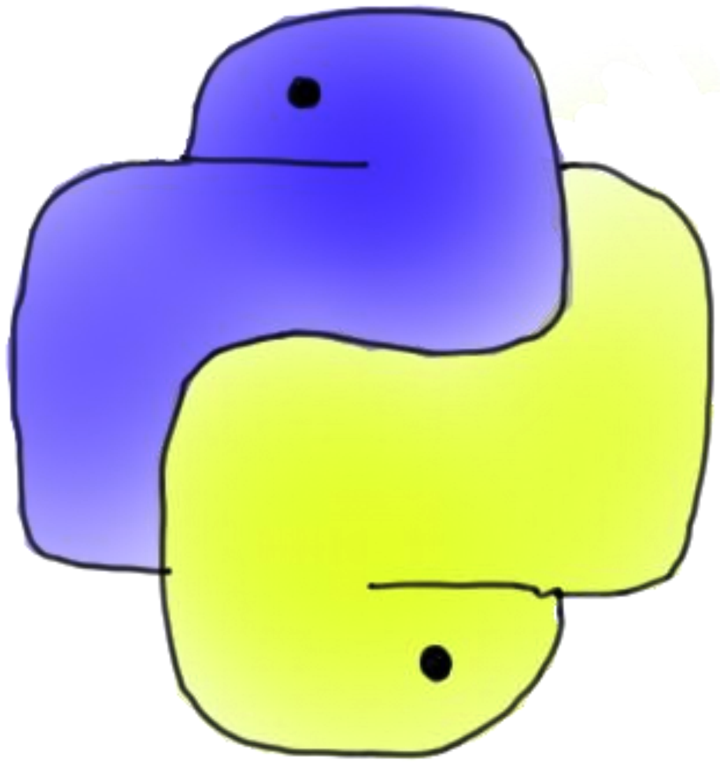


A slice of Python

```
def a_function():
    # a comment
    a = “a string variable”

class A_Class():
    def __init__(self): # Initializer
        self.xx = 42 # instance var

for obj in objects:
    # do something with obj
```

What I inherited

```
C:\Tools\Maya\Scripts
C:\Tools\Maya\3rdParty\Scripts
```

What I inherited

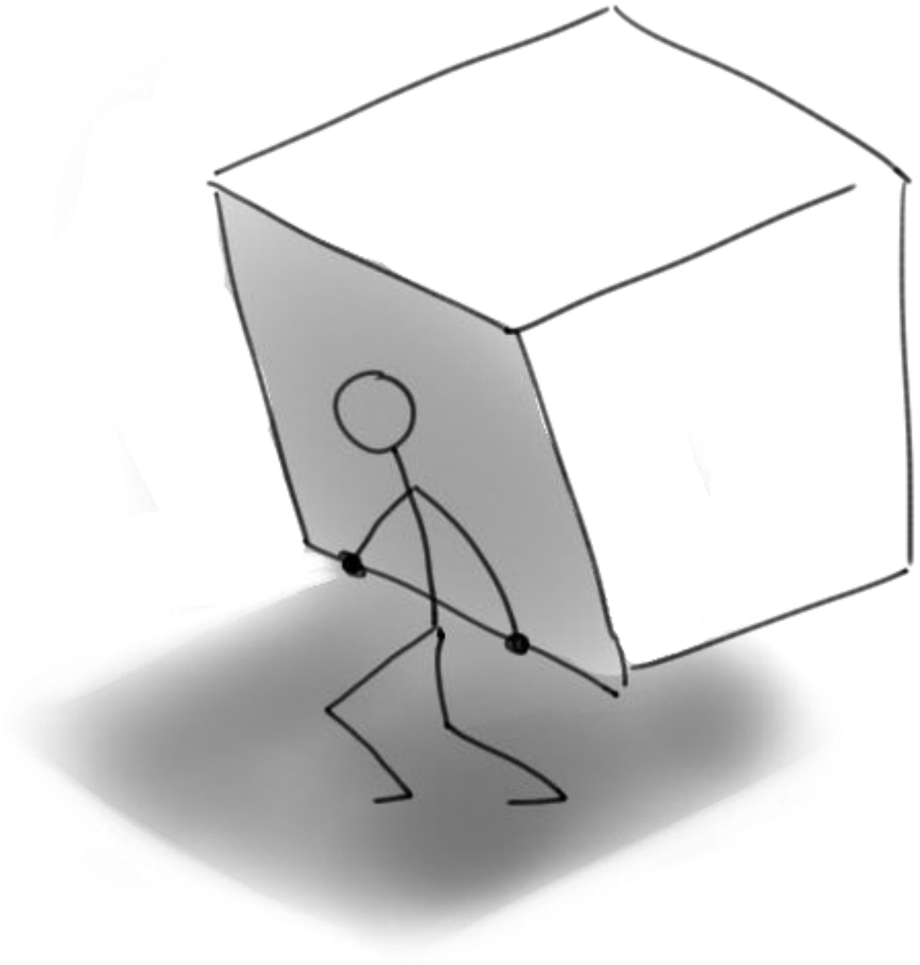
```
C:\Tools\Maya\Scripts\MyExportInt.py
C:\Tools\Maya\3rdParty\Scripts
```

WTF?

What I inherited

```
# This is dumb.
if not ValidateArray(objs):
    for obj in objs:
        # Do stuff

# This is all that is needed.
for obj in objs:
    # Do Stuff
```

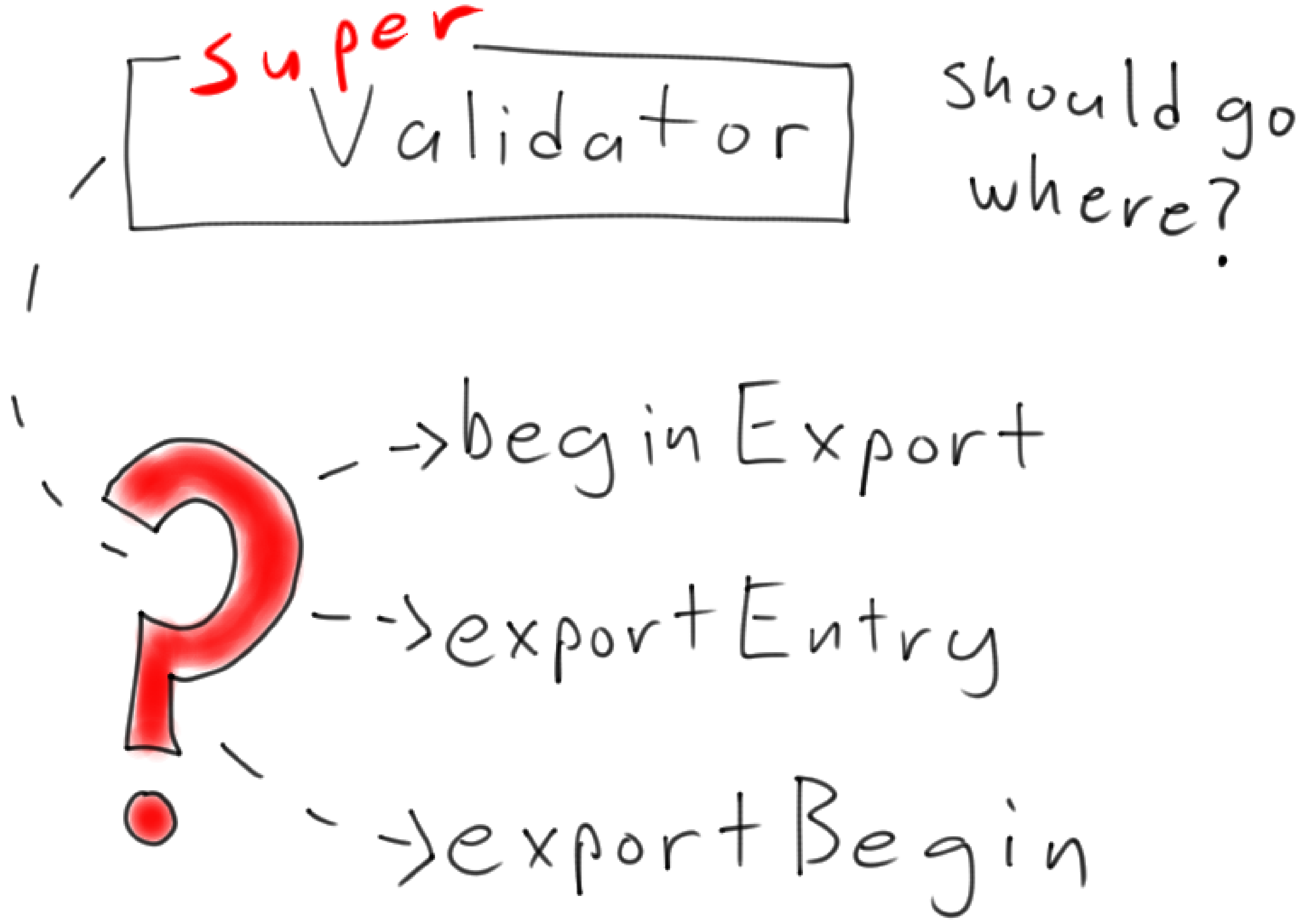
Legacy code can be a burden



Symptoms: confusing logic

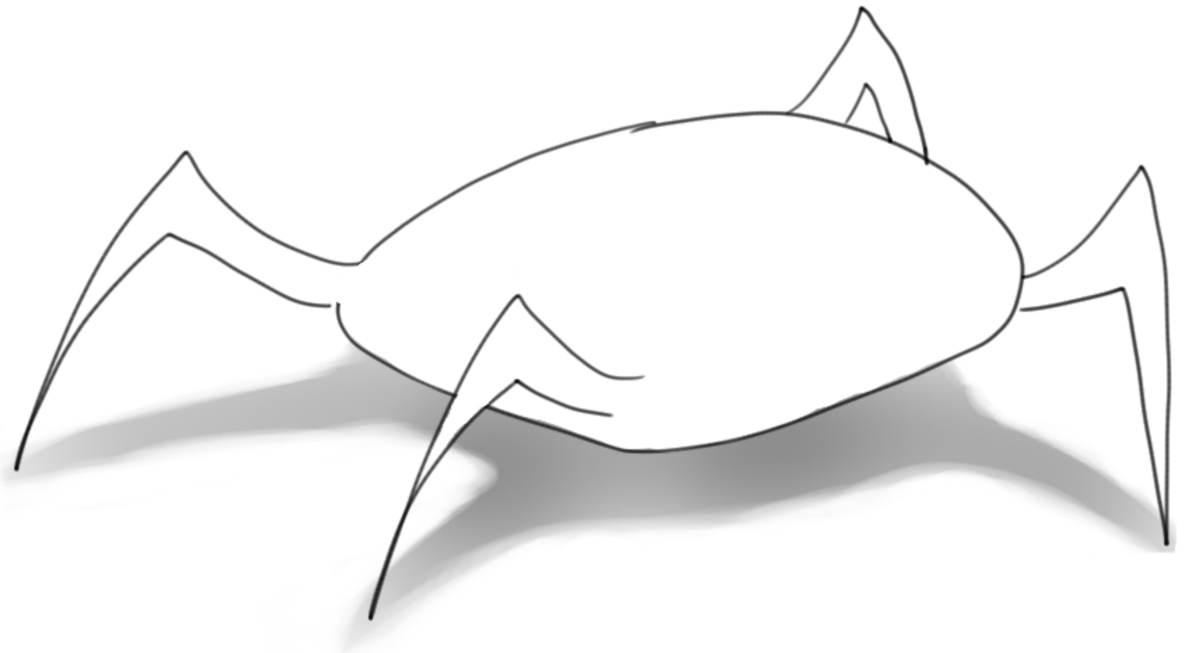
```
for f in files:
    if [f for s in GAME_EXT if s in f]:
        for s in GAME_EXT:
```

Symptoms: How to make changes?

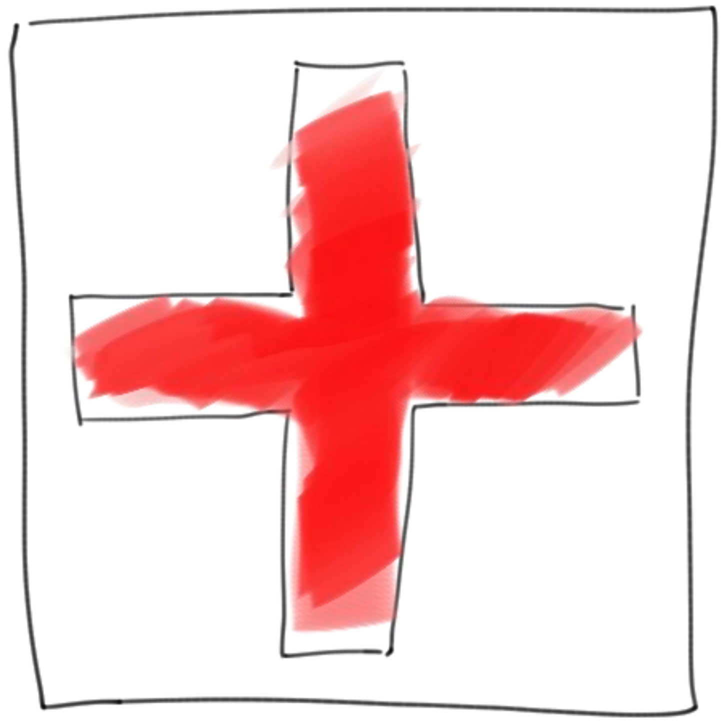


Symptoms: Fear of breakage

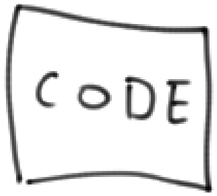
Bad code is an infection that spreads!



