2. Naming

- **Name**: · refer to an object
  - · Pure name: no information in its name to what it refers (=Mac Address)
  - · Address: Encodes information about reffered location
- **Binding**: Binds name to an object ↳ IP-address
- **Context**: Set of bindings (=namespace / scope)
- **Resolution**: Name + context ⇒ Object
- **Dictionary**: Object that has a table of bindings
- **Naming network**: · Directed graph
  - · Nodes: Names or objects
  - · Switching context after each node
  - · If it is a tree: Naming hierarchy
    - ↳ e.g. file system
- **Indirect entry**: · Alias, symbolic link, Verknüpfung
  - · Binds to a pathname
- **Search path**: · Go down the naming network and try to find the to look up name in each context
- **Synonym**: Two names for one object
- **Homonyms**: One name to two objects (=DNS A-records)

3. Classic Operating systems and the Kernel

- **Roles of operating system**
  - **Resource manager**: multiplexes hardware to software
  - **Illusionist**: Virtualizes hardware-resources
  - **Glue**: glue different hardware components together, so that programs can run on different platforms (provides high level view)
- **Domain**: · collection of resources or principles (=invariants)
  - Shared memory domain: cores which share same memory
- **Parts of the OS**
  - Kernel: - Executes in privileged mode
    - Just a special computer program
    - Responds to calls, interrupts & traps
  - **Libraries**: - Support running programs
    - Perform low-level functions
    - Talk to Kernel in higher level
  - Deamon: - Push some things to user-space, if task doesn't need whole view of machine
- **OS models**
  - **Monolithic kernel**: · Do as much in the kernel
    - No daemons
    - If kernel crashes => everything crashes
  - **Microkernel**: · Do as much work in daemons
    - Only memory allocation context switching
    - Make OS more robust to bugs
    - Slower, more context switches

- Exokernel: · Do as much work in libraries
- Multikernel: · Run different kernels on different cores

- Bootstrapping (= Booting)
  1. BIOS: Initializes the hardware and sets up environment for next program
  2. Boot loader: Load OS-kernel into memory and start it
  3. OS-kernel: starts new processes

- Kernel mode
  - User mode (= Running user software + daemons)
  - Kernel mode (= Running the kernel)
  - Protect the kernel from user-software
  - When entering kernel-mode, kernel must save execution-state of user-mode

- Interrupts: · Asynchronous interrupt (= hardware)
  · Synchronous interrupt (= exception)

- Trap: · Intended exception
  · Occur at the request of a programmer
  · Used to implement system calls (= Mode transfer)

- Upcall: · Inverse system call
  · Can be used to let the program know when it was rescheduled

- Kernel works directly on machine hardware

4. Processes

- Execution of a program with restricted rights
- A process bundles a set of hardware and software resources together
- A virtualized machine executing a program
  - One or more virtual CPUs
  - Virtual memory
- Creating a new process

  General: - Process (parent) creates a new process (child)
  - This creates a tree

  1. Approach: - Spawning a completely new process from scratch
  - Need to define a lot of arguments what the process should later run

  2. Approach: - Forking a new process that is an exact copy of the parent
  - Only difference is return value from fork
    - Parent: Childs PID id
    - Child: 0
  - Really expensive: copy whole space
- Init is the first process and therefore the root node
- Running: a process that currently runs
- Runnable: a process that is in standby
- Blocked: a process that waits for event

- Zombies: · Processes that exited
  · Not fully deleted, because parent needs maybe exit-return-value
- Coroutines: · Express concurrency without parallelism
  · Subroutines
- User Threads: · Implemented in a user process
  · Lightweight processes
  · If one thread blocks => process blocks
- Kernel Threads: · Not many
  · Used if Kernel works concurrently

5. Inter-process communication (with shared memory)

- Test and set: · If 0 => set 1 => return true
  · If 1 => return false
- Compare and swap: · Wenn während dem Ändern die Variable nicht verändert wurde, ändere sie
- Load linked: · Load word out of memory atomically
  [Do some work on loaded memory]
- Store conditional: · Store new data if memory location hasn't been altered since load-linked
  → Something like transactional memory implemented with cache-coherency protocol

- Transactional memory
  · Data conflict occurs if another processor reads/writes a value from the transaction's write set, or writes to an address in the transaction's read set
- Spinlock: · Repeat TAS until we have read a 0
  · Only makes sense on multicore-systems

Inter-process communication (without shared-memory)

- Asynchronous/buffered: Sender does not block
- Synchronous/unbuffered: Sender does block

1. One sets up a pipe => Returns two references
2. Fork process

- IPC (= Remote procedure call) avoid writing code again

10. I/O

- Device: · Piece of hardware visible in software
  · Exposes a set of hardware device registers
  · A source of asynchronous interrupts
- Device register: · Reading can return input data or status informations
  · No guarantee that reading from a device register will return the same value that was last written to it
- Asynchronous interrupts: · Signal from device to CPU
  · Eliminates polling problem
  ↳ Always check, if register changes
- Direct Memory Access: · Device can copy data on its own
  · Saves BUS-bandwidth, since the data doesn't need to be copied through the CPU
  · A single interrupt is needed to signal the end of a DMA-copy
- First-level-Interrupt Service routine: (Hardware/user decoupled)
  · Code that is executed after device-interrupt to process the new device state
  1. approach: blocked driver thread that waits for interrupt
  2. approach: inform device-driver thread

