Memory Management

Peter J. Denning

A complex topic

- Memory is fundamental to computing. It holds the representations of all data and instructions. No computation can work without it.
- Memory story less known than CPU to most users. But it's well known in OS and has its own level in the kernel.
- CPU is a chip. Memory is a system. Managing memory is significantly more challenging than managing CPU.

Responsibilities

- Memory manager deals with these issues:
  - Memory hierarchy
  - Private address space for each process (partitioning)
  - Mapping address space to real memory (virtual memory)
  - Minimizing data traffic in the hierarchy (replacement)
  - Prevent thrashing (multiprogramming and working sets)
  - Maximizing system throughput (working sets)

- But not these issues:
  - Defining sharable information objects
  - Uniquely naming information objects anywhere in Internet
  - Controlling access to shared objects

- Dealt with by kernel levels above memory level

Basic Functions

- Representing address spaces
  - Pages, frames, master files
- Address mapping
  - Pages to frames of virtual memory
- Multiprogramming
  - Partitioning memory among address spaces
- Performance
  - Paging algorithms
  - Thrashing avoidance

Memory manager as Abstract Machine

- Internal hidden data structures
  - Page tables, disk tables
- Simple interface
  - `h=CREATE_ADDR_SP(init)`
  - `DELETE_ADDR_SP(h)`

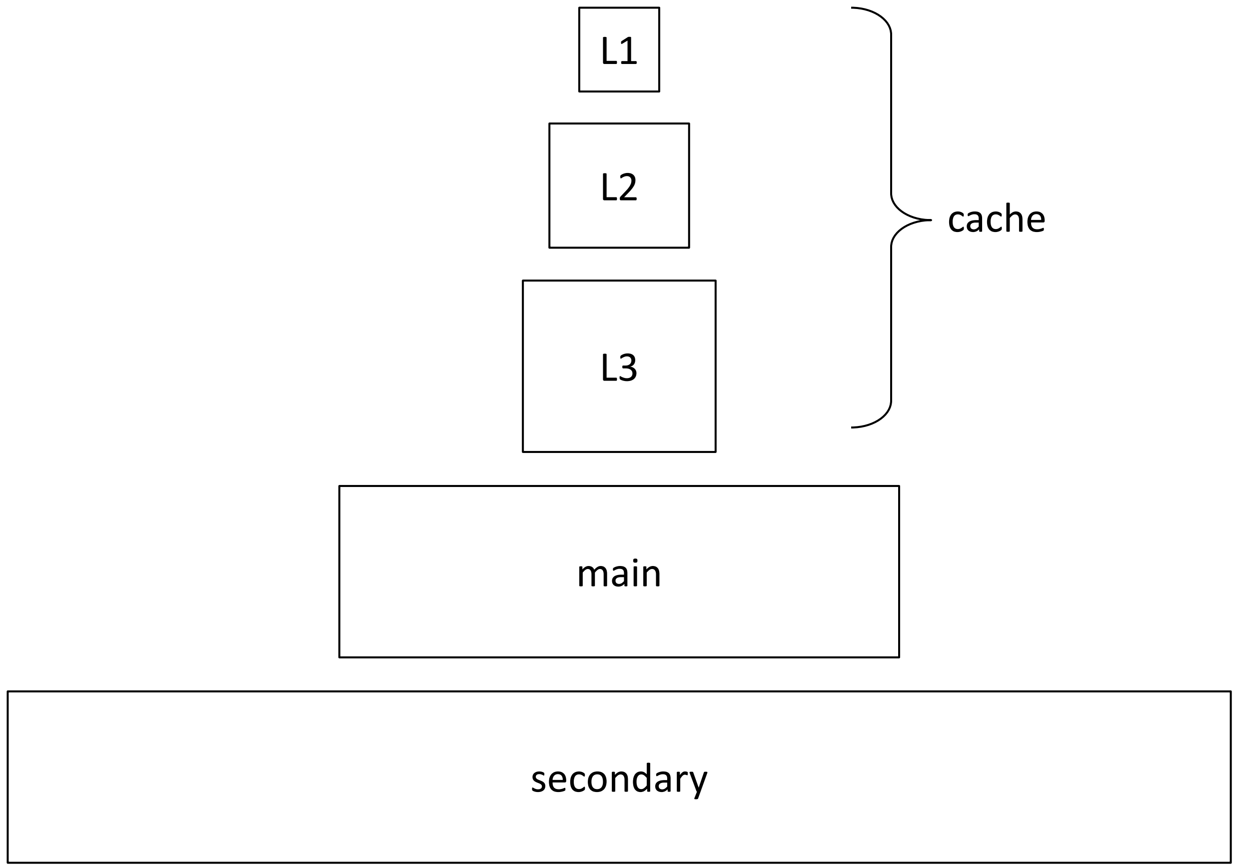
Performance!

