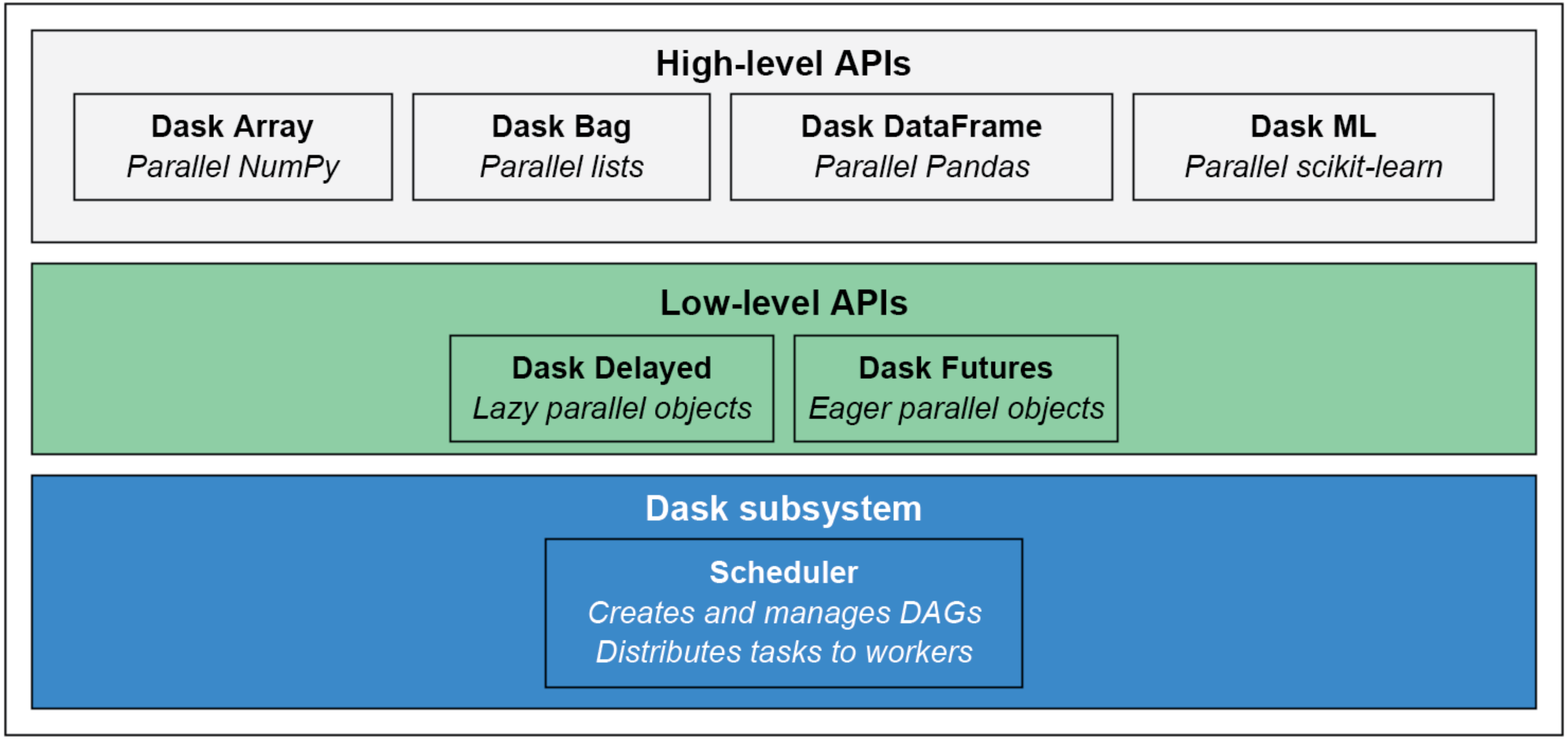
Python & Dask

Parallel & Distributed Computing with Python, Pandas and NumPy

NYU

Dask: https://docs.dask.org/en/stable/



High-level APIs

Dask Array
Parallel NumPy

Dask Bag
Parallel lists

Dask DataFrame
Parallel Pandas

Dask ML
Parallel scikit-learn

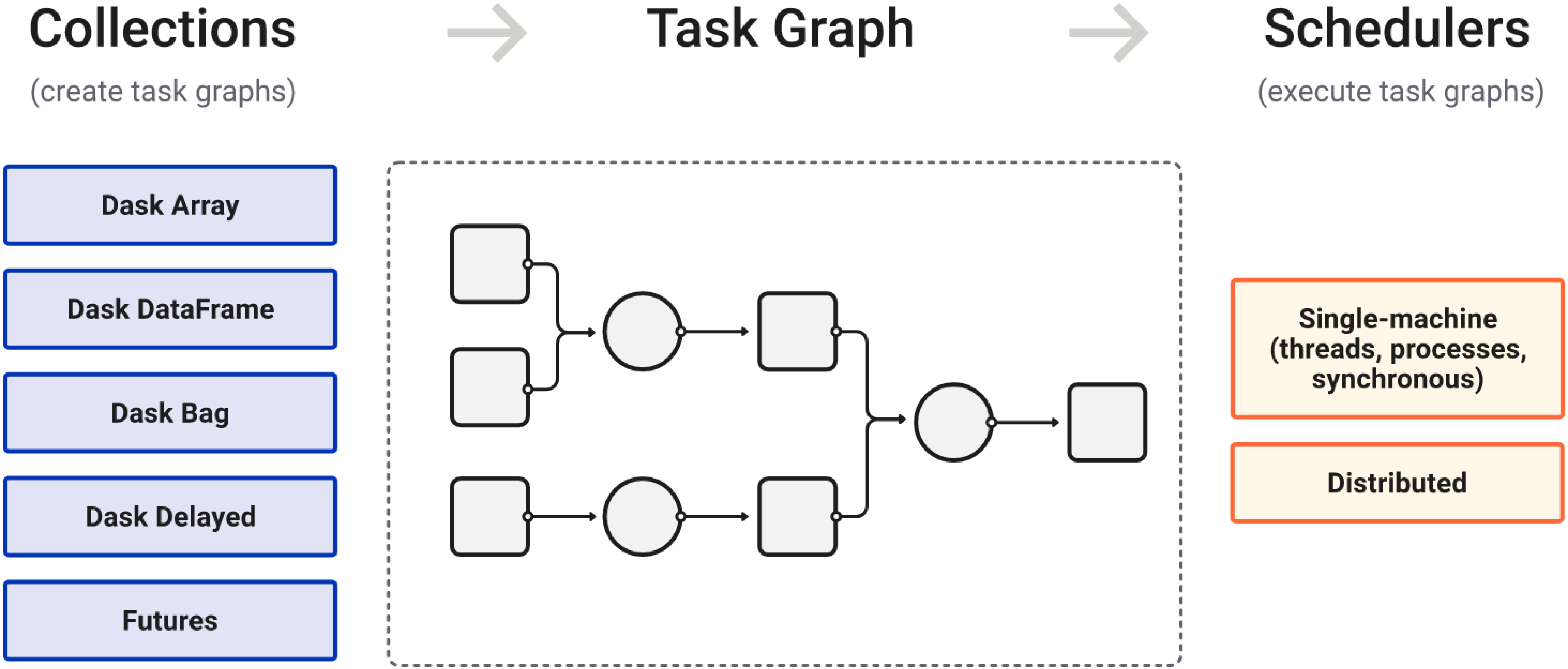
Low-level APIs

Dask Delayed
Lazy parallel objects

Dask Futures
Eager parallel objects

Dask subsystem

Scheduler
Creates and manages DAGs
Distributes tasks to workers



Collections

(create task graphs)

