

# Web Services Wrap-up, Transactions

CS 475, Spring 2019  
Concurrent & Distributed Systems

# Review: Shared Fate

- Two methods/threads/processes running on the same computer generally have **shared fate**
- They will either both crash, or neither will crash



# Review: Split Brain

- When two machines in a distributed system can't talk to each other, they might start believing different things
- Two sides can not reconcile view of world because they can't talk to each other
- We call this a **split brain** problem

# Review: RPC Summary

- Procedure calls
  - Simple way to pass control and data
  - Elegant transparent way to distribute application
  - Not only way...
- Hard to provide true transparency
  - Failures
  - Performance
  - Memory access
  - Etc.
- How to deal with hard problem: give up and let programmer deal with it

# Today

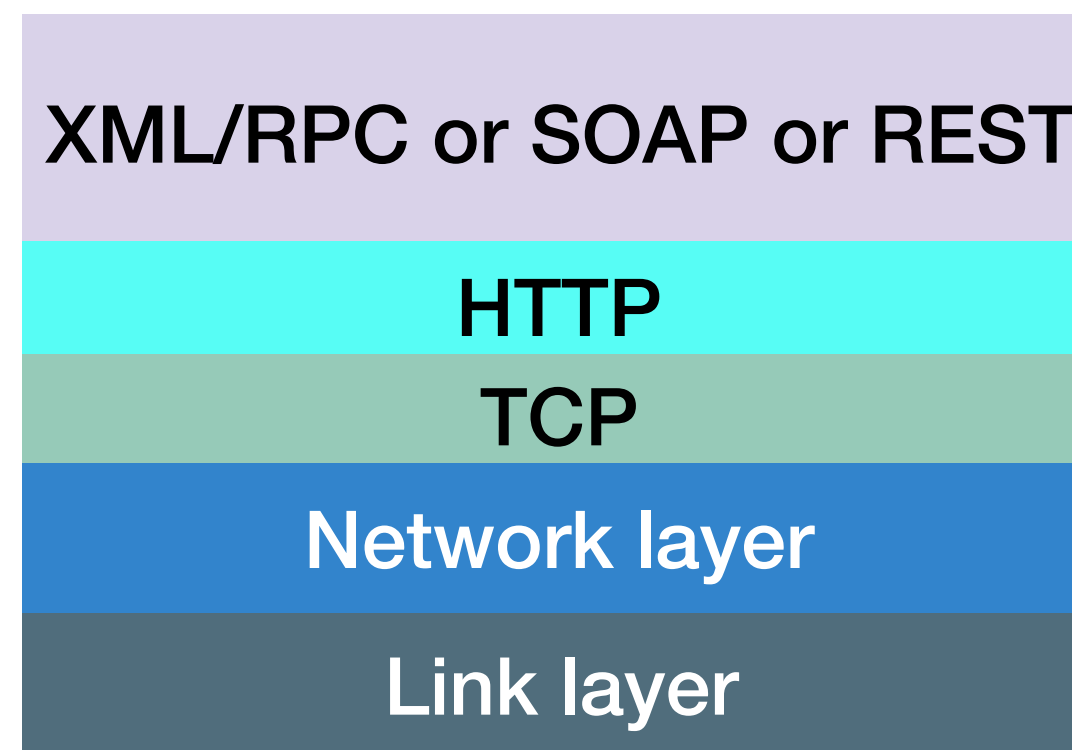
- RPC on the web
- Transactions - NOT yet getting to distributed transactions
- Note - YouTube lecture on Monday, Prof Bell at meeting off-campus
- Reminders:
  - HW3 posted

# RPC on the Web

- How do we do RPC on the web?
- Challenges for scaling up (more clients) and out (heterogeneous clients)
  - Need to get beyond RMI (it's Java only)
  - How do we find API endpoints?
  - How do we format requests?
  - How do we encode data?

# Web Services

- At a high level: any application that invokes computation via the Web
- Several standards:
  - XML/RPC
  - SOAP
  - REST
- All are implemented over HTTP as a communication protocol





# XML/RPC

- A specification for generic RPC, using XML as an interchange format

```
<?xml version="1.0"?> <methodCall>  
  <methodName>SumAndDifference</methodName> <params>  
    <param><value><i4>40</i4></value></param>  
    <param><value><i4>10</i4></value></param> </params>  
</methodCall>
```

- Recall - XML is a markup language — tags and parameters
- Protocols (like in this case, XML/RPC) define what tags mean (e.g. methodCall)



# XML/RPC

- Very simple specification
  - <http://xmlrpc.scripting.com/spec.html> (it's ~ 2 pages)
- Does not have a standard way to specify interfaces or generate stubs
  - Compare to: RMI @Remote interfaces
- No standard for extending protocol, adding authentication, sessions, etc

