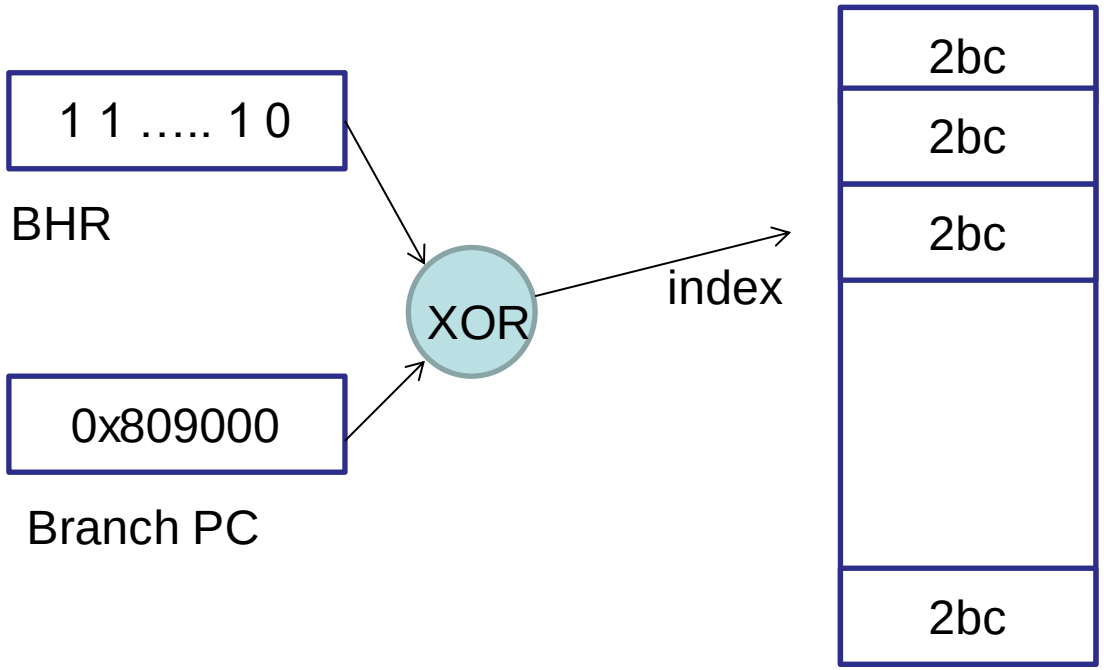
Pipelining

Misc

- ChatGPT
- Cheating
- Feedback loop

Gshare Branch Predictor



McFarling'93

Predictor size: 2^(history length)*2bit

Optional slide

G-Share Algorithm

```
predict_func(pc, actual_dir)
{
  index = pc xor BHR
   taken = 2bit_counters[index] > =2 ? 1 : 0
   correctly_predictied = (actual_dir == taken) ? 1 : 0  // stats
}

updated_func(pc, actual_dir)
{
  index = PC xor BHR
  if (actual_dir) SAT_INC( 2bit_counter[index] )
  else SAT_DEC ( 2bit_counter[index] )
  BHR = BHR << 1 | actual_dir
}
```

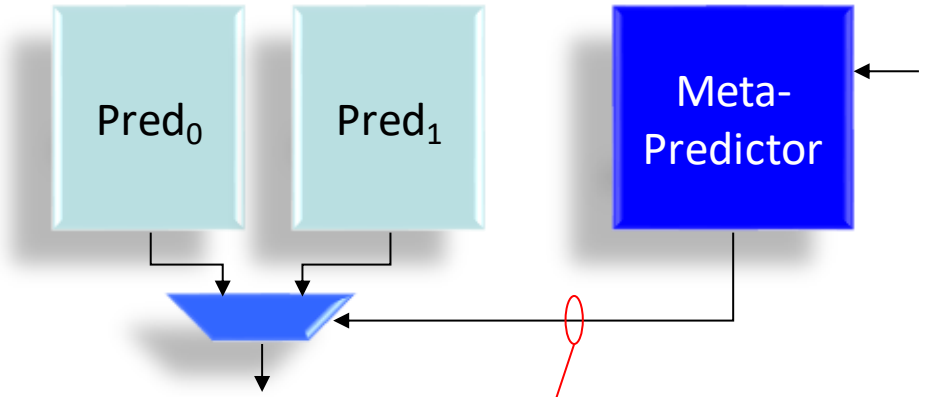
Optional slide

Tournament Predictor (Hybrid Predictor)

- No predictor is clearly the best
  - Different branches exhibit different behaviors
    - Some “constant”, some global, some local
- Idea:
  Let’s have a predictor to predict
  which predictor will predict better ☺

Optional slide

Tournament Predictor (Hybrid Predictor)



Pred$_0$	Pred$_1$	Meta Update
✗	✗	
✗	✓	
✓	✗	
✓	✓	

Optional slide

Common Combinations

- Global history + Local history
- “easy” branches + global history
  - gshare
- short history + long history

