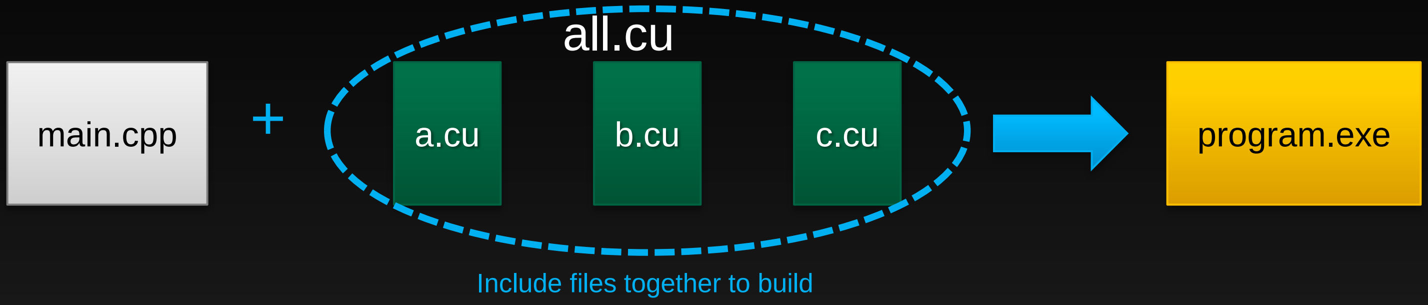
Separate Compilation in CUDA 5.0

by Mike Murphy

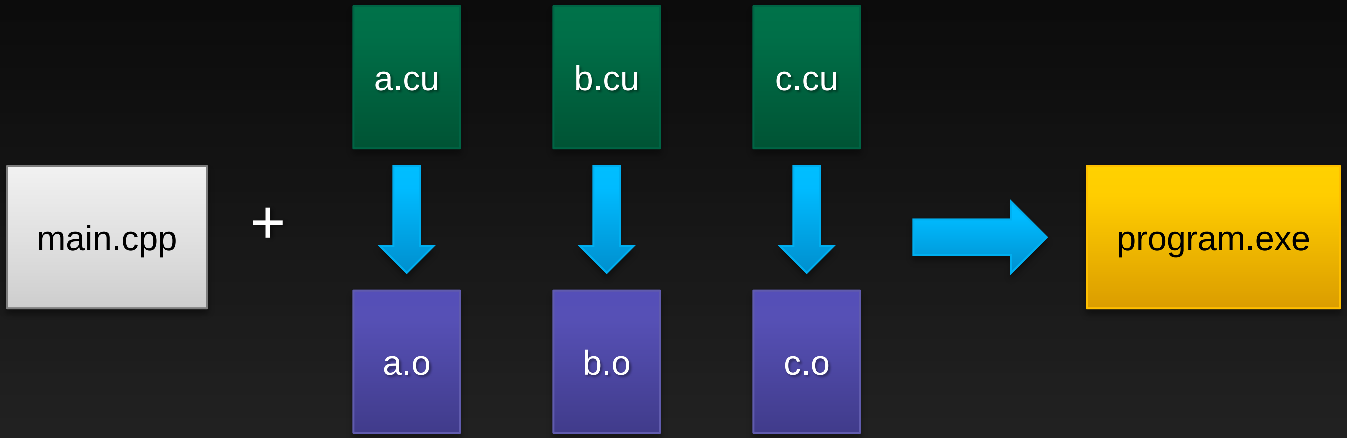
No Separate Compilation in earlier releases



