



# *OpenMP 4 Offloading Features implementation in GCC*

**Ilya Verbin** <[ilya.verbin@intel.com](mailto:ilya.verbin@intel.com)>

**Kirill Yukhin** <[kirill.yukhin@intel.com](mailto:kirill.yukhin@intel.com)>

# Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2015, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

## Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

# *OpenMP 4.0 Device Constructs*

# OpenMP 4.0

## *#pragma omp target*

```
void vec_mult ()
{
    double p[N], v1[N], v2[N];
    init (v1, v2);
    #pragma omp target
    {
        #pragma omp parallel for
        for (int i = 0; i < N; i++)
            p[i] = v1[i] * v2[i];
    }
    output (p);
}
```

# OpenMP 4.0

## *#pragma omp target*

```
void vec_mult ()
{
    double p[N], v1[N], v2[N];
    init (v1, v2);

    #pragma omp target
    {
        #pragma omp parallel for
        for (int i = 0; i < N; i++)
            p[i] = v1[i] * v2[i];
    }

    output (p);
}
```

- Initialize target device
  - Offload target image
  - Allocate memory for p[N], v1[N], v2[N]
  - Copy p[N], v1[N], v2[N] from host
- 
- Copy p[N], v1[N], v2[N] to host
  - Free memory on target



Execution on host



Execution on target



Communication between host and target

# OpenMP 4.0

## *#pragma omp target*

```
void vec_mult ()
{
    double p[N], v1[N], v2[N];
    init (v1, v2);
    #pragma omp target if(1) device(0) map(to: v1[0:N], v2[0:N]) map(from:p[0:N])
    {
        #pragma omp parallel for
        for (int i = 0; i < N; i++)
            p[i] = v1[i] * v2[i];
    }
    output (p);
}
```

# *Implementation in GCC. Compilation*

# Implementation in GCC

## Building host and target compilers

Intel MIC target (accel/offload/offloading) compiler:

```
$ configure --prefix=/install/prefix/ --target=x86_64-intelmic-linux-gnu \
  --enable-as-accelerator-for=x86_64-unknown-linux-gnu
$ make && make install
```

Host compiler:

```
$ configure --prefix=/install/prefix/ --target=x86_64-unknown-linux-gnu \
  --enable-offload-targets=x86_64-intelmic-linux-gnu
$ make && make install
```

More info: <https://gcc.gnu.org/wiki/Offloading>

# Implementation in GCC

## test1.c

```
int foo ()
{
    int myvar = 1;
    int myarray[1000];

    #pragma omp target device(0) map(tofrom: myvar)
        { myvar *= 10; }

    #pragma omp target device(0) map(to: myvar) map(from: myarray)
        { myarray[42] = myvar; }

    return myarray[42];
}

int main ()
{
    return foo ();
}
```

```
$ gcc -fopenmp -c test1.c -o test1.o
```

# Implementation in GCC

## test1.c.004t.gimple

```
int foo ()
{
  int myvar = 1;
  int myarray[1000];

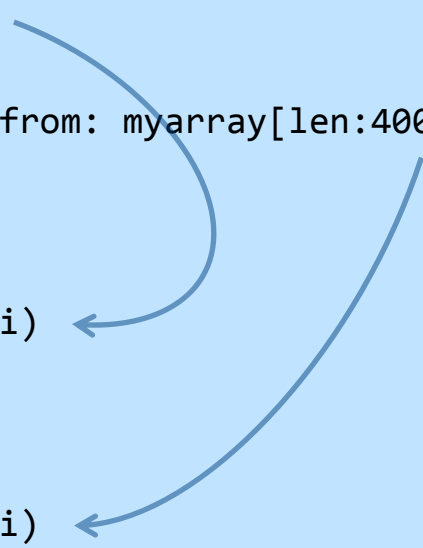
  #pragma omp target device(0) map(tofrom: myvar[len:4])
    { myvar *= 10; }

  #pragma omp target device(0) map(to: myvar[len:4]) map(from: myarray[len:4000])
    { myarray[42] = myvar; }

  return myarray[42];
}

foo._omp_fn.0 (struct .omp_data_t.3 & restrict .omp_data_i)
{
  int *D.1873 = .omp_data_i->myvar;
  *D.1873 = *D.1873 * 10;
}

foo._omp_fn.1 (struct .omp_data_t.4 & restrict .omp_data_i)
{
  int *D.1868 = .omp_data_i->myvar;
  int *D.1870 = .omp_data_i->myarray;
  *D.1870[42] = *D.1868;
}
```



# Implementation in GCC

## test1.c.012t.ompexp

```
int foo ()
{
    .omp_data_arr.myvar = &myvar;
    .omp_data_sizes[1] = { 4 };
    .omp_data_kinds[1] = { GOMP_MAP_TOFROM };

    __builtin_GOMP_target (0, foo._omp_fn.0, 1, &.omp_data_arr, &.omp_data_sizes,
                          &.omp_data_kinds);

    .omp_data_arr.myvar = &myvar;
    .omp_data_arr.myarray = &myarray;
    .omp_data_sizes[2] = { 4, 4000 };
    .omp_data_kinds[2] = { GOMP_MAP_TO, GOMP_MAP_FROM };

    __builtin_GOMP_target (0, foo._omp_fn.1, 2, &.omp_data_arr, &.omp_data_sizes,
                          &.omp_data_kinds);
}

foo._omp_fn.0 (struct .omp_data_t.3 & restrict .omp_data_i)
{
    int *D.1873 = .omp_data_i->myvar;
    *D.1873 = *D.1873 * 10;
}

foo._omp_fn.1 (struct .omp_data_t.4 & restrict .omp_data_i)
{
    int *D.1868 = .omp_data_i->myvar;
    int *D.1870 = .omp_data_i->myarray;
    *D.1870[42] = *D.1868;
}
```

# Implementation in GCC

## test1.o

```
$ gcc -fopenmp -c test1.c -o test1.o
```

```
.text
```

```
{ .data, .bss, etc. }
```

```
.gnu.offload_vars
```

```
.gnu.offload_funcs
```

```
.gnu.offload_lto_foo._omp_fn.0
```

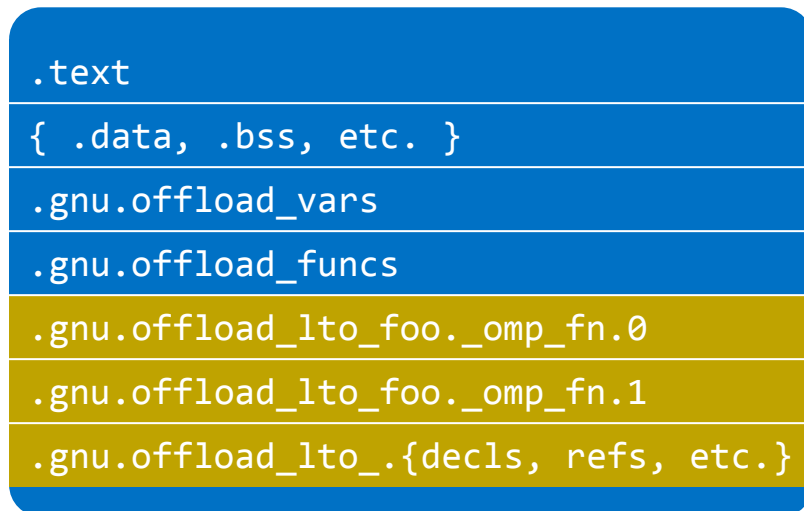
```
.gnu.offload_lto_foo._omp_fn.1
```

```
.gnu.offload_lto_.{decls, refs, etc.}
```

Fat object, similar to general LTO (<https://gcc.gnu.org/onlinedocs/gccint/LTO-object-file-layout.html>)

# Implementation in GCC test1.o

```
$ gcc -fopenmp -c test1.c -o test1.o
```



```
<main>:
  55 48 89 e5 b8 00 00 00
  ...
<foo>:
  55 48 89 e5 48 81 ec d0
  ...
<foo._omp_fn.0>:
  55 48 89 e5 48 89 7d f8
  ...
<foo._omp_fn.1>:
  55 48 89 e5 48 89 7d f8
  ...
```

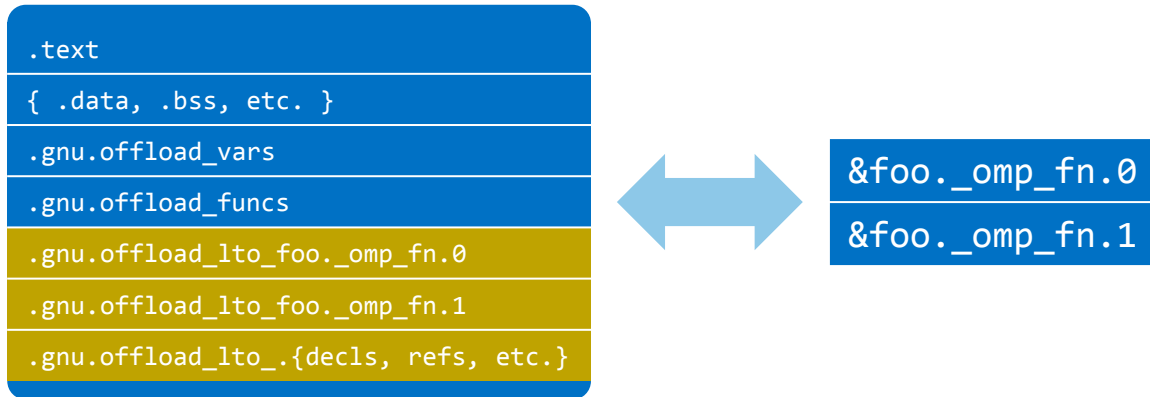
```
&foo._omp_fn.0
&foo._omp_fn.1
```



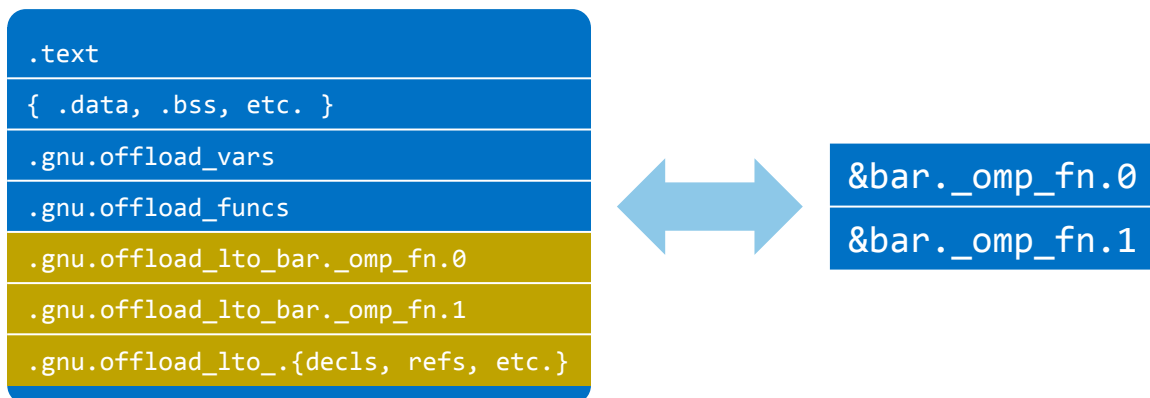
# Implementation in GCC

## test1.o and test2.o

```
$ gcc -fopenmp -c test1.c -o test1.o
```



```
$ gcc -fopenmp -c test2.c -o test2.o
```



# Implementation in GCC

## Link-time compilation

```
$ gcc -fopenmp test1.o test2.o -o a.out
```

```
    |
collect2
    |
collect-ld
    |
ld
    |
liblto_plugin.so
    |
lto-wrapper
    |
intelmic-mkoffload
    |
intelmic-gcc
```

.text
{ .data, .bss, etc. }
.gnu.offload_vars
.gnu.offload_funcs
.gnu.offload_lto_foo._omp_fn.0
.gnu.offload_lto_foo._omp_fn.1
.gnu.offload_lto_{decls, refs, etc.}

.text
{ .data, .bss, etc. }
.gnu.offload_vars
.gnu.offload_funcs
.gnu.offload_lto_bar._omp_fn.0
.gnu.offload_lto_bar._omp_fn.1
.gnu.offload_lto_{decls, refs, etc.}

# Implementation in GCC

## Link-time compilation

```
$ gcc -fopenmp test1.o test2.o -o a.out
```

collect2  
collect-ld  
ld  
liblto\_plugin.so  
lto-wrapper  
intelmic-mkoffload  
intelmic-gcc

.gnu.offload_lto_foo_omp_fn.0	.gnu.offload_lto_bar_omp_fn.0
.gnu.offload_lto_foo_omp_fn.1	.gnu.offload_lto_bar_omp_fn.1
.gnu.offload_lto_{decls, refs, etc.}	.gnu.offload_lto_{decls, refs, etc.}



Target image (DSO)

