

# Tweet/Re-tweet App

Willie Aarnink

ASD User Event Presentation  
October 4, 2011

**verum**<sup>®</sup>

# Contents

- App description
- Architecture
- Decomposition
- Use of ASD
- Integration
- App build
  
- App demo

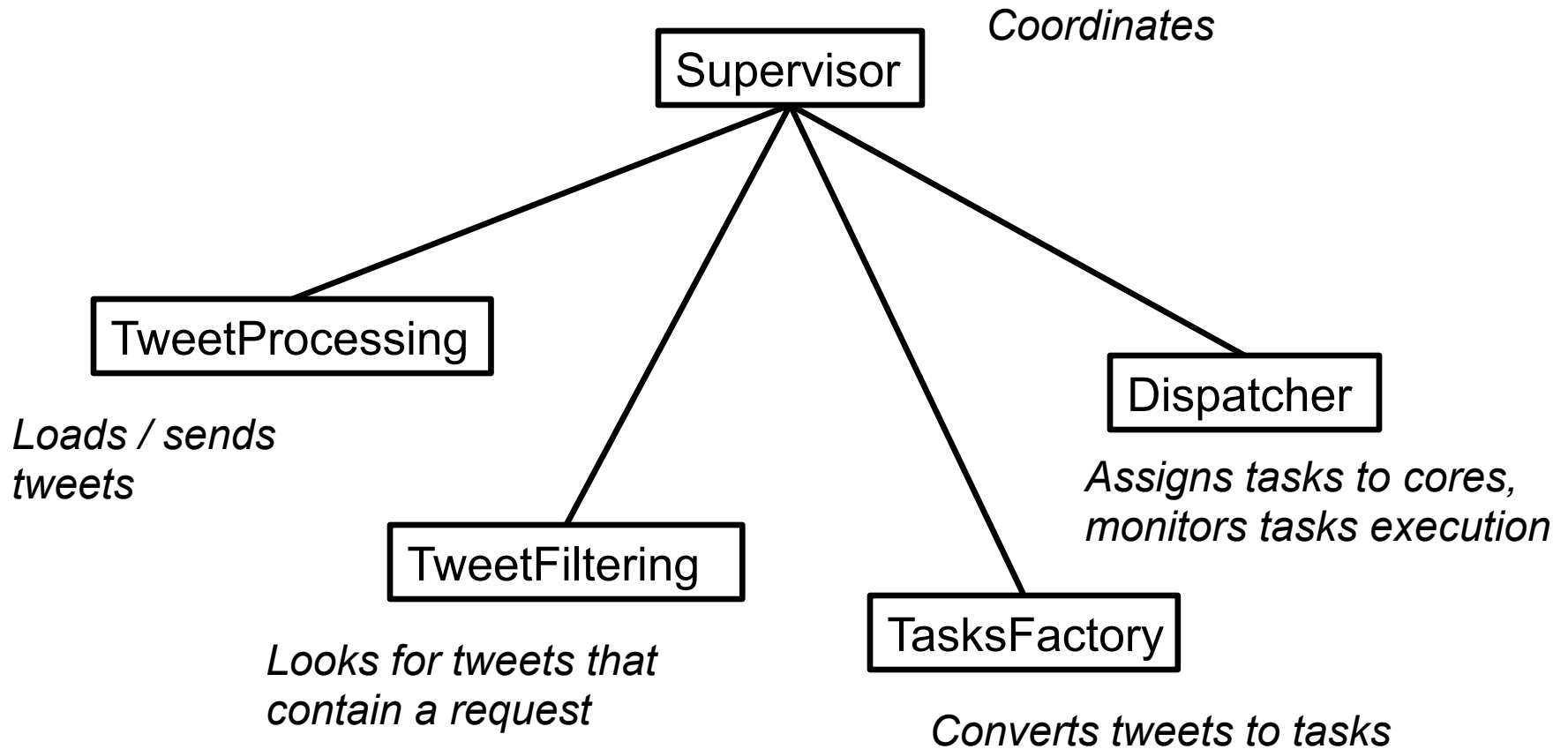
## App description

- Using a Twitter account, it listens for tweets that contain a request, e.g. 'Fibonacci(14)?'
- From such a tweet, it creates a task and distributes it over the cores in the system
- When the task has finished, it creates a tweet containing the answer, e.g. 'Fibonacci(14) = 377'
- It sends the tweet and deletes the task

