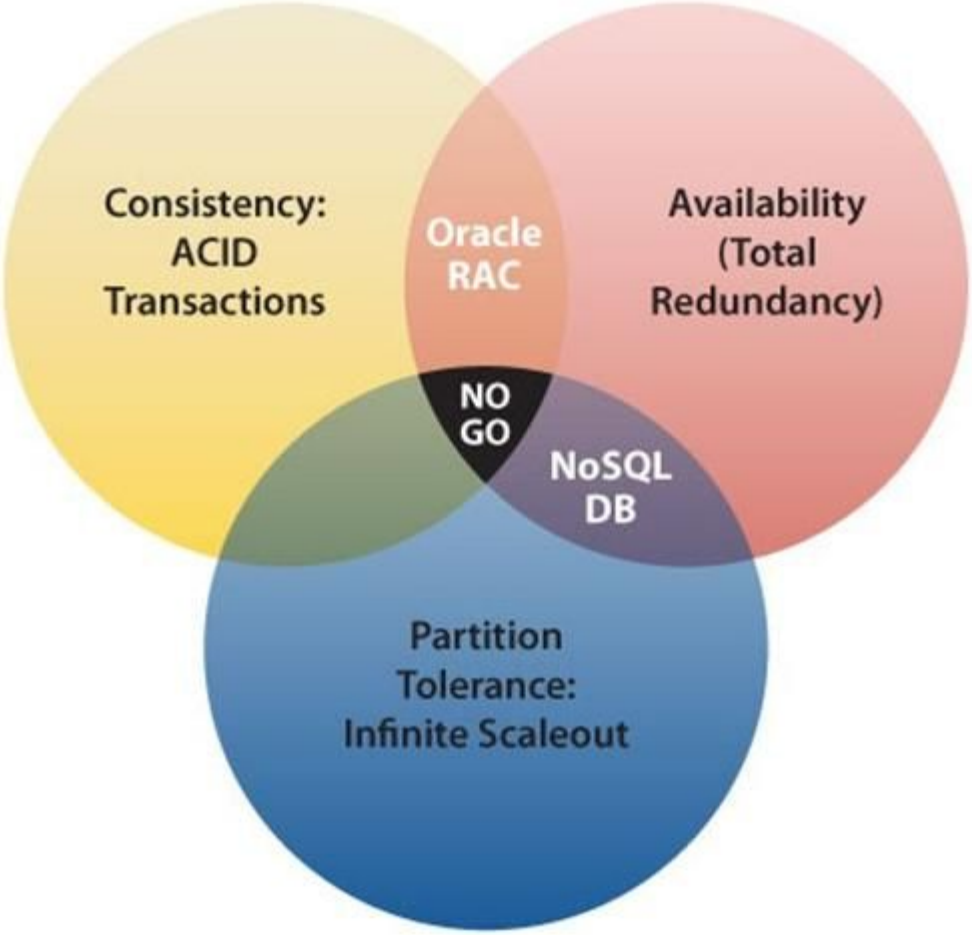
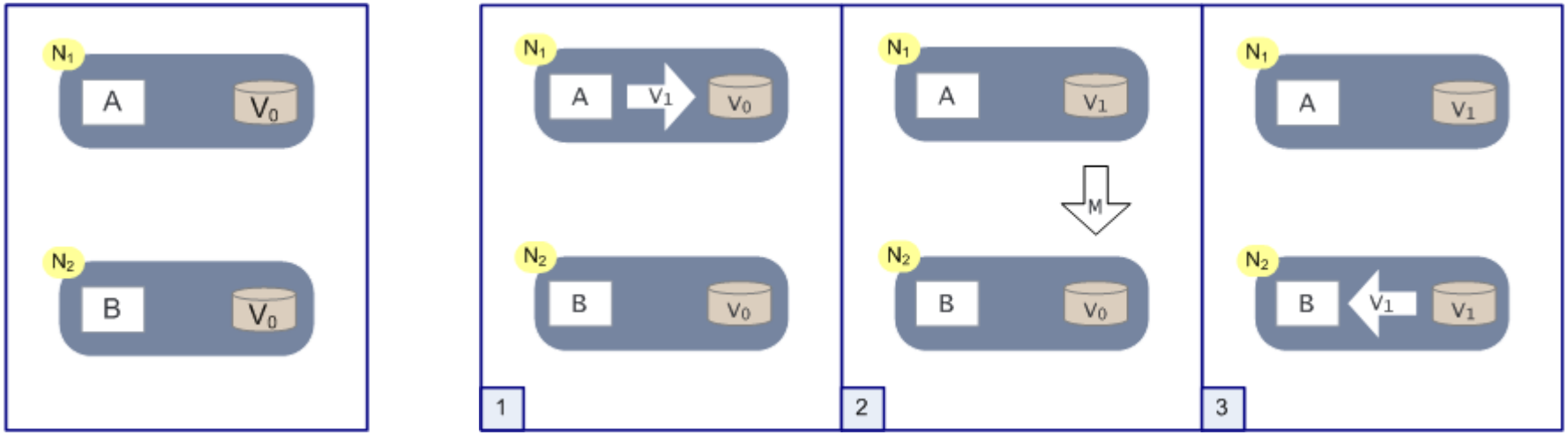
Lecture 34

