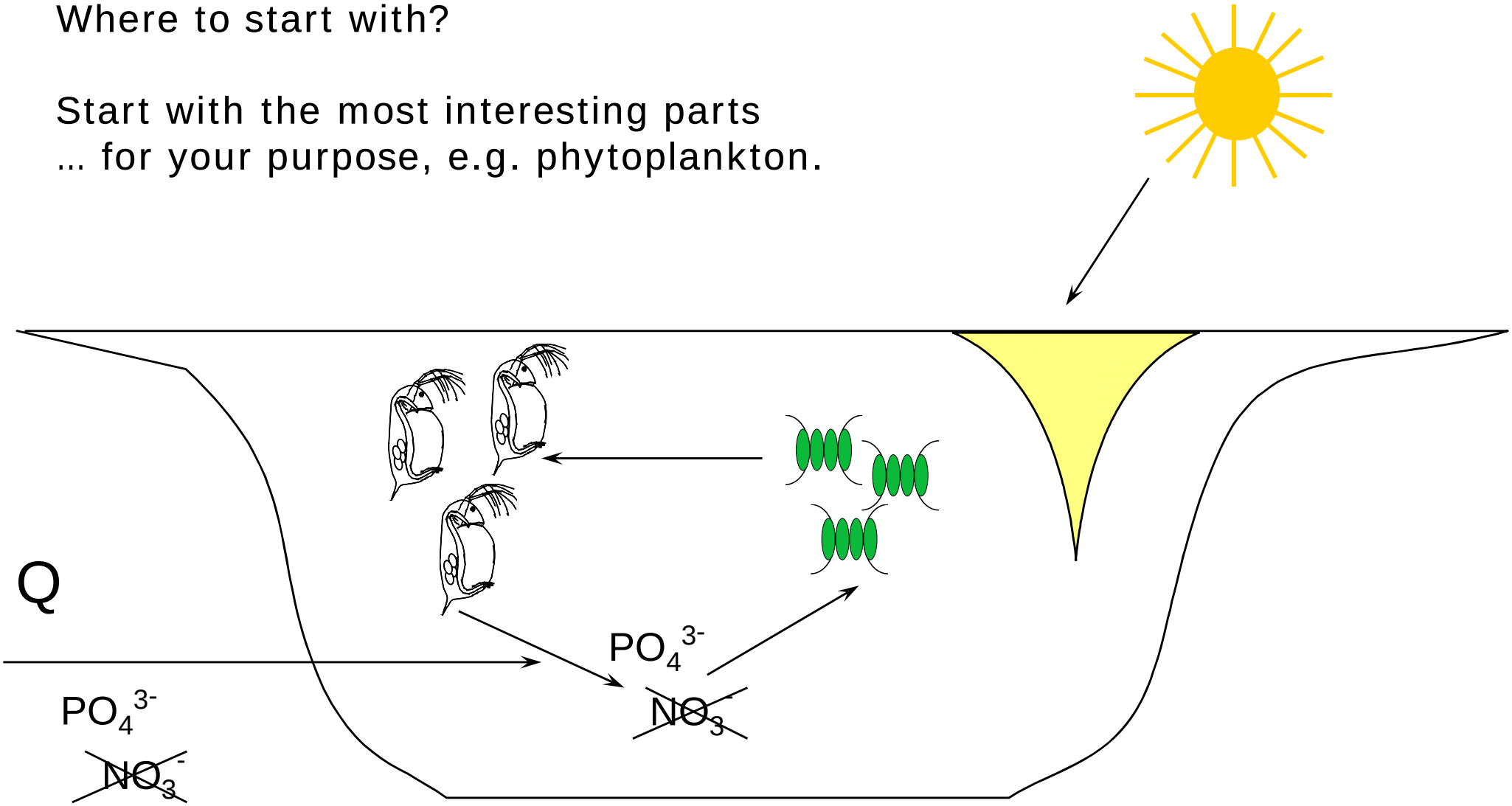
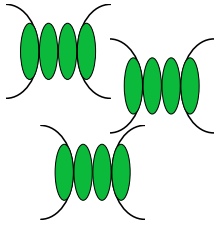
A Model of an ecosystem, e.g., a lake

Where to start with?

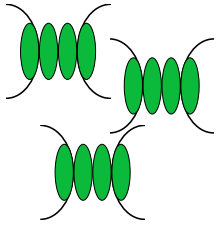
Start with the most interesting parts
... for your purpose, e.g. phytoplankton.



Phytoplankton



Mass balance of phytoplankton

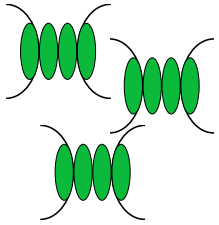


$$N_{today} = N_{yesterday} + growth - death$$

with :

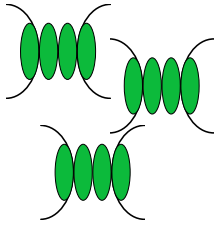
N = Abundance

Mass balance of phytoplankton



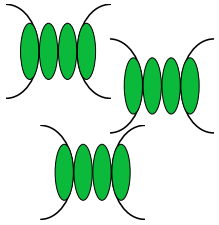
$$\frac{N_t - N_{t-\Delta t}}{\Delta t} = growth - death$$

