# HOB WEB Architecture Overview

Uriel Liu – Chief Architect of Wicresoft

# Revision History

| Version | Author | Description |
|---|---|---|
| 0.1 | Uriel Liu | Initial draft of the design |
| 0.2 | Uriel Liu | Add more information in the design consideration |
| 0.3 | Uriel Liu | Revise the prefetch part |
| 0.4 | Uriel Liu | Add bitbake helper to off load event cache in web server, move user authentication into web server |
| 0.5 | Uriel Liu | Elaborate more in design consideration |
| 0.6 | Uriel Liu | Add complete scenario |
| 0.7 | Uriel Liu | Add scenario to release bitbake server |
| | | |
| | | |

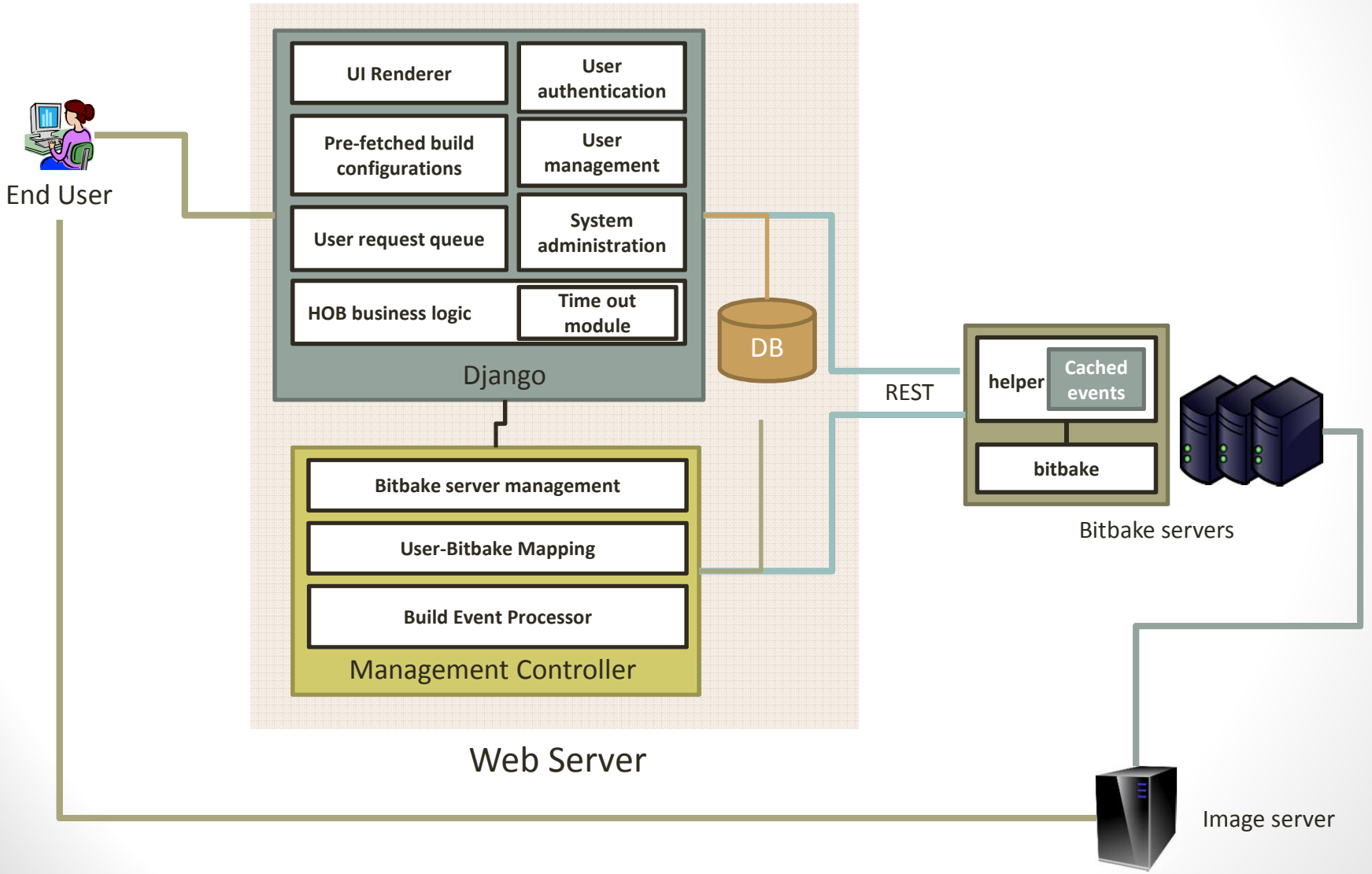
# Goal

- To provide a WEB UI interface for end user to trigger a build with Yocto
- Similar with HOB client, a user can select layers, platforms, recipes and packages
- User can select what's to be included in the final image among the built packages where the accurate accumulated image size can be known beforehand
- User can select among previous successful build configurations to start another build directly without going through the configuration again
- User's authentication is closely linked with session, so that user will be resume to the live build state when user's browser reconnects to HOB WEB site

