

# **ultralight**client

**A pure Java RIA library**

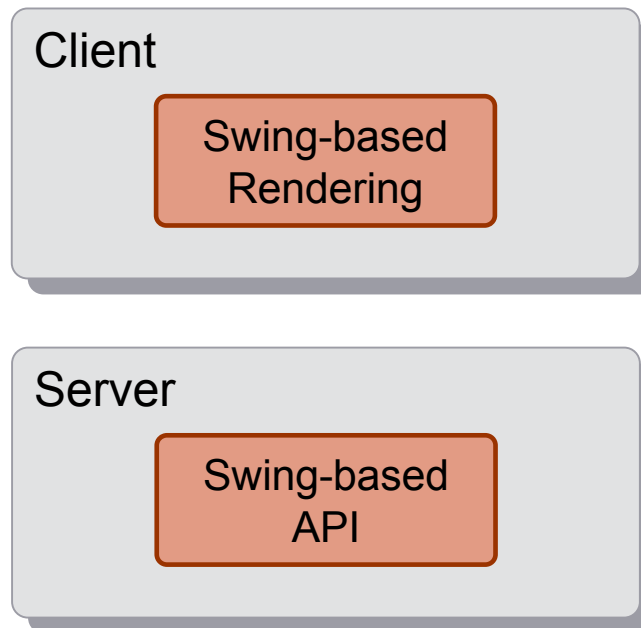
Etienne Studer  
Senior Java Software Developer  
Navis LLC, Oakland

# Outline

- Principles
- Architecture
- Programming Model
- Features & Optimizations
- Extensibility
- Q & A

# ULC in a Nutshell

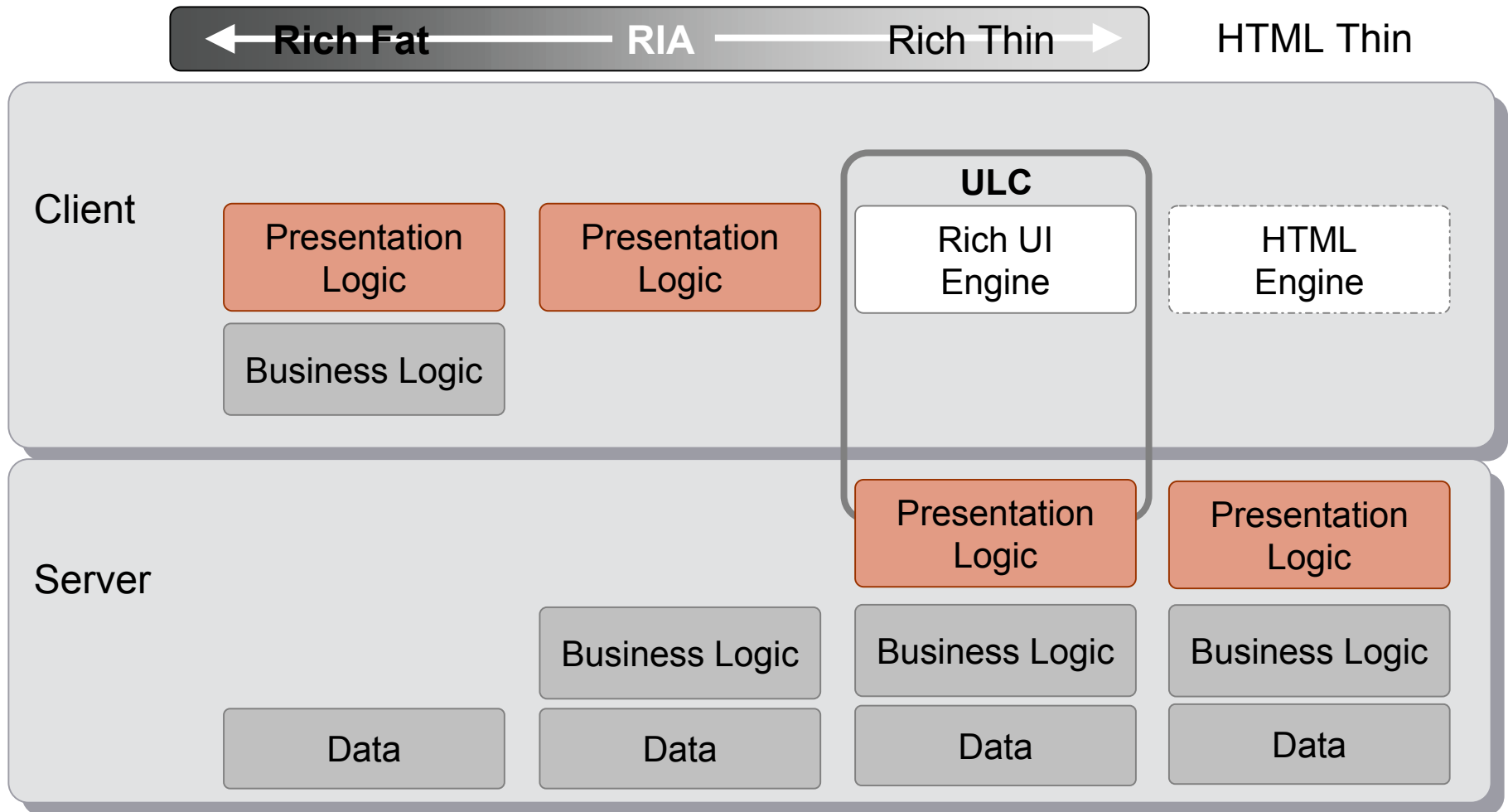
A UI component library for efficient development of Rich Internet Applications (RIA).



