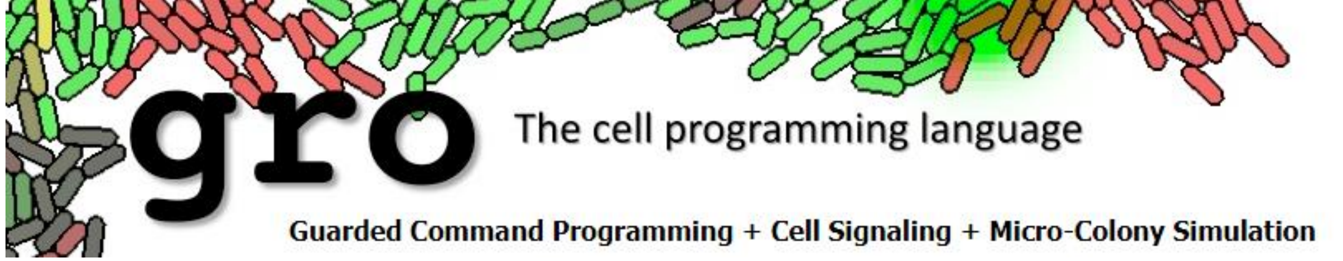
Gro Programming and Simulation:

BE 240 Lecture 5

Cindy Ren

05/07/2020

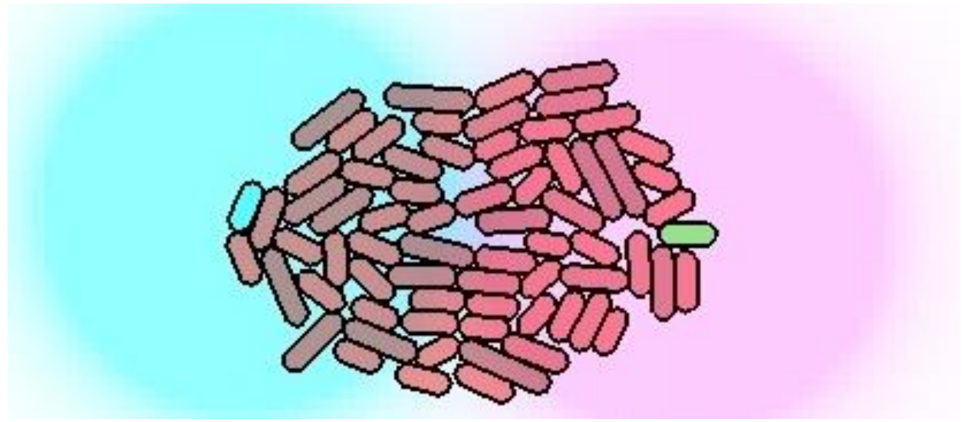
What is gro


gro
The cell programming language
Guarded Command Programming + Cell Signaling + Micro-Colony Simulation

Developed by The Klavins Lab, University Washington, Seattle, WA
http://depts.washington.edu/soslab/gro/index.html

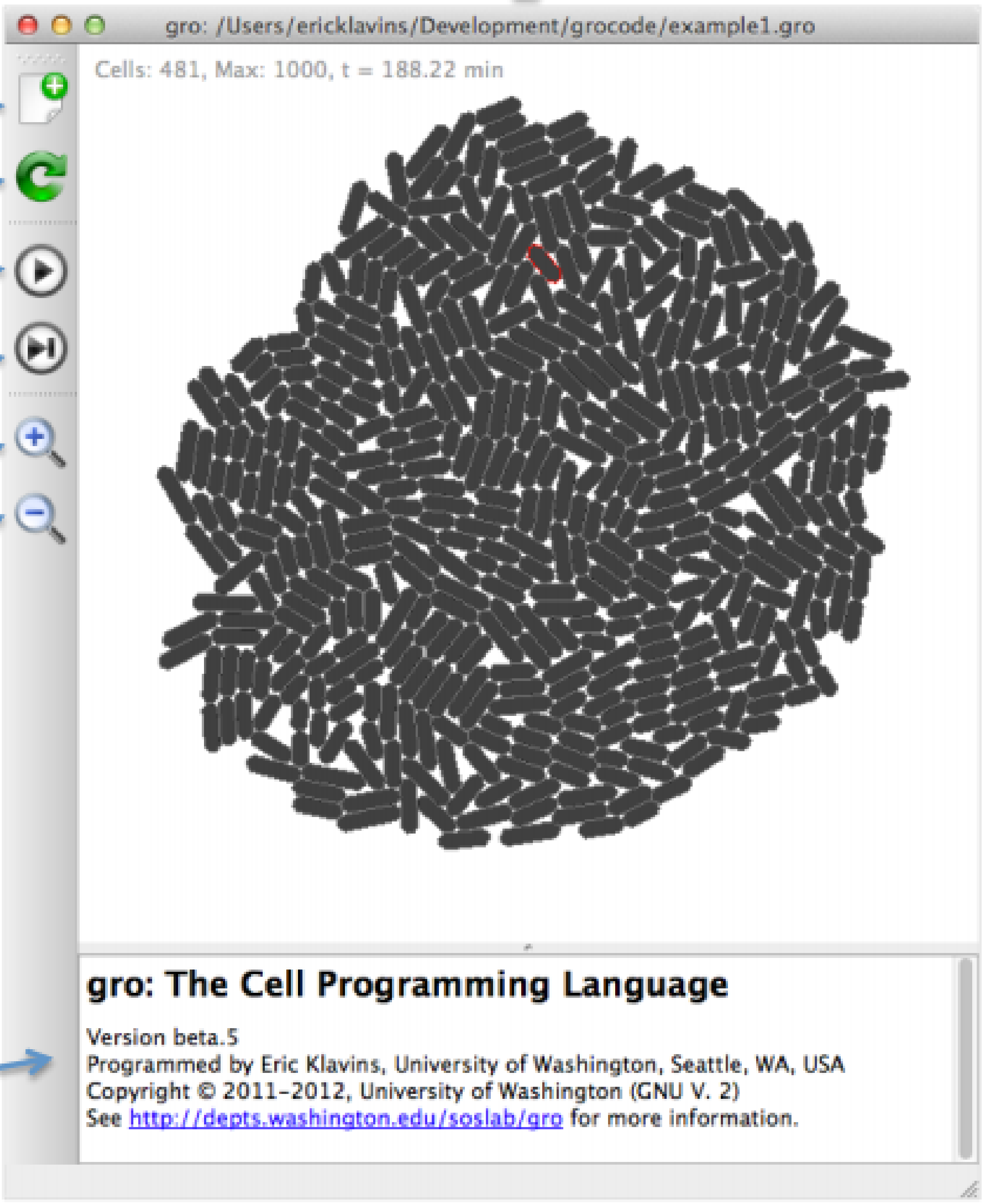
A tool for programming, modeling, specifying and simulating of multicellular behaviors in growing microcolonies in a 2D environment.

- Cell growth/division
- Cell crowding
- Signal diffusion
- Molecular reactions



E. coli microcolonies growing in a single layer ↔ fluorescence microscope

User interface


gro: /Users/ericklavins/Development/grocode/example1.gro
Cells: 481, Max: 1000, t = 188.22 min

gro: The Cell Programming Language

Version beta.5
Programmed by Eric Klavins, University of Washington, Seattle, WA, USA
Copyright © 2011-2012, University of Washington (GNU V. 2)
See http://depts.washington.edu/soslab/gro for more information.

Name of current file.

Open a gro file.

Reload the current file.

Start/stop the smulation.

Simulate one step.

Zoom in

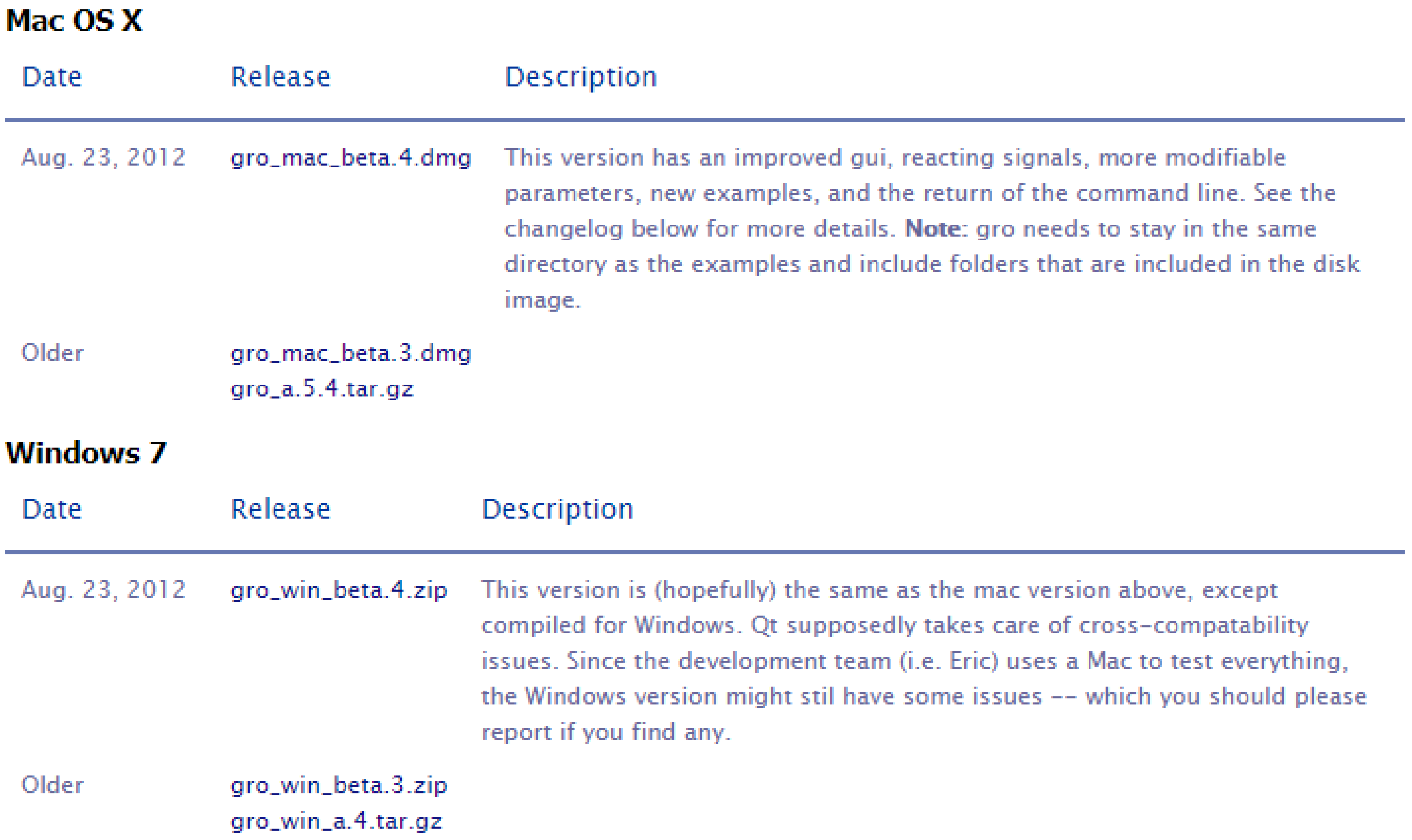
Zoom out.

Note: All UI functions are available in the menu and most have keyboard shortcuts.

Messages and errors show up here.

Installation

http://depts.washington.edu/soslab/gro/download.php



Mac OS X

Date	Release	Description
Aug. 23, 2012	gro_mac_beta.4.dmg	This version has an improved gui, reacting signals, more modifiable parameters, new examples, and the return of the command line. See the changelog below for more details. **Note**: gro needs to stay in the same directory as the examples and include folders that are included in the disk image.
Older	gro_mac_beta.3.dmg gro_a.5.4.tar.gz	

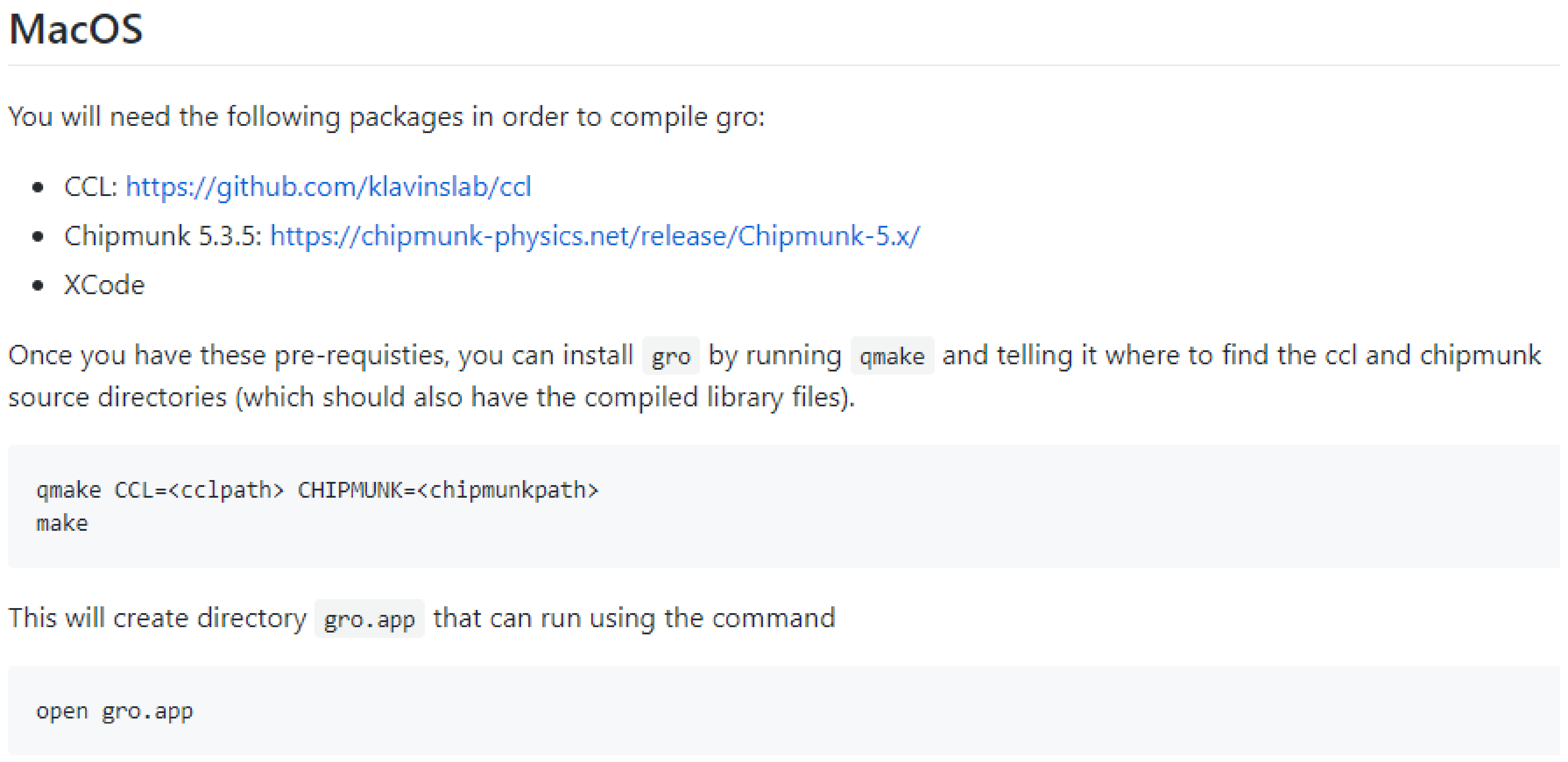
Windows 7

Date	Release	Description
Aug. 23, 2012	gro_win_beta.4.zip	This version is (hopefully) the same as the mac version above, except compiled for Windows. Qt supposedly takes care of cross-compatability issues. Since the development team (i.e. Eric) uses a Mac to test everything, the Windows version might stil have some issues -- which you should please report if you find any.
Older	gro_win_beta.3.zip gro_win_a.4.tar.gz	

small bug: path configuration

Installation

https://github.com/murrayrm/gro



MacOS

You will need the following packages in order to compile gro:

- CCL: https://github.com/klavinslab/ccl
- Chipmunk 5.3.5: https://chipmunk-physics.net/release/Chipmunk-5.x/
- XCode

Once you have these pre-requisties, you can install `gro` by running `qmake` and telling it where to find the ccl and chipmunk source directories (which should also have the compiled library files).

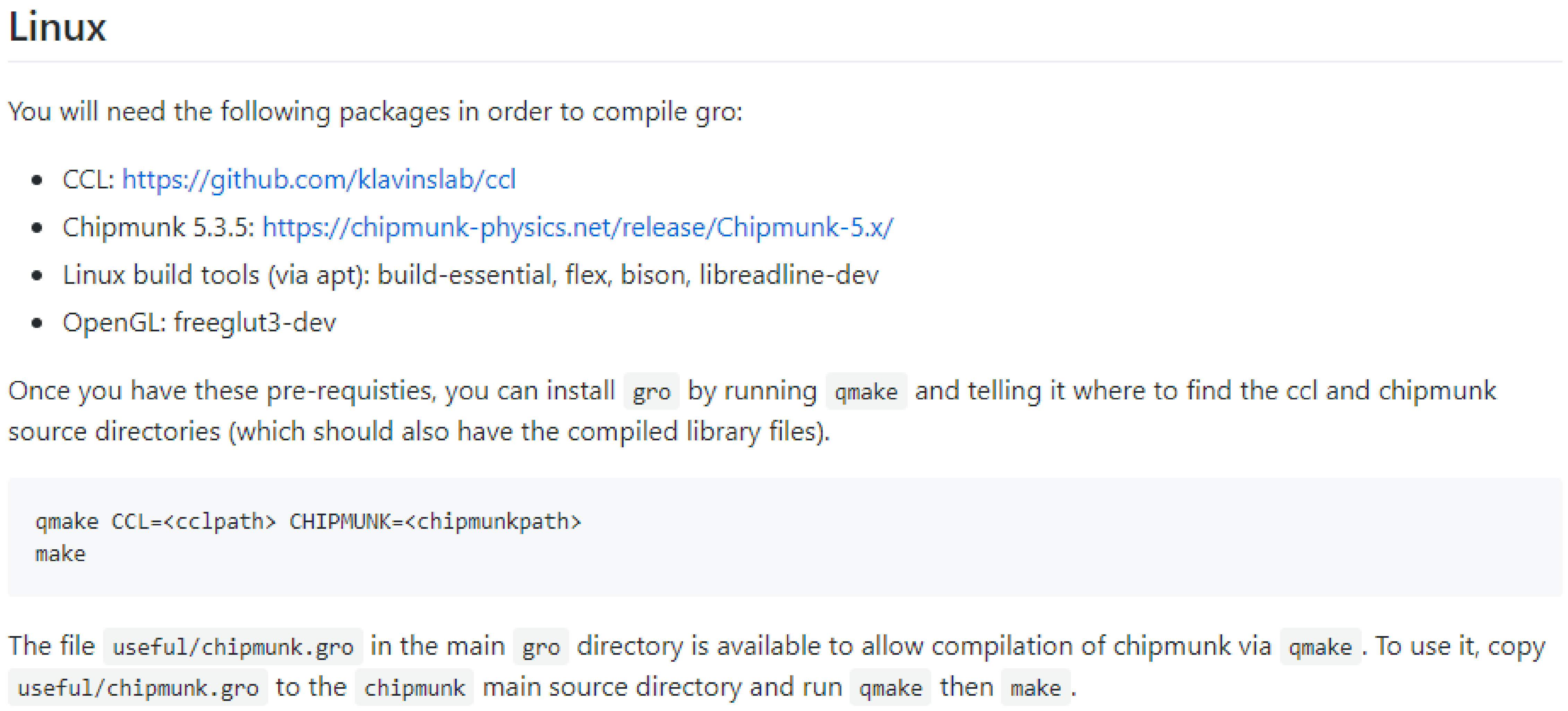
```
qmake CCL=<cclpath> CHIPMUNK=<chipmunkpath>
make
```

This will create directory `gro.app` that can run using the command

```
open gro.app
```

Installation

https://github.com/murrayrm/gro



Linux

You will need the following packages in order to compile gro:

- CCL: https://github.com/klavinslab/ccl
- Chipmunk 5.3.5: https://chipmunk-physics.net/release/Chipmunk-5.x/
- Linux build tools (via apt): build-essential, flex, bison, libreadline-dev
- OpenGL: freeglut3-dev

Once you have these pre-requisties, you can install `gro` by running `qmake` and telling it where to find the ccl and chipmunk source directories (which should also have the compiled library files).

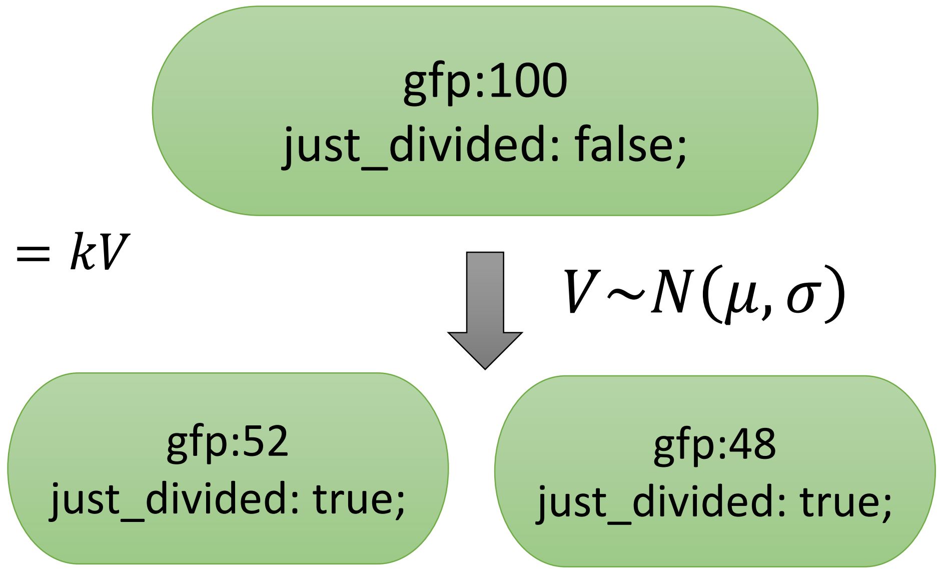
```
qmake CCL=<cclpath> CHIPMUNK=<chipmunkpath>
make
```

The file `useful/chipmunk.gro` in the main `gro` directory is available to allow compilation of chipmunk via `qmake`. To use it, copy `useful/chipmunk.gro` to the `chipmunk` main source directory and run `qmake` then `make`.

Simulation environment

E. coli-like bacteria

- Cell growth: volume grows exponentially $\frac{dV}{dt} = kV$
- Cell division: divides after doubled in size
- Cell death: removed after dead
- Cell crowding: physics contact (Chipmunk)



- Signal diffusion: 2D grid of square elements (Finite element method)

- Molecular reactions: stochastic events (guarded command g:c)

