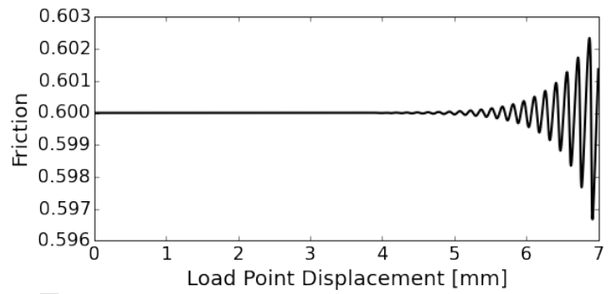
Modeling Rate-and-State Friction with Python

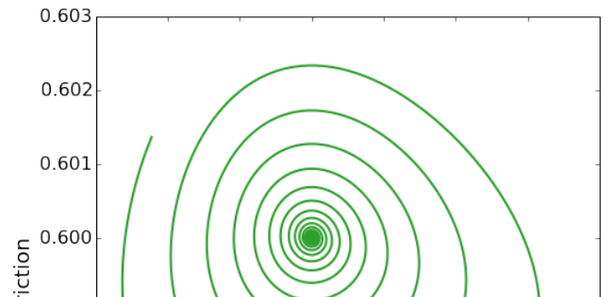
J.R. Leeman, R. May, C. Marone

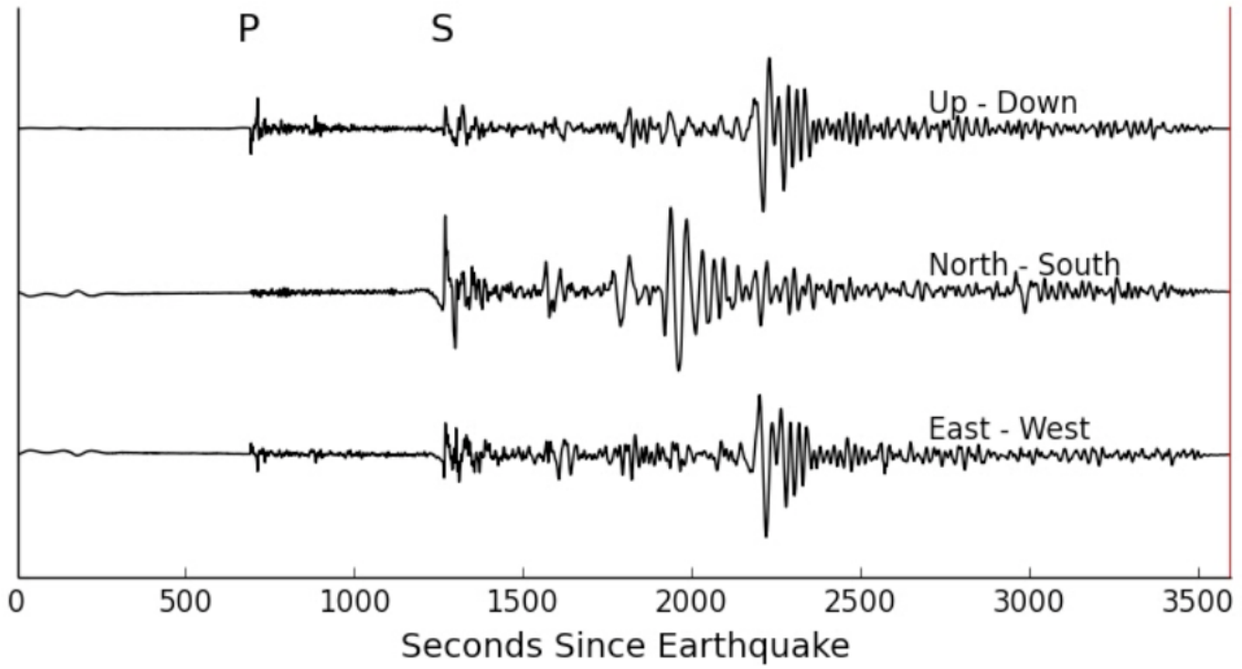
johnrleeman.com/@geo_leeman

Department of Geosciences
The Pennsylvania State University

July 13, 2016







Avoiding Common Academic Software Slip Ups

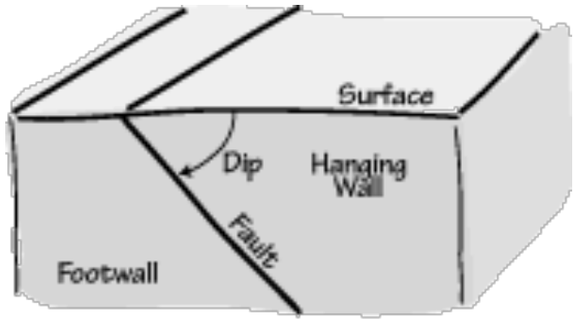
J.R. Leeman, R. May, C. Marone

johnrleeman.com/@geo_leeman
Department of Geosciences
The Pennsylvania State University

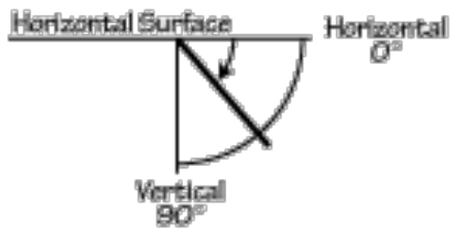
July 13, 2016

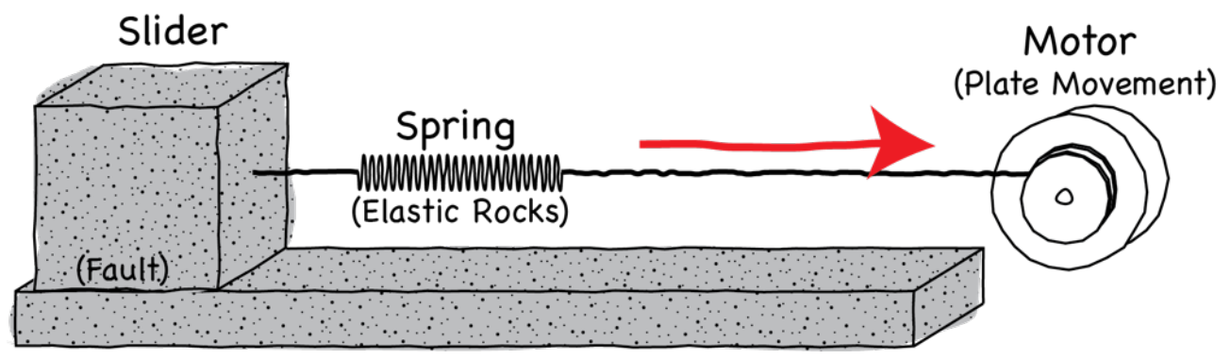
PENN STATE ROCK AND SEDIMENT MECHANICS LABORATORY

We are going to look at a real life problem and then make a tool to help solve it for researchers and educators



C. Ammon



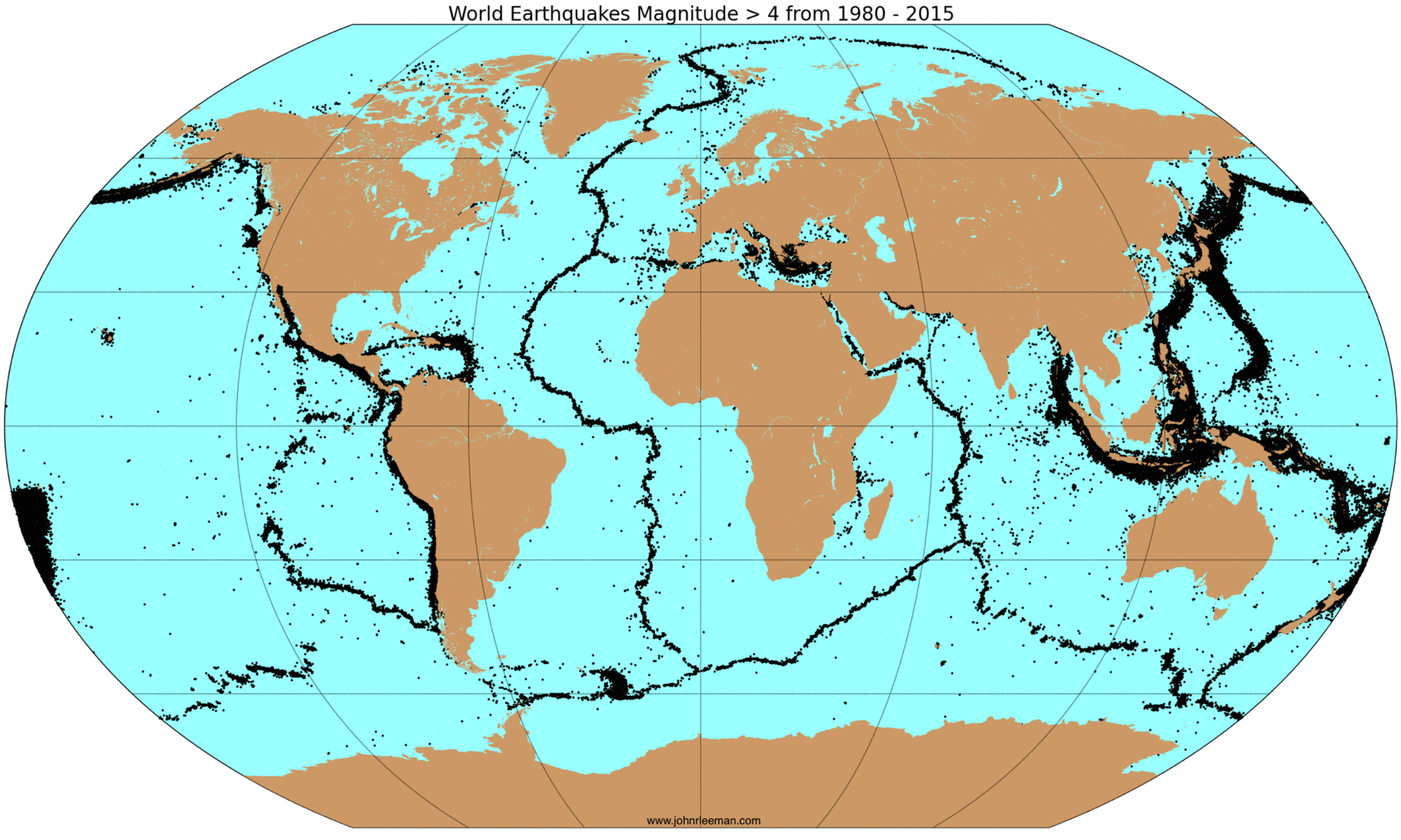


Earthquakes 101

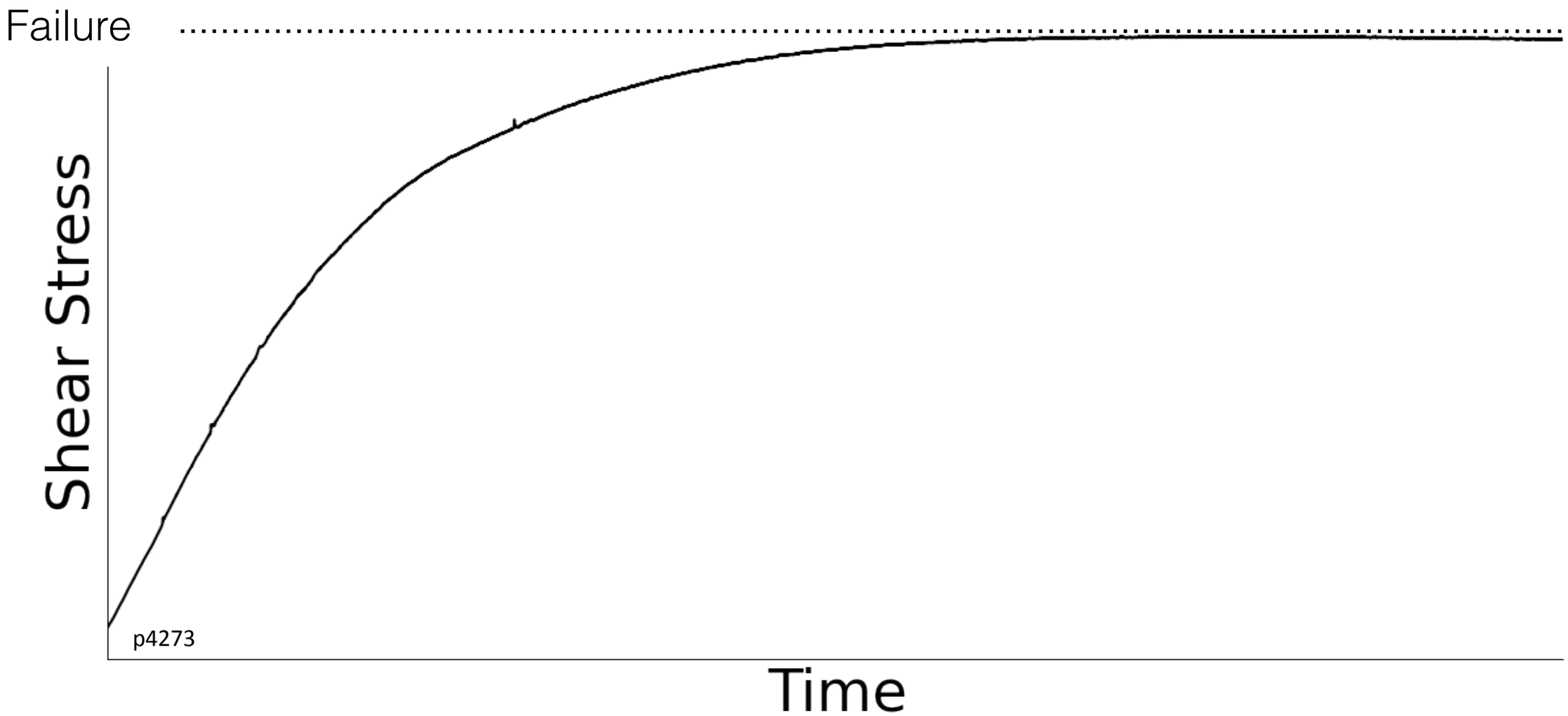
Frictional Theory

Python Implementation

Most earthquakes occur on plate boundaries



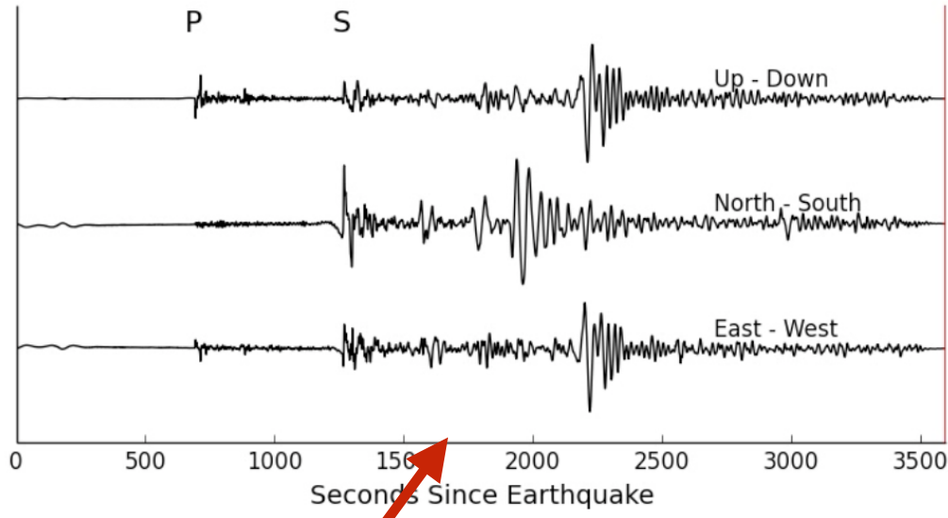
Faults can creep in an aseismic (stable) fashion