- Device driver: · Top-half: called by program
  · Bottom-half: called by interrupt
- character device: Just a bitstream (e.g. keyboard)
- Block device: Has a structure with blocks (e.g. disk)
- Network device:
- Pseudo device: Software service (e.g. random numbers)
- Put memory-management-unit between DMA and memory to restrict on what physical memory locations the device can write/read

9. Scheduling

- Uniprocessor scheduling: Multiplexing tasks on one core
- Non-preemptive: allows a job to finish once it has started
- Batch-scheduling:
  - set of batch jobs, each runs for a finite amount of time
  - Arrival time: time at which it becomes runnable
  - Hold time: start-time − arrival-time
  - Start time: Time batch is executed
  - Execution time: end-time − start-time
  - End time: batch finished
  - Wait time: end-time − arrival-time
- Throughput: Number of batches per time-unit
- Overhead: CPU time scheduler runs itself
- First-come-first-serve
  - FIFO
  - convoy phenomenon
  - 24s, 3s, 3s jobs
    1. (24 + 27 + 30)/3 = 27 mean
    2. (3 + 6 + 24)/3 = 13* mean
- Shortest-job-first:
  - Minimizes average mean waiting time*
  - Does calculation only when job finishes: newly arrived short job can experience long waiting time because of long job already running

- Preemtive: · Scheduler can interrupt jobs
- Shortest-job-first: · Longer jobs are interrupted quite often
- Interactive-workload: · long running blocked process which waits for some event
  · servers / Databases / user-interfaces
  · Respond-time: time to respond to event
- Round-Robin: · Response time unpredictable since it doesn't know where we are in the run queue
  · Scheduling quantum q: how long a task can run
  · 50 processes, switch time 10 µs
    - 100 µs quantum: 9% overhead,
      50 · 1010/2 = 25250 response time
    - 1000 µs quantum: 0,98% overhead,
      50 · 1010/2 = 25250 response time
- Priority-based-scheduling: · Processes with same priority are scheduled with other algorithm
  · Can lead to starvation
  ↳ Priorities change over time
  · Interactive: high priority
  · Batch/Background: low priority

#If round-robin queue is at half distance to current job

- Priority Inversion: 1. Low priority aquires lock
  2. Rescheduled by high priority
  3. High priority wants to aquire lock
  4. High priority blocks
  5. Medium priority is run

  - Priority Inheritance: Gets highest priority while holding lock
    (> Scheduler involved in every lock aquire/release
  - Priority Ceiling: Runs at the priority of the highest-priority process which can hold the lock

- Hard-real-time: - Each task has a done-time
  - e.g. engine management units
- Earliest-deadline first
- Soft-real-time: - Affect quality of result
  - e.g. video encoding
- Multiprocessor-Scheduling:
  - Work conserving: If no core is idle when there is a runnable task
  - Naive sequential multiprocessor scheduling:
    - One queue for all cores
    - Work conserving
    - Not good => Cache of different cores

- Affinity-based scheduling
  - Aims to keep same jobs on same CPU
  - Each core has own queue
  - Not work-conserving
    ↳ steal jobs from other core

Memory Management and Virtual Memory

- Segments : · Before pages
  · Base : starting address } Access right on physical memory
  · Limit : ending address }
  · Each process had its own base+limit
  · But base was not known to process
    ↳ Position-independent code
    ↳ Relocation-register : add offset Base
  · No share of memory
- Segment was a triple : Base, Limit and segment-identifier
- Segment table : · Segment identifier => Base+Limit
  · Can be cached in TLB
  · Entry is independant of process
    ↳ sharing is now possible
- Segments lead to external fragmentation
- Paging : · Divides physical memory into physical pages
  · Divide virtual address space into virtual pages
  · MMU : VPA -> PPA
    ↳ Page table or TLB
- Page fault : process has no right on VPA

- Hierarchical page table: · different layers
  · each layer translates set of bits
- Reverse mapping: · Physical => virtual
  · Mark virtual mappings invalid if physical location is reused
    ↳ Defragmentation
- Paged segmentation: each segment has pages
- Fork copy-on-write: · Allocate physical pages of children lazily
    ↳ Makes fork efficient
  · Two address spaces link to same physical memory until one wants to write
- Caches: · Clean: writes dirty data from cache back
  · Flush: clean + invalidate cache
  · Synonyms: different cache entries (virtual addresses) refer to same physical address
    ↳ cache bomb
  · Homonyms: Multiple physical addresses using same virtual address due to different address-space
  · Virtually indexed + tagged: Suffer from homonyms
    ↳ add address-space-tag
    ↳ Cache doesn't flush on context switch
  · Virtually-indexed, physically tagged:
    - mostly physical tag ≤ virtual tag

· Physically-indexed, virtually tagged
  - Not usable
- TLB: · Shootdown: avoid outdated entries in every TLB

---

18. Demand Paging

1. process accesses virtual address
2. Because that address wasn't accessed lately it was moved from main-memory to disk
3. Page fault occurs
4. That page is loaded back into main-memory to be accessed