# SOAP

- Written in XML
- Extension to XML-RPC
- Defines mechanism to pass commands and parameters for RPC (like XML-RPC)
- Also defines standard for describing the services and interfaces (WSDL, or Web Service Definition Language)
- WSDL can be used to automatically generate stubs for client/server

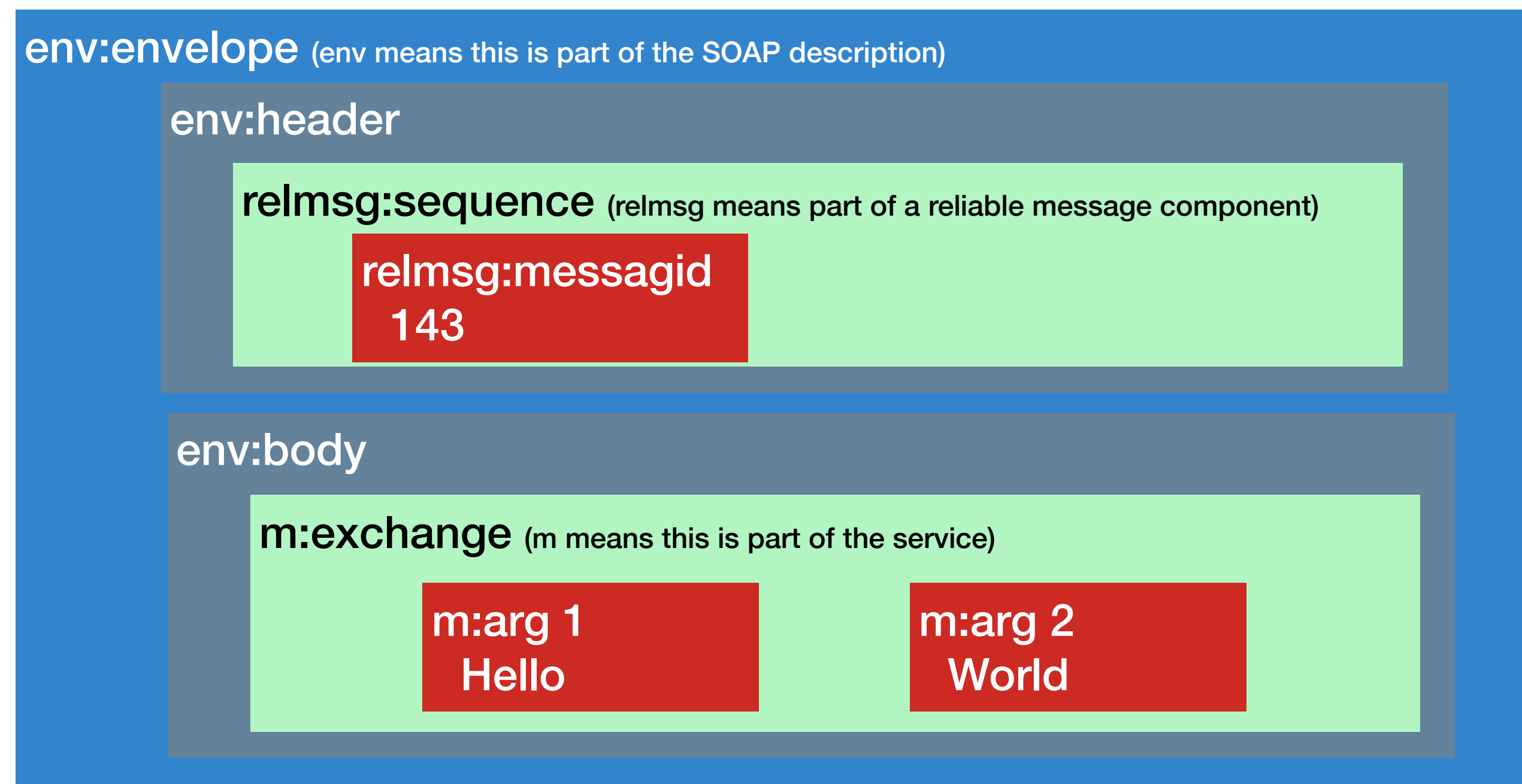
# WSDL

- Written in XML
- Defines a web services:
  - Operations offered by the service (what)
  - Mechanisms to access the service (how)
  - Location of the service (where)

```
<definitions name="MyService">  
  <types>data types used</types>  
  <message>parameters used</message>  
  <portType>set of operations performed</portType>  
  <binding>communication protocols and data formats used</binding>  
  <service>set of ports to service provider endpoints</service>  
</definitions>
```

# SOAP

- SOAP protocol defines how RPC are sent over a network
- WSDL defines how a given service uses SOAP
- SOAP packs messages into an envelope with a header and body
- Envelope abstraction allows SOAP extensions to do more stuff (authentication, etc)





