



# Multiplatform C++

Edouard Alligand, Founder

And now for something completely different...



# Does it blend?

```
#include <iostream>

int main(int argc, char ** argv)
{
    std::cout << "giggidy" << std::endl;
}
```

# Does it blend?

```
#include <iostream>

int main(int argc, char ** argv)
{
    std::cerr.sync_with_stdio(false);
    std::cout << "giggidy" << std::endl;
}
```

# Does it blend?

```
#include <iostream>

int main(int argc, char ** argv)
{
    const char * const char blah [] = "giggidy\n";
    fwrite(blah, sizeof(blah), stdout);
}
```

# Does it blend?

```
int main(int argc, char ** argv)
{
    #if BOOST_OS_WINDOWS
    #ifndef _DEBUG
        _set_abort_behavior(0, _WRITE_ABORT_MSG);
        _set_abort_behavior(1, _CALL_REPORTFAULT);
        _set_error_mode(_OUT_TO_STDERR);
    #endif
    #endif

    const char * const char blah [] = "giggidy\n";
    fwrite(blah, sizeof(blah), stdout);
}
```

# Does it blend?

```
int main(int argc, char ** argv)
{
#ifdef BOOST_OS_WINDOWS
#ifndef _DEBUG
    _set_abort_behavior(0, _WRITE_ABORT_MSG);
    _set_abort_behavior(1, _CALL_REPORTFAULT);
    _set_error_mode(_OUT_TO_STDERR);
#endif
#endif
    std::setlocale(LC_ALL, "en_US.UTF-8");
    const char * const char blah [] = "Привет!\n";
    fwrite(blah, sizeof(blah), stdout);
}
```

And now...



...let's get those compiler  
flags right



# The TWILIGHT ZONE

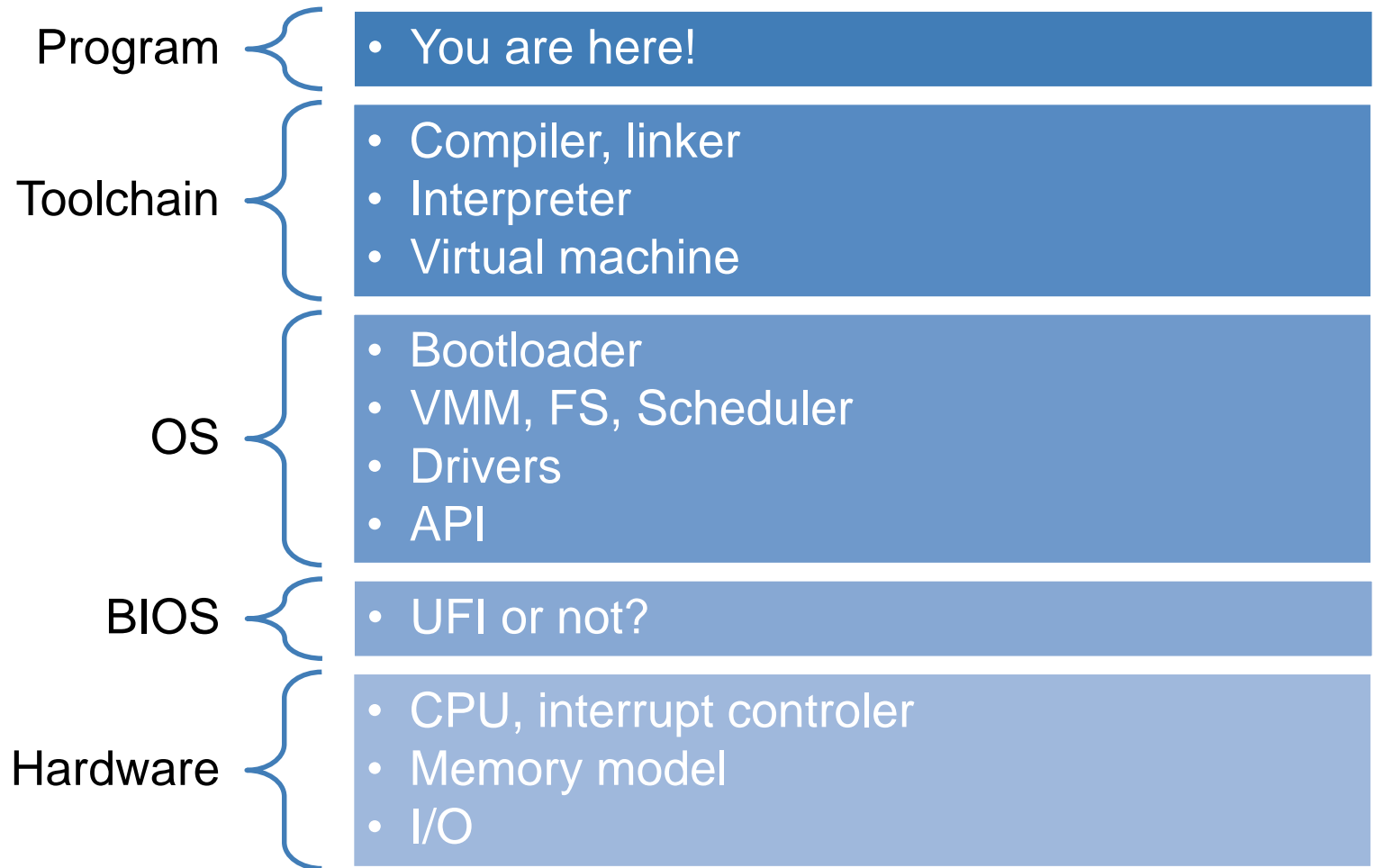
And now for something completely different...



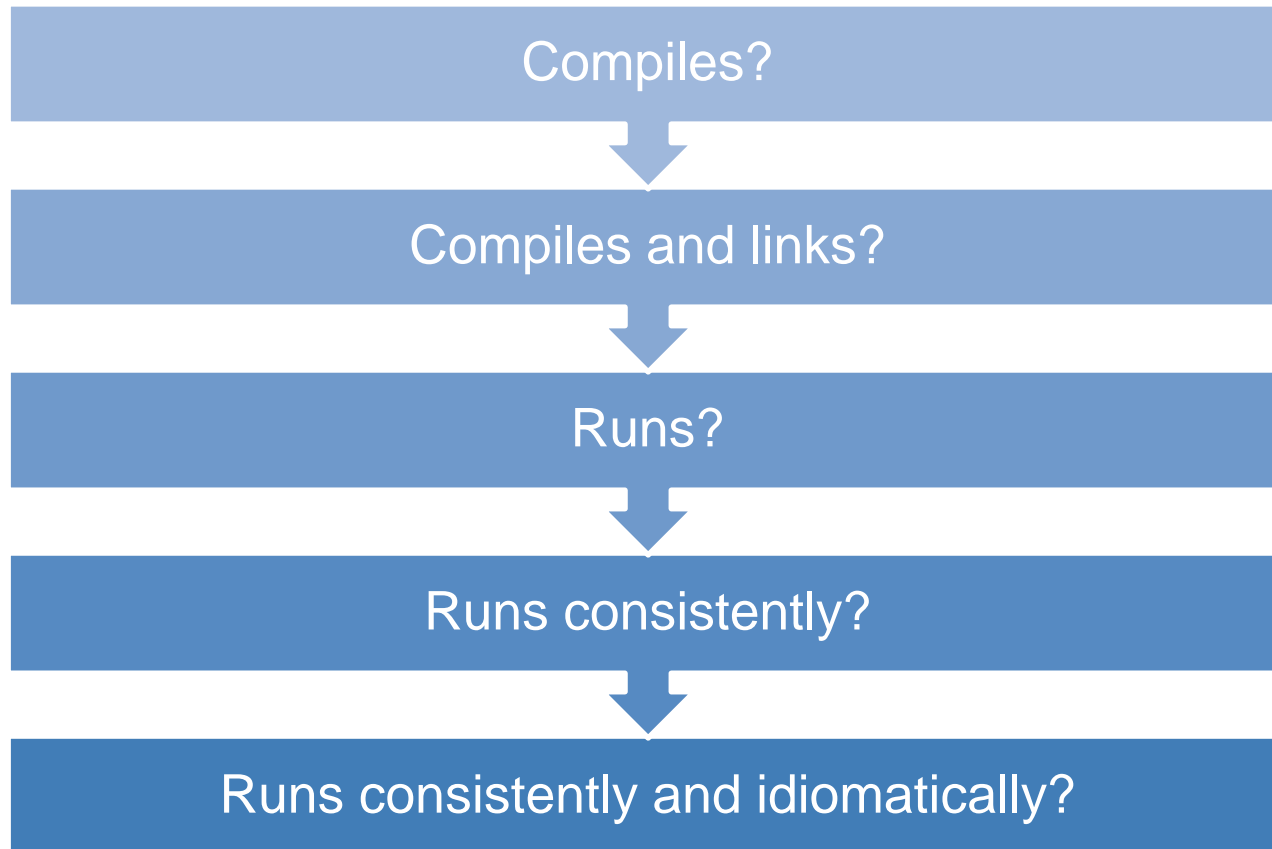
# What is writing software?



