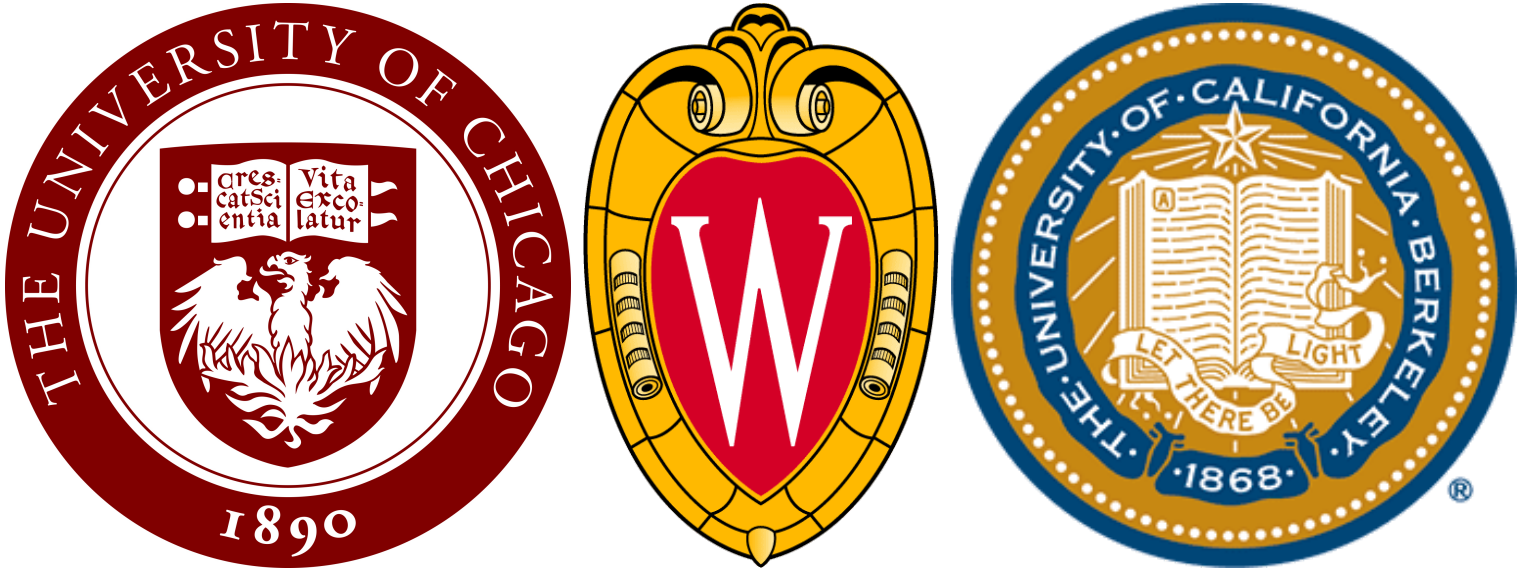
Reproducible Approaches
to Modeling and Simulation in Nuclear Energy

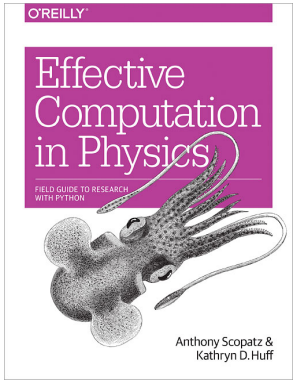
Kathryn Huff
at University of Tennessee Knoxville
March 02, 2016

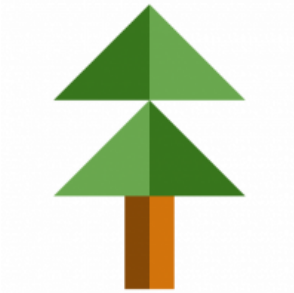

THE UNIVERSITY OF CHICAGO
cres
catSci
entia
Vita
exco
latur
1890
W
THE·UNIVERSITY·OF·CALIFORNIA·BERKELEY·
LET THERE BE LIGHT
·1868·

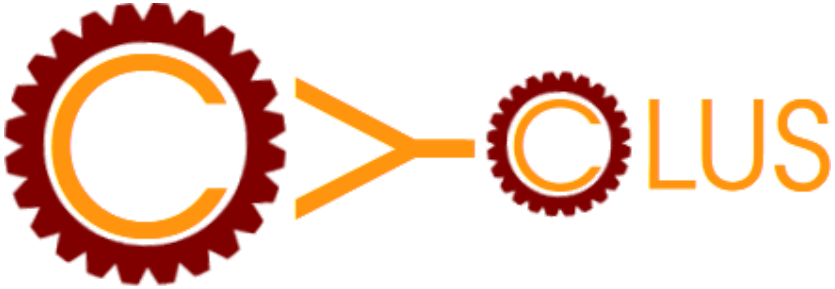

BERKELEY
Institute for
Data Science

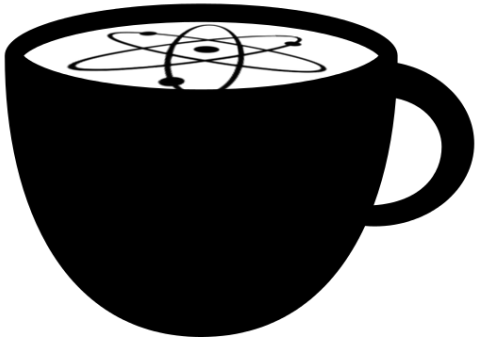

software
carpentry


THW


O'REILLY
Effective
Computation
in Physics
FIELD GUIDE TO RESEARCH
WITH PYTHON
Anthony Scopatz &
Kathryn D. Huff




CYCLUS



Science

- builds and organizes knowledge
- tests explanations about the universe
- systematically,
- objectively,
- transparently,
- and reproducibly.

Otherwise it's not science.

Computers

should...

- improve efficiency,
- reduce human error,
- automate the mundane,
- simplify the complex,
- and accelerate research.

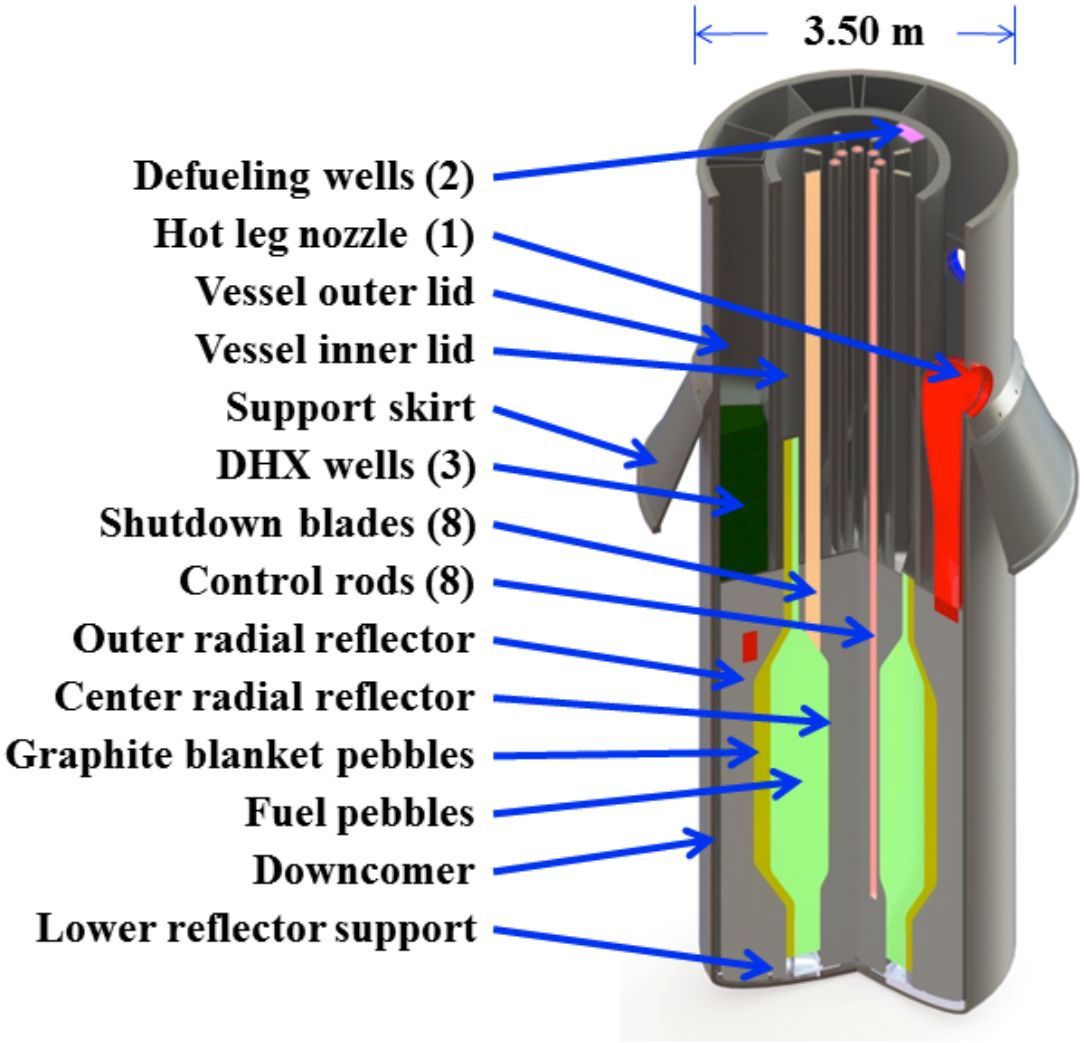
But scientists don't use them effectively.

“ Computational science is a special case of scientific research: the work is easily shared via the Internet since the paper, code, and data are digital and those three aspects are all that is required to reproduce the results, given sufficient computation tools. ” - Stodden, 2010.