Earlier CUDA required single source file for a single kernel

No linking external device code

CUDA 5: Separate Compilation & Linking



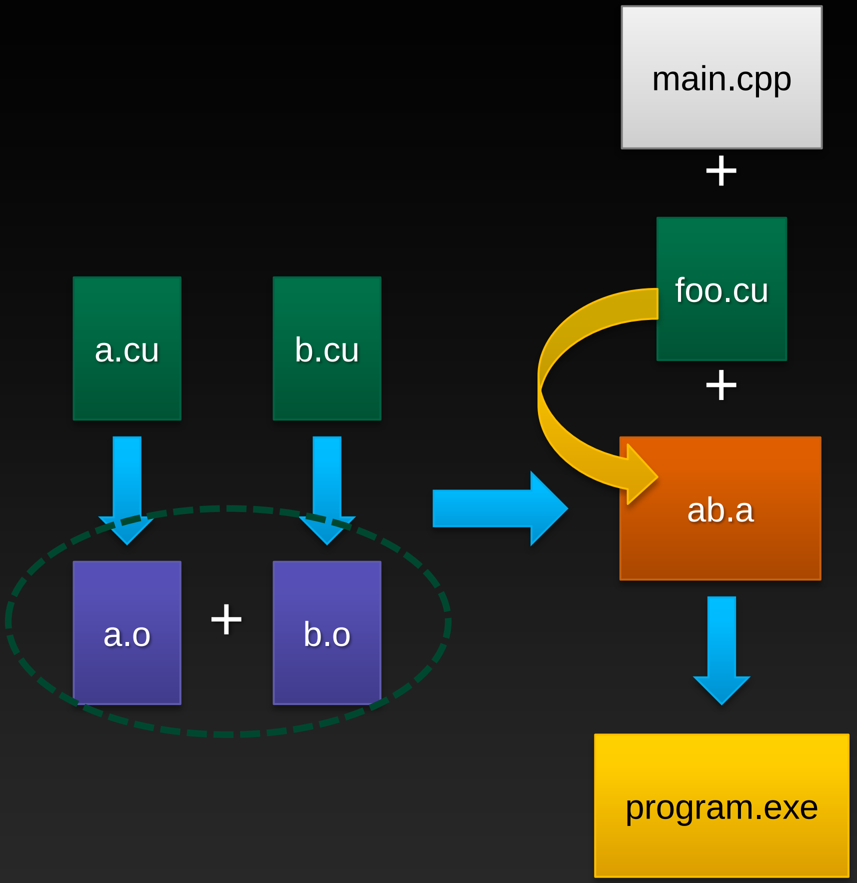
Separate compilation allows building independent object files

CUDA 5 can link multiple object files into one program

Benefits of Separate Compilation

- Eases porting code
  - no longer have to include files together
  - “extern” attribute is respected
- Incremental compilation reduces build time
  - e.g. 47000 line app used to take 50 seconds to build, now when split into multiple files takes 4 seconds to build if only one file changed
- Can create and use 3rd party libraries