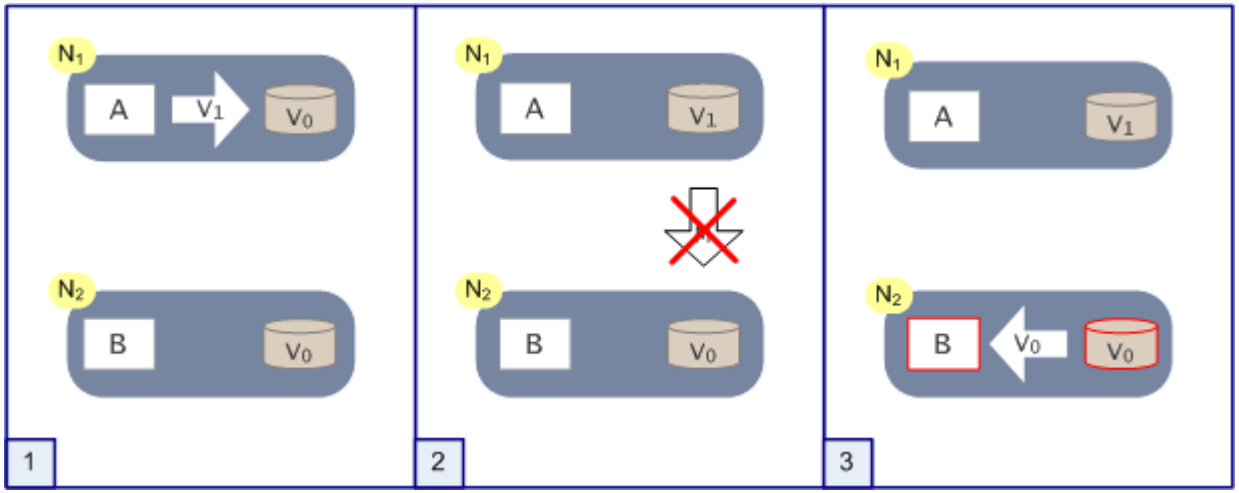
- Administration
- TA's
  - Nodir Kodirov
  - Imran Ahmed

CAP


Consistency:
ACID
Transactions
Oracle
RAC
Availability
(Total
Redundancy)
NO
GO
NoSQL
DB
Partition
Tolerance:
Infinite Scaleout

Simple Example





Choices (http://www.julianbrowne.com/article/viewer/brewers-cap-theorem)

- Drop Partition Tolerance
  - If you want to run without partitions you have to stop them happening. One way to do this is to put everything (related to that transaction) on one machine, or in one atomically-failing unit like a rack. It's not 100% guaranteed because you can still have partial failures, but you're less likely to get partition-like side-effects. There are, of course, significant scaling limits to this.
- **Drop Availability**
  - This is the flip side of the drop-partition-tolerance coin. On encountering a partition event, affected services simply wait until data is consistent and therefore remain unavailable during that time. Controlling this could get fairly complex over many nodes, with re-available nodes needing logic to handle coming back online gracefully.

Choices

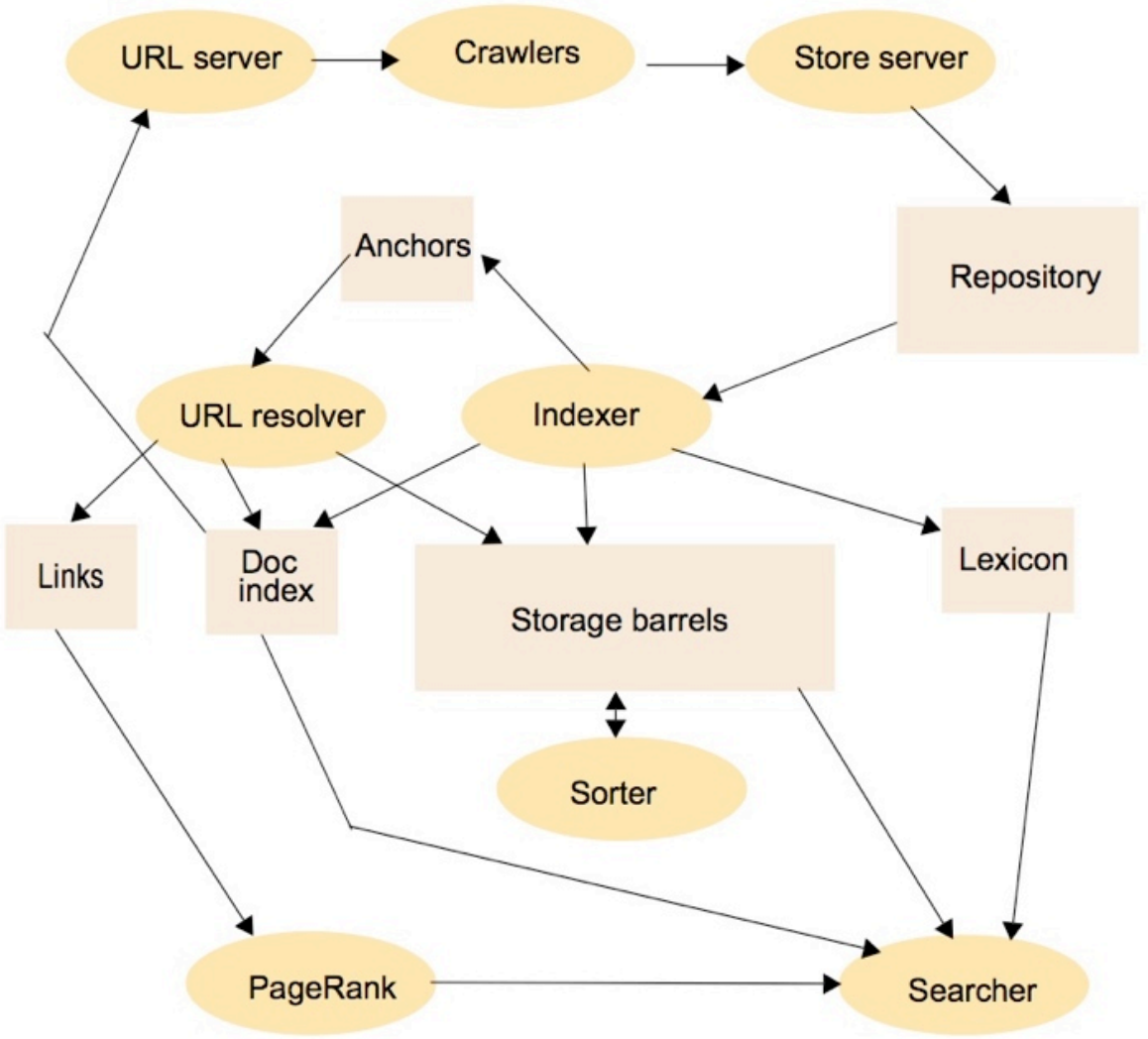
Drop Consistency

- Or, as Werner Vogels puts it, accept that things will become "Eventually Consistent". Vogels' article is well worth a read. He goes into a lot more detail on operational specifics than I do here. Lots of inconsistencies don't actually require as much work as you'd think (meaning continuous consistency is probably not something we need anyway). In my book order example if two orders *are* received for the one book that's in stock, the second just becomes a back-order. As long as the customer is told of this (and remember this is a rare case) everybody's probably happy.

