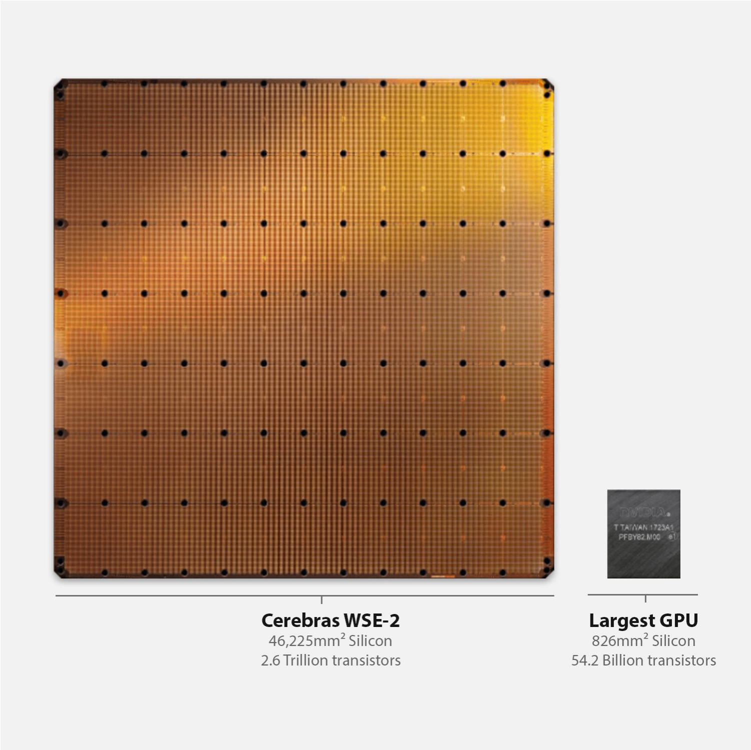
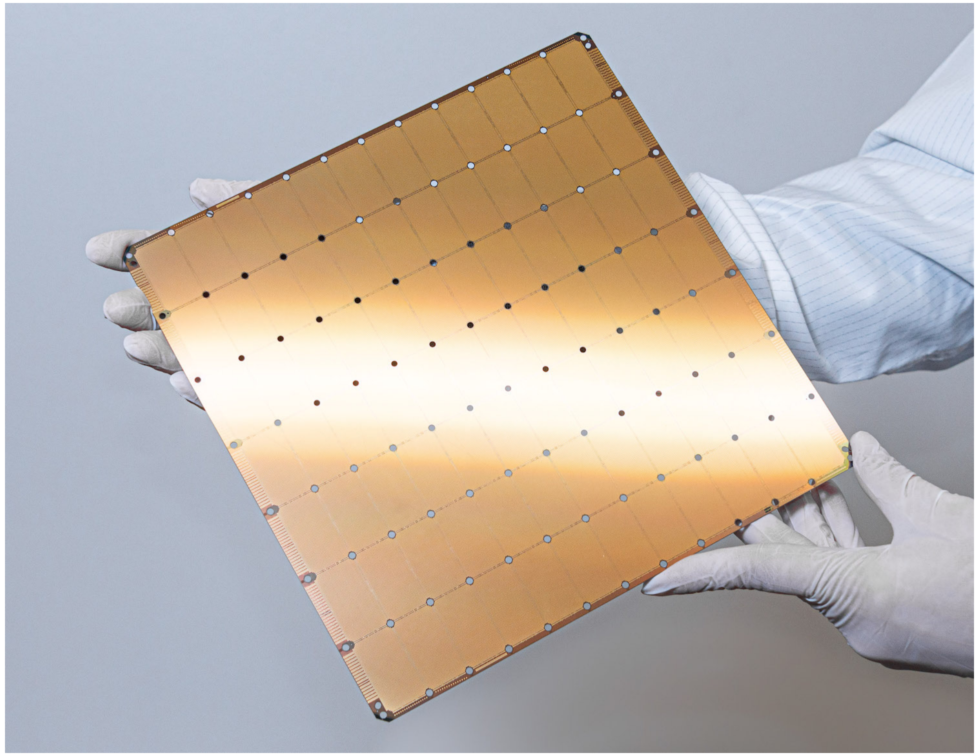
An overview of the CS-2 architecture



