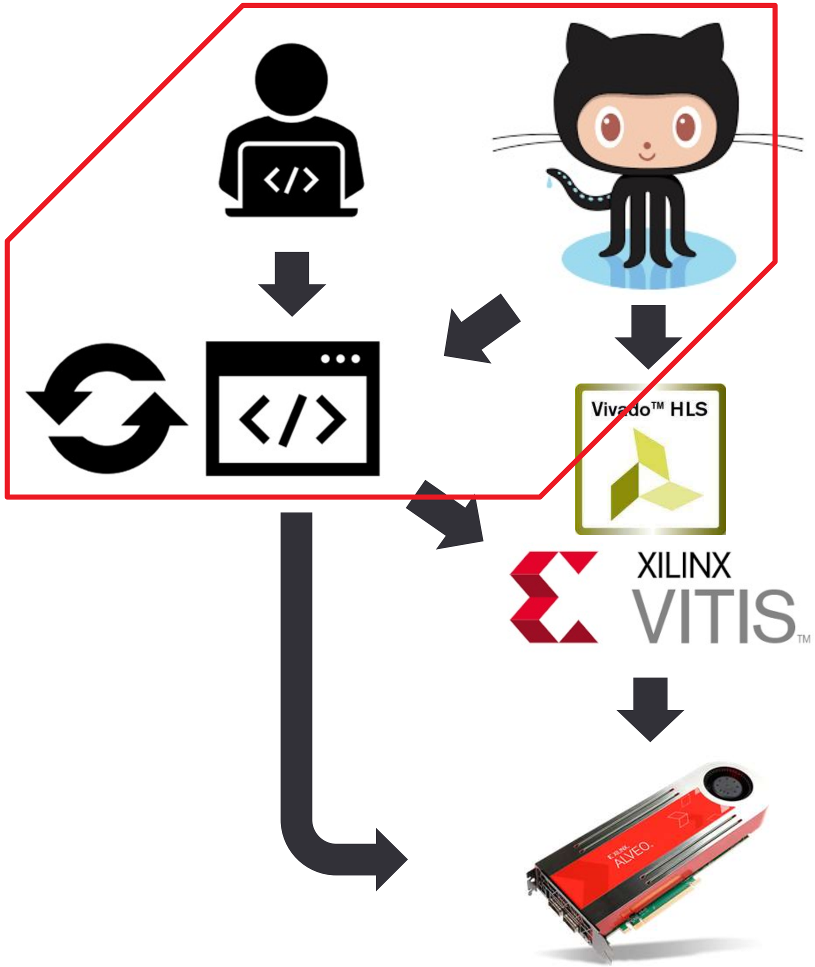
AMD

Using ACCL Emulation/Simulation Flow

Lucian Petrica

Steps to build ACCL-enabled FPGA application

- Clone ACCL repo(s):
  - https://github.com/Xilinx/ACCL
  - https://github.com/Xilinx/pyaccl
- Build and verify your distributed application
  - With or without FPGA acceleration
  - Using ACCL HLS code emulator and RTL simulator
- Build appropriate CCLO kernel and plugins
- Link with Vitis
  - Against platform, protocol offload engine (POE), and any application kernels
- Deploy to FPGA



ACCL emulation flow demonstration

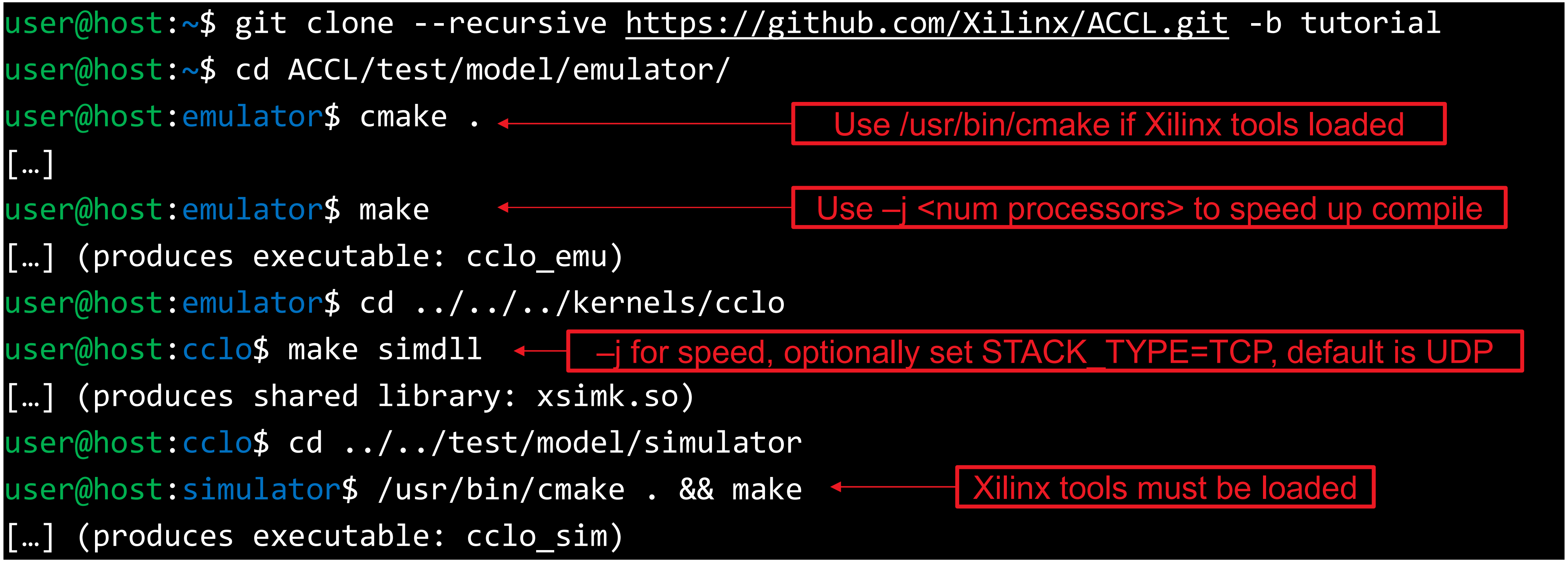
- Learning objectives:
  - become acquainted with ACCL use-cases and API
  - Learn how to use the simulator and emulator for building host- or PL-driven applications

