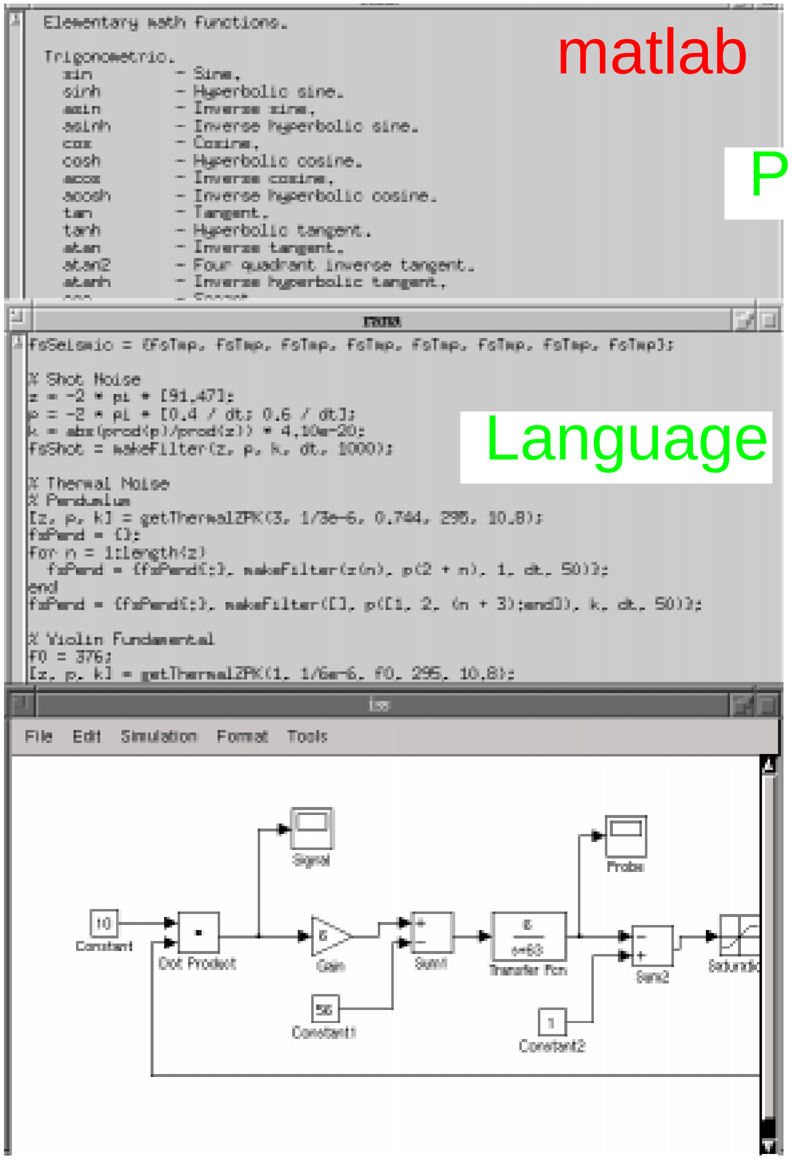
End to End model

Gainesville/FL 3/99   Hiro Yamamoto / Caltech

- Talks given on 3/5,6 in LSC and 3/9 in STAIC
- Outside of End to End model - Hiro Yamamoto
  - What/why is E2E
  - Status
- Inside of End to End model - Biplab Bhawal
  - Optics implementation
- IOO implementation - Sergei Klimenko
- Developed by
  - Base design and core part
    - Mat Evans / CIT
    - Malik Rakhmanov / CIT
    - Hiro Yamamoto / CIT
  - GUI / Alfi
    - Ed Maros / CIT
  - Time domain modal model and summation cavities
    - Biplab Bhawal / CIT
  - Mechanics
    - Somuya Mohanty / Penn
    - Giancarlo Cella / Pisa
  - subsystem development
    - IOO : Sergei Klimenko / Florida
    - PSL : Rick Savage / Hanford , Peter King / CIT

What is E2E

matlab vs. e2e



matlab

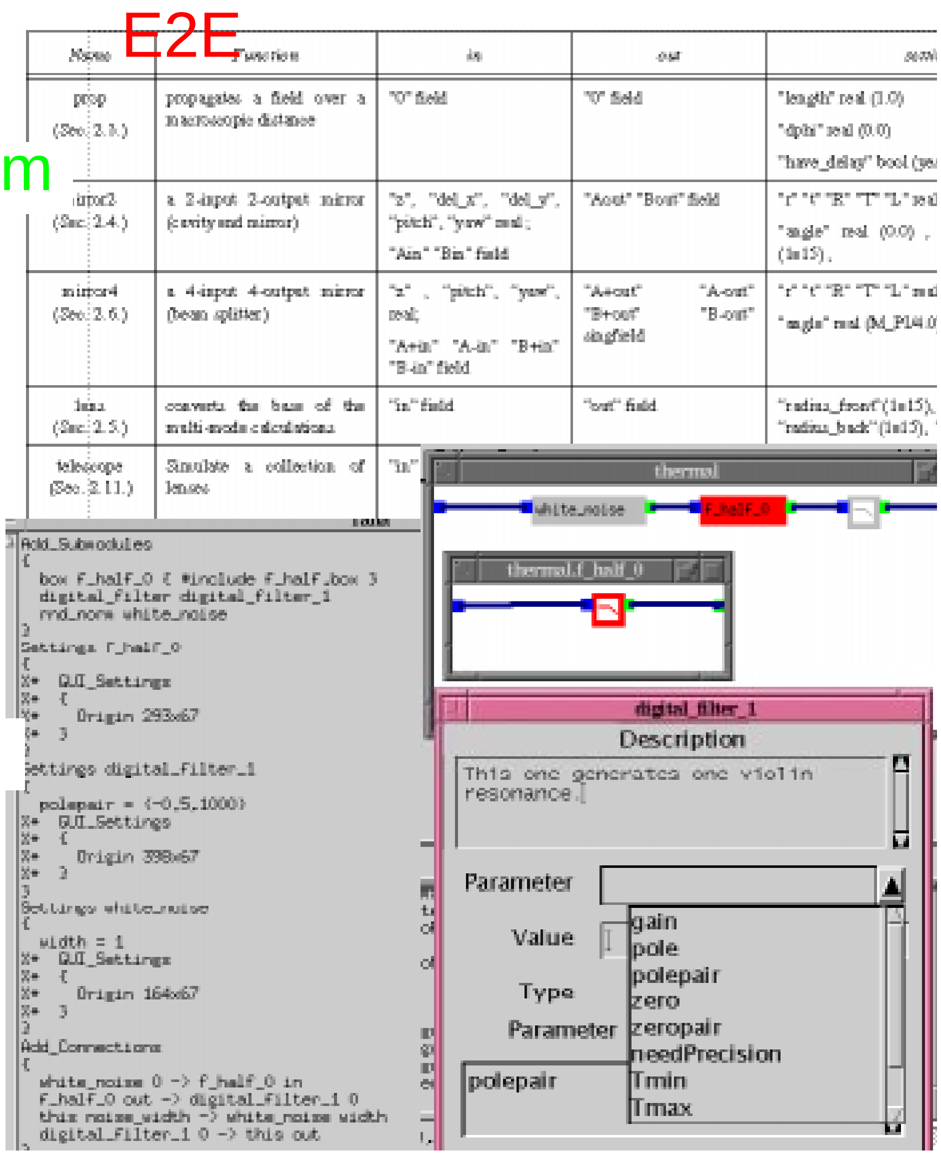
Program

Language

GUI

```
Elementary math functions.

Trigonometric.
  sin         - Sine.
  sinh        - Hyperbolic sine.
  asin        - Inverse sine.
  asinh       - Inverse hyperbolic sine.
  cos         - Cosine.
  cosh        - Hyperbolic cosine.
  acos        - Inverse cosine.
  acosh       - Inverse hyperbolic cosine.
  tan         - Tangent.
  tanh        - Hyperbolic tangent.
  atan        - Inverse tangent.
  atan2       - Four quadrant inverse tangent.
  atanh       - Inverse hyperbolic tangent.
  sec         - Secant.
```

```
fsSeismic = {fsTmp, fsTmp, fsTmp, fsTmp, fsTmp, fsTmp, fsTmp, fsTmp};

% Shot Noise
z = -2 * pi * [91.47];
p = -2 * pi * [0.4 / dt; 0.6 / dt];
k = abs(prod(p)/prod(z)) * 4.10e-20;
fsShot = makeFilter(z, p, k, dt, 1000);

% Thermal Noise
% Pendulum
[z, p, k] = getThermalZPK(3, 1/3e-6, 0.744, 295, 10.8);
fsPend = {};
for n = 1:length(z)
  fsPend = {fsPend{:}, makeFilter(z(n), p(2 + n), 1, dt, 50)};
end
fsPend = {fsPend{:}, makeFilter([], p([1, 2, (n + 3):end]), k, dt, 50)};

% Violin Fundamental
f0 = 376;
[z, p, k] = getThermalZPK(1, 1/6e-6, f0, 295, 10.8);
```



