Large-scale and Dynamic Simulations

What if I have 10.000 entities to simulate?

- Sometimes simulation scenarios include many entities
  - 200 overlapping WiFi nodes
  - 150.000 lemmings dropping off a cliff
- Not a good strategy to hardcode this into the simulation
- Need to come up with scalable methods

An example

- Randomly placing 200 buckets in a field
- under x % of buckets you put an easter egg
- You distribute 10 kids in the field and they turn over one bucket at a time, moving to the closest one after each try
- How long before all eggs are found?

Some new components here

- First, you need to be able to describe the algorithm for how a kid operates (complex kid-object)
- Need to keep track of movement and initial coordinates
- Need to generate buckets and kids according to distributions
- Need to have a config file to feed simulation parameters for auto-generation of scenario and simulate many different scenarios.

Object class bucket

- Bucket(x,y,egg)
- int location = (x,y);
- boolean egg;

- Insert_into_matrix_of_buckets();

Object class kid

- Kid(x,y, speed)
- int location = (x,y);

- While(){
- Find_next_bucket(x,y);
- Move_to_next_bucket();
- this.x,y = x,y;
- if (bucket.egg), global egg—;
- }

Config file

Param

buckets = 200

kids = 10

eggs = 15 (%)

speed = 2 (m/s)

field = 200, 200 (size of field in m)

Starting the simulation

- Need to use the config parameters in simulator

```
Open (File);
Read (parameters);
For (i = 0: i < buckets: i++){
  x,y = random(size_x, size_y);
  if (random() < prob_egg) egg_present = true;
  new Bucket bucket =(x,y, egg_present);
  bucket.Insert_into_matrix_of_buckets();
  }
```

initiating kids

```
For (i = 0: i < kids: i++){
  x,y = random(size_x, size_y);
  new Kid kid =(x,y,speed);
  }
```

Alternative way

- Config file contains all objects to insert in simulation, external generation
- Write script in favourite language and generate file with all objects to read in by simulator

```
New File file = open(config_file);
For (i = 0: i < buckets: i++){
  x,y = random(size_x, size_y);
  if (random() < prob_egg) egg_present = true;
  writeToFile(type = Bucket\nl X,Y = x,y\nl, egg =
  +egg_present.toString()
  }
```

