CPSC 313: Computer Hardware and Operating Systems

Unit 3: Caching

Speedup

Administration

- Quiz 3 is next week (sign up for Quiz 3, viewing, and retake times!)
- Lab 6: due shortly
- Lab 7: out Friday (not as big as Lab 5, but still substantial)
- Tutorial 6: this week
- Drop with W deadline on Friday: We want you in the class! (But… if you're certain for some reason that you want to drop, don't miss the deadline!)

As always: Check the syllabus for details and deadlines!

Today

- Learning Objectives
  - Apply Amdahl's law to evaluate cache performance by calculating speedup
  - Apply caching principles to writes
- Reading
  - Pages 622-24
  - Section 6.4.5

Part 1: Evaluating Caches

- By now you should be pretty good at calculating hit/miss rates and average *read* access times.
- Sometimes we also want to compare different cache architectures or determine how a change might influence performance.
- We often use speedup to describe how much a change improves (or harms) performance.

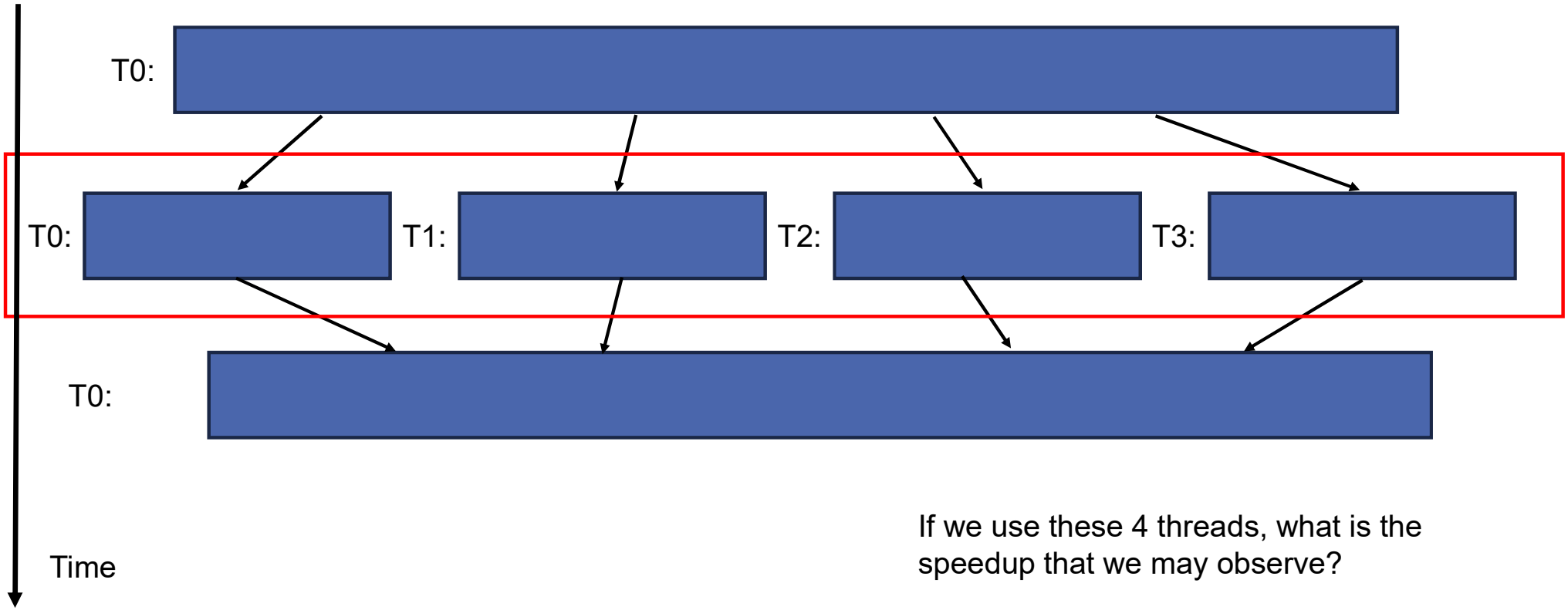
Amdahl's Law

- Who is Amdahl and what is his law?
  - Gene Amdahl : American computer architect who worked at IBM for many years and then started Amdahl Corporation.

- Amdahl's Law:
  - Originally designed to quantify the opportunities and benefits of *parallelism*.

If you have a program that has a sequential part and a parallelizable part, when you parallelize it, you will only improve the performance of the latter part. So, it's good to know just how much parallelizing the application will help.

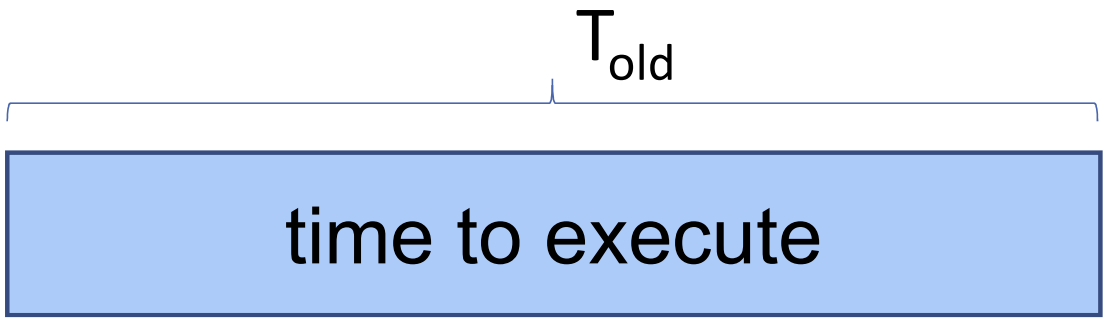
Parallel Merge Sort



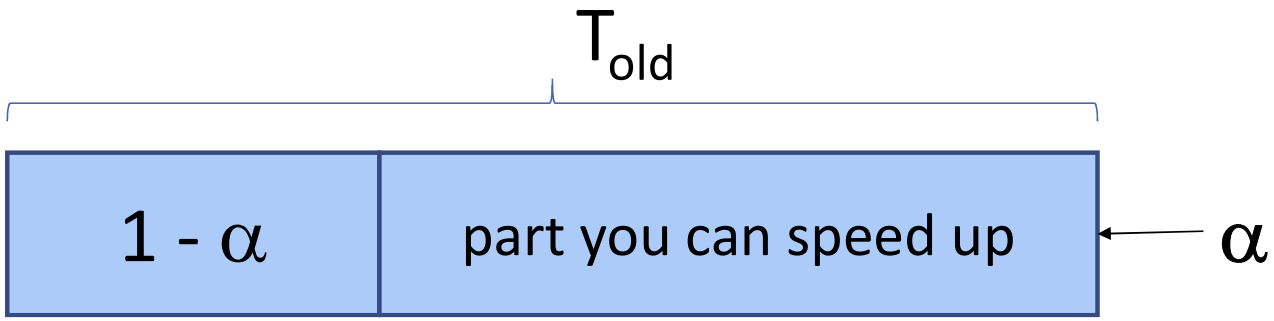
If we use these 4 threads, what is the speedup that we may observe?

