COS3043
System Fundamentals

Lecture 2 (Part 2)

List of Discussion

Part 1

- OS Structure Overview
- The SPIN Approach

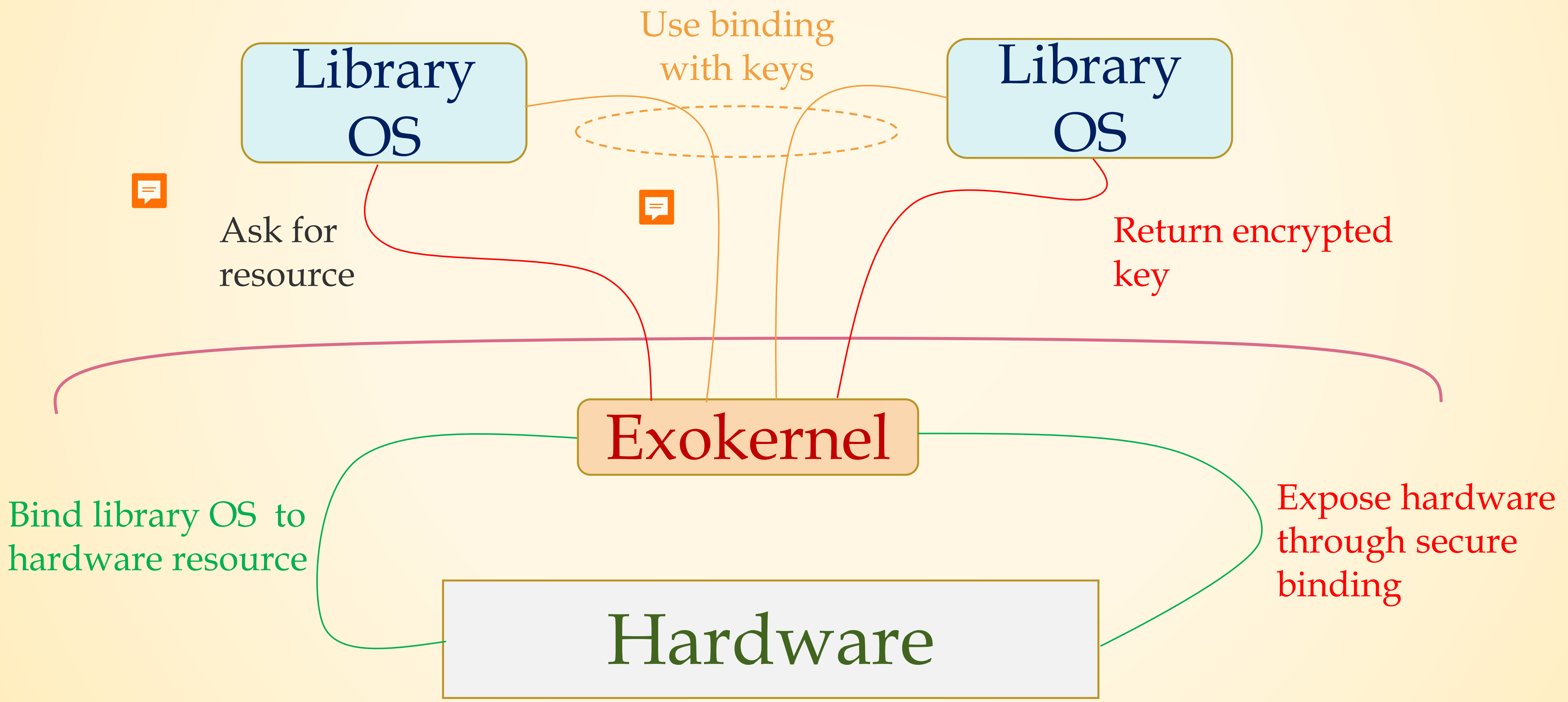
Part 2

- The Exokernel Approach
- The L3 Microkernel Approach

Exokernel Approach

Exokernel Approach to Extensibility

Decouple Authorization from Use



Examples of Candidate Resources

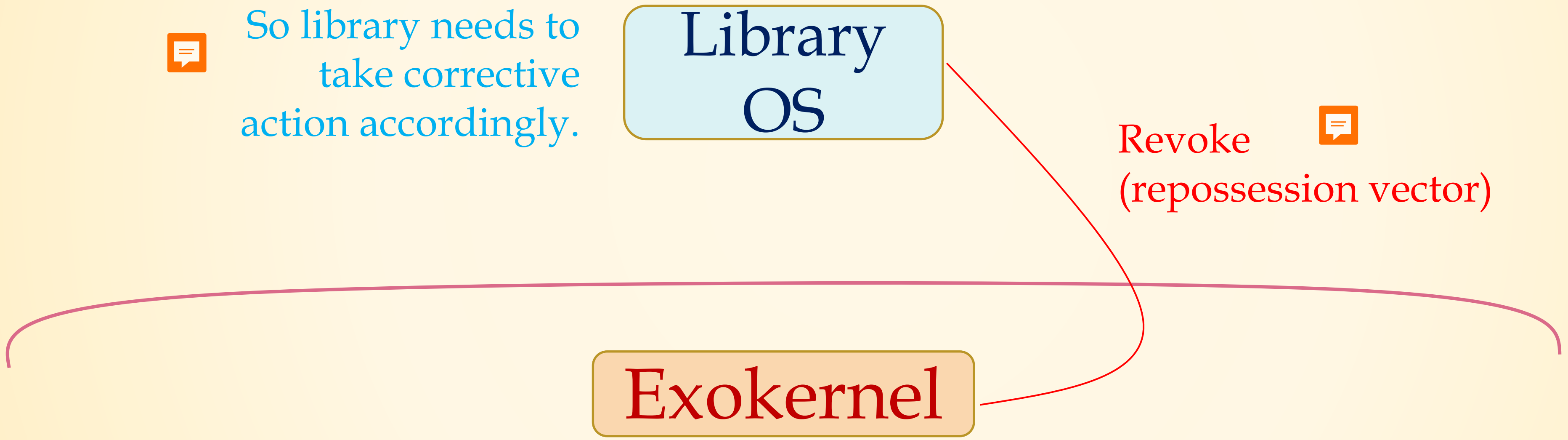
- TLB Entry
  - Virtual to physical mapping done by library OS
  - Binding presented to Exokernel
  - Exokernel verifies and puts it into hardware TLB
  - Once in hardware TLB entry, process in library OS can use the TLB memory multiple times without Exokernel intervention at hardware speed.
  - NOTE TLB = translation lookaside buffer (a type of memory cache)
- Packet Filter (for filtering all arrival network packets)
