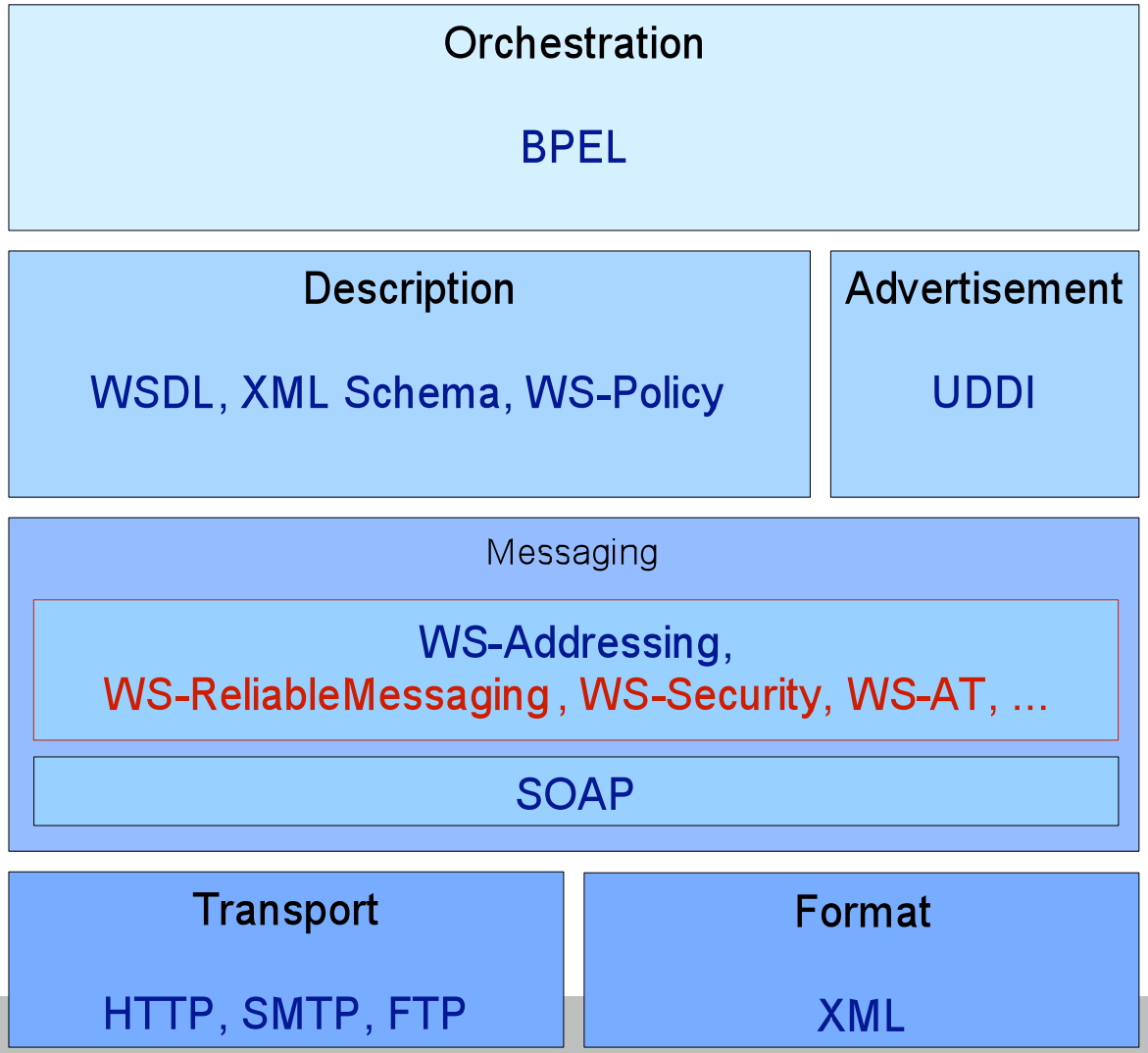
MTAT.03.229
Enterprise System Integration

Lecture 11: Integrity Aspects in Enterprise System Integration

Marlon Dumas

marlon . dumas ät ut . ee

Web Service Technology Stack



Integrity

- Goal: To ensure applications work correctly despite technical or business failures
- Technical failures
  - Network outage
  - Database crashes
  - Concurrency conflicts
  - Program logic violations
- Business exceptions
  - out of stock exception, resource unavailable
  - order cancellation
  - Deviations from the “happy path”

Types of Integrity

Communication reliability: Ensuring that messages are delivered despite network issues

Transactional integrity: Ensuring that an application does not reach an inconsistent state

- Data integrity
- Process integrity

Integrity Mechanisms (cont.)

