Functional-Structural Plant Modelling with GroIMP and XL

Tutorial and Workshop at Agrocampus Ouest, Angers,
5-7 May, 2015

Winfried Kurth

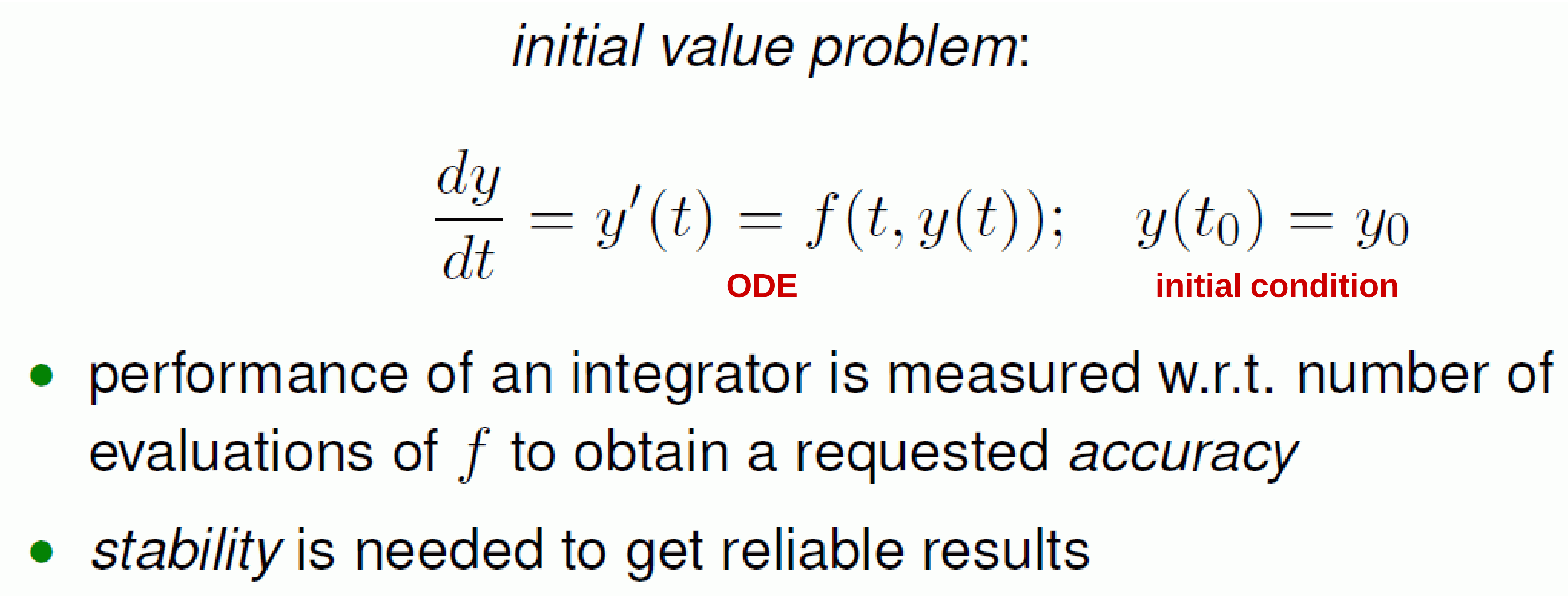
University of Göttingen,
Department Ecoinformatics, Biometrics and Forest Growth

Solving ordinary differential equations in XL

The Problem

- Development of structures often described by L-systems or graph grammars (in discrete time)
- functional parts often described by ordinary differential equations (ODEs) (in continuous time)
- examples: biosynthesis and transport of hormones, photosynthesis, carbon transport, xylem sap flow
- ODEs often not analytically solvable
- thus numerical solutions needed (numerical integrators)

mathematical formalism:



initial value problem:

$$\frac{dy}{dt} = y'(t) = f(t, y(t)); \quad y(t_0) = y_0$$

ODE

initial condition

- performance of an integrator is measured w.r.t. number of evaluations of f to obtain a requested *accuracy*
- *stability* is needed to get reliable results

mathematical formalism:

initial value problem:

$$\frac{dy}{dt} = y'(t) = f(t, y(t)); \quad y(t_0) = y_0$$

ODE initial condition

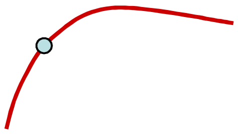
- performance of an integrator is measured w.r.t. number of evaluations of f to obtain a requested *accuracy*
- *stability* is needed to get reliable results

simplest discrete solution scheme:

Euler integrator

$$y_{n+1} = y_n + h \cdot f(t, y_n)$$

step size



mathematical formalism:

initial value problem:

$$\frac{dy}{dt} = \underset{\textbf{ODE}}{y'(t) = f(t, y(t))}; \quad \underset{\textbf{initial condition}}{y(t_0) = y_0}$$

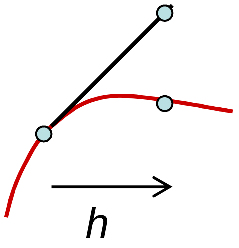
- performance of an integrator is measured w.r.t. number of evaluations of f to obtain a requested *accuracy*
- *stability* is needed to get reliable results

simplest discrete solution scheme:

Euler integrator

$$y_{n+1} = y_n + h \cdot f(t, y_n)$$

step size



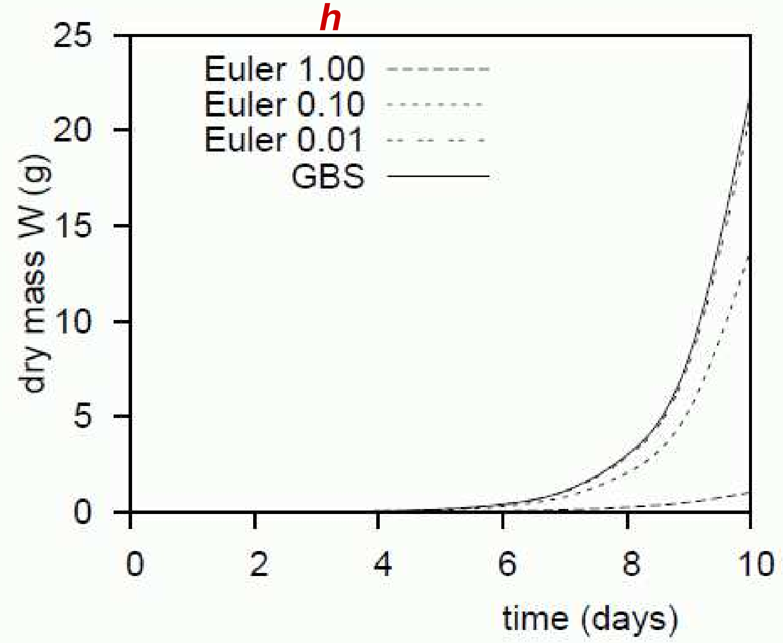
Reasons why people use Euler integration:

