Introduction to Data Management
CSE 344

Lecture 28: Finale

Announcements

- HW8 is due on tonight
- Final exam
  - Monday Dec 12, 2:30 – 4:20pm, JHN 102
  - Closed book, you can bring 2 sheets of notes
  - Content: everything
  - Closed book
  - Review session this Saturday afternoon: EEB 125, 3:30-4:30pm

How To Study

- Go over the lecture and section notes
- Read the book
- Go over the assignments
- Practice
  - Practice web quiz posted
  - Finals & midterms from past 344s
- Ask course staff questions!
- The goal of the final is to help you learn!

Today

- Implement page rank using Spark

- Selected topics

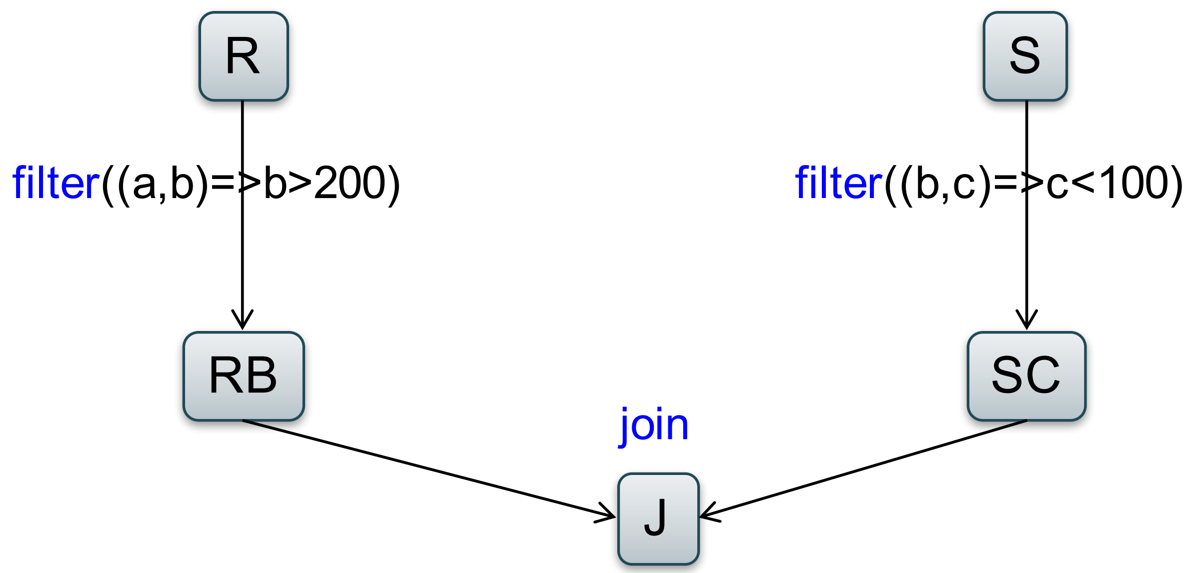
R(A,B)
S(A,C)

```
SELECT count(*)  FROM R, S
WHERE R.B > 200 and S.C < 100  and R.A = S.A
```

Example

```
R = spark.textFile("R.csv").map(parseRecord).persist();
S = spark.textFile("S.csv").map(parseRecord).persist();
RB = R.filter((a,b) => b > 200).persist();
SC = S.filter((a,c) => c < 100).persist();
J = RB.join(SC).persist();
result = J.count();
```

result:



Transformations:	
map(f : T => U):	RDD[T] => RDD[U]
flatMap(f: T => Seq(U)):	RDD[T] => RDD[U]
filter(f:T=>Bool):	RDD[T] => RDD[T]
groupByKey():	RDD[(K,V)] => RDD[(K,Seq[V])]
reduceByKey(F:(V,V) => V):	RDD[(K,V)] => RDD[(K,V)]
union():	(RDD[T],RDD[T]) => RDD[T]
join():	(RDD[(K,V)],RDD[(K,W)]) => RDD[(K,(V,W))]
cogroup():	(RDD[(K,V)],RDD[(K,W)]) => RDD[(K,(Seq[V],Seq[W]))]
crossProduct():	(RDD[T],RDD[U]) => RDD[(T,U)]

Outputs 1 object per input (map)

Output multiple objects per input (flatMap)

Actions:	
count():	RDD[T] => Long
collect():	RDD[T] => Seq[T]
reduce(f:(T,T)=>T):	RDD[T] => T
save(path:String):	Outputs RDD to a storage system e.g. HDFS

Outputs 1 object per input (reduce)

