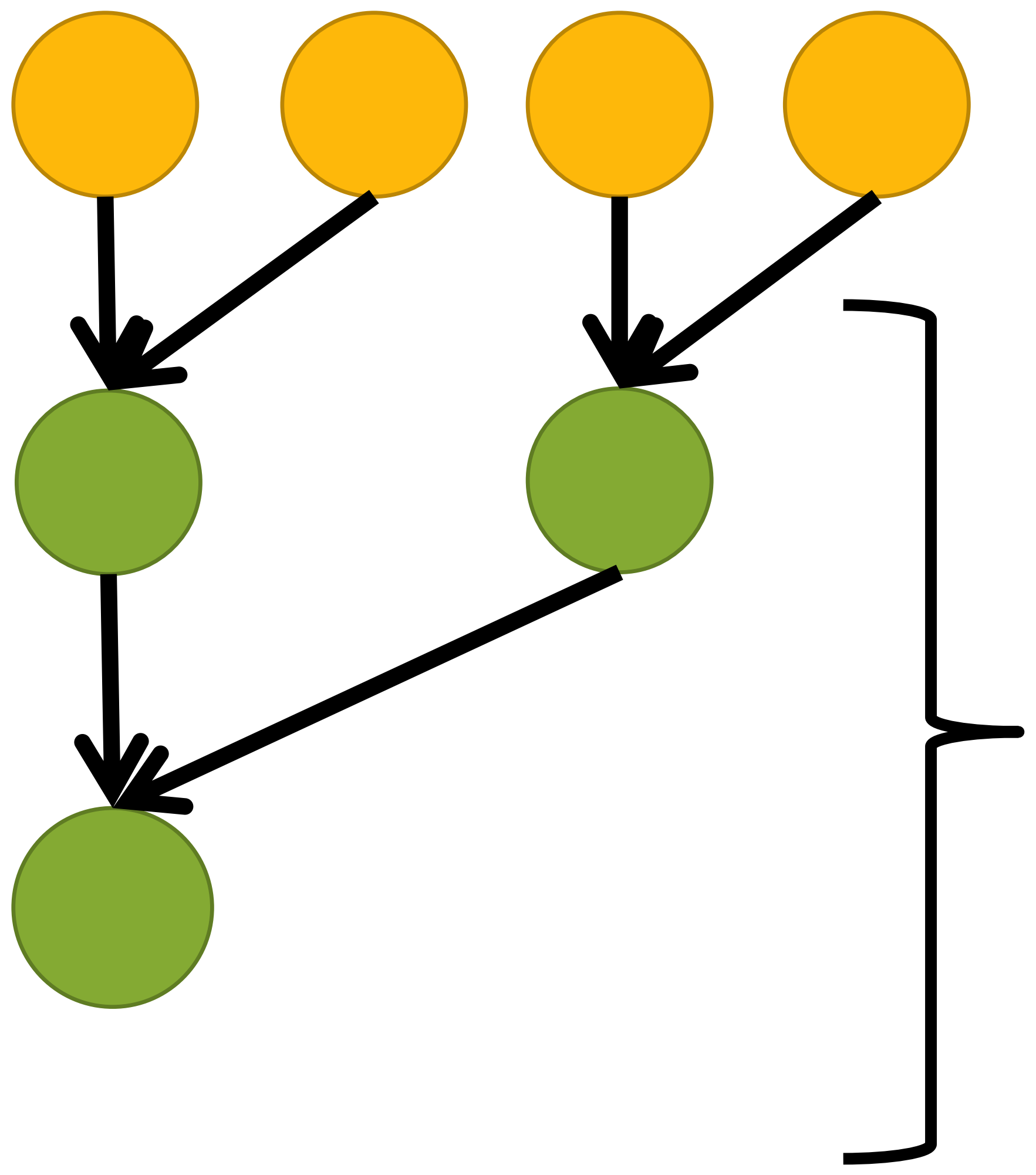
local_malloc: malloc() for OpenCL `__local` memory

John Kloosterman, Joel Adams (advisor), Calvin College, Grand Rapids, MI

Abstract

One of the complexities of writing kernels in OpenCL is managing the scarce per-workgroup `__local` memory on a device. For instance, temporary blocks of `__local` memory are necessary to implement algorithms like non-destructive parallel reduction. However, all `__local` memory must be allocated at the beginning of a kernel, and programmers are responsible for tracking which buffers can be reused in a kernel. We propose and implement an extension to OpenCL C that provides a `malloc()`-like interface for allocating workgroup memory. This extension was implemented by using an extension to the Clang compiler to perform a source-to-source transformation on OpenCL C programs.

Motivation: parallel reduction



Must be stored in temporary per-workgroup buffer in `__local` memory space.

Without `local_malloc()`, this buffer must be allocated at the beginning of the kernel, instead of by the implementation of the algorithm.

Benefits: module reusability, reuse of __local memory by other code, memory allocation near use

API

- ○ `local_malloc({bytes})` allocates `__local` memory
- ○ `local_free({bytes})` frees the previously allocated `{bytes}` memory.
- ○ Host C++ library performs code transformations.

