Large-Scale Multi-dimensional Asynchronous Replica Exchange Simulations on XSEDE using the BigJob Protocol: a.k.a the “CDI Project”

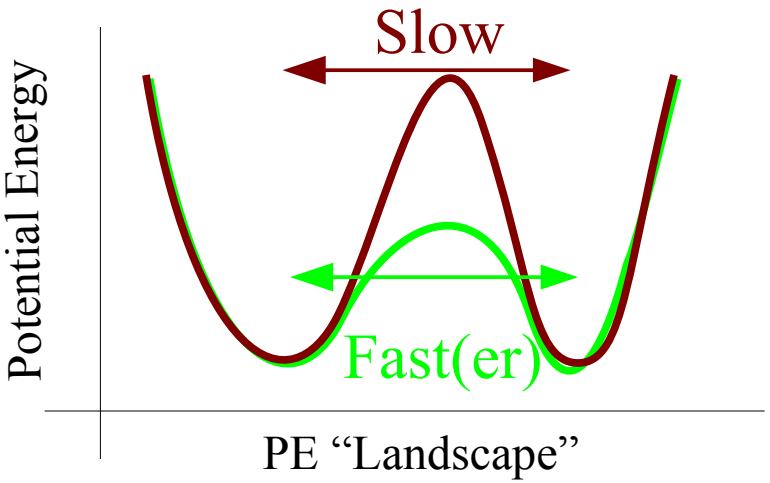
Emilio Gallicchio

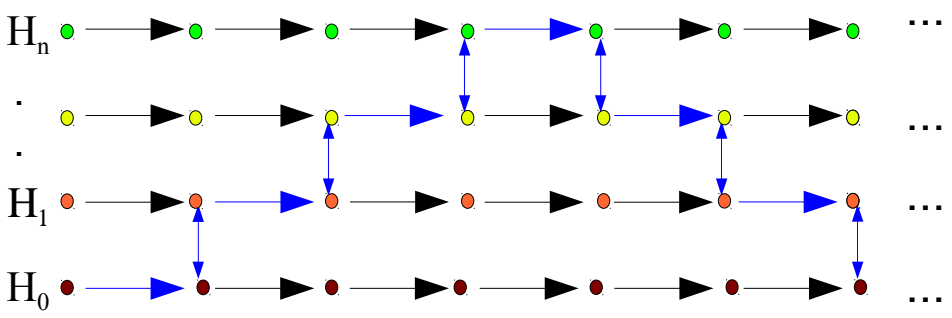
emilio@biomaps.rutgers.edu

Shantenu Jha, Ronald Levy, Darrin York, Melissa Romanus, Brian Radak, Tai-Sung Lee, Dai Wei, Peng He

Why Replica Exchange?

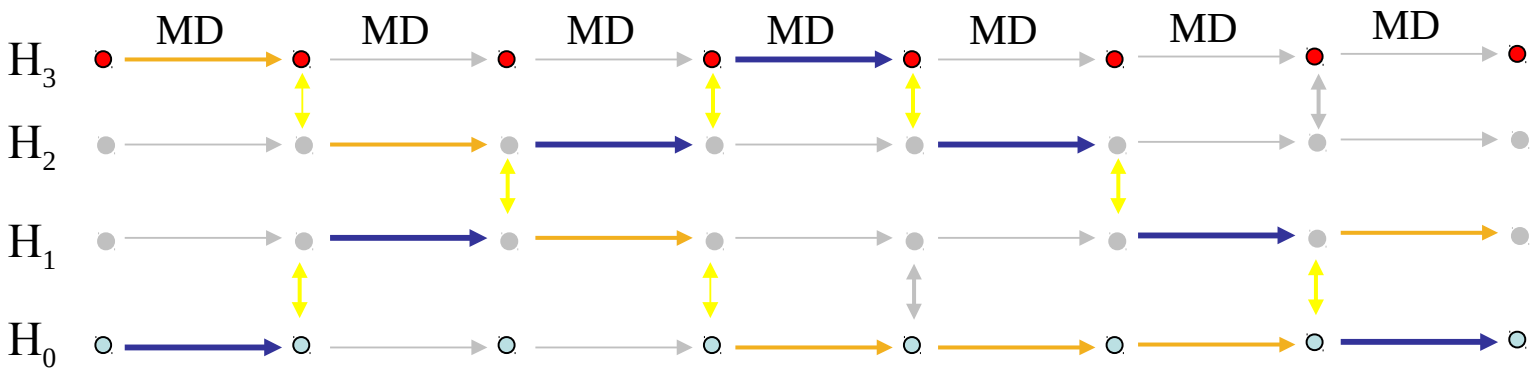
RE is based on the Hamiltonian-hopping idea:





- Thermodynamic equilibrium between Hamiltonian states must be maintained.
- Serial Implementations (serial tempering, etc.):
  - free energy weights are adjusted to regulate time spent at each state.
- Parallel Implementations (replica exchange):
  - runs as many MD replicas as states; Hamiltonian swaps between replicas
  - overall each state is equally sampled automatically
  - suitable for parallel environments