Many Shades of `persist()`

- `persist()` executes the computation and caches the results in memory
- But if memory runs out, then some parts are not computed
- You can set the level of persistence using different parameters

persist() parameters

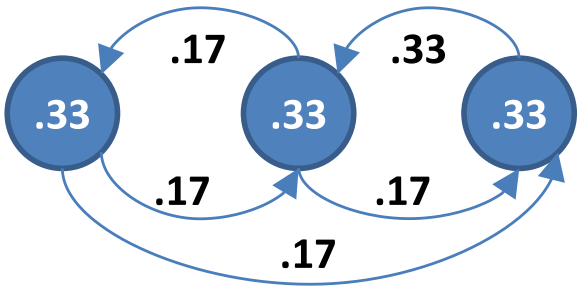
Storage Level	Meaning
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER (Java and Scala)	Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER (Java and Scala)	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as the levels above, but replicate each partition on two cluster nodes.
OFF_HEAP (experimental)	Similar to MEMORY_ONLY_SER, but store the data in off-heap memory. This requires off-heap memory to be enabled.

Another Example: PageRank

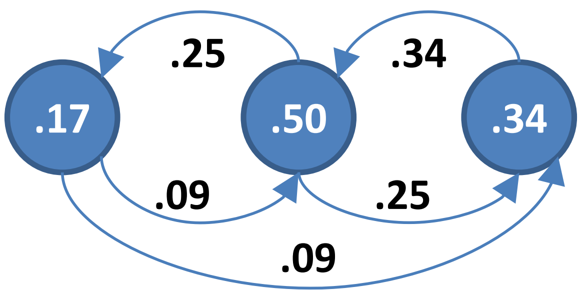
- Page Rank is an algorithm that assigns to each page a score such that pages have higher scores if more pages with high scores link to them

- Page Rank was introduced by Google, and, essentially, defined Google

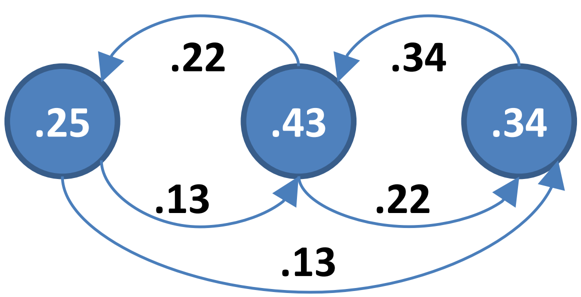
PageRank toy example



Superstep 0

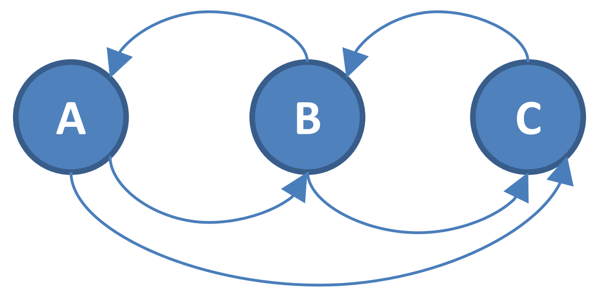


Superstep 1

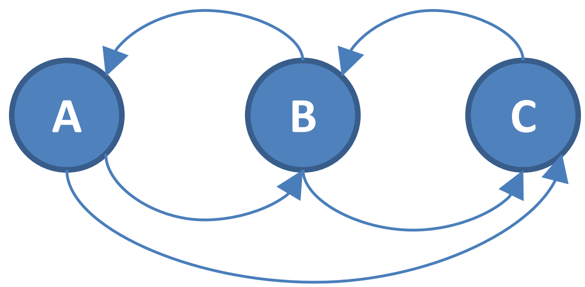


Superstep 2

Input graph



http://www.slideshare.net/sscdotopen/large-scale/20



PageRank

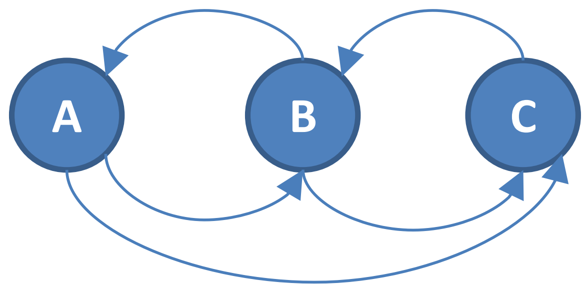
```
for i = 1 to n:
   r[i] = 1/n

repeat
   for j = 1 to n: contribs[j] = 0
   for i = 1 to n:
      k = links[i].length()
      for j in links[i]:
          contribs[j] += r[i] / k
    for i = 1 to n: r[i] = contribs[i]
until convergence

/* usually 10-20 iterations */
```

r[i] = prob. that we are at node i

Random walk interpretation:

Start at a random node i
At each step, randomly choose an outgoing link and follow it.