Mass balance of phytoplankton



$$\frac{dN}{dt} = growth - death$$

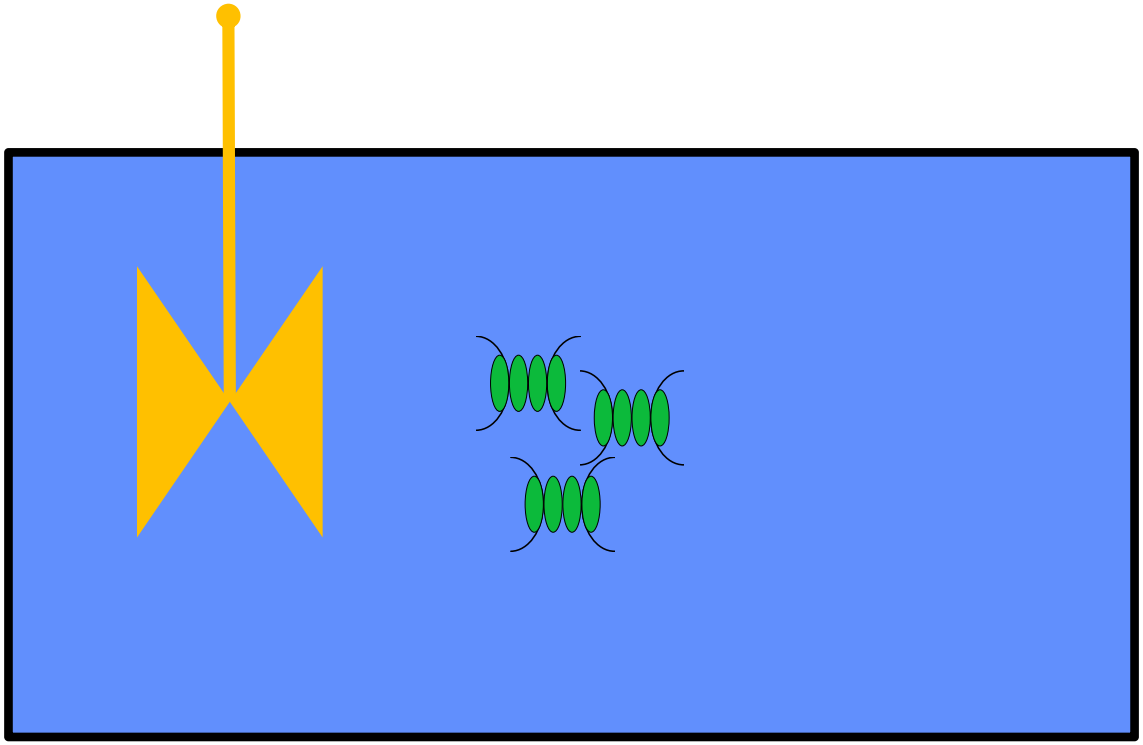
Mass balance of phytoplankton



$$\frac{dN}{dt} = growth - death$$

But what is N? N in a culture, a lake, the world???

