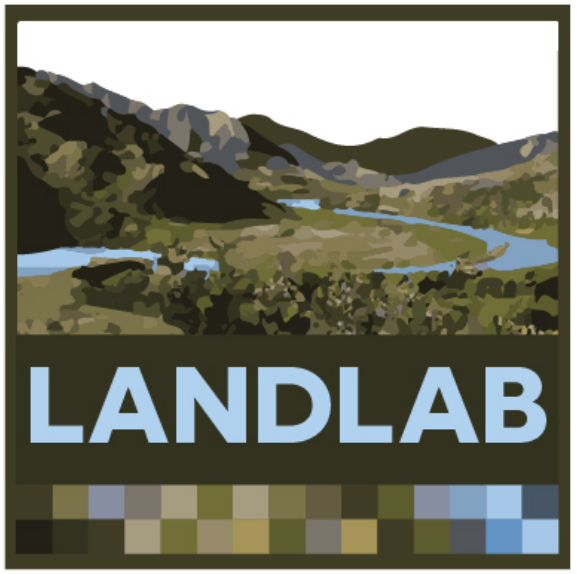
Modeling with the Landlab toolkit

Nicole Gasparini, Tulane University

UNIVERSITY of WASHINGTON

University of Colorado Boulder

Tulane University



CSDMS
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

HYDROSHARE

Material in this presentation comes from the Landlab team (in no particular order):

Tulane Team : Jordan Adams(*) & Nathan Lyons[#]

University of Colorado Boulder : Greg Tucker, Eric Hutton, Katy Barnhart[#], & Margaux Mouchene[#]

University of Washington : Erkan Istanbulluoglu, Sai Nudurupati*, and Christina Bandaragoda

Cardiff : Dan Hobley([#])

Goals for this class:

- You will have a new appreciation for how numerical models are used and what we can learn from them.
- You will have a better idea of what Landlab is and what it can do.
- You will be able to build a Landlab model by lunch.
- You will have a tool that will enable you to better understand surface processes and/or a tool that you can use for research and/or a tool that you can use when teaching.

I have made these assumptions for today:

- You have installed the Anaconda Python distribution and Landlab on your laptop.
- You are familiar with coding of some sort and understand coding basics such as loops, if statements, arrays…
- You will actively participate in this class, following along with examples on your computer, trying the exercises, using the Landlab help pages, and googling for help when you need it.
  - In other words, I want you to be on your computer while I'm lecturing, but hopefully not on email or Twitter.
- You will try to solve problems on your own, and you will also help your classmates and work together.

Assumptions I make in every class I teach:

- If you have a question you will ask it.
- If you are uncomfortable you will tell me.
- If I ask you a question, you will try to answer, but if you cannot, it is fine to say “I pass”.
- You will treat me and your classmates with respect. We are all working towards the same goal of learning about modeling with Landlab today. Leave bias outside the classroom *to the best of your ability*.

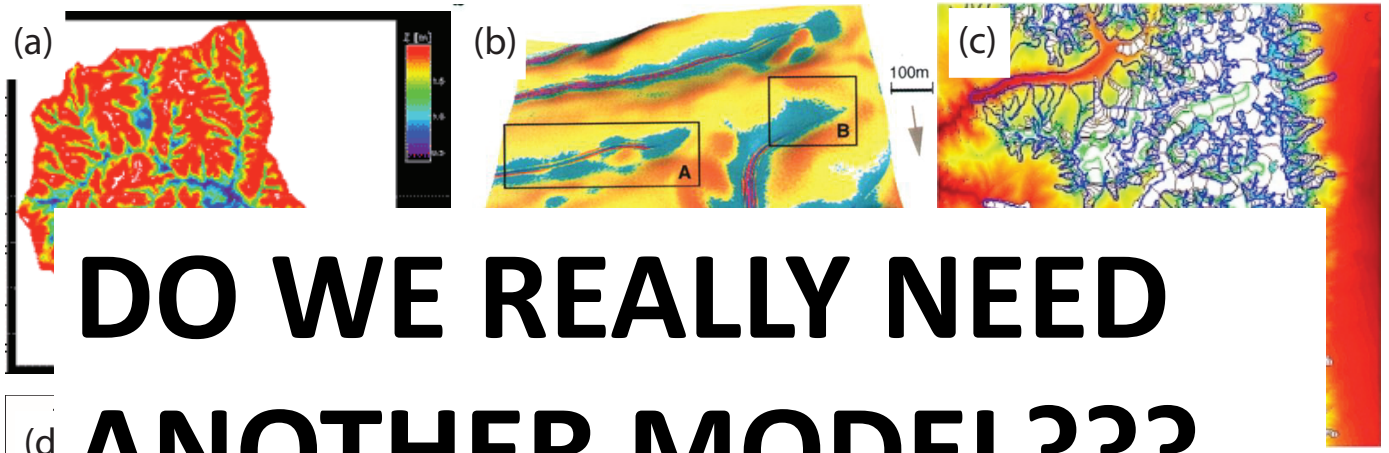
- Similarly,
  - I will do my best to answer all of your questions.
  - I will treat you with respect and without bias *to the best of my ability*.

Lots of modeling approaches out there...

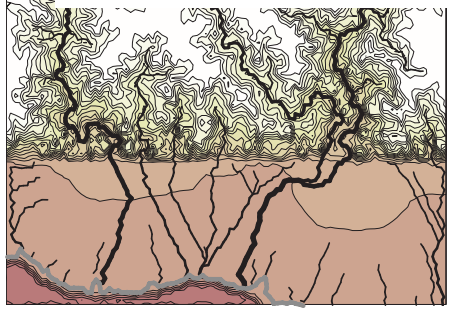
CATCHMENT HYDROLOGY
(Ivanov et al., 2004)

SOIL EROSION
(Mitas and Mitasova, 1998)

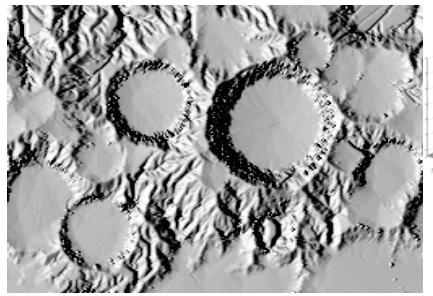
GLACIER DYNAMICS
(Kessler et al., 2006)



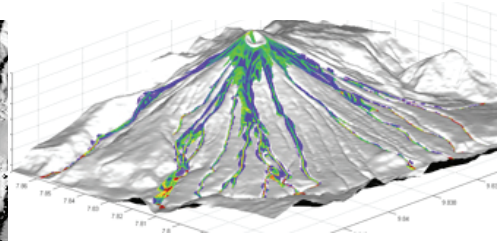
DO WE REALLY NEED ANOTHER MODEL???



LANDSCAPE EVOLUTION
(Tucker and Hancock, 2010)



IMPACT CRATERING AND DEGRADATION
(Howard, 2007)



LAVA FLOWS
(Kelfoun et al., 2009)

Challenges with many academic models:

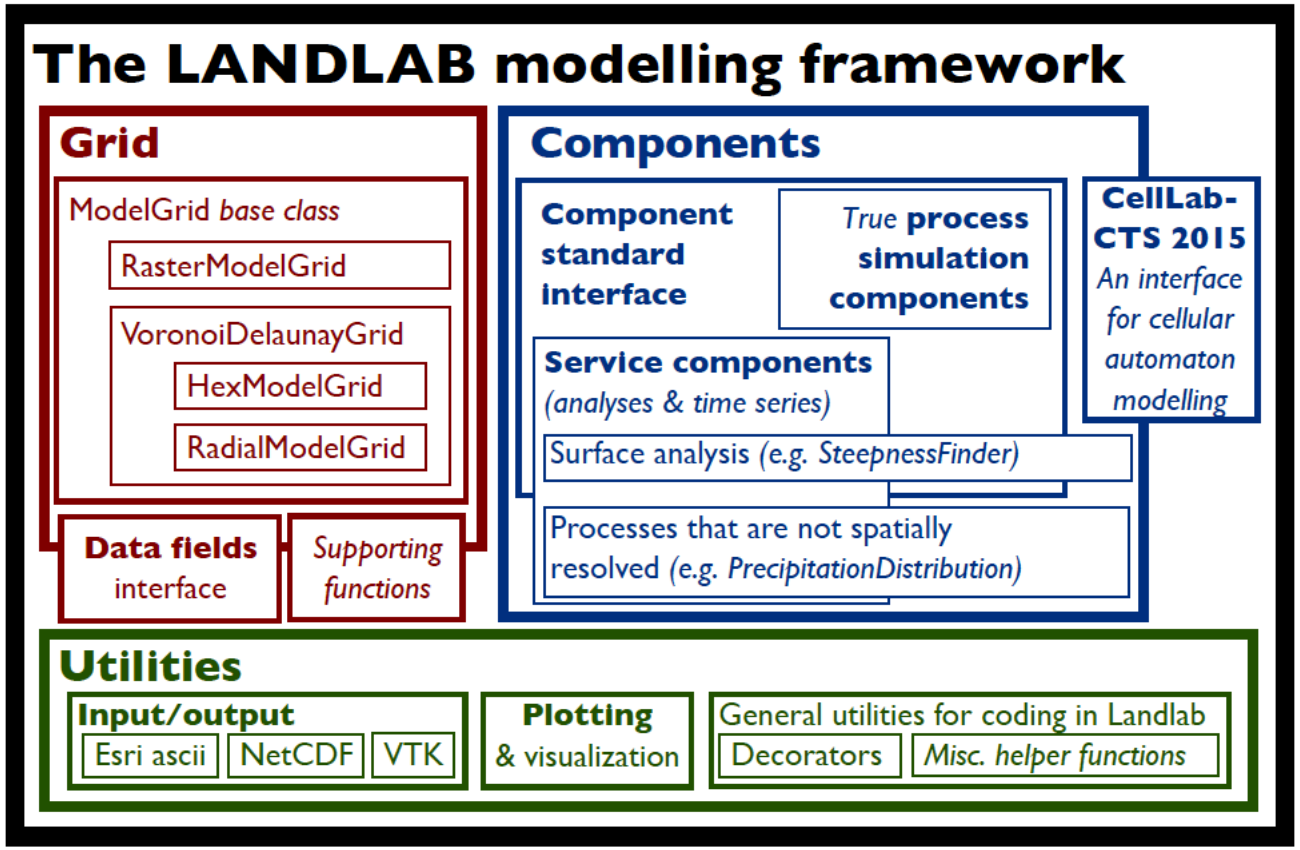
- built to answer a specific research question and not easily adaptable.
- poorly documented.
- not tested.
- platform dependent.
- not open source.