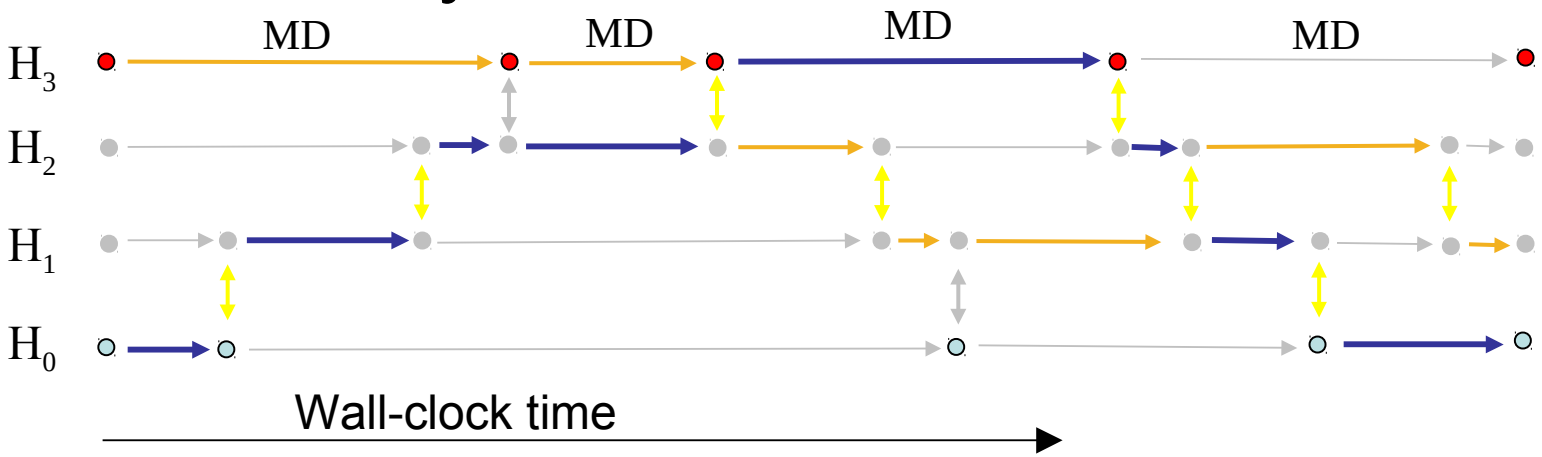
Synchronous REMD



Not scalable to many (1000's) replicas/states

- After a fixed number of MD steps all replicas stop computing and exchanges occur simultaneously.
- All replicas need to run at the same time
- Termination of one replica causes termination of the whole RE simulation.
- Not suitable for large RE calculations on limited resources
- Not suitable for pools of delocalized, heterogeneous, dynamic, unreliable, possibly at times isolated processors.
- Multi-dimensional RE schemes are difficult to implement within existing MD engines codebases.
- Overall speed is the speed of the slowest processor. CPU resources – often divided among a minority of fast processors and majority of slow processors - are not efficiently utilized.

