Reliability

Reliable Messaging

Transactions

Queues

Outline

- Reliable messaging
- Queued messaging
- ACID transactions in WCF
- Sagas and compensating transactions

Building reliable systems is hard

- **Reliable transmission isn't always guaranteed**
  - Connections can be flakey
  - Messages can get lost
  - Messages can arrive out of order
  - Messages can arrive more than once!
- **If you've got state, you've got other problems**
  - Distributed systems lead to weird failure modes
  - Must be able to recover after a crash
  - Concurrency adds even more complexity

WCF reliability solutions

Reliable sessions

- Connection maintenance and verification
- Exactly once delivery
- Optional ordered message delivery

Queues

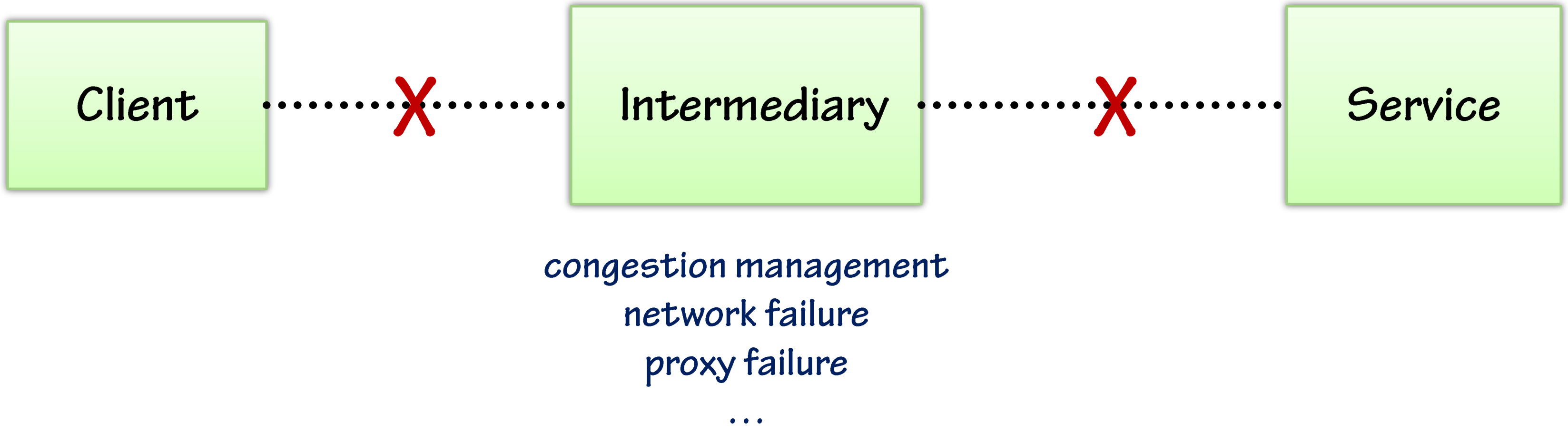
- Durable messaging
- Asynchronous messaging
- Loose-coupling
- Good for managing loads

Transactions

- ACID transactions surfaced via System.Transactions
- Transactions can flow between services

Why would I want reliable sessions?

- **Messages can get lost, reordered, even over TCP**
  - Consider proxies or other intermediaries
  - SOAP routers can drop messages



Reliable sessions in WCF

- **WCF makes it easy to use sessions**
  - Most bindings automatically support sessions
  - Use InstanceContextMode.PerSession to pin instances to sessions
  - These sessions aren't necessarily "reliable"
- **You make the session reliable by using reliable messaging**
  - Enable <reliableSession> on the binding
  - Uses WS-ReliableMessaging to ensure delivery
  - Not possible with inherently one-way transports
  - But it is possible with one-way operations

Reliable session features

- **Reliable sessions provide many features**
  - Dropped connections will be reestablished
  - Service notified if client goes away unexpectedly
  - Dropped messages will be retried
  - Duplicate messages will be ignored
- **Ordered messages are an optional guarantee**
  - A message sent out of order will be buffered and sent in order
  - Buffer has limited size, controlled via binding
- **There is a cost for all these features**
  - Measure and decide if it’s worth it

Enabling reliable sessions

- **To enable this feature, configure your binding appropriately**
  - netTcpBinding (off by default)
  - wsHttpBinding (off by default)
  - wsDualHttpBinding (always on)
  - For custom bindings, use <reliableSession/> element

Enable RM on the binding →

```
<bindings>
  <wsHttpBinding>
    <binding name="MyBinding">
      <reliableSession ordered="true"
         bufferedMessagesQuota="100"/>
    </binding>
  </wsHttpBinding>
</bindings>
```

Why would I want queued messaging?

