Parallel Systems

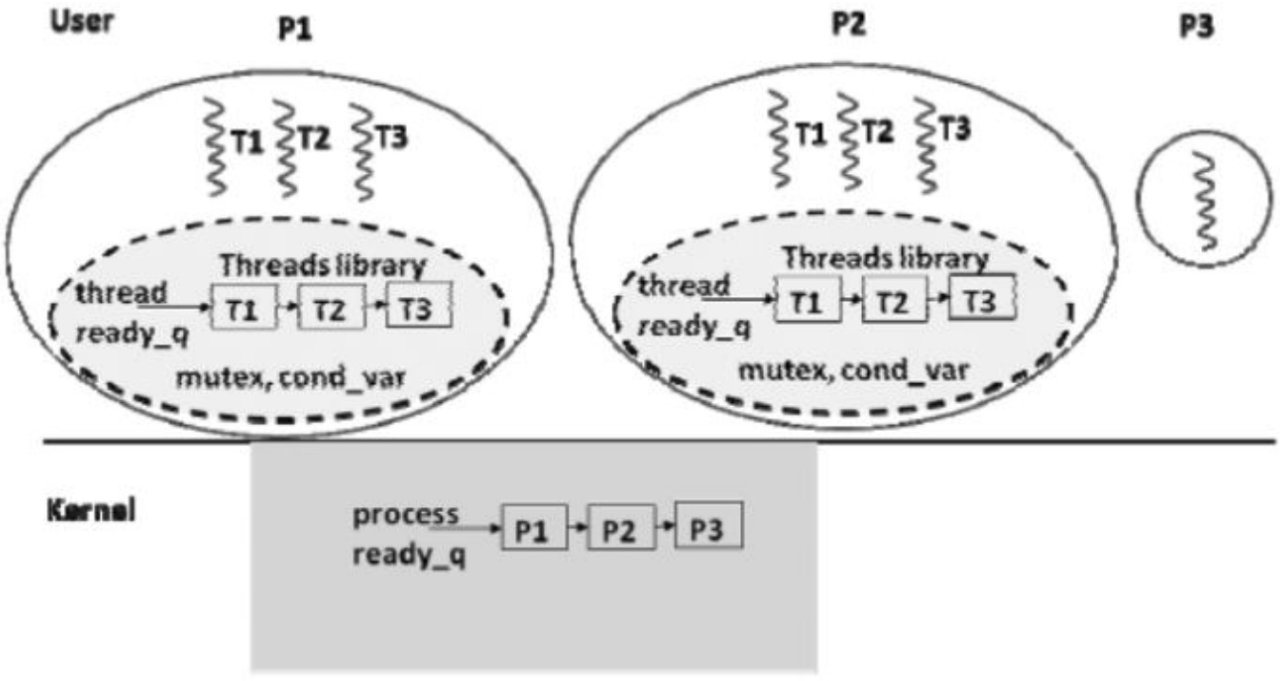
We want a way to implement a way where multiple processes can happen in **parallel** with **threads**

Threads

- Share the addr space of a process
  - Each thread will have a distinct stack but share other parts of the address space
- User-level threads
- Kernel-level threads

User Level Threads

- Threads are scheduled inside the process
- We are essentially switching what the process is doing by changing its current thread



