SQL Server: Deadlock Analysis and Prevention

Module 6: Example Deadlock Scenarios

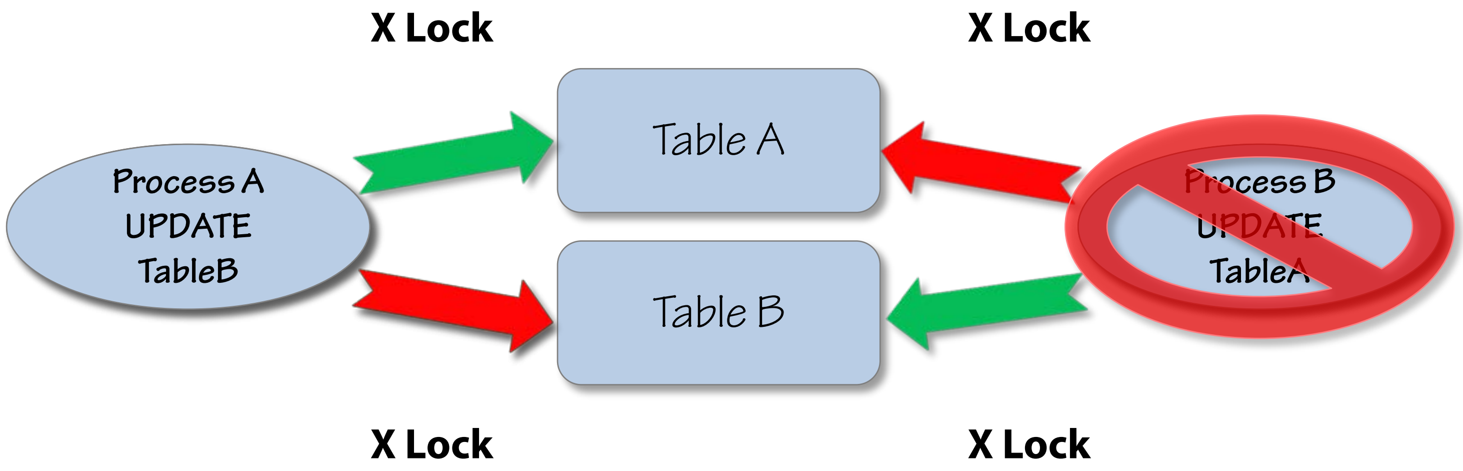
Jonathan M. Kehayias
Jonathan@SQLskills.com

Introduction

- **There are some common deadlock scenarios that occur in the field**
- **Understanding these scenarios can make it much easier to design changes that will prevent them**
- **Looking at common deadlock scenarios can assist with understanding them to best prevent or reduce their occurrences**
- **In this module we'll cover:**
  - Reverse object order deadlocks
  - Bookmark lookup deadlocks
  - Serializable deadlocks
  - Cascading constraint deadlocks
  - Lock escalation deadlocks
  - Memory deadlocks
  - Intra-query parallelism deadlocks
  - Multi-victim deadlocks

