# **MOOSE** to PyMOOSE: Interfacing MOOSE with Python

Subhasis Ray

National Centre for Biological Sciences

Tata Institute of Fundamental Research

Bangalore, Karnataka, India

# The People

Upi

Niraj

Raamesh

# Genesis of MOOSE

GEneral NEural SImulation System has been serving for neuronal and biochemical simulations for last 20 years.

It employed the idea of connecting the simulation entities through messages.

The scripting language is dynamic – you can add and delete fields to/from objects at runtime.

Imitated UNIX file-system tree convention to organize and traverse objects ( which are called elements).

All object access is path-based.

# Genesis of MOOSE

*“They did it by making the **single worst strategic mistake** that any software company can make:*

*They decided to rewrite the code from scratch.”*

*- Joel Spolsky, Things You Should Never Do, Part I*

*“All repairs tend to destroy the structure, to increase the entropy and disorder of the system.”*

*-Frederick P. Brooks, "The Mythical Man-Month”*

The need for modernization was pressing, with obvious limitations of the scripting language.

Some aspects of the GENESIS source code are problematic.

# Problems with GENESIS

Messaging: implemented as linked list in genesis – forces sequential traversal.

Scheduling: the scheduler looks through all objects to check if it should be processed.

Encapsulation: Object safety was lacking – receiver of a message would actually peep into the memory allocated to the sender object to get message values.

Extension: To introduce a new object type one has to prepare three associated files – C header, C source and a genesis script. A preprocessor would look into the header and create a mapping between object names and c pointers.

# The creation of MOOSE

Another misuse of Michelangelo's work : with due apology

# GENESIS of MOOSE

- MOOSE ( Multiscale Object Oriented Simulation Environment) started as a backward compatible successor of GENESIS.
- The core system has been written from scratch.
- The underlying architecture changed completely.
- MOOSE inherits GENESIS parser and can run GENESIS scripts.

# A sample session

```
create compartment /squid
setfield /squid Ra  7.6e6 \
      Rm 4.2e4 Cm 7.8e-9 \
      Em -0.07
create table /Vm
call /Vm TABCREATE  \
    {RUNTIME / PLOTDT} 0 1
setfield /Vm step_mode 3
addmsg /squid /Vm INPUT Vm
setclock 0 {SIMDT}
setclock 1 {PLOTDT}
useclock /squid 0
useclock /Vm 1
reset
setfield /squid inject 0
step 0.005 -t
setfield /squid inject
    {INJECT}
step 0.040 -t
setfield /squid inject 0
step 0.005 -t
tab2file squid.plot /Vm
    table
quit
```

# A sample session

```
create compartment /squid
setfield /squid Ra  7.6e6 \
      Rm 4.2e4 Cm 7.8e-9 \
      Em -0.07
create table /Vm
call /Vm TABCREATE  \
    {RUNTIME / PLOTDT} 0 1
setfield /Vm step_mode 3
addmsg /squid /Vm INPUT Vm
setclock 0 {SIMDT}
setclock 1 {PLOTDT}
useclock /squid 0
useclock /Vm 1
reset
setfield /squid inject 0
step 0.005 -t
setfield /squid inject
    {INJECT}
step 0.040 -t
setfield /squid inject 0
step 0.005 -t
tab2file squid.plot /Vm
    table
quit
```

# A sample session

```
create compartment /squid
setfield /squid Ra  7.6e6 \
      Rm 4.2e4 Cm 7.8e-9 \
      Em -0.07
create table /Vm
call /Vm TABCREATE  \
    {RUNTIME / PLOTDT} 0 1
setfield /Vm step_mode 3
addmsg /squid /Vm INPUT Vm
setclock 0 {SIMDT}
setclock 1 {PLOTDT}
useclock /squid 0
useclock /Vm 1
reset
setfield /squid inject 0
step 0.005 -t
setfield /squid inject
    {INJECT}
step 0.040 -t
setfield /squid inject 0
step 0.005 -t
tab2file squid.plot /Vm
    table
quit
```

# A sample session

```
create compartment /squid
setfield /squid Ra  7.6e6 \
      Rm 4.2e4 Cm 7.8e-9 \
      Em -0.07
create table /Vm
call /Vm TABCREATE  \
    {RUNTIME / PLOTDT} 0 1
setfield /Vm step_mode 3
addmsg /squid /Vm INPUT Vm
setclock 0 {SIMDT}
setclock 1 {PLOTDT}
useclock /squid 0
useclock /Vm 1
reset
setfield /squid inject 0
step 0.005 -t
setfield /squid inject
    {INJECT}
step 0.040 -t
setfield /squid inject 0
step 0.005 -t
tab2file squid.plot /Vm
    table
quit
```

# A sample session

```
create compartment /squid
setfield /squid Ra  7.6e6 \
      Rm 4.2e4 Cm 7.8e-9 \
      Em -0.07
create table /Vm
call /Vm TABCREATE  \
    {RUNTIME / PLOTDT} 0 1
setfield /Vm step_mode 3
addmsg /squid /Vm INPUT Vm
setclock 0 {SIMDT}
setclock 1 {PLOTDT}
```

```
useclock /squid 0
useclock /Vm 1
reset
setfield /squid inject 0
step 0.005 -t
setfield /squid inject
    {INJECT}
step 0.040 -t
setfield /squid inject 0
step 0.005 -t
tab2file squid.plot /Vm
    table
quit
```

# A sample session

```
create compartment /squid
setfield /squid Ra  7.6e6 \
      Rm 4.2e4 Cm 7.8e-9 \
      Em -0.07
create table /Vm
call /Vm TABCREATE  \
    {RUNTIME / PLOTDT} 0 1
setfield /Vm step_mode 3
addmsg /squid /Vm INPUT Vm
setclock 0 {SIMDT}
setclock 1 {PLOTDT}
```

```
useclock /squid 0
useclock /Vm 1
reset
setfield /squid inject 0
step 0.005 -t
setfield /squid inject
    {INJECT}
step 0.040 -t
setfield /squid inject 0
step 0.005 -t
tab2file squid.plot /Vm
    table
quit
```

# A sample session

```
create compartment /squid
setfield /squid Ra  7.6e6 \
      Rm 4.2e4 Cm 7.8e-9 \
      Em -0.07
create table /Vm
call /Vm TABCREATE  \
    {RUNTIME / PLOTDT} 0 1
setfield /Vm step_mode 3
addmsg /squid /Vm INPUT Vm
setclock 0 {SIMDT}
setclock 1 {PLOTDT}
```

```
useclock /squid 0
useclock /Vm 1
reset
setfield /squid inject 0
step 0.005 -t
setfield /squid inject
    {INJECT}
step 0.040 -t
setfield /squid inject 0
step 0.005 -t
tab2file squid.plot /Vm
    table
quit
```

# A sample session

```
create compartment /squid
setfield /squid Ra  7.6e6 \
      Rm 4.2e4 Cm 7.8e-9 \
      Em -0.07
create table /Vm
call /Vm TABCREATE  \
    {RUNTIME / PLOTDT} 0 1
setfield /Vm step_mode 3
addmsg /squid /Vm INPUT Vm
setclock 0 {SIMDT}
setclock 1 {PLOTDT}
useclock /squid 0
useclock /Vm 1
reset
setfield /squid inject 0
step 0.005 -t
setfield /squid inject
    {INJECT}
step 0.040 -t
setfield /squid inject 0
step 0.005 -t
tab2file squid.plot /Vm
    table
quit
```

# A sample session

```
create compartment /squid
setfield /squid Ra  7.6e6 \
      Rm 4.2e4 Cm 7.8e-9 \
      Em -0.07
create table /Vm
call /Vm TABCREATE  \
    {RUNTIME / PLOTDT} 0 1
setfield /Vm step_mode 3
addmsg /squid /Vm INPUT Vm
setclock 0 {SIMDT}
setclock 1 {PLOTDT}
useclock /squid 0
useclock /Vm 1
reset
setfield /squid inject 0
step 0.005 -t
setfield /squid inject
    {INJECT}
step 0.040 -t
setfield /squid inject 0
step 0.005 -t
tab2file squid.plot /Vm
    table
quit
```

# Architecture of MOOSE

- Three types of objects -

## System objects

constitute the backbone of the engine, e.g. Shell, ClockJob, ClockTick etc.

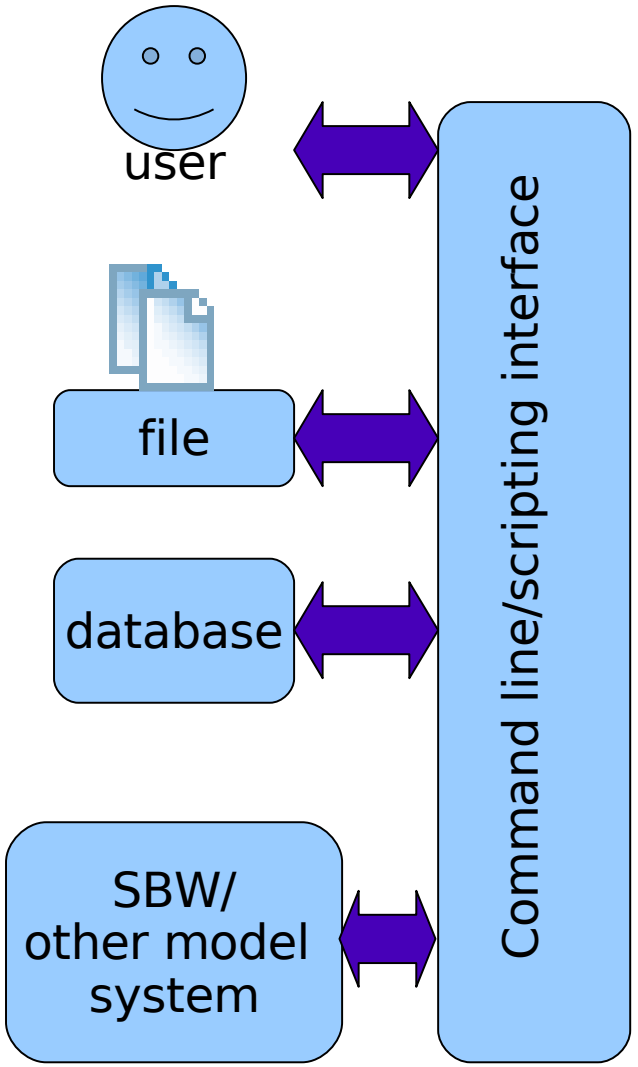
## Simulation entities

representative of biophysical / biochemical objects that represent the states / parameters we are studying

## Utility objects

provide various other functionalities like inspecting data, interpolation, etc.

