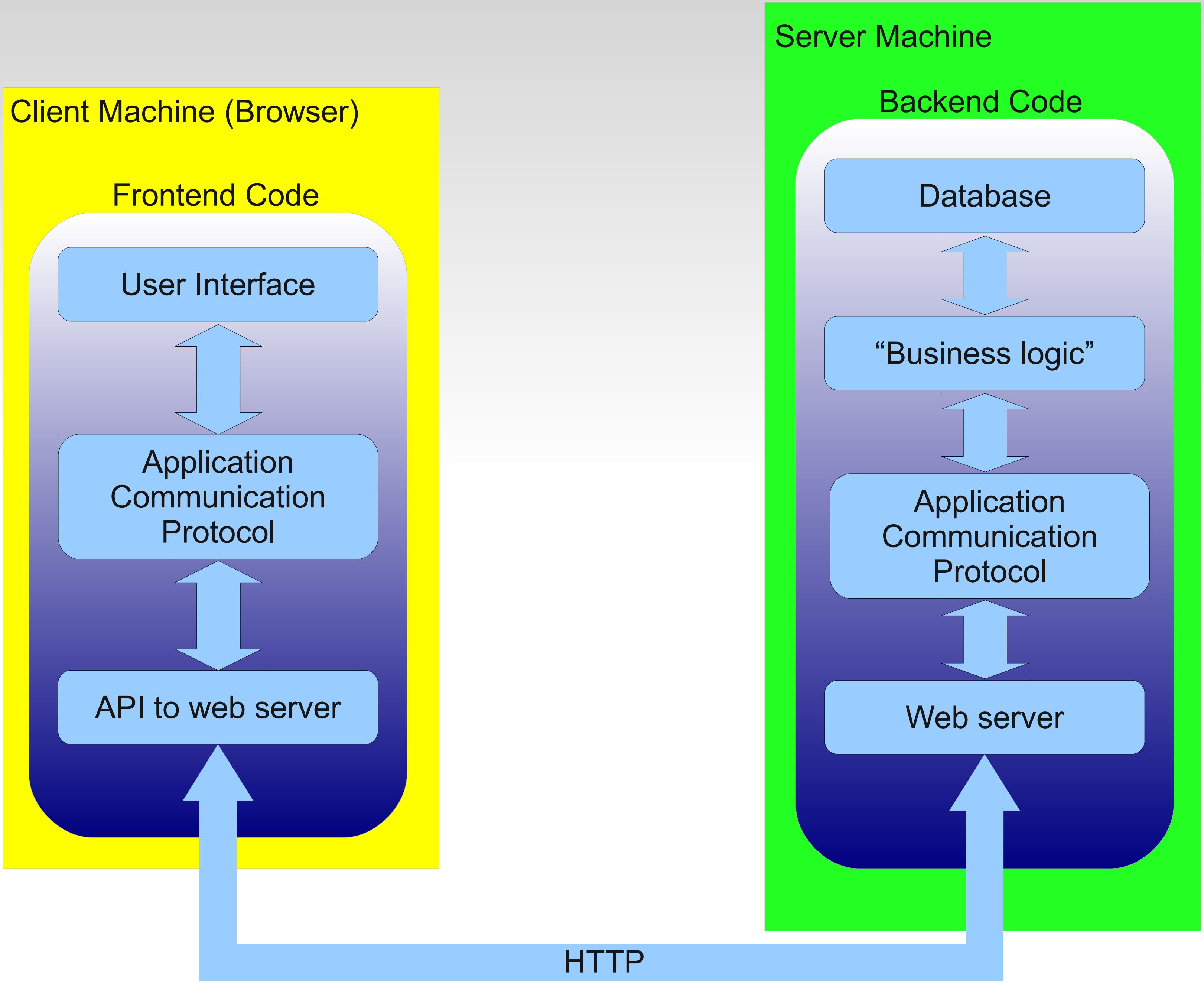
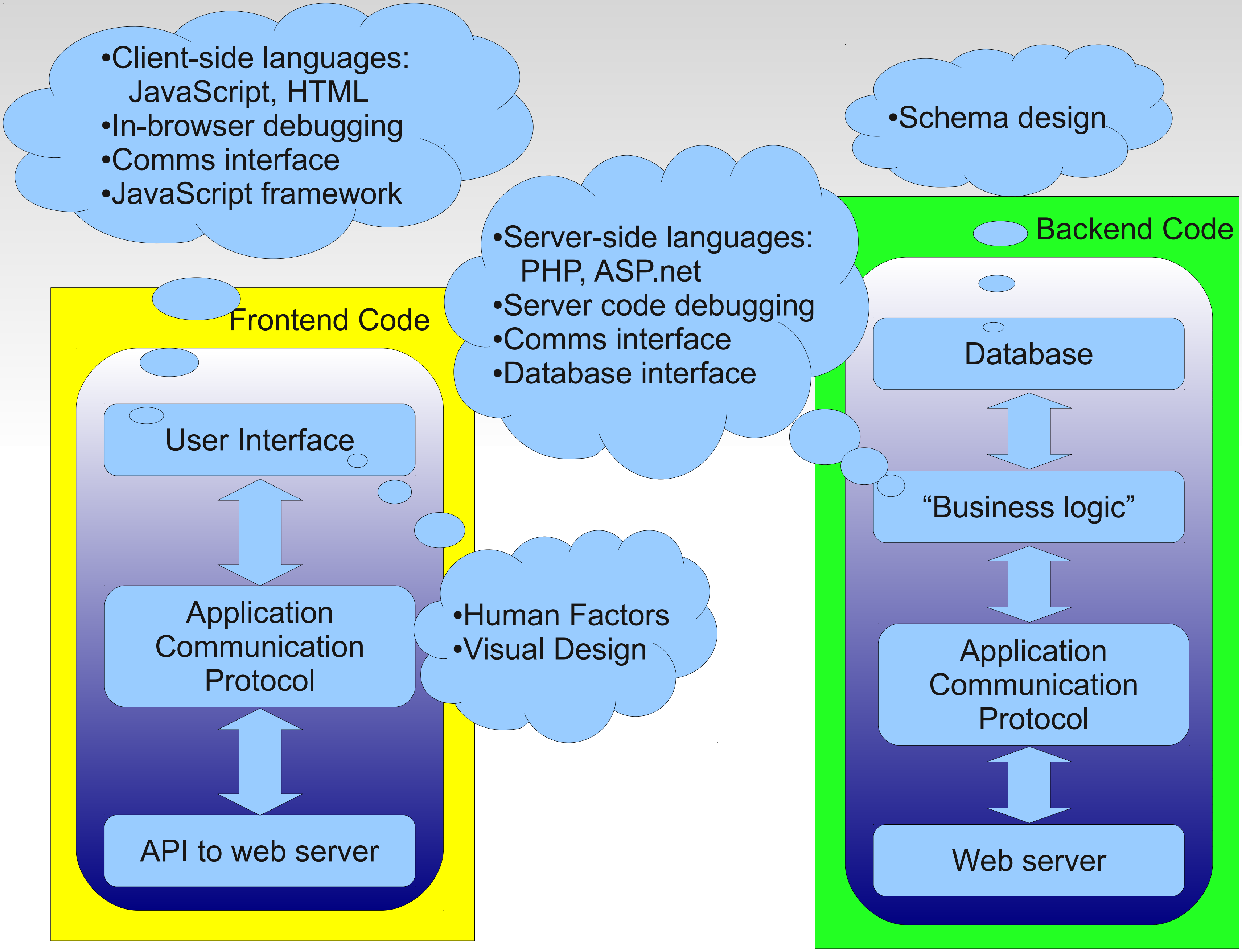
LIBERATED: A fully in-browser client and server web application debug and test environment

