Partitioning From Zero to Hero

In this session:

In this session:

- Some queries can be tuned to use a single partition (partition elimination)
- Can solve specific latch contention issues by spreading writes
- You can backup only read-write partitions – faster backups.
- You can restore the important partitions first – shorter RTO.

Hi, I'm Daniel.

- SQL Server developer since 1999
- Owner and principal consultant of Structured Concepts AB
- Organizer of Data Saturday Stockholm
- Co-organizer of Group By
- Occasional blogger at sqlsunday.com
- Author of callfordataspeakers.com

Email: daniel@strd.co
Twitter: @dhmacher
Blog: sqlsunday.com

Sponsors rock our world.

SOLISYON

DATAmasterminds

SIGMADATA CONSULTING

redgate

macaw
Challenge accepted.

DCAC
Denny Cherry
& Associates Consulting

solarwinds

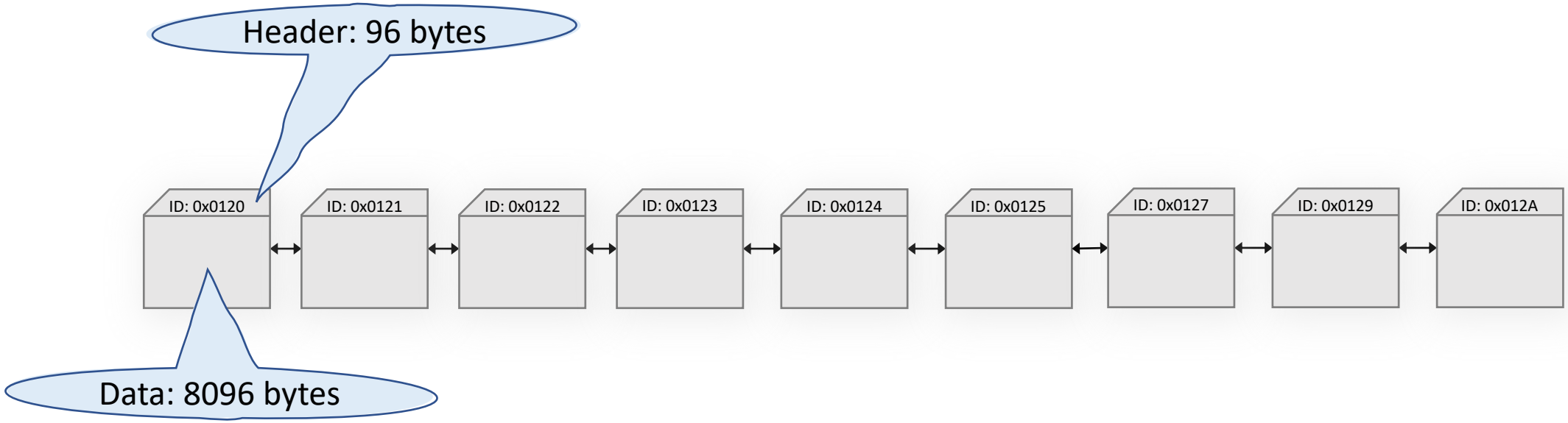
POWER BI SENTINEL™
Governance, Disaster Recovery and Auditing for Power BI

ADVANCING ANALYTICS

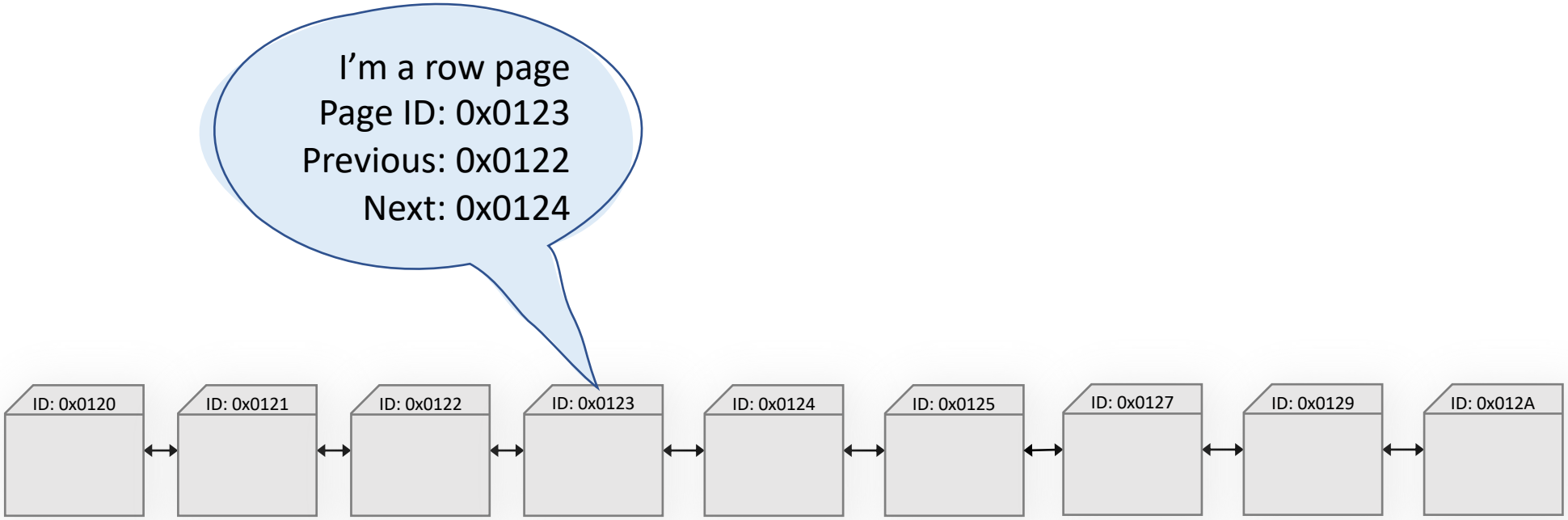
Simplifications ahead

How pages work

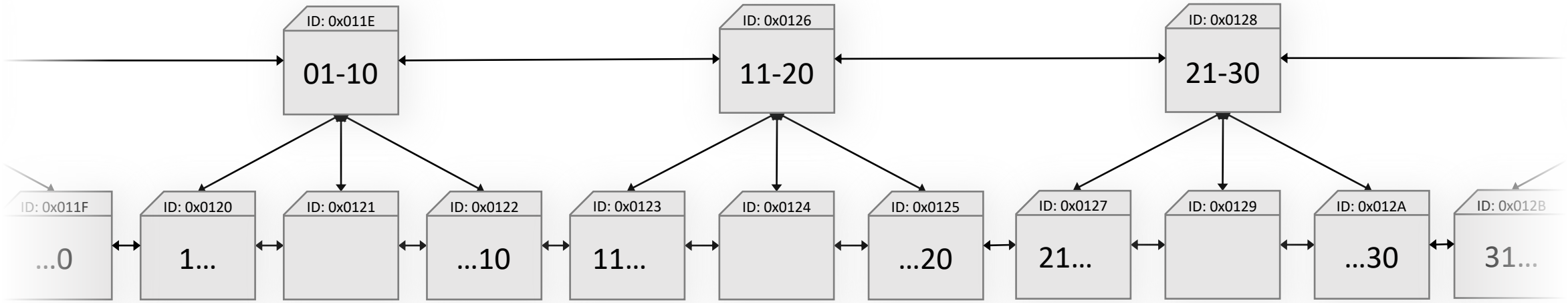
How pages work



How pages work

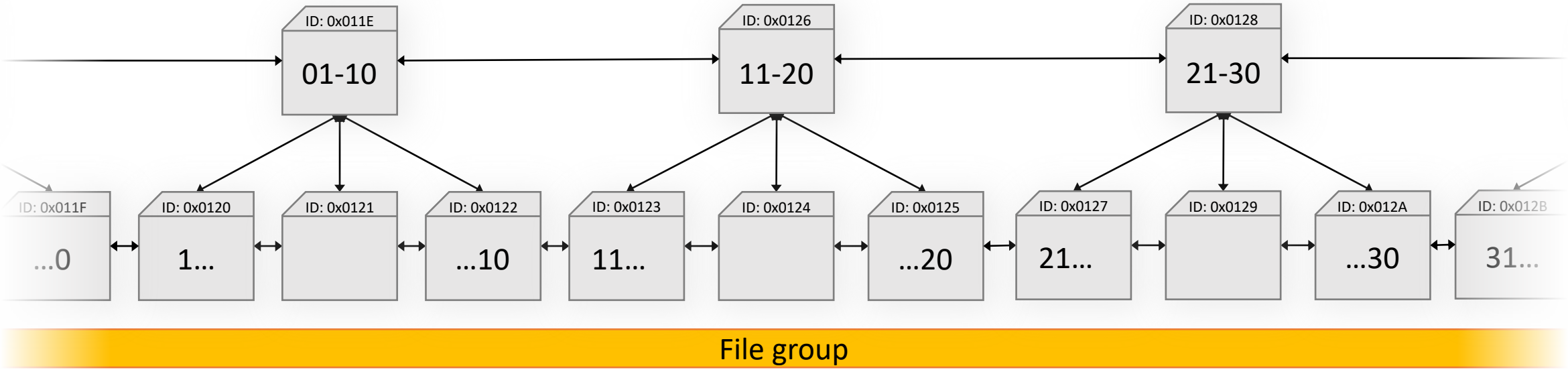


Indexes can have multiple levels



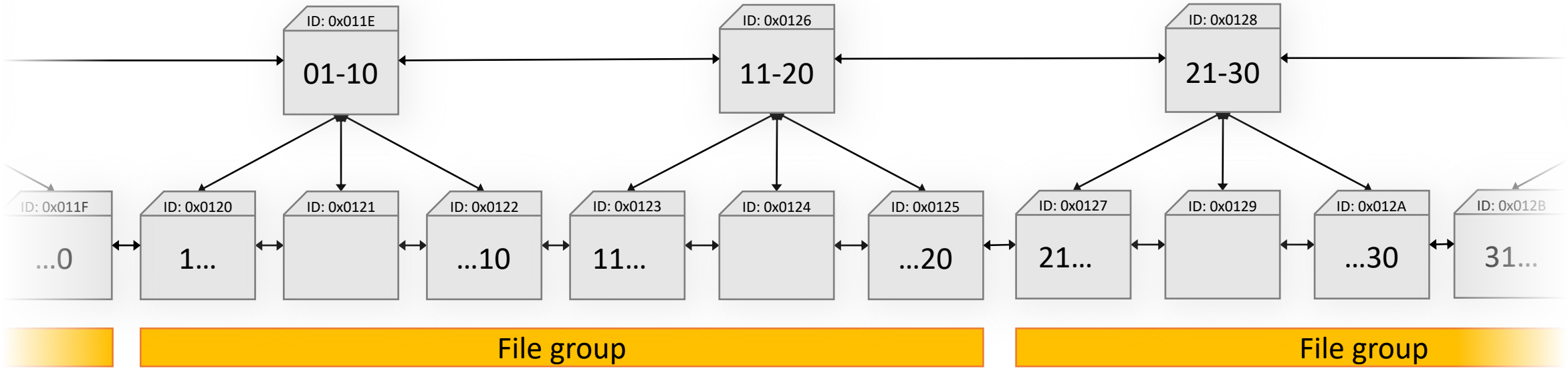
What is partitioning?

Regular table: one partition on one filegroup



What is partitioning?

Partitioned table: *multiple* partitions, on *one or more* filegroups.



Setting up partitioning

How: the partition function

```
CREATE PARTITION FUNCTION AnnualFunction(date)
AS RANGE RIGHT
        FOR VALUES ( '2020-01-01' , '2021-01-01' , '2022-01-01' );
```

Setting up partitioning

How: the partition function

```
CREATE PARTITION FUNCTION AnnualFunction(date)
AS RANGE RIGHT
        FOR VALUES ( '2020-01-01' , '2021-01-01' , '2022-01-01' );
```

2020-01-01 – 2020-12-31 2021-01-01 – 2021-12-31 2022-01-01 – 2022-12-31

Setting up partitioning

```
CREATE PARTITION FUNCTION AnnualFunction(date)
AS RANGE RIGHT
       FOR VALUES ( '2020-01-01' , '2021-01-01' , '2022-01-01' );

CREATE PARTITION SCHEME Annual
AS PARTITION AnnualFunction
TO ( [FILEGROUP_A] , [FILEGROUP_B] , [FILEGROUP_C] , [FILEGROUP_D] );
```

Where: the partition scheme

Setting up partitioning

```
CREATE PARTITION FUNCTION AnnualFunction(date)
AS RANGE RIGHT
       FOR VALUES ( '2020-01-01' , '2021-01-01' , '2022-01-01' );
```

```
CREATE PARTITION SCHEME Annual
AS PARTITION AnnualFunction
TO ( [FILEGROUP_A] , [FILEGROUP_B] , [FILEGROUP_C] , [FILEGROUP_D] );
```

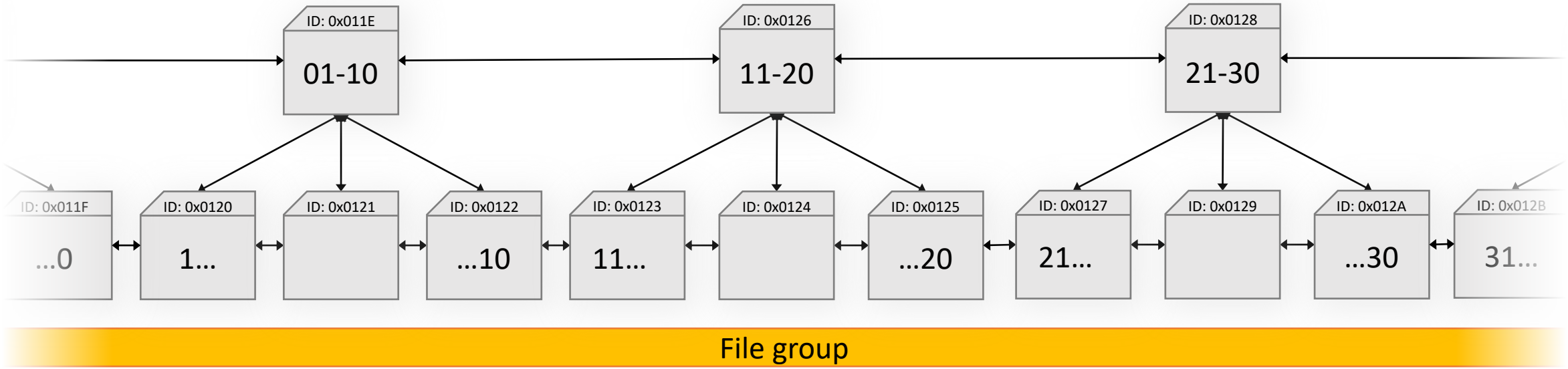
```
CREATE TABLE dbo.partitioned_table (
    TransactionDate    date NOT NULL,
    ...
    ...
    ...
) ON Annual(TransactionDate);
```