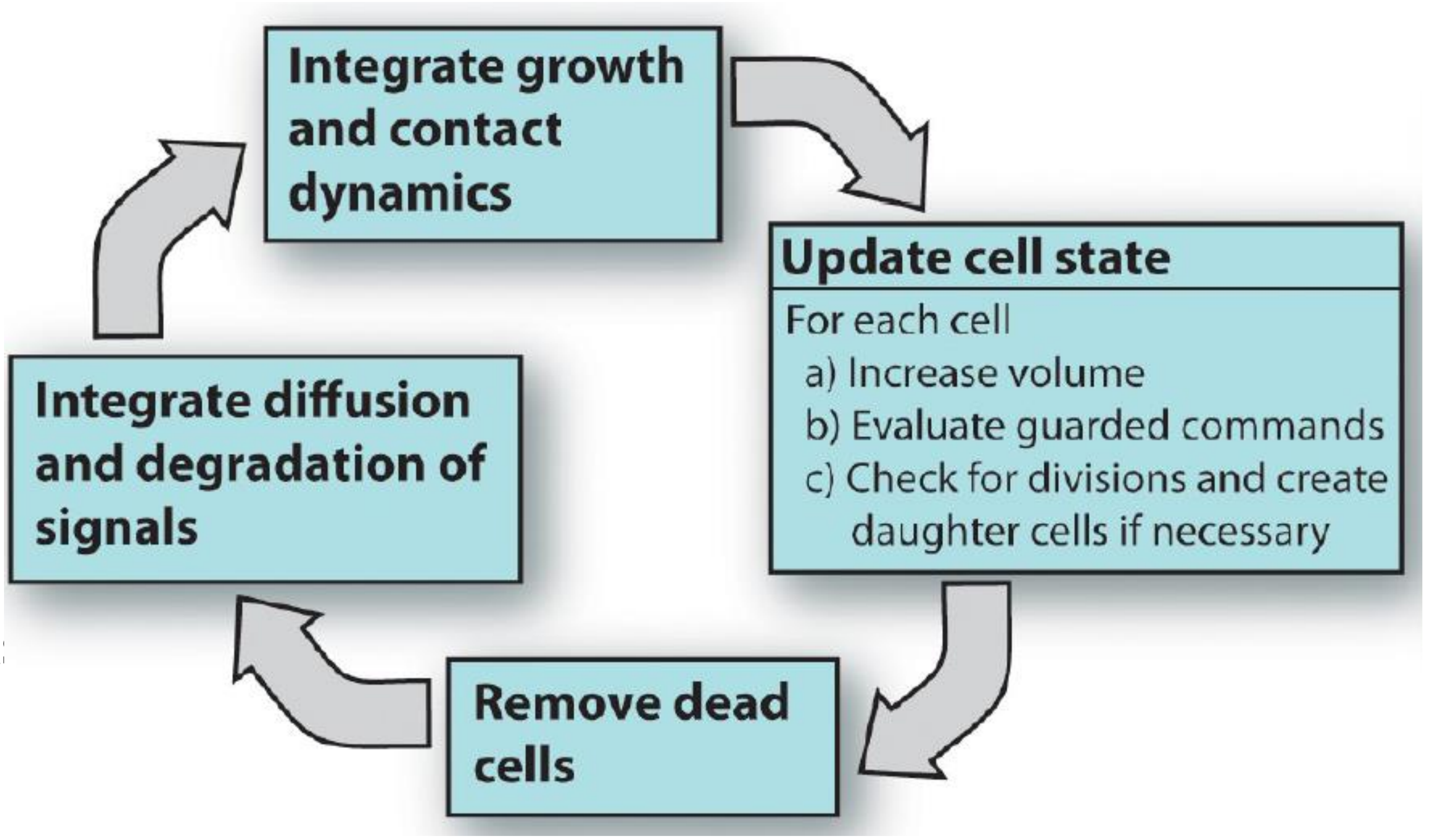
- Chemostat mode

Simulation environment

E. coli-like bacteria

- Cell growth
- Cell division
- Cell death
- Cell crowding

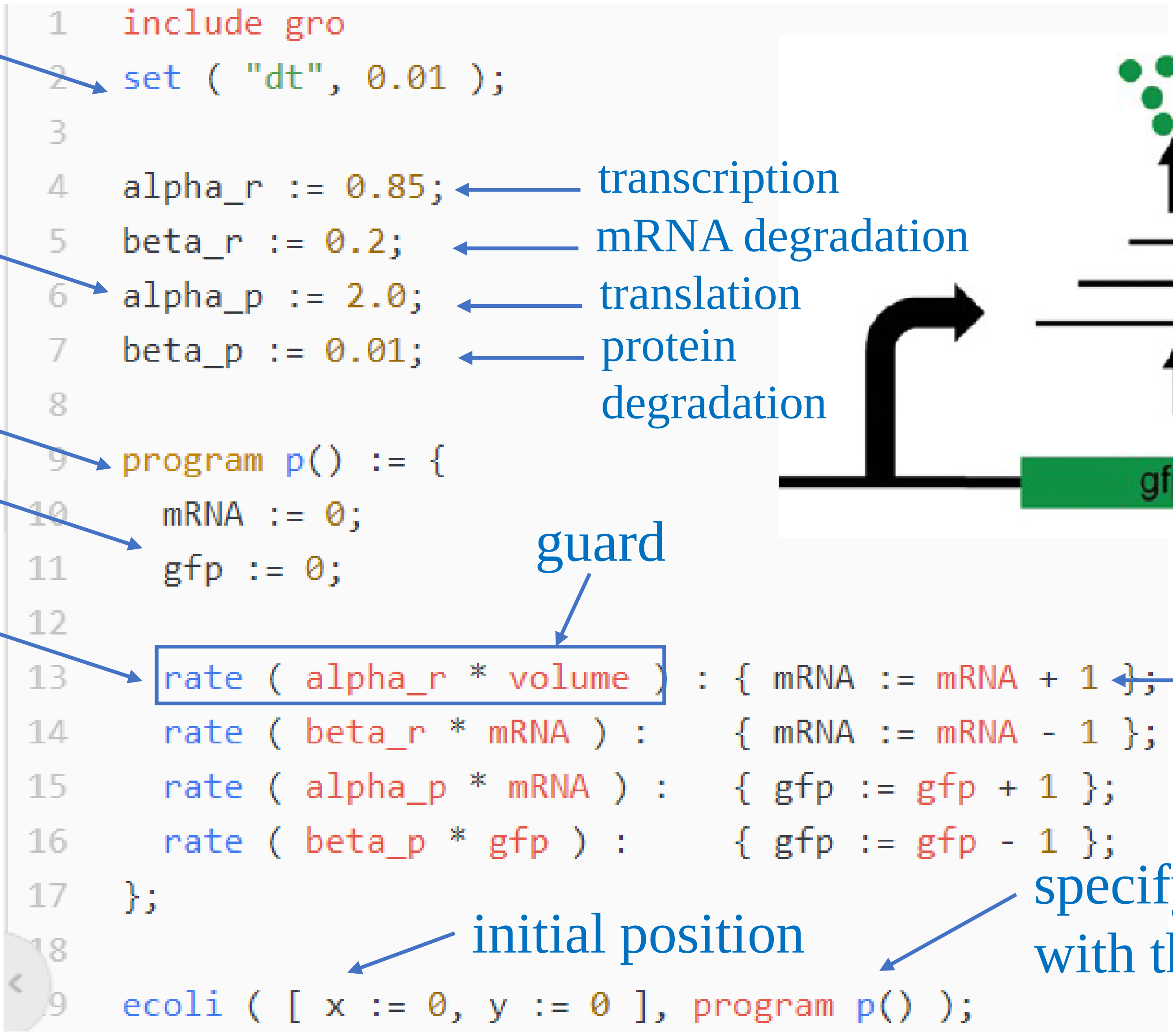
- Signal diffusion
- Molecular reaction


Integrate growth and contact dynamics
Update cell state
For each cell
a) Increase volume
b) Evaluate guarded commands
c) Check for divisions and create daughter cells if necessary
Remove dead cells
Integrate diffusion and degradation of signals

Program p ()

gro programming: inside the cell

example_gfp_simple.gro



```
include gro
set ( "dt", 0.01 );

alpha_r := 0.85;
beta_r := 0.2;
alpha_p := 2.0;
beta_p := 0.01;

program p() := {
  mRNA := 0;
  gfp := 0;

  rate ( alpha_r * volume ) : { mRNA := mRNA + 1 };
  rate ( beta_r * mRNA ) :    { mRNA := mRNA - 1 };
  rate ( alpha_p * mRNA ) :   { gfp := gfp + 1 };
  rate ( beta_p * gfp ) :     { gfp := gfp - 1 };
};

ecoli ( [ x := 0, y := 0 ], program p() );
```

timestep

parameters

cell program

initialization

guarded command

guard

command

transcription

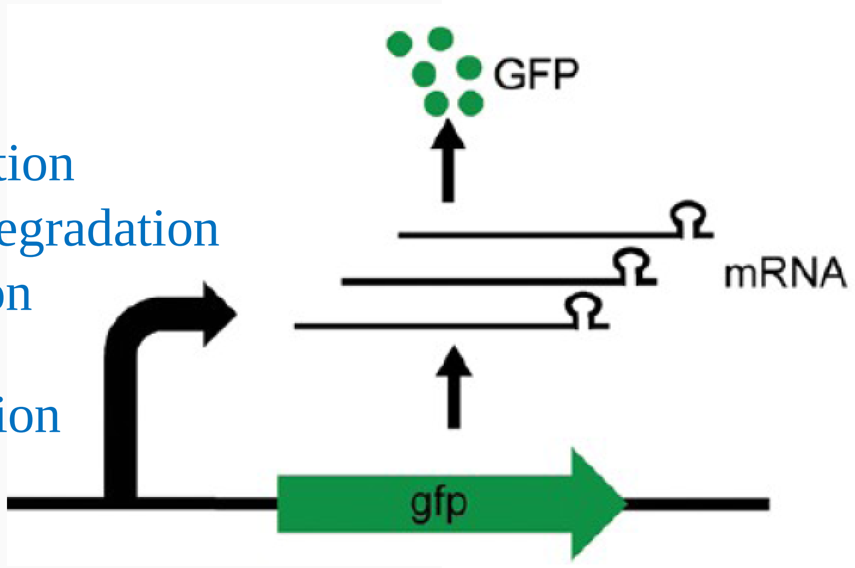
mRNA degradation

translation

protein degradation

initial position

specify the E. coli with the program



rate(p) is true every dt timestep with probability p*dt.

gfp production is balanced by the dilution (growth/division).

Program p ()

gro programming: more functions

Mass action propensities:

```
rate ( alpha_r * volume ) : { mRNA := mRNA + 1 };
rate ( beta_r * mRNA ) :    { mRNA := mRNA - 1 };
rate ( alpha_p * mRNA ) :   { gfp := gfp + 1 };
rate ( beta_p * gfp ) :     { gfp := gfp - 1 };
```

Define functions:

$$f_{hill}(v,k,x) = v\frac{x}{k+x}$$

```
fun hill  v k x . v * x / ( k + x );
```

Non-mass action propensities:

```
rate ( hill v k x ) : { gfp := gfp + 1 };
```

Other functions:

$$f_{logistic}(v,kmax,x) = v\left(1-\frac{x}{kmax}\right)x$$

```
fun logistic  v kmax x . v * ( 1 - x / kmax ) * x;
```

$$f_{fact}(n) = n!$$

```
fun fact n .
  if n <= 0
    then 1
    else n * fact (n-1)
  end;
```

```
+, -, *, ^, /, %
sqrt(x), sin(y)
```

gro simulation example_gfp_const.gro

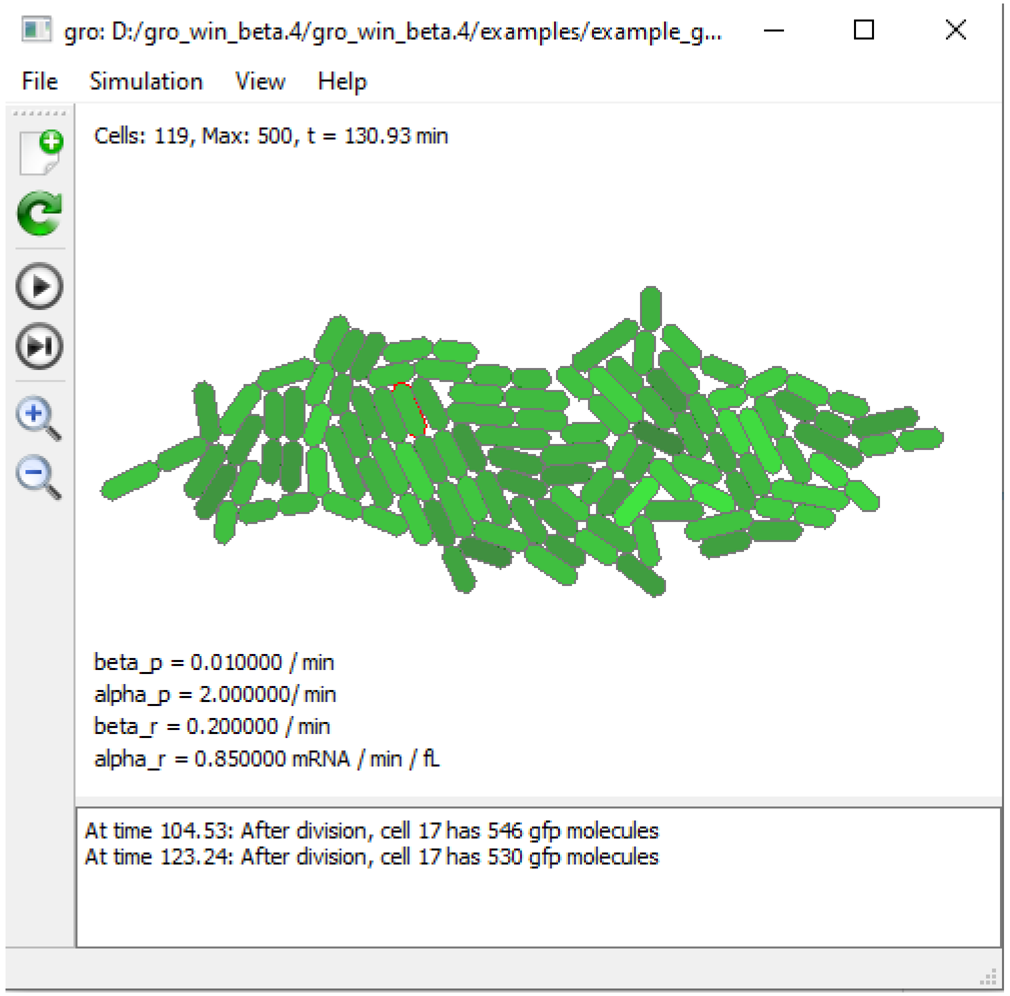
- Keep track of the time

```
program p() := {
  mRNA := 0;
  gfp := 0;

  rate ( alpha_r * volume ) : { mRNA := mRNA + 1 };
  rate ( beta_r * mRNA ) :    { mRNA := mRNA - 1 };
  rate ( alpha_p * mRNA ) :   { gfp := gfp + 1 };
  rate ( beta_p * gfp ) :     { gfp := gfp - 1 };

  r := [ t := 0 ];
  selected & just_divided : {
    print ( "At time ", r.t, ": After division, cell ",
     id, " has ", gfp, " gfp molecules" )
  };
  true : { r.t := r.t + dt };
};
```

hide time in a record

track the cell id

time increments



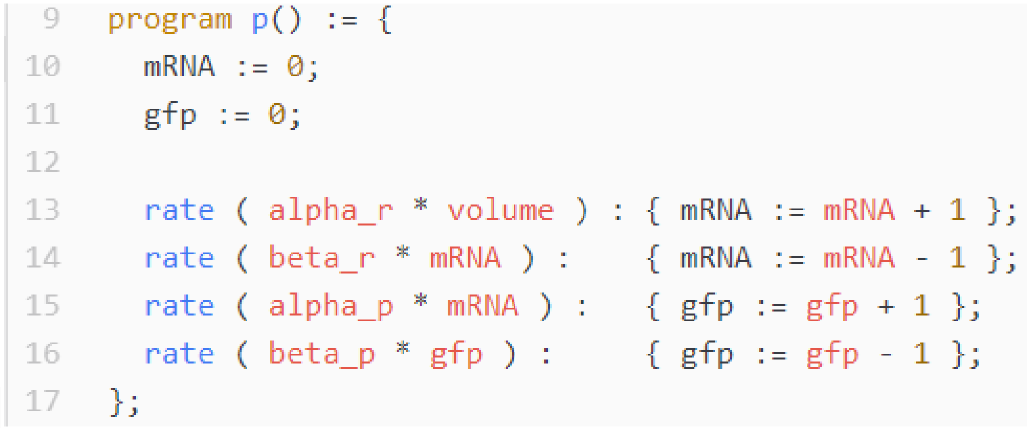
Cells: 119, Max: 500, t = 130.93 min

beta_p = 0.010000 / min
alpha_p = 2.000000/ min
beta_r = 0.200000 / min
alpha_r = 0.850000 mRNA / min / fL