Derrell Lipman
University of Massachusetts Lowell

Overview of the Client/Server Environment



Many Skill Sets Required



- Client-side languages: JavaScript, HTML
- In-browser debugging
- Comms interface
- JavaScript framework

Frontend Code

User Interface

Application Communication Protocol

API to web server

- Server-side languages: PHP, ASP.net
- Server code debugging
- Comms interface
- Database interface

- Human Factors
- Visual Design

- Schema design

Backend Code

Database

“Business logic”

Application Communication Protocol

Web server

Debugging Difficulties

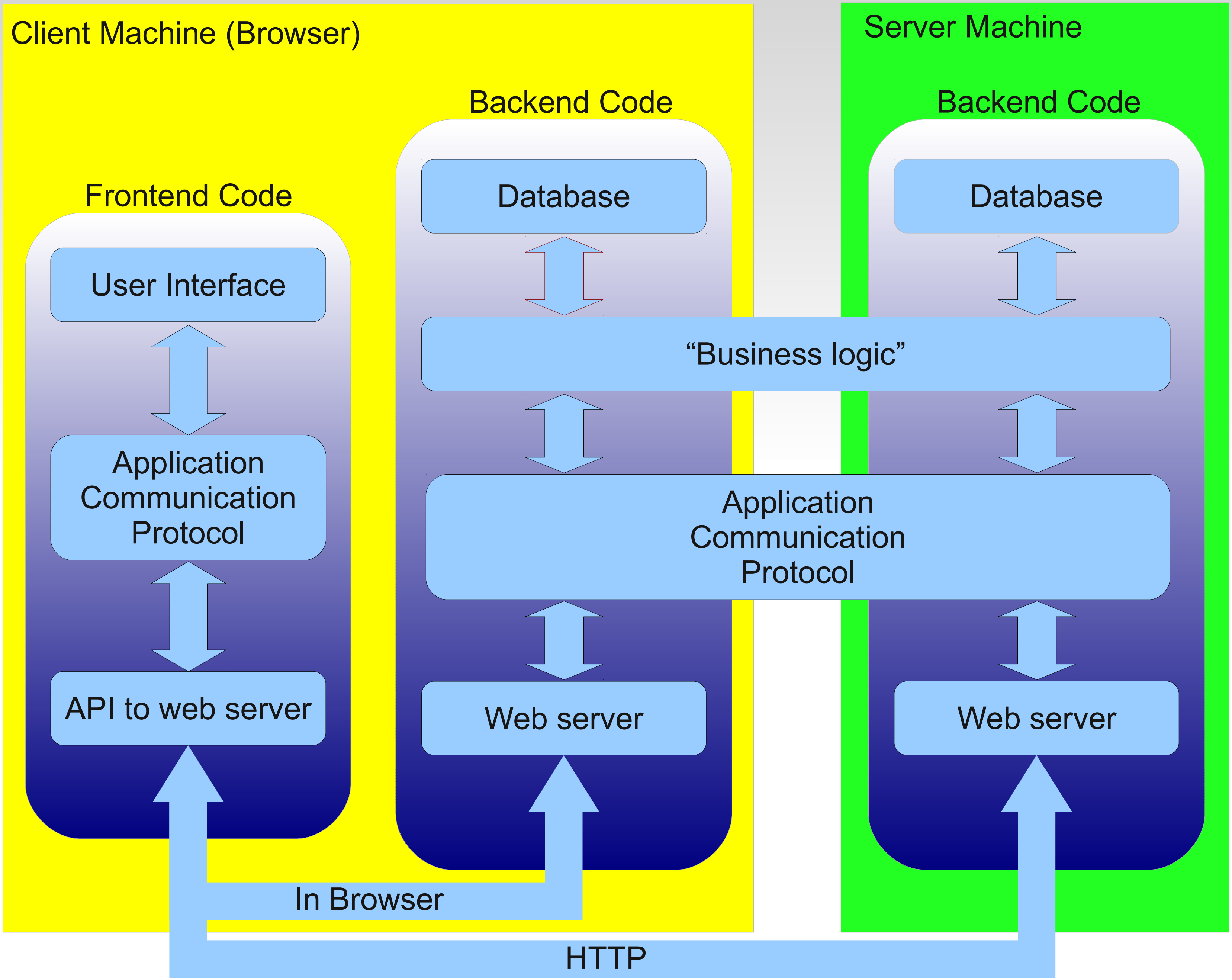
- Client/server interaction is asynchronous
- Not possible to step into server code from browser
- Different people (skill sets) required at client and server
- Methodologies, techniques differ between client and server

My Research Question

Is it feasible to design an architecture and framework for client/server application implementation that allows:

1. *all application development to be accomplished primarily in a single language,*
2. *application frontend and backend code to be entirely tested and debugged within the browser environment, and*
3. *fully-tested application-specific backend code to be moved, entirely unchanged, from the browser environment to the real server environment, and to run there?*

Desired Architecture



Research Sub-questions

How much of a compromise does this architecture impose, i.e., what common facilities become unavailable or more difficult to use?

