Because you're a nice person

Library UX

Using abstraction towards friendlier APIs

Mali Akmanalp
(@makmanalp)

bit.ly/abstraction-talk

(@makmanalp)

UX

== User Experience

== How this makes me feel

UX

== User Experience

== Usability

```python
import urllib2

gh_url = 'https://api.github.com/user'

req = urllib2.Request(gh_url)

password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(None, gh_url, 'user', 'pass')

auth_manager = urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_manager)

urllib2.install_opener(opener)

handler = urllib2.urlopen(req)

print handler.read()
```

https://www.kennethreitz.org/python-for-humans/

```python
import requests

url = 'https://api.github.com/user'
auth = ('username', 'password')

r = requests.get(url, auth=auth)
print r.content
```

https://www.kennethreitz.org/python-for-humans/

“For Humans”

Why care about UX?

Good UX
reduces mistakes.

```python
import urllib2

gh_url = 'https://api.github.com/user'

req = urllib2.Request(gh_url)

password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(None, gh_url, 'user', 'pass')

auth_manager = urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_manager)

urllib2.install_opener(opener)

handler = urllib2.urlopen(req)

print handler.read()
```

```python
import urllib2

gh_url = 'https://api.github.com/user'

req = urllib2.Request(gh_url)

password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(None, gh_url, 'user', 'pass')

auth_manager = urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_manager)

urllib2.install_opener(opener)

handler = urllib2.urlopen(req)

print handler.read()
```

```python
import urllib2

gh_url = 'https://api.github.com/user'

req = urllib2.Request(gh_url)

password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(None, gh_url, 'user', 'pass')

auth_manager = urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_manager)

urllib2.install_opener(opener)

handler = urllib2.urlopen(req)

print handler.read()
```

```python
import urllib2

gh_url = 'https://api.github.com/user'

req = urllib2.Request(gh_url)

password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(None, gh_url, 'user', 'pass')

auth_manager = urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_manager)

urllib2.install_opener(opener)

handler = urllib2.urlopen(req)

print handler.read()
```

Good UX
minimizes distractions.

Good UX
makes complex tasks routine.

Good UX
drives adoption.

django

Flask
web development,
one drop at a time

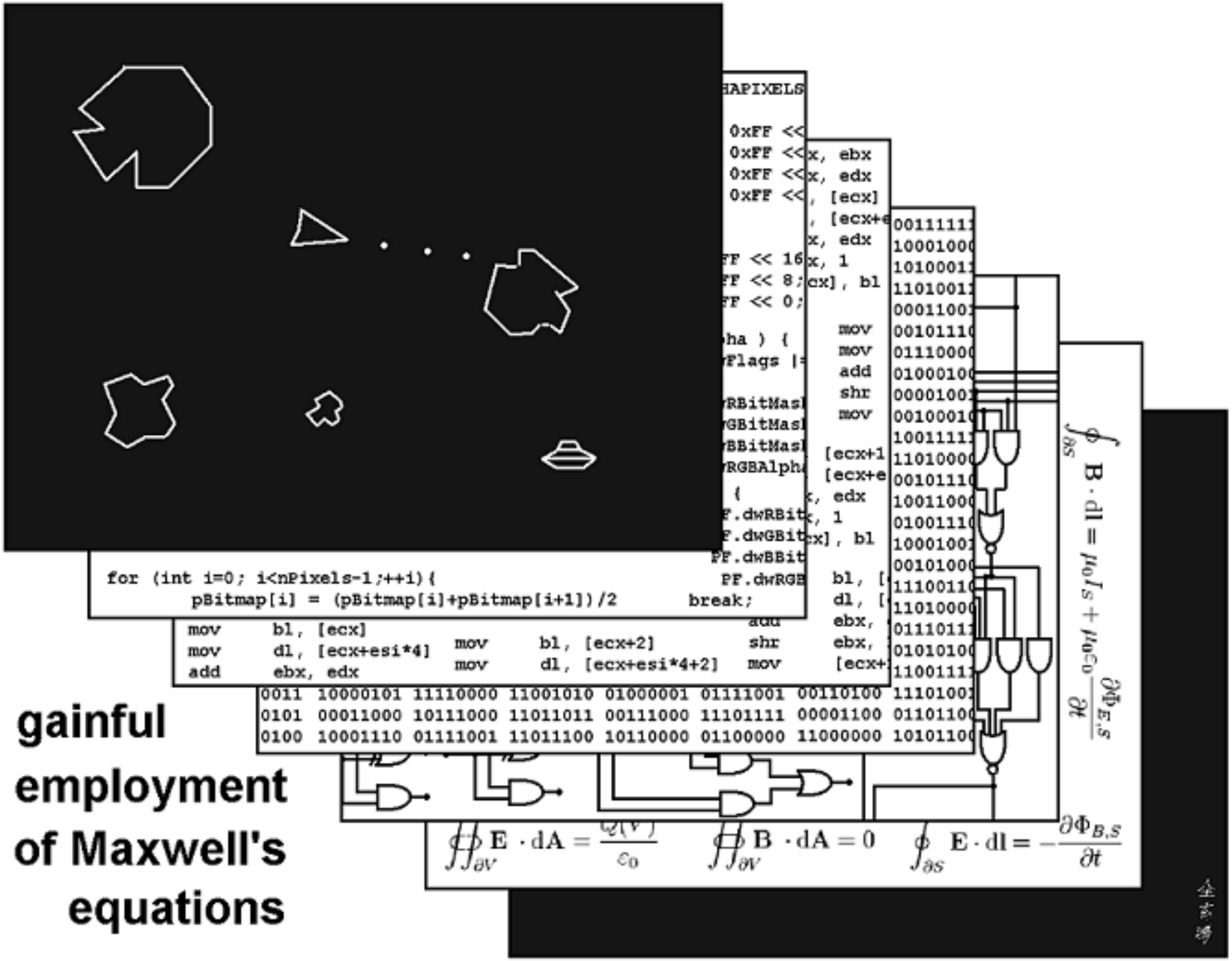
K

Requests
http for humans

gensim

TextBlob

?


for (int i=0; i<nPixels-1;++i){
pBitmap[i] = (pBitmap[i]+pBitmap[i+1])/2
break;
$\oint_{\partial S} \mathbf{B} \cdot d\mathbf{l} = \mu_0 I_S + \mu_0 \varepsilon_0 \frac{\partial \Phi_{E,S}}{\partial t}$
$\oiint_{\partial V} \mathbf{E} \cdot d\mathbf{A} = \frac{Q(V)}{\varepsilon_0}$
$\oiint_{\partial V} \mathbf{B} \cdot d\mathbf{A} = 0$
$\oint_{\partial S} \mathbf{E} \cdot d\mathbf{l} = -\frac{\partial \Phi_{B,S}}{\partial t}$

