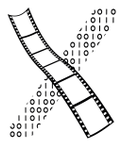


Media
Computing
Group

RWTH AACHEN
UNIVERSITY

Mobile Application Development
L12: Storage & Communication

Jonathan Diehl (Informatik 10)
Hendrik Thüs (Informatik 9)

Data Storage & Communication

- Serialization & File Management
- SQLite Database
- CoreData (iOS only)
- Remote (Web) Objects

Serialization & File Management

Object Serialization on iOS

- Simple Data Types: Property Lists
  - Reading: `[NSDictionary dictionaryWithContentsOfFile:path];`
  - Writing: `[dictionary writeToFile:path atomically:YES];`
  - Advanced operations: NSPropertyListSerialization
- Custom Data: NSCoding
  - `[NSKeyedArchiver archiveRootObject:root toFile:path];`
  - `id object = [NSKeyedUnarchiver unarchiveObjectWithFile:path];`
  - Custom model classes must implement the NSCoding protocol

File Management on iOS

- NSFileHandle
  - access file, open read and write streams
- NSFileManager
  - list directories, copy/move files, etc.
- Standard Paths
  - `[[NSBundle mainBundle] pathForResource:@"MyList" ofType:@"plist"];`
  - `[UIImage imageNamed:@"myImage"];`
  - `NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);`

Serialization on Android: JSON

```
// A modifiable set of name/value mappings

[
    {"id":1,"name":"Alice","key":42},
    {"id":2,"name":"Bob","key":99},
    {"id":3,"name":"Eve","key":42}
]
```

Serialization on Android: JSON

```
JsonString = "[{\"id\":1,\"title\":\"objA\"},{\"id\":
2,\"name\":\"objB\"}]";

JSONArray objects = new JSONArray(JsonString);

for (int i = 0; i < objects.length(); i++) {
    id = objects.getJSONObject(i).getInt("id");
    title = objects.getJSONObject(i).getString("title")
       .toString();
    ...
}
```

File Management on Android

- Internal Files
  - FileOutputStream
  - FileInputStream
  - fileList()
- External Files
  - File root = Environment.*getExternalStorageDirectory();*
  - FileWriter
  - FileInputStream
  - root. listFiles()

SQLite

Database Fundamentals

- Relational Database
  - is organized in tables with fields and rows
    - fields have a fixed type
      - number, character, data...
- Standard Query Language (SQL)
  - standardized language to describe data access
  - specify table, fields, order, conditions, ...

SQLite Characteristics

- Relational Database in a single file
  - no server
- Simple, compact, fast, reliable
- Limitations
  - limited concurrency
  - single client
  - “small” data set

Working with SQLite on iOS

- C-Based API
  - http://www.sqlite.org/cintro.html

Working with SQLite on Android

- SQLiteOpenHelper
  - Creates tables
  - Upgrades tables
  - Handles name and version of DB

Working with SQLite on Android

```
// Insert

db = new MyDb(this);
String sql = "INSERT INTO… ";
db.getWritableDatabase().execSQL(sql);
```

```
// Select

String sql = "SELECT …";
Cursor result = db.getWritableDatabase()
  .rawQuery(sql, null);
String value;
while (result.moveToNext()) {
   value = result.getString(…);
}
```