- Dask Array
- Dask DataFrame
- Dask Bag
- Dask Delayed
- Futures

→

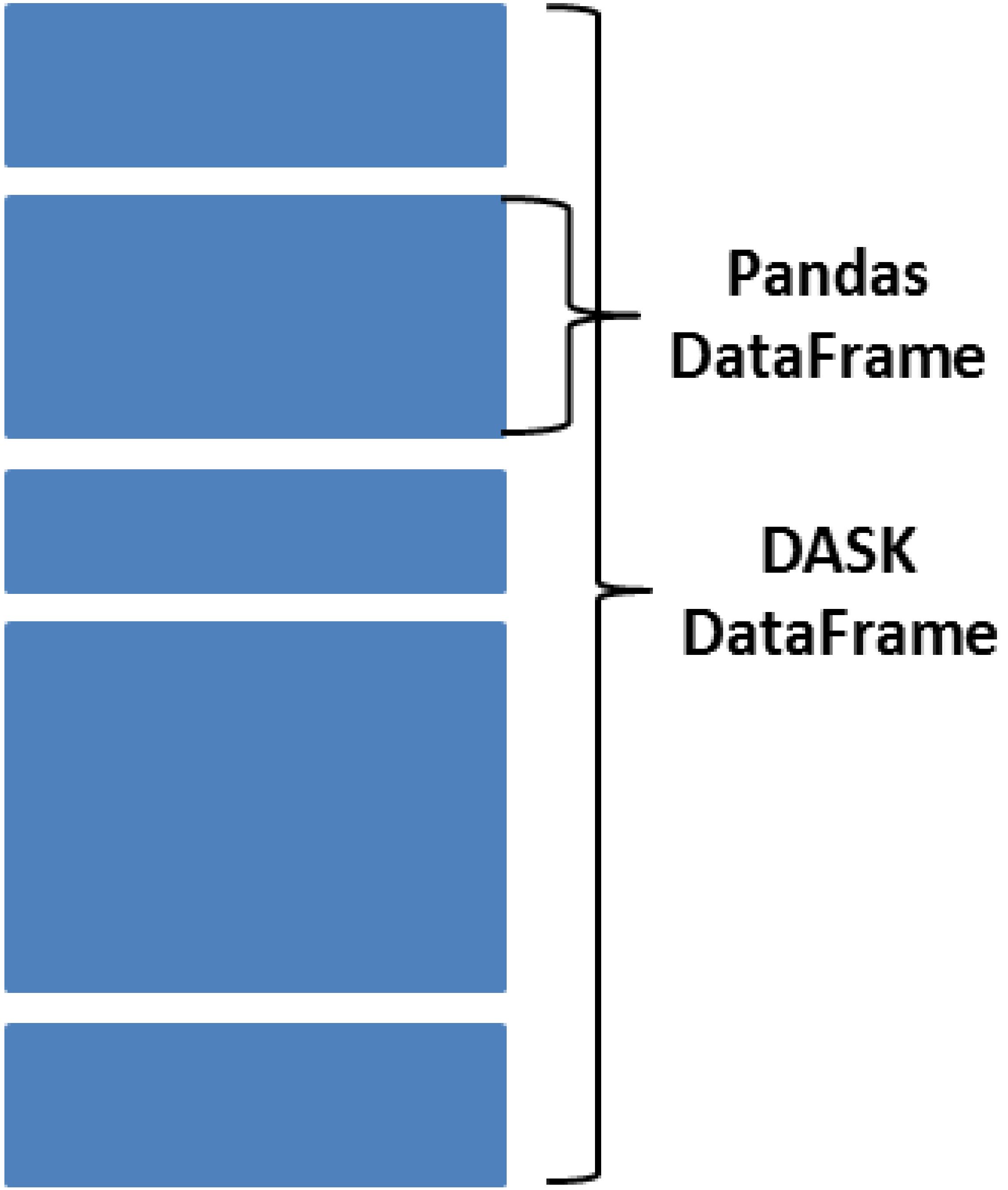
Task Graph

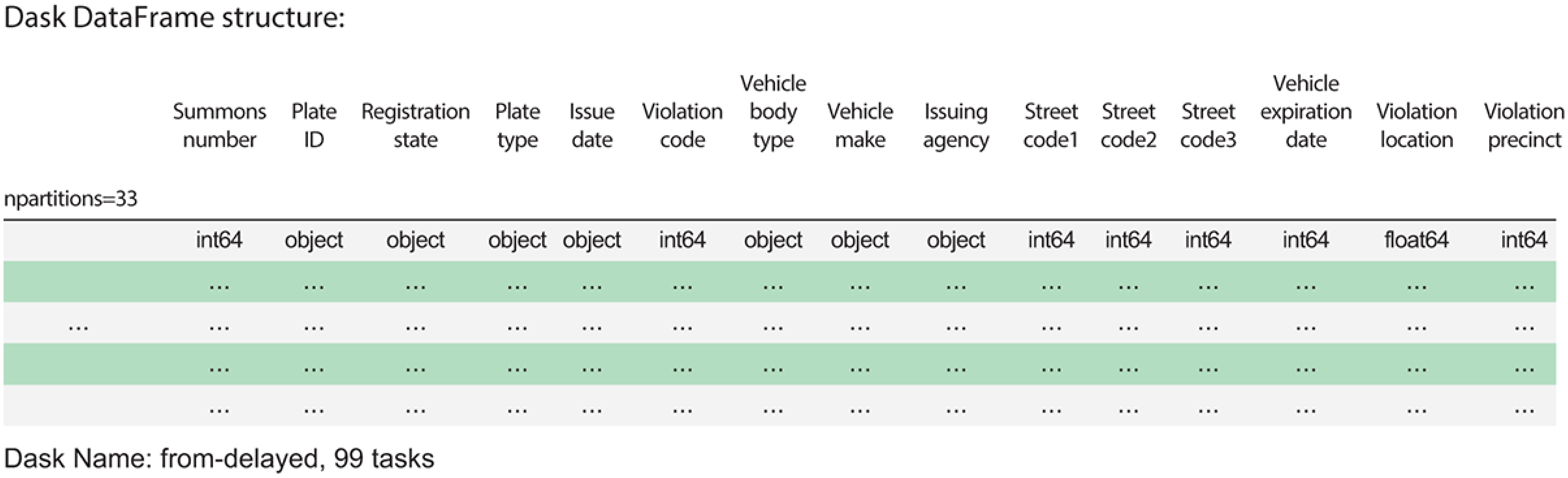
→

Schedulers

(execute task graphs)

- Single-machine (threads, processes, synchronous)
- Distributed


Pandas
DataFrame
DASK
DataFrame

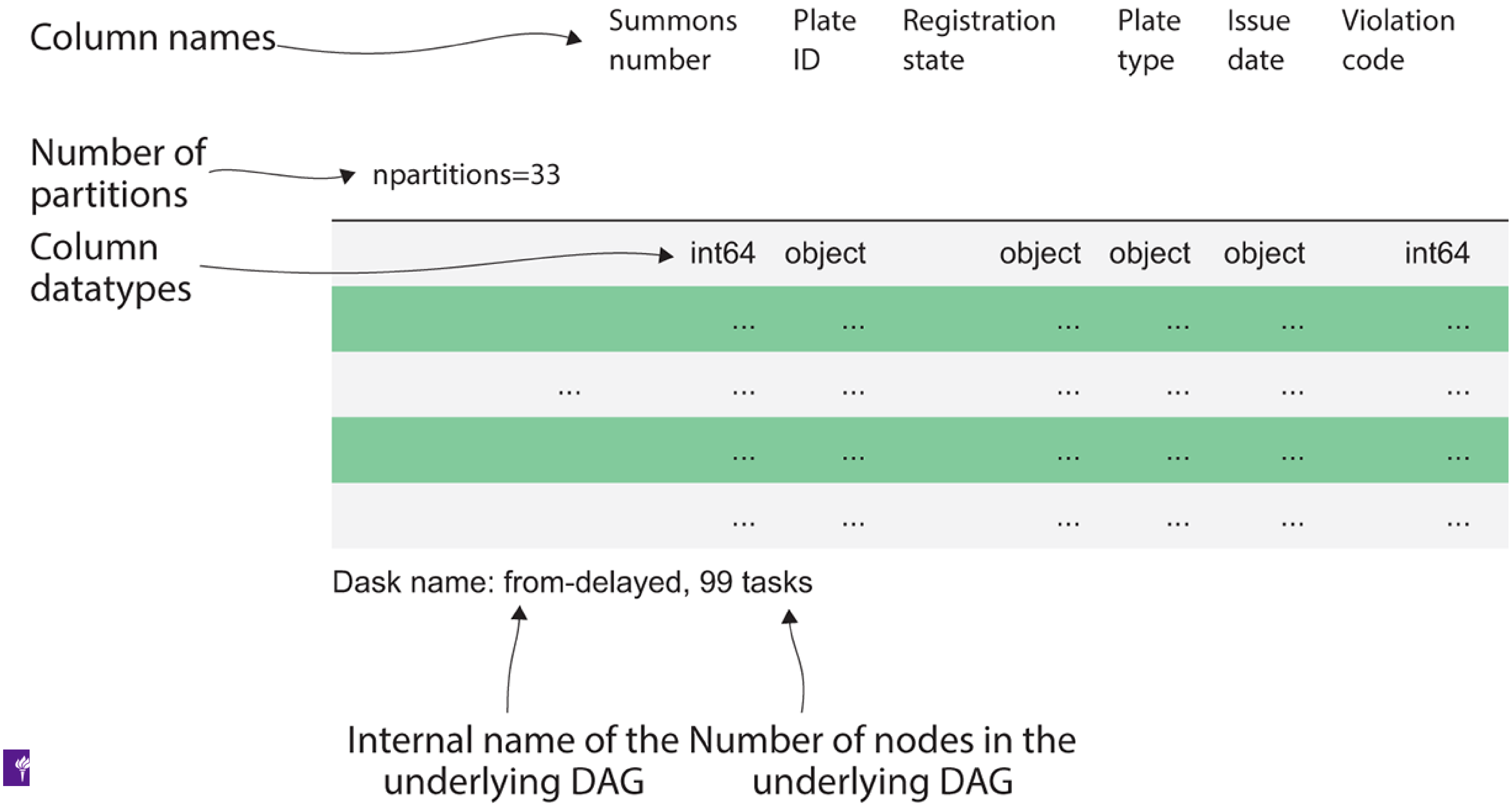


Dask DataFrame structure:

	Summons number	Plate ID	Registration state	Plate type	Issue date	Violation code	Vehicle body type	Vehicle make	Issuing agency	Street code1	Street code2	Street code3	Vehicle expiration date	Violation location	Violation precinct
npartitions=33															
	int64	object	object	object	object	int64	object	object	object	int64	int64	int64	int64	float64	int64
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Dask Name: from-delayed, 99 tasks