Repeat for a very long time



PageRank

```
for i = 1 to n:
  r[i] = 1/n

repeat
  for j = 1 to n: contribs[j] = 0
  for i = 1 to n:
    k = links[i].length()
    for j in links[i]:
       contribs[j] += r[i] / k
   for i = 1 to n: r[i] = contribs[i]
until convergence

/* usually 10-20 iterations */
```

```
links = spark.textFile(..).map(..);
// RDD of (URL, {links}) pairs
ranks = ... // RDD of (URL, 1/n) pairs

for (k = 1 to ITERATIONS) {
  // Build RDD of (targetURL, float) pairs
  // with contributions sent by each page
  contribs = links.join(ranks).flatMap {
    (url, (links,rank)) =>
     links.map(dest => (dest, rank/links.size))
   };
  // Sum contributions by URL and get new ranks
  ranks = contribs.reduceByKey((x,y) => x+y);
}
```

Conclusions

- Parallel databases
  - Predefined relational operators
  - Optimization
  - Transactions
- MapReduce
  - User-defined map and reduce functions
  - Must implement/optimize manually relational ops
  - No updates/transactions
- Spark
  - Predefined relational operators
  - Must optimize manually
  - No updates/transactions

Key Data Management Concepts

- **Data models**: how to describe real-world data
  - Relational, XML, graph data (RDF)
- **Schema**
- **Declarative query language**
  - Say what you want not how to get it
- **Data independence**
  - Physical independence: Can change how data is stored on disk without maintenance to applications
  - Logical independence: can change schema w/o affecting apps
- **Query optimizer** and compiler
- **Transactions**: isolation and atomicity

Review this slide during the quarter!

What is this class about?

- **Focus: Using DBMSs**
- Relational Data Model
  - SQL, Relational Algebra, Relational Calculus, datalog
- Semistructured Data Model
  - JSon, CouchDB (NoSQL)
- Conceptual design
  - E/R diagrams, Views, and Database normalization
- Transactions
- Parallel databases, MapReduce, and Spark
- Data integration and data cleaning

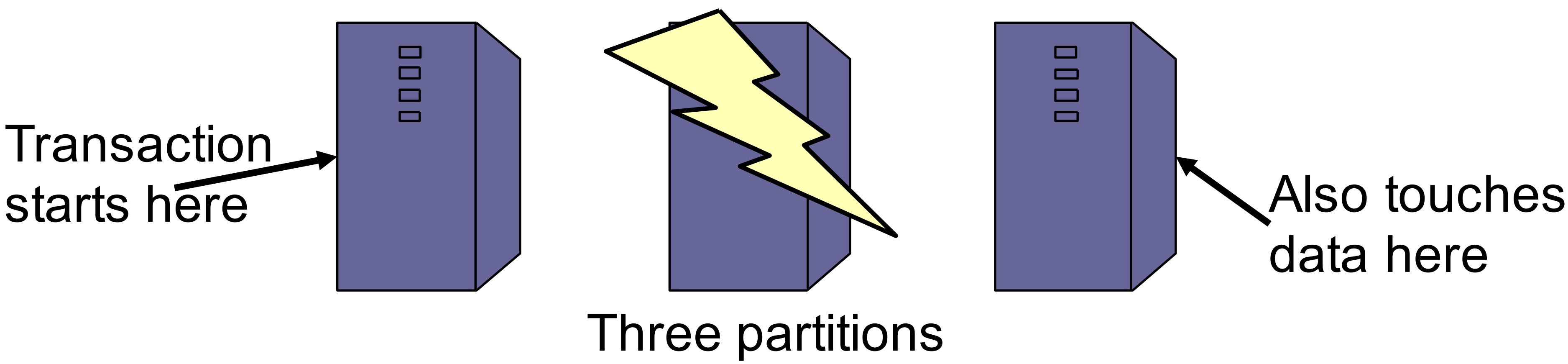
Distributed Transactions

Parallel DBMSs and Transactions

- Scaling a relational DBMS is hard
- We saw how to scale queries with parallel DBMSs
- Much more difficult to scale ***transactions***
- ***Because need to ensure ACID properties***
  - Hard to do beyond a single machine