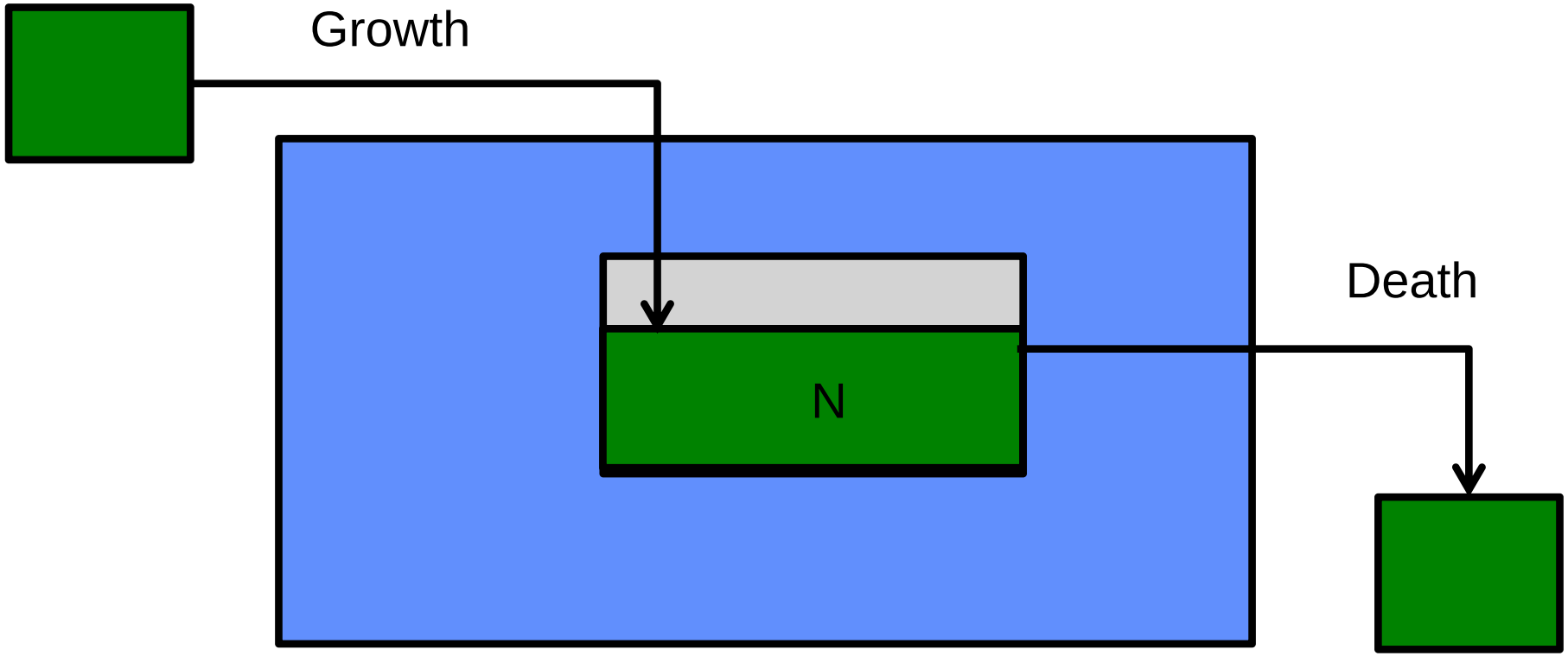
A batch culture



$$\frac{dN}{dt} = growth - death$$

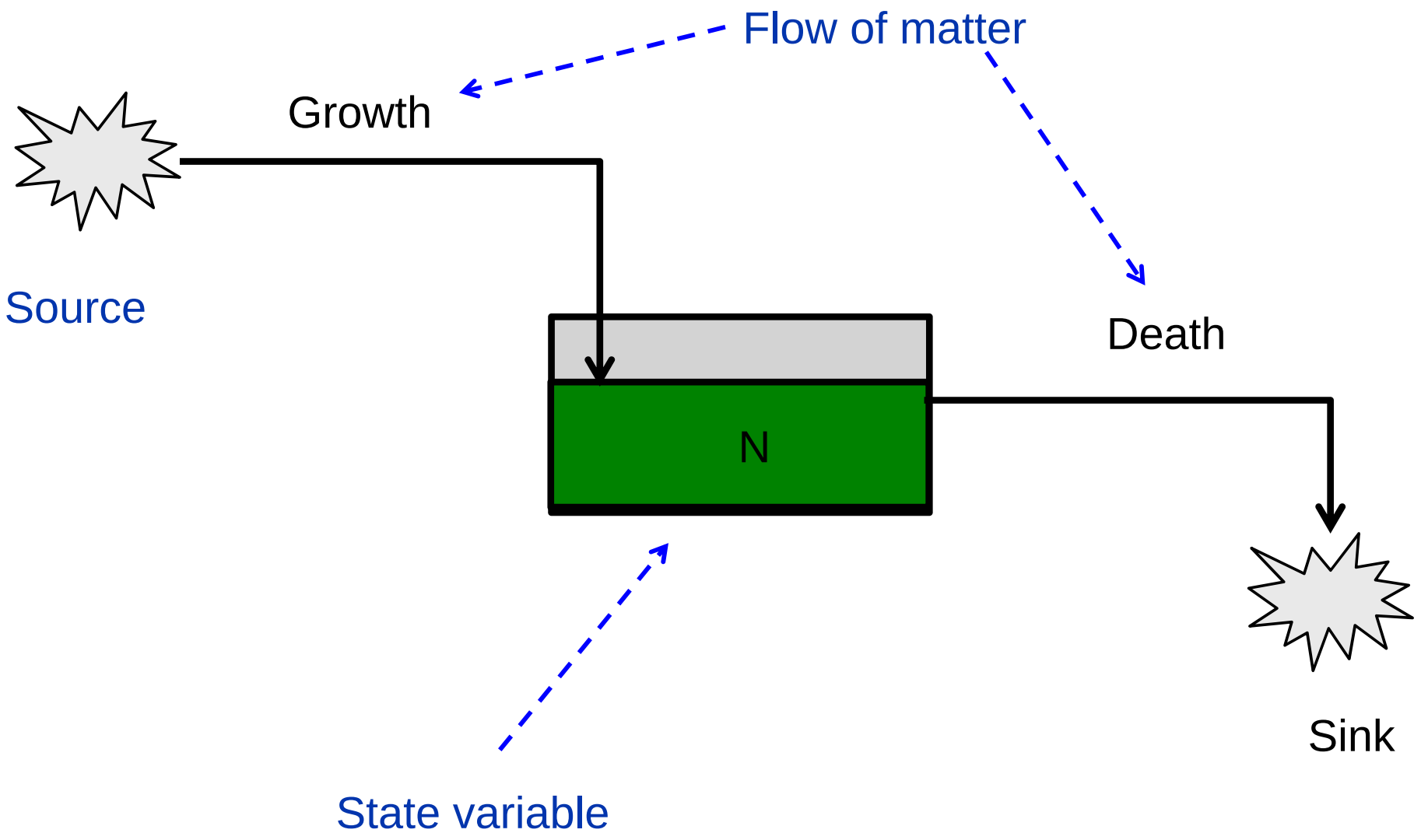
N is abundance per volume (e.g., per litre)

Not only that Phytoplankton is in a pool
Phytoplankton itself is a pool!



$$\frac{dN}{dt} = growth - death$$

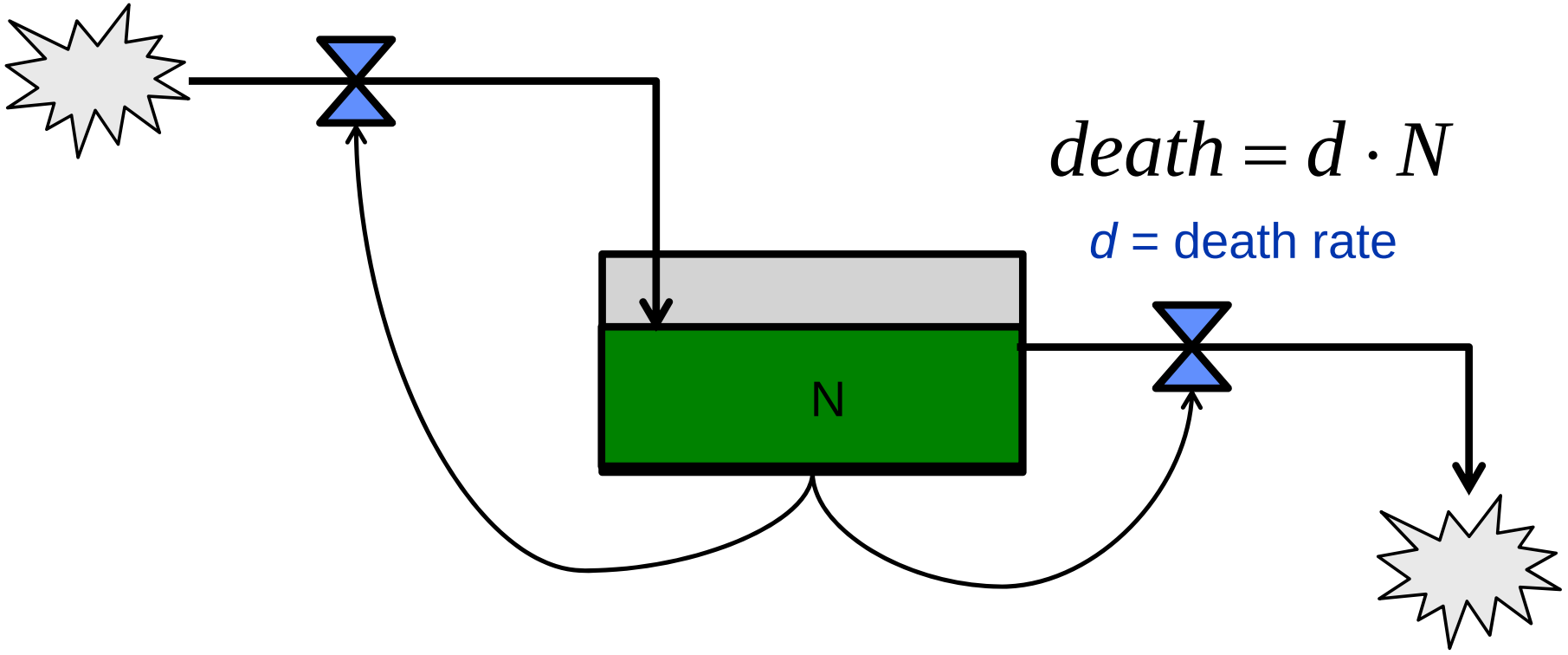
Source, sink, state and flow



Phytoplankton divides itself!
A part of the cells die.

