

# Ros on Windows

and an intro to **Catkin!**

# Notice

## Demos

Due to the time constraints I've used screenshots.

If you would like to see live demos → see me after the talk.

# The Ugly Duckling



# Conundrum

- Linux devs don't want to do windows programming
- Windows devs hesitate at our build/runtime environment

# Why Develop?

- Linux devs don't want to do windows programming
- Windows devs dont want to do our build environment

Why should we annoy each other?

# Why Annoy Each Other?

The control team needs to provide:

- Diagnostics
  - Monitoring Programs
  - Configuration Tools
- 
- Factory
  - Test Engineers
  - End Users

Some sensors only have windows drivers.

Windows components or robots need to interface with you.

You want to utilise an offboard windows pc

# Goals

Our needs are fairly modest

Monitoring and Diagnostics

Windows Sensors

Software System Bridging

Simple User Apps

Mostly ros runtime, comms, msgs and guis

Not trying to recreate the entire ros ecosystem

# History

Have seen 3 attempts to port ros to windows.  
None survived. Why?

## Build Environment

Avoid the build environment → Version upgrades hard to maintain

Cross Platform Build Environment?  
No Makefiles, No Bash....Python? Cmake?

# Catkin

Rosbuild → Catkin

[ Make + Cmake + Python ] → [ Cmake + Python ]

**Very Disruptive!!!** (just wait till Groovy)

## Benefits

- Much faster (only 1 cmake invocation)
- Cross Platform (incl. Cross-compiling)
- Separates source from build
- More standard usage (cmake, os integration, packaging)
- Has an install step

