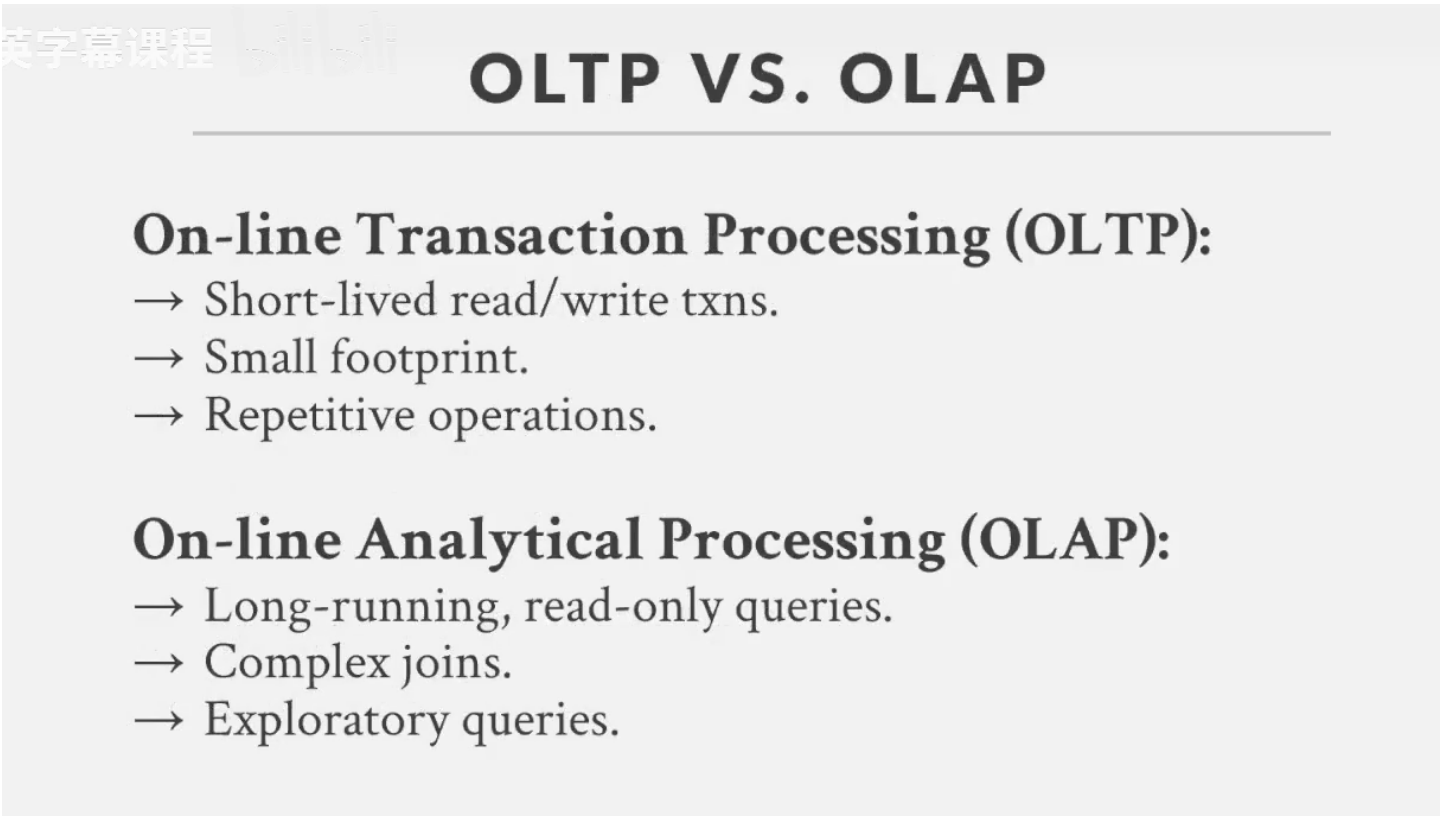
分布式OLTP DB

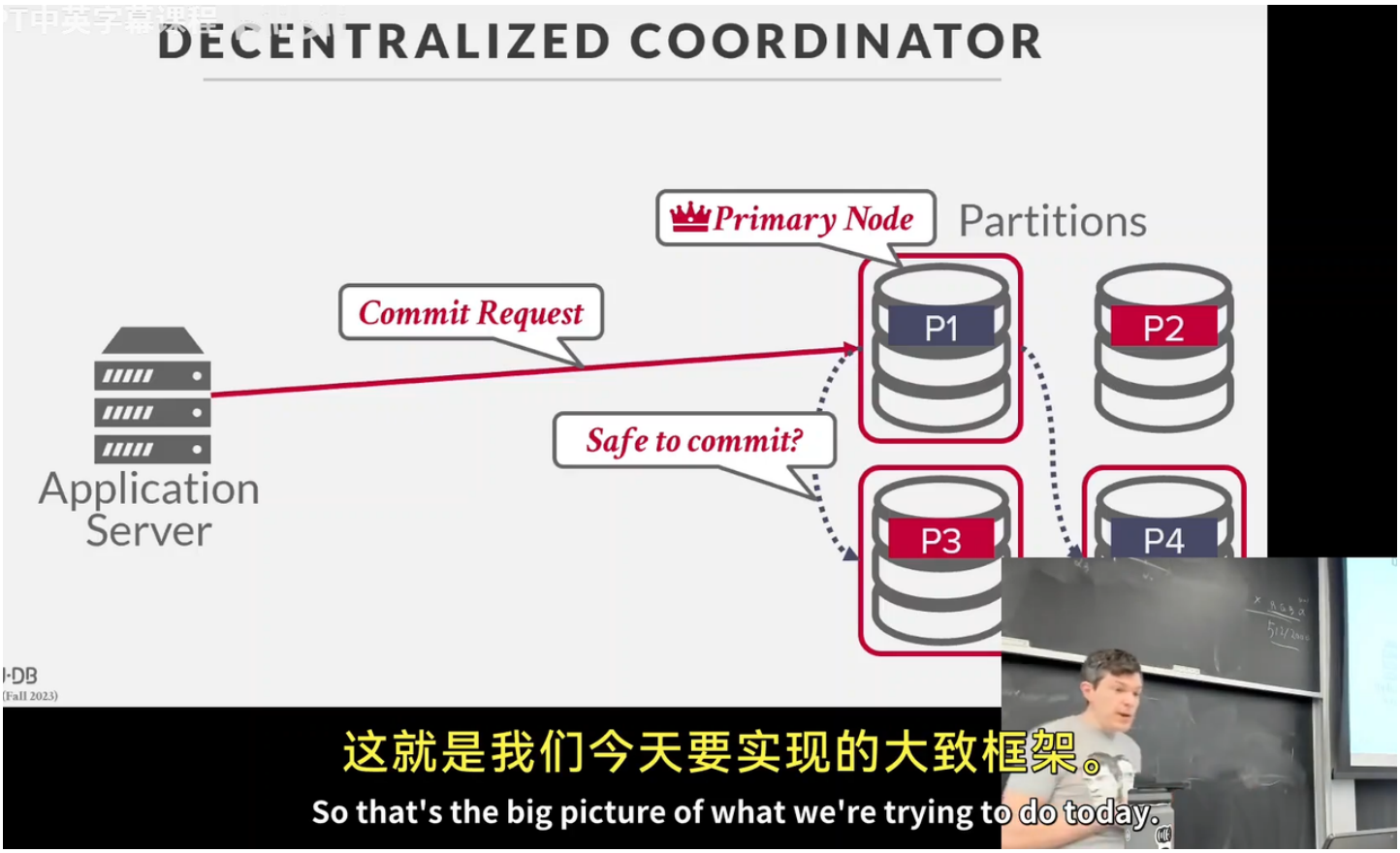
上节回顾

OLTP VS . OLAP

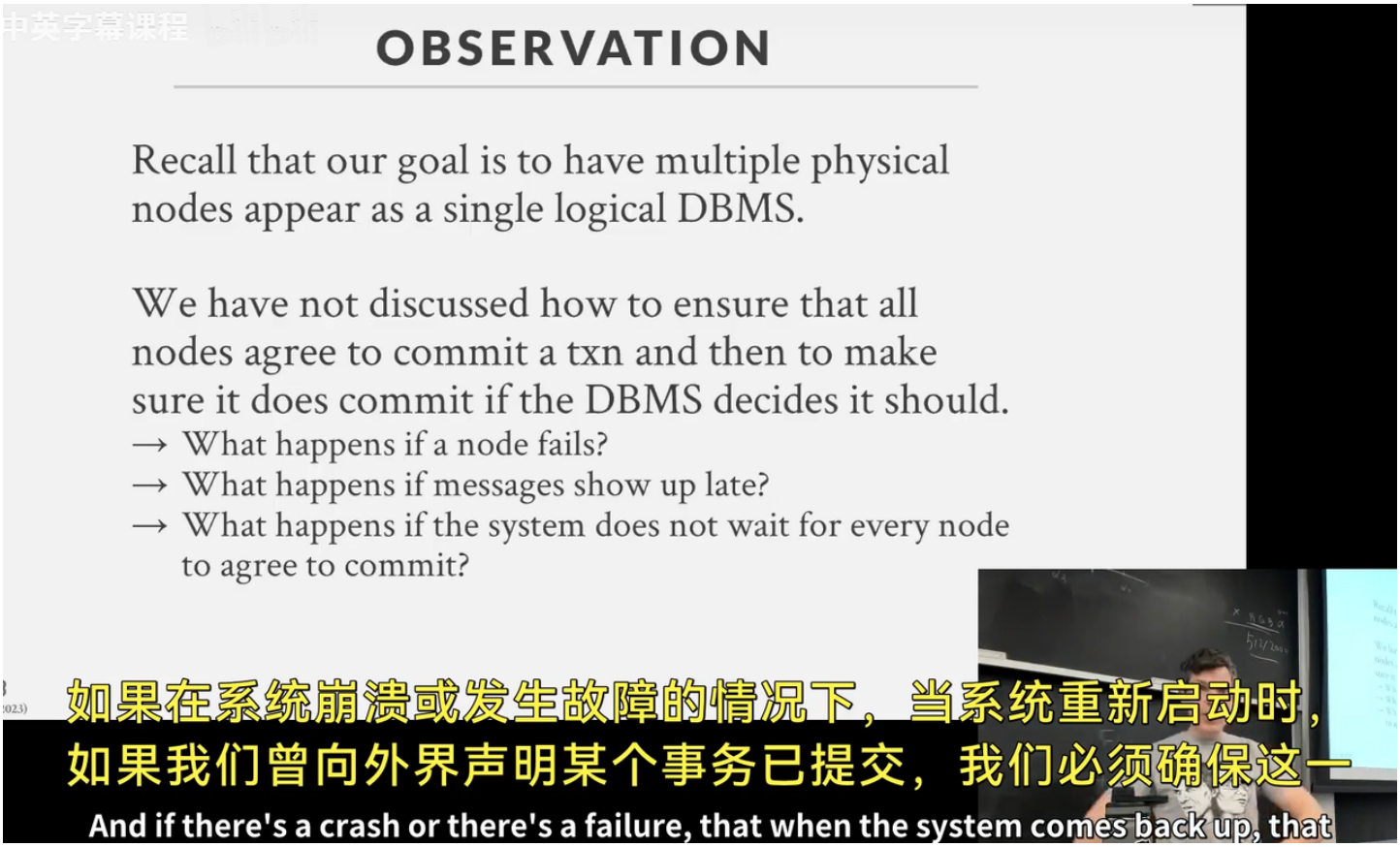
> 执行时间30/50ms以上的事务被认定是长事务



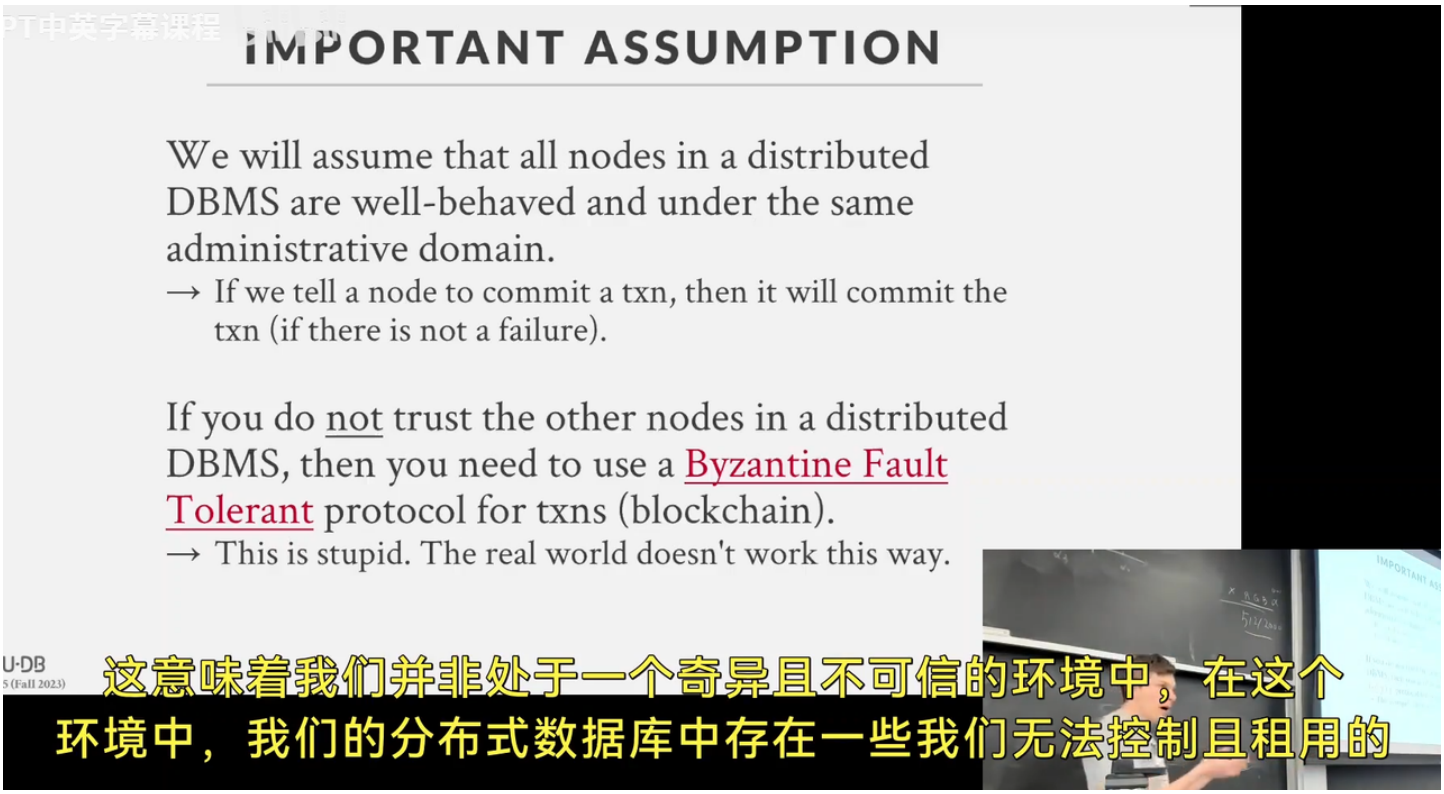
今天要实现的大致框架



> 目标：多个物理节点被视作一个逻辑上的DBMS



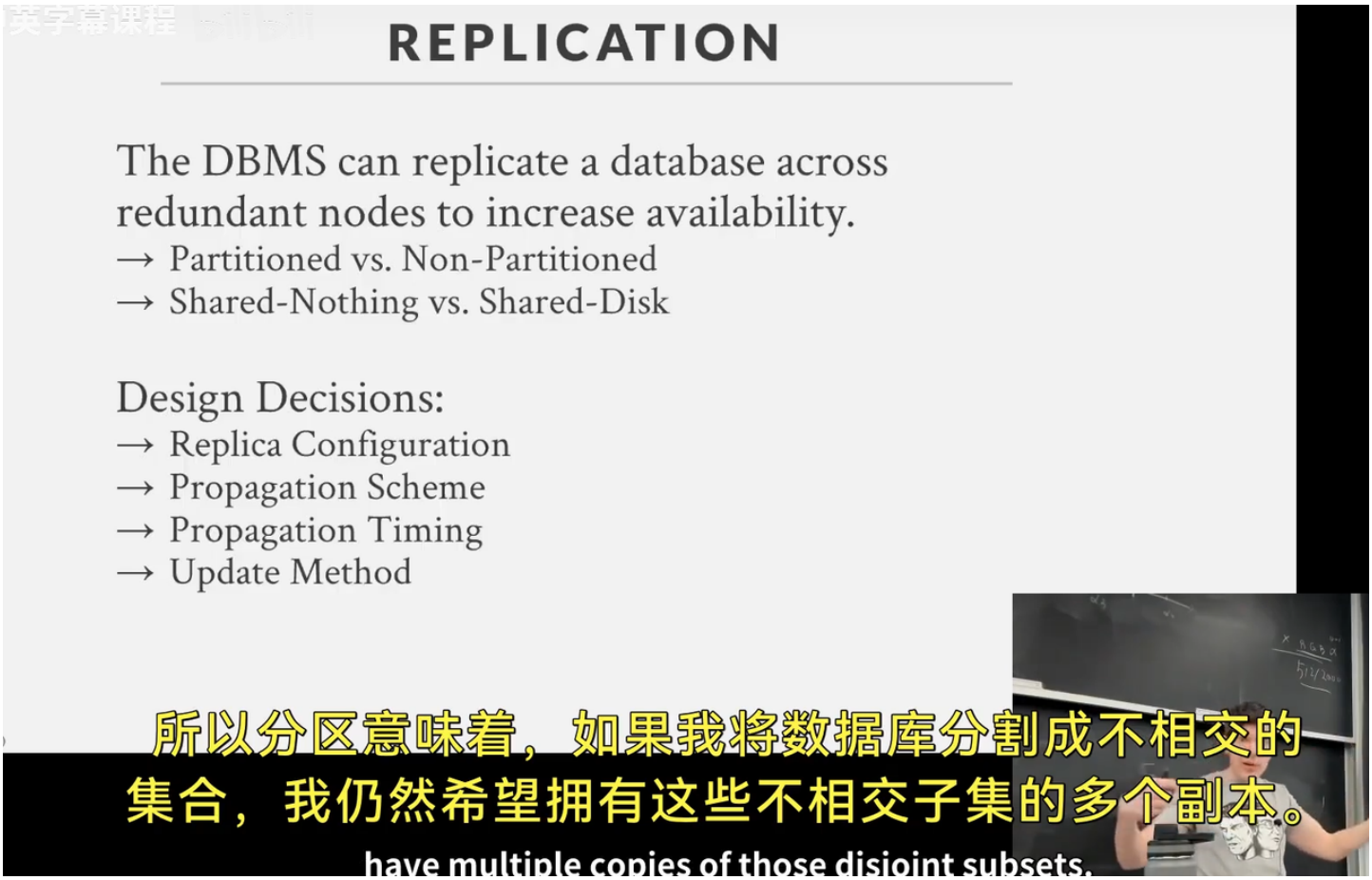
环境假设：无需保证拜占庭协议



Agenda

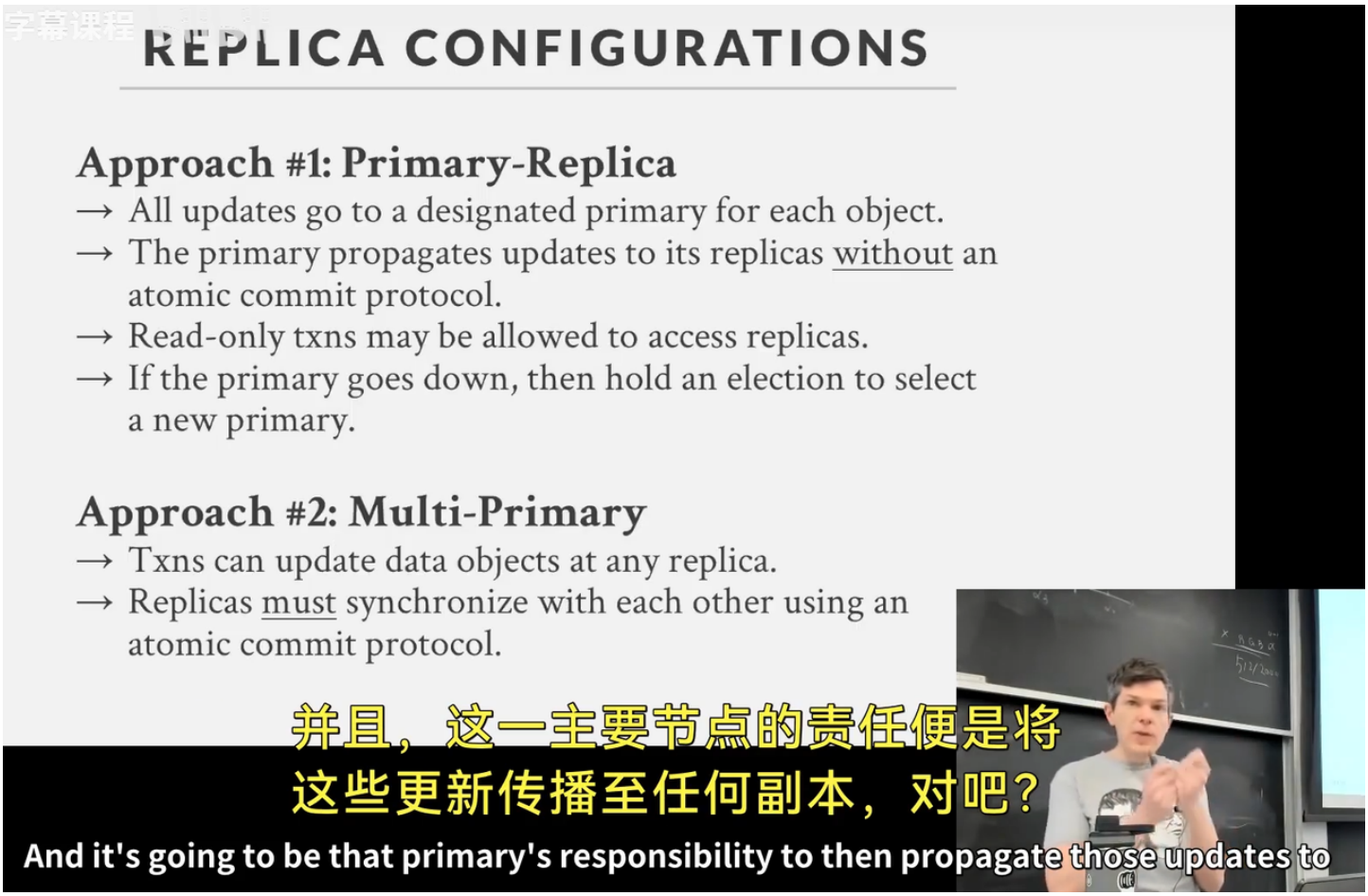
复制

> 可以对只读查询进行负载分流

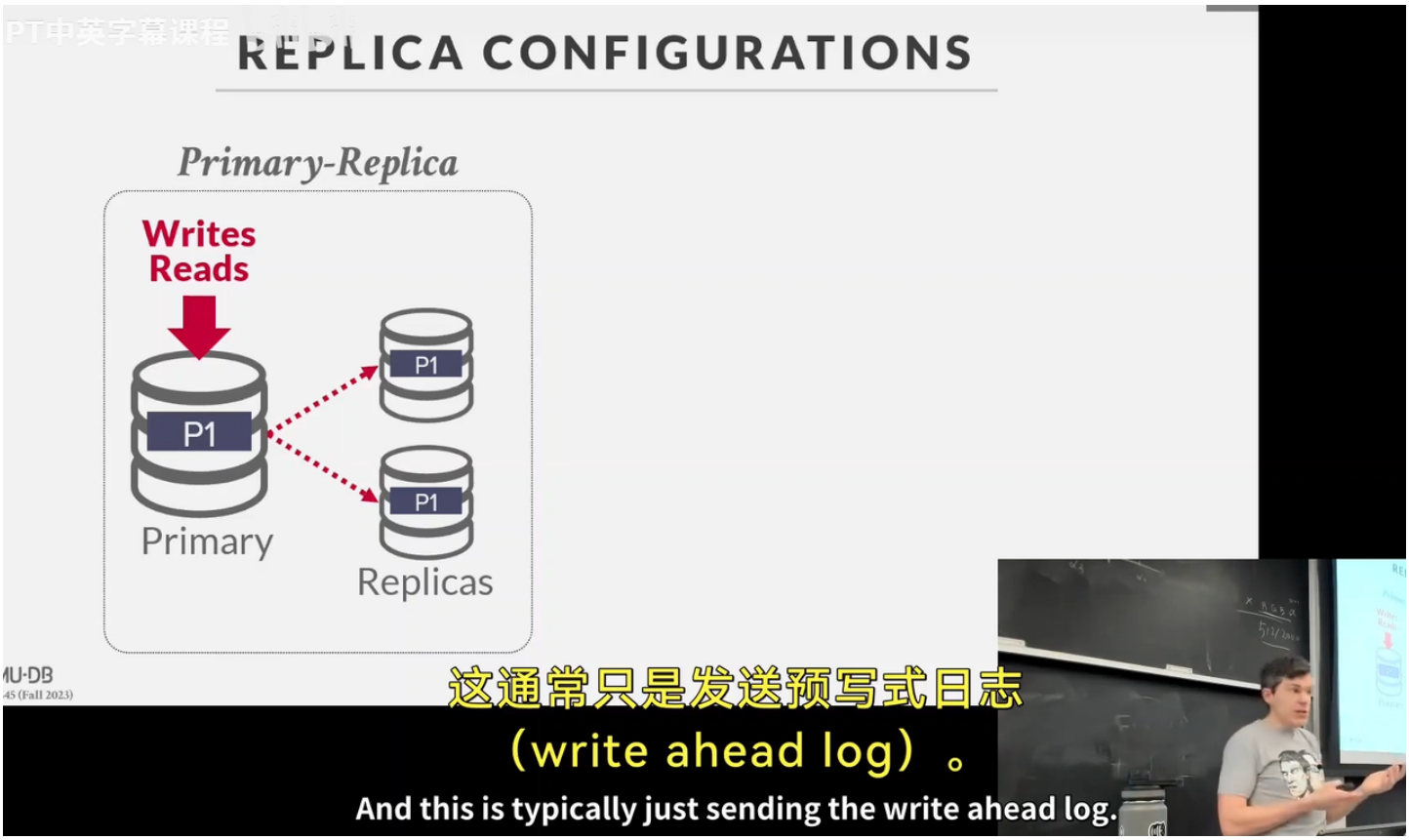

REPLICATION
The DBMS can replicate a database across redundant nodes to increase availability.
→ Partitioned vs. Non-Partitioned
→ Shared-Nothing vs. Shared-Disk
Design Decisions:
→ Replica Configuration
→ Propagation Scheme
→ Propagation Timing
→ Update Method
所以分区意味着，如果我将数据库分割成不相交的集合，我仍然希望拥有这些不相交子集的多个副本。
have multiple copies of those disjoint subsets.