E2E

Name	Function	in	out	sett
prop (Sec. 2.3.)	propagates a field over a macroscopic distance	"0" field	"0" field	"length" real (1.0) "dphi" real (0.0) "have_delay" bool (ye
mirror2 (Sec. 2.4.)	a 2-input 2-output mirror (cavity and mirror)	"x", "del_x", "del_y", "pitch", "yaw" real; "Ain" "Bin" field	"Aout" "Bout" field	"r" "t" "R" "T" "L" real "angle" real (0.0) , (1e15) ,
mirror4 (Sec. 2.6.)	a 4-input 4-output mirror (beam splitter)	"x" , "pitch", "yaw", real; "A+in" "A-in" "B+in" "B-in" field	"A+out" "A-out" "B+out" "B-out" singfield	"r" "t" "R" "T" "L" real "angle" real (M_PI/4.0
lens (Sec. 2.5.)	converts the base of the multi-mode calculations	"in" field	"out" field	"radius_front" (1e15), "radius_back" (1e15),
telescope (Sec. 2.11.)	Simulate a collection of lenses	"in"		

```
Add_Submodules
{
  box f_half_0 { #include f_half.box }
  digital_filter digital_filter_1
  rnd_norm white_noise
}
Settings f_half_0
{
%*  GUI_Settings
%*  {
%*    Origin 293x67
%*  }
}
Settings digital_filter_1
{
  polepair = (-0.5,1000)
%*  GUI_Settings
%*  {
%*    Origin 398x67
%*  }
}
Settings white_noise
{
  width = 1
%*  GUI_Settings
%*  {
%*    Origin 164x67
%*  }
}
Add_Connections
{
  white_noise 0 -> f_half_0 in
  f_half_0 out -> digital_filter_1 0
  this noise_width -> white_noise width
  digital_filter_1 0 -> this out
}
```

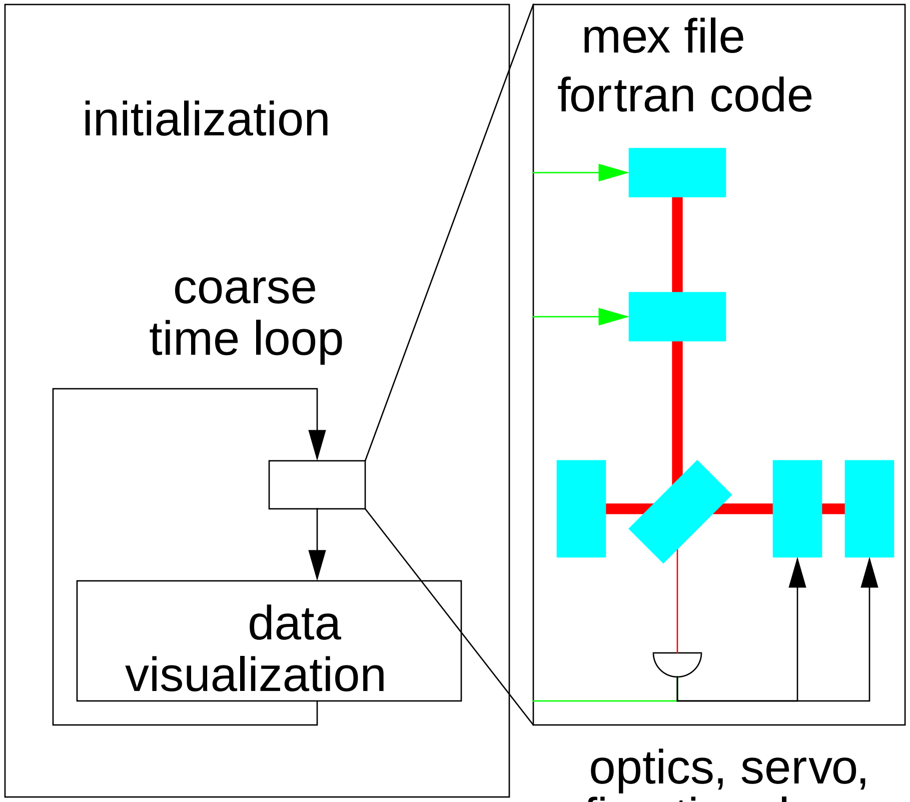
Description: This one generates one violin resonance.

Parameter: polepair

Why E2E
smac vs e2e

SMAC in matlab

could not utilize the power of matlab

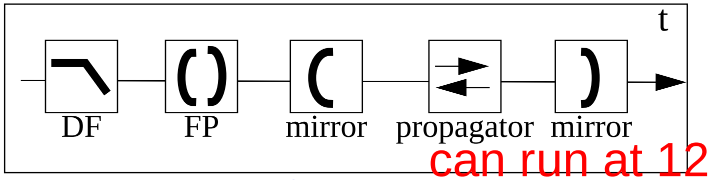


run at 64bit

optics, servo,
fine time loop
hardcoded in here

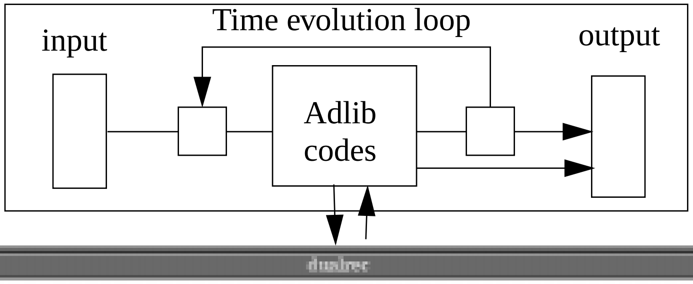
IFO simulation in E2E

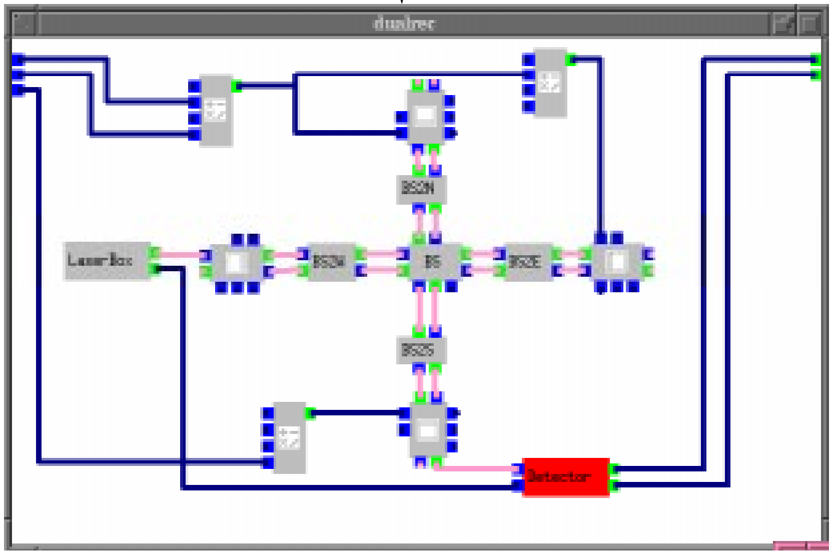
adlib framework / e2e toolbox for Interferometer