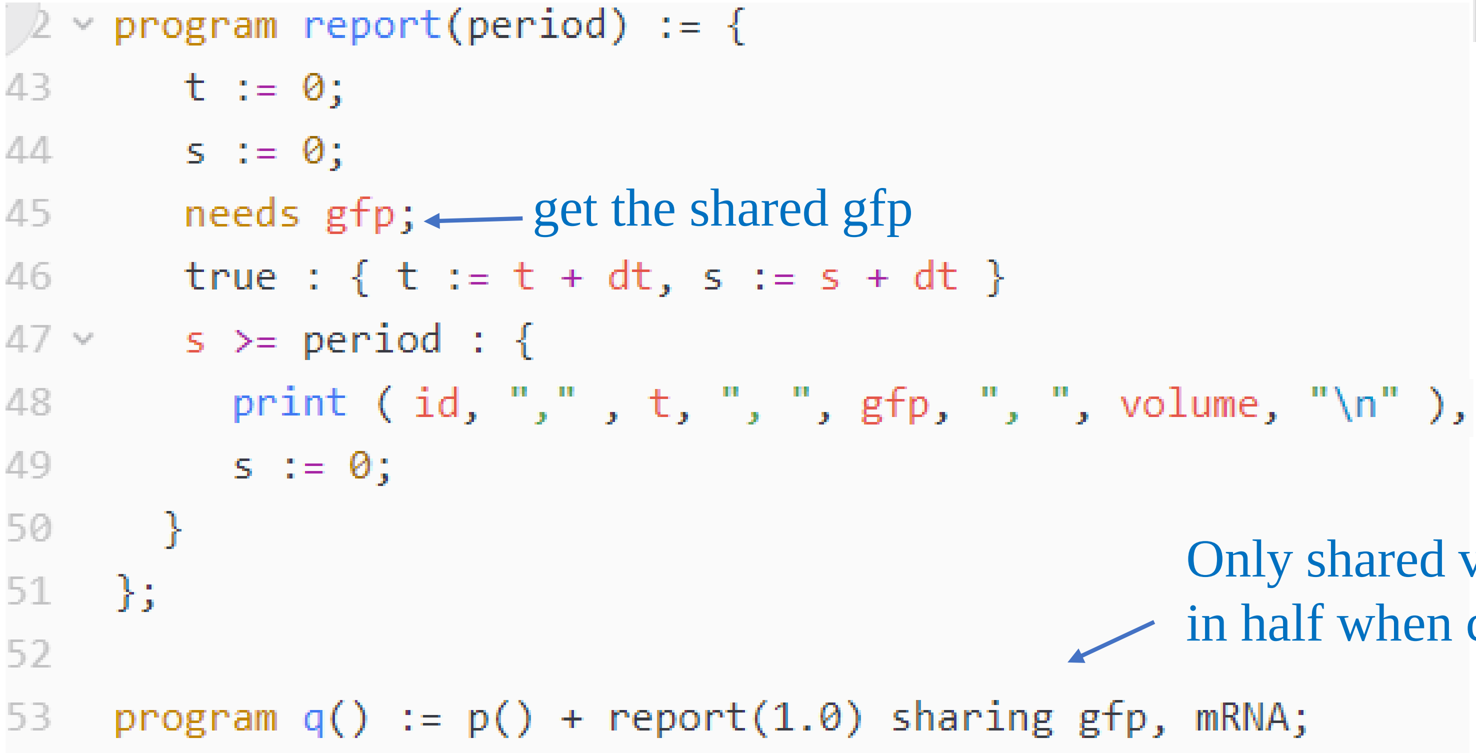
```
At time 104.53: After division, cell 17 has 546 gfp molecules
At time 123.24: After division, cell 17 has 530 gfp molecules
```

gro simulation example_gfp_const.gro

- Keep track of the time by composing two program



```
program p() := {
  mRNA := 0;
  gfp := 0;

  rate ( alpha_r * volume ) : { mRNA := mRNA + 1 };
  rate ( beta_r * mRNA ) :    { mRNA := mRNA - 1 };
  rate ( alpha_p * mRNA ) :   { gfp := gfp + 1 };
  rate ( beta_p * gfp ) :     { gfp := gfp - 1 };
};
```



```
program report(period) := {
    t := 0;
    s := 0;
    needs gfp;
    true : { t := t + dt, s := s + dt }
    s >= period : {
      print ( id, "," , t, ", ", gfp, ", ", volume, "\n" ),
      s := 0;
   }
};

program q() := p() + report(1.0) sharing gfp, mRNA;
```

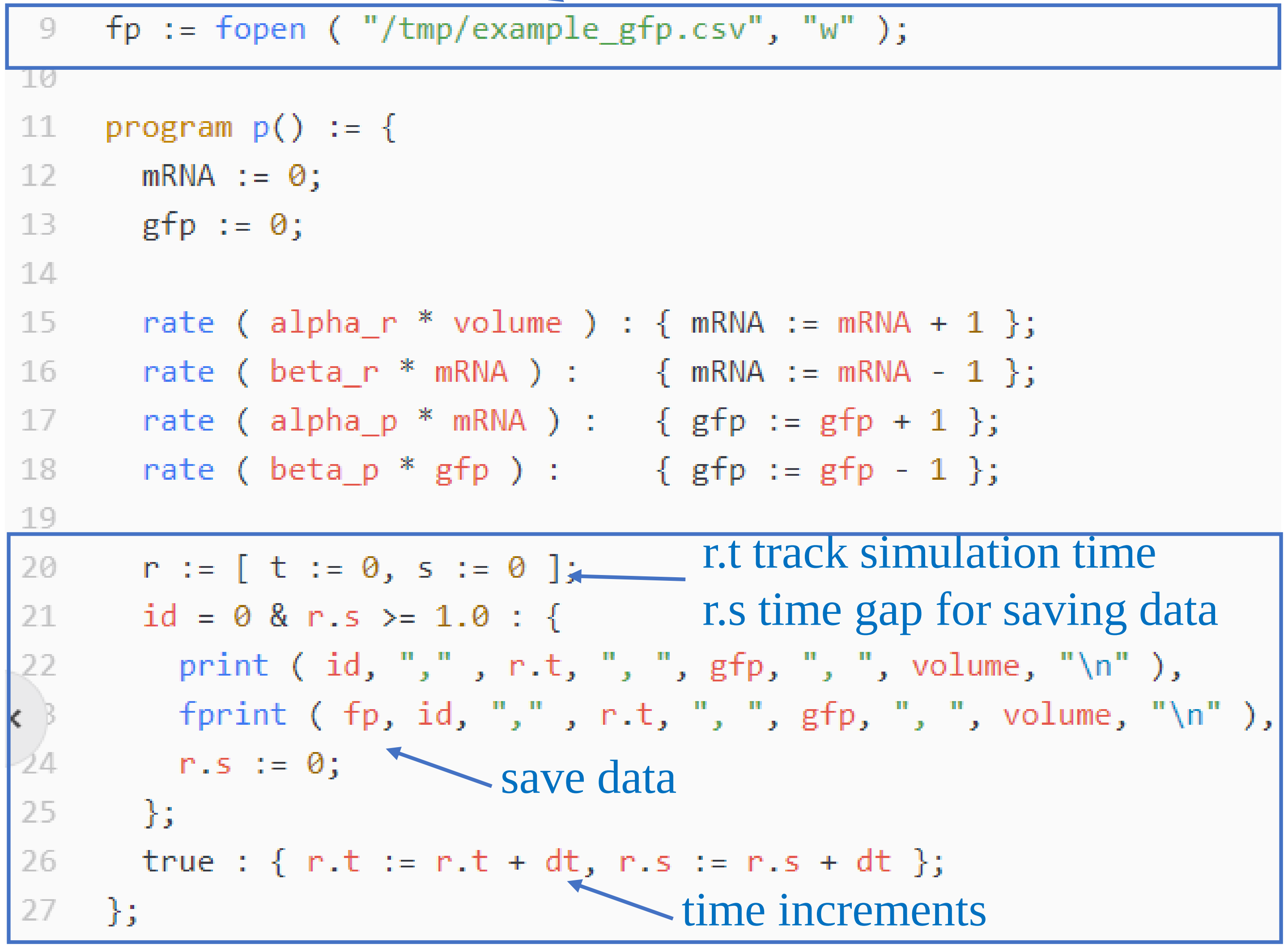
get the shared gfp

Only shared variables get cut in half when cell divides

gro simulation example_gfp_const.gro

- Save data in csv

write here



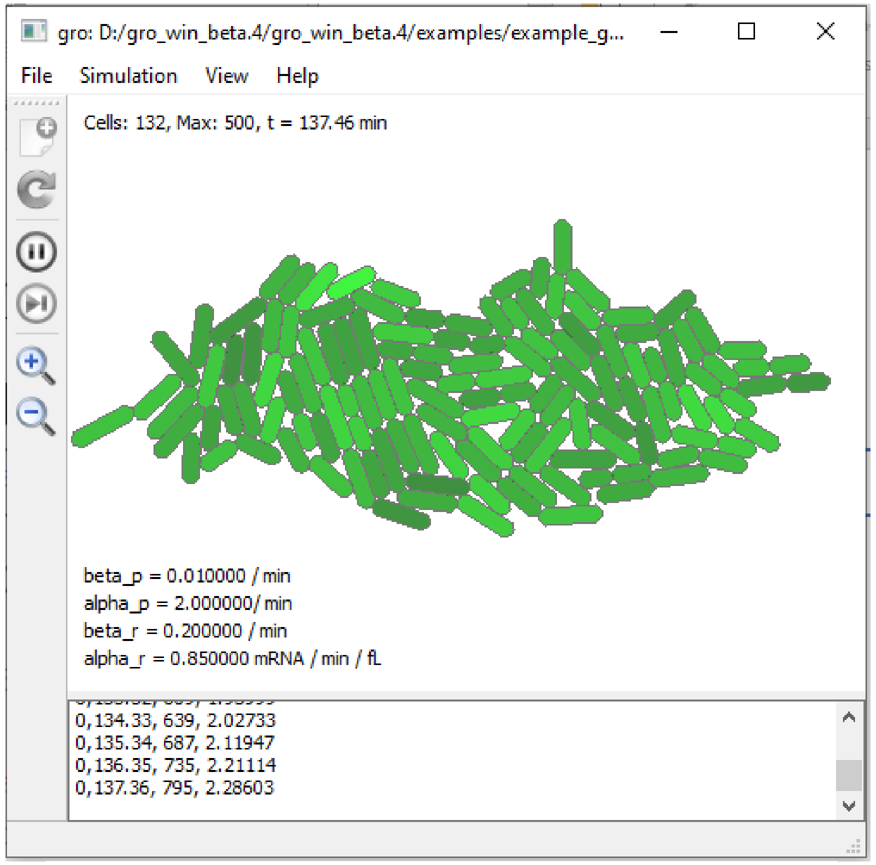
```
fp := fopen ( "/tmp/example_gfp.csv", "w" );

program p() := {
  mRNA := 0;
  gfp := 0;

  rate ( alpha_r * volume ) : { mRNA := mRNA + 1 };
  rate ( beta_r * mRNA ) :    { mRNA := mRNA - 1 };
  rate ( alpha_p * mRNA ) :   { gfp := gfp + 1 };
  rate ( beta_p * gfp ) :     { gfp := gfp - 1 };

  r := [ t := 0, s := 0 ];
  id = 0 & r.s >= 1.0 : {
    print ( id, "," , r.t, ", ", gfp, ", ", volume, "\n" ),
    fprint ( fp, id, "," , r.t, ", ", gfp, ", ", volume, "\n" ),
    r.s := 0;
  };
  true : { r.t := r.t + dt, r.s := r.s + dt };
};
```

r.t track simulation time

r.s time gap for saving data

save data

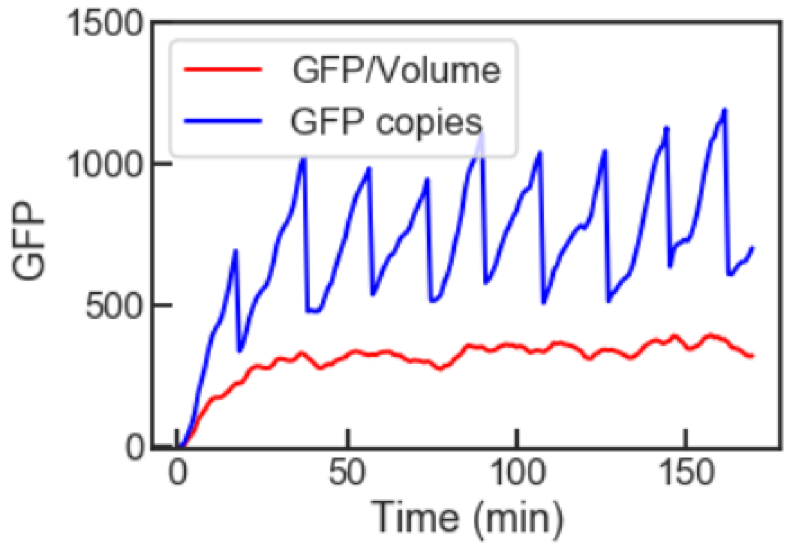
time increments



Cells: 132, Max: 500, t = 137.46 min

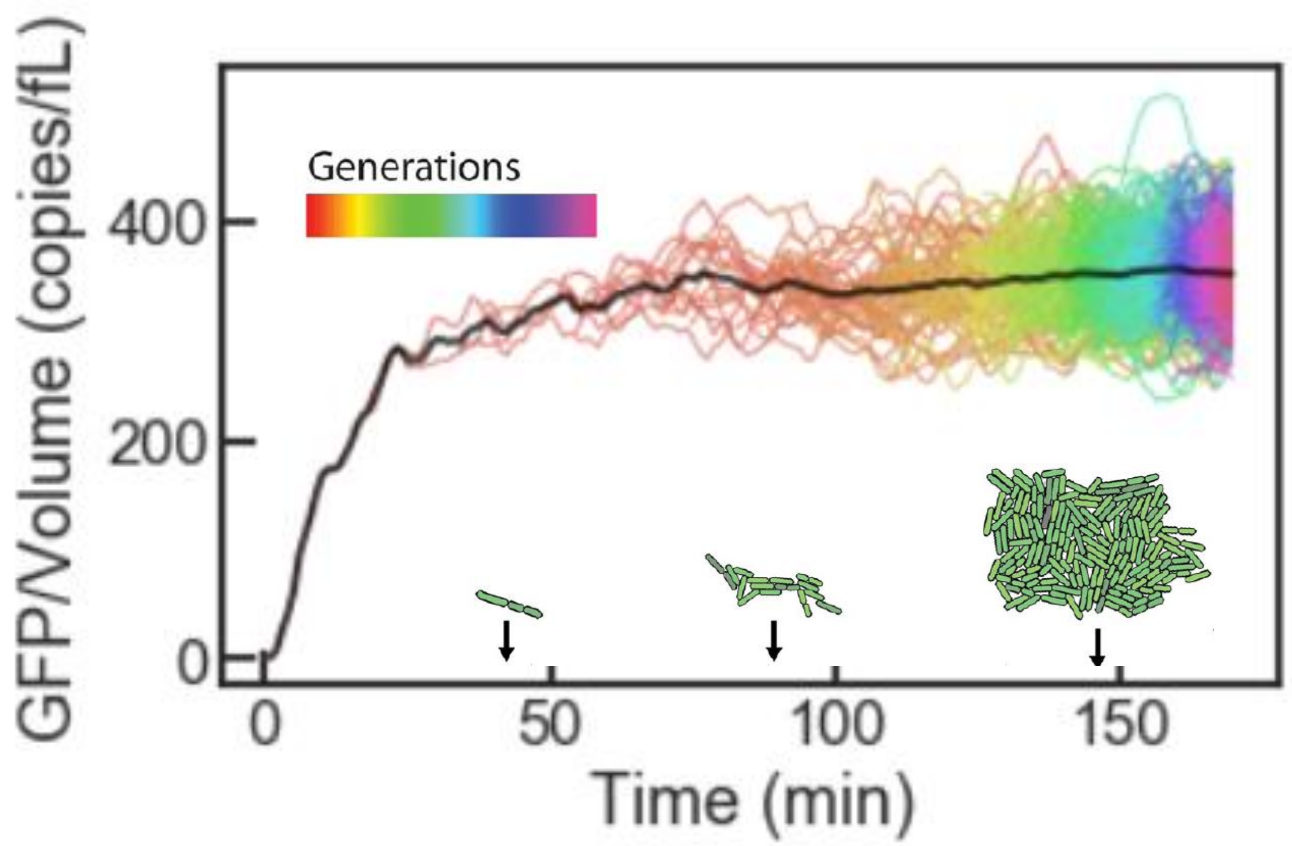
beta_p = 0.010000 / min
alpha_p = 2.000000/ min
beta_r = 0.200000 / min
alpha_r = 0.850000 mRNA / min / fL

