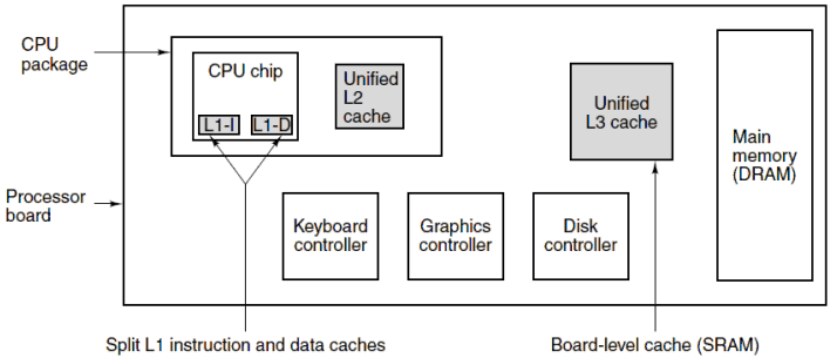
Ch 4.5 Improving performance

4 basic techniques:

1. Caches - faster access to data
2. Branch prediction - (smart) guessing at next instruction
3. Out-of-order execution
4. Speculative execution

Caches - bring blocks of data closer to the action (the datapath); closer = faster access
Caches only work because of 1) spatial locality, 2) temporal locality.
Definitions: split v. unified cache, cache hit, cache miss, cache lines (fixed size mem blocks)
/* You probably get more cache-talk in CSC 420, Operating Systems */
Two flavors: direct-mapped cache, N-way set-associative cache



Branch prediction - branches (if stmts, loops) can stall pipelining
Simple heuristic: Assume all backward branches will be taken and forward branches will not (if case, loops!); if you get it wrong, then undo (complex!)

Try to do better than this:

- Static branch prediction - by compiler, example: loop a milion times
- Dynamic branch prediction - during execution, keep track of last branch taken

Out-of-order execution - while waiting, execute non-dependent instructions “out of order”
Superscalar architecture - multiple functional units (ALU, mem access) in datapath

RAW dependence - read after write, waiting for register to be stored (written)
WAR dependence - write after read, waiting for register read so that value can be over-written

Retired - instruction complete
Register renaming - shadow/secret registers used to bypass dependency conflicts

Speculative execution - executing code before it is known to be needed
Basic block - linear sequence of code without branching; usually short, which hinders pipeline

hoisting - moving slow operations above a branch; compiler duty, not hardware

Sometimes cheaper to execute both branches (a, b) than to wait and choose

```
if( cond)
   a;
else
   b;
```

“Speculative execution introduces some interesting problems.”