Lecture 2: Virtual Memory.

Segmentation. Logical address = segment number + offset. The MMU checks if the offset is within the limit before translating to a physical address. Out of limit $\rightarrow$ segmentation fault. May cause external fragmentation.

Paging. Page memory into fixed-size chunks (usually 4 KiB). Logical address = page number + offset. Bilevel and multi-level paging reduces page table size, but need multiple memory accesses.

Inverted page table. Physical $\rightarrow$ virtual. Implemented as a hash table in hardware. Reduced size (no need to support huge virtual address space), increased complexity.

Table 1: Comparison of Address Translation Methods

Method	Advantages	Disadvantages
Segmentation	Fast context switching; meaningful protection for compilers.	Suffers from external fragmentation.
Paging (Single-Level)	No external fragmentation; fast and easy allocation.	Large table size, proportional to virtual memory.
Two-Level Paging	Manages sparse address spaces efficiently.	Multiple memory references per access (mitigated by TLB).
Inverted Table	Table size is proportional to physical memory.	Hash function and management are complex to implement efficiently.

Demand paging. On page faults, write back replaced page if dirty. Which page to replace?

Effective access time. EAT $= (1-p) \times t_{\text{hit}} + p \times t_{\text{fault}}$, where p is the page fault rate.

Bélády's anomaly. FIFO may increase page faults when increasing number of frames.

Second chance. A queue of pages. On replacement, always check the front; if reference bit is 1, set to 0 and move to back of queue; if 0, replace. The reference bit is set to 1 on access.

Clock algorithm. Use a circular list and a reference bit. On replacement, if reference bit is 1, set to 0 and move on; if 0, replace.

N-th chance. Give each page N chances before replacement by a counter. Typically $N = 2$ for dirty pages and $N = 1$ for others.

Clustering. On a page fault, also load adjacent pages.

Performance thrashing. If the working set size > number of frames, high page fault rate $\rightarrow$ low CPU utilization $\rightarrow$ more page faults. Hence performance actually degrades with too high degree of multiprogramming.

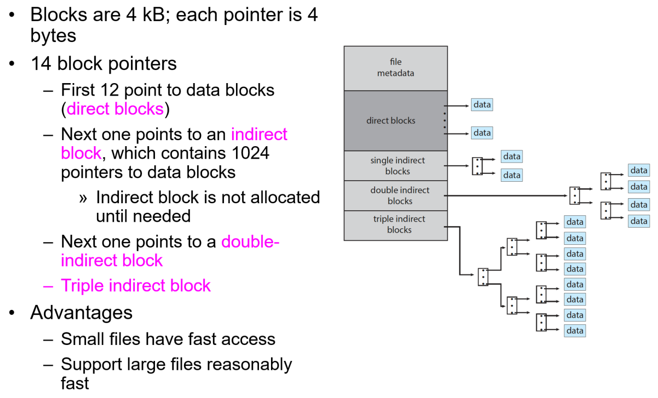
File system buffer caching. Use free frames to cache file system data. On page replacement, evict pages not in the cache first. Delay writes to disk by caching writes in memory and writing them back periodically or when the cache is full. True LRU is preferable since disk accesses are much less frequent.

Lecture 3: File System Design.

Open file table. Contains entries for each open file, including file pointer, mode, and reference count. One per process and one system-wide.

BSD file system. Every file or directory has an inode containing metadata (type, permissions, owner, size, timestamps) and pointers to data blocks. When fetching a file, do root (inode 2) $\rightarrow$ root block $\rightarrow$ next directory inode $\rightarrow$ next directory block $\rightarrow \cdots \rightarrow$ target file inode $\rightarrow$ target file blocks. The data block of a directory contains entries of (filename, inode number) pairs.

