IHME

Massive Simulations In Spark: Distributed Monte Carlo For Global Health Forecasts

Kyle Foreman, PhD

07 June 2016

UNIVERSITY of WASHINGTON

Institute for Health Metrics and Evaluation

Simulations in Spark

- **Context**
- Motivation
- SimBuilder
- Backends
- Benchmarks
- Discussion

What is IHME?

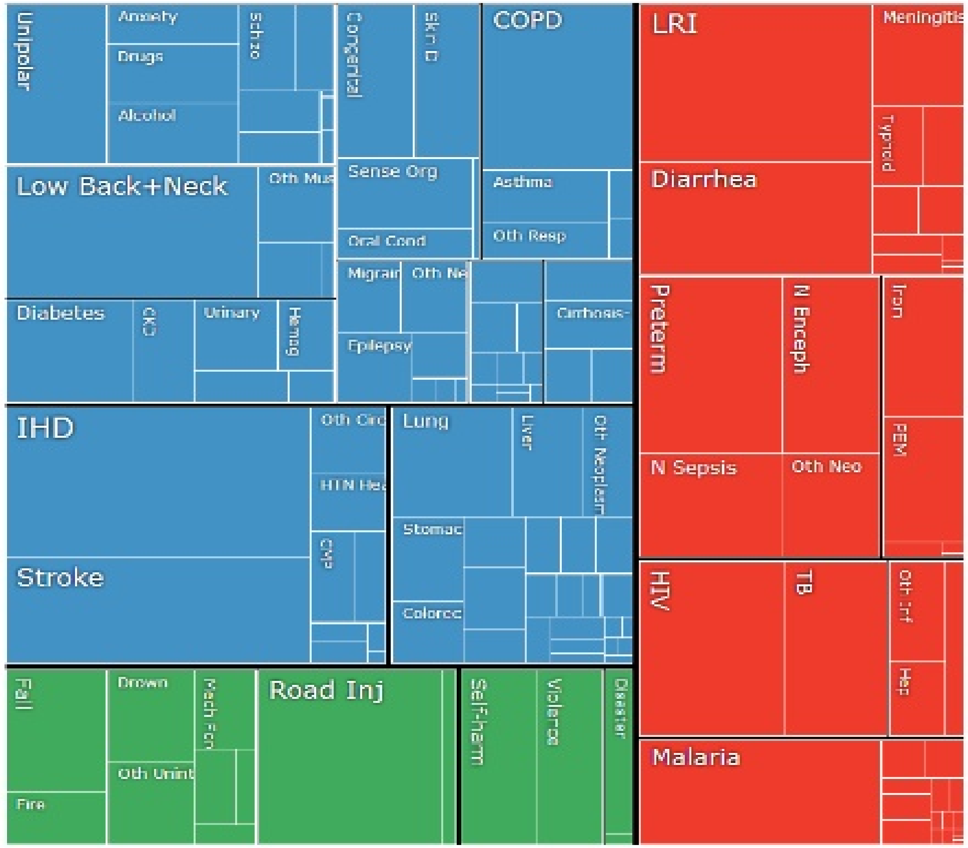
- Independent research center at the University of Washington
- Core funding by Bill & Melinda Gates Foundation and State of Washington
- ~300 faculty, researchers, students and staff
- Providing rigorous, scientific measurement
  - What are the world's major health problems?
  - How well is society addressing these problems?
  - How should we best dedicate resources to improving health?

Goal:
improve health
by providing the best
information
on population health

IHME

Global Burden of Disease

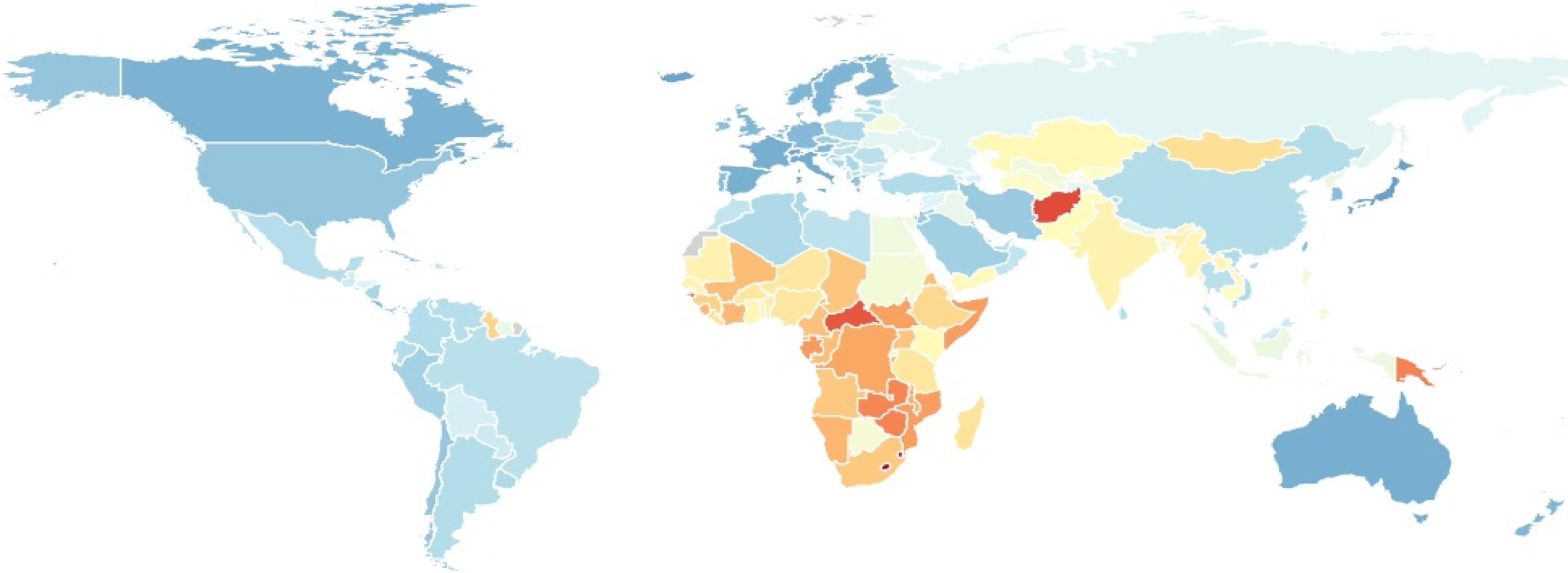
- A systematic scientific effort to quantify the comparative magnitude of health loss due to diseases, injuries and risk factors
  - 188 countries
  - 300+ diseases
  - 1990 – 2015+
  - 50+ risk factors



