Launching Activiti 6

June 10, Paris

Alfresco

Joram Barrez
& Tijs Rademakers

Activiti Leads, Alfresco

Alfresco

Defining moments of mankind (*)

(*) Random pick and highly subjective

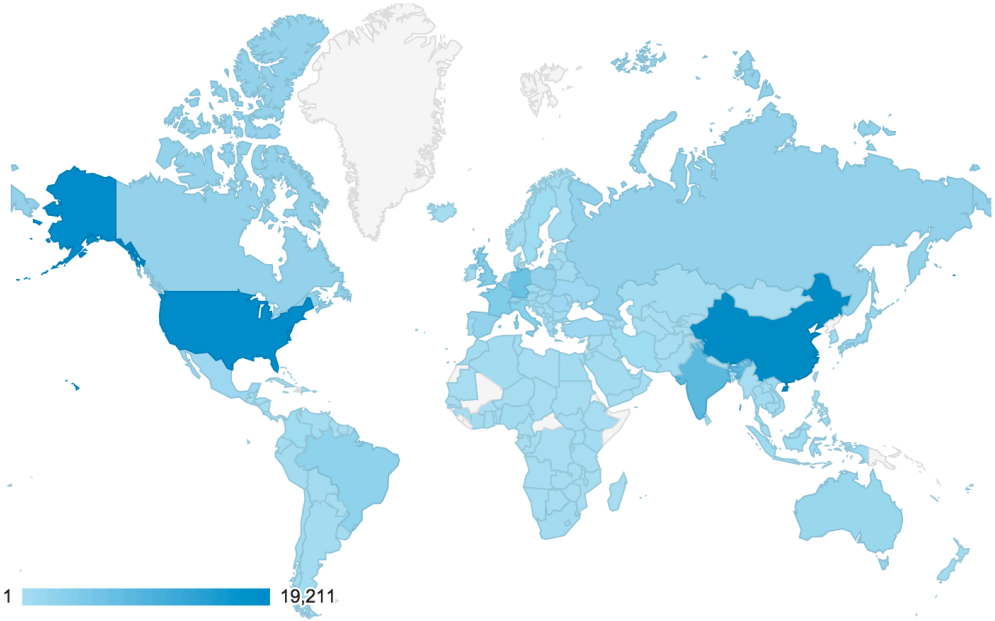
How did we get there?

Five years ago …

- 1 March 2010 : Joram & Tom join Alfresco
- 17 May 2010 : Alfresco launches Activiti 5.0.alpha1
- Huge interest from day 1

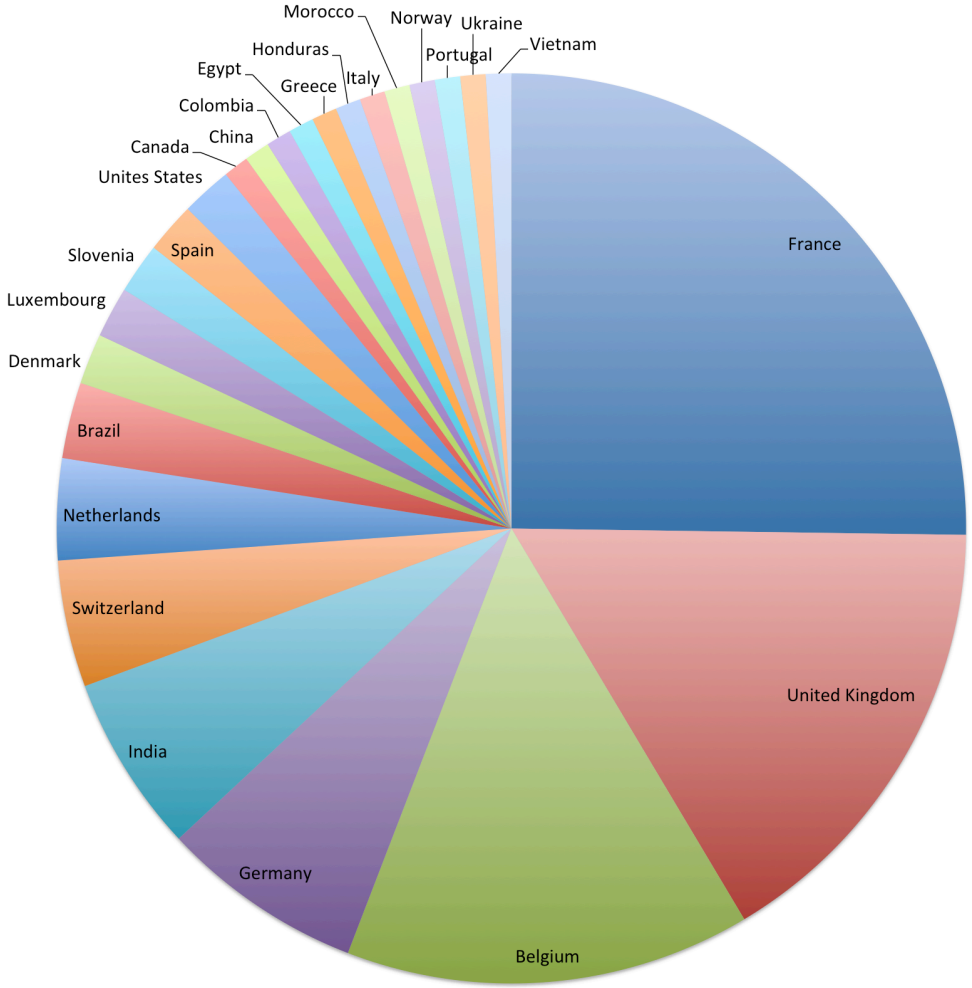
The past five years

- Huge uptake beyond anything we had imagined
- > 100 contributors
- Across the whole **globe**, all kinds of **scale** in all sorts of **industries**



Activiti.org stats:

- Displayed here: unique users Jan-May 2015
- Over 5 years: **900 000 unique visitors**



Attendees Today

Why Activiti 6?

