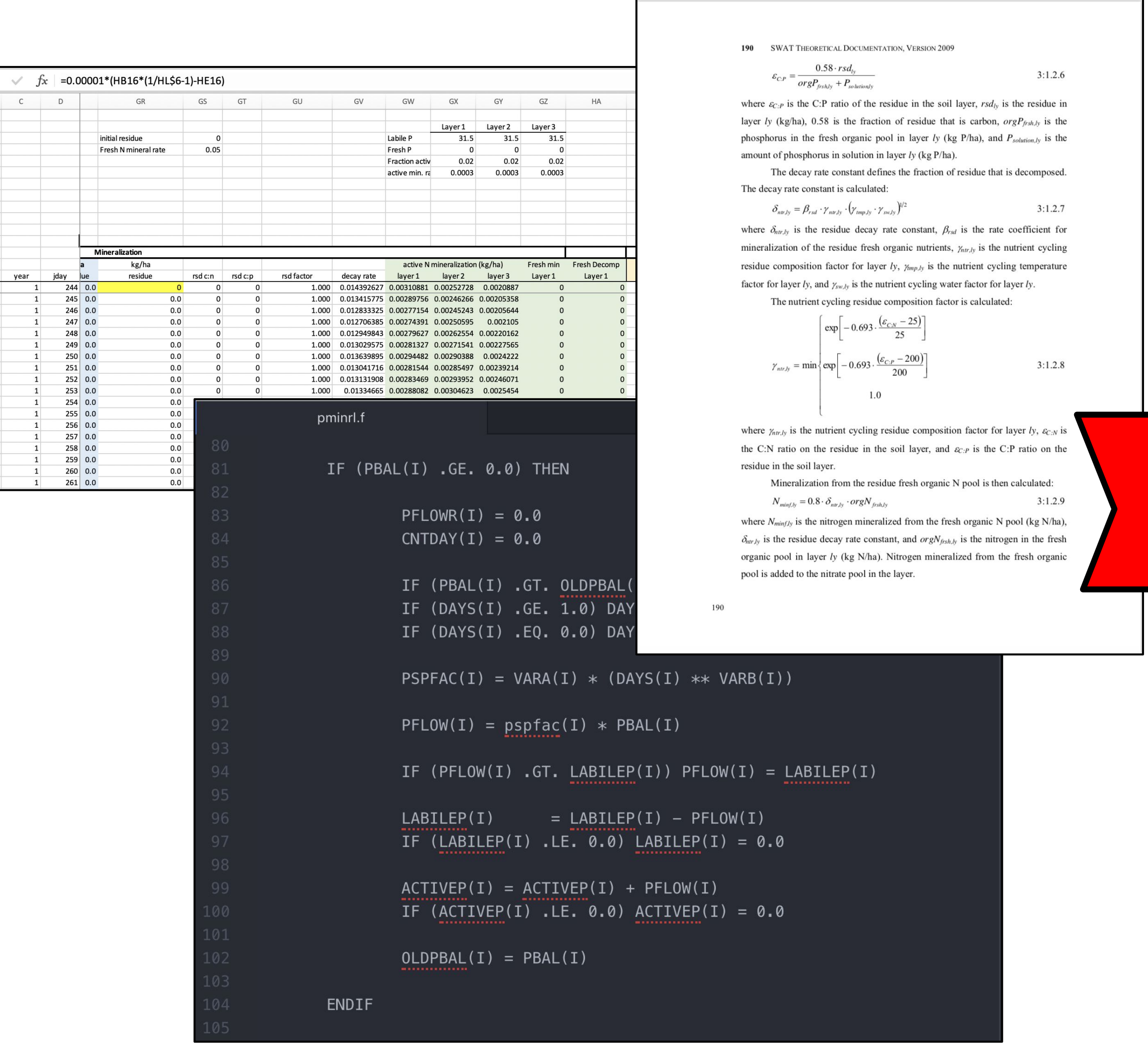
Programming RuFaS

A Tour of the Coding Process

Pseudocode



=0.00001*(HB16*(1/HL$6-1)-HE16)

C	D		GR	GS	GT	GU	GV	GW	GX	GY	GZ	HA
									Layer 1	Layer 2	Layer 3	
			initial residue	0				Labile P	31.5	31.5	31.5	
			Fresh N mineral rate	0.05				Fresh P	0	0	0	
								Fraction activ	0.02	0.02	0.02	
								active min. ra	0.0003	0.0003	0.0003	
			Mineralization									
		a	kg/ha					active N mineralization (kg/ha)			Fresh min	Fresh Decomp
year	jday	lue	residue	rsd c:n	rsd c:p	rsd factor	decay rate	layer 1	layer 2	layer 3	Layer 1	Layer 1
1	244	0.0	0	0	0	1.000	0.014392627	0.00310881	0.00252728	0.0020887	0	0
1	245	0.0	0.0	0	0	1.000	0.013415775	0.00289756	0.00246266	0.00205358	0	0
1	246	0.0	0.0	0	0	1.000	0.012833325	0.00277154	0.00245243	0.00205644	0	0
1	247	0.0	0.0	0	0	1.000	0.012706385	0.00274391	0.00250595	0.002105	0	0
1	248	0.0	0.0	0	0	1.000	0.012949843	0.00279627	0.00262554	0.00220162	0	0
1	249	0.0	0.0	0	0	1.000	0.013029575	0.00281327	0.00271541	0.00227565	0	0
1	250	0.0	0.0	0	0	1.000	0.013639895	0.00294482	0.00290388	0.0024222	0	0
1	251	0.0	0.0	0	0	1.000	0.013041716	0.00281544	0.00285497	0.00239214	0	0
1	252	0.0	0.0	0	0	1.000	0.013131908	0.00283469	0.00293952	0.00246071	0	0
1	253	0.0	0.0	0	0	1.000	0.01334665	0.00288082	0.00304623	0.0025454	0	0
1	254	0.0	0.0									
1	255	0.0	0.0									
1	256	0.0	0.0									
1	257	0.0	0.0									
1	258	0.0	0.0									
1	259	0.0	0.0									
1	260	0.0	0.0									
1	261	0.0	0.0									

pminrl.f

```

      IF (PBAL(I) .GE. 0.0) THEN

          PFLOWR(I) = 0.0
          CNTDAY(I) = 0.0

          IF (PBAL(I) .GT. OLDPBAL(
          IF (DAYS(I) .GE. 1.0) DAY
          IF (DAYS(I) .EQ. 0.0) DAY

          PSPFAC(I) = VARA(I) * (DAYS(I) ** VARB(I))

          PFLOW(I) = pspfac(I) * PBAL(I)

          IF (PFLOW(I) .GT. LABILEP(I)) PFLOW(I) = LABILEP(I)

          LABILEP(I)      = LABILEP(I) - PFLOW(I)
          IF (LABILEP(I) .LE. 0.0) LABILEP(I) = 0.0

          ACTIVEP(I) = ACTIVEP(I) + PFLOW(I)
          IF (ACTIVEP(I) .LE. 0.0) ACTIVEP(I) = 0.0

          OLDPBAL(I) = PBAL(I)

      ENDIF

```

190 SWAT THEORETICAL DOCUMENTATION, VERSION 2009

$$\varepsilon_{C:P} = \frac{0.58 \cdot rsd_{ly}}{orgP_{frsh,ly} + P_{solution,ly}} \qquad 3:1.2.6$$

where $\varepsilon_{C:P}$ is the C:P ratio of the residue in the soil layer, rsd_{ly} is the residue in layer ly (kg/ha), 0.58 is the fraction of residue that is carbon, $orgP_{frsh,ly}$ is the phosphorus in the fresh organic pool in layer ly (kg P/ha), and $P_{solution,ly}$ is the amount of phosphorus in solution in layer ly (kg P/ha).

The decay rate constant defines the fraction of residue that is decomposed. The decay rate constant is calculated:

$$\delta_{ntr,ly} = \beta_{rsd} \cdot \gamma_{ntr,ly} \cdot \left(\gamma_{tmp,ly} \cdot \gamma_{sw,ly}\right)^{1/2} \qquad 3:1.2.7$$

where $\delta_{ntr,ly}$ is the residue decay rate constant, β_{rsd} is the rate coefficient for mineralization of the residue fresh organic nutrients, $\gamma_{ntr,ly}$ is the nutrient cycling residue composition factor for layer ly, $\gamma_{tmp,ly}$ is the nutrient cycling temperature factor for layer ly, and $\gamma_{sw,ly}$ is the nutrient cycling water factor for layer ly.

