Two-Phase Commit is Evil

Rob Daigneau

Rob.Daigneau@ArcSage.com

DevTeach

Overview

- Review of Transaction Concepts
- Distributed Transactions
- 2PC Considerations
- The CAP Theorem
- Alternatives to 2PC

About Me

- Principal of 

- Author
  "Design Patterns for the Service Layer
  Fundamental Design Solutions for SOA, REST, EDA, and the Cloud"
  - *Estimated release date == 2009, Addison Wesley*
- Host of **www.DesignPatternsFor.Net**
- Chief Architect 

- Director of Architecture 

DevTeach

Review of Transaction Concepts

DevTeach

Why Do We Need Transactions Anyway?

- It's all about Data Integrity

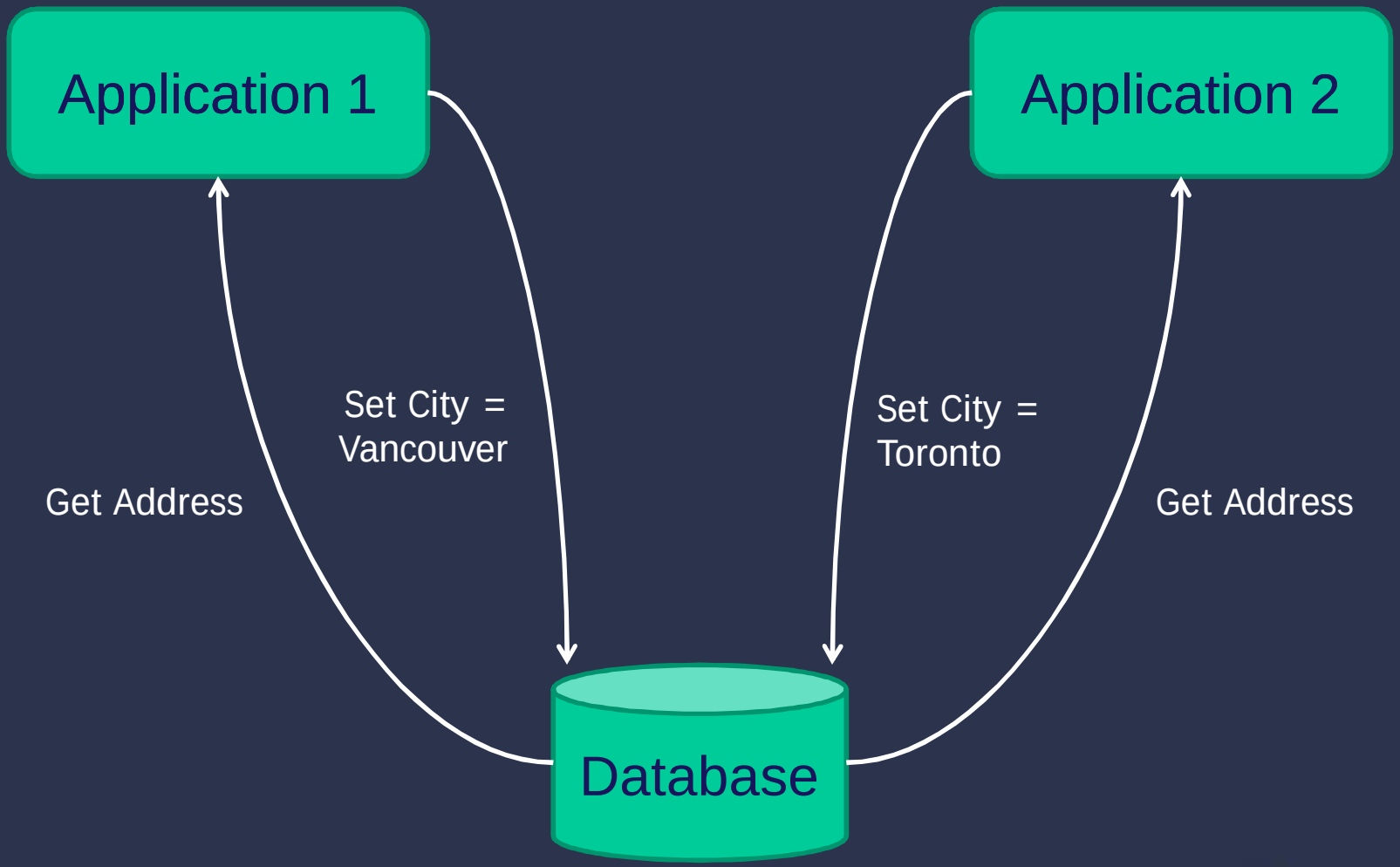
  … while attempting to provide Concurrent Data Access

  for Read or Write operations

- Classic concurrency problems …
  - The Lost Update problem
  - Unrepeatable Read

The Lost Update Problem

Toronto? What happened?

