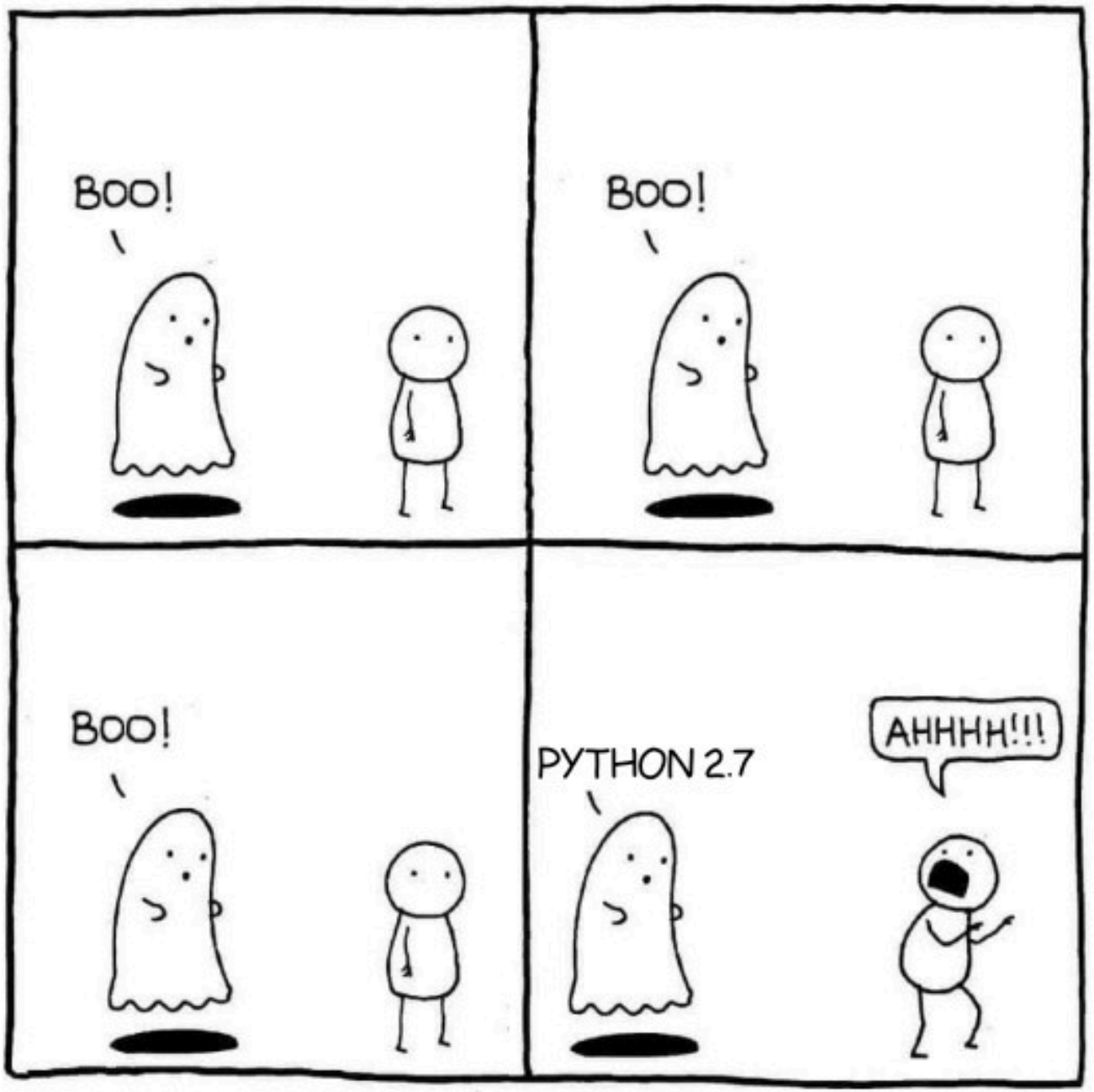
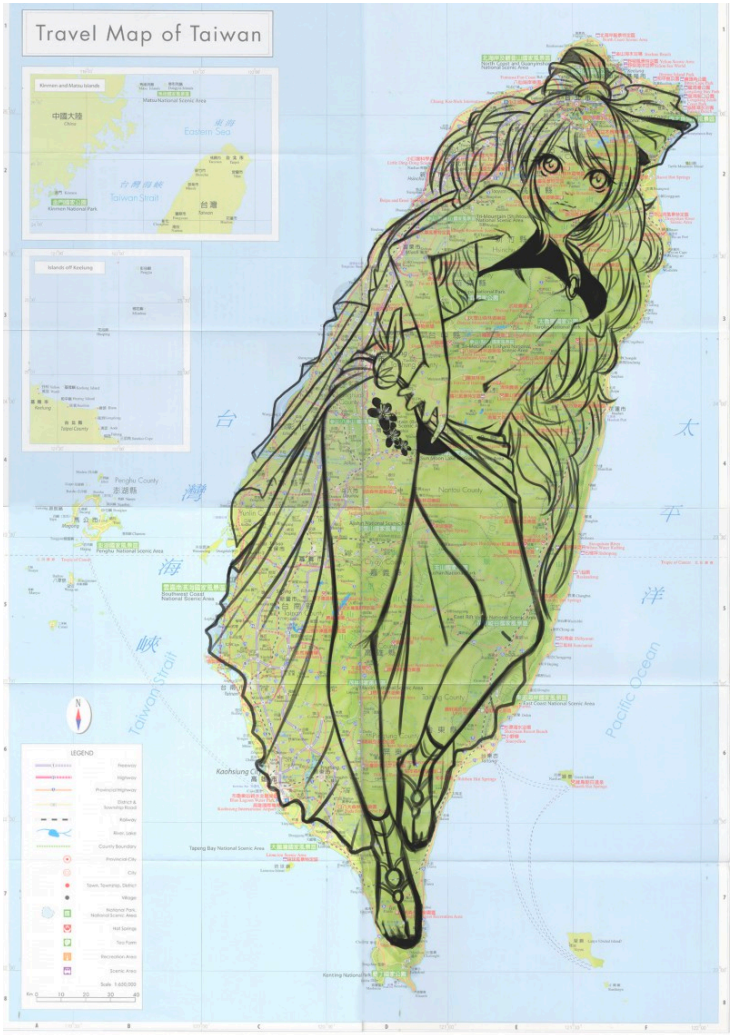

BOO!
BOO!
BOO!
PYTHON 2.7
AHHHH!!!

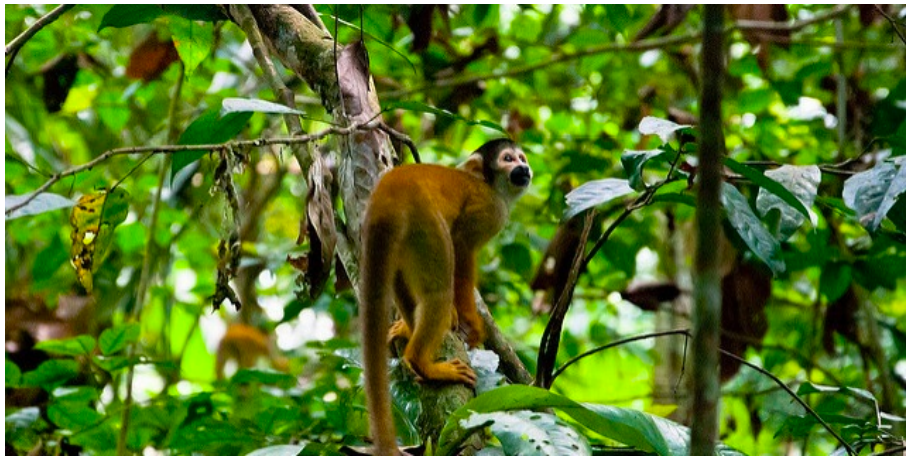
SupportingPython3
in Large Scale Project

Kir Chou

A9 (Amazon Product Search)

Kir Chou





amazon.co.jp

note35

kir.chou

Outline

- ❑ Background & Talks
- ❑ How to support python3?
- ❑ Execution Plan
- ❑ Lessons Learned
- ❑ Things can help you

Background

https://pythonclock.org/

Python 2.7 EOL
2020 Jan 1st

$Python2_{only}$ = Technical Debt

Talks:
Why Python3?

