Applying context-free grammar to hierarchically organized and variably shaped arrays

Robert Schweppe, Stephan Thober, Sebastian Müller, Luis Samaniego

FortranCon 2020 - virtual on Zoom

2020-07-02

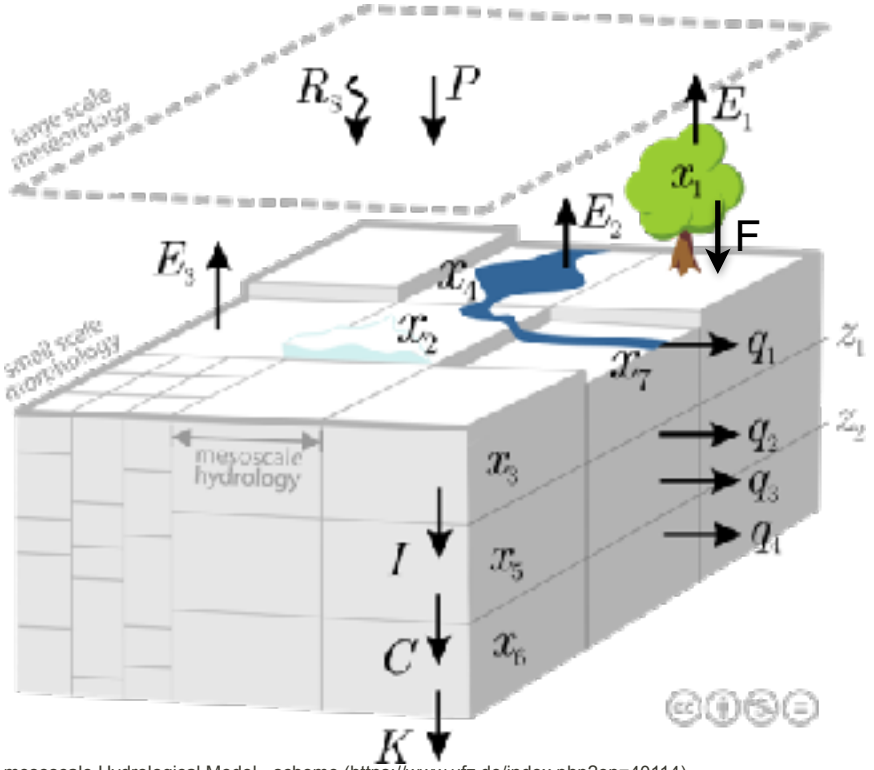
robert.schweppe@ufz.de

Our Fortran background
field: hydrological modelling

- Fortran development since group foundation
- Current topics:
    - Environmental model development
    - Hybrid OpenMP/MPI tree parallelization
    - Unit testing
    - build automation (CMake), CI
- Mostly scientific, some engineering background

Fortran Boardgame Image from https://insidehpc.com/

Environmental modelling
model internals

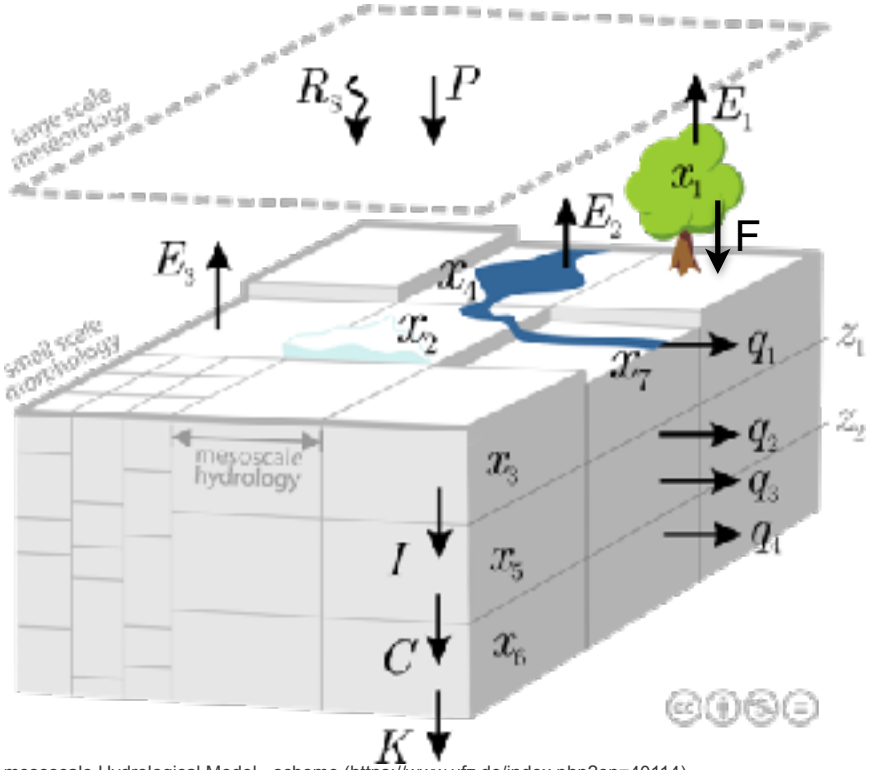


mesoscale Hydrological Model - scheme (https://www.ufz.de/index.php?en=40114)

Environmental modelling
model internals

- Process representation, e.g.:

$$\Delta S_i(t) = P_i(t) - E_i(t) - Q_i(t)$$



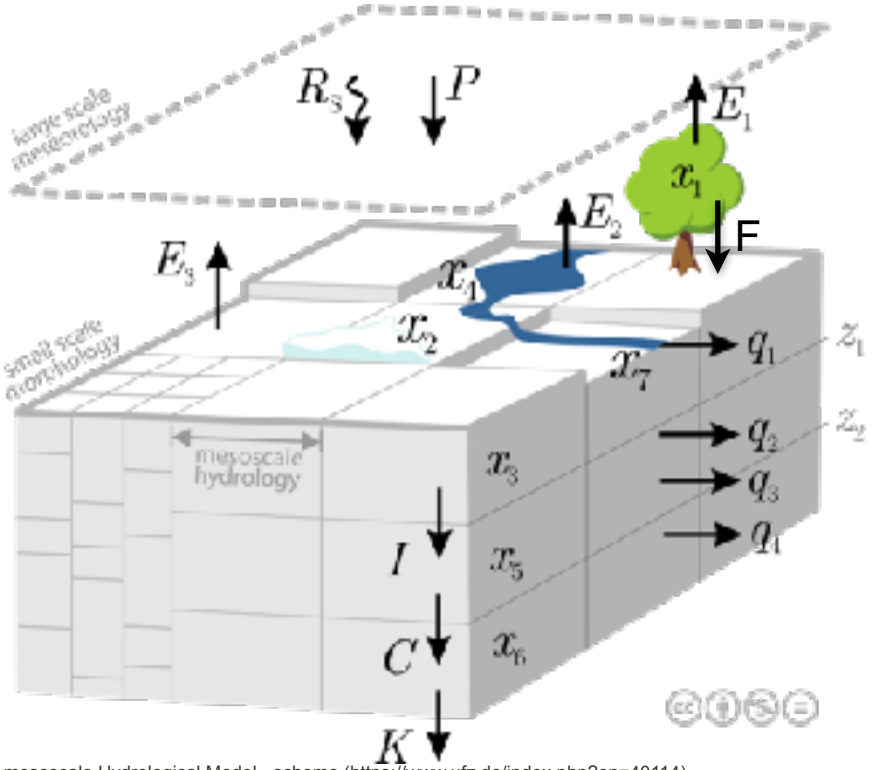
mesoscale Hydrological Model - scheme (https://www.ufz.de/index.php?en=40114)

Environmental modelling
model internals

- Process representation, e.g.:

$$\Delta S_i(t) = P_i(t) - E_i(t) - Q_i(t)$$

$$x_{1i} = P_i(t) - E_{1i}(t) - F_i(t)$$



mesoscale Hydrological Model - scheme (https://www.ufz.de/index.php?en=40114)

Environmental modelling
model internals

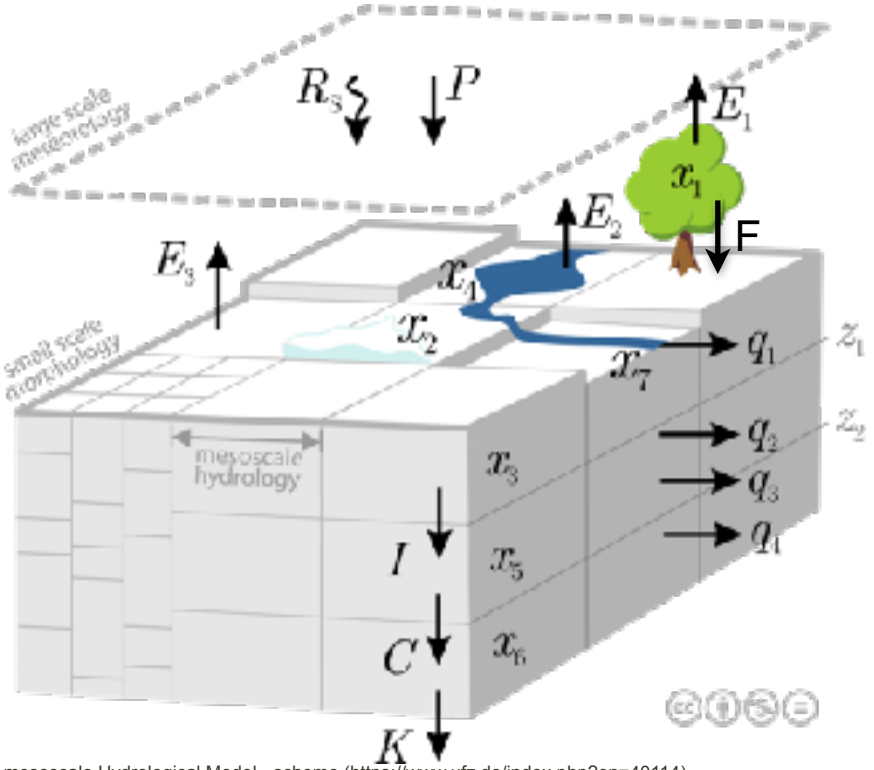
- Process representation, e.g.:

$$\Delta S_i(t) = P_i(t) - E_i(t) - Q_i(t)$$

$$x_{1i} = P_i(t) - E_{1i}(t) - F_i(t)$$

- More specifically, e.g.:

$$F_i(t) = max\{P_i(t) + x_1 i(t-1) - \beta_{1i}(t), 0\}$$



mesoscale Hydrological Model - scheme (https://www.ufz.de/index.php?en=40114)