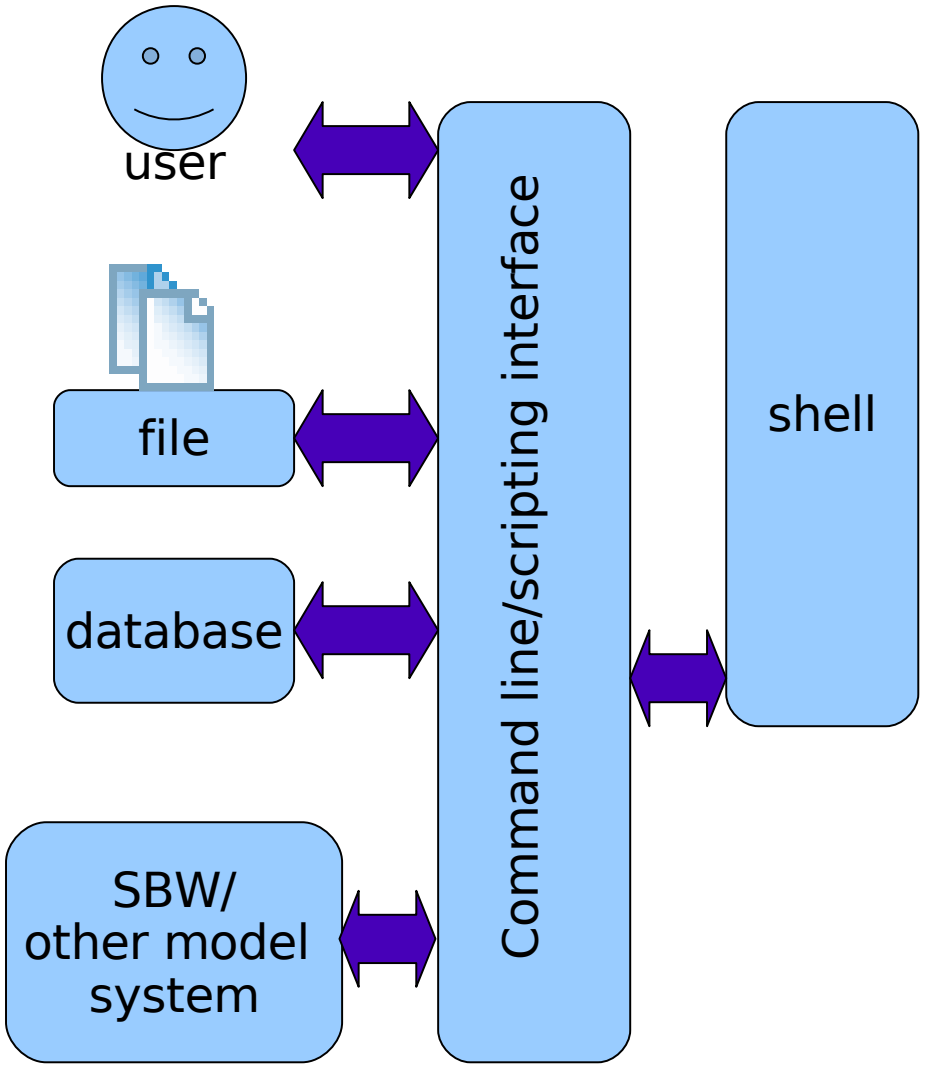
# MOOSE Architecture




The command line / scripting interface is where modeler interacts with MOOSE.

Derived from:
http://moose.ncbs.res.in/images/stories/architecture_65.jpg

# MOOSE Architecture



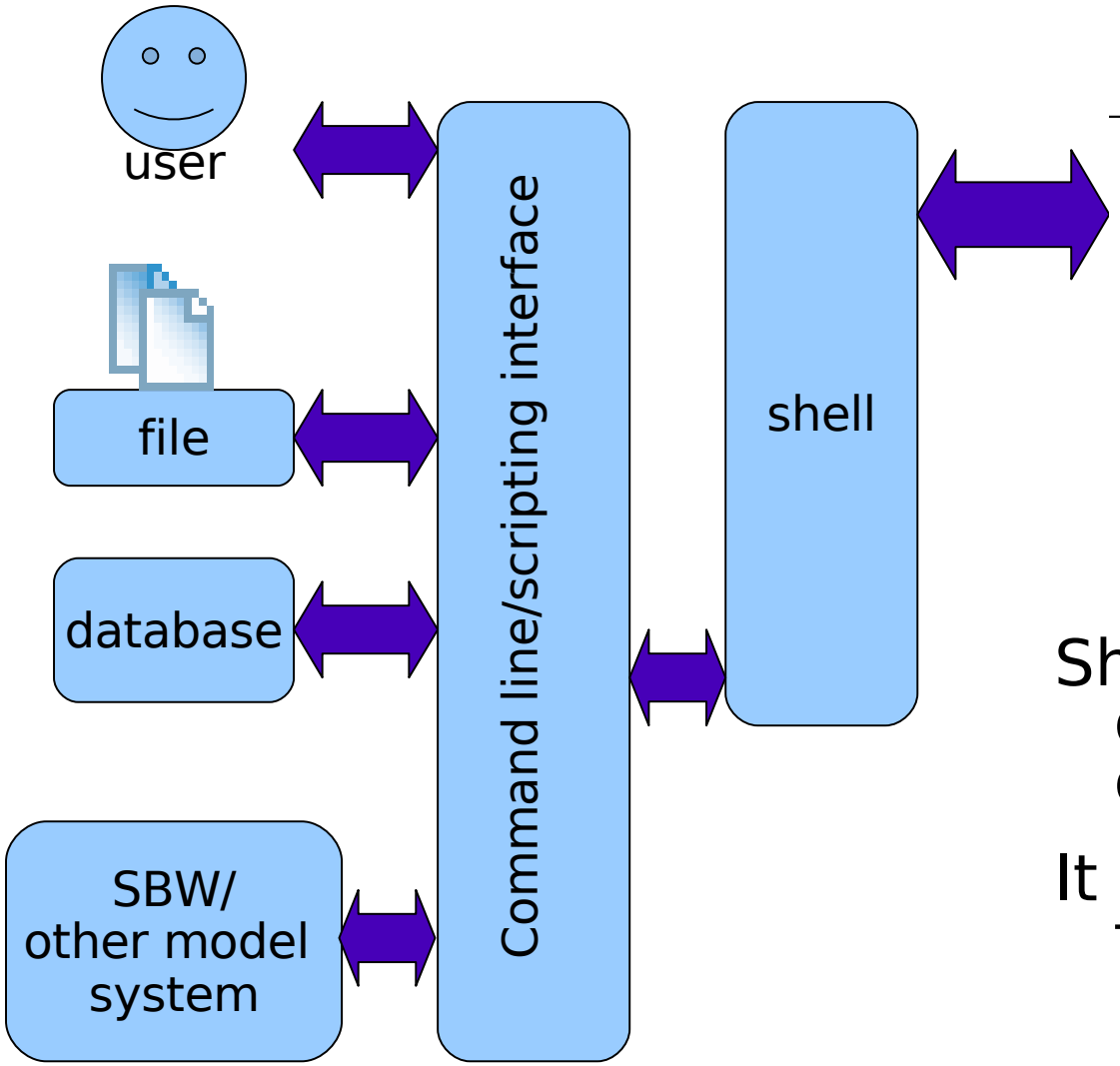

The user interface talks to

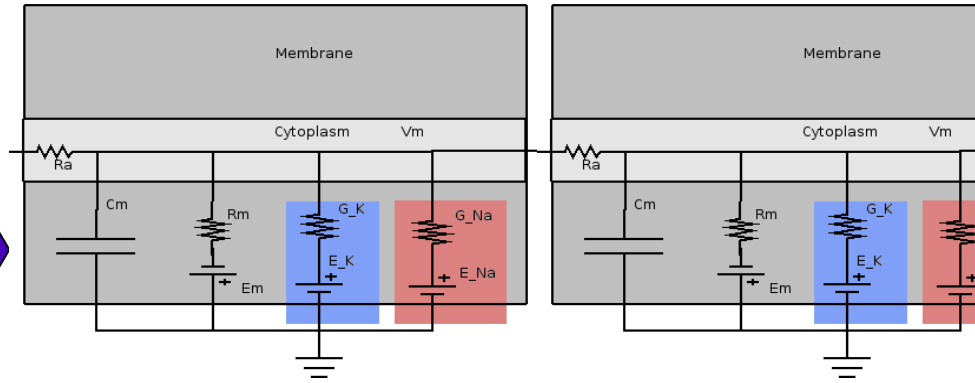
*Shell* - the single point of access to all functionalities available to the user