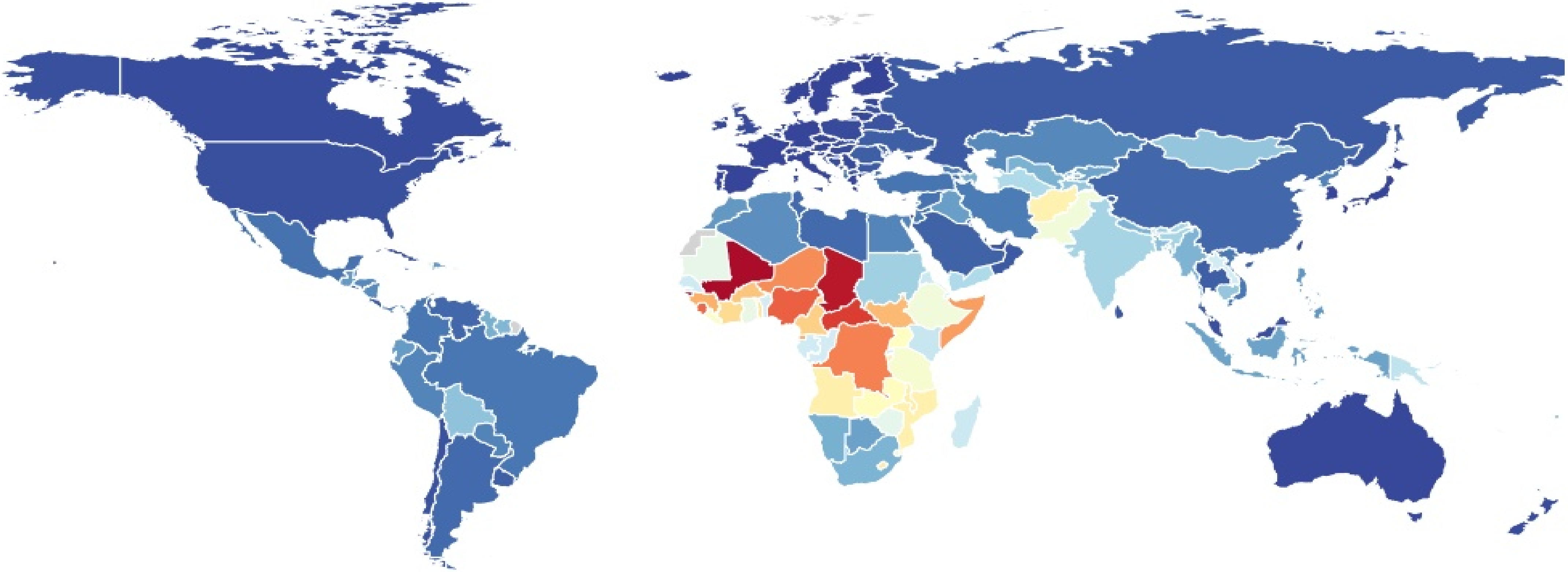
GBD Collaborative Model

1,100 experts in 106 countries

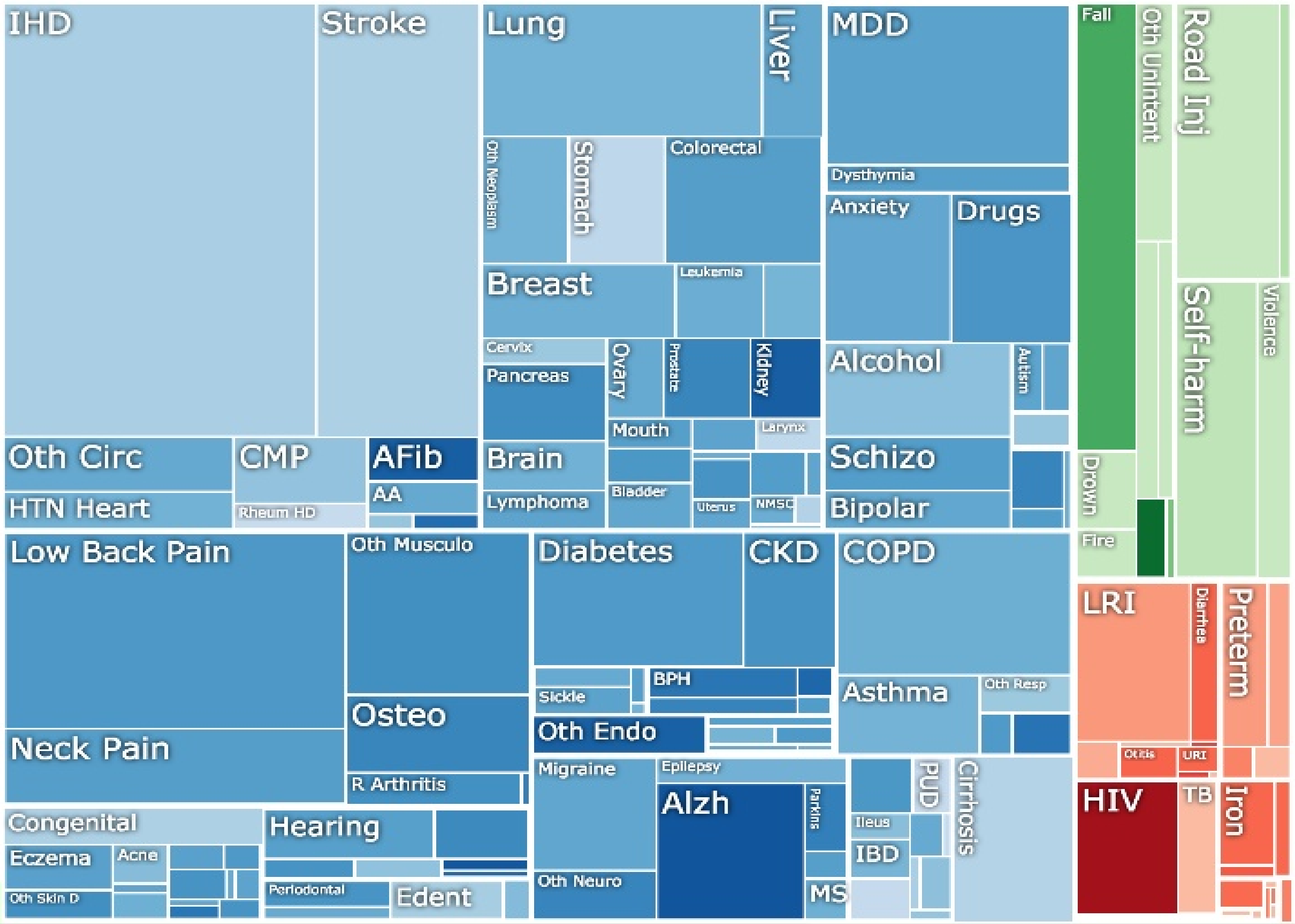