Environmental modelling
model internals

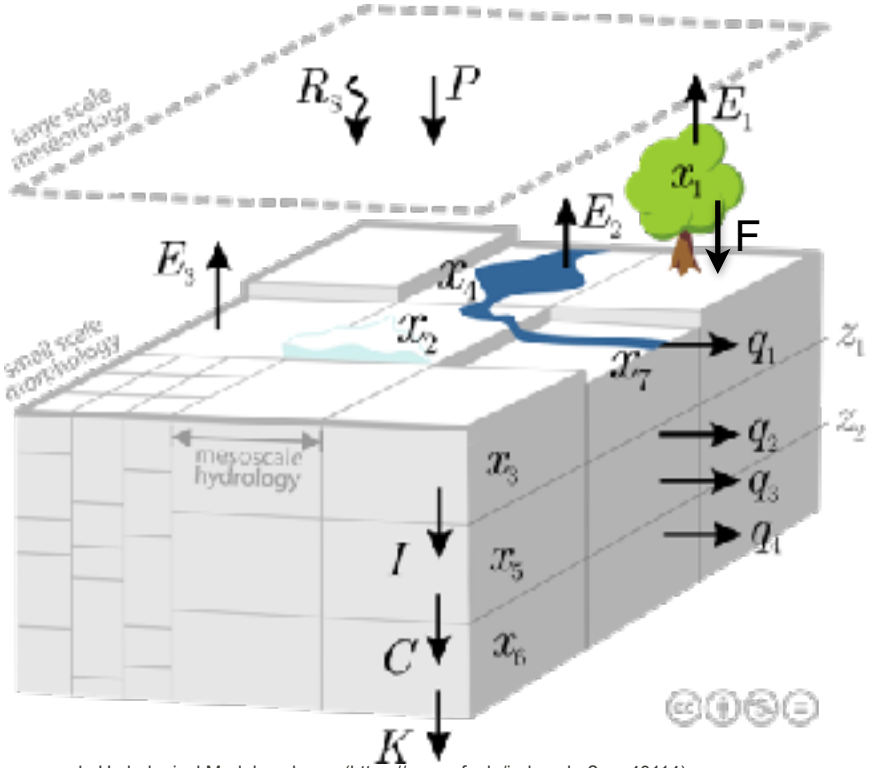
- Process representation, e.g.:

$$\Delta S_i(t) = P_i(t) - E_i(t) - Q_i(t)$$

$$x_{1i} = P_i(t) - E_{1i}(t) - F_i(t)$$

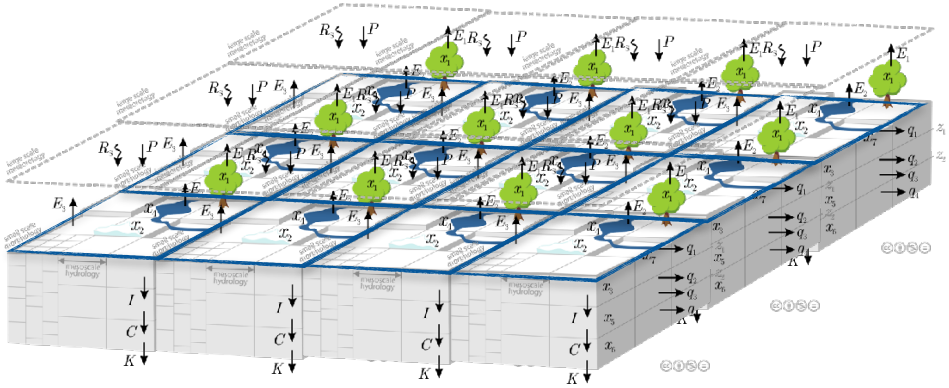
- More specifically, e.g.:

$$F_i(t) = max\{P_i(t) + x_1 i(t-1) - \beta_{1i}(t), 0\}$$



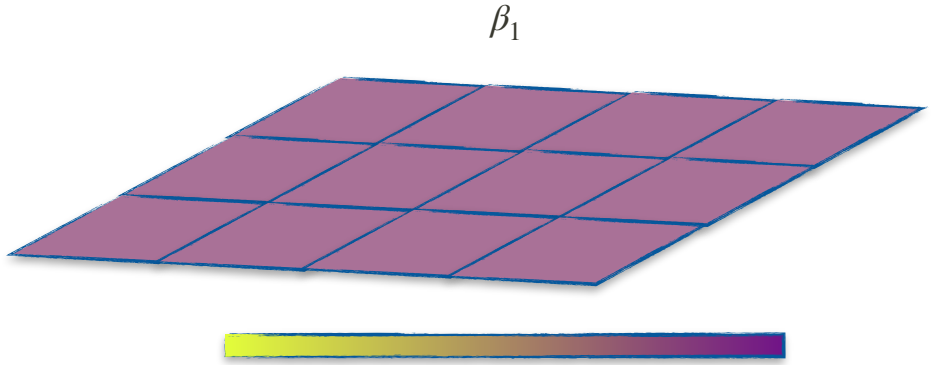
mesoscale Hydrological Model - scheme (https://www.ufz.de/index.php?en=40114)

Environmental modelling
spatially distributed modelling



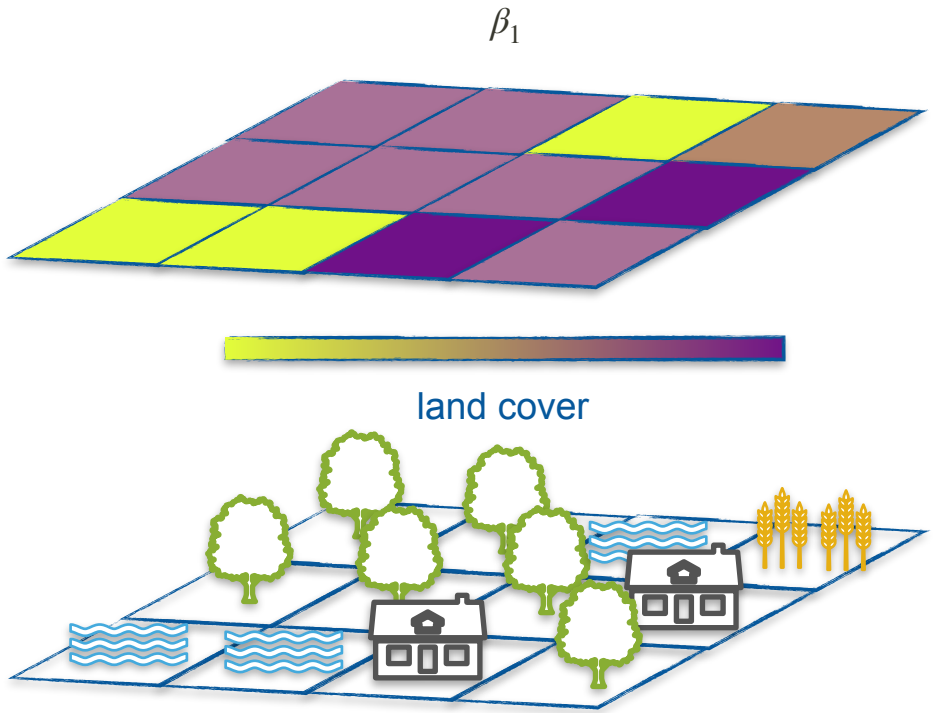
Environmental modelling
parameter estimation methods

- Constant value



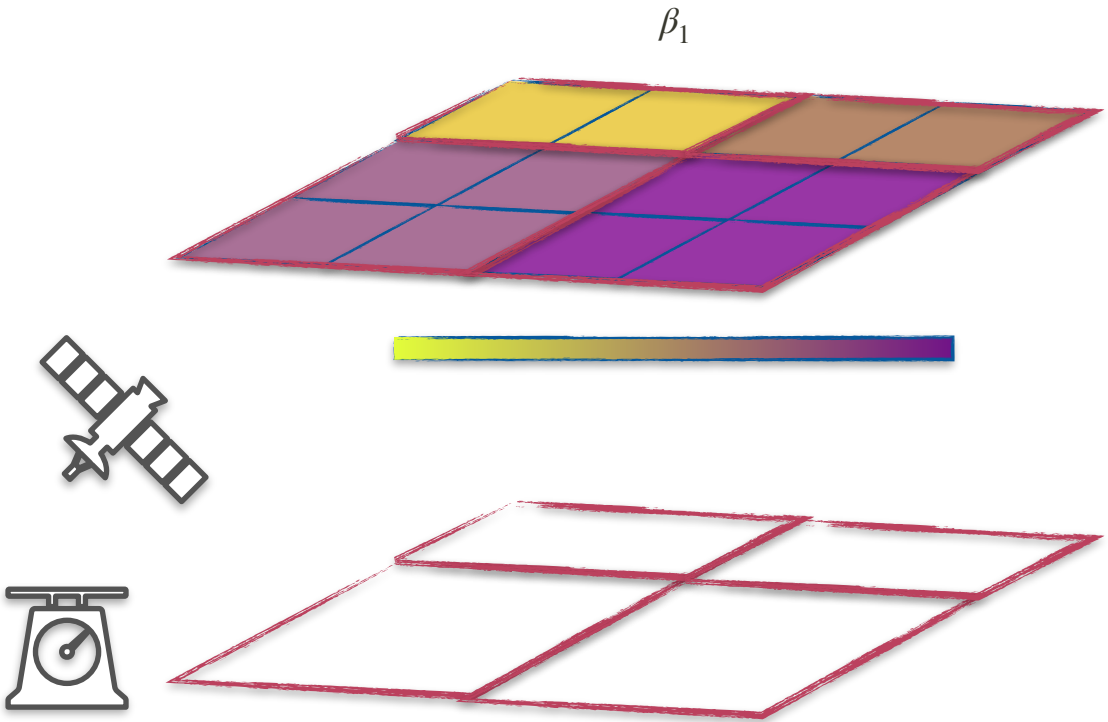
Environmental modelling
parameter estimation methods

- Constant value
- Lookup-table



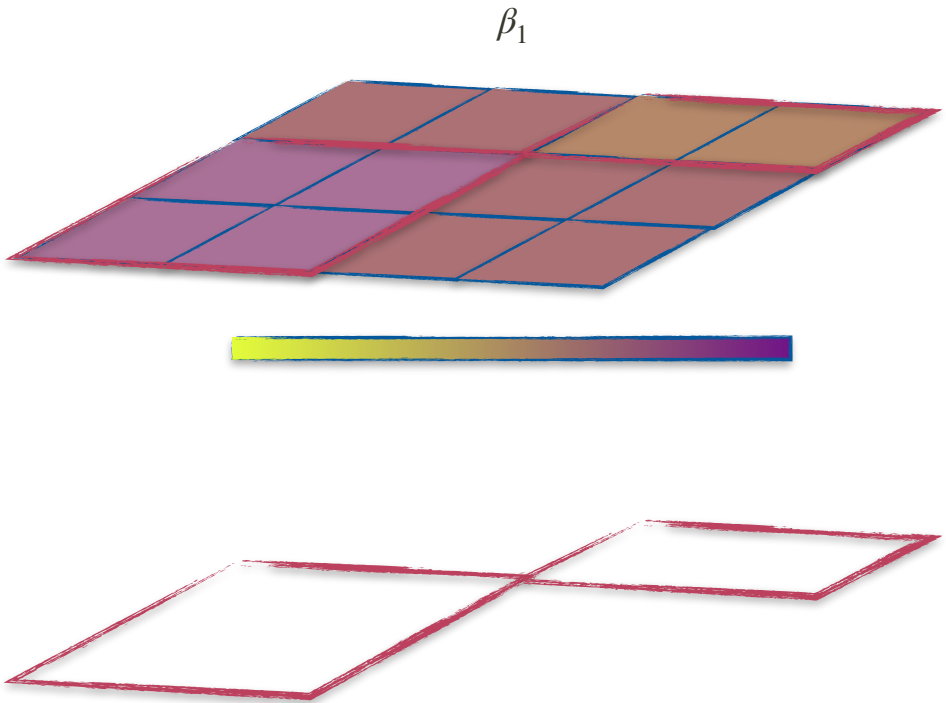
Environmental modelling
parameter estimation methods

- Constant value
- Lookup-table
- Calibration



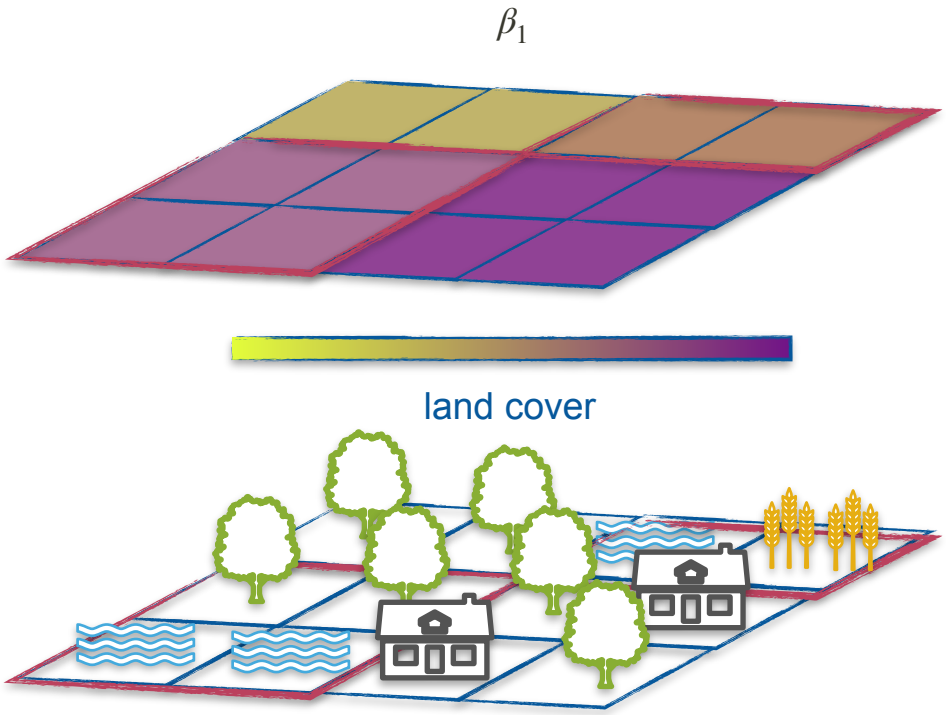
Environmental modelling
parameter estimation methods