# You're always multiplatform



# What is a multiplatform C++ program?



And now for something completely different...



# Who we are

Fast!

ACID operations

Automatic distribution

Automatic replication

Unlimited storage



Transactional

Self-configuration

Multi-platform

Multi-language

And so much more!



# What we did

multiplatform



C/C++



multilangage



# How it builds

## Windows

- From XP to 8 32 and 64-bit
- Visual Studio 2012
- Dinkum's STL

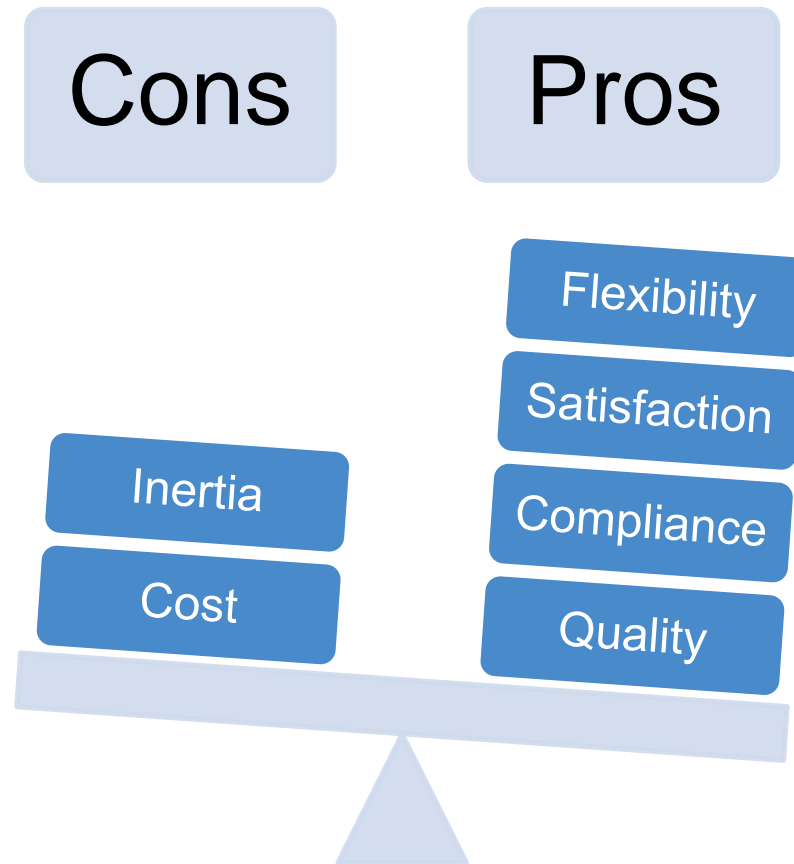
## FreeBSD

- 9.x 64-bit
- Clang 3.3
- libc++

## Linux

- 2.x – 3.x 64-bit
- Glibc 2.5+
- gcc 4.6.4
- libstd++

# Aftermath



# Topics not covered

MacOS