Death Rate in 2013 (age-standardized)



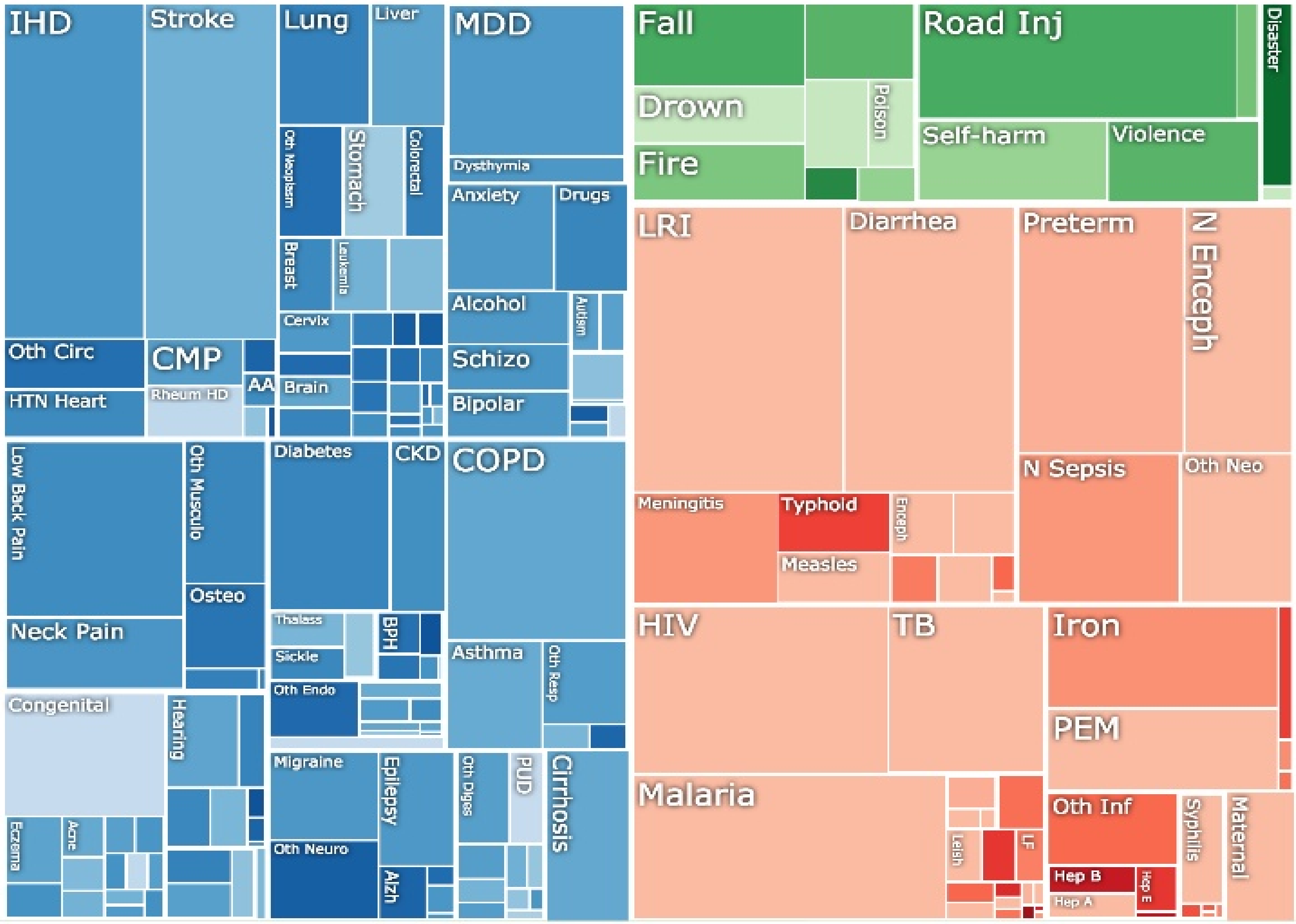
Childhood Death Rate in 2013



DALYs in High Income Countries (2010)