- Constant value
- Lookup-table
- Calibration and extrapolation …
    - Based on proximity



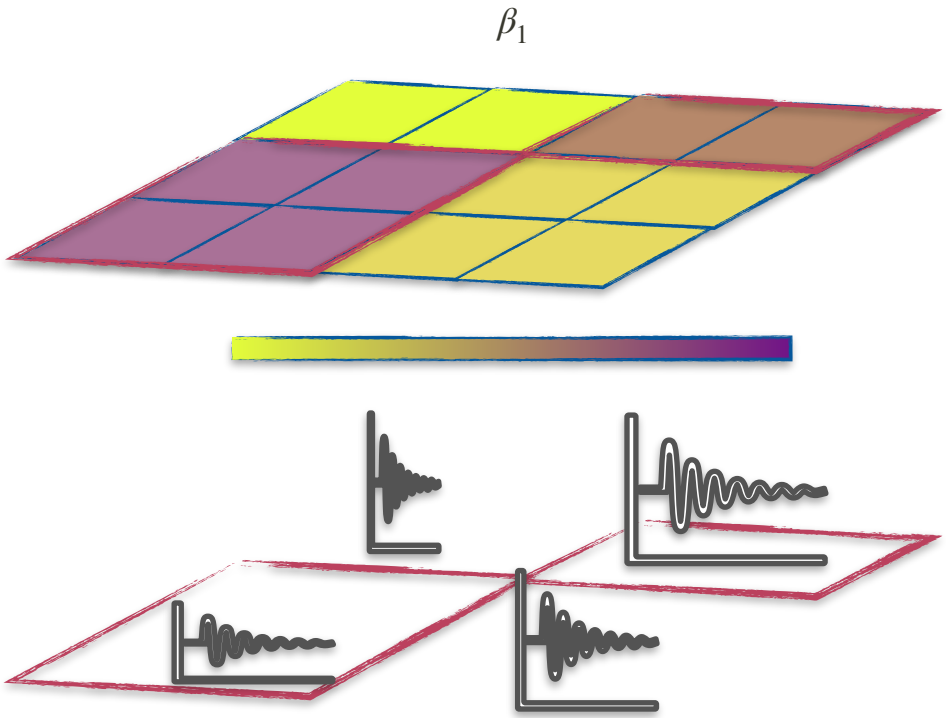
Environmental modelling
parameter estimation methods

- Constant value
- Lookup-table
- Calibration and extrapolation …
  - Based on proximity
  - Based on physiographic properties



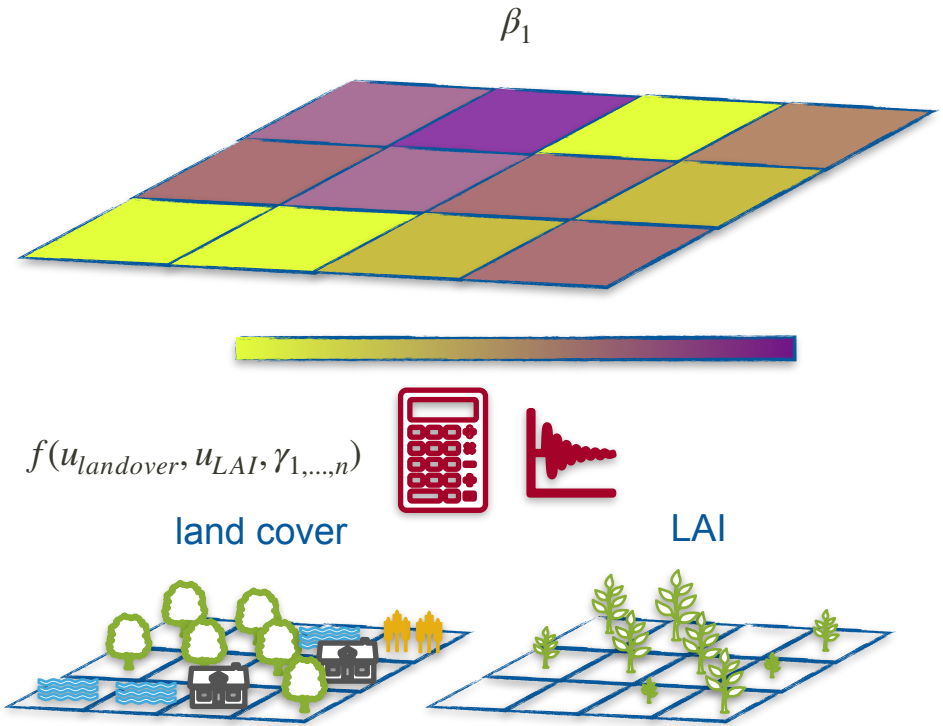
Environmental modelling
parameter estimation methods

- Constant value
- Lookup-table
- Calibration and extrapolation …
  - Based on proximity
  - Based on physiographic properties
  - Based on signatures

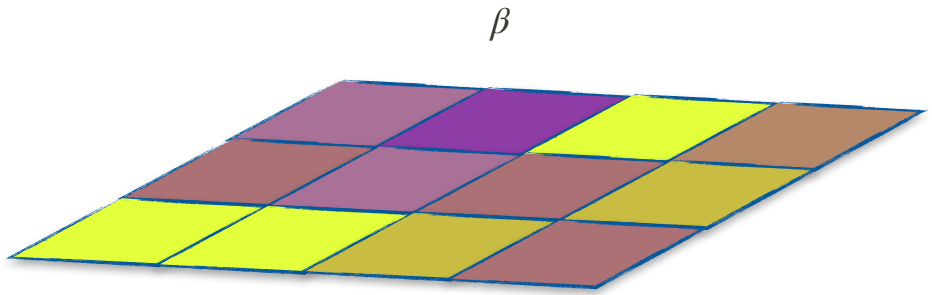


Environmental modelling
parameter estimation methods

- Constant value
- Lookup-table
- Calibration and extrapolation …
  - Based on proximity
  - Based on physiographic properties
  - Based on signatures
- Regression (and calibration)