- Many types of behaviors, many combinations

Optional slide

Target Address Prediction

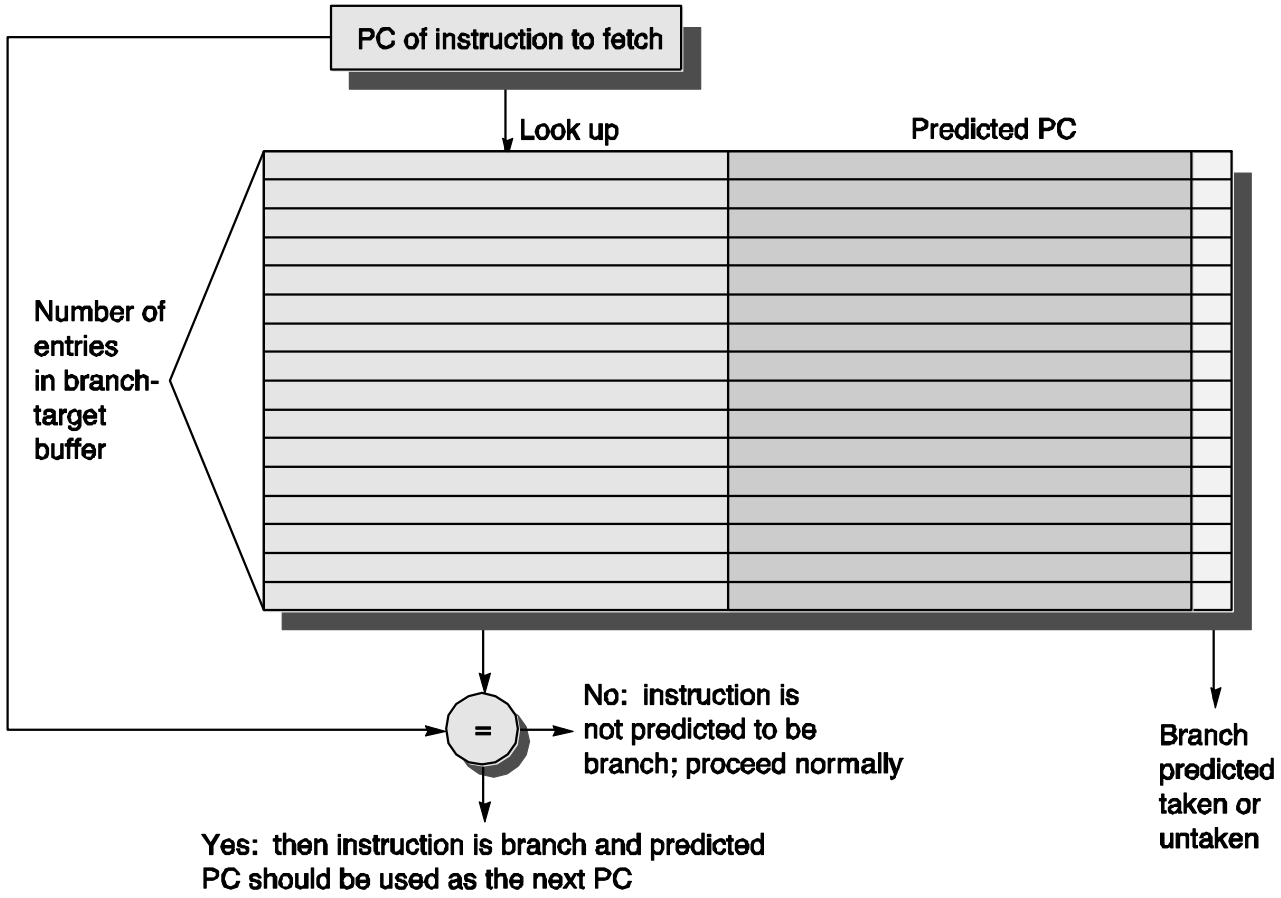
Q: Why do we need to predict target address?

- Branch Target Buffer (BTB)
  - Need target address one cycle after fetching a branch
  - For some branches (e.g., indirect) target known only after EX stage, which is way too late
  - Even easily-computed branch targets need to wait until instruction decoded and direction predicted in ID stage (still at least one cycle too late)
  - So, we have a fast predictor for the target that only needs the address of the branch instruction

Branch Target Buffer (BTB)

- Use PC (all bits) for lookup
  - Match implies this is a branch
- If match and predict bits: taken, set PC to predicted PC
- If branch predict wrong: must recover (same as branch hazards we've already seen)
- If decode indicates branch w/ no BTB match, two choices:
  - look up prediction now and act on it
  - just predict not taken
- When branch resolved, update BTB (at least prediction bits, maybe more)

Branch Target Buffer (BTB)

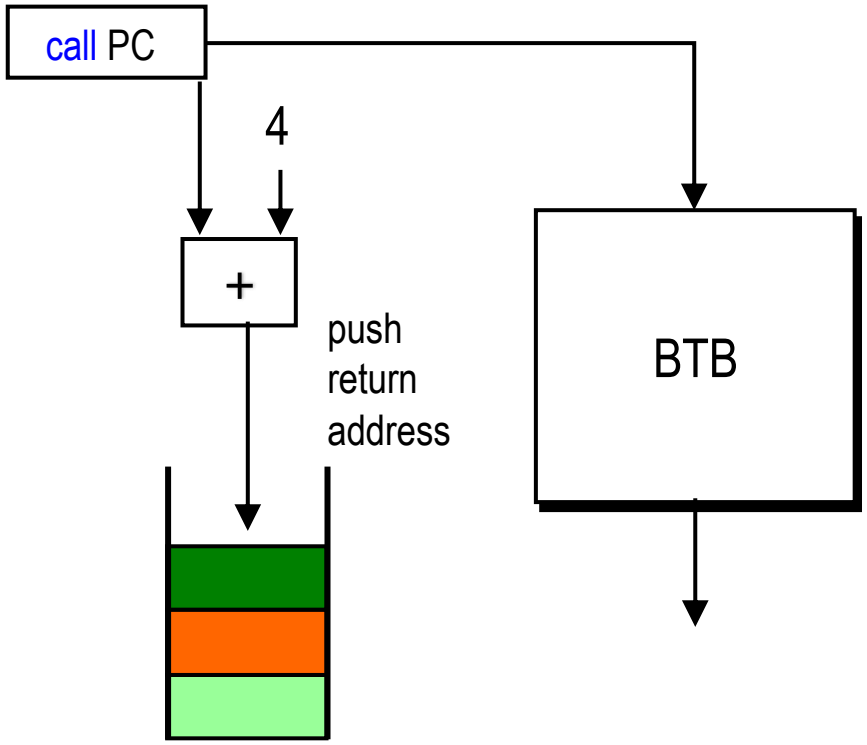


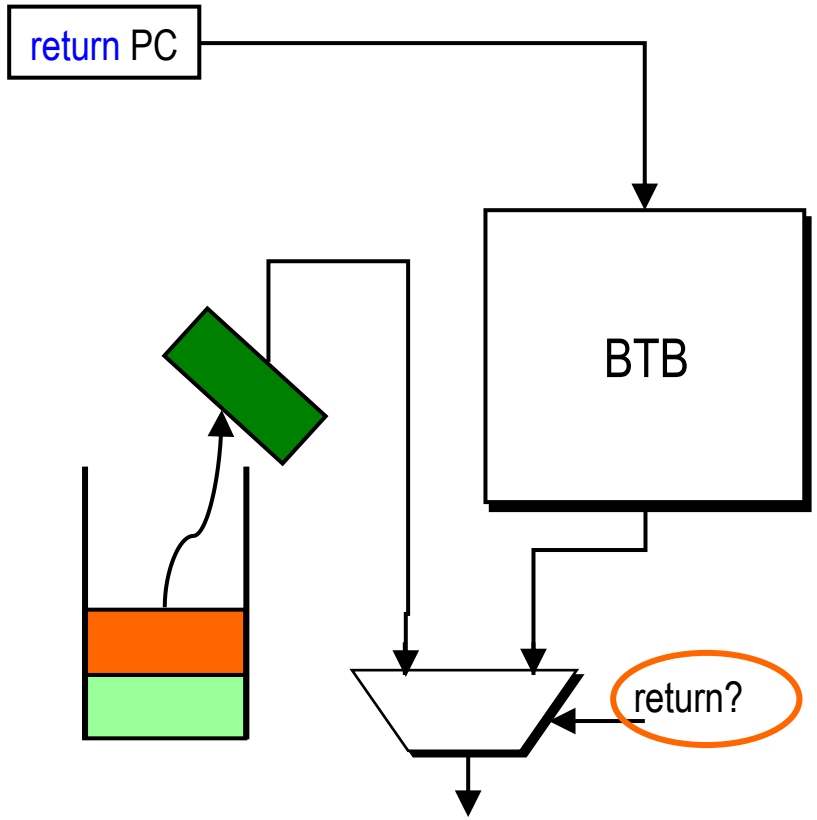
What about indirect jumps/returns?