What if we put in 8 or 256 threads?

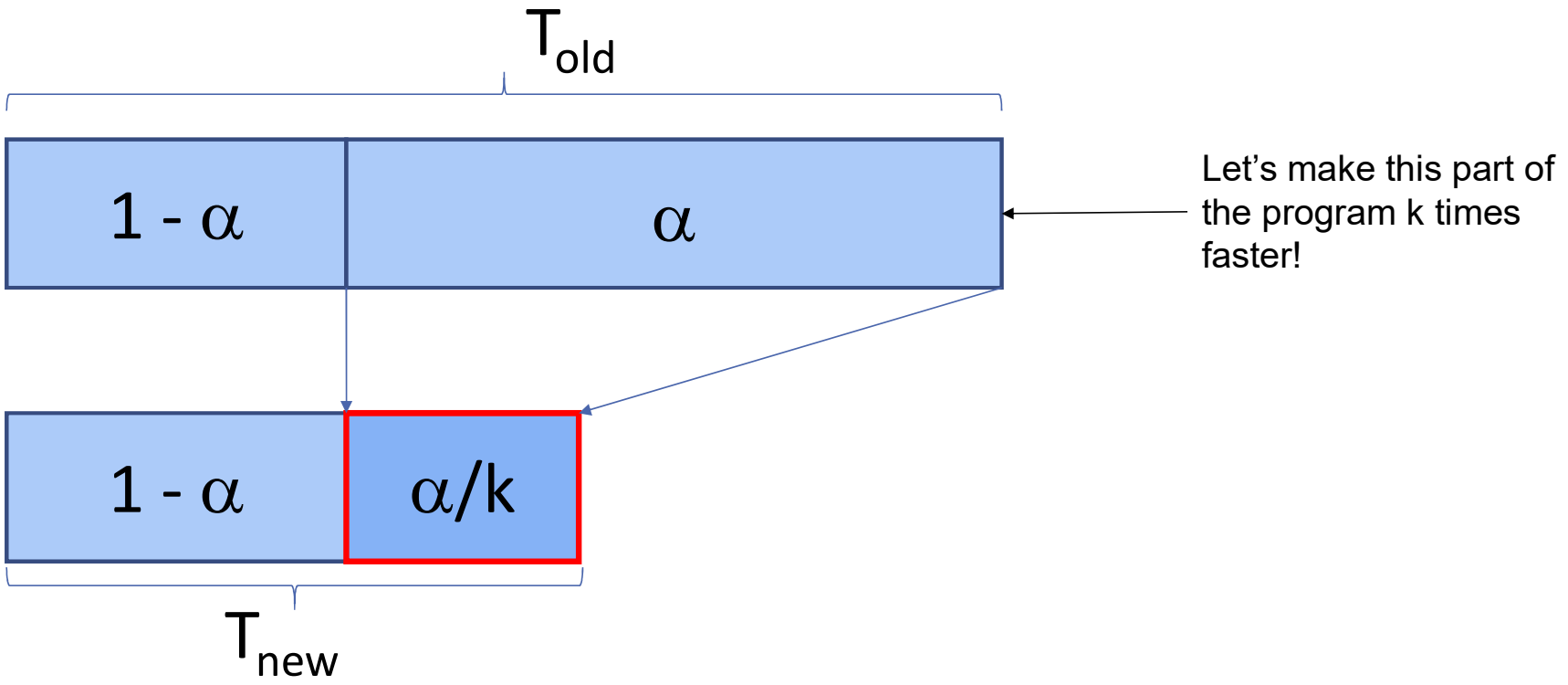
Amdahl's Law Visually



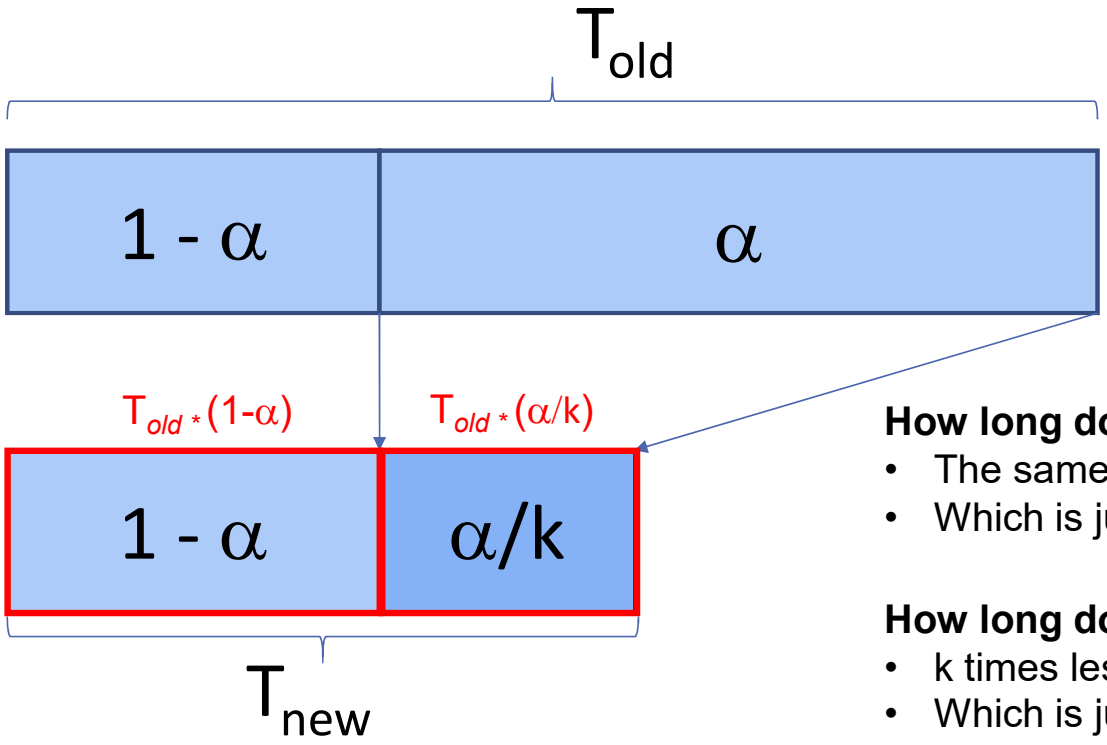
Amdahl's Law Visually



Amdahl's Law Visually



Amdahl's Law Visually -- Computing T_{new}



How long does it take for the part that is unchanged?