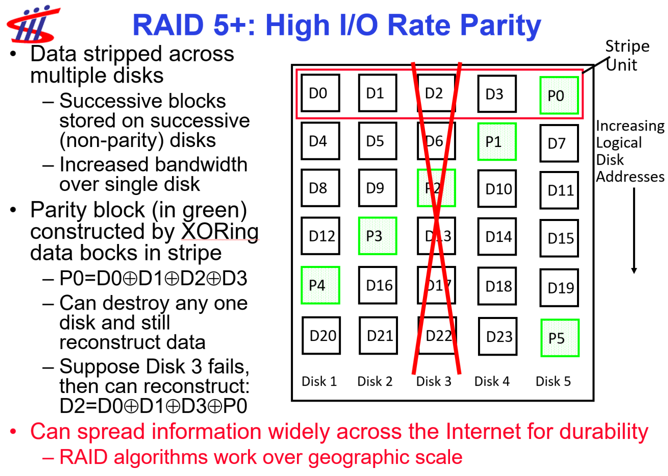
FAT file system. A linked list of blocks, with a file allocation table mapping each block to the next block in the file or an end-of-file marker. Simple but inefficient for large files.



- Blocks are 4 kB; each pointer is 4 bytes
- 14 block pointers
  - First 12 point to data blocks (direct blocks)
  - Next one points to an indirect block, which contains 1024 pointers to data blocks
    - » Indirect block is not allocated until needed
  - Next one points to a double-indirect block
  - Triple indirect block
- Advantages
  - Small files have fast access
  - Support large files reasonably fast

FFS (BSD 4.2). Divide volumes into cylinder groups, each containing its own inodes, data blocks, and a bitmap for allocation. Reserve some space (10%) to ensure contiguous allocation.

RAID. Redundant Array of Independent Disks. RAID 1: mirroring. RAID 5: block-level striping with distributed parity. RAID 6: double distributed parity.



RAID 5+: High I/O Rate Parity

- Data stripped across multiple disks
  - Successive blocks stored on successive (non-parity) disks
  - Increased bandwidth over single disk
- Parity block (in green) constructed by XORing data bocks in stripe
  - P0=D0⊕D1⊕D2⊕D3
  - Can destroy any one disk and still reconstruct data
  - Suppose Disk 3 fails, then can reconstruct: D2=D0⊕D1⊕D3⊕P0
- Can spread information widely across the Internet for durability
  - RAID algorithms work over geographic scale

Transactional File Systems. Better reliability. Journaling file systems: apply system metadata updates using transactions; non-directory files may be updated in-place. Full logging file systems: all changes are treated as transactions.

The ACID properties. Atomicity, Consistency, Isolation, Durability.

LFS. Log-structured file system. Treat the disk as a circular log and write all modifications sequentially to the end of the log.

Basic idea. Temporal locality $\rightarrow$ write-friendly; less read-friendly, but can be compensated by caching.

Log retrieval. Maintains an inode map to locate inodes (gets written like logs), and a map of inode map whose location is fixed (used in crash recovery).

Garbage collection. Most implementations avoid purely circular logs and divide up their storage into MiB-size segments. The head of the log simply advances into non-adjacent segments which are already free. If space is needed, the least-full segments are reclaimed first. This decreases the I/O load of garbage collection, but becomes increasingly ineffective as the file system fills up.

Cleaning cost. When cleaning a segment with u fraction of live data, the cost per byte of live data is $2/(1-u)$.

NTFS. Uses B+ trees to store file system metadata. Each file has a master file table (MFT) entry containing attributes (metadata, data runs, etc.). Supports journaling for metadata updates.

MFT. Each file has at least one MFT entry (1 KiB). Small files may be stored directly in the MFT entry (resident data). Larger files use extents (data runs) to point to clusters on disk.

Lecture 4: Scheduling and Queueing.

CPU burst length prediction. Use exponential averaging: $S_{n+1} = \alpha T_n + (1-\alpha)S_n$, where S_{n+1} is the predicted length of the next CPU burst, T_n is the actual length of the current CPU burst, and α is a parameter ($0 \le \alpha \le 1$).

Multi-level feedback scheduling. Multiple RR queues with different priority levels and time quanta. A process starts in the highest priority queue and is demoted to a lower priority queue if it uses up its time quantum. A process that waits too long in a lower priority queue may be promoted to a higher priority queue (aging).

Lottery scheduling. Each process has a number of lottery tickets. The scheduler randomly selects a ticket to determine which process to run next. High priority queues $\rightarrow$ more tickets per process.

Real-time scheduling. Hard real-time: missing a deadline is catastrophic. Soft real-time: missing a deadline degrades performance.