Working with SQLite on Android

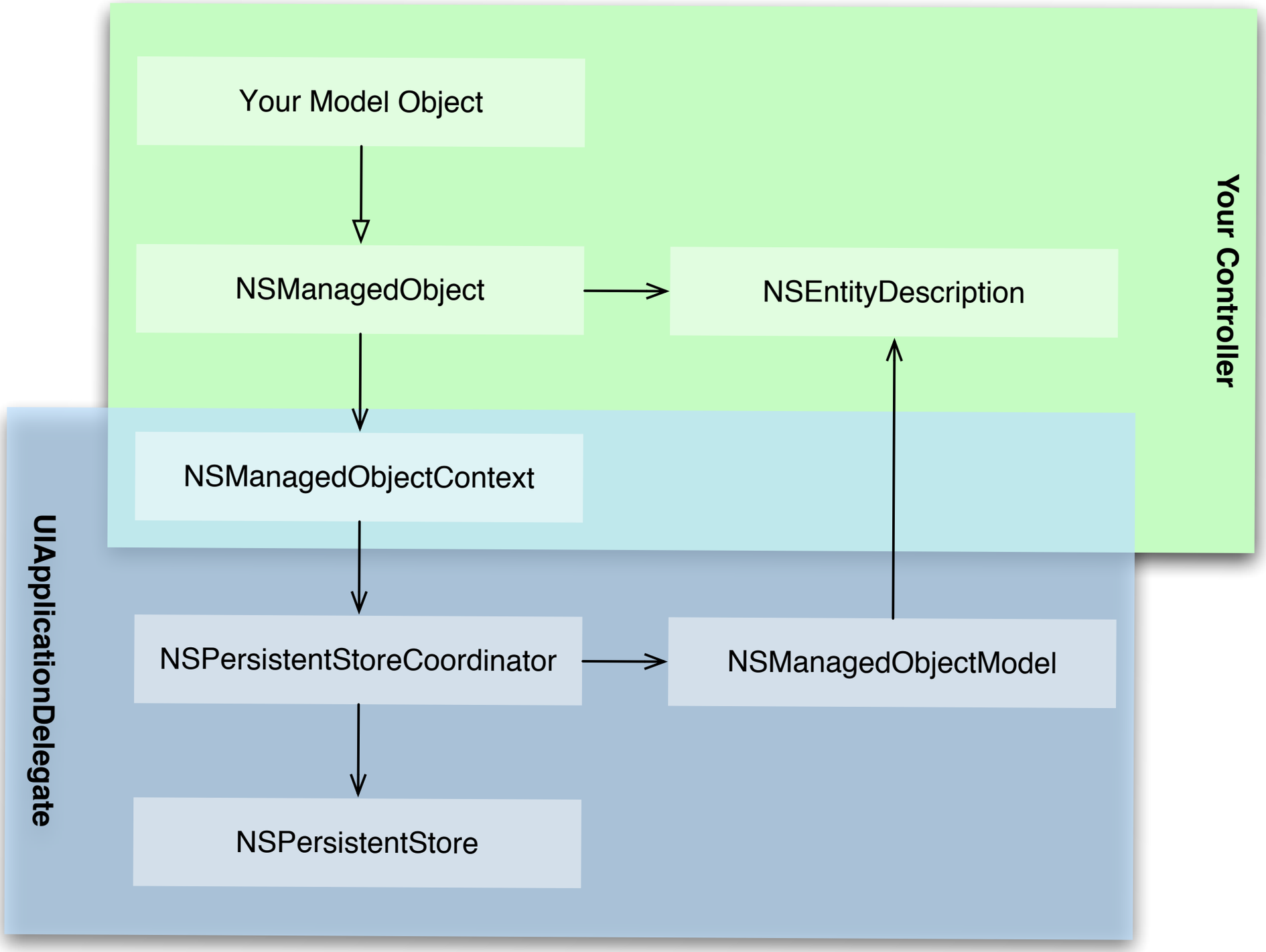
Demo

Core Data

Core Data

- High level relational data storage
- Automatic persistence
- High performance access
- Get undo/redo for free
- Available data stores
  - memory
  - binary file
  - SQLite

Core Data Stack



CoreDataDemo – CoreDataDemo.xcdatamodel

Run Stop Scheme Breakpoints Editor View Organizer

CoreDataDemo › CoreData… › CoreData… › CoreDataDemo.xcdatamodel › Attendee

ENTITIES
- Attendee
- Event

FETCH REQUESTS

CONFIGURATIONS
- Default

Attributes

Attribute	Type
name	String

Relationships

Relationship	Destination	Inverse
event	Event	attendee

Fetched Properties

Fetched Property	Predicate

Entity

Name: Attendee
Class: NSManagedObject
Abstract Entity
Parent Entity: No Parent Entity

User Info

Key	Value

Versioning

Hash Modifier:
Renaming ID:

Entity Sync

Synchronization: Disabled

File Template Library

Objective-C class – An Objective-C class with a header

UIViewController subclass – An Objective-C view controller subclass

Objective-C category – An Objective-C category

Outline Style Add Entity Add Attribute Editor Style

Custom Model Class

```
// Recipe.h
@interface Recipe : NSManagedObject {
}

@property(retain) NSString *title;
```

```
// Recipe.m
#import "Recipe.h"

@implementation Recipe

@dynamic title;

@end
```

Managing Managed Objects

- Create Object

```
[NSEntityDescription insertNewObjectForEntityForName:
  @"Person" inManagedObjectContext:context]
```