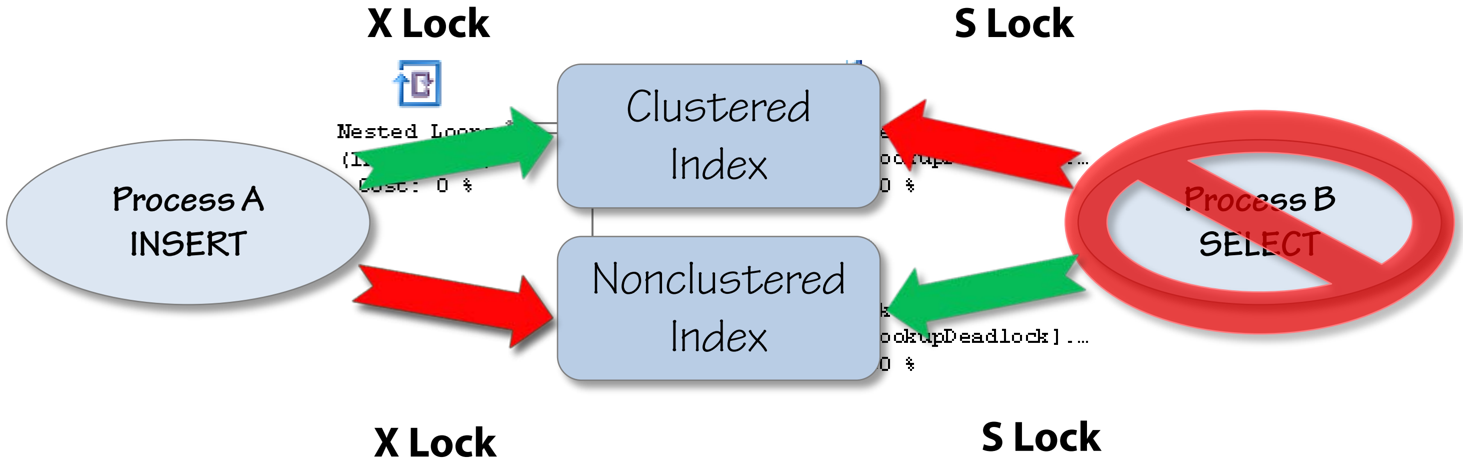
Reverse Object Order Deadlocks

- Two processes access objects in reverse order



Bookmark Lookup Deadlock

- Bookmark lookup deadlock



Serializable Deadlock

- Serializable lock-conversion deadlock

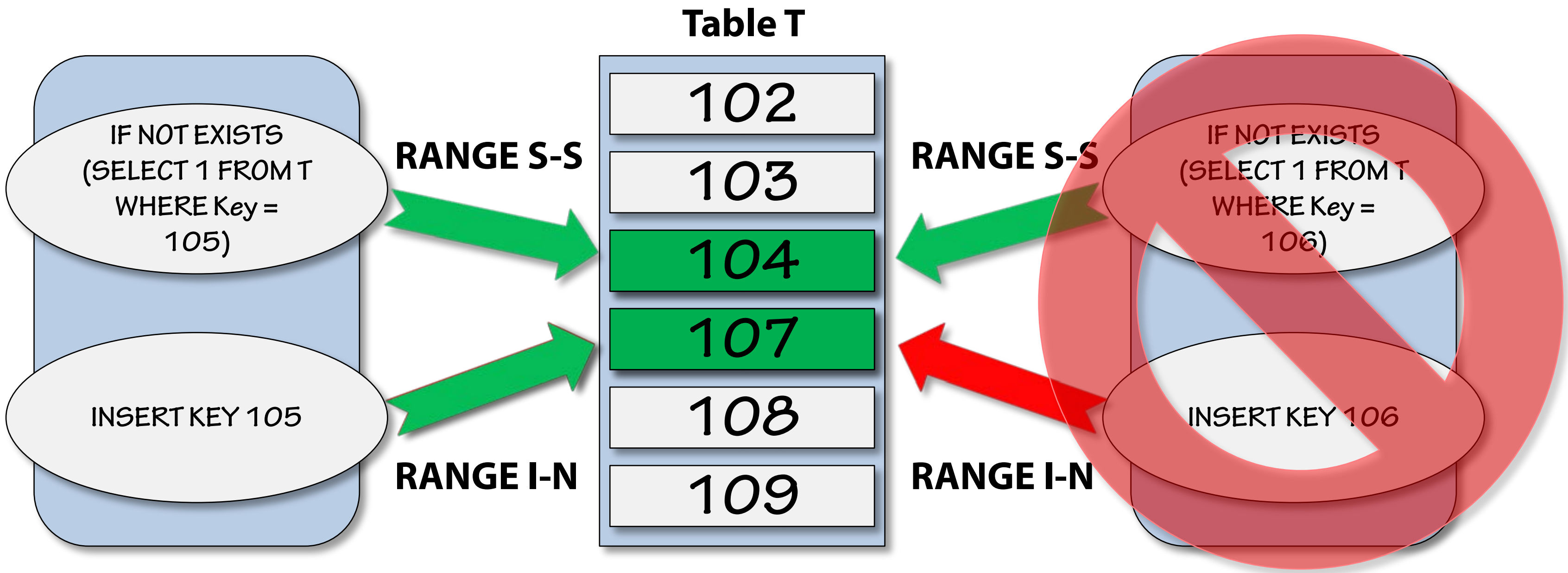
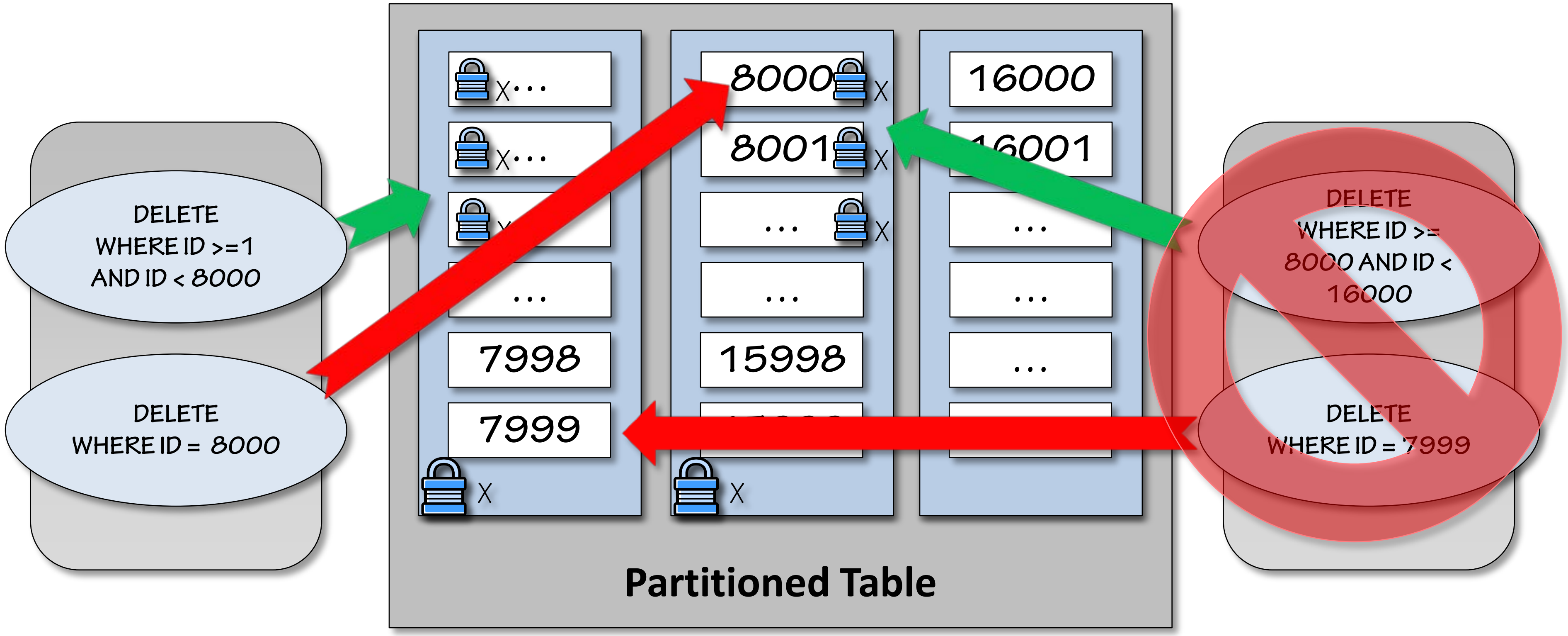


Table T
102
103
104
107
108
109
IF NOT EXISTS (SELECT 1 FROM T WHERE Key = 105)
INSERT KEY 105
RANGE S-S
RANGE I-N
IF NOT EXISTS (SELECT 1 FROM T WHERE Key = 106)
INSERT KEY 106
RANGE S-S
RANGE I-N

Cascading Constraint Deadlocks

- Cascade operations for constraint enforcement switch to serializable isolation under the covers to prevent concurrent operations from inserting values into child tables that would violate the foreign key constraint being enforced
- Resulting deadlock is similar to a serializable deadlock with the exception that the isolation level reported by the deadlock graph will not be serializable, it will instead be the session isolation level
- SQLCAT whitepaper discusses these deadlocks in depth (http://bit.ly/RefConstDeadlock)