The nutrient cycling residue composition factor is calculated:

$$\gamma_{ntr,ly} = \min\begin{cases} \exp\left[-0.693 \cdot \frac{(\varepsilon_{C:N} - 25)}{25}\right] \\ \exp\left[-0.693 \cdot \frac{(\varepsilon_{C:P} - 200)}{200}\right] \\ 1.0 \end{cases} \qquad 3:1.2.8$$

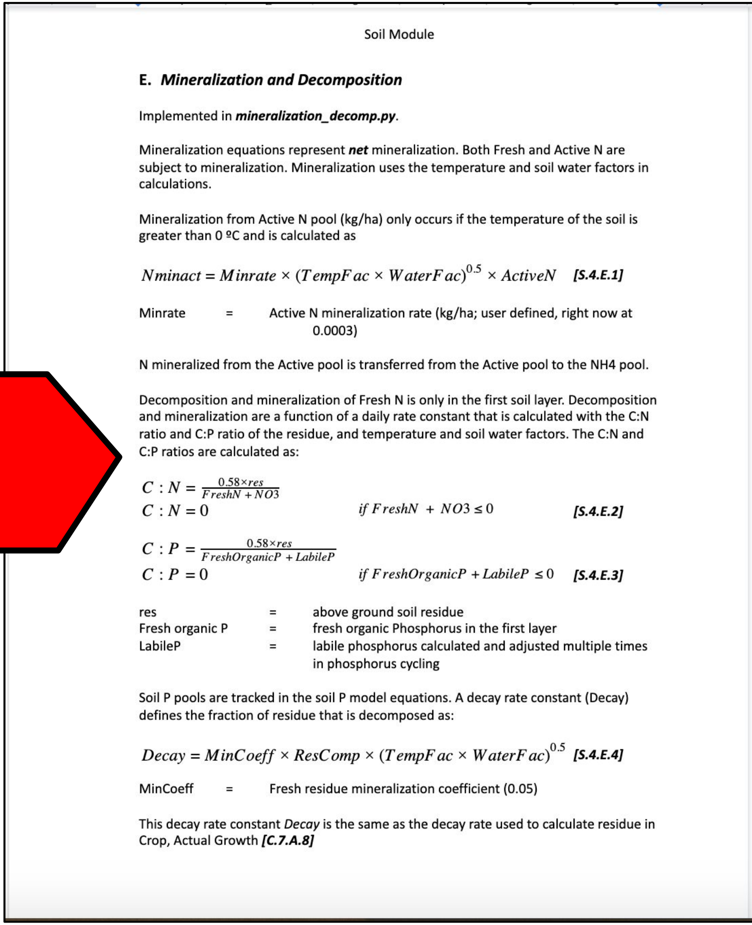
where $\gamma_{ntr,ly}$ is the nutrient cycling residue composition factor for layer ly, $\varepsilon_{C:N}$ is the C:N ratio on the residue in the soil layer, and $\varepsilon_{C:P}$ is the C:P ratio on the residue in the soil layer.

Mineralization from the residue fresh organic N pool is then calculated:

$$N_{minf,ly} = 0.8 \cdot \delta_{ntr,ly} \cdot orgN_{frsh,ly} \qquad 3:1.2.9$$

where $N_{minf,ly}$ is the nitrogen mineralized from the fresh organic N pool (kg N/ha), $\delta_{ntr,ly}$ is the residue decay rate constant, and $orgN_{frsh,ly}$ is the nitrogen in the fresh organic pool in layer ly (kg N/ha). Nitrogen mineralized from the fresh organic pool is added to the nitrate pool in the layer.

190



Soil Module

E. *Mineralization and Decomposition*

Implemented in ***mineralization_decomp.py***.

Mineralization equations represent ***net*** mineralization. Both Fresh and Active N are subject to mineralization. Mineralization uses the temperature and soil water factors in calculations.

Mineralization from Active N pool (kg/ha) only occurs if the temperature of the soil is greater than 0 ºC and is calculated as

$$Nminact = Minrate \times (TempFac \times WaterFac)^{0.5} \times ActiveN \qquad \textbf{[S.4.E.1]}$$

Minrate = Active N mineralization rate (kg/ha; user defined, right now at 0.0003)

N mineralized from the Active pool is transferred from the Active pool to the NH4 pool.

Decomposition and mineralization of Fresh N is only in the first soil layer. Decomposition and mineralization are a function of a daily rate constant that is calculated with the C:N ratio and C:P ratio of the residue, and temperature and soil water factors. The C:N and C:P ratios are calculated as:

$$C:N = \frac{0.58 \times res}{FreshN + NO3}$$
$$C:N = 0 \qquad if\ FreshN + NO3 \le 0 \qquad \textbf{[S.4.E.2]}$$

$$C:P = \frac{0.58 \times res}{FreshOrganicP + LabileP}$$
$$C:P = 0 \qquad if\ FreshOrganicP + LabileP \le 0 \qquad \textbf{[S.4.E.3]}$$

res = above ground soil residue
Fresh organic P = fresh organic Phosphorus in the first layer
LabileP = labile phosphorus calculated and adjusted multiple times in phosphorus cycling

Soil P pools are tracked in the soil P model equations. A decay rate constant (Decay) defines the fraction of residue that is decomposed as:

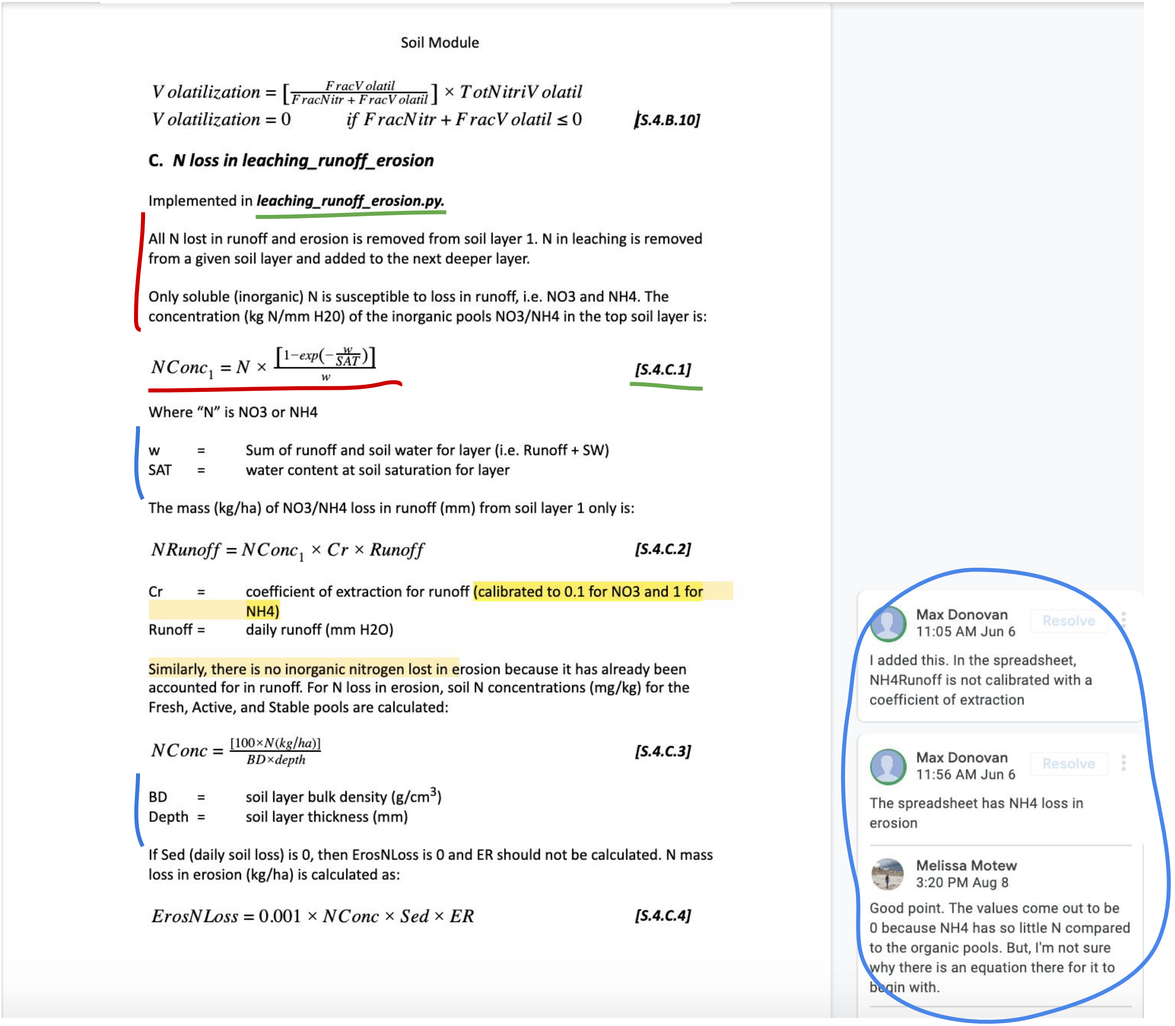
$$Decay = MinCoeff \times ResComp \times (TempFac \times WaterFac)^{0.5} \qquad \textbf{[S.4.E.4]}$$

MinCoeff = Fresh residue mineralization coefficient (0.05)

This decay rate constant *Decay* is the same as the decay rate used to calculate residue in Crop, Actual Growth ***[C.7.A.8]***

Pseudocode

- Readable
- Descriptive
- Programmer Friendly
- Code Compatible



Soil Module

$$Volatilization = [\frac{FracVolatil}{FracNitr + FracVolatil}] \times TotNitriVolatil$$

$$Volatilization = 0 \qquad if\ FracNitr + FracVolatil \leq 0 \qquad \textbf{[S.4.B.10]}$$

C. *N loss in leaching_runoff_erosion*

Implemented in ***leaching_runoff_erosion.py.***

All N lost in runoff and erosion is removed from soil layer 1. N in leaching is removed from a given soil layer and added to the next deeper layer.

