Module 7

Specifications

Introduction to the Module

- This module is dedicated to specifications
  - The docstring at start of a function (DEMO)
  - Also the website documentation (DEMO)
- Useful for knowing how a function works
  - What if you didn't write the definition
  - Many functions you cannot see definition
- But why have an entire module on them?
  - Why not just say write good comments?

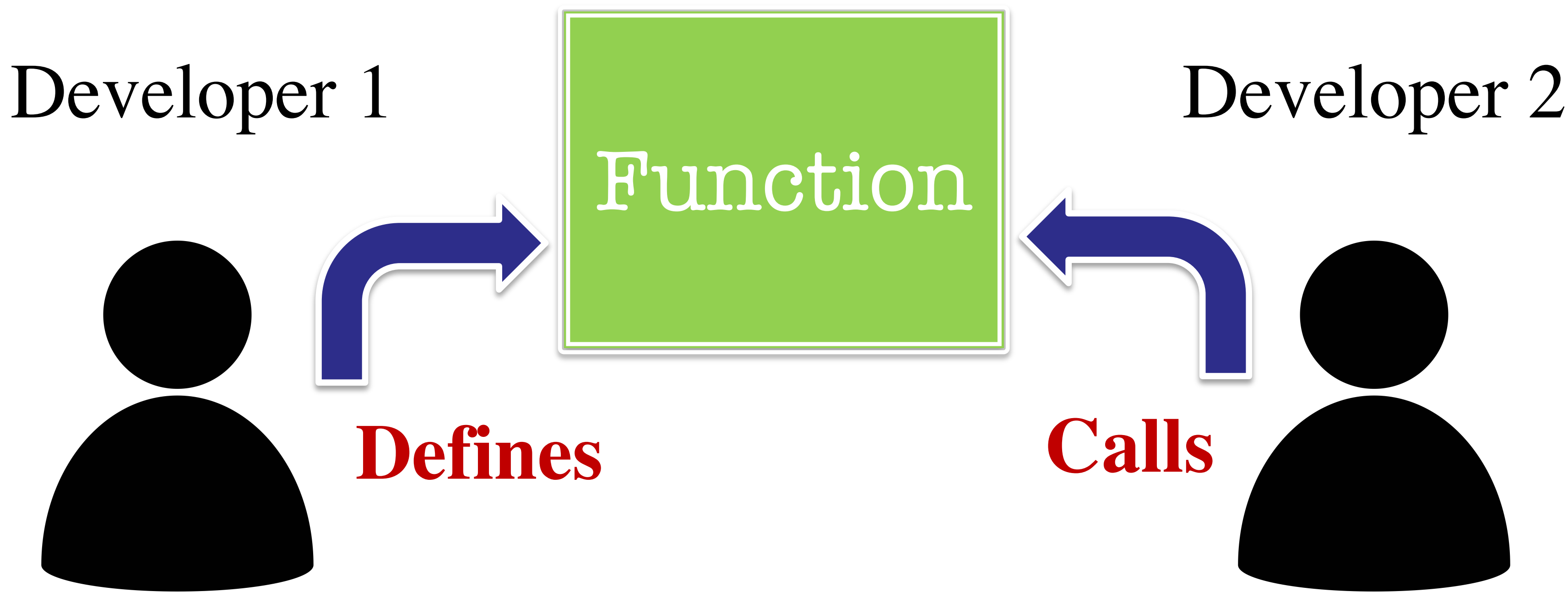
What Makes a Specification "Good"?

- Software development is a **business**
  - Not just about coding – business processes
  - Processes enable better code development
- Complex projects need **multi-person** teams
  - Lone programmers do simple contract work
  - Team must have people working separately
- Processes are about how to **break-up** the work
  - What pieces to give each team member?
  - How can we fit these pieces back together?