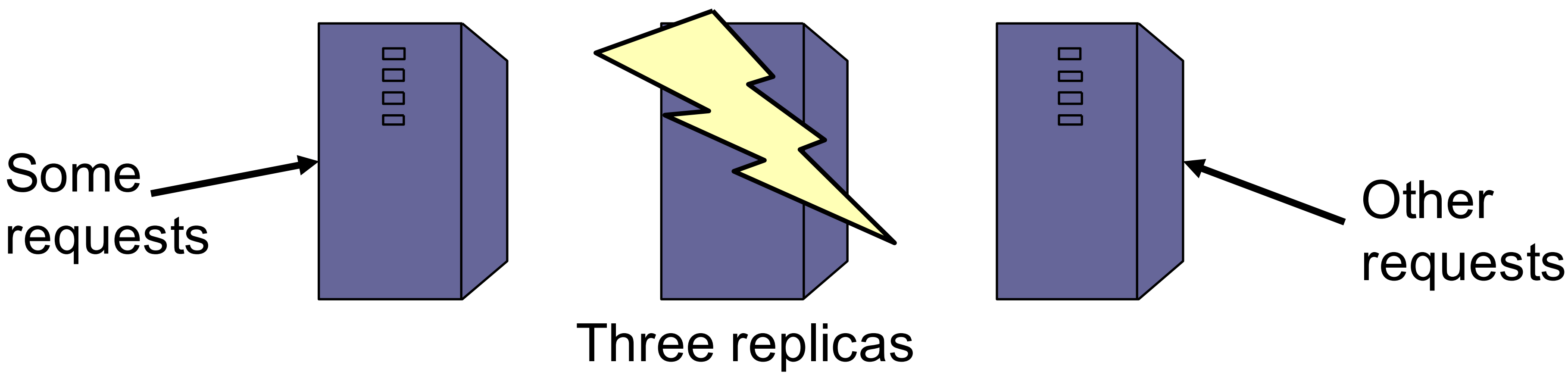
Scale Through Partitioning

- Partition the database across many machines in a cluster
- Spread queries across these machines
- Can increase throughput
- Easy for reads but writes become expensive!
- Need 2PC (two phase commit) to ensure serializability



Scale Through Replication

- Create multiple copies of each database partition
- Spread queries across these replicas
- Can increase throughput and lower latency
- Can also improve fault-tolerance
- Easy for reads but writes become expensive!



Scaling Transactions

- Need to partition the db across machines

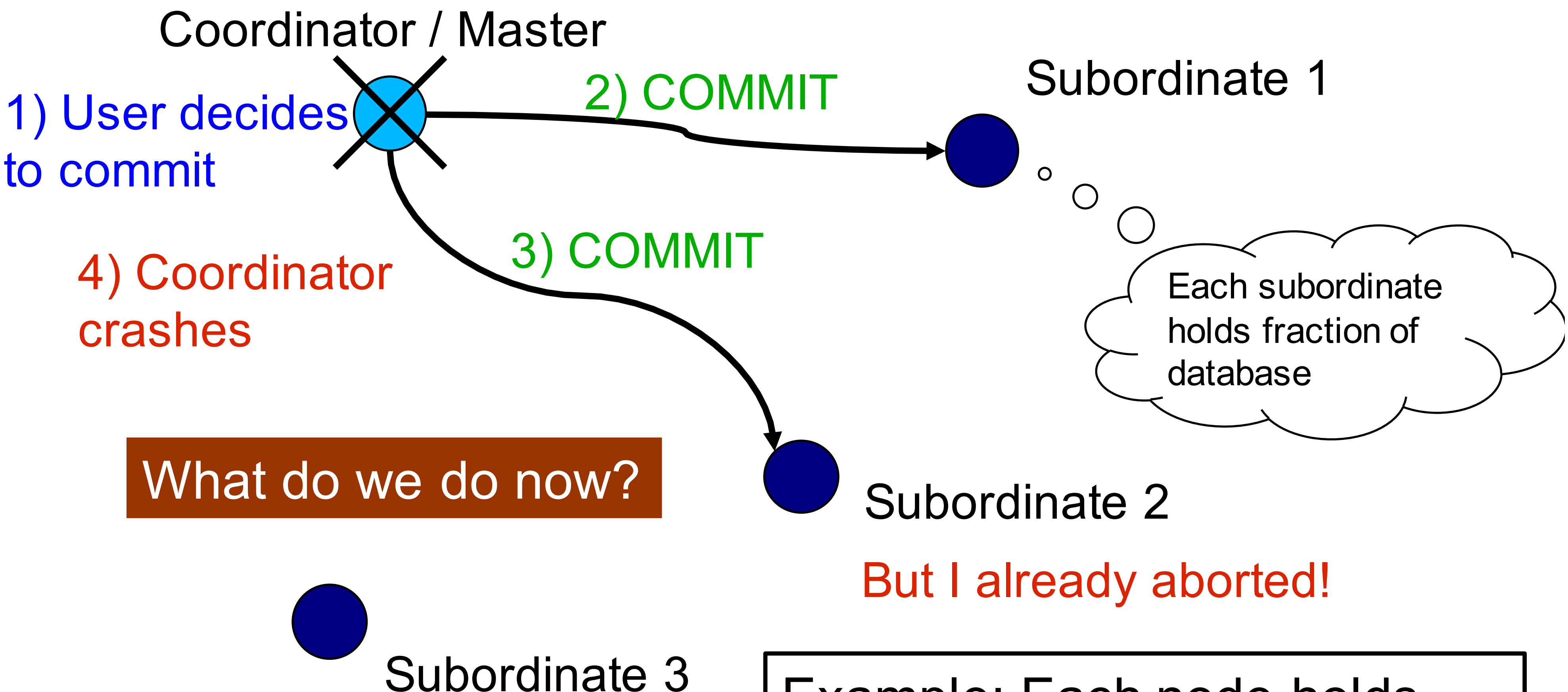
- If a transaction touches one machine
  - Life is good