- Part 1: Host-driven applications

- Part 2: PL-driven, streaming applications

Cloning the Repo
Building Simulator and Emulator
Running Tests

Cloning the Repo, building Simulator, Emulator



```
user@host:~$ git clone --recursive https://github.com/Xilinx/ACCL.git -b tutorial
user@host:~$ cd ACCL/test/model/emulator/
user@host:emulator$ cmake .
[…]
user@host:emulator$ make
[…] (produces executable: cclo_emu)
user@host:emulator$ cd ../../../kernels/cclo
user@host:cclo$ make simdll
[…] (produces shared library: xsimk.so)
user@host:cclo$ cd ../../test/model/simulator
user@host:simulator$ /usr/bin/cmake . && make
[…] (produces executable: cclo_sim)
```

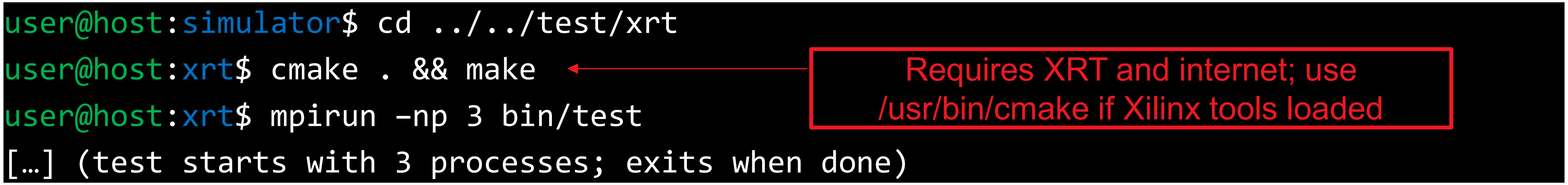
Use /usr/bin/cmake if Xilinx tools loaded

Use –j <num processors> to speed up compile

–j for speed, optionally set STACK_TYPE=TCP, default is UDP

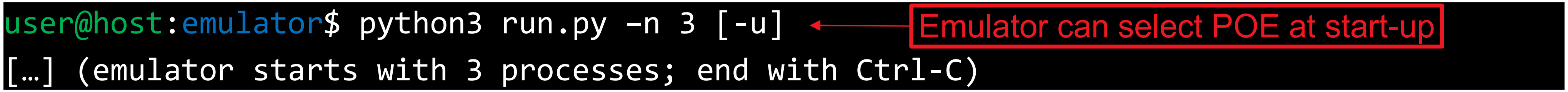
Xilinx tools must be loaded

Building and running Tests



```
user@host:simulator$ cd ../../test/xrt
user@host:xrt$ cmake . && make
user@host:xrt$ mpirun -np 3 bin/test
[…] (test starts with 3 processes; exits when done)
```

Requires XRT and internet; use /usr/bin/cmake if Xilinx tools loaded



```
user@host:emulator$ python3 run.py -n 3 [-u]
[…] (emulator starts with 3 processes; end with Ctrl-C)
```