.text
{ .data, .bss, etc. }
.gnu.offload_vars
.gnu.offload_funcs

.text
{ .data, .bss, etc. }
.gnu.offload_vars
.gnu.offload_funcs

.text
{ .data, .bss, etc. }
.gnu.offload_vars
.gnu.offload_funcs

# Implementation in GCC

## Link-time compilation

```
$ gcc -fopenmp test1.o test2.o -o a.out
```

collect2

collect-ld

ld

liblto\_plugin.so

lto-wrapper

intelmic-mkoffload

intelmic-gcc

.gnu.offload_lto_foo_omp_fn.0	.gnu.offload_lto_bar_omp_fn.0
.gnu.offload_lto_foo_omp_fn.1	.gnu.offload_lto_bar_omp_fn.1
.gnu.offload_lto_{decls, refs, etc.}	.gnu.offload_lto_{decls, refs, etc.}

.text
{ .data, .bss, etc. }
.gnu.offload_vars
.gnu.offload_funcs

.text
{ .data, .bss, etc. }
.gnu.offload_vars
.gnu.offload_funcs

```
<foo._omp_fn.0>:
 55 48 89 e5 48 89 7d f8
...
<foo._omp_fn.1>:
 55 48 89 e5 48 89 7d f8
...
<bar._omp_fn.0>:
 55 48 89 e5 48 89 7d f8
...
<bar._omp_fn.1>:
 55 48 89 e5 48 89 7d f8
...
```

```
&foo._omp_fn.0
&foo._omp_fn.1
&bar._omp_fn.0
&bar._omp_fn.1
```

(R\_X86\_64\_RELATIVE)

Target image (DSO)

.text
{ .data, .bss, etc. }
.gnu.offload_vars
.gnu.offload_funcs

# Implementation in GCC

## Link-time compilation

```
$ gcc -fopenmp test1.o test2.o -o a.out
```

```
|  
collect2  
|  
collect-ld  
|  
ld  
|  
liblto_plugin.so  
|  
lto-wrapper  
|  
intelmic-mkoffload  
|  
intelmic-gcc
```

.text
{ .data, .bss, etc. }
.gnu.offload_vars
.gnu.offload_funcs

.text
{ .data, .bss, etc. }
.gnu.offload_vars
.gnu.offload_funcs

.gnu.offload\_images

.text

{ .data, .bss, etc. }

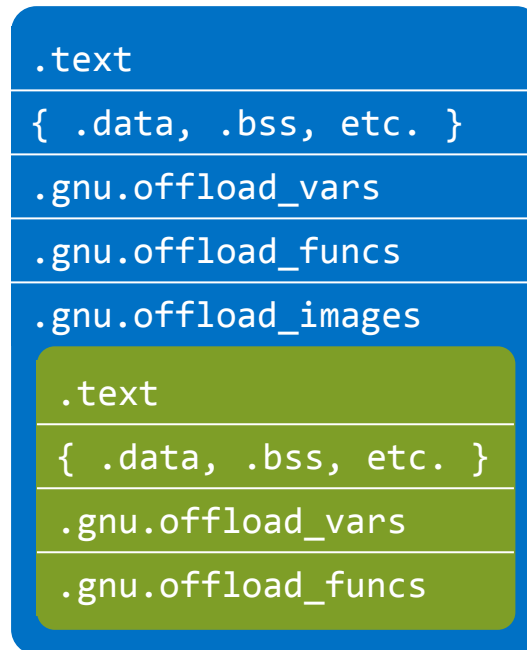
.gnu.offload\_vars

.gnu.offload\_funcs

# Implementation in GCC Linking

```
$ gcc -fopenmp test1.o test2.o -o a.out
```

```
|  
collect2  
|  
collect-ld  
|  
ld  
|  
liblto_plugin.so  
|  
lto-wrapper  
|  
intelmic-mkoffload  
|  
intelmic-gcc
```



Final a.out (fat binary)



```
<init>  
<fini>  
<main>  
<foo>  
<foo._omp_fn.0>  
<foo._omp_fn.1>  
<bar>  
<bar._omp_fn.0>  
<bar._omp_fn.1>
```



```
<init>  
<foo._omp_fn.0>  
<foo._omp_fn.1>  
<bar._omp_fn.0>  
<bar._omp_fn.1>
```

# *Implementation in GCC. Execution*

# Implementation in GCC Startup

\$ ./a.out

init

foo

foo.\_omp\_fn.0

...

&foo.\_omp\_fn.0

&foo.\_omp\_fn.1

&bar.\_omp\_fn.0

&bar.\_omp\_fn.1

foo.\_omp\_fn.0

...