# ULC Principles

- Be rich
- Be thin
- Be efficient
- Be standard

# Architectures



# Web Architecture & Swing – Why?

- Leverage the power of Swing
  - Rich user interface features
  - Desktop integration
  - Coherent and proven programming model
- Protection of investment
  - Don't break the Web model
  - Web infrastructure
    - Application server, load balancing, security, proxies
  - Swing know-how and components
    - Leverage the Java standard

# Programming Model I

- Swing-like server-side programming model
  - Swing-like API
  - Component-oriented
  - Event-driven
  - Model-view-controller paradigm
- No distributed programming
- Automatic state synchronization between client & server
- Distributed garbage collection

**Non-trivial user interfaces:**

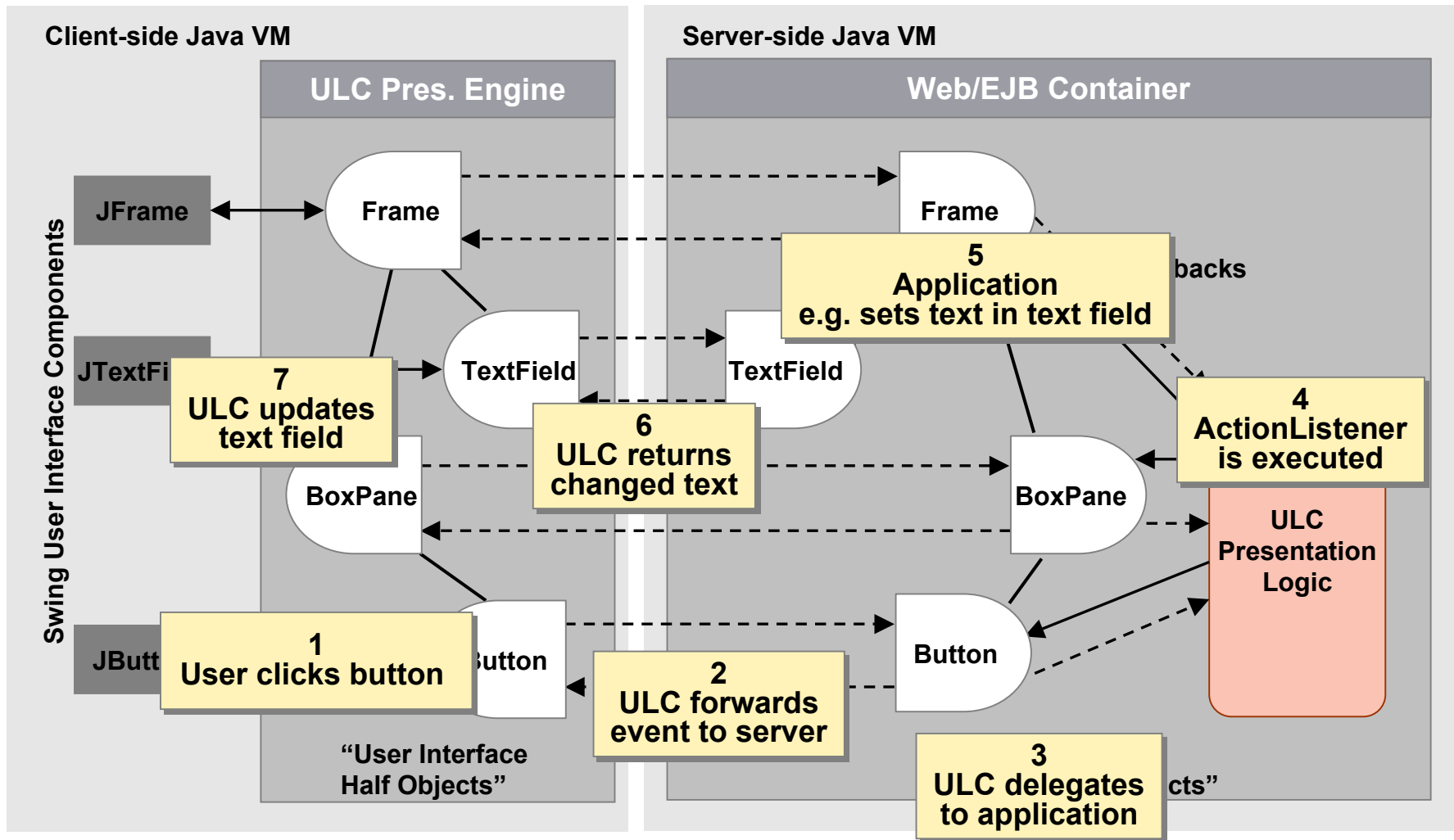
**→ considerably less effort compared to HTML**

