SUMMIT2017
PASS

Everything You Always Wanted to Know About Data Partitioning

..but were afraid to ask

Dmitri V. Korotkevitch

Please silence cell phones

Explore everything PASS has to offer

24 HOURS OF PASS
Free online webinar events

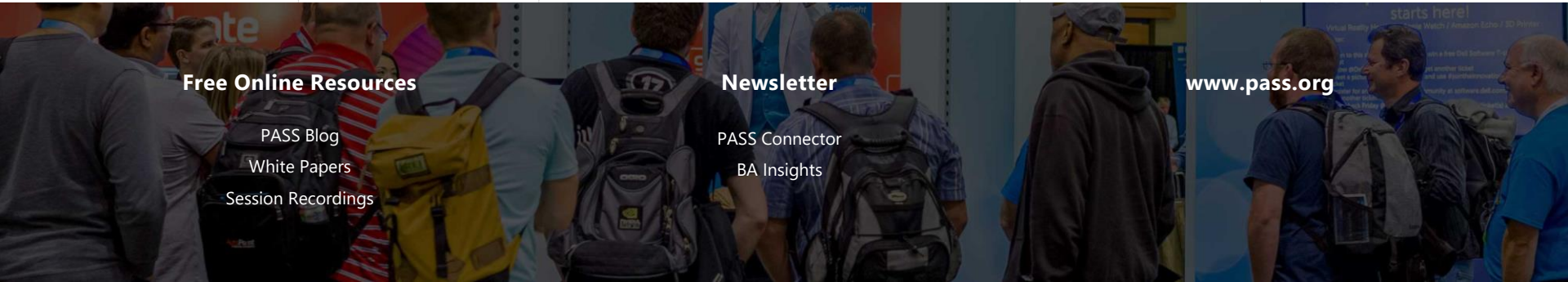
LOCAL GROUPS
Local user groups around the world

SQLSATURDAY PASS
Free 1-day local training events

VIRTUAL GROUPS
Online special interest user groups

BUSINESS ANALYTICS DAY PASS
Business analytics training

PASS VOLUNTEERS
Get involved



Free Online Resources

PASS Blog
White Papers
Session Recordings

Newsletter

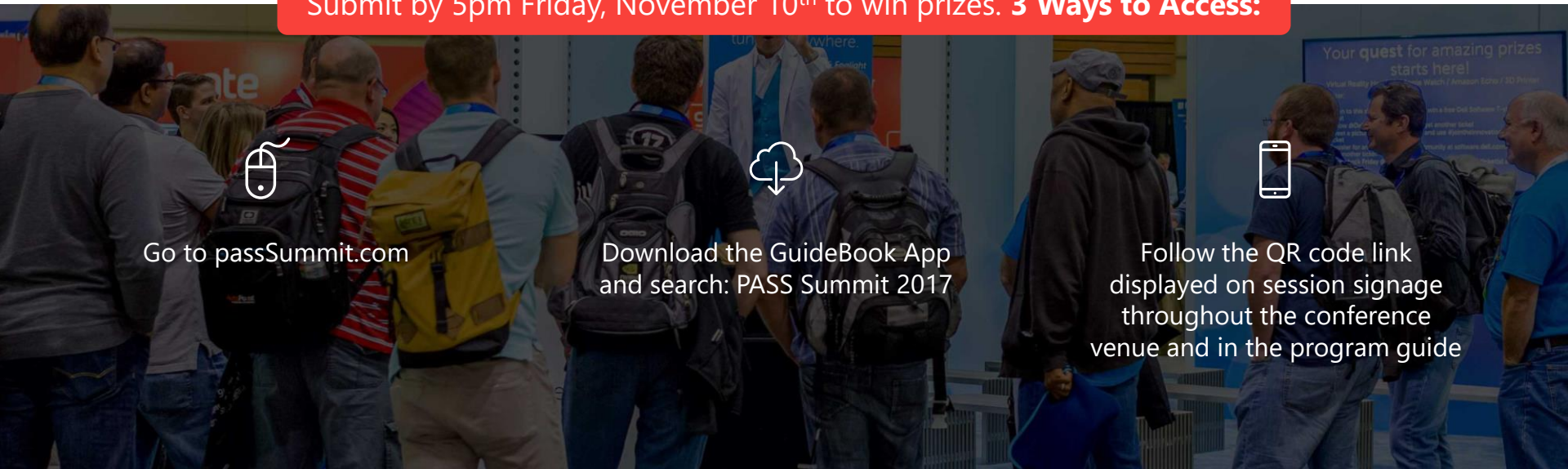
PASS Connector
BA Insights

www.pass.org

Session evaluations

Your feedback is important and valuable.

Submit by 5pm Friday, November 10th to win prizes. **3 Ways to Access:**



Go to passSummit.com

Download the GuideBook App and search: PASS Summit 2017

Follow the QR code link displayed on session signage throughout the conference venue and in the program guide

Dmitri Korotkevitch

Principal DB Engineer @ chewy.com

http://aboutsqlserver.com dk@aboutsqlserver.com

@aboutsqlserver

20+ years in IT

15+ years working with SQL Server

Microsoft Data Platform MVP

Microsoft Certified Master

Author:

- Pro SQL Server Internals
- Expert SQL Server In-Memory OLTP

Agenda

Implementing Data Partitioning in SQL Server

- Architecture
- Implementation
- Common Tasks

Why do we partition the data?

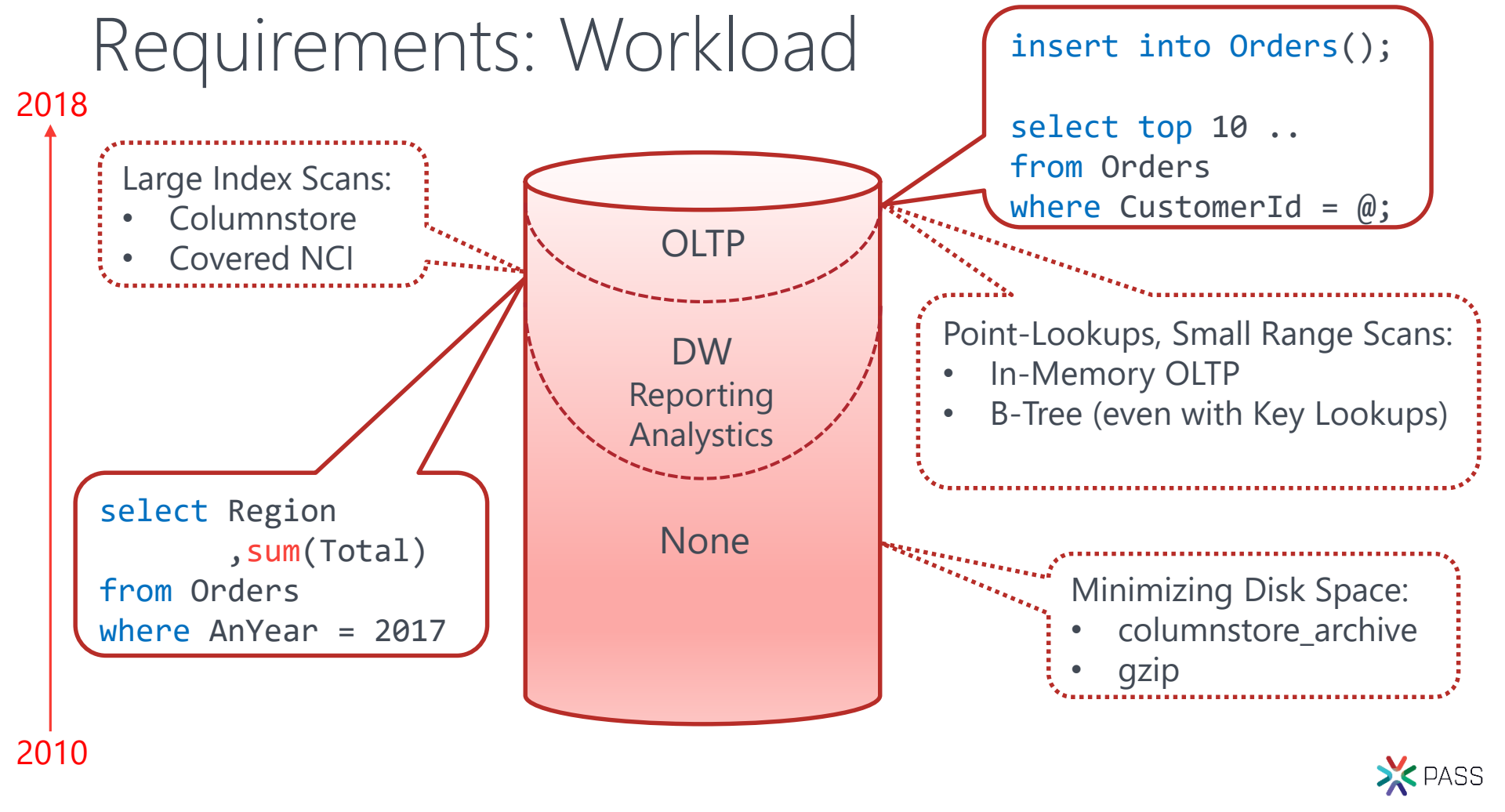
We want to separate data with different requirements

Workload / Performance

Availability

Maintenance

Requirements: Workload



Partitioned Views

```
create table dbo.Orders2017_11
(
    OrderDate datetime2(0) not null,
    OrderId int identity(1,1) not null,
    WarehouseId int not null,
    DeliveryInstructions varchar(1000) null,
    index O11 unique clustered(OrderDate,OrderId) on [CurrentOrders],
    check(OrderDate >= '2017-11-01' and OrderDate < '2017-12-01')
);
create table dbo.Orders2017_10
(
    OrderDate datetime2(0) not null,
    OrderId int identity(1,1) not null,
    WarehouseId int not null,
    DeliveryInstructions varchar(1000) null,
    index O10 unique clustered(OrderDate,OrderId) on [RecentOrders],
    index O10_NCI nonclustered(WarehouseId) with (data_compression=row),
    check(OrderDate >= '2017-10-01' and OrderDate < '2017-11-01')
);
create table dbo.Orders2017
(
    OrderDate datetime2(0) not null,
    OrderId int identity(1,1) not null,
    WarehouseId int not null,
    index O2017 clustered columnstore on [OldOrders],
    check(OrderDate >= '2017-01-01' and OrderDate < '2017-10-01')
);
```

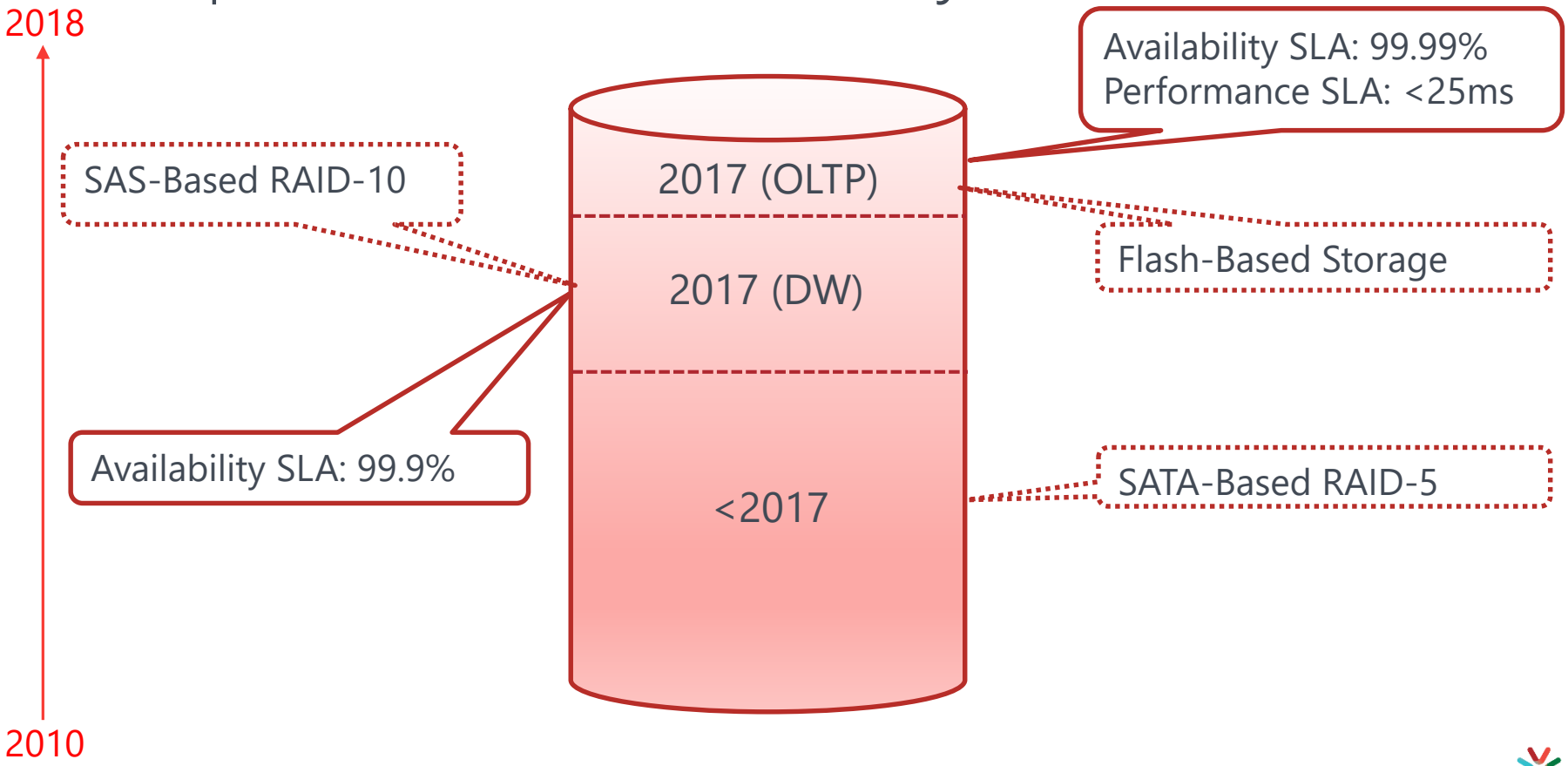
```
create view dbo.Orders
as
    select
        OrderDate
        ,OrderId
        ,WarehouseId
        ,DeliveryInstructions
    from dbo.Orders2017_11

    union all

    select
        OrderDate
        ,OrderId
        ,WarehouseId
        ,DeliveryInstructions
    from dbo.Orders2017_10

    union all

    select
        OrderDate
        ,OrderId
        ,WarehouseId
        ,NULL as DeliveryInstructions
    from dbo.Orders2017
```