*Not to be confused with the idea of UNIX shell*

Derived from:
http://moose.ncbs.res.in/images/stories/architecture_65.jpg

# MOOSE Architecture



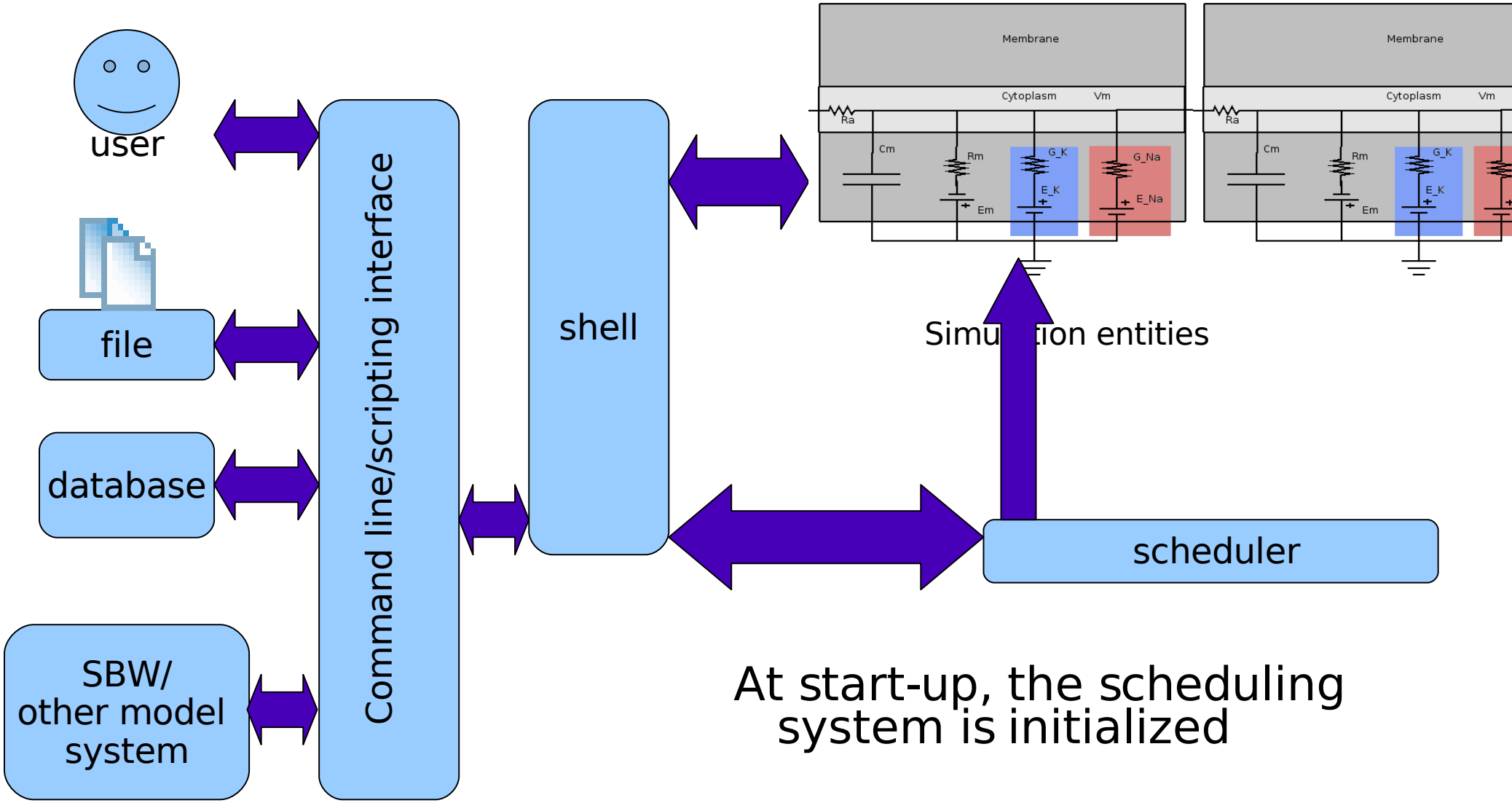

Simulation entities

Shell creates, deletes and queries other MOOSE objects.

It also executes the built-in functionalities like file I/O.

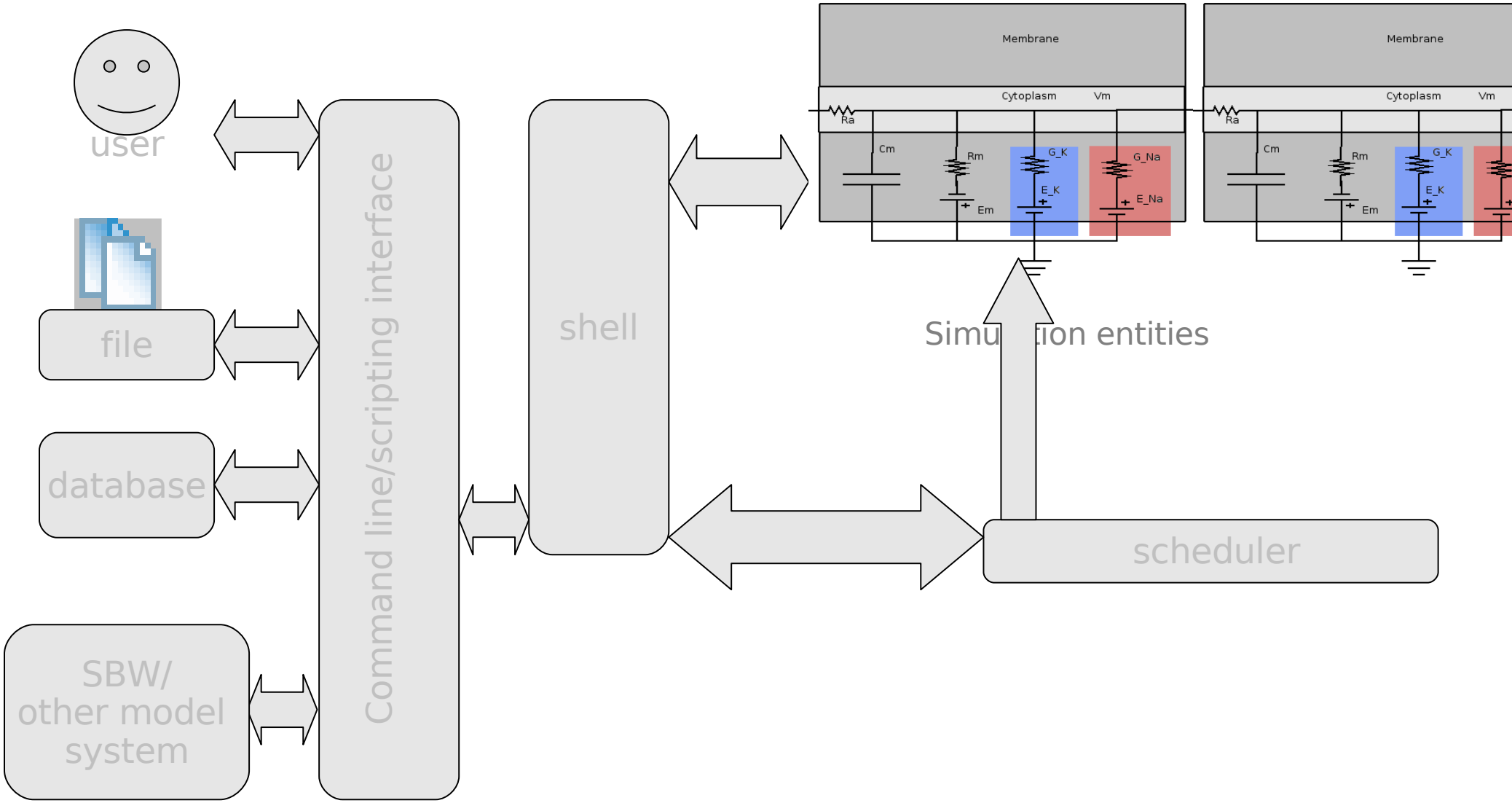
Derived from:
http://moose.ncbs.res.in/images/stories/architecture_65.jpg