  - Codes loaded into kernel by library OS
  - Once loaded, on every packet arrival, it is checked by Exokernel at hardware speed.

Implementing Secure Binding

- Three methods
  - ➢ Hardware mechanisms
    - ✓ TLB entry, page frame
  - ➢ Software Caching
    - ✓ "Shadow" TLB in software in each library
  - ➢ Downloading code into kernel
    - ✓ Functionally equivalent to SPIN extensions with Logical Protection Domain.
    - ✓ But in terms of level of protection, SPIN has more protection as it is enforced at compilation and verified at runtime due to its programming language.
    - ✓ However, Exokernel only allows trusted code to be downloaded to implement secure binding.

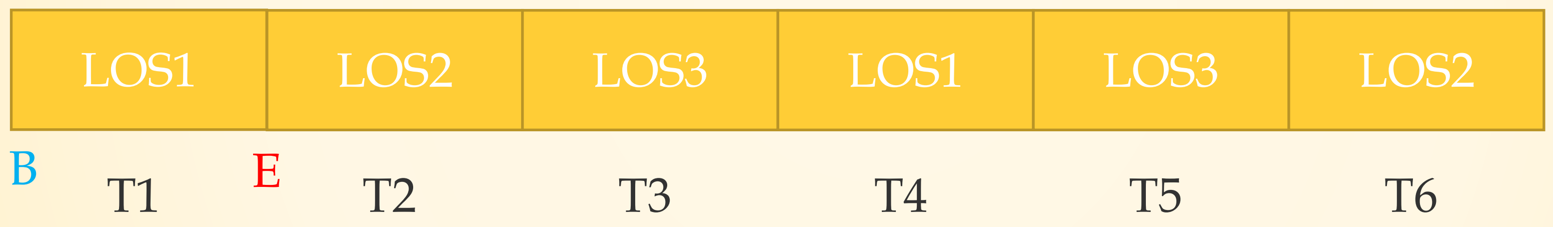
Revocation of Resources

Space (memory) and time (CPU)



