Distributed Password Cracking Platform

Dimitar Pavlov
Gerrie Veerman
UvA SNE Master Students
08-02-2012

***Supervisors*:**
Marc Smeets
Michiel van Veen

The project

- **Research Question**:

*How can a **scalable**, **modular** and **extensible** middleware solution be designed for the purposes of **password cracking**, so that it is based on **existing cracking tools** and allows for the use of a **dynamic** and **adjustable cracking strategy**?*

- **Why:** The need for a distributed password cracking system, which can work with both CPU and GPU capabilities
- **Approach**: -Formulate system requirements
  -Research and creation of system designs
  -Proof of Concept
- **Related Work**:
  - KPMG's previous research projects
  - Other work

Making the scope clear

- **What we did:**
  - Use existing cracking tools
  - Set requirements and make a distributed system design which is scalable, modular and extensible
  - Develop the basis for such a design

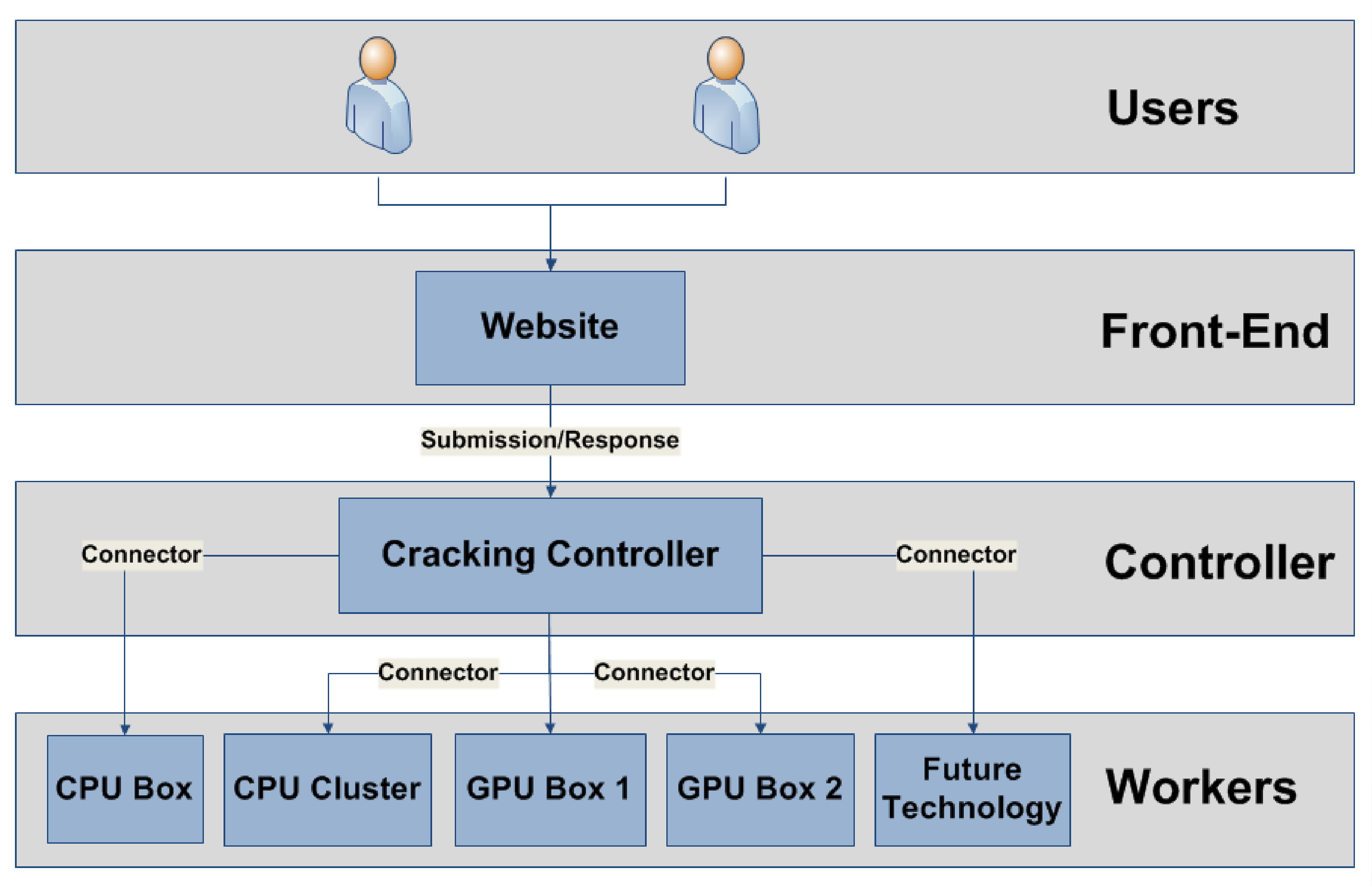
- **What we didn’t do:**
  - Create our own cracking tool
  - Design of cracking strategy