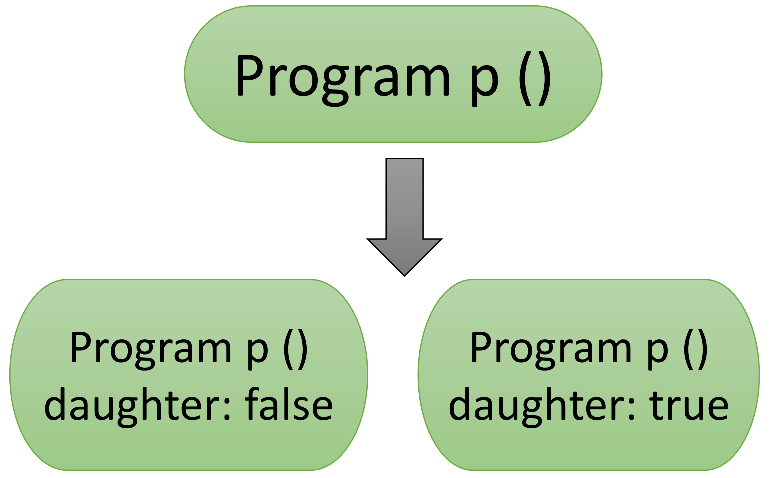
```
0,134.33, 639, 2.02733
0,135.34, 687, 2.11947
0,136.35, 735, 2.21114
0,137.36, 795, 2.28603
```

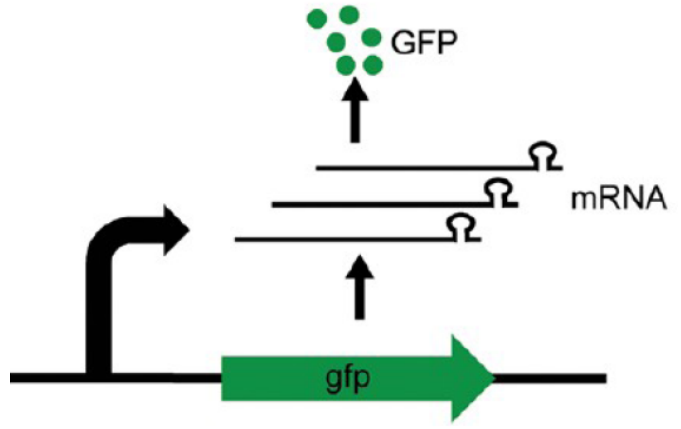


gro simulation

example_gfp_const.gro







Cell divides →

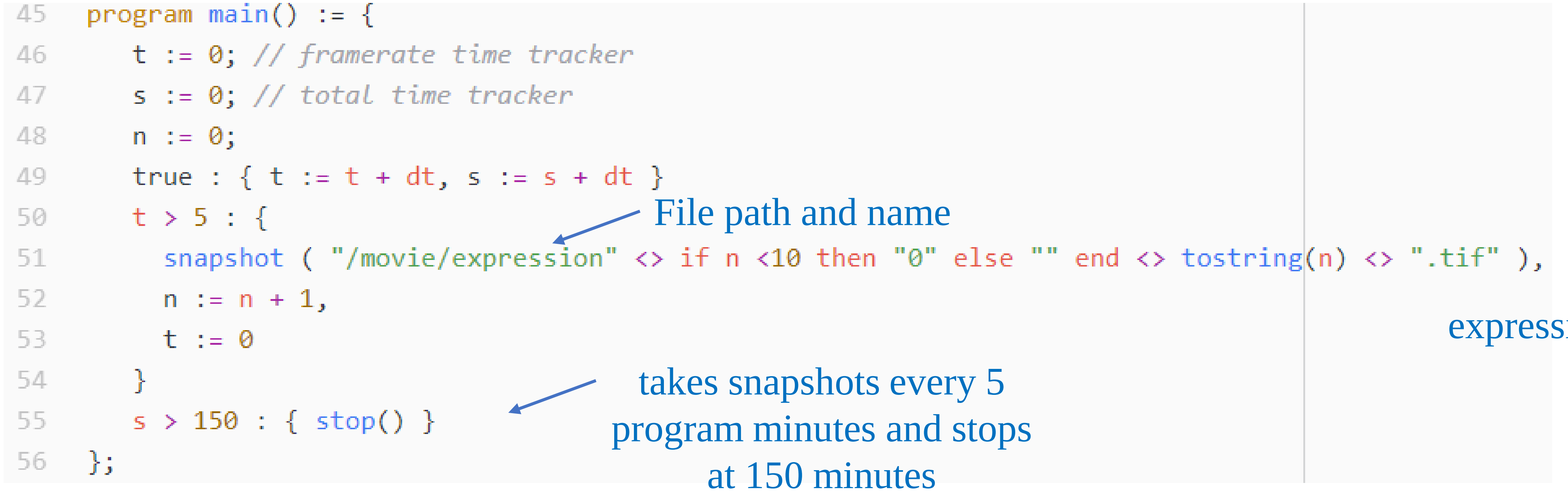
- program is copied
- numerical variables are cut in half approximately

→ 2^n copies of program after n generations

gro simulation

example_gfp_const.gro

- Save snapshots for movies



```
program main() := {
   t := 0; // framerate time tracker
   s := 0; // total time tracker
   n := 0;
   true : { t := t + dt, s := s + dt }
   t > 5 : {
     snapshot ( "/movie/expression" <> if n <10 then "0" else "" end <> tostring(n) <> ".tif" ),
     n := n + 1,
     t := 0
   }
   s > 150 : { stop() }
};
```

File path and name

expressionNN.tif

takes snapshots every 5 program minutes and stops at 150 minutes

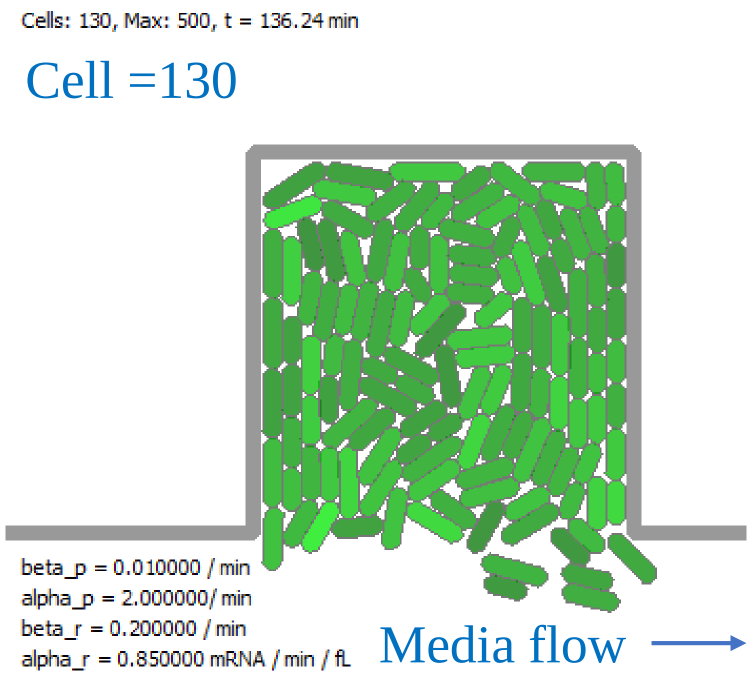
gro simulation example_gfp_const.gro

- Chemostat Mode

```
include gro
chemostat( true );
set ( "chemostat_width", 40 );
set ( "chemostat_height", 200 );
```

set chemostat size

Chemostat mode allows you to keep a more or less fixed population of cells in the simulation.



```
Cells: 130, Max: 500, t = 136.24 min
```

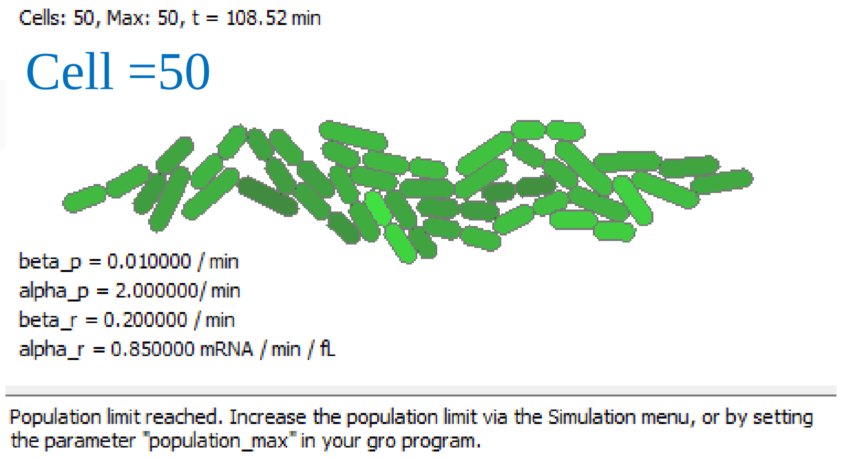
Cell =130

```
beta_p = 0.010000 / min
alpha_p = 2.000000/ min
beta_r = 0.200000 / min
alpha_r = 0.850000 mRNA / min / fL
```

Media flow

- Set population cap

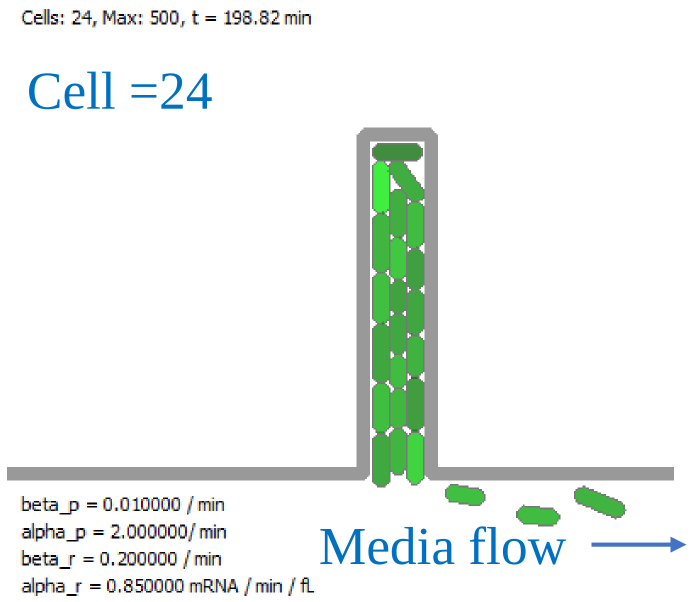
```
set ( "population_max", 50);
```



```
Cells: 50, Max: 50, t = 108.52 min
```

Cell =50

```
beta_p = 0.010000 / min
alpha_p = 2.000000/ min
beta_r = 0.200000 / min
alpha_r = 0.850000 mRNA / min / fL

Population limit reached. Increase the population limit via the Simulation menu, or by setting
the parameter "population_max" in your gro program.
```



```
Cells: 24, Max: 500, t = 198.82 min
```

Cell =24

```
beta_p = 0.010000 / min
alpha_p = 2.000000/ min
beta_r = 0.200000 / min
alpha_r = 0.850000 mRNA / min / fL
```