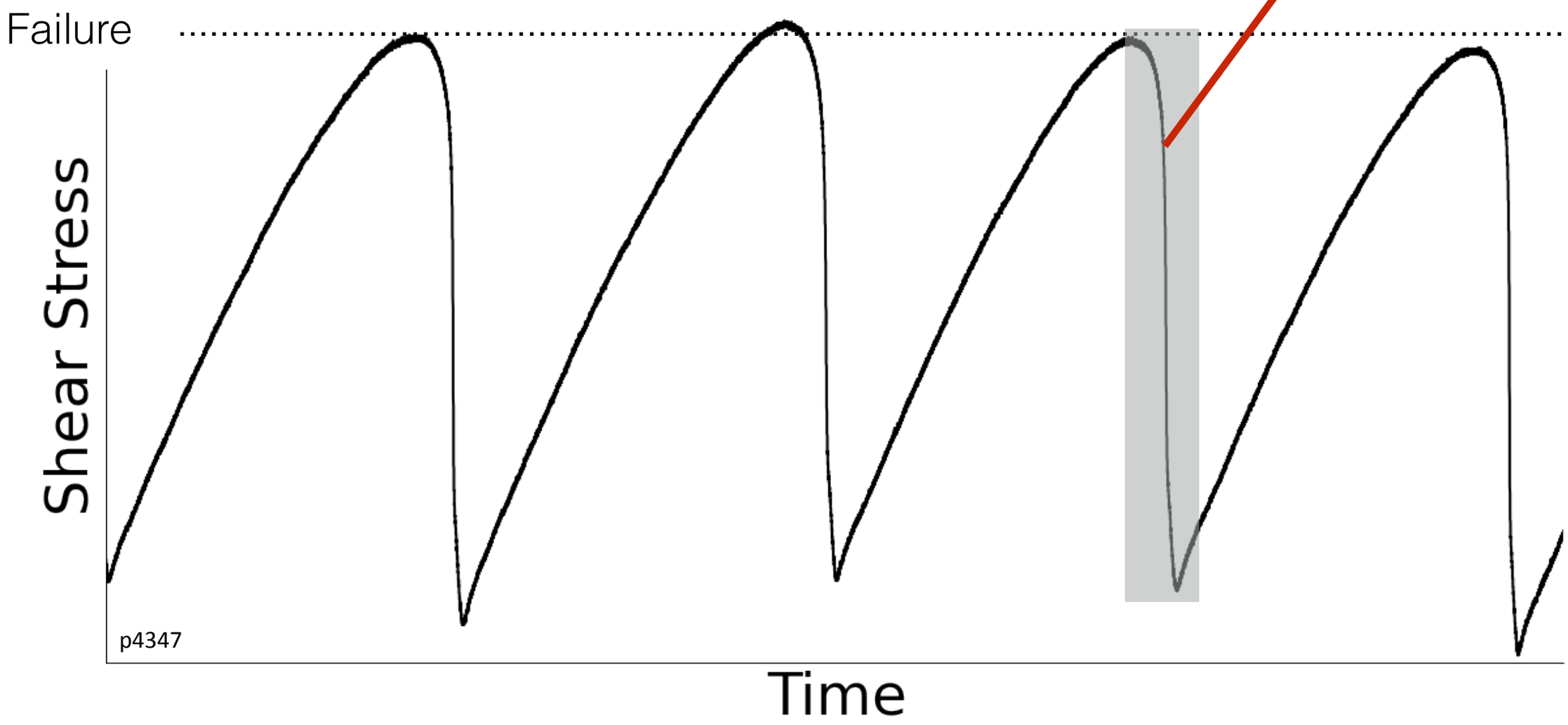


Creeping faults present some infrastructure risks

Images: Dr. Peggy Hellweg, Wikipedia, USGS

Faults can fail in an unstable and dynamic way that produces seismic radiation

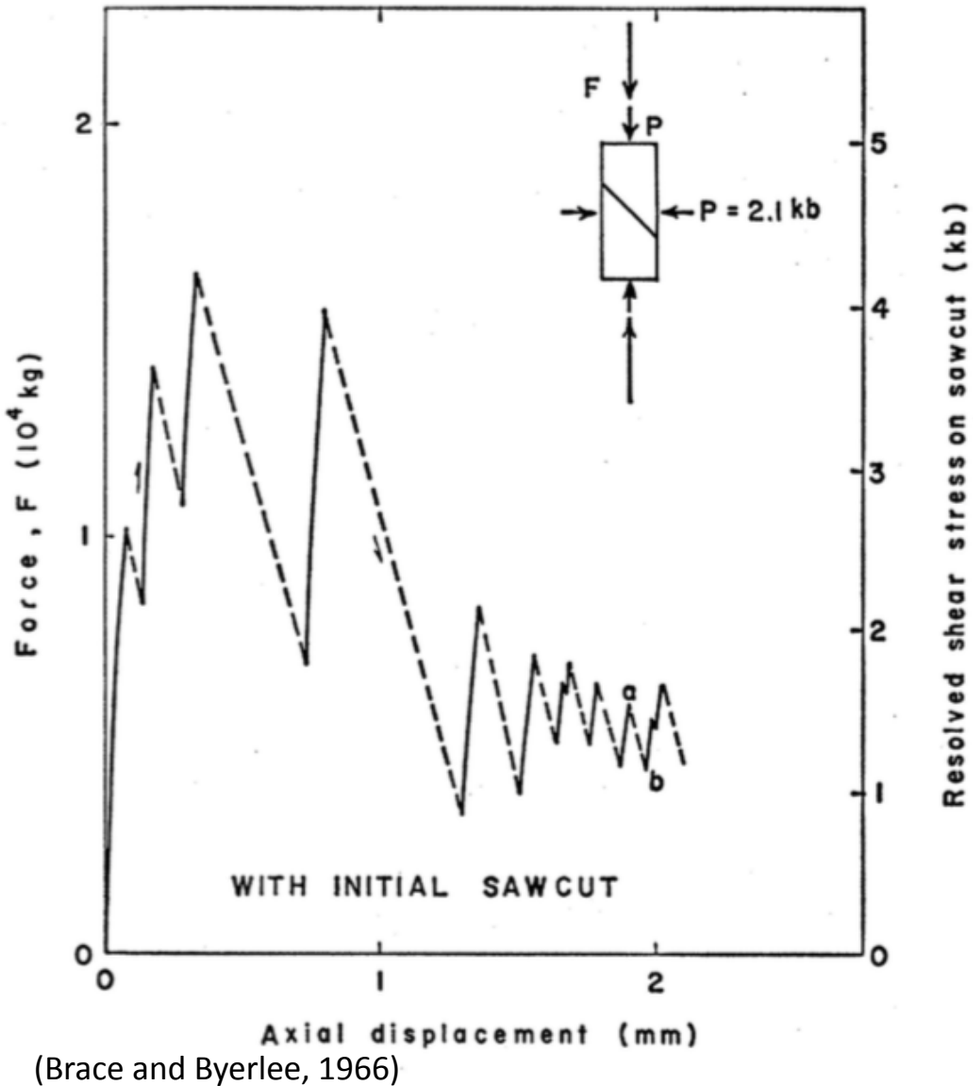




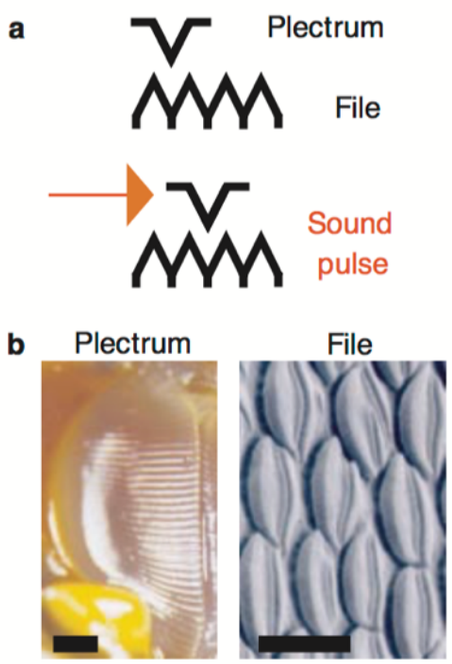
We have conceptualized earthquakes as a stick-slip process

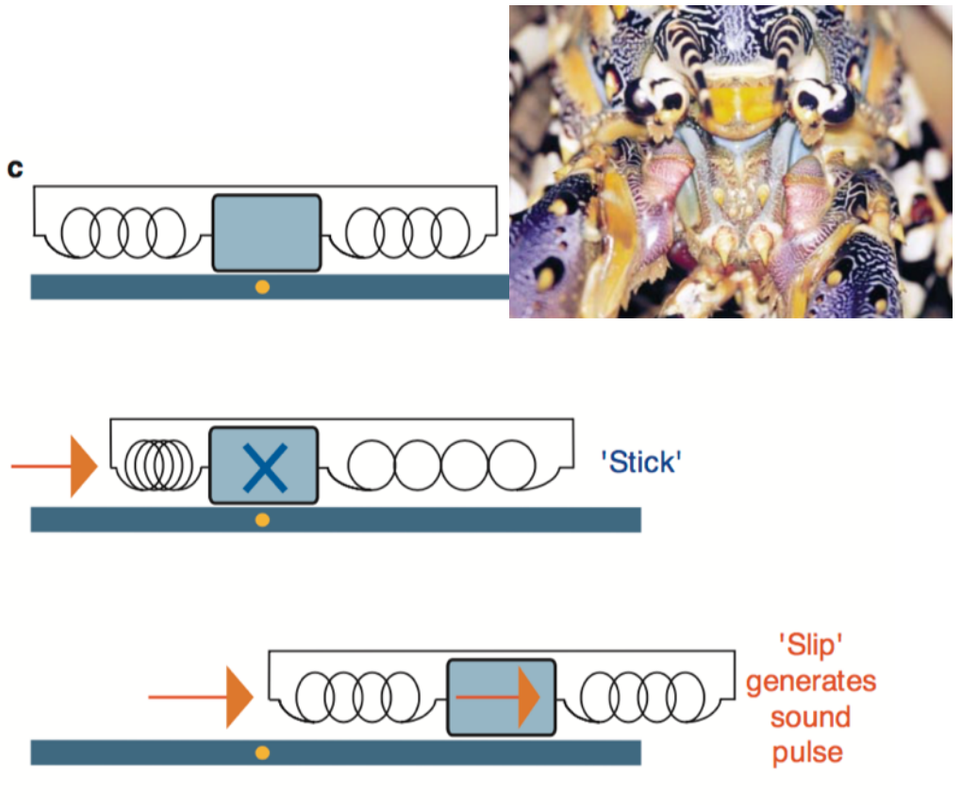
Stick-Slip as a Mechanism for Earthquakes

Abstract. *Stick-slip often accompanies frictional sliding in laboratory experiments with geologic materials. Shallow-focus earthquakes may represent stick-slip during sliding along old or newly formed faults in the earth. In such a situation, observed stress drops represent release of a small fraction of the stress supported by the rock surrounding the earthquake focus.*



(Brace and Byerlee, 1966)