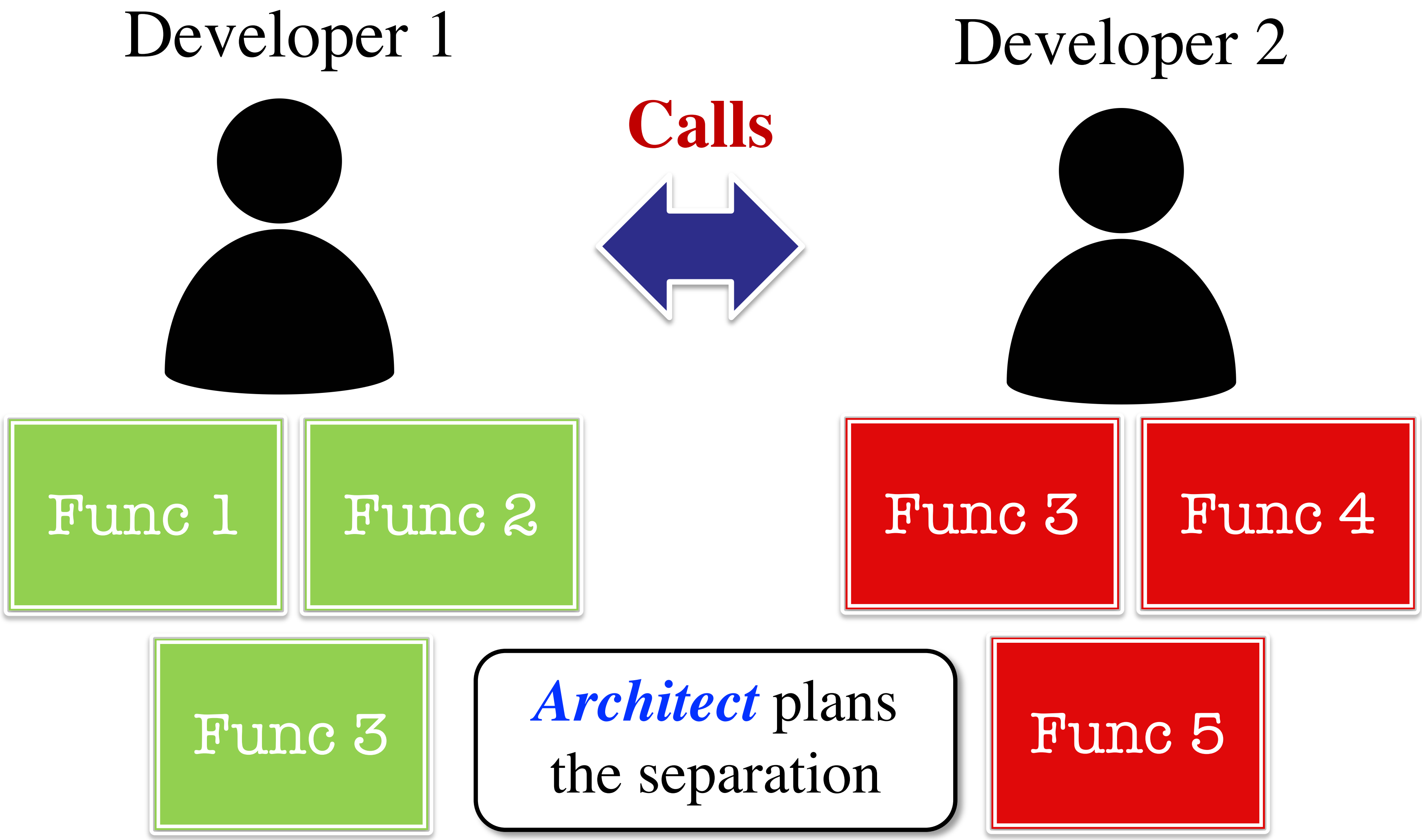
Focusing on the Basic Process

- May have heard of some of these processes
  - **Design**: Waterfall vs. Iterative vs. Agile
  - **Deployment**: DevOps
- These are beyond the scope of this course
  - Need a stronger programming background
- But there is a basic principal underlying all
  - Enabling communication and integration
  - They leverage functions to split up work