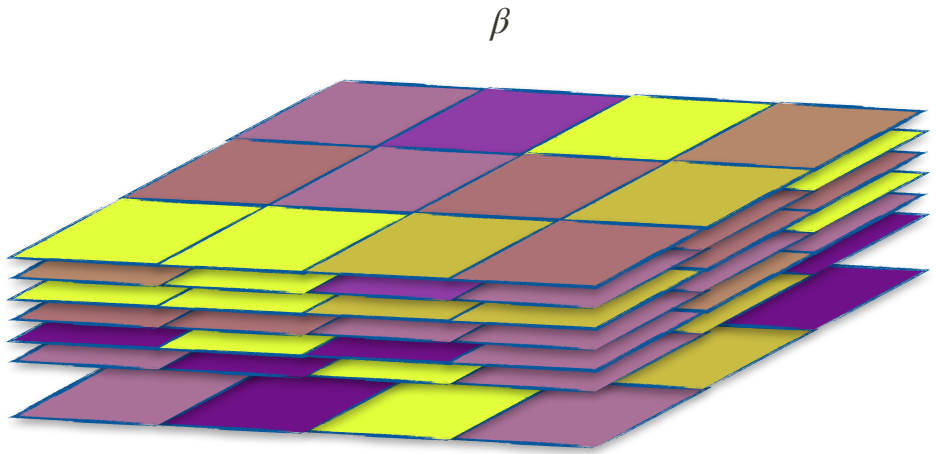


Environmental modelling
parameter estimation



Environmental modelling
parameter estimation

$$n_\beta = 28i$$

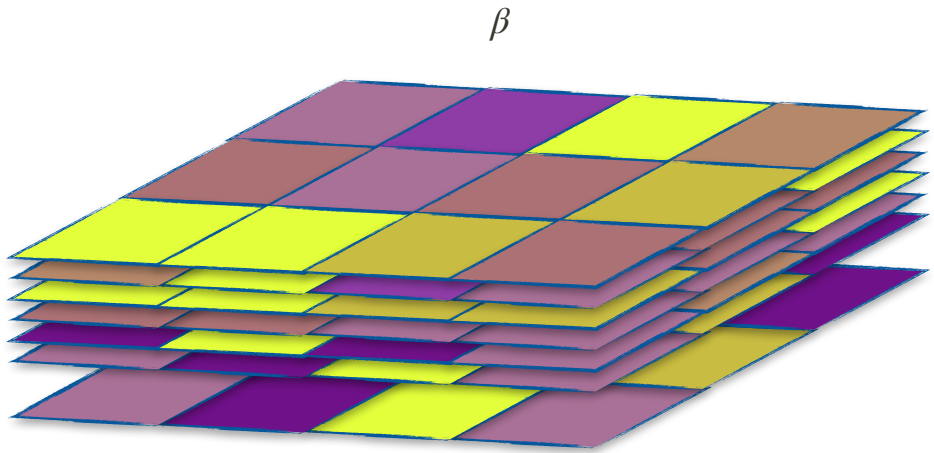


Environmental modelling
parameter estimation

$$n_\beta = 28i$$

- This example

$$n_\beta = 392$$



Environmental modelling
parameter estimation

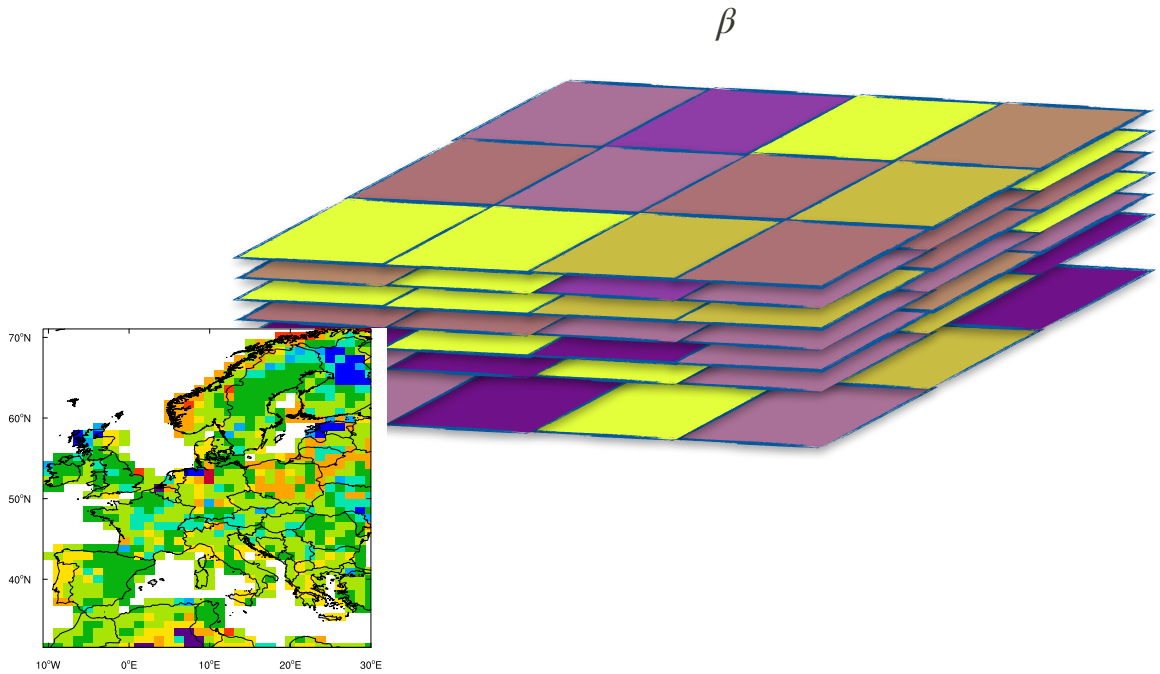
$$n_\beta = 28i$$

- This example

$$n_\beta = 392$$

- Europe at 1 degree resolution

$$n_\beta = 2.2 * 10^4$$



Environmental modelling
parameter estimation

$$n_\beta = 28i$$

- This example

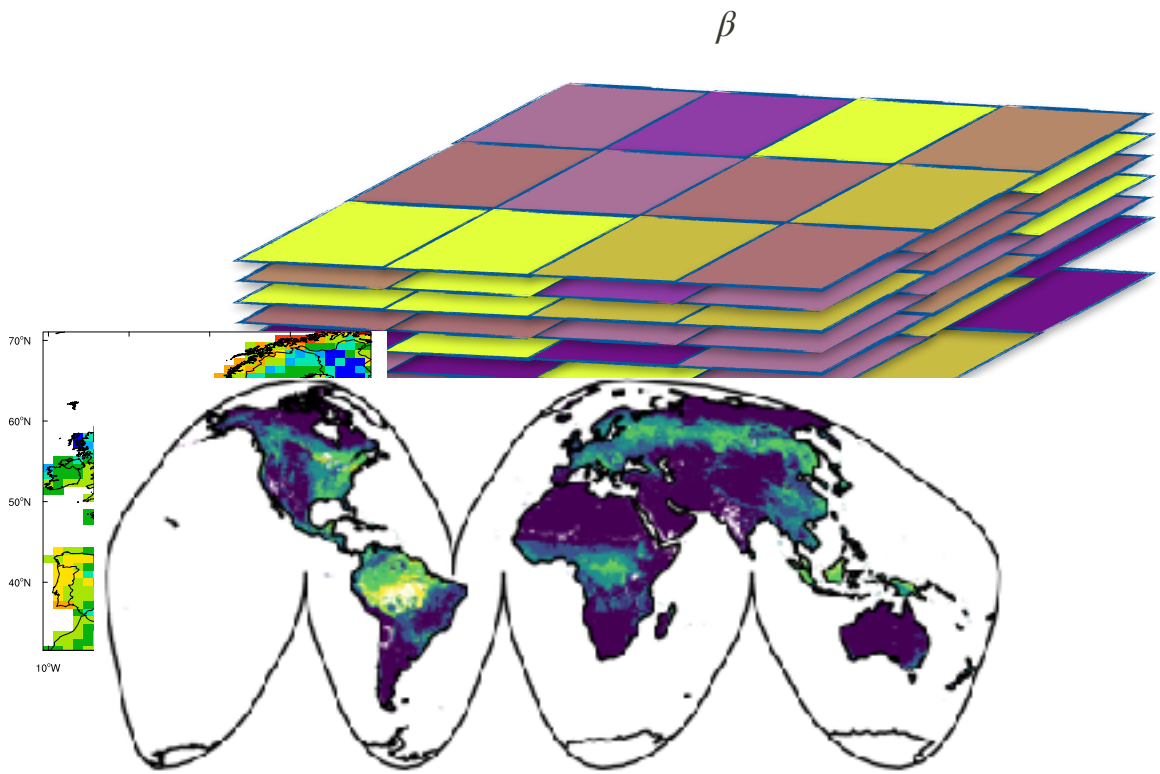
  $$n_\beta = 392$$

- Europe at 1 degree resolution

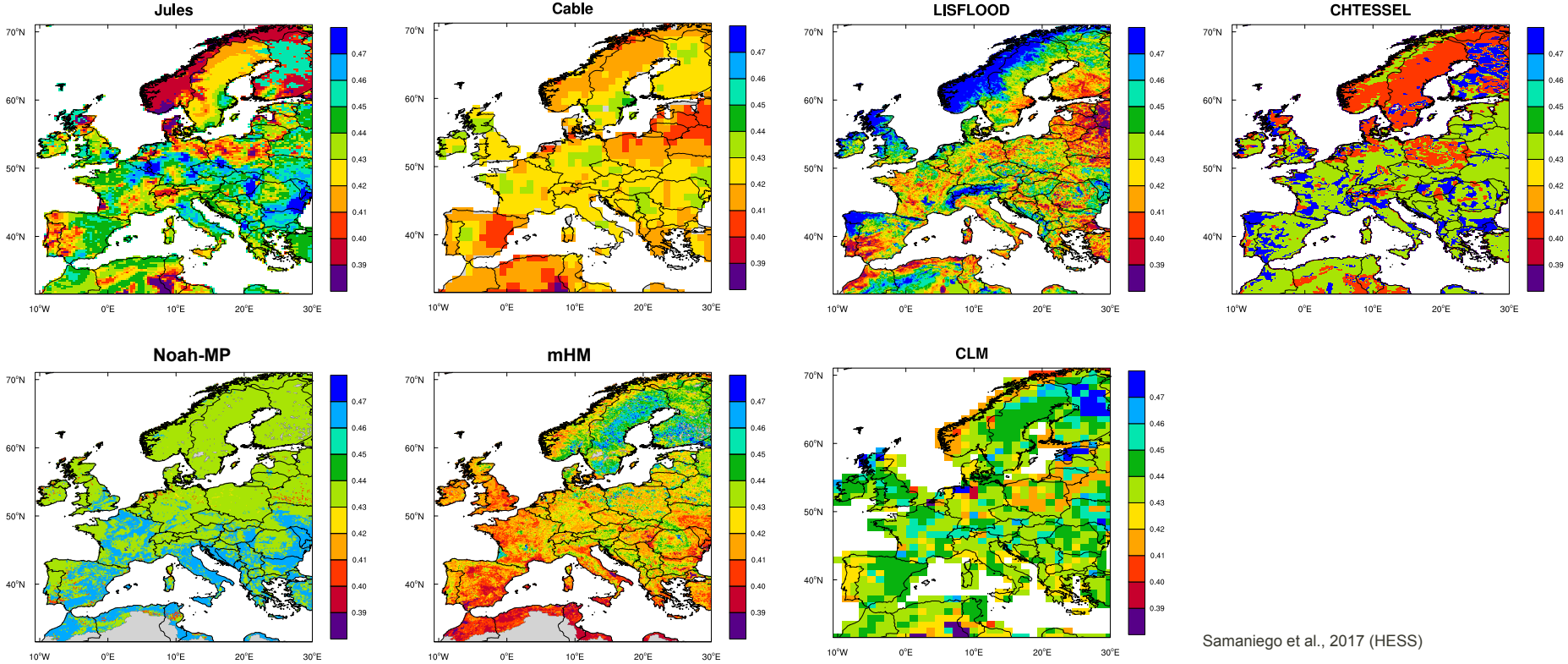
  $$n_\beta = 2.2 * 10^4$$

- Globe at 0.1 degree resolution

  $$n_\beta = 5.4 * 10^7$$



Environmental modelling
parameter distribution in land-surface models

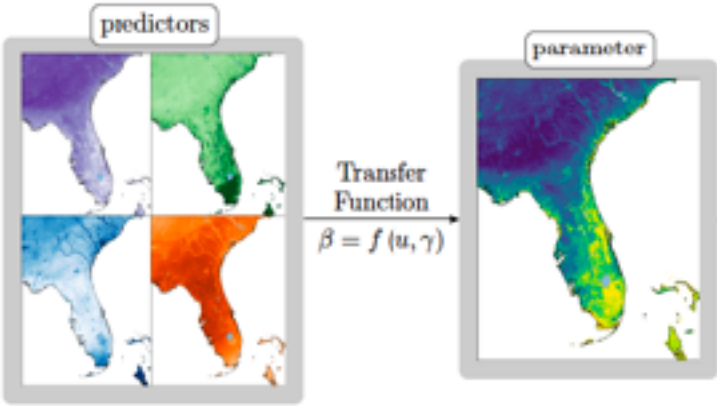


Samaniego et al., 2017 (HESS)