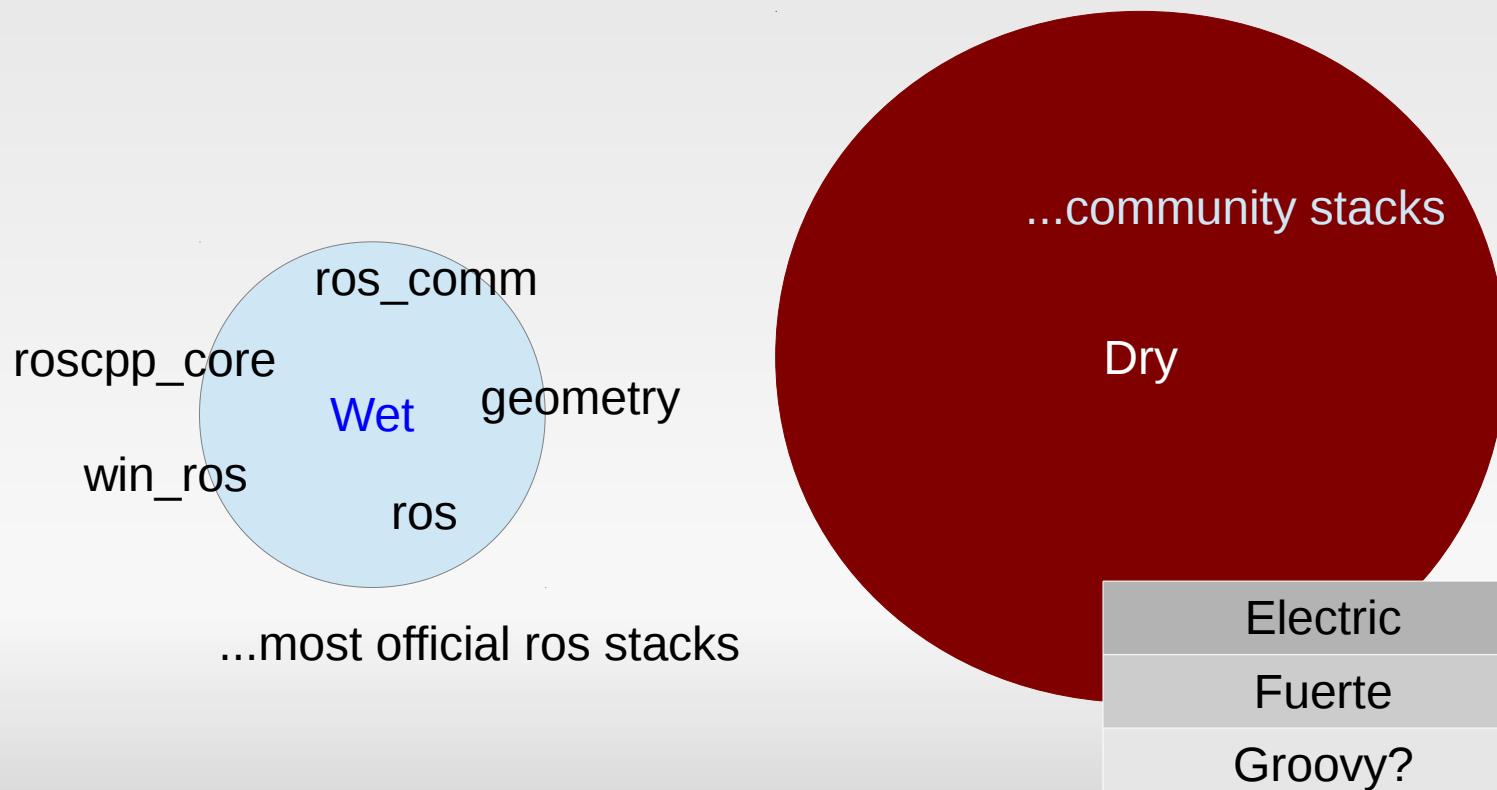
*Troy Straszheim is a legend*

# Catkin Status

Packages can either be rosbUILD or catkin

Mutually Exclusive!!!

- Catkinized packages → ‘wet’,
- Rosbuild packages → ‘dry’



# Catkinization

```
project(roscpp_tutorials)
catkin_project(roscpp_tutorials)
find_package(Boost COMPONENTS date_time thread)
find_package(ROS COMPONENTS catkin genmsg roscpp)
include_directories(${ROS_INCLUDE_DIRS})
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_CURRENT_BINARY_DIR}/bin)
add_service_files(DIRECTORY srv FILES TwoInts.srv)
generate_messages(DEPENDENCIES std_msgs)
macro(rostutorial T)
    add_executable(${T} ${T}/.${T}.cpp)
    find_package(ROS COMPONENTS catkin genmsg rostime roscpp rosconsole roscpp_serialization)
    target_link_libraries(${T} ${ROS_LIBRARIES} ${Boost_LIBRARIES})
    add_dependencies(${T} roscpp_tutorials_gen.cpp)
    install(TARGETS ${T} RUNTIME DESTINATION share/roscpp_tutorials/bin)
endmacro()
foreach(dir listener talker ... )
    rostutorial(${dir})
endforeach()
add_executable(time_api_sleep time_api/sleep/sleep.cpp)
target_link_libraries(time_api_sleep ${ROS_LIBRARIES})
install(TARGETS time_api_sleep RUNTIME DESTINATION share/roscpp_tutorials/bin)
```

# Catkin Demo

```
> rosinstall -catkin ./src catkin_demo.rosinstall  
> mkdir build  
> cd build  
> cmake ../src  
> make -j3  
> make install
```

Can test (rosrun, roslaunch..) inside build or install environments

# Catkin Demo – Cmake Invocation

```
~/windows_on_ros/demo/native : cmake
File Edit View Bookmarks Settings Help
yujin@yujin-kobuki1-laptop:~/windows_on_ros/demo/native$ cmake ../src
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- +++ catkin
-- Found PythonInterp: /usr/bin/python (found version "2.7.3")
-- Looking for include files CMAKE_HAVE_PTHREAD_H
-- Looking for include files CMAKE_HAVE_PTHREAD_H - found
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
TODO: implement add_roslaunch_check() in rostest-extras.cmake.
-- BUILD_SHARED_LIBS is on.
-- ~~ traversing stacks/projects in dependency order ~~
-- ~~
-- +++ genmsg
-- +++ genpy
-- +++ gencpp
-- +++ rosCPP_core
-- Looking for execinfo.h
-- Looking for execinfo.h - found
-- Performing Test HAVE_CXXABI_H
-- Performing Test HAVE_CXXABI_H - Failed
-- Looking for backtrace
-- Looking for backtrace - found
-- Boost version: 1.46.1
-- Found the following Boost libraries:
--   date_time
--   thread
--   +++ std_msgs
--   std_msgs: 32 messages
--   +++ rospack
-- Rospack building shared objects.
-- Boost version: 1.46.1
-- Found the following Boost libraries:
--   system
--   filesystem
[~] ~/windows_on_ros/demo/native : cmake
```