The BASE Jump

- The notion of accepting eventual consistency is supported via an architectural approach known as BASE (**B**asically **A**vailable, **S**oft-state, **E**ventually consistent). BASE, as its name indicates, is the logical opposite of ACID, though it would be quite wrong to imply that any architecture should (or could) be based wholly on one or the other.

Outline architecture of the original Google search engine [Brin and Page 1998]

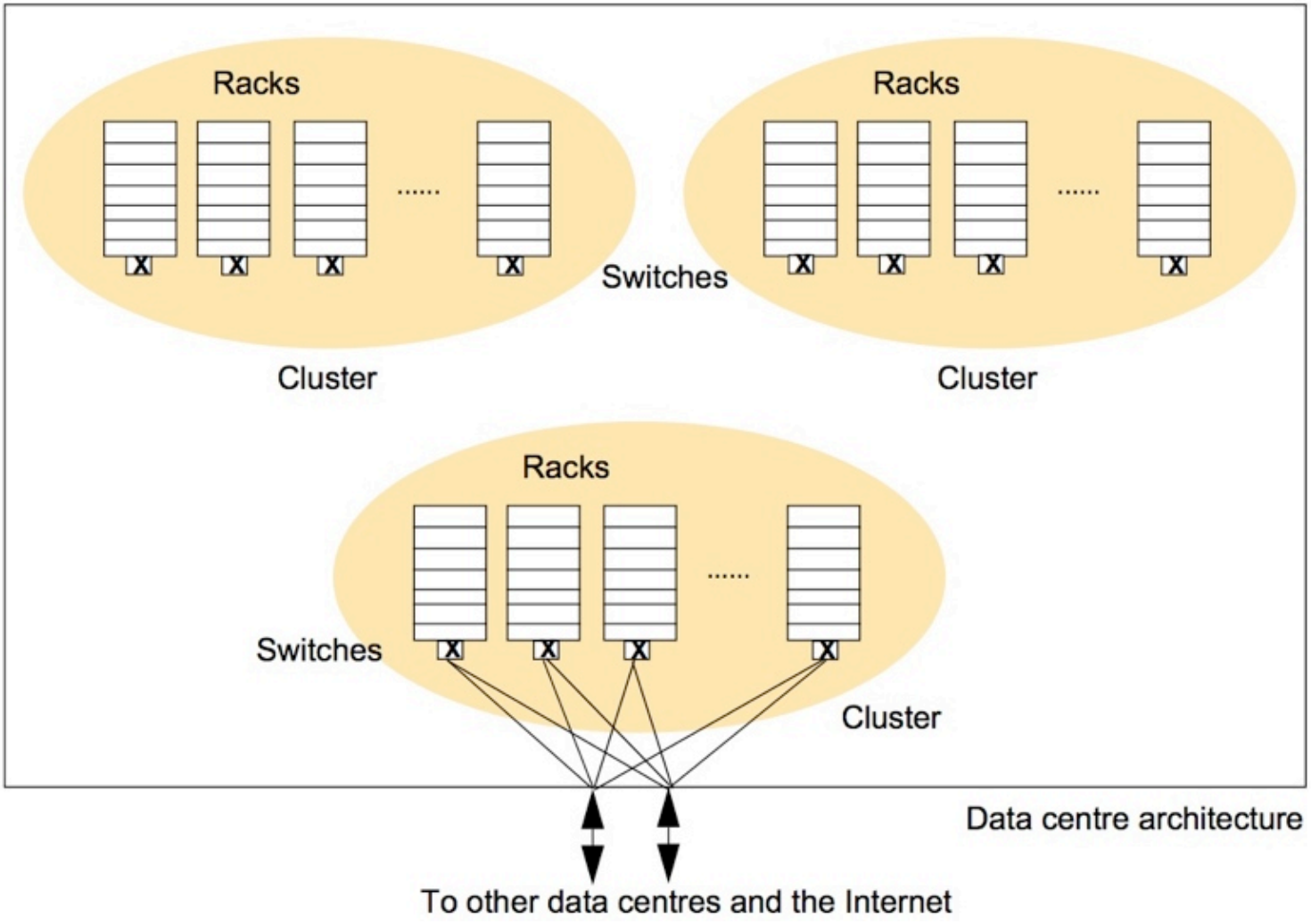


Example Google applications

Application	*Description*
Gmail	Mail system with messages hosted by Google but desktop-like message management.
Google Docs	Web-based office suite supporting shared editing of documents held on Google servers.
Google Sites	Wiki-like web sites with shared editing facilities.
Google Talk	Supports instant text messaging and Voice over IP.
Google Calendar	Web-based calendar with all data hosted on Google servers.
Google Wave	Collaboration tool integrating email, instant messaging, wikis and social networks.
Google News	Fully automated news aggregator site.
Google Maps	Scalable web-based world map including high-resolution imagery and unlimited user-generated overlays.
Google Earth	Scalable near-3D view of the globe with unlimited user-generated overlays.
Google App Engine	Google distributed infrastructure made available to outside parties as a service (platform as a service).