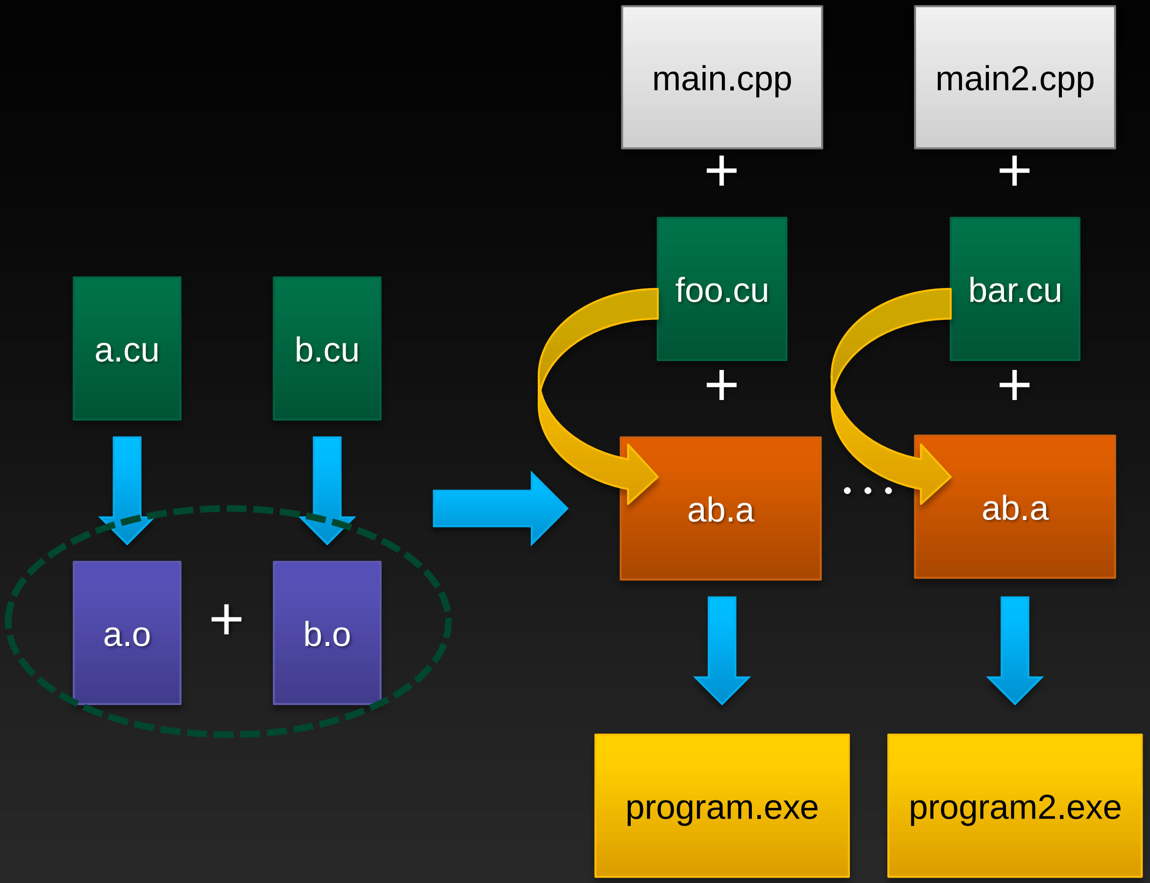
CUDA 5: Library Support



Can combine object files into static libraries

Link and externally call *device* code

CUDA 5: Library Support



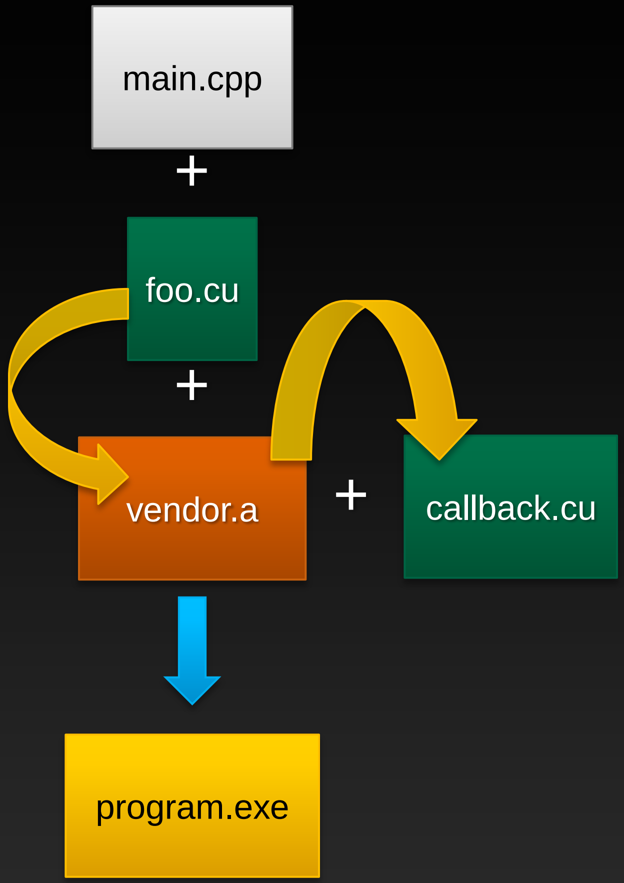
Can combine object files into static libraries

Link and externally call *device* code

Facilitates code reuse, reduces compile time

CUDA 5: Callbacks

Enables closed-source device libraries to call user-defined device callback functions



Separate Compilation Features

- SM_2x and above (Fermi & Kepler, no support for sm_1x)
- All platforms (Linux, Windows, and MacOS)
- All CUDA features
- Optimized and Debug (-G) compilations
- Support both previous whole-program compilation and new separate compilation.
  - Default is whole-program compilation, have to opt in to separate compilation.