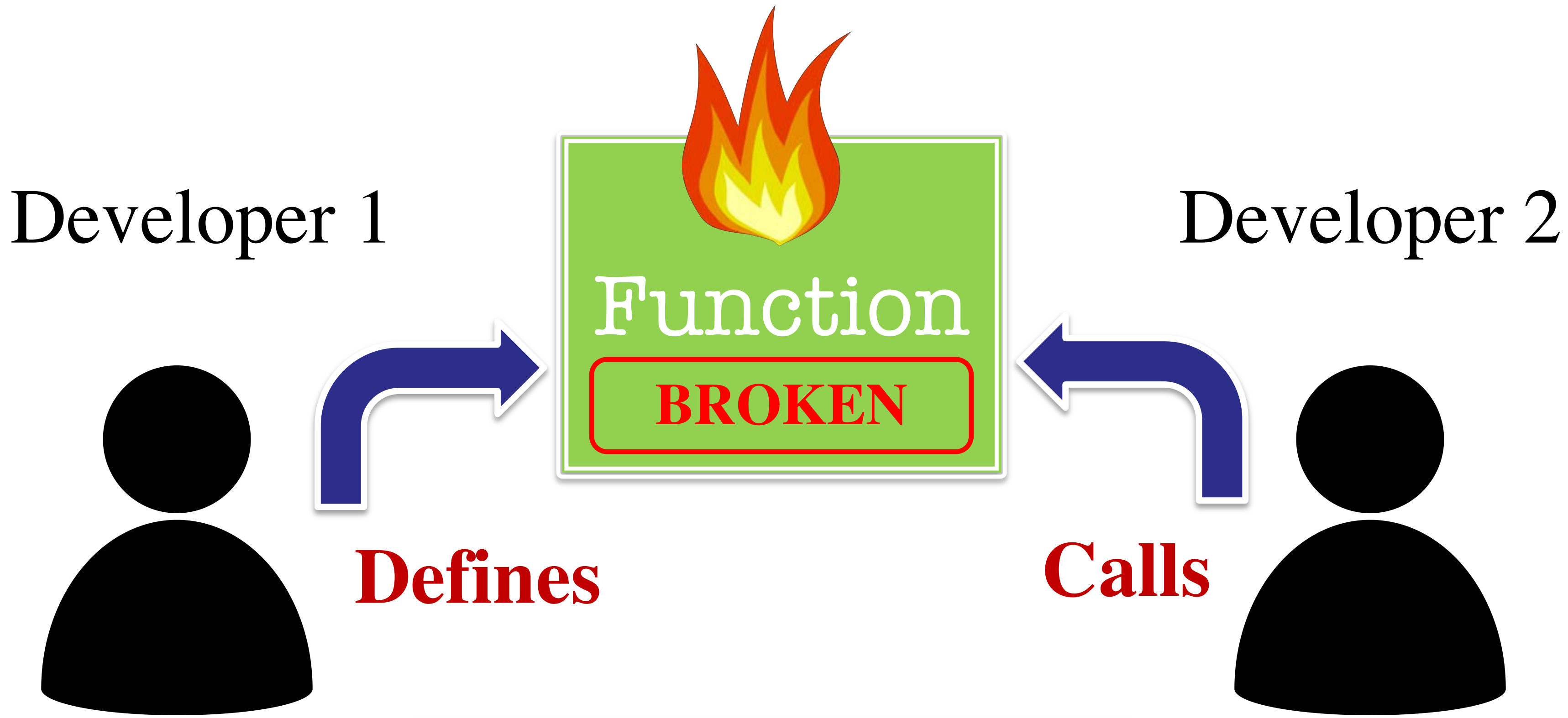
Functions as a Way to Separate Work



Working on Complicated Software



What Happens When Code Breaks?