# Web Services Standards Overview

## Interoperability Issues

**Basic Profile**  
WS-BaseProfile-1.0  
Final Specification

**Basic Profile**  
WS-BaseProfile-1.1  
Working Draft

**Basic Profile**  
WS-BaseProfile-1.2  
Working Draft

**Attachments Profile**  
WS-Attachments-1.0  
Final Specification

**Simple SOAP Binding Profile**  
WS-SimpleSOAPBinding-1.0  
Final Specification

**Basic Security Profile**  
WS-Security-1.0  
Final Specification

**WS-Security-1.1**  
Working Draft

**WS-Security-1.2**  
Working Draft

**WS-Security-1.3**  
Working Draft

**WS-Security-1.4**  
Working Draft

**WS-Security-1.5**  
Working Draft

**WS-Security-1.6**  
Working Draft

**WS-Security-1.7**  
Working Draft

**WS-Security-1.8**  
Working Draft

**WS-Security-1.9**  
Working Draft

**WS-Security-1.10**  
Working Draft

**WS-Security-1.11**  
Working Draft

**WS-Security-1.12**  
Working Draft

**WS-Security-1.13**  
Working Draft

## Business Process Specifications

**Business Process Language for Web Services 1.0**  
BPEL4WS-1.0  
Final Specification

**Business Process Language for Web Services 2.0**  
BPEL4WS-2.0  
Final Specification

**Business Process Language for Web Services 3.0**  
BPEL4WS-3.0  
Final Specification

**Business Process Language for Web Services 4.0**  
BPEL4WS-4.0  
Final Specification

**Business Process Language for Web Services 5.0**  
BPEL4WS-5.0  
Final Specification

**Business Process Language for Web Services 6.0**  
BPEL4WS-6.0  
Final Specification

**Business Process Language for Web Services 7.0**  
BPEL4WS-7.0  
Final Specification

**Business Process Language for Web Services 8.0**  
BPEL4WS-8.0  
Final Specification

**Business Process Language for Web Services 9.0**  
BPEL4WS-9.0  
Final Specification

**Business Process Language for Web Services 10.0**  
BPEL4WS-10.0  
Final Specification

**Business Process Language for Web Services 11.0**  
BPEL4WS-11.0  
Final Specification

**Business Process Language for Web Services 12.0**  
BPEL4WS-12.0  
Final Specification

**Business Process Language for Web Services 13.0**  
BPEL4WS-13.0  
Final Specification

**Business Process Language for Web Services 14.0**  
BPEL4WS-14.0  
Final Specification

**Business Process Language for Web Services 15.0**  
BPEL4WS-15.0  
Final Specification

**Business Process Language for Web Services 16.0**  
BPEL4WS-16.0  
Final Specification

**Business Process Language for Web Services 17.0**  
BPEL4WS-17.0  
Final Specification

**Business Process Language for Web Services 18.0**  
BPEL4WS-18.0  
Final Specification

**Business Process Language for Web Services 19.0**  
BPEL4WS-19.0  
Final Specification

**Business Process Language for Web Services 20.0**  
BPEL4WS-20.0  
Final Specification

## Metadata Specifications

**WSDL 1.0**  
WSDL-1.0  
Final Specification

**WSDL 2.0**  
WSDL-2.0  
Final Specification

**WSDL 3.0**  
WSDL-3.0  
Final Specification

**WSDL 4.0**  
WSDL-4.0  
Final Specification

**WSDL 5.0**  
WSDL-5.0  
Final Specification

**WSDL 6.0**  
WSDL-6.0  
Final Specification

**WSDL 7.0**  
WSDL-7.0  
Final Specification

**WSDL 8.0**  
WSDL-8.0  
Final Specification

**WSDL 9.0**  
WSDL-9.0  
Final Specification

**WSDL 10.0**  
WSDL-10.0  
Final Specification

**WSDL 11.0**  
WSDL-11.0  
Final Specification

**WSDL 12.0**  
WSDL-12.0  
Final Specification

**WSDL 13.0**  
WSDL-13.0  
Final Specification

**WSDL 14.0**  
WSDL-14.0  
Final Specification

**WSDL 15.0**  
WSDL-15.0  
Final Specification

**WSDL 16.0**  
WSDL-16.0  
Final Specification

**WSDL 17.0**  
WSDL-17.0  
Final Specification

**WSDL 18.0**  
WSDL-18.0  
Final Specification

**WSDL 19.0**  
WSDL-19.0  
Final Specification

**WSDL 20.0**  
WSDL-20.0  
Final Specification

## Management Specifications

**Management Using Web Services 1.0**  
MWS-1.0  
Final Specification

**Management Using Web Services 2.0**  
MWS-2.0  
Final Specification

**Management Using Web Services 3.0**  
MWS-3.0  
Final Specification

**Management Using Web Services 4.0**  
MWS-4.0  
Final Specification

**Management Using Web Services 5.0**  
MWS-5.0  
Final Specification

**Management Using Web Services 6.0**  
MWS-6.0  
Final Specification