- **Reliable messaging != durable messaging**
  - If either side goes down, so does the session and your state
- **Queues increase availability**
  - Service can be down, but client can still submit work
- **Queues are great for systems handling burst loads**
  - Smooths out the load over time
- **Often used with compensating transactional architectures**

Queued messaging

```
[ServiceContract]
interface IOrderManager {
    [OperationContract(IsOneWay = true)]
    void PlaceOrder(Order order);
}

class OrderManager : IOrderManager {
    void PlaceOrder(Order order) {
        // process order...
    }
}
```

Operations must be one-way

Address format for private queues

```
<endpoint
   address="net.msmq://MyServer/private/MyQueue/"
   binding="netMsmqBinding"
   bindingConfiguration ="MyQueueBinding"
   contract="IOrderManager" />
<bindings>
  <netMsmqBinding>
          <binding name="MyQueueBinding">
                   <security mode="None"/>
          </binding>
  </netMsmqBinding>
</bindings>
```

Why would I want transactions?

- **Concurrency is hard when you have state**
  - High degree of concurrency desirable
  - Multiple readers, writers
  - Everyone wants a consistent view of the data
- **Sometimes things don't go the way you planned**
  - Business logic dictates that the work you're doing is invalid
  - A resource you need isn't available
  - Your service crashes in the middle of a critical operation
- **ACID transactions can simplify your world**

ACID transactions

- **ACID transactions can simplify your world**
  - Atomic
  - Consistent
  - Isolated
  - Durable

Transactions

- Consider an operation that must perform its work atomically

```
[ServiceContract]
public interface ITransferFunds
{
    [OperationContract]
    bool Transfer(Account from, Account to, decimal amount);
}
```

System.Transactions

- In .NET 2.0, transactions became first-class citizens

Also on [ServiceBehavior]

Transaction scope delimits ACID tx

```
void Transfer(Account from, Account to, decimal amount)
{
    try {
        TransactionOptions txopt = new TransactionOptions();
        txopt.IsolationLevel = IsolationLevel.ReadCommitted;
        txopt.Timeout = new TimeSpan(0, 0, 10);
        TransactionScopeOption required =
           TransactionScopeOption.Required;
        using (TransactionScope txScope =
           new TransactionScope(required, txopt)) {
            //
            // use ADO.NET, MSMQ, other transacted RM...
            //
            txScope.Complete();
        }
    }
    catch (TransactionAbortedException x) {
        // notification if things go badly
    }
}
```

Declarative transactions

- WCF can obtain a transaction scope for you automatically
  - You give up some control
  - Simplifies code

```
[OperationBehavior(TransactionScopeRequired = true,
                   TransactionAutoComplete = true)]
void Transfer(Account from, Account to, decimal amount)
{
    //
    // do work here...
    //
}
```

You can also set TransactionAutoCompleteOnSessionClose

Flowing transactions between services

- WCF supports flowing an ACID transaction between services
  - Breaks the “services are autonomous” tenet
  - Services become tightly coupled
  - Requires a high degree of trust between the services
  - Uses WS-AtomicTransaction on the wire

```
[ServiceContract]
public interface ITransferFunds
{
  [OperationContract]
  [TransactionFlow(TransactionFlowOption.Mandatory)]
  bool Transfer(Account from, Account to, decimal amount);
}
```

To flow or not to flow

- **Transactions are meant to be short-lived**
  - Measured in nanoseconds, milliseconds at worst
  - Locks are being held while transaction is active
- **Flowed transactions can increase transaction time**
  - You may be holding locks while making round-trips to the service
  - Longer transactions can lead to less concurrency, less scalability

Saga

- **A saga is a sequence of transactions**
  - Breaks up the work into a sequence of smaller transactions
  - Some of these transactions move the work forward
  - Others move the work back (compensate) if something goes wrong
- **More work required on your part to build a saga**
  - When something goes wrong, need to know how to compensate
  - Not as simple as ITransaction.Rollback()
- **Queues often used to store intermediate state of work in a saga**

Combining queues and transactions

- **Two transactions are required for any queued message**
- **Client sends message(s)**
  - Sender starts a transaction
  - Sends message(s)
  - Commit/abort transaction $\longrightarrow$ *Message(s) in Q only after commit*
- **Service receives message(s)**
  - Receiver starts a transaction
  - Receives message(s) $\longrightarrow$ *Message(s) removed from Q*
  - Commit/abort transaction $\longrightarrow$ *Message(s) put back in Q on abort*

