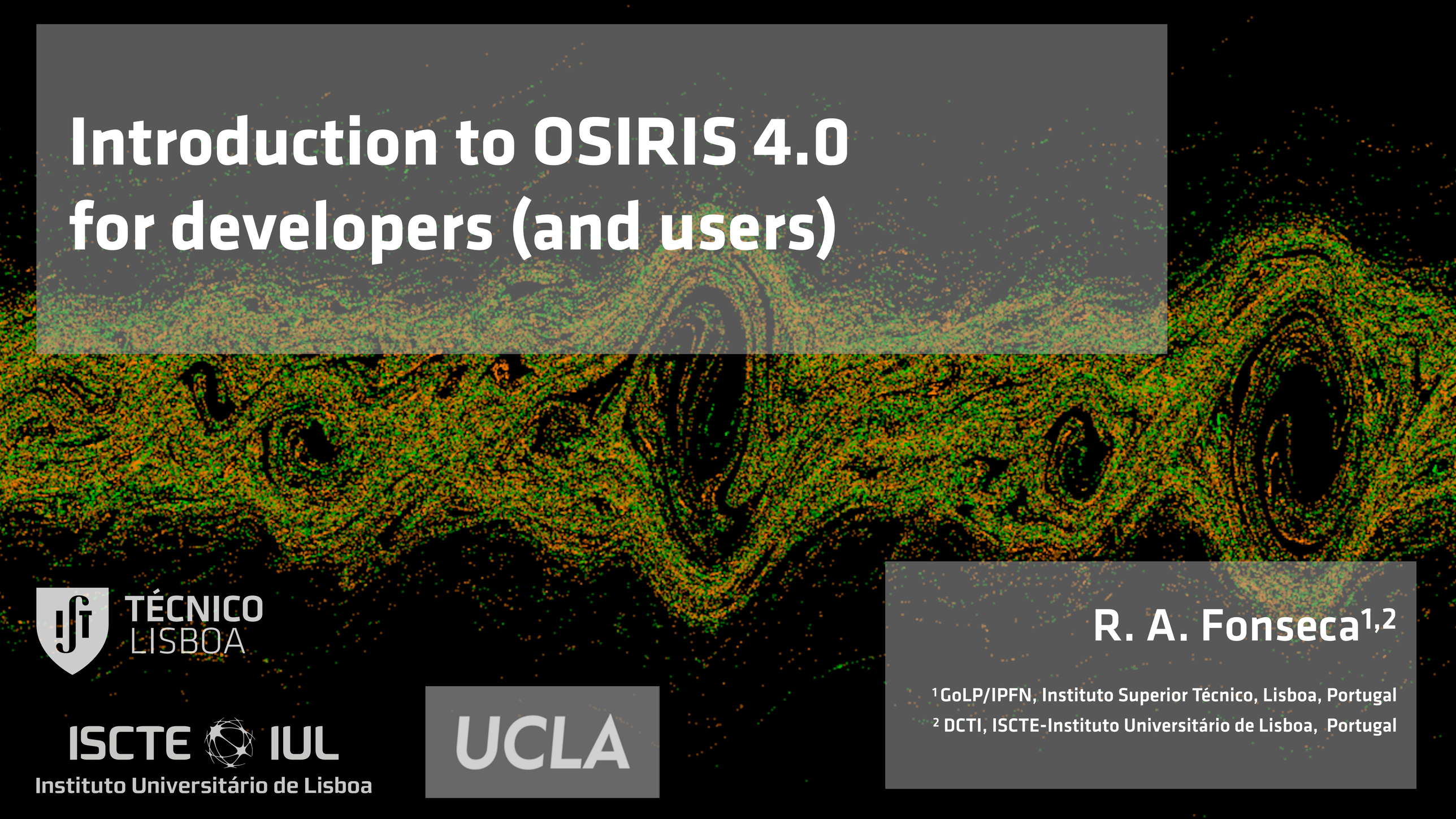
Introduction to OSIRIS 4.0 for developers (and users)



TÉCNICO LISBOA

ISCTE IUL
Instituto Universitário de Lisboa

UCLA

R. A. Fonseca[1,2]

[1] GoLP/IPFN, Instituto Superior Técnico, Lisboa, Portugal
[2] DCTI, ISCTE-Instituto Universitário de Lisboa, Portugal

The work of many people

IST

- J. L. Martins, T. Grismayer, J. Vieira, F. Gaudio, G. Inchingolo , P. Ratinho, K. Schoeffler, M. Vranic, U. Sinha, T. Mehrling, A. Helm, L. O. Silva

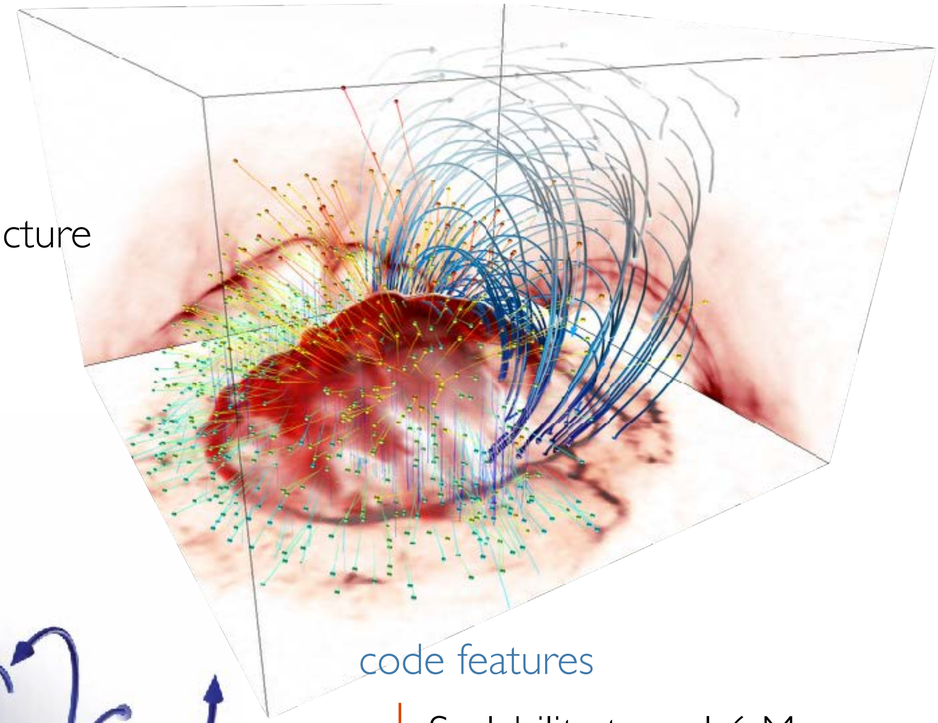
UCLA

- A. Tableman, A. Davidson, J. May, K. Miller, H. Wen, P. Yu, T. Dalichaouch, F. Tsung, V.K.Decyk, W. An, X. Xu, W. B. Mori

osiris 3.0

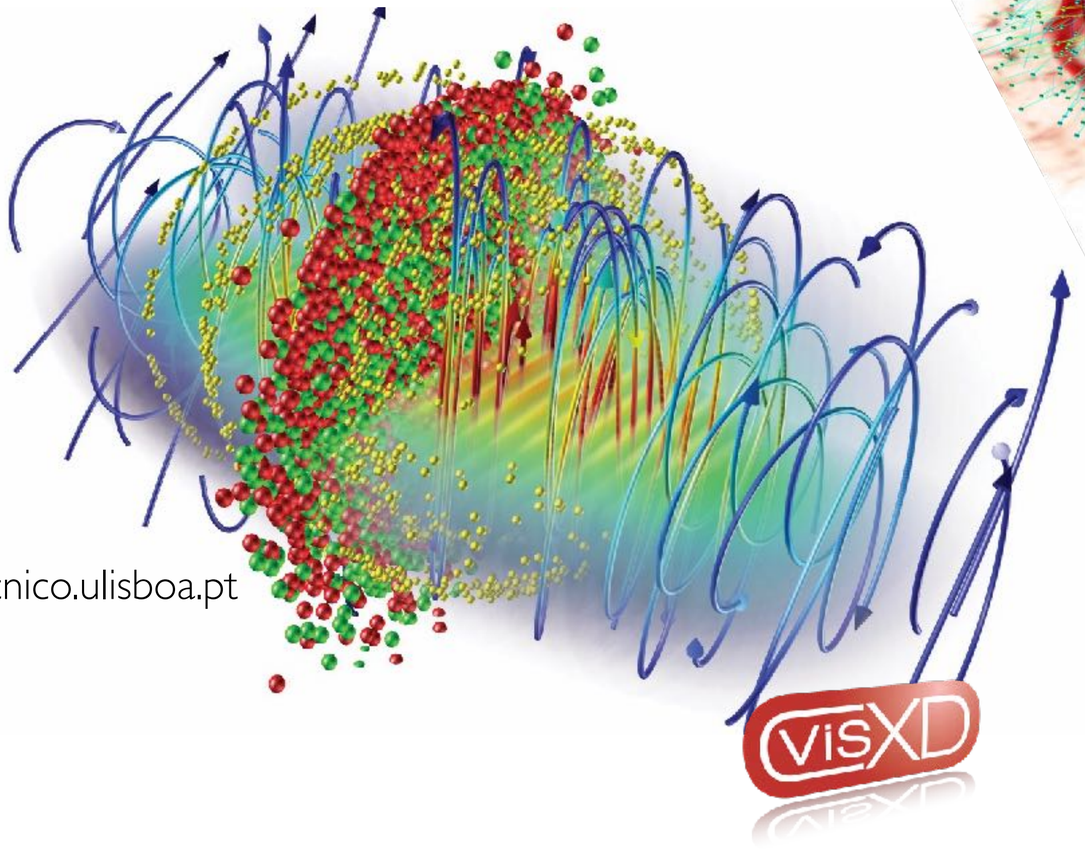
osiris framework

- Massivelly Parallel, Fully Relativistic Particle-in-Cell (PIC) Code
- Visualization and Data Analysis Infrastructure
- Developed by the osiris.consortium
  ⇒ UCLA + IST



