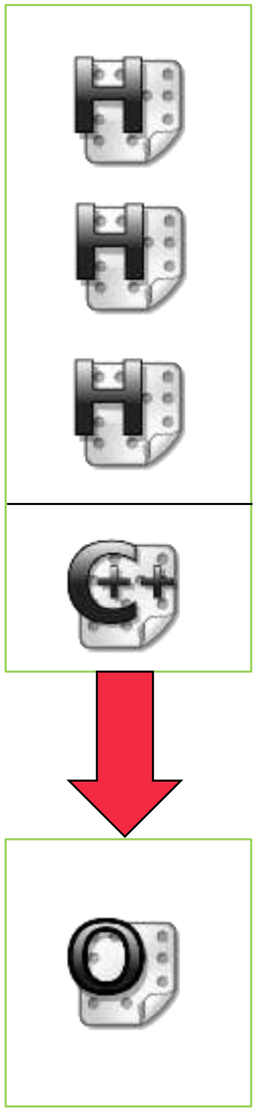
05.05.10

Shared Libraries

Dynamic Load Libraries in MITK

GERMAN
CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

A single compile process

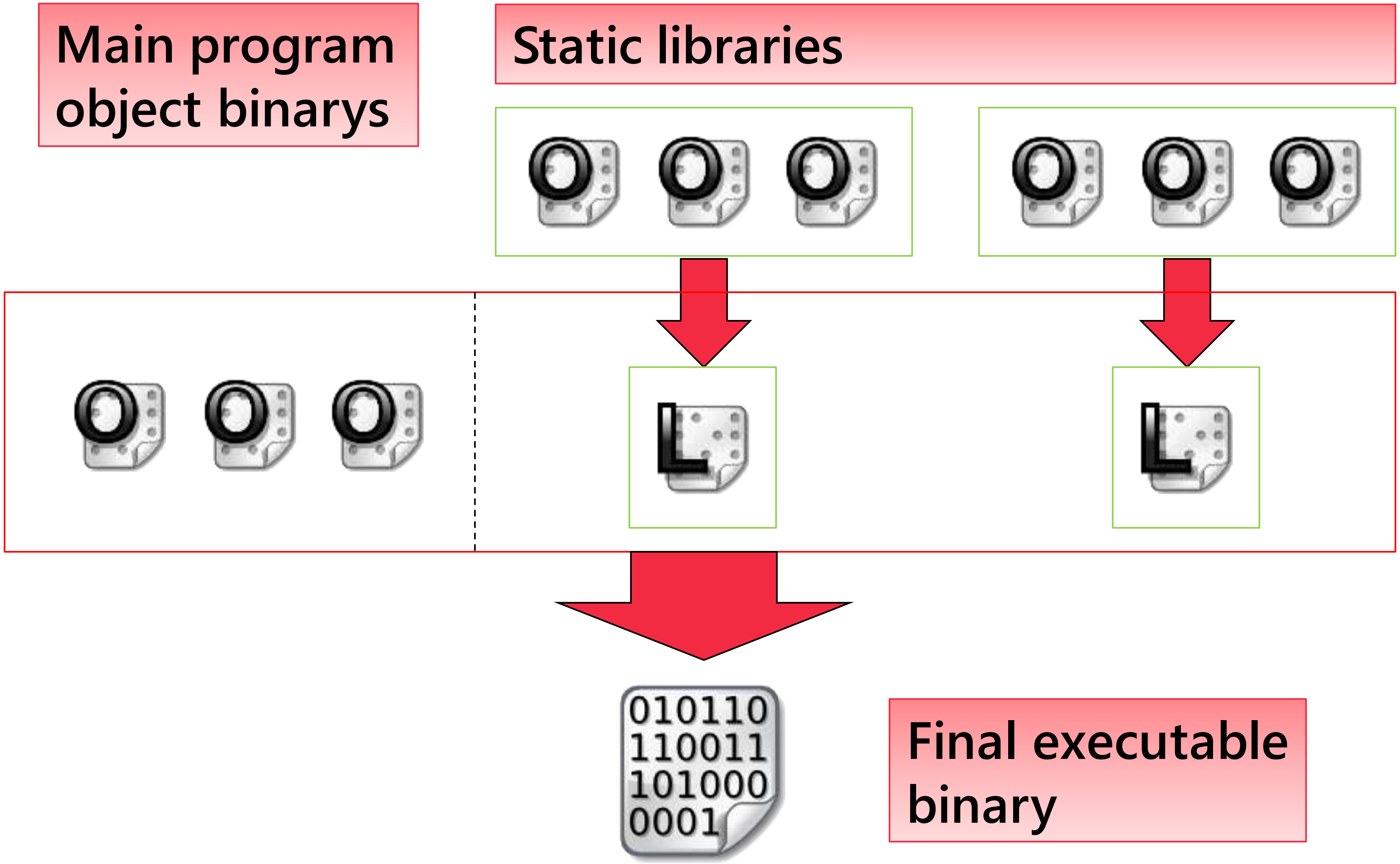


Multiple header files from this and other classes that this source .cpp depends on

