EFDA
EUROPEAN FUSION DEVELOPMENT AGREEMENT
Task Force
INTEGRATED TOKAMAK MODELLING

4/05/2009

ISIP tools training

B. Guillerminet for the ISIP team

https://portal.efda-itm.eu/portal/authsec/portal/itm/ISIP
isip@mail.efda-itm.eu

ITM framework

- **What is it?**
  - Set of tools: design, running, visualization, post-processing … tools
  - Set of resources: gateway, euforia, data
- **Status-Road map**
- **Contents of the training**
- **Introduction to KEPLER**
- **KEPLER in practice**
- **Advanced use of KEPLER**

ITM framework: what is it?

- **Set of tools**:
  - Data:
    - Standard naming: **CPO** (comprehensive description of a Tokamak/plasma through the CPOs)
    - Standard access: **UAL** (local, remote, GRID, HPC). Hidden data storage. jTraverser, Jscope
    - Experimental data: **exp2ITM**
  - Parameters setting of a simulation
    - Shot, time step, duration, Heating systems, …: **ISE**
  - Workflow/orchestration: **KEPLER**
  - Integration of codes: **FC2K**, **WS2K**
  - Monitoring: **ISE** (UAL data), **Migrating desktop** (jobs), **Kepler actors** (wf data)
  - Visualization: **ISE, Numplot, Visit, Matlab, Scilab**
  - Post-processing: **Numpy, Matlab, Scilab**
- **Set of resources:**
  - Cluster at Portici: **Gateway**
  - GRID & HPC computers: **EUFORIA**
  - **HPC-FF** at Juelich

Status-Road map

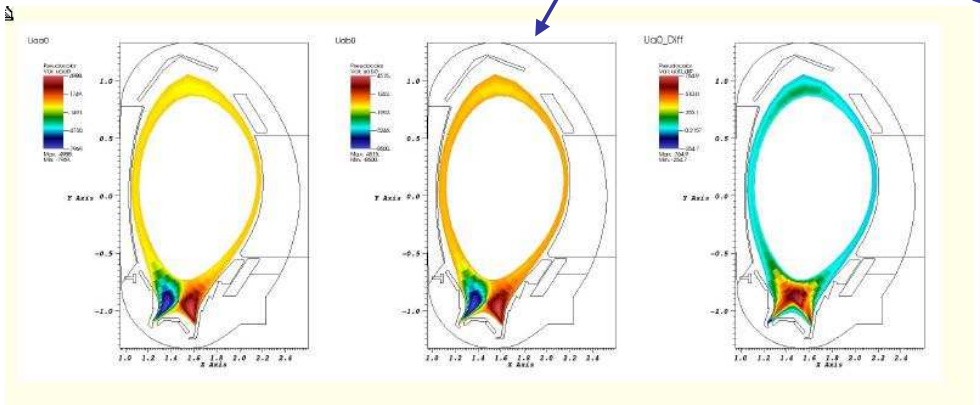
- **Current version: v1.0** (released on December 08)
  - Data structure: 4.06d
  - UAL (CPO & time slice)
    - Local on the gateway: Fortran (g95 & pgi), C, C++, Java (1.5 or +), Matlab.
    - GRID: Fortran (g95) March 2009
    - HPC-FF: Fortran (g95 & pgi) 28 April 2009
  - Data:
    - Shot 3, run 1
  - Experimental data: exp2ITM for TS & JET
    - JET: ?
    - Tore Supra: 40000
  - Data storage:
    - MDSplus
    - HDF5
    - Memory (thread version => ok for multicores but not available for multi-nodes)
  - ISE (ITM Simulation Editor): released in June 2008 but due to change in the run number management, it must be reengineered
  - KEPLER: release 1.0, one UALinit & one UALcollector
  - FC2K: version 1.4 no time slice
  - Visualization: using KEPLER actors
  - Matlab

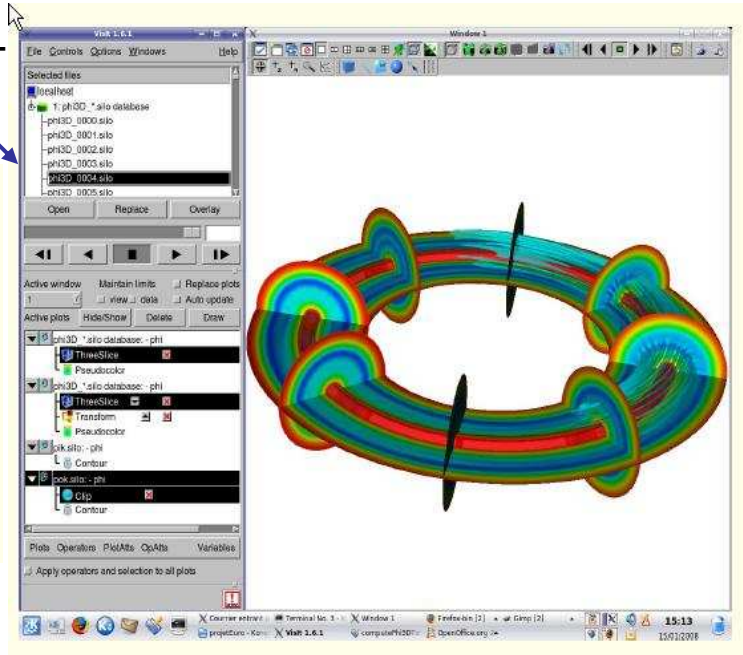
- **Additional tools:**
  - Numplot: standalone tool (available now)

Status-Road map

- **version: v1.1** (released on May/June 09)
  - Data structure: 4.06d
  - KEPLER: release 1.0, new version of UALinit & UALcollector (occurences + several UALinit or UALcollector)
  - FC2K: version 1.4b with time slice

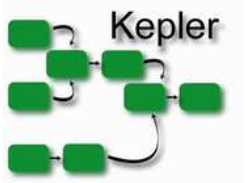
- **version: v1.2** (released on July 09)
  - Data structure: 4.07
  - ISE (ITM Simulation Editor)
  - Visualization: using numpy actors + VISIT





Training

- **Contents:**
  - Data (presented by F Imbeaux):
    - **CPO**
    - **UAL**
    - **exp2ITM**
  - Workflow/orchestration: **KEPLER**
  - Integration of codes: **FC2K**
  - **ISE** (not presented but … slides available)
  - **Numpy, Visit, Matlab** (not presented but …)

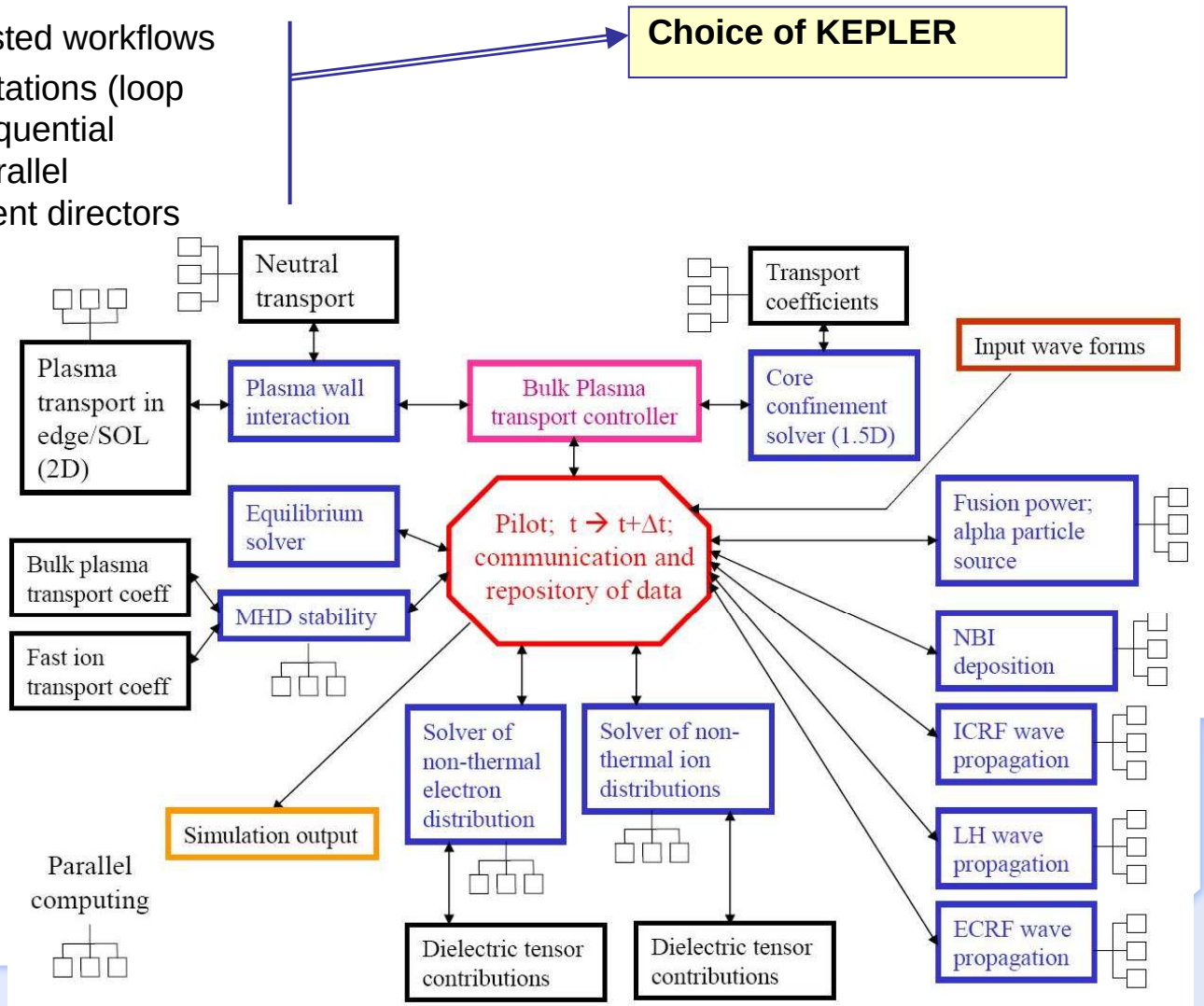


Introduction to KEPLER

- **Workflow design:**
  - Why an orchestration tool
  - Introduction to KEPLER: terminology
  - A few actors
  - Computation model & directors
  - Fusion workflow:
    - Building simple workflows:
      - getting a CPO & plotting some data
      - Reading & updating a CPO

Orchestration tool

- **Fusion simulation:**
  - Complex workflow => nested workflows
  - Different model of computations (loop on model time, solver, sequential processing, branches, parallel computation …) => different directors

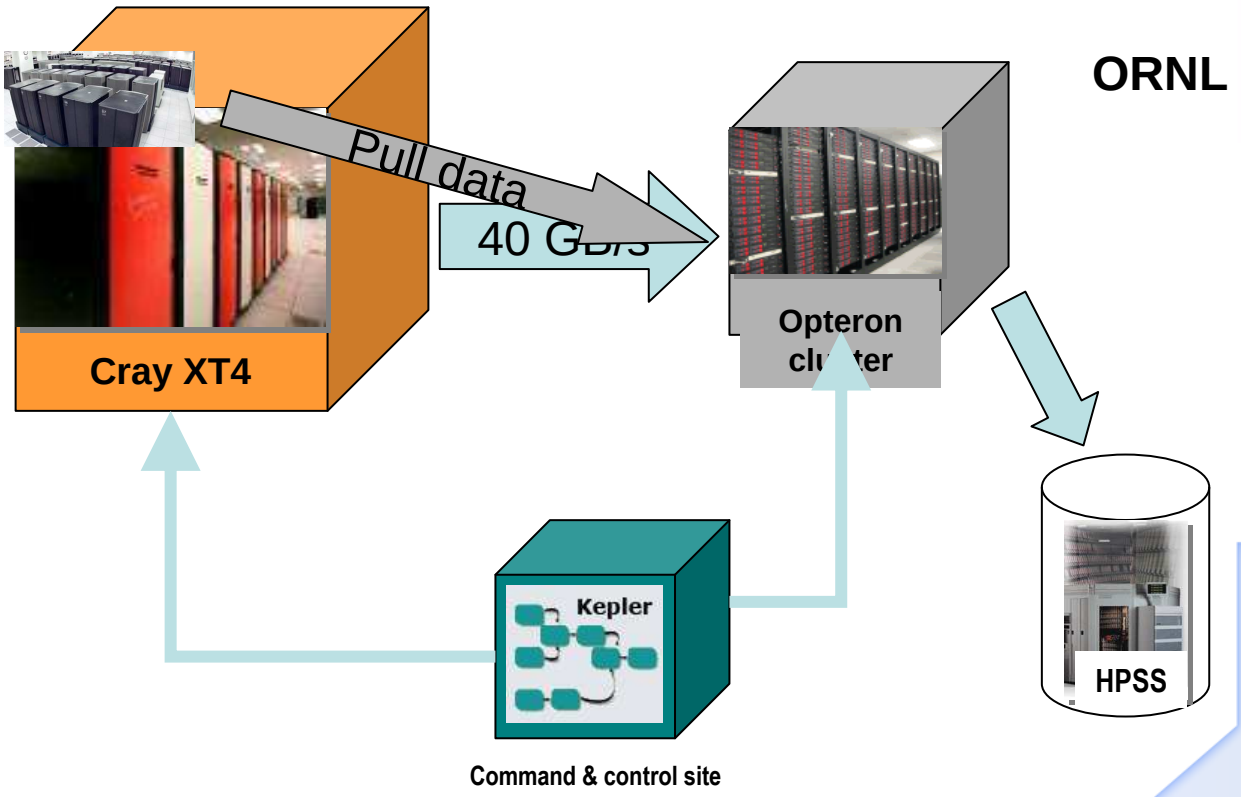


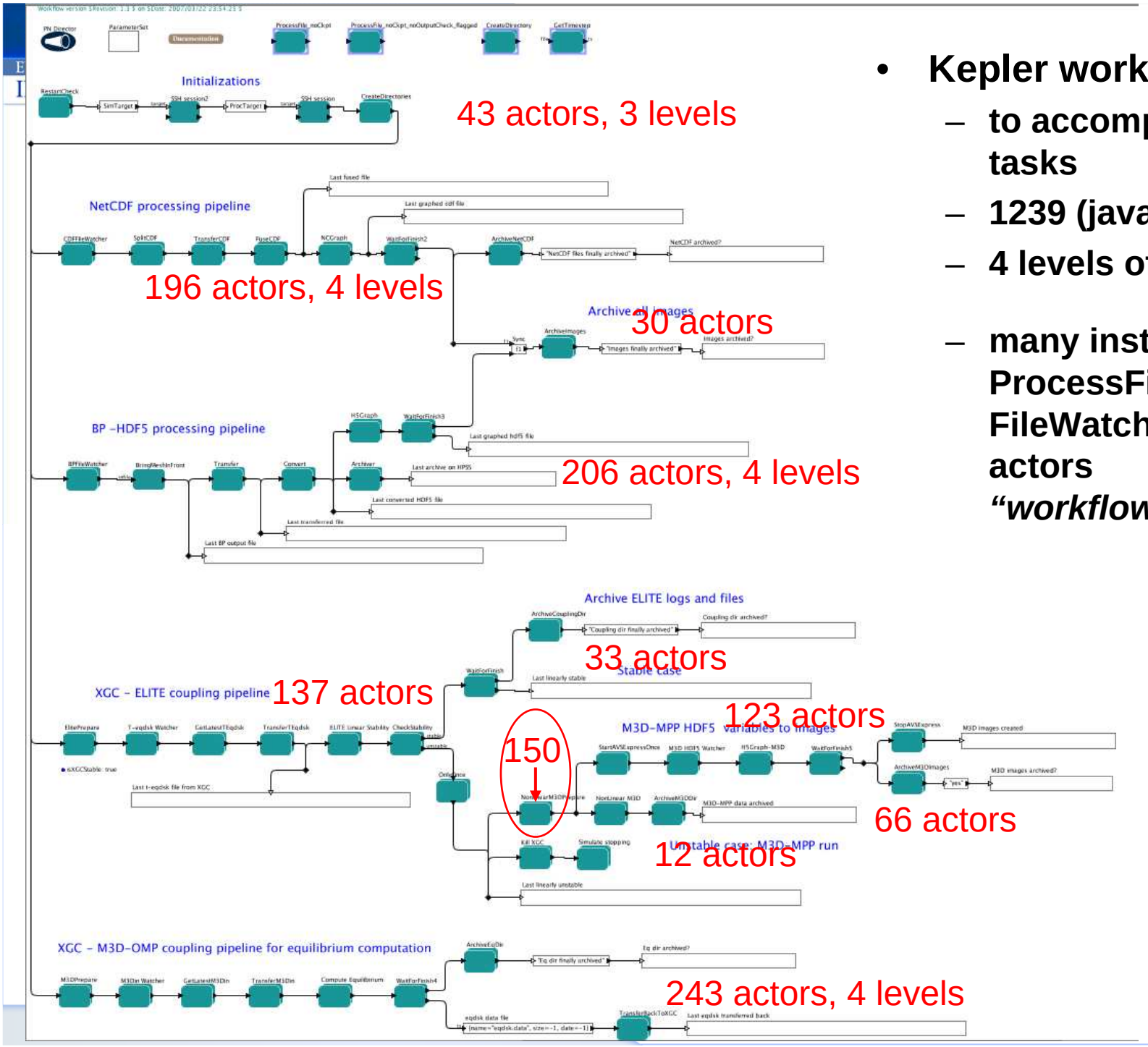
KEPLER

Based on Ptolemy II (Berkeley), San Diego, world widely used, friendly tool

Used for:

- **Simplify and automate the workflow (SDM, CPES …)**
- **Coarse grained programming (one instruction is a big chunk): assembling components**
- **Graphical design**
- **Mixing complex models of computation**





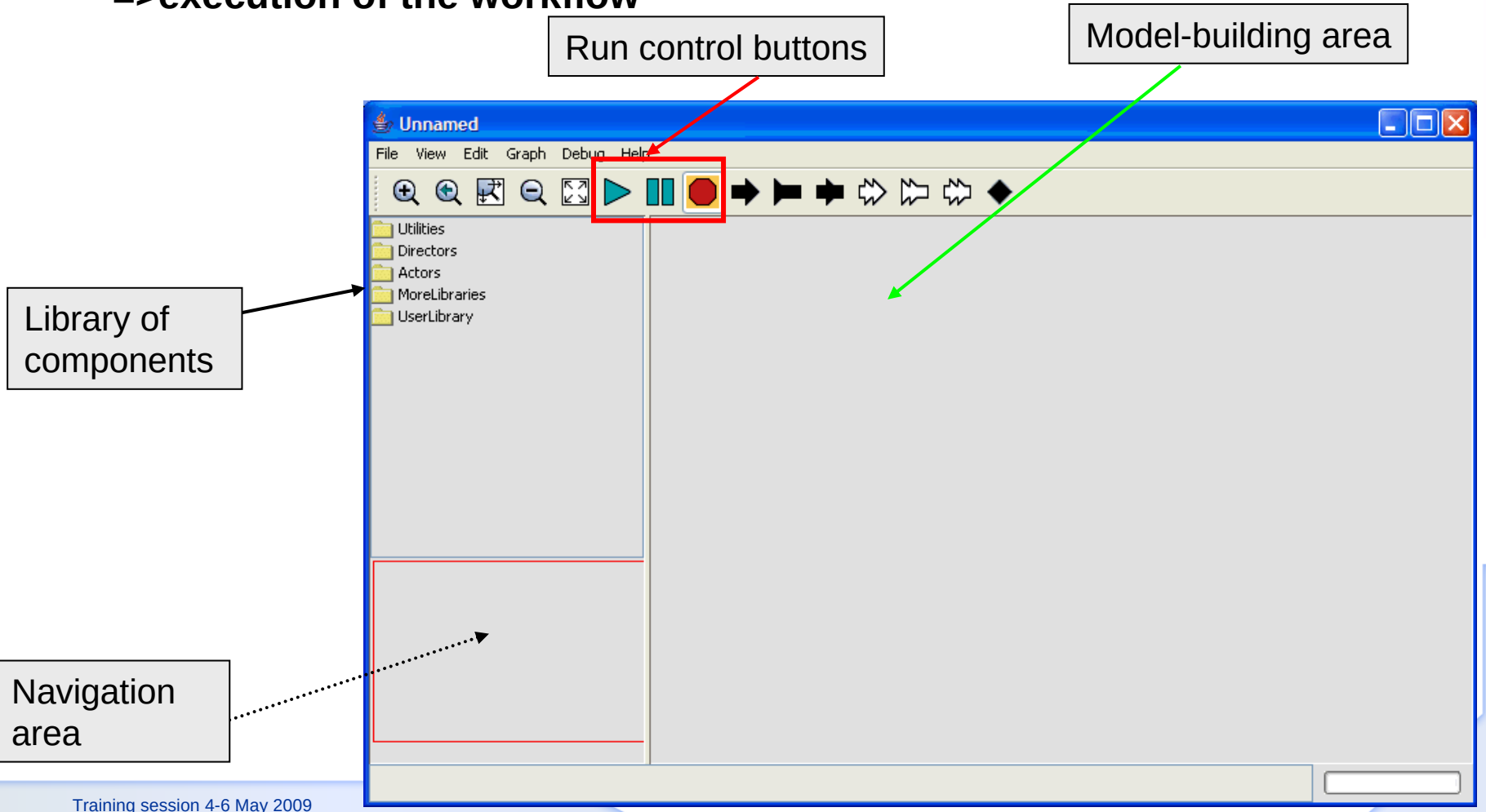
- **Kepler workflow**
  - **to accomplish all these tasks**
  - **1239 (java) actors**
  - **4 levels of hierarchy**
  - **many instances of ProcessFile and FileWatcher composite actors *"workflow templates"***

KEPLER

2 main usages

=>design of workflow

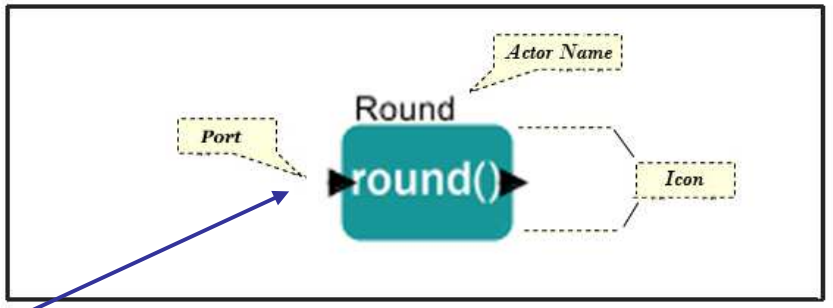
=>execution of the workflow

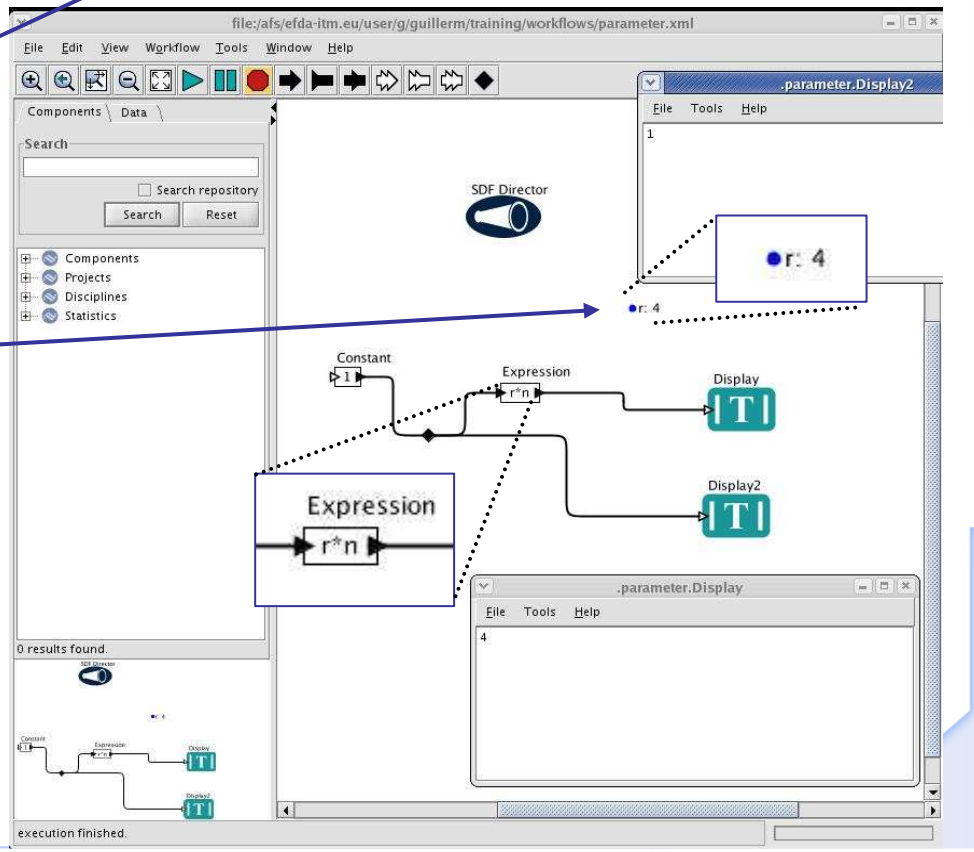


KEPLER

Terminology:

- **Manager**
  - **Single (suspend/resume…)**
- **Directors**
  - **Scheduler of the actors (components)**
- **Actors**
  - **Ports: Input & Output of an actor**
  - **Parameters:**
    - **Shared by the actors**
  - **Will be detailed in the next session**



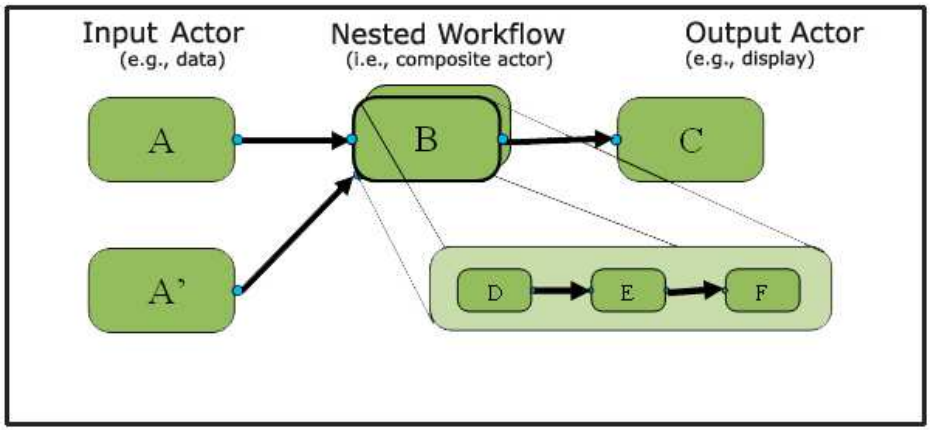


Kepler

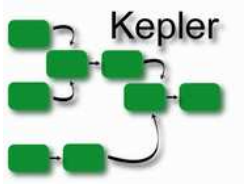
ACTORS

Components of the workflow:

- **Actors could be nested**
- **Fusion codes: Helena, Mishka, Orbit …**
- **Categories**
  - **sources**
    - **Const, string, clocks**
    - **Ramp, sinewave, shell, …**
  - **sinks**
    - **Display, XYZplotter, timedplotter**
    - **Recorder, …**
  - **Array**
    - **Arrayextract, arrayminimum, arraysort, …**
  - **Conversion**
    - **Complextopolar, stringtoxml, …**
  - **Flow control**
    - **Switch, Sampledelay, Exit, …**
  - **I/O**
    - **Filereader, writer, …**
  - **Math**
    - **Average, …**
  - **Matrix**
  - **Random**
  - **Signal processing**



- **Creation of actors will be detailed in the next session**
- **See "ActorReference.pdf"**



Computation models & directors

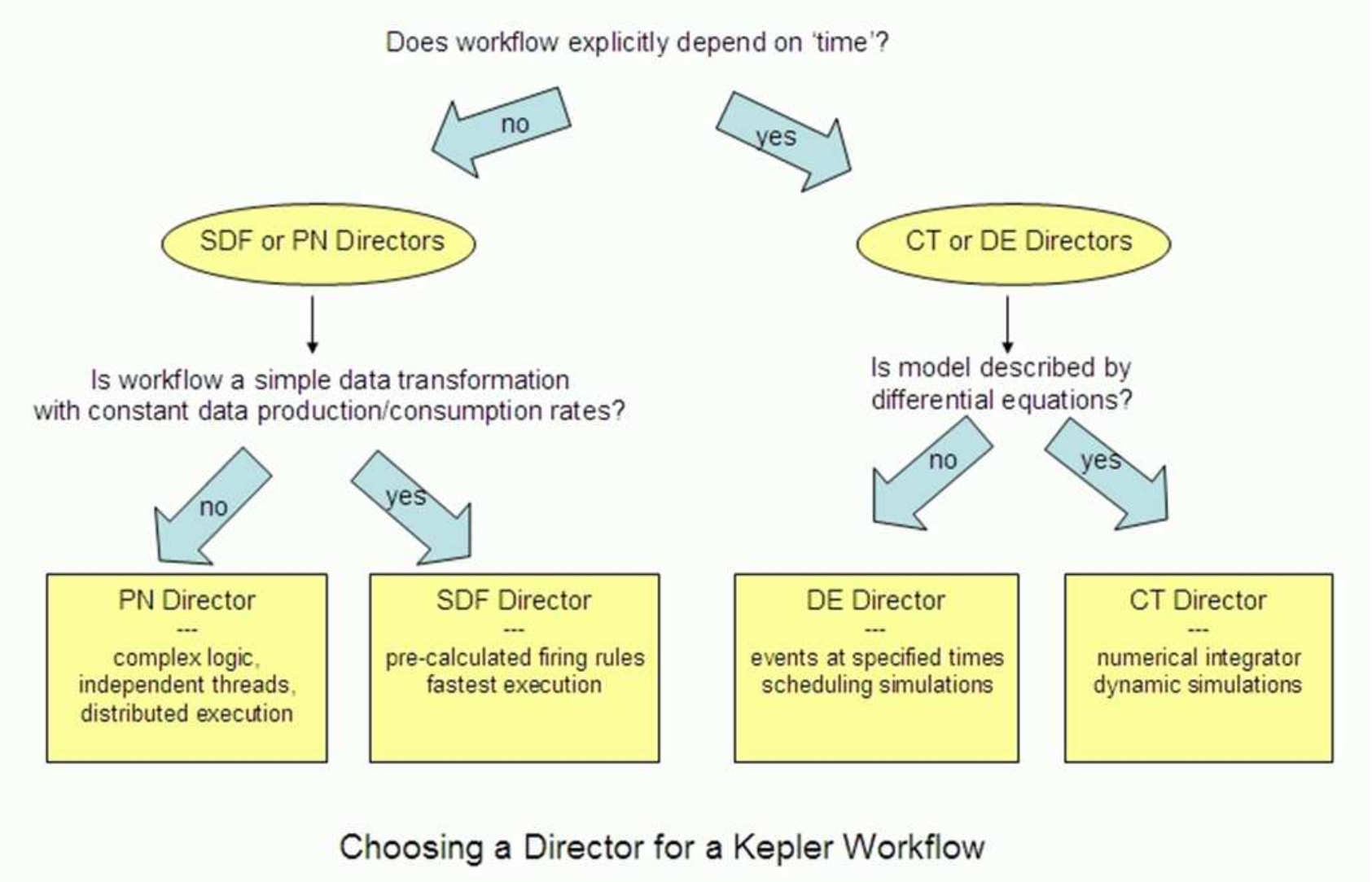
Directors

- Tell the actors when it has to produce its output = order in which they should execute
- 5 basic schedulers (directors):
- DATA DRIVEN
  - SDF
    - Synchronous data flow: fairly simple, sequential workflow
  - DDF
    - Dynamic data flow = SDF with loops
  - PN
    - Parallel processing on distributed computing systems
- TIME DRIVEN
  - CT
    - Continuous time driven
  - DE
    - Discrete event: modelling time
- MIXING DIRECTORS