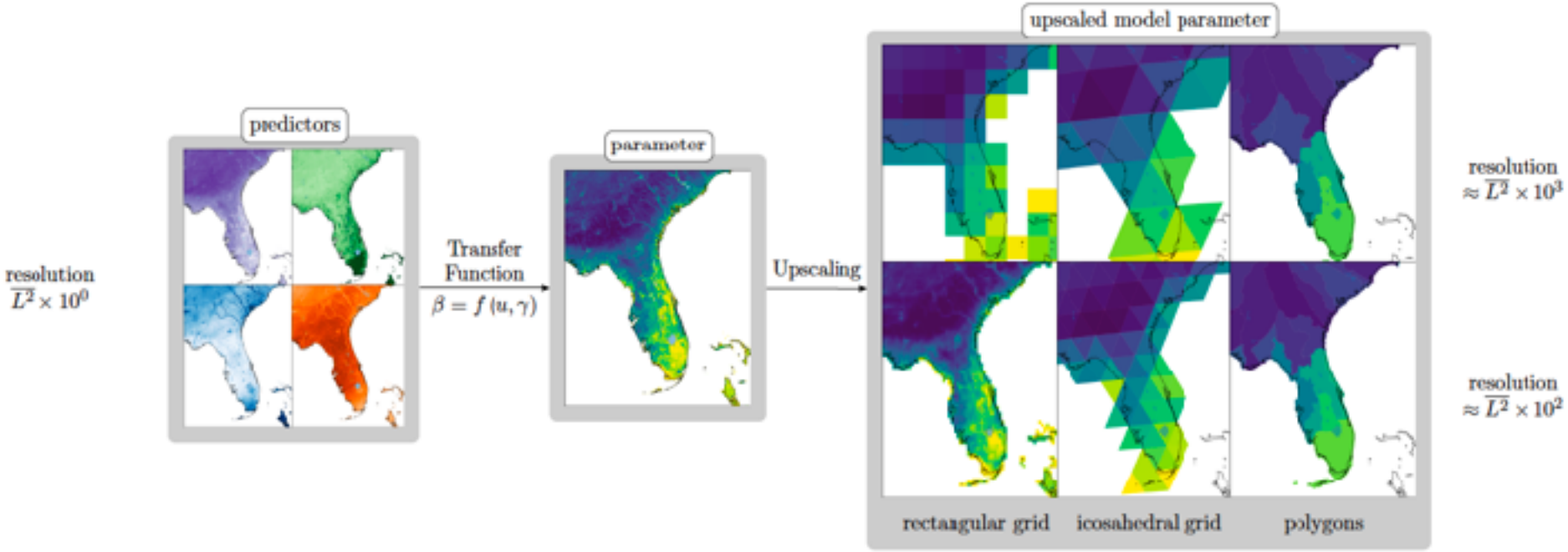
MPR
schematic overview

MPR
schematic overview

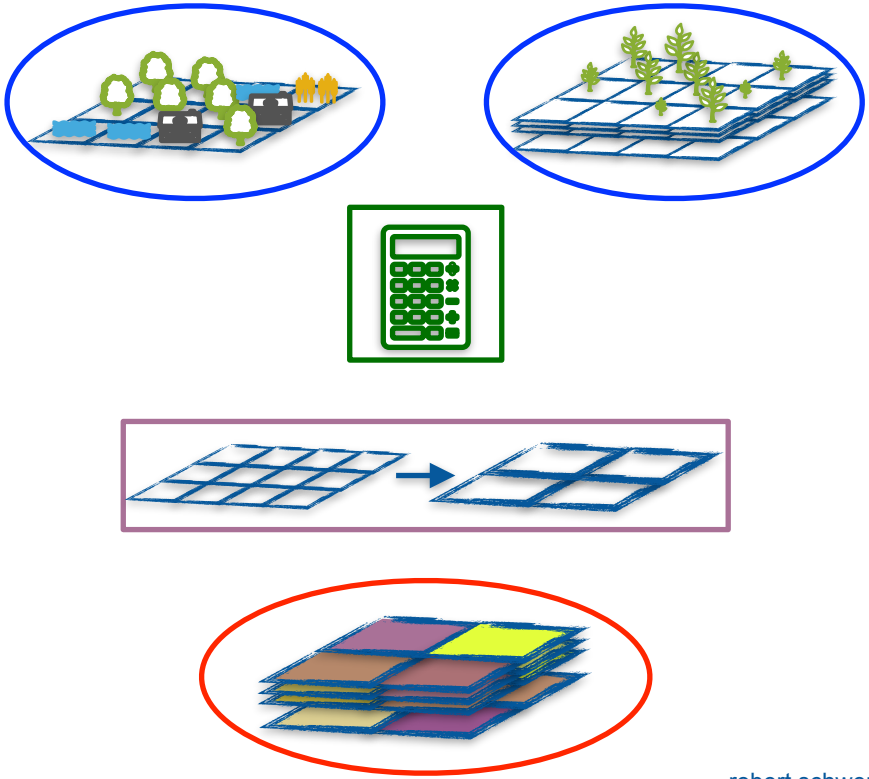
resolution
$\overline{L^2} \times 10^0$



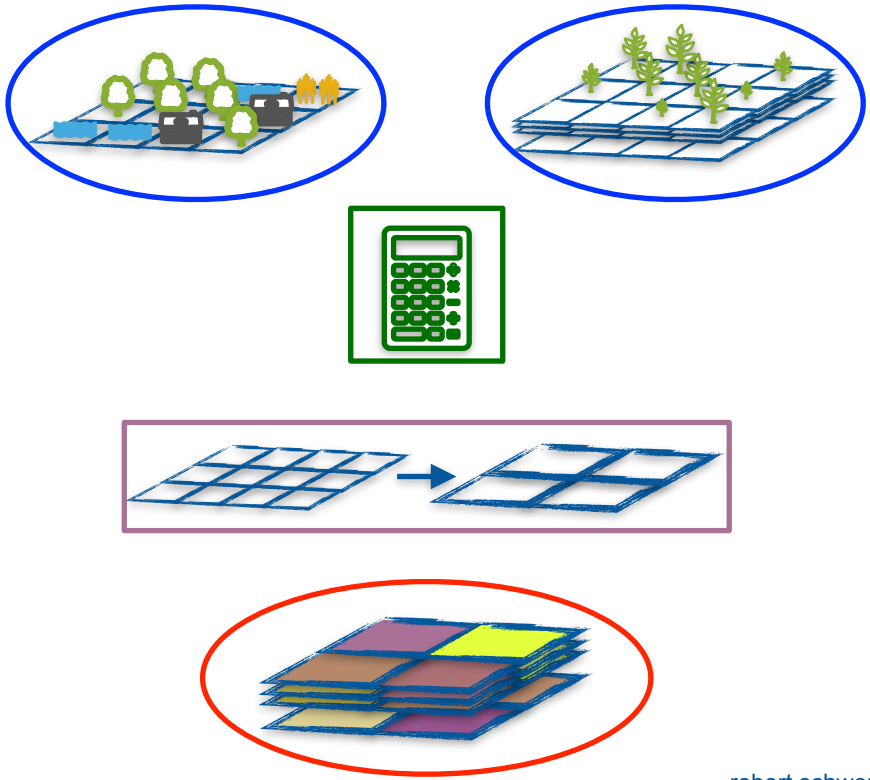
MPR
schematic overview



MPR configuration
an example



MPR configuration
an example



namelist configuration

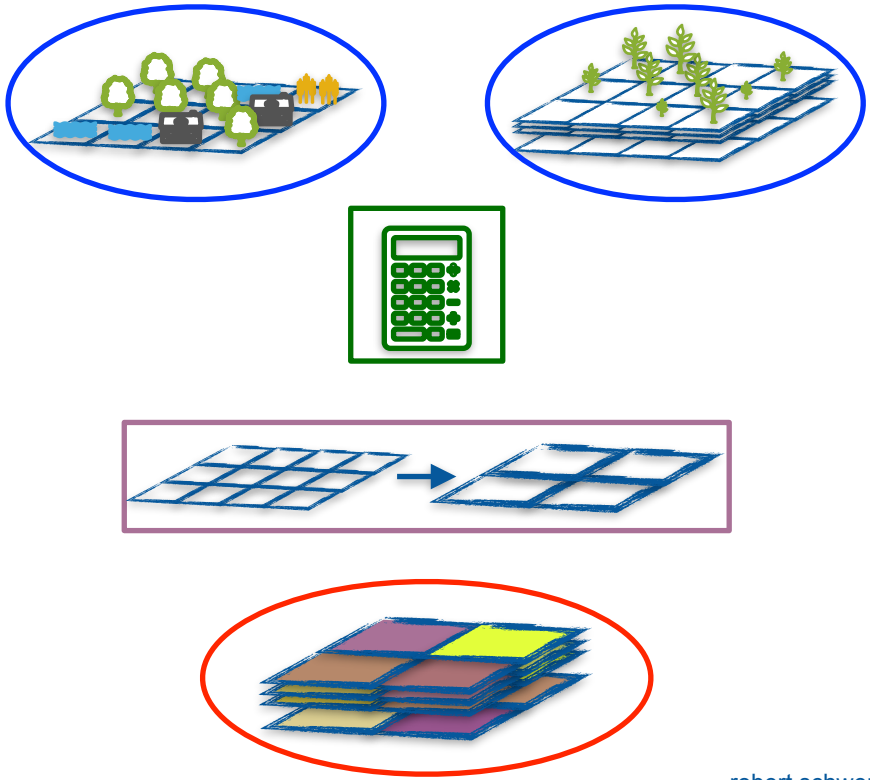
```
&main
out_filename = "MyParams.nc"
coordinate_aliases(:,1) = “x_in", "x_out"
coordinate_aliases(:,2) = “y_in", "y_out"
coordinate_aliases(:,3) = “t_in", "t_out"
/

&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out", "x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/

&Coordinates
coord_names(1:3) = "x_out", "y_out", "t_out"
coord_stagger(1:3) = "center", "center", "end"
coord_from_range_step(1:2) = 0.125, 0.125
coord_from_values_bound(3) = 0.0
coord_from_values(:,3) = 3.0, 6.0, 9.0, 12.0
/

&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

MPR configuration
an example

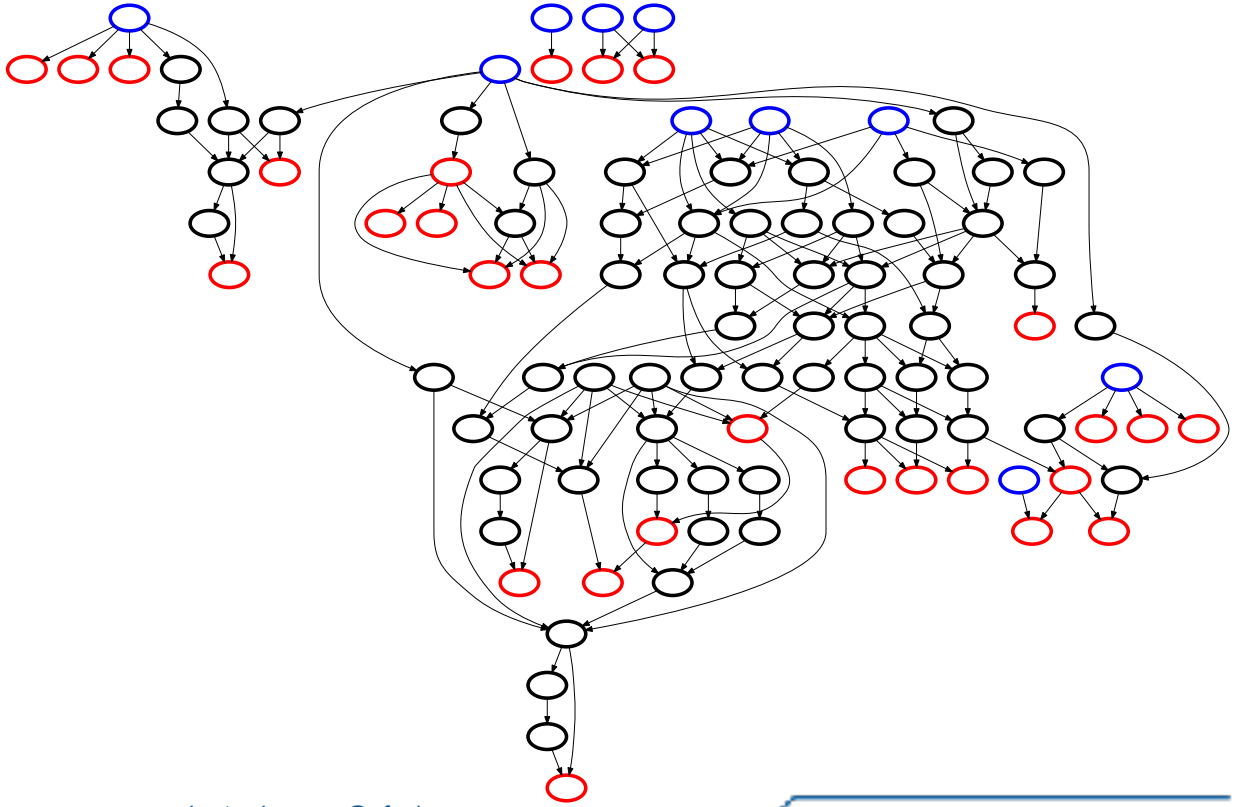


namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

MPR
application in mesoscale Hydrological Model

predictor variables	11
global parameters	52
model parameters	28



Library Requirements

Library Requirements

Requirement	Specification
capability	• standalone executable • flexible API for coupling • transfer functions: context-free grammar

Library Requirements

Requirement	Specification
capability	• standalone executable • flexible API for coupling • transfer functions: context-free grammar
usability	• few dependencies • simple setup/configuration

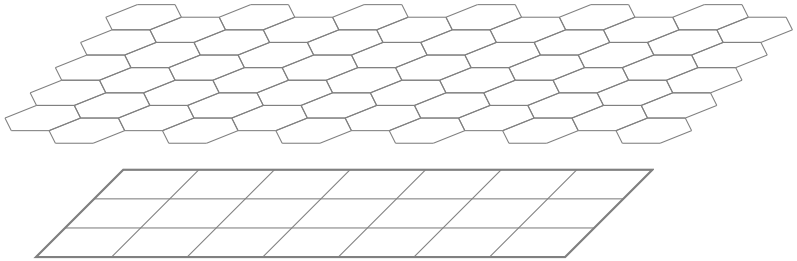
Library Requirements

Requirement	Specification
capability	• standalone executable • flexible API for coupling • transfer functions: context-free grammar
usability	• few dependencies • simple setup/configuration
I/O	• support of netCDF4

Library Requirements

Requirement	Specification
capability	• standalone executable • flexible API for coupling • transfer functions: context-free grammar
usability	• few dependencies • simple setup/configuration
I/O	• support of netCDF4
data handling	• efficient, reusable types • object-orientation

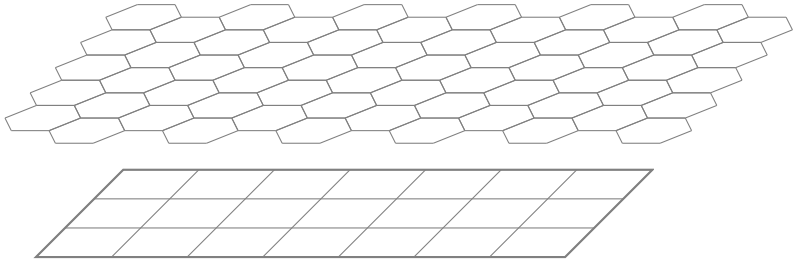
MPR
features for coordinate handling



MPR features for coordinates:

- first-order conservative remapping

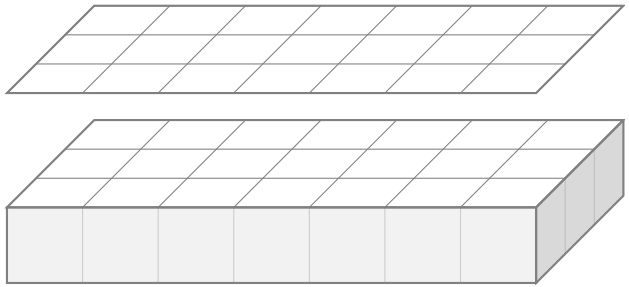
MPR
features for coordinate handling



MPR features for coordinates:

- first-order conservative remapping
- coordinate-specific aggregation function
- remapping of irregular shapes

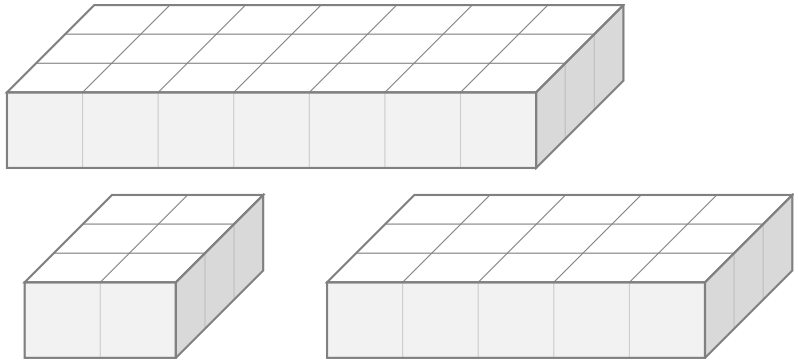
MPR
features for coordinate handling



MPR features for coordinates:

- first-order conservative remapping
- coordinate-specific aggregation function
- remapping of irregular shapes
- broadcasting

MPR
features for coordinate handling



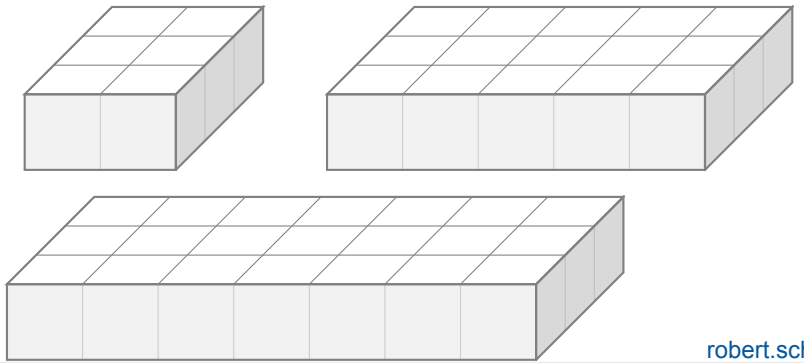
MPR features for coordinates:

- first-order conservative remapping
- coordinate-specific aggregation function
- remapping of irregular shapes
- broadcasting
- splitting

MPR
features for coordinate handling

MPR features for coordinates:

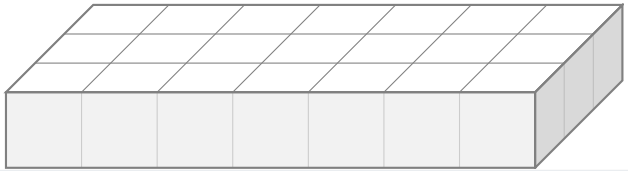
- first-order conservative remapping
- coordinate-specific aggregation function
- remapping of irregular shapes
- broadcasting
- splitting
- concatenation

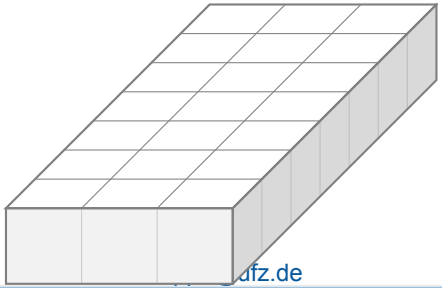


MPR
features for coordinate handling

MPR features for coordinates:

- first-order conservative remapping
- coordinate-specific aggregation function
- remapping of irregular shapes
- broadcasting
- splitting
- concatenation
- transposing





Implementation details
Transfer Function

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

Fortran code