© Pearson Education 2012

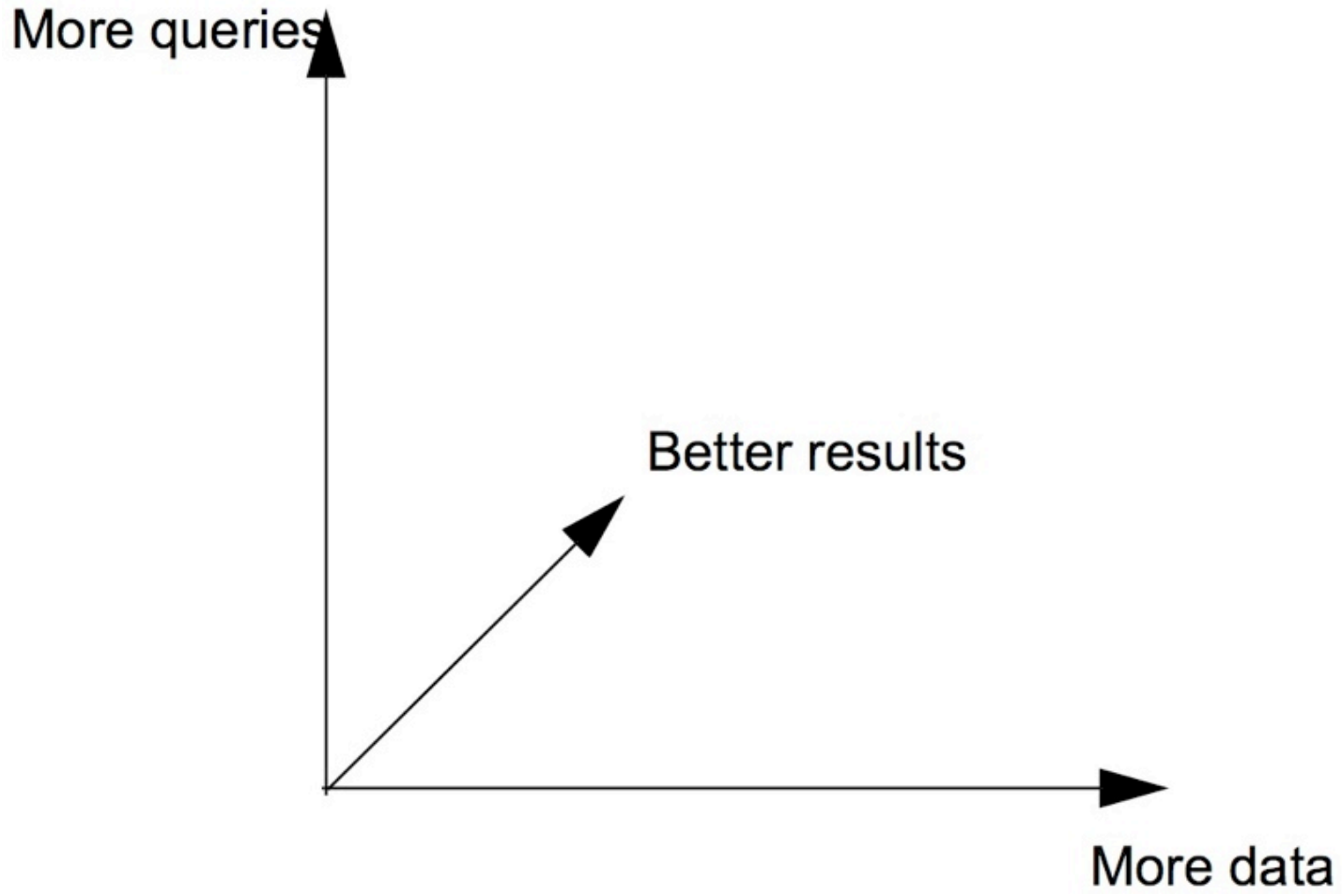
Organization of the Google physical infrastructure



(To avoid clutter the Ethernet connections are shown from only one of the clusters to the external links)

