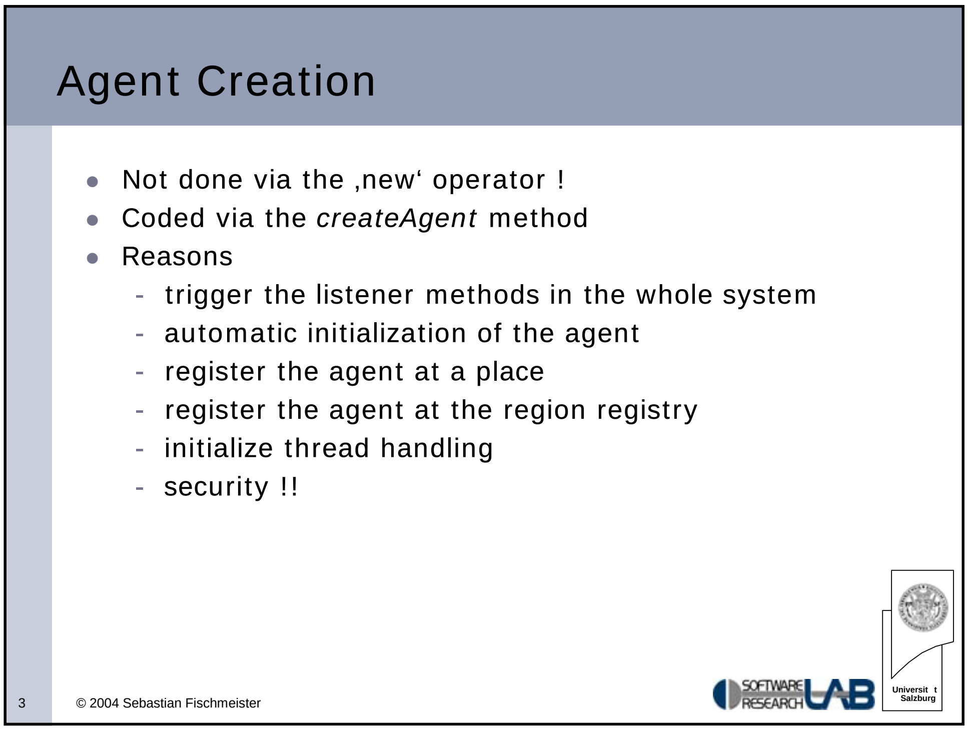
Software Technologies

Mobile Code

Sebastian Fischmeister
fischmeister@softwareresearch.net
University of Salzburg

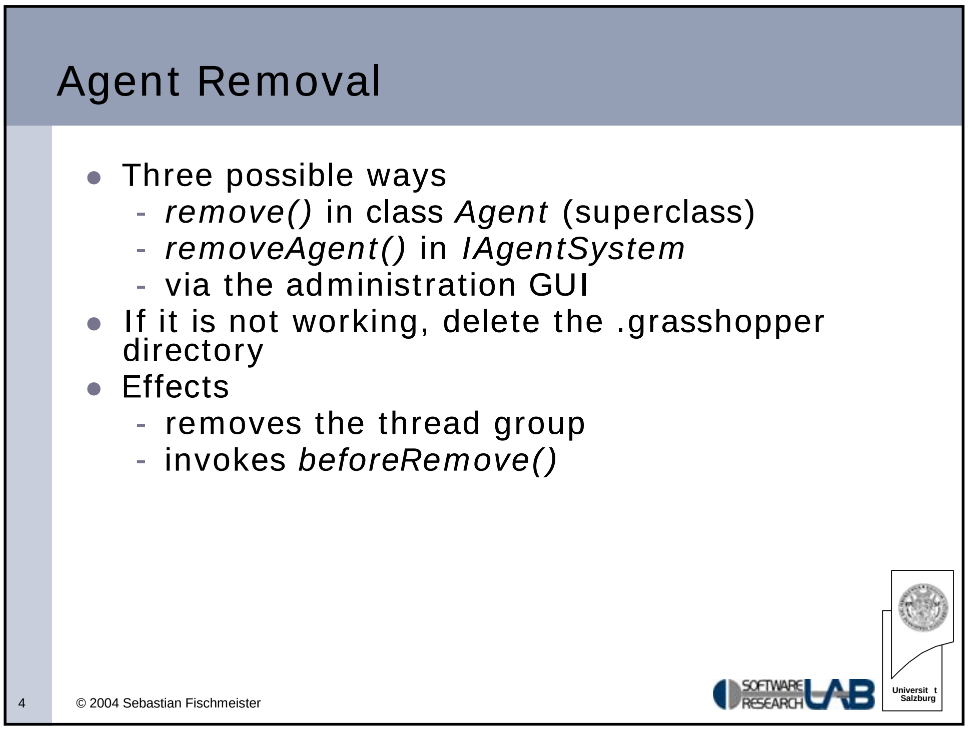
Contents

Universit t Salzburg



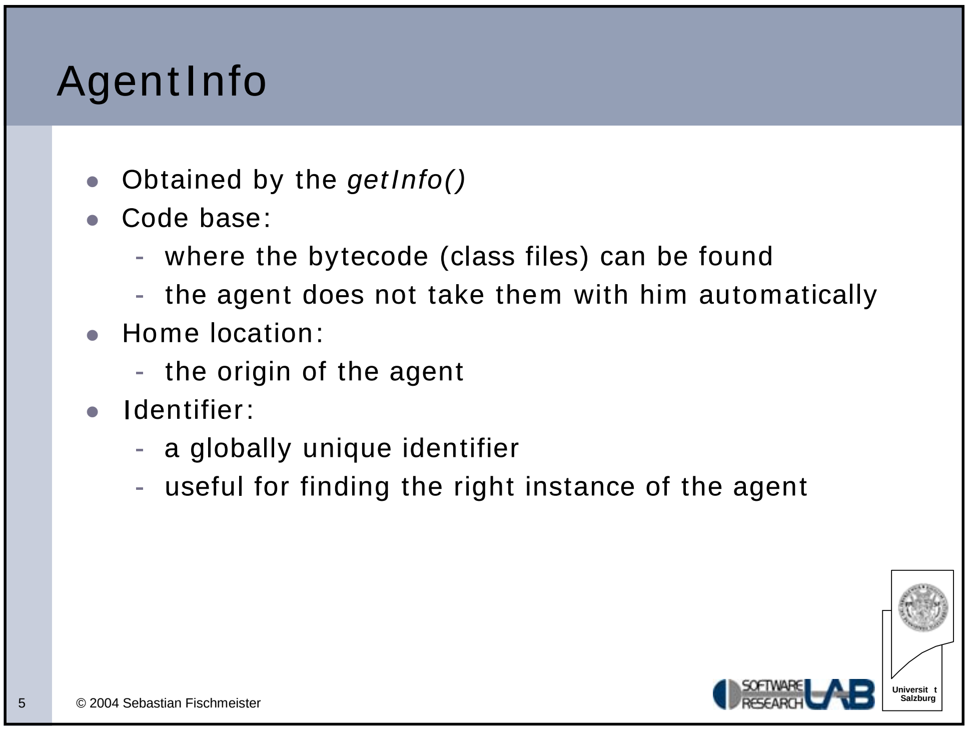
Agent Creation

- Not done via the ‚new‘ operator !
- Coded via the *createAgent* method
- Reasons
  - trigger the listener methods in the whole system
  - automatic initialization of the agent
  - register the agent at a place
  - register the agent at the region registry
  - initialize thread handling
  - security !!