# Design Consideration

- Users:
  - Two different roles – user and administrator
  - Use local DB to store user's successful build configuration
  - Keep track of user's build configuration at key steps from where the user left (maybe 5 entries at most as an example)
- Bitbake server management:
  - Dynamically addition/removal of the connection between web server and bitbake server
  - Assume bitbake server could only serve one requestor at a time to assure performance and responsiveness, therefore we need to enforce a strict single-entrance policy
- Web UI:
  - The status of building packages and building image will be reflected in WEB UI
  - Need a administrator UI to check the overall system status, add or remove bitbake servers, user managements or so
  - Pre-fetch all the possible build configuration of layers/machines/recipes/ in web server to provide better user experience (minimize the communication between Web and bitbake server
- Build:
  - How to utilize intermediate outputs to provide better UE via speeding up build process

# Architecture



# Django

- We will use Django as our web server
  - The user authentication and user management are natively provided by Django, we'll leverage this and maybe provide another consistent user management frontend for administrator to manage users
- User request queue
  - To queue the requests of users when all the active bitbake servers are occupied for earlier users (need more discussion: how to provide a good user experience for those waiting users?)
- HOB business logic
  - To mimic the entire HOB business logic, the timeout module will help to watch active user's idle time to make sure the occupied bibake could be released for other waiting users properly
- System administration
  - To provide an interface for administrator to manage the system including add/removal of bitbake servers, dashboard of bitbake server status and queued requests or so

# Database

- Store user's build configuration
  - To save the build configuration of previous successful builds
  - To save the configuration of last incomplete builds(eg. Network down unexpectedly, the user was distracted by other urgent thing) for user to easily access later
  - (Option and TBD) to save the build logs (or maybe saved in file formats)

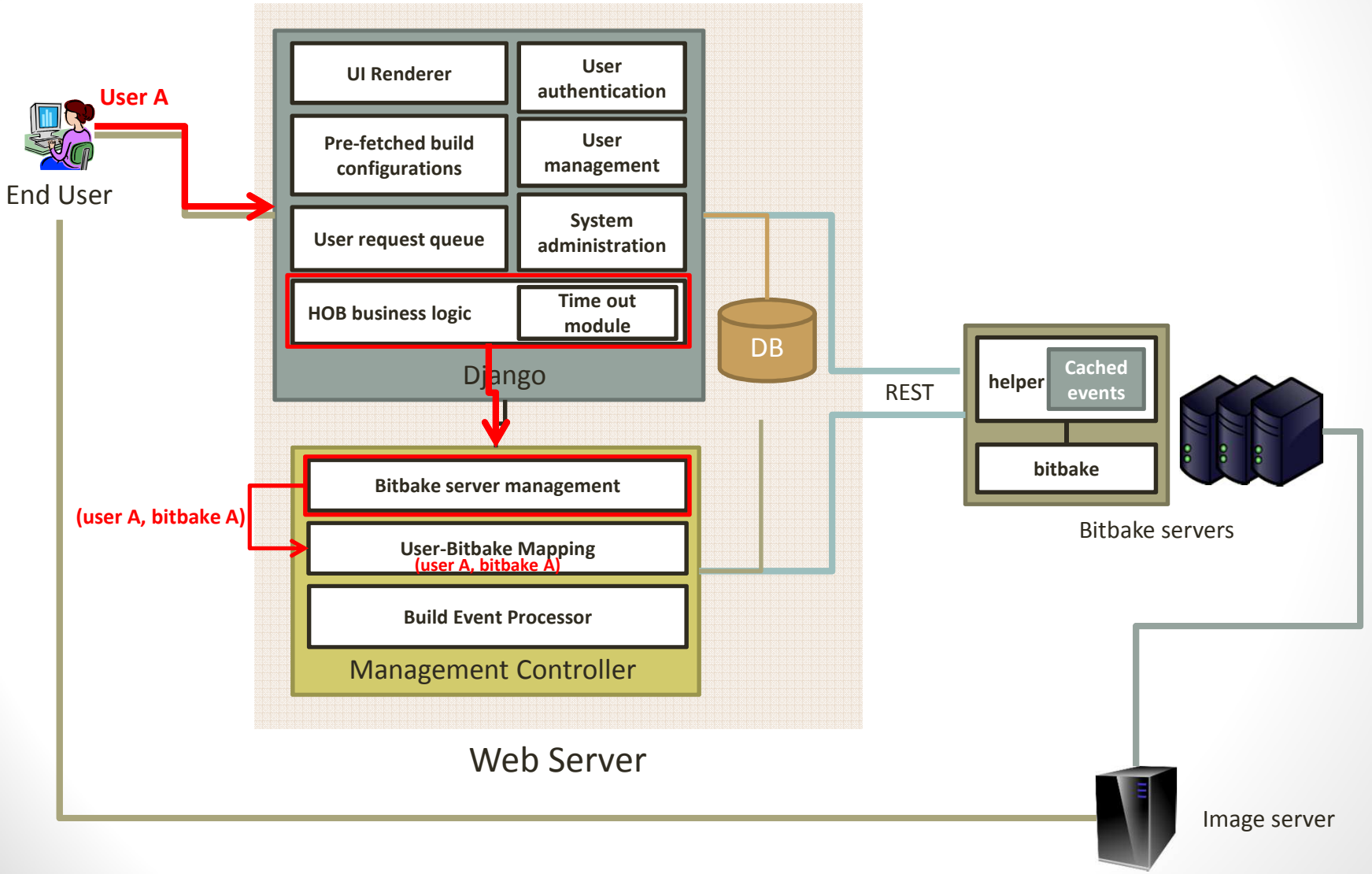
# Management controller

- The major management module to manage bitbake server status, user/bitbake server mapping and the others
- Bitbake server management
  - Administrator can dynamically add/remove a bitbake server
  - A health check thread will rountinely polling bitbake server to get the latest status
  - Server status could be [Available] [Reserved] [Down] and the others
- User-bitbake mapping
  - When a user logins the system, an available bitbake server will be reserved for him and we will keep the user/bitbake mapping information here
- Event Processer