$$growth = b \cdot N$$

b = birth rate

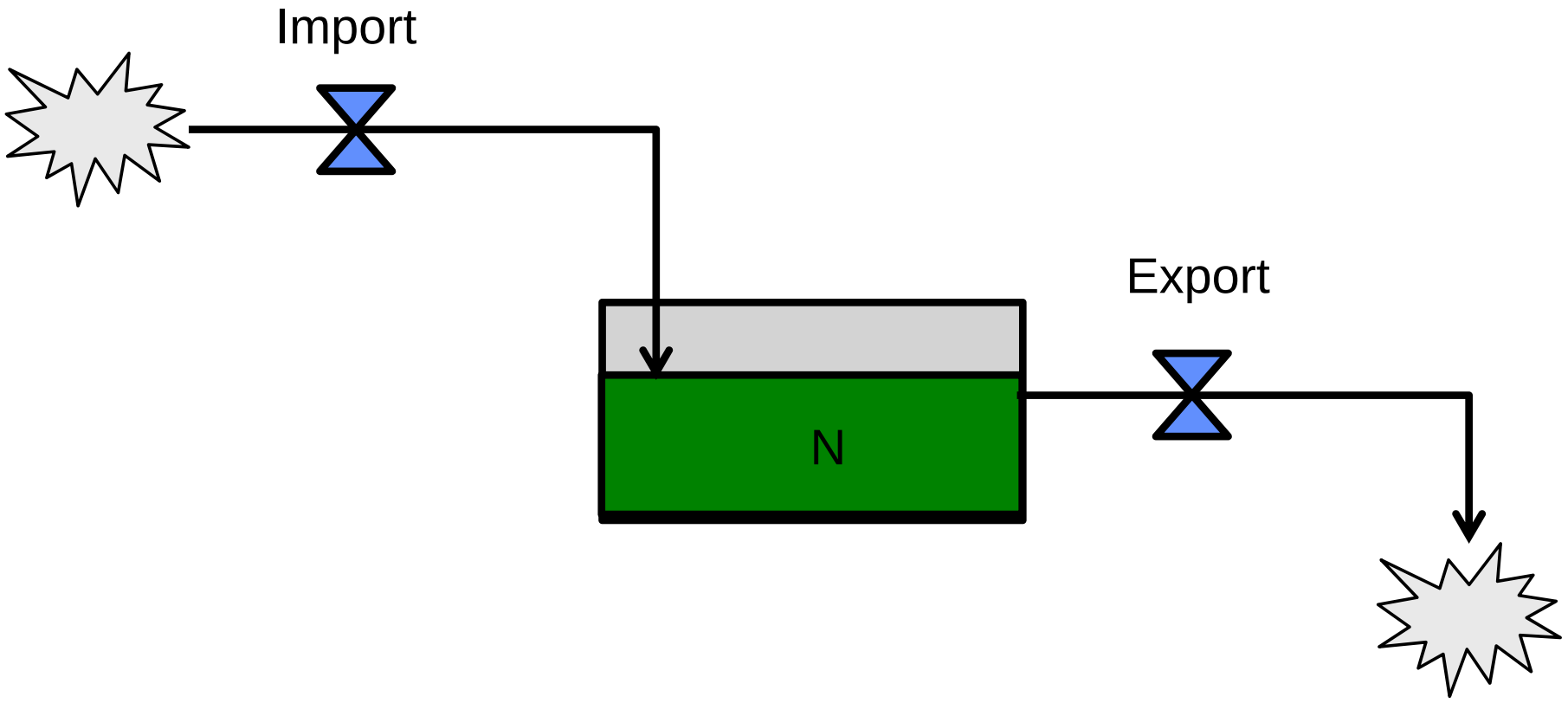


$$death = d \cdot N$$

d = death rate

$$\frac{dN}{dt} = growth - death = b \cdot N - d \cdot N$$

The opposite: Import and export



$$\frac{dN}{dt} = \text{import} - \text{export}$$

Elementary growth

$$\frac{dN}{dt} = b \cdot N - d \cdot N$$

$$\frac{dN}{dt} = (b - d) \cdot N = r \cdot N$$

In R: Exponential growth, solved analytically

$$\frac{dN}{dt} = rN$$

$$\int_0^t \frac{dN}{dt} = \int_0^t rN$$

$$\int_0^t \frac{1}{N} dN = r \int_0^t rN$$

$$\ln(N) = rt + c$$

$$N_t = N_0 e^{r \cdot t}$$

```
## parameters, initial values,
## time steps
r    <- 0.5
N0   <- 10
dt   <- 0.1
time <- seq(0, 10, dt)

## analytical solution
N <- N0 * exp(r * time)
plot(time, N, type="l")
```

Exp. growth solved stepwise, numerically

```
N <- numeric(length(time))

N[1] <- N0
for (i in 2:length(time)) {
  N[i] <- N[i-1] + r * N[i-1] * dt
}
plot(time, N, type = "l")
```

This is called the Euler method.

Exp. Growth: „individual-based"

```
inds <- 1:10
N[1] <- length(inds)
for (i in 2:length(time)) {
  zufall  <- runif(length(inds))
  newinds <- subset(inds, zufall < r * dt)
  inds    <- c(inds, newinds)
  N[i]    <- length(inds)
}
plot(time, N, type = "l")
inds
```

How is growth limited?