DALYs in Low & Middle Income Countries (2010)



Simulations in Spark

- Context
- **Motivation**
- SimBuilder
- Backends
- Benchmarks
- Discussion

Forecasting the Burden of Disease

1. Generate a baseline scenario projecting the GBD 25 years into the future
   - Mortality, morbidity, population, and risk factors
   - Every country, cause, sex, age
2. Create a comprehensive simulation framework to assess alternative scenarios
   - Capture the complex interrelationships between risk factors, interventions, and diseases to explore the effects of changes to the system
3. Build a modular and flexible platform that can incorporate new models to answer detailed "what if?" questions
   - E.g. introduction of a new vaccine, effects of global warming, scale up of ART coverage, risk of global pandemics, etc

Simulations

- Takes into account interdependencies between risk factors, diseases, etc.
  - Use draws of model parameters to advance from t to t+1, allowing for forecasts of other quantities (e.g. risk factors and mortality) to interact
- Modular structure allows for detailed "what ifs"
  - E.g. scale up of coverage of a new intervention
- Allows us to incorporate alternative models to test sensitivity to model choice and specification

Basic Simulation Process

1. Fit statistical models (`PyMB`) capturing most important relationships as statistical parameters
2. Generate many draws from those parameters, reflecting their uncertainty and correlation structure
3. Run Monte Carlo simulations to update each quantity over time, taking into account its dependencies

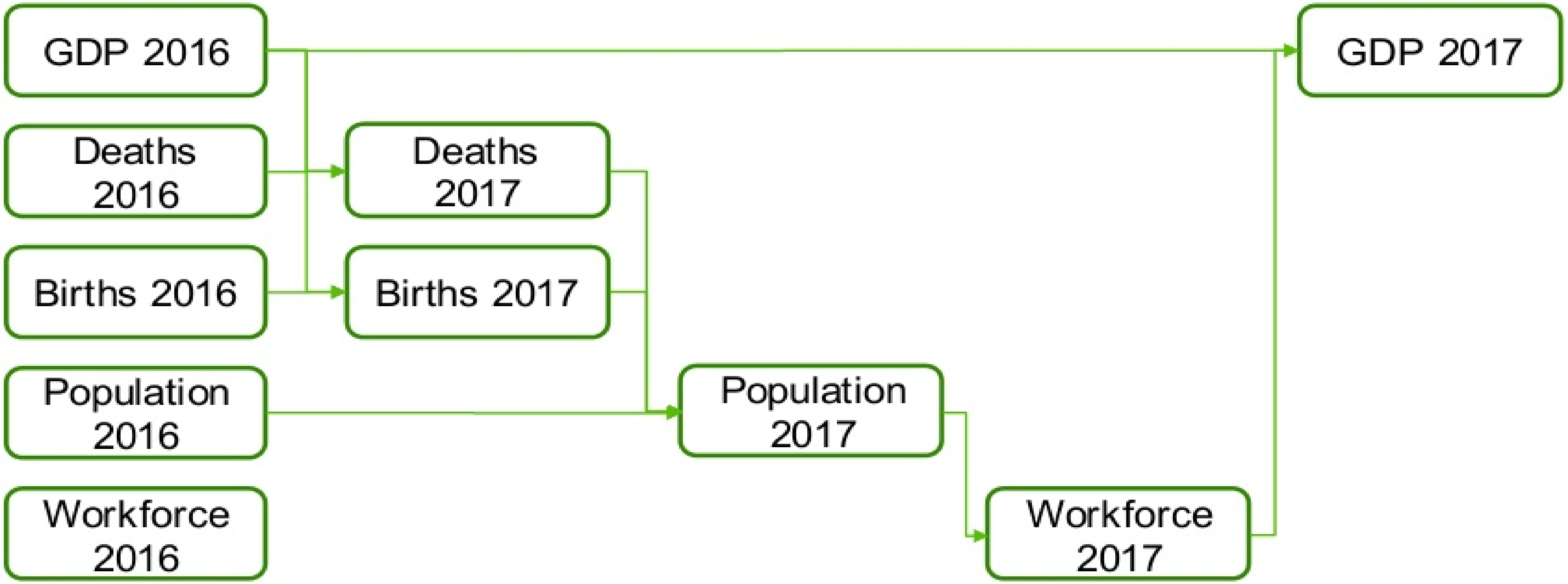
Simulations in Spark

- Context
- Motivation
- **SimBuilder**
- Backends
- Benchmarks
- Discussion

SimBuilder

- Directed Acyclic Graph (DAG) construction and validation
  - `yaml` and `sympy` specification
  - `curses` interface
  - `graphviz` visualization
- Flexible execution backends
  - `pandas`
  - `pyspark`

Example Global Health DAG



Creating a DAG with SimBuilder

1. Create a YAML for the overall simulation
   - Specify child models and the dimensions of the simulation
2. Build separate YAML files for each child model
   - Dimensions along which the model varies
   - Location of historical data
   - Expression for how to generate the quantity in t+1
3. Run SimBuilder to validate and explore DAG

