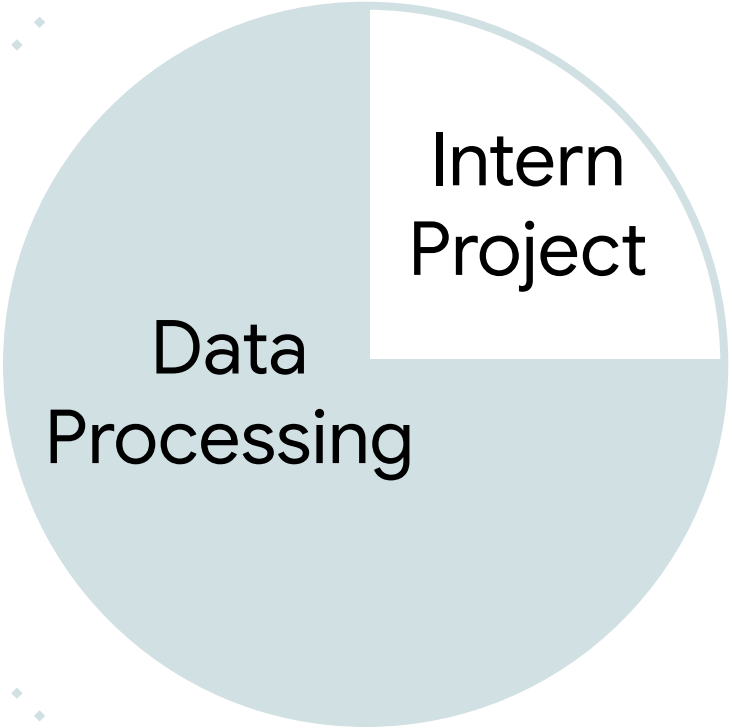
How to Design a Successful (Intern) Project with Apache Beam?

Kir Chou
PyCon TW 2023

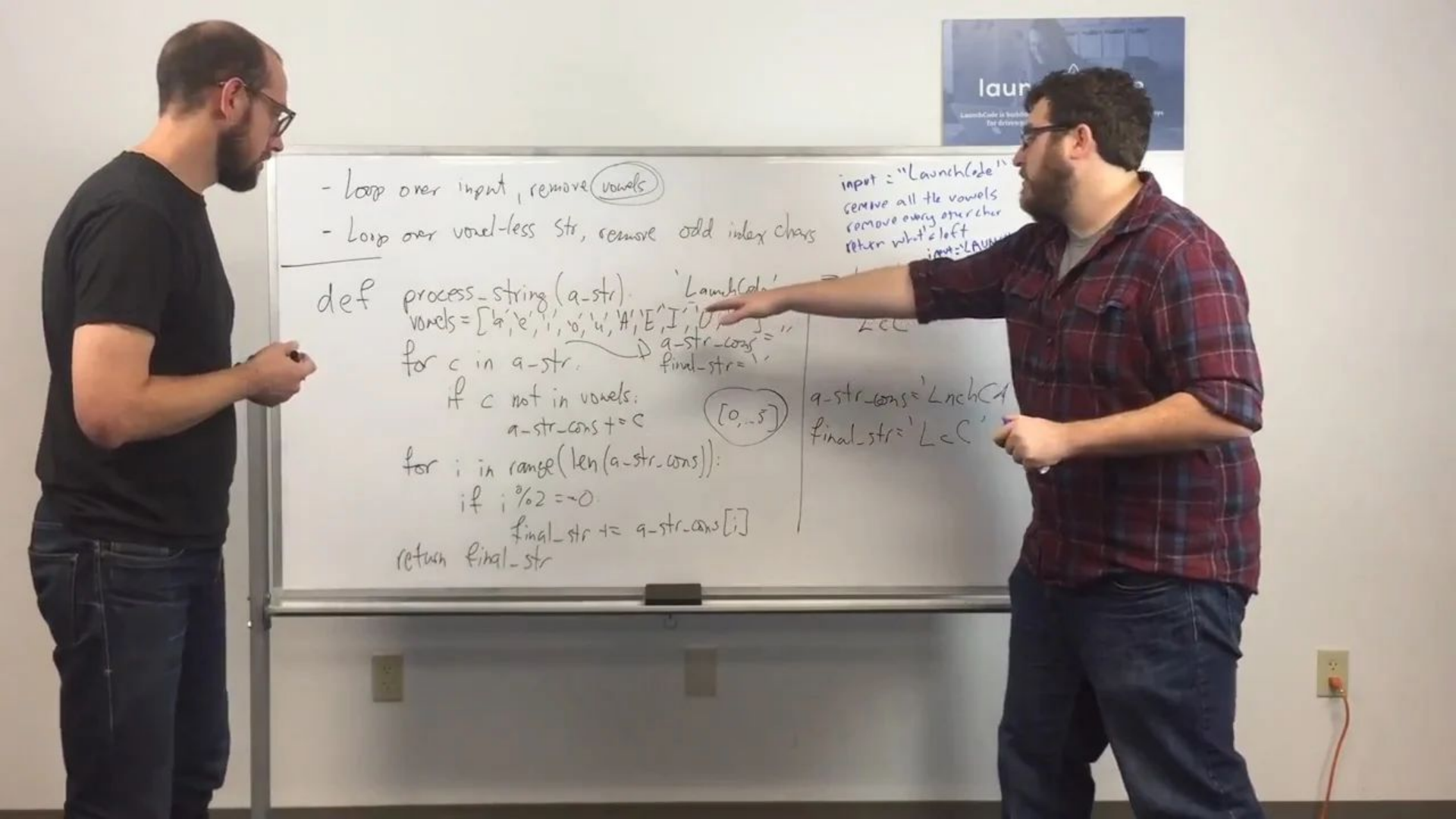

Intern
Project
Data
Processing

note35.github.io/about

Slide

Code @ Colab

Anyone knows Apache Beam?


- Loop over input, remove vowels
- Loop over vowel-less str, remove odd index chars
input = "LaunchCode"
remove all the vowels
remove every other char
return what's left
def process_string(a_str):
vowels = ['a','e','i','o','u','A','E','I',
for c in a_str:
if c not in vowels:
a_str_cons += c
for i in range(len(a_str_cons)):
if i%2 == 0:
final_str += a_str_cons[i]
return final_str
a_str_cons = ''
final_str = ''
[0,..5]
a_str_cons = 'LnchC'
final_str = 'LcC'

20s

Given two tables:
- Table A: **hash key (string)** to **case sensitive index (string)**
- Table B: **hash key (string)** to data in whatever type

Both tables **hash key** are based on the **case sensitive index**

Generate a new table that maps **new hash key** based on **case insensitive index** to data in Table B.

Input Table A

```
hash('PHP'): 'PHP'
hash('python'): 'python'
hash('C++'): 'C++'
```

Input Table B

```
hash('PHP'): 'The best'
hash('python'): 'Better than the best'
```

Output table

```
hash('php'): 'The best'
hash('python'): 'Better than the best'
hash('c++'): '?'
```

```
def generate_case_insensitive_index_to_content(
  hash_to_case_sensitive_index: dict[int, str],
  hash_to_content: dict[int, str]
) -> dict[int, str]:
  case_insensitive_index_to_content: dict[int, str] = {}

  for existing_hash, index in hash_to_case_sensitive_index.items():
    case_insensitive_index: str = index.lower()
    new_hash: int = magic_hash(case_insensitive_index)

    if existing_hash in hash_to_content:
      case_insensitive_index_to_content[new_hash] =\
        hash_to_content[existing_hash]
    else: case_insensitive_index_to_content[new_hash] =\
        NEW_CONTENT_PREFIX.format(case_insensitive_index)

  return case_insensitive_index_to_content
```

```
def generate_case_insensitive_index_to_content(
  hash_to_case_sensitive_index: dict[int, str],
  hash_to_content: dict[int, str]
) -> dict[int, str]:
  case_insensitive_index_to_content: dict[int, str] = {}

  for existing_hash, index in hash_to_case_sensitive_index.items():
    case_insensitive_index: str = index.lower()
    new_hash: int = magic_hash(case_insensitive_index)

    if existing_hash in hash_to_content:
      case_insensitive_index_to_content[new_hash] =\
        hash_to_content[existing_hash]
    else: case_insensitive_index_to_content[new_hash] =\
        NEW_CONTENT_PREFIX.format(case_insensitive_index)

  return case_insensitive_index_to_content
```

```
hash('PHP'): 'PHP'
hash('python'): 'python'
hash('C++'): 'C++'
```

