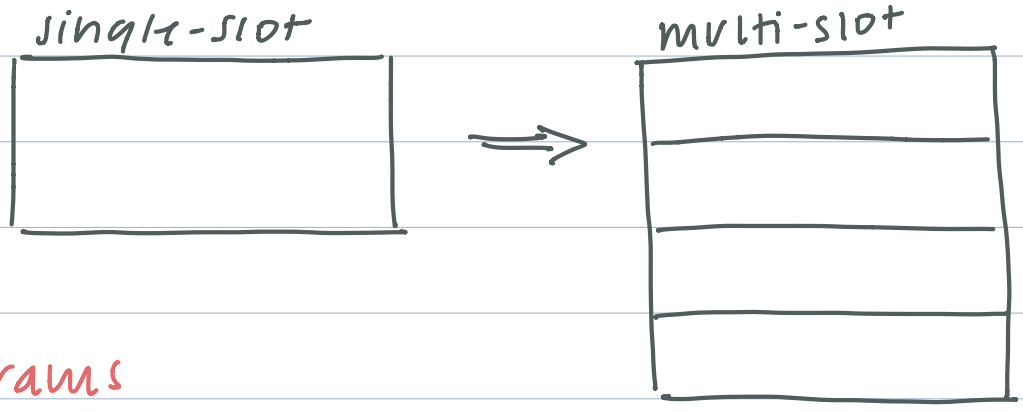
CS61 Lecture 18: Storage 3

October 31, 2019

Continue designing & modeling caches

single slot → multislot caches

Multislot Cache



params

- int nslots (number of slots in cache)
- bool is_direct_mapped
- char replacement (replacement algo if fully associative)
- cache_slot[]
  - valid (false if empty)
  - mindex (index into data array, similar to address)
  - last_access (helps w/ replacement)

Replacement Scheme

1. Random
2. Least Recently Used

Direct-Mapped Cache

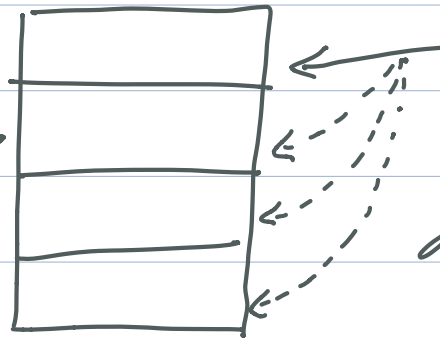
Ref A → f

only one slot that a certain reference can go into

Fully-Associative Cache

Ref A → f

could be put anywhere



Summary: How caches ↑ perform

	read	write
1. PREFETCHING	☑	☐
2. BATCHING	☑	☑

1. PREFETCHING

bring data closer before it's needed

- eg: prefetch file from disk when opened

works well when user ops are anticipated correctly

2. BATCHING

change a large number of small requests into small number of large requests

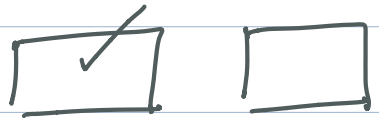
- eg: sequentially reading 4096 bytes from disk
- eg: stdio cache and file cache faster by writing in chunks

works well when per-request cost R is high and per-unit cost U is low

$C = NU + R$

locality benefits

3. WRITE COALESCING



eliminate all but last write

- eg: index variable in loop

works well for technologies where write is complex (ex: flash memory)

Unix File (a sequence of bytes)

1. ones that you can read like a book
   RANDOM ACCESS FILES in Unix
2. ones that are like listening to your grandmother tell a story
   STREAMS in Unix

	RAM File	Stream File
length	finite	possibly infinite
skip around	yes (file seekable) ← seek call	no (not seekable)
capture in memory?	yes (mappable) ← map call	no (not mappable)

File Descriptor

- threads / remembers the current file position
  - read (fd, buf, len)
  - write (fd, buf, len)
  - lseek (