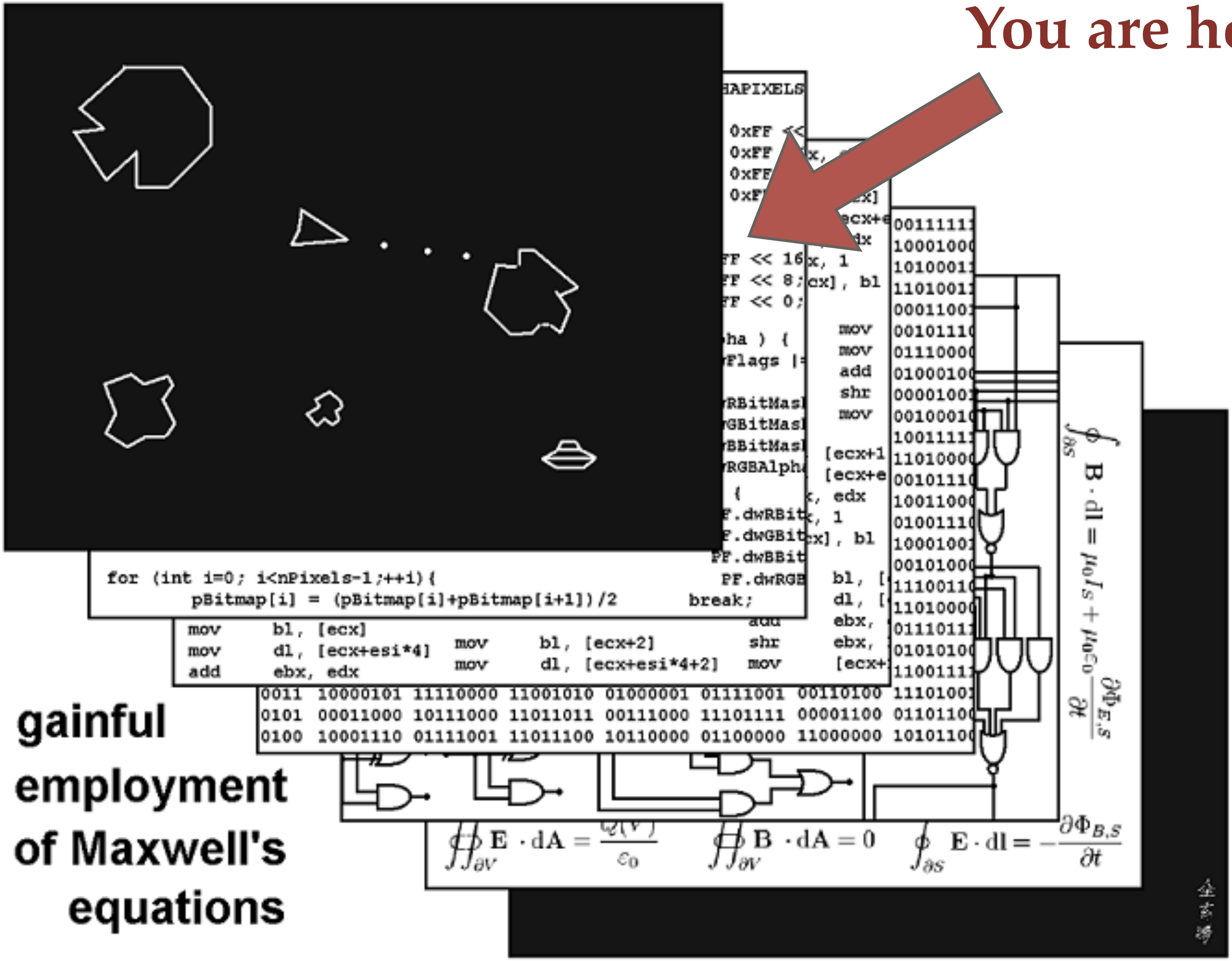
gainful employment of Maxwell's equations

http://abstrusegoose.com/307

Claim:

We are primarily in the business of dealing with abstractions.

You are here



http://abstrusegoose.com/307

Abstraction is about hiding details in a controlled way.

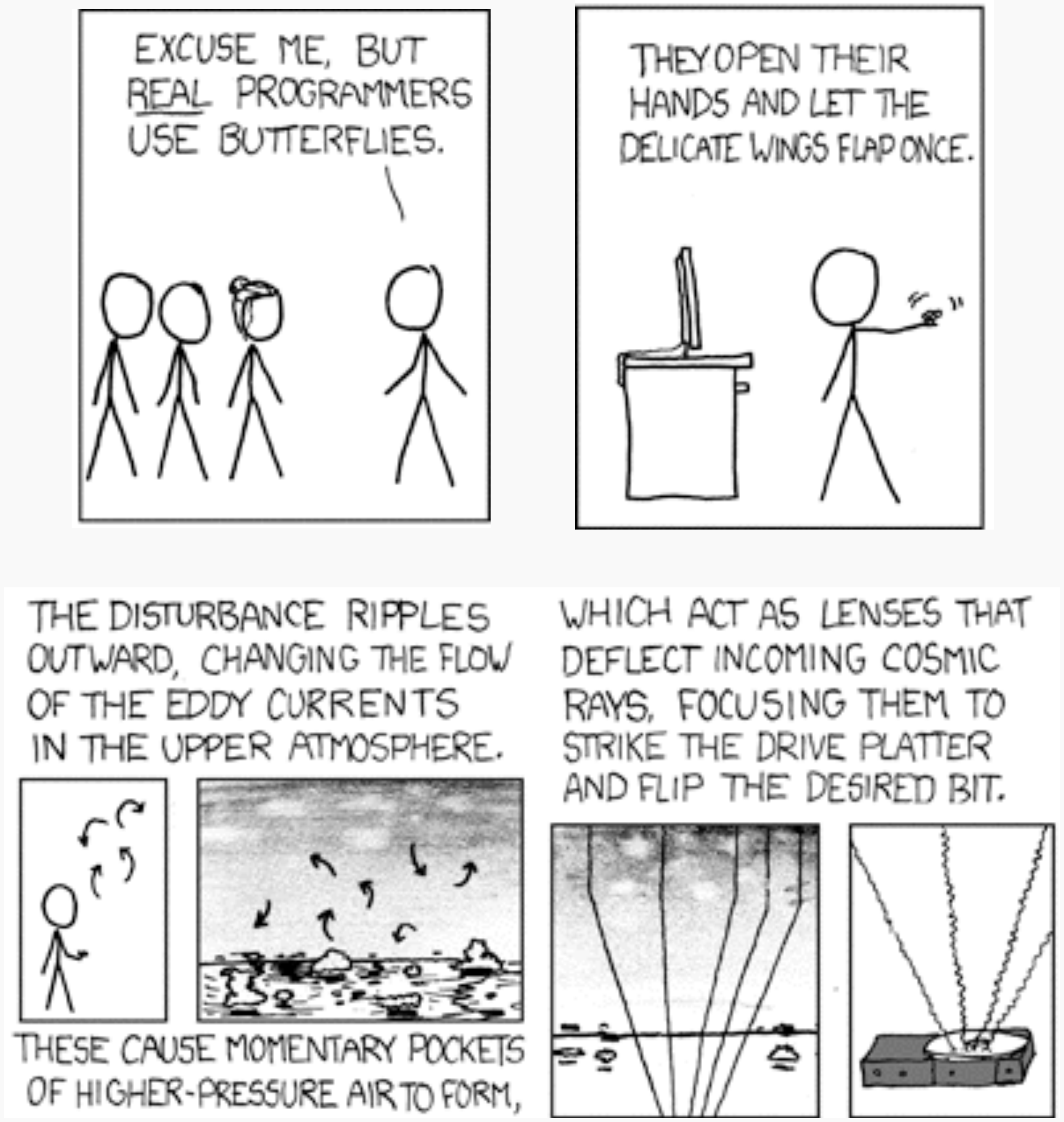
Hiding details
helps reduce mistakes.

```python
import requests

url = 'https://api.github.com/user'
auth = ('username', 'password')

r = requests.get(url, auth=auth)
print r.content
```

Hiding details
makes complex tasks routine.


EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.

https://xkcd.com/378/

Hiding details
provides a stable interface.

Claim:
Good abstraction is aligned with good UX.