- Guido van Rossum (PyCascades 2018)
  BDFL Python 3 retrospective

- Victor Stinner (PyCon 2018)
  Python 3: ten years later

Talks:
How to make code Python2/3 compatible?

- Ned Batchelder (PyCon2012)
  Pragmatic Unicode
  Difference of String in python 2 and 3

- Brett Cannon (PyCon2015)
  author of SupportingPython3
  How to make your code Python 2/3 compatible
  Strategy to support python3

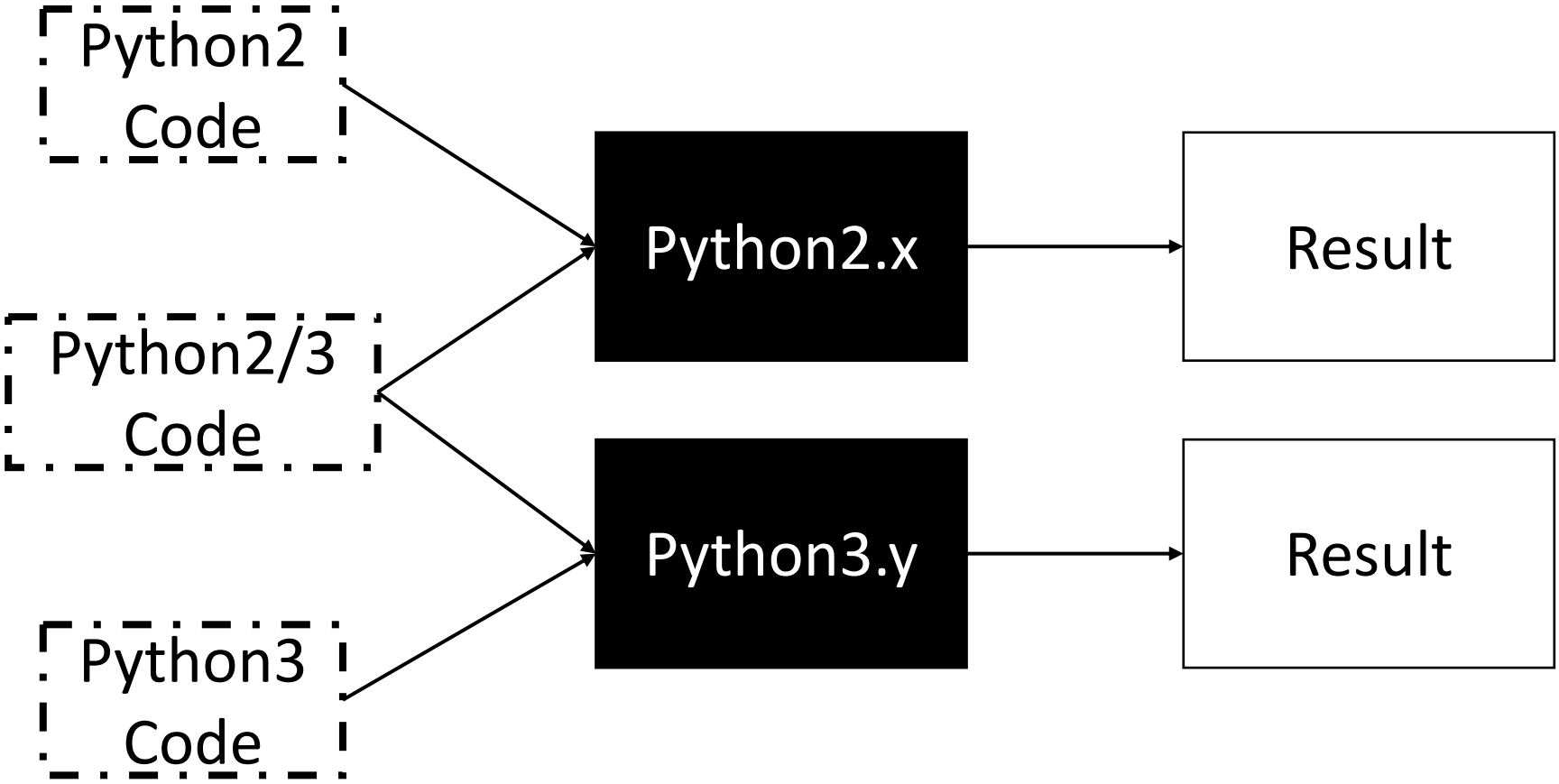
Talks:
Learn from others

- Jason Fried (PyCon2018)
  Fighting the Good Fight:
  Python 3 in your organization
  Facebook 5 years journey from python2 to 3

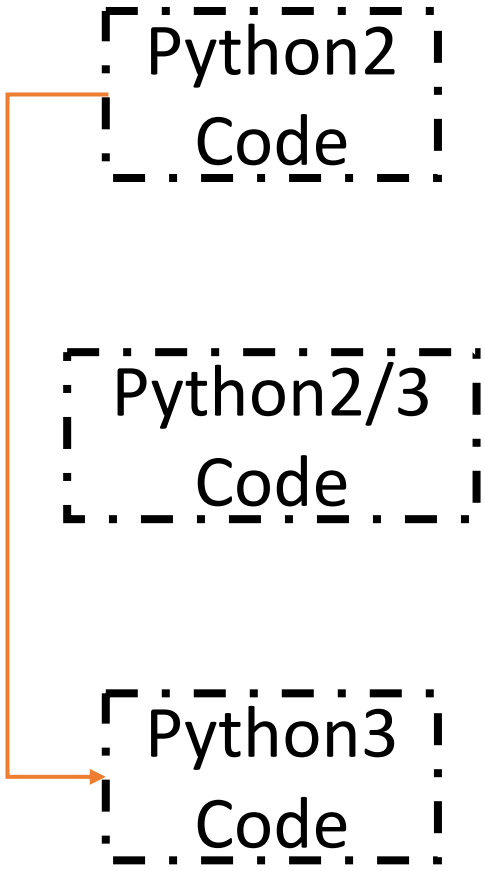
- Max Bélanger and Damien DeVille
  How we rolled out one of the largest Python 3 migrations ever
  Dropbox journey since 2015

How to support python3?