**Management Using Web Services 7.0**  
MWS-7.0  
Final Specification

**Management Using Web Services 8.0**  
MWS-8.0  
Final Specification

**Management Using Web Services 9.0**  
MWS-9.0  
Final Specification

**Management Using Web Services 10.0**  
MWS-10.0  
Final Specification

**Management Using Web Services 11.0**  
MWS-11.0  
Final Specification

**Management Using Web Services 12.0**  
MWS-12.0  
Final Specification

**Management Using Web Services 13.0**  
MWS-13.0  
Final Specification

**Management Using Web Services 14.0**  
MWS-14.0  
Final Specification

**Management Using Web Services 15.0**  
MWS-15.0  
Final Specification

**Management Using Web Services 16.0**  
MWS-16.0  
Final Specification

**Management Using Web Services 17.0**  
MWS-17.0  
Final Specification

**Management Using Web Services 18.0**  
MWS-18.0  
Final Specification

**Management Using Web Services 19.0**  
MWS-19.0  
Final Specification

**Management Using Web Services 20.0**  
MWS-20.0  
Final Specification

## Presentation Specifications

**Web Services for Remote Portals 1.0**  
WSRP-1.0  
Final Specification

**Web Services for Remote Portals 2.0**  
WSRP-2.0  
Final Specification

**Web Services for Remote Portals 3.0**  
WSRP-3.0  
Final Specification

**Web Services for Remote Portals 4.0**  
WSRP-4.0  
Final Specification

**Web Services for Remote Portals 5.0**  
WSRP-5.0  
Final Specification

**Web Services for Remote Portals 6.0**  
WSRP-6.0  
Final Specification

**Web Services for Remote Portals 7.0**  
WSRP-7.0  
Final Specification

**Web Services for Remote Portals 8.0**  
WSRP-8.0  
Final Specification

**Web Services for Remote Portals 9.0**  
WSRP-9.0  
Final Specification

**Web Services for Remote Portals 10.0**  
WSRP-10.0  
Final Specification

**Web Services for Remote Portals 11.0**  
WSRP-11.0  
Final Specification

**Web Services for Remote Portals 12.0**  
WSRP-12.0  
Final Specification

**Web Services for Remote Portals 13.0**  
WSRP-13.0  
Final Specification

**Web Services for Remote Portals 14.0**  
WSRP-14.0  
Final Specification

**Web Services for Remote Portals 15.0**  
WSRP-15.0  
Final Specification

**Web Services for Remote Portals 16.0**  
WSRP-16.0  
Final Specification

**Web Services for Remote Portals 17.0**  
WSRP-17.0  
Final Specification

**Web Services for Remote Portals 18.0**  
WSRP-18.0  
Final Specification

**Web Services for Remote Portals 19.0**  
WSRP-19.0  
Final Specification

**Web Services for Remote Portals 20.0**  
WSRP-20.0  
Final Specification

## Transaction Specifications

**WS-Transaction 1.0**  
WS-Transaction-1.0  
Final Specification

**WS-Transaction 2.0**  
WS-Transaction-2.0  
Final Specification

**WS-Transaction 3.0**  
WS-Transaction-3.0  
Final Specification

**WS-Transaction 4.0**  
WS-Transaction-4.0  
Final Specification

**WS-Transaction 5.0**  
WS-Transaction-5.0  
Final Specification

**WS-Transaction 6.0**  
WS-Transaction-6.0  
Final Specification

**WS-Transaction 7.0**  
WS-Transaction-7.0  
Final Specification

**WS-Transaction 8.0**  
WS-Transaction-8.0  
Final Specification

**WS-Transaction 9.0**  
WS-Transaction-9.0  
Final Specification

**WS-Transaction 10.0**  
WS-Transaction-10.0  
Final Specification

**WS-Transaction 11.0**  
WS-Transaction-11.0  
Final Specification

**WS-Transaction 12.0**  
WS-Transaction-12.0  
Final Specification

**WS-Transaction 13.0**  
WS-Transaction-13.0  
Final Specification

**WS-Transaction 14.0**  
WS-Transaction-14.0  
Final Specification

**WS-Transaction 15.0**  
WS-Transaction-15.0  
Final Specification

**WS-Transaction 16.0**  
WS-Transaction-16.0  
Final Specification

**WS-Transaction 17.0**  
WS-Transaction-17.0  
Final Specification

**WS-Transaction 18.0**  
WS-Transaction-18.0  
Final Specification

**WS-Transaction 19.0**  
WS-Transaction-19.0  
Final Specification

**WS-Transaction 20.0**  
WS-Transaction-20.0  
Final Specification

## Resource Specifications

**WS-Resource 1.0**  
WS-Resource-1.0  
Final Specification

**WS-Resource 2.0**  
WS-Resource-2.0  
Final Specification

**WS-Resource 3.0**  
WS-Resource-3.0  
Final Specification

**WS-Resource 4.0**  
WS-Resource-4.0  
Final Specification

**WS-Resource 5.0**  
WS-Resource-5.0  