∅

```
def generate_case_insensitive_index_to_content(
  hash_to_case_sensitive_index: dict[int, str],
  hash_to_content: dict[int, str]
) -> dict[int, str]:
  case_insensitive_index_to_content: dict[int, str] = {}

  for existing_hash, index in hash_to_case_sensitive_index.items():
    case_insensitive_index: str = index.lower()
    new_hash: int = magic_hash(case_insensitive_index)

    if existing_hash in hash_to_content:
      case_insensitive_index_to_content[new_hash] =\
        hash_to_content[existing_hash]
    else: case_insensitive_index_to_content[new_hash] =\
        NEW_CONTENT_PREFIX.format(case_insensitive_index)

  return case_insensitive_index_to_content
```

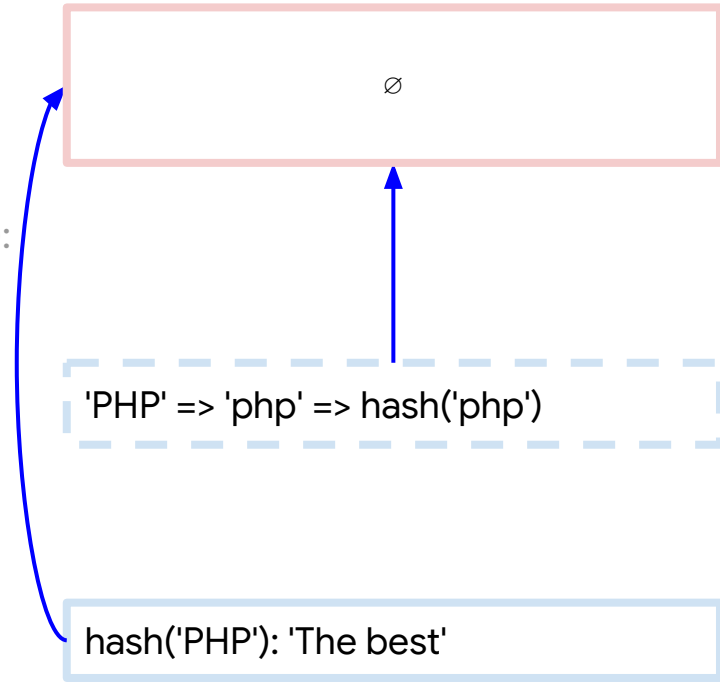
hash('PHP'): 'PHP'
hash('python'): 'python'
hash('C++'): 'C++'

∅

hash('PHP'): 'PHP'

'PHP' => 'php' => hash('php')

```python
def generate_case_insensitive_index_to_content(
  hash_to_case_sensitive_index: dict[int, str],
  hash_to_content: dict[int, str]
) -> dict[int, str]:
  case_insensitive_index_to_content: dict[int, str] = {}

  for existing_hash, index in hash_to_case_sensitive_index.items():
    case_insensitive_index: str = index.lower()
    new_hash: int = magic_hash(case_insensitive_index)

    if existing_hash in hash_to_content:
      case_insensitive_index_to_content[new_hash] =\
        hash_to_content[existing_hash]
    else: case_insensitive_index_to_content[new_hash] =\
        NEW_CONTENT_PREFIX.format(case_insensitive_index)

  return case_insensitive_index_to_content
```



```
def generate_case_insensitive_index_to_content(
  hash_to_case_sensitive_index: dict[int, str],
  hash_to_content: dict[int, str]
) -> dict[int, str]:
  case_insensitive_index_to_content: dict[int, str] = {}

  for existing_hash, index in hash_to_case_sensitive_index.items():
    case_insensitive_index: str = index.lower()
    new_hash: int = magic_hash(case_insensitive_index)

    if existing_hash in hash_to_content:
      case_insensitive_index_to_content[new_hash] =\
        hash_to_content[existing_hash]
    else: case_insensitive_index_to_content[new_hash] =\
        NEW_CONTENT_PREFIX.format(case_insensitive_index)

  return case_insensitive_index_to_content
```

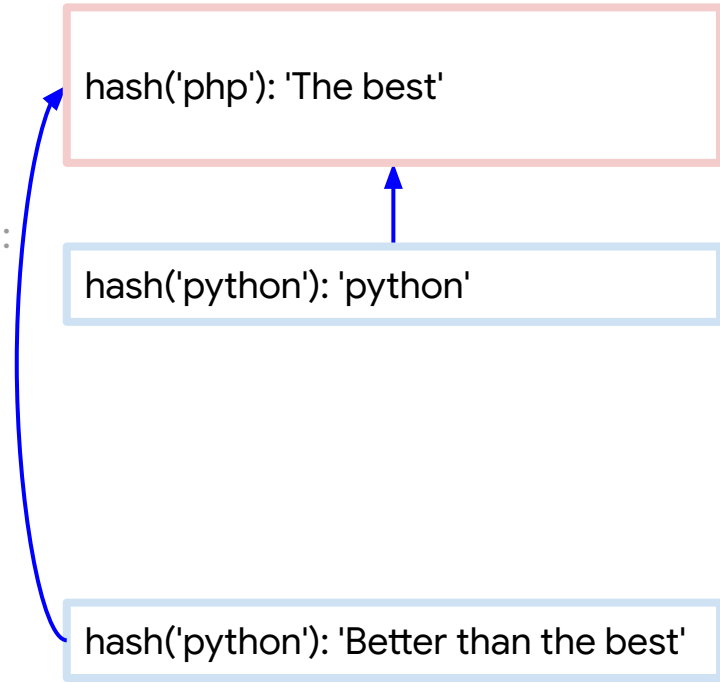
hash('PHP'): 'PHP'
hash('python'): 'python'
hash('C++'): 'C++'

hash('php'): 'The best'

hash('python'): 'python'

'python' => 'python' => hash('python')

```python
def generate_case_insensitive_index_to_content(
  hash_to_case_sensitive_index: dict[int, str],
  hash_to_content: dict[int, str]
) -> dict[int, str]:
  case_insensitive_index_to_content: dict[int, str] = {}

  for existing_hash, index in hash_to_case_sensitive_index.items():
    case_insensitive_index: str = index.lower()
    new_hash: int = magic_hash(case_insensitive_index)

    if existing_hash in hash_to_content:
      case_insensitive_index_to_content[new_hash] =\
        hash_to_content[existing_hash]
    else: case_insensitive_index_to_content[new_hash] =\
        NEW_CONTENT_PREFIX.format(case_insensitive_index)

  return case_insensitive_index_to_content
```



```
def generate_case_insensitive_index_to_content(
  hash_to_case_sensitive_index: dict[int, str],
  hash_to_content: dict[int, str]
) -> dict[int, str]:
  case_insensitive_index_to_content: dict[int, str] = {}

  for existing_hash, index in hash_to_case_sensitive_index.items():
    case_insensitive_index: str = index.lower()
    new_hash: int = magic_hash(case_insensitive_index)

    if existing_hash in hash_to_content:
      case_insensitive_index_to_content[new_hash] =\
        hash_to_content[existing_hash]
    else: case_insensitive_index_to_content[new_hash] =\
        NEW_CONTENT_PREFIX.format(case_insensitive_index)

  return case_insensitive_index_to_content
```