can run at 128bit

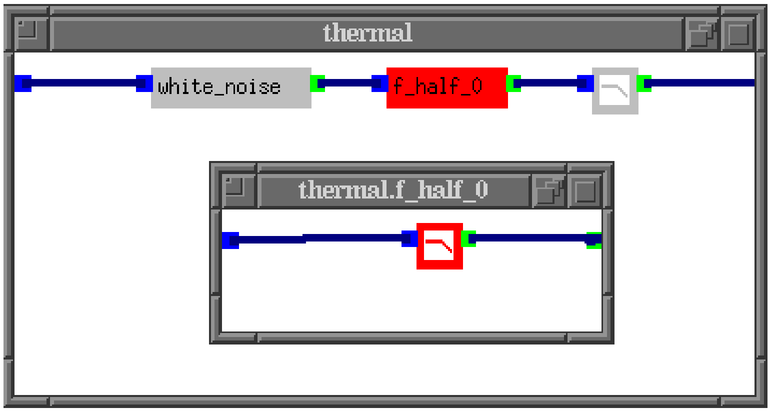
time loop customized for time domain simulation

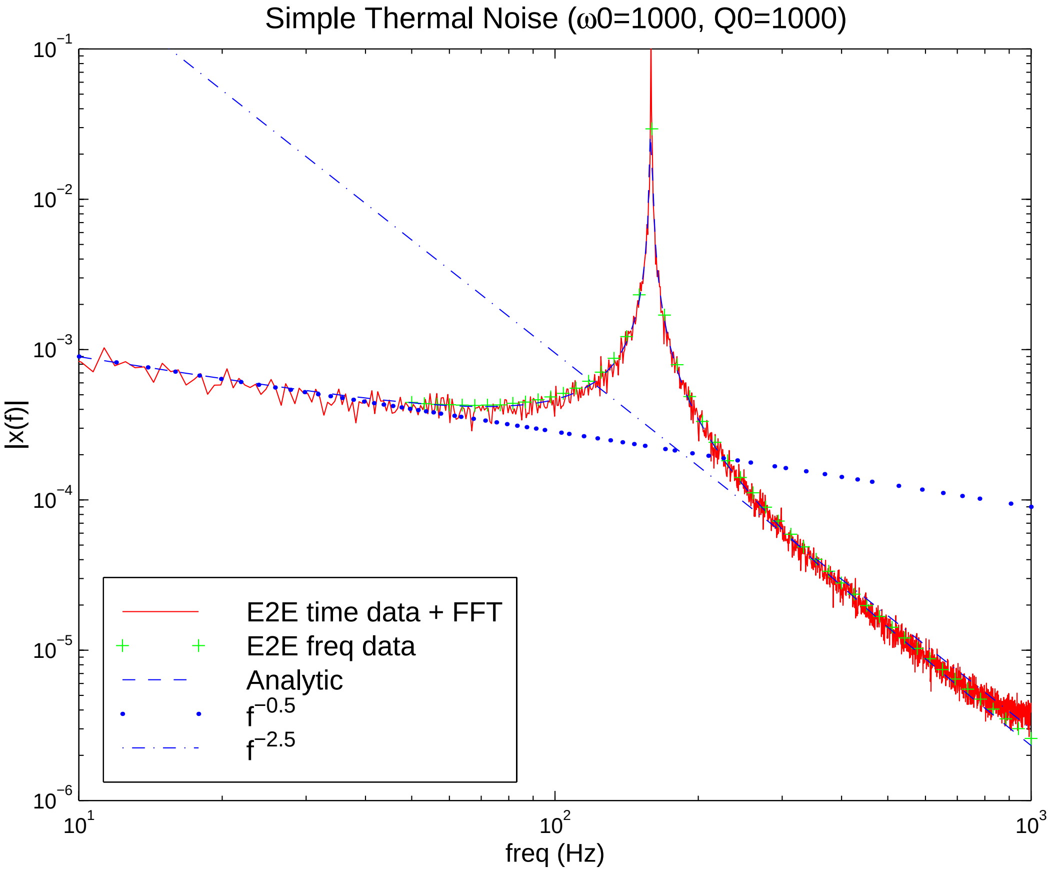




What can be done

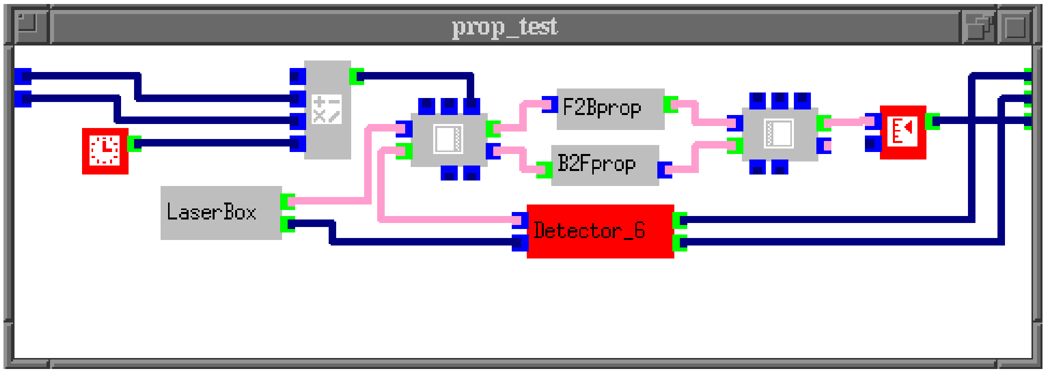
Thermal noise using digital filter

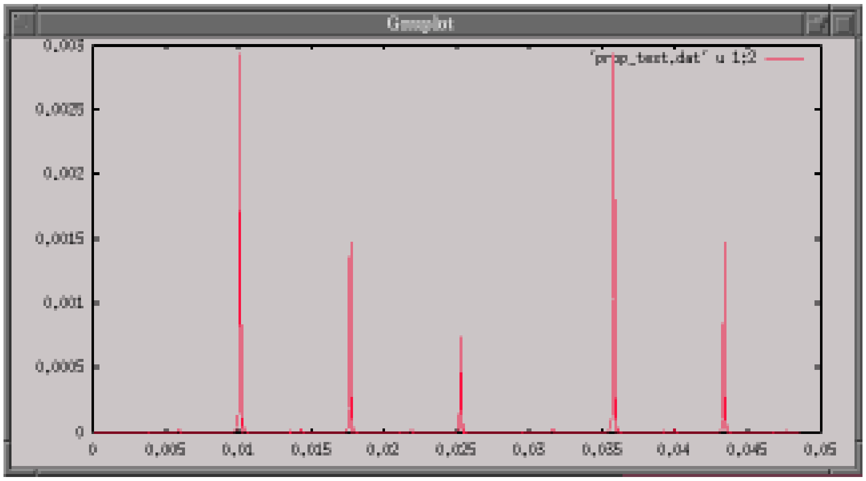


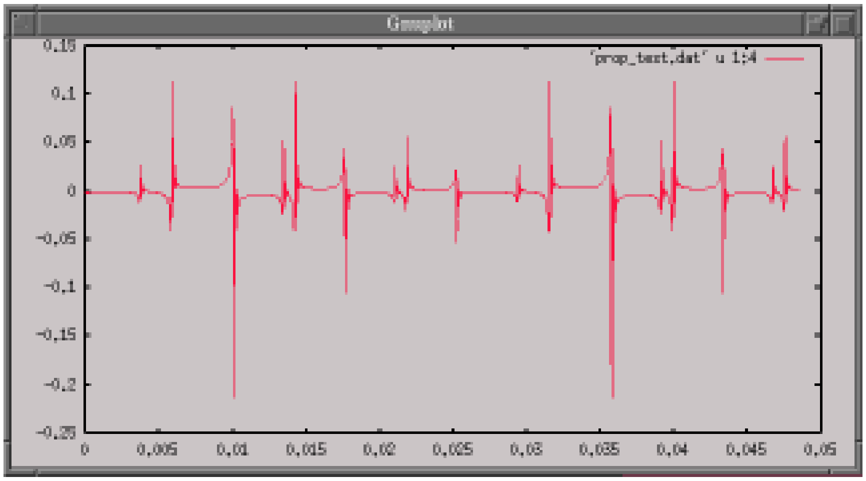


What can be done

FP with mode

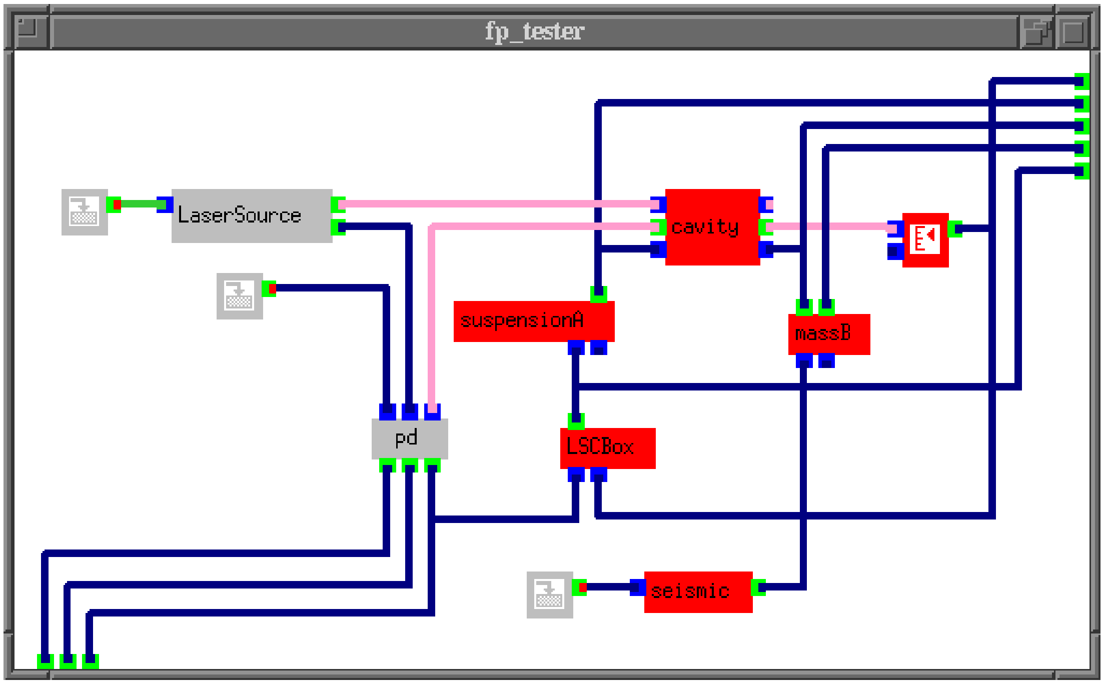


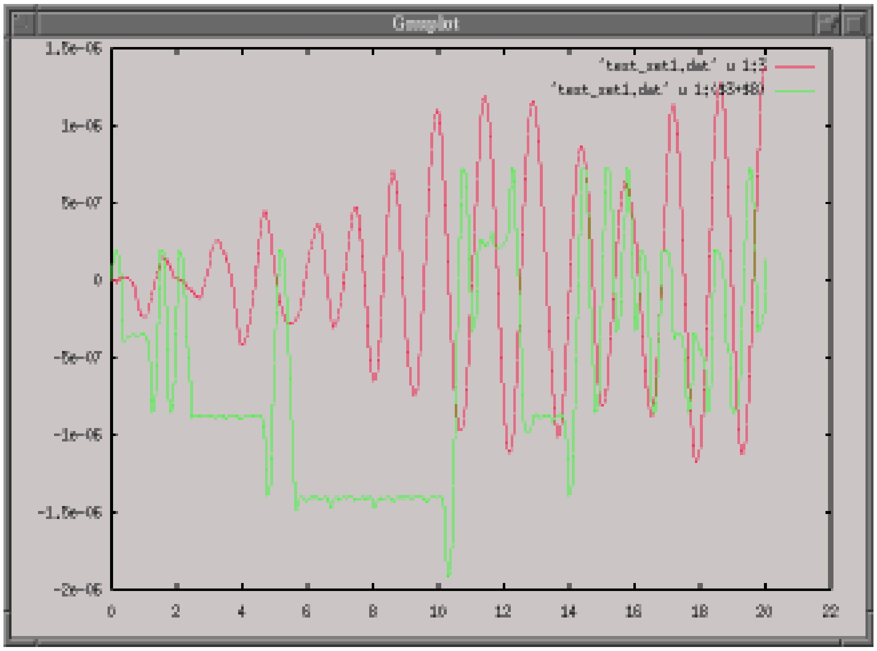




What can be done

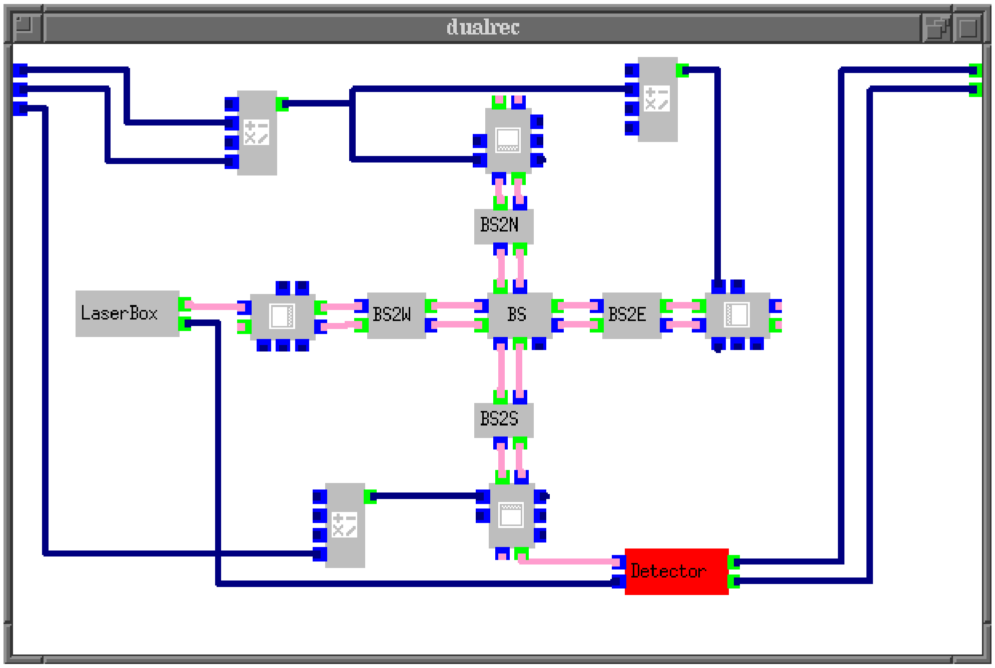
Lock Acuisition

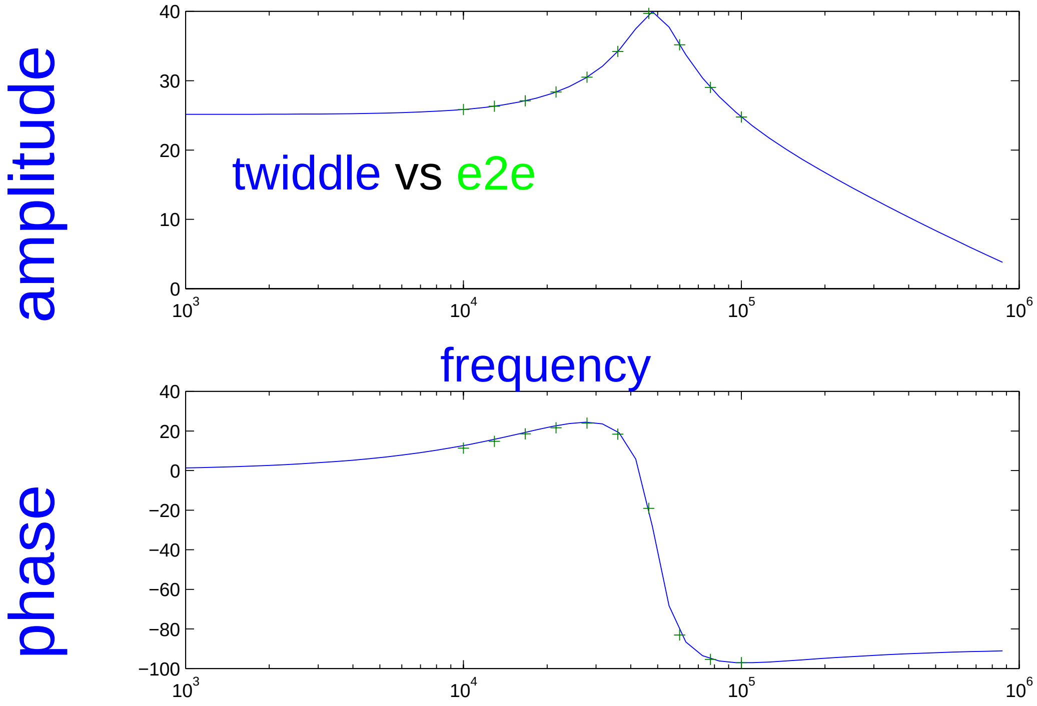




What can be done

