Rich Cloud-Based Web Applications with CloudBrowser 2.0

Xiaozhong Pan and Godmar Back

Virginia Tech

Outline

- History and motivation
- Background of this work
  - CloudBrowser 1.0
- CloudBrowser 2.0
  - Application Deployment
  - Distributed implementation
- Evaluation
- Related work
- Conclusion

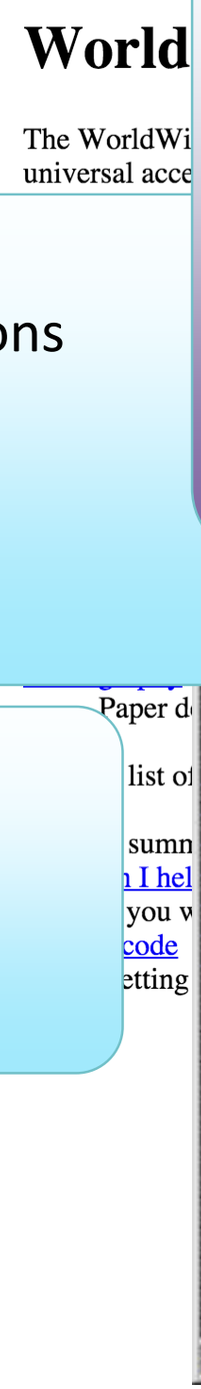
A Brief History of Web Applications

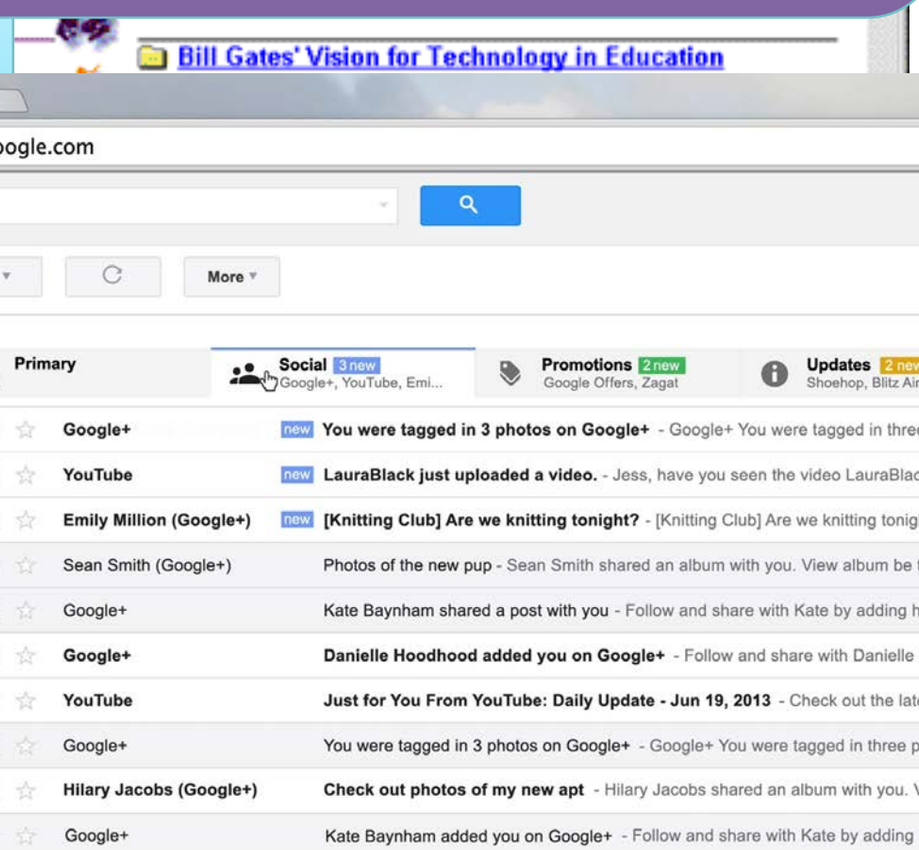
Today, practically every new application is in some form “cloud-based” and accessible through a web interface.

Rich Internet Applications
- Ajax (MS 1999)
- jQuery (2006)
- ZK (2006)

Simple Interactive Applications
- PHP, JSP, ASP…
- 1994-

Static Pages
- HTML
- 1990





Filing your Taxes (in the U.S.)

TaxACT Online Deluxe

Print | E-file | Register Now | Sign out

Basic Info | Life Events | Federal Q&A | State Q&A | Review | Filing | Next Year

Import | Personal Info | Dependents | Filing Status | Estimated Payments | Jump To Forms & Topics

Federal Refund: $2,744

Name, Social Security Number and Date of Birth

Bookmarks

Please complete the taxpayer information below.

First name:

Middle initial:

Last name:

Social security number:

Date of birth (mm/dd/yyyy):

Occupation:

Should you file a Form 1040NR - U.S. Nonresident Alien Income Tax Return?

Form 1040

Back | Continue

Help | Forms | Images | Tools

Popular Help Topics

- Name, Birthdate and Social Security Number Mismatch
- Social Security Number Change
- What Do I Need to File?

TaxTutor Guidance

- Name or Address Change
- Social Security Number (SSN)
- Individual Taxpayer Identification Numbers (ITINs) for Aliens

IRS Form Instructions

Need more assistance?

Search Answer Center: Go

Browse Answer Center

How to amend your Federal tax return | Privacy Policy

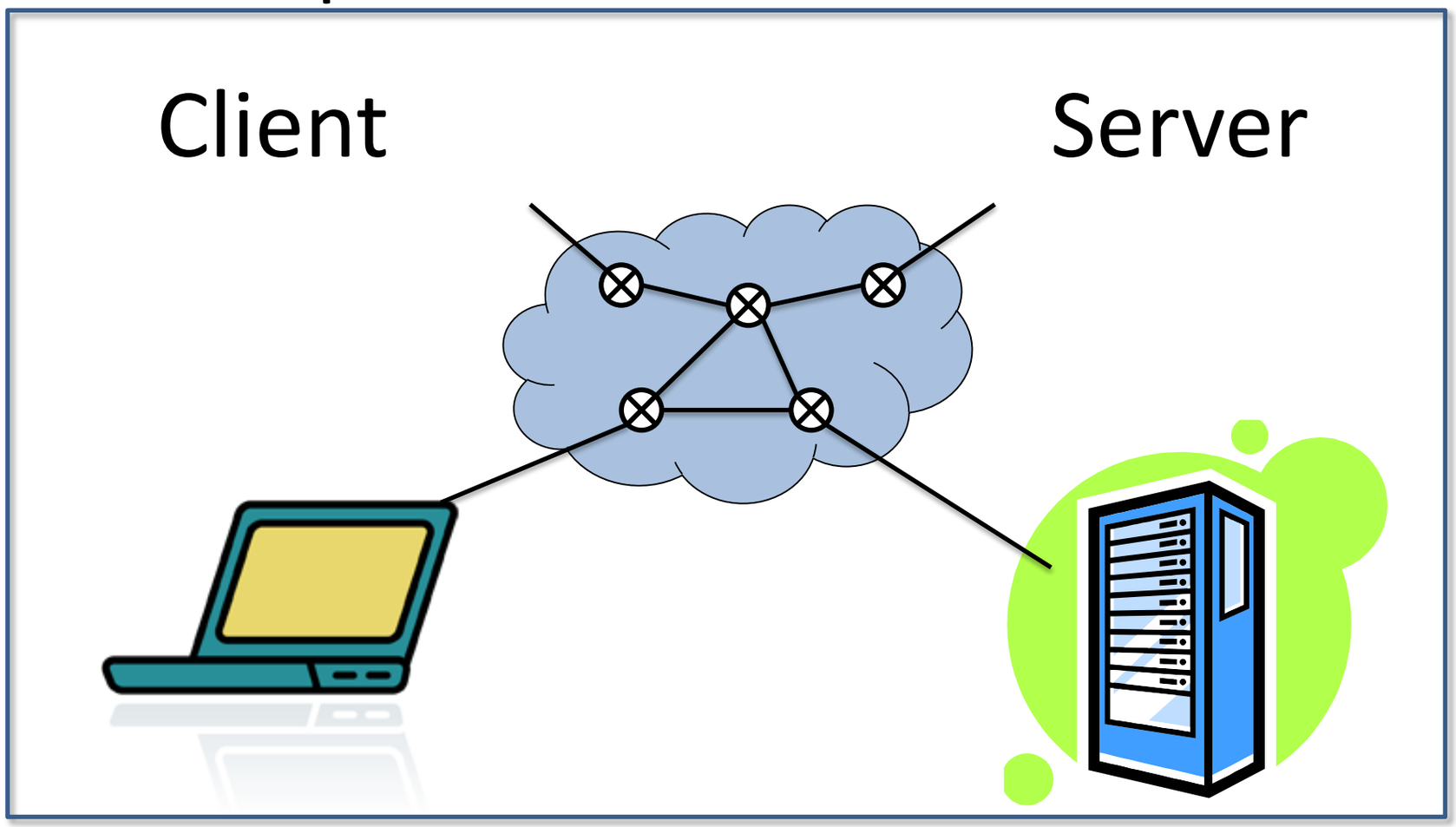
Target Applications

- Single Page applications with deep navigation space
- Users will connect and disconnect multiple times, possible using multiple devices
- Users wish for (most) actions to take immediate effect, i.e., to be remembered