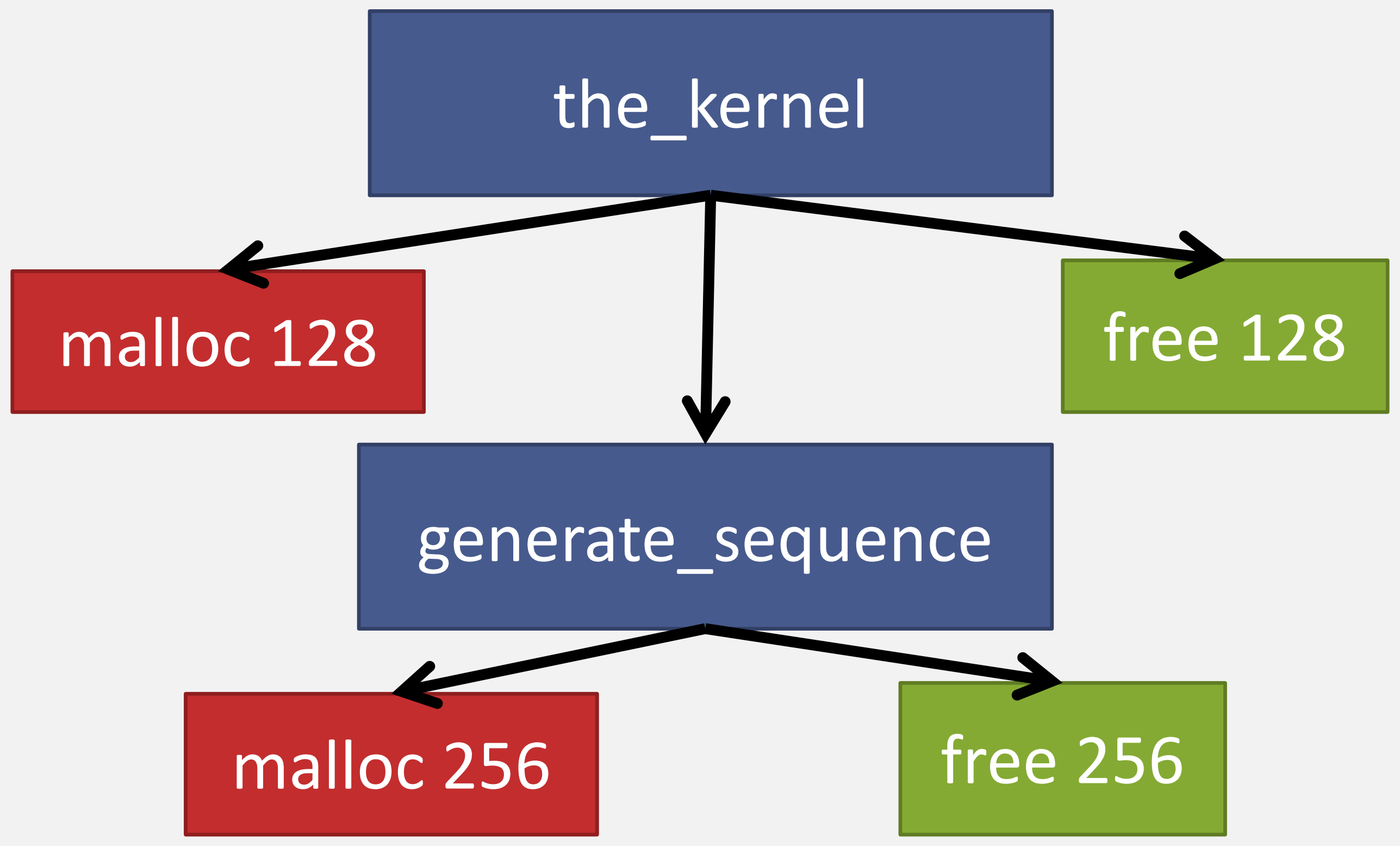
0. Source code before processing

```
void generate_sequence(void) {
   __local uchar *buf = local_malloc(256);
   buf[get_local_id(0)] = get_local_id(0);
   local_free(256);
}

__kernel void the_kernel(void)  {
    __local uchar *buf = local_malloc(128);
    generate_sequence();
    local_free(128);
}
```

