

Modeling Earth Surface Dynamics with Landlab

Nicole Gasparini (She/Her), Tulane University

Huge thanks to Katy Barnhart, CU Boulder and Mark Piper, CSDMS



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 (#)

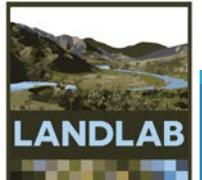
University of Washington : Erkan Istanbulluoglu, Sai Nudurupati(*), and Christina Bandaragoda

Cardiff : Dan Hobley(#)



Goals for this class:

- You will have a better idea of what Landlab is and what it can do.
- You will know where to look to get help if you want to do more with Landlab.
- You will play with some Landlab coupled models.



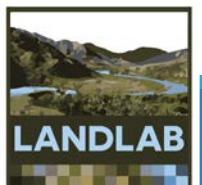
Assumptions I make in every class I teach:

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

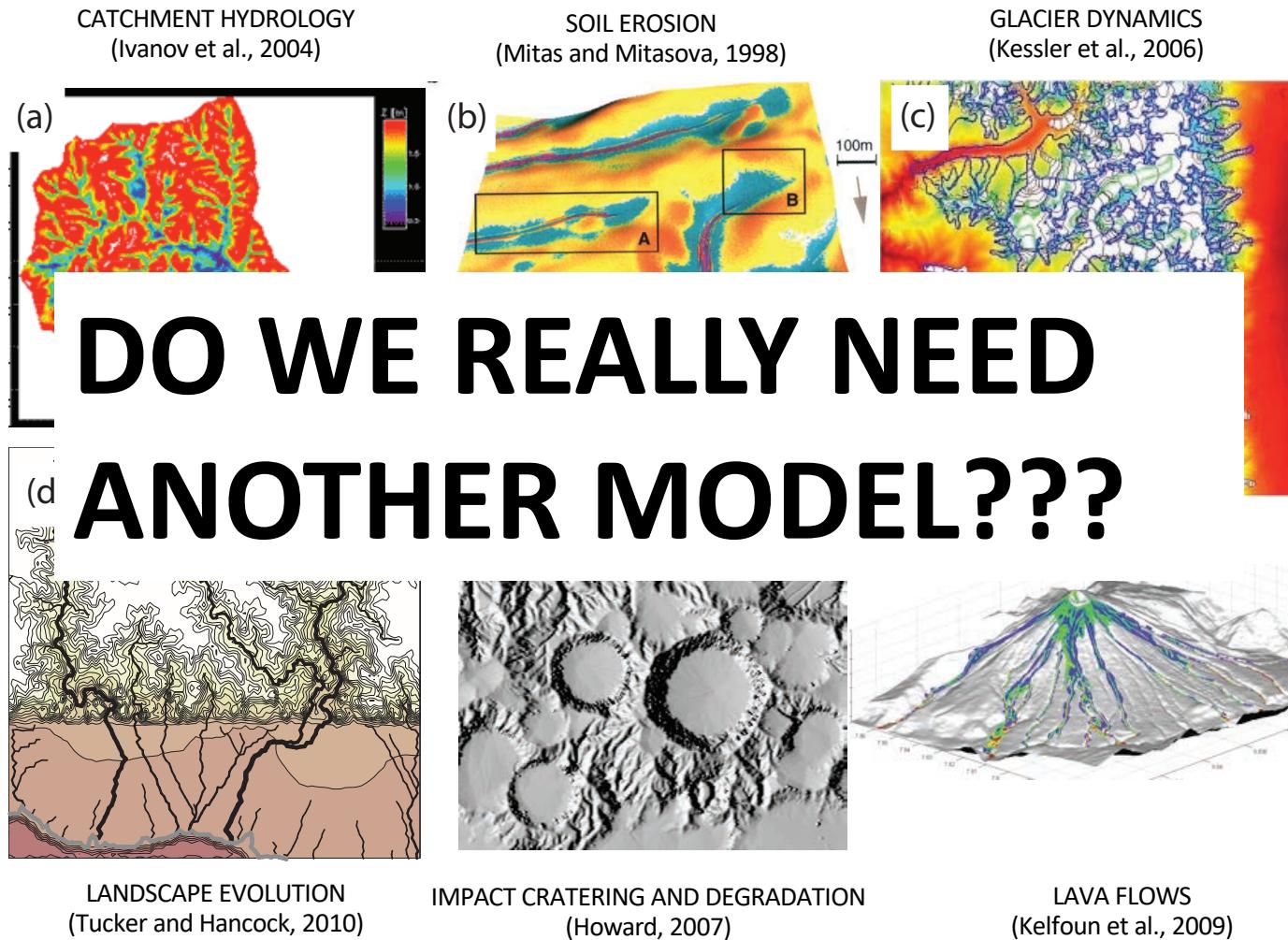


Overview for clinic:

- Introduction to Landlab, <http://landlab.github.io/#/>
 - Other important websites coming.
- Running Landlab on CSDMS JupyterHub
- All materials will be available at
<https://github.com/landlab/General-Landlab-Clinic-2020-21>



Lots of modeling approaches out there...



Challenges with many academic models:

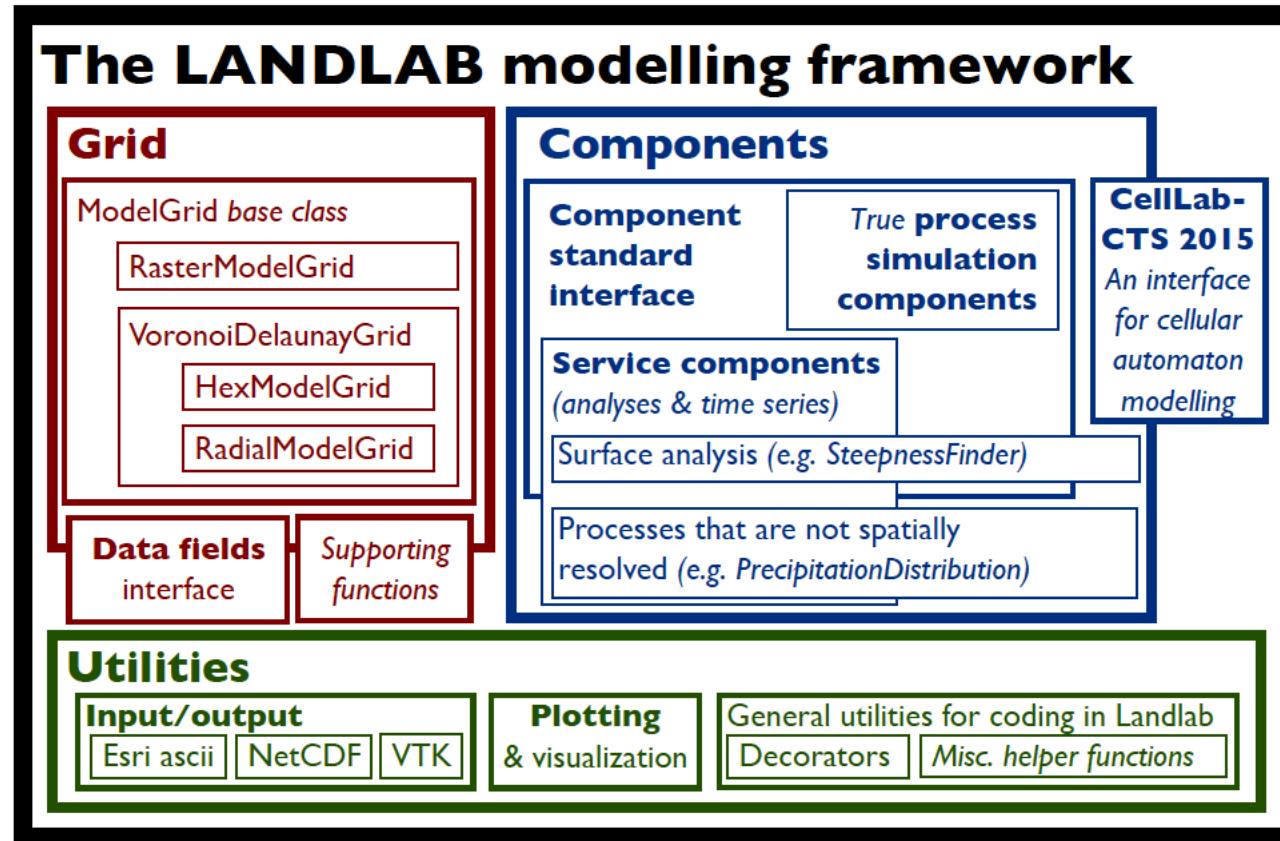
- built to answer a specific research question and not easily adaptable.
- poorly documented.
- not tested.
- platform dependent.
- not open source.

What is Landlab?

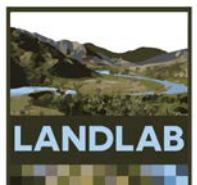
- A toolkit for building models of processes that happen on the earth surface – e.g. overland flow, bedrock river incision, growth and death of vegetation.
- Not a landscape evolution model (LEM), but you can build LEMs with Landlab if you want to.
- Open source.
- **A Python library.**
- Anyone can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



What's in the Landlab library?



(Hobley et al., 2017, ESURF)

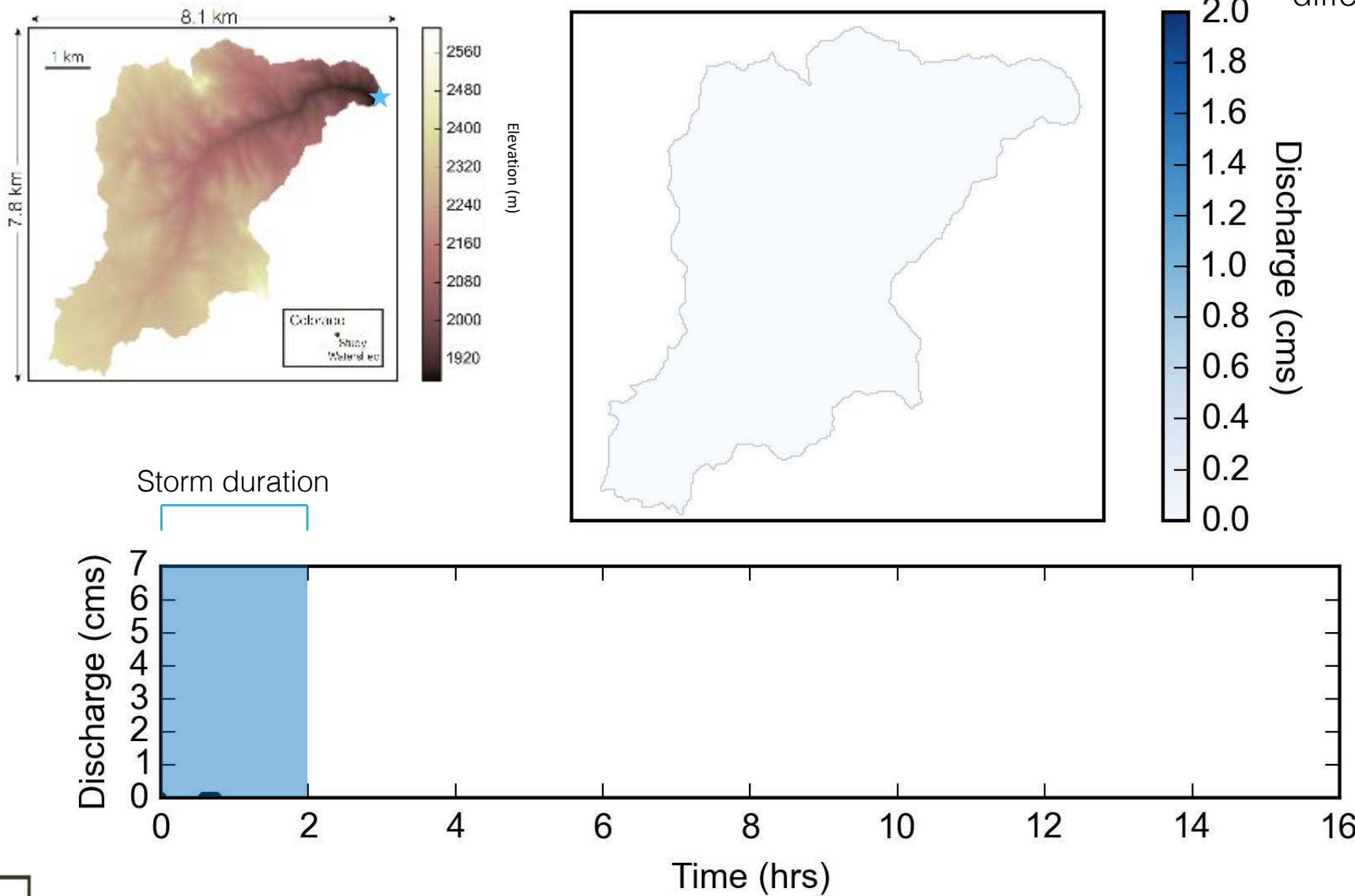


Landlab is also:

- Extensively documented.
- Has supporting materials such:
 - 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**.