- branch predictor does really well with conditional jumps
- BTB does really well with unconditional jumps (jump, jal, etc.)
- indirect jumps often jump to different destinations, even from the same instruction. Indirect jumps most often used for return instructions.
- procedure calls and returns
  - calls are always taken
  - return address almost always known
- return address stack (RAS)
  - on a procedure call, push the address of the instruction after the call onto the stack

Return Address Stack (RAS)

- May not know it is a return instruction prior to decoding
  - rely on BTB for speculation, fix once recognize return





Function Calls

```
    main()
    {
0x800   foo();
0x804   printf("still hungry\n");
        ....
0x900   foo();
0x904   printf("full\n");
    }
```

```
foo(){
  .....
  return
  }
```

0x804

Calculating the Cost of Branches

Factors to consider:

- branch frequency (every 4-6 instructions)
- correct prediction rate
  - 1 bit: ~ 80% to 85%
  - 2 bit: ~ high 80s to 90%
  - correlated branch prediction: ~ 95%
- misprediction penalty
  - Alpha 21164: 5 cycles; 21264: 7 cycles
  - UltraSPARC 1: 4 cycles
  - Pentium Pro: at least 9 cycles, 15 on average
- or misfetch penalty
  - have the correct prediction but not know the target address yet (may also apply to unconditional branches)

Calculating the Cost of Branches

What is the probability that a branch is taken?

- Given:
  - 20% of branches are unconditional branches
  - of conditional branches,
    - 66% forward branches are evenly split between taken & not taken
    - the rest backward branches are almost always taken

Calculating the Cost of Branches

What is the contribution to CPI of conditional branch stalls, given:

- 15% branch frequency (percentage)
- a BTB for conditional branches only w/
  - 10% miss rate
  - 3-cycle miss penalty
  - 92% prediction accuracy
  - 7 cycle misprediction penalty
- base CPI is 1

BTB result	Prediction	Frequency (per instruction)	Penalty (cycles)	*Stalls*
miss	--	.15 * .10 = .015	3	.045
hit	correct	.15 * .90 * .92 = .124	0	0
hit	incorrect	.15 * .90 * .08 = .011	7	.076
Total contribution to CPI				**.121**

Issues Affecting Accurate Branch Prediction

- Aliasing
  - More than one branch may use the same BHT/PHT entry
    - Constructive
      - Prediction that would have been incorrect, predicted correctly
    - Destructive
      - Prediction that would have been correct, predicted incorrectly
    - Neutral
      - No change in the accuracy

More Issues

- Training time
  - Need to see enough branches to uncover pattern
  - Need enough time to reach steady state
- “Wrong” history
  - Incorrect type of history for the branch
- Stale state
  - Predictor is updated after information is needed
- Operating system context switches
  - More aliasing caused by branches in different programs

“Real” Branch Predictors

- Alpha 21264
  - 8-stage pipeline, mispredict penalty 7 cycles
  - 64 KB, 2-way instruction cache with line and way prediction bits (Fetch)
    - Each 4-instruction fetch block contains a prediction for the next fetch block
  - Hybrid predictor (Fetch)
    - 12-bit GAg (4K-entry PHT, 2 bit counters)
    - 10-bit PAg (1K-entry BHT, 1K-entry PHT, 3-bit counters)

UltraSPARC-III

- 14-stage pipeline, bpred accessed in instruction fetch stages 2-3
- 16K-entry 2-bit counter Gshare predictor
  - Bimodal predictor which XOR's PC bits with global history register (except 3 lower order bits) to reduce aliasing
- Miss queue
  - Halves mispredict penalty by providing instructions for immediate use

Pentium III

- Dynamic branch prediction
  - 512-entry BTB predicts direction and target, 4-bit history used with PC to derive direction
- Static branch predictor for BTB misses
- Return Address Stack (RAS), 4/8 entries
- Branch Penalties:
  - Not Taken: no penalty
  - Correctly predicted taken: 1 cycle
  - Mispredicted: at least 9 cycles, as many as 26, average 10-15 cycles

AMD Athlon K7

- 10-stage integer, 15-stage fp pipeline, predictor accessed in fetch
- 2K-entry bimodal, 2K-entry BTAC
- 12-entry RAS
- Branch Penalties:
  - Correct Predict Taken: 1 cycle
  - Mispredict penalty: at least 10 cycles