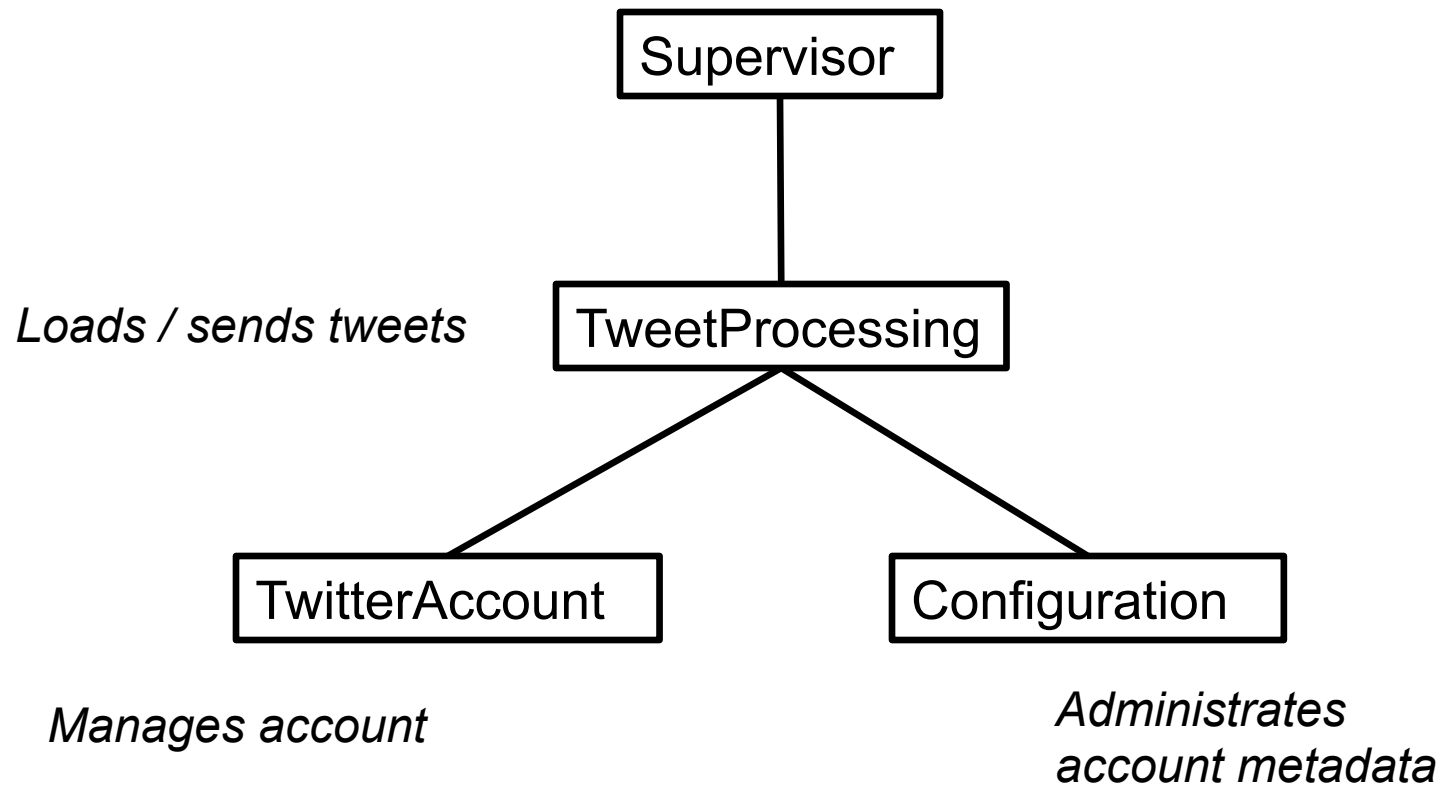
# Architecture

- The App has two main responsibilities:
  - Loading/sending tweets
  - Dispatching and monitoring tasks
- These responsibilities have been assigned to two main components
- Supported by other “helper” components
- Overall coordination is done by a supervisor component
- → yields a hierarchical controller pattern