What: assigning the index to the partition scheme

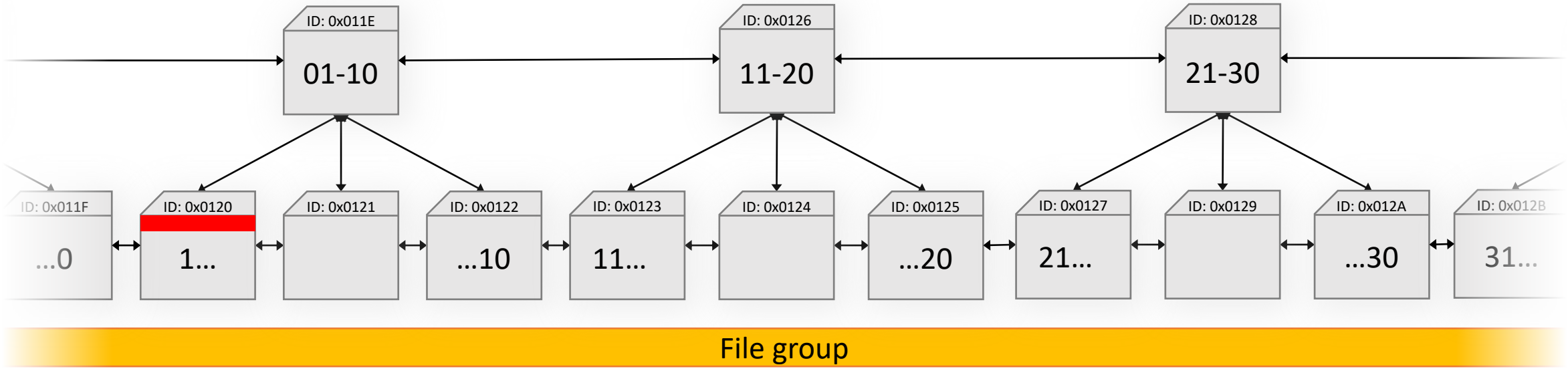

38
DEMO

A regular delete

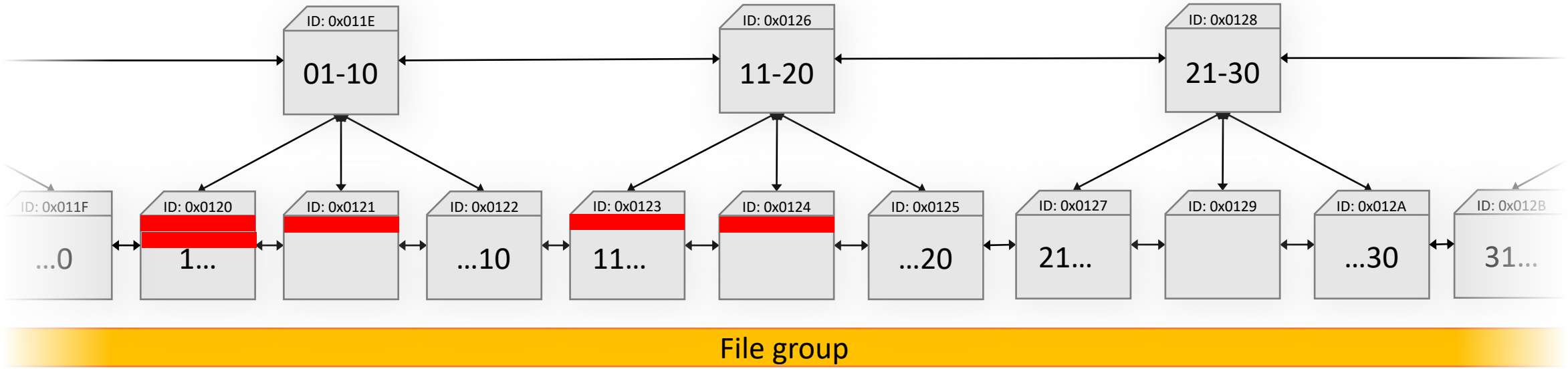
```
DELETE FROM dbo.regular_table
WHERE id BETWEEN 1 AND 20;
```