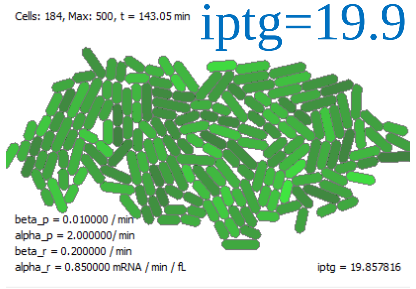
Media flow

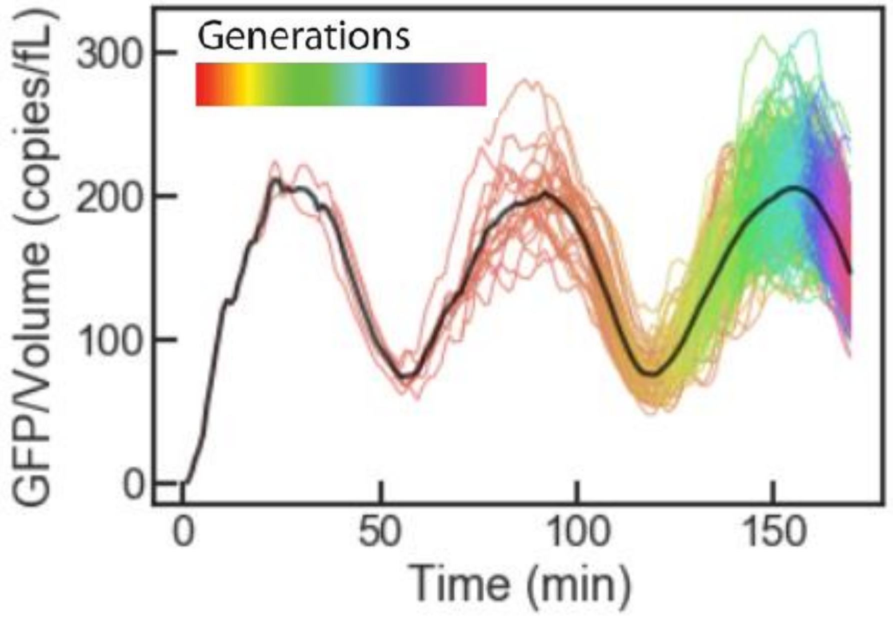
example_gfp_sin.gro

gro simulation

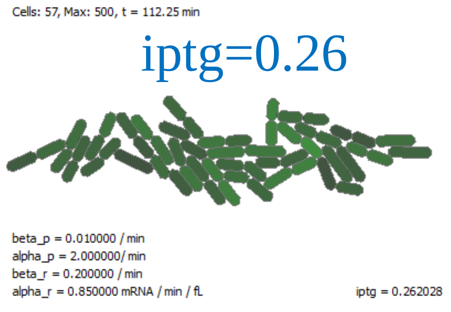
- Global control

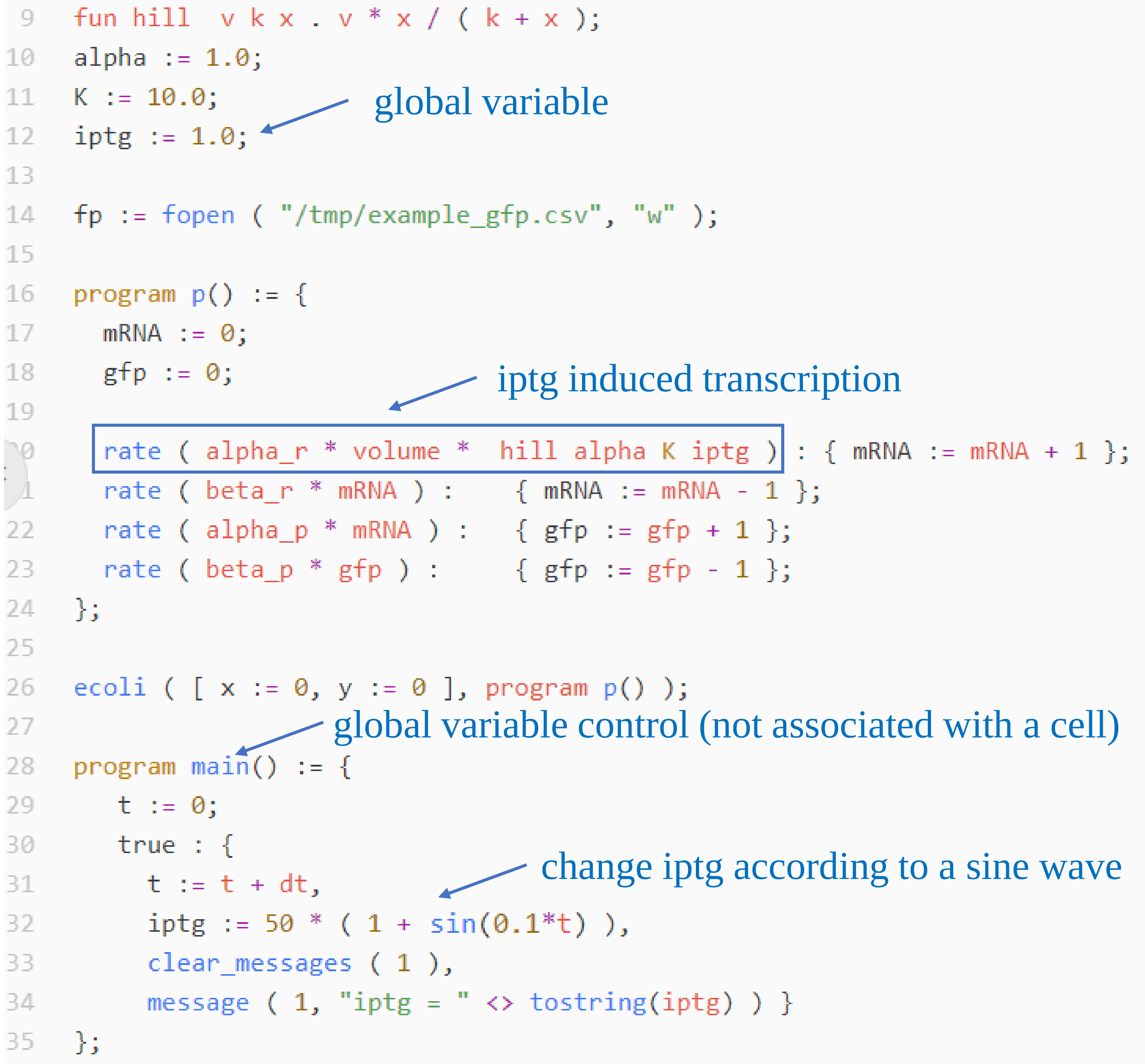
iptg=19.9





iptg=0.26





```
fun hill  v k x . v * x / ( k + x );
alpha := 1.0;
K := 10.0;
iptg := 1.0;

fp := fopen ( "/tmp/example_gfp.csv", "w" );

program p() := {
  mRNA := 0;
  gfp := 0;

  rate ( alpha_r * volume *  hill alpha K iptg ) : { mRNA := mRNA + 1 };
  rate ( beta_r * mRNA ) :    { mRNA := mRNA - 1 };
  rate ( alpha_p * mRNA ) :   { gfp := gfp + 1 };
  rate ( beta_p * gfp ) :     { gfp := gfp - 1 };
};

ecoli ( [ x := 0, y := 0 ], program p() );

program main() := {
   t := 0;
   true : {
     t := t + dt,
     iptg := 50 * ( 1 + sin(0.1*t) ),
     clear_messages ( 1 ),
     message ( 1, "iptg = " <> tostring(iptg) ) }
};
```

global variable

iptg induced transcription

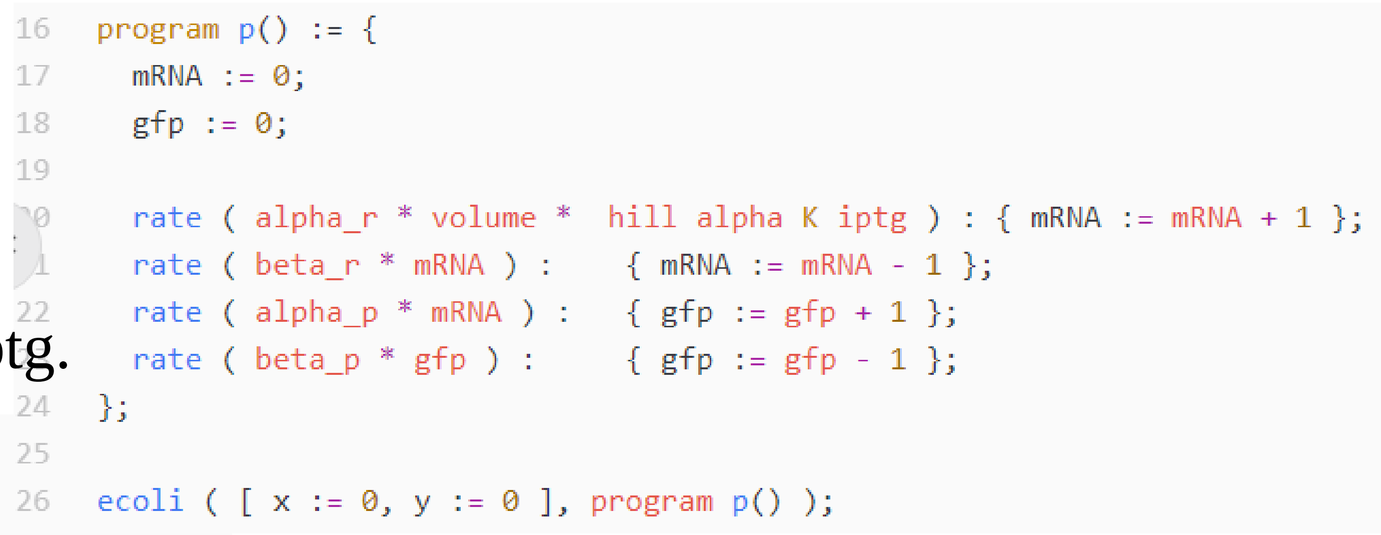
global variable control (not associated with a cell)

change iptg according to a sine wave

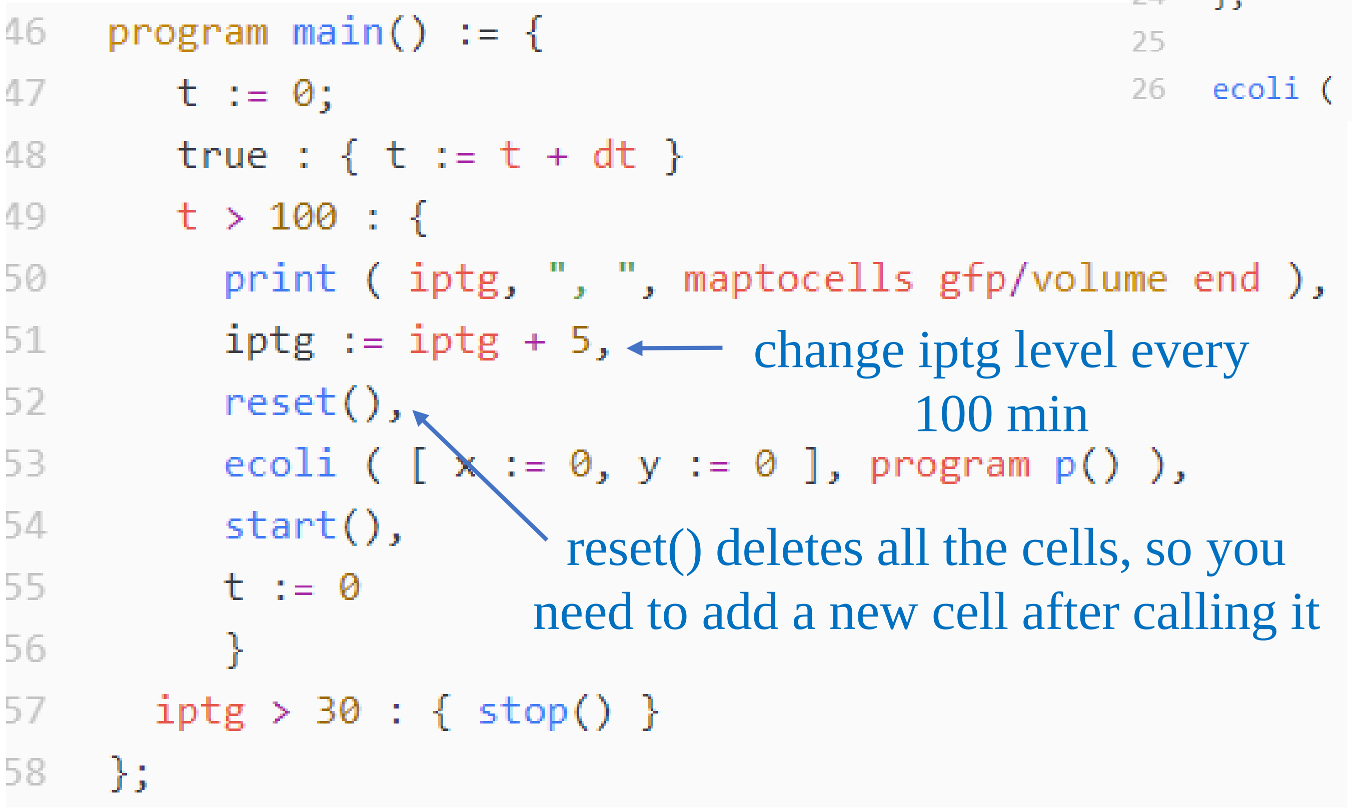
gro simulation

- Simulation control

Run multiple simulations with different iptg.



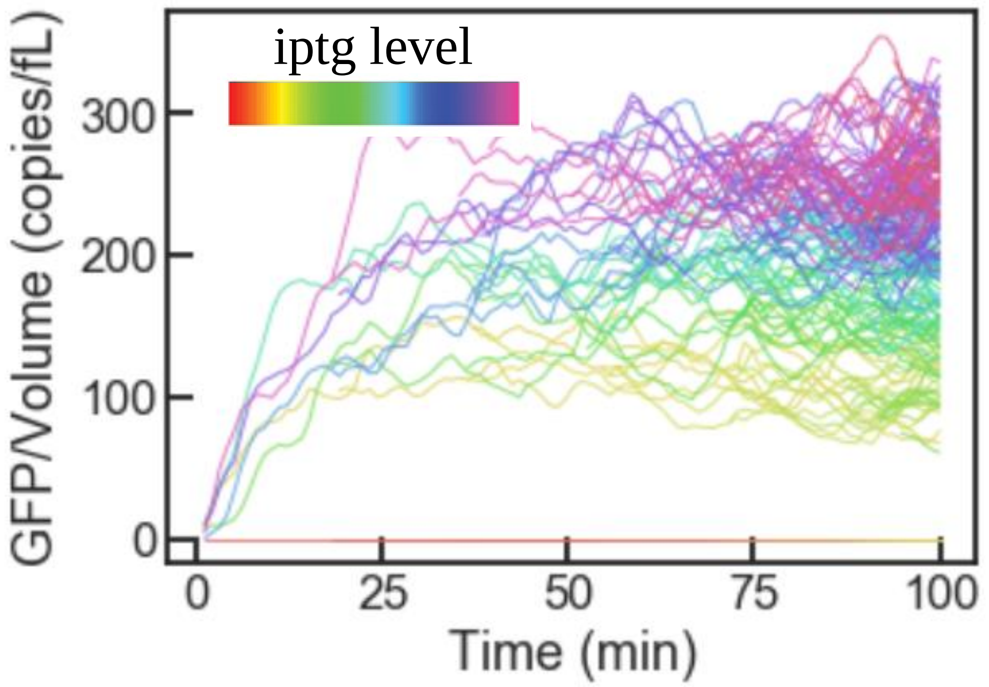
```
program p() := {
  mRNA := 0;
  gfp := 0;

  rate ( alpha_r * volume *  hill alpha K iptg ) : { mRNA := mRNA + 1 };
  rate ( beta_r * mRNA ) :    { mRNA := mRNA - 1 };
  rate ( alpha_p * mRNA ) :   { gfp := gfp + 1 };
  rate ( beta_p * gfp ) :     { gfp := gfp - 1 };
};

ecoli ( [ x := 0, y := 0 ], program p() );
```



```
program main() := {
   t := 0;
   true : { t := t + dt }
   t > 100 : {
     print ( iptg, ", ", maptocells gfp/volume end ),
     iptg := iptg + 5,
     reset(),
     ecoli ( [ x := 0, y := 0 ], program p() ),
     start(),
     t := 0
     }
  iptg > 30 : { stop() }
};
```

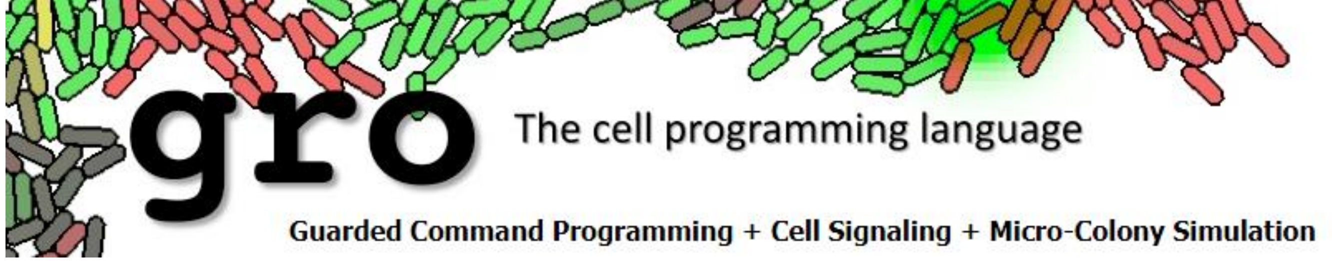
change iptg level every 100 min

reset() deletes all the cells, so you need to add a new cell after calling it



start(), stop(), and reset()

What is gro


gro
The cell programming language
Guarded Command Programming + Cell Signaling + Micro-Colony Simulation

A tool for the simulation of multicellular behaviors in a 2D environment.