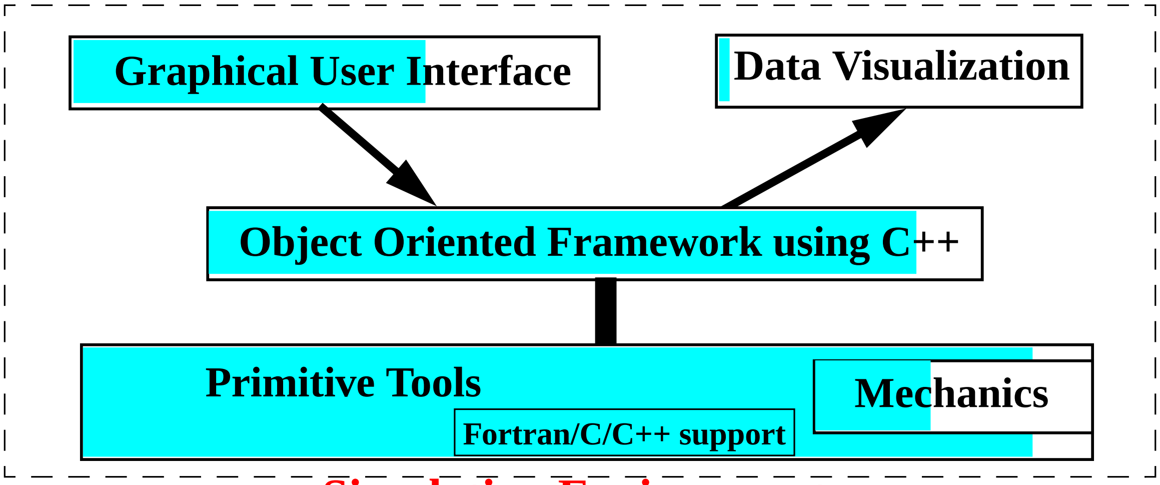
time domain simulation of dual recycling conf.





Status overview

e2e simulation



Graphical User Interface

Data Visualization

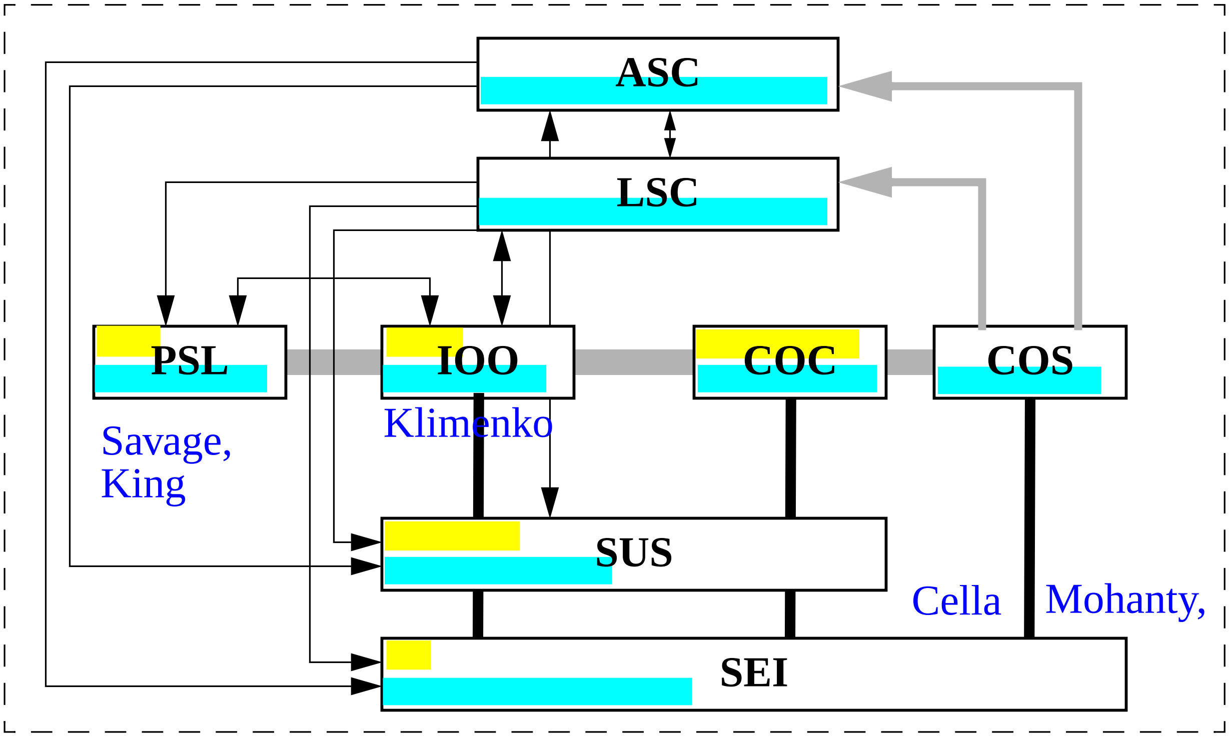
Object Oriented Framework using C++

Primitive Tools

Fortran/C/C++ support

Mechanics

Simulation Environment



ASC

LSC

PSL

IOO

COC

COS

Savage,
King

Klimenko

SUS

Cella

Mohanty,

SEI

Subsystem Implementation

🟦 Primitive tools completed

🟨 Explicit construction completed

Distribution

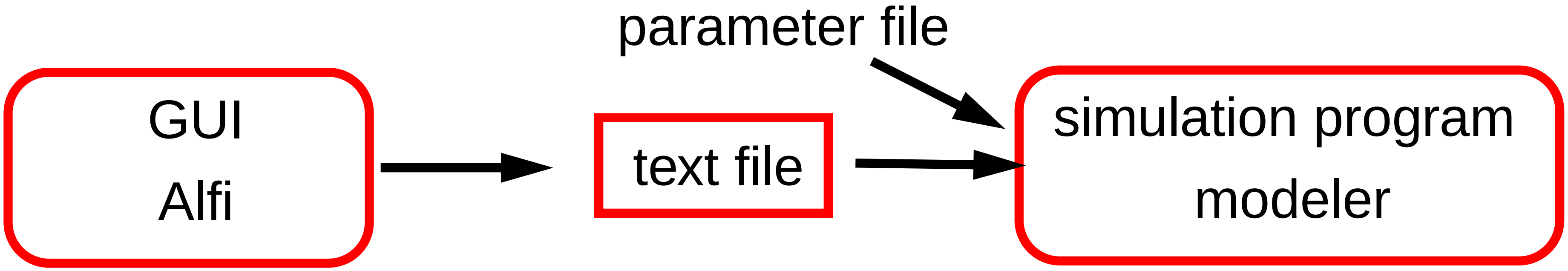
- Necessary Computer Resource to use e2e package
  - » If you have SUN SparcStation, you don’t need any just to use the program. If not, motif/lestif/windows needed.
  - » most likely, you do not need to rebuild e2e programs, just as you do not compile matlab
  - » to build the simulation code
    - – ansi standard C++ compiler, egcs used internally
    - – dynamic linking will be supported
  - » to compile alfi
    - – xwWindow graphics package - free
  - » proper MOU/Attachment
- Distributed site
  - » Hanford - program only
  - » Livingston - program only
  - » Florida - program and source code
  - » MIT - to be installed