The past five years

- The API still looks very much the same as five years ago
- The concepts have been kept stable
- The database schema evolved, but still looks very familiar
- The coverage of BPMN 2.0 has expanded
- The number of features/integrations has only grown

- Our core goal has always been stability
- Without comprimising in use cases

- +30 releases on this foundation

In the past five years …

- The codebase has grown *seriously*
  - By different people, with different focus/use cases
  - (of course, all code still has to pass the *gatekeepers, i.e. us*)

- We have learnt **a lot** on how Activiti is used (and sometimes how it is *mis*used/*ab*used)
  - Feedback through various open source channels
  - Feedback from running it at Alfresco's customers

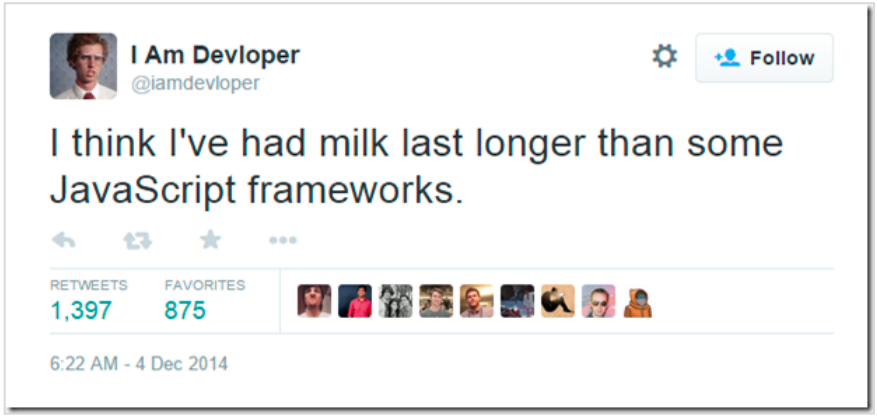
- Software architectures are changing
  - Lightweight, REST, container-less, reactive, …
  - With more and more data produced/processed

Five years ago …

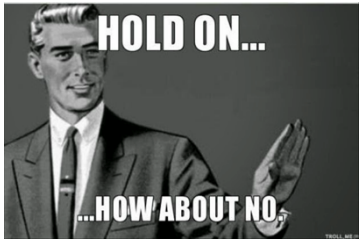
- We chose the right things:
  - Clean and easy to use API
  - Lightweight, embeddable engine
  - Standardized on BPMN 2.0
  - Stateless scaling
  - Many pluggability points
  - Clear docs (and a lot)
  - Liberal Apache license
  - Have a UI that demonstrates the features

- Today, all the above still applies (very much so!)

Five years is a long time in IT …



So are you saying Activiti 5 is … *bad*?



- Activiti v6 is **not a rewrite**
- Activiti v6 is **not a new engine**

- Activiti v6 is **an evolution** of the core engine, taking in account all we have learned in the past five years and what *we (*)* believe that is needed to cope with the use cases of 2020 in the simplest and cleanest way.

- After five years of we were really tired of that '5'

(*) the Activiti *community*

“Making sure we are ready for the next five years”

Did Activiti v6 just drop out of the sky?

https://github.com/jbarrez/activiti-neo4j

Neo4j

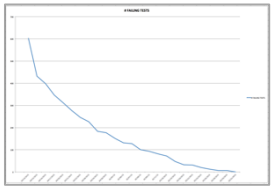
July 2013

Jan 2014

Alfresco

(commercial offering)

May 2015



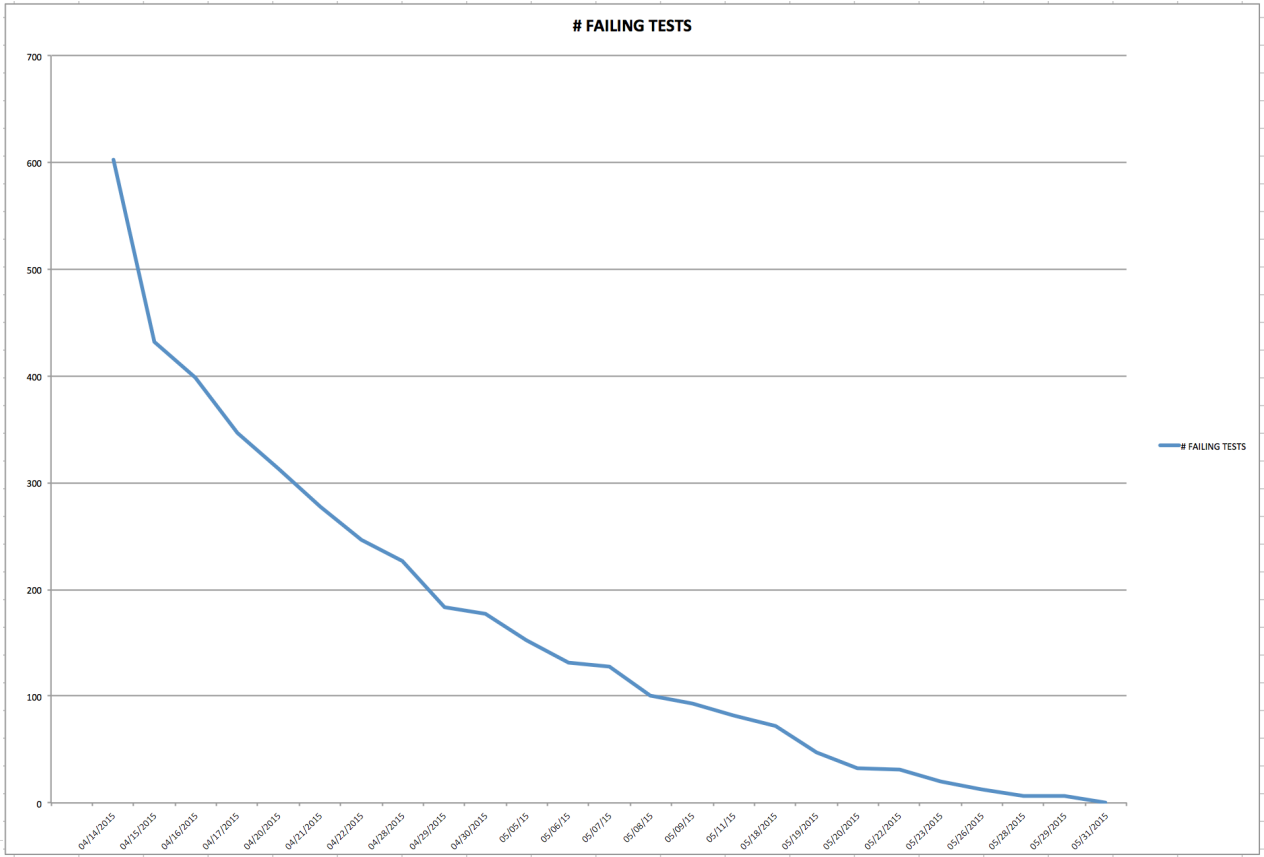
LAUNCH

Nov 2013

Dec 2014

V6 works starts 'officially'
On https://github.com/Activiti/Activiti/commits/experimental