Takeaways

Branch Hazard

Types of branches

Approaches to resolving branch hazard

Branch Prediction

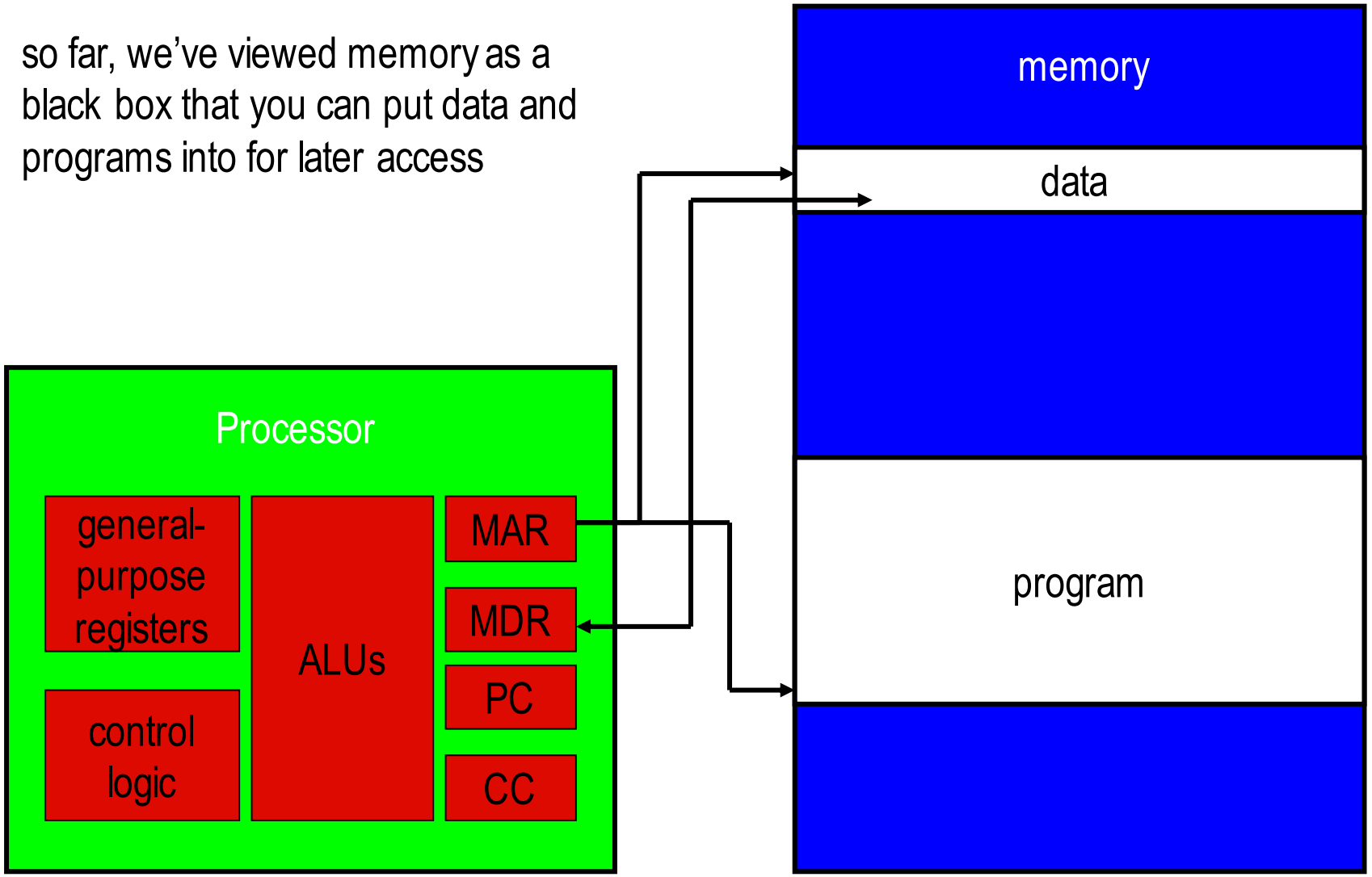
Branch Target Buffer

Return Address Stack

Memory Hierarchy

Physical Memory Systems

so far, we've viewed memory as a black box that you can put data and programs into for later access



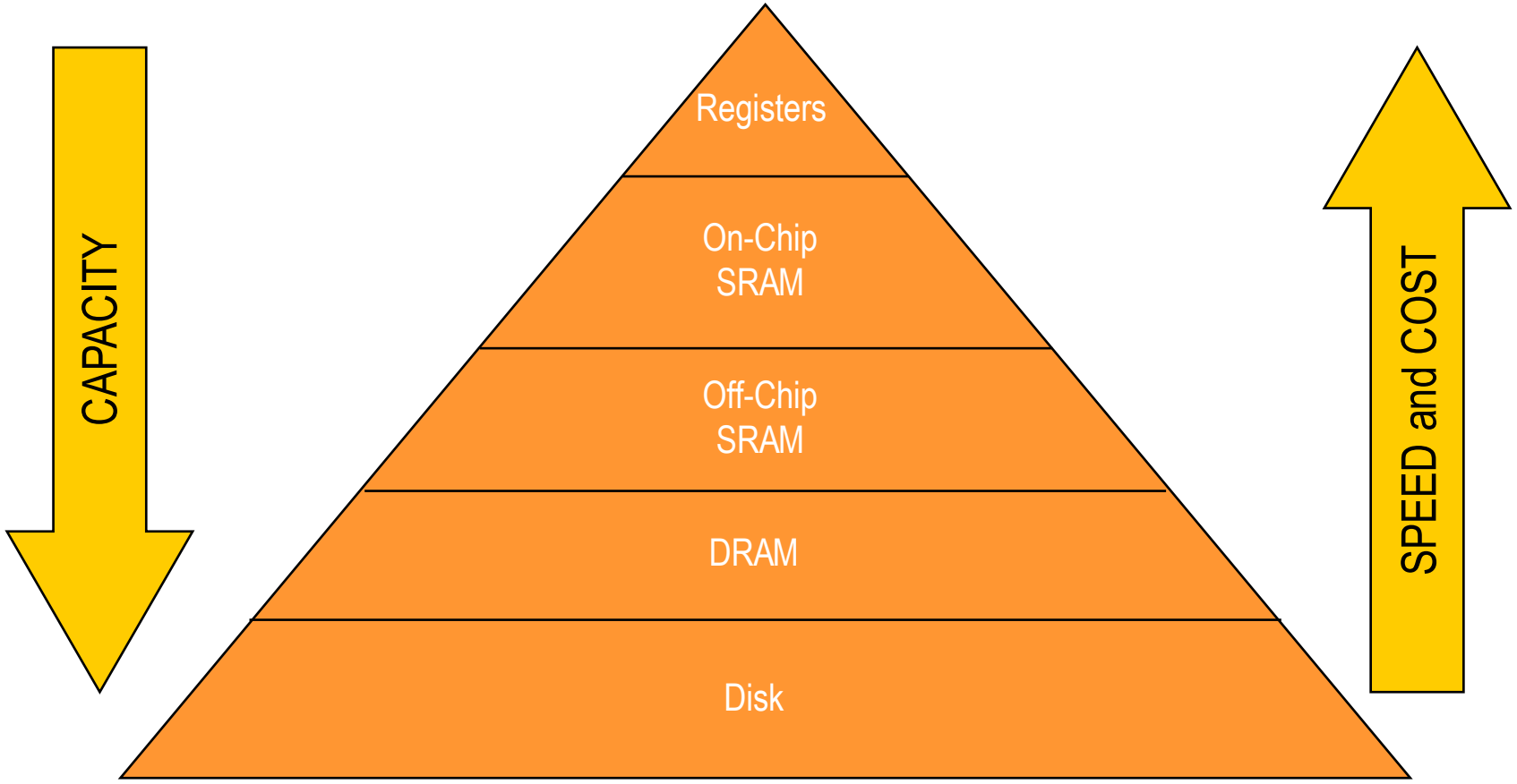
Model of Memory Hierarchy

Types of Memories

- SSD: $0.75 /GB, HD: $0.1-0.2/GB
- DRAM: $20-25/GB
- more on bandwidth later

Type	Size	Latency	Cost/bit
Register	< 1KB	< 1ns	$$$$
On-chip SRAM	8KB-6MB	< 2ns	$$$
Off-chip SRAM	1Mb – 16Mb	< 10ns	$$
DRAM	64MB – 1TB	< 100ns	$
Disk (SSD, HD)	40GB – 1PB	< 20ms	< $1/GB

