of excellence in developing countries.
- Develop high-level scientific programmes keeping in mind the needs of developing countries, and provide an international forum of scientific contact for scientists from all countries.
- Conduct research at the highest international standards and maintain a conducive environment of scientific inquiry for the entire ICTP community.
- Thanks to the generous funding from the Italian Government, UNESCO and the IAEA, ICTP has been able to initiate and implement various schemes of support and assistance to scientists from developing countries.

The Abdus Salam
International Centre
for Theoretical Physics

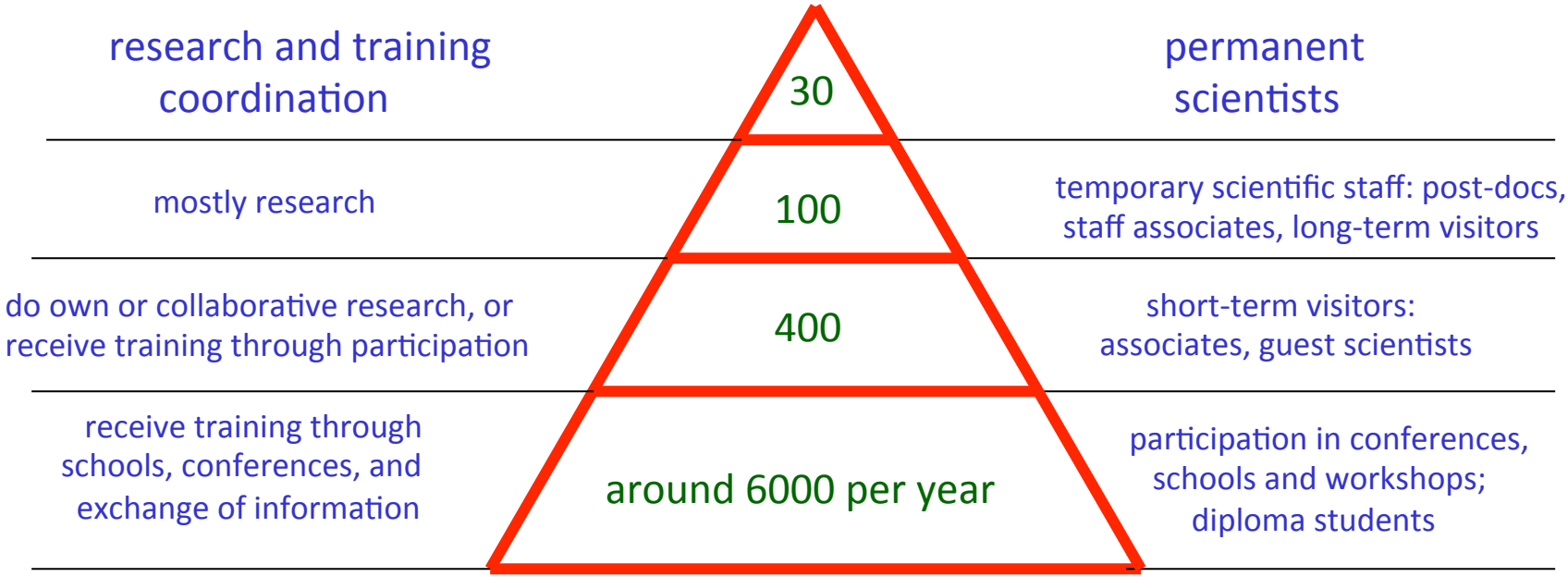
United Nations
Educational, Scientific and
Cultural Organization

IAEA
International Atomic Energy Agency

ICTP Partner Institutes

- Mesoamerican Centre for Theoretical Physics has been established in Mexico in collaboration with the Universidad Autónoma de Chiapas (UNACH).
- The ICTP Eurasian Centre for Advanced Research (ICTP – ECAR) is a new regional centre of ICTP, which is currently in the process of being established in Turkey based on the agreement between ICTP and İzmir Institute of Technology (IZTECH).
- The ICTP South American Institute for Fundamental Research, ICTP SAIFR, is a regional centre for theoretical physics created in collaboration with the State University of Sao Paulo (UNESP) and the Sao Paulo Research Funding Agency (FAPESP).
- Future centres are planned for Rwanda and China.

ICTP from Trieste to the World



Over 200.000 visit/year to the ICTP media (see www.ICTP.TV) for remote training!