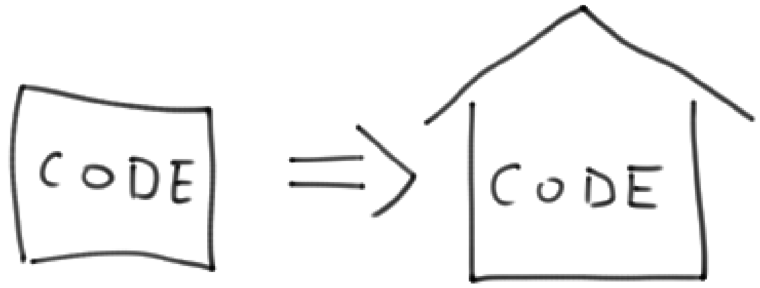
Not all hope is lost



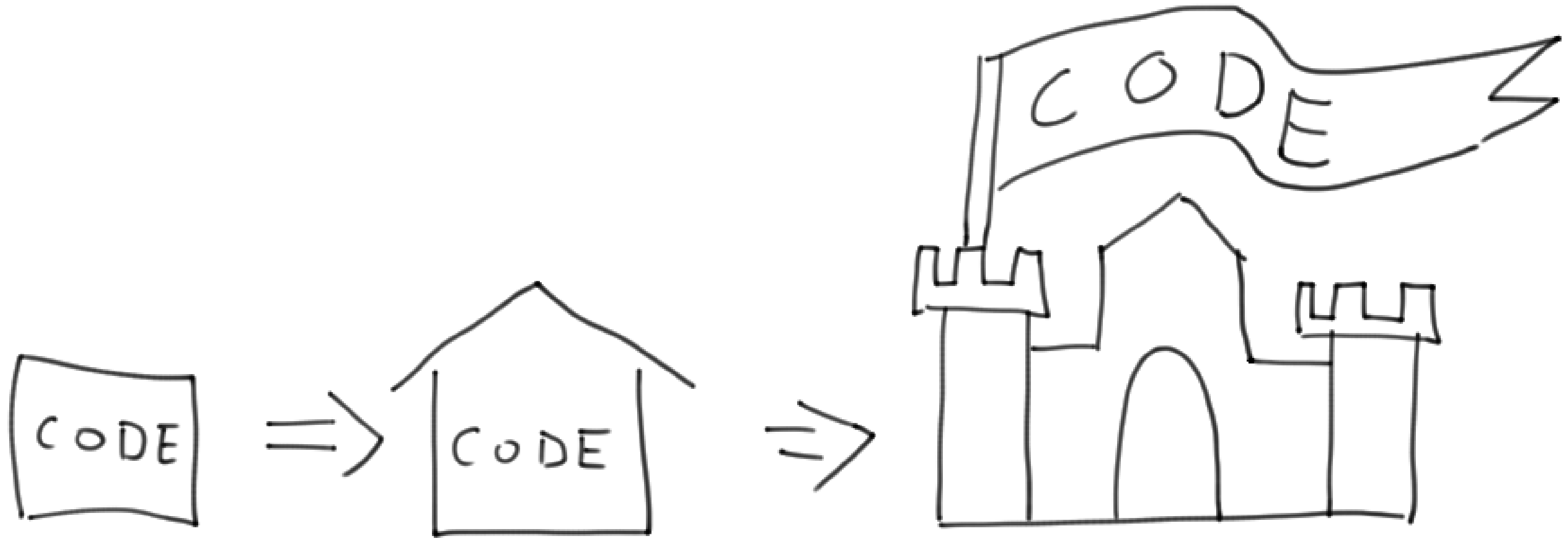
Not all hope is lost



Not all hope is lost



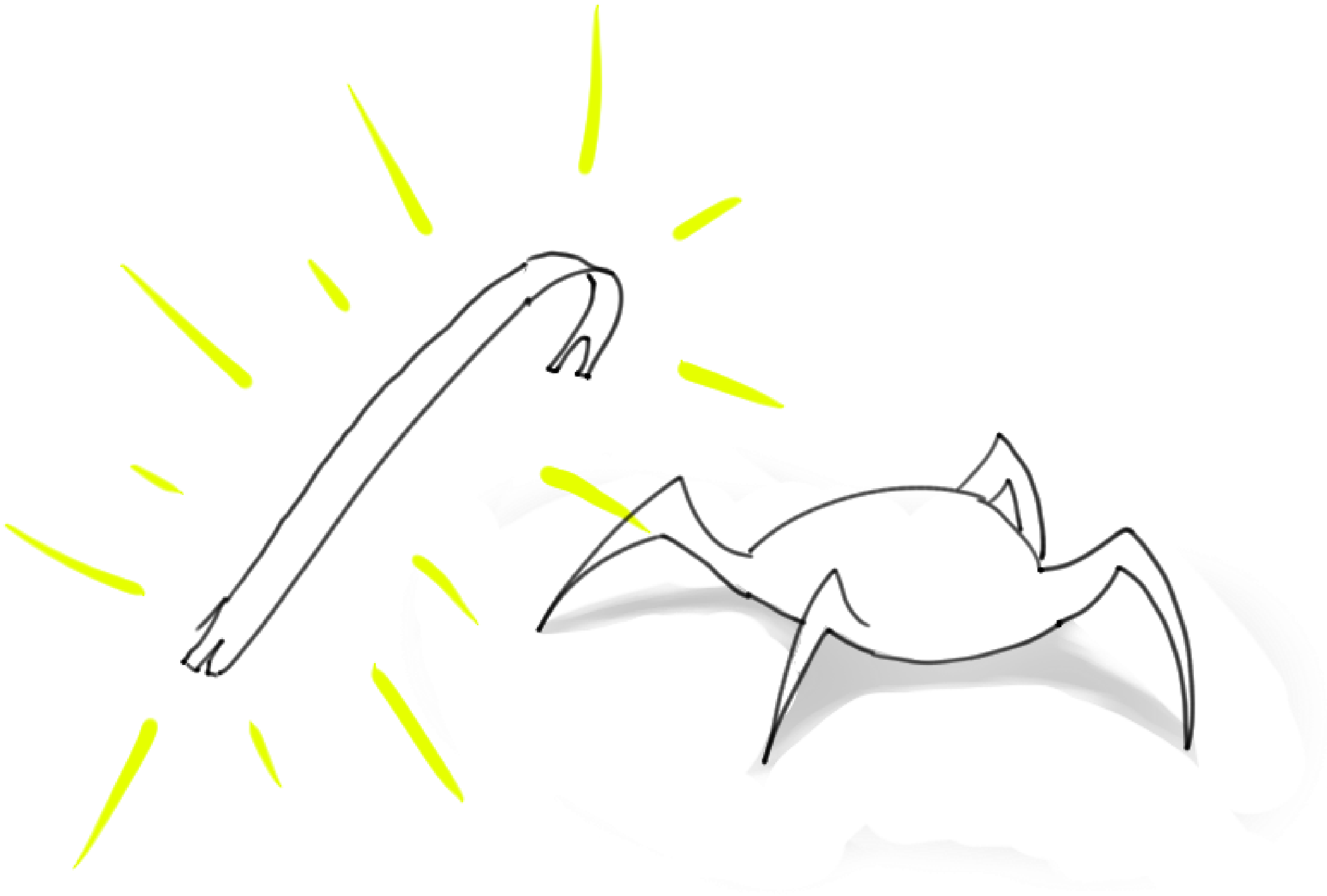
Not all hope is lost



PhD in

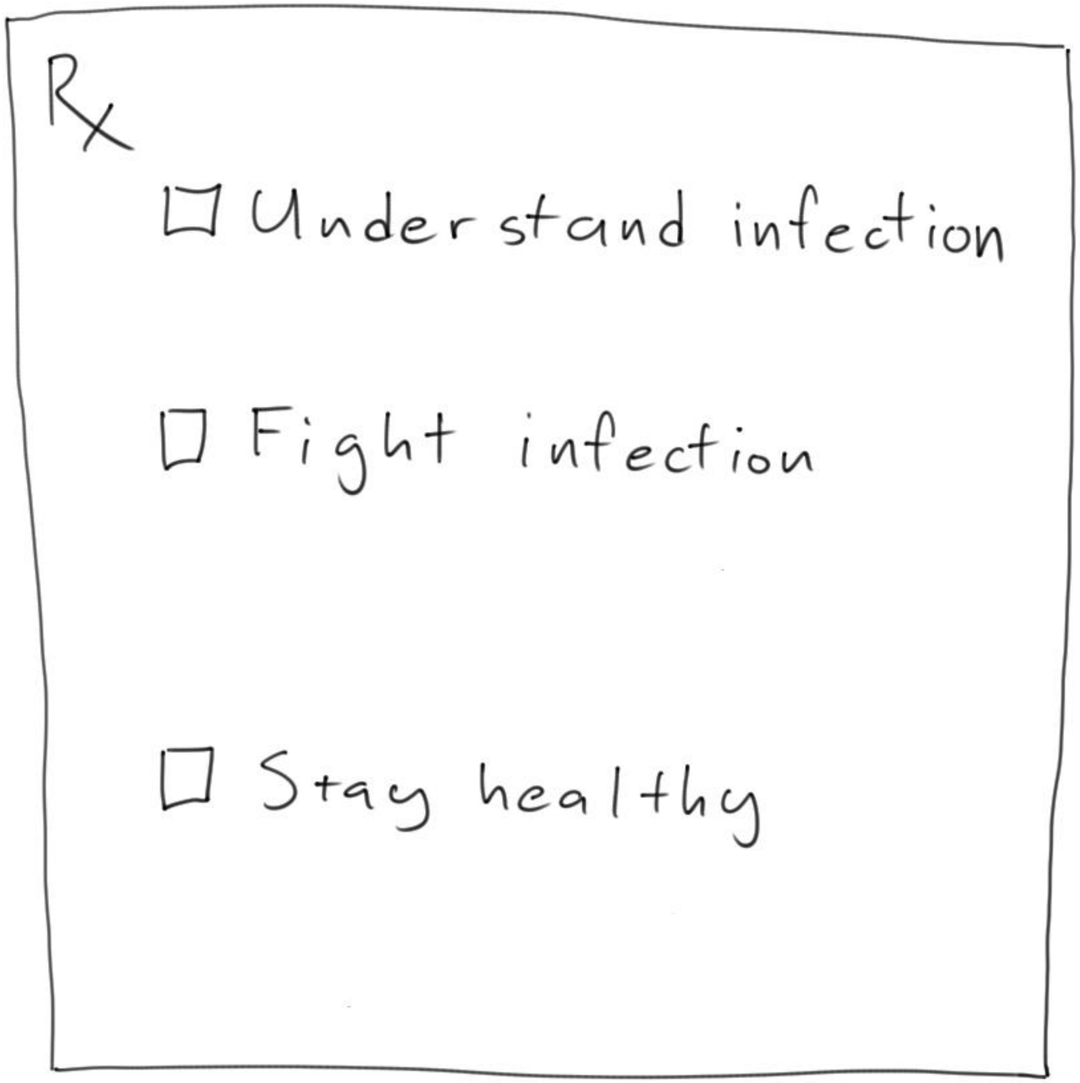
Kicking Ass!

The Prescription



The Prescription

The Prescription: Continuous Maintenance



℞

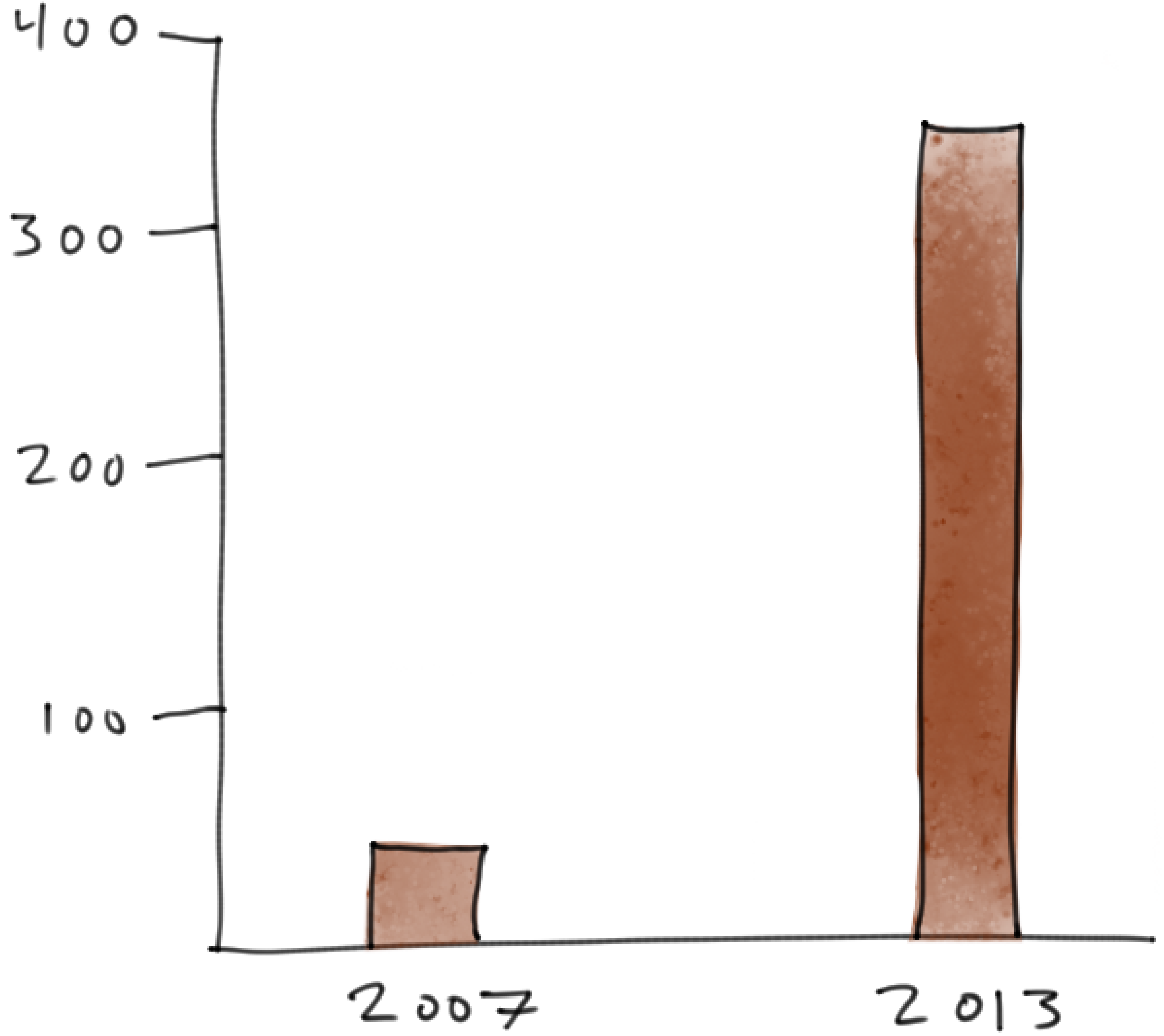
- [ ] Understand infection
- [ ] Fight infection
- [ ] Stay healthy

Rx

- ☐ Understand infection
- ☐ Fight infection
- ☐ Stay healthy

Bad Code is an infection that spreads

Bad Code is an infection that spreads



How bad code happens

New to the language/system/toolchain etc.

New to the language/system/toolchain etc.

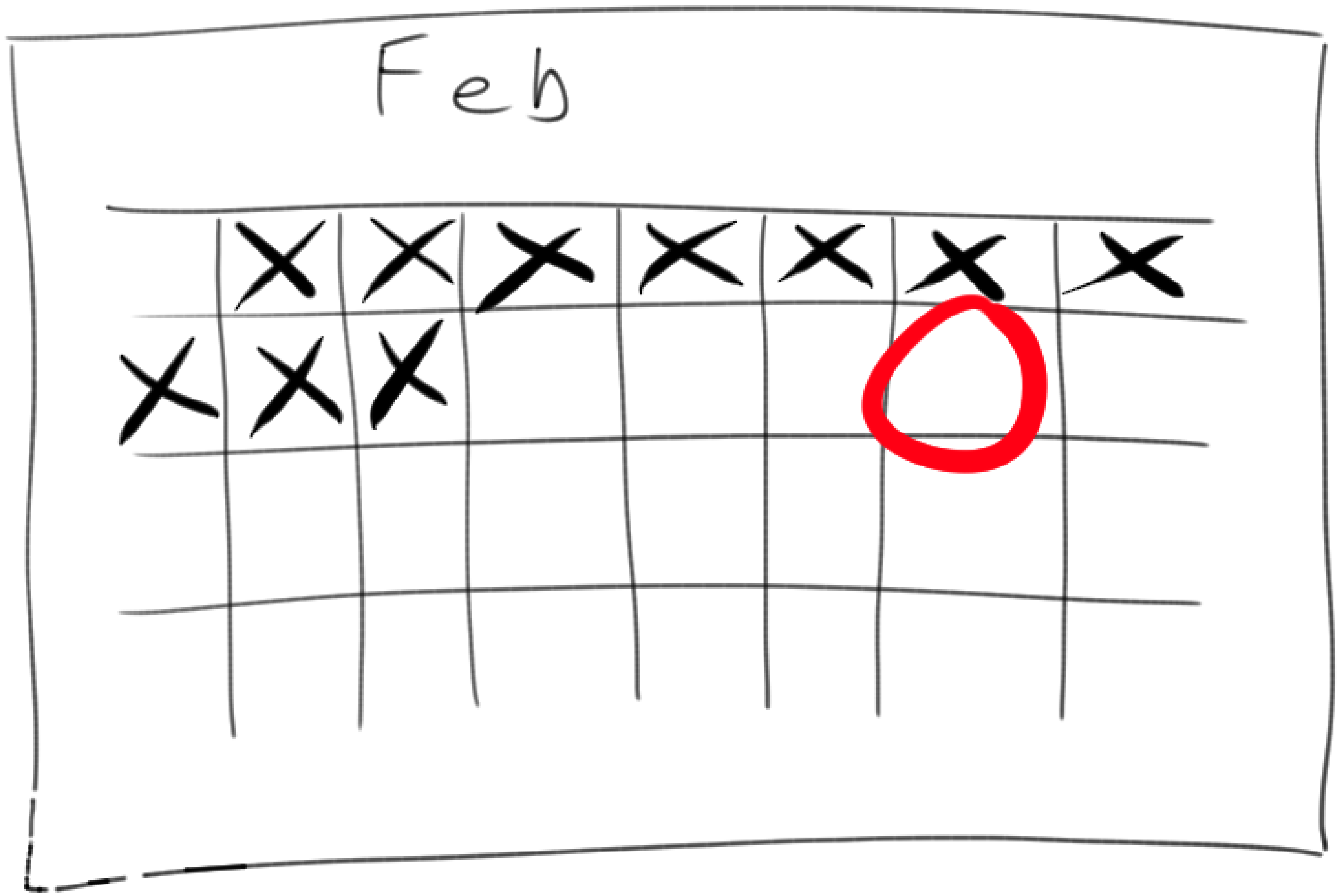
Esperanto: head beetle

The criteria changes

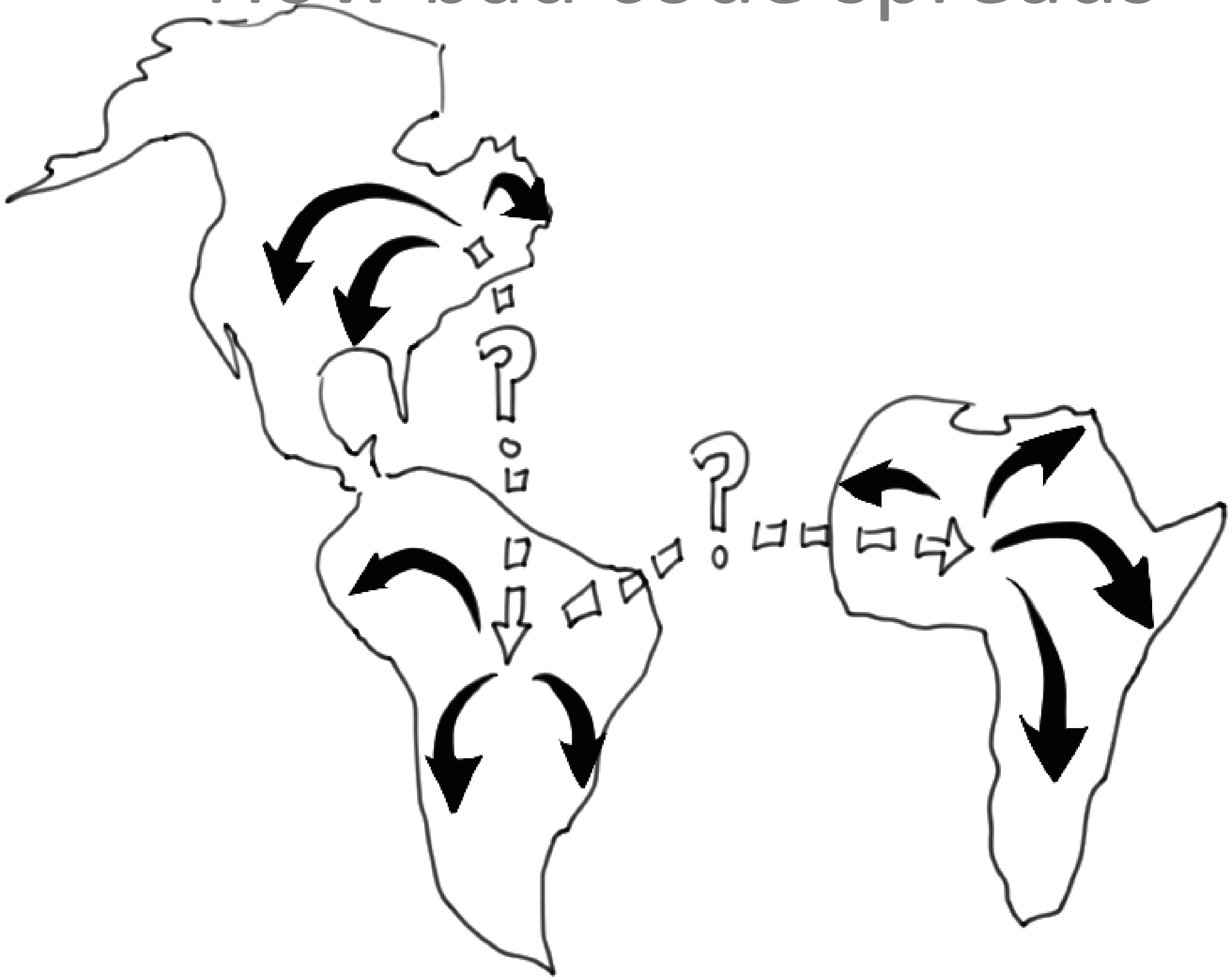
DUKE NUKEM

FOREVER

Everyone has deadlines.