Ideal World

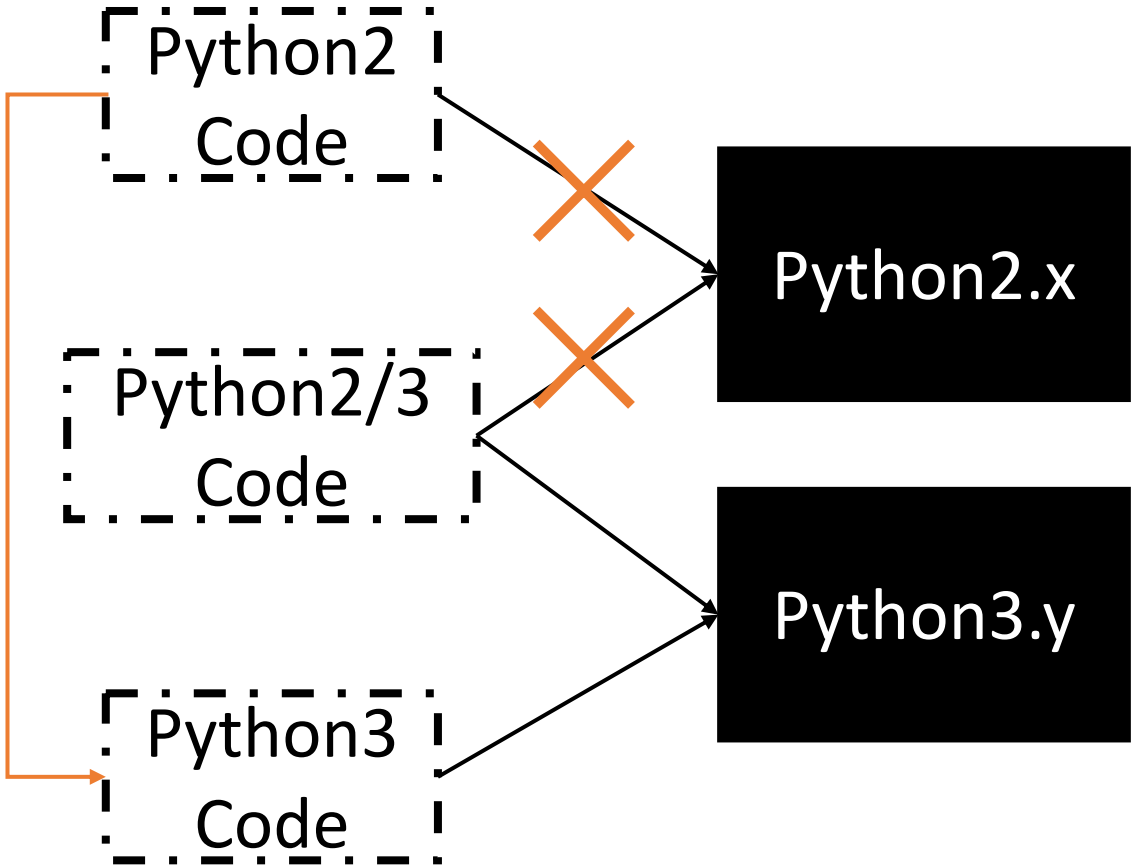

Python2
Code
Python2/3
Code
Python3
Code
Python2.x
Python3.y
Result
Result

Migration in Ideal World



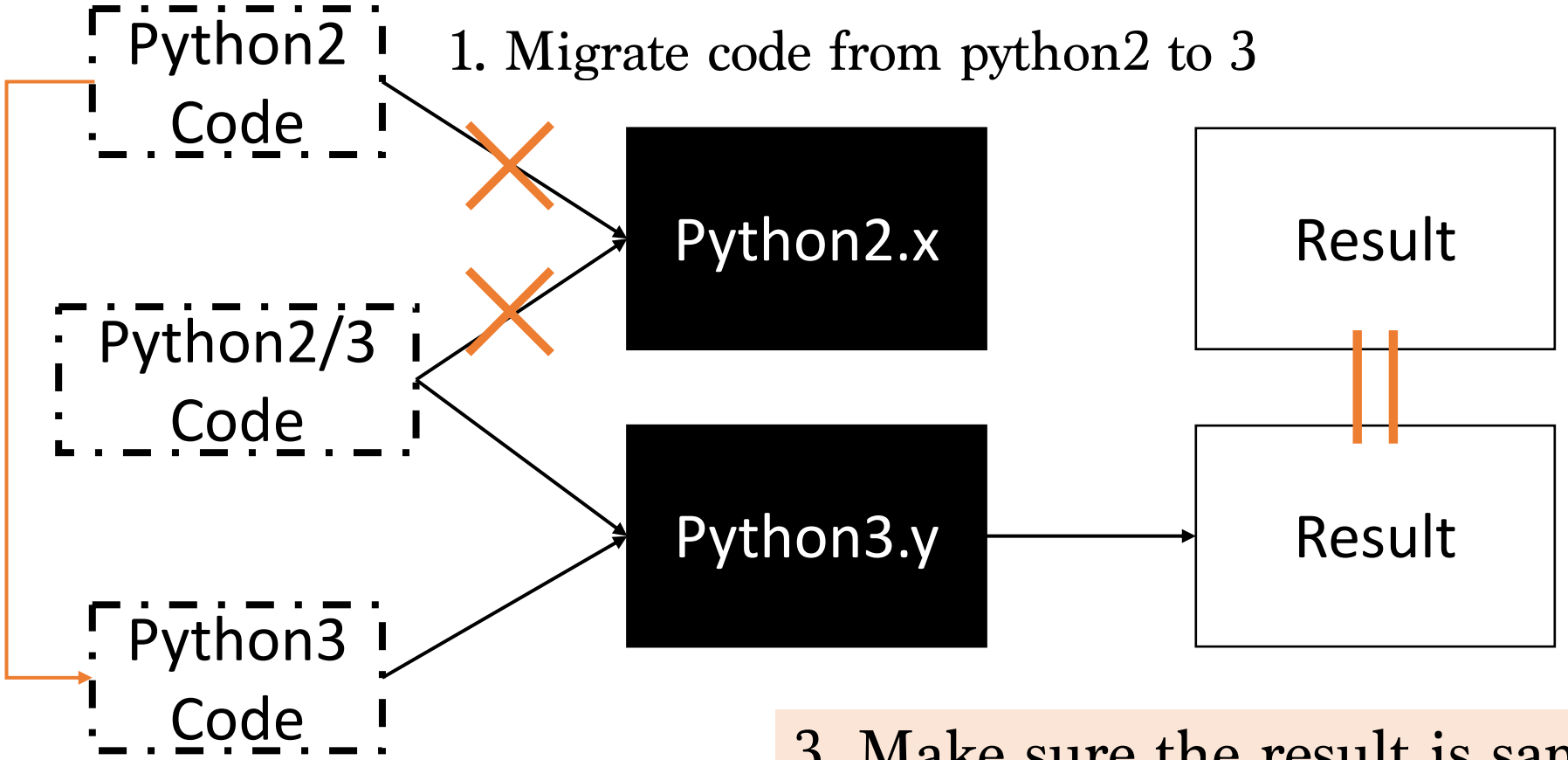
1. Make all code in python3

Migration in Ideal World



2. Stop executing code by python2 interpreter

Migration in Ideal World



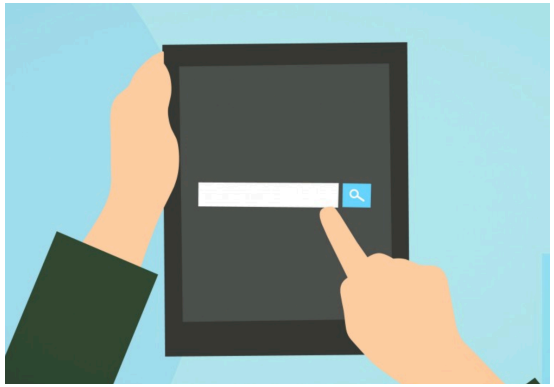
3. Make sure the result is same


WHAT I THOUGHT
WHAT IT
ACTUALLY LOOKS LIKE