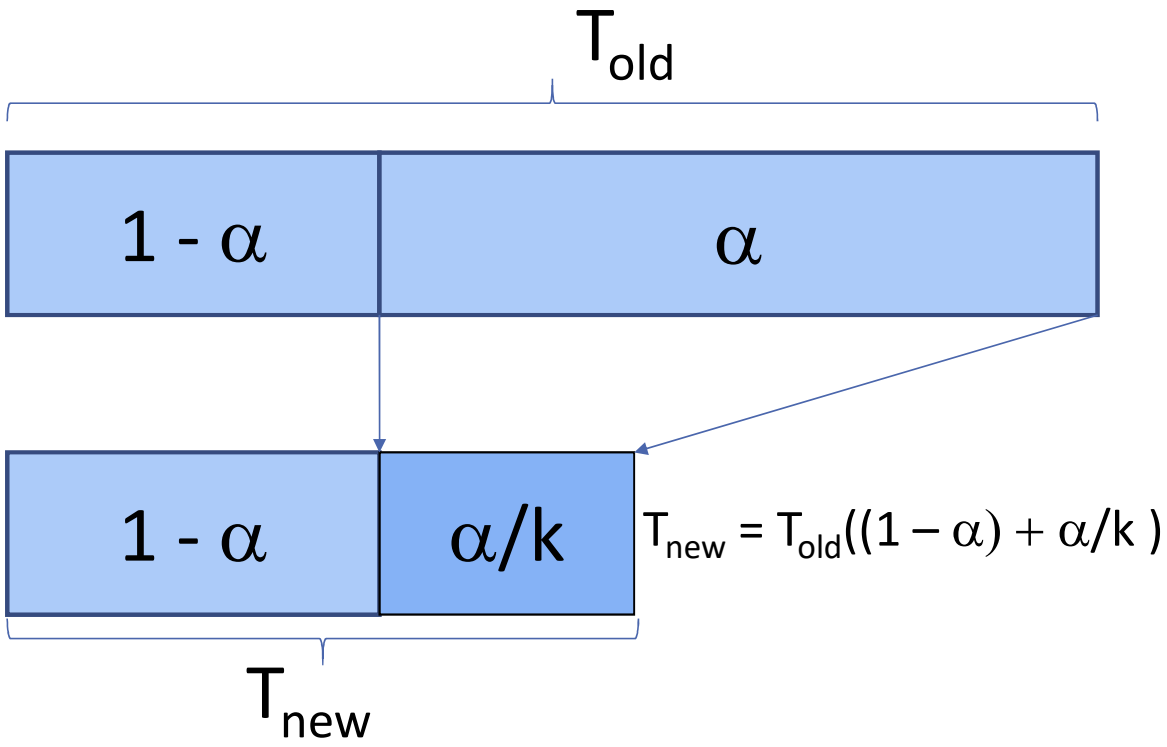
- The same amount of time it took originally
- Which is just $T_{old} * (1-\alpha)$

How long does it take for the part that is changed?

- k times less than it took originally
- Which is just $T_{old} * (\alpha/k)$

$$T_{new} = T_{old}(1-\alpha) + T_{old}(\alpha/k) = T_{old}((1-\alpha) + \alpha/k)$$

Amdahl's Law Visually -- Computing Speedup



Speedup = $\frac{T_{old}}{T_{new}}$

Speedup = $\frac{T_{old}}{\left(1-\alpha+\frac{\alpha}{k}\right)T_{old}}$

Speedup = $\frac{1}{\left(1-\alpha+\frac{\alpha}{k}\right)}$

Amdahl's Law more Formally

- Amdahl's Law States: Overall speedup is a function of both the amount you improve some part of the system (k) and the fraction of time you spend in that part of the system (α).
- Given:
  - Latency (i.e., time to execute) of the original system is T_{old}
  - You can speed up part of a system by a factor of *k*
  - The program spends α (< 1) of its time in the part of the system you're going to speed up.
- Latency after the improvement:

$$T_{new} = T_{old}(1 - \alpha) + T_{old}(\alpha/k) = T_{old}((1 - \alpha) + \alpha/k)$$

Amdahl's Law Example for Caching

$$\text{Speedup} = \frac{1}{\left(1 - \alpha + \frac{\alpha}{k}\right)}$$

- Suppose that a program that takes 100 seconds to complete spends 25% of its time in memory instructions. If we make memory instructions twice as fast, how much faster will the program run?
  - α =
  - k =
  - T_{old} =
  - Speedup =
  - Latency =

Amdahl's Law Example (1)

$$\text{Speedup} = \frac{1}{\left(1 - \alpha + \frac{\alpha}{k}\right)}$$

- Suppose that a program that takes 100 seconds to complete spends 25% of its time in memory instructions. If we make memory instructions twice as fast, how much faster will the program run?
  - α = .25
  - k =
  - T_{old} =
  - Speedup =
  - Latency =

Amdahl's Law Example (1)

$$\text{Speedup} = \frac{1}{\left(1 - \alpha + \frac{\alpha}{k}\right)}$$

- Suppose that a program that takes 100 seconds to complete spends 25% of its time in memory instructions. If we make memory instructions twice as fast, how much faster will the program run?
  - α = .25
  - k = 2
  - T_{old} =
  - Speedup =
  - Latency =

Amdahl's Law Example (1)

$$\text{Speedup} = \frac{1}{\left(1 - \alpha + \frac{\alpha}{k}\right)}$$

- Suppose that a program that takes 100 seconds to complete spends 25% of its time in memory instructions. If we make memory instructions twice as fast, how much faster will the program run?
  - α = .25
  - k = 2
  - T_{old} = 100
  - Speedup =
  - Latency =

Amdahl's Law Example (1)