Final Specification

**WS-Resource 6.0**  
WS-Resource-6.0  
Final Specification

**WS-Resource 7.0**  
WS-Resource-7.0  
Final Specification

**WS-Resource 8.0**  
WS-Resource-8.0  
Final Specification

**WS-Resource 9.0**  
WS-Resource-9.0  
Final Specification

**WS-Resource 10.0**  
WS-Resource-10.0  
Final Specification

**WS-Resource 11.0**  
WS-Resource-11.0  
Final Specification

**WS-Resource 12.0**  
WS-Resource-12.0  
Final Specification

**WS-Resource 13.0**  
WS-Resource-13.0  
Final Specification

**WS-Resource 14.0**  
WS-Resource-14.0  
Final Specification

**WS-Resource 15.0**  
WS-Resource-15.0  
Final Specification

**WS-Resource 16.0**  
WS-Resource-16.0  
Final Specification

**WS-Resource 17.0**  
WS-Resource-17.0  
Final Specification

**WS-Resource 18.0**  
WS-Resource-18.0  
Final Specification

**WS-Resource 19.0**  
WS-Resource-19.0  
Final Specification

**WS-Resource 20.0**  
WS-Resource-20.0  
Final Specification

## Dependencies



innoQ Embedded GmbH  
Hohenzollern 12  
D-40080 Ratingen  
Phone: +49 21 02 77 62-100  
info@innoq.com - www.innoq.com

innoQ Software GmbH  
Gartenstraße 11  
D-40080 Ratingen  
Phone: +49 21 41 343 81 11

# SOAP

- SOAP has LOTS of extensions (60+)
  - Reliable messaging
  - Security
  - Addressing
  - Transactions
- SOAP supports a lot of complexity **in the protocol itself**
- Problem: just to get a minimal, small example working, you need to do a lot of boilerplate

# REST: REpresentational State Transfer

- Defined by Roy Fielding in his 2000 Ph.D. dissertation
- “Throughout the HTTP standardization process, I was called on to defend the design choices of the Web. That is an extremely difficult thing to do... I had comments from well over 500 developers, many of whom were distinguished engineers with decades of experience. That process honed my model down to a core set of principles, properties, and constraints that are now called REST.”
- Interfaces that follow REST principles are called RESTful



# Principles of REST

- Client server: separation of concerns (reuse)
- Stateless: each client request contains all information necessary to service request (scaling)
- Cacheable: clients and intermediaries may cache responses. (scaling)
- Layered system: client cannot determine if it is connected to end server or intermediary along the way. (scaling)
- Uniform interface for resources: a single uniform interface (URIs) simplifies and decouples architecture (change & reuse)

# REST - URI Design

- URIs represent a contract about what resources your server exposes and what can be done with them
- Leave out **anything that might change**
  - Content author names, status of content, other keys that might change
  - File name extensions: response describes content type through MIME header not extension (e.g., .jpg, .mp3, .pdf)
  - Server technology: should not reference technology (e.g., .cfm, .jsp)
- Endeavor to make all changes backwards compatible
  - Add new resources and actions rather than remove old
- If you must change URI structure, support old URI structure **and** new URI structure

# Example URI Design

- The candy web service!
- Tracks information about candy
- <http://api.jonbell.net/candy/twix>
  - GET this URI to find out about twix bar
  - POST to the URI to set up a new twix bar
  - DELETE this URI to eat a twix

# Transactions

# Transactions

```
boolean transferMoney(Person from, Person to, float
amount){
    if(from.balance >= amount)
    {
        from.balance = from.balance - amount;
        to.balance = to.balance + amount;
        return true;
    }
    return false;
}
```

What can go wrong here?

# Transactions: Classic Example

```
boolean transferMoney(Person from, Person to, float amount){  
    if(from.balance >= amount)  
    {  
        from.balance = from.balance - amount;  
        to.balance = to.balance + amount;  
        return true;  
    }  
    return false;  
}
```

**transferMoney(P1, P2, 100)**

P1.balance (200) >= 100

P1.balance = 200 - 100 = 0

P2.balance = 200 + 100 = 300

return true;

**transferMoney(P1, P2, 200)**

P2.balance (200) > 200

P1.balance = 100 - 200 = -100

P2.balance = 300 + 200 = 500

return true;

What's wrong here?

