What we will see here:

1. Some design patterns (Wrapper, Interfaces, Factory);
2. Testing without and with mocking;
3. Debugging sessions;
4. Unit tests vs integration tests;

* In a bioinformatics context;
* Using python;
* Somewhat inefficient code but easy to understand

… but I just realised on sunday evening, that this is too much, will probably need to do this in 2 meetings;

… this is an experiment, maybe next time get rid of the bioinformatics context and go directly to the point?

Obs:

- That is a lot of things, but it is better to slow down and understand stuff than just rushing to present everything;

- Many of these things are fairly recent to me, feel free to correct me at any time;

- The choice of python is because it is the most used language in the group (I’d prefer to do in C++, but all of these things can also be done in C++);

Contamination Finder tool

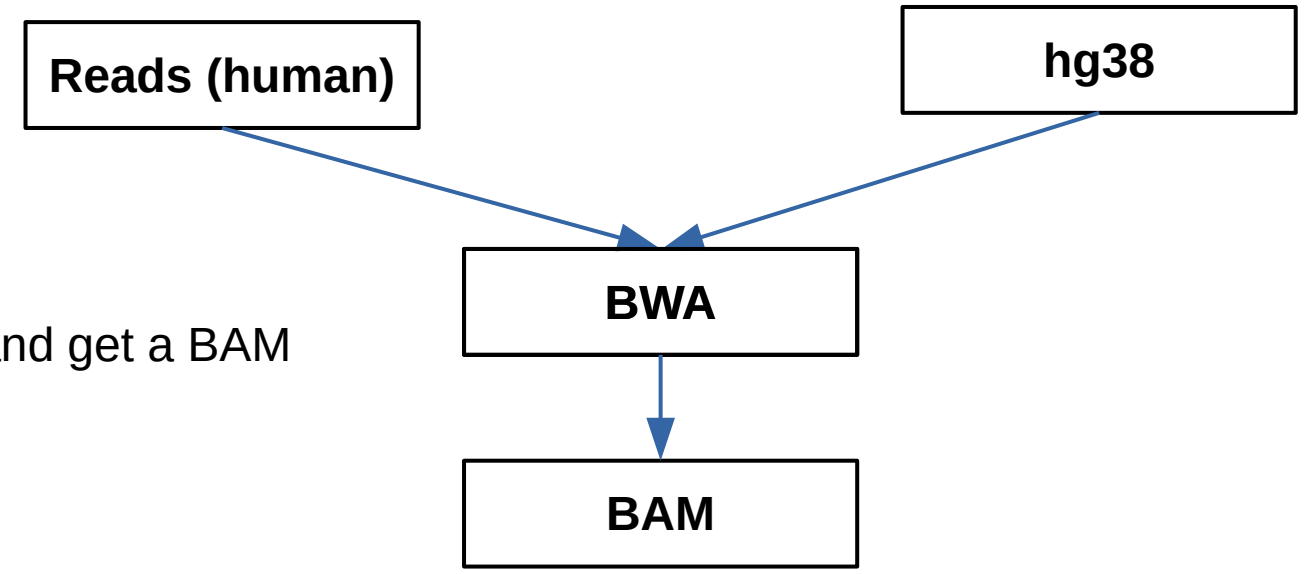
Pipeline:

1. Get reads **Reads (human)**

Contamination Finder tool

Pipeline:

1. Get reads



2. Align to ref and get a BAM

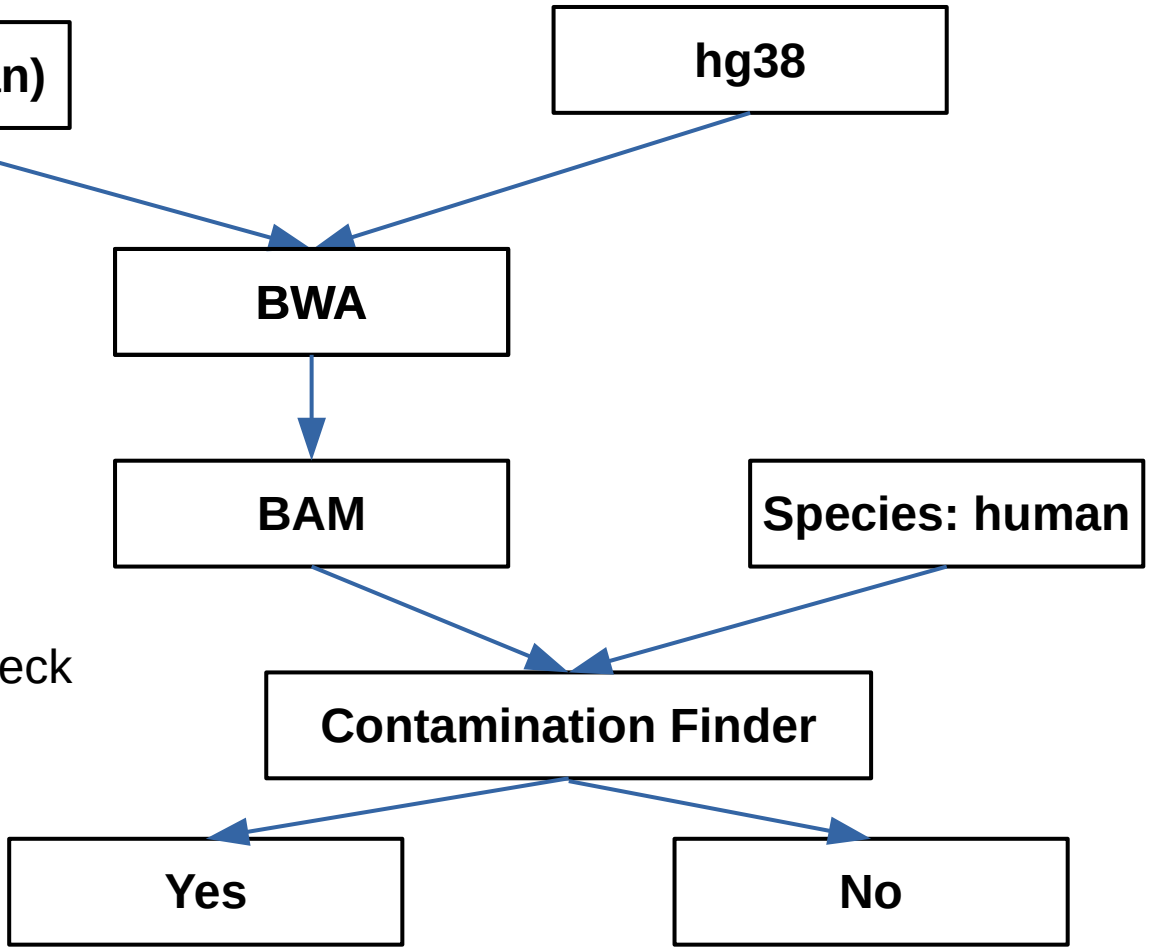
Contamination Finder tool

Pipeline:

1. Get reads

2. Align to ref and get a BAM

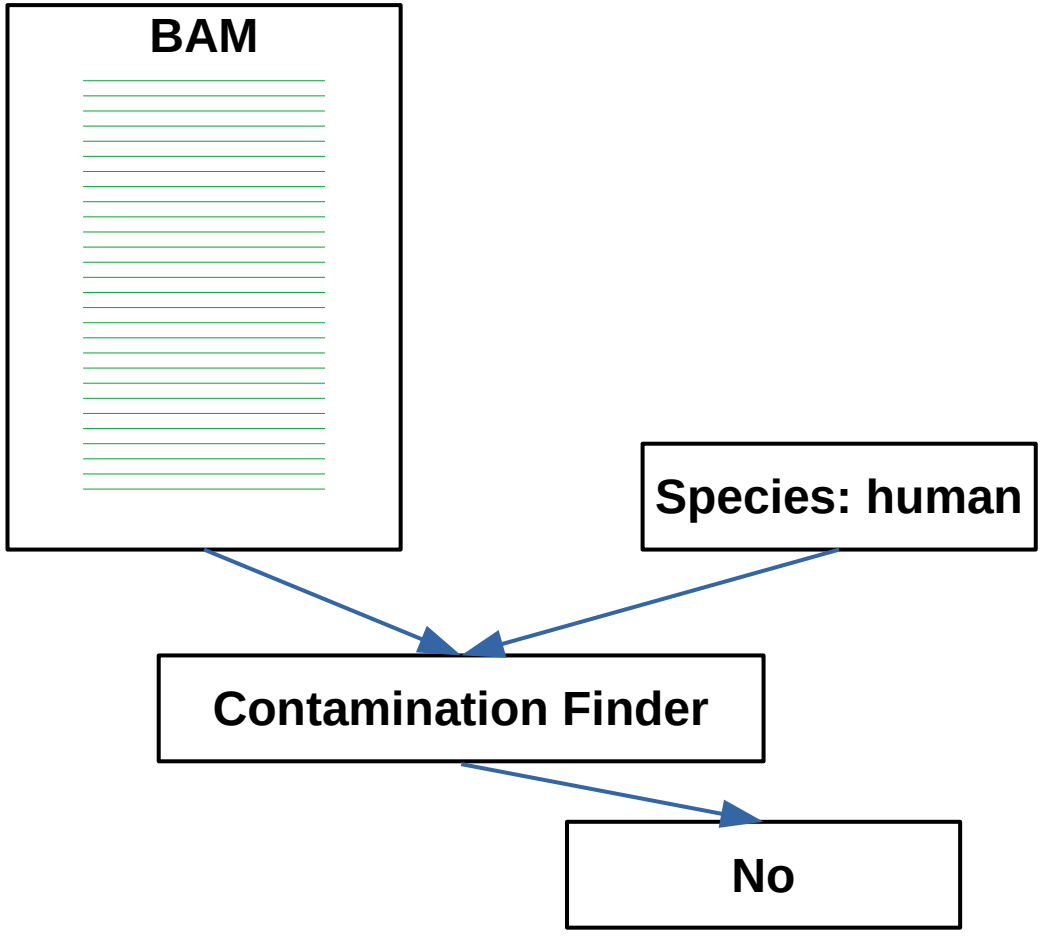
3. Give BAM and species to Contamination Finder tool to check if there is contamination.