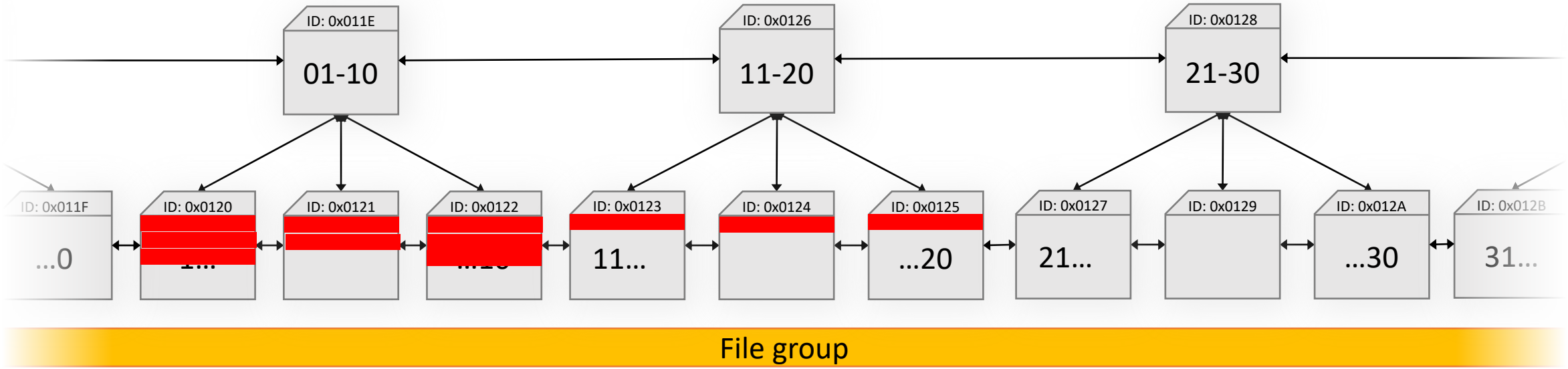
A regular delete



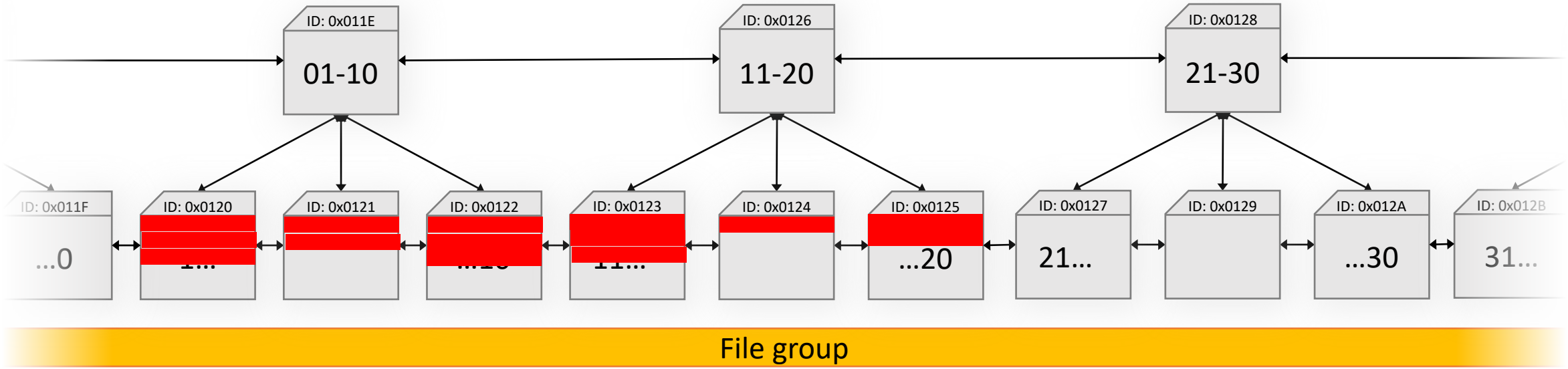
A regular delete



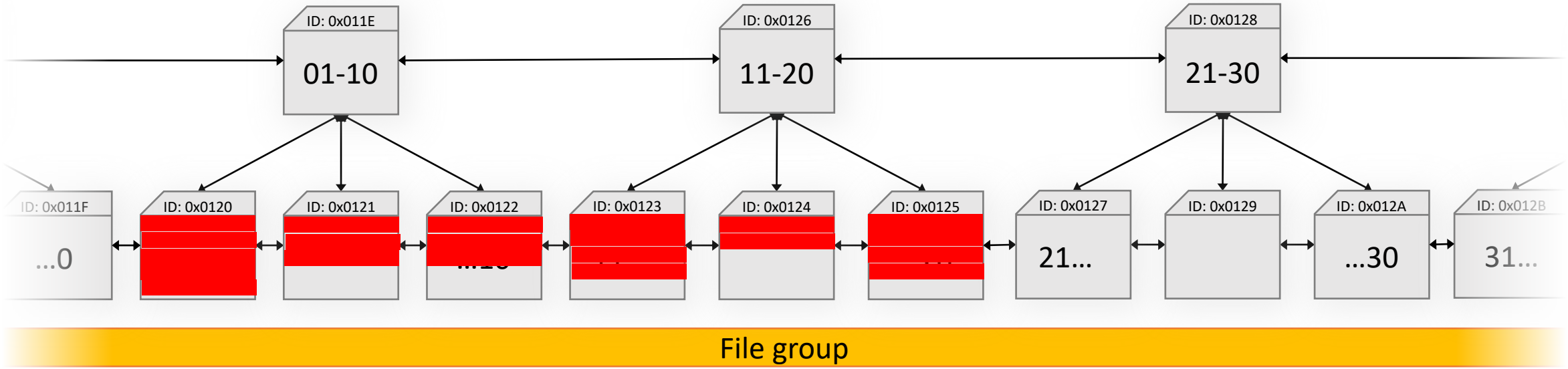
A regular delete



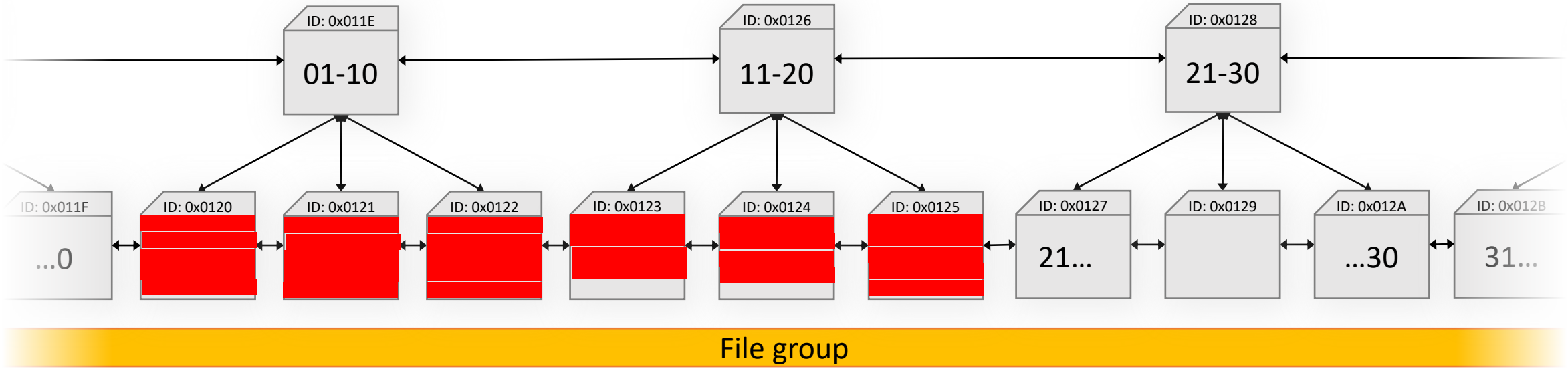
A regular delete



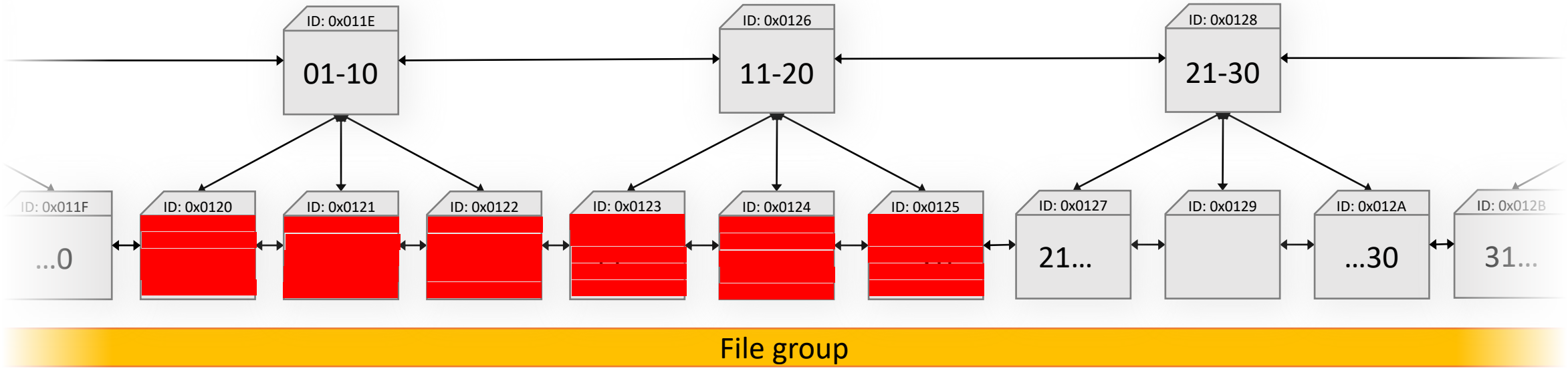
A regular delete



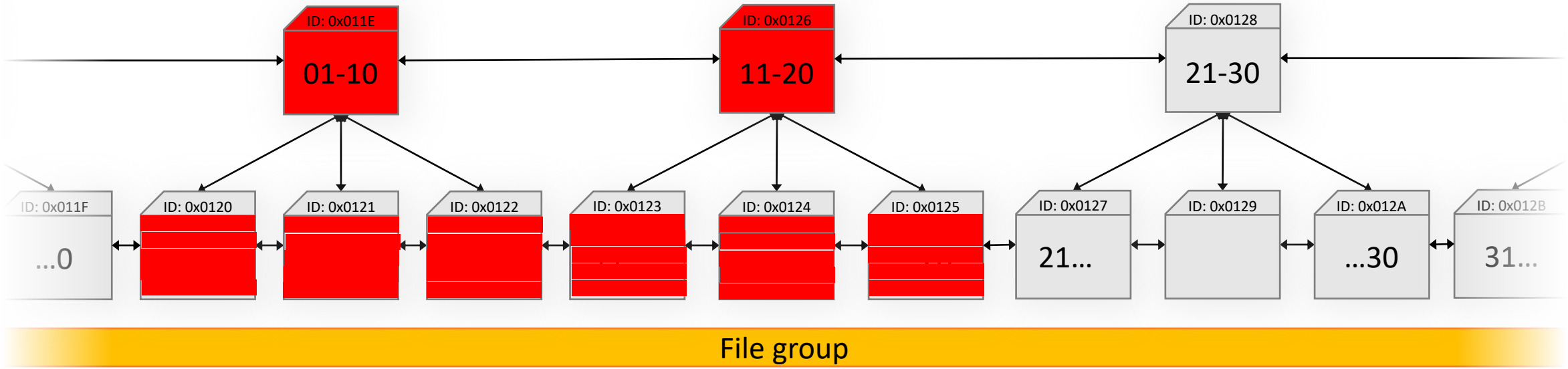
A regular delete



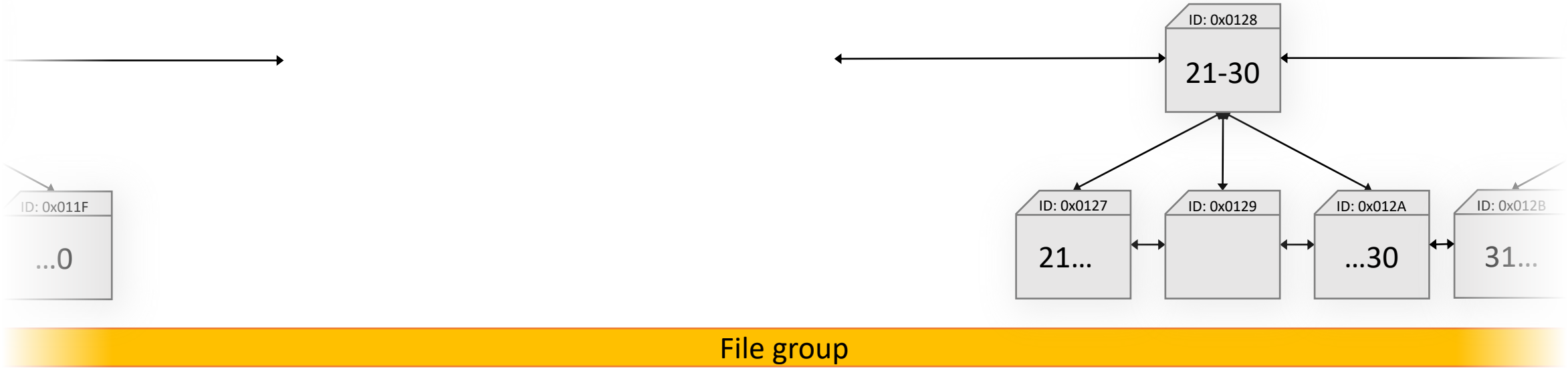
A regular delete



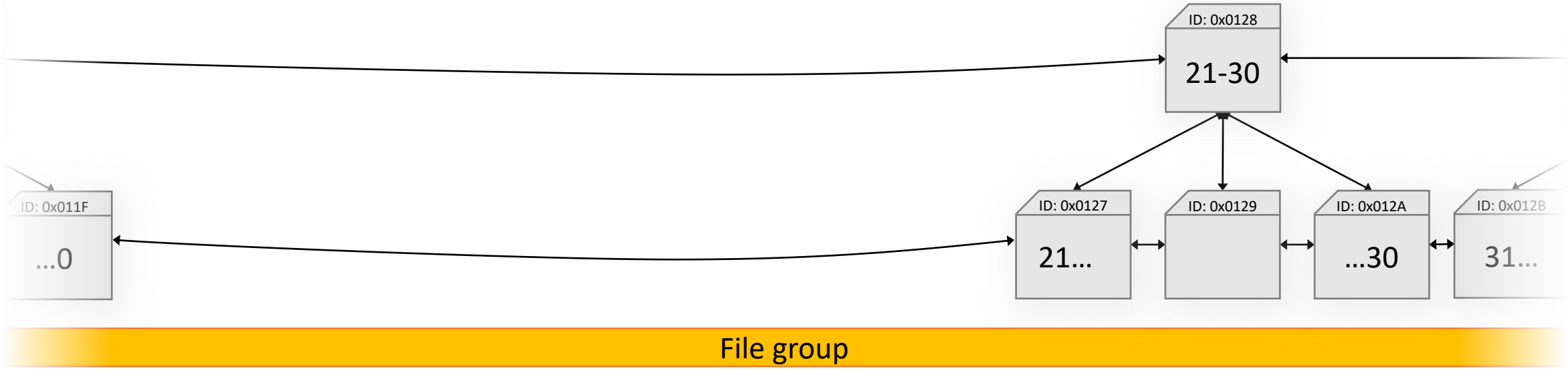
A regular delete



A regular delete

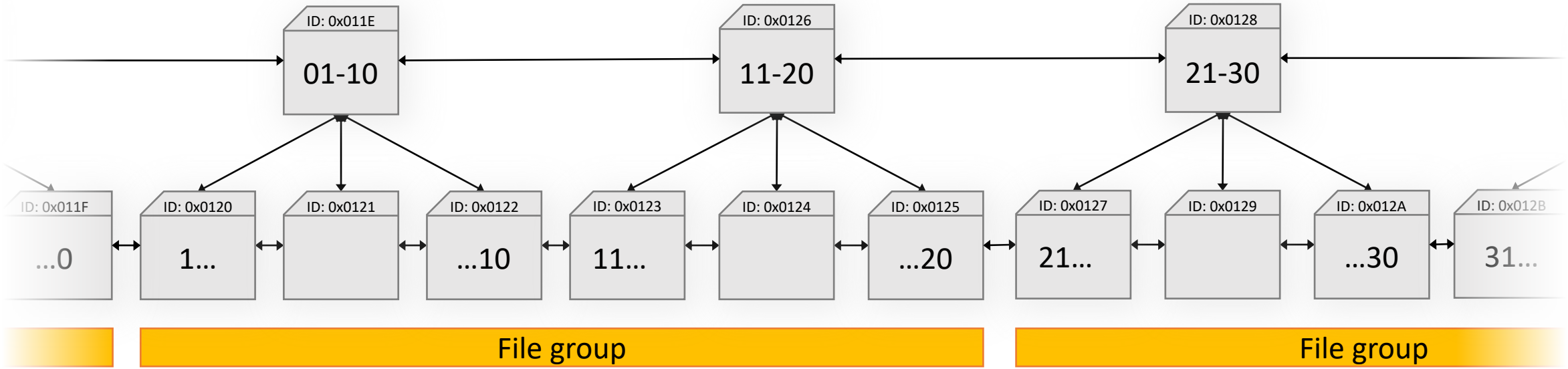


A regular delete



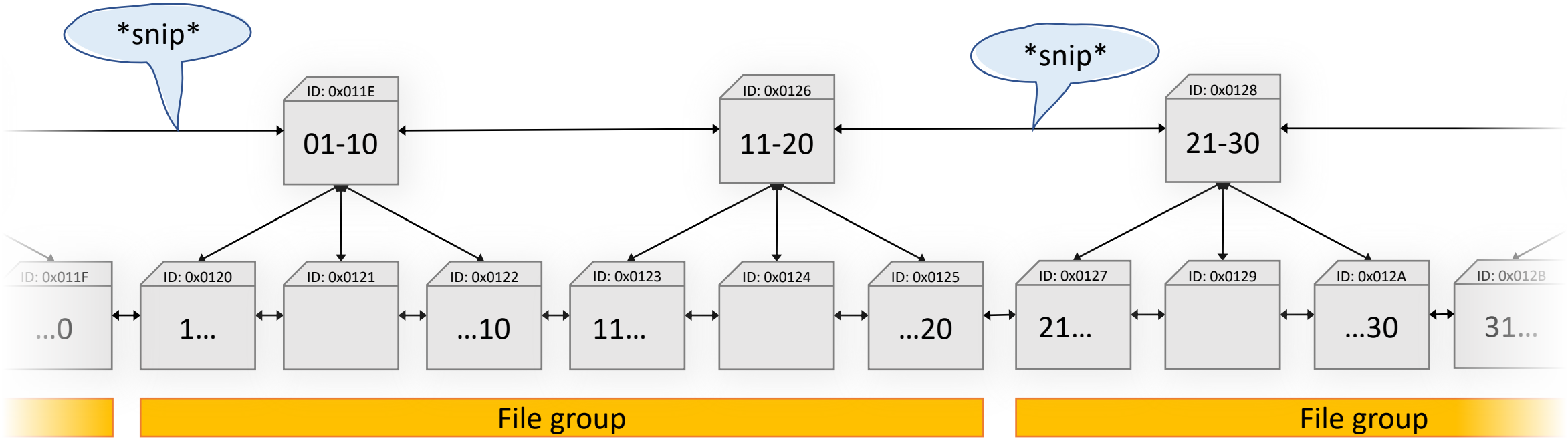
Truncating a partition

```
TRUNCATE TABLE dbo.partitioned_table
WITH (PARTITIONS (2));
```

