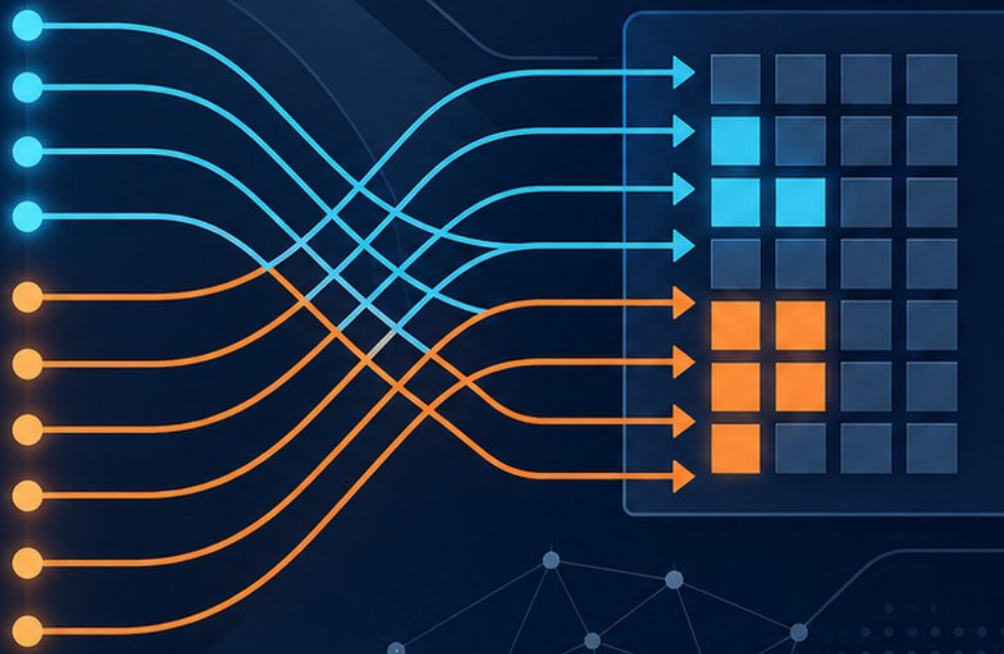
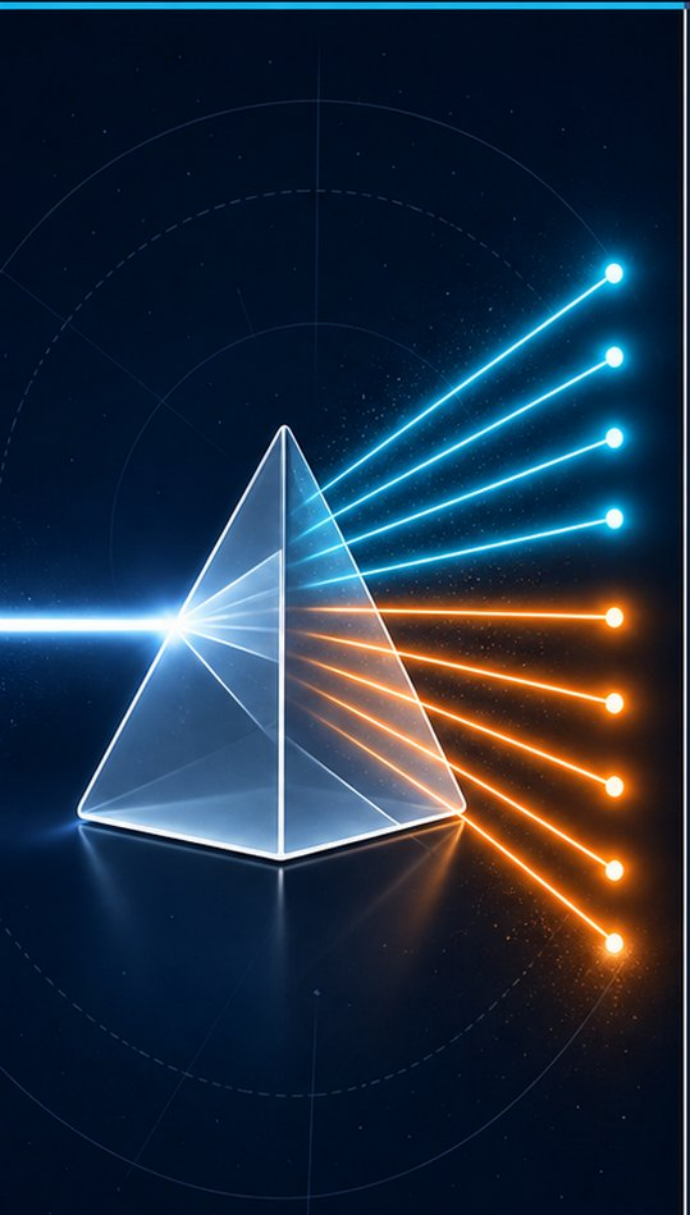
# Introduction to Parallel Computing: Why Do Many Things at Once?