Pros

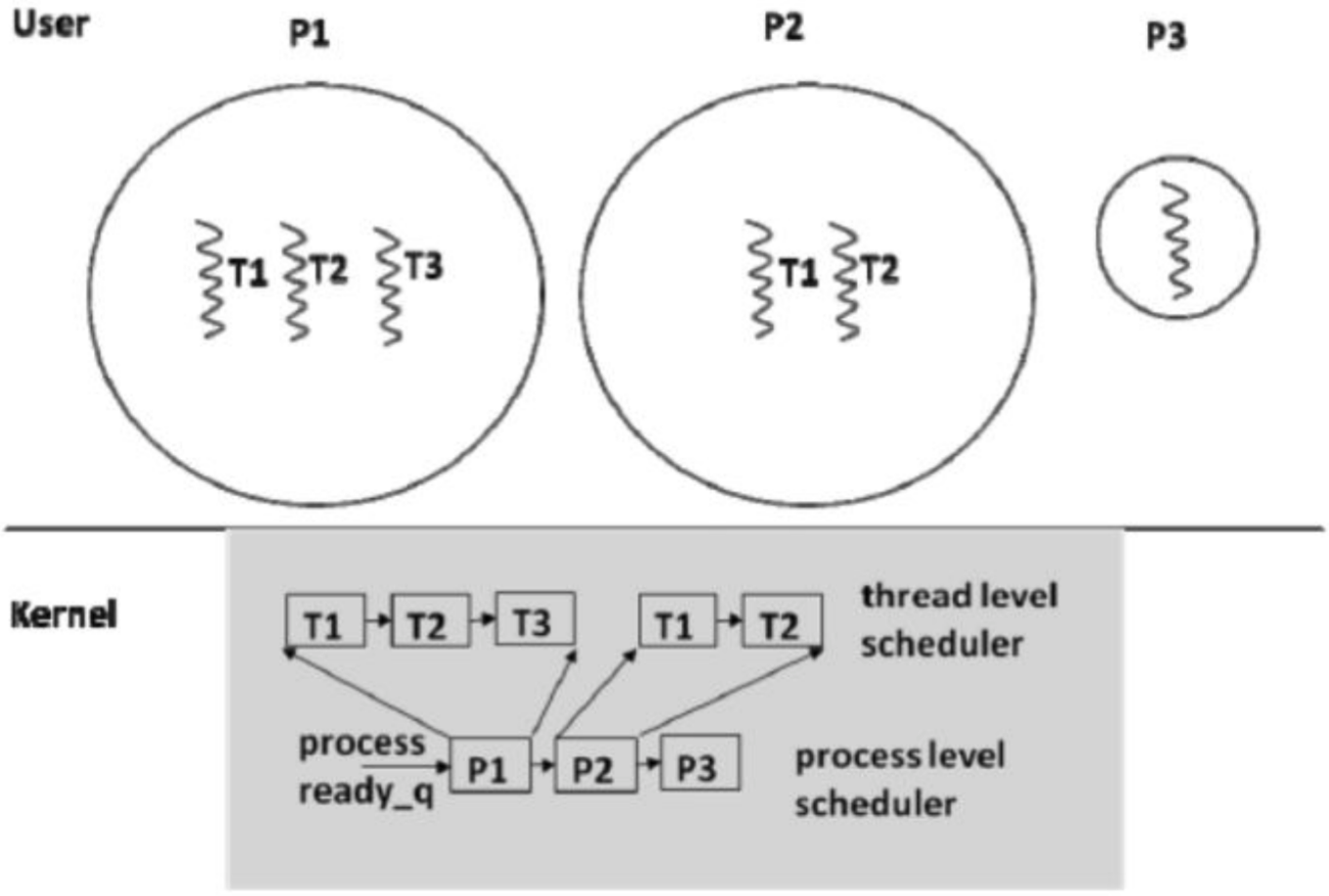
- OS independent and customizable scheduling
- Thread switching is cheap because you do not have to do anything on the kernel level

Cons

- When a thread blocks a process (i.e. system call), it must make a blocking call. This can be done by passing a syscall through the threading library or the kernel makes an upcall to the user space to tell the threading library to block
- No performance advantage on a multiprocessor compared to a single processor. Why?

Kernel-Level Threads

Thread scheduling is handled on the kernel level (wow)



Pros

- Blocking calls are easy
- You can have true multiprocessing

Cons

- Thread switching is expensive
- The Kernel become more complicated because we now have thread and process scheduling
- Thread packages are non-portable - native to an OS