Communication reliability: Provided by the middleware/app. containers

Data integrity: Enforced by the DBMS

Process integrity: Requires multiple strategies because processes are:

- Long-lived
- Span multiple systems
- Systems not synchronized and sometimes disconnected.

Reliable Message Delivery

Reliable message delivery policies:

- At least once
- Exactly once
- Ordered delivery

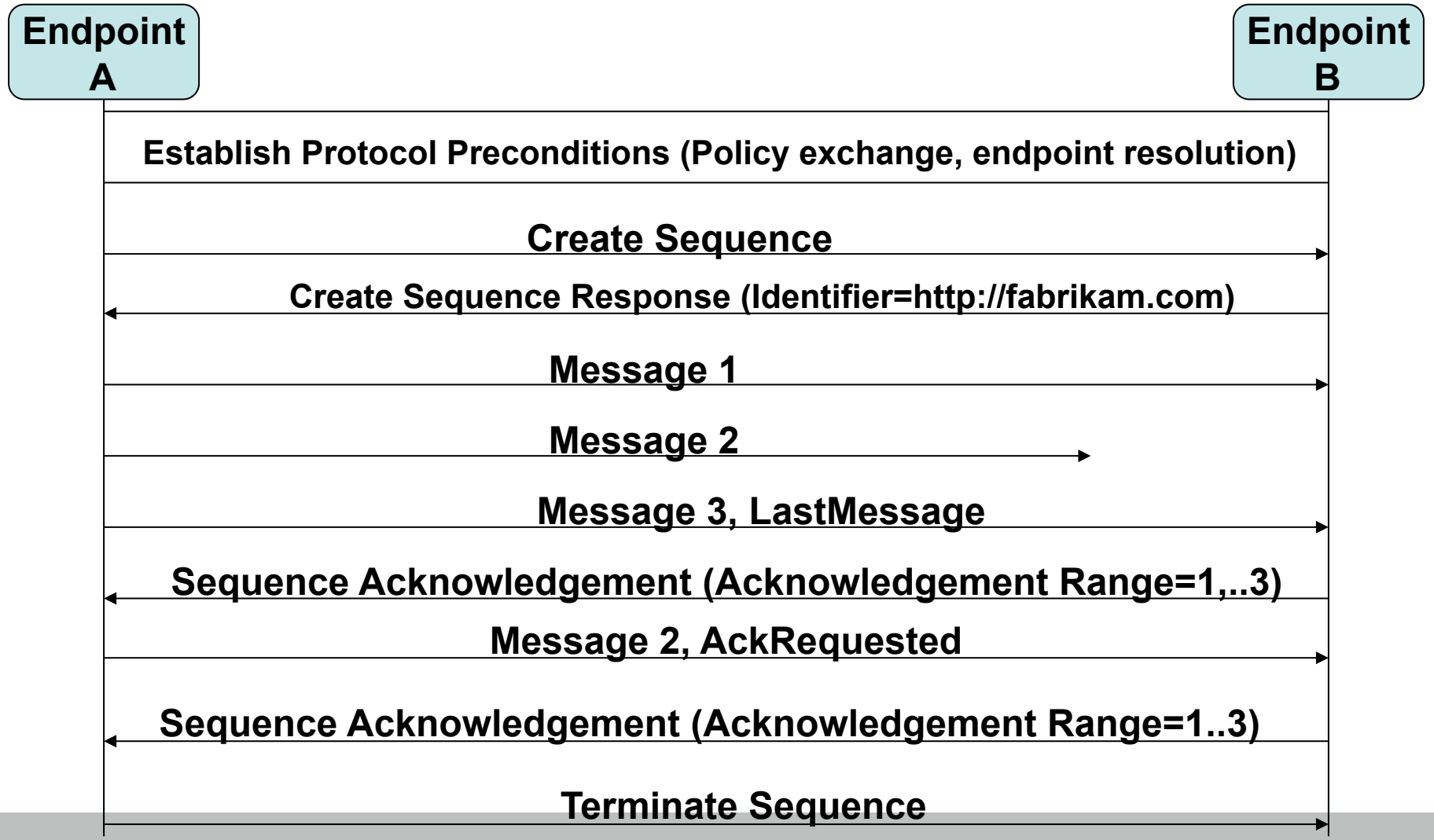
At runtime this is achieved through:

- Retries (also called *retransmissions*)
- Acknowledgements
- Message identifiers
- Sequence numbers

Retries are managed by several parameters:

- *Inactivity timeout*
- *Retransmission interval*: base retransmission interval (e.g. 1 sec), linear backoff (1, 2, 3,…), exponential backoff (1, 2, 4, 8, …)

WS-ReliableMessaging (WS-RM)



WS-RM: Creating a sequence

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2005/01/rm">
  xmlns:wsrm="http://schemas.xmlsoap.org/ws/2003/03/addressing">
<S:Header> ... </S:Header>
<S:Body>
    <wsrm:CreateSequence>
          <wsrm:AcksTo>  <wsa:Address>http://example.com/rm/endpoint</
    wsa:Address>              </wsrm:AcksTo>
          <wsrm:Expires>PT0S</wsrm:Expires> </wsrm:CreateSequence>
</S:Body>
</S:Envelope>
```

WS-RM: Create Sequence Response

```
<S:Envelope …>
<S:Header> ... </S:Header>
<S:Body>
    <wsrm:CreateSequenceResponse>
      <wsrm:Identifier>http://example.com/seq/564</wsrm:Identifier>
      <wsrm:Expires>PT0S</wsrm:Expires>
    </wsrm:CreateSequenceResponse>
</S:Body>
</S:Envelope>
```

WS-RM: Message in a sequence

```
<S:Envelope …>
<S:Header> ...
    <wsrm:Sequence>    <wsrm:Identifier>http://example.com/seq/564</
    wsrm:Identifier>
          <wsrm:MessageNumber>1</wsrm:MessageNumber>
    </wsrm:Sequence>
</S:Header>
<S:Body>
  Payload goes here
</S:Body>
```

WS-RM: Acknowledging messages