TÉCNICO LISBOA

UCLA



Ricardo Fonseca: ricardo.fonseca@tecnico.ulisboa.pt
Frank Tsung: tsung@physics.ucla.edu

http://epp.tecnico.ulisboa.pt/

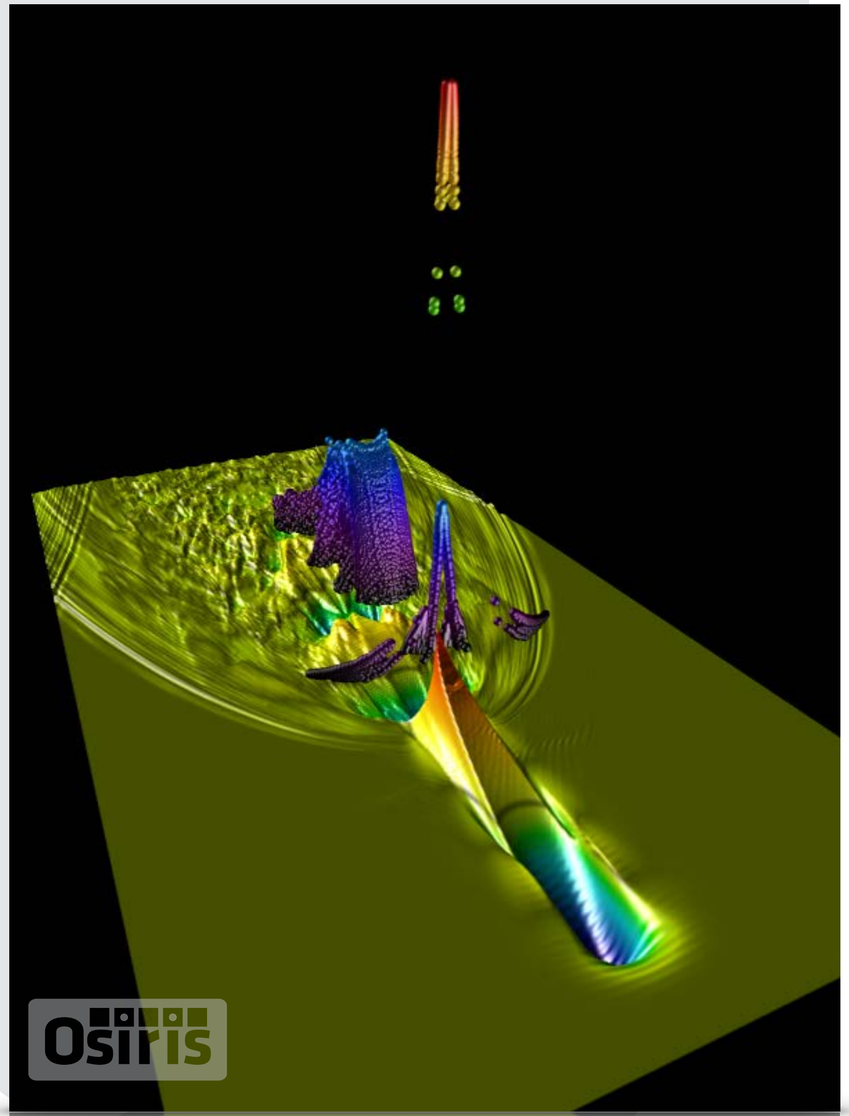
http://plasmasim.physics.ucla.edu/

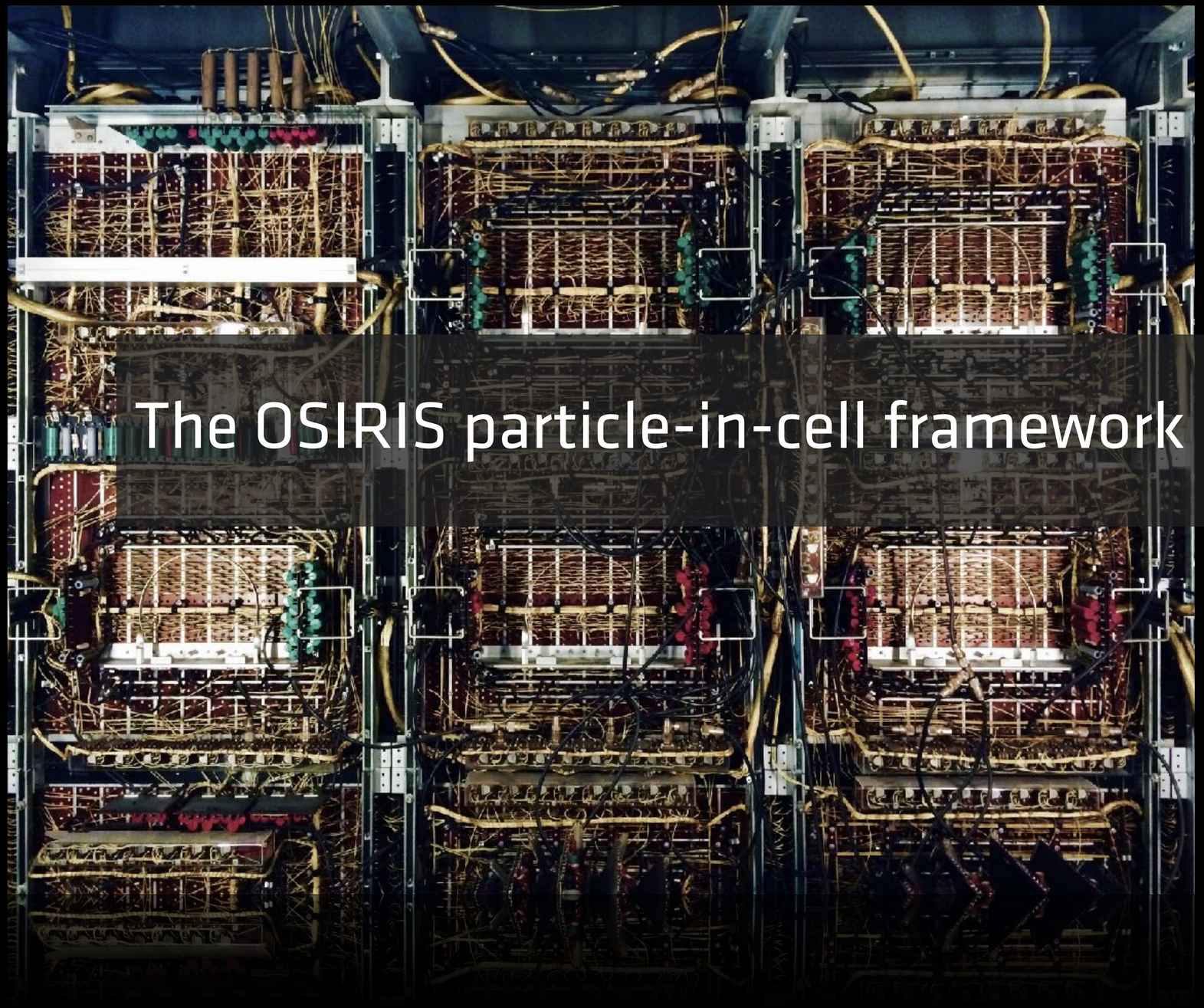
code features

- Scalability to ~ 1.6 M cores
- SIMD hardware optimized
- Parallel I/O
- Dynamic Load Balancing
- QED module
- Particle merging
- GPGPU support
- Xeon Phi support

Outline

- **The OSIRIS Particle-In-Cell infrastructure**
  - Overview of the OSIRIS framework
  - The need for a new structure
- **Moving into 4.0**
  - Main goals
  - New language features/techniques available
  - New file structure and features
- **Developing a new simulation mode**
  - The new simulation class
  - Adding new features to the code
- **Managing the code**
  - Flash introduction to our GitHub repository
- **Overview**