Agent Removal

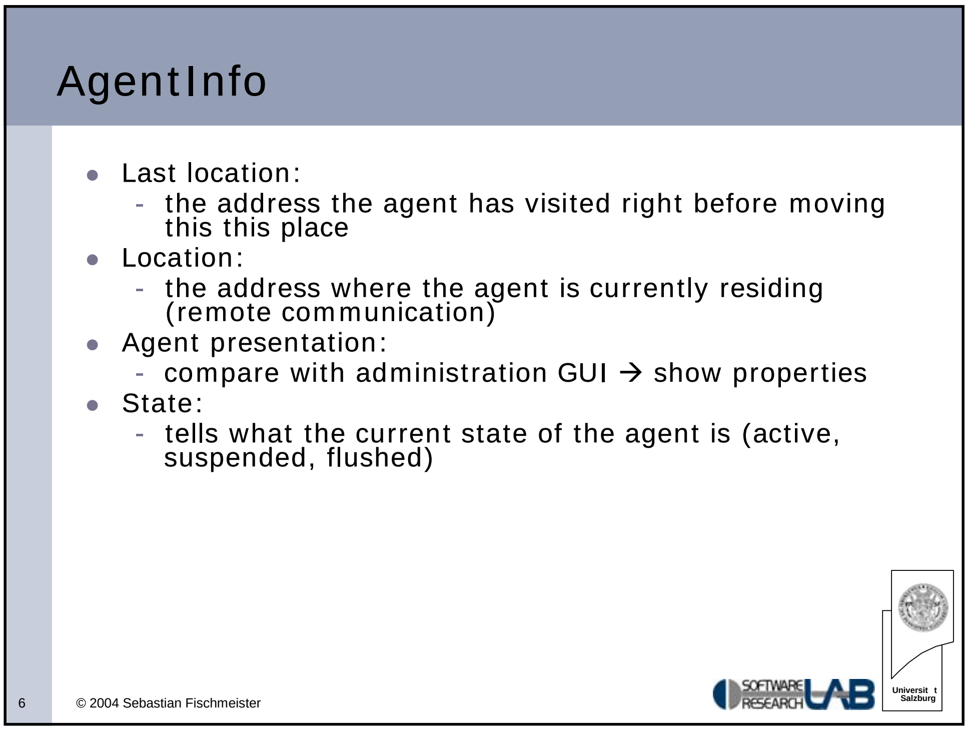
- Three possible ways
  - *remove()* in class *Agent* (superclass)
  - *removeAgent()* in *IAgentSystem*
  - via the administration GUI
- If it is not working, delete the .grasshopper directory
- Effects
  - removes the thread group
  - invokes *beforeRemove()*



AgentInfo

- Obtained by the *getInfo()*
- Code base:
  - where the bytecode (class files) can be found
  - the agent does not take them with him automatically
- Home location:
  - the origin of the agent
- Identifier:
  - a globally unique identifier
  - useful for finding the right instance of the agent