```
<S:Envelope …>
<S:Header> ...
    <wsrm:SequenceAcknowledgement>
      <wsrm:Identifier>http://example.com/seq/564</wsrm:Identifier>
      <wsrm:AcknowledgementRange Upper="3" Lower="1"/>
    </wsrm:SequenceAcknowledgement>
</S:Header>
<S:Body/>
```

Process Integrity

- Integrity across multiple steps or tasks
- Approaches to achieve process integrity:
  - Off-the-Shelf Online Transaction Processing (OLTP)
  - Custom-coded transactional steps
  - Queue-based transaction chains
  - Compensating actions

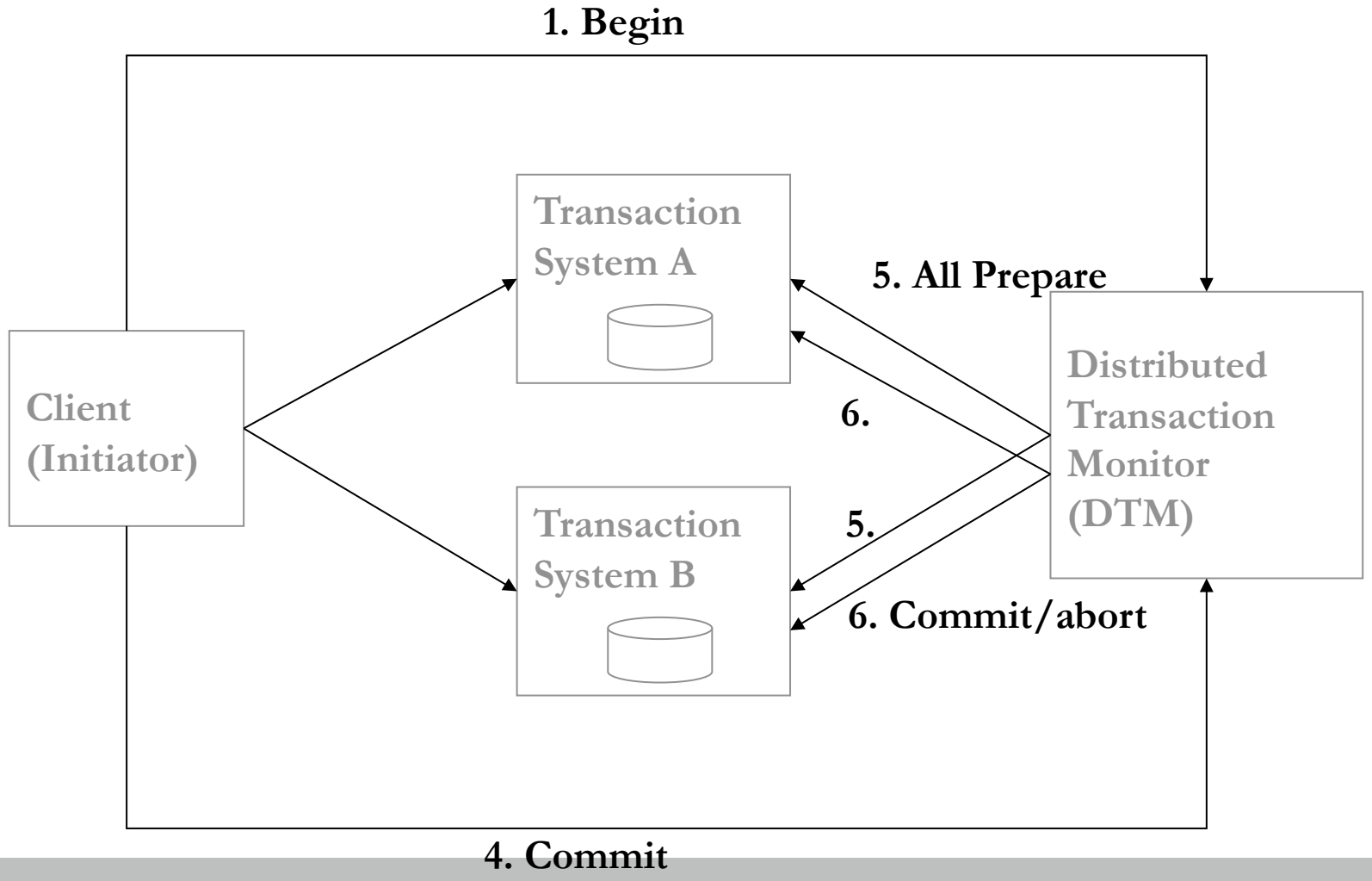
Online Transaction Processing (OLTP)

- Enables large numbers of users to manipulate shared data concurrently.
- E.g. bank account systems, flight reservation systems.
- Based on the concept of ACID Transaction

ACID Transactions

- **Atomicity**
  - if any one part of a transaction fails, the entire transaction is rolled back
- **Consistency**
  - Inconsistencies during transaction, but not at the end.
- **Isolation**
  - The internal state of a running transaction is never visible to any other running transaction.
  - Other transactions see either the before or after view.
- **Durability**
  - Committed updates of a transaction are permanent.

Distributed Atomic Transactions