# MOOSE Architecture



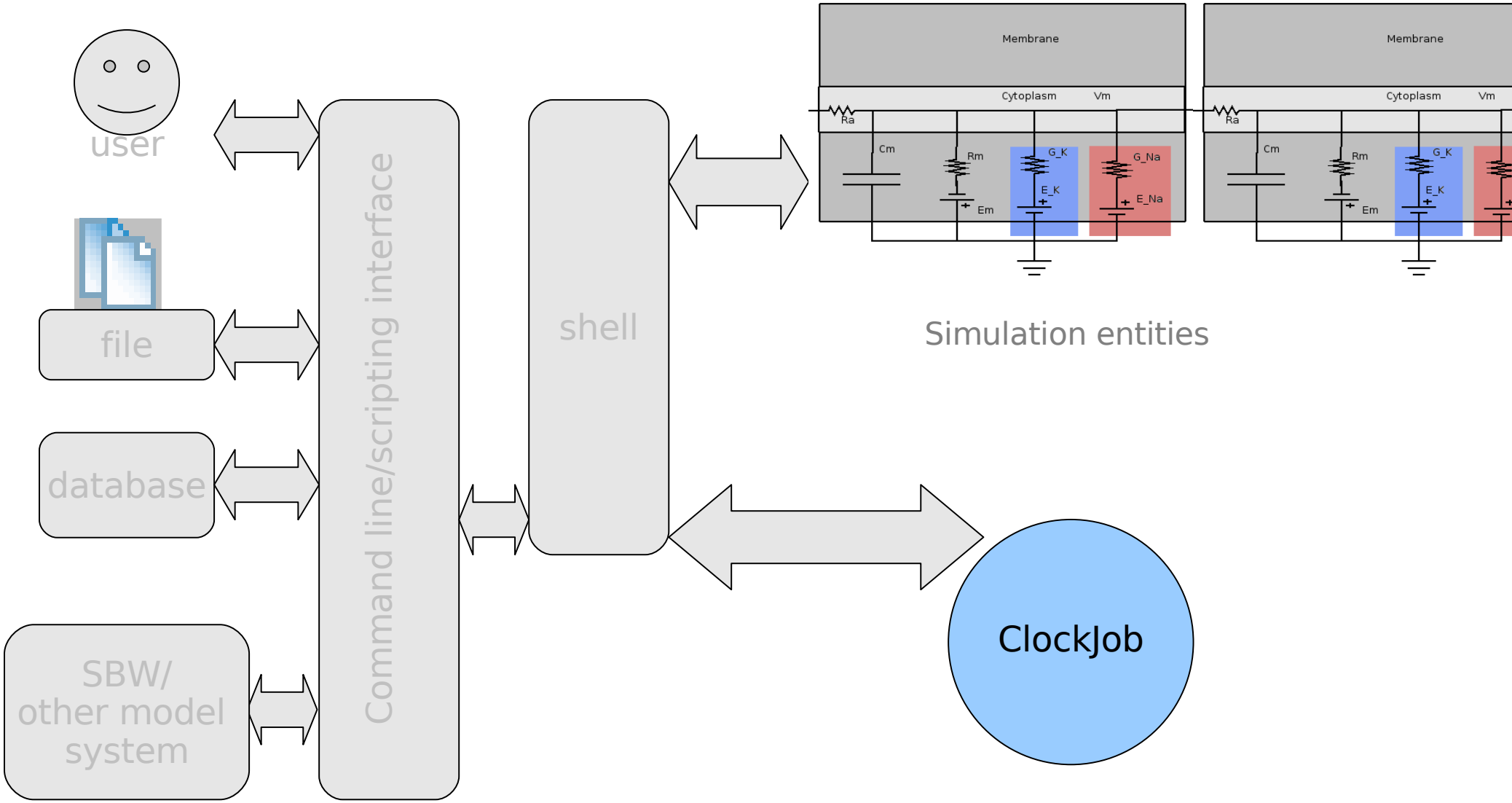

Derived from:
http://moose.ncbs.res.in/images/stories/architecture_65.jpg

# MOOSE Architecture



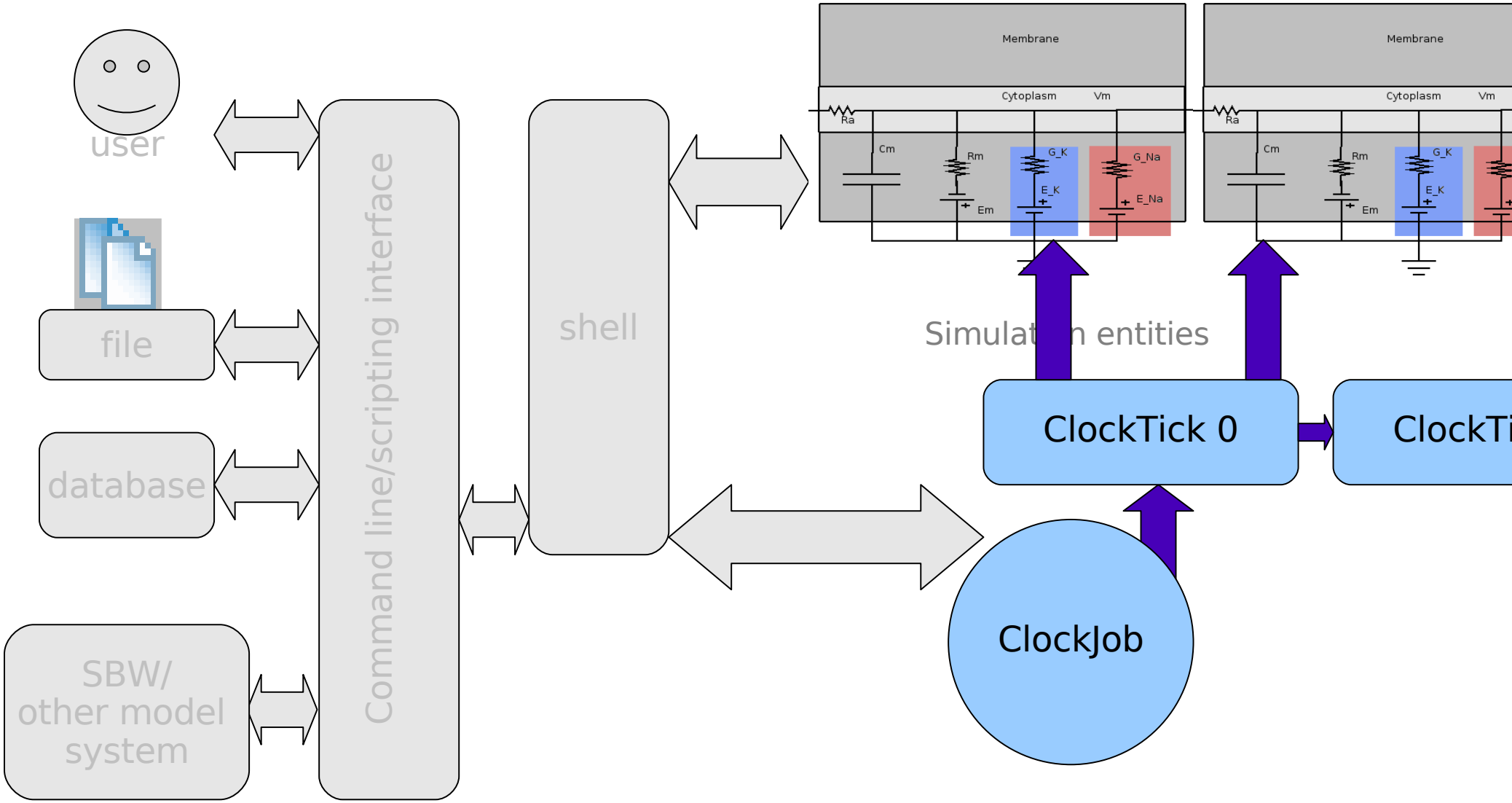

Derived from:
http://moose.ncbs.res.in/images/stories/architecture_65.jpg

# MOOSE Architecture



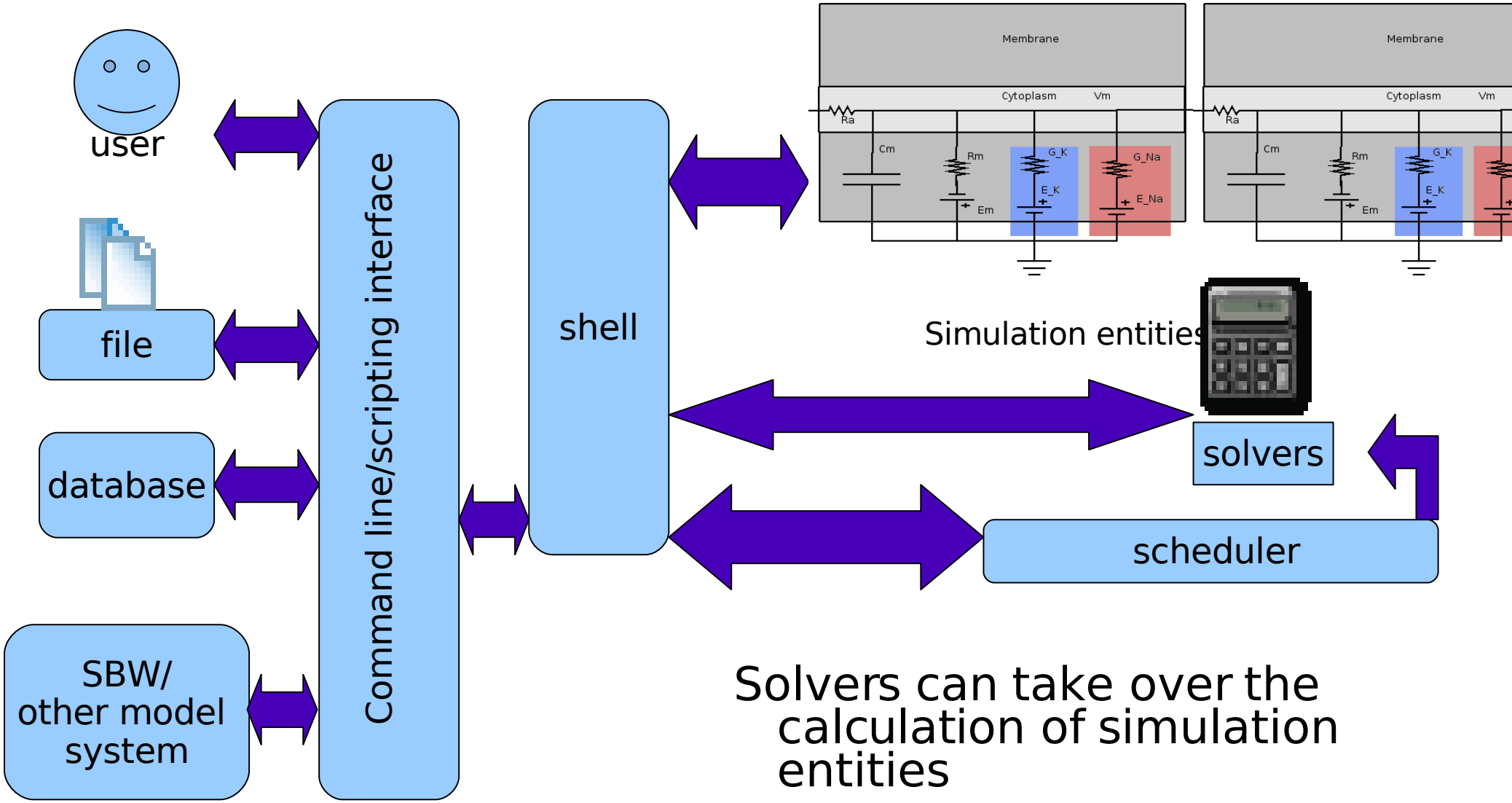

Derived from:
http://moose.ncbs.res.in/images/stories/architecture_65.jpg

# MOOSE Architecture




Derived from:
http://moose.ncbs.res.in/images/stories/architecture_65.jpg

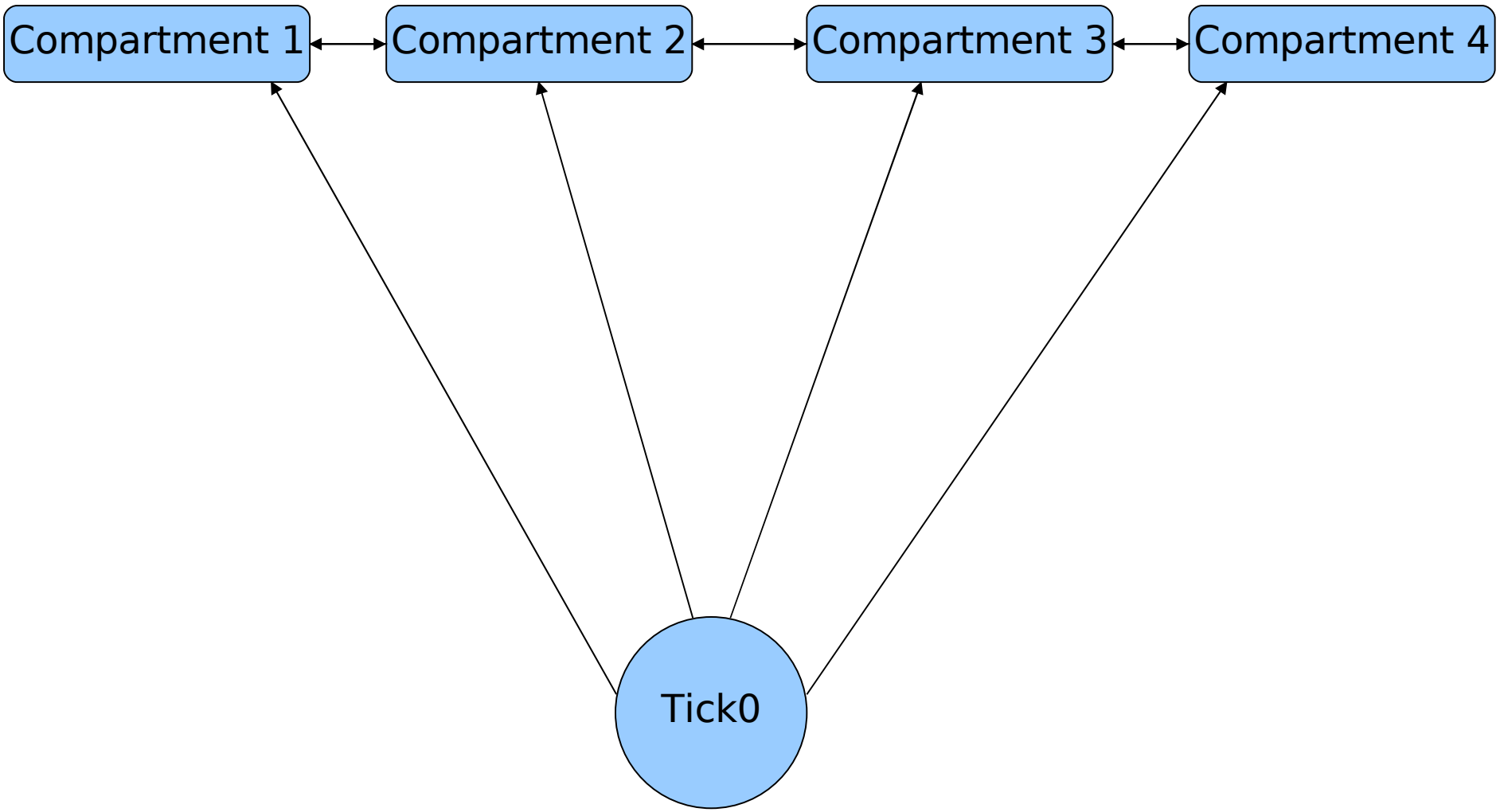
# MOOSE Architecture



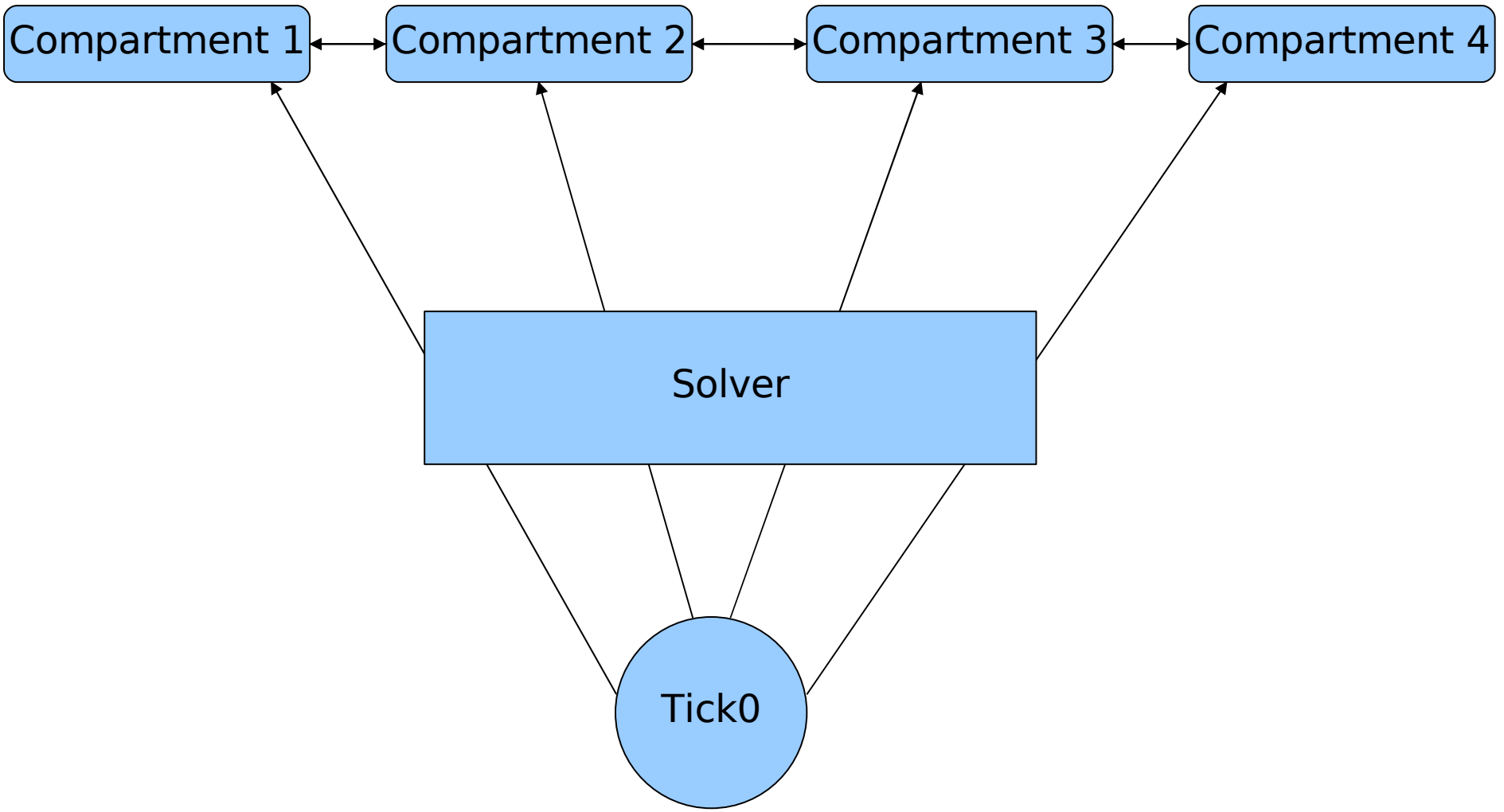

Solvers can take over the calculation of simulation entities

Derived from:
http://moose.ncbs.res.in/images/stories/architecture_65.jpg

