# Mifos Grails Frontend Spike

## Unfrak Your App With Grails

Adam Monsen

Grameen Foundation

March something, 2010

# Intro

- Mifos frontend spike using Grails application framework
- plan for incremental migration

# Hey, Adam

- start Eclipse, start WTP in debug mode

# Help me with this talk

- let's blitz through the slides and get to code
- interrupt me for egregious errors
- save talking points until later
  - please take notes
- this material will be used in a talk at LinuxFest Northwest
- your feedback is appreciated!

# Unimog

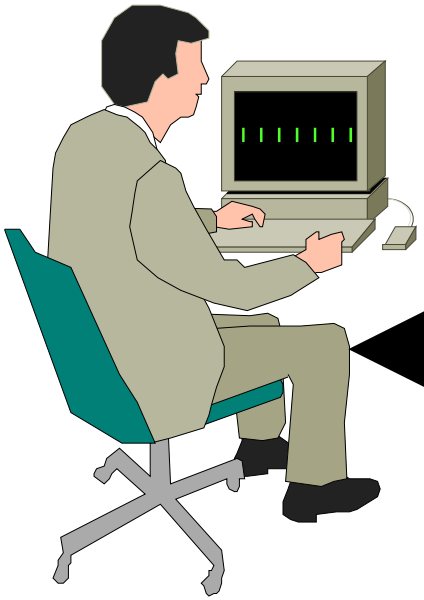
# Jargon

- "backend" is the existing Mifos, slightly modified so it can talk with a Grails frontend
- "frontend" is a new Grails-based application
- "API" describes how the backend and frontend will communicate

# Migration plan

- backend works as-is, but will redirect to frontend as each new service (and corresponding UI) is completed
- likewise, frontend will fallback to backend when necessary

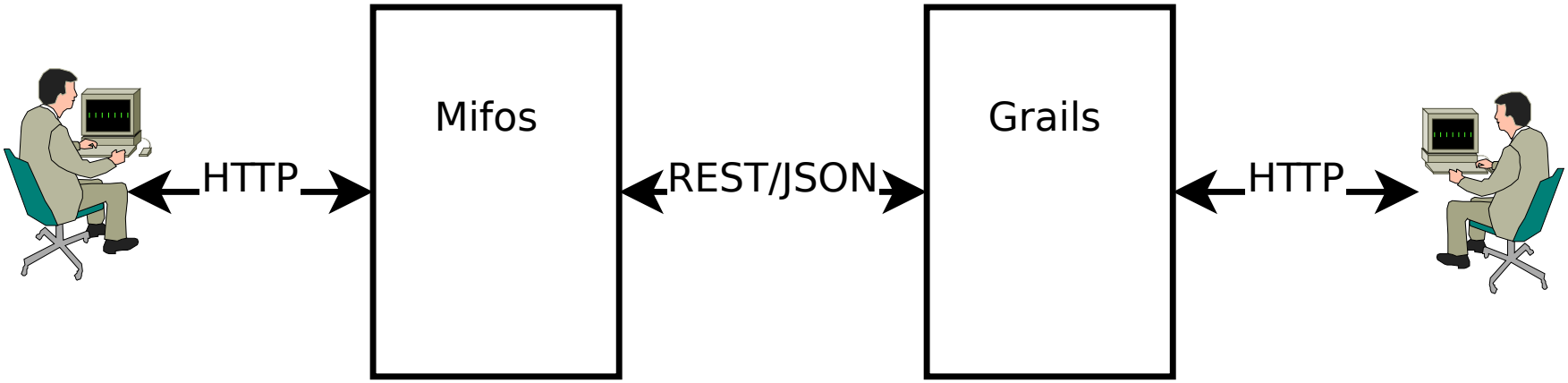
# Current architecture



HTTP

Mifos

# Interim architecture



# The API

- list clients
  - `http://localhost:8083/mifos/i/v1/clients`
- fetch a client
  - `http://localhost:8083/mifos/i/v1/client/3`

# New backend controller

- REST in, JSON out
- trunk patch
  - BackendBridgeController talks REST, JSON
  - main web.xml: give all of /i/* to Cheetah servlet (Spring MVC)
  - cheetah-servlet.xml: map /v1/* to new REST/JSON controller