Default Core Services in Exokernel

- Memory Management
  - Software TLBs
- CPU Scheduling
  - Linear vector of time slots/time slices



Research on OS Performance

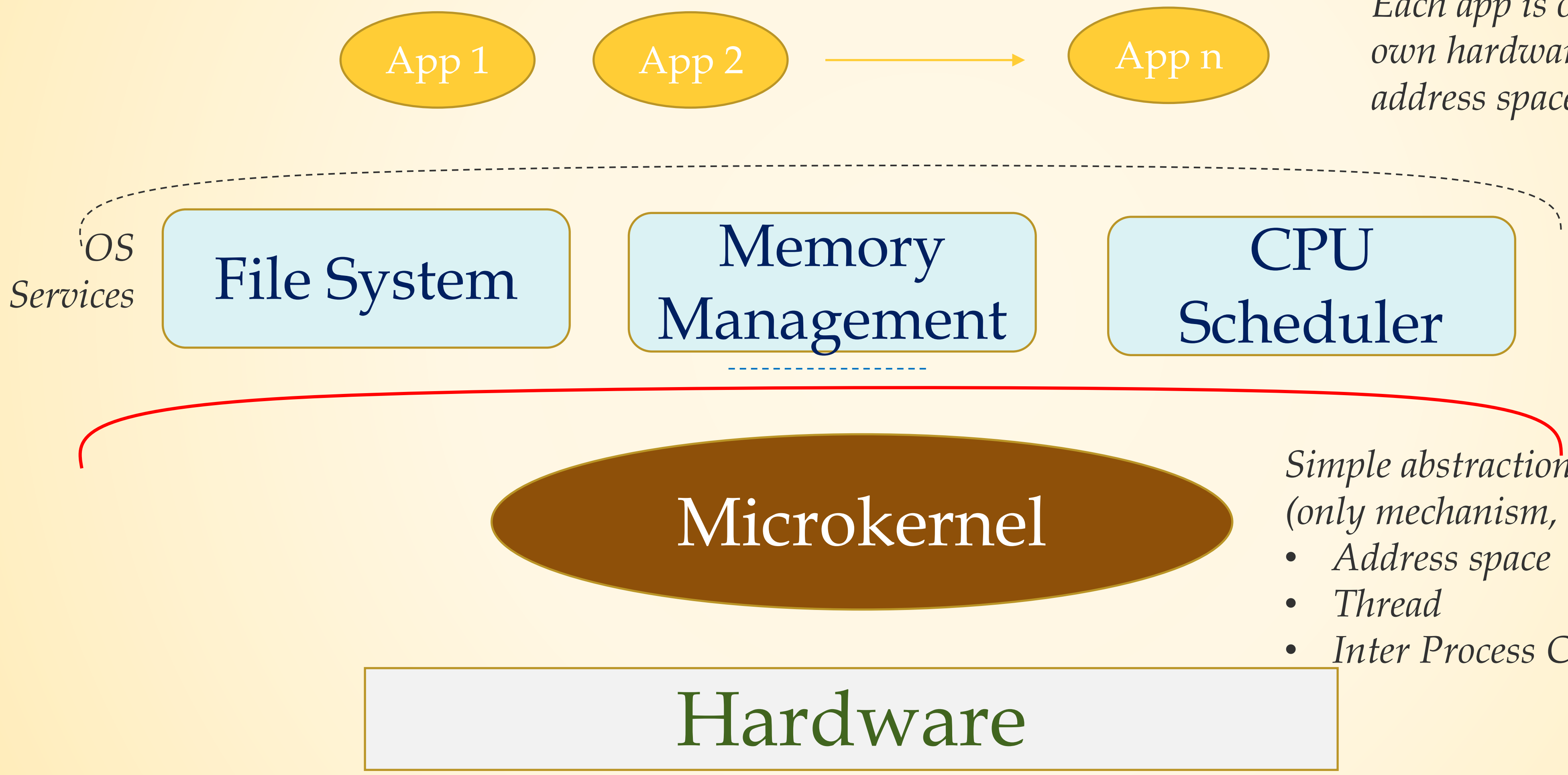
- From which perspective?
  - Time
  - Space
- Equal or better?
- Comparison on:
  - Monolithic
  - Microkernel
  - Extended kernel: SPIN/Exokernel

L3 Microkernel Approach

Introduction

- Both SPIN and Exokernel started with:
  - ➢Microkernel-based design compromising on performance due to frequent border crossing.
- Example of microkernel-based OS - Mach 1990s has create a misconception that microkernel-based structure is not going to breakthrough the barrier of performance although the fact that Mach was designed to focus on portability rather than extensibility.
- So at the later stage, researchers started to think can we achieve the goal of performance with microkernel? Here it comes the L3 microkernel approach.