The Two Phase Commit (2PC) Protocol



2PC Standards

- X/Open standard for Distributed Transaction Processing
  - includes XA interface which transaction monitors use to interact with participants.
- Most commercial DBMS and Queue managers support XA interface.
- Commercial DTMs include:
  - IBM CICS (Customer Information Control System)
  - Tuxedo (Oracle – formerly BEA)
  - Java Transaction Service (JTS)

Web Service Transaction Standards

- WS-AtomicTransaction (WS-AT): For short-lived atomic transactions (based on 2PC)
- WS-BusinessActivity (WS-BA): For long-running transactions transactions – low adoption
- Both define SOAP headers to be included in messages between services involved in a transaction
- For these standards to work, all services involved in a transaction must implement the protocols

Issues with ACID Transactions

Not always practical:

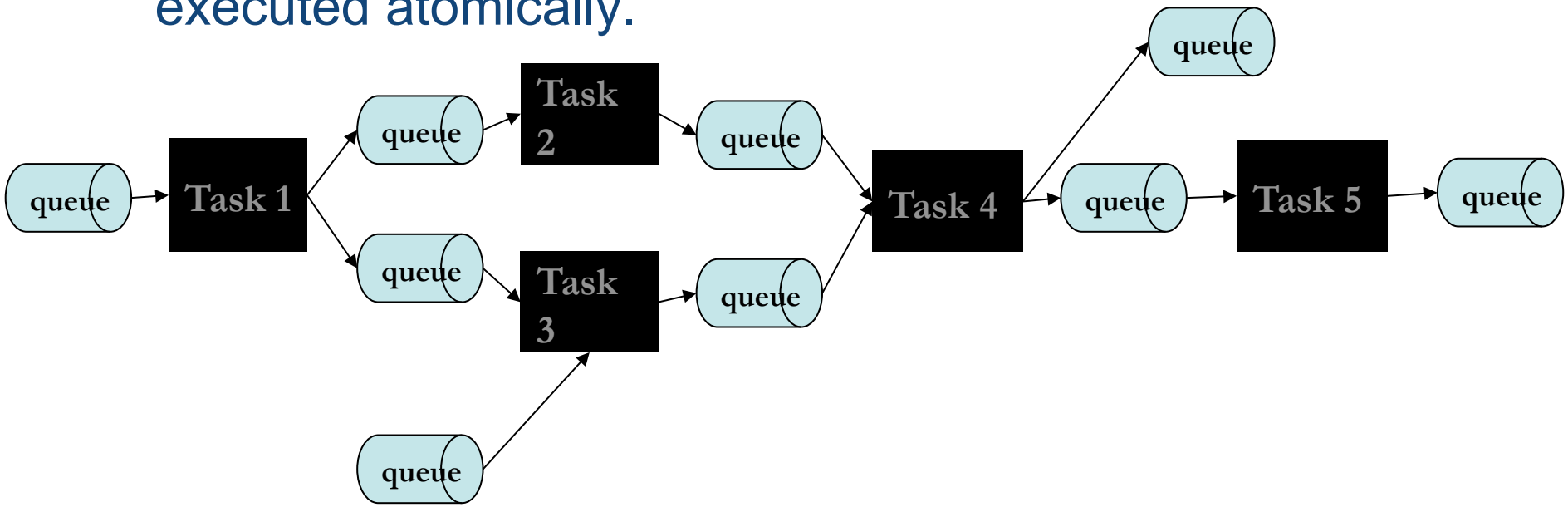
- Designed for short-lived transactions
  - Depending on the implementation – other clients may be slowed or even “halted” for long periods of time while a transaction is happening
- Performance
  - Even in single systems, ensuring complete isolation can be too expensive
  - In distributed systems coordination and network latency leads to major performance problems.