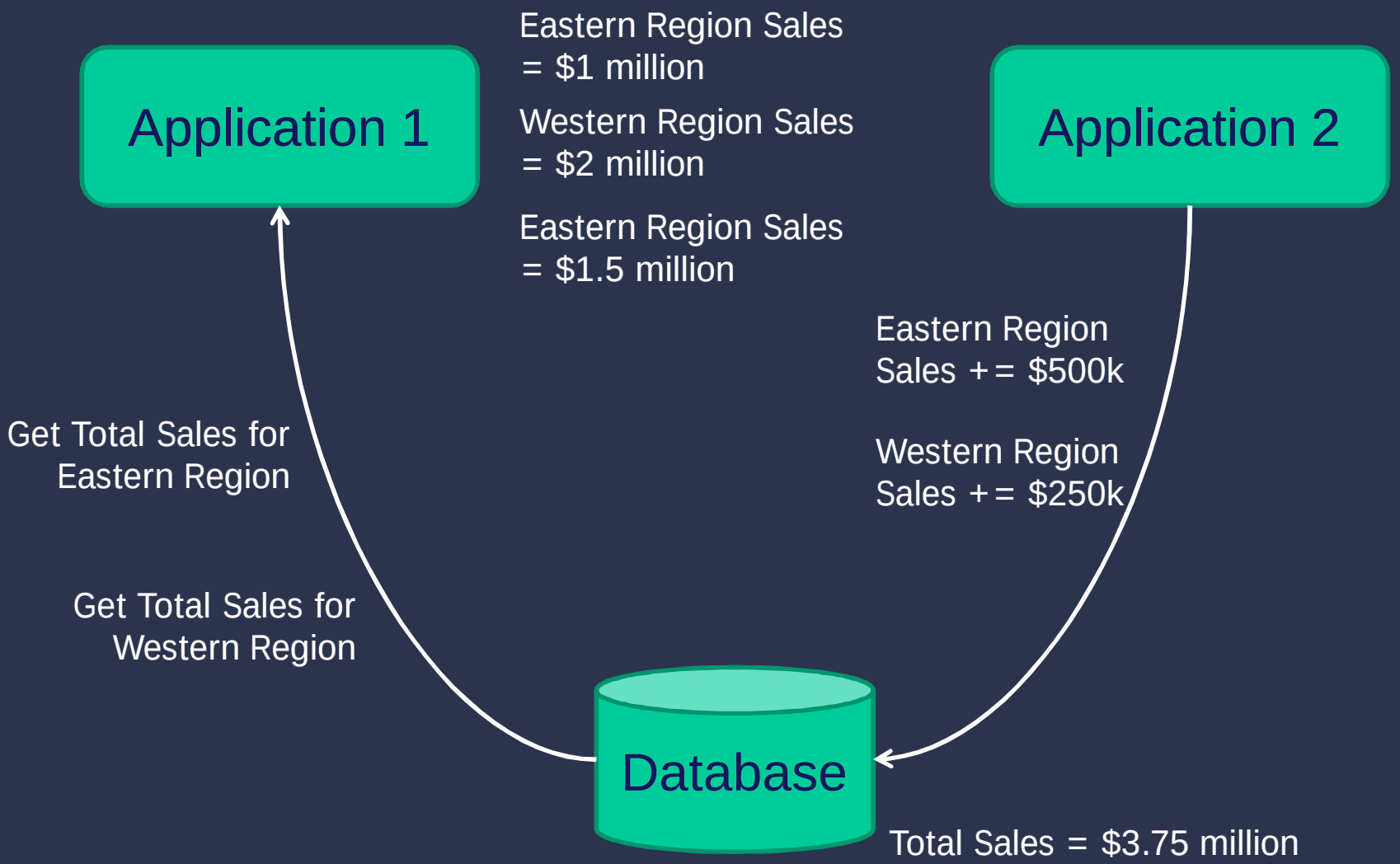


DevTeach

The Lost Update Problem

- Last one in wins
  - No warning to loser

Unrepeatable Reads

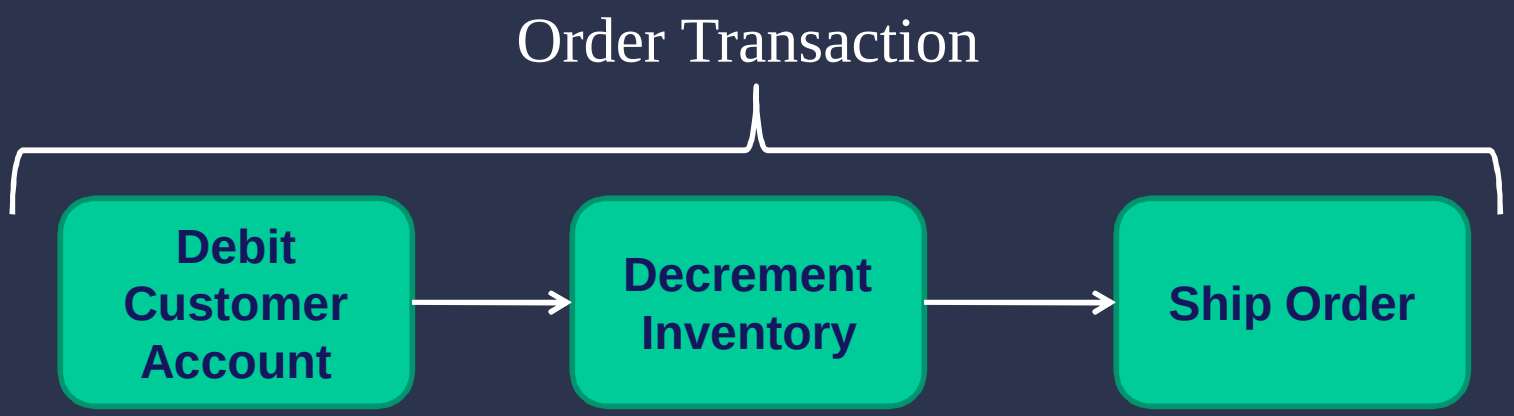


Unrepeatable Read

- "Reader applications" are unable to get the same results with each read of the same data
  - Occurs because "Writer applications" are operating on the same data
  - i.e. Read and Write operations are interleaved

What is a Transaction?

- A logical unit of work
  - May contain one or many steps



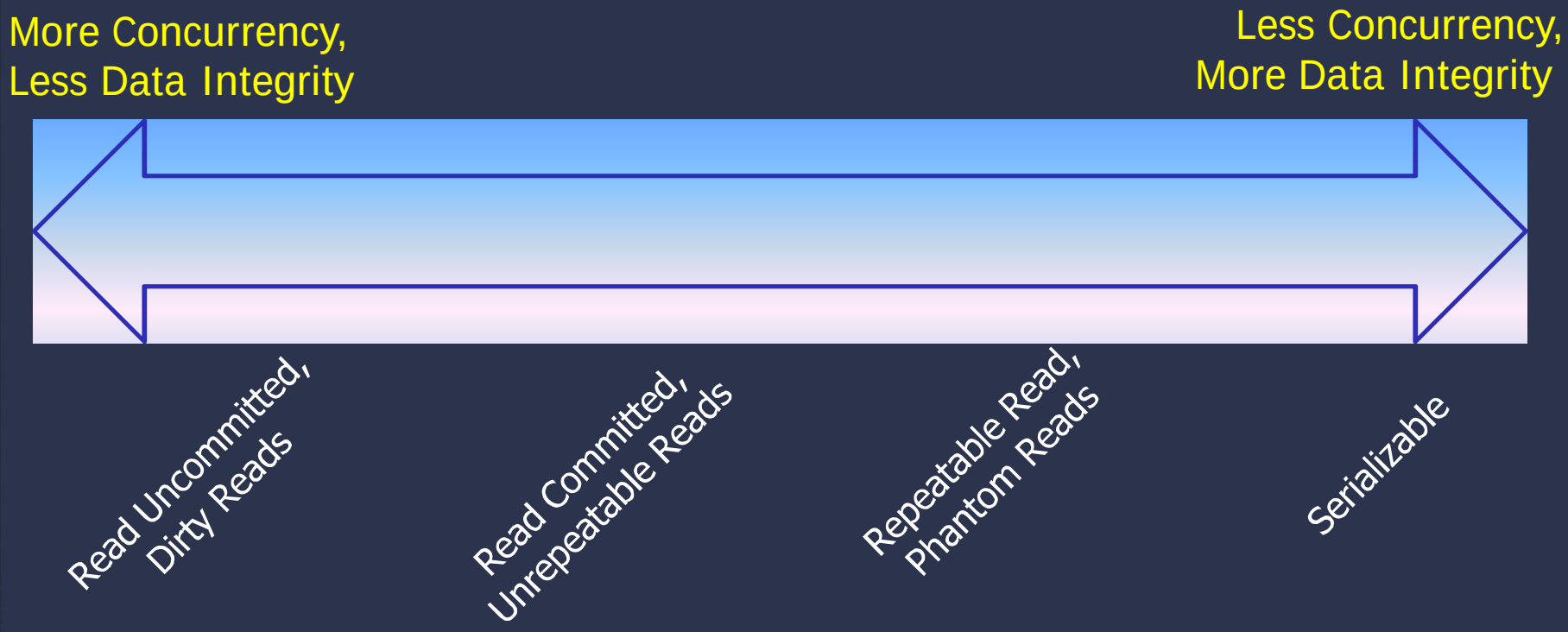
DevTeach

ACID

- **Atomicity**
  - "It all goes or none of it goes"
- **Consistency**
  - Data must be valid before and after the operation
- **Isolation**
  - Ensure that intermediate states of data aren't visible until committed
- **Durability**
  - The result of the data operation is successfully persisted to the resource and can survive crashes or exceptions

Transaction Isolation Levels

- Typically set at the database or by the ORM



Patterns for Concurrency

- **Patterns of Enterprise Application Architecture** ***(David Rice)***
  - Meant to prevent the Lost Update Problem
  - Controlled by custom code, Datasets, ORMs, etc.
- **Optimistic Offline Lock**
  - Guid or Timestamp is checked at commit
  - Downside = potential for lost work
- **Pessimistic Offline Locks**
  - Preference is given to ensuring no lost work
  - Takes early “Logical Exclusive Locks”
  - Bad for high concurrency, scalability
    - What if you don’t release your locks?