- Delete Object

```
[context deleteObject:managedObject];
```

- Update Object

```
person.name = @"Peter";
```

- Save Changes

```
NSError *error;
[context save:&error];
```

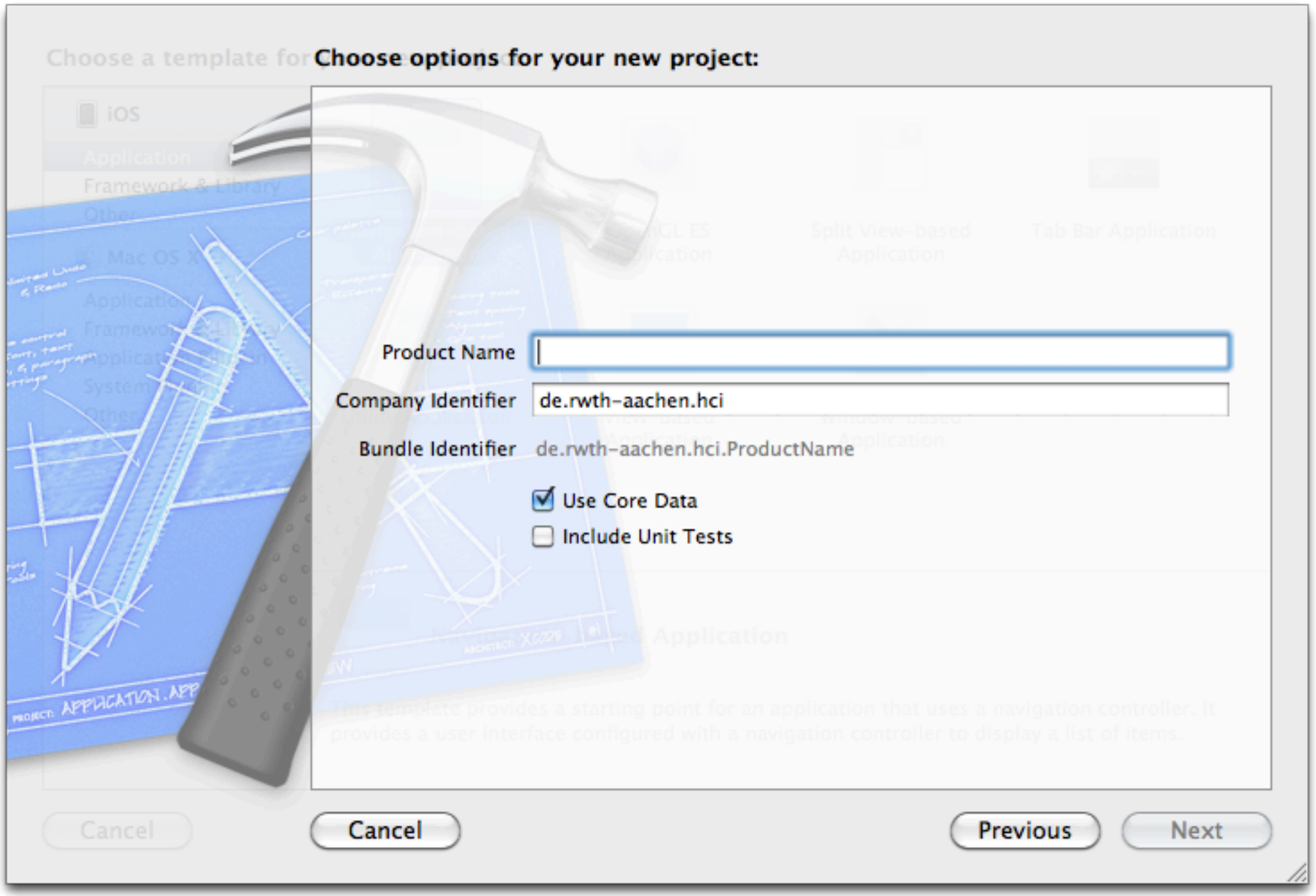
Fetching Managed Objects

- NSFetchRequest
  - describes a fetch (SQL query)
- NSFetchedResultsController
  - tailored to provide data for UITableViews
  - automatically tracks changes and updates the table