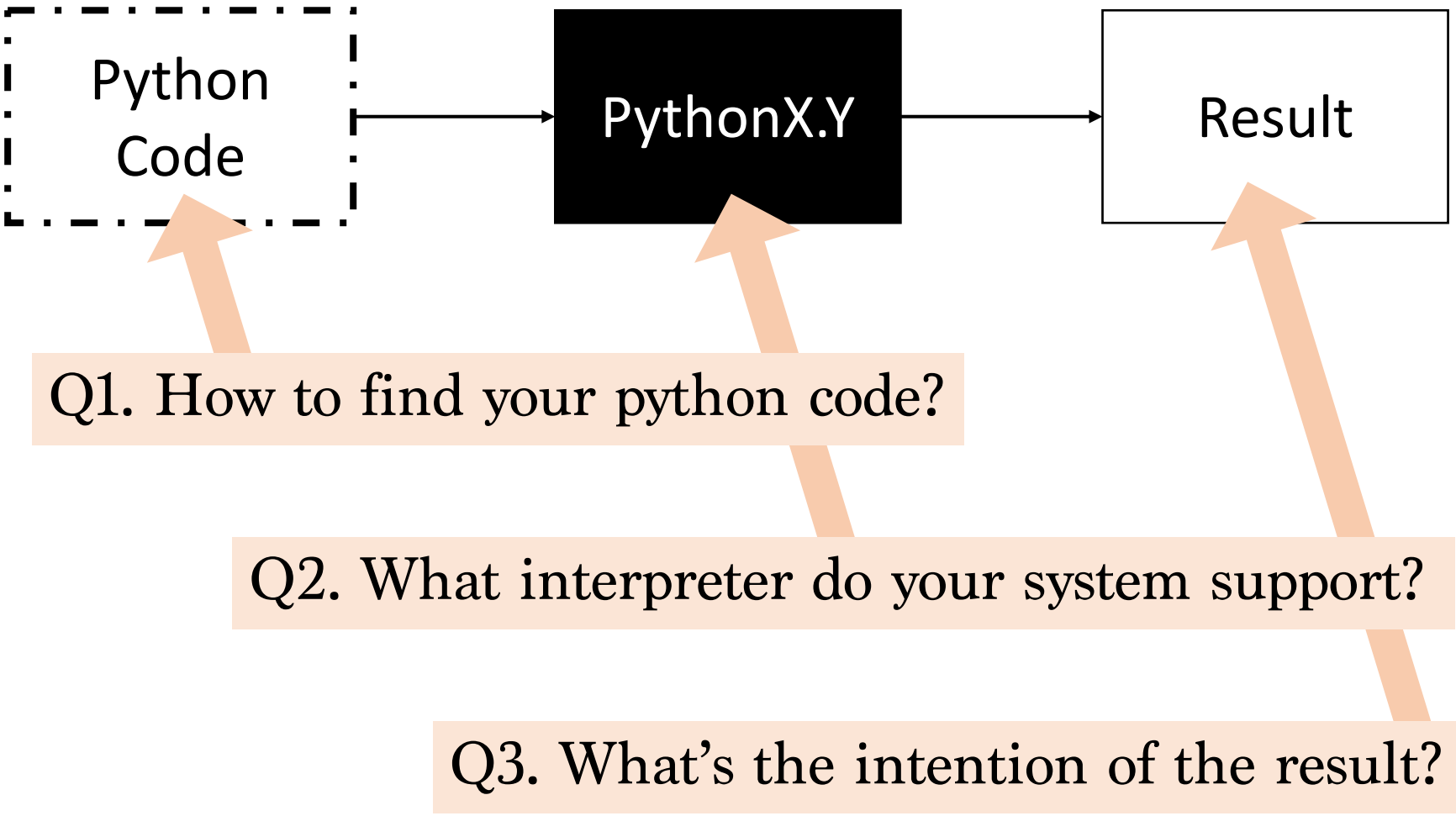
真・How to support python3?

Large Scale

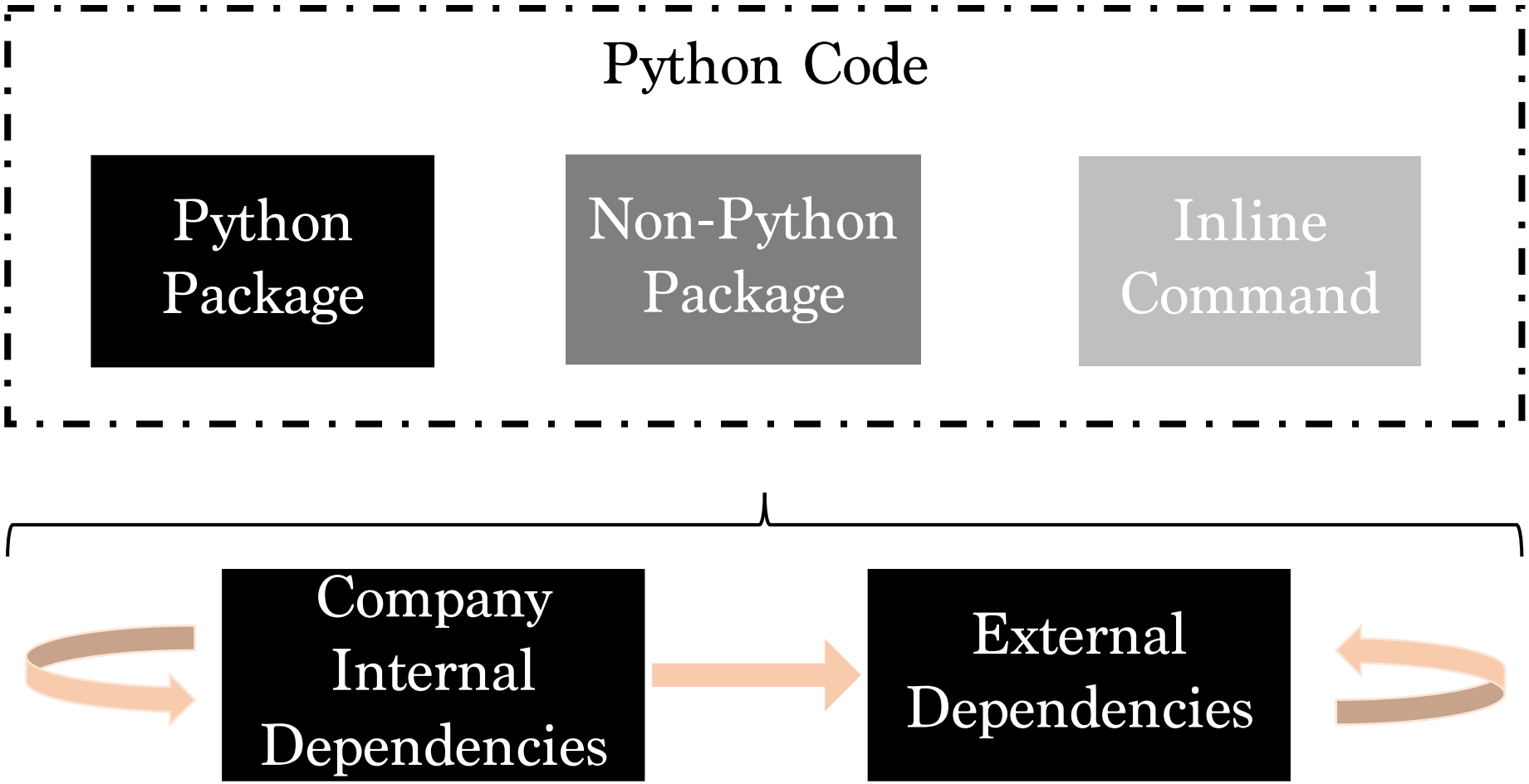
- Packages number: 3 digits
  - 10+ years old packages
  - Complicated relationship between packages
- Source line of code: Million level
- Number of production environment: 2 digits
- Build system for python: >1
- Deployment system: >1



Large Scale Project @ Real World



Q1. How to find your python code?