Whose fault is it?
Who must fix it?

Purpose of a Specification

- To clearly layout **responsibility**
  - What does the function promise to do?
  - What is the allowable use of the function?
- From this responsibility we determine
  - If definer implemented function properly
  - If caller uses the function in a way allowed
- A specification is a **business contract**
  - Requires a formal documentation style
  - Agile etc. are ways to safely *modify* contract

So Why Do You Need to Know This?

- We have taught you how to write functions
  - You know all the technical details you need
- But not how to use code to solve problems
  - You are given a specification of a problem
  - You write code to a specification
- This means understanding specifications
  - What makes a good specification?
  - How do we cope with bad specifications?

Anatomy of a Specification

```
def greet(n):
    """Prints a greeting to the name n

    Greeting has format 'Hello <n>!'
    Followed by conversation starter.

    Parameter n: person to greet
    Precondition: n is a string"""
    print('Hello '+n+'!')
    print('How are you?')
```

One line description, followed by blank line

More detail about the function. It may be many paragraphs.

Parameter description

For a later video

Idea Behind Python Specifications

- One line summary for the TL;DR
  - To quickly determine if function appropriate
  - Often omits small, but important details
- Details are the fine print
  - What exactly does this function do
  - Helps me determine if I am on the fence
- Parameters help me arrange arguments
  - Line them up with comments in details text

Anatomy of a Specification

```
def to_centigrade(x):
    """Returns: x converted to centigrade

    Value returned has type float.

    Parameter x: temp in fahrenheit
    Precondition: x is a float"""
    return 5*(x-32)/9.0
```

"Returns" indicates a fruitful function

More detail about the function. It may be many paragraphs.

Parameter description

Goal: Separate procedures/fruitfuls

Python Docstring Conventions

- Python has guidance for docstrings
  - PEP 257 (linked on Canvas page)
  - Python Enhancement Proposals
- But gives too much flexibility for a beginner
  - Writing specifications is harder than coding
  - Learning to specify a large part of a CS degree
- Our version is more structured
  - Will hopefully cut down on mistakes (for now)
  - But adheres to the basic PEP guidelines

Anatomy of a Specification

```
def to_centigrade(x):
    """Returns: x converted to centigrade

    Value returned has type float.

    Parameter x: temp in fahrenheit
    Precondition: x is a float"""
    return 5*(x-32)/9.0
```

One line description, followed by blank line

More detail about the function. It may be many paragraphs.