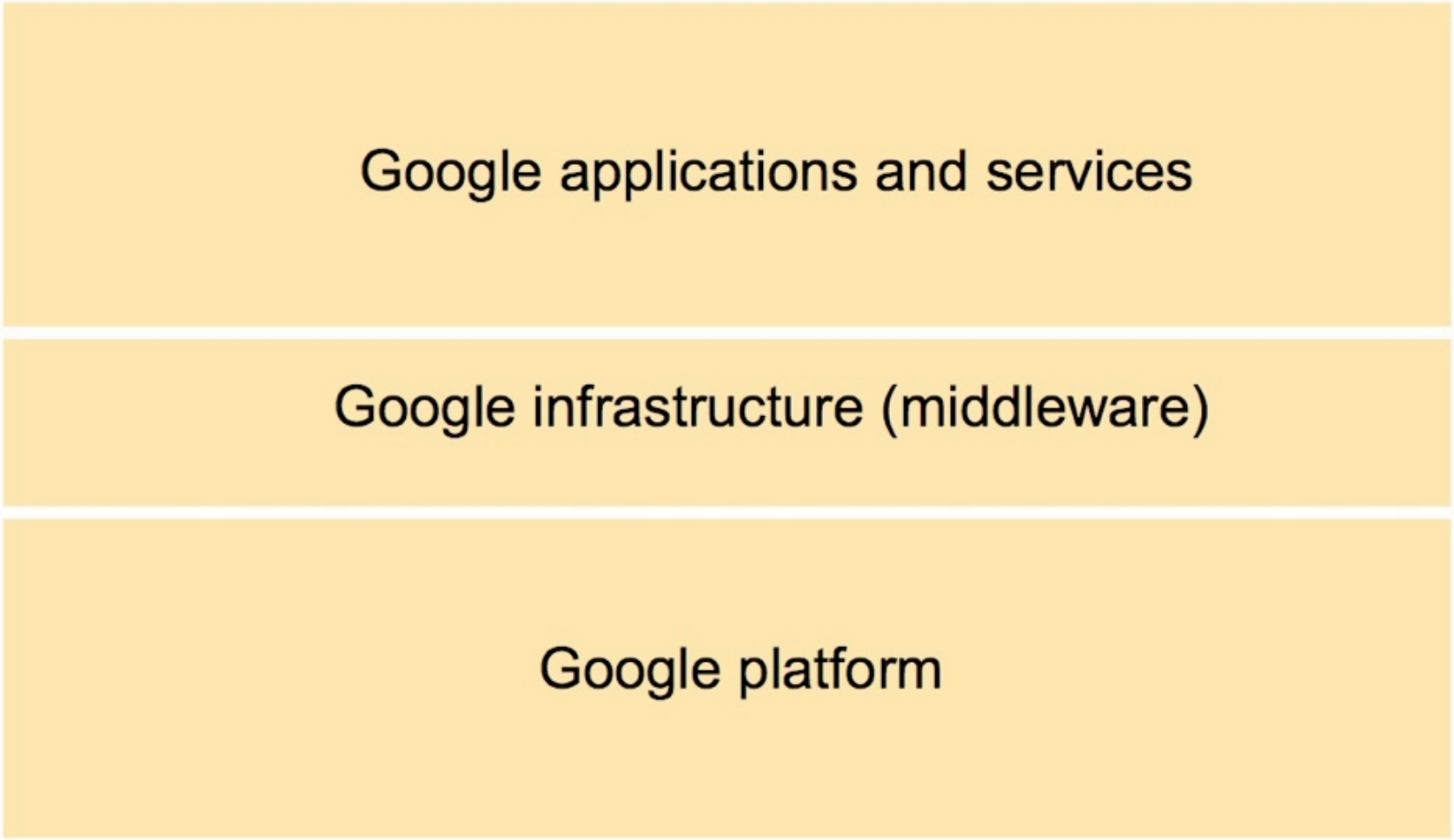
© Pearson Education 2012

The scalability problem in Google



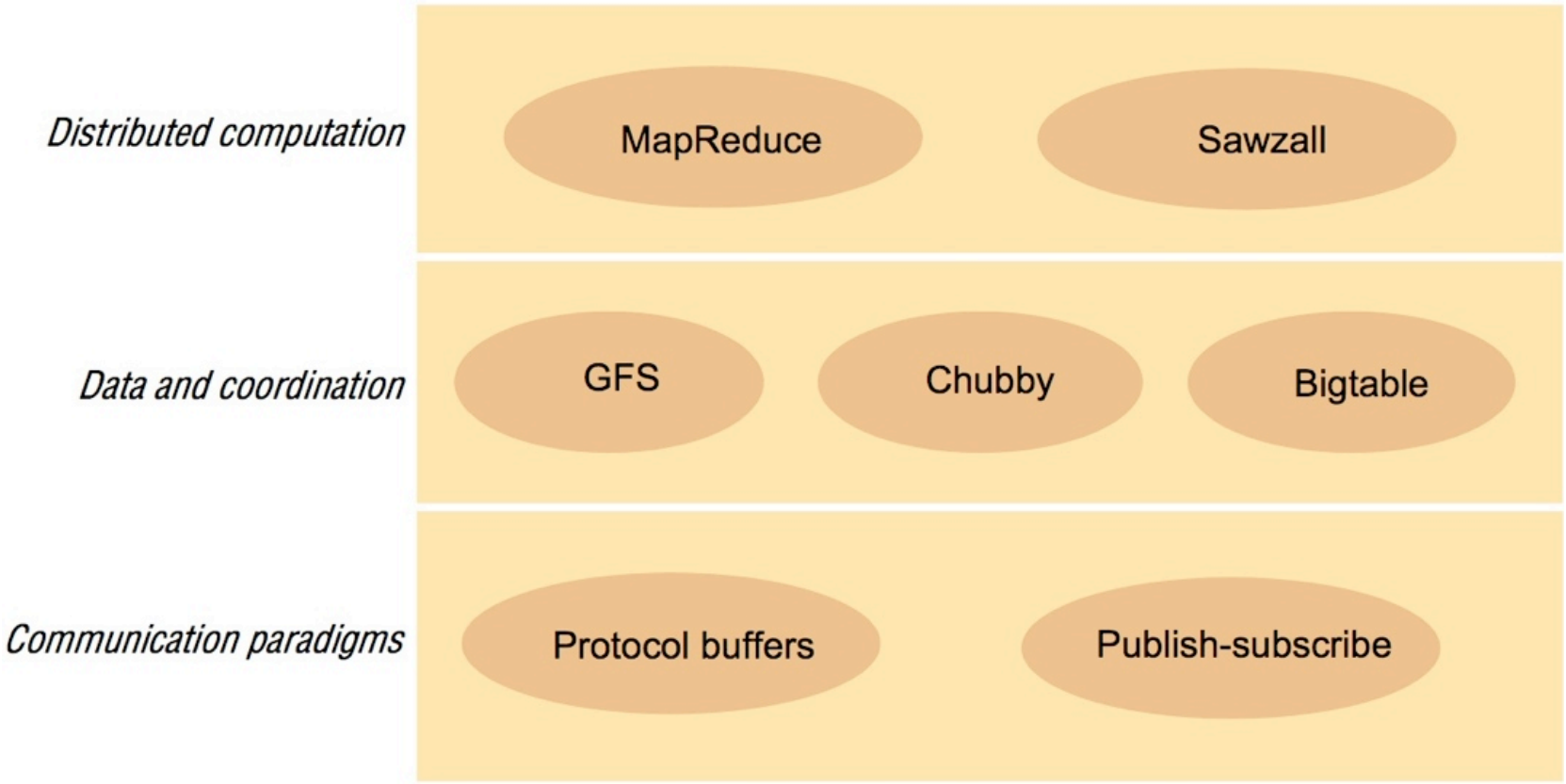
© Pearson Education 2012

The overall Google systems architecture



© Pearson Education 2012

Google infrastructure



© Pearson Education 2012

Protocol buffers example

```
message Book {
    required string title = 1;
    repeated string author = 2;
    enum Status {
        IN_PRESS = 0;
        PUBLISHED = 1;
        OUT_OF_PRINT = 2;
    }
    message BookStats {
        required int32 sales = 1;
        optional int32 citations = 2;
        optional Status bookstatus = 3 [default = PUBLISHED];
    }
    optional BookStats statistics = 3;
    repeated string keyword = 4;
}
```