如果数据库未进行分区，这在大多数数据库系统中是常见情况，那么我希望能够利用副本进行负载分流，例如处理只读

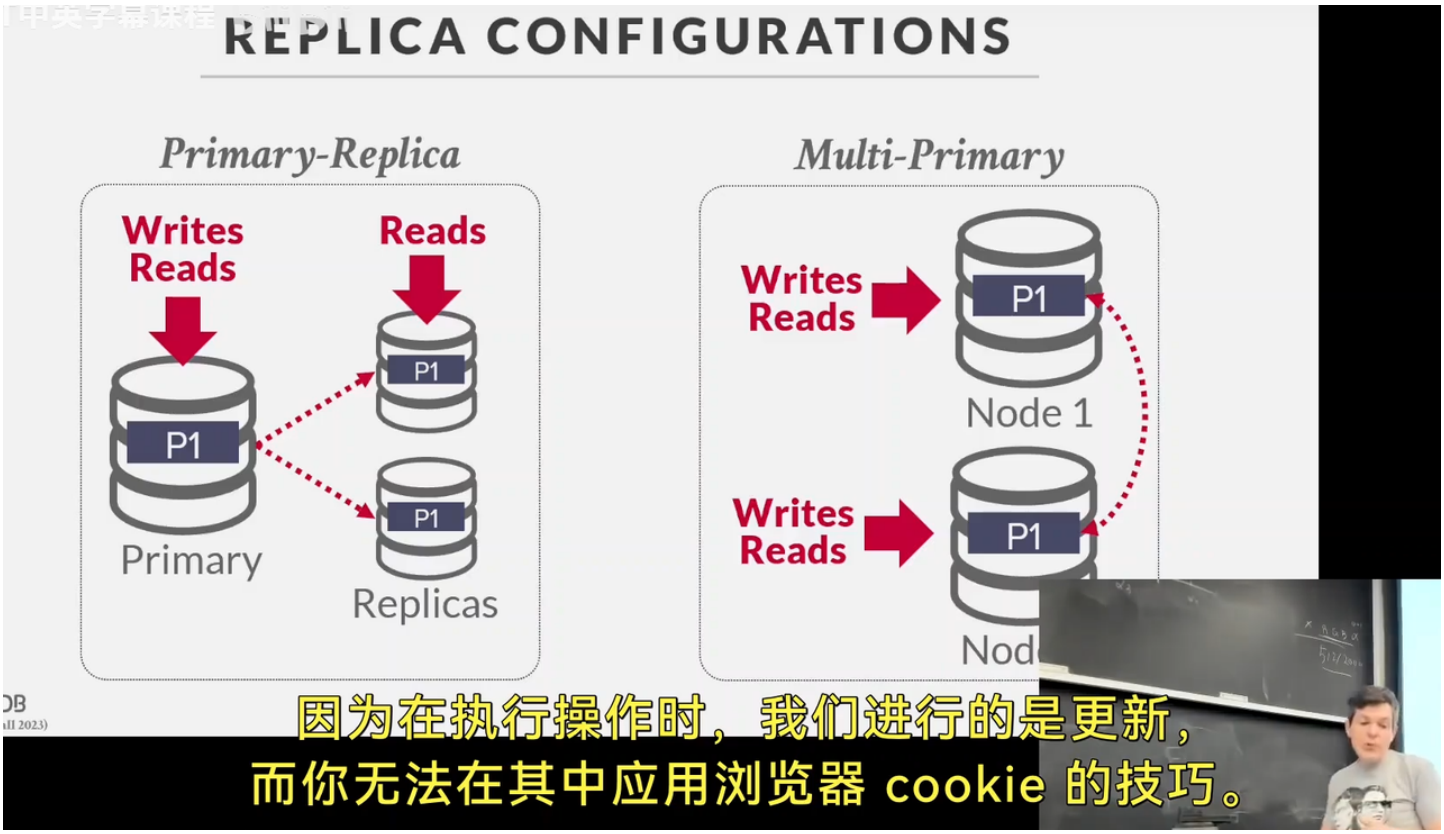
1. 复制的设置



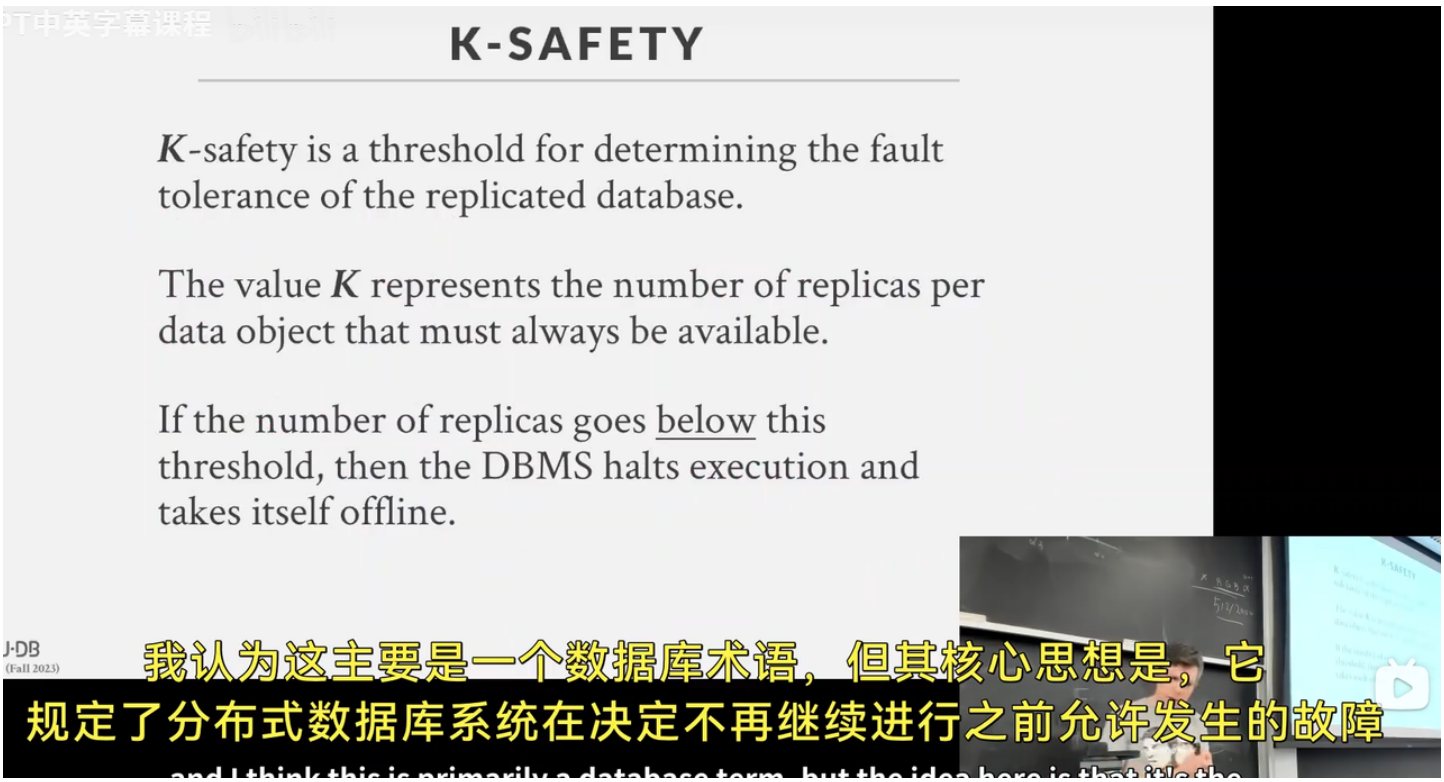
2. 单一主备份



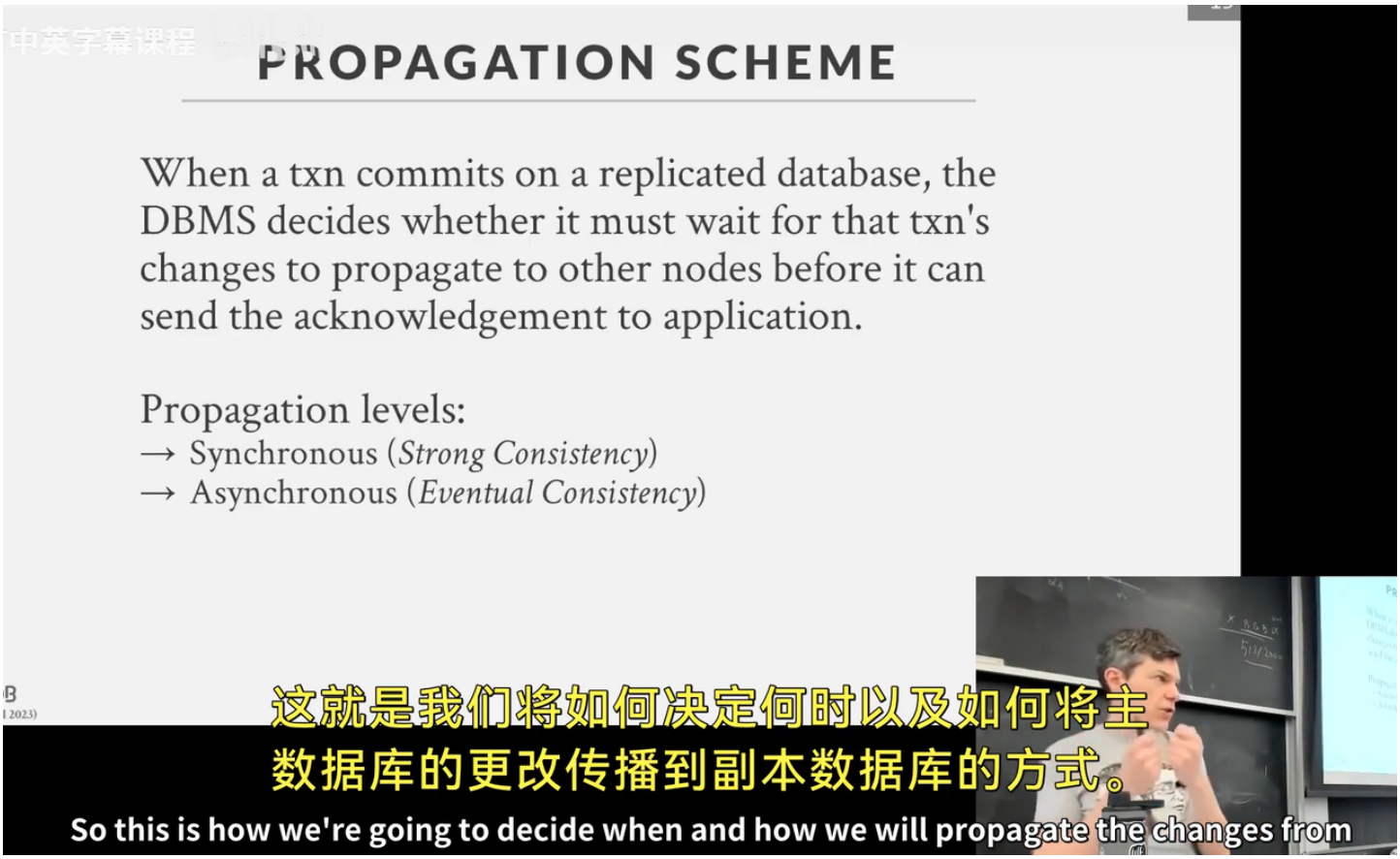
- 读写都发给主，写回传播到副本（只是WAL）
- 中间件可以将只读操作路由到副本上查询