Cerebras WSE-2
46,225mm² Silicon
2.6 Trillion transistors

Largest GPU
826mm² Silicon
54.2 Billion transistors

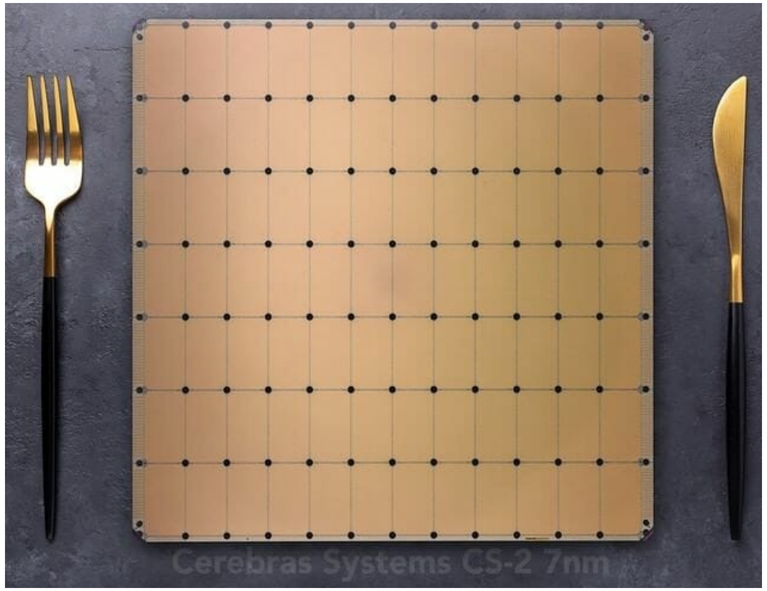


We have a (fairly big) box

- This contains all the cooling and infrastructure support
  - 15RU and draws around 23kW of power
- Uses standards-based power and network connections
  - 12x standard 100 Gigabit Ethernet links and converts standard TCP-IP traffic into Cerebras protocol at full line rate

- This all serves the Cerebras Wafer Scale Engine (WSE)
  - A wafer-parallel compute accelerator, containing hundreds of thousands of independent processing elements (PEs)
  - The WSE is the reason for the tutorial!

Cerebras Wafer Scale Engine (WSE)

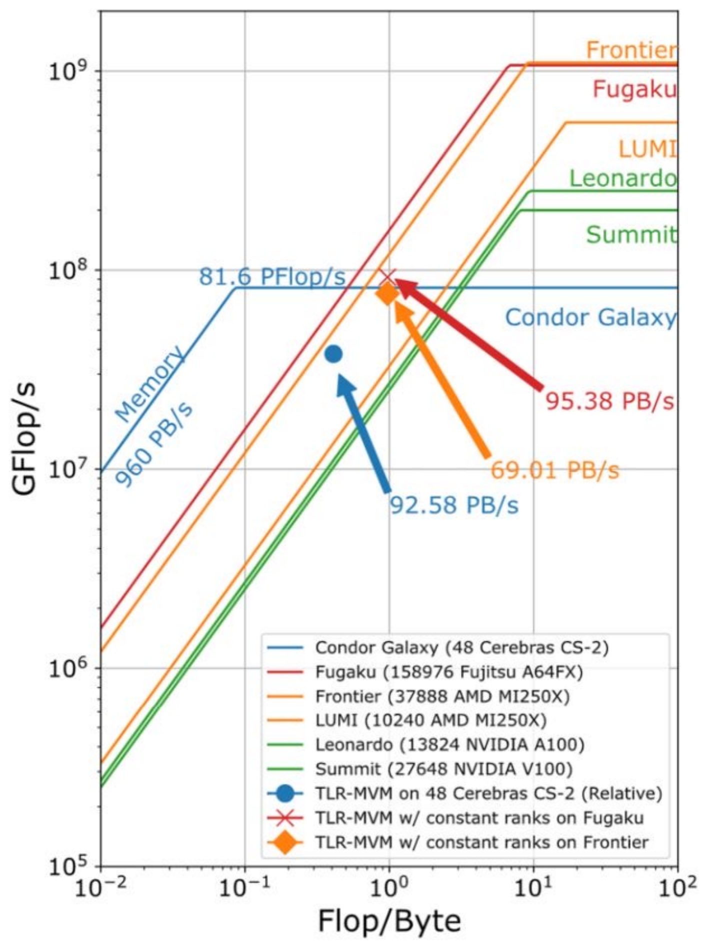


- Physically it is about as big as a dinner plate and built upon a 7nm process technology
- The current generation CS-2 that we will use today contains:
  - Approximately 850,000 cores
    - Each core is individually programmable
  - 40GB on-chip SRAM memory
  - A total of 20PB/s aggregate memory bandwidth
- The flexibility of the individual, independent cores and the large amount of memory means that, depending on the workload, the CS-2 is capable of delivering the performance of many GPUs
  - Although of course this depends heavily on you programming it effectively, which is the topic of the tutorial today!

What types of application suit the WSE?