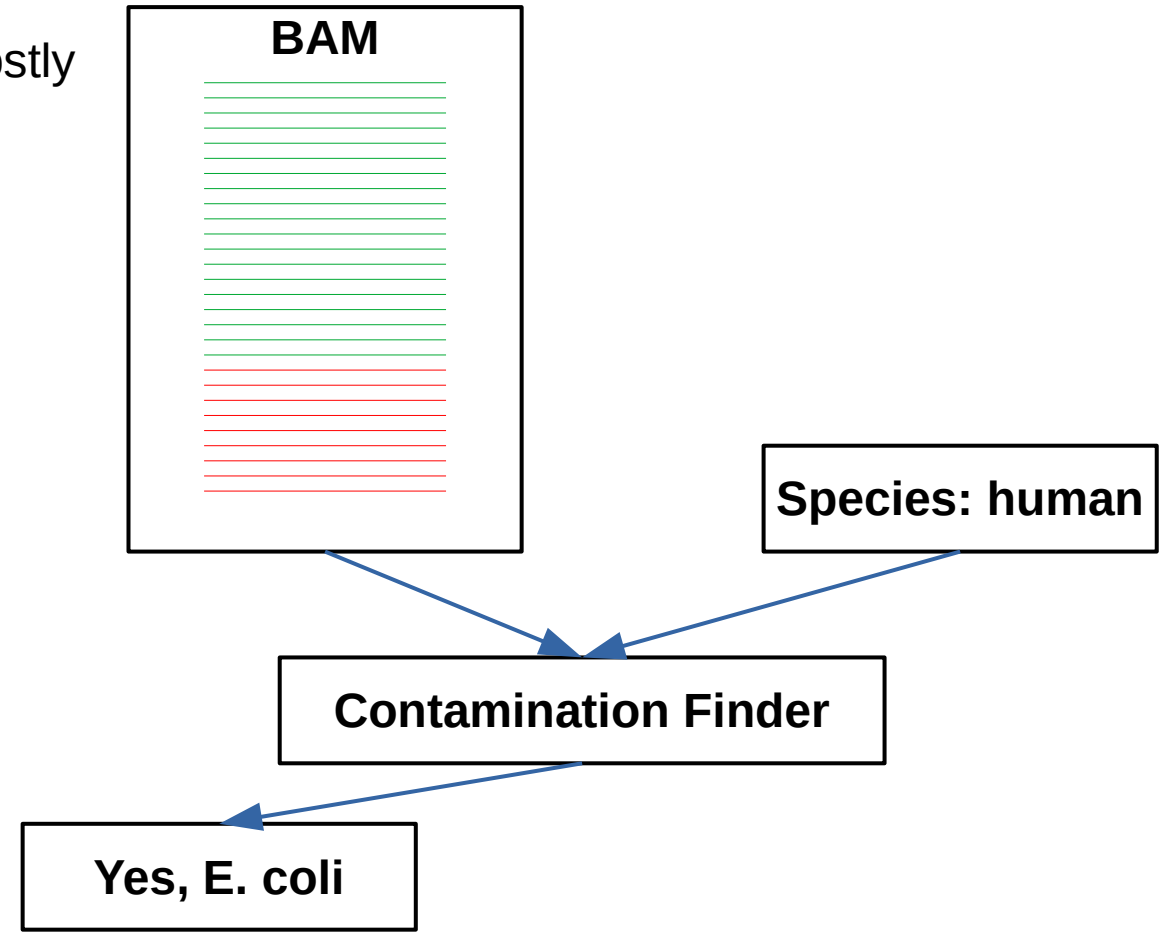
How to find contamination (not very realistic)

1. Get only the unmapped reads from the BAM;
2. Search them in a read index (e.g. http://www.bigsi.io/);
3. Get the most probable species for each dataset, and count the frequency of each species;
4. The most-frequent species is the putative contaminator;

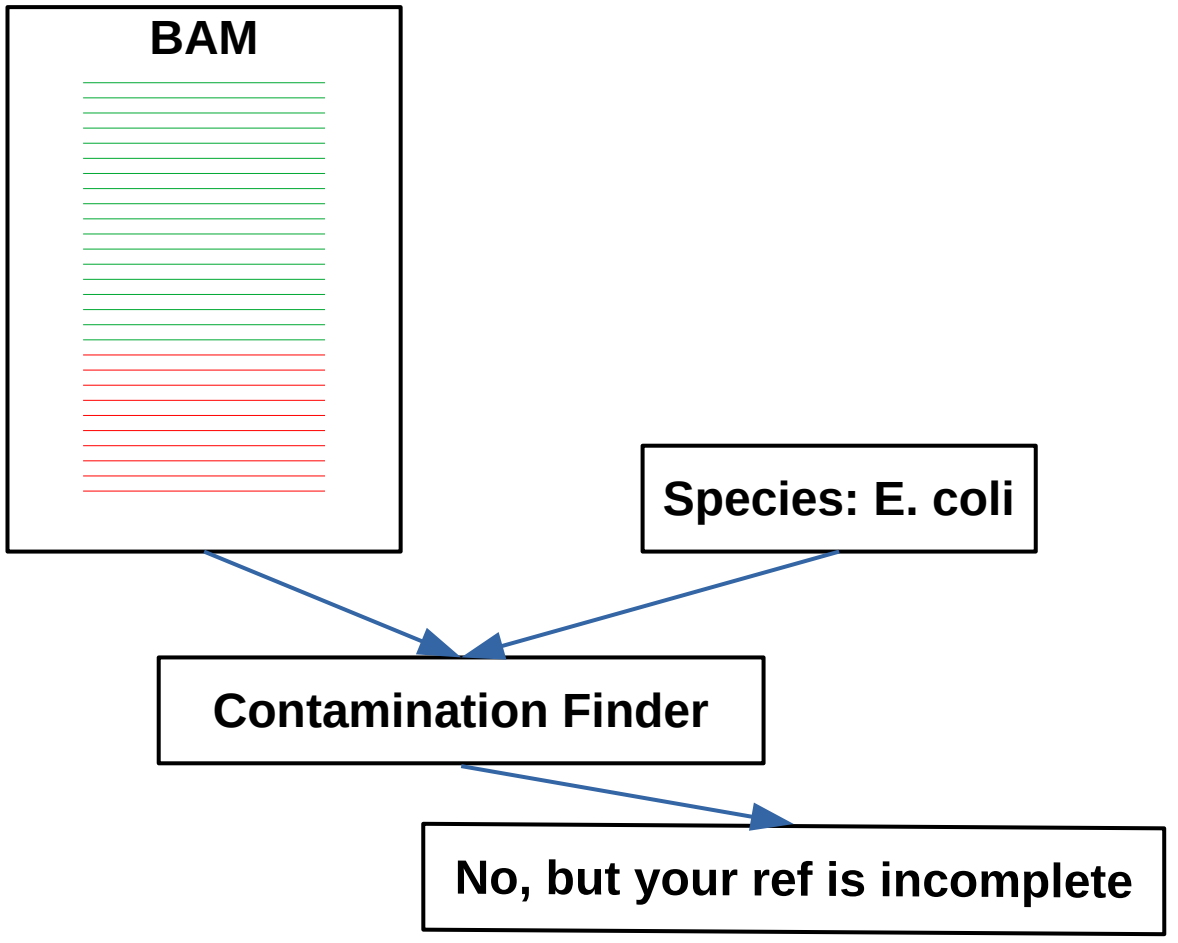
Use case #1:
All reads are mapped to the ref