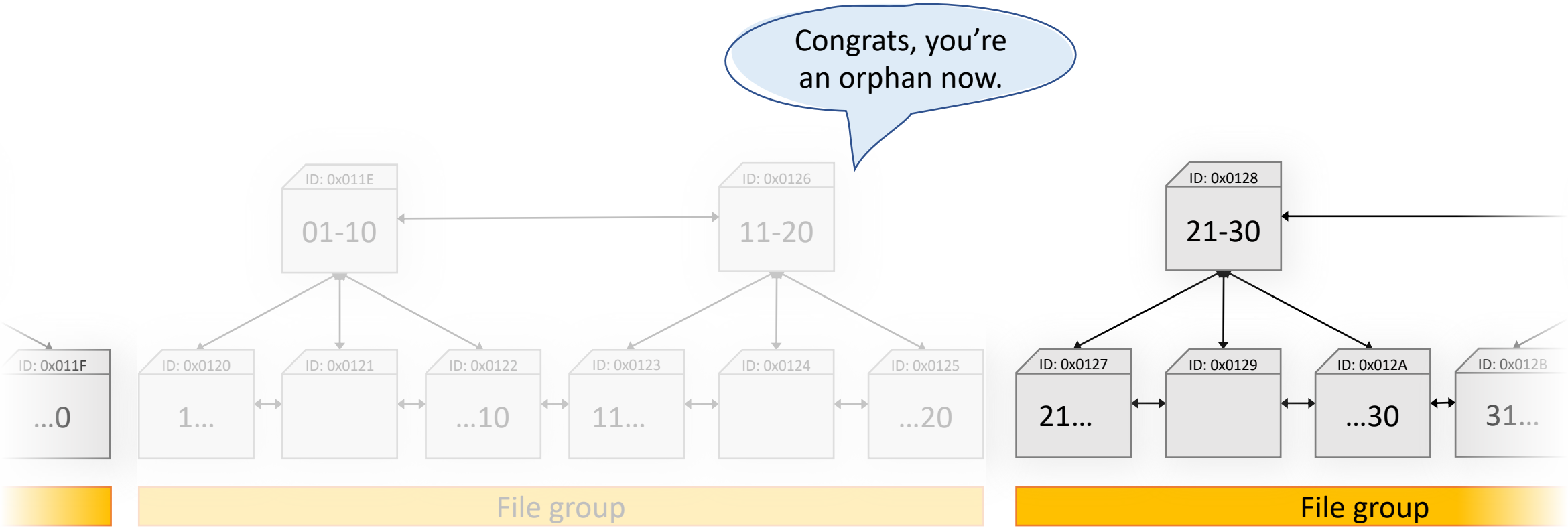


* Truncating individual partitions is available on SQL Server 2016+

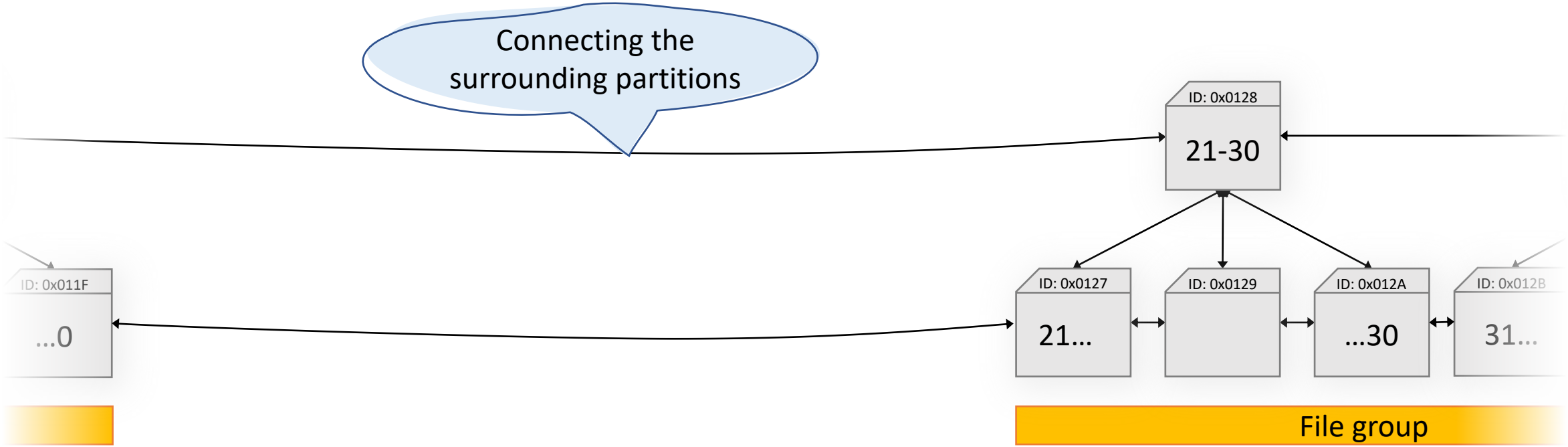
Truncating a partition



Truncating a partition



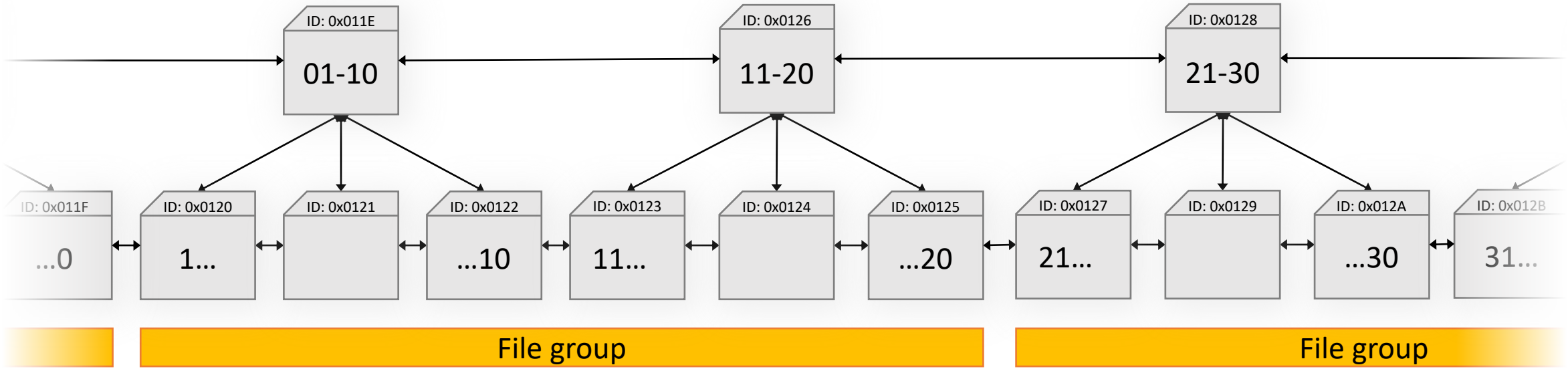
Truncating a partition




DEMO

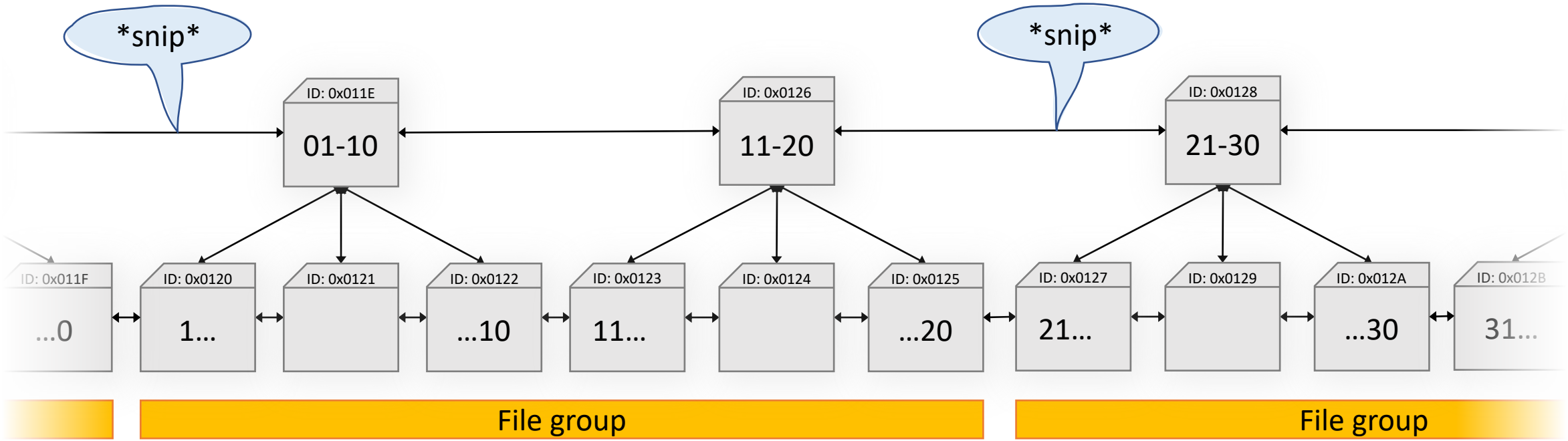
Switching a partition

```
ALTER TABLE dbo.partitioned_table
SWITCH PARTITION 2 TO dbo.regular_table;
```



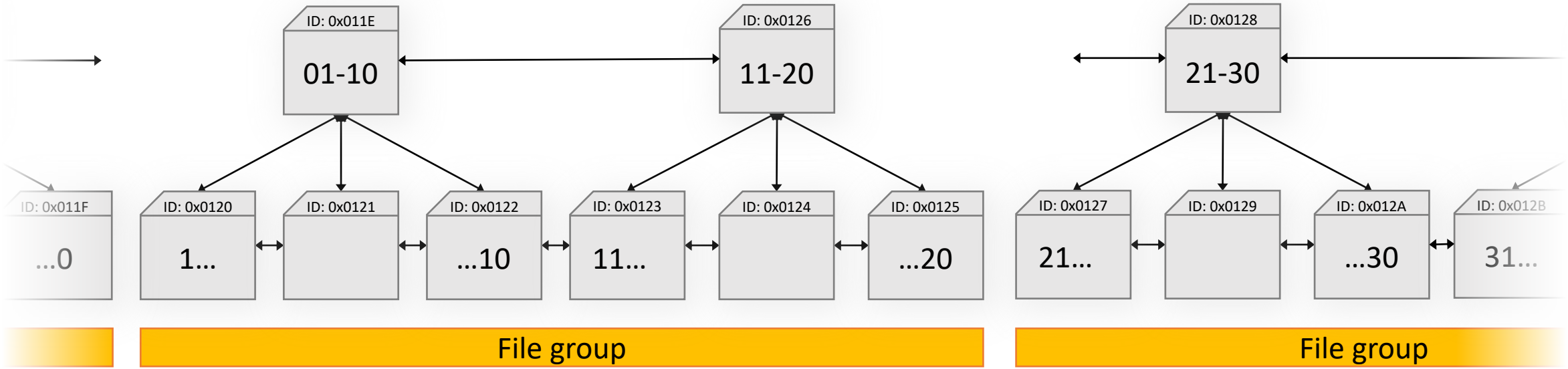
Switching a partition

```
ALTER TABLE dbo.partitioned_table
SWITCH PARTITION 2 TO dbo.regular_table;
```



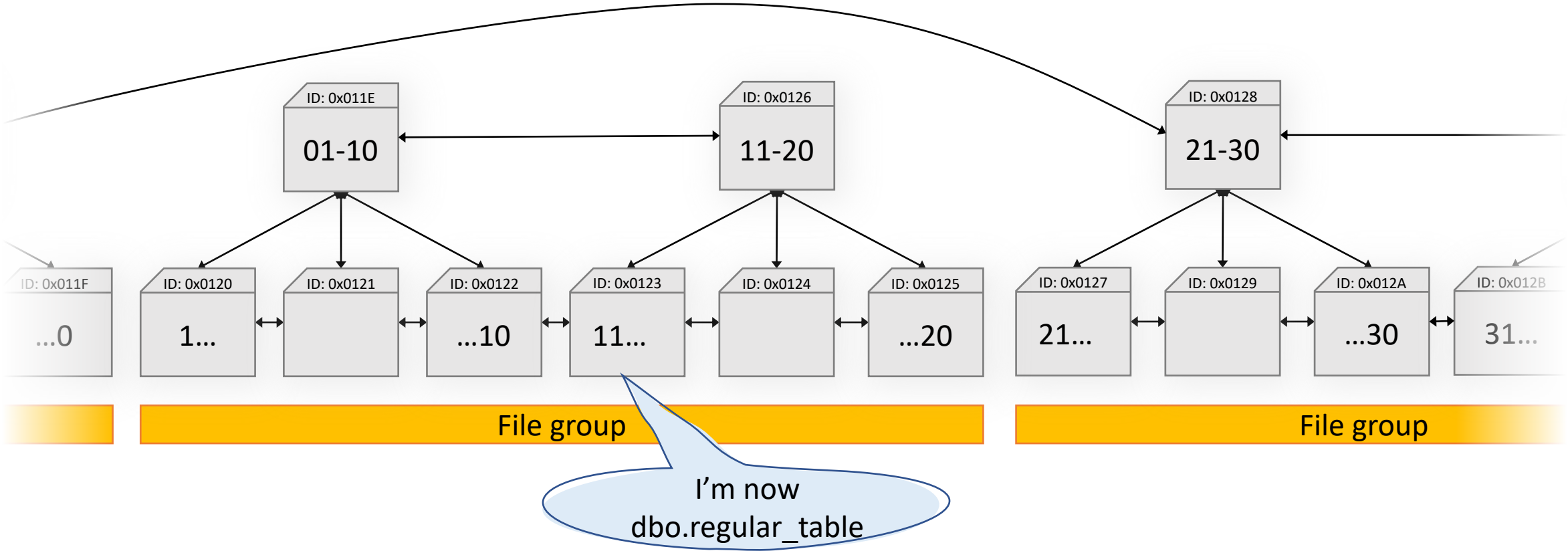
Switching a partition

```
ALTER TABLE dbo.partitioned_table
SWITCH PARTITION 2 TO dbo.regular_table;
```



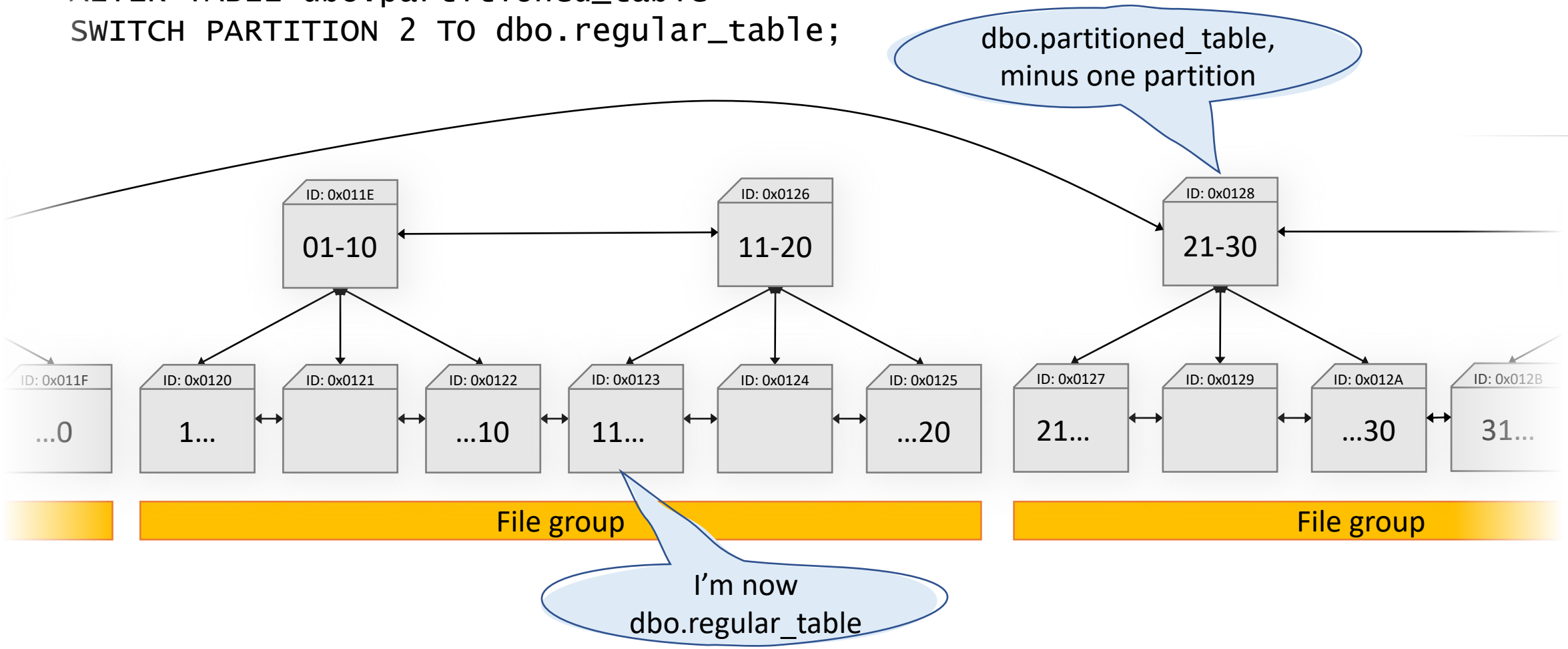
Switching a partition

```
ALTER TABLE dbo.partitioned_table
SWITCH PARTITION 2 TO dbo.regular_table;
```