- Looks like a large main memory but we have small main-memory + big disk
- Demand paging is lazy (=Lazy swapper)
- Goal: minimize page faults
- Effective Access time
  - page fault rate p: $0 \leq p \leq 1$
  - EAT = $((1-p) \cdot m) + (p \cdot (o+m))$

    m: memory access (normal access)

    $o+m$: paging overhead (moving data) + page fault + exchange
- Page replacement policy
  - What page in main memory gets thrown out
  - Optimal page replacement: · Minimizes page faults
    · List of accesses known
  - First-in-first-out: · Increasing size doesn't improve performance (=Belady's Anomaly)

- Least-recently used: · Needs to know not only the page faults also the page accesses
- 2nd change: · Has an accessed bit
  · Round robin clock manner or first in first out
  · Each time page is accessed, set it to one
  · Look for page to remove, go around clock, set all accessed bit to 0, and if an accessed bit is already zero, replace it
  · If no hardware access bit
  ↳ Mark virtual-page invalid

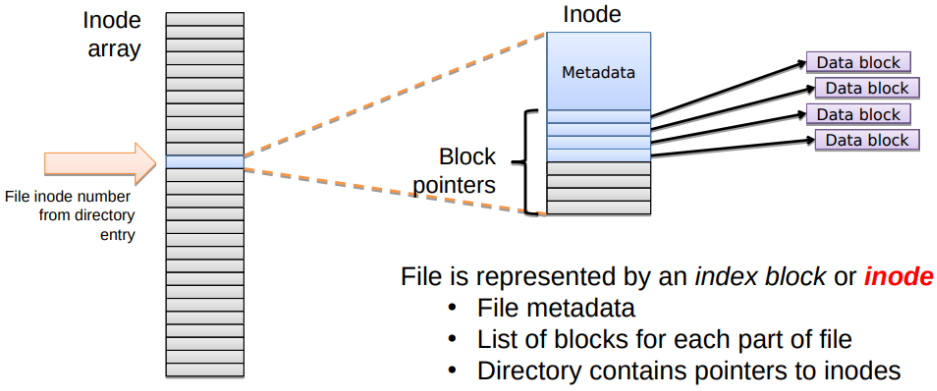
- Global physical page allocation doesn't work because each process needs some pages in main-memory to retain runnable
- Local physical page allocation
  · Measure page faults per process
  · Increase memory for high-faults processes
  ↳ Not good for large processes => working set
- Working set
  · Set of pages a process recently used from time $t-\tau$ to t
- Trashing: · Total working sets => main memory

19. File system

- Virtual memory: Main memory
- File system: Hard-disk
- Access control matrix: · principles (users) x files
- Access control list: · stored with the file as metadata
  · Column of access control matrix
  · Read/write/Execute
- File is unstructured vector of bytes
- A directory/folder implements naming context
  · Binding "." to itself
  · Binding ".." to parent
  } in root, both are bound to root /
  · Is basically just a structured file
- open file: · direct access (randomly) or sequential access
- Memory mapped file: copied to main memory and accessed there
- Executable file: · just file with metadata "executable"

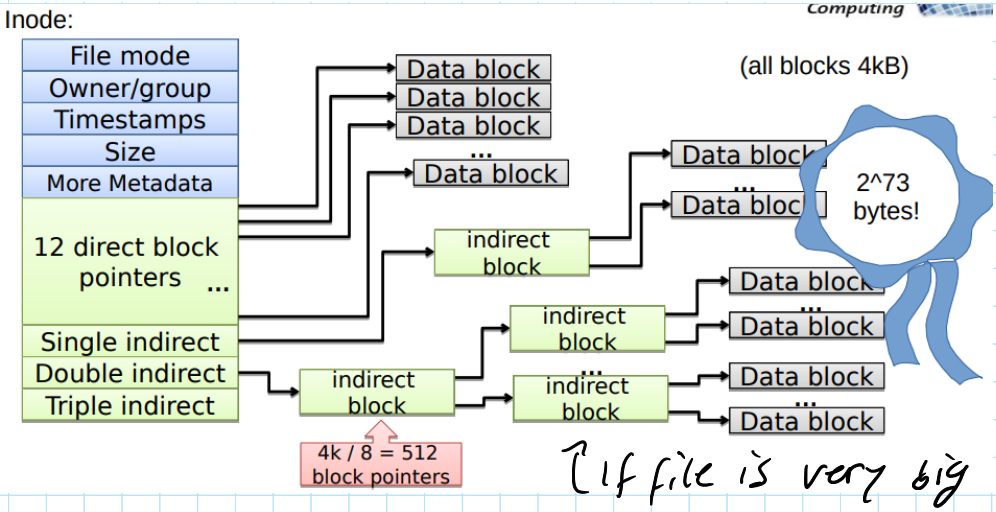
20. File system implementation

- Volume: · generic name for storage device
  · Consists of a number of fixed sized blocks
  · Logical block address treatens the volume as a linear array of blocks
- Partition: · Split up a volume
- Logical volumes: · Merge multiple physical devices into one big volume
- FAT file system: · File allocation table
  · FAT 32/64
  · exFAT
  · Has an entry for each block of volume
  · Markers for free/used
  · No access control
  · If file is too large for one block, make linked list
  · No access control
  · Little meta-data
  · Poor locality: files can get fragmented by linked list
  · Free space: linear search
  · Loose fat-table => everything gone
- FFS: · Fast filing system
  · Each file has an inode (=index node)
  · If file is small enough it is saved in inode
  · Inode is filled with list of addresses of blocks