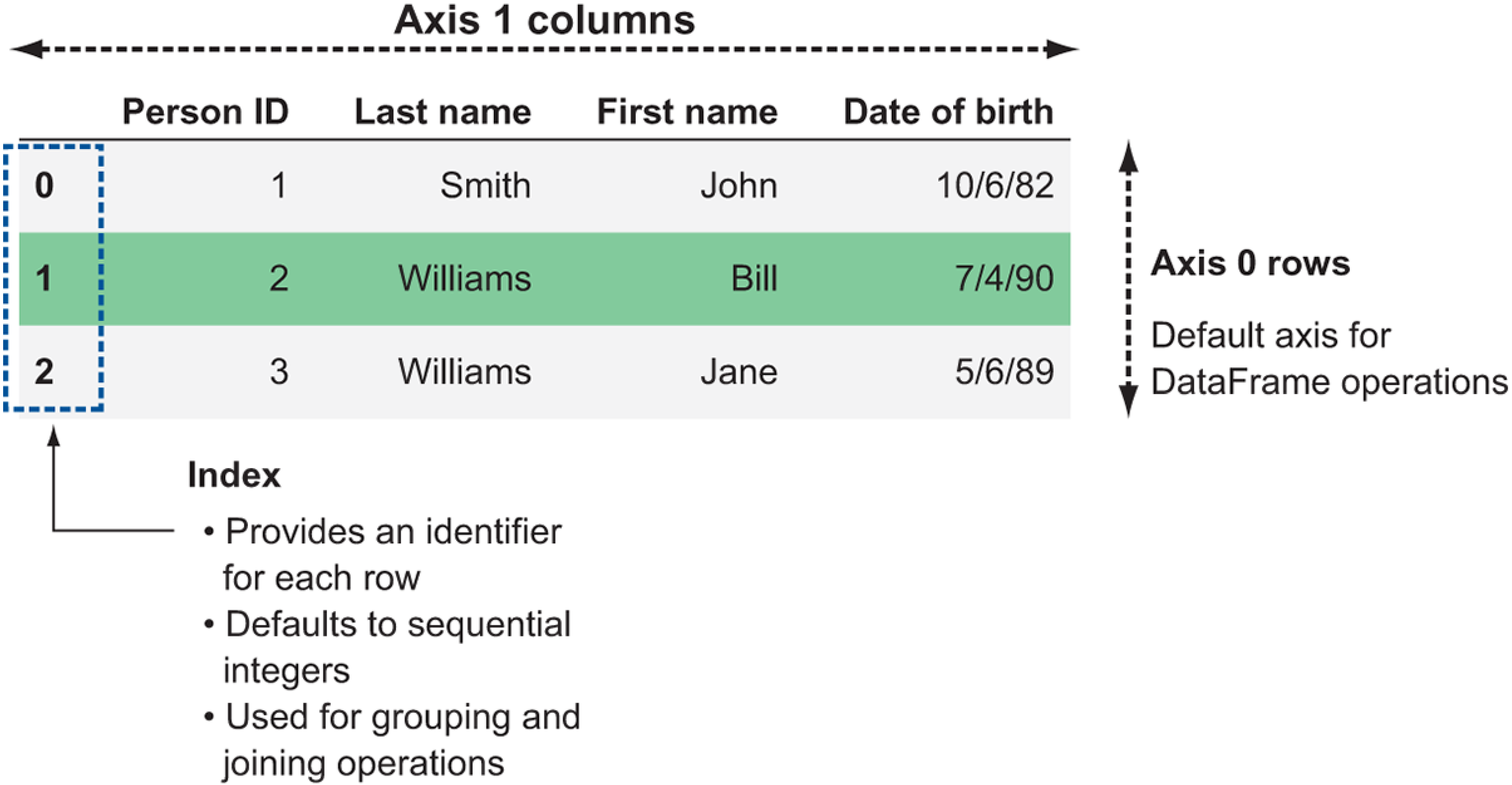
Dask DataFrame structure



npartitions=33	Summons number	Plate ID	Registration state	Plate type	Issue date	Violation code
	int64	object	object	object	object	int64
	...	...	...	...	...	...
...	...	...	...	...	...	...
	...	...	...	...	...	...
	...	...	...	...	...	...

Dask name: from-delayed, 99 tasks

Column names
Number of partitions
Column datatypes
Internal name of the underlying DAG
Number of nodes in the underlying DAG



Axis 1 columns

	Person ID	Last name	First name	Date of birth
0	1	Smith	John	10/6/82
1	2	Williams	Bill	7/4/90
2	3	Williams	Jane	5/6/89

Axis 0 rows

Default axis for DataFrame operations

Index

- Provides an identifier for each row
- Defaults to sequential integers
- Used for grouping and joining operations

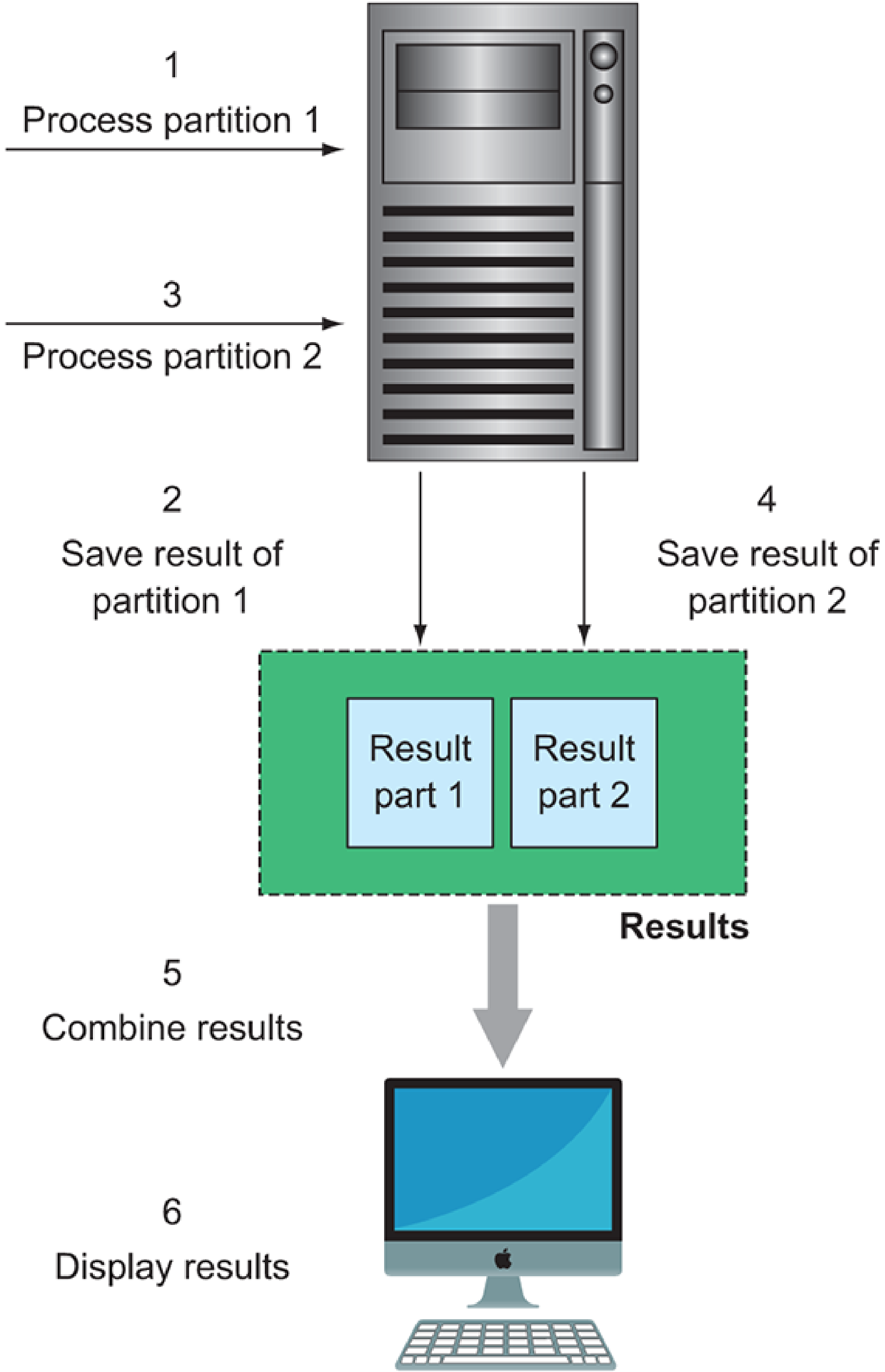
Dask DataFrame

Partition 1

	Person ID	Last name	First name	Date of birth
0	1	Smith	John	10/6/82
1	2	Williams	Bill	7/4/90
2	3	Williams	Jane	5/6/89
3	4	Jackson	Cathy	1/24/74
4	5	Johnson	Stuart	6/5/95

Partition 2

	Person ID	Last name	First name	Date of birth
5	6	Smith	James	4/16/84
6	7	Anderson	Felicity	9/15/76
7	8	Christiansen	Liam	10/2/92
8	9	Carter	Nancy	2/5/86
9	10	Davidson	Christina	8/11/93



