Modern computers are distributed single user systems. Diversity in number of cores and caches is large, even within an instruction set architecture. An open question in resource management is how to execute applications efficiently across ranges of machines. Classical timesharing operating systems offer no solution, especially not as current applications often contain large network or multimedia components that cannot be isolated in user tasks, but cross all system layers. Optimizing buffering and scheduling to match hardware can offer significant gains over fixed solutions: cache aware Unix pipes show a 3x improvement over Linux defaults. To maximize throughput, applications must avoid cache collisions and main memory roundtrips, but also task switches and interrupts. What makes this problem challenging, is that I/O workloads are particularly sensitive to memory system parameters, such as cache size and memory latency, which are perhaps the most diverse parameters in any architecture. Contrary to the parallel and distributed machines of the past, modern systems are cheap and widespread, and therefore have small configuration budgets and non-expert users.

**Contribution** This thesis proposes quantitative optimization at the operating system level to efficiently support diversity with minimal manual configuration. It applies the model to the real and immediate problem of offering high throughput to networking applications, for which it shows factors of magnitude cost savings on common platforms and portability that extends to heterogeneous multicore machines. It offers an evolutionary path from classical timesharing OS design by extending the well known Unix model. To optimize throughput end-to-end, the thesis introduces a single I/O stack that spans user applications, operating system kernel and device hardware. Logically, I/O processing consists of four concerns: communication between components, computation over streams, control of the runtime system and interfacing to applications.

**Communication** based on shared memory ringbuffers removes copying, pagetable manipulation and locking from the datapath to, for example, increase Unix pipe throughput up to 4x. A multi ring architecture satisfies all access control and hardware constraints. To support block filtering, modification and fragmentation without resorting to private copies, applications are given logically private views as sequential streams through a translation step similar to virtual memory: indirect rings store indices that are cheap to update, byte-addressable and valid across virtual memory regions. Rings unlock contingent optimizations. Splicing of indices increases application-level IPC throughput 3-10x over copying of data. Zero copy TCP reassembly replaces data in reconstruction windows with indices. Large rings also reduce computation cost when signal rate is throttled to match ring length. Signaling overhead is further reduced (100x) by moving from task switching to userlevel threading. How much to throttle and which thread model to use depends on hardware parameters. A high level programming model avoids hardcoding these decisions at compile time. Pipelines are a commonly applied in streaming I/O and map naturally onto distributed platforms.

At application load, the control system optimizes pipeline allocation to maximize end-to-end flow for the given hardware. In systems design, heuristics are often employed. We demonstrate that combinatorial optimization is not only more robust to input, but cost-effective. The thesis summarizes the degrees of freedom in resource management, compares search strategies based on speed and transparency and demonstrates an iterative approach to model building. For real applications, a mixed integer program solution is found in milliseconds.

**Interfaces** are based on proven Unix pipelines. The fast communication and computation primitives allow the standard Unix programming model to be used for high throughput networking and multimedia tasks and in kernel and device contexts. By slightly extending Unix shell syntax we can describe more task patterns, such as parallelism. Even more than in Unix, everything is a file when volatile kernel streams and pipelines are accessible as Unix pipes and filepaths in a virtual filesystem. The OS enforces Unix permissions on streams and filters to guarantee strong isolation, but we argue that in many common scenarios, isolation can be reduced and data sharing increased at no loss of threat protection.

**Evaluation** A combination of performance evaluation and real world deployment validates the approach. An open source reference implementation, Streamline, replaces native I/O in Linux 2.6 on x86 and amd64 platforms and on the heterogeneous multicore IXP network processors. Throughput per cycle increases 1.4x for the (small packet) Bind DNS daemon, 3x for the (large packet) MPlayer video player and 3-10x for the (shared access) Tcpdump traffic monitor. The system saturates the network for demanding applications even on the complex IXP. For instance, we demonstrate Gigabit linerate for an edge-host intrusion prevention system that practices defense in depth by combining payload scanning, behavioral detection and application layer filtering. Aside from these immediate applications, this thesis demonstrates that modern computer systems are capable of solving the kind of complex configuration problems that their own diversity poses.