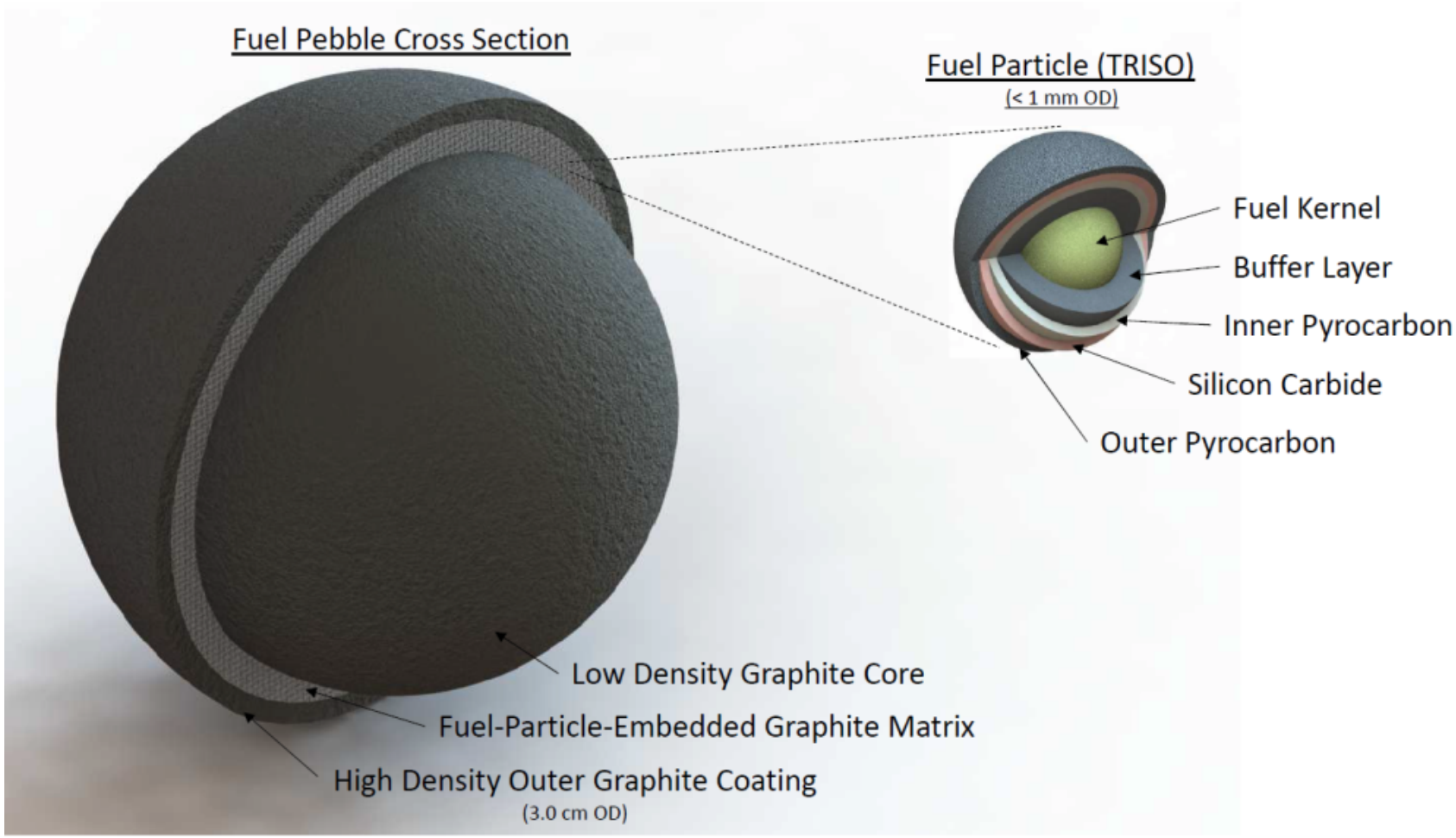

FHR
Fluoride Salt Cooled High
Temperature Reactor


THE UNIVERSITY OF CALIFORNIA
LET THERE BE LIGHT
1868

Mk1 Reactor Cross Section

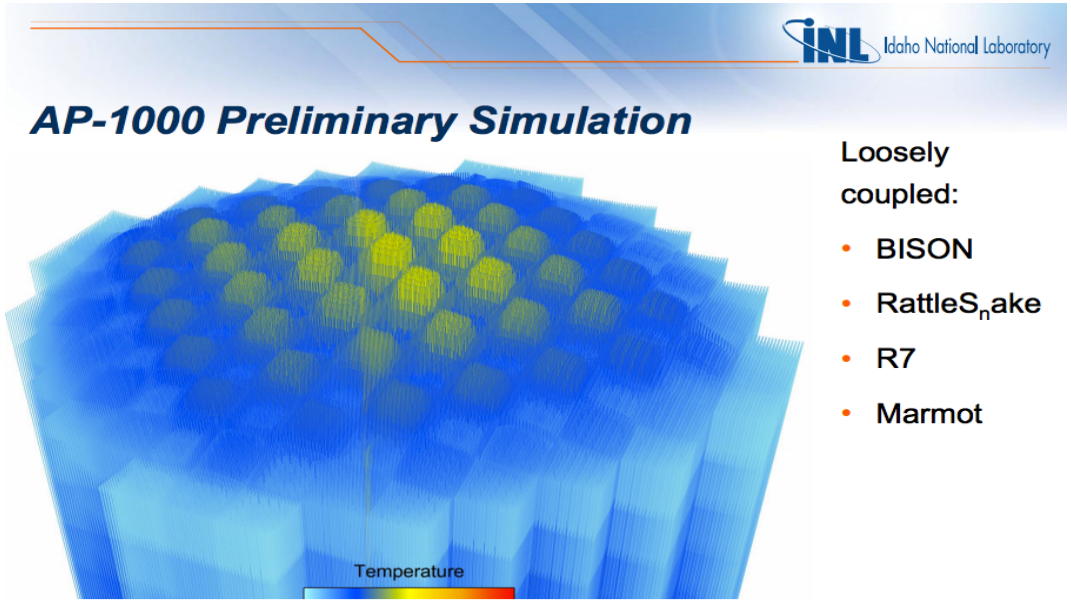


PB-FHR Fuel Geometry



Coupled Multi-Physics Analysis

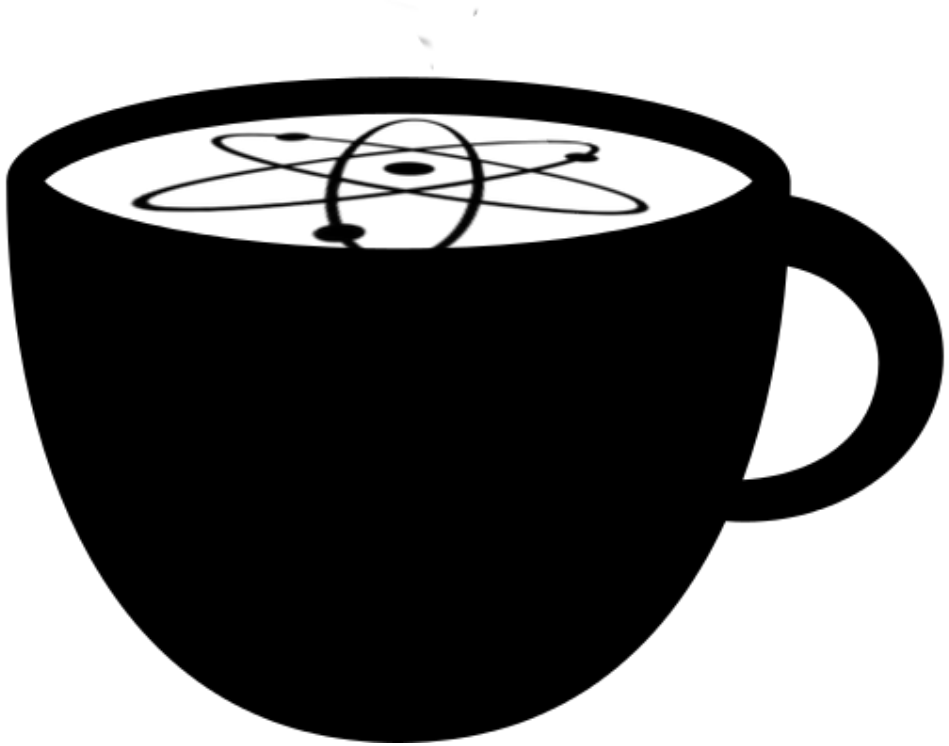
Severe accident **neutronics** and **thermal hydraulics** can be simulated beautifully for simple geometries and well studied materials. (below, INL BISON work.)



Temperature and magnified displacement results for an AP1000 core. Each rod shown is a unique BISON simulation.

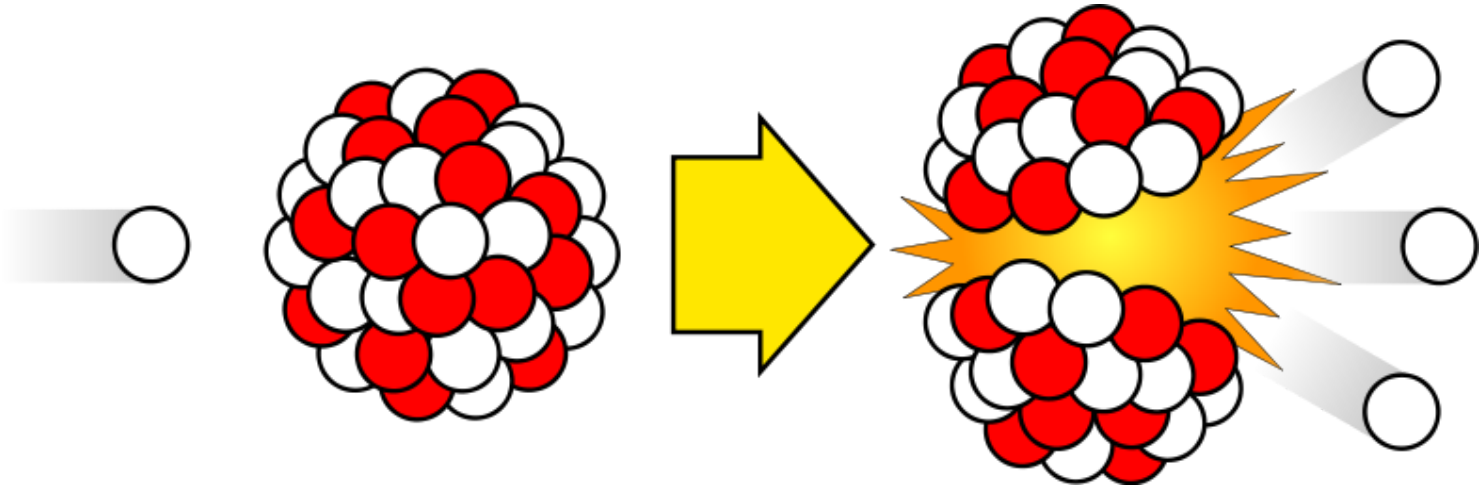
FHR, Accident Transient Analysis

- Collect experimental data
- Conduct algebraic, static, and benchmark simulations
- Develop 0D coupled neutronics/TH model (PyRK)
- Develop 3D neutronics/TH model
- Compare 0D and 3D simulations
- Couple additional physics (e.g. fuel performance)

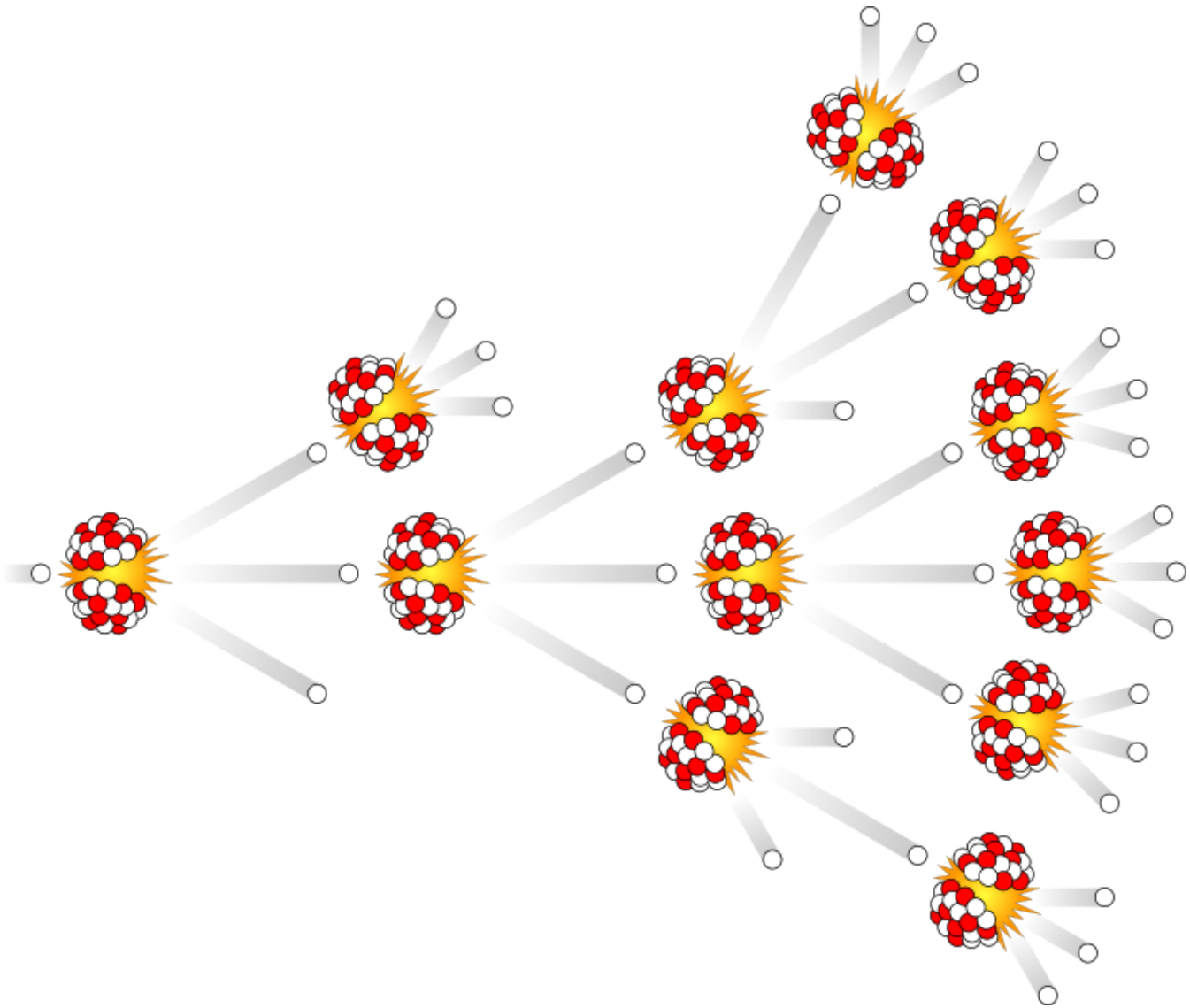


PyRK: Python for Reactor Kinetics

Review of Nuclear Reactor Kinetics



$$\sigma(E, \vec{r}, \hat{\Omega}, T, x, i)$$



$k = 1$

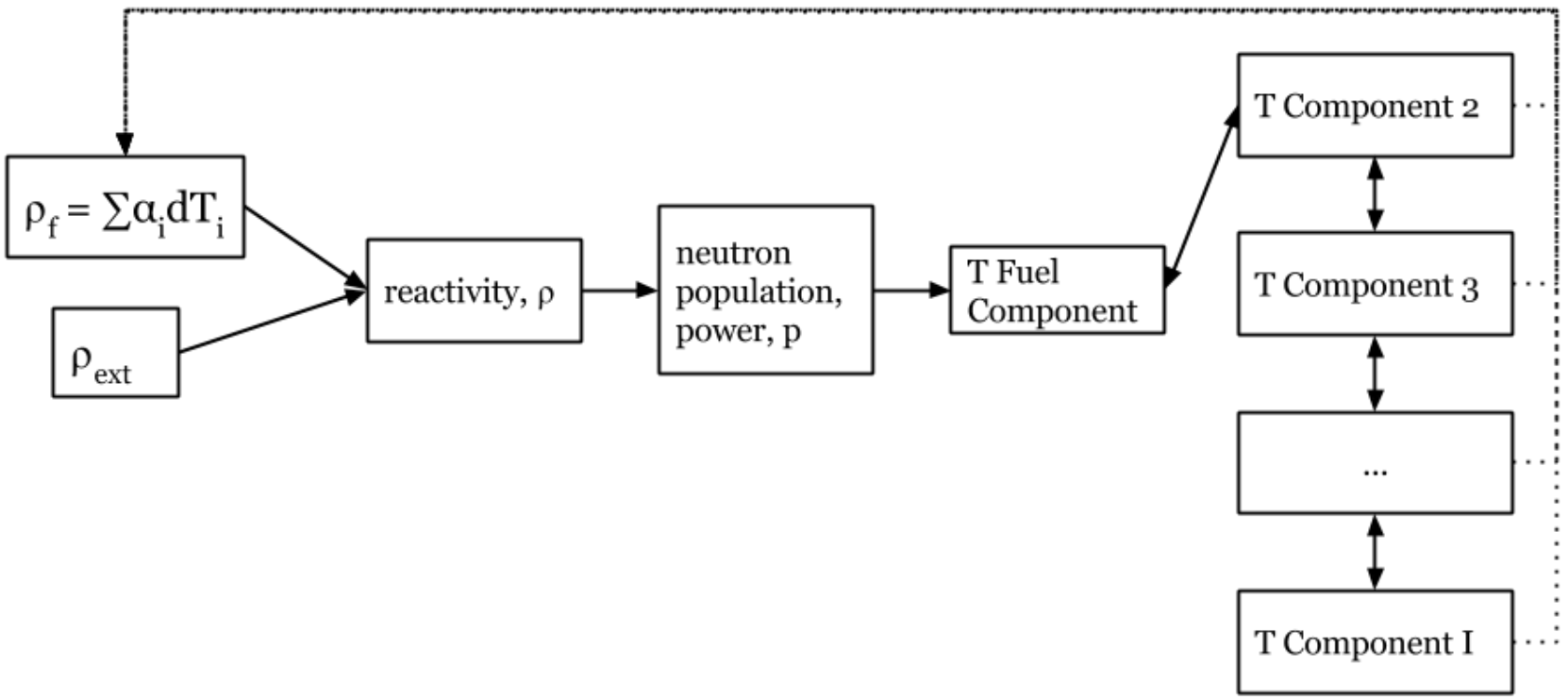
Reactivity

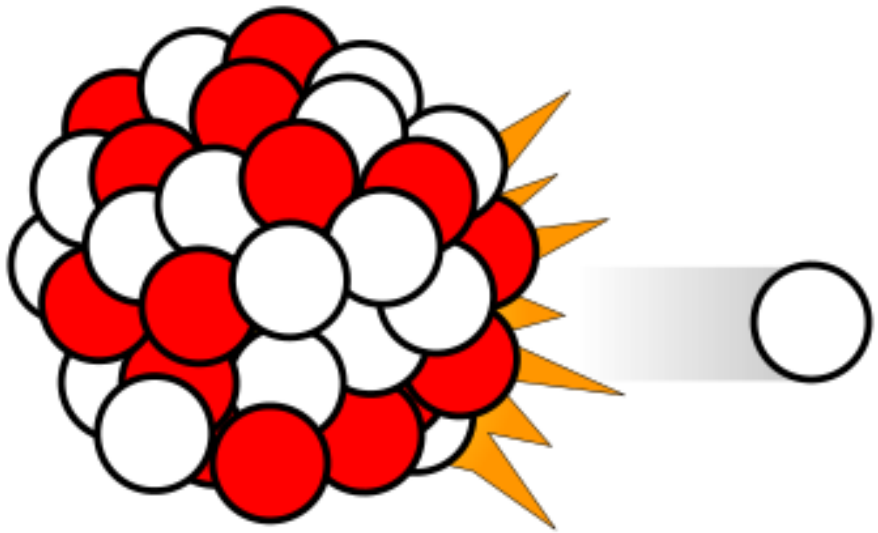
$$k = \text{"neutron multiplication factor"}$$

$$= \frac{\text{neutrons causing fission}}{\text{neutrons produced by fission}}$$

$$\rho = \frac{k - 1}{k}$$

$$\rho = \text{reactivity}$$


$\rho_f = \sum \alpha_i dT_i$
ρ_{ext}
reactivity, ρ
neutron population, power, p
T Fuel Component
T Component 2
T Component 3
...
T Component I