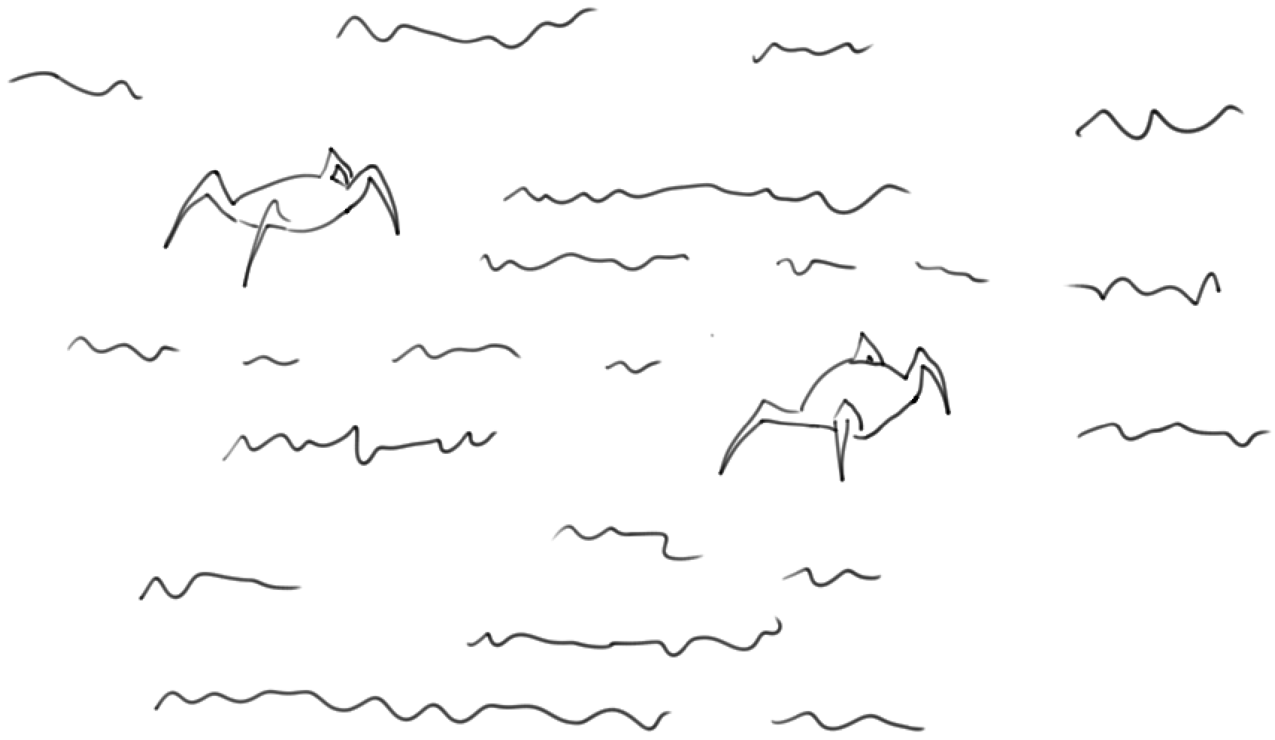
How bad code spreads



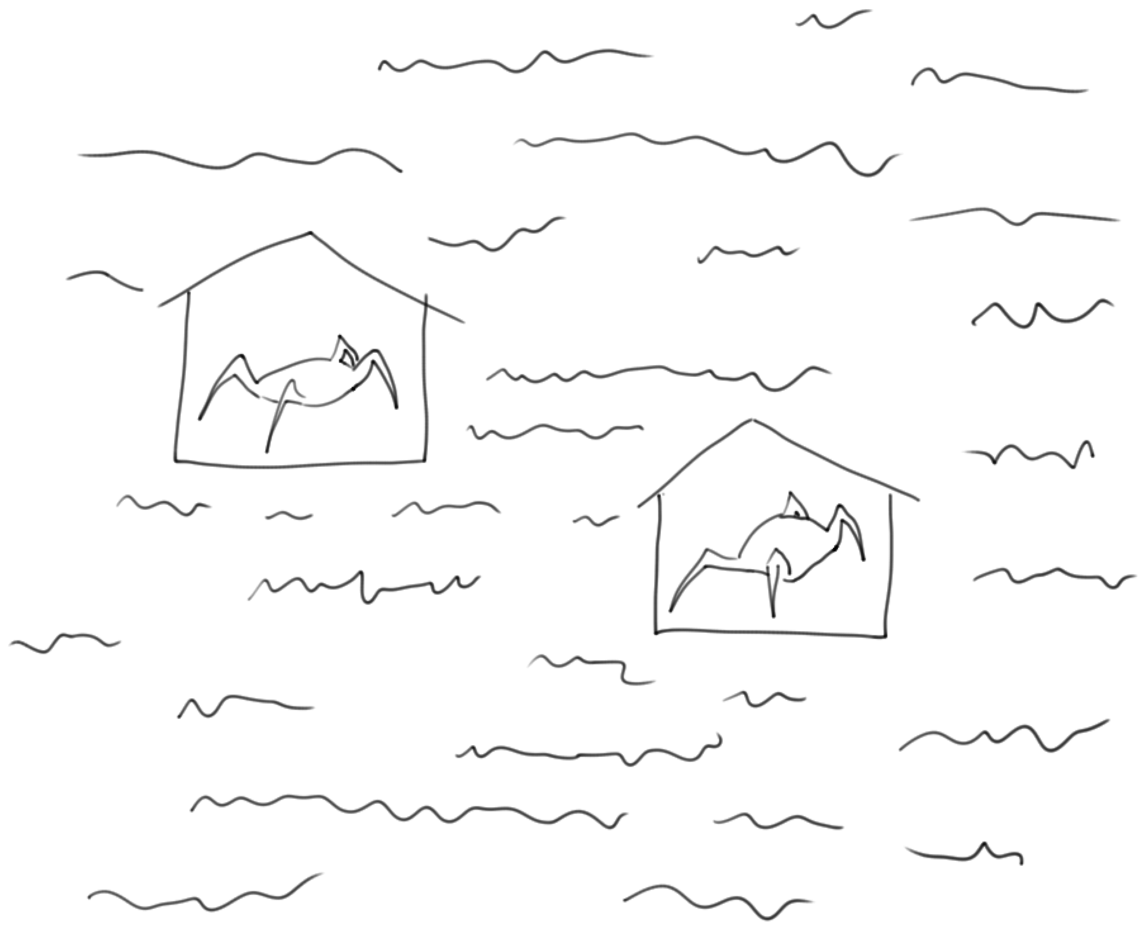
Others will reference the code.

Others will reference the code.

Growing systems entrench bad code



Growing systems entrench bad code



Rx

Overview

- ☐ Understand infection
  - new to system
  - bad code is referenced
- ☐ Fight infection
- ☐ Stay healthy

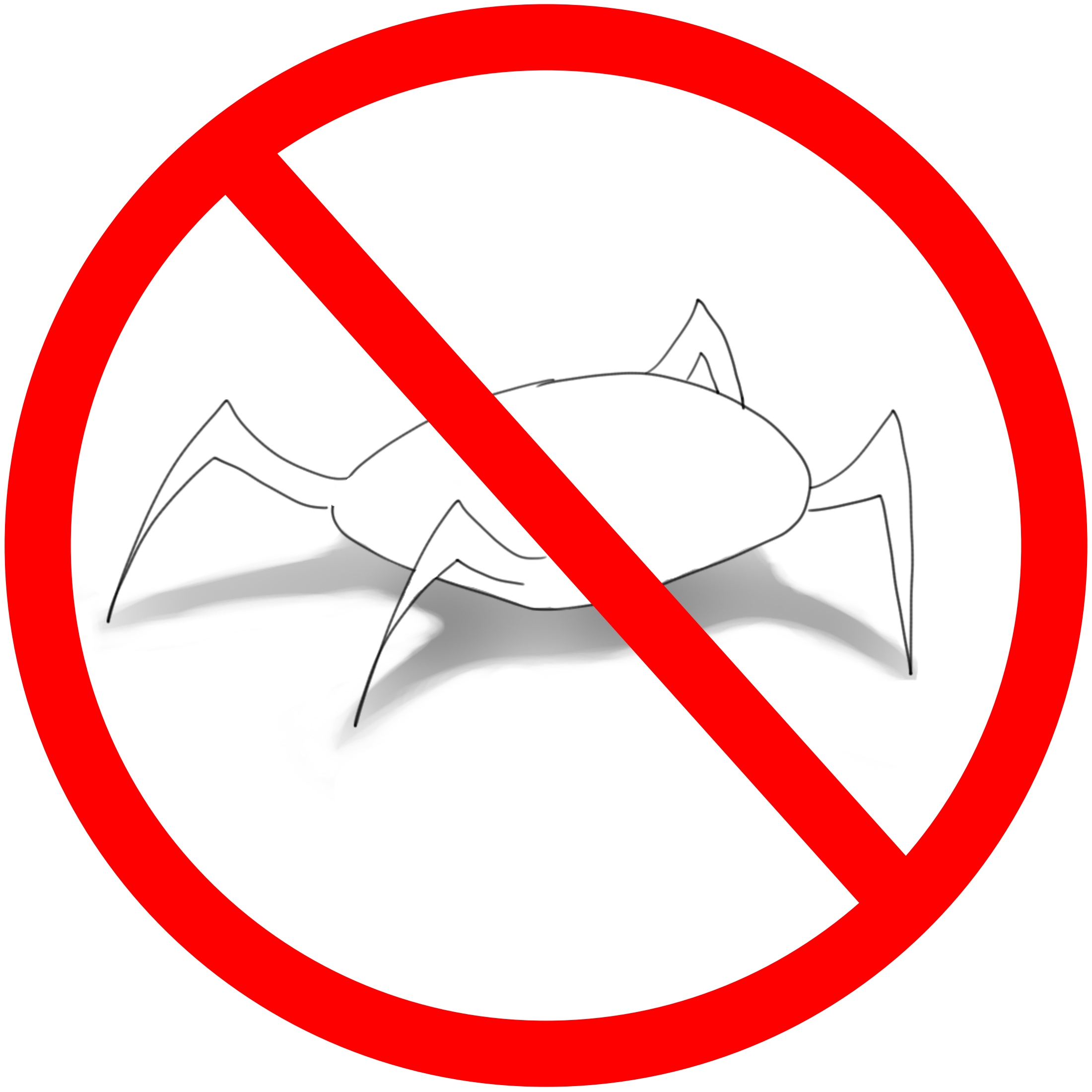
Rx

Overview

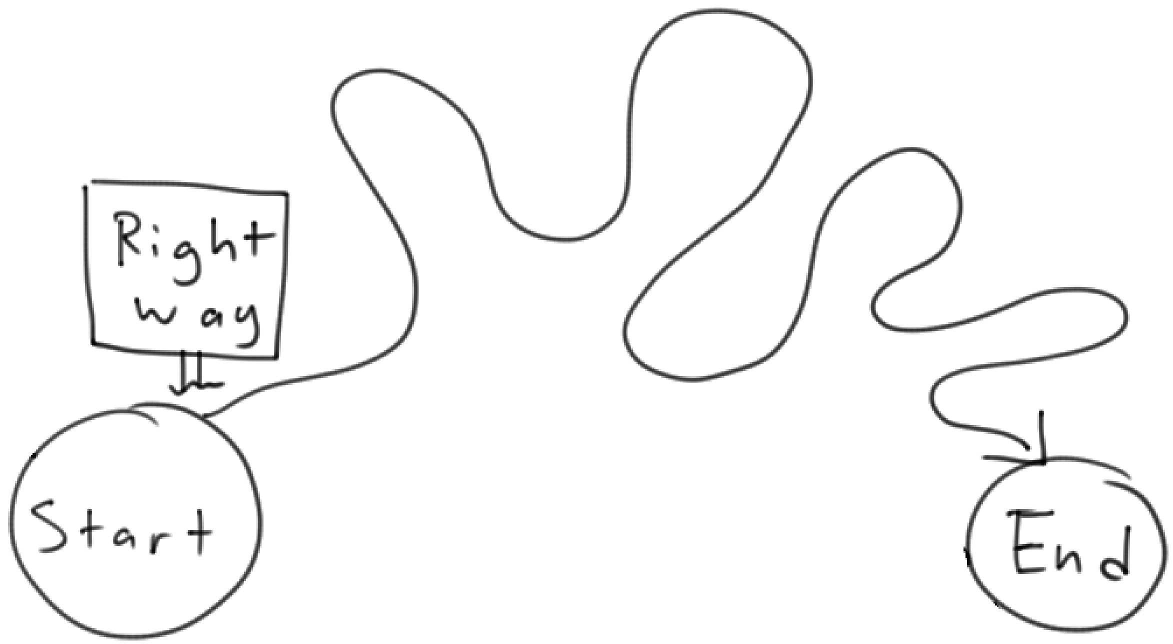
- [✓] Understand infection
  - new to system
  - bad code is referenced
- [ ] Fight infection
- [ ] Stay healthy

You must fight this infection regularly

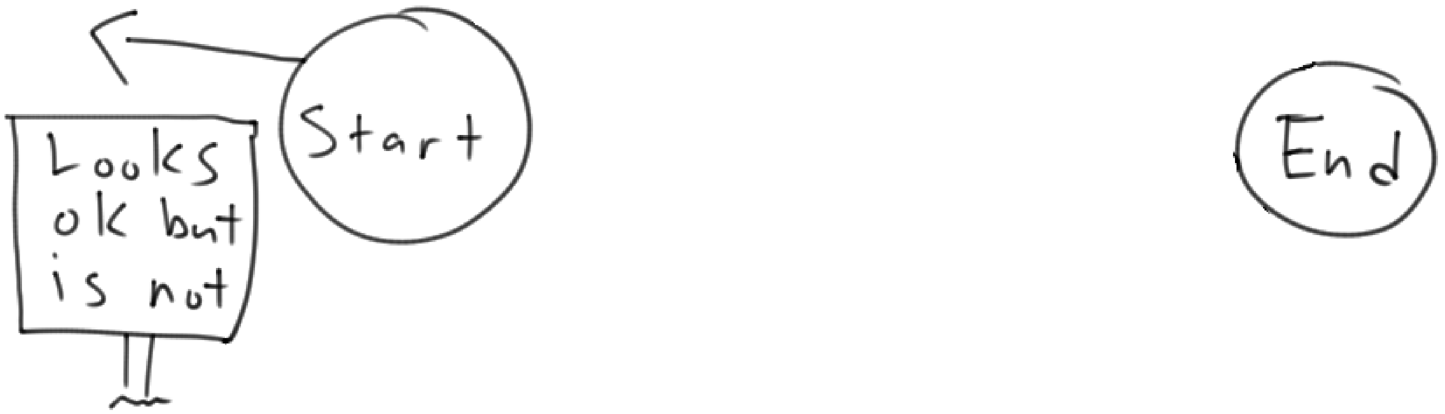
Don’t write bad code, write less optimal code