Parameter description

Precondition specifies assumptions we make about the arguments

Preconditions are a Promise

- If precondition true
  - Function must work
- If precondition false
  - Function might work
  - Function might not
- Assigns responsibility
  - How tell fault

```
>>> to_centigrade(32.0)
0.0
>>> to_centigrade('32')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "temperature.py", line 19 ...
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

Precondition violated

What if it Just Works?

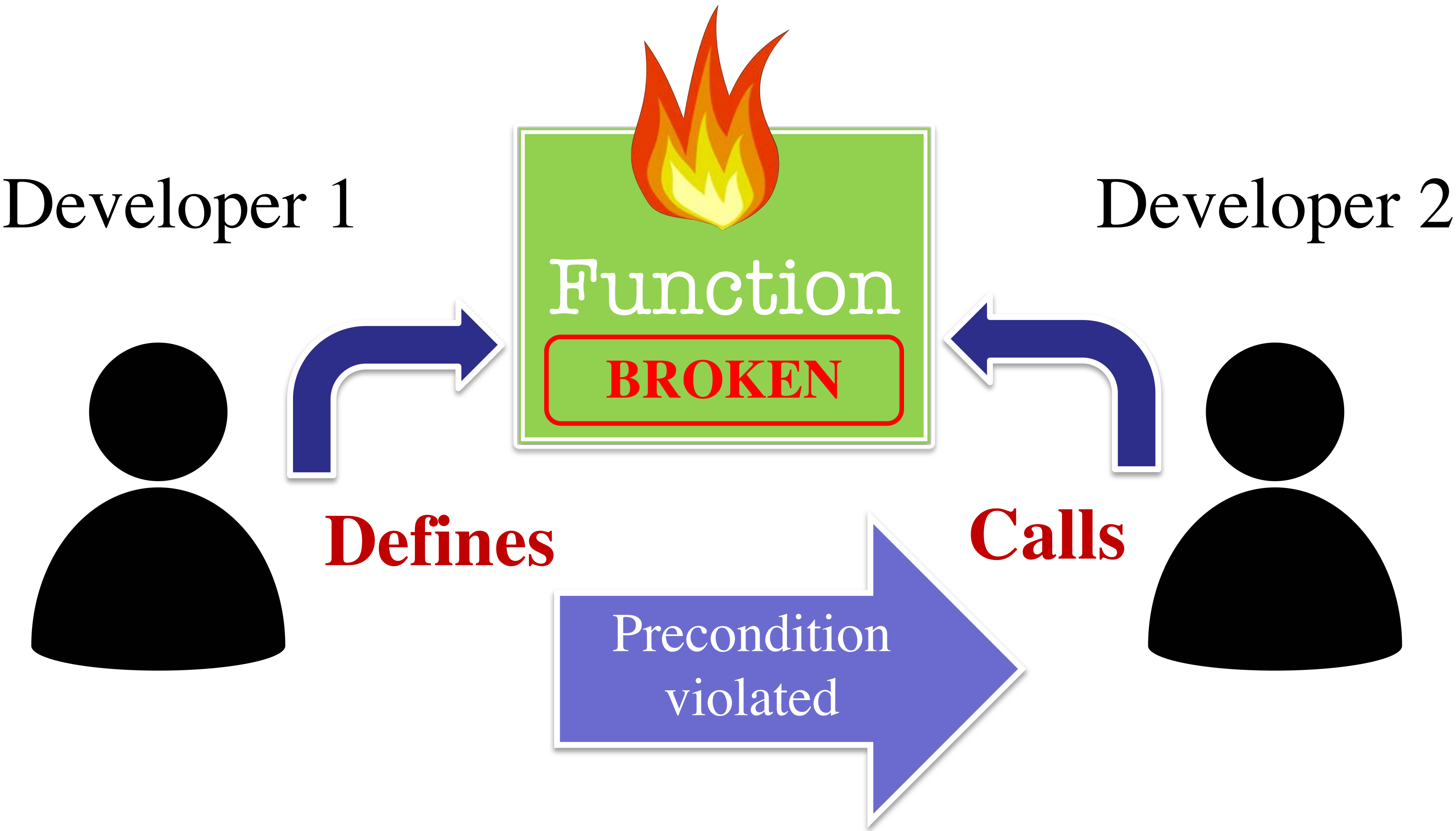
- Violation != crash
  - Sometimes works anyway
  - *Undocumented* behavior
- But is **bad practice**
  - Definer may change the definition at any time
  - Can do anything so long as specification met
  - Caller code breaks
- Hits MS developers a lot