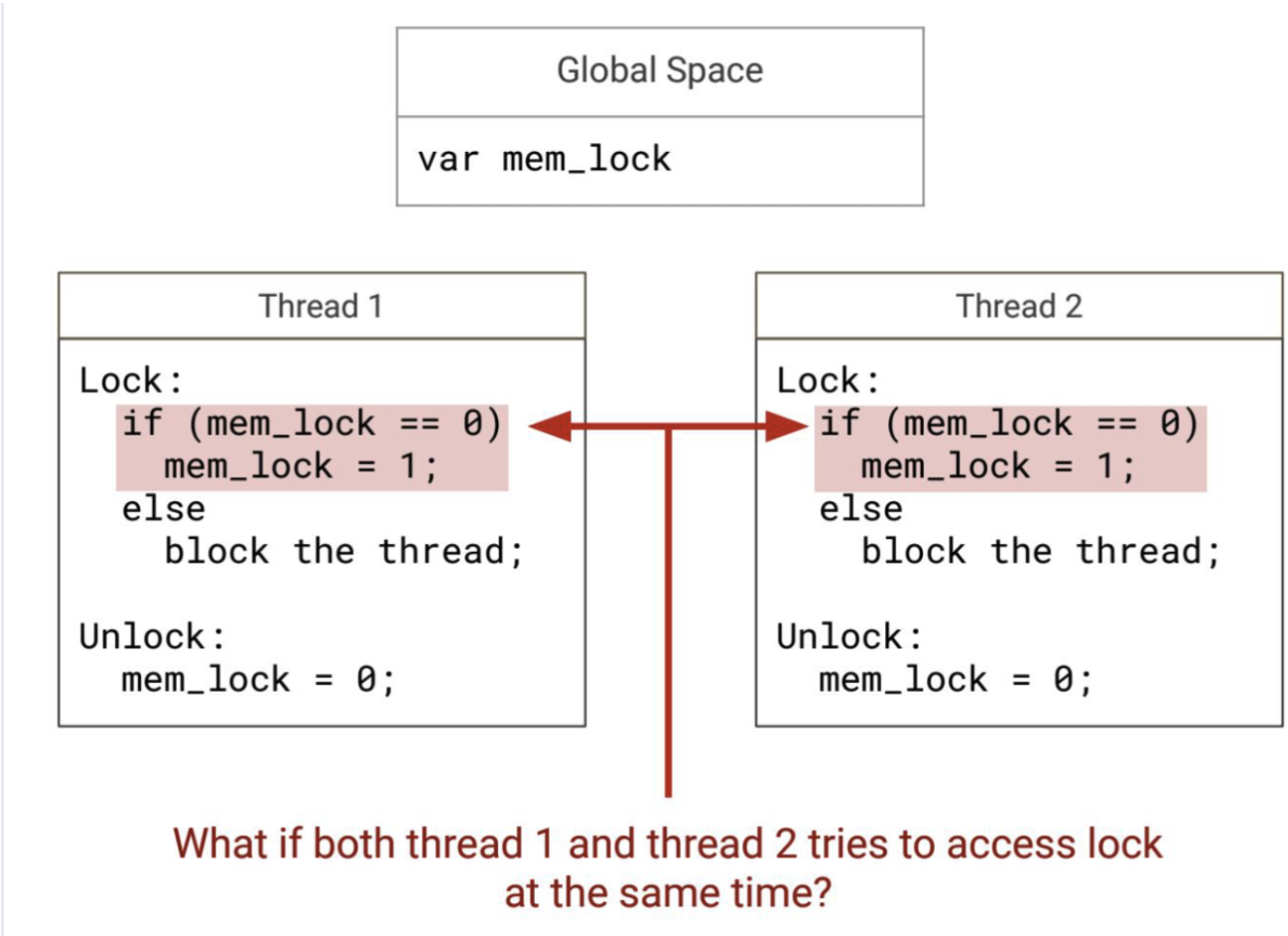
Basic Locking Mechanism

Implementing a mutually exclusive lock

- Use `mem_lock`, which is a global variable to both threads
- If `mem_lock` is 1, then some thread currently has the lock

Lock Algorithm

1. Read a memory location
2. Test if the value is 0
3. Set the memory location to 1



New Instruction: Test and Set

`test-and-set(<memaddr>)` returns the currrent value in `<memaddr>` and sets `<memaddr>` to 1

- Created because we need atomicity

```
int lock(int L) {
    int x;

    //while value is set to 1
    //i.e. someone else has the lock
    while ((X = test_and_set(L)) == 1) {
        block(current_thread);
        //put into lock queue in threading library
    }
    return 1;
```

```c
}

int unlock(int L) {
    L = 0;
    return 1;
}
```

System Guarantees

1. Threads of the same process must share the same page table
2. Threads of the same process have identical views of the memory hierarchy despite being on different processors
3. Threads are guaranteed atomicity for synchronization operations while executing concurrently