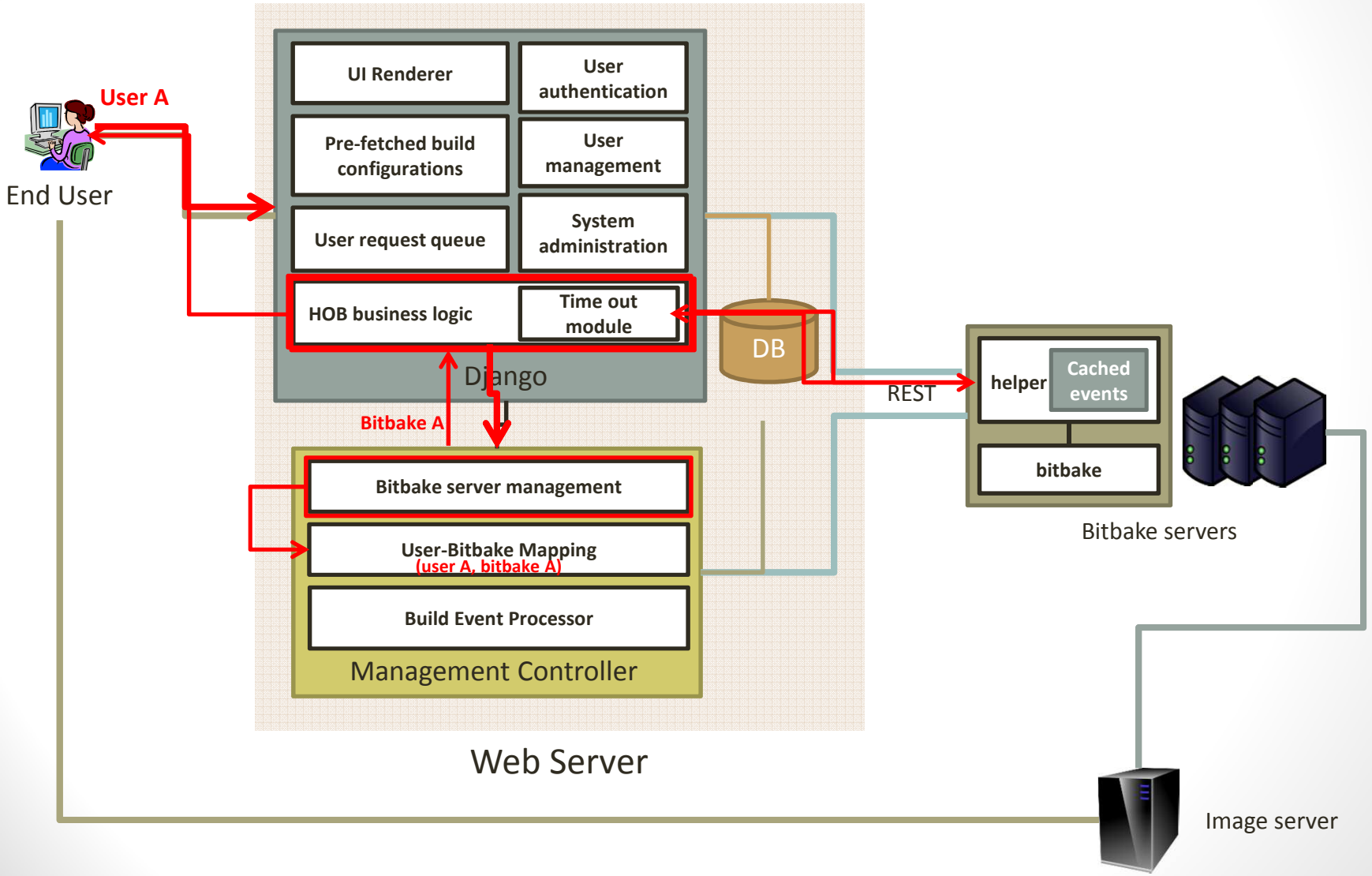
# Management controller – cont.

- Build Event Processer
  - When bitbake starts to build package or build image which takes longer time than the other steps, this module will periodically poll bitbake servers to get the cached events and then process them to the most valuable build status when user's browser tries to get the build status through Django(via AJAX).
  - The most valuable build status consists of 3 major parts:
    - Latest event (latest build event including build package, percentage and the others)
    - Major error (all the error that user concerns)
    - Past build progress (a summary telling how many packages/recipes are built or so)