Distribution - 2
for now

- Physics & programming
  - LIGO Subsystem code development needs lots of help by subsystem experts.
  - When a new development starts, we will discuss about the basic physics and code implementation.
  - We will keep in touch to exchange the up-to-date knowledge.
  - Documentation will be updated and maintained in the e2e web page. Need help.
  - Bug report handling will be established.
  - Be collaborative, cooperative, patient
- Source code maintained at and distributed from CIT / LIGO
  - Code and program structure are not finalized.
  - No systematic maintenance plan established.
  - CVS at CIT will keep the official source code.
  - New code should be submitted to CIT/LIGO and will be included in the CIT/LIGO CVS.
  - Do not distribute binaries nor source codes.
  - Any party interested in e2e should contact CIT/LIGO.

Program structure overview

adlib = simulation and alfi = GUI



parameter file

GUI

Alfi

text file

simulation program

modeler

alfi does not know anything about primitives.

primitive editor

xxx.prm

xxx.xbm : icon
# inputs
# outputs
input types
output types
comments

description of setup to simulation

separate implimentation

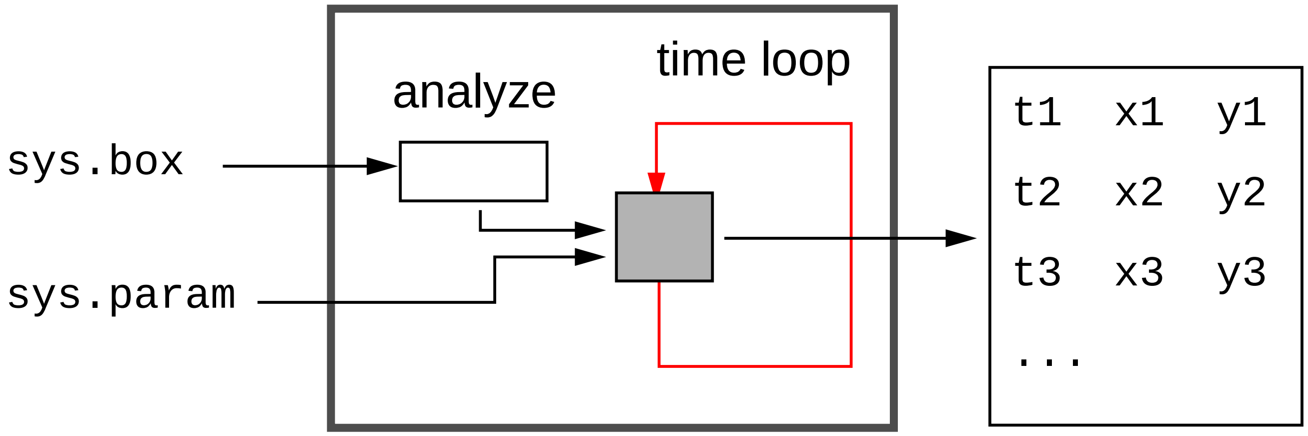
adlib does not know anything about primitives

* static link option
  in main program
    add_submodule(xxx())
    add_submodule(yyy())
    ... [all primitives]

* dynaic link option
    place xxx.so file
        in proper place

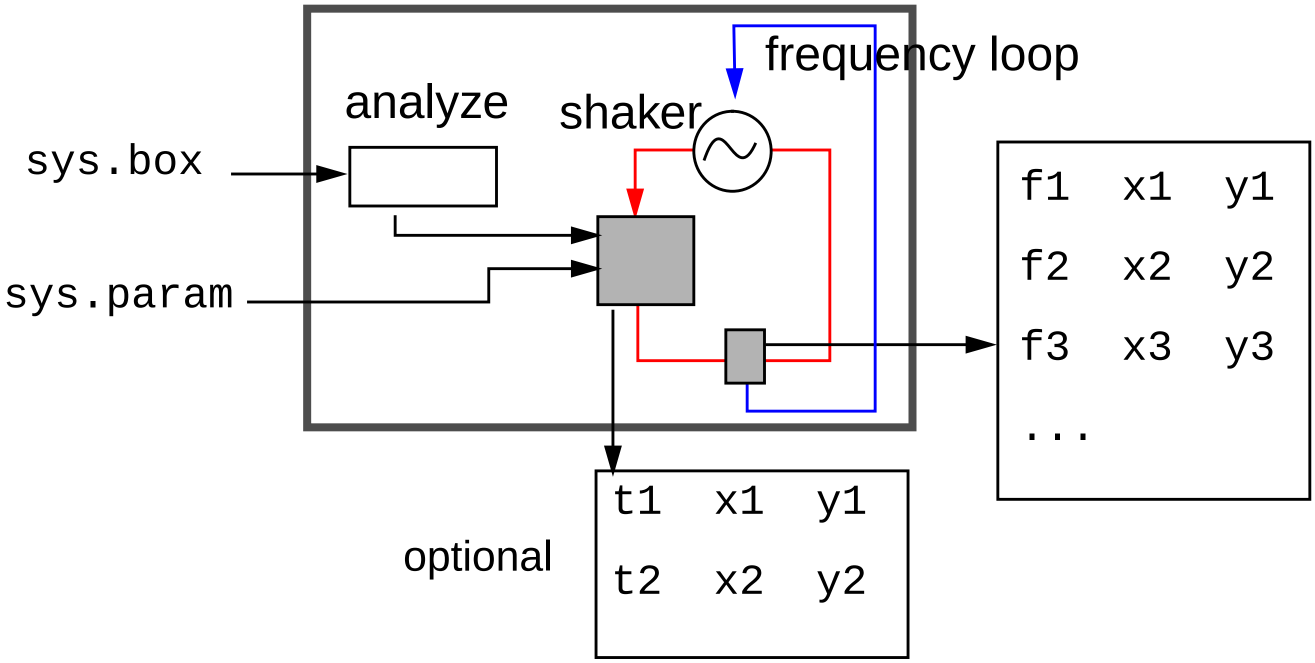
Program structure overview

modeler - time series generator



t1 x1 y1
t2 x2 y2
t3 x3 y3
...