# Goals

Our needs are fairly modest

Monitoring and Diagnostics

Windows Sensors

Software System Bridging

Simple User Apps

Mostly ros runtime, comms, msgs and guis

Not trying to recreate the entire ros ecosystem

# Plan

- Linux Devs ← static mingw cross compiler + qt
- Windows Devs ← msvc sdk + whatever they want

# Mingw Cross

<http://mxe.cc/>

- Bootstraps a mingw cross compiler for \*nix
- Produces static libraries/binaries
- Build recipes for boost and qt (many others)
- Build recipes for apr, apr-util, log4cxx (we added)
- Supplies a cmake toolchain module

```
> rosinstall -catkin ./src mingw_fuerte.rosinstall  
> cd src/win_ros/mingw/mingw_cross  
> make install #installs mingw-cross + some utils
```

# Mingw Demo

```
> rosinstall -catkin ./src mingw_fuerte.rosinstall
> mkdir mingw
> cd mingw
> cmake
    -DCMAKE_TOOLCHAIN_FILE=${MINGW_CMAKE_TOOLCHAIN_FILE}
    -C ${MINGW_CMAKE_ROS_CONFIG_FILE}
    ./src
> cd ros_tutorials/roscpp_tutorials; make -j5
> cd ../../qt_ros/qt_tutorials; make -j5
```

[http://www.ros.org/wiki/mingw\\_cross/Mingw Build Environment - Fuerte](http://www.ros.org/wiki/mingw_cross/Mingw+Build+Environment+-+Fuerte)

# Cmake Toolchain Files

```
set(BUILD_SHARED_LIBS OFF)
set(CMAKE_SYSTEM_NAME Windows)
set(MSYS 1)
set(CMAKE_FIND_ROOT_PATH /home/yujin/mingw/usr/i686-pc-mingw32)
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_C_COMPILER /home/yujin/mingw/usr/bin/i686-pc-mingw32-gcc)
set(CMAKE_CXX_COMPILER /home/yujin/mingw/usr/bin/i686-pc-mingw32-g++)
set(CMAKE_Fortran_COMPILER /home/yujin/mingw/usr/bin/i686-pc-mingw32-gfortran)
set(CMAKE_RC_COMPILER /home/yujin/mingw/usr/bin/i686-pc-mingw32-rc)
set(PKG_CONFIG_EXECUTABLE /home/yujin/mingw/usr/bin/i686-pc-mingw32-pkg-config)
set(QT_QMAKE_EXECUTABLE /home/yujin/mingw/usr/bin/i686-pc-mingw32-qmake)
set(CMAKE_INSTALL_PREFIX /home/yujin/mingw/usr/i686-pc-mingw32 CACHE PATH "Installation Prefix")
set(CMAKE_BUILD_TYPE Release CACHE STRING "Debug|Release|RelWithDebInfo|MinSizeRel")
```

Toolchain Configuration!