Need isolation (prevent overdrawing)

# Transactions: Classic Example

```
boolean transferMoney(Person from, Person to, float amount){  
    synchronized(from){  
        if(from.balance >= amount)  
        {  
            from.balance = from.balance - amount;  
            to.balance = to.balance + amount;  
            return true;  
        }  
        return false;  
    }  
}
```

**transferMoney(P1, P2, 100)**

P1.balance (200) >= 100  
P1.balance = 200 - 100 = 0  
P2.balance = 200 + 100 = 300  
return true;

**transferMoney(P1, P2, 200)**

P1.balance <= 200  
return false;

Adding a lock: prevents accounts from being overdrawn

**But: shouldn't we lock on to also?**



# Transactions: Classic Example

```
boolean transferMoney(Person from, Person to, float amount){
    synchronized(from, to){
        if(from.balance >= amount)
        {
            from.balance = from.balance - amount;
            to.balance = to.balance + amount;
            return true;
        }
        return false;
    }
}
```

**transferMoney(P1, P2, 100)**

P1.balance (200) >= 100

P1.balance = 200 - 100 = 0

P2.balance = 200 + 100 = 300

return true;

**transferMoney(P1, P2, 200)**

P1.balance <= 200

return false;

Locking on both from, to at same time

# Transactions: Classic Example

```
boolean transferMoney(Person from, Person to, float amount){  
    synchronized(from, to){  
        if(from.balance >= amount)  
        {  
            from.balance = from.balance - amount;  
            to.balance = to.balance + amount;  
            return true;  
        }  
        return false;  
    }  
}
```

**transferMoney(P1, P2, 100)**

P1.balance (200) >= 100

P1.balance = 200 - 100 = 0



**transferMoney(P1, P2, 200)**

P1.balance <= 200

return false;

Problem: P1.balance was deducted P2.balance not incremented! (“Atomicity violation”)

# Transactions

- How can we provide some consistency guarantees **across operations**
- Transaction: unit of work (grouping) of operations
  - Begin transaction
  - Do stuff
  - Commit OR abort
- Why distributed transactions?
  - Data might be huge, spread across multiple machines
  - Scale performance up
  - Replicate data to tolerate failures

# Properties of Transactions

- Traditional properties: ACID
- **Atomicity**: transactions are “all or nothing”
- **Consistency**: Guarantee some basic properties of data; each transaction leaves the database in a valid state
- **Isolation**: Each transaction runs as if it is the only one; there is some valid serial ordering that represents what happens when transactions run concurrently
- **Durability**: Once committed, updates cannot be lost despite failures


# Concurrency control: Consistency & Isolation

# 2-phase locking

- Simple solution for isolation
- Phase 1: acquire locks (all that you might need)
- Phase 2: release locks
  - You can't get any more locks after you release any
  - Typically: locks released when you say “commit” or “abort”

# NOT 2-phase locking

```
boolean transferMoney(Person from, Person to, float amount){  
    from.lock();  
    if(from.balance >= amount)  
    {  
        from.balance = from.balance - amount;  
        from.unlock();  
        to.lock();  
        to.balance = to.balance + amount;  
        to.unlock();  
        return true;  
    }  
    else  
        from.unlock();  
    return false;  
}
```




**Invalid: other  
transactions could read  
an inconsistent system  
state at this point!**



# 2-phase locking

```
boolean transferMoney(Person from, Person to, float amount){  
    from.lock();  
    if(from.balance >= amount)  
    {  
        from.balance = from.balance - amount;  
        to.lock();  
        to.balance = to.balance + amount;  
        to.unlock();  
        from.unlock();  
        return true;  
    }  
    else  
        from.unlock();  
    return false;  
}
```



**Might deadlock if one  
transaction gives from  
P1->P2, other P2->P1**

# Serializability

- Ideal isolation semantics
- Slightly stronger than sequential consistency
- Definition: execution of a set of transactions is equivalent to *some* serial order
  - Two executions are equivalent if they have the same effect on program state and produce the same output
  - Just like sequential consistency, but the outcome must be equivalent to an ordering where *nothing* happens concurrently, no re-ordering of events between multiple transactions.

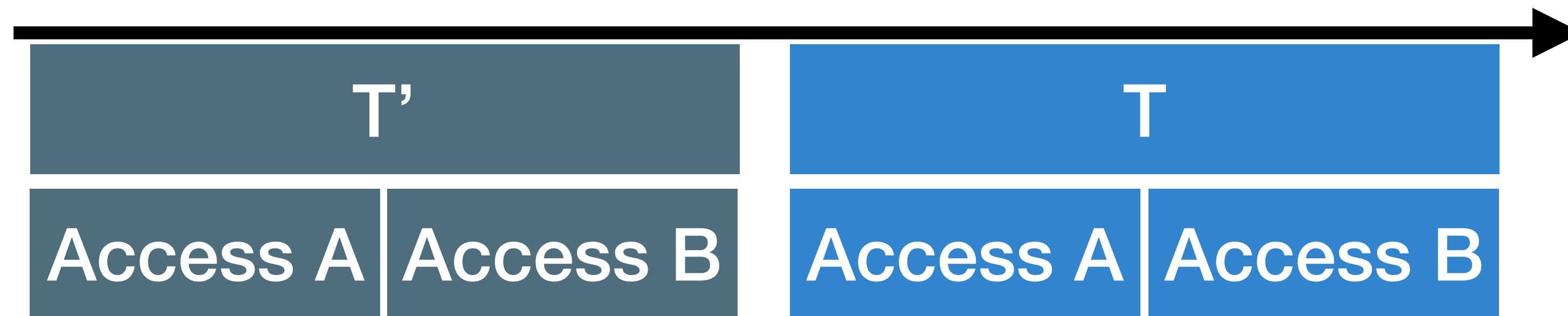
# 2-Phase Locking Ensures Serializability of Transactions