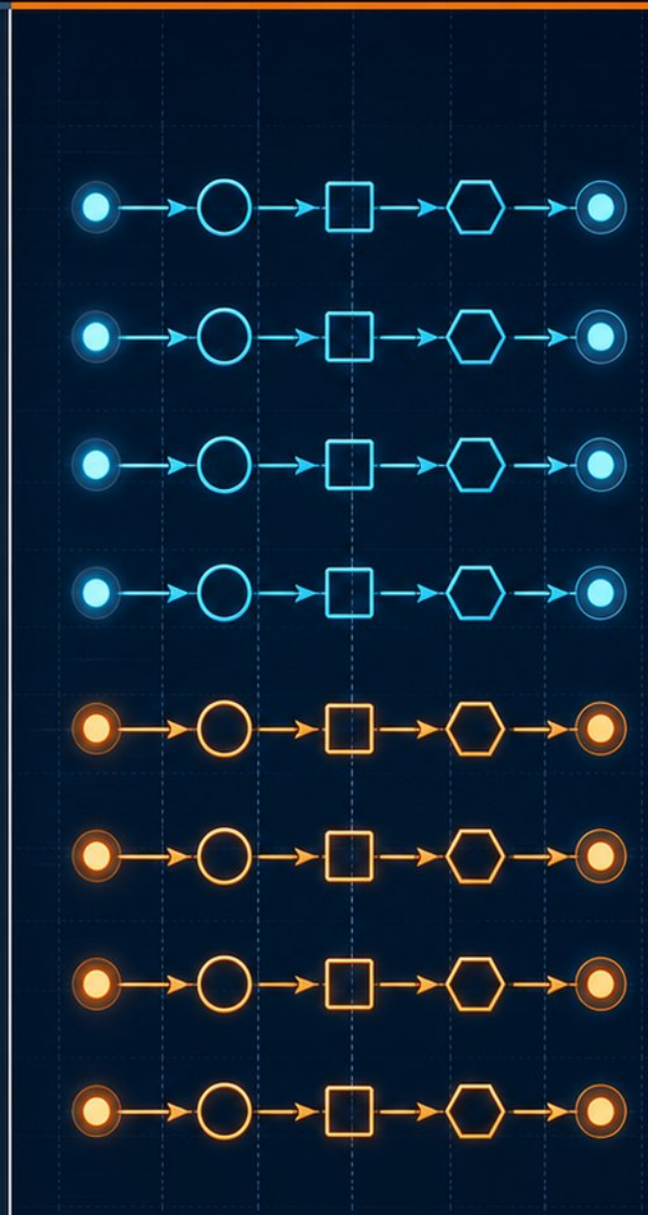
- Performing many calculations simultaneously to solve problems faster
- The ‘free lunch’ ended as single-core frequency scaling hit power limits around 2005
- Like adding more lanes to a highway, more prep stations in a kitchen, or more checkout lines
- Web browsers, video games, smartphone apps, and video encoding all rely on multicore processors today