Asynchronous REMD

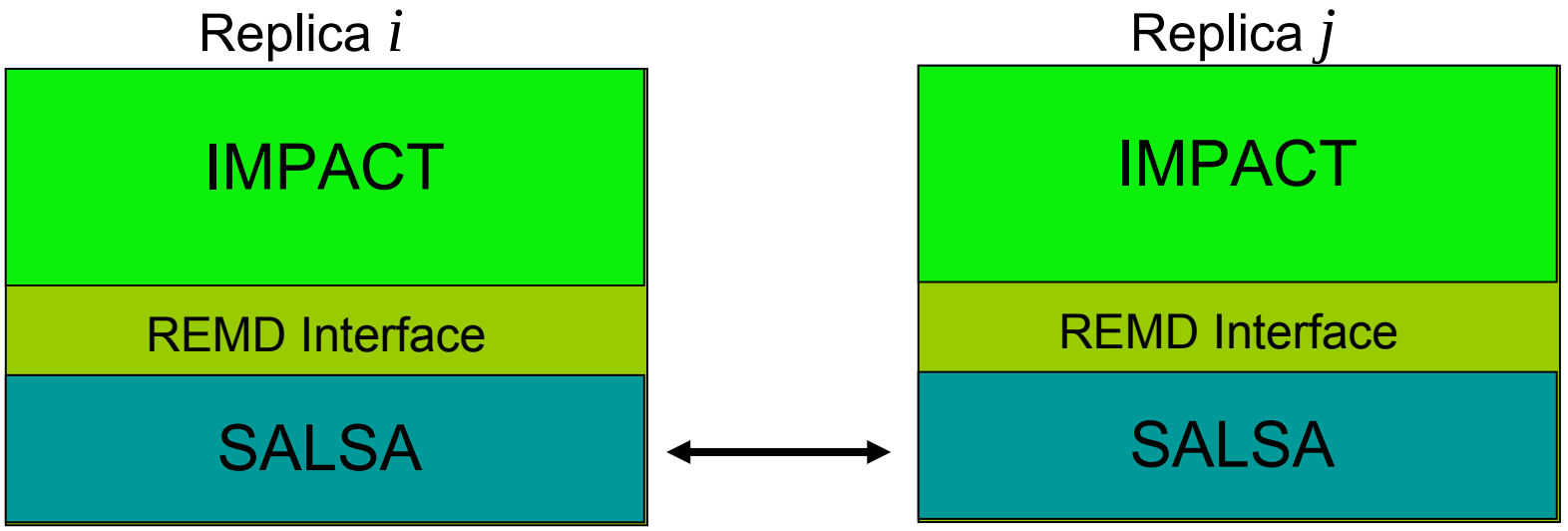


- Exchanges are performed at random times between random pairs of available replicas.
- Replicas run independently in between exchanges.
- Termination of one replica does not affect the others.
- Not all replicas need to be running at the same time.
- The number of running replicas can vary depending on available resources.
- Satisfies microscopic reversibility.
- Does not rely on centralized master gather/scatter mechanism.
- Each replica can complete a different number of MD steps in between swaps.
- Suitable for large RE applications on dynamic and heterogeneous computational resources.
- Fast and slow processors automatically operate at near 100% utilization. Overall speed scales as global processing capacity.

Network-Based Asynchronous REMD (ca. 2007)

With Manish Parashar

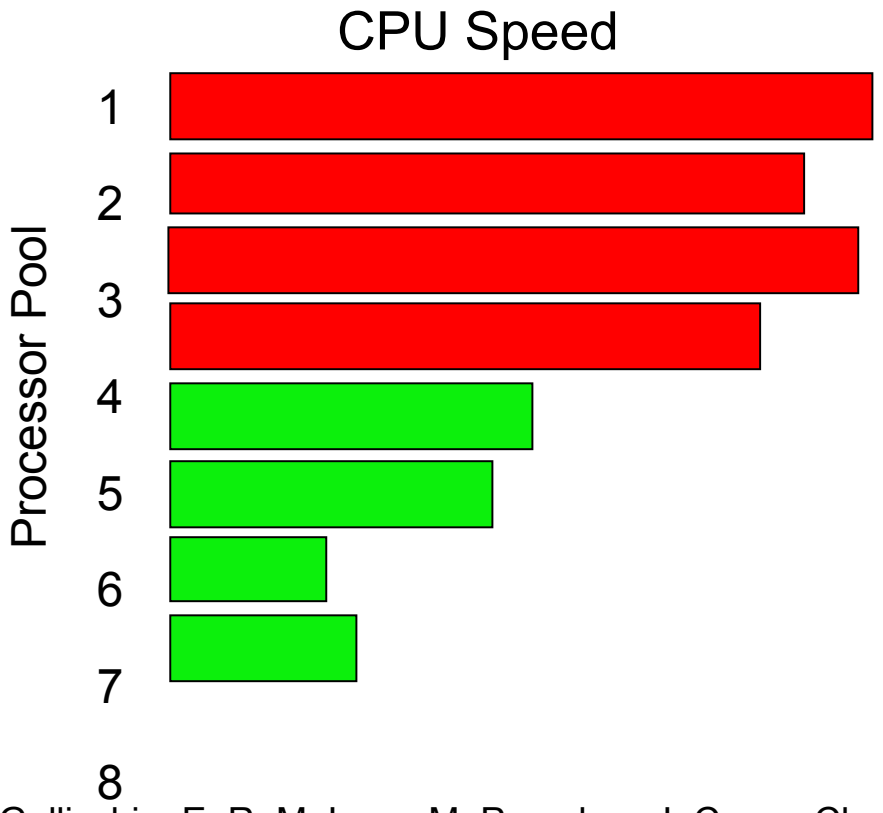
- Distributed “bulletin board” system.

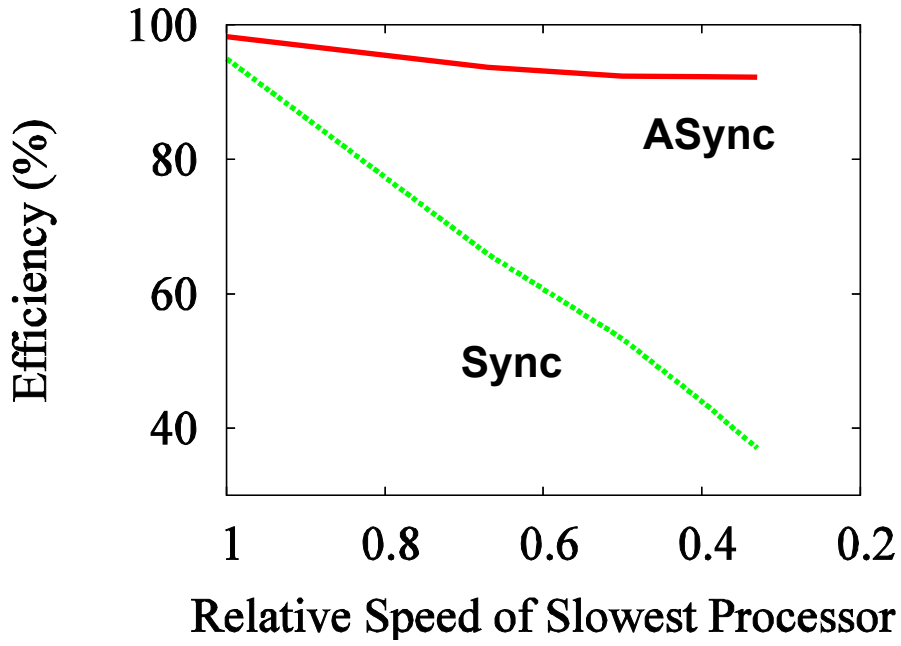


Gallicchio, E; R. M. Levy; M. Parashar; Asynchronous replica exchange for molecular simulations. J. Comp. Chem., 29, 788 (2008).