DevTeach

Distributed Transactions

DevTeach

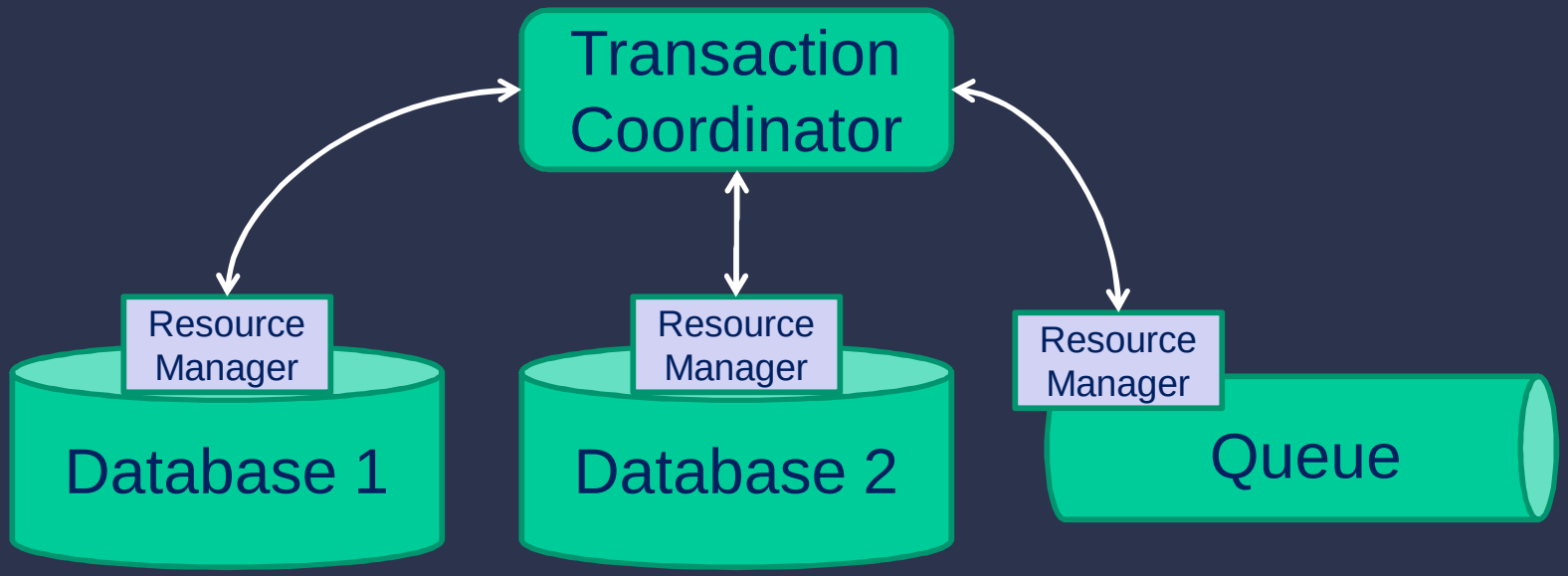
Distributed Transactions

- Transaction that spans multiple resources
  - Databases, queues, files
- Key Concept = Two-Phase Commit (2PC)
- SIDEBAR: WS-Atomic Transactions
  - Defines how distributed transactions are propagated across services
    - WS* Services (i.e. SOAP/WSDL)
  - Keep this in mind if you use WCF !!!!

DevTeach

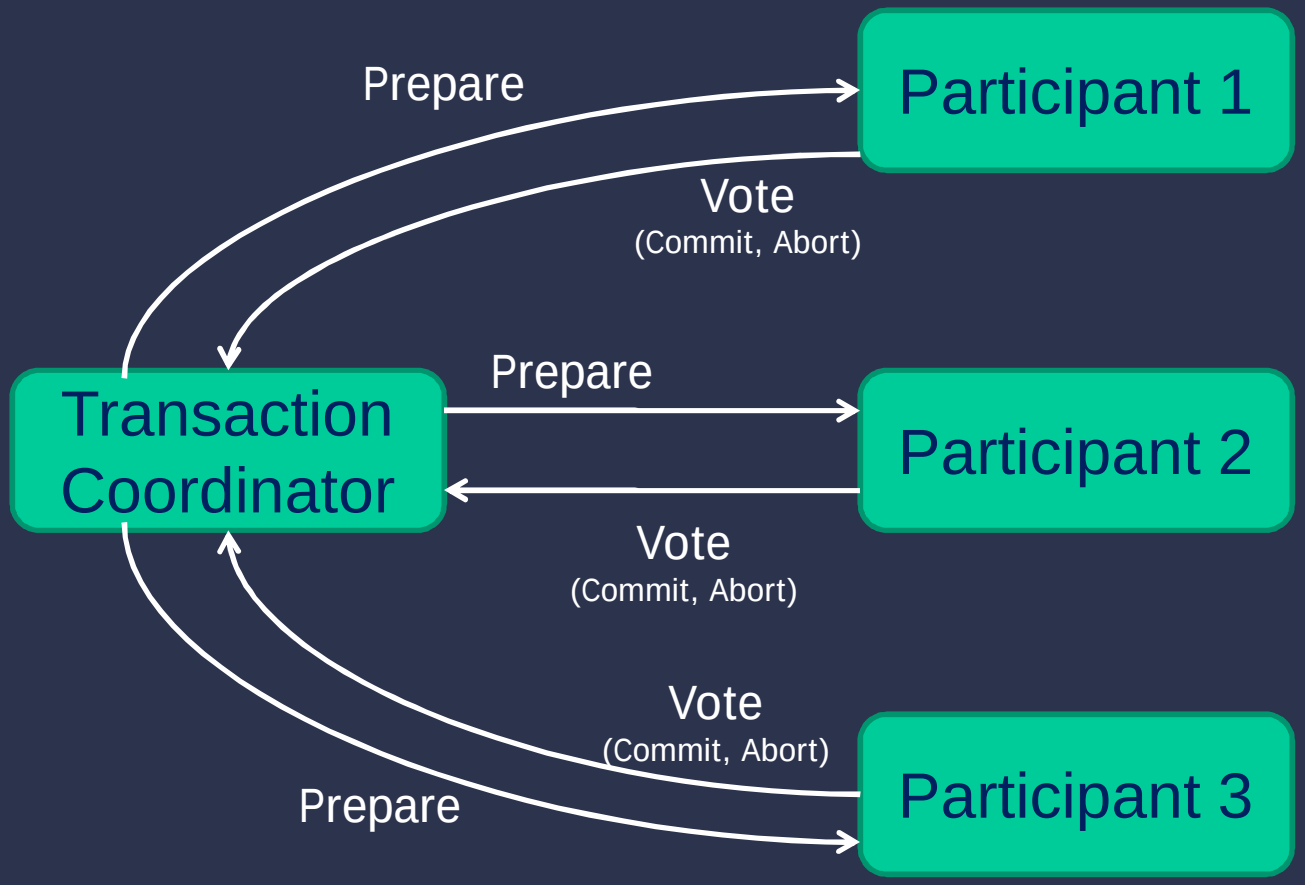
2PC Roles

- Transaction Coordinator (TC)
- Resource Managers (RM), Transaction Participants