Only soluble (inorganic) N is susceptible to loss in runoff, i.e. NO3 and NH4. The concentration (kg N/mm H20) of the inorganic pools NO3/NH4 in the top soil layer is:

$$NConc_1 = N \times \frac{[1-exp(-\frac{w}{SAT})]}{w} \qquad \textbf{[S.4.C.1]}$$

Where "N" is NO3 or NH4

w = Sum of runoff and soil water for layer (i.e. Runoff + SW)
SAT = water content at soil saturation for layer

The mass (kg/ha) of NO3/NH4 loss in runoff (mm) from soil layer 1 only is:

$$NRunoff = NConc_1 \times Cr \times Runoff \qquad \textbf{[S.4.C.2]}$$

Cr = coefficient of extraction for runoff (calibrated to 0.1 for NO3 and 1 for NH4)
Runoff = daily runoff (mm H2O)

Similarly, there is no inorganic nitrogen lost in erosion because it has already been accounted for in runoff. For N loss in erosion, soil N concentrations (mg/kg) for the Fresh, Active, and Stable pools are calculated:

$$NConc = \frac{[100 \times N(kg/ha)]}{BD \times depth} \qquad \textbf{[S.4.C.3]}$$

BD = soil layer bulk density (g/cm^3)
Depth = soil layer thickness (mm)

If Sed (daily soil loss) is 0, then ErosNLoss is 0 and ER should not be calculated. N mass loss in erosion (kg/ha) is calculated as:

$$ErosNLoss = 0.001 \times NConc \times Sed \times ER \qquad \textbf{[S.4.C.4]}$$

Max Donovan
11:05 AM Jun 6

I added this. In the spreadsheet, NH4Runoff is not calibrated with a coefficient of extraction

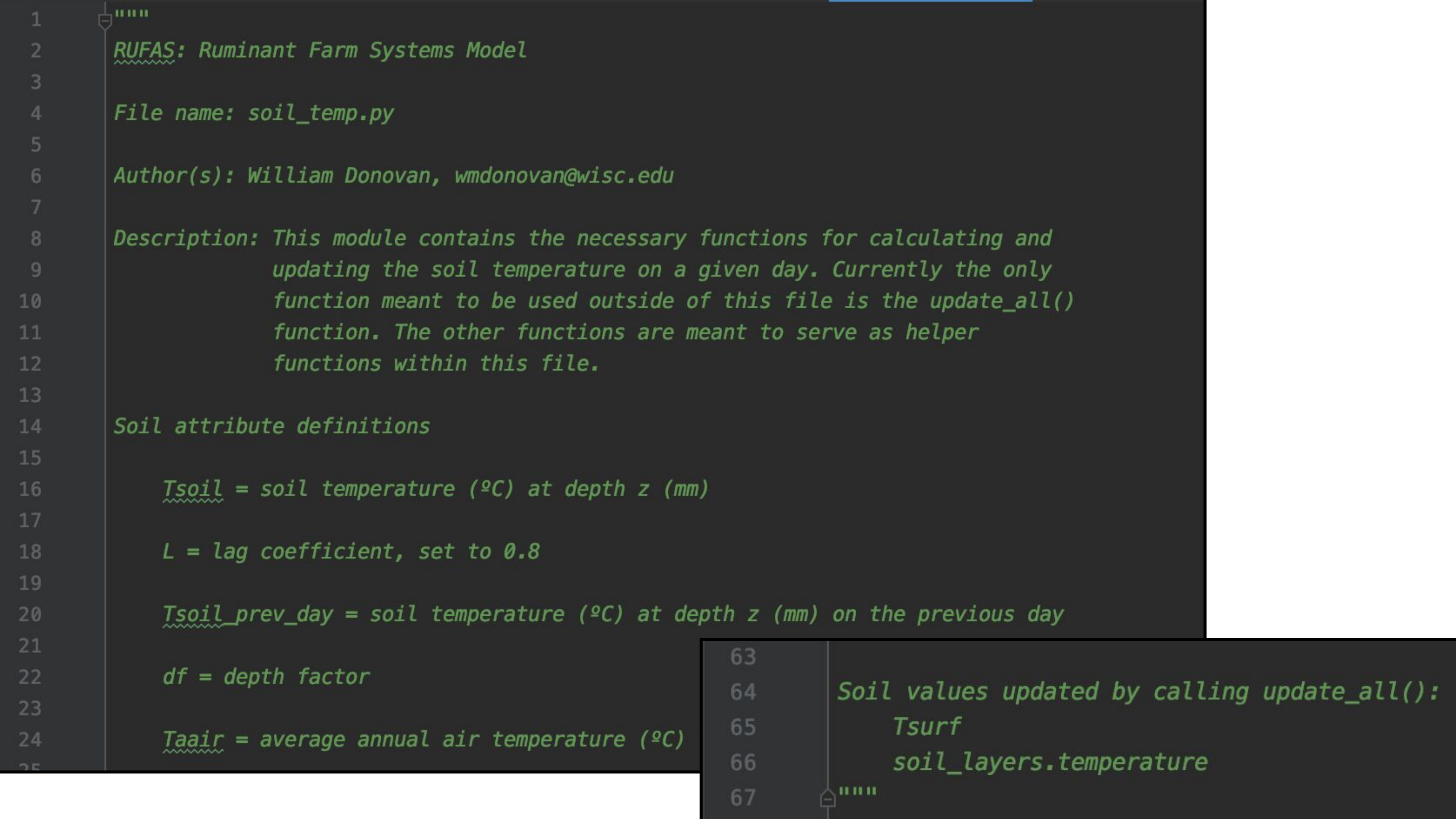
Max Donovan
11:56 AM Jun 6

The spreadsheet has NH4 loss in erosion

Melissa Motew
3:20 PM Aug 8

Good point. The values come out to be 0 because NH4 has so little N compared to the organic pools. But, I'm not sure why there is an equation there for it to begin with.

Development

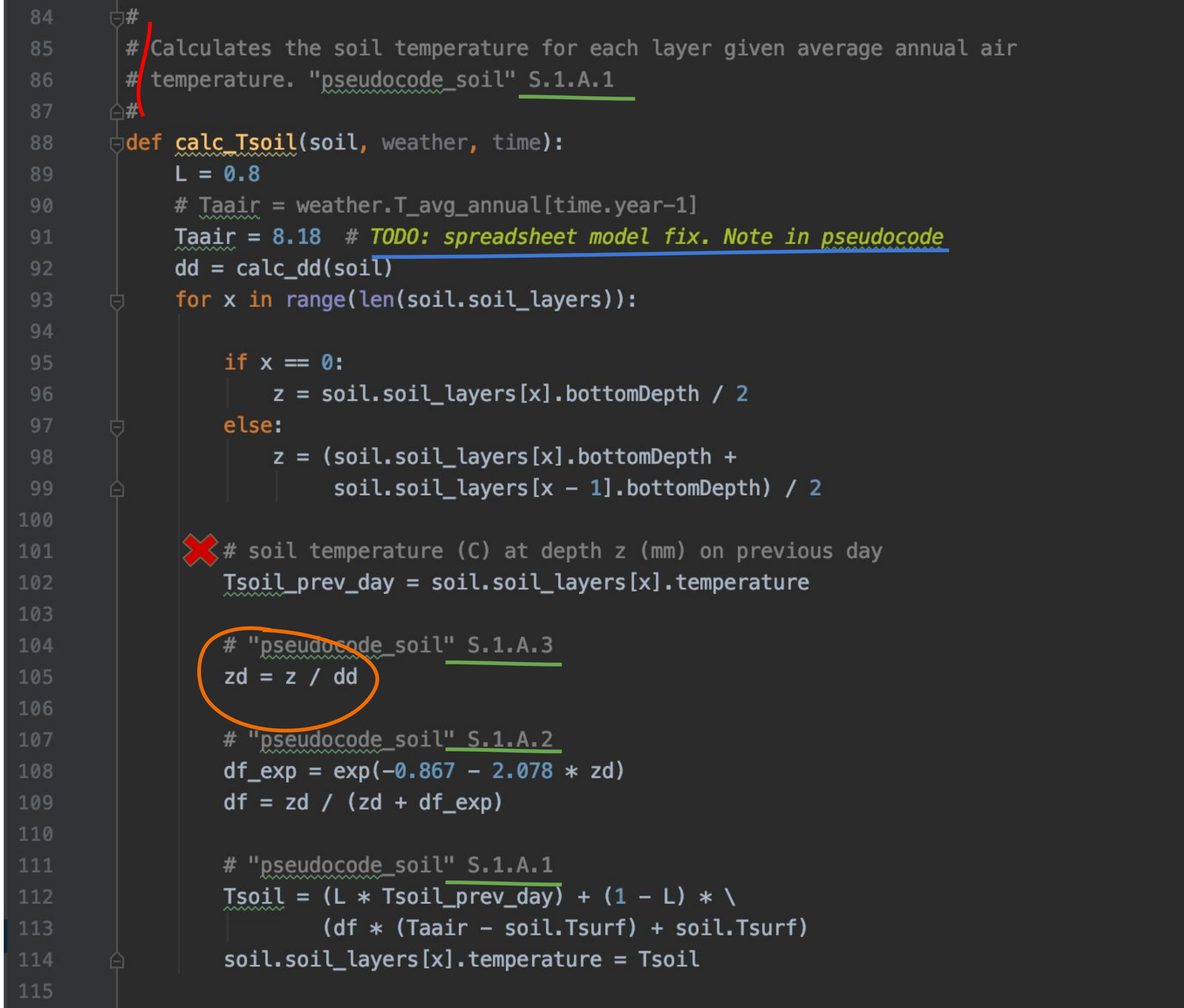


```
"""
RUFAS: Ruminant Farm Systems Model

File name: soil_temp.py

Author(s): William Donovan, wmdonovan@wisc.edu

Description: This module contains the necessary functions for calculating and
             updating the soil temperature on a given day. Currently the only
             function meant to be used outside of this file is the update_all()
             function. The other functions are meant to serve as helper
             functions within this file.

Soil attribute definitions

    Tsoil = soil temperature (ºC) at depth z (mm)

    L = lag coefficient, set to 0.8

    Tsoil_prev_day = soil temperature (ºC) at depth z (mm) on the previous day

    df = depth factor

    Taair = average annual air temperature (ºC)
```

```

Soil values updated by calling update_all():
    Tsurf
    soil_layers.temperature
"""
```