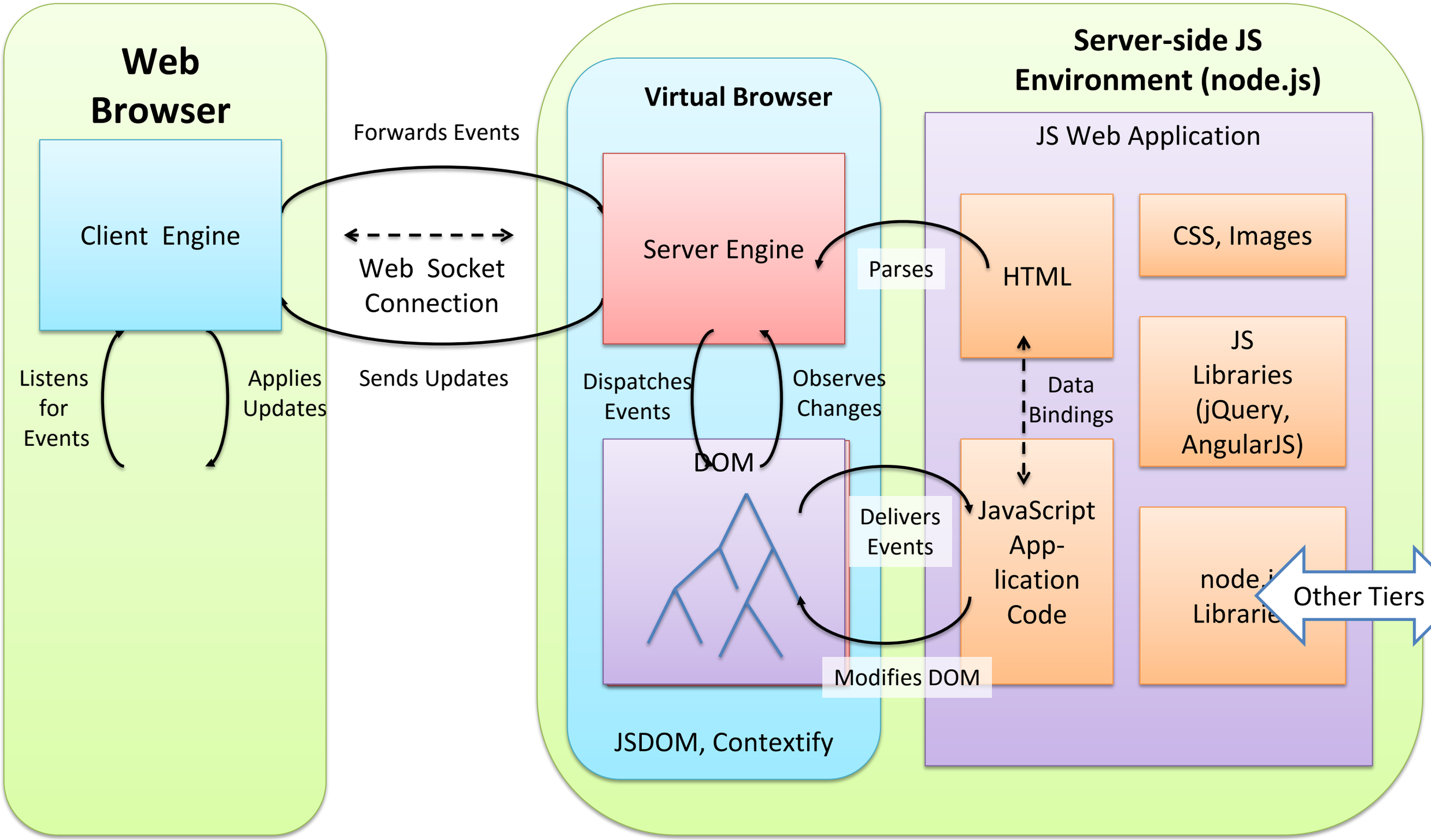
- Examples:
  - Tax preparation
  - Configuration management applications
  - Business Workflow applications

Motivation

- What makes creating web-based cloud applications hard?
  - Distributed programming model
  - Client-side DOM is ephemeral



CloudBrowser Architecture



Benefits of CloudBrowser 1.0

- Single language : application logic is entirely in JavaScript
- Single place : client-server communication is abstracted away
- Reuse existing technology and developer experience: JavaScript libraries, CSS frameworks
- Continuous experience for user

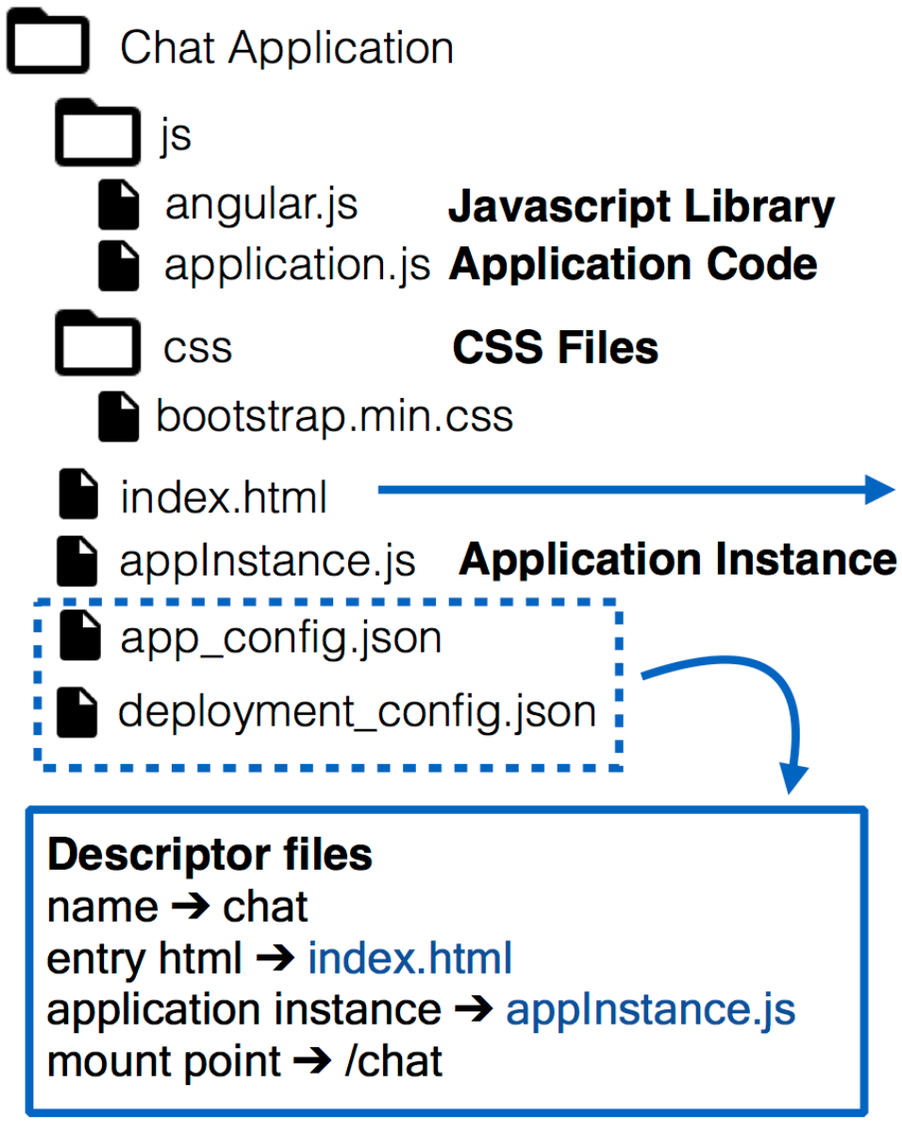
Limitations

- Virtual DOM lacks layout attributes
  - Use CSS-based libraries
- Added Latency
  - Do not dispatch all events
- Resource Consumption
  - High-level JS libraries
- Scalability
  - Node.js is single-threaded, event-based

CloudBrowser 2.0 Contributions

Problem	Solution
Unsuitable for PaaS deployment	Application Bundle
Lack of way to share data across virtual browsers	App Instance Concept
Lack of virtual browser management	Application Instantiation Strategies
Non-scalable, single-process implementation	Multi-process Implementation
Lack of isolation for applications	Encapsulated API layer