Just because we're really proud of this graph



Activiti v6 Core engine evolution design goals

- Backwards compatibility with Activiti v5
  - Crucial and non-negotiable
  - API, DB Schema, concepts, integrations, …
- Simple(r) runtime structure and execution
- Removal of bloated parts of PVM within the core engine
  - ActivityImpl, ExecutionImpl,. …
  - Direct execution of Bpmn Model
  - Straight algorithm for defining runtime structure
- Dynamic process instance/definition modification support
- Performance, performance, performance

Activiti 5 Problems

V5 Problem 1 – Model Transformation

- BPMN 2.0 XML -> Internal model (PVM datastructures) -> PVM operations
  - ActivityImpl, ExecutionImpl, …
  - not 1-1 with BPMN concepts
    - Leakage between layers
  - Example (from the userguide):

```
1  public class ThrowsExceptionBehavior implements ActivityBehavior {
2
3    public void execute(ActivityExecution execution) throws Exception {
4      String var = (String) execution.getVariable("var");
5
6      PvmTransition transition = null;
7      try {
8        executeLogic(var);
9        transition = execution.getActivity().findOutgoingTransition("no-exception");
10     } catch (Exception e) {
11       transition = execution.getActivity().findOutgoingTransition("exception");
12     }
13     execution.take(transition);
14   }
15
16 }
```

V5 Problem 1 – Model Transformation

- V6 version:

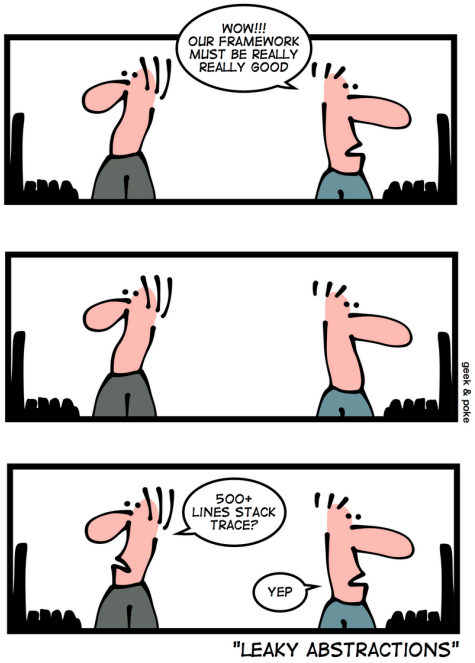
```
public class ThrowsExceptionBehavior implements ActivityBehavior {

  public void execute(ActivityExecution execution) {
    String var = (String) execution.getVariable("var");

    SequenceFlow sequenceFlow = null;
    try {
      executeLogic(var);
      sequenceFlow = findSequenceFlow(execution, "no-exception");
    } catch (Exception e) {
      sequenceFlow = findSequenceFlow(execution, "exception");
    }

    execution.setCurrentFlowElement(sequenceFlow);
    Context.getCommandContext().getAgenda().planContinueProcessOperation(execution);
  }

  protected SequenceFlow findSequenceFlow(ActivityExecution execution, String sequenceFlowId) {
    FlowNode currentFlowNode = (FlowNode) execution.getCurrentFlowElement();
    for (SequenceFlow sequenceFlow : currentFlowNode.getOutgoingFlows()) {
      if (sequenceFlow.getId() != null && sequenceFlow.getId().equals(sequenceFlowId)) {
        return sequenceFlow;
      }
    }
    return null;
  }
```

- (Sequence flow lookup can be simpler, just wanted to show the nuts and bolts)

V5 Problem 1 – Model Transformation

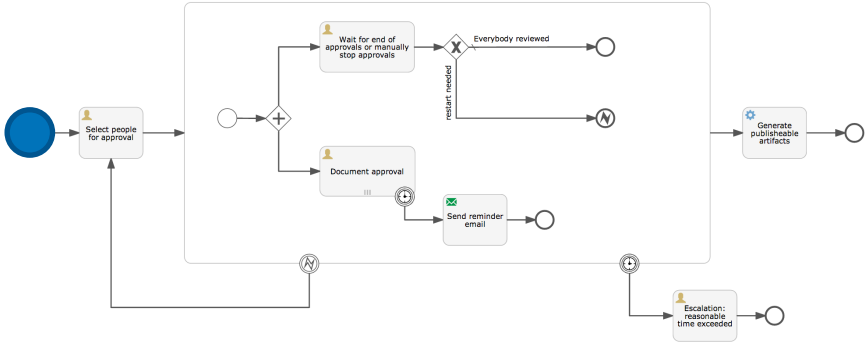
- Hard to reason about patterns in structure for runtime execution
- The problem is not clear for simple to even complex process definitions

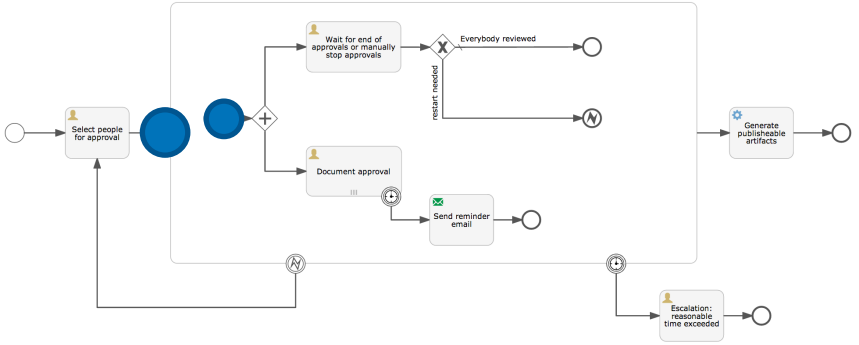
- But once you hit the _really_ complex process definitions …
  - And yeah, that tends to happen
  - Think something like a nested use of multi instance parallel nested embedded subprocesses with multiple compensation events and handlers with a couple of boundary events.
  - Then you are happy if you can reason in BPMN to get out of that