- Scaling poorly across multiple nodes (e.g. FFTs, particle simulation)
  - The WSE has a fabric that is high bandwidth and low-latency, allowing for excellent parallel efficiency for non-linear and highly communicative codes
  - The CS-2 system has 850k cores and can fit problems on an individual chip that take tens to hundreds of traditional small compute nodes.
- Application is constrained by data access
  - The WSE has 40 GB of SRAM uniformly distributed across the wafer that is 1 cycle away from the processing element
    - Speeds up memory access by orders of magnitude
  - The CS-2 system is capable of 1.2 Tb/s bandwidth onto the chip

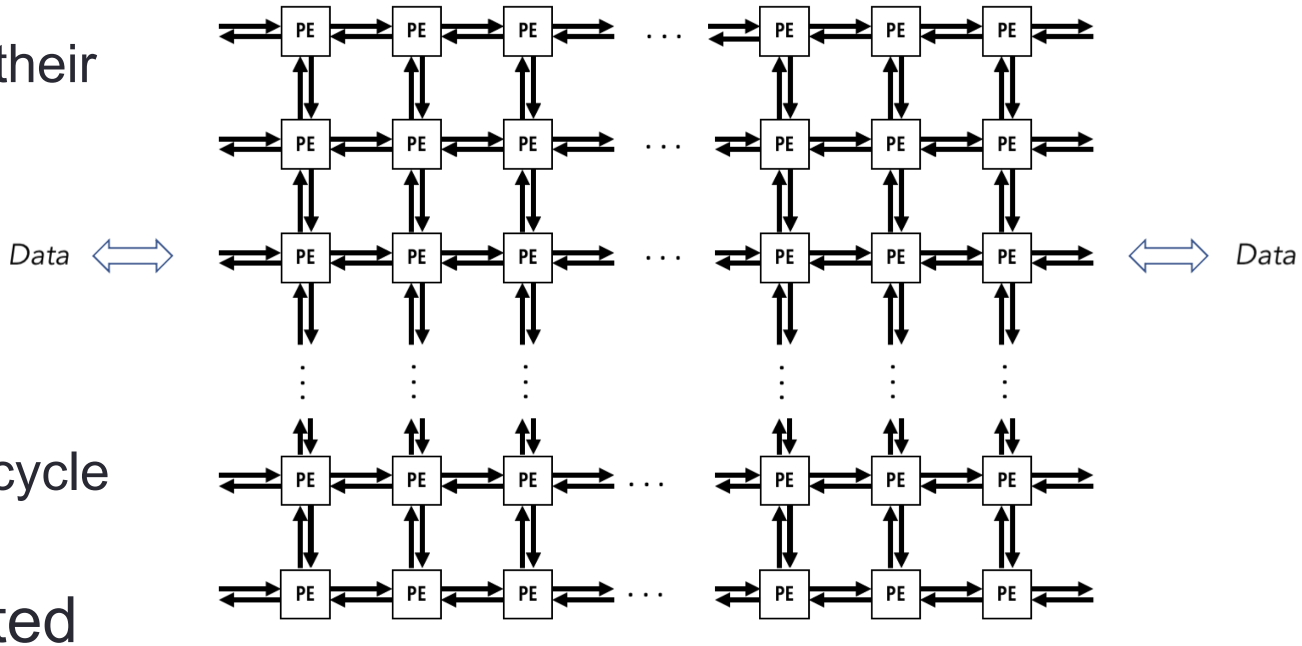
Cerebras CS-2 achieves real memory bandwidth performance that rivals best-case performance on world-leading supercomputers



https://sc23.supercomputing.org/2023/08/a-look-at-the-2023-gordon-bell-prize-finalists/