# Tasks and Decomposition: Breaking Work into Manageable Pieces

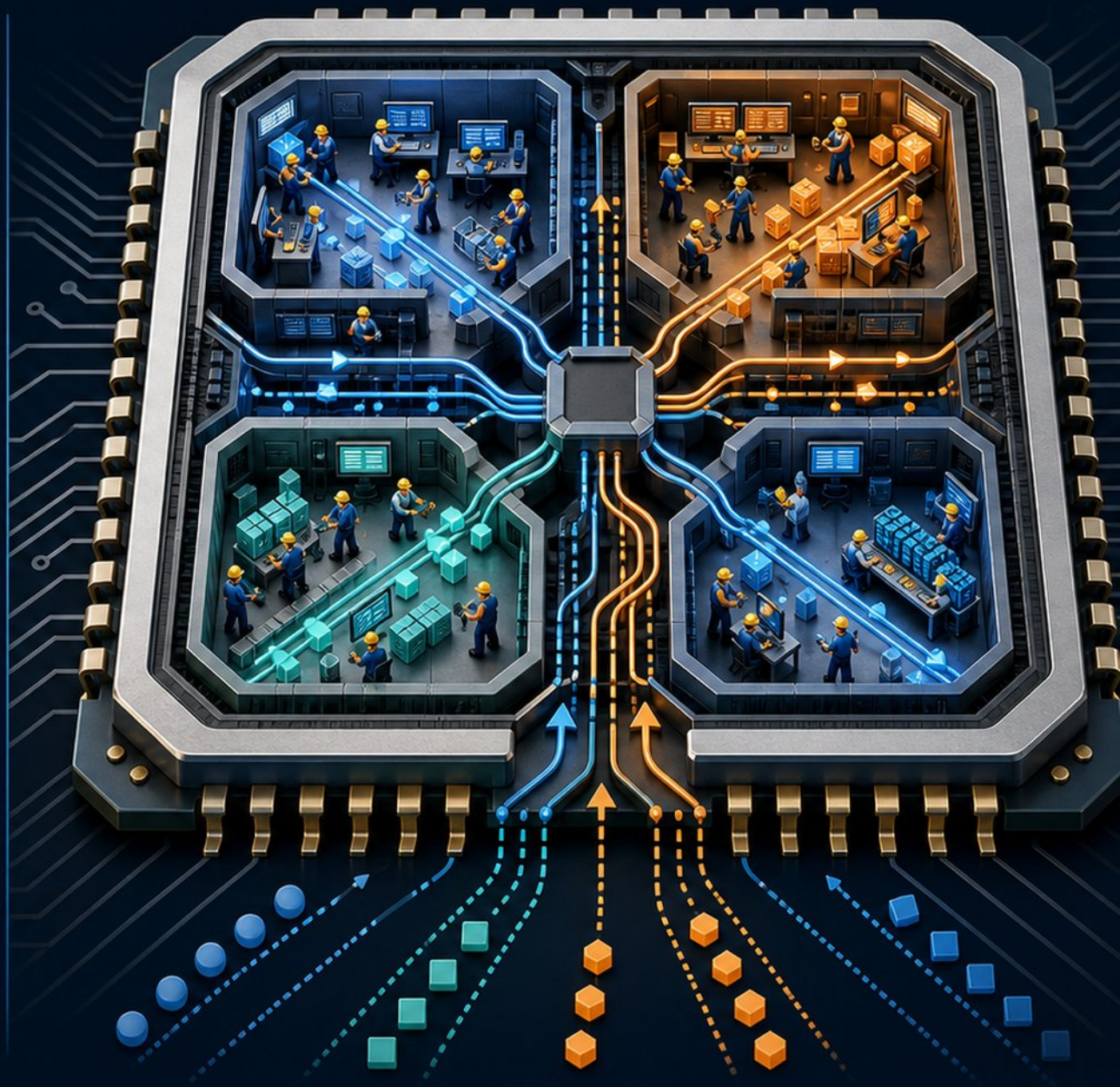


- Identify independent work units that can run simultaneously without waiting for data from each other
- Task parallelism runs different tasks on same or different data; data parallelism applies the same operation across chunks
- Coarse-grained tasks reduce communication overhead but risk load imbalance; fine-grained improves balance but increases management cost
- Example: Batch image resize uses data parallelism; processing different report sections simultaneously uses task parallelism
- Pitfall: Failing to recognize hidden data dependencies that prevent true parallel execution

