

Modeling earth-surface dynamics with Landlab

The Landlab development team:

Jordan Adams (Tulane U.)

Nicole Gasparini (Tulane U.)

Dan Hobley (Univ. of Colorado, Now at Cardiff)

Eric Hutton (CSDMS)

Erkan Istanbulluoglu (Univ. of Washington)

Jennifer Knuth (Univ. of Colorado)

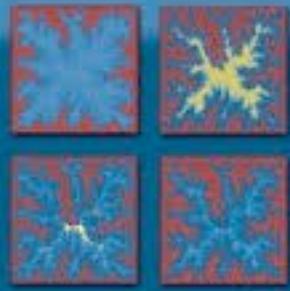
Sai Siddharta Nudurupati (Univ. of Washington)

Greg Tucker (Univ. of Colorado)



Jon Pelletier

Quantitative Modeling of Earth Surface Processes



Cambridge

Numerical Methods in the Hydrological Sciences

George Hornberger
University of Virginia
Department of Environmental Sciences

Patricia Wiberg
University of Virginia
Department of Environmental Sciences

Published by the American Geophysical Union

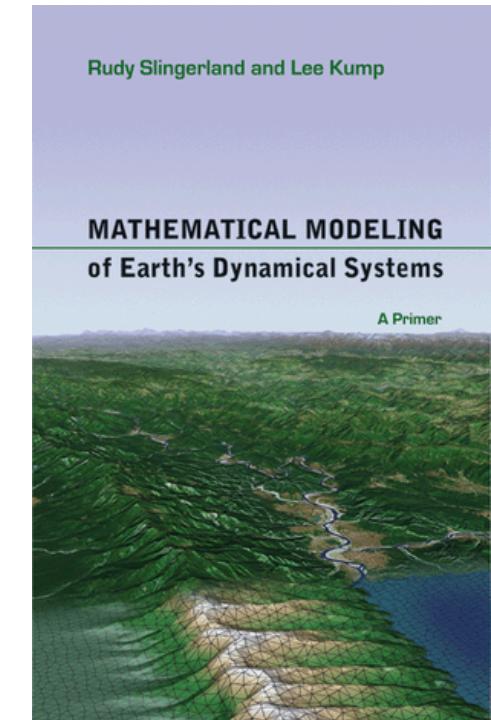
SOFTWARE

NUMERICAL ALGORITHM

Rudy Slingerland and Lee Kump

MATHEMATICAL MODELING of Earth's Dynamical Systems

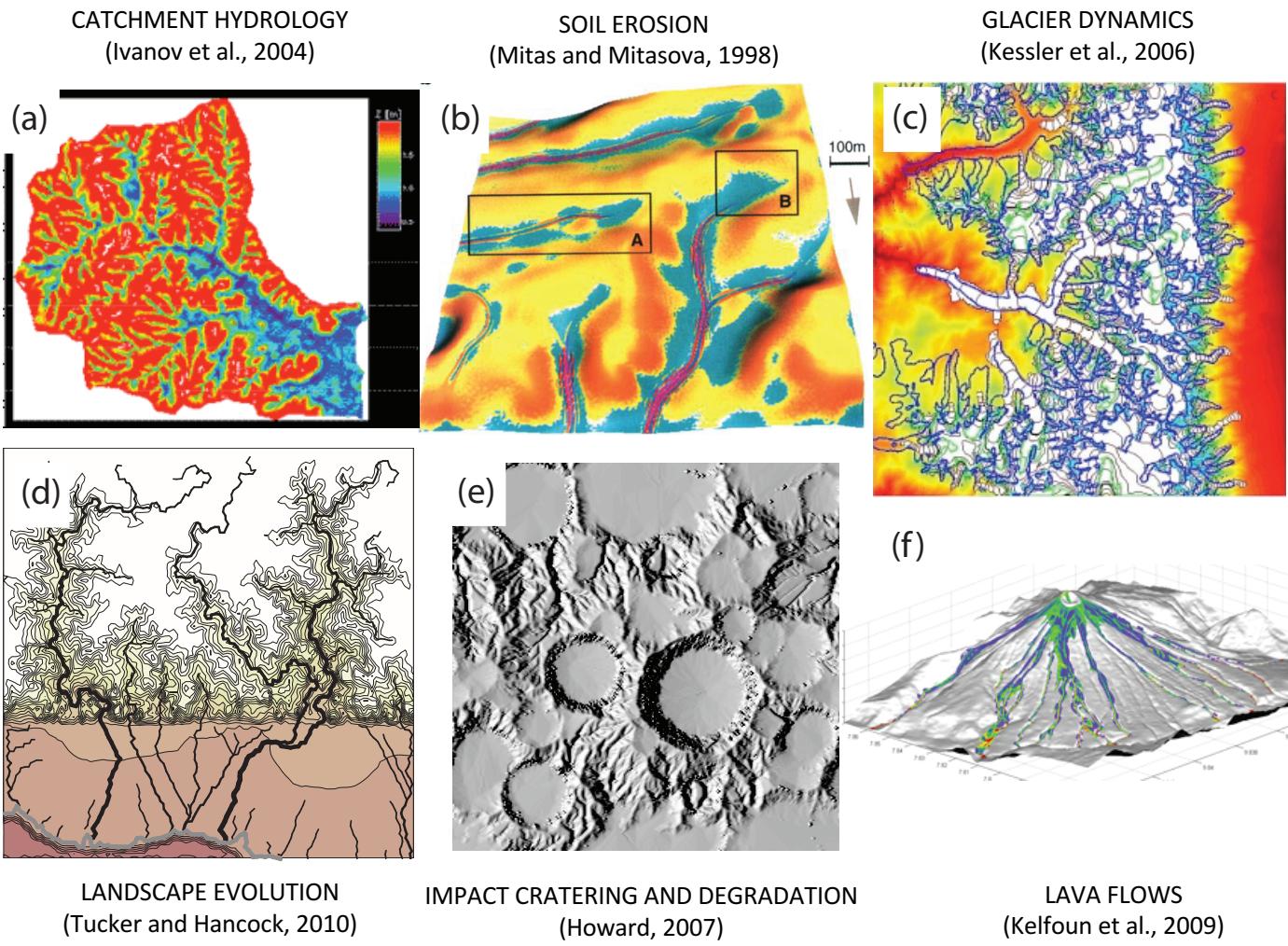
A Primer

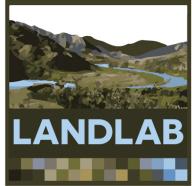


DYNAMICAL MODEL

NATURE

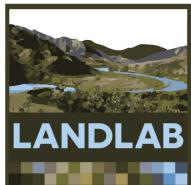
2D models of earth-surface processes



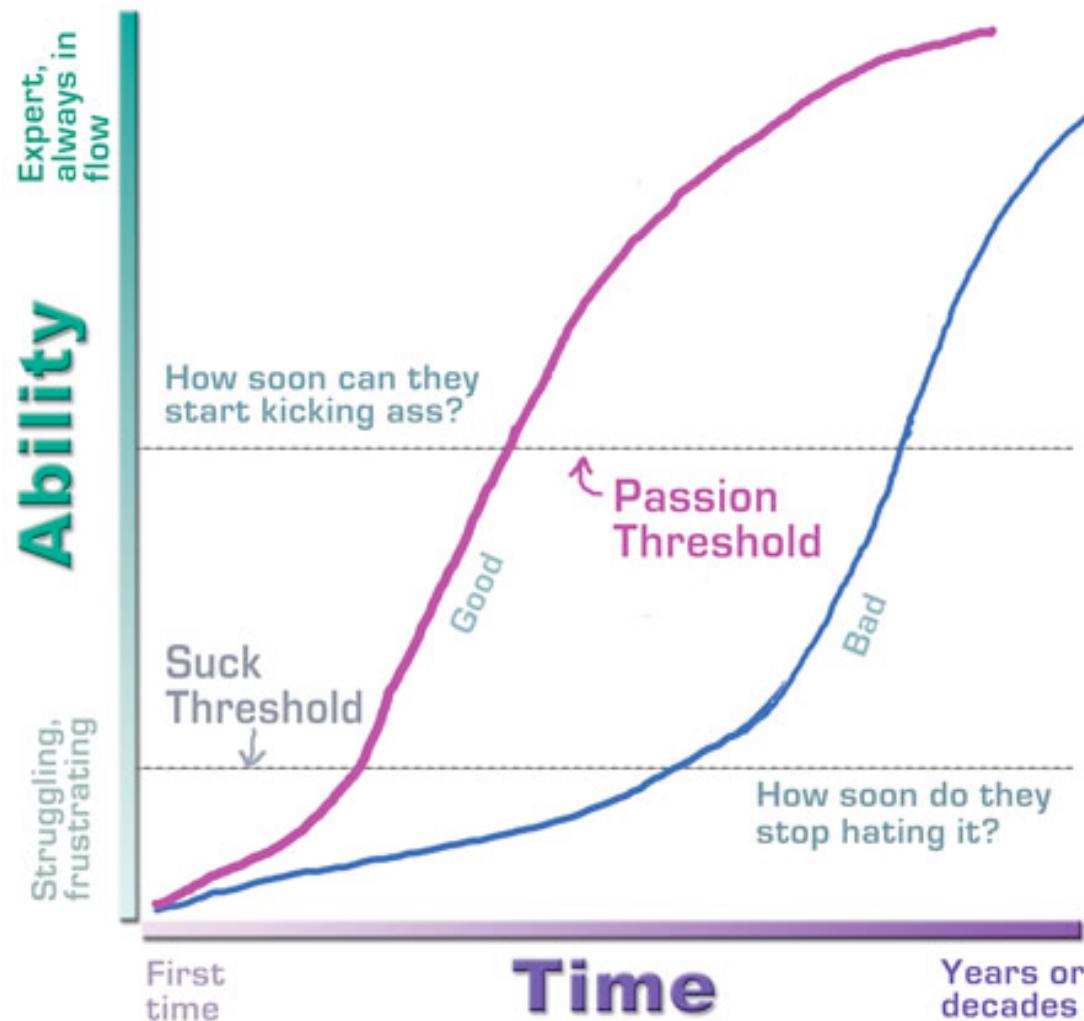


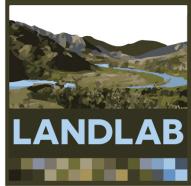
What is Landlab?

- A Python-language programming library
- Supports efficient creation and/or coupling of 2D numerical models
- Geared toward (but not limited to) earth-surface dynamics



The Kick Ass Curve



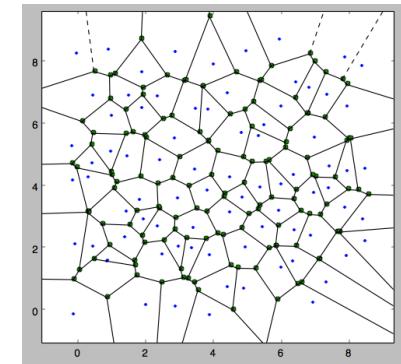


What Landlab provides

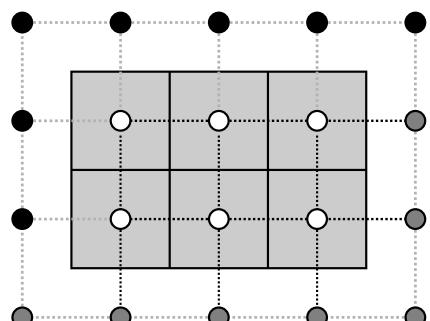
1. Grid creation and management

- Create a structured or unstructured grid in one or a few lines of code
- Attach data to grid elements
 - Facilitates staggered-grid schemes
 - Passing the grid = passing the data

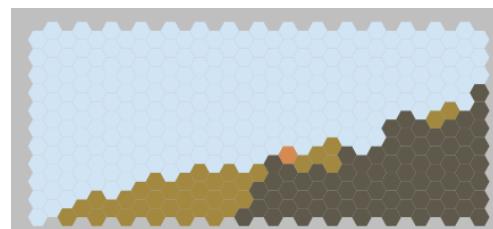
VORONOI / DELAUNAY



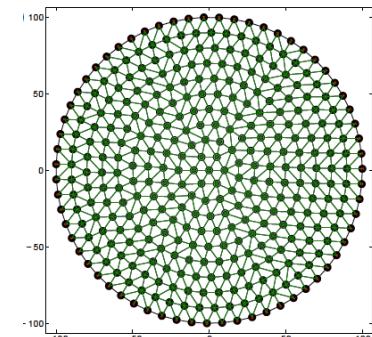
RASTER

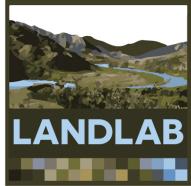


HEXAGONAL



RADIAL

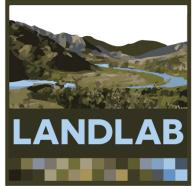




What Landlab provides

2. Coupling of components

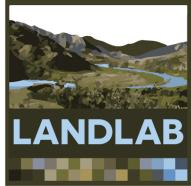
- A *component* models a single process (e.g., lithosphere flexure, incident solar radiation, flow routing across terrain)
- Components have a standard interface and can be combined by writing a short Python script
- Save development time by re-using components written by others
- Components also include analytical tools for analyzing landscapes (e.g. channel steepness, hillslope length, ...)



