**cache hit:** CPU needs item in cache (fast)

**cache miss:** CPU needs item not in cache
  — item loaded into cache for future use, replacing some other item

**optimal replacement:** on cache miss, loaded item replaces item that will not be needed for the *longest time in the future*

[ more realistic scheme: **LRU replacement** — replace least recently used item
  — provably within small constant factor of optimal, but much harder to analyze ]

**fully associative** — any item in memory can go anywhere in the cache
  [ real caches have limited associativity, which causes “unlucky” memory-access patterns to go same place in cache
  …effectively shrinks cache in these cases ]

**temporal locality** — same item is re-used for several computations that are close to one another in time ⇒ still in-cache ⇒ efficient

[ there is also **spatial locality** — items close to one another in main memory are used close in time … exploited by **cache lines**, TBD ]

**cache complexity** — the number of cache misses \( Q(n; Z) \) required for a given algorithm running on a problem of size \( n \) with cache of size \( Z \)
  … usually only given as **asymptotic** result for large \( n, Z \), ignoring constant factors

**asymptotic notation:**
  we say a function \( f(n) \) is \( O(g(n)) \) if \( g(n) \) is an **asymptotic upper bound** for \( f(n) \), ignoring constant factors. Technically, if \( f(n) < C \cdot g(n) \) for some constant \( C \) for all sufficiently large \( n \) (i.e., for all \( n > N \) for some \( N \))

  we say a function \( f(n) \) is \( \Omega(g(n)) \) if \( g(n) \) is an **asymptotic lower bound** for \( f(n) \), ignoring constant factors. Technically, if \( f(n) > C \cdot g(n) \) for some constant \( C \) for all sufficiently large \( n \) (i.e., for all \( n > N \) for some \( N \))

  we say a function \( f(n) \) is \( \Theta(g(n)) \) if \( g(n) \) is an **asymptotic tight bound** for \( f(n) \), ignoring constant factors. Technically, if \( f(n) \) is both \( O(g(n)) \) and \( \Omega(g(n)) \)