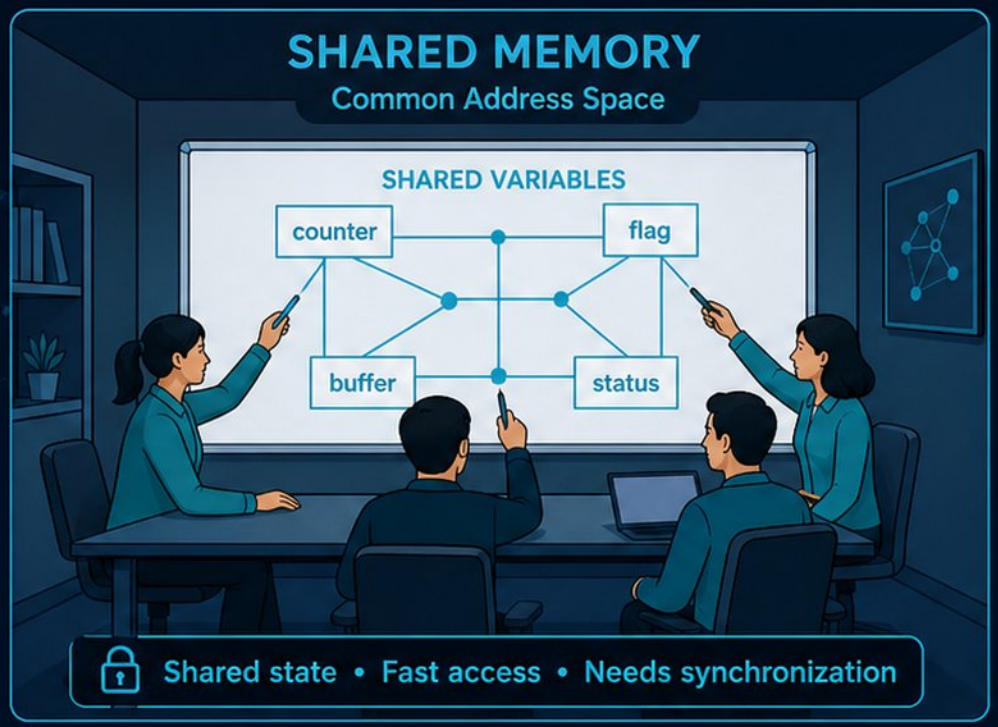


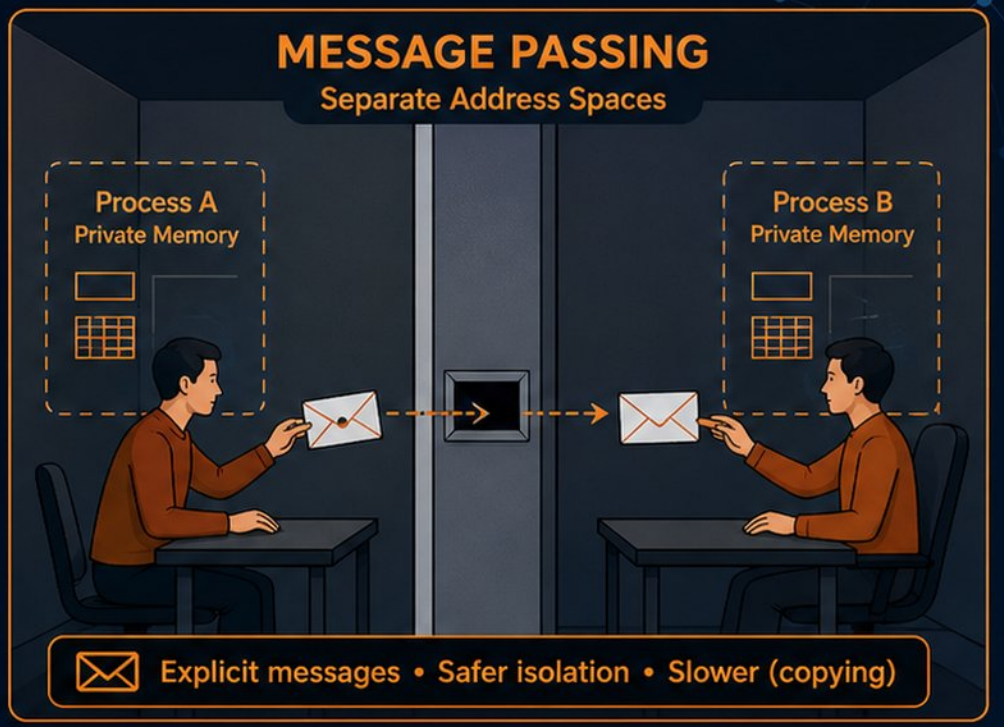
# Threads, Processes, and How Work Actually Runs

- Process: independent program with its own memory space; threads share memory within a process.
- OS scheduler maps tasks to cores via time-slicing, context switching, and core affinity.
- GPUs use many simpler cores for massive data parallelism—ideal for graphics and ML.
- Hardware concurrency (physical cores) vs. software parallelism (expressed concurrent design).