What Landlab provides

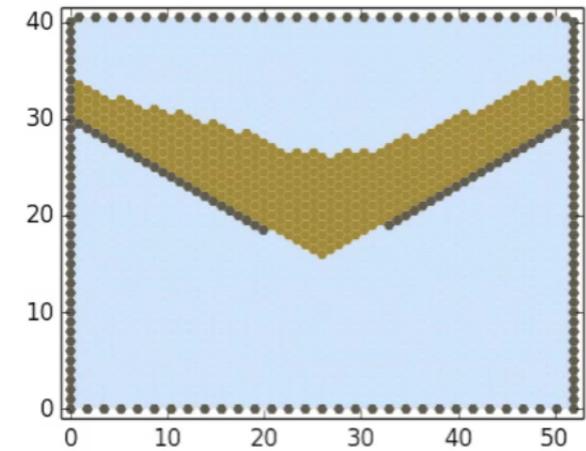
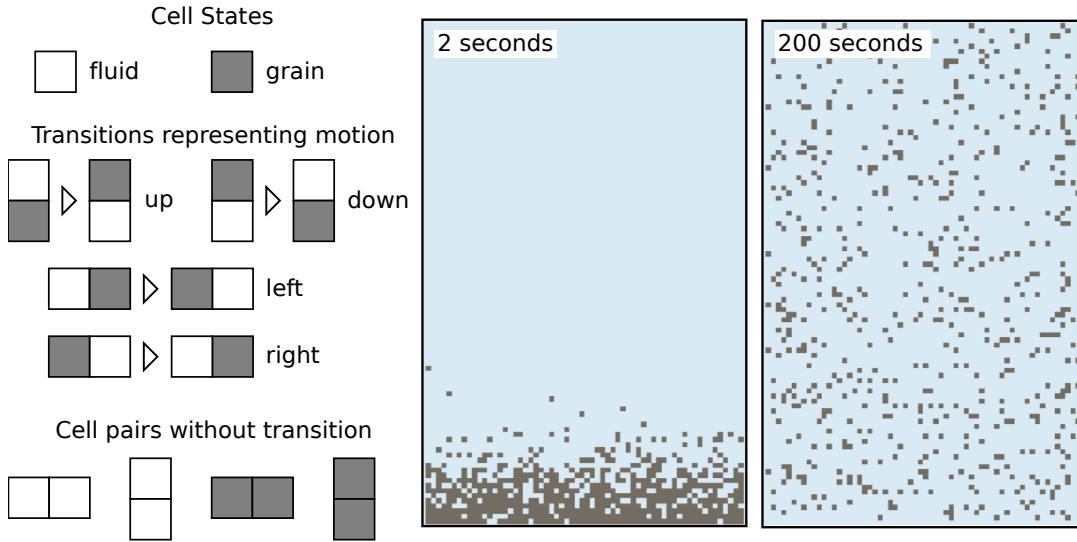
3. Input and output

- Read model parameters from a formatted text file
- Read in digital terrain data (e.g., DEMs) → grid
- Write gridded output to files (netCDF format)
- Plot data using Matplotlib graphics library

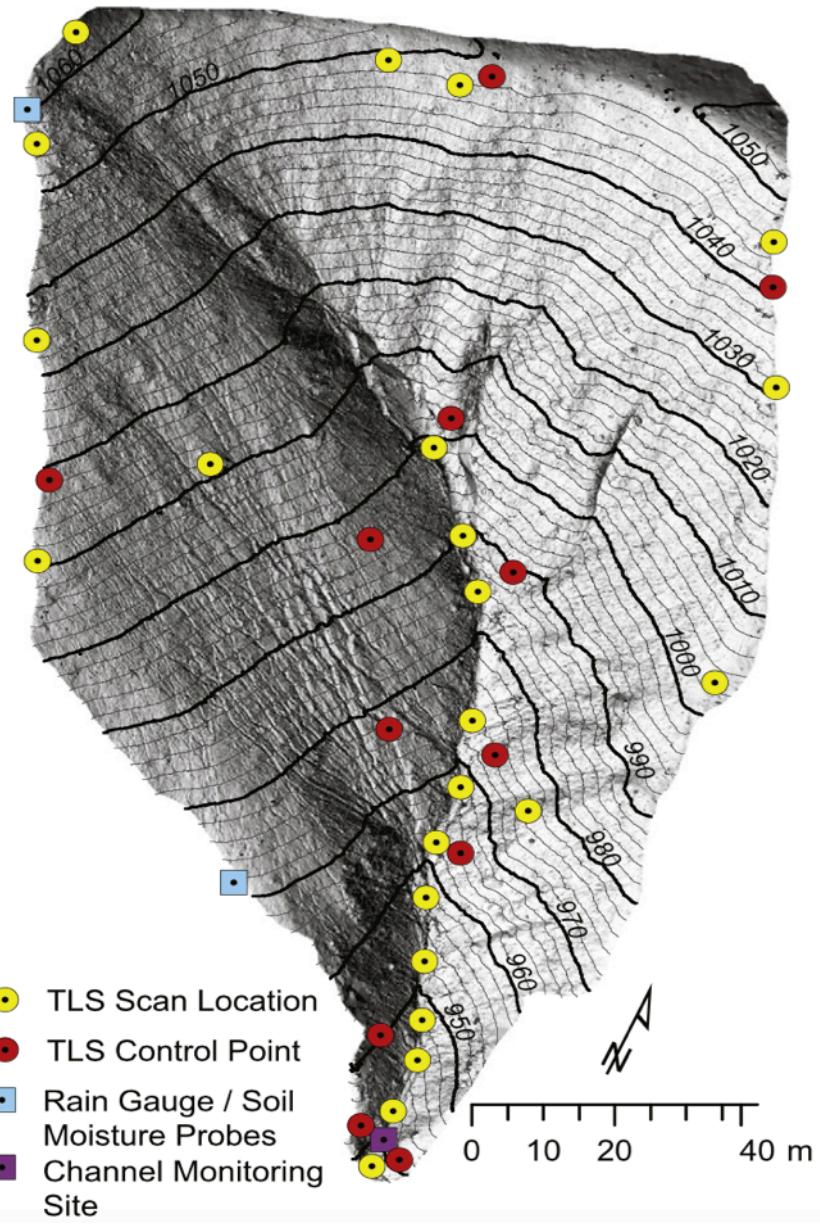


What Landlab provides

- ## 4. Support for cellular-automaton modeling
- CellLab-CTS: Continuous-time stochastic CA model “engine”

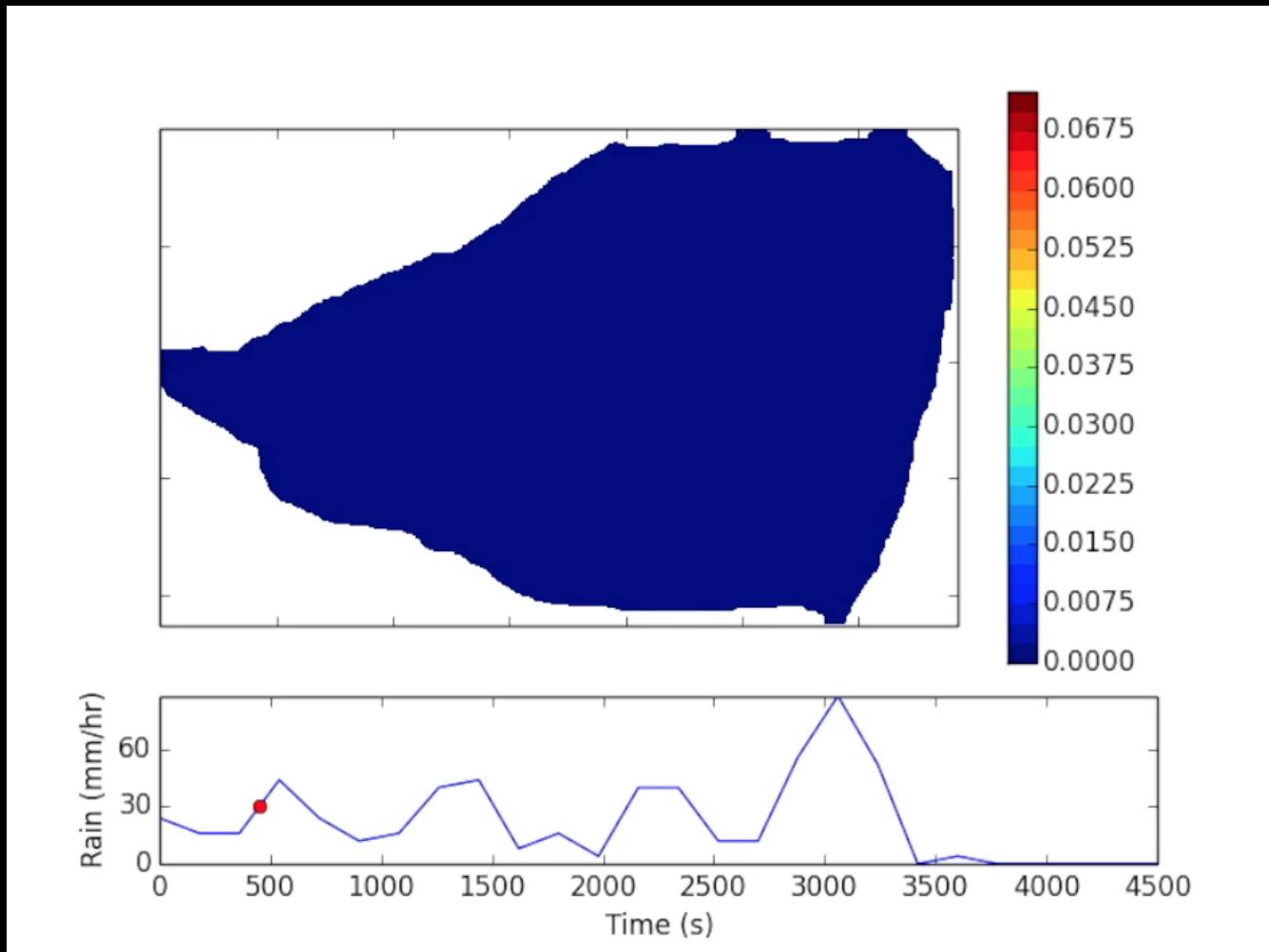


Examples of Landlab-built models



(Source: Francis Rengers, USGS)

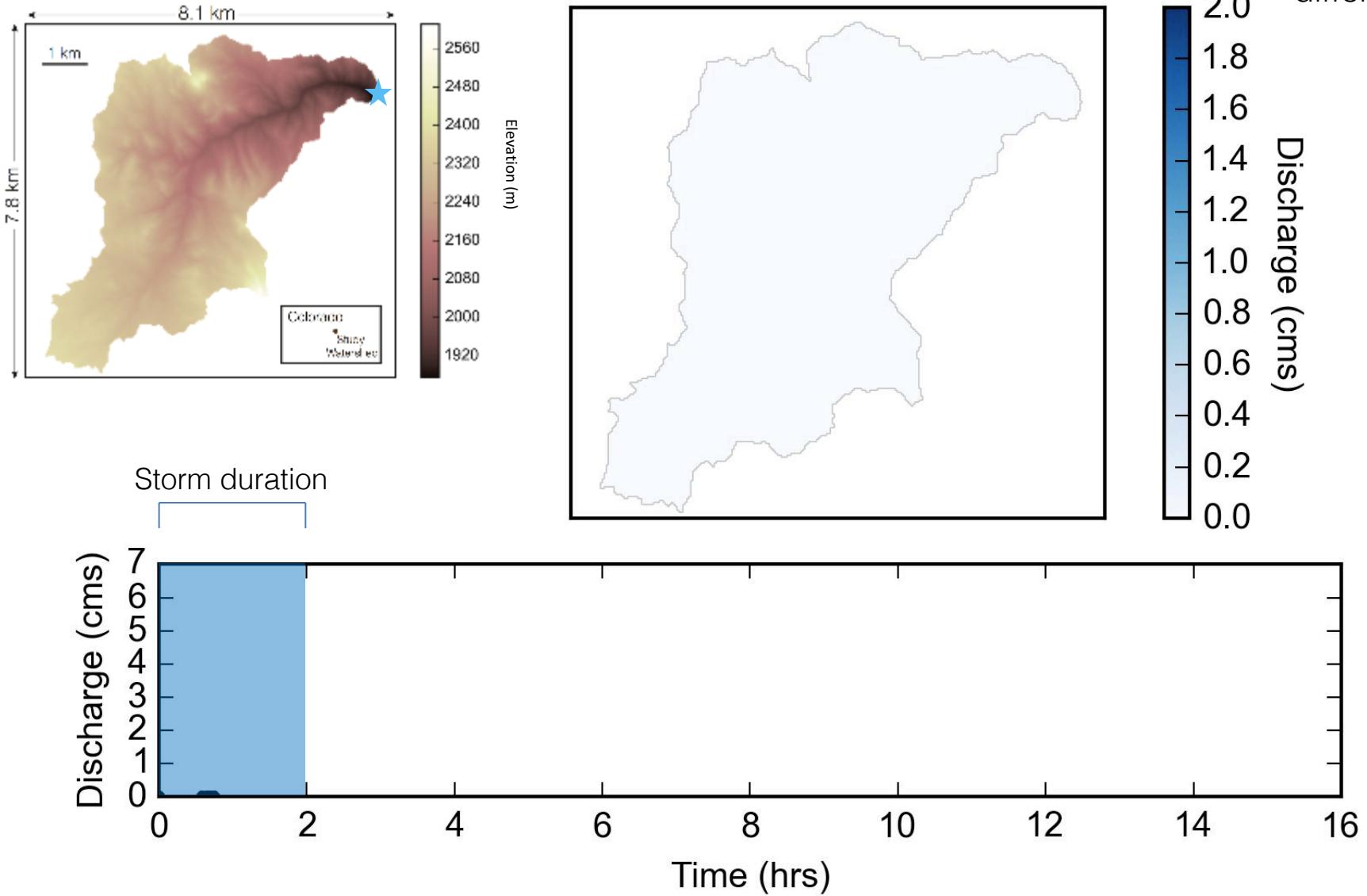
Storm runoff patterns in the Transverse Ranges



(Source: Francis Rengers, USGS)

Application in a real world setting: Spring Creek, CO.