Issues with ACID Transactions (ctd)

- Integration with Legacy/Packaged Applications
  - XA interface generally not supported.
  - Even if an ERP such as SAP used XA-capable database, all access to SAP modules must go through SAP API which is non-transactional.
- Organizational Challenges
  - 2PC requires tight coupling and therefore trust and cooperation between parties involved.
  - 2PC requires that all partners use the same DTM. Unrealistic in a cross-organizational setting.

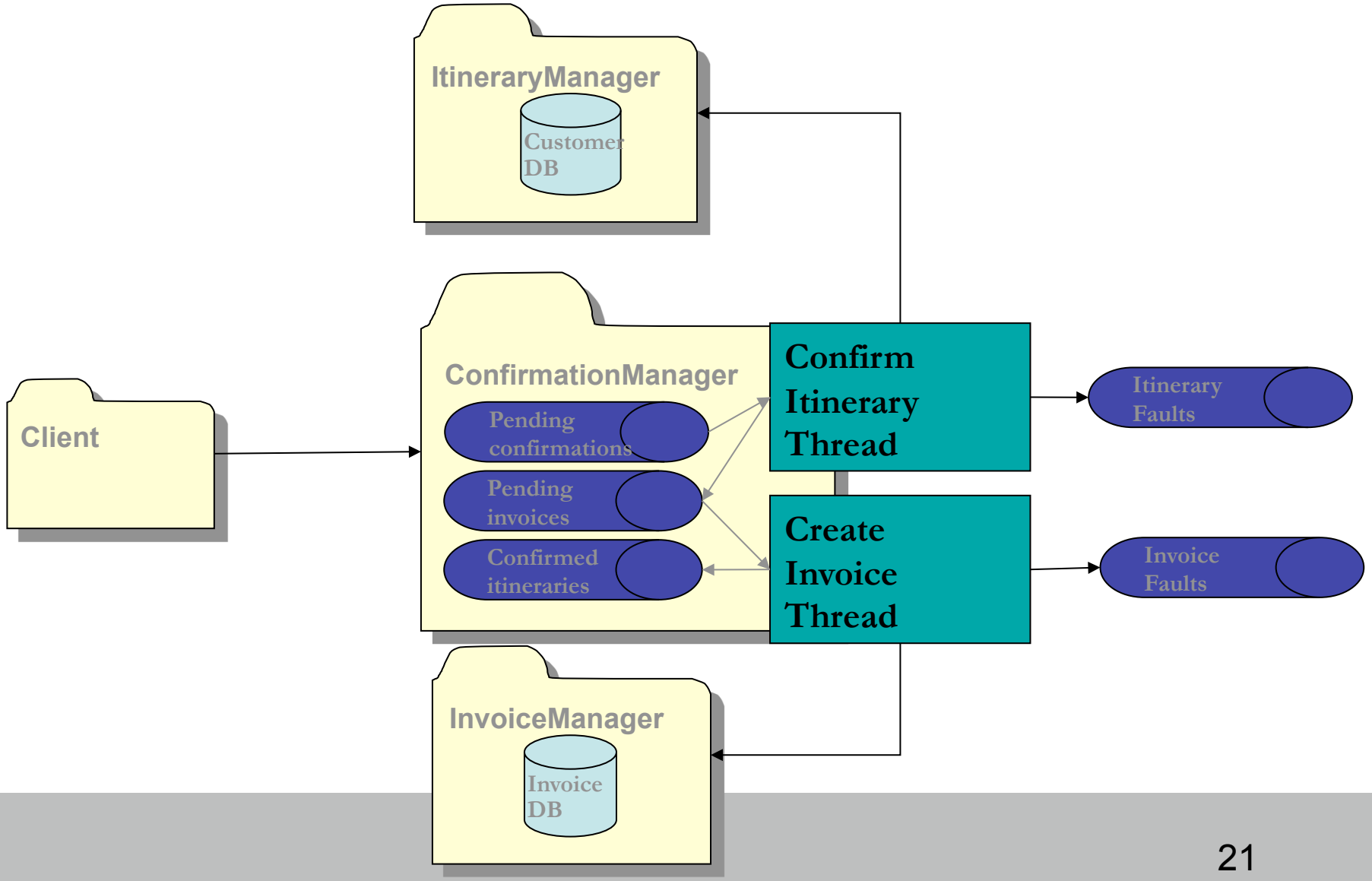
Queue-Based Transaction Chains

- Idea: Implement a process as a series of tasks that communicate via persistent queues and each task is executed atomically.



- The persistent queues participate in transactions, so if a task fails, its rollback will undo changes made to any input or output queues.

Queue-Based Transaction Chain



Advantages of Transaction Chains

- Can be applied to distributed processes where no central control exists, even if different transaction mechanisms are used in each step.
- Each transactional step is short-lived, even though the overall business process may be long-lived.
- The persistent queues also decouple the systems, allowing them to cope with periodic disconnection.