File is represented by an *index block* or ***inode***

- File metadata
- List of blocks for each part of file
- Directory contains pointers to inodes

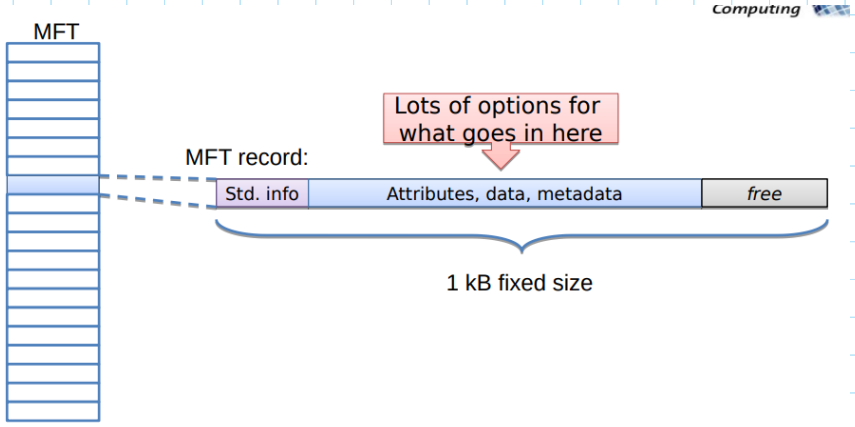


If file is very big

- Free blocks via bitmap

- Superblock: - Holds all information about layout of file system
  - Backups across the whole disk

- NTFS: - Master file Table
  - Each entry describes one file



- Attributes are variable
- Small files saved in the attributes

21. Network Stack

- **Multiplexing**: send packets from multiple connections to one connection at a lower level
- **Demultiplexing**: On lower level receive at one connection and demultiplex into multiple connections one level higher
  - ↳ Needs encapsulation/decapsulation headers to work
- State machines are needed for each connection which contains flow-control-window, congestion-control,...
  - ↳ Run in kernel because of timer-scheduling
- **Header space**:
  - receive/send set
  - represent all sets of possible headers of a packet
  - At each stage header-space gets smaller
- **Protocol graph**:
  - Represents what is going on in stack
  - Nodes: De/multiplexing, ICMP echo,...
  - ← demultiplexing | → multiplexing
  - Can have cycles: Tunnels
    - → need to capsulate IP-packet, twice
- **Packet descriptors**:
  - Avoid copying of packet data
  - Used for en/decapsulation
  - Simply add in front of linked list
- Receiver Side Scaling: Hardware: Hash function
- Flow Steering: Software: Table

22. Virtualization

- Hardware devices => system calls
- Interrupts => Signals
- Guest operating system: runs inside a virtual machine
- Hypervisor:
  - Runs on real, physical hardware
  - supports multiple virtual machines
  - Is an operating system itself
  - An OS can be extended to act as hypervisor
  - Type 1: functions as an OS Kernel
  - Type 2: Runs on top of a conventional OS
- Containers:
  - System-level virtualization
  - OS provides illusion of multiple containers of that OS
  - Limiting namespace
- Server consolidation: Running many services on one physical machine but many virtual machines
- Resource isolation: Performance hängt nicht von anderen VMs ab
- Trap and Emulate:
  - Run privileged code in user mode
  - Any kernel-mode cause trap to VM

- Strictly virtualizable: if OS can be perfectly emulated
- Not strictly virtualizable: · If instructions work both in user and kernel-mode but do something different depending on the mode
- Paravirtualization: · specialy modified OS
  · Critical calls are replaced with explicit trap instructions
  · Punch a hole
  · Hypercall
- Binary rewriting: · Patches Kernel on the fly
  · Rewrites unvirtualizable instructions
- MMU: (virtual) virtual address → (virtual) physical address → virtual address → physical address
- Directly writable page tables
  · (virtual) virtual address → physical address
  · Requires paravirtualization
  ↳ Need to check, if mapping only maps to VM-address space
  · All page-table-entries are read-only and can only be modified with hypercall

- Shadow page table: (Virtual) virtual address
  -> (virtual) physical address
  S.P.T -> physical address
- Nested paging: · Enhancement of hardware MMU
  · Allows it to translate through two page tables
- Physical memory allocation: · Fixed area of physical memory
  ↳ RAM
- Double paging: 1. Hypervisor pages out P to disk
  2. Guest OS does the same, but touches P
  3. Triggers page fault in hypervisor
  4. P is loaded into main memory
- Ballooning: · Reclaim memory from guest OS
  1. Installed on guest OS
  2. Hypervisor wants to reclaim memory
  3. Balloon process is inflated
  4. Returns addresses to hypervisor that these memory addresses can be moved to disk
  1. VM maps newly allocated main memory back to balloon
  2. Tells him to inflate

- Paravirtualized devices: virtual
- Device passthrough: mouse
- Give each VM an own MAC-address

---

7. Fault Tolerance & Paxos

- State replication ≙ Blockchain
  - ↳ If all nodes execute the same sequence of commands
- Naive approach: · Use one server as serializer
  · Master-slave
  · Serializer is single point of failure
- Two-phase locking: · Needs all servers to be alive
  1. Ask all servers for lock
  2. Send command to all servers
  3. Give lock back
- Ticket: Used with a counter to check if expired
- Paxos

> Idea: What if a server, instead of only handing out tickets in Phase 1, also notifies clients about its currently stored command? Then, u_2 learns that u_1 already stored c_1 and instead of trying to store c_2, u_2 could support u_1 by also storing c_1. As both clients try to store and execute the same command, the order in which they proceed is no longer a problem.