$\beta_i, \lambda_{d,i}$

PyRK

- 6-precursor-group,
- 11-decay-group Point Reactor Kinetics model
- Lumped Parameter thermal hydraulics model
- Object-oriented, geometry and material agnostic framework

Point Reactor Kinetics

$p =$ reactor power

$\rho(t, T_{fuel}, T_{cool}, T_{mod}, T_{refl}) =$ reactivity

$\beta =$ fraction of neutrons that are delayed

$\beta_j =$ fraction of delayed neutrons from precursor group j

$\zeta_j =$ concentration of precursors of group j

$\lambda_{d,j} =$ decay constant of precursor group j

$\Lambda =$ mean generation time

$\omega_k =$ decay heat from FP group k

$\kappa_k =$ heat per fission for decay FP group k

$\lambda_{FP,k} =$ decay constant for decay FP group k

$T_i =$ temperature of component i

$$\frac{d}{dt}\begin{bmatrix} p \\ \zeta_1 \\ . \\ \zeta_j \\ . \\ \zeta_J \\ \omega_1 \\ . \\ \omega_k \\ . \\ \omega_K \\ T_i \\ . \\ T_I \end{bmatrix} = \begin{bmatrix} \frac{\rho(t,T_i,\cdots)-\beta}{\Lambda}p + \sum_{j=1}^{j=J} \lambda_{d,j}\zeta_j \\ \frac{\beta_1}{\Lambda}p - \lambda_{d,1}\zeta_1 \\ . \\ \frac{\beta_j}{\Lambda}p - \lambda_{d,j}\zeta_j \\ . \\ \frac{\beta_J}{\Lambda}p - \lambda_{d,J}\zeta_J \\ \kappa_1 p - \lambda_{FP,1}\omega_1 \\ . \\ \kappa_k p - \lambda_{FP,k}\omega_k \\ . \\ \kappa_{kp} - \lambda_{FP,k}\omega_k \\ f_i(p, C_{p,i}, T_i, \cdots) \\ . \\ f_I(p, C_{p,I}, T_I, \cdots) \end{bmatrix}$$

Lumped Parameter Heat Transfer

The heat flow out of body i is the sum of surface heat flow by conduction, convection, radiation, and other mechanisms to each adjacent body, j:

$$\begin{aligned} Q &= Q_i + \sum_j Q_{ij} \\ &= Q_i + \sum_j \frac{T_i - T_j}{R_{th,ij}} \end{aligned}$$

$\dot{Q}$ = total heat flow out of body i $[J \cdot s^{-1}]$

Q_i = other heat transfer, a constant $[J \cdot s^{-1}]$

T_i = temperature of body i $[K]$

T_j = temperature of body j $[K]$

j = adjacent bodies $[-]$

R_{th} = thermal resistence of the component $[K \cdot s \cdot J^{-1}]$.

Quality Control

“ Organized Skepticism. Scientists are critical: All ideas must be tested and are subject to rigorous structured community scrutiny.” - R.K. Merton, 1942

Unit Checking

```
class Material(object):
    """This class represents a material. Its attributes are material properties
    and behaviors."""

    def __init__(self, name=None,
                 k=0*units.watt/units.meter/units.kelvin,
                 cp=0*units.joule/units.kg/units.kelvin,
                 dm=DensityModel()):
        """Initalizes a material

        :param name: The name of the component (i.e., "fuel" or "cool")
        :type name: str.
        :param k: The thermal conductivity of the component
        :type k: float.
        :param cp: specific heat capacity, :math:`c_p`, in :math:`J/kg-K`
        :type cp: float, pint.unit.Quantity :math:`J/kg-K`
        :param dm: The density of the material
        :type dm: DensityModel object
        """
        self.name = name
        self.k = k.to('watt/meter/kelvin')
        validation.validate_ge("k", k, 0*units.watt/units.meter/units.kelvin)
        self.cp = cp.to('joule/kg/kelvin')
        validation.validate_ge("cp", cp, 0*units.joule/units.kg/units.kelvin)
        self.dm = dm
```

In PyRK, the Pint package (pint.readthedocs.org/en/0.6/) is used keeping track of units, converting between them, and throwing errors when unit conversions are not sane.

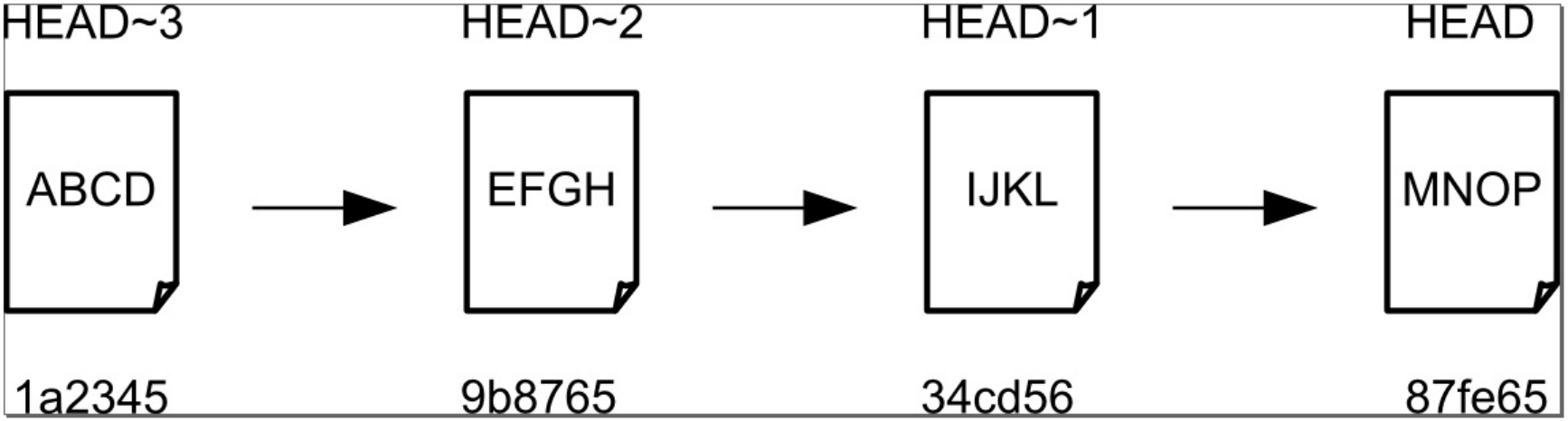
Backing Up Files

- **Good:** hope
- **Better:** nightly emails
- **Best:** remote version control

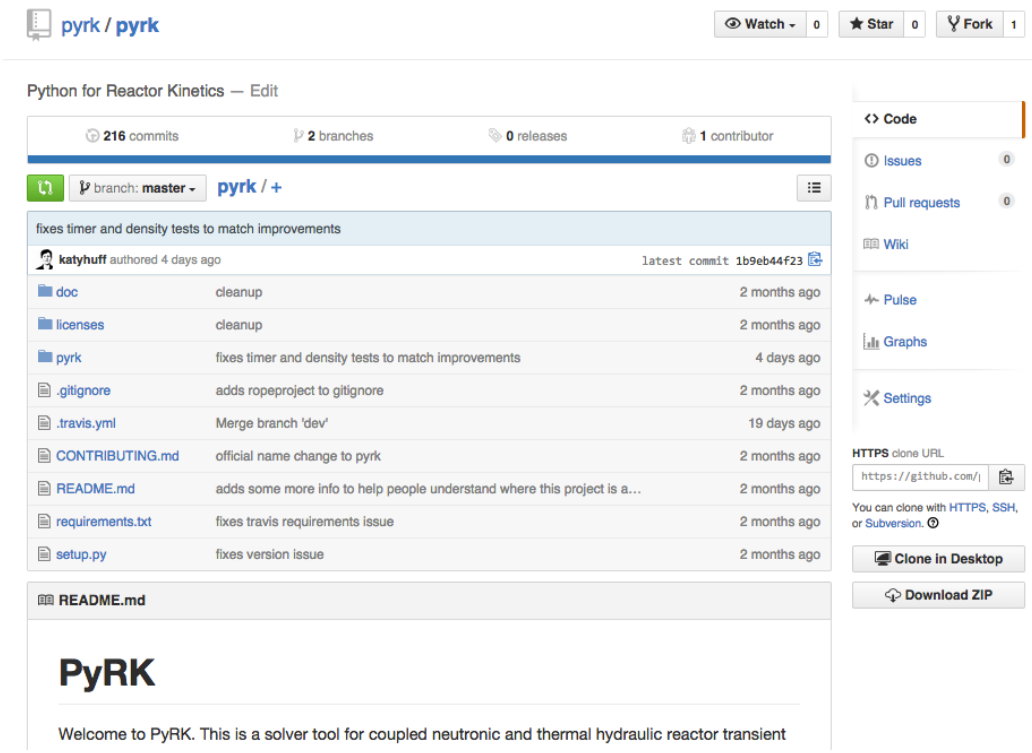
Version Control Systems: cvs, svn, hg, **git**

Managing Changes

- **Good:** naming convention
- **Better:** clever naming convention
- **Best:** local version control

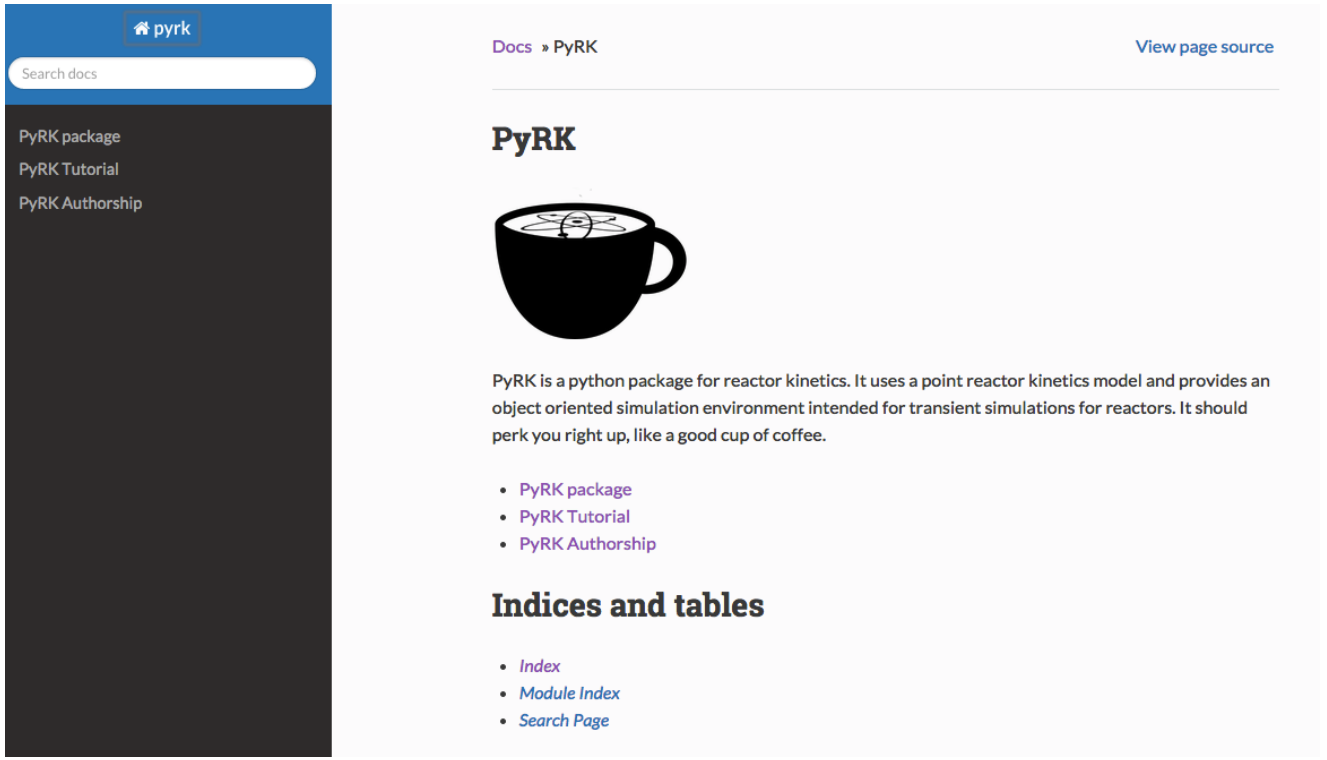


Version Control



Keeping track of versions of the code makes it possible to experiment without fear and placing the code online encourages use and collaboration.