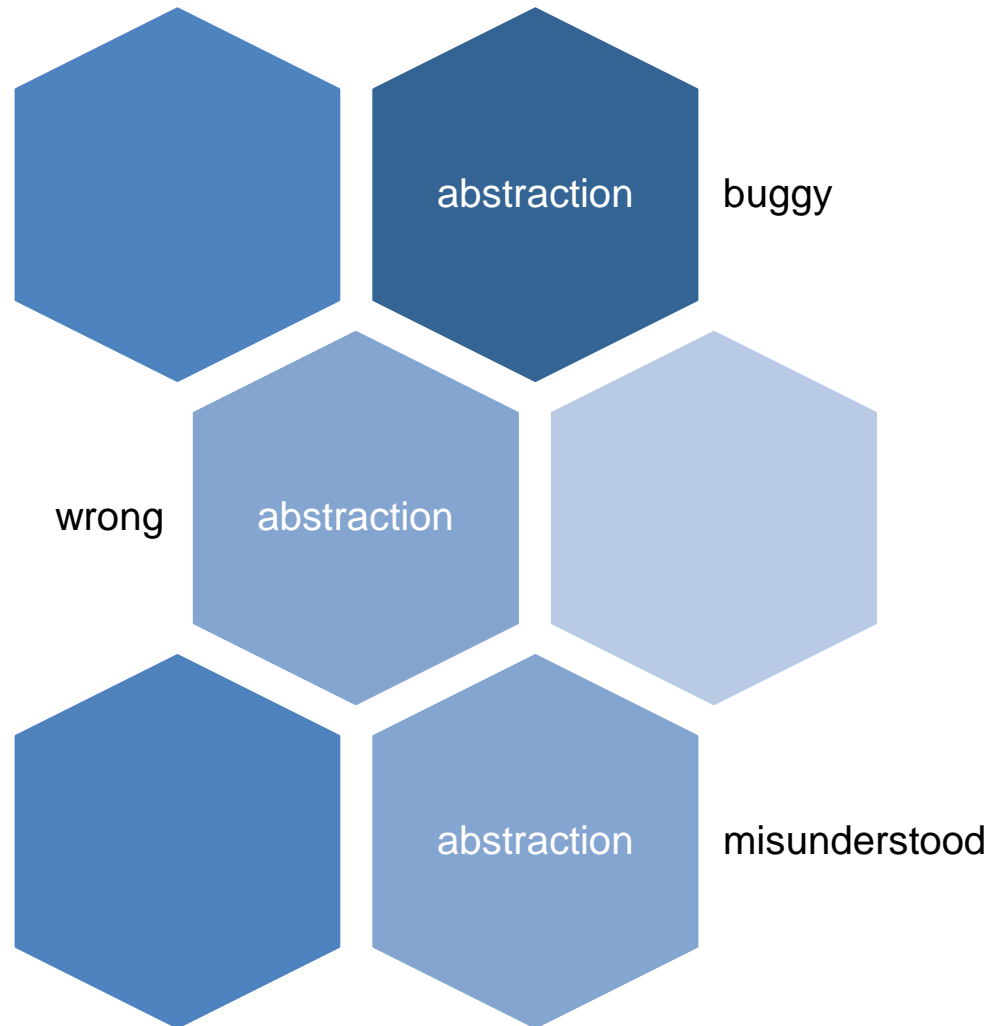
GUI

Editors

Metro

Telephones

# What is hard?



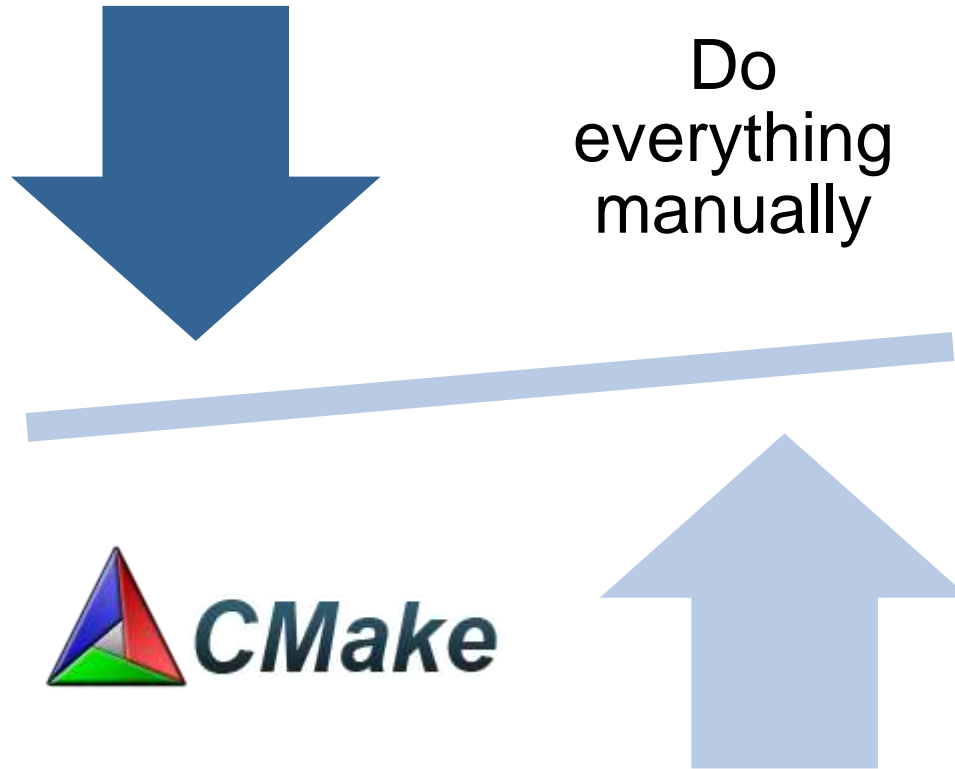
# Toolbox



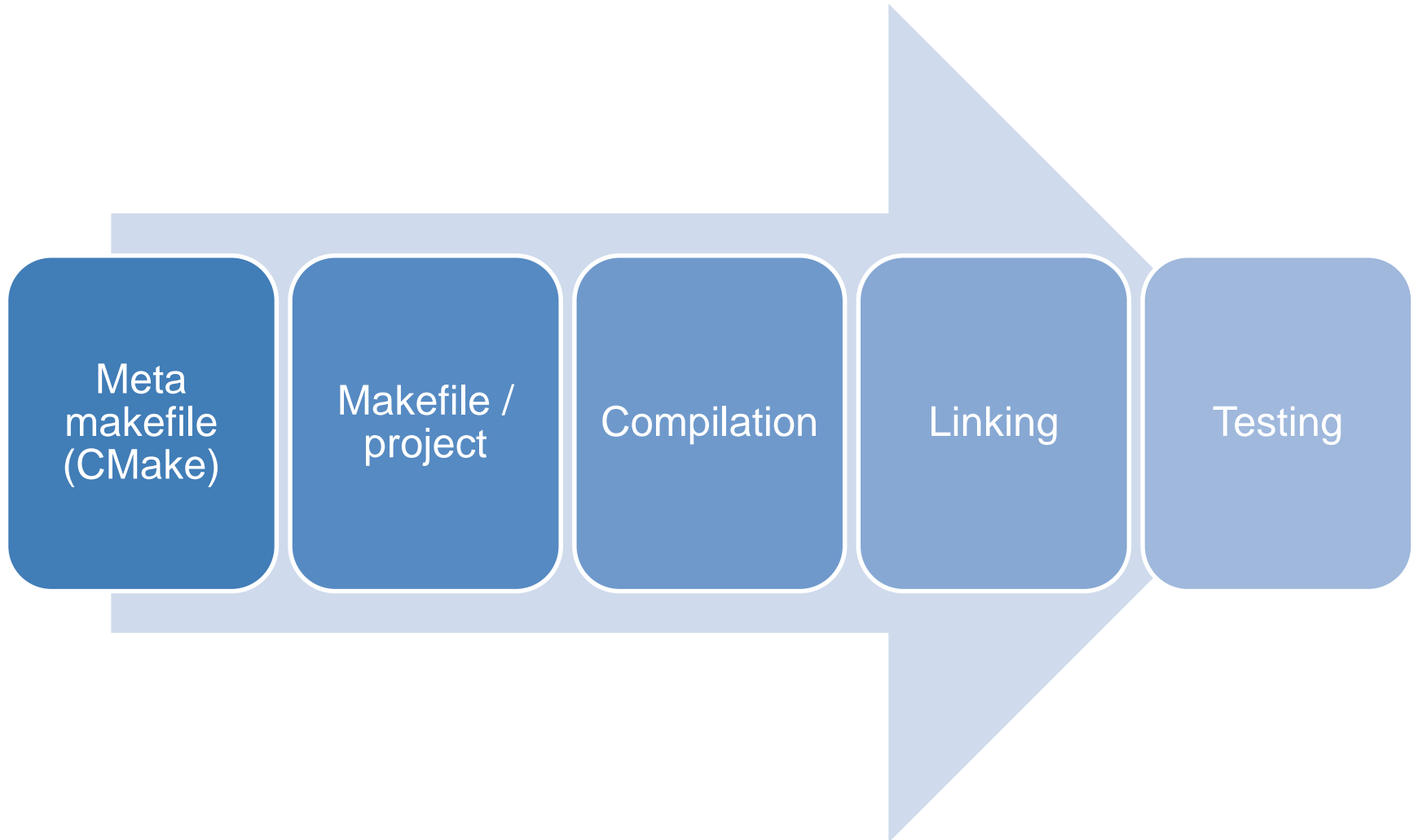
# Important trivialities

Encoding	<ul style="list-style-type: none"><li>• UTF-8</li><li>• BOM-less</li></ul>
Line ending	<ul style="list-style-type: none"><li>• LF (UNIX)</li><li>• Space only (no tabs)</li></ul>
File names	<ul style="list-style-type: none"><li>• No space</li><li>• Lowercase</li></ul>

# How to compile



# The process







But...

*...isn't C++ multiplatform?*

# STL

Recommended

<thread>  
<atomic>  
<chrono>  
<mutex>  
<system\_error>

<iostream>  
<fstream>  
<locale>

Careful!