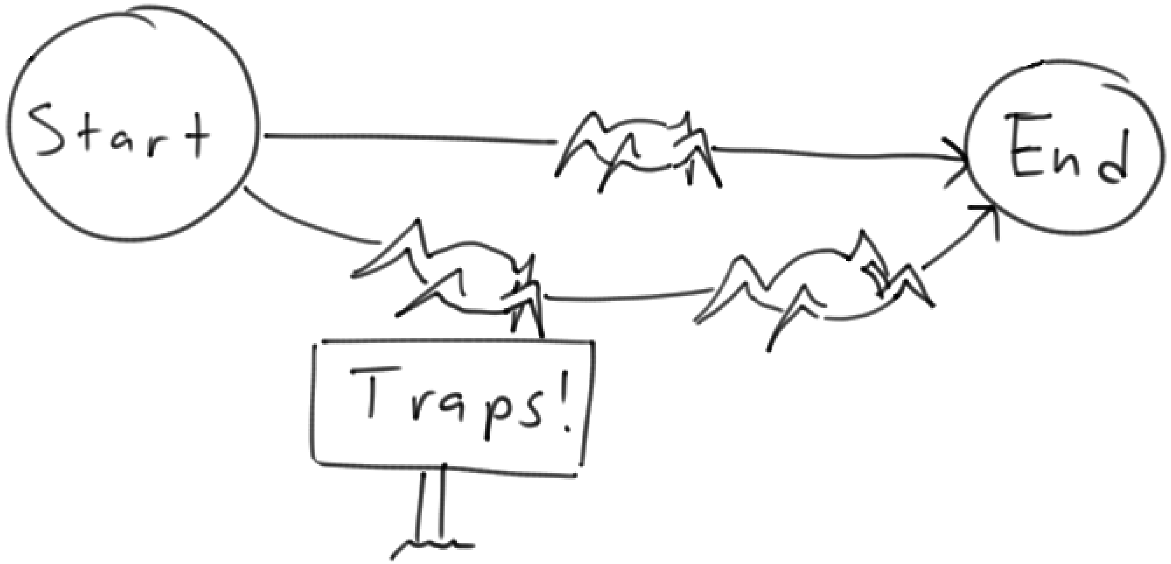
Dangerous shortcuts



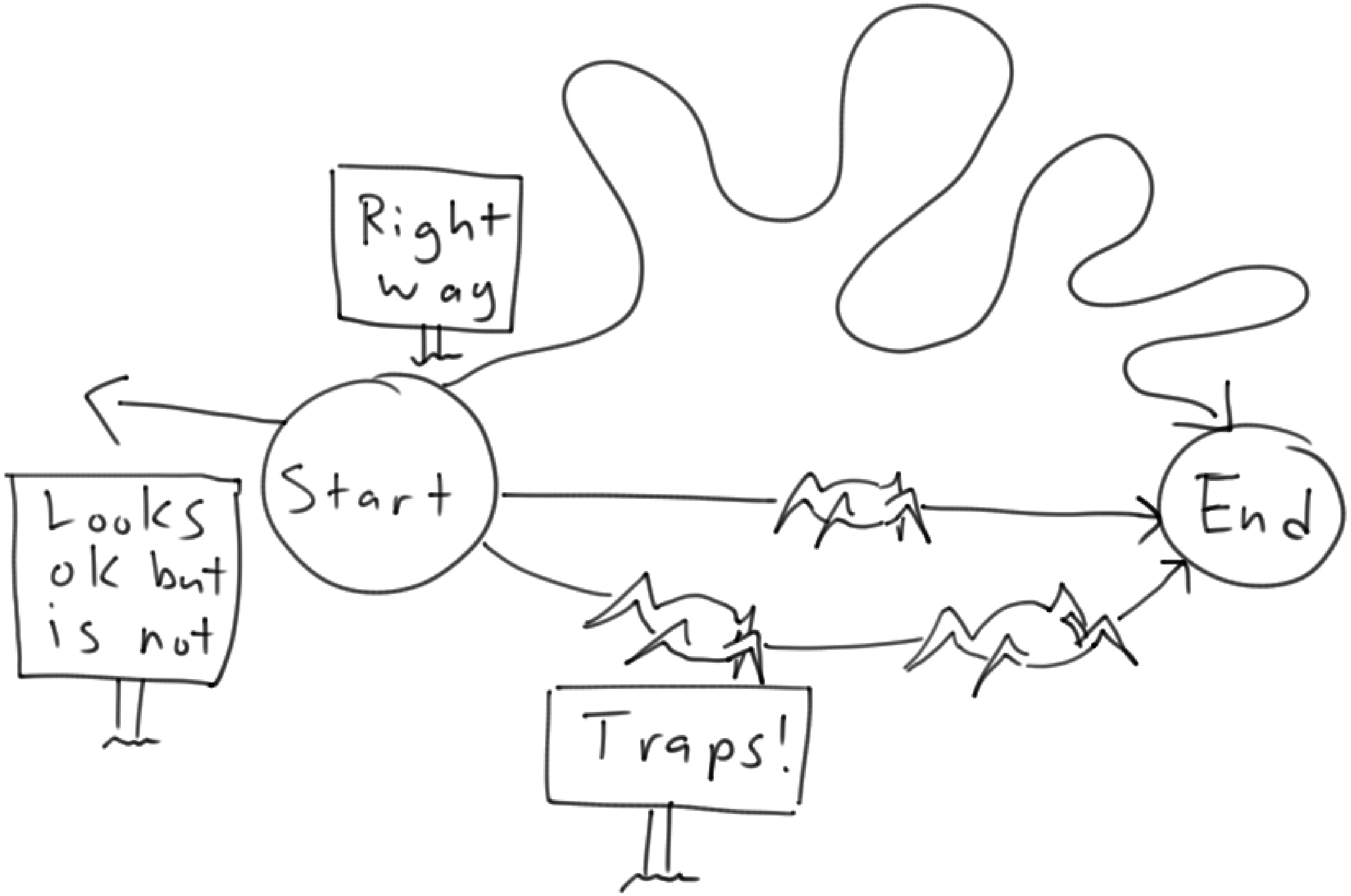
Dangerous shortcuts



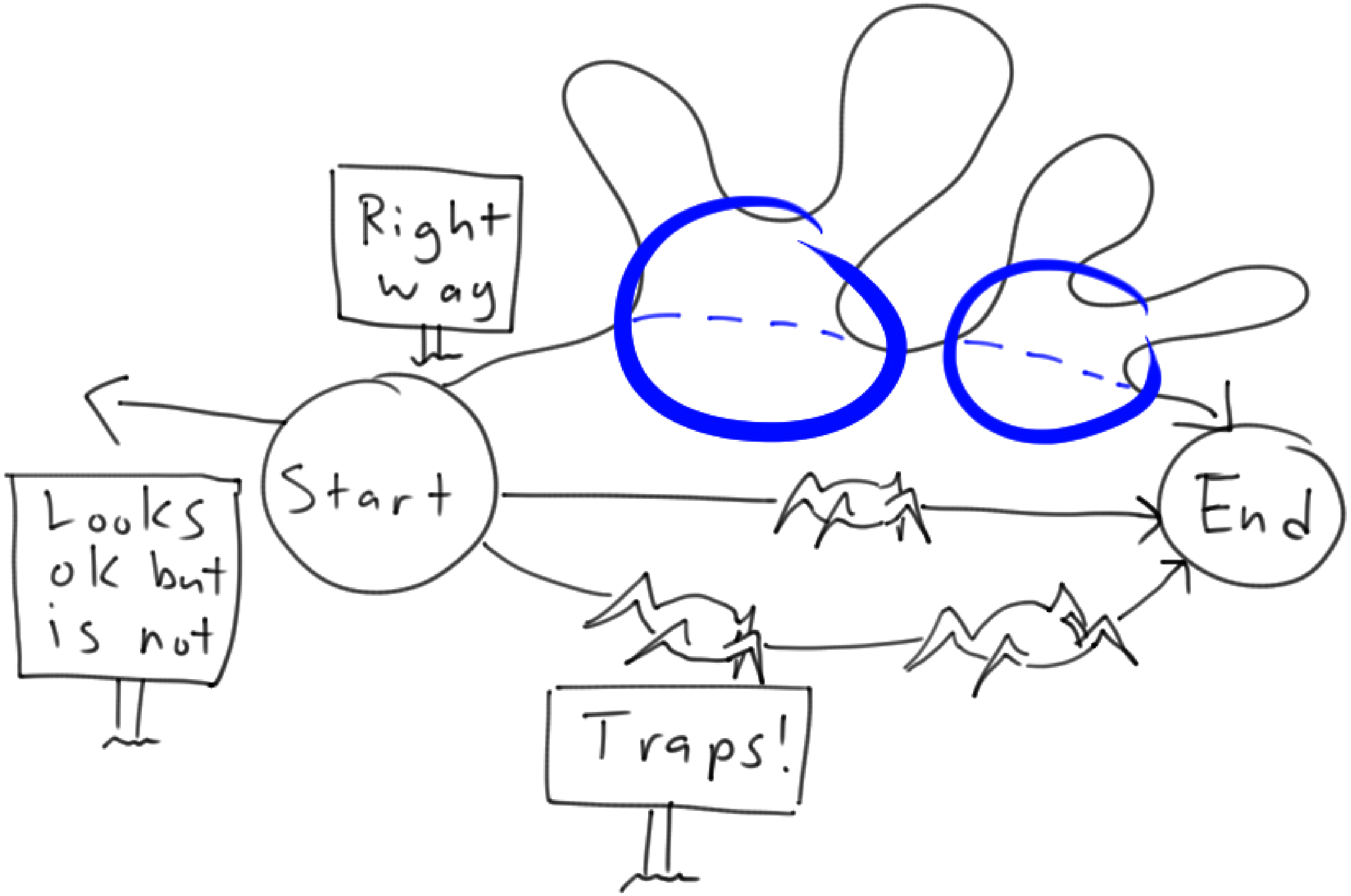
Dangerous shortcuts



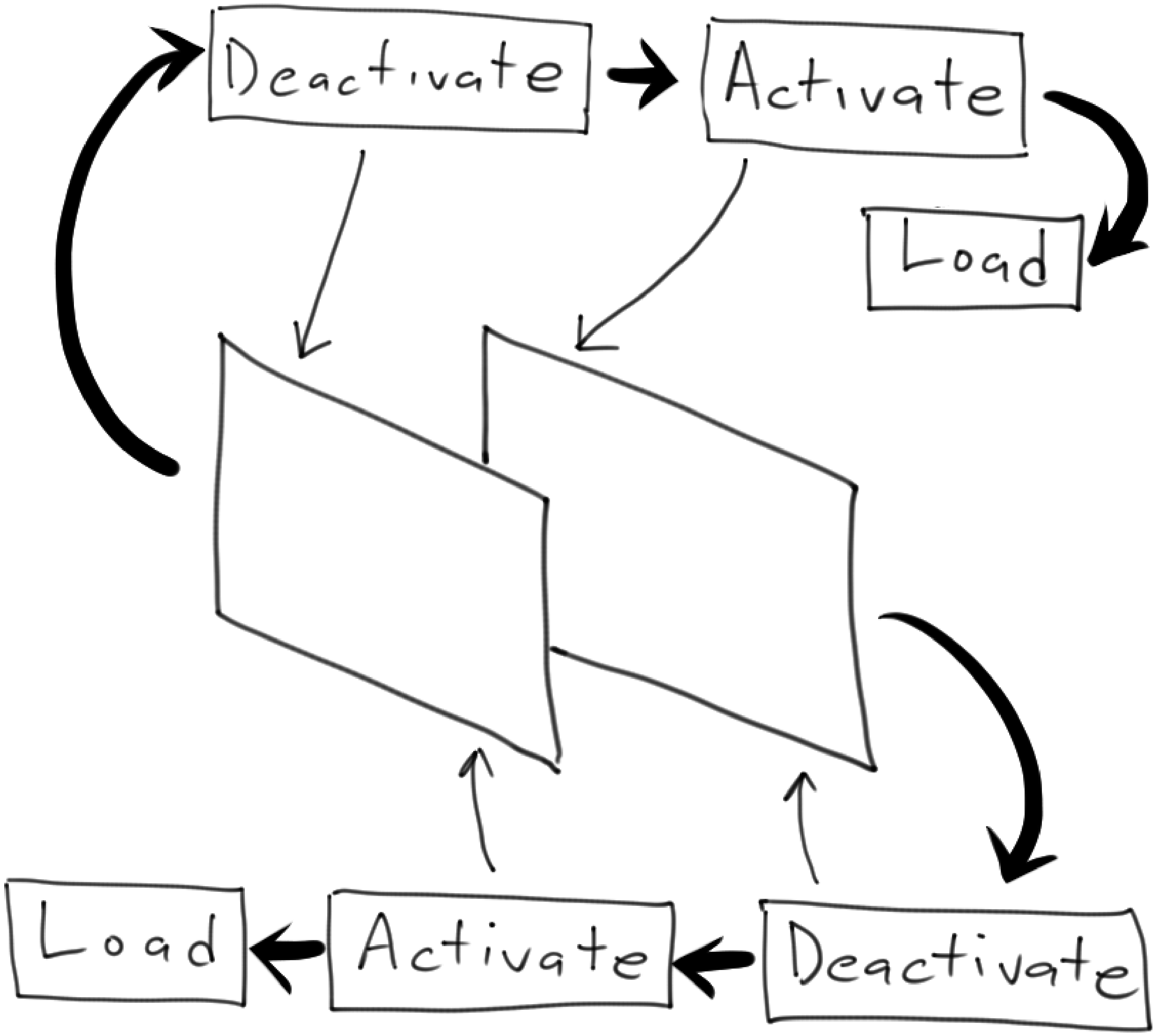
Dangerous shortcuts



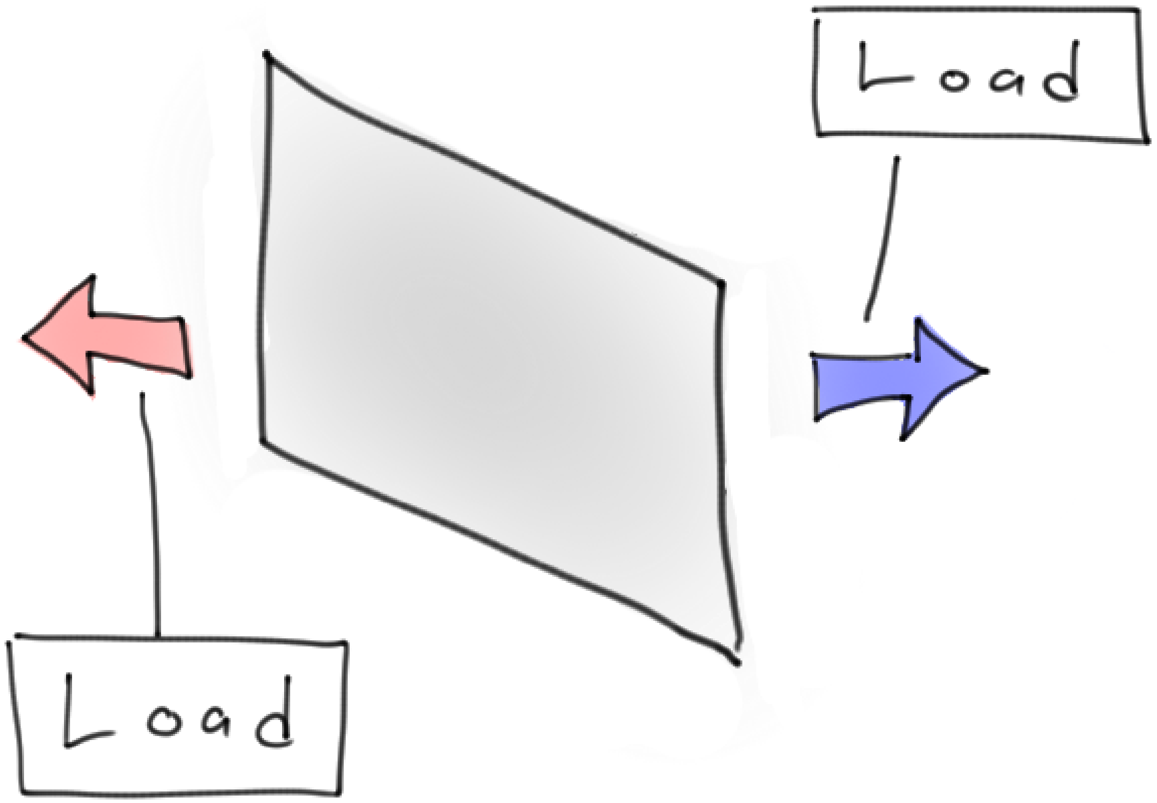
Dangerous shortcuts



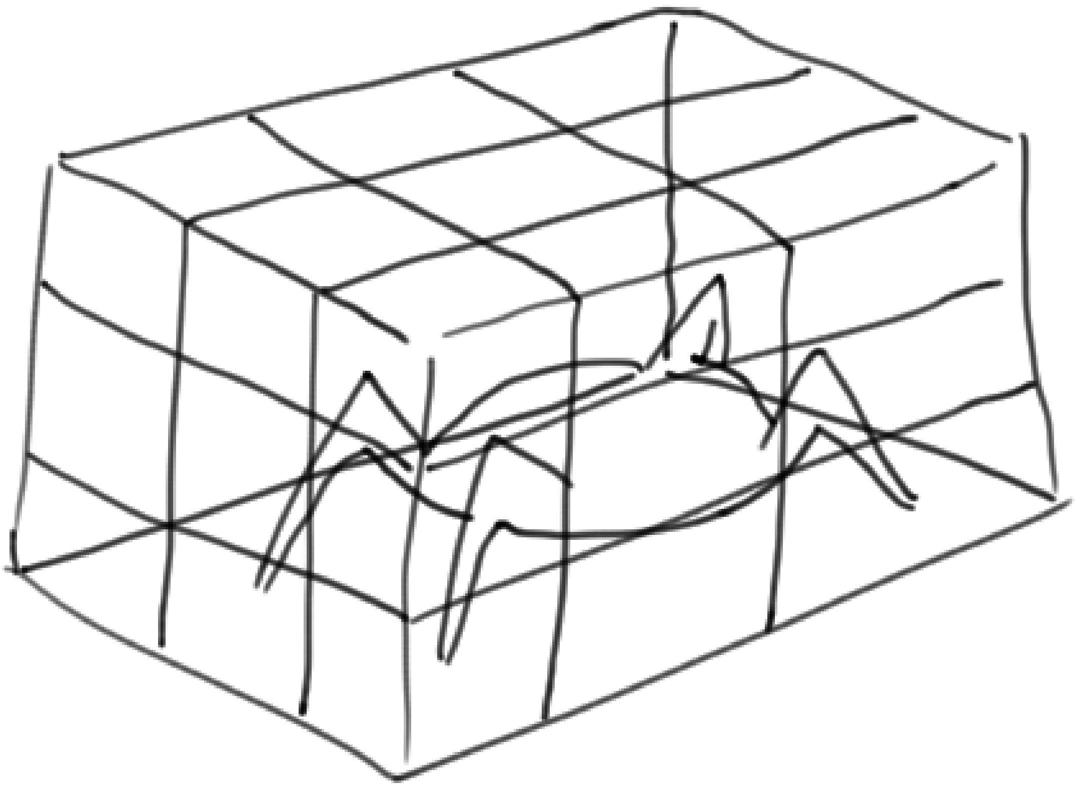
Future proofing



Future proofing



Future proofing: Compartmentalize assumptions



This is bad

```
turnSpeed = strength * 42.57
```

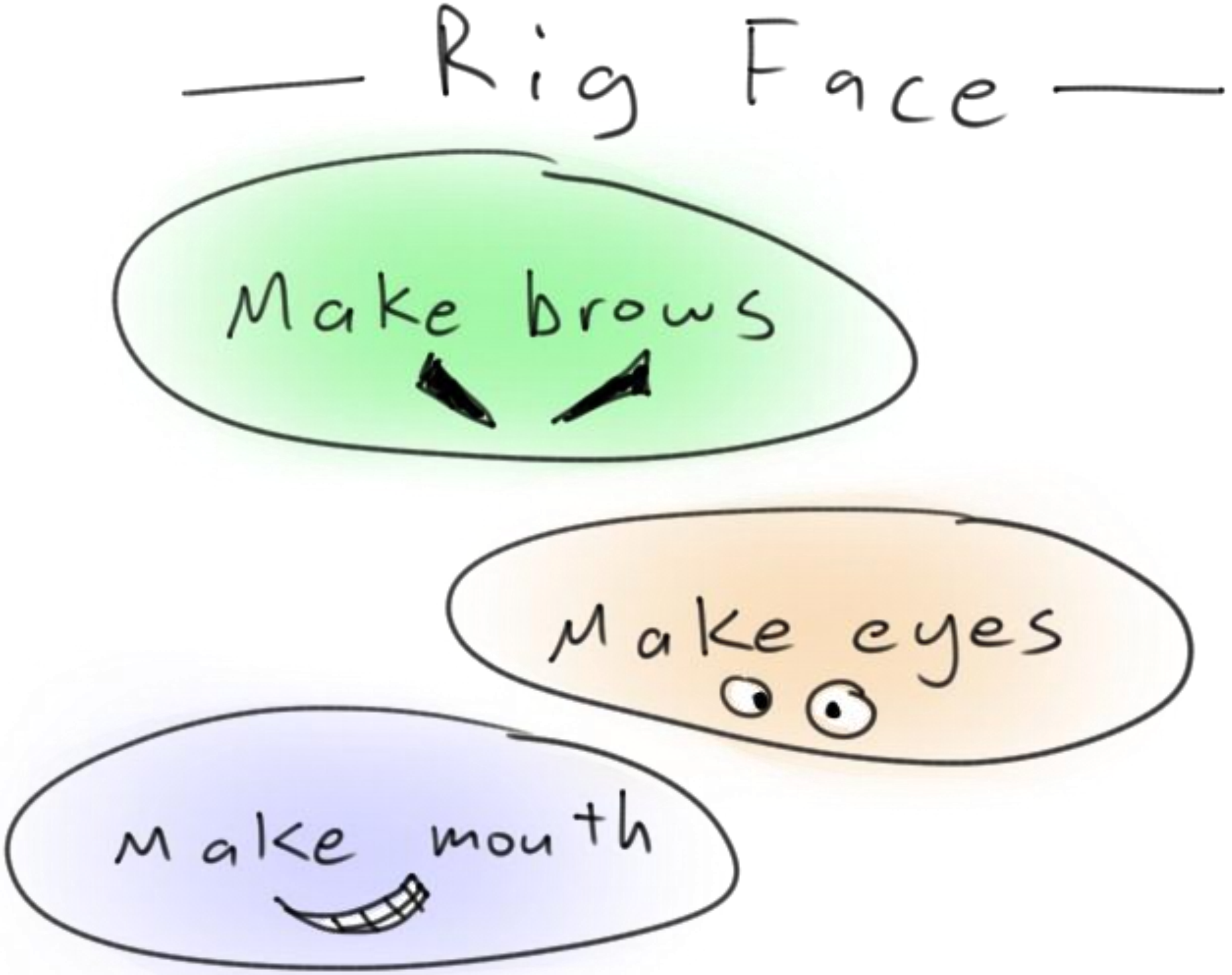
This is better

```
torsionRatio = 42.57

turnSpeed = strength * torsionRatio
```

