Yandex

Transformations 1

Two ways to construct RDDs

- Data in a stable storage (previous video)
  - Example: files in HDFS, objects in Amazon S3 bucket, lines in a text file, ...
- RDD for data in a stable storage has no dependencies

Two ways to construct RDDs

- Data in a stable storage (previous video)
  - Example: files in HDFS, objects in Amazon S3 bucket, lines in a text file, ...
  - RDD for data in a stable storage has no dependencies

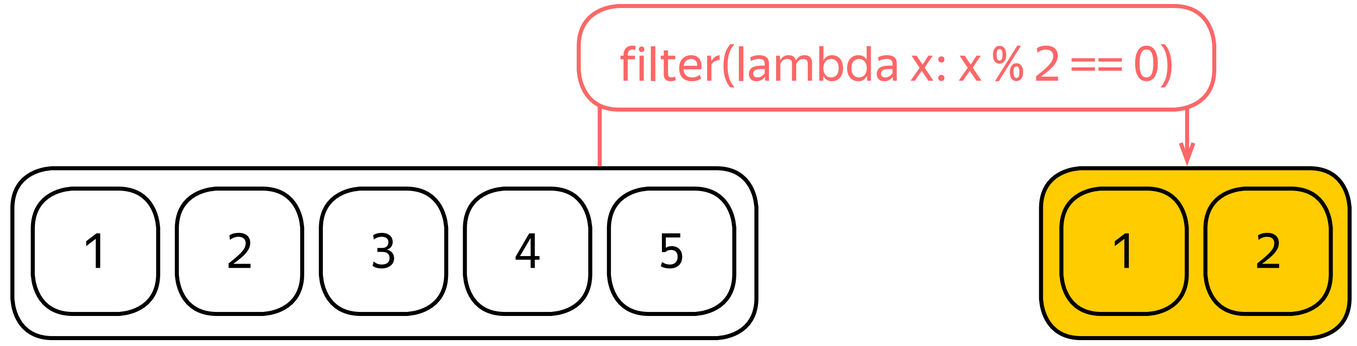
- From existing RDDs by applying a transformation (this video)
  - Example: filtered file, grouped records, ...
  - RDD for a transformed data depends on the source data

Transformations

- Allow you to create new RDDs from the existing RDDs by specifying how to obtain new items from the existing items
- The transformed RDD depends implicitly on the source RDD

Transformations

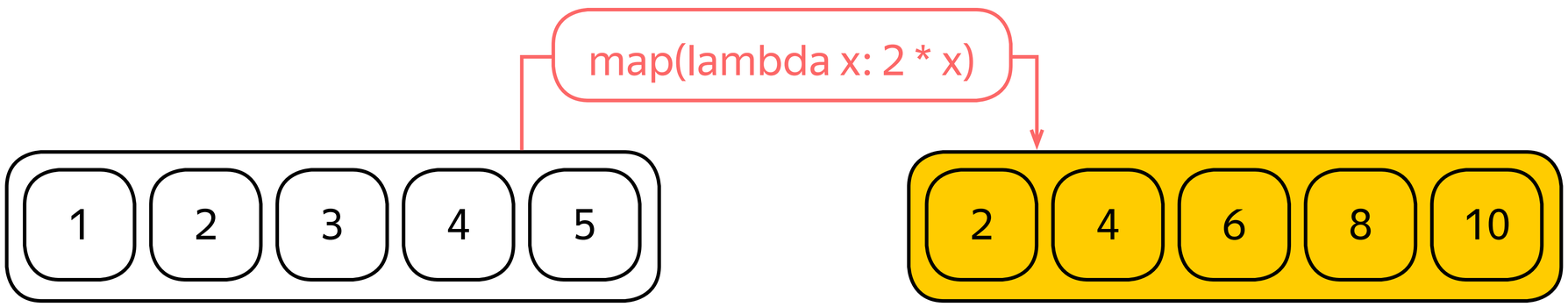
› Def: filter(p: *T ➞ Boolean): RDD[T] ➞ RDD[T]
  › returns a filtered RDD with items satisfying the predicate p



Transformations

- Def: filter(p: *T → Boolean*): *RDD[T] → RDD[T]*
  - returns a filtered RDD with items satisfying the predicate p

- Def: map(f: *T → U*): *RDD[T] → RDD[U]*
  - returns a mapped RDD with items f(x) for every x in the source RDD



Transformations

- Def: filter(p: *T* ➞ *Boolean*): *RDD[T]* ➞ *RDD[T]*
  - returns a filtered RDD with items satisfying the predicate p