PASS

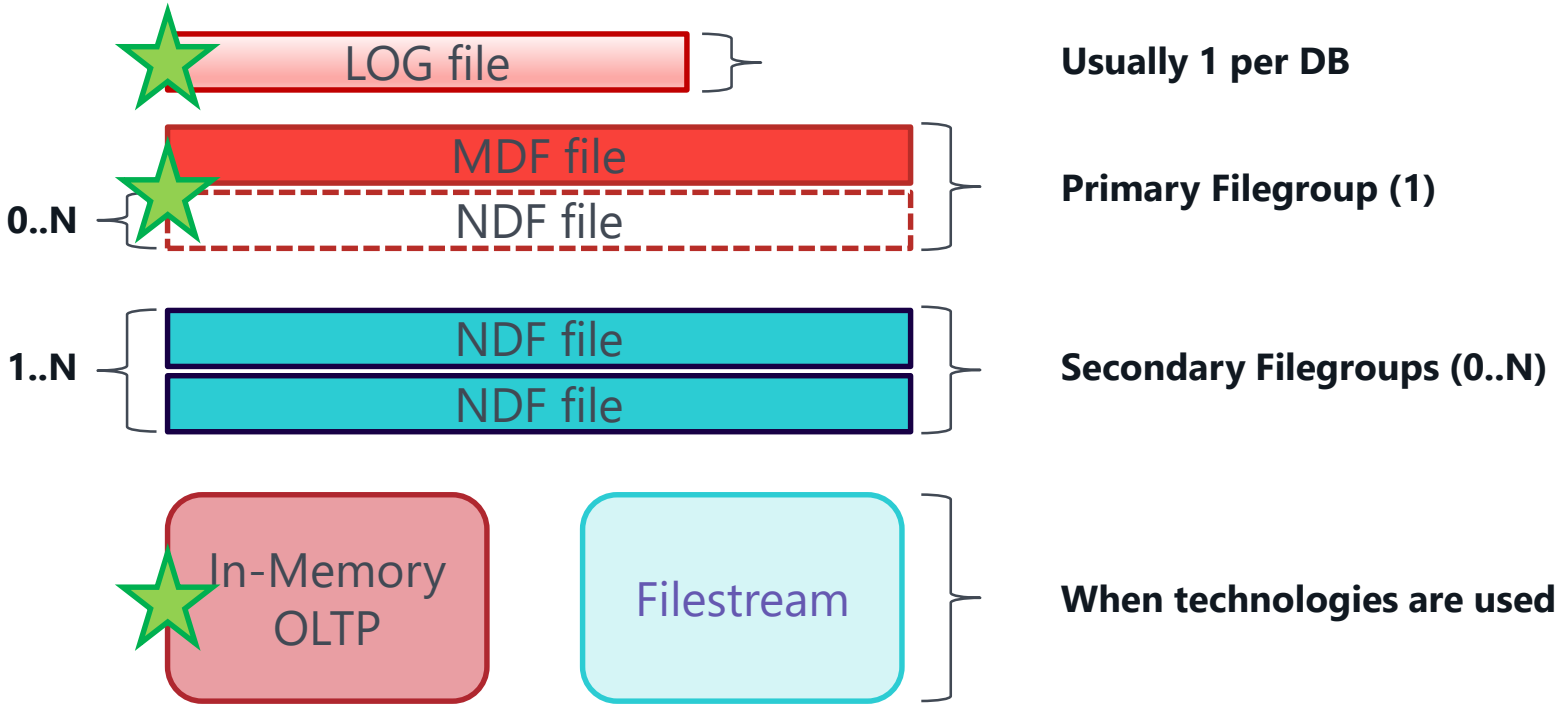
Partitioned Views

Demo

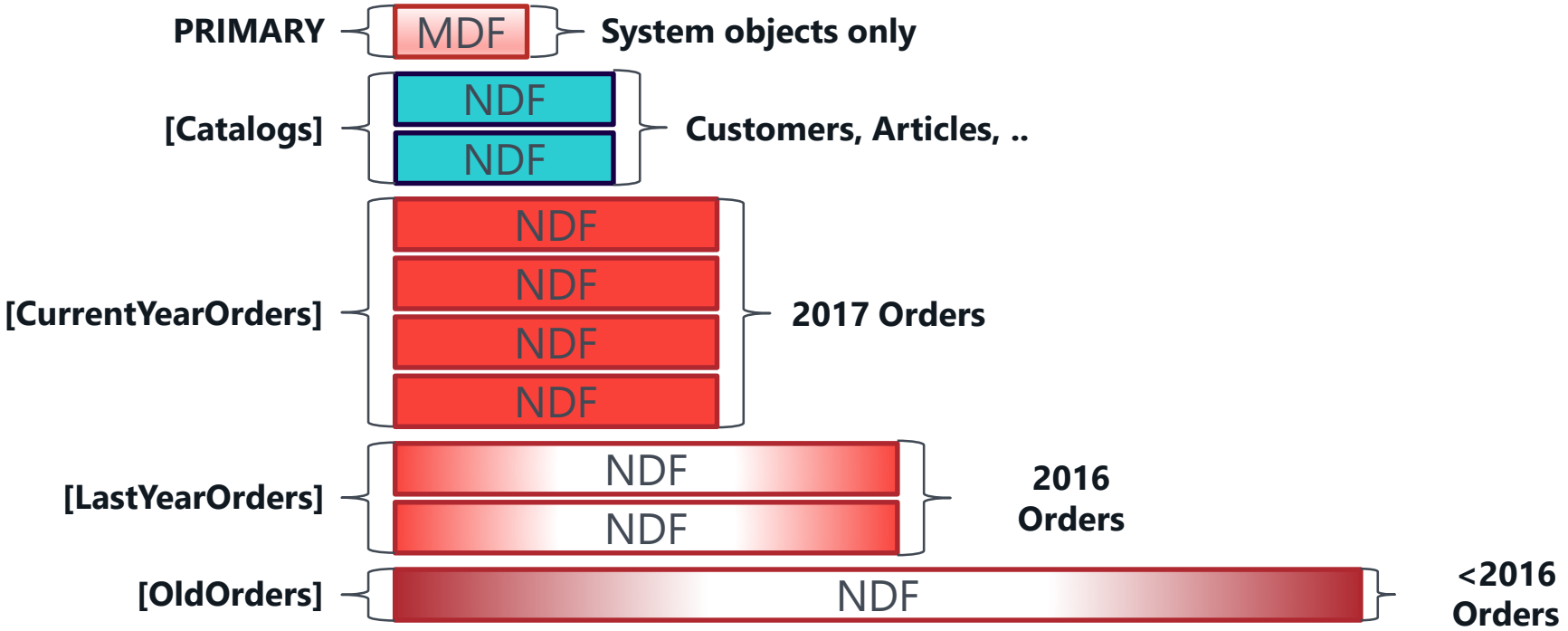
Requirements: Availability



Database Structure



Partitioning-Friendly Database Architecture

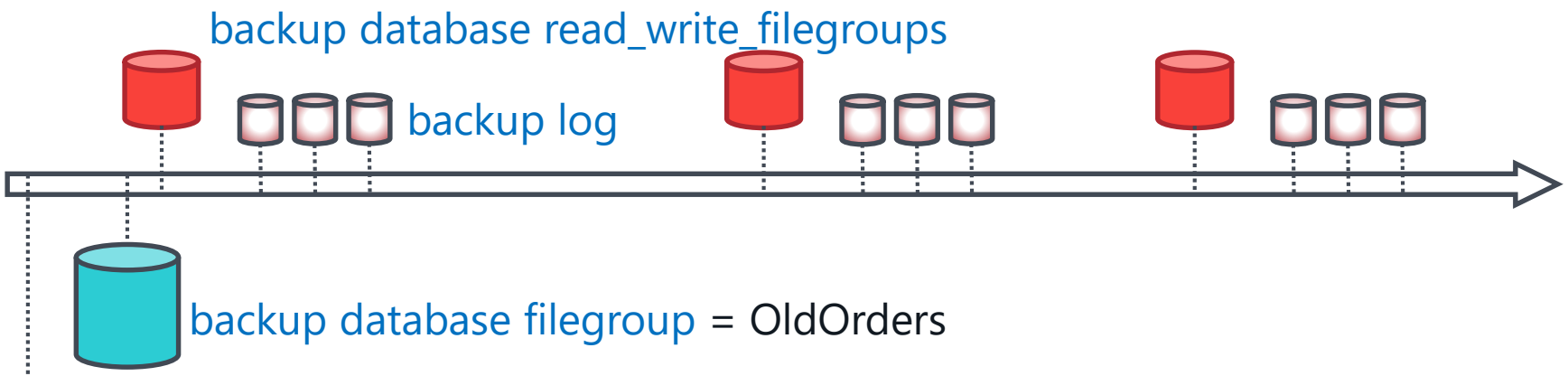


Partial Database Backup: Backup

PRIMARY | [Catalogs] | [CurrentYearOrders] | [LastYearOrders]

[OldOrders]

Read-Only Data

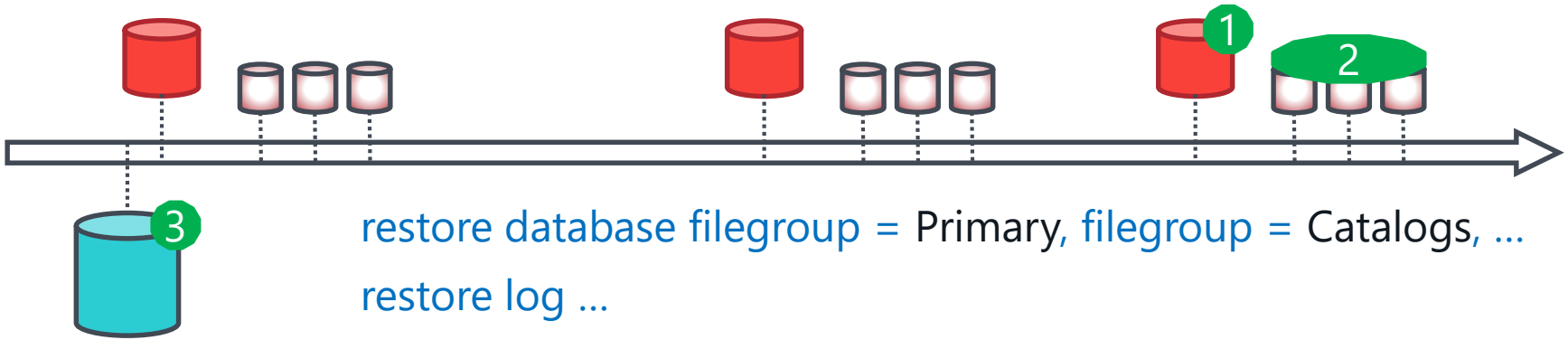


Partial Database Backup: Restore

PRIMARY | [Catalogs] | [CurrentYearOrders] | [LastYearOrders]

[OldOrders]