# CMake Cache Files

```
set(TOOLCHAIN_TUPLE i686-pc-mingw32)
# Boost needs to be hand held on windoze for boost_thread.
# WIN_32_WINNT : needed to find symbols properly
#   refer to http://lists-archives.org/mingw-users/04689-linking-help.html
# BOOST_THREAD_USE_LIB : needed to tell it to cleanup properly
#   refer to http://lists.gnu.org/archive/html/mingw-cross-env-list/2011-05/msg00060.html
set(MINGW_CROSS_COMPILE_FLAGS "-DBOOST_THREAD_USE_LIB -D_WIN32_WINNT=0x0501")
set(Boost_THREADAPI "win32" CACHE STRING "Necessary variable for cmake to find mingw boost, needs cmake v2.8.3+")
#set(Boost_USE_STATIC_LIBS TRUE CACHE STRING "Using static libs for boost.")

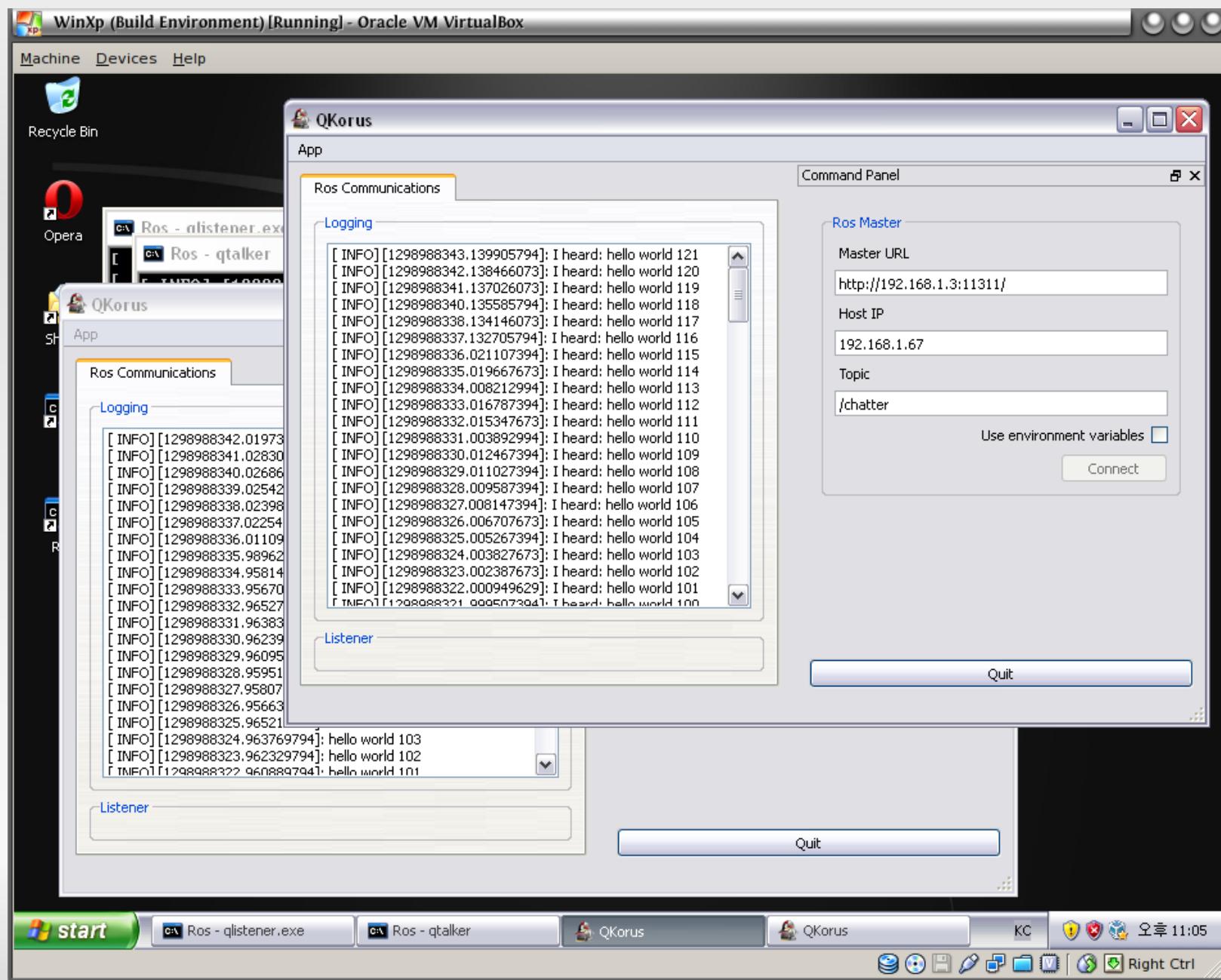
set(CMAKE_C_FLAGS ${MINGW_CROSS_COMPILE_FLAGS} CACHE PATH "Compile flags for c.")
set(CMAKE_CXX_FLAGS ${MINGW_CROSS_COMPILE_FLAGS}

Preload Cmake Variables

set(QT_IS_STATIC 1) # Works on my gentoo (cmake 2.8.1), fails on lucid ubuntu (cmake 2.8.0)
set(QT_QMAKE_EXECUTABLE ${TOOLCHAIN_TUPLE}-qmake CACHE PATH "Qmake location")

# No way of knowing if the target machine has sse, so lets disable by default
# so ros doesn't fall over on the TRY_RUN tests.
set(HAS_SSE_EXTENSIONS_EXITCODE FALSE CACHE BOOL "Cross-compiling variable en/disabling sse extensions.")
set(HAS_SSE2_EXTENSIONS_EXITCODE FALSE CACHE BOOL "Cross-compiling variable en/disabling sse2 extensions.")
set(HAS_SSE3_EXTENSIONS_EXITCODE FALSE CACHE BOOL "Cross-compiling variable en/disabling sse3 extensions.")
```

