THE - Dijkstra; Main point: sequential processes(central abstraction) in layers; Why: handle complexity; *Benefits*: *verify soundness, prove correctness;* Layers adv: modular, abstraction, debug, verify; disadv: difficult to design(circular dependencies), separate layers, inefficient (call through layers); *Sync*: Semaphores - mutex, waiting queues (cooperating processes); No circular wait because uniprocessor system; priority scheduler; **Monitors(Hoare) -** only one process in a monitor procedure at one time; condV; coarse grained, guaranteed cpu access to support os; deterministic; if(not invariant) wait(c)

Nucleus - Per Brinch Hansen; Motivation: freedom of design not in existing OSes; *Main point: Small nucleus (kernel) supports > 1 simultaneous OS tasks;* Msg adv: Easy to validate, explicit wrt semaphores, simpler; disadv: buffer management, memory is not shared, copying overhead, overflow; Layers: Nucleus (schedule, communicate, proc manip primitives), user; Sync: msg passing; Scheduler: Round robin (slow for large n_children); general and flexible

Protection - Butler Lampson; control operates on domains, used for revocation; owner operates on objects

Hydra - Wulf, Cohen; Main point: capability-based OS nucleus
Goal: construct environments to effectively utilize HW resources
Separate mechanism (protection) & policy (security), rejects hierarchy (layers of privilege, not flexible); Rights types: kernel, auxiliary; Amplification: trust the procedure implementor; Capability list: 1 user multiple files; Adv: fine-grained, flexible right amplification; Disadv: most entries = no access allowed, wasting space; expensive search, hard to revoke, difficult to set up; ACL: single file, multiple users, easy mgmt; capability and type matching done for params in procedure

Tenex - Bobrow, Burchfiel; Virtual "machine" = virtual memory; CLI, terminal interface
Backward compatibility - Old sys call to compatibility package (Jsys)
Working set principle - which processes to keep in memory based on inter-fault time (time b/w 2 page faults of a process); not good - apps should not page in and out; Vm features - sharing (page appears in addr space of >1 proc using direct pointers in VM maps), sharing large portions by CoW; File name fields: device, directory, name, extension, version; FS - versioning, 5 level "tree" (symbolic name for file consists of <= 5 fields, conceptually represents a tree of max depth 5)

MULTICS - Jerome Saltzer; Right amplification by rings; Contributed protection systems, domain transitions, multilevel security policies; Permission rather than exclusion (default = not allowed), access check to each obj, principle of least privilege; no group access (as opposed to UNIX); Descriptors, segments, gates, rings - still used in x86; ACL: easy to set up, slow to check (walk through the list); ACL + CL: protection checked at open; FD returned with capability to R/W; HW holds, checks descriptors; restrict portability; mmap

Medusa; multi-user distributed centralized OS; Break into components (disjoint utilities); utility - distributed among Cm nodes for parallelism file system invocation is mostly remote; spin waiting (not good in uniproc, but lock can be released by other proc in DCS so efficient - short critical section); pros: easy to maintain, simple; cons: SPOF, performance, memory consumption; communication: msg pipes; unsealing: kernel obj associated w application, remote access done by first mapping it as XDL => unsealed; then manipulate; then reseal (remove from XDL) - failover; coscheduling

Plan 9; specialized nodes for a particular task, more cost-effective. Everything is a file; all servers = file servers. Local namespaces - relative path names, custom NS for each user, referencing local names more familiar to users. Union combines directories from multiple NSes. 9P - protocol to connect clients with file servers; implements RPCs for all file methods. Storage: snapshots, no backups; new level WORM (slow retrieval, not feasible today b/c large multimedia files daily). UTF-8, rfork, /n/dump: Multithreading: rfork (specify shared, pvt resources, kernel thread)

UNIX; unifying abstraction - file abstraction for data, I/O. No special HW support - portable
Unified namespace - mount; File - linear sequence of bytes;fork optimization - CoW;
set-user-id: right amplification (invoke privileged ops w/o giving privilege to user);
System does buffering (reads, writes) - 2 buffers - system buffer, library buffer
Shell - forked instead of exec b/c shell proc can crash if cmd crashes, so safer to let child execute (process isolation)

Pilot; single user, single address space and language support; limited features for protection and resource allocation; no FS, hints from applications; processes cooperative not competitive; Mesa and Pilot interdependent; no protection against malicious procs, more against errors (lang based); accepts hints from apps; circularity between files and VM; kernel/manager (mechanism/policy); mmap