Fetch Data

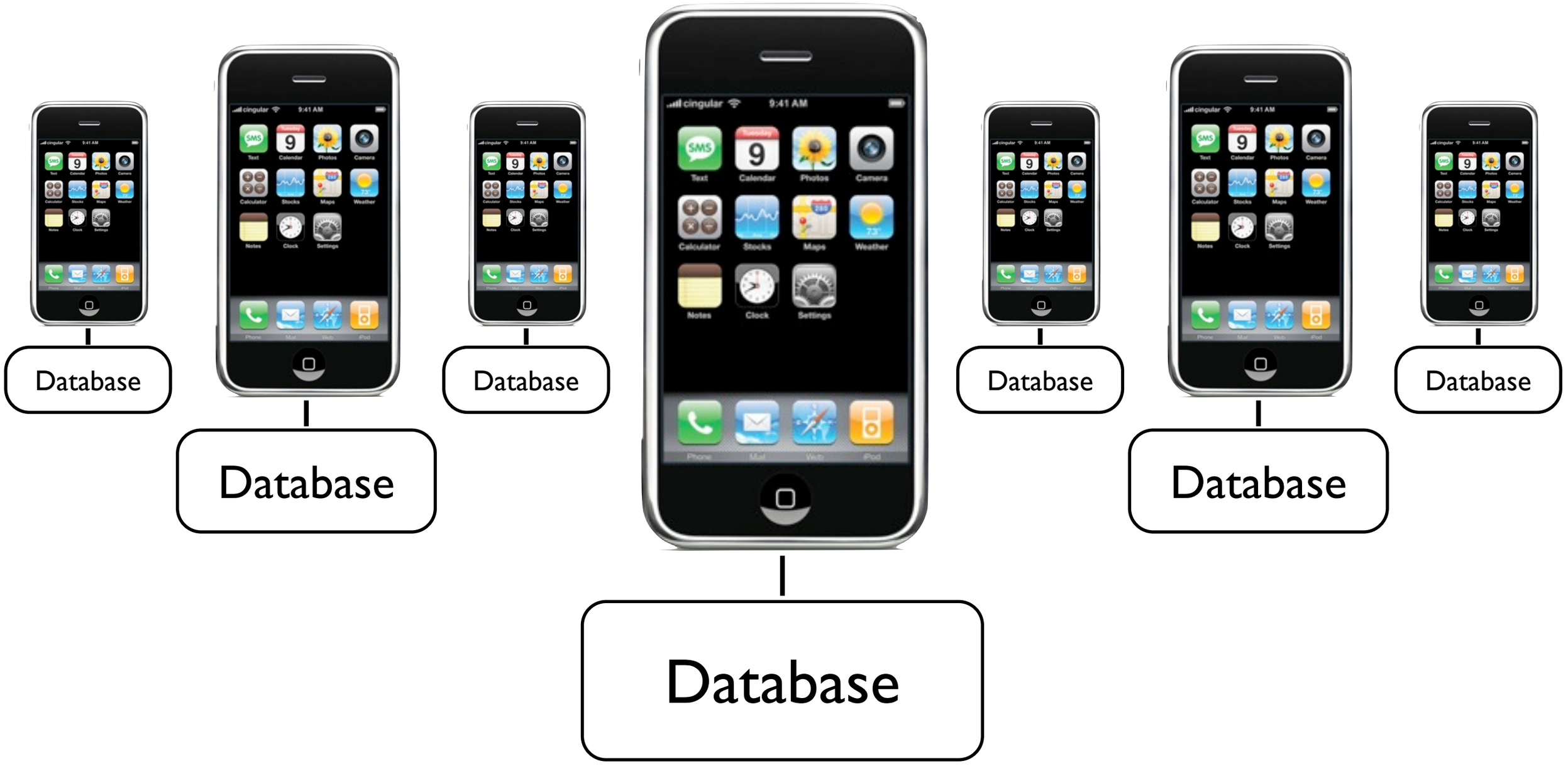
```
//Set up the fetch request
NSFetchRequest *request = [[NSFetchRequest alloc] init];
[request setEntity: [NSEntityDescription entityForName:@"Recipe"
  inManagedObjectContext:managedObjectContext]];

//Set up the sort descriptor
NSSortDescriptor *sort = [[[NSSortDescriptor alloc]
  initWithKey:@"creationDate" ascending:NO] autorelease];
[request setSortDescriptors: [NSArray arrayWithObject:sort]];

// Execute the request
NSError *error;
NSArray *results = [context executeFetchRequest:request
  error:&error];
if (!results) {
  // Handle the error.
}
```