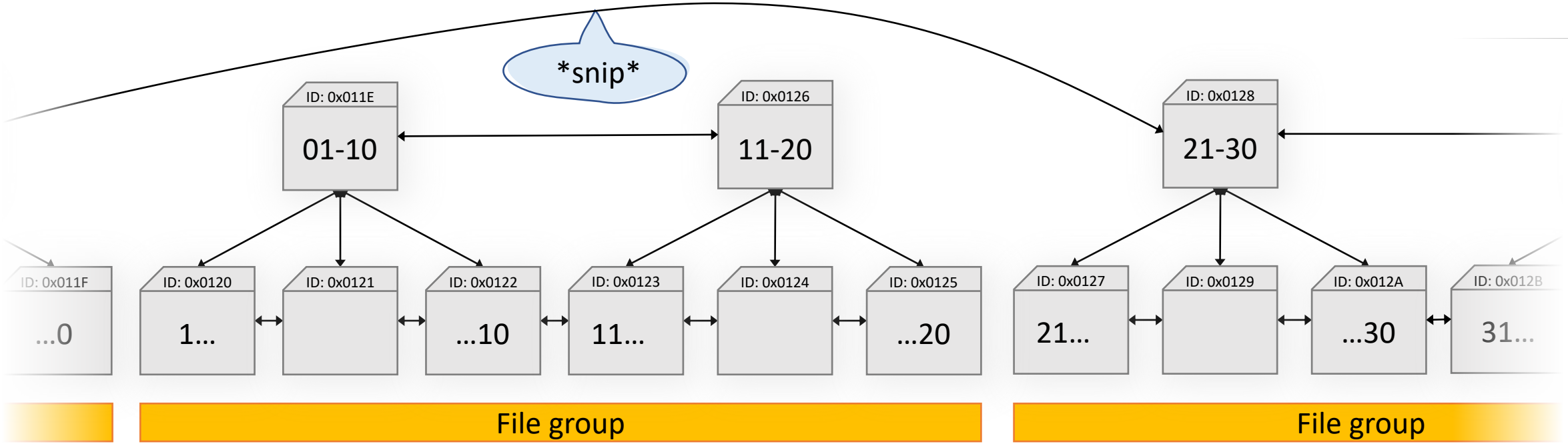
Switching a partition

```
ALTER TABLE dbo.partitioned_table
SWITCH PARTITION 2 TO dbo.regular_table;
```



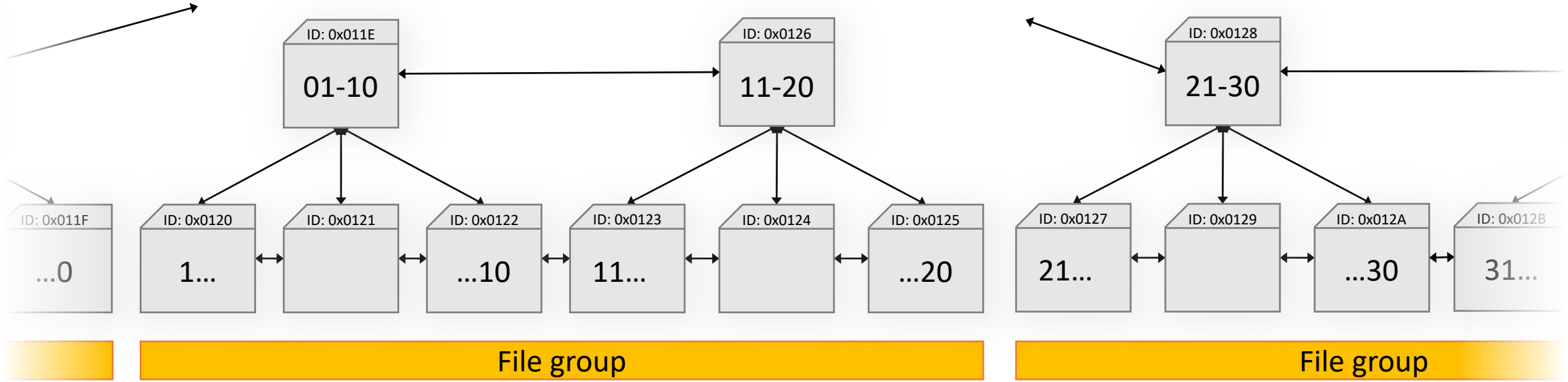
.. and back

```
ALTER TABLE dbo.regular_table
SWITCH TO dbo.partitioned_table PARTITION 2;
```



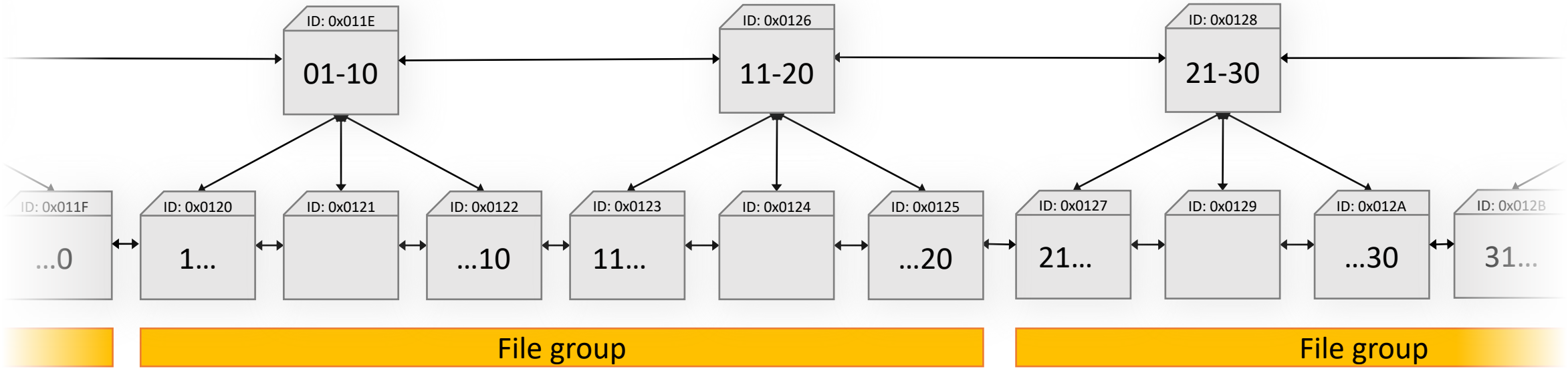
.. and back

```
ALTER TABLE dbo.regular_table
SWITCH TO dbo.partitioned_table PARTITION 2;
```



.. and back

```
ALTER TABLE dbo.regular_table
SWITCH TO dbo.partitioned_table PARTITION 2;
```



* Bonus fact: you can switch two *non-partitioned* tables, too!

These are metadata operations!

- The data doesn’t move.
- Only the pointers are updated.
- Metadata operations require metadata permissions!


DEMO

"Right is right"

… AS RANGE LEFT
FOR VALUES (100, 200, 300)

	$n \leq$	100
100	$< n \leq$	200
200	$< n \leq$	300
300	$< n$	

… AS RANGE RIGHT
FOR VALUES (100, 200, 300)

	$n <$	100
100	$\leq n <$	200
200	$\leq n <$	300
300	$\leq n$	

"Right is right"

```
CREATE PARTITION FUNCTION AnnualFunction(date)
AS RANGE RIGHT
        FOR VALUES ( '2020-01-01' , '2021-01-01' , '2022-01-01' );

CREATE PARTITION SCHEME Annual
AS PARTITION AnnualFunction
TO ( [FILEGROUP_A] , [FILEGROUP_B] , [FILEGROUP_C] , [FILEGROUP_D] );
```

< 2020-01-01

≥ 2020-01-01
< 2021-01-01

"Right is right"

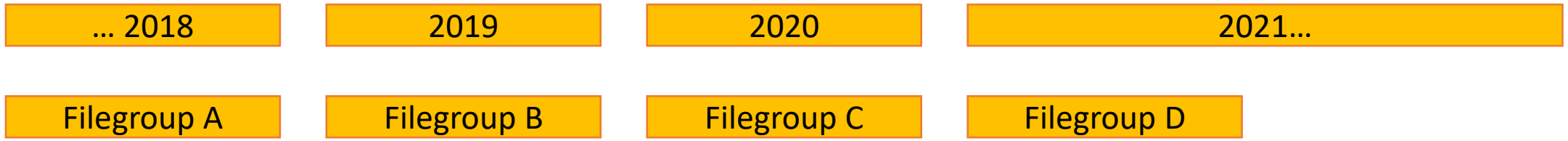
```
CREATE PARTITION FUNCTION AnnualFunction(date)
AS RANGE LEFT
        FOR VALUES ( '2020-01-01' , '2021-01-01' , '2022-01-01' );

CREATE PARTITION SCHEME Annual
AS PARTITION AnnualFunction
                TO ( [FILEGROUP_A] , [FILEGROUP_B] , [FILEGROUP_C] , [..._D] );
```

≤ 2020-01-01

> 2020-01-01
≤ 2021-01-01

Split/merge

- You can *split* an existing partition into two, by adding a new boundary.



Split/merge

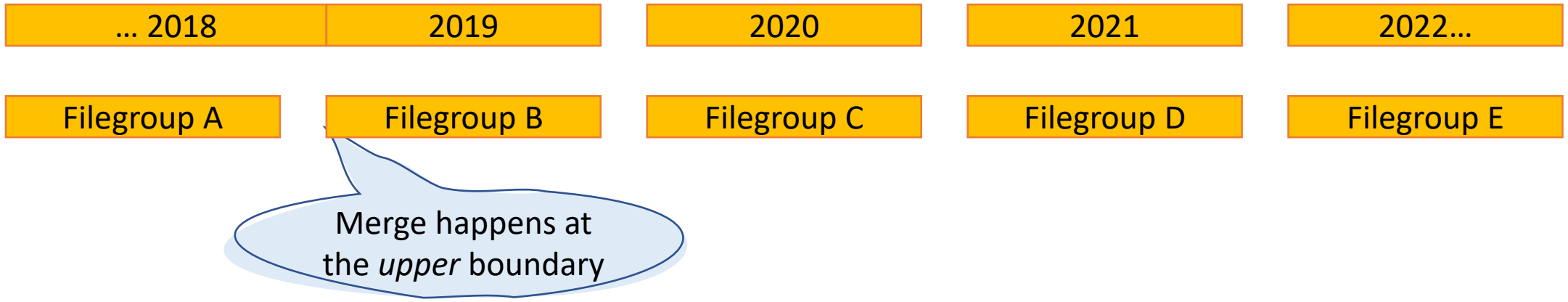
- You can *split* an existing partition into two, by adding a new boundary.

Split/merge

- You can *split* an existing partition into two, by adding a new boundary.

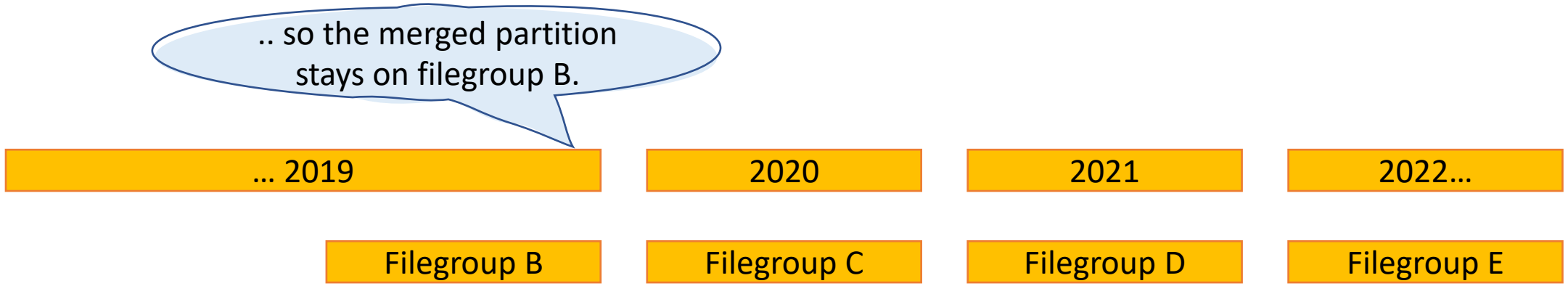
Split/merge

- You can *split* an existing partition into two, by adding a new boundary.
- Two partitions can also be *merged* into one, by removing a boundary.



Split/merge

- You can *split* an existing partition into two, by adding a new boundary.
- Two partitions can also be *merged* into one, by removing a boundary.