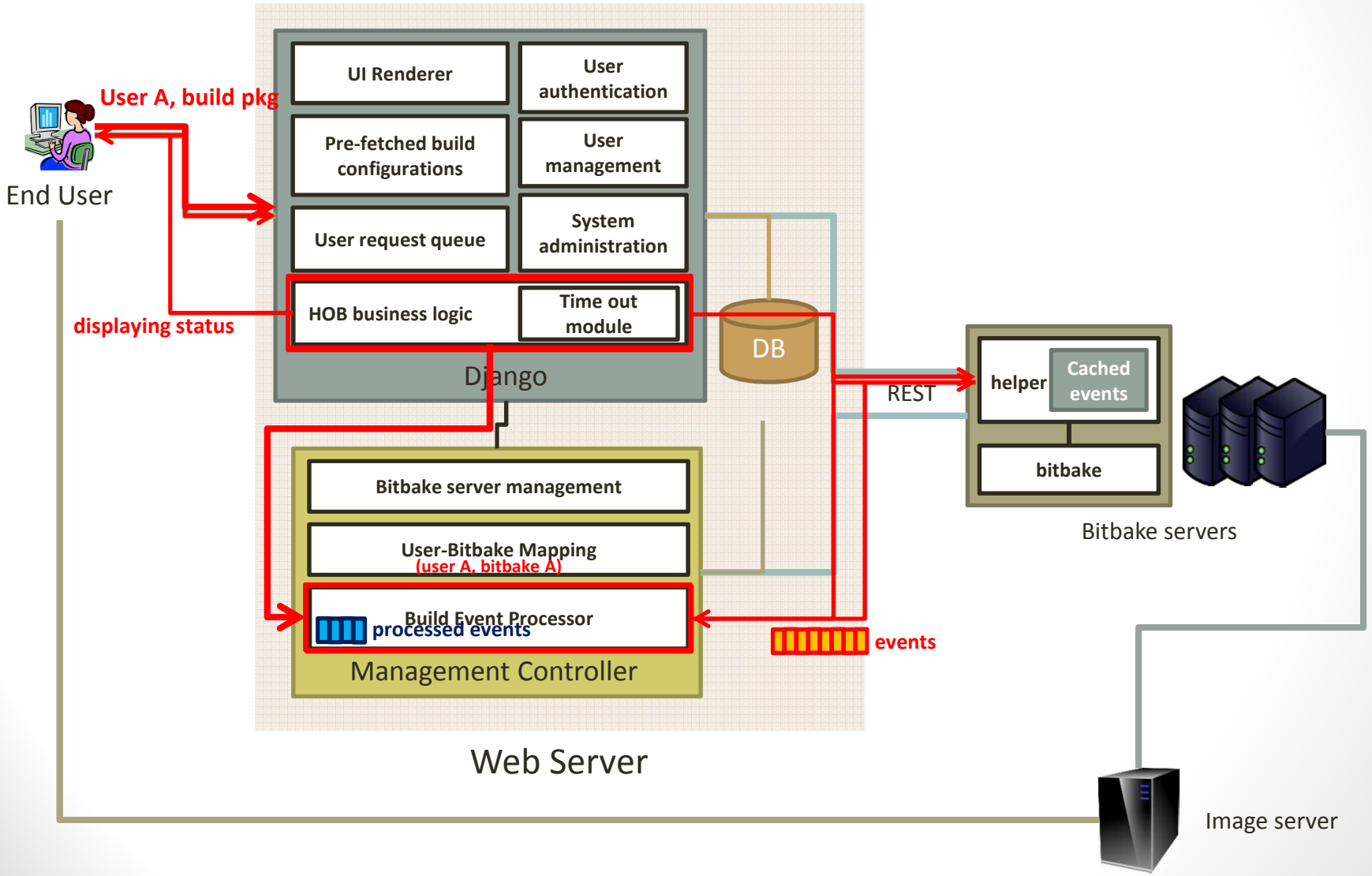
# A complete scenario – Reserve a bitbake server