LANDLAB

What is Landlab?

- A toolkit for building models of processes that happen on the earth surface – e.g. overland flow, bedrock river incision, growth and death of vegetation.
- Not a landscape evolution model (LEM), but you can build LEMs with Landlab if you want to.
- Open source.
- **A Python library**.
- Anyone can contribute to the library (following some basic rules).
- Continuously tested.

What is Landlab?

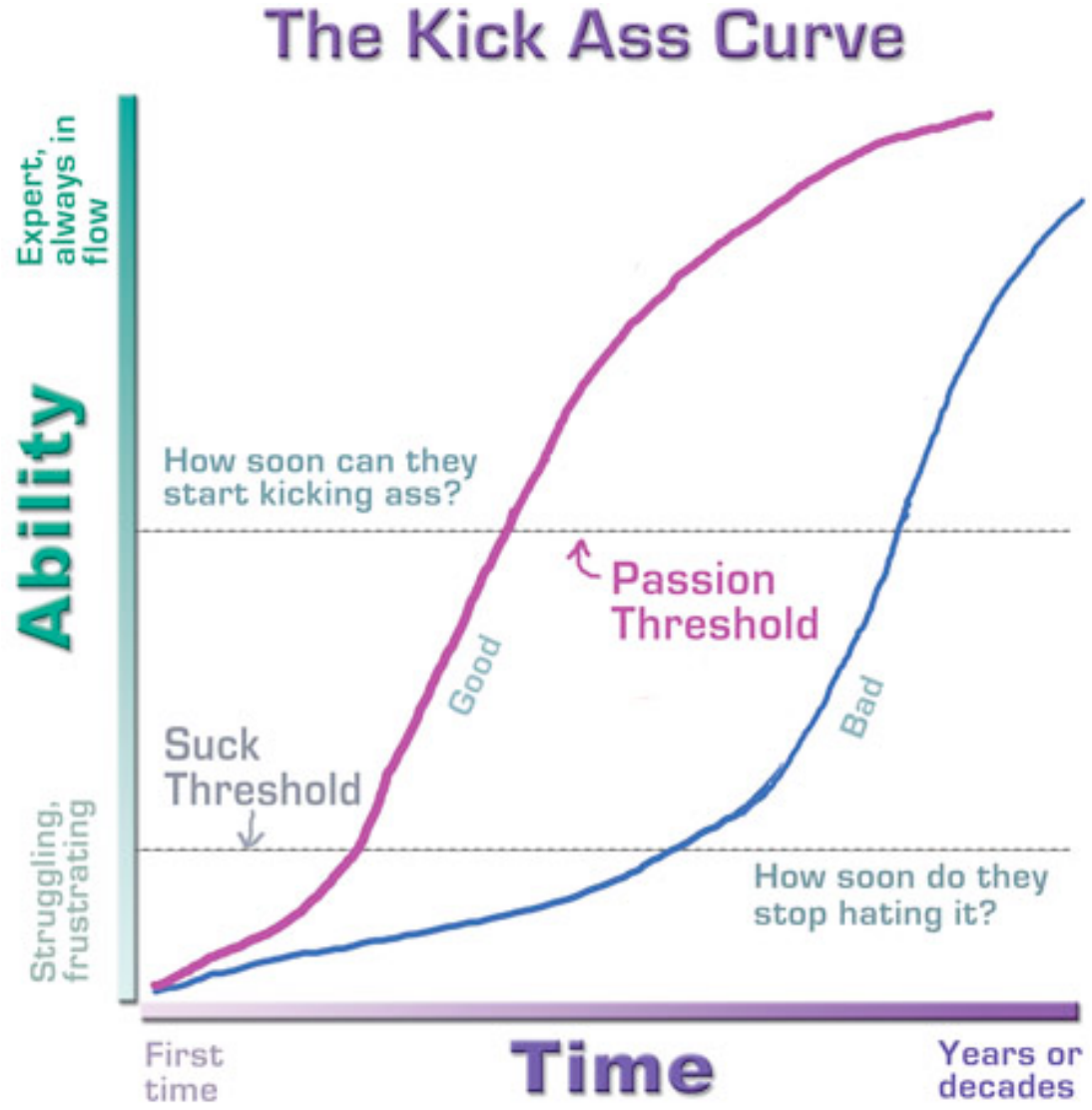


(Hobley et al., 2017, ESURF)

Landlab is also:

- Extensively document.
- Has supporting materials such as:
  - Tutorials for learning Landlab.
  - Teaching materials for learning concepts in geomorphology and hydrology.
- An issues page where you can search for answers and post questions.
- A community of users, including **YOU**.

GOAL: Make the time to pass the “suck threshold” short.



http://headrush.typepad.com/creating_passionate_users/2005/10/getting_users_p.html

Earth Surf. Dynam., 5, 21–46, 2017
www.earth-surf-dynam.net/5/21/2017/
doi:10.5194/esurf-5-21-2017

Earth Surface Dynamics

Creative computing with Landlab: an open-source toolkit for building, coupling, and exploring two-dimensional numerical models of Earth-surface dynamics

Daniel E. J. Hobley[1,2,3], Jordan M. Adams[4], Sai Siddhartha Nudurupati[5], Eric W. H. Hutton[6], Nicole M. Gasparini[4], Erkan Istanbulluoglu[5], and Gregory E. Tucker[1,2]

Earth Surf. Dynam., 6, 49–75, 2018
https://doi.org/10.5194/esurf-6-49-2018

Earth Surface Dynamics

Earth Surf. Dynam., 6, 563–582, 2018
https://doi.org/10.5194/esurf-6-563-2018

A hydroclimatological approach to predicting regional landslide probability using Landlab

Ronda Strauch[1], Erkan Istanbulluoglu[1], Sai Siddhartha Nudurupati[1], Christina Bandaragoda[1], Nicole M. Gasparini[2], and Gregory E. Tucker[3,4]

A lattice grain model of hillslope evolution

Gregory E. Tucker[1], Scott W. McCoy[2], and Daniel E. J. Hobley[3]

THE GEOLOGICAL SOCIETY OF AMERICA®

Off-fault deformation rate along the southern San Andreas fault at Mecca Hills, southern California, inferred from landscape modeling of curved drainages

Harrison J. Gray[1], Charles M. Shobe[1], Daniel E.J. Hobley[2], Gregory E. Tucker[1], Alison R. Duvall[3], Sarah A. Harbert[3], and Lewis A. Owen[4]

Geosci. Model Dev., 9, 823–839, 2016
www.geosci-model-dev.net/9/823/2016/
doi:10.5194/gmd-9-823-2016

Geoscientific Model Development

CellLab-CTS 2015: continuous-time stochastic cellular automaton modeling using Landlab

Gregory E. Tucker[1,2], Daniel E. J. Hobley[1,2], Eric Hutton[3], Nicole M. Gasparini[4], Erkan Istanbulluoglu[5], Jordan M. Adams[4], and Sai Siddartha Nudurupati[5]

Geosci. Model Dev., 10, 1645–1663, 2017
www.geosci-model-dev.net/10/1645/2017/
doi:10.5194/gmd-10-1645-2017

Geoscientific Model Development

The Landlab v1.0 OverlandFlow component: a Python tool for computing shallow-water flow across watersheds

Jordan M. Adams[1], Nicole M. Gasparini[1], Daniel E. J. Hobley[2], Gregory E. Tucker[3,4], Eric W. H. Hutton[5], Sai S. Nudurupati[6], and Erkan Istanbulluoglu[6]

Geosci. Model Dev., 10, 4577–4604, 2017
https://doi.org/10.5194/gmd-10-4577-2017

Geoscientific Model Development

The SPACE 1.0 model: a Landlab component for 2-D calculation of sediment transport, bedrock erosion, and landscape evolution

Charles M. Shobe, Gregory E. Tucker, and Katherine R. Barnhart

LANDLAB

Journal of Geophysical Research: Earth Surface

RESEARCH ARTICLE
10.1029/2017JF004509

Key Points:
- We model expansion of fluvial networks into low-relief landscapes with depressions, representing the postglacial U.S. Central Lowland
- When depressions are connected to

Modeled Postglacial Landscape Evolution at the Southern Margin of the Laurentide Ice Sheet: Hydrological Connection of Uplands Controls the Pace and Style of Fluvial Network Expansion

Jingtao Lai[1] and **Alison M. Anders**[1]

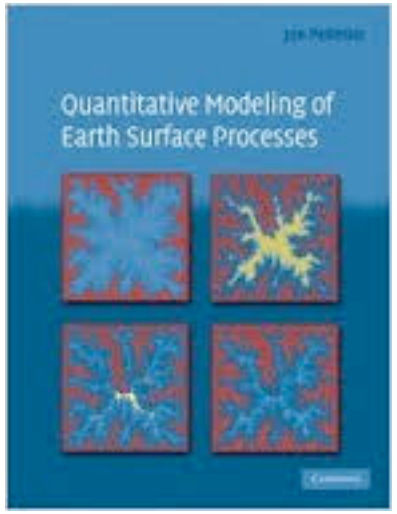
https://doi.org/10.5194/esurf-2018-13

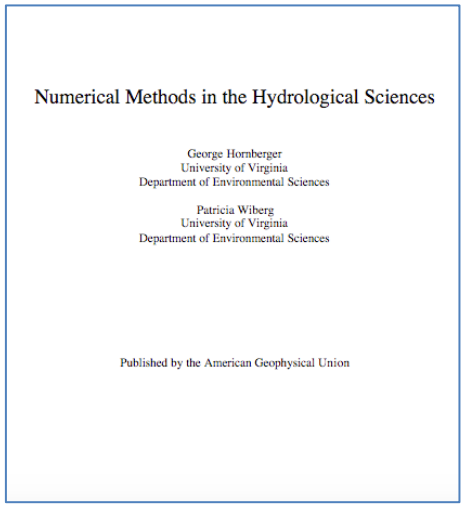
Research article

Effect of changing vegetation on denudation (part 2): Landscape response to transient climate and vegetation cover

Manuel Schmid[1], **Todd A. Ehlers**[1], **Christian Werner**[2], **Thomas Hickler**[2,3], **and Juan-Pablo Fuentes-Espoz**[4]

Why build numerical models?
What are numerical models?