Symmetric Multiprocessor (SMP)

Each CPU has an identical view of the system resources

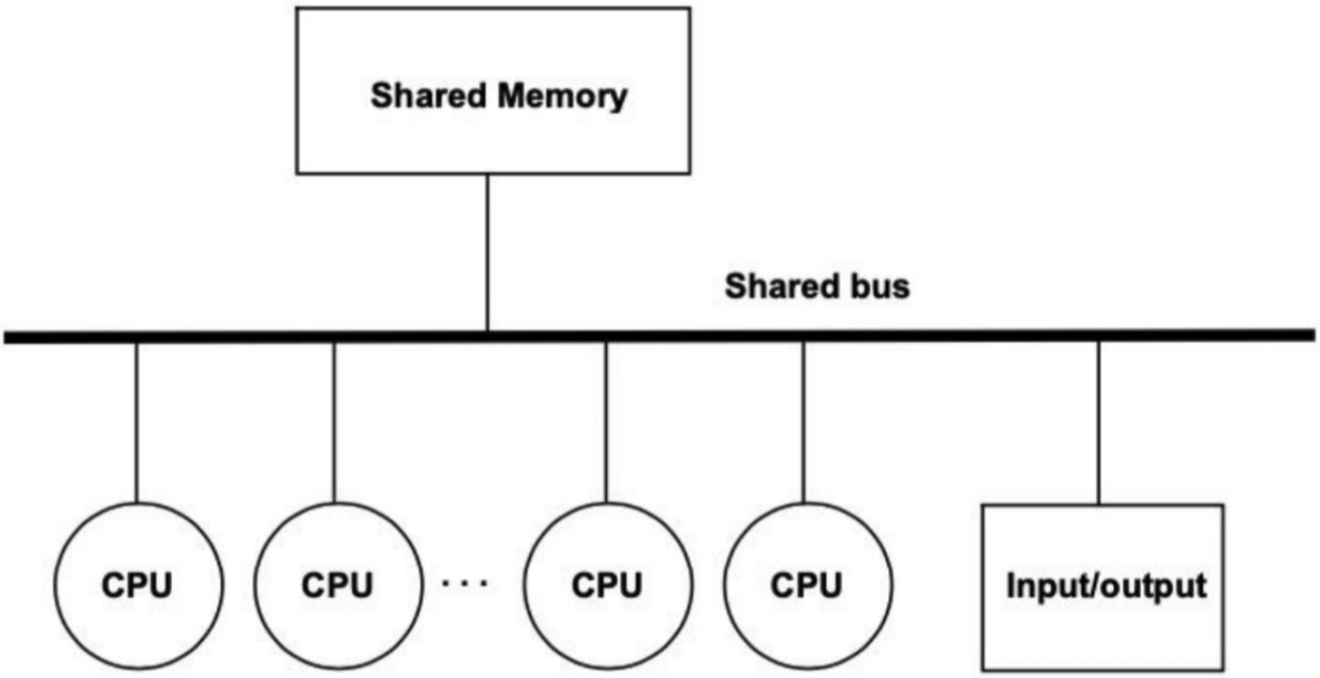
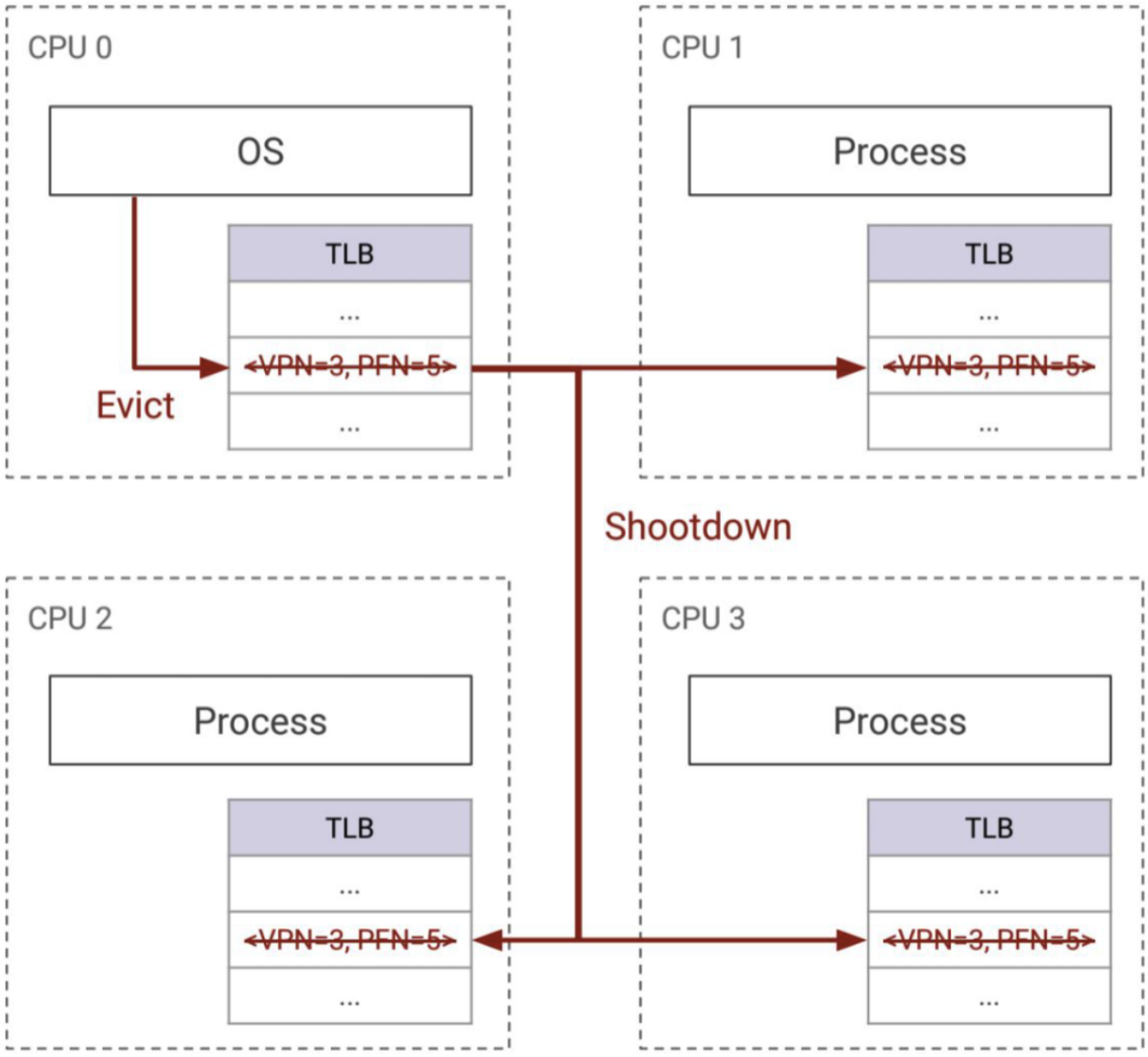