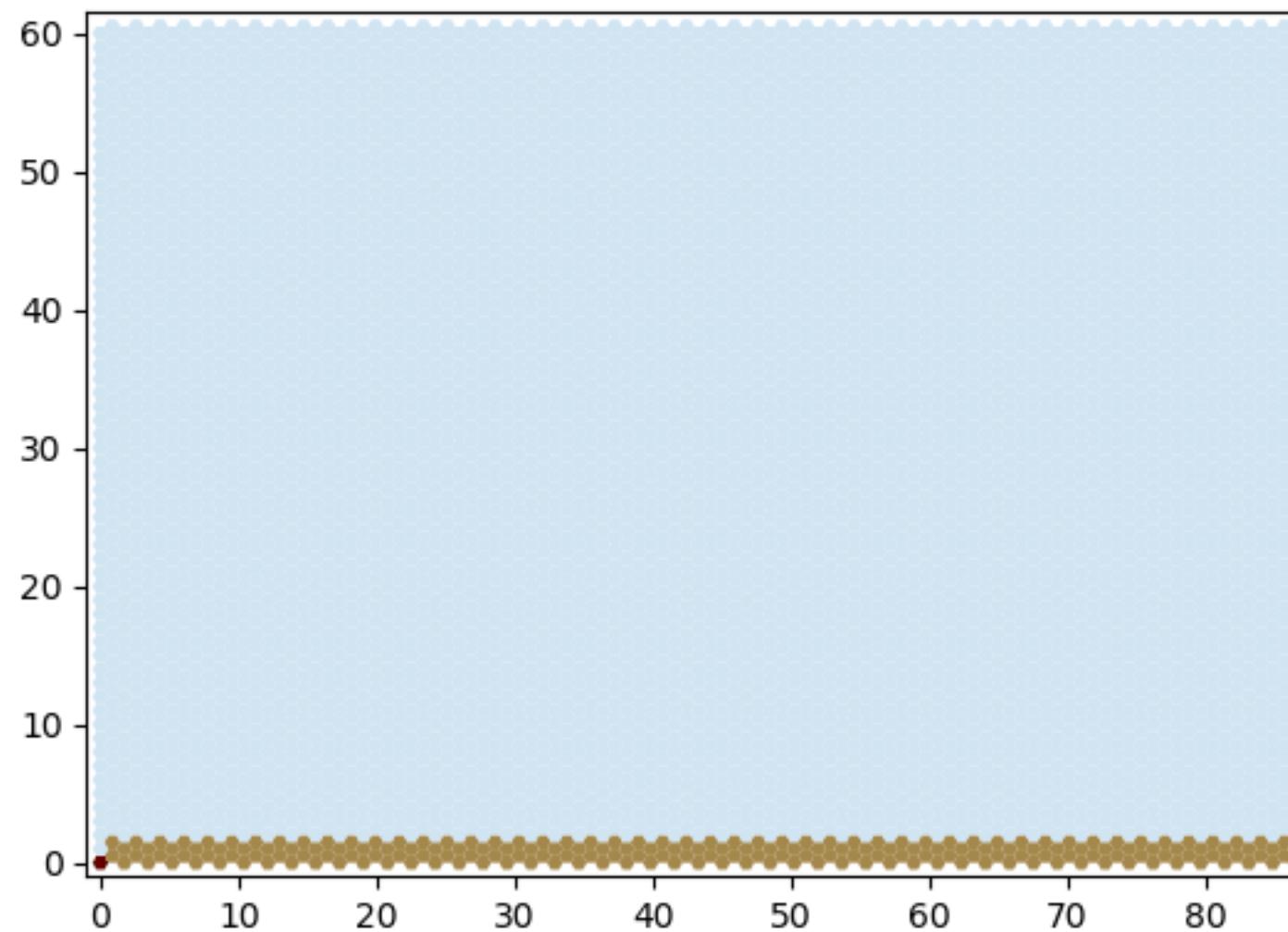
Application in a real world setting: Spring Creek, CO.



*Note scale differences

Source:
Jordan Adams,
See Adams et al.,
2017

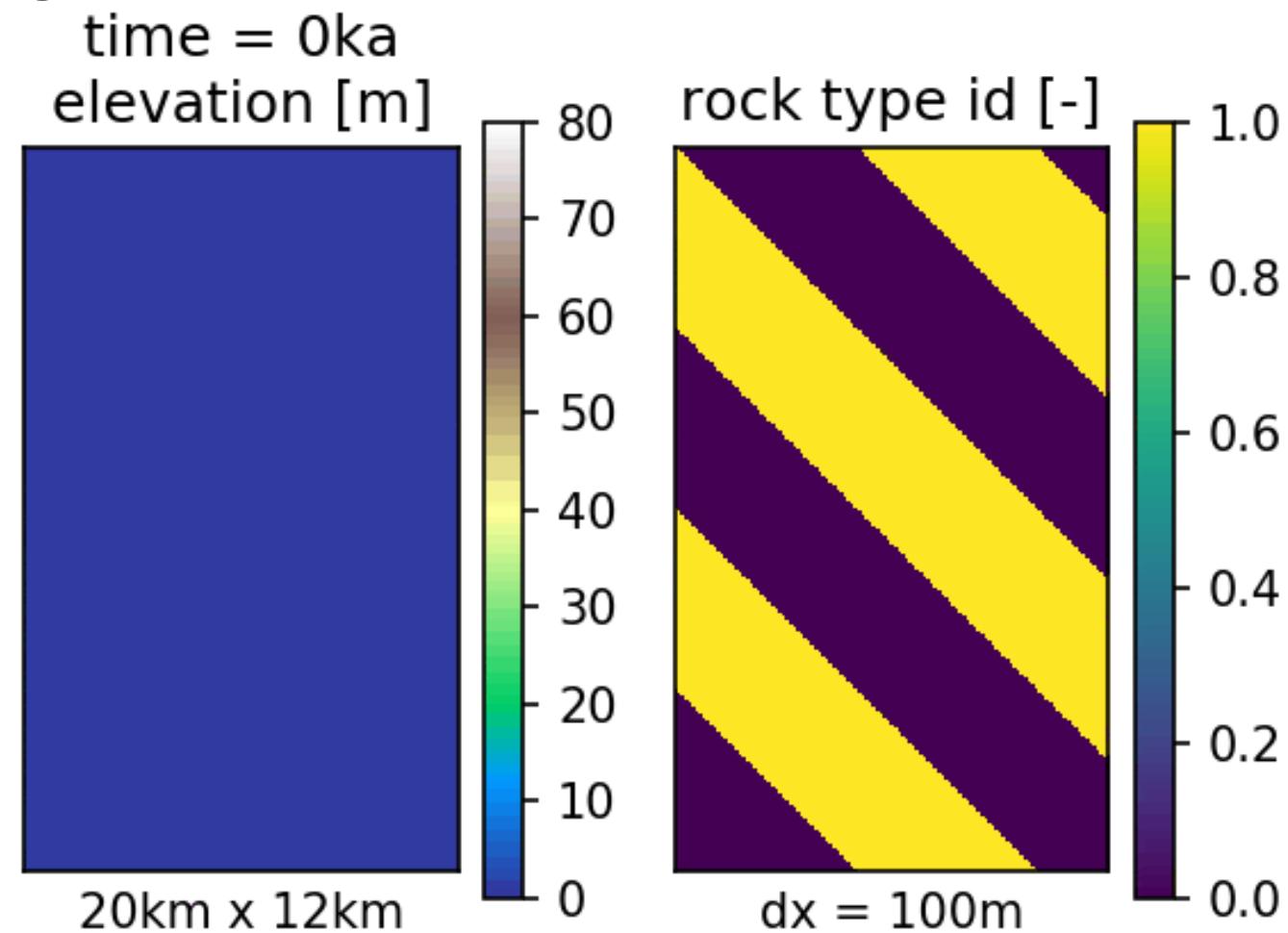
Hogback produced using CellLab, Tucker et al., 2018



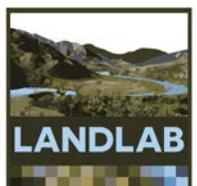
Lithology and LithoLayers

Two new Landlab components for spatially variable rock type.

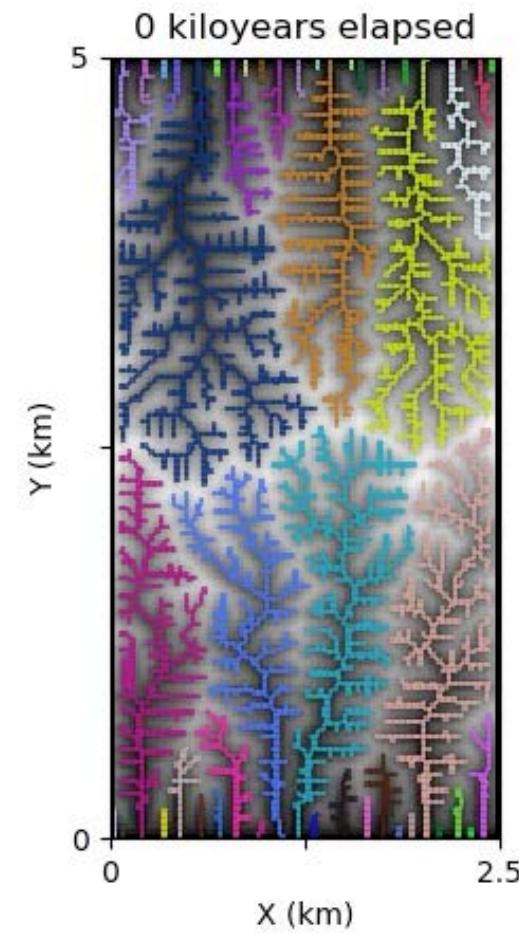
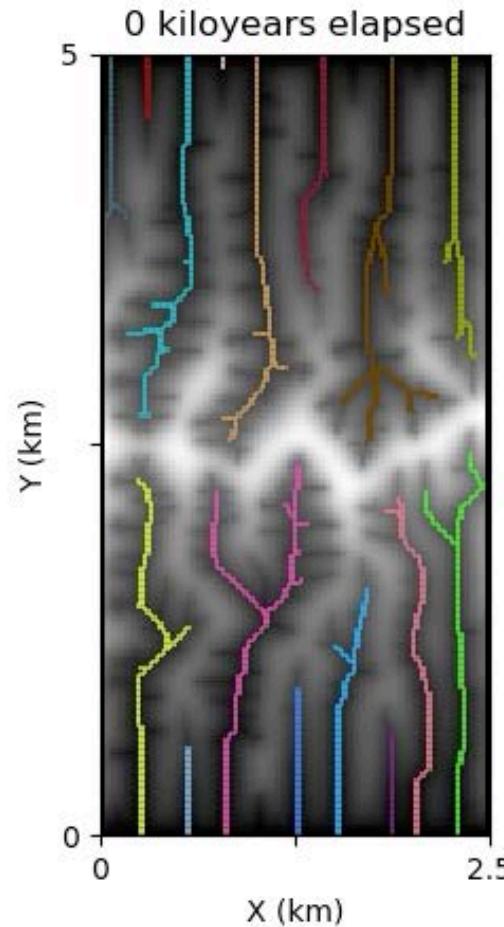
Barnhart et al., (2018). Lithology: A Landlab submodule for spatially variable rock properties. Journal of Open Source Software, 3(30), 979, <https://doi.org/10.21105/joss.00979>