&foo.\_omp\_fn.0

&foo.\_omp\_fn.1

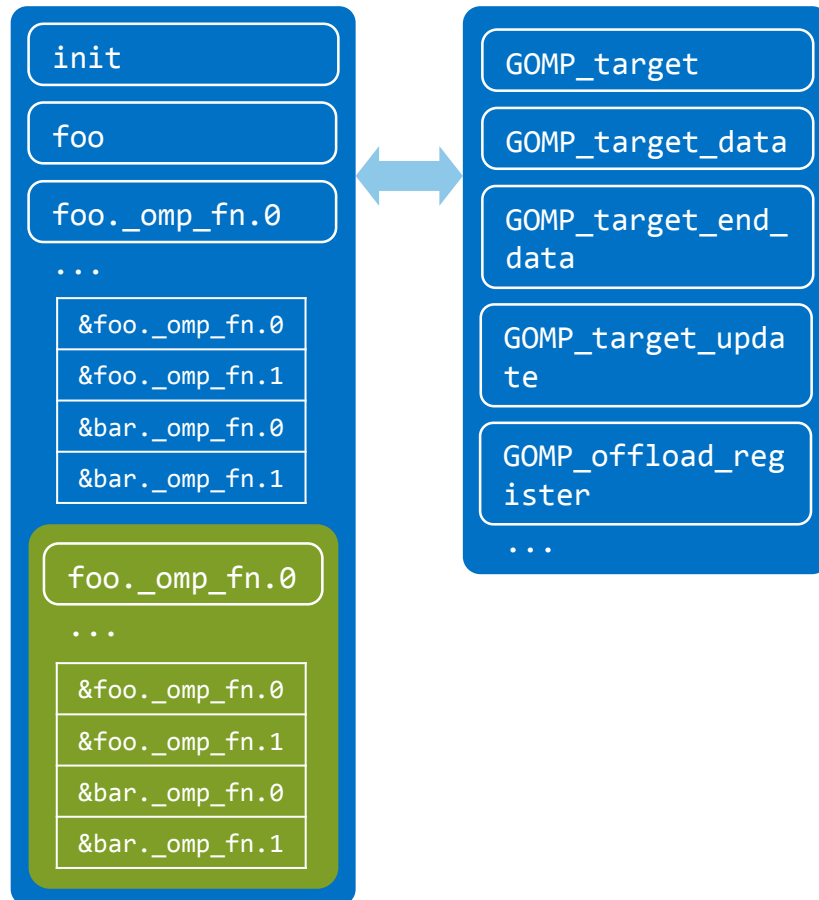
&bar.\_omp\_fn.0

&bar.\_omp\_fn.1

# Implementation in GCC Startup

\$ ./a.out

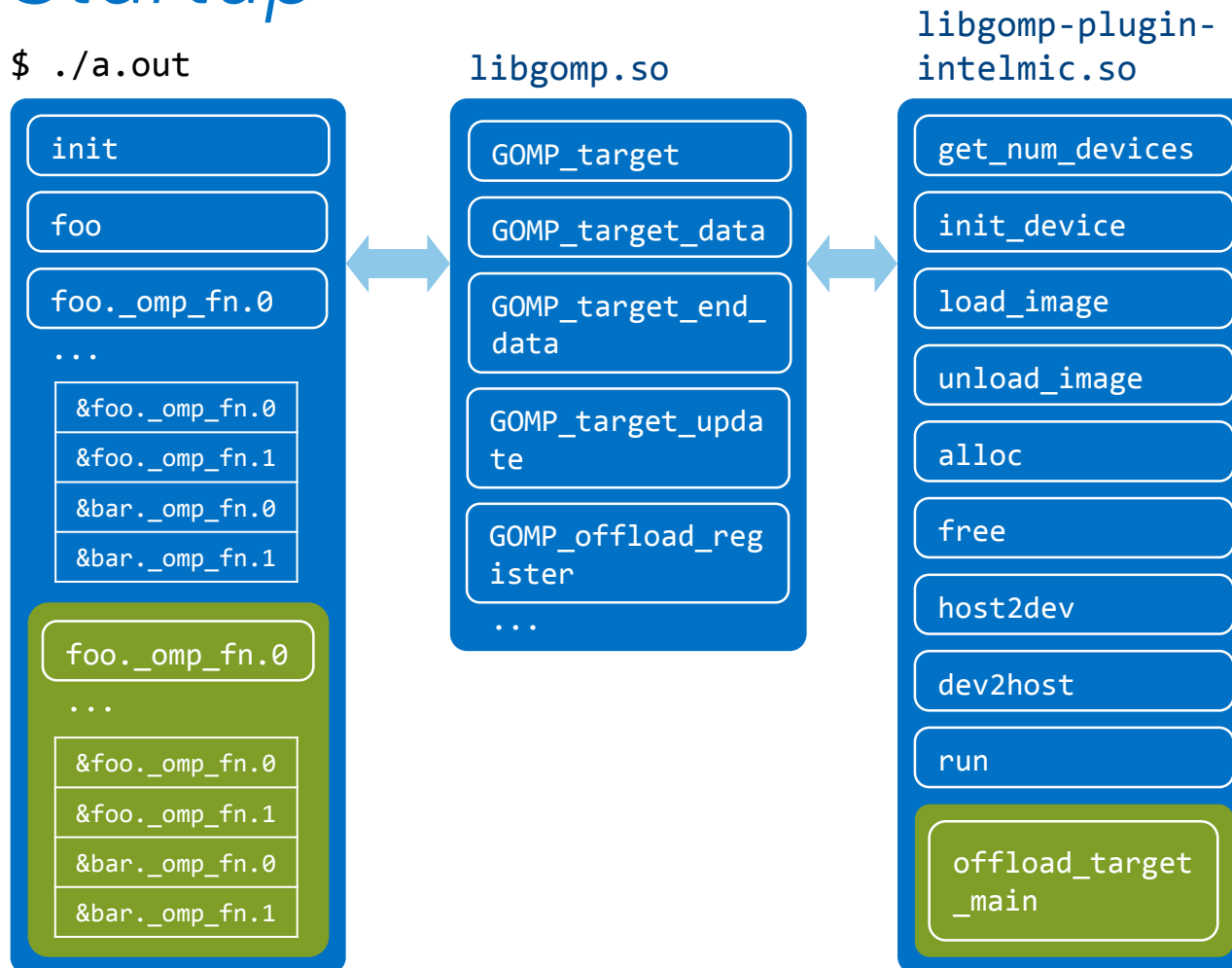
libgomp.so



# Implementation in GCC

## Startup

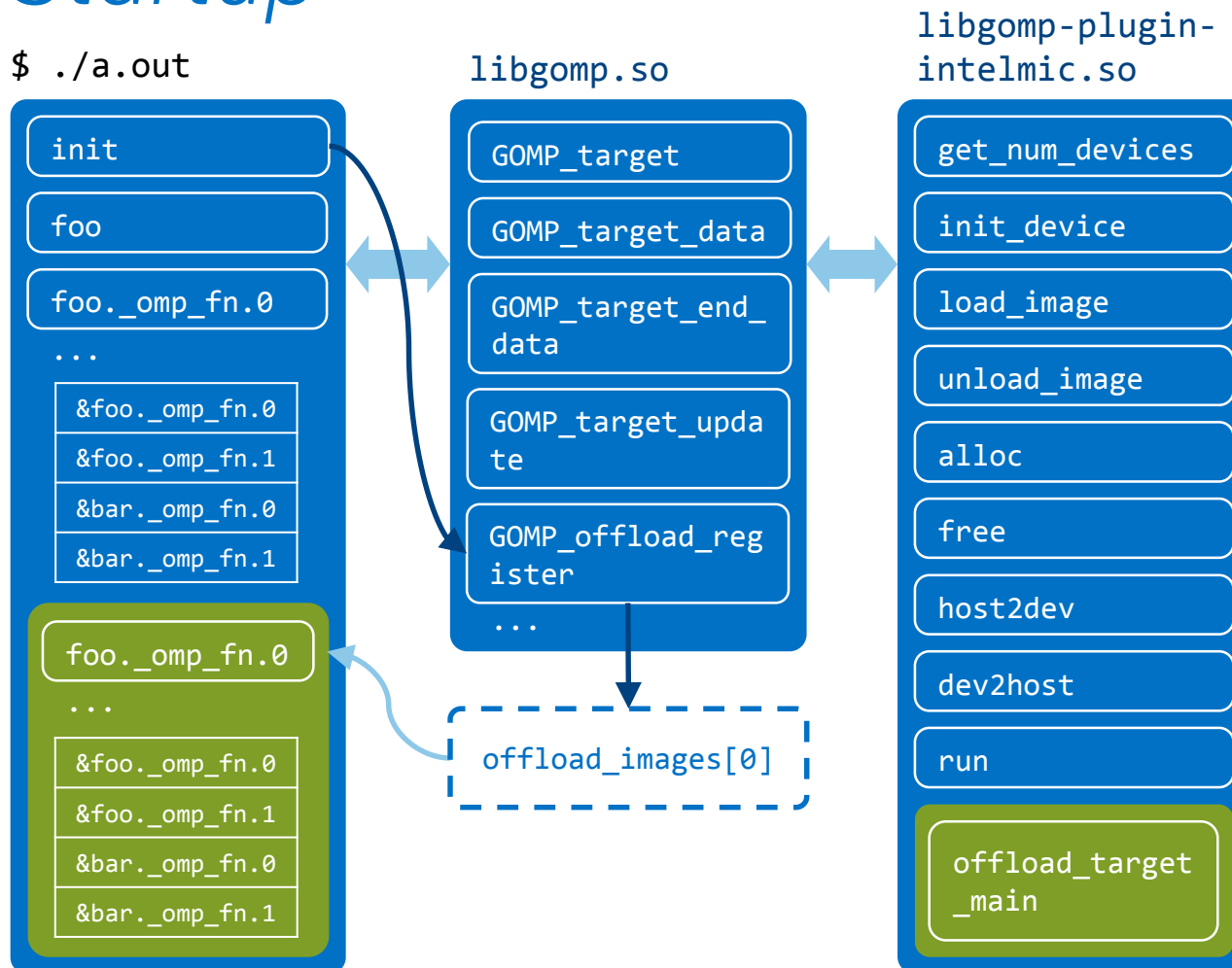
\$ ./a.out



# Implementation in GCC

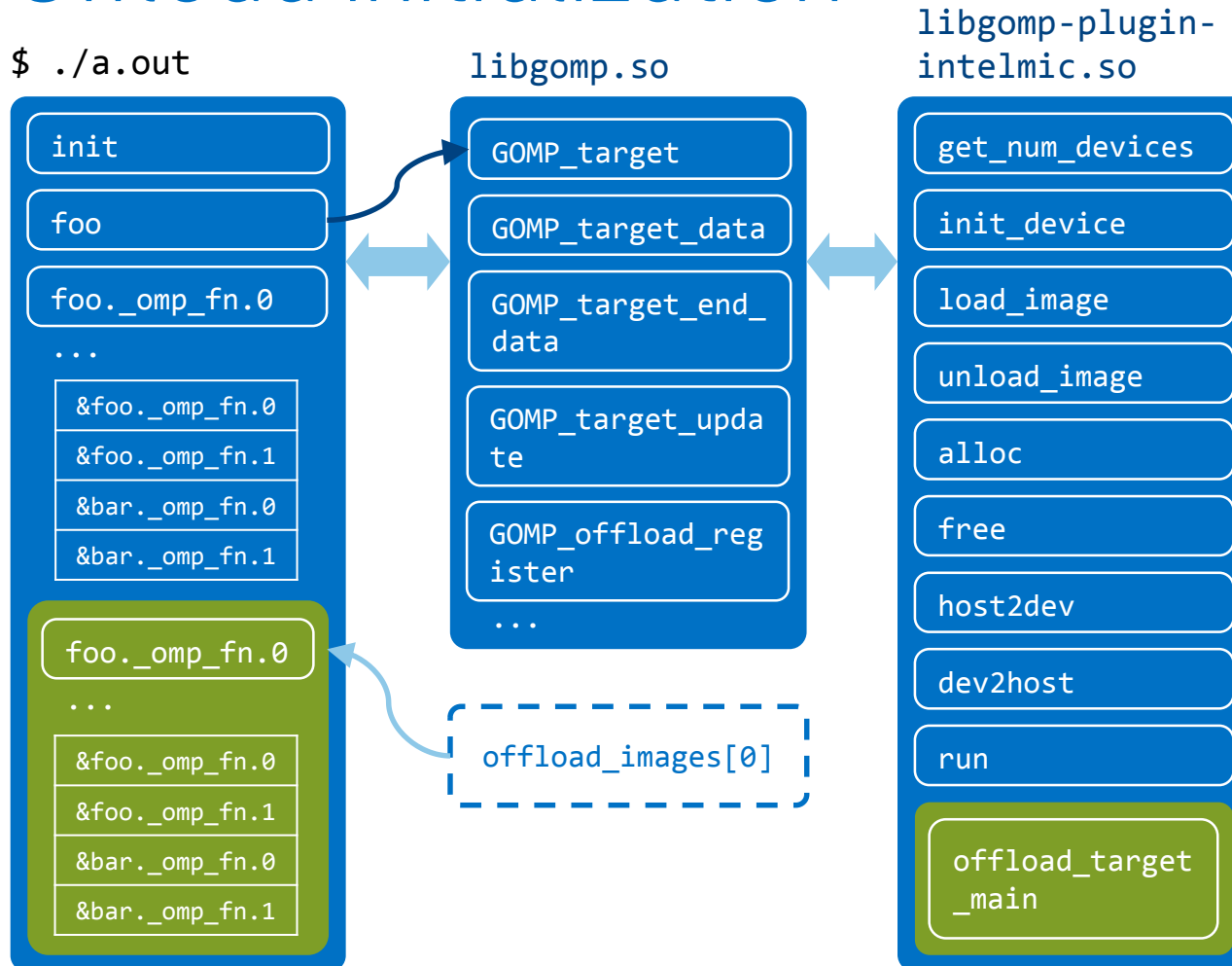
## Startup

\$ ./a.out



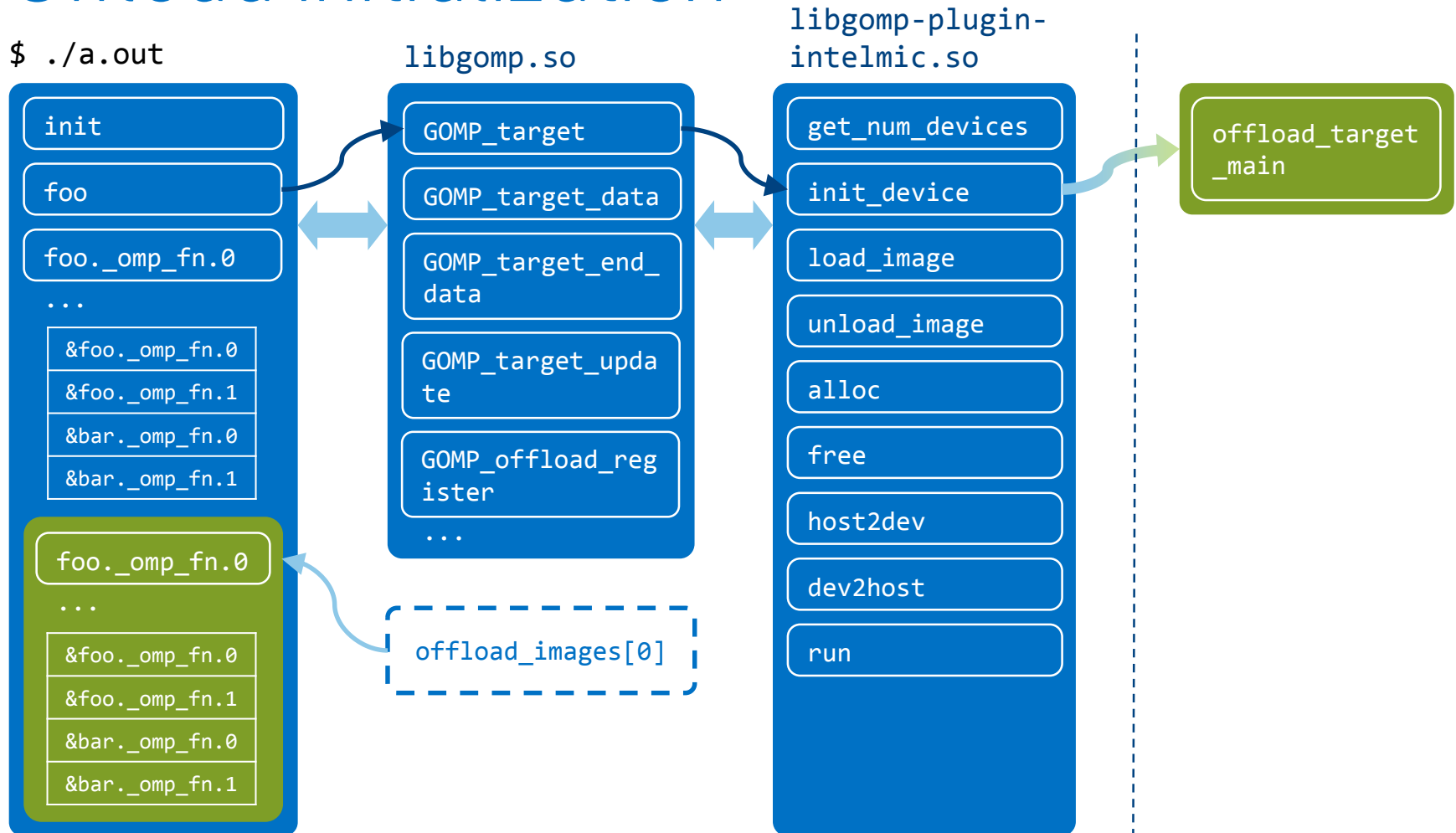
# Implementation in GCC

## Offload initialization



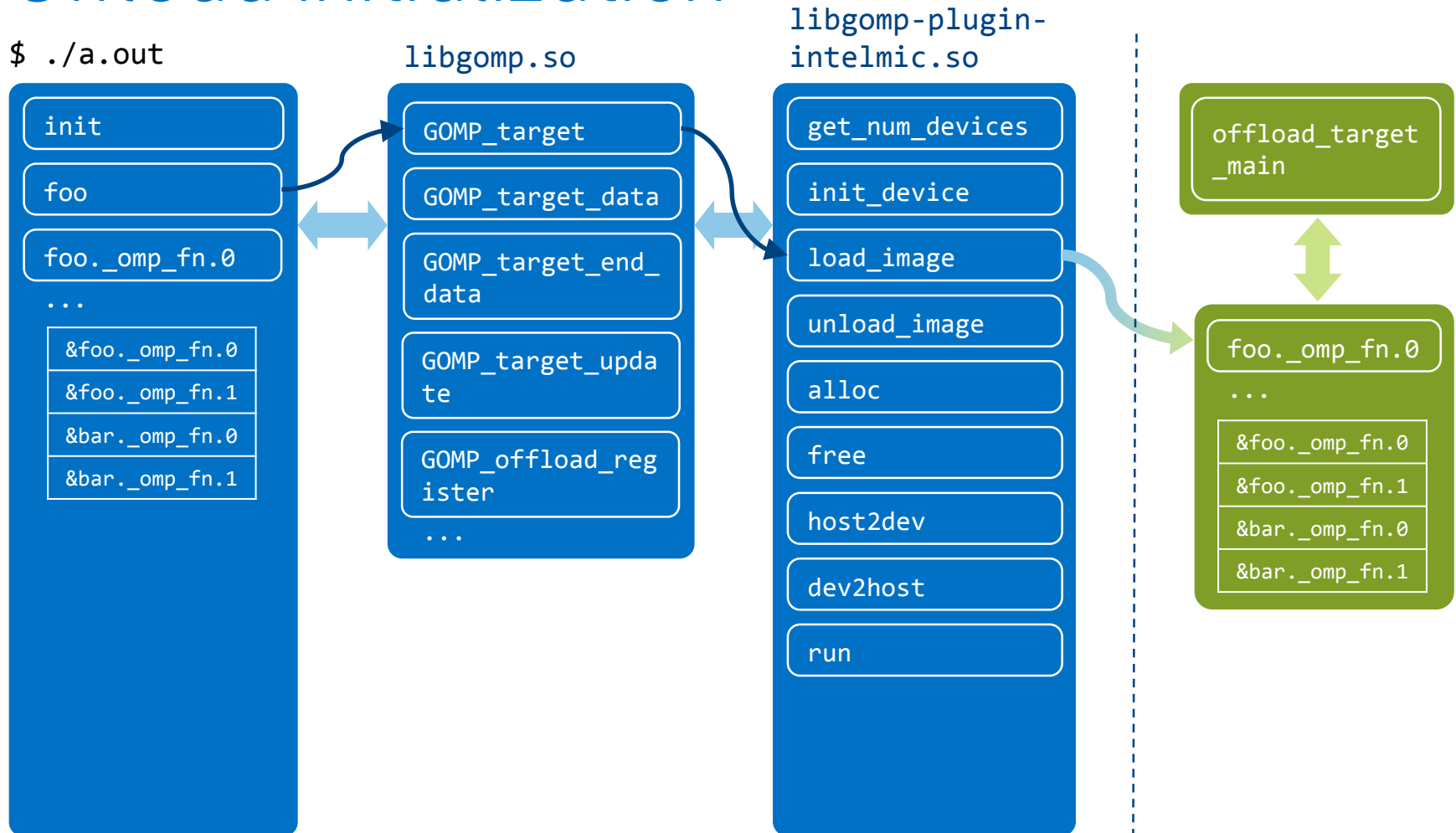
# Implementation in GCC

## Offload initialization



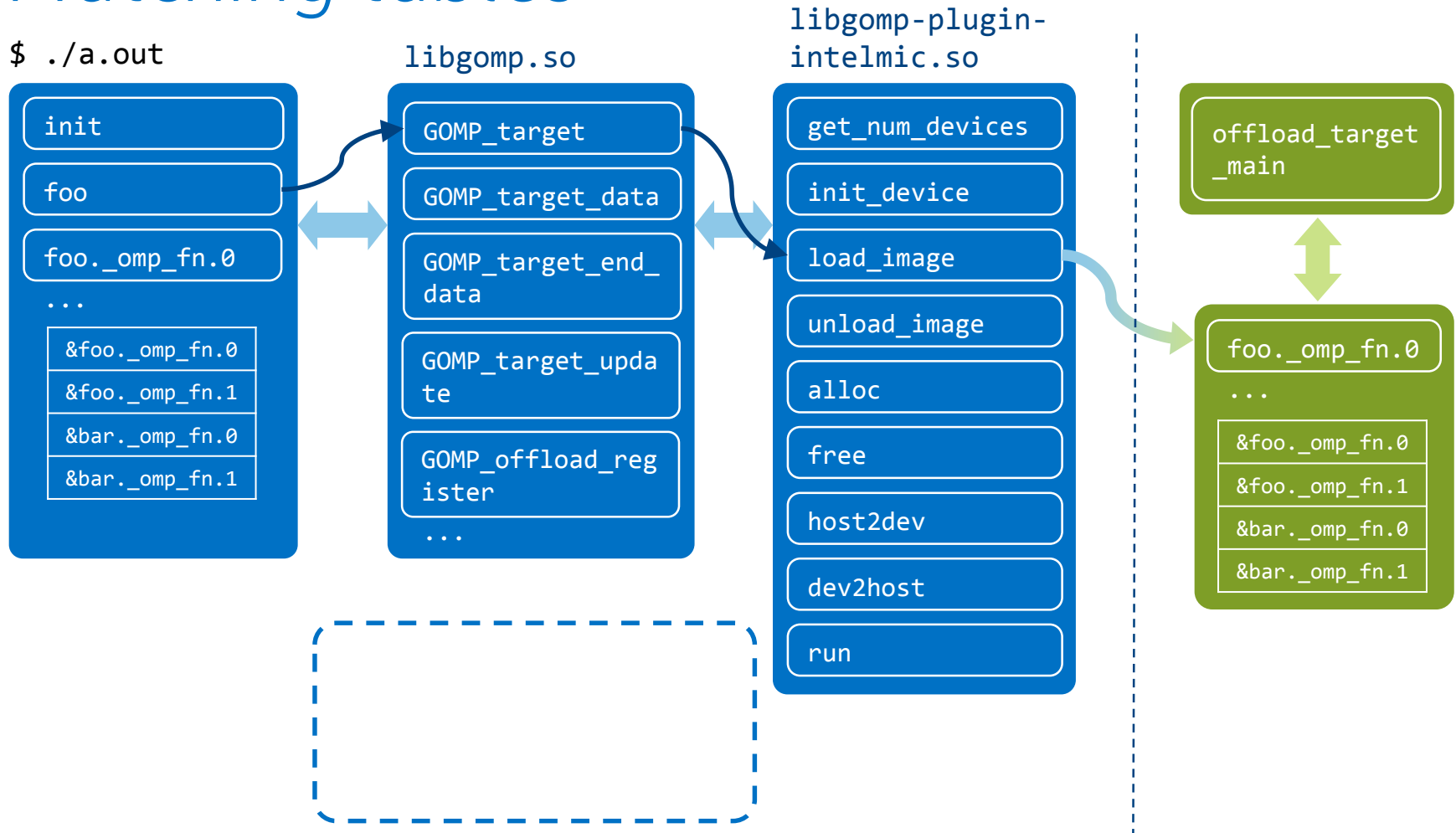
# Implementation in GCC

## Offload initialization



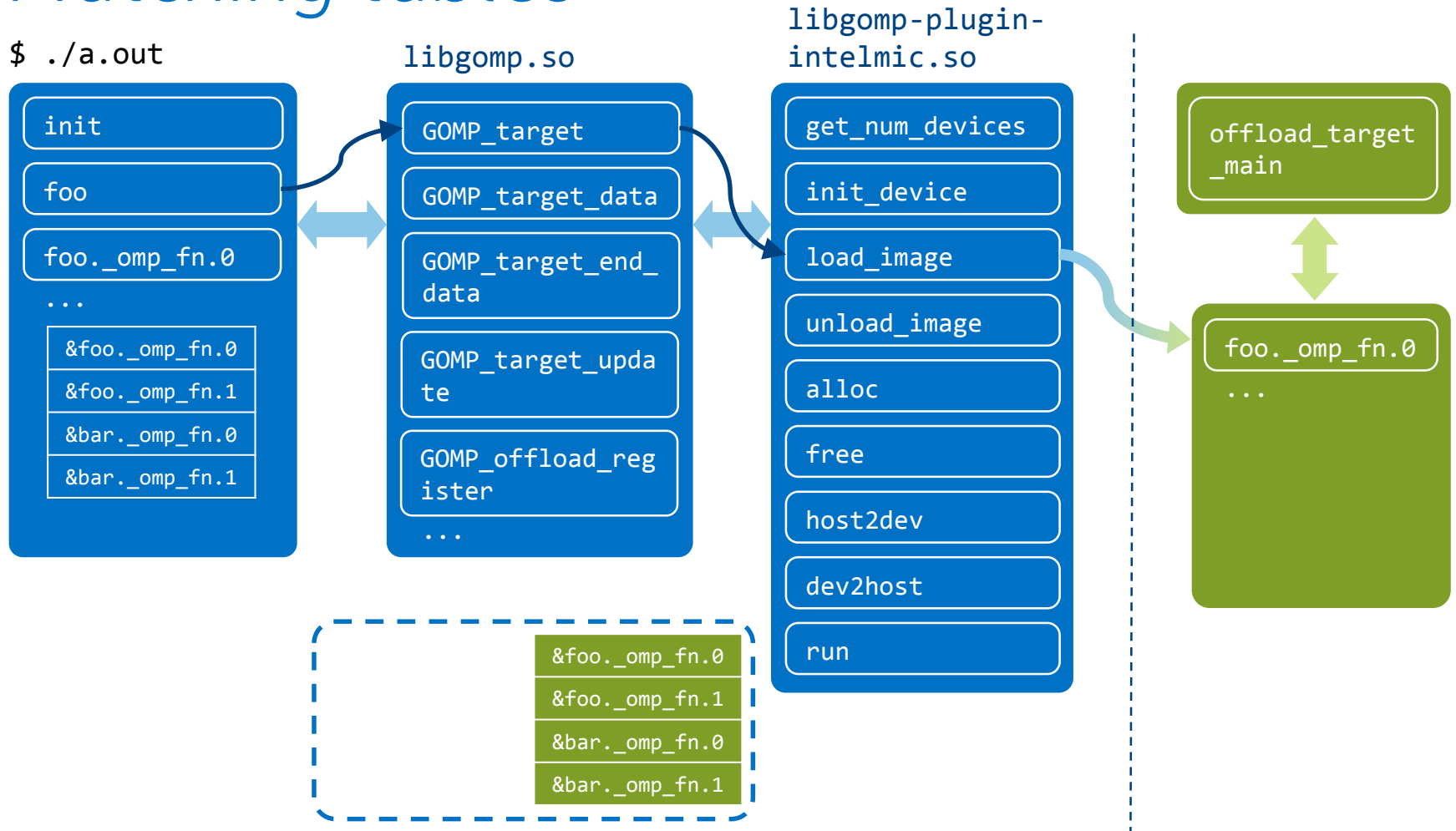
# Implementation in GCC

## Matching tables



# Implementation in GCC

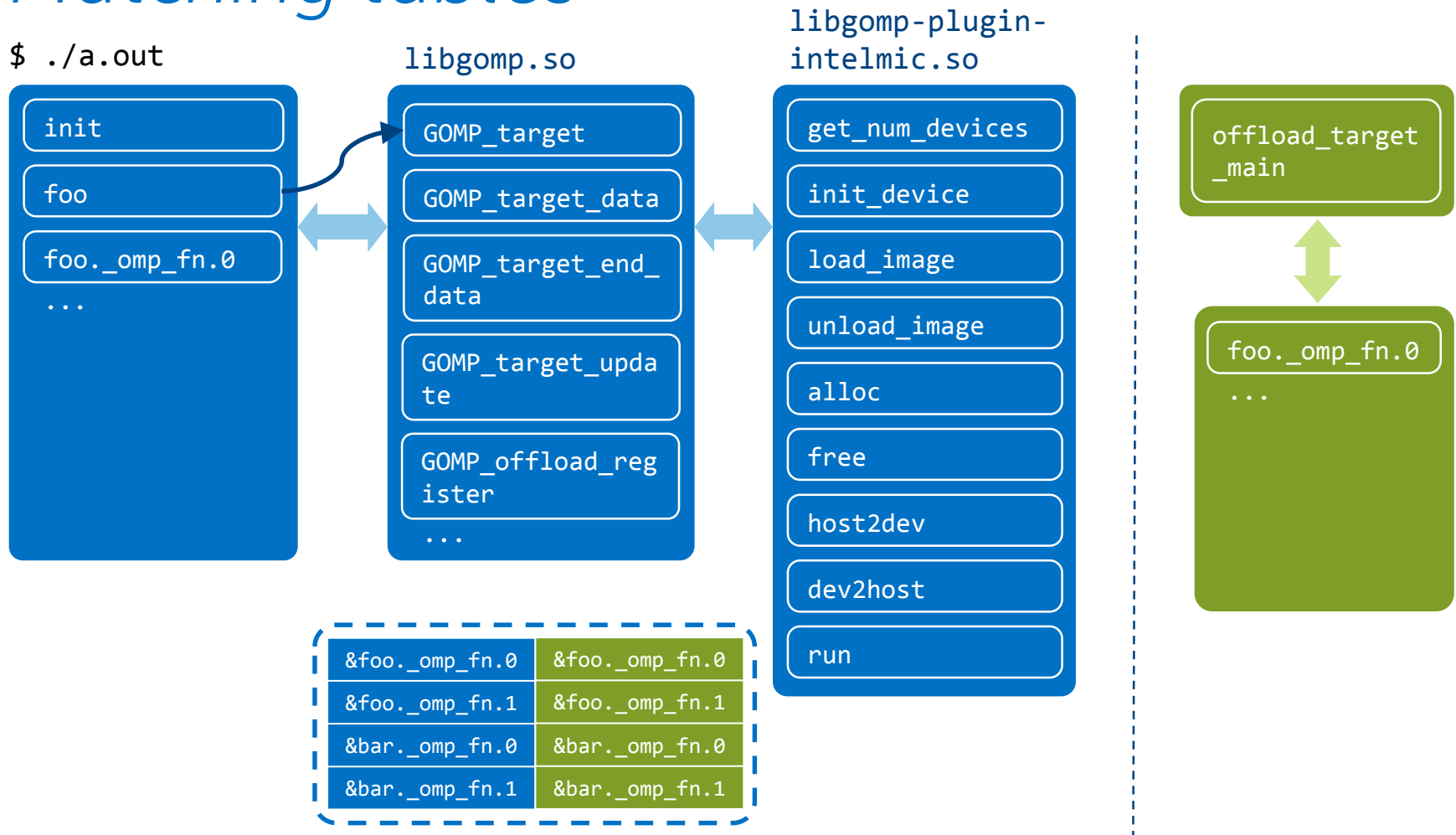
## Matching tables



# Implementation in GCC

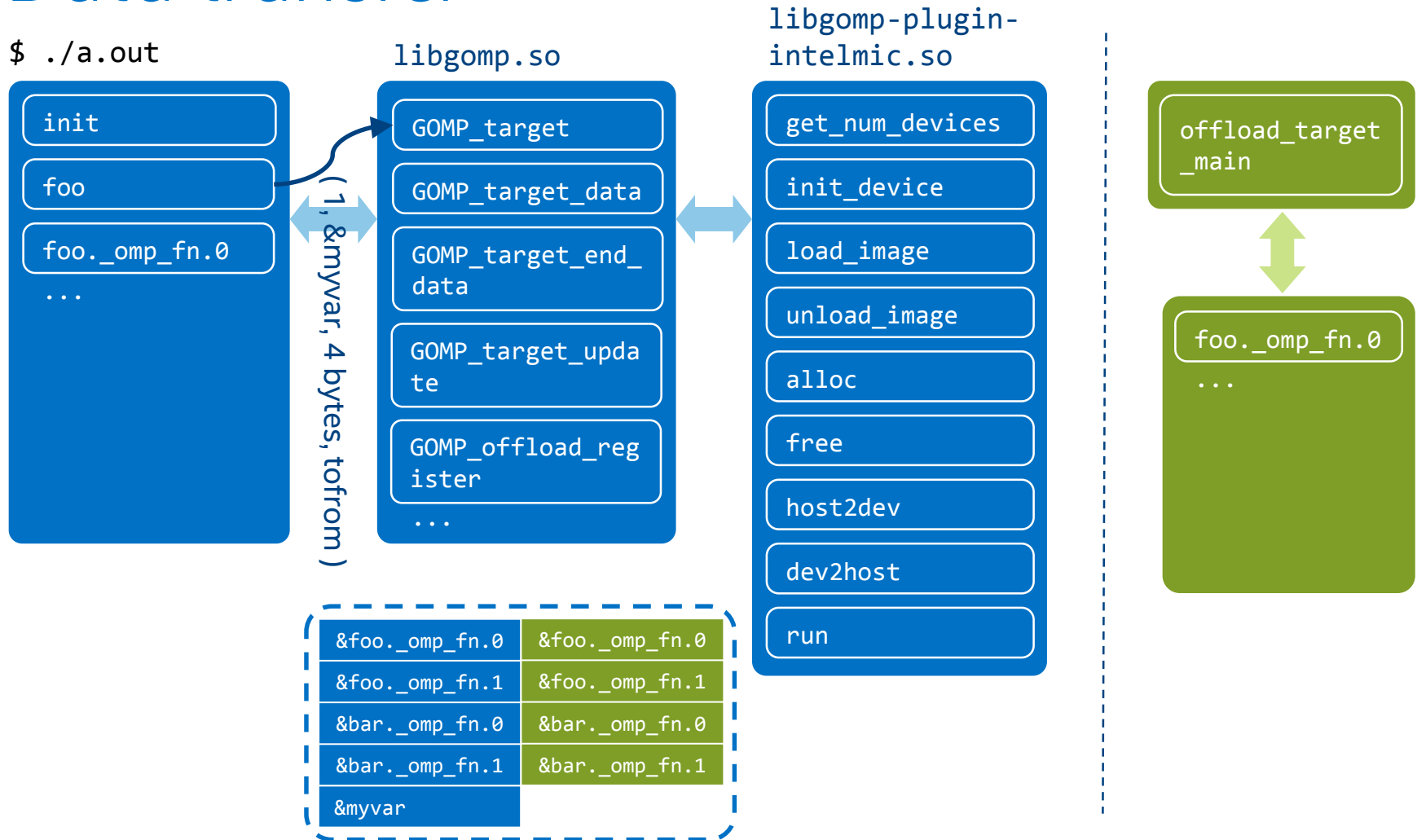
## Matching tables

\$ ./a.out



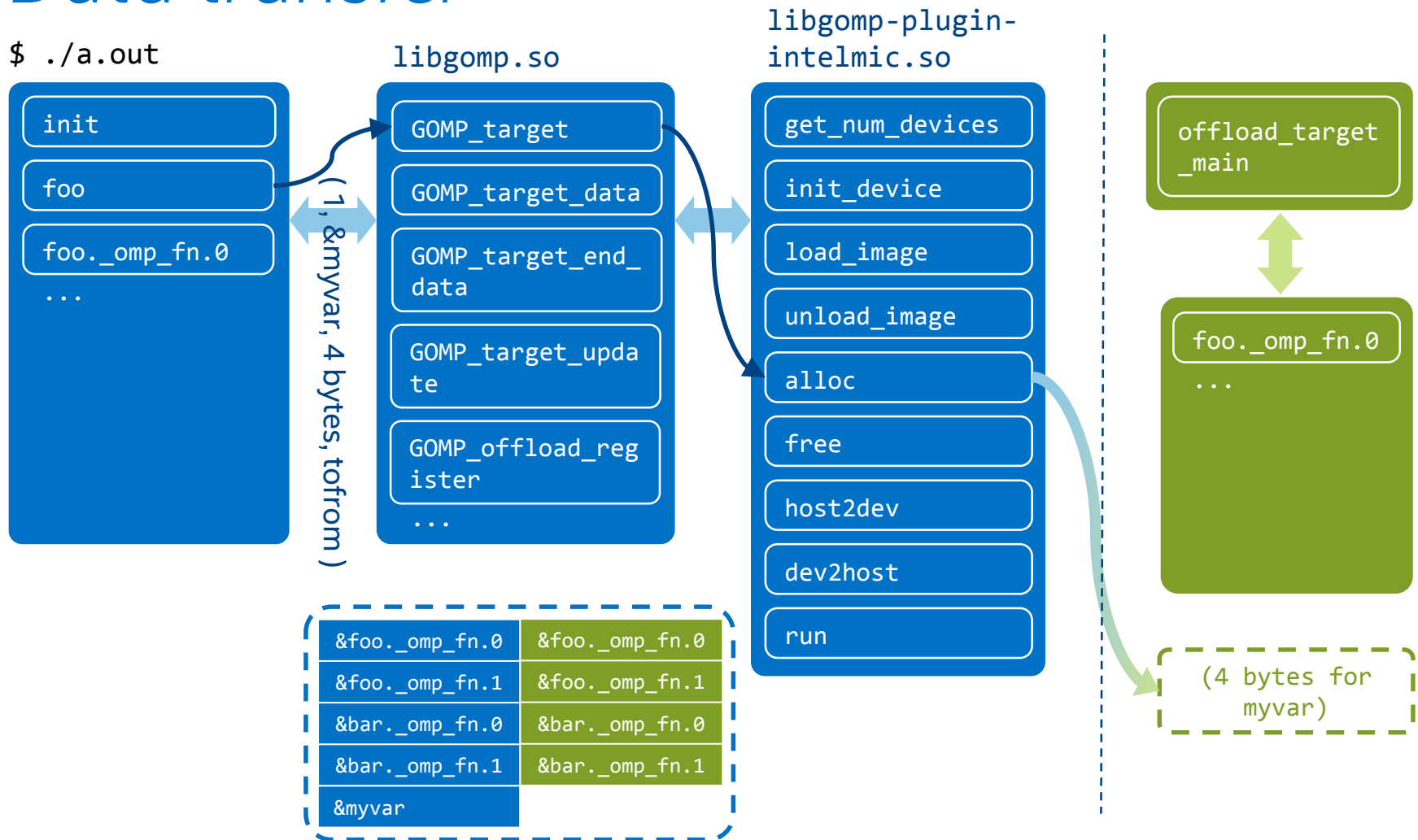
# Implementation in GCC

## Data transfer



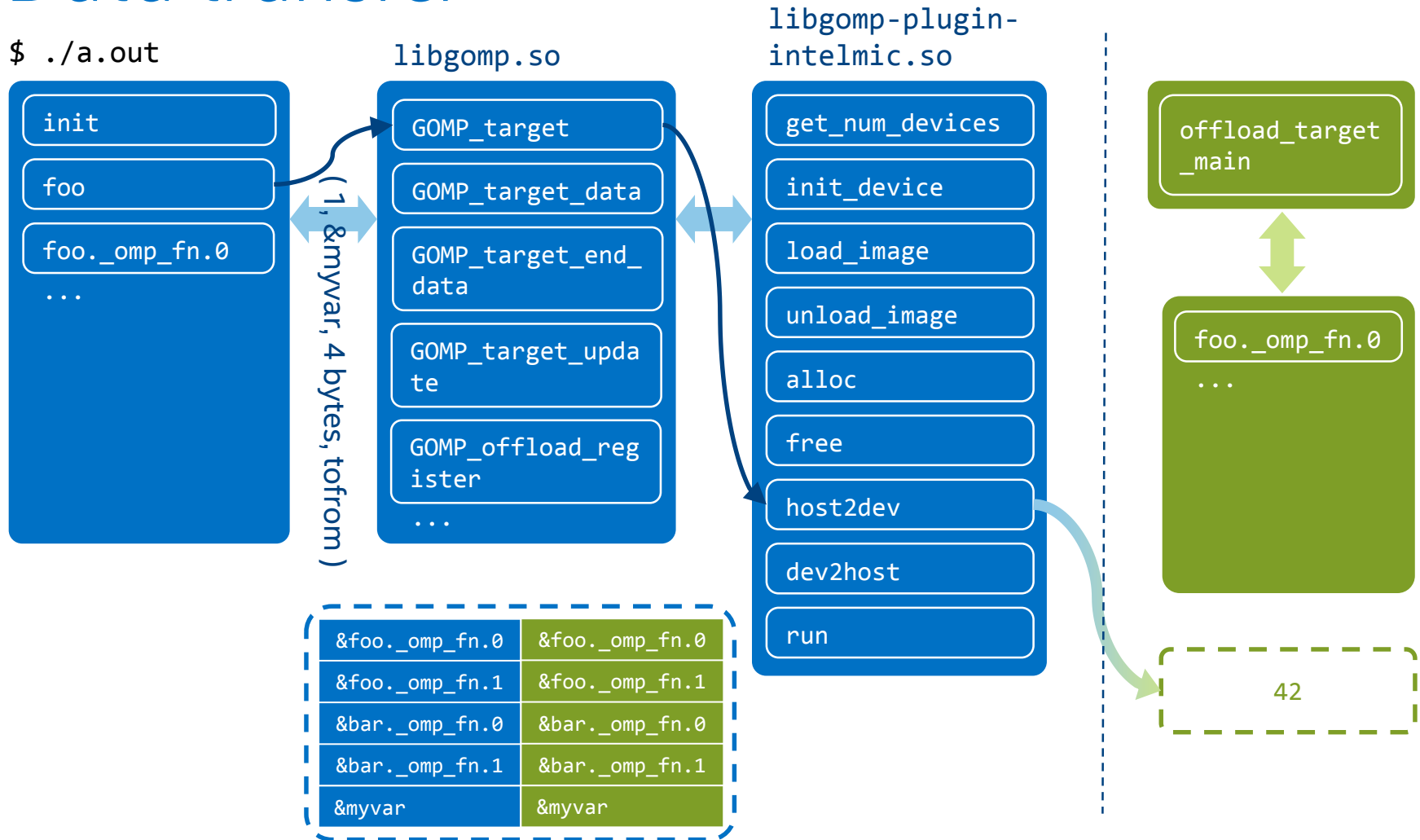
# Implementation in GCC

## Data transfer



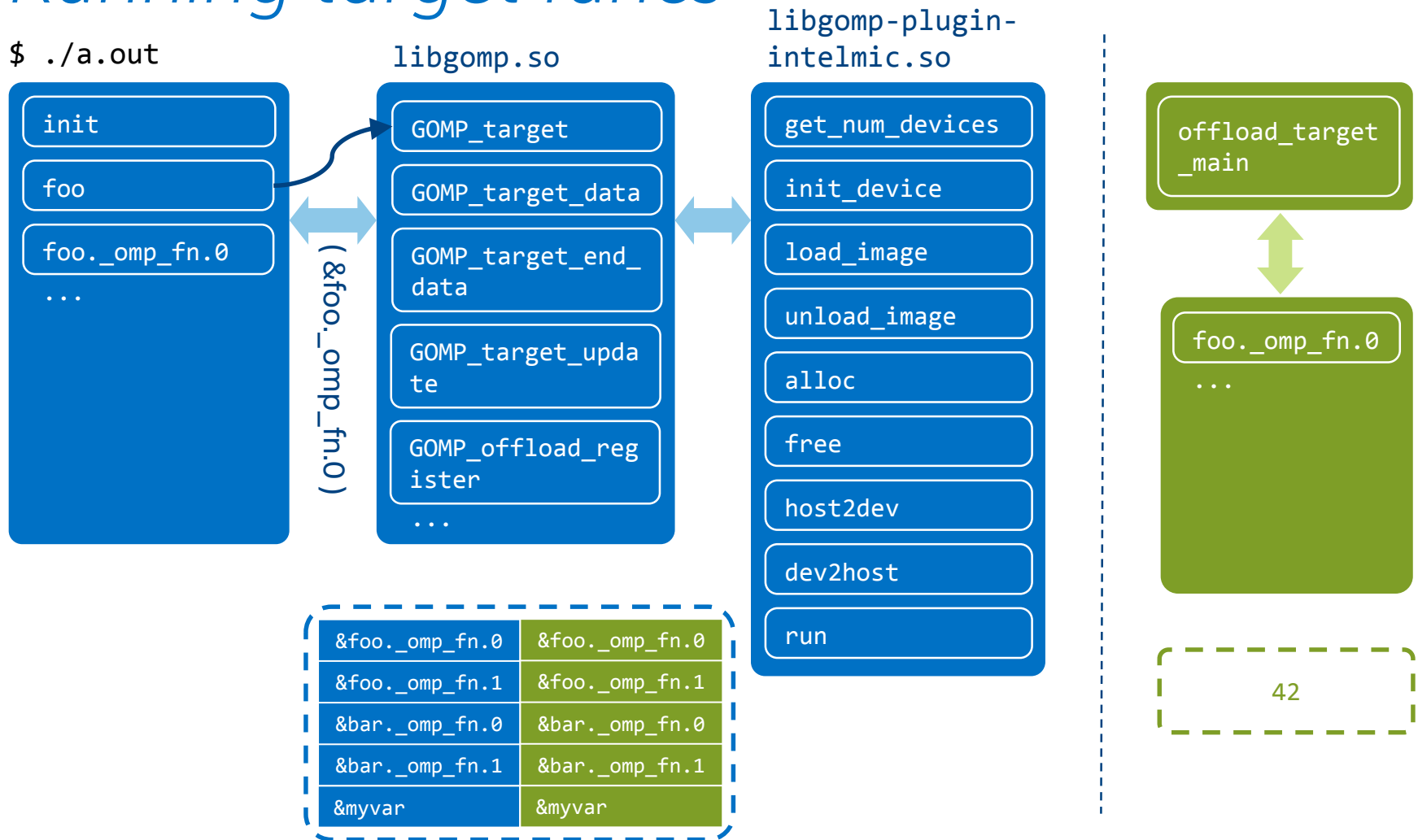
# Implementation in GCC

## Data transfer



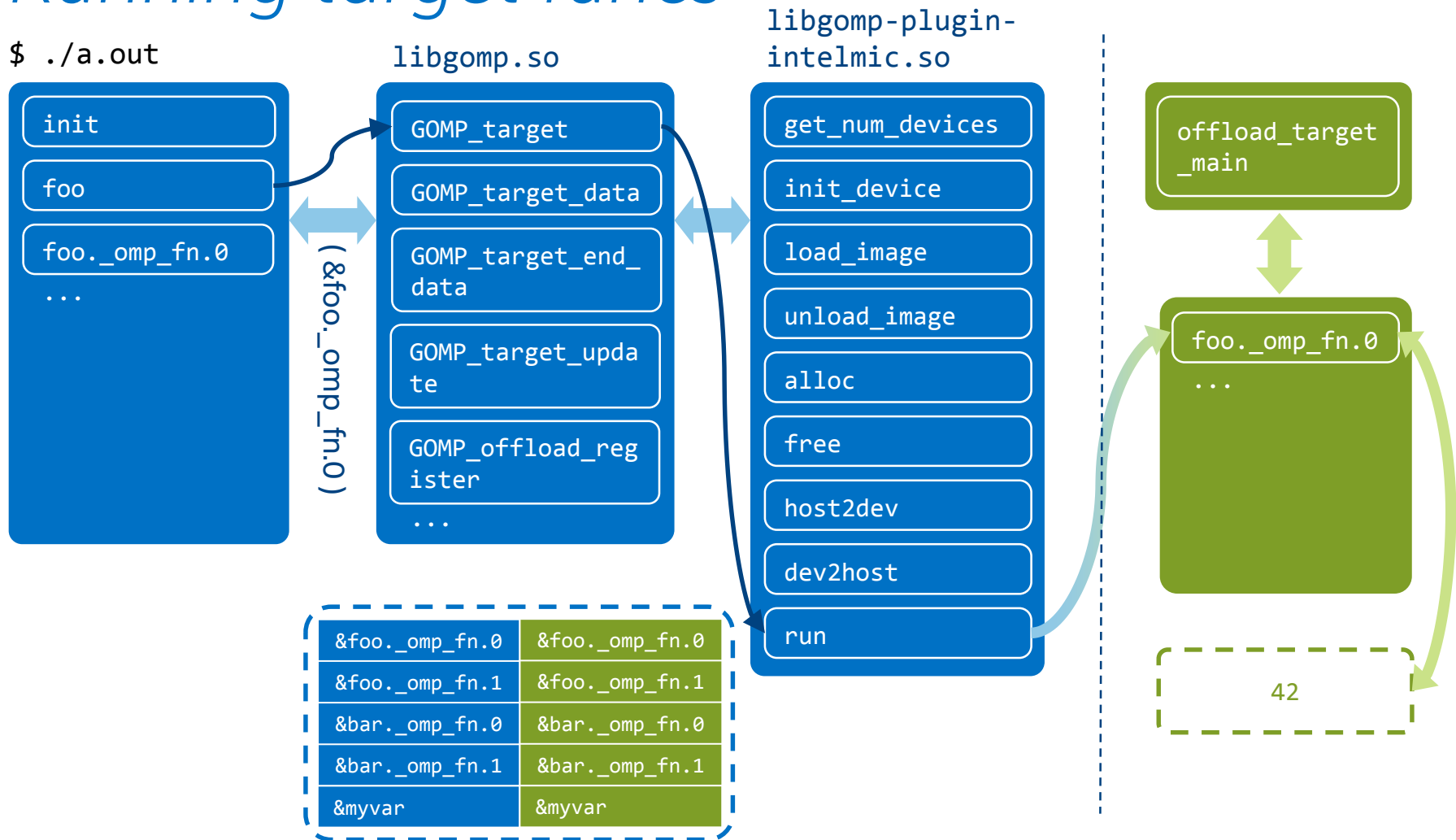