```
elemental function f1(x1, x2, a, b, c) result(y)
  real, intent(in) :: x1, x2
  real, intent(in) :: a, b, c
  real :: y


  y = a + b * x1 + c * x2

end function f1


subroutine caller()
  real, dimension(:,:,:) :: LAI, land_cover, y
  real :: a, b, c

  beta1 = f1(LAI, land_cover, a, b, c)


end subroutine caller
```

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

Fortran code

```
elemental function f1(x1, x2, a, b, c) result(y)
  real, intent(in) :: x1, x2
  real, intent(in) :: a, b, c
  real :: y

  y = a + b * x1 + c * x2

end function f1

subroutine caller()
  real, dimension(:,:,:) :: LAI, land_cover, y
  real :: a, b, c

  beta1 = f1(LAI, land_cover, a, b, c)

end subroutine caller
```

rank?

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

Fortran code

```
elemental function f1(x1, x2, a, b, c) result(y)
  real, intent(in) :: x1, x2
  real, intent(in) :: a, b, c
  real :: y

  y = a + b * x1 + c * x2

end function f1

subroutine caller()
  real, dimension(:,:,:) :: LAI, land_cover, y
  real :: a, b, c

  beta1 = f1(LAI, land_cover, a, b, c)

end subroutine caller
```

rank?
type?

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

Fortran code

```
elemental function f1(x1, x2, a, b, c) result(y)
  real, intent(in) :: x1, x2
  real, intent(in) :: a, b, c
  real :: y

  y = a + b * x1 + c * x2

end function f1

subroutine caller()
  real, dimension(:,:,:) :: LAI, land_cover, y
  real :: a, b, c

  beta1 = f1(LAI, land_cover, a, b, c)

end subroutine caller
```

rank?
type?
non-elemental?

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

Fortran code

```
elemental function f1(x1, x2, a, b, c) result(y)
  real, intent(in) :: x1, x2
  real, intent(in) :: a, b, c
  real :: y

  y = a + b * x1 + c * x2

end function f1

subroutine caller()
  real, dimension(:,:,:) :: LAI, land_cover, y
  real :: a, b, c

  beta1 = f1(LAI, land_cover, a, b, c)

end subroutine caller
```

rank?
type?
non-elemental?
interface?

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

Fortran code

```
elemental function f1(x1, x2, a, b, c) result(y)
  real, intent(in) :: x1, x2
  real, intent(in) :: a, b, c
  real :: y

  y = a + b * x1 + c * x2

end function f1

subroutine caller()
  real, dimension(:,:,:) :: LAI, land_cover, y
  real :: a, b, c

  beta1 = f1(LAI, land_cover, a, b, c)

end subroutine caller
```

type?
non-elemental?
interface?

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

type?
non-elemental?
interface?

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

Fortran code

```
type Da (…) # p = packed real array
type Pa (…) # x = real scalar
```

type?
non-elemental?
interface?

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

Fortran code

```
type Da (…) # p = packed real array
type Pa (…) # x = real scalar

function f1(x, p) result(y)
  type(Da), intent(in) :: x(:)
  type(Pa), intent(in) :: p(:)
  real, allocatable :: y(:)

  allocate(y(size(x(1)%x))
  y = p(1)%p + p(2)%p * x(1)%x + p(3)%p * x(2)%x

end function f1
```

type?
non-elemental?
interface?

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

Fortran code

```
type Da (…) # p = packed real array
type Pa (…) # x = real scalar

function f1(x, p) result(y)
  type(Da), intent(in) :: x(:)
  type(Pa), intent(in) :: p(:)
  real, allocatable :: y(:)

  allocate(y(size(x(1)%x))
  y = p(1)%p + p(2)%p * x(1)%x + p(3)%p * x(2)%x

end function f1


subroutine caller()
  real, dimension(:,:,:) :: LAI, land_cover, y
  real :: a, b, c

  y = f1([Da(LAI), Da(land_cover)],
         [Pa(a), Pa(b), Pa(c)])

end subroutine caller
```

type?
non-elemental?
interface?

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

Fortran code

```
type Da (…) # p = packed real array
type Pa (…) # x = real scalar

function f1(x, p) result(y)
  type(Da), intent(in) :: x(:)
  type(Pa), intent(in) :: p(:)
  real, allocatable :: y(:)

  allocate(y(size(x(1)%x))
  y = p(1)%p + p(2)%p * x(1)%x + p(3)%p * x(2)%x

end function f1


subroutine caller()
  real, dimension(:,:,:) :: LAI, land_cover, y
  real :: a, b, c

  y = f1([Da(LAI), Da(land_cover)],
         [Pa(a), Pa(b), Pa(c)])

end subroutine caller
```

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

Fortran code

```
type Da (…) # p = packed real array
type Pa (…) # x = real scalar

function f1(x, p) result(y)
  type(Da), intent(in) :: x(:)
  type(Pa), intent(in) :: p(:)
  real, allocatable :: y(:)

  allocate(y(size(x(1)%x))
  y = p(1)%p + p(2)%p * x(1)%x + p(3)%p * x(2)%x

end function f1


subroutine caller()
  real, dimension(:,:,:) :: LAI, land_cover, y
  real :: a, b, c

  y = f1([Da(LAI), Da(land_cover)],
         [Pa(a), Pa(b), Pa(c)])

end subroutine caller
```

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

Fortran code

```
type Da (…) # p = packed real array
type Pa (…) # x = real scalar

function f1(x, p) result(y)
  type(Da), intent(in) :: x(:)
  type(Pa), intent(in) :: p(:)
  real, allocatable :: y(:)

  allocate(y(size(x(1)%x))
  y = p(1)%p + p(2)%p * x(1)%x + p(3)%p * x(2)%x

end function f1


subroutine caller()
  real, dimension(:,:,:) :: LAI, land_cover, y
  real :: a, b, c

  y = f1([Da(LAI), Da(land_cover)],
         [Pa(a), Pa(b), Pa(c)])

end subroutine caller
```

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
upscale_ops(1:3,4) = "1.0", "1.0", "1.0"
/
&Parameters
parameter_names(1:3) = "a", "b", "c"
parameter_values(1:3) = 0.5, -0.3, 1.2
/
```

Implementation details
Transfer Function

Fortran code

```
type Da (…) # p = packed real array
type Pa (…) # x = real scalar

function f1(x, p) result(y)
  type(Da), intent(in) :: x(:)
  type(Pa), intent(in) :: p(:)
  real, allocatable :: y(:)

  allocate(y(size(x(1)%x))
  y = p(1)%p + p(2)%p * x(1)%x + p(3)%p * x(2)%x

end function f1


subroutine caller()
  real, dimension(:,:,:) :: LAI
  real :: a, b, c

  y = f1([Da(LAI), Da(land_cove
         [Pa(a), Pa(b), Pa(c)])

end subroutine caller
```

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
                          = "1.0", "1.0", "1.0"


                        ) = "a", "b", "c"
                        3) = 0.5, -0.3, 1.2
```

- simple string search & replace operations

Implementation details
Transfer Function

Fortran code

```
type Da (…) # p = packed real array
type Pa (…) # x = real scalar

function f1(x, p) result(y)
  type(Da), intent(in) :: x(:)
  type(Pa), intent(in) :: p(:)
  real, allocatable :: y(:)

  allocate(y(size(x(1)%x))
  y = p(1)%p + p(2)%p * x(1)%x + p(3)%p * x(2)%x

end function f1


subroutine caller()
  real, dimension(:,:,:) :: LAI
  real :: a, b, c

  y = f1([Da(LAI), Da(land_cove
         [Pa(a), Pa(b), Pa(c)])

end subroutine caller
```

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
                          = "1.0", "1.0", "1.0"

                        ) = "a", "b", "c"
                        3) = 0.5, -0.3, 1.2
```

- simple string search & replace operations
- *real conversion necessary (templating?)*

Implementation details
Transfer Function

Fortran code

```
type Da (…) # p = packed real array
type Pa (…) # x = real scalar

function f1(x, p) result(y)
  type(Da), intent(in) :: x(:)
  type(Pa), intent(in) :: p(:)
  real, allocatable :: y(:)

  allocate(y(size(x(1)%x))
  y = p(1)%p + p(2)%p * x(1)%x + p(3)%p * x(2)%x

end function f1


subroutine caller()
  real, dimension(:,:,:) :: LAI
  real :: a, b, c

  y = f1([Da(LAI), Da(land_cove
         [Pa(a), Pa(b), Pa(c)])

end subroutine caller
```

namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
                          = "1.0", "1.0", "1.0"

                        ) = "a", "b", "c"
                        3) = 0.5, -0.3, 1.2
```

- simple string search & replace operations
- *real conversion necessary (templating?)*
- *grammar? (a+b == b+a)*

Implementation details
Transfer Function

Fortran code