# C to the rescue!

`open()`

`fopen()`

`memcpy()`

`memcmp()`

`blah()`

# You didn't expect it to be that easy, did you?

```
const unsigned char buf[1] = { 0 };
static_assert(sizeof(buf) == 1, "unexpected size");

#if BOOST_OS_WINDOWS
    int fd = ::_open("file", _O_BINARY | _O_RDONLY, 0);
    ::_lseeki64(fd, 0, SEEK_END);
    ::_write(fd, buf, sizeof(buf));
    ::_close(fd);
#else
    int fd = ::open("file", O_RDONLY, 0);
    ::lseek(fd, 0, SEEK_END);
    ::write(fd, buf, sizeof(buf));
    ::close(fd);
#endif
```

# Use Boost

Predef

ASIO

Date Time

Filesystem

Program  
options

Property  
trees

# Careful with third party libraries!

$$\mathcal{N}_{problems} > (\mathcal{N}_{libraries} \cdot \mathcal{N}_{toolchains} \cdot \mathcal{N}_{architectures})$$

# C++ features support

$$|\mathcal{F}_{clang} \cap \mathcal{F}_{gcc} \cap \mathcal{F}_{mvcc}|$$

# #if hell

- Cannot be avoided
- Use Boost.Predef
- Regroup! Abstract!





# Boost.Pref



- Header only and in Boost
- Externalizes the problem
- Simple macros

# Boost.Predef example 1

```
void func(void)
{
    #if BOOST_OS_WINDOWS
        // something Windows
    #endif
    #if BOOST_OS_BSD_FREE
        // something FreeBSD
    #endif
    #if BOOST_OS_LINUX
        // something Linux
    #endif
}
```

## Boost.Predef example 2

```
void func(void)
{
    #if BOOST_COMP_GNUC
        static_assert(BOOST_COMP_GNUC
                        > BOOST_VERSION_NUMBER(4, 0, 0),
                        "invalid gcc version");
    #endif

    #if BOOST_ARCH_X86_64
        // something AMD64
    #endif
    #if BOOST_ARCH_IA64
        // something IA64
    #endif
}
```

# Obvious Issues



# Compilation!

```
# Building mode at the end of script because of string "already defined operation" errors in boost
SET(CMAKE_CXX_DISABLED_WARNINGS "/wd4511 /wd4505 /wd4510 /wd4127 /wd4244 /wd4287 /wd4345 /wd4355 /wd4503 /wd4618 /wd4510 /wd4714 /wd4324 /wd4913")

# include(DisableAutomaticClassMembers)
# DetermineServicePackN("${CMAKE_SYSTEM}")
# message(STATUS "Detected: ${CMAKE_SYSTEM}")
# use ISO volatile behavior for better performance
# give that build flags
# we use Windows XP and later API on Windows
# 32x compiles boost
# CMT == 3 parallel compilations
SET(CMAKE_CXX_FLAGS_INIT "/volatile:isn /bigobj /MP3 /DNIN32 /D_WINDOWS /D WIN32_WINNT=0x0501 /DWINVER=0x0501 /DNTDDI_VERSION=0x0501R000 /DNIN32_LEAN_AND_MEAN")

# The will automatically use the USE_DEBUG if _DEBUG is defined
# CMAKE_SUPPRESS_DEVELOPER_WARNING is incompatible with the
SET(CMAKE_CXX_FLAGS_DEBUG_INIT "/Ox /Zd /Zc:forScope-1 /W4 /ZI /Gw /Os /RTC1 /D_SECURE_SCL=1 /D_HAS_ITERATOR_DEBUGGING=1")
SET(CMAKE_CXX_FLAGS_RELEASE_INIT "/MD /Ox /Ob2 /OI /OI /y /GR /GF /GS- /D_SECURE_SCL=0 /D_HAS_ITERATOR_DEBUGGING=0 /ORDERING")
# 32bit is no longer required as it's the default since MSVC11
SET(CMAKE_CXX_FLAGS_RELWITHDEBINFO_INIT "/MD /ZI /Ox /Ob2 /OI /OI /y /GR /GF /GS- /D_SECURE_SCL=0 /D_HAS_ITERATOR_DEBUGGING=0 /ORDERING /OTHER_O0_THREADING_T0 (/MSVC)")

if(RSBOX MATCH "clang" CLANG ${CMAKE_CXX_COMPILER})
    CLANG
    # we're using clang!

    # we prefer to exclude our clang libraries because we might have a different Clang version than the FreeBSD on which amd
    # is used
    # starting with FreeBSD 10.1 it will be much more simple as FreeBSD 10.0.4 will be fully clangized
    # we don't include libgcc_s and libstdc++ because we use the system's defaults
    FIND_FILE(CLANG_RT libckcrt.so.1 PATHS /usr/local/lib/ /usr/lib/ /lib/ ENV LIB)
    SET(FILENAME_COMPONENT(ACTUAL_CLANG_RT ${CLANG_RT} REALPATH)
    FIND_FILE(CLANG_STDCPP libc++.so.1 PATHS /usr/local/lib/ /usr/lib/ /lib/ ENV LIB)
    SET(FILENAME_COMPONENT(ACTUAL_STDCPP ${CLANG_STDCPP} REALPATH)
    SET(CLANG_ADDITIONAL_LIBS ${ACTUAL_CLANG_RT} ${ACTUAL_STDCPP})

    INSTALL(PROGRAMS ${CLANG_ADDITIONAL_LIBS} DESTINATION lib)

    # -gnu-linkage -gnu-virtual-dtor so otherwise we get a lot of error for shared_ptr
    # -gnu-unused-variable == gch translator
    # -gnu-char-subscript == using char for subscript can be a problem?
    # -gnu-parentheses-equality == no warning if there are too many parentheses
    SET(CLANG_WARNINGS "-Wall -gnu-used-function -gnu-delete-non-virtual-dtor -gnu-unused-variable -gnu-char-subscript -gnu-parentheses-equality -gnu-nested-a")

    # we need to tell clang to link against libcxx otherwise exceptions won't work
    SET(CMAKE_CXX_FLAGS_INIT "${flags}+libc++ -fcxx-exceptions -fexceptions -fvirtuality=hidden -fvirtuality=inline-hidden -D__PREVIEW_MEMORY_POOL__")
    SET(CMAKE_CXX_FLAGS_RELEASE_INIT "-O2 -DNDEBUG")
```

- Different compilers
  - Different errors
  - Different options
  - Different macros