- simple and intuitive
- unintentionally
- unaware of the unsuitability of Euler integration
- unaware of other superior integration schemes

Problems with Euler integration: 2 examples

Exponential growth:

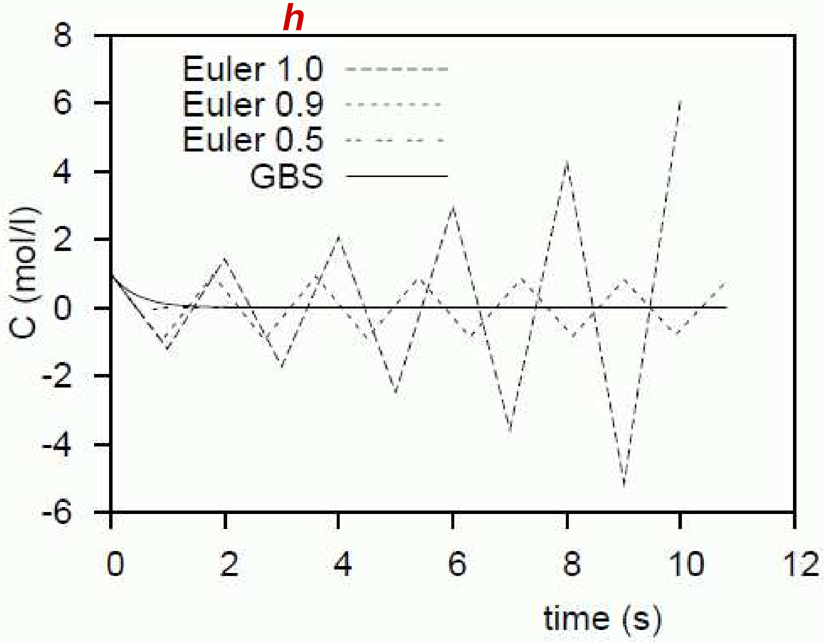
$$\frac{dW}{dt} = rW \qquad (r > 0)$$



→ inaccurate

Exponential decay:

$$\frac{dC}{dt} = -kC \qquad (k > 0)$$

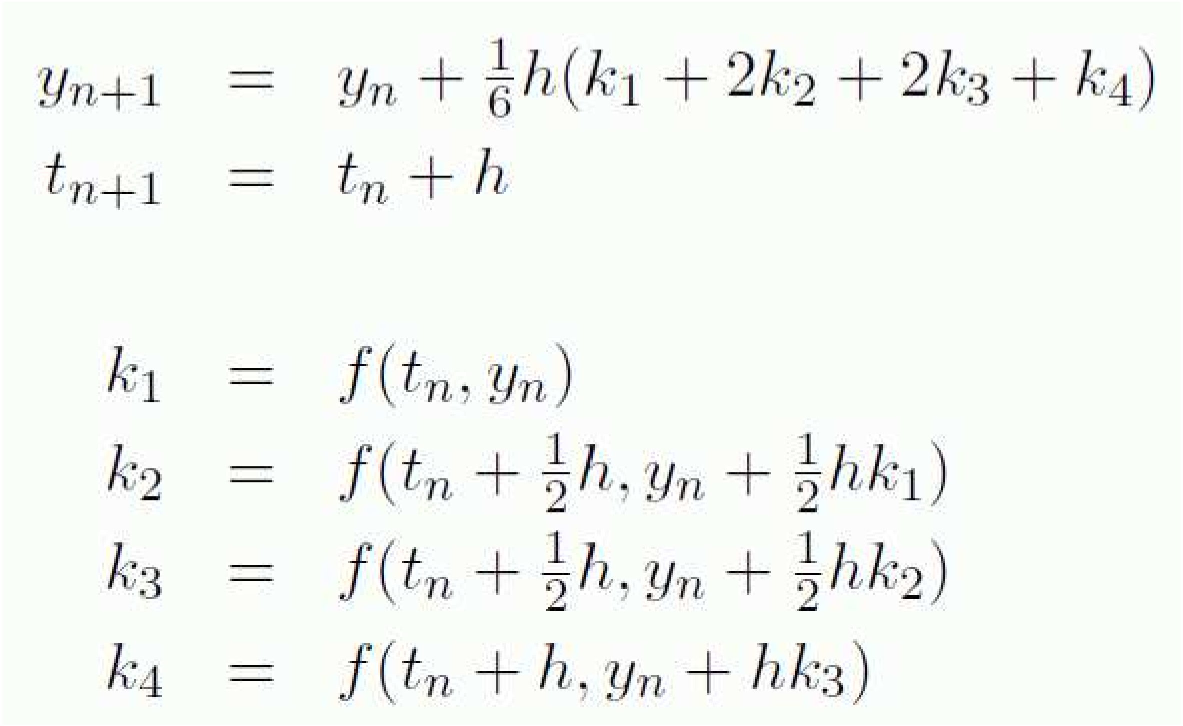


→ unstable

(Reference: GBS = Gragg-Bulirsch-Stoer integrator, a more accurate method)

