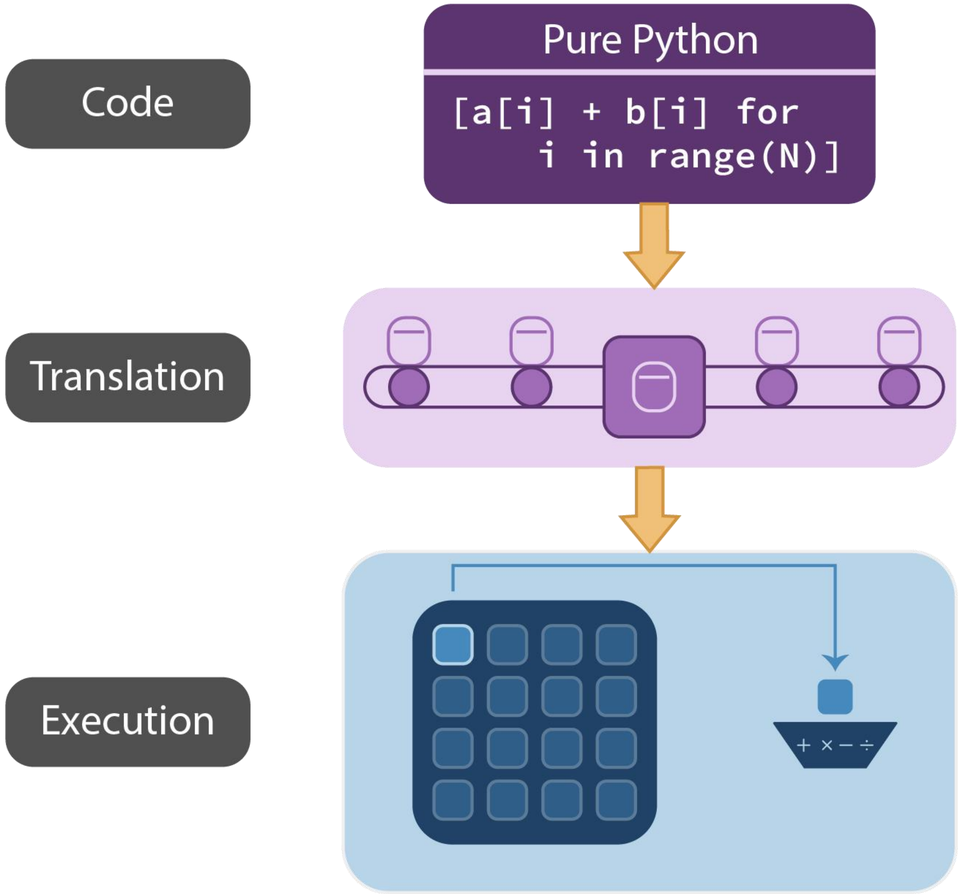
A Practical Guide to GPU Hardware for Python Programmers

PyCon 2026

Vyas Ramasubramani, Senior Software Engineer

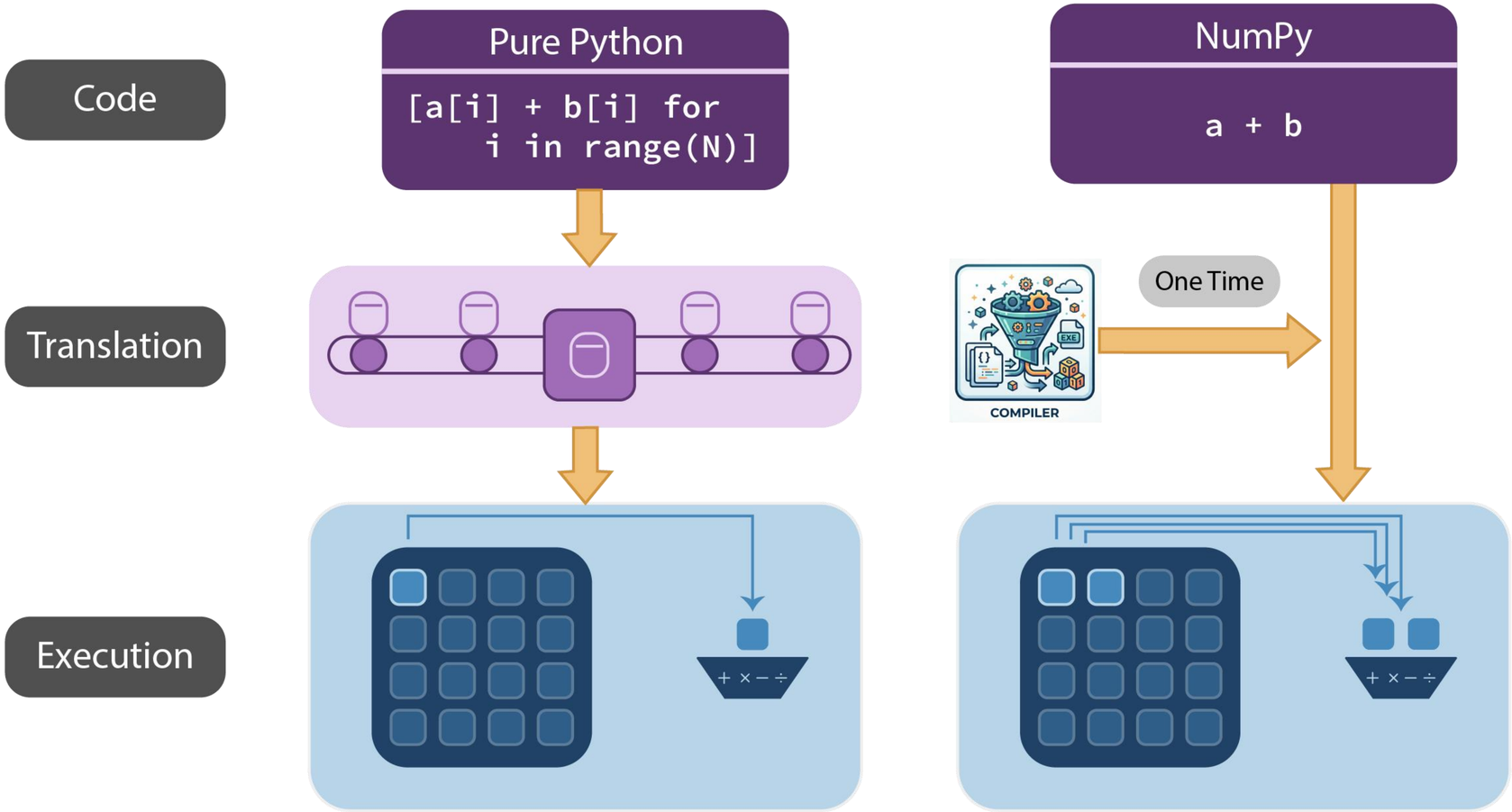
Where Hardware Matters

Most crucial hardware features are present in familiar places



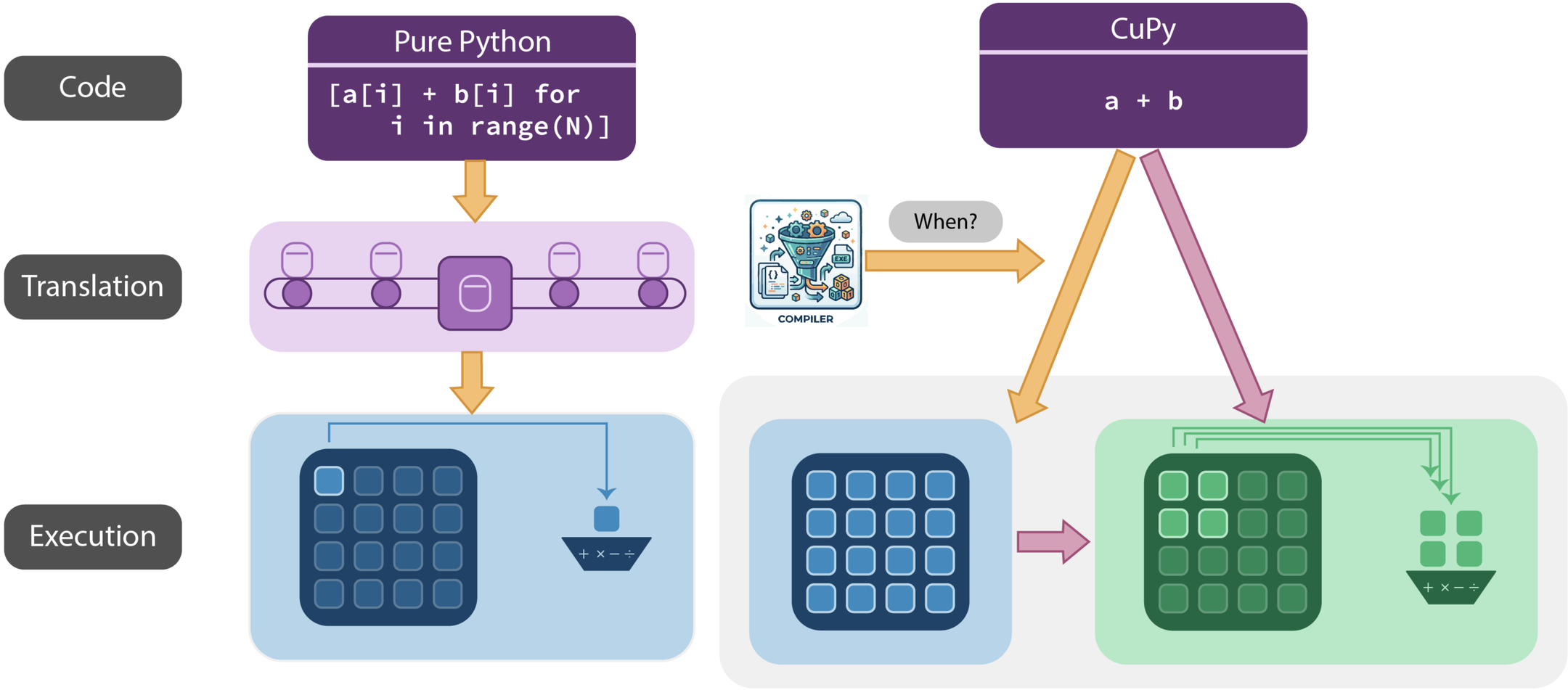
Where Hardware Matters

The transition from pure Python to NumPy exposes critical hardware and software interactions