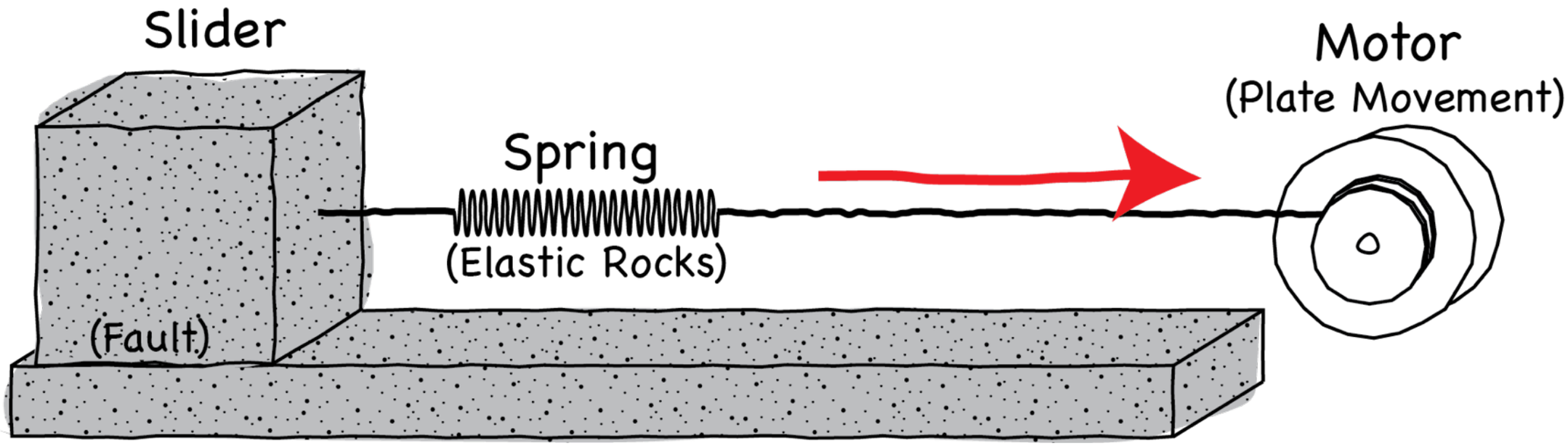




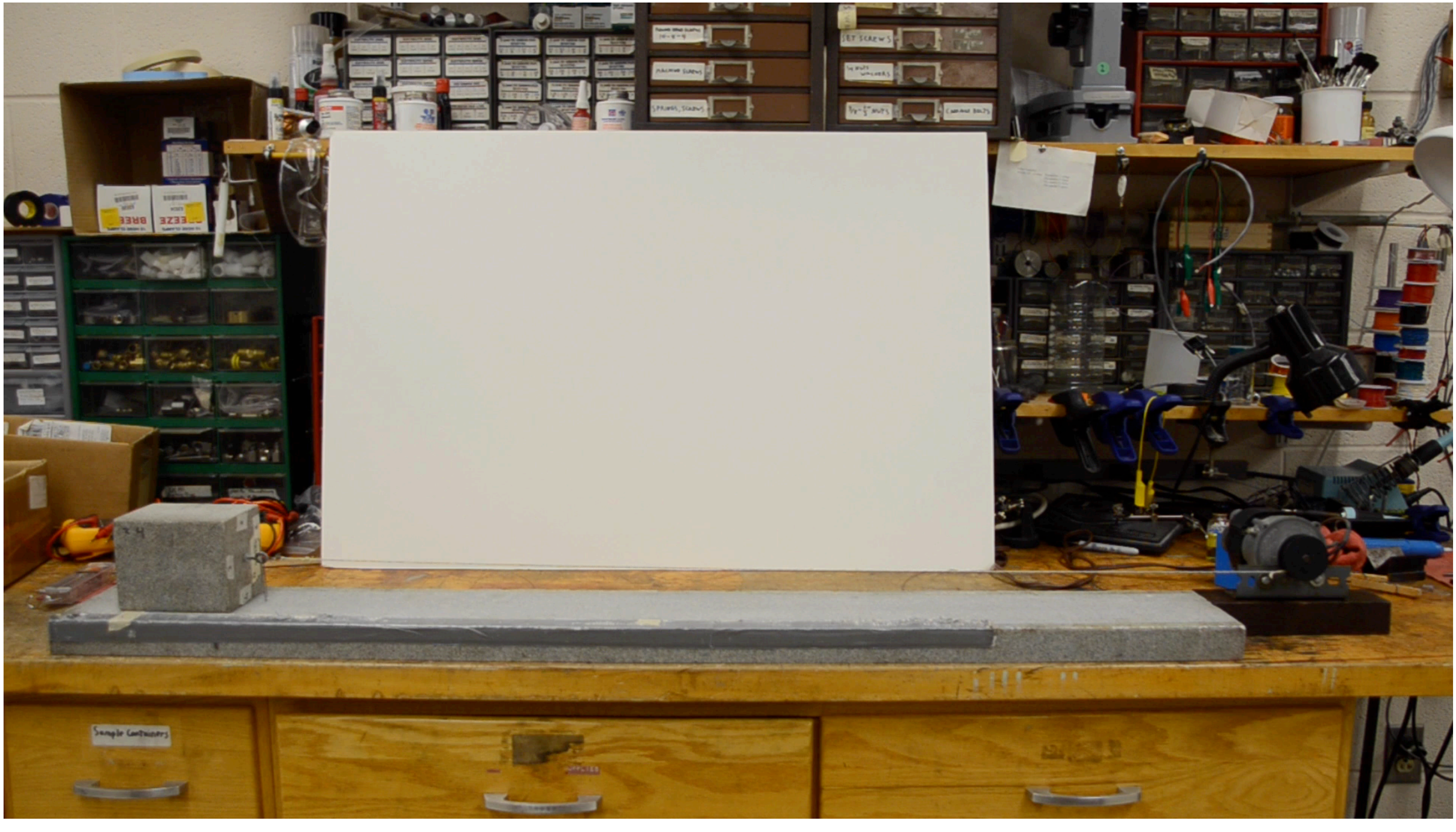
(Patek, 2001)

cimindustry.com

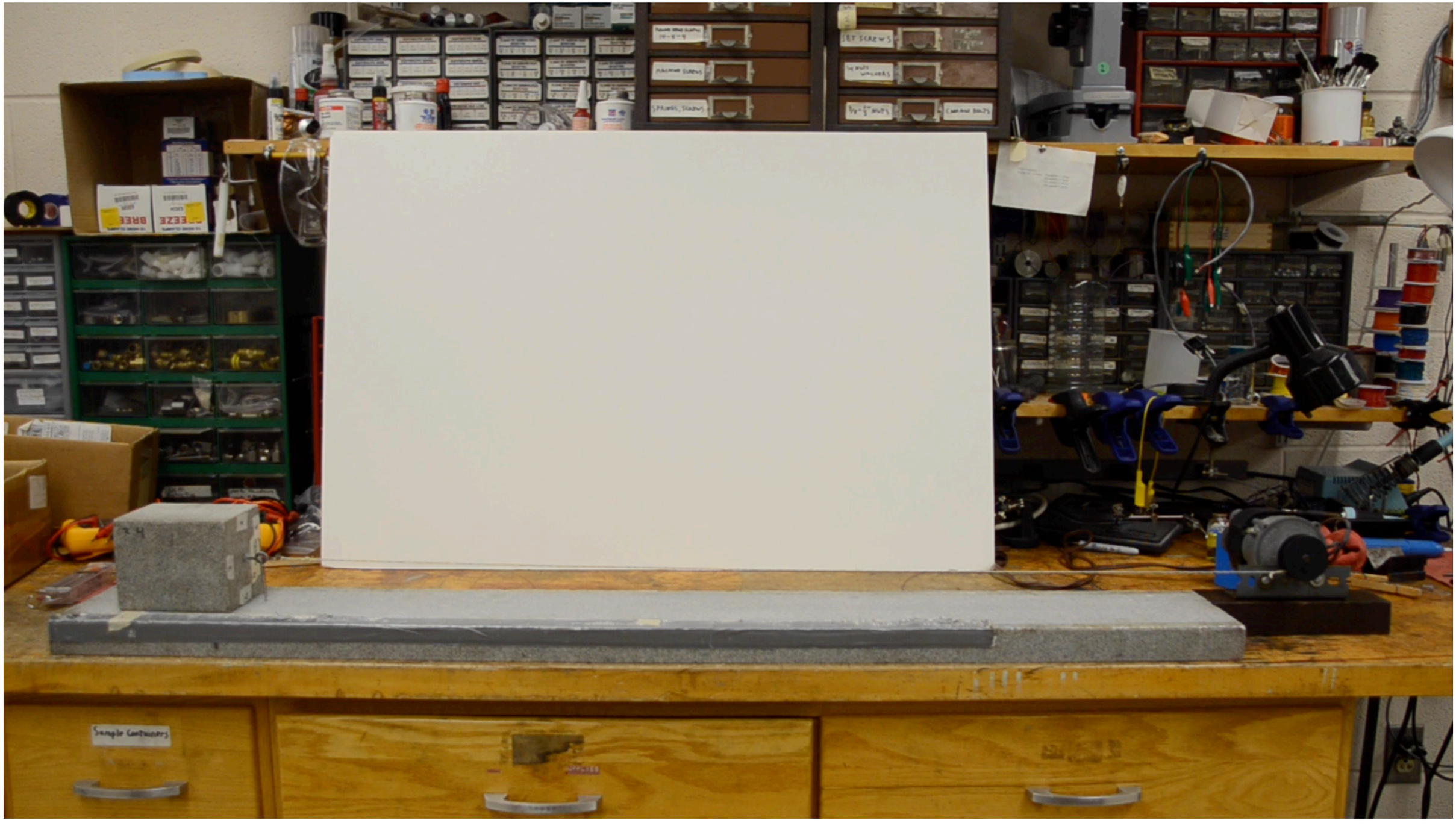
The spring-slider is the simplest physical model



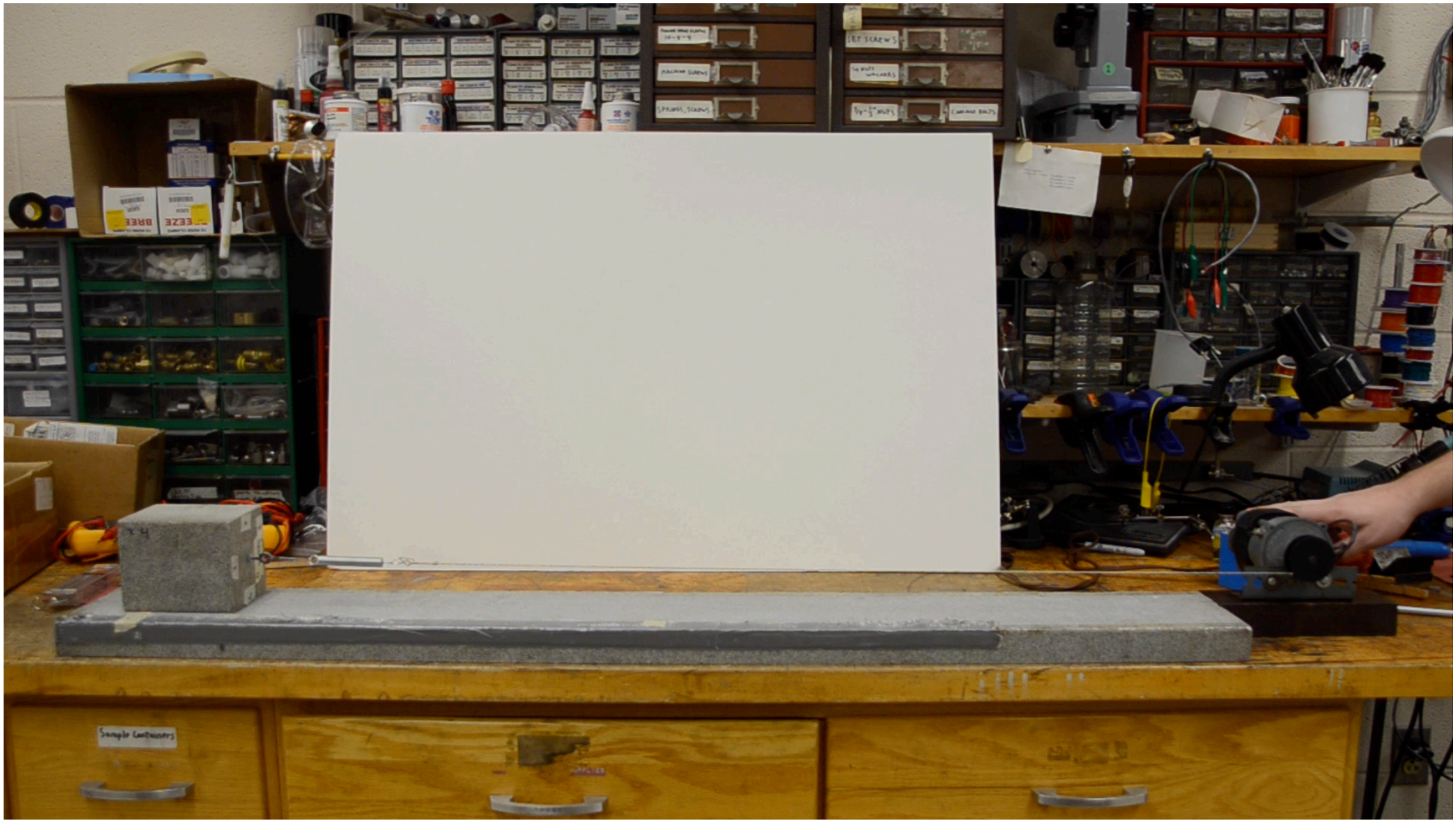
Small changes in stiffness can completely change the behavior of even the simplest system



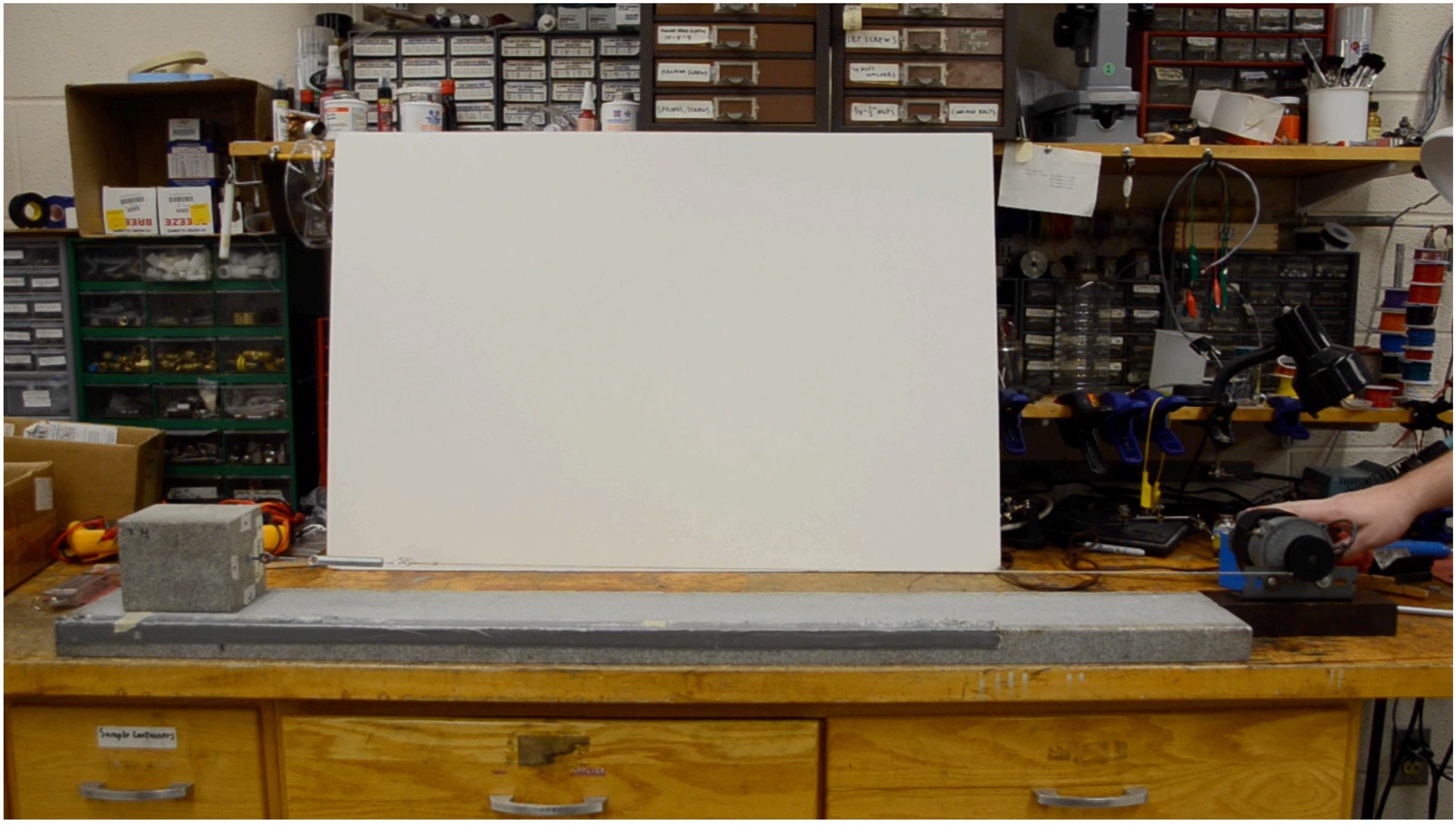
Small changes in stiffness can completely change the behavior of even the simplest system