Compartmentalize assumptions

```
def calcTorsionRatio( vehicle=None,
                      weight=None):
    # We'll leave the complex math for later.
    return 42.57

turnSpeed = strength * calcTorsionRatio()
```

Group things by responsibility



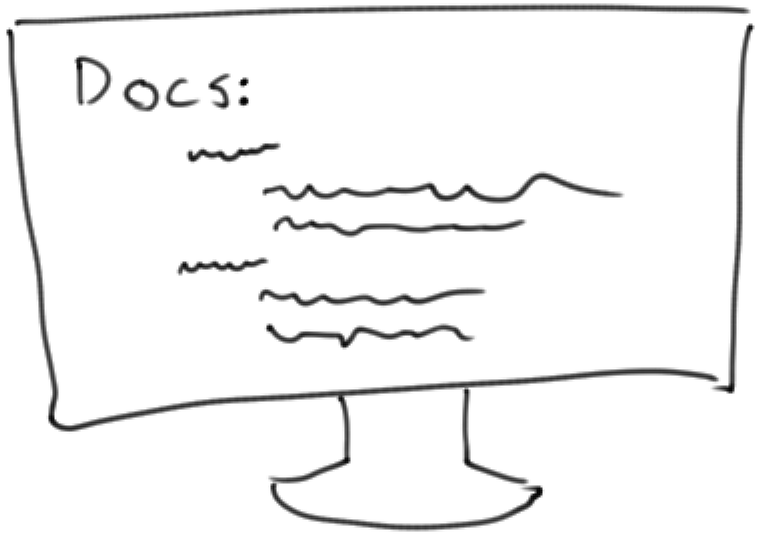
Group things by responsibility

```
def rigFace():
    browL = "browLeft"
    browR = "browRight"
    cheekL = "cheekL"
    cheekR = "cheekR"
    upperLipL = "upperLipL"
    upperLipR = "upperLipR"
    ...
    browLPos = getPosition( browL )
    browRPos = getPosition( browR )
    cheekLPos = getPosition( cheekL )
    cheekRPos = getPosition( cheekR )
    upperLipL = getPosition( upperLipL )
```

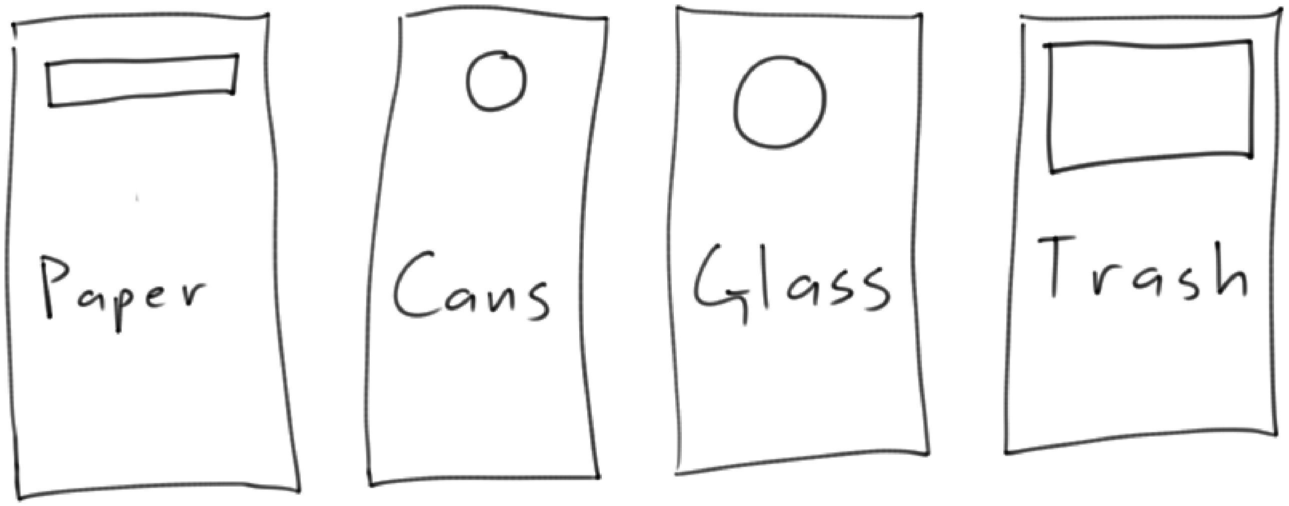
Group things by responsibility

```
def rigFace():
    browL = "browLeft"
    browLPos = getPosition( browL )
    # Code to build left brow

    browR = "browLeft"
    browRPos = getPosition( browR )
    # Code to build right brow

    cheekL = "cheekL"
    cheekLPos = getPosition( cheekL )
    # Code to build left cheek
```

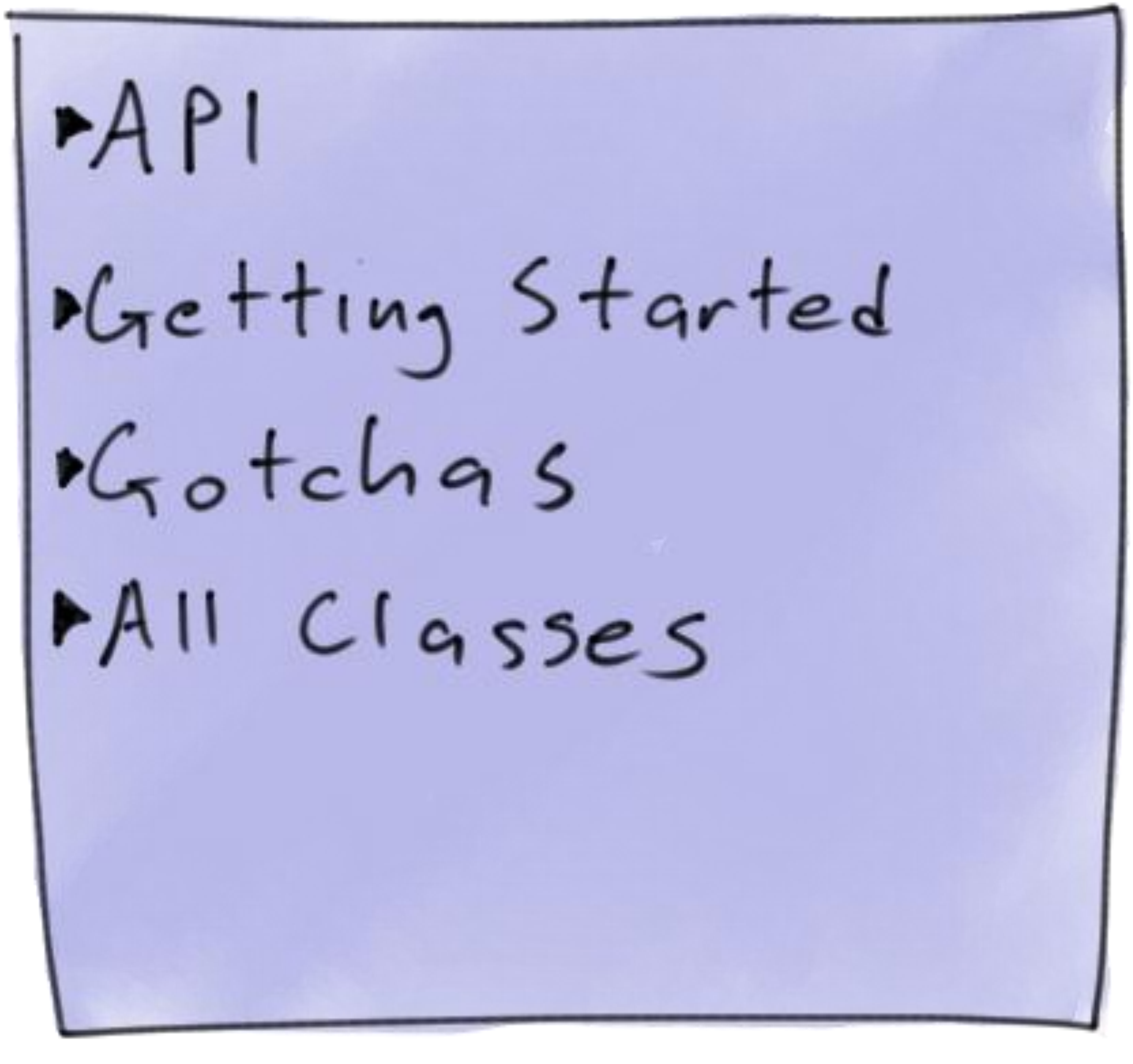
Tutorials, on boarding, documentation



Help people fit in



Documentation



- API
- Getting Started
- Gotchas
- All Classes

Documentation

```
def updateEmitterCloud():
    ...

def updateCloudEmitter():
    ...
```

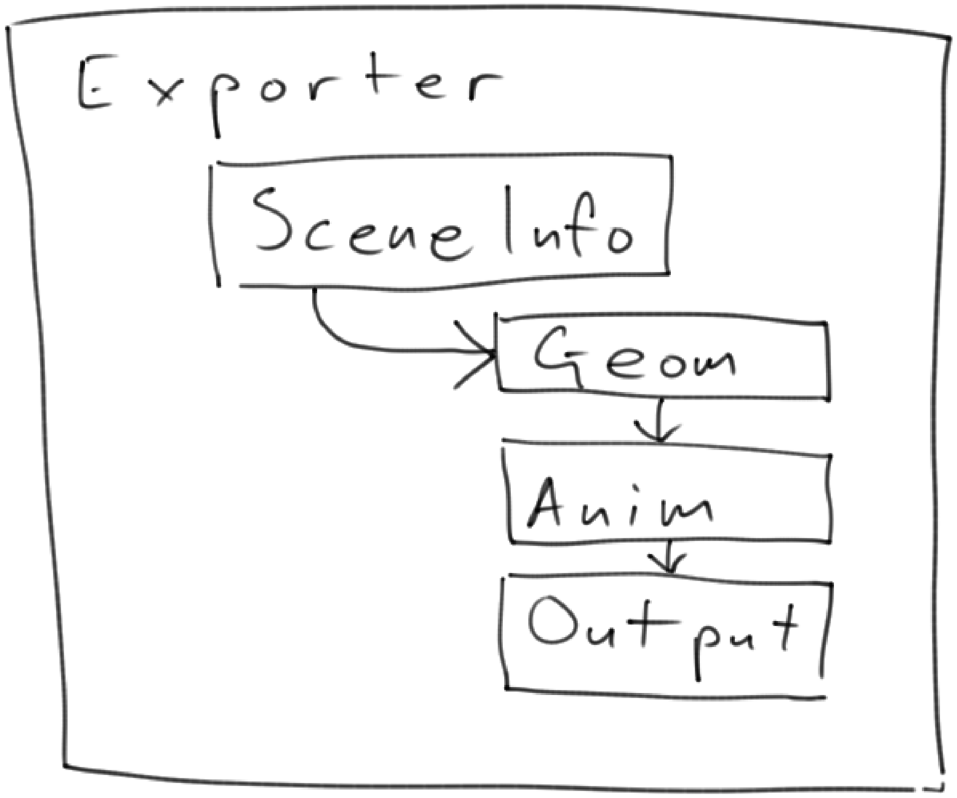
Documentation

```
def getCollision(obj):
    for child in listRelatives(obj):
        if child.name() == 'collision':
            return child
```

Documentation

```
def getCollision(obj):
    '''
    Return collision or None if not
  found.
    '''
    for child in listRelatives(obj):
        if child.name() == 'collision':
            return child
```

Good documentation lets new people see the whole system.



Documentation

```
class Vector(VectorN):
  def __init__(self, *args, **kwargs):
    if args:
      if len(args)==1 and hasattr(args[0], '__iter__'):
        args = args[0]
      try:
        self.assign(args)
      except:
        if isinstance(args, _api.MPoint) and args.w != 1.0:
          args = copy.deepcopy(args).cartesianize()
        if isinstance(args, _api.MColor) and args.a != 1.0:
          pass
        if isinstance(args, _api.MVector):
          args = tuple(args)
```

This is a dumb comment

```
// increment i
i++;
```

This is a good comment

```
# Go through collision largest
# to smallest
for col in reversed(getCollision()):
    ...
```

This is a good comment

```
# Get the short name without namespace
name.rsplit('|',1)[-1].rsplit(':',1)[-1]
```

This is kind of awkward

```
shortNameNoNamespace =
    name.rsplit('|',1)[-1].rsplit(':',1)[-1]
```

This is really awful

```
shrtNameNoNs =
    name.rsplit('|',1)[-1].rsplit(':',1)[-1]
```

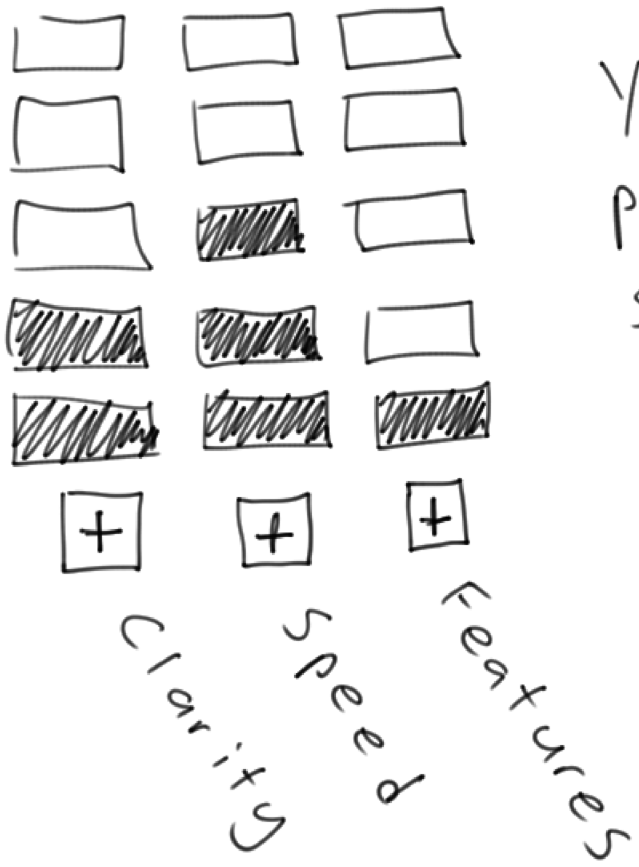
So much nicer!

```
# Get the short name without namespace
name.rsplit('|',1)[-1].rsplit(':',1)[-1]
```

Sometimes hacks are required

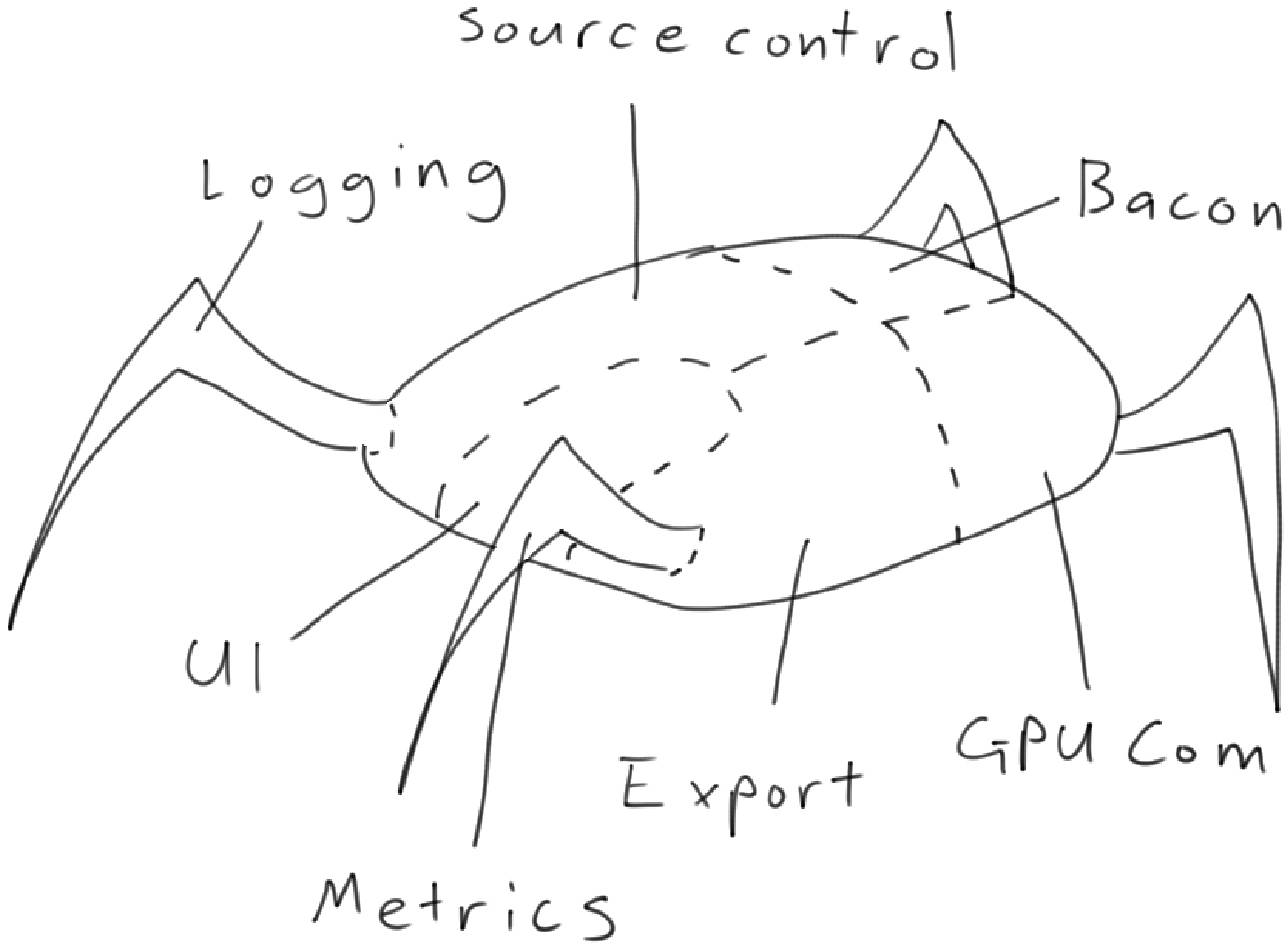
```
#       x x                       x x
#     x     x                   x     x
#   x   x   x                   x   x   x
#   x         x     x x x x   x         x
#     x x       x x         x x     x x
#         x     x             x   x
#           x x                 x    WARNING:  The buffer
#             x   x x     x x   x    MUST be accessed
#             x   x x     x x   x    directly, backwards,
#             x                 x    due to a bug in
#             x x             x x    the API ...
#           x     x         x     x
#       x x     x x         x x     x x
#     x       x   x         x   x       x
#     x x   x     x   x   x       x   x x
#       x x       x x x x x x     x x x
```