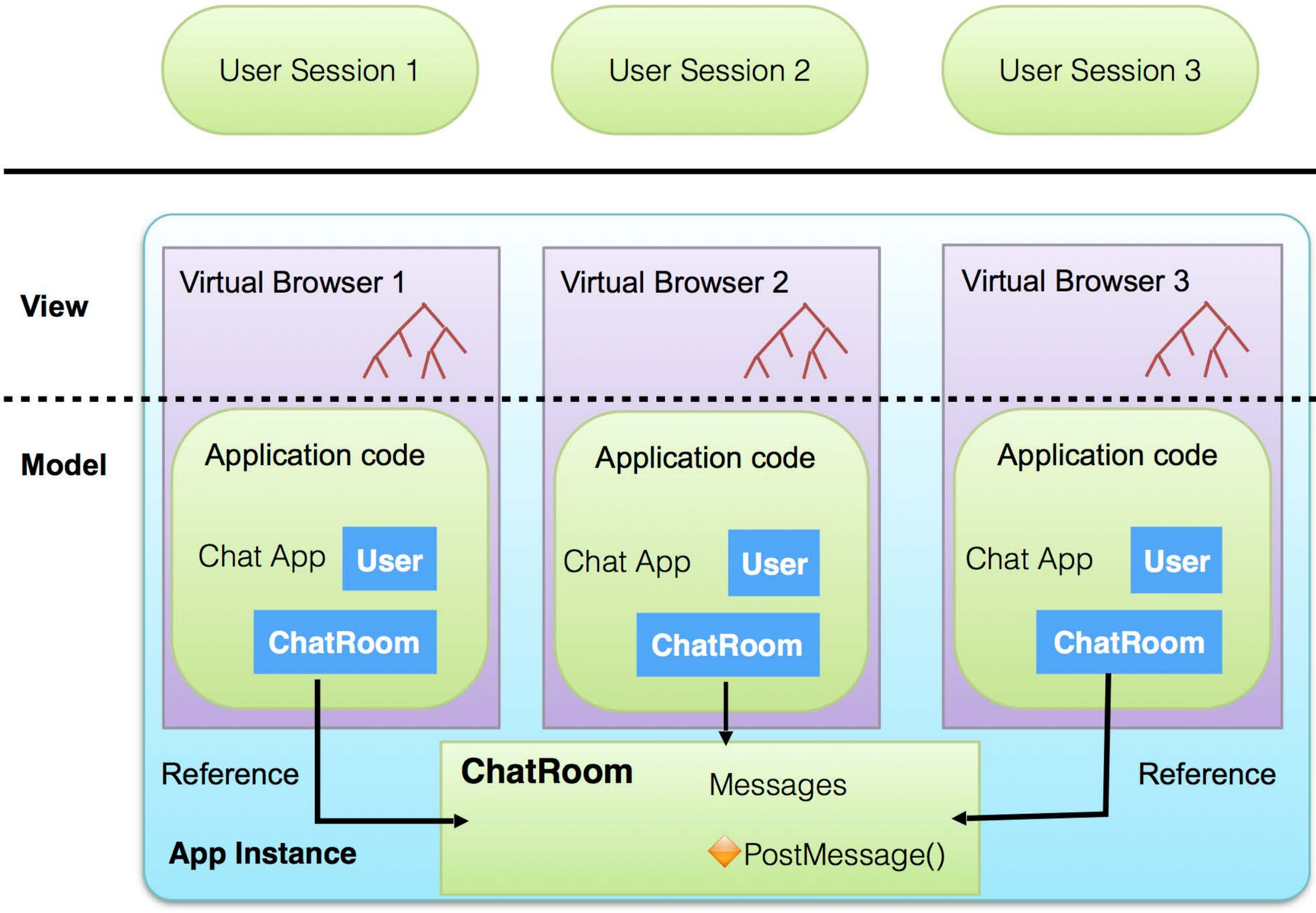
CloudBrowser Application Bundles




```
<!DOCTYPE html>
<html>
<head>
   <title>Open Chat Application</title>
   <link href="css/bootstrap.min.css" rel="stylesheet" />
   <script type="text/javascript" src="js/angular.js"></script>
   <script type="text/javascript" src="js/application.js"></script>
</head>
<body ng-app="Chat4" ng-controller="ChatCtrl">
<!--main div -->
<div class="container-fluid">
   <h1>Chat Room</h1>
   <!-- chat window-->
   <div class="panel panel-primary">
    …
   </div>
</div>
</body>
</html>
```

- CloudBrowser is PaaS service, supports deployment of multiple applications
- Applications are defined by application bundles

App Instance



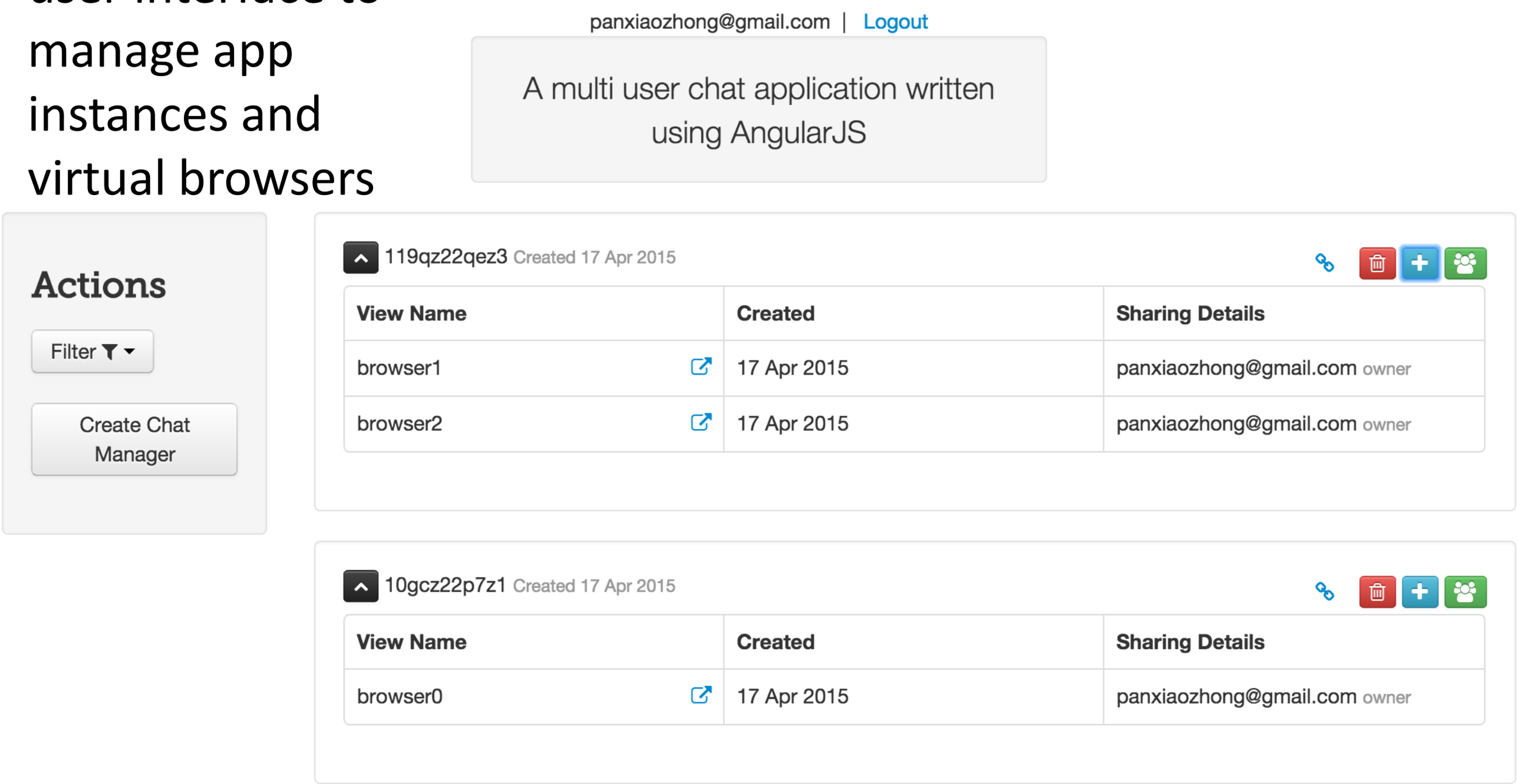
Application Instantiation Strategies

- multiInstance : most flexible model
- singleBrowserPerUser : multi-user applications, e.g. chatroom
- singleInstancePerUser : single-page style applications
- singleAppInstance : non-interactive information page, e.g. weather

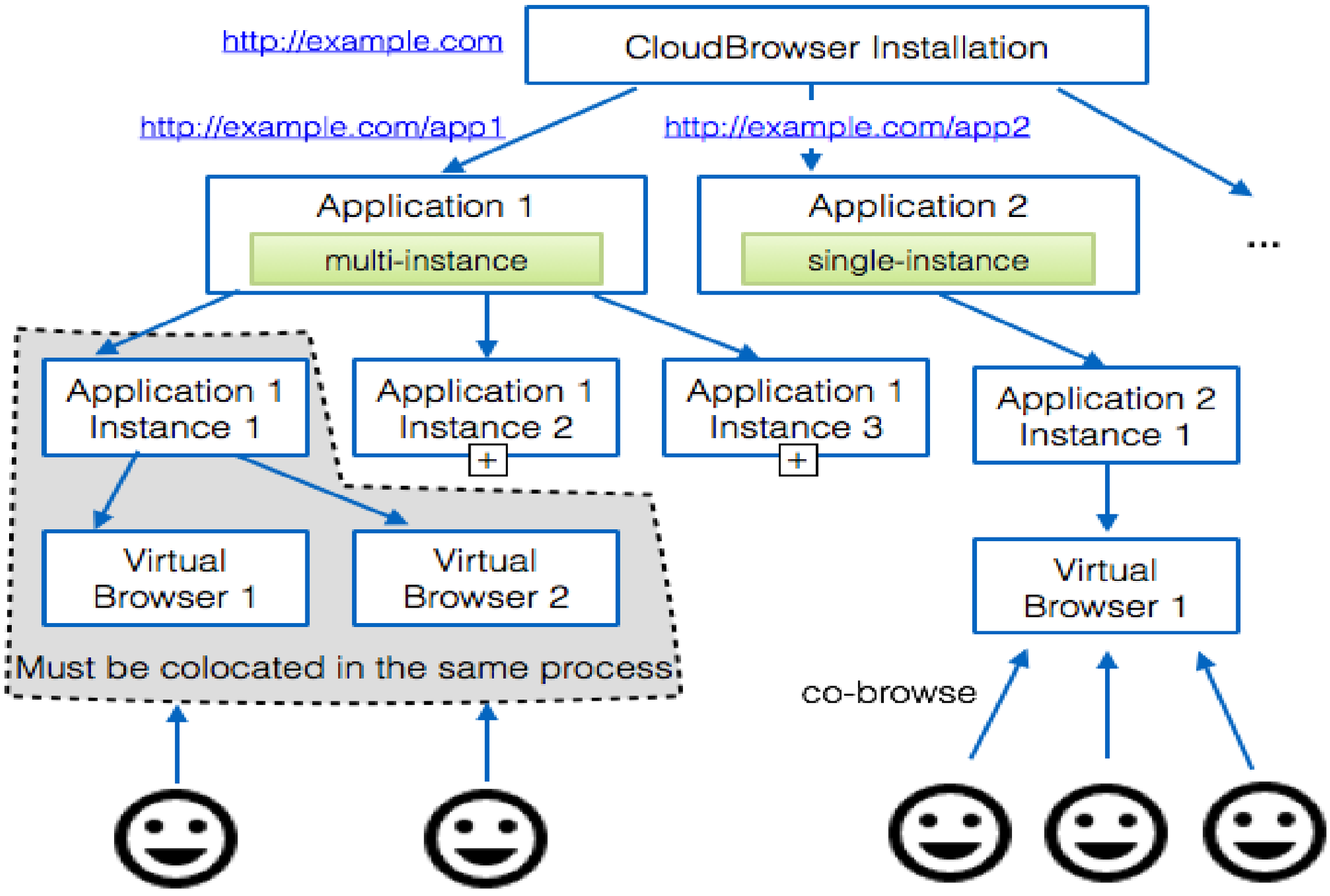
Instantiation Strategy	Number of App Instances	Number of Browsers Per App Instance	User Manages App Instance	User Manages Virtual Browser
multiInstance	any	any	Yes	Yes
singleBrowserPerUser	any	1 per user	Yes	No
singleInstancePerUser	1 per user	1 total	No	No
singleAppInstance	1 total	1 total	No	No