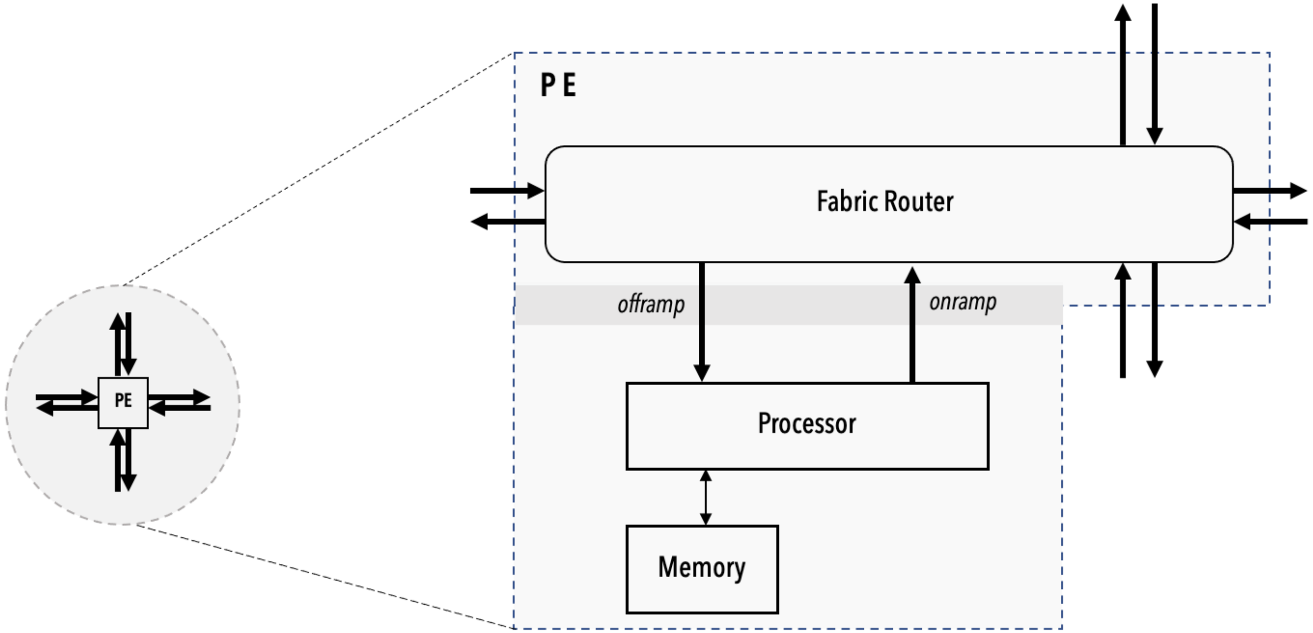
WSE conceptual high-level view

- Very many individual Processing Elements (PEs)
    - These run independent of each other (e.g. their own program counter)
- PEs are connected by 2D rectangular mesh across the chip
    - 32-bit messages (called wavelets) can be communicated with neighbours in a single cycle
- The 40GB of WSE memory is distributed amongst the PEs
    - Each PE has its own private chunk of memory



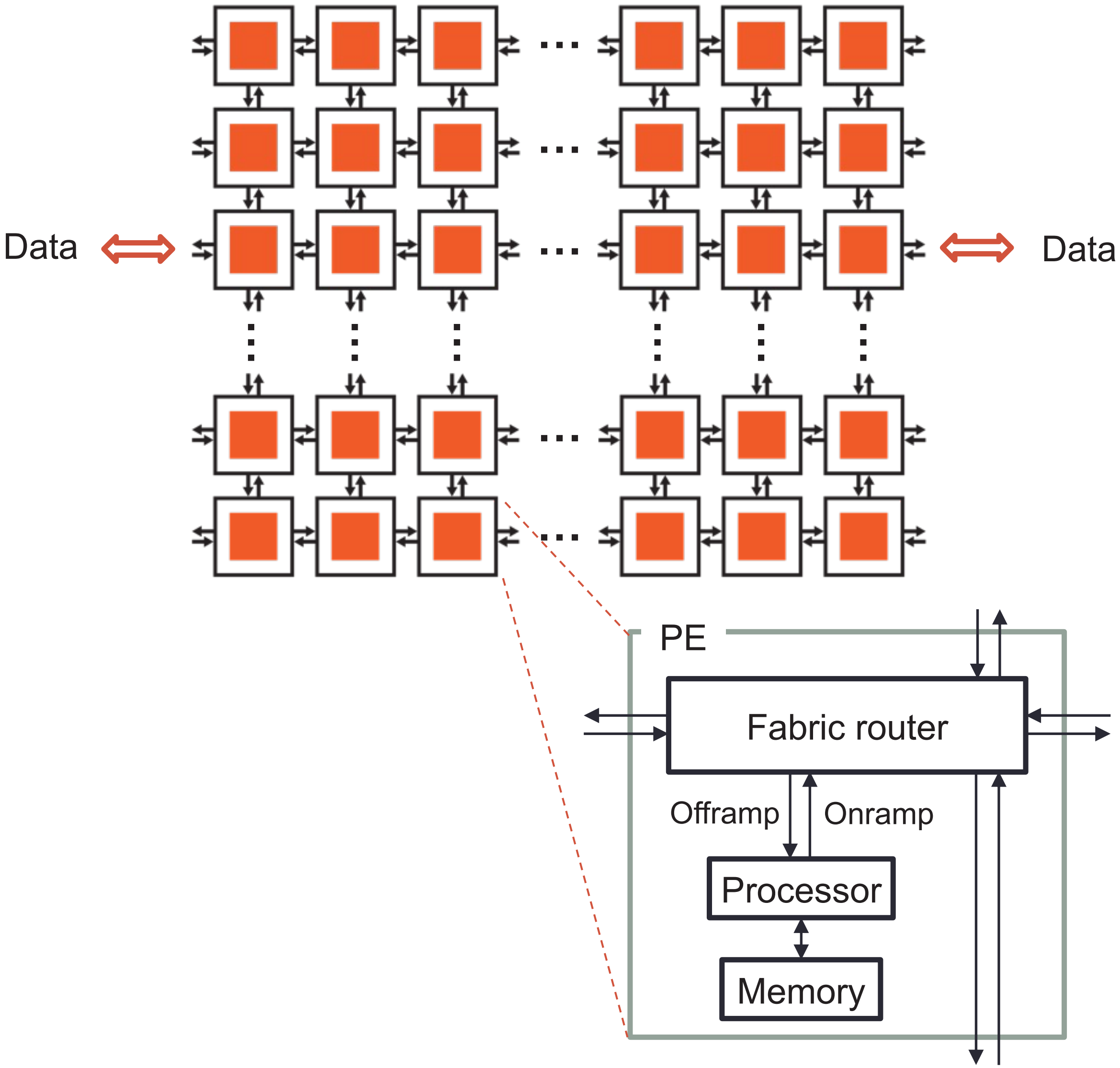
Within each Processing Element (PE)