Creating a DAG with SimBuilder

sim.yaml

```yaml
name: gdp_pop
models:
  - name: gdp_per_capita
  - name: workforce
  - name: mortality
  - name: gdp
  - name: pop
global_parameters:
  - name: loc
    set: List
    values: ["USA", "MEX", "CAN", ..., ]
  - name: t
    set: Range
    start: 2014
    end: 2040
  - name: draw
    set: Range
    start: 0
    end: 999
```

Creating a DAG with SimBuilder

pop.yaml


```yaml
name: pop
version: 0.1
variables: [loc,t]
history:
    - type: csv
      path: pop.csv
```

Creating a DAG with SimBuilder

gdp_per_capita.yaml

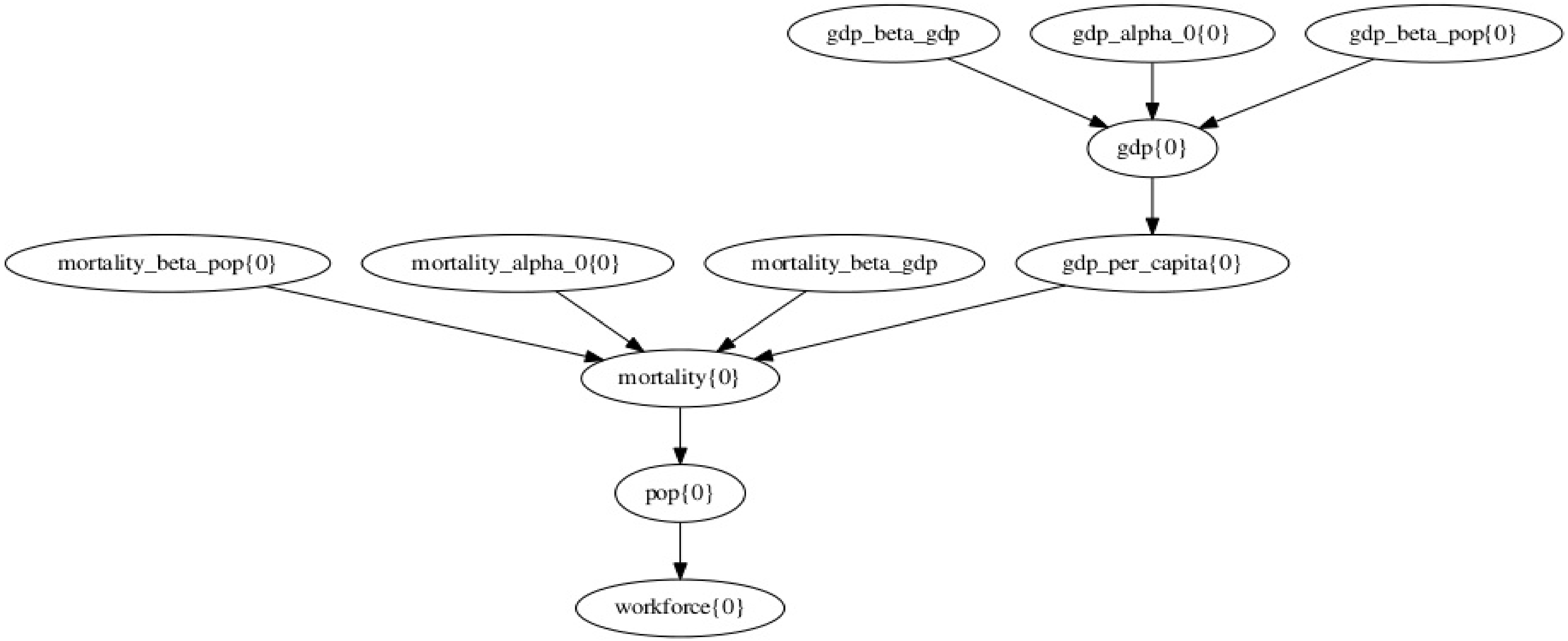
```yaml
name: gdp_per_capita
version: 0.1
expr: gdp(draw, t, loc)/pop(draw, t-1, loc)
variables: [draw, t, loc]
history:
  - type: csv
    path: gdp_pc_w_draw.csv
```