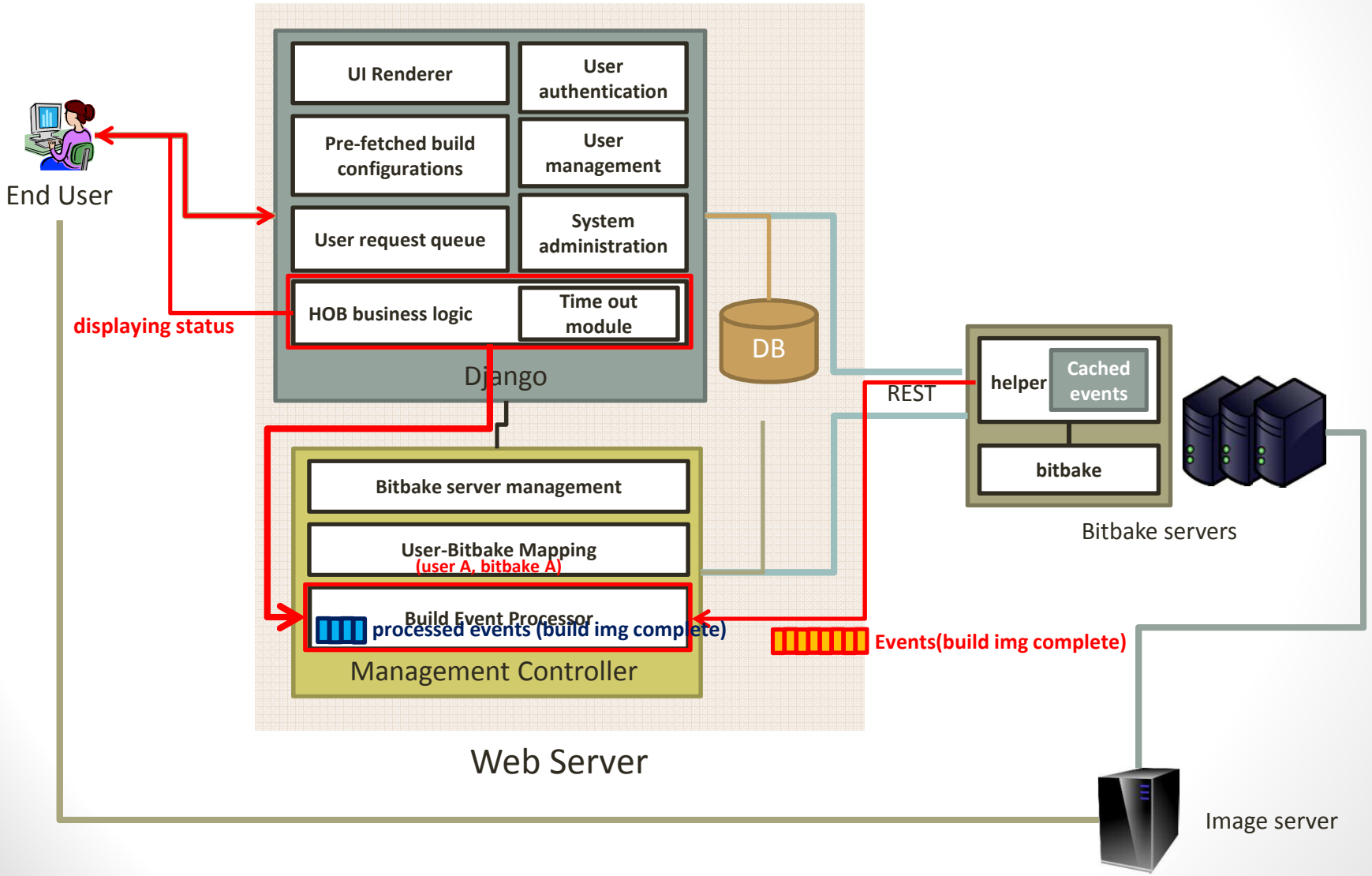
# A complete scenario – Normal flow



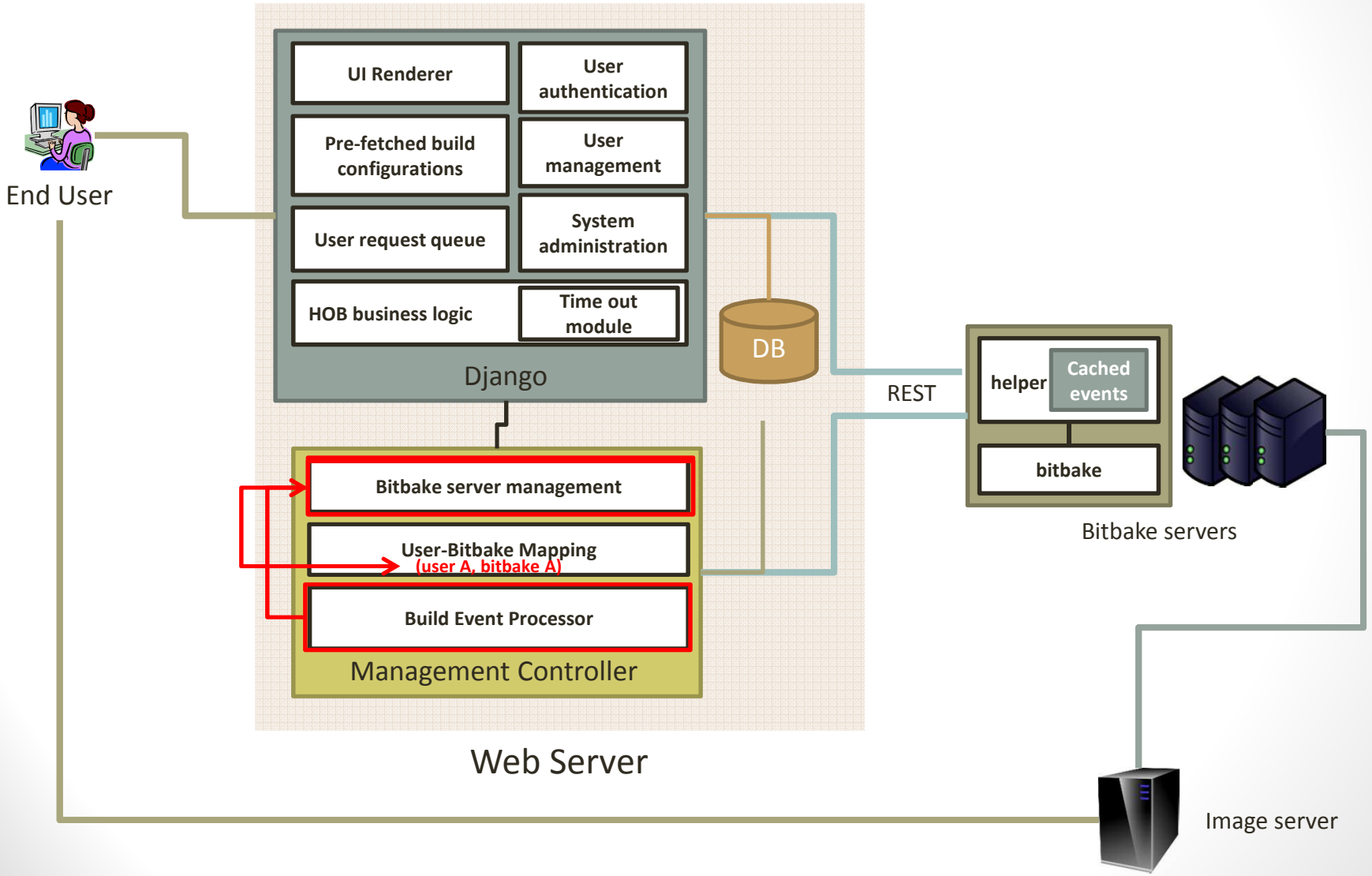