Network-Based Asynchronous REMD (ca. 2007)

Utilization Efficiency of a Heterogeneous Pool of Processors





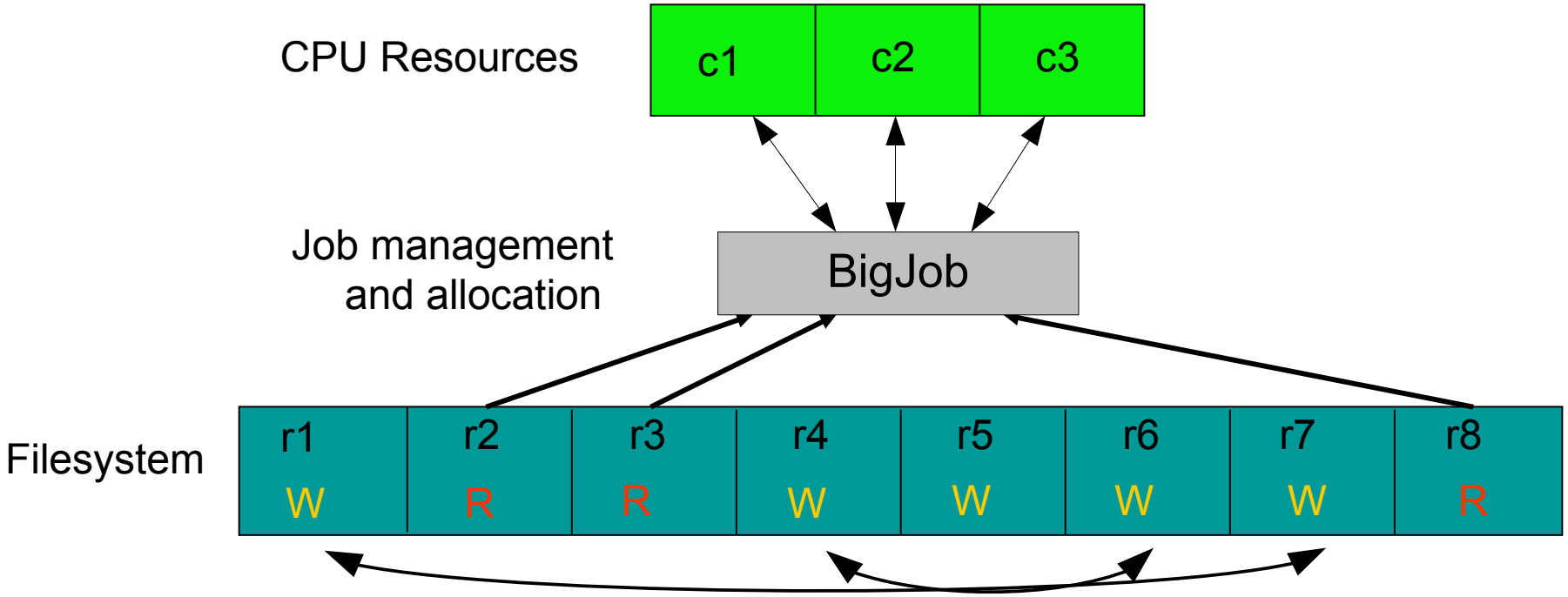
Gallicchio, E; R. M. Levy; M. Parashar; J. Comp. Chem., 29, 788 (2008).

CDI-Type II: Mapping Complex Biomolecular Reactions with Large Scale Replica Exchange Simulations on National Production Cyberinfrastructure

Levy, Jha, York, Lee, Gallicchio

Award: $1.6M, 4 years, start 2011

A File-Based Implementation of Asynchronous REMD using BigJob



- Exchanges of parameters are performed for pairs of replicas currently not running.
- Plus: Does not require code changes in the MD engine.
- Minus: Exchanges are possible only when replicas are waiting to run. Low exchange frequency.