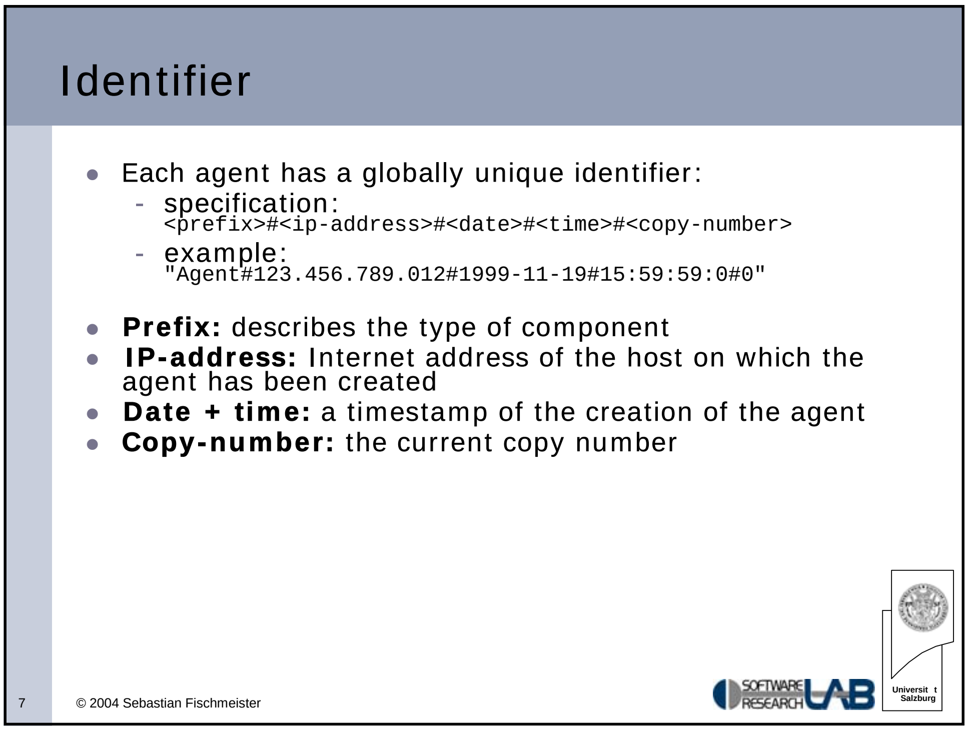
5



AgentInfo

- Last location:
  - the address the agent has visited right before moving this this place
- Location:
  - the address where the agent is currently residing (remote communication)
- Agent presentation:
  - compare with administration GUI → show properties
- State:
  - tells what the current state of the agent is (active, suspended, flushed)

6

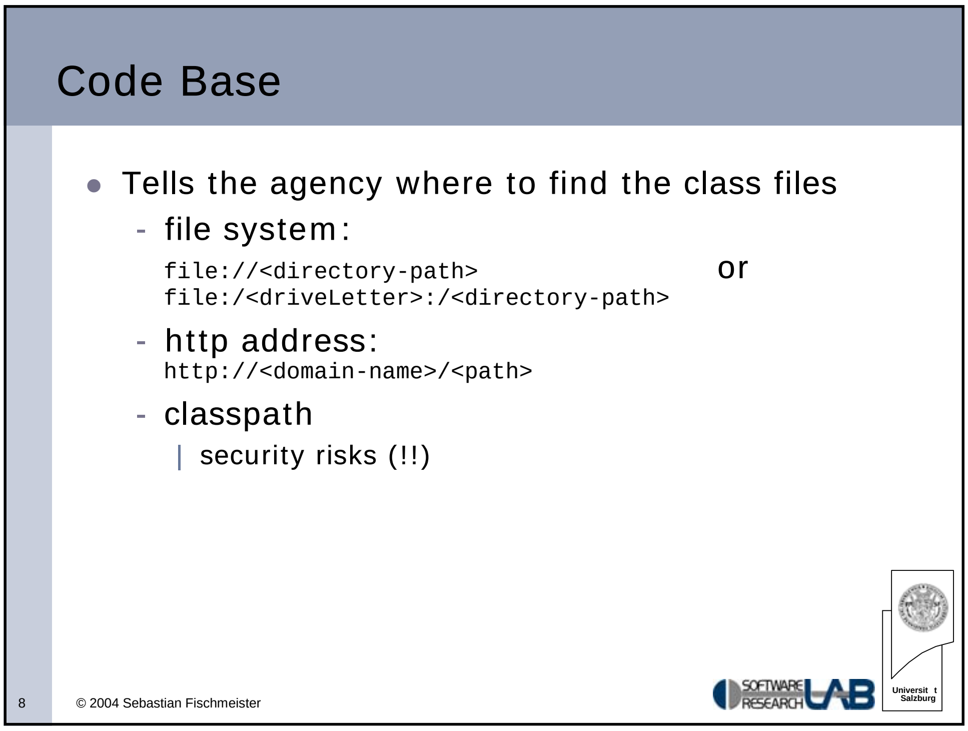


Identifier

- Each agent has a globally unique identifier:
  - specification:
    `<prefix>#<ip-address>#<date>#<time>#<copy-number>`
  - example:
    `"Agent#123.456.789.012#1999-11-19#15:59:59:0#0"`