# A complete scenario – build package



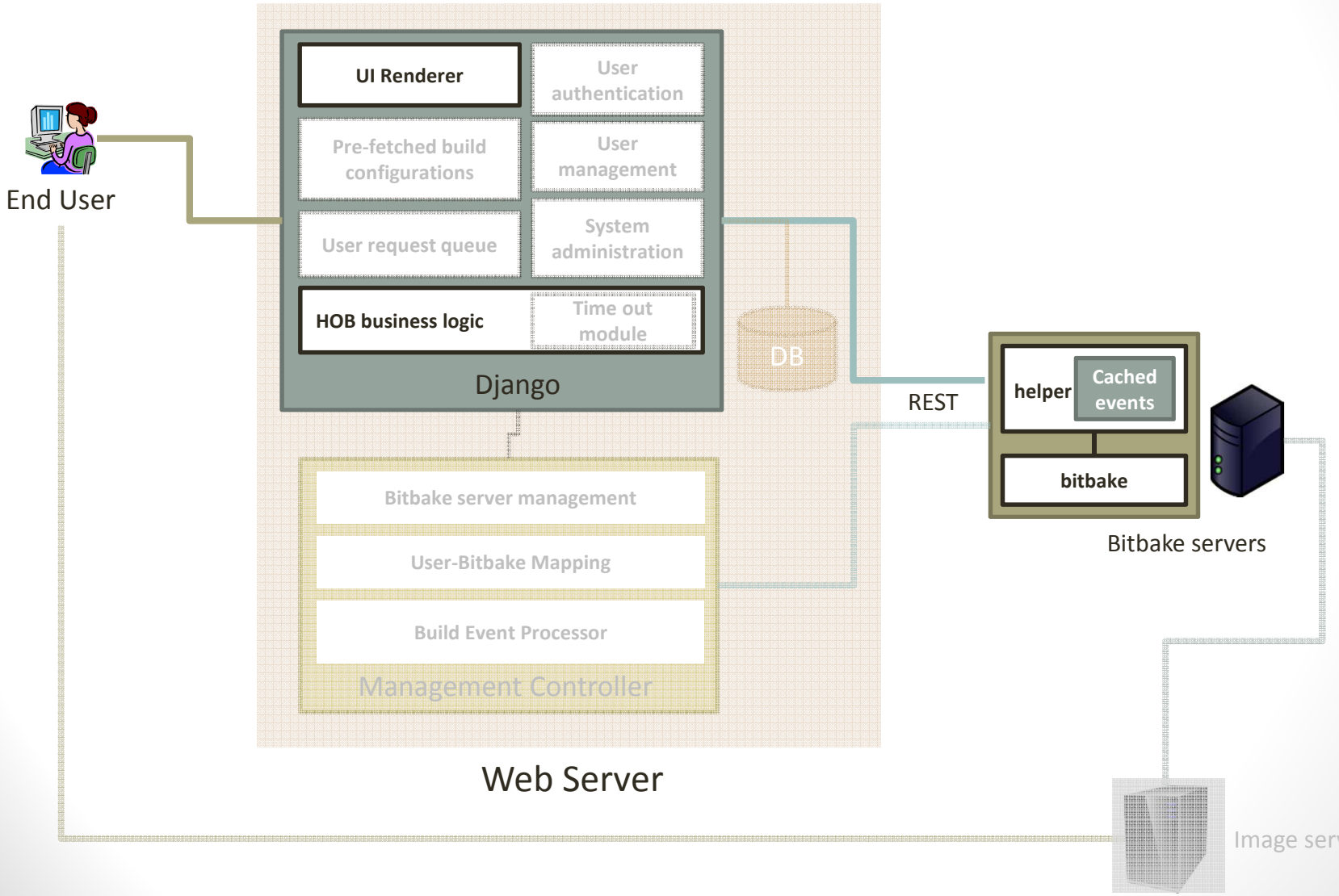
# A complete scenario – build complete



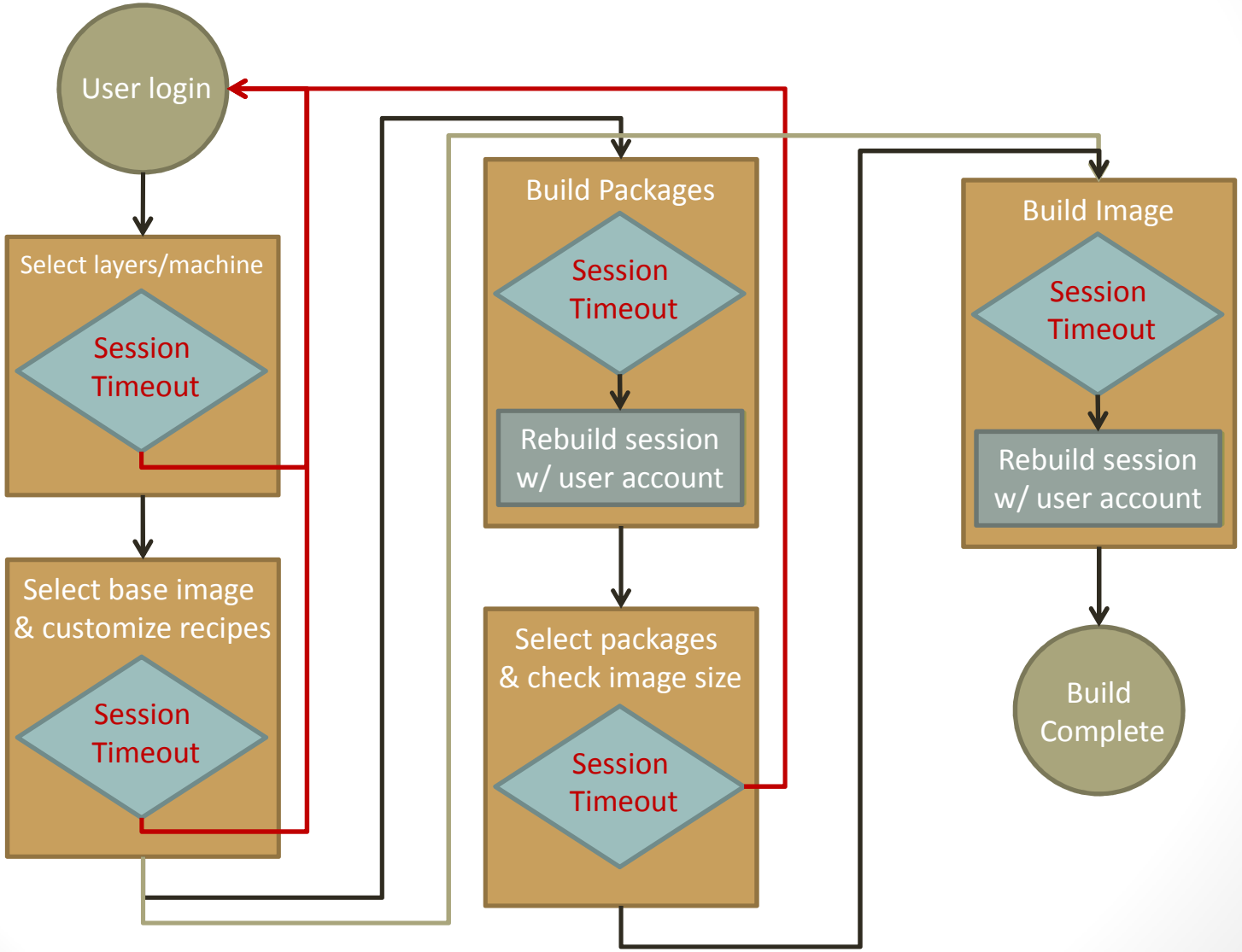