Research & Creation

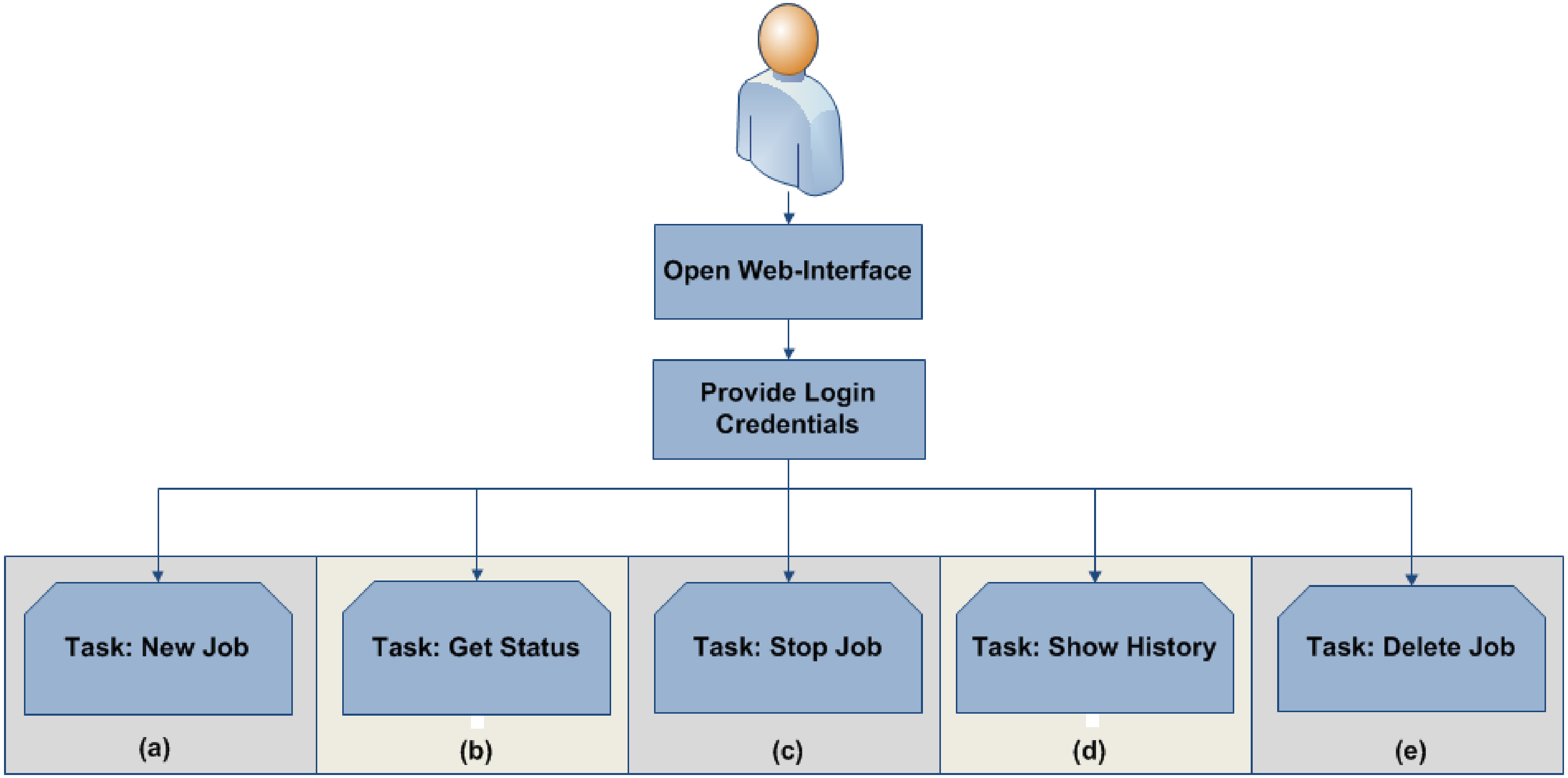
- **Distributed Systems**
  - Architectures
  - Communication
- **Cracking Tools**
  - CPU
  - GPU
  - Both