Small changes in stiffness can completely change the behavior of even the simplest system



Small changes in stiffness can completely change the behavior of even the simplest system



We can model the system with the "rate-and-state" equations

$$\frac{d\mu}{dt} = k\left(V - V_0 \exp\left[\frac{\mu - \mu_0 - b\ln(\frac{V_0\theta}{D_c})}{a}\right]\right)$$

$$\frac{d\mu}{dt} = k(V_{lp} - V)$$

There are many proposed state relations in the wild

Aging

$$\frac{d\theta}{dt} = 1 - \frac{V_{\text{slider}}\theta}{D_c}$$

Slowness

$$\frac{d\theta}{dt} = -\frac{V_{\text{slider}}\theta}{D_c}\ln\left(\frac{V_{\text{slider}}\theta}{D_c}\right)$$

PRZ

$$\frac{d\theta}{dt} = 1 - \left(\frac{V_{\text{slider}}\theta}{2D_c}\right)^2$$

Nagata

$$\frac{d\theta}{dt} = 1 - \frac{V_{\text{slider}}\theta}{D_c} - \frac{c}{b}\theta\frac{d\mu}{dt}$$

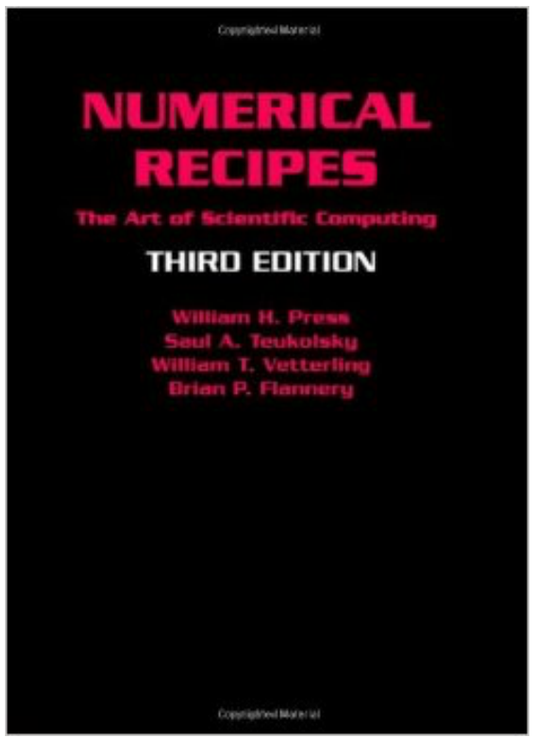
We can create small fault zones in the laboratory

We can create small fault zones in the laboratory

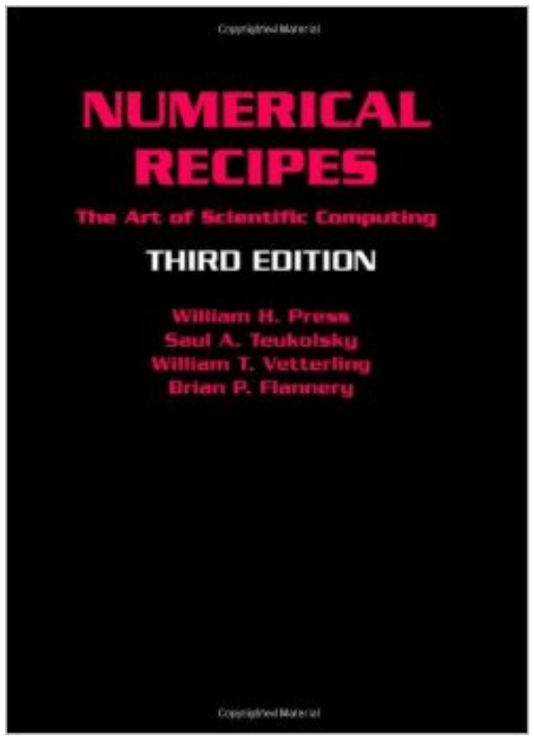
Theory is when you know everything, but nothing works. Practice is when everything works, but no one knows why. In our lab, theory and practice are combined: nothing works and no one knows why.

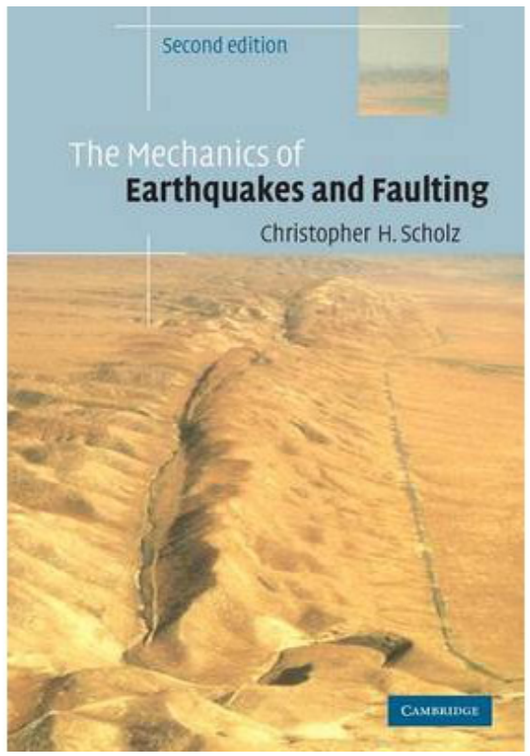
Everyone had their own program to solve the RSF equations

Everyone had their own program to solve the RSF equations

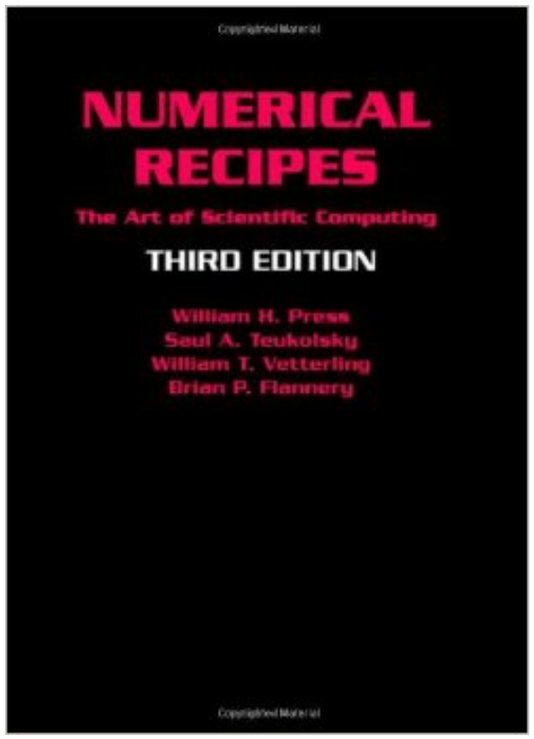


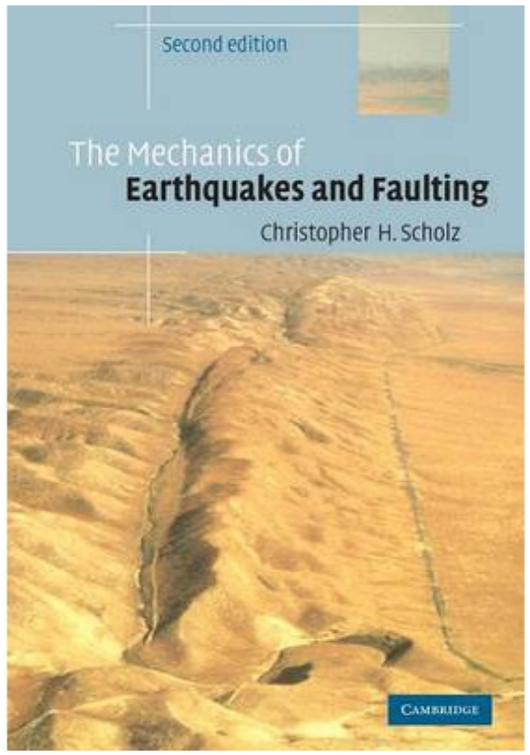
Everyone had their own program to solve the RSF equations



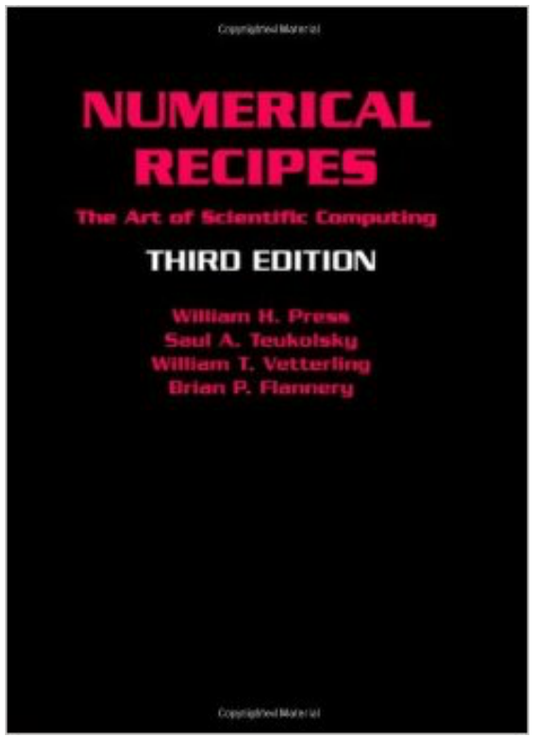


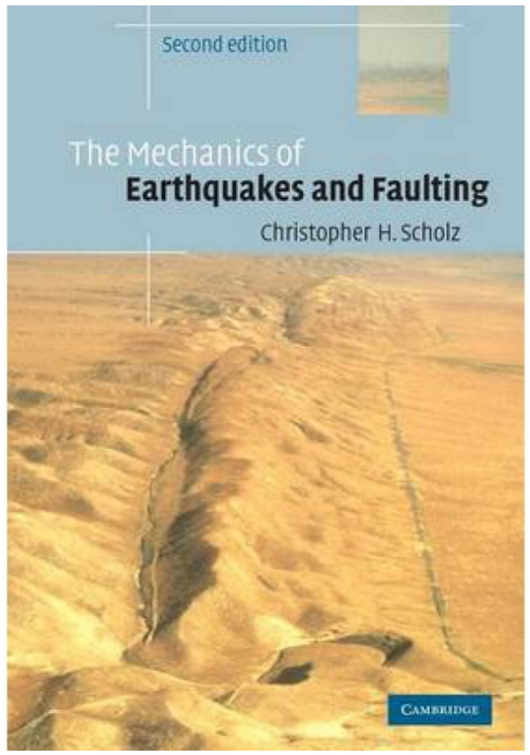
Everyone had their own program to solve the RSF equations



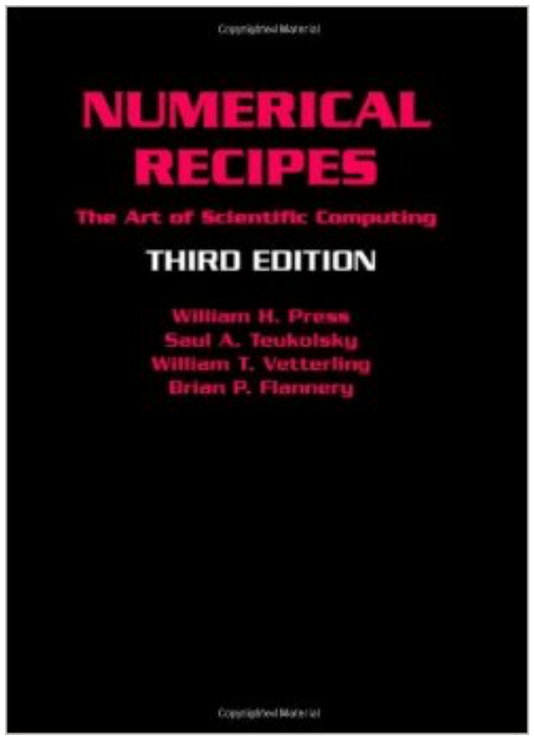


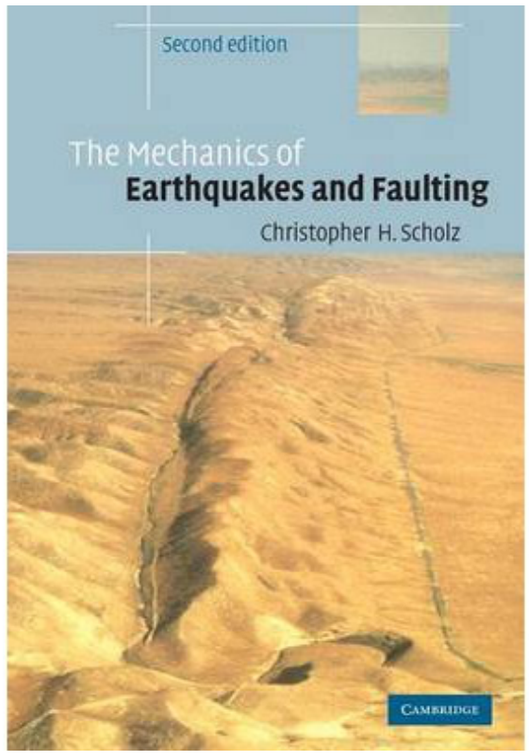
Everyone had their own program to solve the RSF equations