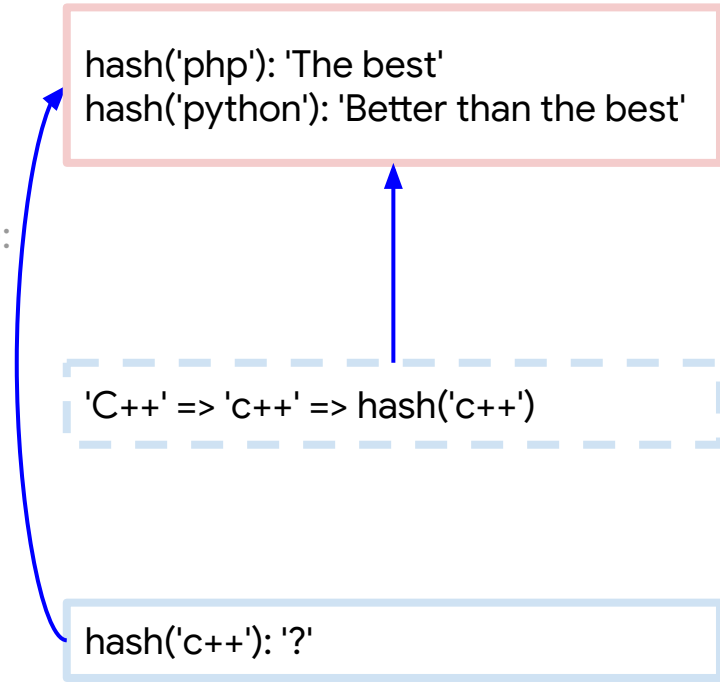
hash('PHP'): 'PHP'
hash('python'): 'python'
hash('C++'): 'C++'

hash('php'): 'The best'
hash('python'): 'Better than the best'

hash('C++'): 'C++'

'C++' => 'c++' => hash('c++')

```python
def generate_case_insensitive_index_to_content(
  hash_to_case_sensitive_index: dict[int, str],
  hash_to_content: dict[int, str]
) -> dict[int, str]:
  case_insensitive_index_to_content: dict[int, str] = {}

  for existing_hash, index in hash_to_case_sensitive_index.items():
    case_insensitive_index: str = index.lower()
    new_hash: int = magic_hash(case_insensitive_index)

    if existing_hash in hash_to_content:
      case_insensitive_index_to_content[new_hash] =\
        hash_to_content[existing_hash]
    else: case_insensitive_index_to_content[new_hash] =\
      NEW_CONTENT_PREFIX.format(case_insensitive_index)

  return case_insensitive_index_to_content
```



```
def generate_case_insensitive_index_to_content(
  hash_to_case_sensitive_index: dict[int, str],
  hash_to_content: dict[int, str]
) -> dict[int, str]:
  case_insensitive_index_to_content: dict[int, str] = {}

  for existing_hash, index in hash_to_case_sensitive_index.items():
    case_insensitive_index: str = index.lower()
    new_hash: int = magic_hash(case_insensitive_index)

    if existing_hash in hash_to_content:
      case_insensitive_index_to_content[new_hash] =\
        hash_to_content[existing_hash]
    else: case_insensitive_index_to_content[new_hash] =\
        NEW_CONTENT_PREFIX.format(case_insensitive_index)

  return case_insensitive_index_to_content
```

```
hash('php'): 'The best'
hash('python'): 'Better than the best'
hash('c++'): '?'
```

Path to
modern data processing

```
def generate_case_insensitive_index_to_content(
  hash_to_case_sensitive_index: dict[int, str],
  hash_to_content: dict[int, str]
) -> dict[int, str]:
  case_insensitive_index_to_content: dict[int, str] = {}

  for existing_hash, index in hash_to_case_sensitive_index.items():
    case_insensitive_index: str = index.lower()
    new_hash: int = magic_hash(case_insensitive_index)

    if existing_hash in hash_to_content:
      case_insensitive_index_to_content[new_hash] =\
        hash_to_content[existing_hash]
    else: case_insensitive_index_to_content[new_hash] =\
        NEW_CONTENT_PREFIX.format(case_insensitive_index)

  return case_insensitive_index_to_content
```

```
def generate_case_insensitive_index_to_content(
  hash_to_case_sensitive_index: dict[int, str],
  hash_to_content: dict[int, str]
) -> dict[int, str]:
  case_insensitive_index_to_content: dict[int, str] = {}

  for existing_hash, index in hash_to_case_sensitive_index.items():
    case_insensitive_index: str = index.
    new_hash: int = magic_hash(case_i

    if existing_hash in hash_to_content
      case_insensitive_index_to_content[new_hash] =\
      hash_to_content[existing_hash]
    else: case_insensitive_index_to_content[new_hash] =\
      NEW_CONTENT_PREFIX.format(case_insensitive_index)

  return case_insensitive_index_to_content
```

hash('php'): 'The best'
hash('python'): 'Better than the best'
hash('c++'): '?'

hash('C++'): 'Stop others from learning it'

1970/01/01 11:11:11


```
hash('php'): 'The best'
hash('python'): 'Better than the best'
hash('c++'): 'Stop others from learning it'
```

1970/01/01 11:11:11

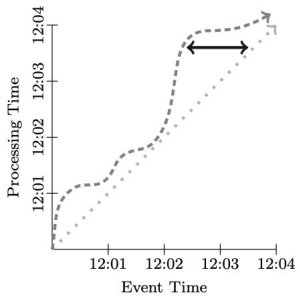
```
hash('C++'): 'The legend'
```

1970/01/01 11:11:12 ⭕

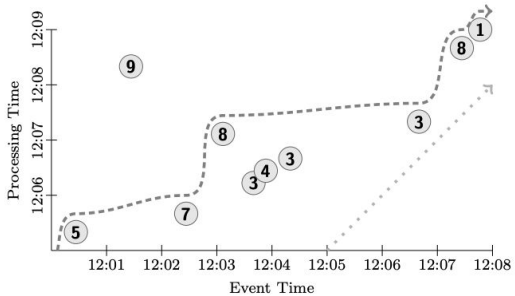
1970/01/01 11:11:10 ❌

Recap: modern data processing

- **Massive-Scale**: data is too big to handle by one instance
- **Unbounded**: data keeps coming, and requires to handle them when receiving
  - eg: peak keywords detection of the search engine
- **Out-of-Order**: the event time and the process time may be different
  - eg: game score update for users who play it without the internet access tentatively