# Windows vs UNIXes – Some major differences

## Windows

UTF-16

Drive letters, UNC

GUI

Local library 1st

Locks files like there is no tomorrow

## UNIX

Depends

Mount points

Terminals

System library 1<sup>st</sup>

Rarely locks files

# Paths

`C:\Users\Edouard\AppData\Roaming\My Application\Settings`

---

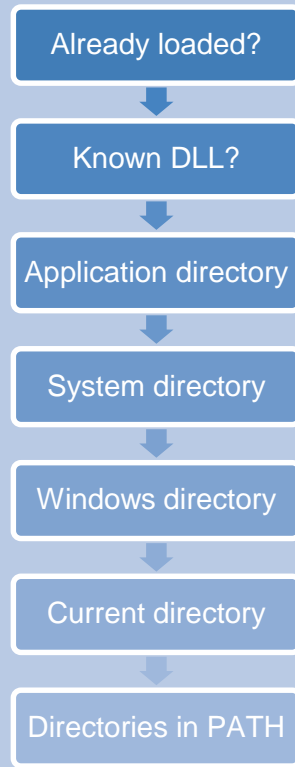
`\\MyServer\Share\Music`

---

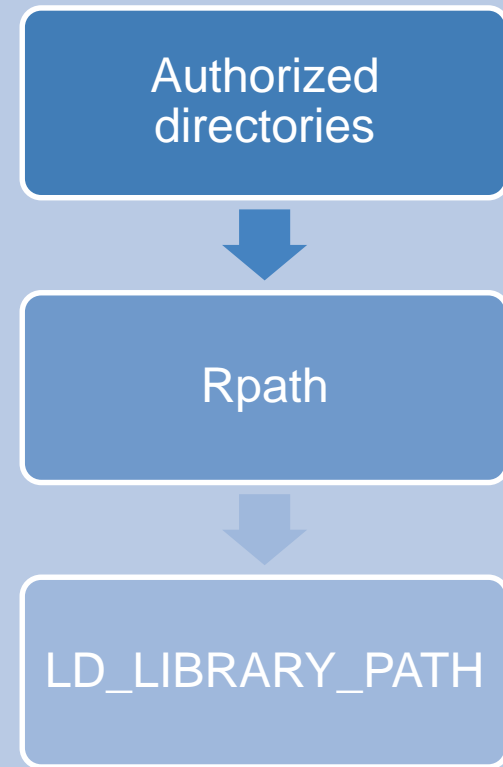
`~edouard/.app`

# Library search order

## Windows

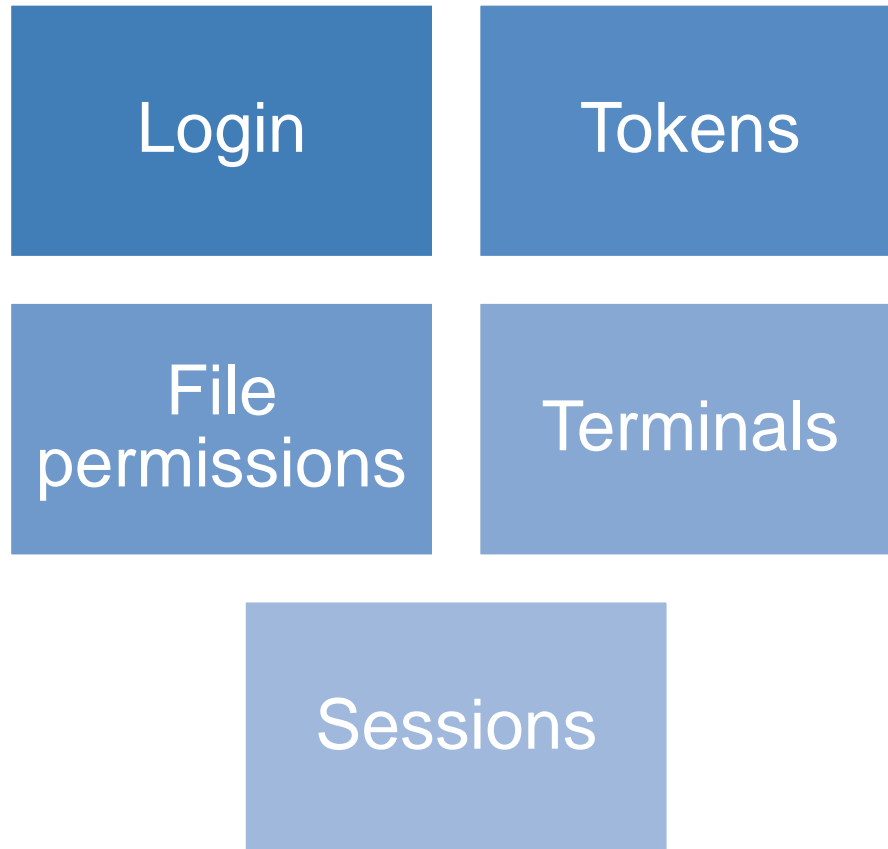


## UNIX

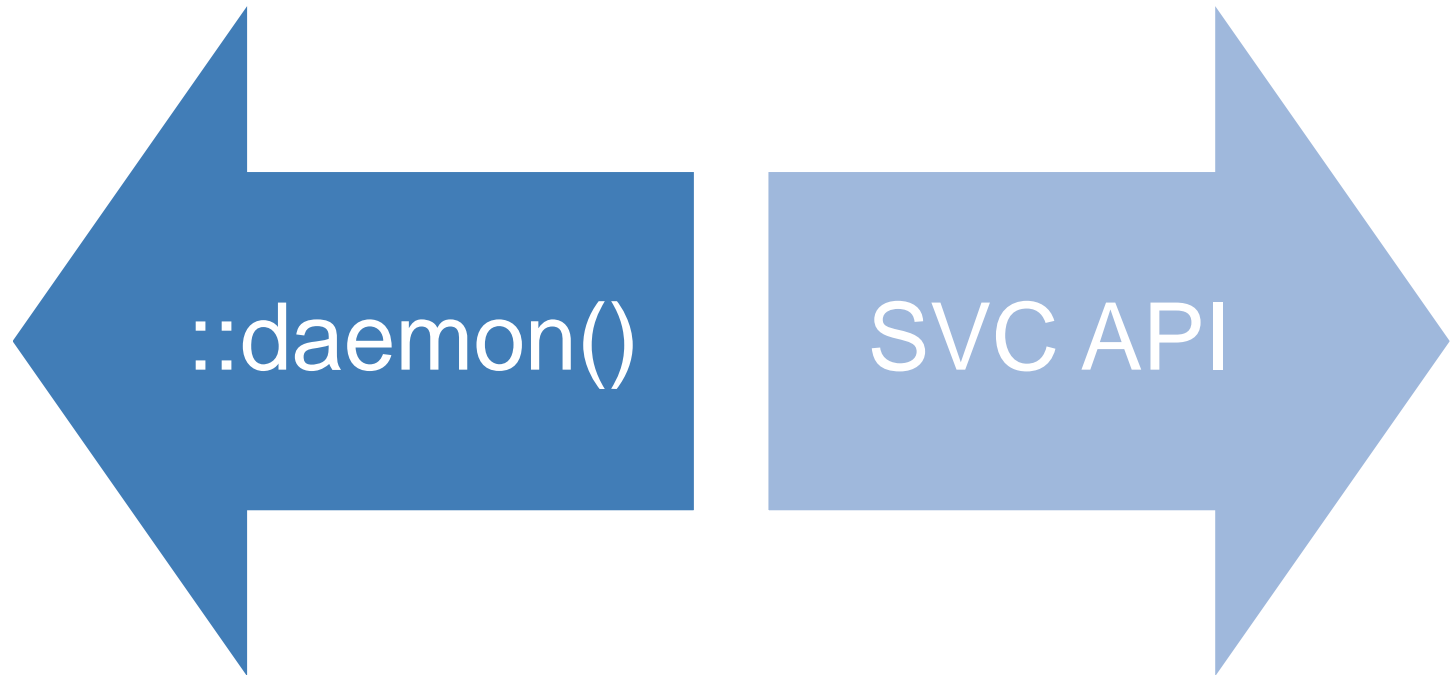




# Credentials

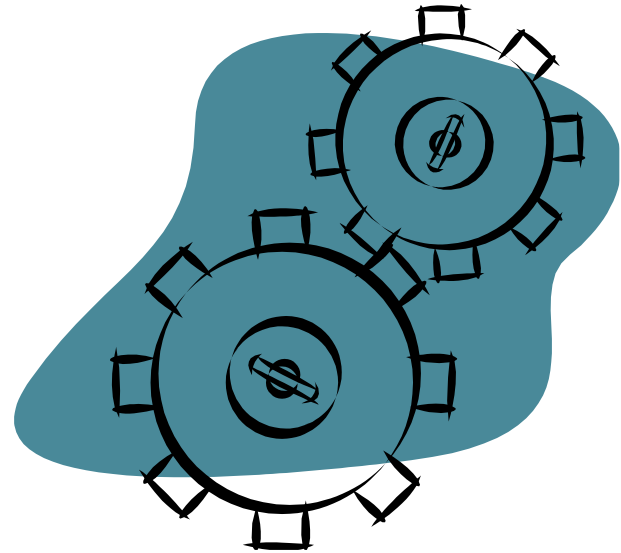


# Daemons



# Configuration

- /proc
- Windows registry
- sysctl
- Configuration files nightmare



# Multithreading C++ 11

Use

`std::thread`  
`std::*mutex`  
`std::condition_variable`

`std::async`  
`std::future`

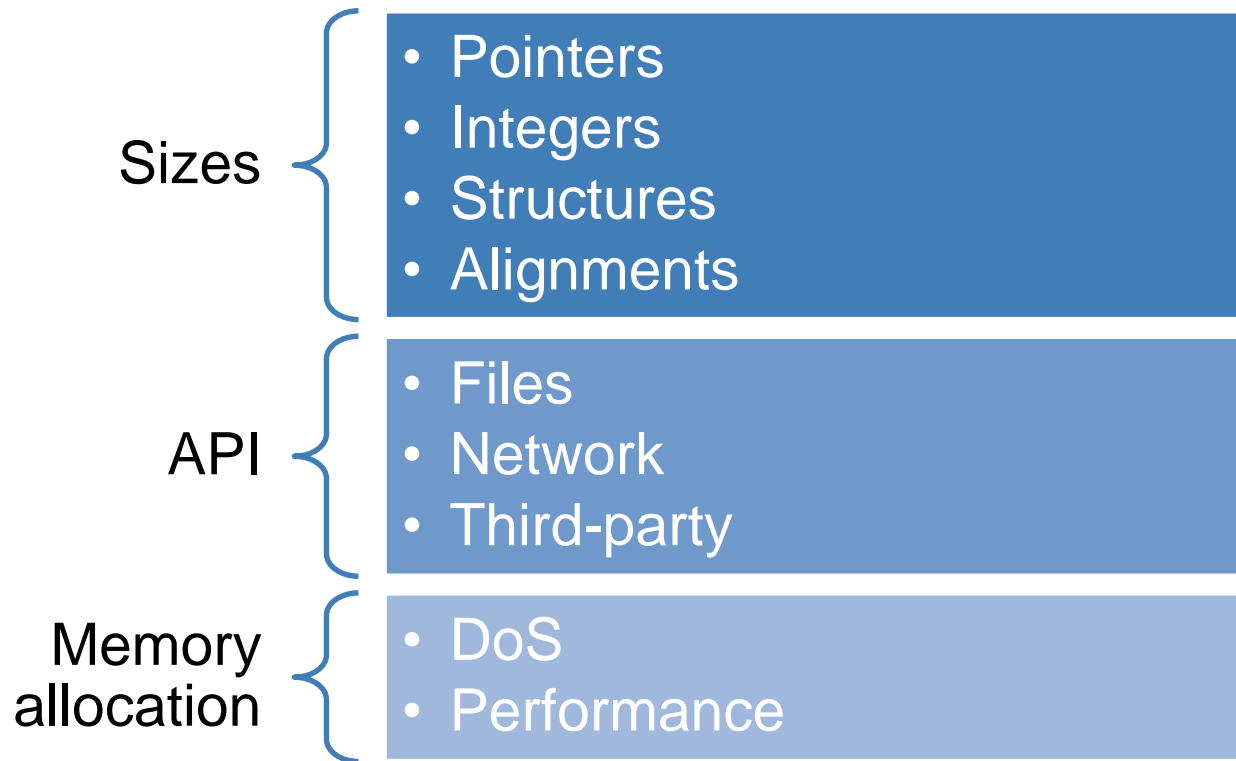
Avoid

# Serialization

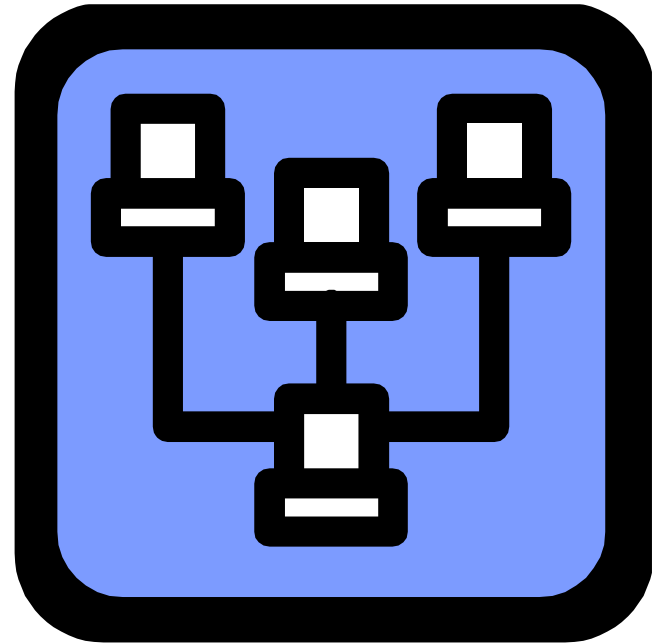