modeler_freq - frequency series generator

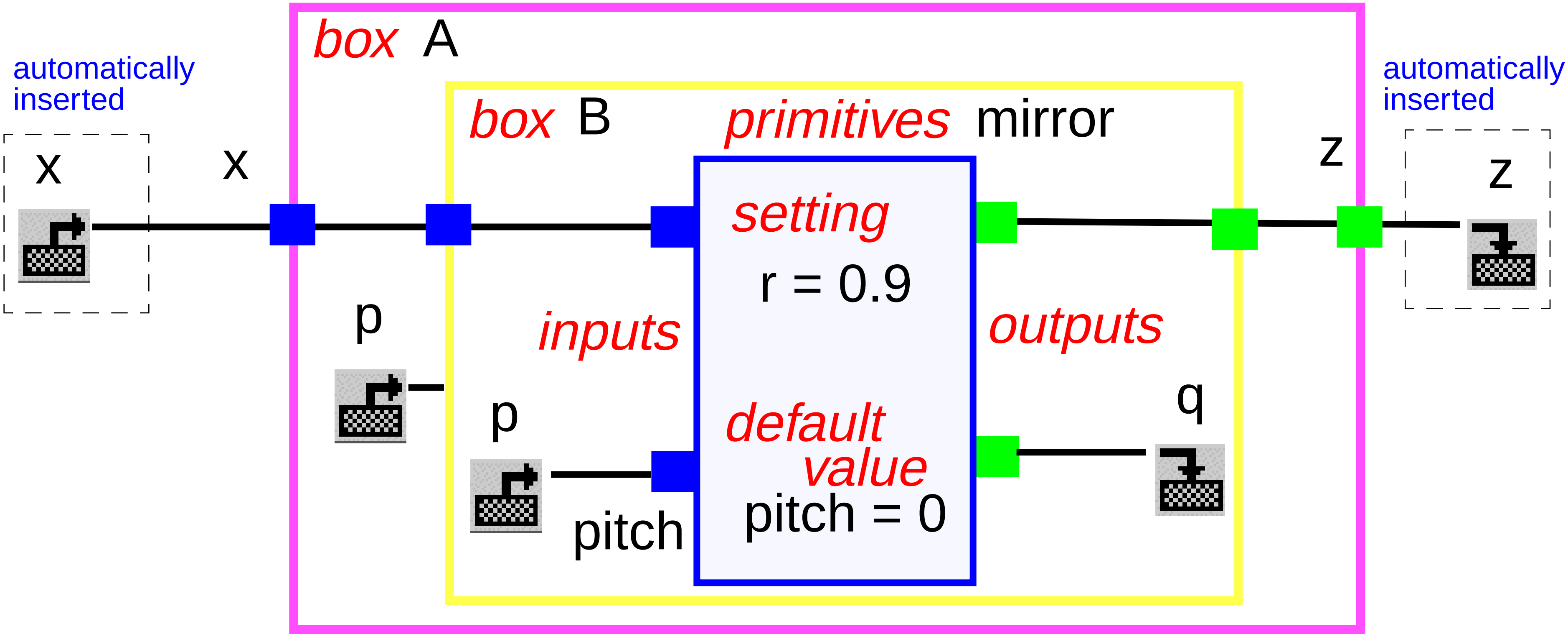


f1 x1 y1
f2 x2 y2
f3 x3 y3
...

t1 x1 y1
t2 x2 y2

Data I/O

how to set parameters



input : may change during run

setting : does not change during run

data_in : value source

data_out -> output file
data_viewer -> console

Parameter file

x = (1,2),(3,2)   (for vector_complex)

p=1   or   B:p = 1 or   A:B:p = 1

output file

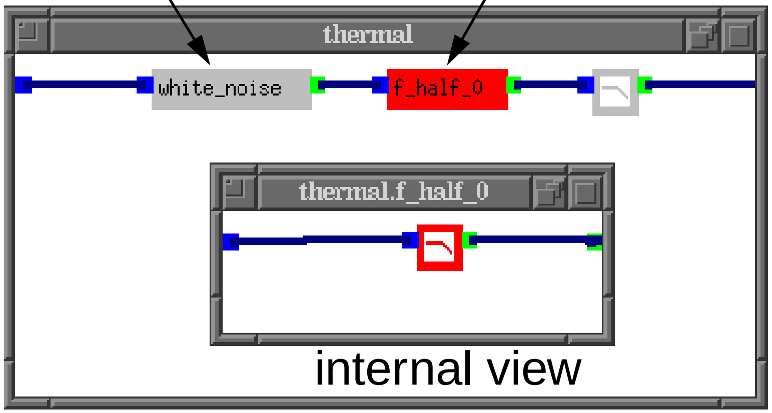
t1	z1	q1
t2	z2	q2
t3	z3	q3

Objects in Adlib

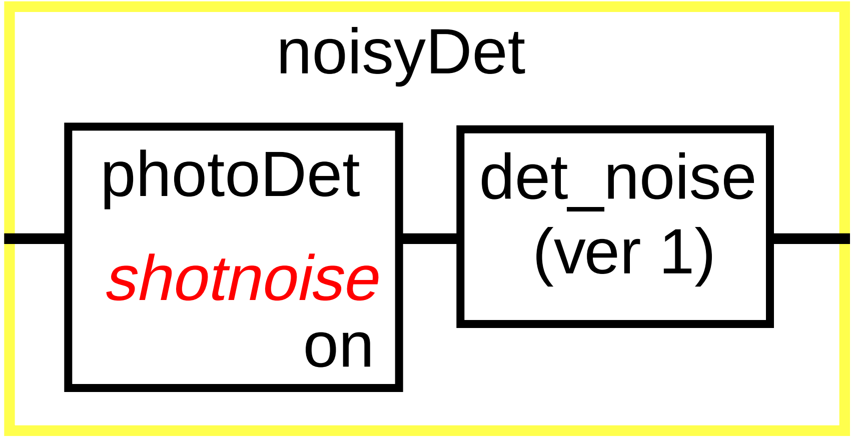
instanciation and inheritance, or which box to edit