ABSTRACTION IN PYTHON

Functions

```
>>> a
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])


>>> np.resize(a, (2,6))
array([[ 2.,  8.,  0.,  6.,  4.,  5.],
       [ 1.,  1.,  8.,  9.,  3.,  6.]])
```

Classes

```
class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String(50))
    dessert = Column(String(50))
```

Classes

```
mali = User(name="mali",
            fullname="Mali Akmanalp",
            password="pumpkin")
```

Classes

```
>>> mali.name
"mali"

>>> mali.name = "mali2"

>>> session.add(mali)
>>> session.commit()
```

PITFALLS

Leaky abstractions

Leaky Abstractions

```python
size = 1000
big_table = [list(range(size)) for _ in range(size)]
```

Leaky Abstractions

```
[[ 2,   8, …,   0,   6],
 [ 4,   5, …,   1,   1],
 [ …,   …, …,   …,   …],
 [ …,   …, …,   …,   …],
 [ 8,   2, …,   5,   6],
 [ 8,   9, …,   3,   6]])
```

Leaky Abstractions

```
In [5]: %%timeit
   ...: for i in range(size):
   ...:     for j in range(size):
   ...:         x = big_table[j][i]
   ...:
10 loops, best of 3: 163 ms per loop
```

Leaky Abstractions

```
In [5]: %%timeit
   ...: for i in range(size):
   ...:     for j in range(size):
   ...:         x = big_table[i][j]
   ...:
10 loops, best of 3: 97.1 ms per loop
```

Leaky Abstractions

```
In [5]: %%timeit
   ...: for i in range(size):
   ...:     for j in range(size):
   ...:         x = big_table[i][j]
   ...:
10 loops, best of 3: 97.1 ms per loop
```

Leaky Abstractions

```
In [5]: %%timeit
   ...: for i in range(size):
   ...:     for j in range(size):
   ...:         x = big_table[i][j]
   ...:
10 loops, best of 3: 97.1 ms per loop
```

Leaky Abstractions

```
[[ 2,   8, …,   0,   6],
 [ 4,   5, …,   1,   1],
 [ …,   …, …,   …,   …],
 [ …,   …, …,   …,   …],
 [ 8,   2, …,   5,   6],
 [ 8,   9, …,   3,   6]])
```

Leaky Abstractions

```
[[ 2,   8, …,   0,   6],
 [ 4,   5, …,   1,   1],
 [ …,   …, …,   …,   …],
 [ …,   …, …,   …,   …],
 [ 8,   2, …,   5,   6],
 [ 8,   9, …,   3,   6]])
```

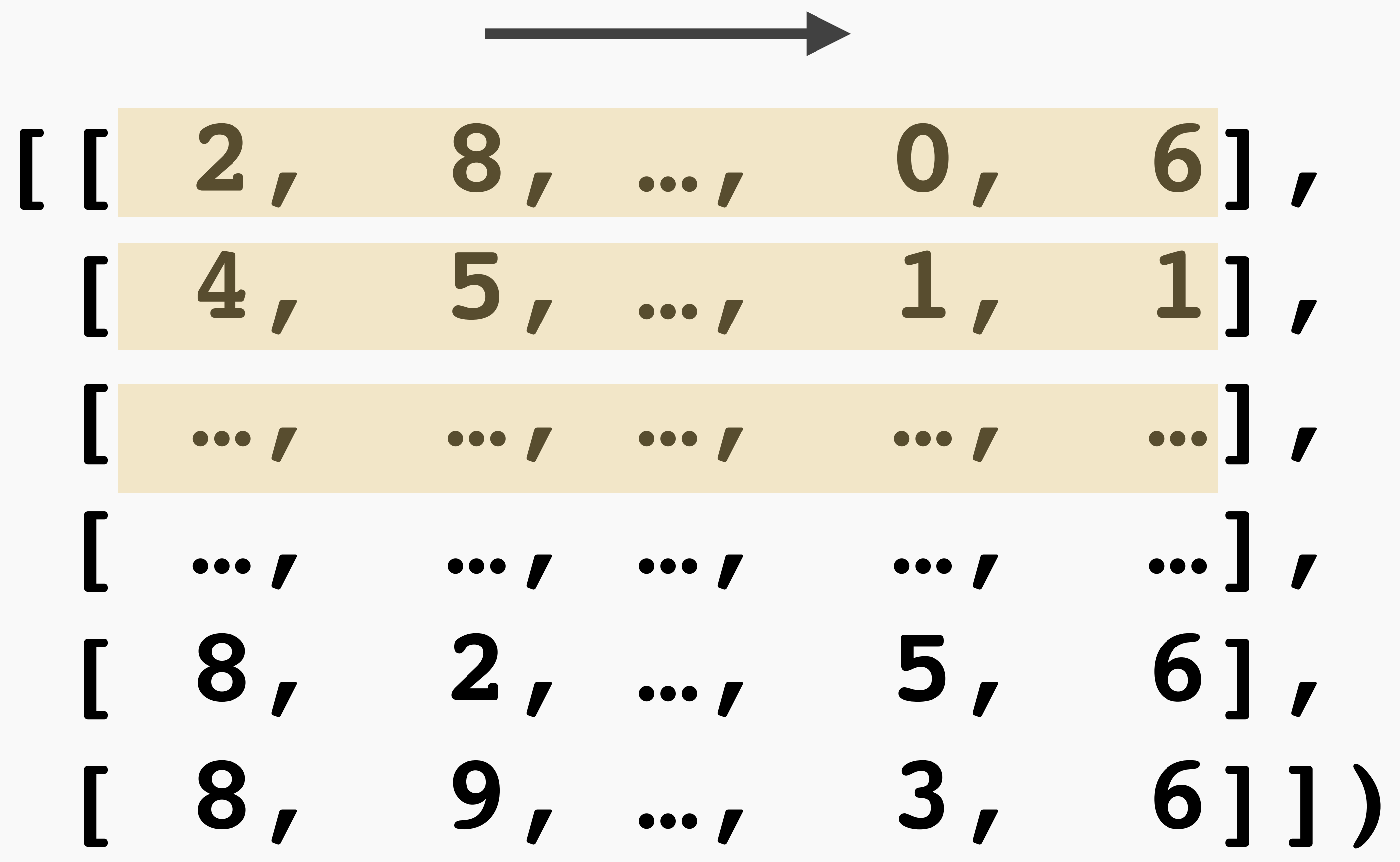
Leaky Abstractions

```
[[ 2,  8, …,  0,  6],
 [ 4,  5, …,  1,  1],
 [ …,  …, …,  …,  …],
 [ …,  …, …,  …,  …],
 [ 8,  2, …,  5,  6],
 [ 8,  9, …,  3,  6]])
```

Leaky Abstractions

```
[[ 2,   8, …,   0,   6],
 [ 4,   5, …,   1,   1],
 [ …,   …, …,   …,   …],
 [ …,   …, …,   …,   …],
 [ 8,   2, …,   5,   6],
 [ 8,   9, …,   3,   6]])
```

Leaky Abstractions



```
[[ 2,   8, …,   0,   6],
 [ 4,   5, …,   1,   1],
 [ …,   …, …,   …,   …],
 [ …,   …, …,   …,   …],
 [ 8,   2, …,   5,   6],
 [ 8,   9, …,   3,   6]])
```

Leaky Abstractions

```
[[ 2,   8, …,   0,   6],
 [ 4,   5, …,   1,   1],
 [ …,   …, …,   …,   …],
 [ …,   …, …,   …,   …],
 [ 8,   2, …,   5,   6],
 [ 8,   9, …,   3,   6]])
```

Leaky Abstractions

```
[[ 2,   8, …,   0,   6],
 [ 4,   5, …,   1,   1],
 [ …,   …, …,   …,   …],
 [ …,   …, …,   …,   …],
 [ 8,   2, …,   5,   6],
 [ 8,   9, …,   3,   6]])
```

Leaky Abstractions

```
[[ 2,   8, …,   0,   6],
 [ 4,   5, …,   1,   1],
 [ …,   …, …,   …,   …],
 [ …,   …, …,   …,   …],
 [ 8,   2, …,   5,   6],
 [ 8,   9, …,   3,   6]])
```

Leaky Abstractions

```
[[ 2,   8, …,   0,   6],
 [ 4,   5, …,   1,   1],
 [ …,   …, …,   …,   …],
 [ …,   …, …,   …,   …],
 [ 8,   2, …,   5,   6],
 [ 8,   9, …,   3,   6]])
```