Better integration methods exist –

for example: the Runge-Kutta method

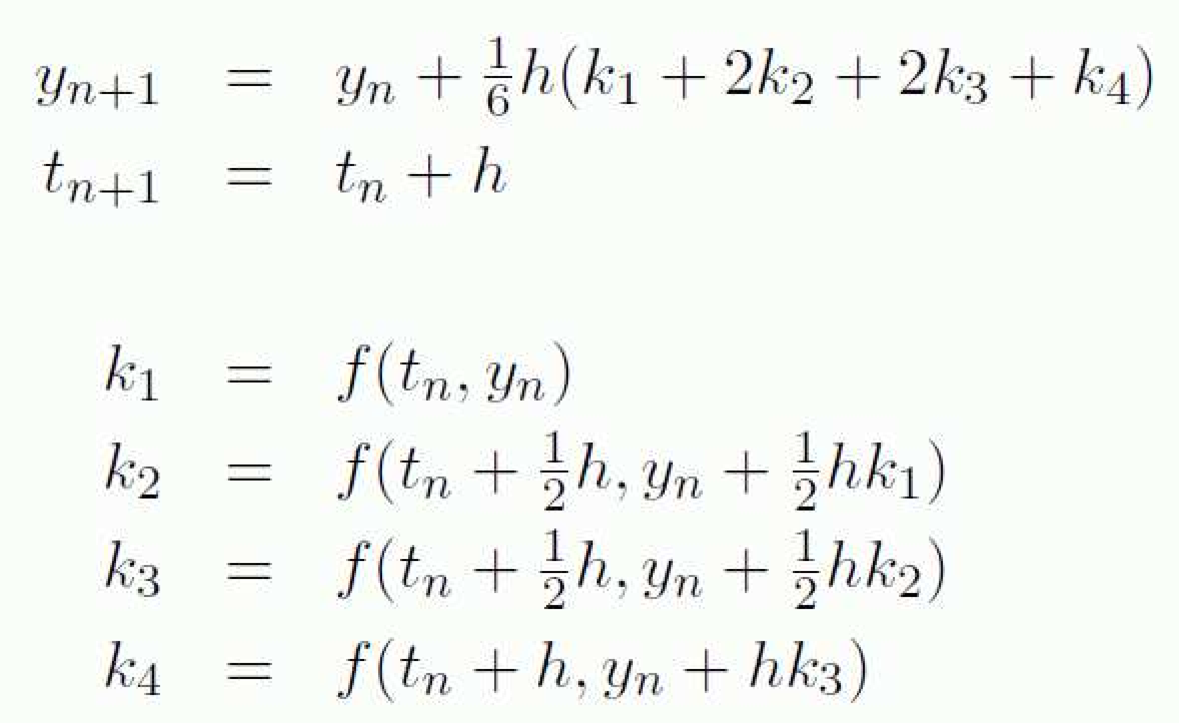


$$\begin{aligned} y_{n+1} &= y_n + \tfrac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \\ t_{n+1} &= t_n + h \end{aligned}$$

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hk_1) \\ k_3 &= f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hk_2) \\ k_4 &= f(t_n + h, y_n + hk_3) \end{aligned}$$

Better integration methods exist –

for example: the Runge-Kutta method



$$\begin{aligned}
y_{n+1} &= y_n + \tfrac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \\
t_{n+1} &= t_n + h \\
\\
k_1 &= f(t_n, y_n) \\
k_2 &= f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hk_1) \\
k_3 &= f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hk_2) \\
k_4 &= f(t_n + h, y_n + hk_3)
\end{aligned}$$

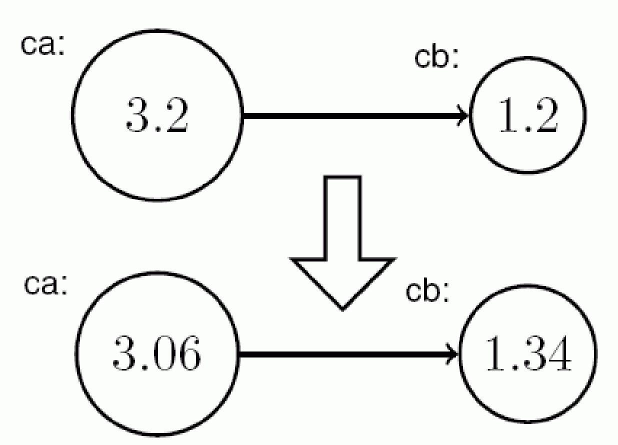
But: this requires much efforts to implement it in Java or XL within a plant model