Refactor your code

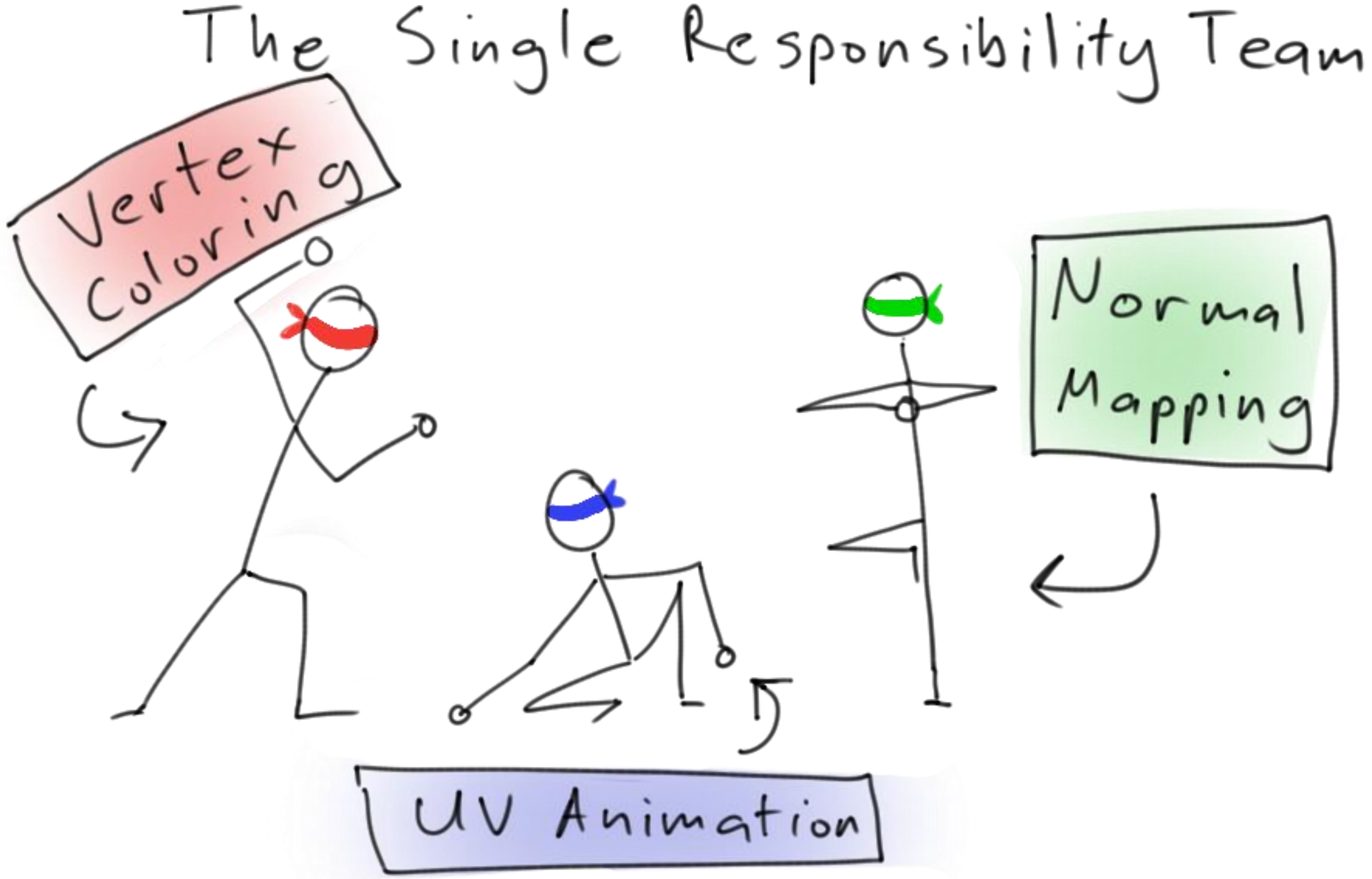


You have $\boxed{2}$ points to spend

Breakdown monolithic classes/functions



By single responsibilities



Separate functionality and gui

Functionality
GUI

Separate functionality and gui

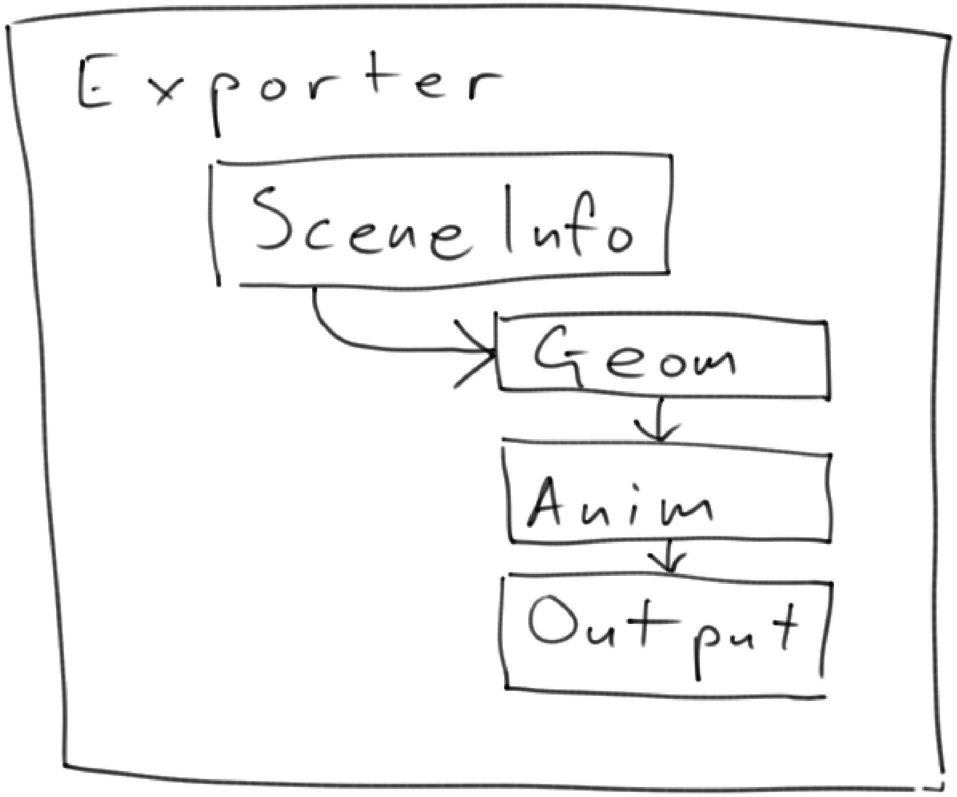
Separate functionality and gui

```
def onClick():
    for obj in selected():
        # Make LODs
```

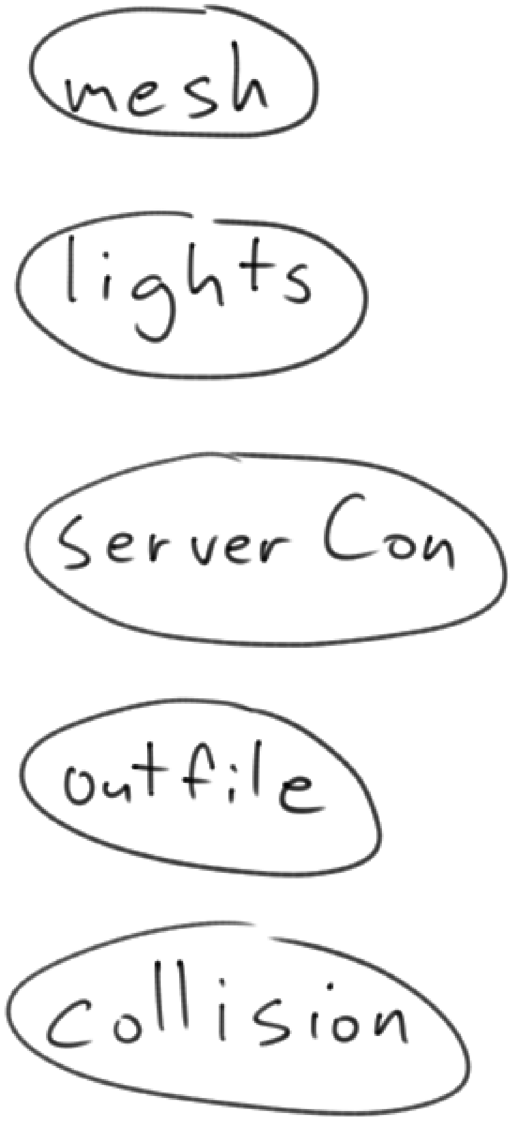
Separate functionality and gui

```
def onClick():
    makeLODs( selected() )

def makeLODs(objects):
    # do the real work
```

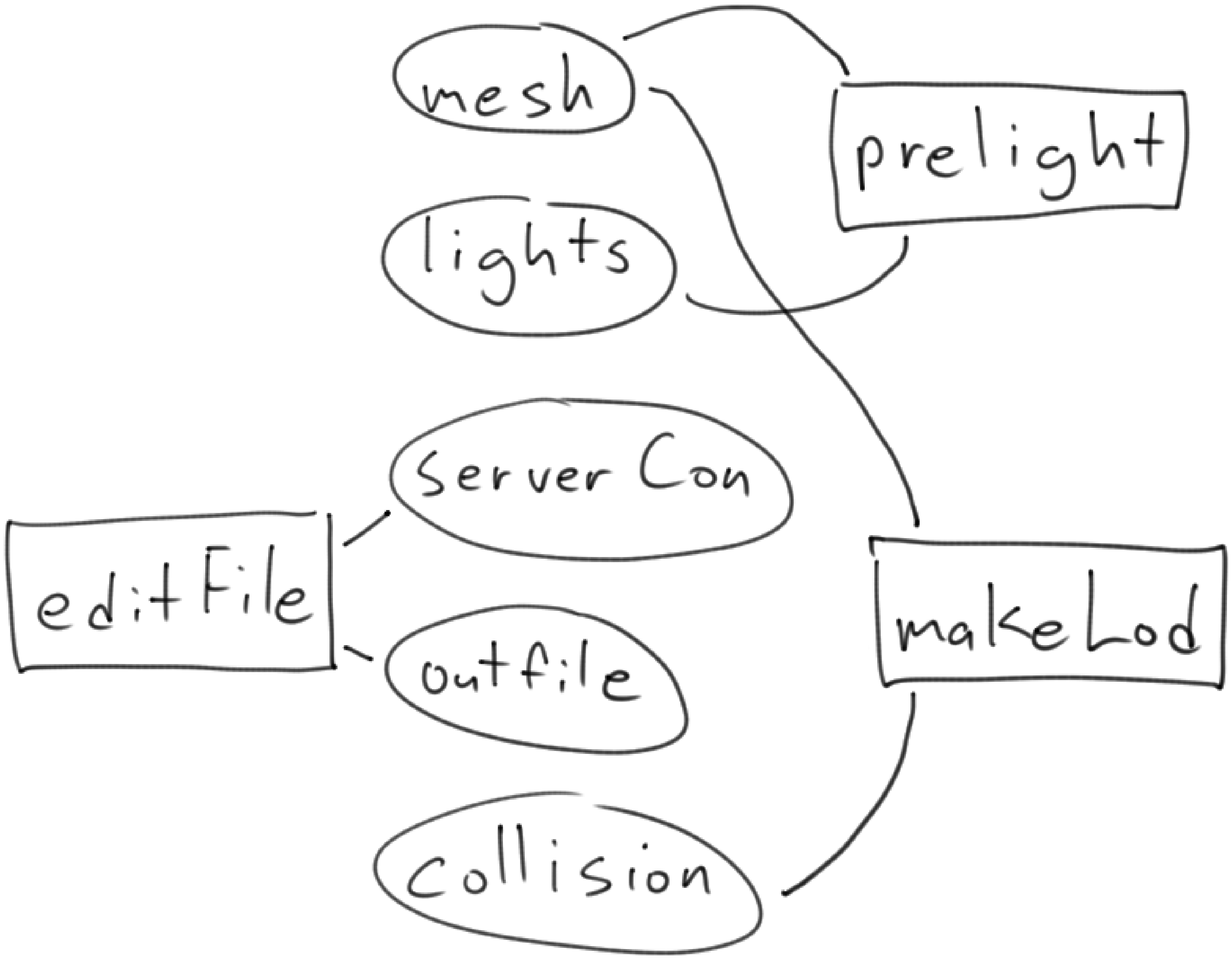
Draw to understand



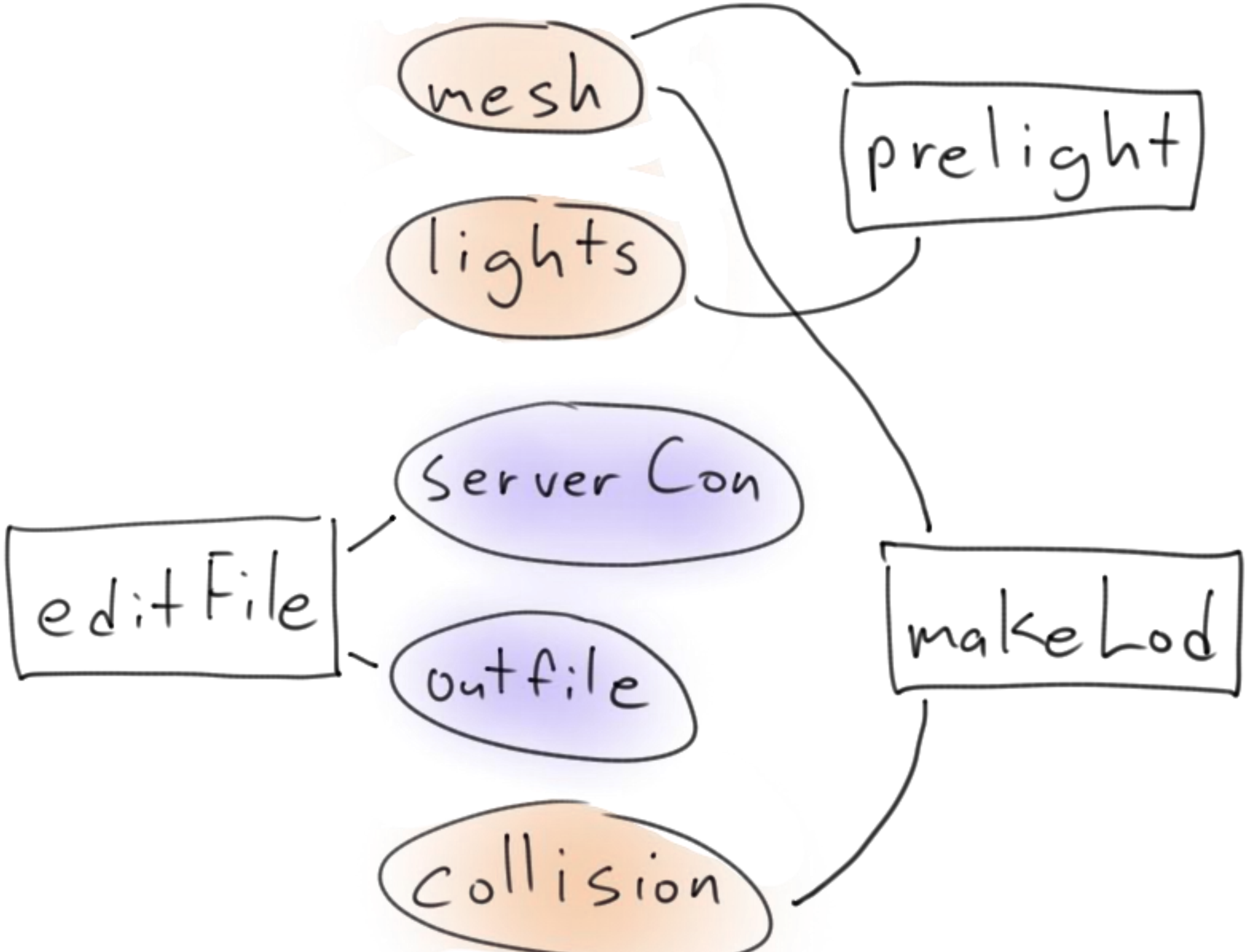
Draw to understand



Draw to understand



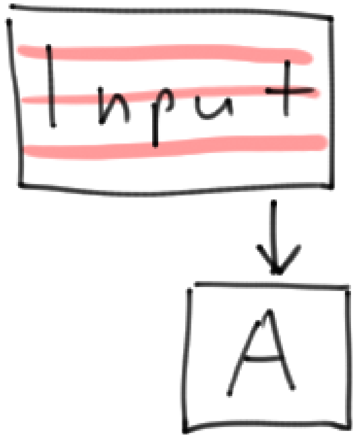
Draw to understand



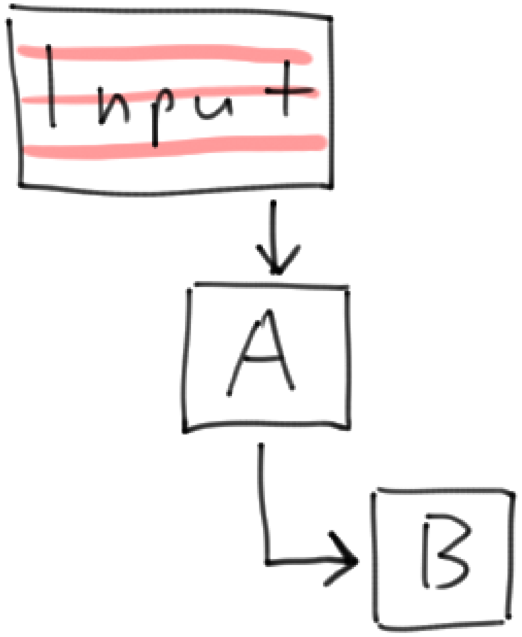
Globals

```
# Keep track of exported
gModels    = [ ]
gCollision = [ ]
```