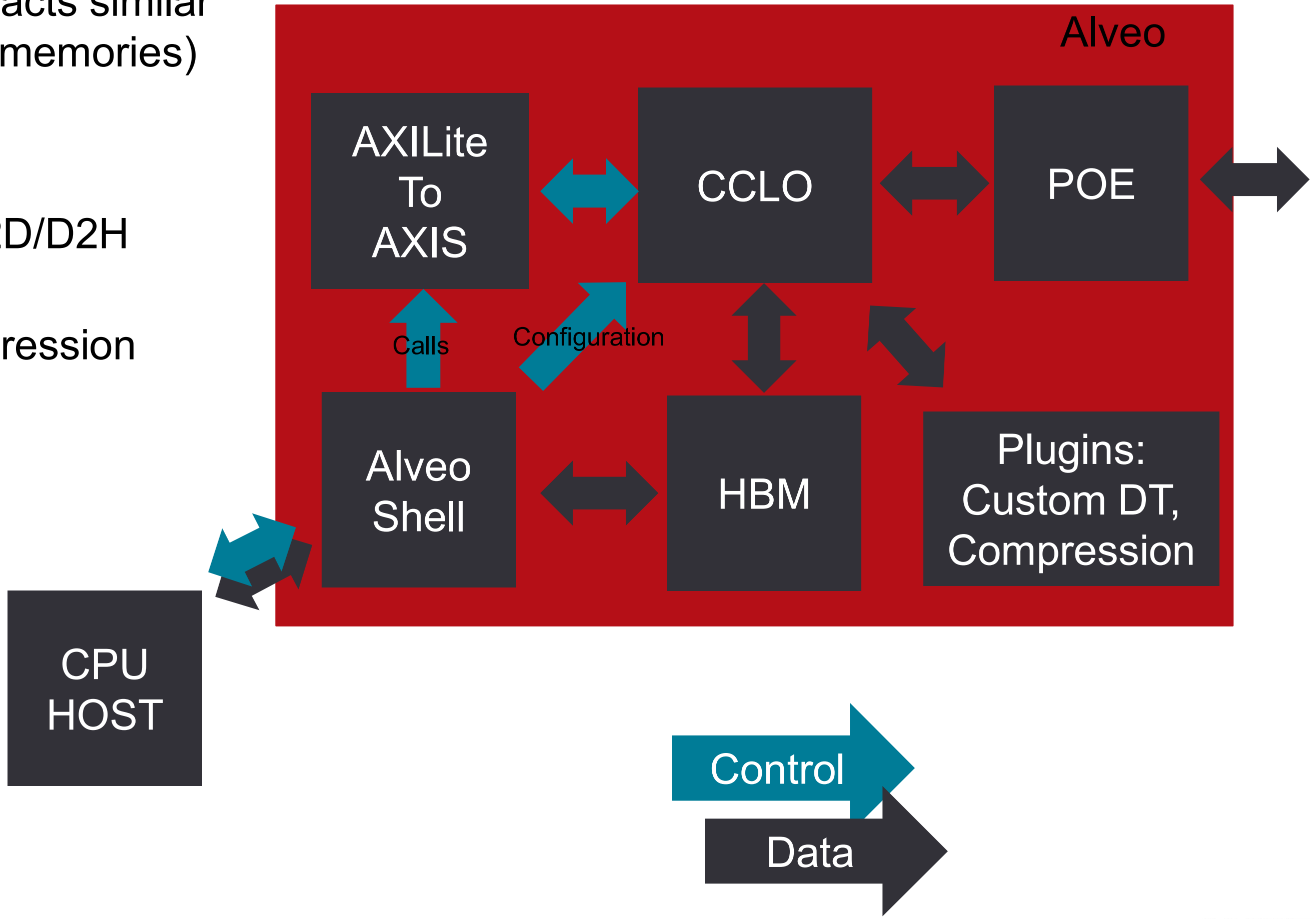
Emulator can select POE at start-up

Example host-driven ACCL Application
Scatter - Vadd - Gather

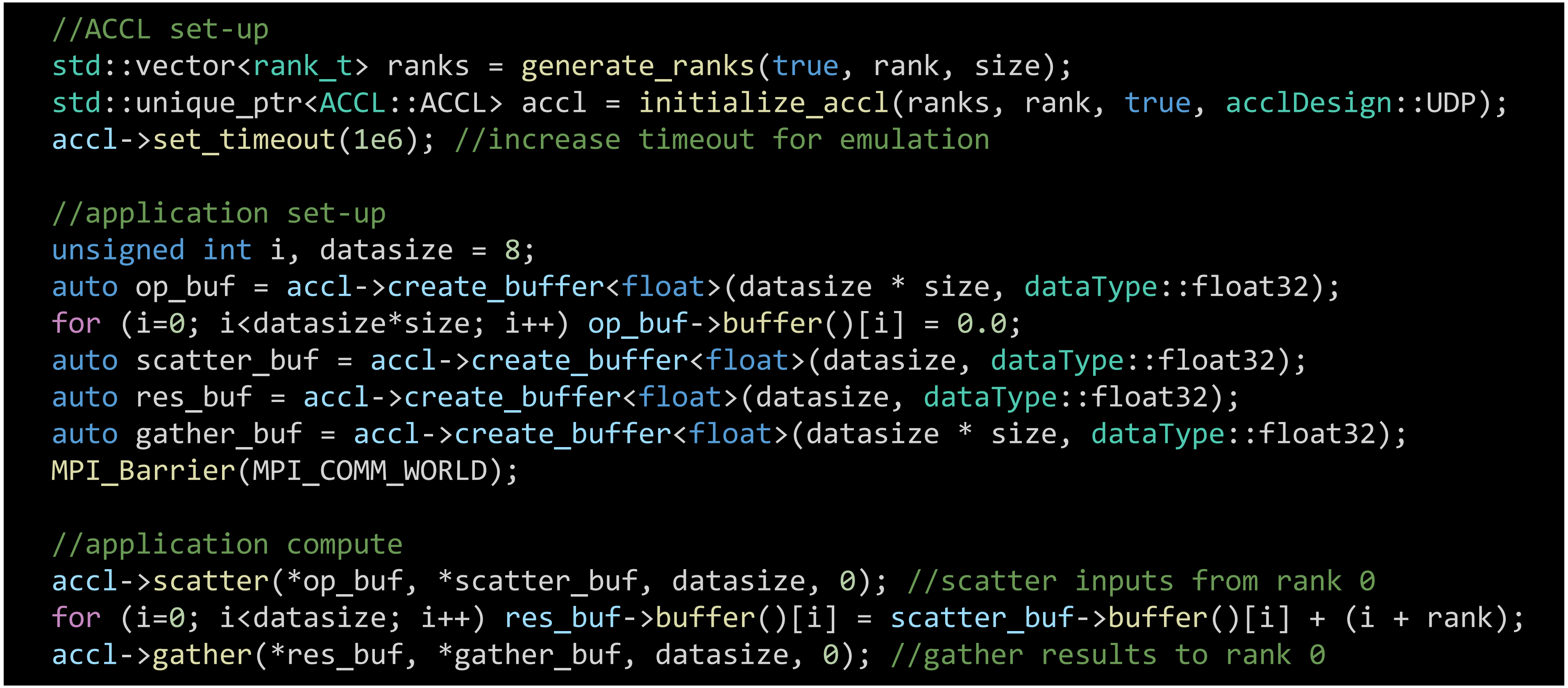
Typical ACCL system for host-driven applications

- Most generic way of using ACCL, FPGA acts similar to smart NIC (moves data between host memories)
- Host configures ACCL
- Host issues ACCL calls
- Data moves via FPGA memories and H2D/D2H copies
- Possibly traversing plugins for e.g. compression