Two fundamentally different approaches:

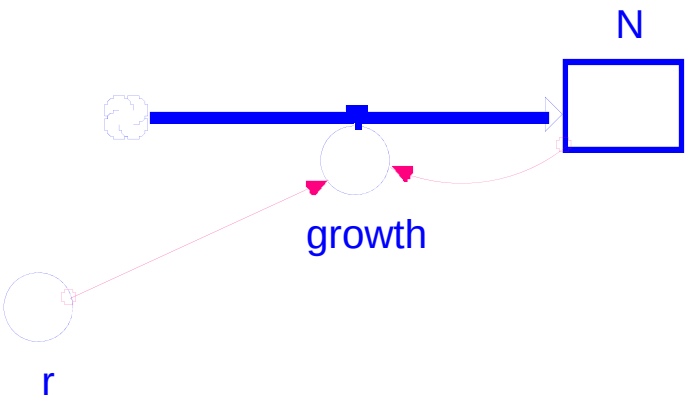
1. Carrying capacity concept
   - Growth rate decreases if „carrying capacity“ is approached.

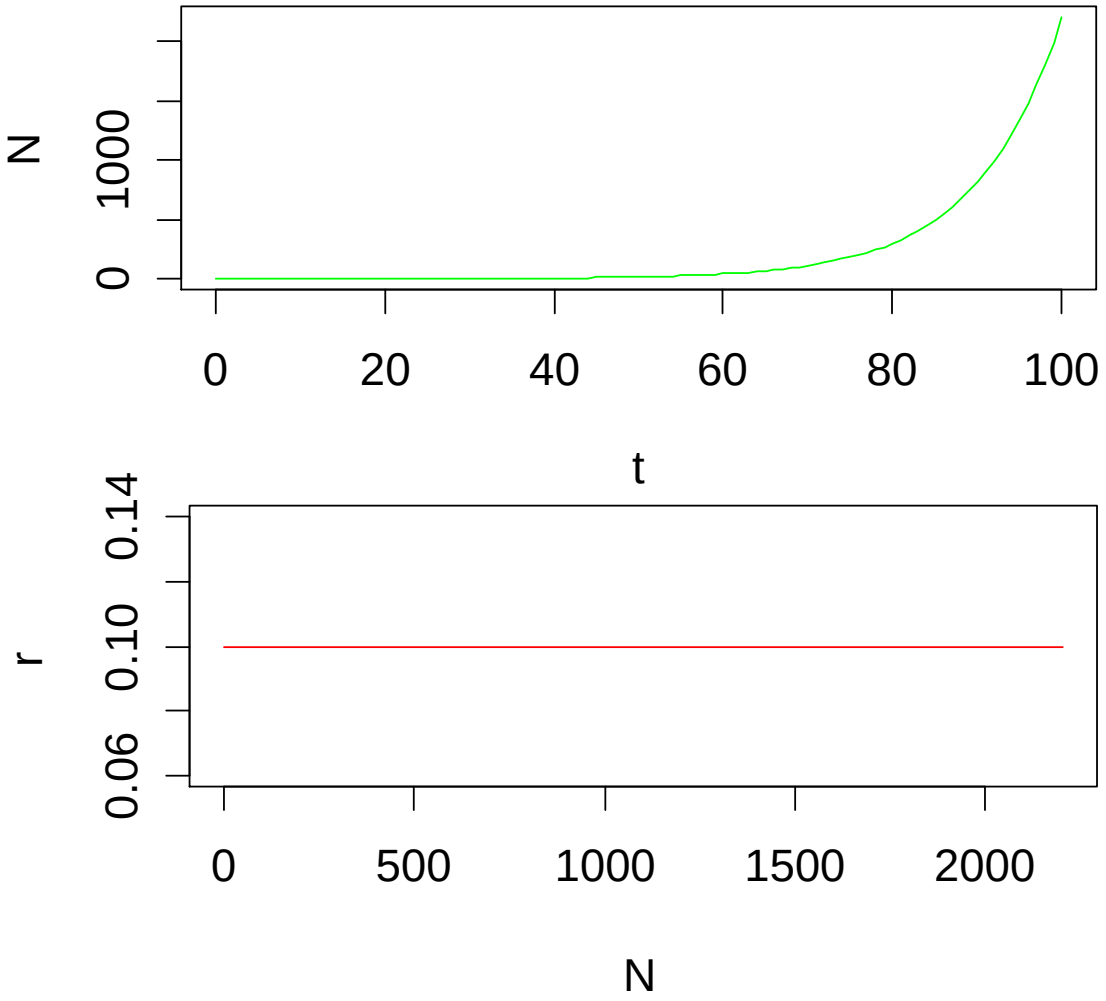
2a) Limiting resource
   - Phosphorus, nitrogen, …

2b) Grazing and predation

Abundance is controlled by another species (i.e., interaction)