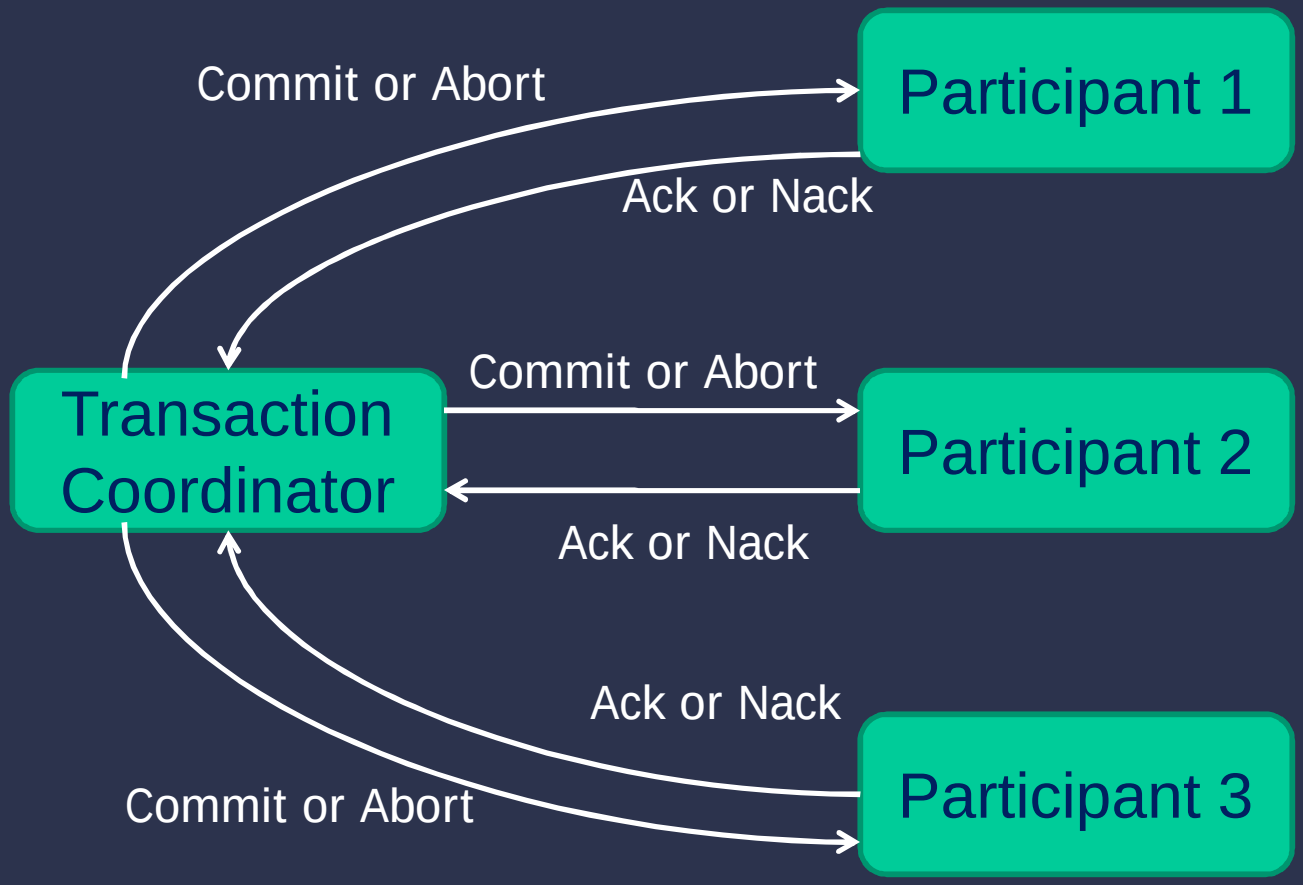
2PC Phase 1

- TC acquires votes from each participant



2PC Phase 2

- TC evaluates votes and tells participants to commit or abort

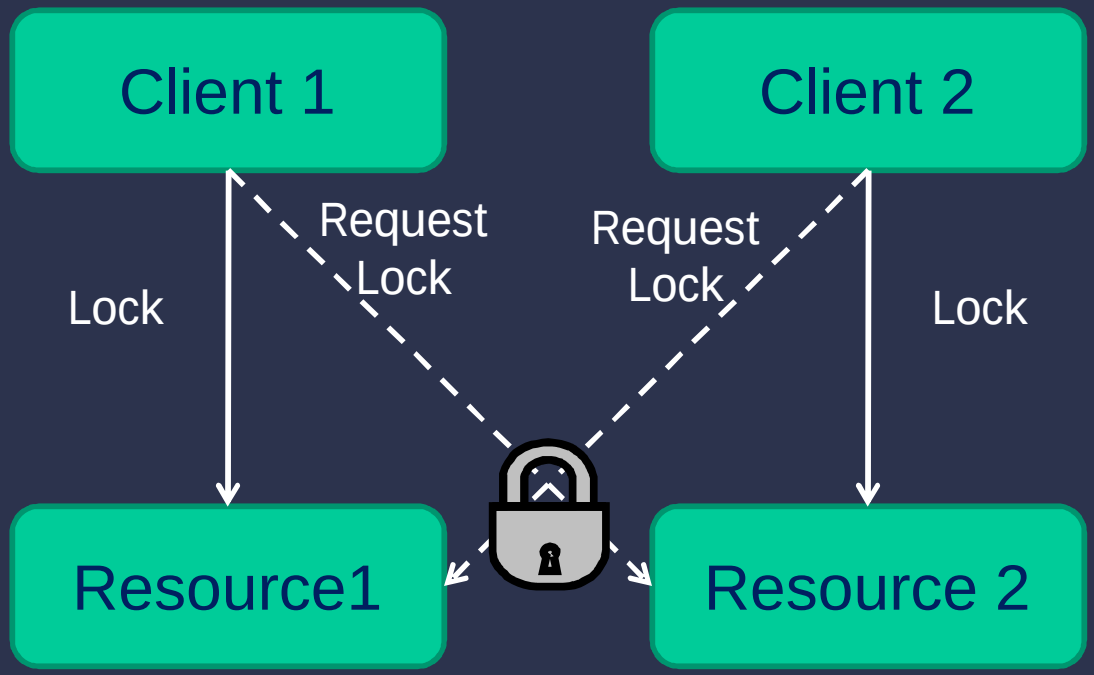


The Key Problem with 2PC

- Resource locks may be held for excessive lengths of time
  - Each participant has to wait for …
    - Other participants to complete
    - TC to tell each participant to commit or abort
- Result =
  - Client requests can be blocked, time-out
    - Also depends on Isolation level you set
  - Worst-case scenario = Deadlocks

The Deadly Embrace … Deadlocks

- The odds of deadlocks are increased with 2PC



Other Problems with 2PC

- These problems may occur during any phase …
- If TC fails, RMs may block indefinitely
  - In many cases you'll have to kill the transaction
- If TC directs participants to commit, and a participant is down …
  - TC and other participants may hang
- An extremely chatty protocol
  - Lots of network traffic, really poor performance for web services
- When should the TC decide that the transaction should time out?
- Increases coupling, decreases service autonomy

2PC Considerations

DevTeach

Are You Actually Dealing with Distributed Resources?

- Are you writing to multiple resources?
  - i.e. Physical databases, queues, files, etc.
  - e.g. Have you created a WCF services that uses transactions, yet writes to the same resource?
    - WCF will automatically use 2PC (rather than local transactions) when transactions are flowed, even if against the same resource

WCF Example

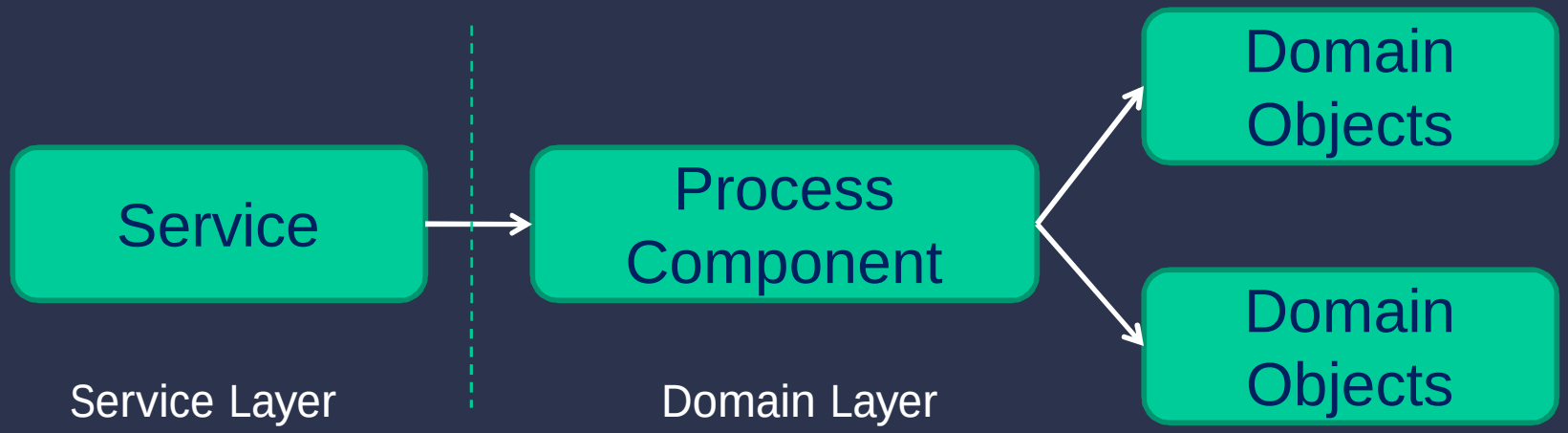
```
<bindings>
  <wsHttpBinding>
    <binding name =
          “Dangerous”
    transactionFlow=true />
  </wsHttpBinding>
</bindings>
```

```
[ServiceContract]
interface IFooBar
{
  [OperationContract]
  [TransactionFlow(
            TransactionFlowOption.Allowed)]
  SomeMessage FooBar(SomeMessage request);
}

public class FooBarClass: IFooBar
{
  [OperationBehavior(
          TransactionScopeRequired=true,
          TransactionAutoComplete=true)]
  public SomeMessage FooBar(SomeMessage request)
  {
    // implementation here;
  }
}
```

What to Do?