- Relevant examples: ACCL XRT tests

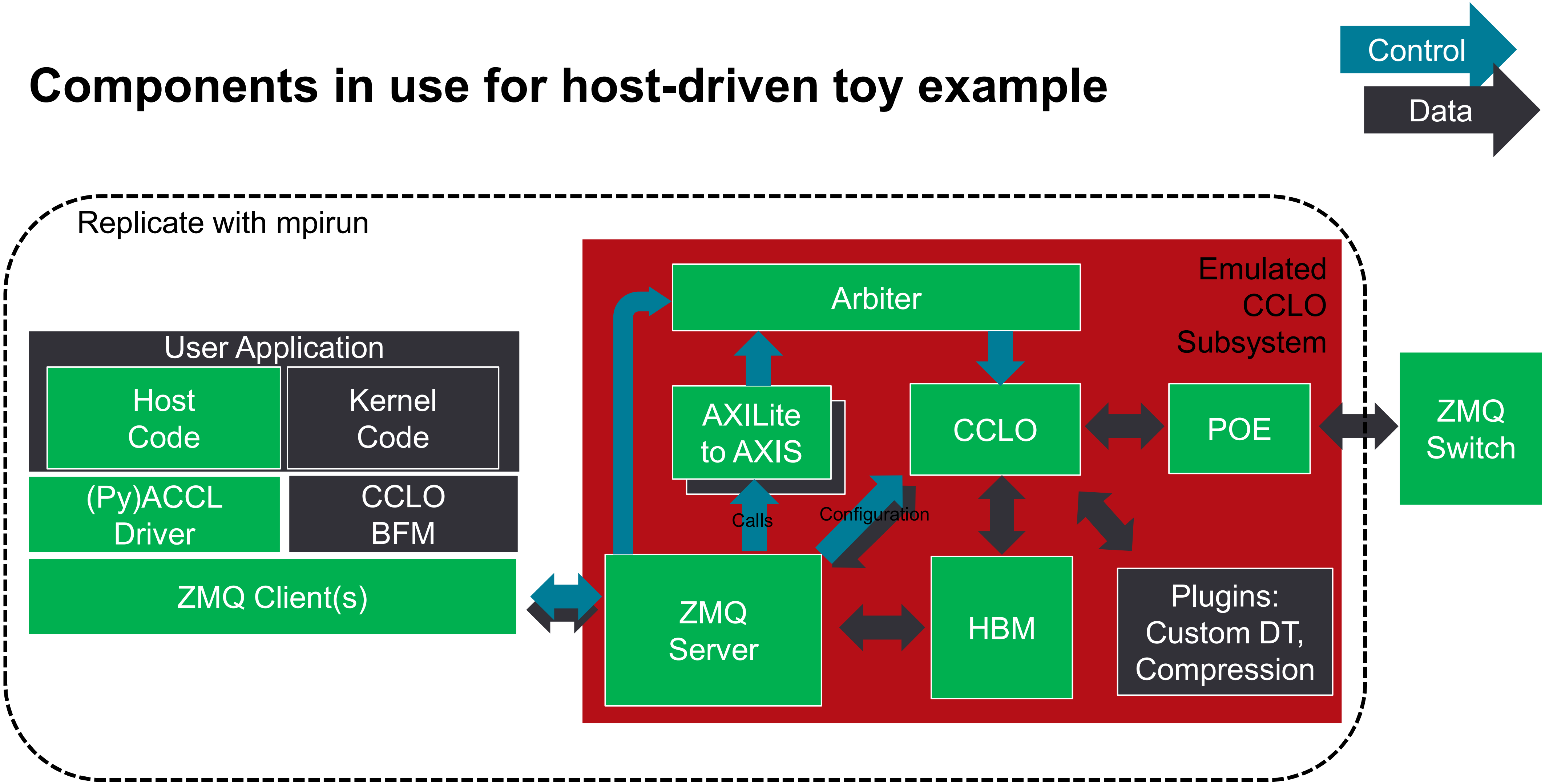


Code for host-driven toy application



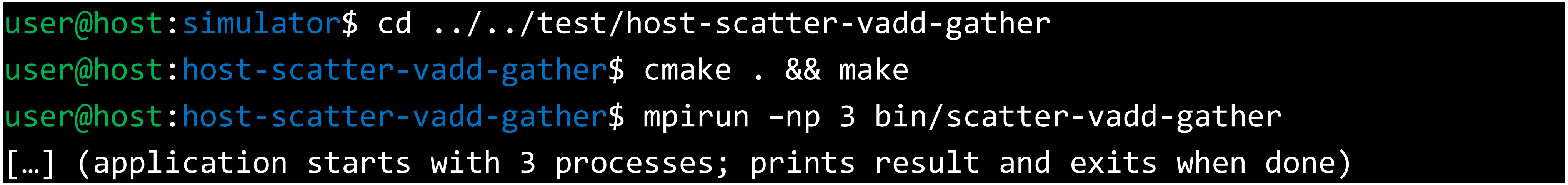
```
//ACCL set-up
std::vector<rank_t> ranks = generate_ranks(true, rank, size);
std::unique_ptr<ACCL::ACCL> accl = initialize_accl(ranks, rank, true, acclDesign::UDP);
accl->set_timeout(1e6); //increase timeout for emulation

//application set-up
unsigned int i, datasize = 8;
auto op_buf = accl->create_buffer<float>(datasize * size, dataType::float32);
for (i=0; i<datasize*size; i++) op_buf->buffer()[i] = 0.0;
auto scatter_buf = accl->create_buffer<float>(datasize, dataType::float32);
auto res_buf = accl->create_buffer<float>(datasize, dataType::float32);
auto gather_buf = accl->create_buffer<float>(datasize * size, dataType::float32);
MPI_Barrier(MPI_COMM_WORLD);

//application compute
accl->scatter(*op_buf, *scatter_buf, datasize, 0); //scatter inputs from rank 0
for (i=0; i<datasize; i++) res_buf->buffer()[i] = scatter_buf->buffer()[i] + (i + rank);
accl->gather(*res_buf, *gather_buf, datasize, 0); //gather results to rank 0
```