Memory Hierarchy

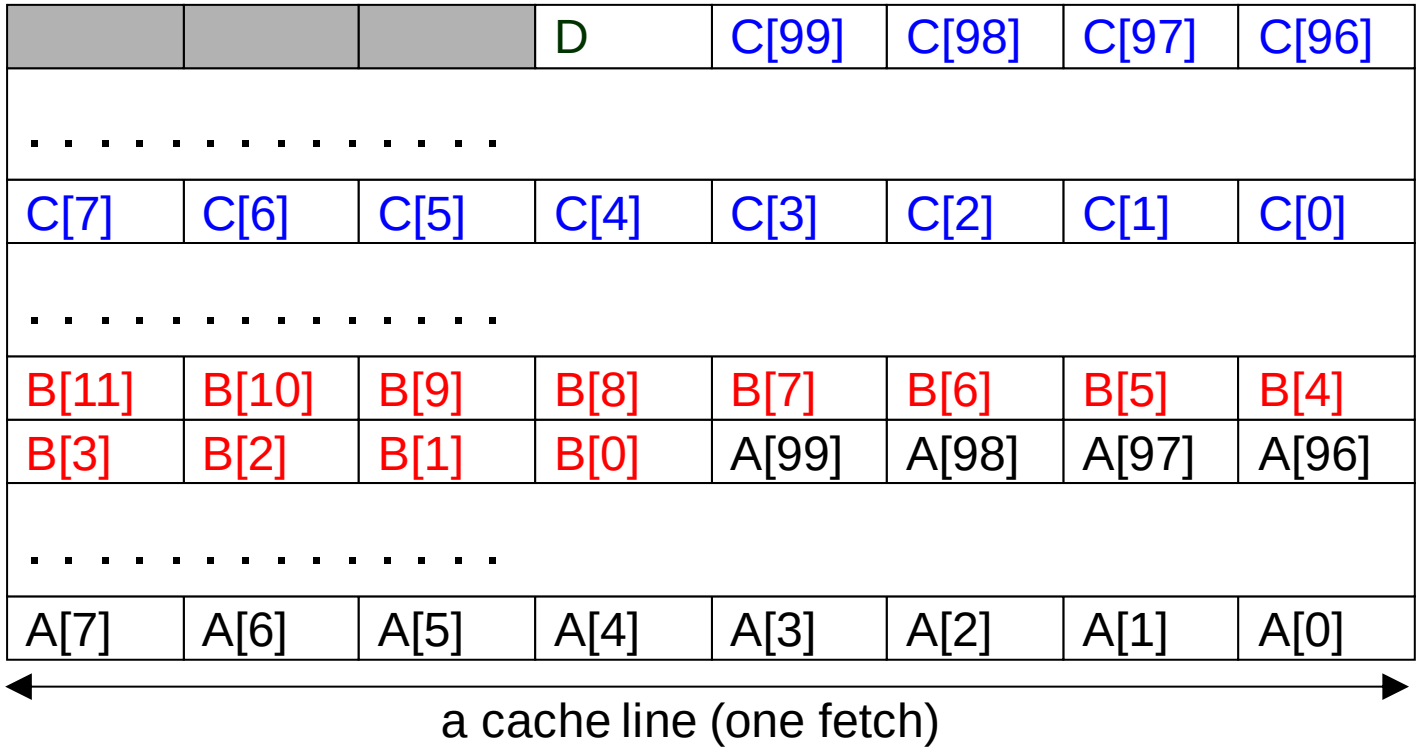


Why Does a Hierarchy Work?

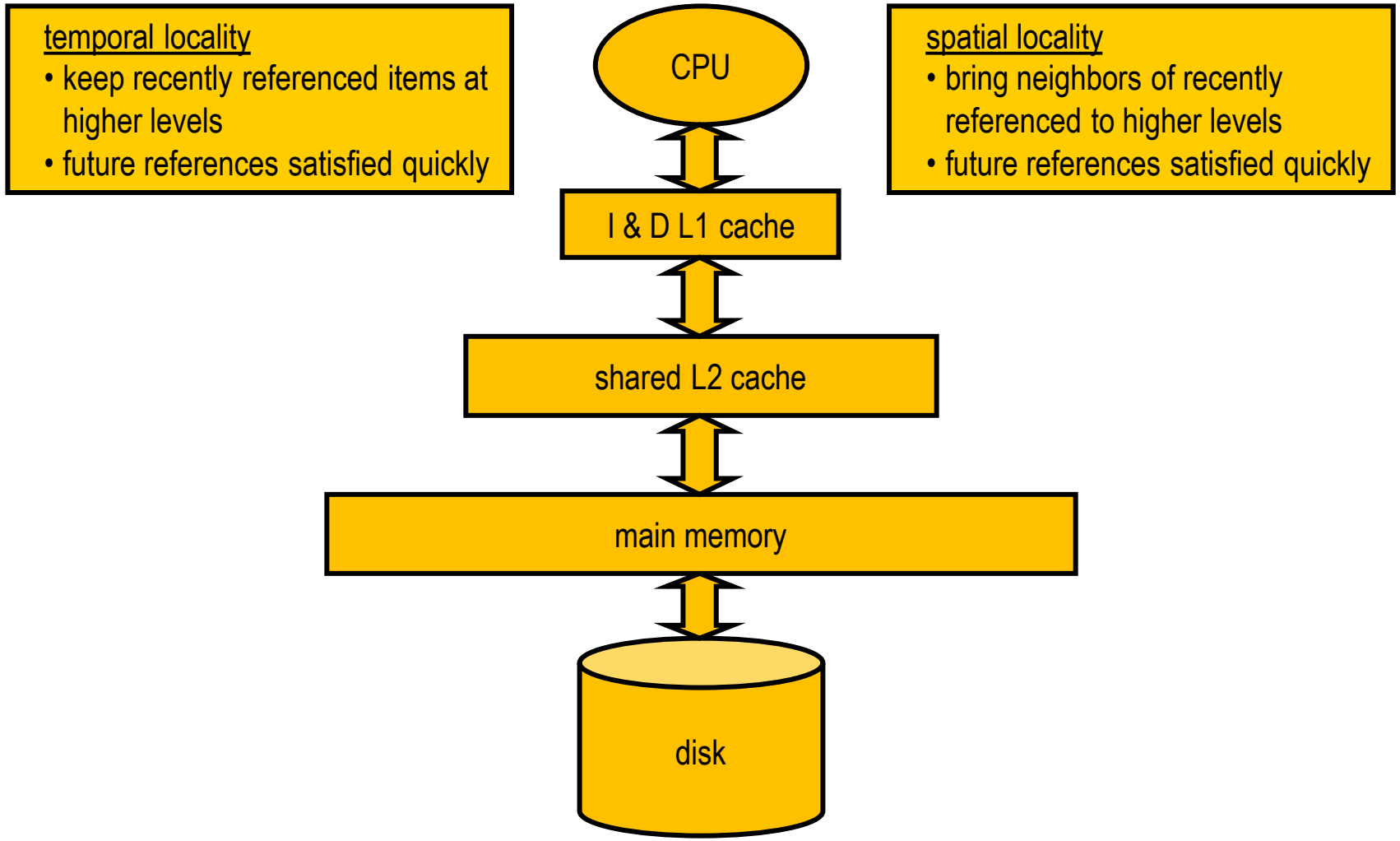
- locality of reference
  - ✓ temporal locality
    - reference same memory location many times (close together, in time)
  - ✓ spatial locality
    - reference near neighbors around the same time