Read-Only Data



restore database filegroup = Primary, filegroup = Catalogs, …

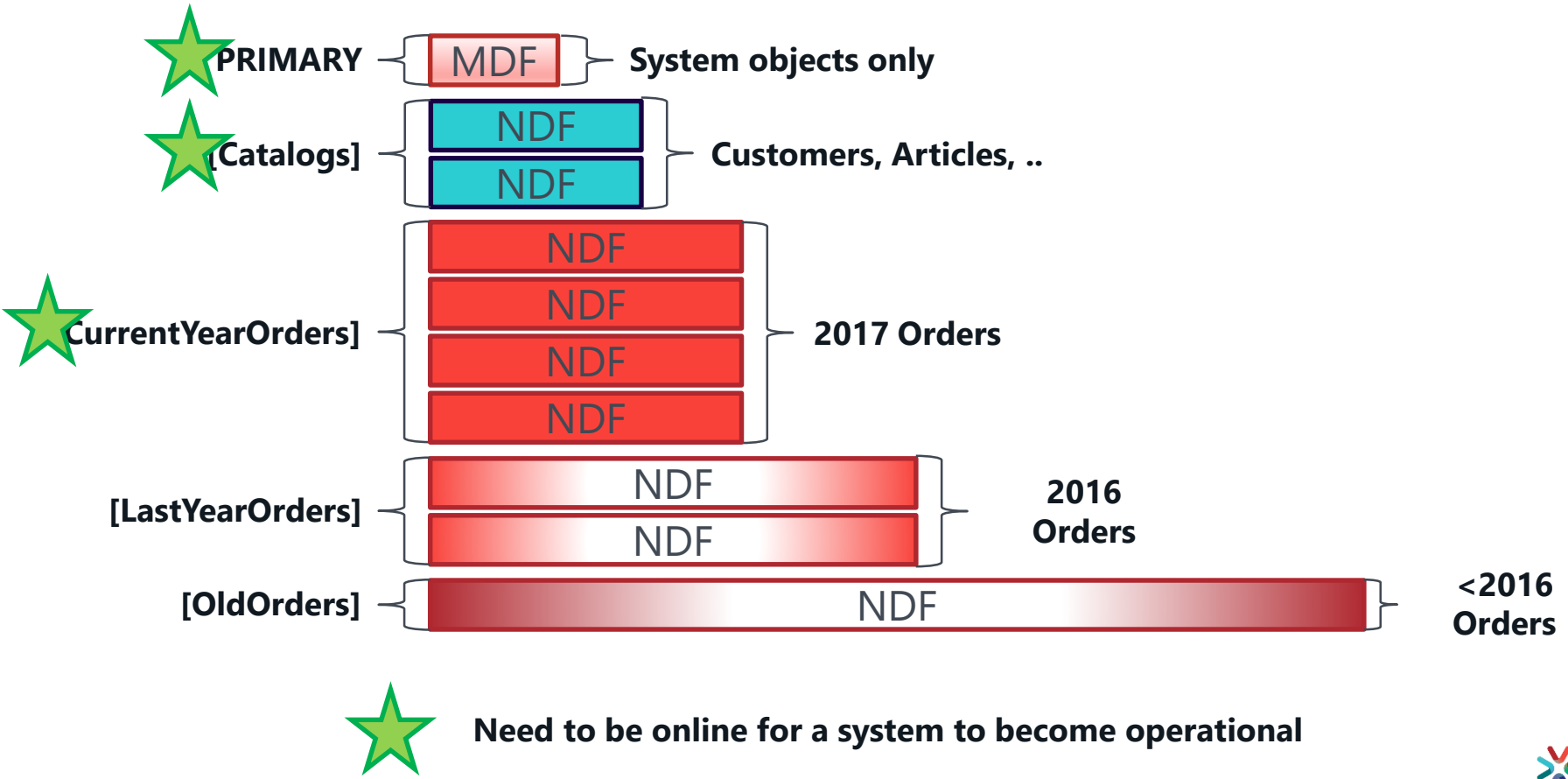
restore log …

restore database filegroup = OldOrders …

Partial Database Backup

Demo

Partitioning-Friendly Database Architecture



Piecemeal Restore

Demo

Design Considerations

Keep PRIMARY filegroup empty

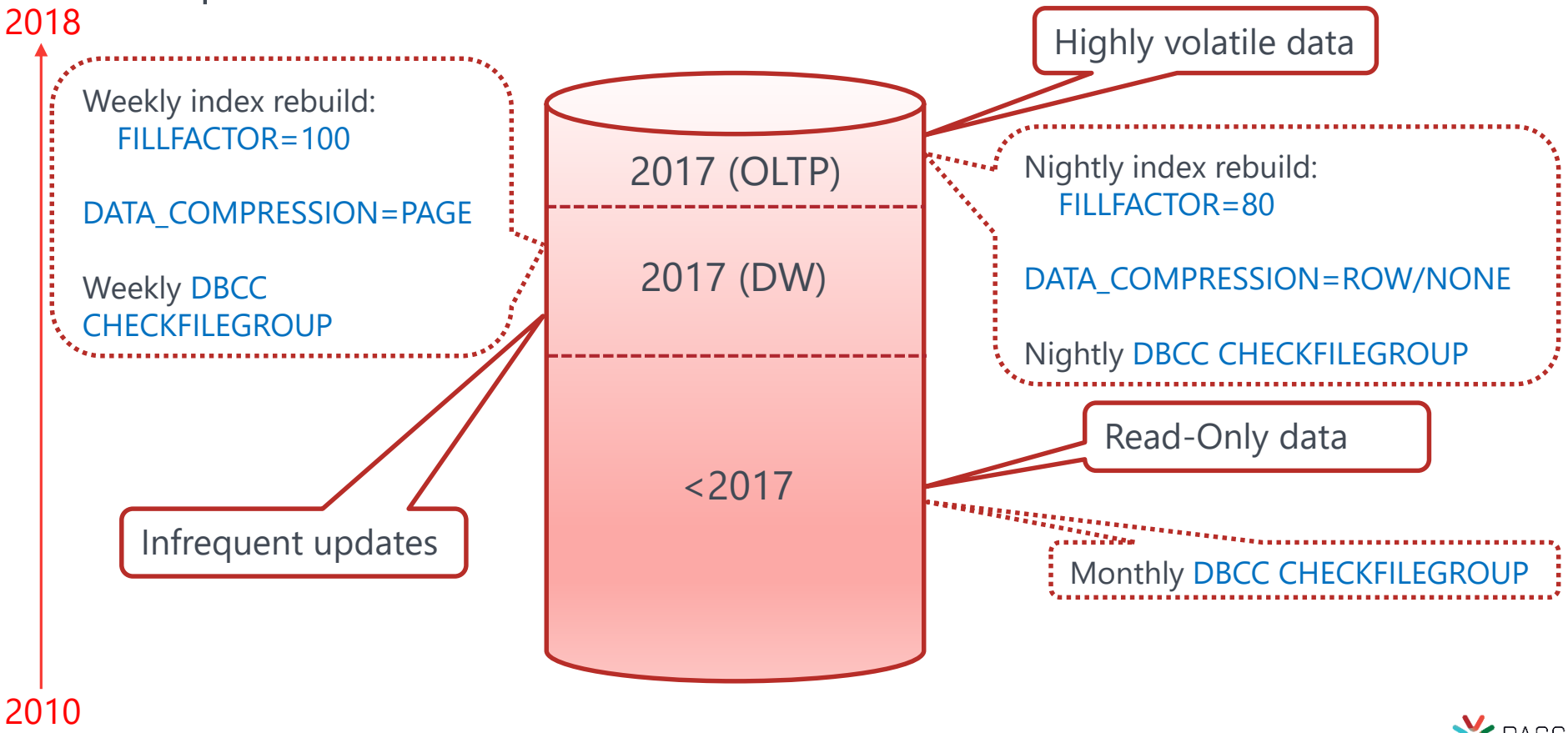
Keep related entities together (e.g. Orders, OrderLineItems)

Keep all table objects (indexes, LOB data) together

Create multiple data files for volatile data

Put LARGE read-only tables to separate filegroup(s)

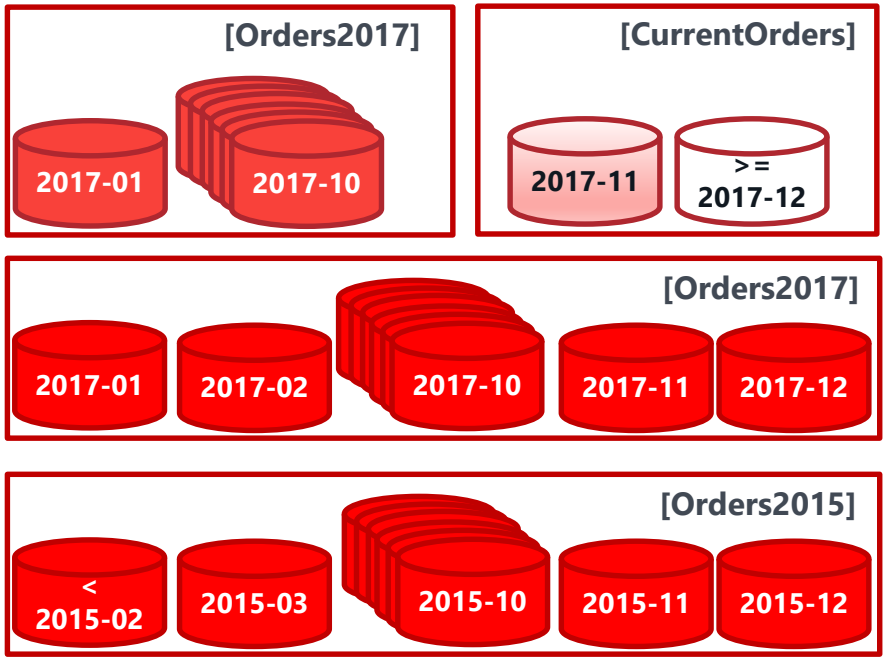
Requirements: Maintenance



Index Maintenance

Demo

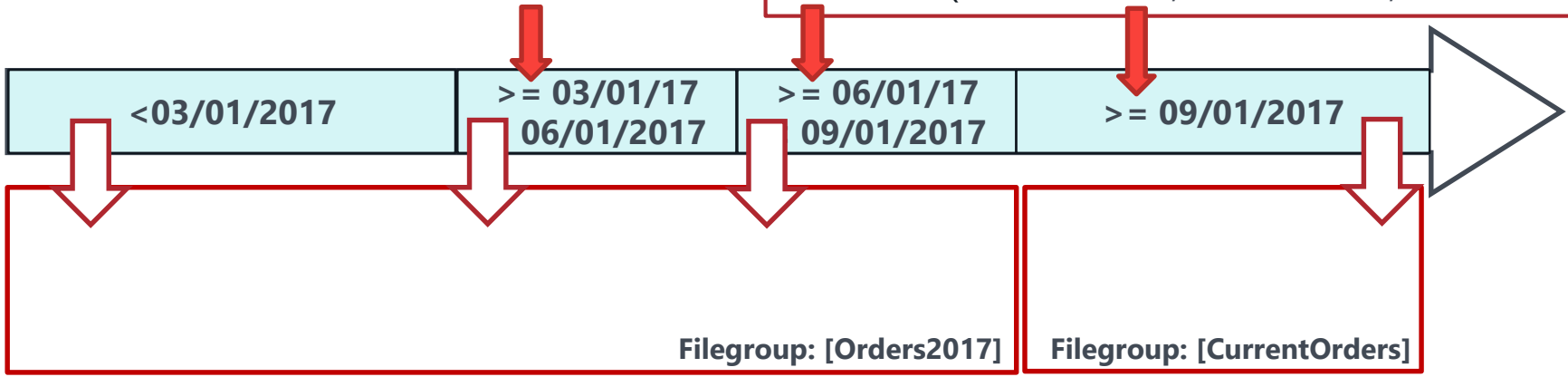
Partitioned Tables




```
create partition function pfOrders(datetime2(0))
as range right
for values('2015-02-01', '2015-03-01',
/*...*/ '2016-01-01','2016-02-01','2016-03-01',
/*...*/ '2017-01-01','2017-02-01','2017-03-01',
/*...*/ '2017-10-01','2017-11-01','2017-12-01');

create partition scheme psOrders
as partition pfOrders
to ([Orders2015],[Orders2015],
/*...*/ [Orders2016],[Orders2016],[Orders2016],
/*...*/ [Orders2017],[Orders2017],[Orders2017],
/*...*/ [CurrentOrders],[CurrentOrders]);

create table dbo.OrdersPT
(
    OrderId int not null,
    OrderDate datetime2(0) not null,
    OrderNum varchar(32) not null,
    Amount money not null,
    /* Other Columns */
);

create unique clustered index IDX_OrdersPT_OrderDate_OrderId
on dbo.OrdersPT(OrderDate, OrderId)
with
(
    data_compression = page on partitions(1 to 24),
    data_compression = page on partitions(25 to 31),
    data_compression = none on partitions(32 to 36)
)
on psOrders(OrderDate);
```

Partitioning Objects

```
create partition function pfOrders(datetime2(0))
as range right
for values('2017-03-01','2017-06-01','2017-09-01')
```




```
create ind          .Orders(...)
on psOrders(OrderDate)
```

>= 2017-09-01

```
create index
on dbo.OrderLineItems(...)
on psOrders(OrderDate)
```

>= 2017-09-01

```
create partition scheme psOrders
as partition pfOrders
to ([Orders2017],[Orders2017],
[Orders2017],[CurrentOrders])
```

PASS

Partitioned Tables

Demo

Partitioned Tables: Partition Switch

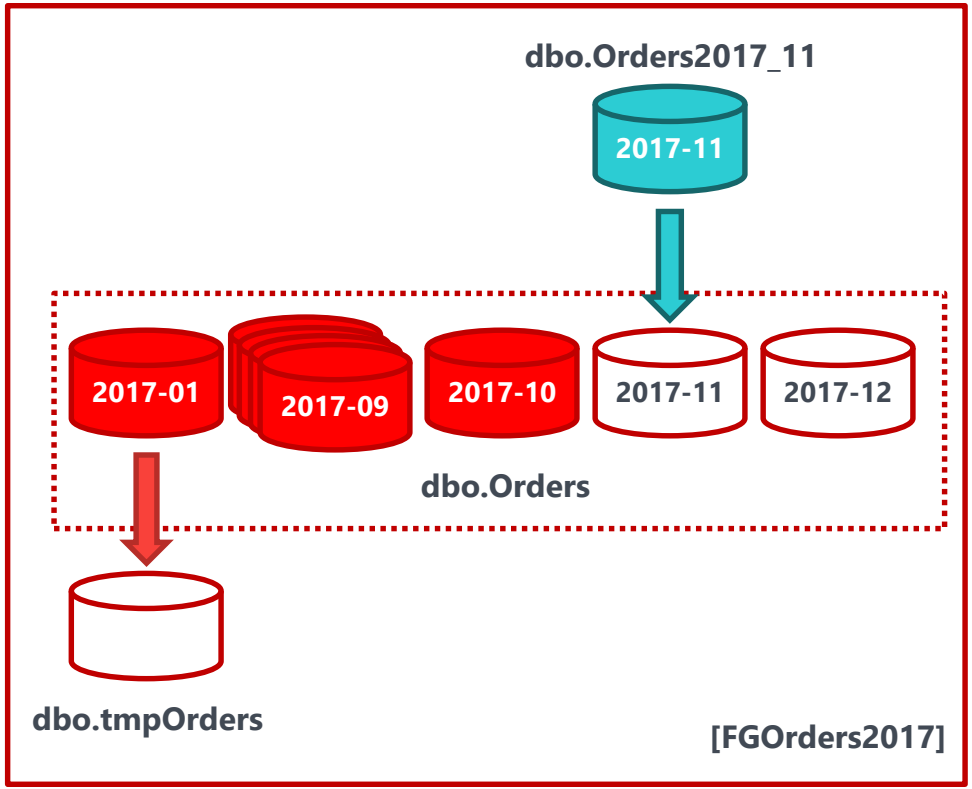
Each partition is the separate data structure

You can switch partitions to and from partitioned table when:

Source and destination are identical and reside on the same filegroup

All indexes are aligned

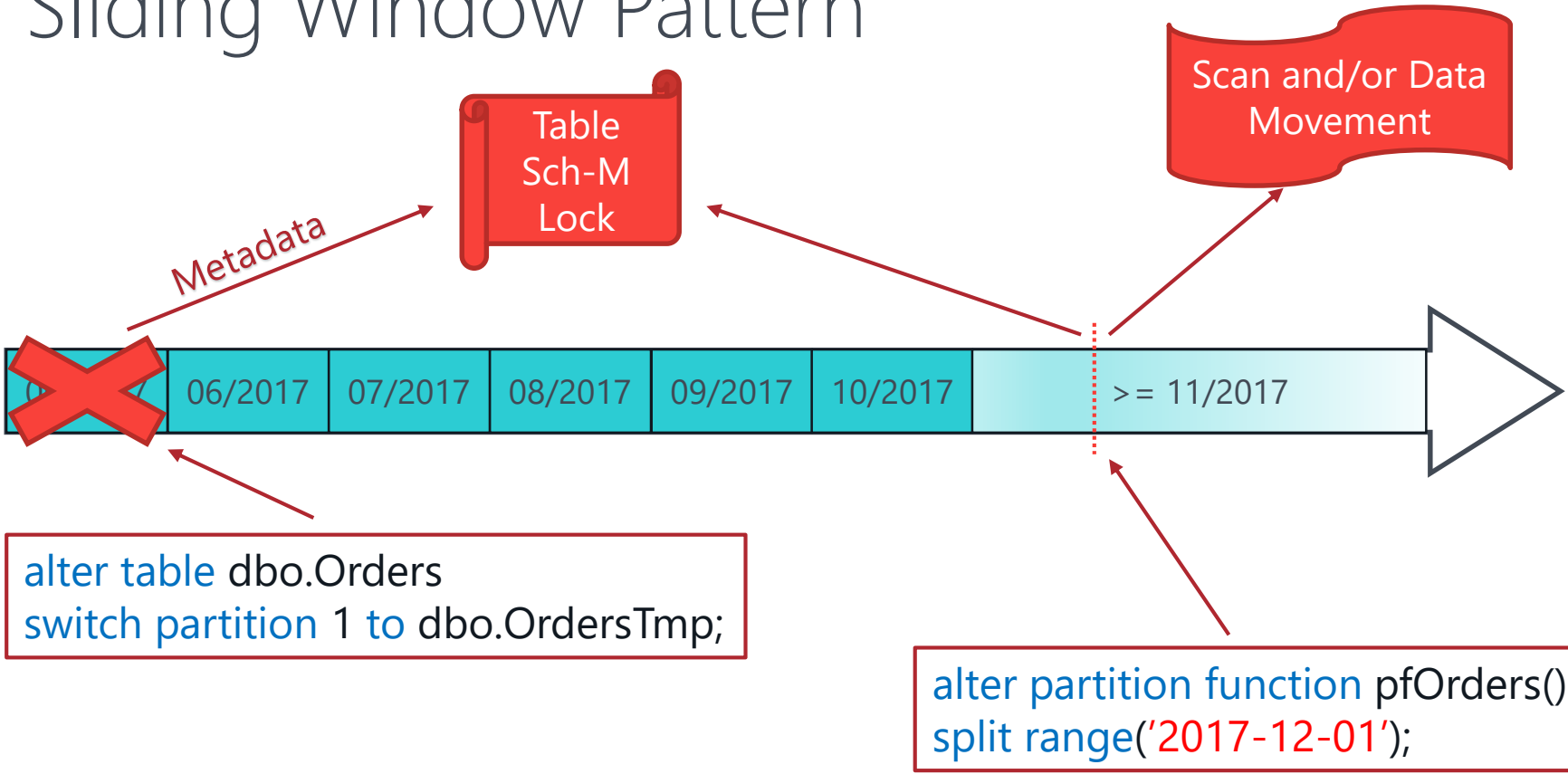
Destination table/partition is empty



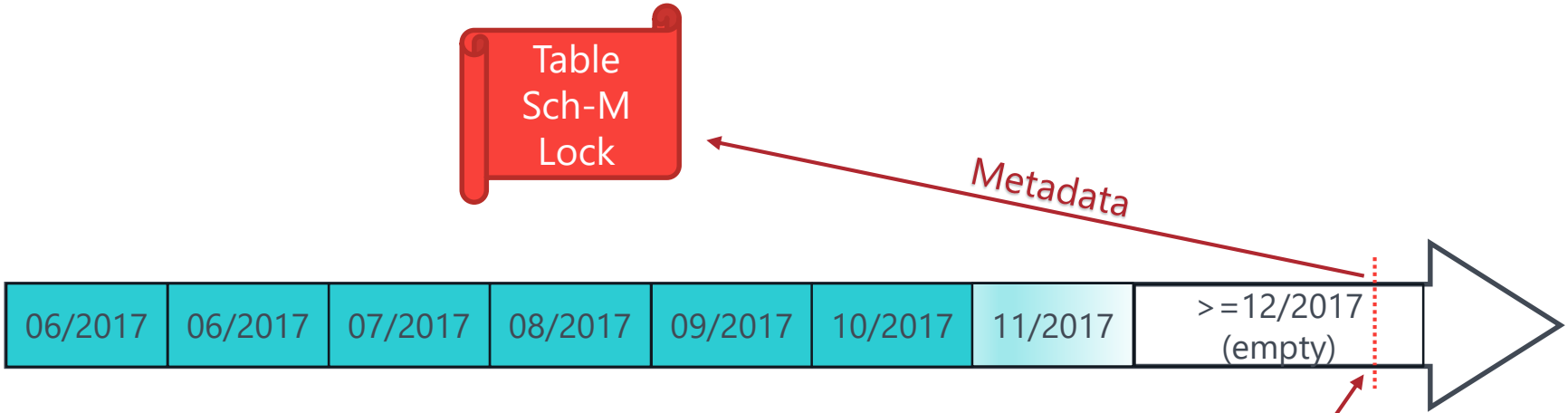
Partition Switch: Import Data

Demo

Sliding Window Pattern



Sliding Window Pattern




```
alter table dbo.Orders
switch partition 1 to dbo.OrdersTmp;
```

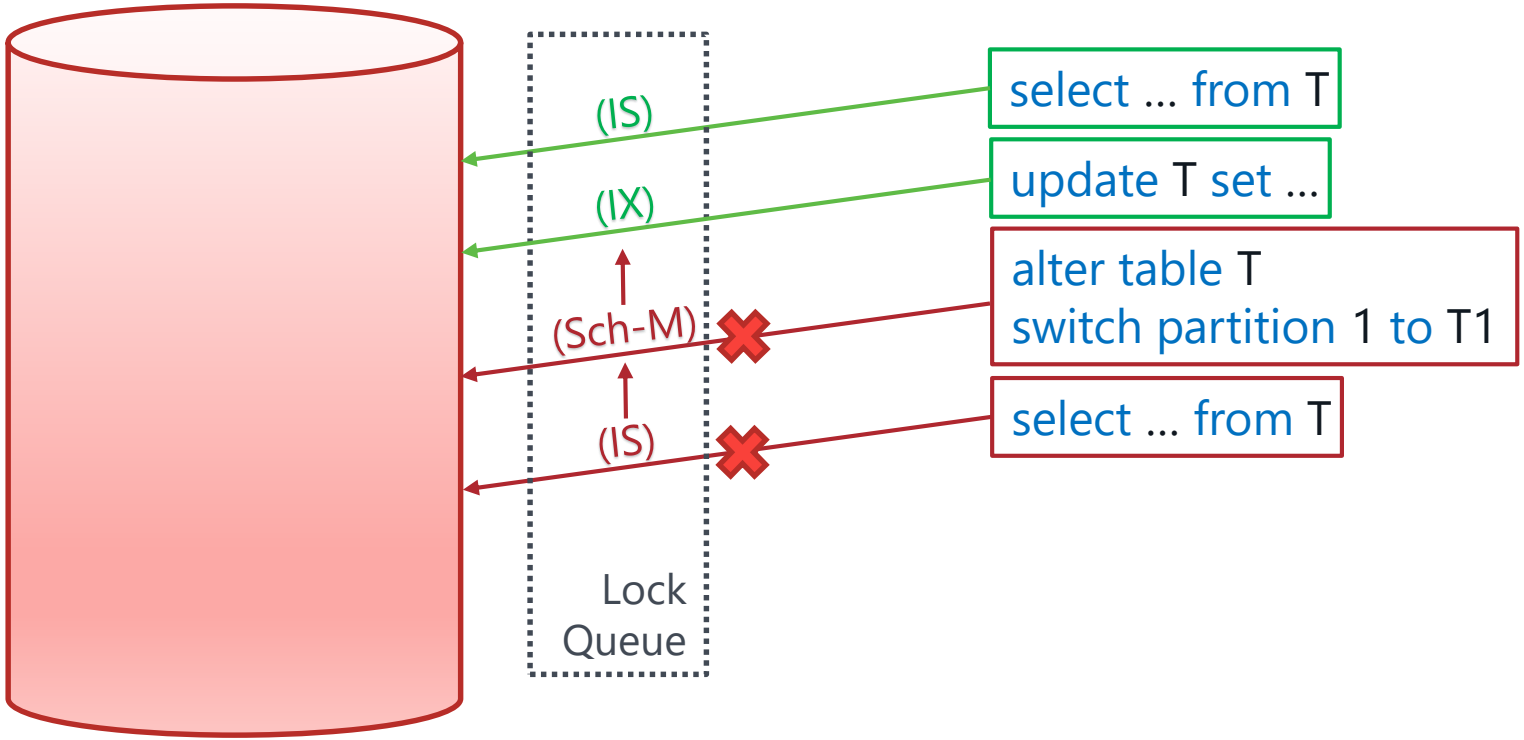
```
alter partition function pfOrders()
split range('2018-01-01');
```

PASS

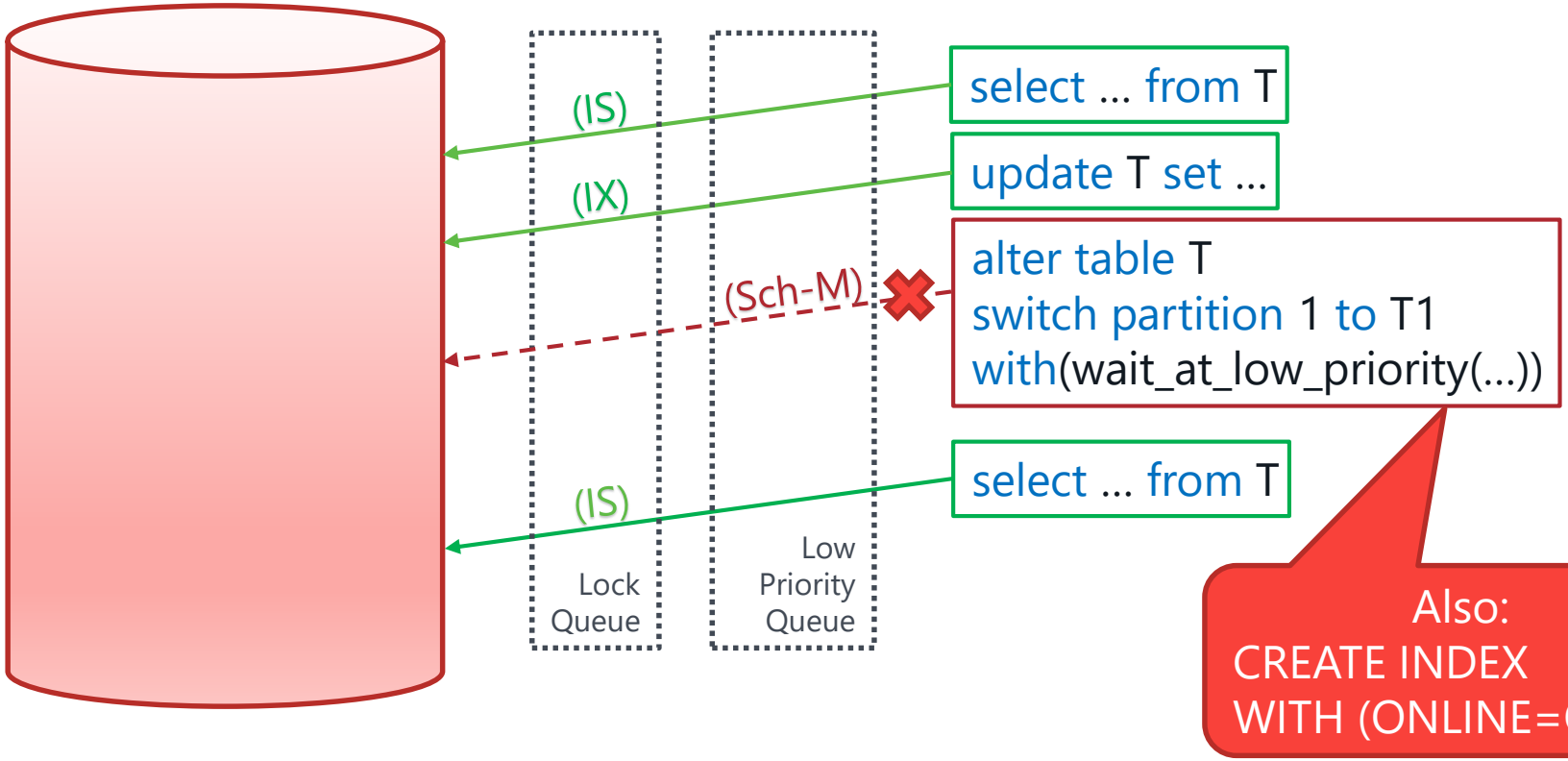
Sliding Window Pattern

Demo

Low Priority Locks (SQL Server 2014+)



Low Priority Locks (SQL Server 2015+)



Low Priority Locks

Demo

Partitioned Tables vs. Partitioned Views

Partitioned Views

Available in all versions and editions

Maximum 255 tables

Each table may have its own schema

Each table has its own statistics

Each table may have its own compression

May use different technologies (B-Tree, In-Memory OLTP, Columnstore) on per-table basis