# Implementation in GCC

## Running target funcs



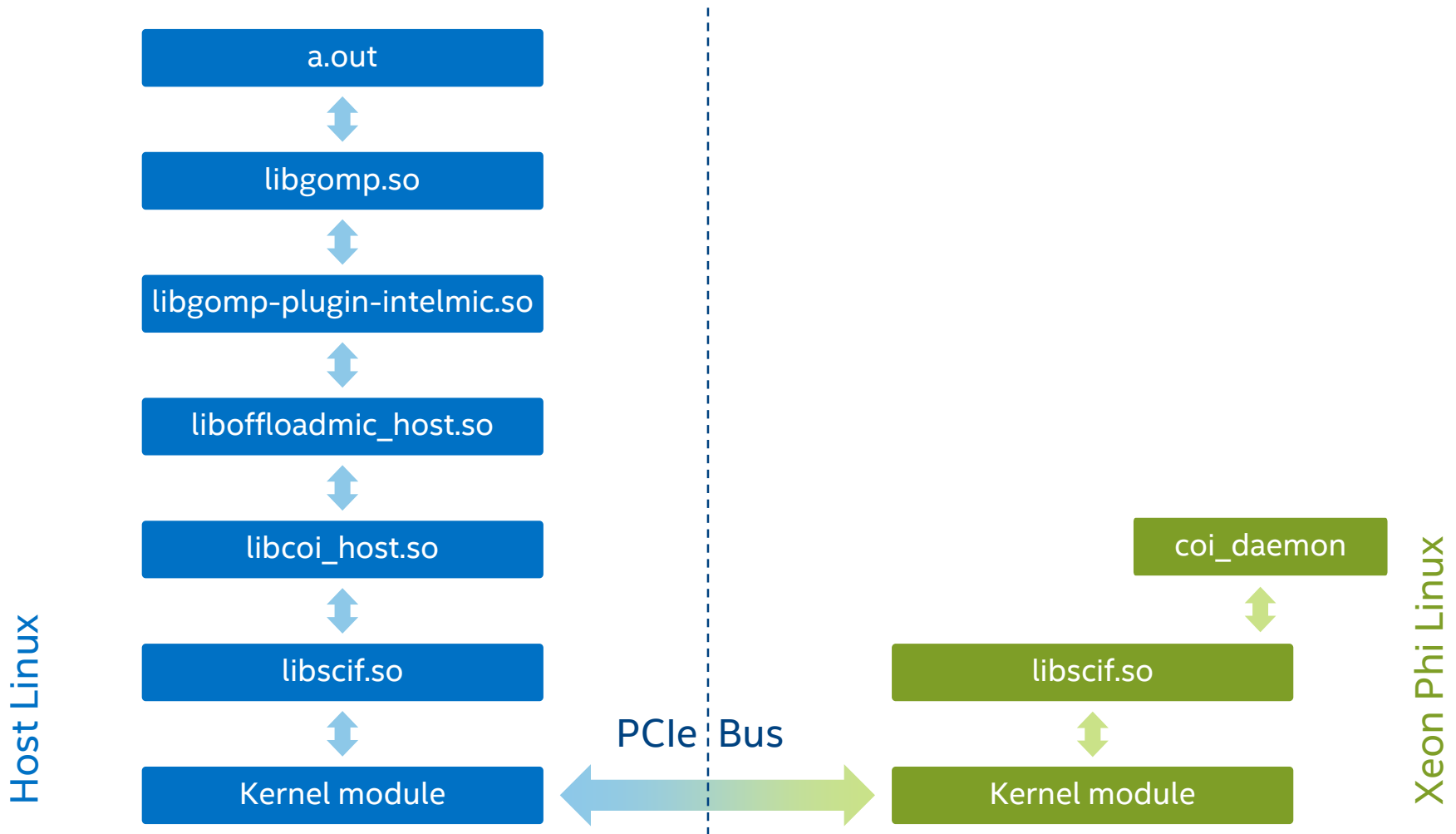
# Implementation in GCC

## Running target funcs



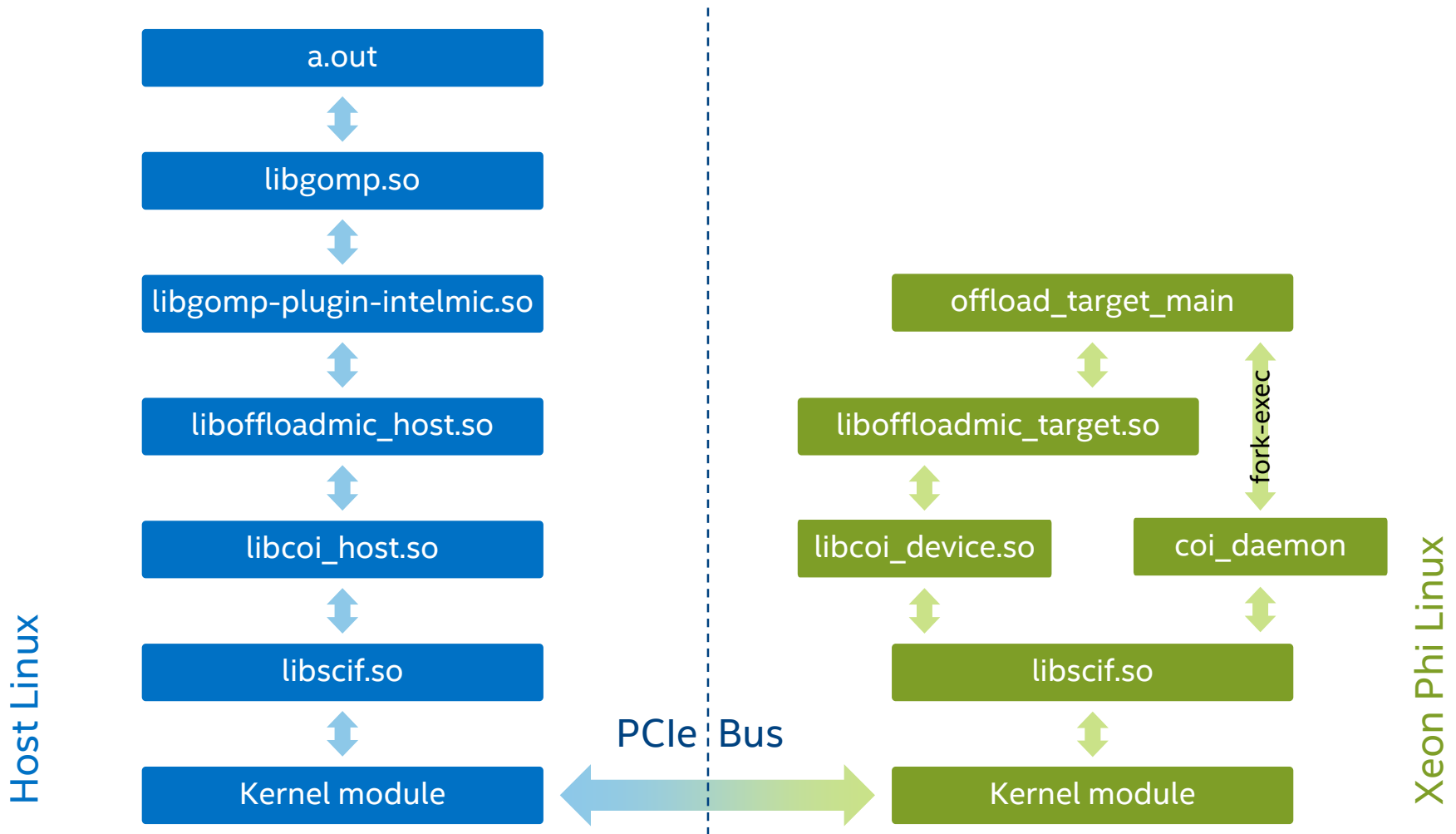
# Implementation in GCC

## Execution on Intel MIC



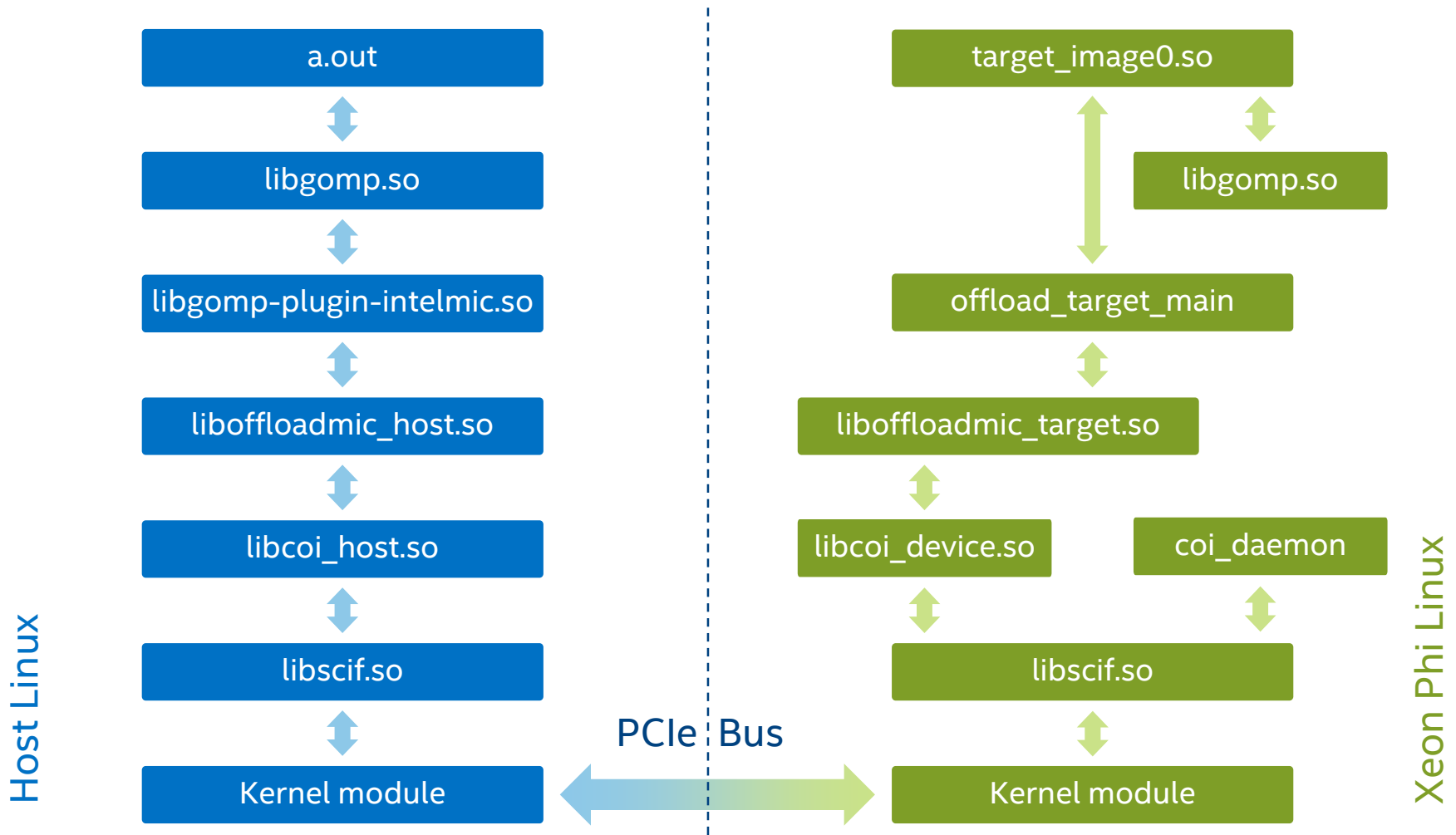
# Implementation in GCC

## Execution on Intel MIC



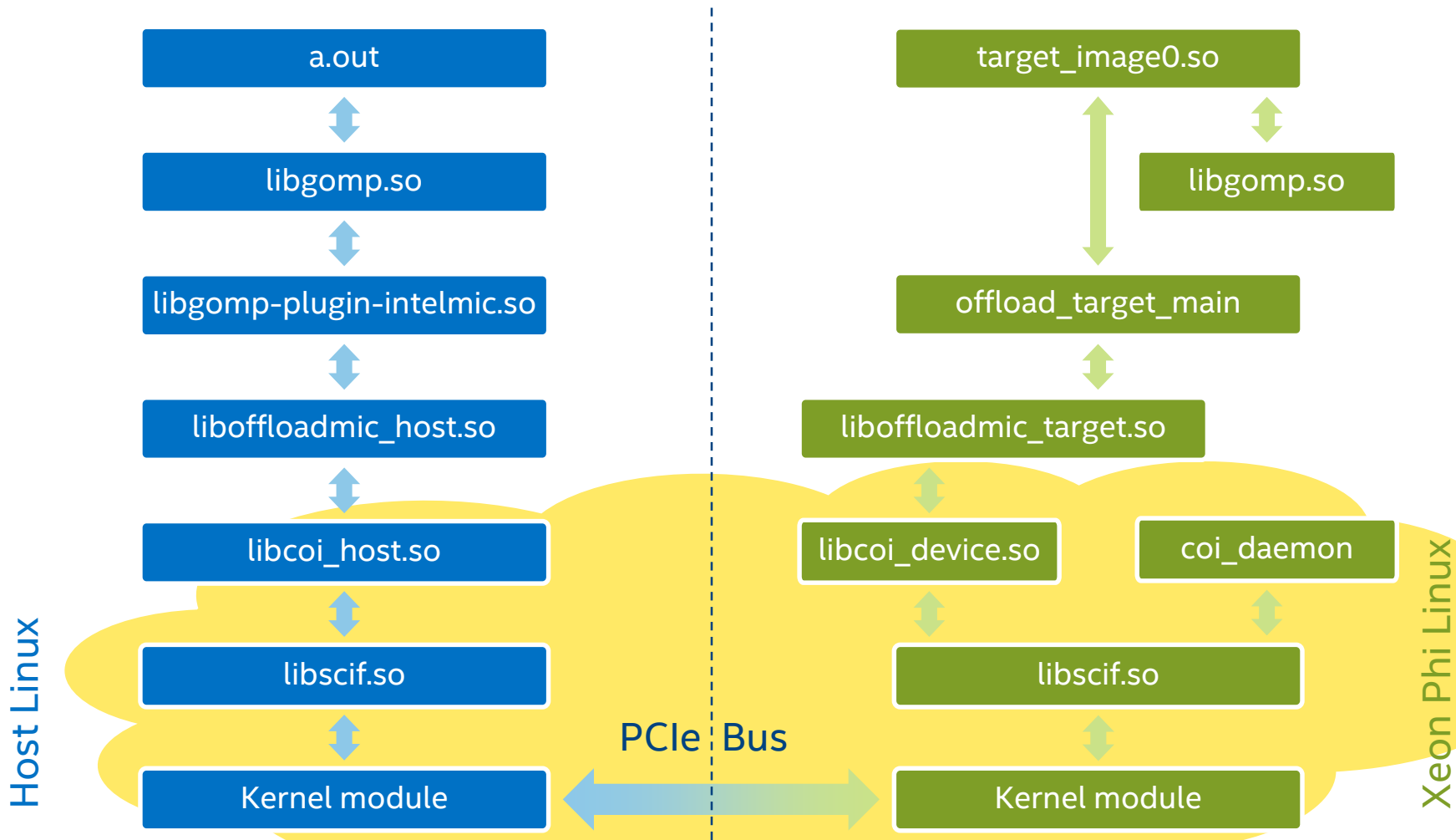
# Implementation in GCC

## Execution on Intel MIC

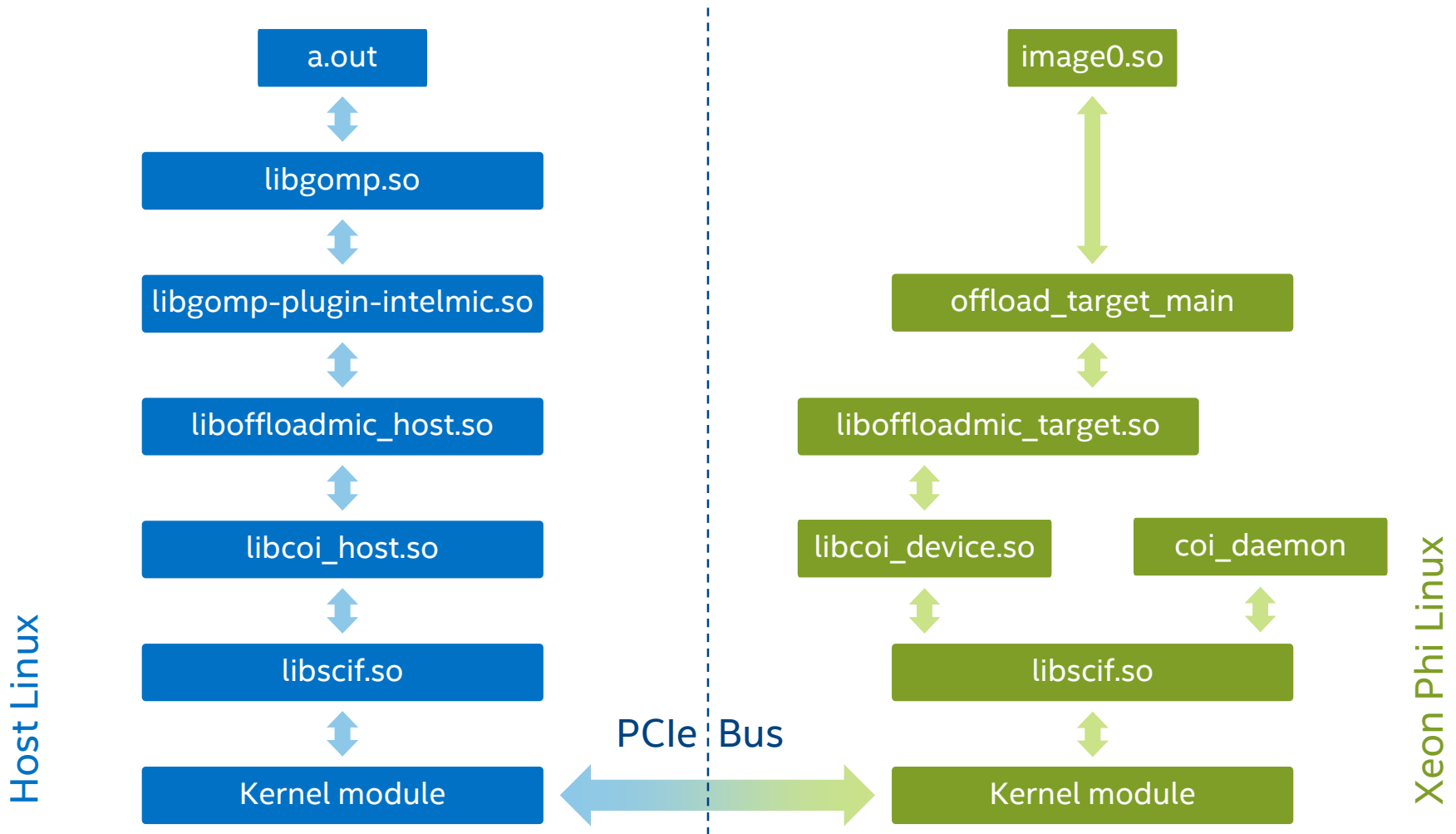


# Implementation in GCC Execution on Intel MIC

## Manycore Platform Software Stack

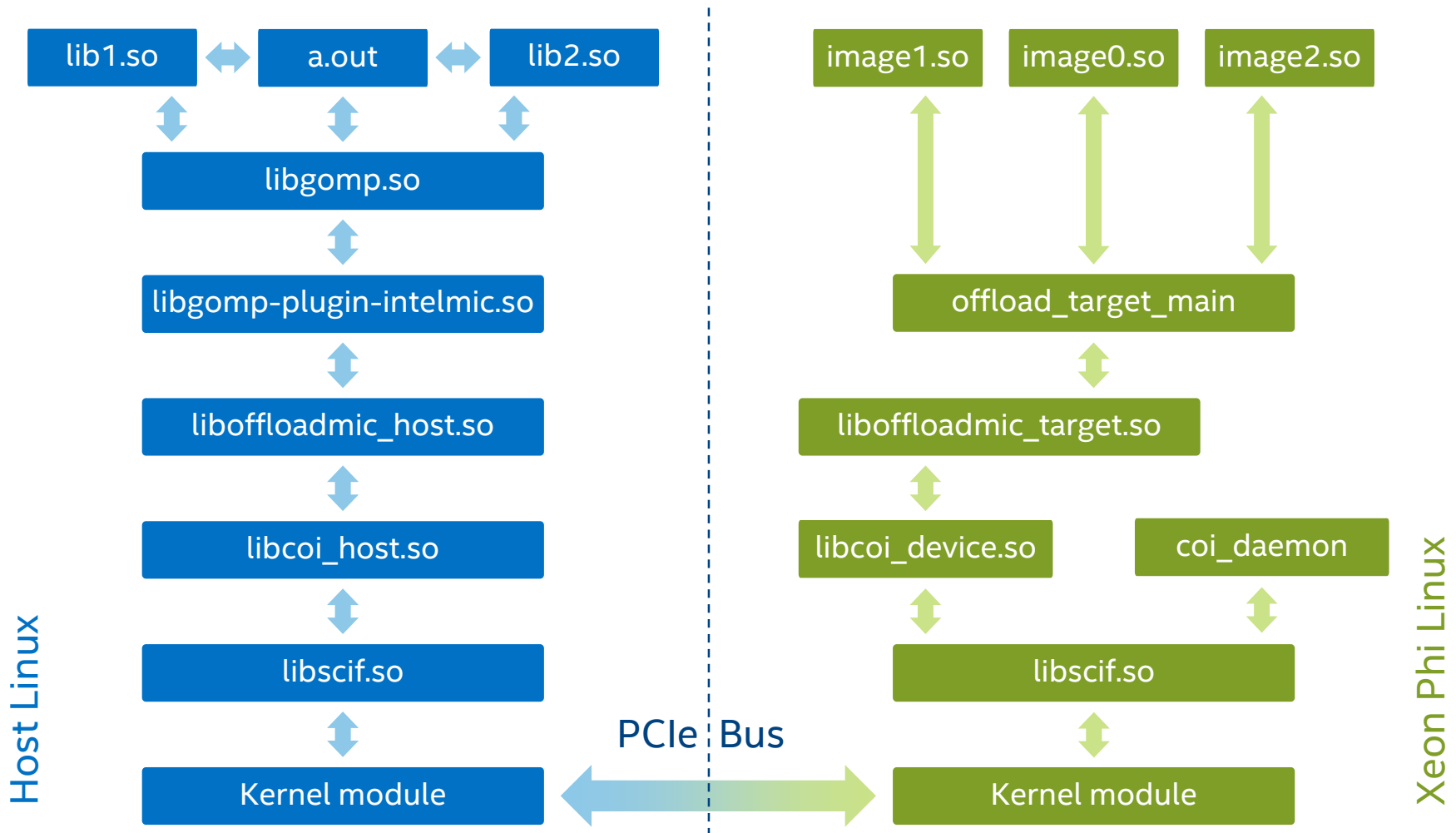


# Implementation in GCC Offloading from DSO



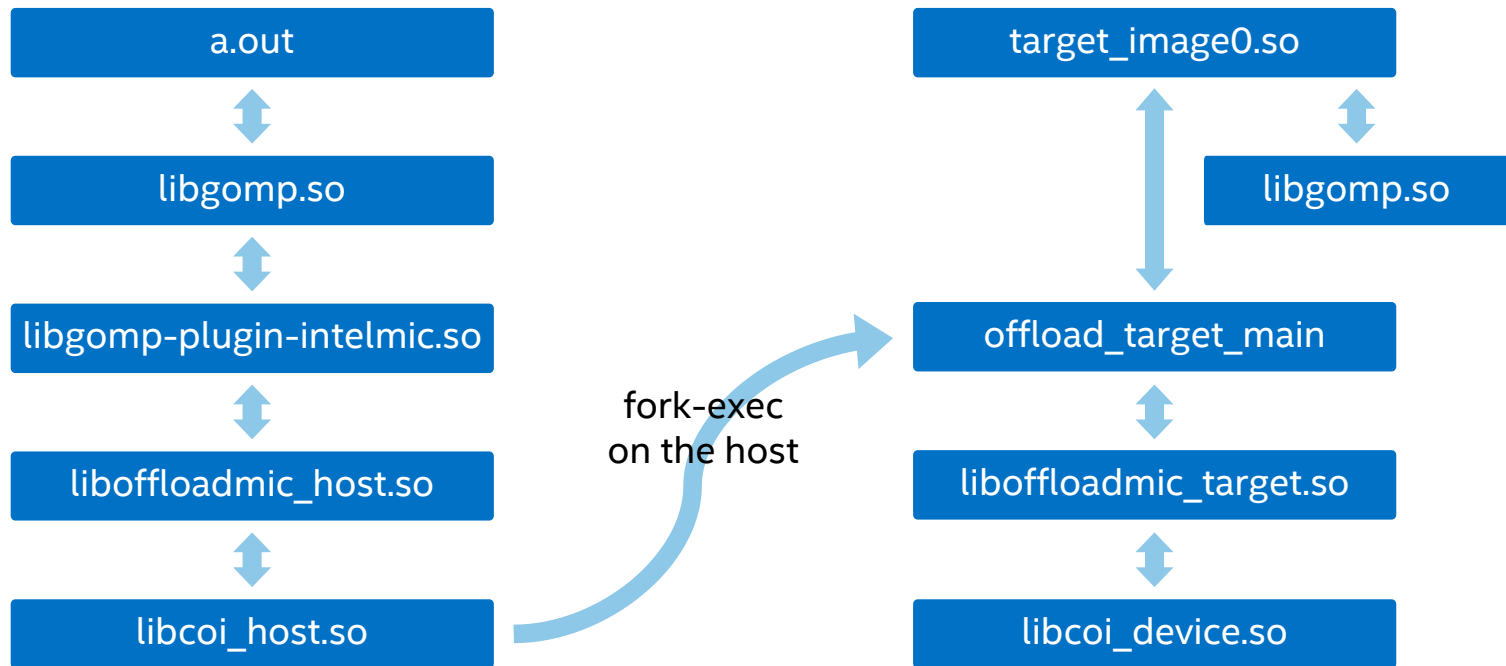
# Implementation in GCC

## Offloading from DSO



# Implementation in GCC

## Intel MIC emulation



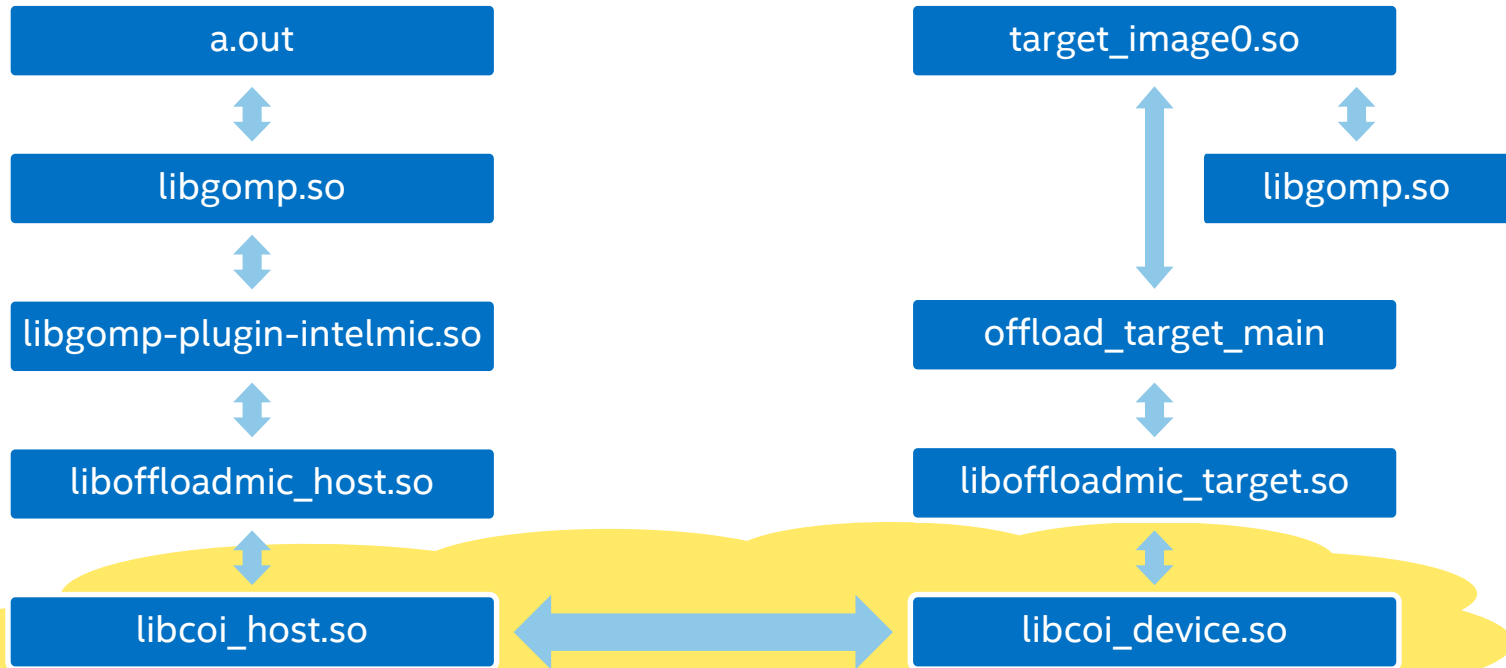
Host Linux

# Implementation in GCC

## Intel MIC emulation

COI emulator

Host Linux



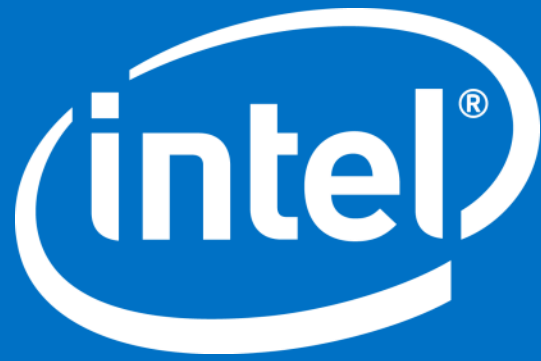
# Current Status

OpenMP 4.0 offloading support introduced in GCC v5

- Intel Xeon Phi (a.k.a. KNL) is supported as back-end
- Work is needed in PTX

OpenMP 4.1 development is in progress

- Main offloading features: asynchronous execution, unstructured data mapping
- Target is GCC v6



# OpenMP 4.0

## *#pragma omp target data*

```
int foo (int *input, int N)
{
    int res;