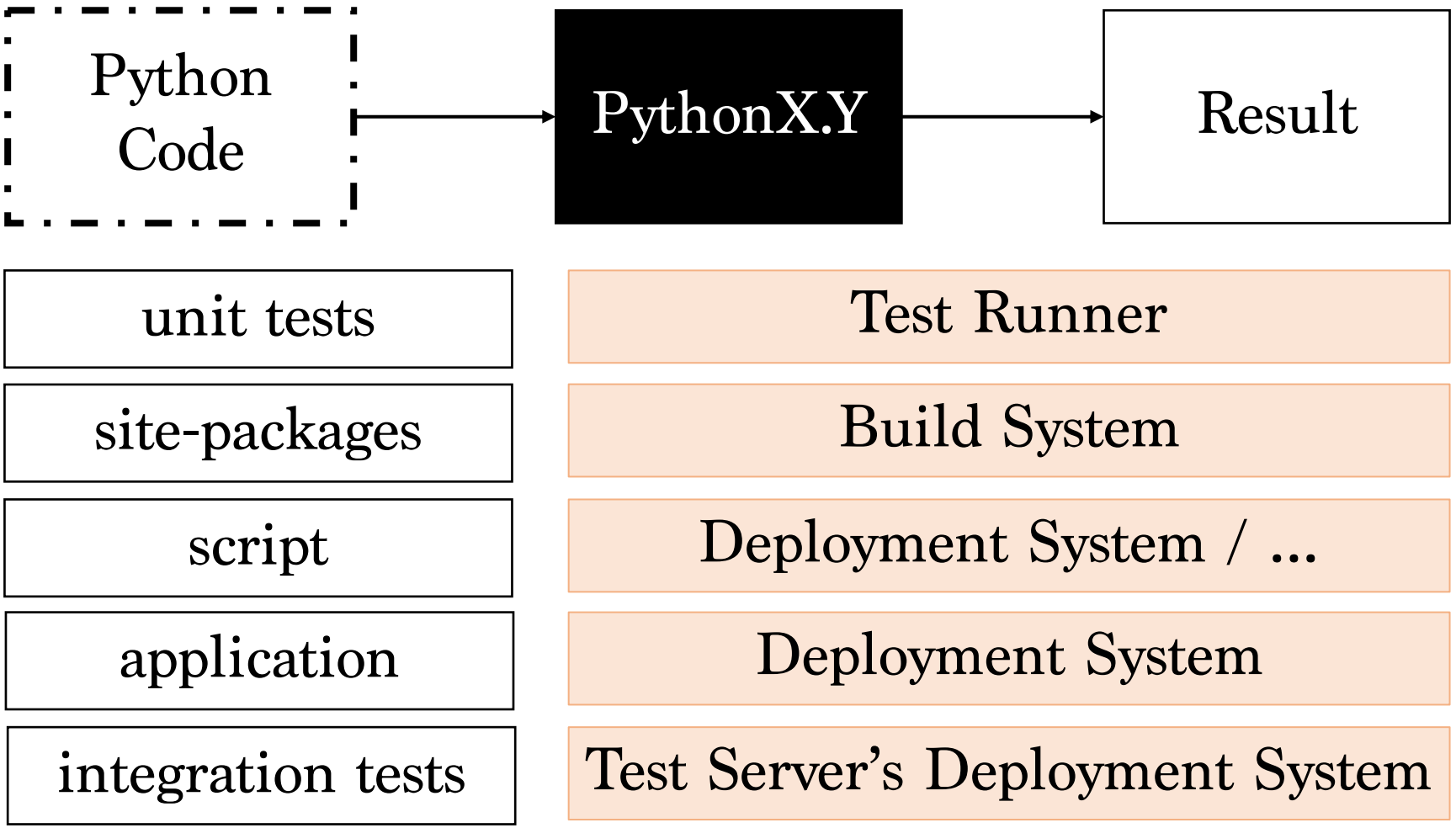


Q2. What interpreter do your system support?

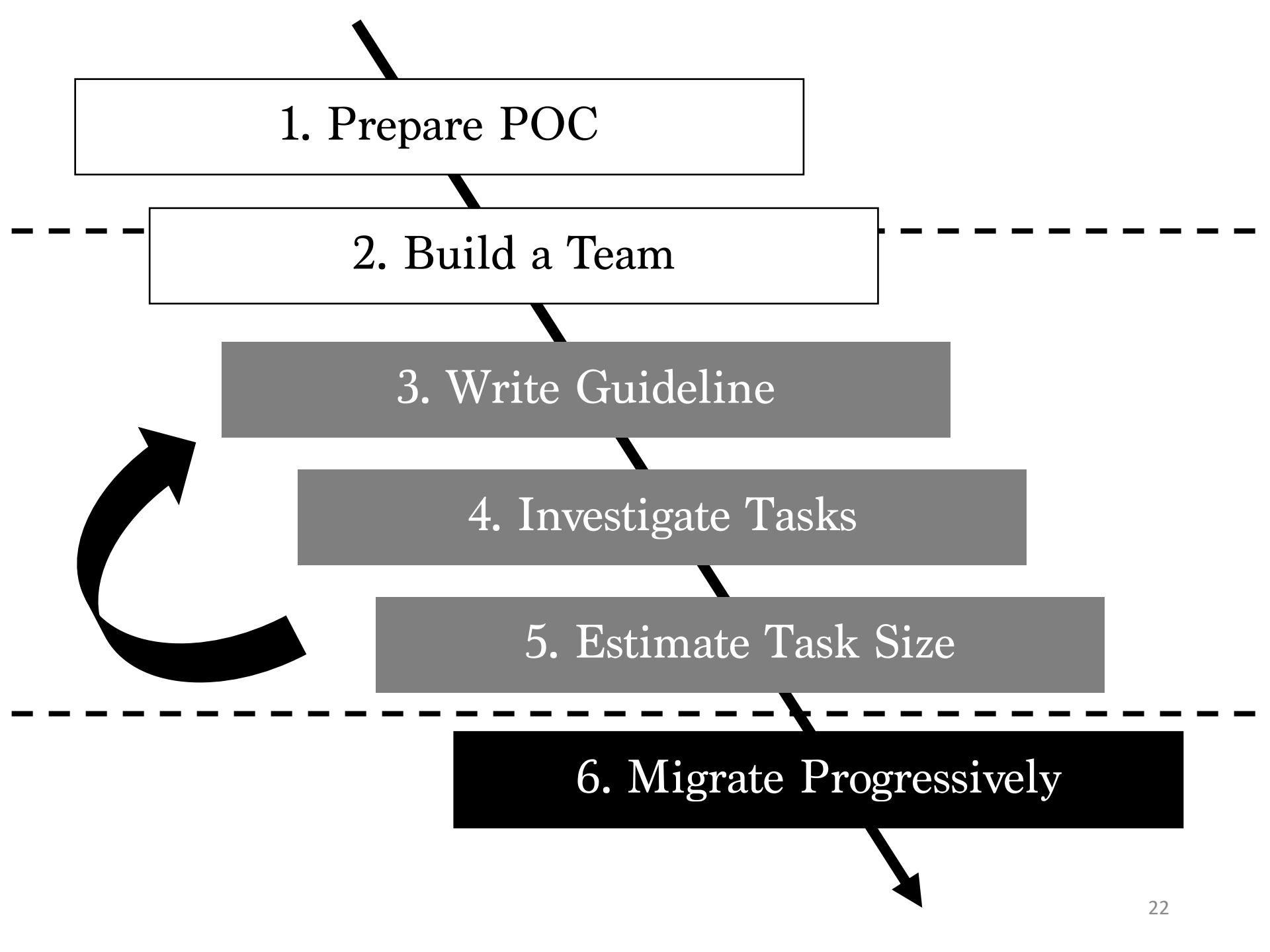
- Build & Deployment System
  - Default version of the system
  - Virtualenv
  - Dockerfile
  - ...