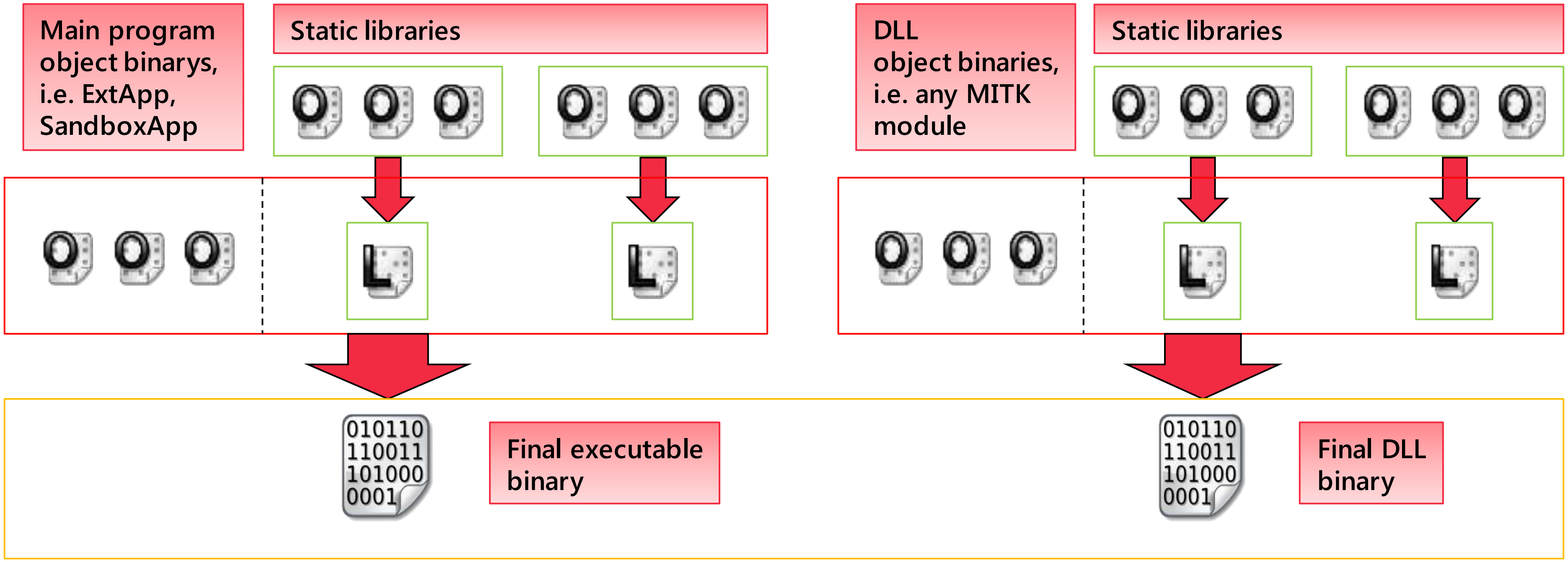
The source .cpp to be compiled

Generated object binary
- Contains almost final compiled machine code
- References to other object binarys still need to resolved by the linker

Link process to generate final executable



Now with DLLs



Application binaries are linked at runtime

Problems

- Windows requires to explicitly specify which methods/global variables are exported/imported from a DLL
- Singletons in static libraries

Singleton problem

- if multiple dlls or the executable include a static library with a singleton, it will be also multiple times instantiated.
- Problems arise, if objects are exchanged between the dlls and the executable, that depend on the singleton.
- i.e. Lists of allocated objects, global time for modification detection.
- A simple solution is, use only a „single“ singleton by consolidiating the static library into a single dll, where other dlls or the executable import from.
- However, its best to change design and to not depend on singletons at all

Why shared libraries

- Statically linked libraries / executables:
  - Linked at generation
  - Each executable contains the full binary code of the static library (causing redundancy on disk and memory)
- Dynamically linked libraries:
  - Linked at runtime
  - The binary code can be shared across multiple executables (saving disk space)
  - This works in process virtual adress space, too (saving memory).
    - read-only pages are automatically shared across multiple processes.
    - writable pages may be manually shared by the DLL.
  - Allows optional dynamic loading of plugins/bundles etc..

The MITK solution to the im/export problem

- Cmake generates special headers for each module
- From MitkExt:

```
#ifndef MitkExt_EXPORTS_H
  #define MitkExt_EXPORTS_H
  #if defined(WIN32)
    #ifdef mitkCoreExt_EXPORTS
          #define MitkExt_EXPORT __declspec(dllexport)
    #else
          #define MitkExt_EXPORT __declspec(dllimport)
    #endif
  #else
    #define MitkExt_EXPORT
  #endif
  #ifndef _CMAKE_MODULENAME
    #ifdef mitkCoreExt_EXPORTS
      #define _CMAKE_MODULENAME "MitkExt"
    #endif
  #endif
#endif
```

The header in use

```
#include "MitkExtExports.h"

class MitkExt_EXPORT MovieGenerator : public
itk::LightObject
{
public:
  int Method(int x)
  {
      return x;
  }
  ...
};
```

Problems with templates

```
template <class T> class ~~MitkExt_EXPORT~~
SampleTemplateClass
{
public:
        T TestMethod(T x)
        {
                return x*x;
        }
     ...
};
```

Do not use the Export Macro with templates. Templates are first instantiated when required. Visual C++ will produce linker errors, cause it assumes, that a template specialization can be imported from a DLL, which doesn‘t provide it.