Application Management

Landing Page: user interface to manage app instances and virtual browsers



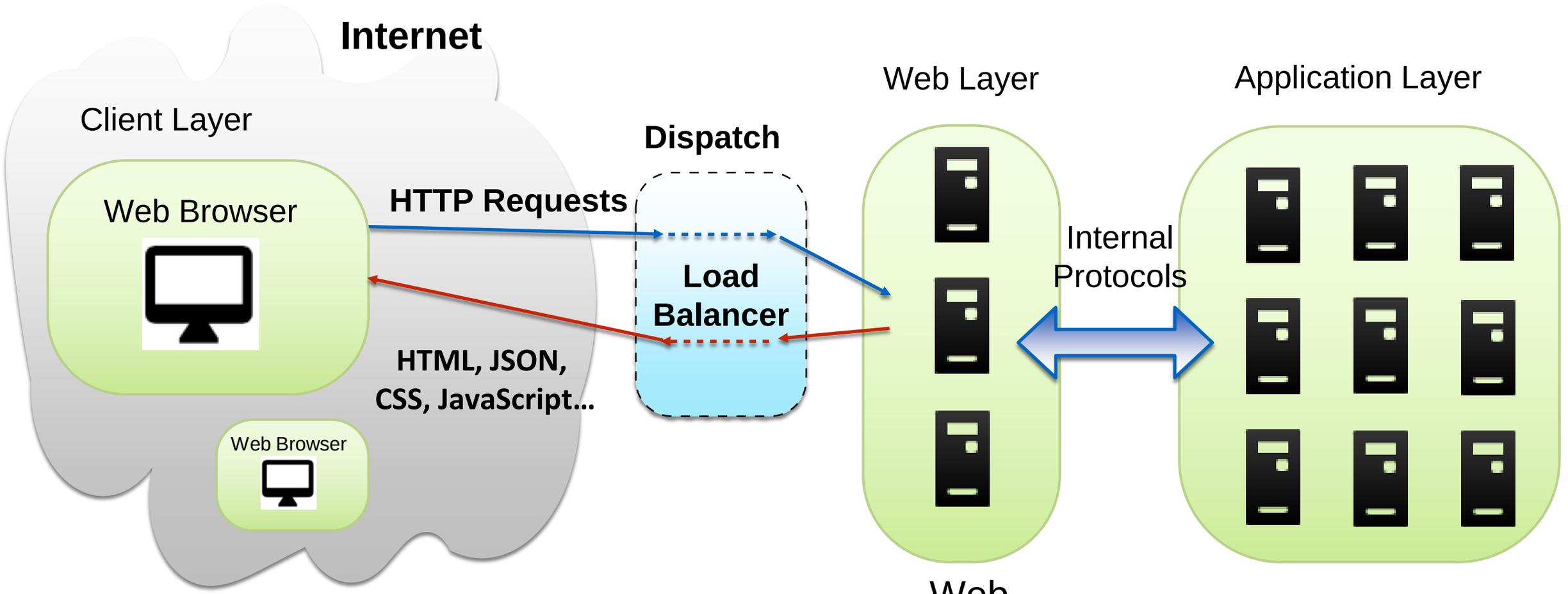
Application Deployment Model



Scaling to Multiple Processes

- Key Insight:
  - Traditional techniques for session-state replication cannot be used
  - Application deployment imposes restrictions on the mapping of virtual browsers to processes

Scalable Web Applications



HTML, CSS, Javascript, Bootstrap, JQuery, Angular, Reactjs …

Web Servers

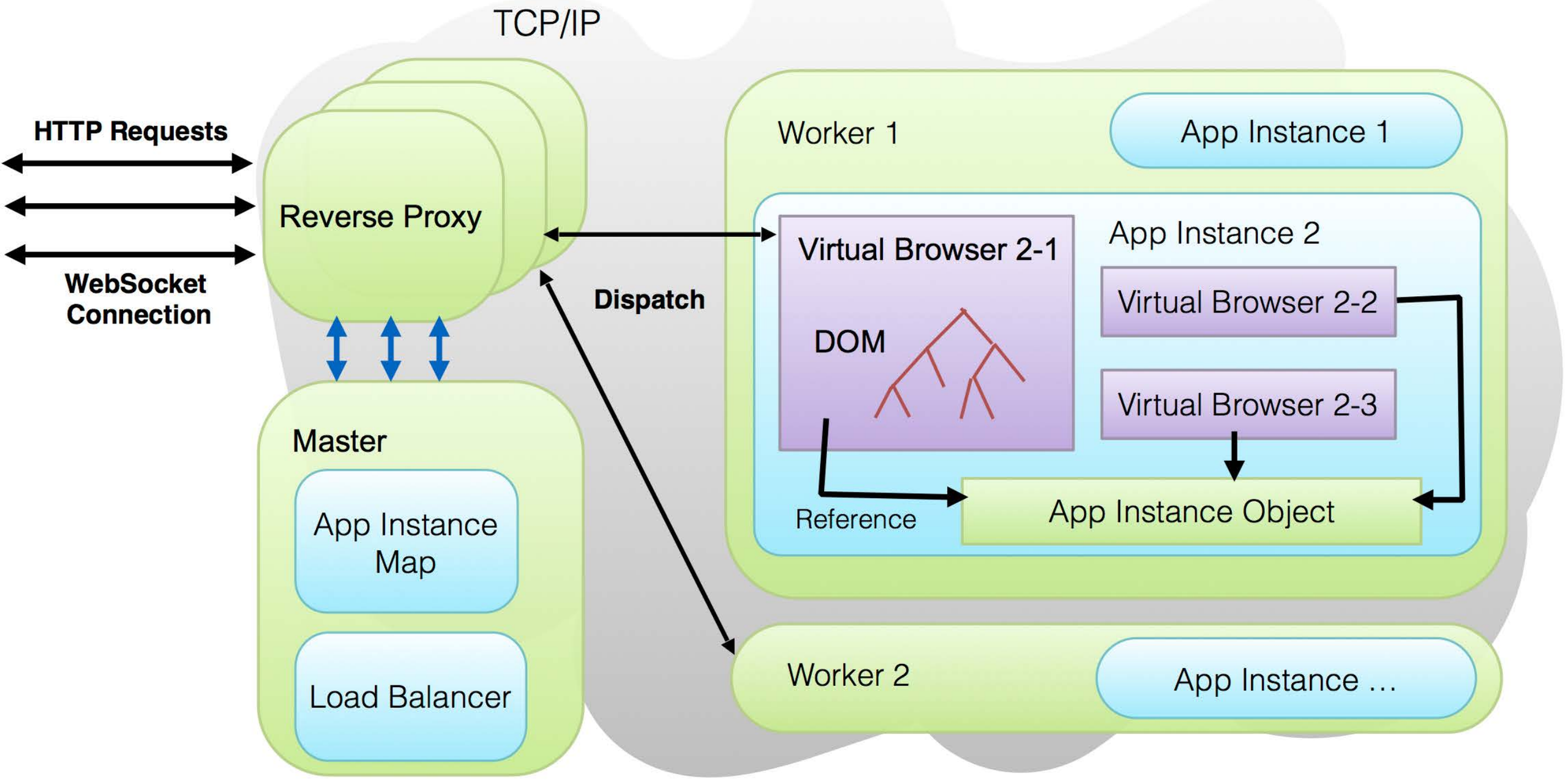
Tomcat, Apache, PHP, JSP, Ruby on Rails …