Everyone had their own program to solve the RSF equations



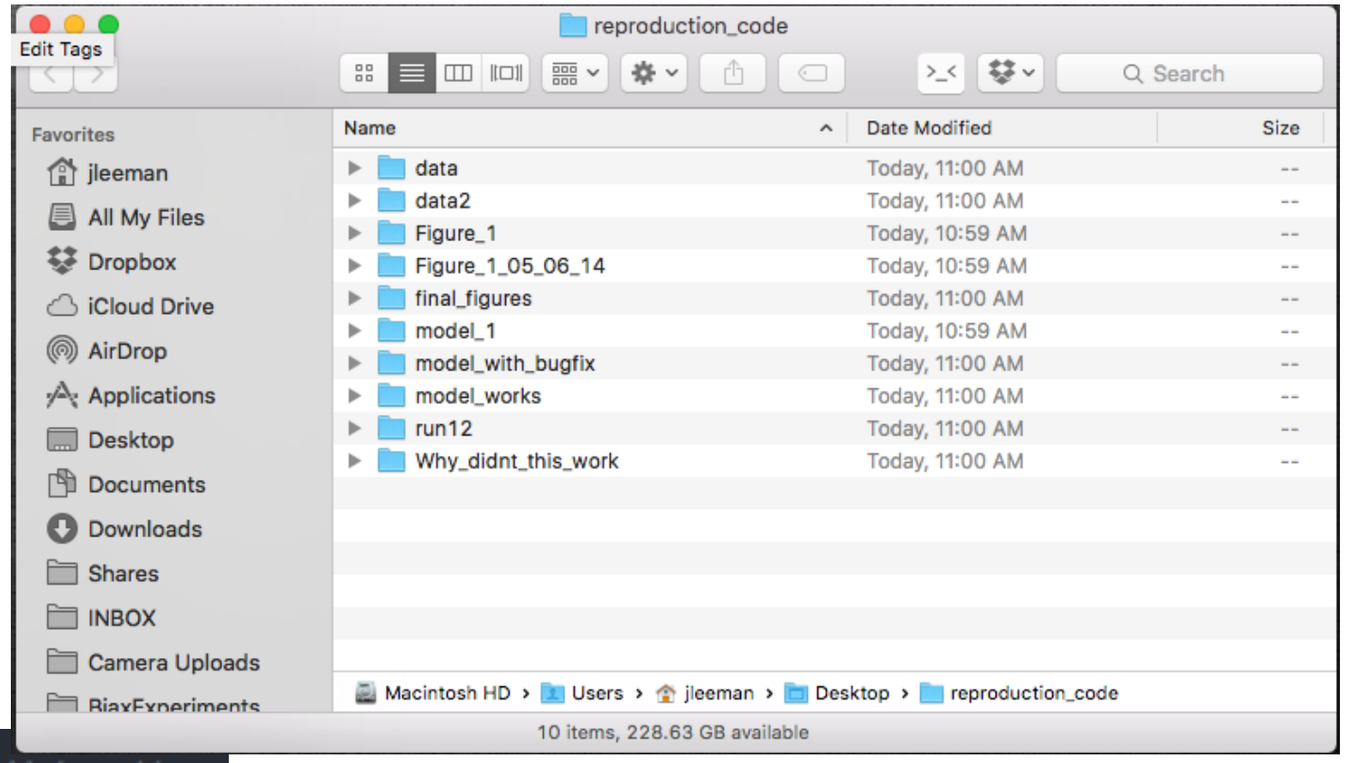


Everyone had their own program to solve the RSF equations

Everyone had their own program to solve the RSF equations

```
switch(rsp->law)       /* set up initial value of state (note I use psi for theta)*/
{           /*  assume steady state for initial v*/
       case 'o':
                psi1 = psi2 = -log(v_s/rsp->v_ref);
   psi_err_scale = fabs(log(v_s/rsp->v_ref));
                break;
       case 'r':
       case 'd':
                psi1 = rsp->dc1/v_s;        /*this will be the same for theta version of r and d, note that  */
                psi2 = rsp->dc2/v_s;      /*  s and S will also use same state */
                break;
       case 'p':
   rsp->vr_dc1 = rsp->v_ref/(TWO*rsp->dc1);    /*modify for Perrin-Rice law*/
   rsp->vr_dc2 = rsp->v_ref/(TWO*rsp->dc2);    /*note: this is set above as rsp.vr_dc..*/
                psi1 = (TWO*rsp->dc1)/v_s;
                psi2 = (TWO*rsp->dc2)/v_s;
                break;
       case 'j':                                        /*check this later, just a guess now 15/7/94 */
   fprintf(stderr,"rice law not implemented\n");
   return(-1);
                break;
}
if(rsp->one_sv_flag)
 rsp->vr_dc2 = SMALL_NUM;  /*b2 is zero for 1sv model, set dc2 to very large number*/

for(i=1;i<=(rsp->vs_row-rsp->first_row);i++) /*fill with prejump mu*/
{
 mod_mu[i] = mu_s+rsp->lin_term*(x[i]-x[(rsp->vs_row-rsp->first_row)]);
 v_slider[i] = v_s;
 state[i] = psi1;
 sd[i]=0.0;
 phi[i] = phi_ref - epsilon*log(psi1*rsp->vr_dc1);
}

i=(rsp->vs_row-rsp->first_row+1);    /*first point after vs_row*/
```

If you can get those programs and try to reproduce the results, it often ends in tears

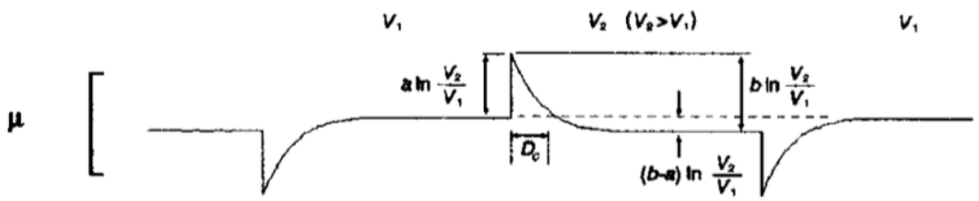



```
140  # Moduli from Dvorkin 1999, density from wikipedia
141  feldspar = Componet('Feldspar','Solid')
142  feldspar.volume_percent = 0.26
143  feldspar.bulk_modulus = 76e9
144  feldspar.shear_modulus = 26e9
145  feldspar.density = 2560
146
147  ## ONLY Works with this commented out?
148  # model.critical_porosity = 0.34
149
150  model = MediumModel([quartz, clay, feldspar])
151  model.critical_porosity = 0.4
152  model.n = 8
```

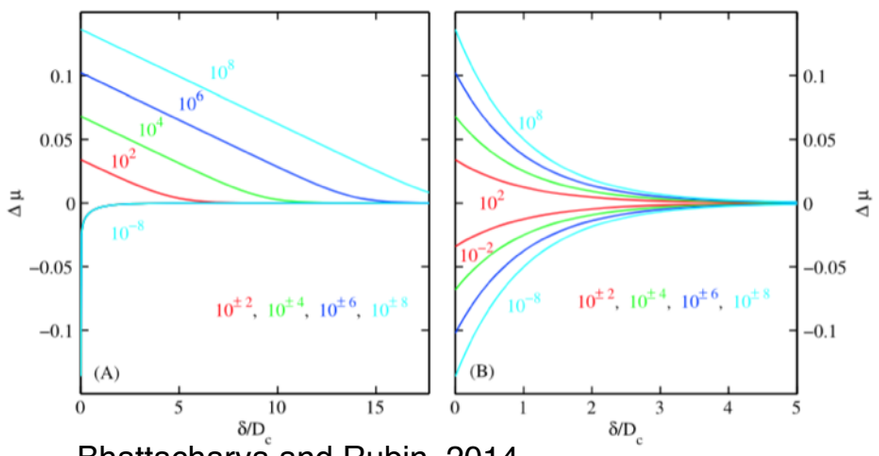
If you can get those programs and try to reproduce the results, it often ends in tears


```
140   # Modul
141   feldspa
142   feldspa
143   feldspa
144   feldspa
145   feldspa
146
147   ## ONLY
148   # model.
149
150   model =
151   model.c
152   model.n = 8
```

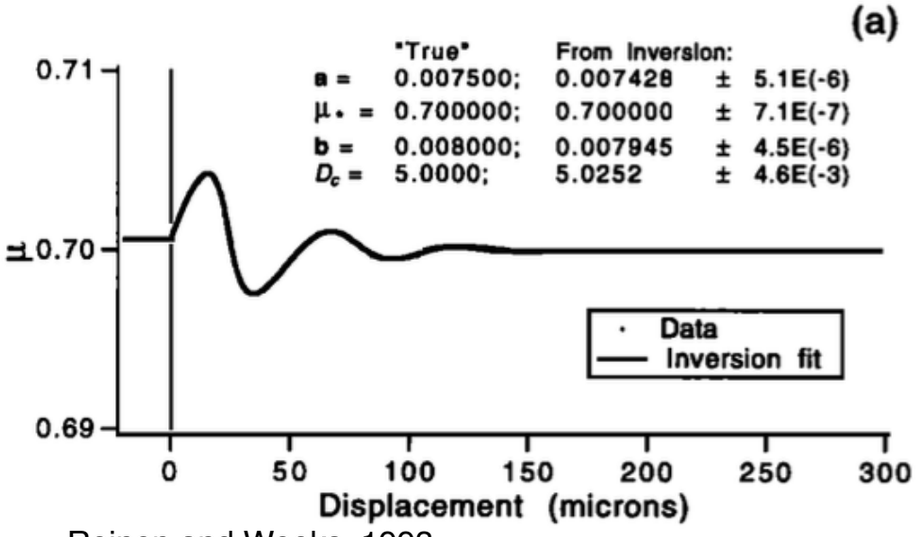
The results often looked “right” or “comparable”