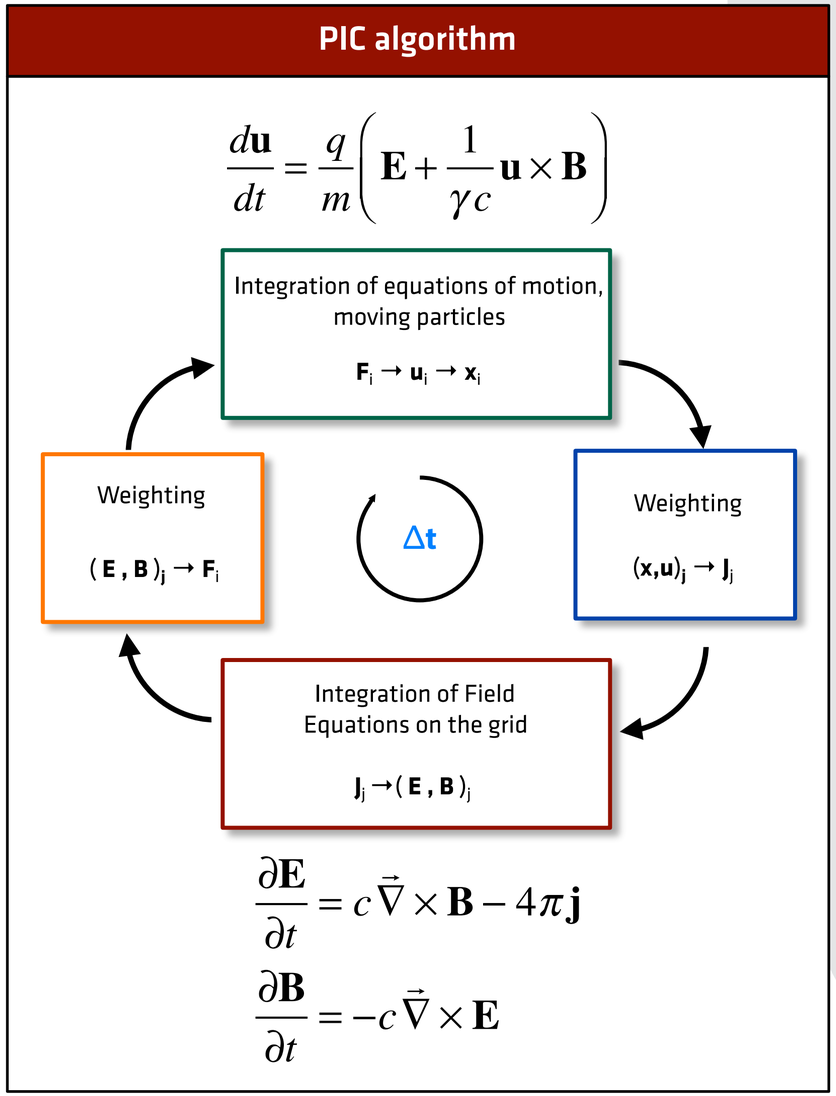
The OSIRIS particle-in-cell framework

TÉCNICO LISBOA

UNIVAC 1 - 1951

Internal view

The OSIRIS framework



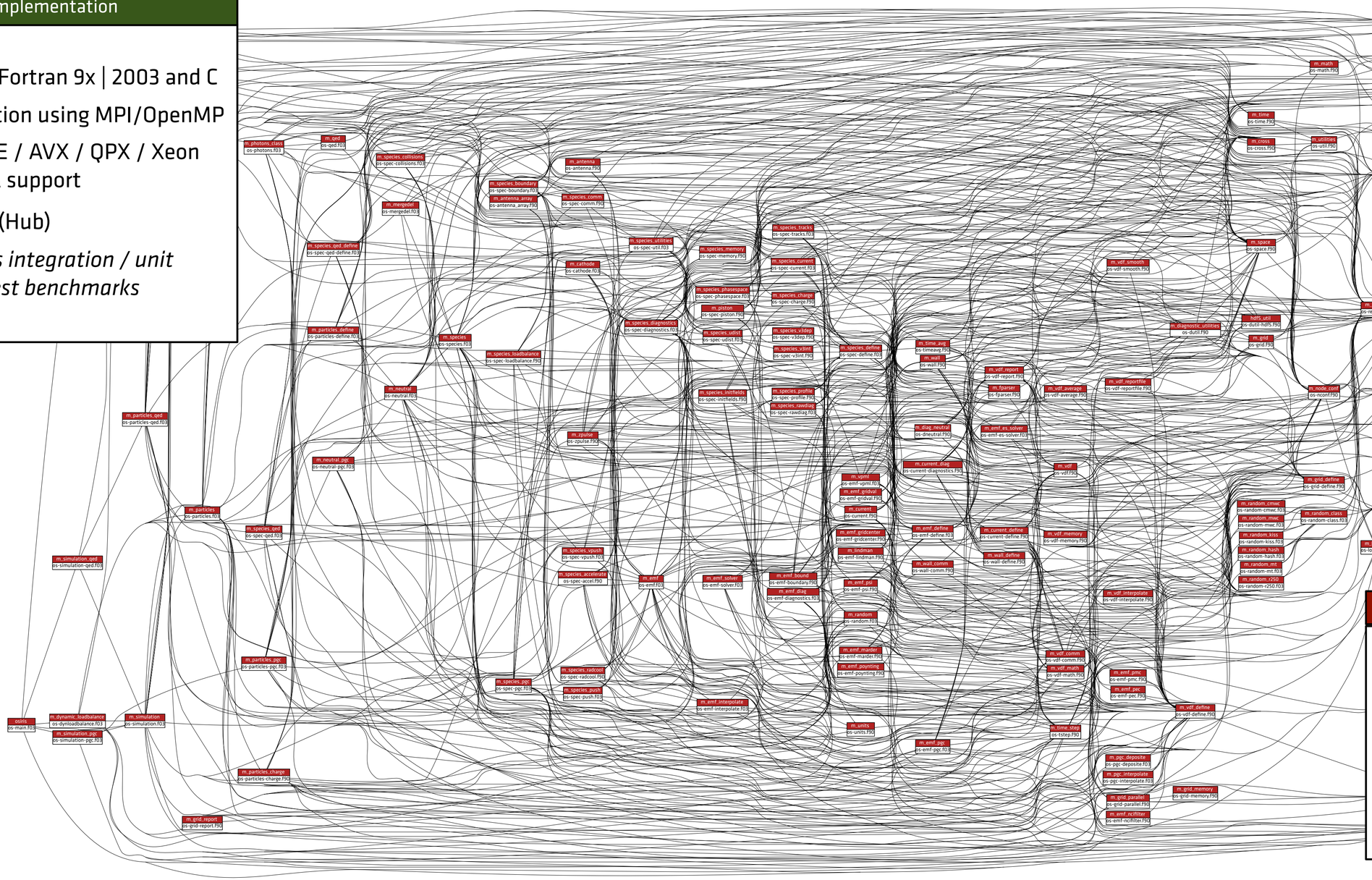
- **Fully Relativistic, Electromagnetic Particle-In-Cell Code**
  - Massively Parallel
  - Dynamic Load Balancing
  - High-order particle interpolation
  - Multiple field solvers / particle pushers
  - Particle merging
  - Advanced Hardware Support
- **Extended Diagnostic Capabilities**
  - Griding of arbitrary particle quantities
  - Spatial / Temporal averaging of grid quantities
  - Particle Selection / Tracking
  - Tight integration with visXD
- **Additional Simulation Modes / Physics models**
  - ADK tunnel / impact ionization
  - Binary collisions
  - High-density hybrid
  - Radiation Cooling | QED module
  - *Ponderomotive guiding center*
  - *Quasi-3D geometry*

Osiris 3.0

OSIRIS source tree

Implementation

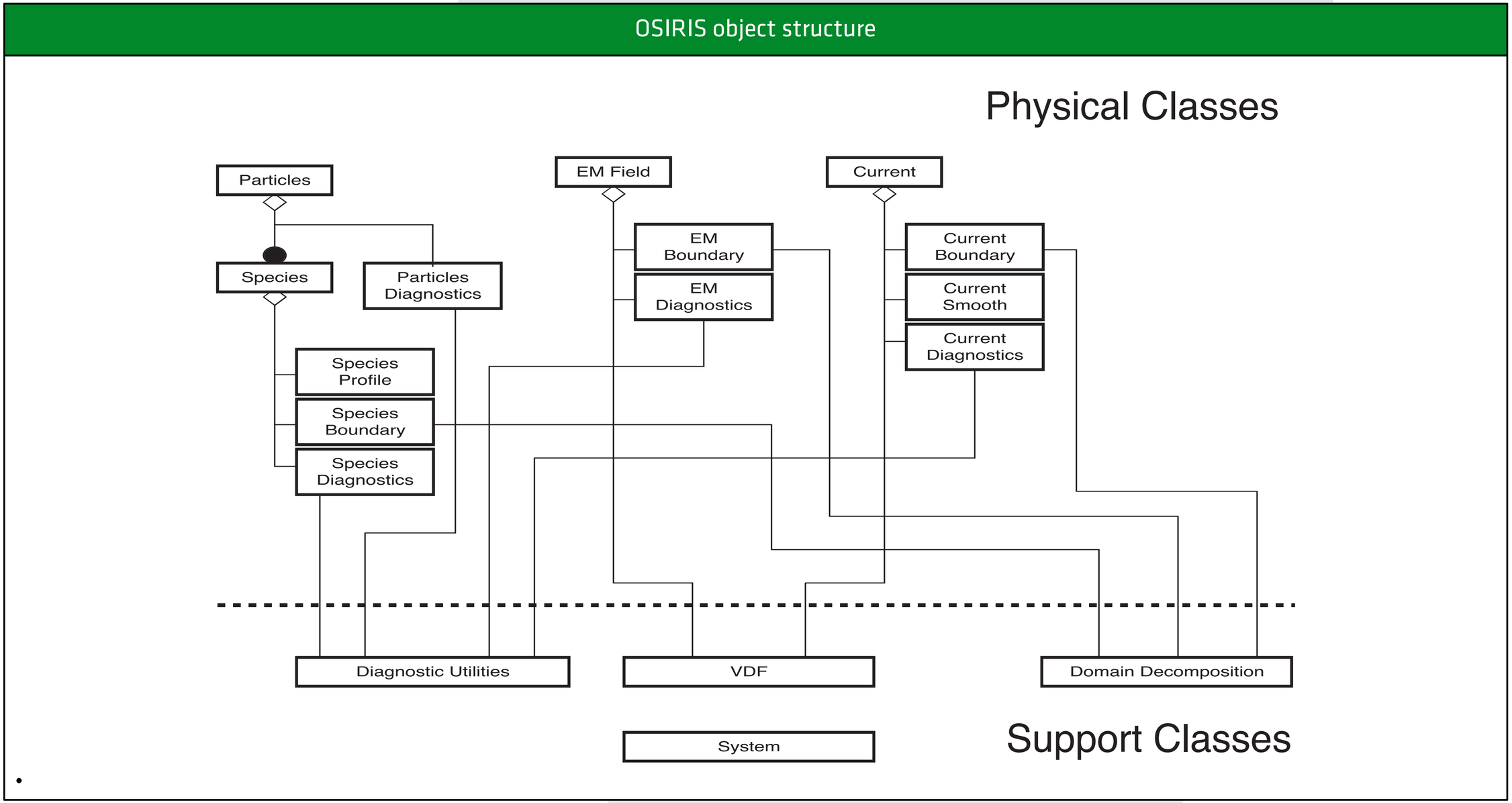
- Written in Fortran 9x | 2003 and C
- Parallelization using MPI/OpenMP
- Explicit SSE / AVX / QPX / Xeon Phi / CUDA support
- SVN → Git(Hub)
- *Continuous integration / unit testing / test benchmarks*