Back-end Servers

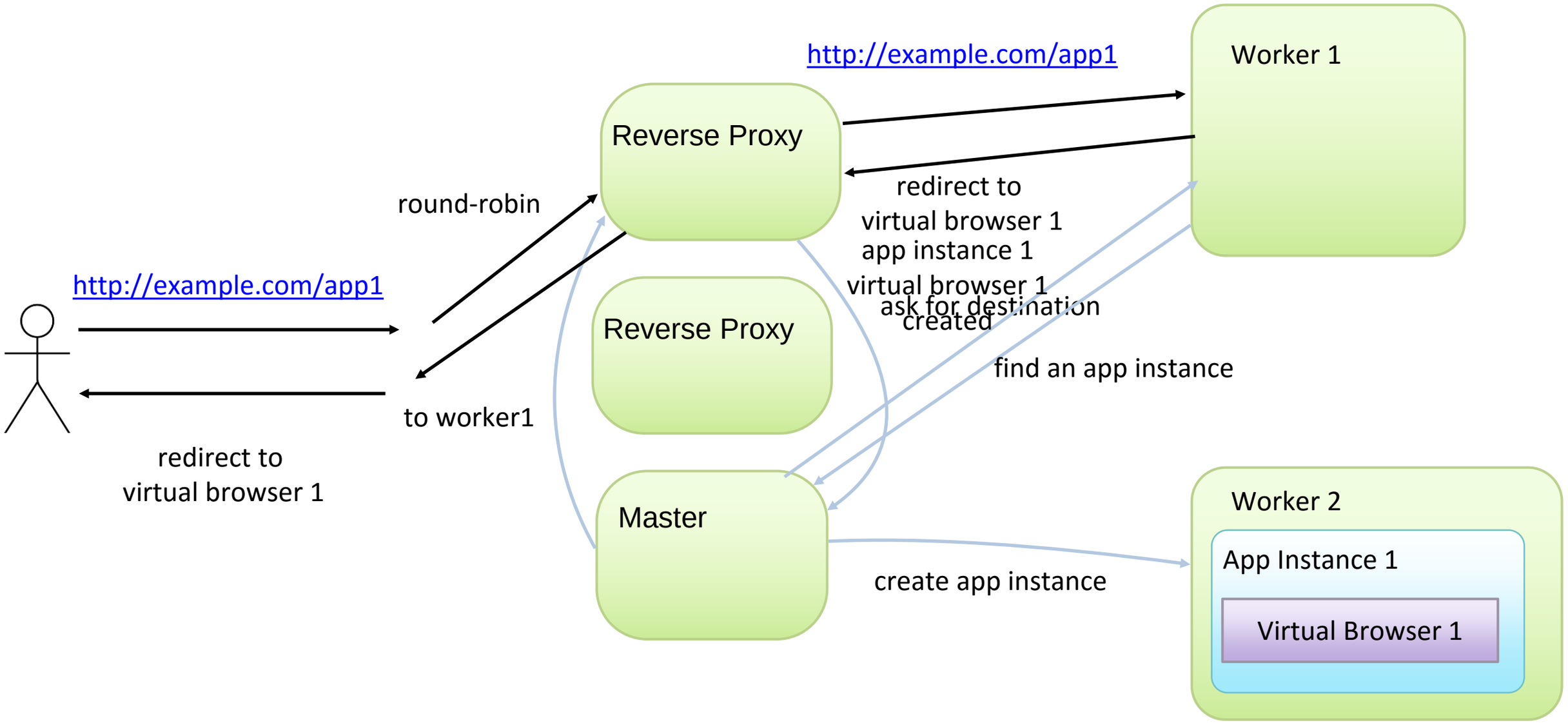
MySql, ActiveMQ, Memcached, Redis …

Session State Replication

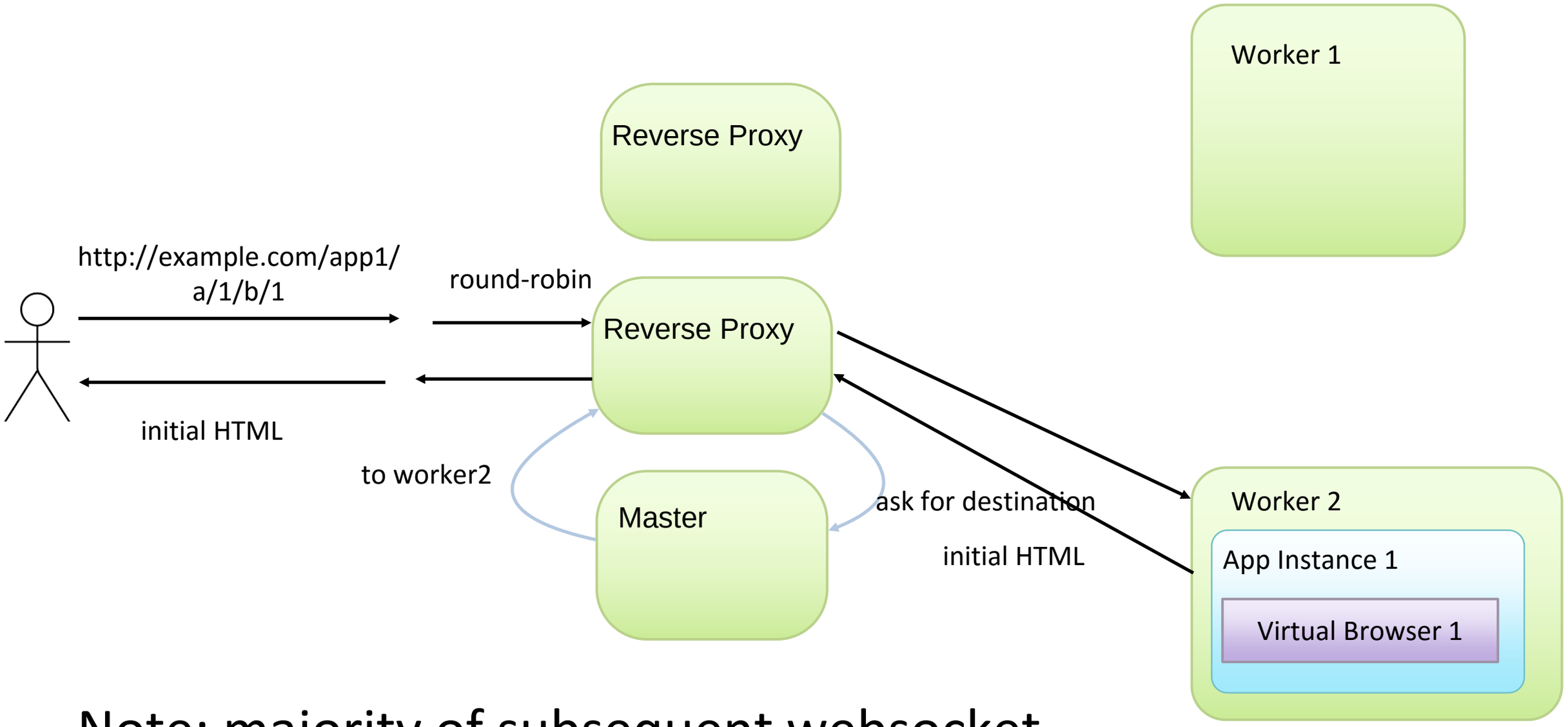
CloudBrowser 2.0 Architecture



CloudBrowser2.0 — Request Dispatch



CloudBrowser2.0 — Request Dispatch



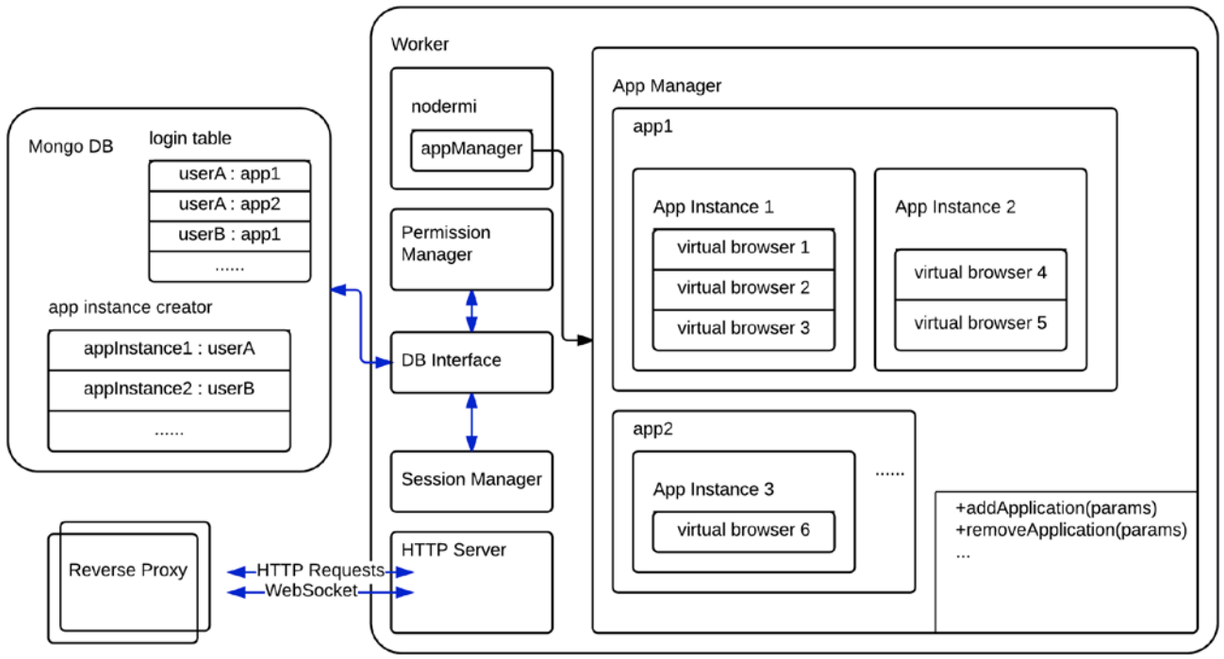
Note: majority of subsequent websocket traffic is directly forwarded to worker