Review - Microkernel-Based OS Structure



Each app is on its own hardware address space.

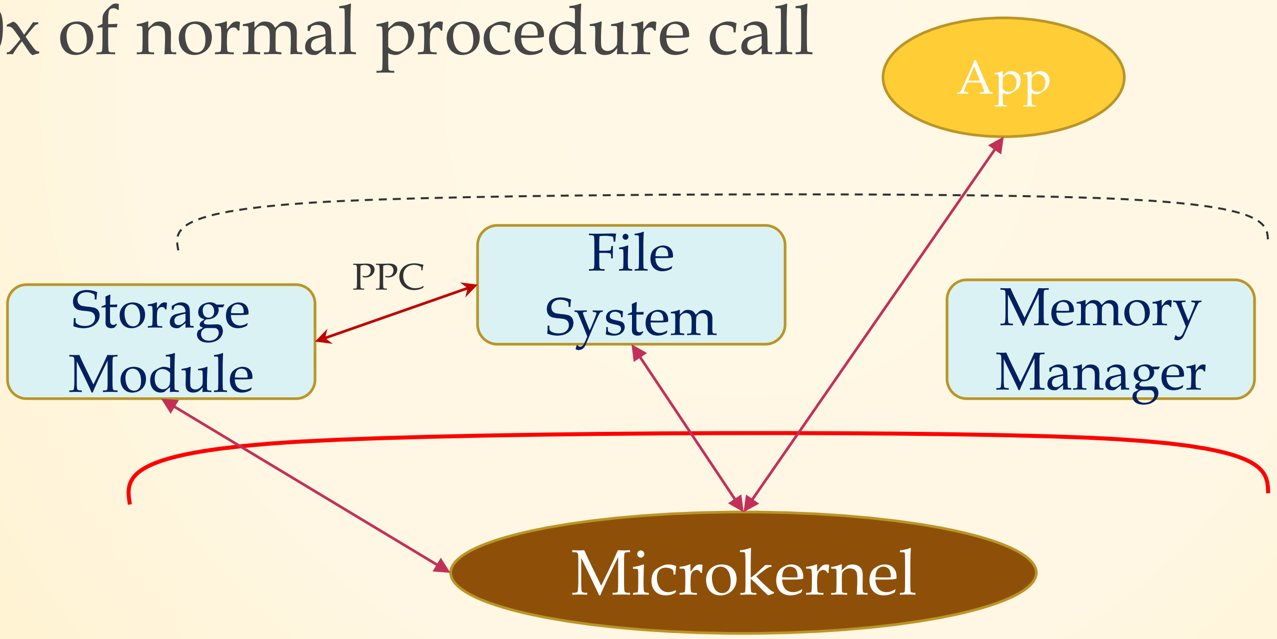
Each service is on its own hardware address space.

Simple abstraction (only mechanism, no policy):

- *Address space*
- *Thread*
- *Inter Process Communication (IPC)*

Review - Potential Performance Loss of Microkernel

- Border Crossing
  - ➢Implicit + explicit costs
- Protected Procedure Call (PPC)
  - ➢100x of normal procedure call