Use case #2:
Unmapped reads
map to E. coli mostly



Use case #3:
Unmapped reads map the same species the ref is



IMPLEMENTATION

BAM module only

Wrapper/Facade design pattern

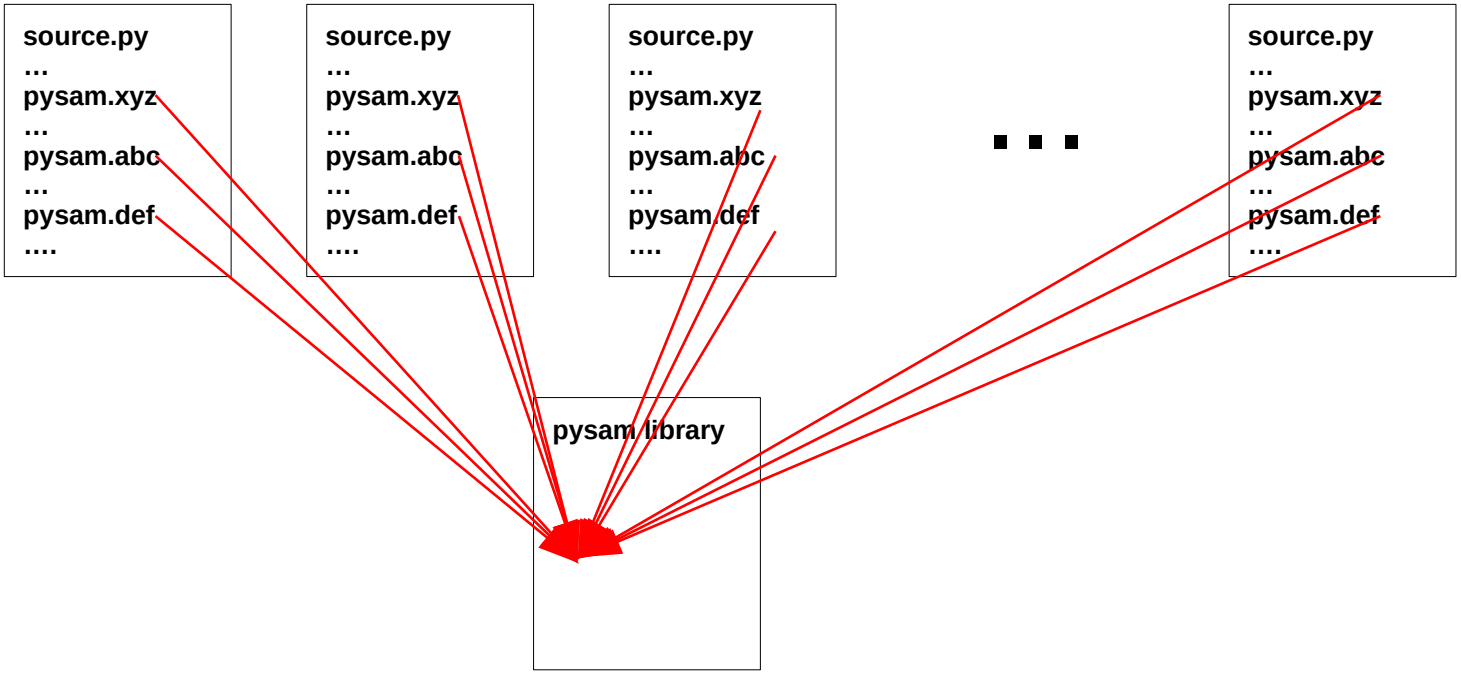
1. Provide a context-specific interface to more generic functionality:
pysam.AlignedSegment interface (generic functionality) is huge:
https://pysam.readthedocs.io/en/latest/api.html#pysam.AlignedSegment

Our context-specific interface:

```
class BamRecord:
    def __init__(self, record: AlignedSegment)

    def is_unmapped(self) -> bool

    def get_sequence(self) -> str

    def __eq__(self, other: "BamRecord")
```