CloudBrowser 2.0 — Load Balancing

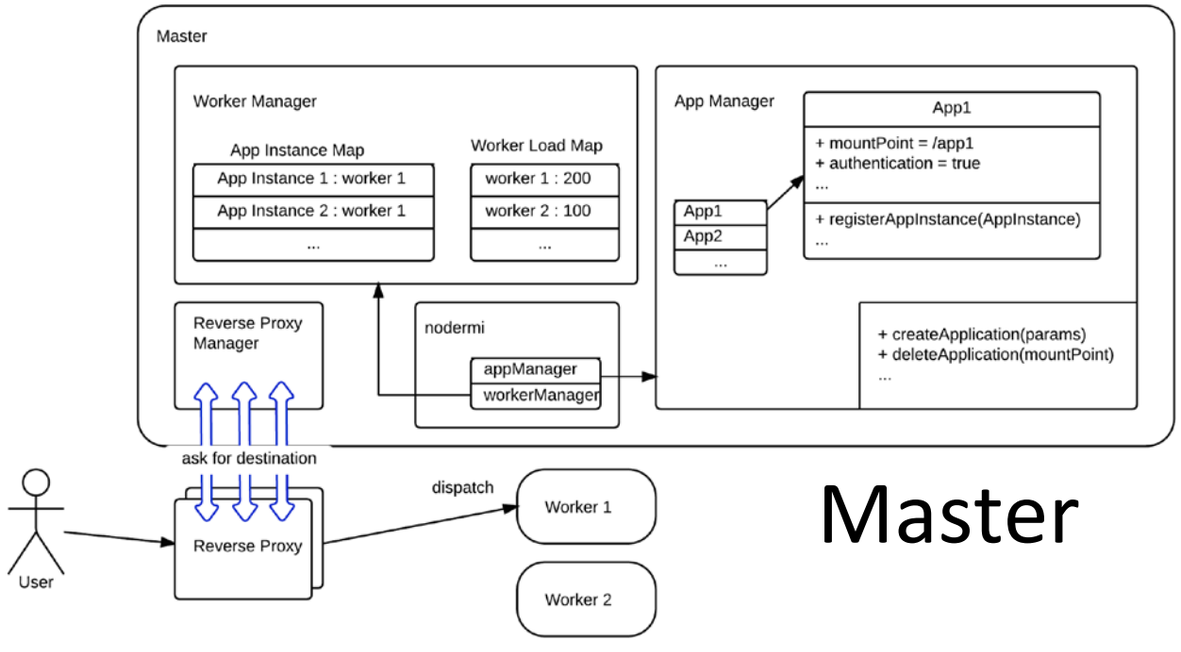
- Round-robin: assign new requests to workers in round-robin fashion
- Load-based
  - Workers periodically report their load to the master
  - Master selects the worker with the lowest load
  - To handle bursts of requests, master projects impact on selected worker's load

Master-Worker Communication

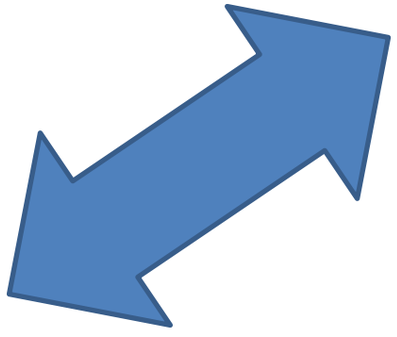
- Turned original non-distributed code base into distributed application



Worker

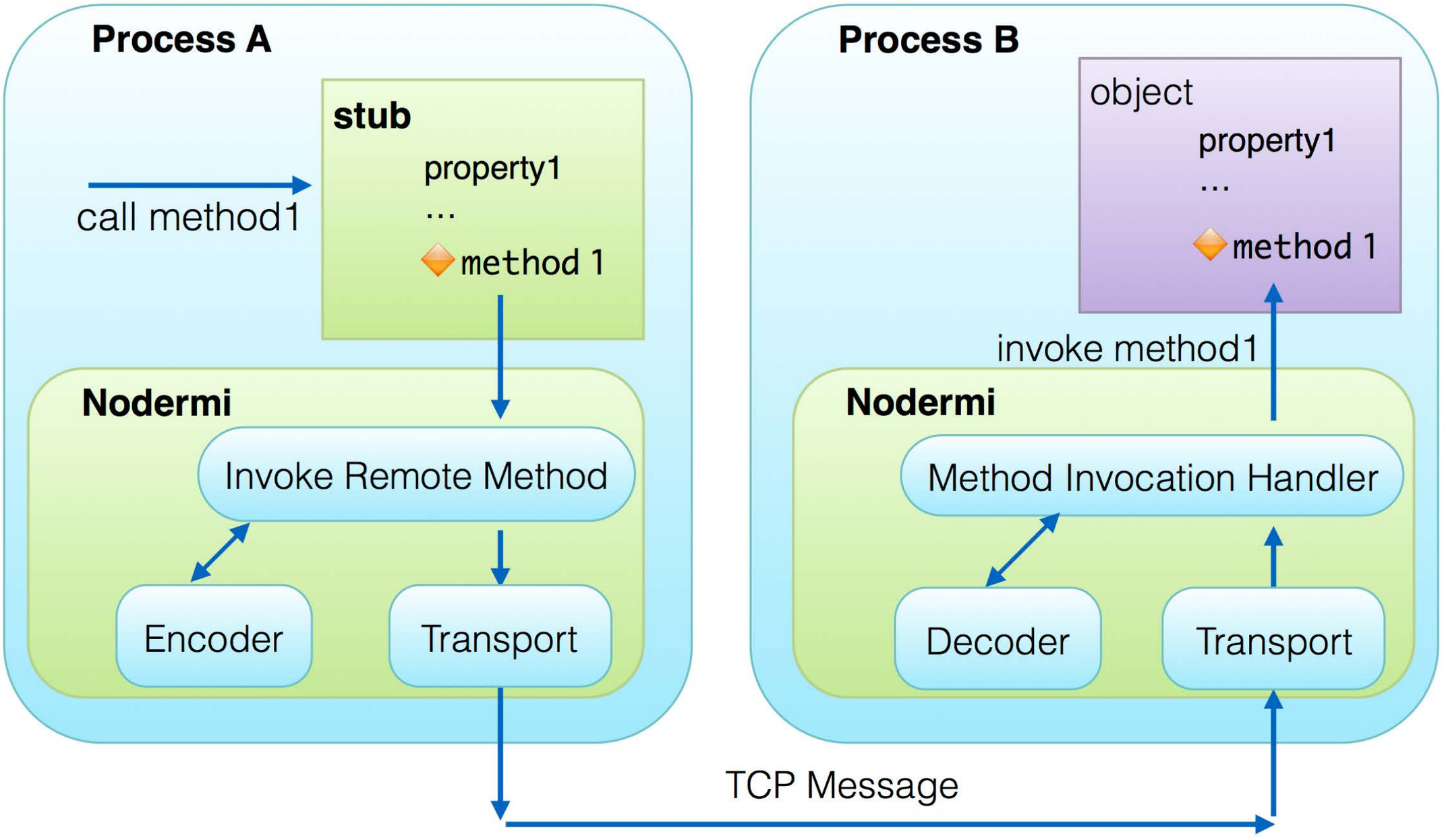


Master



Solution:
Remote Method Invocation
“nodermi”

nodermi — Overview



- Nodermi creates stubs to represent remote objects
- Method calls on stubs trigger remote method invocations

nodermi - HighLights

- Automatic stub creation for JS object methods
  - No IDL or annotations needed
- Marshaling
  - Handles primitive types, arrays, and built-in types
  - Handles cyclic object graphs
- Reference management
  - Avoid zig-zag
  - Avoids need for forwarding
- Distributed garbage collection
  - Using stub maps

Nodermi - Limitations

- Does not handle property assignments
  - Only supports remote method invocations
- All calls are asynchronous
  - I.e., instead of return values, callback functions are passed and invoke remotely
    - Software engineering challenge
- Does not handle distributed cycles that span across processes
  - Application must avoid those

Evaluation — Goal

- Evaluate scalability with respect to the number of workers
- Measure cost of different kinds of applications and client libraries
- Identify bottlenecks of the system