    int *tmp = (int *) malloc (N * sizeof (int));

    #pragma omp target data map(alloc: tmp[0:N]) map(to: input[0:N]) map(from: res)
    {
        #pragma omp target
        {
            #pragma omp parallel for
            for (int i = 0; i < N; i++)
                tmp[i] = input[i] * i;
        }

        do_something_on_host ();

        #pragma omp target
        {
            res = 0;

            #pragma omp parallel for reduction(+:res)
            for (int i = 0; i < N; i++)
                res += tmp[i];
        }
    }

    free (tmp);
    return res;
}
```

# OpenMP 4.0

## *#pragma omp target data*

```
int foo (int *input, int N)
{
    int res;
    int *tmp = (int *) malloc (N * sizeof (int));

    #pragma omp target data map(alloc: tmp[0:N]) map(to: input[0:N]) map(from: res)
    {
        #pragma omp target
        {
            #pragma omp parallel for
            for (int i = 0; i < N; i++)
                tmp[i] = input[i] * i;
        }

        do_something_on_host ();

        #pragma omp target
        {
            res = 0;

            #pragma omp parallel for reduction(+:res)
            for (int i = 0; i < N; i++)
                res += tmp[i];
        }
    }

    free (tmp);
    return res;
}
```

# OpenMP 4.0

## *#pragma omp target update*

```