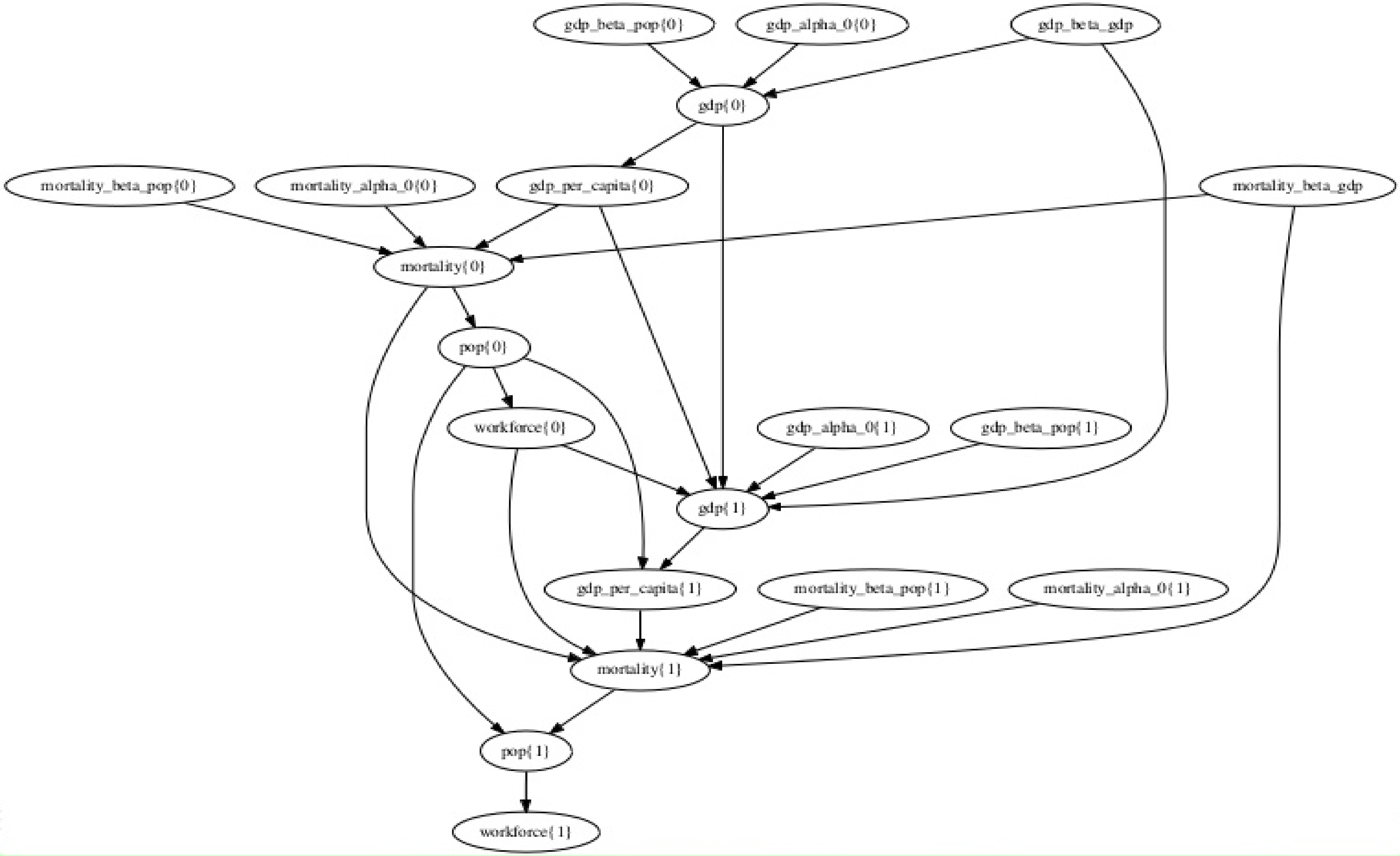
Creating a DAG with SimBuilder

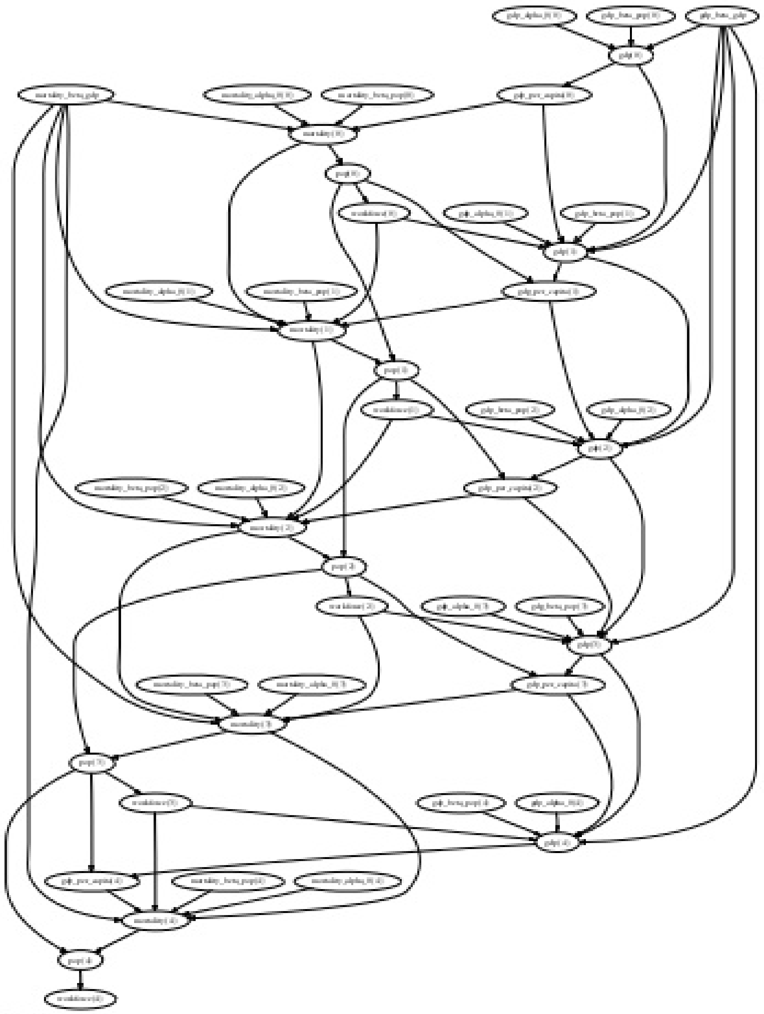
gdp.yaml

```yaml
name: gdp
version: 0.1
expr: gdp(d,loc,t-1) * exp(
                    alpha_0(loc, t, draw) +
                    beta_gdp(draw) * log(gdp_per_capita(draw, loc, t-1)) +
                    beta_pop(draw, t) * workforce(loc, t)
                    )
variables: [draw ,loc,t]
history:
  - type: csv
    path: gdp_w_draw.csv
model_parameters:
  - name: alpha_0
    variables: [draw, loc, t]
    history:
      - type: csv
        path: gdp/alpha_0.csv
  - name: beta_gdp
    variables: [d]
    history:
      - type: csv
        path: gdp/beta_gdp.csv
  - name: beta_pop
    variables: [d]
    history:
      - type: csv
        path: gdp/beta_pop.csv
```