OSIRIS in numbers

- Start year: 1998
- 4480852 characters
- 143310 lines
  - 118289 (Fortran 95|03)
  - 19249 (C)
- 187 source files

Old OSIRIS object structure



OO concepts implemented in Fortran95

- No real inheritance
  - Use derived types
  - Include superclass as structure member
- No function pointers / virtual methods
  - Run-time options selected through conditional statements

```
type t_photons

  ! parent class
  type( t_species ) :: t_species
  (…)

  ! number of pairs created
  integer :: num_pairs = 0
  (…)
end type t_photons

(…)

subroutine update_boundary_phot( this, no_co, dt )

  type( t_photons ), intent(inout) :: this
  (…)

  ! Call superclass method
  call update_boundary( this%t_species, no_co, dt )

  ! Do additional work if needed
  (…)

end subroutine update_boundary_phot
```

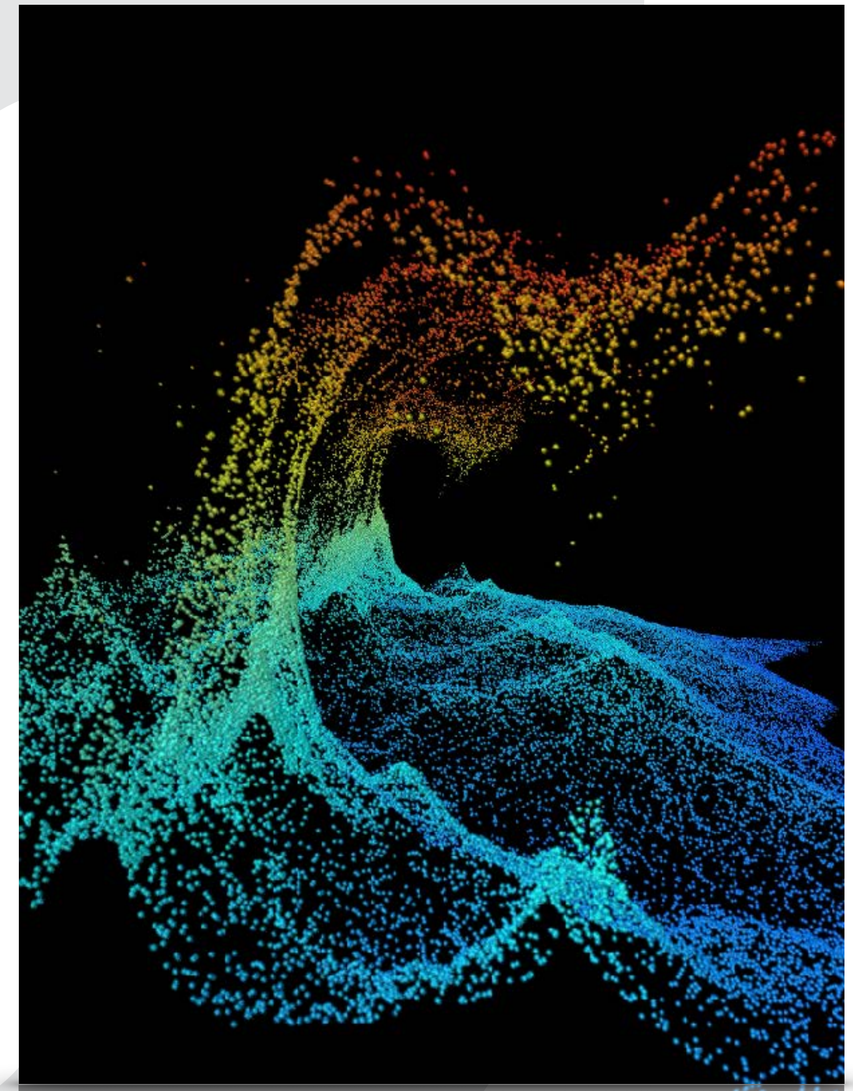
- Difficult to maintain and expand
  - Changes in one module affects all code
  - Some options only available at compile time

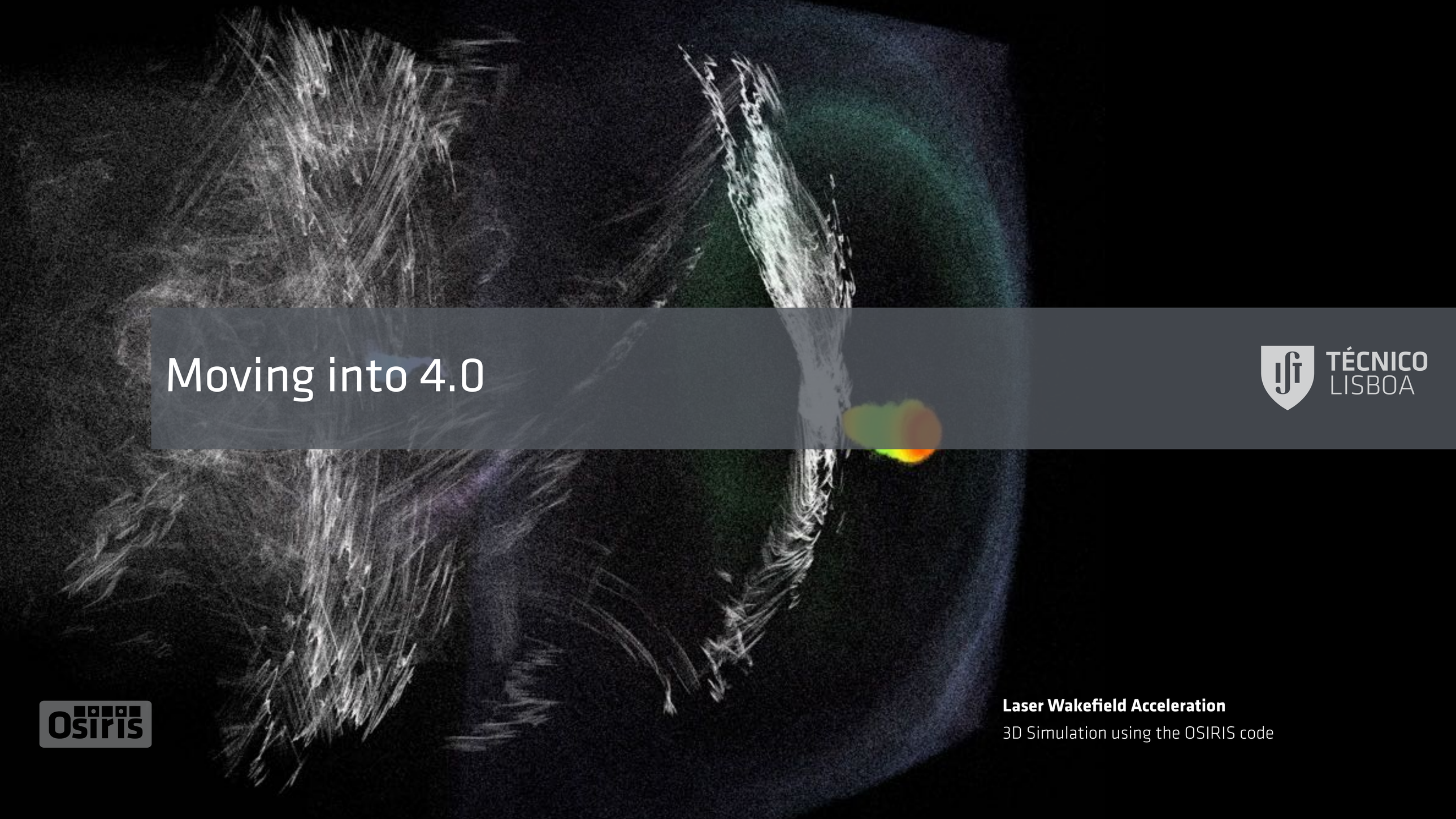
```
subroutine push_species( this, emf, jay, … )

  type( t_species ), intent(inout) :: this

  ! (…)

    select case ( this%push_type )
      case ( p_std )
        call advance_deposit( this, emf, jay, … )

      case ( p_pgc )
         call advance_deposit_emf_pgc( this, emf, jay, … )

      case ( p_qed )
         call advance_deposit_qed( this, emf, jay, … )

  ! (…)

    end select

  ! (…)

end subroutine push_species

subroutine advance_emf( this, jay, … )

  type( t_emf ), intent( inout )  ::  this

  ! (…)
  if ( this%use_pgc ) then
      call advance_pgc( this , dt , coordinates , no_co )
  endif

end subroutine advance_emf
```

Why move to a new code structure?