- **Prefix:** describes the type of component
- **IP-address:** Internet address of the host on which the agent has been created
- **Date + time:** a timestamp of the creation of the agent
- **Copy-number:** the current copy number

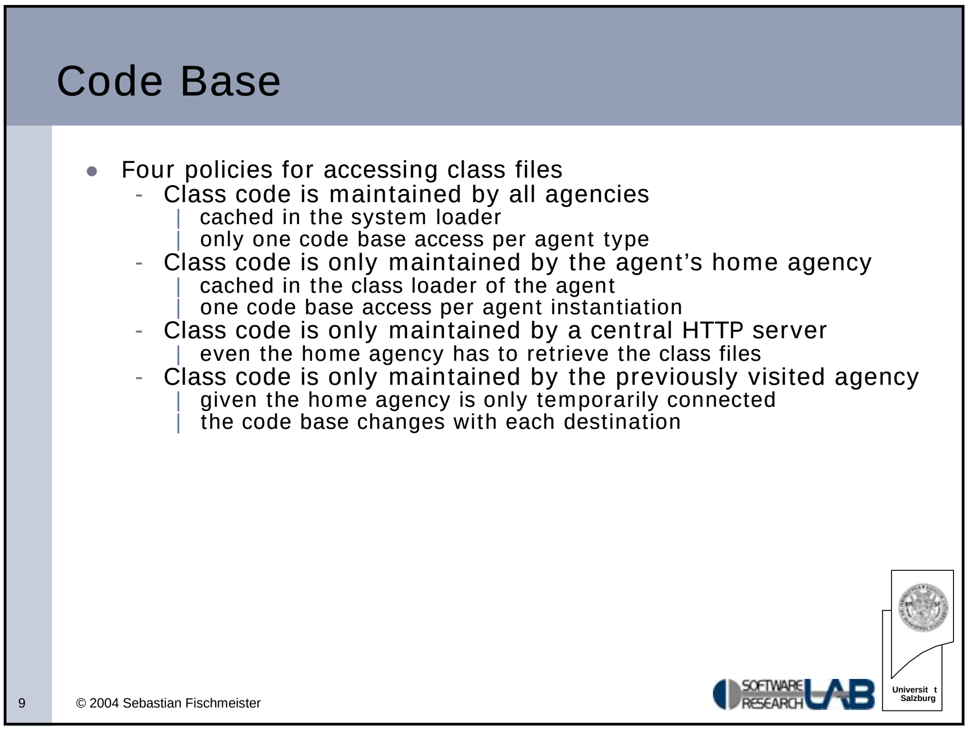
7 © 2004 Sebastian Fischmeister



Code Base

- Tells the agency where to find the class files
  - file system:
    ```
    file://<directory-path>                  or
    file:/<driveLetter>:/<directory-path>
    ```
  - http address:
    `http://<domain-name>/<path>`
  - classpath
    - security risks (!!)