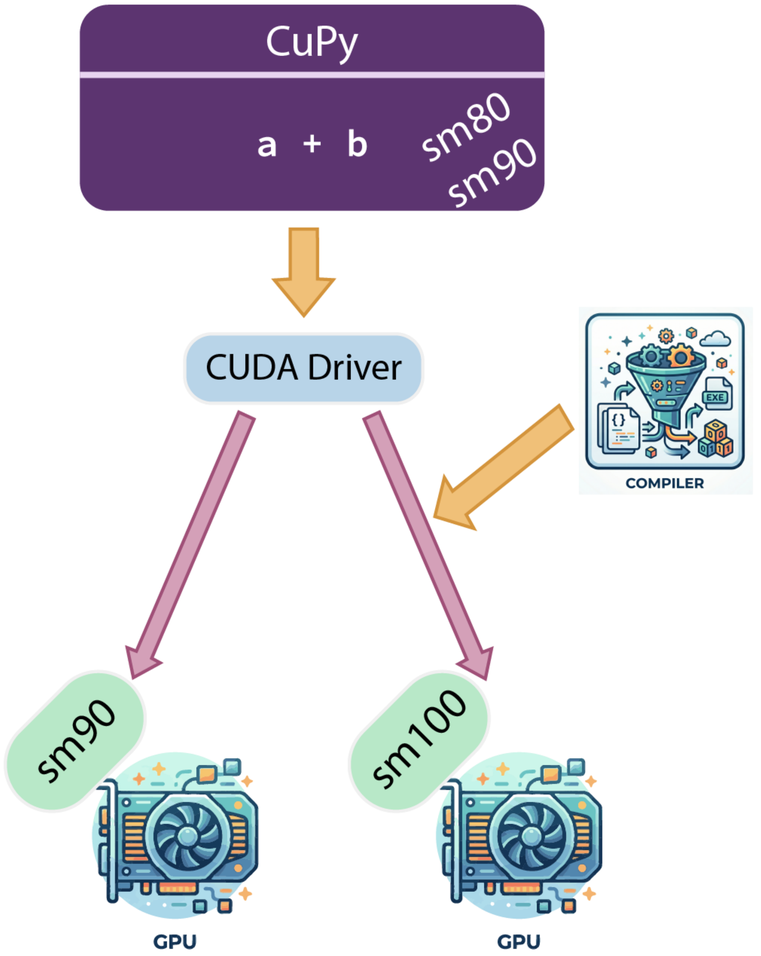
Where Hardware Matters

Going from NumPy to CuPy adds all the relevant critical complexity



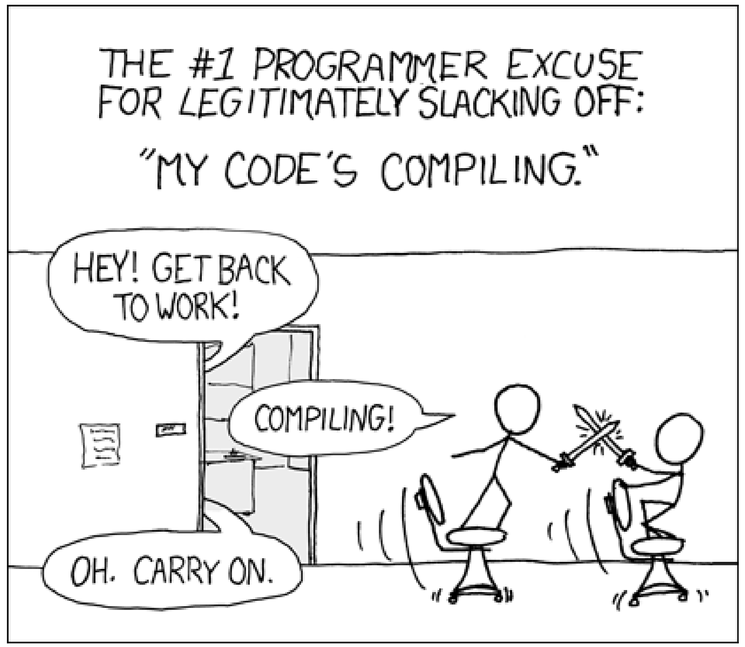
Compilation Overhead

Just-in-time (JIT) compilation is central to all GPU execution pipelines

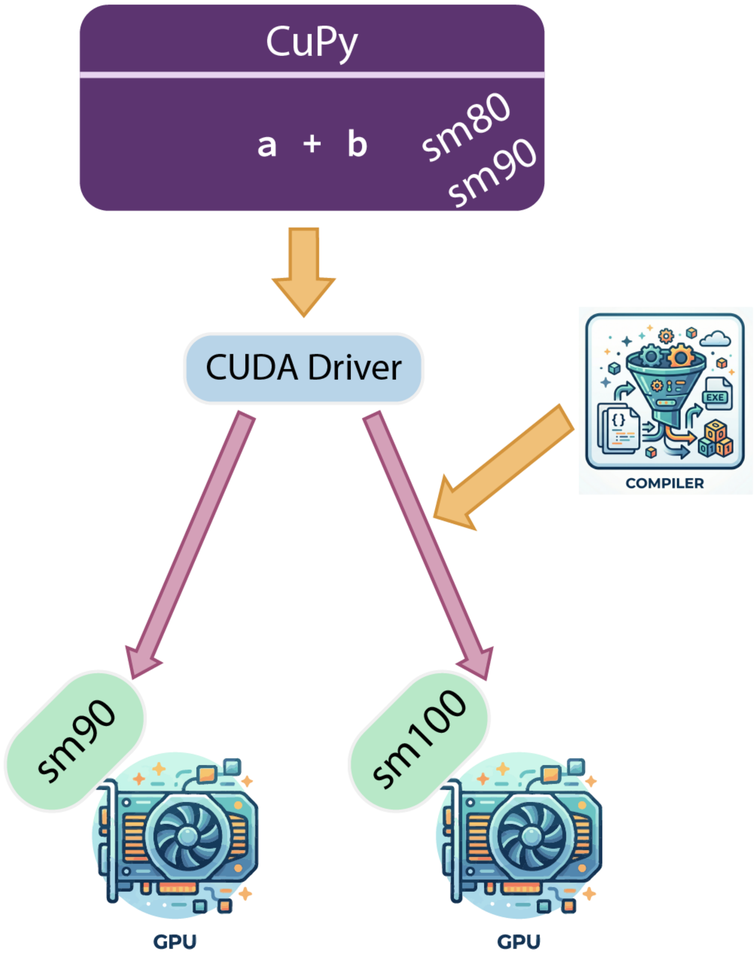


Compilation Overhead

Just-in-time (JIT) compilation is central to all GPU execution pipelines

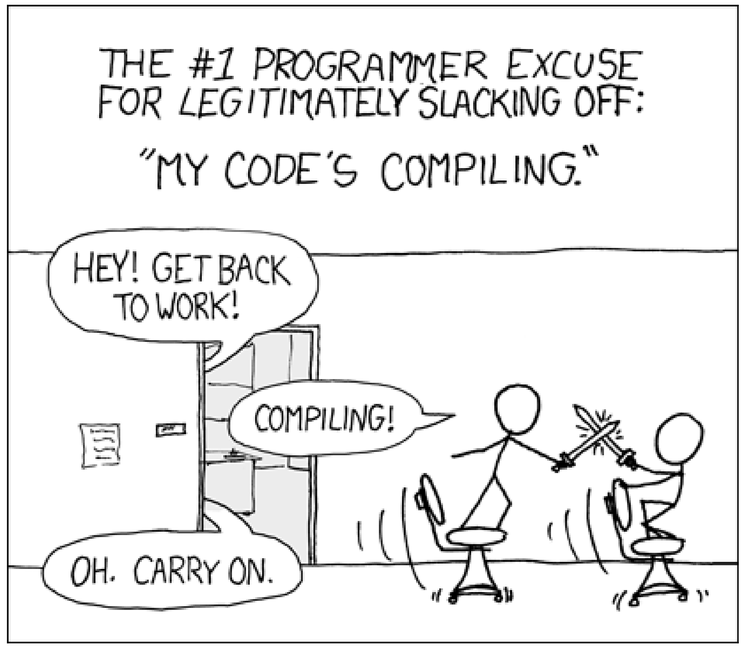


https://xkcd.com/303/

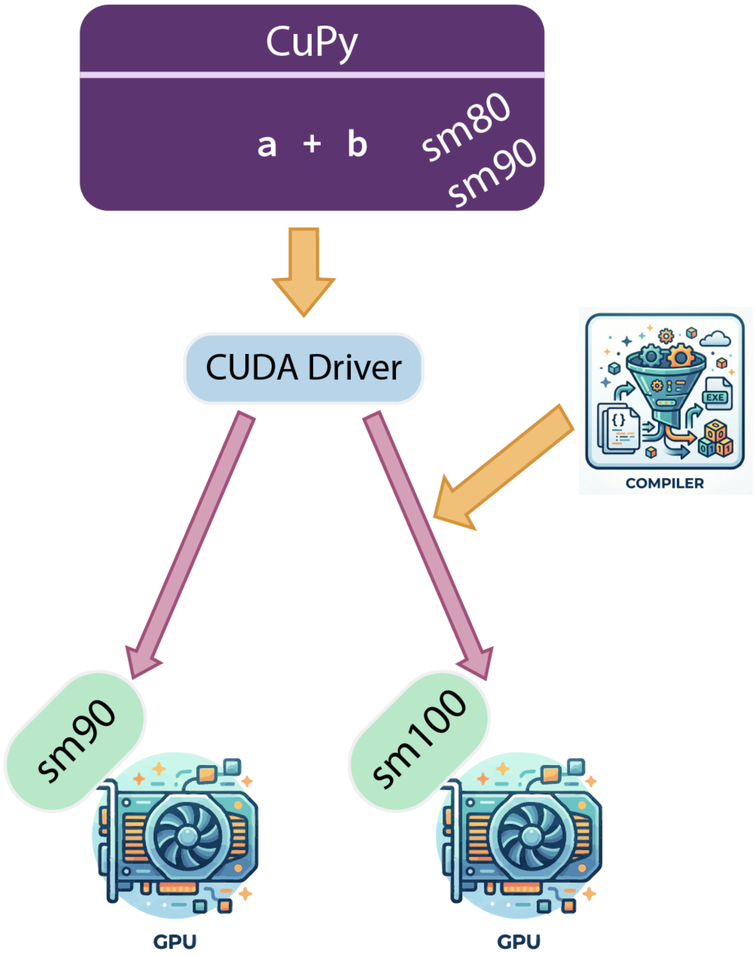


Compilation Overhead

Just-in-time (JIT) compilation is central to all GPU execution pipelines



https://xkcd.com/303/

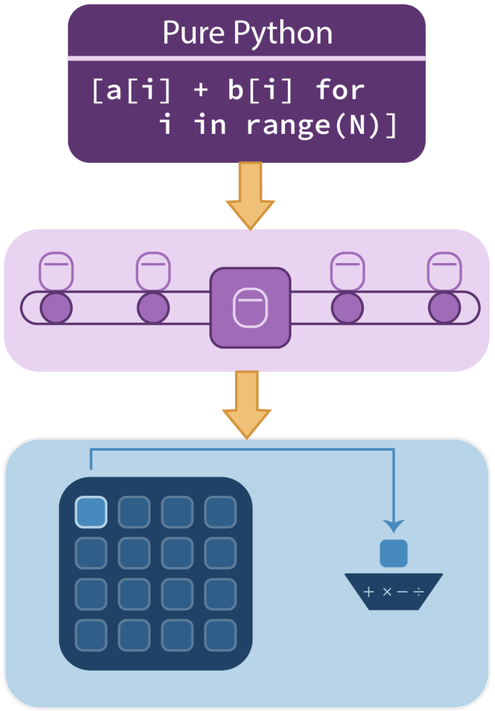


Persist JIT caches