# Shared Memory and Message Passing: Two Communication Models



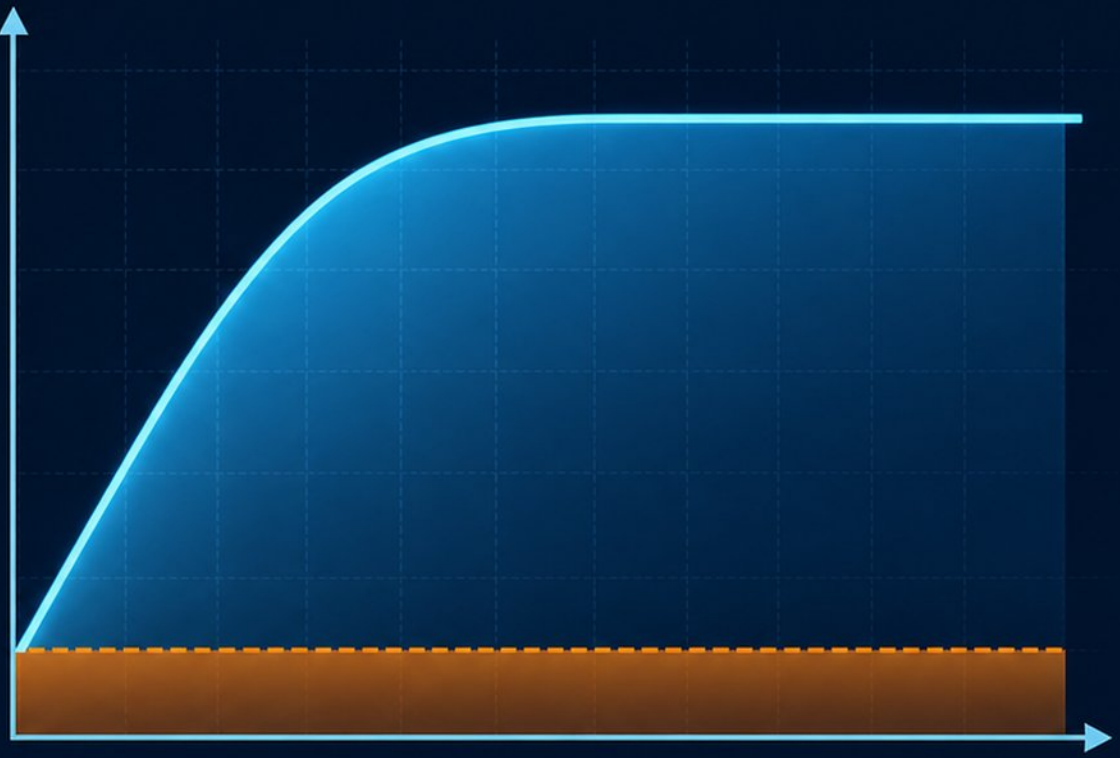

- "Shared memory: threads access common variables directly—fast like a shared whiteboard, but needs locks to prevent chaos."
- "Message passing: processes send explicit messages—safer isolation like mailing letters, but slower due to data copying."
- "Shared memory suits multicore CPUs within one machine; message passing (e.g., MPI) powers clusters and distributed systems."
- "Choose direct access for speed with synchronization, or structured messages for safety and scalability across machines."