EDF. Earliest Deadline First. Optimal for hard real-time scheduling on a uniprocessor.

Response time and utilization. For major scheduling algorithms, the response time is linear to utilization up to a certain threshold, after which it skyrockets rapidly.

Queueing. Response time = queueing time + service time.

Burstiness of arrival. Usually memoryless: arrival time follows exponential distribution; number of arrivals in a time interval follows Poisson distribution.

Service time distribution. Squared coefficient of variance $C = \sigma^2/\mu^2$. $C = 1$ for exponential distribution; $C = 0$ for deterministic distribution. Typical disk response time $C \approx 1.5$.

Queue parameters. λ: arrival rate; T_{ser}: mean service time; $\mu = 1/T_{\text{ser}}$: service rate; $\rho = \lambda/\mu$: utilization; T_{q}: mean queueing time; L_{q}: mean queue length; C: squared coefficient of variance of service time. By Little's Law, $L_{\text{q}} = \lambda T_{\text{q}}$.

Queueing time. For a queue with one server, Markovian arrivals, general service time distribution (M/G/1):

$$T_{\text{q}} = \frac{\rho}{1-\rho} \cdot \frac{1+C}{2} \cdot T_{\text{ser}}. \tag{4.1}$$

If service time is also Markovian (M/M/1), then

$$T_{\text{q}} = \frac{\rho}{1-\rho} \cdot T_{\text{ser}}. \tag{4.2}$$

Lecture 5: Time in Distributed Systems.

Cristian's algorithm. A client requests time from a time server. The server responds with its current time. The client sets its clock to the received time plus half the round-trip time (RTT).

Accuracy. RTT/2 − minimum one-way delay.

NTP. Network Time Protocol. Hierarchical structure of time servers (stratum 0: atomic clocks; stratum 1: directly connected to stratum 0; stratum 2: connected to stratum 1 and 2; stratum 3: connected to any).

One-round NTP exchange. Client sends request at (clinet time) t_0; server receives at t_1 (server time); server sends response at t_2; client receives at t_3. Then time adjustment $= (t_1 - t_0 + t_2 - t_3)/2$.

Multiple-round NTP exchange. Sample 8 times and choose the one with the minimum RTT $(t_1 - t_0 + t_3 - t_2)$.

The Berkeley algorithm. No authority server. A master polls other machines for their times, computes an average, and instructs each machine to adjust its clock accordingly. Change the update rate instead of setting the time directly to avoid log jumps.

Logical clock. Does not care the actual time, only the order of events.

Lamport clock. Each process maintains a logical clock (integer counter). On each event, increment the counter by 1. On sending a message, include the counter value. On receiving a message, set the counter to max(local counter, received counter) + 1.

Total ordering. To totally order events, use (logical clock, process ID) pairs and compare lexicographically.

Vector clock. Each process maintains a vector of counters, one for each process. On each event, increment its own counter by 1. On sending a message, include the vector. On receiving a message, update each entry to the maximum of the local and received vectors, then increment its own counter by 1.

Lecture 6: Mutual Exclusion.

Totally-ordered multicasting. Every process maintains a local queue of messages ordered by (timestamp, process ID). 1. Sender timestamps the message with its logical clock and multicasts it to all processes (including itself). 2. On receiving a message, a process enqueues it and sends an acknowledgment back to the sender. 3. A process can deliver a message when (a) it has received the message and (b) it has received acknowledgments from all processes for messages with smaller timestamps.

Centralized algorithm. Coordinator maintains a queue. If lock is occupied, put requesting processes into queue. When lock is released, give the lock to the first one in the queue. No coordinator fault tolerance.

Leader election: the Bully algorithm. Each process has a unique ID. A process that detects the leader's failure initiates an election by sending an `election` message to all processes with higher IDs. If no one responds, it becomes the leader and sends a `coordinator` message to all processes. If a higher-ID process receives an `election` message, it responds with an `OK` message and starts its own election.

Fully decentralized algorithm. Have n coordinators instead of one. A requesting process must get permission of $m > n/2$ coordinators before entering critical section.