> 对于数据库给定对象的所有副本，我们可以容忍多少节点发生故障
>
> K是可用副本数

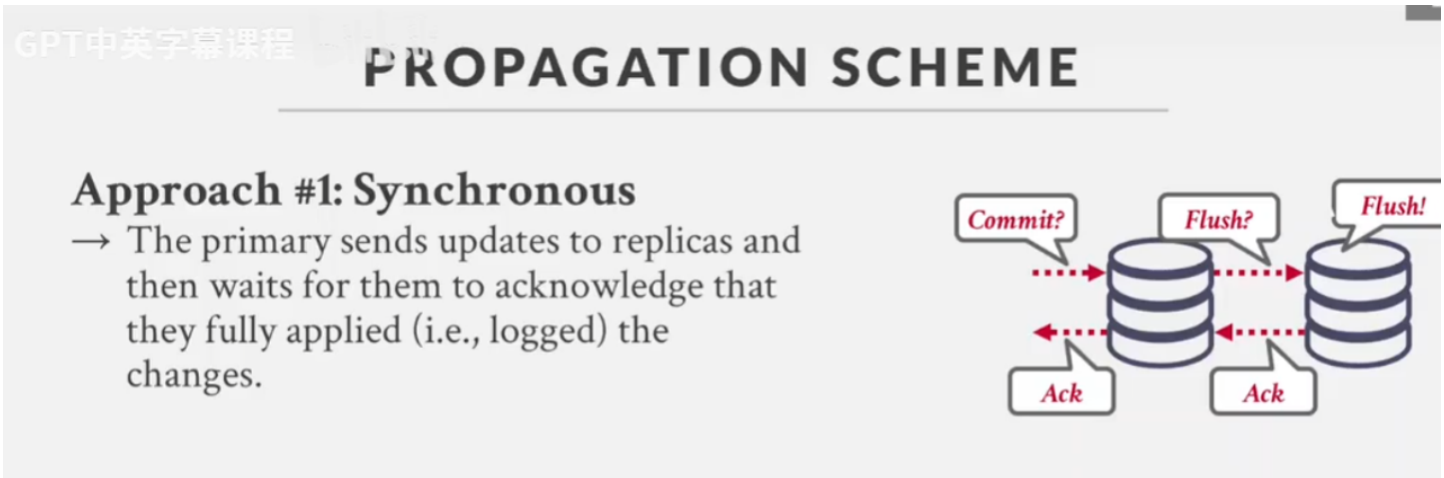


传播方案



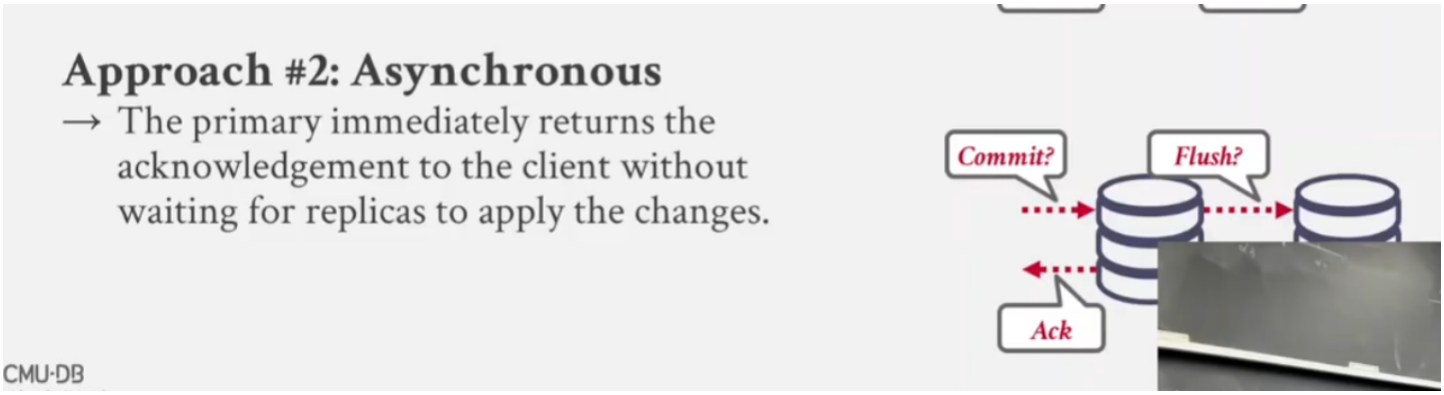
同步（强一致性）

> 需要等待副本成功写入后才进行事务提交



异步（最终一致性）

副本接收到直接返回ACK



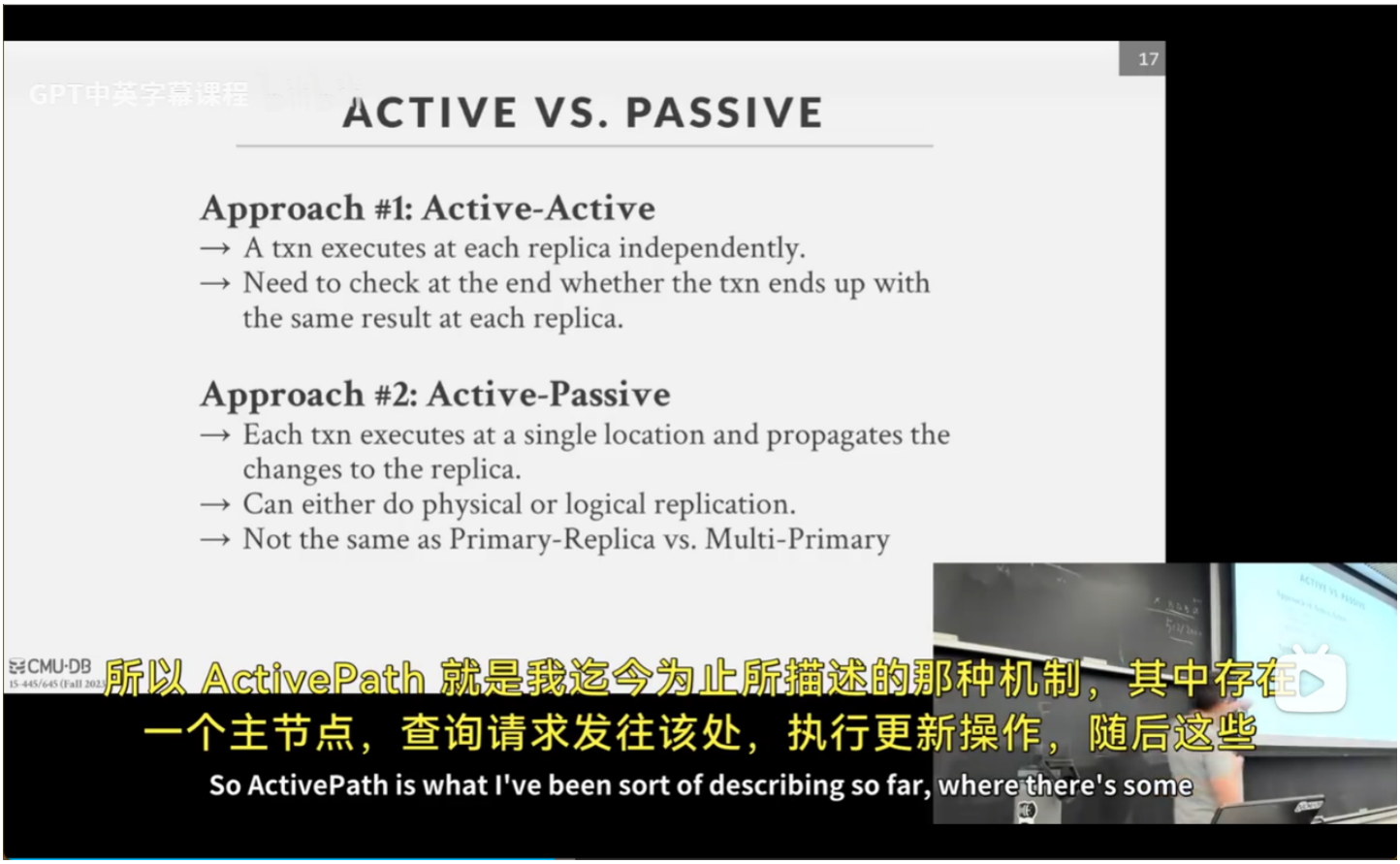
Propagation Timing

连续传播：一旦产生一个更新，立即传播log message

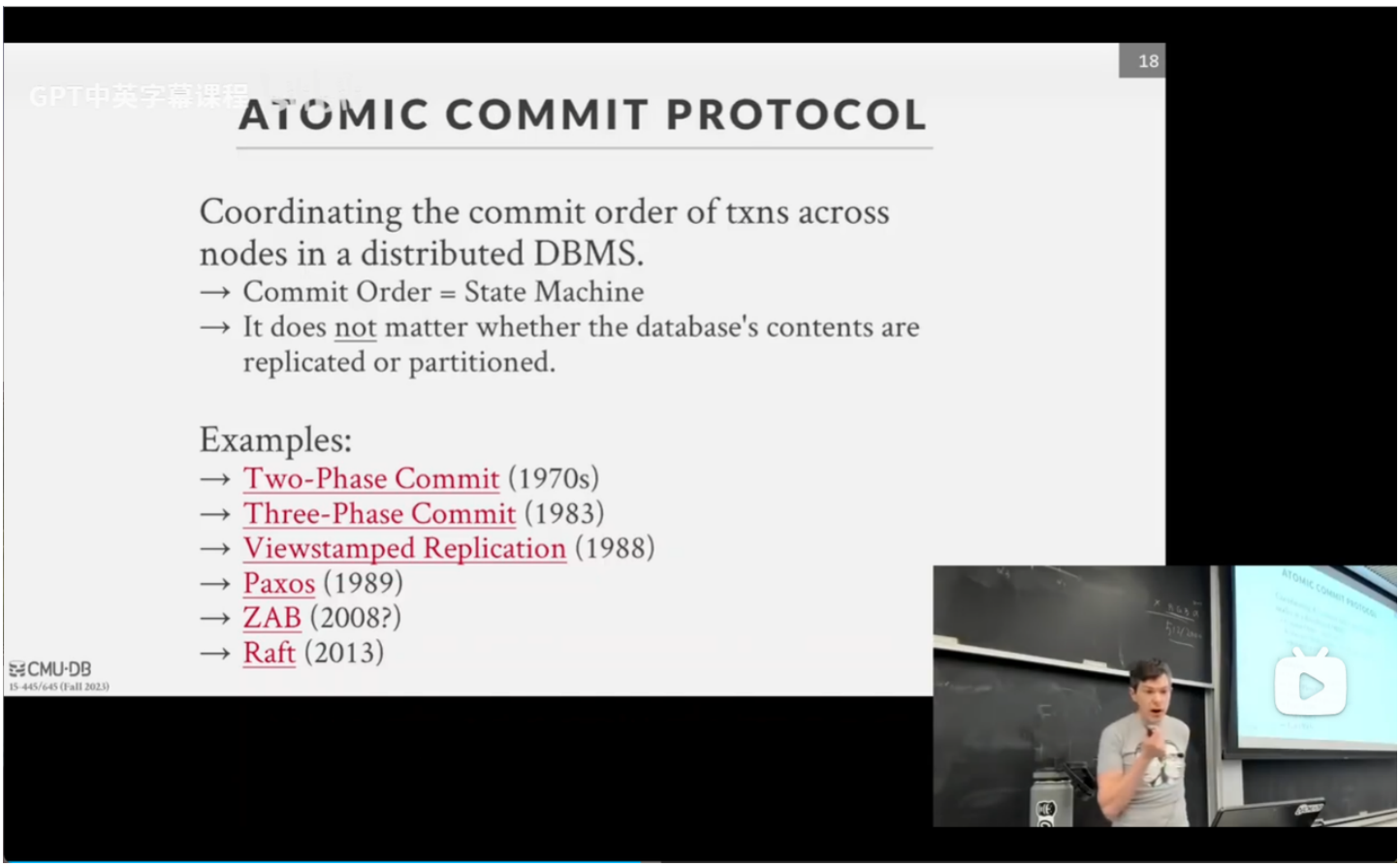
提交传播：等到事务提交时才传播log message

Active-Active：每个事务在每个副本独立执行，破坏只读查询的效率性

Active-Passive：每个事务只在本地修改并传播到其他副本



Atomic Commit Protocol

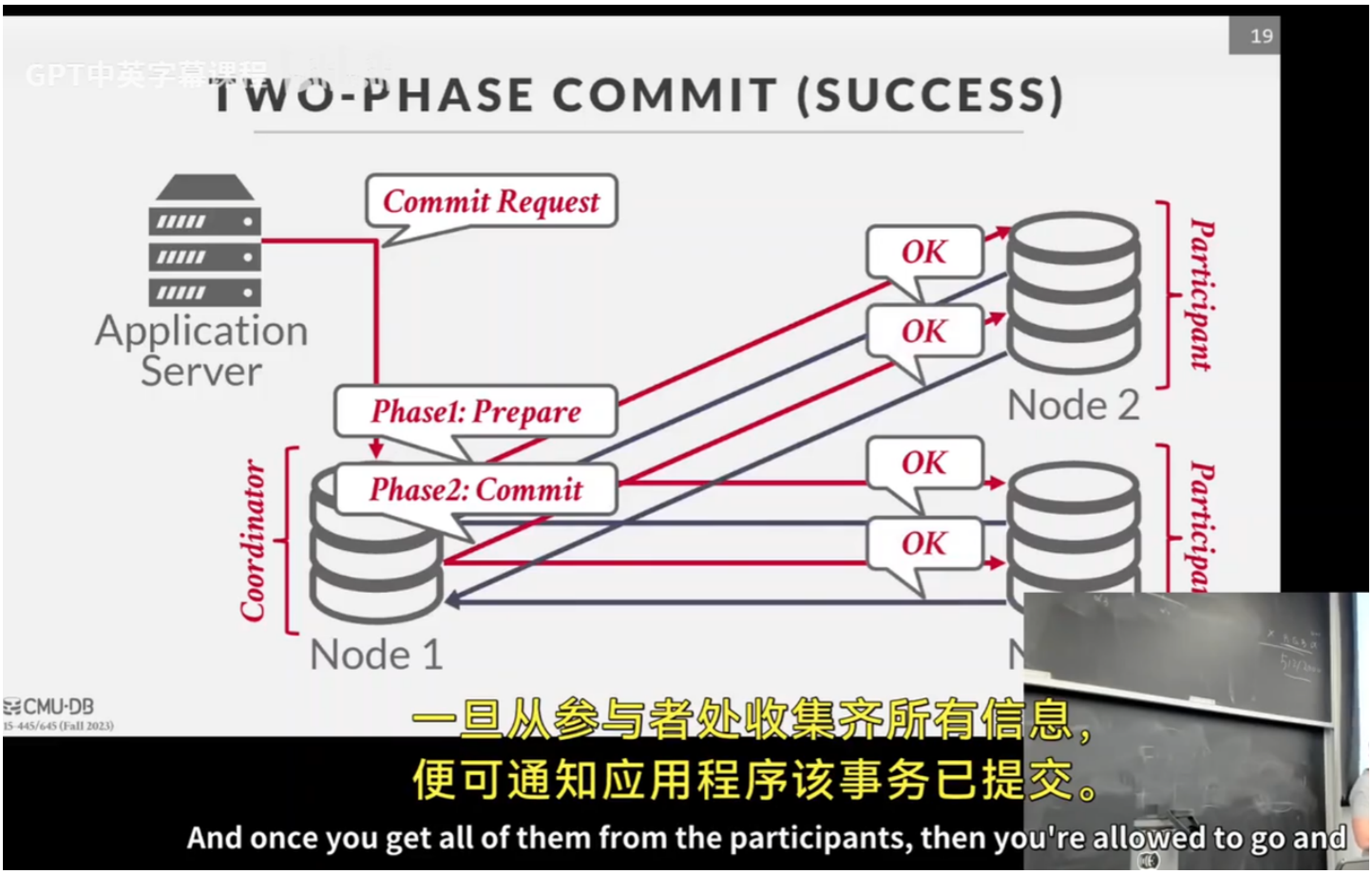


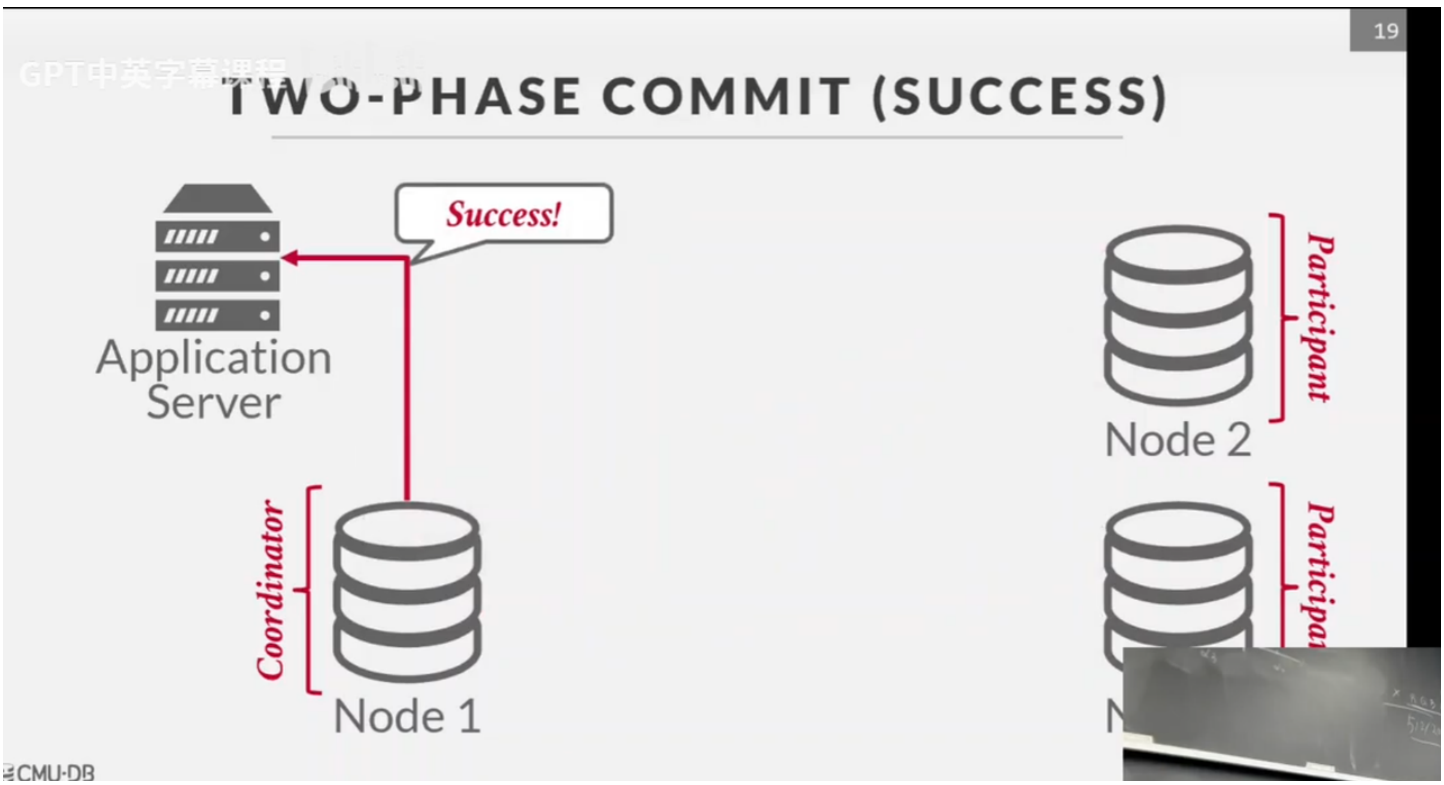