Does this new architecture create new problems of its own?

The Pieces of the Puzzle

- Switchable transports
  - Add local transport to talk to in-browser server
  - Means to select transport

The Pieces of the Puzzle

- Switchable transports
  - Add local transport to talk to in-browser server
  - Means to select transport
- JavaScript code to run in-browser or on the real server
  - Server-side JavaScript
  - Application communication protocol server
  - Database operation abstraction

The Pieces of the Puzzle

- Switchable transports
  - Add local transport to talk to in-browser server
  - Means to select transport
- JavaScript code to run in-browser or on the real server
  - Server-side JavaScript
  - Application communication protocol server
  - Database operation abstraction
- Glue: In-browser vs. real server
  - Interface between database abstraction and the simulated in-browser, and real databases
  - Interface from incoming request to server code

The Pieces of the Puzzle

- Switchable transports
  - Add local transport to talk to in-browser server
  - Means to select transport
- JavaScript code to run in-browser or on the real server
  - Server-side JavaScript
  - Application communication protocol server
  - Database operation abstraction
- Glue: In-browser vs. real server
  - Interface from incoming request to server code
  - Interface between database abstraction and the simulated in-browser, and real databases
- An application to show that the architecture works

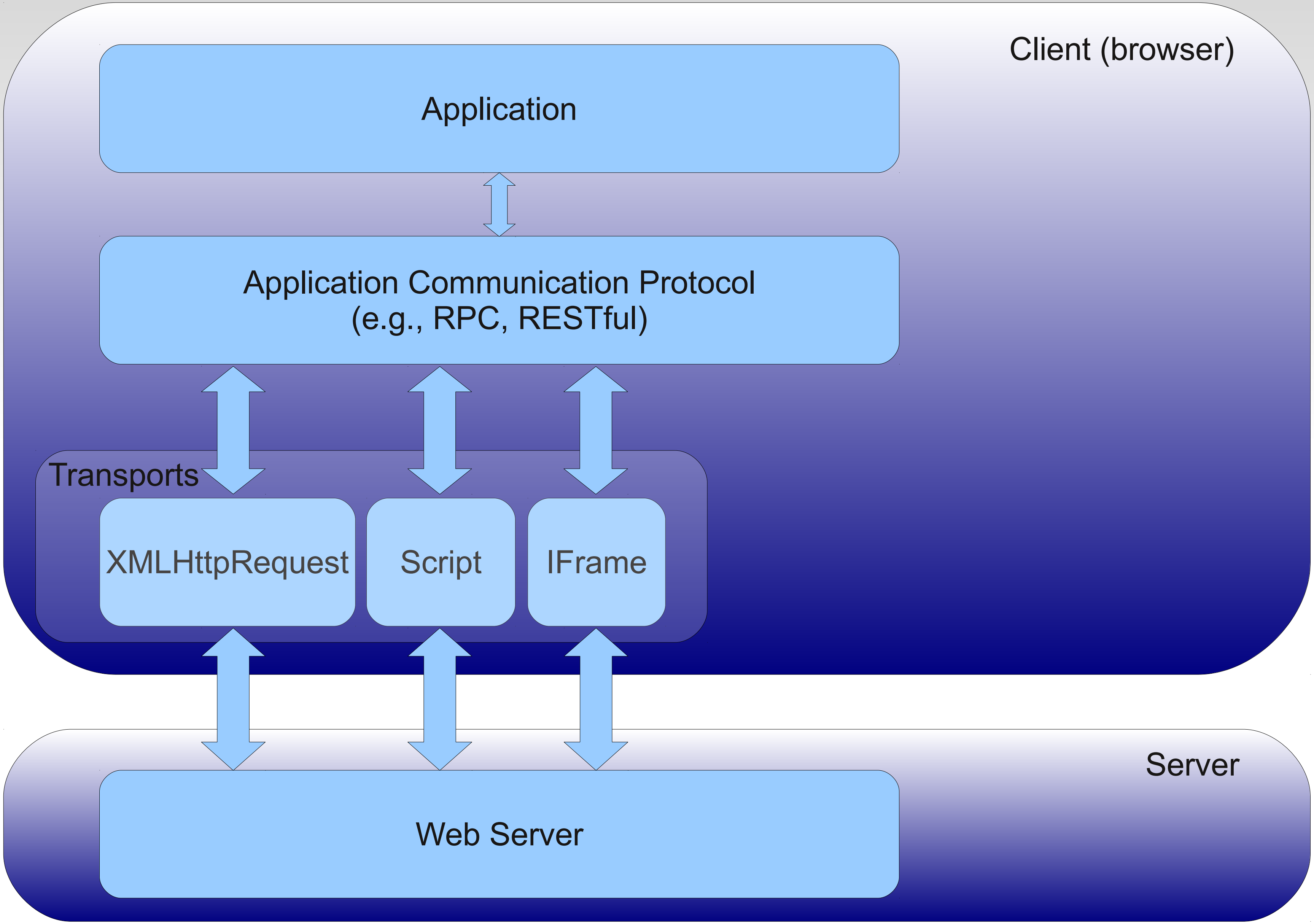
Introducing LIBERATED

- ***Liberates the developer from the hassles of traditional client/server application debugging and testing***
- Currently supports:
  - Simulation environment (in browser)
  - App Engine
  - Jetty / SQLite
- Implemented with *qooxdoo* JavaScript framework
  - Provides traditional class-based object programming model
  - LIBERATED could be used when developing a non-qooxdoo-based application

The Pieces of the Puzzle

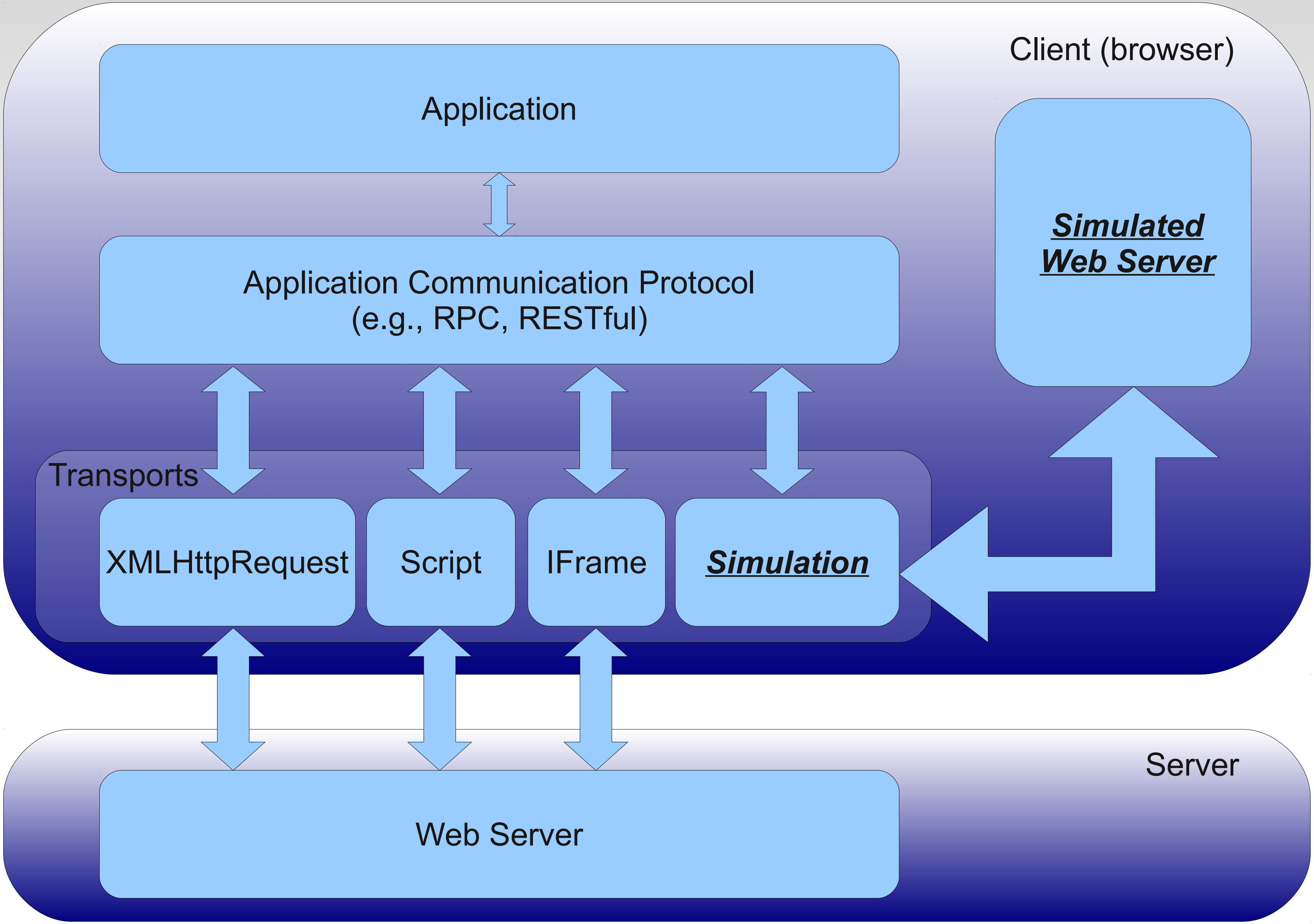
- Switchable transports
  - Add local transport to talk to in-browser server
  - Means to select transport
- JavaScript code to run in-browser or on the real server
  - Server-side JavaScript
  - Application communication protocol server
  - Database operation abstraction
- Glue: In-browser vs. real server
  - Interface from incoming request to server code
  - Interface between database abstraction and the simulated in-browser, and real databases
- An application to show that the architecture works