Speedup = $\frac{1}{\left(1-\alpha+\frac{\alpha}{k}\right)}$

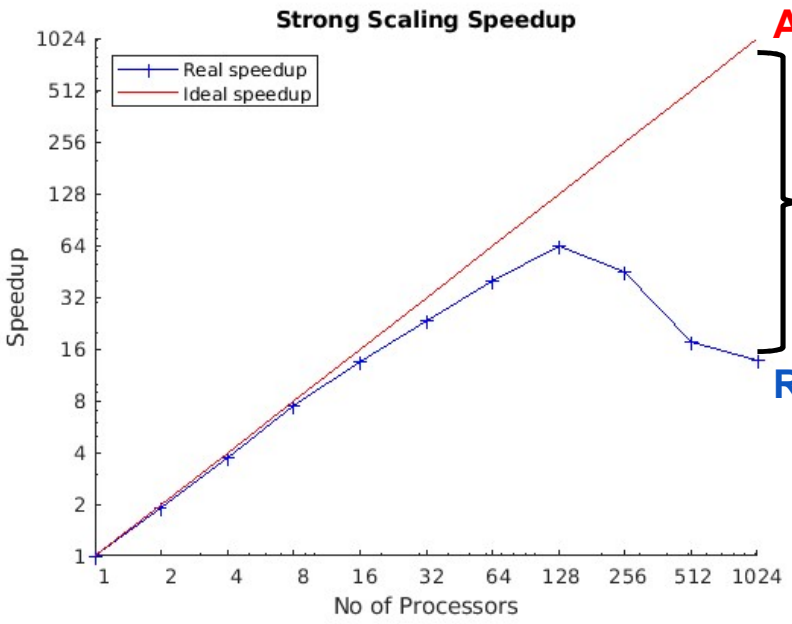
- Suppose that a program that takes 100 seconds to complete spends 25% of its time in memory instructions. If we make memory instructions twice as fast, how much faster will the program run?
  - α = .25
  - k = 2
  - T_{old} = 100
  - Speedup = $\frac{1}{\left(1-.25+\frac{.25}{2}\right)} = \frac{1}{.875}$ = 1.14
  - Latency =

Amdahl's Law Example (1)

$$\text{Speedup} = \frac{1}{\left(1 - \alpha + \frac{\alpha}{k}\right)}$$

- Suppose that a program that takes 100 seconds to complete spends 25% of its time in memory instructions. If we make memory instructions twice as fast, how much faster will the program run?
  - α = .25
  - k = 2
  - T_{old} = 100
  - Speedup = $\frac{1}{\left(1 - .25 + \frac{.25}{2}\right)} = \frac{1}{.875}$ = 1.14
  - Latency = 100 * .875 = 87.5 seconds

In reality, it's not that simple...



Amdahl's Law (theoretical)

Overheads:
- Resource Contention
- Synchronization
- Communication Cost
- Data Transfer Cost

Real Performance (actual)

https://hpc-wiki.info/hpc/Scaling

Amdahl's Law Example (2)

$$\text{Speedup} = \frac{1}{\left(1 - \alpha + \frac{\alpha}{k}\right)}$$

- Suppose that a program that takes 100 seconds to complete spends 33% of its time in memory instructions. If we add a cache that makes memory instructions four times faster, how much faster will the program run?
  - α =
  - k =
  - T_{old} =
  - Speedup =
  - Latency =

Amdahl's Law Example (2)

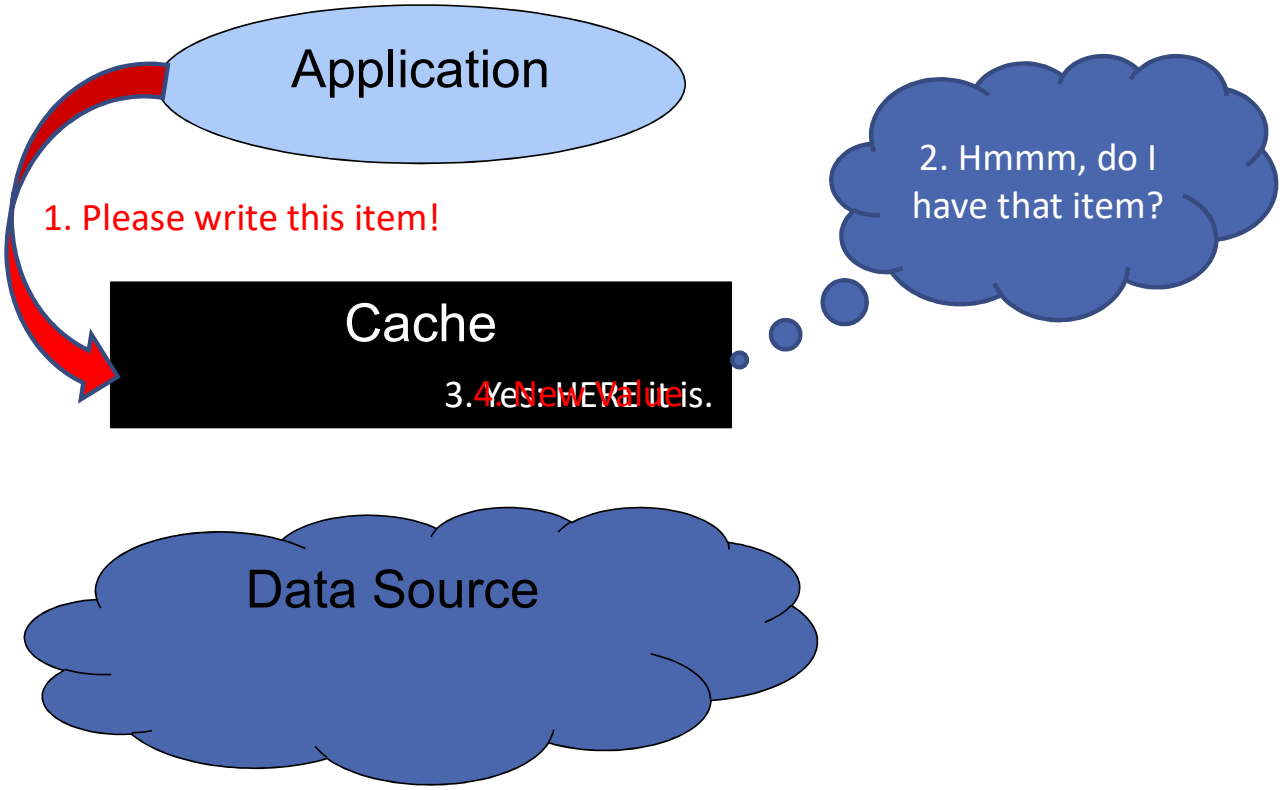
$$\text{Speedup} = \frac{1}{\left(1 - \alpha + \frac{\alpha}{k}\right)}$$

- Suppose that a program that takes 100 seconds to complete spends 33% of its time in memory instructions. If we add a cache that makes memory instructions four times faster, how much faster will the program run?
  - α = .33
  - k = 4
  - T_{old} = 100
  - Speedup = $\frac{1}{\left(1 - .33 + \frac{.33}{4}\right)} = \frac{1}{.7525}$ = 1.32
  - Latency = 100 * .7525 = 75.3 seconds