© Pearson Education 2012

Summary of design choices related to communication paradigms

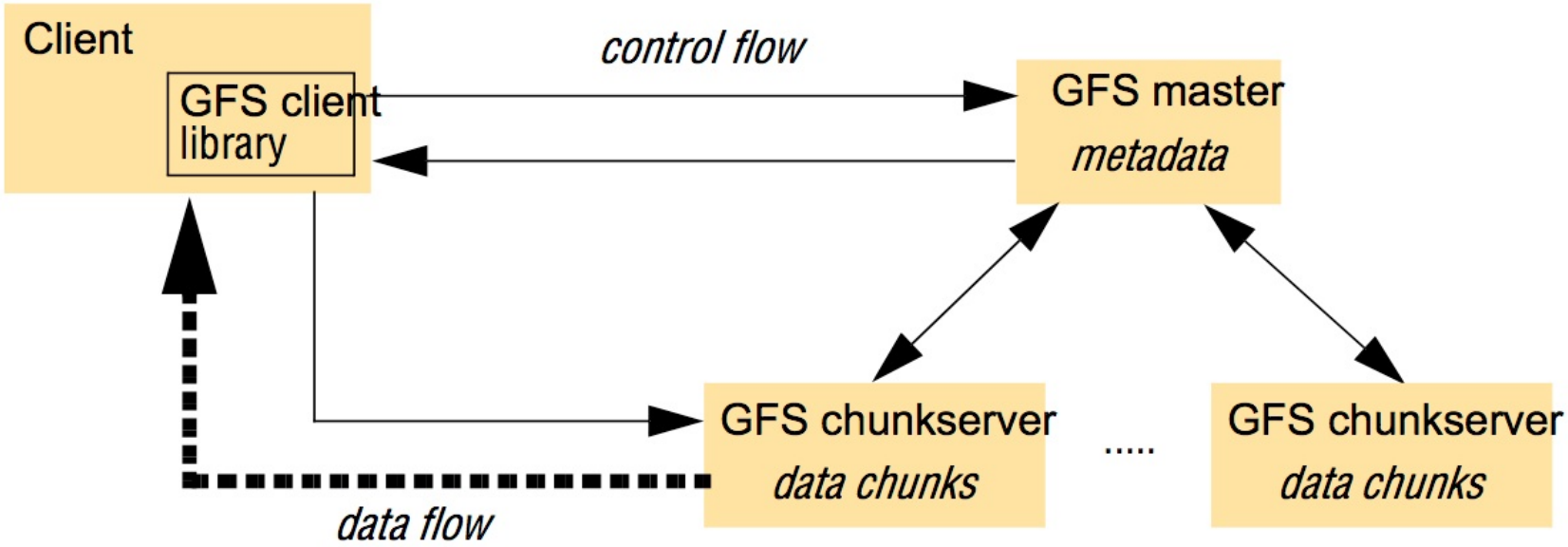
Element	*Design choice*	*Rationale*	*Trade-offs*
Protocol buffers	The use of a language for specifying data formats	Flexible in that the same language can be used for serializing data for storage or communication	-
	Simplicity of the language	Efficient implementation	Lack of expressiveness when compared, for example, with XML
	Support for a style of RPC (taking a single message as a parameter and returning a single message as result)	More efficient, extensible and supports service evolution	Lack of expressiveness when compared with other RPC or RMI packages
	Protocol-agnostic design	Different RPC implementations can be used	No common semantics for RPC exchanges

Summary of design choices related to communication paradigms

Publish-subscribe	Topic-based approach	Supports efficient implementation	Less expressive than content-based approaches (mitigated by the additional filtering capabilities)
	Real-time and reliability guarantees	Supports maintenance of consistent views in a timely manner	Additional algorithmic support required with associated overhead