- `CUDA_CACHE_PATH`
- `JAX_COMPILATION_CACHE_DIR`

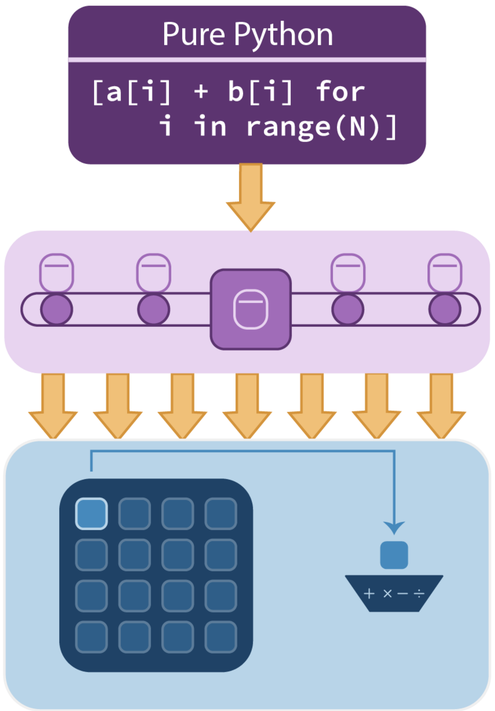
Function Call Overhead

Python functions are expensive



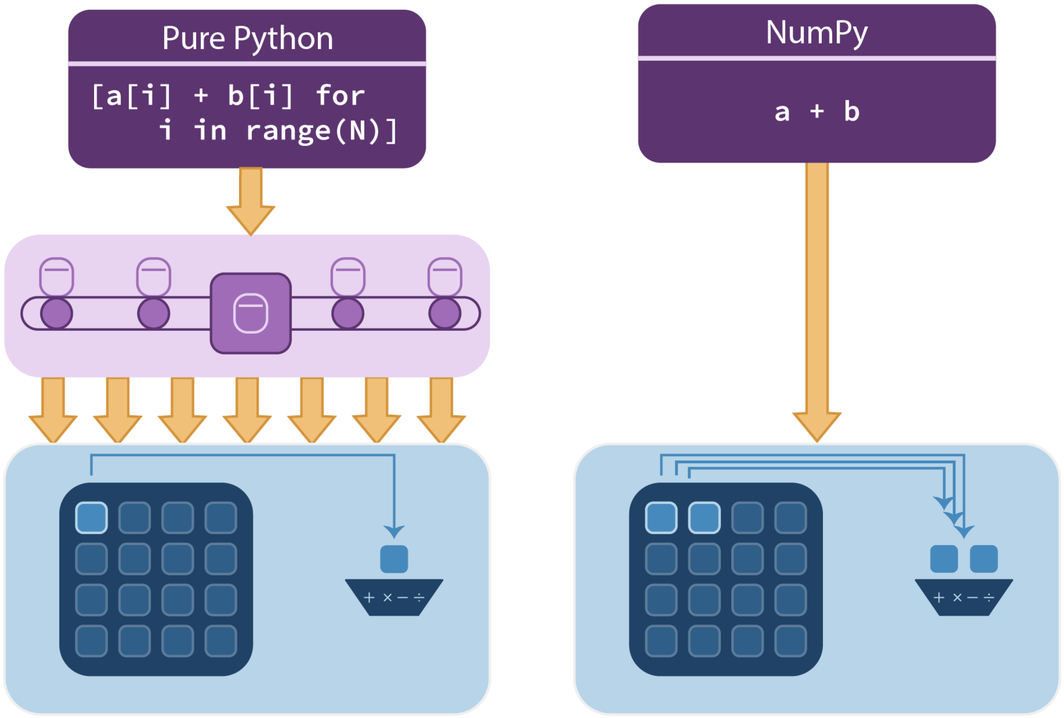
Function Call Overhead

Looping in Python means calling one Python function at a time



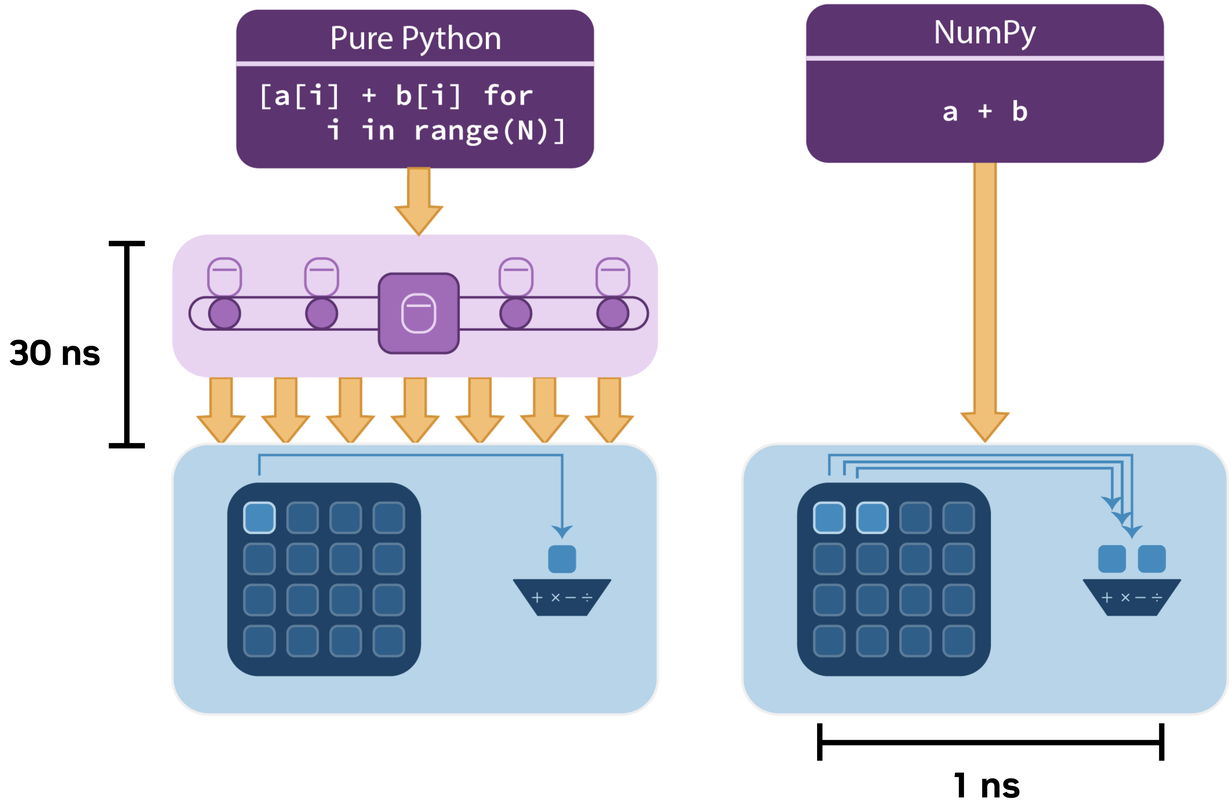
Function Call Overhead

One of the benefits of NumPy is that all these calls happen in C



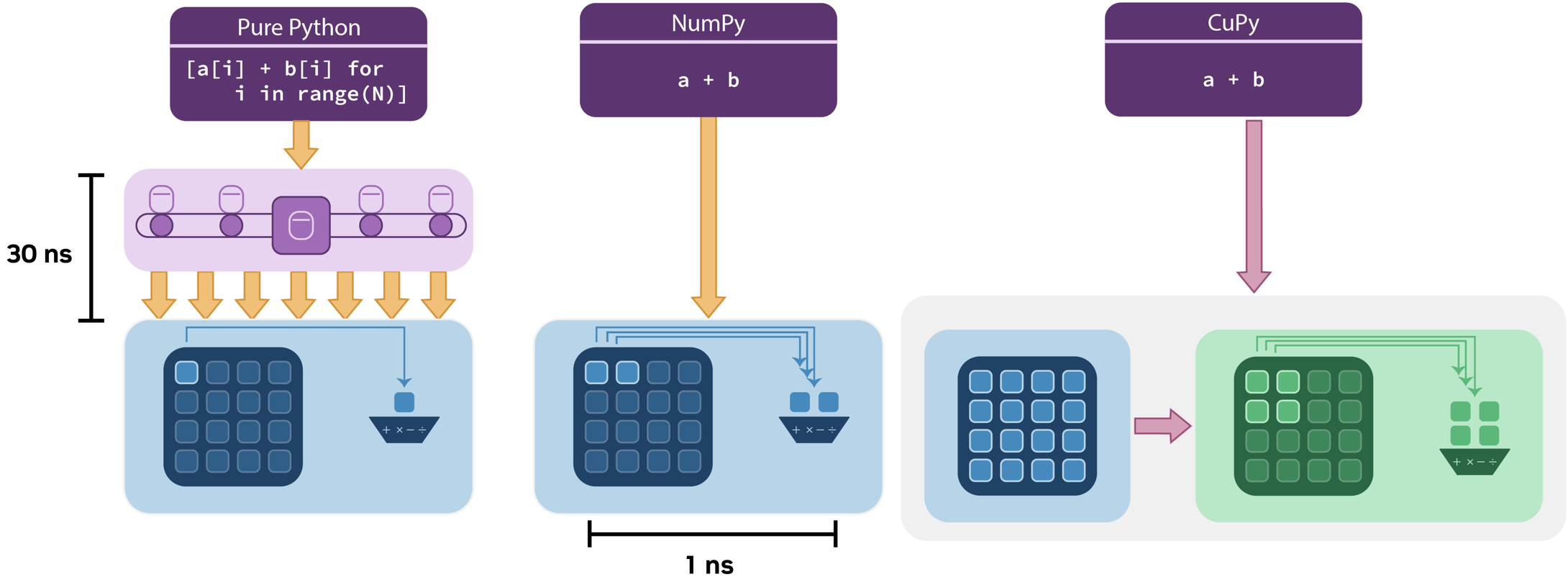
Function Call Overhead

One of the benefits of NumPy is that all these calls happen in C



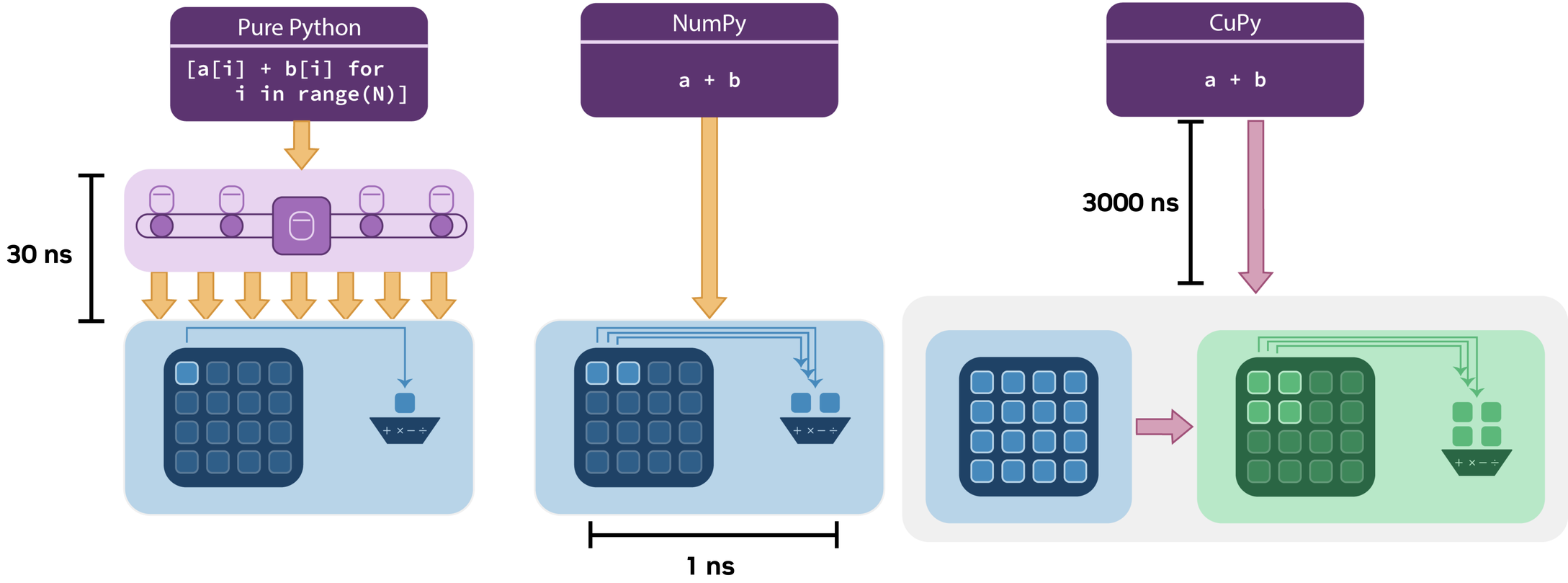
Function Call Overhead

With any GPU library, function calls (kernel launches) must cross hardware boundaries