Transports



Client (browser)
Application
Application Communication Protocol
(e.g., RPC, RESTful)
Transports
XMLHttpRequest
Script
IFrame
Server
Web Server

Adding a Simulation Transport



The Pieces of the Puzzle

- Switchable transports
  - Add local transport to talk to in-browser server
  - Means to select transport
- JavaScript code to run in-browser or on the real server
  - Server-side JavaScript
  - Application communication protocol server
  - Database operation abstraction
- Glue: In-browser vs. real server
  - Interface from incoming request to server code
  - Interface between database abstraction and the simulated in-browser, and real databases
- An application to show that the architecture works

Server-side JavaScript

- Rhino
    - Mozilla Foundation
- SpiderMonkey
    - Embedded in some Mozilla products
- V8 (Node.js)
    - Used in the Chrome browser

- Via Rhino: any Java environment

In-browser or real server:
Application Communication Protocol

- Common approaches
  - REST (Representational State Transfer)
  - RPC (Remote Procedure Call)
    - XML-RPC
    - JSON-RPC (Very easy to implement in JavaScript)
- “Grow your own”

In-browser or real server:
Database Abstraction
Abstract class: Entity

Entity Class

Class Functions

•query ( )
- Retrieve data from one or more entity types in the database

•registerPropertyTypes ( )
- Specify the field values for this type of entity

•registerDatabaseProvider ( )
- A specific database server registers its handlers for all primitive operations

•putBlob ( )
•getBlob ( )
•removeBlob ( )
- Manage large objects

Entity Instance

Instance Properties

•data
- Per-entity field data

•brandNew
- Whether this entity was first retrieved from the database

•uid
- Unique auto-generated key, if no key field is specified

Instance Methods

•put ( )
- Write this entity to database

•removeSelf ( )
- Delete this entity from database

Entity Type definition for a simple counter

```
qx.Class.define("example.ObjCounter",
{
  extend : liberated.dbif.Entity,

  construct : function(id) {
    this.setData({ "count" : 0 });
    this.base(arguments, "counter", id);
  },

  defer : function() {
    var Entity = liberated.dbif.Entity;

    Entity.registerEntityType(example.ObjCounter, "counter");

    Entity.registerPropertyTypes(
      "counter",
      { "id" : "String", "count" : "Integer" },
      "id");
  }
});
```