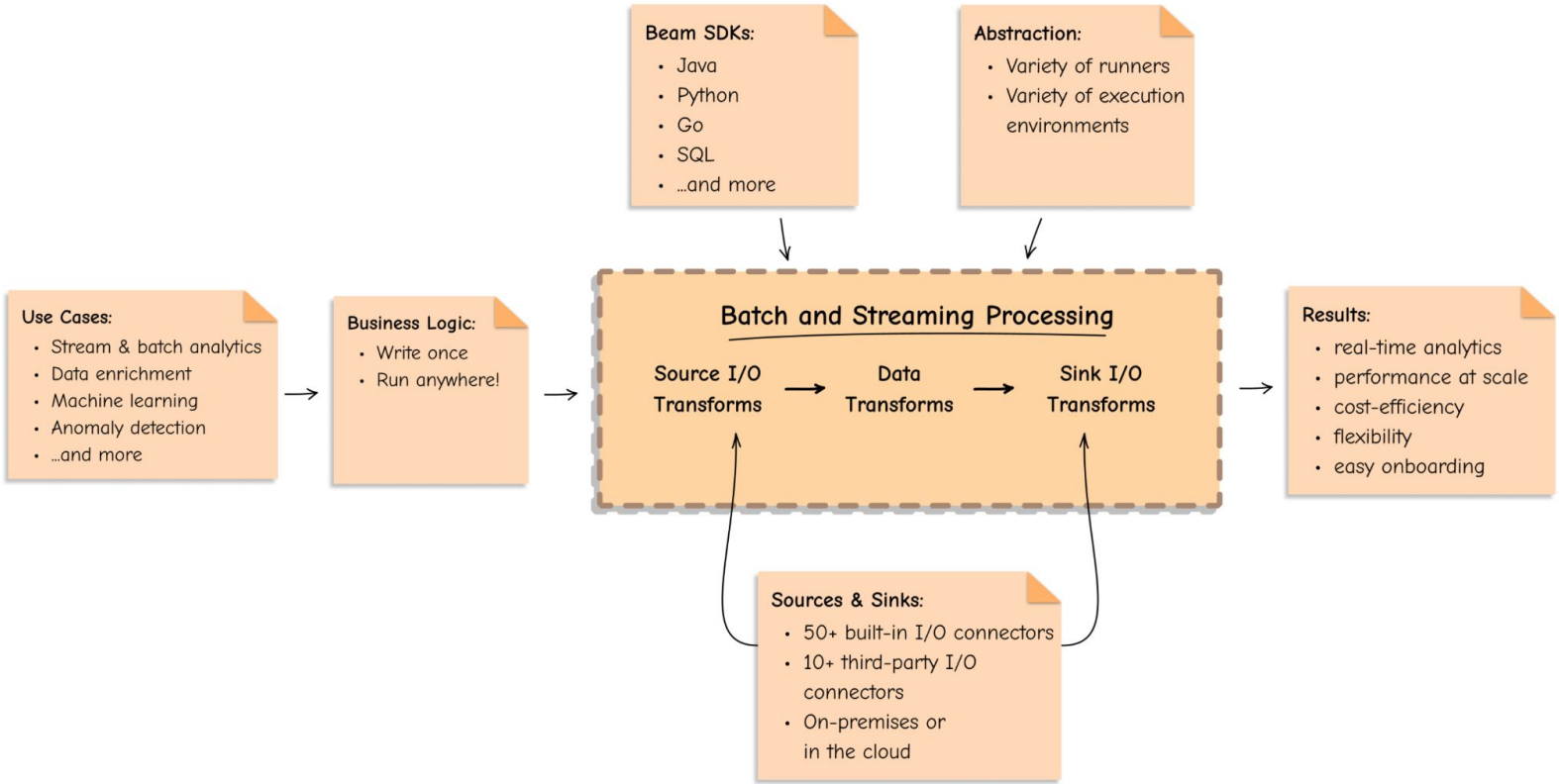


The processing time happens after the event time.



The tasks are unbounded and out-of-order.

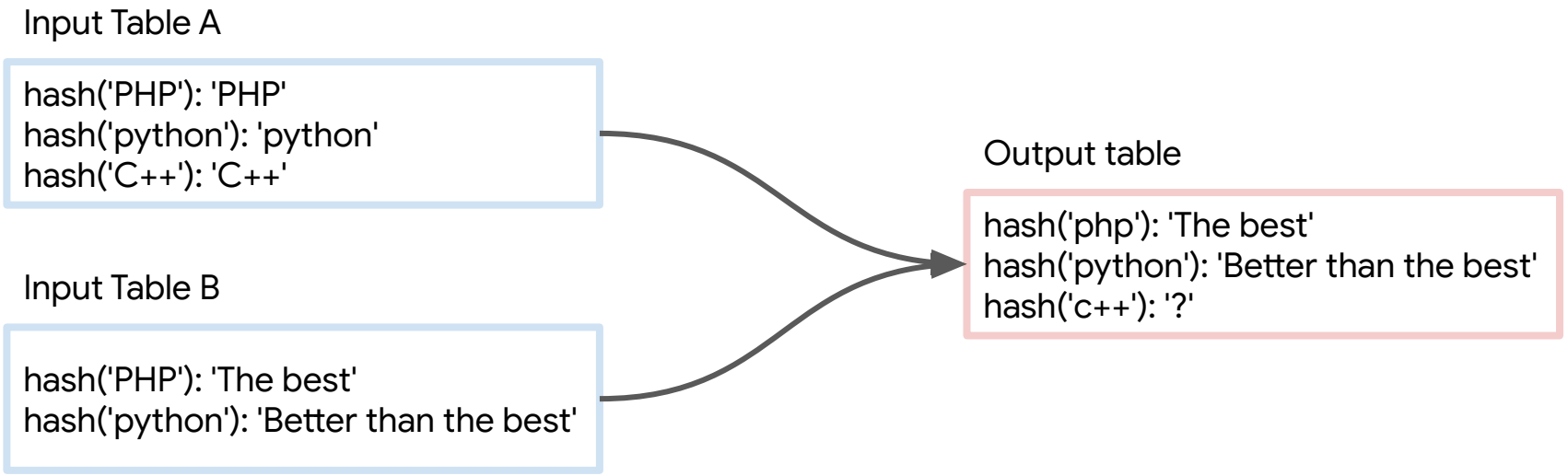
beam


Beam SDKs:
• Java
• Python
• Go
• SQL
• ...and more
Abstraction:
• Variety of runners
• Variety of execution environments
Use Cases:
• Stream & batch analytics
• Data enrichment
• Machine learning
• Anomaly detection
• ...and more
Business Logic:
• Write once
• Run anywhere!
Batch and Streaming Processing
Source I/O Transforms
Data Transforms
Sink I/O Transforms
Results:
• real-time analytics
• performance at scale
• cost-efficiency
• flexibility
• easy onboarding
Sources & Sinks:
• 50+ built-in I/O connectors
• 10+ third-party I/O connectors
• On-premises or in the cloud

Given two tables:
- Table A: **hash key (string)** to **case sensitive index (string)**
- Table B: **hash key (string)** to data in whatever type

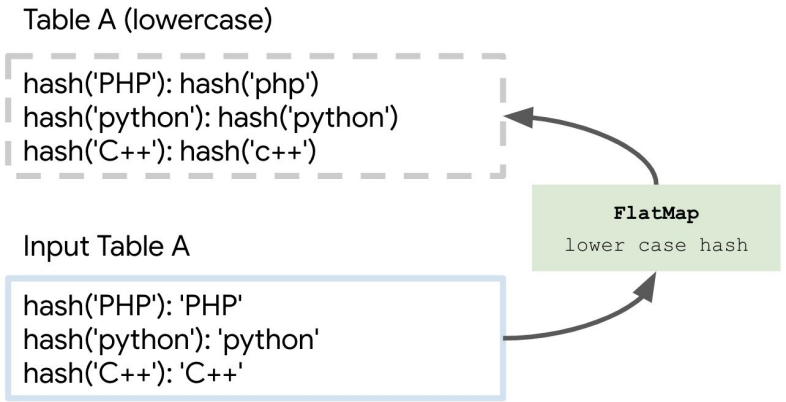
Both tables **hash key** are based on the **case sensitive index**

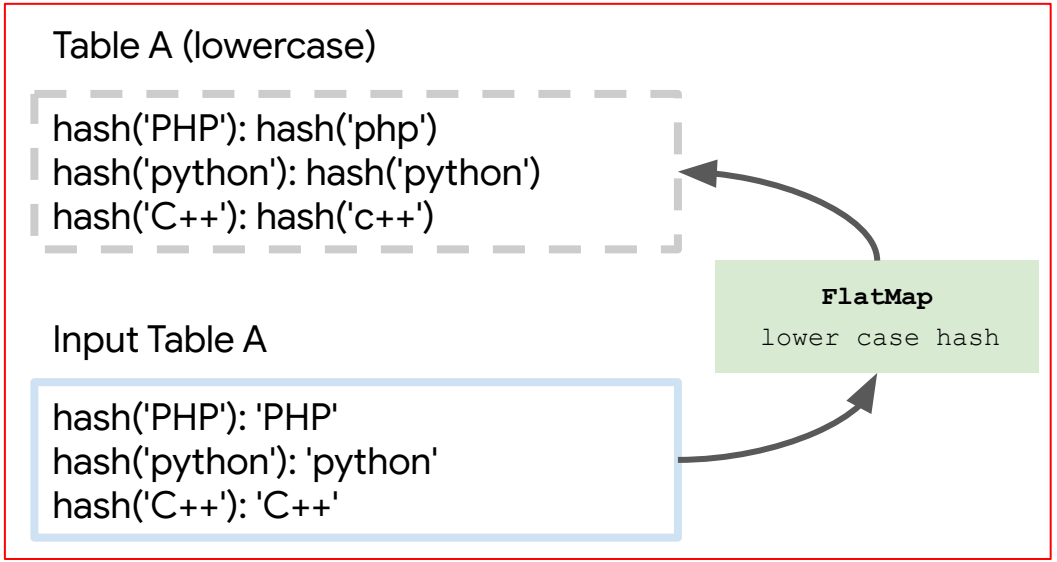
Generate a new table that maps **new hash key** based on **case insensitive index** to data in Table B.



```
def main():
  p = beam.Pipeline() # Run locally with the direct runner.

  new_hash_collection: list[tuple[int, int]] = LANGUAGE_INDEX_COLLECTION
    | "Map existing hash to new hash" >> beam.Map(
      lambda i: (i[0], magic_hash(i[1].lower())))

  new_language_content = (
    ({
      "index": LANGUAGE_INDEX_COLLECTION,
      "content": LANGUAGE_CONTENT_COLLECTION,
      "new_hash": new_hash_collection,
    })
    | "CoGroupByKey" >> beam.CoGroupByKey()
    | "Maybe rehash" >> beam.ParDo(
      lambda hash_to_all: maybe_rehash(hash_to_all))
  )
  p.run()
```

```
def main():
 p = beam.Pipeline() # Run locally with the direct runner.

 new_hash_collection: list[tuple[int, int]] = LANGUAGE_INDEX_COLLECTION
  | "Map existing hash to new hash" >> beam.Map(
   lambda i: (i[0], magic_hash(i[1].lower())))

 new_language_content = (
  ({
   "index": LANGUAGE_INDEX_COLLECTION,
   "content": LANGUAGE_CONTENT_COLLECTION,
   "new_hash": new_hash_collection,
  })
  | "CoGroupByKey" >> beam.CoGroupByKey()
  | "Maybe rehash" >> beam.ParDo(
   lambda hash_to_all: maybe_rehash(hash_to_all))
 )
 p.run()
```