AsyncRE: Python Utility for File-Based Asynchronous RE

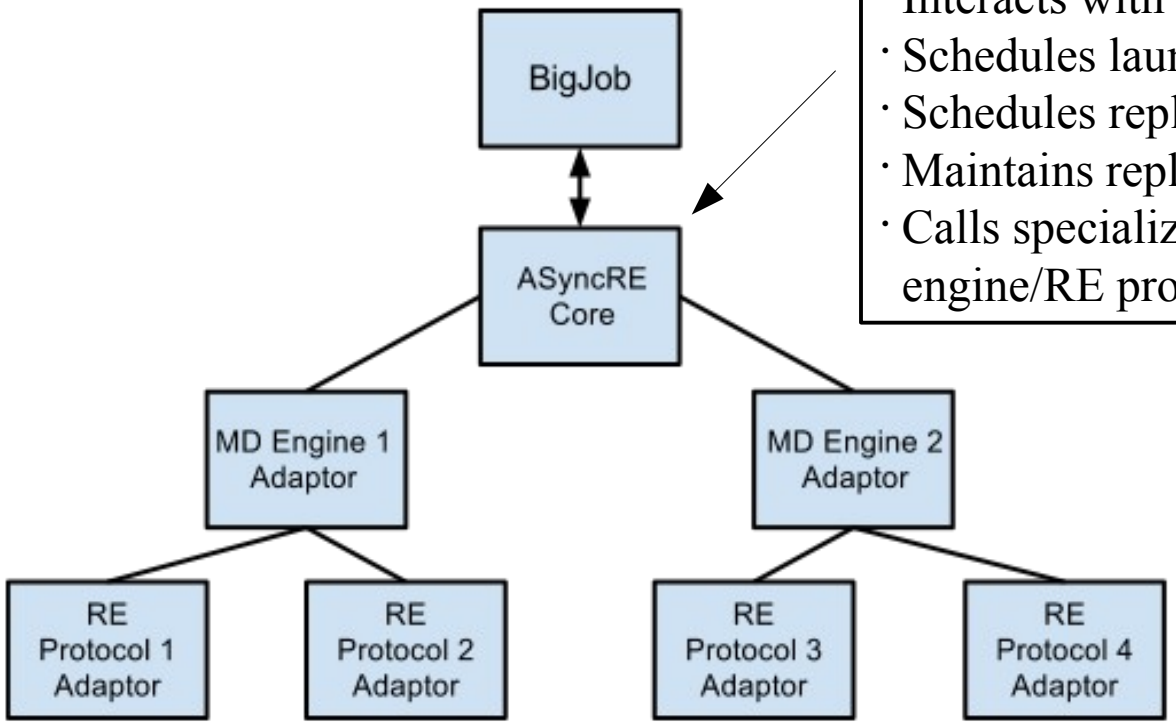
Main architecture:

- Replicas are prepared in individual directories (r1, …, rM)) starting from template input files, or similar.
- A subset of replicas is launched via BigJob. They enter running "R" state.
- When a replica completes a run it enters a "W" (wait) state.
- Exchanges of thermodynamic parameters are conducted by Gibbs sampling among waiting replicas
- Cycle is repeated.

- Detection of failed replicas: automatically resubmitted.
- Checkpointing and restart.
- Resilient execution (i.e. file system errors).
- Run and "forget"

AsyncRE Software Implementation

Modularization to support multiple RE modalities:

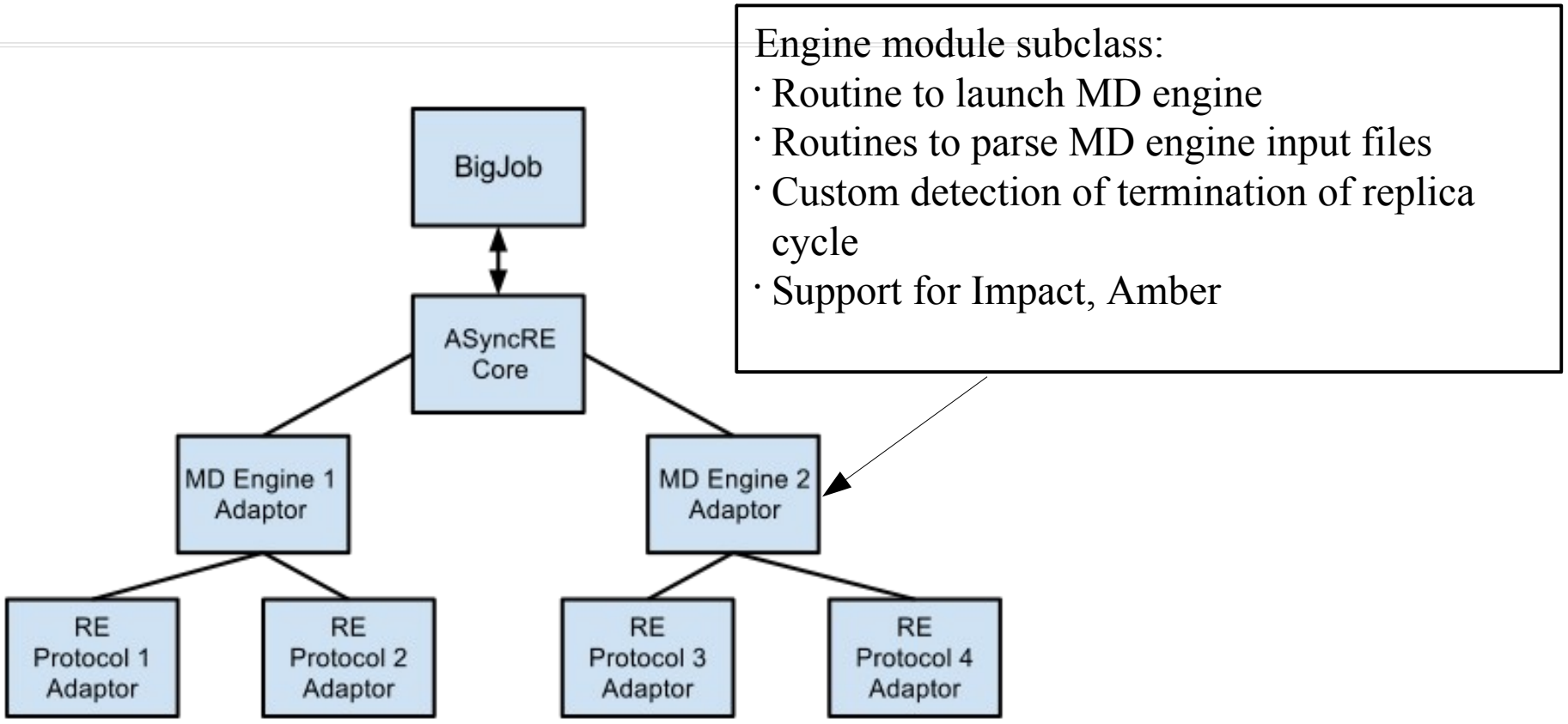


Core module class:

- Implements main logic of AsyncRE scheme
- Interacts with BigJob
- Schedules launching of replicas
- Schedules replica exchanges
- Maintains replica statuses (R, W, cycle #, …)
- Calls specialized routines based on MD engine/RE protocol combinations

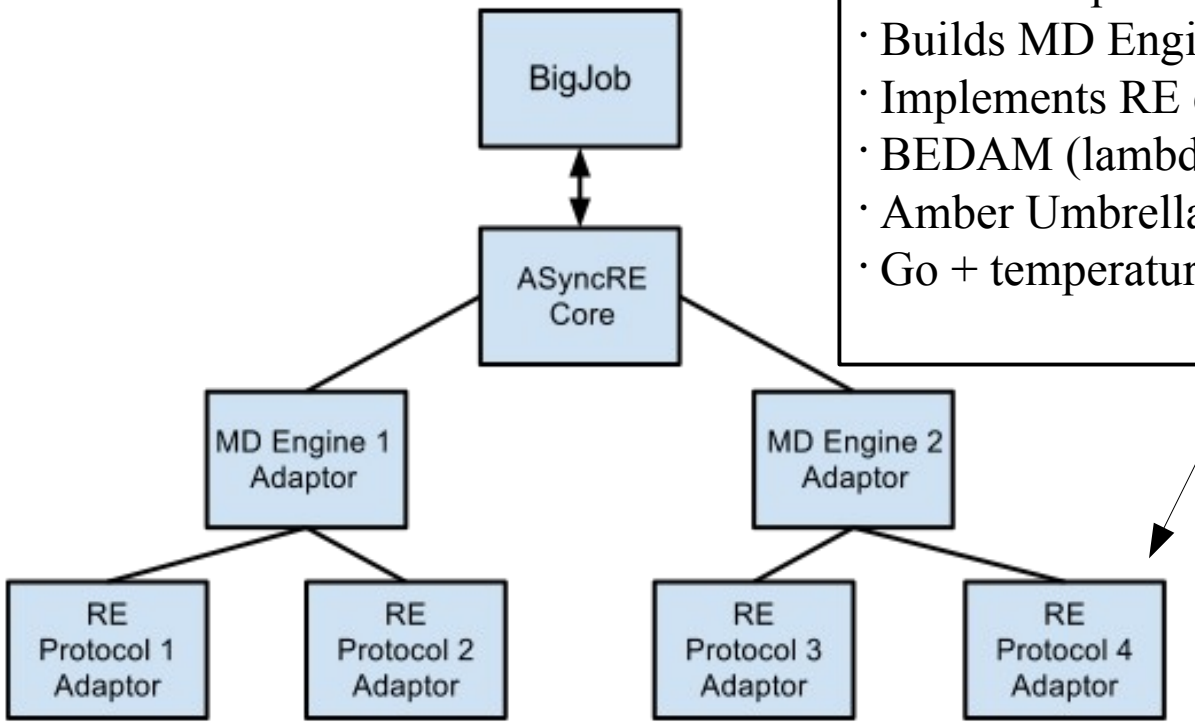
AsyncRE Software Implementation

Modularization to support multiple RE modalities:



AsyncRE Software Implementation

Modularization to support multiple RE modalities:



RE Application module subclass:

- Reads custom settings
- Builds RE parameters table
- Builds MD Engine/Application input files
- Implements RE exchange criterion
- BEDAM (lambda, lambda+temperature)
- Amber Umbrella sampling
- Go + temperature ...

Examples of Application-Level Custom Routines

Builds an Input file for a replica:

```
def _buildInpFile(self, replica):
    basename = self.basename
    stateid = self.status[replica]['stateid_current']
    cycle = self.status[replica]['cycle_current']

    template = "%s.inp" % basename
    inpfile = "r%d/%s_%d.inp" % (replica, basename, cycle)
    lambd = self.lambdas[stateid]
    # read template buffer
    tfile = open(template, "r")
    tbuffer = tfile.read()
    tfile.close()
    # make modifications
    tbuffer = tbuffer.replace("@n@",str(cycle))
    tbuffer = tbuffer.replace("@nm1@",str(cycle-1))
    tbuffer = tbuffer.replace("@lambda@",lambd)
    # write out
    ofile = open(inpfile, "w")
    ofile.write(tbuffer)
    ofile.close()
```