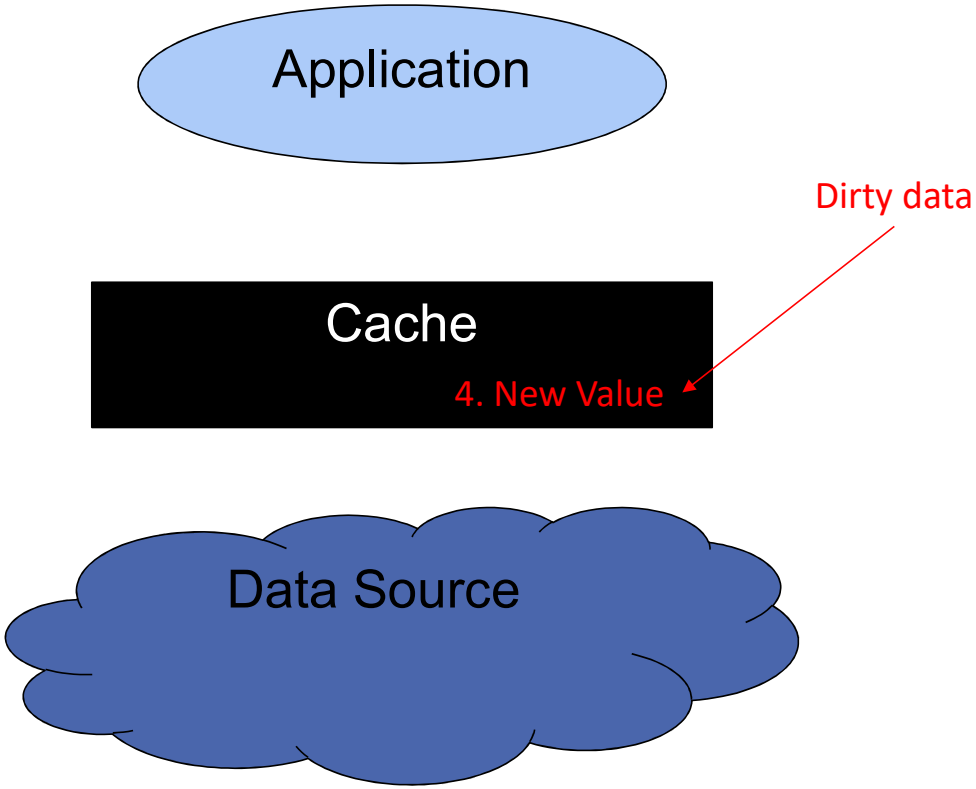
Part 2: What about writes?

- So far, we have focused exclusively on read caching.
- Does caching not apply to writes?
  - Of course it does! In fact, the first exercise you did in this unit on caching demonstrated effects of write caching!