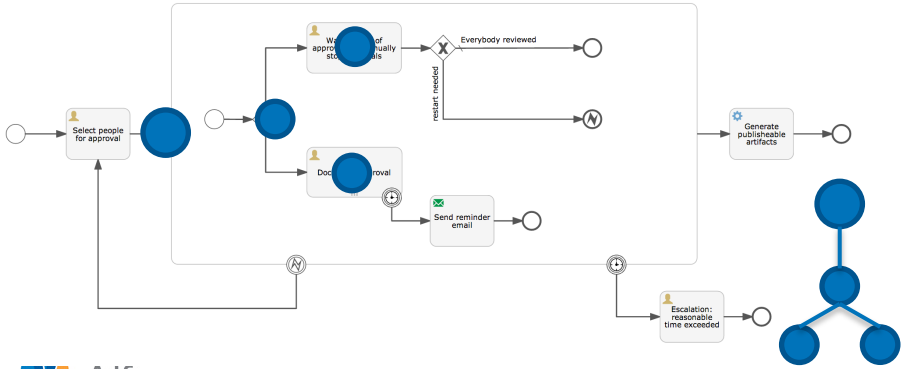


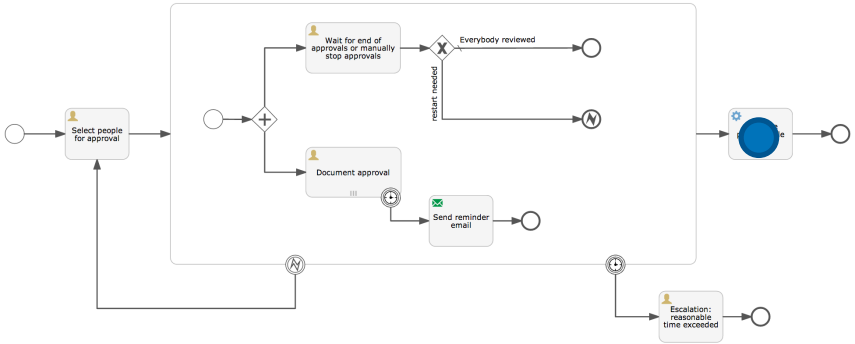
See "The Law of Leaky Abstractions" from Joel Spolsky
(http://www.joelonsoftware.com/articles/LeakyAbstractions.html)
Over 10 years old but still true.

Execution tree









Execution tree

- Stored relationally

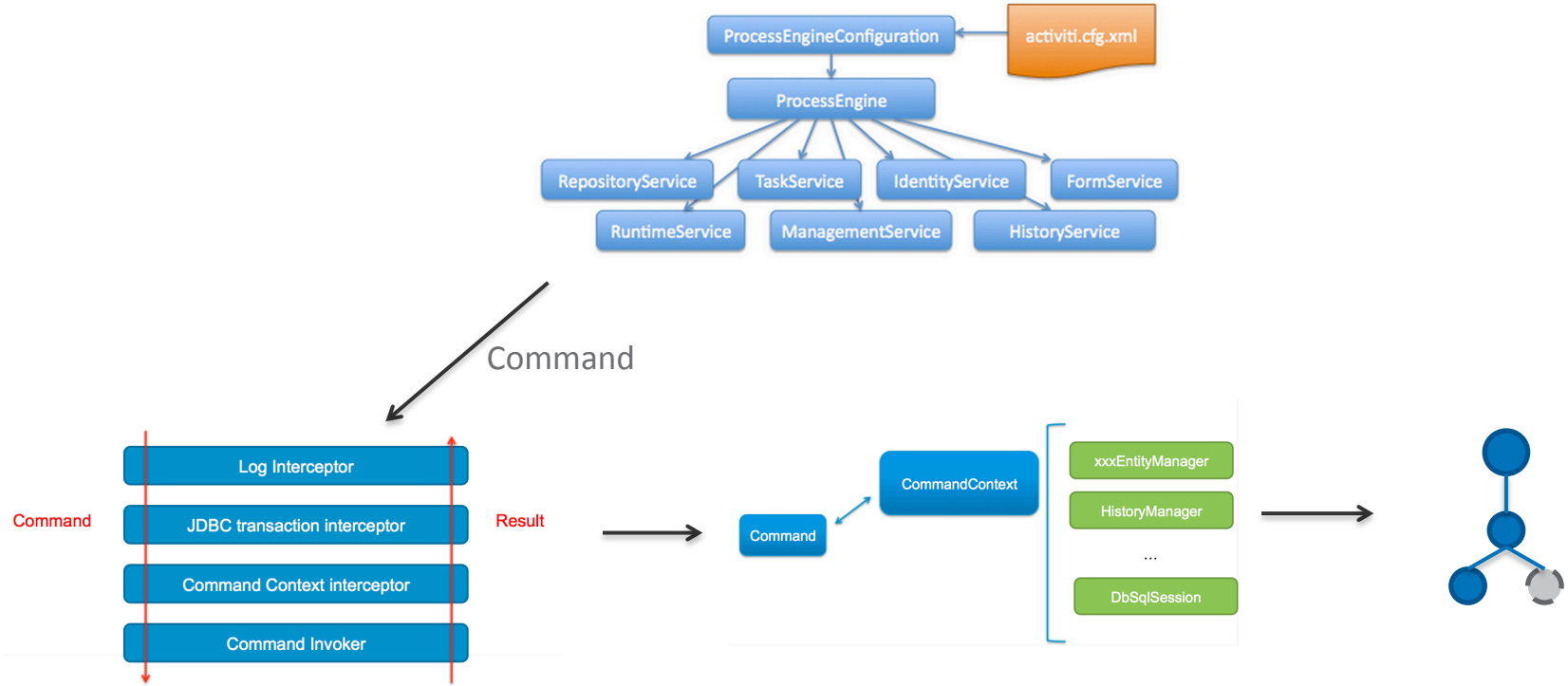
ID_	REV_	PROC_INST_ID_	BUSINESS_KEY_	PARENT_ID_
11	1	11	NULL	NULL
2504	1	2504	NULL	NULL
2505	1	2504	NULL	2504
2514	1	2514	NULL	NULL
2515	1	2514	NULL	2514
2522	1	2522	NULL	NULL
2523	1	2522	NULL	2522
5007	1	5007	NULL	NULL
5015	1	5015	NULL	NULL
NULL	NULL	NULL	NULL	NULL