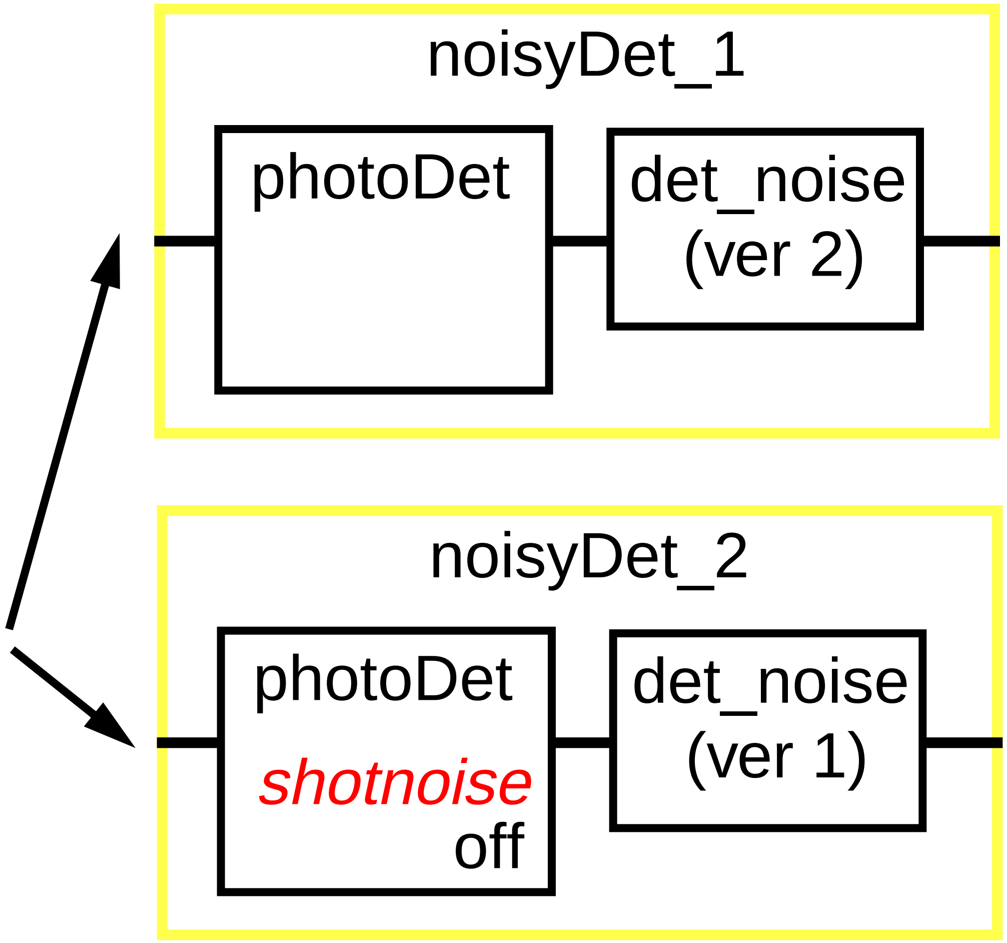
modified

unmodified



internal view



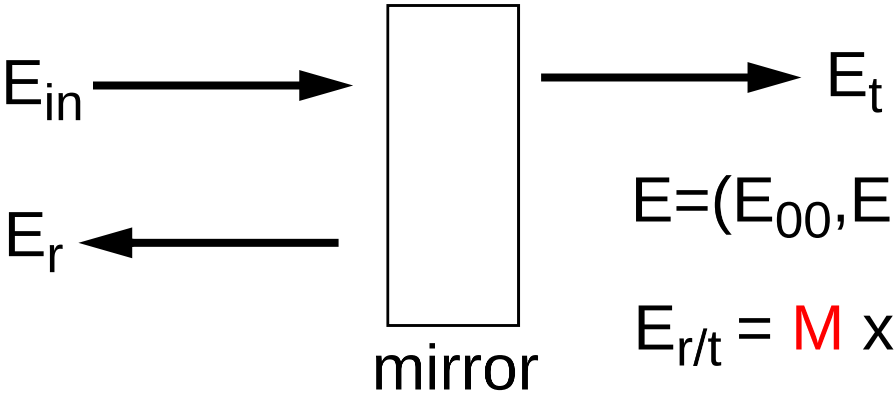


file menu -> box
right button -> original box

double click
right button -> internal view

How to add mirror distortion

mode decomposition matrix



E_{in}

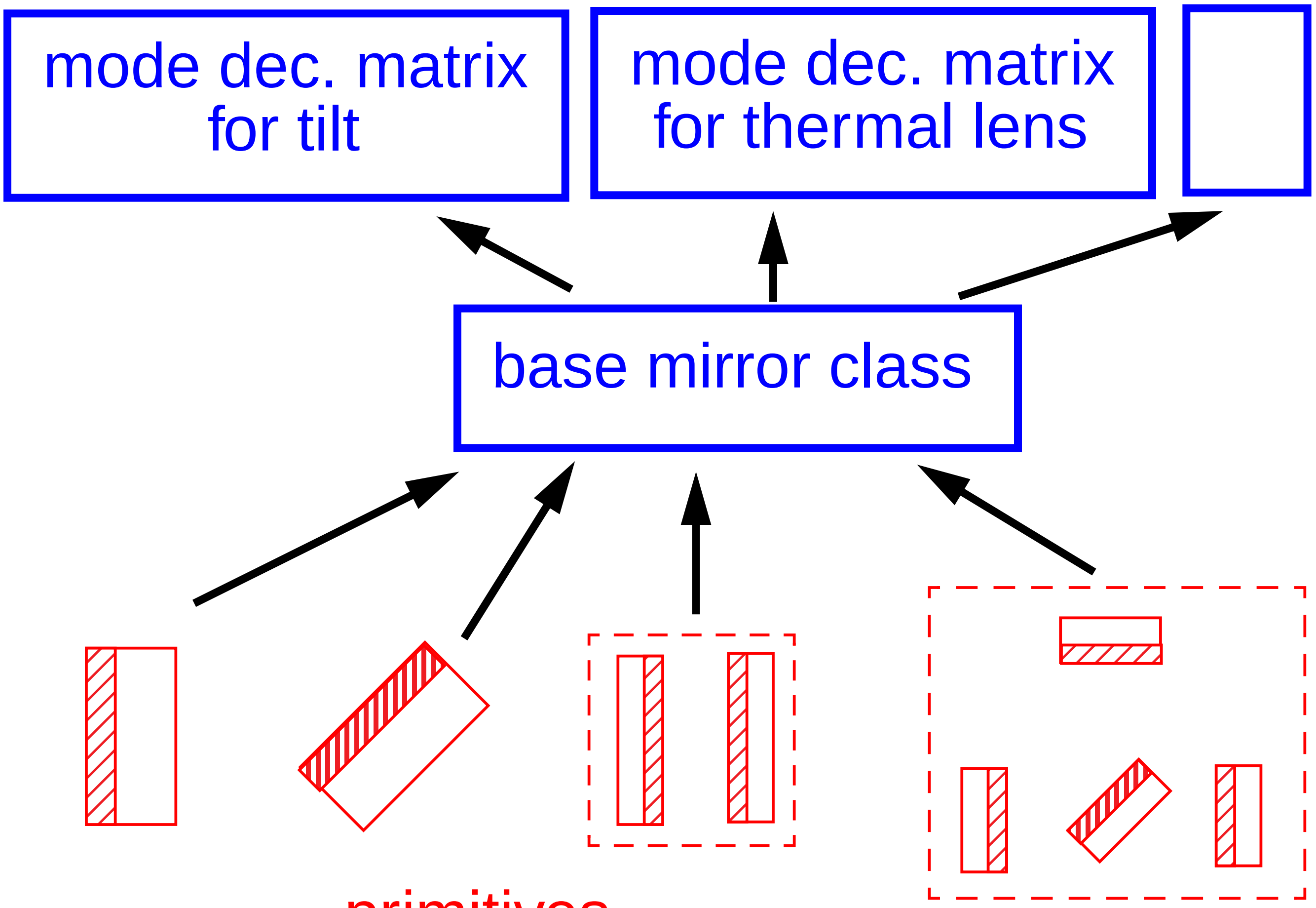
E_t

E_r

mirror

$E=(E_{00},E_{10},\ldots)$

$E_{r/t} = M \times E_{in}$



mode dec. matrix for tilt

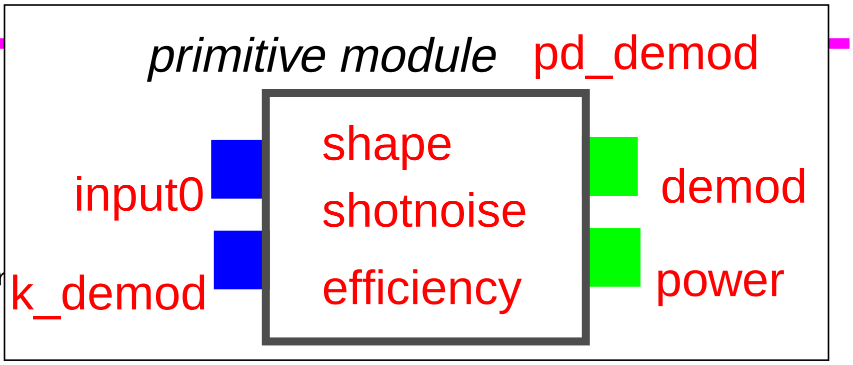
mode dec. matrix for thermal lens

base mirror class

primitives

How to add a new module

standard interface - fortran/c/c++



```
class pd_demod : public primitive
{
  public:
  pd_demod(const string& name_arg = "",
  ~pd_demod();

  module* new_type(const string& name_ar
  void action();
}
================================================================================
circular_detector pd_demod::detector[circular_detector::num_shapes];
// public
pd_demod::pd_demod
(const string& name_arg, const module* parent_arg)
 : primitive(name_arg, parent_arg, "pd_demod", 2, 2),
   in_code(0),
   default_k_demod(0.0),k_demod(NULL),
   set_shape(0),
   set_shotnoise(false),
   set_efficiency(1.0)
{
  setup_input(0, data_ref(&default_input, data_ref::Type_Field),(const void**)(&input));
  setup_input(1, data_ref(&default_k_demod, data_ref::Type_Real),(const void**)(&k_demod));
  set_input_name(1, "k_demod");
```

inputs

```
  setup_output(0, data_ref(&demod_output, data_ref::Type_Complex));
  set_output_name(0, "demod");
  setup_output(1, data_ref(&power_output, data_ref::Type_Real));
  set_output_name(1, "power");
```

outputs

```
  add_auxiliary(data_ref(&set_shape, data_ref::Type_Integer), "shape");
  add_auxiliary(data_ref(&set_shotnoise, data_ref::Type_Boolean), "shotnoise");
  add_auxiliary(data_ref(&set_efficiency, data_ref::Type_Real), "efficiency");
}
```

settings

```
module* pd_demod::new_type
(const string& name_arg, const module* parent_arg) const
{ return new pd_demod(name_arg, parent_arg); }

void pd_demod::action()
{
  [ PHYSICS COMES HERE ]
}
```