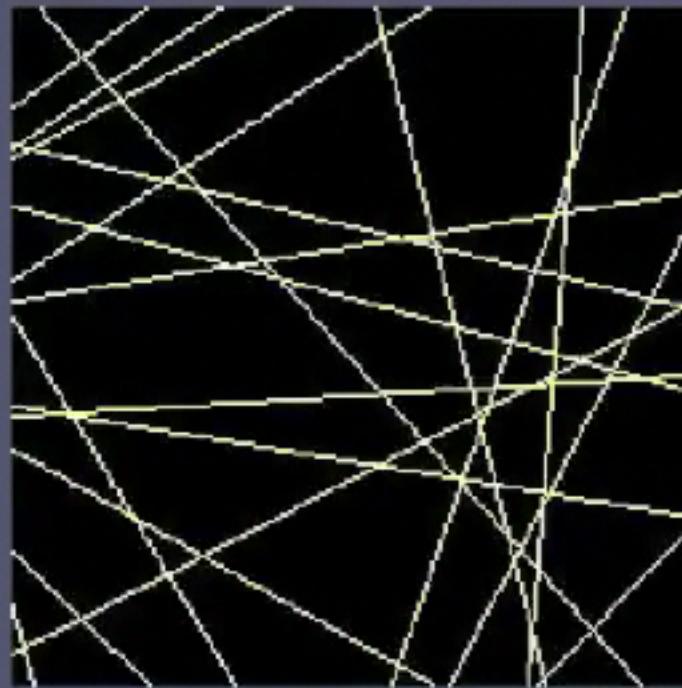
*Note scale differences



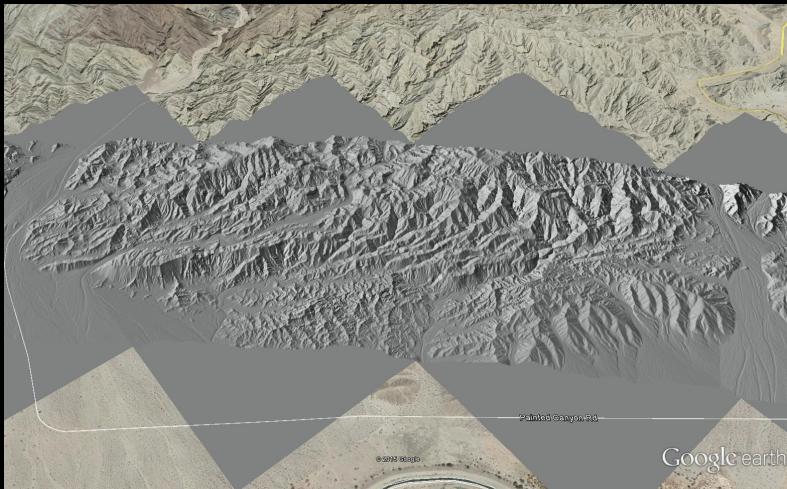
(source: Jordan Adams, Tulane University)

Cellular automaton model of weathering along fractures

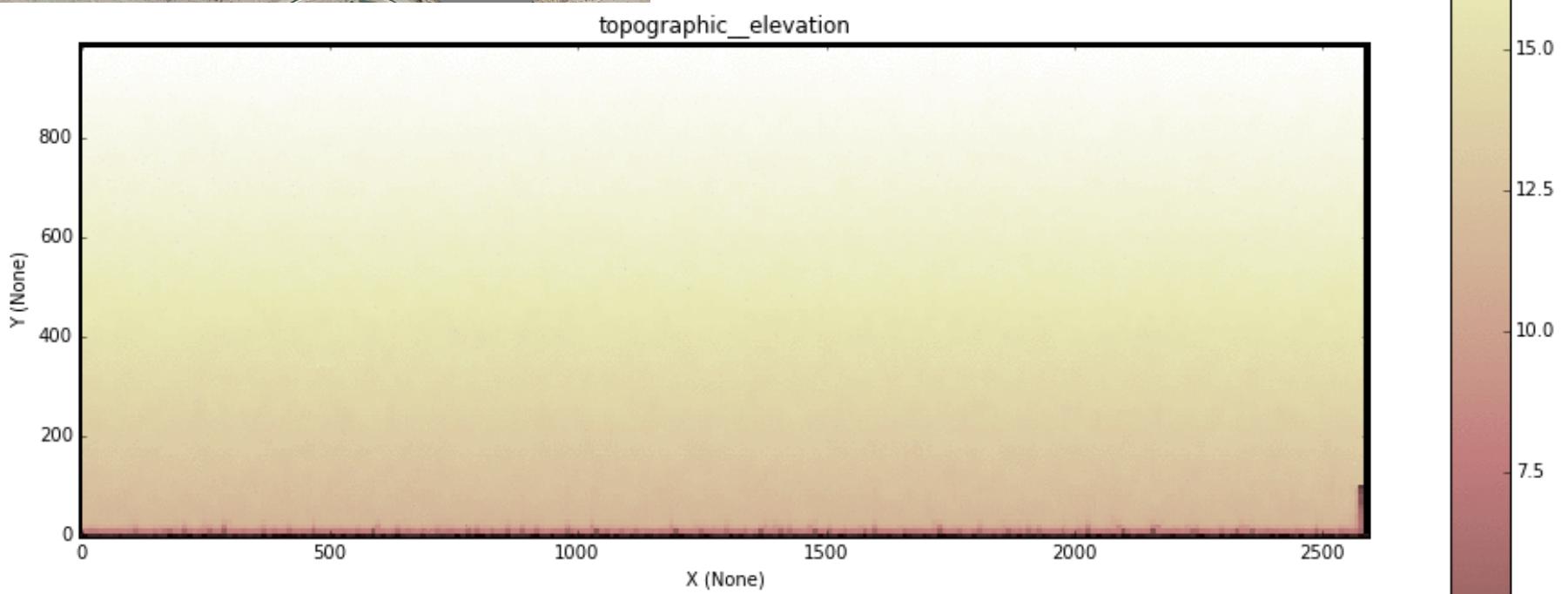
(Source: Greg Tucker, CU-Boulder)



Why do strike-slip faults sometimes show distributed shear, and sometimes not?

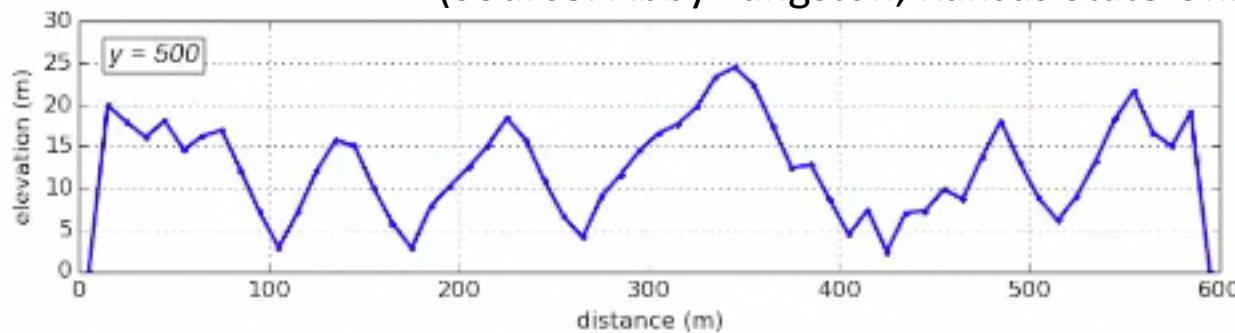
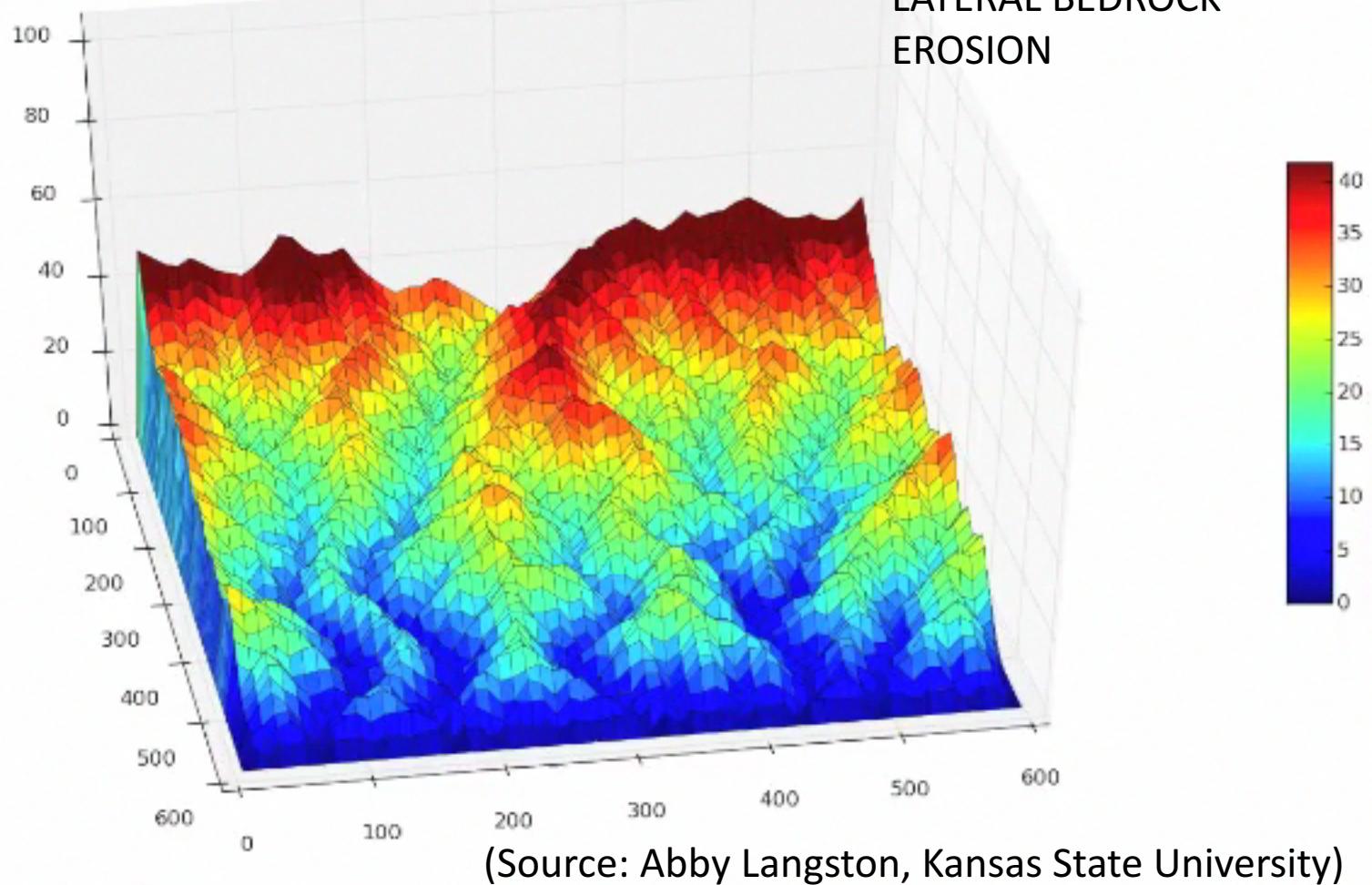


(Source: Harrison Gray, CU-Boulder)

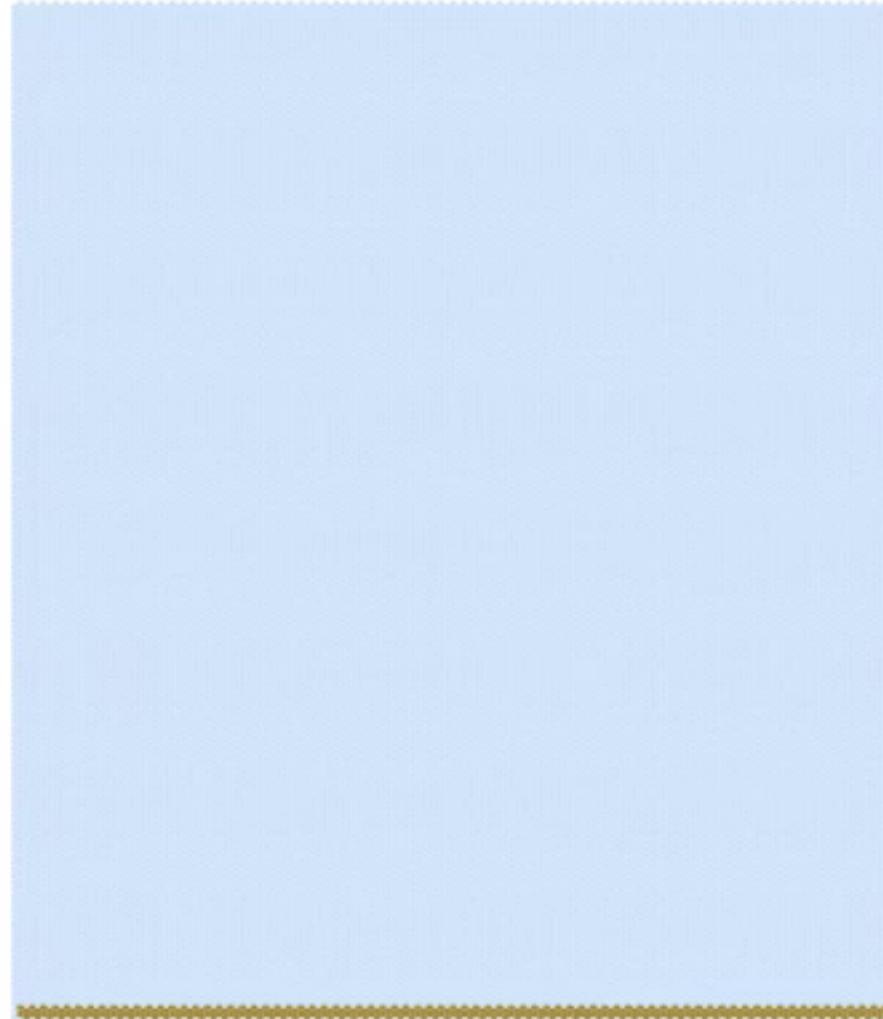


time = 0ky

VALLEY WIDENING BY LATERAL BEDROCK EROSION



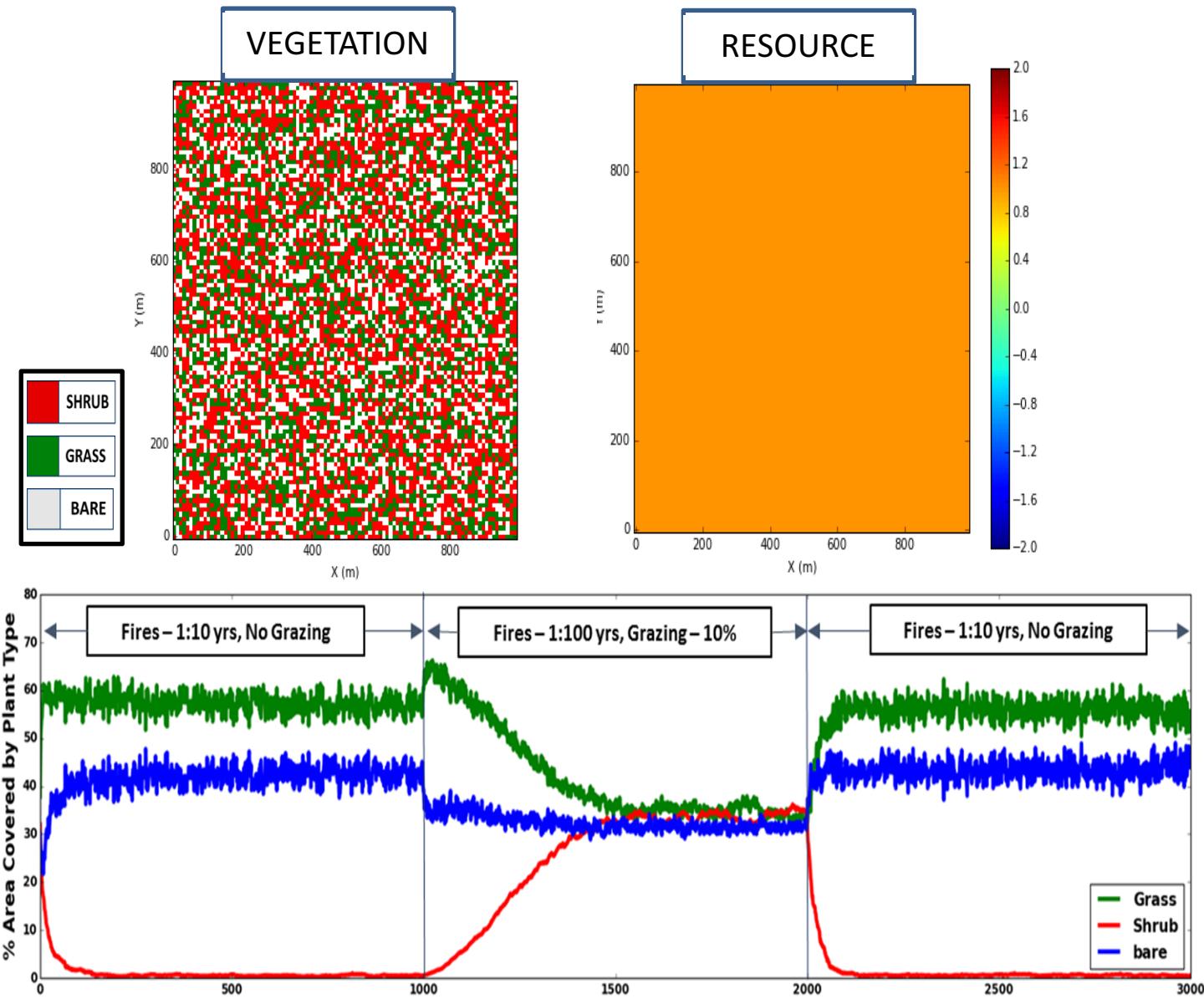
Weathering & disturbance similar to slip rate