Components in use for host-driven toy example



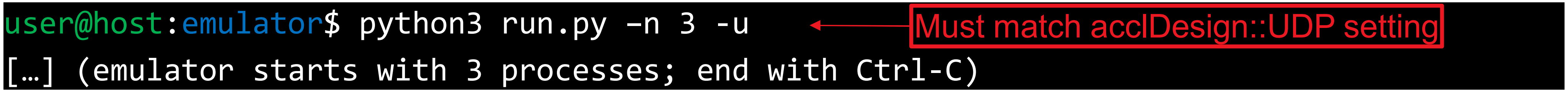
Running host-driven toy application

Start host code



```
user@host:simulator$ cd ../../test/host-scatter-vadd-gather
user@host:host-scatter-vadd-gather$ cmake . && make
user@host:host-scatter-vadd-gather$ mpirun -np 3 bin/scatter-vadd-gather
[…] (application starts with 3 processes; prints result and exits when done)
```

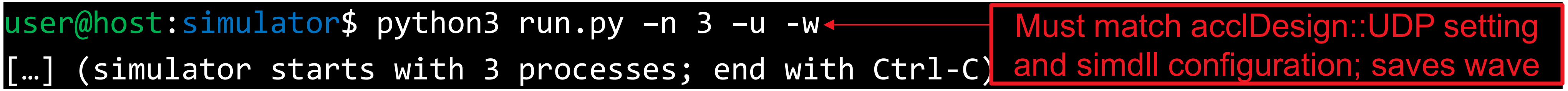
Start emulator



```
user@host:emulator$ python3 run.py -n 3 -u
[…] (emulator starts with 3 processes; end with Ctrl-C)
```

Must match acclDesign::UDP setting

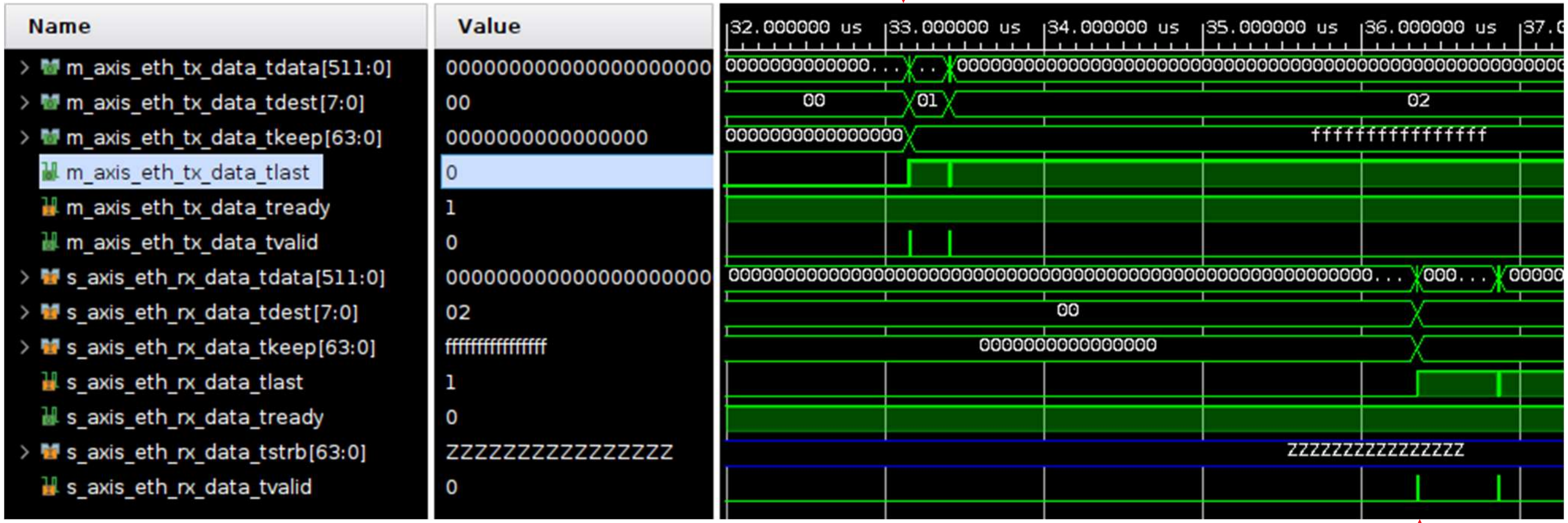
Or start simulator



```
user@host:simulator$ python3 run.py -n 3 -u -w
[…] (simulator starts with 3 processes; end with Ctrl-C)
```

Must match acclDesign::UDP setting and simdll configuration; saves wave

Sample Waveform



Name	Value
m_axis_eth_tx_data_tdata[511:0]	000000000000000000000
m_axis_eth_tx_data_tdest[7:0]	00
m_axis_eth_tx_data_tkeep[63:0]	0000000000000000
m_axis_eth_tx_data_tlast	0
m_axis_eth_tx_data_tready	1
m_axis_eth_tx_data_tvalid	0
s_axis_eth_rx_data_tdata[511:0]	000000000000000000000
s_axis_eth_rx_data_tdest[7:0]	02
s_axis_eth_rx_data_tkeep[63:0]	ffffffffffffffff
s_axis_eth_rx_data_tlast	1
s_axis_eth_rx_data_tready	0
s_axis_eth_rx_data_tstrb[63:0]	ZZZZZZZZZZZZZZZZ
s_axis_eth_rx_data_tvalid	0

Scatter (root sends to two peers)