Fetched Results Controller

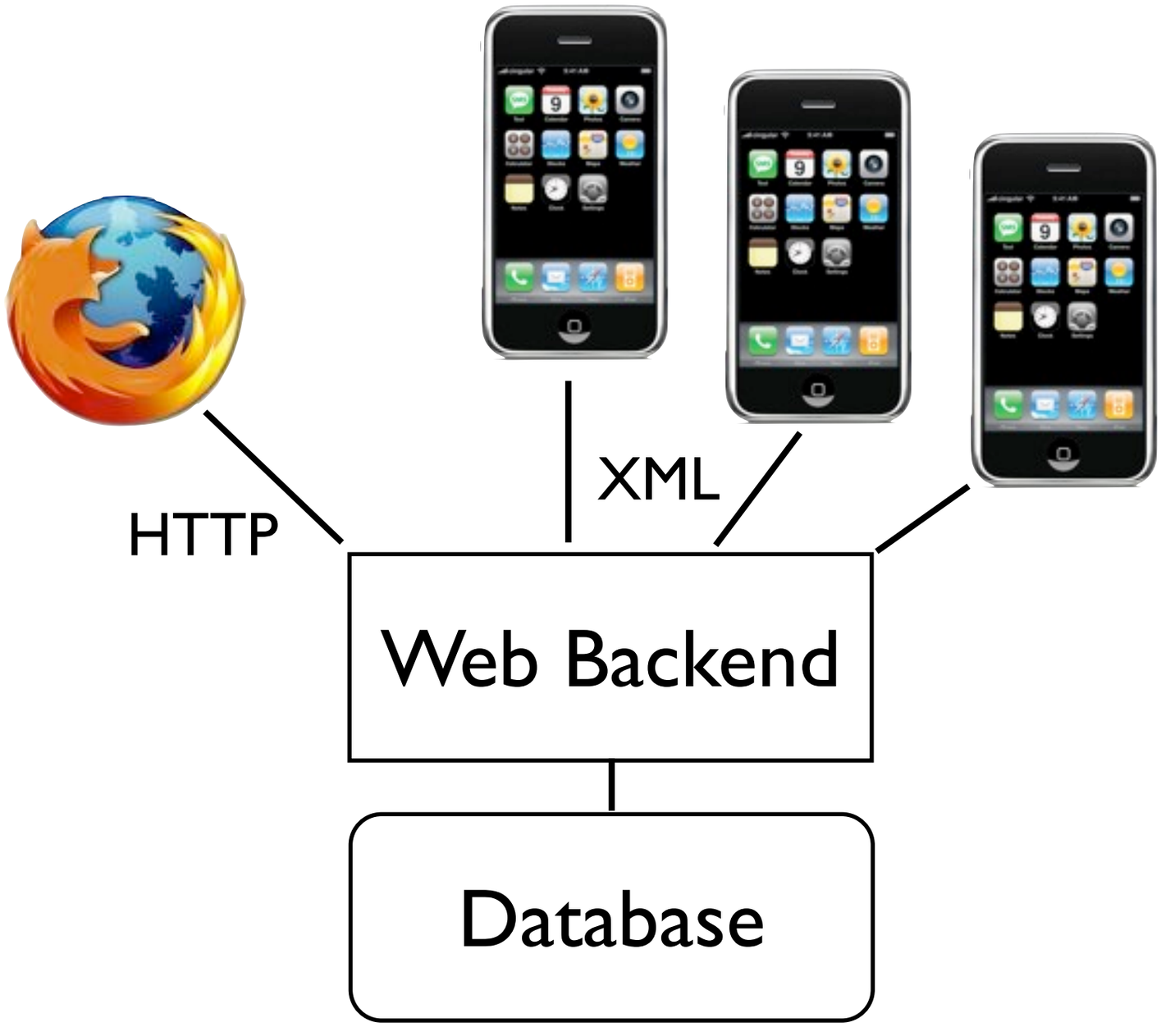


Remote (Web) Objects

Web-Backend

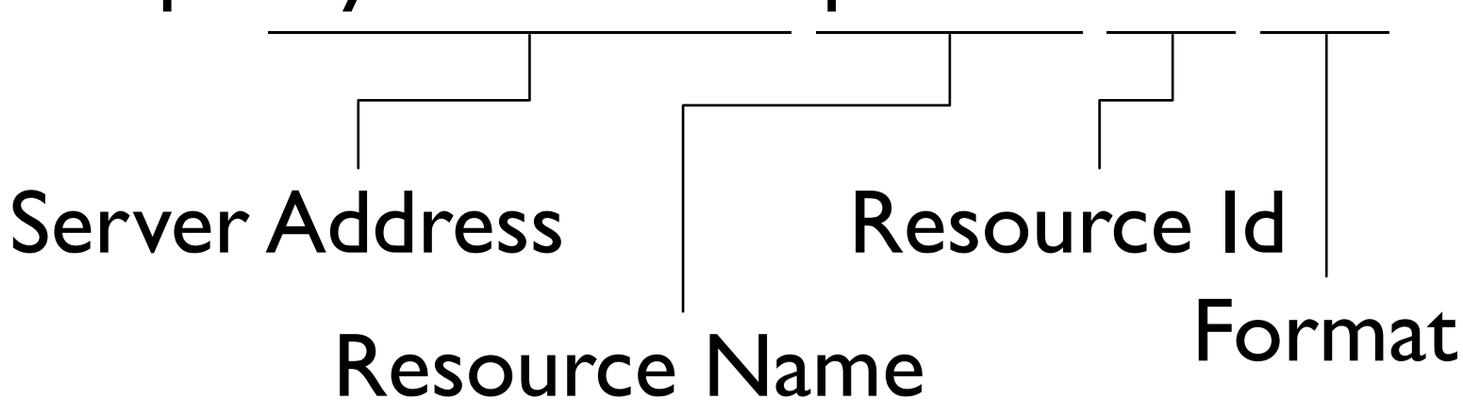

Database
Database
Database
Database
Database
Database
Database
Database

Web-Backend



Representational State Transfer

- Resource manipulation via HTTP operations
- URL describes resource



- Data encoded in XML, JSON, plist, ...
- HTTP operation defines REST operation

REST Operations

- GET: read resource (all by id)

```
GET http://server/people/1
```

- POST: update resource

```
POST http://server/people/1
person[name]=Peter&person[email]=...
```

- PUT: create resource

```
PUT http://server/people
person[name]=Peter&person[email]=...
```

- DELETE: delete resource

```
DELETE http://server/people/1
```

HTTP Requests on iOS

```
// create request
NSURL *url = [NSURL URLWithString:@"http://server/person/1"];
request = [[NSMutableURLRequest alloc] initWithURL:url];

// set the HTTP operation
[request setHTTPMethod:@"POST"];

// set the post data
NSString *bodyString = @"person[name]=Paul&person[age]=21";
[request setHTTPBody:[bodyString
  dataUsingEncoding:NSUTF8StringEncoding]];

// fire the request
connection = [NSURLConnection
  connectionWithRequest:request delegate:self];
[connection start];

// clean up
[request release];
```

HTTP Requests on iOS (cont.)