- Consider Refactoring
  - Rather than calling multiple services …
  - Refactor and consolidate business logic
    - Reposition common business logic in the Domain Layer rather than the Service Layer
    - Let the Domain Layer components manage the transaction



You are Dealing with Distributed Resources …

- Do you “own” the resources, or do they exist in some other domain or business?
  - 2PC may be Ok if …
    - You own the resources AND
    - You make sure that the transactions stay “relatively short” in duration AND
    - Concurrency problems are rare OR
    - You’re Ok with lower Isolation Levels (e.g. Dirty Reads)
  - Example
    - Reading a message from a queue and writing data from the message to a database

The CAP Theorem

DevTeach

A Fundamental Trade-off

- *www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf*
- Dr Brewer contends that a fundamental trade-off exists …

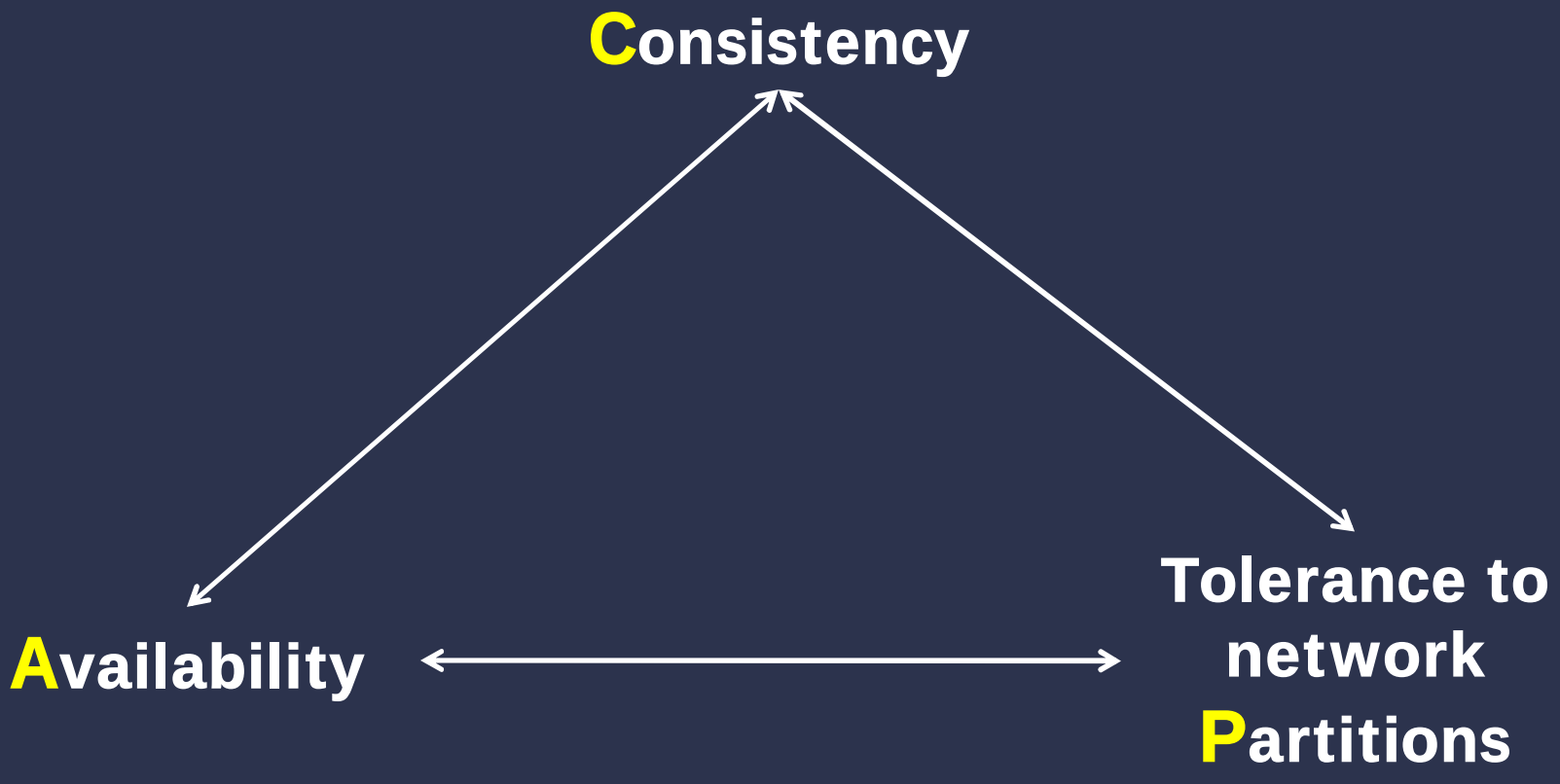
High Availability

Data Consistency

The CAP Theorem

- *Brewer, Dr Eric A; www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf*

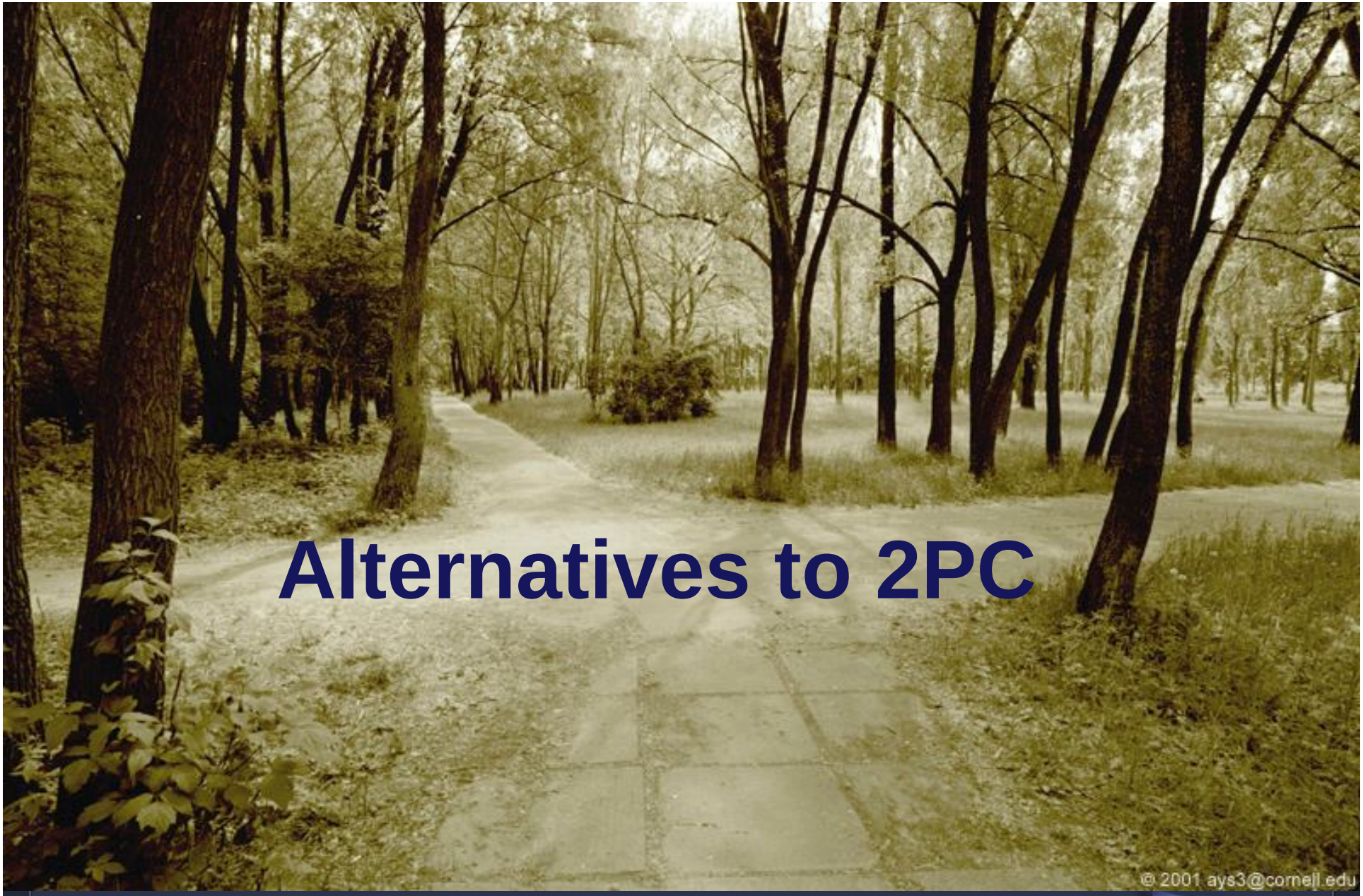
Pick two …



DevTeach

The Basic Trade-off

- Highly Available and Scalable distributed systems …
  - Forgo up-to-the-minute data consistency
    - Stale data can be Ok for a while
    - Assume that data consistency will catch up
- Consider the following …
  - Travel web sites (e.g. flight reservations)
  - Financial services (e.g. trades)