Automated Documentation



Automated documentation creates a browsable website explaining the most recent version of the code.

Error Detection

“ The scientific method’s central motivation is the ubiquity of error—the awareness that mistakes and self-delusion can creep in absolutely anywhere and that the scientist’s effort is primarily expended in recognizing and rooting out error. ” - Donoho, 2009.

Error Detection

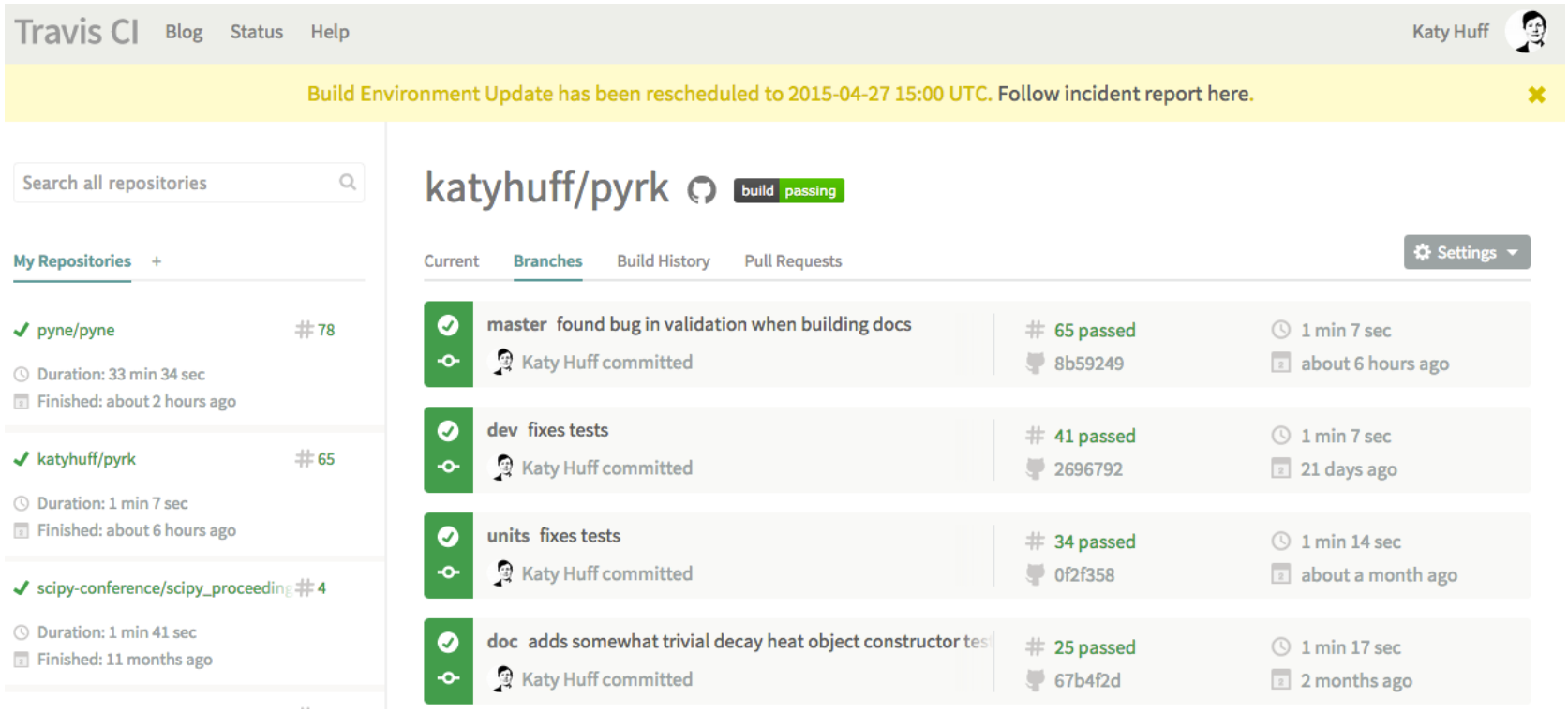
- **Good:** show results to experts
- **Better:** integration testing
- **Best:** unit test suite, continuous integration

Test Suite

```
from nose.tools import assert_equal, assert_almost_equal, assert_true, \
    assert_false, assert_raises, assert_is_instance, with_setup

import reactivity_insertion as ri
import timer
from ur import units

ti = timer.Timer(t0=0.0*units.seconds,
                 tf=10.0*units.seconds,
                 dt=0.01*units.seconds)


def test_default_ri():
    default = ri.ReactivityInsertion(timer=ti)
    assert_equal(default.reactivity(0), 0.0*units.delta_k)
    assert_equal(default.reactivity(1), 0.0*units.delta_k)
    assert_equal(default.reactivity(10), 0.0*units.delta_k)
    assert_equal(default.reactivity(1000), 0.0*units.delta_k)


def test_default_step_ri():
    step = ri.StepReactivityInsertion(timer=ti)
    assert_equal(step.t_step, 1.0*units.seconds)
    assert_equal(step.reactivity(0), 0.0*units.delta_k)
    assert_equal(step.reactivity(1), 0.0*units.delta_k)
    assert_equal(step.reactivity(101), 1.0*units.delta_k)
    assert_equal(step.reactivity(1000), 1.0*units.delta_k)
```

The classes and functions that make up the code are tested individually for robustness using nose.

Continuous Integration



The tests are run every time a change is made to the repository online. The results are public. If a main branch has a failed test, I get an email.

PB-FHR Reactivity Insertion

Pebble-Bed, Fluoride Salt Cooled, High-Temperature Reactor

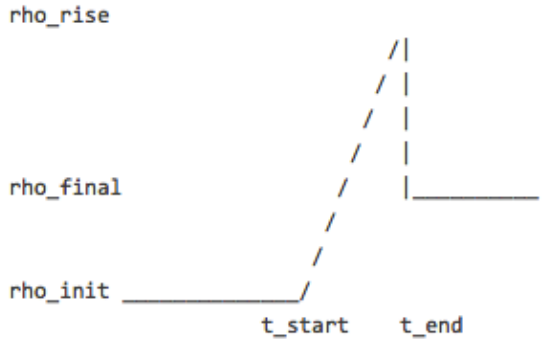
- Molten FLiBe Coolant
- Annular Core
- Annular Pebble Fuel
- Steady Inlet Temperature
- Ramp Reactivity Insertion, 600pcm over 10s

Ramp Reactivity Insertion

class **pyrk.reactivity_insertion.RampReactivityInsertion**(*timer, t_start=<Quantity(1.0, 'second')>, t_end=<Quantity(2.0, 'second')>, rho_init=<Quantity(0.0, 'delta_k')>, rho_rise=<Quantity(1.0, 'delta_k')>, rho_final=<Quantity(1.0, 'delta_k')>*) [source]

Bases: `pyrk.reactivity_insertion.ReactivityInsertion`

Returns a ramp:



f(*x*) [source]

slope() [source]

Ramp Reactivity Insertion

```
# External Reactivity
from reactivity_insertion import RampReactivityInsertion
rho_ext = RampReactivityInsertion(timer=ti,
                                  t_start=t_feedback + 10.0*units.seconds,
                                  t_end=t_feedback + 20.0*units.seconds,
                                  rho_init=0.0*units.delta_k,
                                  rho_rise=600.0*units.pcm,
                                  rho_final=600.0*units.pcm)
```

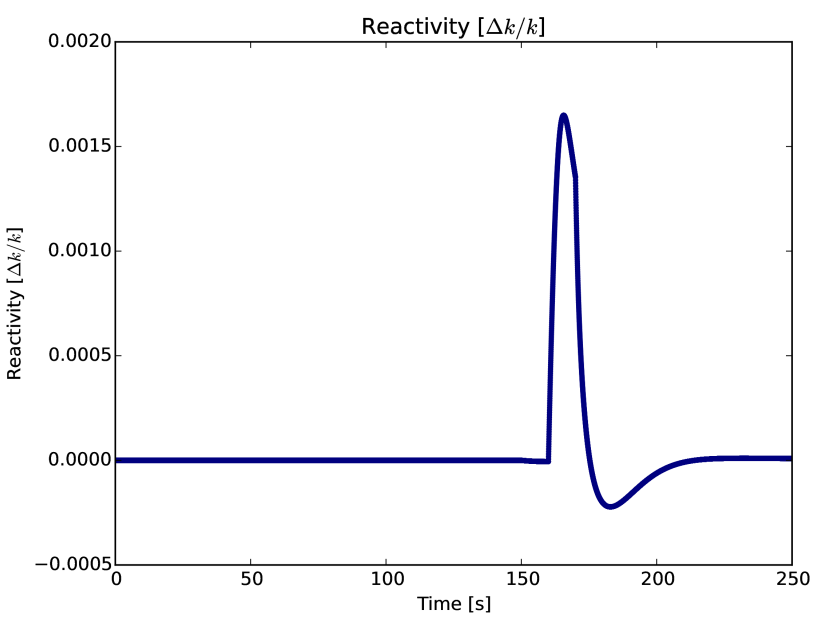
Components

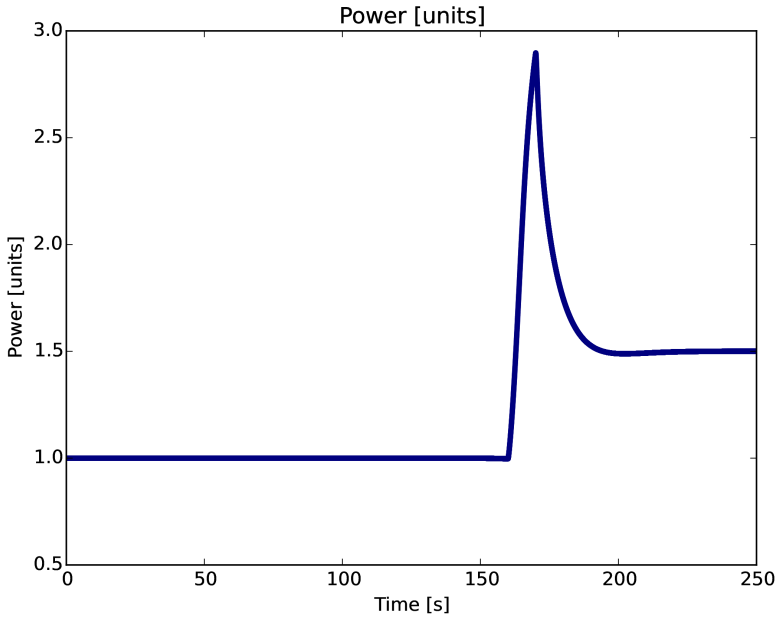
```
fuel = th.THComponent(name="fuel",
                      mat=TRISO(),
                      vol=vol_fuel,
                      T0=t_fuel,
                      alpha_temp=alpha_fuel,
                      timer=ti,
                      heatgen=True,
                      power_tot=power_tot/n_pebbles,
                      sph=True,
                      ri=r_mod,
                      ro=r_fuel
                      )

mod = th.THComponent(name="mod",
                     mat=Graphite(),
                     vol=vol_mod
```

Heat Transfer

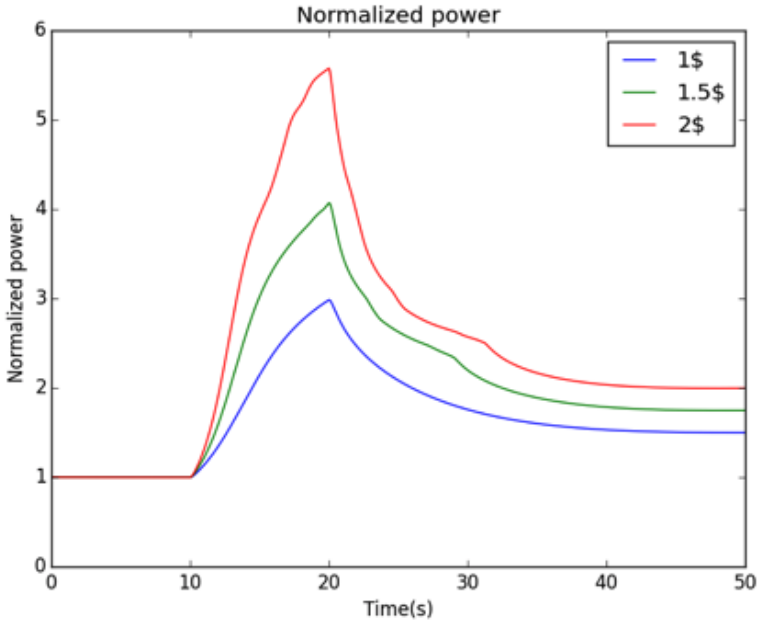
```
# The coolant convects to the pebbles
cool.add_convection('pebble', h=h_cool, area=a_pb)
cool.add_advection('cool', m_flow/n_pebbles, t_inlet, cp=cool.cp)
```