Code

- In-line Comments
- Active Development
- Comparable Variable Names
- Pseudocode Indexing



```
#
# Calculates the soil temperature for each layer given average annual air
# temperature. "pseudocode_soil" S.1.A.1
#
def calc_Tsoil(soil, weather, time):
    L = 0.8
    # Taair = weather.T_avg_annual[time.year-1]
    Taair = 8.18  # TODO: spreadsheet model fix. Note in pseudocode
    dd = calc_dd(soil)
    for x in range(len(soil.soil_layers)):

        if x == 0:
            z = soil.soil_layers[x].bottomDepth / 2
        else:
            z = (soil.soil_layers[x].bottomDepth +
                 soil.soil_layers[x - 1].bottomDepth) / 2

        # soil temperature (C) at depth z (mm) on previous day
        Tsoil_prev_day = soil.soil_layers[x].temperature

        # "pseudocode_soil" S.1.A.3
        zd = z / dd

        # "pseudocode_soil" S.1.A.2
        df_exp = exp(-0.867 - 2.078 * zd)
        df = zd / (zd + df_exp)

        # "pseudocode_soil" S.1.A.1
        Tsoil = (L * Tsoil_prev_day) + (1 - L) * \
                (df * (Taair - soil.Tsurf) + soil.Tsurf)
        soil.soil_layers[x].temperature = Tsoil

```

Equations taken from SWAT 2009 documentation. Implemented in ***soil_temp.py***.

A. *Base equation*

$$T_{soil} = \left(L \times T_{soil,d-1}\right) + (1-L) \times \left[df \times \left(T_{aair} - T_{surf}\right) + T_{surf}\right]$$

[S.1.A.1]

T_{soil} = soil temperature (^{O}C) at depth z (mm)
L = lag coefficient, set to 0.8
$T_{soil,d-1}$ = soil temperature (^{O}C) at depth z (mm) on previous day
df = depth factor
T_{aair} = Average annual air temperature (^{O}C)
T_{surf} = Daily soil surface temperature (^{O}C)

$$df = \frac{zd}{[zd+exp(-0.867-2.078\times zd)]}$$

[S.1.A.2]

zd = ratio of depth at the center of soil layer to damping depth

$$zd = \frac{z}{dd}$$

[S.1.A.3]

dd = damping depth (mm)
z = depth at the center of the soil layer

Damping depth is a function of soil water and a max damping depth as:

$$dd = dd_{max} \times exp\left\{ln\left(\frac{500}{ddmax}\right) \times \left[\frac{(1-scale)}{(1+scale)}\right]^2\right\}$$

[S.1.A.4]

dd_{max} = maximum damping depth (mm)
scale = scaling factor for soil water

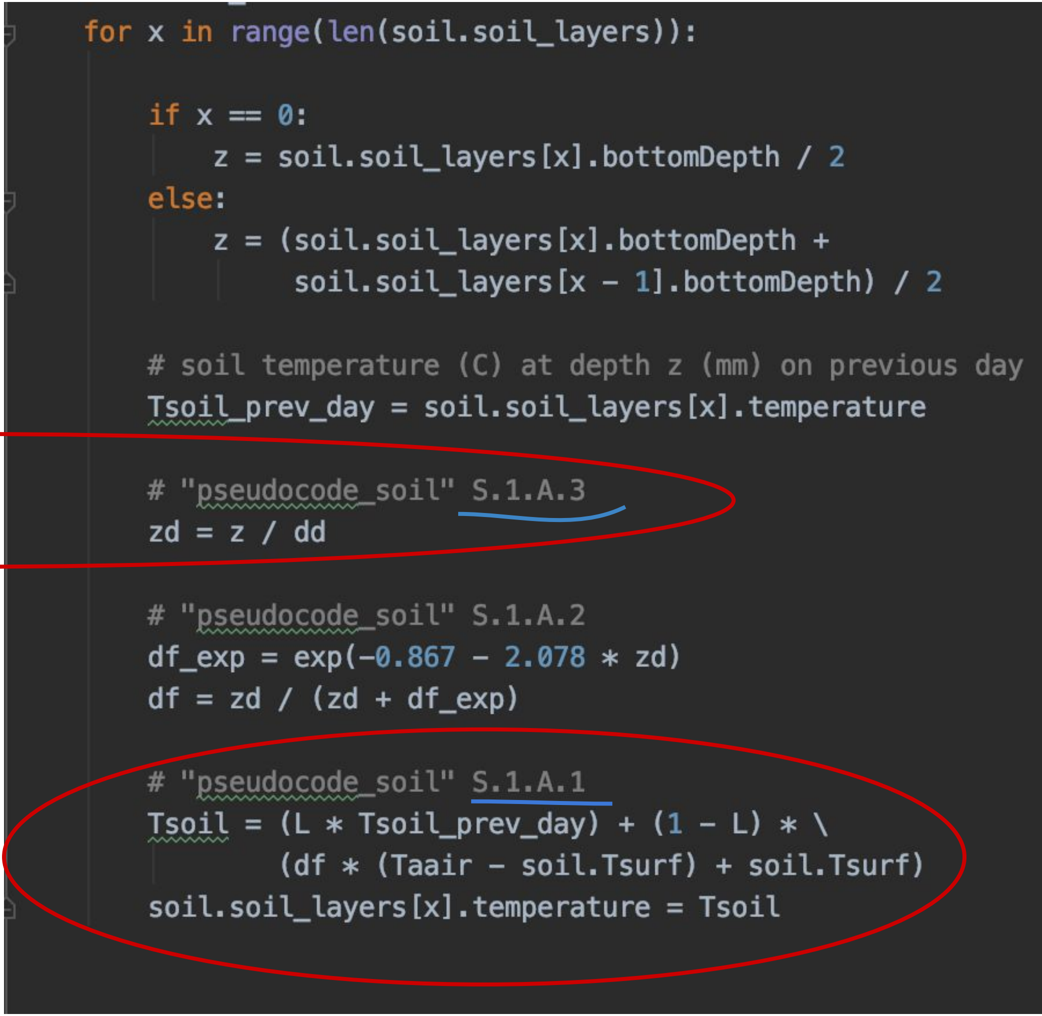
$$scale = \frac{SW}{[(0.356-0.144\times bd)\times ztot]}$$

[S.1.A.5]

bd = soil bulk density (g/cm^3)
SW = total soil water in the profile (mm)
Ztot = total soil profile depth