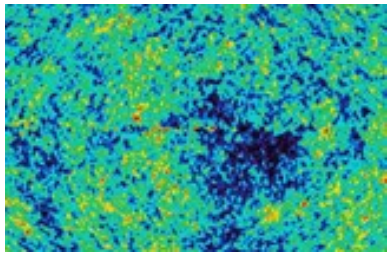

The Abdus Salam
International Centre
for Theoretical Physics
IAEA
The Abdus Salam
International Centre
for Theoretical Physics
50th Anniversary 1964–2014
IAEA
Research
Programmes
Scientific Calendar
PRE-PHD PROGRAMMES
ICTP Postgraduate Diploma Programme
ICTP/IAEA Sandwich Training Education Programme
DEGREE PROGRAMMES
Joint ICTP/SISSA PhD Programme in Physics and Mathematics
Joint PhD Programme, Earth Science and Fluid Mechanics
Joint Masters in Physics
Joint ICTP/Collegio Carlo Alberto Program in Economics
International Master, Physics of Complex Systems
Masters in Medical Physics
Masters in High Performance Computing
CAREER DEVELOPMENT
Federation Scheme
Associates Scheme
LABORATORY OPPORTUNITIES
Training and Research in Italian Laboratories
ICTP-ELETTRA Users Programme
ICTP Laboratories
SCIENTIFIC OUTREACH
Office of External Activities
ICTP Partner Institutes
ICTP in Africa
Science Dissemination Unit
African Review of Physics

ICTP Scientific Calendar

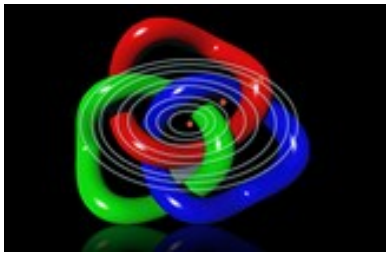
- Schools, Conferences, Workshops around the year
- Half of them on subjects related to main research areas (core)
- The rest on many subjects:
  medical physics, optics, nano physics, plasma physics, electronics, high performance computing, biophysics, satellite navigation, science dissemination and e-learning, m-science, entrepreneurship, nuclear physics (IAEA), teacher training, 3-D Printing, etc...
- http://www.ictp.it/scientific-calendar.aspx

Scientific Sections

High Energy Cosmology and Astroparticle Physics

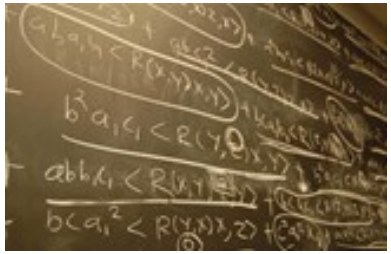


Condensed Matters and Statistical Physics



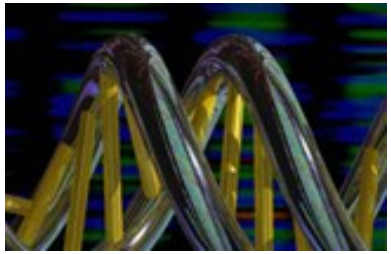
Earth System Physics

Mathematics



Applied Physics

New areas



High-Performance Scientific Computing activities at the ICTP

- HPC service and HPC application consulting
- HPC and Scientific Programming Dissemination (2016)
  - The CODATA-RDA School of Research Data Science (August 2016)
  - Introductory School on Parallel Programming and Parallel Architecture for High-Performance Computing (October 2016)

Introductory School on Pa... | Welcome to Master in Hig...

www.mhpc.it

MHPC

Portal_ICTP | Helpdesk | RT | ICTP | PhoneBook | Google_Calendar | SIS | Google | GoogleMap | Tuttocitta' | ICTP - Time and ... | ICTP - FABS Help ... | CPU-Bench

MHPC
Master in High Performance Computing

SISSA | ICTP The Abdus Salam International Centre for Theoretical Physics

ABOUT | OBJECTIVES | COURSES | APPLY | FAQ | PEOPLE | SPONSORS

MHPC

The Master in High-Performance Computing (MHPC) is a high-level degree program that aims to train students to solve complex problems with HPC techniques.

WHY

Set in a stimulating research environment, the MHPC is an innovative, hands-on training and education program to prepare students for exciting careers in the fast-growing field of HPC.