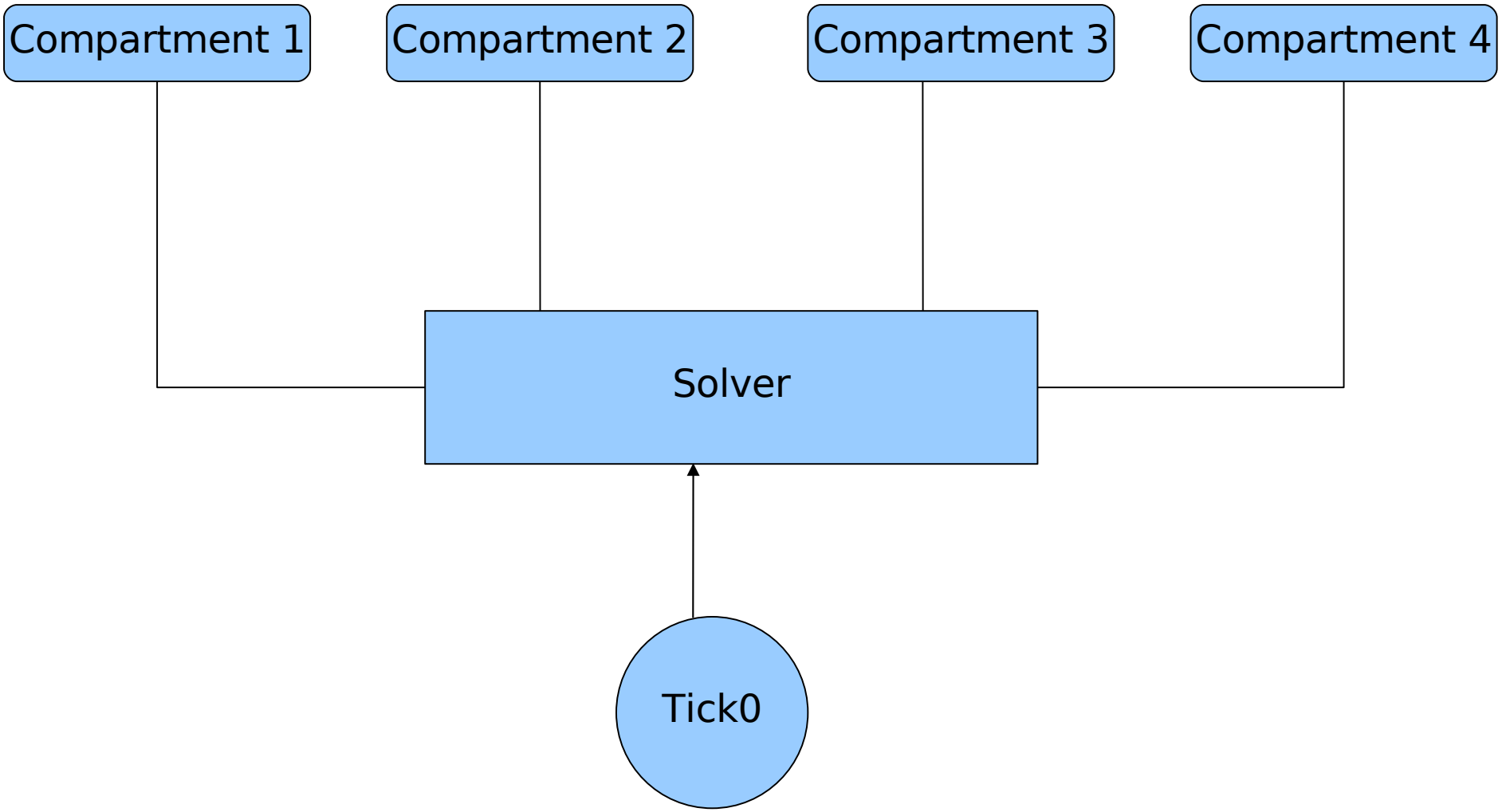
# Solvers: Avoid the penalty of object orientation



# Solvers: Avoid the penalty of object orientation



# Solvers: Avoid the penalty of object orientation



# MOOSE is young – but powerful

MOOSE runs many chemical kinetics simulations 40 to 50 times faster than GENESIS

Hines solver for branched neurons is under development -

current version ( *without optimization* ) has reached at least 50% the speed of the optimized solver of GENESIS.

Performance of MOOSE is competitive to other general purpose simulators.

It is easy to add new classes to MOOSE. The API is developer friendly.

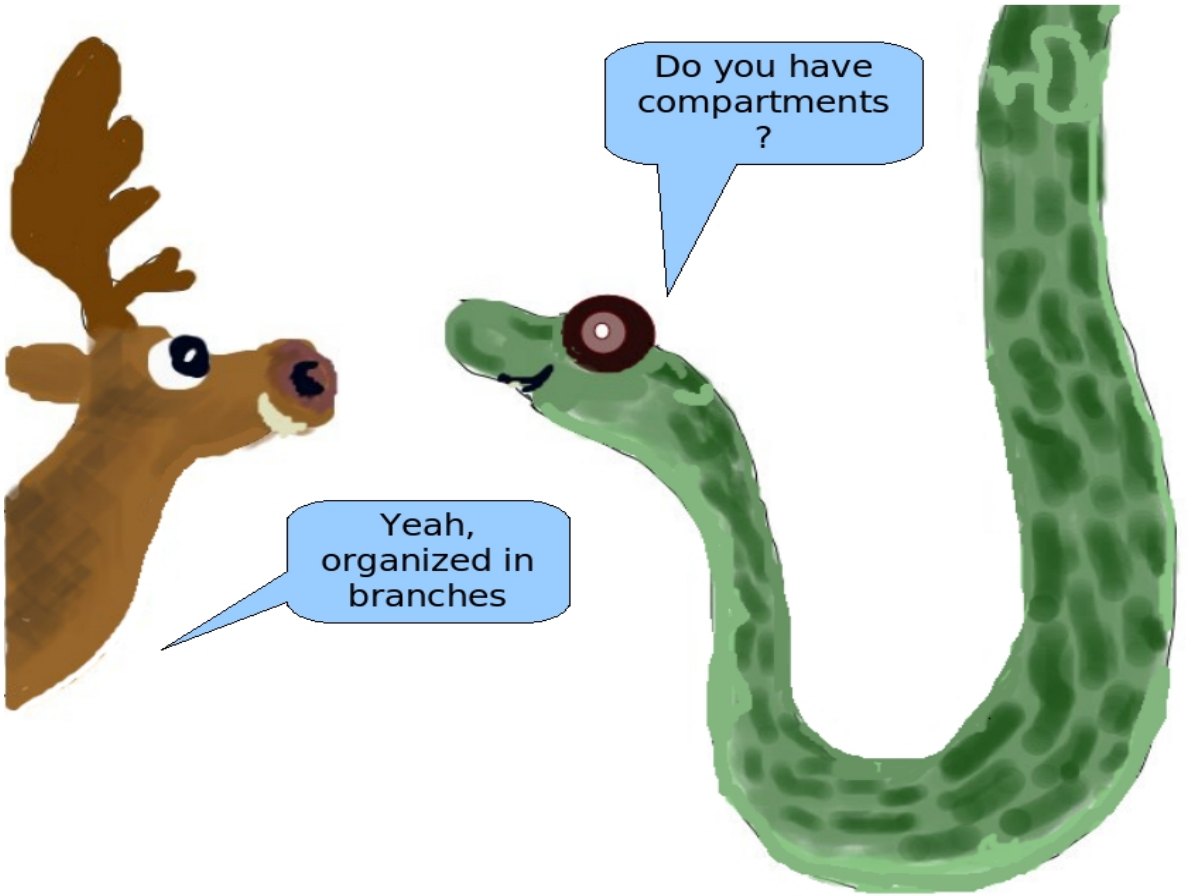
# MOOSE meets Python

The limitations of the GENESIS scripting language start hurting as we go into larger models :

need for a general purpose language was obvious.

Python seemed to be the best choice.

# MOOSE meets Python




Write code to embed MOOSE into python or

make MOOSE a loadable library for python.

# SWIG bridges the gap

- Several options for interface generator:
  - Boost.Python – lots of features, *but uses its own build system "bjam"*
  - SIP – Trolltech's tool to make PyQt – *but it is Qt specific*
  - SWIG – Mature, active, easy to learn, can handle many languages other than Python

# How to do it with SWIG

- Additional C++ classes to wrap MOOSE classes

  Create another layer of C++ code that talks to MOOSE objects.

  These are the classes actually seen by SWIG and made available to Python.

  This allows the MOOSE developer to be Python agnostic.