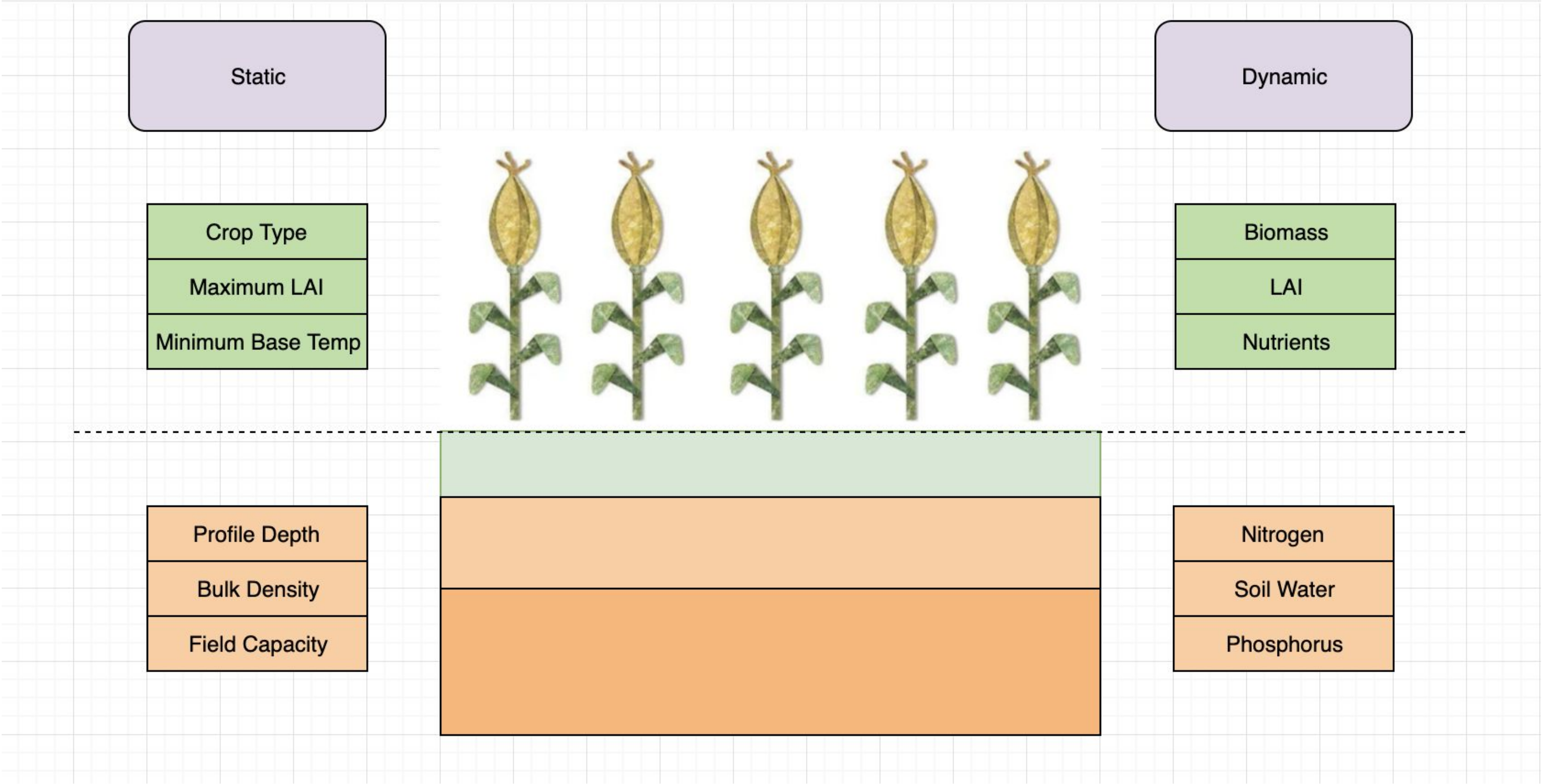
$$dd_{max} = 1000 + \frac{(2500\times bd)}{[bd+686\times exp(-5.63\times bd)]}$$

[S.1.A.6]



```
for x in range(len(soil.soil_layers)):

    if x == 0:
        z = soil.soil_layers[x].bottomDepth / 2
    else:
        z = (soil.soil_layers[x].bottomDepth +
             soil.soil_layers[x - 1].bottomDepth) / 2

    # soil temperature (C) at depth z (mm) on previous day
    Tsoil_prev_day = soil.soil_layers[x].temperature

    # "pseudocode_soil" S.1.A.3
    zd = z / dd

    # "pseudocode_soil" S.1.A.2
    df_exp = exp(-0.867 - 2.078 * zd)
    df = zd / (zd + df_exp)

    # "pseudocode_soil" S.1.A.1
    Tsoil = (L * Tsoil_prev_day) + (1 - L) * \
            (df * (Taair - soil.Tsurf) + soil.Tsurf)
    soil.soil_layers[x].temperature = Tsoil
```