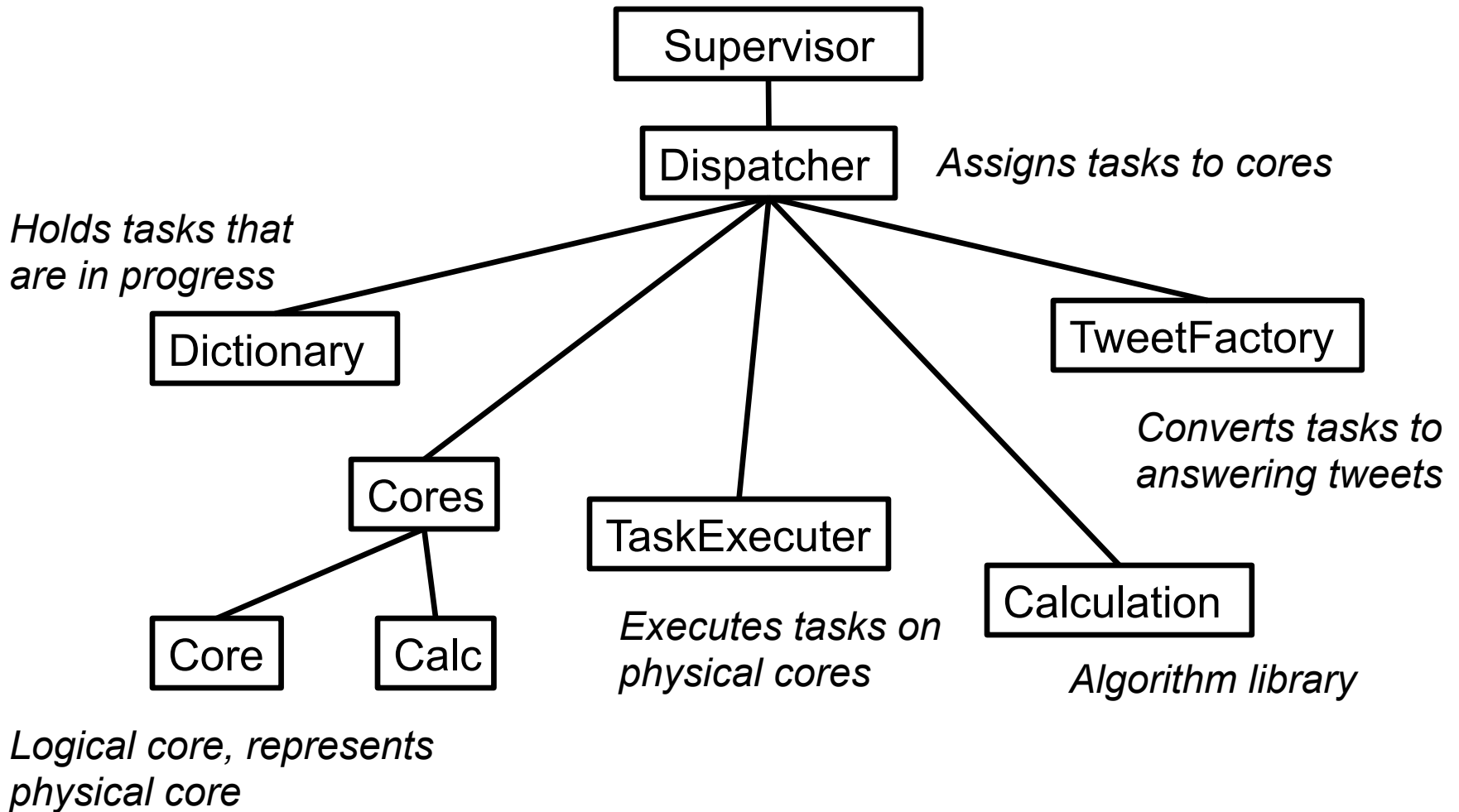
# Decomposition (1/3)



## Decomposition (2/3)



# Decomposition (3/3)



# Twitter

- <https://dev.twitter.com>
- Used the so-called REST API:
  - Access time lines (load tweets)
  - Status updates (send tweets)
  - ...
- Use OAuth for authentication: app need not know username / password of twitter account

## Loading / sending tweets

- Use 'Tweetsharp' library (c#, codeproject)
- Per call, max 20 tweets are loaded (the most recent ones)
- You have to work your way along timeline to load all tweets of interest (note: this logic was also modeled with ASD)
- You can only send one tweet per call
- Observe your rate limit (350/hour)

# Task

- In case a tweet contains a request, a task is created from it
- A task has state: received, processing, processed
- Task is perceived as data by the Dispatcher ASD component
- But still task has been modeled by ASD (!)
- When finished, an “answering” tweet is created from it. Task then is deleted

# Dispatcher

- Keeps a list of tasks in progress, in a dictionary
- Tries to allocate a free (logical) core
- Upon success, tries to find task to execute
- Starts any found task on the allocated (physical) core
- Checks for other, finished tasks
- Releases (logical) cores where finished tasks were executed
- Converts tasks to answering tweets
- In this way, max. 1 task is executing on 1 core

## Use of ASD

- Formal verification finds (exceptional) paths in your models you didn't think of, in advance
- Still, you must include these in your models
- Yield a **complete** model !
- But do we still need unit tests (TDD)? **Yes**:
  - handwritten code
  - Storage parameters
  - Did we model the right feature?