- Endianness
- Floats
- Alignment
- Sizes



# 32-bit vs 64-bit



# Networking



Create your own library...  
...if you like pain and failure.

# Boost.ASIO custom socket option example

```
#if BOOST_OS_WINDOWS
// on Windows we use the better and more secure
// SO_EXCLUSIVEADDRUSE option
int optval = 1;
auto native_socket = acceptor.native_handle();
if (::setsockopt(native_socket,
    SOL_SOCKET,
    SO_EXCLUSIVEADDRUSE,
    reinterpret_cast<const char *>(&optval), sizeof(optval)) != 0)
{ /* error management */ }
#else
acceptor.set_option(boost::asio::ip::tcp::acceptor::reuse_address(true));
#endif
```

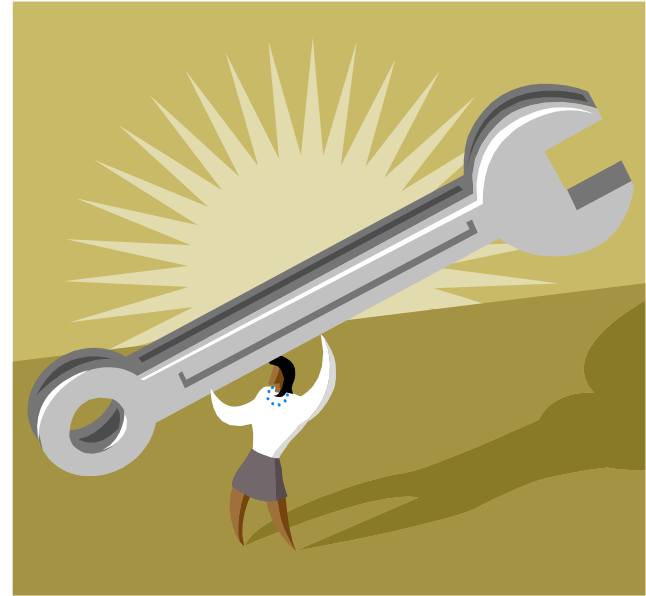


# Subtle Issues



# Debugging

- `DEBUG=1`
- `_DEBUG=1`
- `_SECURE_SCL=1`
- `_HAS_ITERATOR_DEBUGGING=1`
- `_GLIBCXX_DEBUG=1`



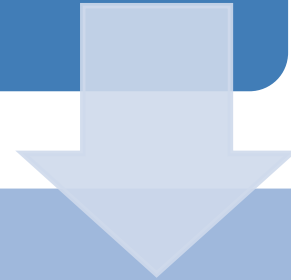
# Localization

- Character set
- Reading direction
- Language
- Time zone
- Currencies



# High resolution timestamp on UNIX

gettimeofday()