A Write Cache Hit

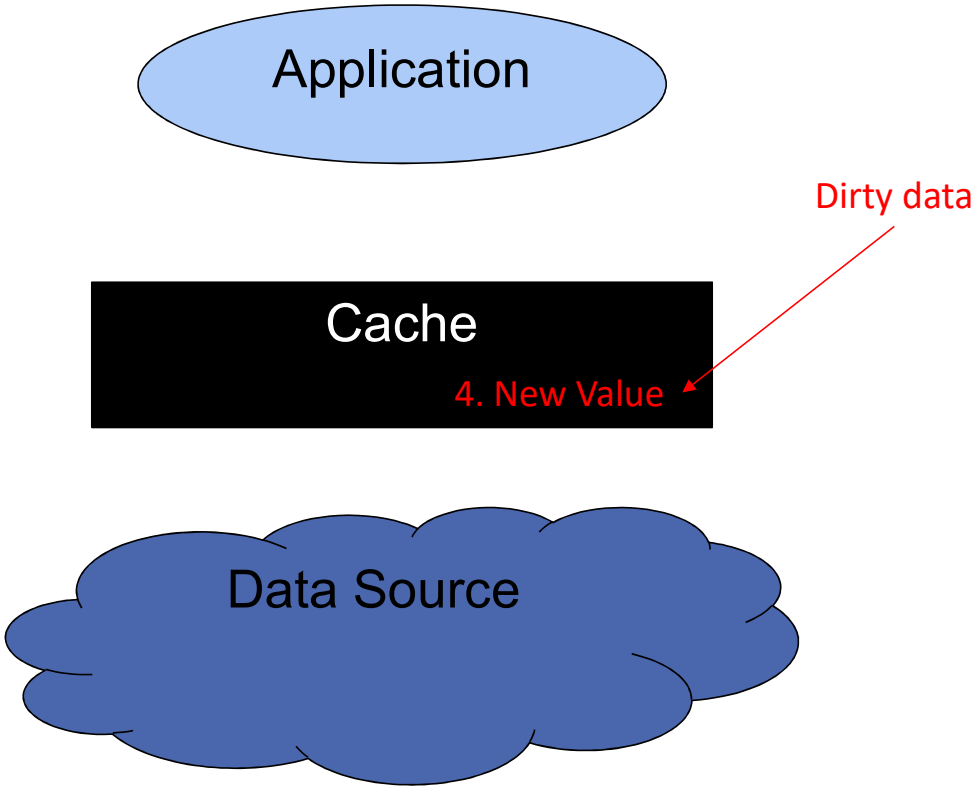


A Write Cache Hit: Policy Decisions

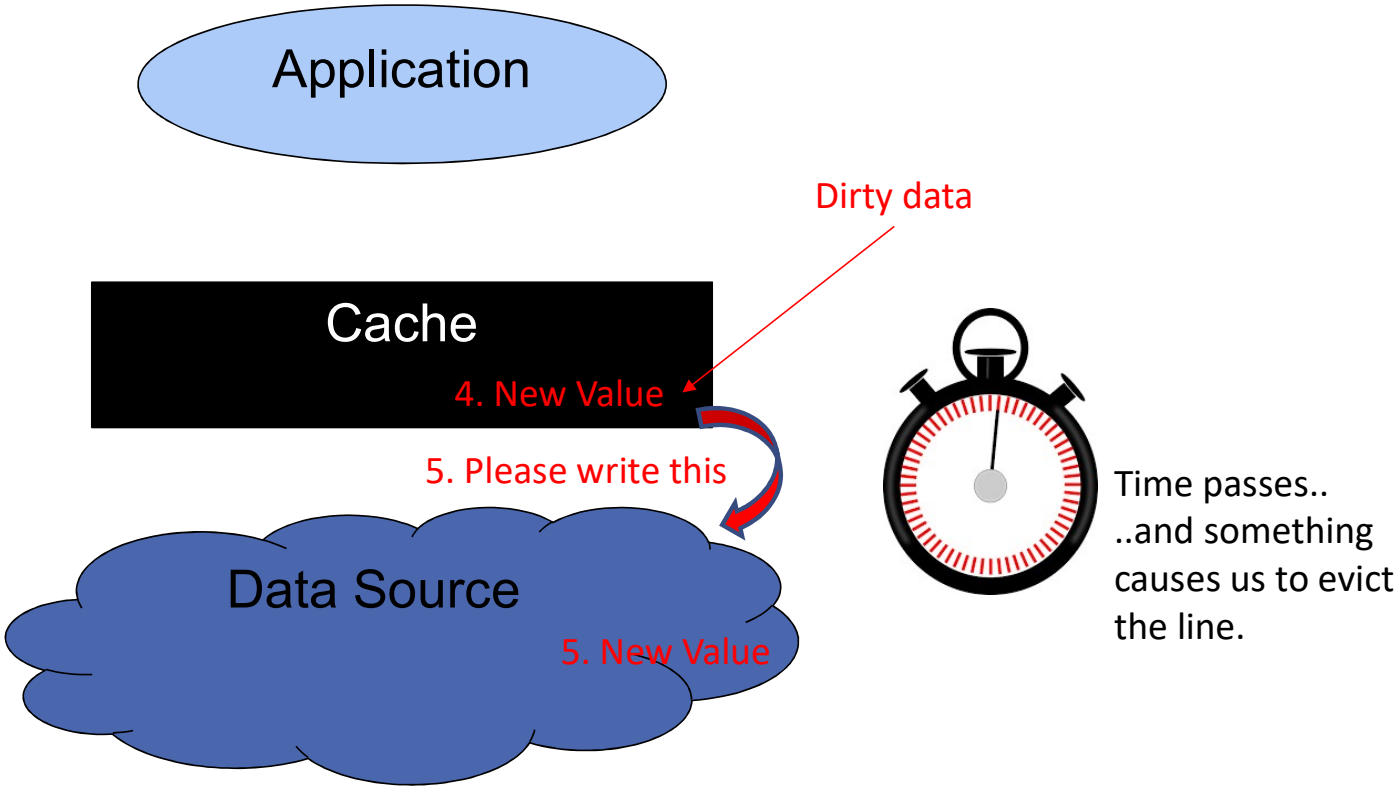


When do we write new data back to the source?

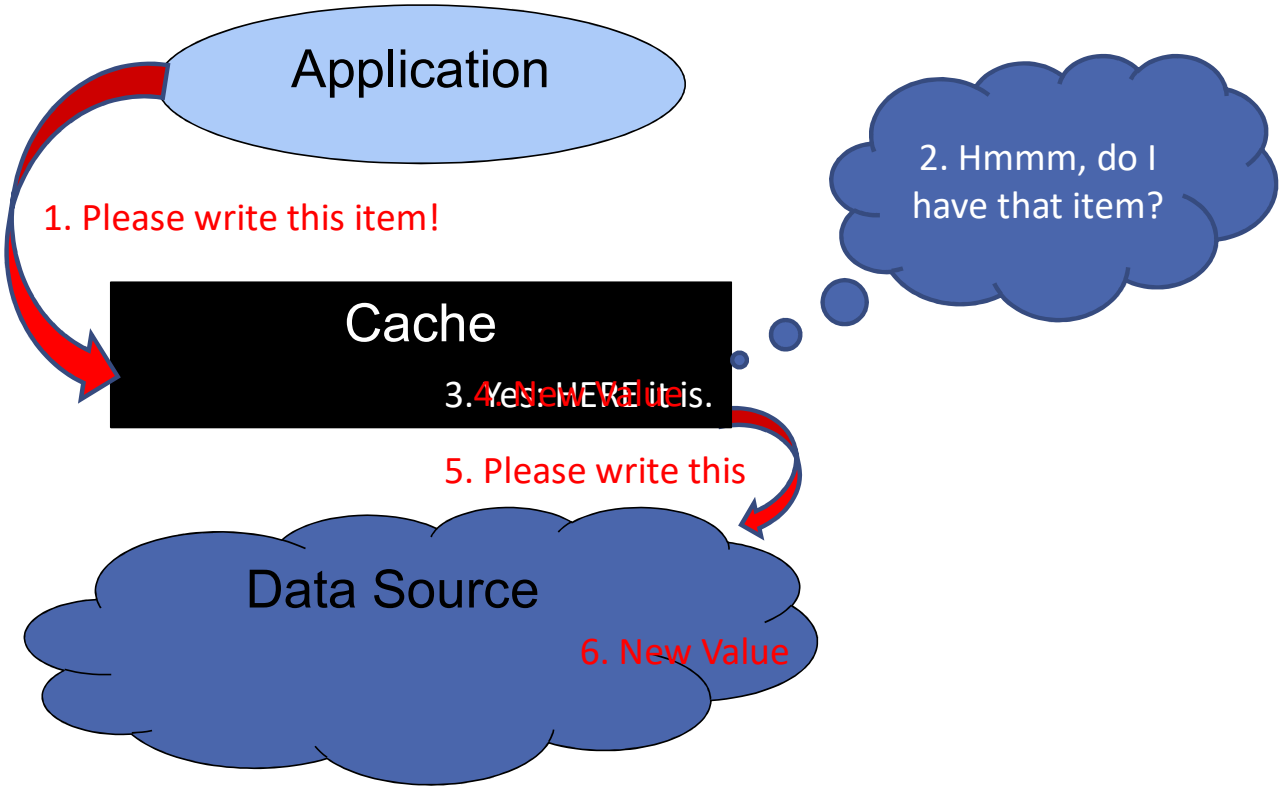
A Write Cache Hit: **Writeback** Caching



A Write Cache Hit: **Writeback** Caching



A Write Cache Hit: **Write Through**



Write Policy Trade-offs

- Advantages of writeback cache (= disadvantage of write through)
  -
  -
- Advantages of writethrough cache (= disadvantage of write back)
  -
  -