- Beware: workflows and actors could depend on the director

Kepler

How to choose a director



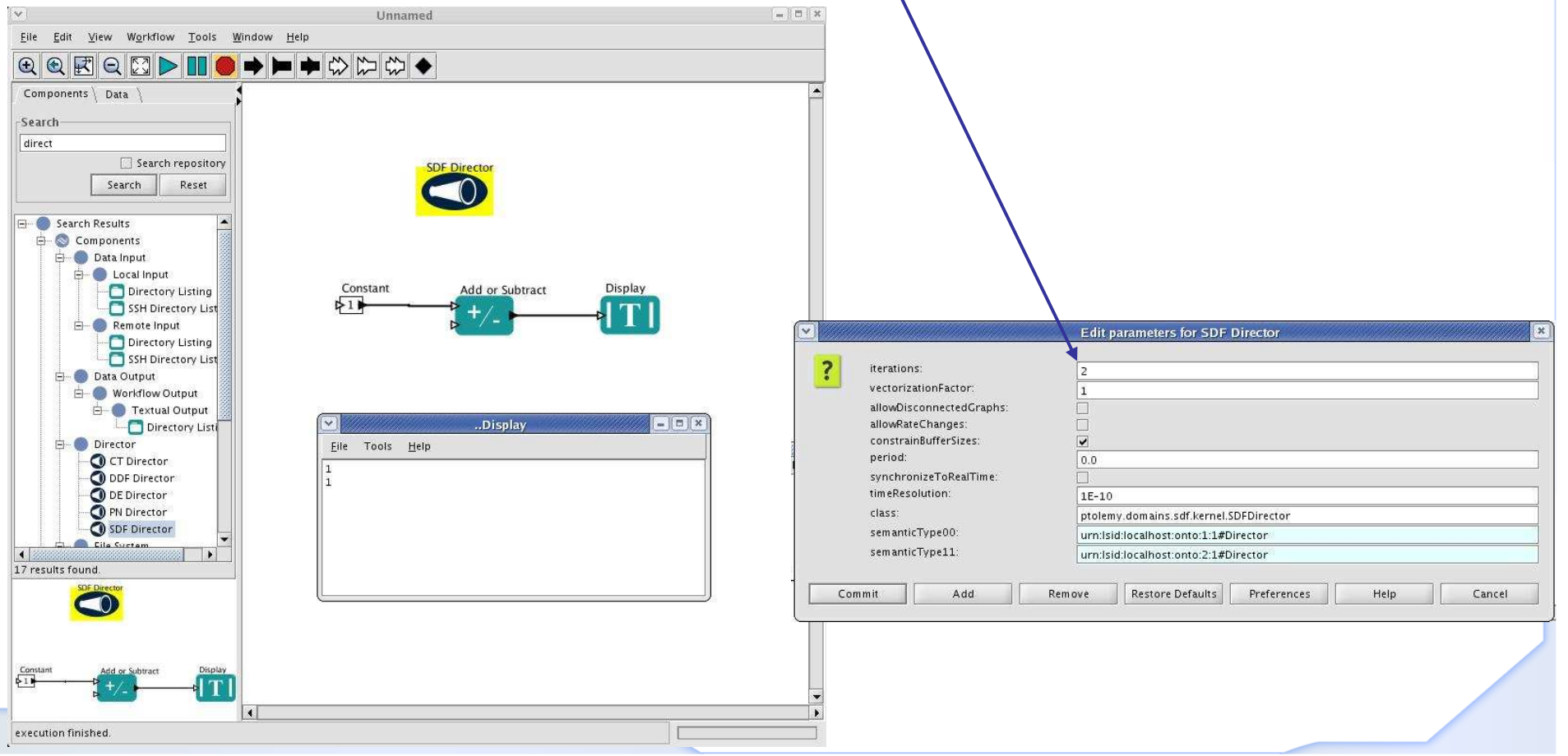
Choosing a Director for a Kepler Workflow

Kepler

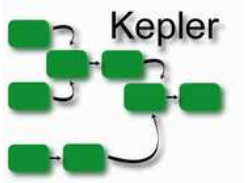
Design of a workflow (1)

Design

1. **Choose your actors (for instance constant, addsubstract & display: use the search cmd)**
2. **Drag & drop them in the design area**
3. **Connect the actors => draw a link between input & output port**
4. **Define the director (SDF) and its parameters (number of iterations)**



Creating an actor in 5 steps



1. **Identify your CPOs in & out**
2. **Turn code into a subroutine with CPOs as argument**
3. **To make sure the code handles the CPOs correctly, run it in a "testbed"**
4. **Make a library of your routine (compilation with the –fPIC option is essential).**
5. **Run FC2K**

Prerequisite:

1. **Install a private version of Kepler in your directory (available on /afs/efda-itm.eu/project/switm/kepler/4.06d/kepler.tar)**

   >cp /afs/efda-itm.eu/project/switm/kepler/4.06d/kepler.tar $HOME

   >tar xvf kepler.tar

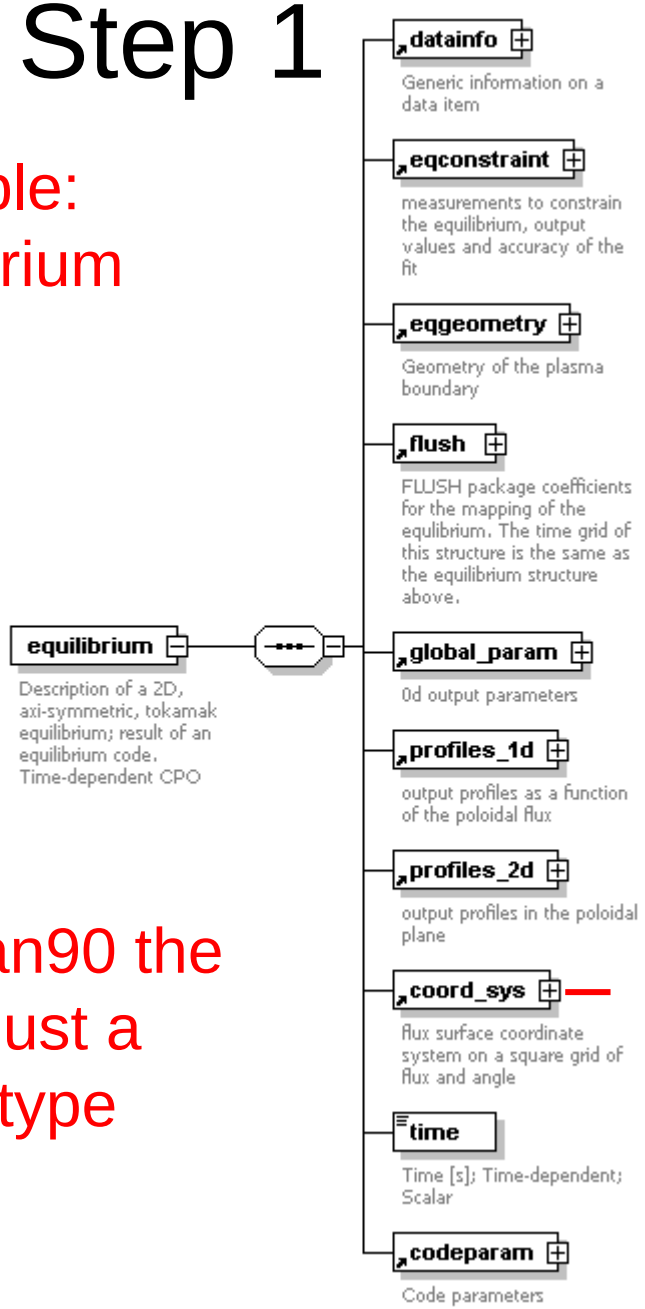
2. **Set the environment variables with ITMv1**

   >source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.06d (where kepler is the directory of your KEPLER version; use public for the standard KEPLER version)

Step 1

- A data structure describes how the data in a CPO are organised.
- The data structures are defined with XML schemas.
- The code developer only needs to know about the organisation itself.
- ITM tools automatically generate type declarations for inclusion in a code so that access to data is readily available.

Example: equilibrium CPO



In Fortran90 the CPO is just a derived type

Step 2

- **Turn code into a subroutine with CPOs as argument (example of a code which extract the pressure from the equilibrium CPO).**

```
subroutine cpo2ip(equi_in, ip)
  use euitm_schemas
  use euITM_routines
  implicit none
  integer,parameter :: DP=kind(1.0D0)

  type (type_equilibrium),pointer :: equi_in(:)
  integer :: ip(20)

  write(*,*) 'pressure: ',equi_in(1)%profiles_1d%pressure
  do i=1,20
    ip(i) = int(equi_in(1)%profiles_1d%pressure(i))
  enddo
  write(*,*) 'ip:',ip
  return
end subroutine cpo2ip
```

Input arguments can be CPOs, integer, floating point, …: single or array

All the ITM type declarations are included here

Declaration of the equilibrium CPO

Get the pressure and fill a local array

Step 3

- **To make sure the code handles the CPOs correctly run it in a "testbed"; F90 example:**

```
program test_bed

use euITM_schemas
use euITM_routines

implicit none

type (type_equilibrium),pointer :: cpotest(:)

character(len=5)::treename

integer :: idx, shot, run

interface
  subroutine mycode(cpotest)
  use euITM_schemas
  type (type_equilibrium),pointer :: cpotest(:)
  end subroutine mycode
end interface

shot = 180
run = 1
treename = 'euitm'

 call euitm_open(treename,shot,run,idx)

 call euitm_get(idx,"equilibrium",cpotest)

call mycode(cpotest)

stop
end
```

Example for "mycode"

Specify "mycode"

Run "mycode"

- One can start with a very simple test_bed program.
- Shown on the left is an example with only an equilibrium CPO as input
- Programs of this type with their Makefile are available on the Gateway for copying

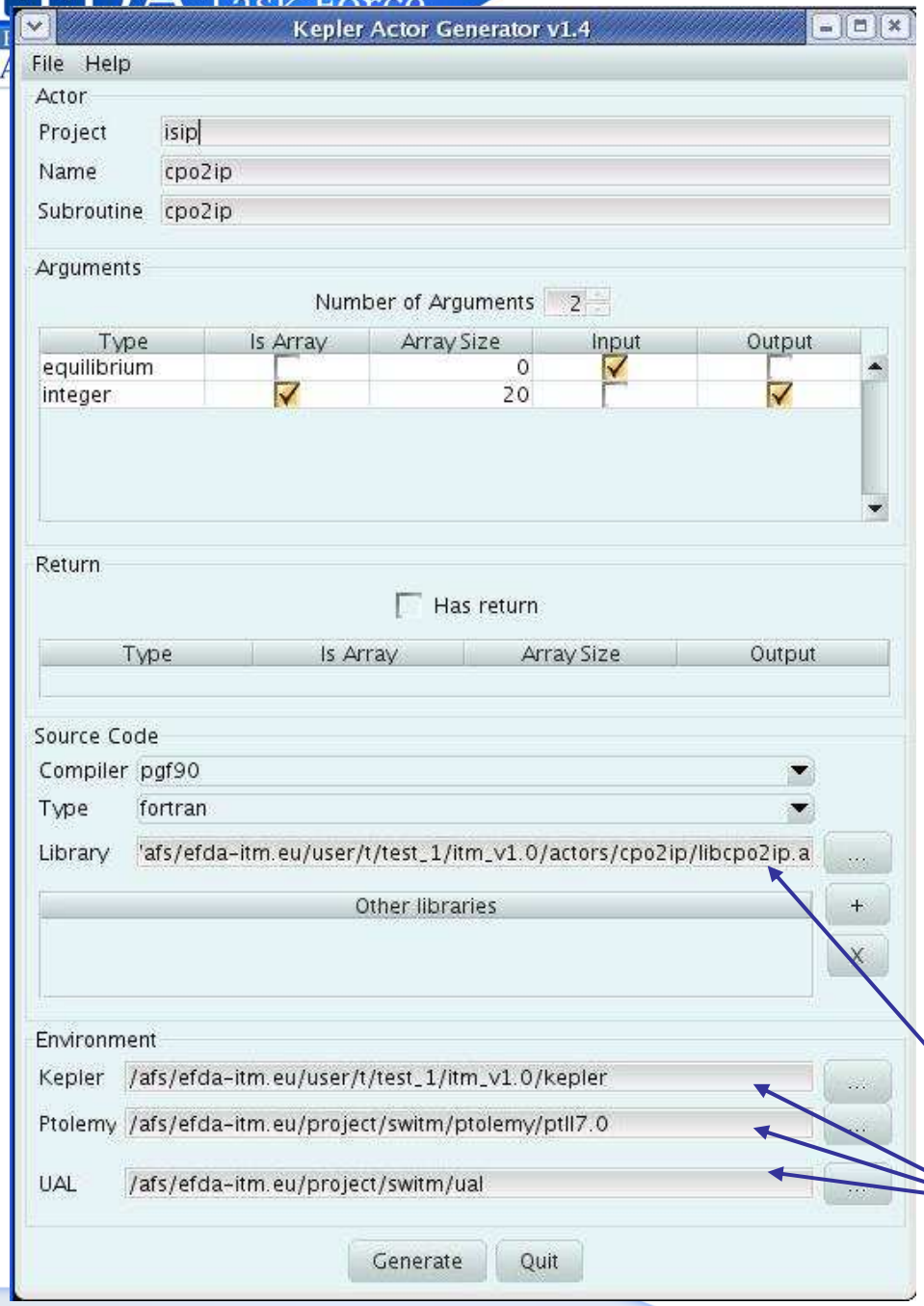
Step 4

- Once the code runs correctly in the test bed it is time to make a Kepler actor of it.

- Make a library of your routine (either static, mylib.a, or dynamic, mylib.so; compilation with the –fPIC option is essential). Example for cpo2ip.f90:

- F90=pgf90
- COPTS= -r8 -Mnosecond_underscore -fPIC
- LIBS= -L/afs/efda-itm.eu/project/switm/ual/lib -lUALFORTRANInterface_pgi
- INCLUDES= -I/afs/efda-itm.eu/project/switm/ual/include/amd64_pgi

- all: libcpo2ip.a
- libcpo2ip.a:cpo2ip.f90
- 	$(F90) $(COPTS) -c cpo2ip.f90 ${INCLUDES} $(LIBS)
- 	ar -rv libcpo2ip.a cpo2ip.o
- clean:
- 	rm *.a *.o

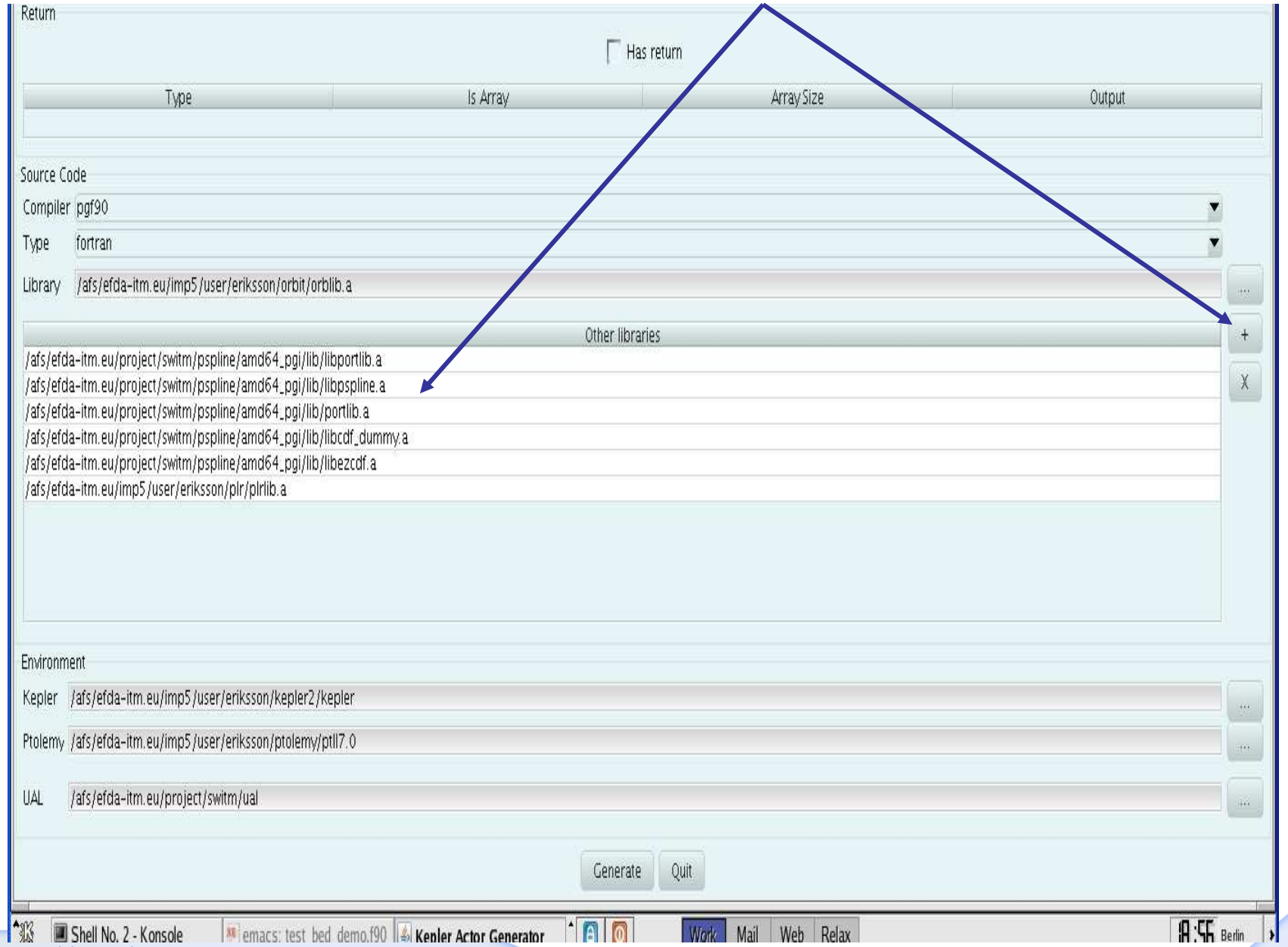


Step 5

- **Run the KEPLER actor generator script: FC2K**

- isip = folder for description storage
- Name of the actor
- Name of the subroutine (no underscore allowed)
- Input & output arg.
- g95, pgf90, C, C++
- Your code
- Use $KEPLER, $PTII & $UAL