Alternatives to 2PC

DevTeach

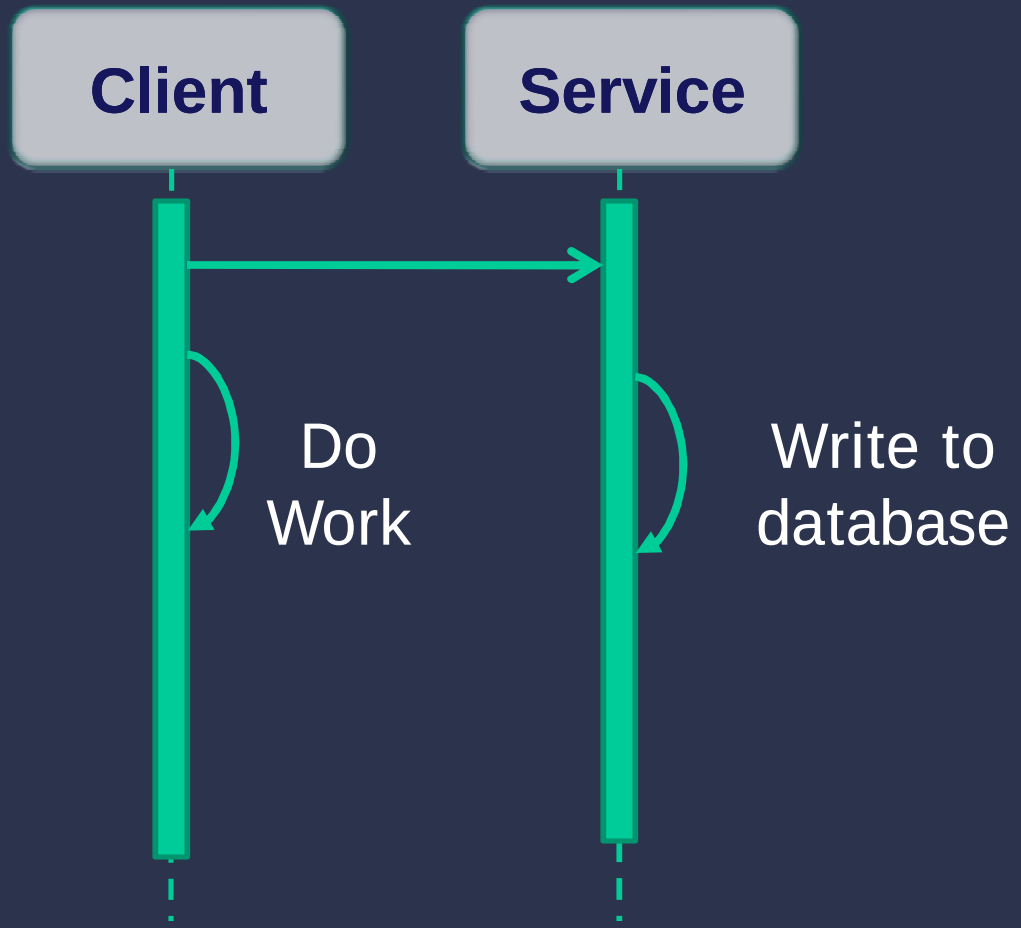
What to Do?

- How can distributed systems achieve high availability and scalability?
  - Forget 2PC
  - Do use Optimistic Off-line Locks
  - Set “Expiration Dates” on data and locks
    - Initiate “Data Refresh Requests” on expiration
- Partitioning
  - Provides isolation, independent scaling and tuning for a functional domain
  - Contrasted with large, monolithic systems

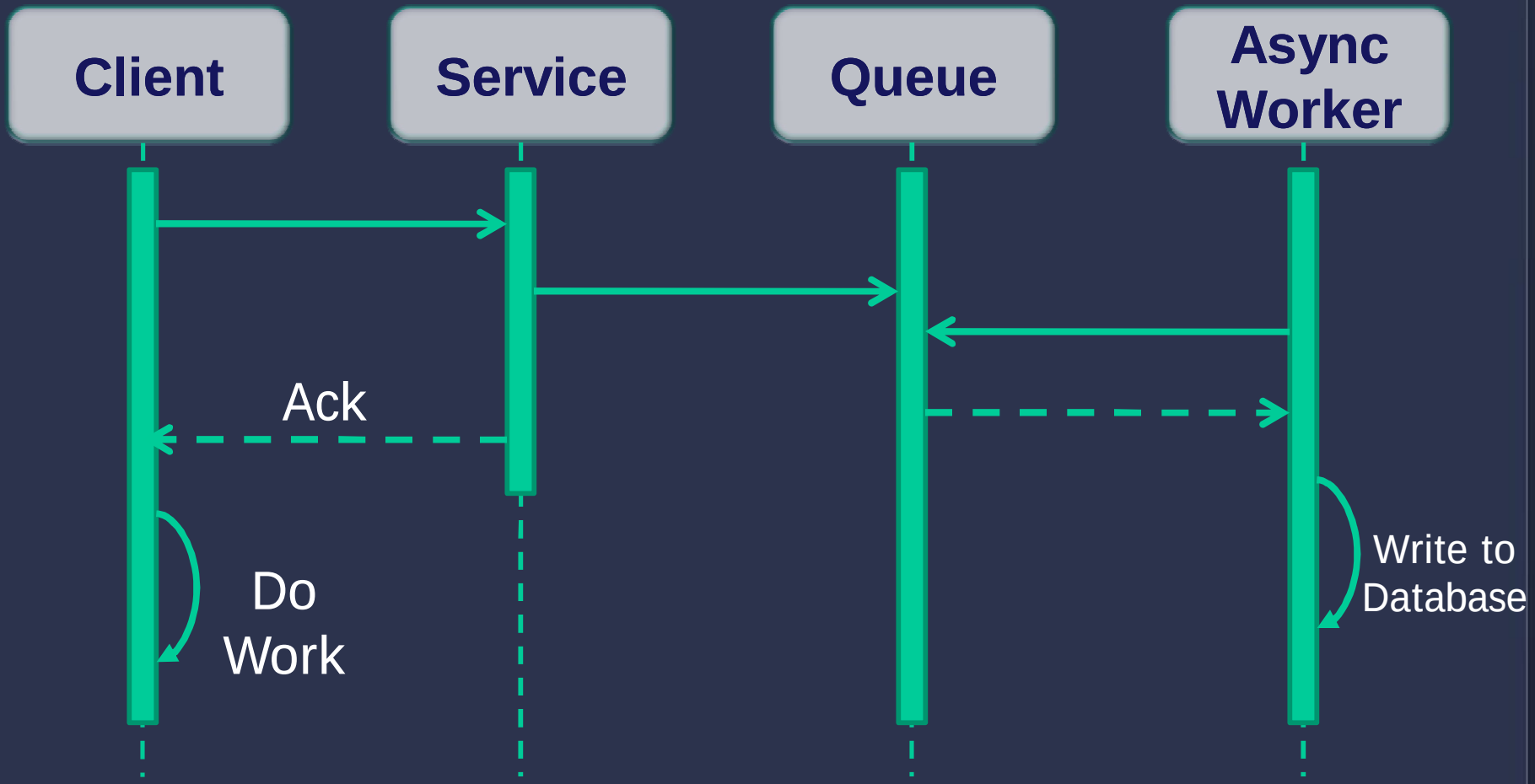
Consider Asynchronous Messaging

- Key Enablers …
  - Queues
  - Messages
- Service Design Patterns to Consider …
  - Event Sink
  - Request/Acknowledge
  - Asynchronous Response-Pull
  - Asynchronous Callback Message
- Compensating Transactions