- **Main issue - Adding new features**
  - This generally meant changing the main code and critical routines
  - At best these changes made the code more complicated
  - At worst they impacted on the performance of the normal simulation modes and could cause it to crash

- **Additional issues**
  - A growing portion of the source code is now written in C, and Fortran C interoperability had to be done by ourselves, and updated for each new compiler/system
  - Some routines are very similar and would benefit from template based code generation
  - Flat file hierarchy was becoming difficult to manage





Moving into 4.0

TÉCNICO LISBOA

OSIRIS

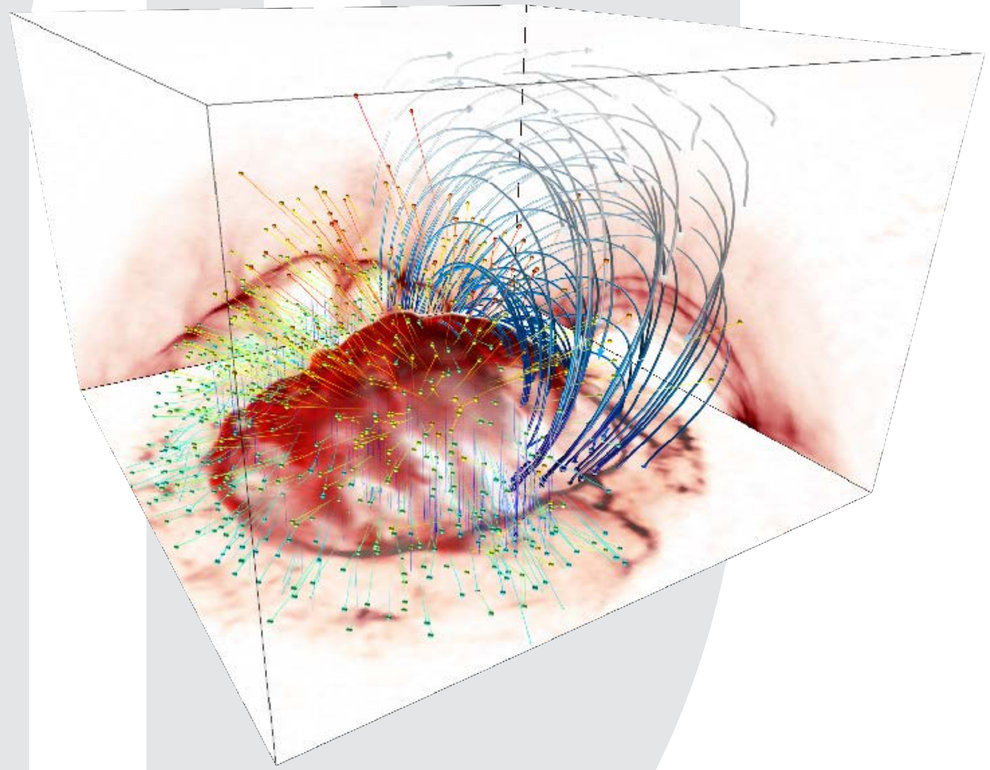
Laser Wakefield Acceleration

3D Simulation using the OSIRIS code

Main goals

- **Allow for the development of new features with minimal impact to the rest of the code**
  - Only change 1 file in the main code structures
  - New features are encapsulated inside a separate folder
  - Easy to remove from code, encouraging new developers and simplifying maintenance

- **Modernize code structure by taking advantage of new language features**
  - Clean up base code structure removing references to other sim. modes
  - Improve interface with C code
  - Replace similar routines with template based programming



Fortran 2003

Features highlight

- Object-oriented support
  - Type extension and inheritance
  - Polymorphism
  - Dynamic type allocation
  - Type-bound procedures (methods)
- Procedure pointers
- Standardized interoperability with C
- Enhanced Integration with the host operating system
- There is finally (2016) a widespread support of most of these features

Fortran 2003 Standard, final draft (ISO/IEC 1539-1:2004(E))

- **Simplify code structure**
  - All object oriented concepts can now be expressed in a more natural way
  - Separate different simulation modes into different object
- **Improve selection of runtime options**
  - No need for conditional code execution
- **Simplify Fortran-C interface**
  - Easier interface to SIMD hardware accelerated code
- **Remove most of POSIX interface module**
  - Improve portability

Compiler Support for Fortran 2003

Compiler	OSIRIS F2003 support
GNU gfortran 5.x	full
GNU gfortran 6.x	full
Intel ifort 2016, 2017, 2018	full
Intel ifort 2015	*incomplete*
Intel ifort 2013sp1	*incomplete*
PGI pgfortran 15.10	*incomplete*
IBM xlf2003 14.1 BG/Q	full

Although the F2003 standard has been around for a while, many compilers don't fully support it

- **Using only a subset of F2003 features**
  - Lowest common denominator approach
- **Only 1 feature missing in some compilers**
  - Using type bound procedures that are not defined in the same module as the class definition (through the use of explicit interfaces)
- **Workaround for compilers missing this feature**
  - Defining type bound procedures in the same module as the class definition that just call the routines defined in other files
  - Done through a preprocessor macro at compile time
- **With this workaround all compilers listed successfully compiled and ran the code**

Template based code generation

- **Fortran 2003 does not fully support template based code generation**
  - Does support parameterized derived types
- **Some routines are very similar**
  - Writing / maintaining all versions is error prone
  - Using code templates greatly simplifies code and minimizes errors
- **Can be implemented using a preprocessor**
  - C preprocessor can be applied to Fortran files before calling the Fortran compiler
  - Most Fortran compilers are compatible with C preprocessor output
    - Others need preprocessor output to be sanitized before compilation
- **Repeat for each variant**
  - Define new function name
  - Define template parameters
  - Include template code

template.f03

```
function FNAME(x)
   PTYPE :: x, FNAME
   FNAME = x*x*x
end function FNAME
```

generator.f03

```
#define FNAME cube_real
#define PTYPE real
#include “template.f03”

#define FNAME cube_cmplx
#define PTYPE complex
#include “template.f03”

#define FNAME cube_dbl
#define PTYPE double precision
#include “template.f03”
```

Example: Charge deposition

Generate versions for interpolation levels 1 to 4

deposit.f03

```
! Linear interpolation
#define DEPOSIT_RHO_2D   deposit_rho_2d_s1
#define SPLINE spline_s1
#define LP 0
#define UP 1
#include "template.f03"

! Quadratic interpolation
#define DEPOSIT_RHO_2D   deposit_rho_2d_s2
#define SPLINE spline_s2
#define LP -1
#define UP 1
#include "template.f03"

! Cubic interpolation
#define DEPOSIT_RHO_2D   deposit_rho_2d_s3
#define SPLINE spline_s3
#define LP -1
#define UP 2
#include "template.f03"

! Quartic interpolation
#define DEPOSIT_RHO_2D   deposit_rho_2d_s4
#define SPLINE spline_s4
#define LP -2
#define UP 2
#include "template.f03"
```

template.f03

```
subroutine DEPOSIT_RHO_2D( rho, ix, x, q, np )

  implicit none

  type( t_vdf ), intent(inout) :: rho
  integer, dimension(:,:), intent(in) :: ix
  real(p_k_part), dimension(:,:), intent(in) :: x
  real(p_k_part), dimension( : ), intent(in) :: q
  integer, intent(in) :: np

  ! local variables
  integer :: l, i1, i2, k1, k2
  real(p_k_fld) :: dx1, dx2, lq
  real(p_k_fld), dimension(LP:UP) :: w1, w2

  do l = 1, np

    i1 = ix(1,l)
    i2 = ix(2,l)
    dx1 = real( x(1,l), p_k_fld )
    dx2 = real( x(2,l), p_k_fld )
    lq  = real( q(l) , p_k_fld )

    ! get spline weitghts for x and y
    call SPLINE( dx1, w1 )
    call SPLINE( dx2, w2 )

    ! Deposit Charge
    do k2 = LP, UP
      do k1 = LP, UP
        rho%f2(1,i1 + k1,i2 + k2) = rho%f2(1,i1 + k1,i2 + k2) + &
                                    lq * w1(k1)* w2(k2)
      enddo
    enddo

  enddo

end subroutine DEPOSIT_RHO_2D
```

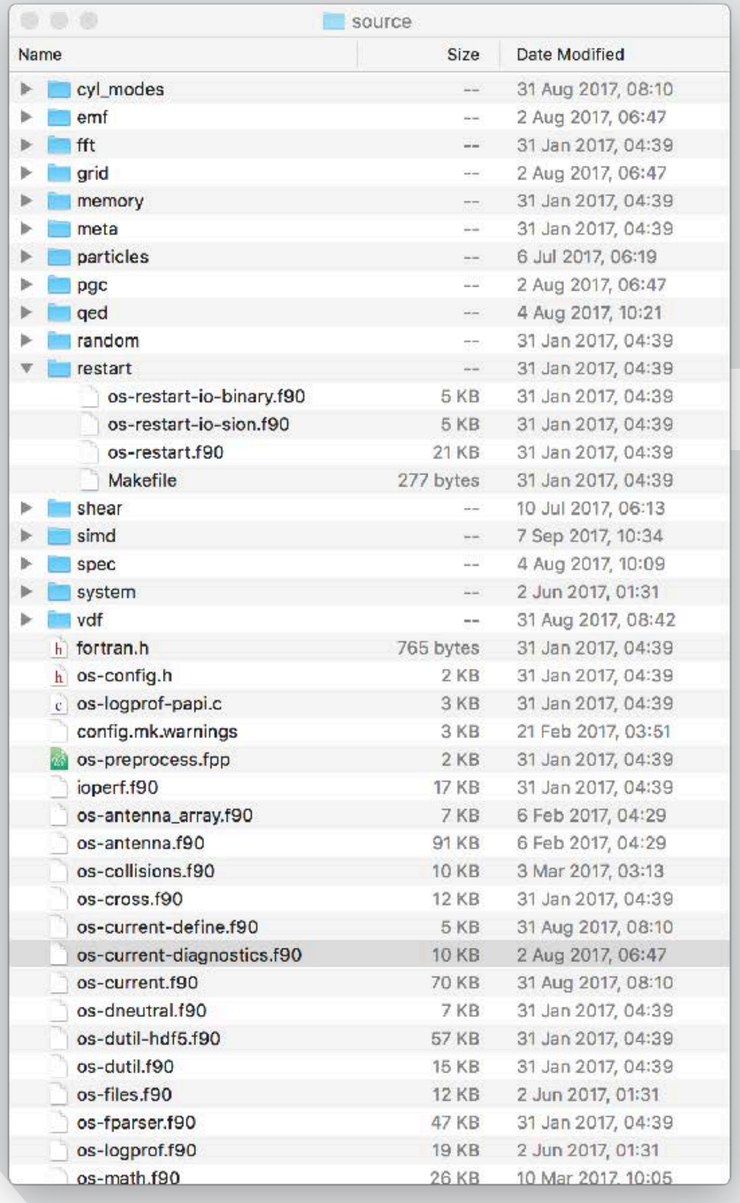
New source tree structure

- **OSIRIS 3.x used a flat source tree**
  - 145 files in the “source” folder
  - Single Makefile

- **OSIRIS 4.x implements an actual source tree**
  - Use 1 folder for every class / group
  - Each folder has its own Makefile
  - Nested folders are possible
  - Migration from 3.x structure not yet completed