Function Call Overhead

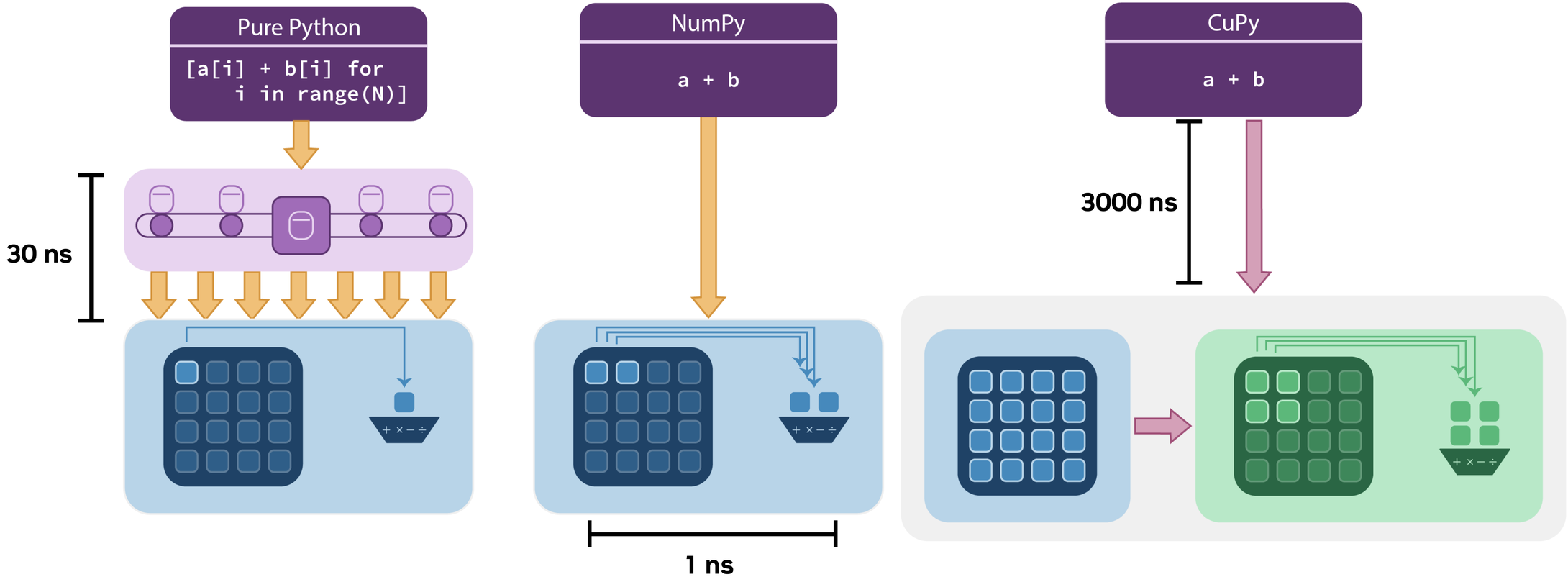
With any GPU library, function calls (kernel launches) must cross hardware boundaries



Function Call Overhead

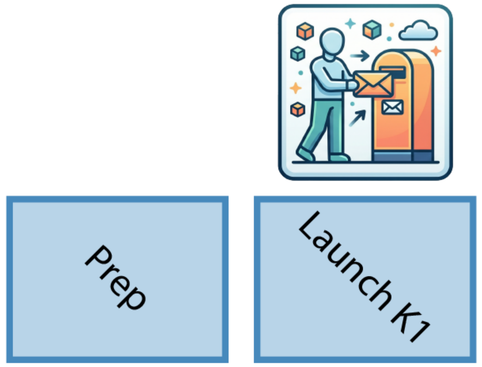
With any GPU library, function calls (kernel launches) must cross hardware boundaries