Two-Phase Commit

> 我们必须等待所有参与者返回确认信息后，才能进行下一阶段

> 任何一个节点都可能导致整个系统崩溃

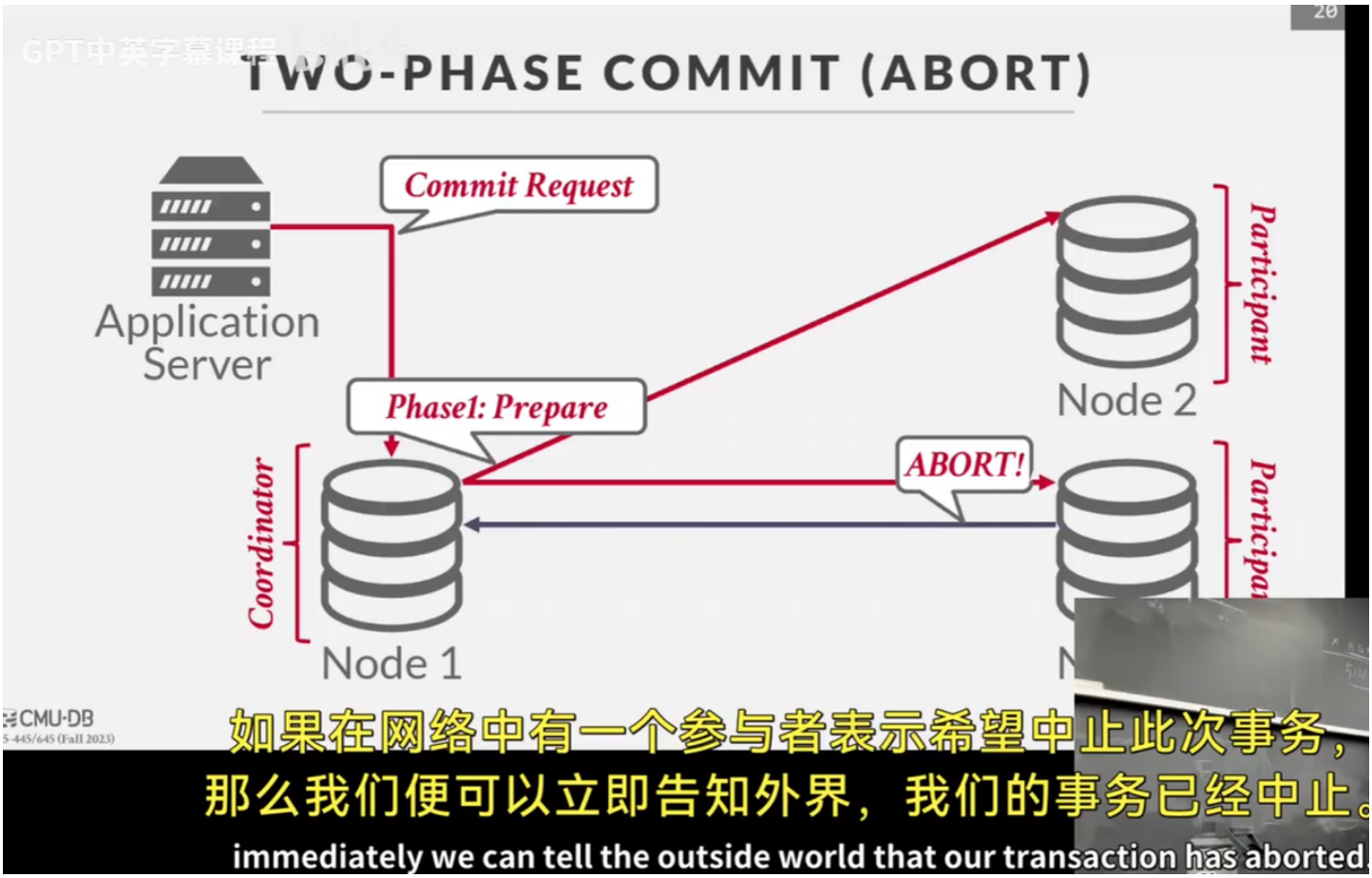
success

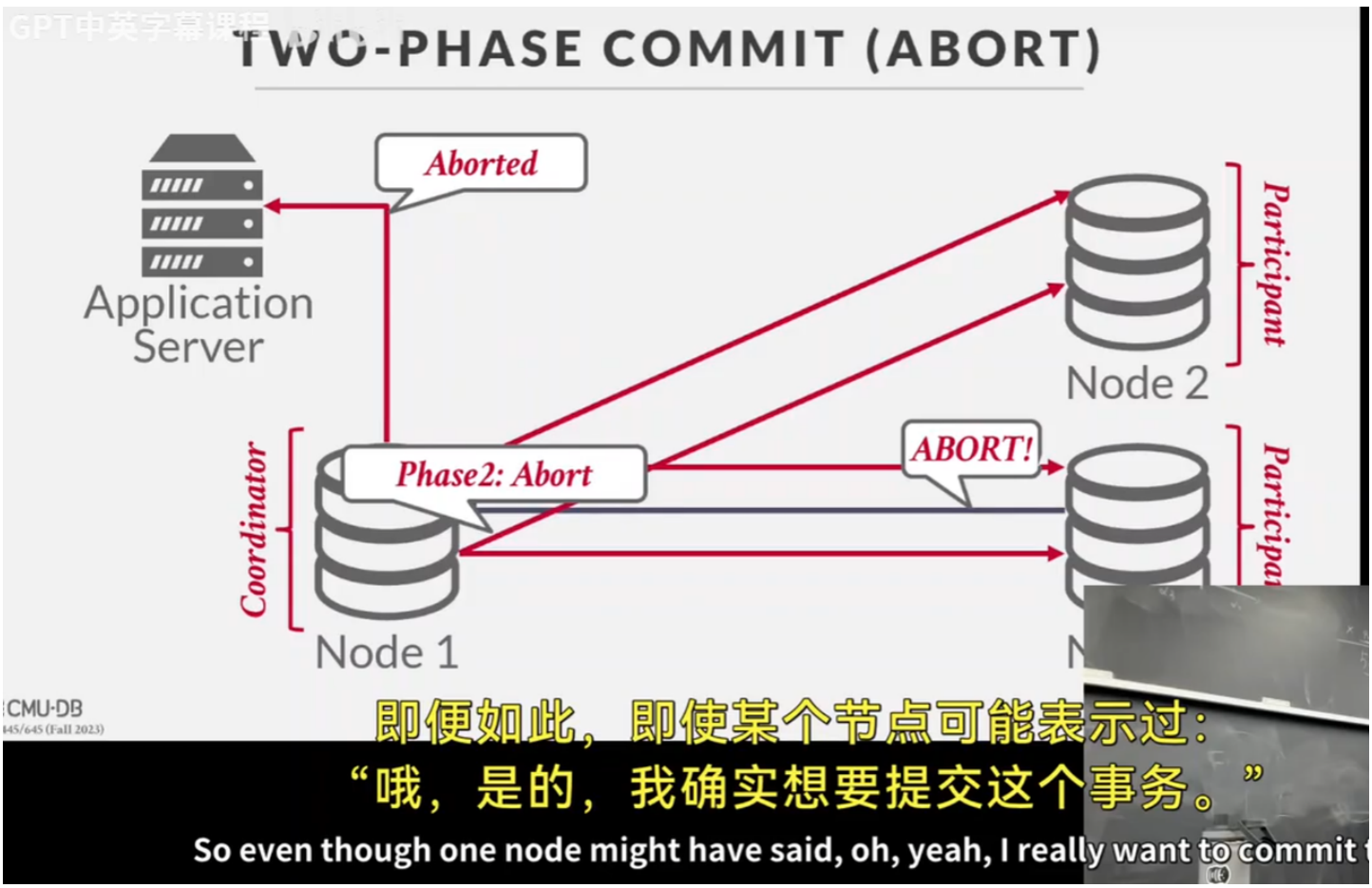




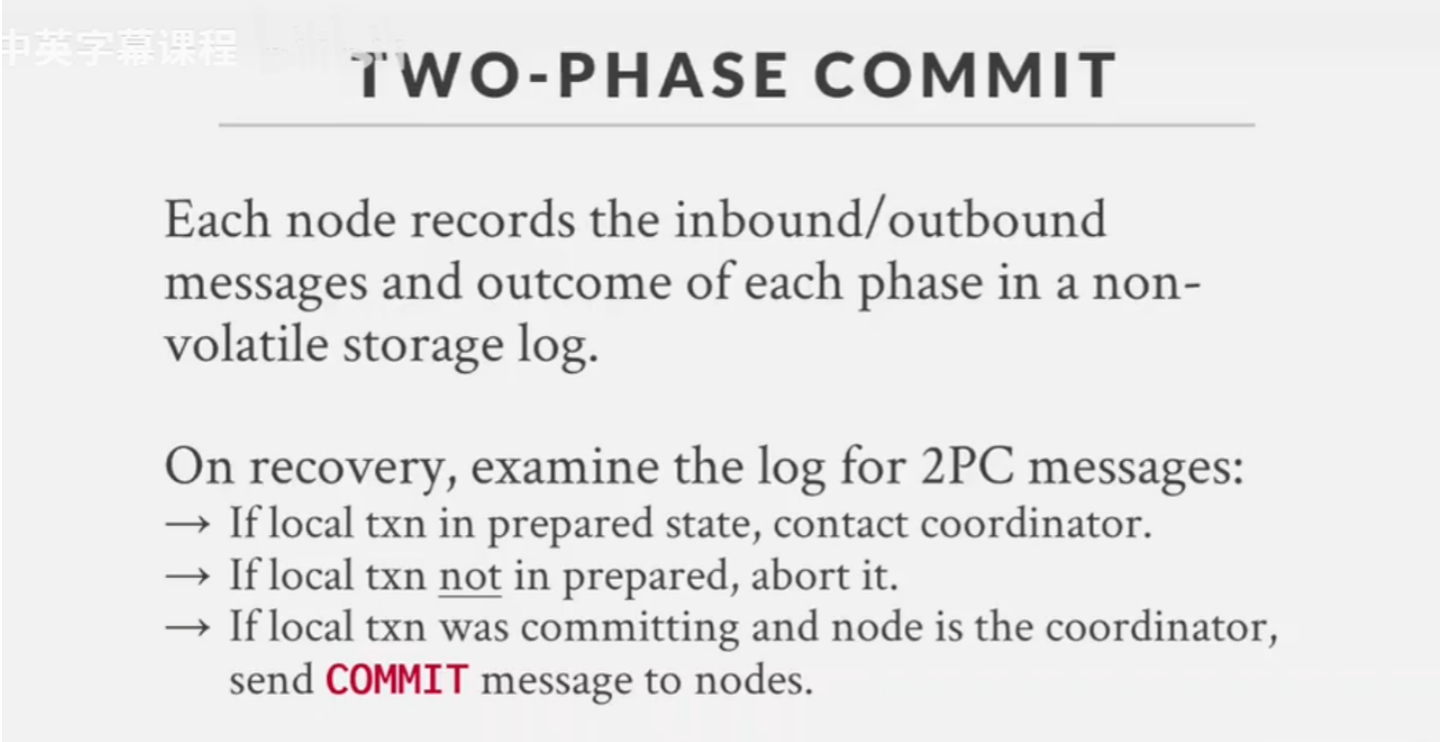
abort

> 当收到abort，直接向application server回复aborted


TWO-PHASE COMMIT (ABORT)
Commit Request
Application Server
Phase1: Prepare
Coordinator
Node 1
ABORT!
Node 2
Participant
CMU-DB
如果在网络中有一个参与者表示希望中止此次事务，
那么我们便可以立即告知外界，我们的事务已经中止。
immediately we can tell the outside world that our transaction has aborted