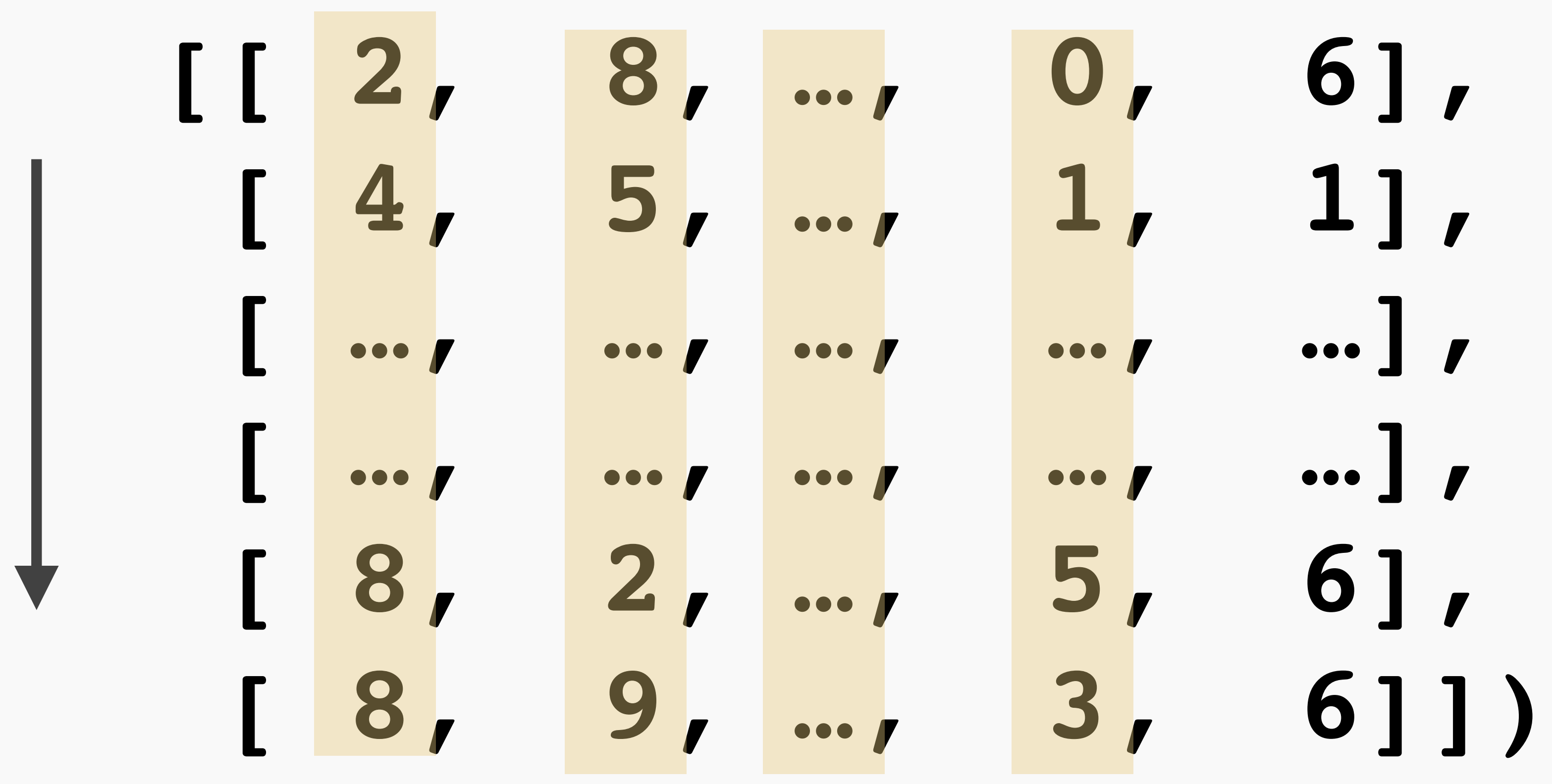
Leaky Abstractions

```
[[ 2,  8, …,  0,  6],
 [ 4,  5, …,  1,  1],
 [ …,  …, …,  …,  …],
 [ …,  …, …,  …,  …],
 [ 8,  2, …,  5,  6],
 [ 8,  9, …,  3,  6]])
```

Leaky Abstractions

```
[[ 2,   8,  …,   0,   6],
 [ 4,   5,  …,   1,   1],
 [ …,   …,  …,   …,   …],
 [ …,   …,  …,   …,   …],
 [ 8,   2,  …,   5,   6],
 [ 8,   9,  …,   3,   6]])
```

Leaky Abstractions



```
[[ 2,   8,  …,   0,   6],
 [ 4,   5,  …,   1,   1],
 [ …,   …,  …,   …,   …],
 [ …,   …,  …,   …,   …],
 [ 8,   2,  …,   5,   6],
 [ 8,   9,  …,   3,   6]])
```

Under-abstraction

Guts everywhere

State everywhere

Control flow everywhere

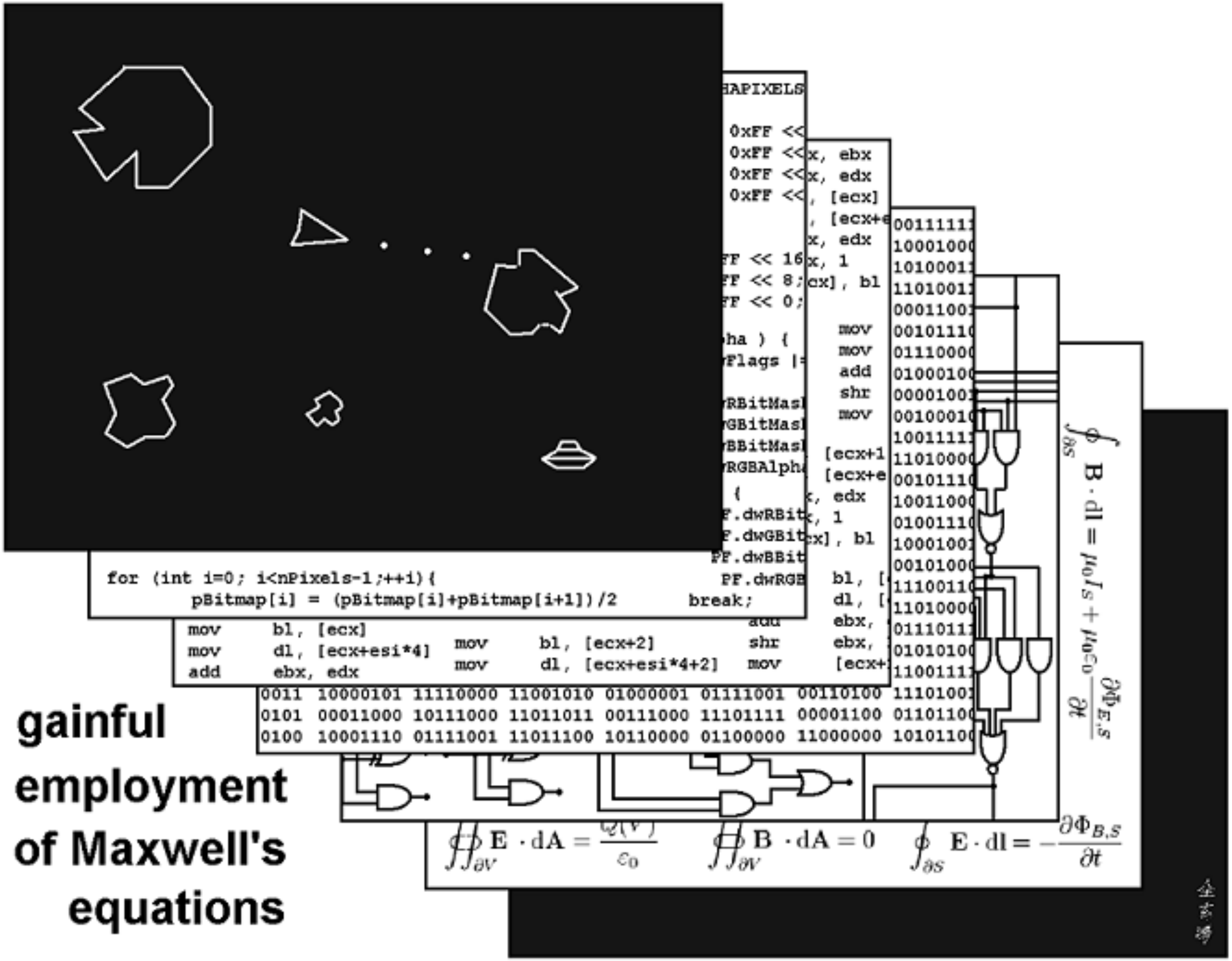
Over-abstraction

Coupling:
To change one thing, you must change all things.