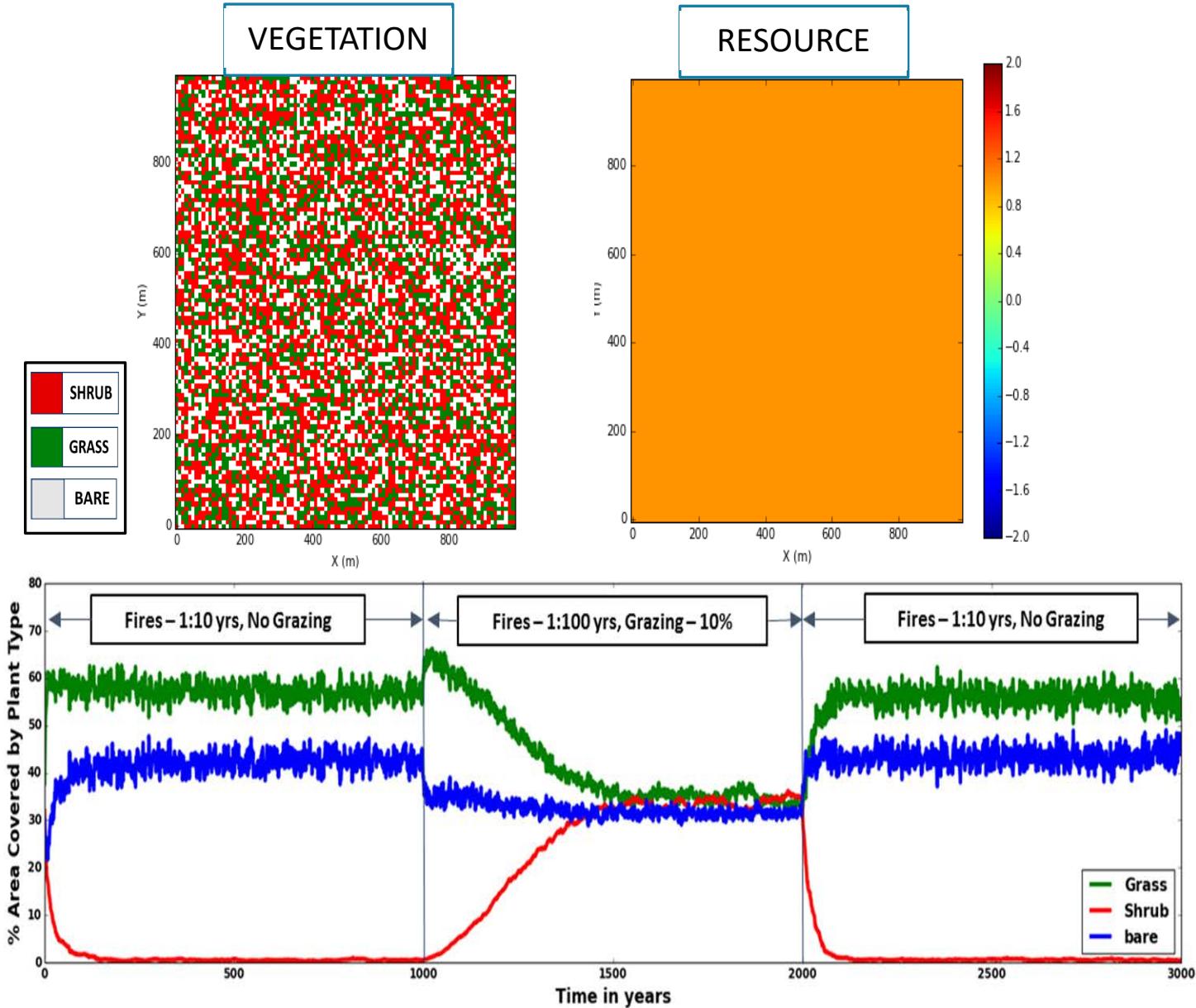
Animation shows soft (purple) and hard (yellow) rocks dipping to the north east being uplifted and eroded by a simple landscape evolution model. Right panel shows rock type exposed at the surface.



Drainage rearrangement, sensitivity to drainage density.
See Nathan Lyons et al., in review at ESURF.



Climate Change Experiments #1





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. Adams⁴, Sai Siddhartha Nudurupati⁵, Eric W. H. Hutton⁶, Nicole M. Gasparini⁴, Erkan Istambuloglu⁵, and Gregory E. Tucker^{1,2}

<https://doi.org/10.5194/esurf-2020-12>
Preprint. Discussion started: 4 March 2020
© Author(s) 2020. CC BY 4.0 License.



Short communication: Landlab v2.0: A software package for Earth surface dynamics

Katherine R. Barnhart^{1,2}, Eric W. H. Hutton^{3,4}, Gregory E. Tucker^{1,2,3}, Nicole M. Gasparini⁵, Erkan Istambuloglu⁶, Daniel E. J. Hobley⁷, Nathan J. Lyons⁵, Margaux Mouchene⁸, Sai Siddhartha Nudurupati⁶, Jordan M. Adams⁹, and Christina Bandaragoda⁶

¹University of Colorado at Boulder, Cooperative Institute for Research in Environmental Sciences, Boulder, Colorado

²University of Colorado at Boulder, Department of Geological Sciences, Boulder, Colorado

³University of Colorado at Boulder, Community Surface Dynamics Modeling System Integration Facility, Boulder, Colorado

⁴University of Colorado at Boulder, Institute for Arctic and Alpine Research, Boulder, Colorado

⁵Tulane University, Department of Earth and Environmental Sciences, New Orleans, Louisiana

⁶University of Washington, Department of Civil and Environmental Engineering, Seattle, Washington

⁷Cardiff University, School of Earth and Ocean Sciences, Cardiff, Wales, United Kingdom

⁸Univ. Grenoble Alpes, INRAE, ETNA, F-38402 St-Martin-d'Hères, France

⁹Delgado Community College, Division of Science and Math, New Orleans, Louisiana

Correspondence: Katherine Barnhart (katherine.barnhart@colorado.edu)



A hydroclimatological approach to predicting regional landslide probability using Landlab

Ronda Strauch¹, Erkan Istambuloglu¹, Sai Siddhartha Nudurupati¹, Christina Bandaragoda¹, Nicole M. Gasparini², and Gregory E. Tucker^{3,4}

Geosci. Model Dev., 9, 823–839, 2016
[www.geosci-model-dev.net/9/823/2016/](https://doi.org/10.5194/gmd-9-823-2016)
doi:10.5194/gmd-9-823-2016
© Author(s) 2016. CC Attribution 3.0 License.



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

Gregory E. Tucker^{1,2}, Daniel E. J. Hobley^{1,2}, Eric Hutton³, Nicole M. Gasparini⁴, Erkan Istambuloglu⁵, Jordan M. Adams⁴, and Sai Siddhartha Nudurupati⁵

Geosci. Model Dev., 10, 1645–1663, 2017
[www.geosci-model-dev.net/10/1645/2017/](https://doi.org/10.5194/gmd-10-1645-2017)
doi:10.5194/gmd-10-1645-2017
© Author(s) 2017. CC Attribution 3.0 License.



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

Jordan M. Adams¹, Nicole M. Gasparini¹, Daniel E. J. Hobley², Gregory E. Tucker^{3,4}, Eric W. H. Hutton⁵, Sai S. Nudurupati⁶, and Erkan Istambuloglu⁶

Geosci. Model Dev., 10, 4577–4604, 2017
<https://doi.org/10.5194/gmd-10-4577-2017>
© Author(s) 2017. This work is distributed under the Creative Commons Attribution 4.0 License.



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

Earth Surf. Dynam., 6, 563–582, 2018
<https://doi.org/10.5194/esurf-6-563-2018>
© Author(s) 2018. This work is distributed under the Creative Commons Attribution 4.0 License.



A lattice grain model of hillslope evolution

Gregory E. Tucker¹, Scott W. McCoy², and Daniel E. J. Hobley³

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 Lai¹  and Alison M. Anders¹ 

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

© Author(s) 2018. This work is distributed under the Creative Commons Attribution 4.0 License.



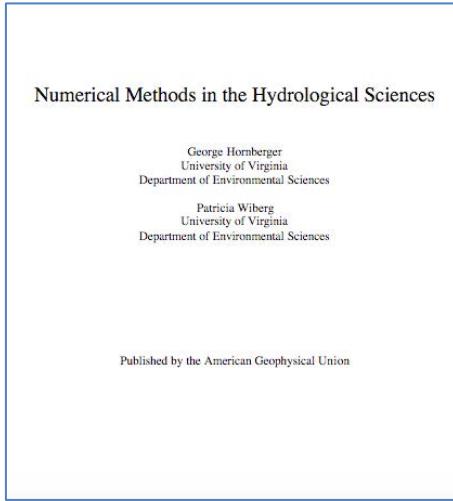
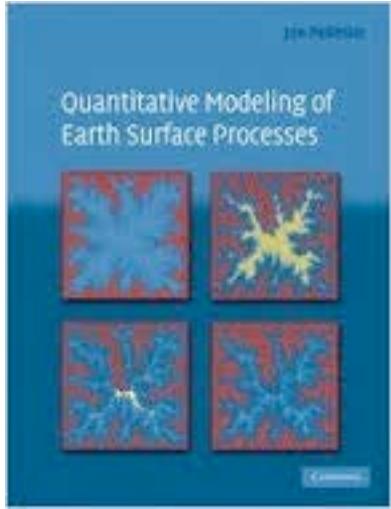
Research article

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

Manuel Schmid¹, Todd A. Ehlers¹ , Christian Werner¹ , Thomas Hickler^{2,3}, and Juan-Pablo Fuentes-Espoz⁴

Why build numerical models?

What are numerical models?

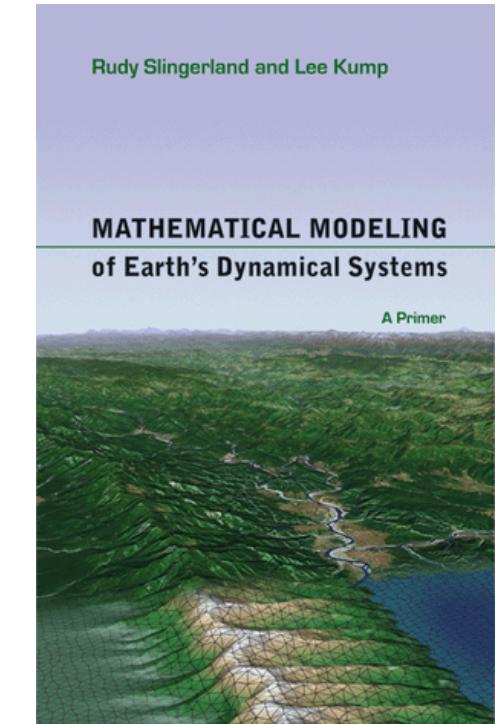


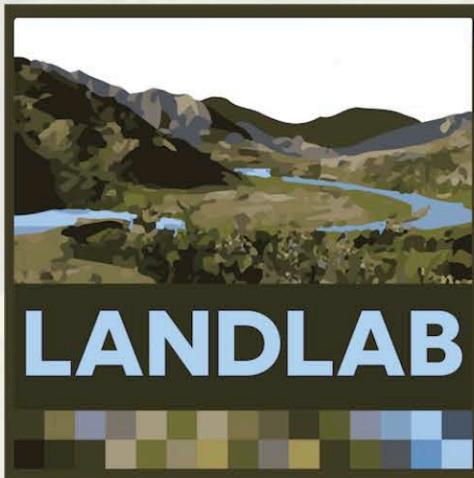
DYNAMICAL
MODEL

NUMERICAL
ALGORITHM

SOFTWARE

NATURE



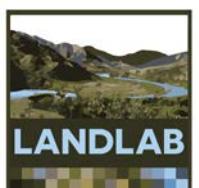


a python toolkit for modeling earth surface processes

[Install](#)[User Guide](#)[Tutorials](#)[Reference Manual](#)[Support](#)[Cite](#)

What is Landlab?

Landlab is a Python-based modeling environment that allows scientists and students to build numerical landscape models. Designed for disciplines that quantify earth surface dynamics such as geomorphology, hydrology, glaciology, and stratigraphy, it can also be used in related fields.



User Guide

The User Guide describes Landlab by topic area.

Users brand-new to Landlab should start with [10 minutes to Landlab](#).

Further information on any specific method can be obtained in the [API reference](#).

The Landlab project creates an environment in which scientists can build a numerical surface process model without having to code all of the individual components. Surface process models compute flows of mass, such as water, sediment, glacial ice, volcanic material, or landslide debris, across a gridded terrain surface. Surface process models have a number of commonalities, such as operating on a grid of points and routing material across the grid. Scientists who want to use a surface process model often build their own unique model from the ground up, re-coding the basic building blocks of their surface process model rather than taking advantage of codes that have already been written.

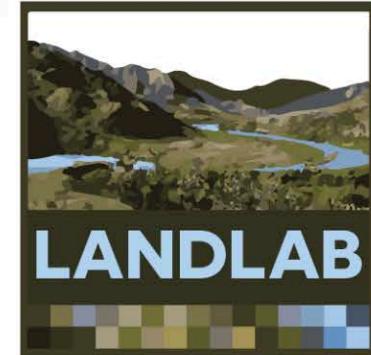
A list of papers and presentations using Landlab can be found [here](#).