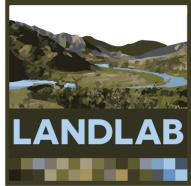
$$W' = D' = 1$$

(Source: Greg Tucker, CU-Boulder)

Climate Change Experiments #1



(Source: Sai Nudurupati, U. Washington)



Using Landlab grids

- Aim: make it easier to set up a 2D numerical model grid
- Grid data and functions contained in a single Python object

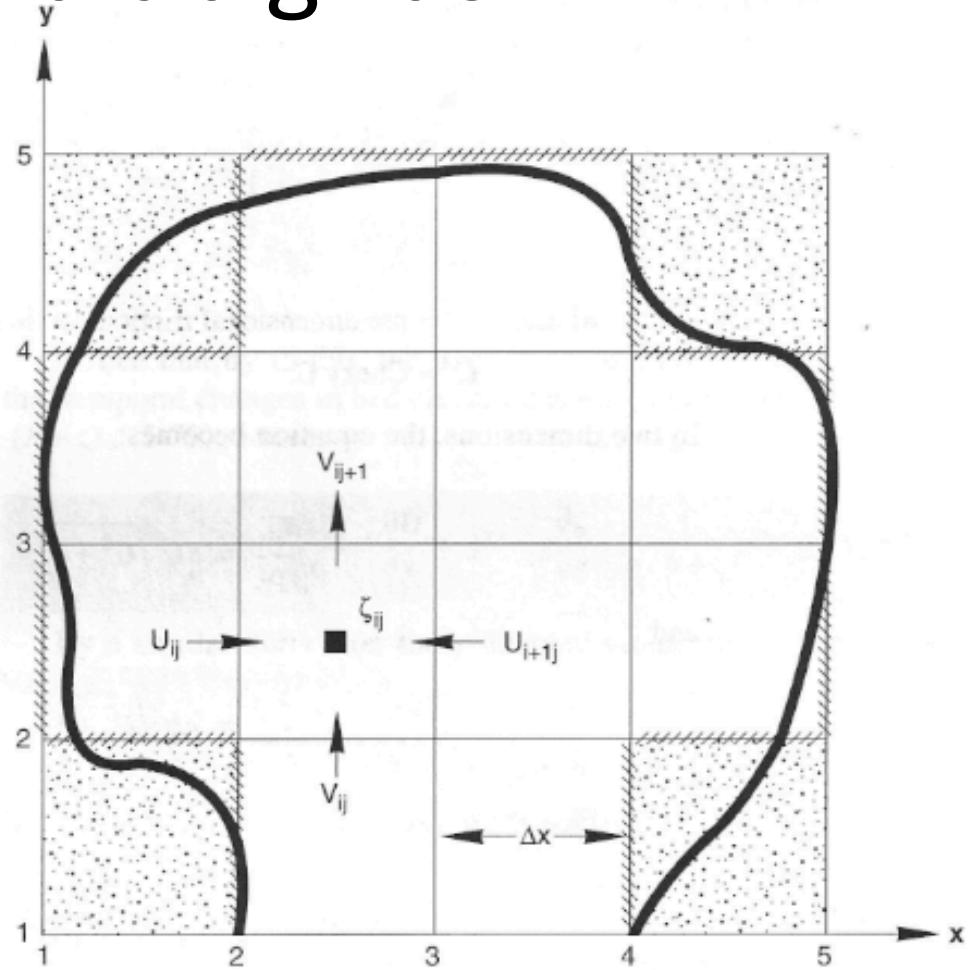
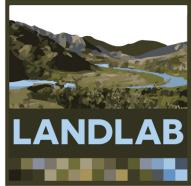


Figure 5-19

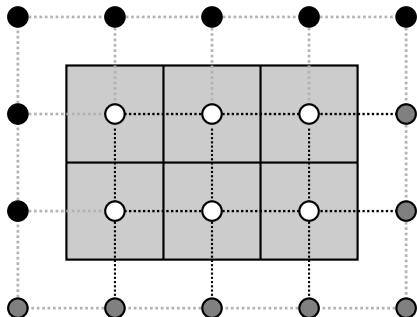
Discretization grid for 2-D circulation model.

Slingerland, Harbaugh, and Furlong (1994)

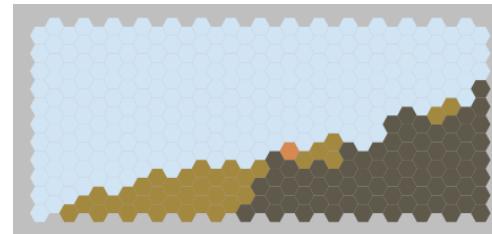


Currently four grid types are available:

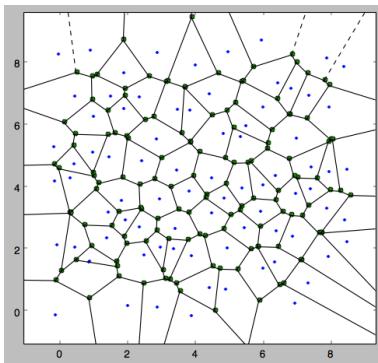
- `RasterModelGrid`



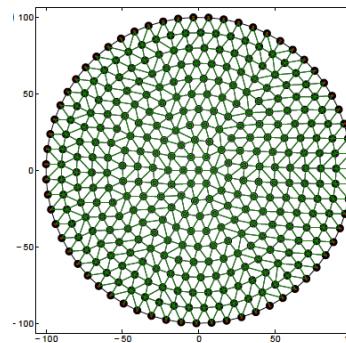
- `HexModelGrid`

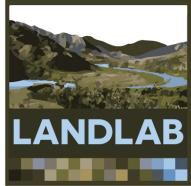


- `VoronoiModelGrid`



- `RadialModelGrid`

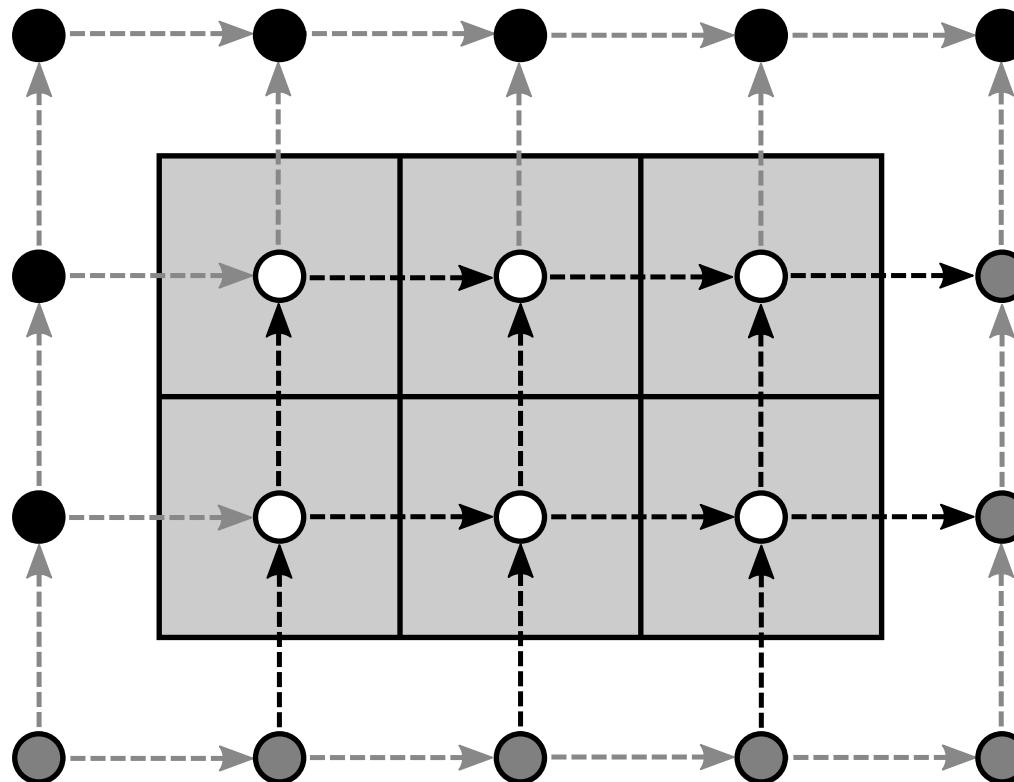


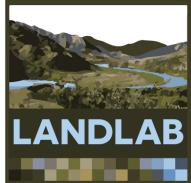


Example: creating a grid

```
>>> from landlab import RasterModelGrid  
>>> rg = RasterModelGrid((4, 5), 10.0)  
>>> rg.number_of_nodes
```

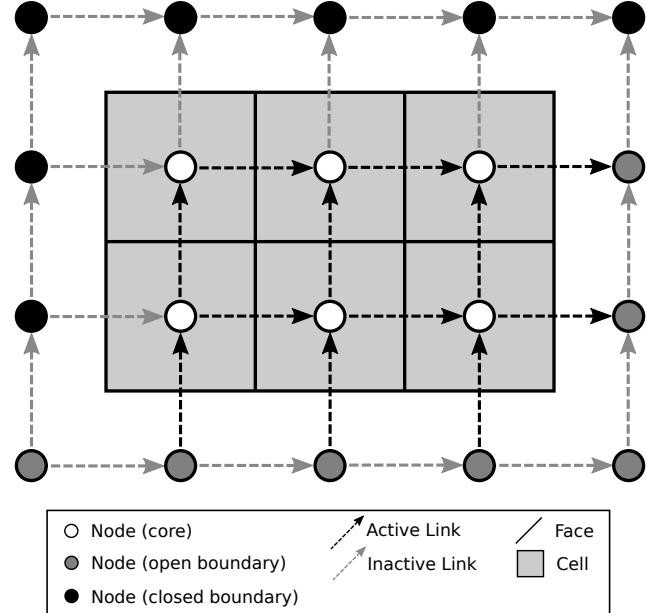
20

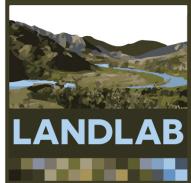




Grid elements: nodes

```
>>> rg.number_of_node_rows  
4  
>>> rg.number_of_node_columns  
5  
>>> rg.x_of_node  
array([ 0., 10., 20., 30., 40., 0., 10., 20., 30., 40., 0.,  
       10., 20., 30., 40., 0., 10., 20., 30., 40.])  
>>> rg.y_of_node  
array([ 0., 0., 0., 0., 0., 10., 10., 10., 10., 10., 20.,  
       20., 20., 20., 20., 30., 30., 30., 30., 30.])
```

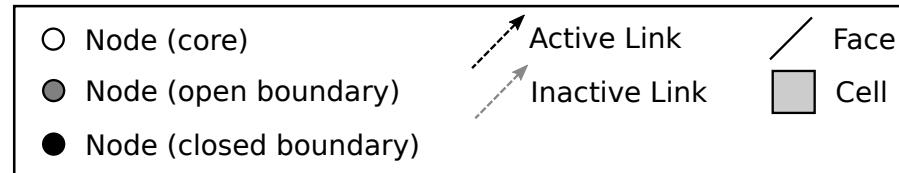
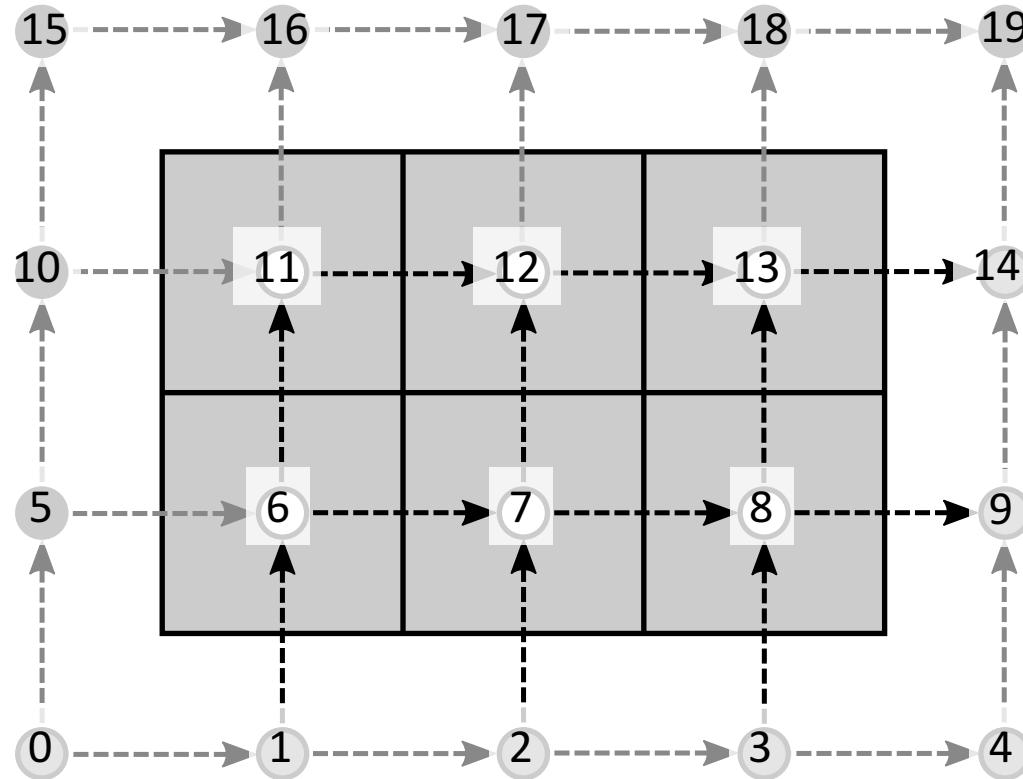


