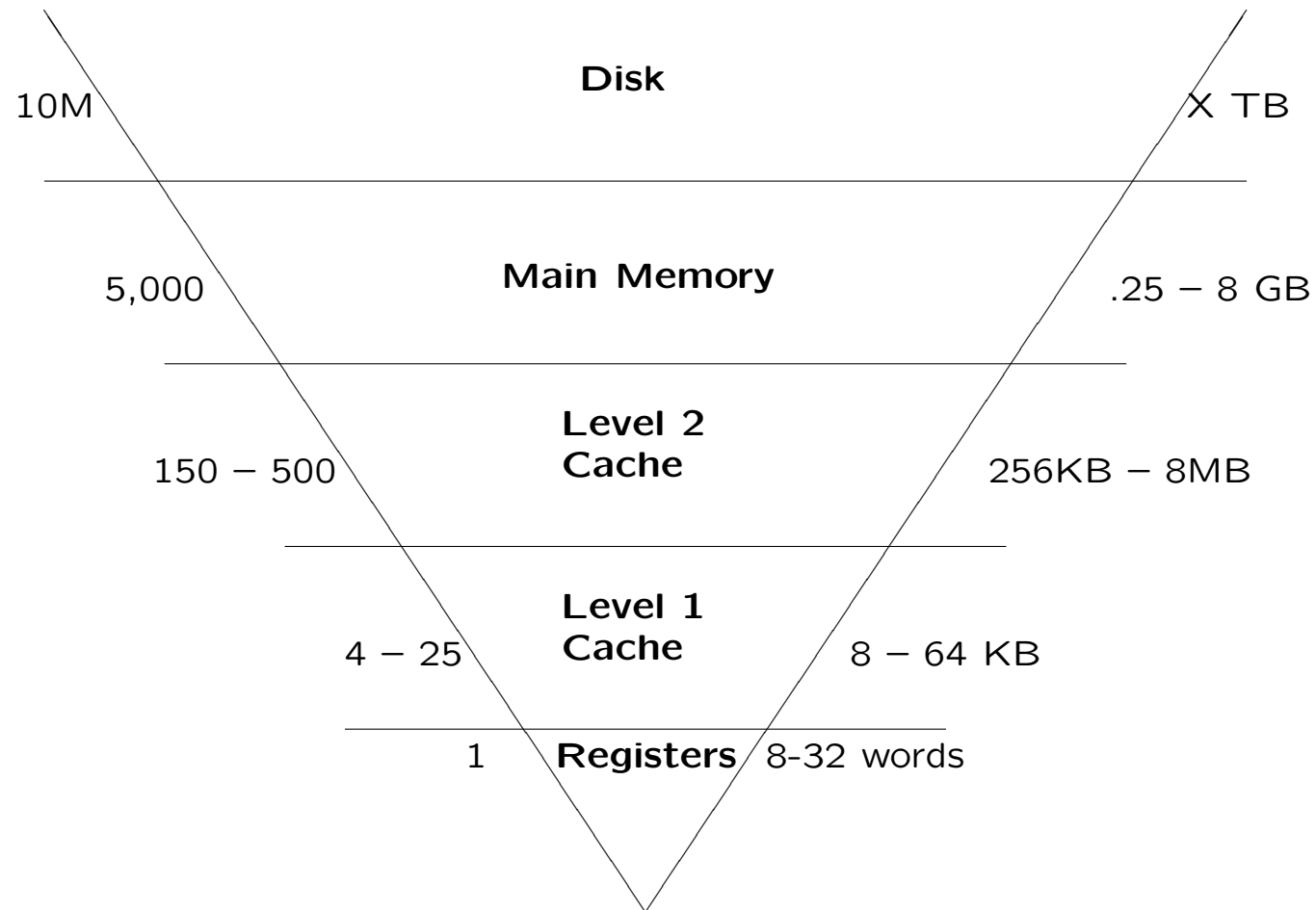


# 1. Memory Hierarchy – overview

- **Cache** smaller, faster area where memory locations may be temporarily stored and operated on
- Cache effectiveness tied to two properties of data access:
  1. **Spatial locality:** Data located close in memory to the present reference is likely to be used soon
  2. **Temporal locality:** Recently accessed is likely to be used again soon