Introduction to Python

- [Introduction to Python](#)
 - [Why Python?](#)
 - [Getting to know Python](#)
 - [NumPy, SciPy, and Efficient Coding Style](#)
 - [Cython](#)

The Landlab Grid

- [Introduction to Landlab's Gridding Library](#)
 - [How a Grid is Represented](#)
 - [Basic Grid Elements](#)
 - [Creating a Grid](#)
 - [Adding Data to a Landlab Grid Element using Fields](#)
 - [Representing Gradients in a Landlab Grid](#)



[INDEX](#)

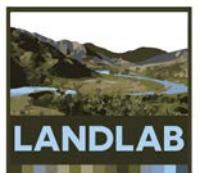
[Search](#)

[Table of Contents](#)

[User Guide](#)

- [Introduction to Python](#)
- [The Landlab Grid](#)
- [Model with Landlab and Components](#)
- [Landlab Tutorial Library](#)
- [Additional resources](#)
- [Presentations, Clinics, and Classroom Use](#)
- [Overland flow User Guide](#)
- [CellLab-CTS User Guide](#)
- [Major Version Transition Guides](#)

[Previous topic](#)





Landlab: A modular Earth Surface Dynamics modeling library

Landlab is an open-source Python-language package for numerical modeling of Earth surface dynamics. It contains:

- A gridding engine which represents the model domain. Regular and irregular grids are supported.
- A library of process components, each of which represents a physical process (e.g., generation of rain, erosion by flowing water). These components have a common interface and can be combined based on a user's needs.
- Utilities that support general numerical methods, file input/output, and visualization.

In addition Landlab contains a set of Jupyter notebook tutorials providing an introduction to core concepts and examples of use.

Landlab was designed for disciplines that quantify Earth surface dynamics such as geomorphology, hydrology, glaciology, and stratigraphy. It can also be used in related fields. Scientists who use this type of model often build their own unique model from the ground up, re-coding the basic building blocks of their landscape model rather than taking advantage of codes that have already been written. Landlab saves practitioners from the need for this kind of re-invention by providing standardized components that they can re-use.

Watch the webinar [Landlab Toolkit Overview](#) at CSDMS to learn more.

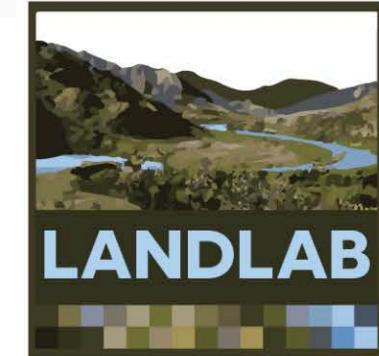
The most current source code is always available from our [git repository](#).

If you are interested in the state of current development, we compile [ongoing development](#). If you are interested in contributing but don't know how to get started (THANK YOU!), we compile [desired contributions](#) and have a page all about [development](#).

Many Jupyter notebooks exist describing use of Landlab. Find an overview [here](#). A subset of these notebooks are designed specifically for the classroom. Information about them and how to set them up for classroom use is described [on this page](#).

Landlab 2.0

In late December 2019 Landlab switched to version 2.0-beta. Landlab will be in 2.0-beta until the Landlab 2.0 publication is finalized.



[INDEX](#)

[Search](#)

[Table of Contents](#)

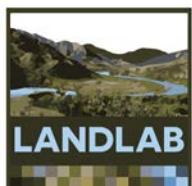
[Landlab: A modular Earth Surface Dynamics modeling library](#)

- [Landlab 2.0](#)
- [Supported Python Versions](#)
- [Documentation Outline](#)
 - [Acknowledgements](#)
- [Funding](#)
 - [Citing Landlab](#)
 - [Contact](#)

[Next topic](#)

[Installation Instructions](#)

[This Page](#)



Supported Python Versions

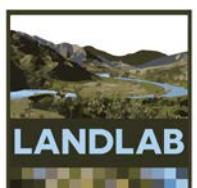
Python 3.6, 3.7, and 3.8

Documentation Outline

- Installation Instructions
 - Conda instructions
 - Pip Instructions
 - Conda Environment
 - Additional Resources
- User Guide
 - Introduction to Python
 - The Landlab Grid
 - Model with Landlab and Components
 - Landlab Tutorial Library
 - Additional resources
 - Presentations, Clinics, and Classroom Use
 - Overland flow User Guide
 - CellLab-CTS User Guide
 - Major Version Transition Guides
- API reference
 - Basic Model Interface
 - Cellular Automata (CA)
 - Command Interface
 - Components
 - Core Landlab Classes
 - Data Record
 - Landlab Fields
 - Landlab Framework
 - Landlab Graph
 - Landlab Grids
 - Input/Output (IO)
 - Layers
 - Plotting and Visualization
 - Testing Tools
 - Utilities and Decorators
 - Values
 - Full Index

Quick search

Go





Components

This section contains documentation and API reference information for the following categories of components:

Hillslope geomorphology

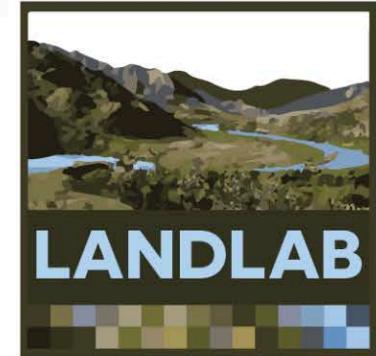
- `LinearDiffuser`: Model soil creep using “linear diffusion” transport law (no depth dependence)
- `PerronNLDiffuse`: Model soil creep using implicit solution to nonlinear diffusion law
- `DepthDependentDiffuser`: depth dependent diffusion after Johnstone and Hilley (2014)
- `TransportLengthHillslopeDiffuser`: Hillslope diffusion component in the style of Carretier et al. (2016), and Davy and Lague (2009)
- `TaylorNonLinearDiffuser`: Model non-linear soil creep after Ganti et al. (2013)
- `DepthDependentTaylorDiffuser`: Model depth dependent non-linear soil creep after Ganti et al. (2012) and Johnstone and Hilley (2014)

Fluvial geomorphology

- `FastscapeEroder`: Compute fluvial erosion using stream power theory (“fastscape” algorithm)
- `StreamPower`: Compute fluvial erosion using stream power theory (also uses “fastscape” algorithm but provides slightly different formulation and options)
- `SedDepEroder`: Compute fluvial erosion using using “tools and cover” theory
- `StreamPowerSmoothThresholdEroder`: Compute fluvial erosion using stream power theory with a numerically smoothed threshold
- `DetachmentLtdErosion`: Solve stream power equations, but without stability checks
- `ErosionDeposition`: Fluvial erosion in the style of Davy and Lague (2009)
- Stream Power with Alluvium Conservation and Entrainment

Flow routing

- The Landlab FlowDirectors: Components for Flow Direction
 - `FlowDirectorSteepest`
 - `FlowDirectorD8`
 - `FlowDirectorMFD`
 - `FlowDirectorDinf`



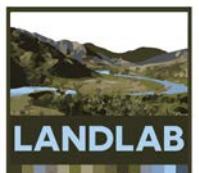
INDEX

Search

Table of Contents

Components

- [Hillslope geomorphology](#)
- [Fluvial geomorphology](#)
- [Flow routing](#)
- [Shallow water hydrodynamics](#)
- [Land surface hydrology](#)
- [Groundwater hydrology](#)
- [Landslides](#)
- [Vegetation](#)
- [Biota](#)
- [Precipitation](#)
- [Weathering](#)
- [Terrain Analysis](#)
- [Tectonics](#)
- [Fire](#)
- [Fracture Generation](#)





LinearDiffuser: Model soil creep using “linear diffusion” transport law (no depth dependence)

```
class LinearDiffuser(grid, linear_diffusivity=0.01, method='simple', deposit=True)[source]
```

Bases: [landlab.core.model_component.Component](#)

This component implements linear diffusion of a Landlab field.

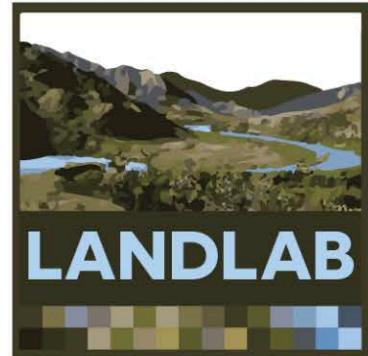
Component assumes grid does not deform. If the boundary conditions on the grid change after component instantiation, be sure to also call [updated_boundary_conditions](#) to ensure these are reflected in the component (especially if fixed_links are present).

The method keyword allows control of the way the solver works. Options other than ‘simple’ will make the component run slower, but give second order solutions that incorporate information from more distal nodes (i.e., the diagonals). Note that the option ‘resolve_on_patches’ can result in somewhat counterintuitive behaviour - this option tells the component to treat the diffusivity as a field **with directionality to it** (i.e., that the diffusivities are defined on links). Thus if all links have the same diffusivity value, with this flag active “effective” diffusivities at the nodes will be *higher* than this value (by a factor of root 2) as the diffusivity at each patch will be the mean vector sum of that at the bounding links.

The primary method of this class is [run_one_step](#).

Examples

```
>>> from landlab import RasterModelGrid
>>> import numpy as np
>>> mg = RasterModelGrid((9, 9))
>>> z = mg.add_zeros("topographic_elevation", at="node")
>>> z.reshape((9, 9))[4, 4] = 1.
>>> mg.set_closed_boundaries_at_grid_edges(True, True, True, True)
>>> ld = LinearDiffuser(mg, linear_diffusivity=1.)
>>> for i in range(1):
...     ld.run_one_step(1.)
... np.isclose(z[mg.core_nodes], sum([1.] * 9))
```



[INDEX](#)

[Search](#)

[Previous topic](#)

[Components](#)

[Next topic](#)

[PerronNLDiffuse: Model soil creep using implicit solution to nonlinear diffusion law](#)

[This Page](#)

[Show Source](#)

[Quick search](#)

 Go



Search or jump to

[landlab / landlab](#)[Used by](#) 17[Unwatch](#) 25[Star](#) 170[Fork](#) 177[Code](#)[Issues 245](#)[Pull requests 19](#)[Actions](#)[Projects 1](#)[Security](#)[Insights](#)[Settings](#)

Branch: master

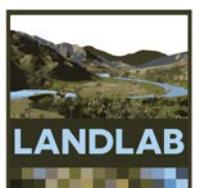
[landlab / notebooks /](#)[Create new file](#)[Upload files](#)[Find file](#)[History](#)

nathanlyons component to model species evolution (#1065) ...

Latest commit 6aab9b 9 days ago

..

	teaching	Post 2.0 documentation update (#1081)	last month
	tutorials	component to model species evolution (#1065)	9 days ago
	conftest.py	landlab v2 (#1077)	last month
	exclude.yml	landlab v2 (#1077)	last month
	landlab_header.png	landlab v2 (#1077)	last month
	test_notebooks.py	landlab v2 (#1077)	last month
	welcome.ipynb	Post 2.0 documentation update (#1081)	last month



The first example uses set_watershed_boundary_condition which finds the outlet for the user.