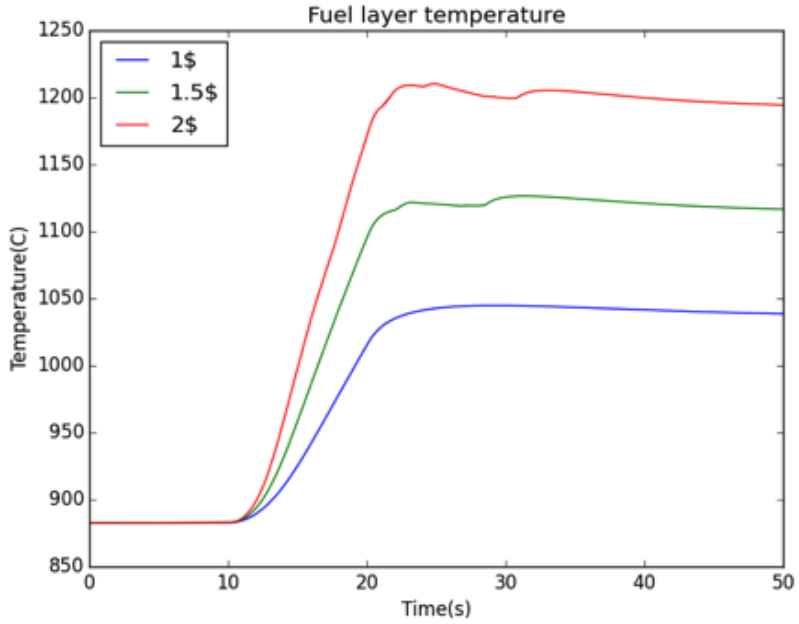
Ramp Reactivity Insertion



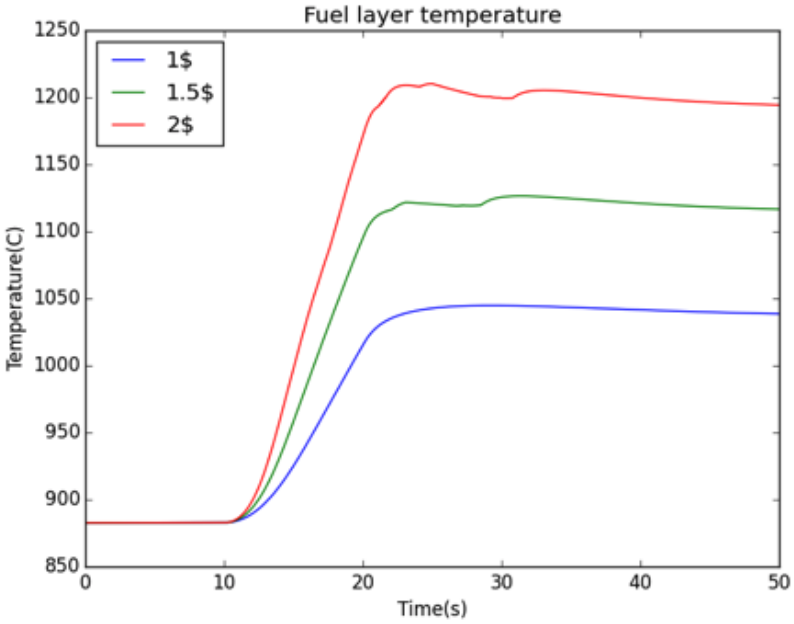


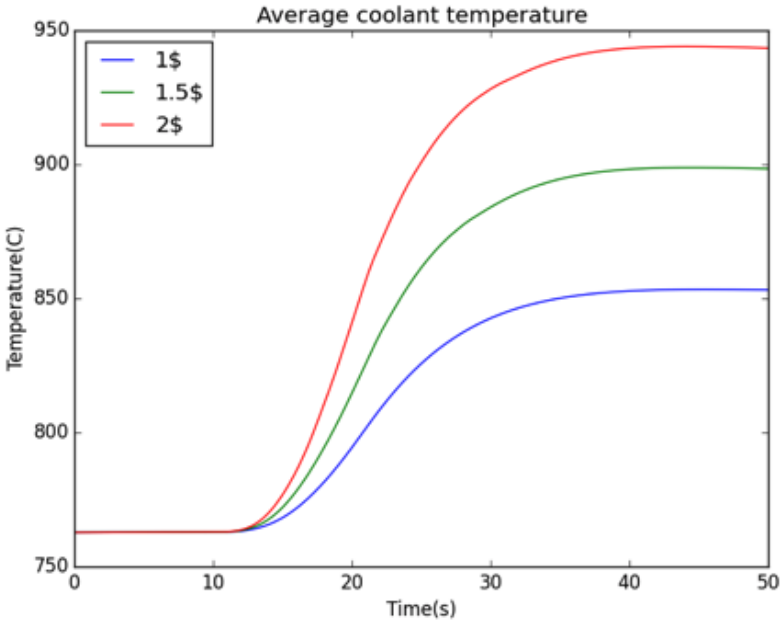
Fuel Layer Temperature

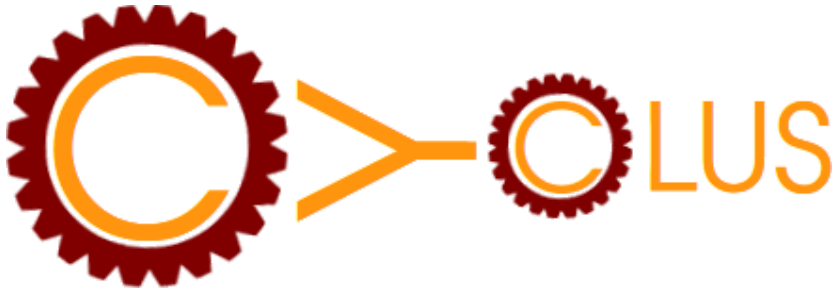




Coolant Temperature



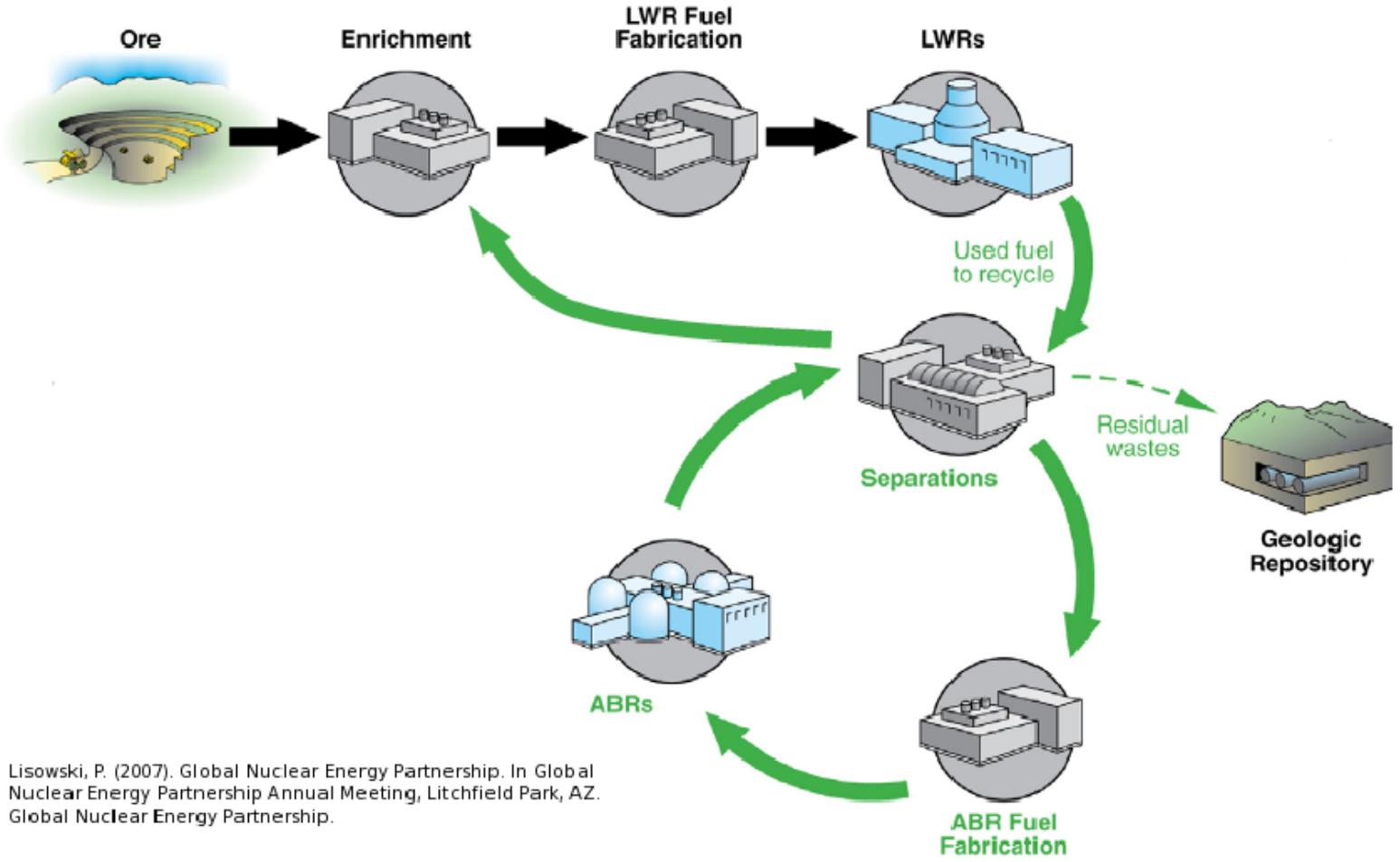




A Nuclear Fuel Cycle Simulation Framework

The Nuclear Fuel Cycle

Hundreds of discrete facilities mine, mill, convert, fabricate, transmute, recycle, and store nuclear material.



Lisowski, P. (2007). Global Nuclear Energy Partnership. In Global Nuclear Energy Partnership Annual Meeting, Litchfield Park, AZ. Global Nuclear Energy Partnership.

Fuel Cycle Metrics

- Mass Flow
  - inventories, decay heat, radiotoxicity,
  - proliferation resistance and physical protection (PRPP) indices.
- Cost
  - levelized cost of electricity,
  - facility life cycle costs.
- Economics
  - power production, facility deployments,
  - dynamic pricing and feedback.
- Disruptions
  - reliability, safety,
  - system robustness.

Current Simulators

- CAFCA (MIT)
- COSI (CEA)
- DANESS (ANL)
- DESAE (Rosatom)
- Evolcode (CIEMAT)
- FAMILY (IAEA)
- GENIUSv1 (INL)
- GENIUS v2 (UW)
- NFCSim (LANL)
- NFCSS (IAEA)
- NUWASTE (NWTRB)
- ORION (NNL)
- MARKAL (BNL)
- VISION (INL)

State of the Art

Performance

- **Speed** interactive time scales
- **Fidelity:** detail commensurate with existing challenges
- **Detail:** discrete material and agent tracking
- **Regional Modeling:** enabling international socio-economics

Beyond the State of the Art

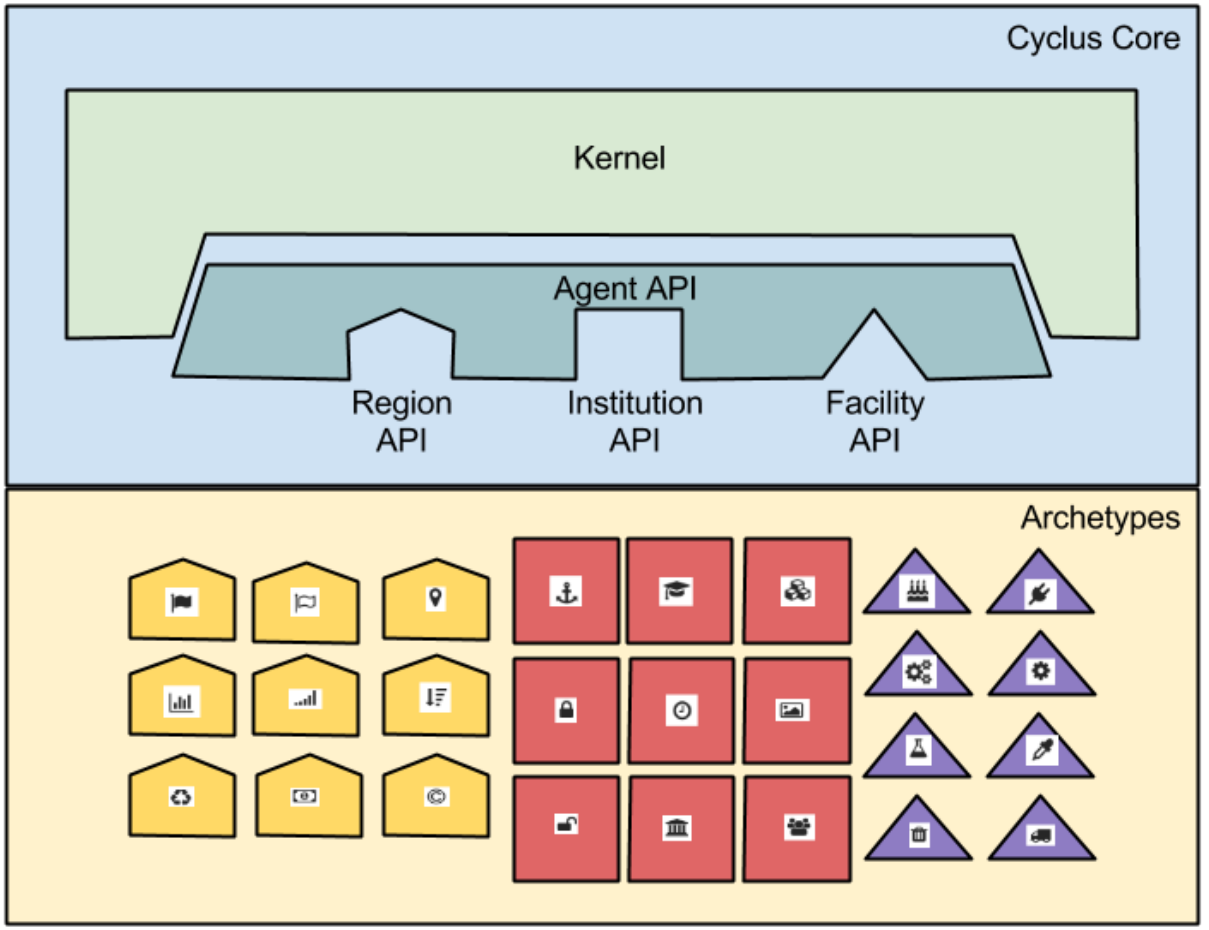
Access

- **Openness:** for collaboration, validation, and code sustainability.
- **Usability:** for a wide range of user sophistication

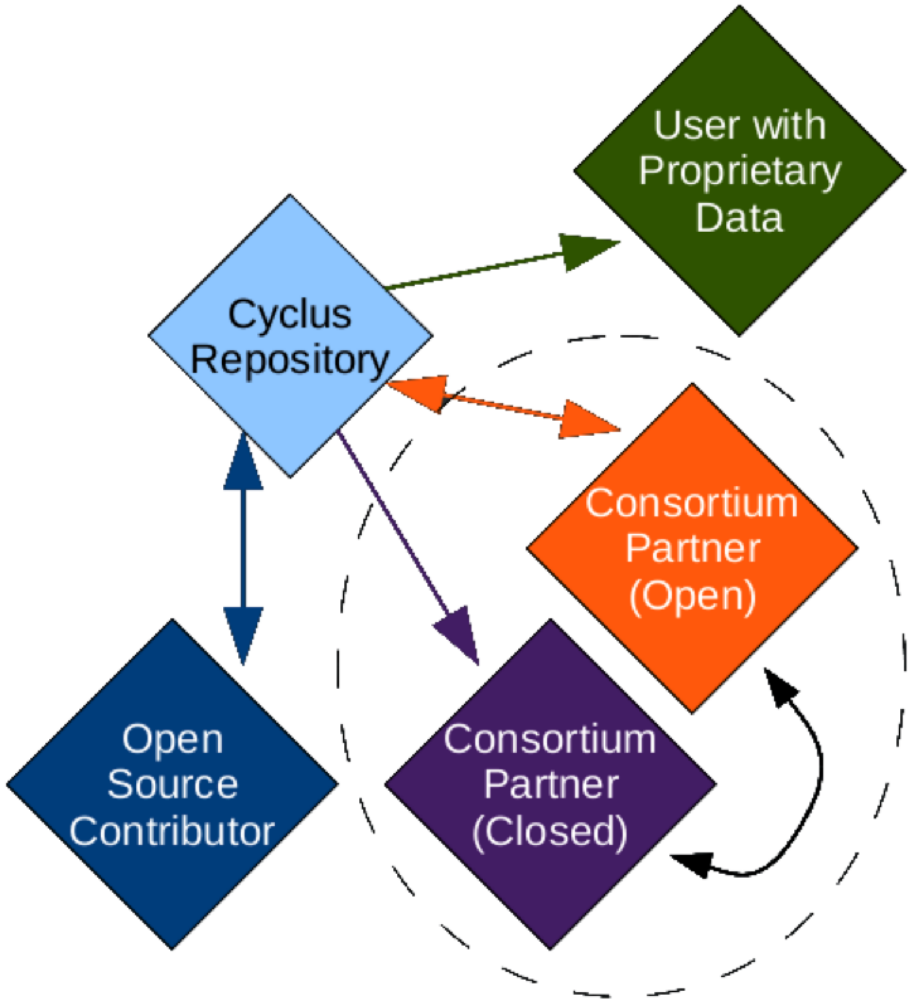
Extensibility

- **Modularity:** core infrastructure independent of proprietary or sensitive data and models
- **Flexibility** with a focus on robustness for myriad potential developer extensions.