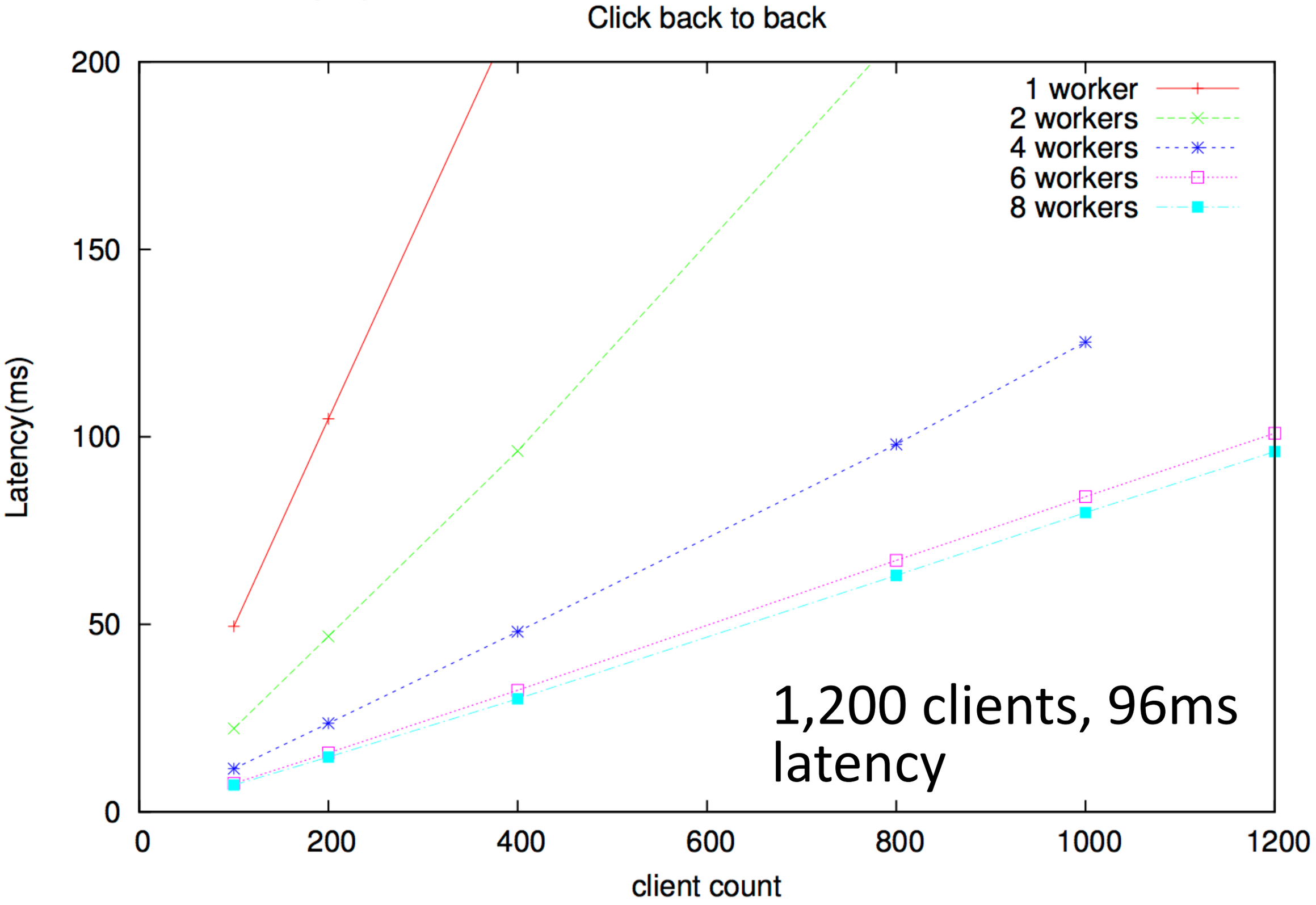
Evaluation — Methodology

- Benchmark applications
  - Click : simple application with no libraries
  - jQuery Chat : complex application with low level libraries
  - Angular Chat : complex application with advanced libraries
- Benchmark tool
  - Simulate how concurrent users interact with the applications
  - Use an expect-like configuration to describe scenarios
- Performance under different concurrent levels and different number of workers
  - Number of users for less than 100ms latency

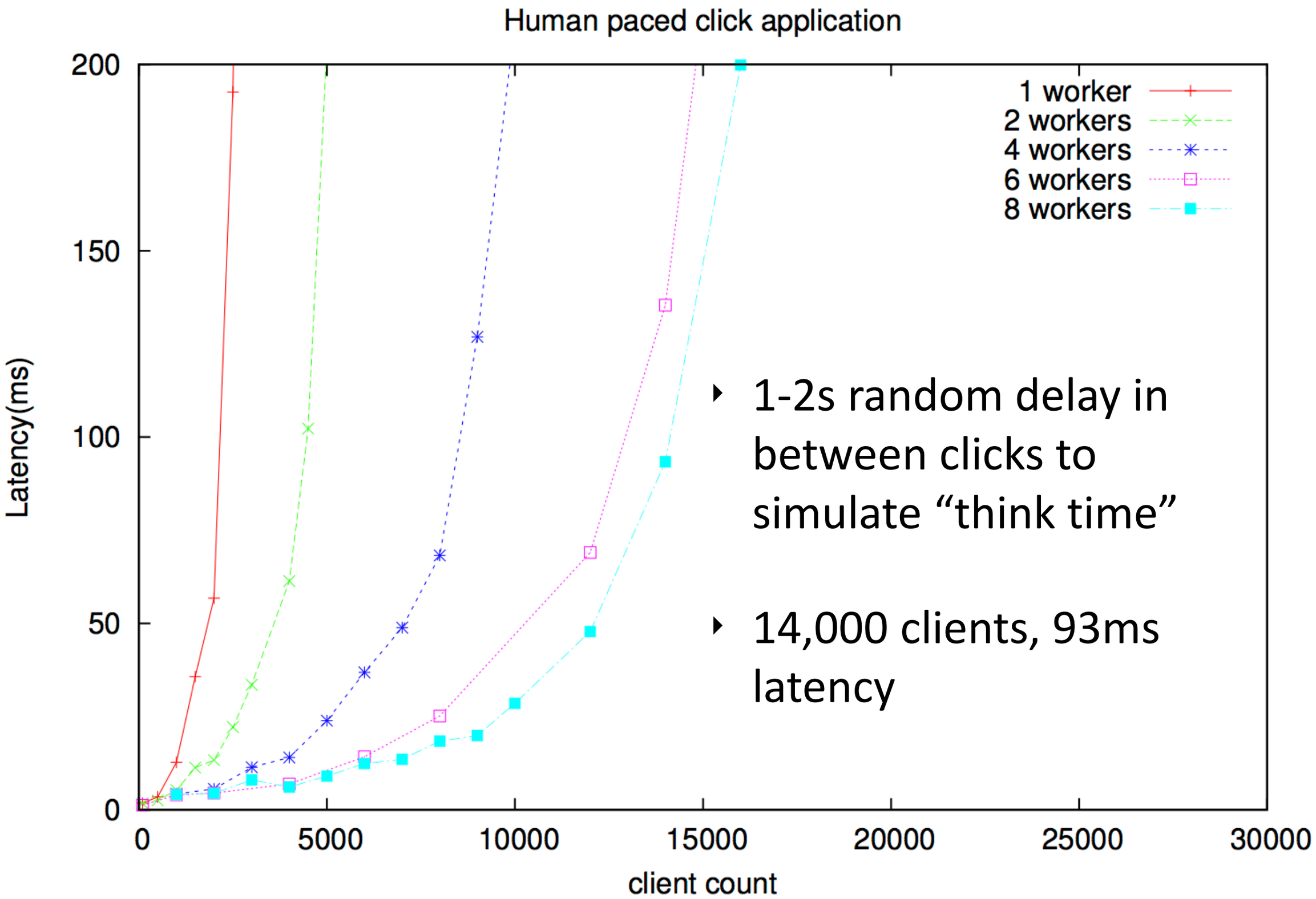
Evaluation - Testbed

- Server : 2*4 2.27GHz cores, 38GB RAM
- Client : 2*4 900MHz cores, 16GB RAM
- Connected by 1GB ethernet

Click Application

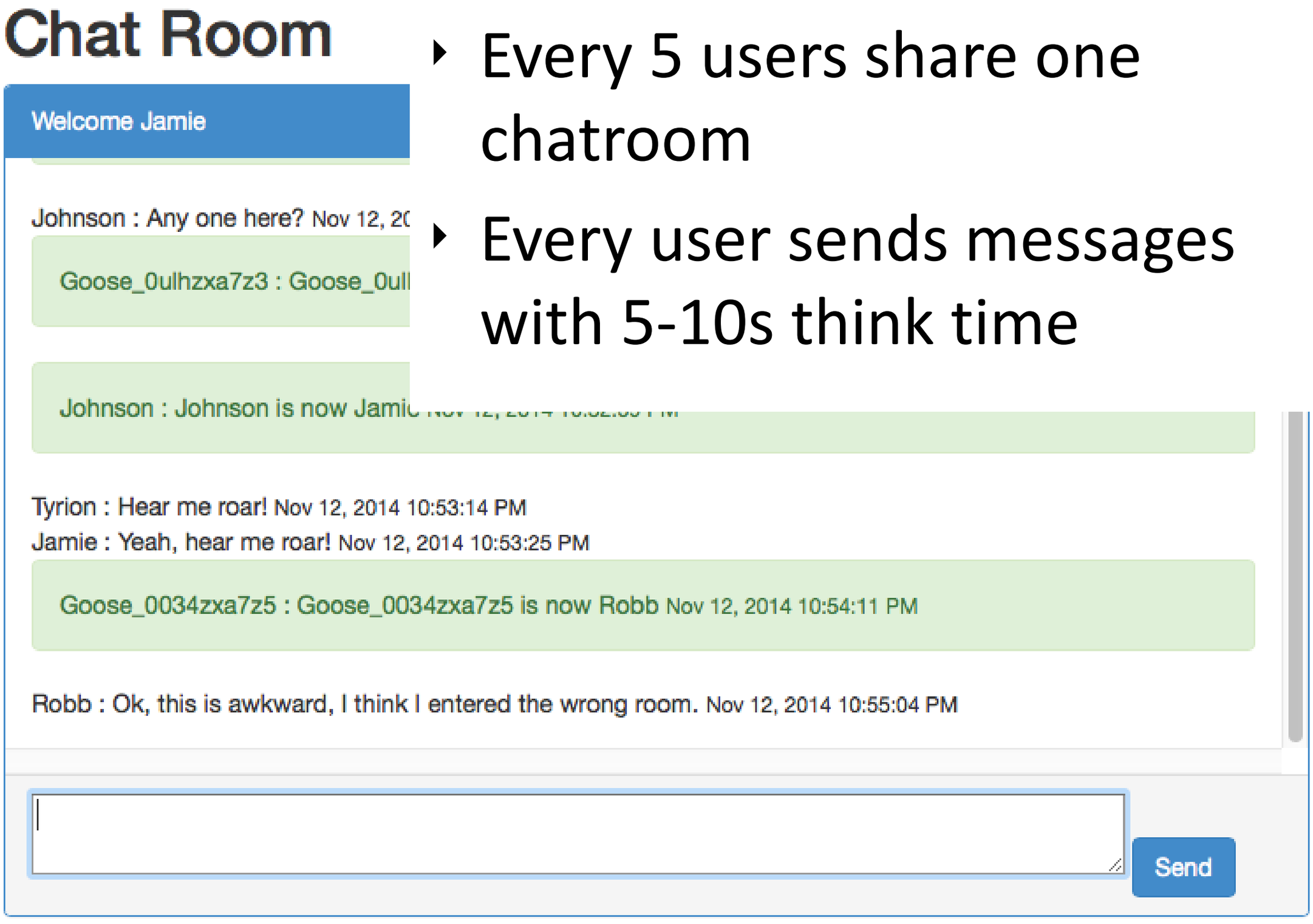


Click Application (Human Paced)