Structure



Static

Crop Type

Maximum LAI

Minimum Base Temp

Profile Depth

Bulk Density

Field Capacity

Dynamic

Biomass

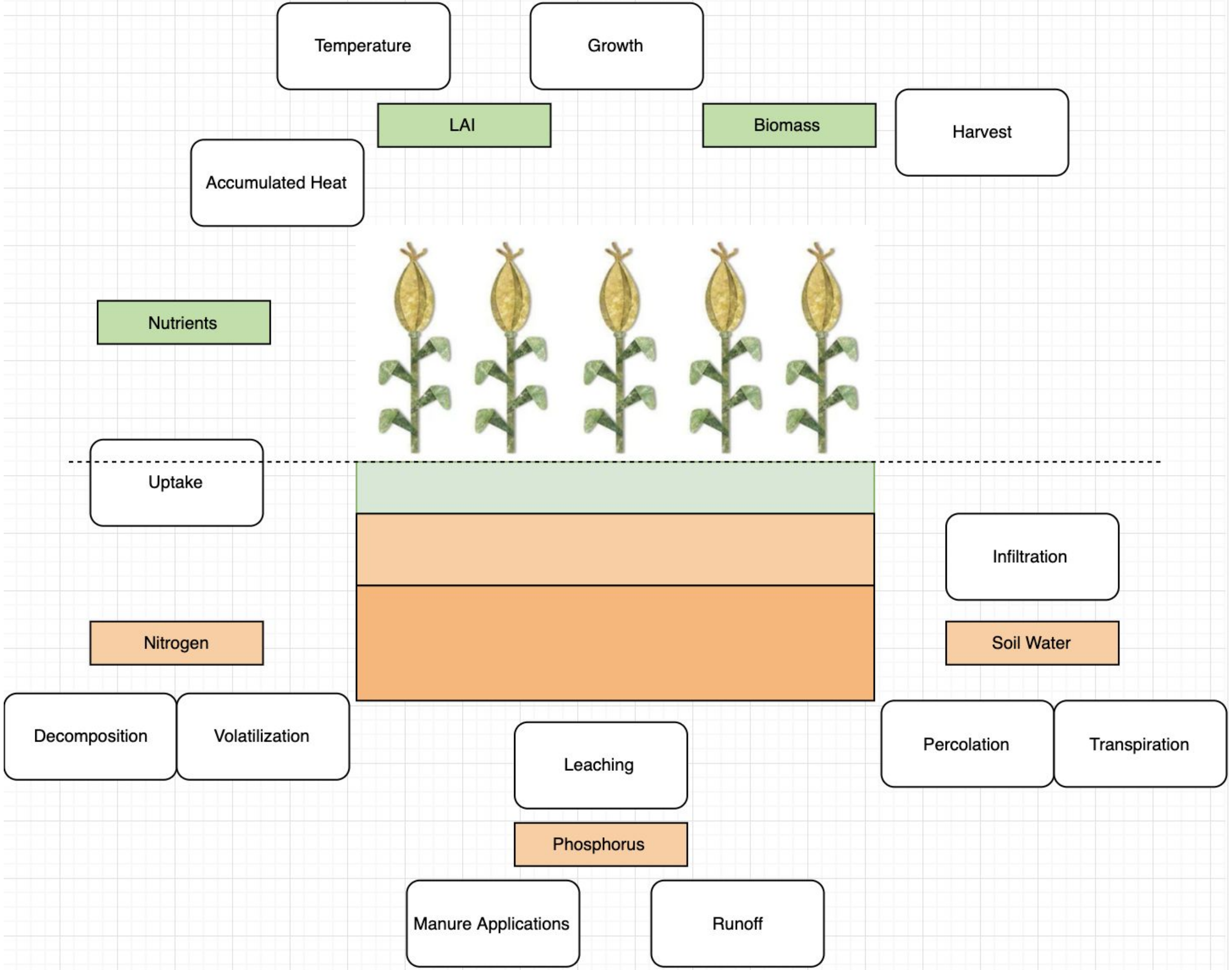
LAI

Nutrients

Nitrogen

Soil Water

Phosphorus



Temperature

Growth

LAI

Biomass

Harvest

Accumulated Heat

Nutrients

Uptake

Infiltration

Nitrogen

Soil Water

Decomposition

Volatilization

Percolation

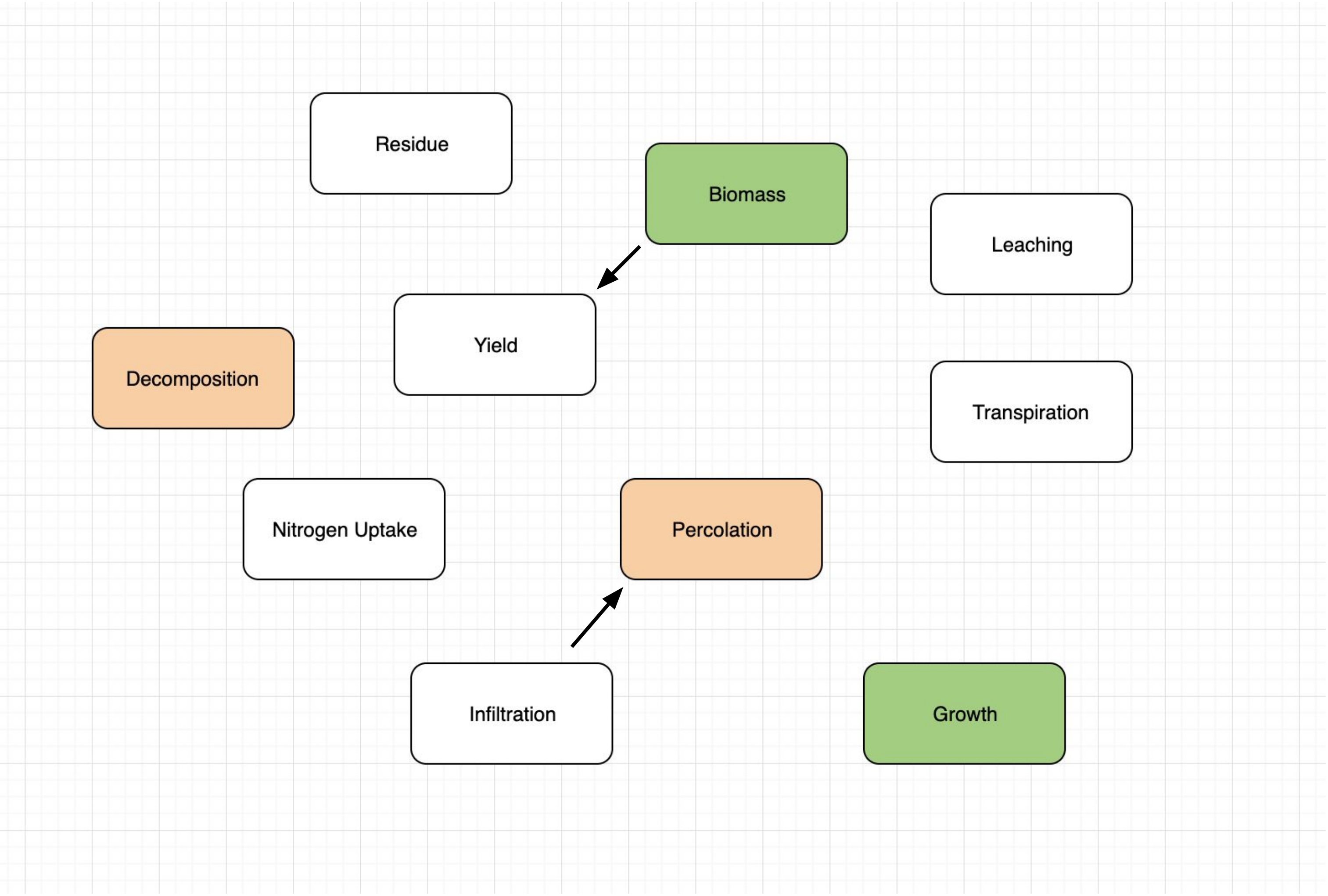
Transpiration

Leaching