- How does your script decide the interpreter?
  - By interpreter (*pythonX.Y xxx.py*)
  - By shebang *(#!/usr/bin/env python)*
  - By build system to wrap the script
  - ...

Q3. What's the intention of the result?



Execution Plan


1. Prepare POC
2. Build a Team
3. Write Guideline
4. Investigate Tasks
5. Estimate Task Size
6. Migrate Progressively

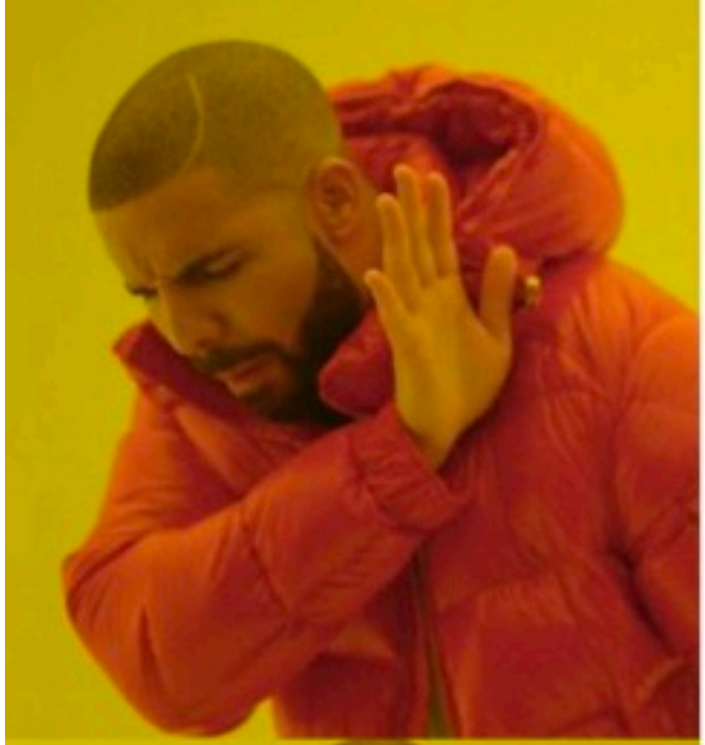
1. Prepare PoC

Example @ my github

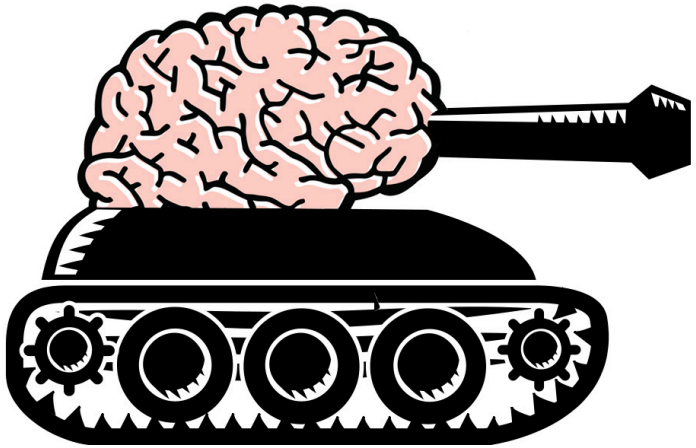
1. Find a tiny migratable package
2. Learn the approach to support python3
3. Prove of concept to support python3

4. Broadcast the idea to other knowledgeable people and human resource allocator

Do you need a team for Migration?

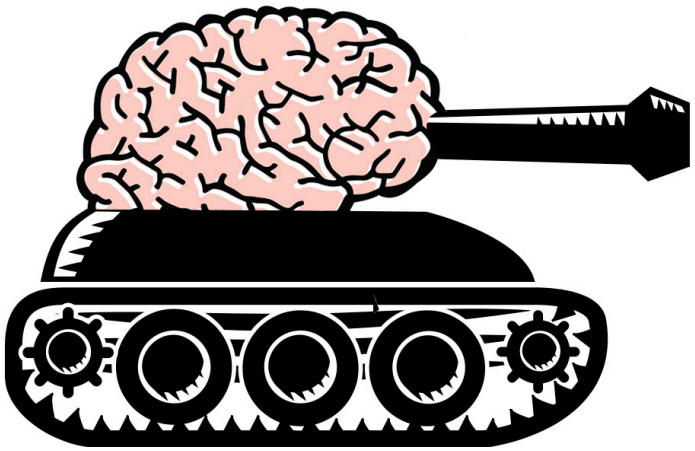


VOLUNTEER-BASED?