- Fuse kernels (`cupy.fuse, torch.compile`)
- Hide kernel latency



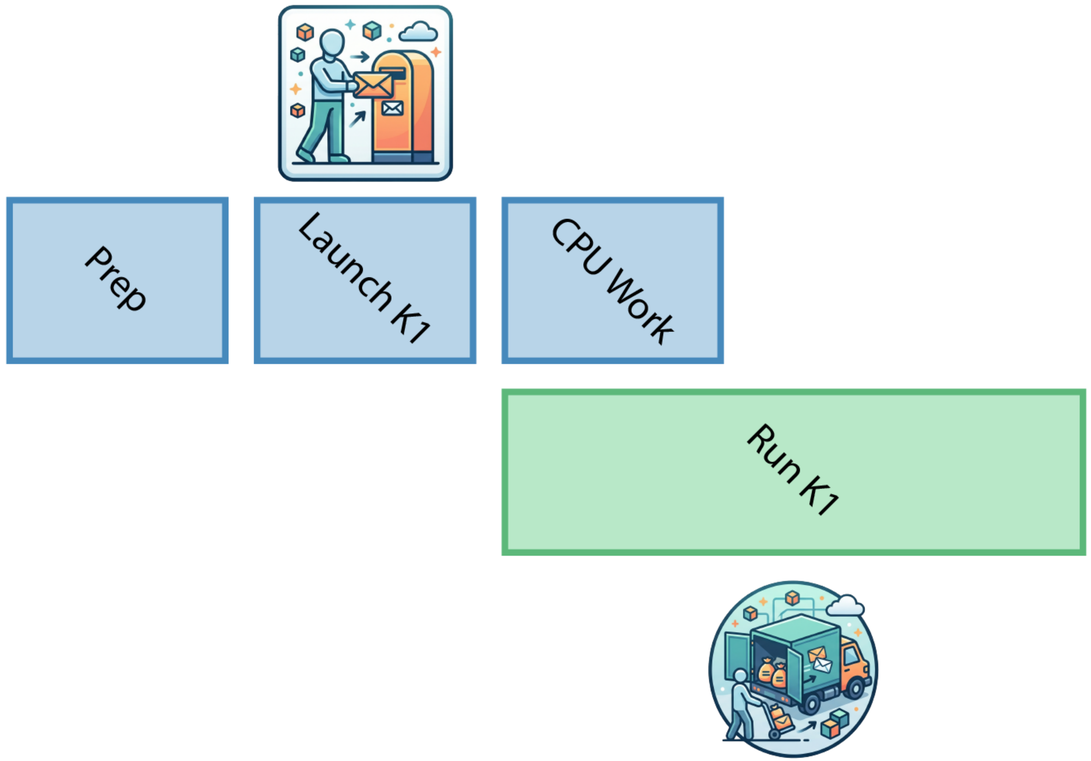
GPU Execution is Asynchronous

Once a kernel is launched, the CPU can keep working



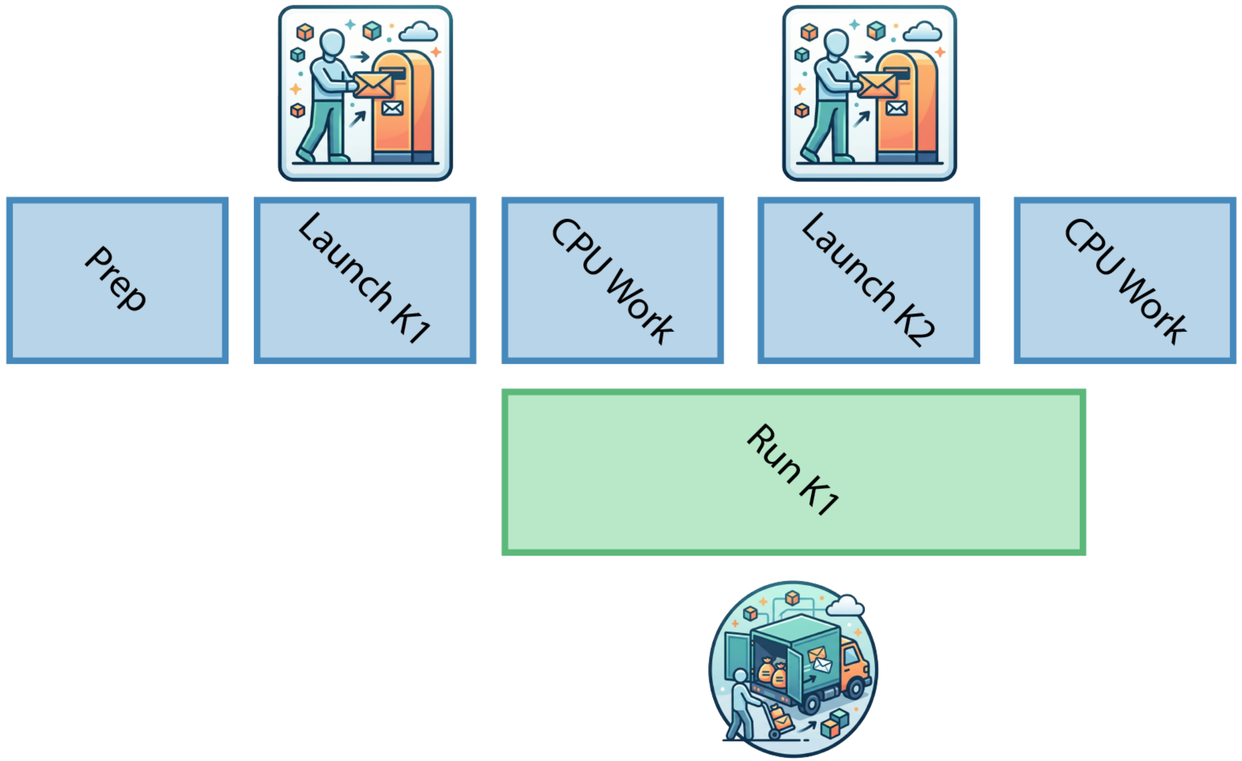
GPU Execution is Asynchronous

Once a kernel is launched, the CPU can keep working



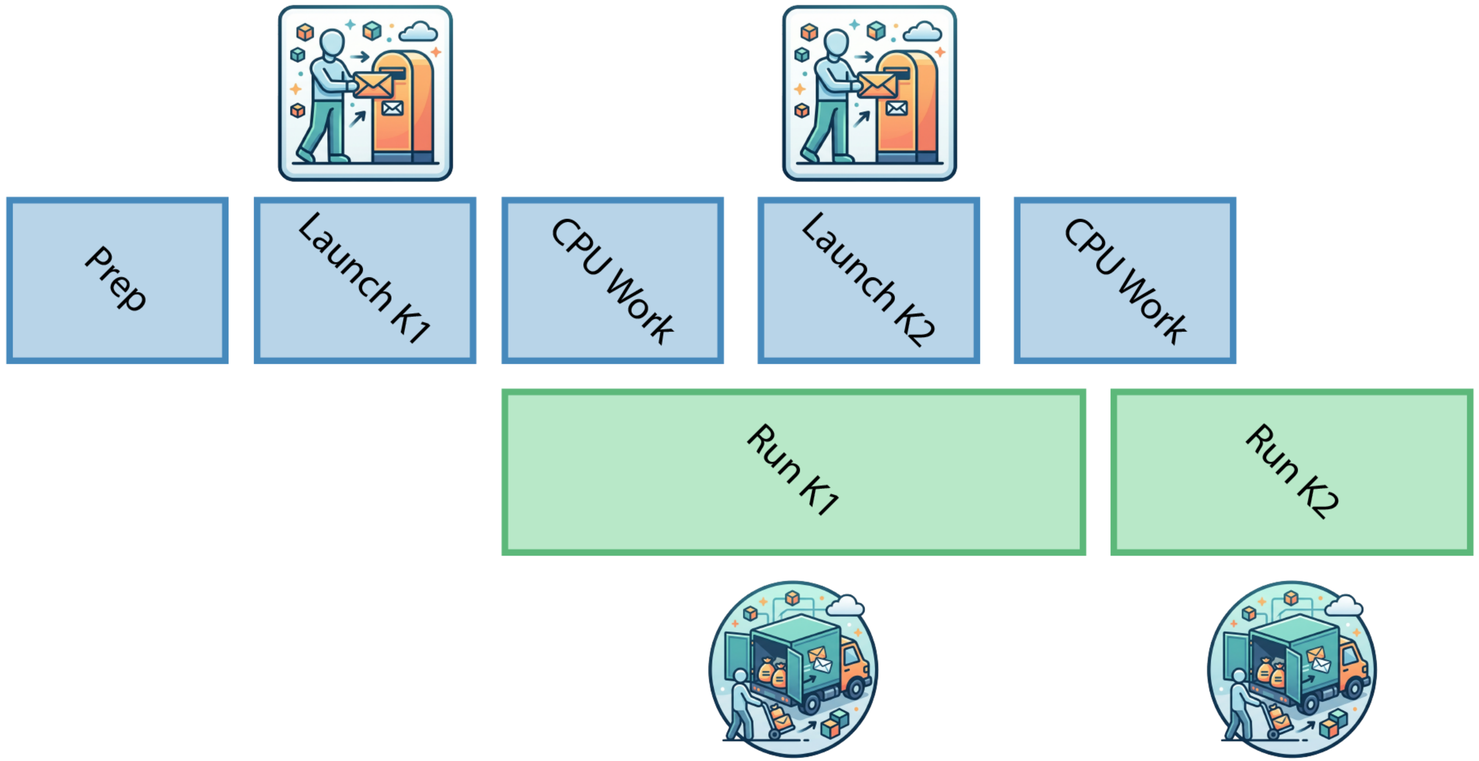
GPU Execution is Asynchronous

Once a kernel is launched, the CPU can keep working



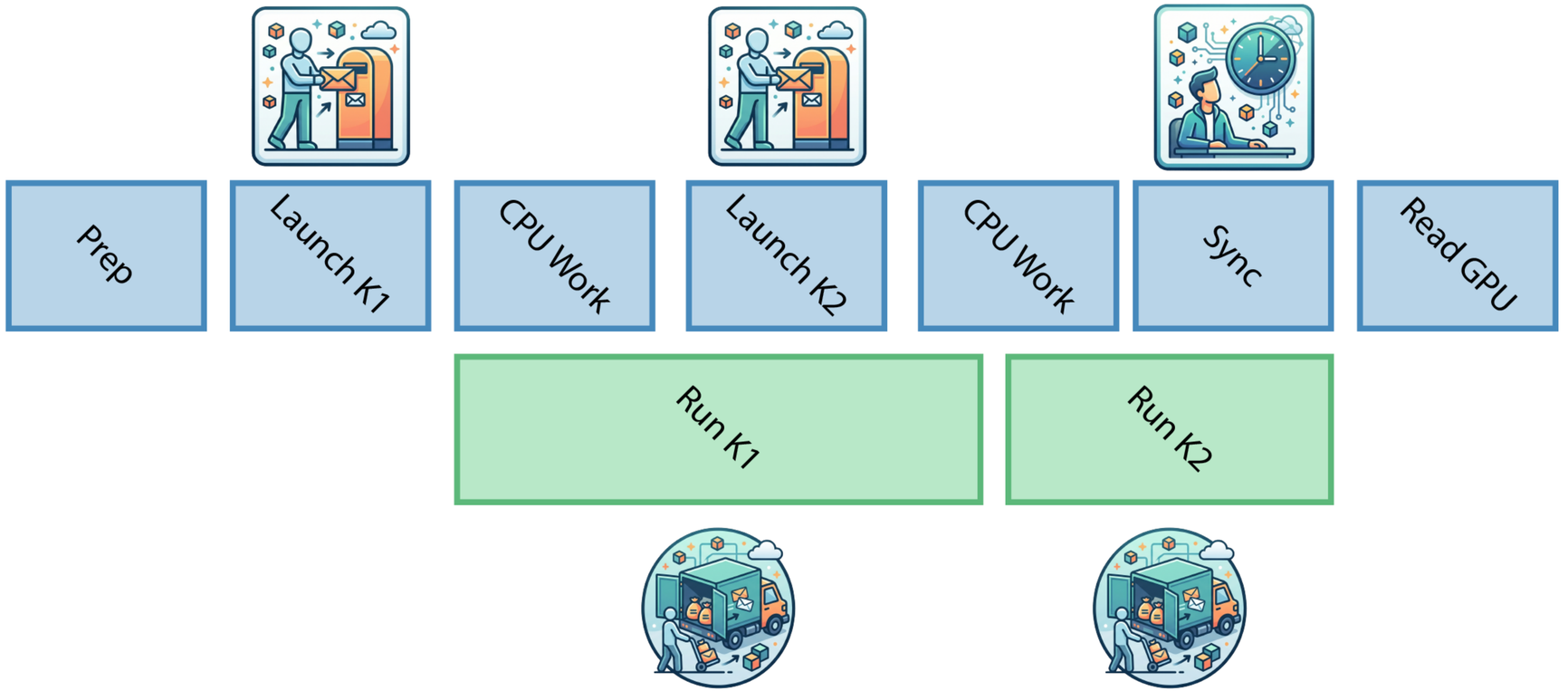
GPU Execution is Asynchronous

Once a kernel is launched, the CPU can keep working



GPU Execution is Asynchronous

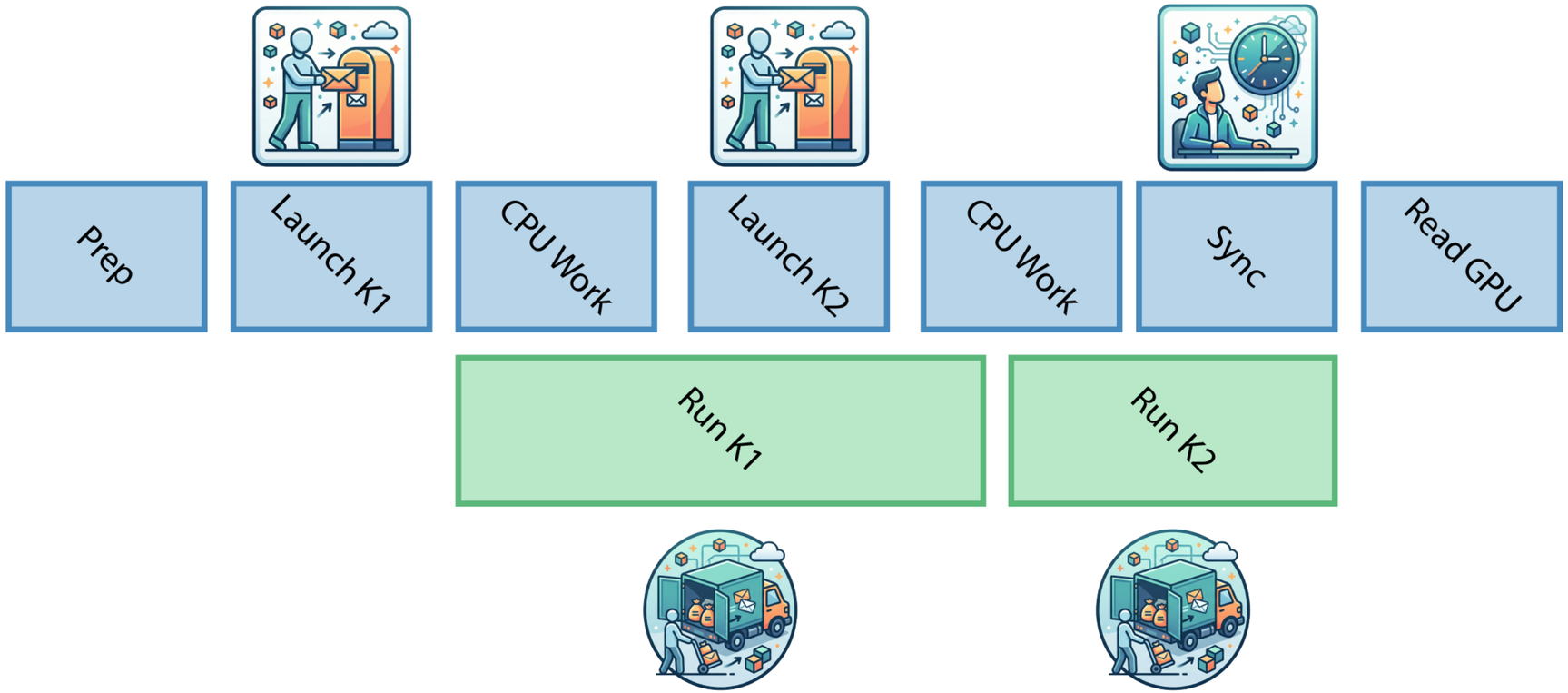
Once a kernel is launched, the CPU can keep working



GPU Execution is Asynchronous

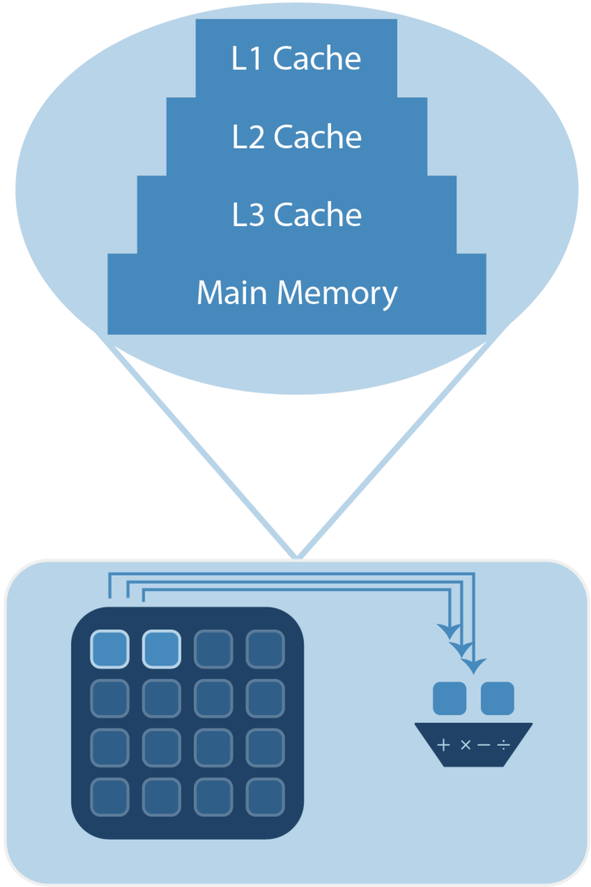
Once a kernel is launched, the CPU can keep working