Lock Escalation Deadlocks



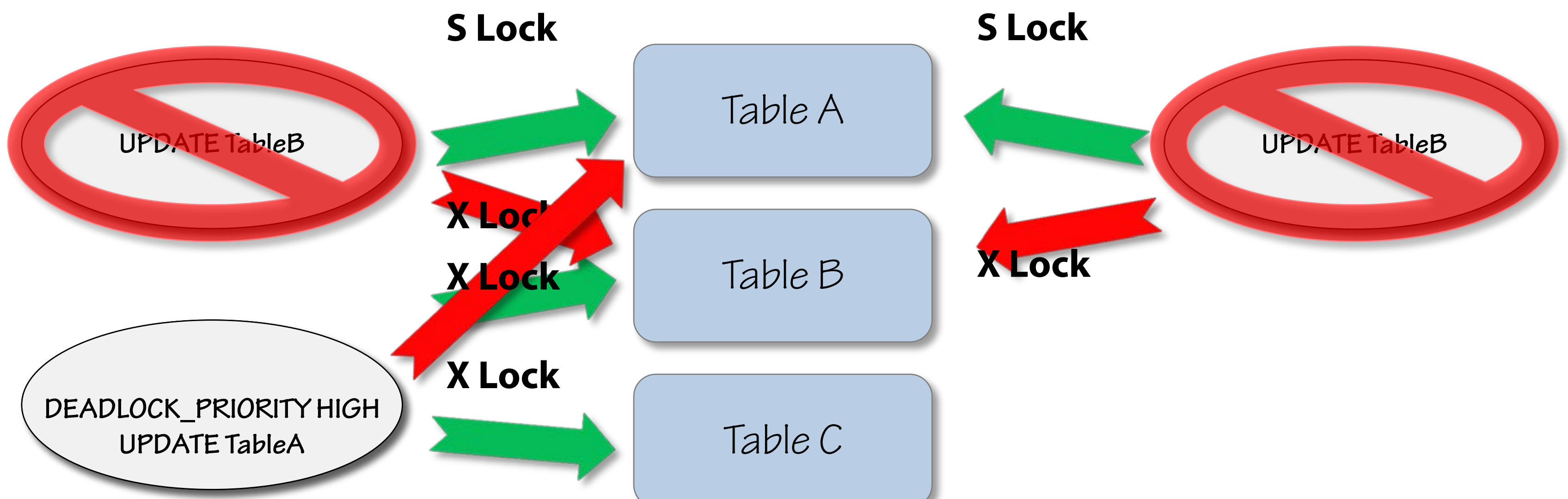
Memory Grant Deadlocks

- **Execution memory in SQL Server is a limited resource that is controlled internally by a semaphore**
  - Semaphores track the availability of a resource and provide controlled access to tasks requesting usage of the resource
- **Large sort and hash operations require execution memory grants to be able to execute efficiently**
  - If no execution memory is available the queries requesting a grant will wait with a RESOURCE_SEMAPHORE wait type
- **If a transaction has acquired locks on a resource, blocking can occur if a query inside of the same transaction has to wait for execution memory to be granted**
- **If another transaction is waiting for an execution memory grant ahead of the active transaction and starts executing it may request a lock that conflicts with the existing lock, creating a deadlock scenario**

Intra-query Parallelism Deadlocks

- **Deadlock between parallel threads within the same session that is executing using parallelism**
- **Generally associated with a bug in SQL Server query optimization that may not be easily fixed, or could cause query plan regressions and may be to risky to fix**
- **Can be identified in the deadlock graph by all processes having the same spid**
  - The deadlock resources will be on exchangeEvent resources only
- **May require reducing parallelism for the query, rewriting the query, or tuning the database indexes to change the plan being used during the query execution**

Multi-victim Deadlocks



Resolving Deadlocks

- **Change indexing to cover queries**
- **Enable Read Committed Snapshot Isolation**
  - Writers won’t block readers
  - Reads won’t block writers
- **Change isolation level**
- **Use locking hints to force specific lock types to prevent lock conversion**

Summary

- **Prevention of common deadlock scenarios is possible once you understand the root cause of the deadlock**
  - Understanding common deadlock patterns will help with being able to make the necessary changes to prevent/reduce their occurrences
- **Using NOLOCK may be appropriate in some scenarios, but other options exist to mitigate deadlocks where this might have been used in past versions of SQL Server**
  - Row versioning through snapshot or read committed snapshot isolation
  - Using locking hints to change lock granularity
  - Creating covering indexes to prevent deadlock scenarios from occurring
- **The next module will look at:**
  - Handling deadlocks