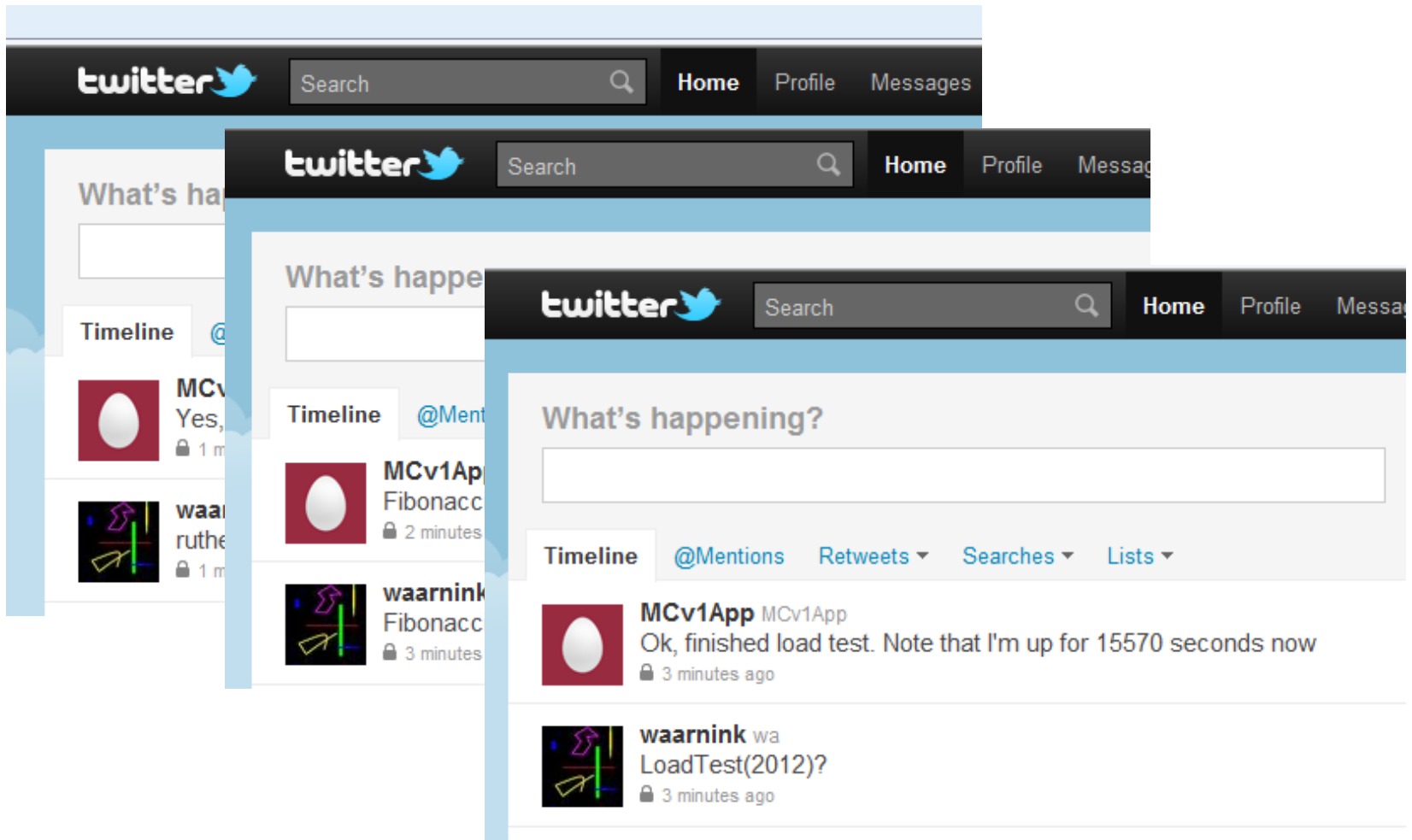
# Integration

- Generated c# code was integrated in VS2010
- Ndepend was used to determine KPI's:
  - Lines of code, CC, LCOMHS, Namespace Ca, Namespace Ce
- NCover was used to measure test coverage (tests made in Nunit)
- FxCop was used for static code analysis (use generated code attribute)
- No integration problems encountered.  
Preferred tooling can be applied

## App build

- App was build using VS2010 development environment (Professional edition, c# Express edition. Both worked Ok)
- 4 assemblies, 1 executable
- No problems expected when building using tools like FinalBuilder and InstallShield

# App demo (1/2)



## App demo (2/2)

- People who have a twitter account at hand, can take part in the demo:
- Request to follow MCv1App
- Valid requests:
  - Ruthere?
  - Fibonacci(8)?      Valid numbers: [0..92]
  - LoadTest(2001)?      Valid numbers: [0..50000]