Numerical Methods in the Hydrological Sciences

George Hornberger
University of Virginia
Department of Environmental Sciences

Patricia Wiberg
University of Virginia
Department of Environmental Sciences

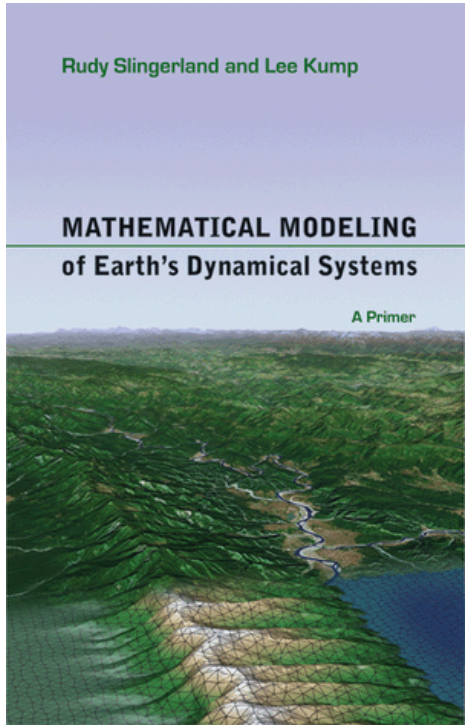
Published by the American Geophysical Union

NATURE

DYNAMICAL MODEL

NUMERICAL ALGORITHM

SOFTWARE



LANDLAB

landlab.github.io

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

https://github.com/landlab/landlab/wiki/Teach-Yourself-Landlab!

http://landlab.readthedocs.io/en/latest/#developer-documentation

https://github.com/landlab/landlab/wiki/Tutorials

https://github.com/landlab/tutorials

https://github.com/landlab/landlab_teaching_tools

https://github.com/landlab

https://github.com/landlab/landlab/issues

Your average undergrad knows nothing about Landlab and numerical modeling.

- Step in their shoes!
- They can still learn from models.
- You have already downloaded the landlab_teaching_tools repository (repo).
  - Navigate to geomorphology_exercises/hillslope_notebooks/ in a terminal.
  - Open the jupyter notebook
  - $ jupyter notebook

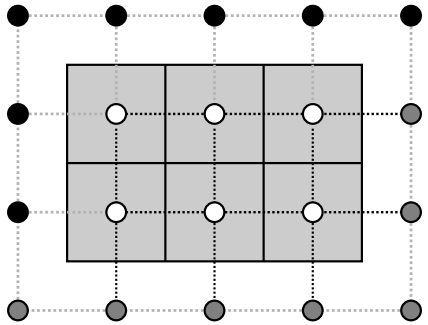
Congratulations!

- You have run a Landlab model!
- Now build your own model.
- First, a little more about what is in the Landlab library and other miscellany.

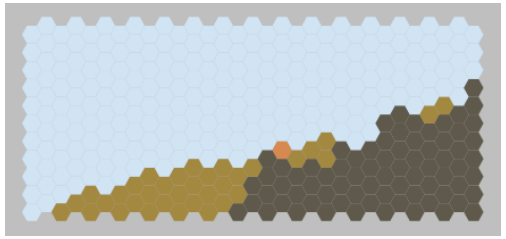
What Landlab provides

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

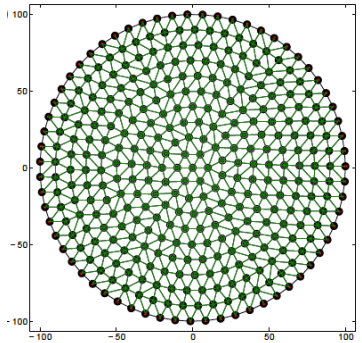
RASTER



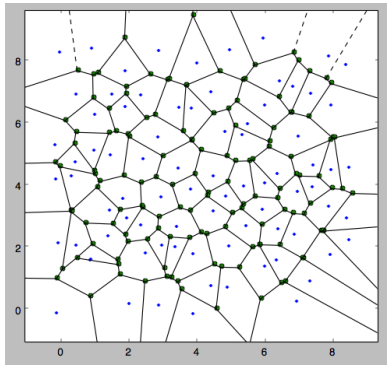
HEXAGONAL



RADIAL



VORONOI / DELAUNAY



Using Landlab grid

Aim: make it easier to set up a 2D numerical model grid

Grid data and functions contained in a single Python object

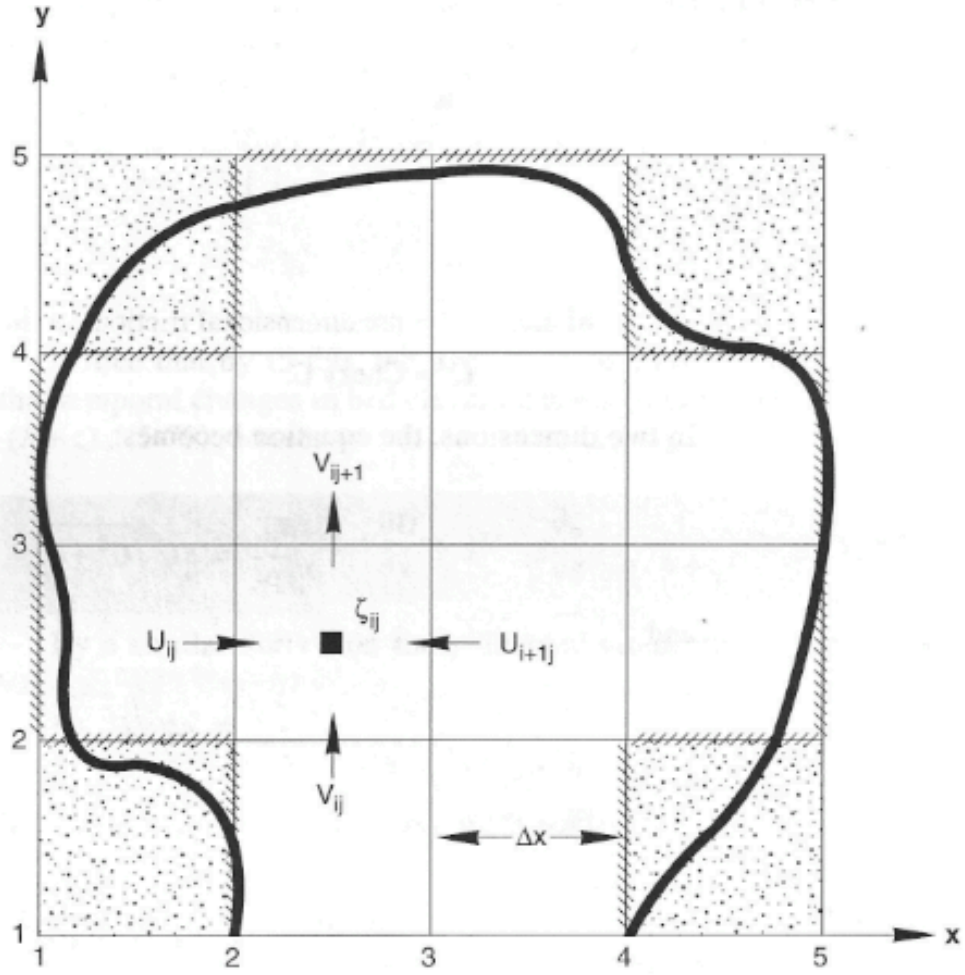
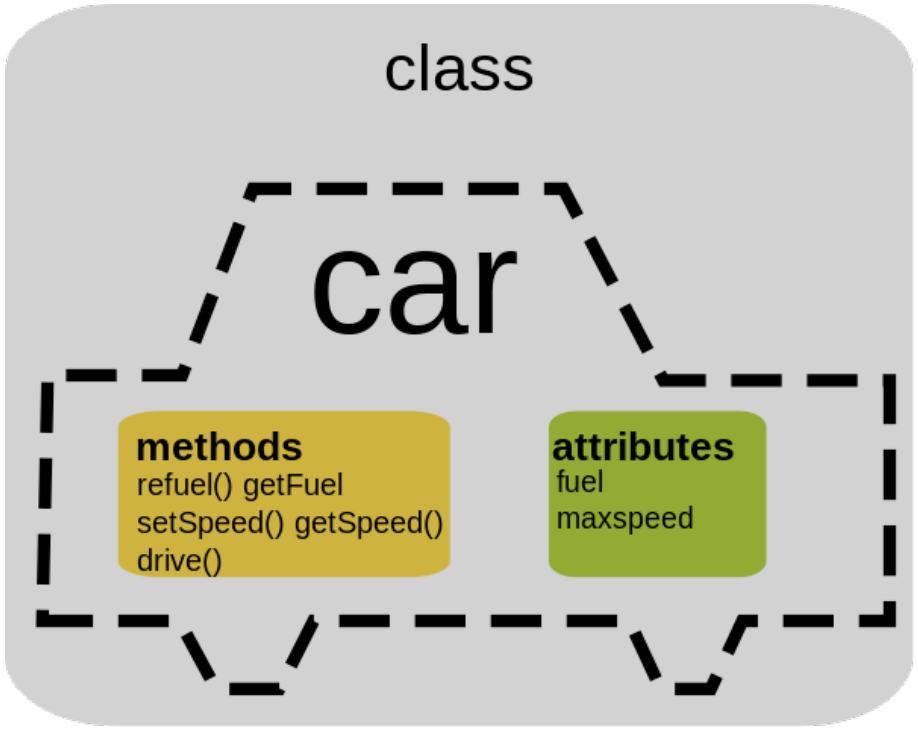


Figure 5-19 Discretization grid for 2-D circulation model.