int foo (int *input, int N)
{
    int res;
    int *tmp = (int *) malloc (N * sizeof (int));

    #pragma omp target data map(alloc: tmp[0:N]) map(to: input[0:N]) map(from: res)
    {
        #pragma omp target
        {
            for (int i = 0; i < N; i++)
                tmp[i] = input[i] * i;
        }

        update_input_array_on_the_host (input);

        #pragma omp target update to(input[0:N])

        #pragma omp target
        {
            res = 0;

            for (int i = 0; i < N; i++)
                res += tmp[i] * input[i];
        }
    }

    free (tmp);
    return res;
}
```

# OpenMP 4.0

## *#pragma omp target update*

```
int foo (int *input, int N)
{
    int res;
    int *tmp = (int *) malloc (N * sizeof (int));

    #pragma omp target data map(alloc: tmp[0:N]) map(to: input[0:N]) map(from: res)
    {
        #pragma omp target
        {
            for (int i = 0; i < N; i++)
                tmp[i] = input[i] * i;
        }

        update_input_array_on_the_host (input);

        #pragma omp target update to(input[0:N])

        #pragma omp target
        {
            res = 0;

            for (int i = 0; i < N; i++)
                res += tmp[i] * input[i];
        }
    }

    free (tmp);
    return res;
}
```

# OpenMP 4.0

## *#pragma omp {,end} declare target*

```