# New frontend

- create grails app
- create controller in Grails
  - so far: one controller, two views
  - no persistence mapping (no domain)
  - install grails “rest” plugin
    - grails install-plugin rest

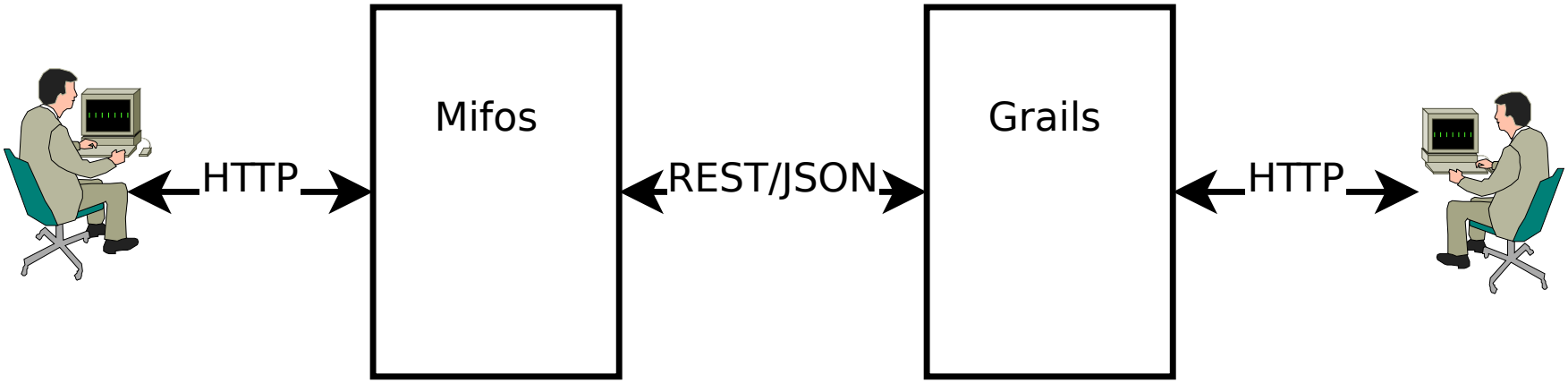
# Demonstration

- start grails app
- start Mifos in WTP (debug)
- hit `http://localhost:8080/patio/test`
- change client name, show change in Grails app
  - `http://localhost:8083/mifos/`