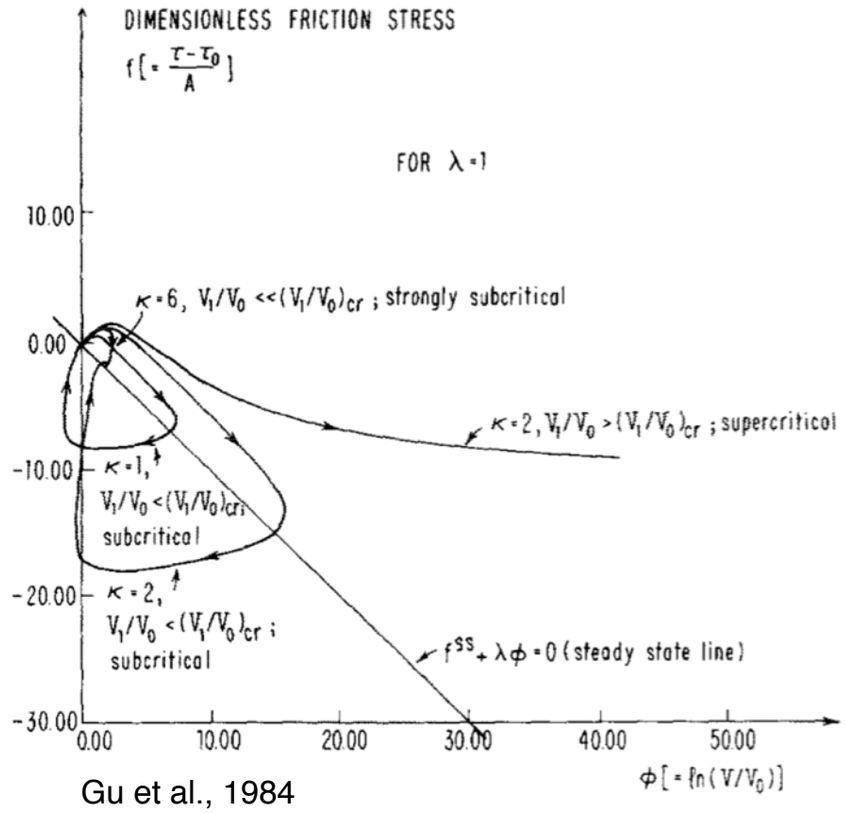
Marone, 1998



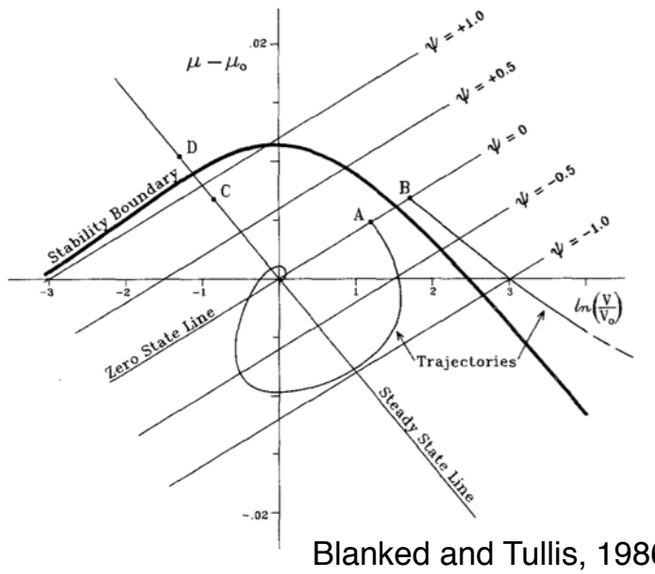
Bhattacharya and Rubin, 2014



Reinen and Weeks, 1993

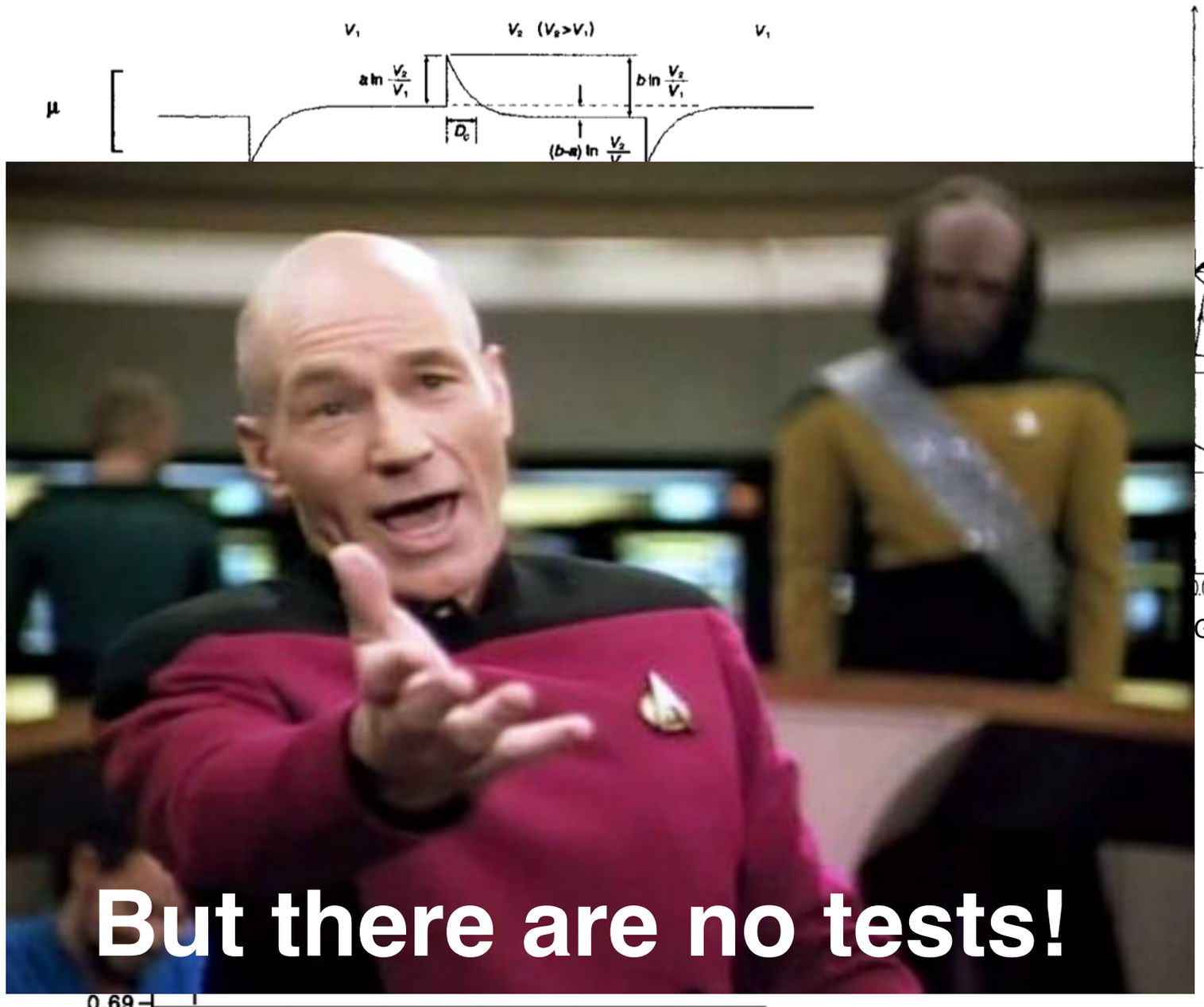


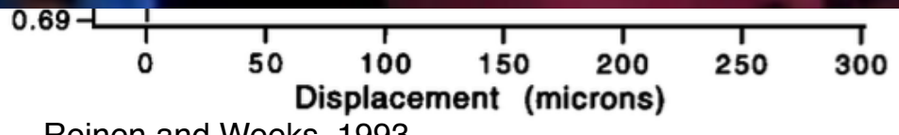
Gu et al., 1984



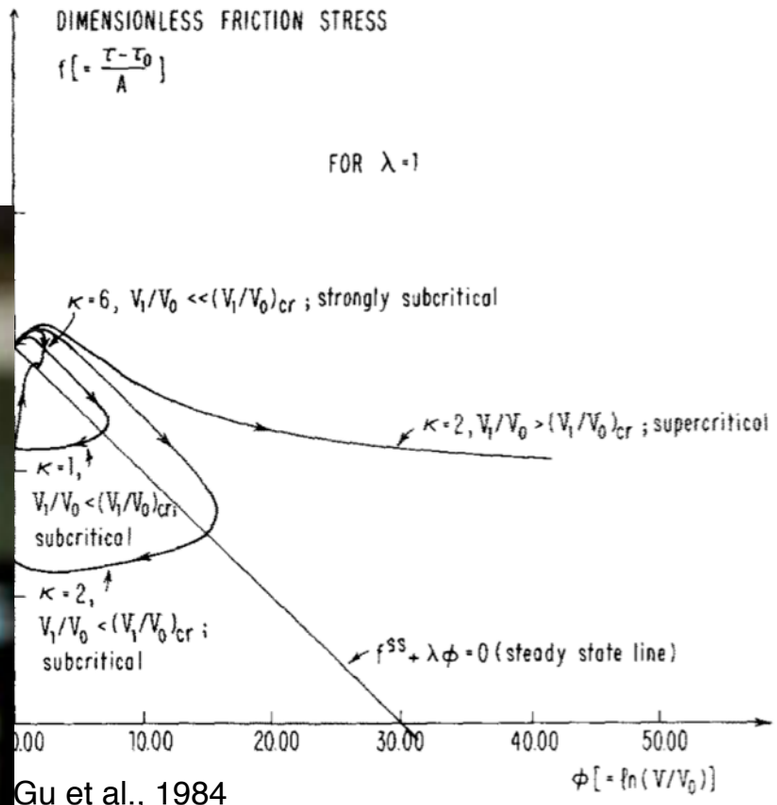
Blanked and Tullis, 1986

The results often looked “right” or “comparable”

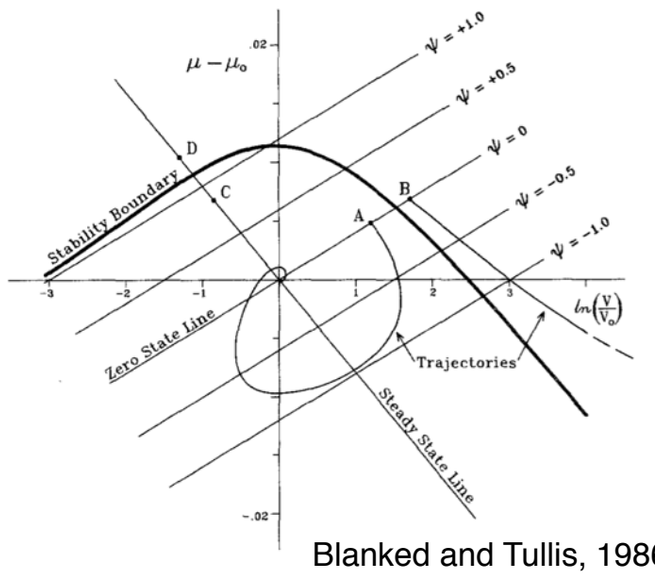




Reinen and Weeks, 1993



Gu et al., 1984



Blanked and Tullis, 1986

The tools were not friendly to students either

```
This program solves the rate and state friction laws with 1-d elastic interaction. One or two state variables can be used.

Usage: ./rsfs op_file law v_init v_load hold_time op_inc_hold op_inc_disp max_disp k a b1 Dc1 b2 dc2 -svtd
where law is: d, r, p, or s
       d=Dieterich, slowness law, r=Runia, slip law,
       p=Perrin-Rice, quadratic law, s=Segall and Rice eq'n 17
-t = output time vs mu
-v = output ln(v_s/v_init) vs mu
-s = output time vs state(theta)
-d = output Slider_Displacement vs mu
-f = output time vs porosity
-p = stop calculation at peak friction on reload, write a data point there
-r = set ref fric to 0.6 at 1 mic/s, rather than to arb. value to give 0.6 as initial value

LoadPointDisplacement vs mu is always output, and lp_dis vs. porosity is output if s law is used
Command line definitions:
       v_init is the initial, steady-state velocity prior to the hold
       v_load is the reload velocity following the hold
       hold_time is the hold time in seconds
       op_inc_hold is the time increment for output during the hold
       op_inc_disp is the disp increment for output following the hold
       disp is the final disp for the calc, following the hold
       To do a 1 state variable model, set dc2 < 0

       To do a vel step, set hold_time to a small number and op_inc_hold < hold_time, and use different v_init and v_load
       Note: k should  have dimensions of 1/length


One or more ASCII text files are written as output. Each row has two columns (time, displacement, friction, etc --as chosen by command line options summarized above) and the output frequency is specified by the co
mmand line parameters op_inc_hold and op_inc_disp

       Two example command lines are:

rsfs junk d 1 1 100 .1 .1 200 1e-3 0.005 0.01 10 1 -10 -t

rsfs junk d 1 10 1e-7 5e-8 0.1 300 1e-3 0.005 0.01 10 1 -10 -t

       The first example specifies a slide-hold-slide test with one state variable. The output file name is junk, the dieterich state evolution law is used, the initial and re-load velocities are 1 micron/s, hold
 time is 100 sec, output is written every 0.1 sec during the hold and 0.1 micron after the hold up to a maximum of 200 microns, stiffness is 0.001 (friction) per micron, and constitutive parameters are a=0.005, b=
0.01 and Dc of 10 microns. Note that there is nothing intrinsic about the units; the length unit could just as well be taken as meters and the time as hours. The only requirement is that all units be consistent. T
he files junk.dis and junk.tim are output. Each file contains a one line header and then x,y pairs of: junk.dis: load point displacement and friction junk.tim time and friction.

       The second example specifies a velocity step test from 1 mic/s to 10 mic/s. There are two data points written during the 'hold' which lasts only 1e-7 seconds (these are irrelevant to the rest of the calcul
ation) and then data are written every 0.1 micron up to a maximum of 300 microns. The friction parameters and output files are the same as the first example.
```

The tools were not friendly to students either

```
This program solves the rate and state friction laws with 1-d elastic interaction. One or two state variables can be used.

Usage: ./rsfs op_file law v_init v_load hold_time op_inc_hold op_inc_disp max_disp k a b1 Dc1 b2 dc2 -svtd
where law is: d, r, p, or s
       d=Dieterich, slowness law, r=Runia, slip law,
       p=Perrin-Rice, quadratic law, s=Segall and Rice eq'n 17
-t = output time vs mu
-v = output ln(v_s/v_init) vs mu
-s = output time vs state(theta)
-d = output Slider_Displacement vs mu
-f = output time vs porosity
-p = stop calculation at peak friction on reload, write a data point there
-r = set ref fric to 0.6 at 1 mic/s, rather than to arb. value to give 0.6 as initial value

LoadPointDisplacement vs mu is always output, and lp_dis vs. porosity is output if s law is used
Command line definitions:
       v_init is the initial, steady-state velocity prior to the hold
       v_load is the reload velocity following the hold
       hold_time is the hold time in seconds
       op_inc_hold is the time increment for output during the hold
       op_inc_disp is the disp increment for output following the hold
       disp is the final disp for the calc, following the hold
       To do a 1 state variable model, set dc2 < 0

       To do a vel step, set hold_time to a small number and op_inc_hold < hold_time, and use different v_init and v_load
       Note: k should  have dimensions of 1/length


One or more ASCII text files are written as output. Each row has two columns (time, displacement, friction, etc --as chosen by command line options summarized above) and the output frequency is specified by the co
mmand line parameters op_inc_hold and op_inc_disp

       Two example command lines are:

rsfs junk d 1 1 100 .1 .1 200 1e-3 0.005 0.01 10 1 -10 -t

rsfs junk d 1 10 1e-7 5e-8 0.1 300 1e-3 0.005 0.01 10 1 -10 -t

       The first example specifies a slide-hold-slide test with one state variable. The output file name is junk, the dieterich state evolution law is used, the initial and re-load velocities are 1 micron/s, hold
 time is 100 sec, output is written every 0.1 sec during the hold and 0.1 micron after the hold up to a maximum of 200 microns, stiffness is 0.001 (friction) per micron, and constitutive parameters are a=0.005, b=
0.01 and Dc of 10 microns. Note that there is nothing intrinsic about the units; the length unit could just as well be taken as meters and the time as hours. The only requirement is that all units be consistent. T
he files junk.dis and junk.tim are output. Each file contains a one line header and then x,y pairs of: junk.dis: load point displacement and friction junk.tim time and friction.

       The second example specifies a velocity step test from 1 mic/s to 10 mic/s. There are two data points written during the 'hold' which lasts only 1e-7 seconds (these are irrelevant to the rest of the calcul
ation) and then data are written every 0.1 micron up to a maximum of 300 microns. The friction parameters and output files are the same as the first example.
```

The requirements were simple:

The requirements were simple:

Research

The requirements were simple:

Research

Teaching

The requirements were simple:

Research

Teaching

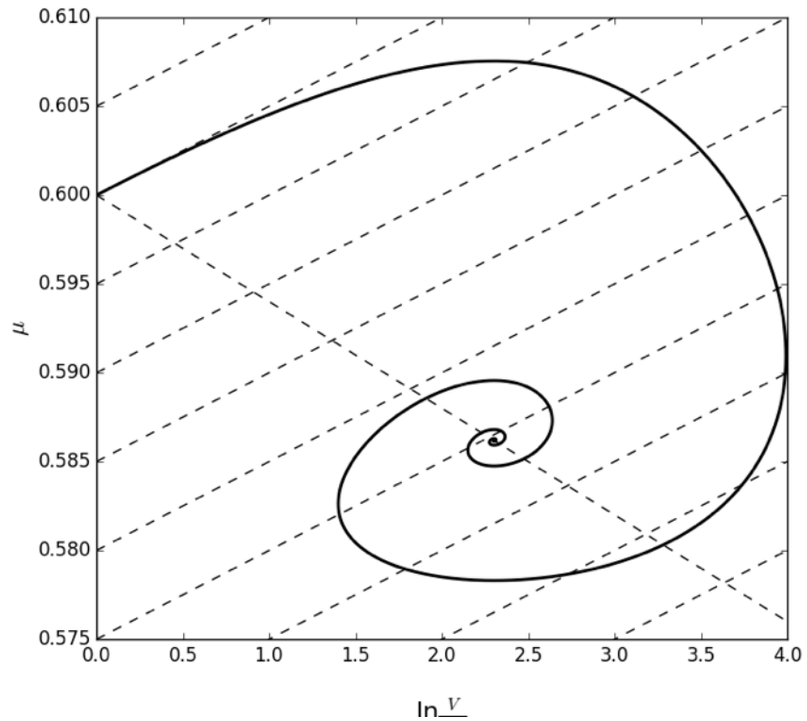
Best Practices

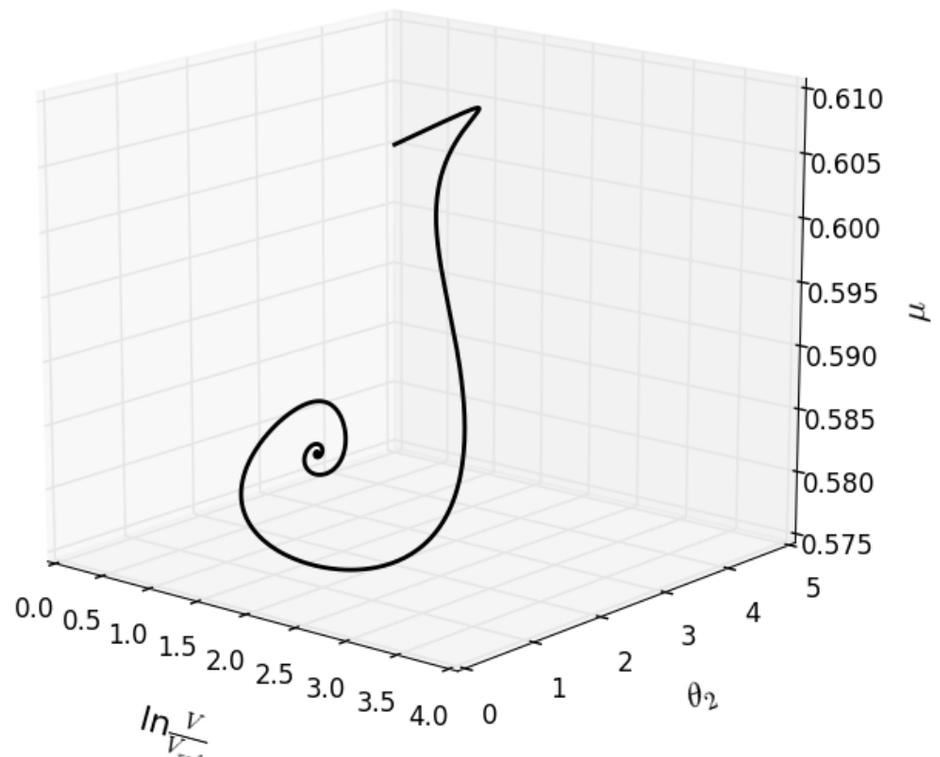
Include the common state relations and elastic coupling