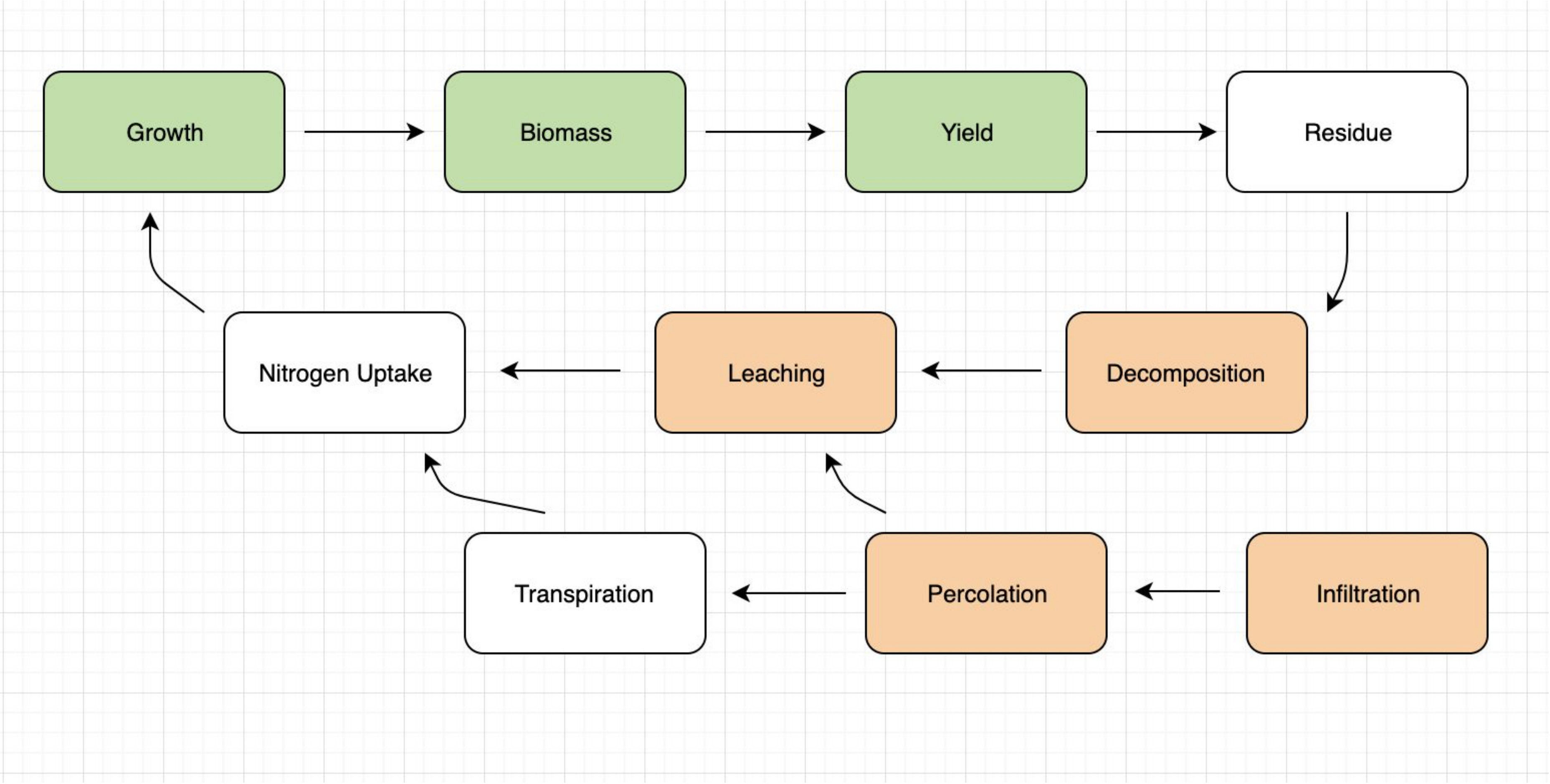
Phosphorus

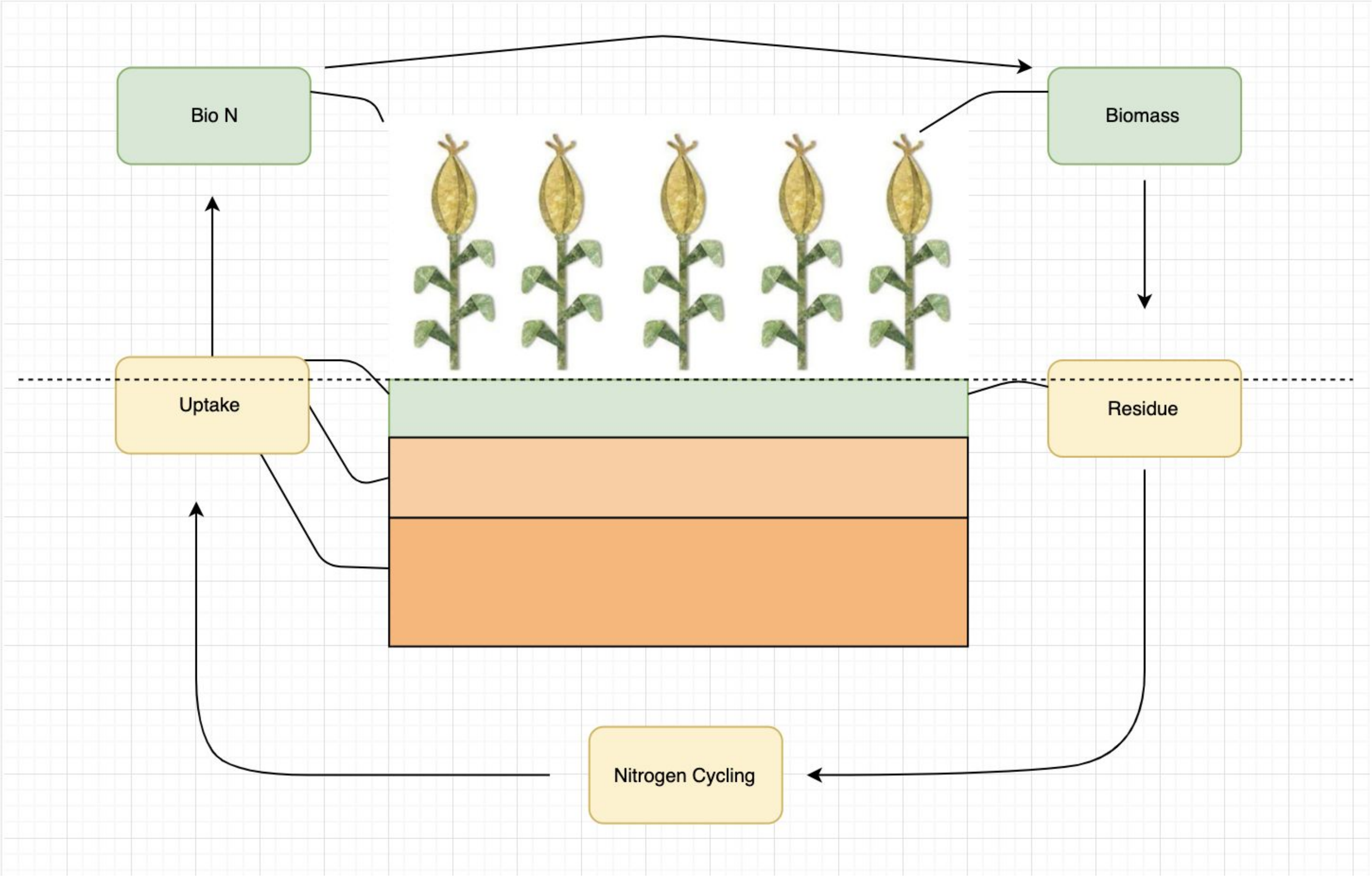
Manure Applications

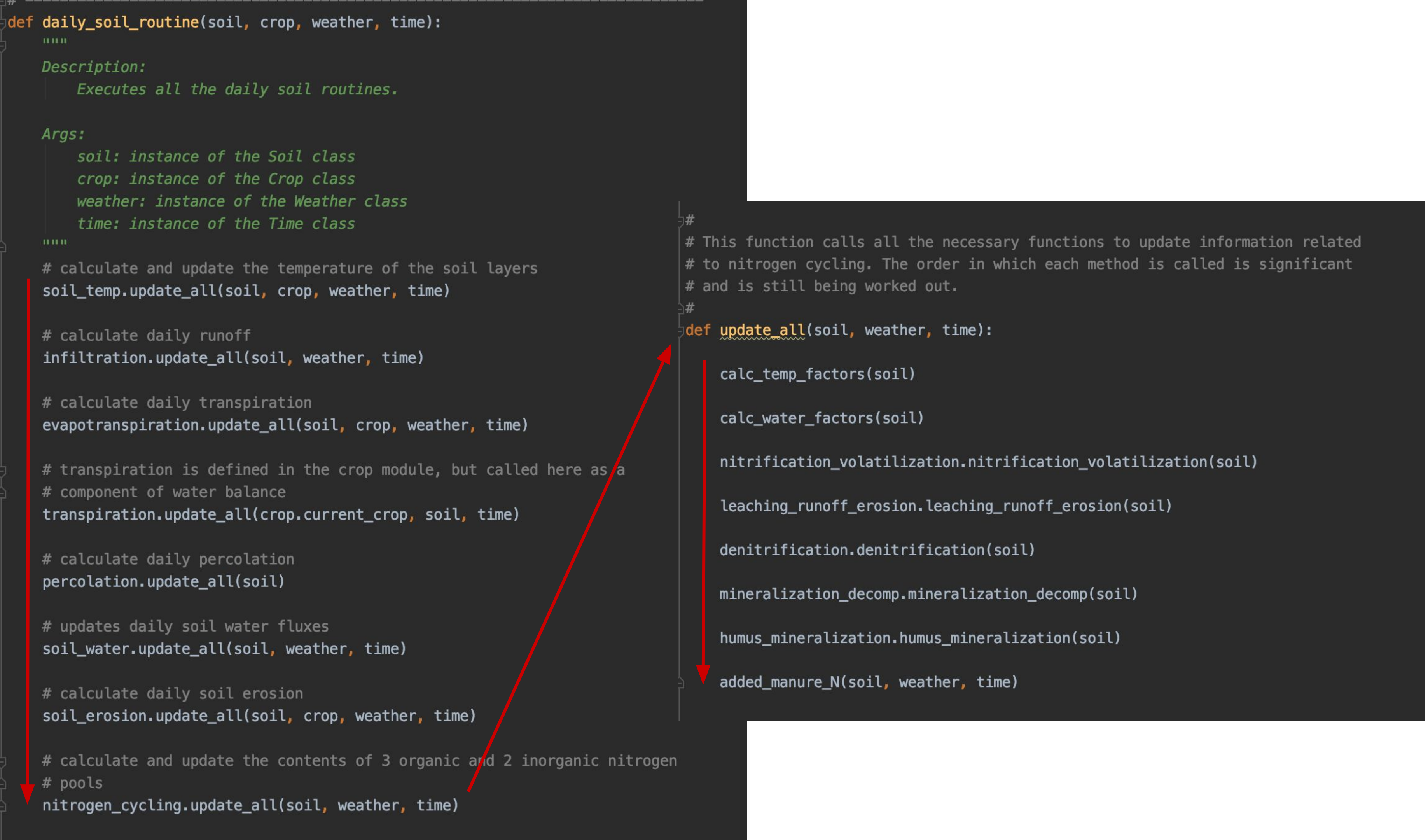
Runoff


Residue
Biomass
Leaching
Yield
Decomposition
Transpiration
Nitrogen Uptake
Percolation
Infiltration
Growth

Order of Operations

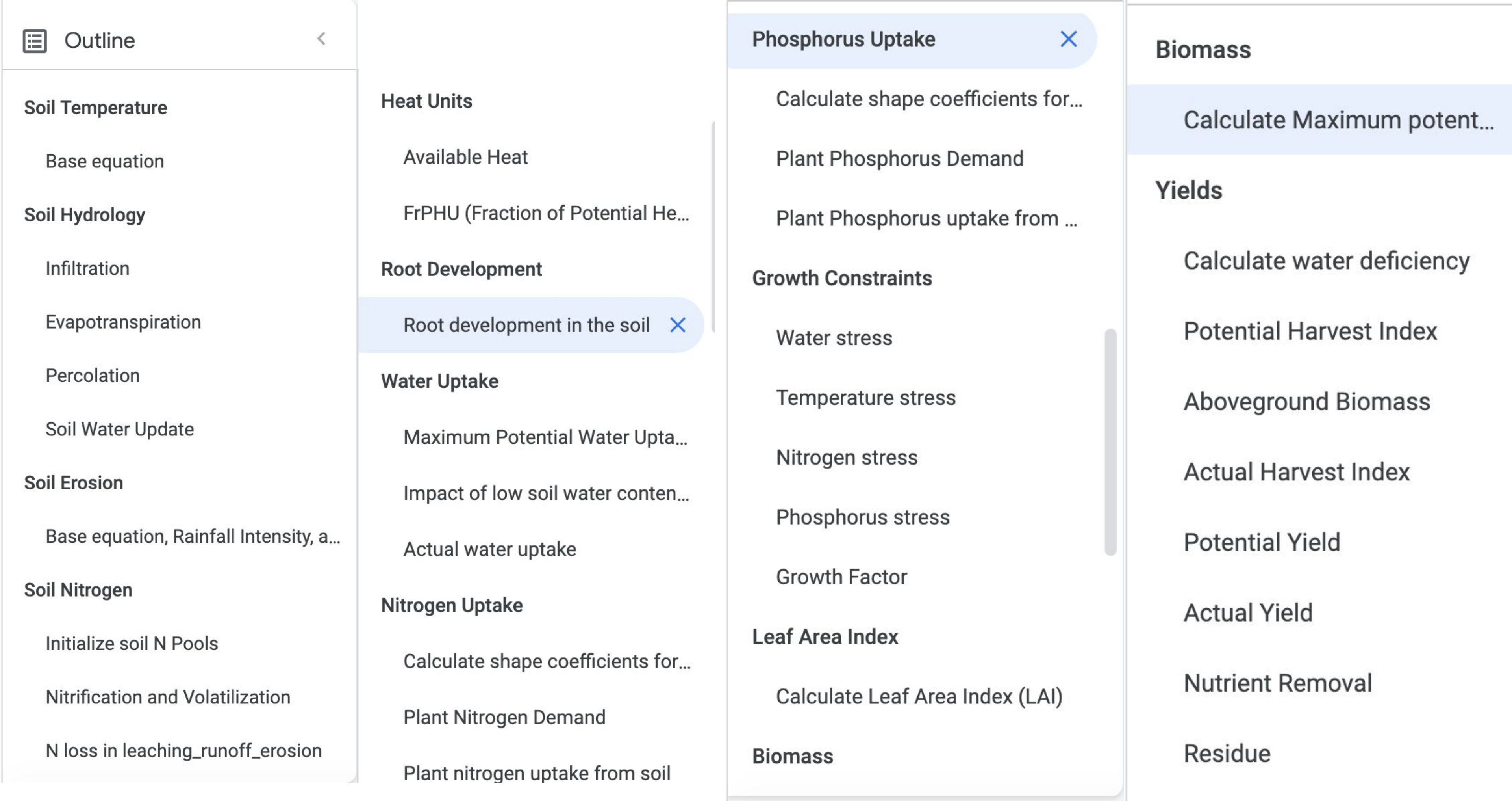

Growth
Biomass
Yield
Residue
Nitrogen Uptake
Leaching
Decomposition
Transpiration
Percolation
Infiltration