Mesa; notify places process on run queue, but does not switch control to it; reason for change - performance(avoid extra context switches in Hoare), remove scheduling from inside the monitor; Deadlock(circular and un-notifiable wait), priority inversion addressed by temp priority inheritance; thread.stop - threads check to see if they should shut down by polling variable

Sprite; network OS, name and location transparency, caching network FS; app interface - extends Unix; single uniform NS, process migration, shared memory; kernel - multiprocessor, RPC; (longest) prefix tables; FS caching - server (reduce delay caused by disk access), client(reduce no. of calls to non local disk), read reuse and client write; sequential - versioning, concurrent - caching disabled; files as backing store;double caching

Distributed V; msg-based IPC for diskless WS and large file servers; no significant penalty for remote access; sync msgs (static, fixed size kernel buffers), small fixed size msgs (reduce queuing, buffering), separate data transfer facility (efficient transfer large data); inefficient: short msg inefficient use of large packet size, sync com prevent I/O and computation overlap, separate data and control increase nw ops; remote ops: implemented in kernel, raw ethernet frames, no per-pack ack

Microkernel: IPC, VMem, scheduling;adv easy to make it reliable and secure, more stable, extensible, configurable; disadv: lots of system call and context switch; L4 address space: delegate pager responsibility, recursive; Linux on L4: runs as user process, syscalls from process to LServer, LServer acts as pager for user process; binary compatibility: shared lib(emulation layer between LAPI and LServer), trampoline (syscalls in statically linked unmodified Linux libraries reflected back to emulation layer); sp ex evaluation: pipes/RPC, Vm ops, cache partitioning

FFS; improvement on UNIX FS, larger block size(4096), more bandwidth utilization, inodes and data closer(locality); throughput tied to free space, read as fast as write(In FFS, blocks are placed on disk in a much better layout, even with sync reads, placement roughly corresponds to a good ordering of disk requests, writes have higher overhead because they must allocate new blocks) UFS writes faster(async writes, reorder requests to minimize seek time); extent-based w/ journaling

LFS; FS designed to exploit HW and workload trends: large memories, disk BW scaling but latency is not, smaller file access; FFS problems: sync writes, possibly related files far from each other, inodes placed separately from files; challenges: how to find data(indirection, inode map in log, pointers to imap in checkpoint, keep in memory), free space; adv: cold segments cleaned at much higher use; disadv: inodes appended to log, difficult to find, inode location not fixed in LFS so not as easy as FFS to calculate, manage free space (GC competes with disk read/write); crash recovery: checkpoints

Exokernel; main point: customize OS interface / abstractions - move everything to user level.
Motivation: performance suffers from generality, can't customize/extend; Approach: OS layer exports HW resources directly, OS functionality at user level in untrusted library OS. FS part of LOS (no protection, memerr can corrupt on-disk data structures) - address by SFI; track resource ownership by tables, protection (secure bindings) techniques - HW (protected data sharing via VM HW), software TLB on top of HW TLB, packet filters for downloaded code; disadv: no protection within VAS; resource revocation - if LOS doesn't cooperate, "abort protocol" forces

Grapevine: distributed message(email) and registration service; msg service: delivery, buffering of msgs; reg service: naming, authentication, access control, resource location for clients; independent of each other and use IP to communicate; client user package: name /address transparency; client-server- ethernet, server-server - low B/w modem;scaling issue: distributed lists, solve by indirection; transparency:distribution and replication; why not transparent? Updates are not transactions, duplicate msg delivery

Xen; high performance VMM supporting strict resource control among guest OSes, OS that exposes vHW interface; isolation among VMs, heterogeneous OSes simultaneously; paravirtualization - guest OS modified; x86 CPU easy to virtualize because of rings, memory difficult because HW TLB and no tagging => V-P-M, insert V-M, write protect PT; syscalls routed to guest OS via Xen w/o mods, handled so that no trap to Xen necessary, exceptions propagated to guest OS by Xen events, async notifs; guest OS modification supports event handlers

Rio; write-back performance with write-through reliability; mem perf (eliminates synchronous writes) with disk reliability (protects file cache during normal operation, restore contents on warm reboot - system reset, power not lost); protect file cache in kernel:vm support, read-only file cache; protection domains separate from address spaces, separation of file buffer cache into separate protected module; more reliable than write-through - faults corrupting file cache pages caught with protected file cache; fault injection; checksum, memTest

Soft Updates: order operations so that metadata is consistent in case of crash; create file: write inode safely to disk then update directory block; soft update = enable write-back caching of metadata; collapse multiple updates before going to disk; transfer multiple updates to disk in one write