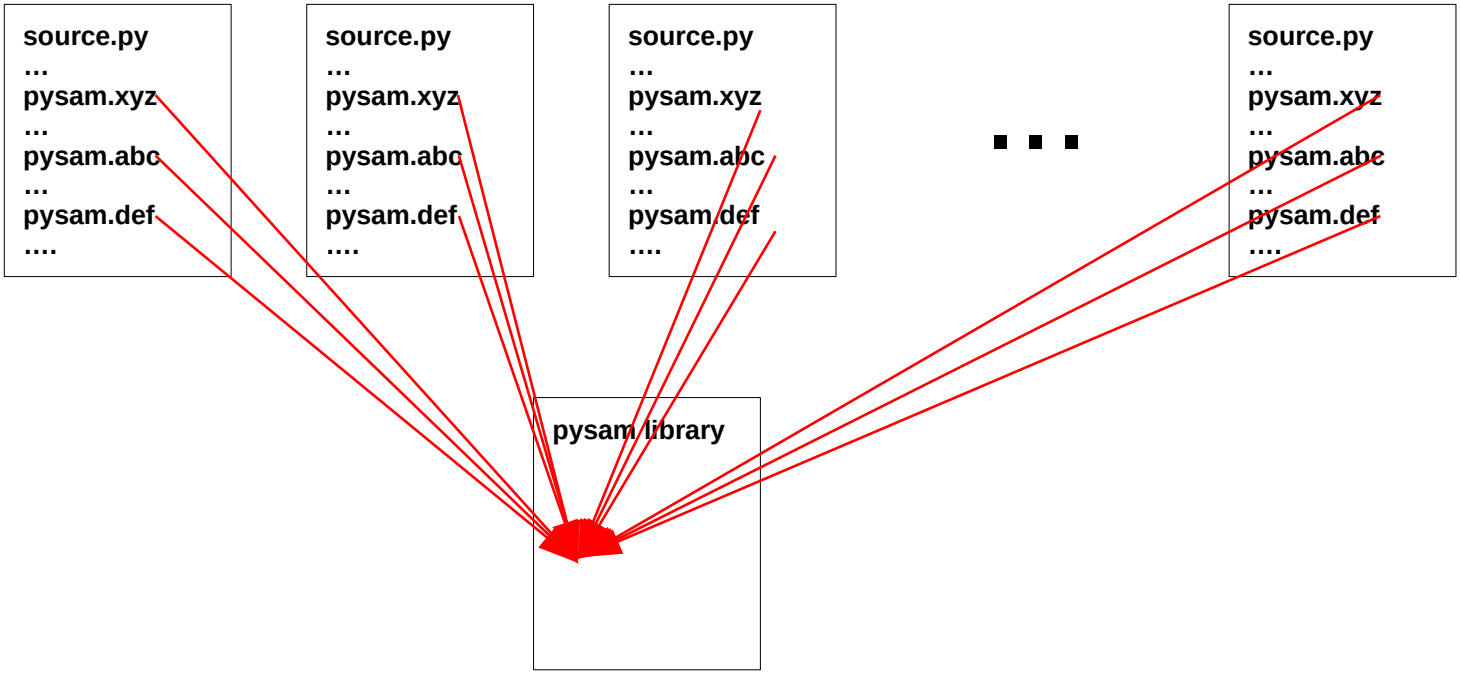
Wrapper/Facade design pattern

2. Our code is now less tightly-coupled to an external dependency, making it more maintainable:



Wrapper/Facade design pattern

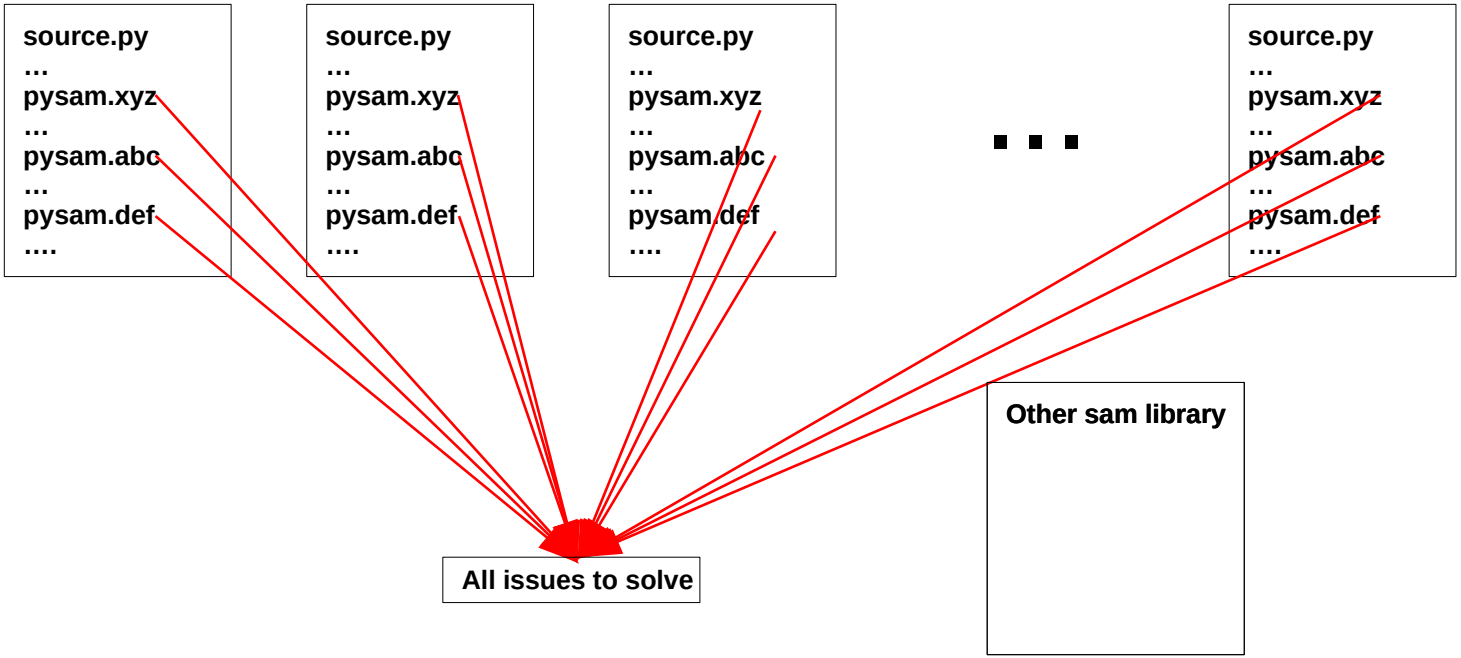
2. Our code is now less tightly-coupled to an external dependency, making it more maintainable:



Let’s change the SAM parsing library, because pysam has a bug or is too slow, etc...

Wrapper/Facade design pattern

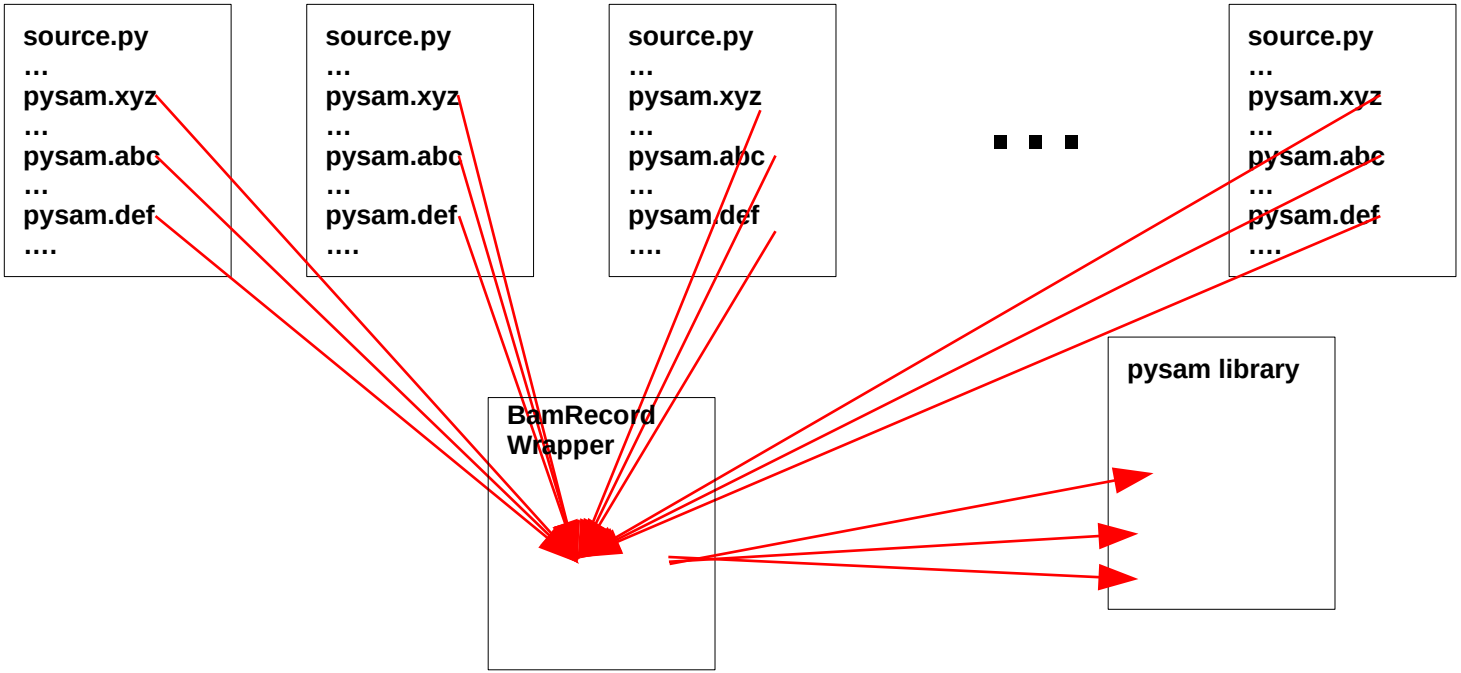
2. Our code is now less tightly-coupled to an external dependency, making it more maintainable:



Let’s change the SAM parsing library, because pysam has a bug or is too slow, etc...

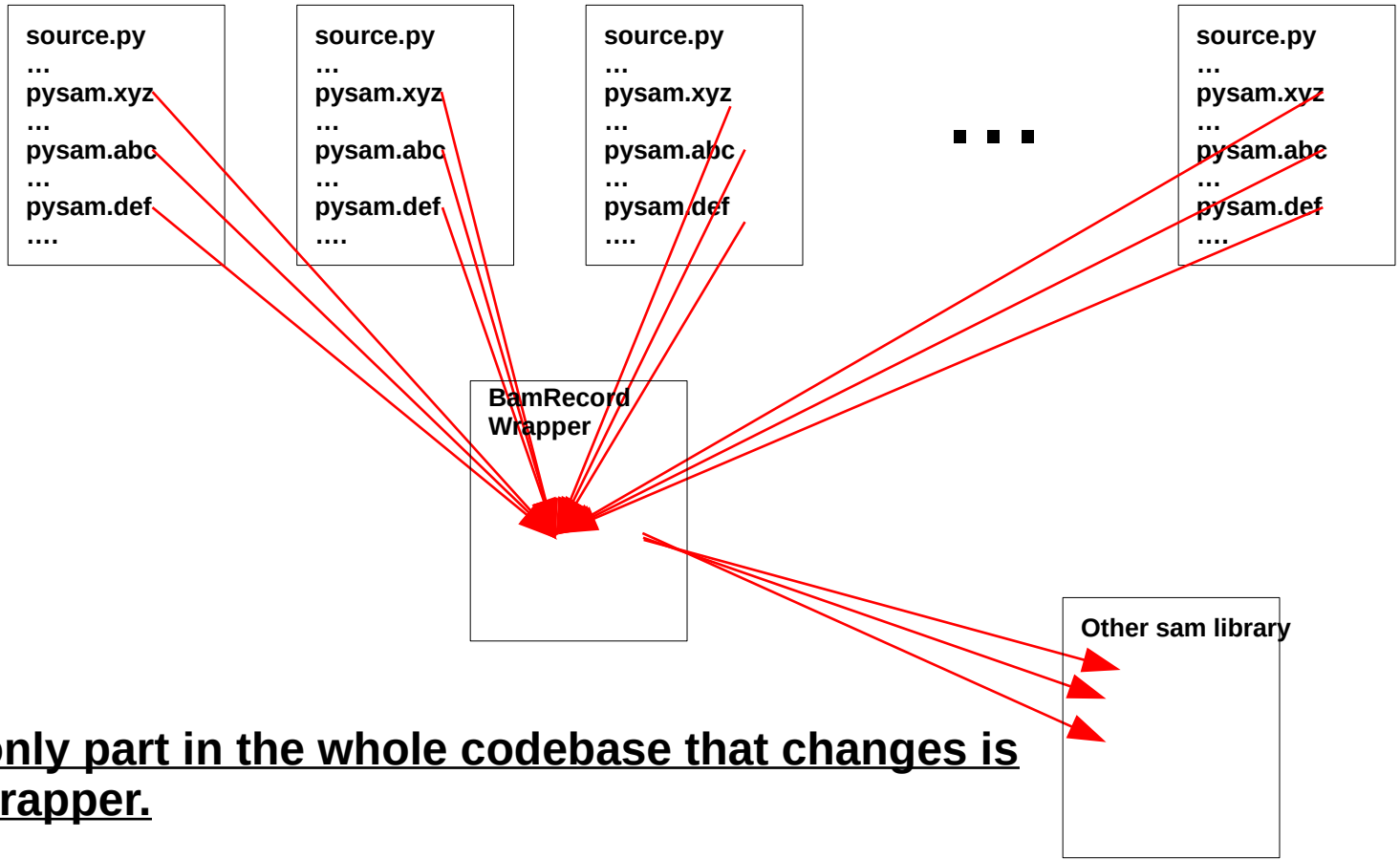
Wrapper/Facade design pattern

2. Our code is now less tightly-coupled to an external dependency, making it more maintainable:



Wrapper/Facade design pattern

2. Our code is now less tightly-coupled to an external dependency, making it more maintainable:



<u>The only part in the whole codebase that changes is the wrapper.</u>

TESTING OF BAM CLASS

Unit vs integration tests

Function1

```
Instruction_1;
Instruction_2;
Instruction_3;
call_function_2();
Instruction_4;
Instruction_5;
call_function_3();
...
```