Online index rebuild on per-table basis (*EE)

Online data movement between FG (*EE)

Significant development / administration overhead

Partitioned Tables

Enterprise Edition feature prior SS2016 SP1

Maximum 15,000 partitions

One schema and indexes across all partitions

One statistics across all partitions

Each partition may have its own compression

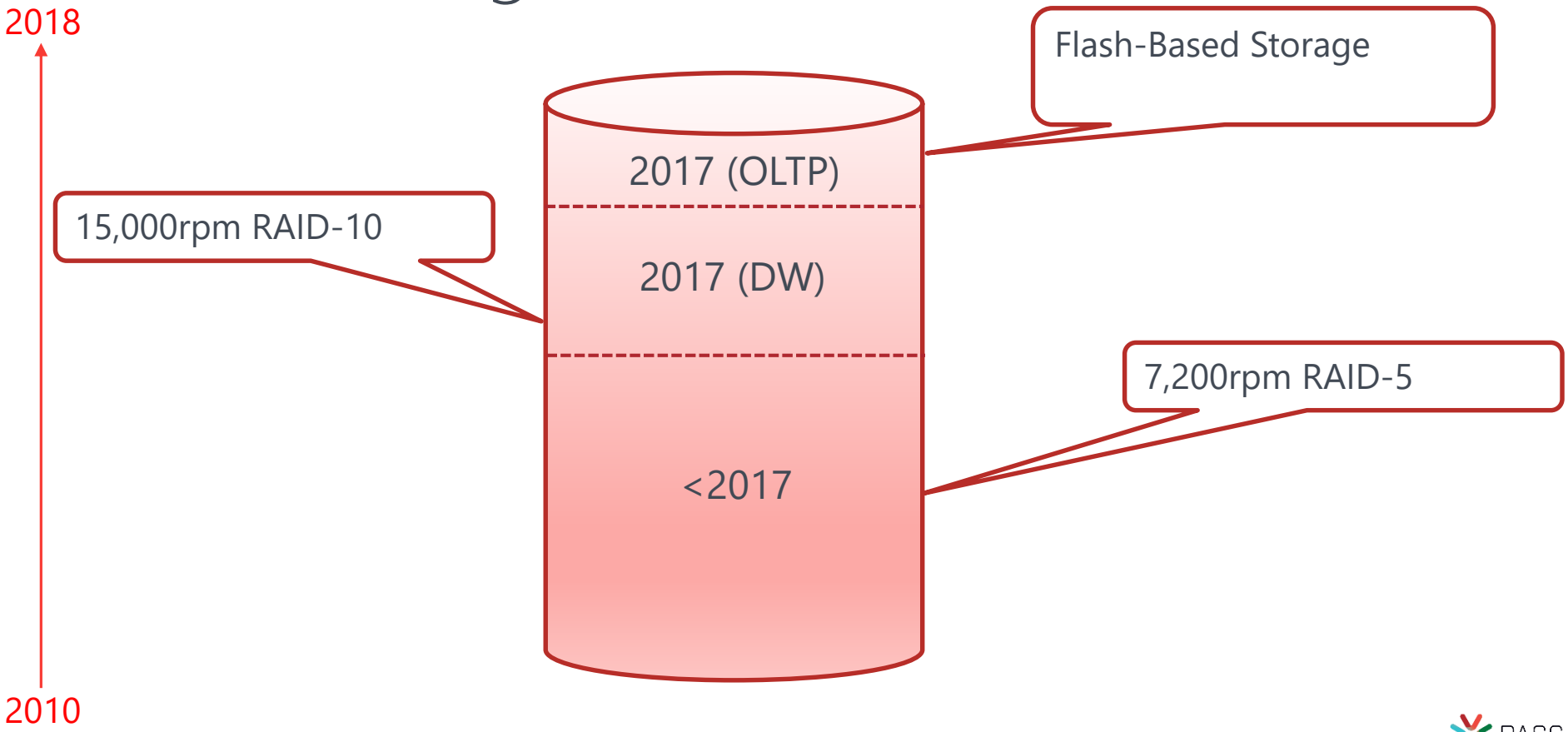
One technology across all partitions (B-Tree or Columnstore)

Online partition index rebuild in SS 2014+

Online data movement requires development

Low development / administration overhead

Tiered Storage



Moving Data

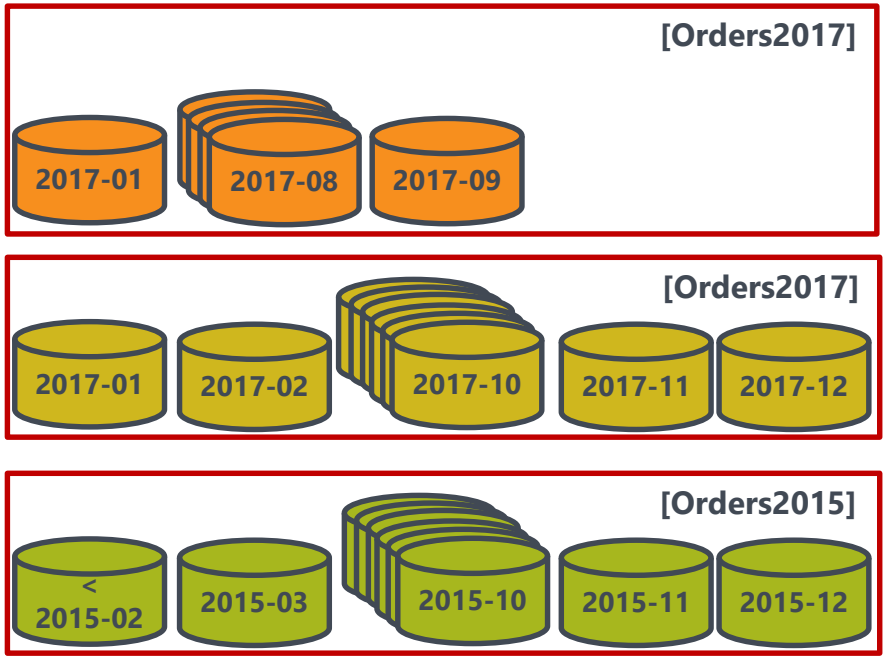
Demo

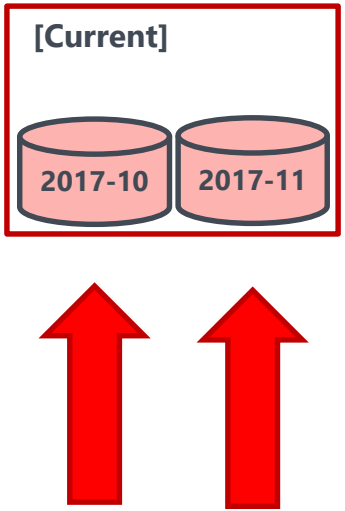
Moving Data

	Online data movement to different filegroup	Online index rebuild with LOB data	Online index rebuild without LOB data	Moving data to different disk array
SQL Server 2005 – 2008R2 Enterprise Edition	Requires development (*)	Not supported	Supported	Supported (Introduce Fragmentation)
SQL Server 2012+ Enterprise Edition		Supported		
Non-Enterprise Editions	N/A			

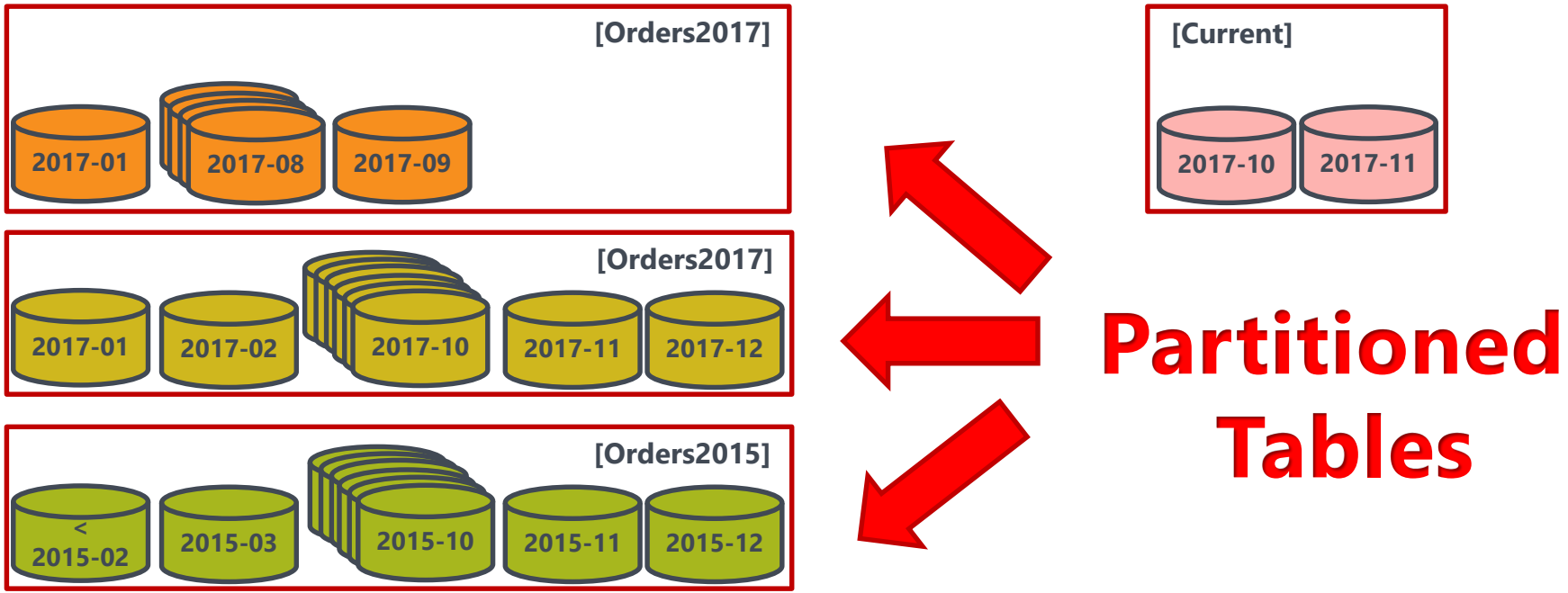
(*) It is easier to keep partitioned table on the single filegroup

Using Partitioned Tables and Views Together

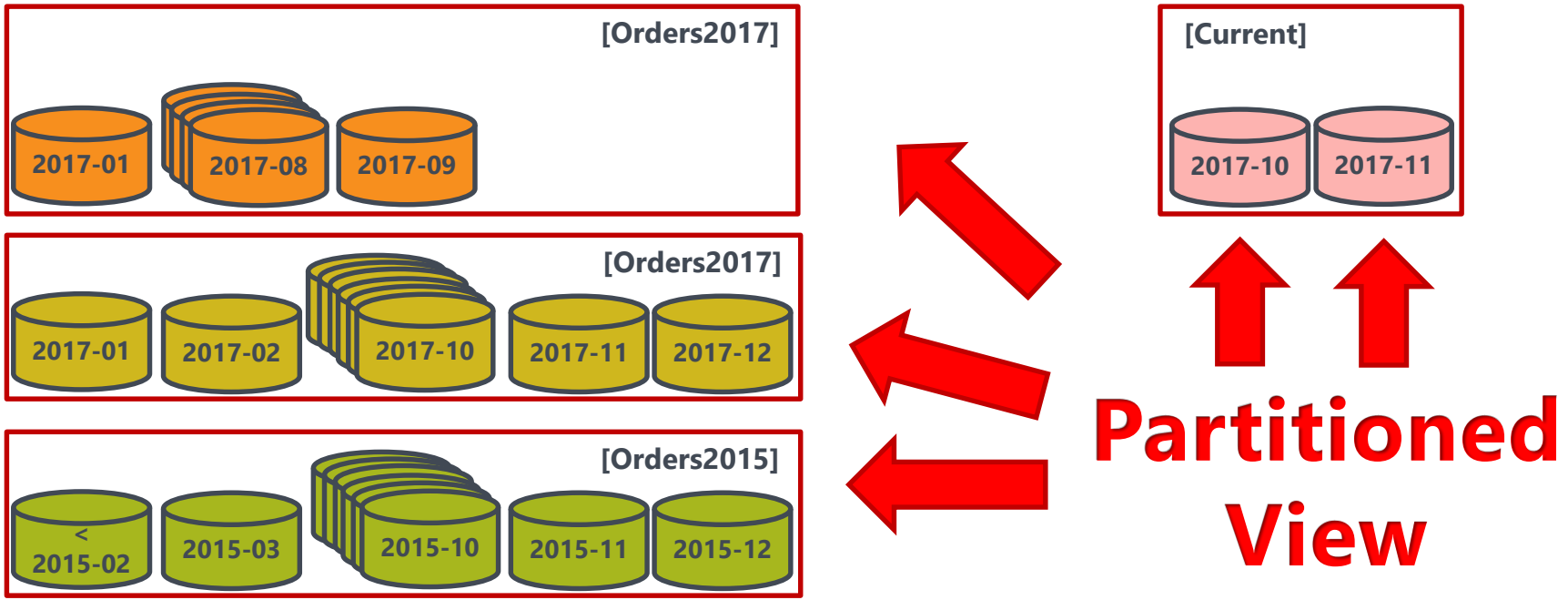




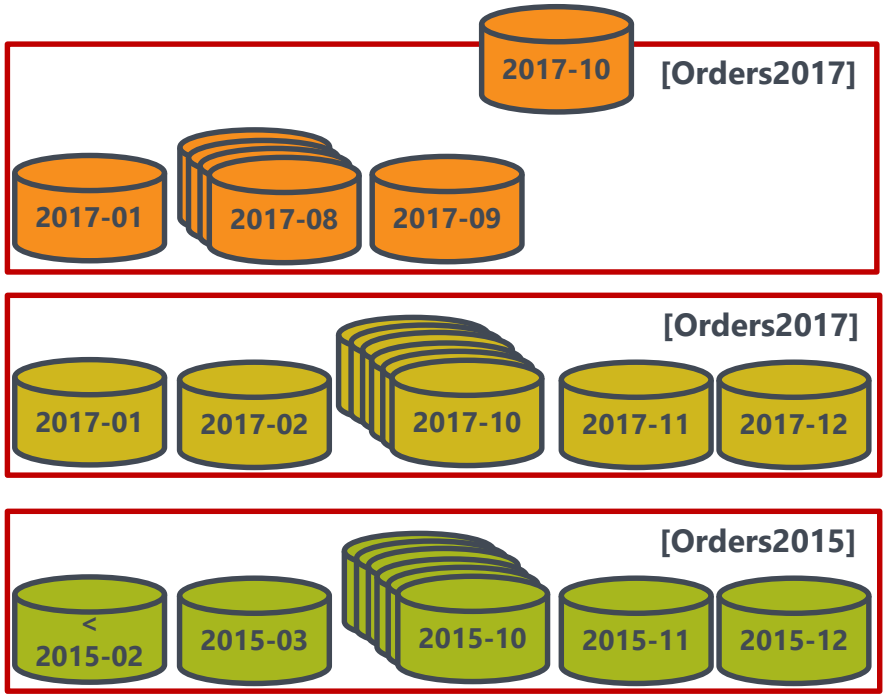
Using Partitioned Tables and Views Together

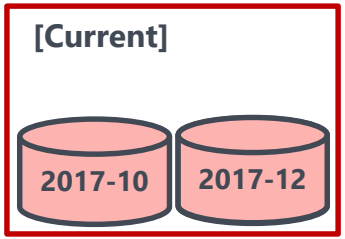


Using Partitioned Tables and Views Together



Using Partitioned Tables and Views Together





Online index rebuild

Partition switch

"Next month" table creation

Alteration of partitioned view

Challenges: CHECK Constraints Management

ALTER TABLE ADD CONSTAINT CHECK(..)