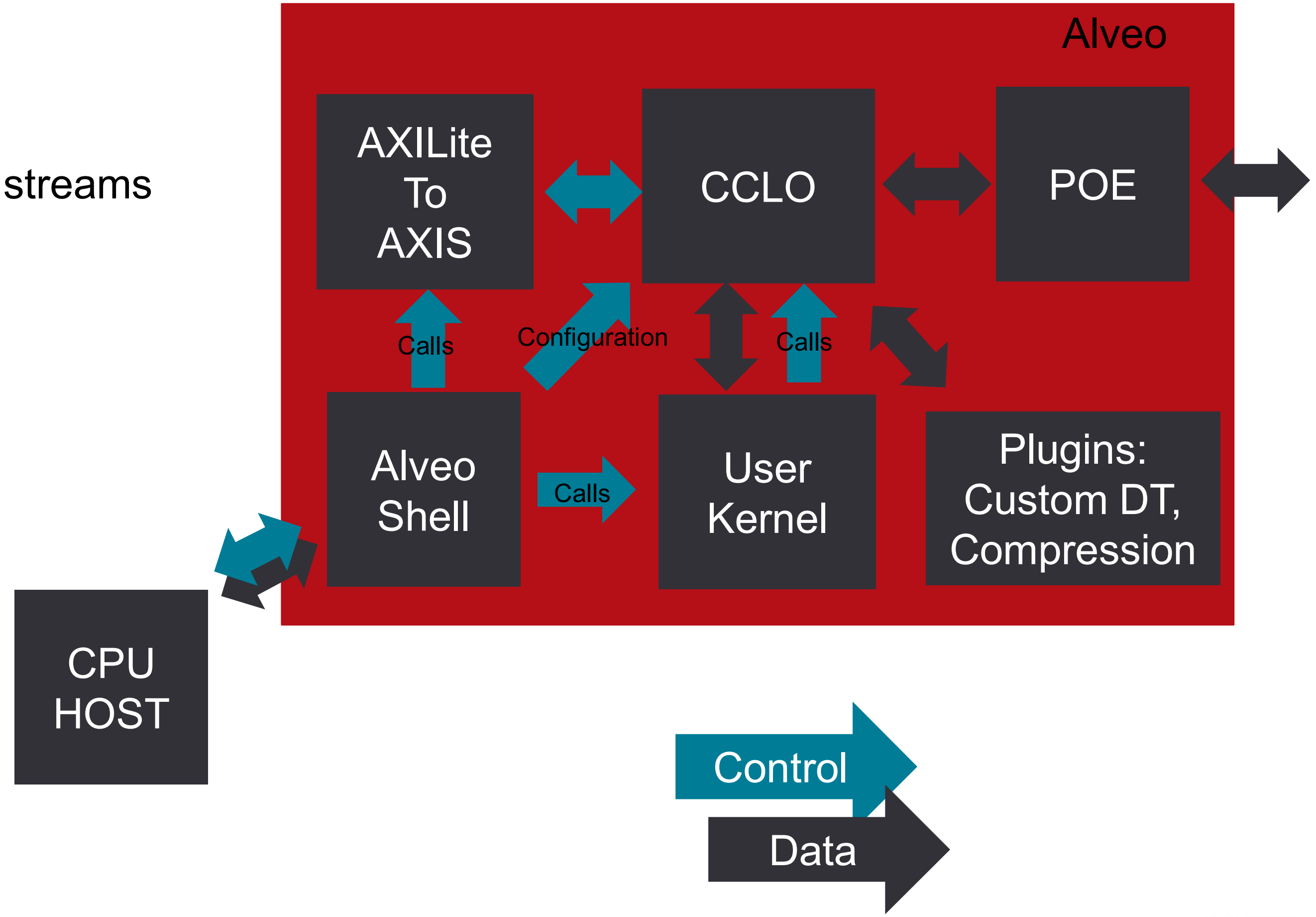
Gather (root receives from two peers)

Example PL-driven ACCL Application
Scatter – PL vadd – Streaming Gather

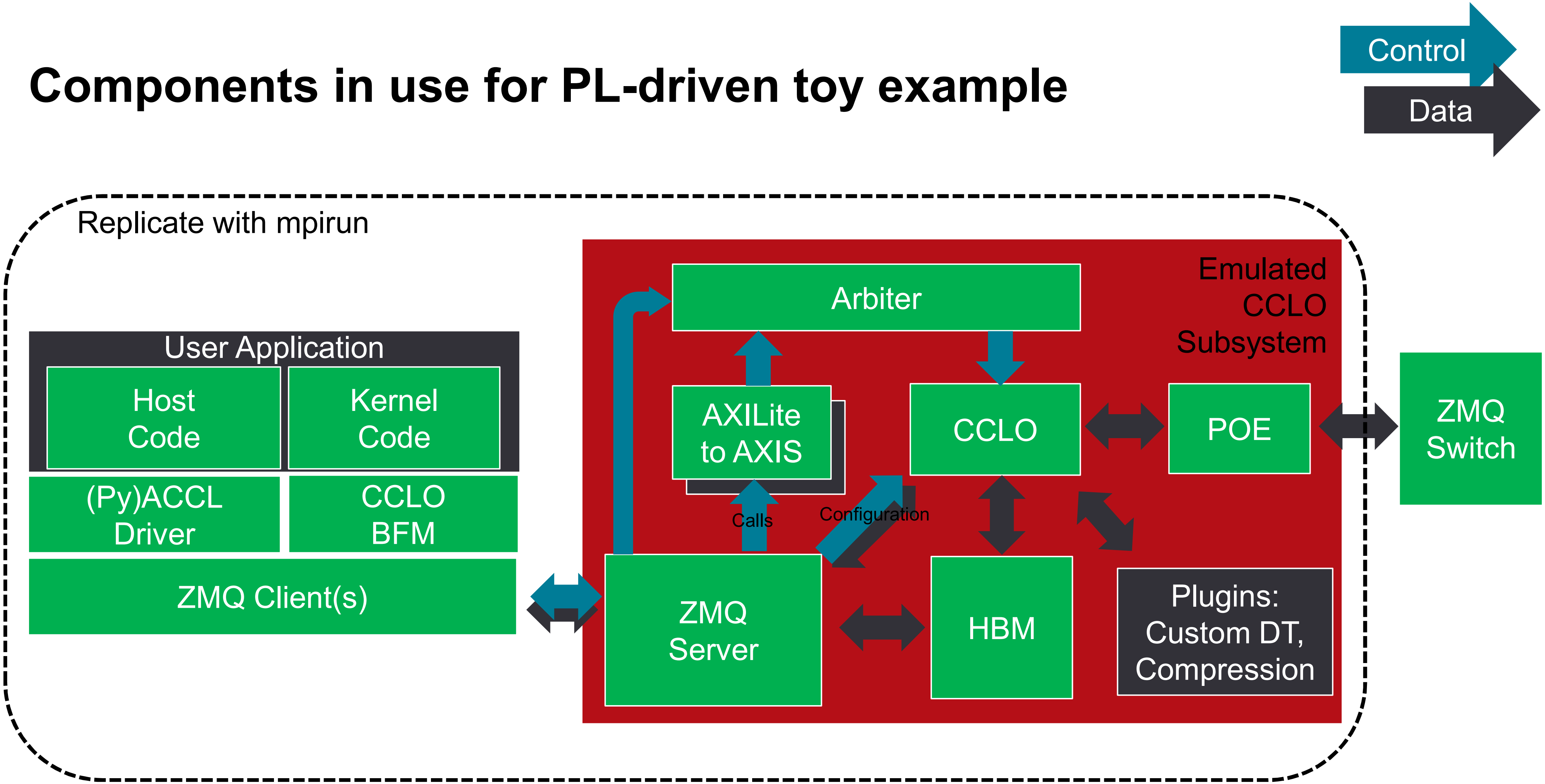
Typical ACCL system for PL-driven applications

