MULTIPLAYER PLATFORM

OPTIMIZATION

CPU	GPU
MIMD, near + far ||, MC+MT	SIMD, near + far ||, MC
CPU pipeline deep: (automobile factory) Intel Core, Intel Pentium, AMD Bulldozer	GPU small data chunks from GPU RAM (bins): 1 Command unit: VLIW, multiple ALUs Adders, multipliers, dividers, special math
CPU pipeline stall: Structural, data, control (unpredictable, embedded?)	GPU X branching (VLIW) Lock-step (predictable?)
Too sequential, many variables Unpredictable branch, virtual call (rollback, always) Random access, over access, over code	Many variables Branch (sync wait) Random access, over access

DOTS	GDC
1990: actor model, 2000: component (game obj.) OOP: data + code cache miss (lookup, virtual call) Every update(): physics + AI + game logic (all over)	Overwatch: input -> authority server -> user (state) Packet loss: input feedback rate up (sliding win., redo) Latency: action misprediction (correction)
Entity: id (world) Component: physical, speed, health, inventory (state) System: Physics, AI, game logic (behaviour)	Improbable (SpatialOS) run systems in clusters Raft (consensus), fault tolerance, latency, monitor Game client can be worker? Trust? Data intensive?
Entity: composition vs inheritance Systems: {components...} (functional, pure?) Easily add/replace, reduce mutate.	Area heavily occupied? Persistent? Compression? Physics LOD? Large effect? Map changes? (JPEG) Net latency? Screen latency? (VR/AR, satellite?)
Unity: ECS, C# job sys, burst compiler (LLVM) No GC, custom memalloc? RW components. (as attrs)	A bug's life (scale). Edge of tomorrow (rewind). Cloth sim. Tree falls in jungle. (more users => acc.) Space trash disaster. (one keeps sim => realistic)

DESIGN	PERFORMANCE
Coupling: loose? Tight? (code) Class: all vars used in update() (tight) Component: vars processed separately (loose)	More indirections soln.? Too much? Component: respects data, code cache (CPU like GPU? Purity in shaders)
How to manage complexity on growing codebase? More constraints? Visitor? Functional? ECS? One place mutate (side effects)? Decoupling?	Disk defrag., array sort, precomputed buffer (recalc?) Merge sort, bubble sort: I{next} (perf) Array vs linked list (perf). Tress? Graphs? (CPU, GPU)
Pure vs mutation? (a = sort(a)) Interface = iceberg (RBIL regs, now stack) (SDS, screw, clamp, USB, staircase, lift, escalator)	Promise.all() barrier (minimize sync, delay) 200 texts load (all?) -> process (foreach HL logic) Lazy vs eager exec (when better?)
Music? (rock, soulful, boy/girl singer, instrument) Physics: Einstein vs Newtonian calc (impl.) MotoGP: skeleton, rough, texture renderer	Serial access vs random, buffer swap vs copy Serial = dependency. How to measure? (metric) Flash SSD vs Intel optane (DSA)

SOFTWARE INTERFACE ARCHITECTURE	PERFORMANCE
OS in C vs HL? (complex algos) Beginner friendly, easy impl., (faster?, like RISCV) Decoupled libs, open interface for apps (ints. custom)	Struct serial access: reorder fields by usage? By profiling which algo used more (Amdahl auto?) Struct<pad?>, <as-is>, <A> <A>, <B> (GCC attr)
Predict throughput/latency given use case? Group similar use cases, auto design SIA (synthesis?) SIA impl. By compiler? Windows? Android?	Which list impl.? (NativeArray<temp/job/perm.>) How to specify? <A> <A> <B>? Hint? Constraint? Test/opt.other DS for CPU? GPU? ECS? (perf-stat)
OS present SIA by app request? (OpenGL, DirectX) Impl. Insight? Ifs, loops, vars, deps, security, perf? (Like Healthline insight on food from papers)	Serapate opt. logic? Compiler impl.? (profiling/OM) Write HL programs? (a = sort(a), matrix dense/sparse) Shared interface among implementations. (contract)
Say 10 pkgs impl. interface X. App dynamically choose based on arch. / availability? (per/another plugin) (Interface more imp. than version)	Which layer need not be optimized? Changes as we try to solver larger problems. Low data, space search problems more concurrent.