Cohesion:

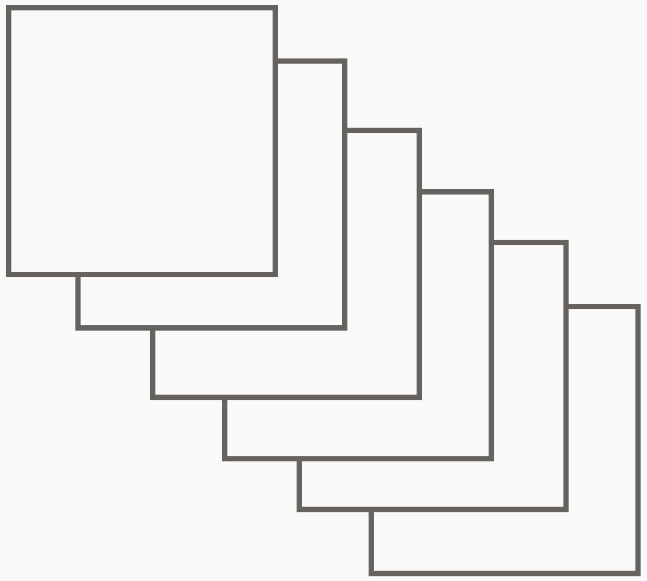
A thing that does too many things at the same time.

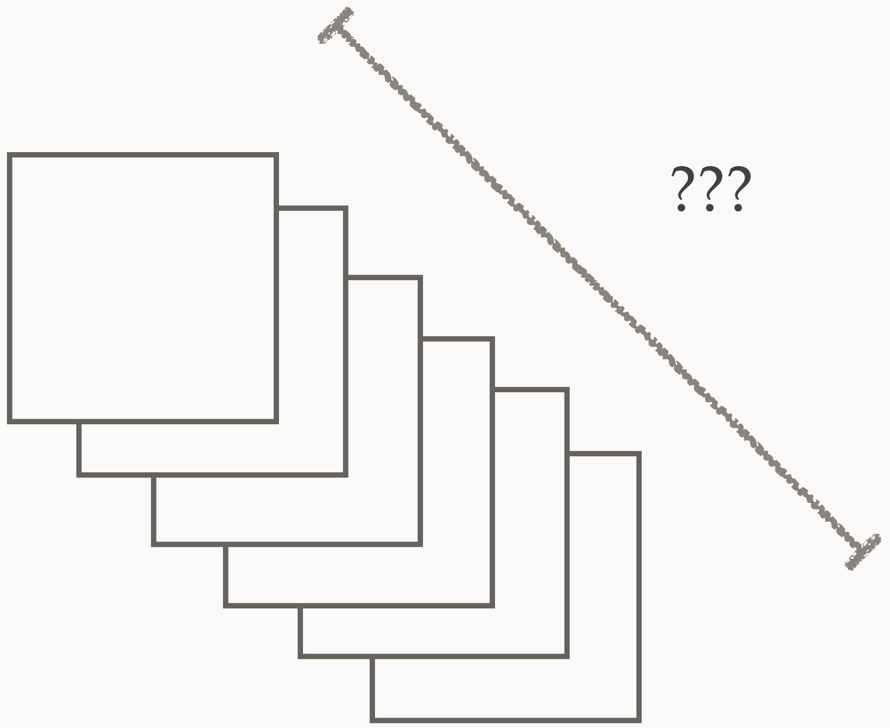
DECIDING ON THE LEVEL OF ABSTRACTION

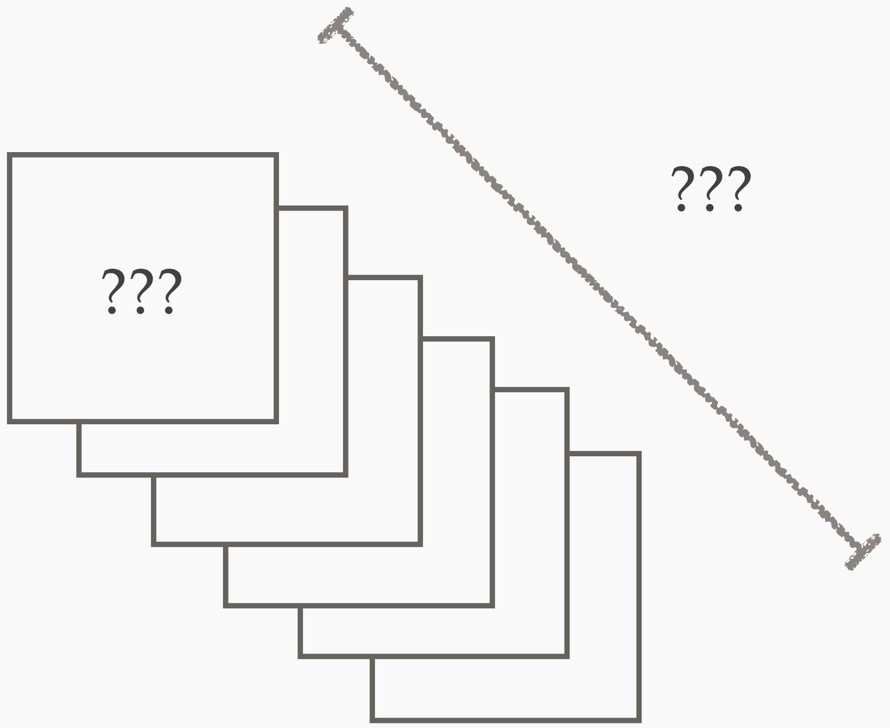

for (int i=0; i<nPixels-1;++i){
pBitmap[i] = (pBitmap[i]+pBitmap[i+1])/2
$\oint_{\partial S} \mathbf{B} \cdot d\mathbf{l} = \mu_0 I_S + \mu_0 \varepsilon_0 \frac{\partial \Phi_{E,S}}{\partial t}$
$\oint_{\partial S} \mathbf{E} \cdot d\mathbf{l} = -\frac{\partial \Phi_{B,S}}{\partial t}$
$\oiint_{\partial V} \mathbf{B} \cdot d\mathbf{A} = 0$

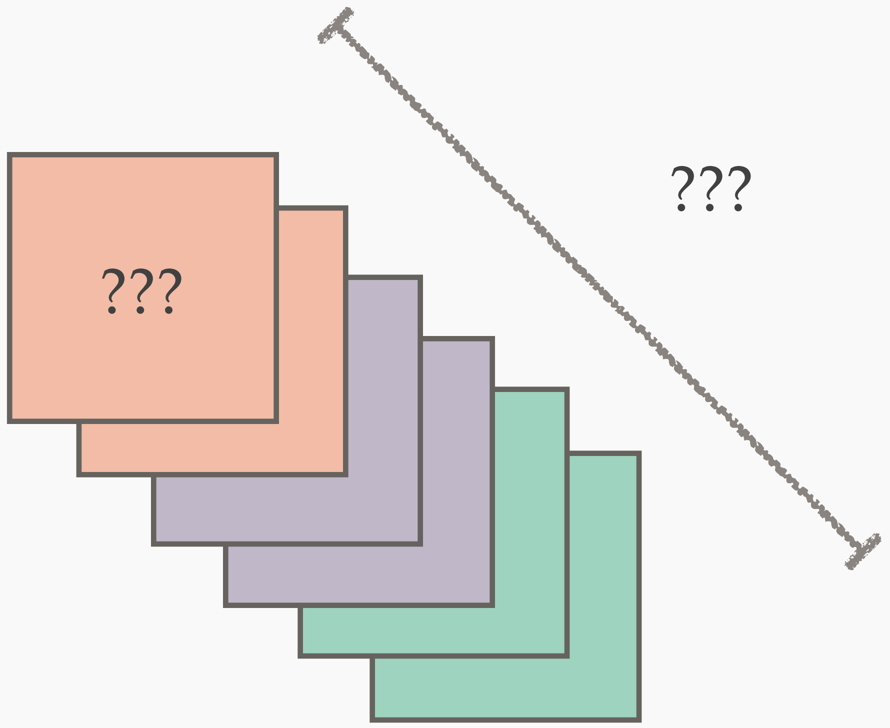
gainful employment of Maxwell's equations