- Launch kernels while GPU is working
- Keep both CPU and GPU always busy



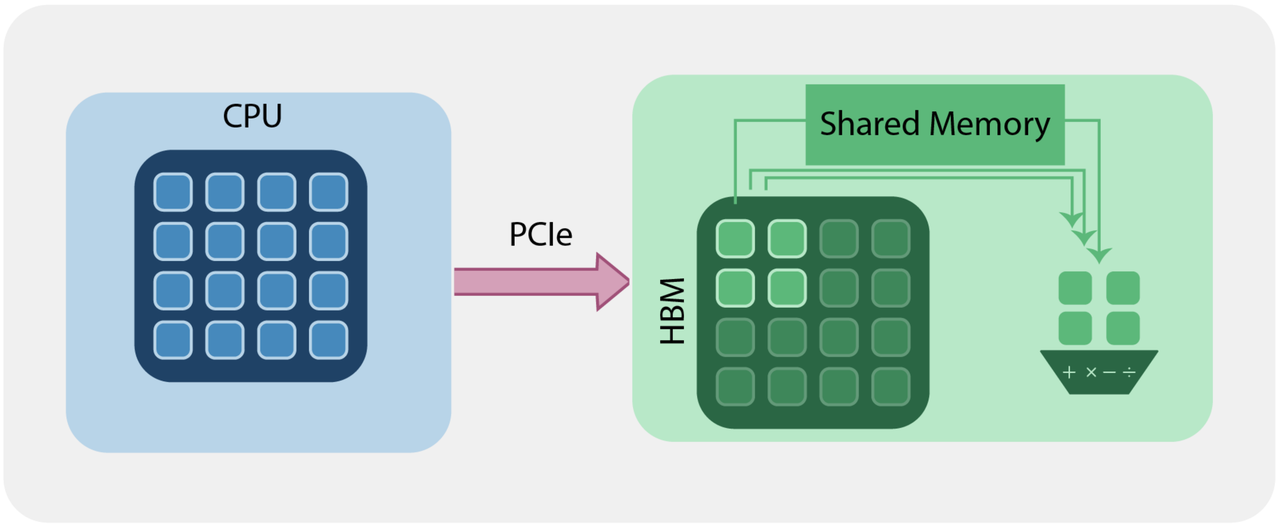
Memory Hierarchies and Migration

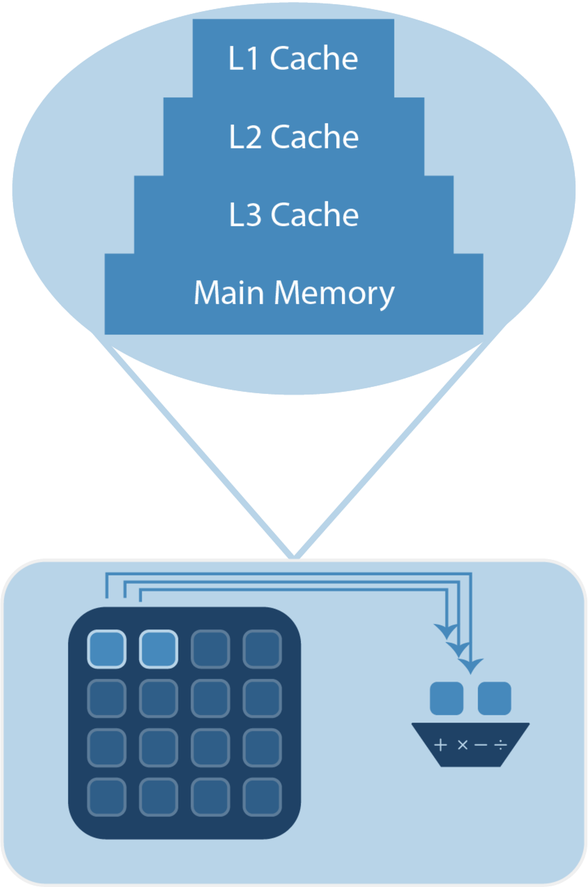
CPUs have deep memory hierarchies that are almost entirely automatically managed



Memory Hierarchies and Migration

GPU memory transfers must be managed explicitly

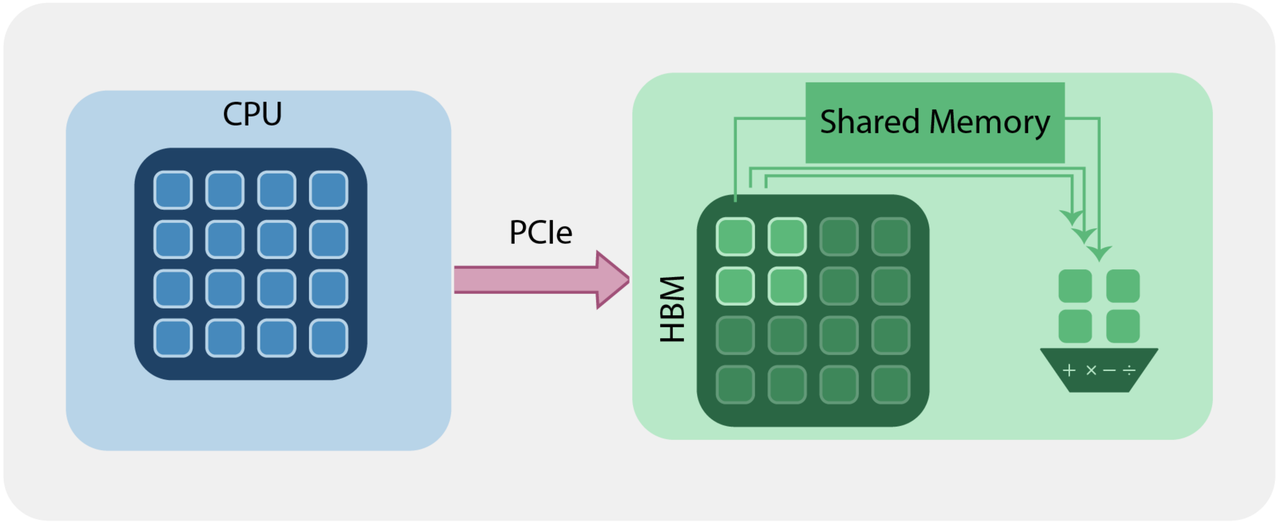


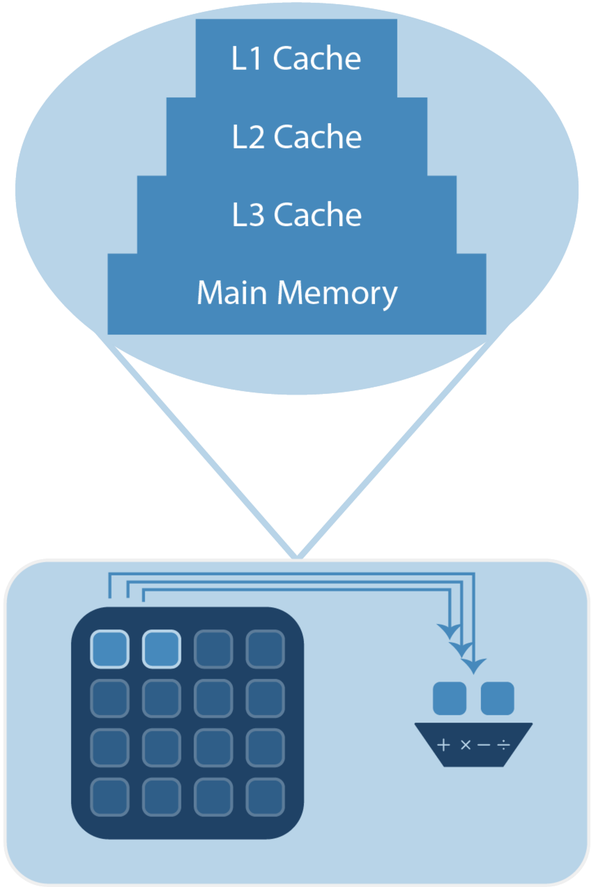


Memory Hierarchies and Migration

GPU memory transfers must be managed explicitly

- Stage memory transfers, kernel launches, and other CPU work
- For kernel authors: use special memory spaces like shared memory