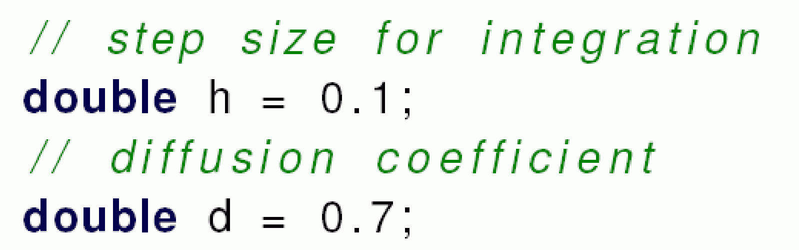
An example occurring in plant models

Model of diffusion

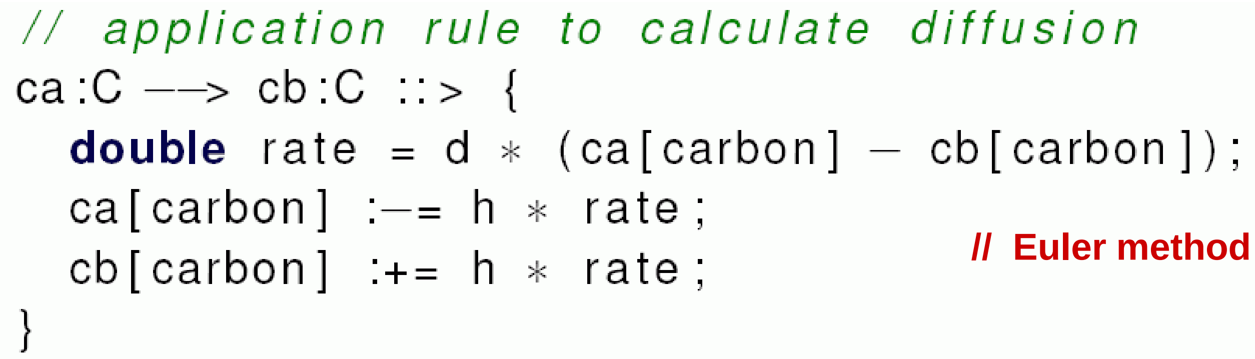
version without rate assigment operator

$$\frac{d[\text{carbon}]}{dt} = d \cdot \Delta[\text{carbon}]$$





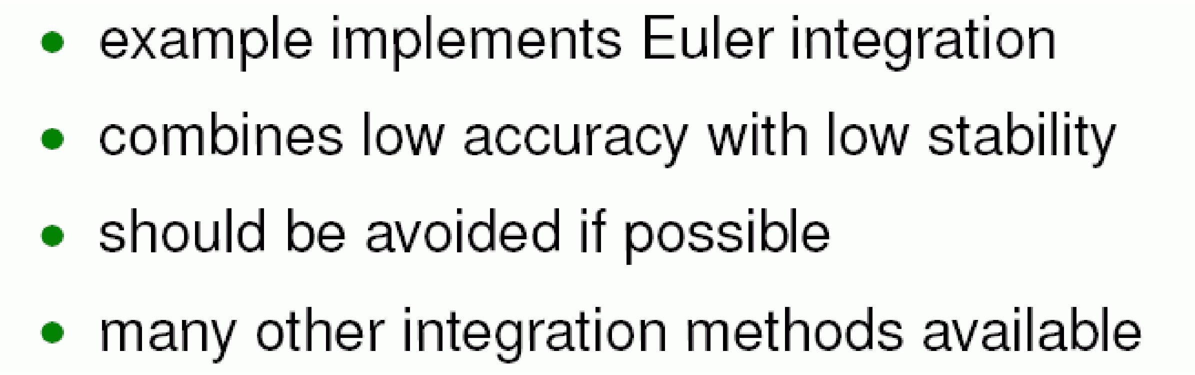
```
// step size for integration
double h = 0.1;
// diffusion coefficient
double d = 0.7;
```



```
// application rule to calculate diffusion
ca:C --> cb:C ::> {
  double rate = d * (ca[carbon] - cb[carbon]);
  ca[carbon] :-= h * rate;
  cb[carbon] :+= h * rate;
}
```

// Euler method

(Hemmerling 2010)



- example implements Euler integration
- combines low accuracy with low stability
- should be avoided if possible
- many other integration methods available

The rate assignment operator

Syntax in XL:

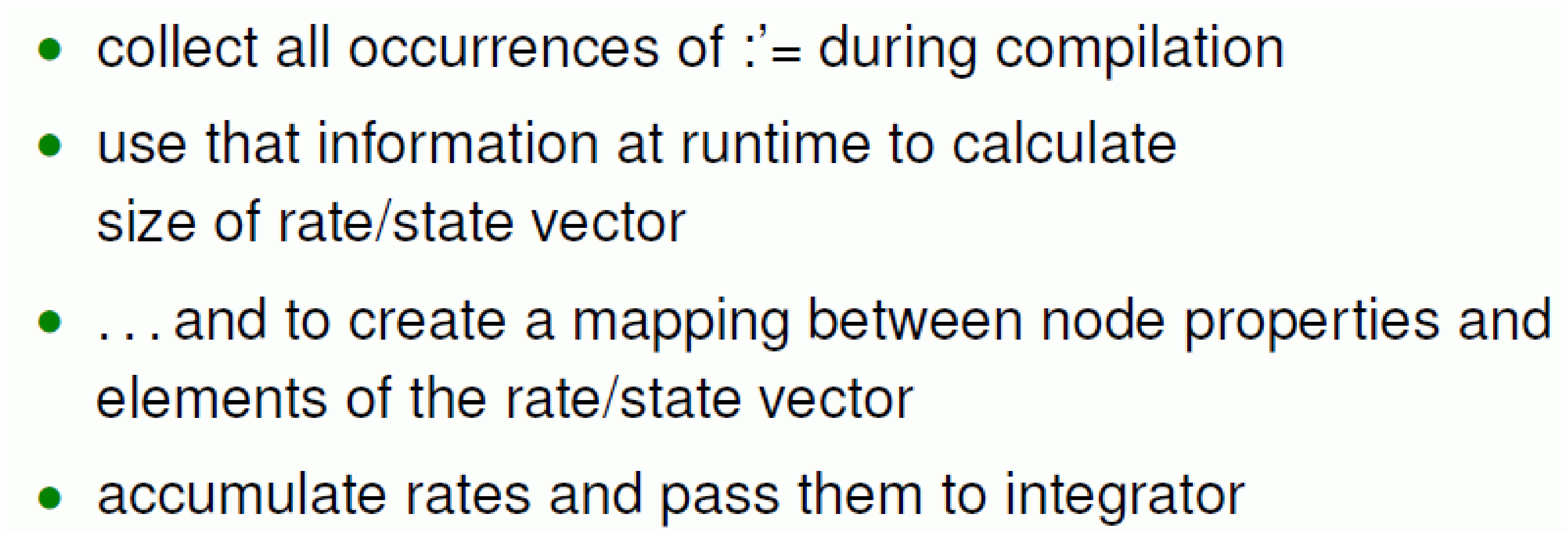
node_type[*attribute_name*] **:‘=** *value*

example:

```
c:C ::> { c[carbon] :‘= productionRate; }
```

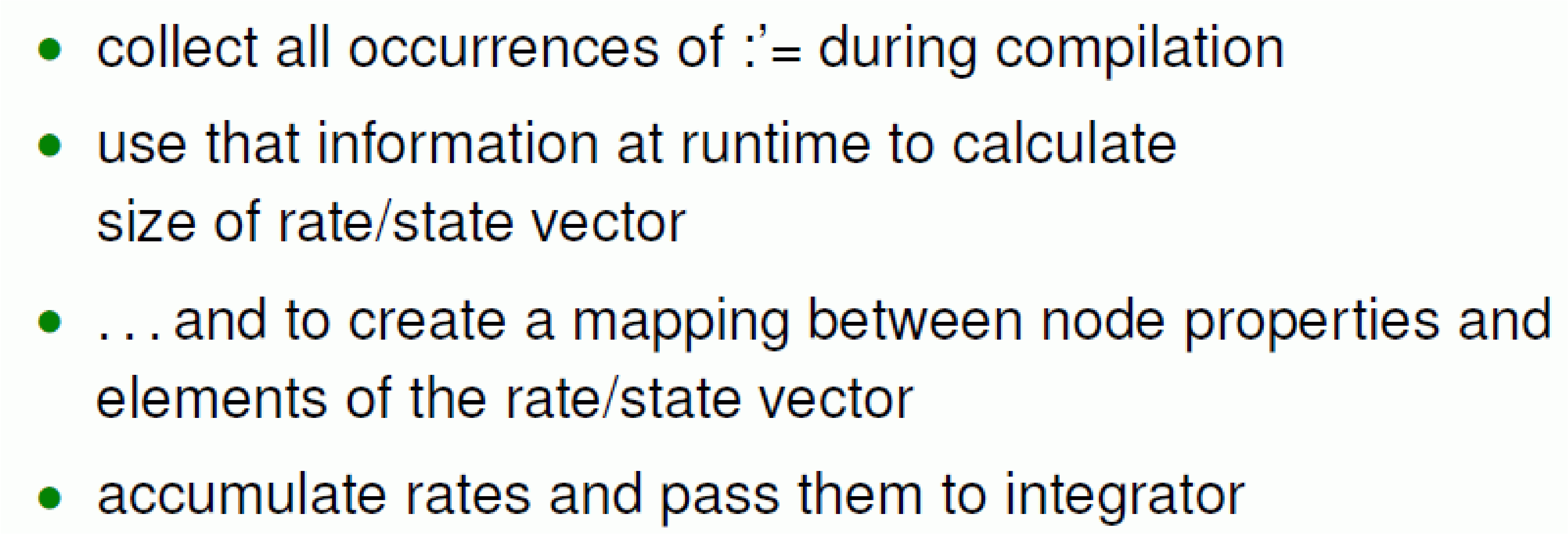
(Hemmerling 2010)