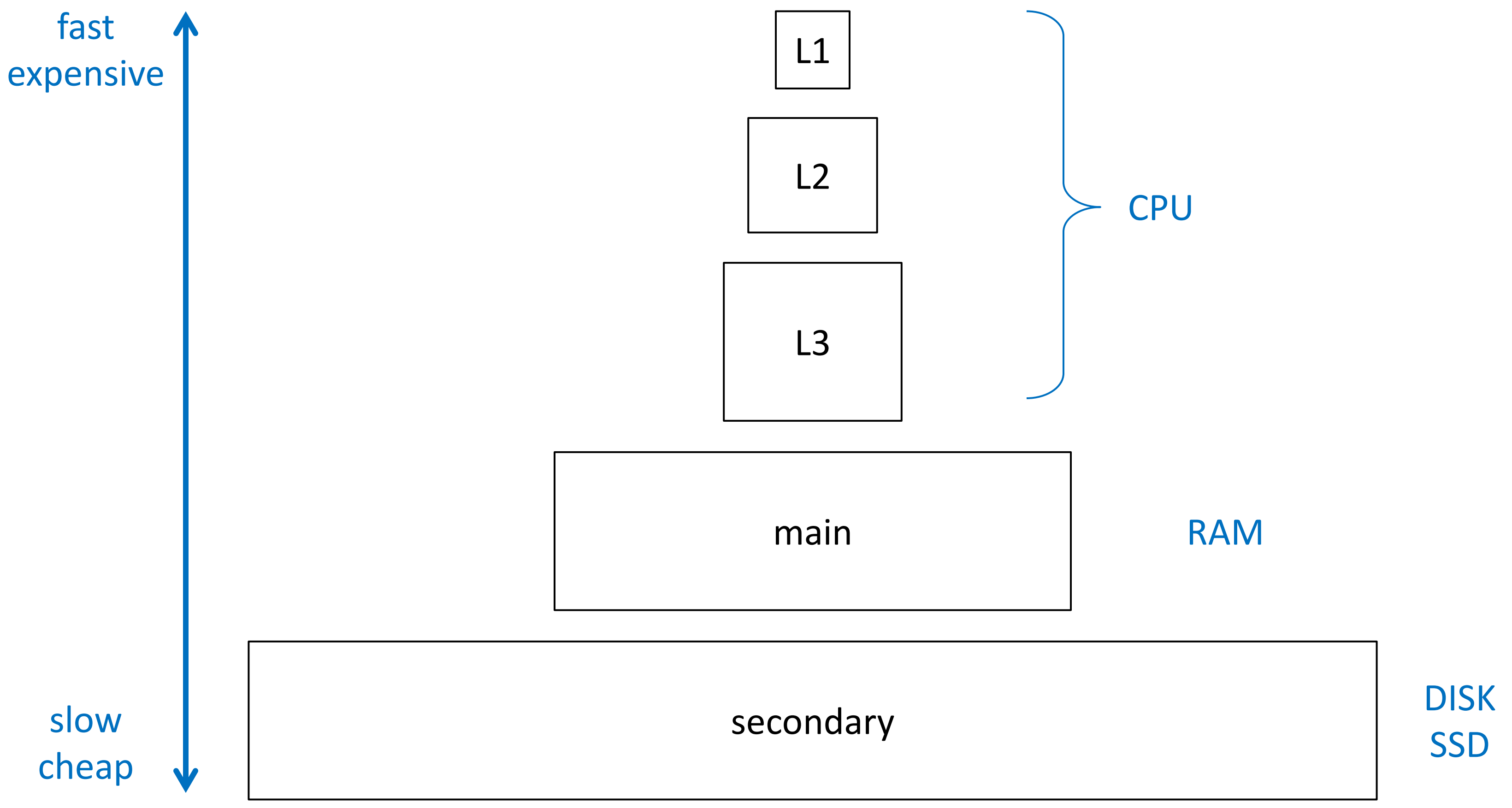
- Make address mapping extremely fast
- Find replacement algorithms to minimize paging
- Avoid thrashing
- Optimize system throughput