Always scans one of the indexes

Acquire and held Schema Modification (SCH-M) lock for duration of the scan

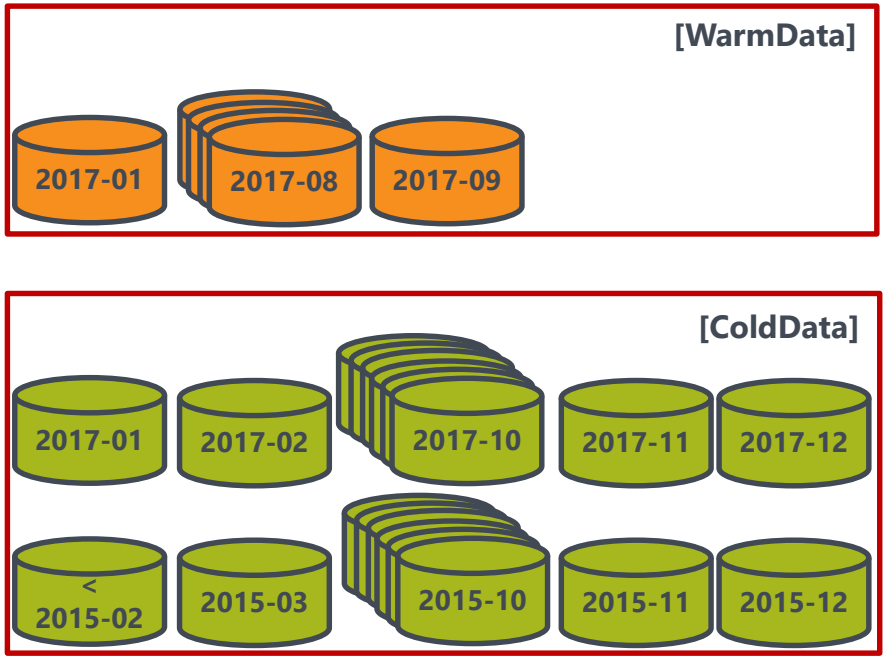
```
create table dbo.Orders2017
(
    constraint CHK_Orders2017_01_10 check
    (OrderDate >= '2017-01-01' and OrderDate < '2017-10-01'),

    constraint CHK_Orders2017_01_11 check
    (OrderDate >= '2017-01-01' and OrderDate < '2017-11-01'),

    constraint CHK_Orders2017_01_12 check
    (OrderDate >= '2017-01-01' and OrderDate < '2017-12-01'),

    constraint CHK_Orders2017 check
    (OrderDate >= '2017-01-01' and OrderDate < '2018-01-01'),
)
```

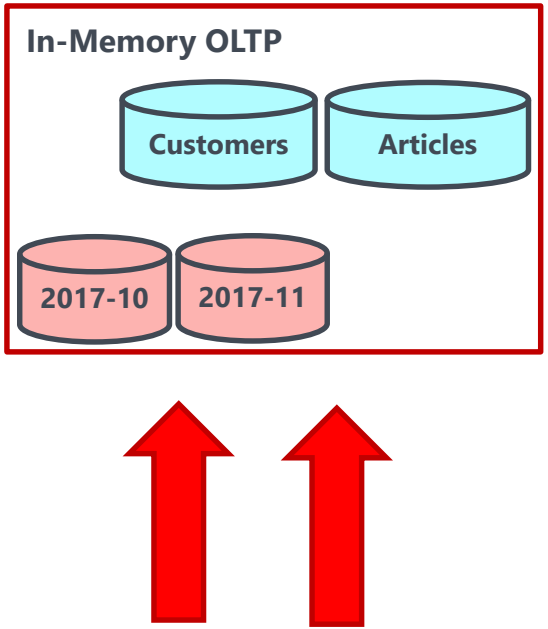
PASS

Combining Partitioning Tables and Views

Demo

Going Beyond B-Tree

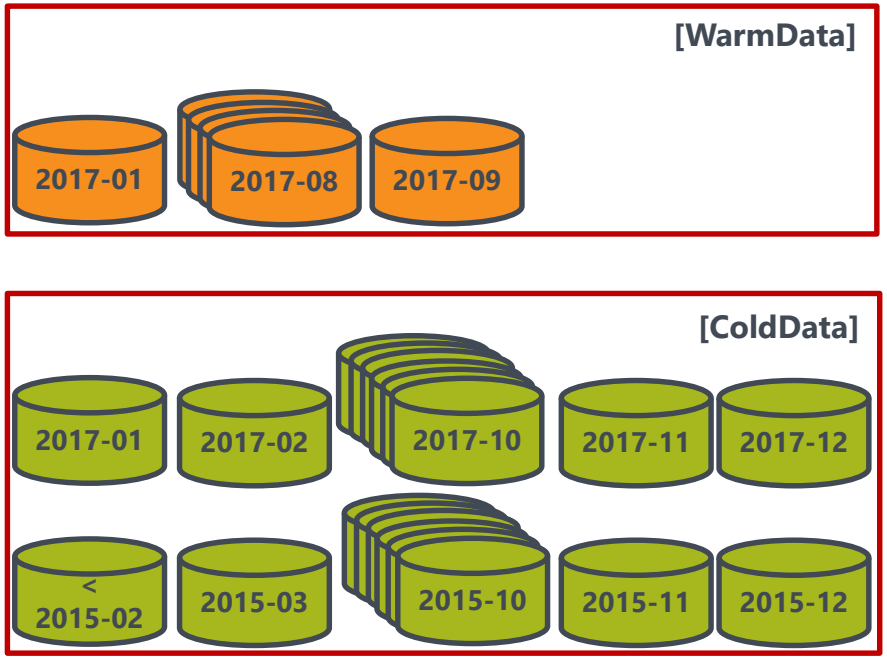


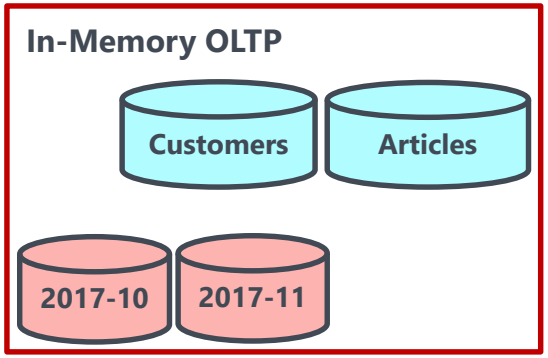


Memory-Optimized Tables

PASS

Going Beyond B-Tree



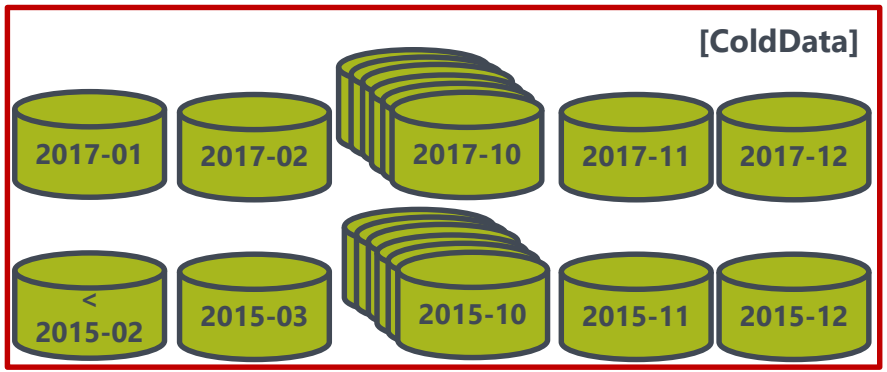


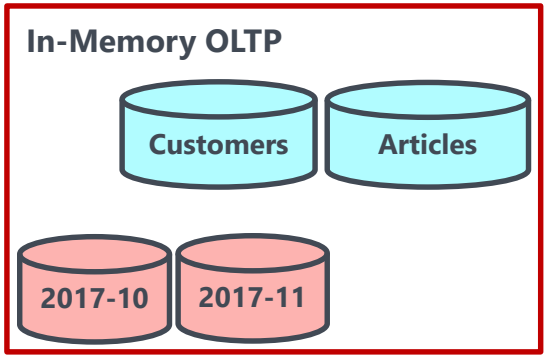
Partitioned Table (B-Tree):