What does the operator in the background?



- collect all occurrences of :’= during compilation
- use that information at runtime to calculate size of rate/state vector
- . . . and to create a mapping between node properties and elements of the rate/state vector
- accumulate rates and pass them to integrator

What does the operator in the background?



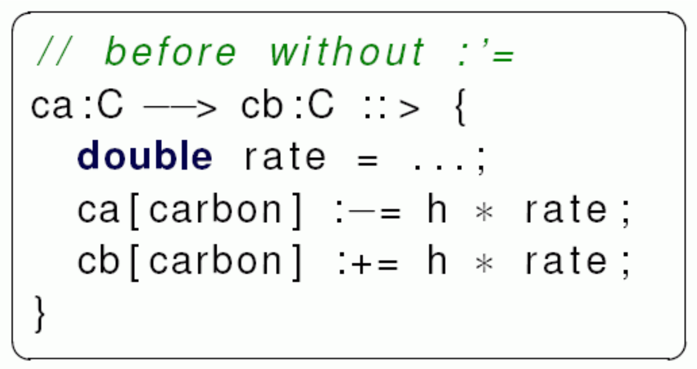
- collect all occurrences of :’= during compilation
- use that information at runtime to calculate size of rate/state vector
- . . . and to create a mapping between node properties and elements of the rate/state vector
- accumulate rates and pass them to integrator

The integrator itself is not fixed.

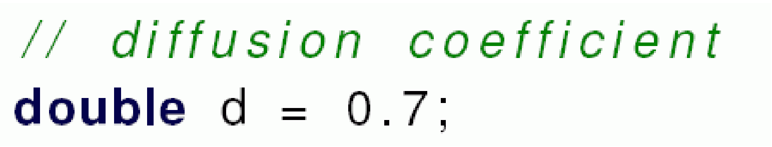
It can be chosen by the user from numerics libraries: e.g.,

```
setSolver(new org.apache.commons.math.ode.nonstiff.AdamsBashforthIntegrator
(3, 0, 1, 1E-4, 1E-4));
```

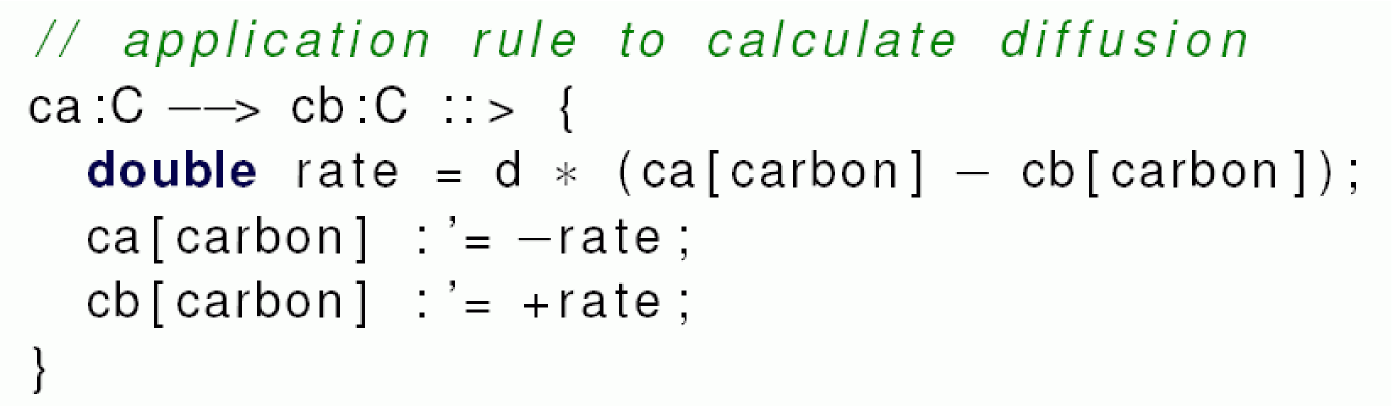
The diffusion example again, with rate assignment operator:



```
// before without :'=
ca:C --> cb:C ::> {
  double rate = ...;
  ca[carbon] :-= h * rate;
  cb[carbon] :+= h * rate;
}
```