Extensibility



Openness


User with Proprietary Data
Cyclus Repository
Consortium Partner (Open)
Open Source Contributor
Consortium Partner (Closed)

Growing Ecosystem

Name	**Description**	**Institutions**
Bright-lite	Spectral model reactor facility	UT-Austin
Nuclear Fuel Invetory Model	ORIGEN-based, reactor facility	UT-Knoxville
CommodConverter	Conversion or storage facility	Berkeley UT-Knoxville
MktDrivenInst	Market-driven deployment	Berkeley LLNL
SeparationsMatrix	Aqueous Separations	Berkeley LLNL
StreamBlender	Fuel reprocessing/fabrication	Berkeley LLNL

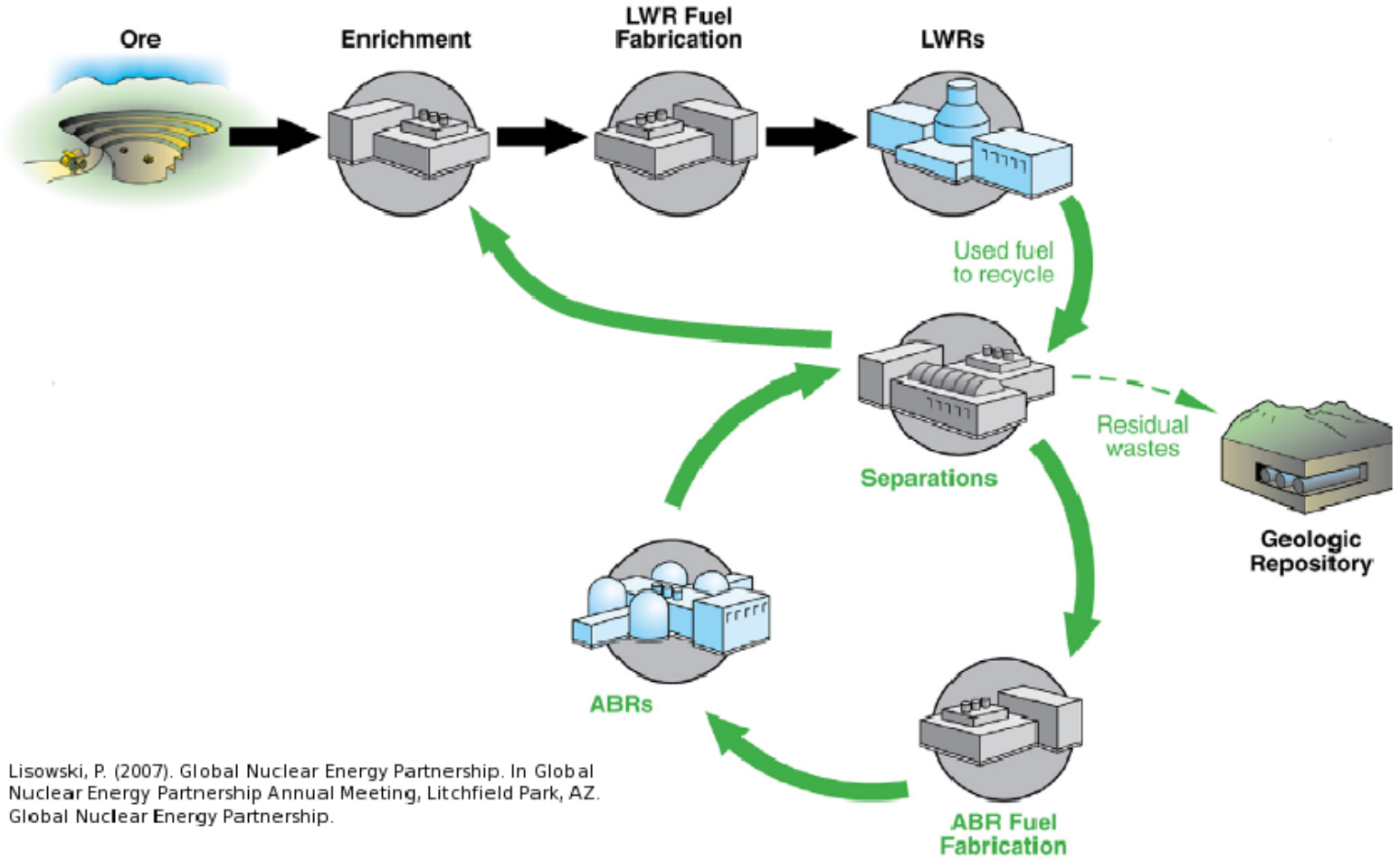
...Well Beyond

Algorithmic Sophistication

- **Efficient:** memory-efficient isotope tracking
- **Customizable:** constrained fuel supply
- **Dynamic:** isotopic-quality-based resource routing
- **Physics-based:** fuel fungibility

Agent Based Systems Analysis

An agent-based simulation is made up of actors and communications between those actors.



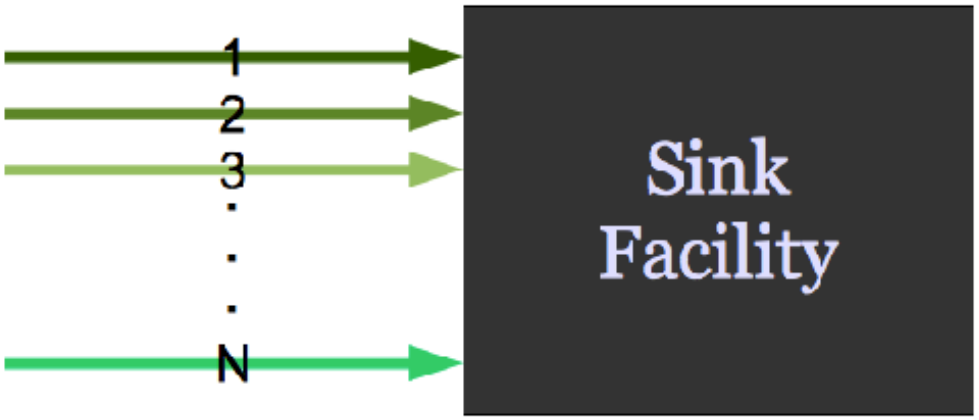
Lisowski, P. (2007). Global Nuclear Energy Partnership. In Global Nuclear Energy Partnership Annual Meeting, Litchfield Park, AZ. Global Nuclear Energy Partnership.

Agent Based Systems Analysis

A facility might create material.

Agent Based Systems Analysis

It might request material.

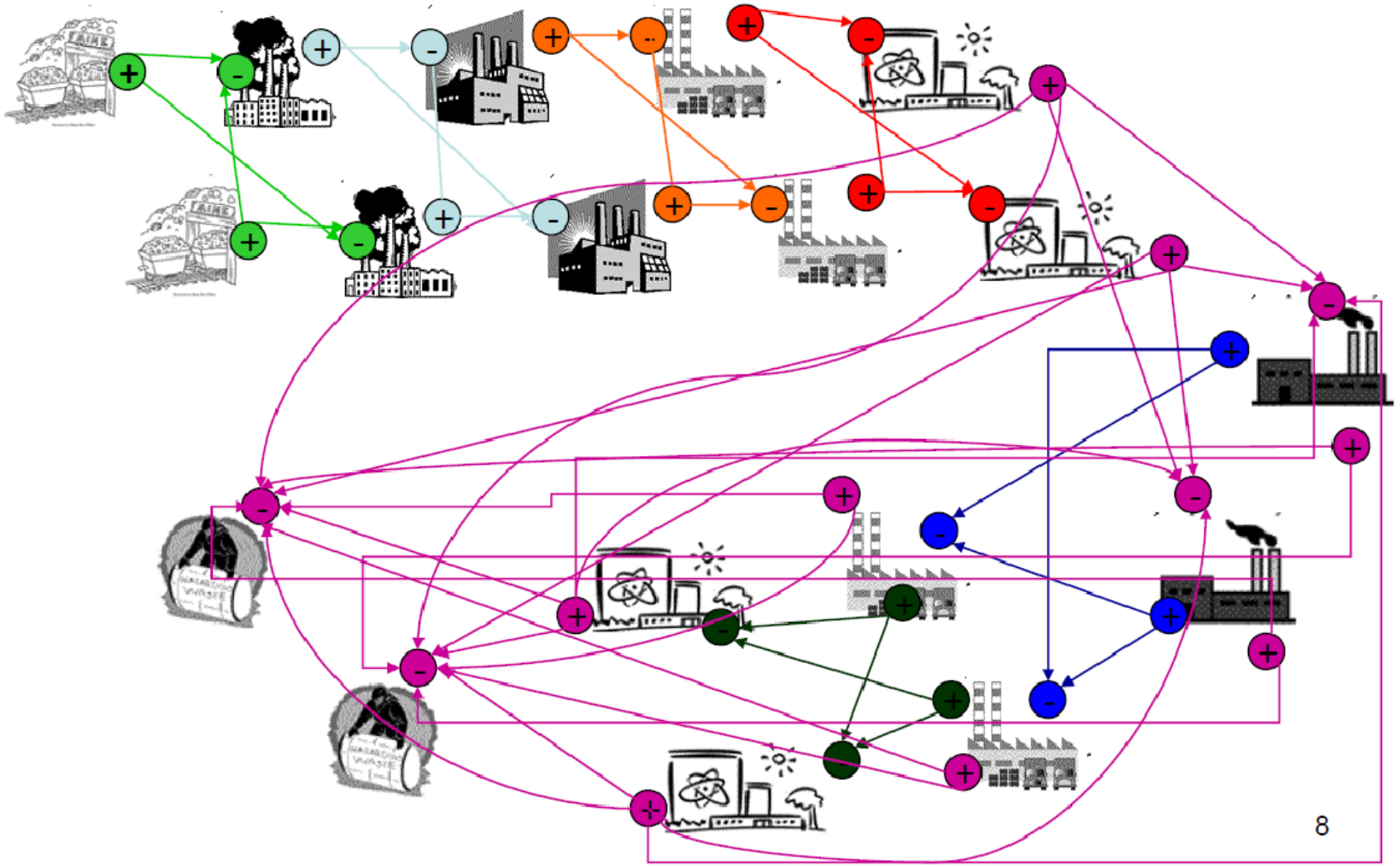


Agent Based Systems Analysis

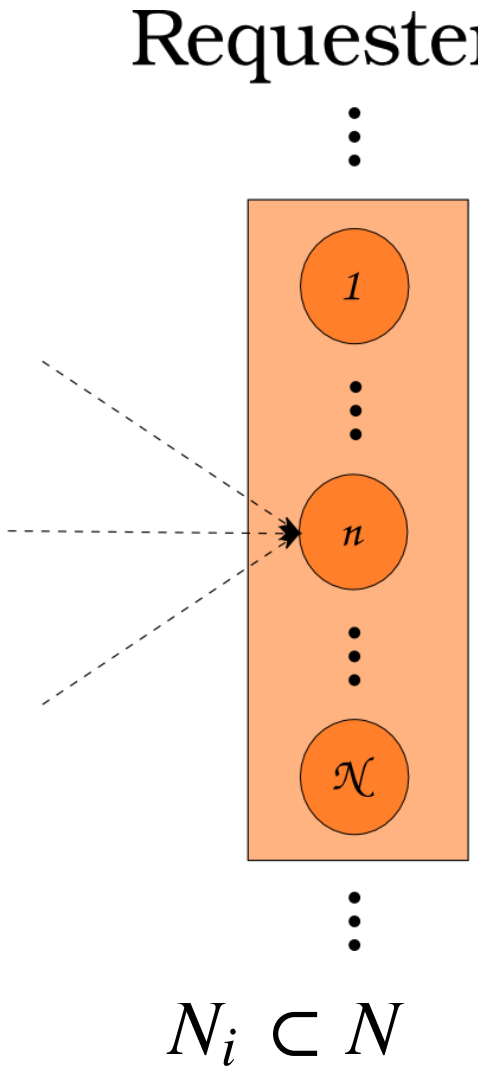
It might do both.