- Def: map(f: *T* ➞ *U*): *RDD[T]* ➞ *RDD[U]*
  - returns a mapped RDD with items f(x) for every x in the source RDD

- Def: flatMap(f: *T* ➞ *Array[U]*): *RDD[T]* ➞ *RDD[U]*
  - same as map but flattens the result of f
  - generalizes map and filter

Filtered RDD

› Y = X.filter(p) # where X : RDD[T]

› Y.partitions() → *Array[Partition]*

› Y.iterator(p: *Partition*, parents: *Array[Iterator[T]]*) → *Iterator[T]*

› Y.dependencies() → *Array[Dependency]*

Filtered RDD

› Y = X.filter(p) # where X : RDD[T]
  › Y.partitions() ➞ *Array[Partition]*
    › return the same partitions as X

  › Y.iterator(p: *Partition*, parents: *Array[Iterator[T]]*) ➞ *Iterator[T]*

› Y.dependencies() ➞ *Array[Dependency]*

Filtered RDD

› Y = X.filter(p) # where X : RDD[T]
  › Y.partitions() → *Array[Partition]*
    › return the same partitions as X

  › Y.iterator(p: *Partition*, parents: *Array[Iterator[T]]*) → *Iterator[T]*

› Y.dependencies() → *Array[Dependency]*
  › k-th partition of Y depends on k-th partition of X

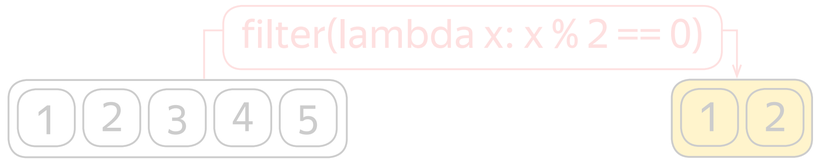
Filtered RDD

- Y = X.filter(p) # where X : RDD[T]
  - Y.partitions() → *Array[Partition]*
    - return the same partitions as X

  - Y.iterator(p: *Partition*, parents: *Array[Iterator[T]]*) → *Iterator[T]*
    - take a parent iterator over the corresponding partition of X
    - wrap the parent iterator to skip items that do not satisfy the predicate
    - return the iterator over partition of Y

- Y.dependencies() → *Array[Dependency]*
  - k-th partition of Y depends on k-th partition of X

Filtered RDD



› Y = X.filter(p) # where X : RDD[T]

› Y.partitions() ➞ *Array[Partition]*

› Y

Note that actual filtering happens not at
the creation time of Y, but at *the access time*
to the iterator over a partition of Y.

Same holds for other transformations – they are lazy,
i.e. they compute the result only when accessed.

› Y.dependencies() ➞ *Array[Dependency]*

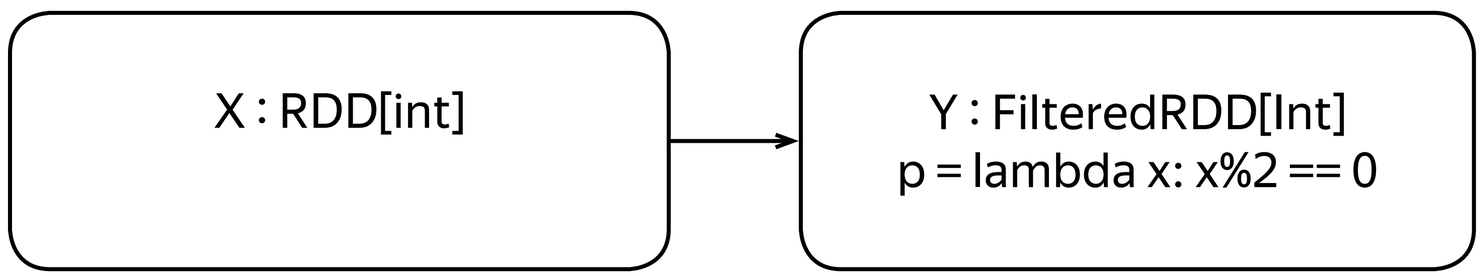
› k-th partition of Y depends on k-th partition of X

On closures

› Y = X.filter(lambda x: x % 2 == 0)
  › predicate closure is captured within the Y (it is a part of the definition of Y)
  › predicate is not guaranteed to execute locally (closure may be sent over the network to the executor)