# Programming Model II

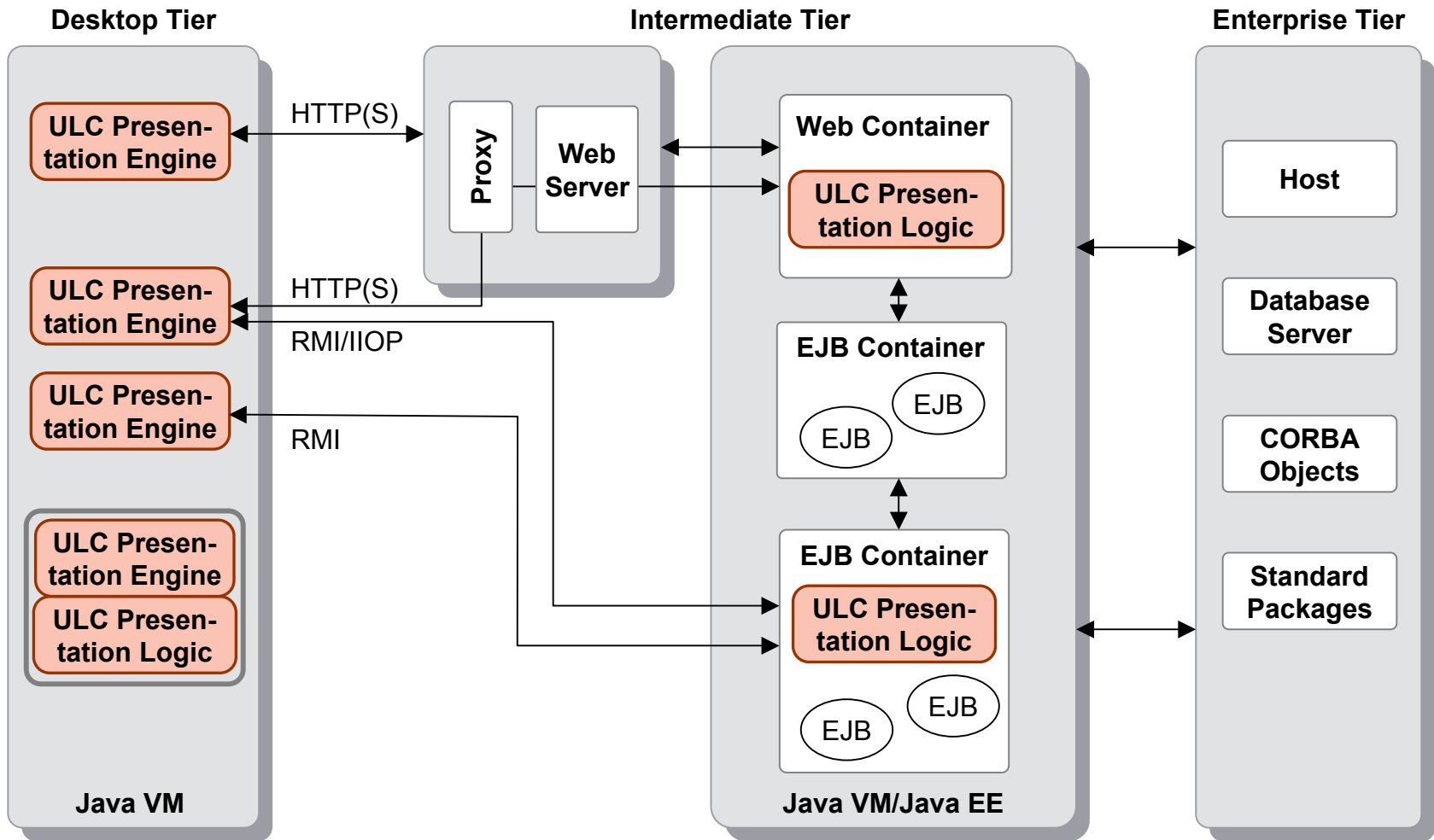
```
import com.ulcjava.base.*;

public class Hello extends AbstractApplication {
    public void start() {
        ULCFrame frame = new ULCFrame("Hello Sample");
        ULCButton button = new ULCButton("Hello");
        button.addActionListener(new IActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Hello");
            }
        });
        frame.add(button);
        frame.setDefaultCloseOperation(ULCFrame.TERMINATE_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

# Basic Idea of ULC – Half Object+Protocol



# Deployment Options



# Additional Features I

- Configurable event handling
  - Synchronous event handling (default)
  - Asynchronous event handling
  - Deferred mode
  - ULC always guarantees event sequence
- Adaptable communication
  - Communication protocol can be configured/replaced

# Additional Features II

- Client environment access
  - Locale, UI defaults, IP address, time zone, ...
  - File system access (read and write files)
  - Open documents with registered application
- Pause/Resume
  - Client can be restarted and set to the server-side state

# Additional Features III

- Server push
  - J2EE specification does not allow for server push
  - Polling timer can simulate server push
    - Lightweight and asynchronous
  - Alternative for highly accurate updates:
    - Open second channel to client to notify client about changes

# Optimizations I

- Low-level events
  - High-volume events are not suited for server-side handling
  - Translation into high-level semantic events (e.g mouse events)
  - Alternatively they may be mapped to state updates (e.g. window moved)
- Optional events
  - Only forwarded to the server if there is at least one listener
- Client state updates
  - Only forwarded to the server when necessary (i.e. with next event)
- Request bundling
  - Multiple requests are forwarded in a single client-server request