- First import what we need.

```
In [ ]: from landlab import RasterModelGrid
import numpy as np
from landlab.plot.imshow import imshow_grid
%matplotlib inline
```

- Now we create a 5 by 5 grid with a spacing (dx and dy) of 1.
- We also create an elevation field with value of 1. everywhere, except at the outlet, where the elevation is 0. In this case the outlet is in the middle of the bottom row, or at location (0,2) and has a node id of 2.

```
In [ ]: mg1 = RasterModelGrid((5,5), 1.)
z1 = mg1.add_ones('node','topographic_elevation')
mg1.at_node['topographic_elevation'][2] = 0.
mg1.at_node['topographic_elevation']
```

- The set_watershed_boundary_condition in RasterModelGrid will find the outlet of the watershed.
- This method takes the node data, in this case z, and, optionally the no_data value.
- This method sets all nodes that have no_data values to closed boundaries.
- This example does not have any no_data values, which is fine.
- In this case, the code will set all of the perimeter nodes as CLOSED_BOUNDARY (boundary status 4) in order to create this boundary around the core nodes.
- The exception on the perimeter is node 2 (with elevation of 0). Although it is on the perimeter, it has a value and it has the lowest value. So in this case node 2 will be set as FIXED_VALUE_BOUNDARY (boundary status 1).
- The rest of the nodes are set as a CORE_NODE (boundary status 0)

```
In [ ]: mg1.set_watershed_boundary_condition(z1)
```

- Check to see that node status were set correctly.
- imshow_grid will default to not plot the value of CLOSED_BOUNDARY nodes, which is why we override this below with the option color_for_closed

```
In [ ]: imshow_grid(mg1, mg1.status_at_node, color_for_closed='blue')
```

The second example uses set_watershed_boundary_condition_outlet_coords

- In this case the user knows the coordinates of the outlet node.
- First instantiate a new grid, with new data values.

```
In [ ]: mg2 = RasterModelGrid((5,5), 10.)
z2 = mg2.add_ones('node','topographic_elevation')
mg2.at_node['topographic_elevation'][1] = 0.
mg2.at_node['topographic_elevation']
```

Editor - /Users/nicolegasparini/LandlabWorkingScripts/nathan_base_landscape.py

```

18
19 # Set constants.
20 dt = 30000
21 m_sp = 0.5
22 n_sp = 1.0
23
24 seed_init_elevation = 27
25 #U = 10.**params['U_exp']
26 U = 1e-3
27 uplift_per_step = U * dt
28 K_sp = 1e-7
29 #k_d = 10.**params['linear_diffusivity_exp']
30
31 dx = 200
32
33 A_c = 5e5
34 k_d = (A_c * K_sp * np.sqrt(A_c / dx**2/3)) / (2 * dx**2/3)
35 k_d = float(k_d)
36 #k_d = 0
37
38 print(A_c, K_sp, k_d)
39
40 # Create a grid with random initial topography.
41 nrows = 100
42 ncols = 100
43 mg_ss = RasterModelGrid(nrows, ncols, dx)
44 z_ss = mg_ss.add_zeros('node', 'topographic__elevation')
45 np.random.seed(seed_init_elevation)
46 z_ss += np.random.rand(z_ss.size)
47
48 # Set boundaries.
49 mg_ss.set_status_at_node_on_edges(right=CLOSED_BOUNDARY,
50                                     top=FIXED_VALUE_BOUNDARY,
51                                     left=CLOSED_BOUNDARY,
52                                     bottom=FIXED_VALUE_BOUNDARY)
53
54 # Instantiate model components.
55 fr = FlowRouter(mg_ss)
56 sp = FastscapeEroder(mg_ss, K_sp=K_sp, m_sp=m_sp, n_sp=n_sp)
57 lf = LinearDiffuser(mg_ss, linear_diffusivity=k_d, deposit=False)
58 hf = SinkFiller(mg_ss)
59
60 imshow_grid(mg_ss, 'topographic__elevation')
61
62 t = 0
63
64 while t<3e6:
65     z_ss[mg_ss.core_nodes] += uplift_per_step
66     hf.run_one_step()
67     fr.run_one_step()
68     sp.run_one_step(dt)
69     lf.run_one_step(dt)
70     t = t+dt
71     print('time:', t)
72
73 imshow_grid(mg_ss, 'topographic__elevation')

```

Variable explorer

Name	Type	Size	Value
A_c	float	1	500000.0
K_sp	float	1	1e-07
U	float	1	0.001
dt	int	1	30000
dx	int	1	200
k_d	float	1	0.08838834764831847
m_sp	float	1	0.5
n_sp	float	1	1.0

IPython console

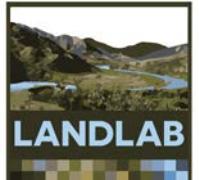
Console 1/A

```

time: 2010000
time: 2040000
time: 2070000
time: 2100000
time: 2130000
time: 2160000
time: 2190000
time: 2220000
time: 2250000
time: 2280000
time: 2310000
time: 2340000
time: 2370000
time: 2400000
time: 2430000
time: 2460000
time: 2490000
time: 2520000
time: 2550000
time: 2580000
time: 2610000
time: 2640000
time: 2670000
time: 2700000
time: 2730000
time: 2760000
time: 2790000
time: 2820000
time: 2850000
time: 2880000
time: 2910000
time: 2940000
time: 2970000
time: 3000000

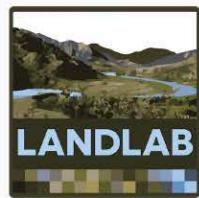
```

In [2]:





Search or jum

<https://github.com/landlab>

Landlab

a python toolkit for modeling earth surface processes

↳ <http://landlab.github.io>

[Repositories 17](#)[Packages](#)[People 15](#)[Teams 2](#)[Projects](#)[Settings](#)[Type: All ▾](#)[Language: All ▾](#)[Customize pins](#)[New](#)

psu-clinic-2020

A Landlab clinic at Penn State, 2020 January 22

[Jupyter Notebook](#)[MIT](#)[0](#)[0](#)[0](#)[0](#)

Updated 1 hour ago

Top languages

[Python](#)[Jupyter Notebook](#)[HTML](#)[Shell](#)[JavaScript](#)

landlab

Landlab codebase, wiki, and tests

[Python](#)[MIT](#)[177](#)[170](#)[245 \(5 issues need help\)](#)[19](#)

Updated 1 hour ago



landlab.github.io

Landlab website

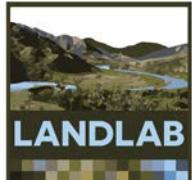
[HTML](#)[0](#)[0](#)[0](#)[1](#)

Updated on Dec 20, 2019

People

[15 >](#)

Invite your teammates...



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

The screenshot shows the GitHub repository page for `landlab / landlab`. The main navigation bar includes links for Pull requests, Issues, Marketplace, and Explore. The Issues tab is selected, showing 178 open issues. A modal window titled "Label issues and pull requests for new contributors" is displayed, explaining that GitHub will help potential first-time contributors discover issues labeled with "help wanted" or "good first issue". Below the modal, there are filters for "is:issue is:open", a "New issue" button, and a list of 178 open issues. The issues are listed in descending order of creation date, with the most recent at the top. Each issue card includes the title, a brief description, the number of comments, and the user who opened it.

Label issues and pull requests for new contributors

Now, GitHub will help potential first-time contributors discover issues labeled with [help wanted](#) or [good first issue](#)

[Go to Labels](#)

Dismiss

Filters ▾ is:issue is:open Labels Milestones New issue

178 Open ✓ 166 Closed

Author ▾ Labels ▾ Projects ▾ Milestones ▾ Assignee ▾ Sort ▾

- ① update lake filling and routing routines [enhancement](#)
#761 opened a day ago by SiccarPoint 5
- ① Unclear testing procedures
#758 opened 5 days ago by SiccarPoint 9
- ① running pytest results in a figure being generated
#756 opened 6 days ago by kbarnhart 3
- ① "Download all tutorials" link on wiki doesn't point to current tutorials
#753 opened 6 days ago by kbarnhart 3
- ① Qs in external for SPACE and ErosionDeposition
#750 opened 13 days ago by kbarnhart 5
- ① Negative runoffs/infiltration [bug](#) [enhancement](#)
#744 opened 14 days ago by SiccarPoint 5
- ① Specify total throw amount in NormalFault instead of rate for run_one_step [bug](#) [enhancement](#)
#741 opened 15 days ago by nreitman 1
- ① Grid and Graph: node and link arrangements should match for hex/rect grids and graphs
#739 opened on Jun 24 by grettucker 1
- ① Graph: FutureWarning from xarray
#738 opened on Jun 24 by grettucker 1
- ① landlab/.conda-recipe/meta.yaml doesn't include sympy
#736 opened on Jun 21 by kbarnhart 1
- ① Docs broken due to new sphinx
#734 opened on Jun 19 by kbarnhart 1
- ① In hex grids, reorient_links argument may be obsolete
#731 opened on Jun 14 by grettucker 1

Your average undergrad knows nothing about Landlab and numerical modeling. Maybe even nothing about coding.

- Step in their shoes!
 - They can still learn from models.
-
- go to: <https://github.com/landlab>
 - Landlab2020



landlab/landlab_teaching_tools x Home x hillslope_diffusion_class_notebook.ipynb

jupyter hillslope_diffusion_class_notebook (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Run Markdown

 landlab/landlab_teaching_tools: Codes for use in undergraduate and graduate courses

Modeling earth surface processes

Linear diffusion exercise with Landlab

This notebook is adapted from *Landscape Evolution Modeling with CHILD* by Gregory Tucker and Stephen Lancaster. This notebook was created by Nicole Gasparini at Tulane University.

For tutorials on learning Landlab, click here: <https://github.com/landlab/landlab/wiki/Tutorials>

What is this notebook?

This notebook illustrates the evolution of landforms dominated by processes that result in linear diffusion of sediment. In other words, the downhill flow of soil is proportional to the (downhill) gradient of the land surface multiplied by a transport coefficient.

The notebook first illustrates a simple example of a diffusing hillslope. We then provide a number of exercises for students to do on their own. This set of exercises is recommended for students in a quantitative geomorphology class, who have been introduced to the linear diffusion equation in class.

Application of linear diffusion transport law:

For relatively gentle, soil-mantled slopes, there is reasonably strong support for a transport law of the form:

$$q_s = -D \nabla z$$

where q_s is the transport rate with dimensions of $L^2 T^{-1}$; D is a transport coefficient with dimensions of $L^2 T^{-1}$; and z is elevation. ∇z is the gradient in elevation. If distance is increasing downslope, ∇z is negative downslope, hence the negative in front of D .

... erosion, are calculated from conservation of mass:

landlab.github.io



jupyter hillslope_diffusion_class_notebook (unsaved changes)



Logout

File Edit View Insert Cell Kernel Widgets Help
Not Trusted | Python 3

Remember that you can always go to the Kernel pulldown menu at the top and choose Restart & Clear Output or Restart & Run.
All you change things and want to start afresh. If you just change one code block and rerun only that code block, only the parts of the code in that code block will be updated. (E.g. if you change parameters but don't reset the code blocks that initialize run time or topography, then these values will not be reset.)

Now on to the code example

Import statements. You should not need to edit this.

```
In [1]: # Code Block 1

import numpy as np
from landlab.plot.imshow import imshow_grid
#below is to make plots show up in the notebook
%matplotlib inline
from matplotlib.pyplot import figure, show, plot,
    xlabel, ylabel, title, legend, ylim
```

We will create a grid with 41 rows and 5 columns, and dx is 5 m (a long, narrow, hillslope). The initial elevation is 0 at all nodes.

We set-up boundary conditions so that material can leave the hillslope at the two short ends.

```
In [ ]: # Code Block 2

# setup grid
from landlab import RasterModelGrid
mg = RasterModelGrid((41, 5), 5.)
z_vals = mg.add_zeros('topographic_elevation', at='node')

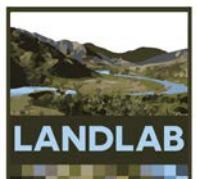
# initialize some values for plotting
ycoord_rast = mg.node_vector_to_raster(mg.node_y)
ys_grid = ycoord_rast[:, 2]

# set boundary condition.
mg.set_closed_boundaries_at_grid_edges(True, False, True, False)
```

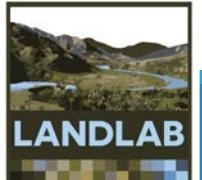
Now we import and initialize the *LinearDiffuser* component.

```
In [ ]: # Code Block 3

from landlab.components import LinearDiffuser
D = 0.01 # initial value of 0.01 m^2/yr
lin_diffuse = LinearDiffuser(mg, linear_diffusivity=D)
```



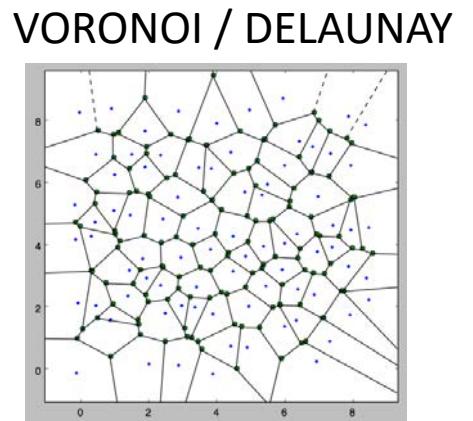
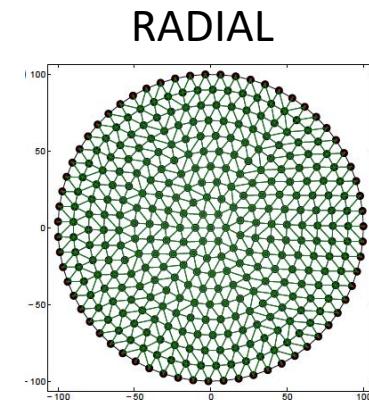
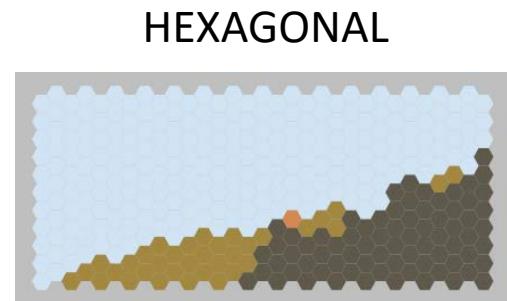
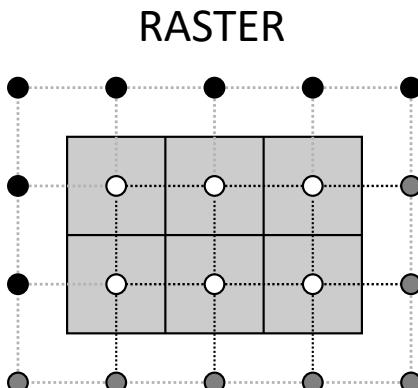
Take ~10 minutes to play with this notebook.