#pragma omp declare target
float Q[N][N];

float foo (const int i, const int j)
{
    return Q[i][j] * Q[j][i];
}
#pragma omp end declare target

float accum ()
{
    float res;

    #pragma omp target map(from:res)
    {
        res = 0.0;

        #pragma omp parallel for reduction(+:res)
        for (int i = 0; i < N; i++)
        {
            float tmp = 0.0;

            for (int j = 0; j < N; j++)
                tmp += foo (i, j);

            res += tmp;
        }
    }

    return res;
}
```

# OpenMP 4.0

## *#pragma omp {,end} declare target*

```
#pragma omp declare target
float Q[N][N];

float foo (const int i, const int j)
{
    return Q[i][j] * Q[j][i];
}
#pragma omp end declare target

float accum ()
{
    float res;

    #pragma omp target map(from:res)
    {
        res = 0.0;

        #pragma omp parallel for reduction(+:res)
        for (int i = 0; i < N; i++)
        {
            float tmp = 0.0;

            for (int j = 0; j < N; j++)
                tmp += foo (i, j);

            res += tmp;
        }
    }

    return res;
}
```

# OpenMP 4.1

## *#pragma omp target {enter,exit} data*

```
void init_data (int *input)
{
    #pragma omp target enter data map(to: input[0:N])
}

void fini_data (int *input)
{