TWO-PHASE COMMIT (ABORT)
Aborted
Application Server
Phase2: Abort
Coordinator
Node 1
ABORT!
Node 2
Participant
CMU-DB
即便如此，即使某个节点可能表示过：
“哦，是的，我确实想要提交这个事务。”
So even though one node might have said, oh, yeah, I really want to commit

recovery



Failures

> 不可用：无法接受任何新的查询，也无法接受任何写入

2PC Optimizations

> 一般采取early ack after prepare

2PC OPTIMIZATIONS

Early Prepare Voting *(Rare)*

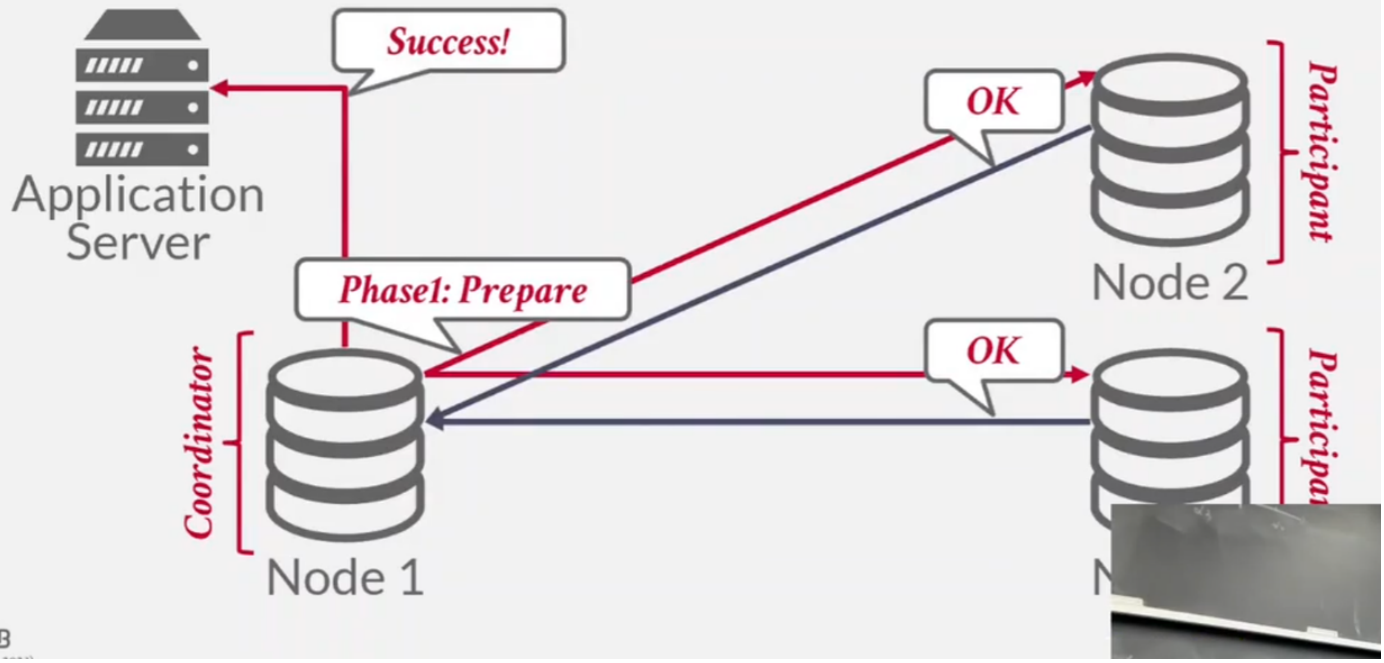
→ If you send a query to a remote node that you know will be the last one you execute there, then that node will also return their vote for the prepare phase with the query result.

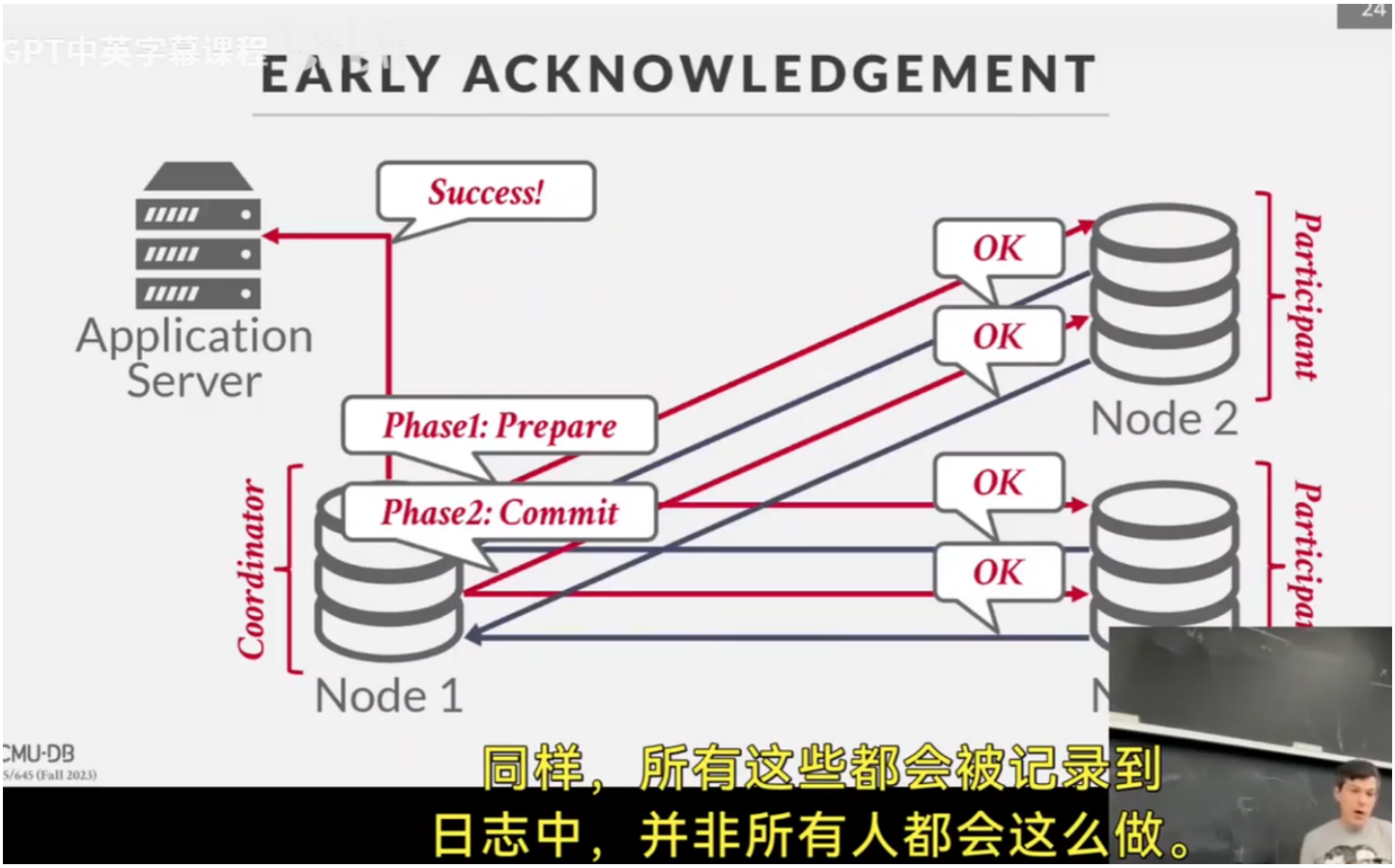
Early Ack After Prepare *(Common)*

→ If all nodes vote to commit a txn, the coordinator can send the client an acknowledgement that their txn was successful before the commit phase finishes.