What Landlab provides

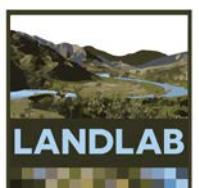
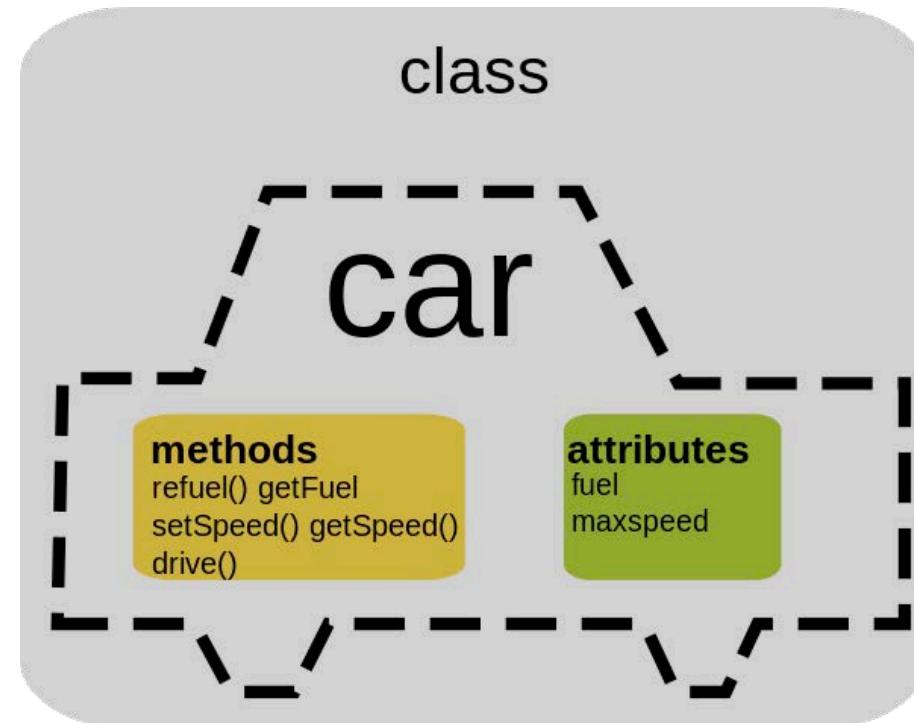
1. Grid creation and management

- 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

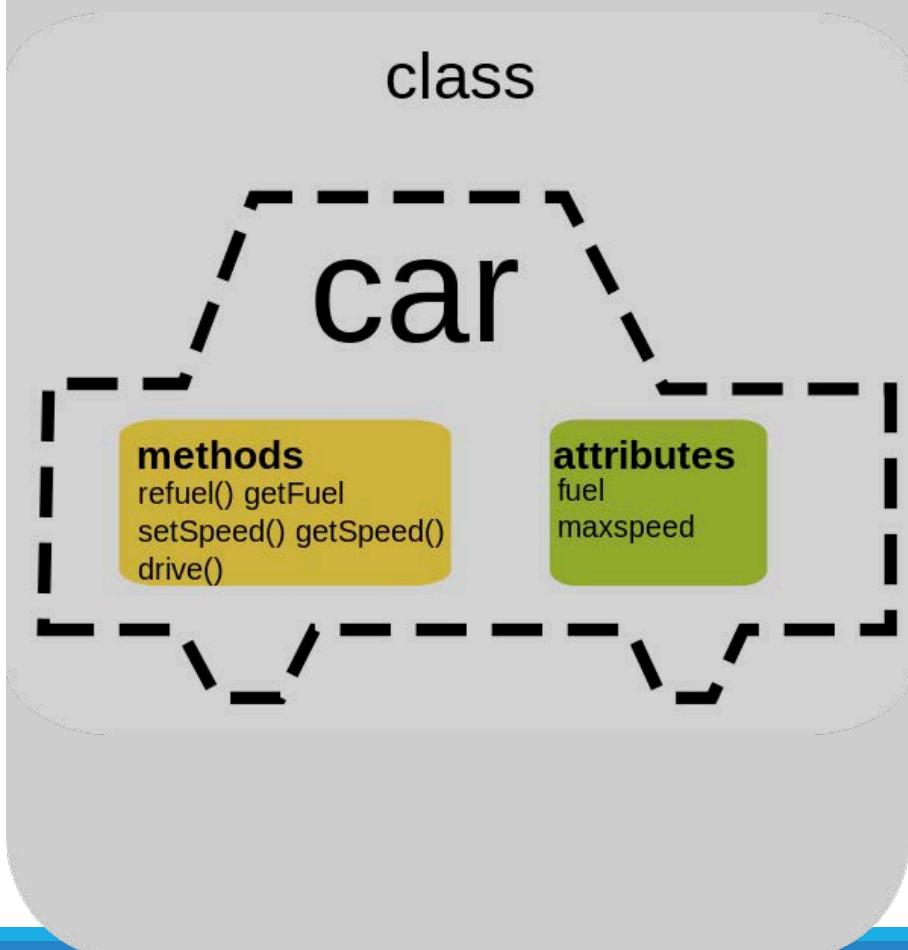


Quick side bar on Object Oriented Programming:

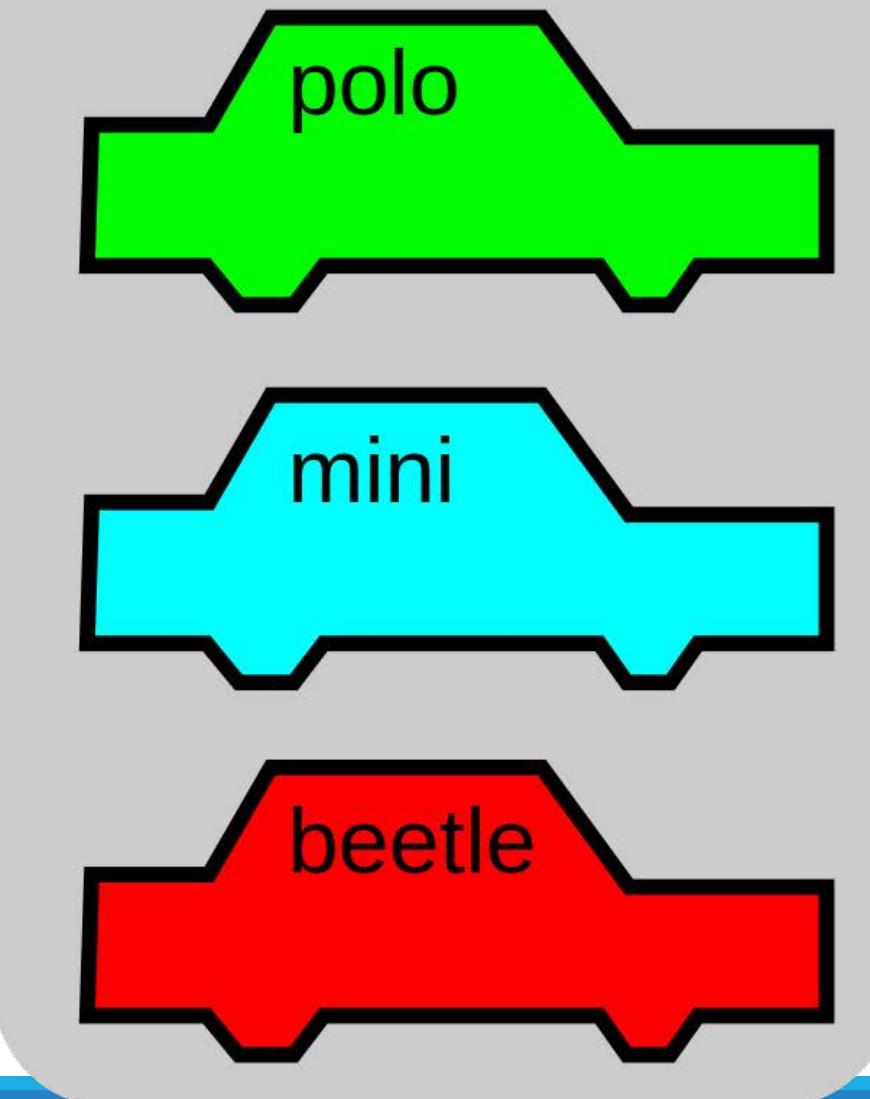
From Wikipedia: “**Object-oriented programming (OOP)** is a programming paradigm based on the concept of "objects", which may contain data, ... often known as *attributes*; and code, in the form of procedures, often known as methods. ”
[\(https://en.wikipedia.org/wiki/Object-oriented_programming\)](https://en.wikipedia.org/wiki/Object-oriented_programming)



class

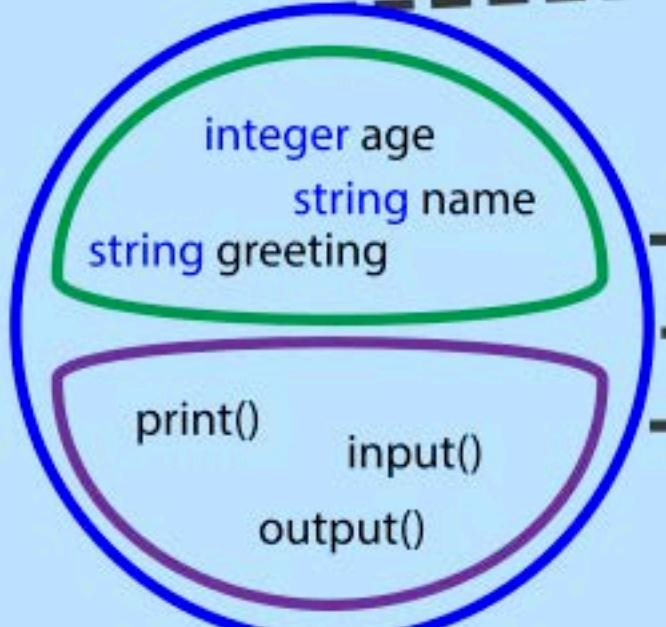


objects



Object Oriented Programming

Class



Objects

Attributes

`age = 15`
`name = "Bob"`
`greeting = "Howdy!"`

`print()`
`input()`
`output()`

Procedural Programming Equivalent

Variables

`age = 15`
`name = "Bob"`
`greeting = "Howdy!"`

`print()`
`input()`
`output()`

Procedures



RasterModelGrid

Boundary nodes

Close B

Core nodes

Close B

Open B

(+)

(+)

(+)

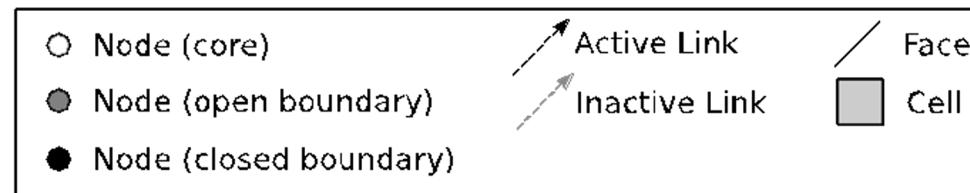
Links

inactive active

Link = directed line segment connecting two adjacent nodes

Link direction is toward upper right half-space by default

Tail node Head node



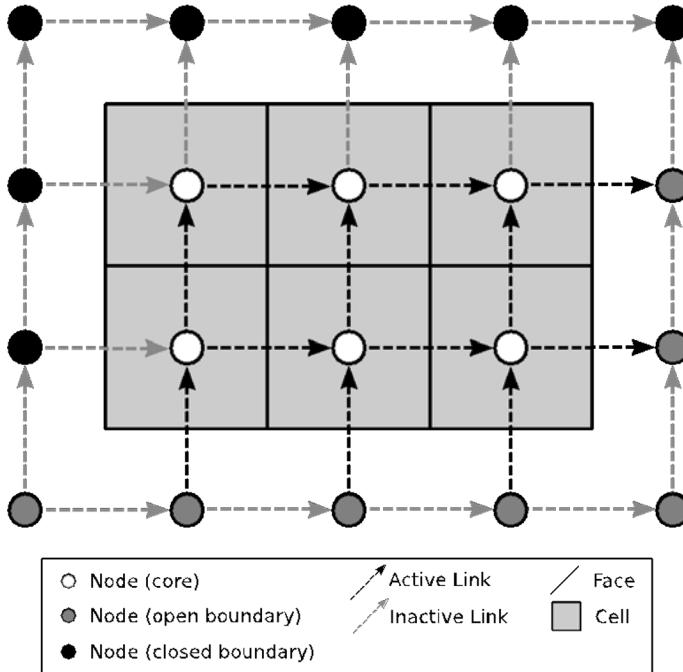
CORE_NODE: 0; FIXED_VALUE_BOUNDARY: 1 (e.g. outlet)