THE TARGET

The master is intended for people with strong interest in advanced programming for scientific computing, software optimization and management of computing platforms.

READ ALL

Workshop Overview: Software Development Basics

Ivan Girotto – igirotto@ictp.it

Information & Communication Technology Section (ICTS)

International Centre for Theoretical Physics (ICTP)

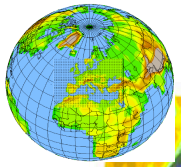
A Roadmap to the Workshop

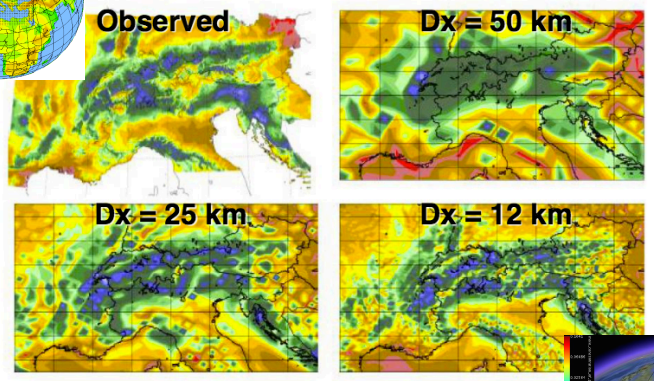
- Focus on software development concepts
- Introduce tools and processes for organizing development and maintenance
- Discuss strategies and best practices
- Explore methodology that encourages collaborative software development
- Favor writing reusable software frameworks
- Work in groups with complementary expertise

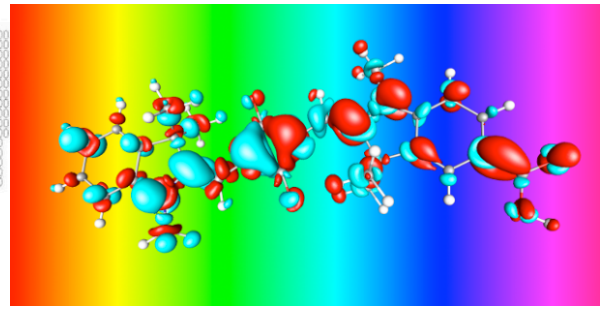
Why use Computers in Science?

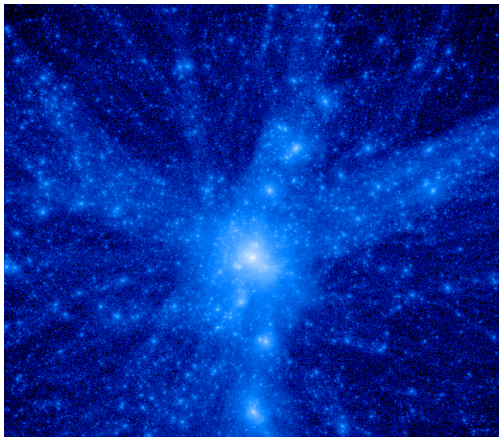
- Use complex theories without a closed solution: solve equations or problems that can only be solved numerically, i.e. by inserting numbers into expressions and analyzing the results
- Do “impossible” experiments: study (virtual) experiments, where the boundary conditions are inaccessible or not controllable
- Benchmark correctness of models and theories: the better a model/theory reproduces known experimental results, the better its predictions

