% Reproducible Parallel Simulations with Harvestr % % UseR! 2012

ro dev='pdf', cache=T, warning=T, error=T, out.width="\\textwidth" , fig.width=4, fig.height=3 or

\includegraphics[width=]{./Bill Gates lazy person.jpg)

# Reproducible Simulation

## Reproducibility

* Same script
* Same results
* Anywhere
  + Single thread
  + Multi-core
  + Cloud Scale

## Everything starts with a seed.

Simulation is based off Pseudo-random number generation (PRNG).

* PRNG is sequential, next number depends on the last state.
* Seeds are used to store the state of a random number generator
* by 'Setting a seed' one can place a PRNG into any exact state.

## Parallel Random Number Generation

Simulation is complicated in new parallel environments.

* PRNG is sequential,
* parallel execution is not,
* and order of execution is not guaranteed.

This is where parallel pseudo-random number generators help out.

## Parallel PRNG

Parallel pseudo-random number generators start with a singe state that can spawn additional streams as well as streams of random numbers.

1. SPRNG
2. L'Ecuyer combined multiple-recursive generator

# Introducing harvestr

## R package harvestr

<https://github.com/halpo/harvestr>

What harvestr does:

* Reproducibility
* Caching
* Under parallelized environments.

## How harvestr works

* Analytical elements are separated into work-flows of dependent elements.
  + Set up environment/seed
  + Generate Data
  + Perform analysis
    - Stochastic
    - Non-Stochastic
  + Summarize
* Results from one step carry to another by carrying the seed with the results.

## **Primary work-flow** for harvestr

* gather(n) - generate n random number streams.
* farm(seeds, expr) - evaluate expr with each seed in seeds.
* harvest(x, fun) - for each data in x call the function fun (based off plyrs llply).

## Example - Simple simulation

**Generate Data**

**Perform Analysis**

**Recombine**

## Stochastic Analysis in harvestr

* gather then farm as before.
* graft to generate seeds

## Example 2 - Stochastic Analysis

**graft to obtain independent RNG sub-streams**

## Example 3 Chained.

I'm really impatient and would like to do this in parallel.

## parallel

Just like plyr argument .parallel.

* uses [plyr](http://cran.r-project.org/package=plyr) and [foreach](http://cran.r-project.org/package=foreach) parallel structures.

## Caching

Results can be made fault tolerant or interruptible by including caching.

Caching in harvestr indexes on

* data
* function
* seed

using the digest function.

|  |
| --- |
| ## So is it really reproducible? |
| # Miscellaneous Extras |
| ## Building blocks Some building blocks that might *might* be helpful. |
| \* plant- for setting up copies of an object with given seeds. \* sprout - for obtaining the sub-streams used with graft. \* reap - single object version of harvest |
| ## In case you are wondering |
| \* Yes it works with Rcpp code, + provided the compiled code uses the RNGScope for RNG in C++. \* **But** take care to not carry C++ reference objects across parallel calls. |