- **System Design**
  - Technical
  - Functional
- **Proof of concept**

System Overview



User Requirements



Open Web-Interface

Provide Login Credentials

Task: New Job

(a)

Task: Get Status

(b)

Task: Stop Job

(c)

Task: Show History

(d)

Task: Delete Job

(e)

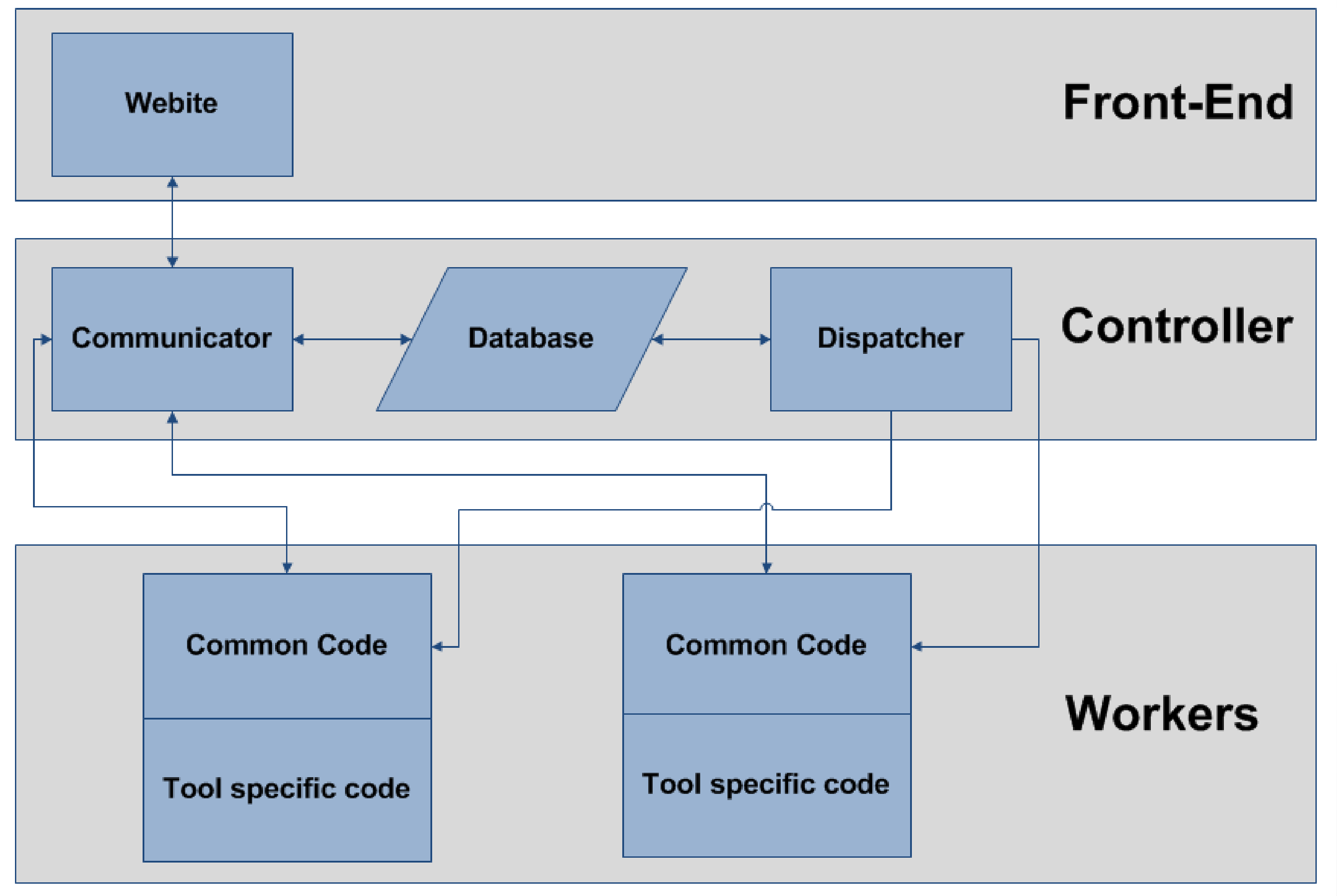
System Requirements

- **Front-end Functionality**
  - User Job Input
  - Current Job Status
  - Job History
  - Stop Job
  - Delete Job
- **Worker Functionality**
  - Register a controller
  - Status request handling
  - Job processing
  - Cracking tool support
- **Controller Functionality**
  - User input and request handling
  - Worker nodes control
  - Dynamic cracking strategy
  - User notifications