  - *Limitation*: source file names must be different, even if they reside in different directories

- **Avoid adding files to the source root**
  - Major features should create a new directory



Organized by folders

- Files related to the same class are together

Separate Makefiles

- Each folder has its own Makefile

Current stats

- 17 folders
- 224 files

New random number generator module

- **Multiple (pseudo) random number generation algorithms**
  - Mersenne-Twister (default)
  - R250
  - Marsaglia MWC (multiply with carry)
  - Marsaglia CMCW (complimentary multiply with carry)
  - Hashing RNG (from Numerical Recipes)
  - Marsaglia KISS
- **Can be selected in the input file**
  - Re-run simulations with different RNG/seed
- **Available as an object for developers**
  - Devs. can use multiple separate RNGs instead of global RNG

osiris input file

```
! Global options
simulation
{
  random_algorithm = "r250",
  random_seed = 9262,
}
```

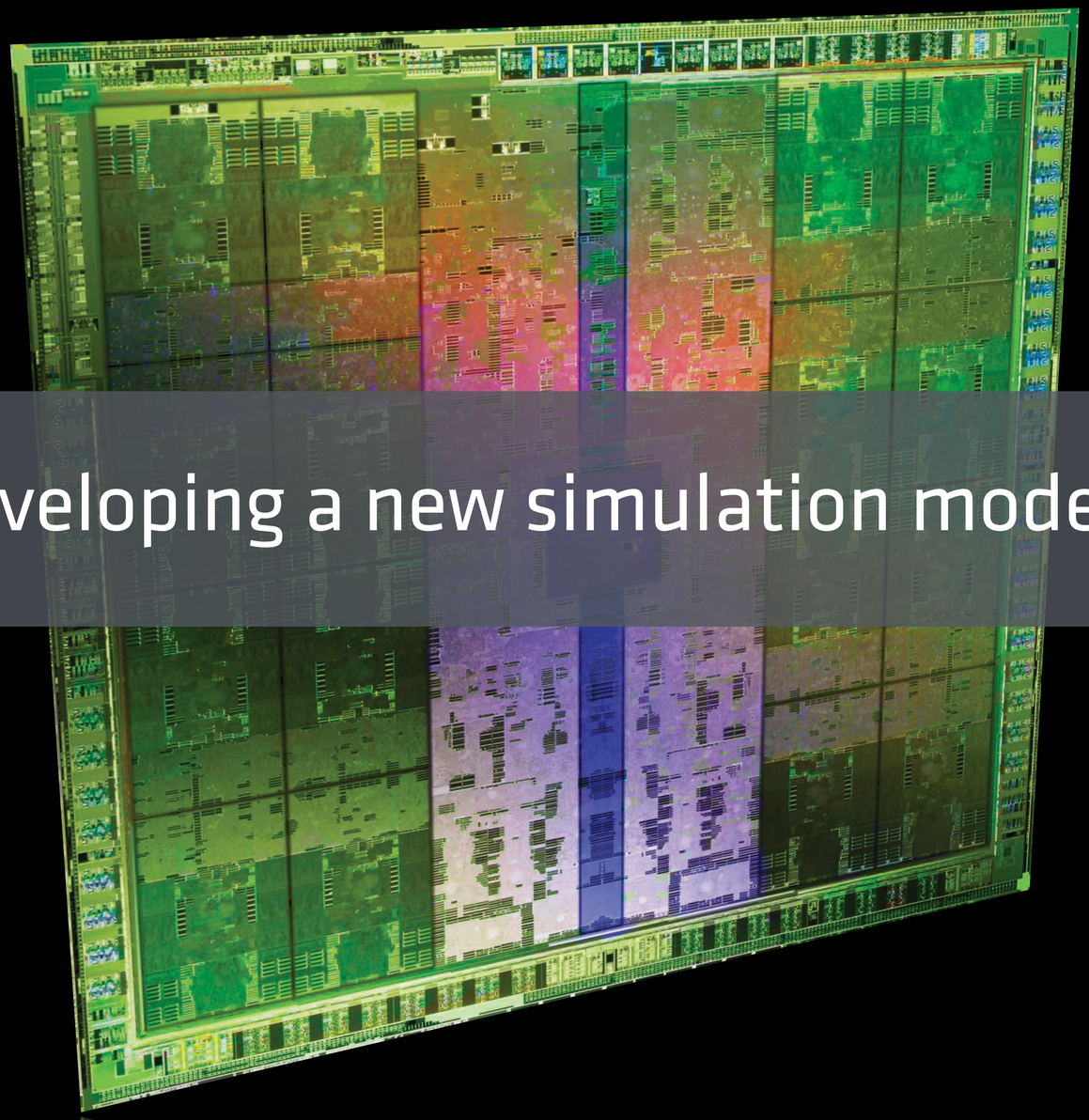
```
! use local RNG

class(t_random_hash) :: local_rng

call local_rng % init_genrand_scalar( seed )

dice = local_rng % genrand_res53()
```

Input file structure remains mainly the same

- **The global structure remains unaltered**
  - The initial simulation section has added relevance
  - *algorithm* parameter defines the type of simulation
- **Different simulation modes may change structure**
  - Remove existing sections
  - Add new ones
  - Change allowed parameters
- **Some sections have changed**
  - Check documentation
  - Only minor changes required to most existing input files

```
simulation
{
  algorithm = "pgc",                    ! Use the Ponderomotive Guiding center
}


node_conf
{
  node_number(1:2) =  2, 2,             ! this is a 4 node run
  if_periodic(1:2) = .true., .true.,    ! using periodic boundaries

  n_threads = 2,                        ! and 2 threads per node
                                        ! (so it should use 8 cores total)
}


!----------spatial grid----------
grid
{  nx_p(1:2) = 128, 128, }            ! we use a 128 x 128 grid


!----------time step and global data dump timestep number----------
time_step
{
  dt      =   0.07,    ! the time step will be 0.07

  ndump   =  1,        ! and the global dumps are set for every
                       ! iteration
}

!----------spatial limits of the simulations----------
space
{
  xmin(1:2) =   -6.4 , -6.4,  ! setup a 12.8 x 12.8 global simulation
                              ! space
  xmax(1:2) =    6.4 ,  6.4,  !  no moving window specified
}


!----------time limits ----------
time
{
  tmin = 0.0, tmax  = 70.0,  ! run up to t = 70.0
}


el_mag_fld
{
  solver =   "yee",     ! default (…)
```

Developing a new simulation mode



NIVIDIA Fermi K20x die

TÉCNICO LISBOA

OSIRIS 4.0 *t_simulation* class

- All simulation objects are contained in a global **t_simulation** object
- This class defines general data structures and methods for a simulation
  - All of these can be overridden by subclasses that implement different simulation modes
- Includes both physical and support classes

- Implements the standard EM-PIC algorithm
  - also considered making it a virtual base class

- Design goals
  - Keep most of existing code-base
  - Allow easy extension and encapsulation of code

```
type :: t_simulation

   type( t_node_conf )        ::  no_co
   type( t_grid )             ::  grid
   type( t_time_step )        ::  tstep
   type( t_restart   )        ::  restart
   type( t_options )          ::  options

   type( t_space           )  ::  g_space
   type( t_time            )  ::  time
   type( t_zpulse_list     )  ::  zpulse_list
   type( t_antenna_array   )  ::  antenna_array

   class( t_current      ), pointer    ::  jay => null()
   class( t_emf          ), pointer    ::  emf => null()
   class( t_particles    ), pointer    ::  part => null()

contains

   procedure :: iter                => iter_sim

   procedure :: allocate_objs       => allocate_objs_sim
   procedure :: init                => init_sim
   procedure :: cleanup             => cleanup_sim
   procedure :: read_input          => read_input_sim

   procedure :: write_checkpoint    => write_checkpoint_sim
   procedure :: list_algorithm      => list_algorithm_sim
   procedure :: test_input          => test_input_sim

end type t_simulation
```

t_simulation methods

Method	Functionality
iter	Implements PIC algorithm iteration: particle/field advance, boundary conditions (including parallel comms.), diagnostics
allocate_objs	Allocates required member objects: *jay*, *emf*, *particles*
init	Initialization code (includes initializing from checkpoint)
cleanup	House cleaning
read_input	Reads in the input file
write_checkpoint	Writes checkpoint information
list_algorithm	Lists algorithm details
test_input	Tests simulation parameters for validity (Courant condition / parallel partition

All of these methods can be overridden

Executing the main program

os-main.f03

```
1  call timer_init()                                                          ! Initialize timers
2  call init_options( opts )                                                  ! Initialize runtime options and store wall clock time
3  call system_init( opts )                                                   ! Initialize system code (this also initializes MPI)
4  call init_mem()                                                            ! Initialize dynamic memory system
5  call print_init_banner()                                                   ! Print Initialization banner
6  call initialize( opts, sim )                                               ! Read input file and setup simulation structure
7  if ( root(sim%no_co) ) then
8      call sim%list_algorithm()                                              ! Print main algorithm parameters
9  endif
10 call status_mem( comm( sim%no_co ) )                                       ! Print dynamic memory status
11 call run_sim( sim )                                                        ! Run simulation
12 call list_total_event_times( n(sim%tstep), comm(sim%no_co), label = 'final' ) ! Write detailed timing information to disk
13 call sim%cleanup( )                                                        ! Cleanup and shut down
14 deallocate( sim )
15 if ( mpi_node() == 0 ) then                                                ! Print final message
16     print *, 'Osiris run completed normally'
17 endif
18 call finalize_mem()                                                        ! Finalize dynamic memory system
19 call system_cleanup()                                                      ! this calls mpi_finalize
```