```
type Da (…) # p = packed real array
type Pa (…) # x = real scalar

function f1(x, p) result(y)
  type(Da), intent(in) :: x(:)
  type(Pa), intent(in) :: p(:)
  real, allocatable :: y(:)

  allocate(y(size(x(1)%x))
  y = p(1)%p + p(2)%p * x(1)%x + p(3)%p * x(2)%x

end function f1


subroutine caller()
  real, dimension(:,:,:) :: LAI
  real :: a, b, c

  y = f1([Da(LAI), Da(land_cove
         [Pa(a), Pa(b), Pa(c)])

end subroutine caller
```

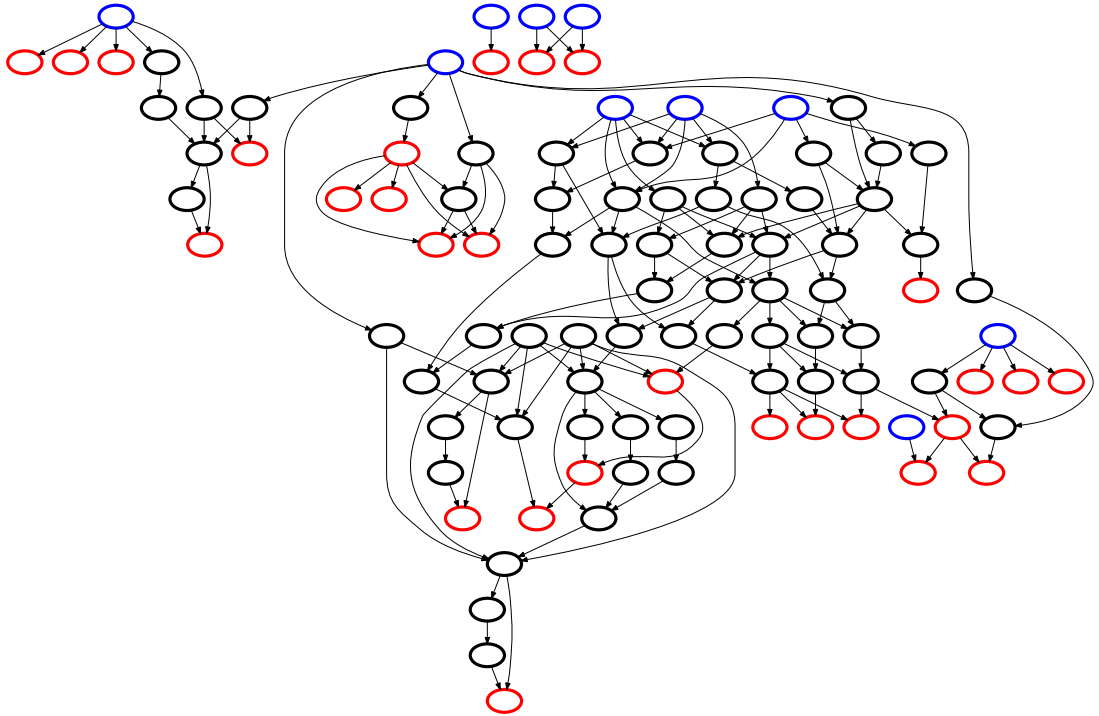
namelist configuration

```
(…)
&Data_Arrays
names(1) = "land_cover"
from_file(1) = "PathTo/MyNetcdfFile.nc"
names(2) = "LAI"
from_file(2) = "PathTo/MyNetcdfFile.nc"
names(3) = “beta1”
transfer_func(3) = "a + b * LAI + c * land_cover”
from_data_arrays(1:2,3) = "LAI", "land_cover"
target_coord_names(1:3,3) = "t_out", "y_out",
"x_out"
= "1.0", "1.0", "1.0"