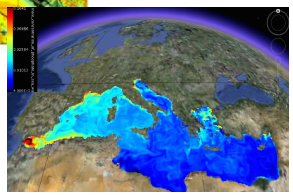
SW in Science

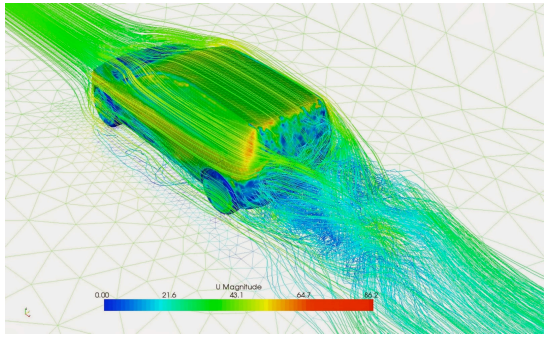




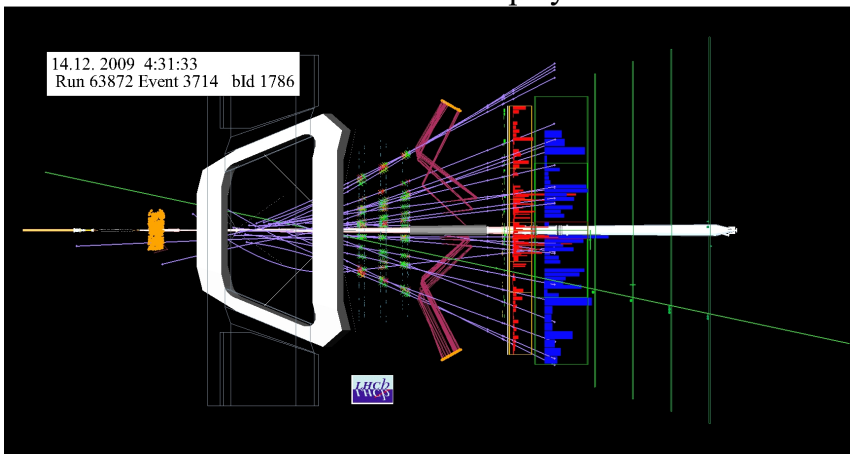




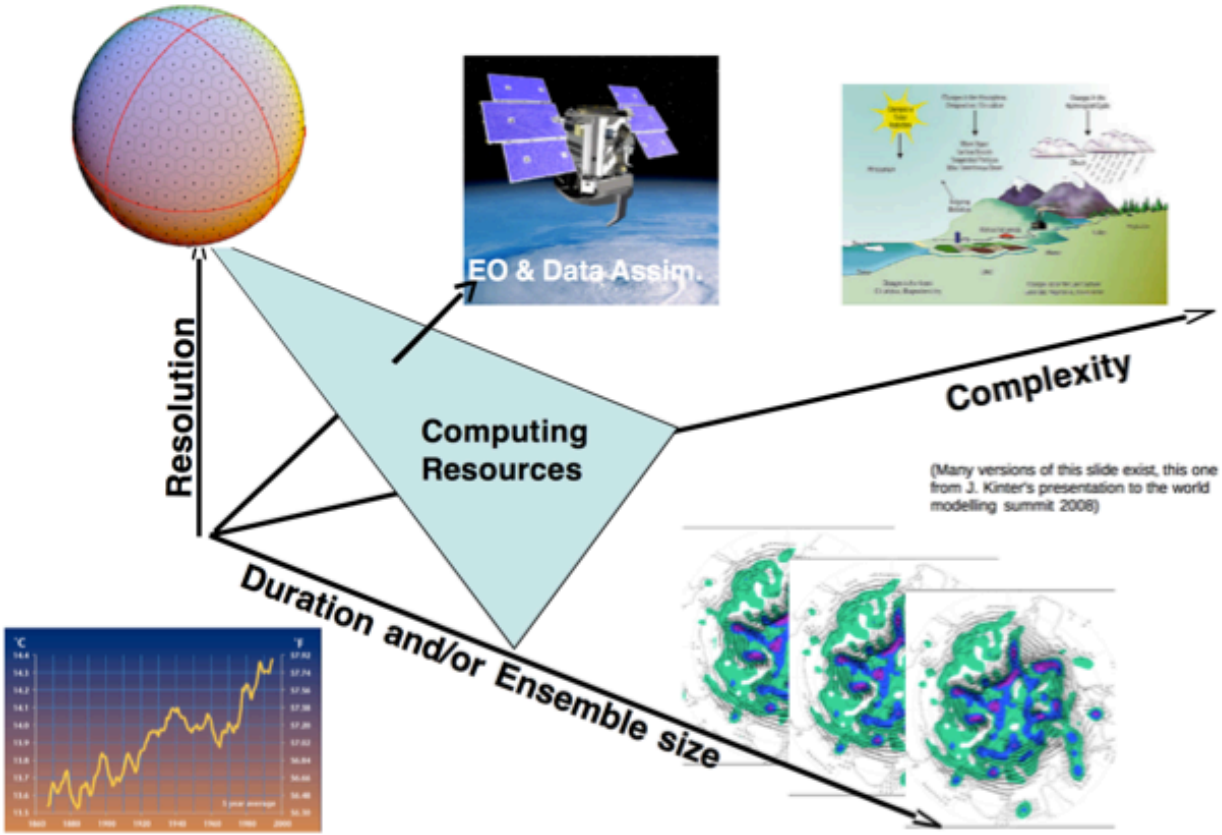




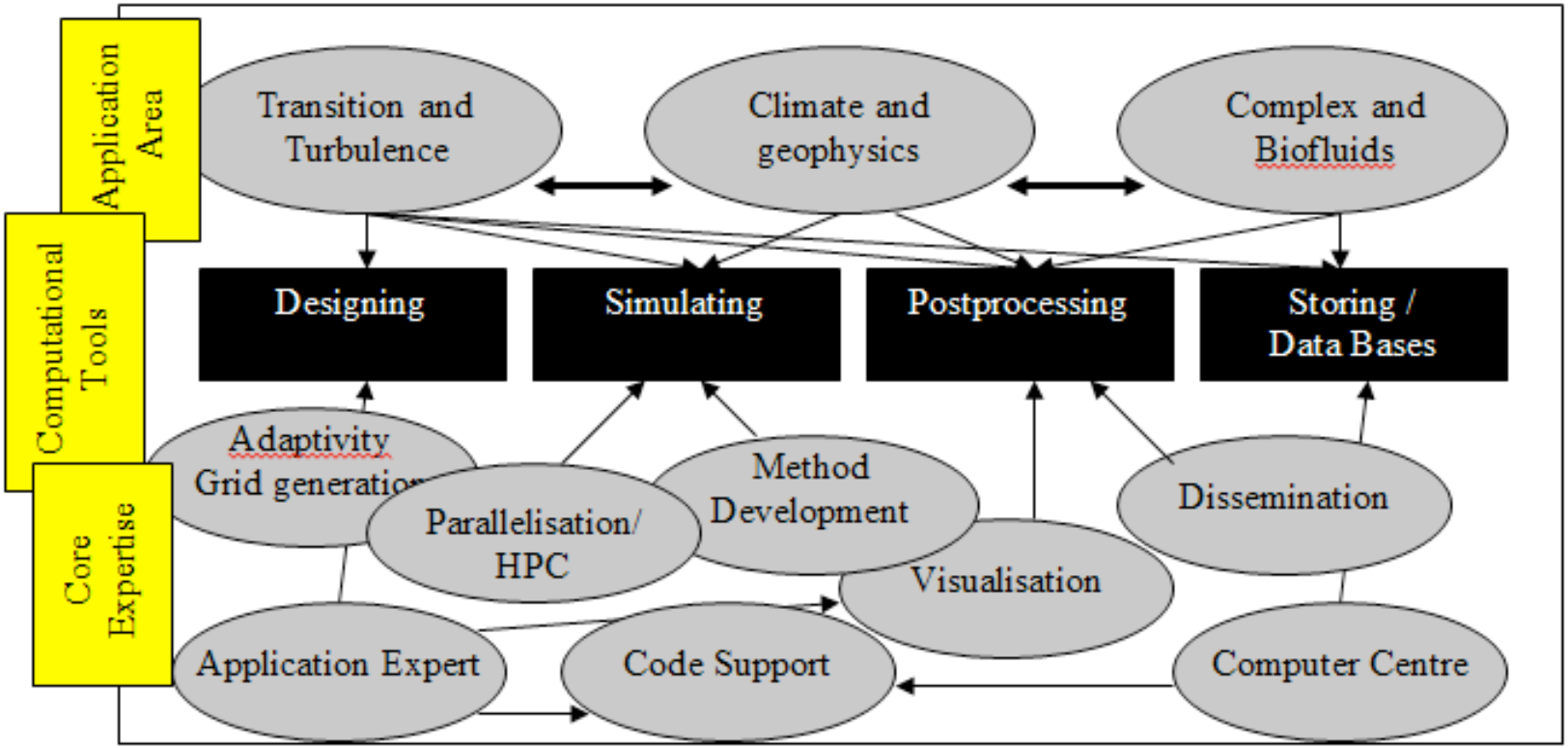
LHCb Event Display

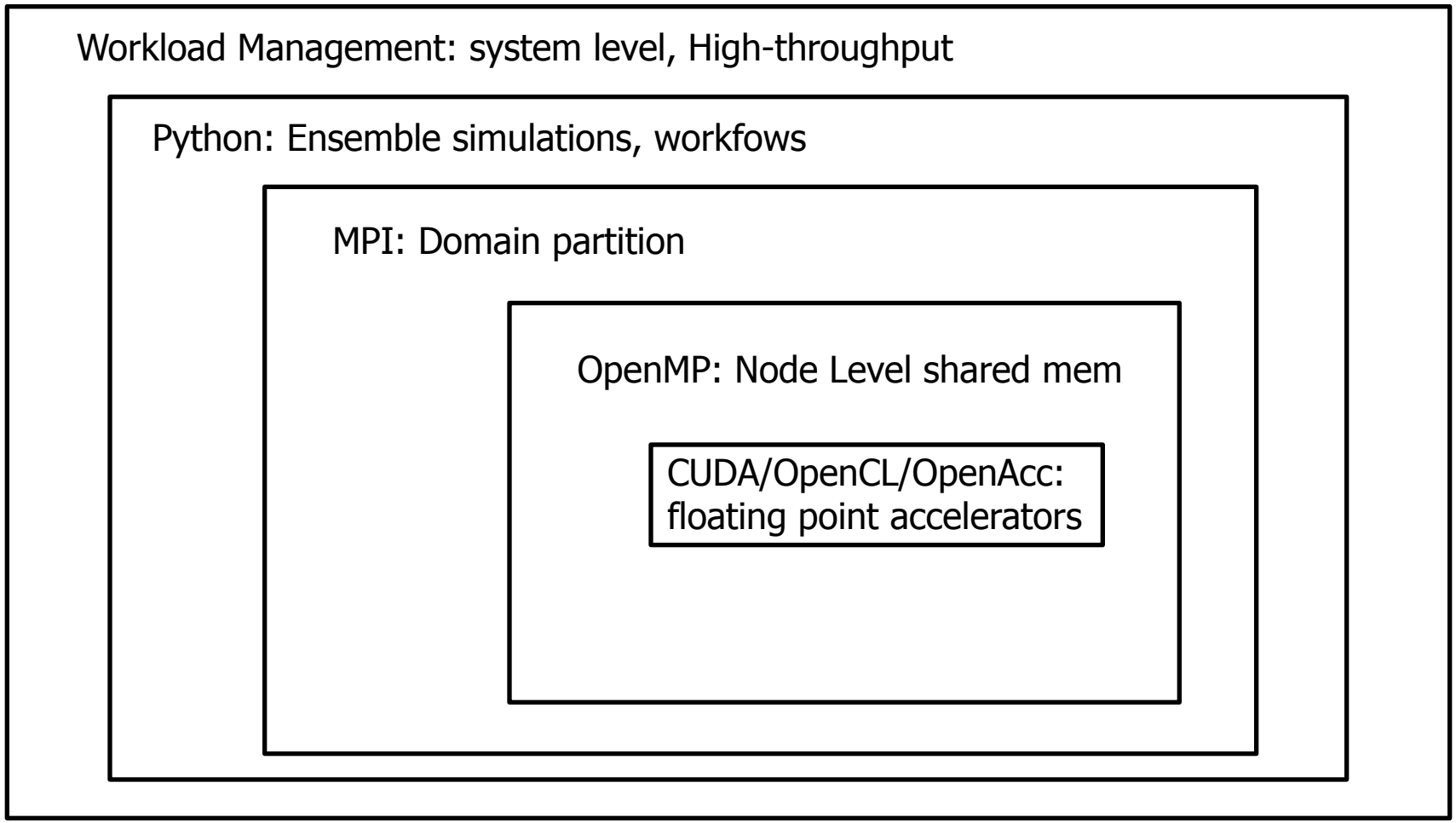


More & More Computing …



SW flow in science




Workload Management: system level, High-throughput
Python: Ensemble simulations, workfows
MPI: Domain partition
OpenMP: Node Level shared mem
CUDA/OpenCL/OpenAcc:
floating point accelerators

Conventional Software Development Process

- Start with set of requirements defined by customer (or management):
  - features, properties, boundary conditions
- Typical Strategy:
  - Decide on overall approach on implementation
  - Translate requirements into individual subtasks
  - Use project management methodology to enforce timeline for implementation, validation and delivery
- Close project when requirements are met

What is Different in the Scientific Software Development Process?

- Requirements often are not that well defined
- Floating-point math limitations and the chaotic nature of some solutions complicate validation
- An application may only be needed once
- Few scientists are programmers (or managers)
- Often projects are implemented by students (inexperienced in science and programming)
- Correctness of results is a primary concern, less so the quality of the implementation

Why Worry About This Now?