1. Client: c command to execute
t current ticket number

2. Client: · increases $t = t + 1$
· ask all servers with this ticket t

3. Servers: · if $t > T_{max}$
· $T_{max} = t$ ← I will not accept proposals numbered less than t
· reply with OK and previously stored command and his ticket number $ok(T_{store}, C_{store})$

4. Client: · If majority replies
· Pick largest T_{store} among all replies
· If $T_{store} > 0$ ← If I am the first one
· $c = C_{store}$
· send propose (t, c) to same majority

5. Servers: · if $t = T_{max}$
· $C = c$
· $T_{store} = t$
· send success

My command gets chosen

6. Client: If majority answers success
· send execute(c) to every server

↳ only one round of paxos to get to only one command to execute
↳ Paxos is able to not terminate

8. Consensus

- n nodes
- f nodes might crash
- n-f nodes are correct
- Agreement: All correct nodes decide for the same value
- Termination: All correct nodes terminate
- Validity: The decision value must be the input value of a node
- Configuration C: · Defined system at some point C
- Univalent C: If resulting decision is seattled
- Bivalent C: If we doesn't know resulting decision
- Configuration Tree: · leaves must be univalent
  · One path is one asynchronous execution of the algorithm
- Critical configuration: · If parent is bivalent
  · Childrens are univalent
- Randomized Consensus (= Ben-Or)
  · Either all nodes start with same input-bit which makes consensus easy, or nodes toss a coin until a large number of nodes get the same outcome

while true

- Broadcast v_i and round

1. Wait until a majority of myValue messages are received
2. If all contain same value → propose(v, round)
   else → propose(⊥, round)
3. Wait until majority of propose messages are received
4. If all messages proposes same value → decided = true
   → decide for v
5. If there is at least one proposal for v → $v_i = v$
6. else choose v_i randomly

↑ If all other send ⊥

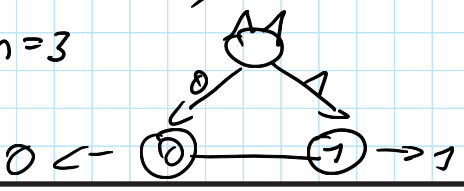
make more nodes seattle on v

→ slow, because of tossing u.a.r. coins

- shared coin: 1. choose 0 = 1/n else 1
  2. Broadcast myCoin
  3. Wait for n-f coins
  4. Make array out of it
  5. Send array to others
  6. Receive n-f arrays
  7. If we have a 0 among all elements return 0, else return 1

11. Byzantine Agreement

- Byzantine: Machine with misfunction
- Byzantine Agreement: Finding consensus with byzantine nodes involved
- f-resilent: Finding consensus with f byzantine nodes
- Correct-input validity: Decision value must come from a correct node
- All-same validity: If all correct nodes start with same value, decided value must be starting value
- Median validity: Accept a value close to median
- Byzantine agreement with one byzantine-node
  1. Send value to all other nodes
  2. Receive values from all other nodes
  3. Store all values in a set
  4. Send set to all other nodes
  5. Receive sets from all other nodes
  6. Pick smallest value that was twice in all the sets
  - ↳ One time through the correct node
    One time through another correct node via forwarding
  - Provides only any-input validity
  - Why it doesn't work with n=3



- Synchronous algorithm needs $f+1$ rounds if we pick the minimum seen value
- Asynchronous Byzantine Agreement
  1. send (input bit, round)
  2. repeat
     3. wait for $n-f$ messages
     4. if at least $n/2+3f+1$ messages have same value $\Rightarrow$ decided true
     5. if at least $n/2+f+1$ messages with same value $\Rightarrow x_i = x$
     6. Else: choose x_i at random
        ↳ slows algorithm down
     7. send (bit x, round $+1$)
  3. until decided = true

---

12. Broadcast & shared Coins

- Instead of every node throwing a local coin, the nodes throw a shared coin
- Naive Bitstring: $n/2+f+1$ nodes $1 \to 1 \to 1$
  $n/2-f-1$ nodes $0 \to 0 \to 0$
  } No Termination
  ↳ Naive Bitstring 0000
- Shared Blackboard: · Trusted authority
  · Nodes can read/write on Blackboard
  1. Local coin ± 1
  2. Write it on the blackboard
  3. if $|c| \geq n^2$
  4. return sign(sum(c))

- Central limit theorem: · $X_1, X_2, \dots X_n$ sequence of random variables with same distribution
  · $n \to \infty$: Normal distribution

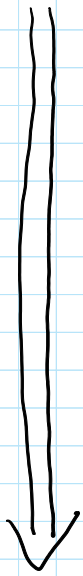
- Reliable Broadcast: · Byzantine Nodes can't send different messages to different nodes