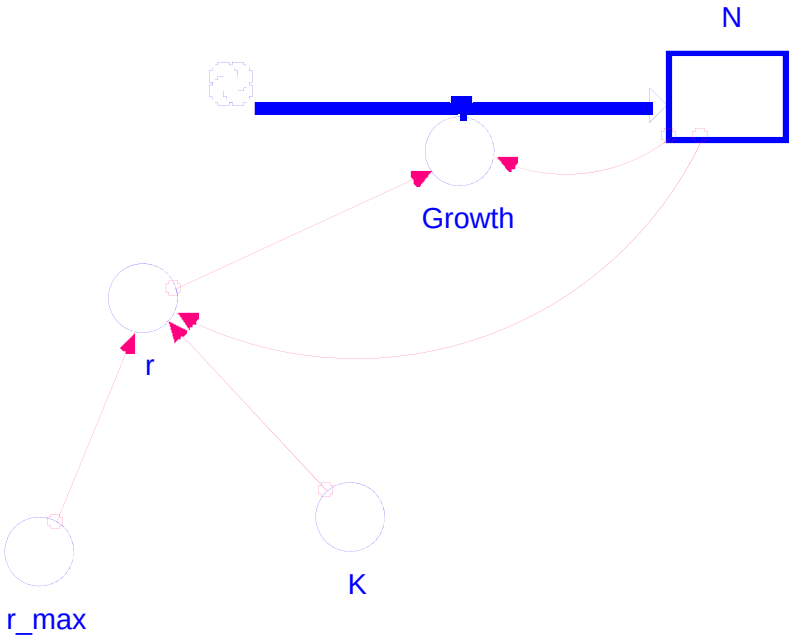
Exponential Growth



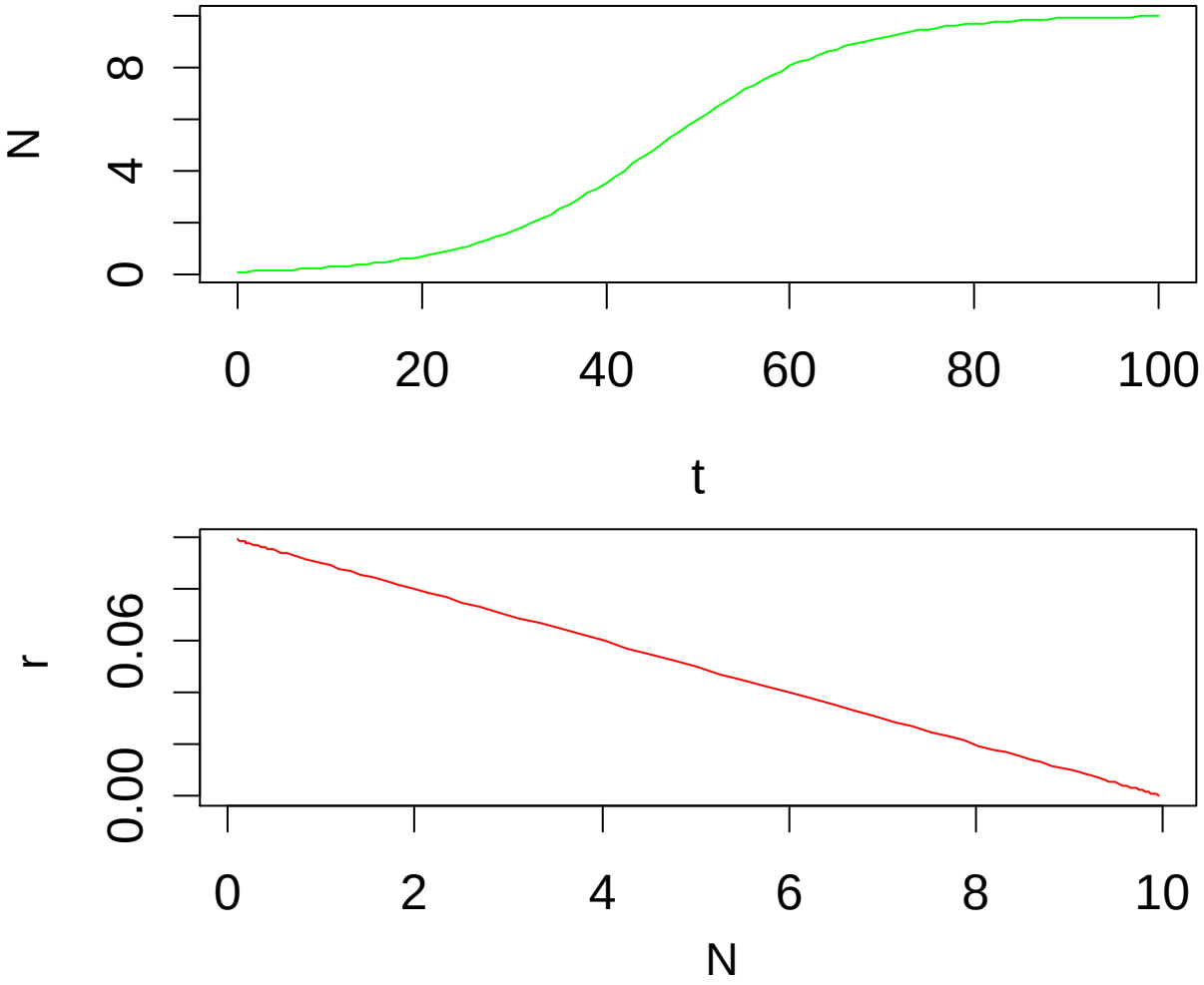


$$\frac{dN}{dt} = r \cdot N$$

Carrying Capacity: Logistic Growth



$$\frac{dN}{dt} = r_{\max} \cdot N \cdot \left(1 - \frac{N}{K}\right)$$



Solution of the Logistic

Analytical solution:

$$N_t = \frac{KN_0e^{rt}}{K + N_0(e^{rt} - 1)}$$

```
logistic <- function(t, r, K, N0) {
  K * N0 * exp(r * t) / (K + N0*(exp(r * t) - 1))
}
r <- 0.1; K <- 10; N0 <- 0.1
times <- 1:100

plot(times, logistic(times, r, K, N0))
```

Numerical simulation with package deSolve

```
library(deSolve)
model <- function (time, y, parms) {
  with(as.list(c(y, parms)), {
    dx1 <-   r * N * (1 - N / K)
    list(c(dx1))
  })
}

y      <- c(N = 0.1)
parms  <- c(r = 0.1, K = 10)
times  <- seq(0, 100, 1)

out <- ode(y, times, model, parms)

plot(out)
```

How does ode work?

- **ode** is a „differential equation solver function“ provided by the R package **deSolve**
- it runs specific integration methods (e.g. lsoda, ode45, rk4, euler) that call the model function for specific time steps with the specified parameters.