Data for the last a few months

PASS

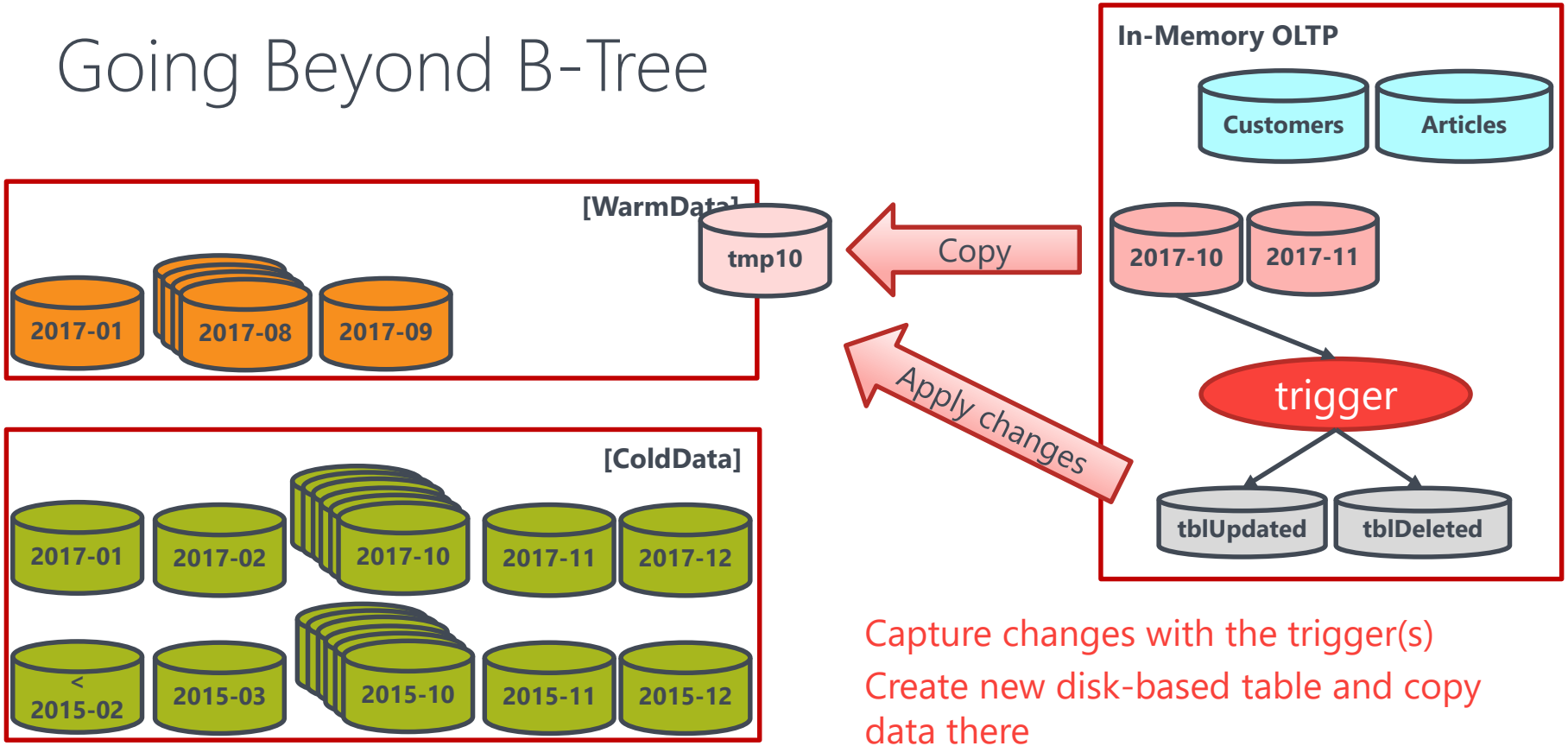
Going Beyond B-Tree





Partitioned Table (Columnstore):
Old data

Going Beyond B-Tree



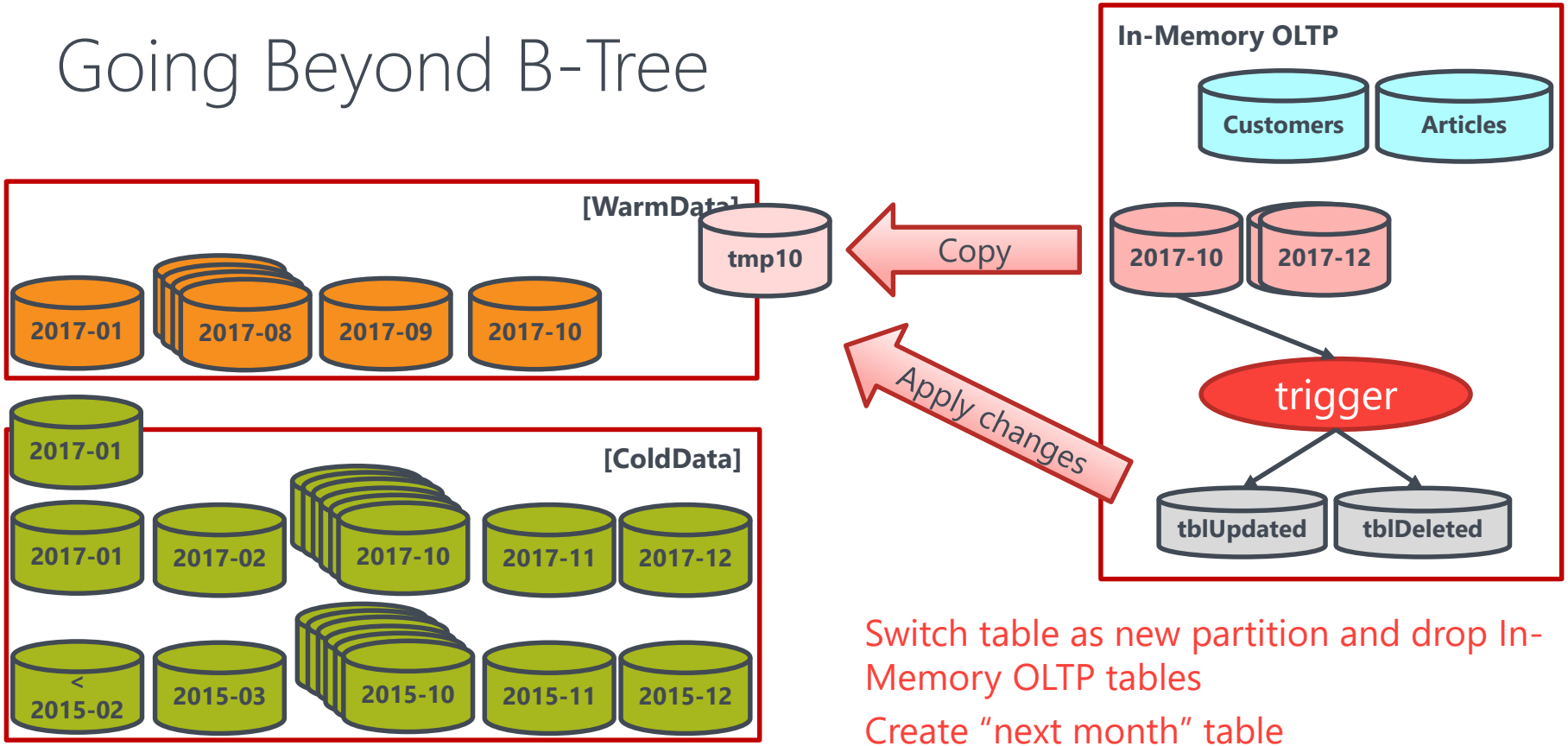
Capture changes with the trigger(s)

Create new disk-based table and copy data there

Apply captured changes

PASS

Going Beyond B-Tree



Switch table as new partition and drop In-Memory OLTP tables

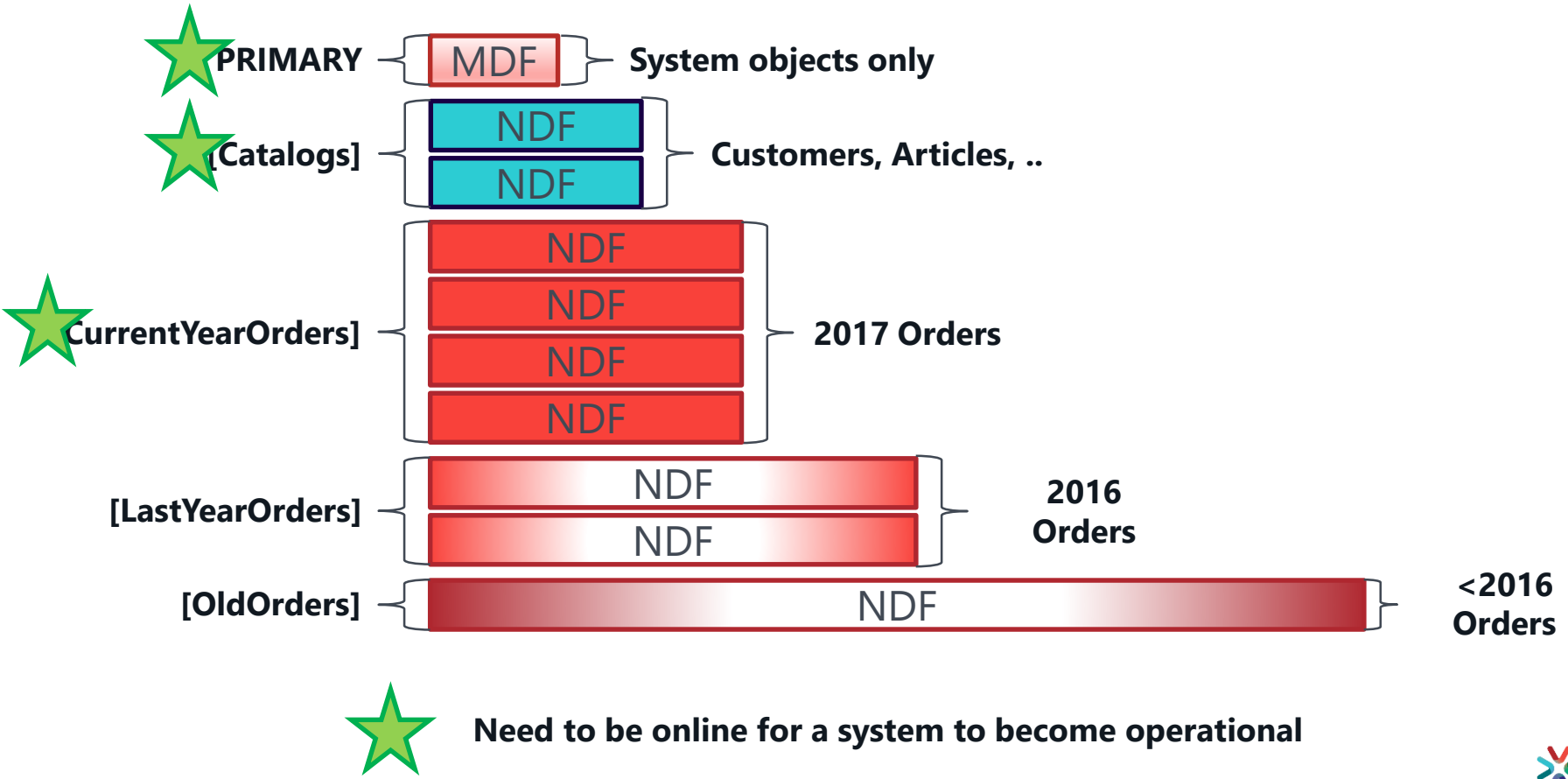
Create “next month” table

Eventually archive old “warm” data to columnstore

Moving In-Memory OLTP Data to Disk

Demo

Partitioning-Friendly Database Architecture



Challenges: Partial DB Availability & Queries

Query would fail if it needs to access offline filegroup

```
One of the partitions of index 'IDX_Orders2017_OrderDate_OrderId'
for table 'dbo.Orders2017'(partition ID 7205759404290867 2)
resides on a filegroup ("OldOrders") that cannot be accessed
because it is offline, restoring, or defunct.
This may limit the query result.
```

Partial DB Availability: Queries

Demo

View Alteration

```
alter view dbo.Orders(OrderDate, OrderId, WarehouseId
    ,CustomerId, ZipCode, Amount, DeliveryInstructions)
as
    select OrderDate, OrderId, WarehouseId, CustomerId
        ,ZipCode, Amount, DeliveryInstructions
    from dbo.Orders2017_11

    union all

    select OrderDate, OrderId, WarehouseId, CustomerId
        ,ZipCode, Amount, DeliveryInstructions
    from dbo.Orders2017_10

    union all

    select OrderDate, OrderId, WarehouseId, CustomerId
        ,ZipCode, Amount, NULL as DeliveryInstructions
    from dbo.Orders2017
```

View Alteration

```
alter view dbo.Orders(OrderDate, OrderId, WarehouseId
    ,CustomerId, ZipCode, Amount, DeliveryInstructions)
as
    select OrderDate, OrderId, WarehouseId, CustomerId
        ,ZipCode, Amount, DeliveryInstructions
    from dbo.Orders2017_11

    union all

    select OrderDate, OrderId, WarehouseId, CustomerId
        ,ZipCode, Amount, DeliveryInstructions
    from dbo.Orders2017_10

    union all

    select OrderDate, OrderId, WarehouseId, CustomerId
        ,ZipCode, Amount, NULL as DeliveryInstructions
    from dbo.Orders2017
    where $Partition.pfOrders2017(OrderDate) >= 9
```

PASS

Multiple Views

vOrders

vOrdersOLTP

```
create table dbo.Orders2017_11
(
    OrderDate datetime2(0) not null,
    OrderId int identity(1,1) not null,
    WarehouseId int not null,
    DeliveryInstructions varchar(1000) null,
    index O11 unique clustered(OrderDate,OrderId) on [CurrentOrders],
    check(OrderDate >= '2017-11-01' and OrderDate < '2017-12-01')
);
create table dbo.Orders2017_10
(
    OrderDate datetime2(0) not null,
    OrderId int identity(1,1) not null,
    WarehouseId int not null,
    DeliveryInstructions varchar(1000) null,
    index O10 unique clustered(OrderDate,OrderId) on [RecentOrders],
    index O10_NCI nonclustered(WarehouseId) with (data_compression=row),
    check(OrderDate >= '2017-10-01' and OrderDate < '2017-11-01')
);
create table dbo.Orders2017
(
    OrderDate datetime2(0) not null,
    OrderId int identity(1,1) not null,
    WarehouseId int not null,
    index O2017 clustered columnstore on [OldOrders],
    check(OrderDate >= '2017-01-01' and OrderDate < '2017-10-01')
);
```

PASS

Challenges: Index Key Size

Partitioning column becomes part of clustered index

➔ Becomes part of row-id in nonclustered indexes

8-byte datetime column:

1 million new rows daily	10 millions new rows daily	100 millions new rows daily	500 millions new rows daily	1 billion new rows daily
~8MB per index per day	~80MB per index per day	~800MB per index per day	~4GB per index per day	~8GB per index per day
~2.8GB per index per year	~28GB per index per year	~280GB per index per year	~2.4TB per index per year	~4.8TB per index per year

Index Key Columns

Demo

Challenges: Uniqueness Support

September 2017

Order Id	Order Date	Order Num	Modify Date
10000	2017-09-01 08:00:00	1-0901-1	2017-09-01 14:00:00
10001	2017-09-01 08:05:00	1-0901-2	2017-09-01 11:55:00
10002	2017-09-01 08:10:00	2-0901-3	2017-09-01 16:05:00
10003	2017-09-01 08:15:00	1-0901-4	2017-09-01 13:20:00

Order Num	Order Date
1-0901-1	2017-09-01 08:00:00
1-0901-2	2017-09-01 08:05:00
2-0901-3	2017-09-01 08:10:00
1-0901-4	2017-09-01 08:15:00

```
create table Orders(..)
on psOrders(OrderDate);
```

```
create unique index Idx1
on Orders(OrderNum)
on psOrders(OrderDate);
```

```
create unique index Idx1 on
Orders(OrderNum, OrderDate)
on psOrders(OrderDate);
```

October 2017

Order Num	Order Date
1-1001-1	2017-10-01 09:00:00
2-1001-2	2017-10-01 09:08:00
2-1001-3	2017-10-01 09:10:00
1-1001-4	2017-10-01 09:15:00

Order Id	Order Date	Order Num	Modify Date
15000	2017-10-01 09:00:00	1-1001-1	2017-10-01 10:00:00
15001	2017-10-01 09:08:00	2-1001-2	2017-10-01 12:30:00
15002	2017-10-01 09:10:00	2-1001-3	2017-10-01 11:20:00
15003	2017-10-01 09:15:00	1-1001-4	2017-10-01 13:50:00

Challenges: Uniqueness Support

September 2017

Order Id	Order Date	Order Num	Modify Date
10000	2017-09-01 08:00:00	1-0901-1	2017-09-01 14:00:00
10001	2017-09-01 08:05:00	1-0901-2	2017-09-01 11:55:00
10002	2017-09-01 08:10:00	2-0901-3	2017-09-01 16:05:00
10003	2017-09-01 08:15:00	1-0901-4	2017-09-01 13:20:00

Prevents partition switch

```
create unique index Idx2
on Orders(OrderNum)
```

Order Num
1-0901-1
1-0901-2
1-0901-4
1-1001-1
1-1001-4
2-0901-3
2-1001-2
2-1001-3

Non-aligned (September and October 2017)

October 2017

Order Id	Order Date	Order Num	Modify Date
15000	2017-10-01 09:00:00	1-1001-1	2017-10-01 10:00:00
15001	2017-10-01 09:08:00	2-1001-2	2017-10-01 12:30:00
15002	2017-10-01 09:10:00	2-1001-3	2017-10-01 11:20:00
15003	2017-10-01 09:15:00	1-1001-4	2017-10-01 13:50:00