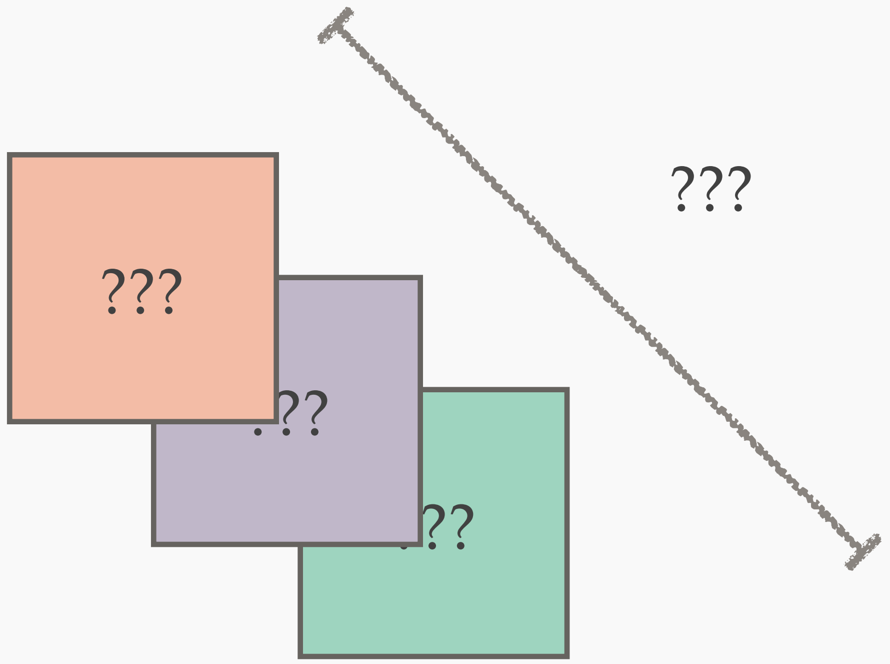
http://abstrusegoose.com/307











?????

Press release first

😄

Press release first

😄

😎

Press release first

Press release first

Press r irst

"Imaginary Code" second

Rewrite usage examples with existing libraries

What does it cost me?

How likely is this to change?

How does this abstraction benefit the user?

Don't
Repeat
Yourself?

Don't
Refactor
Yet!

https://news.ycombinator.com/item?id=12528181

"Prefer duplication over the wrong abstraction"

–Sandi Metz

https://www.sandimetz.com/blog/2016/1/20/the-wrong-abstraction

Incremental architecture

Good architecture and abstraction decisions follow from domain knowledge.

More time on project
More domain knowledge

Earlier on in the project you have less domain knowledge

Build less structure up front

TRICKS OF THE TRADE

Trick:
Abstraction need not mean building a wall.

Flask

```
from flask import app

@app.route('/dessert')
def yum():
    return "donuts!"
```

Flask

```python
from flask import app

@app.route('/dessert')
def yum():
    return "donuts!"
```

Flask

```
from flask import app


def yum():
    return "donuts!"

app.add_url_rule('/', 'dessert', dessert)
```

Flask

```
from flask import app


def yum():
    return "donuts!"

app.add_url_rule('/', 'dessert', dessert)
```

Flask

```
def add_url_rule(self, rule, **options, ...):
    # ...
    rule = self.url_rule_class(rule, **options, ...)
    # ...
    self.url_map.add(rule)
    # ...
```

https://github.com/pallets/flask/blob/501f0431259a30569a5e62bcce68d102fc3ef993/flask/app.py#L1071

Flask

```python
def add_url_rule(self, rule, **options, ...):
    # ...
    rule = self.url_rule_class(rule, **options, ...)
    # ...
    self.url_map.add(rule)
    # ...
```

https://github.com/pallets/flask/blob/501f0431259a30569a5e62bcce68d102fc3ef993/flask/app.py#L1071

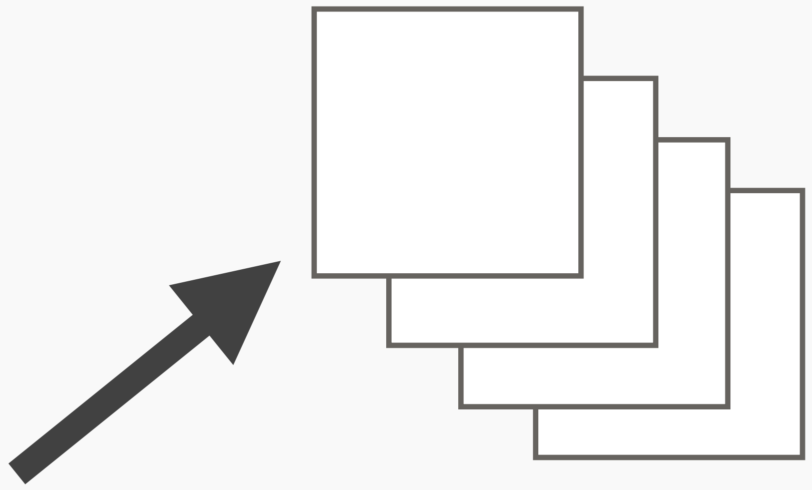
Flask

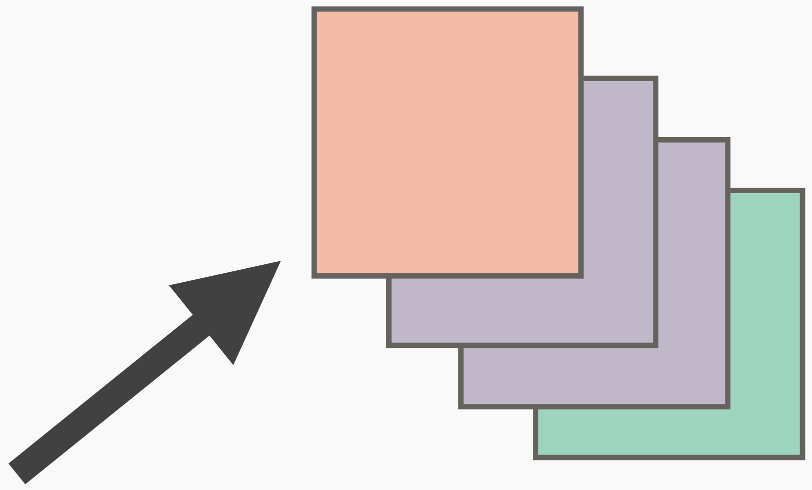
```
def add_url_rule(self, rule, **options, ...):
    # ...
    rule = self.url_rule_class(rule, **options, ...)
    # ...
    self.url_map.add(rule)
    # ...
```

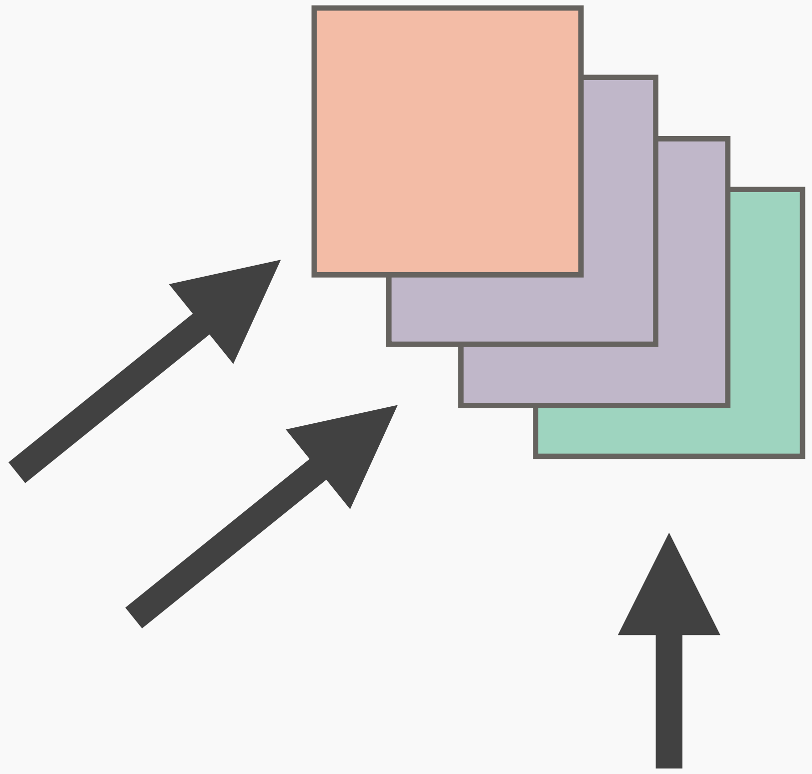
https://github.com/pallets/flask/blob/501f0431259a30569a5e62bcce68d102fc3ef993/flask/app.py#L1071

Trick:

More layers can make things cleaner.