Lamport's mutual exclusion. Based on totally-ordered multicasting. To request the lock, multicast a `request` message with timestamp to all processes. On receiving a `request` message, enqueue it and send an acknowledgment back. To release the lock, multicast a `release` message. On receiving a `release` message, remove the corresponding `request` from the queue. A process can enter critical section when its own `request` is at the front of the queue and it has received acknowledgments from all processes for messages with smaller timestamps.

Ricart-Agrawala algorithm. When node i wants to enter critical section, it multicasts timestamped request to all nodes. These other nodes reply (eventually). When node i receives $n-1$ replies, it enters the critial section. Trick: node j having earlier request does not reply to node i until it has exited the critical section.

The token ring algorithm. Arrange processes in a logical ring. A special message called a token circulates around the ring. A process can enter critical section only when it has the token. After exiting, it passes the token to the next process in the ring.

Table 2: Comparison of Distributed Mutual Exclusion Algorithms

Algorithm	# Messages per cycle	Delay before entry	Problems
Centralized	3	2	Coordinator Crash, Bottleneck
Decentralized	2mk + m, ($k \ge 1$)	2mk	Starvation, Unbounded messages
Lamport	$3(N-1)$	$2(N-1)$	Crash of any process
Ricart & Agrawala	$2(N-1)$	$2(N-1)$	Crash of any process
Token Ring	1 to ∞	0 to $(N-1)$	Lost token, process crash

Lecture 7: Consensus.

Failure models. Fail-stop: process halts and stays halted. Byzantine: process behaves arbitrarily (including maliciously).

The CAP theorem. In a distributed system, it is impossible to simultaneously provide more than two out of the following three guarantees: Consistency, Availability, and Partition tolerance.

Data Replication. Read-only is simple; focus on read-write.

$\boxed{\textit{Primary-backup.}}$ One primary replica handles all writes; backups replicate from the primary. Must wait for all backups to acknowledge before replying to client.

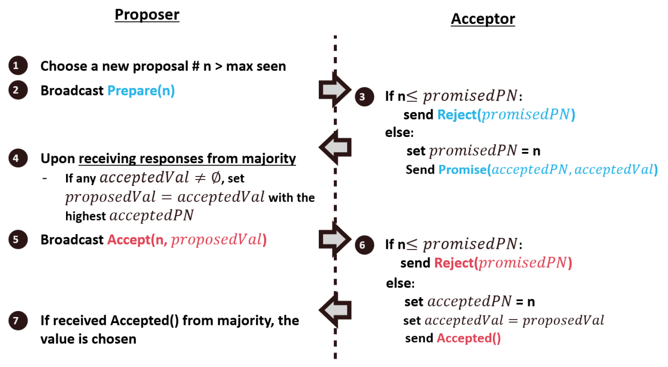
$\boxed{\textit{Quorum-based.}}$ Assume replication across N nodes. A write is successful if it is written to at least W nodes; a read is successful if it reads from at least R nodes. To ensure consistency, must have $W + R > N$. This avoids dependency on the slowest node.

$\boxed{\textit{Replicated state machine model.}}$ A distributed system as a single state machine replicated across multiple nodes which execute exactly same operations. Implementation requires consensus algorithms.

Fischer-Lynch-Patterson (FLP) impossibility. In an asynchronous distributed system, it is impossible to achieve consensus if at least one process may fail.

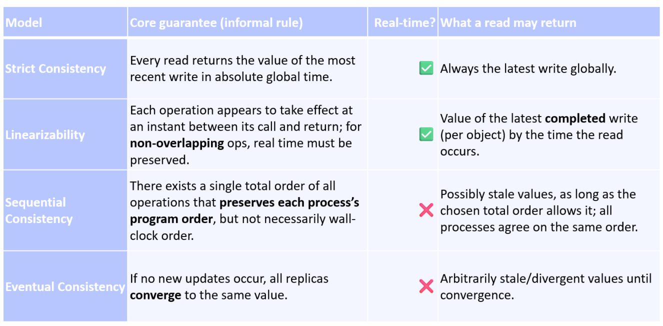
Properties for correct consensus. Termination: every non-faulty process eventually decides on a value. Agreement: no two non-faulty processes decide on different values. Integrity: all deciding processes select a "right" (non-trivial) value.

Paxos. Guarantees safety (agreement and integrity) but not liveness (thus CP). Two roles: proposers and acceptors. Can be used in replicated state machines (decide each log entry separately).