V5 Problem 2 : Execution Tree

- Cause
  - (over)optimization and reuse entities
  - focus => getting the number of DB inserts as low as possible

- Consequence
  - Lots of if-elses for specific patterns
  - Hard to look at execution tree and deduce process definition
    - Which is crucial for dynamic process definition/instance (without hacking)

API Call (refresher)



V5 Problem 3

- Every API call boils down to
  - DB changes through one or more EntityManagers
  - Manipulation of the runtime execution structure
    - Manipulation of internal model (see problem 2)
    - Feeding that model into AtomicOperations (PVM core engine)
- Problem: atomic operations are started from ExecutionEntity object
  - New atomic operations are created to continue process
  - New atomic operations are put on a stack, bound to the ExecutionEntity
  - Multiple stacks when having multiple ExecutionEntities
- Some patterns can’t be executed on v5 due to that
- We tried to fix that, multiple times. Brave men have wept.

V5 Problem 4

- Logic is tightly coupled to DB layer
- DB code is spread between *EntityManagers* and the *Entity* object itself
  - Kinda like we couldn't decide between using DAO or Active Record pattern

- Really (impossible) to change persistence layer in v5

Why a v6?

- We could have tried to solve these problems on the v5 foundation
- But it would be a hack.
- We would get away with it for a while. But it would bite us back on the longer term
- So we decided to solve these problems at once (they are all interconnected, really), let the foundation evolve. Which is a big enough change to call it v6.

Activiti 6 :Solutions

Direct execution of BPMN Model

V5

- BPMN 2.0 XML parsed to PVM representation
- Step in process == ActivityBehavior
- ActivityBehavior
  - interpretes execution tree *through PVM model*
  - manipulates execution tree structure
  - creates new AtomicOperations

V6

- BPMN 2.0 XML parsed to BpmnModel, **1-1 java representation** of XML
- Step in process == ActivityBehavior
- ActivityBehavior
  - interpretes execution tree **through BPMN model**
  - manipulates execution tree structure
  - puts new engine operation on agenda

Execution tree

V5

- Optimize and reuse existing execution entities as much as possible
- No clear guidelines when execution entity is created or reused
  - Many types and flags
    - Scoped
    - Concurrent
    - Concurrent root
    - Event scoped
    - …

V6

- Execution entities are never reused
- Have clear rules when a new execution is created
  - Process instance = execution. Is not used for activity execution
  - Scope execution = parent for other executions within subprocess. Is not used for activity execution

Centralized Agenda

V5

- Multiple stacks of operations during API call execution
- PVM representation is continued through Atomic Operations, executed in the context of an ExecutionEntity.

V6

- One centralized agena per API call
- Behaviours interprete BPMN structure and put new operations on the centralized agena.