# Mingw Result



# MSVC

- Core ros dependency installers (log4cxx, boost, python, rosinstall)
- Windows SDK 7.1 + nmake → Ros SDK
- Python/C++ Runtime Environment
- Support 64 bit
- Support Qt
- Rosdep environment (<http://www.nabber.org/projects/appupdater/>)
- Support Express and Visual Studio Generators
- Port more stacks

Our Goal

Upgrade build environment so windows devs can start helping themselves.

# Building the SDK

Windows SDK 7.1 + nmake is a great combo

- Install ros dependencies
- Download and patch (win-rosinstall,  
win\_ros/win\_patches)
- Configure (cmake invocation)
- Compile (cd build; nmake)
- Install (cd build; nmake install)

Process similar to mingw, no toolchain file.

[http://www.ros.org/wiki/win\\_ros/Msvc Build Environment](http://www.ros.org/wiki/win_ros/Msvc%20Build%20Environment) - Fuerte

# Using the SDK

MS Express, Visual Studio

- Usual MS Project Style
  - Install the SDK
  - Add include and linking directories
  - Add ros environment variables for debugging environment

[http://www.ros.org/wiki/win\\_ros/Msvc SDK - Fuerte](http://www.ros.org/wiki/win_ros/Msvc%20SDK%20-%20Fuerte)

[http://www.ros.org/wiki/win\\_ros/Msvc SDK Projects - Fuerte](http://www.ros.org/wiki/win_ros/Msvc%20SDK%20Projects%20-%20Fuerte)