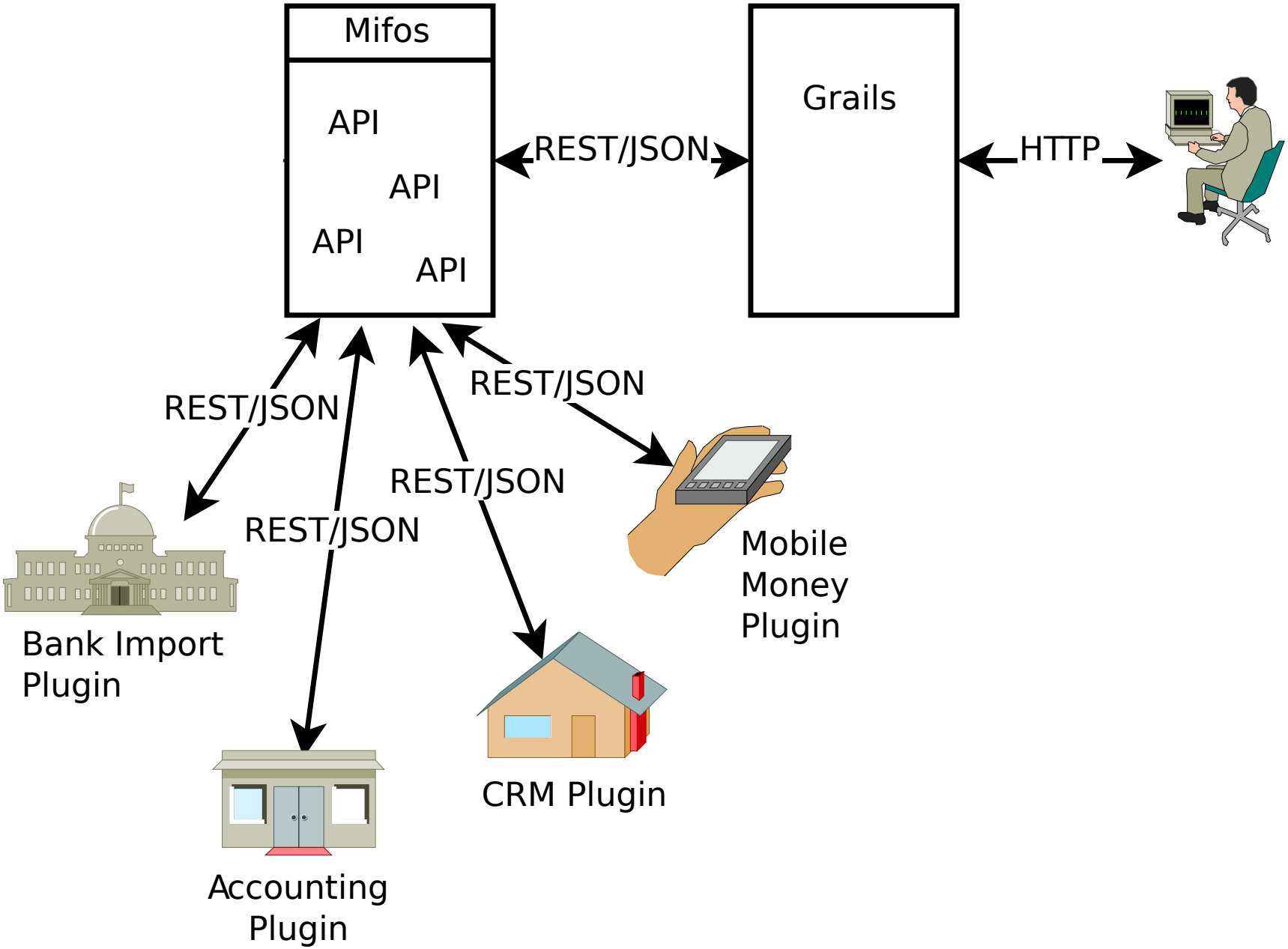
# Awesomeness

- Mifos web application becomes the Mifos *platform*
- rapid development with Grails
  - no configuration changes required for this demo
  - automatic reloading of all Grails classes and resources during development
  - many useful plugins
  - many useful conventions
- JSON
  - compact, efficient, yet readable
  - well-known standard (RFC 4627)

# Interim architecture



# Future architecture



# Questions?

# Potential anti-Awesomeness

- limitations of JSON, REST unknown
- need to keep tight reins on Grails app
  - Groovy is magic (sometimes *too* magic)

# Ideas

- Idea for efficient fetches from backend
  - high-performance REST
- Other random ideas
  - REST JSON best practices

# Security

- between frontend & backend
  - Basic HTTP auth
  - only allow local connections
- frontend
  - Spring (acegi) security
  - can defer to/delegate to/proxy existing backend security

# How do we keep from duplicating validation logic?

- add validation information to API
- don't validate: catch exceptions via the API
- provide JSON service to fetch configuration parameters from backend
  - ie: digits after decimal
  - Grails app can fetch/cache these on startup
- might have to duplicate which fields are required in HTML forms

# Further improvements

- annotations
  - clean, useful, complete documentation of API
    - generate from code/javadoc/annotations!
  - leverage annotations to generate API, too
- implement PUT/POST/DELETE, if helpful/necessary
- make a new Hibernate driver for REST API
  - would allow use of standard GORM conventions
  - backend could throw errors back in JSON, frontend can just display them
- performance
  - use a streaming JSON API instead of writing JSONObject.toString() to output stream
- Move BackendBridgeController into "api" module
  - move out of org.mifos package
  - Left in "application" module to leverage hot code replace
- factor out service-level code in Grails controller
- Spring MVC code
  - use a JSON Spring view(?)
    - wait, I'm getting pushed off the slide!