Example of Locality

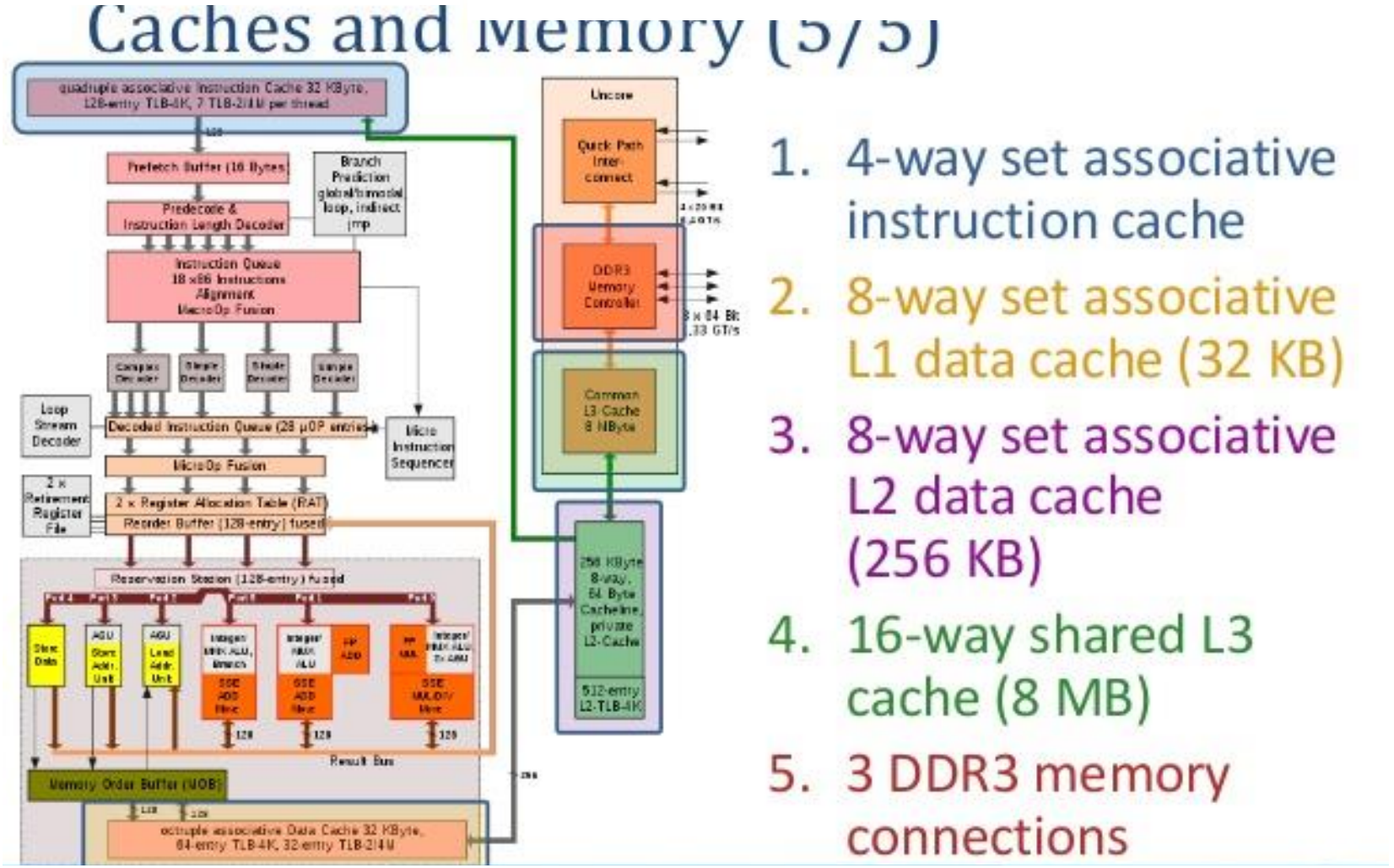
```
int A[100], B[100], C[100], D;
for (i=0; i<100; i++) {
        C[i] = A[i] * B[i] + D;
}
```