## 2. DDOT Naturally Exploits Registers

DDOT: dot product of vectors  $X$  &  $Y$

```
for (dot=0.0, i=0; i < N; i++)  
    dot += x[i] * y[i];
```

- $2N$  FLOPS
- $4N$  references
- $2N$  data

- dot assigned to a register
  - $4N$  references still needed by algorithm
  - $2N$  access of dot now from register
    1. Init main memory read not needed – zeroed
    2.  $N$  reads and  $N$  writes of dot in same register
- Main memory access reduced to  $2N$  of  $X$  &  $Y$ 
  - $X$  &  $Y$  ref irreducible because no reuse
  - Cache still possibly provides benefit (line fill & prefetch)

### 3. Memory Hierarchy – cache basics

Five basic properties key in understanding caching:

1. Inclusive/exclusive
2. Cache line (block) size
  - Strongly correlated with bus width
  - Simplify book-keeping
  - Usually filled in order
  - Optimizes for spatial locality
3. Associativity
  - Direct mapped cache (1-way assoc)
  - Fully associative
  - N-way associative cache
4. Replacement policy
  - Least recently used (LRU)
  - First in first out (FIFO)
  - Random or pseudo-random
5. Write policy
  - Write through
  - Write back
  - Separate/shared
    - Most L1 caches separate instruction/data
    - Most higher level caches shared

## 4. N-way Set Associativity

### Define:

- $N$  : set associativity
- $N_L$  : number of lines in cache
- $A$  : address of start of cache line in memory
- $N_S = N_L/N$  : number of sets in cache

### To find if a line is in the cache:

1. Find set to search
  - $i = A \bmod N_S$
2. Search all  $N$  possible lines for match

### Example

- $N = 4$
  - Cache size 256KB
  - Cache line length 8 byte
  - $N_L = 32K$
  - $N_S = 8K$
  - $A = 134518704$
  - $i = 134518704 \bmod 8192 = 6064$
- ↓
- Search 4 lines in set 6064

## 5. Virtual Memory

- When we include disk in mem heir, referring to virtual memory
  - All users have full address space
  - Only active *pages* (like cache line) of memory actually kept in RAM
  - If address not in RAM, must be loaded from disk
  - Must have tables that map virtual addresses to physical
  - Tables are themselves kept in memory, and can be paged out
  - One reference can cause multiple memory heirarchy hits:
    1. Locate table holding address mapping (multiple hits)
    2. Load memory address
- ⇒ Machines have specialized cache for address translation called Translation Lookaside Buffer (TLB) – holds virtual/physical mapping for pages

## 6. Virtual Memory and TLB

- TLB usually buffers 32-256 pages
- If you exceed the TLB size in your working data set, expect a minimum of doubling of all memory access
- Page size controlled by OS, and is widely varied
- Small page sizes reduces memory waste
- Large page sizes helps with translation (& thus performance):
  1. Page table inversely prop to page size
  2. Not always zeroing sys mem
  3. Disk transfers more efficient for large sizes
  4. TLB misses go down

## 7. Memory Heirarchy Overview

ARCH	Regs	L1-I	L1-D	L2	L3	FP peak
USII	32	16/2	16/1	0.5-4MB	None	2
USIII	32	32/4	64/4	8 MB	None	2
Athlon	8	64	64	.25 - 1MB*	None	2
Opteron	8/16	64/2	64/2	1MB*	None	2/2;2/4
PIII	8	16	16	256-512 KB*	None	1/1;1/4
P4	8	??!	8'	.25-1MB*	None	1/1;2/4
P4E	8/16	??!	16	1MB	None	1/1;2/4
Itan2	128	32	32'	256KB*	1.5-9MB*	4

- ': not used by FPU
- \*: on-chip