- We have the processor itself
  - Commonly referred to as the Compute Engine (CE)
  - Independent and private from any other
- A router
  - Connected with bidirectional links to own CE and router of four neighbours
  - Link to own CE is called *the RAMP* and neighbours are referred to by north, south, east and west
  - This is the only way in which PEs can communicate
- Local (private) memory
  - All data and code for the PE is stored in this memory
  - 48KB per PE



What is supported by the CE

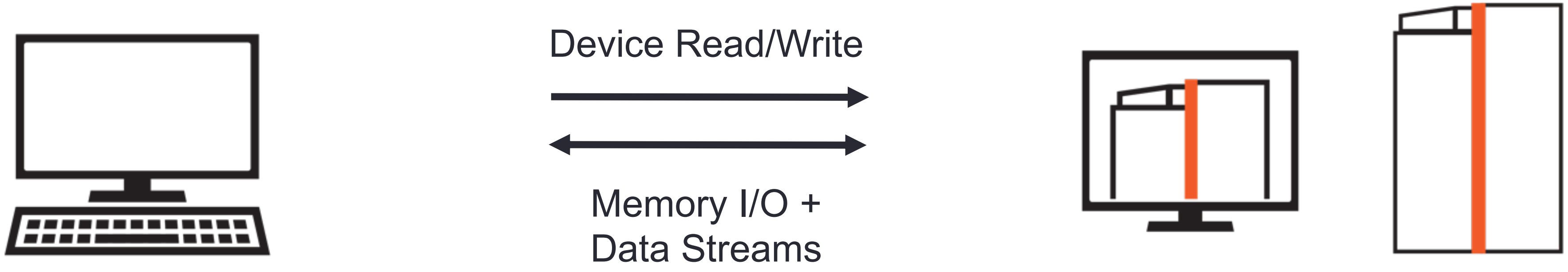
- 16- and 32-bit native FP and integer data types are supported by the CE
- Follows a dataflow execution model
  - Tasks are either triggered or activated
  - Independent programs specified for regions of PEs
- Control flow is straightforward to reason about
  - Tasks are non-preemptive
  - Instruction to activate another task enable state-machine behavior



From a programmer’s perspective

- **Host CPU(s): Python**
- Loads program onto simulator or CS-2 system
- Streams in/out data from one or more workers
- Reads/writes device memory

- **Device: Cerebras Software Language (CSL)**
- Target software simulator or CS-2
- CSL programs run on groups of cores on the WSE, specified by programmer
- Executes dataflow programs



Device Read/Write

Memory I/O +
Data Streams

Programming the WSE

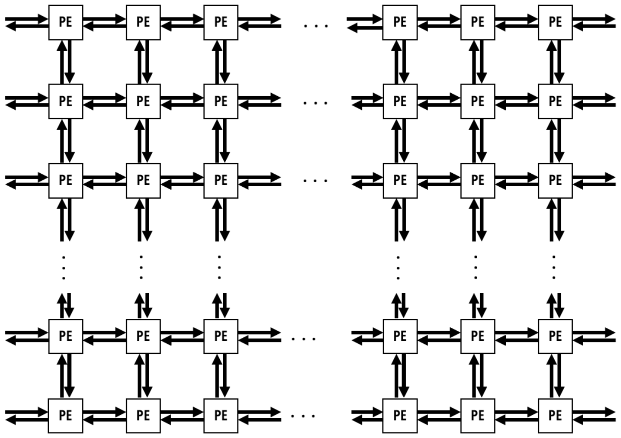
- Cerebras Software Language (CSL) is used to actually program the device
  - As we will see, this provides both a series of abstractions for high level programming with the ability to get down into the low level details (builtins) if desired

- CSL program consists of tasks and functions
  - Functions can be called by the host or another function on the device
  - Tasks are started by the hardware, run until completion, and then at that point the hardware chooses another task to run
    - Can only be *activated,* can not be called by other tasks or functions. Also don’t return a value
    - Each task has an associated ID that is used for tracking and management

```
var result: f32 = 0.0;

task main_task() void {
    result = 5.0;
}
```