- If a transaction touches multiple machines
  - ACID becomes extremely expensive!
  - Need two-phase commit

Two-Phase Commit: The Setting

- Data partitioned across multiple nodes
- Query touches multiple partitions and commits
- Lock multiple partitions at the same time

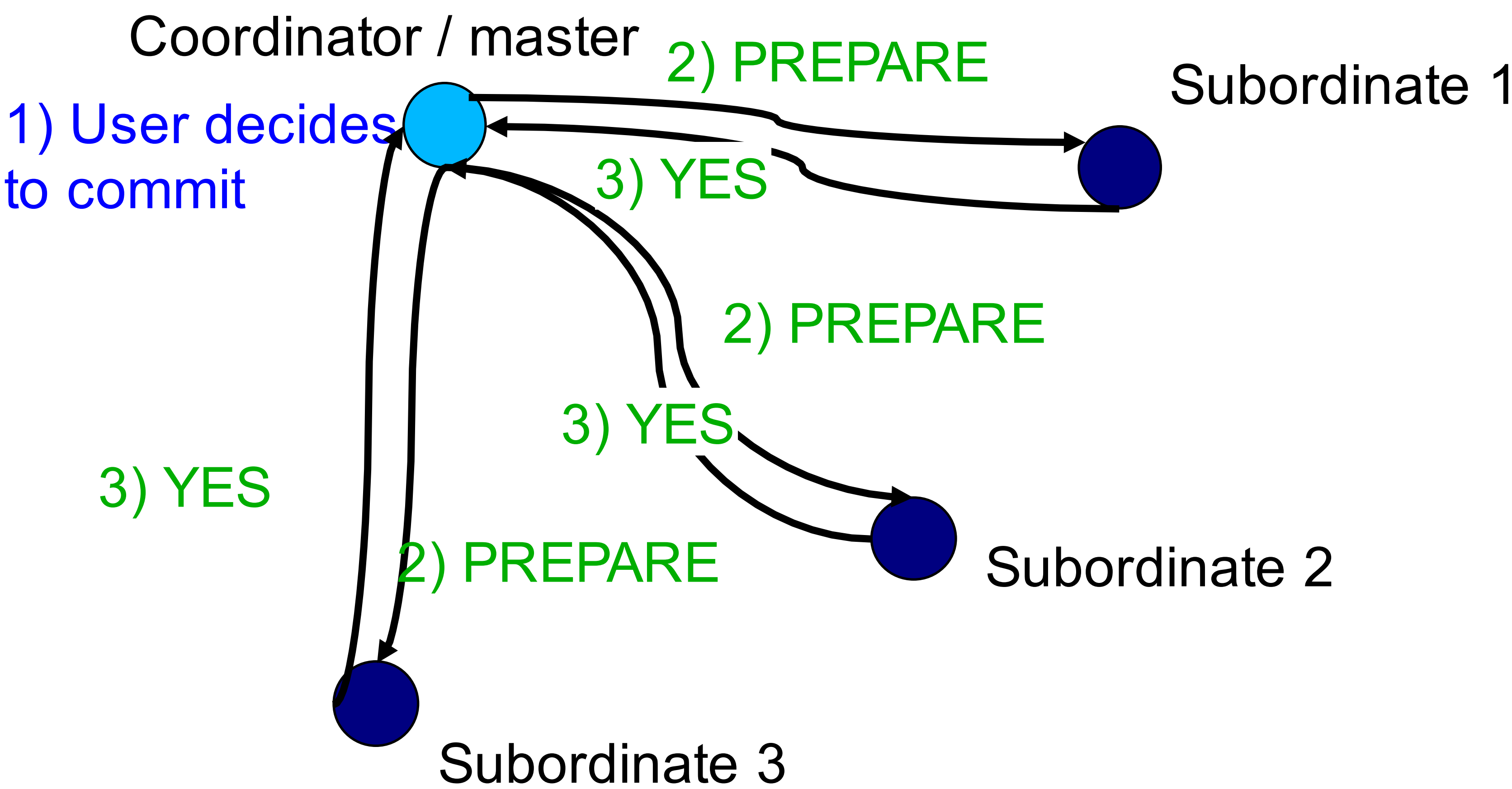
Two-Phase Commit: Motivation



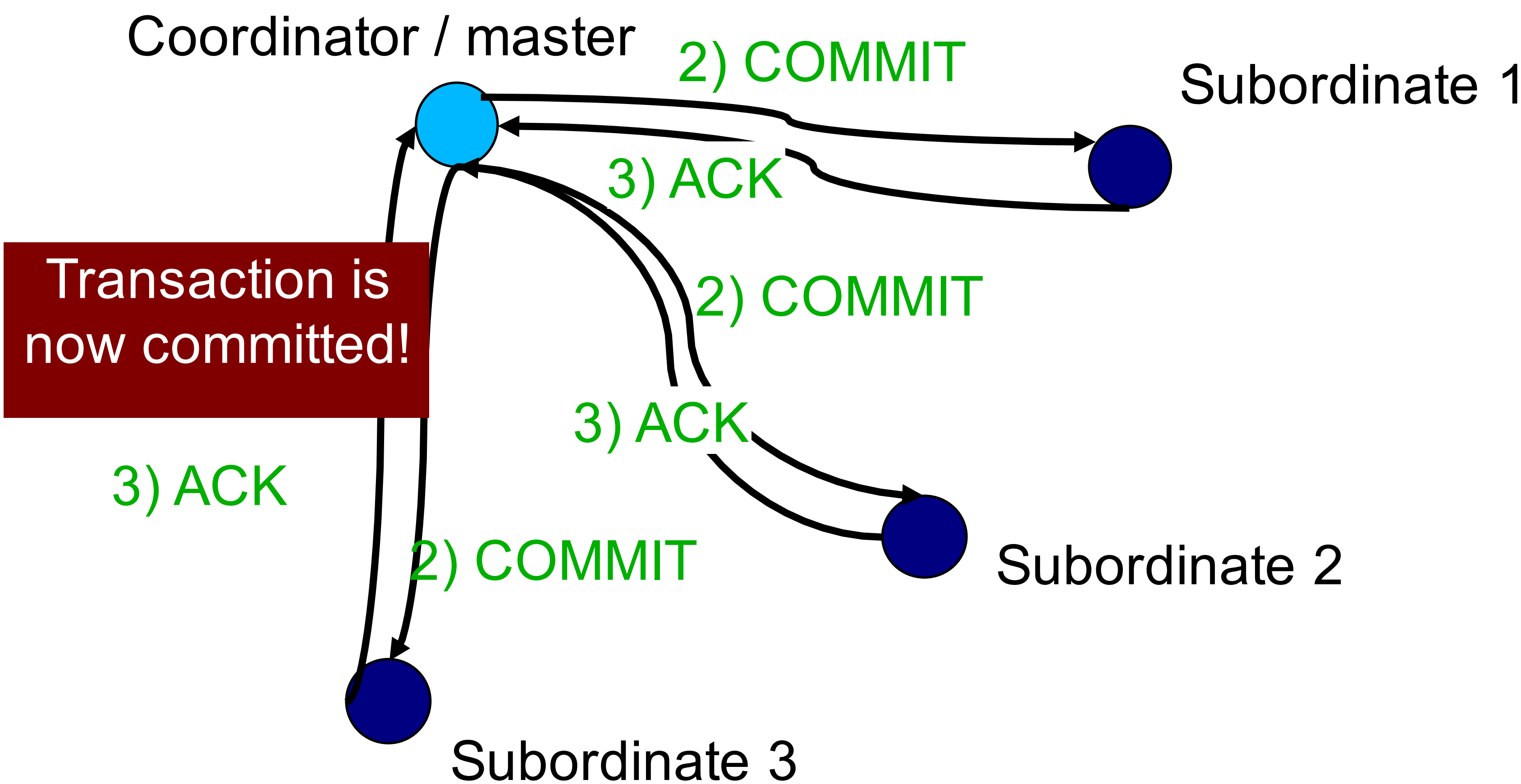
What do we do now?