Unit vs integration tests

Unit test: should test this single unit:

Function1

```
Instruction_1;
Instruction_2;
Instruction_3;
call_function_2();
Instruction_4;
Instruction_5;
call_function_3();
...
```

Unit vs integration tests

Unit test: should test this single unit:

Function1

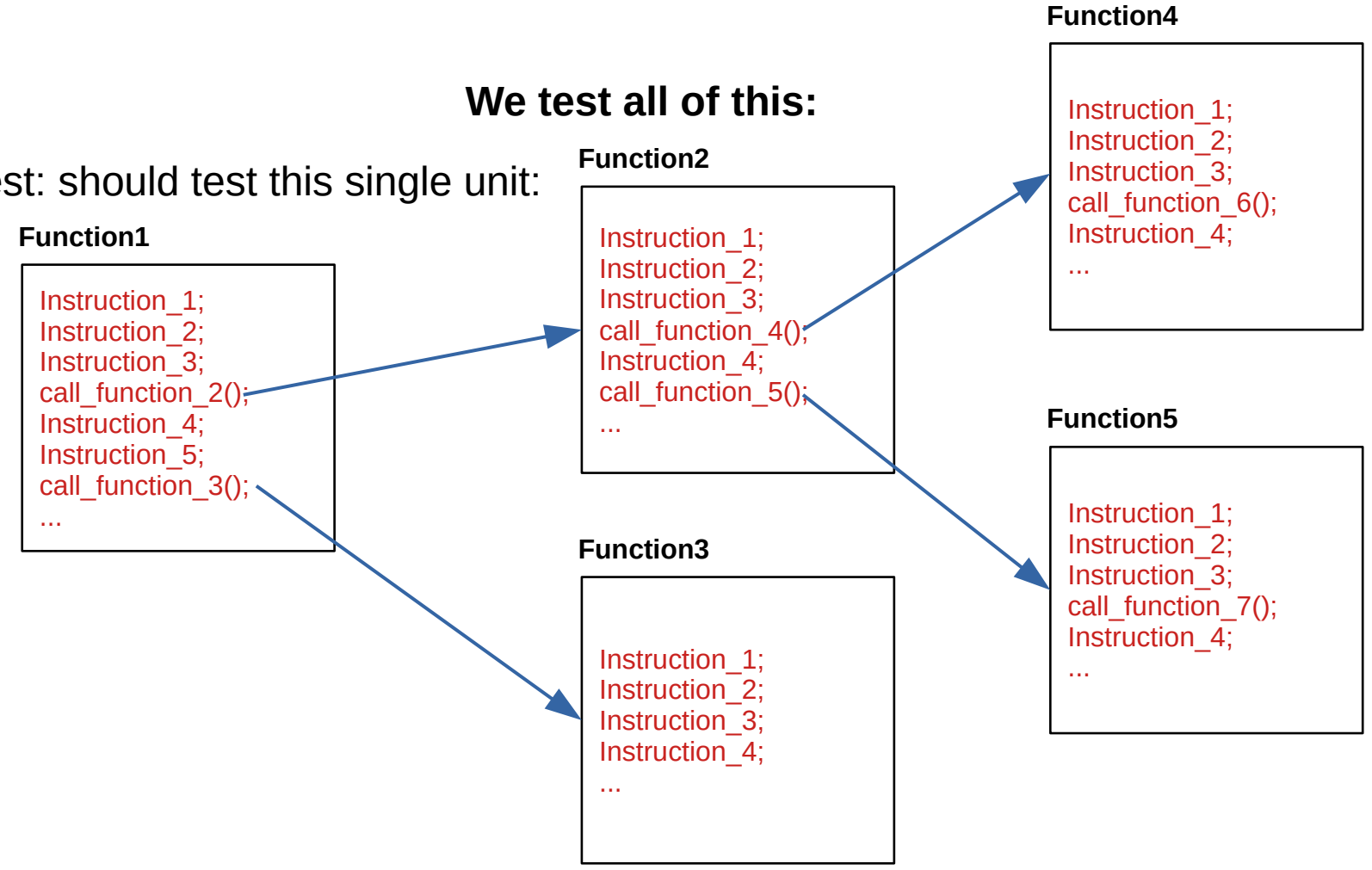
```
Instruction_1;
Instruction_2;
Instruction_3;
call_function_2();
Instruction_4;
Instruction_5;
call_function_3();
...
```

What happens if we use real objects, and not mocking (as we did)?

Unit vs integration tests

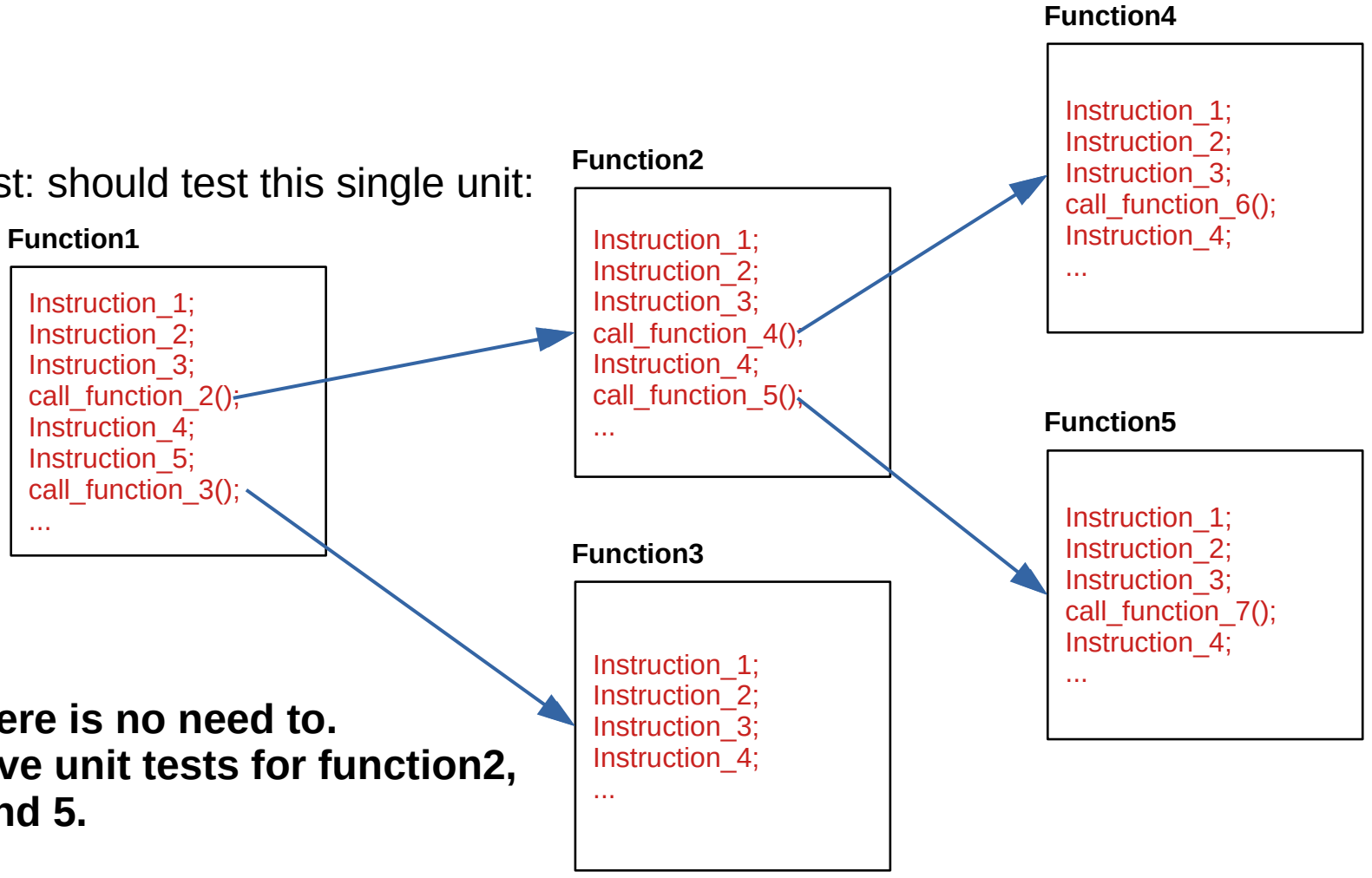
Unit test: should test this single unit:

We test all of this:



Unit vs integration tests

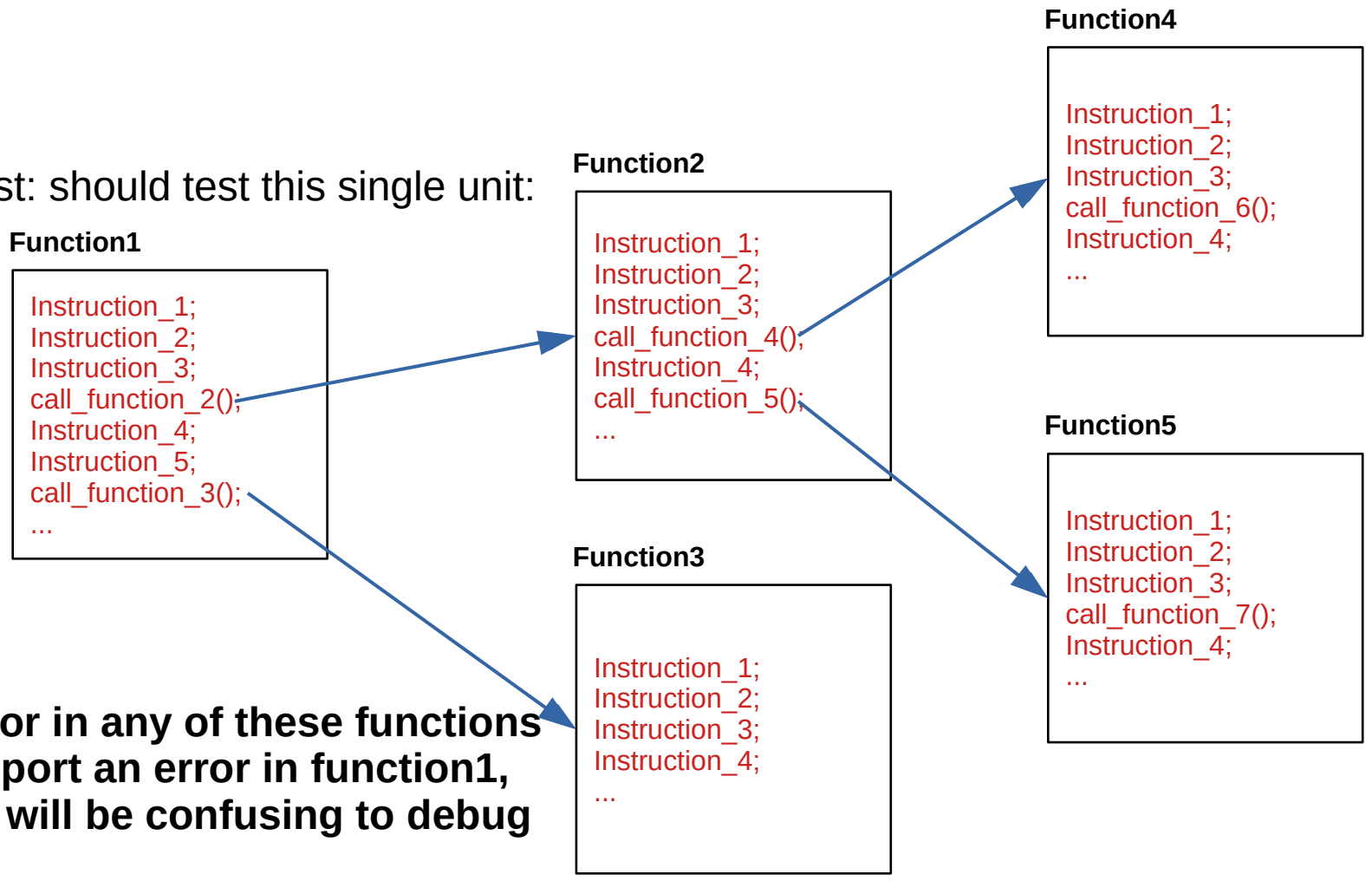
Unit test: should test this single unit:



But there is no need to.
We have unit tests for function2, 3, 4, and 5.

Unit vs integration tests

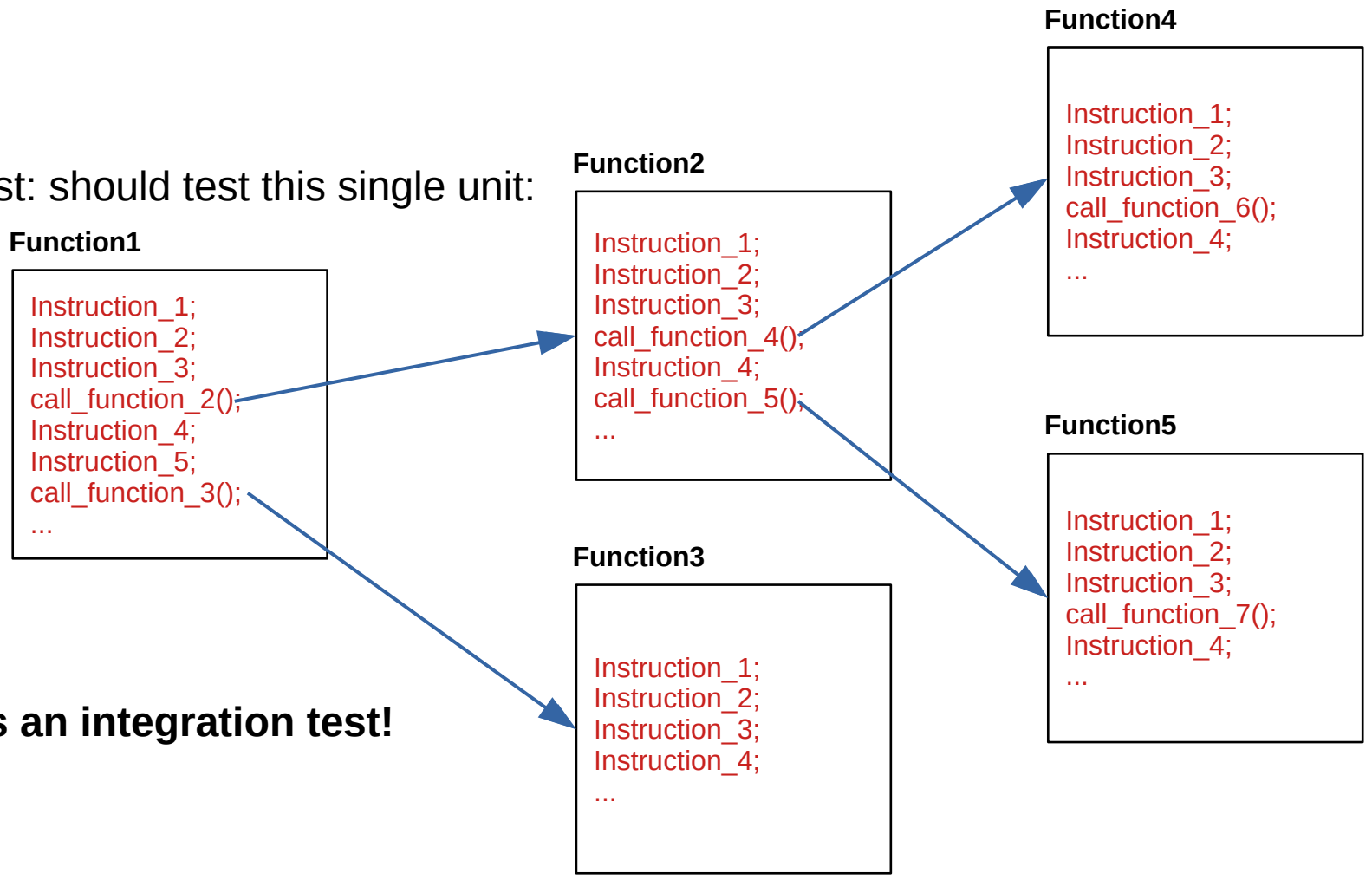
Unit test: should test this single unit:



An error in any of these functions
Will report an error in function1,
And it will be confusing to debug