Gathering Knowledges

THINK TANK



Formulating the Guideline

Consultation

Sharing Knowledges

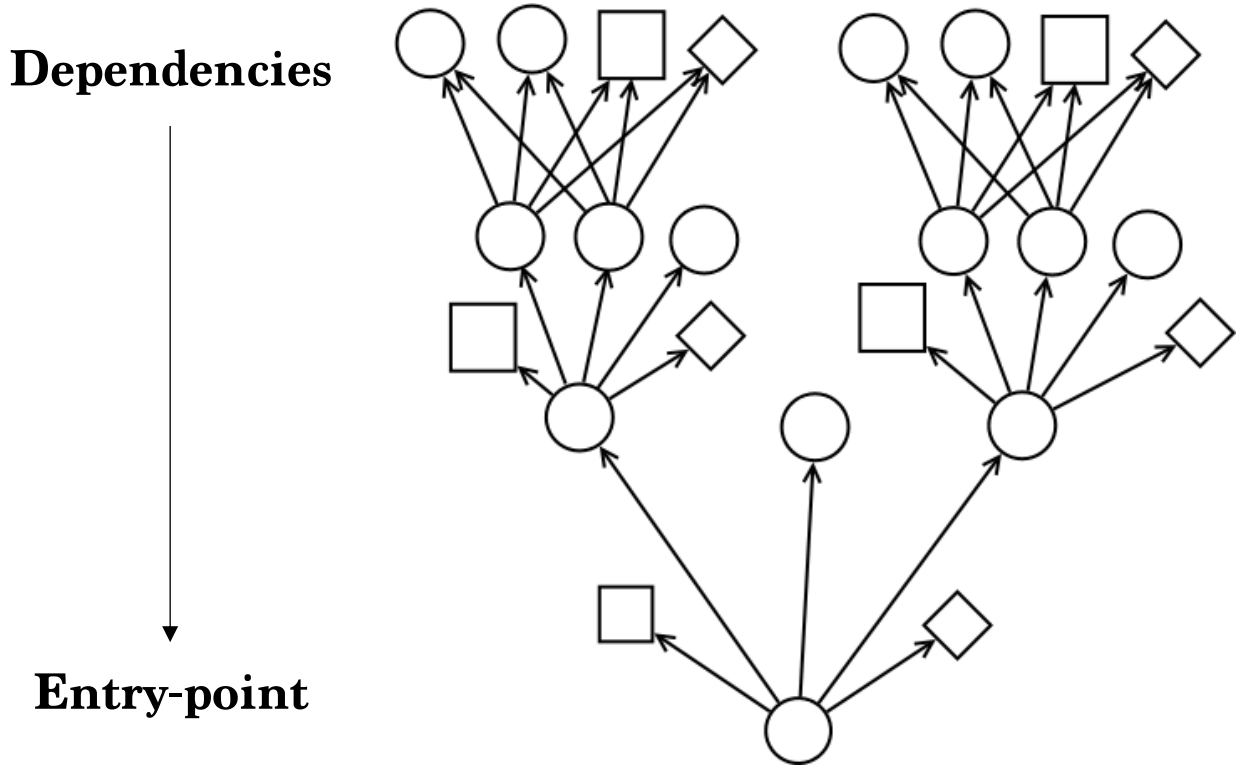
Entry point for external team

3. Write the Guideline

- How to SupportingPython3? (@Background)
  - How to handle complicated case like “str”?
- How to verify the “result”?
  - Test coverage / Local integration test
  - Linter (Optional)
  - Static type checker (Optional)
  - Integration Test (Not always feasible)
  - E2E test (Not always feasible)
- What’s the migration strategy? (Explain later)

4. Investigate Tasks (Q1)

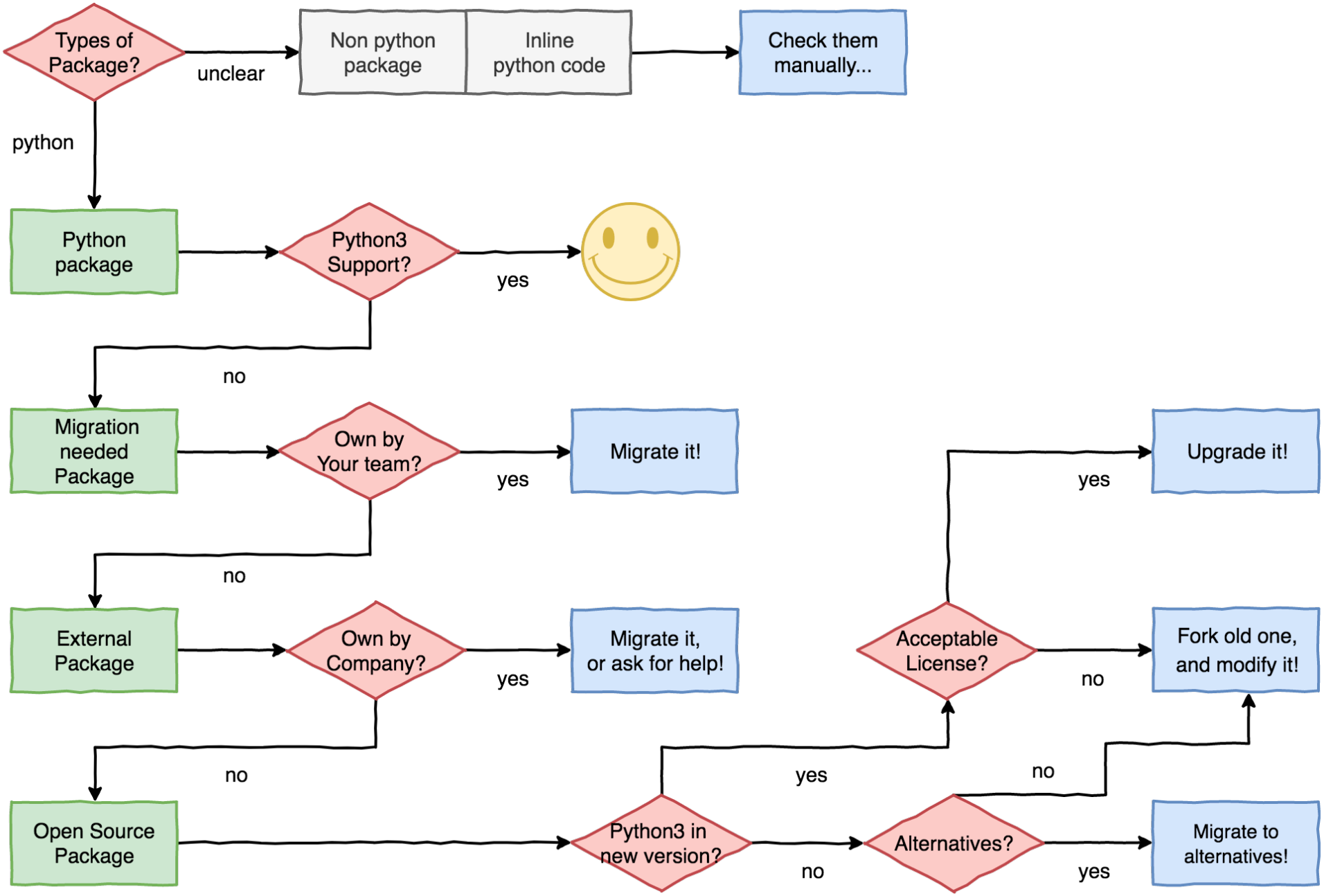
- Get dependency tree by the **Entry-point** (Script/Application)



4. Investigate Tasks (Q1)

🚨 For large scale internal project, there is NO general solution, caniusepython3 can only help external package

- We internally developed script to verify that...


Types of Package?
unclear
Non python package
Inline python code
Check them manually...
python
Python package
Python3 Support?
yes
no
Migration needed Package
Own by Your team?
yes
Migrate it!
no
External Package
Own by Company?
yes
Migrate it, or ask for help!
no
Open Source Package
Python3 in new version?
no
Alternatives?
yes
Migrate to alternatives!
no
Acceptable License?
no
Fork old one, and modify it!
yes
Upgrade it!
yes

4. Investigate Tasks (Q2 & Q3)

- Build System

  PyCon 2019: Thea Flowers (tox, nox)