1
Process partition 1

3
Process partition 2

2
Save result of partition 1

4
Save result of partition 2

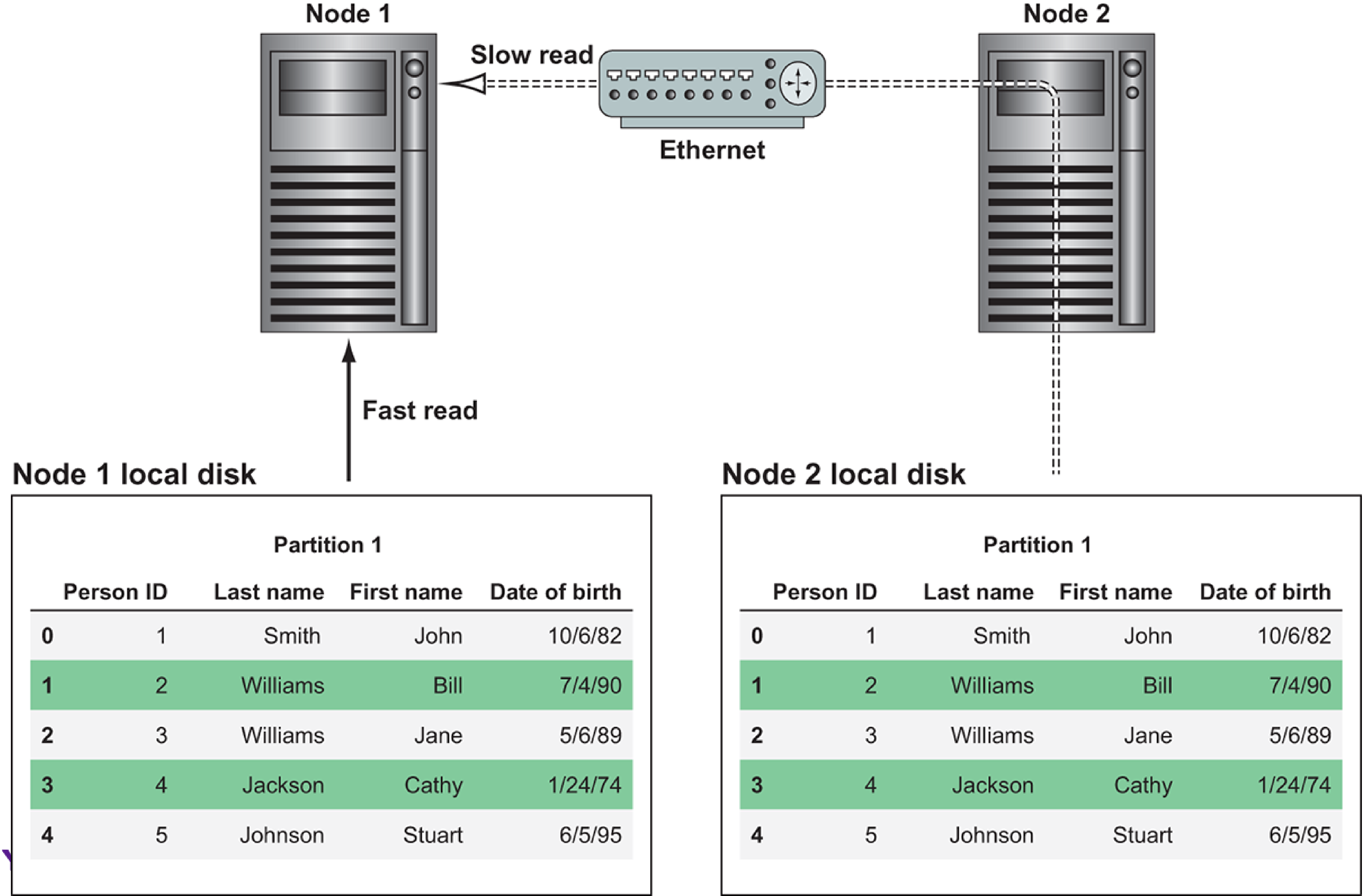
Result part 1

Result part 2

Results

5
Combine results

6
Display results



Node 1 local disk

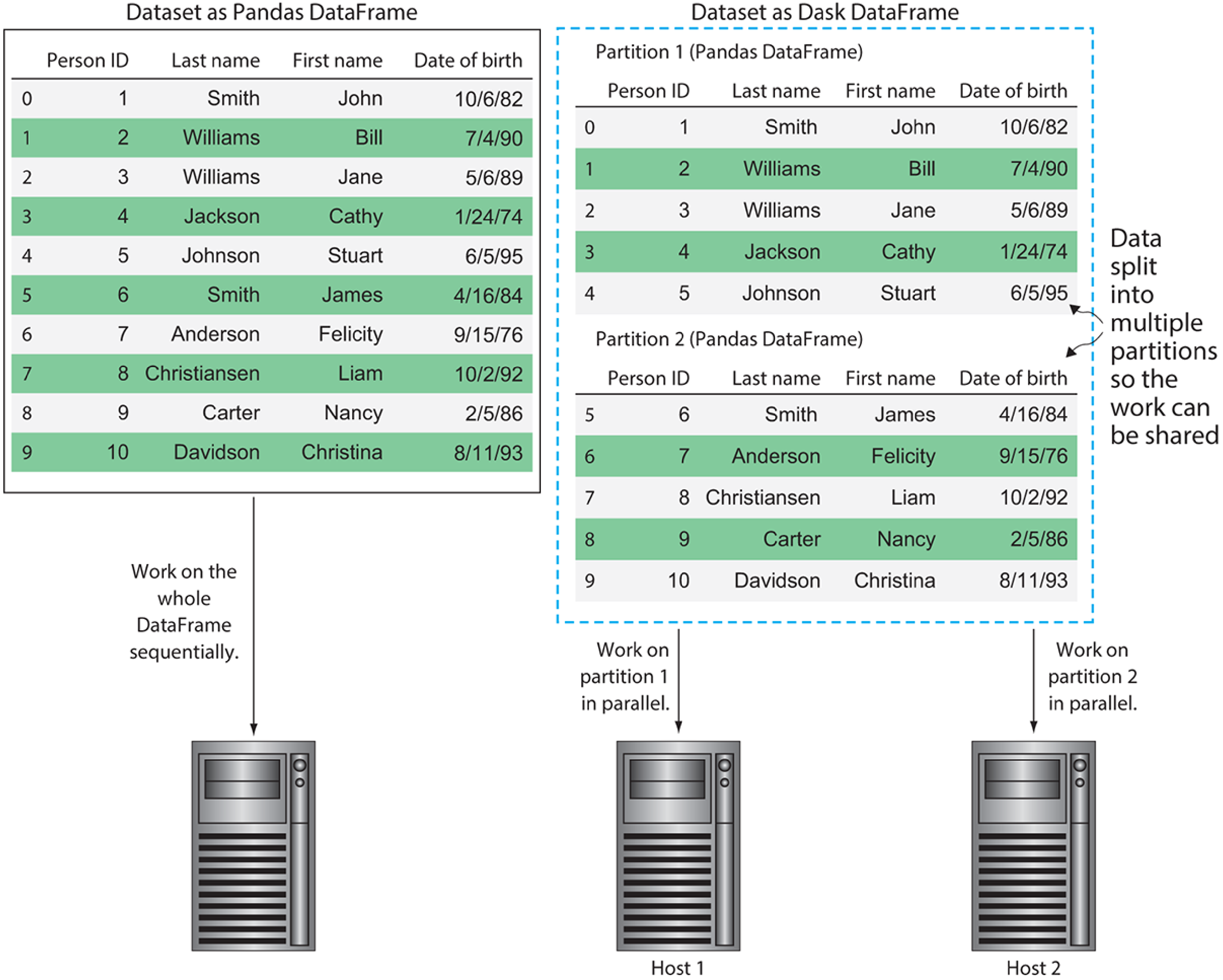
Partition 1

	Person ID	Last name	First name	Date of birth
0	1	Smith	John	10/6/82
1	2	Williams	Bill	7/4/90
2	3	Williams	Jane	5/6/89
3	4	Jackson	Cathy	1/24/74
4	5	Johnson	Stuart	6/5/95

Node 2 local disk

Partition 1

	Person ID	Last name	First name	Date of birth
0	1	Smith	John	10/6/82
1	2	Williams	Bill	7/4/90
2	3	Williams	Jane	5/6/89
3	4	Jackson	Cathy	1/24/74
4	5	Johnson	Stuart	6/5/95



Dataset as Pandas DataFrame

	Person ID	Last name	First name	Date of birth
0	1	Smith	John	10/6/82
1	2	Williams	Bill	7/4/90
2	3	Williams	Jane	5/6/89
3	4	Jackson	Cathy	1/24/74
4	5	Johnson	Stuart	6/5/95
5	6	Smith	James	4/16/84
6	7	Anderson	Felicity	9/15/76
7	8	Christiansen	Liam	10/2/92
8	9	Carter	Nancy	2/5/86
9	10	Davidson	Christina	8/11/93

Work on the whole DataFrame sequentially.

Dataset as Dask DataFrame

Partition 1 (Pandas DataFrame)

	Person ID	Last name	First name	Date of birth
0	1	Smith	John	10/6/82
1	2	Williams	Bill	7/4/90
2	3	Williams	Jane	5/6/89
3	4	Jackson	Cathy	1/24/74
4	5	Johnson	Stuart	6/5/95

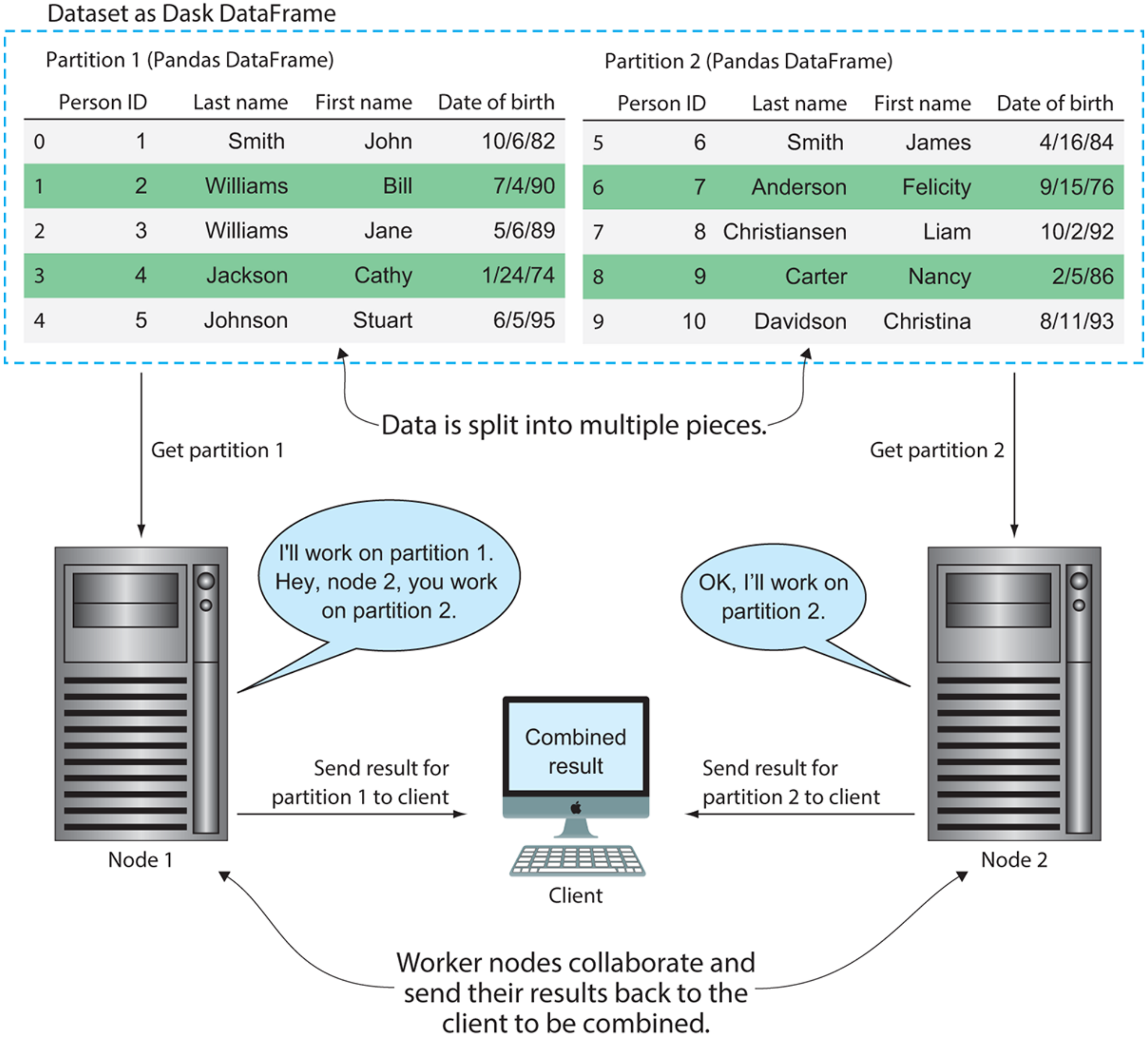
Partition 2 (Pandas DataFrame)

	Person ID	Last name	First name	Date of birth
5	6	Smith	James	4/16/84
6	7	Anderson	Felicity	9/15/76
7	8	Christiansen	Liam	10/2/92
8	9	Carter	Nancy	2/5/86
9	10	Davidson	Christina	8/11/93

Data split into multiple partitions so the work can be shared

Work on partition 1 in parallel.