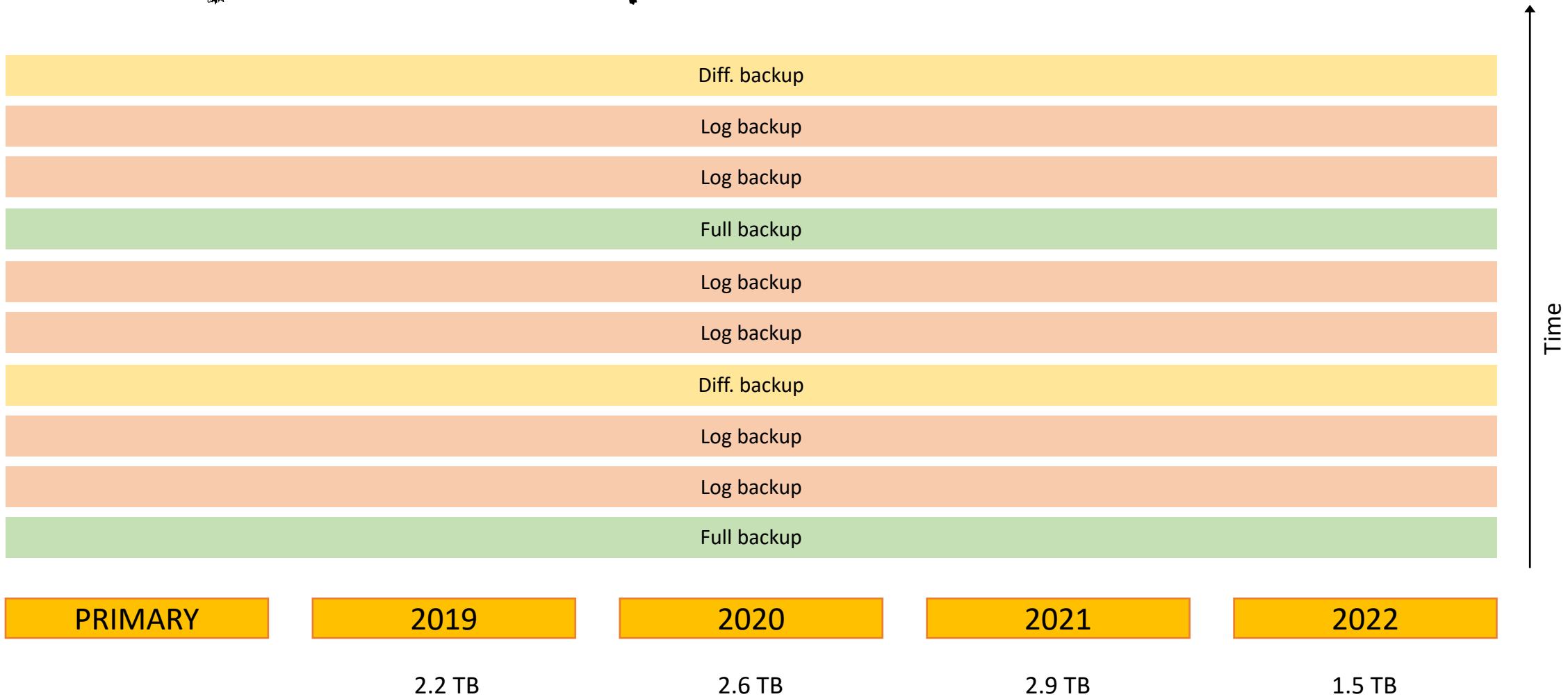



DEMO

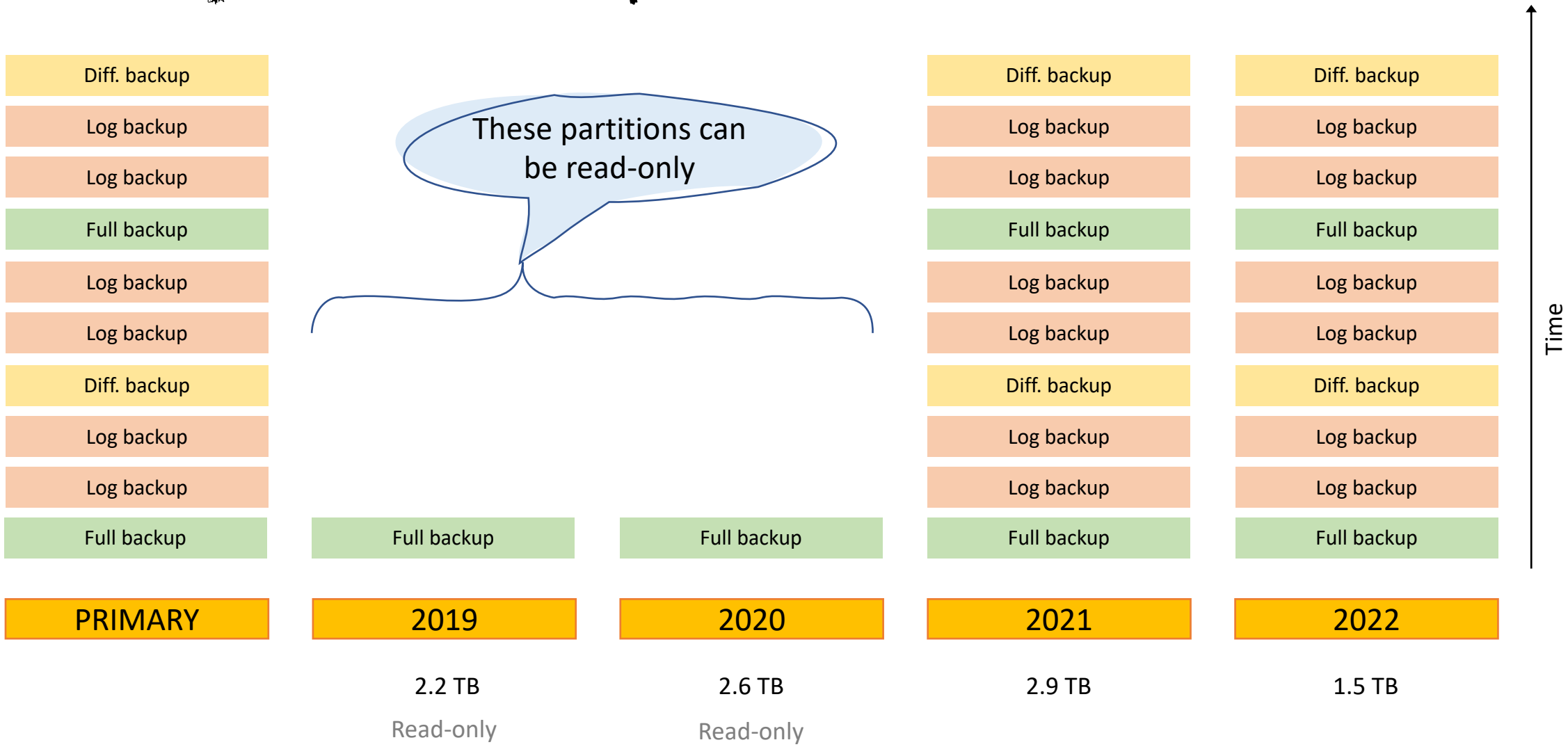
Aligned / non-aligned partitions

- If the table and its indexes all share the same partition scheme, the indexes are “aligned”
- If one or more indexes are not partitioned, or differently partitioned, they are “non-aligned”, and you won’t be able to perform partition-level operations like SPLIT, MERGE, TRUNCATE, SWITCH.

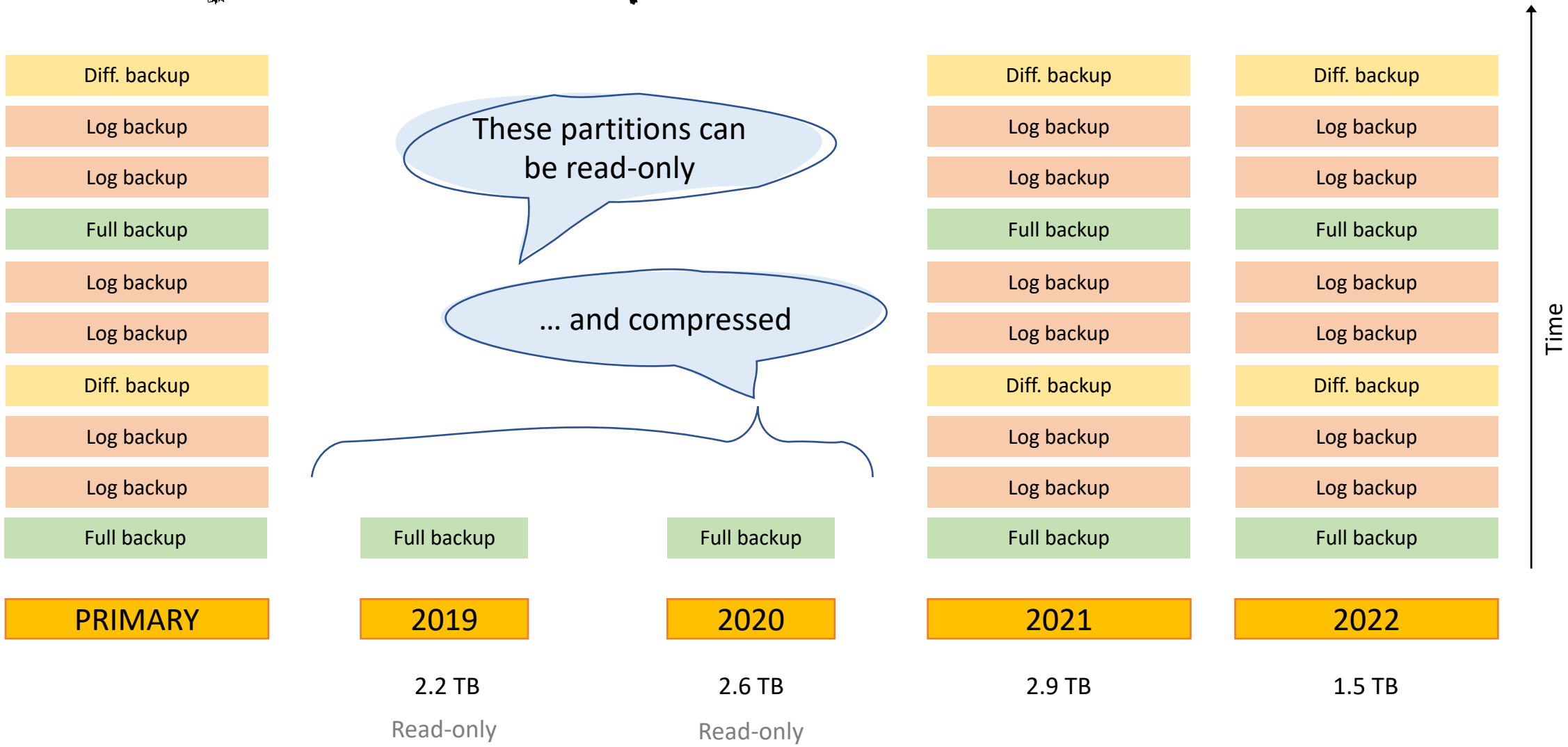
Make your backups smaller



Make your backups smaller



Make your backups smaller



Piecemeal restore (full)

We have:

- An old FULL backup of the read-only filegroups
- A recent FULL backup of the read-write filegroups

Piecemeal restore (full)

We have:

- An old FULL backup of the read-only filegroups
- A recent FULL backup of the read-write filegroups

Full backup, partial, recovery

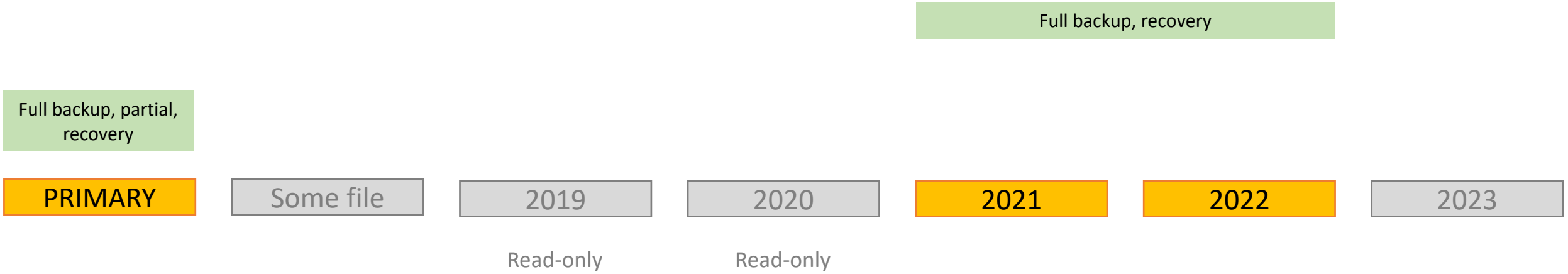
PRIMARY | Some file | 2019 | 2020 | 2021 | 2022 | 2023

Read-only | Read-only

Piecemeal restore (full)

We have:

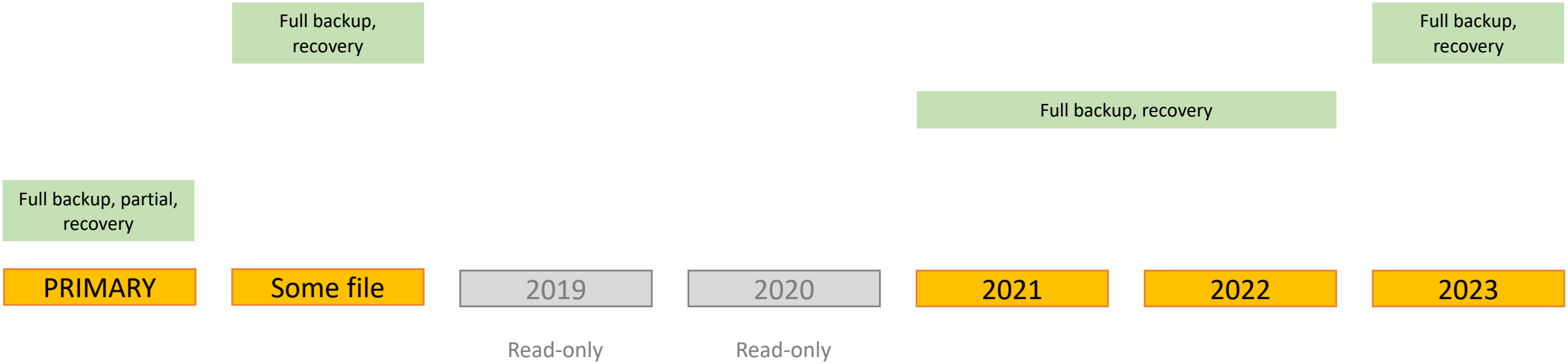
- An old FULL backup of the read-only filegroups
- A recent FULL backup of the read-write filegroups



Piecemeal restore (full)

We have:

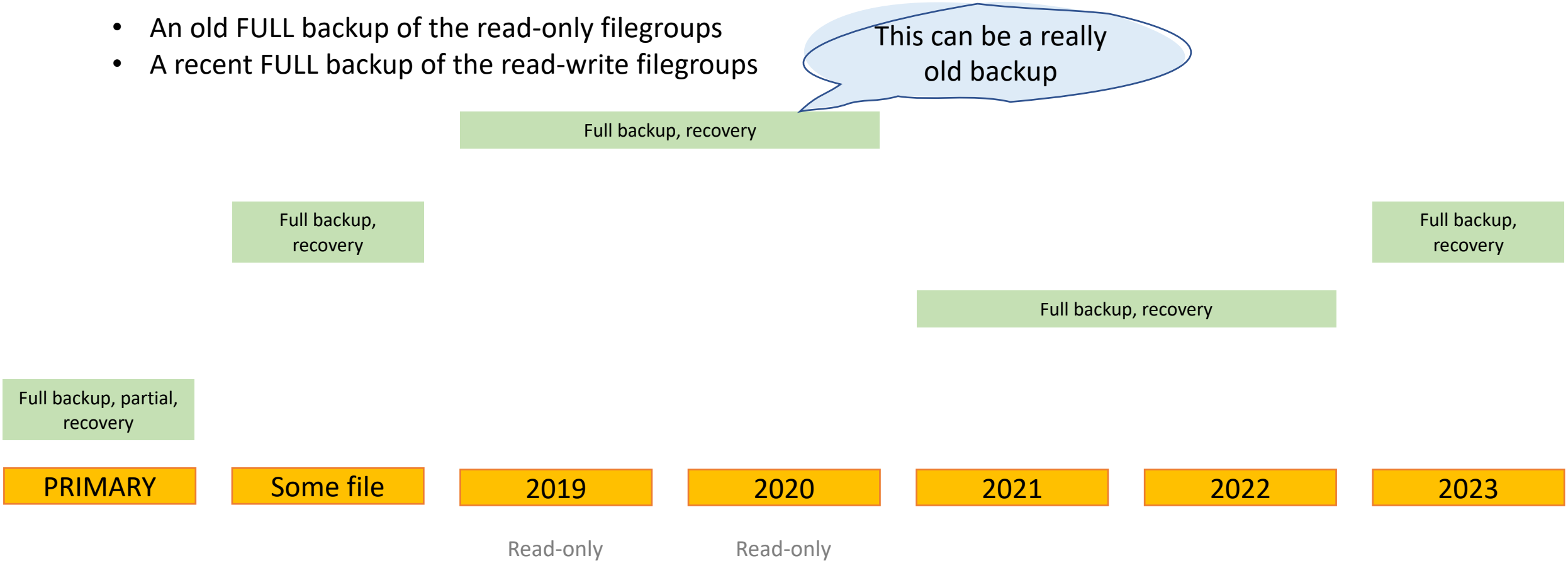
- An old FULL backup of the read-only filegroups
- A recent FULL backup of the read-write filegroups