# How to do it with SWIG – the path not taken

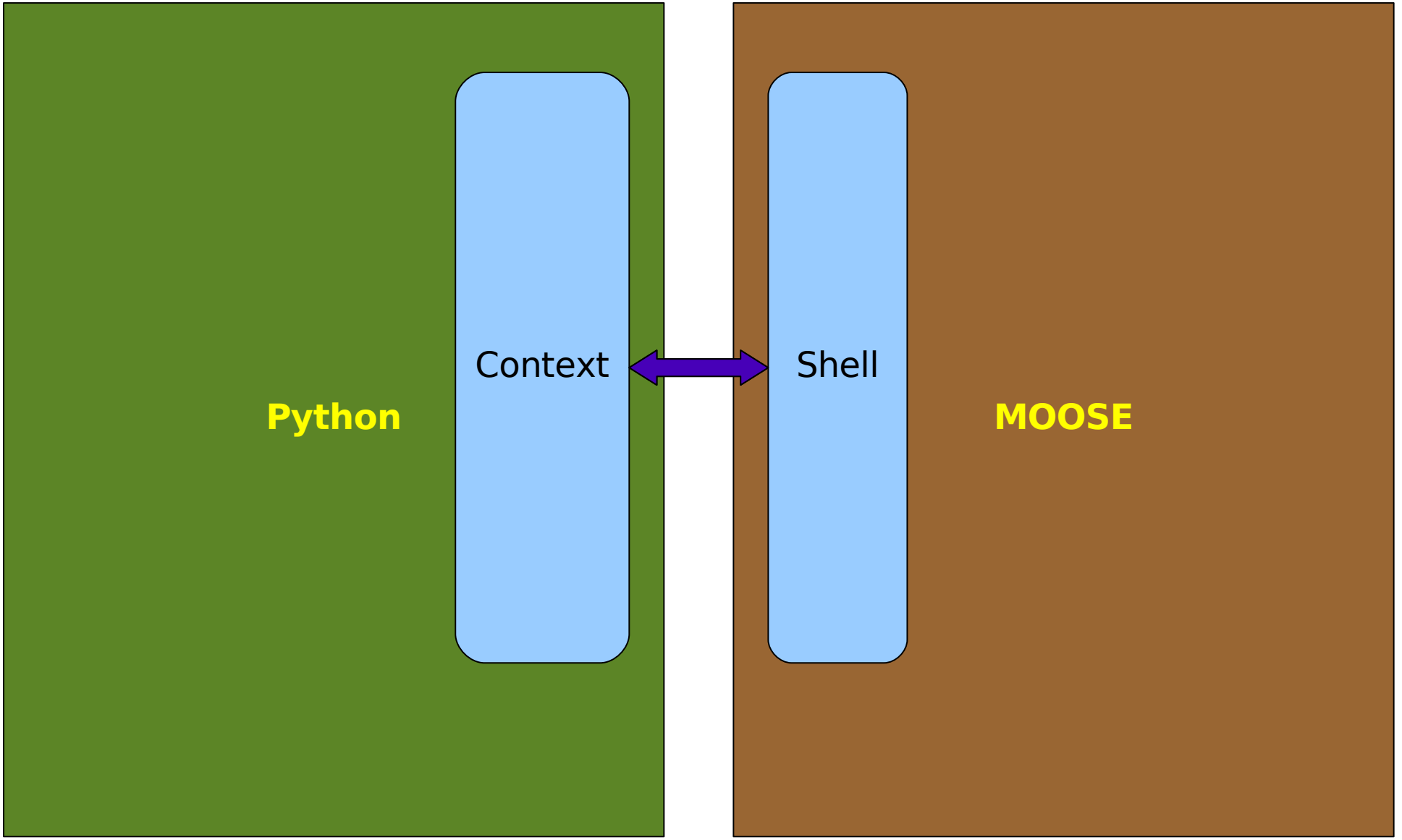
- Directly passing the source code to SWIG:

  Not easy - not clean - the fields are not real C++ class members, but more complex data structures.

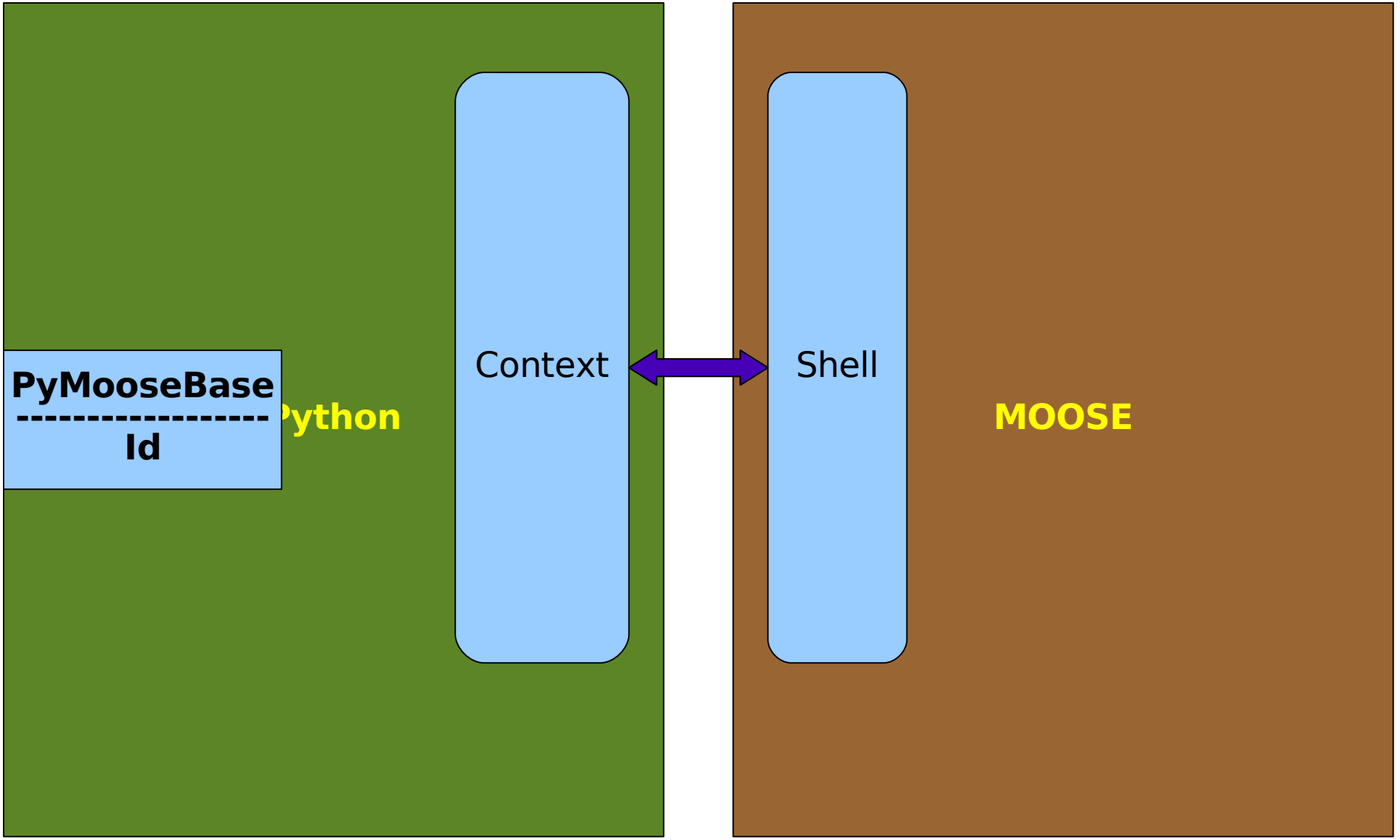
  Classes would not have the python-like feel

  The MOOSE developer has to beware of what SWIG likes and dislikes.