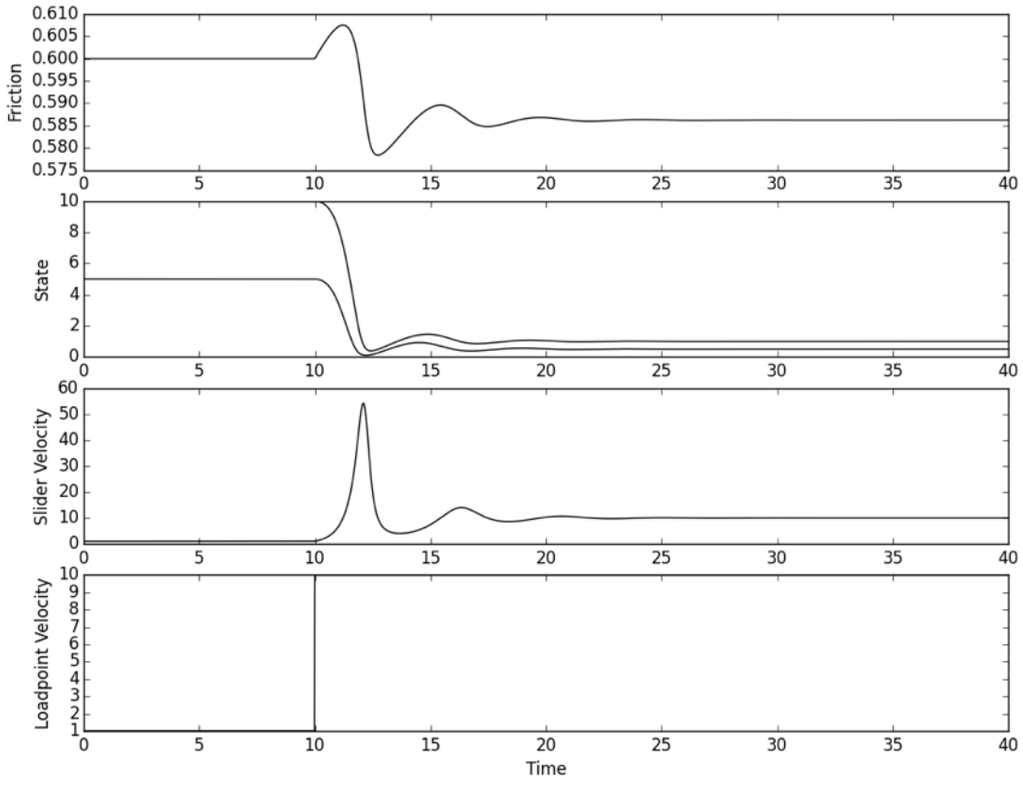
```
model = rsf.Model()

# Set model initial conditions
model.mu0 = 0.6 # Friction initial (at the reference velocity)
model.a = 0.005 # Empirical coefficient for the direct effect
model.k = 1e-3 # Normalized System stiffness (friction/micron)
model.v = 1. # Initial slider velocity, generally is vlp(t=0)
model.vref = 1. # Reference velocity, generally vlp(t=0)

state1 = staterelations.DieterichState()
state1.b = 0.01  # Empirical coefficient for the evolution effect
state1.Dc = 10.  # Critical slip distance

model.state_relations = [state1] # Which state relation we want to use

# We want to solve for 40 seconds at 100Hz
model.time = np.arange(0,40.01,0.01)

# We want to slide at 1 um/s for 10 s, then at 10 um/s for 31
lp_velocity = np.ones_like(model.time)
lp_velocity[10*100:] = 10. # Velocity after 10 seconds is 10 um/s

# Set the model load point velocity, must be same shape as model.model_time
model.loadpoint_velocity = lp_velocity

# Run the model!
model.solve()
```

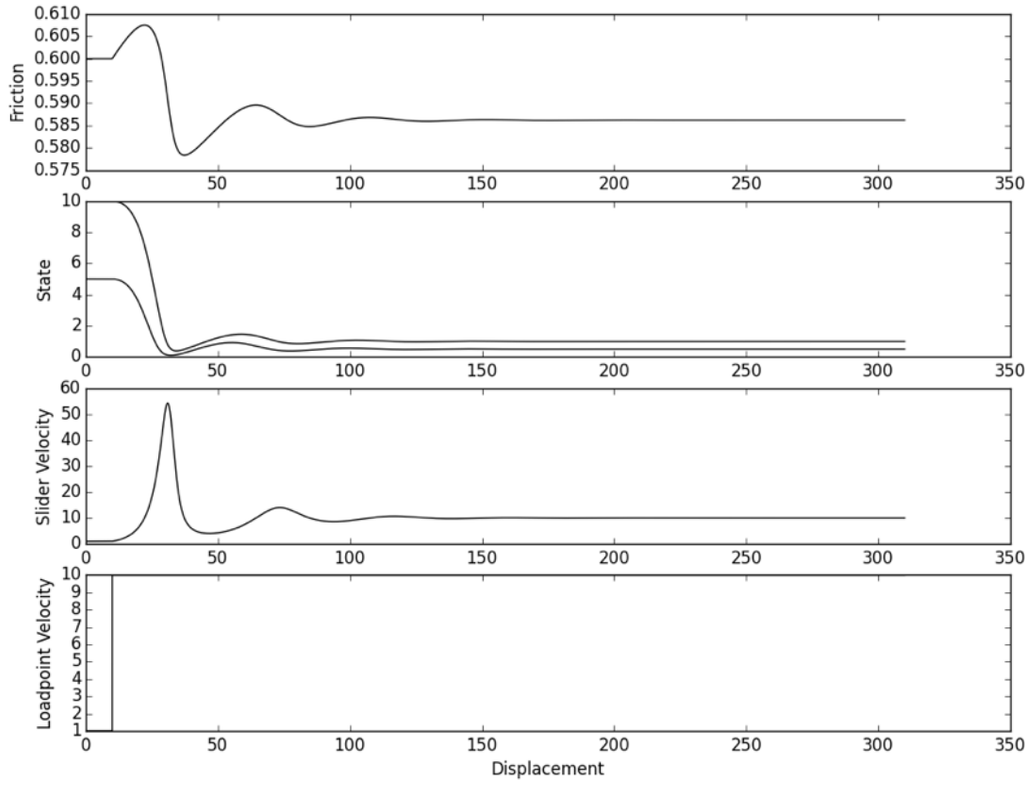
Easily make “standard” plots

```
# Make the phase plot
plot.phasePlot(model)

# Make a plot in displacement
plot.dispPlot(model)

# Make a plot in time
plot.timePlot(model)
```









Allow users to define their own rules

Allow users to define their own rules

Define it:

```
class MyStateRelation(staterelations.StateRelation):
    # Need to provide a steady state calcualtion method
    def set_steady_state(self, system):
        self.state = self.Dc/system.vref

    def evolve_state(self, system):
        return -1 * (system.v * self.state / self.Dc) * log(system.v * self.state / self.Dc)
```

Allow users to define their own rules

Define it:

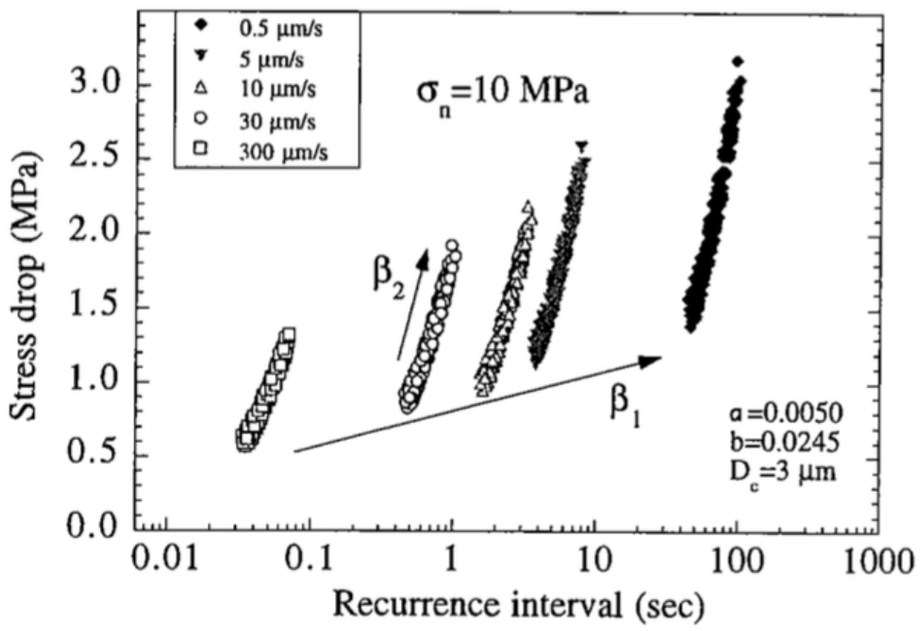
```
class MyStateRelation(staterelations.StateRelation):
    # Need to provide a steady state calcualtion method
    def set_steady_state(self, system):
        self.state = self.Dc/system.vref

    def evolve_state(self, system):
        return -1 * (system.v * self.state / self.Dc) * log(system.v * self.state / self.Dc)
```

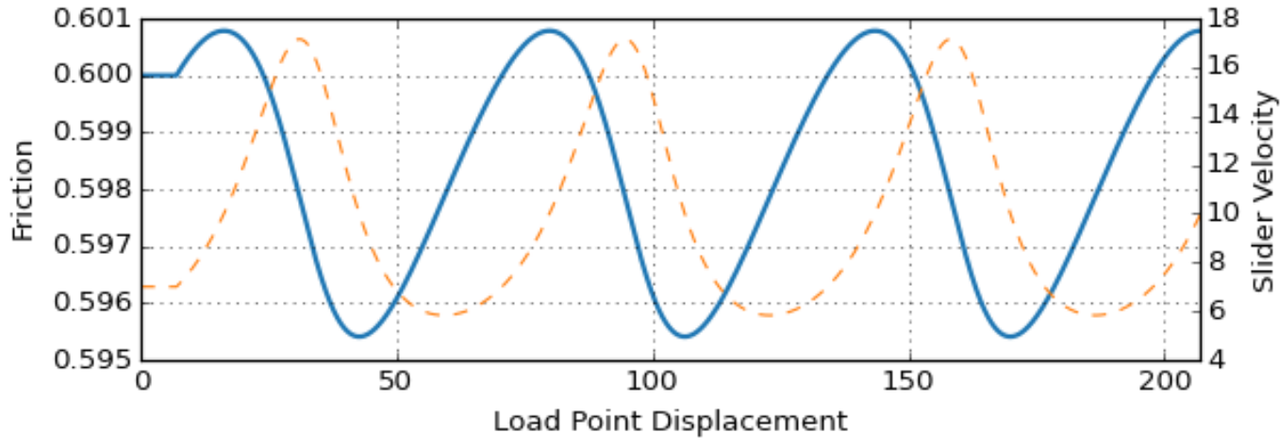
Use it:

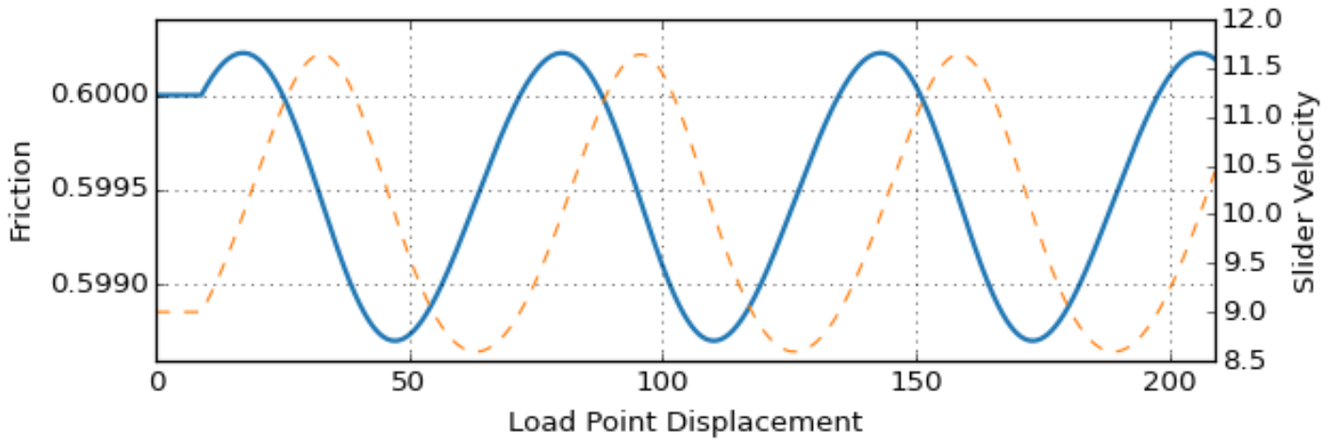
```
state1 = MyStateRelation()
state1.b = 0.005  # Empirical coefficient for the evolution effect
state1.Dc = 10.  # Critical slip distance

model.state_relations = [state1]  # Which state relation we want to use
```

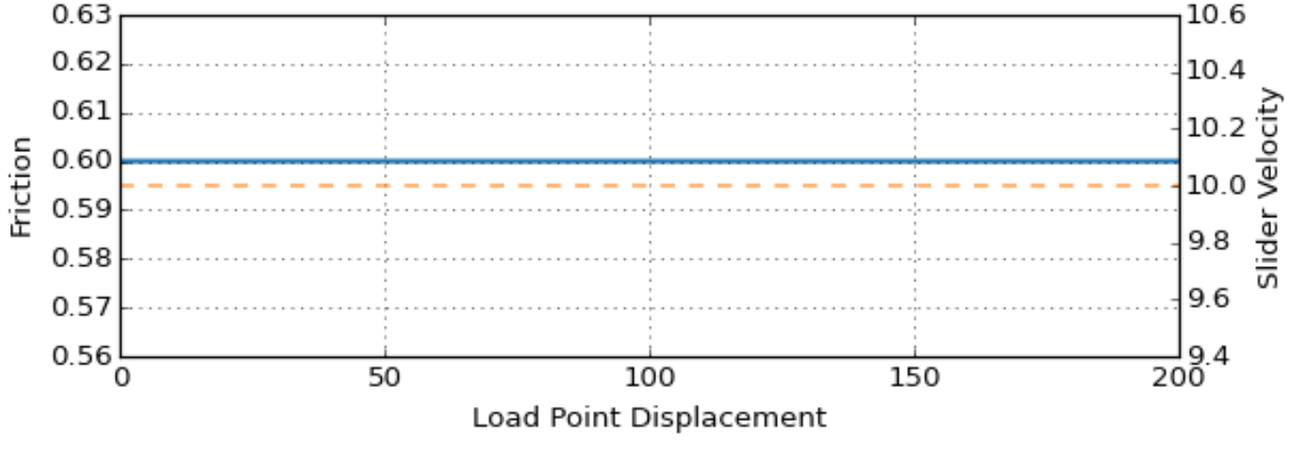
Everything is tested. Against itself which has been compared to another model.