Communication between PEs



- Remember, a wavelet is a 32-bit message communicated with a neighbour in a single cycle
- Each physical channel has 24 virtual communication channels known as ***colors*** that can be used for passing wavelets

- Each wavelet has associated with it a 5-bit identifier which defines which channel it is communicated on
  - Determines the wavelet's routing through the fabric and its consumption
  - This is a bit like a tag in MPI point-to-point communications, and similarly many messages on one color does not block messages with a different color using the same physical link

- Wavelets are consumed by tasks on a PE where a task is registered to execute when a wavelet arrives with a specific color

Three types of task

- Data tasks
  - Which activate upon the arrival of a wavelet with a matching color, these are used to consume messages that are communicated between PEs
  - Wavelet Triggered Task (WTT)

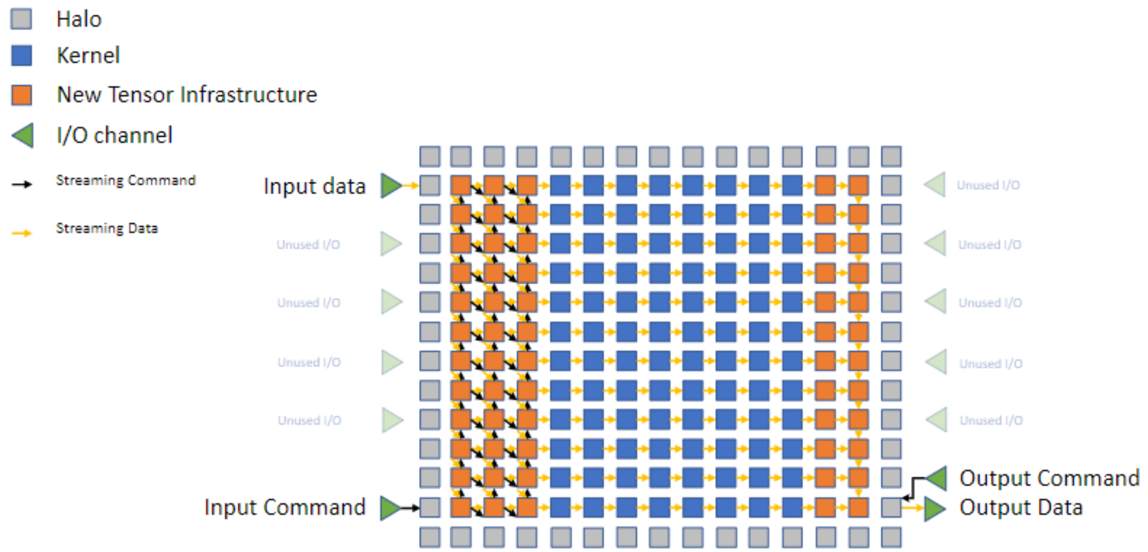
- Local tasks
  - Explicitly activated by other tasks or functions running on the PE

- Control tasks
  - Like a data task (also wavelet triggered)
  - But not routable, useful for e.g. dynamically reconfiguring routes