Bio N
Biomass
Uptake
Residue
Nitrogen Cycling



```
def daily_soil_routine(soil, crop, weather, time):
    """
    Description:
        Executes all the daily soil routines.

    Args:
        soil: instance of the Soil class
        crop: instance of the Crop class
        weather: instance of the Weather class
        time: instance of the Time class
    """
    # calculate and update the temperature of the soil layers
    soil_temp.update_all(soil, crop, weather, time)

    # calculate daily runoff
    infiltration.update_all(soil, weather, time)

    # calculate daily transpiration
    evapotranspiration.update_all(soil, crop, weather, time)

    # transpiration is defined in the crop module, but called here as a
    # component of water balance
    transpiration.update_all(crop.current_crop, soil, time)

    # calculate daily percolation
    percolation.update_all(soil)

    # updates daily soil water fluxes
    soil_water.update_all(soil, weather, time)

    # calculate daily soil erosion
    soil_erosion.update_all(soil, crop, weather, time)

    # calculate and update the contents of 3 organic and 2 inorganic nitrogen
    # pools
    nitrogen_cycling.update_all(soil, weather, time)
```

```
#
# This function calls all the necessary functions to update information related
# to nitrogen cycling. The order in which each method is called is significant
# and is still being worked out.
#
def update_all(soil, weather, time):

    calc_temp_factors(soil)

    calc_water_factors(soil)

    nitrification_volatilization.nitrification_volatilization(soil)

    leaching_runoff_erosion.leaching_runoff_erosion(soil)

    denitrification.denitrification(soil)

    mineralization_decomp.mineralization_decomp(soil)

    humus_mineralization.humus_mineralization(soil)

    added_manure_N(soil, weather, time)
```



Outline

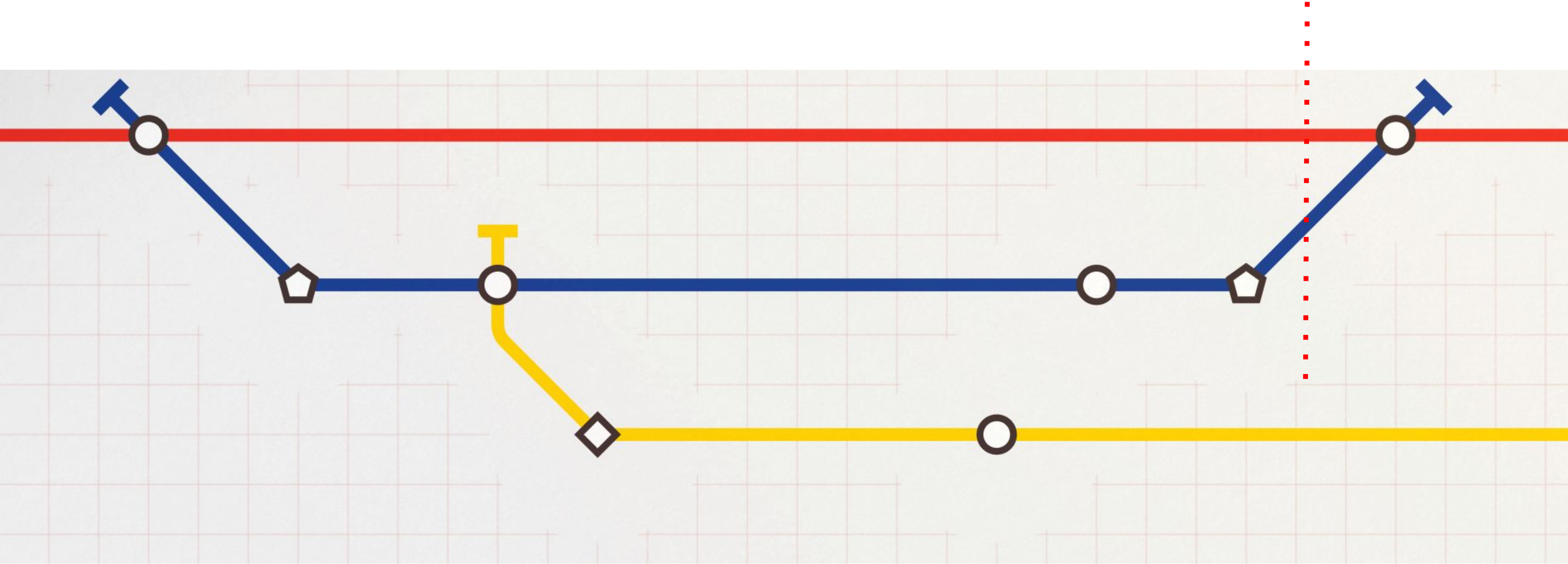
- **Soil Temperature**
  - Base equation
- **Soil Hydrology**
  - Infiltration
  - Evapotranspiration
  - Percolation
  - Soil Water Update
- **Soil Erosion**
  - Base equation, Rainfall Intensity, a...
- **Soil Nitrogen**
  - Initialize soil N Pools
  - Nitrification and Volatilization
  - N loss in leaching_runoff_erosion

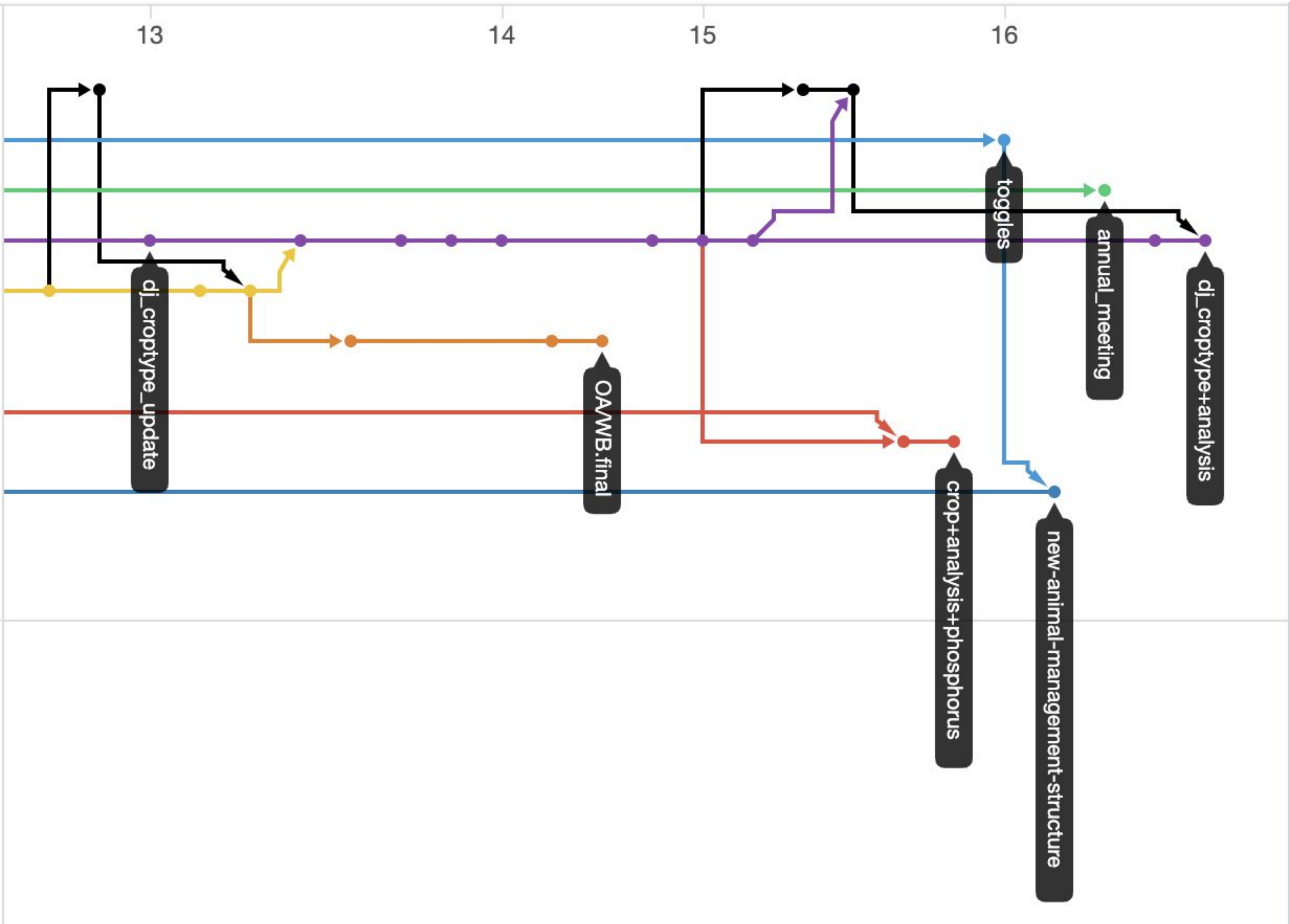
- **Heat Units**
  - Available Heat
  - FrPHU (Fraction of Potential He...
- **Root Development**
  - Root development in the soil ✕
- **Water Uptake**
  - Maximum Potential Water Upta...
  - Impact of low soil water conten...
  - Actual water uptake
- **Nitrogen Uptake**
  - Calculate shape coefficients for...
  - Plant Nitrogen Demand
  - Plant nitrogen uptake from soil

- **Phosphorus Uptake** ✕
  - Calculate shape coefficients for...
  - Plant Phosphorus Demand
  - Plant Phosphorus uptake from ...
- **Growth Constraints**
  - Water stress
  - Temperature stress
  - Nitrogen stress
  - Phosphorus stress
  - Growth Factor
- **Leaf Area Index**
  - Calculate Leaf Area Index (LAI)
- **Biomass**

- **Biomass**
  - Calculate Maximum potent...
- **Yields**
  - Calculate water deficiency
  - Potential Harvest Index
  - Aboveground Biomass
  - Actual Harvest Index
  - Potential Yield
  - Actual Yield
  - Nutrient Removal
  - Residue

Team Development




13
14
15
16
dj_croptype_update
OAWB.final
toggles
annual_meeting
dj_croptype+analysis
crop+analysis+phosphorus
new-animal-management-structure