Example: Each node holds some subset of bank accounts
Transaction transfers money

2PC: Phase 1 Illustrated

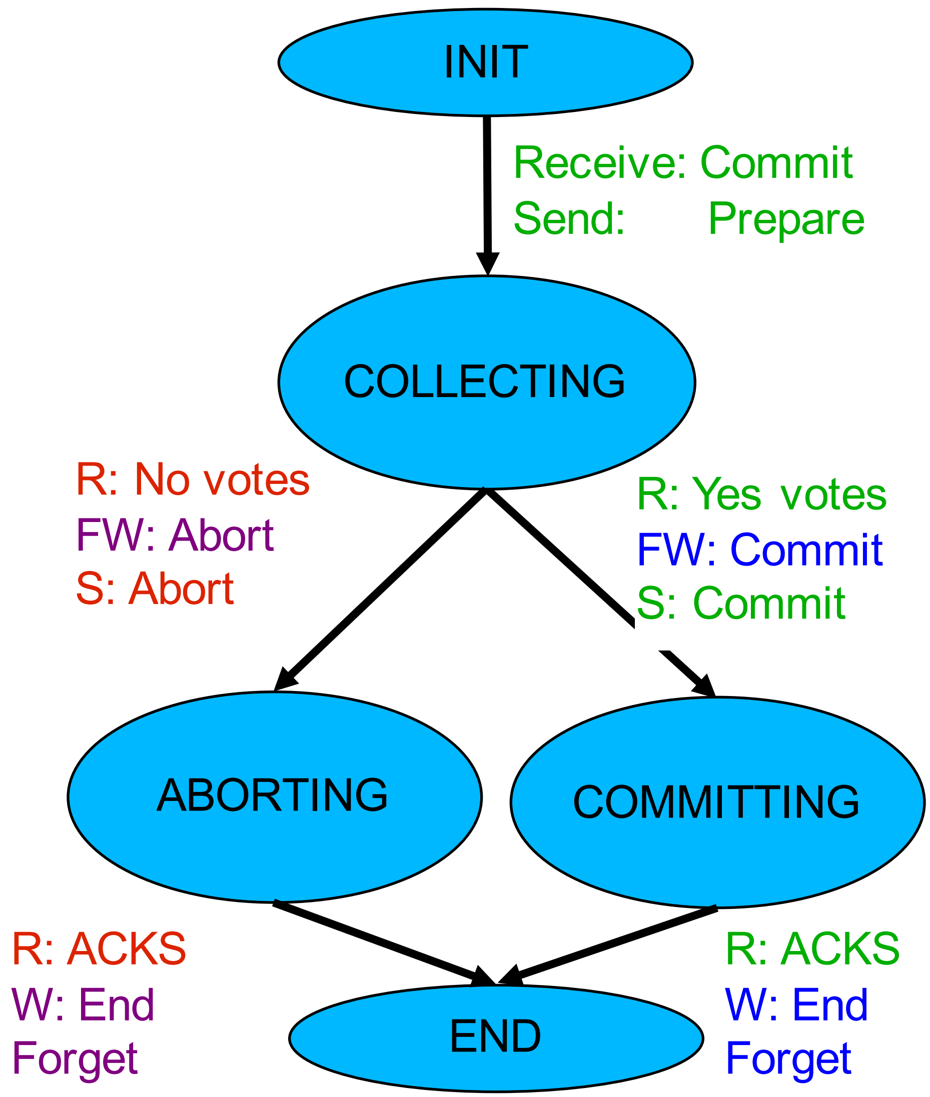


2PC: Phase 2 Illustrated



Coordinator State Machine

- All states involve waiting for messages



ACID vs BASE

- ACID = Atomicity, Consistency, Isolation, and Durability
- BASE = Basically Available, Soft state, Eventually consistent

Writing DB applications

Issuing Queries to DBMS

- Write SQL text on a command prompt provided by DBMS
  - These are called Command Line Interfaces (CLIs)
  - All major DBMS implementations provide this

- Write queries graphically
  - Essentially the same except that queries are constructed via GUIs
  - Advantages?

CLI

- This has been the only way to interact with DBMSs for the first 20 years or so
- Database applications = accounting, business processing
- Users were clerks / accountants in large corporations

IBM System/38

Rise of Programming Languages

- 3rd generation “high level” general purpose programming languages caught on starting in the 80s
- Users start to write apps in those languages instead
  - Procedural languages: Fortran, COBOL, C
  - Object-oriented languages: CLU, C++, Java
- Problem: those languages do not work well with SQL
  - Famous example: “impedance mismatch”

“Impedance” Mismatch

- Issues between general-purpose programming languages and query languages:
  - Data types
  - Object encapsulation, inheritance, polymorphism (for object oriented languages)
  - Transactions
  - Schema changes
  - Imperative and declarative programming styles
  - Security

Dealing with Impedance Mismatch

- Don’t use a DBMS (!)