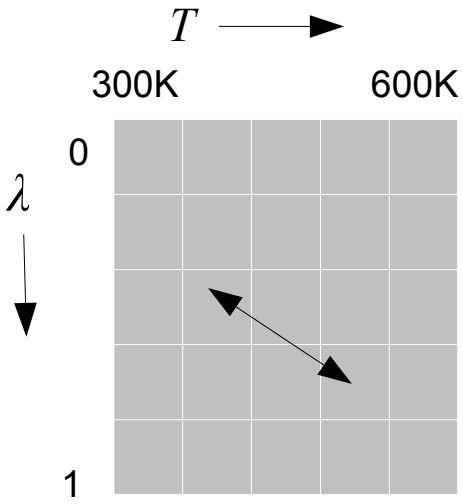
Examples of Application-Level Custom Routines

Launches a replica using BigJob:

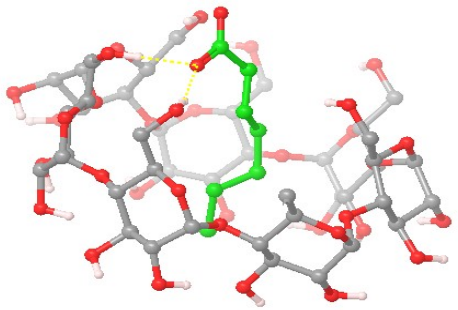
```
def _launchReplica(self,replica,cycle):

    input_file = "%s_%d.inp" % (self.basename, cycle)
    log_file = "%s_%d.log" % (self.basename, cycle)
    err_file = "%s_%d.err" % (self.basename, cycle)

    #Compute Unit Description
    compute_unit_description = {
       "executable": os.getcwd()+"/runimpact",
       "arguments": [input_file],
       "total_cpu_count": int(self.keywords.get('SUBJOB_CORES')),
       "output": log_file,
       "error": err_file,
       "working_directory":os.getcwd()+"/r"+str(replica),
       "spmd_variation":self.keywords.get('SPMD')
    }
    #Compute unit submission
    compute_unit =
        self.pilotcompute.submit_compute_unit(compute_unit_description)
    return compute_unit
```

Example of “Large-Scale” Application (192 replicas)

2D RE BEDAM Binding Free Energy Calculation in combined alchemical (λ) and temperature (T) space



- 24 λ states
- 8 temperatures
- 192 replicas in (λ,T) space



- β-cyclodextrin/heptanotate complex

- High temperature replicas to accelerate sampling at each alchemical window.

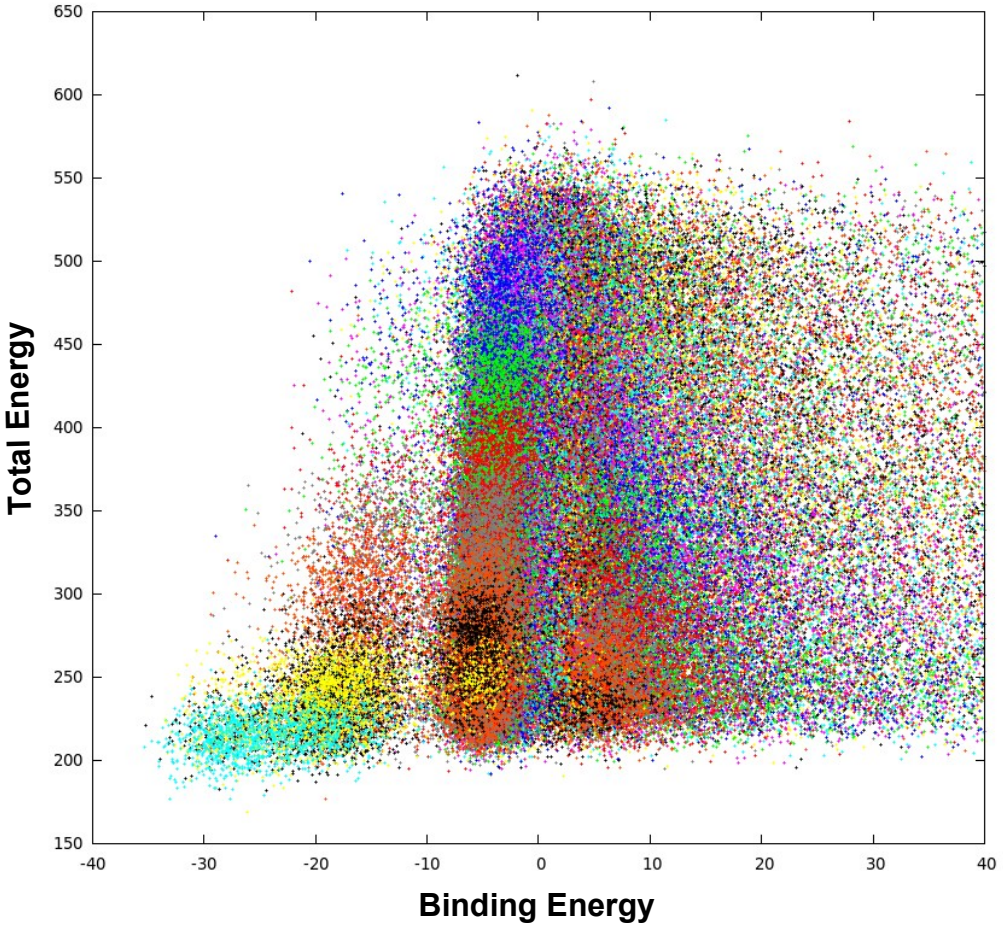
Example of “Large-Scale” Application (192 replicas)