Agent Based Systems Analysis

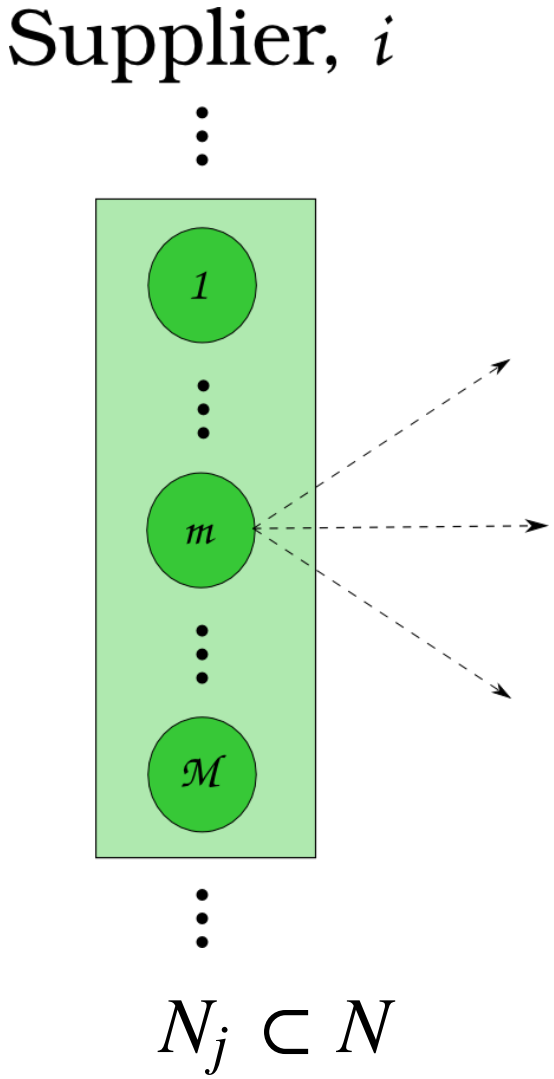
Even simple fuel cycles have many independent agents.



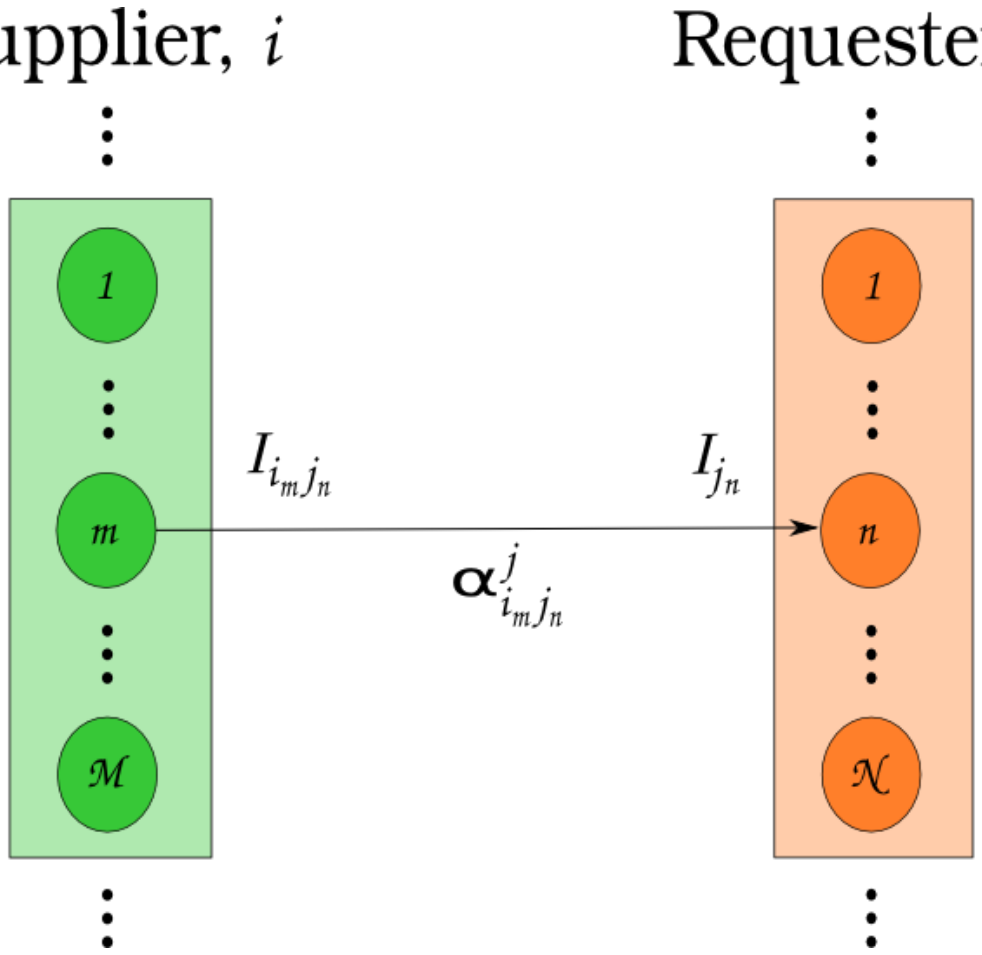
Dynamic Resource Exchange



Dynamic Resource Exchange

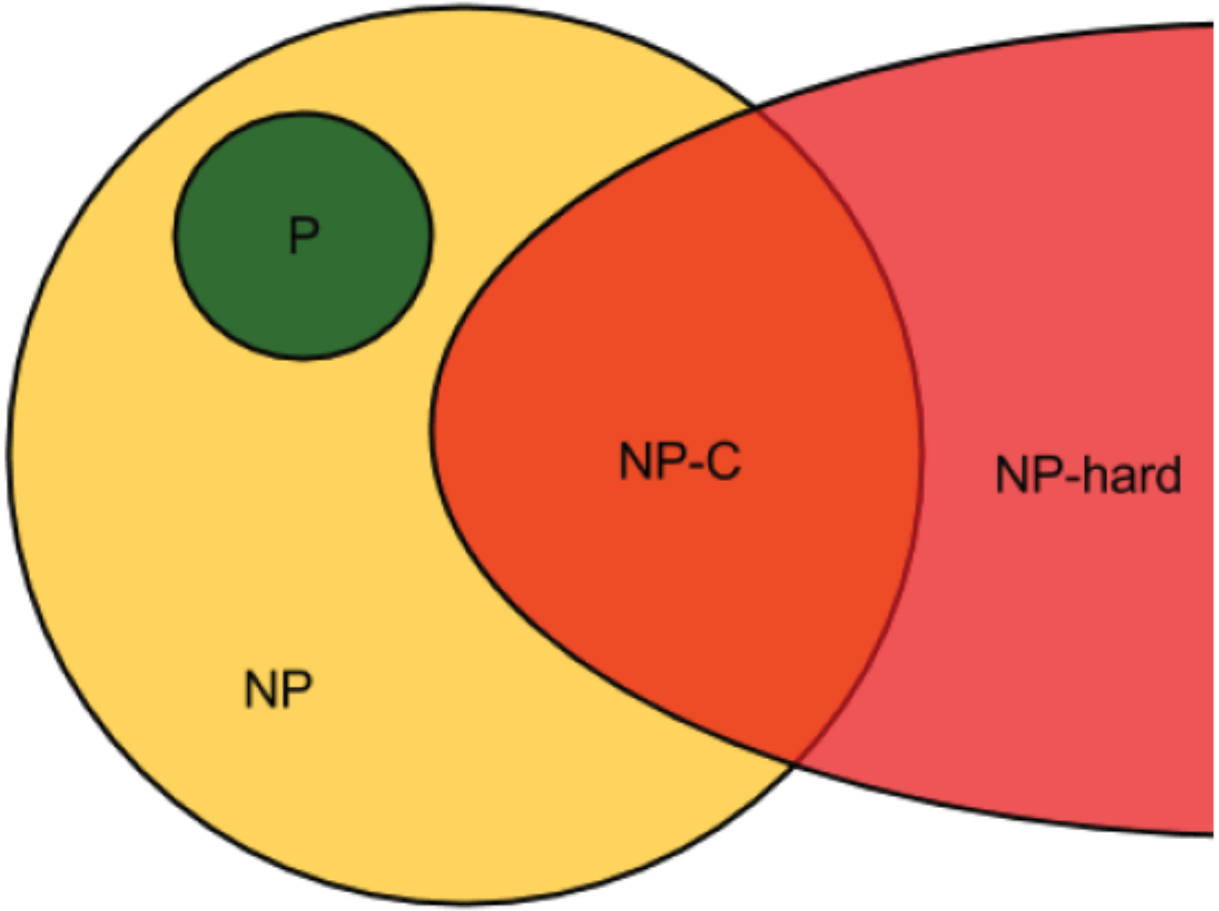


Dynamic Resource Exchange



$$N_i \cup N_j = N$$

Feasibility vs. Optimization



If a decision problem is in NP-C, then the corresponding optimization problem is NP-hard.

Multi-Commodity Transportation Formulation

Set	Description
S	suppliers
R	requesters
I	all supply nodes
I_s	nodes for a supplier s
J	all request nodes
J_r	nodes for a requester r
K_s	constraints for a supplier s
K_r	constraints for a requester r
X	the feasible set of flows between producers and consumers

Variable	Description
$c_{i,j}$	the unit cost of flow from producer node i to consumer node j
$x_{i,j}$	a decision variable, the flow from producer node i to consumer node j
$a^k_{i,j}$	the constraint coefficient for constraint k on flow between nodes i and j
b^k_s	the constraining value for constraint k of supplier s
b^k_r	the constraining value for constraint k of requester r
$\tilde{x_j}$	the requested quantity associated with request node j

Multi-Commodity Transportation Formulation

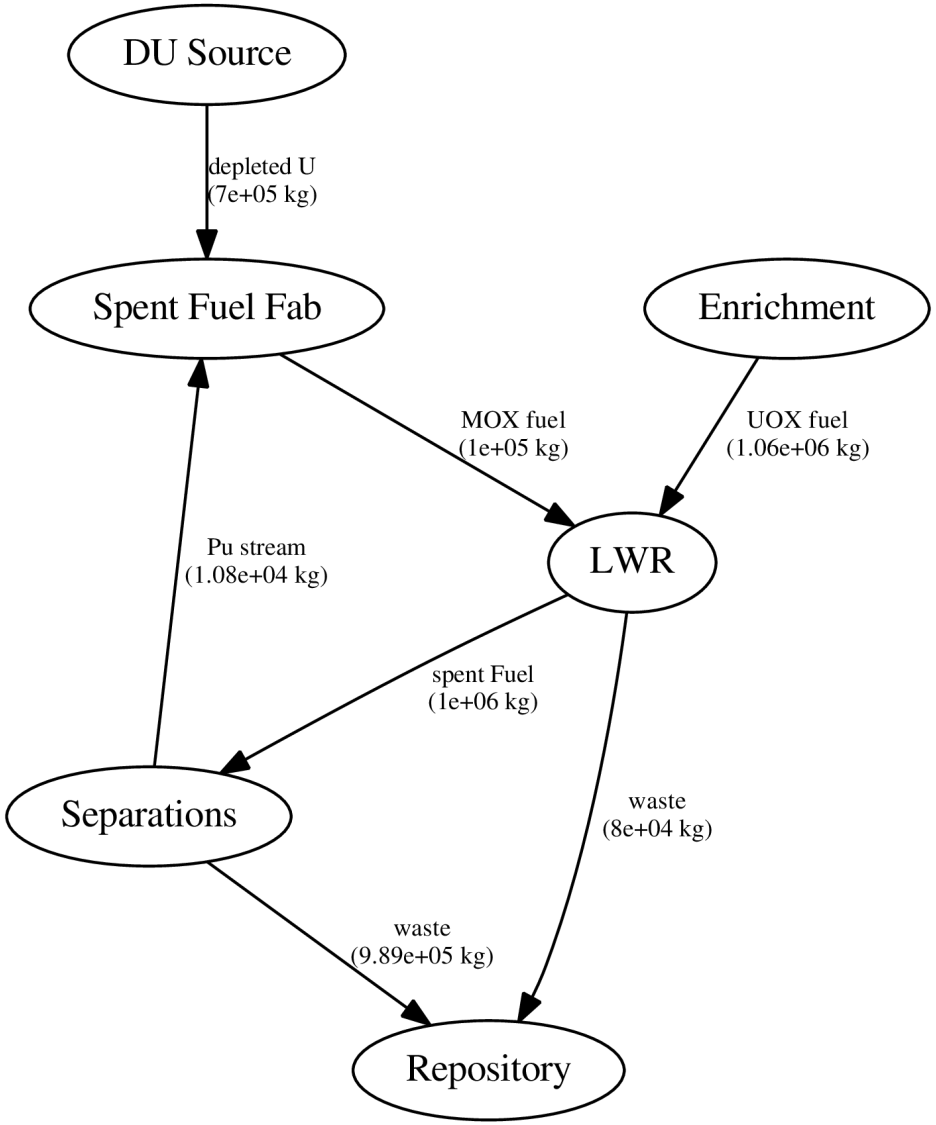
$$\min_x z = \sum_{i \in I} \sum_{j \in J} c_{i,j} x_{i,j}$$

$$s.t \sum_{i \in I_s} \sum_{j \in J} a^k_{i,j} x_{i,j} \leq b^k_s \qquad \forall k \in K_s, \forall s \in S$$