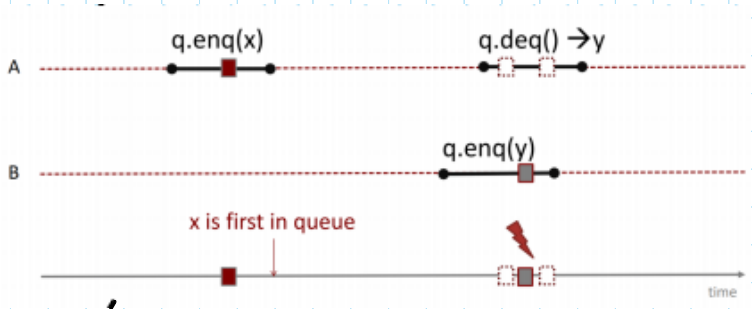
FIFO ← Numbering

- Shared Coin with cryptography
  1. Each node has own shared key
  2. Sign message with current round
  3. Compute hash for all received messages
  4. Take least significant bit of smallest hash

13. Consistency & Logical Time

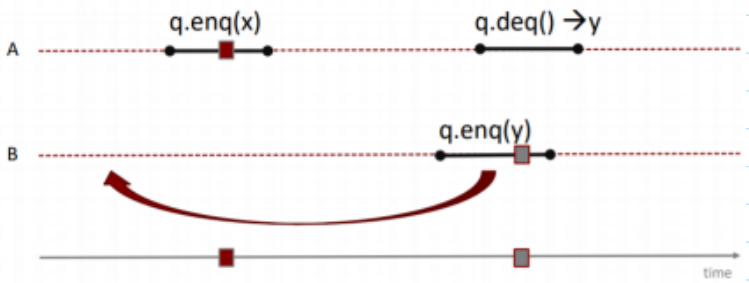
- Linearizable: · If we can place linearizability points from different processors on a timeline such we have a sequential history





↳ Not linearizable

- Sequential Consistency: · History must be sequential consistent only per processor



- $f \to g$: f happened before g
- Logical clock: $g \to h \Rightarrow C_u(g) < C_v(h)$ (local clock)
- Strong logical clock: $C_u(g) < C_v(h) \Rightarrow g \to h$ (shared clock)

19. Time, Clocks & GPS

- Clock error/skew: error between two clocks
- Drift: predictable clock error
- Jitter: unpredictable clock error
- NTP forest: Two devices who communicate may receive the time from different branches → error
- Time to first fix is high, because due to limited data rate, function of the satellite position over time is sent every 30 seconds
- 4 unknowns = • x, y, z
  • Time offset from system time

1. GPS Satelite sends: PRN-Data
2. Receiver detects delays by acquisition
   - 2.1 Stretch to ratio r
   - 2.2. For all doppler shifts
   - 2.3. For all delays d
   - 2.4. Shift PRN by d
   - 2.5. choose the delay d^* that maximises circular-shift
3. Track signals and decode navigation data
4. Solve linear system of equation

↳ With receiver delay, time can be synchronized

- Collective detection

1. Receive 1ms of raw GPS data
2. For all hypothesis (guesses where I am)
    2.1. For all satellites visible within hypothesis
    2.2. $r = r + r_i$ where r_i is the expected signal of satellite i
    2.3. Probability P_n = corralation between |raw signal - r|
3. Choose hypothesis with maximum corralation

Access strategy: z — one Quorum: Q
$P_z(Q)$ = Probability — Quorum system: S

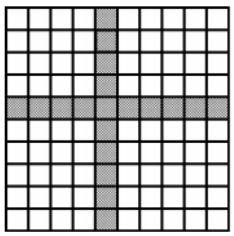
15. Quorum Systems

- Paxos consensus ⇒ Need to ask majority of servers
  ↳ inefficient
- **Quorum system**: · Set of quorums such that every two quorums intersect

1. Client selects a quorum
2. Aquires quorum lock/ticket

- **Singleton**: One quorum
- **Load**: · Access strategy on node v_i is the sum of all probabilities accessing that node
  ↳ $L_z(v_i) = \sum_{Q \in S,\, v_i \in Q} P_z(Q)$
  · Induced by access strategy is the maximal load on a node
  ↳ $L_z(S) = \max_{v_i \in S} L_z(v_i)$
  ↳ Quorum system
  · Quorum system is the access strategy with the minimum load
  ↳ $L(S) = \min_z L_z(S)$
- **Work**: · Of a quorum is number of its nodes
  ↳ $W(Q) = |Q|$
  · Of access strategy is expected number of nodes accessed
  ↳ $W_z(S) = \sum_{Q \in S} \overbrace{P_z(Q)}^{Pr[x]} \cdot \overbrace{W(Q)}^{x}$
  · Of quorum system
  ↳ $W(S) = \min_z W_z(S)$