Proposer
Acceptor
1 Choose a new proposal # n > max seen
2 Broadcast Prepare(n)
3 If n≤ $promisedPN$:
send Reject($promisedPN$)
else:
set $promisedPN$ = n
Send Promise($acceptedPN, acceptedVal$)
4 Upon receiving responses from majority
- If any $acceptedVal \neq \emptyset$, set $proposedVal = acceptedVal$ with the highest $acceptedPN$
5 Broadcast Accept(n, $proposedVal$)
6 If n≤ $promisedPN$:
send Reject($promisedPN$)
else:
set $acceptedPN$ = n
set $acceptedVal = proposedVal$
send Accepted()
7 If received Accepted() from majority, the value is chosen

2PC. Two-phase commit protocol. One coordinator process and several participant processes. Phase 1 (voting): coordinator sends `prepare` to all participants; participants respond with `vote-commit` or `vote-abort`. Phase 2 (commit/abort): if all votes are `vote-commit`, coordinator sends `global-commit`; otherwise, sends `global-abort`. Participants execute accordingly. Vulnerable to coordinator failure.

Consistency models. Strict consistency (strongest), linearizability, sequential consistency, and eventual consistency (weakest).



Model	Core guarantee (informal rule)	Real-time?	What a read may return
Strict Consistency	Every read returns the value of the most recent write in absolute global time.	✅	Always the latest write globally.
Linearizability	Each operation appears to take effect at an instant between its call and return; for **non-overlapping** ops, real time must be preserved.	✅	Value of the latest **completed** write (per object) by the time the read occurs.
Sequential Consistency	There exists a single total order of all operations that **preserves each process's program order**, but not necessarily wall-clock order.	❌	Possibly stale values, as long as the chosen total order allows it; all processes agree on the same order.
Eventual Consistency	If no new updates occur, all replicas **converge** to the same value.	❌	Arbitrarily stale/divergent values until convergence.

Lecture 8: Blockchains.

Data structure. A blockchain is a linked list of blocks, where each block contains a list of transactions and the hash of the previous block.

$\boxed{\textit{Hash pointers.}}$ Serves as both a pointer to the previous block and a cryptographic hash of its contents, ensuring integrity.

$\boxed{\textit{Merkle trees.}}$ Each block contains a Merkle root, which is the root of a binary hash tree constructed from the transactions in the block.

The peer-to-peer (P2P) network. A new node queries a "DNS Seed" to get a list of IP addresses of existing nodes. It then connects to a few neighbors to join the mesh. When a user creates a transaction, they broadcast it to their neighbors, who validate it and propagate it further.

The Nakamoto consensus. Nodes (miners) compete to add the next block by solving a Proof of Work puzzle. The longest valid chain is considered the canonical blockchain.

$\boxed{\textit{The Sybil attack.}}$ An attacker creates many fake identities to gain disproportionate influence in the network. Mitigated by requiring proof of work or stake to create new identities.

$\boxed{\textit{Proof of Work (PoW).}}$ Miners compete to solve a computational puzzle (finding a nonce such that the block's hash meets a certain difficulty target). The first to solve it gets to add the next block and receives a reward.

$\boxed{\textit{Difficulty adjustment.}}$ The network adjusts the difficulty of the puzzle periodically (e.g., every 2016 blocks in Bitcoin) to maintain a consistent block time (e.g., 10 minutes).

$\boxed{\textit{The longest chain rule.}}$ In case of forks, nodes follow the longest valid chain (the one with the most cumulative proof of work).

$\boxed{\textit{k-delay confirmation.}}$ A transaction is considered confirmed after k blocks have been added on top of the block containing it (e.g., $k = 6$ in Bitcoin) to avoid forking and mitigate double-spending attacks.

Ethereum and smart contracts. Ethereum extends the blockchain concept by introducing "contract accounts" with smart contracts, which are self-executing contracts with the terms directly written into code. Users can send transactions to these contracts to trigger functions and change their state.

The replay attack. An attacker intercepts a valid data transmission and fraudulently retransmits it. Mitigated by including nonces in transactions to ensure uniqueness.

Lecture 9: Fault Tolerance.