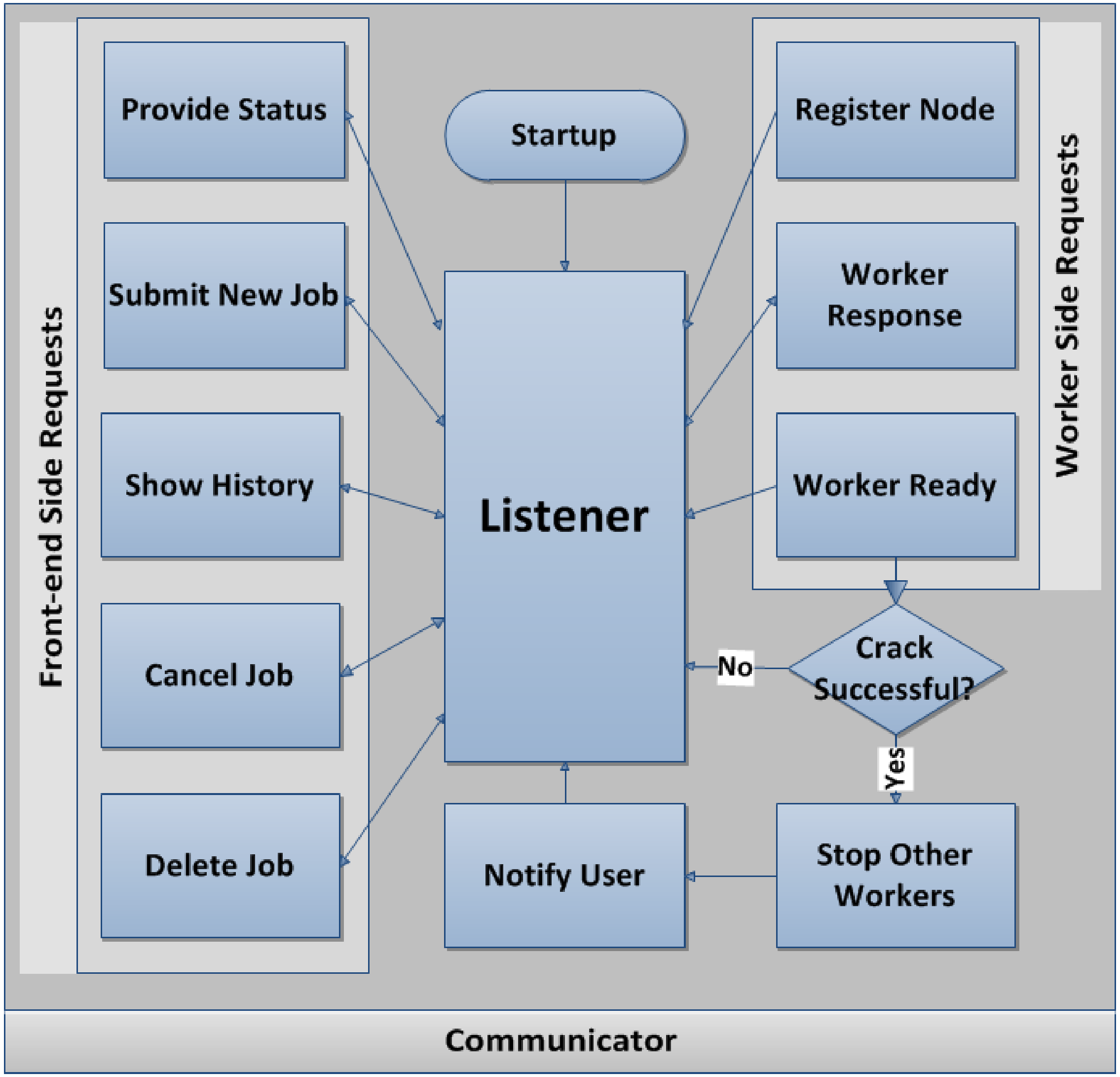
System Design

- System Architecture
- Communication
- Existing Cracking tools

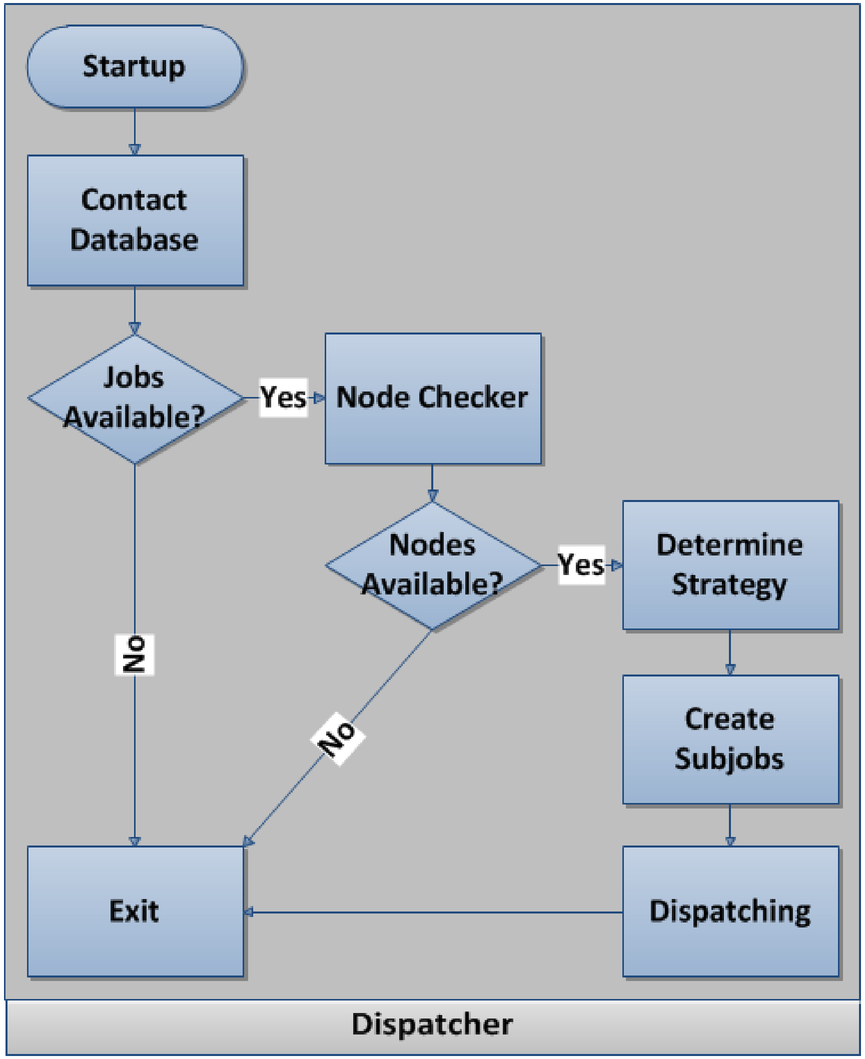