Early ACK After Prepare

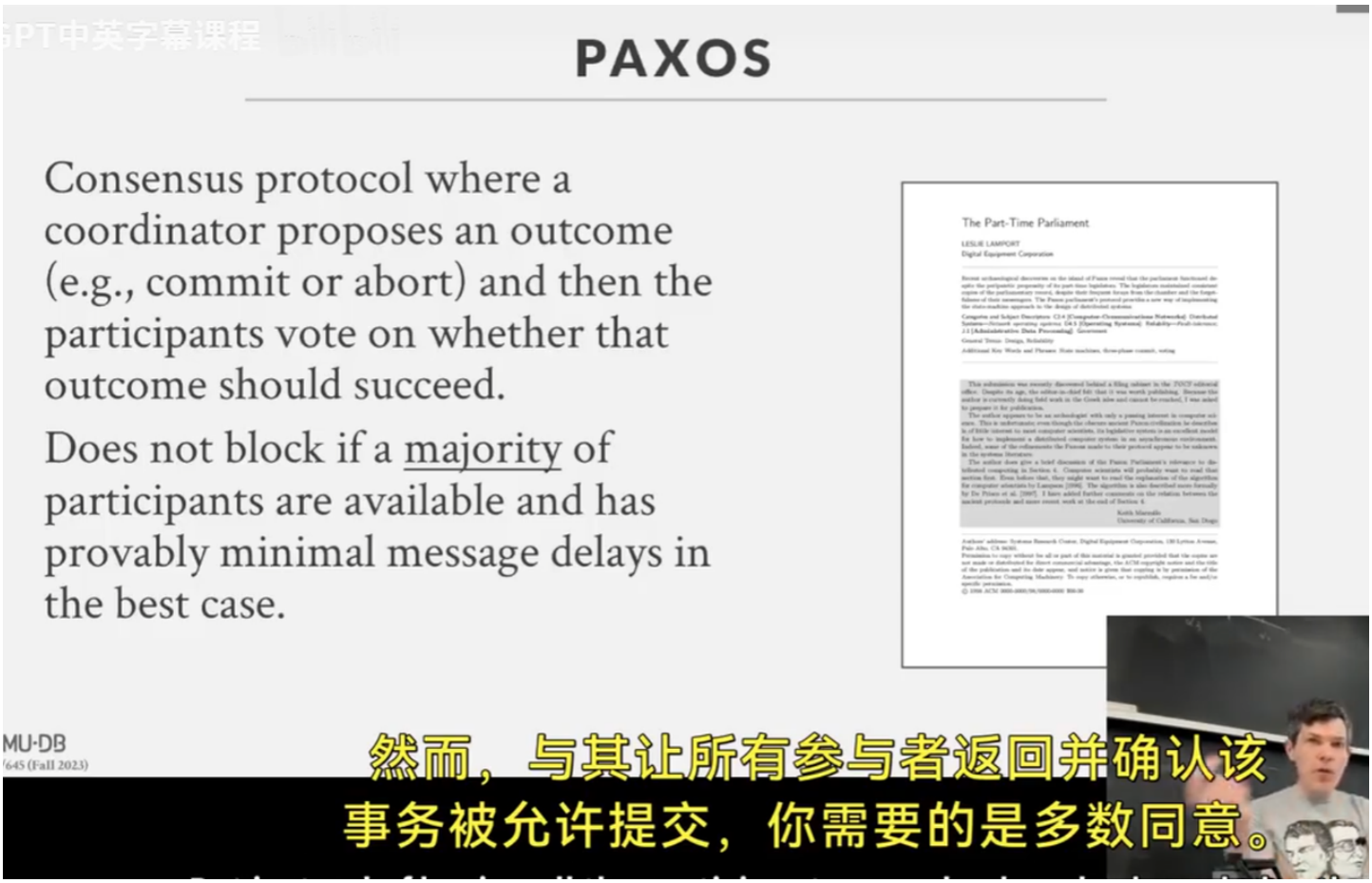
EARLY ACKNOWLEDGEMENT



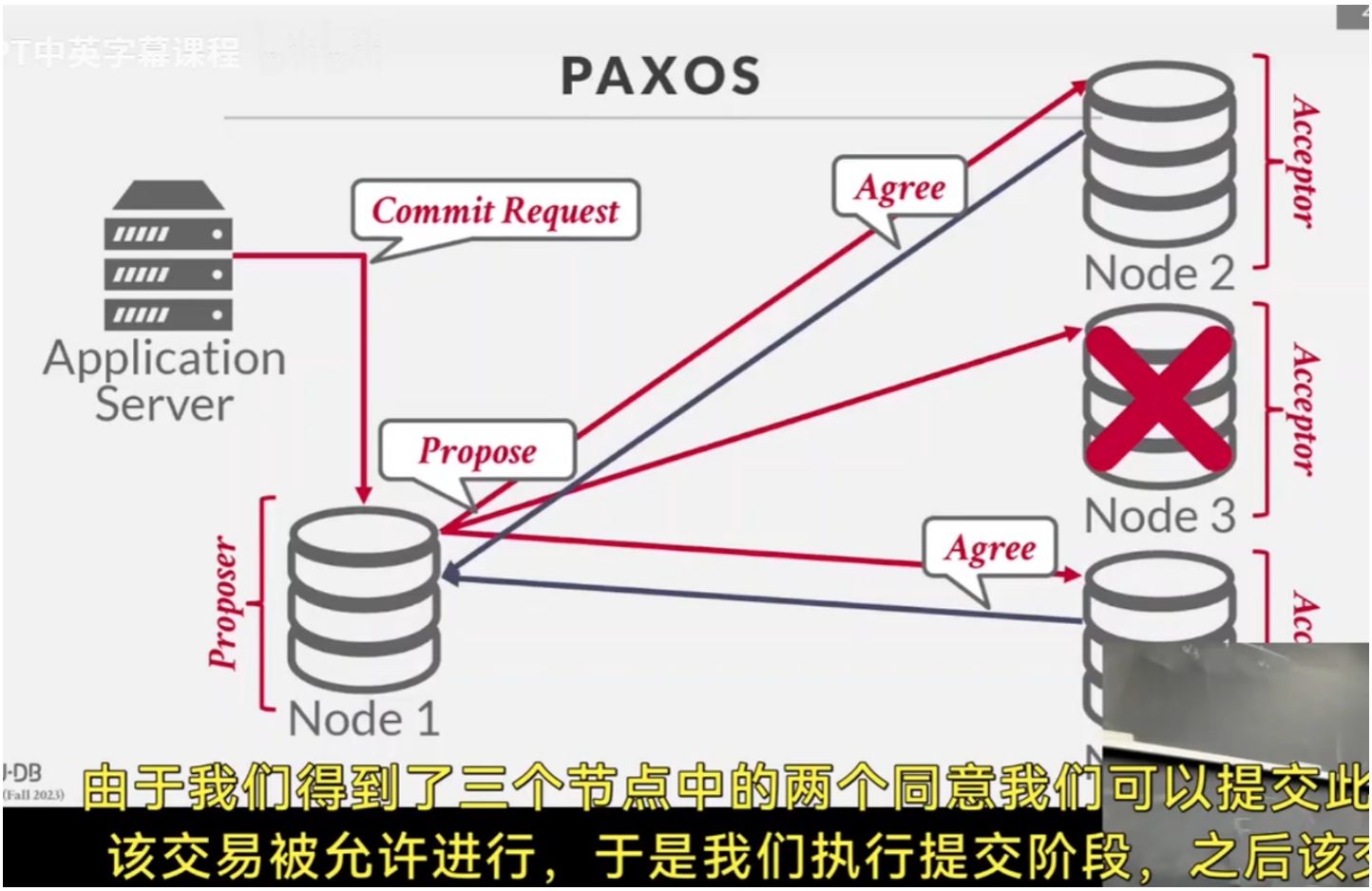


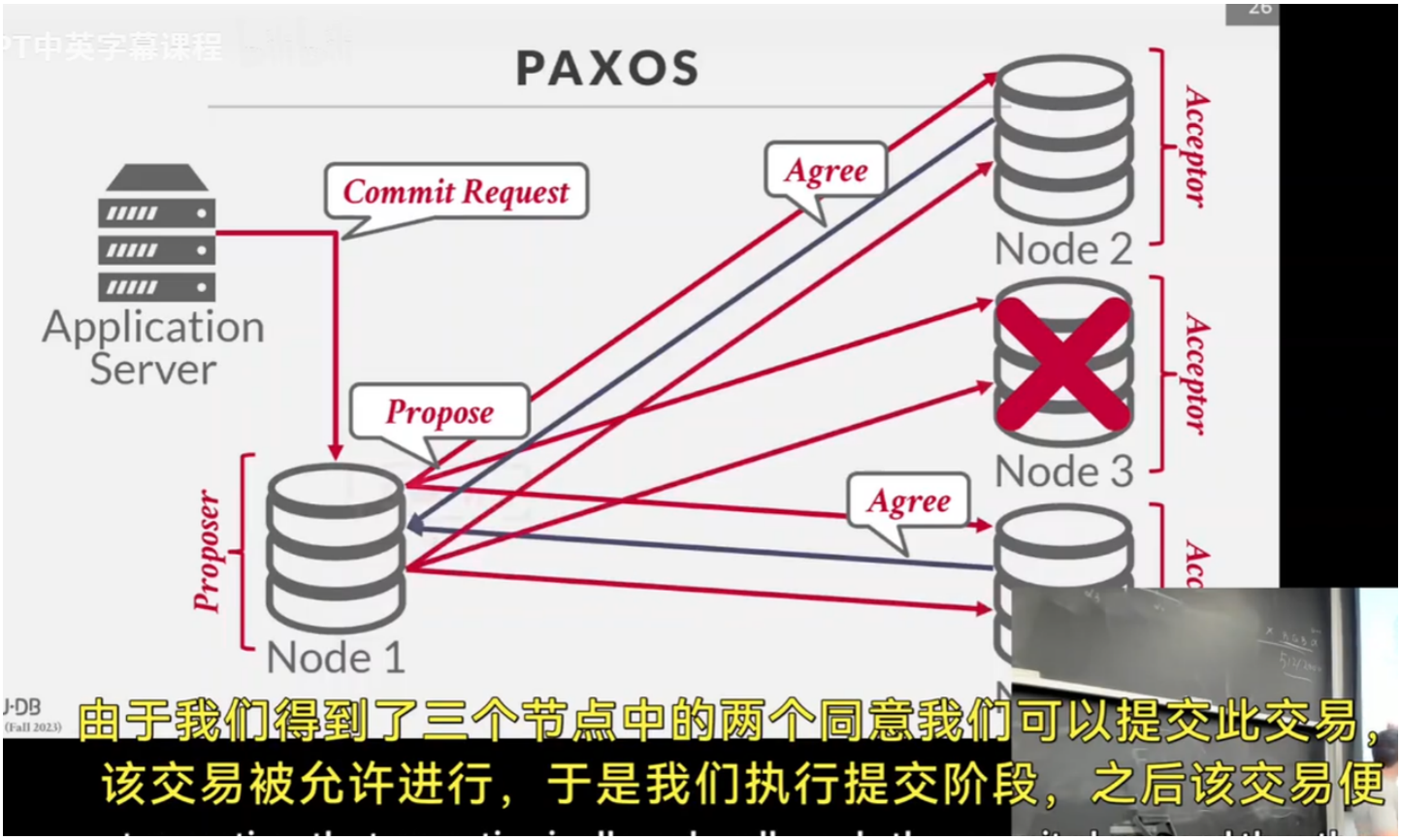
PAXOS

> 多数投票者同意即可成功；
>
> 少数派成员被认为失败或者故障，通过重放日志来恢复正确的状态

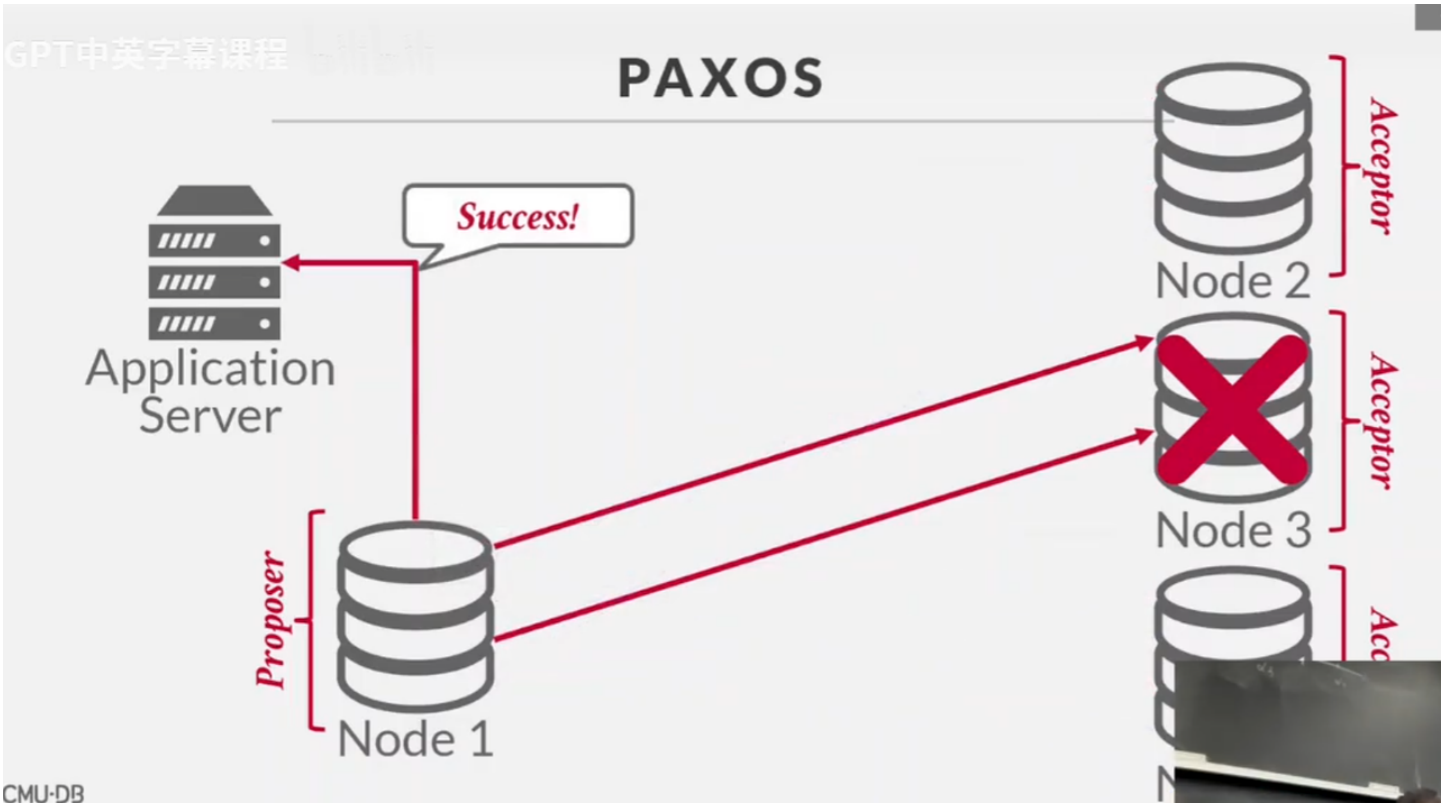


流程

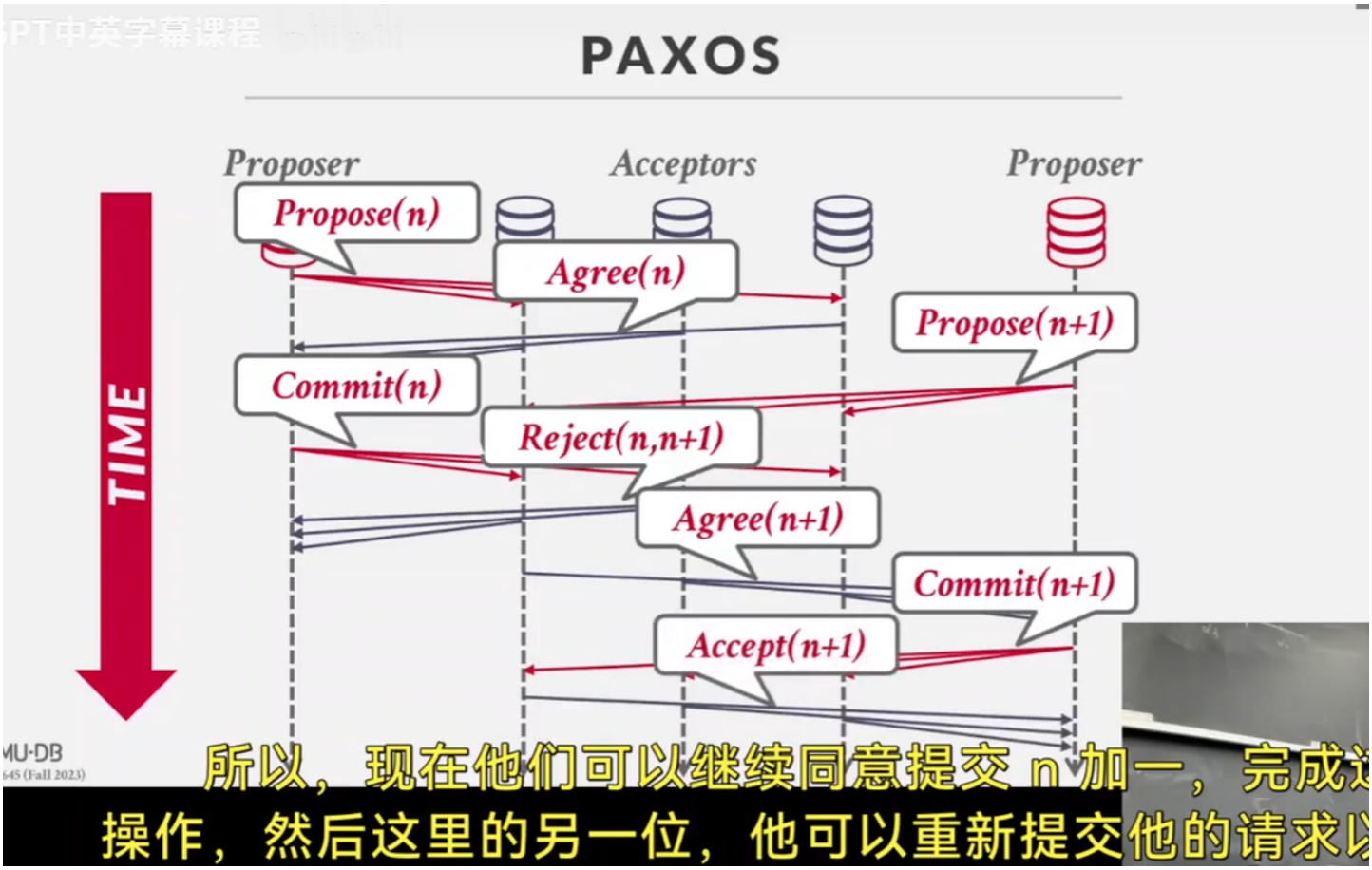




> Node 3重新上线后，可以重放日志获取到新的变更



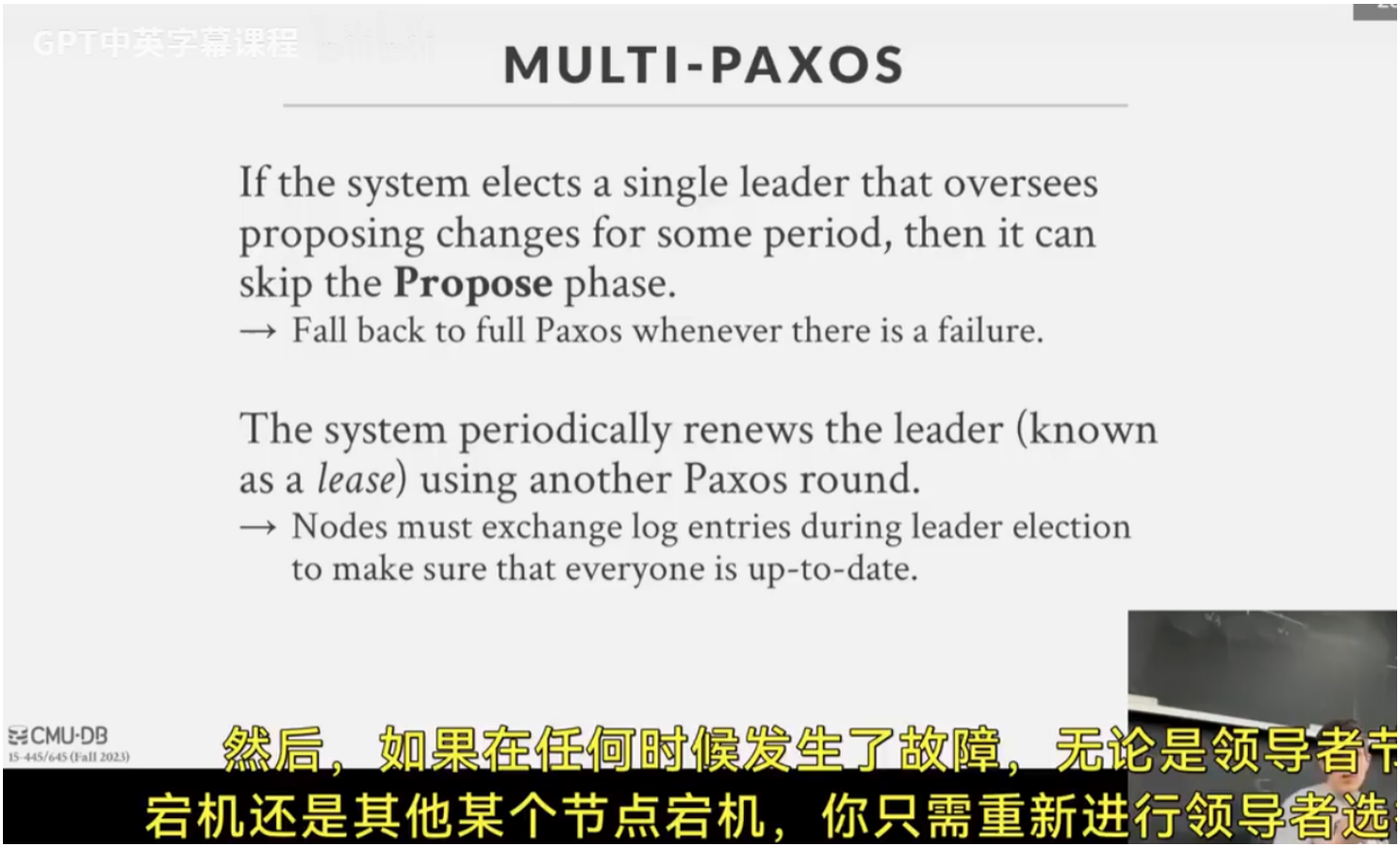
Time



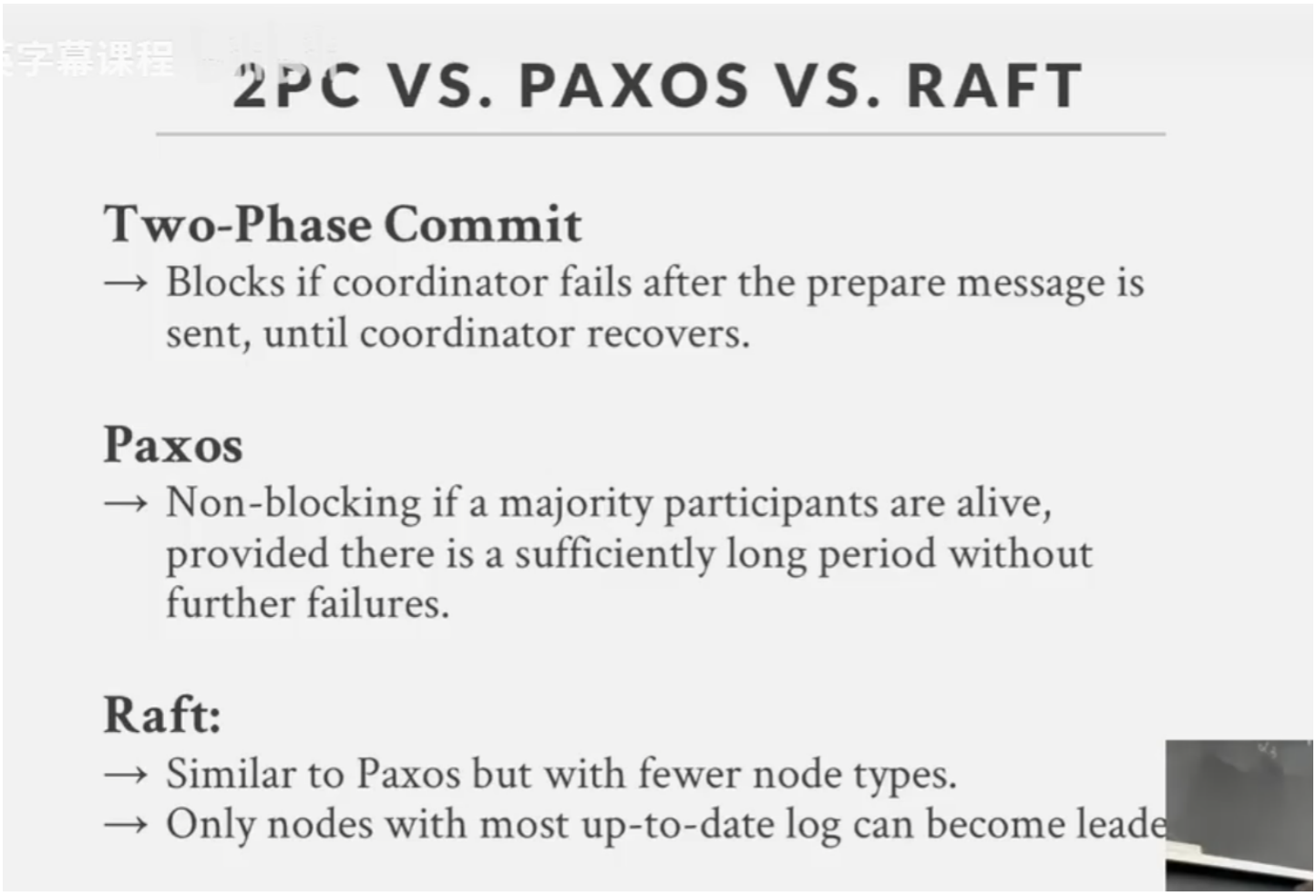