Unit vs integration tests

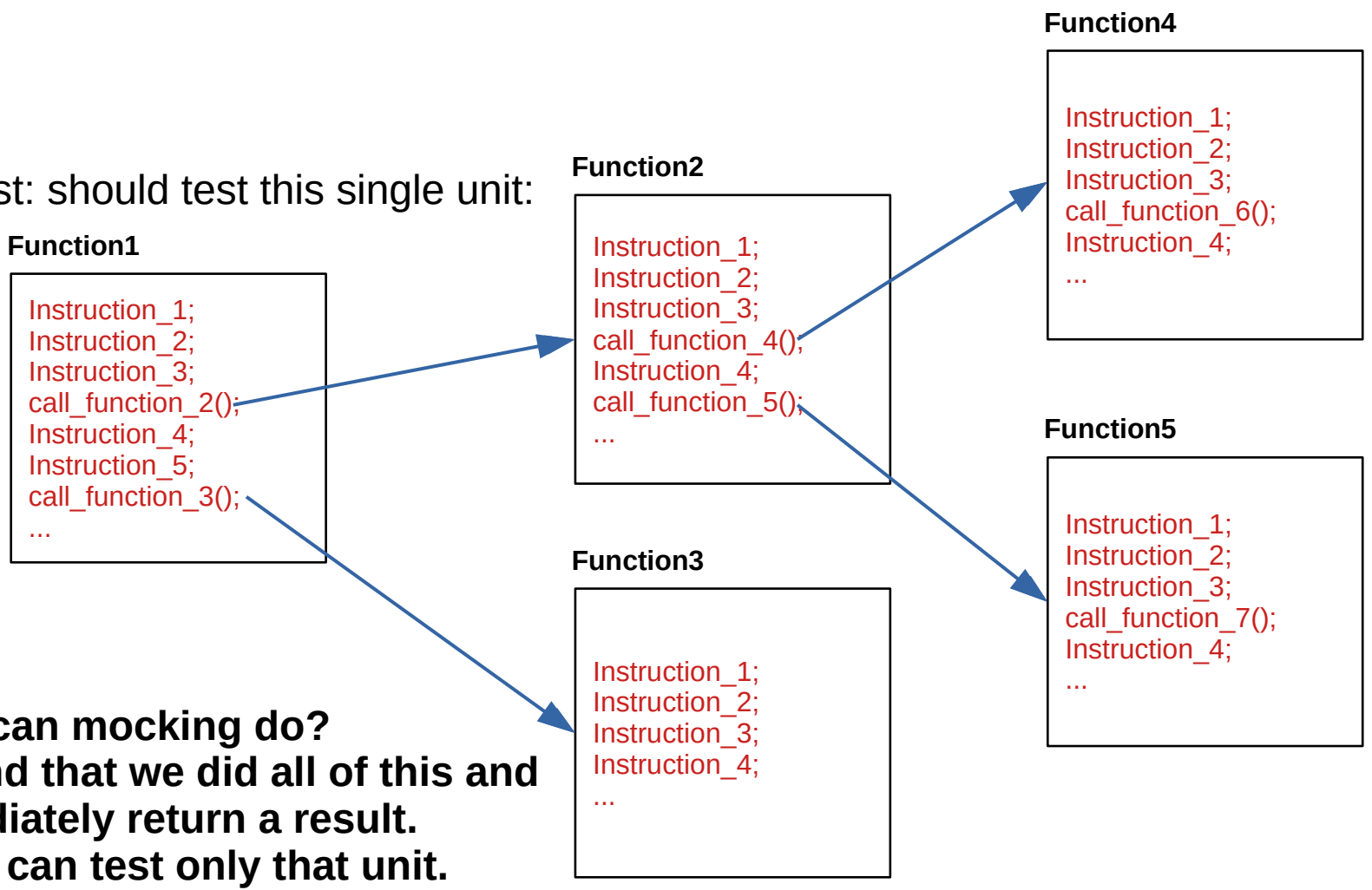
Unit test: should test this single unit:



This is an integration test!

Unit vs integration tests

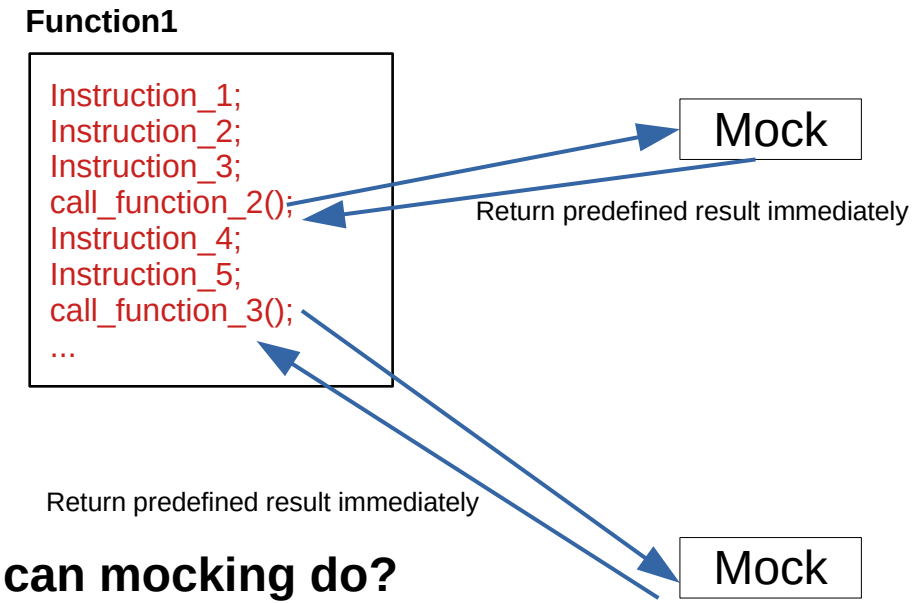
Unit test: should test this single unit:



What can mocking do?
Pretend that we did all of this and Immediately return a result.
So we can test only that unit.

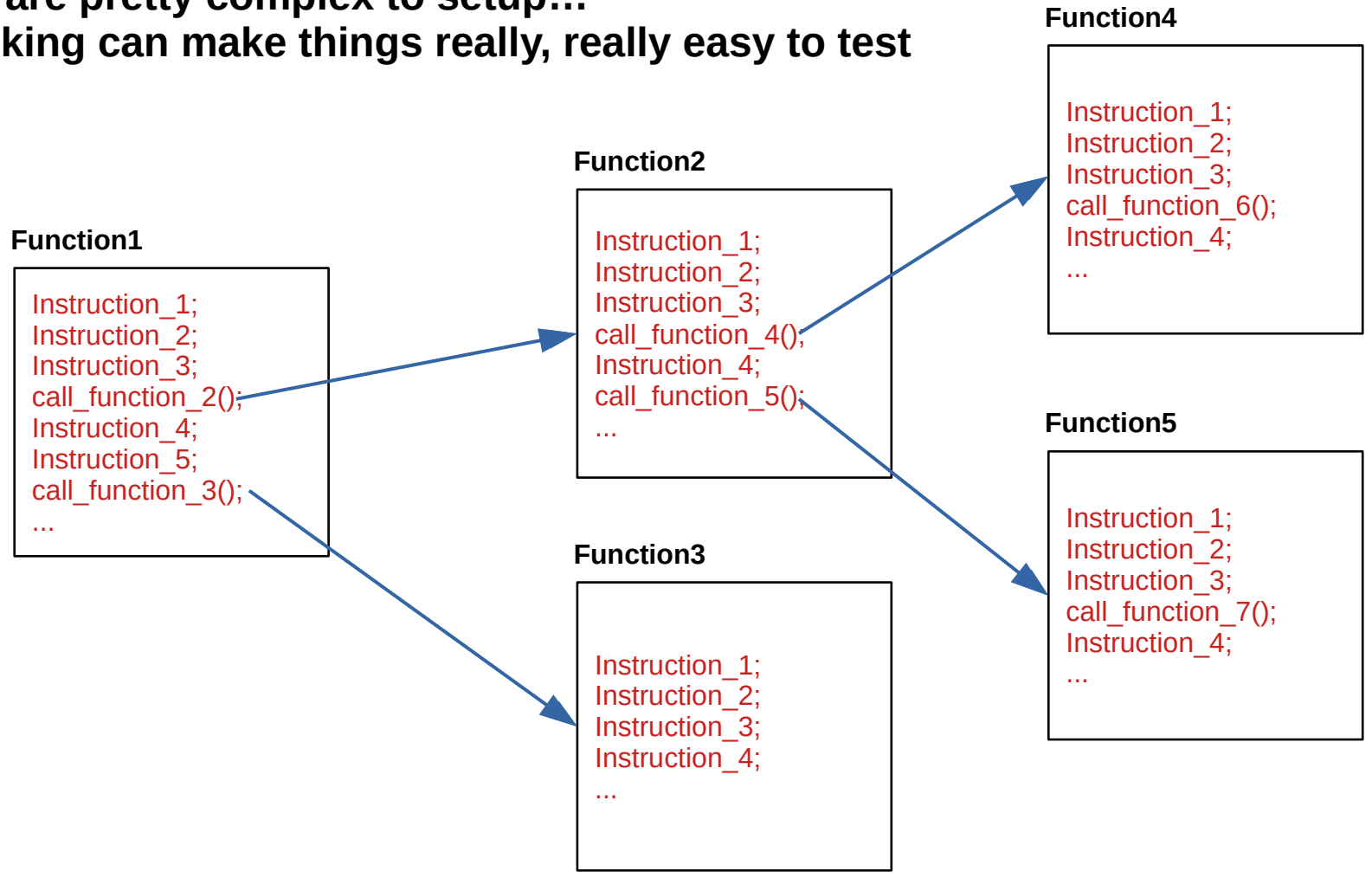
Unit vs integration tests

Unit test: should test this single unit:



What can mocking do?
Pretend that we did all of this and
Immediately return a result.
So we can test only that unit.