```
>>> to_centigrade(32.0)
0.0
>>> to_centigrade(212)
100.0
```

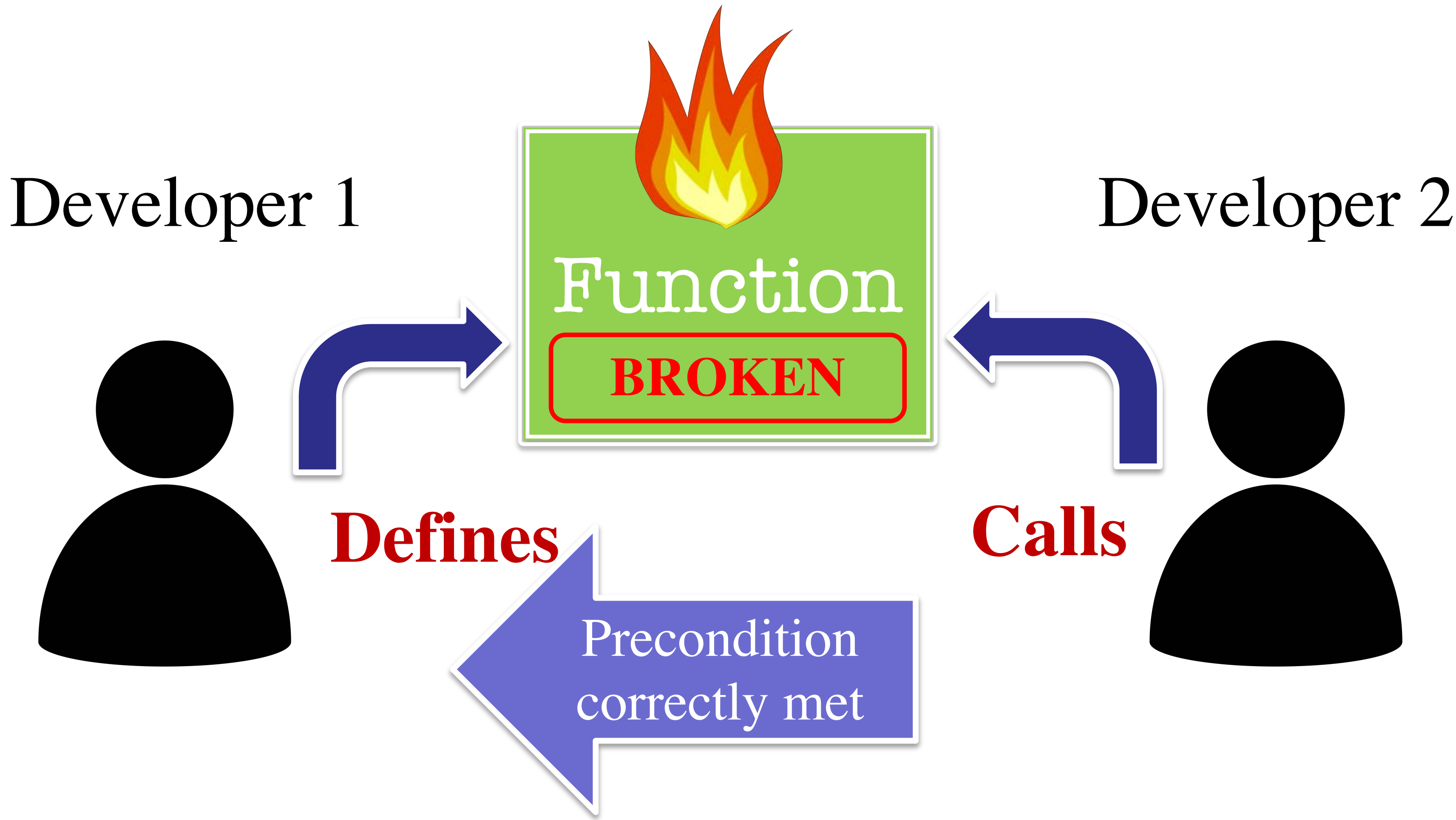
Precondition violated

Precondition violations are unspecified!

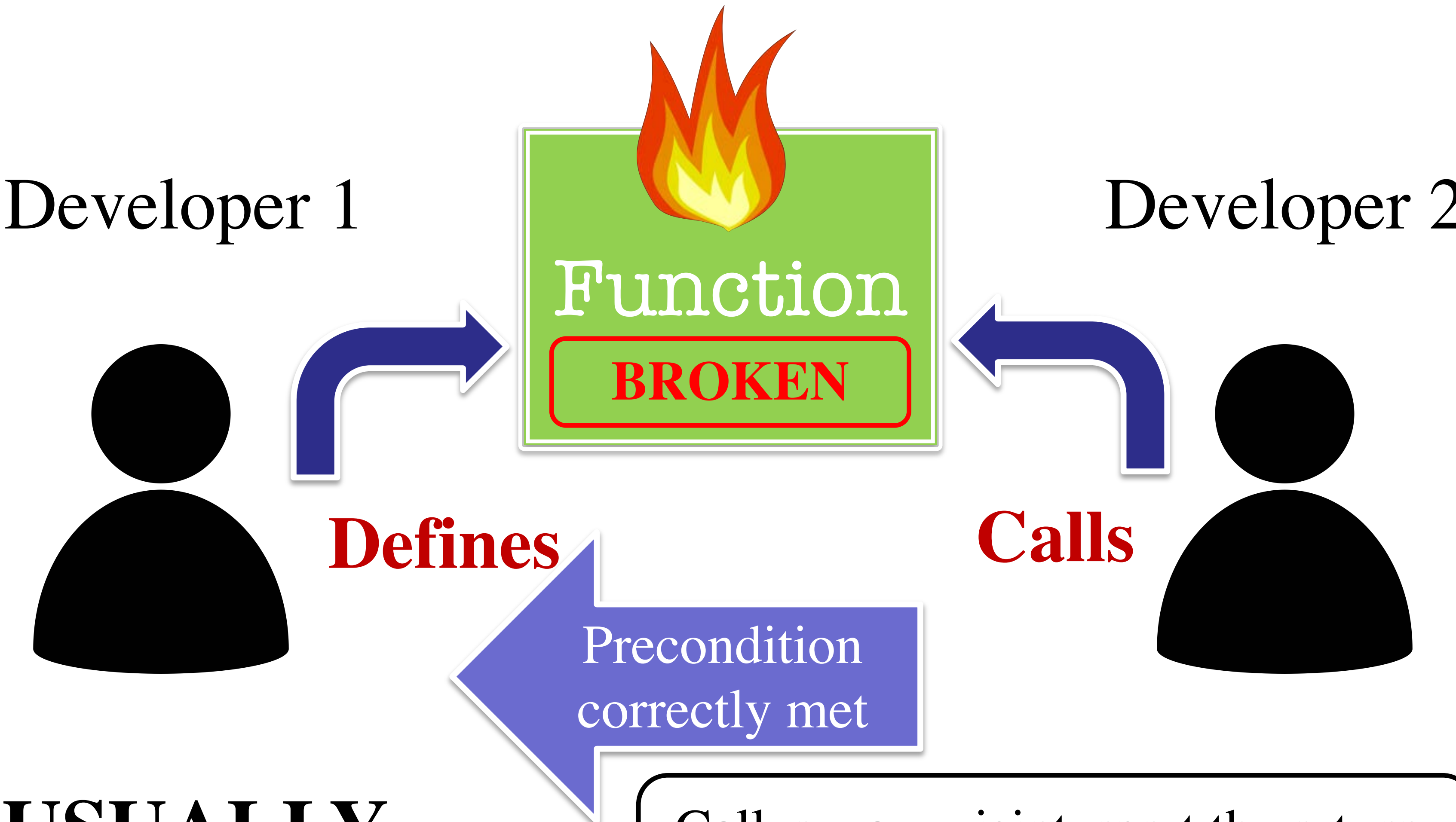
Assigning Responsibility



Assigning Responsibility



Assigning Responsibility



USUALLY

Caller may misinterpret the return.
But not a prob with (this) function.

Two Kinds of Preconditions

Type Restrictions

- **Ex**: `x` is an `int`
- Most common kind
  - Guarantees a set of ops
  - Some language support
- Very easy to check
  - `good = type(x) == int`

General Preconditions

- **Ex**: `fname` is a valid file
- Less common kind
  - Because of function
  - Precondition of called functions is precond
- Not so easy to check

Aside: Singling Out Type Restrictions

- C/Java/etc restrict types in the language
  - Variables themselves have types
  - So int value can only go in an int box
- Call these *statically typed* languages
  - Acts as a way to enforce preconditions
  - Ideal for large and complex software
- But Python (& Javascript) *dynamically typed*
  - Will allow anything, but will crash if misused
  - Must rely entirely on the specification

Is Statically Typed Better?

- Some prefer static typing for beginners
  - Will quickly shut down any violations
  - Many errors become a lot easier to find
- But can create false sense of security
  - Not all preconditions expressable as a type
  - So need to read specification anyway
- Why we focus so much on specifications

Learning to Read Specifications

- Most of the time, you only have specification
  - The definition may be hidden
  - Our it may be too complicated to read
- But not all specifications are good
  - May be incomplete and miss details
  - Particularly common in open source
- Need to evaluate specifications by reading
  - This is an imprecise skill
  - Built up with experience

Things to Look For

- Are the preconditions clear?
  - Can you see what is allowed and what is not?
  - Pay close attention if the precond not typed
  - Try to think of weird cases that my work
- **Fruitful**: Is the return result clear?
  - Look at every case that you thought of above
  - Can you immediately tell the result
- **Procedure**: Is the outcome clear
  - Same as above otherwise

A Simple Case Study