- 1-2s random delay in between clicks to simulate “think time”
- 14,000 clients, 93ms latency

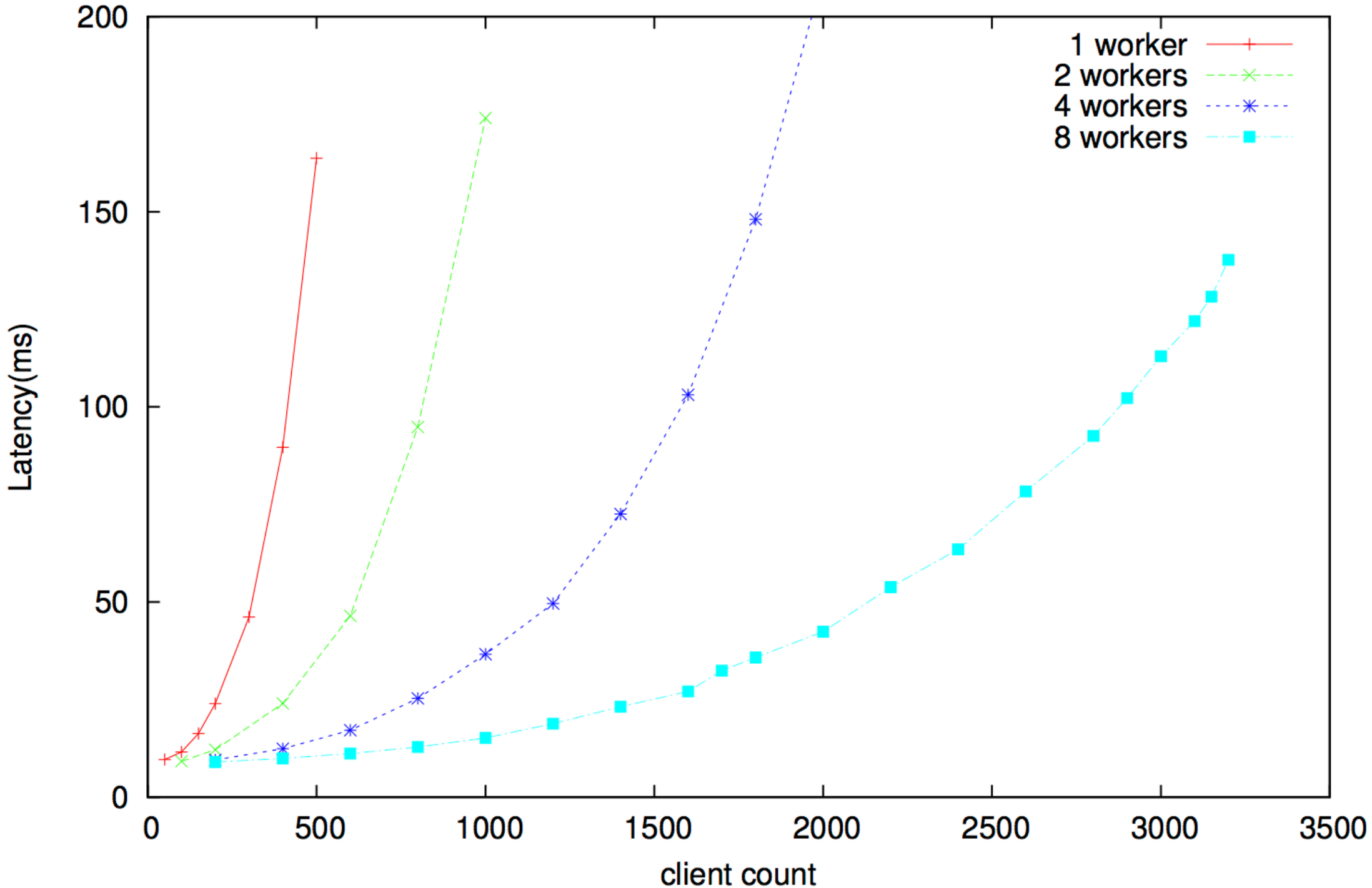
Evaluation — Chat Application



- Every 5 users share one chatroom
- Every user sends messages with 5-10s think time

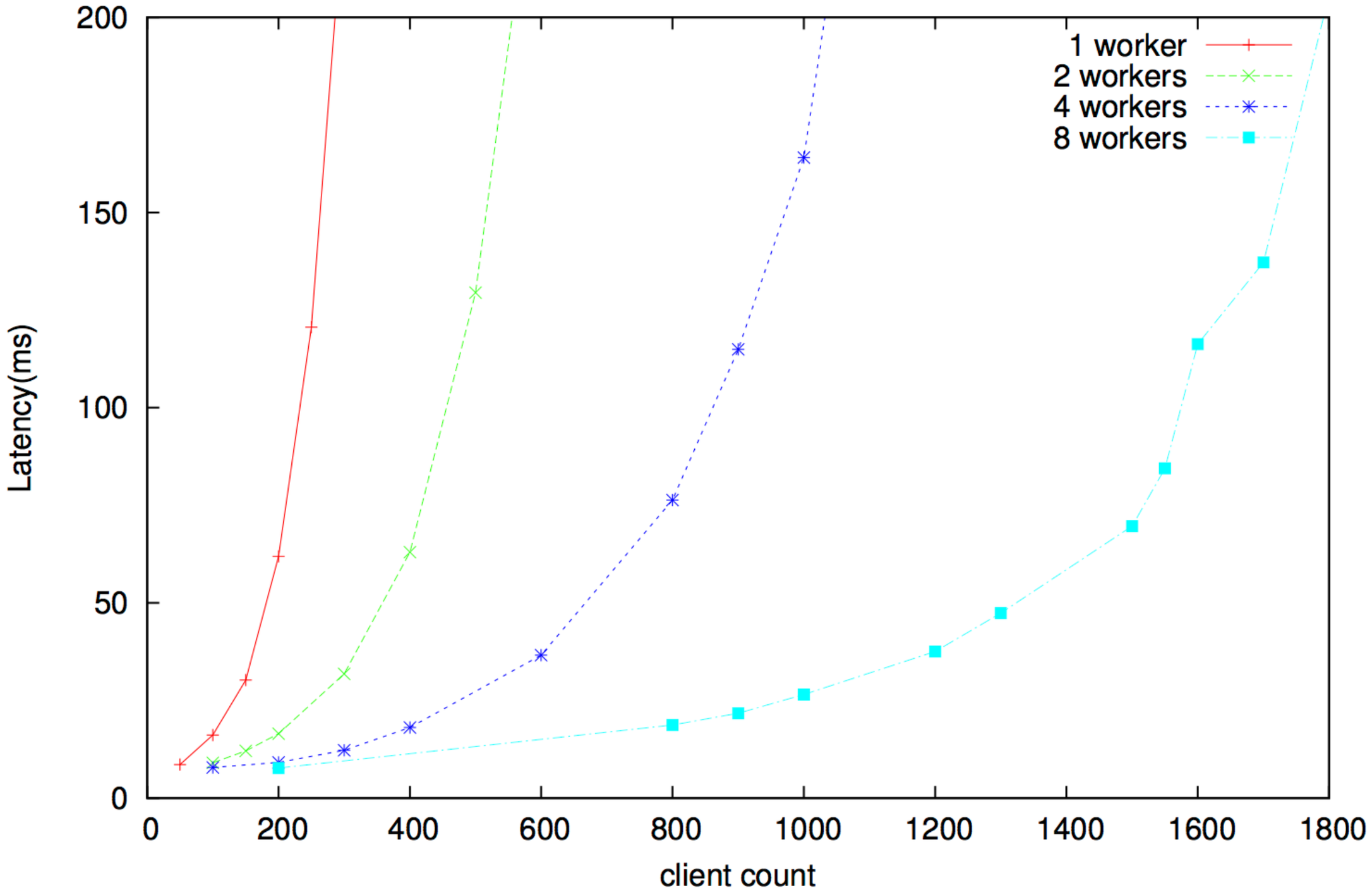
jQuery Chat Application

- 2,800 clients, 92ms latency



AngularJS 1.3 Chat Application

- 1,550 clients, 84ms latency



Evaluation — Summary

- Scales linearly with the number of cores on single machine
- CPU time and memory used by application code becomes bottleneck
- Master and reverse proxies are not bottleneck
- Use of high-level libraries is expensive
  - E.g. AngularJS 1.x “dirty-checking”
- Caveats: small applications

Related Work — Web Frameworks

- Too many to give credit to all, focus on those most related in terms of ideas
- Server-centric
  - ZK [Chen/Yeh 2005]
- Data-driven
  - Meteor [Meteor Dev Group 2011]

Related Work — Thin Clients

- VNC [Richardson 1998]
- X11 [Gettys/Scheifler 1986]
- VMWare Horizon
- Citrix XenDesktop

Related Work — RPC Frameworks

- Dynamic languages
  - Dnode [Halliday 2011] for Node.js
  - Pyro [Jong 1998] for Python
  - Druby [Seki 2012] for Ruby
- Static languages
  - Network objects [Birrell, et al. 1994] for Modula-3
  - Java RMI [Downing 1998] for Java
- Language independent
  - CORBA [Object Management Group 1991]

Conclusion

- CloudBrowser as a PaaS Service
- Scalable implementation of CloudBrowser 2.0
- Refined application deployment models
- Flexible node.js RPC framework : nodermi
  - Helped us leverage existing code base
  - Standalone framework, can be applied to other node.js projects
- Explored limitations of server-centric frameworks

Acknowledgements

- Supported by NSF Award
  CAREER: CCF-SHF Advanced Execution Environments for Next-generation Cloud Applications