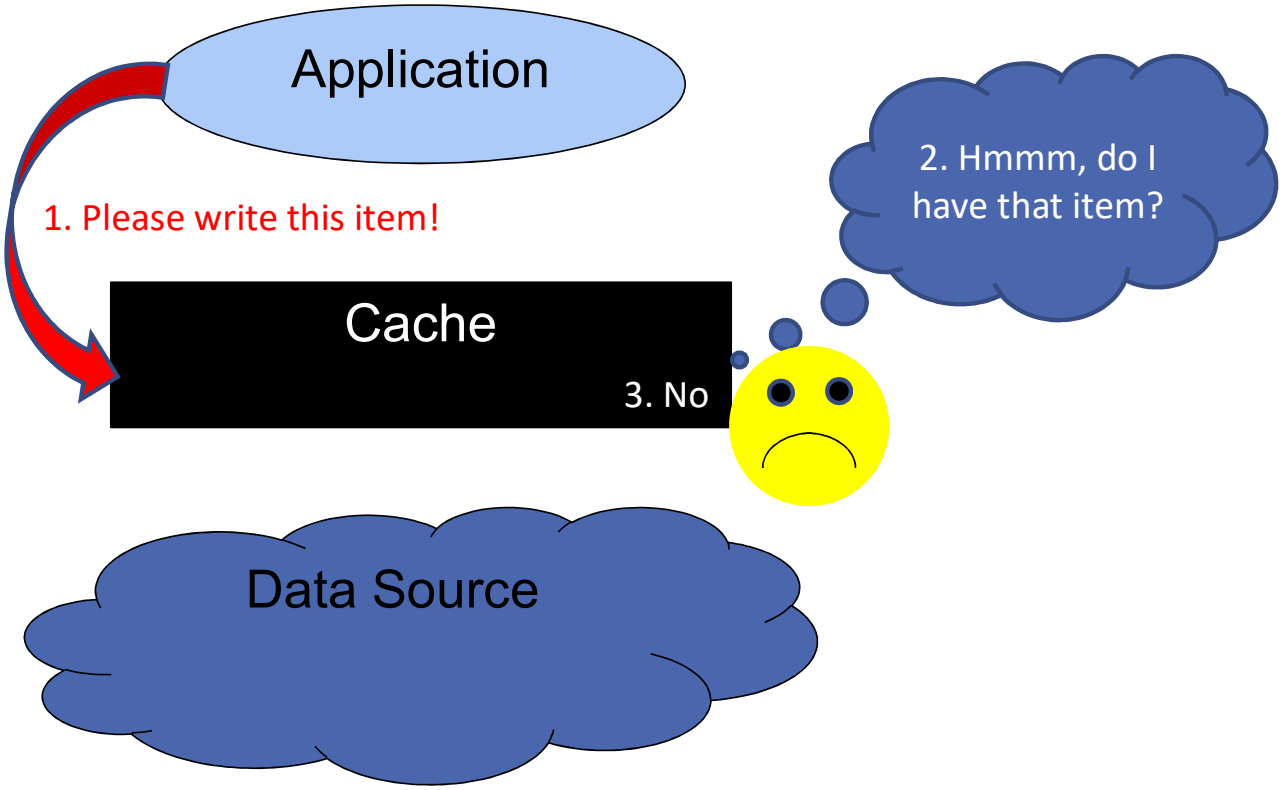
Write Policy Trade-offs

- Advantages of writeback cache (= disadvantage of write through)
  - Faster (cache time, not source time)
  - Write coalescing: update the same item (or multiple items in a cache line) many times before writing it back to the source.
- Advantages of writethrough cache (= disadvantage of write back)
  -
  -

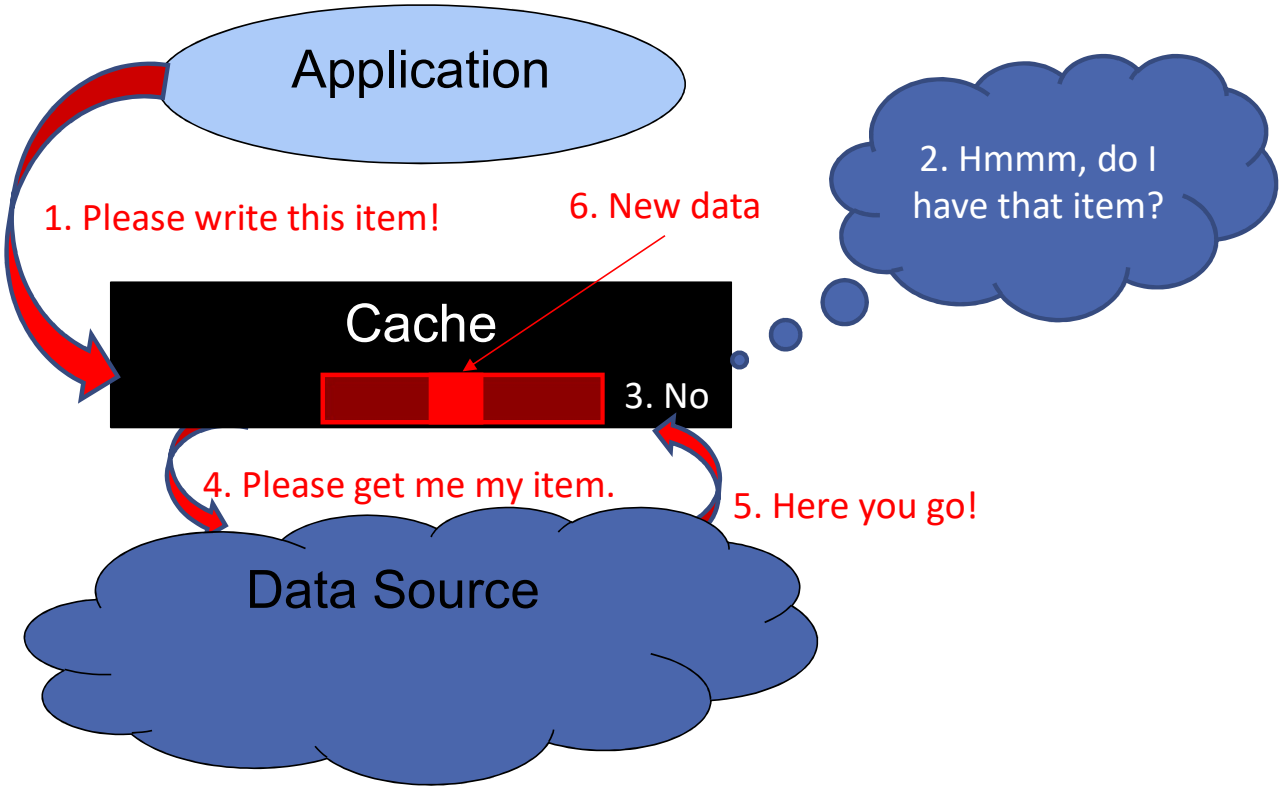
Write Policy Trade-offs

- Advantages of writeback cache (= disadvantage of write through)
  - Faster (cache time, not source time)
  - Write coalescing: update the same item (or multiple items in a cache line) many times before writing it back to the source.
- Advantages of writethrough cache (= disadvantage of write back)
  - No window of vulnerability (new data is in cache, but not source)
  - Typically simpler to implement