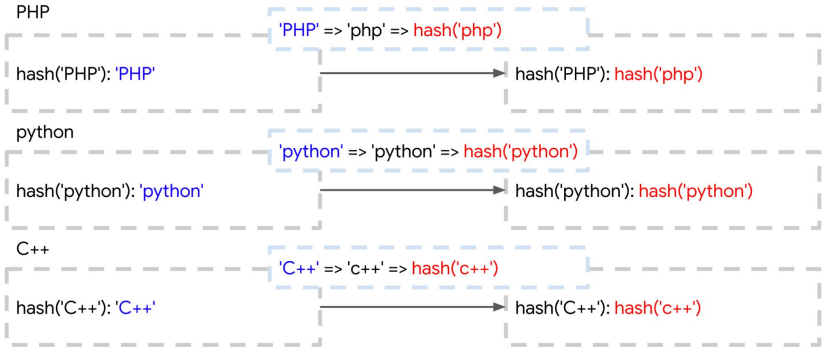
Input Table B

hash('PHP'): 'The best'
hash('python'): 'Better than the best'

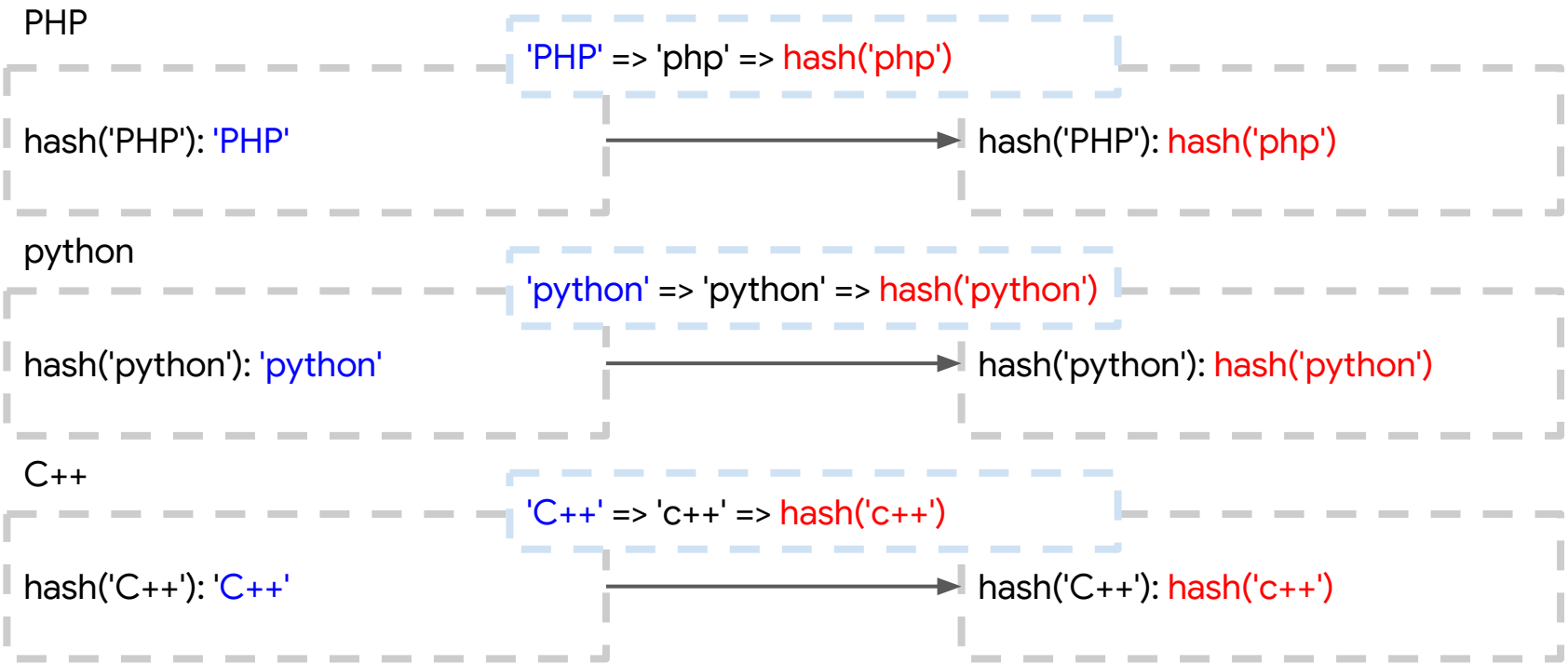
Output table

hash('php'): 'The best'
hash('python'): 'Better than the best'
hash('c++'): '?'

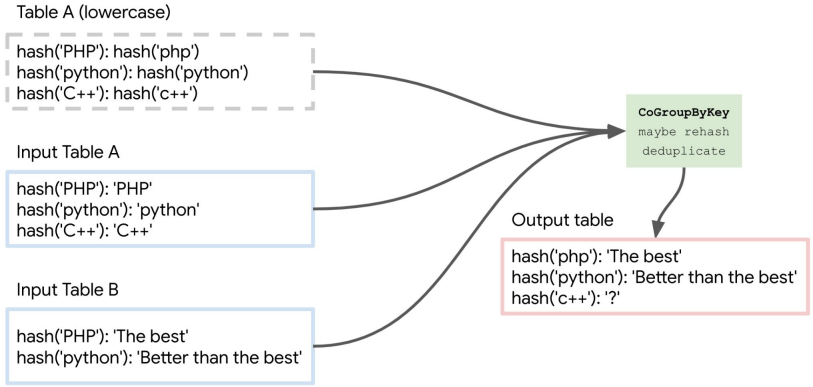
```
def main():
  p = beam.Pipeline() # Run locally with the direct runner.

  new_hash_collection: list[tuple[int, int]] = LANGUAGE_INDEX_COLLECTION
    | "Map existing hash to new hash" >> beam.Map(
    lambda i: (i[0], magic_hash(i[1].lower())))

  new_language_content = (
    ({
      "index": LANGUAGE_INDEX_COLLECTION,
      "content": LANGUAGE_CONTENT_COLLECTION,
      "new_hash": new_hash_collection,
    })
    | "CoGroupByKey" >> beam.CoGroupByKey()
    | "Maybe rehash" >> beam.ParDo(
      lambda hash_to_all: maybe_rehash(hash_to_all))
  )
  p.run()
```



Data is processed **parallelly in different workers.*


PHP
'PHP' => 'php' => hash('php')
hash('PHP'): 'PHP'
hash('PHP'): hash('php')
python
'python' => 'python' => hash('python')
hash('python'): 'python'
hash('python'): hash('python')
C++
'C++' => 'c++' => hash('c++')
hash('C++'): 'C++'
hash('C++'): hash('c++')

```
def main():
  p = beam.Pipeline() # Run locally with the direct runner.

  new_hash_collection: list[tuple[int, int]] = LANGUAGE_INDEX_COLLECTION
    | "Map existing hash to new hash" >> beam.Map(
      lambda i: (i[0], magic_hash(i[1].lower())))

  new_language_content = (
    ({
      "index": LANGUAGE_INDEX_COLLECTION,
      "content": LANGUAGE_CONTENT_COLLECTION,
      "new_hash": new_hash_collection,
    })
    | "CoGroupByKey" >> beam.CoGroupByKey()
    | "Maybe rehash" >> beam.ParDo(
      lambda hash_to_all: maybe_rehash(hash_to_all))
  )
  p.run()
```