arithmetics

# High resolution timestamp on Windows

QueryPerformanceFrequency



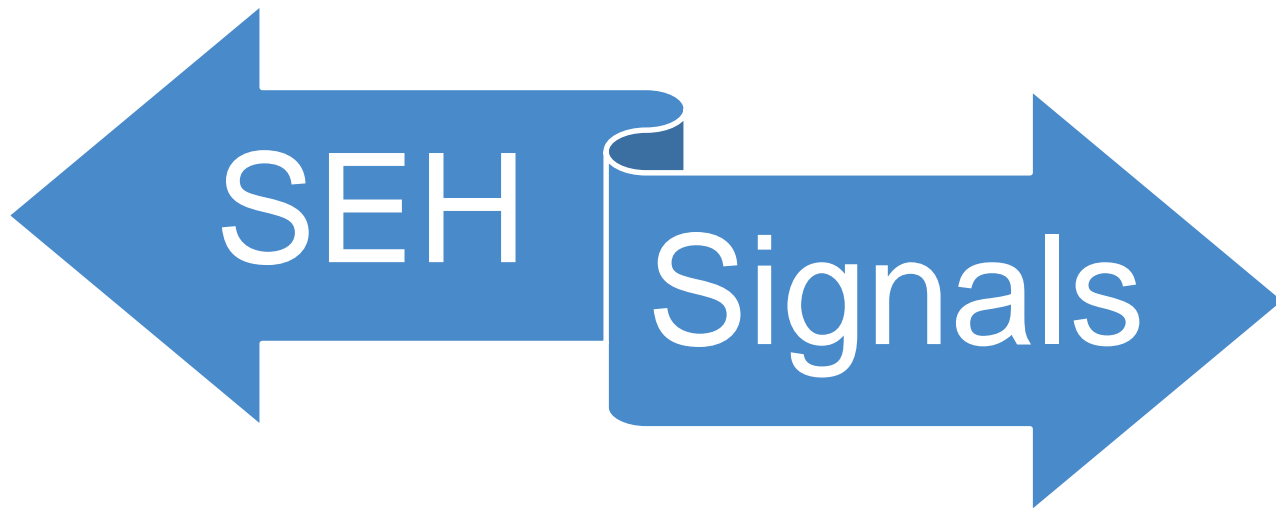
QueryPerformanceCounter

Arithmetics

QueryPerformanceCounter

Arithmetics

# Error management

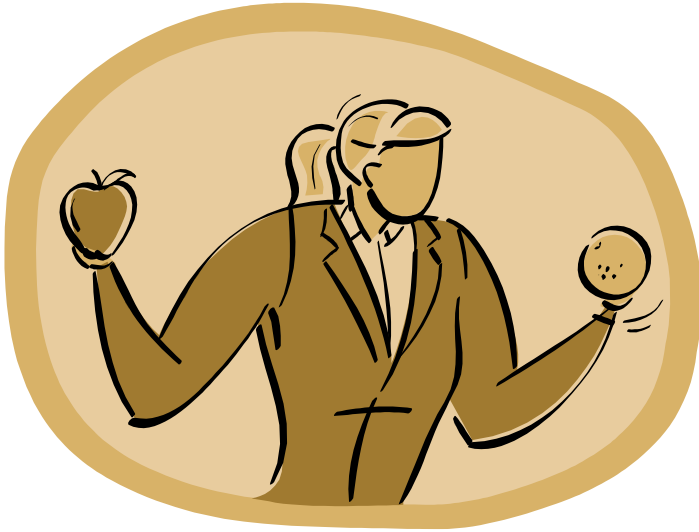


# Differences within the same OS

- Linuxes jungle
- FreeBSD versions incompatible
- NetBSD != OpenBSD != FreeBSD
- Windows 95, 98, Me, 2000, NT, Vista, 7, 8...



# UNIX differences



- **Unavailable functions**
  - `backtrace()`, `fread_unlocked()` (FreeBSD)
- **Different configurations**
- **Different parameters**
  - `statfs()` (FreeBSD vs Linux)
  - `sockets` (Old UNIXes)
- **Different libraries**
  - `epoll()` vs `kqueue()`
  - `libc++` vs `stdlibc++`
  - `glibc` versions



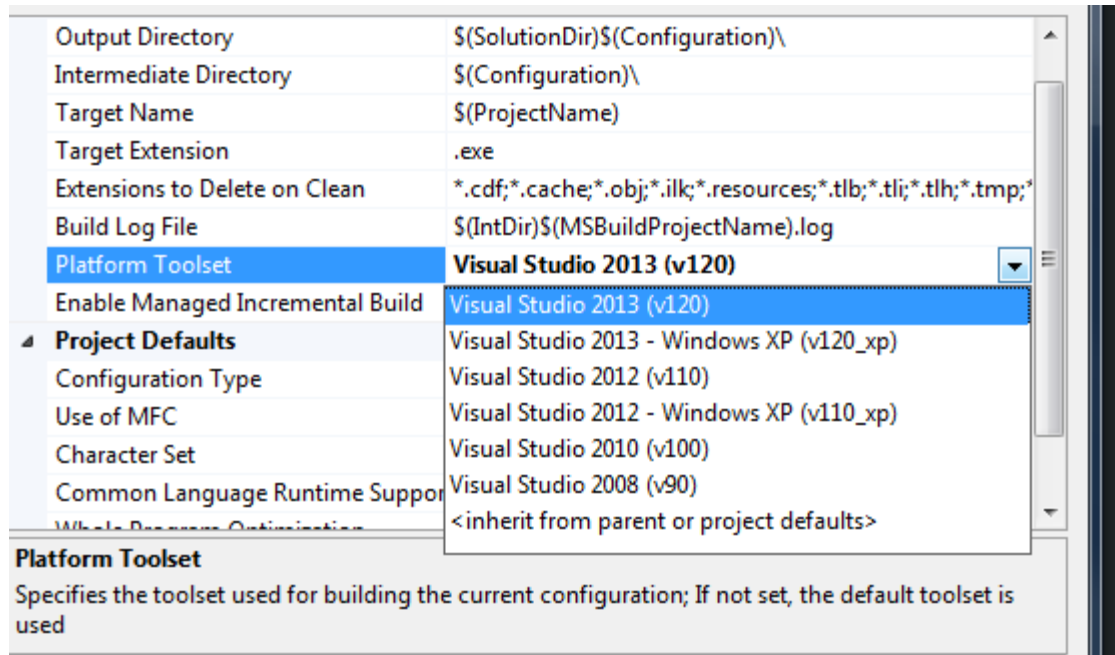
# Supporting different Windows versions

- `_WIN32_WINNT=0x0501`
- `WINVER=0x0501`
- `NTDDI_VERSION=0x05010300`



# Windows XP support

- In CMake
  - « -T » switch (for example -Tv120\_xp)
- In Visual Studio



# Hard™ Problems



# Memory allocation

- Performance
- Fragmentation
- Scalability



# Filesystem

- Performance
- Fragmentation
- Cluster size
- Features (transactions ?)