# Synchronization and Coordination:
## Keeping Tasks Safe and Orderly

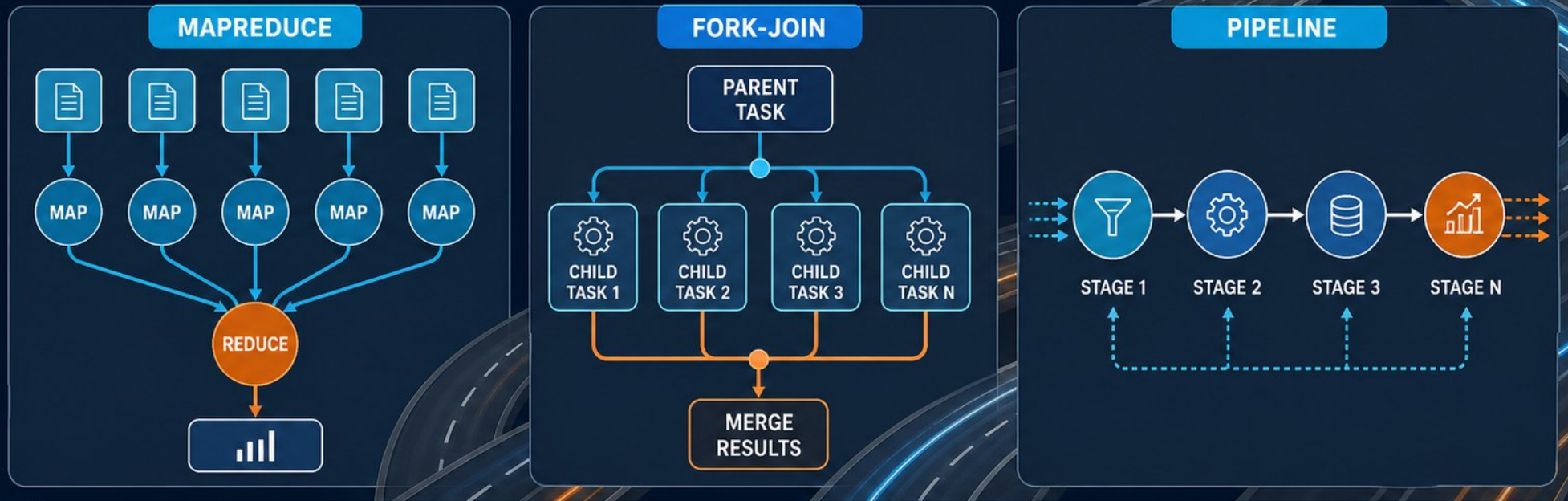
- - Race conditions: unpredictable results when tasks access/modify shared data simultaneously
- - Mutual exclusion (mutex): locks ensure only one thread enters a critical section at a time
- - Barrier synchronization: all concurrent tasks wait at a rendezvous point before continuing
- - Coordination trade-off: too much locking kills performance; too little risks correctness

# Fundamental Limits: Amdahl's Law, Communication, and Load Imbalance

- Amdahl's Law: max speedup limited by the fraction of code that must remain serial (1 / (S + P/N))
- Even with infinite cores, a 5% serial fraction can never exceed 20× speedup
- Communication costs: moving data between workers incurs latency and bandwidth overhead
- Load imbalance: idle workers waiting for the slowest task destroy overall efficiency



# Common Parallel Patterns: MapReduce, Fork-Join, and Pipelines




- **MapReduce:** same operation across large datasets, then combine results (reduce)
- **Fork-Join:** recursive divide-and-conquer—split, solve in parallel, merge results
- **Pipeline:** chain of stages; each stage's output feeds the next concurrently
- Map for independent ops; Fork-Join for recursive; Pipeline for streaming/multi-step workflows

# Pitfalls That Break Parallel Programs:
# Deadlock, False Sharing, and Data Races


- **Deadlock:** threads hold resources each other needs, causing permanent standstill
- **Data races:** unsynchronized access leads to non-deterministic, hard-to-reproduce bugs
- **False sharing:** independent variables on one cache line cause severe slowdown
- **Detection:** use Thread Sanitizer, stress-test, watch for scaling plateaus