Event Sink



Request/Acknowledge

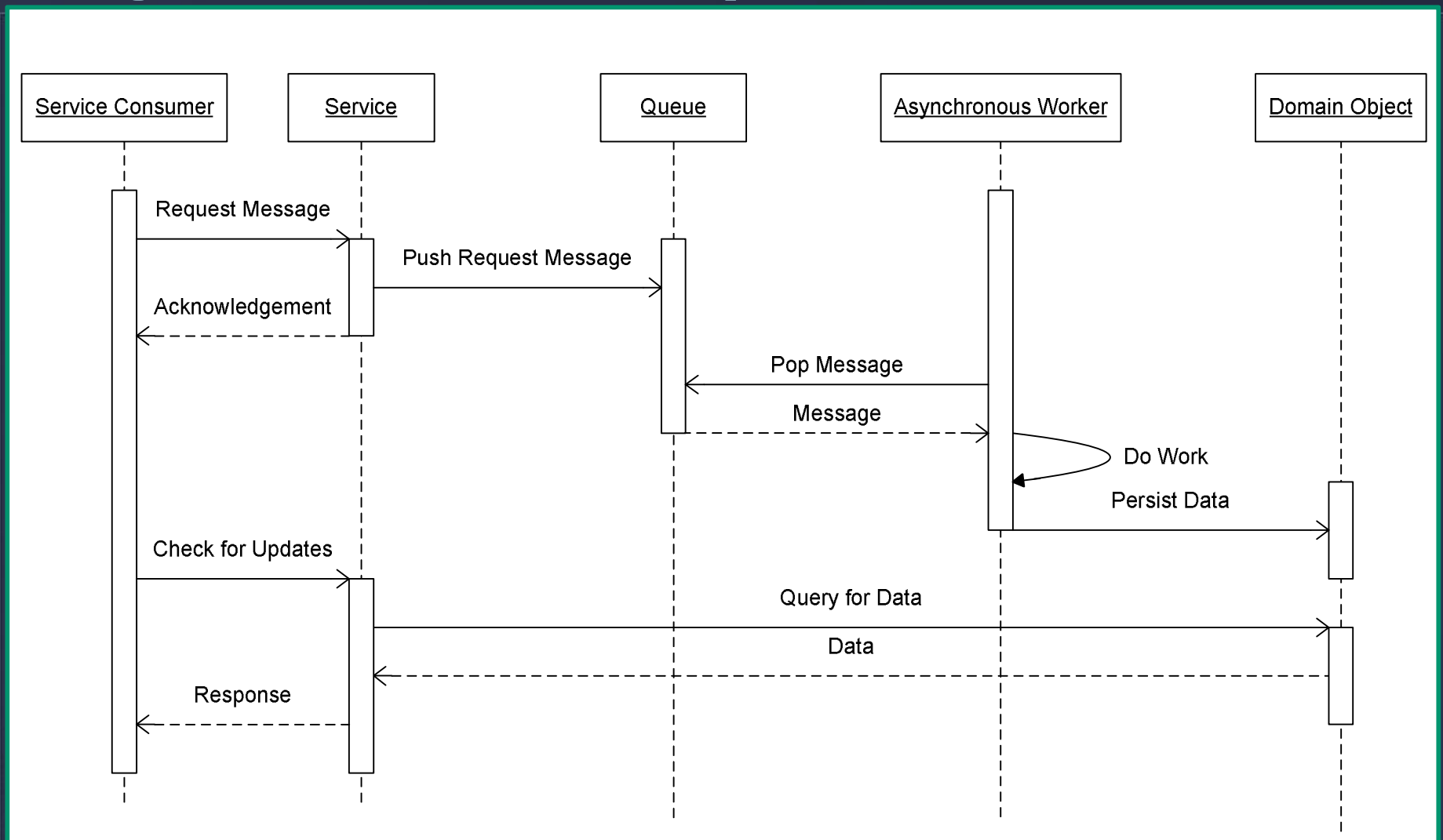


Pattern Considerations

- Event Sinks
  - Provide no means for clients to discover errors
- Request/Acknowledge
  - Nacks (Negative Acknowledgements)

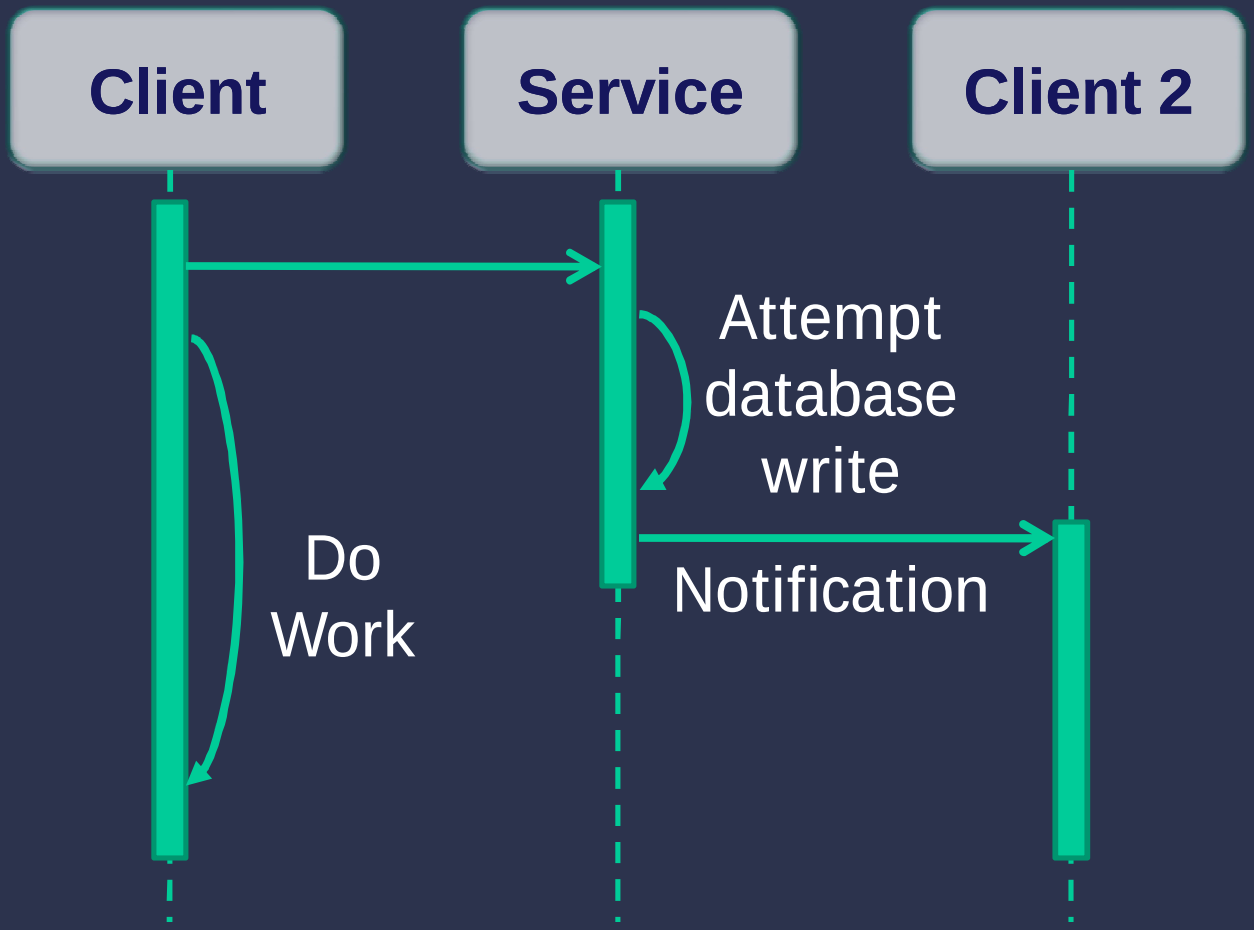
    Usually only tell you about problems with Authentication or Data Validation
- Questions
  - How does client learn about errors?
  - How can client undo operations in remote systems?

Asynchronous Response-Pull



Asynchronous Callback Message

- *a.k.a. Message Relay*



Compensating Transactions

- A command or message that is the logical inverse of some previous command or message

  e.g.