- Allows serializability to be considered at the level of transactions, which might include multiple variables
- If a transaction T accesses variables A and B, and T' accesses variables A and B, then either:



# 2-Phase Locking Ensures Serializability of Transactions

- Allows serializability to be considered at the level of transactions, which might include multiple variables
- If a transaction T accesses variables A and B, and T' accesses variables A and B, then either:



# 2-Phase Locking Ensures Serializability of Transactions

**Individual variable accesses are sequentially consistent, but transactions are not serializable!**

- If a transaction T accesses variables A and B, and T' accesses variables A and B, then either:



# Proof of Serializability - 2PL

- Proof by contradiction
- Is it possible for  $T \rightarrow T'$  and  $T' \rightarrow \dots \rightarrow T$ ? (different order for A and B)
- What would have happened?
  - 1. T releases lock of A
  - 2. T' acquires lock of A
  - 3. T' releases lock of B
  - 4. T acquires lock of B
- Hence,  $1 \rightarrow 2, 3 \rightarrow 4$
- But, required by 2PL:  $4 \rightarrow 1, 2 \rightarrow 3$  (or vv)
- Putting this together would be:  $4 \rightarrow 1 \rightarrow 2, 2 \rightarrow 3 \rightarrow 4$  aka a contradiction

# Concurrency Weirdness



Employee	Salary
Bob	100
Herbert	100
Larry	100
Jon	100

Transaction 1: Update employees, set salary = salary\*1.1

Transaction 2: Hire Carol, Hire Mike



# Concurrency Weirdness



Employee	Salary
Bob	100
Herbert	100
Larry	100
Jon	100

Transaction 1: Update employees, set salary = salary\*1.1

Transaction 2: Hire Carol, Hire Mike

**Can run concurrently: no overlapping locks!**

# Concurrency Weirdness


	Employee	Salary
🔒	Bob	100
🔒	Herbert	100
🔒	Larry	100
🔒	Jon	100
	Carol	100

Transaction 1: Update employees, set salary = salary\*1.1

Transaction 2: Hire Carol, Hire Mike

**Can run concurrently: no overlapping locks!**

# Concurrency Weirdness




Employee	Salary
Bob	110
Herbert	110
Larry	110
Jon	110
Carol	110

Transaction 1: Update employees, set salary = salary\*1.1

Transaction 2: Hire Carol, Hire Mike

**Can run concurrently: no overlapping locks!**

# Concurrency Weirdness



Employee	Salary
Bob	110
Herbert	110
Larry	110
Jon	110
Carol	110
Mike	100

Transaction 1: Update employees, set salary = salary\*1.1

Transaction 2: Hire Carol

**Solution to prevent this: Transaction 1 must always acquire some lock to prevent *any* other transaction from touching the data!**

**Or: ignore this problem and accept the consequences**

# No half measures: How do we ensure the entire transaction happens, or none? (Atomicity, Durability)

# If the machine crashes?

# Fault Recovery

- How do we recover transaction state if we crash?
- Goal:
  - Committed transactions are not lost
  - Non-committed transactions either continue where they were or aborted
- Plan:
  - Consider local recovery
  - Then distributed issues

# Write-ahead logging

- Maintain a complete log of all operations INDEPENDENT of the actual data they apply to
  - E.g. Transaction boundaries and updates
- Transaction operations considered provisional until commit is logged to disk
- Log is authoritative



# Write ahead logging: Begin/commit/abort

- Maintain this big log, with...
- Log Sequence Numbers (LSN) to track entries
- Each record contains an LSN, plus the LSN of the previous transaction
- Transaction ID
- Operation type

# Write ahead logging: update records

- Track all information needed to reproduce transaction
  - prevLSN, transactionID, operationType (like begin/commit/abort)
- Update itself:
  - Update location
  - Old value
  - New value

# Recovering From Failure

- Let's assume we can always read the log
- Analyze the log
- Redo all transactions starting from beginning
- Undo uncommitted transactions
  - We replay all of the transactions for consistency
  - Generalize all operations - don't need to store the results of operations, just the operations

# Write Ahead Logging + Checkpoints

- If you have a checkpoint, you can guarantee that all things before that checkpoint have been flushed to disk
- Hence, no need to replay log after then
- Speeds up recovery
- Reduces log size
- Can always build one checkpoint off an old one
- Why not always checkpoint?