- Object-Oriented DBMS (OO-DBMS)
  - Object instances directly stored in DBMS
  - Write GP code to access objects directly (no more SQL)
  - (yet another data model)
  - Popular in the 90s

  - Very difficult to optimize

Database Drivers

- RDBMS start to provide **drivers** for applications to access persistent data
- Idea: applications embed SQL strings within app code
- Examples with standardized interfaces:
  - ODBC (Open Database Connectivity)
  - JDBC (Java Database Connectivity)

- Each DBMS provides its own driver implementation

Issues with Drivers

- Users need to learn two languages
- Every driver is slightly different in its calling syntax
- Type safety?
- Software engineering nightmare
- Inefficient data serialization between DBMS and application
  - But at least you don't need to write the serialization code

Rise of the Internet

- Web applications become popular in the 2000s
- Database applications = web applications
  - online forums, online stores, etc
- Easy integration with the web server is important

Web Applications

- Typical three-tier web applications
  - Frontend (browser, phone, etc)
  - Middle tier (web server hosting the application)
  - Backend (databases)
- Embedding SQL strings within application becomes tedious and clumsy
  - You only need to learn SQL, php, Javascript, HTML, … to write web apps

Web Frameworks

- MVC design pattern
  - Model
    - Database schemas (e.g., SQL)
  - View
    - Presentation layer (e.g., HTML)
  - Controller
    - Application logic (e.g., php)

- Compare this to E/R diagrams

Web Frameworks

- Idea:
  - Declare models up front
    - i.e., what need to be persistently stored
  - Implement application logic using general purpose language
  - Web framework generates all necessary SQL and create database tables, indexes, etc
- Issue: still need to learn another language for the presentation layer
  - Some frameworks provide that capability as well

Web Frameworks

ASP.NET	PHP Fat-Free Framework	Koa	Zend	Stripes
AngularJS	Lift	web2py	Google Web Toolkit	Grok
Ruby on Rails	CherryPy	(fab)	Play	Zope
ASP.NET MVC	Restlet	Gin	Yii	Orbit
Django	Lithium	Vaadin	Sails.js	TurboGears
Laravel	OpenUI5	Yesod	Sinatra	Merb
Meteor	Tapestry	Compojure	Grails	Ramaze
Spring	Flight	Revel	Tornado	Ratpack
Express	CompoundJS	Martini	Phalcon	Aura
CodeIgniter	ZK	Mithril	Dojo	seaside
Symfony	Flatiron	beego	Struts	Zotonic
Ember.js	Noir	Ring	web.py	PureMVC
Flask	Catalyst	SproutCore	Bottle	Tipfy
JSF	Nitrogen	Mojolicious	Pyramid	Horde
CakePHP	Snap	SilverStripe Sapphire	Kohana	Cappuccino
Flex	Camping	Scalatra	Wicket	Swiz

Model Code Example

```
from django.db import models

class Question(models.Model):
  question_text = models.CharField(max_length=200)
  pub_date = models.DateTimeField('date published')

class Choice(models.Model):
  question = models.ForeignKey(Question,
                               on_delete=models.CASCADE)
  choice_text = models.CharField(max_length=200)
  votes = models.IntegerField(default=0)
```

Retrieving Objects

```
from polls.models import Question, Choice

Question.objects.all()
q = Question(question_text="What's new?",
             pub_date=timezone.now())
q.save()

q.id
>> 1  # automatically assigned by the DBMS
```

Issues with Web Frameworks

- How are objects stored?
  - Physical design problem
- How to debug?
- What if object layout needs to be changed?
- Generated queries are inefficient
  - The “N+1” problem

Recall: BCNF Decomposition

$R(A_1, ..., A_n, B_1, ..., B_m, C_1, ..., C_p)$

$R_1(A_1, ..., A_n, B_1, ..., B_m)$

$R_2(A_1, ..., A_n, C_1, ..., C_p)$

R_1 = projection of R on $A_1, ..., A_n, B_1, ..., B_m$
R_2 = projection of R on $A_1, ..., A_n, C_1, ..., C_p$

Theorem If $A_1, ..., A_n \rightarrow B_1, ..., B_m$
Then the decomposition is lossless

Note: don't necessarily need $A_1, ..., A_n \rightarrow C_1, ..., C_p$

Example

Patient

pno	name	zip
1	p1	98125
2	p2	98112
3	p1	98143

PatientOf

pno	dno	since
1	2	2000
1	3	2003
2	1	2002
3	1	1985

How to reconstruct a Patient object?

ORM: Use nested selects!

Integrating Queries into Languages

- Make query constructs first-class citizens in the programming language itself
- Examples: Microsoft LINQ

```
var numbers = DB.Tables["Numbers"].AsEnumerable();
var numsPlusOne = numbers.Select(n => n.Field<int>(0) + 1);
foreach (var i in numsPlusOne) {
    Log.WriteLine(i);
}
```

- Code is compiled by the C# compiler, which understands query operations

Conclusion

- Various ways to write DB applications
  - CLI
  - Drivers
  - Frameworks
  - Query-integrated languages