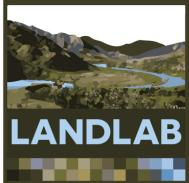


Node numbering

Nodes are
always
sorted by
y coordinate

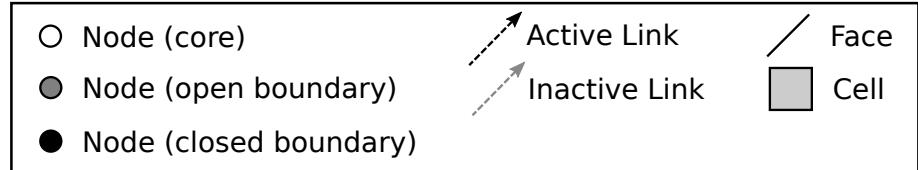
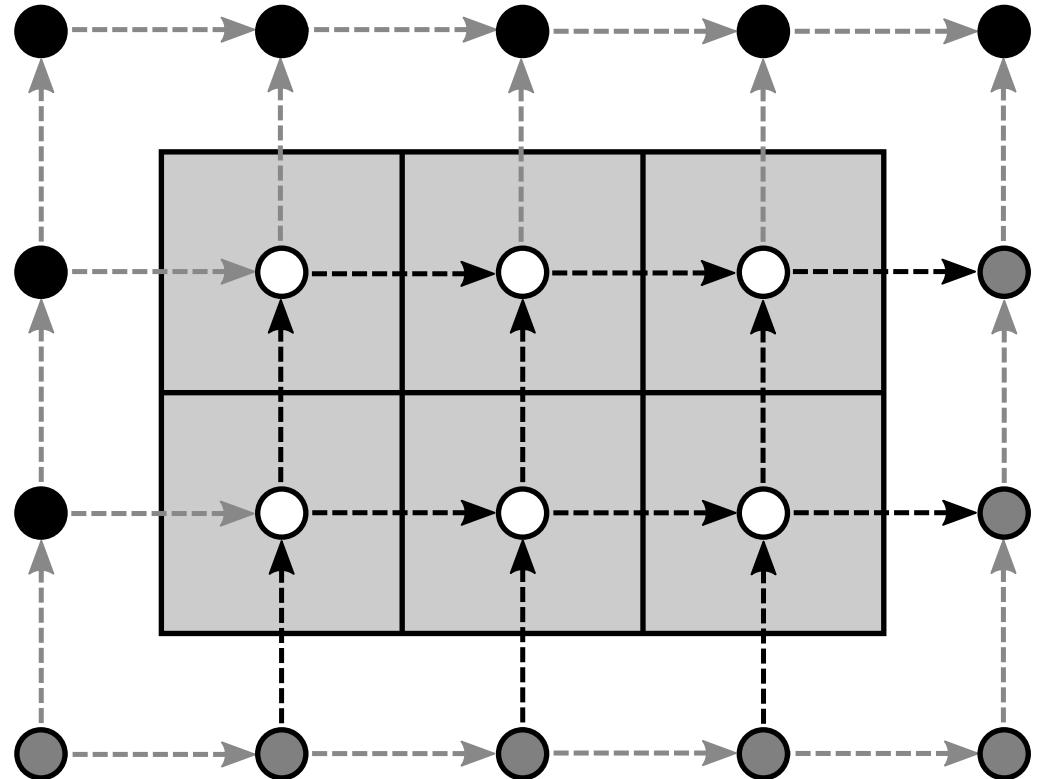
Nodes with
equal y are
sorted by x

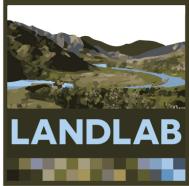




Core and boundary nodes

- Core nodes
- Boundary nodes
 - Open
 - **Fixed value**
 - Fixed gradient
 - Looped
 - Closed

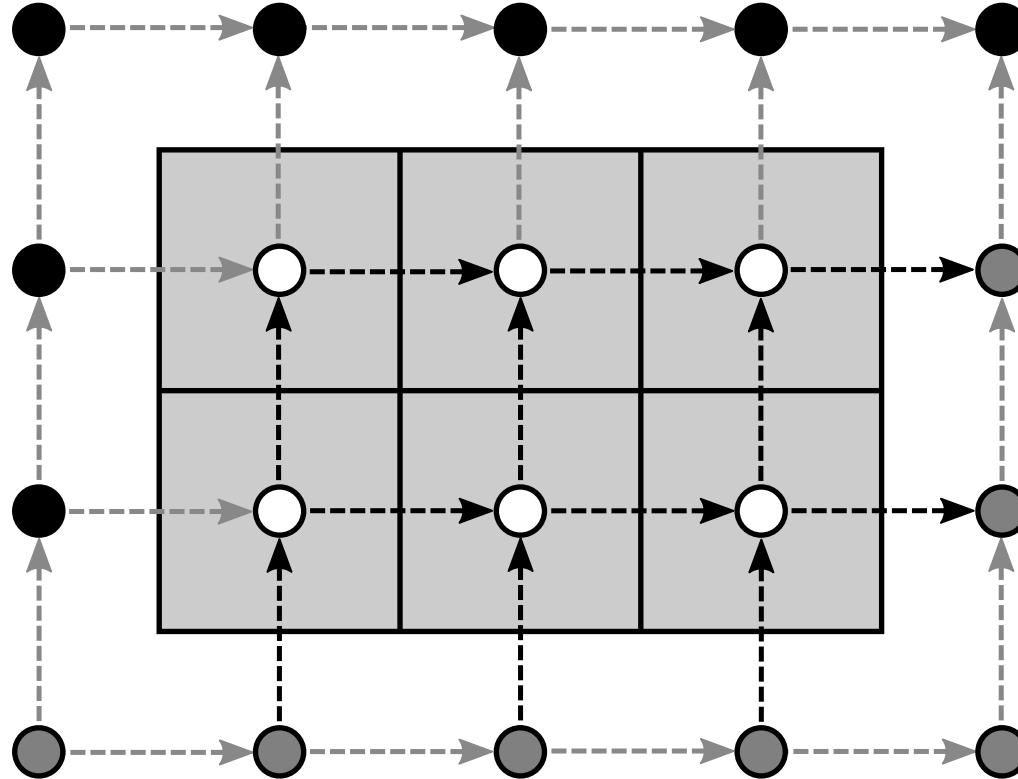




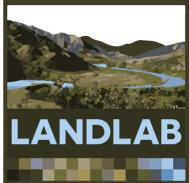
Grid elements: links

Link =
directed line
segment
connecting
two adjacent
nodes

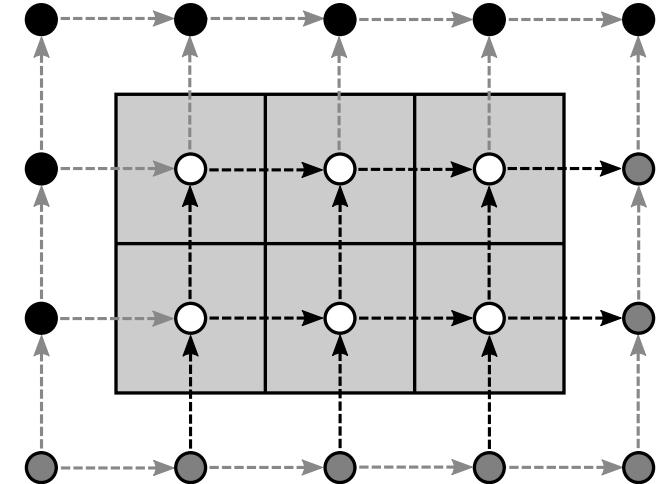
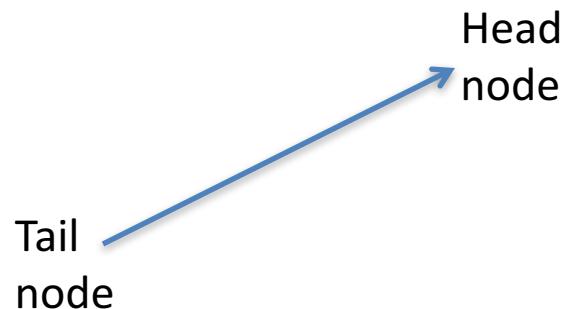
**Link
direction is
toward
upper right
half-space by
default**



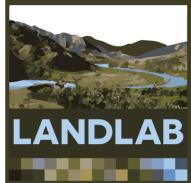
- Node (core)
- Node (open boundary)
- Node (closed boundary)
- Active Link
- Inactive Link
- / Face
- Cell



Grid elements: links



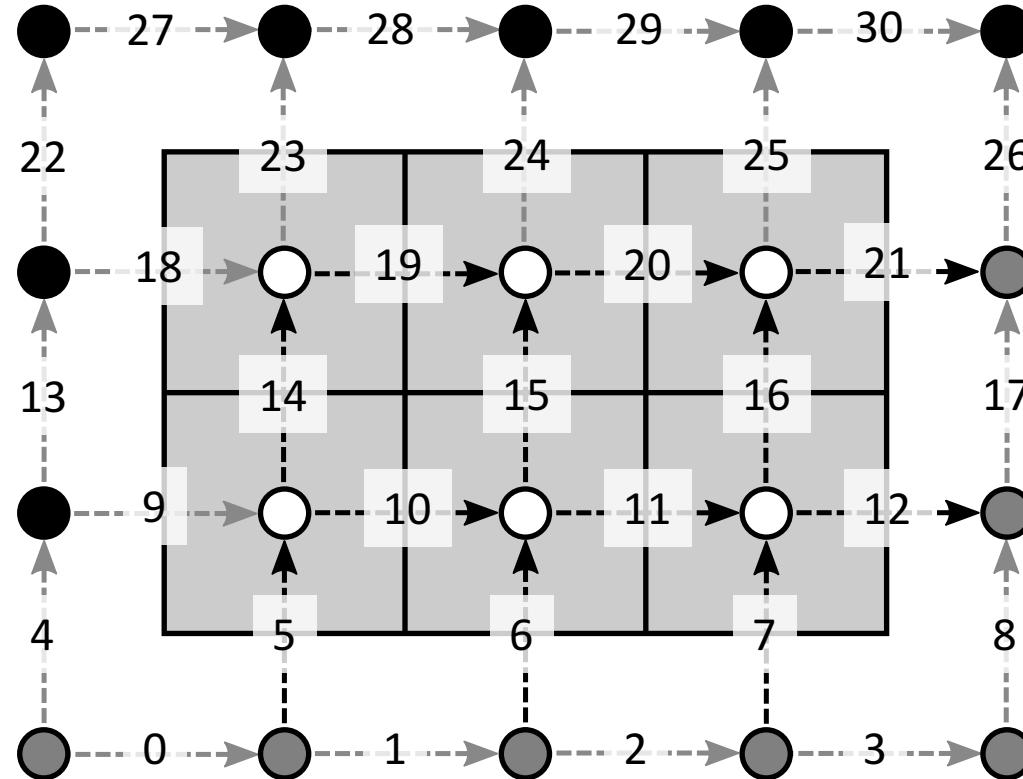
```
>>> rg.number_of_links
31
>>> rg.node_at_link_head
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9,  6,  7,  8,  9, 10, 11, 12, 13,
       14, 11, 12, 13, 14, 15, 16, 17, 18, 19, 16, 17, 18, 19])
>>> rg.node_at_link_tail
array([ 0,  1,  2,  3,  0,  1,  2,  3,  4,  5,  6,  7,  8,  5,  6,  7,  8,
       9, 10, 11, 12, 13, 10, 11, 12, 13, 14, 15, 16, 17, 18])
```



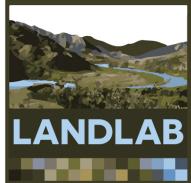
Link numbering

Links are sorted by mid-point y coordinate

Links with equal y are sorted by x



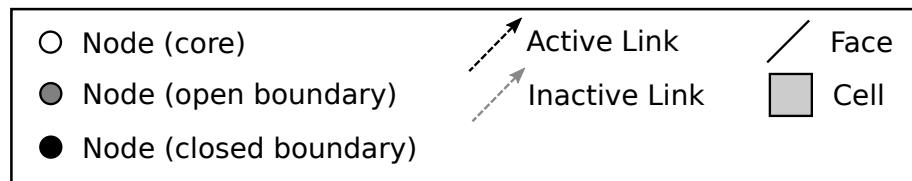
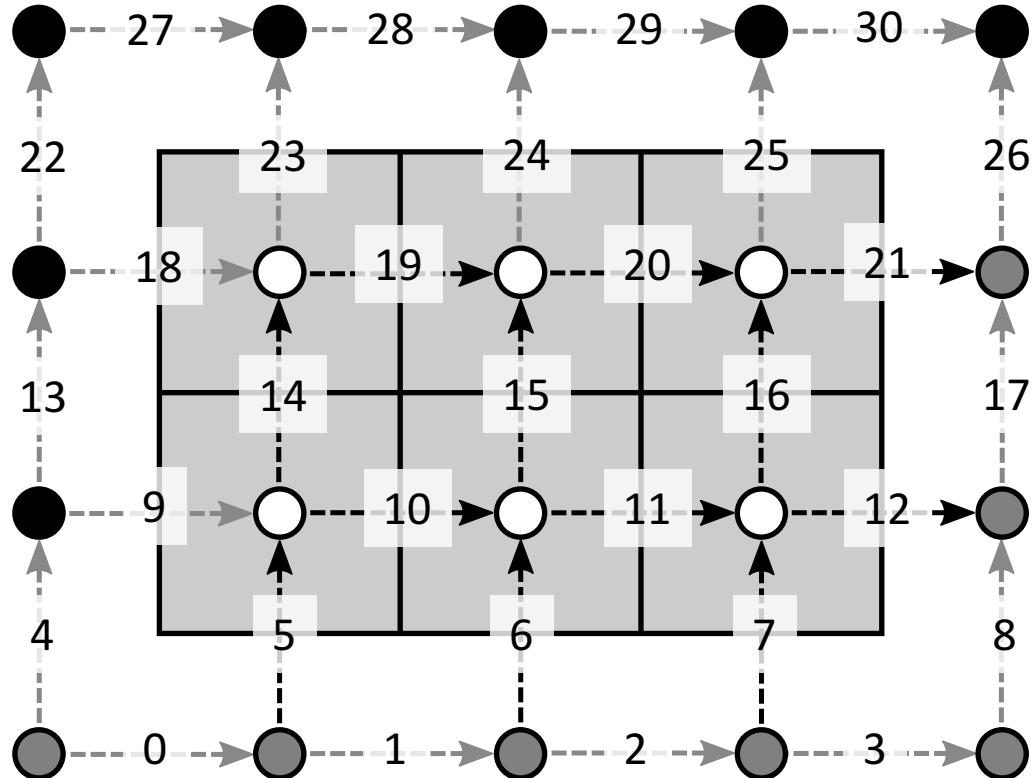
- Node (core)
- Node (closed boundary)
- Node (open boundary)
- Active Link
- ↔ Inactive Link
- / Face
- Cell