TRACKS_CELL_BOUNDARY: 3 (looped boundary); CLOSED_BOUNDARY: 4



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

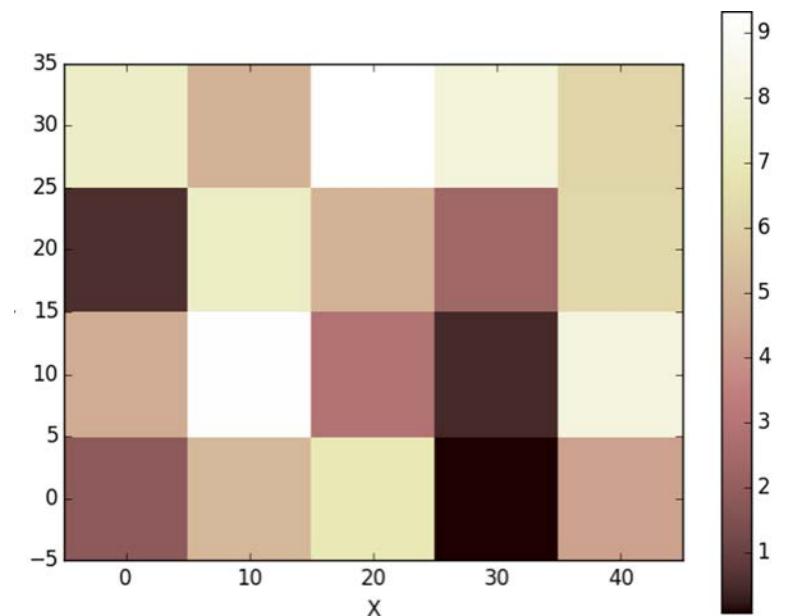
```
>> from landlab import RasterModelGrid  
>> rg = RasterModelGrid((4,5),10.0)
```

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)
```

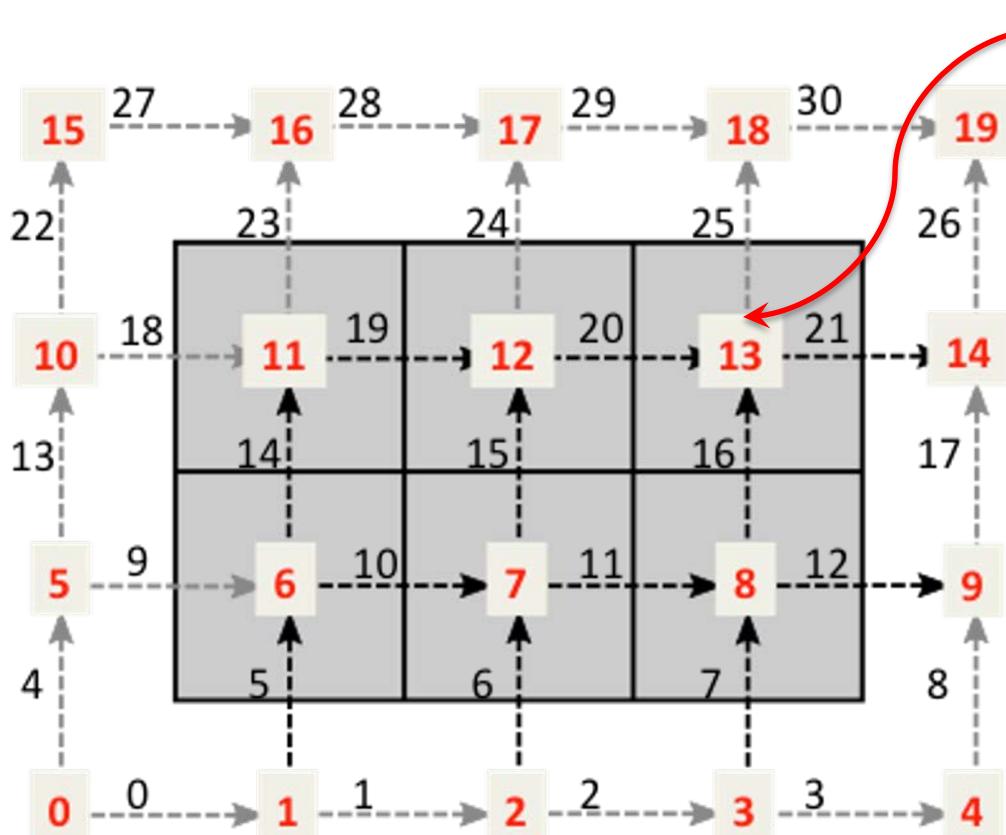
Plot the elevation field

```
>> 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

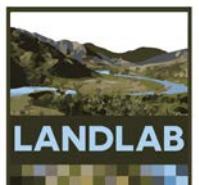
CORE_NODE: 0; FIXED_VALUE_BOUNDARY: 1 (e.g. outlet)

TRACKS_CELL_BOUNDARY: 3 (looped boundary); CLOSED_BOUNDARY: 4

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.





Components

This section contains documentation and API reference information for the following categories of components:

Hillslope geomorphology

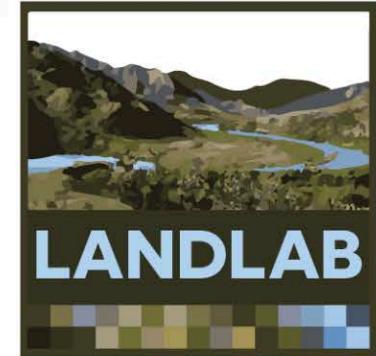
- `LinearDiffuser`: Model soil creep using “linear diffusion” transport law (no depth dependence)
- `PerronNLDiffuse`: Model soil creep using implicit solution to nonlinear diffusion law
- `DepthDependentDiffuser`: depth dependent diffusion after Johnstone and Hilley (2014)
- `TransportLengthHillslopeDiffuser`: Hillslope diffusion component in the style of Carretier et al. (2016), and Davy and Lague (2009)
- `TaylorNonLinearDiffuser`: Model non-linear soil creep after Ganti et al. (2013)
- `DepthDependentTaylorDiffuser`: Model depth dependent non-linear soil creep after Ganti et al. (2012) and Johnstone and Hilley (2014)

Fluvial geomorphology

- `FastscapeEroder`: Compute fluvial erosion using stream power theory (“fastscape” algorithm)
- `StreamPower`: Compute fluvial erosion using stream power theory (also uses “fastscape” algorithm but provides slightly different formulation and options)
- `SedDepEroder`: Compute fluvial erosion using using “tools and cover” theory
- `StreamPowerSmoothThresholdEroder`: Compute fluvial erosion using stream power theory with a numerically smoothed threshold
- `DetachmentLtdErosion`: Solve stream power equations, but without stability checks
- `ErosionDeposition`: Fluvial erosion in the style of Davy and Lague (2009)
- Stream Power with Alluvium Conservation and Entrainment

Flow routing

- The Landlab FlowDirectors: Components for Flow Direction
 - `FlowDirectorSteepest`
 - `FlowDirectorD8`
 - `FlowDirectorMFD`
 - `FlowDirectorDinf`



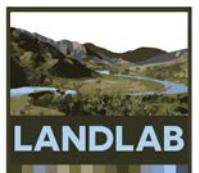
INDEX

Search

Table of Contents

Components

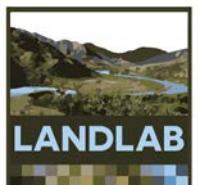
- [Hillslope geomorphology](#)
- [Fluvial geomorphology](#)
- [Flow routing](#)
- [Shallow water hydrodynamics](#)
- [Land surface hydrology](#)
- [Groundwater hydrology](#)
- [Landslides](#)
- [Vegetation](#)
- [Biota](#)
- [Precipitation](#)
- [Weathering](#)
- [Terrain Analysis](#)
- [Tectonics](#)
- [Fire](#)
- [Fracture Generation](#)



What Landlab provides

3. Input, output, visualization

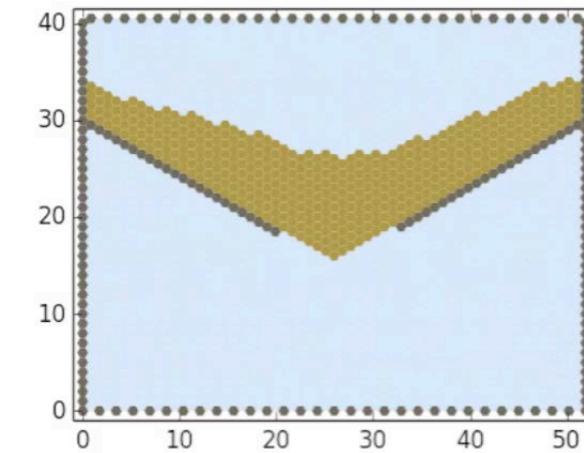
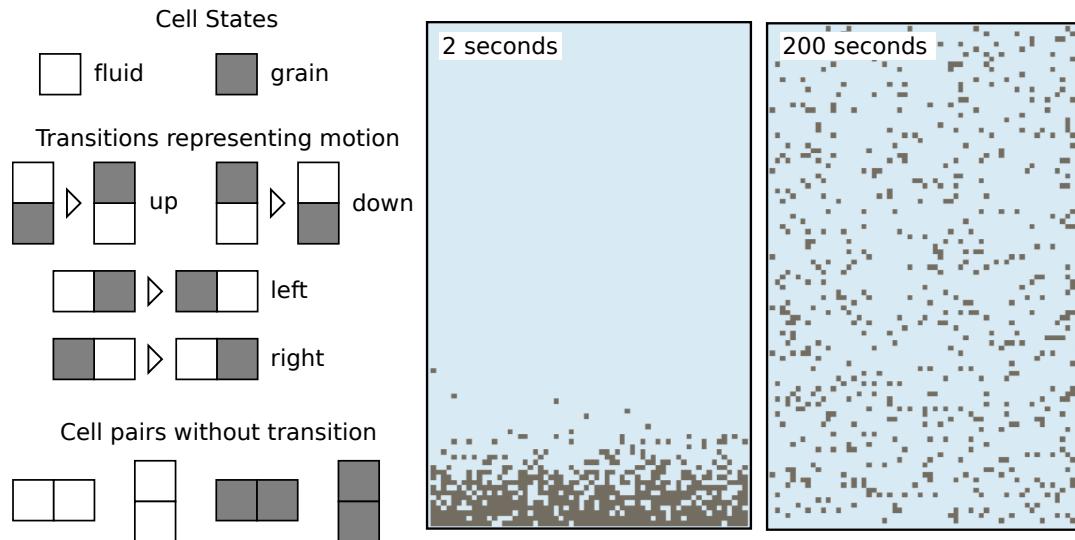
- **Read model parameters** from a formatted text file (optional)
- **Read in** digital terrain data (e.g., **DEMs**) → grid
- **Write gridded output** to files (netCDF format)
- **Plot** data using Matplotlib graphics library



What Landlab provides

4. Support for cellular-automaton modeling

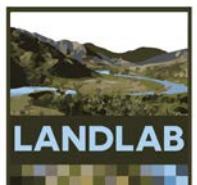
- CellLab-CTS: Continuous-time stochastic CA model “engine”