• Add the libraries



- **The actor generator creates a “wrapper” for the code, it manages the call to UAL etc.**

- Stored in (where xxx is the actor name) :

- $KEPLER/src/cpp/itm/xxx (Makefile, FortranWrap.f90, libxxx.a, libxxx.so, …). “>make” & “>kepler”
- $KEPLER/src/eu/itm/xxx (xxx.java & JavaJniCall.java)

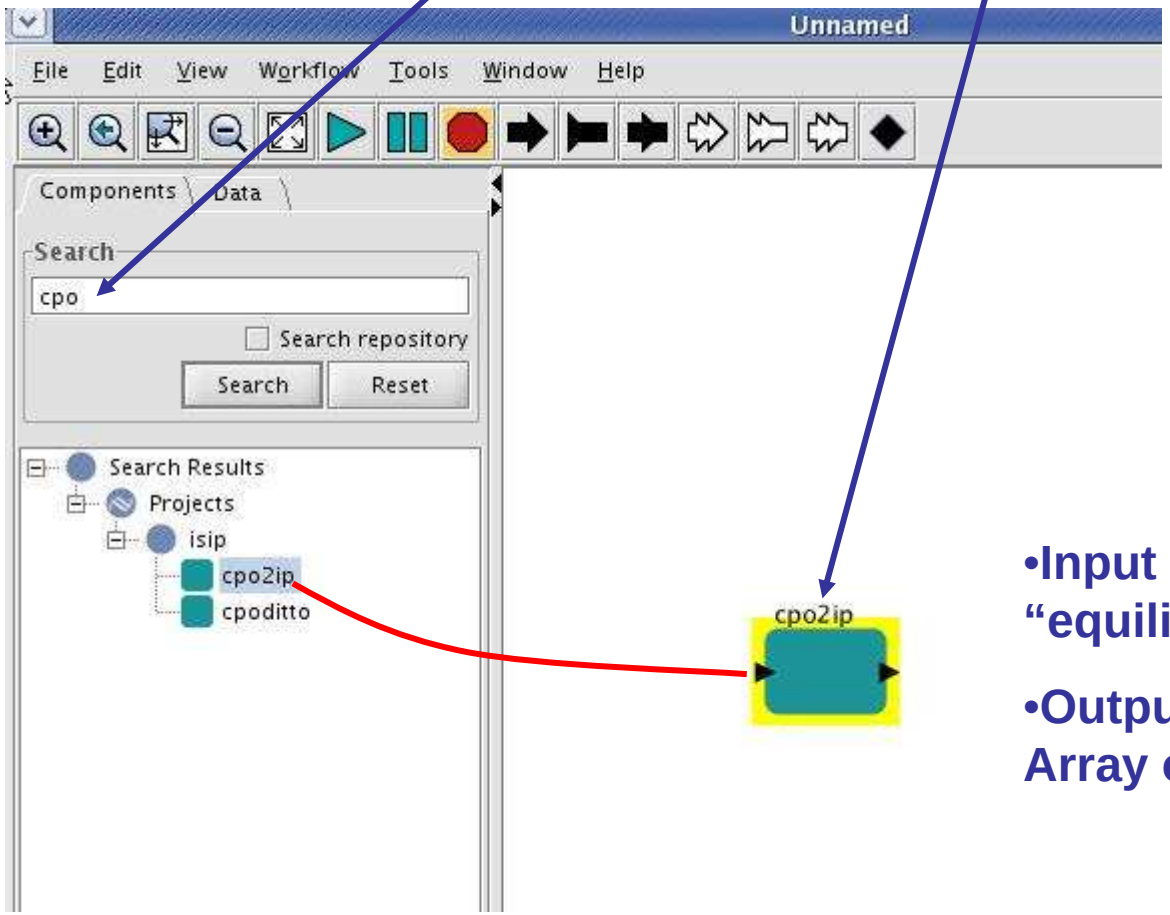
- $KEPLER/kar/actors (xxx.kar)
- $KEPLER/lib (libxxx.so)
- $KEPLER/build/src/cpp/itm/xxx copy of $KEPLER/src/cpp/itm/xxx
- $KEPLER/build/src/eu/itm/xxx copy …
- $HOME/.kepler keeps a cache of your actors!!

TIPS

- A few tools to share actors:
- >rmactor xxx (remove the actor xxx from your $KEPLER version)
- >getactor xxx Get the actor xxx from $KEPLER and build xxx.tar (tar tvf xxx.tar to look at its contents)
- >putactor xxx Put the actor contained in xxx.tar into $KEPLER
  - Then update your KEPLER version by:
  - cd $KEPLER
  - ant buildkarlib
- These scripts are in /afs/efda-itm.eu/project/switm/scripts

Building your workflow

Using KEPLER (“>kepler”), look for your actor and copy it to the design area



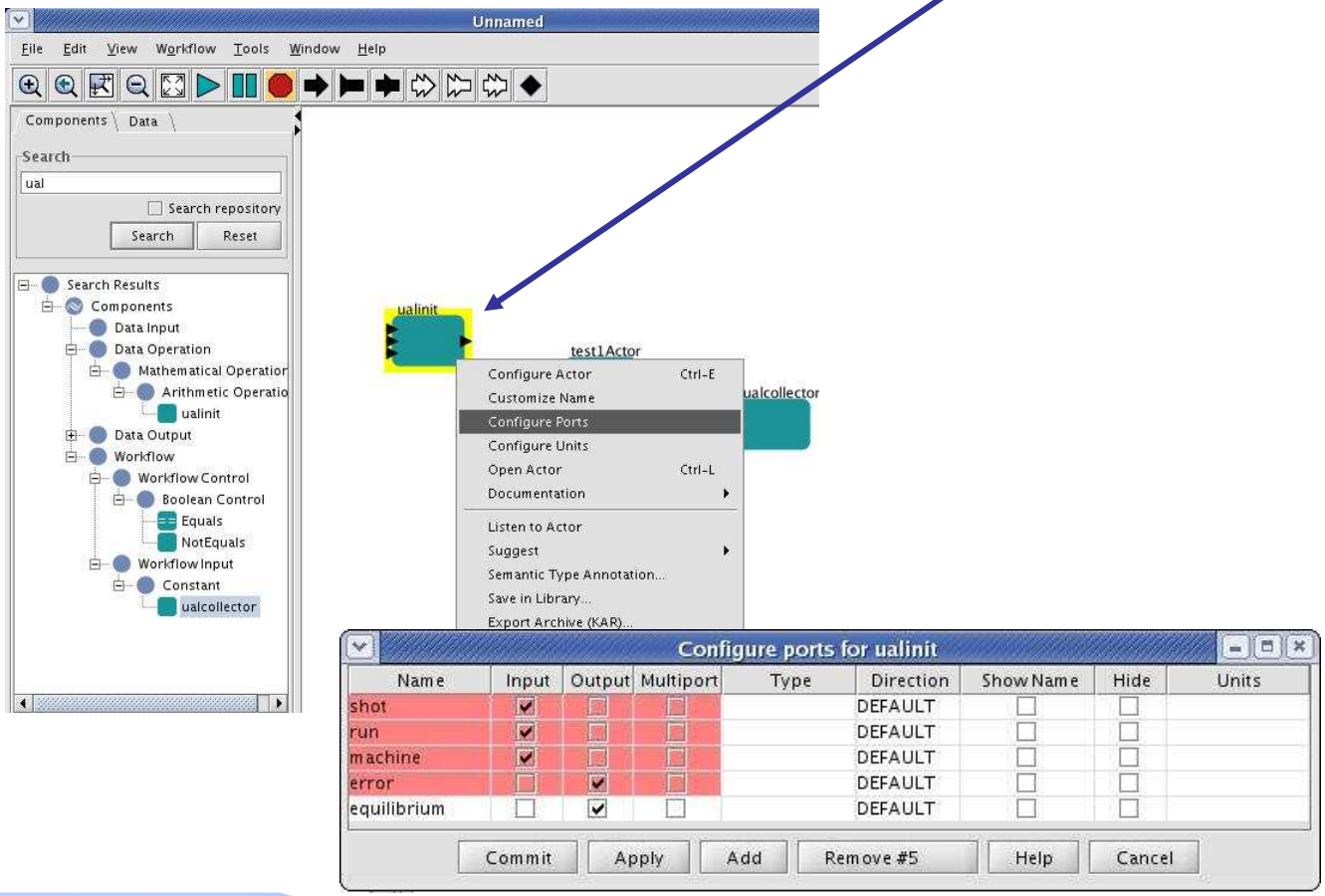
- **Input port: “equilibrium” CPO**
- **Output port: Array of integer**

Kepler

Fusion workflow (1)

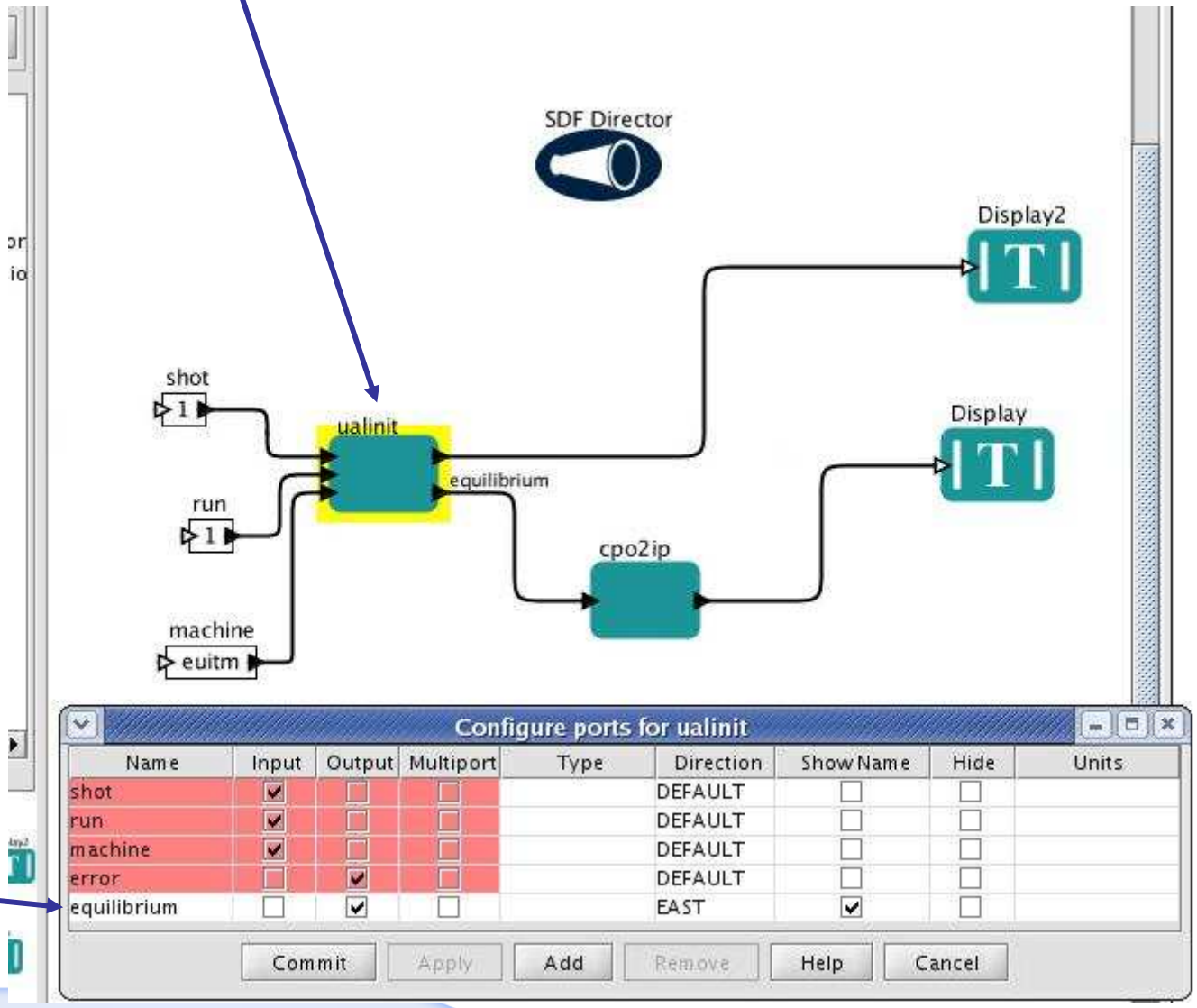
Design

1. **Connection with the UAL: reading the database and storing in memory UALinit**
2. **Insert your actor/workflow**
3. **Store the simulation in the database: UALcollector**



Kepler

Add the actor which connect to the ITM database



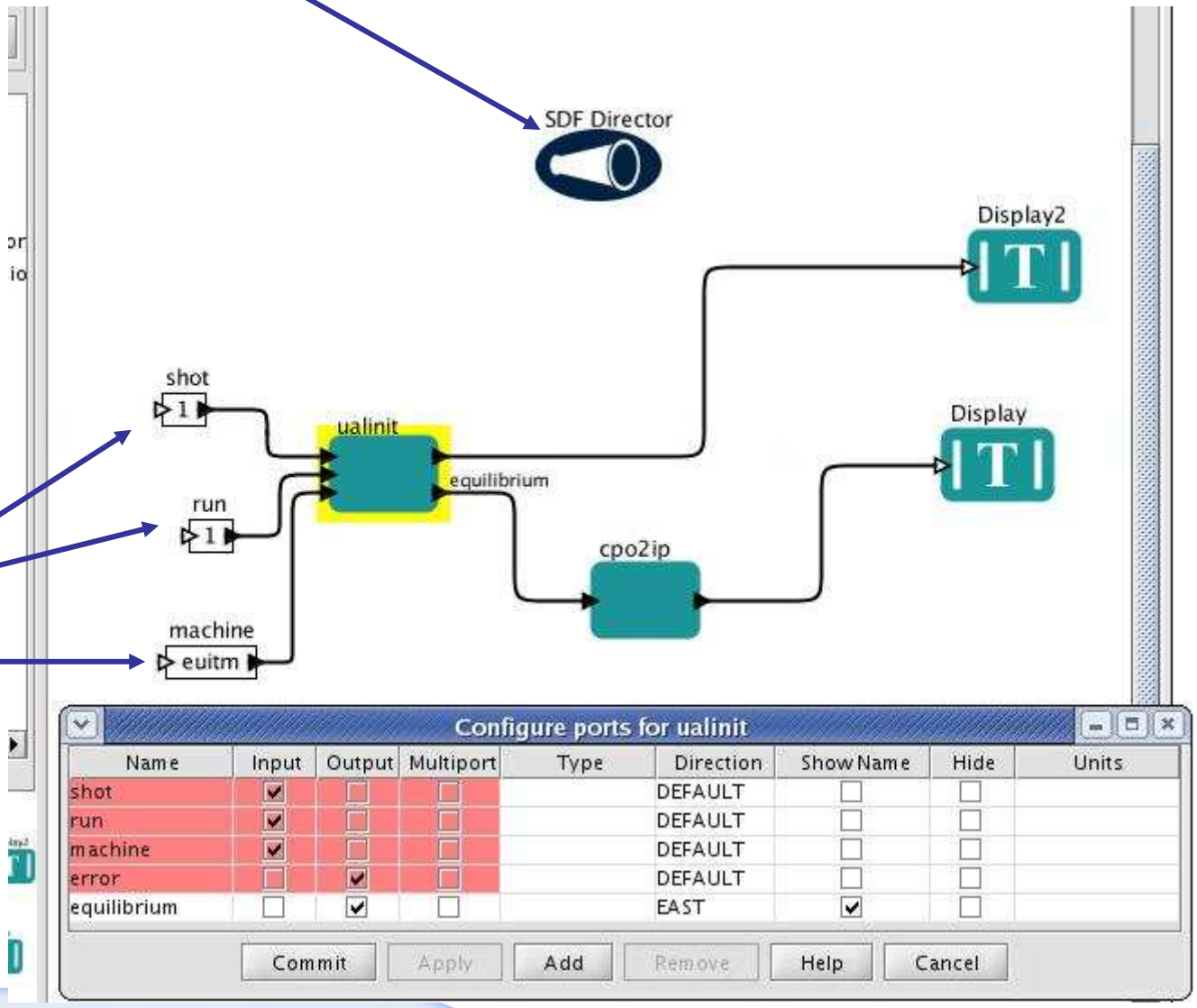
Add the output CPOs

Kepler

Add the Director (Synchronous Data Flow)