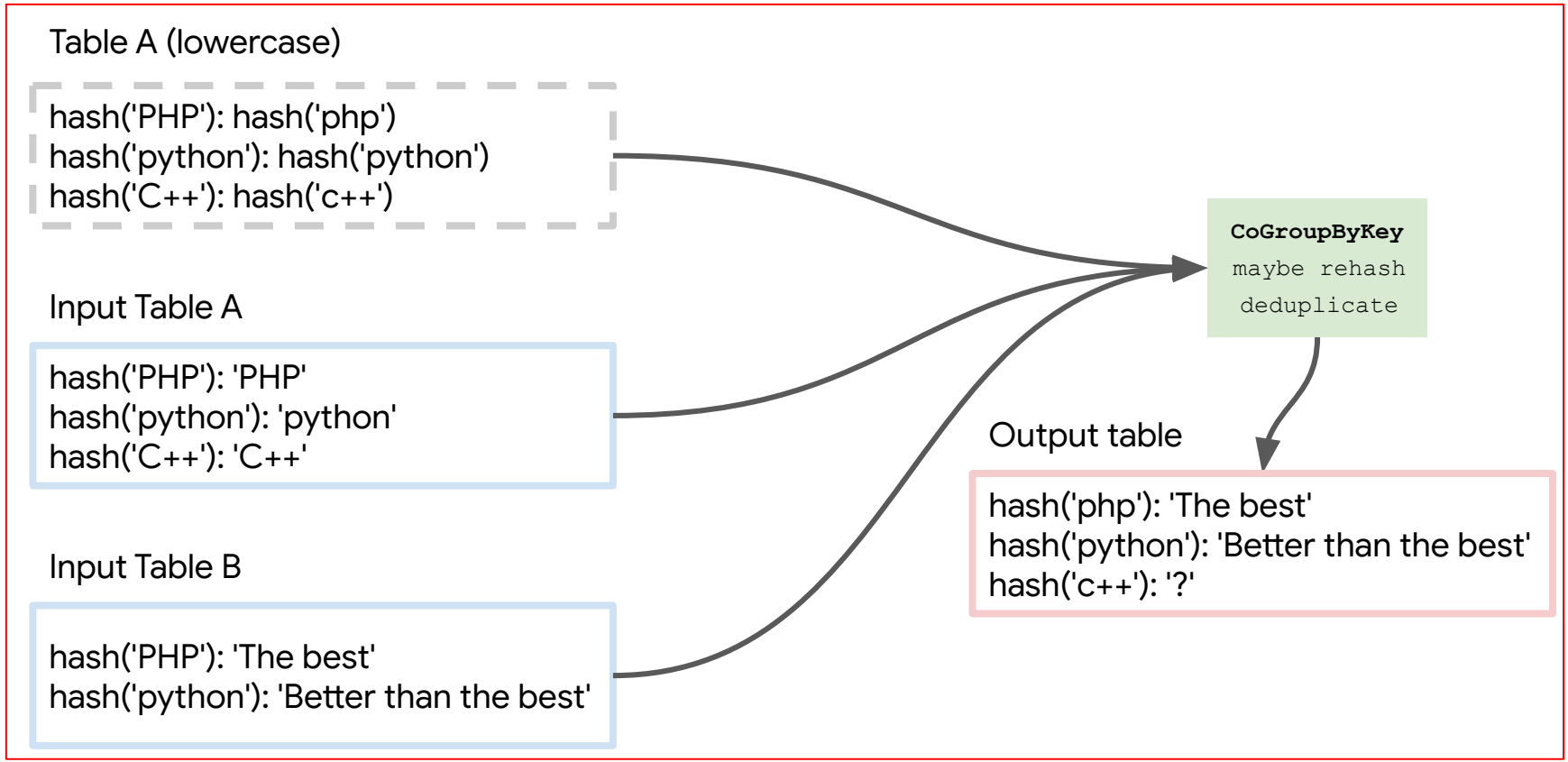


Table A (lowercase)
hash('PHP'): hash('php')
hash('python'): hash('python')
hash('C++'): hash('c++')
Input Table A
hash('PHP'): 'PHP'
hash('python'): 'python'
hash('C++'): 'C++'
Input Table B
hash('PHP'): 'The best'
hash('python'): 'Better than the best'
CoGroupByKey
maybe rehash
deduplicate
Output table
hash('php'): 'The best'
hash('python'): 'Better than the best'
hash('c++'): '?'

```
def main():
  p = beam.Pipeline() # Run locally with the direct runner.

  new_hash_collection: list[tuple[int, int]] = LANGUAGE_INDEX_COLLECTION
    | "Map existing hash to new hash" >> beam.Map(
      lambda i: (i[0], magic_hash(i[1].lower())))

  new_language_content = (
    ({
      "index": LANGUAGE_INDEX_COLLECTION,
      "content": LANGUAGE_CONTENT_COLLECTION,
      "new_hash": new_hash_collection,
    })
    | "CoGroupByKey" >> beam.CoGroupByKey()
    | "Maybe rehash" >> beam.ParDo(
      lambda hash_to_all: maybe_rehash(hash_to_all))
  )
  p.run()
```

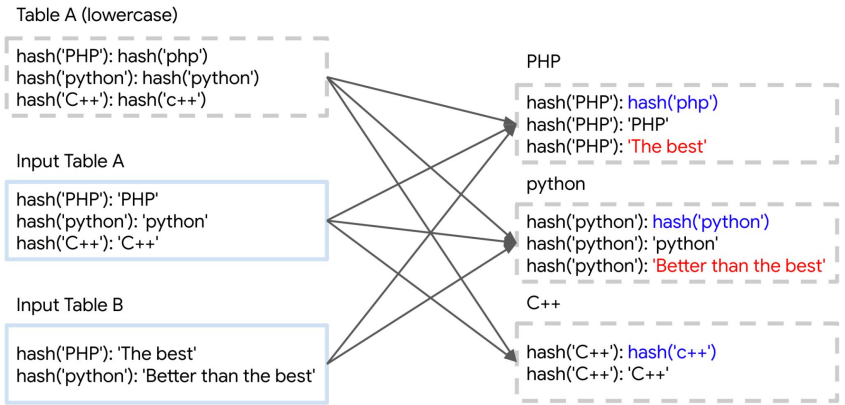


Table A (lowercase)

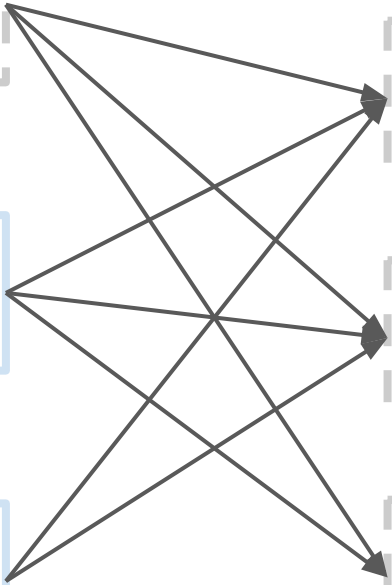
hash('PHP'): hash('php')
hash('python'): hash('python')
hash('C++'): hash('c++')

Input Table A

hash('PHP'): 'PHP'
hash('python'): 'python'
hash('C++'): 'C++'

Input Table B

hash('PHP'): 'The best'
hash('python'): 'Better than the best'



PHP

hash('PHP'): hash('php')
hash('PHP'): 'PHP'
hash('PHP'): 'The best'

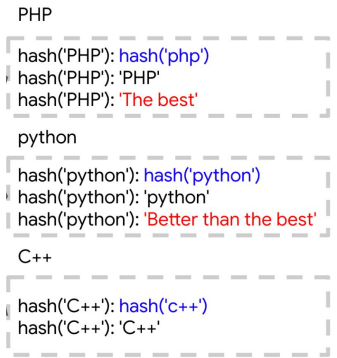
python

hash('python'): hash('python')
hash('python'): 'python'
hash('python'): 'Better than the best'

