**Example of Scalability Problem**

Customer 2 buys widget 3 with store credit:

\[
\begin{align*}
W &= \text{Widgets.READ}(3) \\
C &= \text{Customers.READ}(2) \\
\text{IF} (W.\text{In}_\text{Stock} < 1) & \text{ ABORT} \\
\text{IF} (C.\text{Store}_\text{Credit} < W.\text{price}) & \text{ ABORT} \\
W.\text{In}_\text{Stock} - &= 1 \\
C.\text{Store}_\text{Credit} - &= W.\text{price}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Widget ID</th>
<th>Price</th>
<th>In_Stock</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>99</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>325</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>79</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>199</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer id</th>
<th>Store_Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>500</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>25</td>
</tr>
</tbody>
</table>
Example of Scalability Problem

Customer 2 buys widget 3 with store credit:

```
W = Widgets.READ(3)
C = Customers.READ(2)
IF (W.In_Stock < 1)
    ABORT
IF (C.Store_Credit < W.price)
    ABORT
W.In_Stock -= 1
C.Store_Credit -= W.price
```
Example of Scalability Problem

Customer 2 buys widget 3 with store credit:

\[
\begin{align*}
W &= \text{Widgets.READ}(3) \\
C &= \text{Customers.READ}(2) \\
\text{IF} \ (W.\text{In\_Stock} < 1) &\quad \text{ABORT} \\
\text{IF} \ (C.\text{Store\_Credit} < W.\text{price}) &\quad \text{ABORT} \\
W.\text{In\_Stock} &= 1 \\
C.\text{Store\_Credit} &= W.\text{price}
\end{align*}
\]
Example of Scalability Problem

Customer 2 buys widget 3 with store credit:

\[W = \text{Widgets.READ}(3)\]
\[C = \text{Customers.READ}(2)\]
IF (W.In_Stock < 1)
    ABORT
IF (C.Store_Credit < W.price)
    ABORT
W.In_Stock -= 1
C.Store_Credit -= W.price

Replication Exacerbates the Problem
Replication Exacerbates the Problem

No conflicting transactions can run!
Summary of Performance Issues

• Commit protocol like 2PC
  – Hurts performance
  – Helps with atomicity and isolation guarantees
• Synchronous replication
  – Hurts performance
  – Helps with consistency guarantees

But what if you want atomic, isolated MP Xacts and synchronous replication?

• Need some coordination
  – Costs latency
  – Costs throughput
  – State of the art increases these costs beyond what is strictly necessary
• Calvin
  – Uses determinism to move most of coordination outside of transaction boundaries
    • Drastically improves concurrency and throughput
    • Slightly improves latency
    • Simplifies and reduces monolithic nature of DBMS architecture
Replication Example:
Traditional Mechanism

Customer 2 buys widget 3 with store credit:
W = Widgets.READ(3)
C = Customers.READ(2)
IF (W.In_Stock < 1) THEN ABORT
IF (C.Store_Credit < W.price) THEN ABORT
W.In_Stock -= 1
C.Store_Credit -= W.price

Replication Example:
Traditional Mechanism

Customer 2 buys widget 3 with store credit:
W = Widgets.READ(3)
C = Customers.READ(2)
IF (W.In_Stock < 1) THEN ABORT
IF (C.Store_Credit < W.price) THEN ABORT
W.In_Stock -= 1
C.Store_Credit -= W.price

replicate

replicate
Replication Example: Traditional Mechanism

Customer 2 buys widget 3 with store credit:
W = Widgets.READ(3)
C = Customers.READ(2)
IF (W.In_Stock < 1) THEN ABORT
IF (C.Store_Credit < W.price) THEN ABORT
W.In_Stock -= 1
C.Store_Credit -= W.price

Replication Example: Deterministic Mechanism

Customer 2 buys widget 3 with store credit:
W = Widgets.READ(3)
C = Customers.READ(2)
IF (W.In_Stock < 1) THEN ABORT
IF (C.Store_Credit < W.price) THEN ABORT
W.In_Stock -= 1
C.Store_Credit -= W.price

No conflicting transactions can run!
Replication Example: Deterministic Mechanism

Customer 2 buys widget 3 with store credit:
W = Widgets.READ(3)  
C = Customers.READ(2)  
IF (W.In_Stock < 1) THEN ABORT  
IF (C.Store_Credit < W.price) THEN ABORT  
W.In_Stock -= 1  
C.Store_Credit -= W.price

Customer 2 buys widget 3 with store credit:
W = Widgets.READ(3)  
C = Customers.READ(2)  
IF (W.In_Stock < 1) THEN ABORT  
IF (C.Store_Credit < W.price) THEN ABORT  
W.In_Stock -= 1  
C.Store_Credit -= W.price

Why the previous mechanism doesn’t work without a deterministic DBMS

T1: Customer 2 buys widget 3 with store credit:  
W = Widgets.READ(3)  
C = Customers.READ(2)  
IF (W.In_Stock < 1) THEN ABORT  
IF (C.Store_Credit < W.price) THEN ABORT  
W.In_Stock -= 1  
C.Store_Credit -= W.price

T2: Customer 6 buys widget 3 with store credit:  
W = Widgets.READ(3)  
C = Customers.READ(6)  
IF (W.In_Stock < 1) THEN ABORT  
IF (C.Store_Credit < W.price) THEN ABORT  
W.In_Stock -= 1  
C.Store_Credit -= W.price
Why the previous mechanism doesn’t work without a deterministic DBMS

T1: Customer 2 buys widget 3 with store credit:
W = Widgets.READ(3)
C = Customers.READ(2)
IF (W.In_Stock < 1) THEN ABORT
IF (C.Store_Credit < W.price) THEN ABORT
W.In_Stock -= 1
C.Store_Credit -= W.price

T2: Customer 6 buys widget 3 with store credit:
W = Widgets.READ(3)
C = Customers.READ(6)
IF (W.In_Stock < 1) THEN ABORT
IF (C.Store_Credit < W.price) THEN ABORT
W.In_Stock -= 1
C.Store_Credit -= W.price

T3: Customer 4 buys widget 3 with store credit:
W = Widgets.READ(3)
C = Customers.READ(4)
IF (W.In_Stock < 1) THEN ABORT
IF (C.Store_Credit < W.price) THEN ABORT
W.In_Stock -= 1
C.Store_Credit -= W.price

T2: Customer 6 buys widget 3 with store credit:
W = Widgets.READ(3)
C = Customers.READ(6)
IF (W.In_Stock < 1) THEN ABORT
IF (C.Store_Credit < W.price) THEN ABORT
W.In_Stock -= 1
C.Store_Credit -= W.price

Architecture of a Deterministic DBMS