System Architecture Design



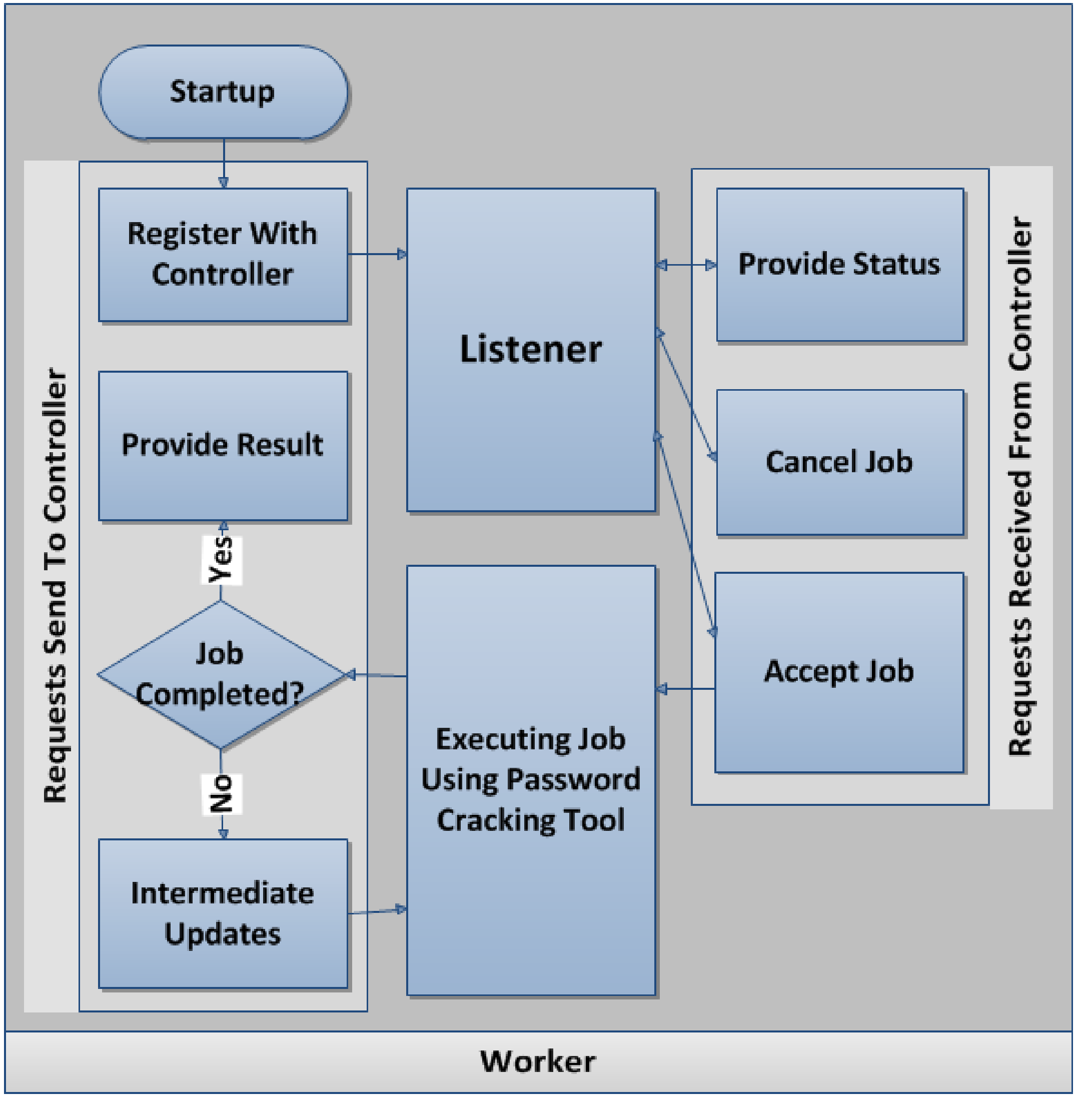
Communicator Workflow



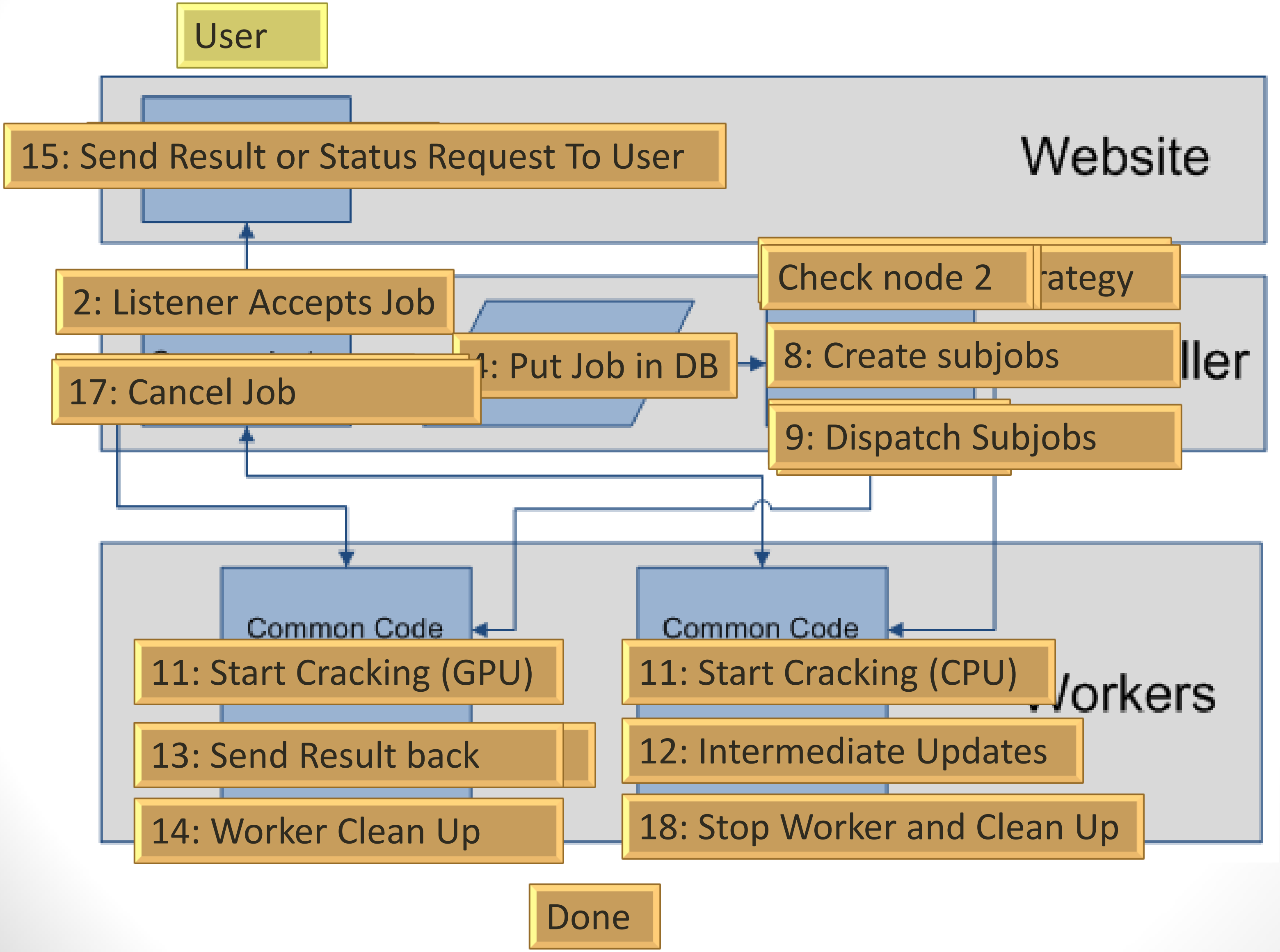
Dispatcher Workflow



Worker Node Workflow



Submitjob Example



User

15: Send Result or Status Request To User

Website

2: Listener Accepts Job

4: Put Job in DB

17: Cancel Job

Check node 2

rategy

8: Create subjobs

ller

9: Dispatch Subjobs

Common Code

Common Code

11: Start Cracking (GPU)

11: Start Cracking (CPU)