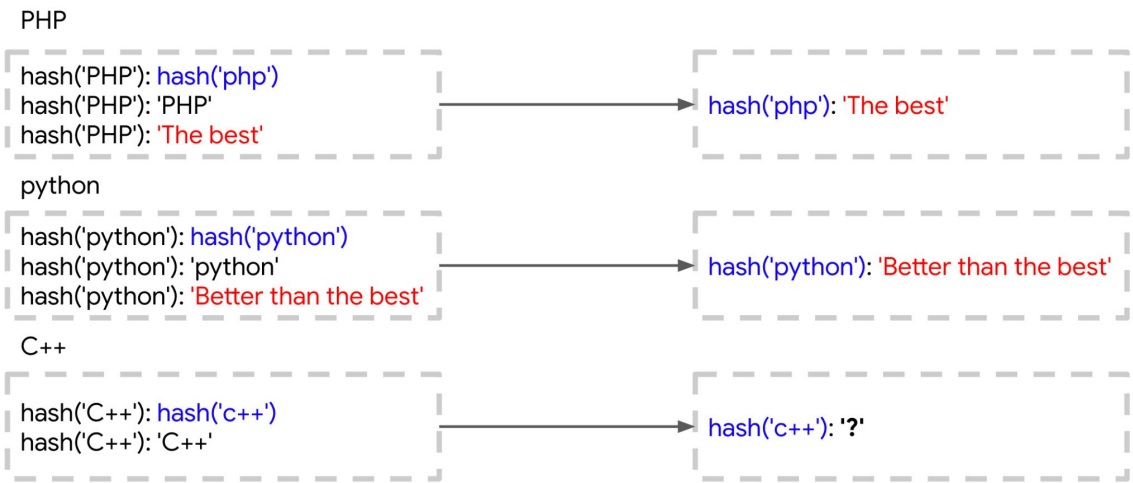
C++

hash('C++'): hash('c++')
hash('C++'): 'C++'

```
def main():
  p = beam.Pipeline() # Run locally with the direct runner.

  new_hash_collection: list[tuple[int, int]] = LANGUAGE_INDEX_COLLECTION
   | "Map existing hash to new hash" >> beam.Map(
    lambda i: (i[0], magic_hash(i[1].lower())))

  new_language_content = (
   ({
    "index": LANGUAGE_INDEX_COLLECTION,
    "content": LANGUAGE_CONTENT_COLLECTION,
    "new_hash": new_hash_collection,
   })
   | "CoGroupByKey" >> beam.CoGroupByKey()
   | "Maybe rehash" >> beam.ParDo(
    lambda hash_to_all: maybe_rehash(hash_to_all))
  )
  p.run()
```



```
def maybe_rehash(hash_to_all):
  for k in hash_to_all[1]["new_hash"]:
    if hash_to_all[1]["content"]:   // dedup
      yield (k, hash_to_all[1]["content"][0])
    else:    // dedup + create
      yield (k,
NEW_CONTENT_PREFIX.format(hash_to_all[1]['index'][0].lower()))
```



Data is processed **parallelly in different workers.*

PHP

hash('PHP'): hash('php')
hash('PHP'): 'PHP'
hash('PHP'): 'The best'

hash('php'): 'The best'

python

hash('python'): hash('python')
hash('python'): 'python'
hash('python'): 'Better than the best'

hash('python'): 'Better than the best'

C++

hash('C++'): hash('c++')
hash('C++'): 'C++'

hash('c++'): **'?'**

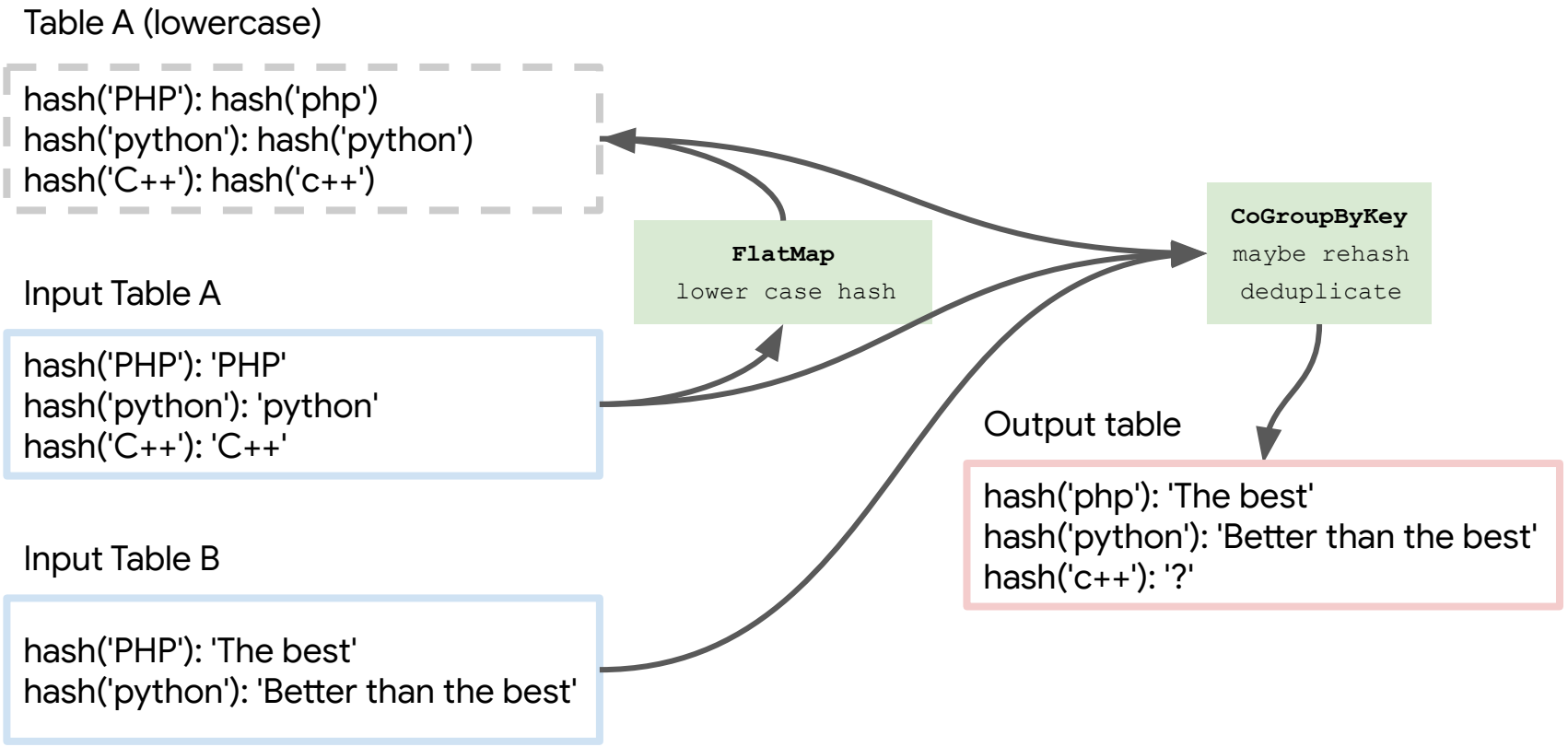


Table A (lowercase)
hash('PHP'): hash('php')
hash('python'): hash('python')
hash('C++'): hash('c++')
Input Table A
hash('PHP'): 'PHP'
hash('python'): 'python'
hash('C++'): 'C++'
Input Table B
hash('PHP'): 'The best'
hash('python'): 'Better than the best'
FlatMap
lower case hash
CoGroupByKey
maybe rehash
deduplicate
Output table
hash('php'): 'The best'
hash('python'): 'Better than the best'
hash('c++'): '?'