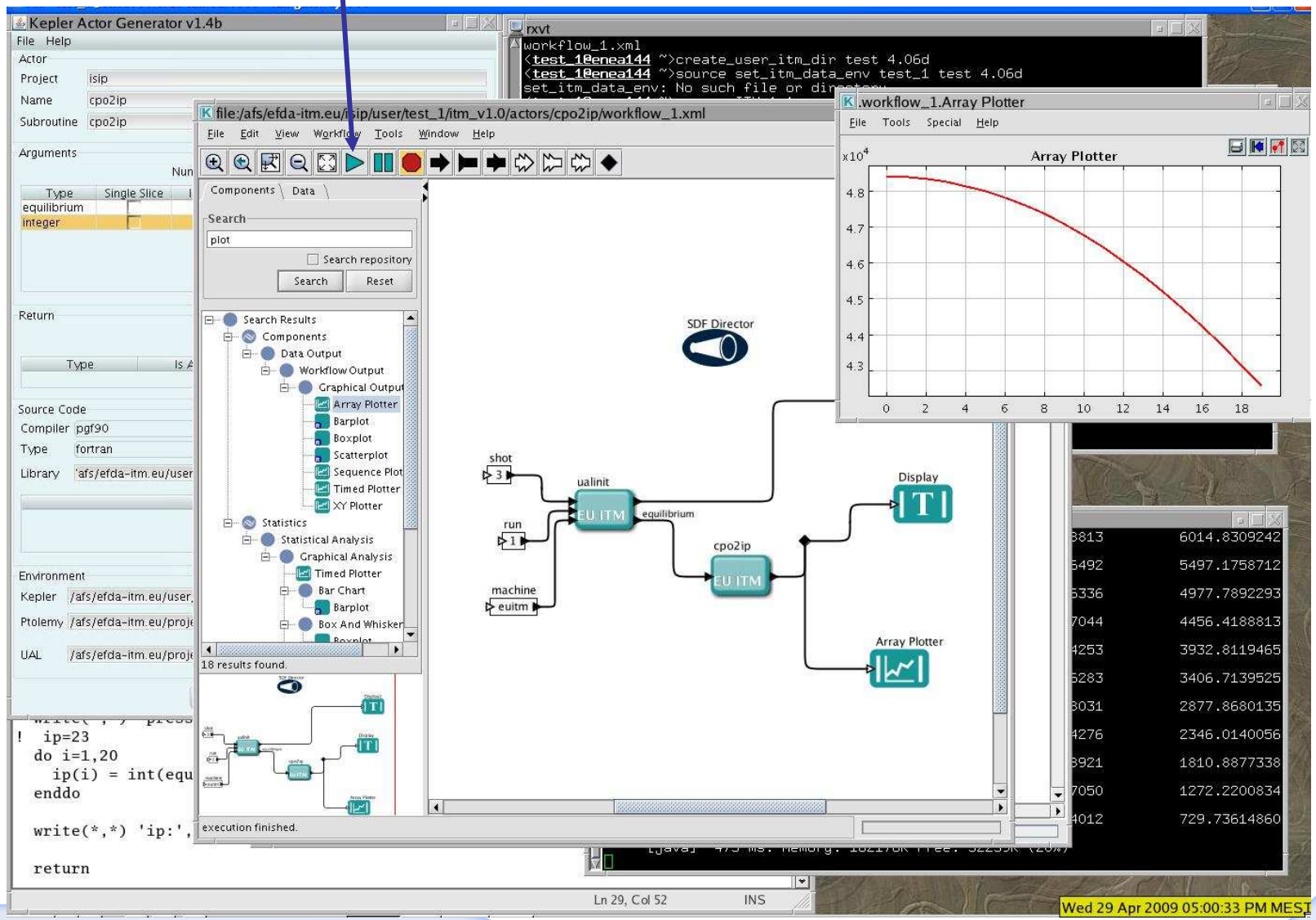
Add the inputs:

- **Shot**
- **Run**
- **machine**



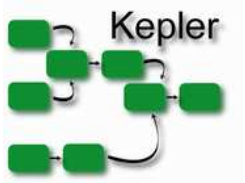
Kepler

Run the workflow



A reminder: Code to KEPLER

- Port your code to the Gateway
- Identify relevant CPOs
- Make a subroutine of your code that has CPOs as input/output or integer, …
- Run it in a test bed to check that the CPOs are correctly implemented.
- Make a library of the routine: mylib.a or mylib.so
- Use FC2K to add your code in $KEPLER
- Include your new actor in a workflow and press run
- And Bob's your uncle (hopefully).

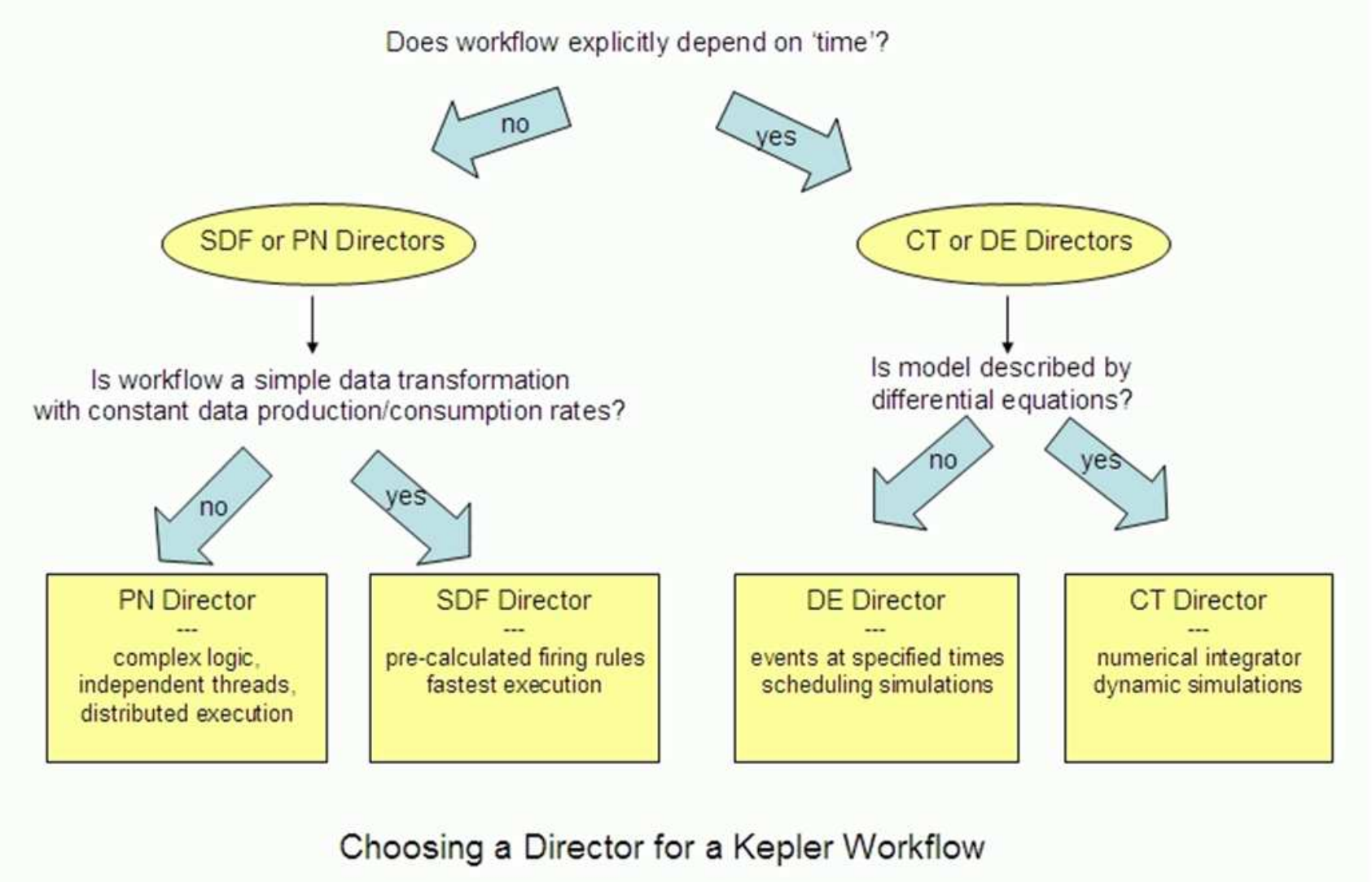


Advanced use of KEPLER

- Directors
- Iterations
- Debug

Kepler

How to choose a director

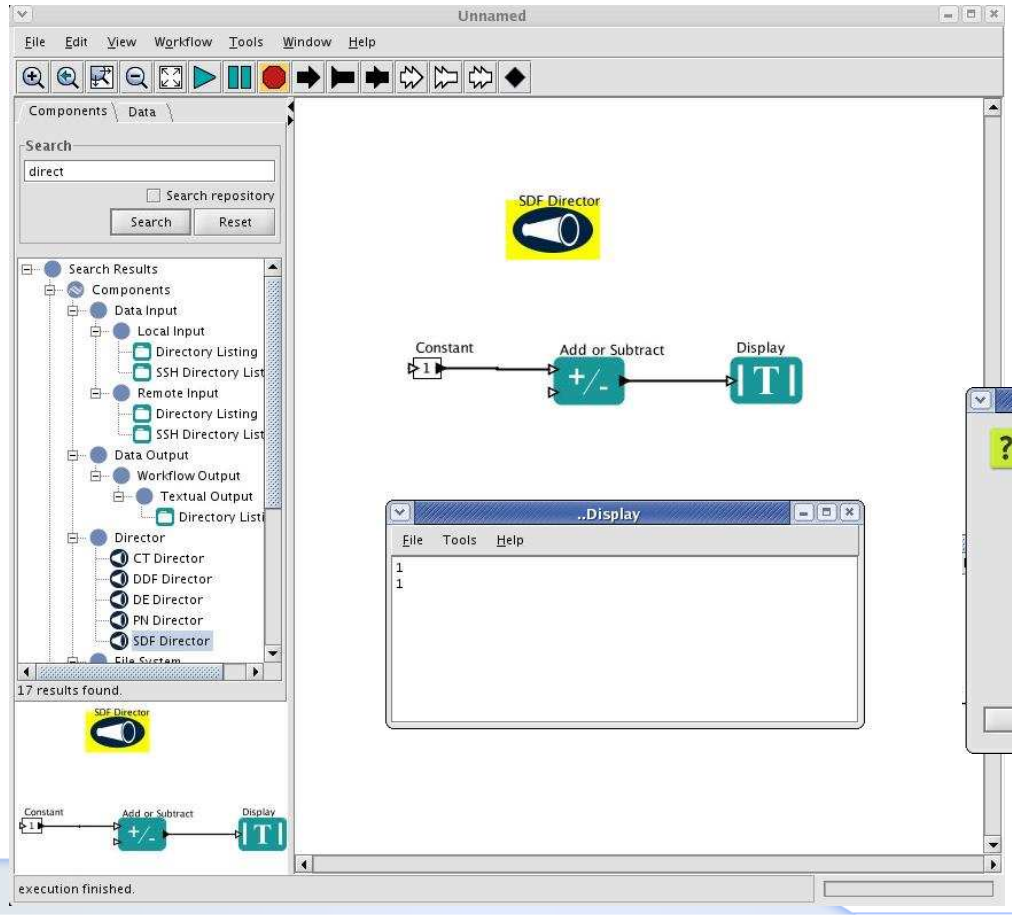


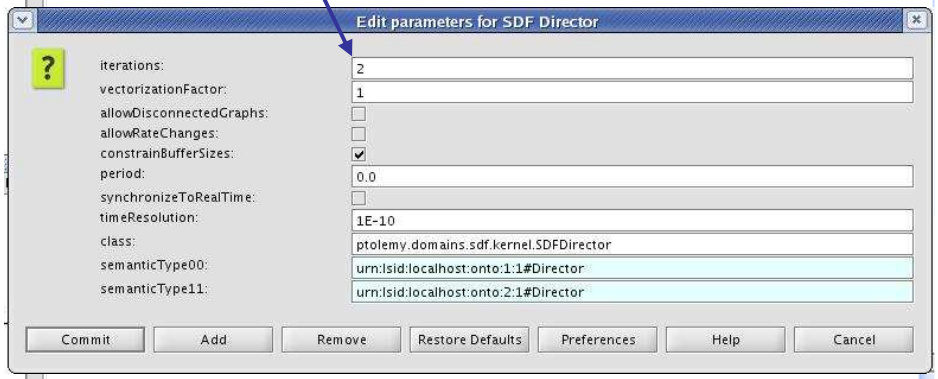
Choosing a Director for a Kepler Workflow

Design of a workflow (1)

Design

1. **Choose your actors (for instance constant, addsubstract & display: use the search cmd)**
2. **Drag & drop them in the design area**
3. **Connect the actors => draw a link between input & output port**
4. **Define the director (SDF) and its parameters (number of iterations)**