Limitations of Transaction Chains

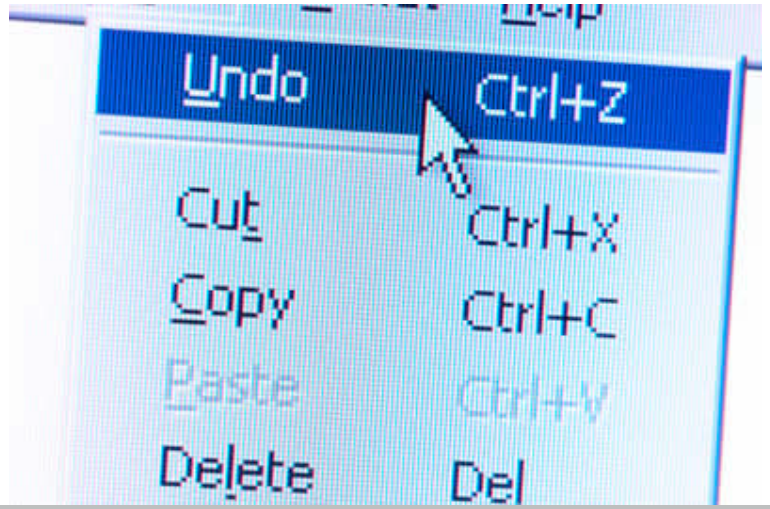
- While each step is performed as an isolated transaction, the overall business process is not.
- If a step later in the process fails in an unrecoverable manner, the effects of earlier steps are not automatically undone.
- May lead to a lot of queues to be managed, so suitable only if the “steps” are large

Compensating Actions

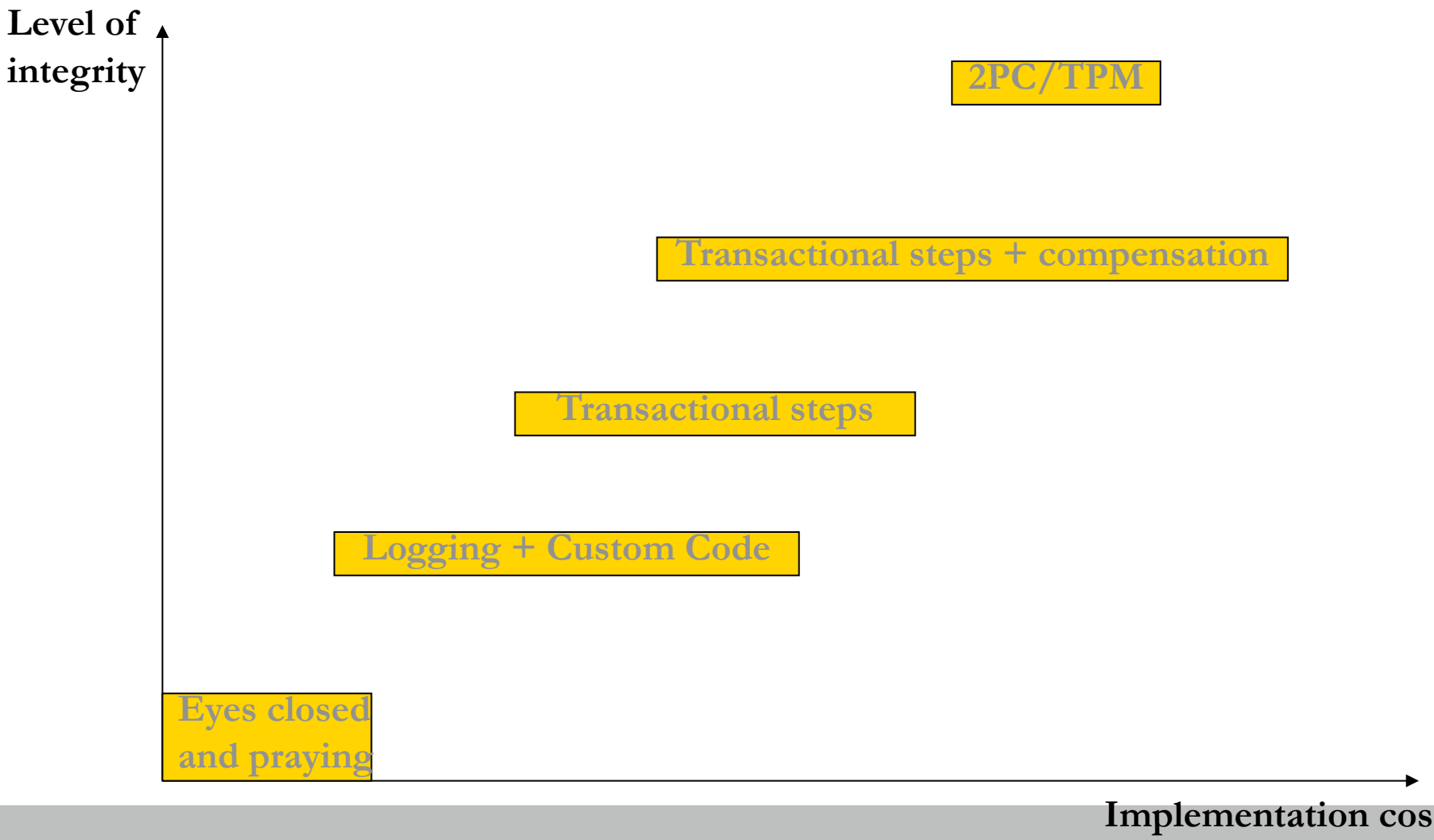
- The step is implemented by a single operation that performs the step atomically
  - The operation may succeed or fail
- If we detect a failure in a later step, a compensating action is executed to redress any undue effects of earlier steps:
  - E.g. issue a credit note to compensate for an erroneously issued invoice.
- Need to log sufficient data to facilitate the compensating action.

Compensating Actions: Drawbacks

- Some business actions cannot simply be “undone”
  - e.g. purchasing a non-refundable ticket
- What happens if the compensating action fails?



Integrity vs. Implementation Cost



References and Acknowledgment

- This lecture material is partly inspired by:
  - Enterprise SOA: Service-Oriented Architecture Best Practices by Dirk Krafzig et al., Prentice Hall, 2004