) = "a", "b", "c"
3) = 0.5, -0.3, 1.2
```

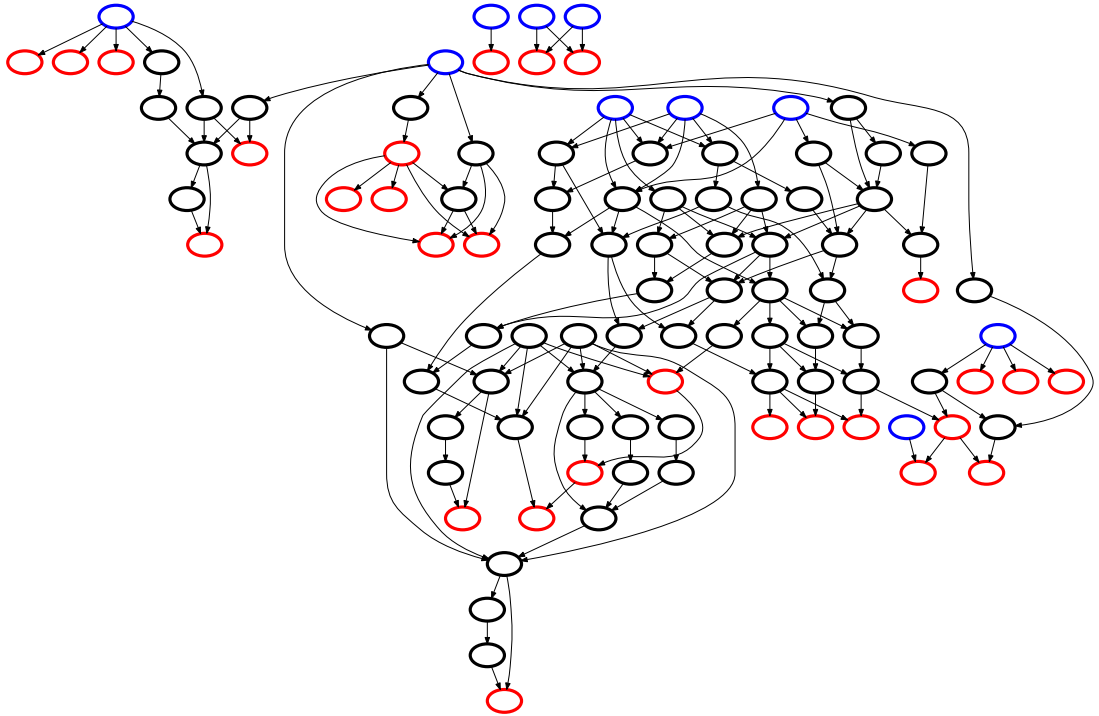
- simple string search & replace operations
- *real conversion necessary (templating?)*
- *grammar? (a+b == b+a)*
- *at runtime in Fortran?*

Open tasks



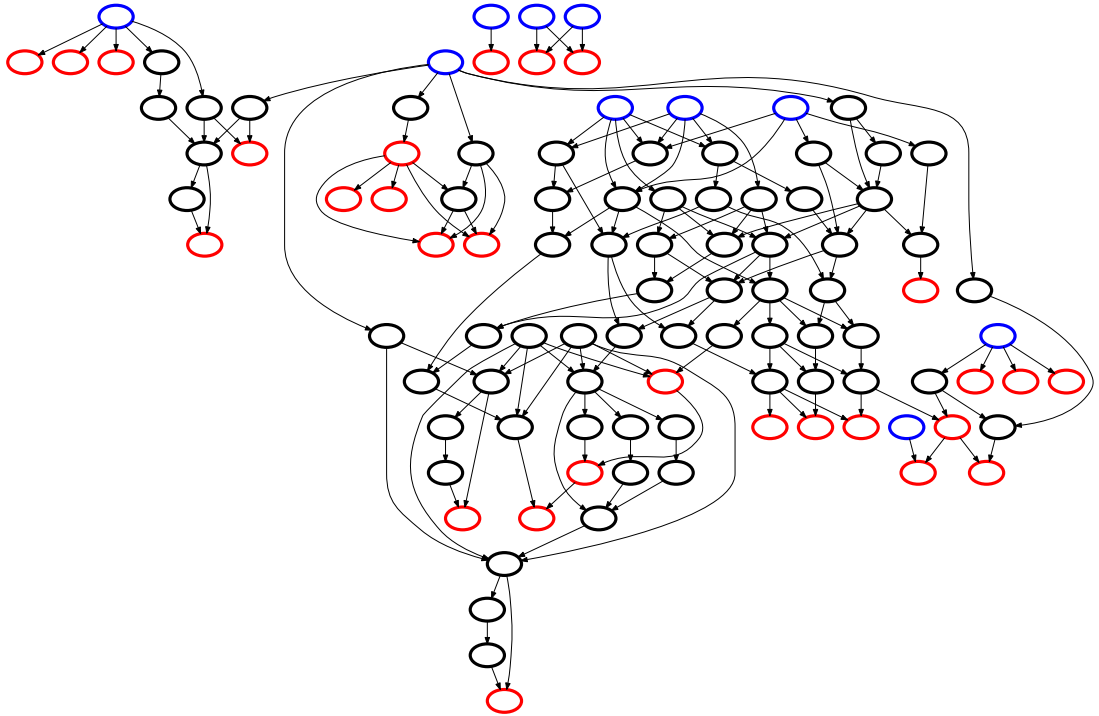
Open tasks

- sophisticated remapping library



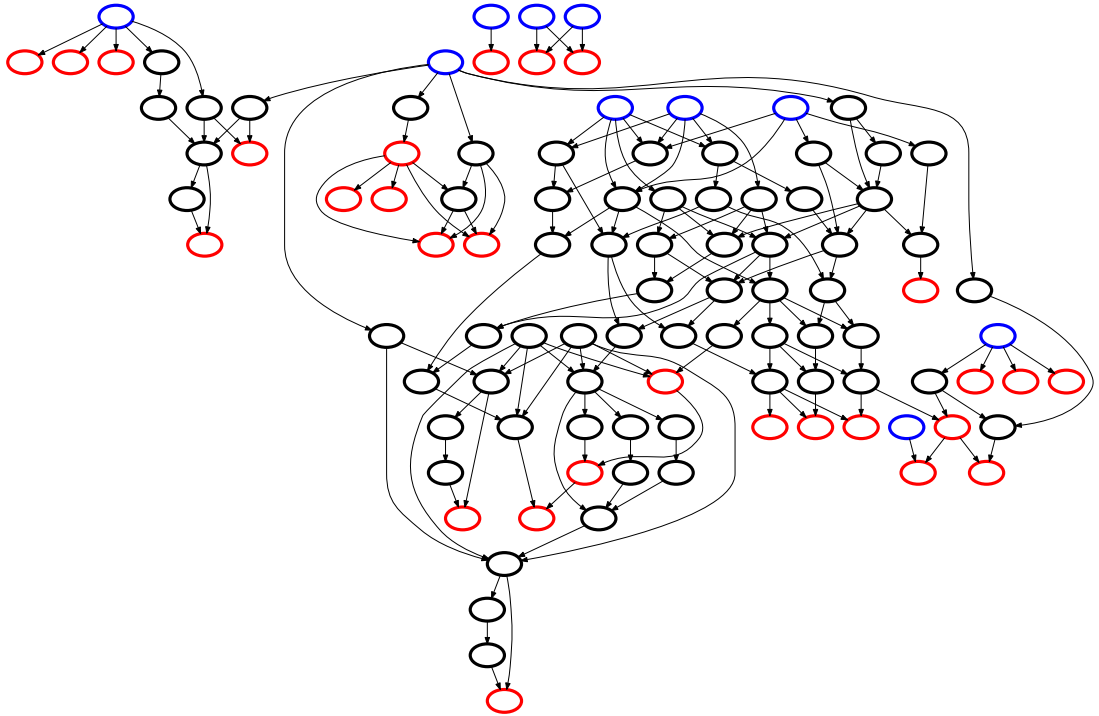
Open tasks

- sophisticated remapping library
- performance improvements



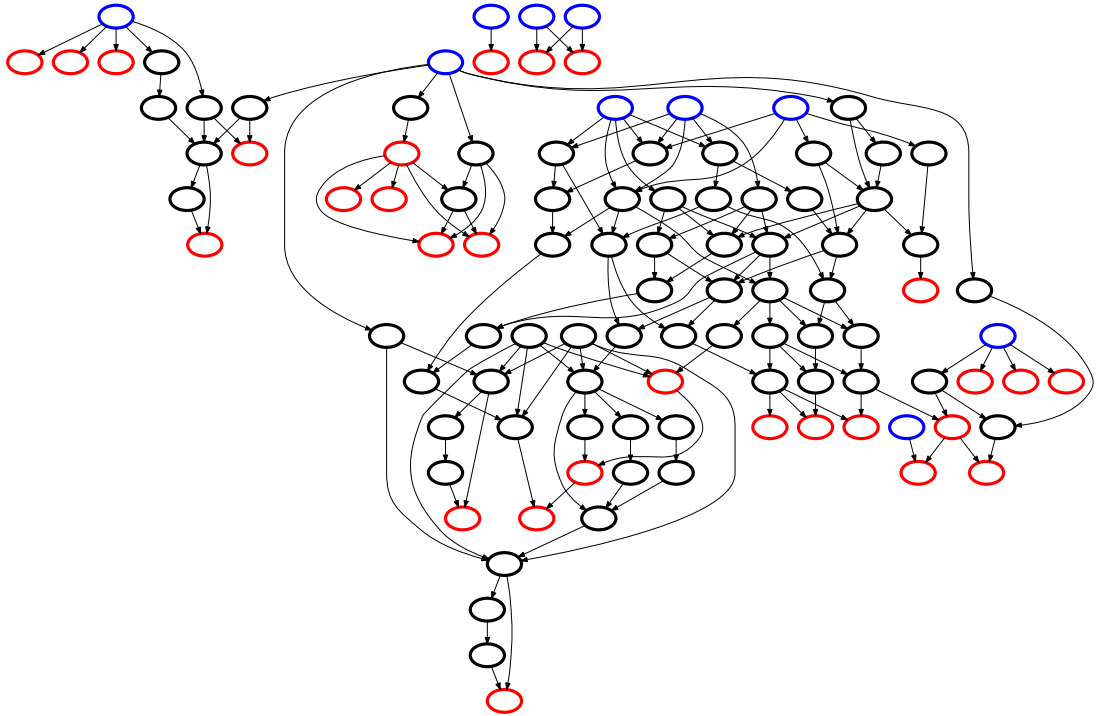
Open tasks

- sophisticated remapping library
- performance improvements
  - parallelization of graph



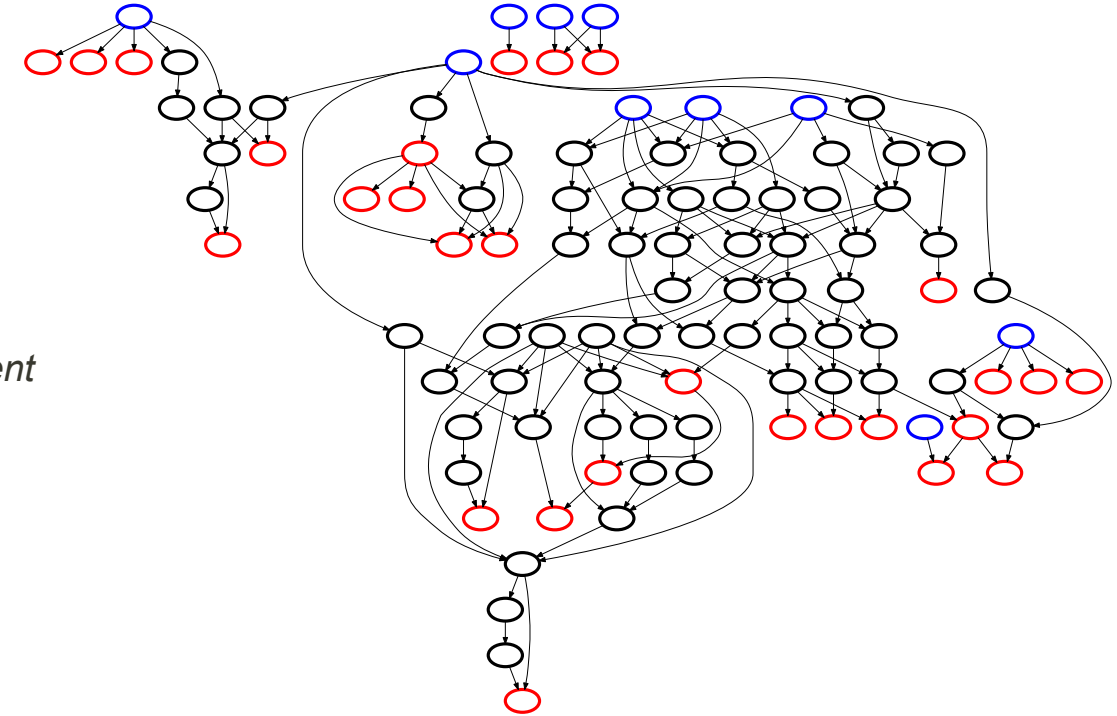
Open tasks

- sophisticated remapping library
- performance improvements
  - parallelization of graph
- unit tests



Open tasks

- sophisticated remapping library
- performance improvements
  - parallelization of graph
- unit tests
- *publish in Geoscientific Model Development*

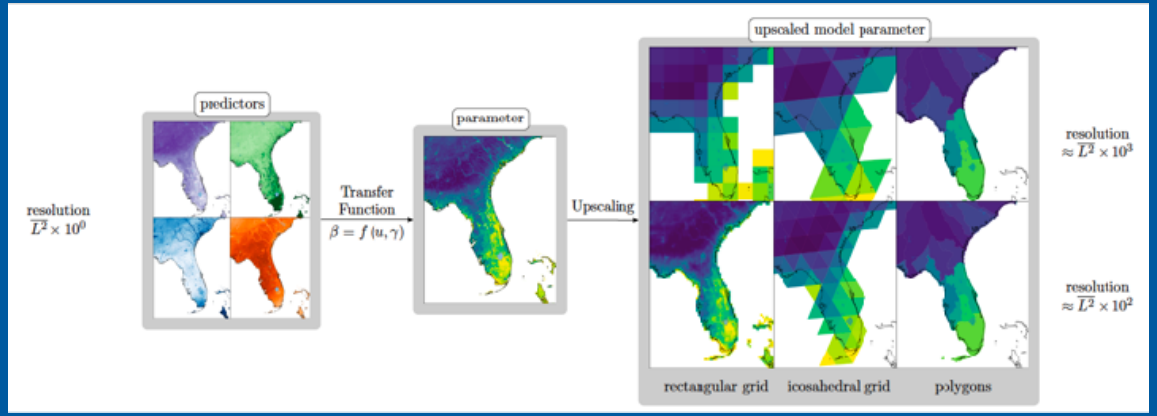


MPR
Summary

robert.schweppe@ufz.de

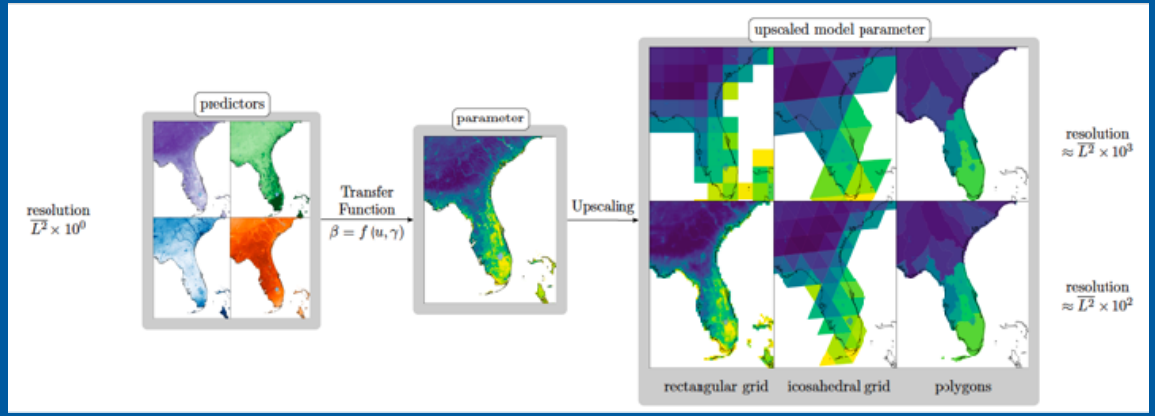
MPR
Summary

- MPR uses transfer functions and upscaling operators to estimate model parameters from high-resolution data



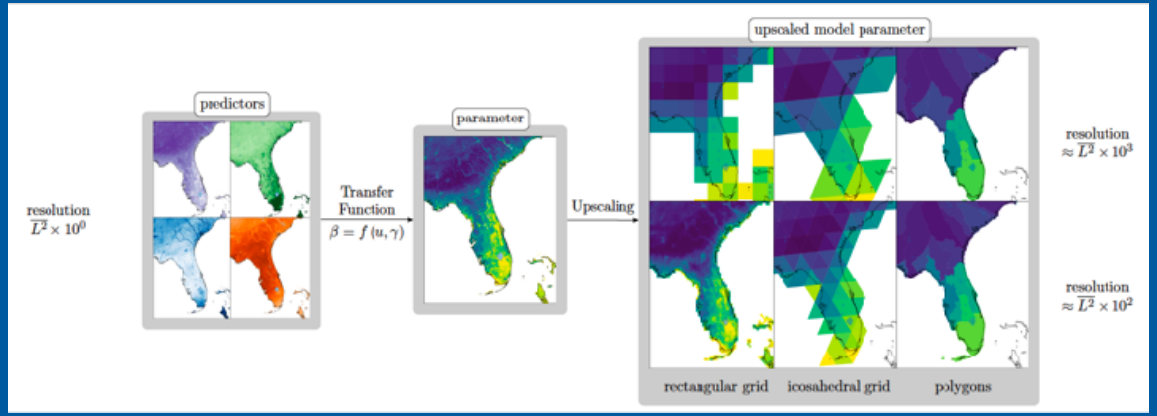
MPR
Summary

- MPR uses transfer functions and upscaling operators to estimate model parameters from high-resolution data
- Simple, flexible, modular setup, can be coupled to any model



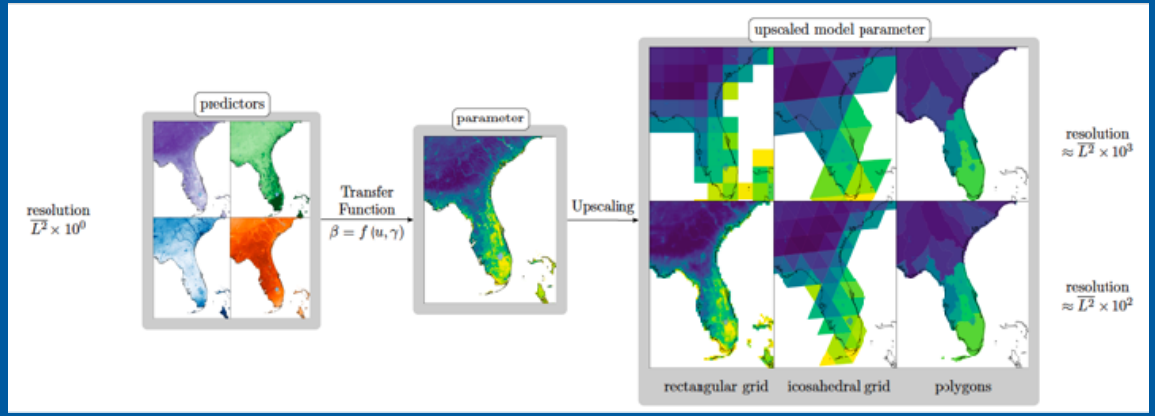
MPR
Summary

- MPR uses transfer functions and upscaling operators to estimate model parameters from high-resolution data
- Simple, flexible, modular setup, can be coupled to any model
- Code development on git.ufz.de/CHS/MPR, available soon, currently on demand only



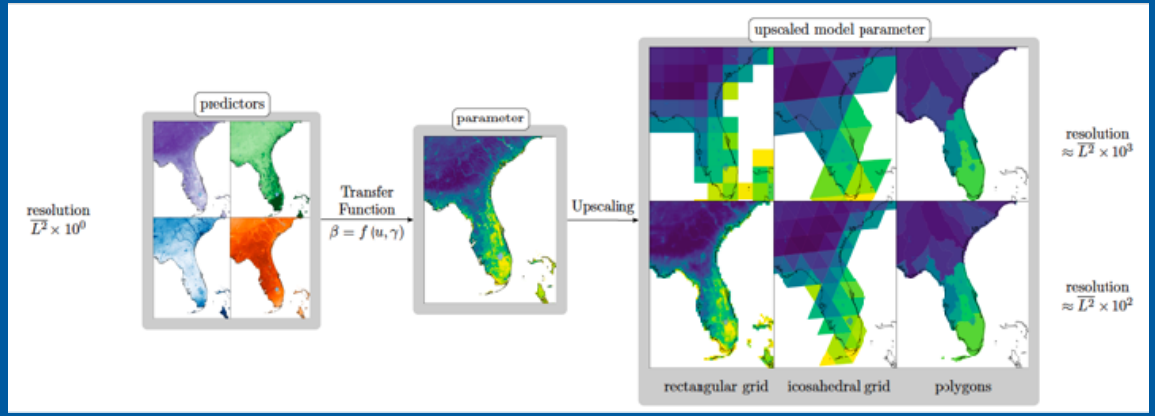
MPR
Summary

- MPR uses transfer functions and upscaling operators to estimate model parameters from high-resolution data
- Simple, flexible, modular setup, can be coupled to any model
- Code development on git.ufz.de/CHS/MPR, available soon, currently on demand only
- outlook: coupling with land-surface model used by European weather agency (ECMWF) in their simulations



MPR
Summary

- MPR uses transfer functions and upscaling operators to estimate model parameters from high-resolution data
- Simple, flexible, modular setup, can be coupled to any model
- Code development on git.ufz.de/CHS/MPR, available soon, currently on demand only
- outlook: coupling with land-surface model used by European weather agency (ECMWF) in their simulations



Thank you for your attention!