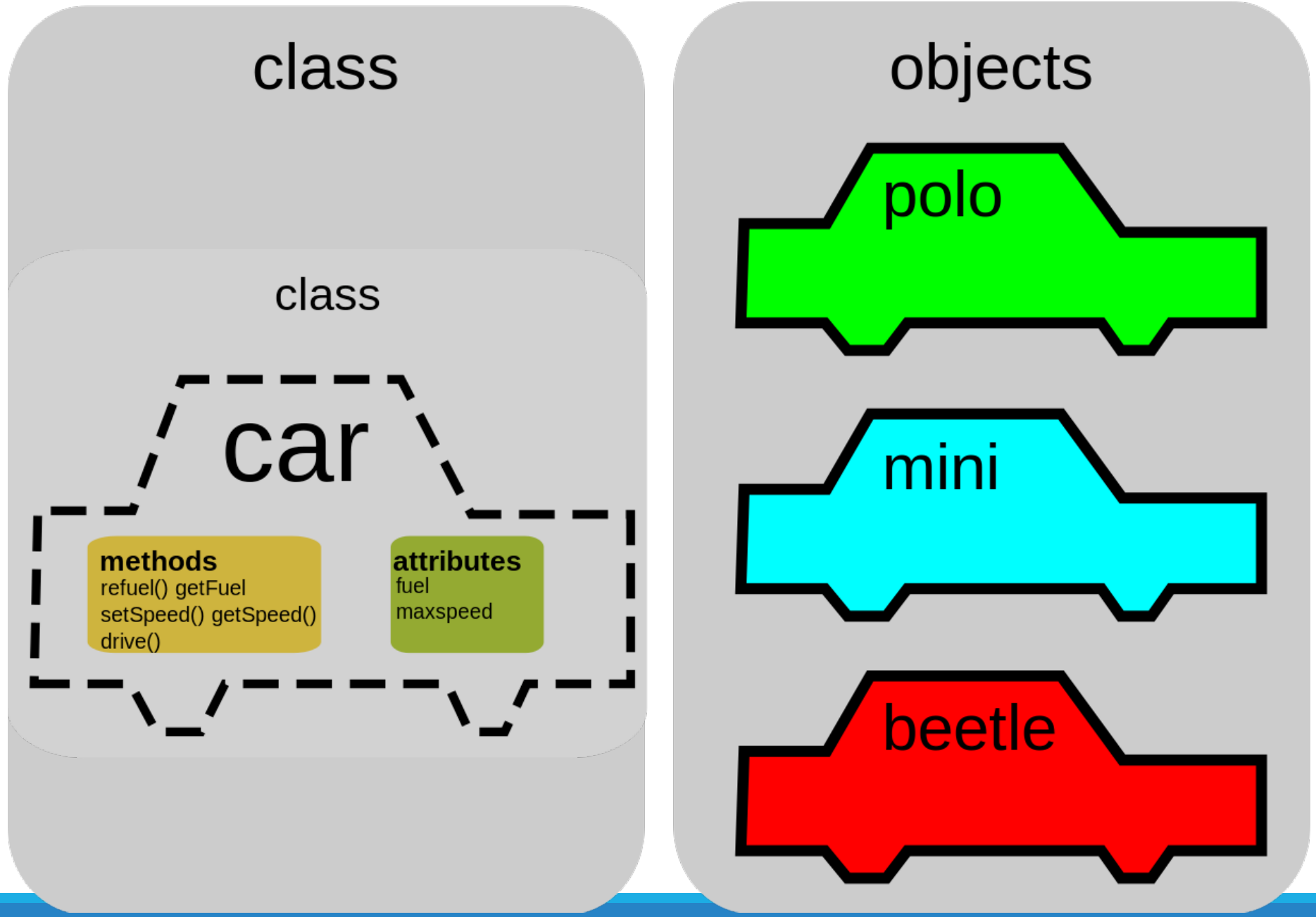
Slingerland, Harbaugh, and Furlong (1994)

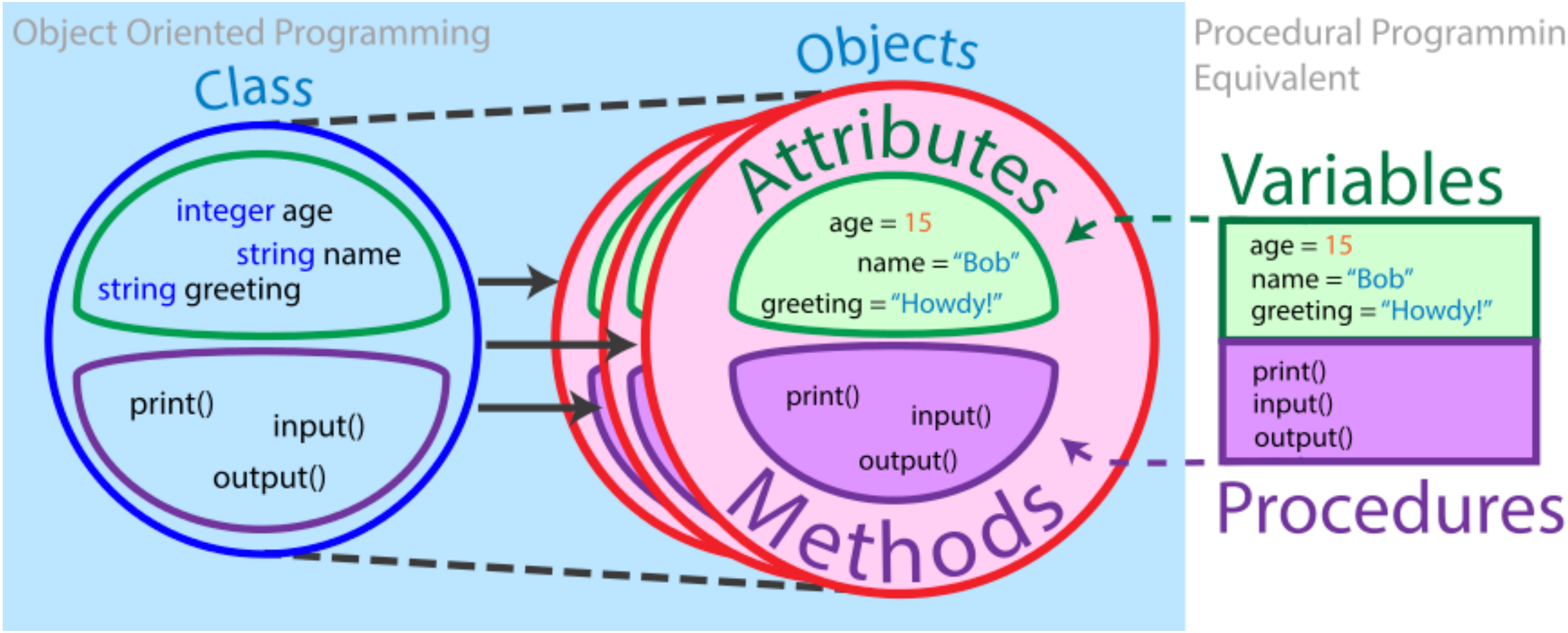
Quick side bar on Object Oriented Programing:

From Wikipedia: “**Object-oriented programming** (**OOP**) is a programming paradigm based on the concept of "objects", which may contain data, ... often known as *attributes;* and code, in the form of procedures, often known as *methods*. ” (https://en.wikipedia.org/wiki/Object-oriented_programming)



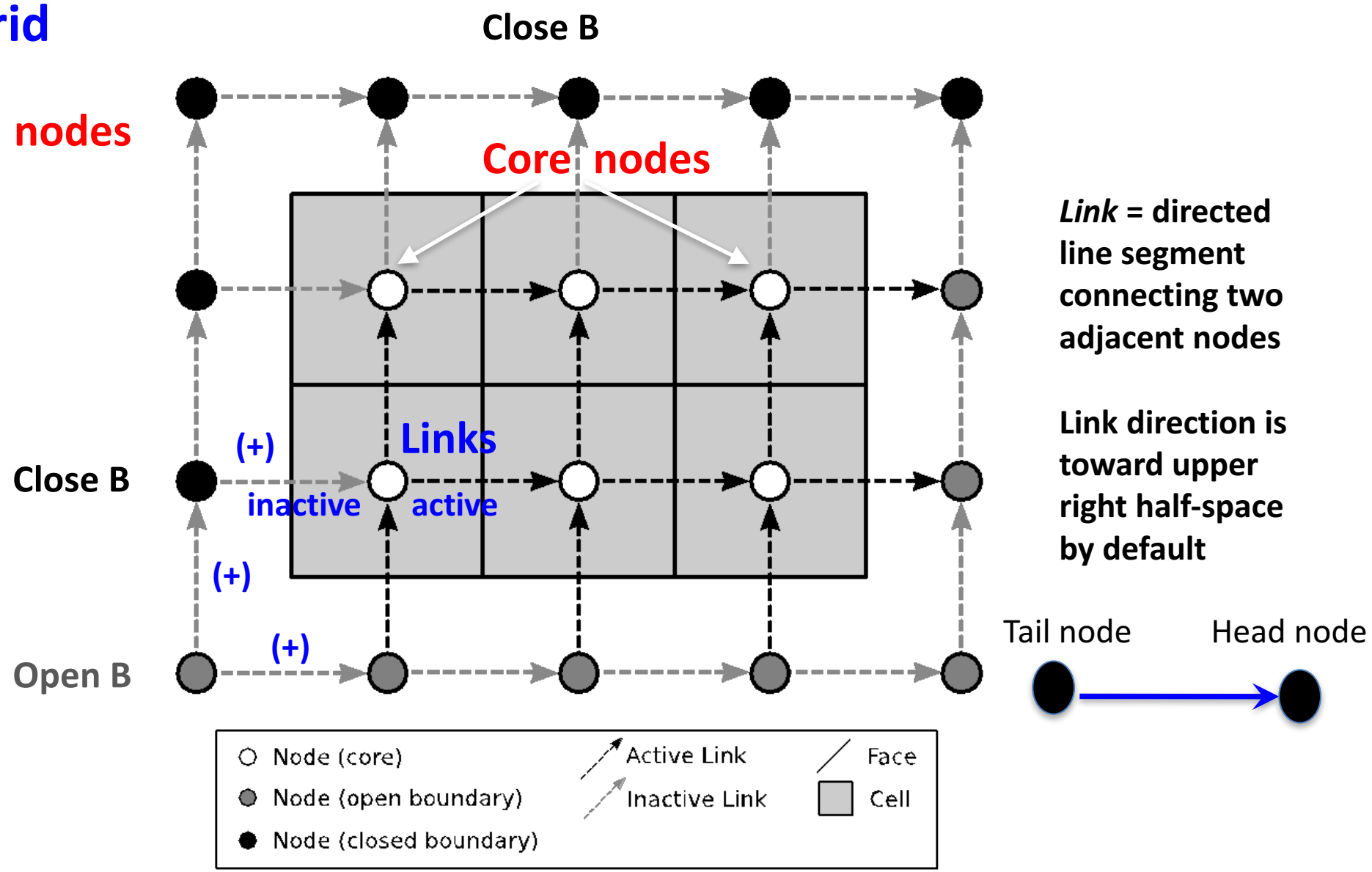
https://en.wikibooks.org/wiki/A-level_Computing/AQA/Paper_1/Fundamentals_of_programming/Elements_of_Object-Oriented_Programming


class
class
car
methods
refuel() getFuel
setSpeed() getSpeed()
drive()
attributes
fuel
maxspeed
objects
polo
mini
beetle



https://en.wikibooks.org/wiki/A-level_Computing/AQA/Paper_1/Fundamentals_of_programming/Elements_of_Object-Oriented_Programming