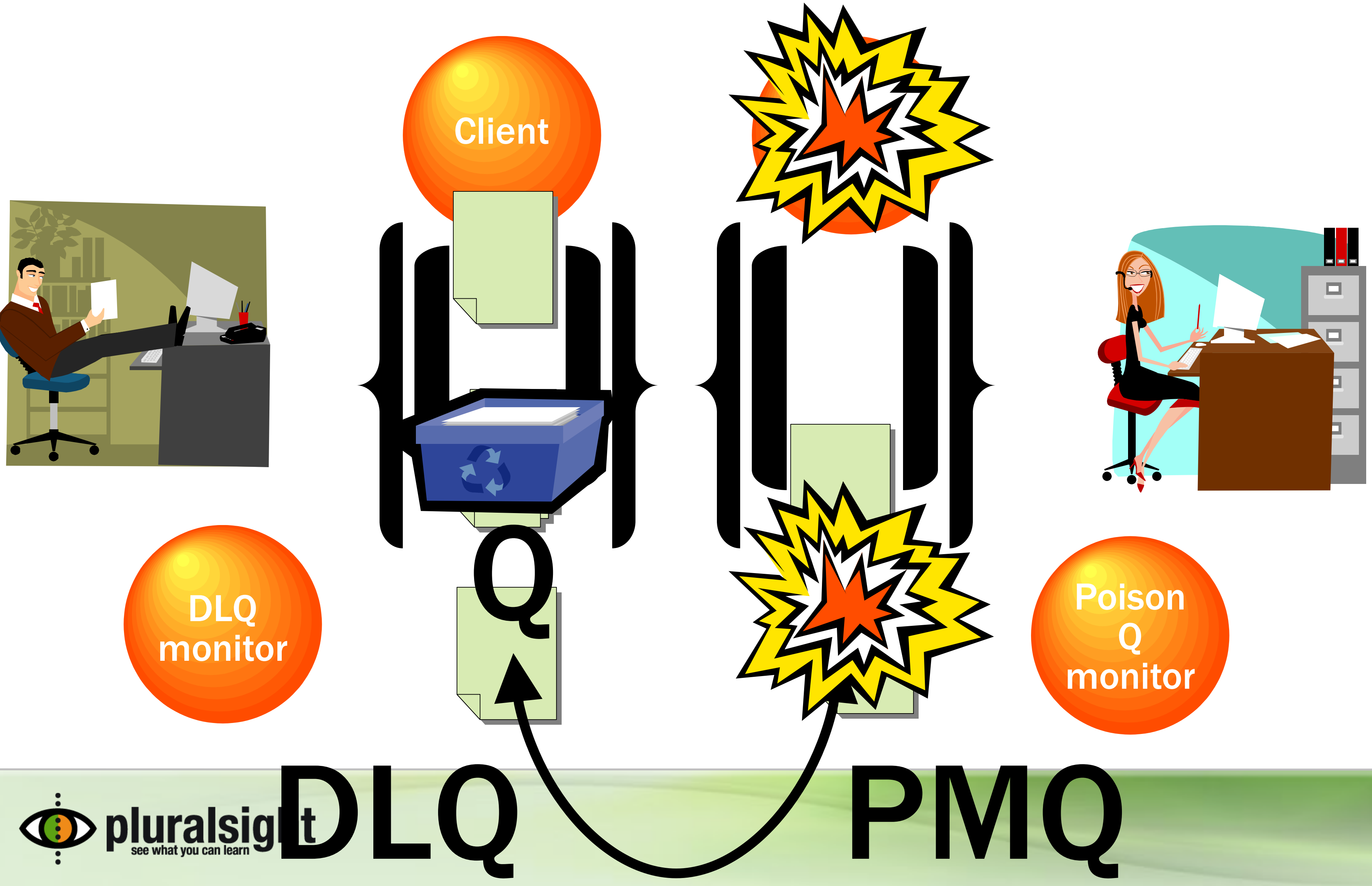
Compensating transactions

- **Compensation transactions are often needed in this approach**
  - A transaction that will "undo" actions that were committed
- **This is an optimistic approach**
  - Just relax and send messages, hoping for the best
  - Compensate later if there’s a problem
  - More complicated to build, but more loosely coupled

Things that can go wrong

- **Use a dead letter queue to track lost messages**
  - Service watching DLQ can automatically retry
  - Potentially alert sysadmin for assistance
- **Use a poison message queue to track bad messages**
  - Service watching PMQ can log bad messages
  - Potentially alert sysadmin for assistance

Dealing with failure



Summary

- Reliable messaging can provide guaranteed delivery
- Ordered messaging buffers messages to deliver in order
- ACID transactions can tremendously simplify building concurrent systems that must manage state
- Queues add complexity, but also many benefits
- Consider using transactions + queues to build a compensating architecture for loosely coupled services

References

- **Pluralsight's WCF Wiki**
  - http://pluralsight.com/wiki/default.aspx/Aaron/WindowsCommunicationFoundationWiki.html
- **Principals of Transaction Processing**
  - Bernstein & Newcomer, Morgan Kaufmann Publishers