Creating a new simulation mode

- **Create a subclass of t_simulation**
  - Add additional member variables (if needed)
  - Add additional methods (if needed)
  - Override existing methods (if needed)
  - In particular *allocate_objs*() will generally need to be overridden
    - Allows for the allocation of different types of *emf*, *jay* or *particles* objects
- **Modify os-main.f03**
  - Add use statement for new module
  - Modify *read_sim_options*() to recognize new input file option
  - Modify *initialize*() routine to allocate simulation objects of the new class
- **Modify Makefile**
  - Include the Makefile for the new simulation mode
- **Remember to put all new code in a separate folder**

```
type, extends( t_simulation ) :: t_simulation_new
    integer :: extra
contains
    procedure :: allocate_objs => allocate_objs_new
end type t_simulation_new
```

```
subroutine allocate_objs_new( sim )
   use m_particles_new
   implicit none
   class( t_simulation_new ), intent(inout) :: sim

   SCR_ROOT('Allocating NEW particle objects.')
   allocate( t_particles_new :: sim%part )

   ! allocate any remaining objects in superclass
   call sim % t_simulation % allocate_objs()

end subroutine allocate_objs_new
```

Run-time selection of simulation mode

new osiris input file

```
! Global options
simulation
{
  algorithm = "new",
}

node_conf
{
  node_number(1:2) =  1,1,
! (…)
```

initialize(opts,sim) : os-main.f03

```
! Read global options
call read_sim_options( opts, input_file )

! Allocate simulation object based on algorithm
select case ( opts%algorithm )
    case( p_standard )
      ! Standard EM-PIC simulation
      allocate( t_simulation :: sim )

    case( p_sim_new )
      ! New type of simulation
      allocate( t_simulation_new :: sim )
end select
```

- Initial section of input file sets simulation mode
- Initialization code reads this first
- Then calls *initialize*() routine
  - Allocates simulation object of corresponding class
  - For this object
    - Calls the *allocate_objs*() method
    - Calls the *read_input*() method
    - Calls the *init*() method
  - Returns to the main program to start simulation

- Any/all of these methods may be overridden from the base *t_simulation* class
  - In most scenarios only the *allocate_objs*() needs to be overridden
  - Actual differences occur only in member objects

Example: new particles

- New simulation mode particles section accepts different parameters
  - Override *allocate_objs*() method in *t_simulation_new* to allocate objects of this new type

- The developer only needs to implement:
  - *allocate_objs_part_new*() - allocate a different species class, *species_new*
  - *read_input_new*() - implement new particles section on input file

- No changes required to other routines
  - The default *read_input*() method of the *sim* object will call the *read_input*() method of the *part* object

```
type, extends( t_particles ) :: t_particles_new
contains
   ! Allocate species_new objects instead
   procedure :: allocate_objs => allocate_objs_part_new
   procedure :: read_input  => read_input_new
end type t_particles_new
```

```
subroutine read_input_new( this, input_file, periodic, &
                if_move, grid, dt, &
                sim_options, ndump_global )

   ! new parameters
   ! ...
   read (input_file%nml_text, nml = nl_particles, iostat = ierr)

end subroutine read_input_new
```

Example: new species pusher

- New simulation mode includes species with a different pusher
  - Override *allocate_objs*() method in *t_particles_new* to allocate objects of this new type

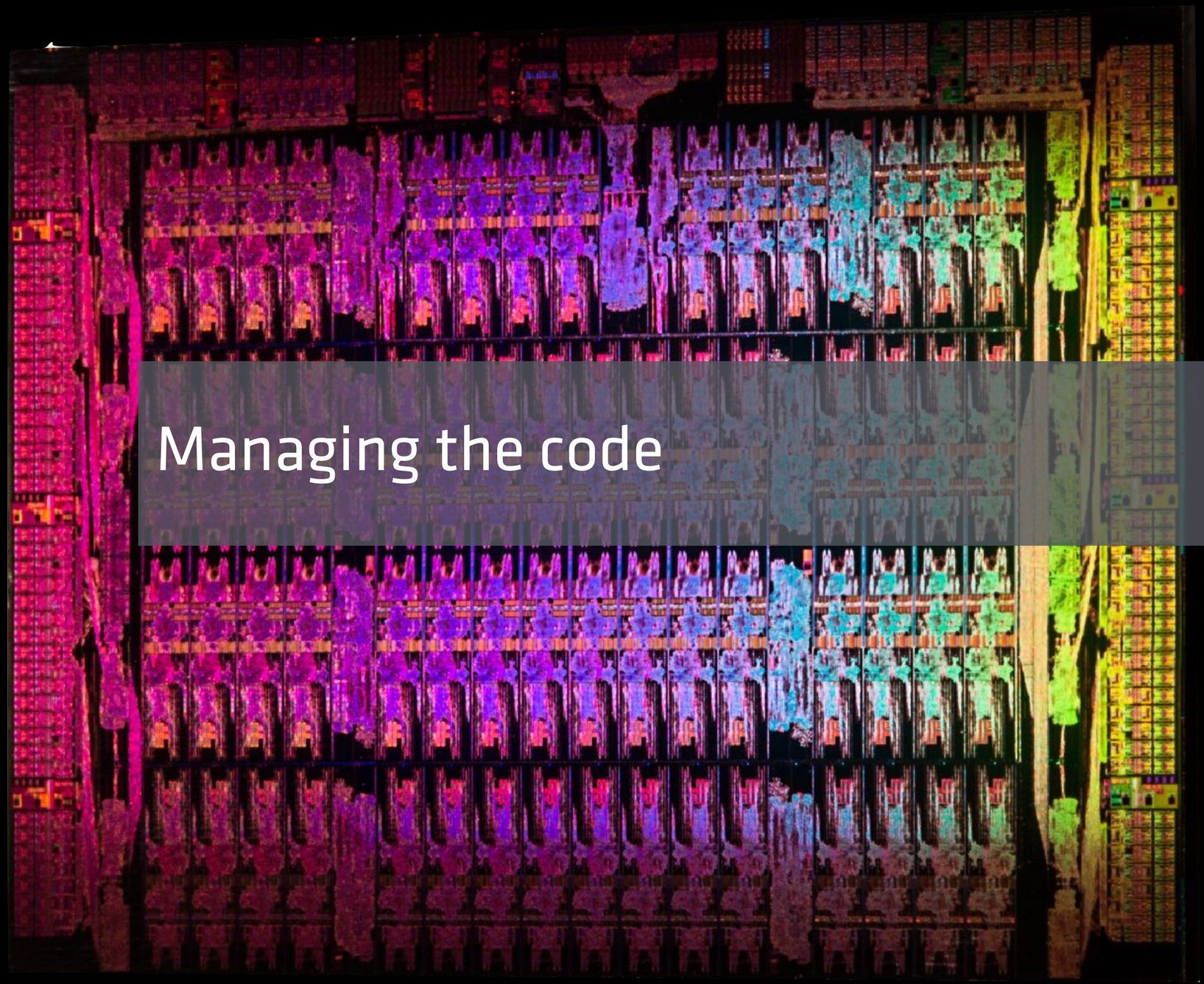
- The developer only needs to implement:
  - *push_species_new*() - the different pusher
  - *list_algorithm_spec_new*() - print information about the new pusher

- No changes required to other routines
  - The default *iter*() routine will call the *advance_deposit*() method of the *part* object
  - Which will then call the *push*() method for all *species* objects

```
type, extends( t_species ) :: t_species_new
   ! no additional class members, just a different pusher
contains
   procedure :: push => push_species_new
   procedure :: list_algorithm => list_algorithm_spec_new
end type t_species_new
```

```
subroutine list_algorithm_spec_new( this )

   implicit none
   class( t_species_new ), intent(in) :: this

   print *, ' '
   print *, trim(this%name),' :'
   print *, '- Type NEW pusher'

end subroutine list_algorithm_spec_new
```