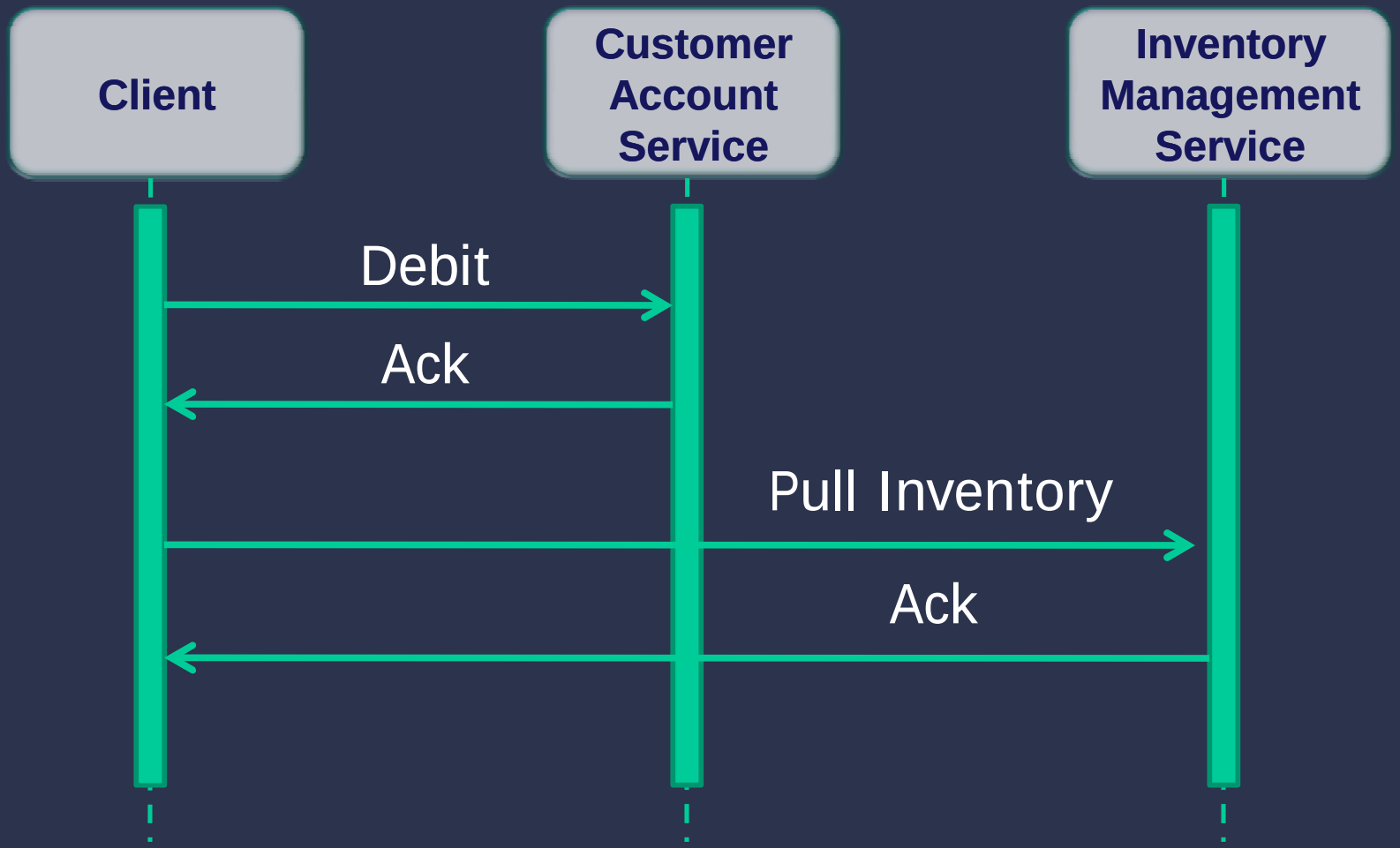
  - Reserve a seat on a Flight, Cancel Flight Reservation
  - Debit Account, Credit Account

- This is the way “Long-Running Transactions” work in the real world

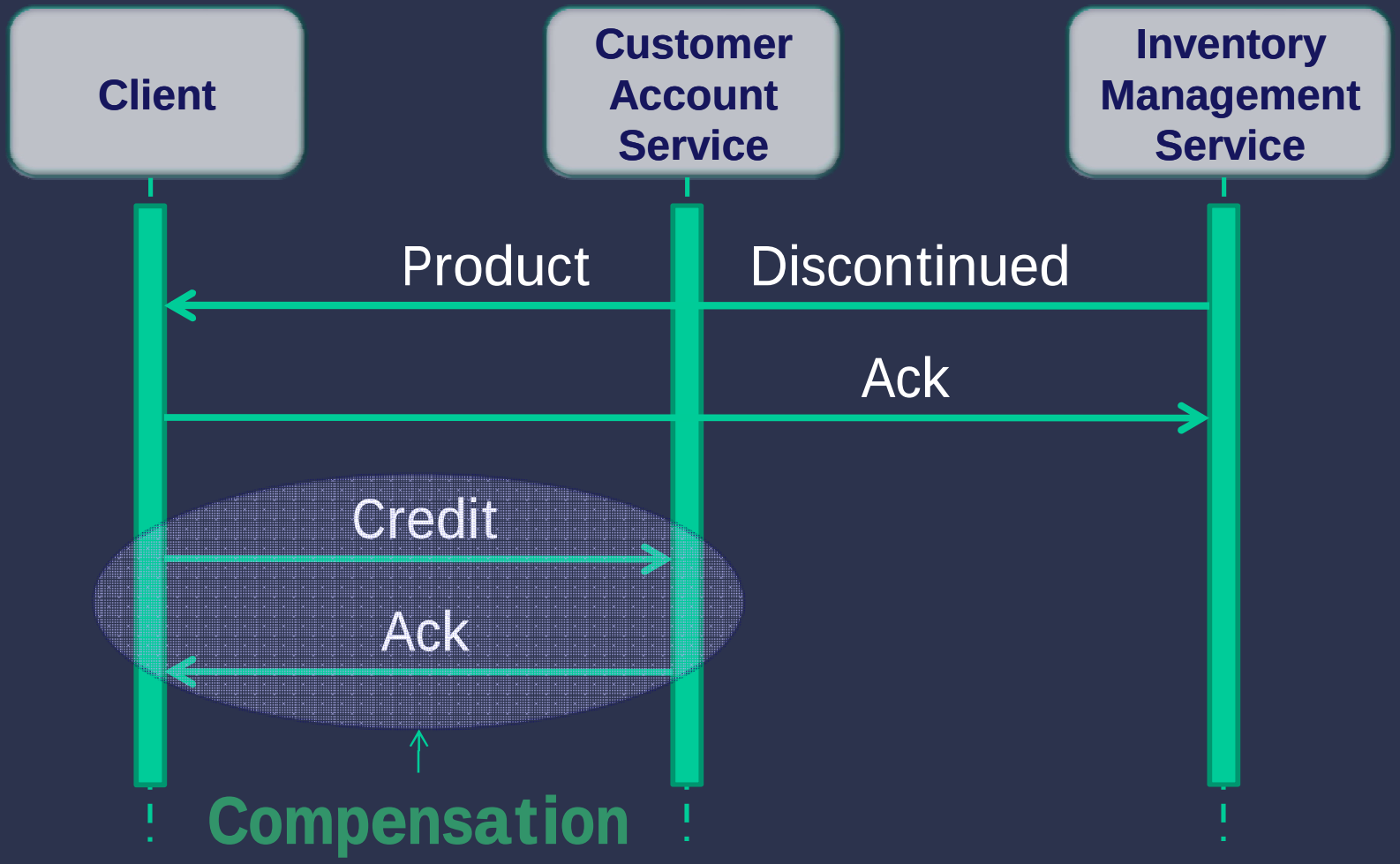
SIDEBAR:
Long-Running businesses processes aren’t really Transactions from the technical perspective

- They are neither **A**tomic or **I**solated.

Compensating Transaction Example, Part 1



Compensating Transaction Example, Part 2



Note: The client is usually not the same instance that sent out the original messages

DevTeach

Prerequisites for Compensating Transactions

- A Coordinator must …
  - Keep record of data commands or messages submitted to each participant
  - Keep record of the sequence of these calls
  - Have a way to associate each call with a unique Instance of the Long-Running Process (i.e. Conversation ID)
  - Be able to associate any Asynchronous Callback Message with a specific Process Instance
  - Know how to “play back” the appropriate compensations for a given Process Instance with the right data

Sounds Complicated …

- It is …
  - Compensations can fail too
  - The data may be visible between the time it is set by the original request and the time that the compensation occurs
- Options …
  - You can “roll your own”
    - Many companies do
  - On the Microsoft platform you can use …
    - Micro-Flows (e.g. Windows Workflow)
    - Orchestrations (e.g. Biztalk)

Conclusion

DevTeach

References

- Hohpe, Gregor; Woolf, Bobby; *Enterprise Integration Patterns. Designing, Building, and Deploying Messaging Solutions*. Addison Wesley
- Hohpe, Gregor; *Your Coffee Shop Doesn't Use Two-Phase Commit*
  - http://www.eaipatterns.com/docs/IEEE_Software_Design_2PC.pdf
- http://en.wikipedia.org/wiki/Two-phase_commit_protocol
- Brewer ,Dr. Eric A. *Towards Robust Distributed Systems*. *www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf*
  - This presentation describes the CAP Theorem
- Transaction Management in the R* Distributed Database Management System
  - http://www.cs.cmu.edu/~natassa/courses/15-823/F02/papers/p378-mohan.pdf
- http://www.DesignPatternsFor.Net