  - Communication controller waits until AWT event queue is empty

# Optimizations II

- Lazy Loading
  - Widgets and models are uploaded on demand
  - Model data is loaded asynchronously
- Caching and sharing
  - Components (widgets, models, etc.) are cached on the client side
  - Model data is transmitted and cached only once even when shared among multiple widgets
- Communication stream
  - Data is serialized and zip-encoded
- Formatters & validators
  - Allow for client-side syntactical validation
  - Example: ULCDateDataType checks for valid date

# Optimizations III

- Enablers
  - Highly interactive user interfaces update widget state accordingly
  - Enablers create a source-target relationship
  - Source: widget determines whether enabler is active
  - Target: widget that gets enabled/disabled depending on the enabler
  - Example: Button (target) gets enabled when table (source) has selection
  - Enablers can be combined using and/or/not

→ Optimizations reduce traffic by 90% compared to HTML

# Extensibility

- Even the most comprehensive widget sets are never good or complete enough
- Extensibility is as important as quality and completeness
- ULC provides an extension API
  - Simple and clear guidelines on how to
    - Extend existing widgets
    - “ULC-ify” off-the-shelf or customized Swing components
  - Infrastructure to simplify extension task
    - Marshalling
    - Remote method invocation

# Testing

- ULC is based on Swing
  - Reuse infrastructure for functional testing of Swing UIs
  - Adaptations for Jemmy and JFC Unit are available
- Pure Java and single JVM at development time offers another advantage for testing
  - Functional tests can peek into server side state to verify results
  - Example:
    - User interface offers only input, but test needs to verify output
    - Functional test accesses database to verify transaction

# Demo



**Q & A**