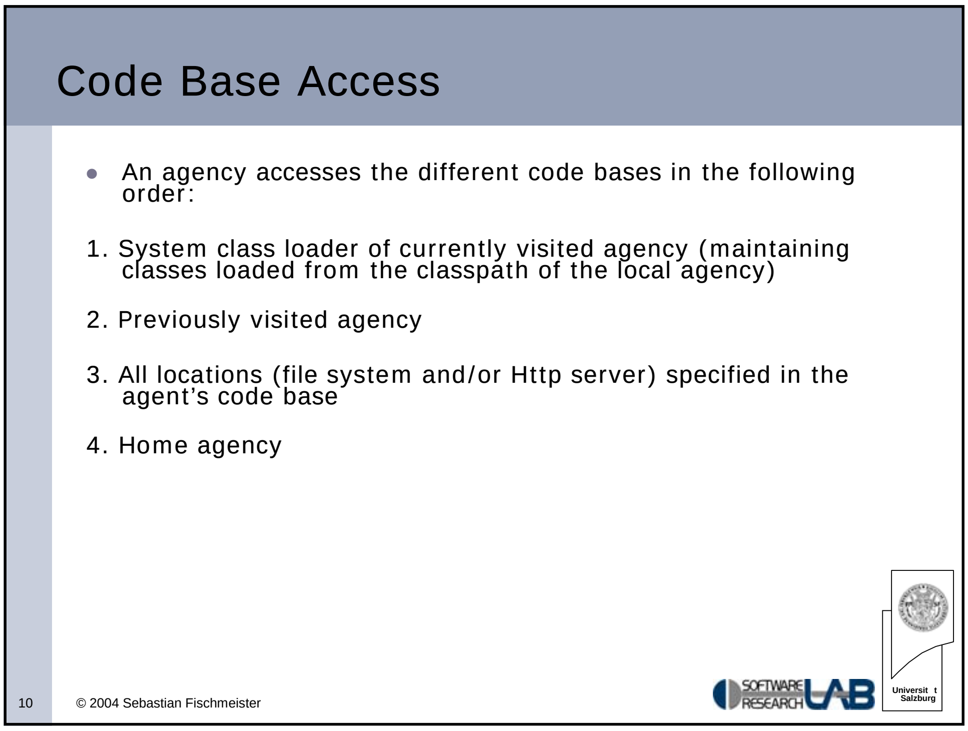
8 © 2004 Sebastian Fischmeister



Code Base

- Four policies for accessing class files
  - Class code is maintained by all agencies
    - cached in the system loader
    - only one code base access per agent type
  - Class code is only maintained by the agent's home agency
    - cached in the class loader of the agent
    - one code base access per agent instantiation
  - Class code is only maintained by a central HTTP server
    - even the home agency has to retrieve the class files
  - Class code is only maintained by the previously visited agency
    - given the home agency is only temporarily connected
    - the code base changes with each destination

9 © 2004 Sebastian Fischmeister



Code Base Access

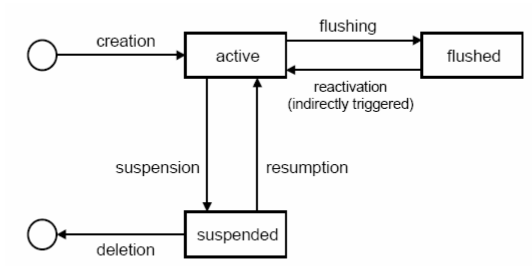
- An agency accesses the different code bases in the following order:

1. System class loader of currently visited agency (maintaining classes loaded from the classpath of the local agency)
2. Previously visited agency
3. All locations (file system and/or Http server) specified in the agent's code base
4. Home agency

10 © 2004 Sebastian Fischmeister

Life Cycle

- Agent states:
  - **active:** up an running
  - **suspended:** suspending the thread, it's down
  - **flushed:** controlled by the persistency manager, reactivated on communication



11

Life Cycle

- How to change the state?
  - use the *AgentSystem* functions
    - *flushAgent()*
    - *reloadAgent()*
    - *resumeAgent()*
    - *saveAgent()*
    - *suspendAgent()*