# A complete scenario – release bitbake server



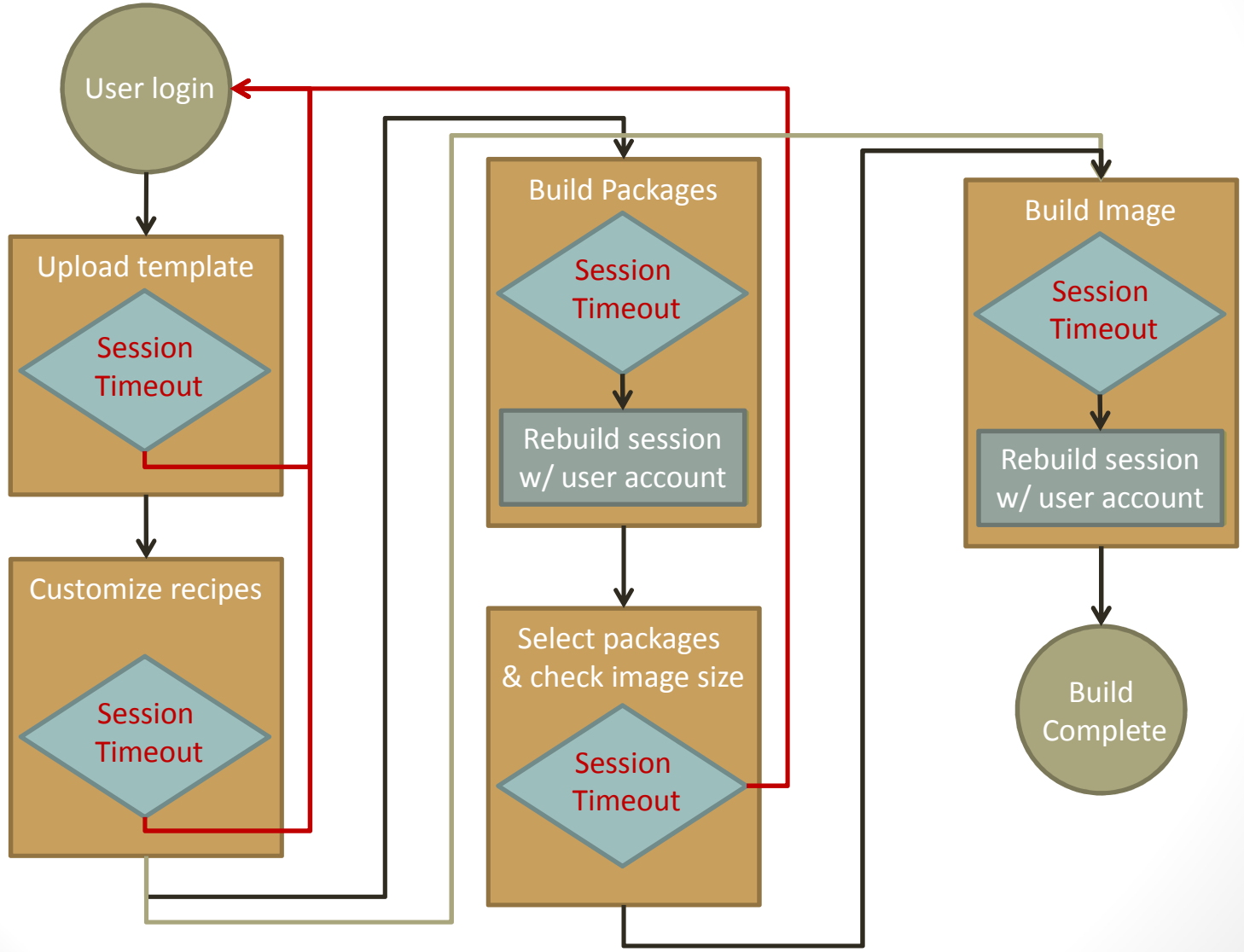
# POC goal (one user / one bitbake)