Work on partition 2 in parallel.



Dataset as Dask DataFrame

Partition 1 (Pandas DataFrame)

	Person ID	Last name	First name	Date of birth
0	1	Smith	John	10/6/82
1	2	Williams	Bill	7/4/90
2	3	Williams	Jane	5/6/89
3	4	Jackson	Cathy	1/24/74
4	5	Johnson	Stuart	6/5/95

Partition 2 (Pandas DataFrame)

	Person ID	Last name	First name	Date of birth
5	6	Smith	James	4/16/84
6	7	Anderson	Felicity	9/15/76
7	8	Christiansen	Liam	10/2/92
8	9	Carter	Nancy	2/5/86
9	10	Davidson	Christina	8/11/93

Data is split into multiple pieces.

Get partition 1

Get partition 2

I'll work on partition 1. Hey, node 2, you work on partition 2.

OK, I'll work on partition 2.

Combined result

Send result for partition 1 to client

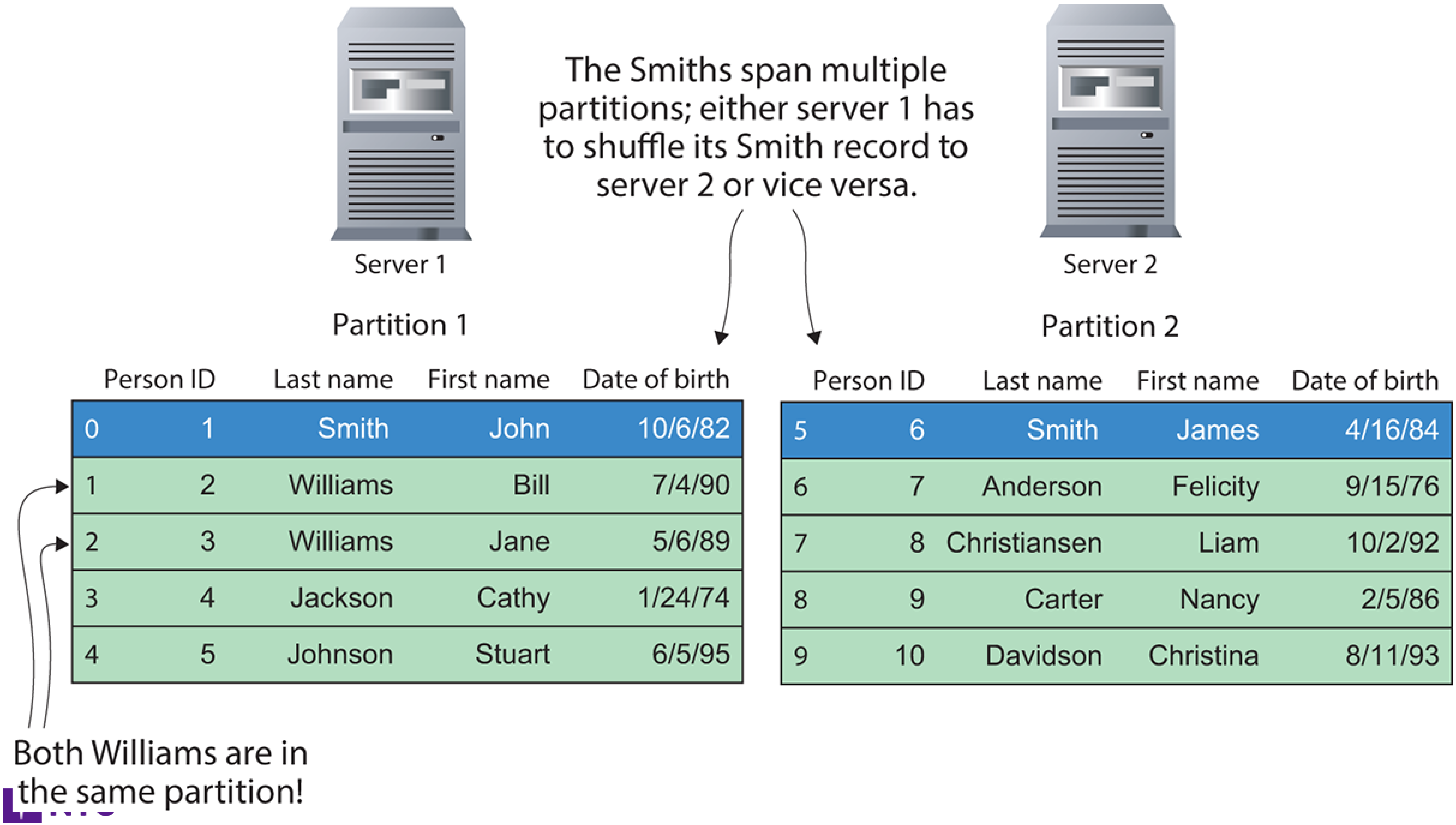
Send result for partition 2 to client

Node 1

Client

Node 2

Worker nodes collaborate and send their results back to the client to be combined.



The Smiths span multiple partitions; either server 1 has to shuffle its Smith record to server 2 or vice versa.

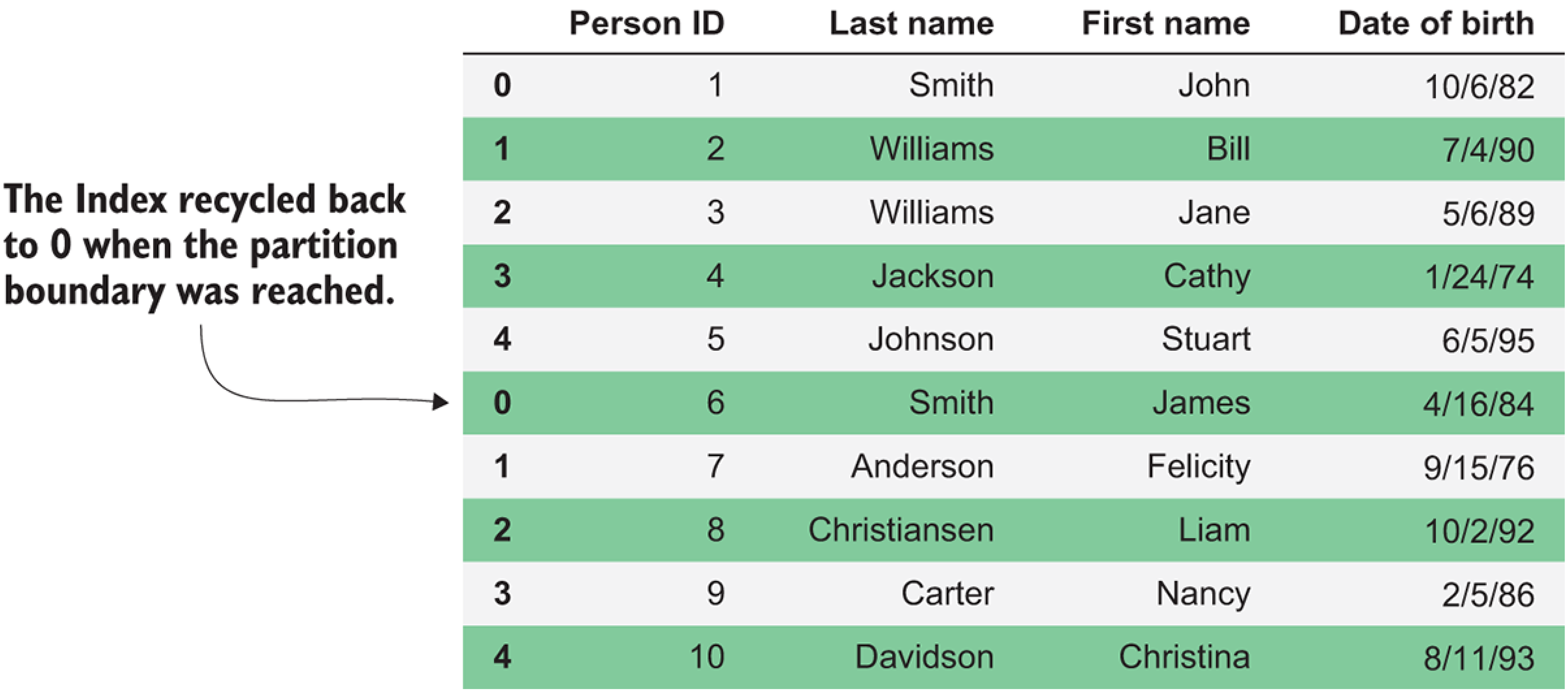
Partition 1

	Person ID	Last name	First name	Date of birth
0	1	Smith	John	10/6/82
1	2	Williams	Bill	7/4/90
2	3	Williams	Jane	5/6/89
3	4	Jackson	Cathy	1/24/74
4	5	Johnson	Stuart	6/5/95