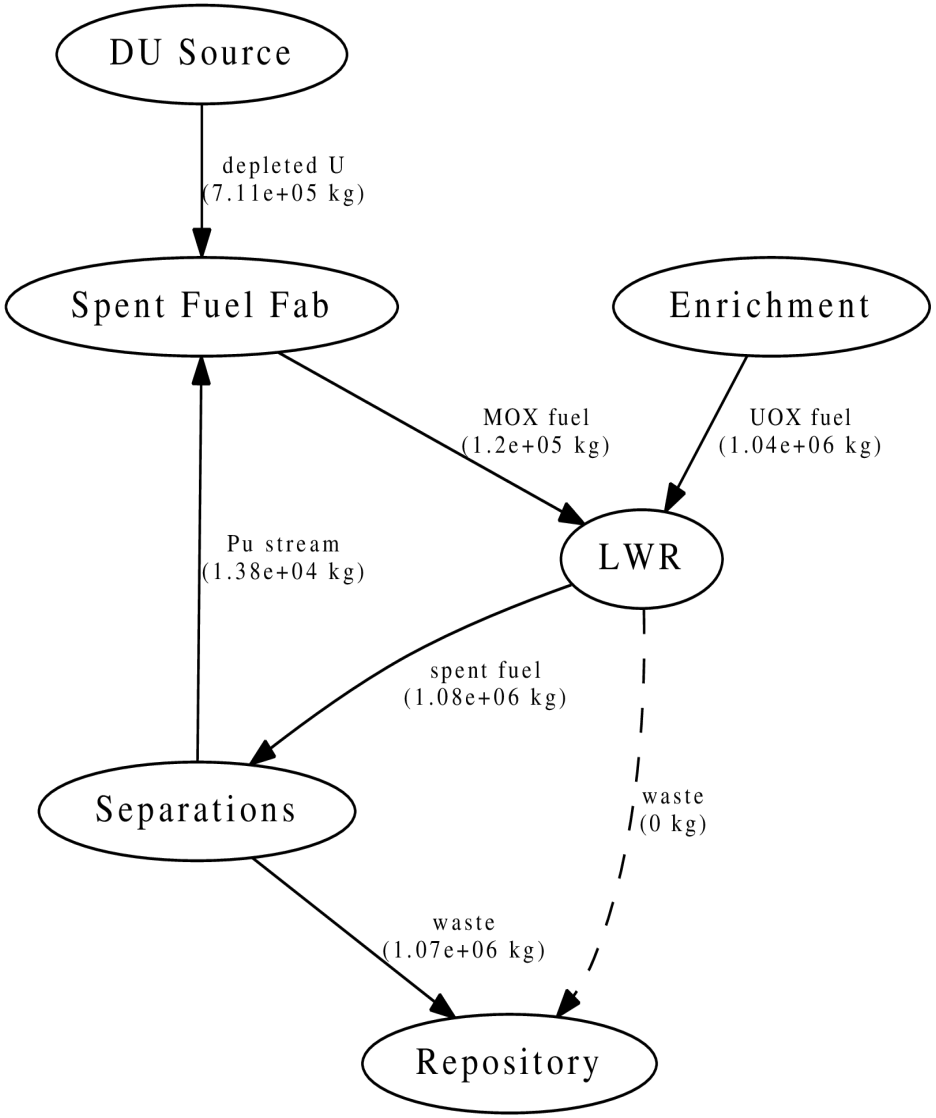
$$\sum_{J \in J_r} \sum_{i \in I} a^k_{i,j} x_{i,j} \leq b^k_r \qquad \forall k \in K_r, \forall r \in R$$

$$x_{i,j} \in [0, x_j] \qquad \forall i \in I, \forall j \in J$$

Dynamic Resource Exchange



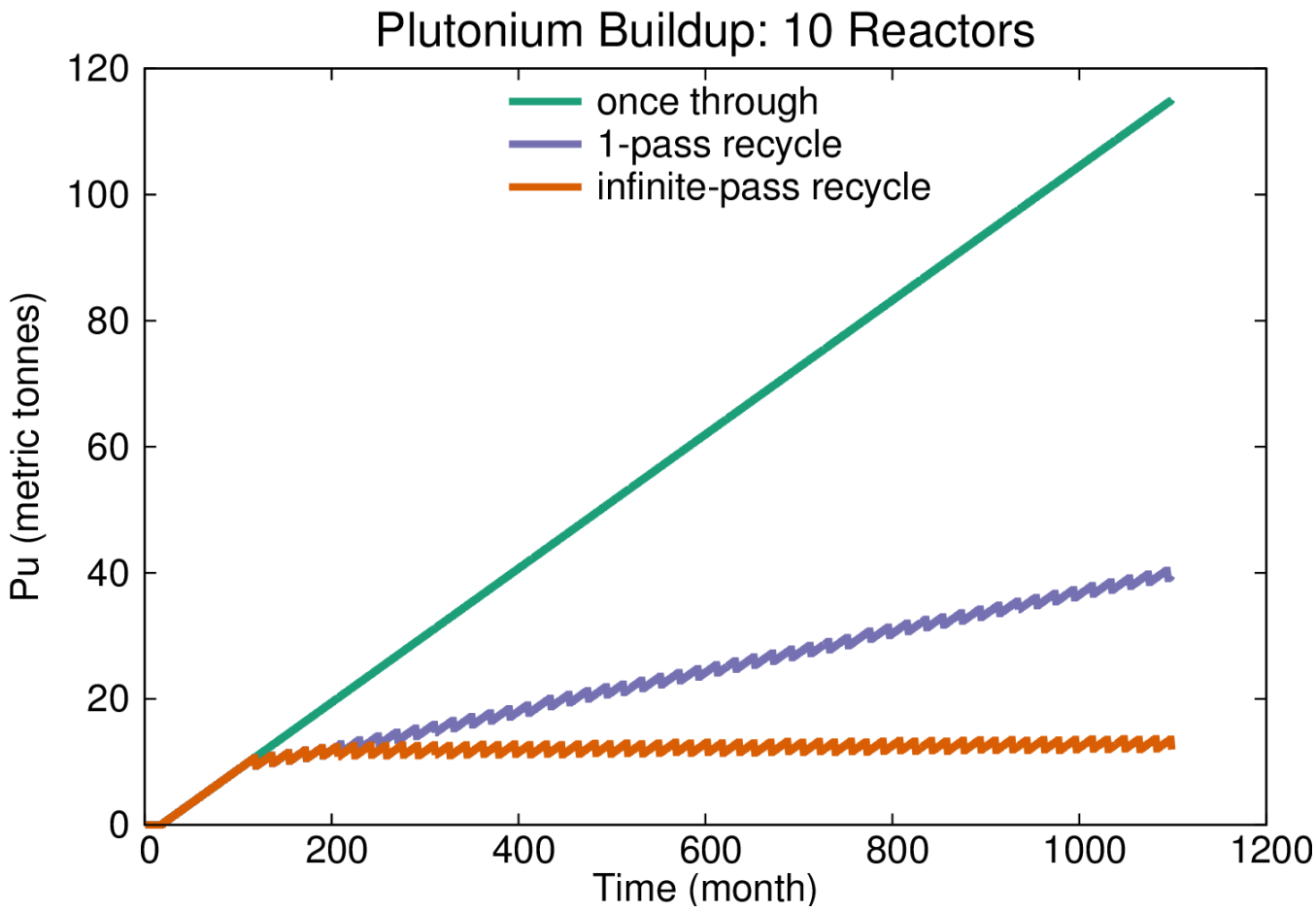
Dynamic Resource Exchange



Dynamic Resource Exchange

```
   <fuel>
     <incommodity>mox</incommodity>
-    <outcommodity>waste</outcommodity>
+    <outcommodity>spent_fuel</outcommodity>
     <inrecipe>mox_fresh_fuel</inrecipe>
     <outrecipe>mox_spent_fuel</outrecipe>
   </fuel>
```

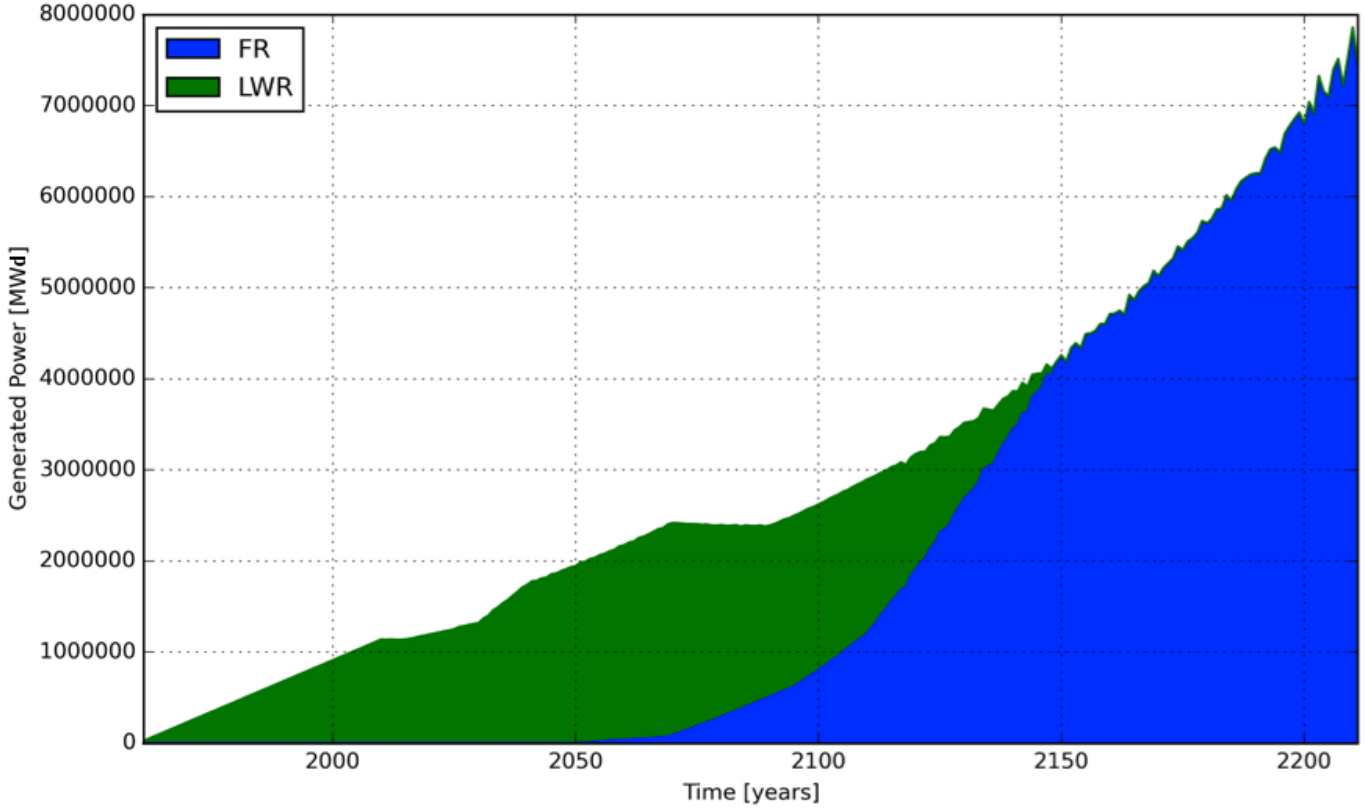
Dynamic Resource Exchange



Transition Analysis

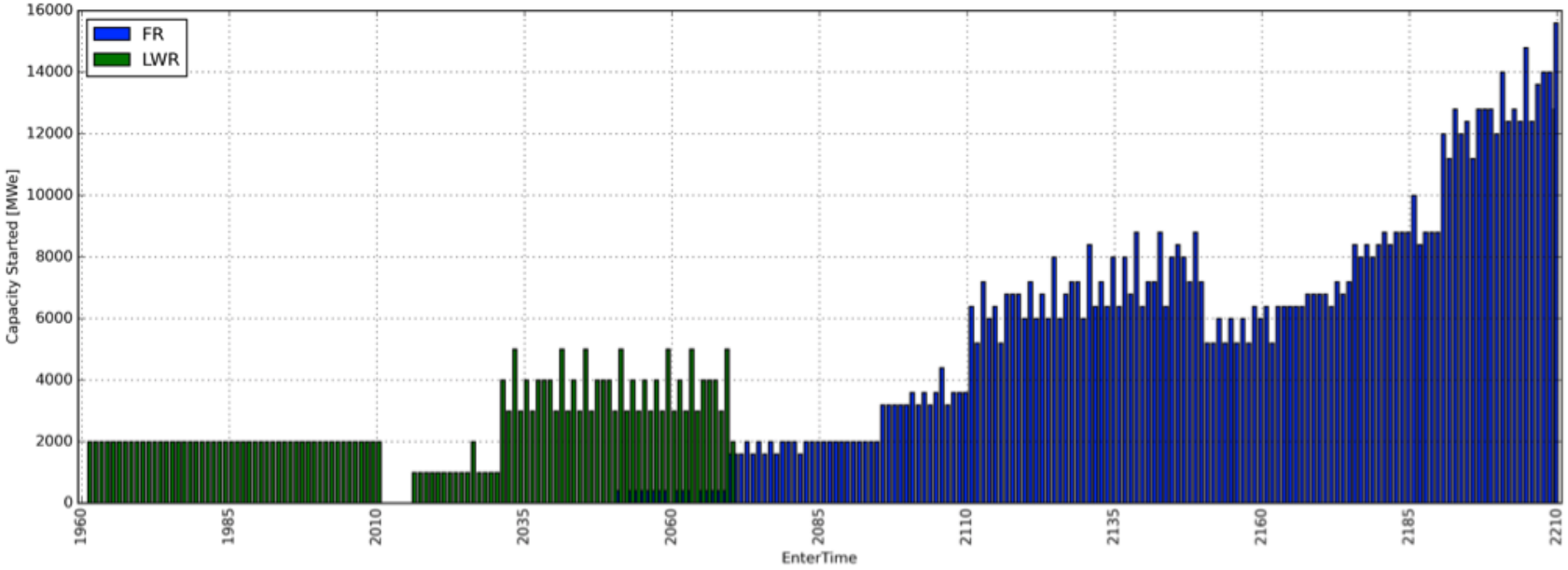
- LWR to SFR
- $T_0 = 2015$
- $T_f <= 2215$
- $C_0 = 100$ GWe LWR
- Annual nuclear energy demand growth: 1%
- Legacy LWRs have either 60-year lifetimes or 80-year lifetimes.
- Spent LWR fuel reprocessed to fabricate FR fuel
- Spent FR fuel reprocessed to fabricate FR fuel

Transition Analysis



Power generated by reactor type.

Transition Analysis



Capacity deployed each year, by reactor type.

Resources

Ok, I'm convinced. So how can one learn this stuff?

Online Resources

- Software Carpentry
- Version Control: Github, Pro Git book
- Testing: nose, goolgetest
- Documentation: Sphinx, Doxygen
- Effective Computation in Physics

Links

- katyhuff.github.io
- fuelcycle.org
- fhr.nuc.berkeley.edu
- pyrk.github.io

Acknowledgements

- Denia Djokic
- Matthew Gidden
- Massimiliano Fratoni
- Ehud Greenspan
- Per Peterson
- Anthony Scopatz
- Xin Wang
- Paul Wilson
- Jasmina Vujic
- and many more...

THE END

Katy Huff

katyhuff.github.io/2016-03-02-utk