RPC implementation

```
qx.Mixin.define("example.MCounter",
{
  construct : function() {
    this.registerService(
      "countPlusOne", this.countPlusOne,[ "counterId" ]);
  },
  members : {
    countPlusOne : function(counter) {
      var            counterObj, counterDataObj;

      liberated.dbif.Entity.asTransaction(
        function() {
          counterObj = new example.ObjCounter(counter);
          counterDataObj = counterObj.getData();
          ++counterDataObj.count;
          counterObj.put();
        }, [], this);

      return counterDataObj.count;
    }
  }
});
```

A second example, using a query

```
// Issue a query of dogs.
results = query(
  "app.Dog",          // Entity type to query.

  {                   // search criteria
    type     : "op",
    method   : "and",
    children :
    [
      { field : "breed",    value : "retriever"                  },
      { field : "training", value : "search"                     },
      { field : "age",      value : 3,           filterOp : ">=" }
    ]
  },

  [                   // result criteria
    { type : "limit",  value : 5  },
    { type : "offset", value : 10 },
    { type : "sort",   field : "age",   order : "desc" }
  ]);
```

SQL equivalent:

```
SELECT * FROM app.Dog
  WHERE breed = 'retriever'
        AND training = 'search'
        AND age >= 3
  SORT BY age DESC
  OFFSET 10
  LIMIT 5;
```

The Pieces of the Puzzle

- Switchable transports
  - Add local transport to talk to in-browser server
  - Means to select transport
- JavaScript code to run in-browser or on the real server
  - Server-side JavaScript
  - Application communication protocol server
  - Database operation abstraction
- Glue: In-browser vs. real server
  - Interface from incoming request to server code
  - Interface between database abstraction and the simulated in-browser, and real databases
- An application to show that the architecture works

Glue: Using common code in browser or real server

- Web server interface:
  - Request arrived
  - Sending response
- Interface with database
  - Simulation database
  - App Engine datastore
  - SQLite

The Pieces of the Puzzle

- Switchable transports
  - Add local transport to talk to in-browser server
- JavaScript code to run in-browser or on the real server
  - Server-side JavaScript
  - Application communication protocol server
  - Database operation abstraction
  - Compiling JavaScript to Java's .class format
- Glue: In-browser vs. real server
  - Interface from incoming request to server code
  - Interface between database abstraction and the simulated in-browser, and real databases
- An application to show that the architecture works

Incorporating into an application: App Inventor Community Gallery

- App Inventor (Google / MIT)
  - Blocks programming language (puzzle pieces)
  - Similar to Scratch, Lego Mindstorms environments
  - Destination: Android phones
- App Inventor Community Gallery
  - Sharing of source code, libraries, components
  - Social aspects: “Like It!”, comments
  - Developed and tested using in-browser backend
    - Unit/regression tests for individual RPC implementations and for full RPC round-trip invocation
  - Runs on App Engine

Related Work

- Nothing else fully answers my research question???
- Areas of related work
  - Google Web Toolkit (GWT)
  - Plain Old Webserver
  - Wakanda

Conclusions

Is it feasible to design an architecture and framework for client/server application implementation that allows:

1. *all application development to be accomplished primarily in a single language,*
2. *application frontend and backend code to be entirely tested and debugged within the browser environment, and*
3. *fully-tested application-specific backend code to be moved, entirely unchanged, from the browser environment to the real server environment, and to run there?*

YES! (with caveats)

Compromises and New Problems Imposed by This Architecture

- Compromises
  - Browser database capabilities are limited
  - Limited number of property types
  - Required schema
  - Conversion from native language to JavaScript
  - Database driver mappings – difficult?
- New problems
  - Server-side JavaScript is still young
    - Plentiful libraries of code available elsewhere are not yet here (but Node is quickly solving this)

Future Work

- Rigorous evaluation of LIBERATED vs. more traditional development paradigms.
- Determine impact of described compromises
  - May require implementing parallel environment in different language
- Object relations
  - Currently ad-hoc, enforced by application
- Better browser-based persistent storage
  - Indexed Database instead of localStorage?
- Additional operators in queries
  - Currently only “and” is supported

Source Code

- *LIBERATED*
  - https://github.com/liberated/liberated
- App Inventor Community Gallery
  - https://github.com/app-inventor-gallery/aig