Multi-Paxos

> 如果系统选举出一个单一领导者，在一段时间内监督提出变更，则可以跳过 Propose 阶段。
>
> 当出现故障时退回到完整的 Paxos 协议。

可以设置自动超时时间，超时后会自动重新进行一次领导者选举



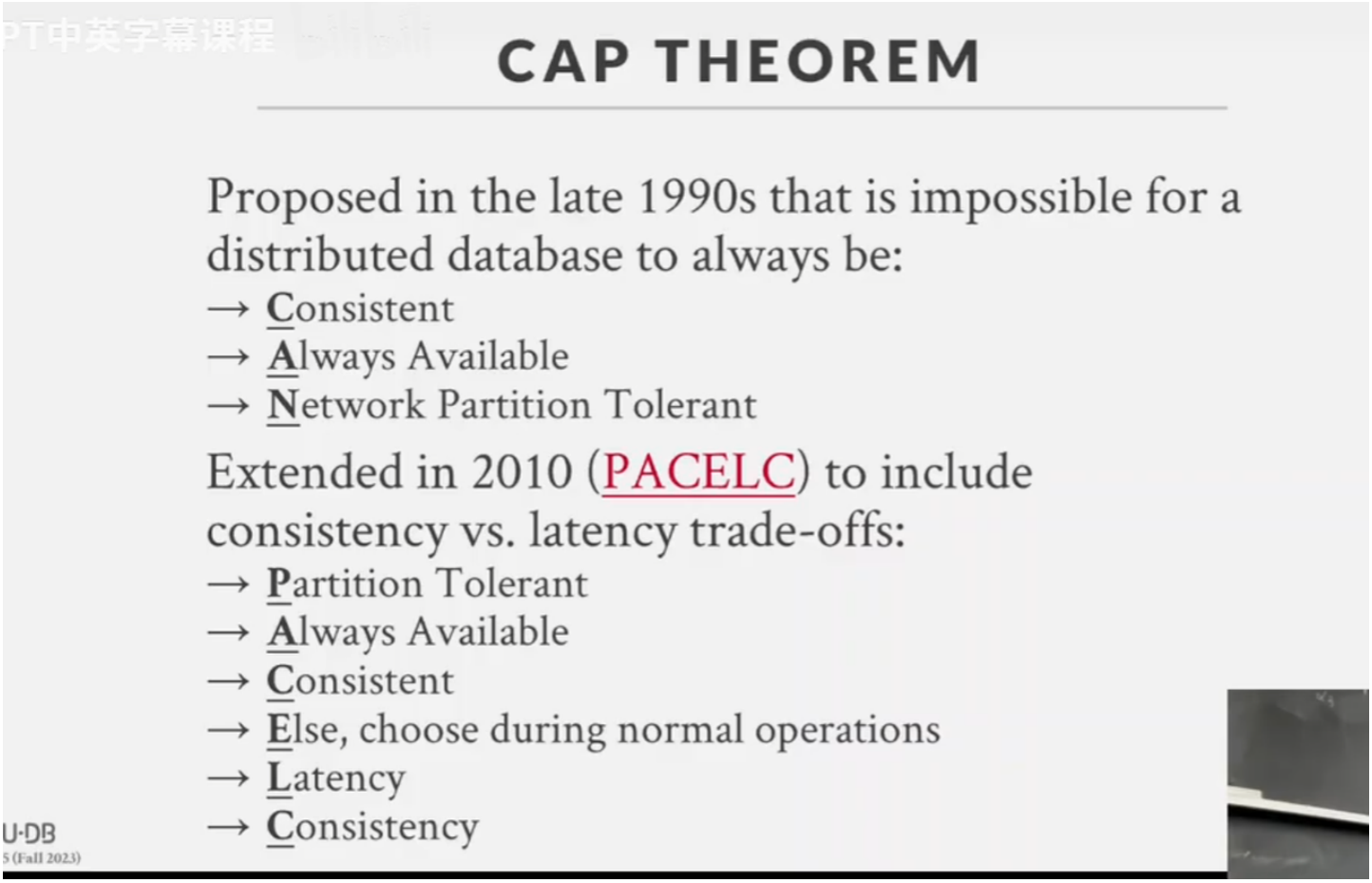
Multi paxos有领导者，而paxos没有领导者，任何人都可以是领导者、参与者；但是彼此的提交会相互覆盖，需要采取退让策略



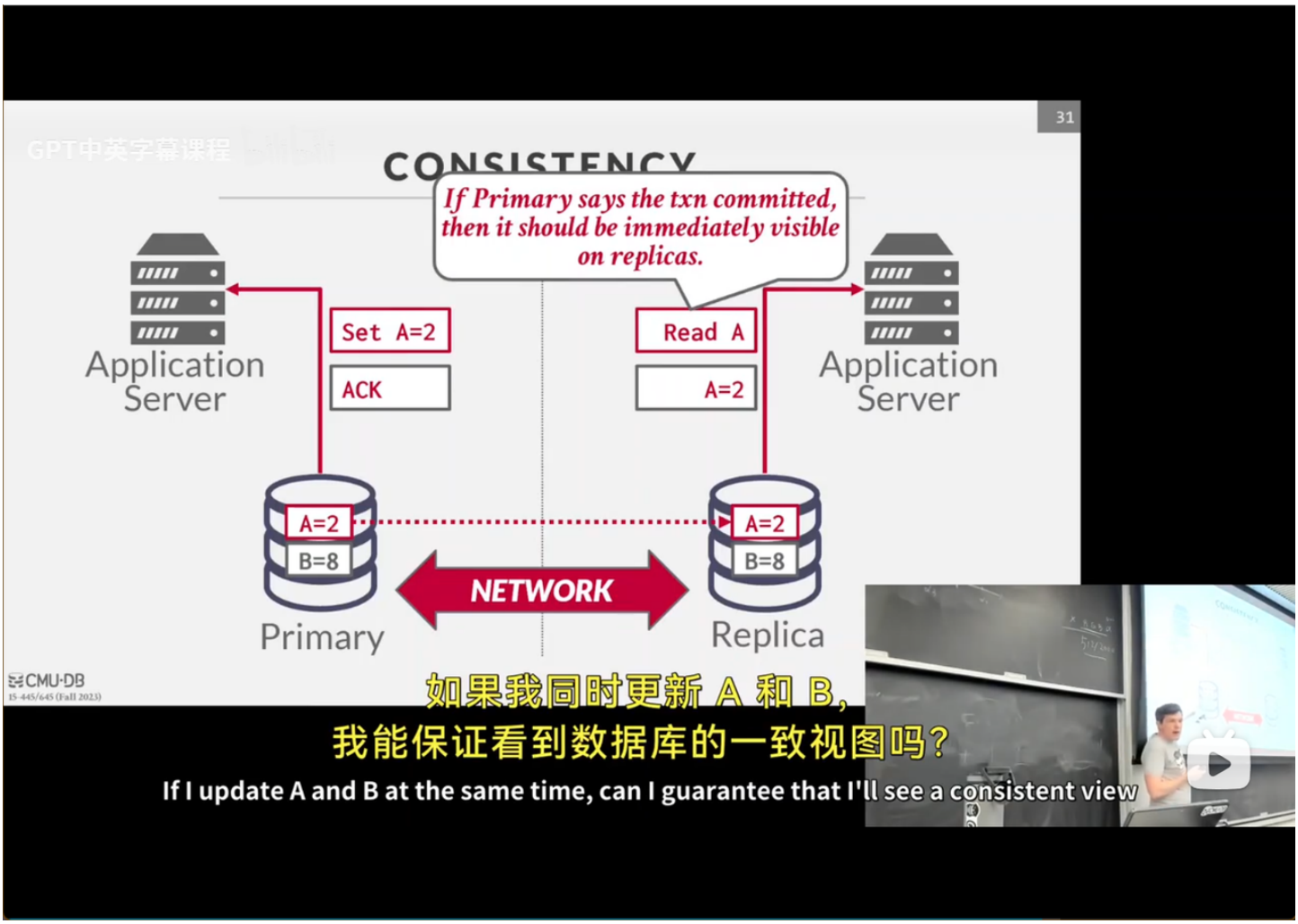
CAP Theorem

分布式数据库只能从CAP中选择两个条件满足

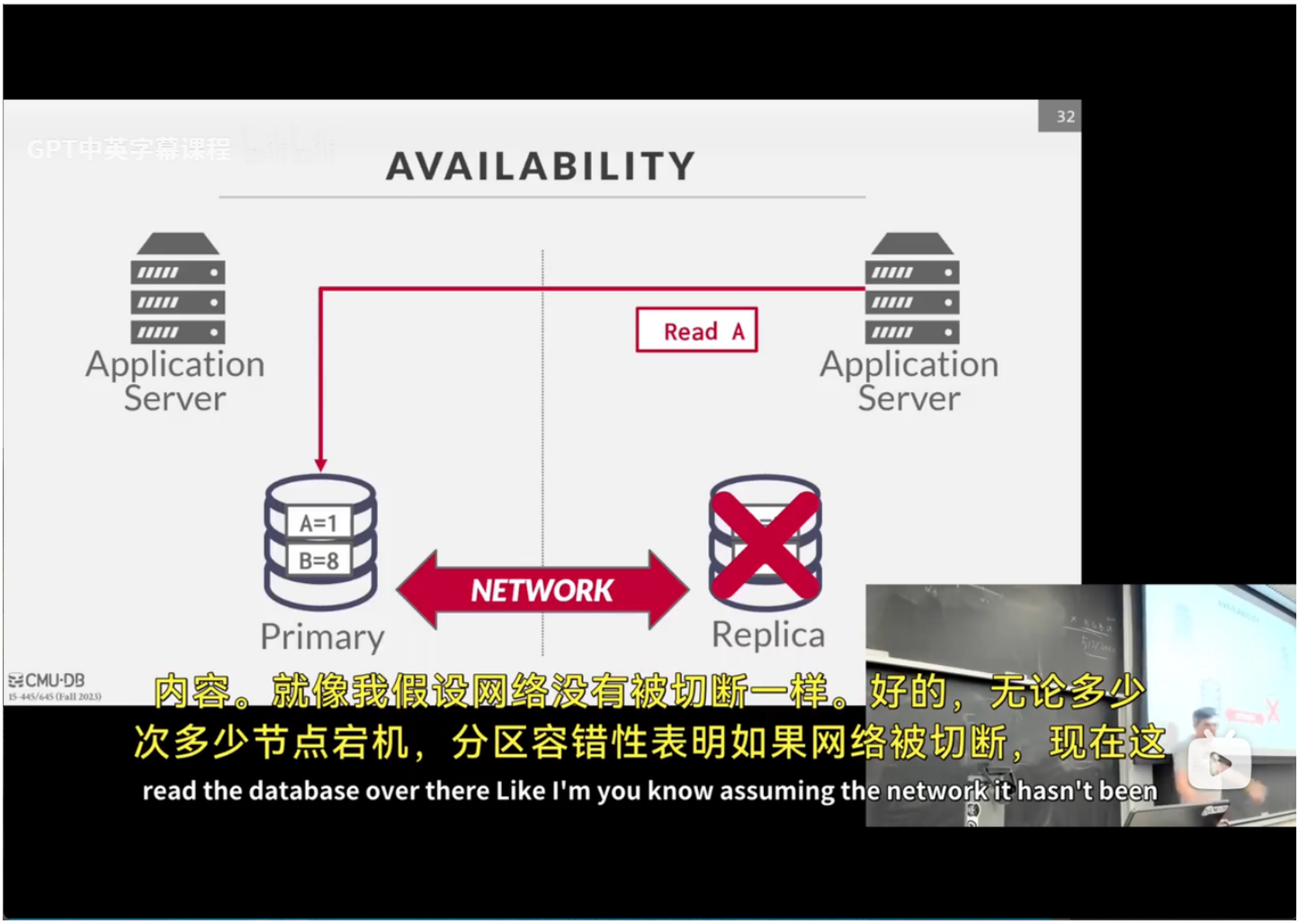
- C是一致性
- A是Always Available
- N是Network Partition Tolerant