MACH: maintain all VM state in machine independent module; large, sparse VAS, CoW, memory mapped files, user level pagers and backing store;DS: address map, mem obj, resident PT, pmap (machine dependent); easy to share mem regions; different regions can have diff pagers; shadow objects: CoW creates a copy only if *extremely* necessary (only modified pages)

VAX: OS is essentially just an extension of user address space of every process, OS can directly access user code and data, kernel stacks in user address space; command interpreter is a part of each user's address space; optimizations: local page replacement(FIFO),page caching,clustering; free list: clean pages evicted from a process resident set, modified list: dirty pages: second chance algo

GMS; probability based (imp for DCS), clusters of computers act like tightly-coupled microprocessor than LAN, globally shared local resources; goal: minimize avg data reference time (go to remote MM instead of disk); maintain page age, find page in cluster, make replace and evict decisions

Cells; resource constraints on smartphones, existing virtualizations require OS mods; container virtualization (OS namespace) - FS (own root partition, no files owned by other VPs), process (only procs running in own NS), NW (IPs, port mappings), device (/dev); foreground, background VP, if fg doesn't acquire exclusive access, HW shared by bg VPs

Scheduler Activations; vessel to execute user thread context, mechanism to notify user of kernel events; stores user thread state in kernel. Design aspects: upcalls (add proc, proc preempted, SA blocked/unblocked), downcalls (hints: add more procs, proc idle), critical sections (on preemption/unblock: recovery - run thread till end of CS; deadlock free); SA not reused to notify b/c contains exec state of thread; goals: other ULTs can run during blocking, no proc idle when ULT ready, no priority inv

Lottery Scheduling; probabilistic mechanism for proportional CPU share; relative rate; scheduling; tickets - abstract, relative, starvation free. Transferable (inheritance via tickets) - solves priority inversion, inflation/deflation, currencies(hierarchical allocation + keep allocation isolated along trust boundaries), compensation tickets - temp inflated value; proportional share is goal; can't express response time differently from share

Taintdroid; android;identify sensitive data, taint and track data flow (vars - local, args, static, classes, arrays; msgs - upper bound of tainted vars in msg; methods; files - same as msg), real-time monitor behavior of running apps, identify misuse of pvt data; works even if data encrypted; significant performance overhead **VM370:** Control program**(**CP)-VMM, CMS (Conversational Monitor System) - guestOS; FS - disk; single dir, no hierarchy, sharing(CP to mount), I/O(record); mem model single addr space, RSCS(store/fwd nw router)

MapReduce: Reporter: provided for MR to report progress,indicate that they are alive; inefficient for multipass algorithms, no efficient primitives for data sharing; intermediate written to local disk, output files written to DFS; utilize data localization by running map tasks on machines with data, one becomes master and assigns tasks to idle. Tasks scheduled based on location of data, map fail - rerun; Workers are periodically pinged by master, writes periodic checkpoints, avoid straggler

GFS; Failures are norm, append writes, co-design apps and FS API, automatic failure detection, tolerance, recovery, concurrent appends by multiple clients, 64MB chunk servers: intra-rack BW > inter-rack BW, garbage collects orphaned chunks, migrates chunks b/w chunkservers, file data not cached on chunkserver or client, append at least once, separate data and control flow: client interacts directly w chunkserver, master server state logs stored on disk, atomic metadata changes by single master: consistency, client deletes file: master records deletion in log, renames file to hidden name, removes every 3 days
Disadv: single master, bad latency, multiple/small file support weak, relaxed consistency

BigTable; distributed multi level map, fault tolerant, persistent, scalable, self managing; *NoSQL benefits:* auto sharding(splits a single dataset into partitions), replication, integrated caching; tablet - row range, unit of distribution and loadBal, split as table increases; single row transactions(atomic read, modify, write sequence), no support for general transaction across keys, components - GFS (persistent data storage, SSTable), Chubby(distributed lock manager, master election), master(load balancing, garbage collection), tablet servers, lib linked to client (read/write directly with tablet server); 3 level (B+ tree); empty cache - 3RTT, stale cache - 6 RTT; locality groups (Segregating columns families that are not typically accessed together enables more efficient reads) and compression

HayStack; goals to serve photos: high throughput, low latency, fault-tolerant, cost-effective, simple; adv: reduced disk I/O, simplified metadata, single photo serving and storage layer; haystack cache uses DHT, add a photo to cache if request directly from browser not CDN, photo fetched from write-enabled store machine; index file allows quick loading of needle metadata without traversing larger Haystack store file; cookies insufficient protection; using column family / tagging helps for albums