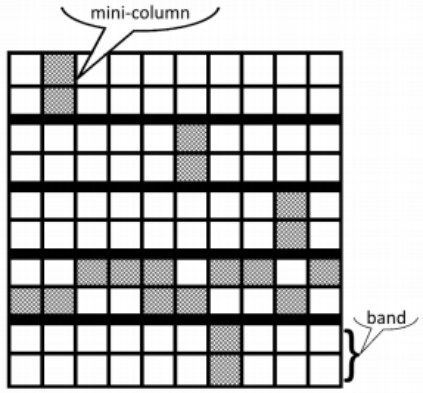
- Grid quorum system



  - $\sqrt{n}$ different quorums
  - Size of each quorum: $2\sqrt{n} - 1$ (overlap)
  - Could get into a deadlock
    - ↳ Avoid by ordering locking, if one is already locked, unlock all
- Resilient: if a quorum system can fail such there is still a single quorum without failed nodes
  - Grid quorum has $\sqrt{n}-1$ resilience
- Failure probability: Probability that at least one node of every quorum fails
- Asymptotic failure: if $n \to \infty$
- Grid: optimal load $\to 1/\sqrt{n}$
  - no fault-tolerance → asymptotic failure: 1
- Majority: fault-tolerance → asymptotic failure: 0
- B-grid system

Load
work
Resilience
failure

Singleton	Majority	Grid	B-grid

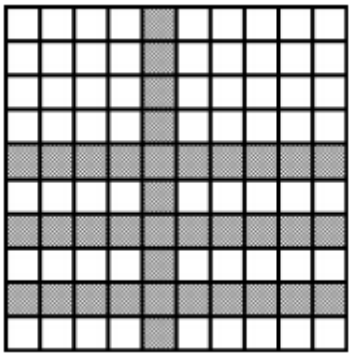


	Singleton	Majority	Grid	B-Grid*
Work	**1**	$> n/2$	$\Theta(\sqrt{n})$	$\Theta(\sqrt{n})$
Load	1	$> 1/2$	$\Theta(\mathbf{1}/\sqrt{\mathbf{n}})$	$\Theta(\mathbf{1}/\sqrt{\mathbf{n}})$
Resilience	0	$< \mathbf{n/2}$	$\Theta(\sqrt{n})$	$\Theta(\sqrt{n})$
F. Prob.**	$1-p$	$\to \mathbf{0}$	$\to 1$	$\to \mathbf{0}$

→ In each band a mini-column must fail

or one element of every mini-column in a band must fail

- Byzantine: We need to ensure that every intersection contains a correct node
- f-disseminating: · every intersection contains f+1 nodes
  ↳ Need to verify data
- f-masking: · every intersection contains 2f+1 nodes
  ↳ No need for verification
- f-masking grid
  · contains one full column and f+1 rows



16. Eventual Consistency & Bitcoin

- CAP theorem:
  - consistency
  - Availability
  - Partition Tolerance

  } choose two of the three

- Available + Partition tolerant ATM
  1. If Bank reachable synchronize local view of balances
  2. Process ATM withdraw

  ↳ Eventual Consistency
  - Weak consistency
  - During a partition, different updates may conflict with each other
  - Conflict resolution mechanism required

- Bitcoin
  - Transaction input: tuple consisting of a reference to previously created output and signature to spend output
  - UTXO: set of unspent transaction outputs
  - Fee is used for transaction processing
  - Unconfirmed new transactions are added to the memory pool
  - Block: - data structure used for incremental changes
    - consists of list of transactions
  - Genesis block: root of tree

- finding a block impose the transactions in its local memory pool to all others
- Blockchain: Longest path from genesis block to current block
  - All nodes eventually agree on blockchain

- Monotonic Read consistency: if u has seen a value it never sees any older value
- Monotonic Write consistency: sequential consistency
- Read-your-write consistency: After writing x you don't see any older value

23. Game Theory

- Social optimum: minimizes the sum of all costs
- Dominant strategy: Best choice which I should choose regardless of the other player
- Nash Equillibrium: A strategy in which no other player can improve by changing its strategy
  - can have multiple nash equilibria
- Nash equilibrium for selfish caching:
  1. $S = \{\}$
  2. Repeat
     - 2.1. Choose node with highest demand out of V
     - 2.2. $S = S + \{v\}$
       $V = V - v$
     - 2.3. Remove every node u from V with $c_{u \leftarrow v} \leq 1$
- Price of anarchy: $\frac{\text{cost(Nash equilibrium with highest cost)}}{\text{cost(social optimum)}}$
- Optimistic price of anarchy: $\frac{\text{cost(Nash equilibrium with lowest cost)}}{\text{cost(social optimum)}}$

- Mixed nash equilibrium: choose strategy with probability

24. Distributed storage

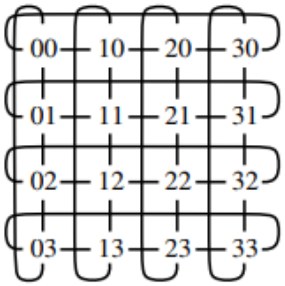
- Consistent hashing

1. Hash each file-name with k hashing functions: $[0, 1)$
2. Hash IP-address with same one hash function: $[0, 1)$
3. Store a copy of file x if $h_i(x) \approx h(IP)$ on node IP

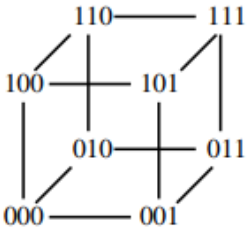
- Each server stores: $\frac{k \cdot m}{n}$ files (k hashes, m files, n servers)

- Churn: Nodes entering and leaving the network frequently
- Tree is not good => fat tree