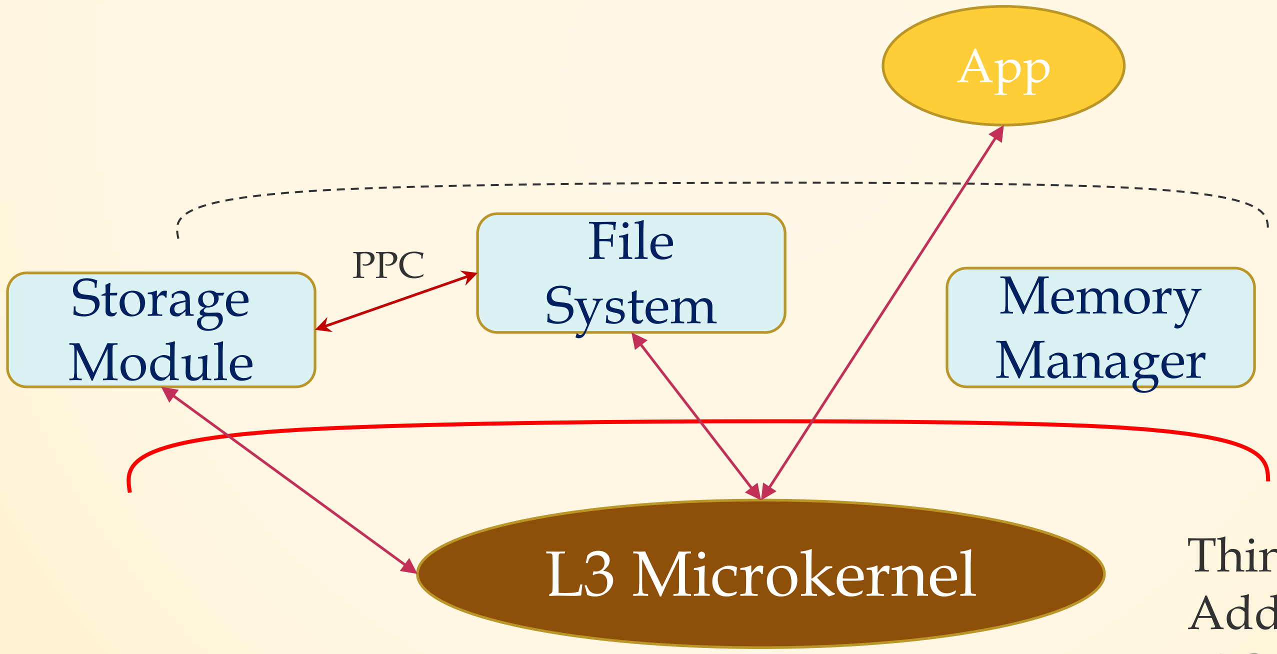


L3 Microkernel Approach

- Proof by construction to debunk the 'myths' about microkernel based OS structure.

Main idea: It's about the efficiency of implementation!!

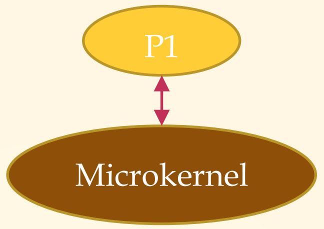
Each in own protection domain

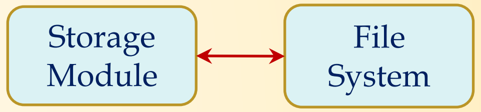


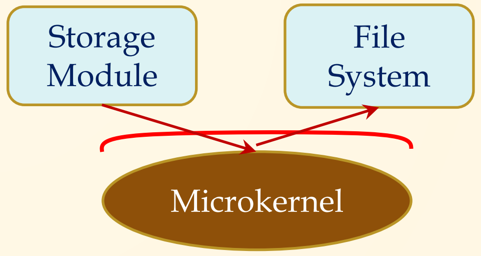
Thinner Microkernel: Address Space, Thread, IPC, UID

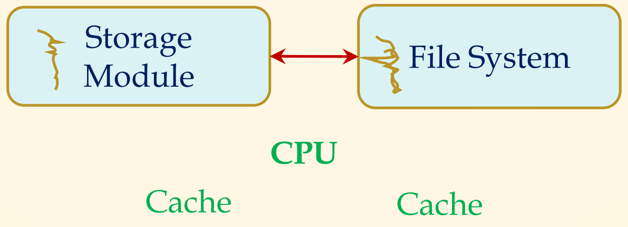
Strikes Against Microkernel

- Kernel - user switches
  - ➢ Border crossing cost
- Address space switches
  - ➢ Basis for PPC to cross domain protection calls
- Thread switches + IPC
  - ➢ Kernel mediation for PPC
- Memory Effects
  - ➢ Locality loss