- Computers become more powerful all the time and more complex problems can be addressed
- Use of computational tools becomes common among non-developers and non-theorists
  - many users could not implement the whole applications that they are using by themselves
- Current hardware trends (SIMD, NUMA, GPU) make writing efficient software complicated
- Solving complex problems requires combining expertise from multiple domains or disciplines

About Software (Observations)

- Most research software is not of high quality
  - Typically written by graduate students:
    - without a good overview of existing software
    - with little software experience
    - with little incentive to write high quality code
  - Often maintained by postdocs:
    - with little time
    - need to consider software a tool to write papers
  - Advised by faculty
    - with no time
    - oftentimes also with little software experience
- How does this affect our field (Reproducibility? Archival? “Standing on the shoulders of giants”?)
- There is a complexity limit to what we can get out of a PhD student.

About Software

- Creating software is an art and science. So:
  - What makes software successful? (Best practices? Lessons learned?)
  - We could learn from the answers
- Use what others have already done (and use for free!):
  - Matlab
  - Linear algebra packages like PETSc, Trilinos
  - Finite element packages like libMesh, FEniCS, deal.II
  - Optimization packages like COIN, CPLEX, SNOPT, …
- On this, build only what is application specific
- Use software design principles

Ways to Move Forward

- Write more modular, more reusable software
  - build frameworks and libraries
- Write software that can be modified on an abstract level or where components can be combined without having to recompile
  - combine scripting with compiled code
- Write software where all components are continuously (re-)tested and (re-)validated
- Write software where consistent documentation is integral part of the development process

Embedded Scripting Language

- Not a new idea, but many scientific tools with scripting have their own "language"
  - script capability added on top of the tool
- Better to add domain specific extensions to an existing, generic scripting language:
  - use a language designed for scripting
  - can import other extensions, if needed
  - better documentation for script language
  - users may already know the syntax
- We will use Python in this workshop

Script Language Benefits

- Portability
  - Script code does not need to be recompiled
  - Platform abstraction is part of script library
- Flexibility
  - Script code can be adapted much easier
  - Data model makes combining multiple extensions easy
- Convenience
  - Script languages have powerful and convenient facilities for pre- and post-processing of data
  - Only time critical parts in compiled language

Modular Programming & Libraries

- Many tasks in scientific computing are similar
  - Tasks differ only in some subset of the calculation
  - Calculations use common operations like fast Fourier transforms (FFT), basic linear algebra, etc.
  - Data can be represented in a structured file format supported by generic analysis & visualization tools
- There is a large potential for code reuse
- Independent modules can be better validated
- Reusable code is better target for optimization

Object Oriented Programming

- Provide levels of abstraction
  - no need to know how something is done
  - opportunity to transparently optimize (for platforms, if certain conditions are given, etc.)
- Organize access to data
  - combine data with functions that modify it
  - control read-only vs. read-write access
  - handle side effects, on-demand computation
- Preserve APIs and favor local changes
  - modifying one part does not break others