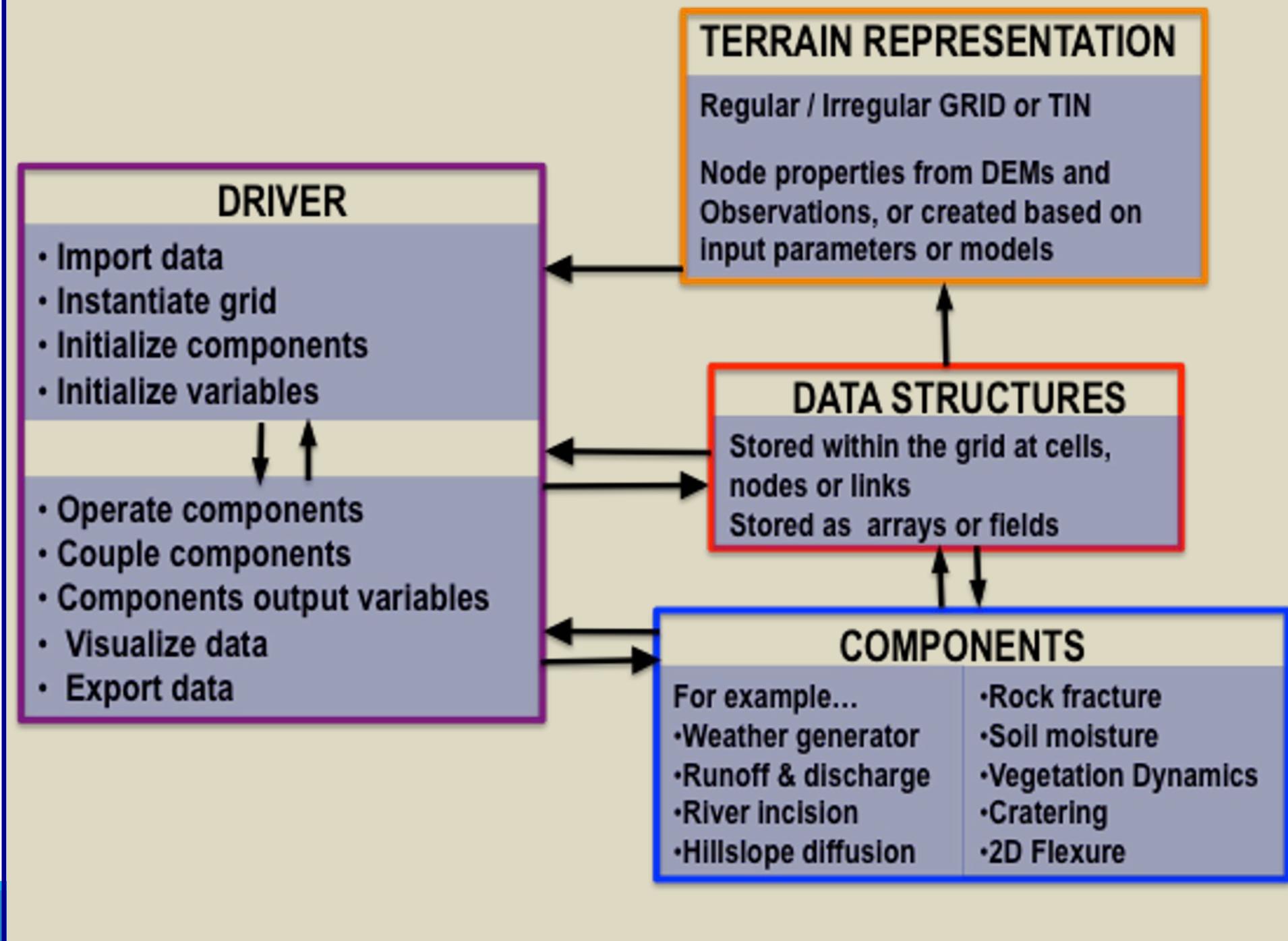


(Tucker et al., 2016 Geoscientific Model Development)

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**.

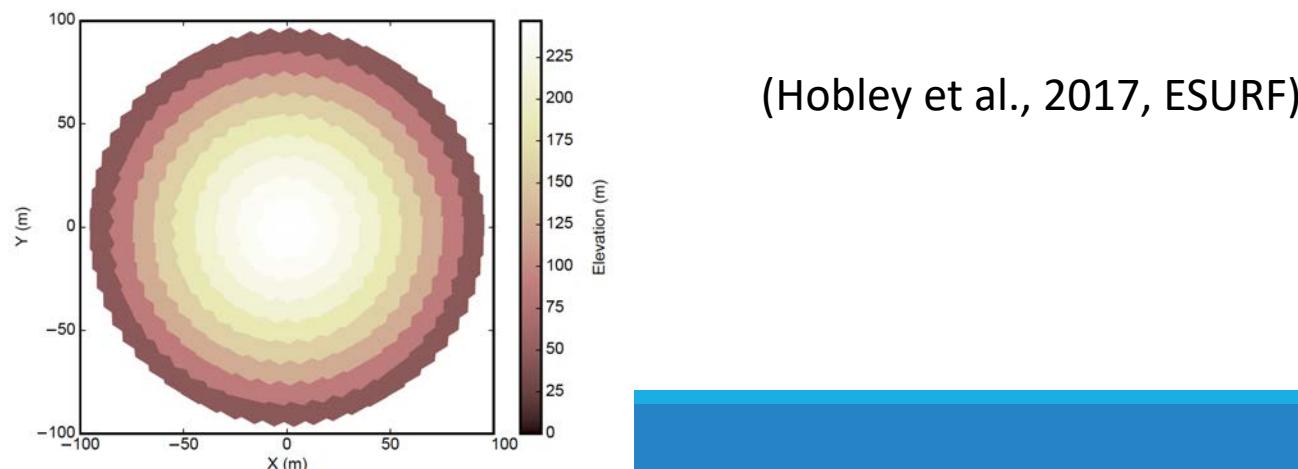




Example single component diffusion model

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
7. >>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
8. >>> for i in range(500):
9. ...     z[mg.core_nodes] += 0.001*dt
10. ...     ld.run_one_step(dt)
11. >>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
12. >>> show()
```



Linear Diffusion Equation

$$q_s = -K_d \nabla z$$

z = elevation

t = time

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

U = rock uplift

q_s = sediment transport rate

x = distance downslope

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

K_d = linear diffusivity

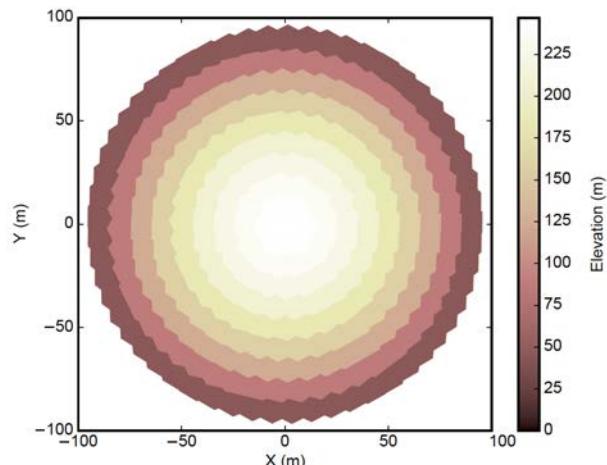


Single component diffusion model

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

Import Statements

```
1. >>> from landlab import RadialModelGrid, imshow_grid
2. >>> from landlab.components import LinearDiffuser
3. >>> from matplotlib.pyplot import show
4. >>> mg = RadialModelGrid(100, radius=100., outer=True)
5. >>> z = mg.add_zeros('node', 'topographic_elevation')
6. >>> dt = 2000. # no longer the stable timestep
7. >>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
8. >>> for i in range(500):
9. ...     z[mg.core_nodes] += 0.001*dt
10. ...     ld.run_one_step(dt)
11. >>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
12. >>> show()
```

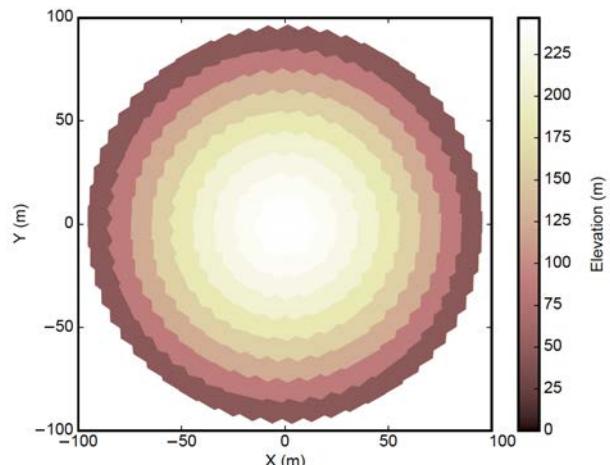


(Hobley et al., 2017, ESURF)

Single component diffusion model

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

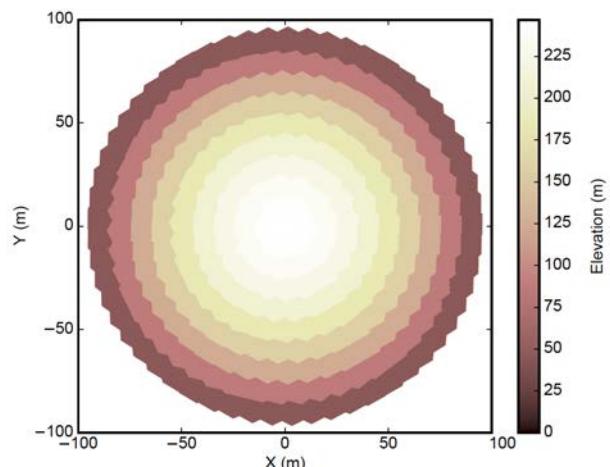
```
1. >>> from landlab import RadialModelGrid, imshow_grid
2. >>> from landlab.components import LinearDiffuser
3. >>>
4. >>> mg = RadialModelGrid(num_shells=10, dr=10.)
5. >>> z = mg.add_zeros('node', 'topographic_elevation')
6. >>> dt = 2000. # no longer the stable timestep
7. >>> ld = LinearDiffuser(mg,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



Single component diffusion model

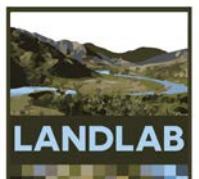
Code to implement a simple diffusion model on a radial Landlab grid, using Landlab components:

```
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("elevation", at="node")
6. >>> dt = 2000. # no longer the stable timestep
7. >>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
8. >>> for i in range(500):
9. ...     z[mg.core_nodes] += 0.001*dt
10. ...     ld.run_one_step(dt)
11. >>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
12. >>> show()
```



(Hobley et al., 2017, ESURF)

Initialize
parameter;
Instantiate
diffusion
object

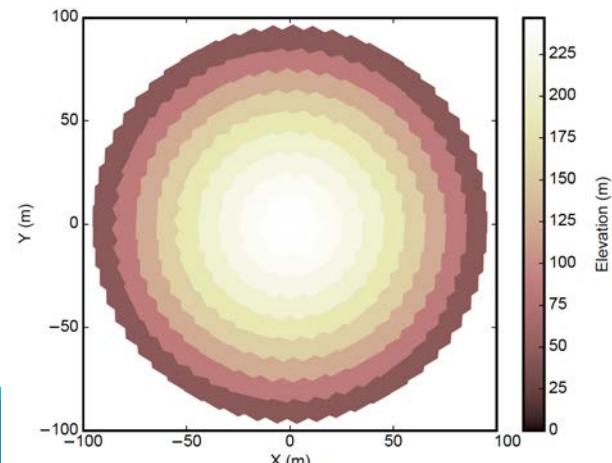


Single component diffusion model

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)
12. >>> show()
```



Time loop;
Do the
calculations!

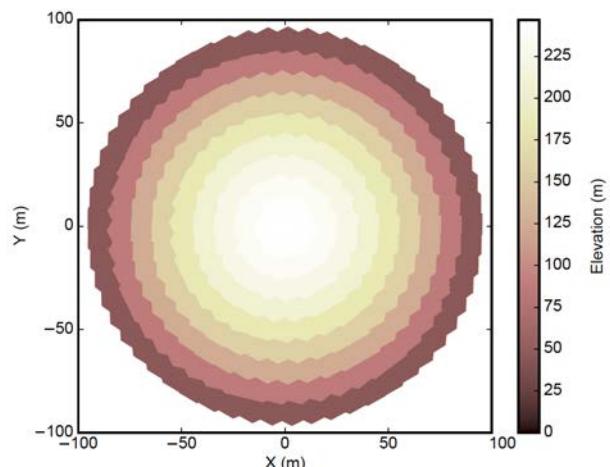
(Hobley et al., 2017, ESURF)

Single component diffusion model

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

```
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
7. >>> ld = LinearDiffuser(mg, linear_diffusivity=1.e-2)
8. >>> for i in range(500):
9. ...     z[mg.core_nodes] += 0.001*dt
10. ...
11. >>> imshow_grid(mg, z, grid_units=('m', 'm'), var_name='Elevation (m)')
12. >>> show()
```

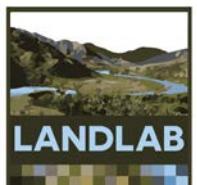
Visualize
data



(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 **Jupyter Notebook (.ipynb)**, which is great for teaching and learning.



Run some more models on CSDMS JupyterHub

- Overland Flow
- Simple Landscape Evolution Model
- Landscape and life evolution