Libraries

- Can link static host libraries (.a,.lib) that contain device code
- Shared libraries (.dylib,.so,.dll) are ignored by device linker
- libcublas_device.a is linkable device library that we ship and is used for dynamic parallelism

Example usage

- nvcc -arch=sm_20 -dc *.cu
  - -c is used for host compile to object, so invented -dc
  - -dc == --device-c == --relocatable-device-code -c
  - Without -dc we default to old whole program compilation
- nvcc -arch=sm_20 *.o
  - Device linker is implicitly run for sm_20 and above, but does nothing if does not find relocatable device code.

- If want to use host linker:
- nvcc -arch=sm_20 *.o -dlink -o link.o
  - create new object; -dlink == --device-link
- g++ *.o -lcudart
  - link all objects, including new link.o
  - CUDA host objects must be passed to both device and host linkers

Demo

Multiple Device Links

- Can do multiple device links within a single host executable
  - nvcc a.o b.o -dlink -o link1.o
  - nvcc c.o d.o -dlink -o link2.o
  - g++ a.o b.o c.o d.o link1.o link2.o
- Useful when separate code sections
  - Similar to how we previously allowed multiple device modules in a single host executable (x.cu and y.cu)
  - If library writer wants to device-link some code together, then user can still invoke device linker on own code
  - Can reduce resource requirements, e.g. if function pointers then may assume that code from another section is reached, and thus require more registers than really needed

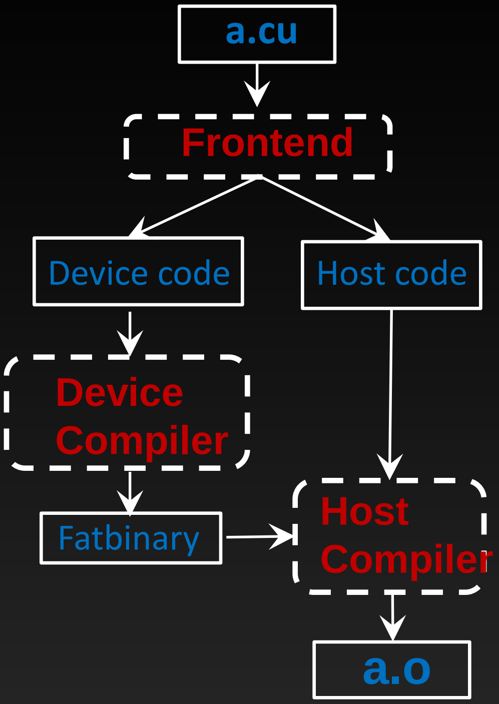
Compatibility warning

- Current 5.0 linker will not JIT to future architectures
  - SASS is linked, not PTX
  - PTX can be input to linker, but is first compiled to SASS then linked
- Must relink objects for each architecture
  - nvcc -arch=compute_20 -code=sm_20,sm_30
- Will support JIT linking in future release