- **Cell growth/division : try with bioscrape lineage**
- **Cell crowding**
- Signal diffusion
- **Molecular reactions (with noise) : try with bioscrape**

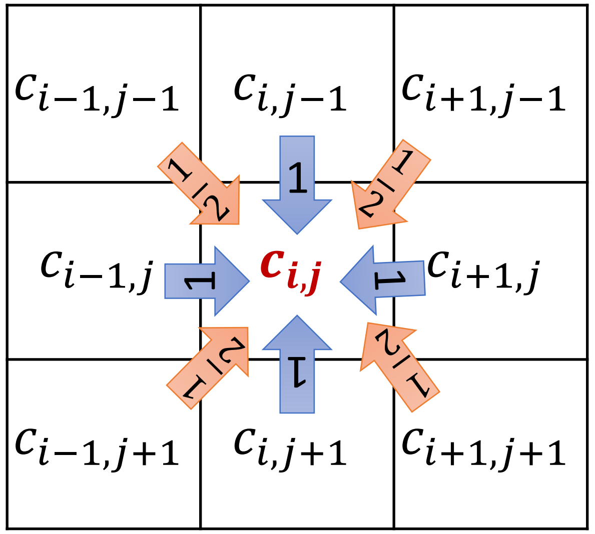
Spatial pattern formation, heterogeneity in populations…

gro signals : finte element model

- The 2D environment of the simulation is gridded up into 0.5 μm * 0.5 μm elements.

- The concentration of each element $c_{i,j}$ is updated at each time step via

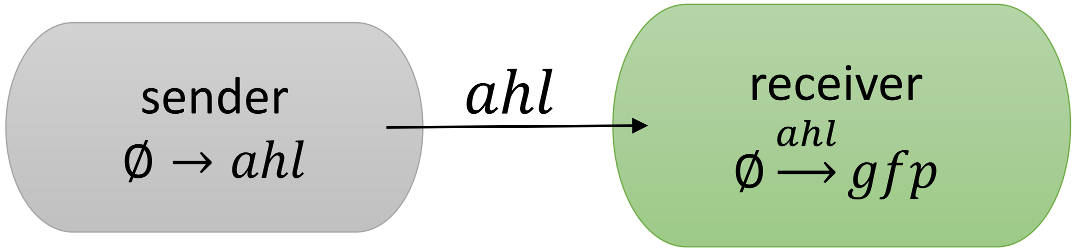
$$\Delta c_{i,j} = k_{diff}\big(0.5c_{i+1,j-1} + c_{i+1,j} + 0.5c_{i+1,j+1} + c_{i,j-1} + c_{i,j+1}$$



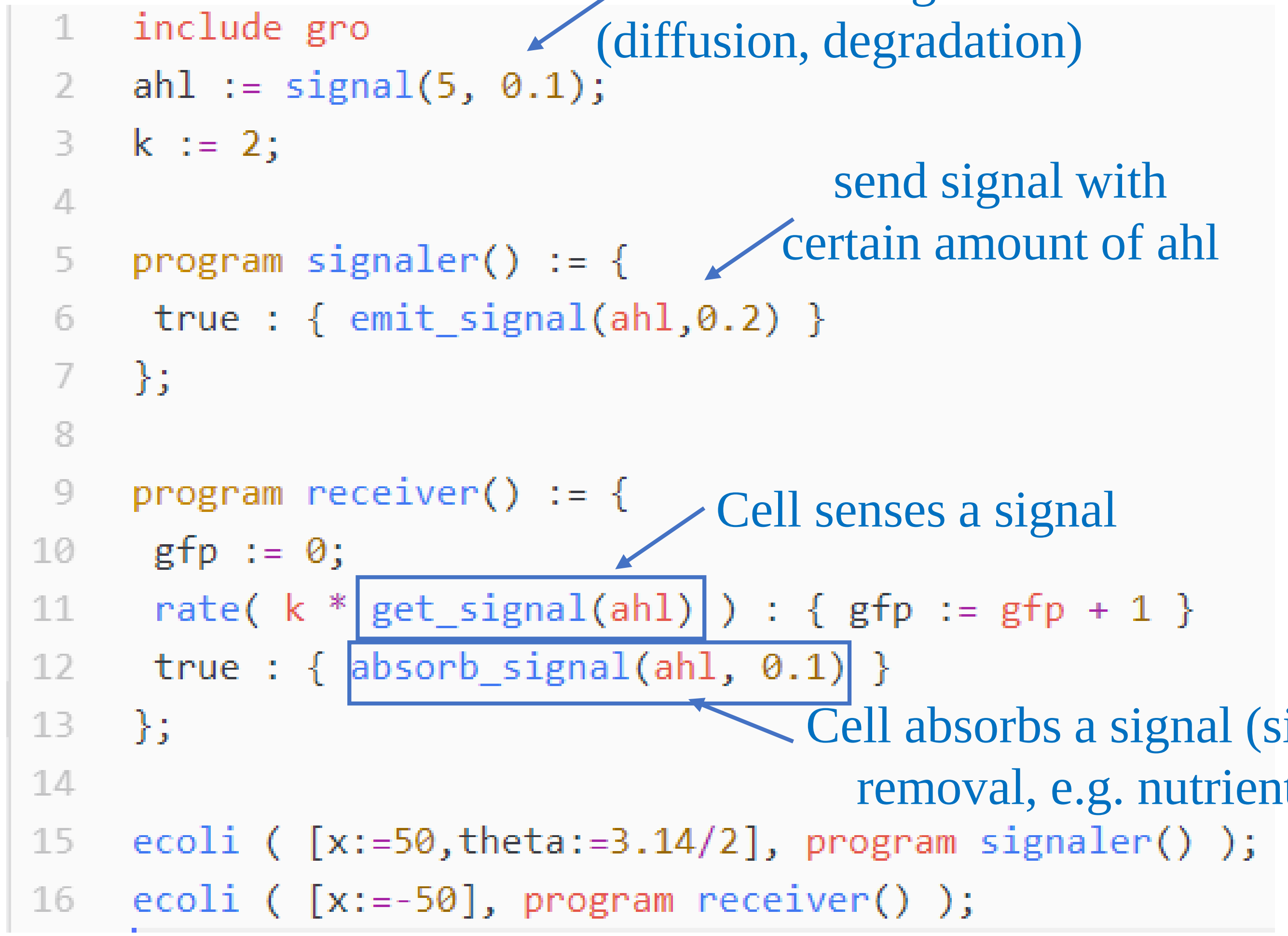
- Unspecified height, the units for signal concentration are proportional to moles / L.
- Euler integration: high diffusion rate of a signal → obvious numerical errors → decrease dt

gro programming: signals

- Sender - receiver



sender
$\emptyset \rightarrow ahl$
ahl
receiver
$\emptyset \xrightarrow{ahl} gfp$



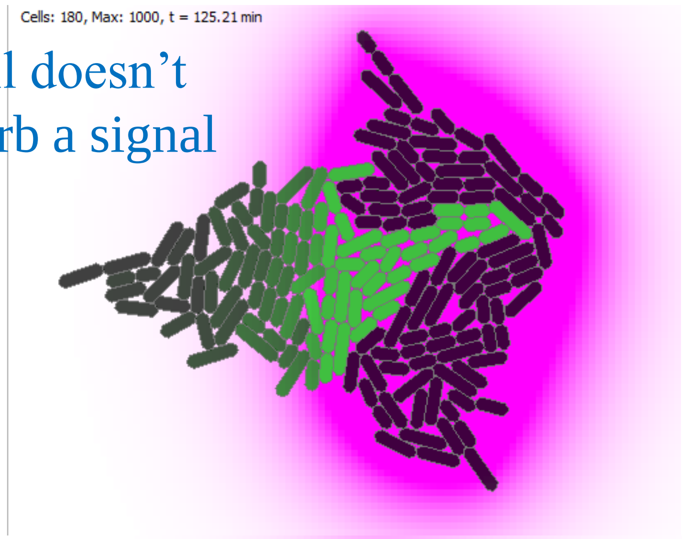
```
include gro
ahl := signal(5, 0.1);
k := 2;

program signaler() := {
 true : { emit_signal(ahl,0.2) }
};

program receiver() := {
 gfp := 0;
 rate( k * get_signal(ahl) ) : { gfp := gfp + 1 }
 true : { absorb_signal(ahl, 0.1) }
};

ecoli ( [x:=50,theta:=3.14/2], program signaler() );
ecoli ( [x:=-50], program receiver() );
```

declare a signal (diffusion, degradation)

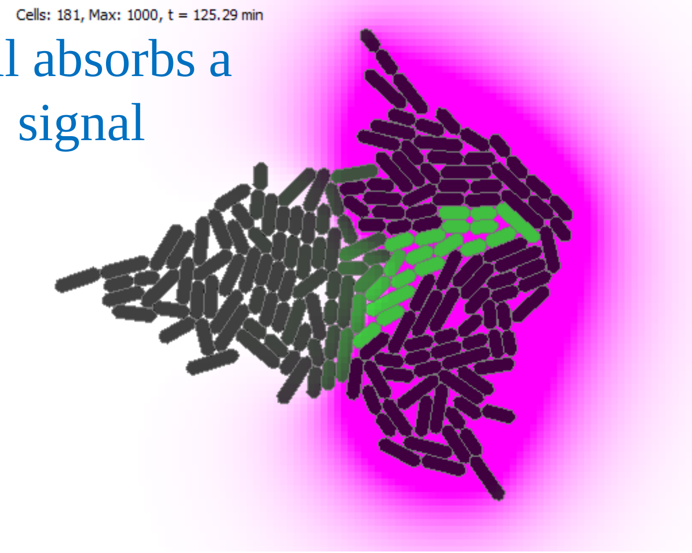
send signal with certain amount of ahl

Cell senses a signal

Cell absorbs a signal (signal removal, e.g. nutrients)



Cell doesn't absorb a signal

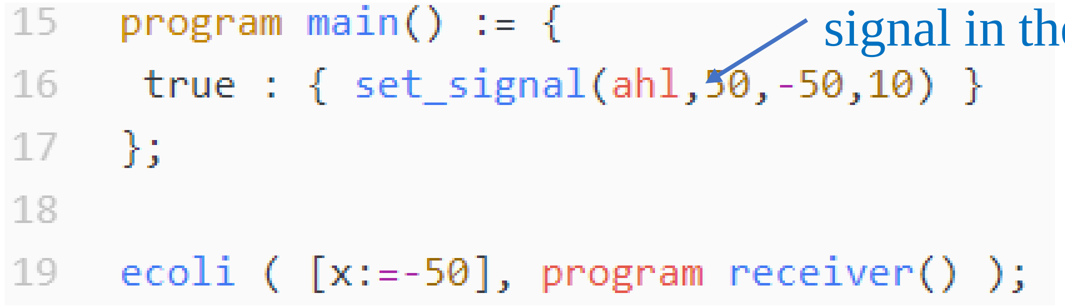


Cell absorbs a signal

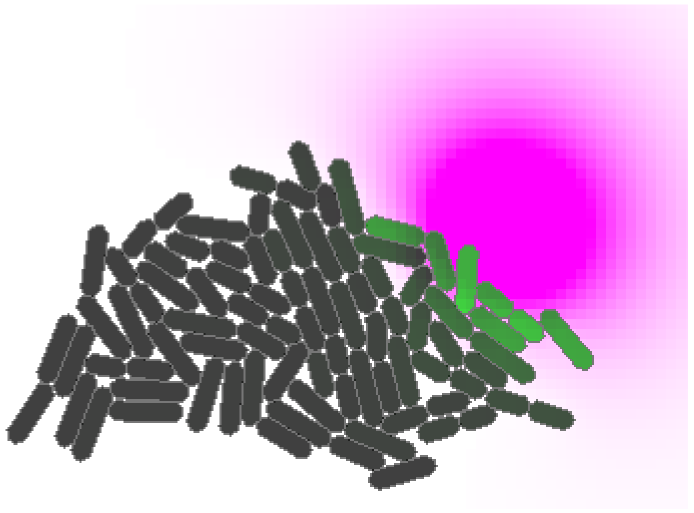
gro programming: signals

- Environment signals

position and amount of signal in the environment



```
program main() := {
 true : { set_signal(ahl,50,-50,10) }
};

ecoli ( [x:=-50], program receiver() );
```