Piecemeal restore (full)

We have:

- An old FULL backup of the read-only filegroups
- A recent FULL backup of the read-write filegroups




SSC
DEMO

Piecemeal restore (full + log)

We have:

- An old FULL backup of the read-only filegroups
- A recent FULL backup of the read-write filegroups
- Transaction log backup(s) for the read-write filegroups

Piecemeal restore (full + log)

We have:

- An old FULL backup of the read-only filegroups
- A recent FULL backup of the read-write filegroups
- Transaction log backup(s) for the read-write filegroups

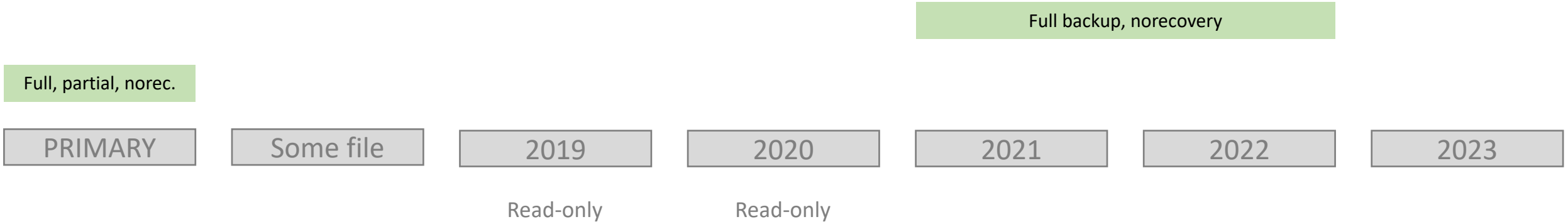
Full, partial, norec.

PRIMARY	Some file	2019	2020	2021	2022	2023
		Read-only	Read-only			

Piecemeal restore (full + log)

We have:

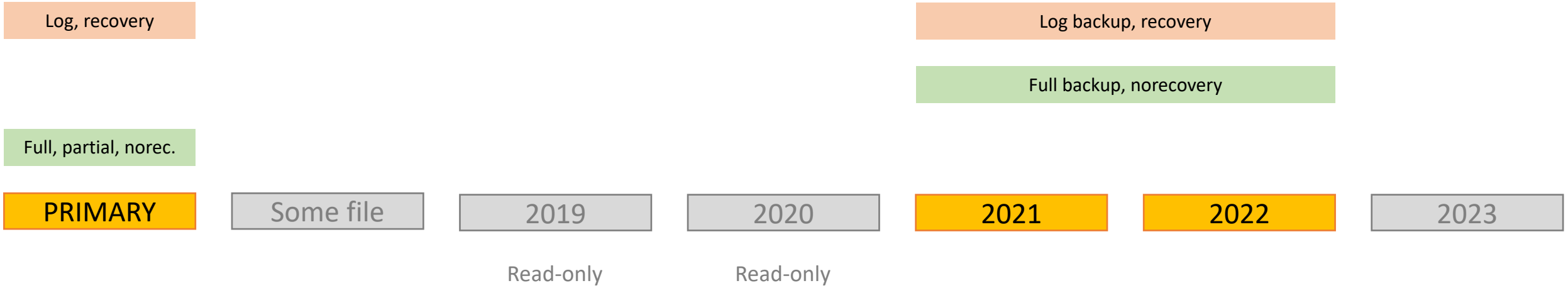
- An old FULL backup of the read-only filegroups
- A recent FULL backup of the read-write filegroups
- Transaction log backup(s) for the read-write filegroups



Piecemeal restore (full + log)

We have:

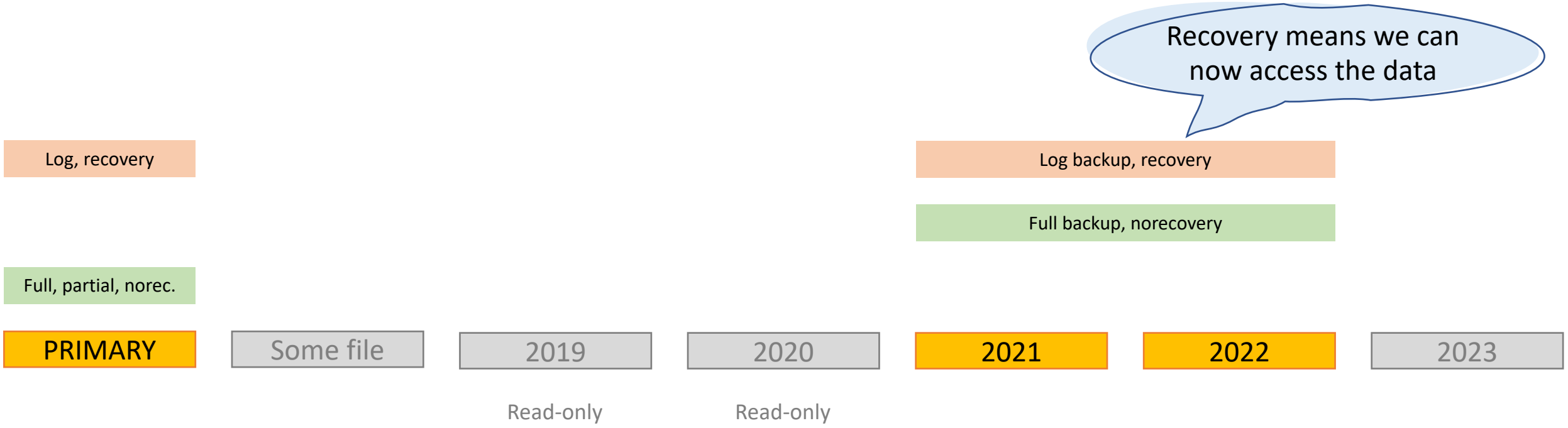
- An old FULL backup of the read-only filegroups
- A recent FULL backup of the read-write filegroups
- Transaction log backup(s) for the read-write filegroups



Piecemeal restore (full + log)

We have:

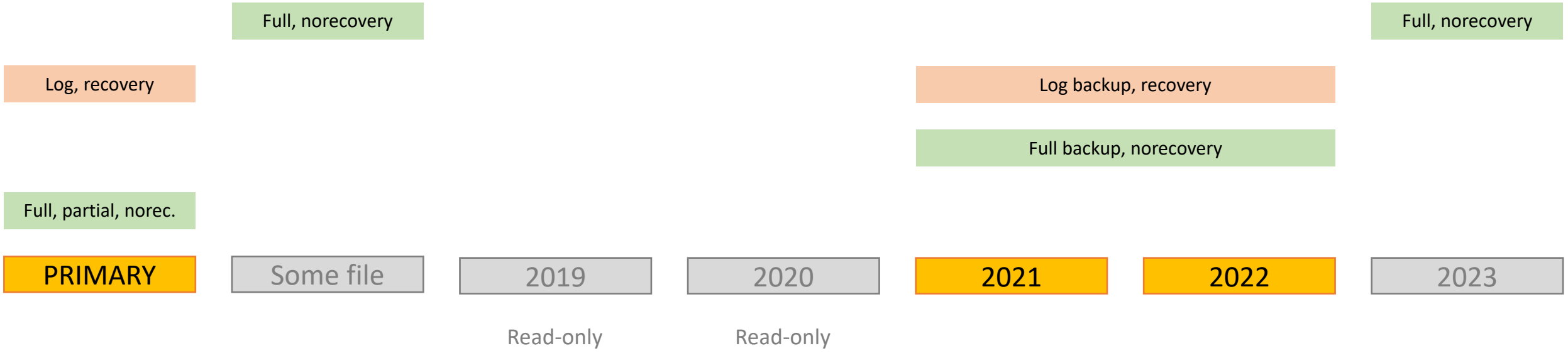
- An old FULL backup of the read-only filegroups
- A recent FULL backup of the read-write filegroups
- Transaction log backup(s) for the read-write filegroups



Piecemeal restore (full + log)

We have:

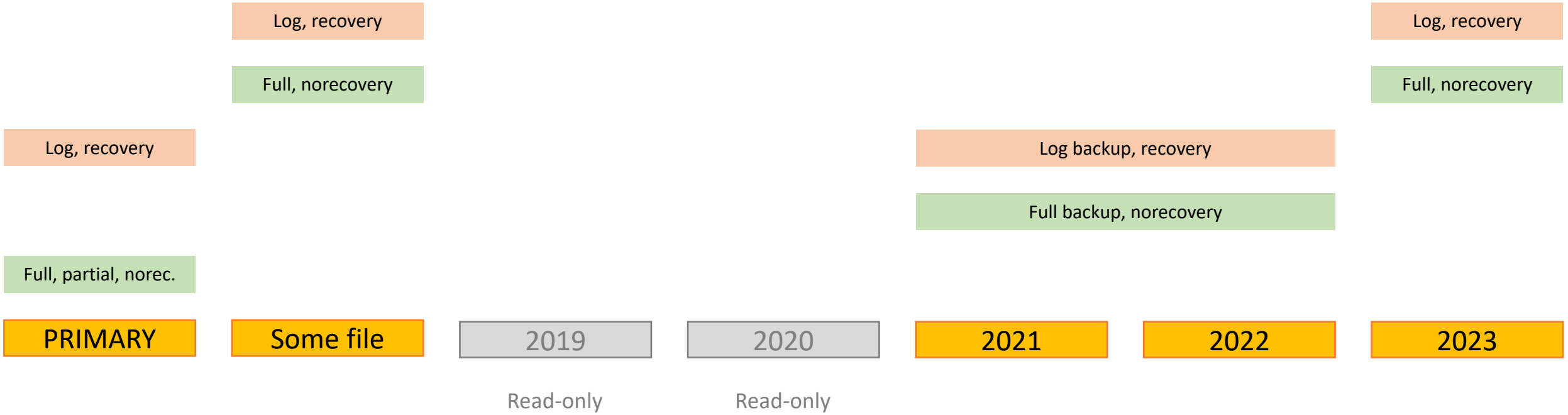
- An old FULL backup of the read-only filegroups
- A recent FULL backup of the read-write filegroups
- Transaction log backup(s) for the read-write filegroups



Piecemeal restore (full + log)

We have:

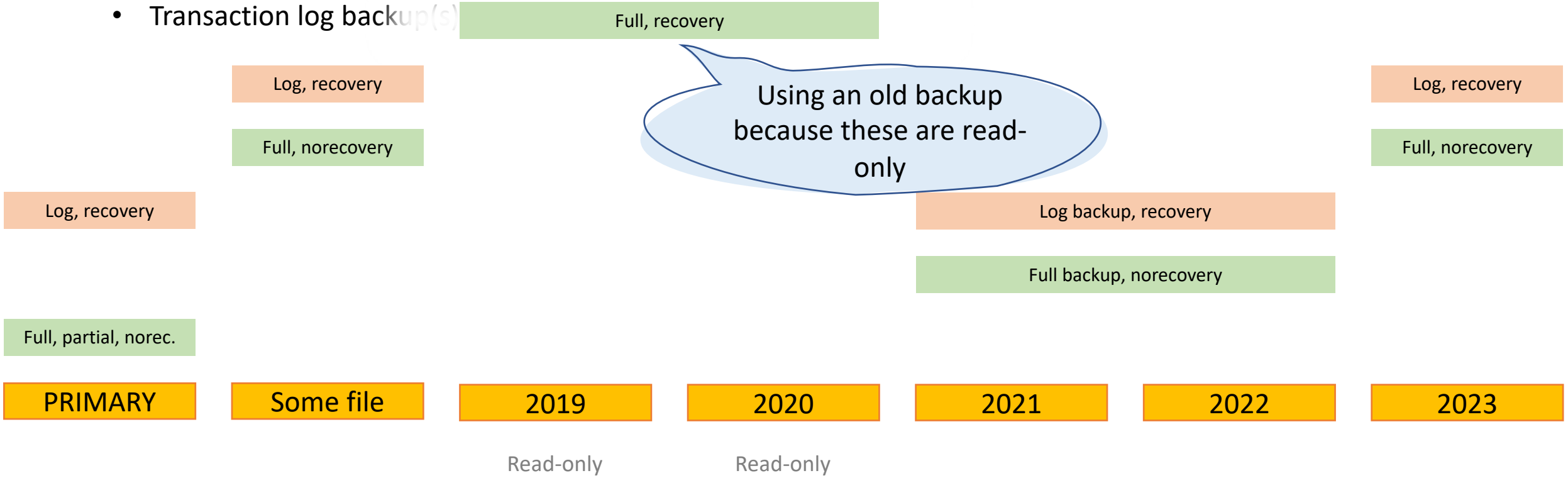
- An old FULL backup of the read-only filegroups
- A recent FULL backup of the read-write filegroups
- Transaction log backup(s) for the read-write filegroups



Piecemeal restore (full + log)

We have:

- An old FULL backup of the read-only filegroups
- A recent FULL backup of the read-write filegroups
- Transaction log backup(s)



DEMO

Seeks & scans work differently

```
SELECT AccountID, SUM(Amount) AS Net_balance_change
FROM dbo.AccountTransactions
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
GROUP BY AccountID;
```