Host runtime

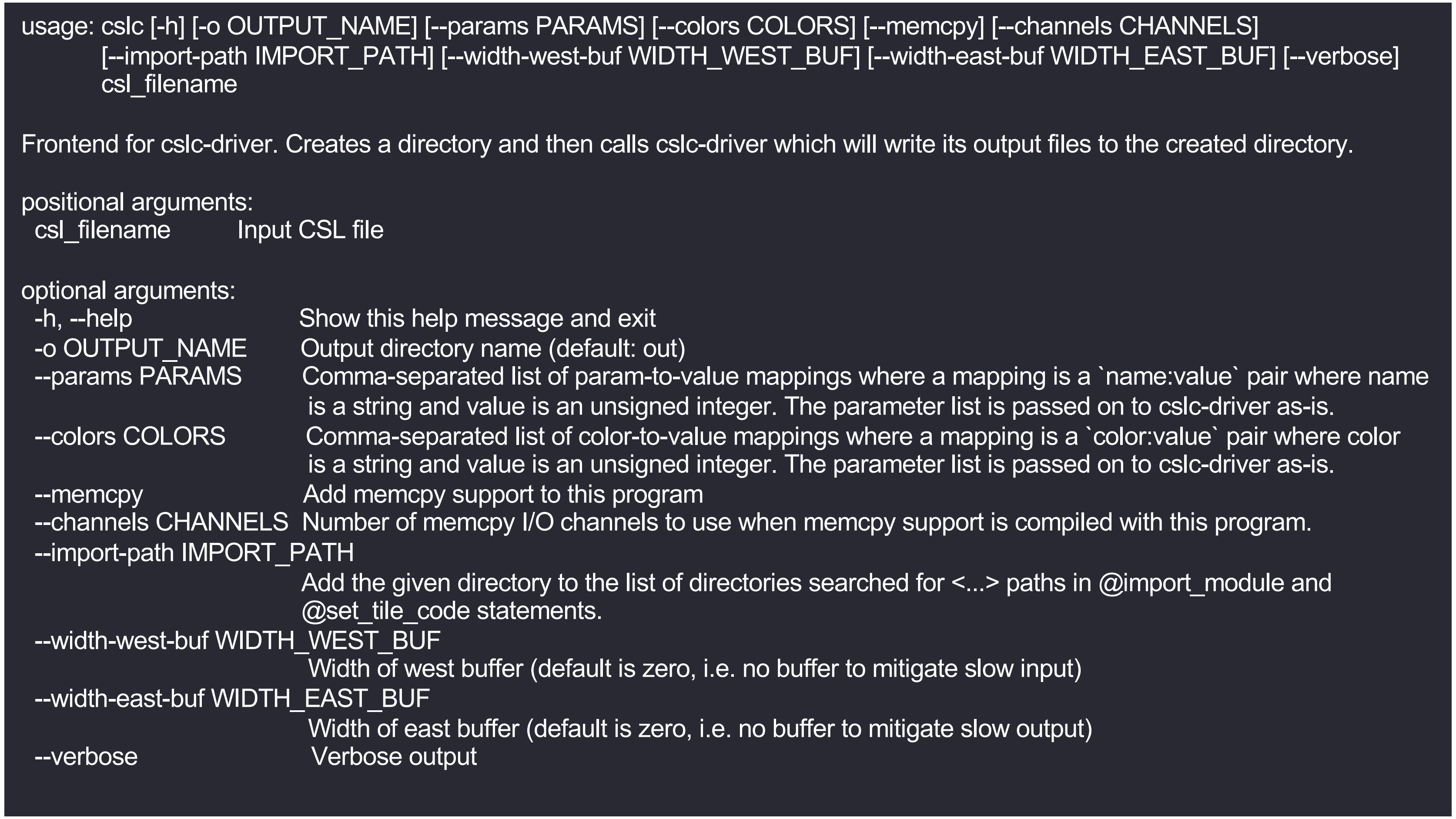
- A host-side runtime known as *SdkRuntime* is provided
- *Memcpy library:*
  - Transfer data on and off the WSE from host program, and launch device functions
  - The host and WSE network interfaces finally route the data into your kernel
    - (last step is implemented on the WSE itself to connect the I/O channel entry-points, which are in fixed locations at the edges of the WSE)

  - Uses additional PEs around the user kernel to route data
  - In addition to a halo around the kernel, the additional support PEs consume three columns on the West of the kernel and two columns on the East.



The CSL compiler

- Code is compiled using the *cslc* command
- Generates .elf files which are the executable that will be loaded onto the CS-2
- Will play with this in a few minutes



```
usage: cslc [-h] [-o OUTPUT_NAME] [--params PARAMS] [--colors COLORS] [--memcpy] [--channels CHANNELS]
            [--import-path IMPORT_PATH] [--width-west-buf WIDTH_WEST_BUF] [--width-east-buf WIDTH_EAST_BUF] [--verbose]
            csl_filename

Frontend for cslc-driver. Creates a directory and then calls cslc-driver which will write its output files to the created directory.

positional arguments:
  csl_filename          Input CSL file

optional arguments:
  -h, --help            Show this help message and exit
  -o OUTPUT_NAME        Output directory name (default: out)
  --params PARAMS       Comma-separated list of param-to-value mappings where a mapping is a `name:value` pair where name
                        is a string and value is an unsigned integer. The parameter list is passed on to cslc-driver as-is.
  --colors COLORS       Comma-separated list of color-to-value mappings where a mapping is a `color:value` pair where color
                        is a string and value is an unsigned integer. The parameter list is passed on to cslc-driver as-is.
  --memcpy              Add memcpy support to this program
  --channels CHANNELS   Number of memcpy I/O channels to use when memcpy support is compiled with this program.
  --import-path IMPORT_PATH
                        Add the given directory to the list of directories searched for <...> paths in @import_module and
                        @set_tile_code statements.
  --width-west-buf WIDTH_WEST_BUF
                        Width of west buffer (default is zero, i.e. no buffer to mitigate slow input)
  --width-east-buf WIDTH_EAST_BUF
                        Width of east buffer (default is zero, i.e. no buffer to mitigate slow output)
  --verbose             Verbose output
```

Use of the simulator

- CS-2 machines are designed to be exclusive use
  - Routinely developing code directly on the CS-2 would be a big pain, due to contending with lots of other users at the same time!