# Asynchronous I/O

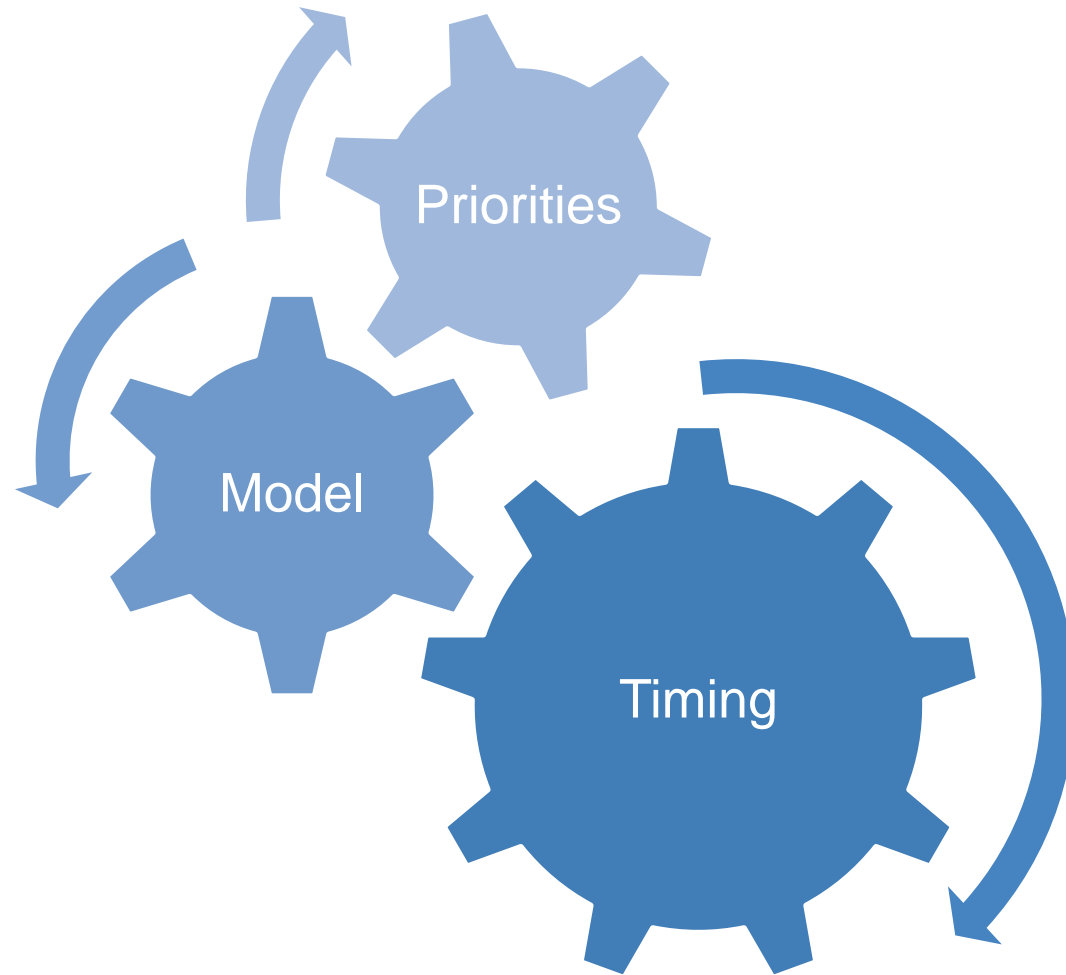
- POSIX: aio(7)
- FreeBSD: kqueue(2)
- Linux: epoll(7)
- Windows: I/O completion ports
- Or use Boost.ASIO!



# Performance discrepancies

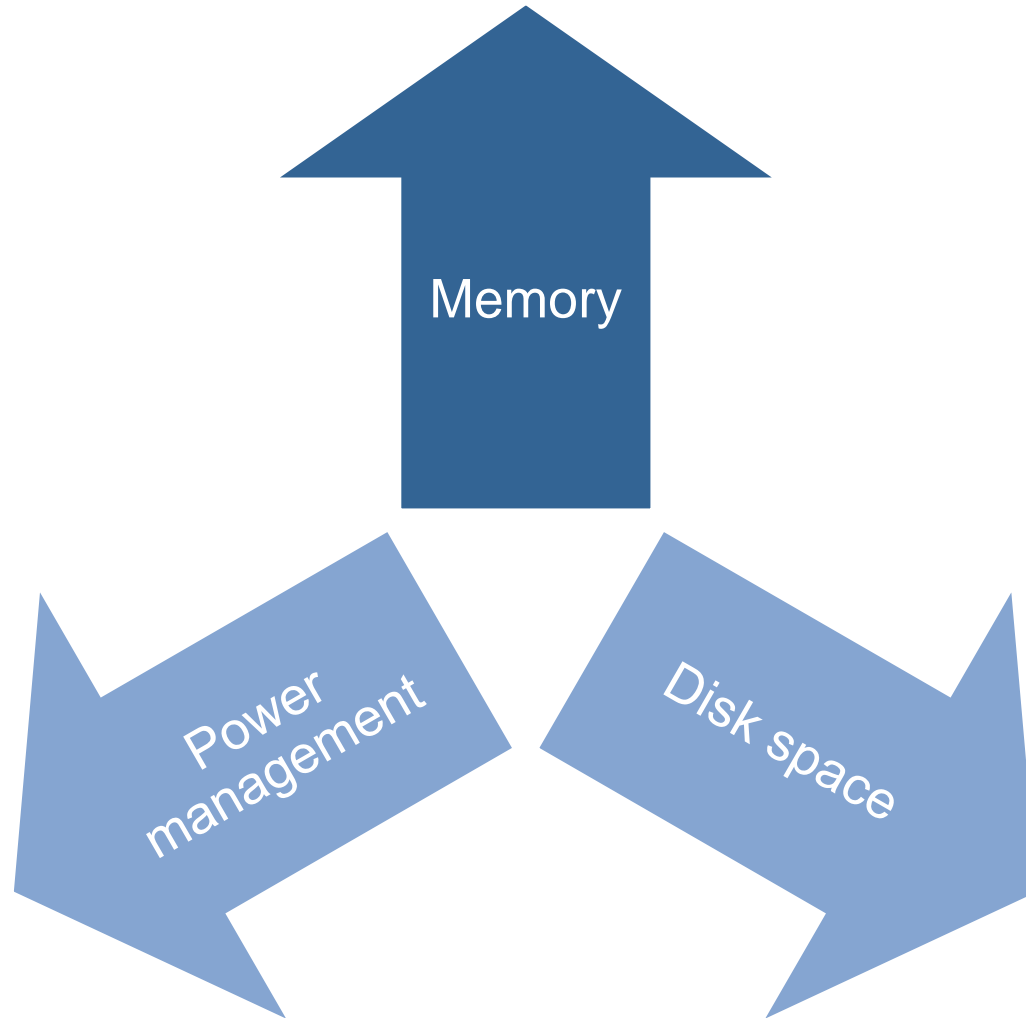


# Multithreading (hard)





# When it goes wrong



# Tools of the trade

## **The Boost libraries**

<http://www.boost.org/>

---

## **CMake**

<http://www.cmake.org/>

---

## **Buildbot**

<http://buildbot.net/>

---

## **Intel Threading Building blocks**

<http://threadingbuildingblocks.org/>

---

## **Valgrind**

<http://valgrind.org/>

---

## **Microsoft Application Verifier**

<http://www.microsoft.com/en-us/download/details.aspx?id=20028>

# Questions and answers



<https://www.quasardb.net/>  
<https://www.bureau14.fr/>  
@edouarda14  
edouard@bureau14.fr