Which Wins? Cheat sheet

Real code, raced on a real laptop. Two rounds each; the rule is what to remember.

No. 6 CPU vs GPU

R1 Multiply two 4096 x 4096 matrices — **GPU**

`GPU 14 ms, CPU 162 ms: GPU 11x faster`

R2 10,000 steps, each needing the step before (x = x * 0.999 + 1) — **CPU**

`CPU 36 ms, GPU 156 ms: CPU 4x faster`

Work that splits into independent pieces goes to the GPU; work that runs step by step stays on the CPU.

Interview: Why do AI models run on GPUs? They are mostly huge matrix multiplies, where every output cell can be computed at the same time.

No. 8 Python vs C

R1 Add up the squares of 100,000,000 numbers in a plain loop — **C**

`C 63 ms, Python 3,882 ms: C 62x faster`

R2 Same job, Python makes one NumPy call: np.dot(a, a) — **Python (NumPy)**

`Python 34 ms, C's loop 63 ms`

A plain Python loop is slow; one library call that runs the loop in compiled code is fast.

Interview: Why is Python slow, and why is it still used for AI? The interpreter runs a loop one step at a time; the heavy loops in AI run in compiled code underneath (NumPy, PyTorch).

No. 9 RAM vs SSD

R1 Look up 100,000 records of 4 KB at random places in the same 1 GB — **RAM**

`RAM 46 ms, SSD 11,249 ms: RAM 245x faster`

R2 The program holding the data is killed (kill -9) — **SSD**

`RAM: 0 bytes left. SSD file: all 1,073,741,824 bytes back, same checksum`

RAM is fast but forgets; disk is slow but remembers. Real systems use both.

Interview: Why do databases write a log to disk before changing data? So that after a crash they can rebuild every change from the log; memory is wiped when the process dies.

No. 10 Threads vs Processes

R1 8 jobs of pure maths, all started at once — **Processes**

`8 processes 132 ms, 8 threads 702 ms: processes 5.3x faster`

R2 1,000 tiny jobs, a new worker started for each — **Threads**

`Threads 65 ms, processes 1,124 ms: threads 17x faster`

Heavy calculation goes to processes; many small or waiting jobs go to threads.

Interview: What is the GIL? In standard Python only one thread runs Python code at a time. Threads still help while waiting on files or the network, and they are cheap to start.

No. 11 Index vs No Index

R1 Find one user by email among 5,000,000 — **Index**

`With index 0.0031 ms, without 121 ms: about 39,000x faster`

R2 Add 1,000,000 new users — **No index**

`Without index 507 ms, with index 920 ms: 1.8x faster; the index also doubled the file (191 MB to 381 MB)`

Lots of reads: add the index. Lots of writes: keep indexes few.

Interview: Why not index every column? Each index makes every insert and update slower and takes disk space.

No. 12 Array vs Linked List

R1 Add up 10,000,000 values — **Array**

`Array 5.3 ms, shuffled linked list 1,060 ms`

R2 Insert 2,000 values at the front of 200,000 — **Linked list**

`Linked list 0.002 ms, array 42 ms`

Arrays win at reading in order (the CPU fetches neighbours together); linked lists win at inserting at the front.

Interview: Why is a linked list slow to walk even though each step is O(1)? Its nodes are scattered in memory, so almost every step is a cache miss; an array's values sit next to each other.

`@whenyoutap · run every race yourself: github.com/whenyoutap`

Times from one Intel Core i9-14900HX laptop. Yours will differ; the winner of each round should not.