RasterModelGrid



Link **= directed line segment connecting two adjacent nodes**

Link direction is toward upper right half-space by default

Tail node Head node

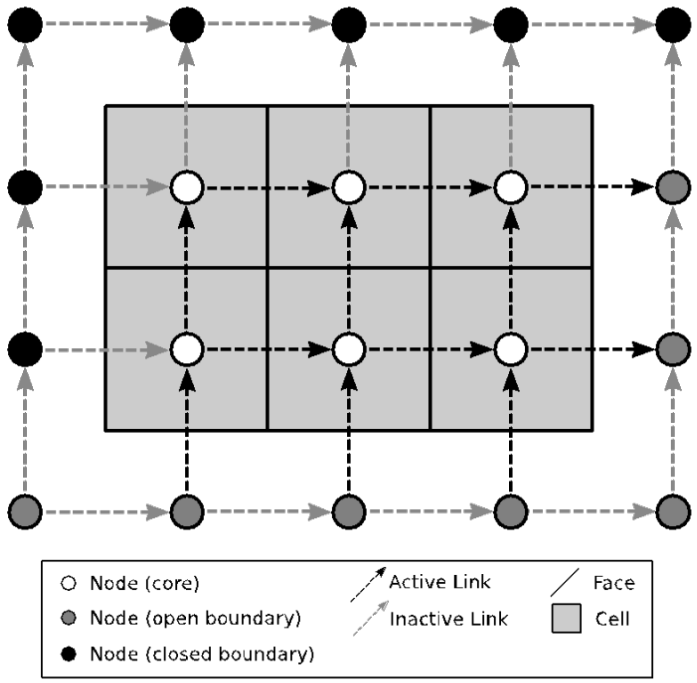
CORE_NODE: 0; FIXED_VALUE_BOUNDARY: 1 (e.g. outlet)
TRACKS_CELL_BOUNDARY: 3 (looped boundary); CLOSED_BOUNDARY: 4

Let’s start playing with Landlab.

- Open up Spyder.
- You can type commands in the command line.
- Or type them in a file that you can save, but copy them to run them on the command line.
- You will want to remember some of these commands for later.

RasterModelGrid example: Create a 4 by 5 grid

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




```
>> rg.number_of_nodes
Out: 20
>> rg.number_of_links
Out: 31
>> rg.number_of_node_rows
Out: 4
>> rg.number_of_node_columns
Out: 5
>> rg.number_of_core_nodes
Out: 6
>> rg.number_of_cells
Out: 6
```

```
In [1]: from landlab import RasterModelGrid

In [2]: RasterModelGrid?
Init signature: RasterModelGrid(*args, **kwds)
Docstring:
A 2D uniform rectilinear grid.

Create a uniform rectilinear grid that has *num_rows* and *num_cols*
of grid nodes, with a row and column spacing of *dx*.

Use the *bc* keyword to specify boundary_conditions along the edge nodes
of the grid. *bc* is a dict whose keys indicate edge location (as
"bottom", "left", "top", "right") and values must be one of "open", or
"closed". If an edge location key is missing, that edge is assumed to be
*open*.

Parameters
----------
shape : tuple of int
    Shape of the grid in nodes.
spacing : float, optional
    Row and column node spacing.
bc : dict, optional
    Edge boundary conditions.

Examples
--------
Create a uniform rectilinear grid that has 4 rows and 5 columns of nodes.
Nodes along the edges will be *open*. That is, links connecting these
nodes to core nodes are *active*.

>>> from landlab import RasterModelGrid
>>> rmg = RasterModelGrid((4, 5), 1.0)
>>> rmg.number_of_node_rows, rmg.number_of_node_columns
(4, 5)
```

```
In [3]: RasterModelGrid.n
```

```
RasterModelGrid.ndim
RasterModelGrid.neighbors_at_node
RasterModelGrid.new_field_location
RasterModelGrid.node_at_cell
RasterModelGrid.node_at_core_cell
RasterModelGrid.node_at_link_head
RasterModelGrid.node_at_link_tail
RasterModelGrid.node_axis_coordinates
RasterModelGrid.node_has_boundary_neighbor
RasterModelGrid.node_is_boundary
RasterModelGrid.node_is_core
RasterModelGrid.node_spacing
```

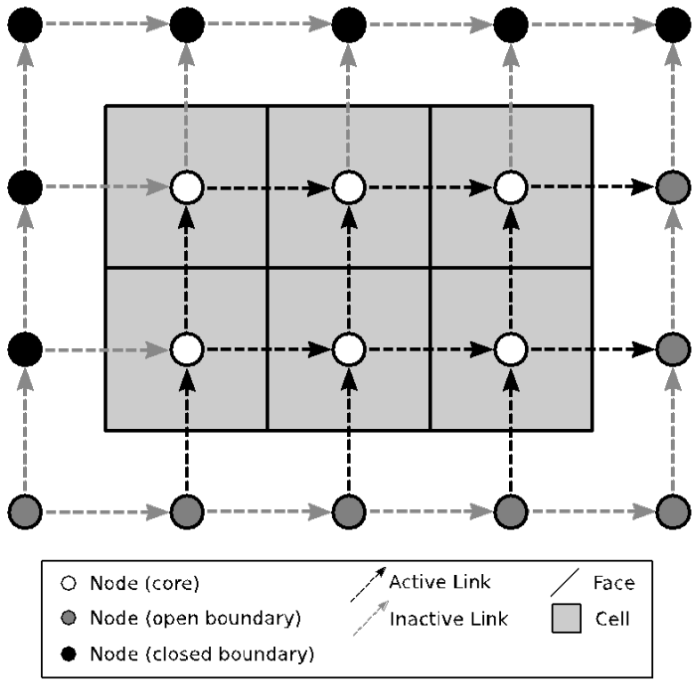
```
In [3]: RasterModelGrid.calc
```

```
RasterModelGrid.calc_aspect_at_cell_subtriangles
RasterModelGrid.calc_aspect_at_node
RasterModelGrid.calc_diff_at_link
RasterModelGrid.calc_distances_of_nodes_to_point
RasterModelGrid.calc_flux_div_at_cell
RasterModelGrid.calc_flux_div_at_node
RasterModelGrid.calc_grad_across_cell_corners
RasterModelGrid.calc_grad_across_cell_faces
RasterModelGrid.calc_grad_along_node_links
RasterModelGrid.calc_grad_at_active_link
RasterModelGrid.calc_grad_at_link
RasterModelGrid.calc_grad_at_patch
```

```
In [3]: RasterModelGrid.calculate_slope_aspect_at_nodes_best_fit_plane?
Signature: RasterModelGrid.calculate_slope_aspect_at_nodes_best_fit_plane(self, nodes,
val)
Docstring:
Calculate slope aspect.

.. note:: This method is deprecated as of Landlab version 1.0.

    Use :func:`calc_slope_at_node, calc_aspect_at_node` instead.



Slope aspect of best-fit plane at nodes.

.. codeauthor:: Katy Barnhart <katherine.barnhart@colorado.edu>

.. note::

    THIS CODE HAS ISSUES (SN 25-Sept-14): This code didn't perform
    well on a NS facing elevation profile. Please check
    slope_aspect_routines_comparison.py under landlab\examples before
    using this.  Suggested alternative:
    calculate_slope_aspect_at_nodes_burrough

Calculates both the slope and aspect at each node based on the
elevation of the node and its neighbors using a best fit plane
calculated using single value decomposition.

Parameters
----------
nodes : array-like
    ID of nodes at which to calculate the aspect
val : ndarray
    Elevation at all nodes
```

RasterModelGrid example: Create a 4 by 5 grid

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




```
>> rg.number_of_nodes
Out: 20
>> rg.number_of_links
Out: 31
>> rg.number_of_node_rows
Out: 4
>> rg.number_of_node_columns
Out: 5
>> rg.number_of_core_nodes
Out: 6
>> rg.number_of_cells
Out: 6
```

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`

Example: attaching data to the grid

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

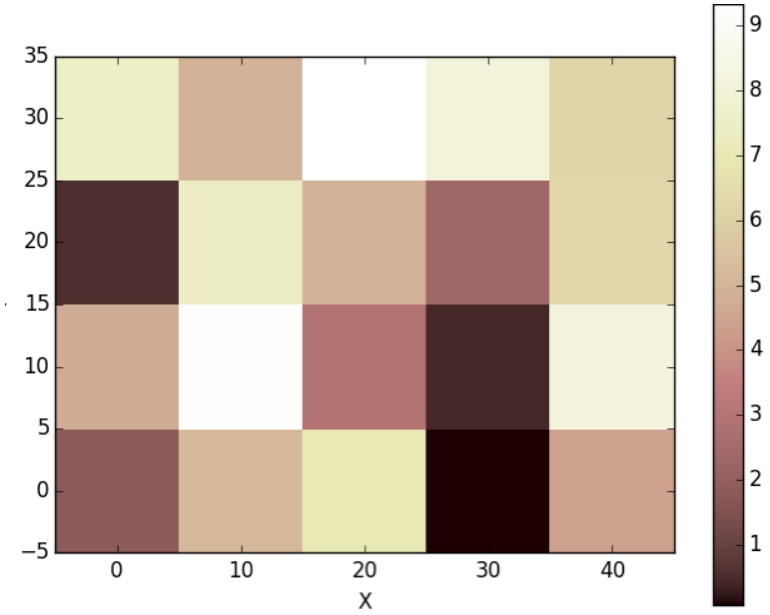
Create a random field and “attach” it to the grid as ‘elevation’

```
>> import numpy as np
>> z=10*np.random.rand(rg.number_of_nodes)
>> rg.add_field('node','topographic__elevation', z)
```