Seeks & scans work differently

```
SELECT AccountID, SUM(Amount) AS Net_balance_change
FROM dbo.AccountTransactions
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
GROUP BY AccountID;
```

AccountID
TransactionDate
TransactionID

Seeks & scans work differently

```
SELECT AccountID, SUM(Amount) AS Net_balance_change
FROM dbo.AccountTransactions
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
GROUP BY AccountID;
```

AccountID
TransactionDate
TransactionID

YEAR(TransactionDate)
AccountID
TransactionDate
TransactionID

Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

- Seek AccountID 810000000-820000000

Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

AccountID
TransactionDate
TransactionID

YEAR(TransactionDate)
AccountID
TransactionDate
TransactionID

29 600 pages

- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2022-06

Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2022-06
- Aggregate on AccountID

Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

AccountID
TransactionDate
TransactionID

29 600 pages

- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2022-06
- Aggregate on AccountID

YEAR(TransactionDate)
AccountID
TransactionDate
TransactionID

1 800 pages

- For partition 2021
- Seek AccountID 810000000-820000000

Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

AccountID
TransactionDate
TransactionID

29 600 pages

- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2022-06
- Aggregate on AccountID

YEAR(TransactionDate)
AccountID
TransactionDate
TransactionID

1 800 pages

- For partition 2021
- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2021-12

Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

AccountID
TransactionDate
TransactionID

29 600 pages

- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2022-06
- Aggregate on AccountID

YEAR(TransactionDate)
AccountID
TransactionDate
TransactionID

1 800 pages

- For partition 2021
- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2021-12
- Aggregate on AccountID

Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

AccountID
TransactionDate
TransactionID

29 600 pages

- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2022-06
- Aggregate on AccountID

YEAR(TransactionDate)
AccountID
TransactionDate
TransactionID

1 800 pages

- For partition 2021
- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2021-12
- Aggregate on AccountID

600 pages

- For partition 2022
- Seek AccountID 810000000-820000000
- Filter TransactionDate 2022-01 – 2022-06
- Aggregate on AccountID

Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

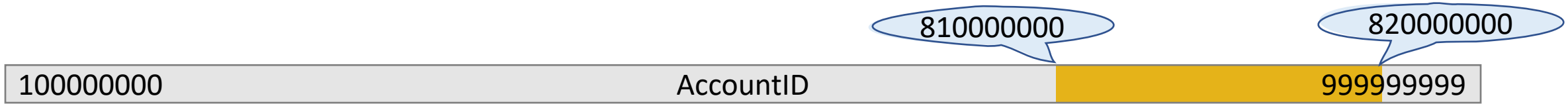
- Seek AccountID 810000000-820000000

100000000	AccountID	999999999

Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

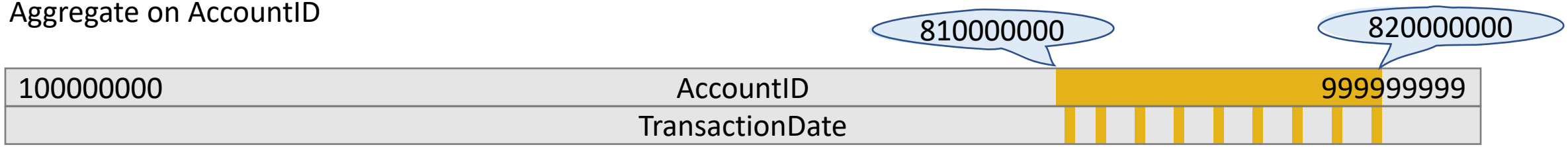
- Seek AccountID 810000000-820000000



Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

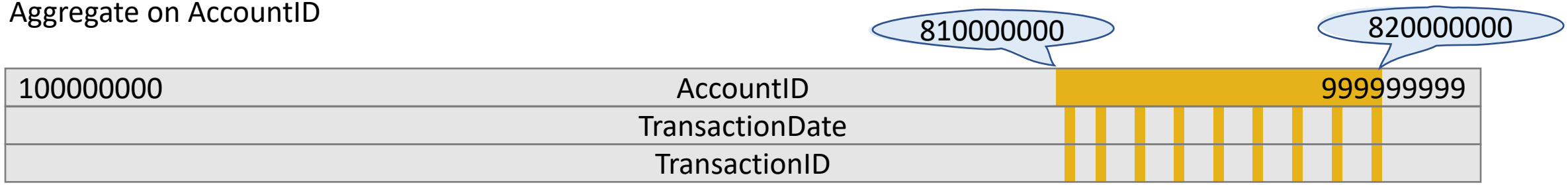
- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2022-06
- Aggregate on AccountID



Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

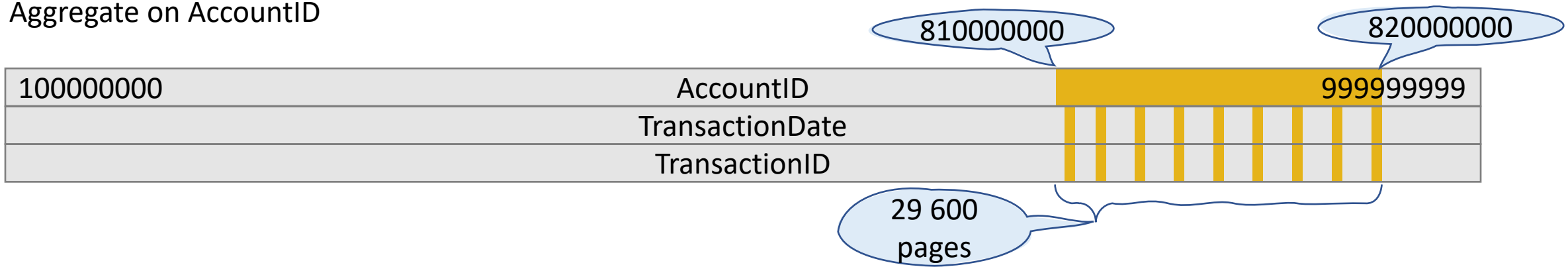
- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2022-06
- Aggregate on AccountID



Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

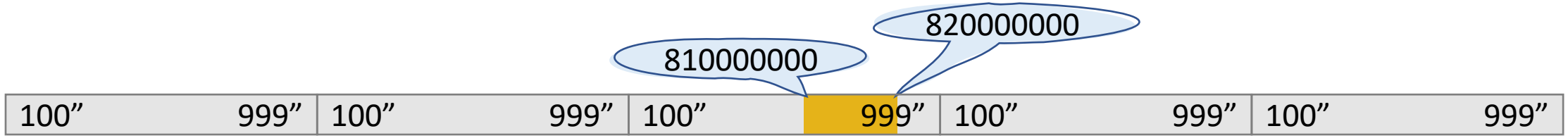
- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2022-06
- Aggregate on AccountID



Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

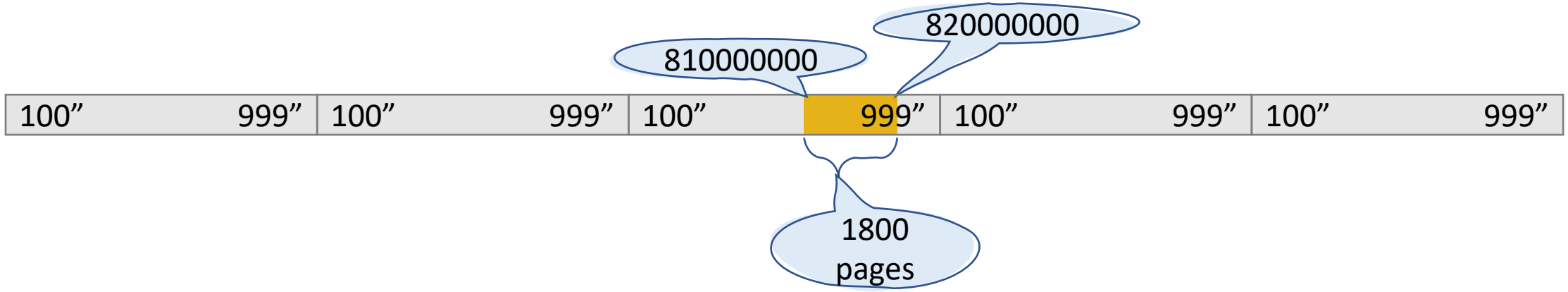
- For partition 2021
- Seek AccountID 810000000-820000000



Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

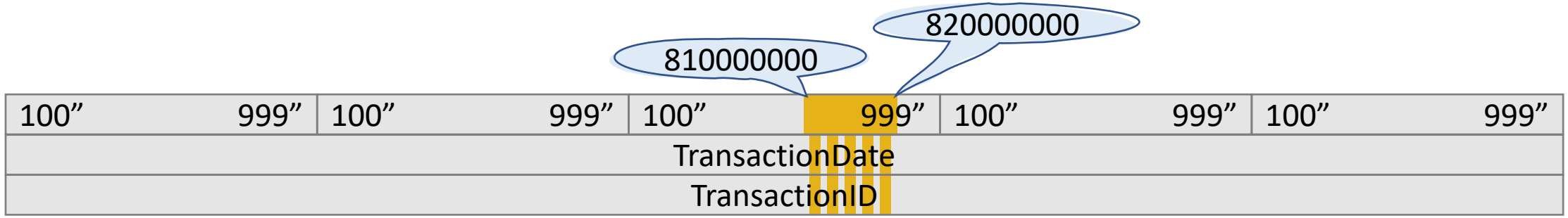
- For partition 2021
- Seek AccountID 810000000-820000000



Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

- For partition 2021
- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2021-12
- Aggregate on AccountID

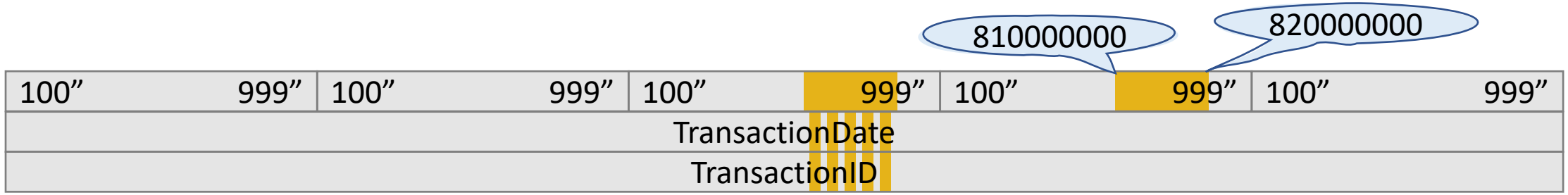


Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

- For partition 2021
- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2021-12
- Aggregate on AccountID

- For partition 2022
- Seek AccountID 810000000-820000000

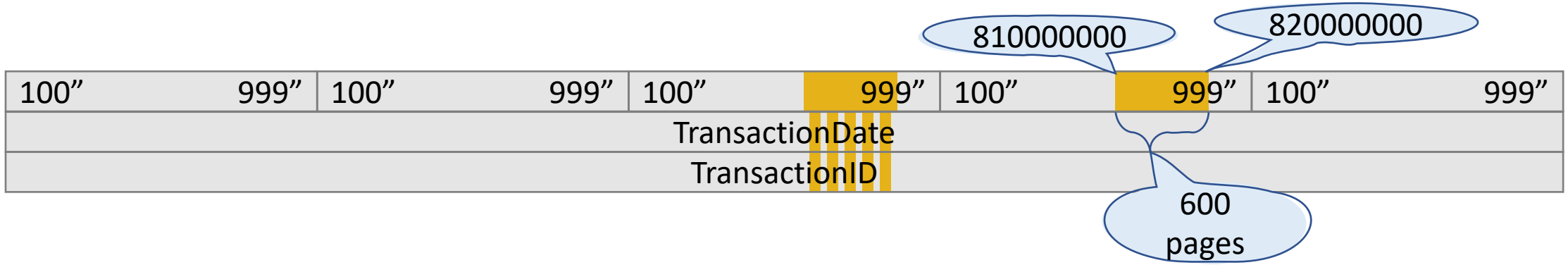


Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

- For partition 2021
- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2021-12
- Aggregate on AccountID

- For partition 2022
- Seek AccountID 810000000-820000000

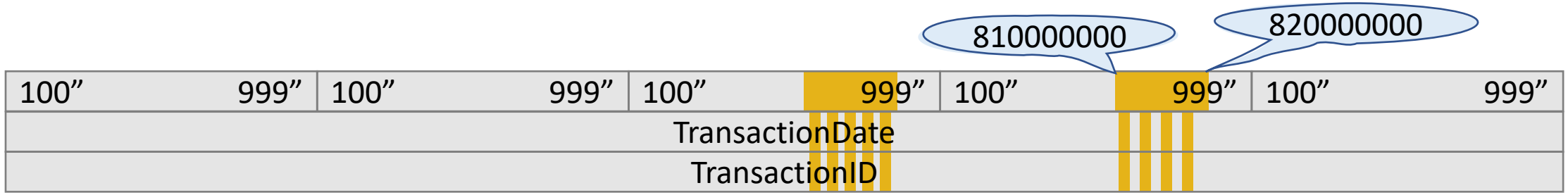


Seeks & scans work differently

```
WHERE TransactionDate BETWEEN '2021-07-01' AND '2022-06-30'
  AND AccountID BETWEEN 810000000 AND 820000000
```

- For partition 2021
- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2021-12
- Aggregate on AccountID

- For partition 2022
- Seek AccountID 810000000-820000000
- Filter TransactionDate 2021-07 – 2021-12
- Aggregate on AccountID




DEMO

Takeaway: Why partitioning?

- Some costly DML operations can be turned into super-fast DDL statements
- Some queries can be tuned to use a single partition (partition elimination)
- Can solve specific latch contention issues by spreading writes
- You can backup only read-write partitions – faster backups
- You can restore the important partitions first – shorter RTO

… or maybe not?

- More complex to maintain
- More complex to develop
- Tuned for non-partitioned ≠ tuned for partitioning

Standing on the shoulders of giants

- Kendra Little
- Cathrine Wilhelmsen
- Uwe Ricken

One more thing

1. Go to evals.datagrillen.com
2. Find my session in the list
3. Write something nice

Email: daniel@strd.co
Twitter: @dhmacher
Blog: sqlsunday.com