```
   // received a chunk of data
- (void)connection:(NSURLConnection *)connection
     didReceiveData:(NSData *)data
{
   [mutableData appendData:data];
}

// finished loading url
- (void)connectionDidFinishLoading:(NSURLConnection *)connection
{
   // interpret data (e.g., load plist data
   NSPropertyListFormat format;
   NSError *error;
   id response = [NSPropertyListSerialization
     propertyListWithData:mutableData options:0 format:&format
     error:&error];
   if(!response) {
      // handle error
   }
}
```

HTTP-GET-Requests on Android

```
HttpClient httpclient = new DefaultHttpClient();
HttpGet httpget = new HttpGet("http://www.spiegel.de");
HttpResponse response;

try {
    response = httpclient.execute(httpget);
    …
}
…
```

HTTP-POST-Requests on Android

```
HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost = new HttpPost("http://www.spiegel.de");
HttpResponse response;

try {
    List<NameValuePair> nameValuePairs = new
        ArrayList<NameValuePair>(2);
    nameValuePairs.add(
        new BasicNameValuePair("id", "12345"));

    response = httpclient.execute(httppost);
    …
}
…
```

AsyncTask

- Background tasks/operations
- 3 generic types
  - Params
  - Progress
  - Result
- 4 steps
  - onPreExecute
  - doInBackground
  - onProgressUpdate
  - onPostExecute

Asynchronous Requests

Demo