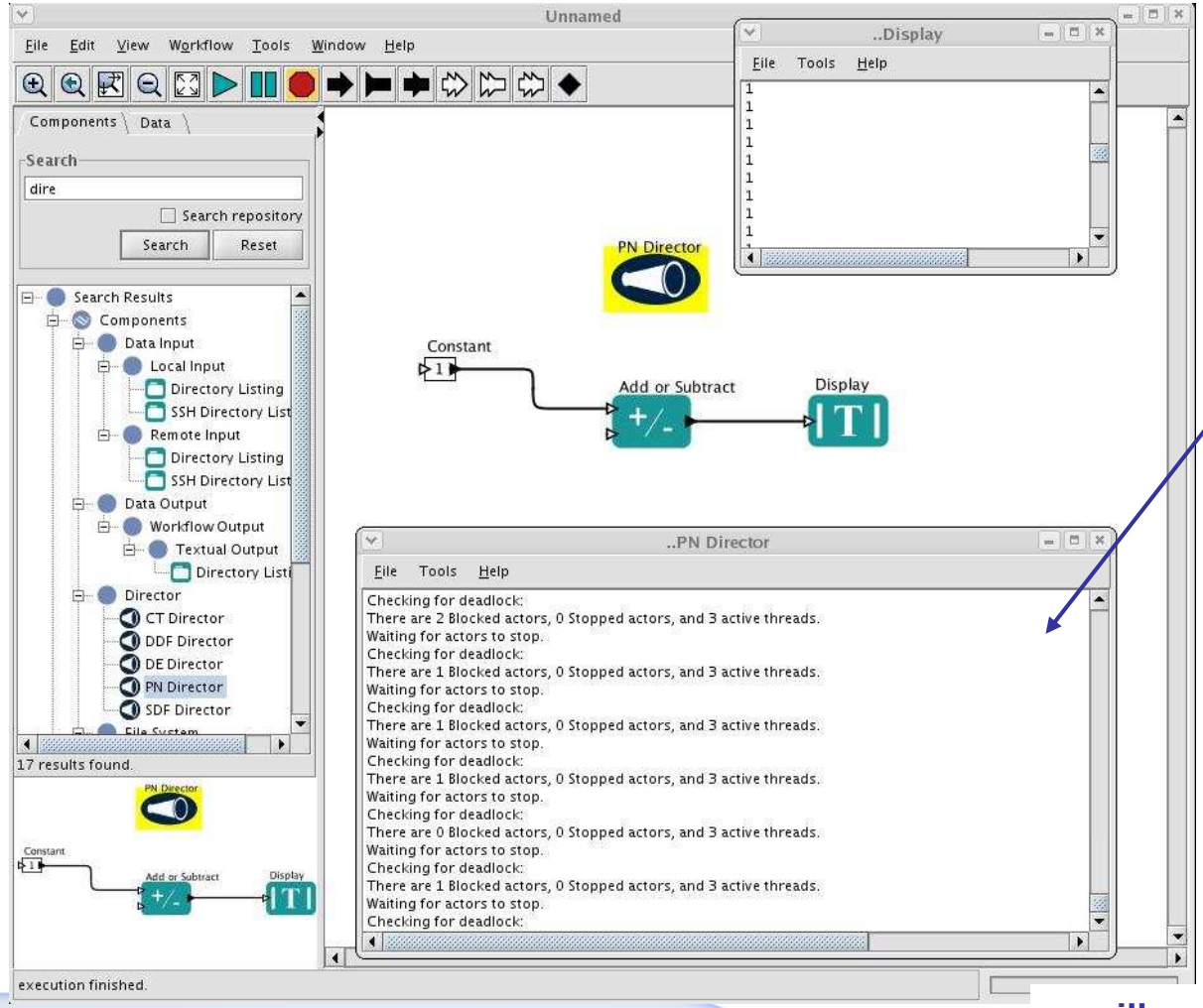


~guillerm/public/training/add_with_SDF.xml

Design of a workflow (2)

Outcome with PN director

1. **With PN, scheduling is done by each thread (actor) => no global scheduling**



Endless

Trick

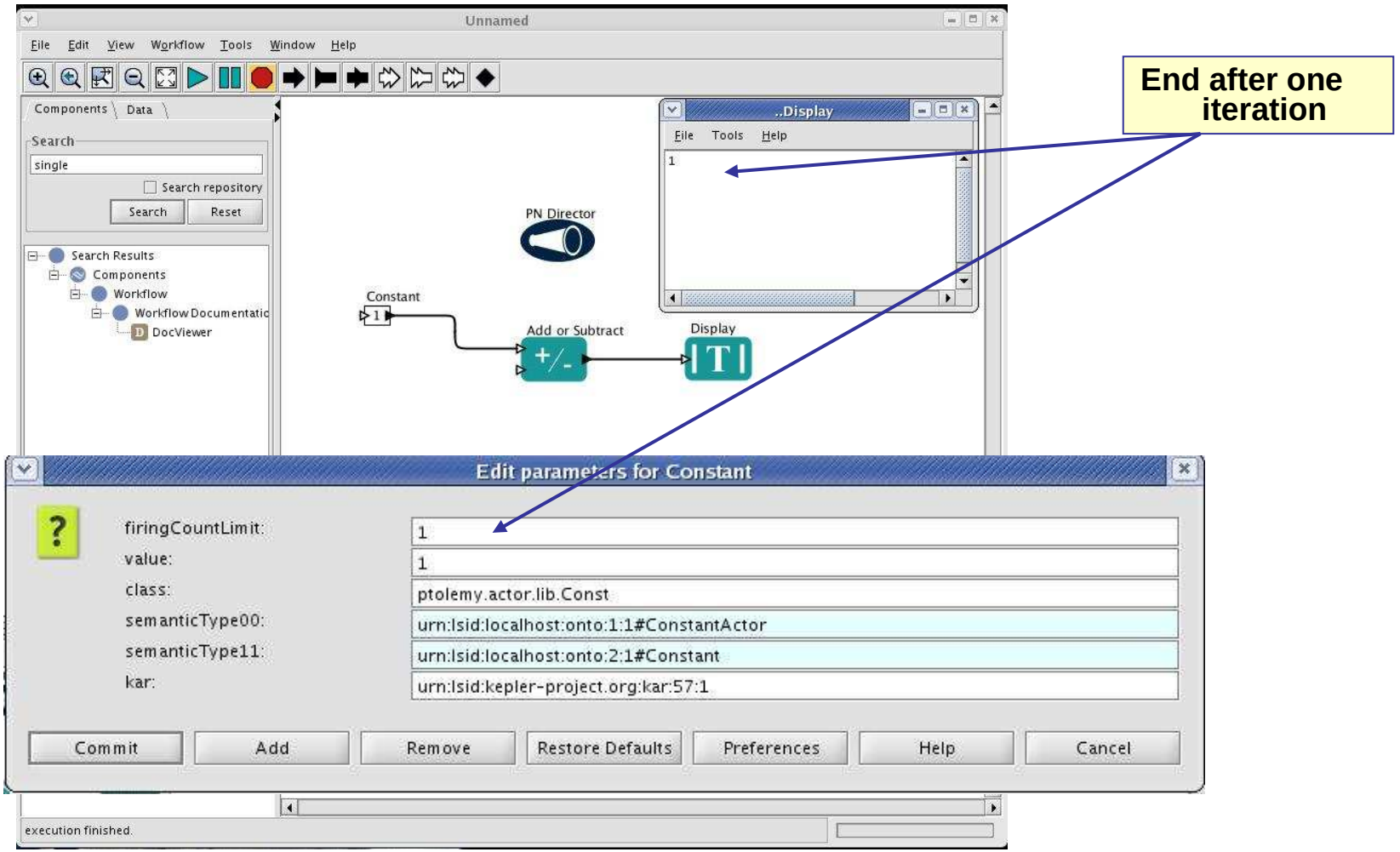
1. **Constant provide a value at each step**
2. **PN will stop if no more datatoken source available**
3. **Use “SingleFireConstant” instead**

~guillerm/public/training/add_with_PN.xml

Design of a workflow (3)

Outcome with PN director

1. With PN & one single fire => OK

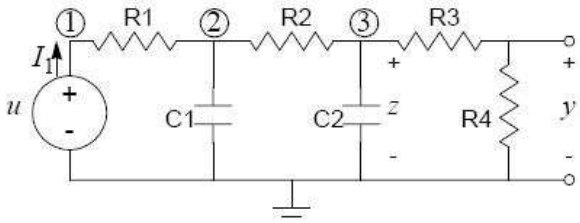


Kepler

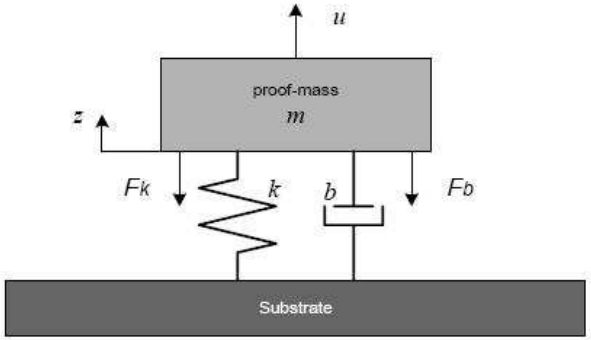
Differential Equations

$$m\ddot{z}(t) + b\dot{z}(t) + kz(t) = u(t) \quad (1)$$
$$y(t) = c \cdot z(t)$$
$$z(0) = 10, \dot{z}(0) = 0.$$

The equations could be a model for an analog circuit as shown in figure 2.1(a), where z is the voltage



(a) A circuit implementation.



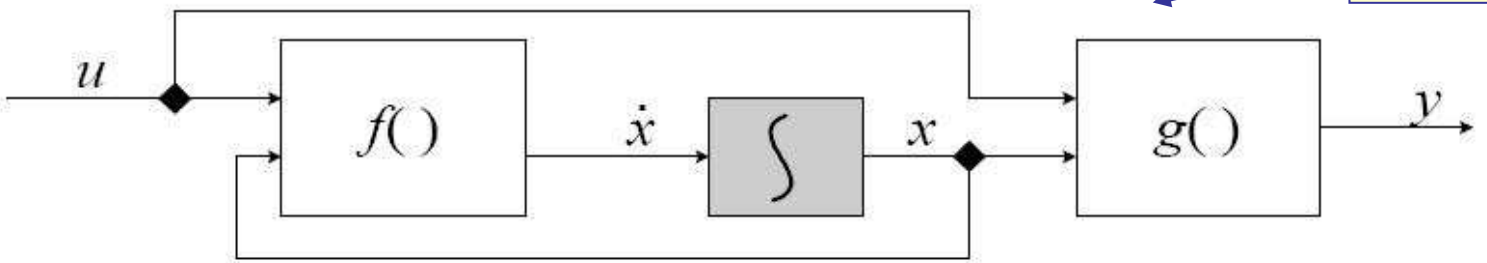
(b) A mechanical implementation.

Could be written as

, an ODE-based continuous-time system has the following form:

$$\dot{x} = f(x, u, t)$$
$$y = g(x, u, t)$$
$$x(t_0) = x_0,$$

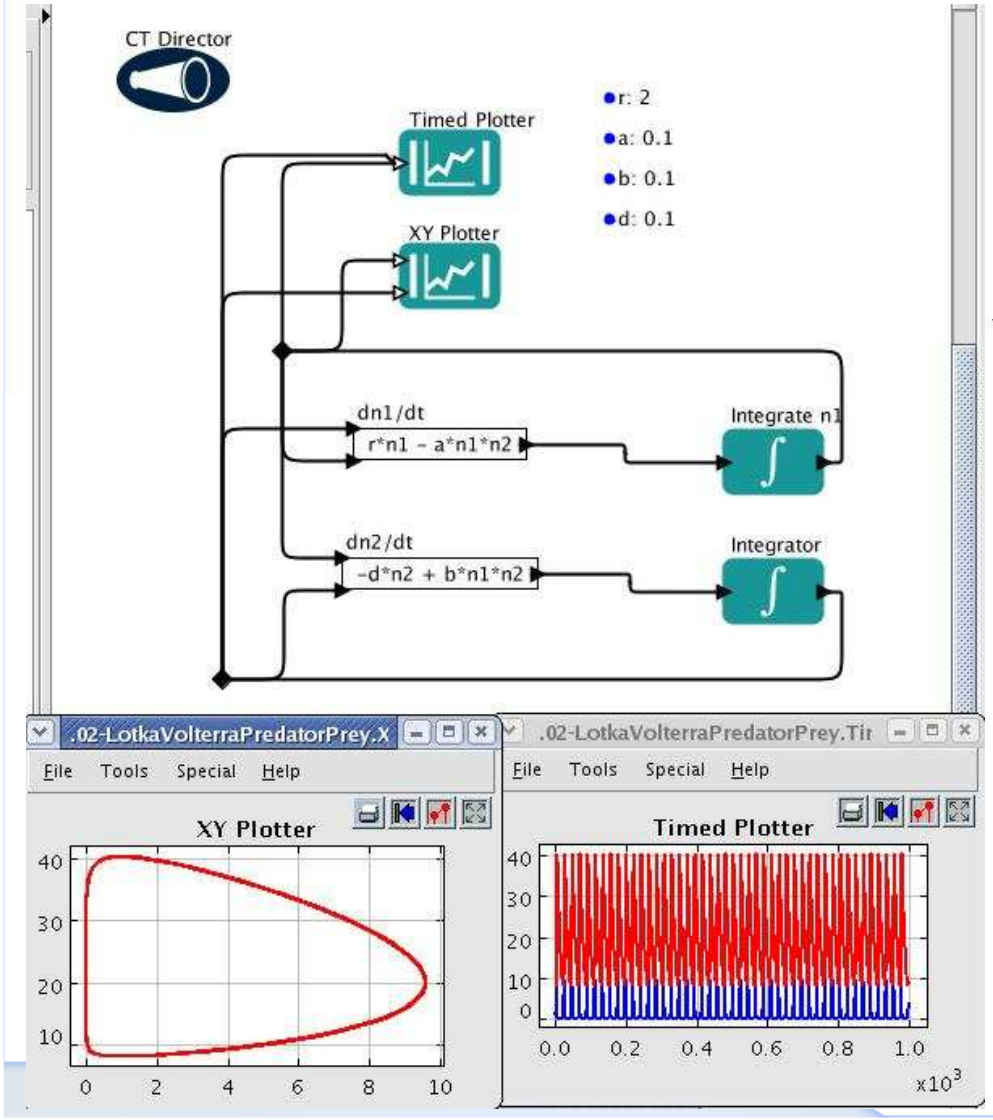
Could be implemented as



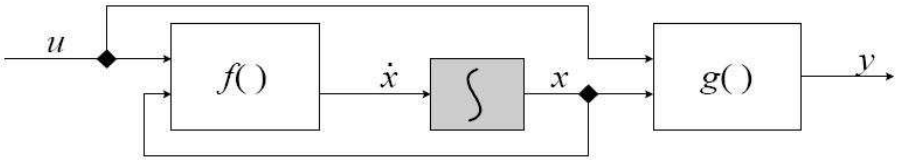
Kepler

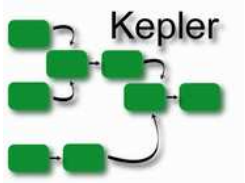
Differential Equations (2)

Using the CT director



Example of 2 coupled differential equations





How to iterate?(1)

Different ways

1. **Iterations in the Director (seen previously)**

2. **Ramp or repeat actors**
   - **Ramp = “for (i=initial;i=i+step;i<final)”**
   - **Repeat = repeat the same input some specified number of times**

3. **Using arrays & Rexpression**

4. **Using composite actors (encapsulate the iterations in a single actor)**

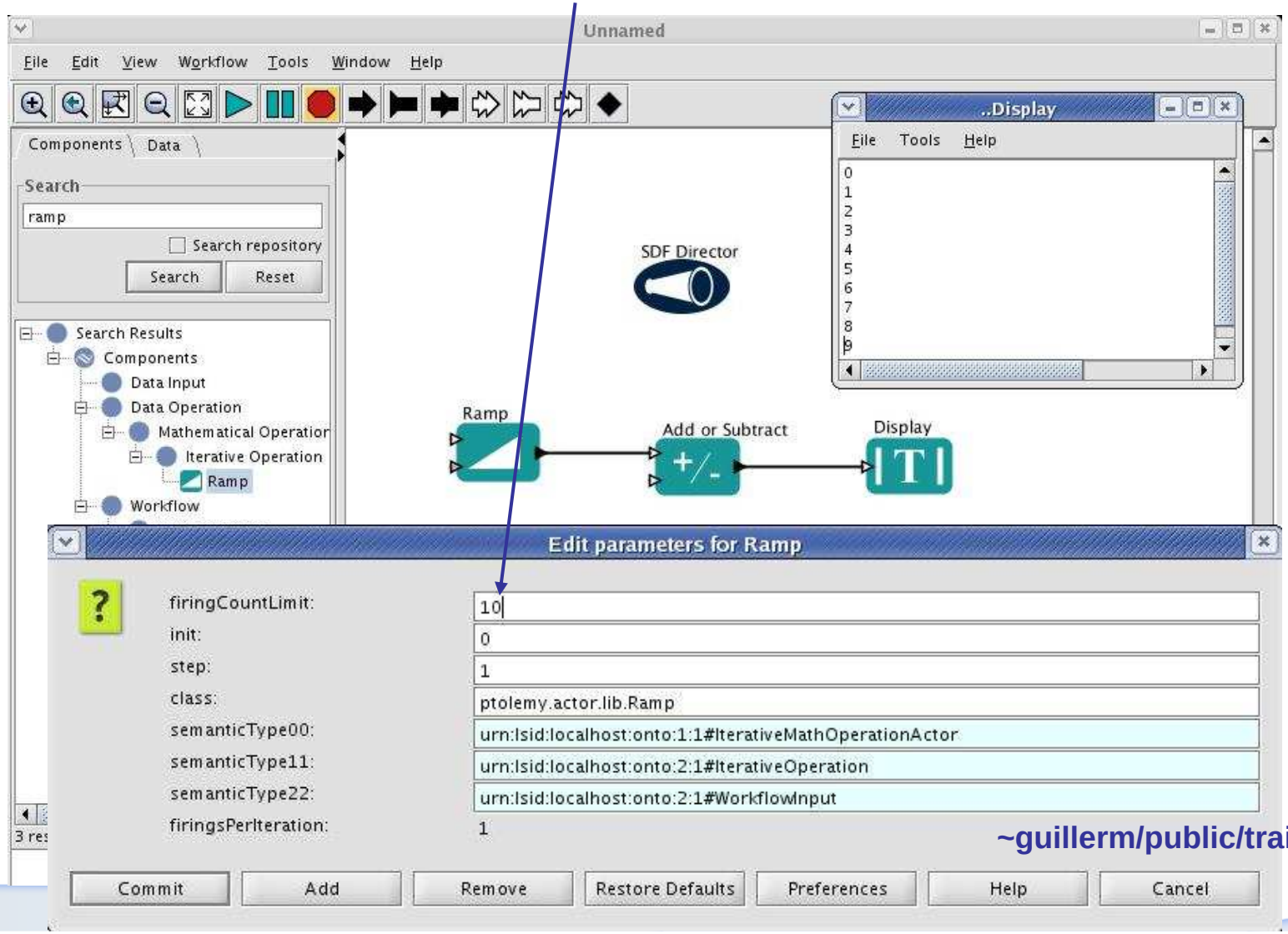
5. **Feedback connections**
   - **Beware: how to start with some directors (SDF for instance)**