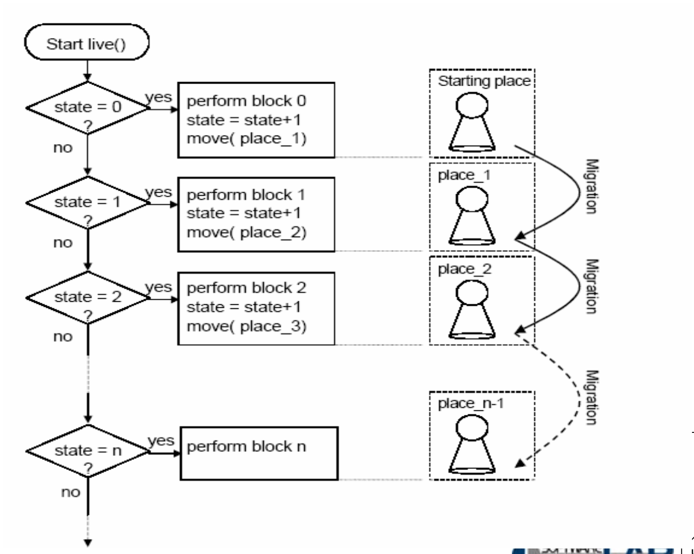
12

Migration Process

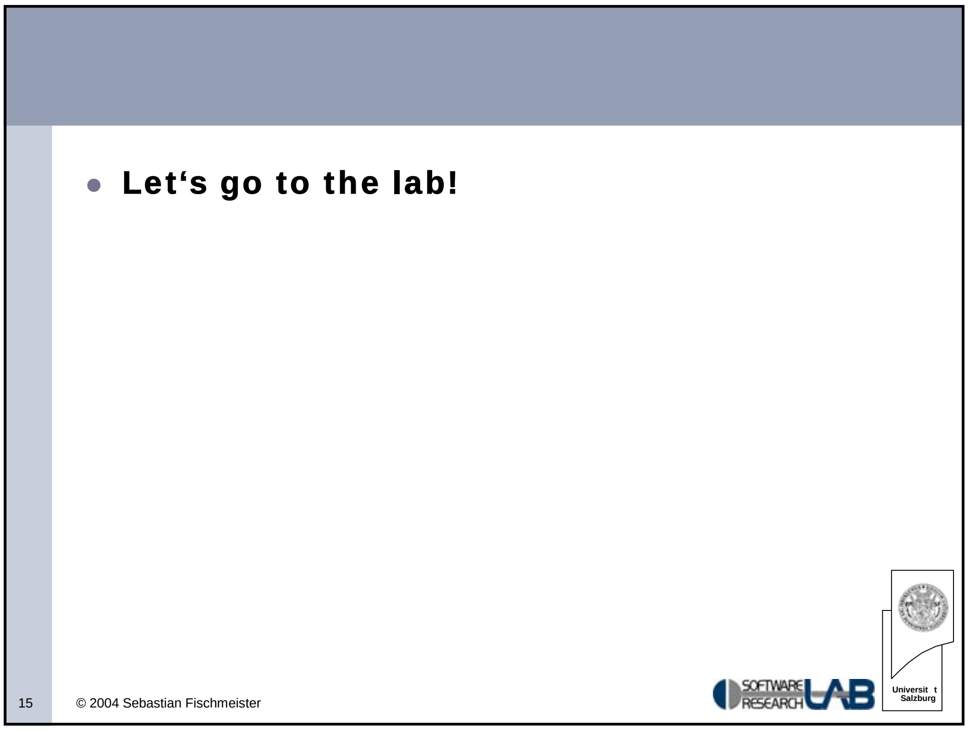
- Grasshopper performs the following steps:
  1. initialize migration (*move()* or *moveAgent()*)
  2. invoke agent‘s *beforeMove()*
     - prepares for moving
     - can throw the VetoException
  3. interrupt agent by stopping the thread
  4. serialize the agent
     - take care of transient declarations
  5. transfer agent‘s data state and additional information (agent name, code base, ...)
  6. create a new instance at the destination with the serialized data
  7. inform the source agency about successful instantiation
  8. invoke agent‘s *afterMove()*
  9. start the thread of the agent

13 © 2004 Sebastian Fischmeister

Structuring an Agent‘s Life



14 © 2004 Sebastian Fischmeister



- **Let‘s go to the lab!**

15 © 2004 Sebastian Fischmeister

SOFTWARE RESEARCH LAB
Universit t Salzburg