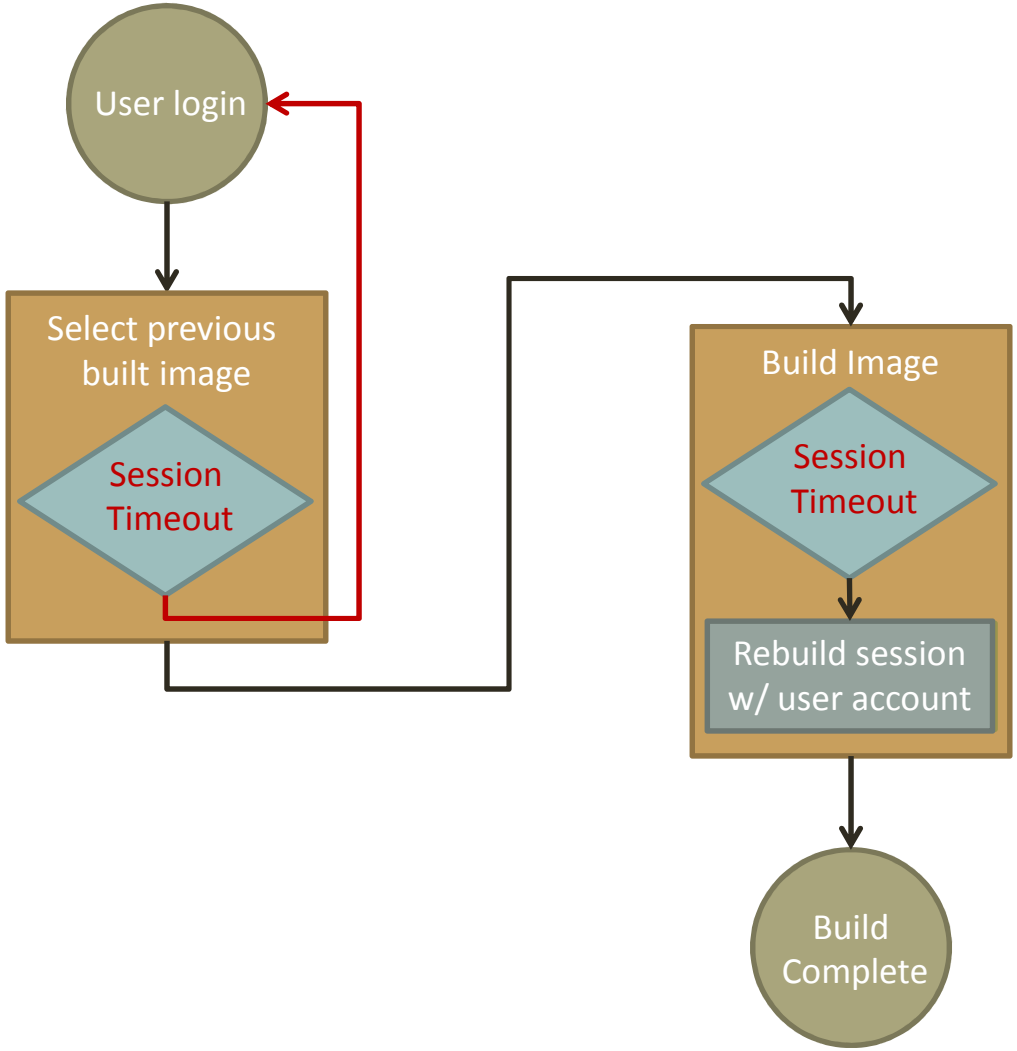
# User Interaction Flow (start from scratch)



# User Interaction Flow (start from template)



# User Interaction Flow (from previous build configuration)



# User State Transition