Persistence Code Everywhere

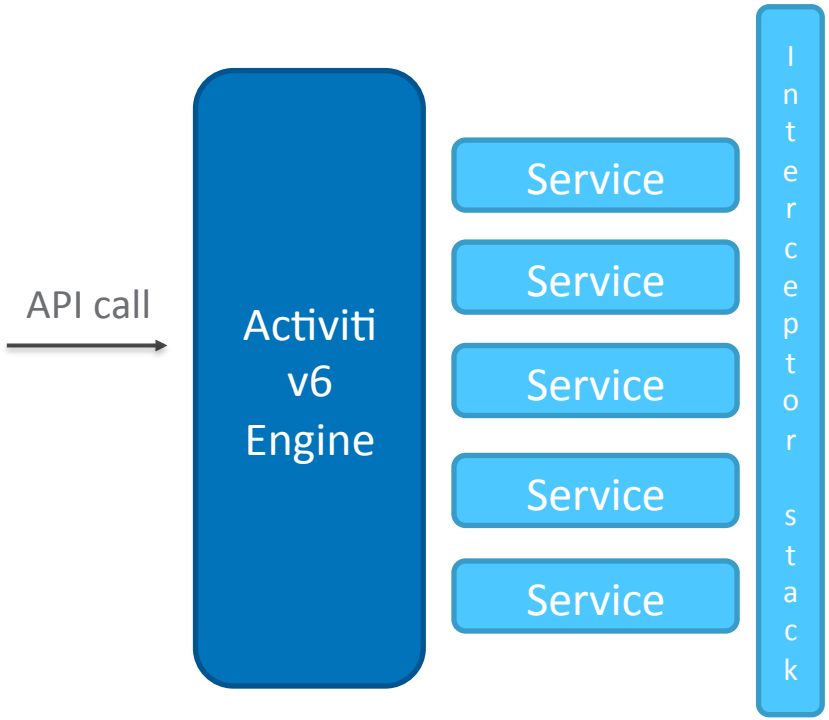
V5

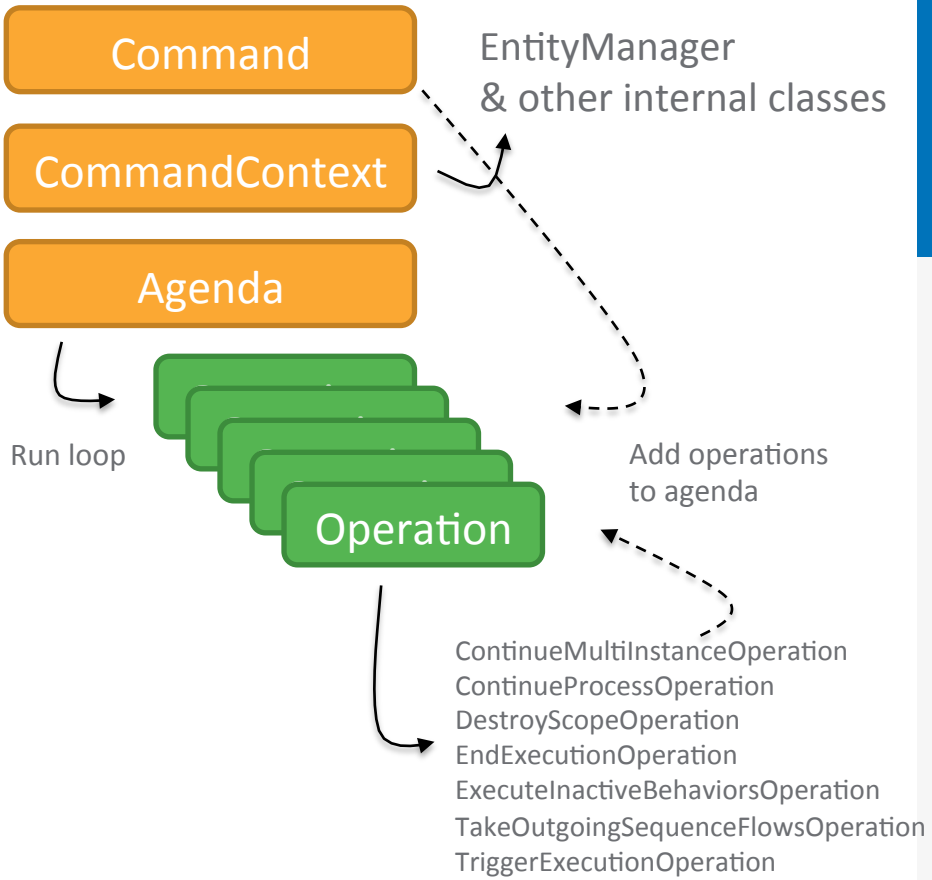
- Persistence code is in EntityManagers, Entity and probably some other places

V6

- **All** persistence code is moved to its EntityManager.
- All EntityManager are interfaces (will be)

High Level Flow



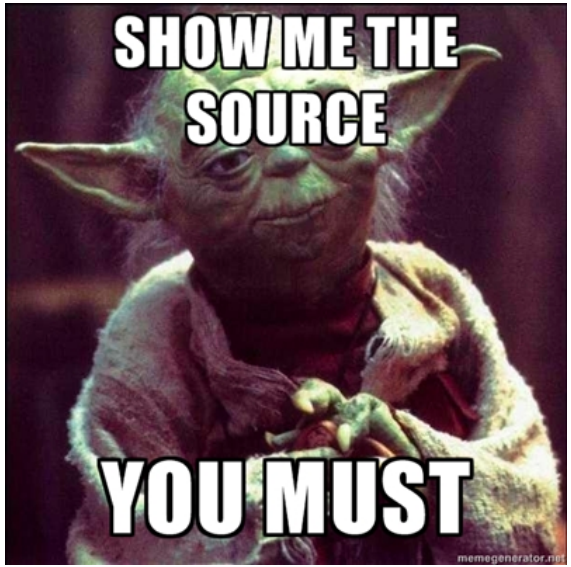


Benefits

- In the ActivityBehaviors, we reason about BPMN structure and patterns
  - Easier to to understand
  - More powerful, as nothing is lost in translation
- The execution tree structure is predictable and simple
  - Allows for more advanced behaviors
  - Hardest part in v5 was often the cleanup. This becomes child's play now.
  - Leads to support for dynamic instances/definitions

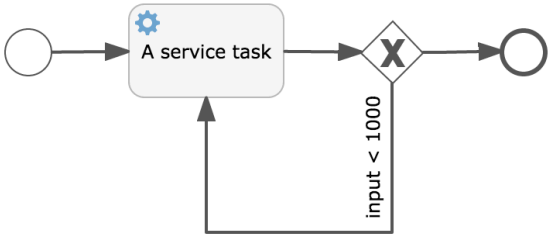
- Note: these are only the biggest changes in the v6 engine.
- Source is at https://github.com/Activiti/Activiti/tree/activiti6

Demo



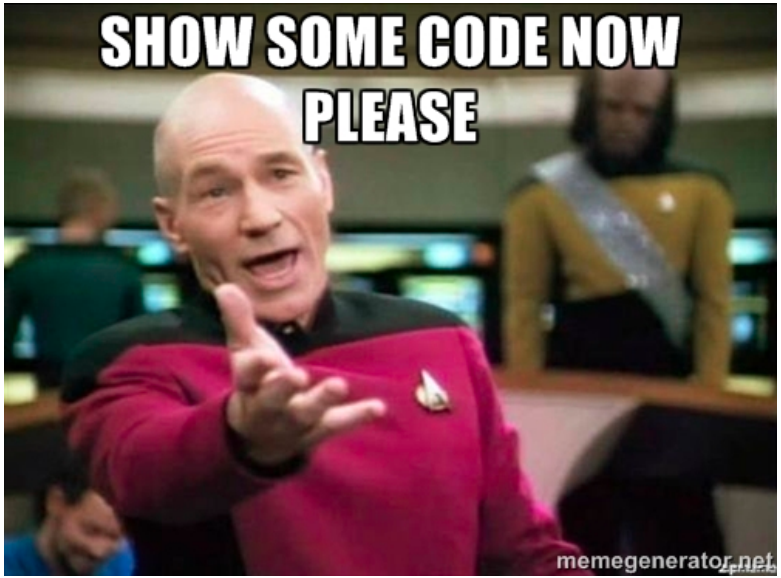
A few examples of why the evolved engine is better