    #pragma omp target exit data map(delete: input[0:N])
}
```

```
int foo (int *input)
{
    int res;

    init_data (input);

    #pragma omp target map(from: res)
    {
        res = 0;

        for (int i = 0; i < N; i++)
            res += input[i];
    }

    fini_data (input);

    return res;
}
```

```
int foo (int *input)
{
    int res;

    #pragma omp target data map(to: input[0:N])
    {
        #pragma omp target map(from: res)
        {
            res = 0;

            for (int i = 0; i < N; i++)
                res += input[i];
        }
    }

    return res;
}
```

# OpenMP 4.1

## Asynchronous execution: depend clause

```
void foo (double *myarray)
{
    int d1, d2, d3;

    /* Target task 1 */
    #pragma omp target nowait depend(out: d1)
    {
        do_something_on_target_1 (myarray);
    }

    /* Target task 2 */
    #pragma omp target nowait depend(in: d1) depend(out: d2)
    {
        do_something_on_target_2 (myarray);
    }

    /* Target task 3 */
    #pragma omp target nowait depend(in: d1) depend(out: d3)
    {
        do_something_on_target_3 (myarray);
    }

    /* Target task 4 */
    #pragma omp target exit data \
        map(from: myarray[0:N]) depend(in: d2, d3)
}
```

Has to be completed  
before Task 2 and Task 3  
can be executed

Can be executed in parallel