Uniqueness Support

Demo

Challenges: Plan Regressions

September 2017

Order Id	Order Date	Order Num	Modify Date	Modify Date
10000	2017-09-01 08:00:00	1-0901-1	2017-09-01 14:00:00	2017-09-01 11:55:00
10001	2017-09-01 08:05:00	1-0901-2	2017-09-01 11:55:00	2017-09-01 14:00:00
10002	2017-09-01 08:10:00	2-0901-3	2017-09-01 16:05:00	2017-09-01 16:05:00
10003	2017-09-01 08:15:00	1-0901-4	2017-10-01 10:40:00	2017-10-02 10:40:00

October 2017

Order Id	Order Date	Order Num	Modify Date	Modify Date
15000	2017-10-01 09:00:00	1-1001-1	2017-10-01 10:00:00	2017-10-01 10:00:00
15001	2017-10-01 09:08:00	2-1001-2	2017-10-01 12:30:00	2017-10-01 11:20:00
15002	2017-10-01 09:10:00	2-1001-3	2017-10-01 11:20:00	2017-10-01 12:30:00
15003	2017-10-01 09:15:00	1-1001-4	2017-10-01 13:50:00	2017-10-01 13:50:00

```
create table Orders(..)
on psOrders(OrderDate);
```

```
create index Idx1
on Orders(ModifyDate)
on psOrders(OrderDate);
```

PASS

Challenges: Plan Regressions

September 2017

Order Id	Order Date	Order Num	Modify Date
10000	2017-09-01 08:00:00	1-0901-1	2017-09-01 14:00:00
10001	2017-09-01 08:05:00	1-0901-2	2017-09-01 11:55:00
10002	2017-09-01 08:10:00	2-0901-3	2017-09-01 16:05:00
10003	2017-09-01 08:15:00	1-0901-4	2017-10-01 10:40:00

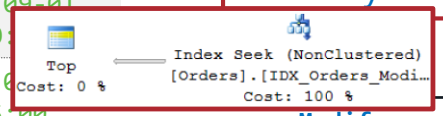
Modify Date
2017-09-01 11:55:00
2017-09-01 14:00:00
2017-09-01 16:05:00
2017-10-02 10:40:00

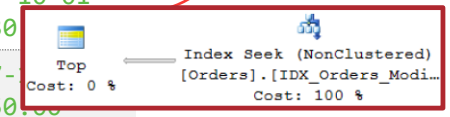
October 2017

Order Id	Order Date	Order Num	Modify Date
15000	2017-10-01 09:00:00	1-1001-1	2017-10-01 10:00:00
15001	2017-10-01 09:08:00	2-1001-2	2017-10-01 12:30:00
15002	2017-10-01 09:10:00	2-1001-3	2017-10-01 11:20:00
15003	2017-10-01 09:15:00	1-1001-4	2017-10-01 13:50:00

Modify Date
2017-10-01 10:00:00
2017-10-01 11:20:00
2017-10-01 12:30:00
2017-10-01 13:50:00

```
select top 3 with ties ...
from Orders
where ModifyDate > '2017-10-01 07:00'
order by ModifyDate
```





Modify Date
2017-10-02 10:40:00
2017-10-01 10:00:00
2017-10-01 11:20:00
2017-10-01 12:30:00

Modify Date
2017-10-02 10:00:00
2017-10-01 10:40:00
2017-10-01 11:20:00
2017-10-01 12:30:00

Challenges: Plan Regressions

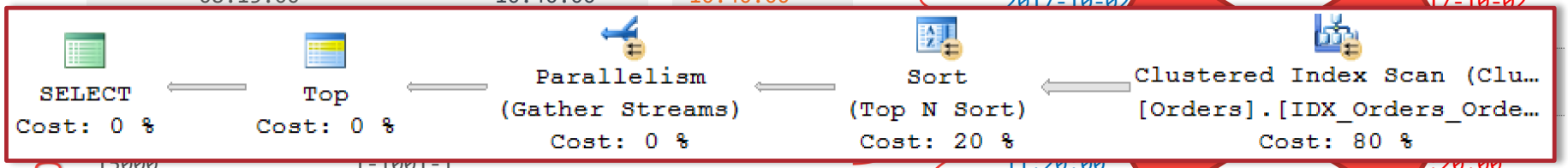
```
select top 3 with ties ...
from Orders
where ModifyDate > '2017-10-01 07:00'
```

September 2017

Order Id	Order Date	Order Num	Modify Date	Modify Date
10000	2017-09-01 08:00:00	1-0901-1	2017-09-01 14:00:00	2017-09-01 11:55:00
10001	2017-09-01 08:05:00	1-0901-2	2017-09-01 11:55:00	2017-09-01 14:00:00
10002	2017-09-01 08:10:00	2-0901-3	2017-09-01 16:05:00	2017-09-01 16:05:00
10003	2017-09-01 08:15:00	1-0901-4	2017-10-01 10:40:00	2017-10-02 10:40:00

October 20

15000	2017-10-01 09:00:00	1-1001-1	2017-10-01 10:00:00	2017-10-01 10:00:00
15001	2017-10-01 09:08:00	2-1001-2	2017-10-01 12:30:00	2017-10-01 11:20:00
15002	2017-10-01 09:10:00	2-1001-3	2017-10-01 11:20:00	2017-10-01 12:30:00
15003	2017-10-01 09:15:00	1-1001-4	2017-10-01 13:50:00	2017-10-01 13:50:00



Modify Date
2017-10-01 11:20:00
2017-10-01 12:30:00

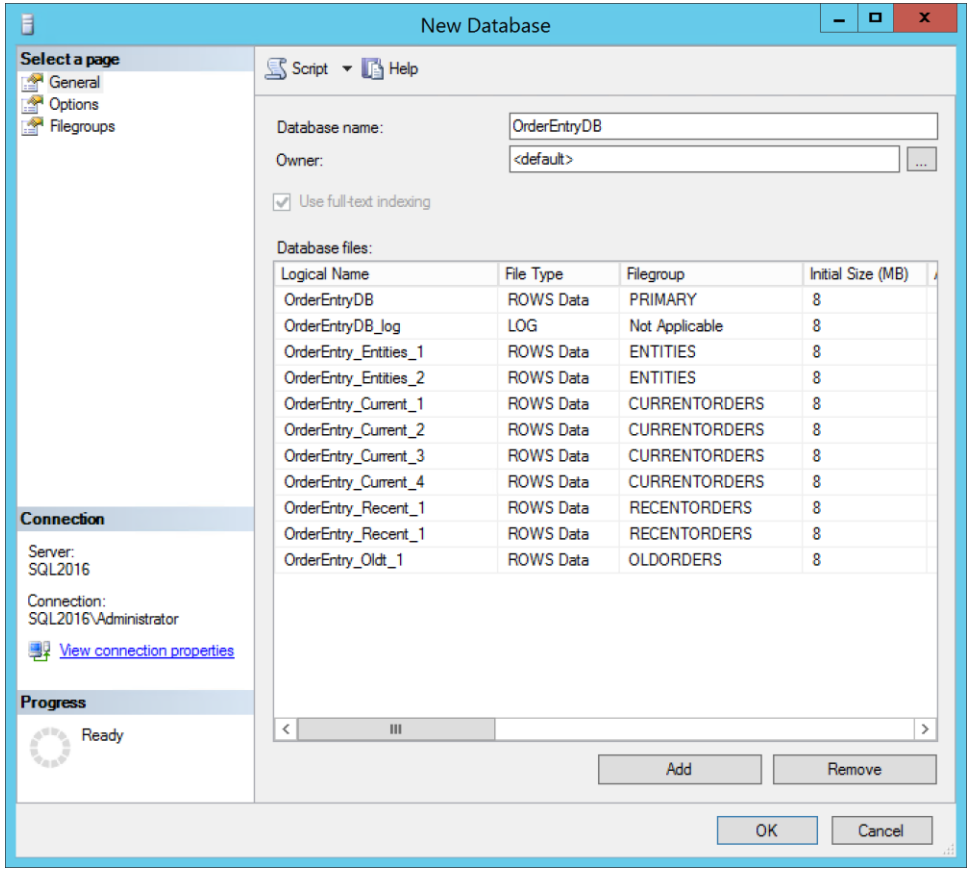
Modify Date
2017-10-01 12:30:00

PASS

Optimizing with $Partition Function

Demo

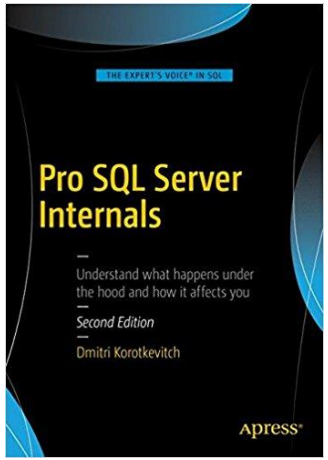
When to Partition?



We covered

Data Partitioning in SQL Server

- How to benefit from data partitioning
  - Improve system availability and performance
  - Reduce maintenance overhead and storage cost
- How to architecting the systems that utilize data partitioning
  - Utilize proper partitioned objects
  - Use different objects and technologies together



Any questions?

dk@aboutsqlserver.com

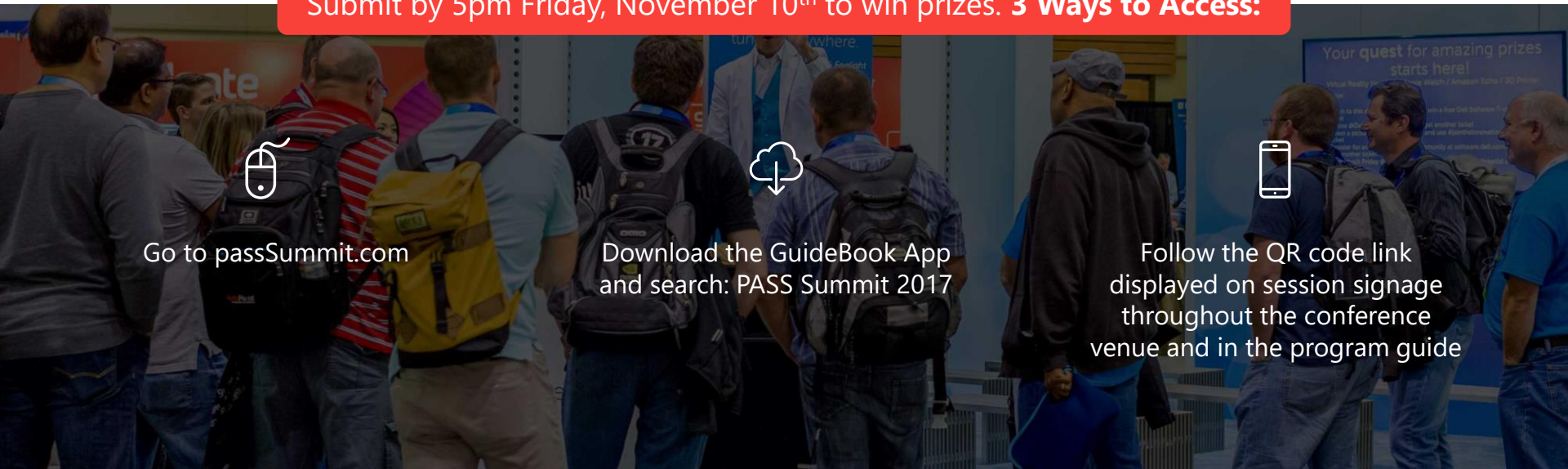
http://aboutsqlserver.com

PASS

Session evaluations

Your feedback is important and valuable.

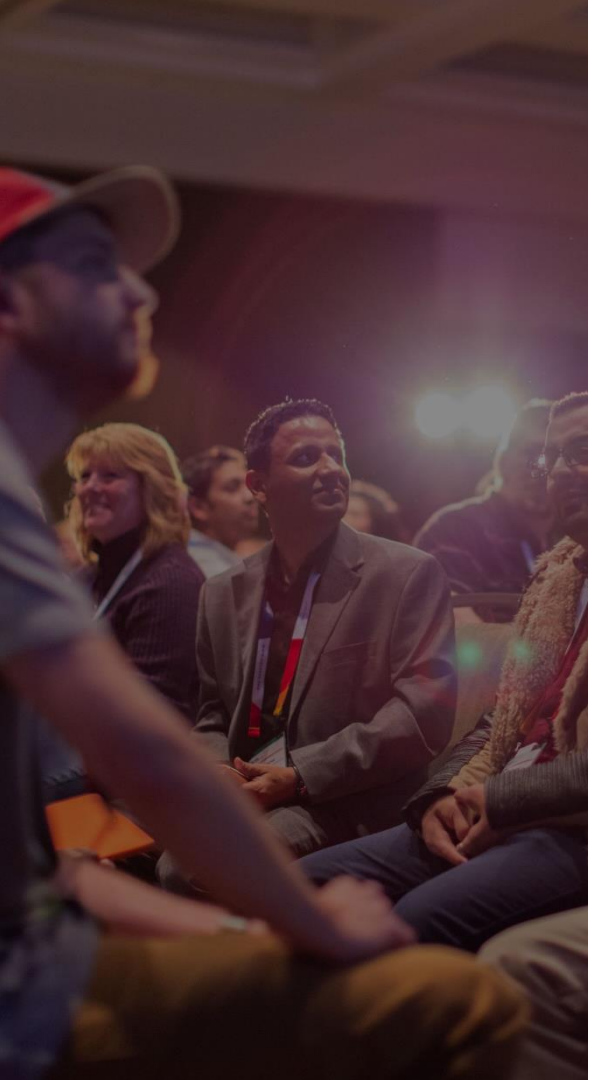
Submit by 5pm Friday, November 10th to win prizes. **3 Ways to Access:**



Go to passSummit.com

Download the GuideBook App and search: PASS Summit 2017

Follow the QR code link displayed on session signage throughout the conference venue and in the program guide



Thank You

Learn more from Dmitri Korotkevitch

http://aboutsqlserver.com dk@aboutsqlserver.com

PASS