Partition 2

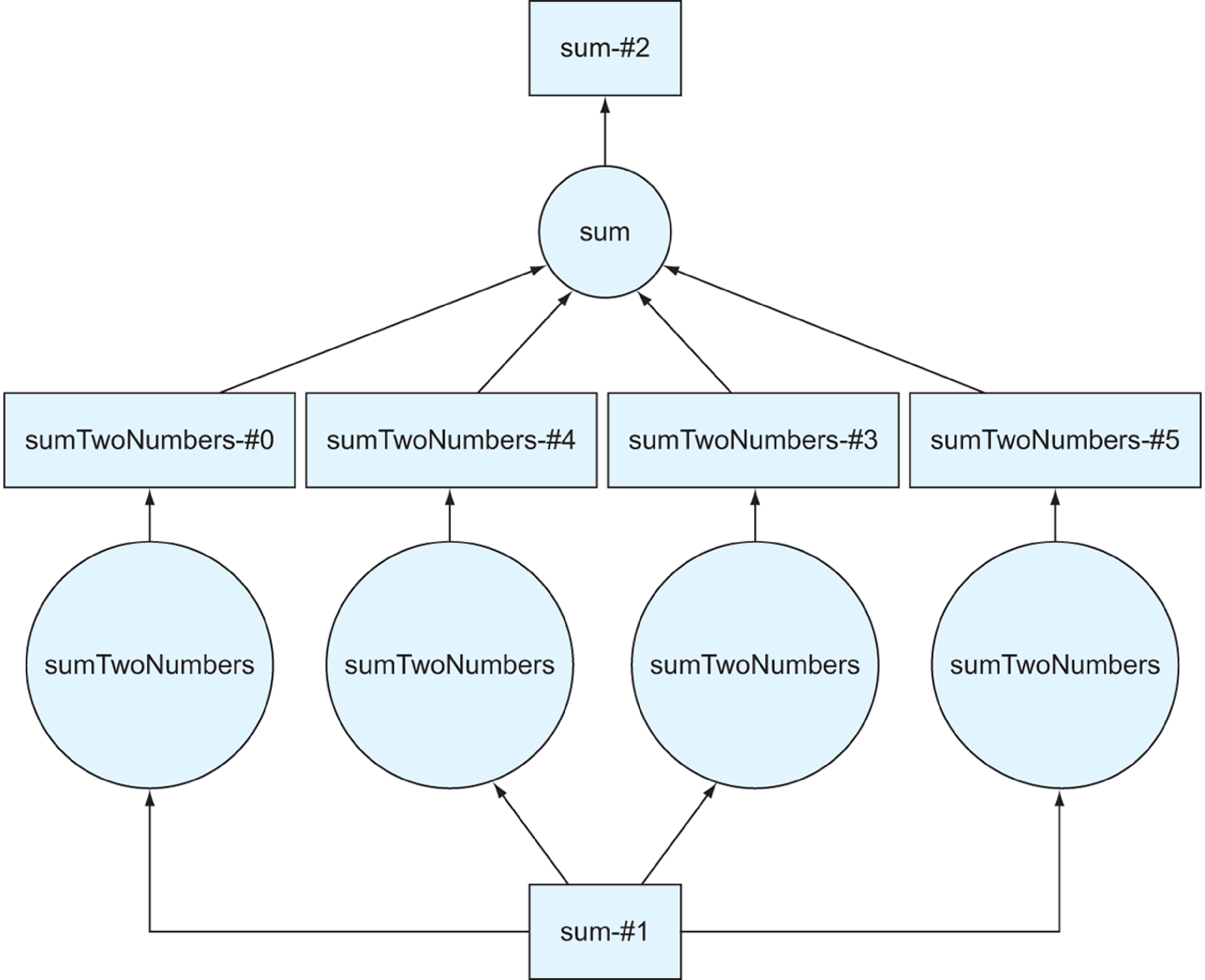
	Person ID	Last name	First name	Date of birth
5	6	Smith	James	4/16/84
6	7	Anderson	Felicity	9/15/76
7	8	Christiansen	Liam	10/2/92
8	9	Carter	Nancy	2/5/86
9	10	Davidson	Christina	8/11/93

Both Williams are in the same partition!



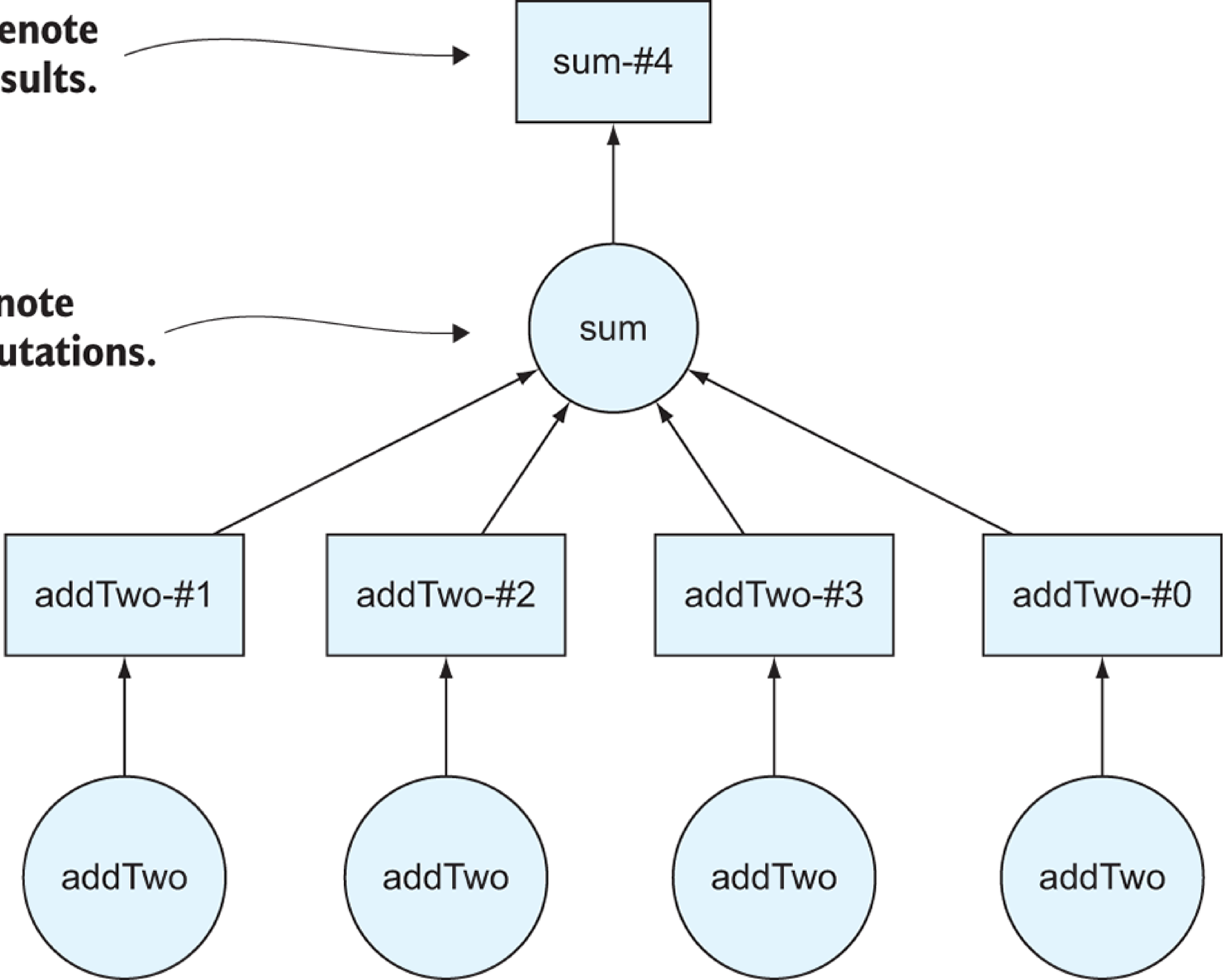
The Index recycled back to 0 when the partition boundary was reached.

	Person ID	Last name	First name	Date of birth
0	1	Smith	John	10/6/82
1	2	Williams	Bill	7/4/90
2	3	Williams	Jane	5/6/89
3	4	Jackson	Cathy	1/24/74
4	5	Johnson	Stuart	6/5/95
0	6	Smith	James	4/16/84
1	7	Anderson	Felicity	9/15/76
2	8	Christiansen	Liam	10/2/92
3	9	Carter	Nancy	2/5/86
4	10	Davidson	Christina	8/11/93