# Distributing Transactions

- System model: data stored in multiple locations, multiple servers participating in a single transaction. One server pre-designated “coordinator”
- Failure model: messages can be delayed or lost, servers might crash, but have persistent storage to recover from

# Distributed Transactions

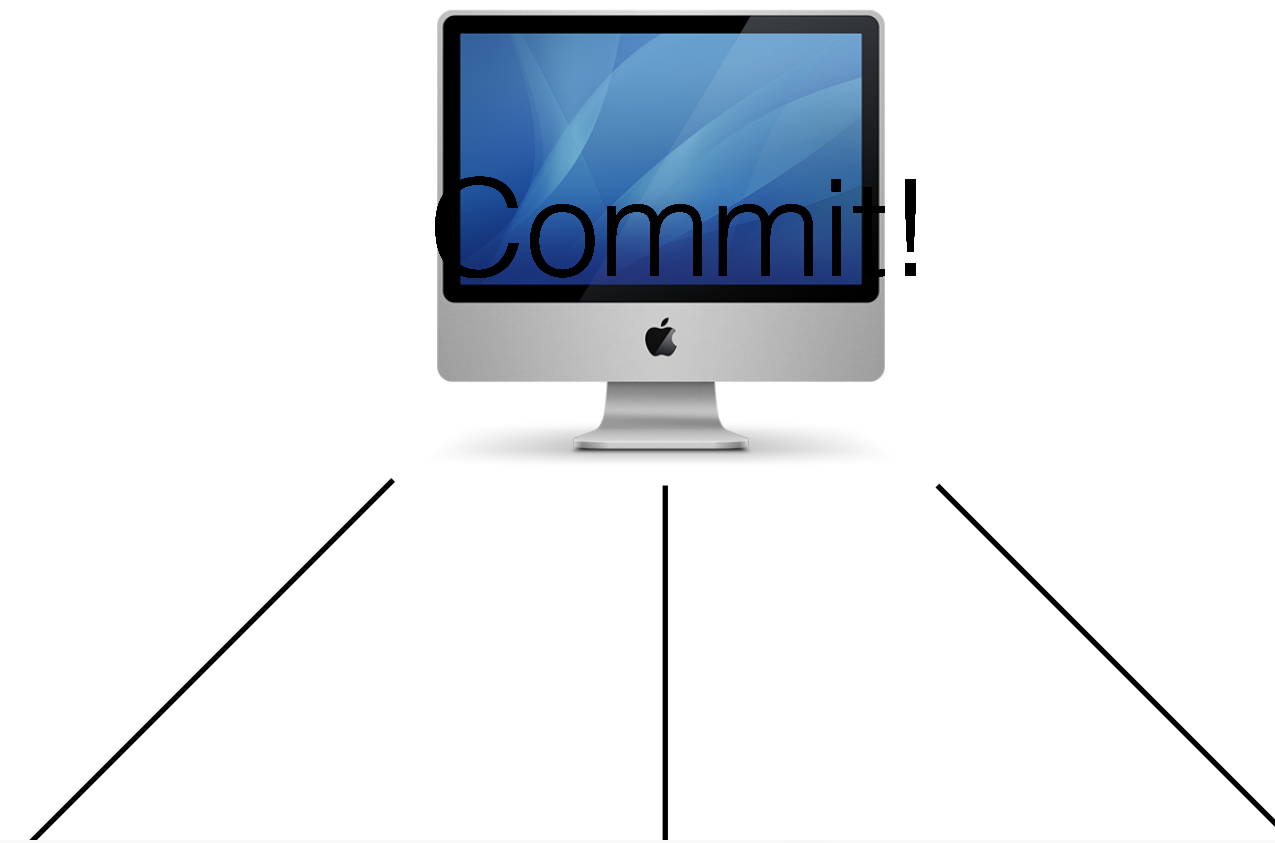
- Coordinator: Begins a transaction
  - Assigns a unique transaction ID
  - Responsible for commit + abort
  - In principle, any client can be the coordinator, but all participants need to agree on who is the coordinator
- Participants: everyone else who has the data used in the transaction

# Naive Distributed Transactions

- Naive protocol: coordinator broadcasts out “commit!” continuously until participants all say “OK!”
- Problem: what happens when a participant fails during commit? How do the other participants know that they shouldn't have really committed and they need to abort?



# Naive Distributed Transactions



We couldn't successfully commit on all 3 machines. But 1-phase commit has no way to go back!



# This work is licensed under a Creative Commons Attribution-ShareAlike license

- This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/>
- You are free to:
  - Share — copy and redistribute the material in any medium or format
  - Adapt — remix, transform, and build upon the material
  - for any purpose, even commercially.
- Under the following terms:
  - Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.