Researches

- FlumeJava 2010: Bounded
  - One pipeline to handle many MapReduce jobs
- Millwheel 2013: Unbounded
  - Fault-tolerant stream processing systems
- Dataflow Model 2015: Bounded + Unbounded
  - A flexible abstraction for modern data processing problems
  - Apache Beam is based on this

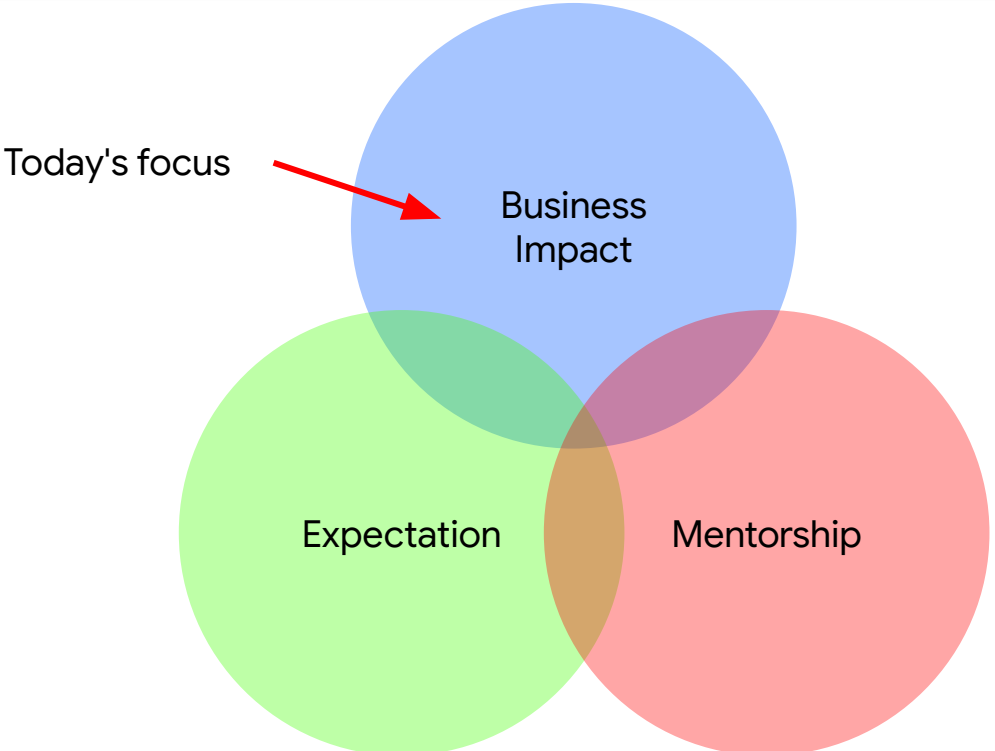
More examples

- Bounded
  - Text analysis
- Unbounded
  - Online game real time scoring system
  - Grocery store's barcode system [PyCon APAC 2018]

A Successful (Intern) Project

note35.github.io/talks

What a Great Software Engineer Intern Host Looks Like



The lifecycle of a project

ideation

Analysis

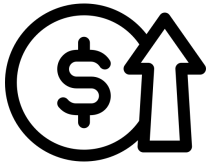
Prototyping

Productization

Experimentation

Launch

Business value



Intern project recommendations

ideation

rare...

Analysis & Prototyping

High effort for both interns and intern hosts.

High risk with potentially **high business impact**.

Productization

Low effort for intern hosts.

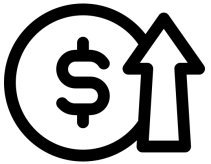
Mostly **expected business impact**.

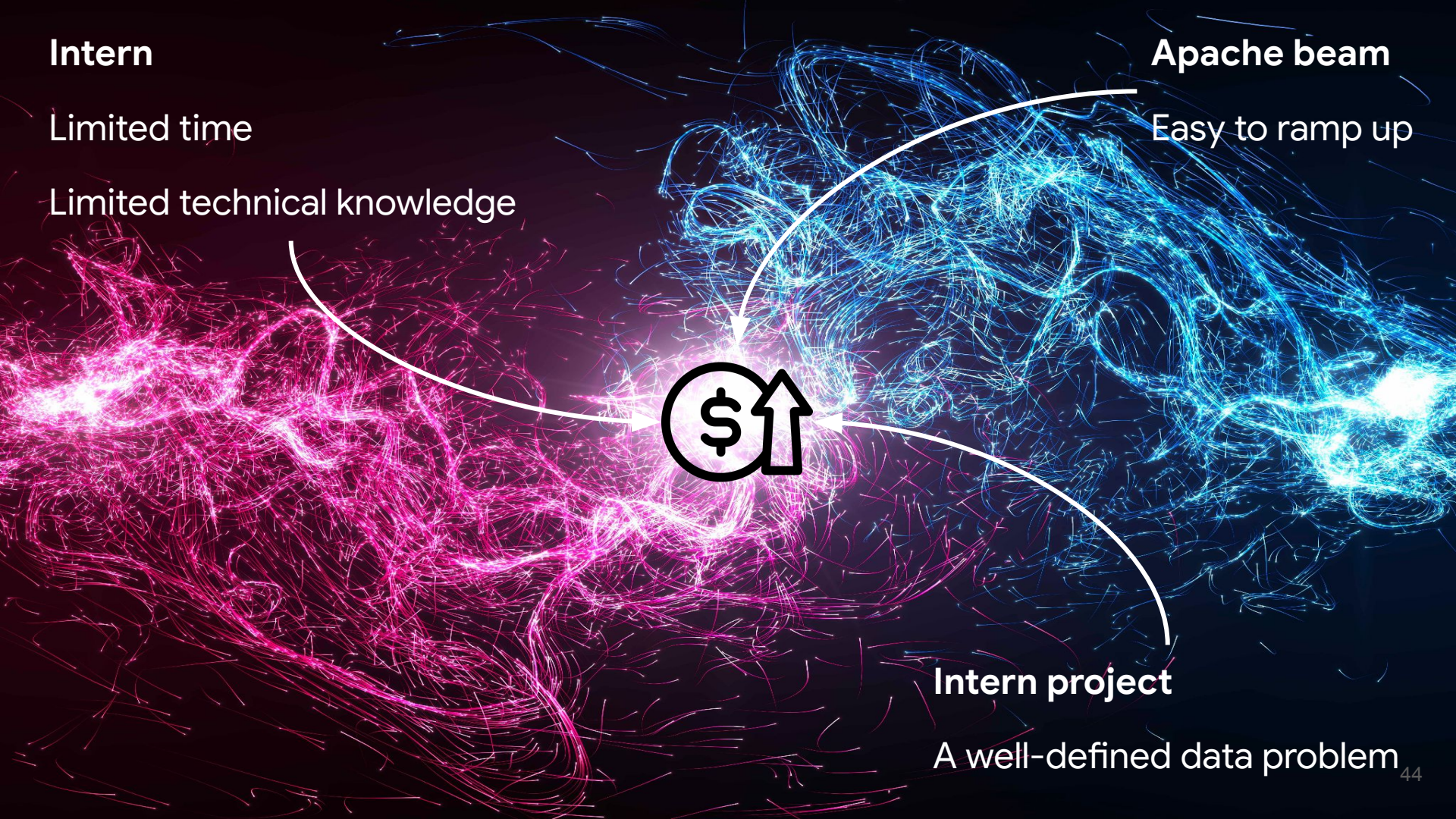
Experimentation & Launch

The works require knowledge and context.

For interns, usually **minor impact**.

Business value




Intern
Limited time
Limited technical knowledge
Apache beam
Easy to ramp up
Intern project
A well-defined data problem

Takeaway

In-memory data processing

Modern data processing

Apach Beam

Apache Beam in the project life cycle

Homework/Promotion

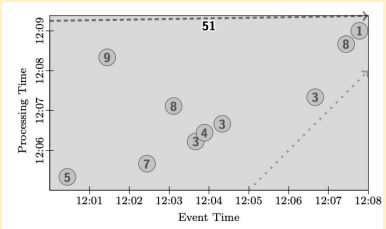
colab

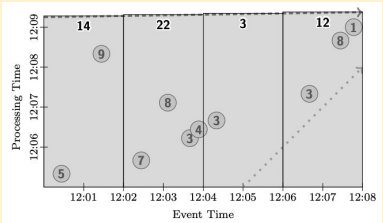
Thank you for listening! 🙏

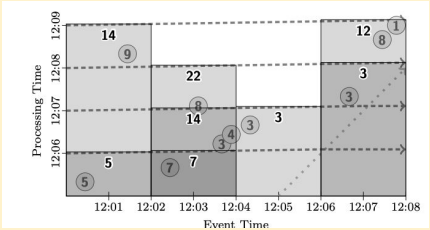
No time to talk slides 

Batch vs Streaming

Simple Batch → Fixed-window Batch → Fixed-window Batchs







Streaming → Partial streaming → Sessions