Memory Hierarchy

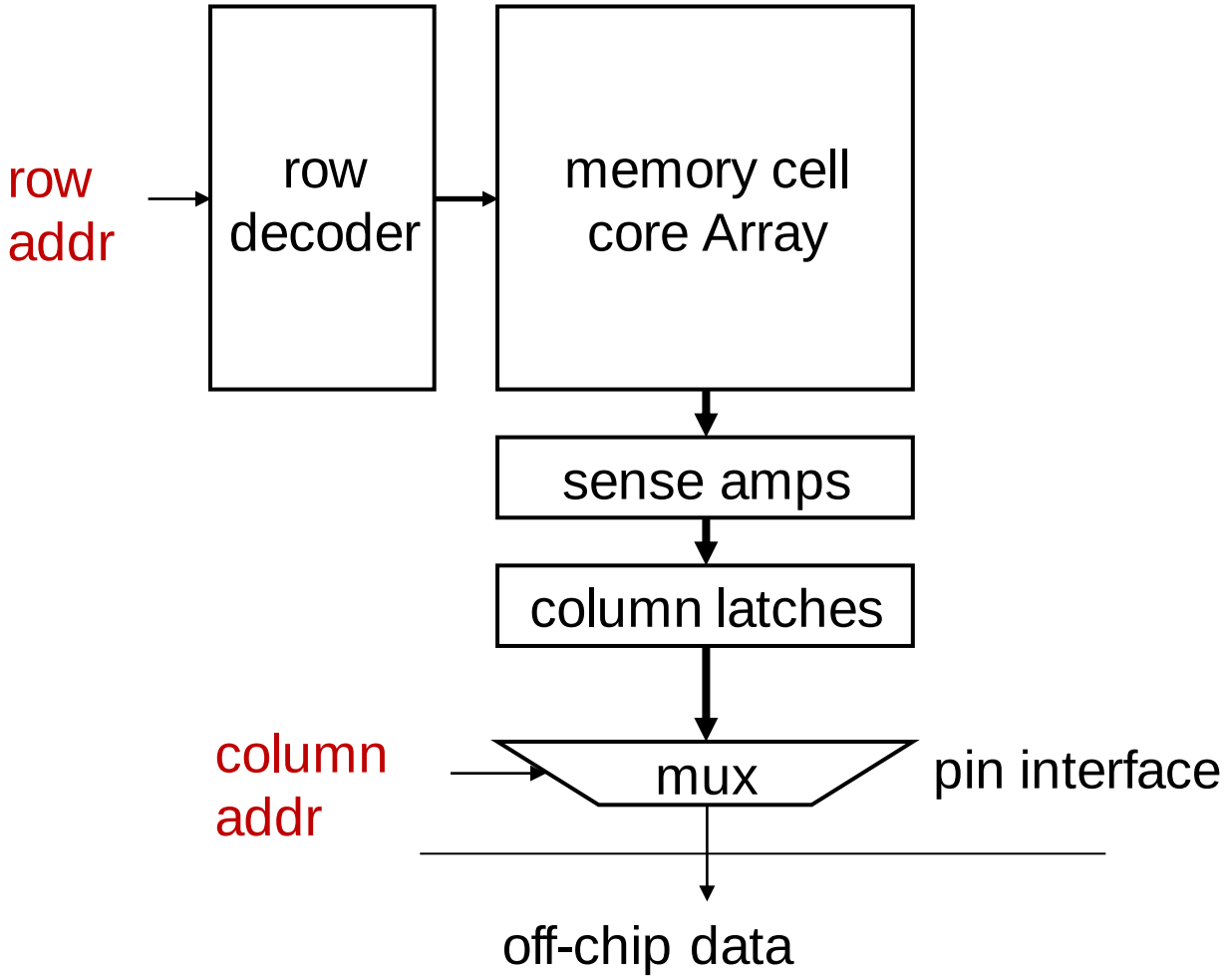


Example: Intel Nehalem Memory Hierarchy

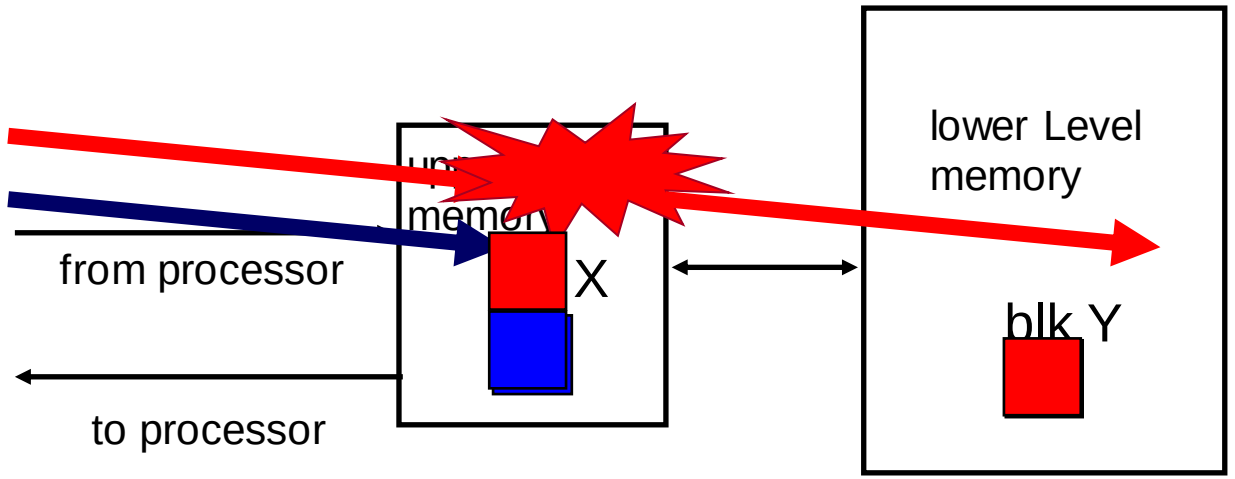


1. 4-way set associative instruction cache
2. 8-way set associative L1 data cache (32 KB)
3. 8-way set associative L2 data cache (256 KB)
4. 16-way shared L3 cache (8 MB)
5. 3 DDR3 memory connections