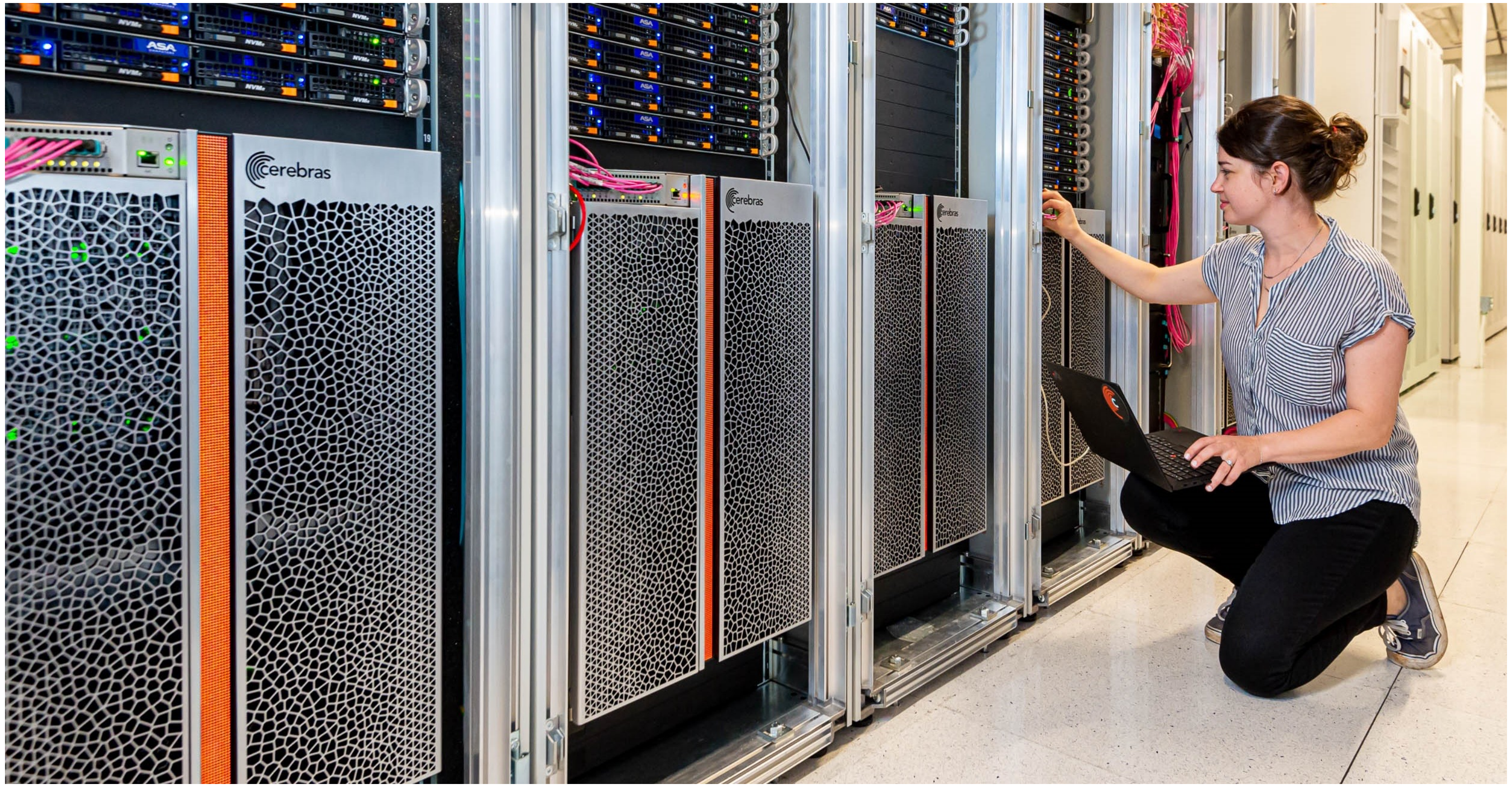
- Cerebras also provide a software simulator of the CS-2
  - Simulates execution of your program by running it on the CPU
  - This is highly accurate and should be used during development to test your code
  - But for performance measurements and production runs obviously we need the CS-2 itself

- We will use the simulator quite heavily during this tutorial
  - But part of our hands-on exercises will be to run each of these on the CS-2 hardware itself once you have done the development

Clustering CS-2 machines

- For ML applications, multiple CS-2 machines can be clustered into Wafer-Scale Cluster

- This is not yet available for SDK applications



Conclusions

- The CS-2 is a powerful architecture with significant raw compute and distributed memory
- We have explored the general way in which the WSE is organised and the key concepts and terminology
- Now we are going to move onto focussing on how to program the machine
  - You will see how these concepts are used practically in codes