6. **Using map (Java)**

Kepler

How to iterate?(2)

Using the RAMP actor (for i=0;i++;i<10)

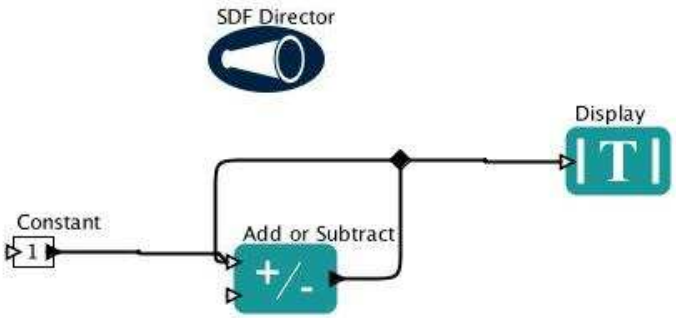


~guillerm/public/training/ramp.xml

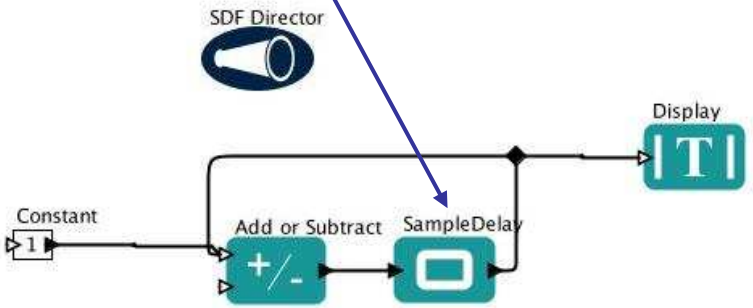
Kepler

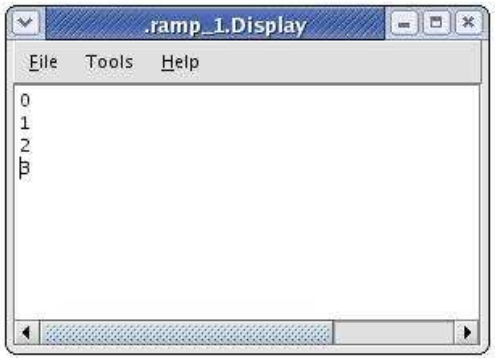
How to iterate?(3)

Using a Feedback connection (SDF)
Don't know how to start => error



Trick:
Add a null time delay => start from sampledelay output (0..3)



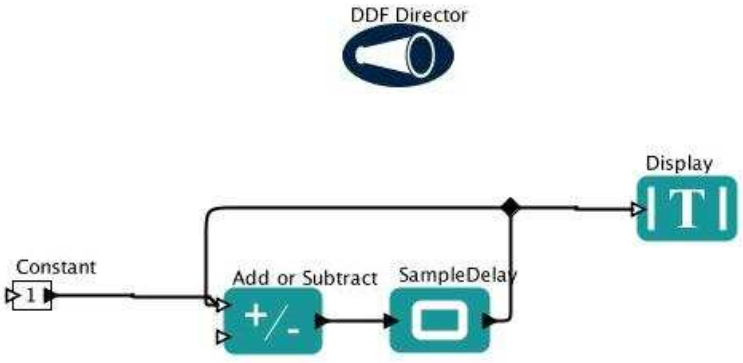


~guillerm/public/training/loop_SDF.xml

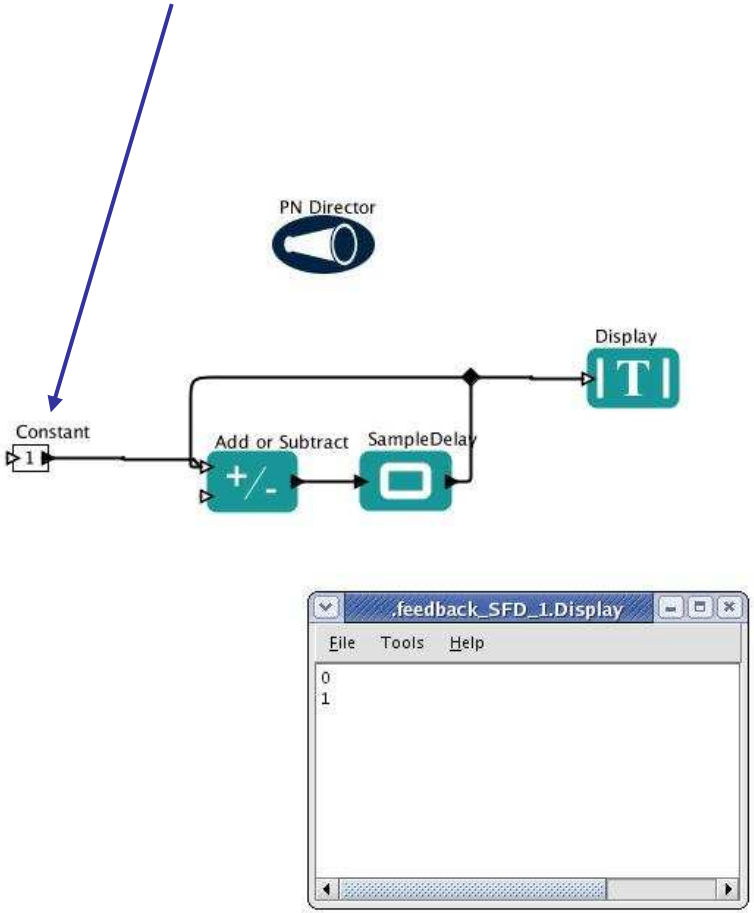
Kepler

How to iterate?(4)

Using a Feedback connection (DDF)
Iterations=4



Using PN:
with single firing constant

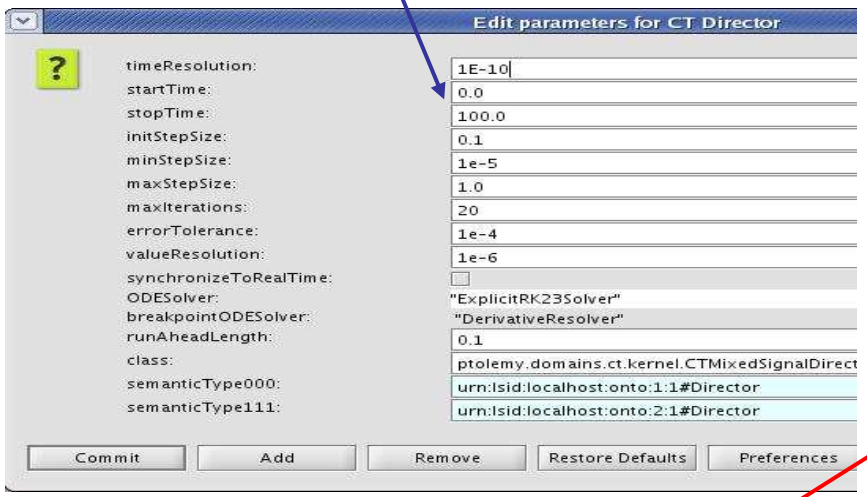


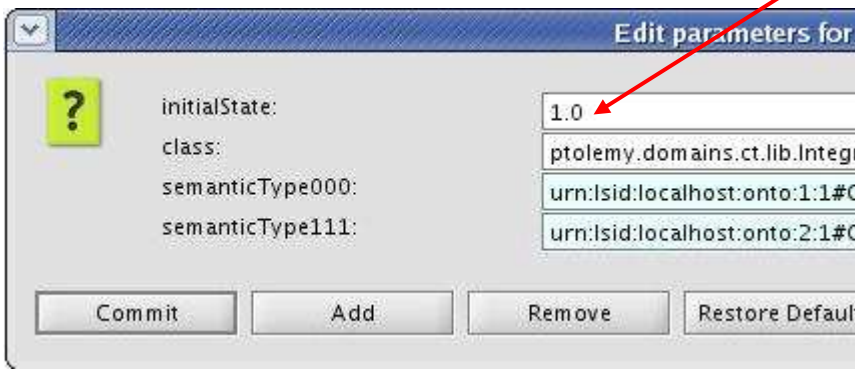
~guillerm/public/training/loop_DDF.xml

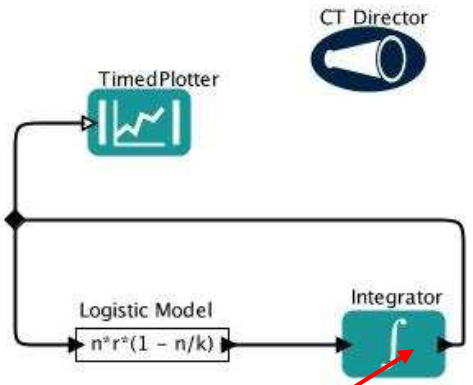
Kepler

How to iterate?(5)

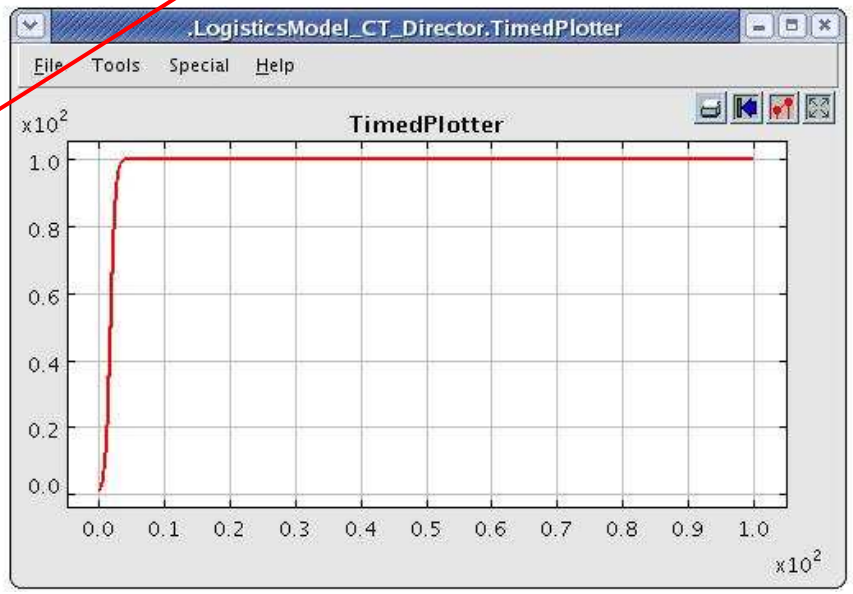
Using a Feedback connection (CT)
Duration=100s, step=0.1s







initial population ● initPop: 1.0
growth factor ● r: 2.6
carrying capacity ● k: 100



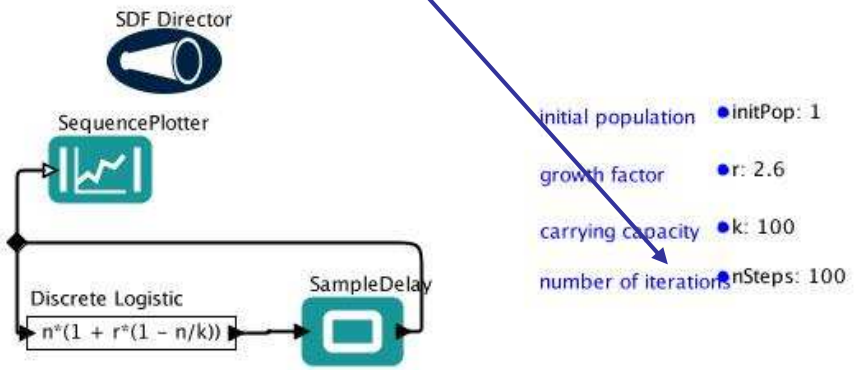
~guillerm/public/training/loop_expression_CT.xml

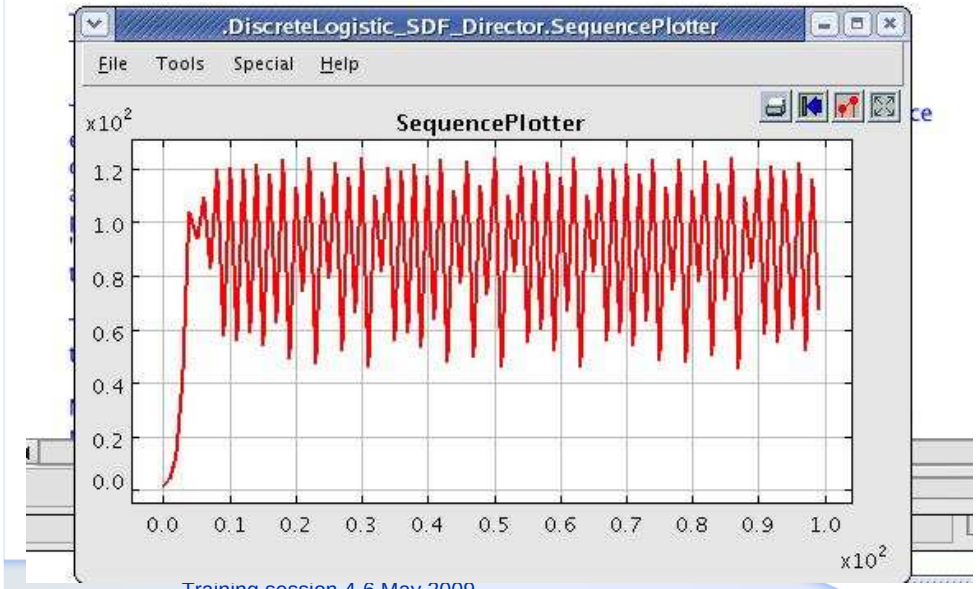
Kepler

How to iterate?(6)

Using a Feedback connection (SDF)

Iterations=100

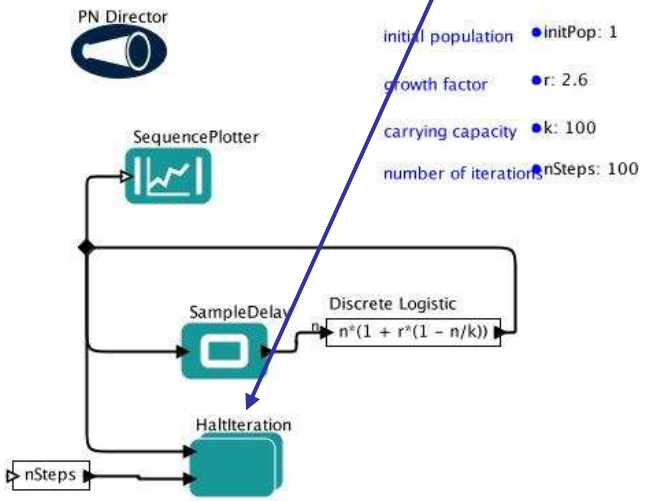


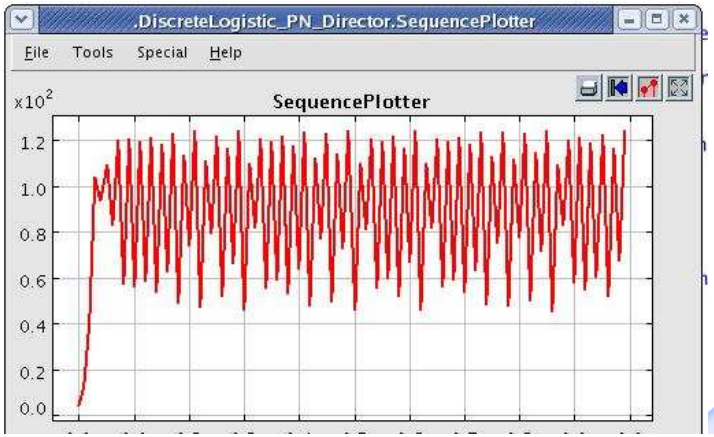


Using a Feedback connection (PN)