- Suitable for low-latency applications
- Host configures ACCL
- PL Kernel issues calls
- PL kernel and CCLO exchange data via streams

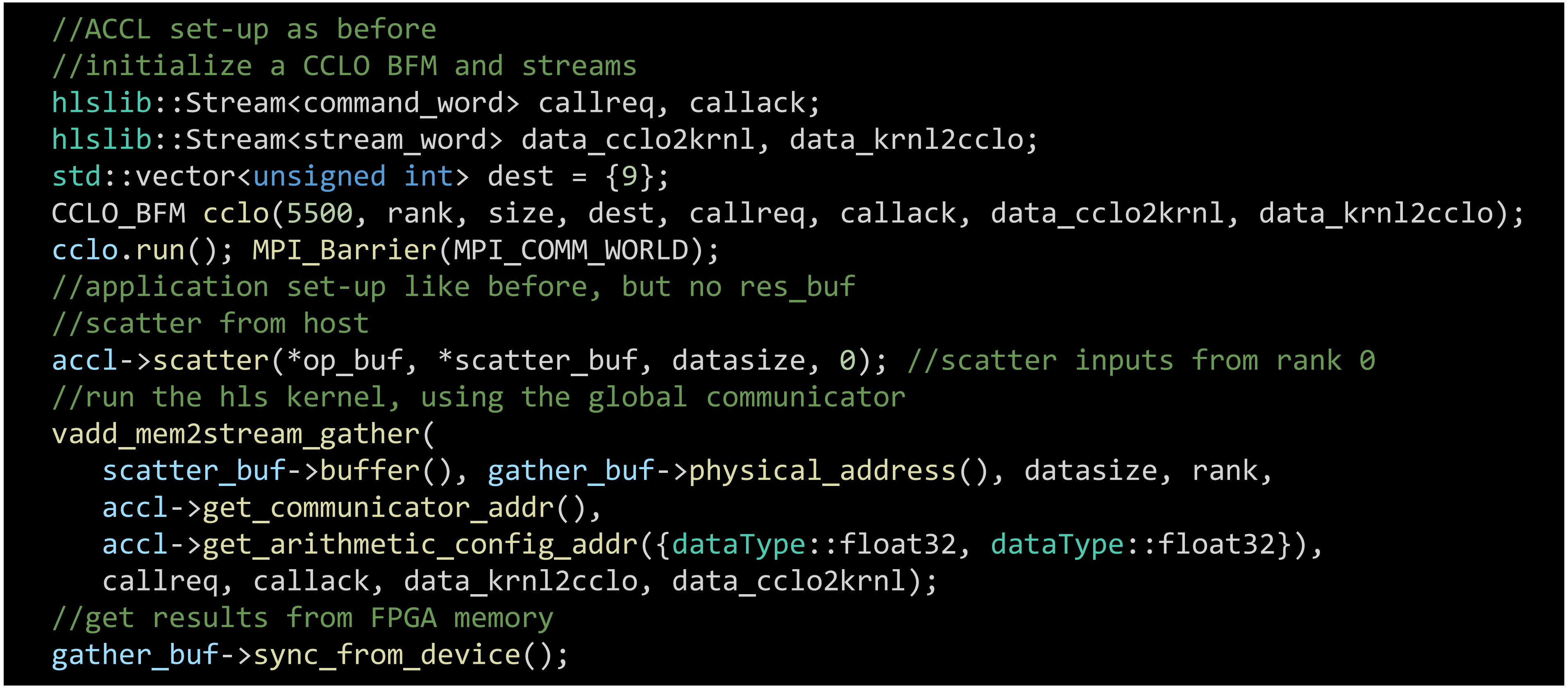
- Relevant example: ACCL HLS tests



Components in use for PL-driven toy example



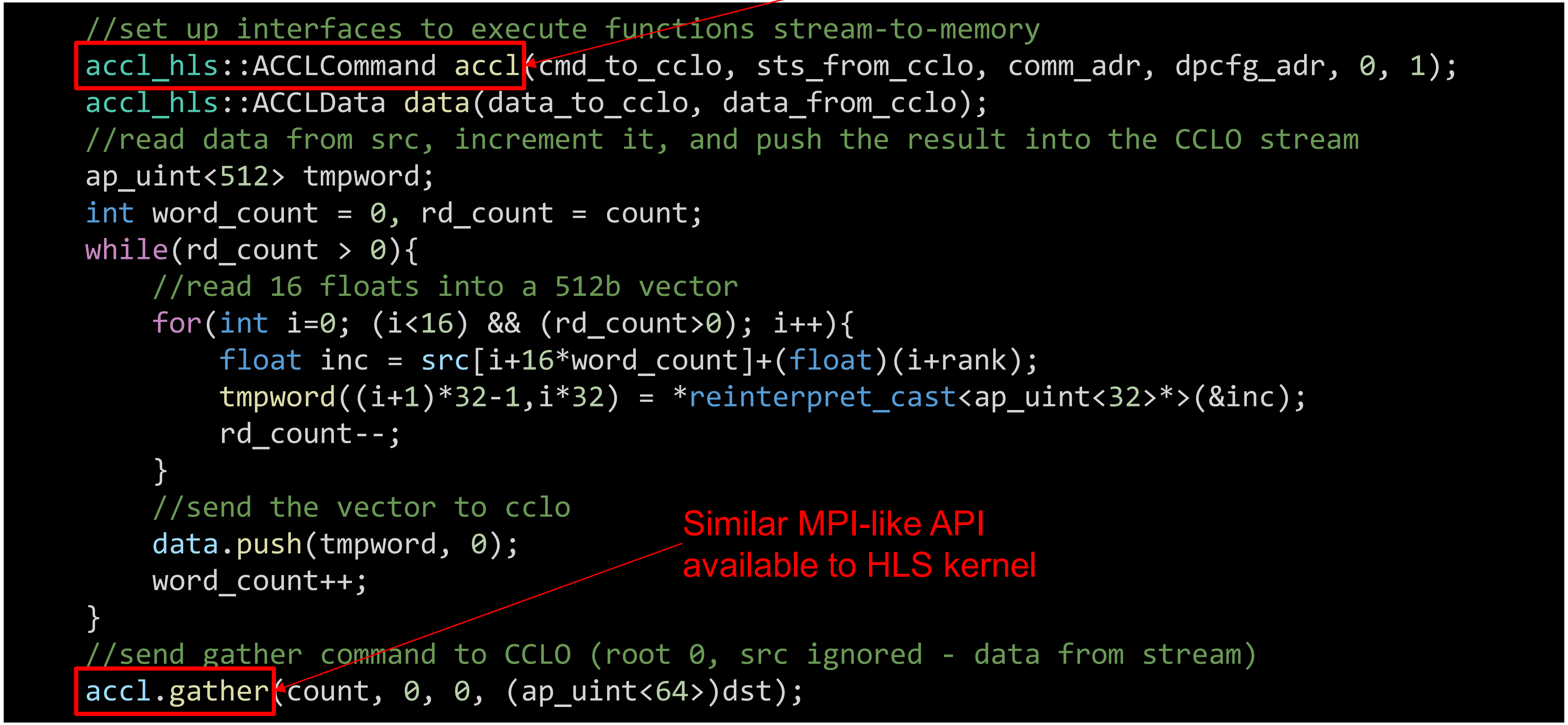
Host code for PL-driven toy application



```
//ACCL set-up as before
//initialize a CCLO BFM and streams
hlslib::Stream<command_word> callreq, callack;
hlslib::Stream<stream_word> data_cclo2krnl, data_krnl2cclo;
std::vector<unsigned int> dest = {9};
CCLO_BFM cclo(5500, rank, size, dest, callreq, callack, data_cclo2krnl, data_krnl2cclo);
cclo.run(); MPI_Barrier(MPI_COMM_WORLD);
//application set-up like before, but no res_buf
//scatter from host
accl->scatter(*op_buf, *scatter_buf, datasize, 0); //scatter inputs from rank 0
//run the hls kernel, using the global communicator
vadd_mem2stream_gather(
   scatter_buf->buffer(), gather_buf->physical_address(), datasize, rank,
   accl->get_communicator_addr(),
   accl->get_arithmetic_config_addr({dataType::float32, dataType::float32}),
   callreq, callack, data_krnl2cclo, data_cclo2krnl);
//get results from FPGA memory
gather_buf->sync_from_device();
```

Kernel code for PL-driven toy application

Hide command streams behind accl_hls interface



```
//set up interfaces to execute functions stream-to-memory
accl_hls::ACCLCommand accl(cmd_to_cclo, sts_from_cclo, comm_adr, dpcfg_adr, 0, 1);
accl_hls::ACCLData data(data_to_cclo, data_from_cclo);