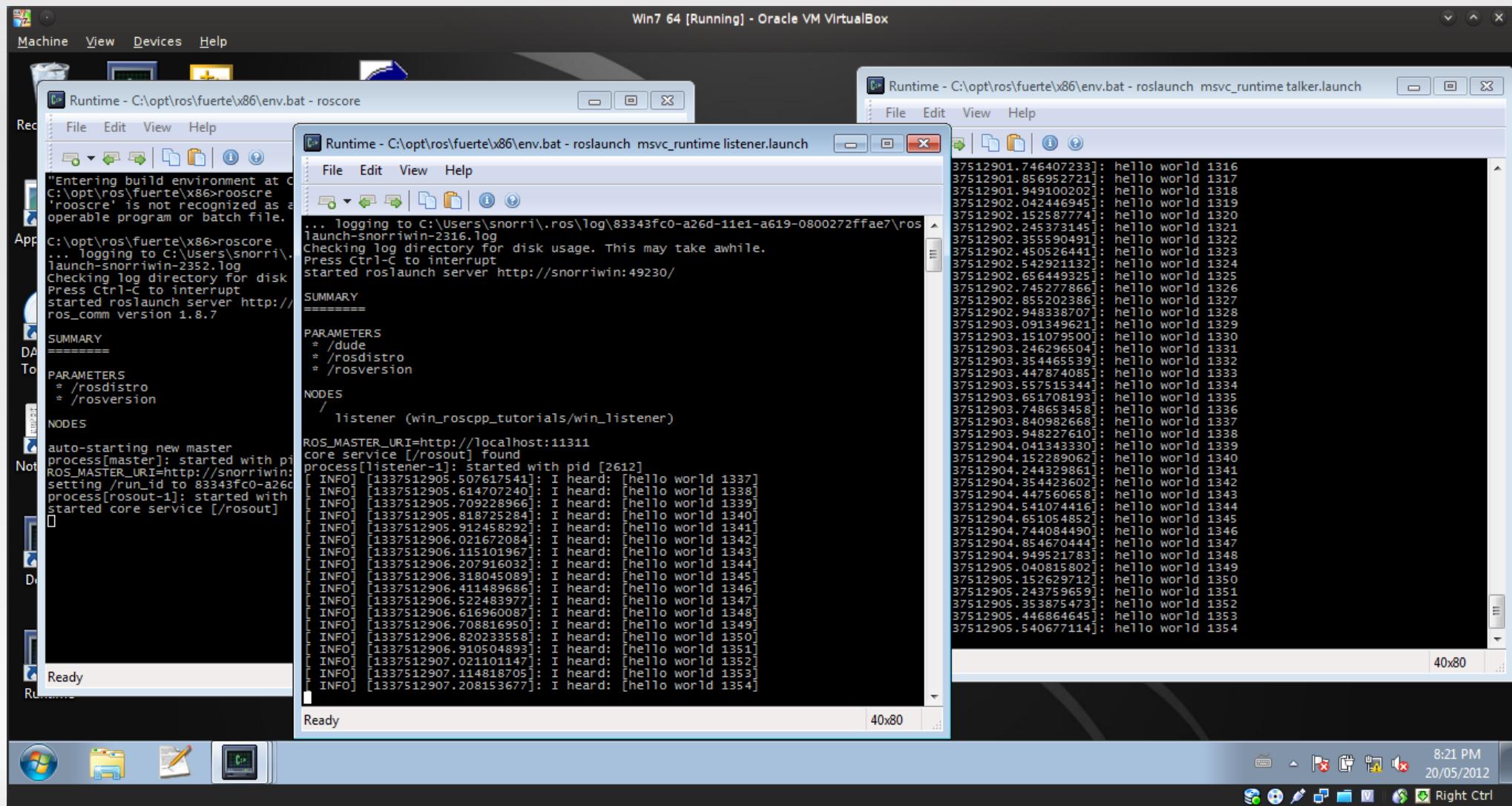
# Msvc Runtime

- Call env.bat in the build or sdk root
- Run ‘roscore’
- Run your project programs
- Introspect with ‘rostopic’ and co.