same with kids objects

config file 2 example

- type = bucket
  X,Y = 13,46
  egg = 0

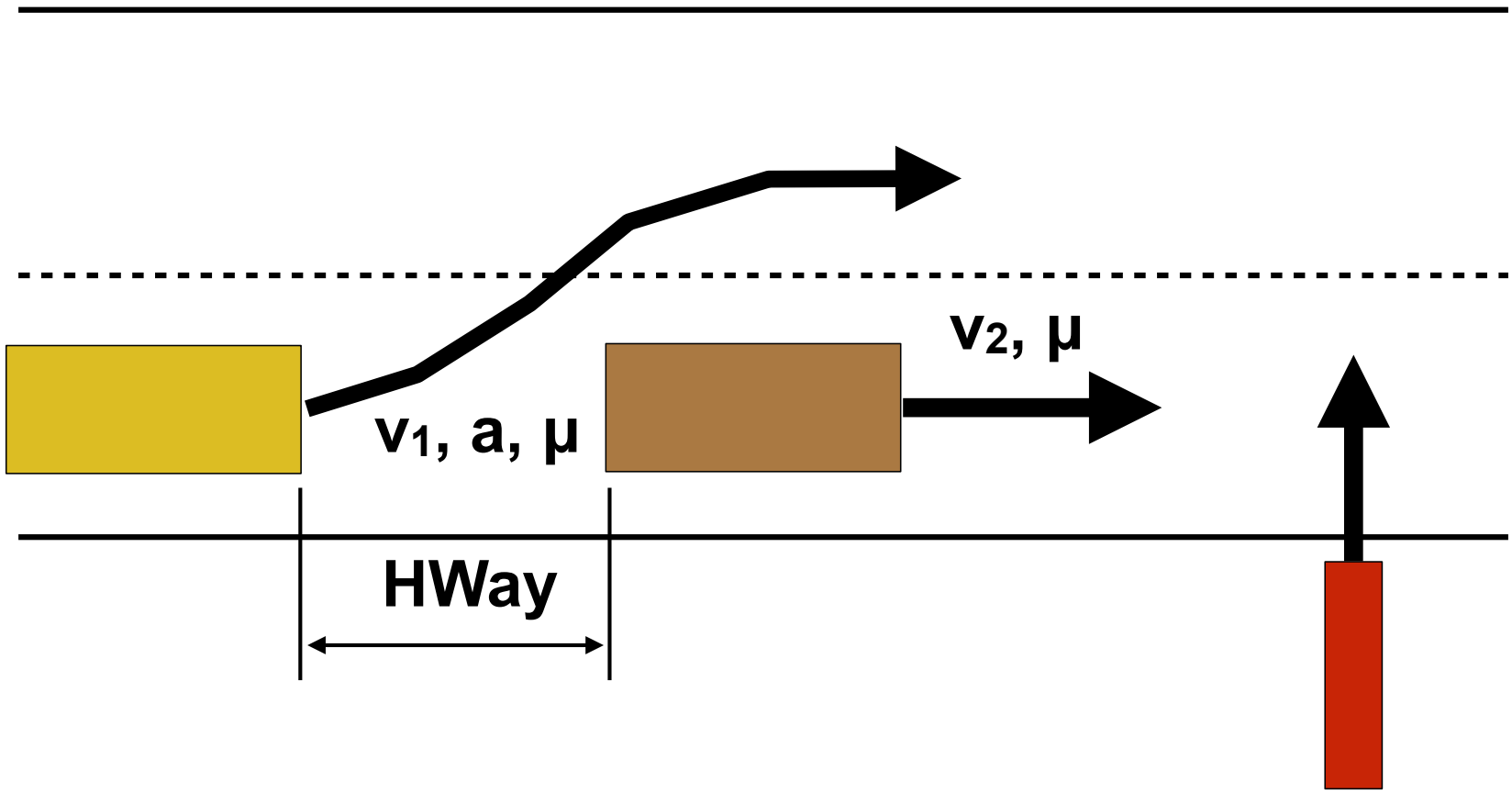
- type = bucket
  X,Y = 77,41
  egg = 1

- type = kid
  X,Y = 91,1
  speed = 2

A dynamic example

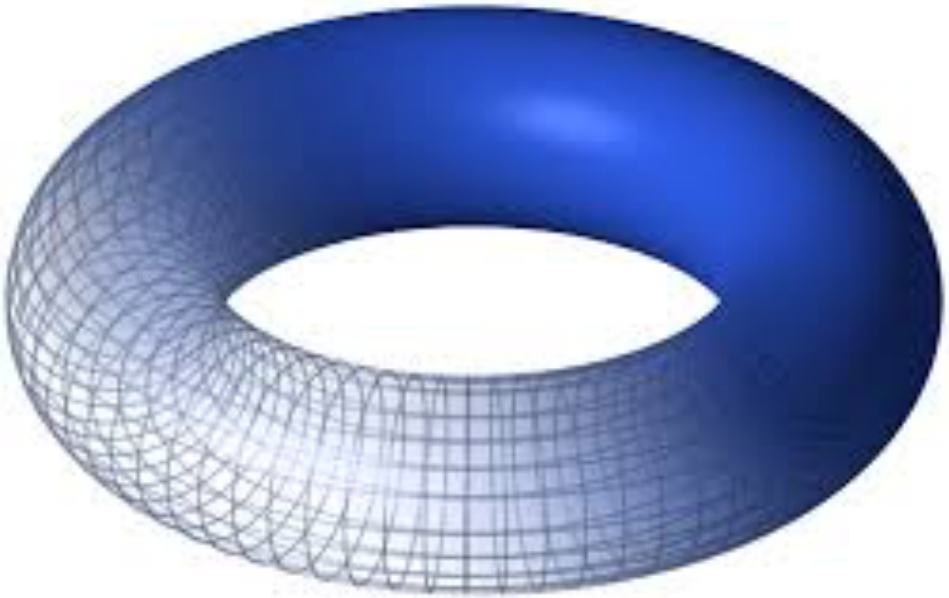
- Let's consider simulating cars on a highway to find probability of crash due to a moose entering the road
- We want
  - long stretch of road (steady state)
  - vehicle generation, moose generation
  - models
    - reaction time, breaking coefficient, headway, movement, overtaking
- We need to handle movement
- Relative location and velocity is dynamic

Scenario



Generation

- Generate according to distribution at beginning or
- Toroid approach



Mobility

- Need variables for {acceleration, speed, location, time}
- calculate time for next event (getting close to car in front)
- Often some kind of random walk, random waypoint etc. in simulations
  - calculate time to next velocity change and insert signal

Data structures to speed up process

- Avoid sending signals to all vehicles
- One approach, keep list of vehicles that can interact
  - array of locations, any event sent to neighbouring vehicles in array for processing
- Update array with every overtaking
- Think of this as a dynamic map of the system

Moose steps out

- MooseEnter(time, x-coordinate)
- Find vehicle closest to x-coordinate, send Event(moose, x-coordinate)
- Vehicle calculates stopping distance d. (never mind the signs)

$$t = \frac{v_0 - 0}{a} \qquad d = v_0 t + \frac{at^2}{2}$$

- if d > (x-coordinate - vehicle location) collision++

Scalability

- Large simulations - high complexity and/or large data volumes problem for your PC platform
- Standard practice, try to parallelise
  - run instances of simulator on different machines
  - program with threads, multi-core support
  - There even exist techniques for distributing simulations over clusters (not always easy)

Visualisation

- Can be very useful to increase understanding and insight
- Signals can be sent to GUI module

Real-time simulation

- Sometimes we want to see what happens in real time
  - normally simulate as fast as possible
- Bind events to real time by comparing with real time clock and use sleep() or similar