Loops and Tail Recursion

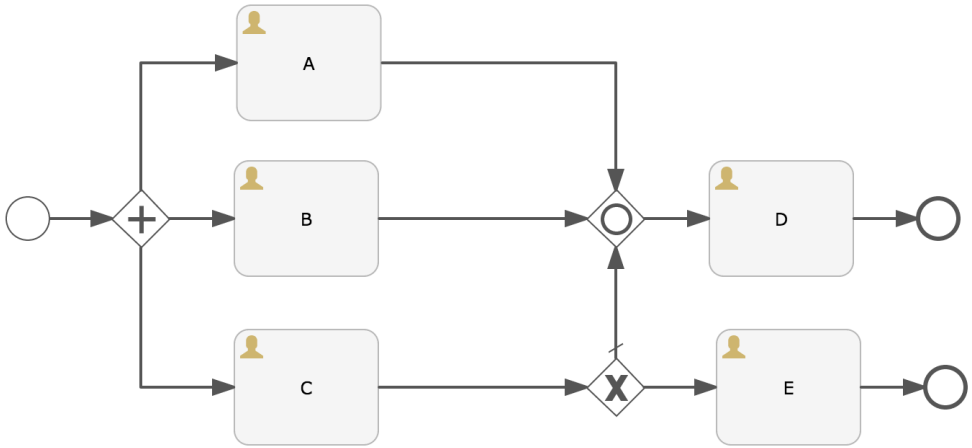


- V5 → StackOverFlowException
- V6 → All good

Demo



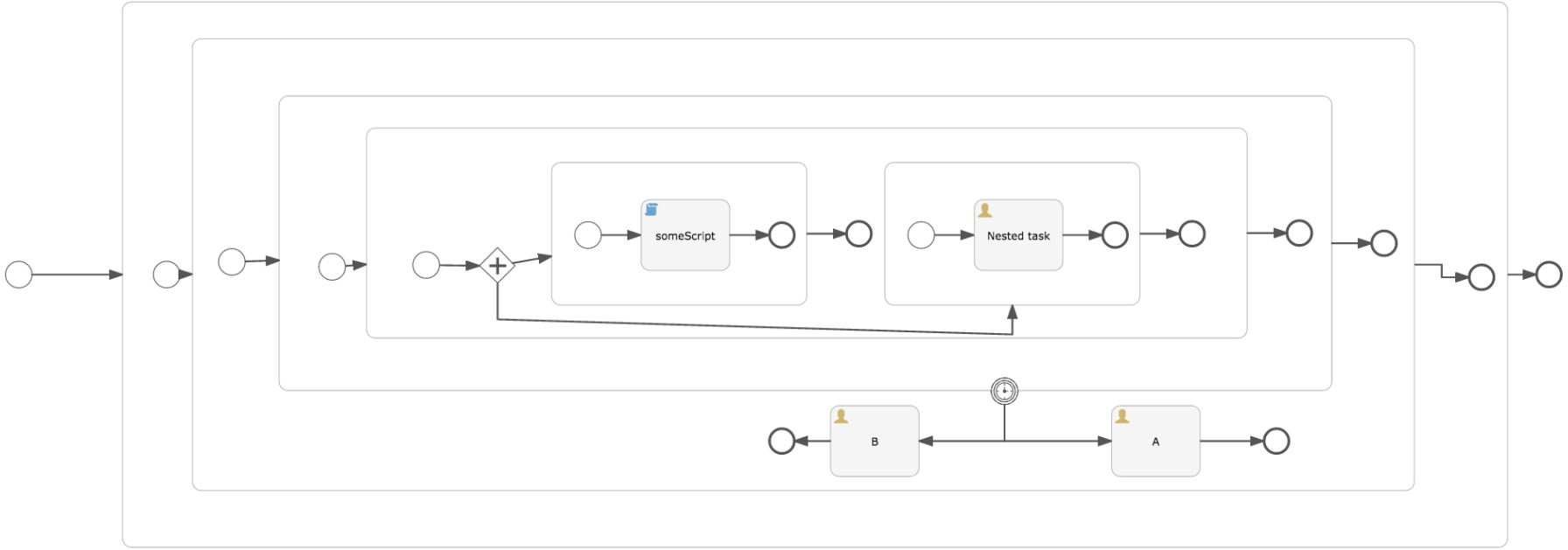
Tricky Inclusive Gateway



- A and B completed. C completes going to E.
- V5 → process instance never completes
- V6 → All good

Concurrency after boundary event

- Timer fires
  - V5 : 1 task
  - V6: 2 tasks (good)



Backwards compatibility

Backwards Compatibility

- API
- DB Schema
- Process Definitions
- Java Delegates
- Integration with other frameworks (Spring, Camel, …)
- Embeddability
- Concepts

What is not backwards compatible?

- Minor stuff that is quickly migrated:
  - Method signature changed for JavaDelegate/ActivityBehaviour
    - ‘throws Exception’ is gone
  - ActivityBehaviours that use PVM datastructures/methods
  - BpmnParseHandlers that act upon PVM ActivityImpl
  - Execution queries that depend on a certain number of executions
  - Assuming that an execution == process instance for simple processes