DAG


gdp_beta_gdp
gdp_alpha_0{0}
gdp_beta_pop{0}
gdp{0}
mortality_beta_pop{0}
mortality_alpha_0{0}
mortality_beta_gdp
gdp_per_capita{0}
mortality{0}
pop{0}
workforce{0}


gdp_beta_pop{0}
gdp_alpha_0{0}
gdp_beta_gdp
gdp{0}
mortality_beta_pop{0}
mortality_alpha_0{0}
gdp_per_capita{0}
mortality_beta_gdp
mortality{0}
pop{0}
workforce{0}
gdp_alpha_0{1}
gdp_beta_pop{1}
gdp{1}
gdp_per_capita{1}
mortality_beta_pop{1}
mortality_alpha_0{1}
mortality{1}
pop{1}
workforce{1}



Simulations in Spark

- Context
- Motivation
- SimBuilder
- **Backends**
- Benchmarks
- Discussion

SimBuilder Backends

- SimBuilder traverses the DAG
  - Backends plugin via API
- Backends provide
  - Data loading strategy
  - Methods to combine data from different nodes to calculate new nodes
  - Method to save computed data

Local Pandas Backend

- Useful for debugging and small simulations
- Master process maintains list of Model objects:
  - `pandas` DataFrames of all model data
    - Indexed by global variables
  - `sympy` expression for how to calculate t+1
    - Vectorized when possible, fallback to row-wise apply
- Simulating over time traverses DAG to join necessary DataFrames and compute results to fill in new columns

Spark DataFrame Backend

- Similar to local backend, swapping in Spark's DataFrame for Pandas'
- Spark context maintains list of Model objects:
  - `pyspark` DataFrames of all model data
    - Columns for each global variable and
  - `sympy` expression for how to calculate t+1
    - Calculated using row-wise apply
- Joins DataFrames where necessary

Spark + Numpy Backend

- Uses Spark to distribute and schedule the execution of vectorized `numpy` operations
- One RDD for each node and year:
  - Containing an ndarray and index vector for each axis
  - Can optionally partition the data array into key-value pairs along one or more axes (similar to `bolt`)
- To calculate a new node, all the parent RDD's are unioned and then reduced into a new child RDD