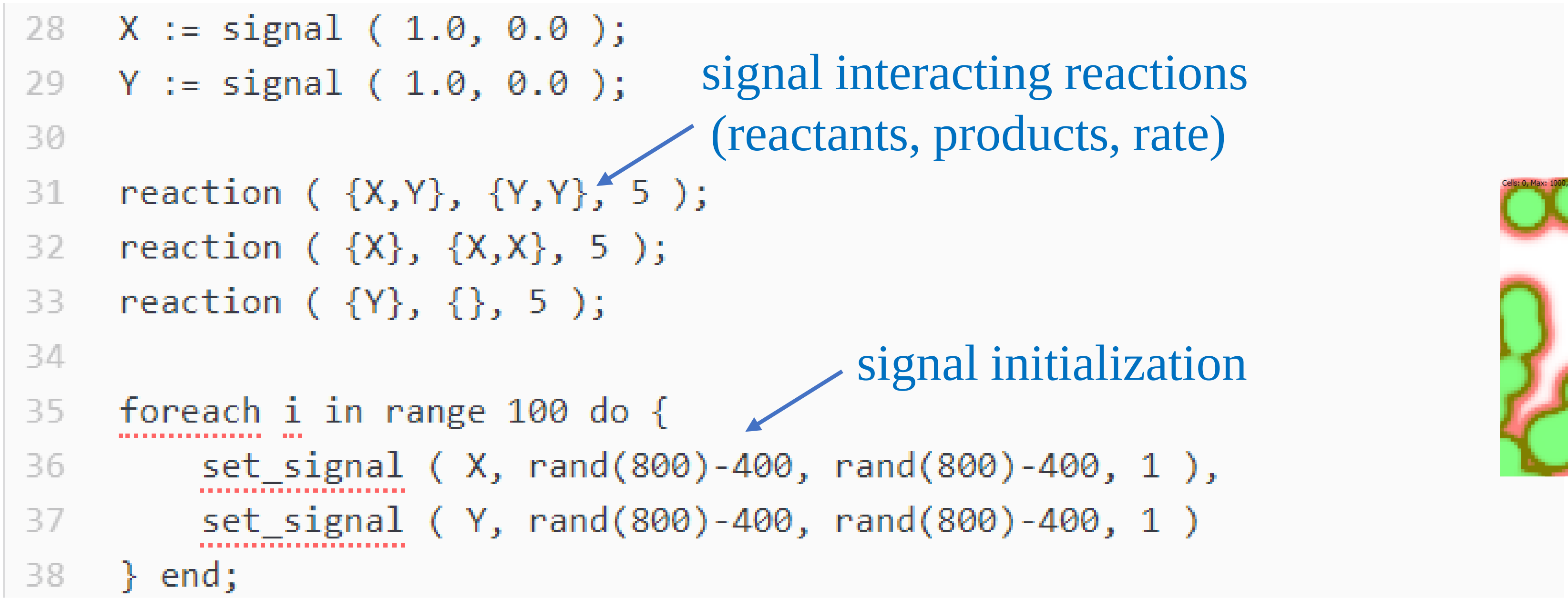


- Reaction-diffusion

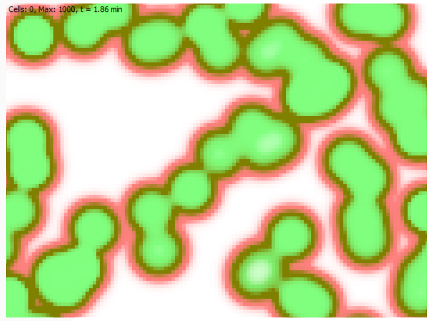
    chemicals that can (1) react with each other (or themselves) and (2) diffuse

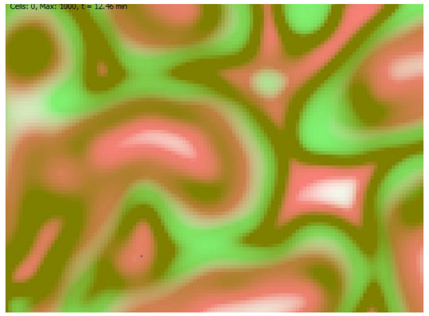
signal interacting reactions (reactants, products, rate)

signal initialization



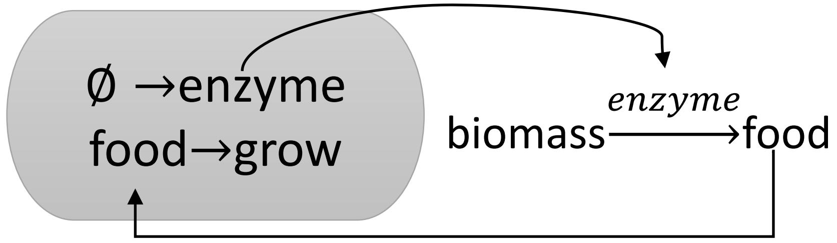
```
X := signal ( 1.0, 0.0 );
Y := signal ( 1.0, 0.0 );

reaction ( {X,Y}, {Y,Y}, 5 );
reaction ( {X}, {X,X}, 5 );
reaction ( {Y}, {}, 5 );

foreach i in range 100 do {
    set_signal ( X, rand(800)-400, rand(800)-400, 1 ),
    set_signal ( Y, rand(800)-400, rand(800)-400, 1 )
} end;
```

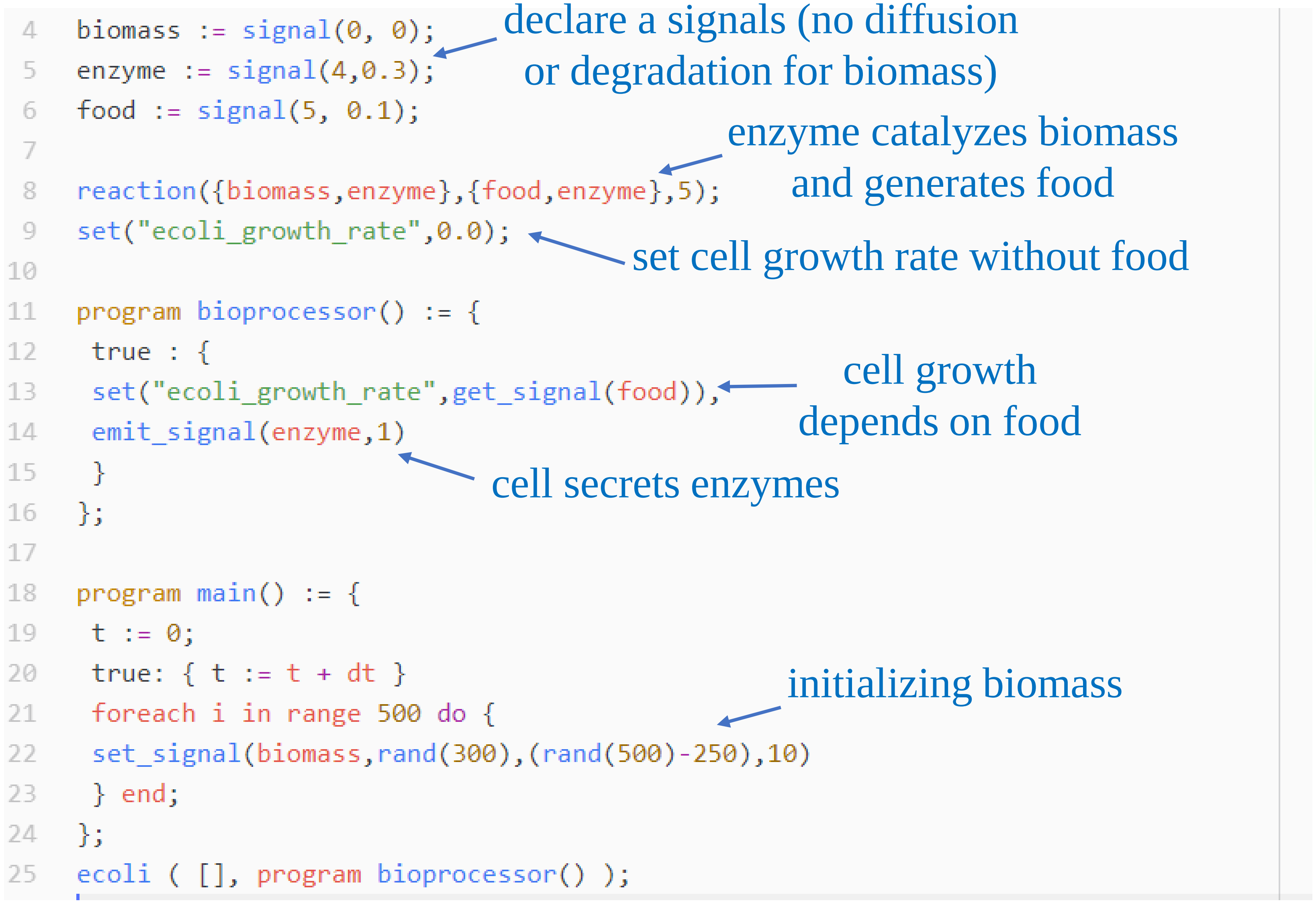




gro programming: signals

- Bioprocessing





```
biomass := signal(0, 0);
enzyme := signal(4,0.3);
food := signal(5, 0.1);

reaction({biomass,enzyme},{food,enzyme},5);
set("ecoli_growth_rate",0.0);

program bioprocessor() := {
 true : {
 set("ecoli_growth_rate",get_signal(food)),
 emit_signal(enzyme,1)
 }
};

program main() := {
 t := 0;
 true: { t := t + dt }
 foreach i in range 500 do {
 set_signal(biomass,rand(300),(rand(500)-250),10)
 } end;
};
ecoli ( [], program bioprocessor() );
```

declare a signals (no diffusion or degradation for biomass)

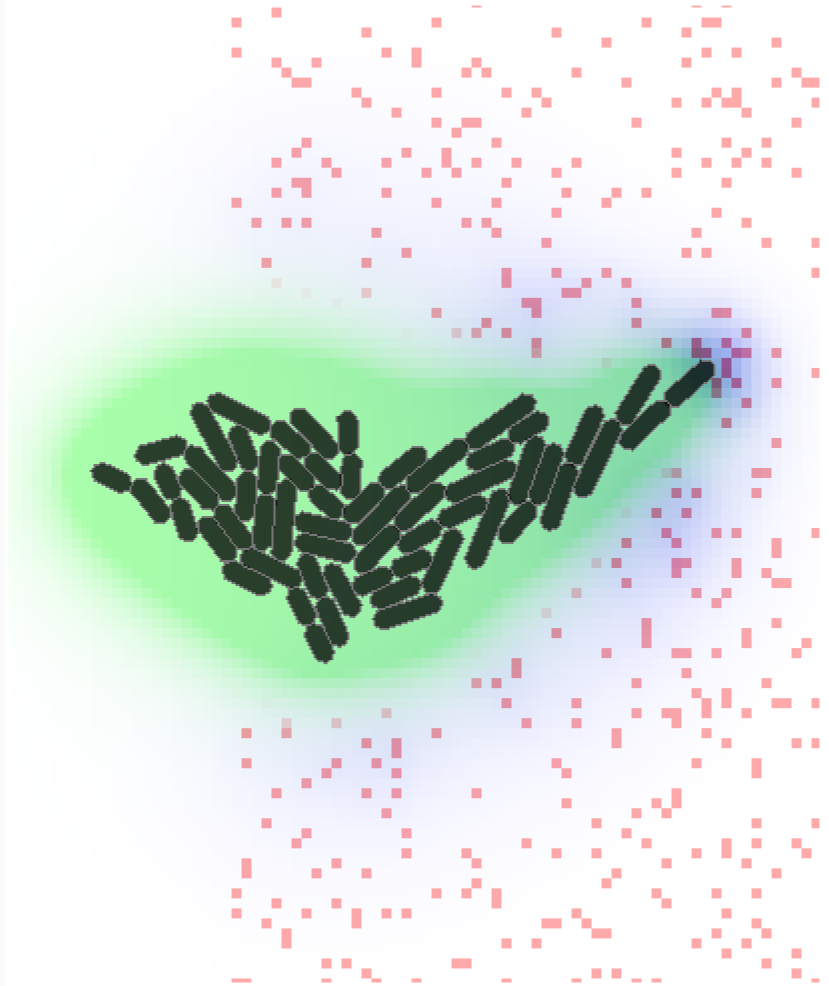
enzyme catalyzes biomass and generates food

set cell growth rate without food

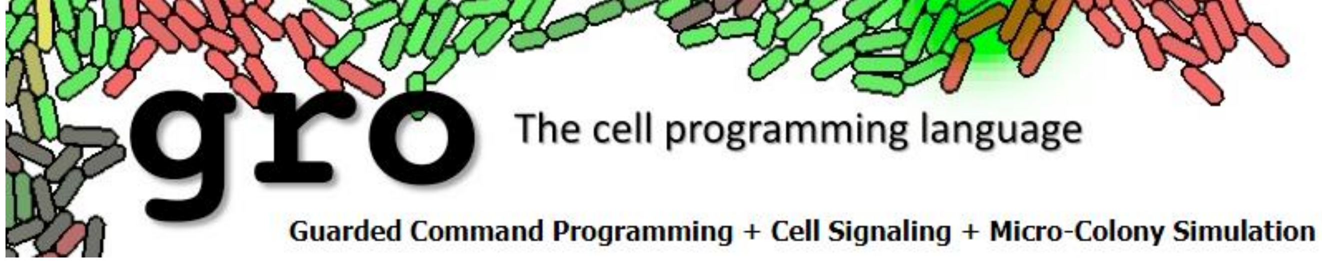
cell growth depends on food

cell secrets enzymes

initializing biomass



What is gro


gro
The cell programming language
Guarded Command Programming + Cell Signaling + Micro-Colony Simulation

A tool for the simulation of multicellular behaviors in a 2D environment.

- **Cell growth/division : try with bioscrape lineage**
- **Cell crowding**
- **Signal diffusion : try with PDE solver**
- **Molecular reactions (with noise) : try with bioscrape**

The environment shapes the microbial organisms.
Cells self-organize into certain patterns via signaling.

Try simulate the same circuit/system in well-mixed (bioscrape) VS spatial (gro).