- some of the functions use exactly the time step specified by the user (**euler**)
- others do extra steps to increase accuracy (**rk4, ...**)
- most other solvers select the time steps automatically to ensure a given accuracy
  - **lsoda, lsode, vode, ode45, …**
  - the tolerance can be adjusted with **atol** and **rtol**

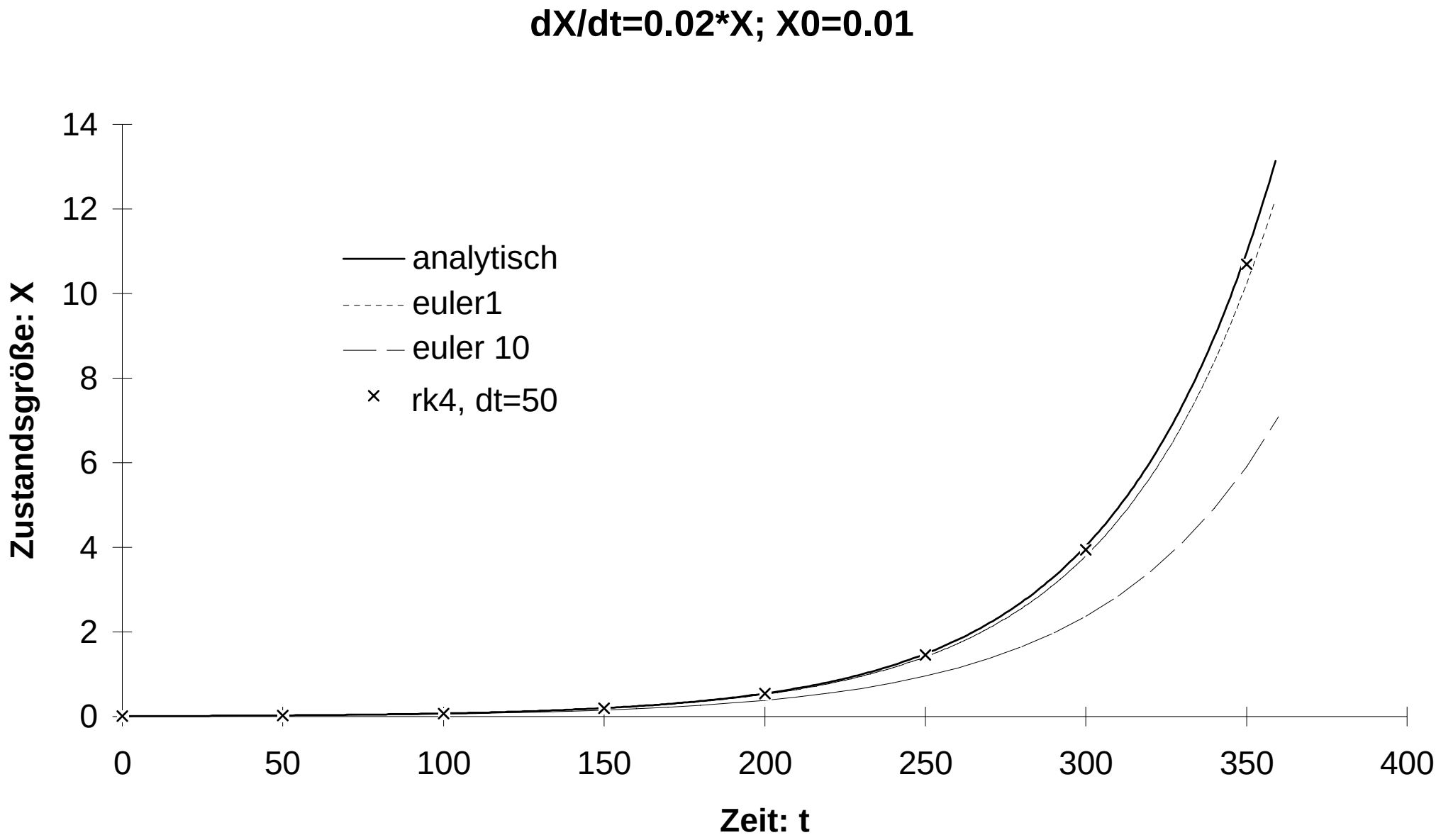
Exercise:

- compare other solvers (`method = “euler“`), especially:
  - **lsoda** (the default and the "first choice" recommended to start with)
  - **euler** (simulates the model step by step without additional measures)

    →modify the time steps

  … other solvers, if you like.

Analytical and numerical integration



Classical Runge-Kutta method of 4th order

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

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

RK4 method

- an explicit method, that uses the value of the last time step y(n)
- value of the new time step y(n+1) is calculated as weighted mean of 4 derivatives of intermediate time steps.

Problems of rk4

Disadvantages: low Precision, high effort

- Precision of the results is unknown
  → large step size: precision too low
  → small step size: sufficient precision, but effort too high

➔ Danger of Instability

- possibility of wrongly negative values → fluctuations
- this is a problem of all fixed step methods, including Euler

Advantages

- If a good choice for step size is know, rk4 can be efficient because:
  – Interpolation step size for external forcing functions is known
  – Possibility to couple several models with different time step.

Other methods

- Explicit, implicit, semi-implicit
  - explicit methods: forward calculation, using only the last time step y(t)
  - implicit methods use the new time step y(t), but require iteration
  - semi-implicit methods use both, y(t) and y(t+1)

- Numerous explicit methods available, e.g. Runge-Kuttas

  > library(deSolve)

  > rkMethods()

- Variable step size methods combine two methods or one method with two time steps. The error is calculated by comparison of the two methods.
  - e.g.: ode23, **ode45** (Dormand-Prince)

- AB-Method (explicit method after Adams-Bashforth),
- Adams-Moulton (pedictor-corrector-method; AB formula and implicit corrector)
- BDF (backward differentiation formula, implicit method)
  - suitable for stiff systems

The lsoda solver

- „Livermore Solver for Ordinary Differential Equations“ (lsoda) von PETZOLD (1983) and HINDMARSH (1983).