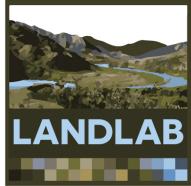


Active and inactive links

ACTIVE:
Connects two core nodes OR
a core and an open
boundary

INACTIVE:
Connects to one or more
closed boundary nodes OR
Connects two open
boundary nodes

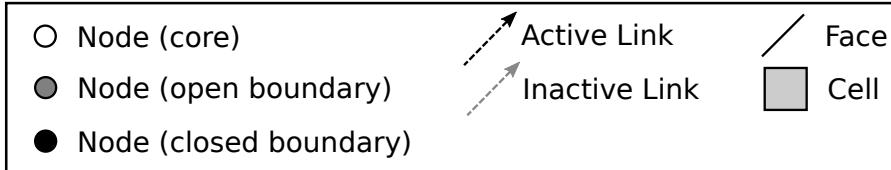
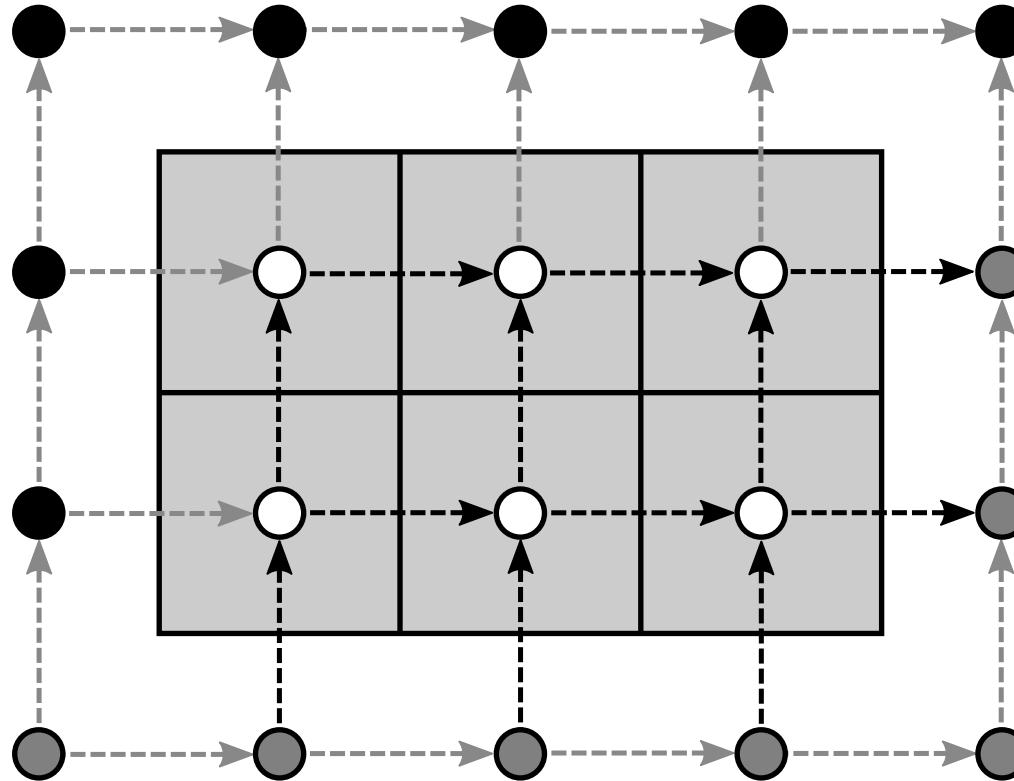


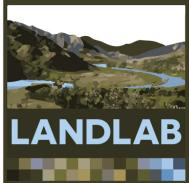


Grid elements: cells

*Cell =
polygon
bounded by
faces and
containing a
*node**

*Perimeter
nodes do not
have cells*



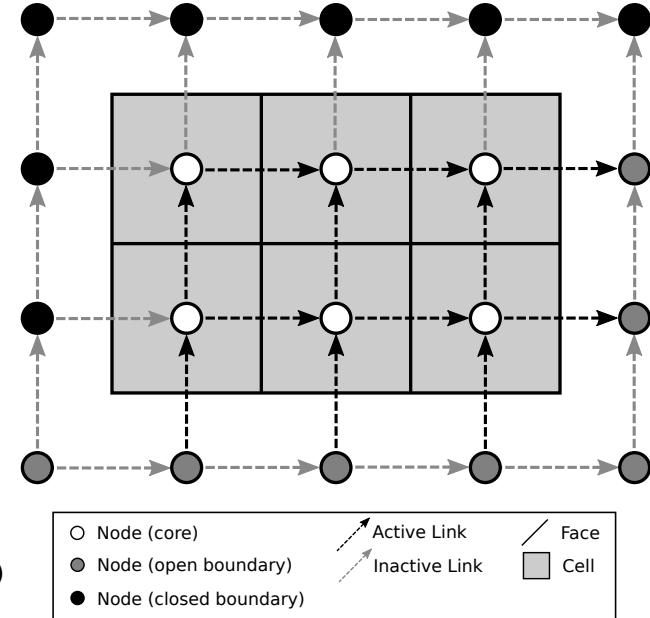


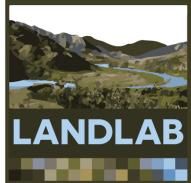
Grid elements: cells

Cells have:

- Area
- Faces
- A node

```
>>> rg.number_of_cells  
6  
>>> rg.area_of_cell  
array([ 100.,  100.,  100.,  100.,  100.,  100.])  
>>> rg.faces_at_cell  
array([[ 4,  7,  3,  0],  
       [ 5,  8,  4,  1],  
       [ 6,  9,  5,  2],  
       [11, 14, 10,  7],  
       [12, 15, 11,  8],  
       [13, 16, 12,  9]])  
>>> rg.node_at_cell  
array([ 6,  7,  8, 11, 12, 13])
```

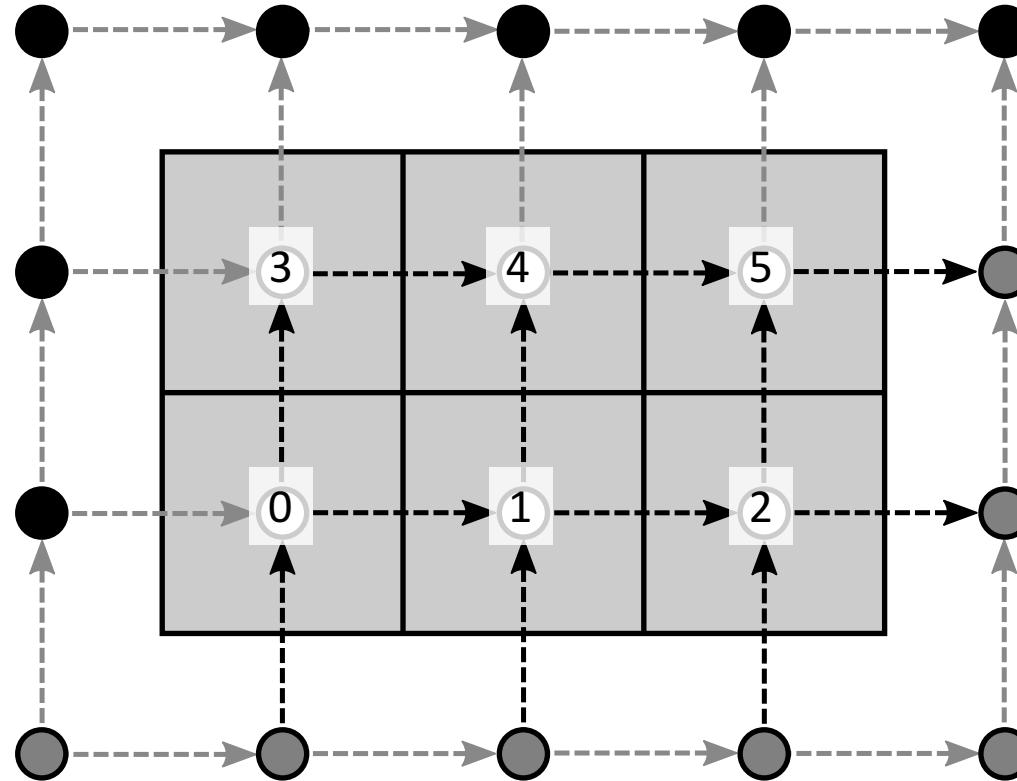




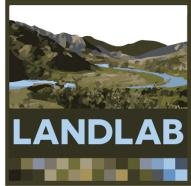
Cell numbering

Cells are sorted by y coordinate

Cells with equal y are sorted by x

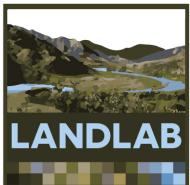


- Node (core)
 - Node (open boundary)
 - Node (closed boundary)
- Active Link Inactive Link Face
Cell



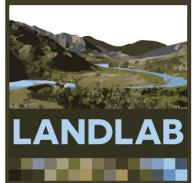
Fields: attaching data to the grid

- A **field** is a NumPy array containing data that are associated with a particular type of grid element (typically nodes or links)
- Fields are 1D arrays
- Values correspond to the element with the same ID. Example: value 5 of a node field belongs to node #5.
- Fields are “attached” to the grid (the grid object includes dictionaries listing all the fields)
- Fields have names (as strings)
- Create fields with grid functions `add_zeros`, `add_ones`, or `add_empty`



Fields: example

```
>>> h = rg.add_zeros('water__depth', at='node')
>>> h
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
       0.,  0.,  0.,  0.,  0.,  0.,  0.])
>>> h[1] = 100.0
>>> h
array([ 0., 100.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
       0.,  0.,  0.,  0.,  0.,  0.,  0.])
>>> rg.at_node['water__depth']
array([ 0., 100.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
       0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

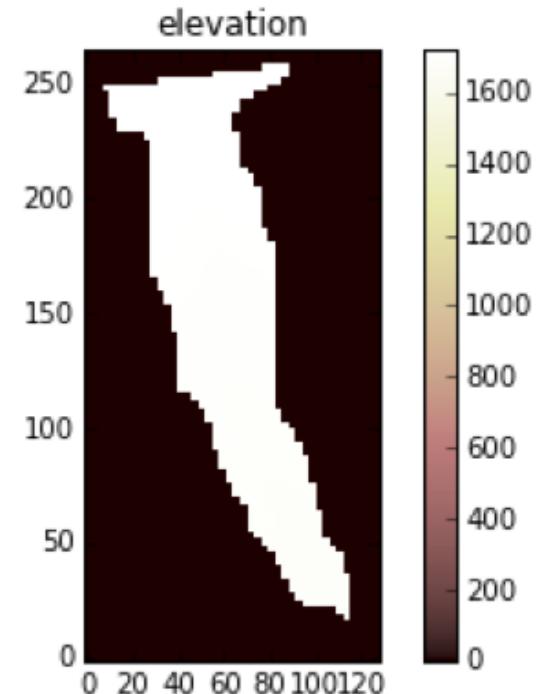


Reading raster digital terrain data

Landlab's `read_esri_ascii` function:

- Reads data from ESRI ASCII raster file
- Creates a `RasterModelGrid` and a data field
- Also: read/write netCDF files
- Example:

```
>>> from landlab.io import read_esri_ascii  
>>> (mg, z) = read_esri_ascii('west_bijou_gully.asc',  
                           name='elevation')
```



Staggered-grid schemes: Scalars at nodes, vectors at links

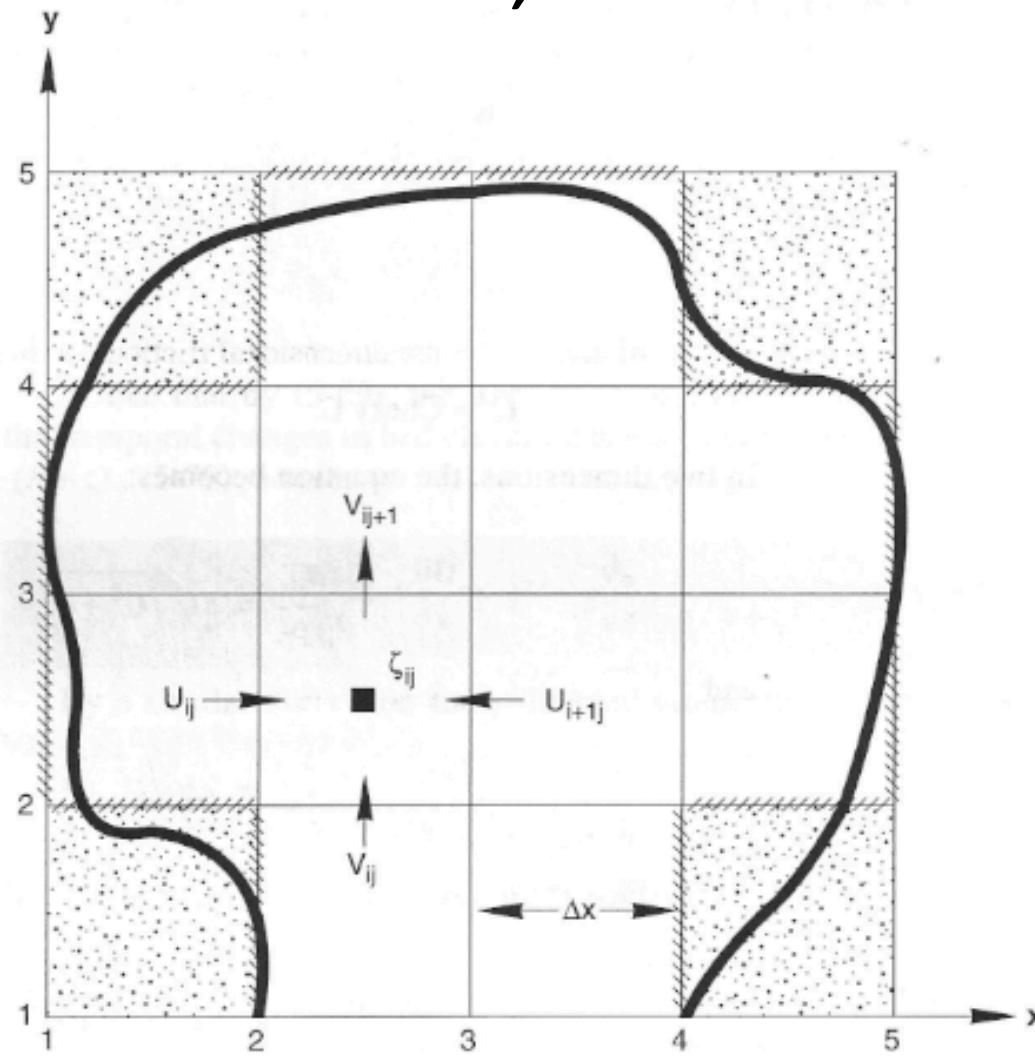


Figure 5-19

Discretization grid for 2-D circulation model. Slingerland, Harbaugh, and Furlong (1994)

Linear diffusion example

$$\frac{\partial \eta}{\partial t} = -\nabla \mathbf{q}_s$$

η = land-surface elevation

t = time

\mathbf{q}_s = sediment flux $[L^2/T]$

$$\mathbf{q}_s = -D \nabla \eta$$

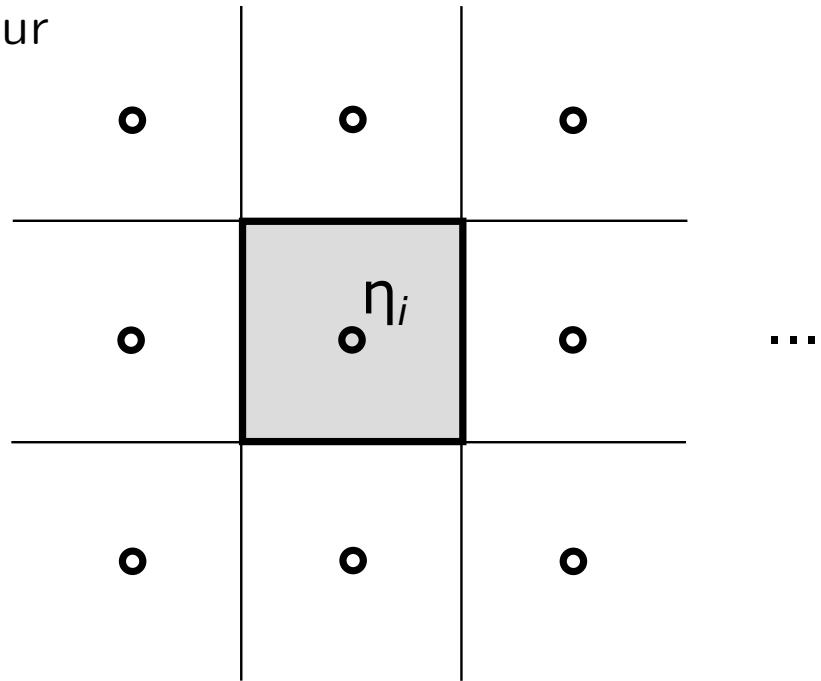
D = transport coefficient $[L^2/T]$

The numerical problem: finite-volume solution scheme

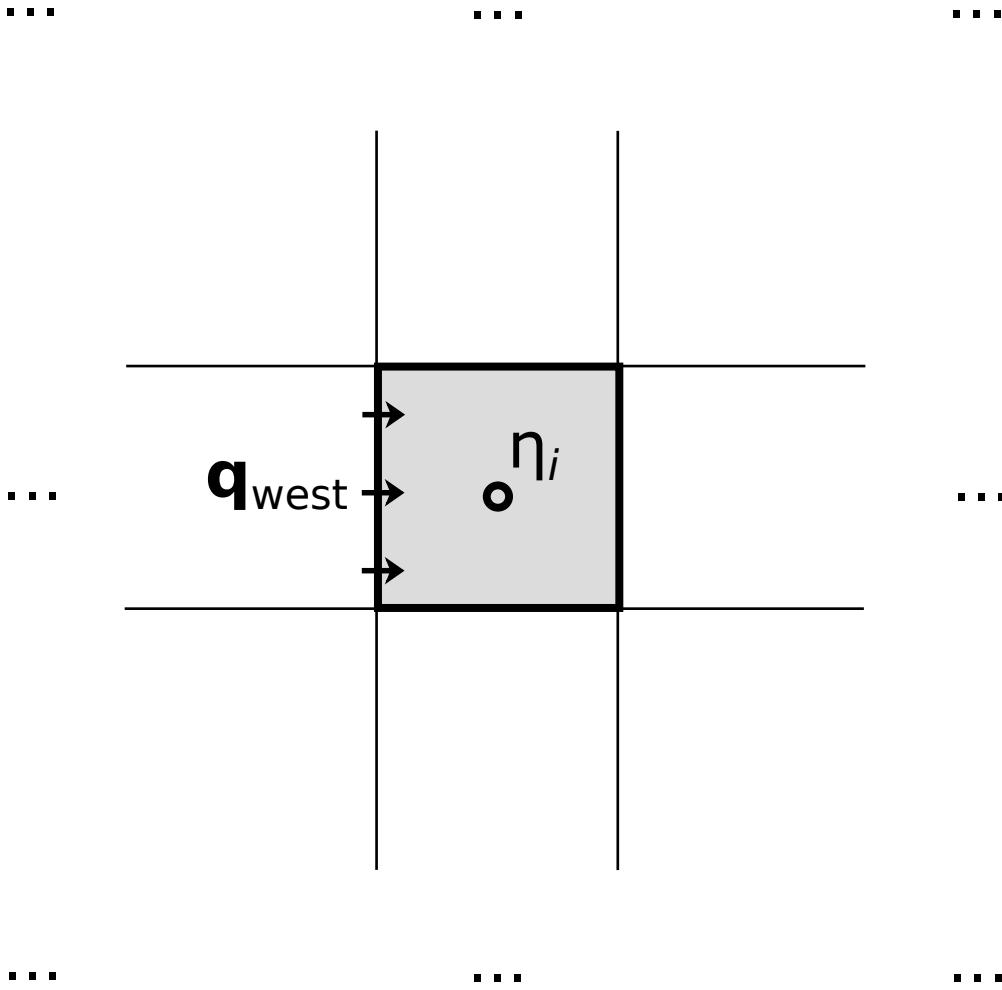
Each interior node i lies within a *cell* whose surface area is A_i .

We can write mass balance for cell i in terms of sediment fluxes across each of its four faces:

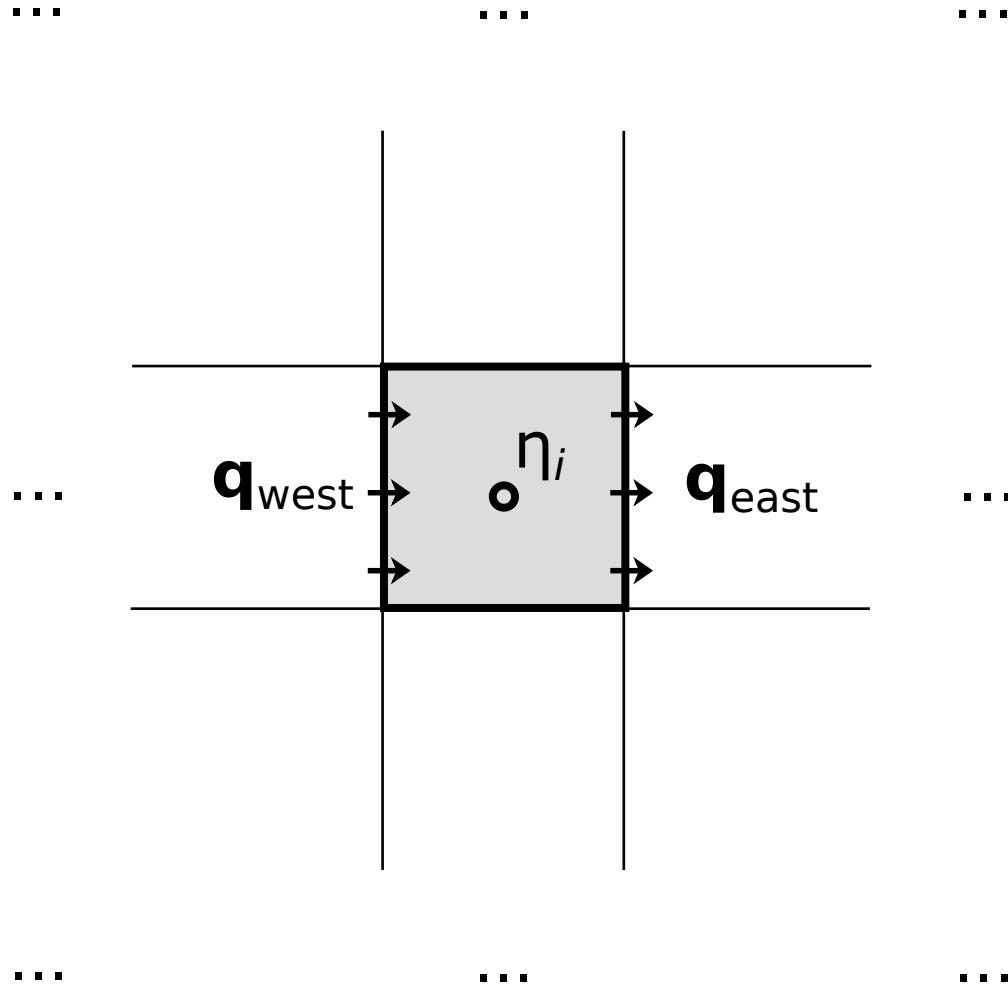
$$\frac{d\eta_i}{dt} = \frac{1}{A_i} \sum_{j=1}^4 \Delta x q_j$$



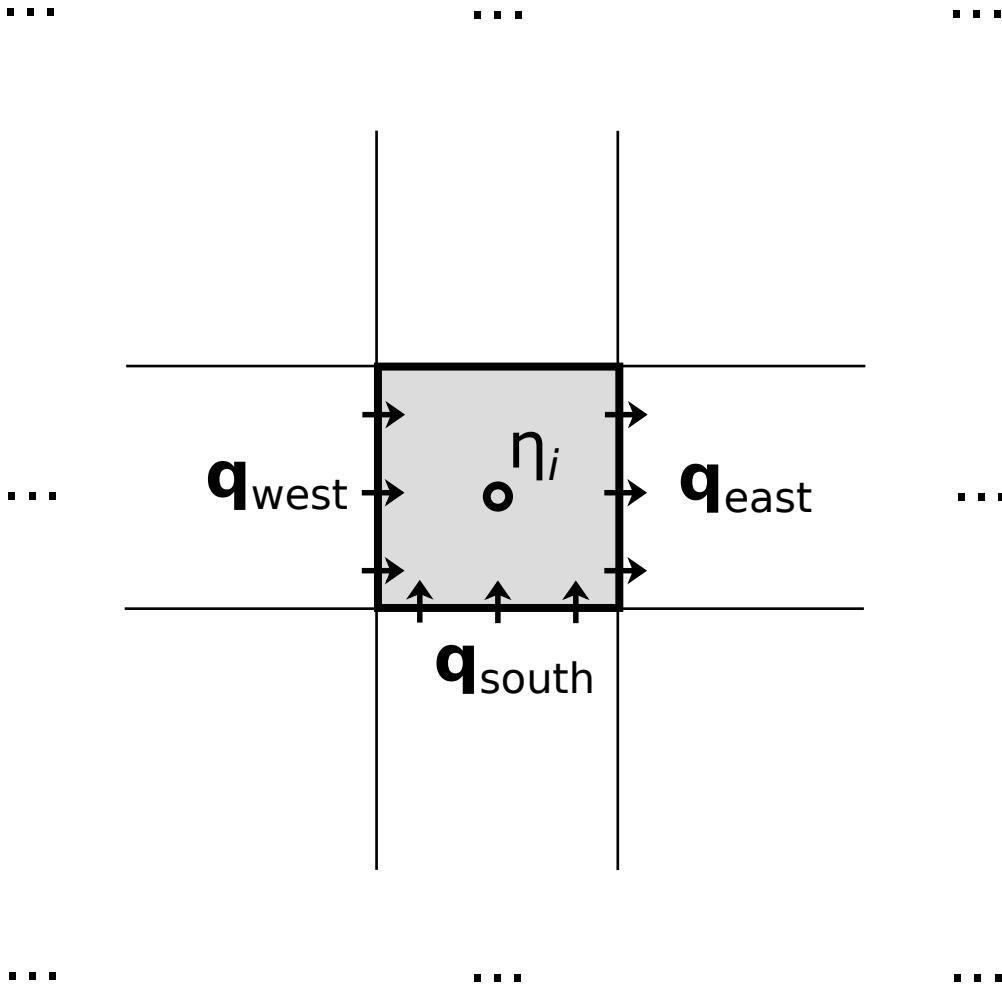
$$\frac{d\eta_i}{dt} = \frac{\Delta x}{A_i} [q_{\text{west}} \dots$$



$$\frac{d\eta_i}{dt} = \frac{\Delta x}{A_i} [\mathbf{q}_{\text{west}} - \mathbf{q}_{\text{east}} \dots$$

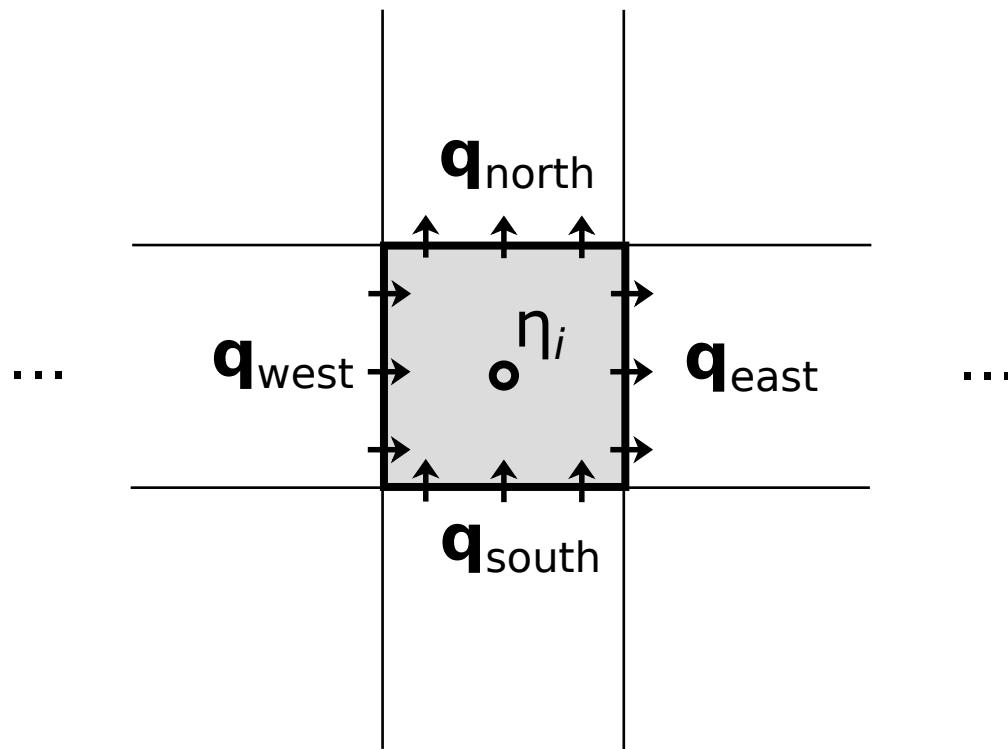


$$\frac{d\eta_i}{dt} = \frac{\Delta x}{A_i} [\mathbf{q}_{\text{west}} - \mathbf{q}_{\text{east}} + \mathbf{q}_{\text{south}} \dots]$$



$$\frac{d\eta_i}{dt} = \frac{\Delta x}{A_i} [\mathbf{q}_{\text{west}} - \mathbf{q}_{\text{east}} + \mathbf{q}_{\text{south}} - \mathbf{q}_{\text{north}}]$$

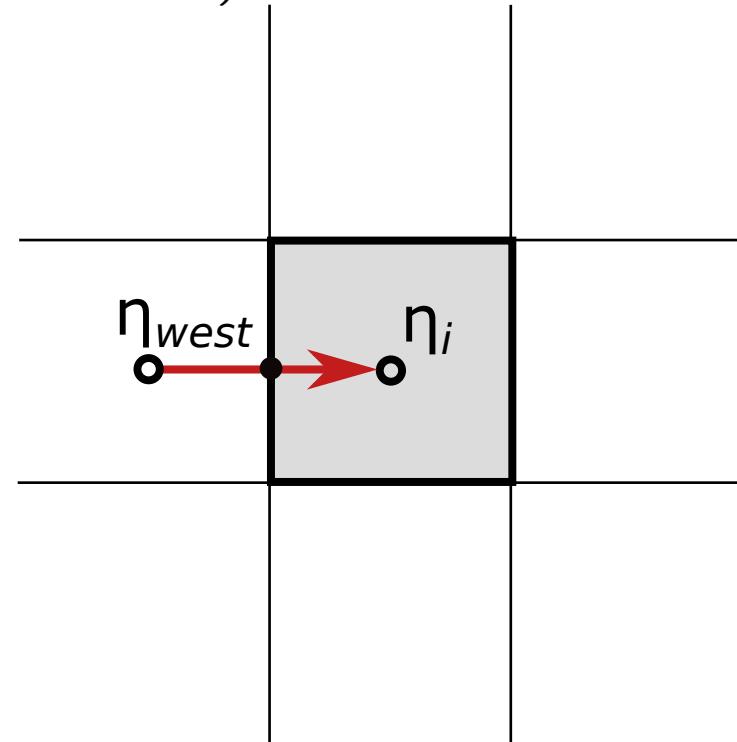
...



...

Flux depends on gradient, which is calculated between adjacent nodes:

$$q_{\text{west}} = -D \frac{\partial \eta}{\partial x} \Big|_{(\text{west face})} \approx -D \left(\frac{\eta_i - \eta_{\text{west}}}{\Delta x} \right)$$

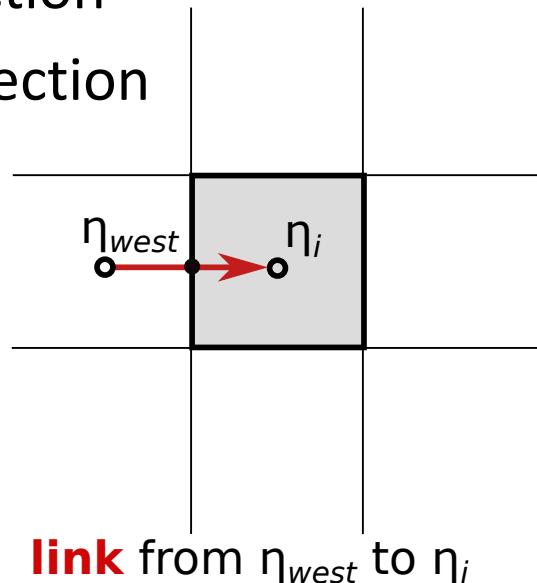


link from η_{west} to η_i

Calculating the gradient of a scalar field