```
def number_vowels(w):
    """
    Returns: number of vowels in string w.

    Parameter w: The text to check for vowels
    Precondition: w string w/ at least one letter and only letters
    """
    …
```

This looks clear, right?

Case Study: Creating Examples

```
def number_vowels(w):
    """
    Returns: number of vowels in string w.

    Parameter w: The text to check for vowels
    Precondition: w string w/ at least one letter and only letters
    """
    …
```

- Let's brainstorm some inputs.
  - w = 'hat'   Answer: 1
  - w = 'heat'   Answer: 2
  - w = 'sky'   Answer: ???

What to do about **y**?

Case Study: Creating Examples

```
def number_vowels(w):
    """
    Returns: number of vowels in string w.

    Parameter w: The text to check for vowels
    Precondition: w string w/ at least one letter and only letters
    """
    …
```

- Can we just figure out y from context?
  - Who said these were English words?
  - Welsh vowels include `w` as well (`'cwtsh'`)
  - In fact, **who said these are words** at all (`'grblx'`)?

Case Study: More Examples

```
def number_vowels(w):
    """
    Returns: number of vowels in string w.

    Parameter w: The text to check for vowels
    Precondition: w string w/ at least one letter and only letters
    """
    …
```

- What does number of vowels mean?
    - How does it handle repeated vowels ('beet')?
    - If no repeats, does upper or lower case matter?

A Better Specification

```
def number_vowels(w):
    """
    Returns: number of vowels in string w.

    Vowels are defined to be 'a','e','i','o', and 'u'. 'y' is a vowel if it is
    not at the start of the word.

    Repeated vowels are counted separately.  Both upper case and
    lower case vowels are counted.

    Examples: ....

    Parameter w: The text to check for vowels
    Precondition: w string w/ at least one letter and only letters
    """
```

More details about outcome

Examples covering the major cases

What Happened Here?

- Specification was very incomplete
  - Relied on confusion btwn string and text
  - We filled in details with assumptions
- This is very dangerous
  - Many of our assumptions are cultural
  - What if working with an international team?
  - Some internationals do not know vowels
- Must be very precise in our reading

What Should You Do?

- You **may not** change the specification
  - Not even if you are the function definer
  - Specification often came from higher up
  - Methodologies have rules for changing spec
- You should **ask for more guidance**
  - From the specification author (not definer)
  - In a course, this is the instructor
- We would rather you get it right first time