Iterations=100 need a stop

See $Kepler/demos/SEEK/DiscreteLogistics_PN_Director.xml

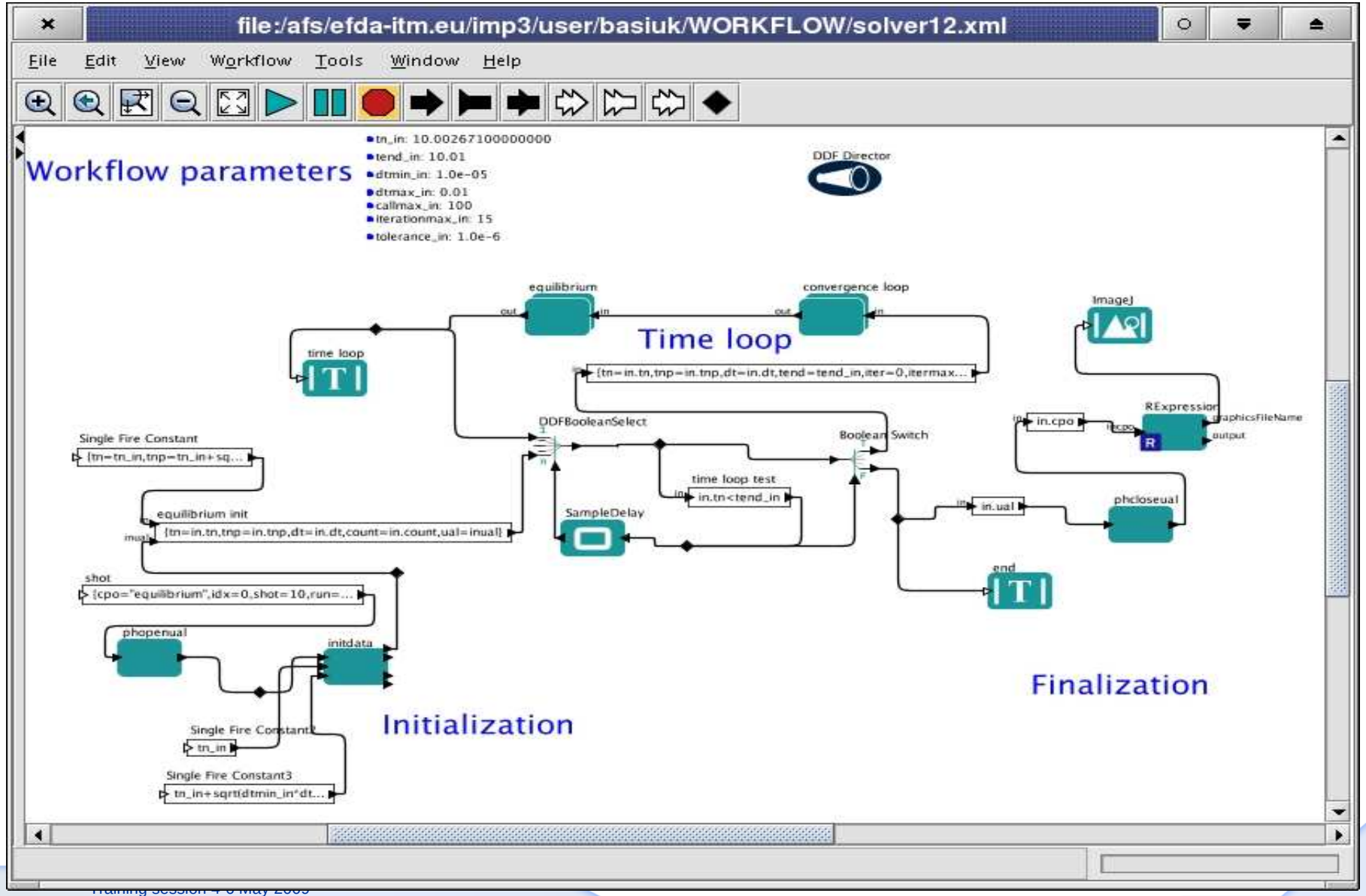




~guillerm/public/training/loop_expression_PN.xml

Kepler

A more complex fusion workflow



How to debug?

Different ways

1. **Animate the workflow**

2. **Listen to:**
   - **Actor**
   - **Port**
   - **Director**

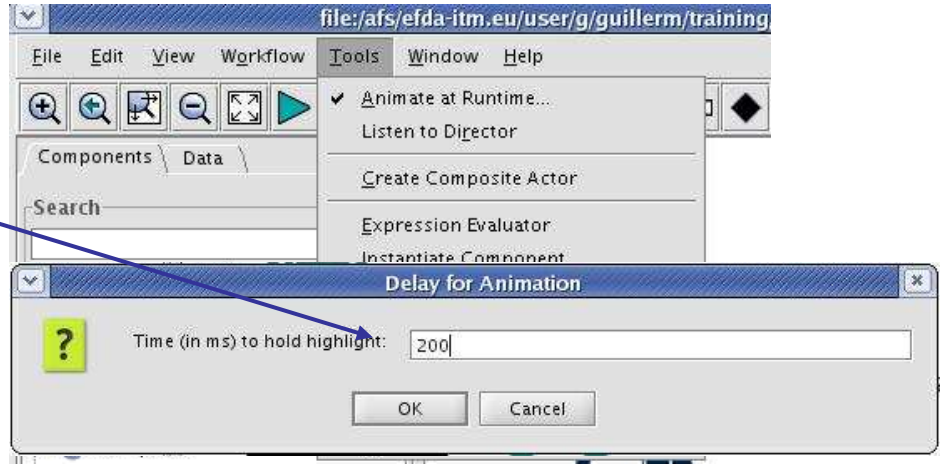
3. **Using the “Provenance” actor**

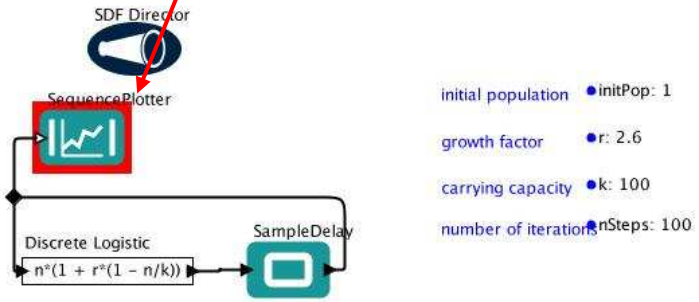
Kepler

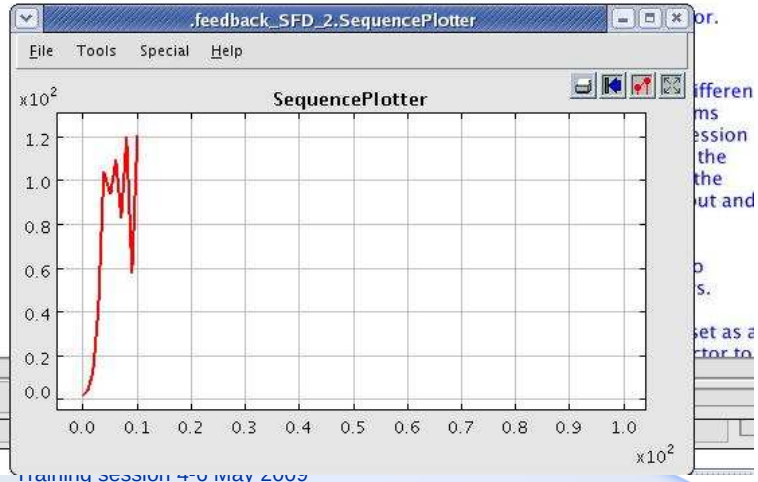
How to debug?(2)

Animate the workflow

- **Specify the time in ms**
- **Run=>active actor is in Red**





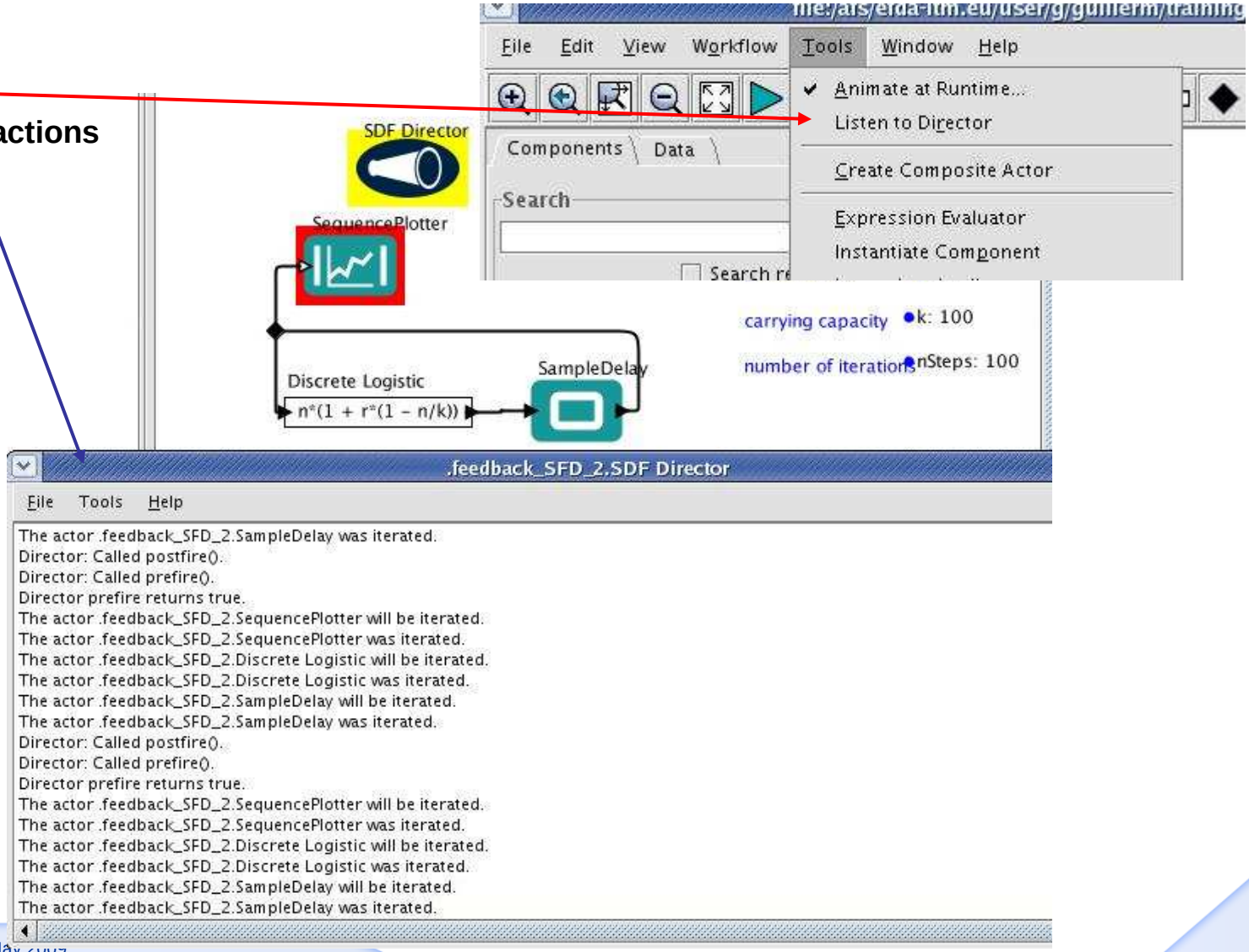


Kepler

How to debug?(3)

Listen to

- **Director**
- **Notify the actions**

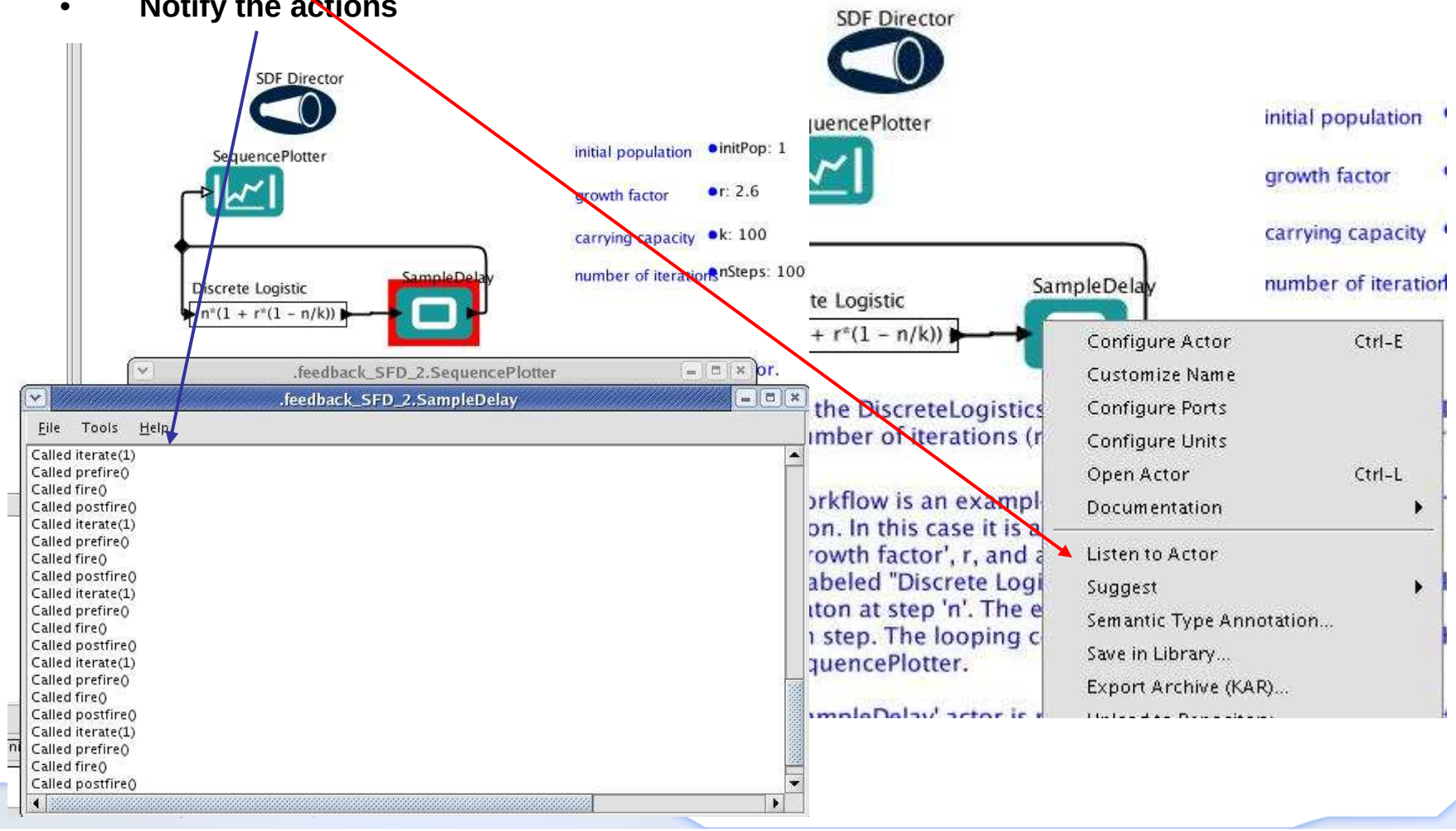


Kepler

How to debug?(4)

Listen to

- **Actors**
- **Notify the actions**



Kepler

How to debug?(5)

Listen to

- **Ports**
- **Show the data**