The Byzantine generals theorem. To tolerate f Byzantine faults, a system must have at least $3f + 1$ nodes.

PBFT. Practical byzantine fault tolerance. Works in an asynchronous network with up to f Byzantine faults among $3f + 1$ replicas.

1. **Pre-prepare:** The primary (leader) node receives a client request, assigns it a sequence number n, and broadcasts a `PRE-PREPARE` message to all other (replica) nodes. This phase ensures a proposed ordering of requests.
2. **Prepare:** Upon receiving a valid `PRE-PREPARE` message, each replica broadcasts a `PREPARE` message to all other nodes, signaling its agreement on the proposed sequence number. A replica enters the "PREPARED" state once it has received $2f+1$ matching and valid messages (including its own). This quorum size is critical: in a system of $3f+1$ nodes, any two quorums of size $2f+1$ are guaranteed to have an intersection of at least $f+1$ nodes. This overlap ensures at least one honest node is common to both sets, which prevents the system from reaching a "PREPARED" state for two conflicting proposals for the same sequence number, thus guaranteeing safety.
3. **Commit:** Once a replica is in the "PREPARED" state, it broadcasts a `COMMIT` message. A replica enters the final "COMMITTED-LOCAL" state and executes the request after it has collected $2f + 1$ matching `COMMIT` messages. This final phase confirms that a sufficient quorum of nodes has seen proof that the network is prepared to commit the request.

Consortium blockchain. A type of blockchain where consensus is controlled by a pre-selected set of nodes (a consortium). It simplifies consensus by establishing node identity but requires algorithms that can handle malicious nodes, like PBFT.

Quantifying availability. Availability = MTBF / (MTBF + MTTR). MTBF: mean time between failures. MTTR: mean time to recover.

Software faults. Bohrbugs: deterministic, repeatable. Heisenbugs: transient, non-deterministic.

Lecture 10: Security.

Public blockchain challenges. centralization (mining pools), etc.

$\boxed{\textit{Blockchain DoS (BDoS).}}$ An attack where a miner finds a block but only broadcasts the header, effectively stalling the network as rational miners will not risk mining a block that can be easily orphaned.

$\boxed{\textit{Miner-extractable value (MEV).}}$ The maximum value a miner can capture from reordering, inserting, or censoring transactions within the blocks they produce.

$\boxed{\textit{Sandwich attack.}}$ A specific MEV strategy where an attacker sees a user's pending trade, places a large trade immediately before it (front-running) to drive up the price, and then places a trade immediately after it (back-running) to sell at the new price, profiting from the user's price slippage.

Access control matrix (ACM). Rows represent subjects (users/processes); columns represent objects (files/resources). Each entry specifies the access rights of a subject to an object.

$\boxed{\textit{Access control list (ACL).}}$ Column slicing. Each object has a list of subjects and their access rights.

$\boxed{\textit{Capability list.}}$ Row slicing. Each subject has a list of objects and their access rights.

Table 1: Comparison: ACLs vs. Capability Lists

Feature	Access Control List (ACL)	Capability List
Storage	With the Object (File)	With the Subject (User/Process)
Analogy	Guest List	Car Key / Ticket
Delegation	Hard (Must update the list)	Easy (Pass the key)
Revocation	Easy (Remove from list)	Hard (Must invalidate the key)
Usage	File Systems, Firewalls	Distributed Systems, Tokens

Buffer overflow attack. An attacker inputs data that exceeds the allocated buffer size, overwriting adjacent memory (especially the return address) to inject malicious code.

Worms. Self-replicating malware that spreads across networks without user intervention.

$\boxed{\textit{Spreading.}}$ Typically, the portion of infected machines obeys the logistic growth model $f = (e^{kt} - 1)/(e^{kt} + 1)$, where k is the spreading rate.

Denial of Service (DoS). An attacker overwhelms a target system with excessive requests, rendering it unavailable to legitimate users.

$\boxed{\textit{The SYN flood attack.}}$ An attacker sends a large number of TCP SYN requests with spoofed IP addresses to exhaust the server's connection table. The server allocates resources for each half-open connection, leading to resource exhaustion. Defense: SYN cookies. The server encodes the connection information into the sequence number of the SYN-ACK response and does not allocate resources until the final ACK is received.