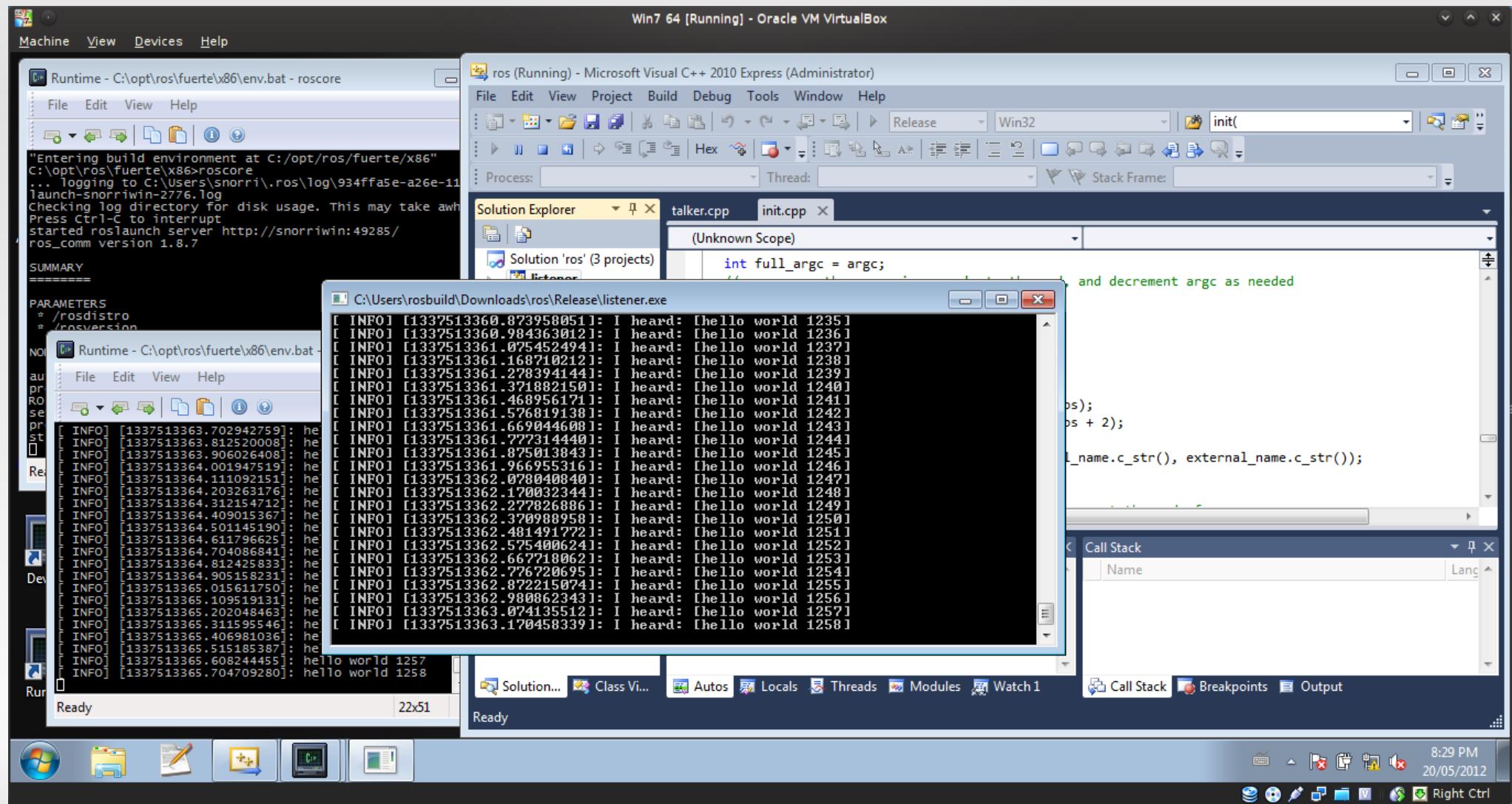
[http://www.ros.org/wiki/win\\_ros/Msvc Runtime Environment - Fuerte](http://www.ros.org/wiki/win_ros/Msvc%20Runtime%20Environment%20-%20Fuerte)

- Alternatively, if you build packages in the build environment, you can also take advantage of roslaunch/rosparam

# MSVC Results – Runtime Environment



# MsVC Results – Project + Debugging



# Msvc - Helping

## Build Environment

- Need a good knowledge of cmake, python, catkin
- Tools, 64bit, qt support, rosdeps, message generation

## Stack Porting

- Add stack to the win\_ros msvc\_fuerte rosinstaller
- Bugfix, port cpp code
- Put patches in win\_ros/win\_patches
- Ticket upstream

[http://www.ros.org/wiki/win\\_ros/Developing](http://www.ros.org/wiki/win_ros/Developing)

[http://www.ros.org/wiki/win\\_ros/Roadmap](http://www.ros.org/wiki/win_ros/Roadmap)

# Summary

- Linux devs + simple tools → Mingw Cross!
- Windows devs → WinRos SDK!

Will windows devs ever appreciate the ros build and runtime environment?

- Build environment is getting close (catkin baby!)
- Can really consider stack porting around Groovy '
  - But nothing to stop getting in now if you know what you are doing!