Consistency



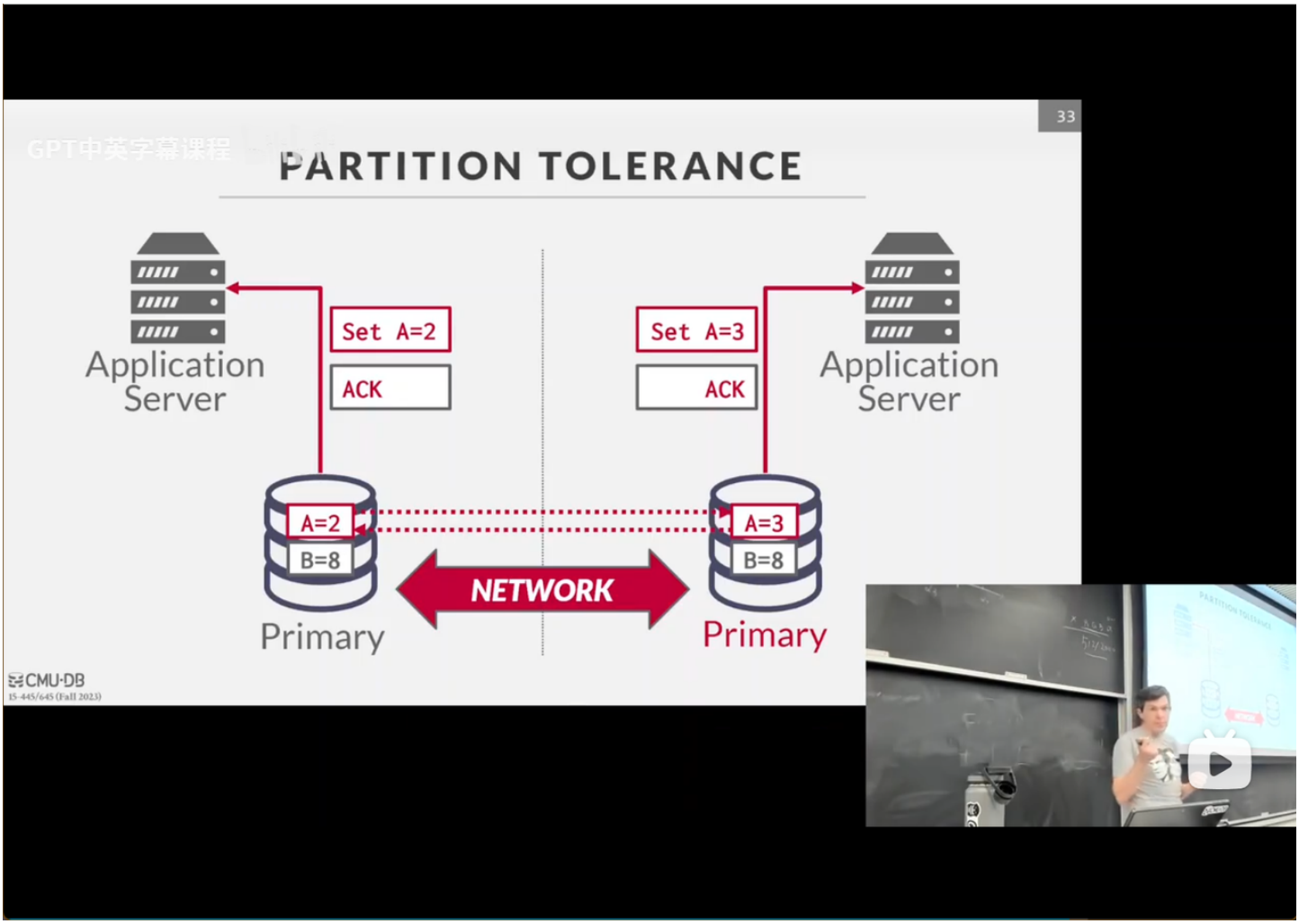
Availability



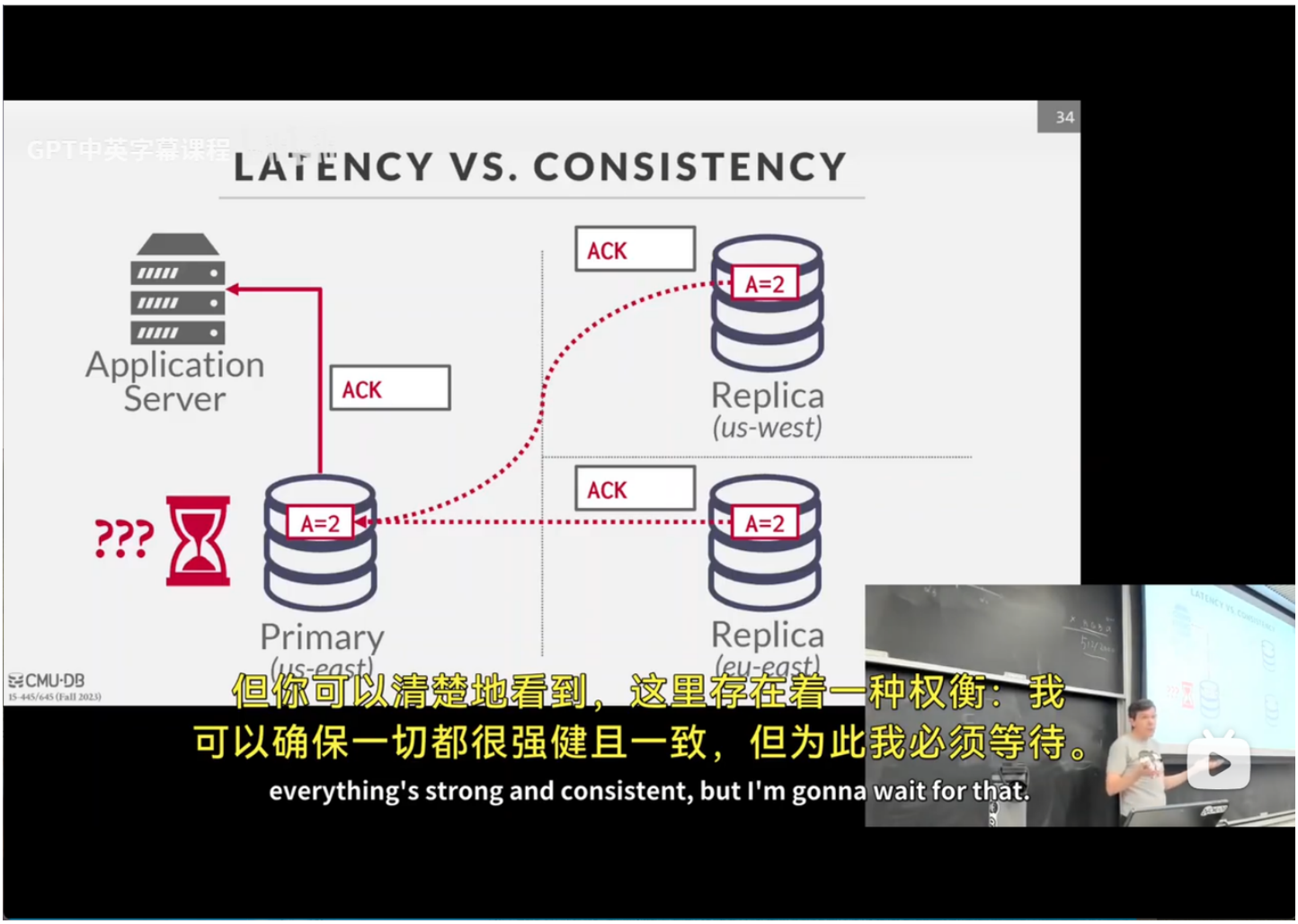
Partition Tolerance

- Split brain

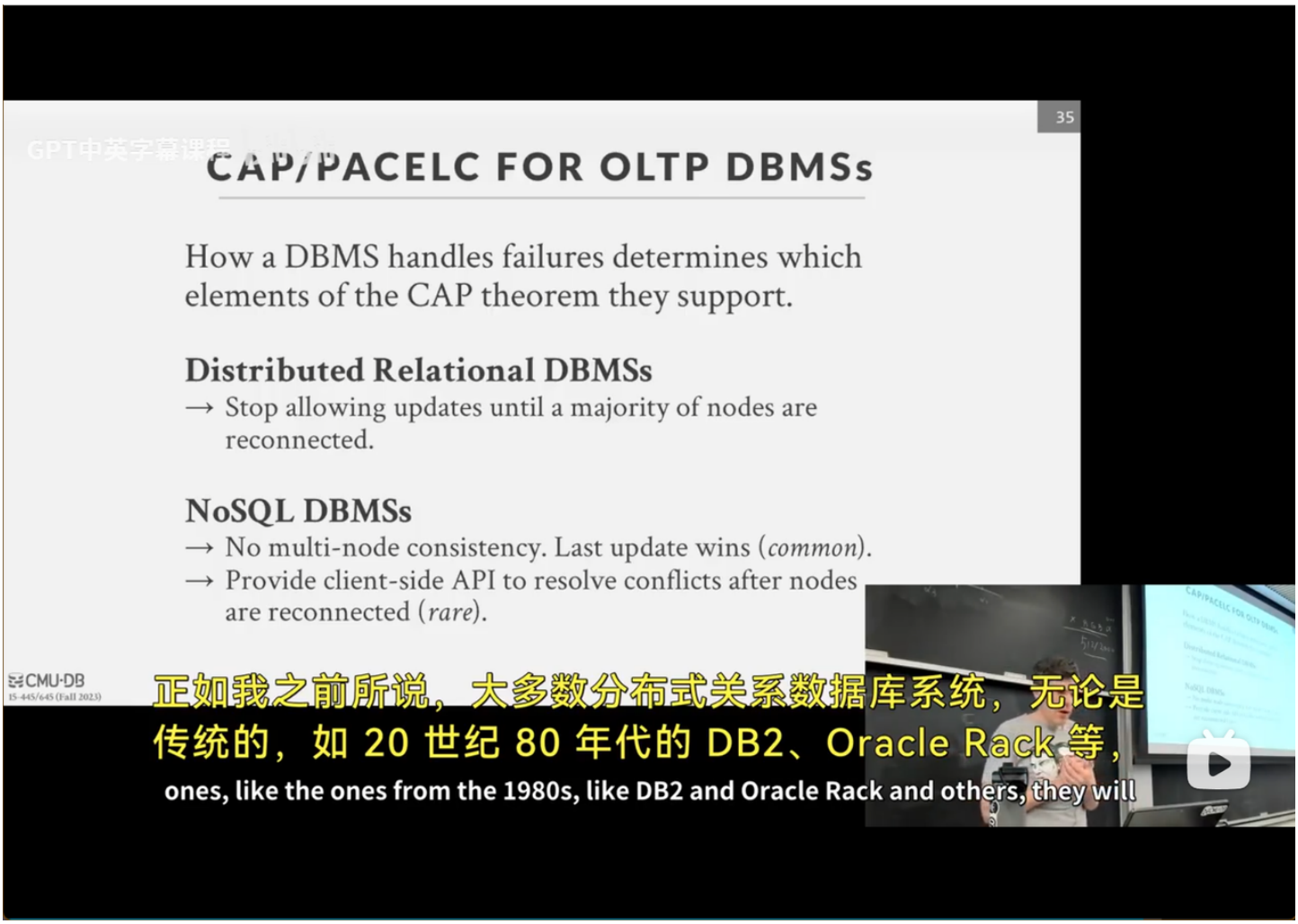
> 两个节点通信断开，副本节点选举自己变成主节点并进行数据更改；然后两个节点恢复通信，两个节点均认为自己是主节点
>
> NOSQL解决方案：类似多版本控制，提出了向量时钟，直接选取版本号最高的值作为查询结果



- 为了确保consistency，我们必须等待

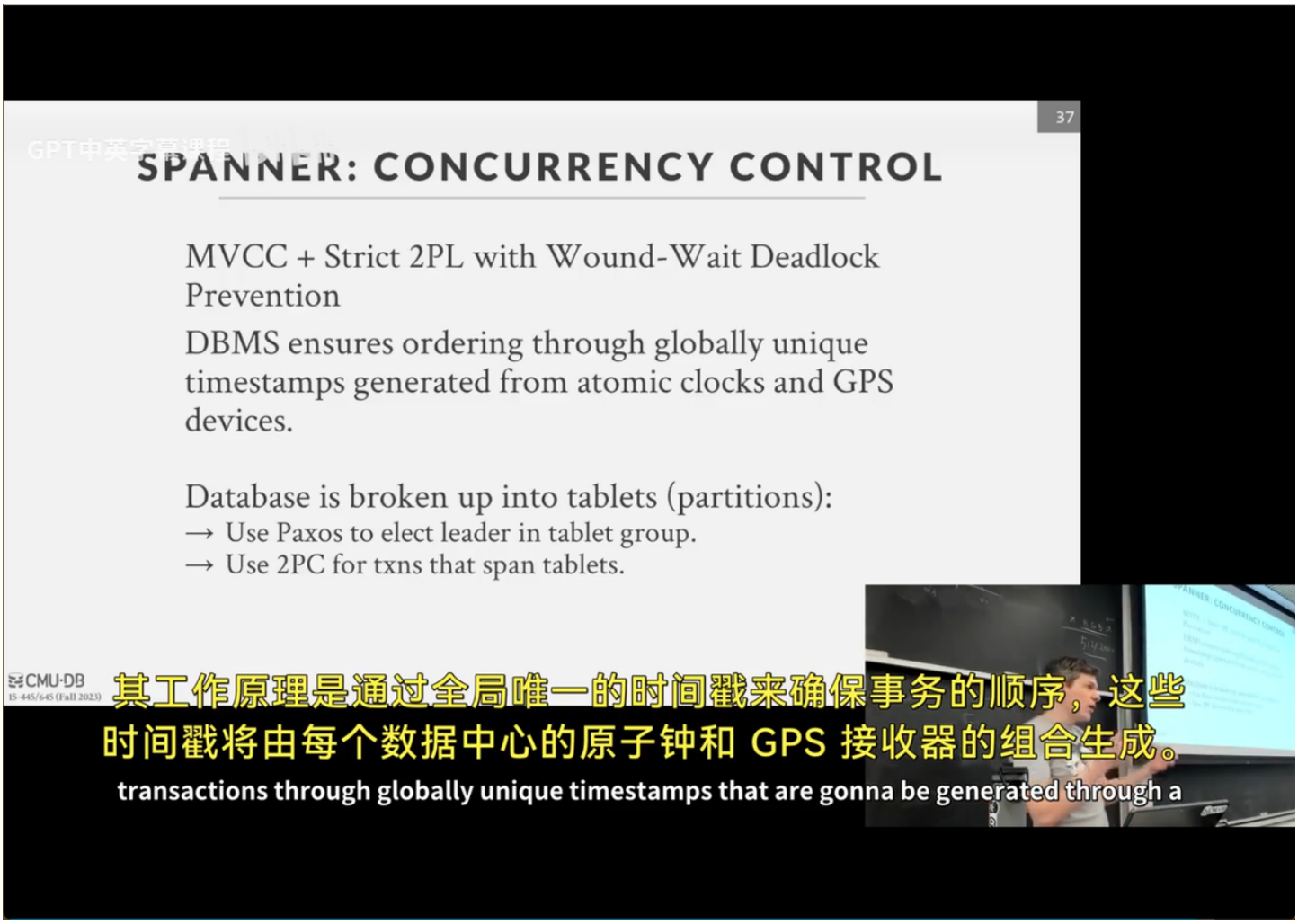


> 传统数据库要求必须有主节点重连才能进行更新
>
> NoSQL一般选择最新的更新

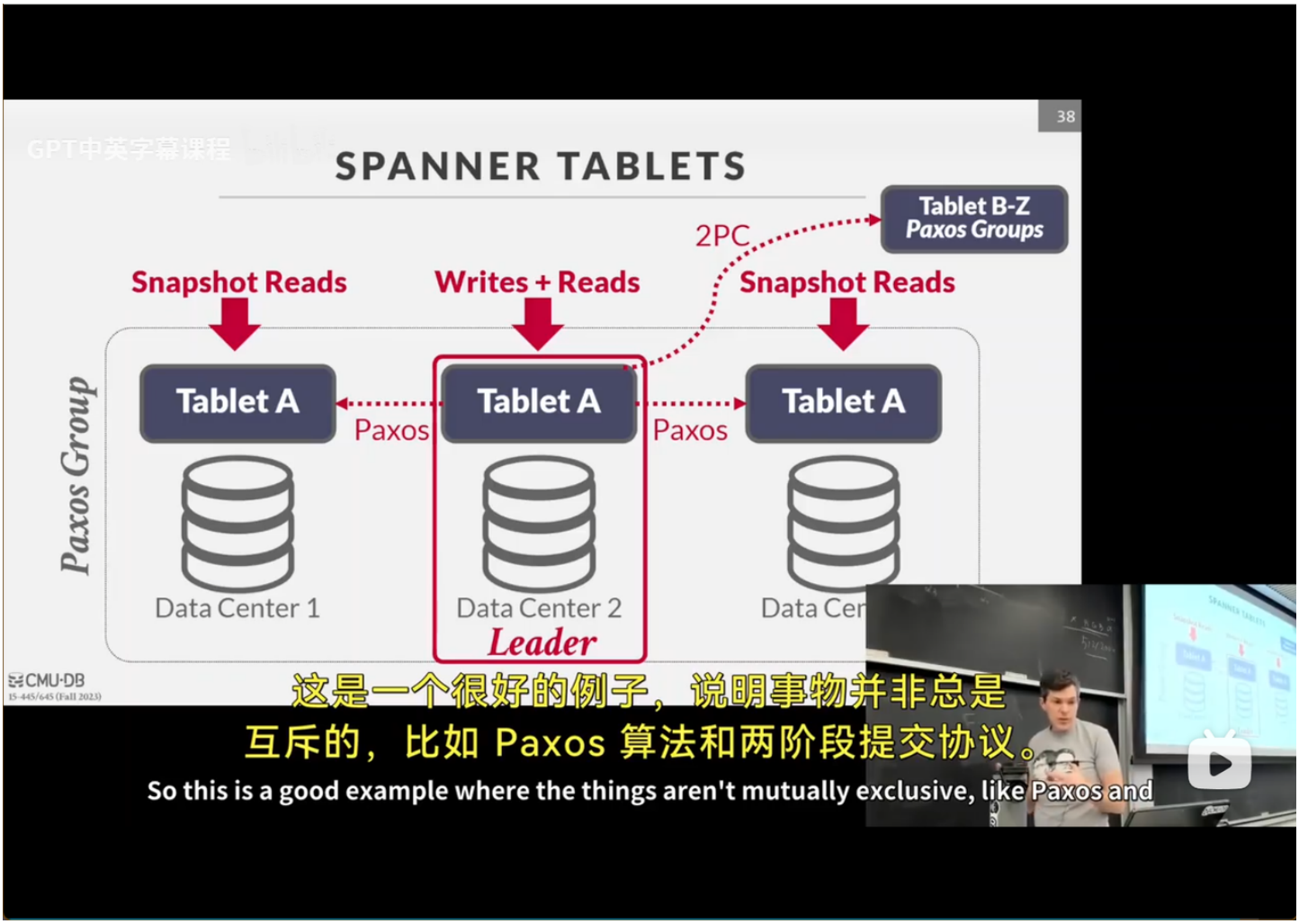


Google Spanner

> 通过全局唯一的时间戳来确保事务的顺序，时间戳由每个数据中心的原子钟和GPS接收器组合生成



> 一个事务需要更新另一个tablet组，需要使用**两阶段提交来执行更新操作**；这些更新随后被传播到其他tablet，进而传递给它们的leader；leader通过paxos算法来保证consistency



> 如何确保严格串行化和时间采样

SPANNER: TRANSACTION ORDERING

DBMS orders transactions based on physical "wall-clock" time.

→ This is necessary to guarantee strict serializability.

→ If T_1 finishes before T_2, then T_2 should see the result of T_1.

Each Paxos group decides in what order transactions should be committed according to the timestamps.

→ If T_1 commits at $time_1$ and T_2 starts at $time_2$ > $time_1$, then T_1's timestamp should be less than T_2's.

这基本上说明了如何确保严格串行化以及进行时间采样。

This basically says how to guarantee strict serializability and do time sampling.