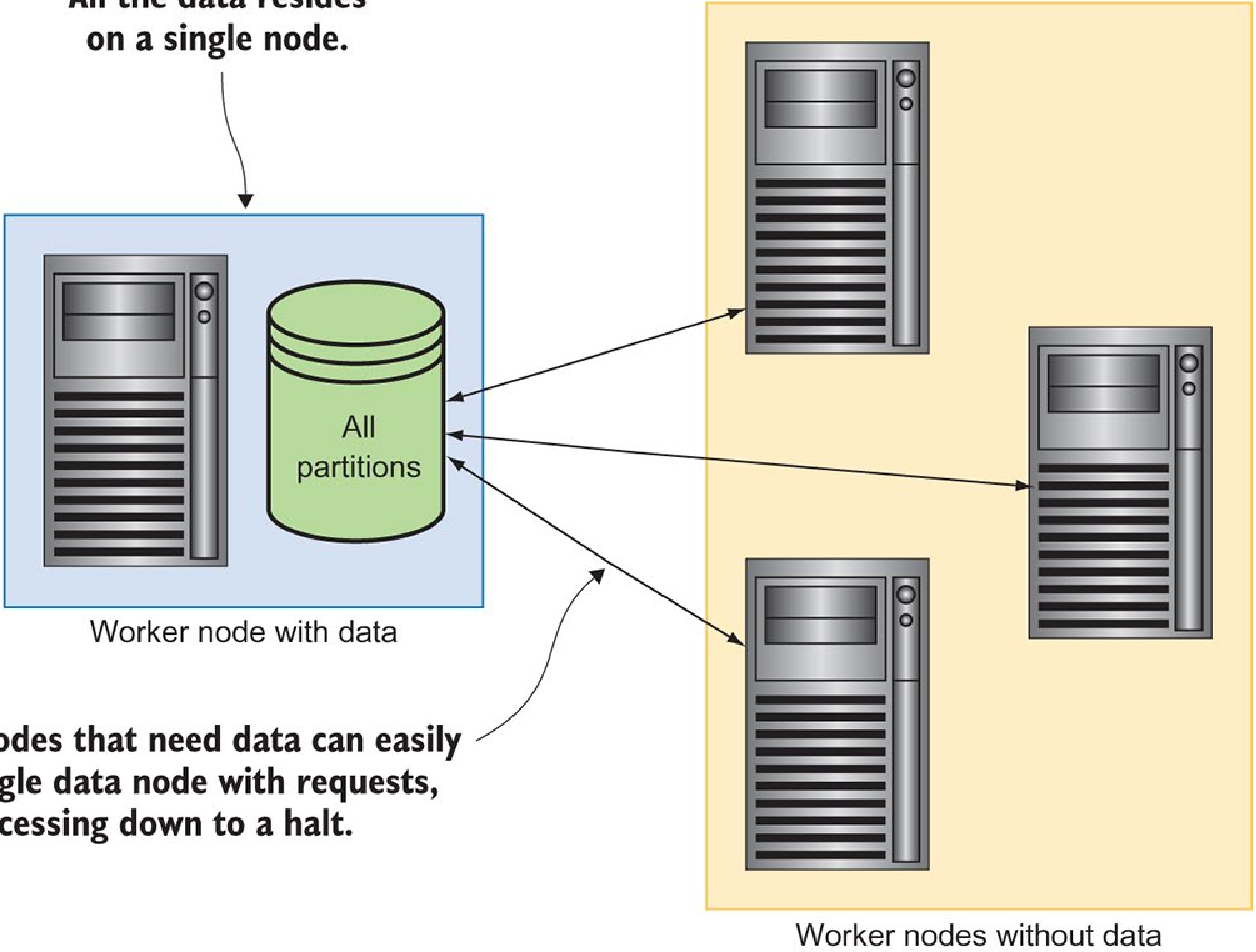

sum-#2
sum
sumTwoNumbers-#0
sumTwoNumbers-#4
sumTwoNumbers-#3
sumTwoNumbers-#5
sumTwoNumbers
sumTwoNumbers
sumTwoNumbers
sumTwoNumbers
sum-#1

Square nodes denote intermediate results.

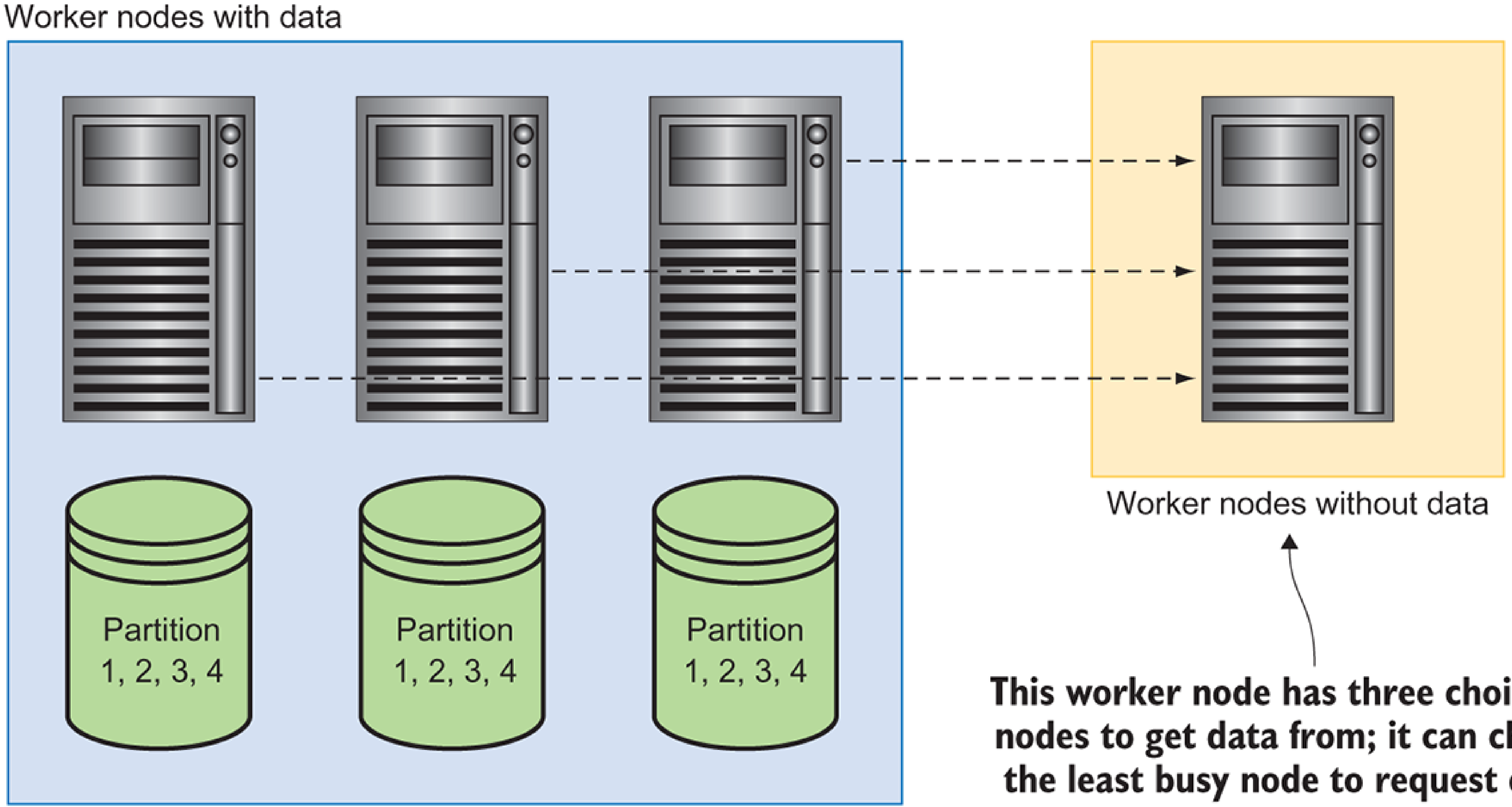
Circle nodes denote functions/computations.



All the data resides on a single node.

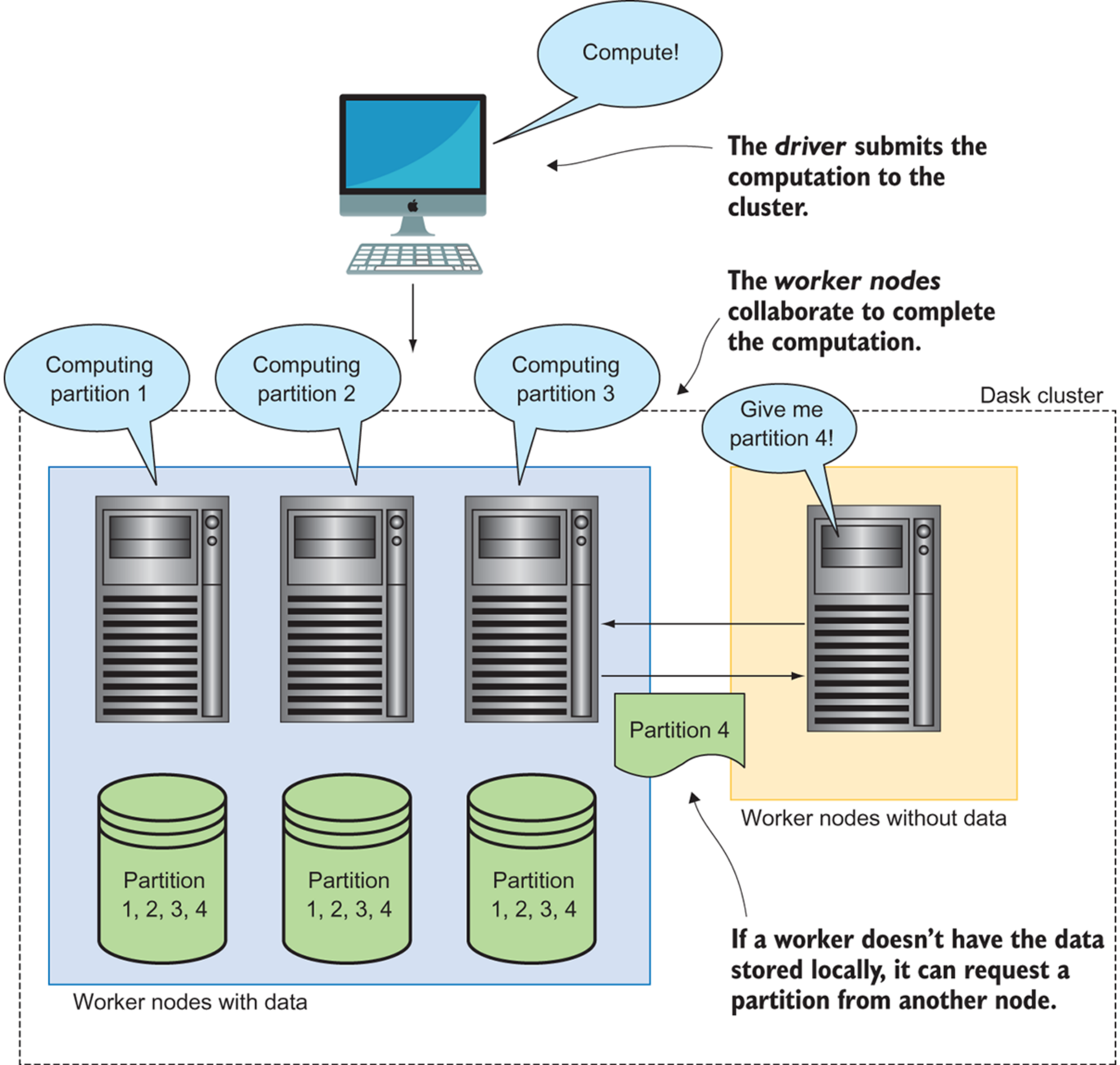


Other worker nodes that need data can easily saturate the single data node with requests, slowing our processing down to a halt.