Workers

13: Send Result back

12: Intermediate Updates

14: Worker Clean Up

18: Stop Worker and Clean Up

Done

Communication

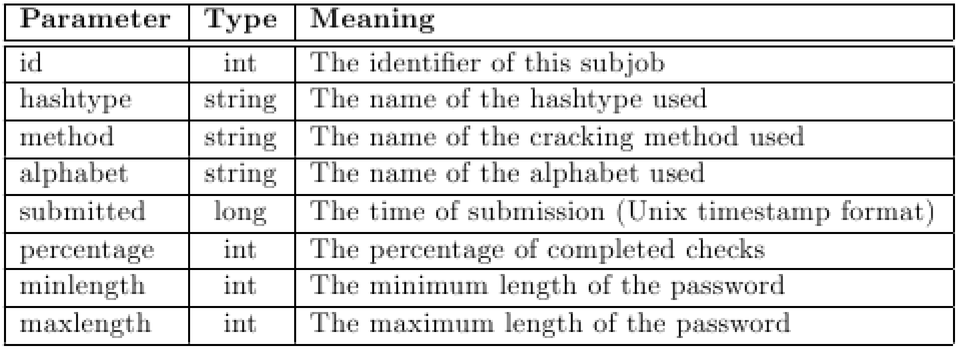
- **Paradigms**
  - Remote Procedure Calls (RPC)
  - Message-oriented communication
- **Protocol**
- **Data Structures**

Communication Messages & Data

- **Protocol**
  - Controller Messages – requestStatus, deleteJob, etc.
  - Worker Messages – requestStatus, stopJob
  - Asynchronous RPC – submitJob, sendResults

- **Data Structures**
  - Reply
  - Hash
  - Job
  - Subjob

Example: **Subjob data structure**



Parameter	Type	Meaning
id	int	The identifier of this subjob
hashtype	string	The name of the hashtype used
method	string	The name of the cracking method used
alphabet	string	The name of the alphabet used
submitted	long	The time of submission (Unix timestamp format)
percentage	int	The percentage of completed checks
minlength	int	The minimum length of the password
maxlength	int	The maximum length of the password

Cracking Tools

- **Existing cracking tools**
  - John the ripper (CPU)
  - oclHashcat-plus (GPU)

Proof of Concept - Overview

Component:	**Progress:**	**Used:**
Website		
Frond-end:	Very simple	<HTML>
Controller		
Communicator:	Finished	<PHP>
Dispatcher:	Very simple strategy	<PHP>
Worker		
Common code:	Finished	<PHP>
Tool specific:	Basic John the Ripper	<PHP>
Database		
Controller:	Finished	<MySQL>
Worker:	Finished	<SQLite>

Proof of Concept

- **Demonstration**
  1. Adding new node
  2. Show database with jobs
  3. Starting dispatcher
  4. Intermediate hashes cracked
  5. Job ready (result?)
  6. Worker Clean up / Ready again

Conclusion

- What was the research question again? ☺
  - How can a **scalable**, **modular** and **extensible** middleware solution be designed for the purposes of **password cracking**, so that it is based on **existing cracking tools** and allows for the use of a **dynamic** and **adjustable cracking strategy**?

- Research
  - **Distributed Architecture**: Centralized
    - Transparency
    - Modularity
    - Concurrency
    - Simplicity
  - **Communication**: Message-Oriented / RPC
  - **Existing Tools**: John the Ripper (CPU) / oclHashcat (GPU)

Project Achievements

- **Functional Specification**:
  - System overview
  - Use-cases
  - System requirements
- **Technical Specification**:
  - User interface
  - Controller
  - Worker
  - Database
  - Communication
- **Proof of Concept**:
  - **Website**: very simple
  - **Controller**: working with simple strategy
  - **Worker**: working with John the Ripper

Future work

- Further development / fine tuning of the system modules
- Extending to support other architectures (Cloud, Cell, etc.)
- Implementing the following for the system:
  - Adding more tools and hashtypes
  - Tweaking for multiple OS's (small changes needed)
  - Proper cracking strategy
  - Security for controller/node communication
  - Development of a proper front-end
- Testing / Benchmarking with many workers

Any Questions?