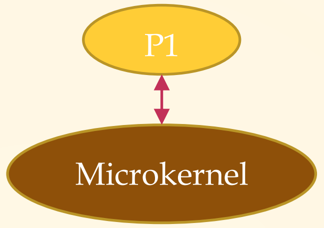




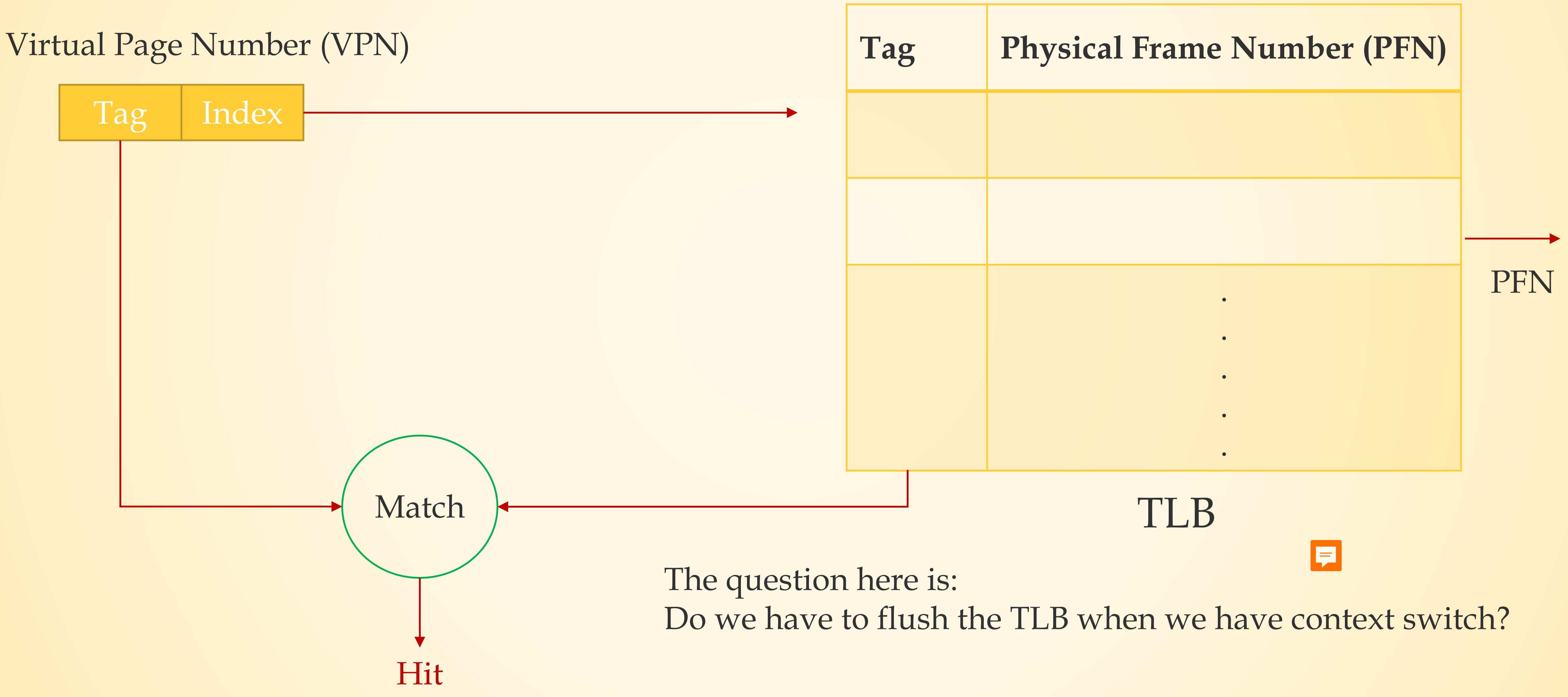


Debunking => Kernel - User Border Crossing Myth

- By empirical proof
- L3
  - 123 processor cycles (including TLB and cache misses) by benchmarking
- Mach 1990s
  - 900 cycles