Plot the elevation field

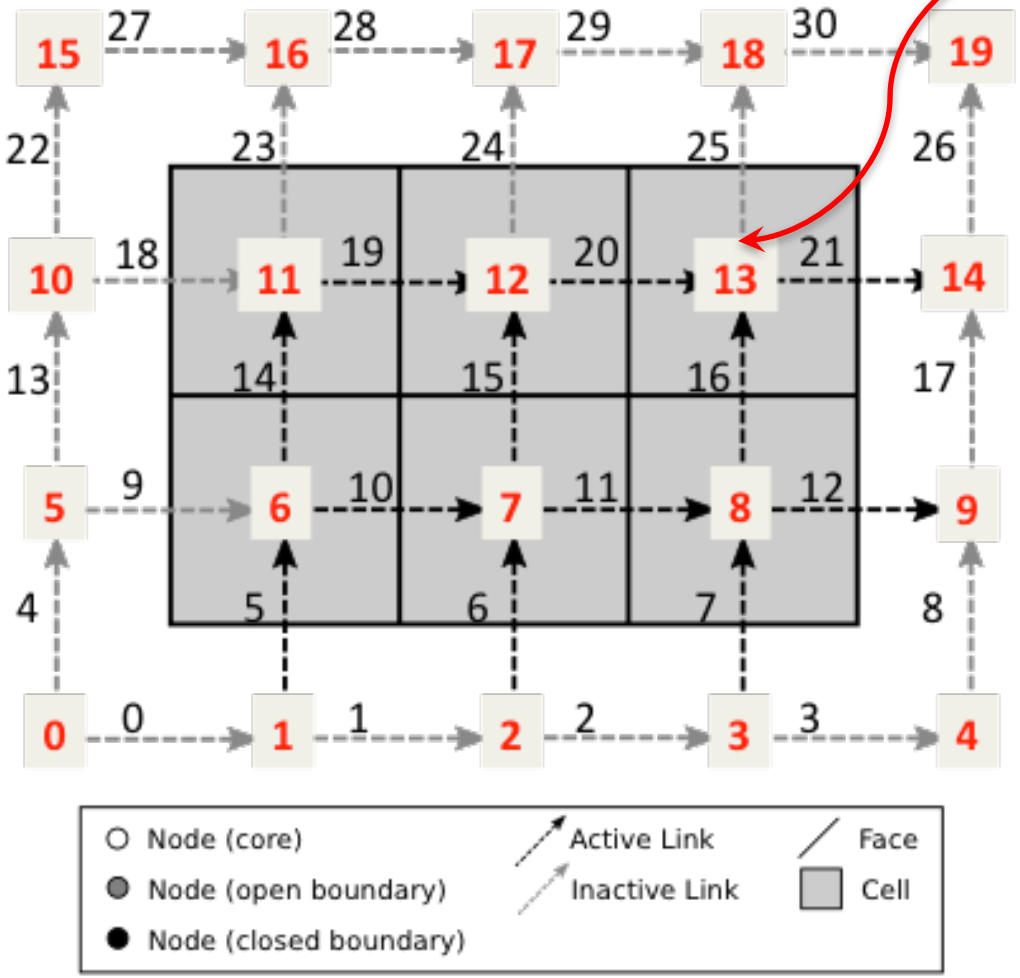
```
>> from landlab.plot import imshow_grid
>> import matplotlib.pyplot as plt
>> imshow_grid(rg, 'topographic__elevation')
>> plt.show()
```

```
>> rg.at_node.keys()
```



Explore node and link structure and “attached” data

Starting from the Southwest corner nodes are numbered to the East in each row.
Links are numbered using the number of their tail nodes.




```
>> rg.status_at_node[13]
Out: 0
>> rg.links_at_node[13]
Out: array([21, 25, 20, 16])
>> rg.link_dirs_at_node[13]
Out: array([-1, -1,  1,  1])
>> rg.node_at_link_tail[20]
Out: 12
>> rg.node_at_link_head[20]
Out: 13
```

```
>> rg.at_node['topographic__elevation'][13]
Out: 2.36
```

Now set ‘elevation’ to 100

```
>> rg.at_node[' 'topographic__elevation'][13]=100
```

OR: z[13]=100

```
>> rg.at_node[' 'topographic__elevation '][13]
Out: 100
```

CORE_NODE: 0; FIXED_VALUE_BOUNDARY: 1 (e.g. outlet)
TRACKS_CELL_BOUNDARY: 3 (looped boundary); CLOSED_BOUNDARY: 4

LANDLAB

What Landlab provides

2. **Components and coupling framework for components**
   - A ***component*** models a **single process** (e.g., lithosphere flexure, linear diffusion, flow routing across terrain)
   - Components have a **standard interface** and can be combined by writing a **short Python script** (model, driver)
   - **Initialized** using input grid and parameters
   - **Update** relevant **'fields'** on the grid
   - Components also include analytical tools for analyzing landscapes (e.g. channel steepness, hillslope length, …)
   - Most components have a method "***run_one_step***" carries out the process.

What Landlab provides

Table 5. Components available in Landlab v.1.0.

Component name	Process simulated/analysis performed	Key reference
LinearDiffuser	Linear diffusion of topography	Culling (1963)
PerronNLDiffuse	Nonlinear hillslope diffusion	Perron (2011)
Flexure	Simple lithospheric flexure under loading	Lambeck (1988), Hutton and Syvitski (2008)
gFlex	A more complex flexure model, utilizing gFlex	Wickert (2016)
FlowRouter	A convergent flow router, following the Fastscape algorithms	Braun and Willett (2013)
DepressionFinderAndRouter	A lake filler that can route flow across depressions	Tucker et al. (2001a)
SinkFiller	An algorithm to fill depressions in a surface	Tucker et al. (2001b)
OverlandFlow	A shallow overland flow approximation	de Almeida et al. (2012), Adams et al. (2016)
KinematicWaveRengers	A solution to the depth varying Manning equation for surface flow	Julien et al. (1995), Rengers et al. (2016)
SoilInfiltrationGreenAmpt	Infiltrate surface water into a soil following the Green–Ampt method	Julien et al. (1995), Rengers et al. (2016)
SoilMoisture	Compute local inter-storm water balance and root-zone soil moisture saturation fraction	Laio et al. (2001)
PotentialEvapotranspiration	Calculate potential evapotranspiration across a surface	ASCE-EWRI (2005), Zhou et al. (2013)
Radiation	Calculate total incident shortwave solar radiation	Bras (1990)
Vegetation	Calculate above-ground live and dead biomass, and leaf area index	Istanbulluoglu et al. (2012), Zhou et al. (2013)
VegCA	Cellular automata algorithm to simulate spatial organization of PFTs	Zhou et al. (2013)
PrecipitationDistribution	Generate a storm sequence of intervals and intensities	Eagleson (1978)
FireGenerator	Produces intervals between fire events, following a Weibull distribution	Polakow and Dunne (1999)
StreamPowerEroder	Implements fluvial erosion according to stream power, using the Fastscape algorithms	Braun and Willett (2013)
FastscapeEroder	An alternative implementation of the Fastscape stream power algorithms	Braun and Willett (2013)
DetachmentLtdErosion	An implementation of stream power erosion *not* based on Fastscape	Howard (1994)
SedDepEroder	Sediment-flux-dependent shear stress based fluvial incision	Hobley et al. (2011)
SteepnessFinder	Calculates steepness indices for a channel network	Wobus et al. (2006)
ChiFinder	Calculates the chi index along a channel network	Perron and Royden (2012)

(Hobley et al., 2017, ESURF)

- Information about corners
- Data Fields in ModelGrid
  - Create Field Arrays
  - Add Fields to a ModelGrid
  - Query Fields
- Gradients, fluxes, and divergences on the grid
- Mappers
- Boundary condition control
- Identifying node subsets
- Surface analysis
- Notes
- Examples
  - Create Field Arrays
  - Add Field Arrays
- Mapping data between different grid elements
  - Grid mapping functions
- Gradient calculators
  - Gradient calculation functions
- Divergence calculation functions
- Grid creation from a formatted input file
- Function/method decorators
  - Grid decorators

Specialized methods and properties for Rectilinear Grids 'raster grids'

Landlab's rectilinear grids are implemented by the class *RasterModelGrid*, which inherits from *ModelGrid* and

properties for radial grids
- Layers
- Components
  - Hillslope geomorphology
  - Fluvial geomorphology
  - Flow routing
  - Shallow water hydrodynamics
  - Land surface hydrology
  - Landslides
  - Vegetation
  - Precipitation
  - Weathering
  - Terrain Analysis
  - Tectonics
  - Fire
  - Initial conditions: random field generators
  - The Component base class
- Input/Output (IO)
- Plotting and Visualization
- Utilities and Decorators
- Cellular Automata (CA)
- Contributing to Landlab
- References

Next topic

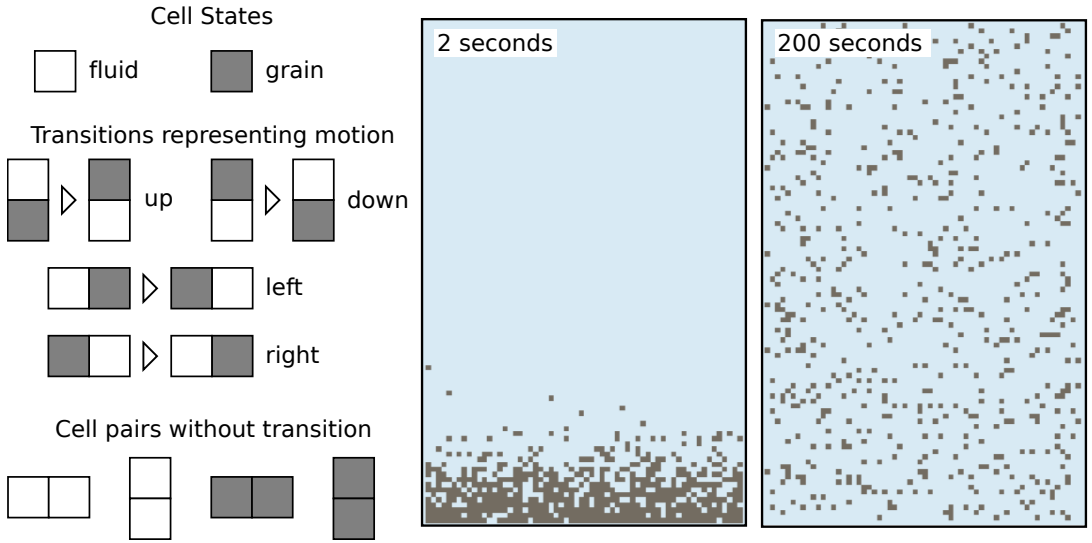
General class methods and

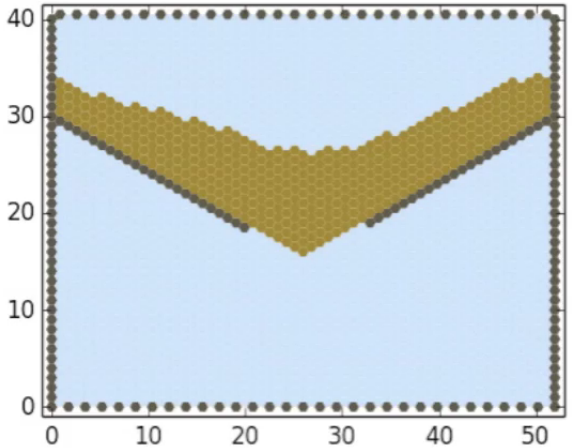
What Landlab provides

3. Input, output, visualization
    - **Read model parameters** from a formatted text file (optional)
    - **Read in** digital terrain data (e.g., **DEM**s) ➔ 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”