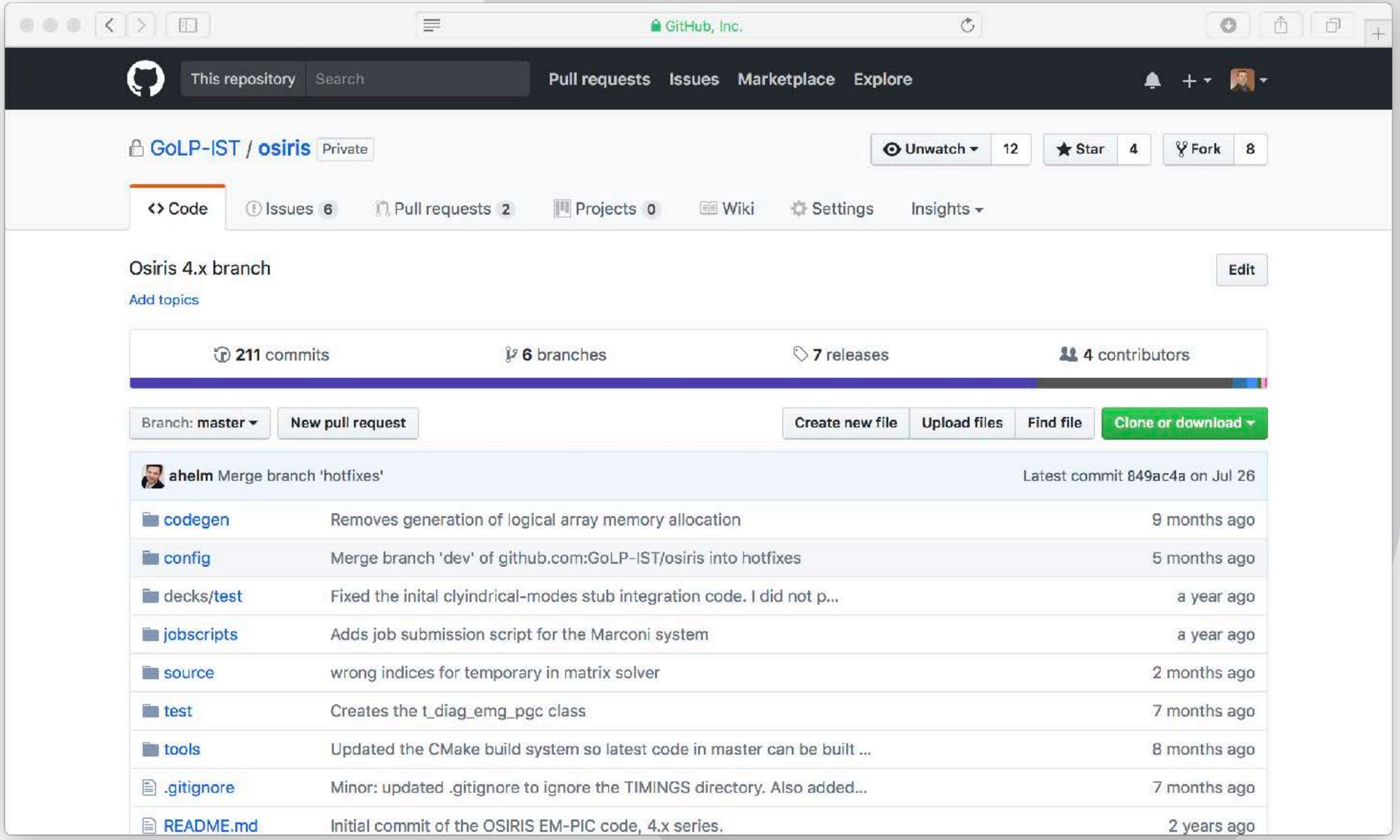
Managing the code



Intel Xeon Phi 5110p die

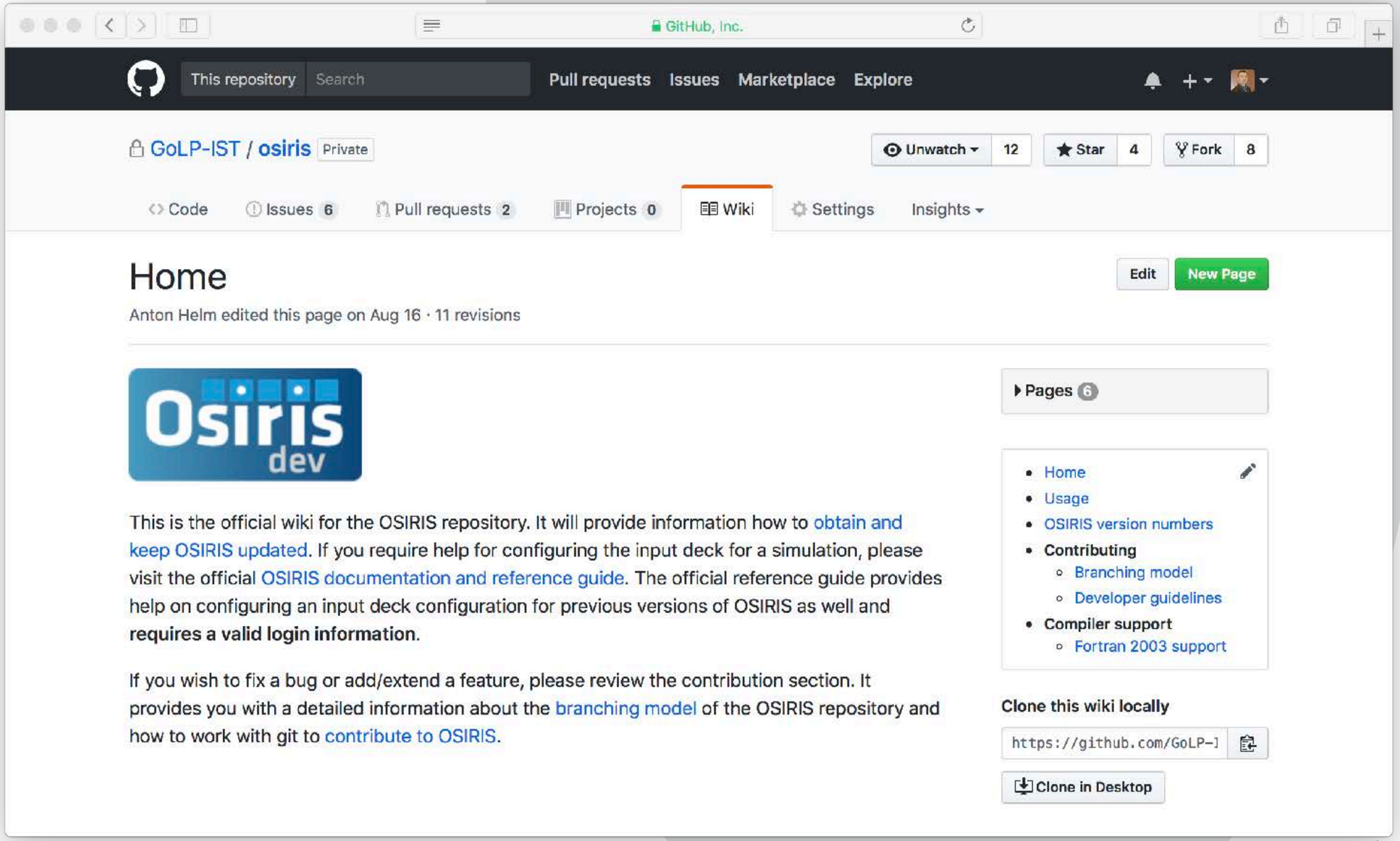
OSIRIS 4.0 is now hosted on GitHub

https://github.com/GoLP-IST/osiris

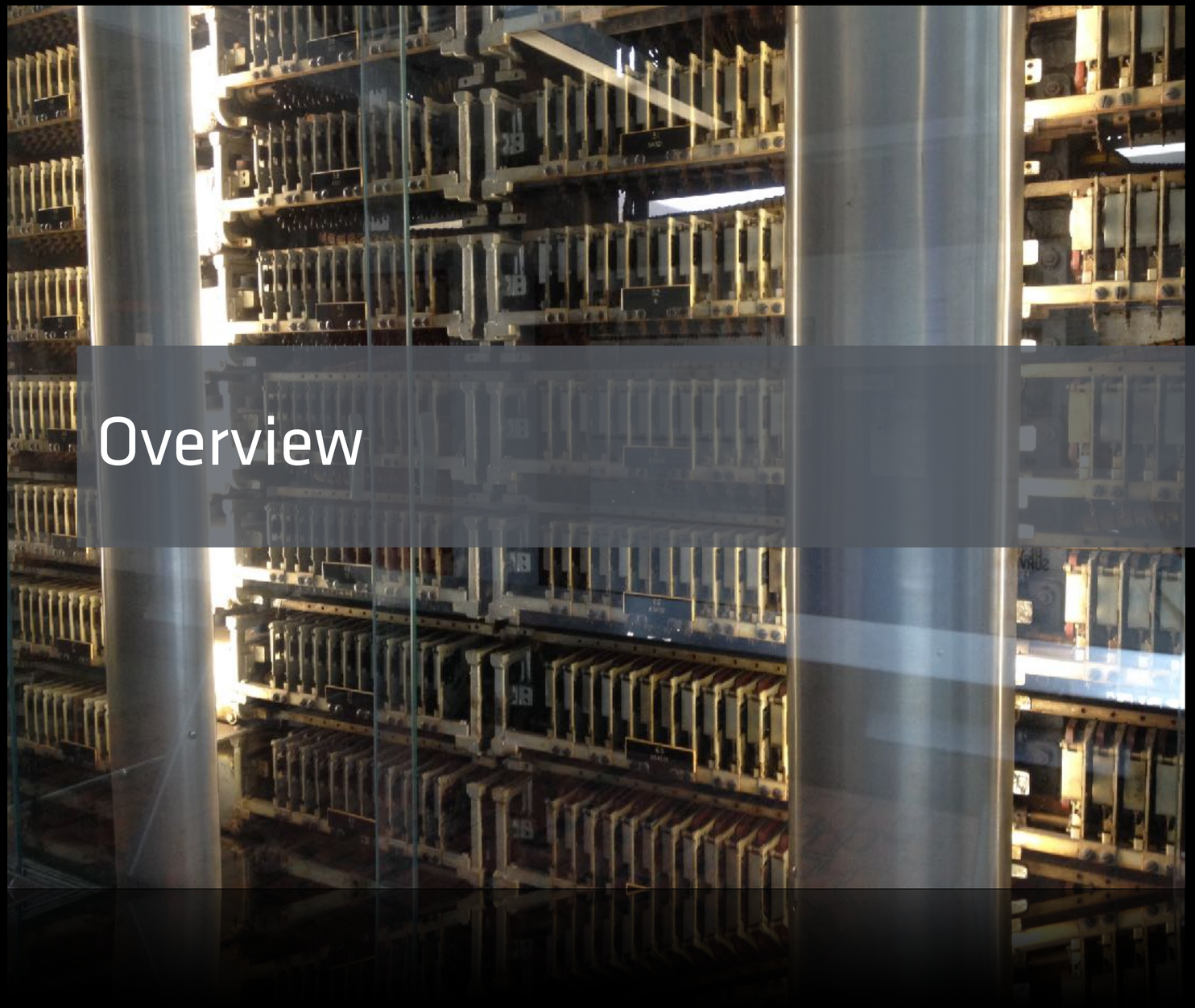


Please visit the wiki!

https://github.com/GoLP-IST/osiris/wiki



Overview



TÉCNICO LISBOA

Harvard Mark I - 1944

Rear view of Computing Section

Overview

- **The OSIRIS particle-in-cell framework**
  - State-of-the-art relativistic EM-PIC algorithm
  - Massively Parallel for large scale simulations / efficient enough to run on laptops
  - Widely used in many kinetic plasma scenarios, from high-intensity laser plasma interaction to astrophysical shocks

- **OSIRIS 4.0 is a robust, extensible framework**
  - Fully object oriented using Fortran 2003
  - Supports many additional simulation modes and physical models
  - Can be safely and efficiently extended to include new features

- **Move to 4.0 now!**
  - The 4.x series is ready for production use
  - All new development must go into this version
  - Go check out the GitHub repository and start using it today!