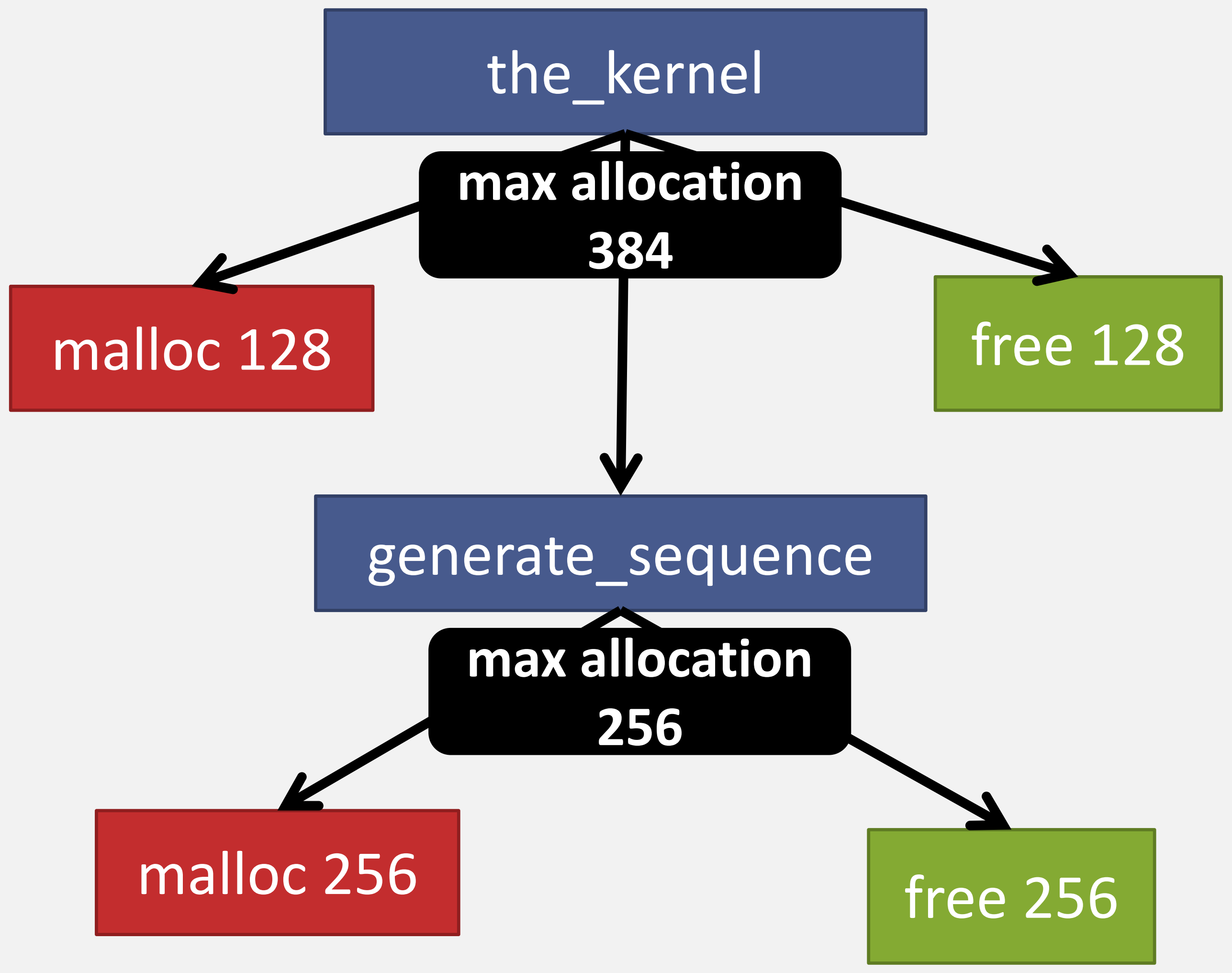
1. Parse code and construct call tree

- ○ No recursion in OpenCL means call graph is a call tree
- ○ Record the sequence of `local_malloc()`s, `local_free()`s, and function calls
- ○ Ignore calls to OpenCL built-ins



2. Compute per-function maximum allocation

- ○ Maximum allocation = the maximum amount of memory a function and its children in the call tree have allocated at any given time
- ○ Amount of memory to reserve for the kernel is the maximum allocation of the root node



3. Rewrite source code

The compiler modifies the kernel by inserting:

1. ① Definitions of `local_malloc()` and `local_free()` at beginning of source
2. ② New parameter in each function def & call to pass buffer and offset of next allocation
3. ③ Code to allocate buffer when kernel starts

```
/* implementations of local_malloc(), local_free() */                          ①

void generate_sequence(LocalMallocState *__local_malloc_state) {               ②
   __local uchar *buf = local_malloc(256, __local_malloc_state);
   buf[get_local_id(0)] = get_local_id(0);
   local_free(256, __local_malloc_state);
}

__kernel void a_kernel(void) {
    __local char __local_malloc_buffer[384];
    LocalMallocState __local_malloc_state_backing;                             ③
    LocalMallocState *__local_malloc_state =
           &__local_malloc_state_backing;
    local_malloc_init(
         __local_malloc_buffer, 384,
         __local_malloc_state);

    __local uchar *buf = local_malloc(128, __local_malloc_state);
    generate_sequence(__local_malloc_state);
    local_free(128, __local_malloc_state);                                     ②
}
```

This source code can then be compiled by vendors' OpenCL implementations at runtime .

References

[1] clang: a C language family frontend for LLVM. http://clang.llvm.org.

This work supported in part by NSF DUE-1225739