Figure 12.25: A Symmetric Multiprocessor (SMP)

Page Tables In SMP

- Sharing address space is trivial since each processor has access to the same page table in memory
- During page replacement, OS evicts the TLB entry associated with evicted page

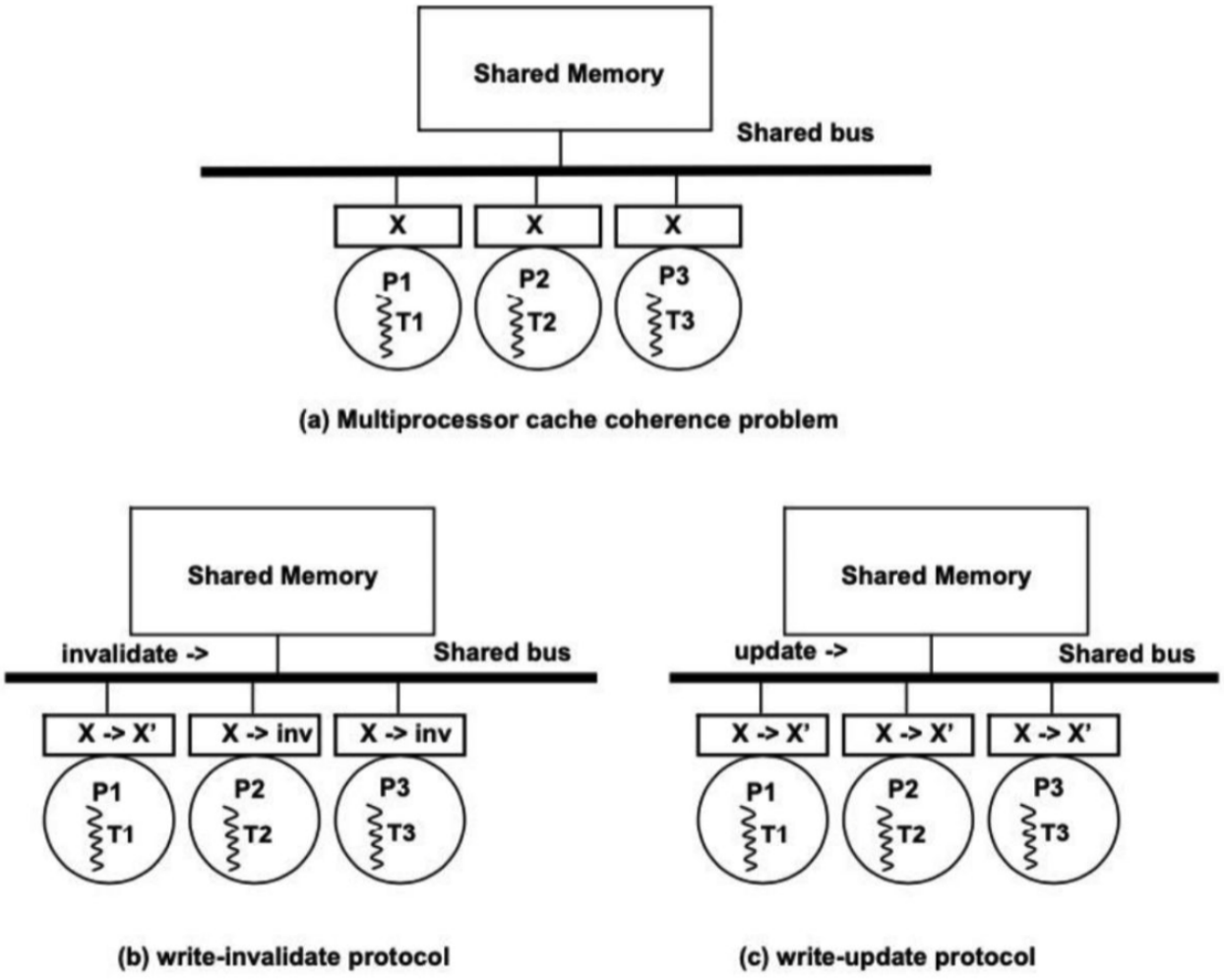
  - To satisfy the first requirement of page table consistency, we also need to invalidate the corresponding TLB entries in the other CPUs
  - This is called TLB shootdown



Memory Hierarchy

Cache Coherence - for a SMP design with per-processor caches, the *hardware* needs to ensure all threads have a consistent view of the shared memory

- **Write Invalidate** - have invalidation lines that all processors are monitoring on. Invalidate the cache entry when written on another processor
- **Write Update** - update the cache entry when peer observed a memory write request on the bus



Networking

Terms	Definitions
Host	Device such as a computer
Network Interface Card	Intermediary item in between network and host
Socket	Provides the API for developing network applications
Protocol Stack	Abstraction layers that define the ways that network devices and software communicate
Internet	The composite of sevberal networks that use the same protocol to communicate

Protocol Stack

Internet Protocol Stack (IP Stack)

- Application Layer - application-defined messages
- Transport Layer - managing fragments of messages
- Network Layer - sending/receiving packets
- Link Layer - sending/receiving frames
- Physical Layer - sending/receiving bits

Transport Layer

- Packetizes the message/data
    - Takes the information that needs to be sent which is given by the application layer and splits it up into packets to be transported
    - Types of protocols:

- Stop and wait
- Pipelined transmission
- Reliable pipelined transmission with windowing

Types of Protocols

Stop and Wait

Steps

- Sender sends a packet
- Receiver receives the packet and sends a positive ACK (acknowledgement)
- Sender waits for ACK

  If no ACK is received then a timeout occur that retransmits the packet

How do we ensure that a delayed packet doesn't get received and ACKed instead of what was meant to be ACKed?