This worker node has three choices of nodes to get data from; it can choose the least busy node to request data, minimizing bottlenecks.

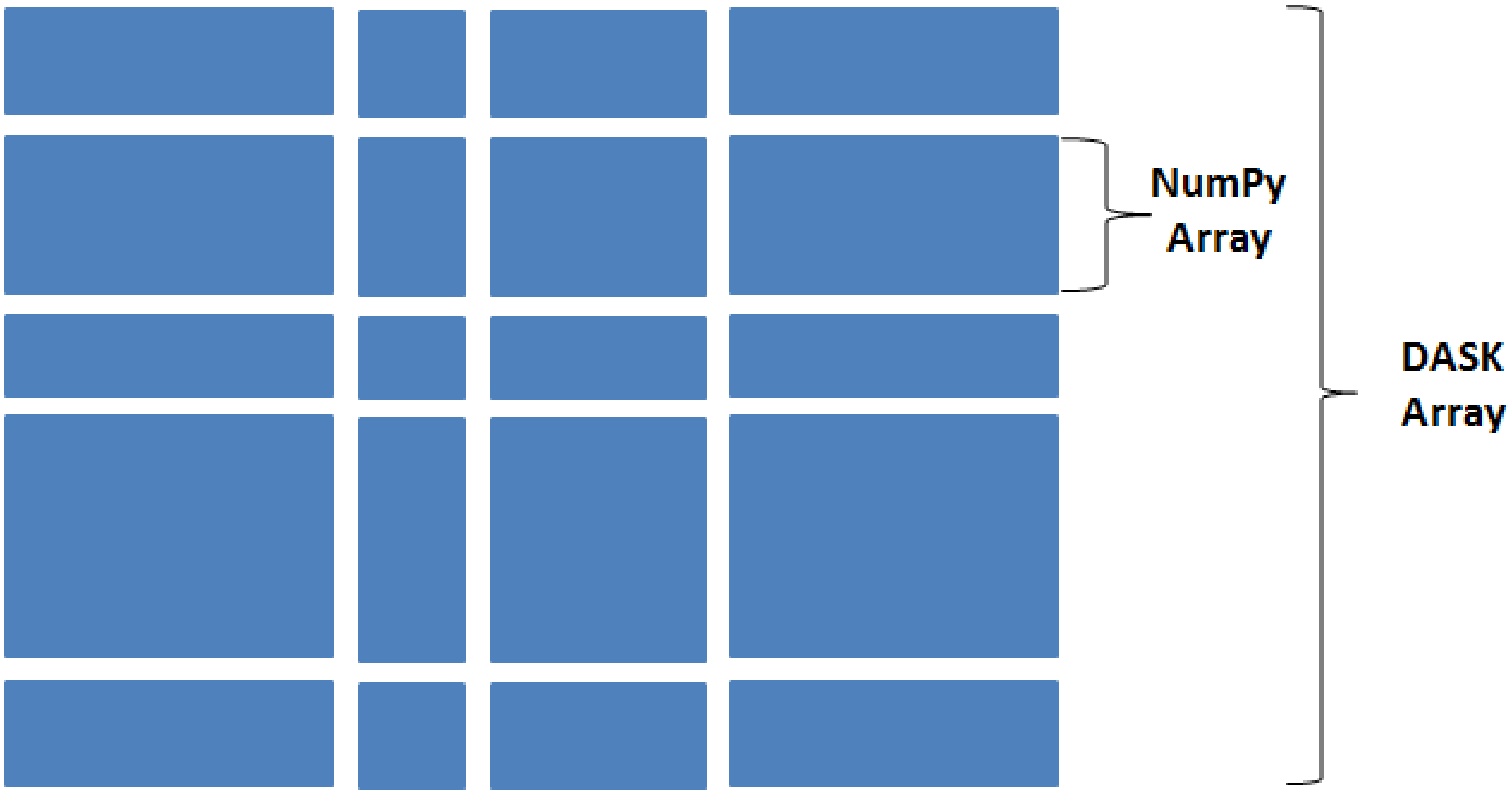
The data is stored in triplicate, eliminating single points of failure and performance bottlenecks.



The *driver* submits the computation to the cluster.

The *worker nodes* collaborate to complete the computation.

If a worker doesn’t have the data stored locally, it can request a partition from another node.



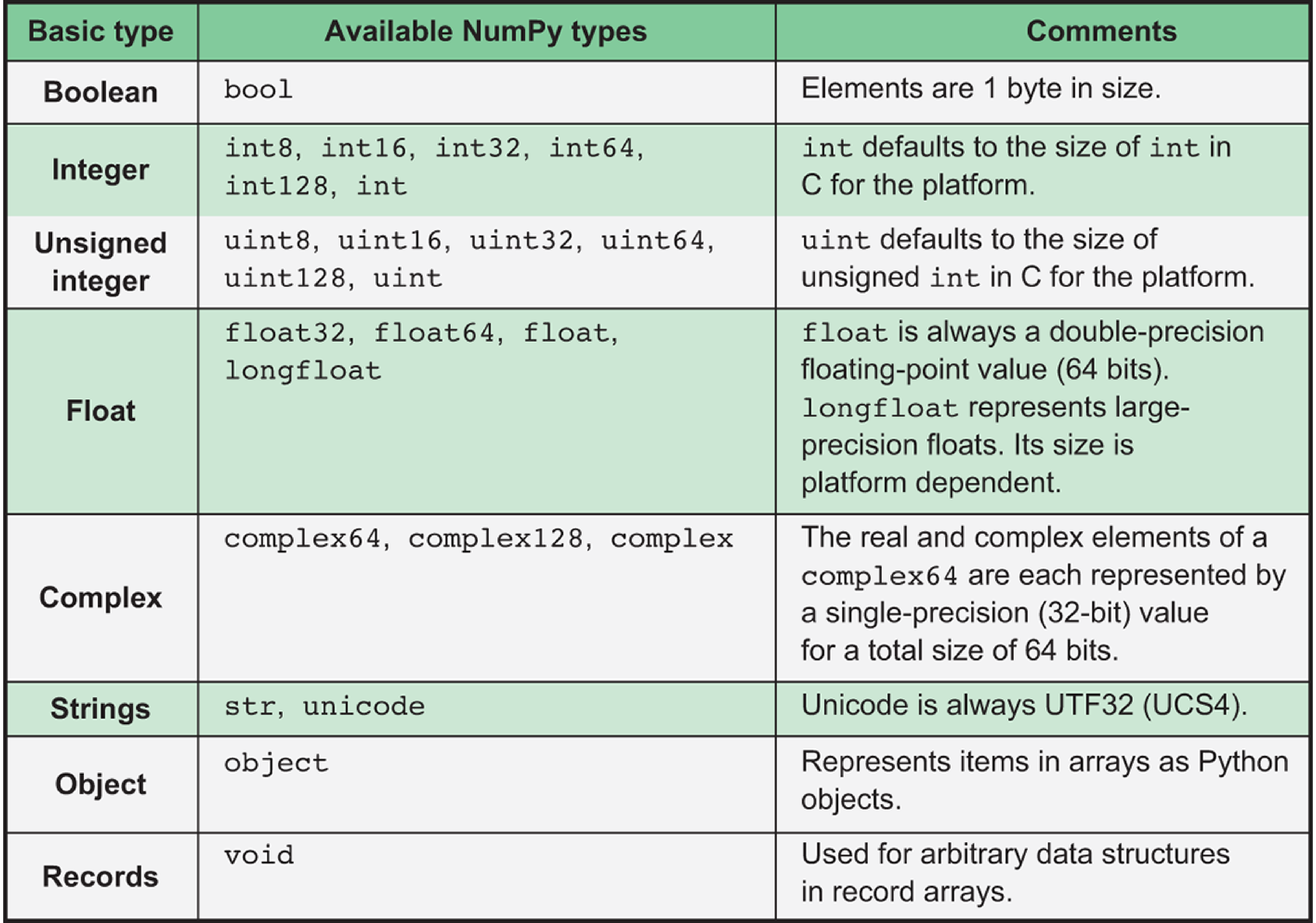
NumPy Array
DASK Array

```
import numpy as np

f = h5py.File('myfile.hdf5’)
x = np.array(f['/small-data’])
x - x.mean(axis=1)
```

```
import dask.array as da

f = h5py.File('myfile.hdf5’)
x = da.from_array(f['/big-data'], chunks=(1000, 1000))
x - x.mean(axis=1).compute()
```



Basic type	Available NumPy types	Comments
Boolean	`bool`	Elements are 1 byte in size.
Integer	`int8, int16, int32, int64, int128, int`	`int` defaults to the size of `int` in C for the platform.
Unsigned integer	`uint8, uint16, uint32, uint64, uint128, uint`	`uint` defaults to the size of unsigned `int` in C for the platform.
Float	`float32, float64, float, longfloat`	`float` is always a double-precision floating-point value (64 bits). `longfloat` represents large-precision floats. Its size is platform dependent.
Complex	`complex64, complex128, complex`	The real and complex elements of a `complex64` are each represented by a single-precision (32-bit) value for a total size of 64 bits.
Strings	`str, unicode`	Unicode is always UTF32 (UCS4).
Object	`object`	Represents items in arrays as Python objects.
Records	`void`	Used for arbitrary data structures in record arrays.