© Pearson Education 2012

Overall architecture of GFS



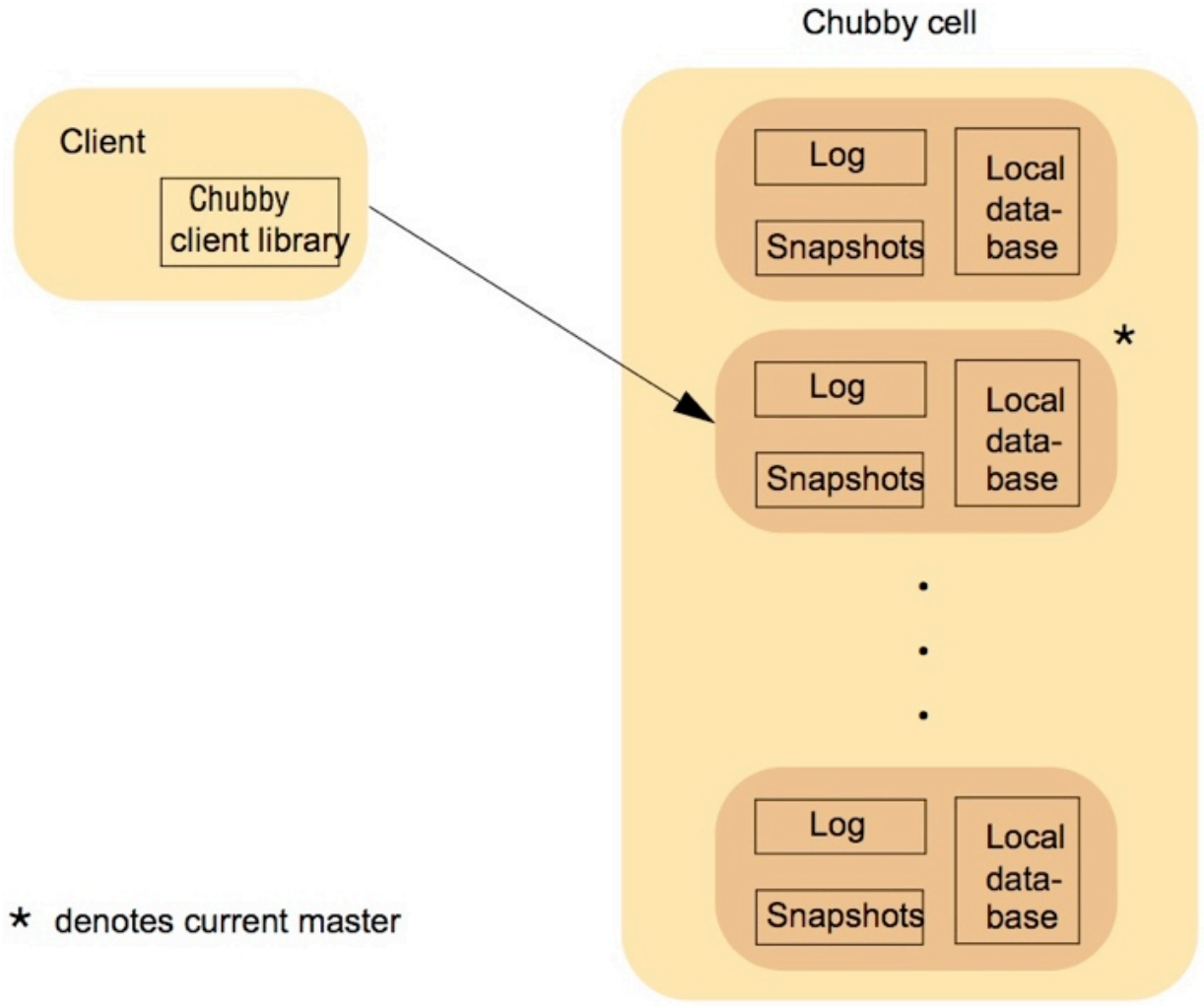
© Pearson Education 2012

Chubby API

Role	*Operation*	*Effect*
General	*Open*	Opens a given named file or directory and returns a handle
	Close	Closes the file associated with the handle
	Delete	Deletes the file or directory
File	*GetContentsAndStat*	Returns (atomically) the whole file contents and metadata associated with the file
	GetStat	Returns just the metadata
	ReadDir	Returns the contents of a directory – that is, the names and metadata of any children
	SetContents	Writes the whole contents of a file (atomically)
	SetACL	Writes new access control list information
Lock	*Acquire*	Acquires a lock on a file
	TryAquire	Tries to acquire a lock on a file
	Release	Releases a lock