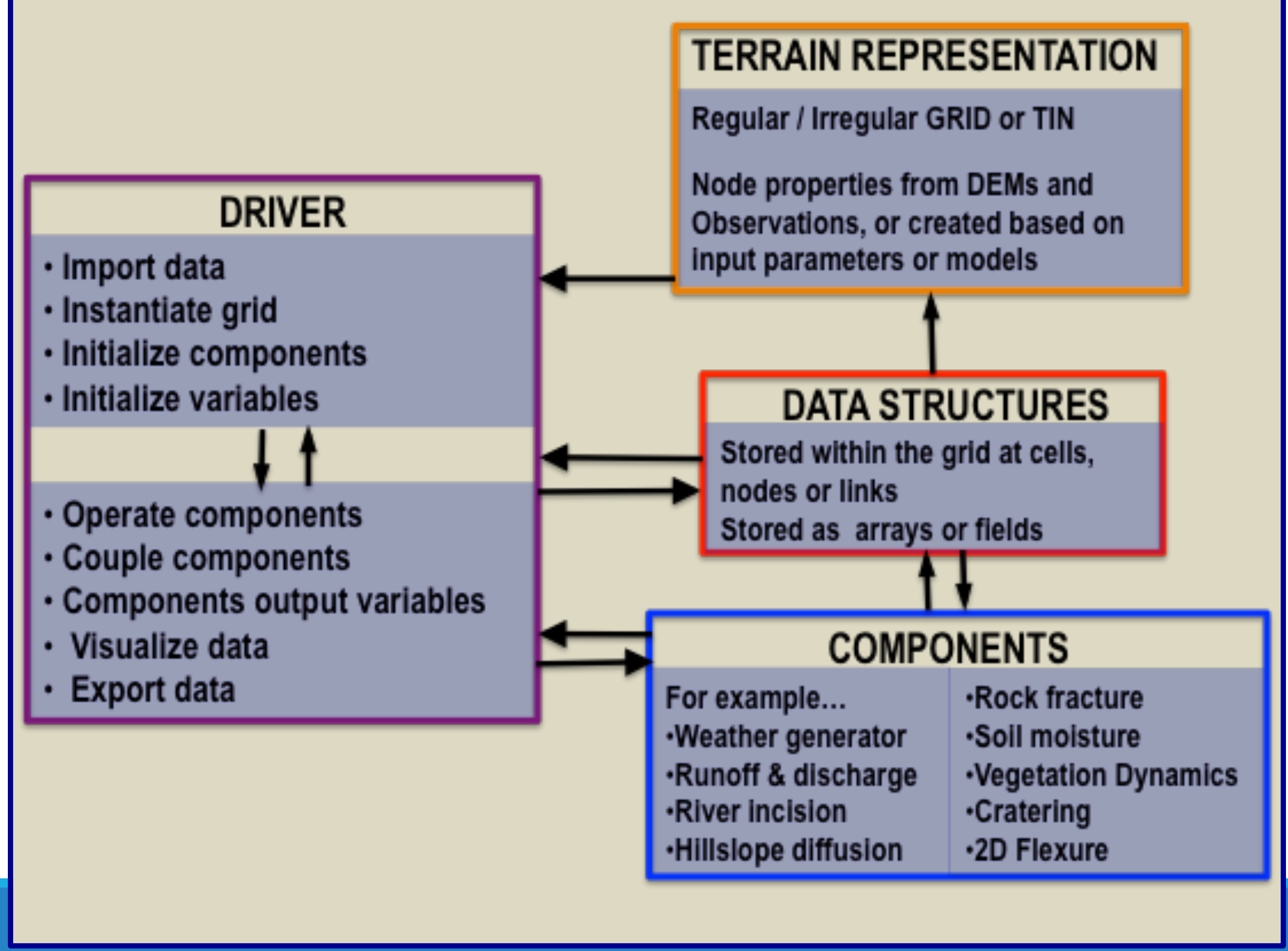




(Tucker et al., 2016 Geoscientific Model Development)

How do you create a Landlab model?

- Your **unique** creation.
- ***Usually*** a model will:
  - Operate on a **grid**;
  - **Create data**;
  - Use components to **update data values through time**;
  - **Visualize data**.
- but it **does not have to do *all* of these things**.

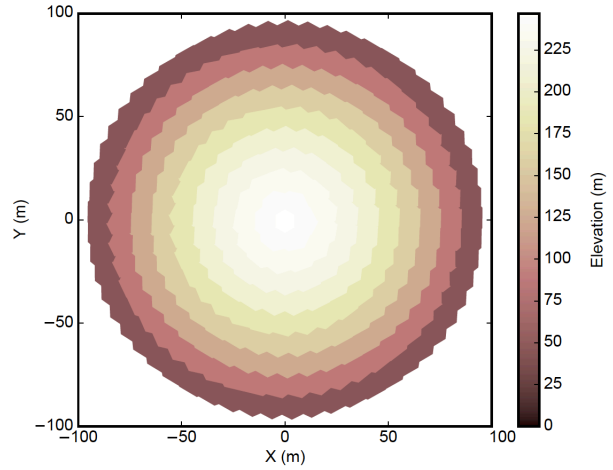

TERRAIN REPRESENTATION
Regular / Irregular GRID or TIN
Node properties from DEMs and Observations, or created based on input parameters or models
DRIVER
• Import data
• Instantiate grid
• Initialize components
• Initialize variables
• Operate components
• Couple components
• Components output variables
• Visualize data
• Export data
DATA STRUCTURES
Stored within the grid at cells, nodes or links
Stored as arrays or fields
COMPONENTS
For example…
•Weather generator
•Runoff & discharge
•River incision
•Hillslope diffusion
•Rock fracture
•Soil moisture
•Vegetation Dynamics
•Cratering
•2D Flexure

LANDLAB

Example single component diffusion model

Code to implement a simple diffusion model on a radial Landlab grid, using Landlab components:

```
>>> from landlab import RadialModelGrid, imshow_grid
>>> from landlab.components import LinearDiffuser
>>> from matplotlib.pyplot import show
>>> mg = RadialModelGrid(num_shells=10, dr=10.)
>>> z = mg.add_zeros('node', 'topographic__elevation')
>>> dt = 2000.  # no longer the stable timestep
>>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
>>> for i in range(500):
...     z[mg.core_nodes] += 0.001*dt
...     ld.run_one_step(dt)
>>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
>>> show()
```



(Hobley et al., 2017, ESURF)

Linear Diffusion Equation

$$q_s = -K_d \nabla z$$

$$q_s = -K_d \frac{dz}{dx}$$

$$\frac{dz}{dt} = U - \frac{dq_s}{dx}$$

$z =$ elevation

$t =$ time

$U =$ rock uplift

$q_s =$ sediment transport rate

$x =$ distance downslope

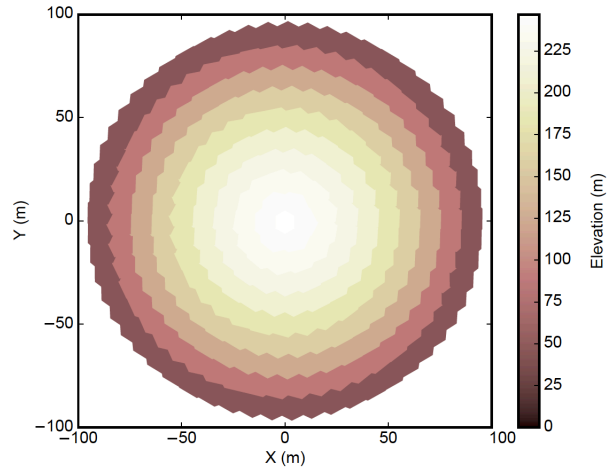
$K_d =$ linear diffusivity

Single component diffusion model

Import Statements

Code to implement a simple diffusion model on a radial Landlab grid, using Landlab components:

```
>>> from landlab import RadialModelGrid, imshow_grid
>>> from landlab.components import LinearDiffuser
>>> from matplotlib.pyplot import show
>>> mg = RadialModelGrid(num_shells=10, dr=10.)
>>> z = mg.add_zeros('node', 'topographic__elevation')
>>> dt = 2000.  # no longer the stable timestep
>>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
>>> for i in range(500):
...     z[mg.core_nodes] += 0.001*dt
...     ld.run_one_step(dt)
>>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
>>> show()
```



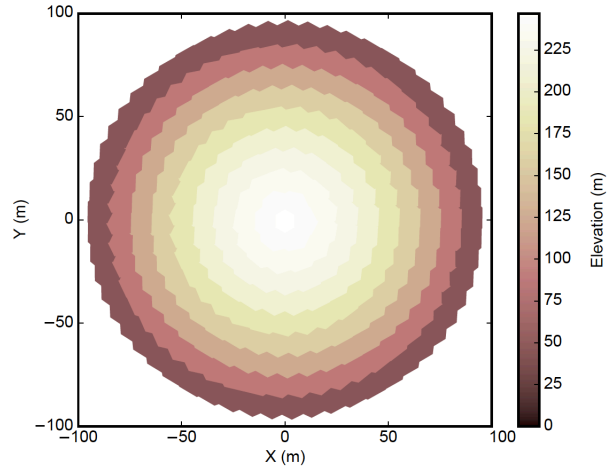
(Hobley et al., 2017, ESURF)

Single component diffusion model

Code to implement a simple diffusion model on a radial Landlab grid, using Landlab components:

Instantiate model grid; Add data/create field

```
>>> from landlab import RadialModelGrid, imshow_grid
>>> from landlab.components import LinearDiffuser
>>> mg = RadialModelGrid(num_shells=10, dr=10.)
>>> z = mg.add_zeros('node', 'topographic__elevation')
>>> dt = 2000.  # no longer the stable timestep
>>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
>>> for i in range(500):
...     z[mg.core_nodes] += 0.001*dt
...     ld.run_one_step(dt)
>>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
>>> show()
```



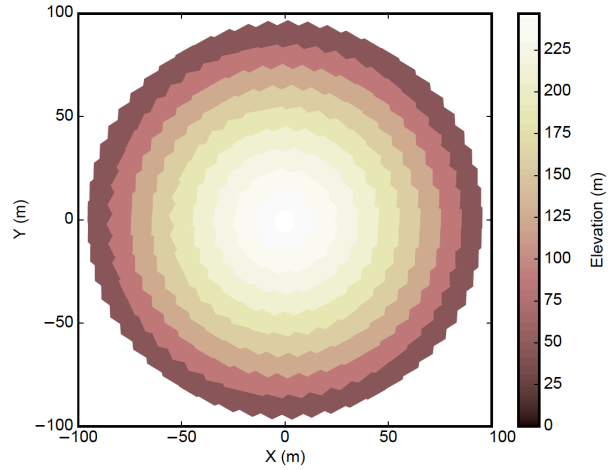
(Hobley et al., 2017, ESURF)