Latency vs. Throughput

CPUs and GPUs take different approaches to managing latency

Low Latency

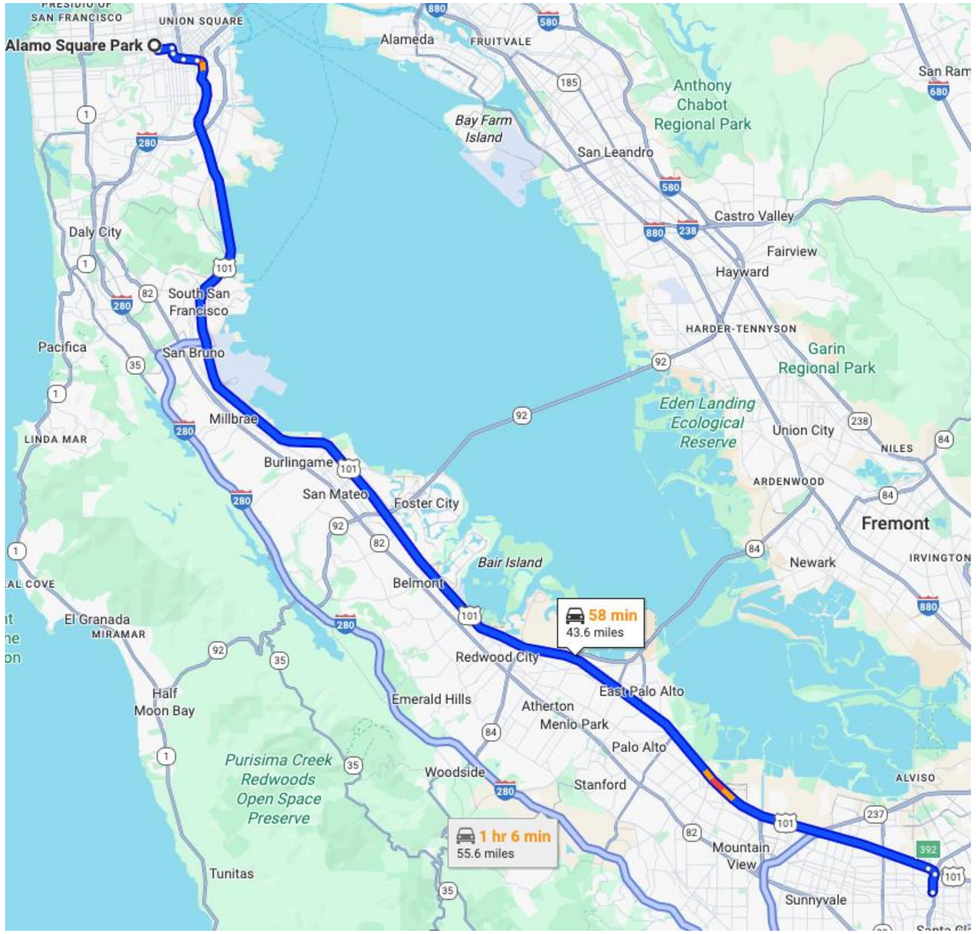
via US-101 S — 59 min

Fastest route, despite the usual traffic — 43.6 miles

Details

via I-280 S — 1 hr 6 min

Some traffic, as usual — 55.6 miles



Latency vs. Throughput

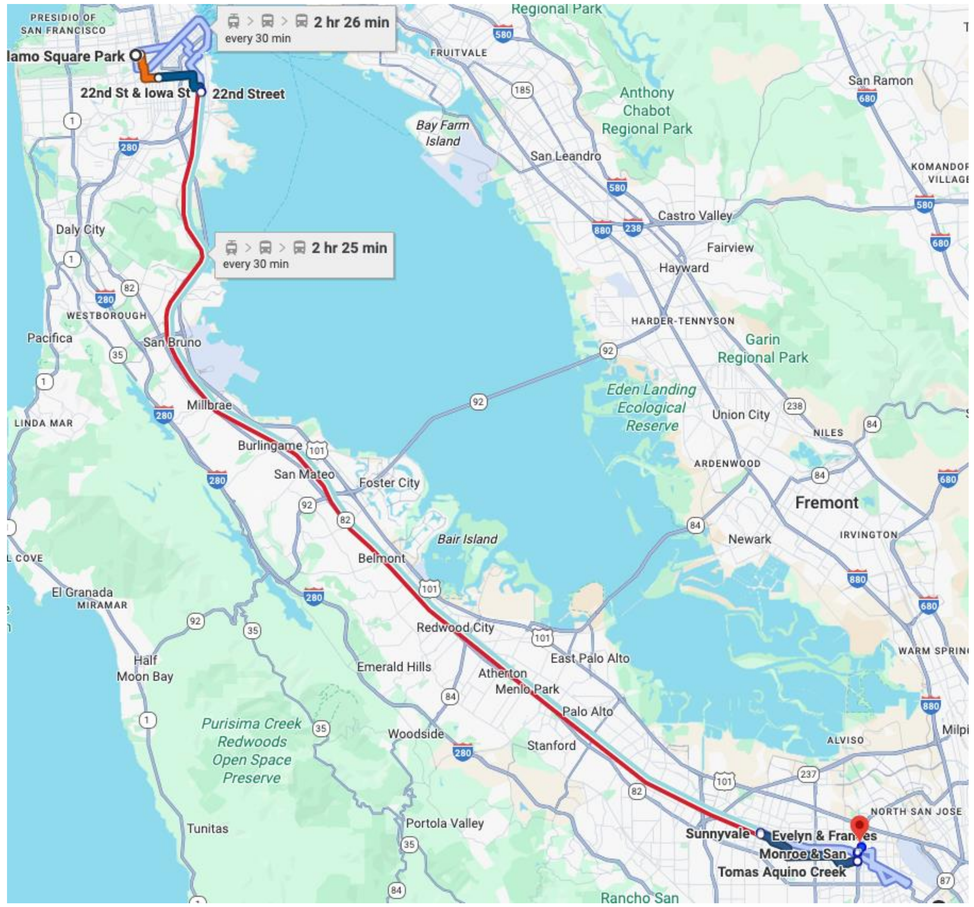
CPUs and GPUs take different approaches to managing latency

Low Latency

via US-101 S — 59 min
Fastest route, despite the usual traffic — 43.6 miles

Details

via I-280 S — 1 hr 6 min
Some traffic, as usual — 55.6 miles



High Throughput

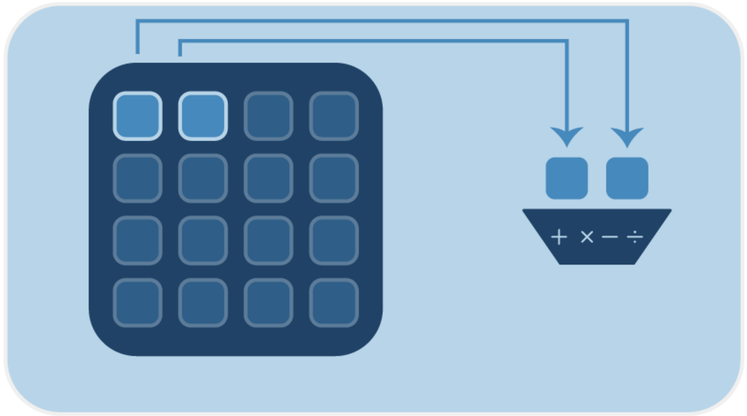
5:32 PM—7:58 PM — 2 hr 26 min
Express Limited 21

6:04 AM (Thursday)—8:29 AM — 2 hr 25 min
Express Limited Local Weekday 21

GPUs Optimize for High Throughput

Modern CPUs support vectorized instructions

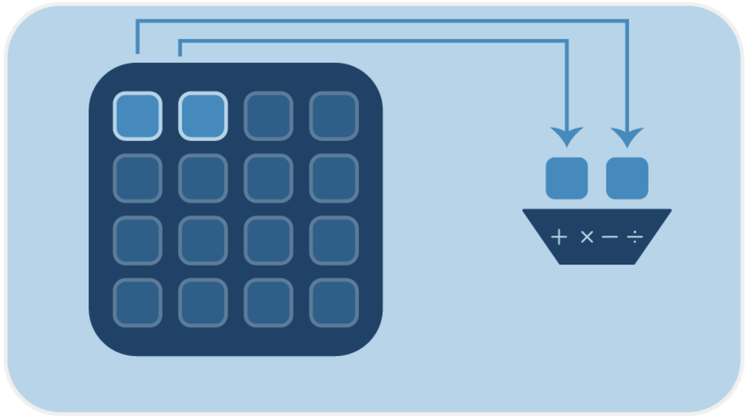
```
v[0] += 1
v[1] += 1
…
```



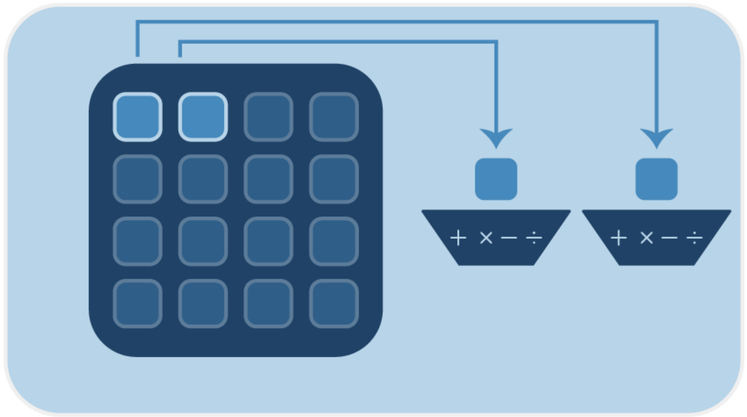
GPUs Optimize for High Throughput

CPU vectorization is determined by compilers from code structure

```
v[0] += 1
v[1] += 1
…
```



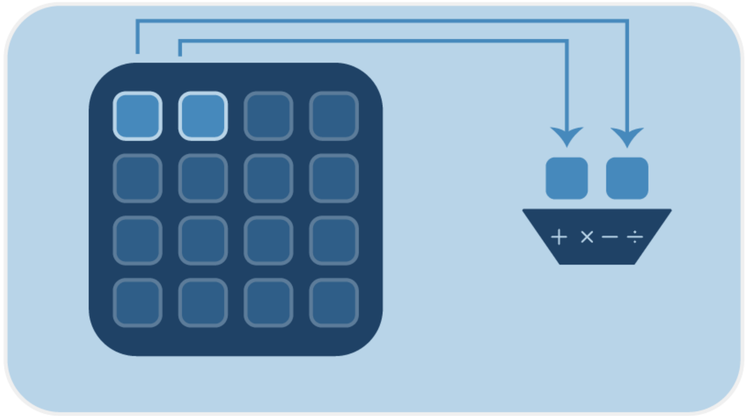

```
if i % 2 = 0:
    v[i] *= 2
else:
    v[i] /= v[i-1]
```

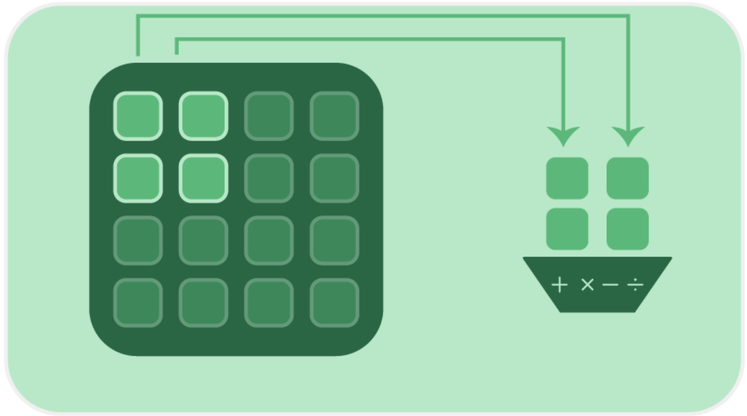


GPUs Optimize for High Throughput

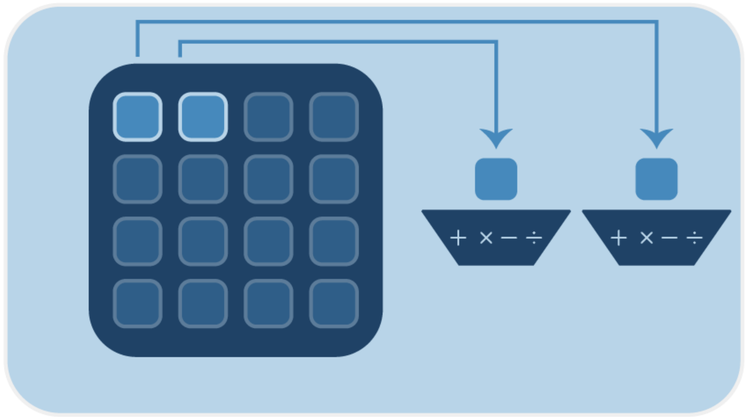
GPUs have orders of magnitude more parallelism that is conditionally leveraged at runtime

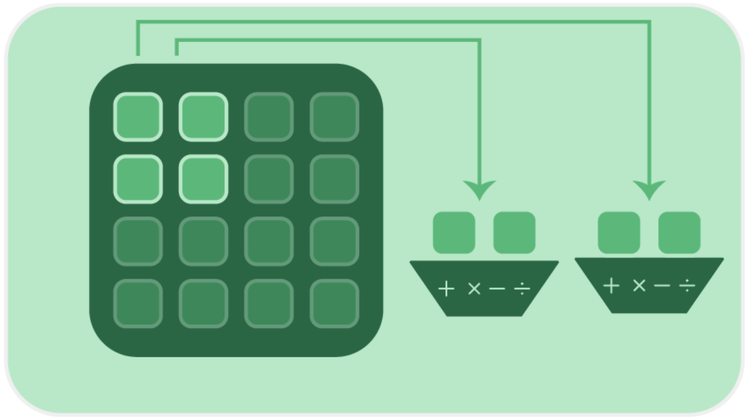
```
v[0] += 1
v[1] += 1
…
```






```
if i % 2 = 0:
    v[i] *= 2
else:
    v[i] /= v[i-1]
```





Why Do GPUs Really Work

GPUs are designed to maximize performance in the face of the memory wall

Representative numbers, not exact

	CPU	GPU
Memory bandwidth	100 Gb/s	5 Tb/s
Latency	100 ns	400 ns
Bytes per latency	10,000	2,000,000