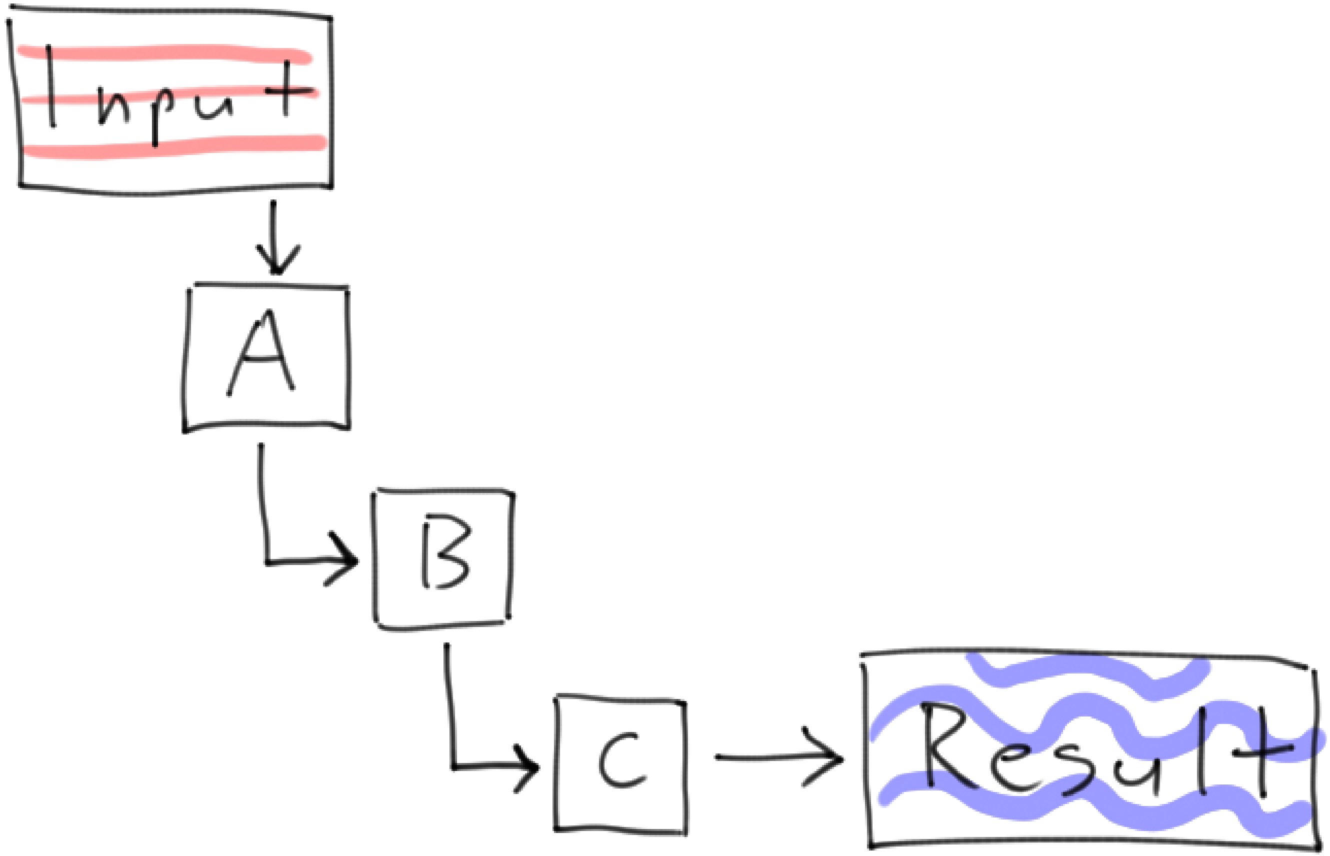
Why globals can be bad



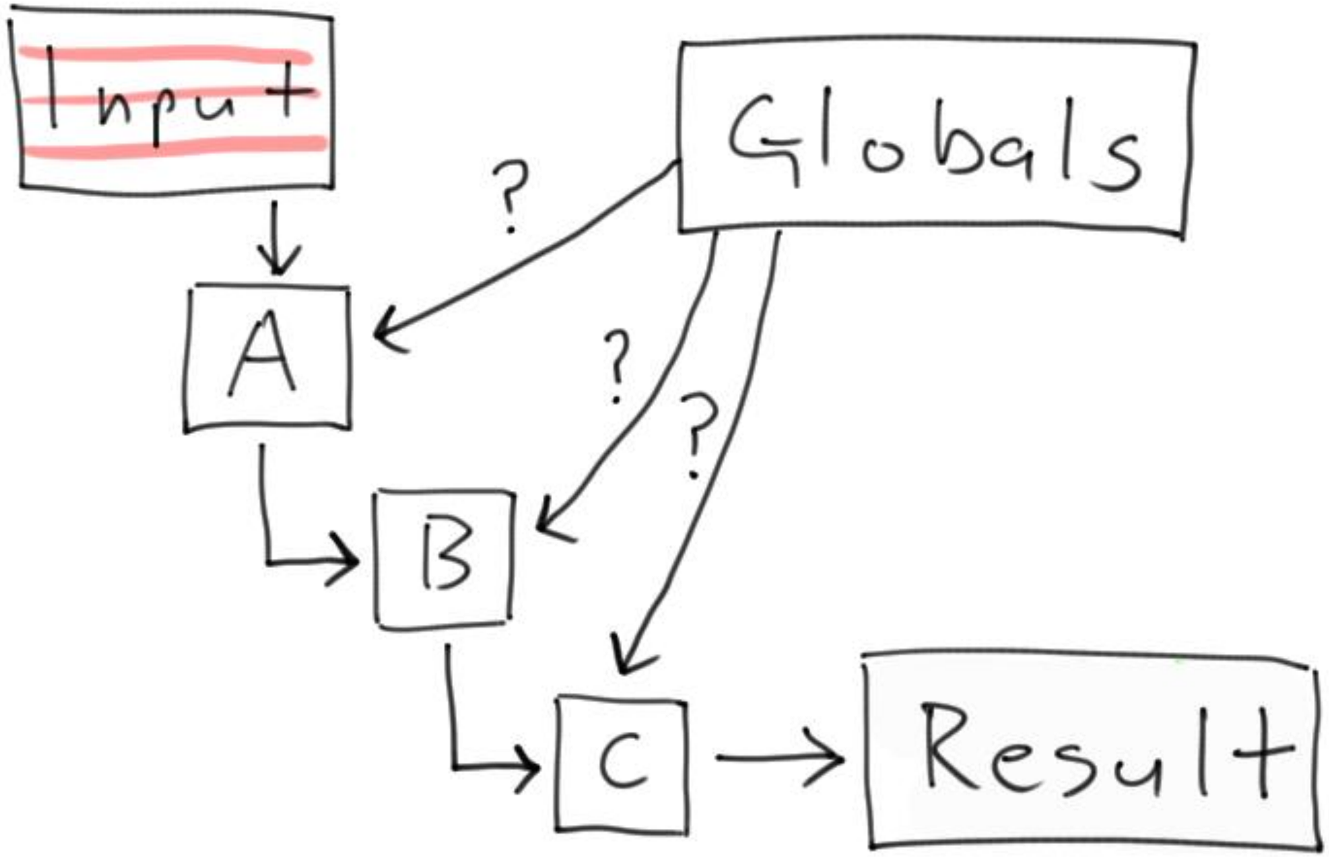
Why globals can be bad



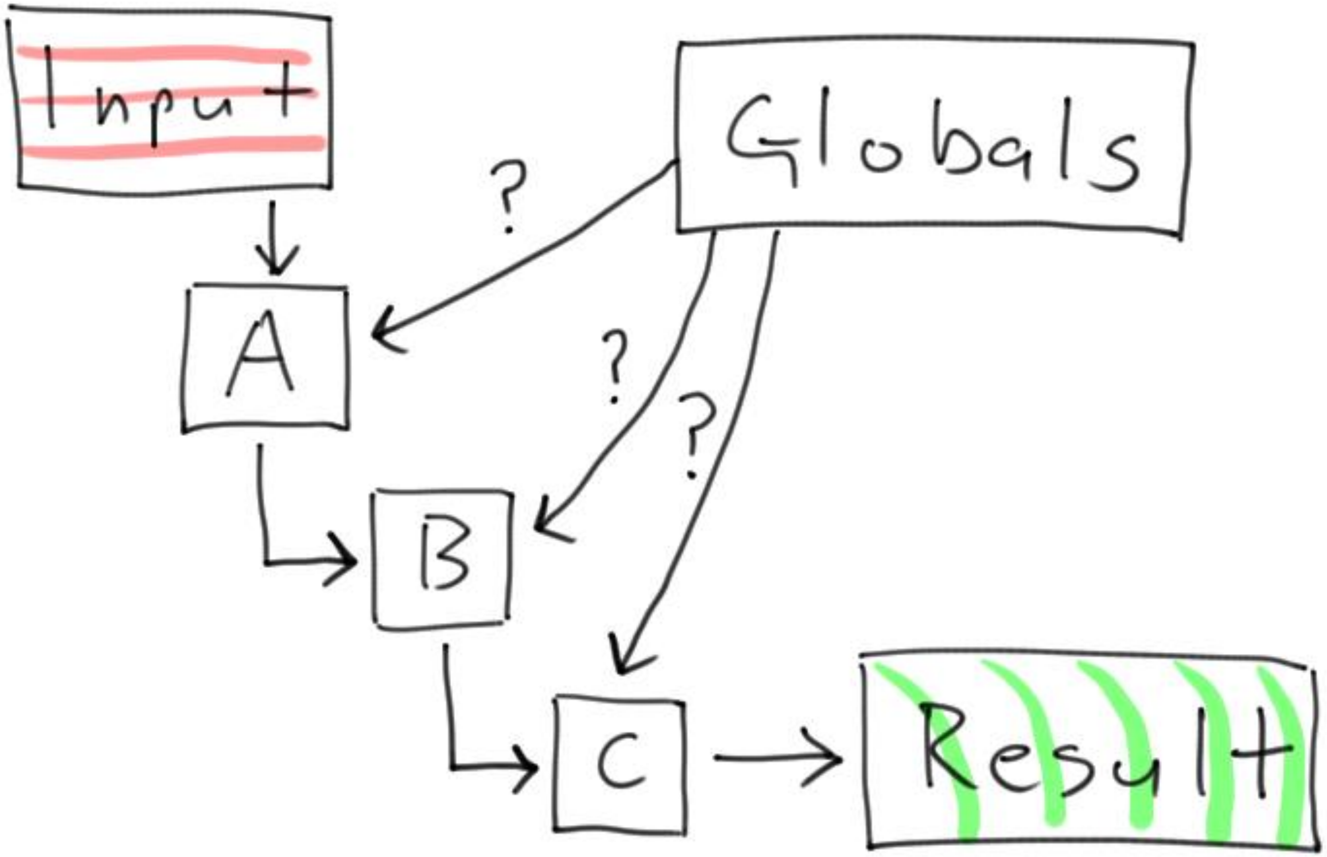
Why globals can be bad



Why globals can be bad



Why globals can be bad



Why globals can be good

```
def a(obj):
    b(obj) # Pass obj along

def b(obj):
    c(obj) # Just keep passing it along
...
def g(obj):
    h(obj) # And passing it some more

def h(obj):
    # Finally do something with obj
    size = obj.size
```

~~Global~~ Scope Issues

```
class Exporter():
    def __init__(self):
        self.meshes = []
        self.collision = []

    def a(self):
        # meshes and collision set here

    def h(self):
        # meshes and collision used here
```

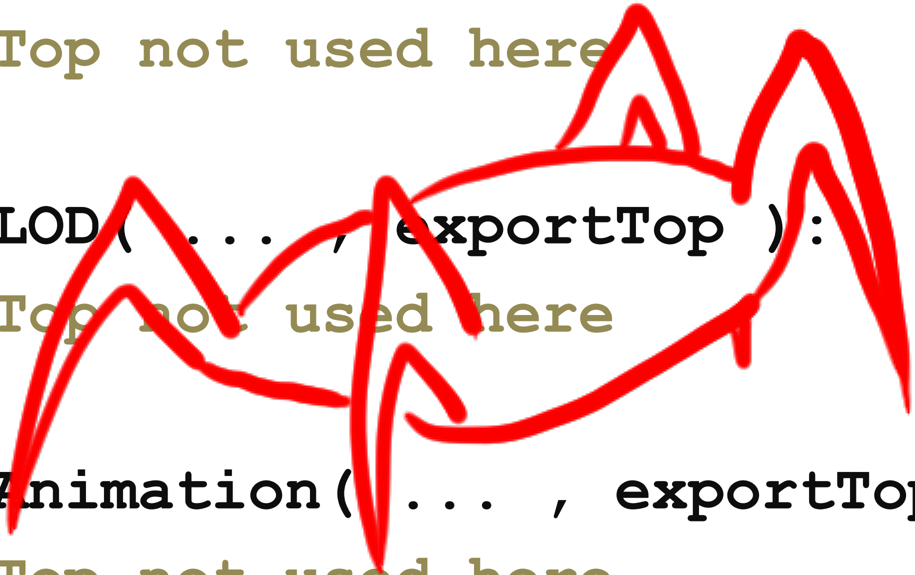
~~Global~~ Scope Issues

```
def prepMesh( ... , exportTop ):
    # exportTop not used here

def combineCollision( ... , exportTop ):
    # exportTop not used here

def generateLOD( ... , exportTop ):
    # exportTop not used here

def resampleAnimation( ... , exportTop ):
    # exportTop not used here
```

~~Global~~ Scope Issues

Lame!

```
def prepMesh( ... , exportTop ):
    # exportTop not used here

def combineCollision( ... , exportTop ):
    # exportTop not used here

def generateLOD( ... , exportTop ):
    # exportTop not used here

def resampleAnimation( ... , exportTop ):
    # exportTop not used here
```



~~Global~~ Scope Issues

```
class ExporterManager():
    def __init__(self):
        # Easy to access as needed
        self.outputFiles = []

    def determineOutput(self):
        # Clear setting mechanism
        # set self.outputFiles
```

Rx

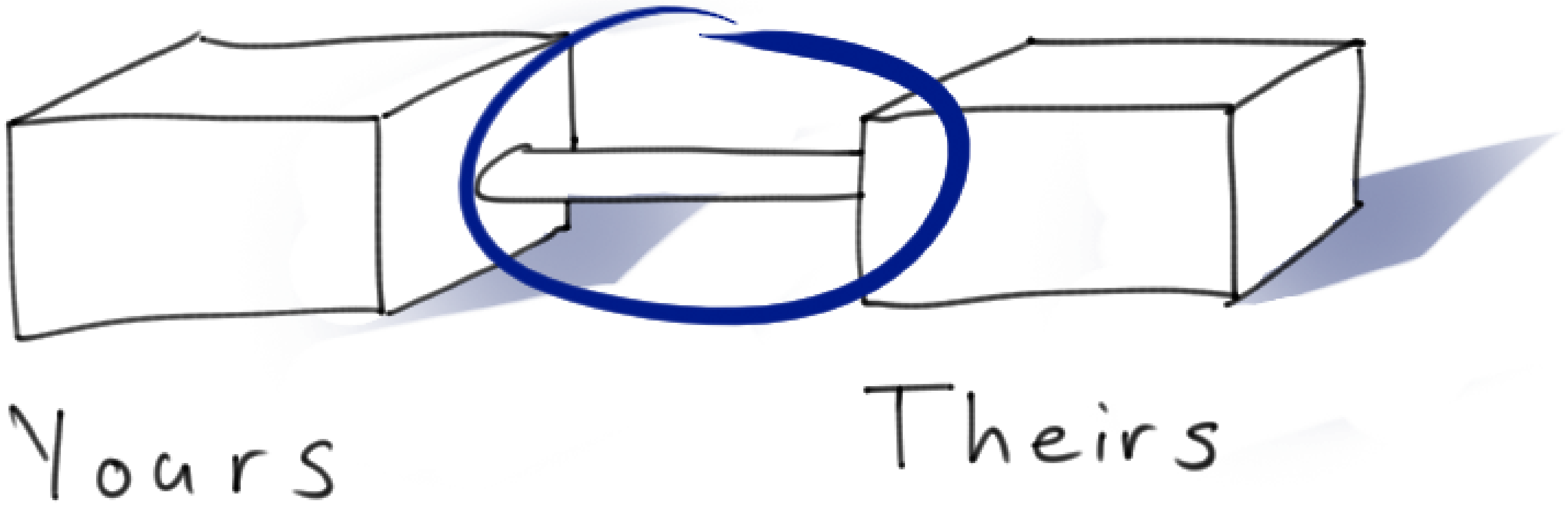
- [x] Understand infection
  - new to system
  - bad code is referenced
- [ ] Fight infection
  - smarter shortcuts allow future growth
  - tutorials + docs
  - Refactoring
- [ ] Stay healthy

Rx

- [x] Understand infection
  - new to system
  - bad code is referenced
- [x] Fight infection
  - smarter shortcuts allow future growth
  - tutorials + docs
  - Refactoring
- [ ] Stay healthy