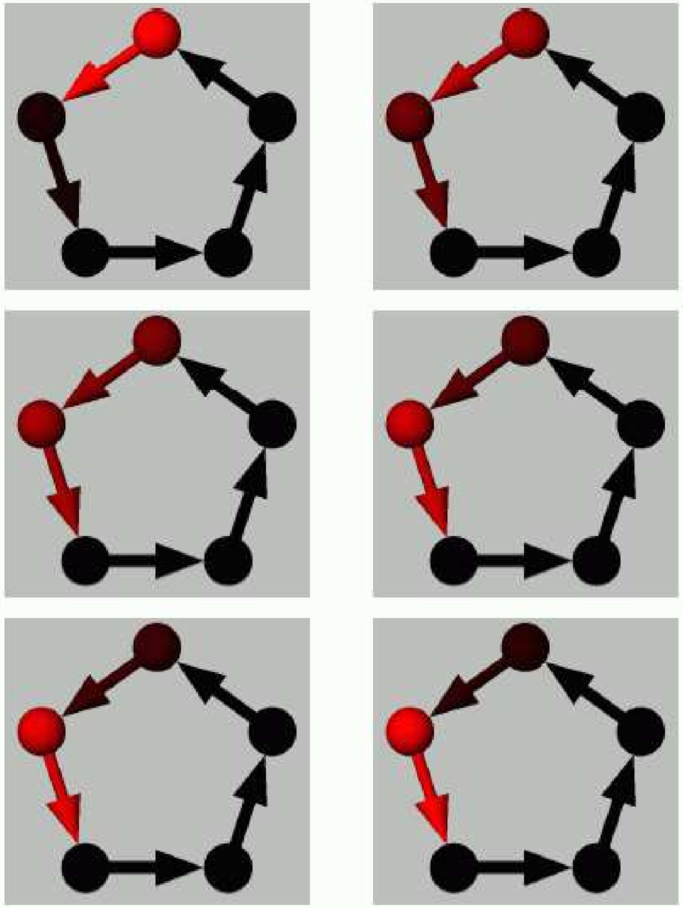
```
// diffusion coefficient
double d = 0.7;
```



```
// application rule to calculate diffusion
ca:C --> cb:C ::> {
  double rate = d * (ca[carbon] - cb[carbon]);
  ca[carbon] :'= -rate;
  cb[carbon] :'= +rate;
}
```

(Hemmerling 2010)

another example:



```
protected void getRate()
[
  x:S –EDGE_0–> y:S –EDGE_0–> z:S ::> {
    float rate = x[c] > 0.001 ? 0 : 0.4 * y[c];
    y[c] :'= –rate;
    z[c] :'= +rate;
  }
]
```

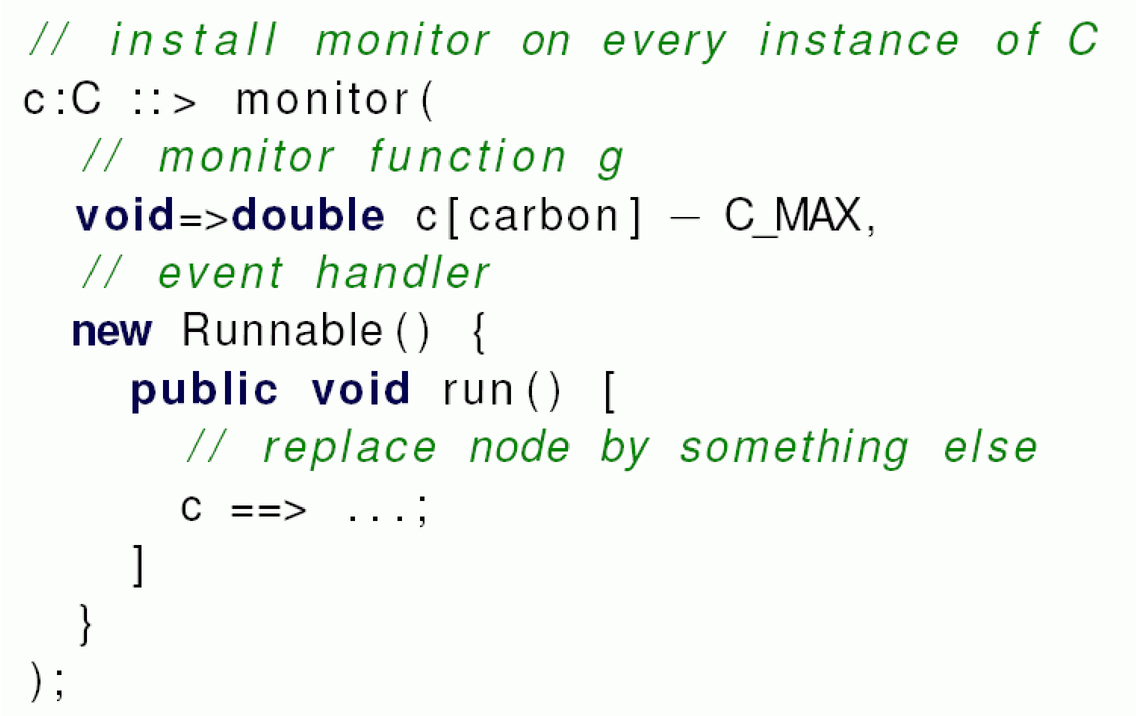
(Hemmerling 2010)

extension by the use of monitor functions:

- e.g., to plot data about the state in regular intervals

- or to stop integration once a condition is fulfilled

A monitor function maps the states to real numbers. Root finding algorithms are used to find its zeros, i.e., exact event time



```
// install monitor on every instance of C
c:C ::> monitor(
  // monitor function g
  void=>double c[carbon] - C_MAX,
  // event handler
  new Runnable() {
    public void run() [
      // replace node by something else
      c ==> ...;
    ]
  }
);
```

(Hemmerling 2010)

Example **simpleode.rgg**: Declarations

```
const double uRate = 0.1;
const double vRate = 0.2;
const double wRate = 1;
const double threshold = 10.0;
const double periodLength = 1.0;

/* growing structure with several variables which are controlled
   by ODEs: */

module C(double len) extends Cylinder(len, 0.05)
   {
   double u = 1;
   double v = 0;
   double w1 = 0;
   double w2 = 1;
   };


/* stable structure which is not influenced by ODEs: */

module S(double len) extends Cylinder(len, 0.05);

double time;

const DatasetRef diagram = new DatasetRef("function plot");
```

Initializations:

```
protected void init()
[
Axiom ==> C(1);
   {
   time = 0;
   /* optionally, some preferred ODE solver can be specified: */
    // setSolver(new org.apache.commons.math.ode.nonstiff.EulerIntegrator(0.1));
    // setSolver(new org.apache.commons.math.ode.nonstiff.ClassicalRungeKuttaIntegrator(0.1));
    // setSolver(new org.apache.commons.math.ode.nonstiff.GraggBulirschStoerIntegrator(0, 0.01, 1E-4, 1E-4));
    // setSolver(new org.apache.commons.math.ode.nonstiff.AdamsBashforthIntegrator(3, 0, 1, 1E-4, 1E-4));
    // setSolver(new org.apache.commons.math.ode.nonstiff.DormandPrince54Integrator(0, 1, 1E-4, 1E-4));

   diagram.clear();
   chart(diagram, XY_PLOT);
   }
]
```