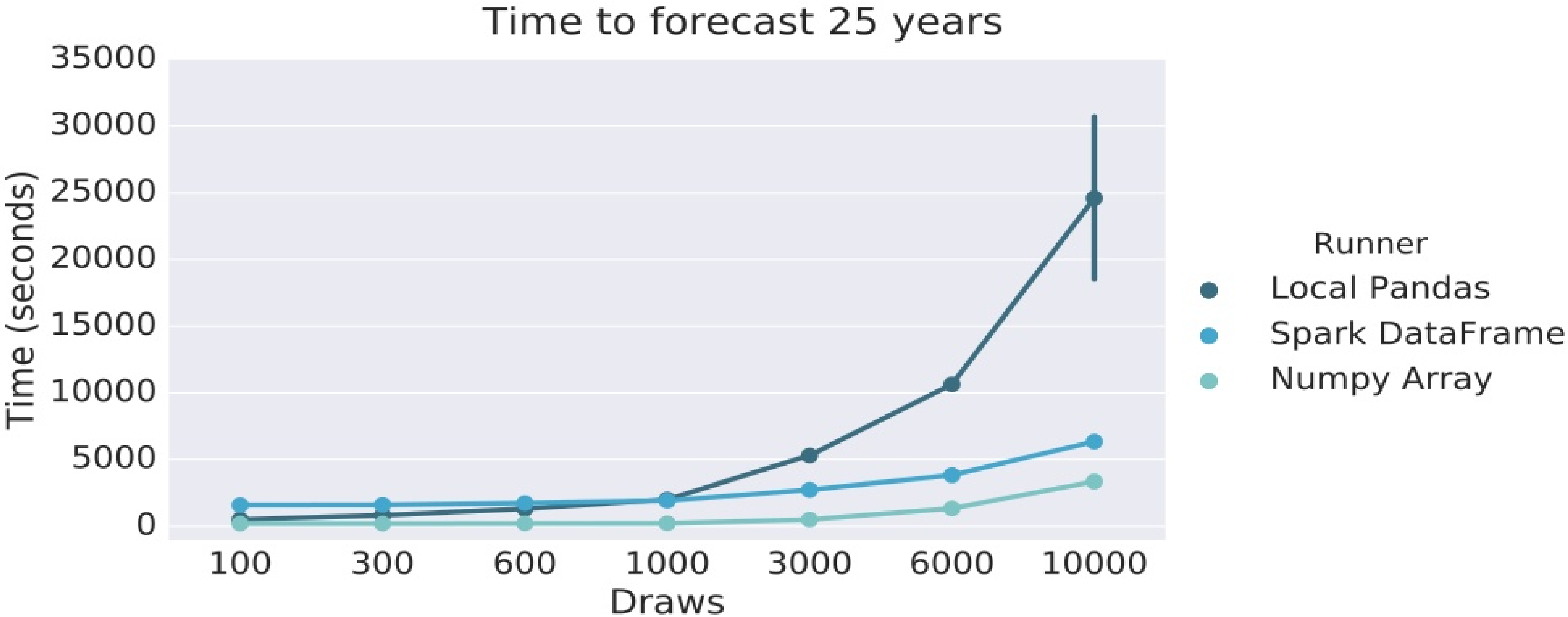
Simulations in Spark

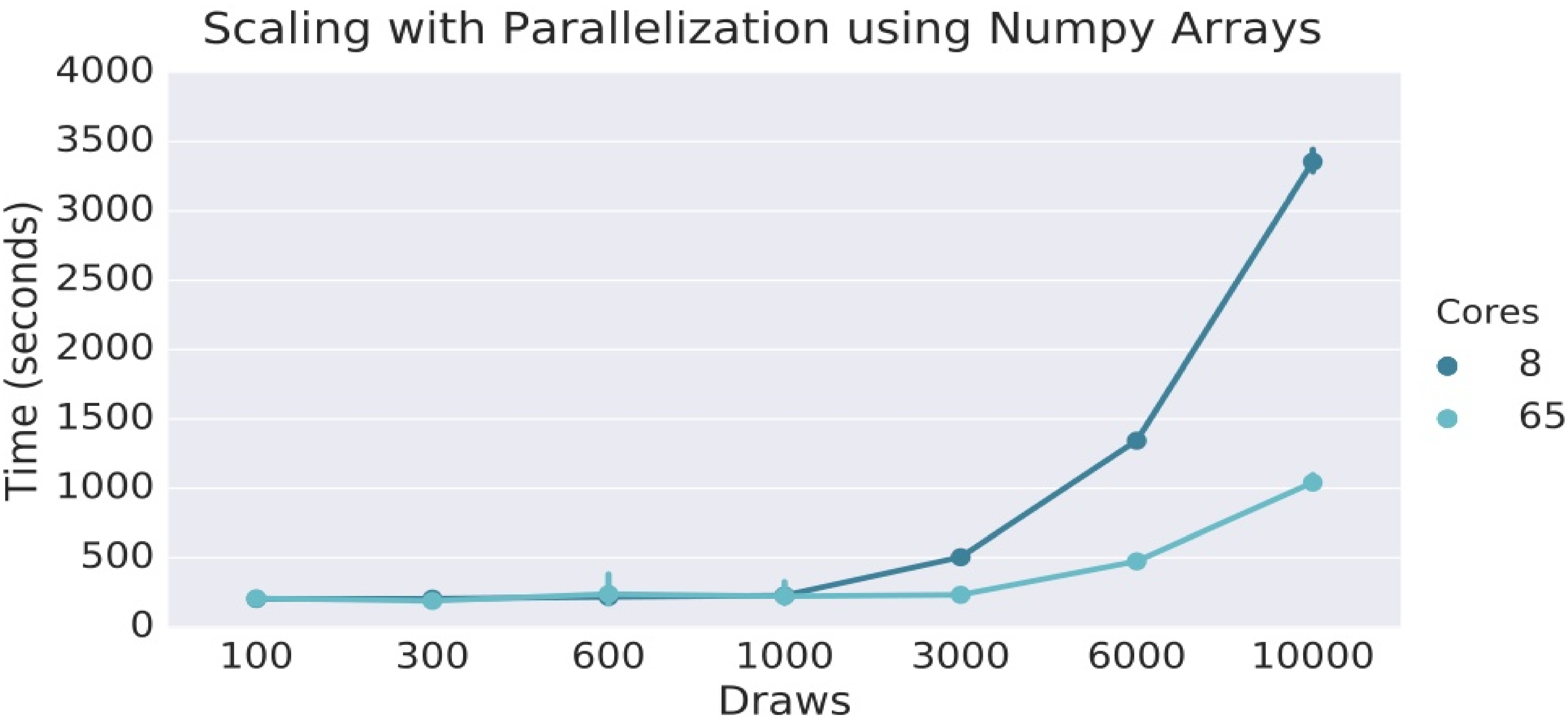
- Context
- Motivation
- SimBuilder
- Backends
- **Benchmarks**
- Discussion

Benchmarking

- Single executor (8 cores, 32GB)
- Synthetic data
  - 65 nodes in DAG
  - 3 countries x 20 ages x 2 sexes x N draws
  - Forecasted forward 25 years

Benchmarks


Time to forecast 25 years
35000
30000
25000
20000
15000
10000
5000
0
Time (seconds)
100
300
600
1000
3000
6000
10000
Draws
Runner
Local Pandas
Spark DataFrame
Numpy Array


Scaling with Parallelization using Numpy Arrays
Time (seconds)
4000
3500
3000
2500
2000
1500
1000
500
0
100
300
600
1000
3000
6000
10000
Draws
Cores
8
65

Simulations in Spark

- Context
- Motivation
- SimBuilder
- Backends
- Benchmarks
- **Discussion**

Limitations of Spark DataFrame

- Majority of execution time is spent joining the DataFrame and aligning the dimension columns
- These times were achieved after careful tuning of partitions
  - Perhaps custom partitioners could reduce runtime further

Taking Advantage of Numpy

- For multidimensional datasets of consistent shape and size, simply aligning indices can eliminate the overhead of joins
  - Including generalizations to axes of size 1 to simplify coding
- Numpy's vectorized operations are highly tuned and scale well
  - Including multithreading when e.g. compiled against `MKL`

Future Development

- Experiment with better partitioning of Numpy RDDs
  - Perhaps use `bolt`?
- Improve partial graph execution
  - Executing the entire DAG at once often crashes the driver
  - Executing each year's partial DAG in sequence results in idle CPU time towards the end of that DAG
- Investigate more efficient Spark DataFrame joining methods for Panel data

Team

Kyle Heuton
krheuton@uw.edu

Chun-Wei Yuan
cwyuan@uw.edu

Kyle Foreman
kfor@uw.edu

healthdata.org