Instead of functions, these can be object dependencies that are pretty complex to setup…
Mocking can make things really, really easy to test



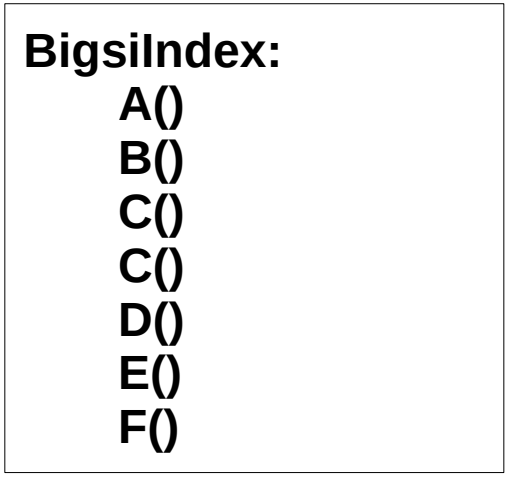
END OF FIRST PART…
BUT THE COOL PART IS THE SECOND PART!

Interfaces

1. Make the code rely on abstract concepts, not on real implementation

Interfaces

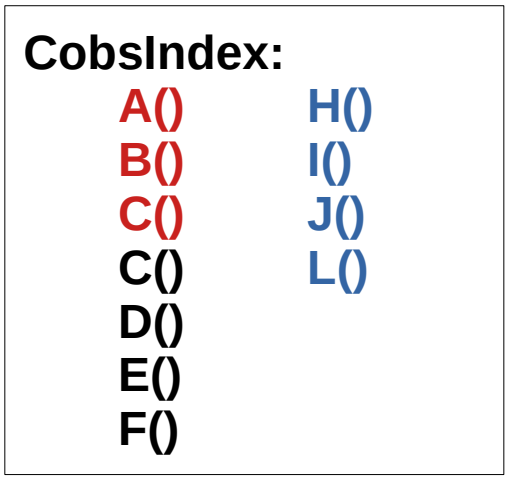
1. Make the code rely on abstract concepts, not on real implementation



If you code based on the concrete implementation of the BigsiIndex, your code will be biased towards this implementation;

Interfaces

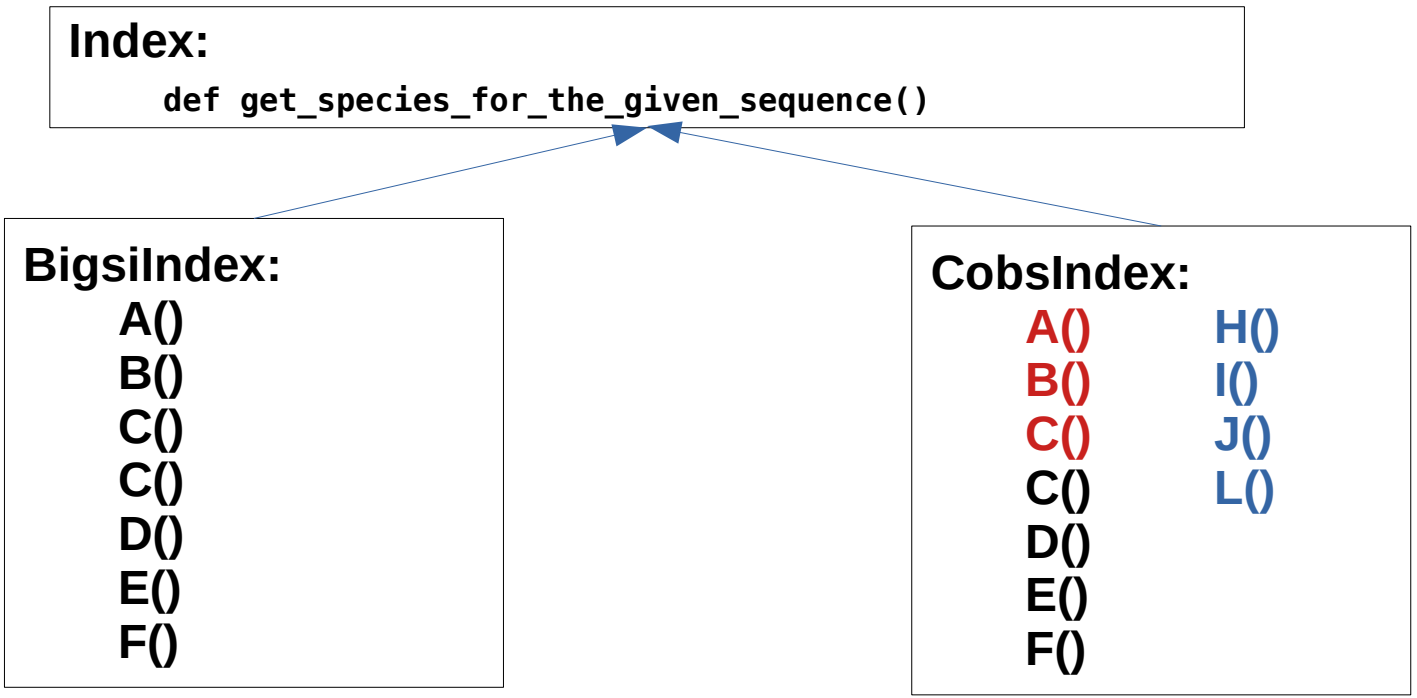
1. Make the code rely on abstract concepts, not on real implementation



If you change index, you might need to recode a lot of stuff because the indexes might have different functionalities

Interfaces

1. Make the code rely on abstract concepts, not on real implementation

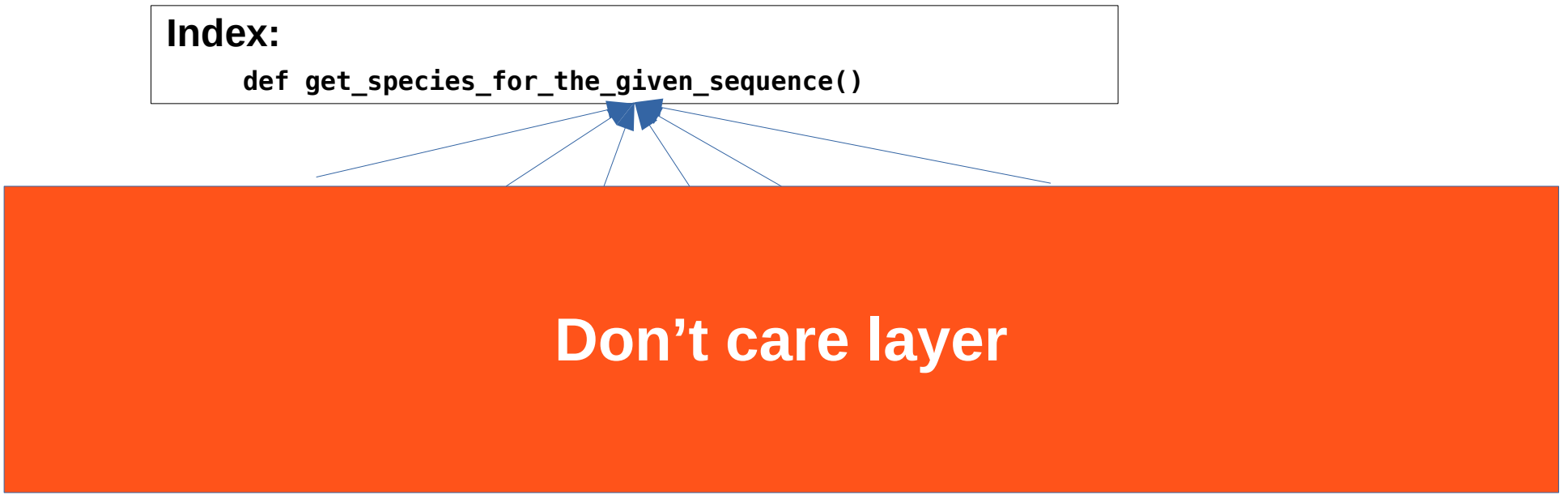


Interfaces/Abstract classes make your code rely on an abstract concept instead of concrete;
And forces any Index to implement these concepts

Interfaces

It makes even easier this presentation:

- With the Index interface, we don't really need to look at the implementation of the BigsiIndex, CobsIndex or ReindeerIndex;
- We know they respect our Index interface, and that is all that matters – that is how we communicate with an Index in our codebase;



SEE THE REST OF IMPLEMENTATION + TESTS