Computational settings

- 192 replicas
- 4 processors per replica
- 384 CPU cores BigJob
- At any one time about 50% of the replicas are running and 50% are exchanging
- An exchange/launching round every 30 seconds
- At every round ~48 “Gibbs” swaps are performed.
- 1 day runtime on Trestles
- 800ns aggregated simulation time

Energy Distributions

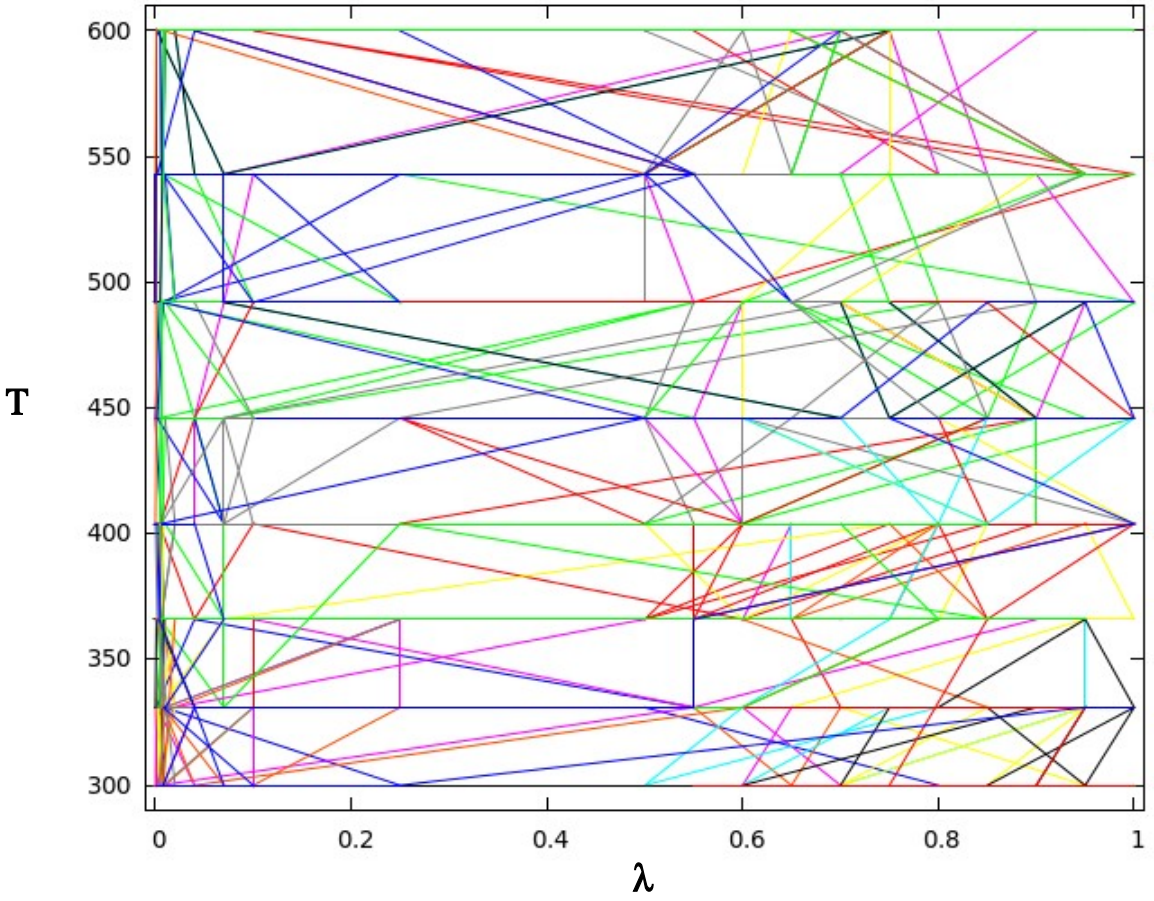
Total Energy/Binding Energy Distributions at the 192 (λ,T) replicas



240,000 data points to process through UWHAM to get binding free energy

Diffusion in Parameter Space

(λ,T) time trajectories for 10 of the 192 replicas



Interesting long-range simultaneous alchemical/temperature exchanges

Future Directions

Tools Development:

- BigJob optimizations
- Tests with ~10^3 replicas.
- Run on multiple XSEDE sites, file staging, etc.
- Data deluge, transfer, analysis, ...
- Adaptive RE: add/remove replicas during run
- Load on front-end host (exchanges can be somewhat expensive).
- Public release.

Science-related:

- Mapping free energy landscapes of complex enzymatic reactions.
- Binding and folding with 2D RE techniques.
- 2D Replica Exchange study of HIV-RT RNAseH dynamics

Acknowledgements

BigJob side:

- Shantenu Jha
- Melissa Romanus
- Andre Luckow
- Andre Merzky
- Yaakoub El Khamra
- Ole Weidner

Molecular modeling side:

- Ron Levy
- Peng He
- Wei Dai
- Darrin York
- Tai-Sung Lee
- Brian Radak