$\boxed{\textit{Reflection DoS.}}$ An attacker sends requests with a spoofed source IP address (the victim's address) to a third-party server, which then sends responses to the victim, overwhelming it with traffic.

$\boxed{\textit{Distributed DoS (DDoS).}}$ An attacker uses multiple compromised machines (botnet) to launch a coordinated DoS attack on a target system.

Secure channels. Requirements: authentication, confidentiality, integrity, and availability.

Key distribution. Symmetric encryption is fast but requires a shared secret key.

5.1.1 Approach 1: Key Distribution Center (KDC)

A trusted server (e.g., Kerberos) knows a secret key for every user.

- Alice asks KDC: "I want to talk to Bob."
- KDC generates a session key K_{AB}.
- KDC encrypts K_{AB} with Alice's key (for Alice) and Bob's key (for Bob).
- **Problem:** The KDC is a single point of failure and a high-value target.

5.1.2 Approach 2: Public Key Infrastructure (PKI)

We use asymmetric cryptography (Public/Private keys).

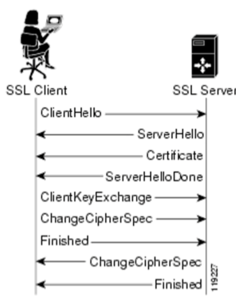
- Alice encrypts a message using Bob's **Public Key**.
- Only Bob can decrypt it using his **Private Key**.

The Problem: How does Alice know the Public Key she found really belongs to Bob? An attacker (Mallory) could publish her own key claiming it is Bob's.
The Solution: Certificate Authorities (CA). A CA is a trusted entity that signs Public Keys.

1. Bob proves his identity to the CA.
2. The CA digitally signs Bob's Public Key, creating a **Certificate**.
3. Alice verifies the CA's signature on the certificate (using the CA's public key, which is pre-installed in her browser/OS).

TLS (Transport Layer Security). Also known as SSL (Secure Socket Layer). Segments data into chunks, compresses, applies MAC, encrypts, and adds a header. Uses asymmetric encryption for key exchange and symmetric encryption for data transfer.

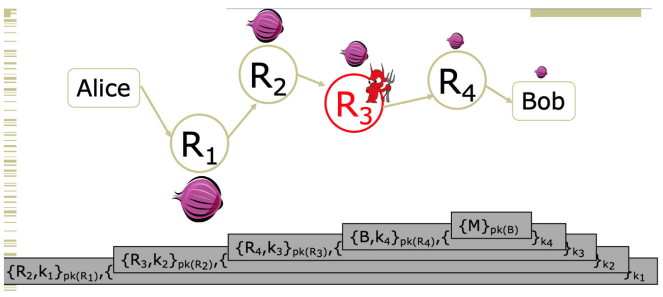
- Handshake Steps:
  1. Clients and servers negotiate exact cryptographic protocols
  2. Client's validate public key certificate with CA public key.
  3. Client encrypt secret random value with servers' public key, and send it as a challenge.
  4. Server decrypts, proving it has the corresponding private key.
  5. This value is used to derive symmetric session keys for encryption & MACs.



Diffie-Hellman key exchange. Allows two parties to establish a shared secret key over an insecure channel. Assume a large prime p and a generator g of a cyclic group (e.g., $\mathbb{Z}_p^*$). Alice chooses a private key a and sends Bob $A = g^a \mod p$. Bob chooses a private key b and sends Alice $B = g^b \mod p$. Both compute the shared secret key: Alice computes $K = B^a \mod p$; Bob computes $K = A^b \mod p$.

Randomized routing. In a network, each node forwards a packet to a randomly chosen neighbor until it reaches the destination.

$\boxed{\textit{Onion routing.}}$ The sender wraps the message in multiple layers of encryption, each layer corresponding to a node in the path. Each node peels off one layer of encryption and forwards the message to the next node, until it reaches the destination.



Routing info for each link encrypted with router's public key Each router learns only the identity of the next router

Note: k1 , k2 , k 3 etc are session keys, so when each router (R1 , R2 , .. Rn) use their private keys to decrypt the packets, they can only then get the next hop (e.g. R2) and the session key (k1) to decrypt the rest of the packet and send it along.

94