```
>>> deta_dx = rg.calc_grad_at_link(eta)
```

- η is a scalar defined at nodes
- One value of $d\eta/dx$ for every link
- Positive when η increases in the link direction
- Negative when η decreases in the link direction

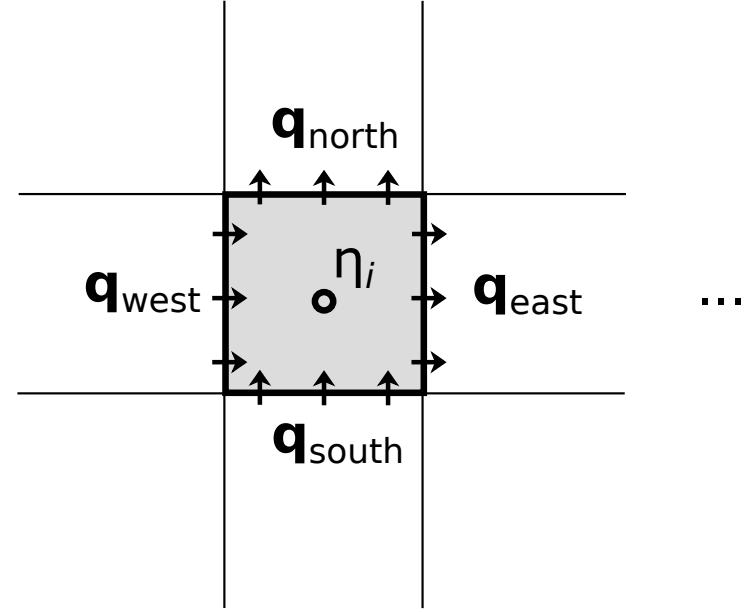


Calculating the divergence of a gradient field

```
>>> q = -D * data_dx
```

```
>>> dqdx = rg.calc_flux_div_at_node(q)
```

- q is a vector defined at links
- One value of $dqdx$ for every node
- Positive when net flux is outwards



Q: What if you need a scalar value at a link?

A: Landlab's mapping functions



```
>>> h_link = rg.map_mean_of_link_nodes_to_link(h)
```



```
>>> h_link = rg.map_value_at_max_node_to_link(w, h)
```

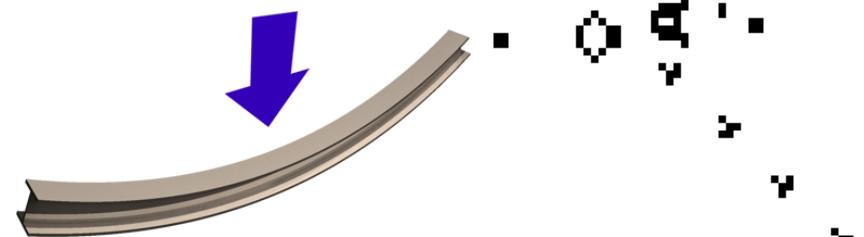
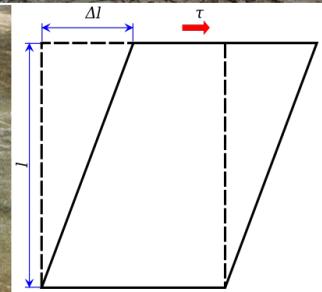
Components

- A **component** is a self-contained piece of code that typically represents one process
- Components have a standardized interface that allows them to be easily coupled with one another using a Python script
- Components are normally implemented as Python classes. For example:

```
>>> ld = LinearDiffuser(rg, linear_diffusivity=0.01)
>>> ld.run_one_step(dt=1.0)
```

The components

- Describe individual surface processes
- “Plug & Play”
- Standard interface
- Use the library, or BYO



Documentation: Users' Guide

The screenshot shows a browser window with the following details:

- Address Bar:** GitHub, Inc. (US) | <https://github.com/landlab/landlab/wiki/User-Guide>
- Toolbar:** Back, Home, Search, Download, etc.
- Page Content:**
 - User Guide** (Section Header)
 - Jenny Knuth edited this page on Mar 1 · 53 revisions
 - Link:** <https://github.com/landlab/landlab/wiki/User-Guide> (highlighted in red)
 - Navigation:** Landlab | About | Examples | User Guide | Developer API | Tutorials | FAQs
 - Sidebar:** Pages (23) with a search bar and a list of pages including Home, About, Build a Model, CellLab CTS 2015 Users Manual, Components, Correcting Python Version, Developing with github and git, Examples, FAQs, Grid, Installing Landlab, Installing Landlab from source code ("developer install"), Installing Landlab with Anaconda, Installing Python, and Introducing Landlab 1.0beta.

User Guide

Jenny Knuth edited this page on Mar 1 · 53 revisions

<https://github.com/landlab/landlab/wiki/User-Guide>

[Landlab](#) | [About](#) | [Examples](#) | [User Guide](#) | [Developer API](#) | [Tutorials](#) | [FAQs](#)

Edit

New Page

Installation

- Instructions for a standard install
- Installing from source code, "developer install"

Basics of Python

If you are new to Python or scientific programming, start with an intro to the nuts and bolts of Landlab:

[Python, NumPy, SciPy, and Cython](#)

- Why Python?
- Getting to know Python
 - If you know MatLab...
- NumPy, SciPy, and efficient coding style
- Cython

Landlab's grid

▼ Pages (23)

Find a Page...

[Home](#)

[About](#)

[Build a Model](#)

[CellLab CTS 2015 Users Manual](#)

[Components](#)

[Correcting Python Version](#)

[Developing with github and git](#)

[Examples](#)

[FAQs](#)

[Grid](#)

[Installing Landlab](#)

[Installing Landlab from source code \("developer install"\)](#)

[Installing Landlab with Anaconda](#)

[Installing Python](#)

[Introducing Landlab 1.0beta](#)

Documentation: Reference / API

landlab.readthedocs.io/en/latest/ <http://landlab.readthedocs.io>



Find Landlab's User Guide on the [Landlab Wiki](#)

Landlab Reference Manual and API Documentation

The *Landlab Developer API* is a general reference manual for Landlab.

Grids

Grid types

As of Landlab version 0.2, there are four types of Landlab grid:

- Raster
- Voronoi-Delaunay
- Hex
- Radial

The base class is `ModelGrid` with subclasses `RasterModelGrid` and `VoronoiDelaunayGrid`.

`VoronoiDelaunayGrid` has two further specialized subclasses: `HexModelGrid` and `RadialModelGrid`.

Methods and properties common to all grids

- General class methods and attributes of the `landlab.grid.base` module
 - Getting Information about a Grid
 - Information about the grid as a whole
 - Information about nodes
 - Information about links
 - Information about cells
 - Information about faces
 - Information about patches
 - Information about corners
 - Data Fields in ModelGrid
 - Create Field Arrays

Table Of Contents

Landlab Reference Manual and API Documentation

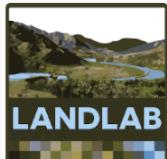
- Grids
 - Grid types
 - Methods and properties common to all grids
 - Specialized methods and properties for Rectilinear Grids 'raster grids'
 - Specialized methods and properties for Voronoi-Delaunay grids
 - Specialized methods and properties for hex grids
 - Specialized methods and properties for radial grids
- Components
 - Hillslope geomorphology
 - Fluvial geomorphology
 - Flow routing
 - Shallow water hydrodynamics
 - Land surface hydrology
 - Landslides
 - Vegetation
 - Precipitation
 - Terrain Analysis
 - Tectonics
 - Fire
 - Initial conditions: random field generators
 - The Component base class
 - Input/Output (IO)

Documentation: source code, tutorials, etc., publicly available on GitHub

← → ↻ <https://github.com/landlab>

This organization Search Pull requests Issues Gist

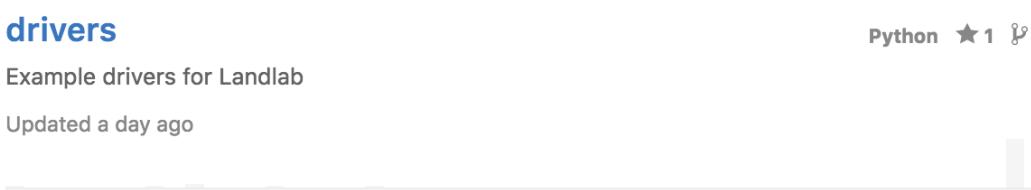
 [This organization](#) [Search](#) [Pull requests](#) [Issues](#) [Gist](#)

 **Landlab**
a python toolkit for modeling earth surface processes
<http://landlab.github.io>

[Repositories](#) [People 9](#) [Teams 2](#) [Settings](#)

[Filters](#) [New repository](#)

landlab
Landlab codebase, wiki, and tests
Updated 6 hours ago  Python ★ 30 ⚡ 62

drivers
Example drivers for Landlab
Updated a day ago  Python ★ 1 ⚡ 2

tutorials
Landlab tutorials
Updated 10 days ago  Jupyter Notebook ★ 0 ⚡ 1

People 9 >

 [Invite someone](#)

This repository Search Pull requests Issues Gist

Unwatch 14 Star 30 Fork 62

Code Issues 89 Pull requests 5 Wiki Pulse Graphs Settings

Tutorials

saisiddu edited this page on Jul 22 · 53 revisions

Edit New Page

[Landlab](#) | [About](#) | [Examples](#) | [User Guide](#) | [Developer API](#) | [Tutorials](#) | [FAQs](#)

The [Landlab tutorials](#) repository contains [IPython notebooks](#) that are both unexpanded (so you can run them in an IPython notebook viewer) and expanded (so you can read them like a regular text tutorial) along with code examples. Landlab's tutorials repo can also be accessed from [nbviewer](#).

IPython notebook tutorials

Instructions on how to run an IPython notebook can be found here:

<https://github.com/landlab/tutorials/blob/master/README.md>

A short IPython notebook tutorial along with a screencast can be found here (the tutorial uses an example with statistics, but you can substitute Landlab!):

<http://www.randalolson.com/2012/05/12/a-short-demo-on-how-to-use-ipython-notebook-as-a-research-notebook/>

[Click here to download all the tutorials](#)

A suggested introduction to Landlab follows roughly this order:

- [Introduction to Python and NumPy](#). *Learn about:* The very basics of Python.
- [Introduction to Landlab: example model of fault-scarp degradation](#). A short overview of some of the things Landlab can do.
- [Introduction to the model grid object](#). Grid topology; how landlab represents data; connectivity of grid elements.
- [Introduction to Landlab data fields](#). How Landlab stores spatial data on the grid; a little on naming conventions.

Pages 26

Find a Page...

[Home](#)

[About](#)

[Build a Model](#)

[CellLab CTS 2015 Users Manual](#)

[Components](#)

[Correcting Python Version](#)

[Developing with github and git](#)

[Examples](#)

[FAQs](#)

[Grid](#)

[Installing Landlab](#)

[Installing Landlab from source code \("developer install"\)](#)

[Installing Landlab with Anaconda](#)

[Installing Python](#)

If you still need to install:

<http://landlab.github.io>

→ Install

Follow instructions

How to update Landlab

In terminal window or command prompt:

`pip uninstall landlab`

`conda install landlab -c landlab`

How to download and run tutorials

- Go to:
<https://github.com/landlab/landlab/wiki/Tutorials>
- Click:
Click here to download all the tutorials
- Save ZIP
- Double-click to unpack
- In terminal or command window, **navigate to new folder**
- Enter: `jupyter notebook`
- Shift-Enter to move through each cell