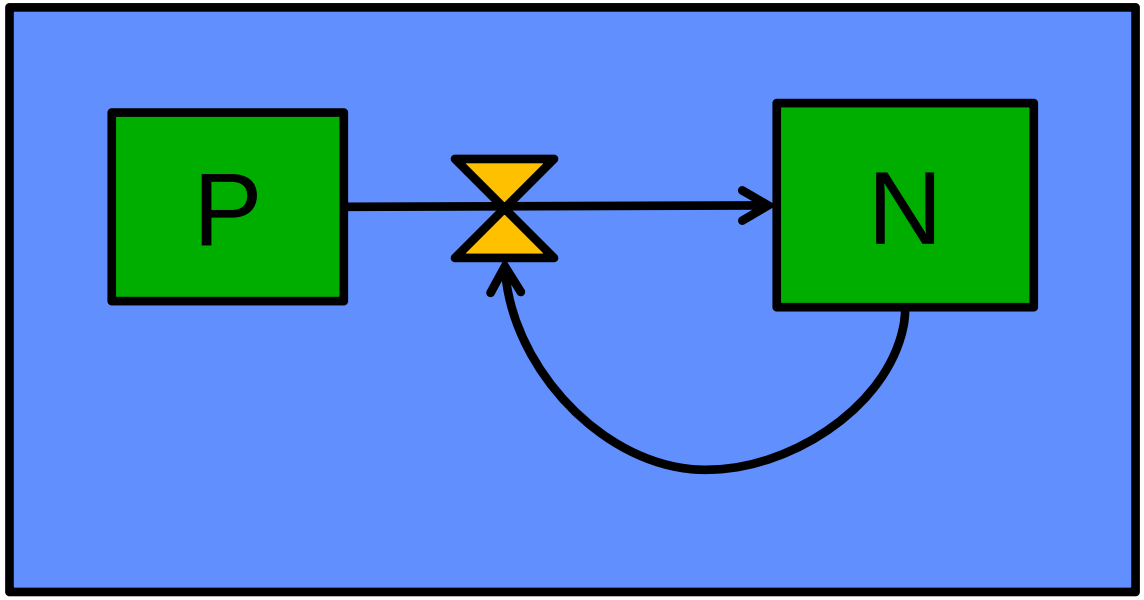
- lsoda selects automatcally one of two integrators:
  - explicit method after Adams for „well behaving systems“
  - implizit BDF (*backward differential formula*)-for stiff problems

- stiffness: state variables have very different „speed of change“

- Additional time saving possible if matrix of derivatives (Jacobian) is known
  - can be provided analytically
  - otherwise approximated, internally

A limiting nutrient

- We have two state variables, the Phytoplankton and a nutrient.

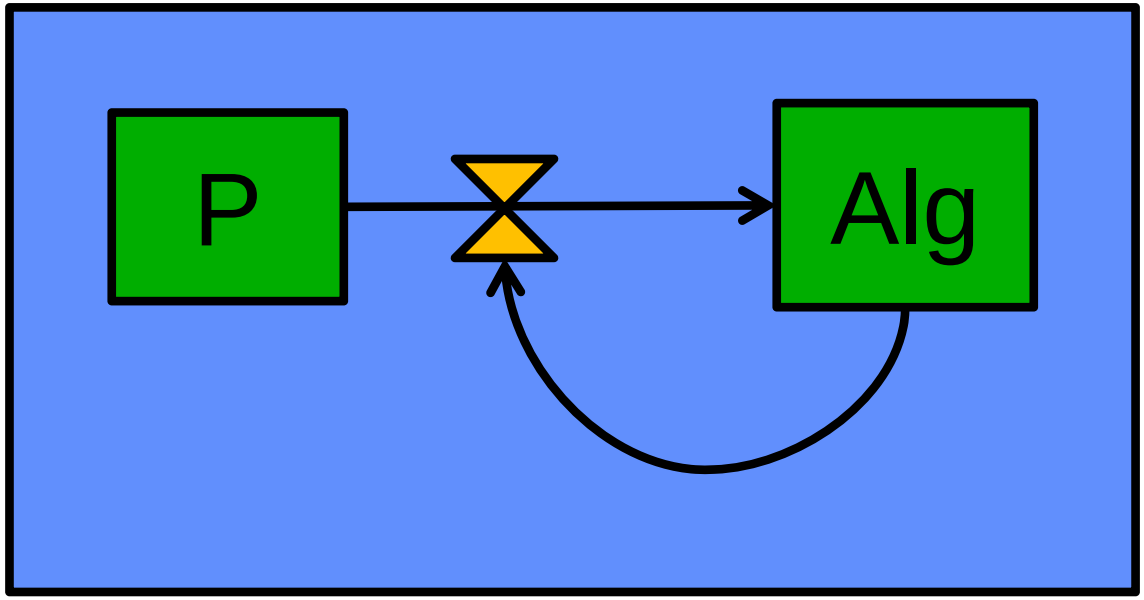


Note: Symbols are often confusing!
Population ecologists use N for abundance
but N is nitrogen in aquatic sciences.

So practical modellers use often abbreviations consiting of multiple letters
(mathematicians doent like this, but it makes programming easier)

A limiting nutrient

- We have two state variables, the Phytoplankton and a nutrient.



$$\frac{d\mathrm{Alg}}{dt} = r_{\max} \cdot f(P) \cdot \mathrm{Alg}$$

$$\frac{dP}{dt} = \ldots$$

Mass balances and conversions

$$\frac{d\mathrm{Alg}}{dt} = r_{\max} \cdot f(P) \cdot \mathrm{Alg}$$

$$\frac{dP}{dt} = \ldots$$

Excercise:

- how can we describe $f(P)$? (It‘s a well-known function)
- what happens with the phosphorus?

Mass balances and conversions

$$\frac{d\mathrm{Alg}}{dt} = r_{\max} \cdot f(P) \cdot \mathrm{Alg}$$

$$\frac{dP}{dt} = -r_{\max} \cdot \frac{1}{Y} \cdot f(P) \cdot \mathrm{Alg}$$

$$f(P) = \frac{P}{k_P + P}$$

Phosphorus dependent growth

```
model <- function (time, y, parms) {
  with(as.list(c(y, parms)), {
    f <- P/(kP + P)
    dAlg <- r * f * Alg
    dP   <- - r * 1/Y * f * Alg
    list(c(dAlg, dP))
  })
}

y      <- c(Alg = 0.1, P = 0.2) # in mg/L
parms  <- c(r = 0.1, kP = 5e-3, Y = 41) # Y = C:P mass ratio
times  <- seq(0, 100, 1)

out <- ode(y, times, model, parms)

plot(out)
```

Molar mass calculations

```
> library(marelac)

>  redfield(1, species="P")
    C   H   O  N P
  106 263 110 16 1


> redfield(1, species="P", method="mass")
        C        H        O        N P
  41.10363 8.558477 56.82016 7.235388 1
```

Exercise

Now, try the same with **`euler`** and **`rk4`**

- then reduce the stepsize (e.g. 1.0, 0.8, 0.5, 0.1)
- this can be done either by modifying the **`times`** vector

Now, try lsoda with **increased** step size!