Typical Memory Organization



Basic Cache Operation



Cache Terminology

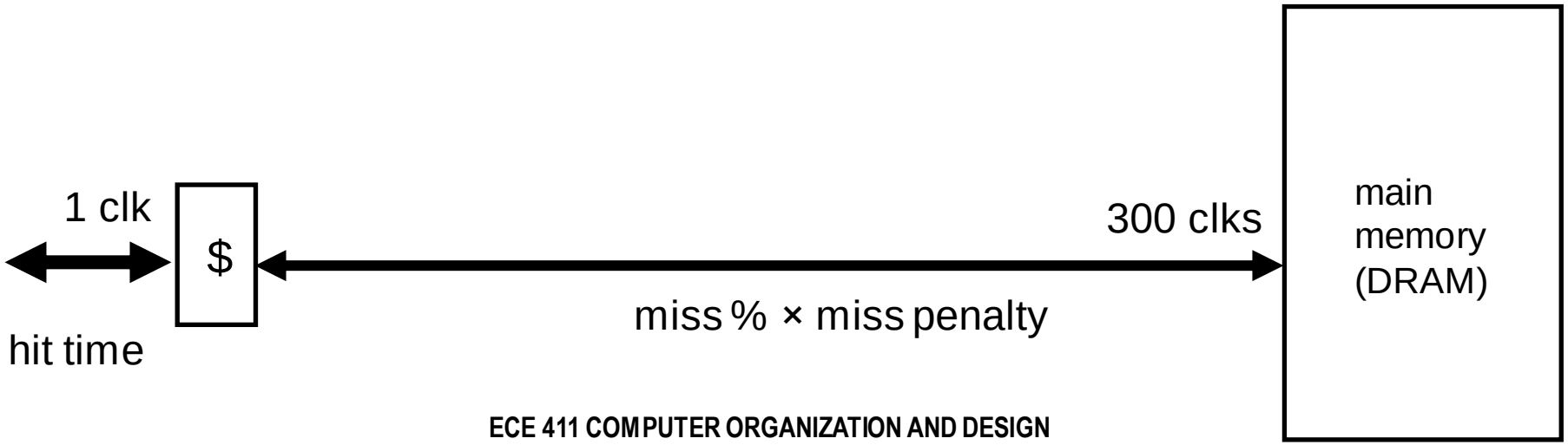
- hit: data appears in some block
  - ✓ hit rate: the fraction of accesses found in the level
  - ✓ hit time: time to access the level (consists of RAM access time + time to determine hit)
- miss: data needs to be retrieved from a block in the lower level (e.g., block Y)
  - ✓ miss rate = 1 - (hit rate)
  - ✓ miss penalty: time to replace a block in the upper level + time to deliver the block to the processor
- hit time << miss penalty

Average Memory Access Time

- average memory-access time
  = hit time + miss rate x miss penalty
- miss penalty: time to fetch a block from lower memory level
  - ✓ access time: function of latency
  - ✓ transfer time: function of bandwidth b/w levels
    - transfer one "cache line/block" at a time
    - transfer at the size of the memory-bus width

Memory Hierarchy Performance

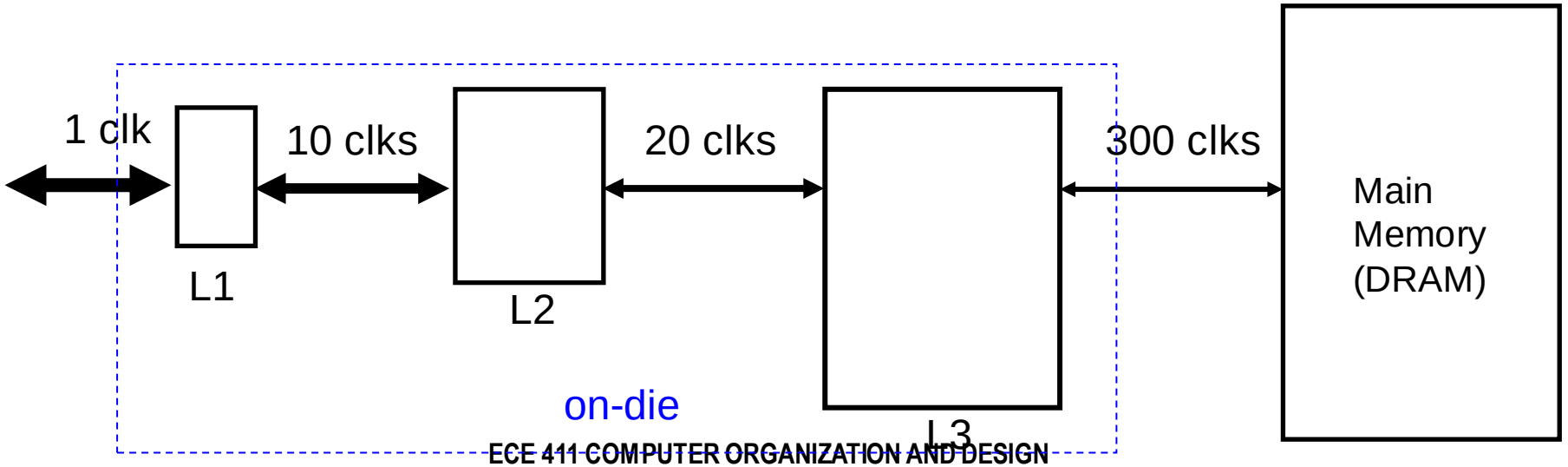
- Average Memory Access Time (AMAT)
  - ✓ = hit time + miss rate × miss penalty
  - ✓ = T_{hit}(L1) + miss%(L1) × T(memory)
- example:
  - ✓ cache hit = 1 cycle
  - ✓ miss rate = 10% = 0.1
  - ✓ miss penalty = 300 cycles
  - ✓ AMAT = 1 + 0.1 × 300 = 31 cycles
- can we improve it?



Reducing Penalty: Multi-Level Cache

- Average Memory Access Time (AMAT)

$$= T_{hit}(L1) + miss\%(L1) \times (T_{hit}(L2) + miss\%(L2) \times (T_{hit}(L3) + miss\%(L3) \times T(memory)))$$
$$= T_{hit}(L1) + miss\%(L1) \times T_{miss}(L1)$$
$$= T_{hit}(L1) + miss\%(L1) \times \{T_{hit}(L2) + miss\%(L2) \times (T_{miss}(L2)\}$$
$$= T_{hit}(L1) + miss\%(L1) \times \{T_{hit}(L2) + miss\%(L2) \times (T_{miss}(L2)\}$$
$$= T_{hit}(L1) + miss\%(L1) \times \{T_{hit}(L2) + miss\%(L2) \times [T_{hit}(L3) + miss\%(L3) \times T(memory)]\}$$



AMAT Example

$$= T_{hit}(L1) + miss\%(L1) \times (T_{hit}(L2) + miss\%(L2) \times (T_{hit}(L3) + miss\%(L3) \times T(memory)))$$

- Example:
  - ✓ miss rate L1=10%, $T_{hit}(L1)$ = 1 cycle
  - ✓ miss rate L2=5%, $T_{hit}(L2)$ = 10 cycles
  - ✓ miss rate L3=1%, $T_{hit}(L3)$ = 20 cycles
  - ✓ T(memory) = 300 cycles
- AMAT = ?
  - ✓ 2.115 (compare to 31 with no multi-levels)
  - ✓ 14.7× speed-up!

A memory system consists of a cache and a main memory. If it takes 1 cycle to complete a cache hit, and 100 cycles to complete a cache miss, what is the average memory access time if the hit rate in the cache is 97%?

- A memory system has a cache, a main memory, and a virtual memory. If the hit rate in the cache is 98% and the hit rate in the main memory is 99%, what is the average memory access time if it takes 2 cycles to access the cache, 150 cycles to fetch a line from main memory, and 100,000 cycles to access the virtual memory?