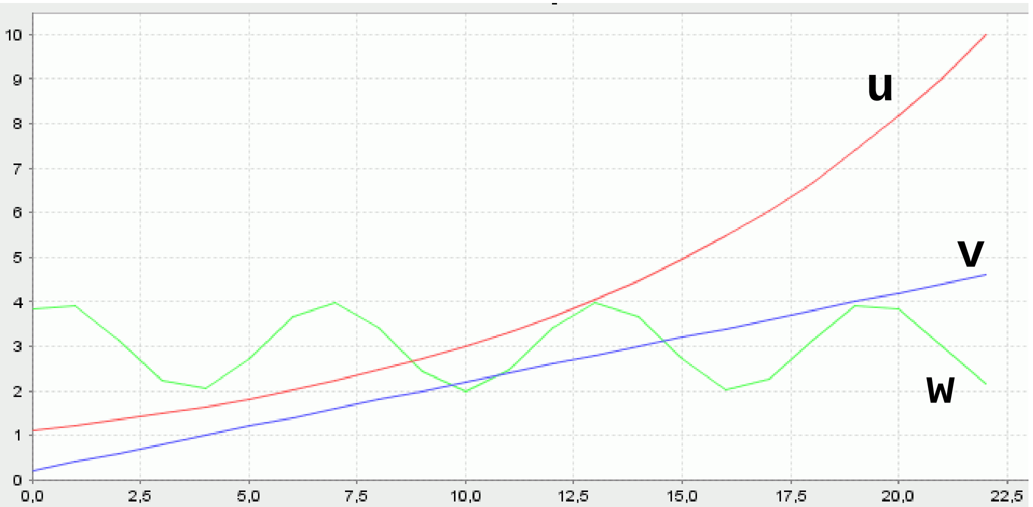
The central part: rate assignment

```
protected void getRate()
   [
   { time :'= 1; }

   /* apply differential increments to the variables of the C nodes.
      ODE for u:  u'(t) = uRate * u(t)  ( => solution u = exp t)
      ODE for v:  v'(t) = vRate         ( => solution v = c*t  )
      ODE for w1: w''(t) = -w(t)        ( => solution w = cos t) */

   c:C ::> {
           c[u]  :'= uRate * c[u];
           c[v]  :'= vRate;
           c[w1] :'= wRate * c[w2];
           c[w2] :'= -wRate * c[w1];
           }
   ]
```

plotted diagram after 1 step:



Translation to 3-d structure
and step control by monitor functions:

```
public void develop()
   [
   /* set monitor to stop integration when variable u reaches
      threshold value and to trigger structural changes: */
   a:C ::> monitor(void=>double a[u] - threshold, new Runnable() {
              public void run() [
                 a ==> s:S RU(10) M(-1) c:C(1)
                    {
                    c[u] = 1;
                    c[v] = 0;
                    c[w1] = 0;
                    c[w2] = 1;
                    s[length] = a[u];
                    s[radius] = 3 + a[w1];
                    println("stopped!");
                    };
              ]
           });
```

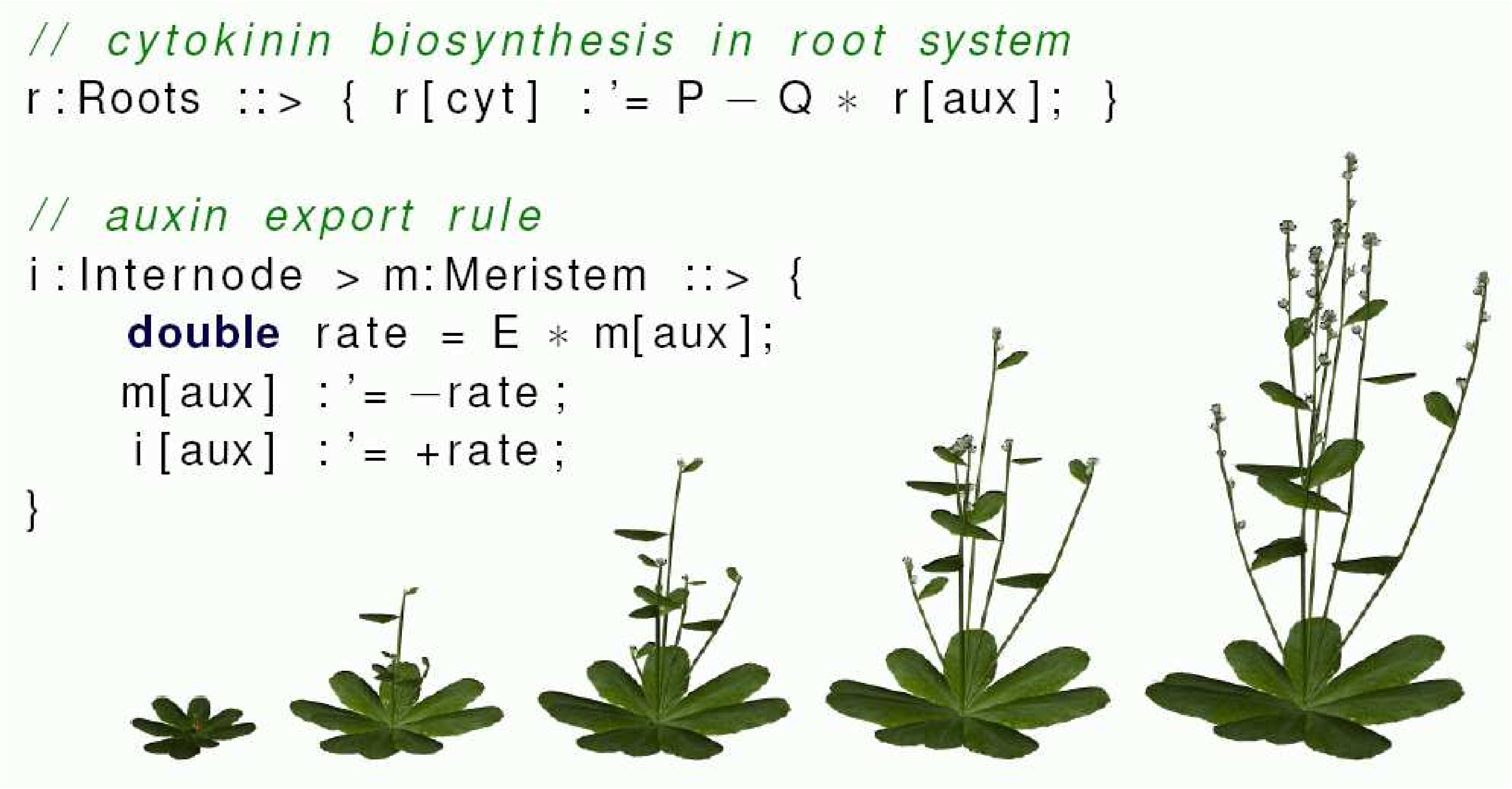
Translation to 3-d structure and step control by monitor functions *(continued)*:

```
/* perform integration
   and trigger visualization and plotting periodically: */
       {
       println("<");

       /* visualize current state in regular intervals: */
       monitorPeriodic(periodLength, new Runnable() {
          public void run() {
             print(".");
             [
             c:C ::> {
                c[length] = c[u];
                c[radius] = 3 + c[w1];
                diagram.addRow().set(0, c[u]).set(1, c[v]).set(2, 3+c[w1]);
                }
             ]
             derive();  /* necessary here for update! */
             }
          });
       integrate();
       println("time = " + time);
       }
  ]
```