Bokeh Charts

Bokeh Glyphs

Bokeh JS

SQLAlchemy ORM

SQLAlchemy Core

Seaborn

Matplotlib

CONCLUSION

Claim:

The <u>right</u> level of abstraction is audience-specific.

Requests vs urllib2

```python
import requests

url = 'https://api.github.com/user'
auth = ('username', 'password')

r = requests.get(url, auth=auth)
print r.content
```

```python
import urllib2

gh_url = 'https://api.github.com/
user'

req = urllib2.Request(gh_url)

password_manager =
urllib2.HTTPPasswordMgrWithDefaultRea
lm()
password_manager.add_password(None,
gh_url, 'user', 'pass')

auth_manager =
urllib2.HTTPBasicAuthHandler(password
_manager)
opener =
urllib2.build_opener(auth_manager)

urllib2.install_opener(opener)

handler = urllib2.urlopen(req)

print handler.read()
```

http://fishsnack.deviantart.com/art/Creation-of-Adam-in-the-21st-Century-280501206

Theory:

"For Humans" adds a layer we didn't know we were missing.

Abstraction isn't a goal,
It's a tool.

Hearing from you makes me happy!

(@makmanalp)

Classes

```
select *
from users
where name = "mali";
```

Classes

```
"""
select *
from users
where name = '{}';
""".format("mali")
```

Classes

```
"""
select *
from users
where name = '{}'
and   dessert = '{}';
""".format("mali", "pie")
```

Classes

```
User.query\
     .filter_by(name=“mali”)\
     .all()
```

Classes

```
User.query\
     .filter_by(
         name=“mali”,
         dessert="cake")\
     .all()
```

Classes

```
class User(Base):
    __tablename__ = 'users'
    id = Column(Integer,primary_key=True)
    name = Column(String(50))
    dessert = Column(String(50))
```

hiding details makes things easy to use??

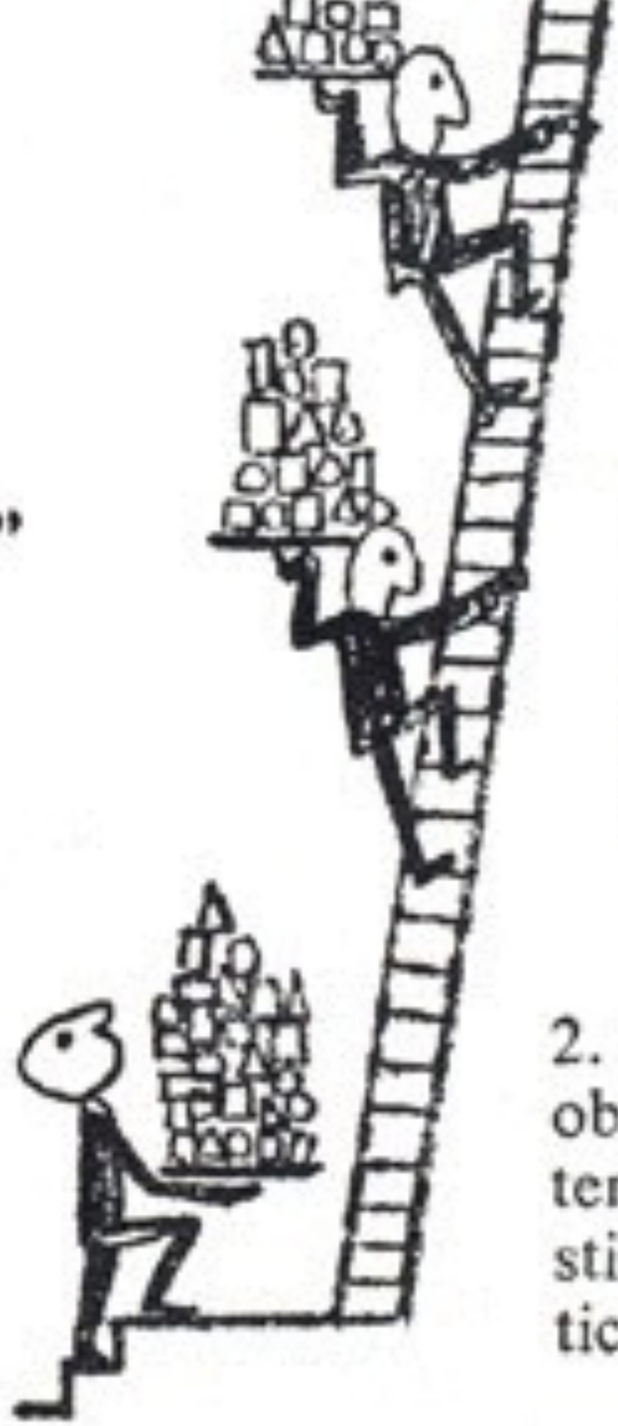
Read a lot of code!

Hiding details
papers over grossness???

4. "cow"

3. "Bessie"

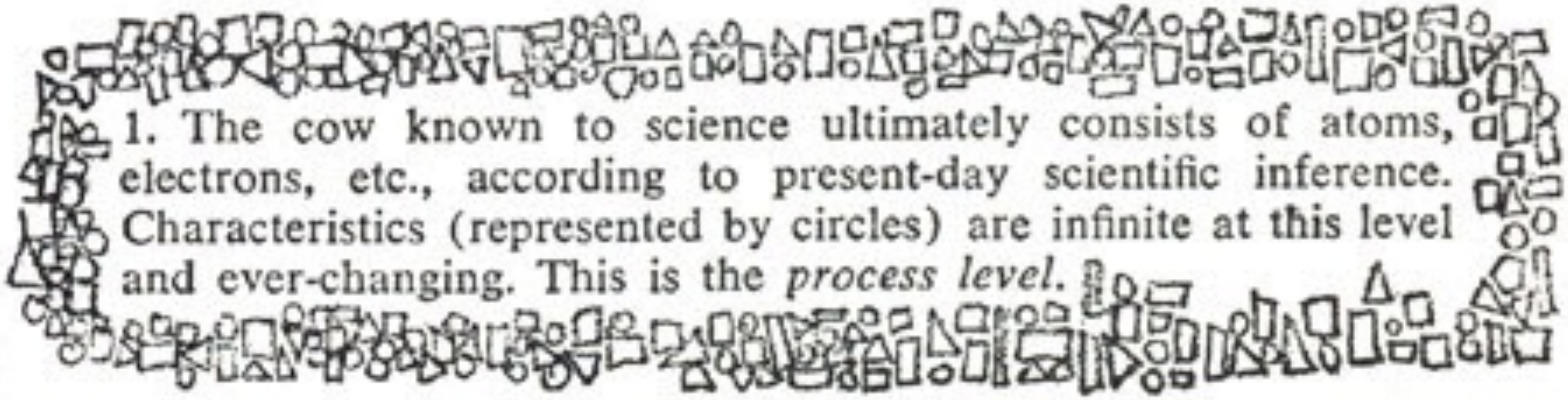
2.