© Pearson Education 2012

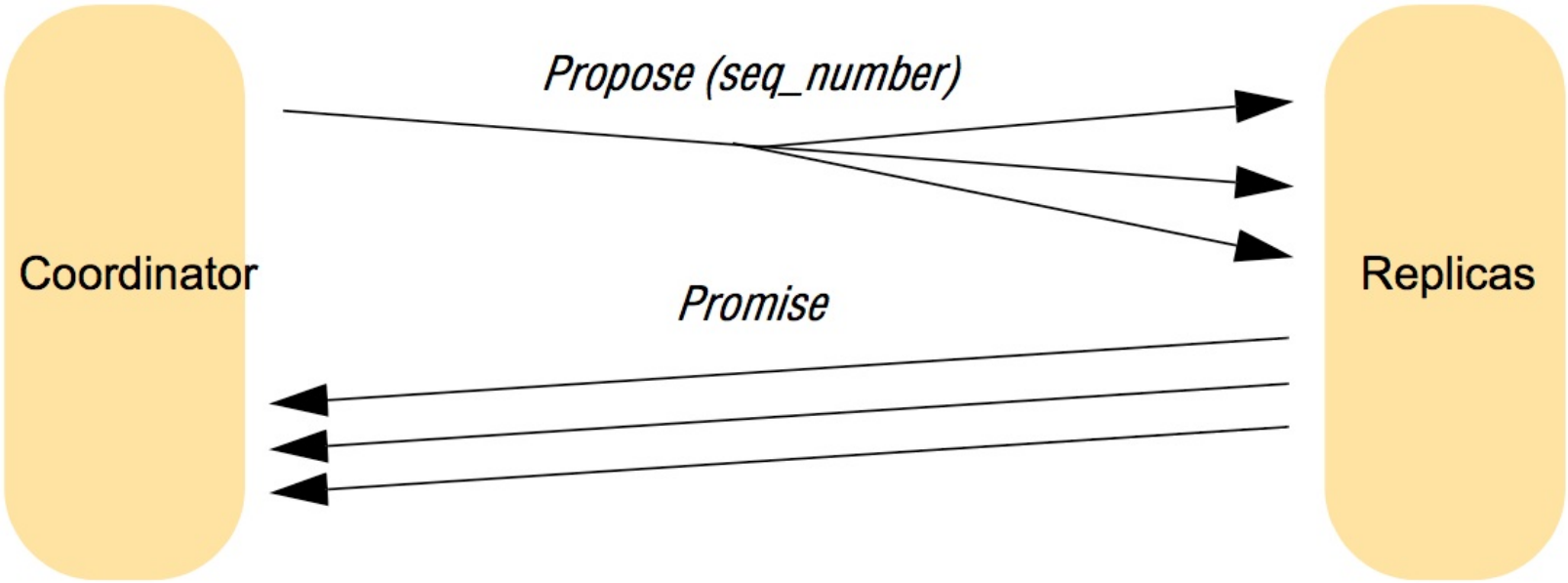
Overall architecture of Chubby



© Pearson Education 2012

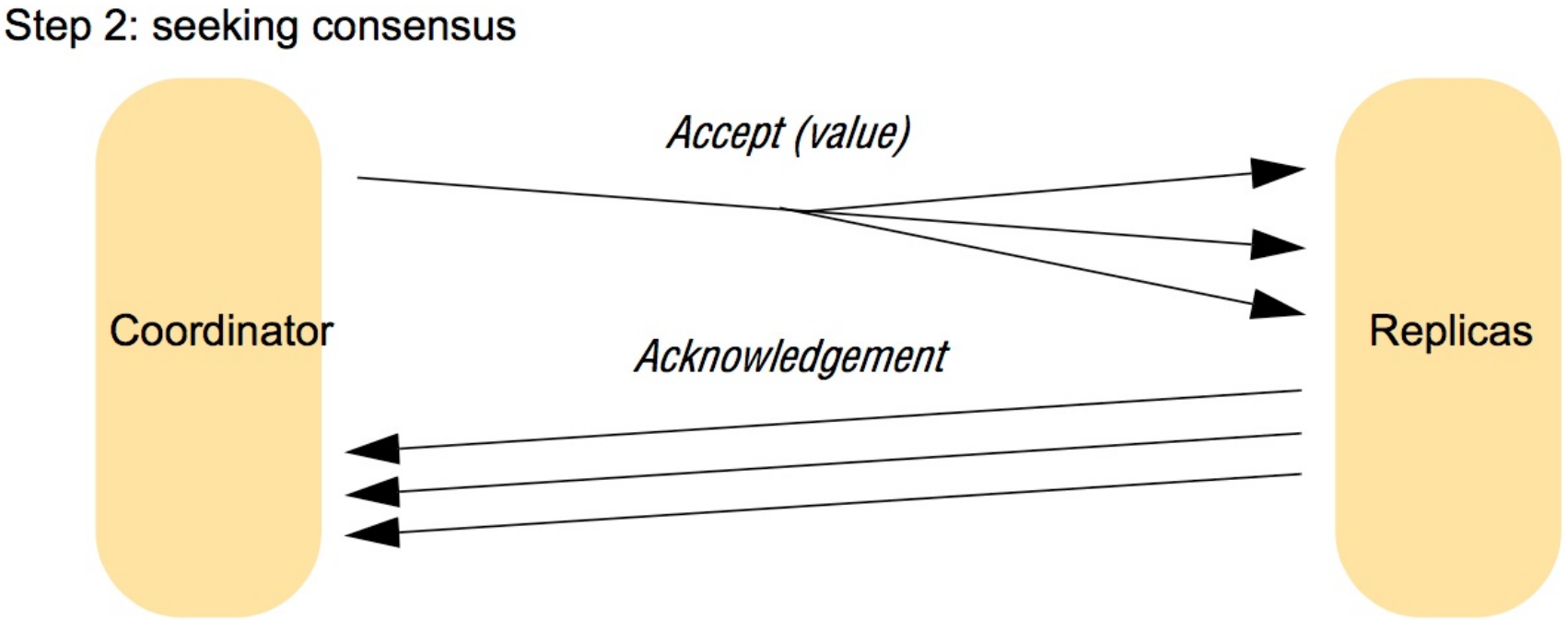
Message exchanges in Paxos (in absence of failures) - step 1

Step 1: electing a coordinator



© Pearson Education 2012

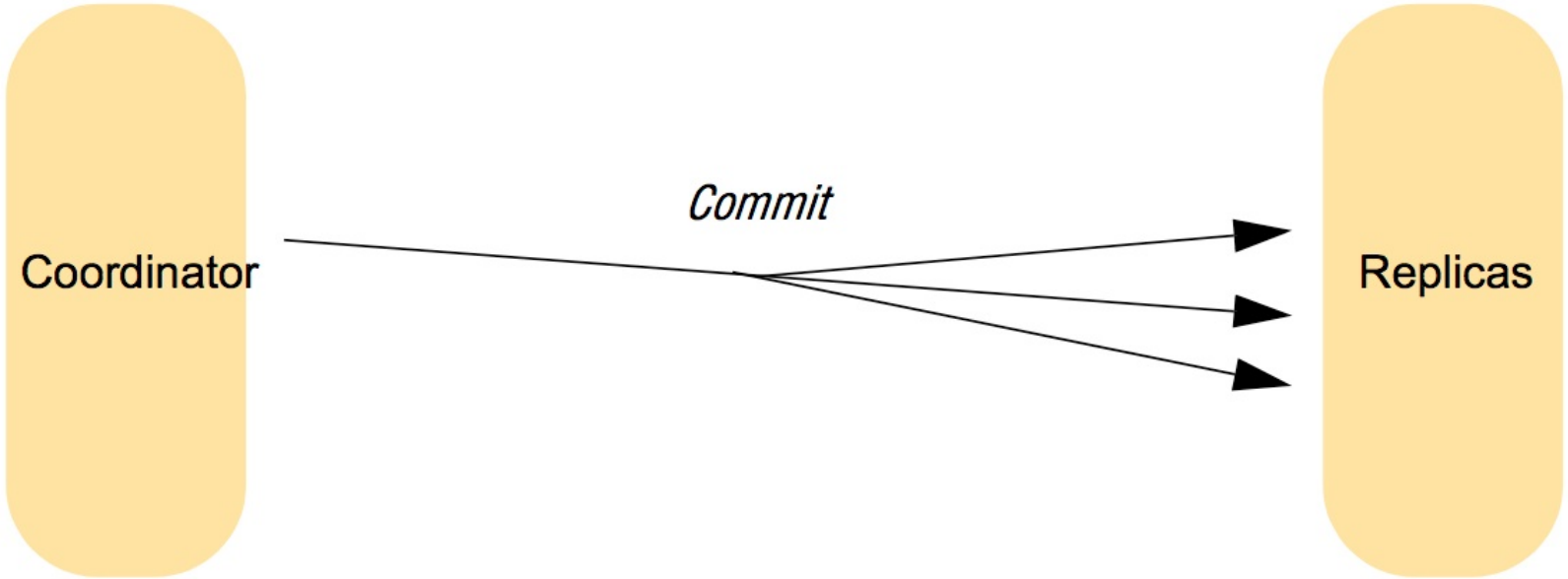
Message exchanges in Paxos (in absence of failures) - step 2



© Pearson Education 2012

Message exchanges in Paxos (in absence of failures) - step 3

Step 3: achieving consensus



© Pearson Education 2012