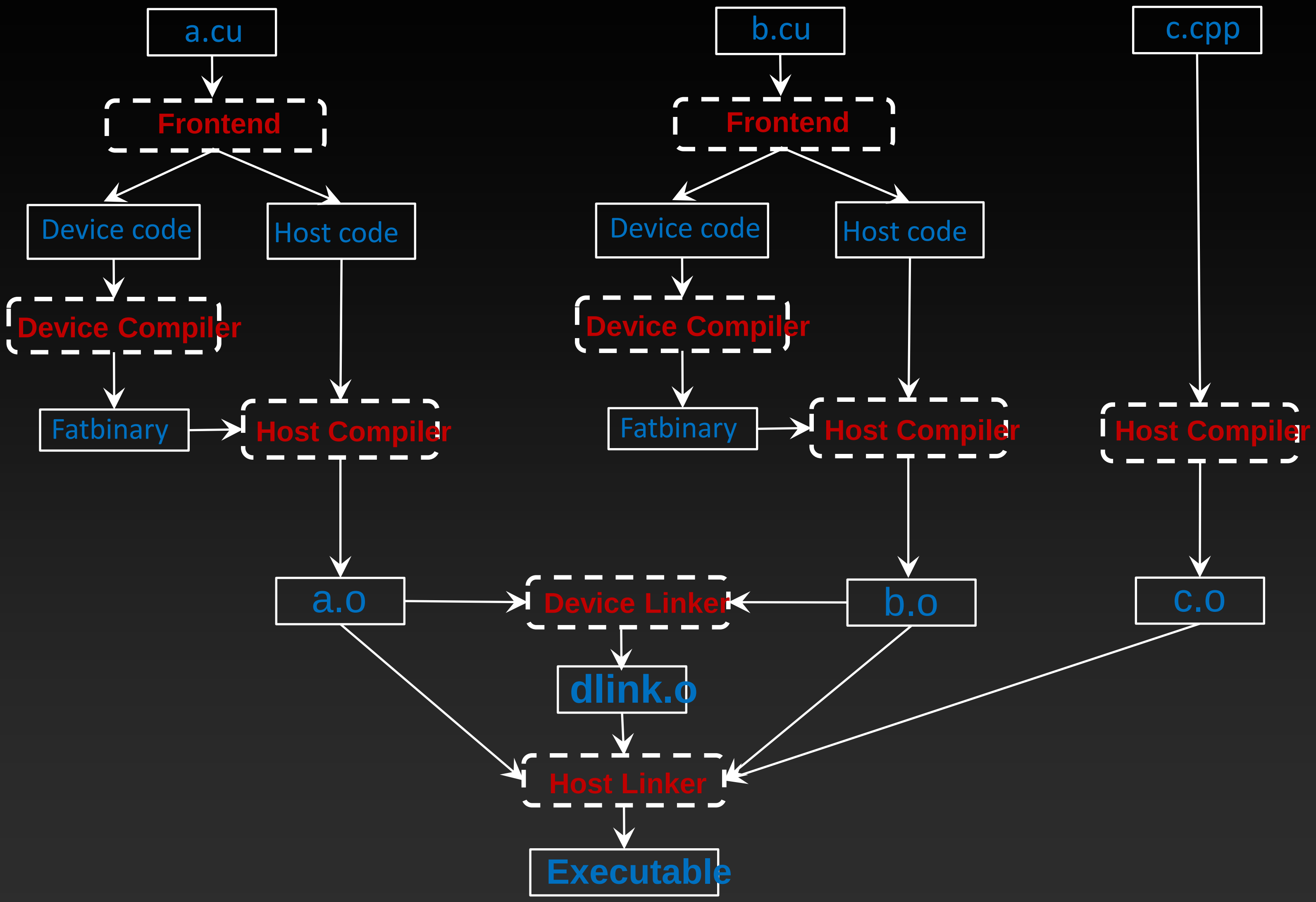
Summary

- Separate Compilation of device code is supported in CUDA 5.0
- Eases porting
- Incremental Recompilation
- Library Support
- For more info, see “Using Separate Compilation in CUDA” section at end of NVCC document.

nvcc compile



nvcc separate compilation and link


a.cu
b.cu
c.cpp
Frontend
Frontend
Device code
Host code
Device code
Host code
Device Compiler
Device Compiler
Fatbinary
Host Compiler
Fatbinary
Host Compiler
Host Compiler
a.o
Device Linker
b.o
c.o
dlink.o
Host Linker
Executable