see **`simpleode.rgg`**

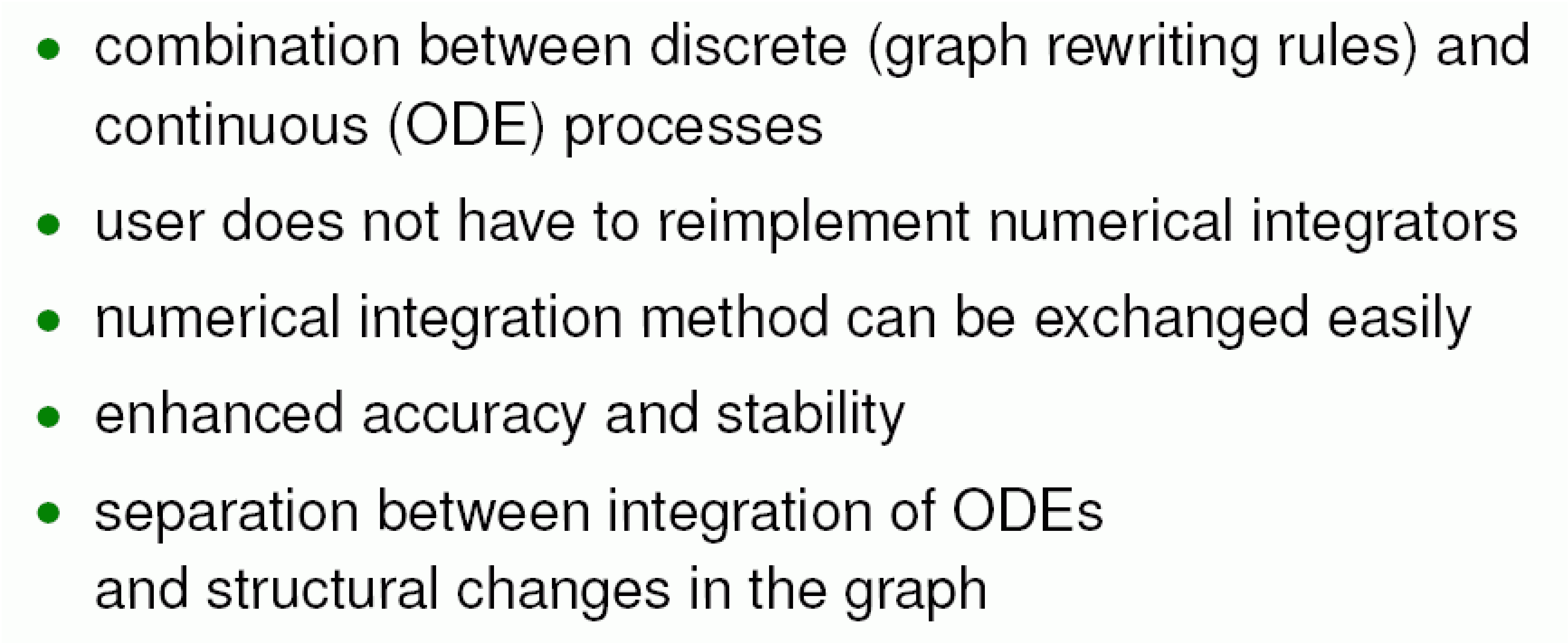
Arabidopsis example (from Hemmerling & Evers 2010):



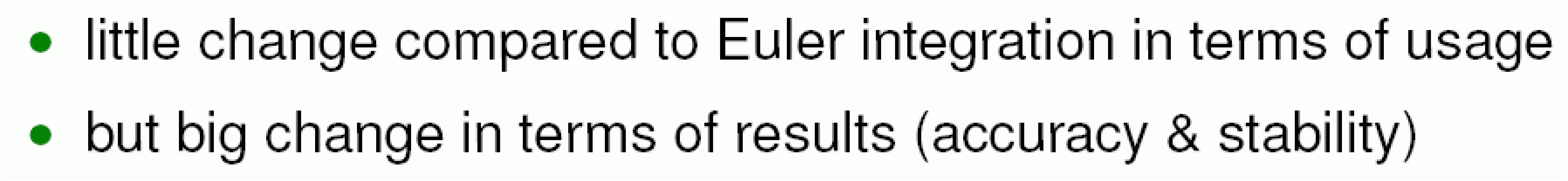
```
// cytokinin biosynthesis in root system
r:Roots ::> { r[cyt] :'= P - Q * r[aux]; }

// auxin export rule
i:Internode > m:Meristem ::> {
    double rate = E * m[aux];
    m[aux] :'= -rate;
    i[aux] :'= +rate;
}
```

rate assignment operator / conclusion:



- combination between discrete (graph rewriting rules) and continuous (ODE) processes
- user does not have to reimplement numerical integrators
- numerical integration method can be exchanged easily
- enhanced accuracy and stability
- separation between integration of ODEs and structural changes in the graph



- little change compared to Euler integration in terms of usage
- but big change in terms of results (accuracy & stability)

(Hemmerling 2010)