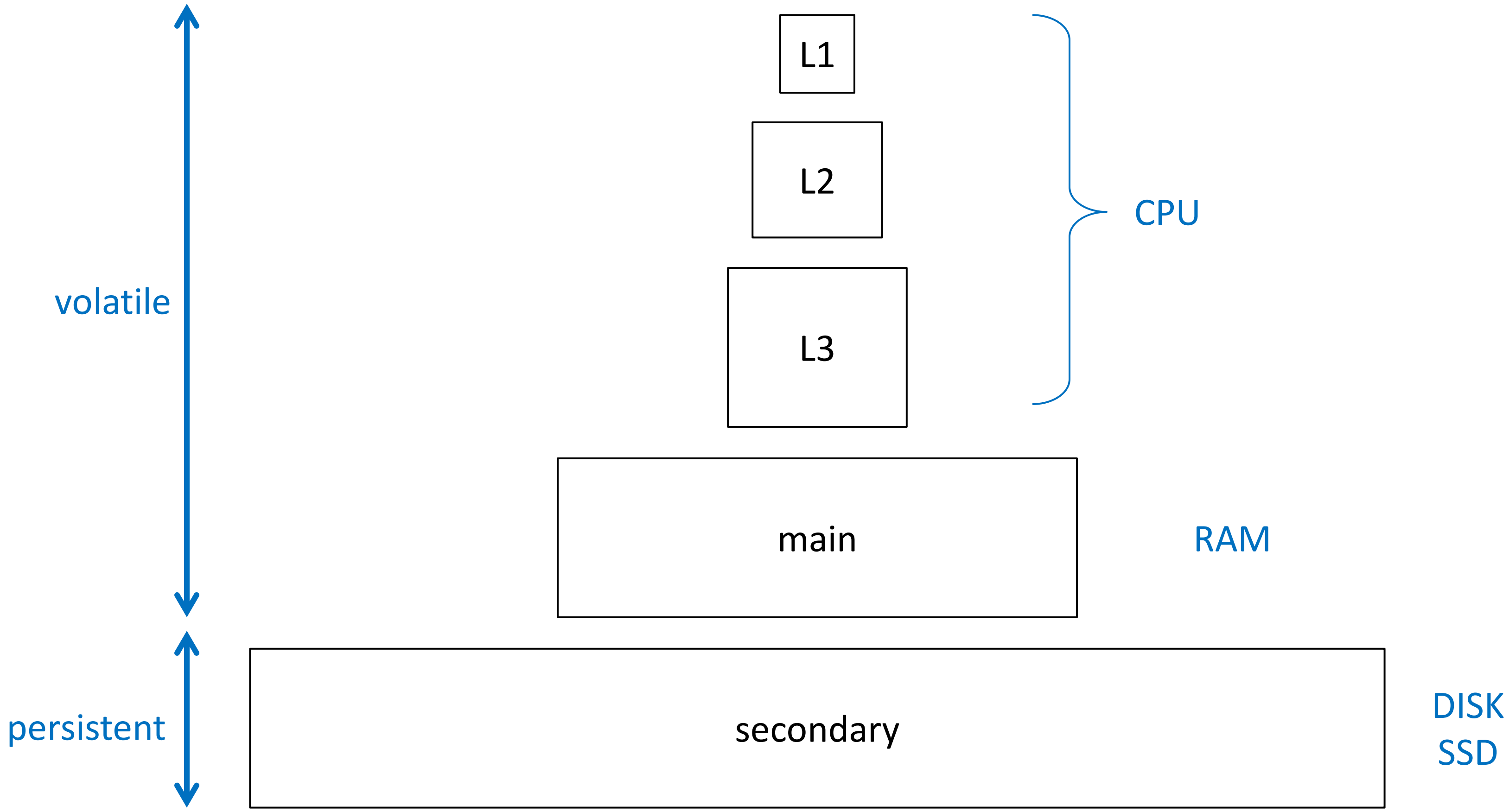
Memory Hierarchy

- Memory is not one device or technology; it is many
  - Level 1, 2, 3 CACHE are hardware devices close to the CPU to enable memory to keep up with CPU
  - RAM (random-access memory) is the main memory holding instructions and data; CPU addresses RAM, caches speed up access
  - DISK is the secondary memory that holds files for extended periods of time; secondary memory includes multiple technologies such as hard disk, SSD (solid state device), Internet caches, Cloud