# Parallelism in Everyday Life: Web Browsers, Video, and Games

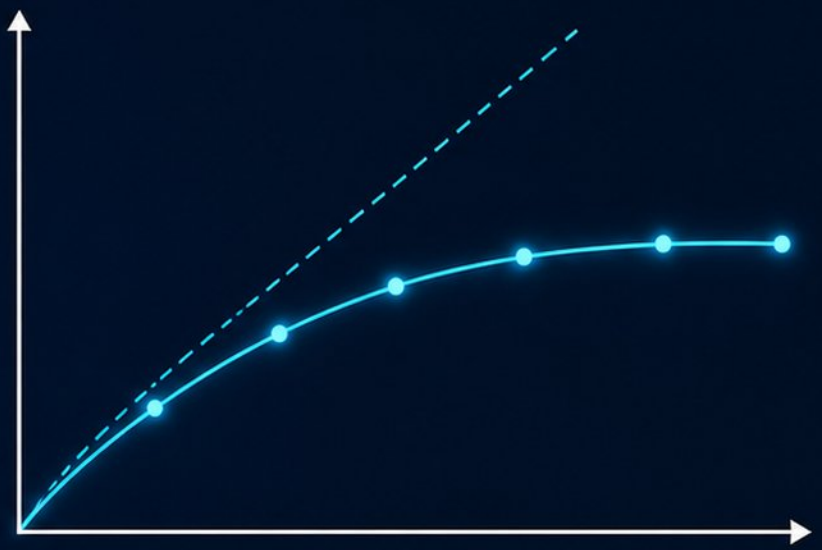
- Modern browsers run tabs and extensions as separate processes for isolation.
- Video tools decode frames, apply effects, and encode output across multiple cores.
- Games run physics, AI, and audio on CPU cores while the GPU renders graphics.
- Spreadsheets recalculate thousands of cells and word processors run spell-check in parallel.
- Smartphone apps and operating systems rely on multi-core for fast, responsive interfaces.

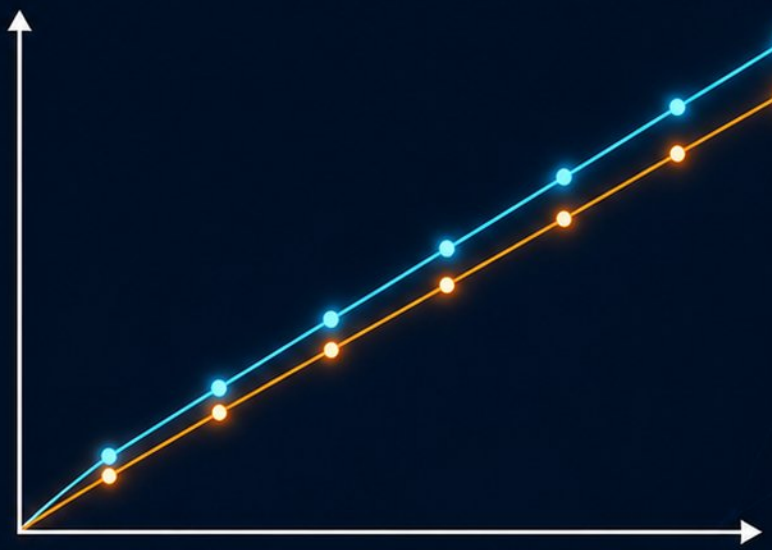
# From Laptop to Data Center: Cloud-Scale Parallelism

- **Desktop parallelism (shared memory):** OpenMP, Python's multiprocessing, and Java's ForkJoinPool leverage multicore CPUs
- **Cluster and cloud parallelism (message passing):** Apache Spark and Hadoop MapReduce distribute computation across many machines
- **Hybrid approaches:** MPI coordinates across nodes while OpenMP manages threads within each node
- **Choosing the right tool:** small tasks fit shared-memory models; terabyte-scale processing needs distributed frameworks

# Evaluating Parallel Performance: Speedup, Scaling, and Efficiency

- **Speedup** = Seq Time / Parallel Time; linear ideal rarely achieved
- **Strong scaling:** fix problem, add cores; limited by Amdahl's Law
- **Weak scaling:** grow problem with cores; guided by Gustafson's Law
- **Efficiency** = Speedup / Cores; drops as overheads dominate new cores





# Deciding When—and When Not—to Parallelize

- - Parallelize only when concurrency gains outweigh management, communication, and synchronization overhead.
- - Checklist: large workload, independent tasks, minimal data dependencies, small serial fraction.
- - Avoid anti-patterns: parallelizing I/O-bound work; creating more threads than available hardware cores.
- - Profile first, identify true bottlenecks, then selectively parallelize only the hot spots.

# Wrap-Up and Your Learning Pathway Forward

- Decompose, communicate, synchronize – then measure speedup and limits
- Start with threads (Python, Java, C++) before OpenMP or distributed frameworks
- GPU (CUDA), clusters (MPI/Spark), and hybrid HPC are natural next steps
- Parallel thinking outlasts any single tool: independence, communication, scalability

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/81ee7f36/public