- Add a sequence number
  - Alternate 0 and 1 (alternating bit protocol)
  - What happens when a bit gets corrupted?
  - Use increasing sequence number

Issues With Stop and Wait

- Very slow
  - Have to wait for packet to be ACK'd before sending another

    - Leads to a ton of waiting time where nothing is happening and everyone is just waiting

Solutions? Pipelined transmission

Pipelined Transmission

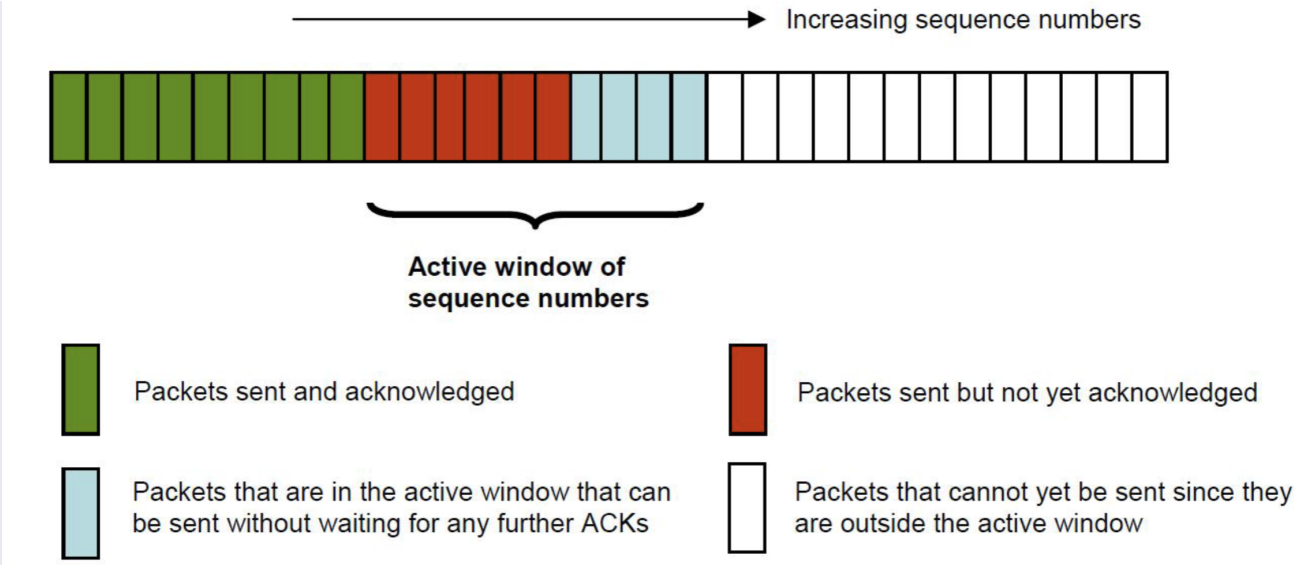
- Sender will send a set of packets before waiting for acknowledgement
- How do we go about acknowledging the packets that were received? Depends on the protocol

Reliable Pipelined Protocol

- Each packet that is received gets its own acknowledgement sent back to the sender
    - By using a sliding window, we can wait for an ACK to come back for the first packet and then shift the window over by a packet
- Can this be optimized?

  - We can send a cumulative ACK

  - A single ACK which will serve as an acknowledgement that all packets up to and including a certain sequence number have been received



Transport Layer Protocols

Transport Protocol	Features	Pros	Cons
TCP	Connection-oriented, self-regulating, data flow as stream, supports windowing and ACKs	Reliable, messages arrive in order, well behaved due to self-policing and reducing network congestion	Complexity in connection setup and tear-down, at a disadvantage when mixed with unregulated flows, no guarantees on delay or transmission rate

Transport Protocol	Features	Pros	Cons
UDP	Connection-less, unregulated, message as datagram, no ACKs or windowing	Simplicity, no frills, especially suited for environments with low chance of packet loss and applications tolerant to packet loss	Unreliable, message may arrive out of order, may contribute to network congestion, no guarantees on delay or transmission rate