Why Do GPUs Really Work

GPUs are designed to maximize performance in the face of the memory wall

Representative numbers, not exact

Daxpy

$ax + y$

	CPU	GPU
Memory bandwidth	100 Gb/s	5 Tb/s
Latency	100 ns	400 ns
Bytes per latency	10,000	2,000,000
Memory efficiency	16e-3	8e-6

Why Do GPUs Really Work

GPUs are designed to maximize performance in the face of the memory wall

Representative numbers, not exact

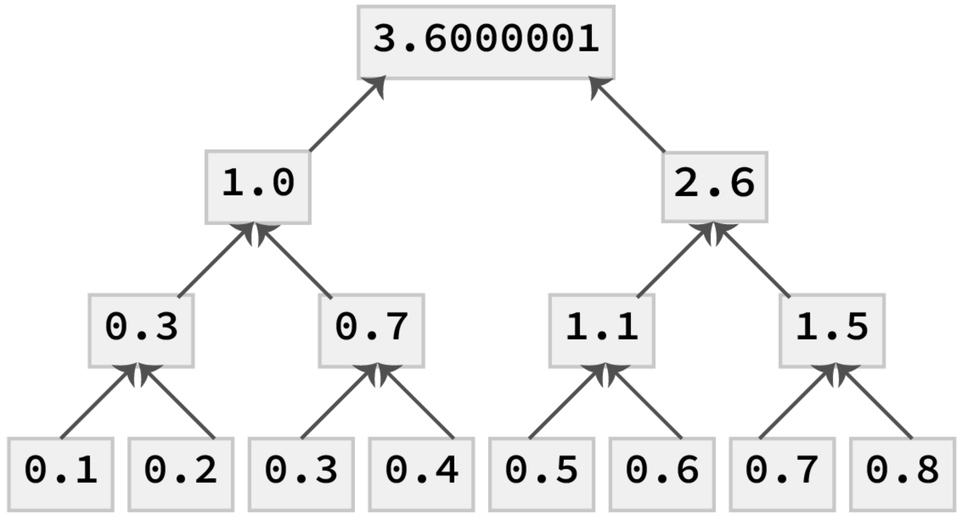
Daxpy

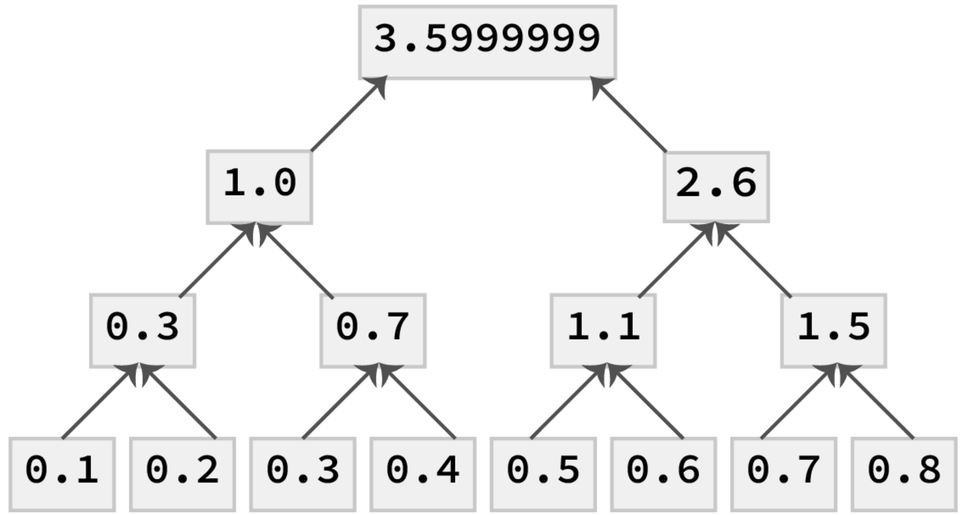
$ax + y$

	CPU	GPU
Memory bandwidth	100 Gb/s	5 Tb/s
Latency	100 ns	400 ns
Bytes per latency	10,000	2,000,000
Memory efficiency	16e-3	8e-6
Threads required	625	125,000
Threads available	200	300,000

Parallel Algorithms Reveal the Limits of Floating Point

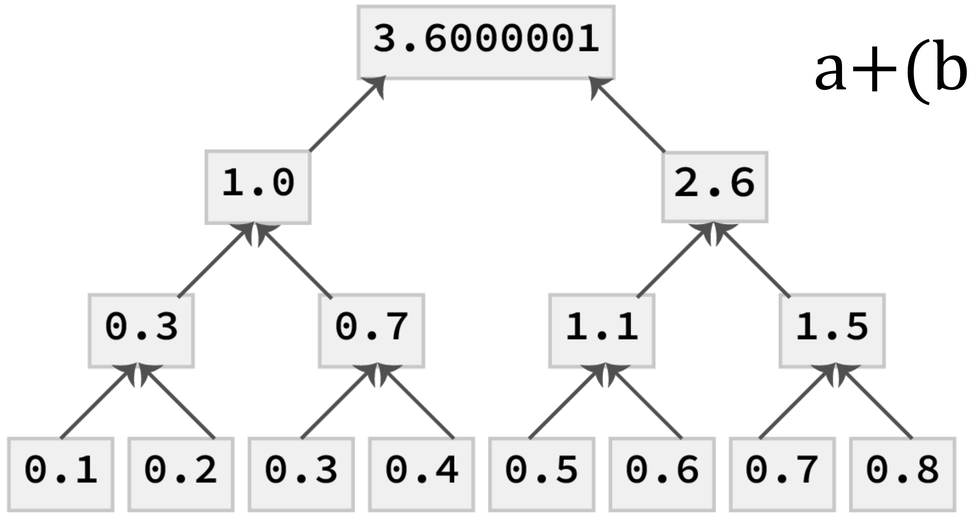
Floating point arithmetic is not associative, so parallel algorithms produce different results



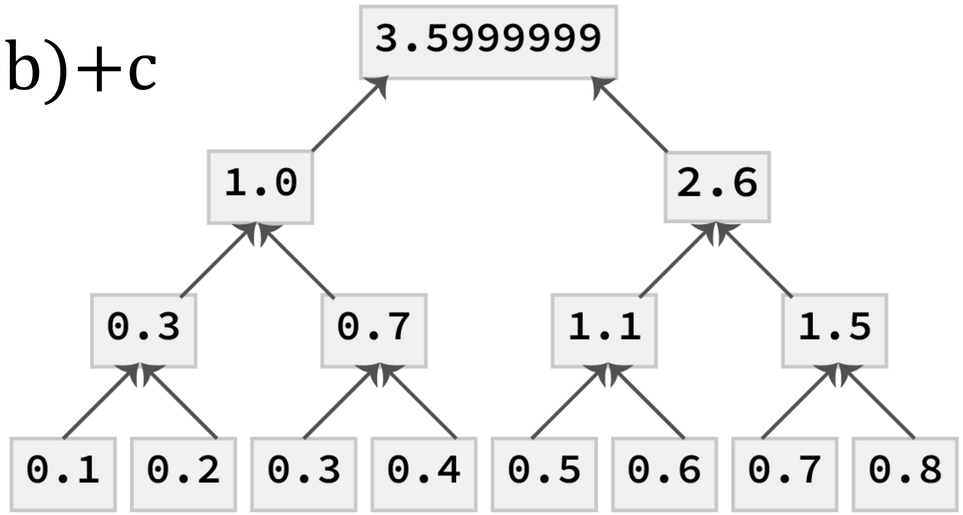


Parallel Algorithms Reveal the Limits of Floating Point

Floating point arithmetic is not associative, so parallel algorithms produce different results



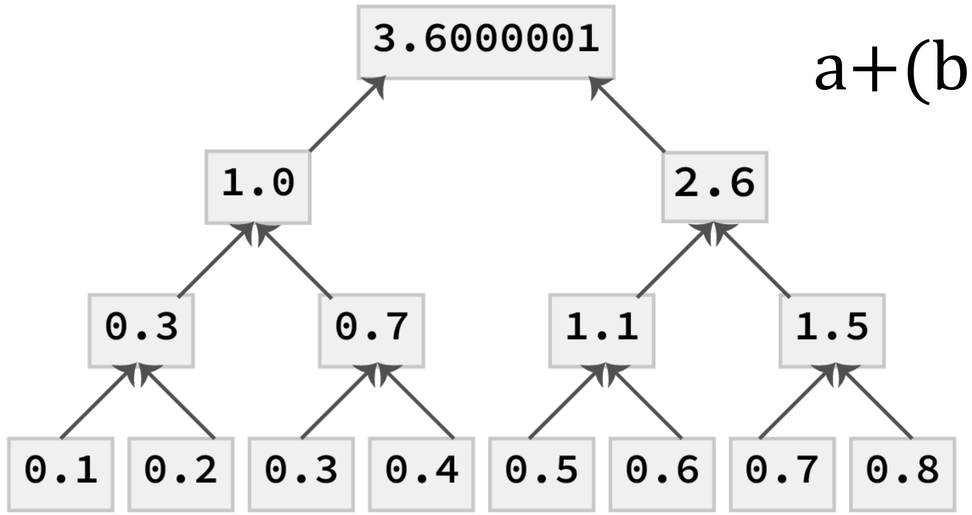
$$a+(b+c) != (a+b)+c$$



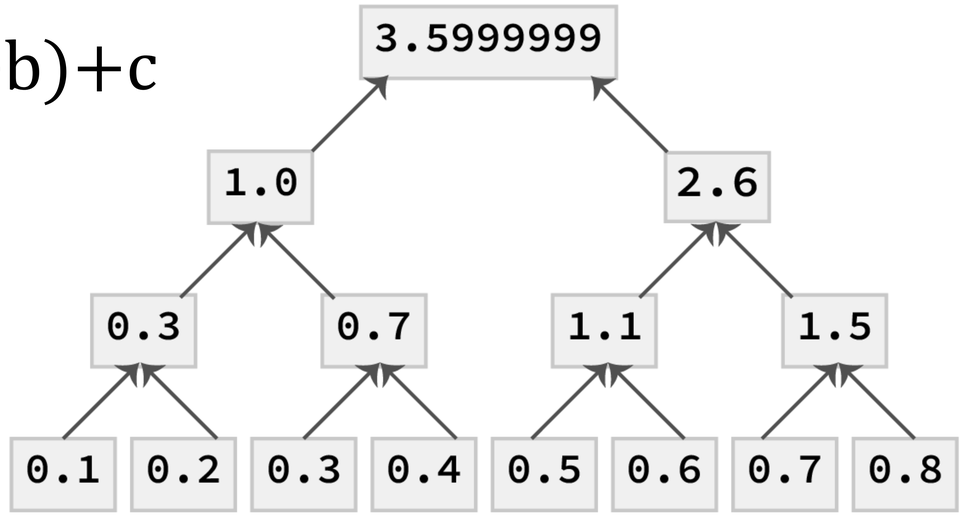
Parallel Algorithms Reveal the Limits of Floating Point

Floating point arithmetic is not associative, so parallel algorithms produce different results

- Use deterministic algorithms (`torch.use_deterministic_algorithms`, `tf.config.experimental.enable_op_determinism`)
- Do not expect equivalence across hardware



$$a+(b+c) != (a+b)+c$$



Summary

These are some key differences in GPU programming that may require your action

Property of GPU Execution	Ways to Handle
JIT compilation	Persistent JIT caches
Kernel launch overhead	Kernel fusion, multi-stream workflows, queuing
Memory transfer overhead	Sequencing memory and compute operations, CPU/GPU pipelining
Floating point error	Use deterministic algorithms, only compare identical hardware