4. The word "cow" stands for the characteristics we have abstracted as common to cow_1, cow_2, cow_3 . . . cow_n. Characteristics peculiar to specific cows are left out.

3. The word "Bessie" (cow_1) is the *name* we give to the object of perception of level 2. The name *is not* the object; it merely *stands for* the object and omits reference to many of the characteristics of the object.

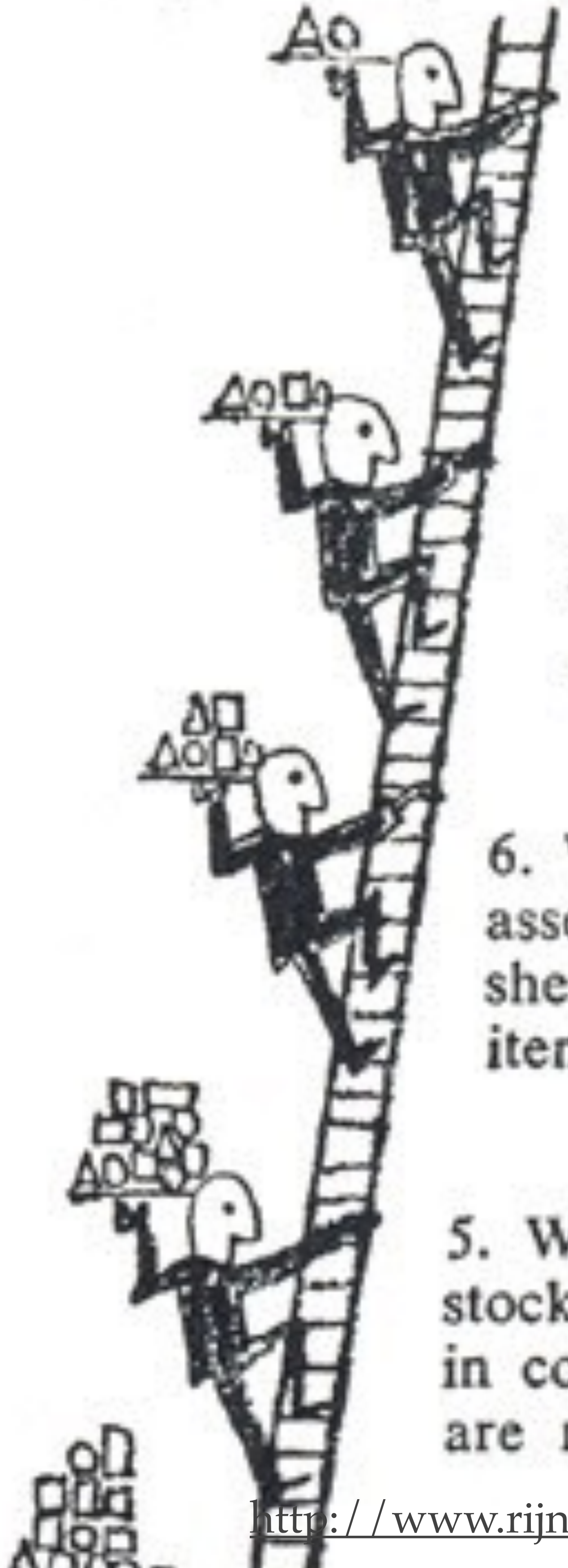
2. The cow we perceive is not the word, but the object of experience, that which our nervous system abstracts (selects) from the totality that constitutes the process-cow. Many of the characteristics of the process-cow are left out.



1. The cow known to science ultimately consists of atoms, electrons, etc., according to present-day scientific inference. Characteristics (represented by circles) are infinite at this level and ever-changing. This is the *process level.*

ABSTRACTION LADDER

Start reading from the bottom *UP*



8. "wealth"

8. The word "wealth" is at an extremely high level of abstraction, omitting *almost* all reference to the characteristics of Bessie.

7. "asset"

7. When Bessie is referred to as an "asset," still more of her characteristics are left out.

6. "farm assets"

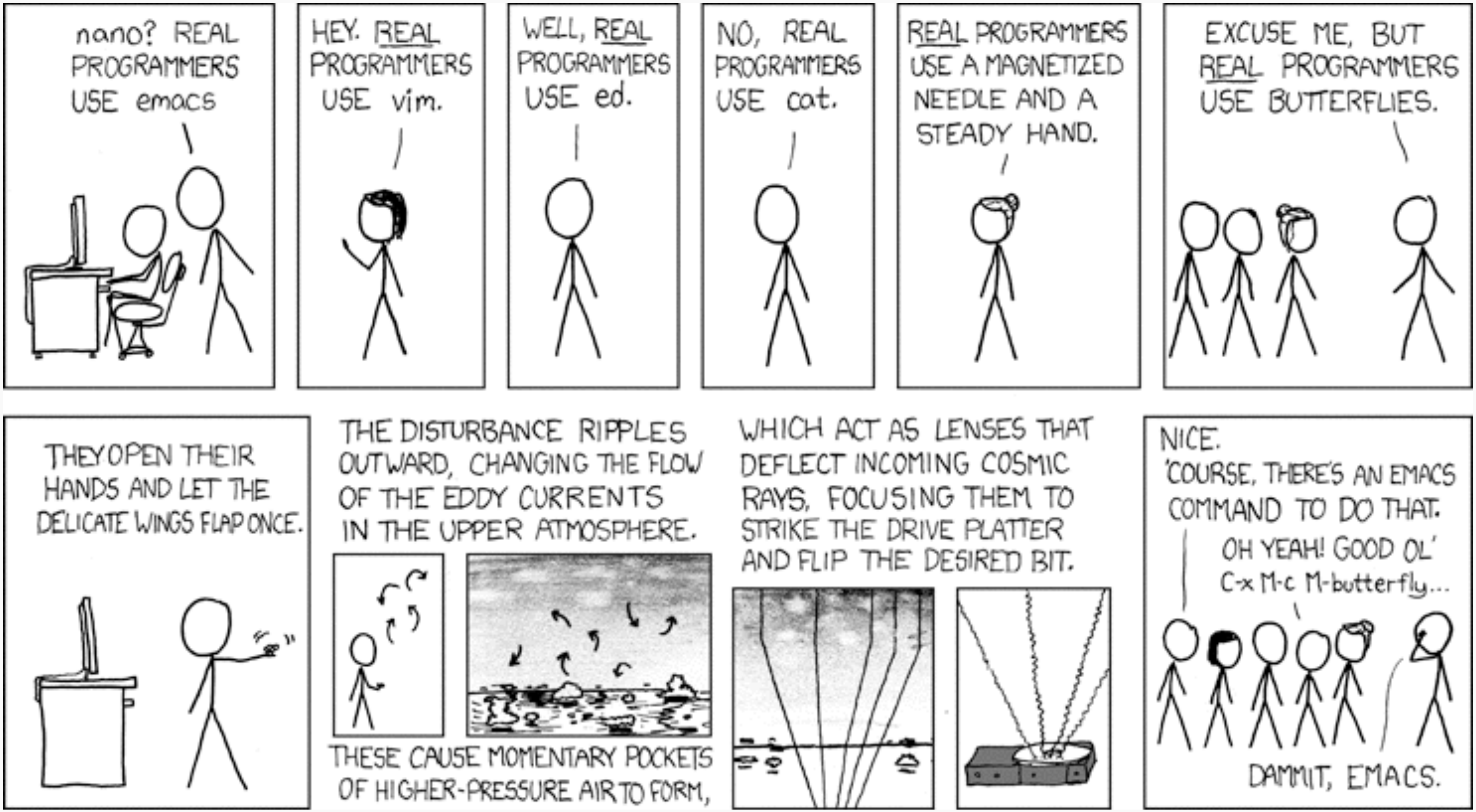
6. When Bessie is included among "farm assets," reference is made only to what she has in common with all other salable items on the farm.

5. "livestock"

5. When Bessie is referred to as "livestock," only those characteristics she has in common with pigs, chickens, goats, etc., are referred to.

http://www.rijnlandmodel.nl/english/general_semantics/abstraction_ladder.htm

PITFALLS


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

https://xkcd.com/378/