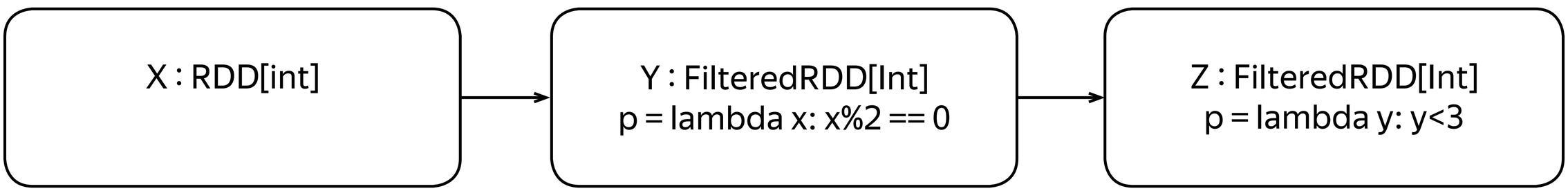
Dependency graph

› Y = X.filter(lambda x: x % 2 == 0)



Dependency graph

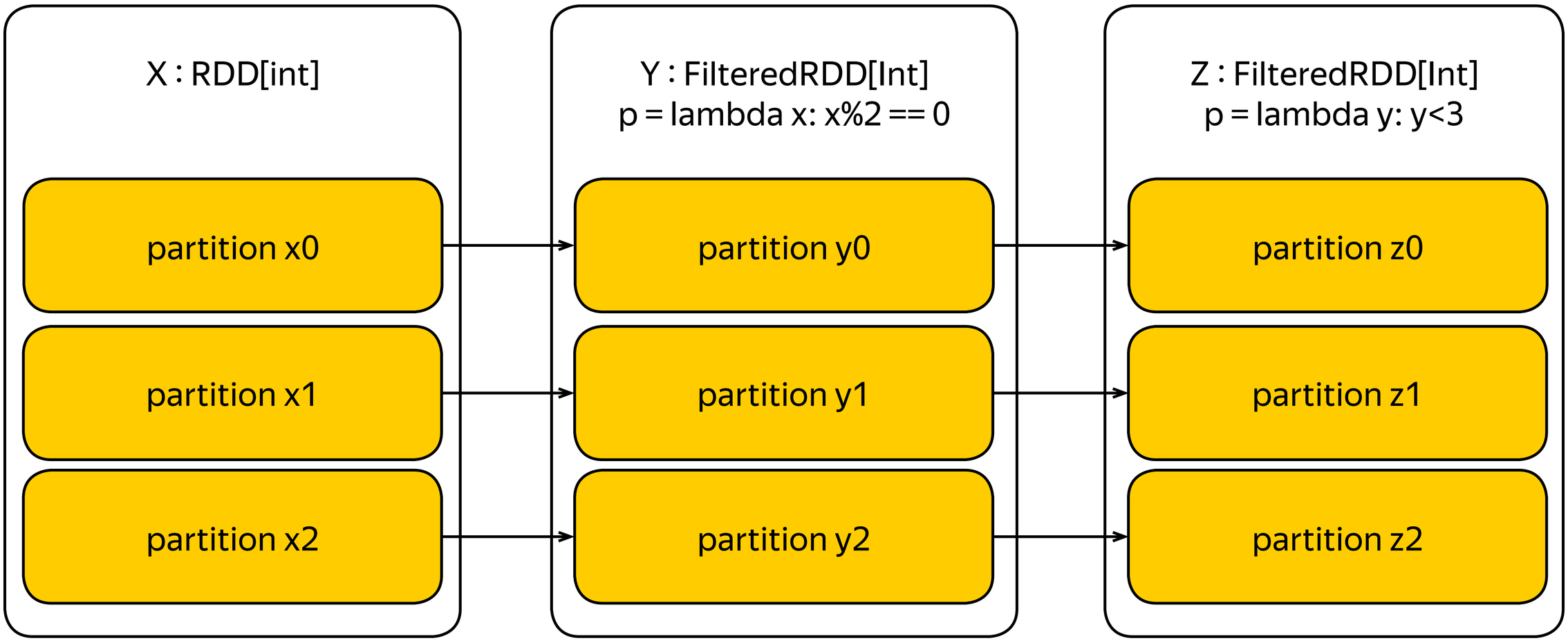
› Z = X.filter(lambda x: x % 2 == 0).filter(lambda y: y < 3)



RDD Graph, shows dependencies between RDDs

Partition dependency graph

› Z = X.filter(lambda x: x % 2 == 0).filter(lambda y: y < 3)



Partition Graph, shows dependencies between partitions

Quiz & short break

BigDATAteam