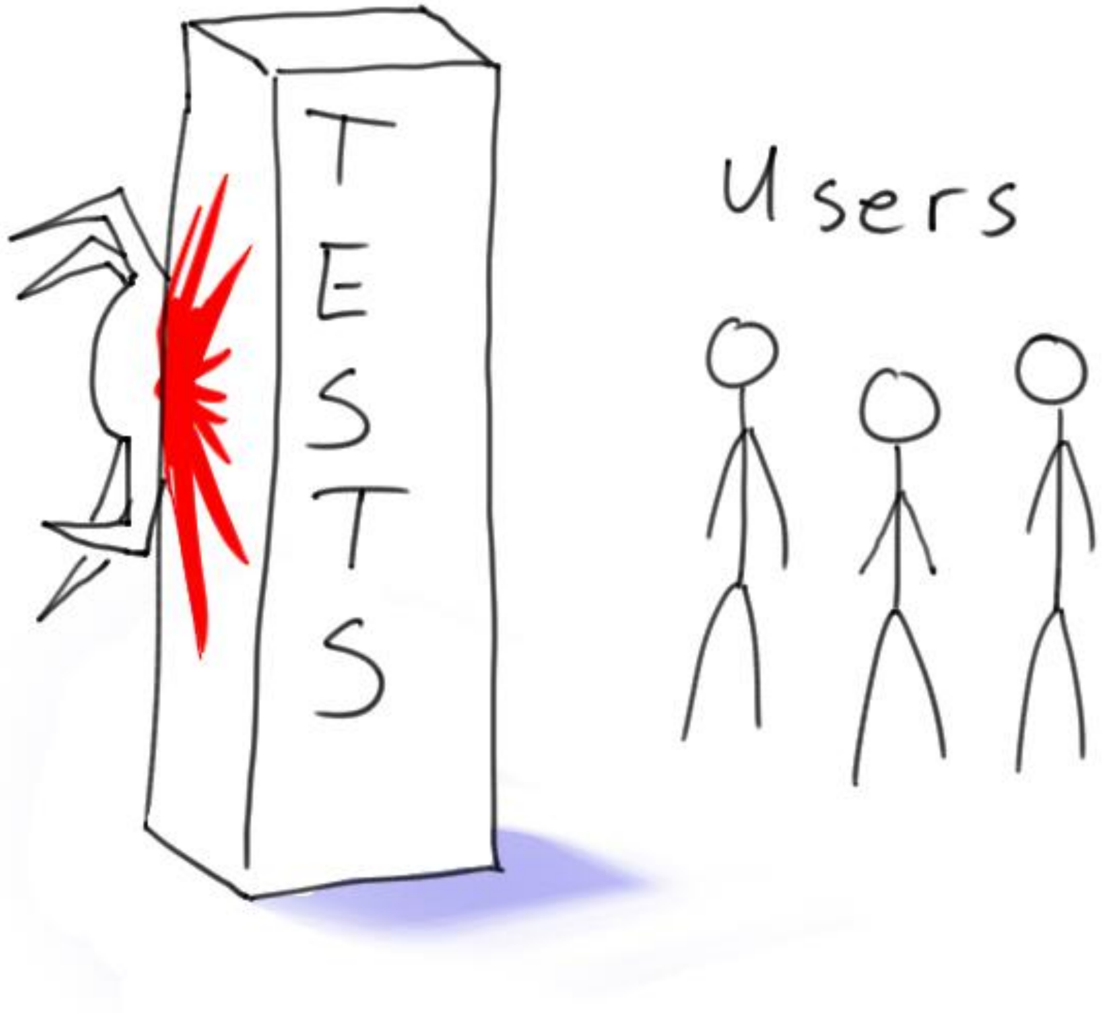
Testing will save you

Testing

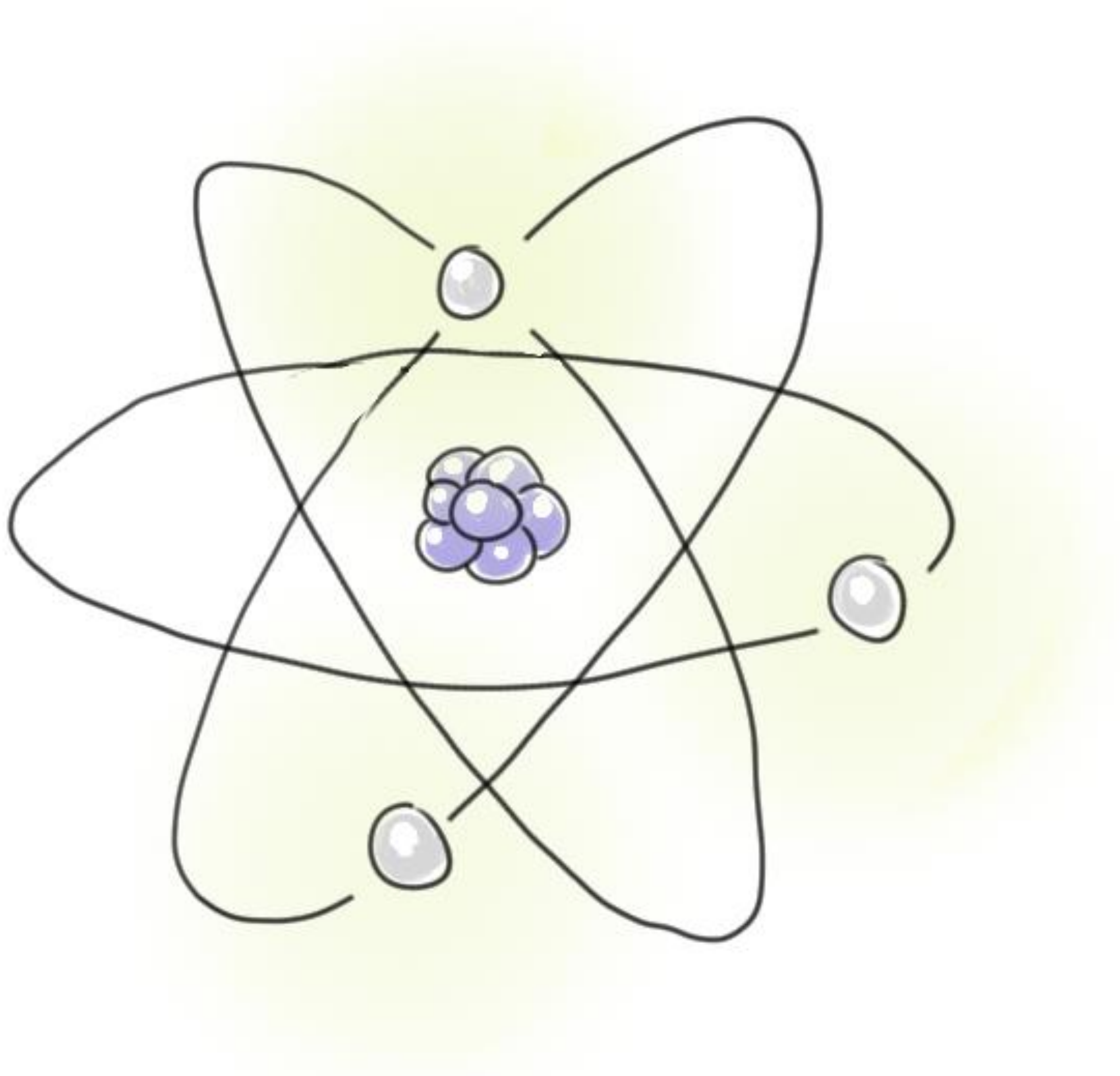


Automate the testing

Automate the testing



Unit tests

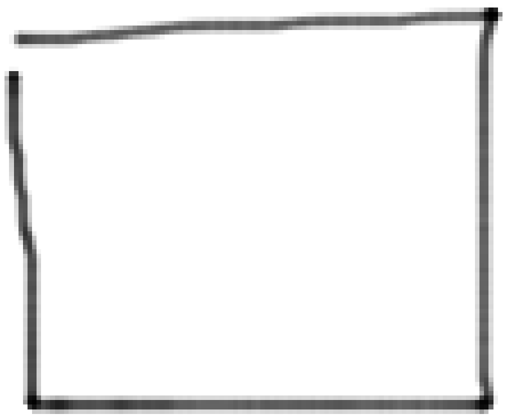


Unit tests

```
def test_returnsNoneWithNoCollision():

def test_findsValidCollision():

def test_errorOnMultipleCollision():
```

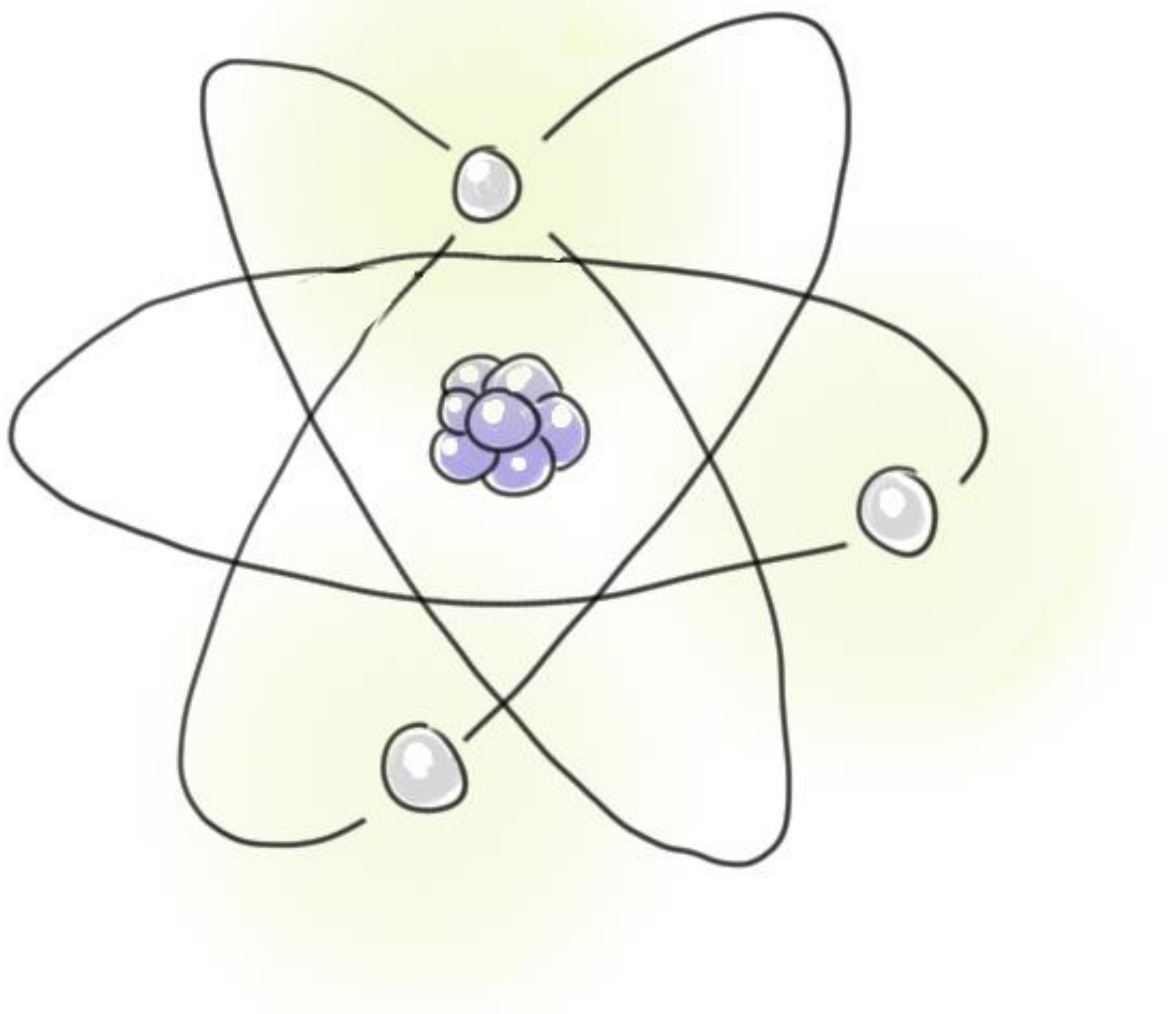
Testable Code = Better Code

Automated testing



```
pendown
right 90
draw 15
right 90
draw 12
right 90
draw 15
right 90
draw 12
```

Automated testing



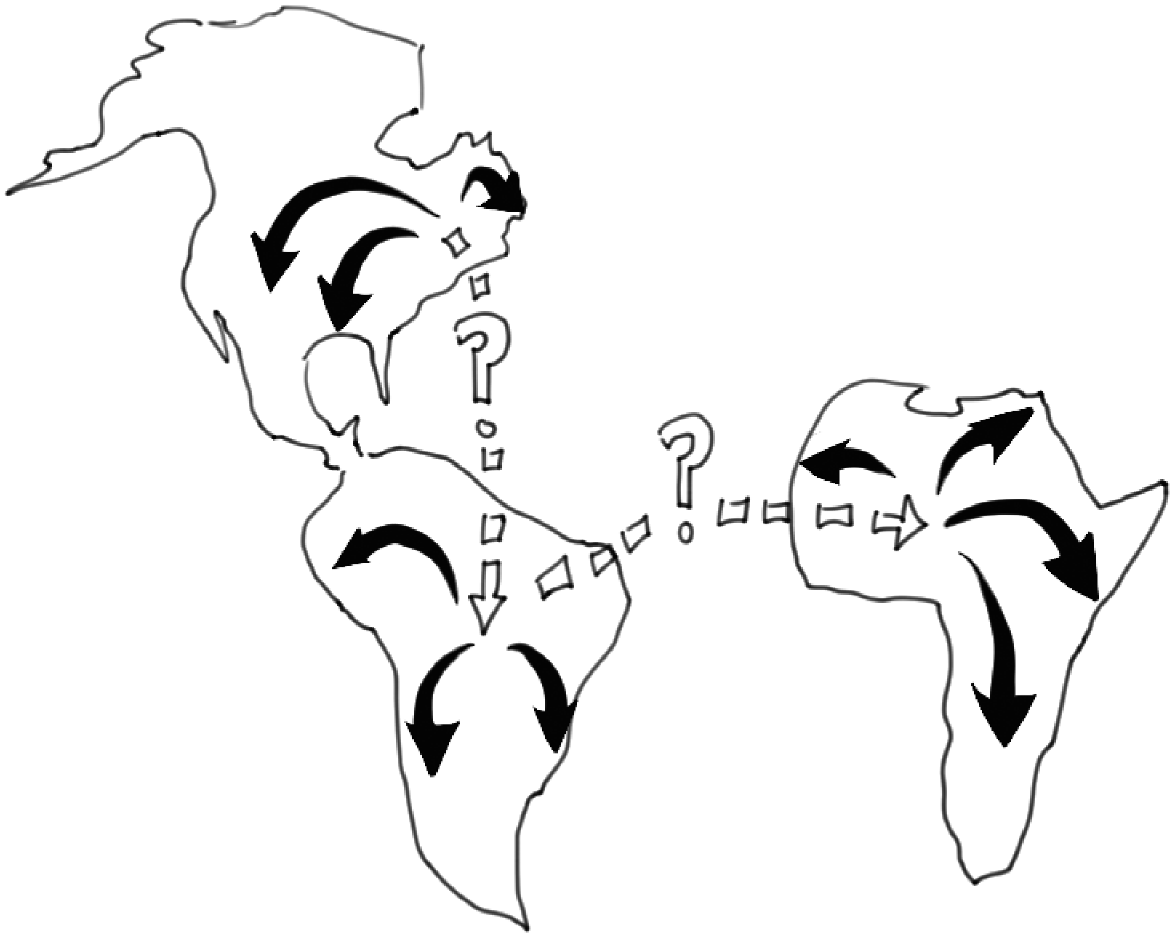
Super Automated Testing

Rx

Overview

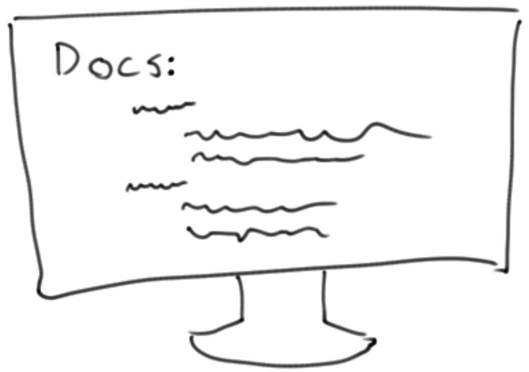
- ✓ Understand infection
  - new to system
  - bad code is referenced
- ✓ Fight infection
  - smarter shortcuts allow future growth
  - tutorials + docs
  - Refactoring
- ✓ Stay healthy
  - Automated tests
  - Fast unit tests
  - Other tests as needed

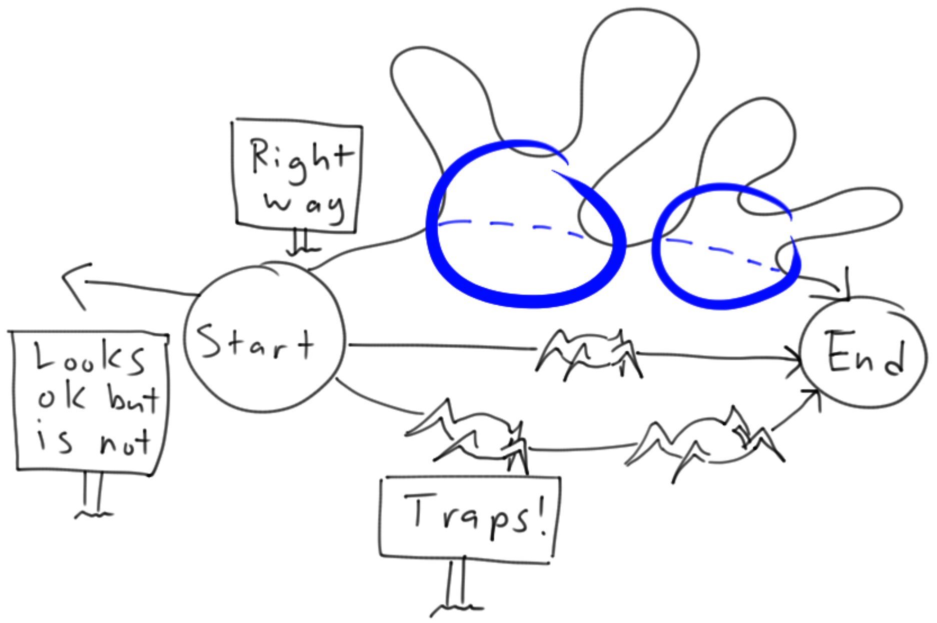
Bad code is an infection that spreads

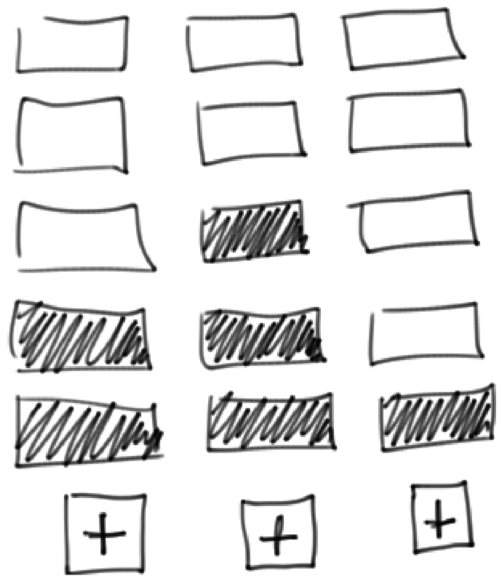


Fight the infection with regular maintenance.

Fight the infection with regular maintenance.







The principles of good code are also ones of testable code.

`Testable Code = Good Code`

Resources

The Art of Readable Code

Dustin Boswell and Trevor Foucher

Working Effectively with Legacy Code

Michael Feathers

Dive Into Python

Mark Pilgrim

(esp. Ch 7.3 on testing)

Thanks!

patc@arena.net

The Art of Readable Code

Dustin Boswell and Trevor Foucher

Working Effectively with Legacy Code

Michael Feathers

Dive Into Python

Mark Pilgrim

(esp. Ch 7.3 on testing)