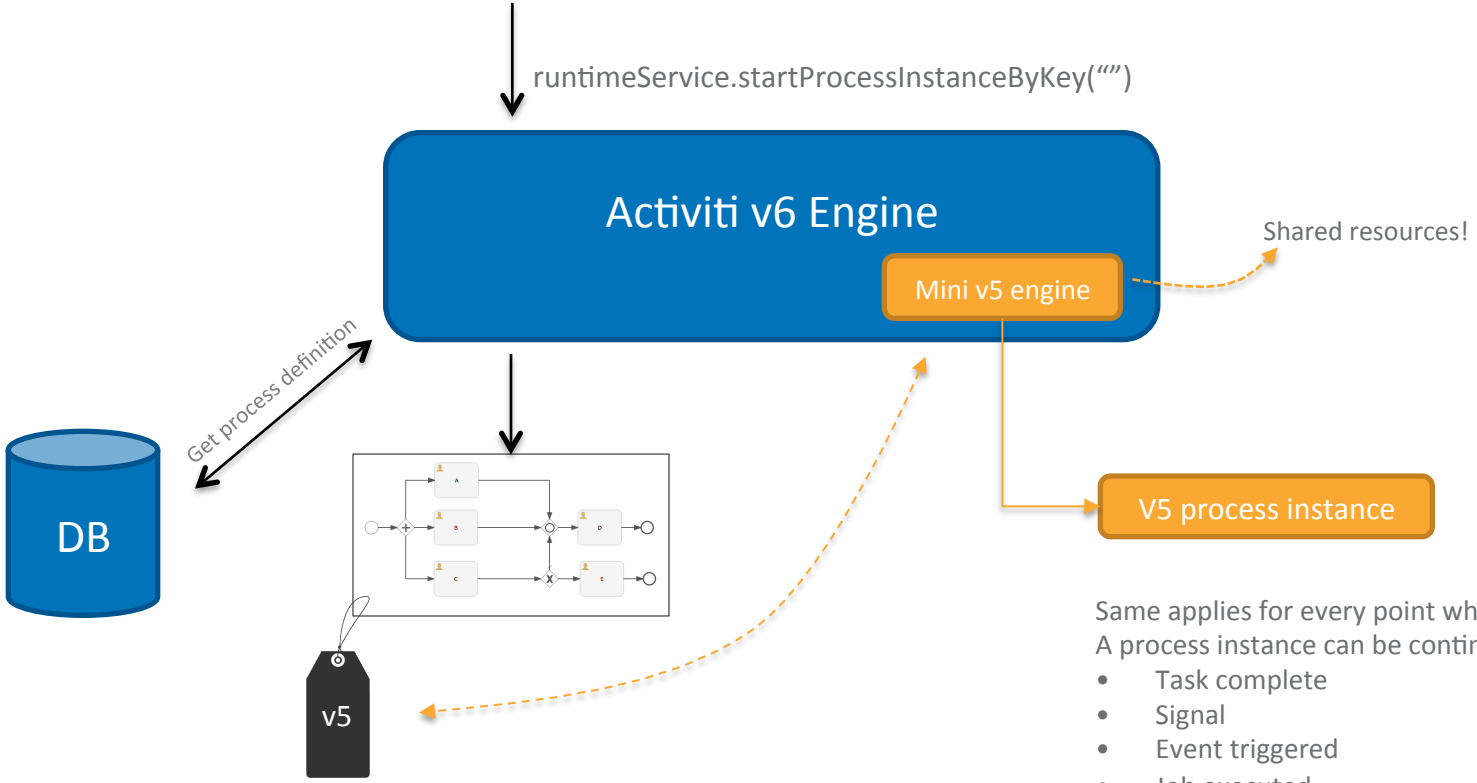
- Expect a Migration Guide soon

Backwards Compatibility – How?

- The upgrade of 5.x.x → 6.0.0 is like a regular update
- Drop jar, auto upgrade db (or execute DDL changes)

- BUT: all existing process definitions / deployments are tagged as ‘v5’

Backwards Compatibility – How?



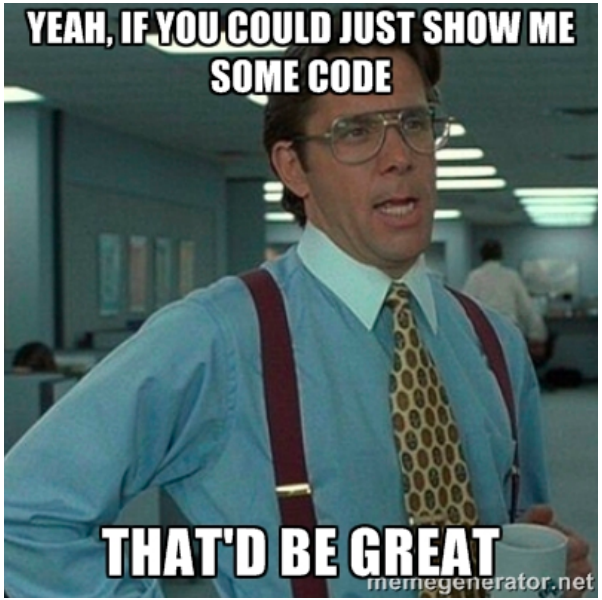
Same applies for every point where
A process instance can be continued:

- Task complete
- Signal
- Event triggered
- Job executed
- …

Migration of process definition to v6

- Backwards compatibility is enabled by having ‘activiti5-compatibility’ dependency on classpath
- Old process instances keep running against v5 ‘mini-engine’
  - Runtime execution is different from v6
- Running process definition in v6 is
  - Migrating your custom logic
  - Deploying a new version of the process definition

Demo


YEAH, IF YOU COULD JUST SHOW ME SOME CODE
THAT'D BE GREAT
memegenerator.net

Roadmap and wild ideas

- All of the next slides contain
  - Real roadmap items
  - Well worked out ideas
  - Half baked ideas
  - Dreams

- We have a roadmap discussion after this session!

Bulk Insert/Delete

- Lower traffic between engine and database
- Work already good under way
  - Bulk insert PR
  - Execution bulk delete in v6
    - Includes reordering for correct order
- But needs performance tests to validate
- Avoid delete queries if no data exists
  - Eg execution → related tasks, jobs, variables, …

BPMN Support

- Ad-hoc subprocess
- Event subprocess
  - More event types
  - Non-interrupting
- Java-based gateway
- Message intermediate throw event + better pool support
- Event-based gateway parallel semantics
- Escalation

Dynamic process definitions/instances

- Support for dynamic features
  - Tasks, processes related to process instance
  - Re-route a process instance in a custom way
  - Store the result as a new process definition

- Algorithms are most likely applicable to process definition migration logic

Variables

- New variable scopes
  - Global / Local (v5 semantics)
  - Scoped (within subprocess)
- Variable mapper
  - Multi instance finished
  - Call activity finished
- System variables
- Transient variables

Execution tree fetching

- Fetch the execution tree when only one execution is needed
- Subsequent execution queries don’t hit the DB

- Cache the trees beyond the API call context?
  - Solve cluster cache problem?

- Always or configurable?
  - Need performance metrics

History

- Async history
  - Performance ++
  - What about transactions?

Runtime Swapping

- Swap out behaviours, listeners, expressions at runtime
- With specific conditions
  - Allow to patch in flight process instance by overriding behaviour

More hook points

- Pre/post transaction commit
- Pre/post execution listener/behaviour/delegate
- Pre/post process first parse/ subsequent cache load
- …

Delegates, ActivityBehaviours and Listener scope

- Allow for
  - singletons (process engine scope - v5 semantics)
  - Stateful instances process definition scope
  - Stateful instances process instance scope

Debug Mode

- Collecting data during runtime execution
  - Agenda operations
  - Execution tree changes
  - % of time spent in DB vs Logic

Simplify Unit Testing

- Currently, writing a unit test
  - Create process definition xml
  - Put it on classpath, link it to unit test (@Deployment or RepositoryService)
  - Write unit tests moving process instance

- Ideas
  - Meta notation (annotation/methods/…) putting process in certain state
  - Generating a unit test skeleton from the Modeler

Real Concurrency

- The new foundation *should* allow for real parallelism
- Haven't tried it yet
- Will lead to a lot of fun with transactions

Docs

- Now: tech guide
- Future
  - Work more from use cases and examples
  - Tijs will write a new book?
  - Living book?

Other

- Community Driven
- Bring it on