//read data from src, increment it, and push the result into the CCLO stream
ap_uint<512> tmpword;
int word_count = 0, rd_count = count;
while(rd_count > 0){
    //read 16 floats into a 512b vector
    for(int i=0; (i<16) && (rd_count>0); i++){
        float inc = src[i+16*word_count]+(float)(i+rank);
        tmpword((i+1)*32-1,i*32) = *reinterpret_cast<ap_uint<32>*>(&inc);
        rd_count--;
    }
    //send the vector to cclo
    data.push(tmpword, 0);
    word_count++;
}
//send gather command to CCLO (root 0, src ignored - data from stream)
accl.gather(count, 0, 0, (ap_uint<64>)dst);
```

Similar MPI-like API available to HLS kernel

Running PL-driven toy application

Start host code

```
user@host:simulator$ cd ../../test/pl-scatter-vadd-gather
user@host:pl-scatter-vadd-gather$ cmake . && make
user@host:pl-scatter-vadd-gather$ mpirun –np 3 bin/scatter-vadd-gather
[…] (application starts with 3 processes; prints result and exits when done)
```

Allows user kernels to attach to CCLO streams

Start emulator

```
user@host:emulator$ python3 run.py –n 3 –u --no-kernel-loopback
[…] (emulator starts with 3 processes; end with Ctrl-C)
```

Or start simulator

```
user@host:simulator$ python3 run.py –n 3 –u –w --no-kernel-loopback
[…] (simulator starts with 3 processes; end with Ctrl-C)
```

Disclaimer & Attribution

Timelines, roadmaps, and/or product release dates shown in these slides are plans only and subject to change.

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

AMD