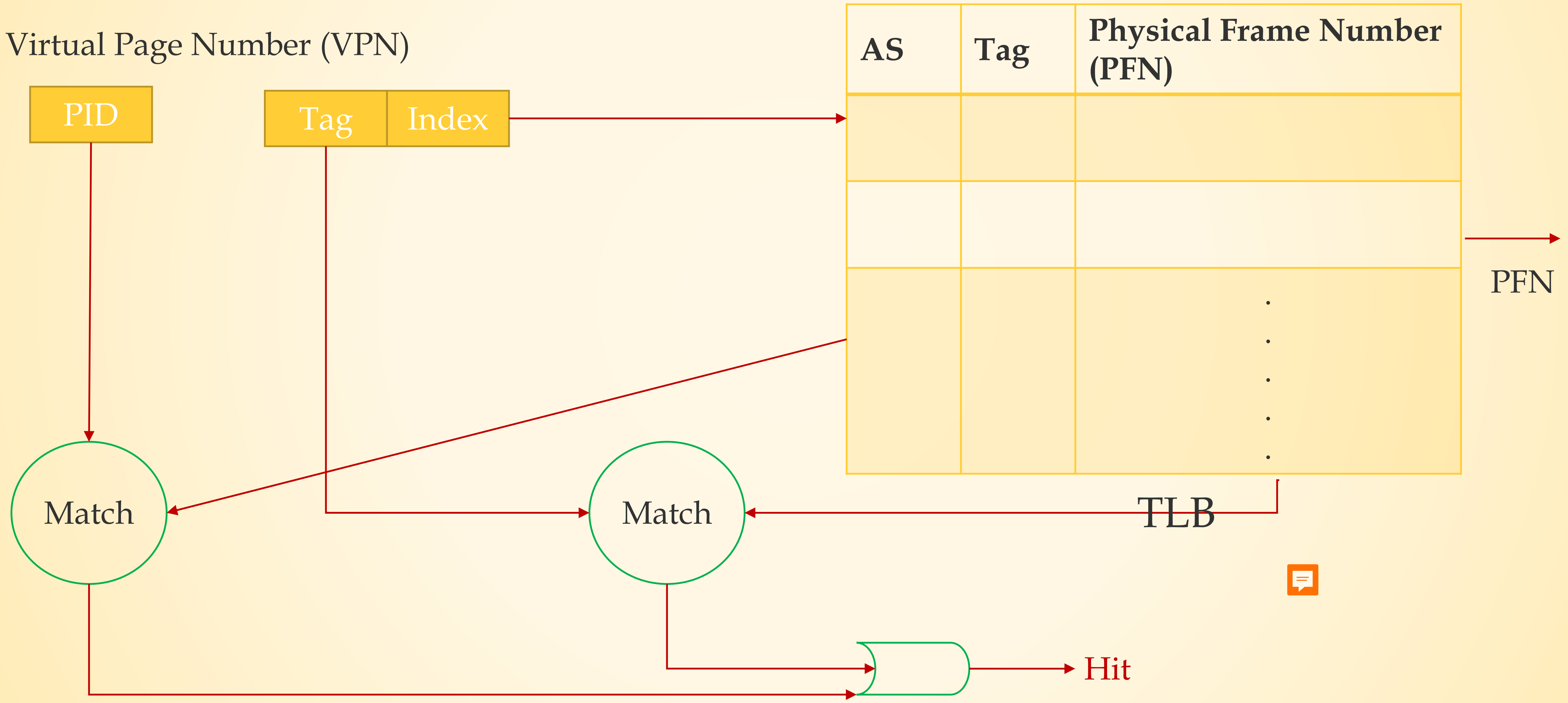


Debunking => Address Space Switches



The question here is:
Do we have to flush the TLB when we have context switch?

Address Space Switches – With AS Tagged TLB

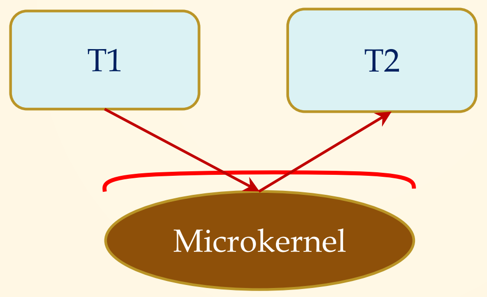


Suggestion for Avoiding TLB Flush

- Liedke – the L3 author suggested:
  - Exploit architectural features
    (ex: segment registers for x86 processors)
  - Share hardware address space for protection domains.

Debunking => Thread Switch + IPC Myth

- By construction, it is shown to be competitive to SPIN and Exokernel.
- Switch involves saving all the volatile stage of processor.



Debunking => Memory Effects Myth

- For small protection domains:
  - ➢Share hardware address space for small protection domains (by use of Segment Registers).
- For large protection domains:
  - ➢It will be the same costs even for monolithic.

Principles of L3 for OS Structure

- Minimal abstraction in microkernel.
- Microkernels are processor specific in implementation – no portability.
- The above two **=> Efficient processor independent abstractions at higher layers.**