$M(m,1)$

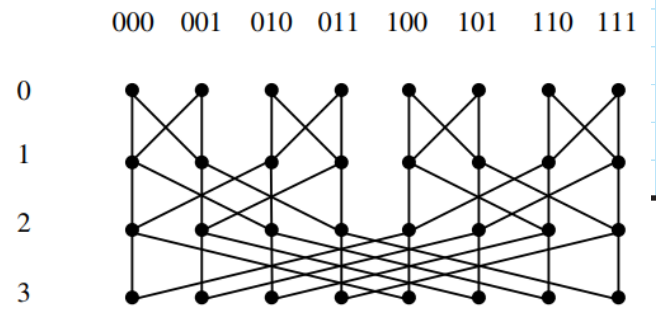


$T(4,2)$

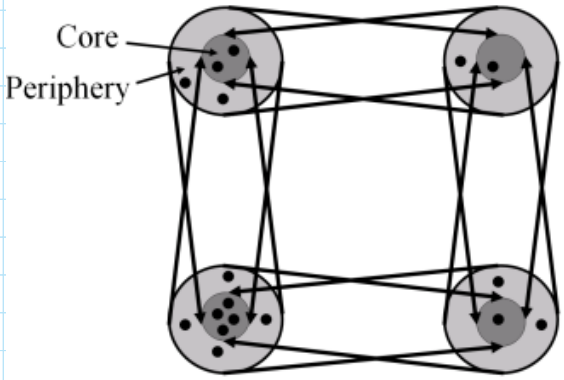


$M(2,3)$

- $M(2,d) = T(2,d)$: 
- Butterfly network



- Distributed hash table with churn

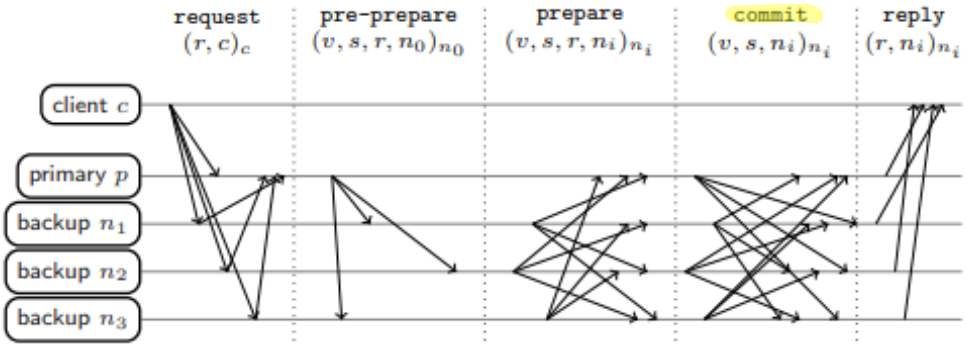


25. Authenticated Agreement

- With cryptography, byzantine lies may be detected easily
- $msg(x)_u$ = message x signed by node u

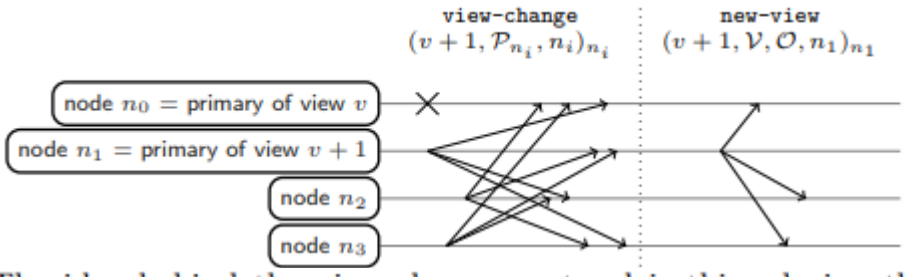
PBFT

- If a backup defect faulty behavior in the primary, they start a new view and the next node (round robin) becomes the primary (=view change)



Note that the agreement protocol can run for multiple requests in parallel. Since we are in the variable delay model and messages can arrive out of order, we thus have to wait in Algorithm 25.17 Line 4 until a request has been executed for all previous sequence numbers.

The client only considers the request to have been processed once it received $f + 1$ `reply`-messages sent by the nodes in Algorithm 25.17 Line 5. Since a correct node only sends a `reply`-message once it executed the request, with $f + 1$ `reply`-messages the client can be certain that the request was executed by a correct node.



- The idea behind the view change protocol is this: during the view change protocol, the new primary gathers prepared-certificates from $2f + 1$ nodes, so for every request that some correct node executed, the new primary will have at least one prepared-certificate.

26. Advanced Blockchain

- Selfish mining: keep blocks secret to get the others doing nonsense work

1. If public blockchain has grown by 1
2. Publish secret block and mine on this block
   ↳ power of other miners distribute

- DAG-blockchain
  - Be able to have two parents
  - Every block gets a weight
  - What is the most important parent
    - Find common anchestor
    - Better parent, more anchestors
  - Transactions of not priority parents aren't lost, if they don't contradict some other transaction of other parents

1. Client makes shared account with rollback timelock
2. Server must sign that
3. Broadcast to Blockchain

} Initialisation

---

4. Client makes transactions, signs it
5. Client sends transaction to server
6. Server stores transaction
7. After business is done, server releases last transaction to Blockchain with shortest time