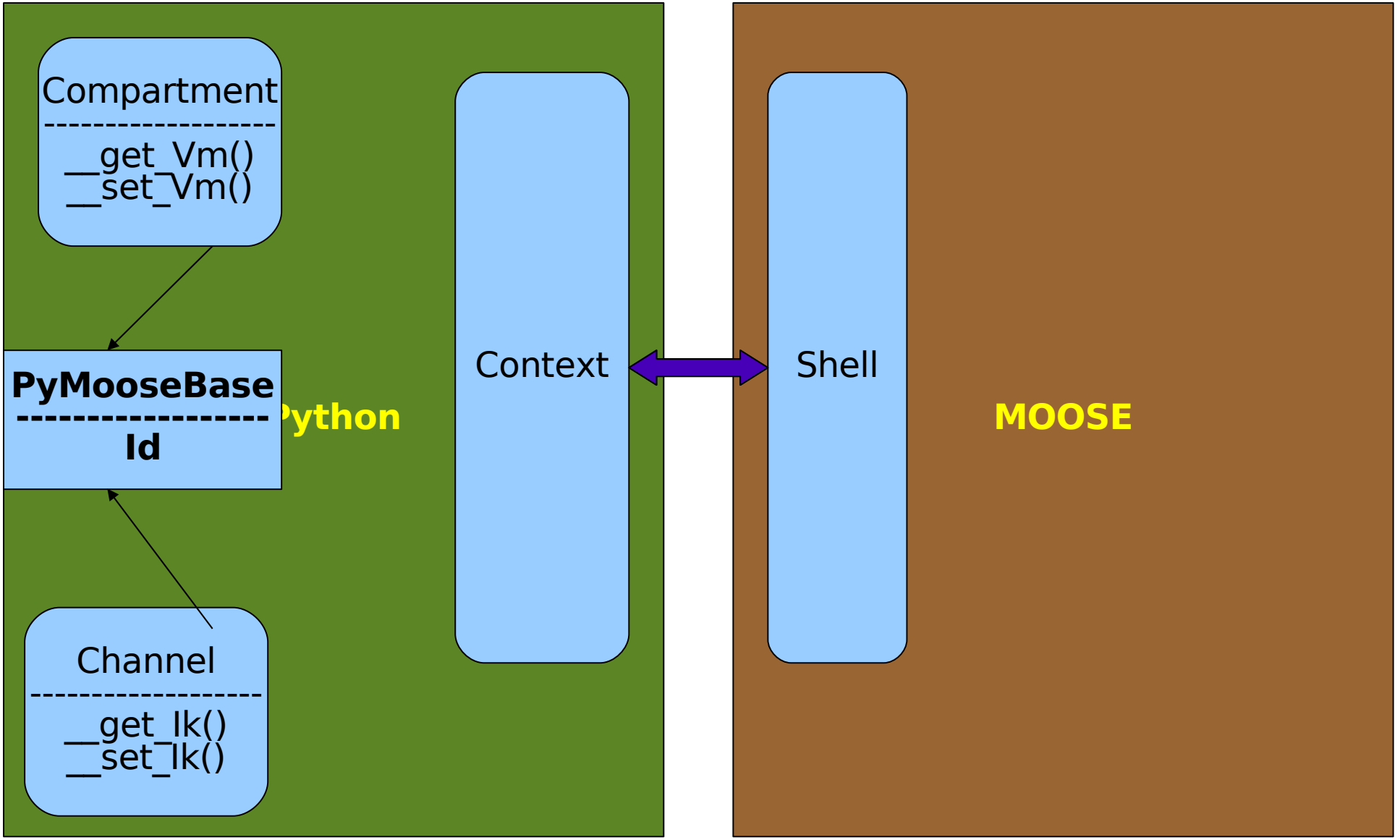
# Architecture of PyMOOSE



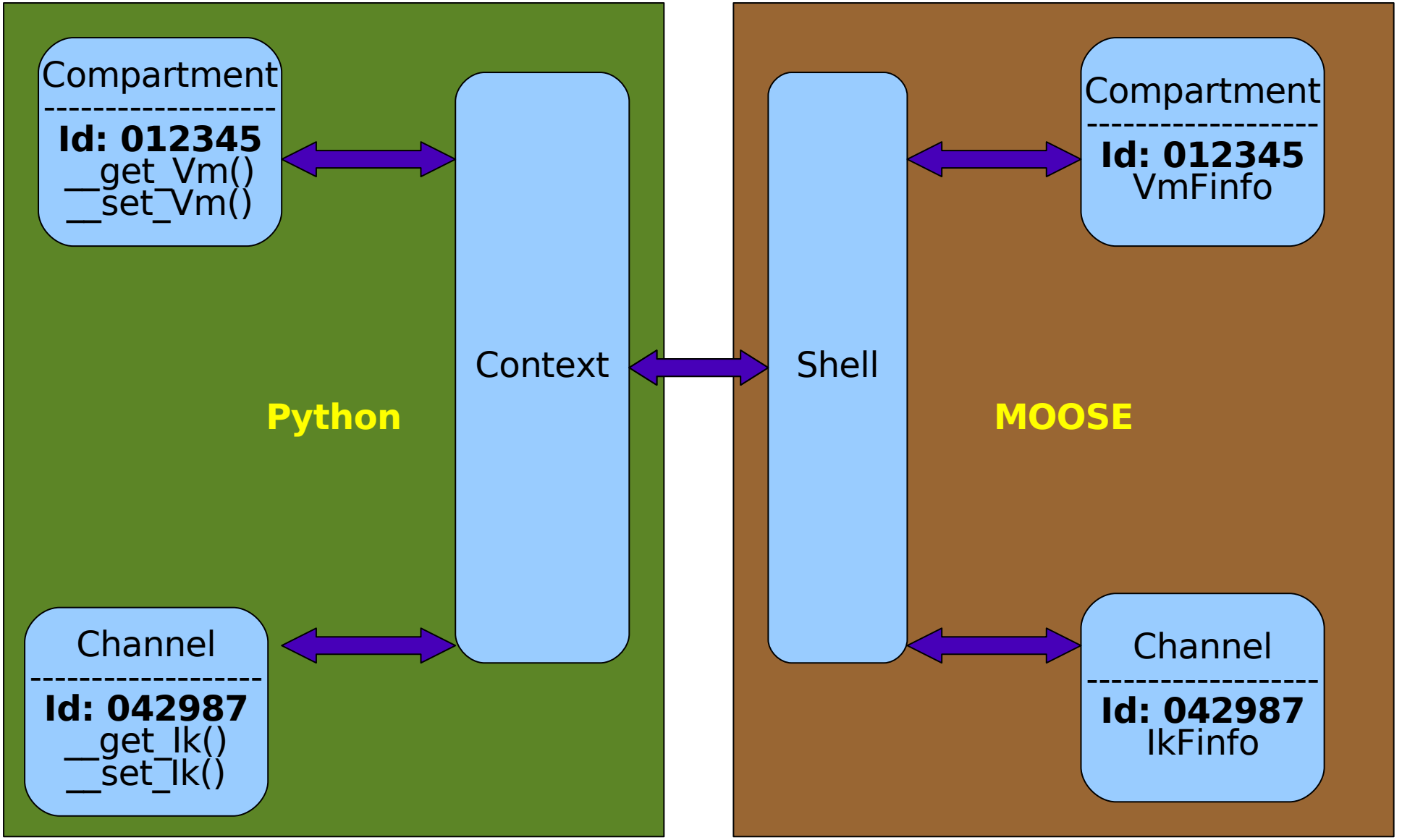
# Architecture of PyMOOSE



# Architecture of PyMOOSE



# Architecture of PyMOOSE



# Let the computer do it!

Given the infrastructure created with Context and Base class, writing wrappers is easy.

MOOSE statically initializes data structures for the fields in the MOOSE classes. Exploited this to automate the creation of wrapper classes.

Pieces of code in Class-info initializer of MOOSE create C++ header and source files as well as swig interface file for PyMOOSE.

Minor changes need to be introduced manually in the generated code.

# It's easy to convert to PyMOOSE

```
float EREST = -0.07
# ... ... ...
function calc_Na_m_A( v )
        float v
        if ( { abs { EREST + 0.025 - v } } < 1e-6 )
                v = v + 1e-6
        end
        return { ( 0.1e6 * ( EREST + 0.025 - v ) ) /
            ( { exp { ( EREST + 0.025 - v )/ 0.01 } }
              - 1.0 ) }
end
# ... ... ...
create Compartment /squid
setfield /squid Rm {RM}
# ... ... ...
for ( i = 0 ; i <= NDIVS; i = i + 1 )
        setfield /squid/Na/xGate/A table[{i}]
        { calc_Na_m_A { v } }
# ... ... ...
end
# ... ... ...
setclock 0 {SIMDT} 0
setclock 1 {PLOTDT} 0
# ... ... ...
setfield /squid inject {INJECT}
step 0.040 -t
setfield /Vm print "squid.plot"
```

```
from moose import *
from math import *

EREST = -0.07

def calc_Na_m_A( v ):
    if fabs(EREST+0.025-v) < 1e-6:
        v = v + 1e-6
    return 0.1e6 * (EREST + 0.025 -v) /
            (exp((EREST + 0.025 - v)/0.01) - 1.0)
# ... ... ...
squid = Compartment('squid')
squid.Rm = RM

for i in range(NDIVS+1):
        Na.xGate.A[i] = calc_Na_m_A ( v )
# ... ... ...

context = squid.getContext()
context.setClock(0, SIMDT, 0)
context.setClock(1, PLOTDT, 0)
# ... ... ...
squid.inject = INJECT
context.step(0.040)
from pylab import *
plot(Vm)
show()
```

# Issues PyMOOSE faces

- Object life cycle management poses a major decision problem – right now we are escaping the python convention of object deletion on going outside scope

SWIG-generated-code uses lots of C++ pointers – the related memory management issues do not pose any problem yet – but it may need to be fine tuned.

# Epilogue – what we have

MOOSE is capable of executing many GENESIS scripts with little or no modification.

It takes moderate effort to plug other simulation programs into MOOSE: already the SMOLDYN ( Steve Andrews: *http://www.smoldyn.org* ) program has been incorporated into MOOSE with help from Steve.

MOOSE has been designed with parallelization in mind:

the messaging architecture and unique object Id scheme have been created for clean parallelization.

We also look forward to computer-grids: the uniprocessor version has been run on EU-India Grid and Garuda grid.

# Epilogue – what next?

- Plans for release 2
  - Currently we are working on array-type objects which will reduce the overhead of an array of simple objects.
  - Instead of having element to element messaging, array-messages will connect whole array-objects with the connection rule.
  - Solver is undergoing optimization and will be part of this release.
- Plans for release 3
  - Parallel version of MOOSE is undergoing experimentation

# Epilogue – further goals with PyMOOSE

MOOSE retains the GENESIS scripting interface for backward compatibility. But it has more features which are not handled by GENESIS scripting language.

Instead of extending the GENESIS scripting language, we plan to *switch to Python*.

Use PyMoose to talk to other simulation softwares. We are interested in standards for simulator interactions.

Compatibility with SBML and NeuroML. May use third party libraries to read and write these dialects. But need to bridge the conceptual gaps.

# MOOSE is free software

- MOOSE is available at:
  - http://moose.ncbs.res.in
  - http://moose.sourceforge.net/

- Get latest development snapshot:

```
svn co
  http://moose.svn.sourceforge.net/svnroot/moose/moose/trunk
  moose
```

- It is published under LGPL (Lesser GNU Public License).

# Thanks to ...

- Guidance, comments and support :

  Upi

  Lab-mates

- Funding:
  - DAE
  - DBT
  - NCBS/TIFR
  - EU-India Grid project
  - FACETS ( travel bursary )
  - NIH ( Collaboration with Ravi Iyengar )
- CDAC – GARUDA Grid