Unit and Regression Testing

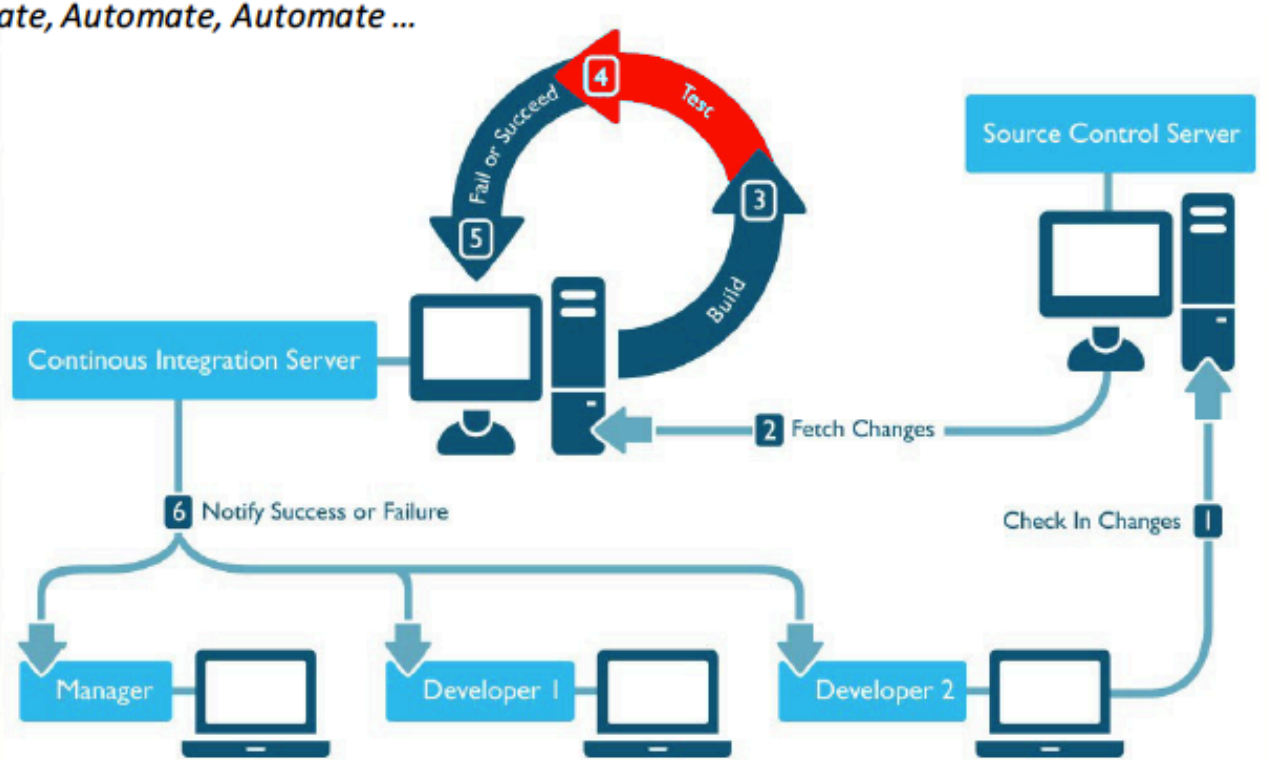
- Complex software cannot be fully tested, but
  - Many components can be tested individually
  - Testing of individual units is fast, can be automated
  - When testing individual units, you can also test for the correct handling of incorrect use or data
  - Failures in individual units may not always show up in testing the entire application for current use case
  - After fixing a bug, build minimal test case exposing the bug and add to a library of regression tests in order to keep it from reappearing

Importance of Tests and Validation

- With a larger user base comes responsibility
  - a test suite confirms available functionality
- No new code should break existing functionality
- Changes may have unintended side effects
- The more flexible a software is, the more potential for users to use it in unexpected ways
- Applications can fail on platforms due to broken compilers or system libraries
- Writing tests helps understanding a feature

Contiguous Integration

Automate, Automate, Automate ...



Embedded Documentation

- Three types of documentation needed
  - Information for developers who want to add code
    - Documentation of the API (e.g. via doxygen)
    - Comments in the code that explain choices
  - Information for users that want to use a feature
    - Reference manual for visible commands (can be automated and cross-linked with developer manual
  - Information for users that want to learn using a tool
    - write tutorials and HOWTO segments
    - often better written as standalone documents
  - It can be helpful to write documentation first

Source Code Management

- Not only a way to archive sources, but a tool for communication between developers
- Distributed source code management makes concurrent development easier
- Work with feature branches and merge often
- Commit changes in small increments and do not combine unrelated changes in on commit
- Have consistent, documented “whitespace rules” and best enforce them before committing

What makes such projects successful?

- Success or failure of scientific software projects is not decided on technical merit alone
- The true factors are beyond the code! It is not enough to be a good programmer! In particular, what counts:
  - Utility and quality
  - Documentation
  - Community
- All of the big libraries/packages provide this for their users.

The Bottom Line

- Many of these concepts and methods can help improve scientific software development
- Important: it is not the tools by themselves, but how they are used that makes the difference
- Fight the urge to take shortcuts and see the restrictions that modular and object oriented programming imposes as opportunities
- Finding the right balance is key to success
- Never underestimate the longevity of your code

Conclusions

- Computational science has spent too much time where everyone writes their own software.
- By building on existing, well written and well tested, software packages:
  - We build codes much faster
  - We build better codes
  - We can solve more realistic problems
- Scientific software development has to be recognized as a task requiring trained specialists and dedication of time and resources to produce dependable results

Thanks for your attention!!