  - Assure a reasonable coverage
  - Assure unit test passed in both versions

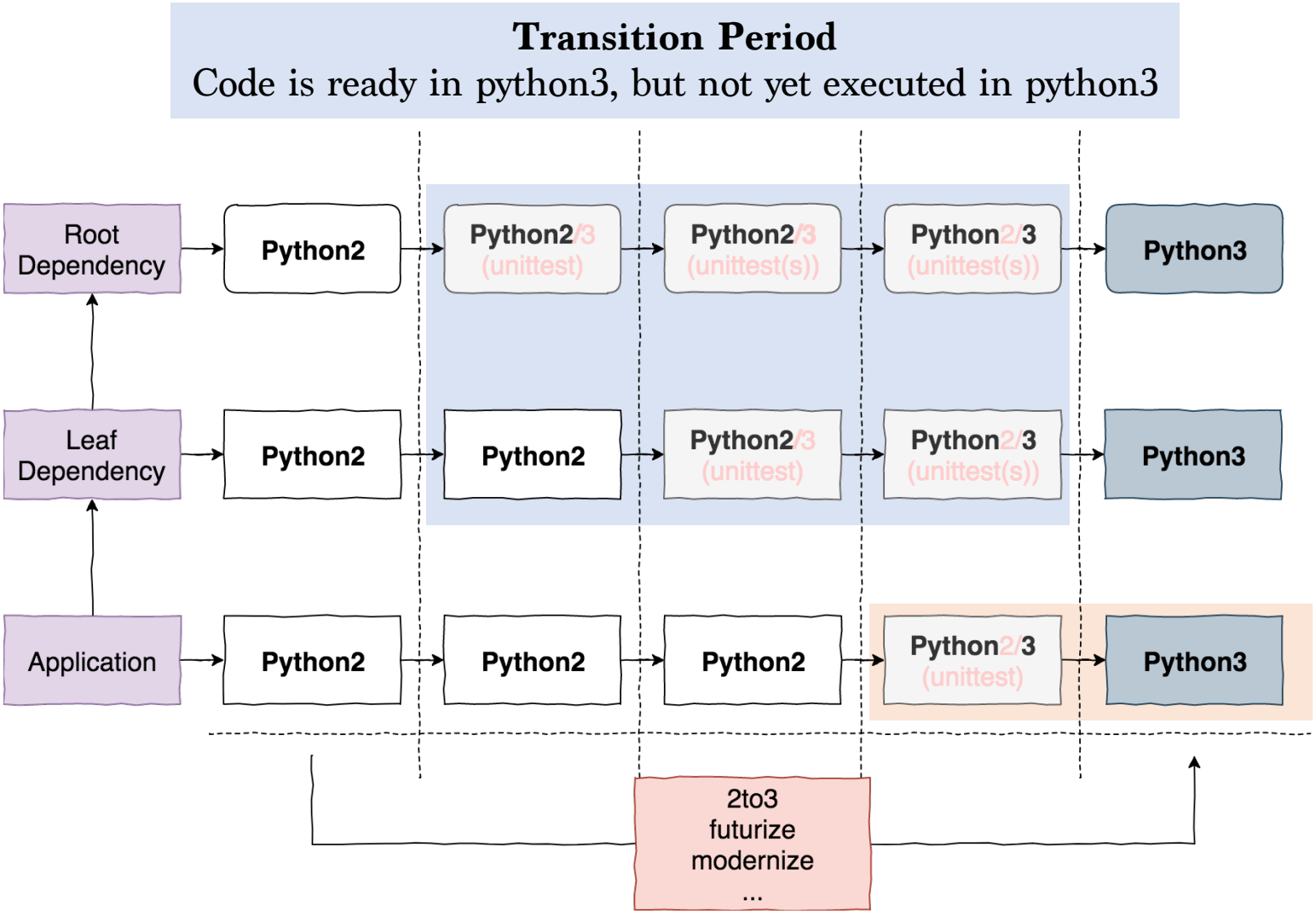
- Deployment System (In production)
  - Have site-packages in both versions
  - Execute part of the script/service in python3

5. Estimate Task Size

🚨 It's hard to measure the task size in general, here are few matrix we used

- Original code quality & test coverage
- Source lines of code (SLoC)
- Internal dependency but own by other team
- Edge case (Orphaned open source package)
- Build system efforts
- Deployment system efforts

6. Migrate Progressively (Transition Period)



6. Migrate Progressively (Progress Tracker)

- In the example
  - **CryptoUtil** was migrated and run in production service
  - **IndexingUtil** was migrated but not yet in any production service

	Size	Inspected	WIP	2/3 Compatible	Unit Tested	Partially In Prod	All In Prod
① Z dependency	1						
② Y dependency	4						
③ X Application	5						

Lessons Learned

Always be ready to face Unexpected Chores

- Build System / Deployment System
  - The old system only supports up to python34
    - ∵ Python34 is EOL, most packages used old system
      - ∴ Work on build system migration in parallel
    - ∵ The old system never run certain package's unit test
      - ∴ More bugs...
  - The new build system need workaround to execute linter, type checker in different python version
- Many unexpected package without python3 support
  - 3rd party package – orphaned package
  - Internal package

String

The most difficult part of migration

- If you haven't started anything
  - Follow the community approach is recommended
- Or, if you confirm the case can be covered by ascii
  - The ambiguous type: **str** can be accepted except some corner cases
- Write the guideline and give the presentation periodically!

Things can help you

Things can help you

- 2 major 3rd party libraries: **six/future**
  - **Select 1 of them, and add them to the guideline**

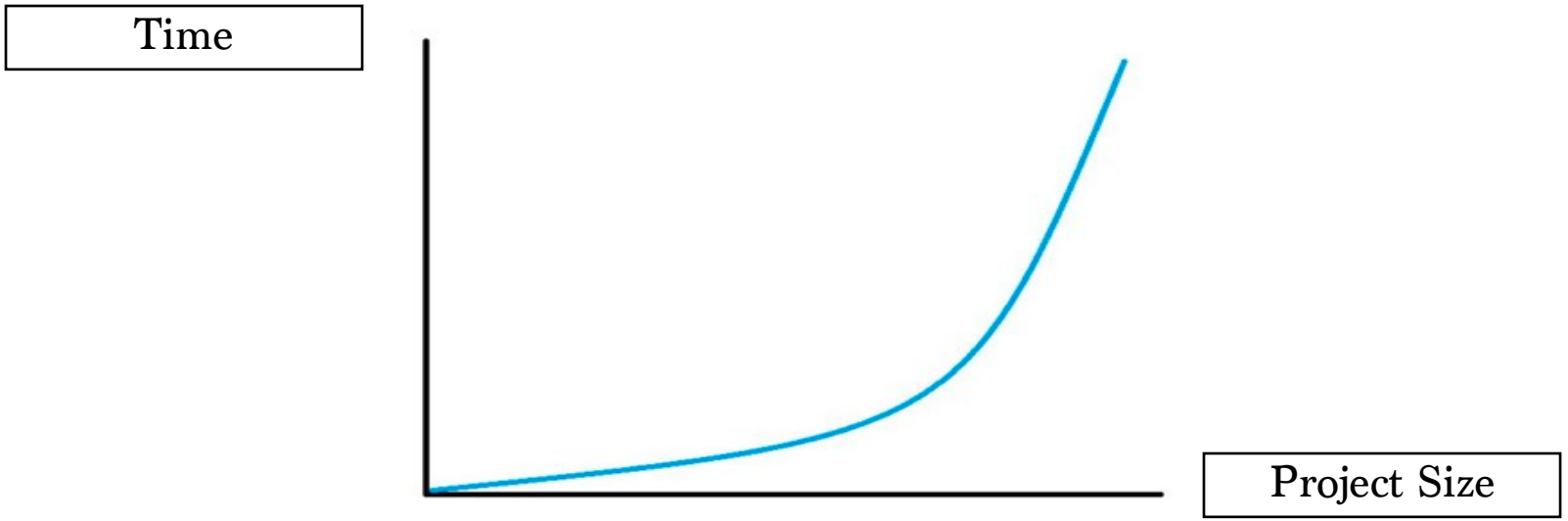
- 2 built-in solution: **2to3** and **__future__**
  - **2to3** is NOT very helpful as you need to understand the code in most cases
  - **__future__** is helpful, but you need to use them carefully
    - The usage of **unicode_literals** eventually be dropped

Things can help you (cont.)

- Linter
  - **pylint** or **flake8** are recommended
  - Since you need to touch most packages, it's good to unify the syntax in pass
  - **autopep8, black, and yapf**
- Static Type Checker
  - **mypy** is recommended (pyre-check only support 3.5+)
  - Types help the progress of code review
    - By dropbox and our experience

Summary

- SupportPython3 is **difficult** and **long**



- What's the benefit for doing this?
  - Get better understanding of extremely old packages
  - Make code with higher quality
  - Ready for next "**migration**"


Thank You!