- DISK is persistent (not erased until told) and low-cost per byte, but slow access time
- RAM is volatile (erased on power failure), much higher cost per byte, fast access time
- CACHE is volatile, very fast, expensive, accelerates access to RAM, works automatically in the background


L1
L2
L3
cache
main
secondary


fast
expensive
slow
cheap
L1
L2
L3
main
secondary
CPU
RAM
DISK
SSD


L1
L2
L3
CPU
volatile
main
RAM
persistent
secondary
DISK
SSD

Data Traffic in the Hierarchy

- Data in the hierarchy are constantly on the move
- CPU accessing RAM triggers moves
  - Data up from RAM into caches
  - Data up from DISK to RAM
  - Data down when no longer needed by CPU

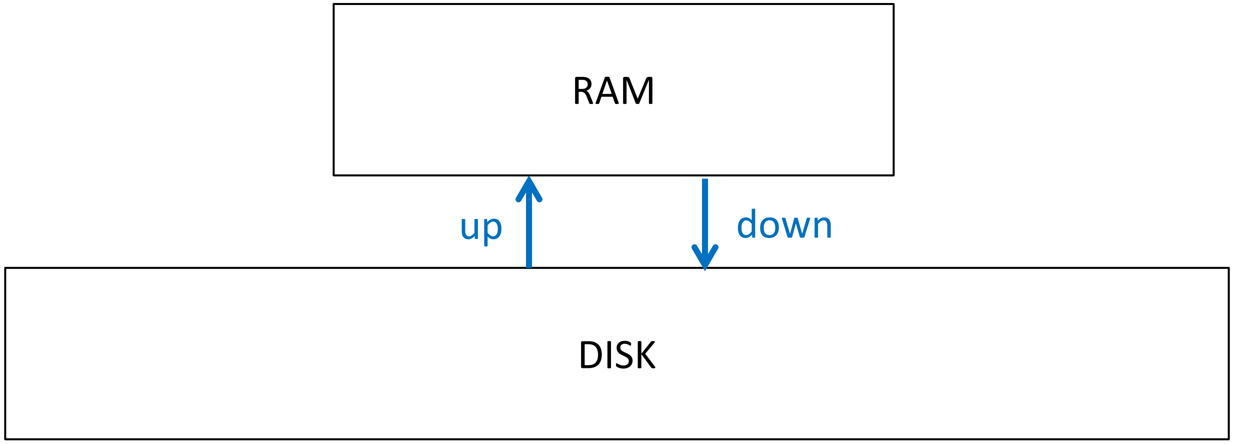
- Other sources of data moves
  - Users and processes exchanging messages
  - File operations open, close, read, write
  - Storing and retrieving from the Cloud
  - Accessing web pages
  - Making backups

- We will focus on up and down moves between RAM and DISK
  - A performance bottleneck
  - Principles for managing them are mirrored in the caches
  - Keep things simple to start with
- Page: unit of storage and transfer
  - All pages are of the same size: power of 2
  - Page size is design choice; common choices are 512 (2^9), 1024 (2^{10}), and 4096 (2^{12}) bytes

up = move a page from secondary to main

down = move a page from main to secondary

Because RAM is volatile, OS keeps **master copies** of all files on DISK. Pages moved up into RAM are a subset of those on DISK. Pages modified in RAM must be moved down to maintain master file consistency.



Speed gap in the Hierarchy

- RAM access times 10^{-8} (10's of nanoseconds)
- Disk access times 10^{-2} (10's of milliseconds)
- Gap -- one up-down transfer costs 10^{6} RAM accesses
  - Very expensive
  - Makes DISK a strong bottleneck
- Avoid gap as much as possible
  - Minimize transfers
  - Switch CPU to another process while one waiting for transfer to complete (multiprogramming)

What Must be Done

- Mapping problem: design a very fast and efficient mechanism to map CPU-generated addresses to physical memory addresses
- Replacement problem: design replacement policies that minimize page traffic
- Sharing problem: allow multiple processes to share the main memory without thrashing