Single component diffusion model

Code to implement a simple diffusion model on a radial Landlab grid, using Landlab components:

Initialize parameter; Instantiate diffusion object

```
>>> from landlab import RadialModelGrid, imshow_grid
>>> from landlab.components import LinearDiffuser
>>> from matplotlib.pyplot import show
>>> mg = RadialModelGrid(num_shells=10, dr=10.)
>>> dt = 2000.  # no longer the stable timestep
>>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
>>> for i in range(500):
...     z[mg.core_nodes] += 0.001*dt
...     ld.run_one_step(dt)
>>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
>>> show()
```



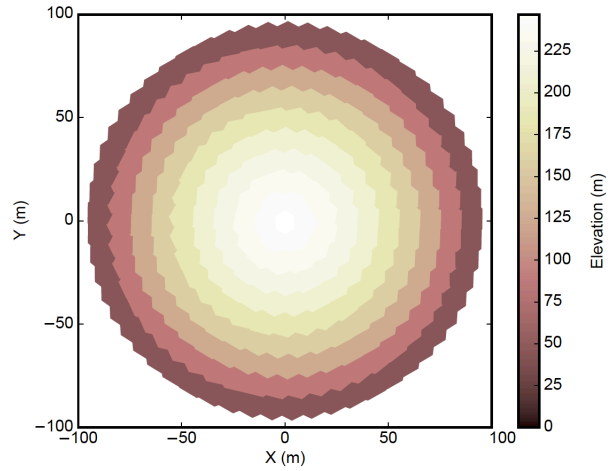
(Hobley et al., 2017, ESURF)

Single component diffusion model

Code to implement a simple diffusion model on a radial Landlab grid, using Landlab components:

```
1. >>> from landlab import RadialModelGrid, imshow_grid
2. >>> from landlab.components import LinearDiffuser
3. >>> from matplotlib.pyplot import show
4. >>> mg = RadialModelGrid(num_shells=10, dr=10.)
5. >>> z = mg.add_zeros('node', 'topographic__elevation')
6. >>> dt = 2000.  # no longer the stable timestep
8. >>> for i in range(500):
9. ...     z[mg.core_nodes] += 0.001*dt
10. ...     ld.run_one_step(dt)
                                                    Elevation (m)')
12. >>> show()
```

Time loop; Do the calculations!



(Hobley et al., 2017, ESURF)

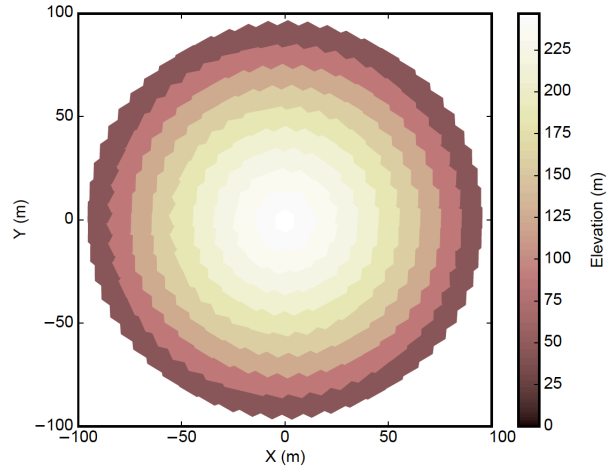
Single component diffusion model

Code to implement a simple diffusion model on a radial Landlab grid, using Landlab components:

```
>>> from landlab import RadialModelGrid, imshow_grid
>>> from landlab.components import LinearDiffuser
>>> from matplotlib.pyplot import show
>>> mg = RadialModelGrid(num_shells=10, dr=10.)
>>> z = mg.add_zeros('node', 'topographic__elevation')
>>> dt = 2000.  # no longer the stable timestep
>>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
>>> for i in range(500):
...     z[mg.core_nodes] += 0.001*dt
```

Visualize data

```
>>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
>>> show()
```



(Hobley et al., 2017, ESURF)

Every model is some variant of the previous example

- Can be on the **command line** (as last example).
- Can be a **Python (.py) file** that is run from the command line.
- Can be an **iPython notebook (.ipynb)**, which is great for teaching and learning.

Make a model!

1. Make a linear diffusion model on a 50 by 50 raster model grid with dx of 20.0 m, dt = 1000 yr, linear diffusivity of 0.1 m^2/yr, and rock uplift rate of 5e-4 m/yr. Run until landscape is no longer changing, i.e. steady state.
    - I suggest doing this in a .py file, in the spyder console. You can access the command line if you need help.
2. Try changing the boundary conditions so that the East and West edges are closed.
3. Repeat step 1 on a Hex grid.
4. Change the size of your grid.
5. Change something about your plots, like the color bar.

- Hints – There are tutorials in the tutorials repo on boundary conditions and grids. There is a reference manual. You already worked on a jupyter notebook of a linear diffusion model. Talk to your neighbors. Talk to me.

Congratulations!!!

- You have coded with Landlab!!!
- So far you have a linear diffusion model:

$$\frac{dz}{dt} = U - \frac{dq_s}{dx}$$

$$q_s = -K_d \frac{dz}{dx}$$

$z =$ elevation
$t =$ time
$U =$ rock uplift
$q_s =$ sediment transport rate
$x =$ distance downslope
$K_d =$ linear diffusivity

Congratulations!!!

- You have coded with Landlab!!!
- So far you have a linear diffusion model:

$$\frac{dz}{dt} = U - \frac{dq_s}{dx}$$

$$q_s = -K_d \frac{dz}{dx}$$

$z =$ elevation
$t =$ time
$U =$ rock uplift
$q_s =$ sediment transport rate
$x =$ distance downslope
$K_d =$ linear diffusivity

Congratulations!!!

- You have coded with Landlab!!!
- So far you have a linear diffusion model:

$$\frac{dz}{dt} = U - \frac{dq_s}{dx}$$

$$q_s = -K_d \frac{dz}{dx}$$

$z =$ elevation
$t =$ time
$U =$ rock uplift
$q_s =$ sediment transport rate
$x =$ distance downslope
$K_d =$ linear diffusivity

- Now add to channels to your model.

$$\frac{dz}{dt} = U - \frac{dq_s}{dx} - KA^mS^n$$

$K =$ fluvial erodibility
$A =$ drainage area (proxy for fluvial discharge)
$S = -\frac{dz}{dx} =$ slope
$m, n =$ positive exponents

Add channels to your model.

1. First you need to route flow.
2. Then you need to model fluvial incision.
3. Use a raster grid, 100 by 100, dx = 100., rock uplift rate = 0.001 m/yr, time step = 1000 years, linear diffusivity = 0.1 m^2/yr, erodibility = 1e-5 yr^{-1}, m=0.5, n=1.
4. When successful, try adding some random noise to the initial elevation.
5. You can also play with pit-filling.

Some Hints:

```
In [20]: from landlab.components import FlowAccumulator

In [21]: from landlab.components import FlowRouter

In [22]: from landlab.components import SinkFiller

In [23]: from landlab.components import FastscapeEroder
```

landlab.readthedocs.io/en/latest/#flow-routing

Flow routing

- The Landlab FlowDirectors: Components for Flow Direction
- FlowAccumulator: Component to do FlowAccumulation with the FlowDirectors
- FlowRouter: Calculate flow direction and accumulation from topography
- DepressionFinderAndRouter: Handle depressions in terrain by calculating extent and drainage of “lakes”
- PotentialityFlowRouter: Find flow directions and accumulation using potential-field theory
- SinkFiller: Permanently fill pits in a topography

LANDLAB

Congratulations! You have made a LEM!

- How about some “GIS” on your landscape?
- Plot channel profiles, slope-area data, channel steepness and chi.
- If you don’t know about slope-area data, stay tuned for a derivation on the board!!!
- If you don’t know about chi and channel steepness, don’t worry about it.

Some hints:

```
In [25]: from landlab.plot import channel_profile as prf
```

```
In [27]: from landlab.components import ChiFinder, SteepnessFinder
```

```
# find the location of the largest channels, set initially to find 3 chans
profile_IDs = prf.channel_nodes(mg1, mg1.at_node['topographic__steepest_slope'],
                                mg1.at_node['drainage_area'],
                                mg1.at_node['flow__receiver_node'],
                                number_of_channels=3)

# find the upstream distances in these channels
dists_upstr = prf.get_distances_upstream(
    mg1, len(mg1.at_node['topographic__steepest_slope']),
    profile_IDs, mg1.at_node['flow__link_to_receiver_node'])

# channel profiles
plt.figure(1)
plt.plot(dists_upstr[0], z1[profile_IDs[0]], 'b-', label='channel 1')
plt.plot(dists_upstr[1], z1[profile_IDs[1]], 'k-', label='channel 2')
plt.plot(dists_upstr[2], z1[profile_IDs[2]], 'r-', label='channel 3')
plt.xlabel('distance upstream (m)')
plt.ylabel('elevation (m)')
plt.legend(loc='upper left')
title_text = '$K_{sp}$='+str(K_sp) + '; $time$='+str(total_time) + 'yr; $dx$='+str(dxy) + 'm'
plt.title(title_text)

# slope-area data in just the profiled channels
plt.figure(2)
plt.loglog(mg1.at_node['drainage_area'][profile_IDs[0]],
           mg1.at_node['topographic__steepest_slope'][profile_IDs[0]], 'b.',
           label='channel 1')
plt.loglog(mg1.at_node['drainage_area'][profile_IDs[1]],
           mg1.at_node['topographic__steepest_slope'][profile_IDs[1]], 'k.',
           label='channel 2')
plt.loglog(mg1.at_node['drainage_area'][profile_IDs[2]],
           mg1.at_node['topographic__steepest_slope'][profile_IDs[2]], 'r.',
           label='channel 3')
plt.legend(loc='lower left')
plt.xlabel('drainage area (m^2)')
plt.ylabel('channel slope [m/m]')
title_text = '$K_{sp}$='+str(K_sp) + '; $time$='+str(total_time) + 'yr; $dx$='+str(dxy) + 'm'
plt.title(title_text)
```

Follow your own interests:

Some Ideas:

1. Make erosion maps and “sediment flux”
2. Ecohydrology notebook in the tutorials repo
3. Overland flow notebook in the tutorials repo or the teaching notebook
4. How to make a component notebook in the tutorials repo
5. ????

THANK YOU!

- The Landlab team wants to work with you. Let us know if you have successes or failures. Feel free to email me (ngaspari@tulane.edu) and use the GitHub Issues page if you have problems.
- Please fill out this anonymous survey: https://goo.gl/forms/JiBiEhXUAJXBDRu02
  - (I emailed it to you.)