We found that some “well established” results were functions of how the model was setup

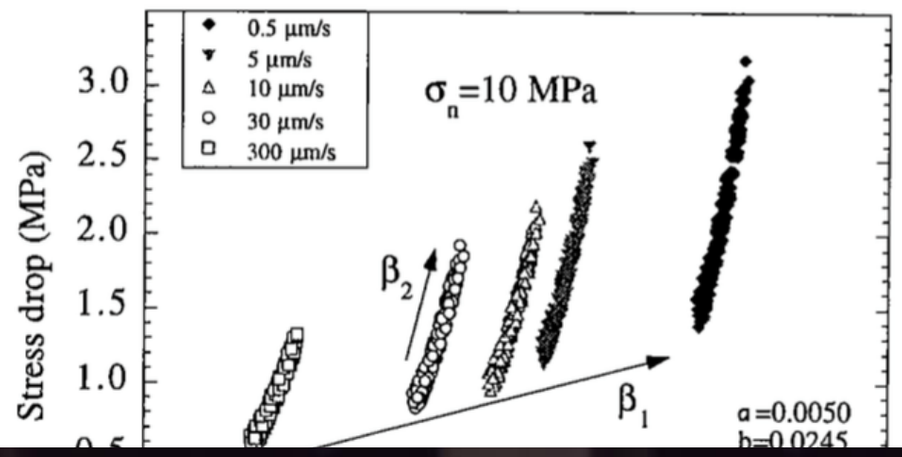


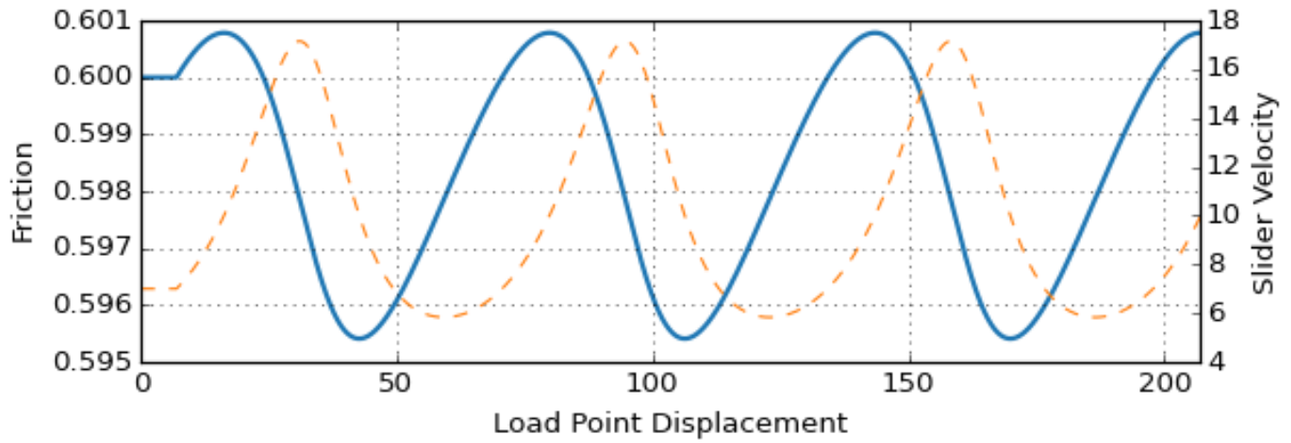


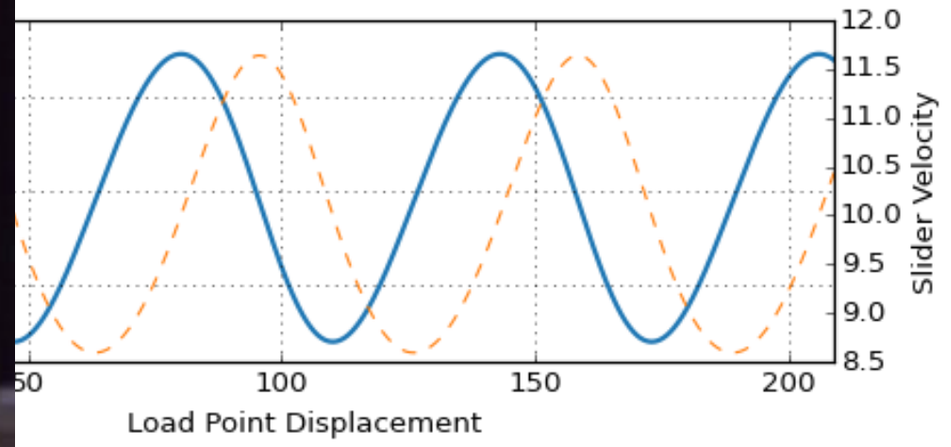


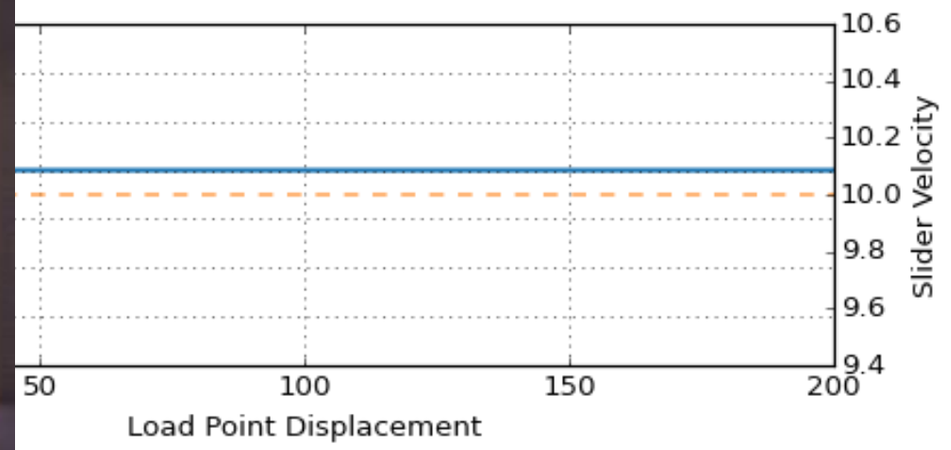


We found that some “well established” results were functions of how the model was setup







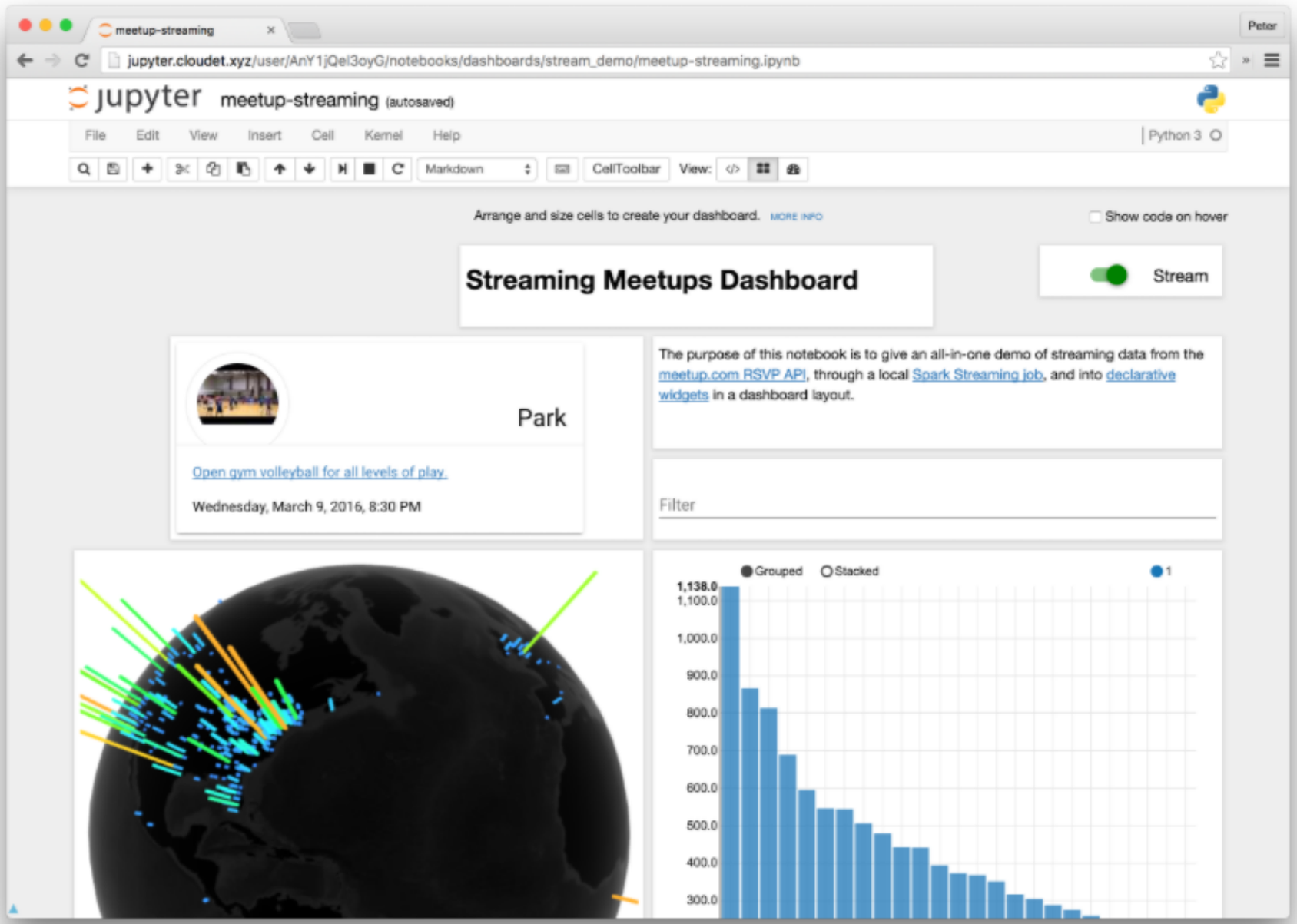


Dashboards make a great way to deploy for student interaction

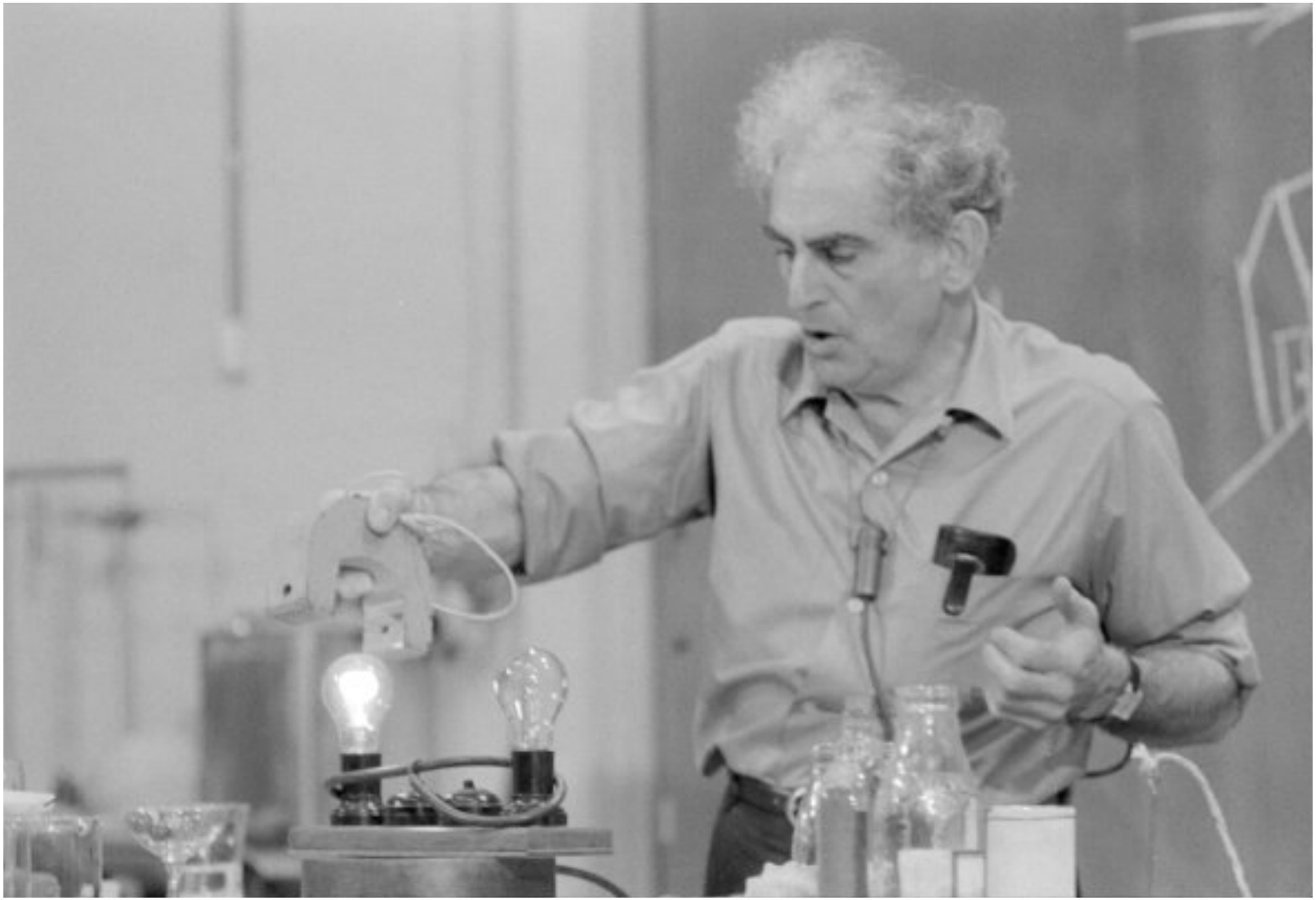
pypi package 0.6.0 | build passing | Google Group

Jupyter Dashboards Layout

Extension for Jupyter Notebook that enables the layout and presentation of dashboards from notebooks.



Give it a try! There are many potential applications: rsfmodel.com



National Library of Australia