A Write Cache Miss



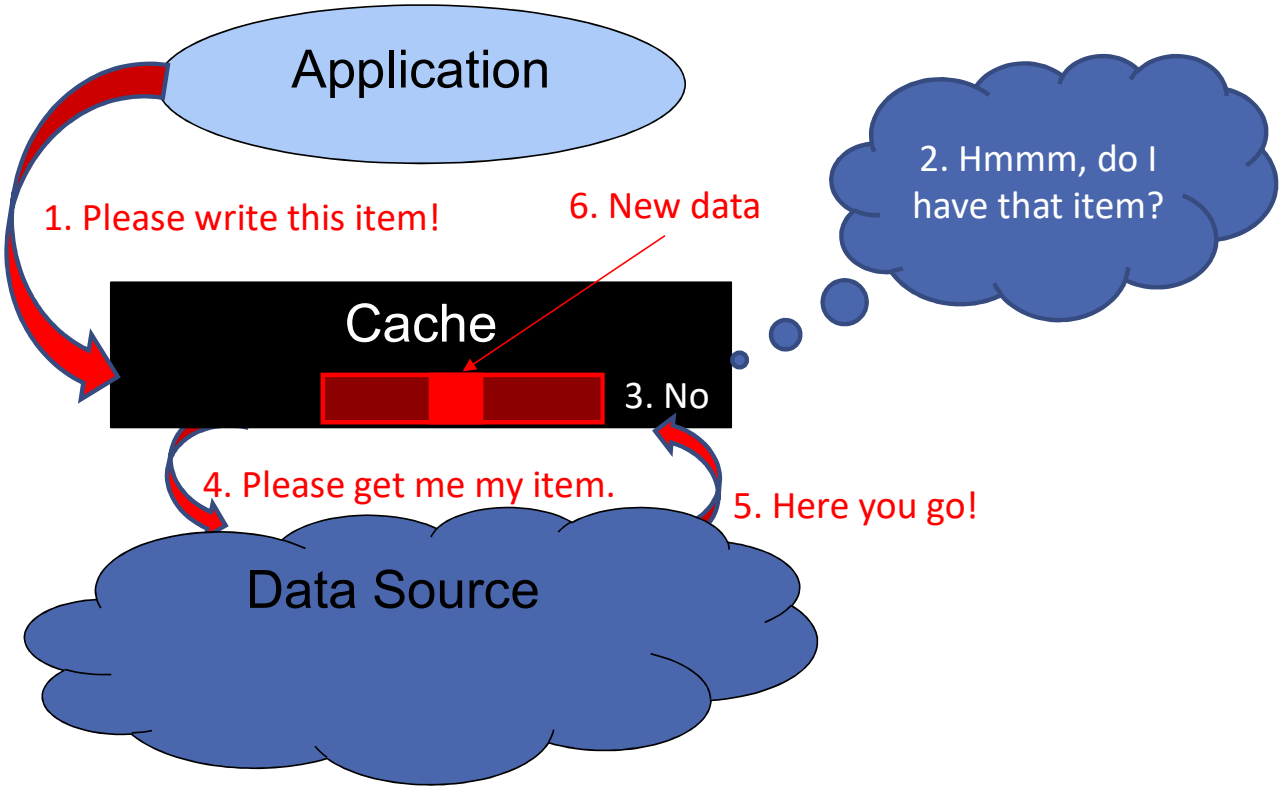
A Write Cache Miss – Write Allocate



Important: We're writing; don't we already know the new data?

- What's the bright red part of the rectangle above?
- What's the dim red part of the rectangle above?

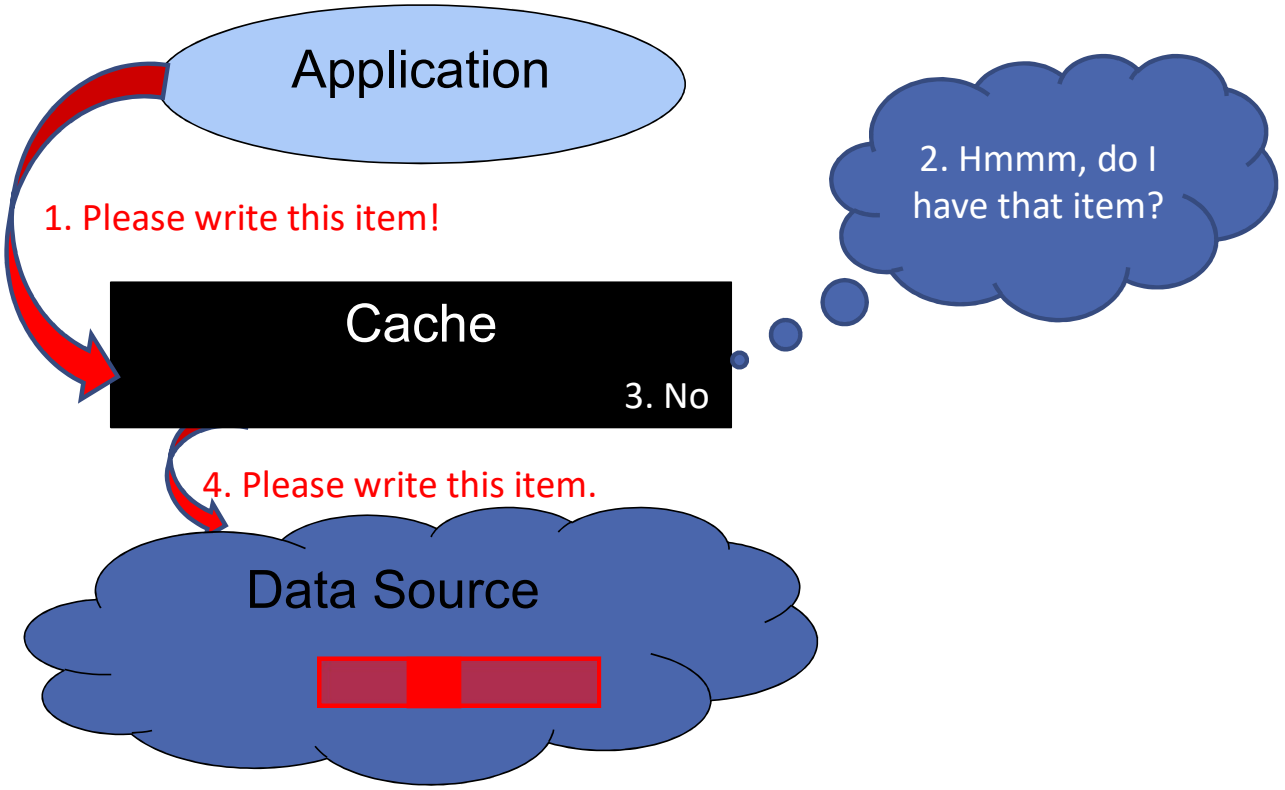
A Write Cache Miss – Write Allocate



Important: We're writing; don't we already know the new data?

- What's the bright red part of the rectangle above? The newly written *part* of the line.
- What's the dim red part of the rectangle above? The *rest* of the cache line!

A Write Cache Miss – No Write Allocate



Write Allocation Tradeoffs

- Advantages of write-allocate (= disadvantage of no write allocate)
  -
  -
- Advantages of no write-allocate (= disadvantage of write allocate)
  -
  -

Write Allocation Tradeoffs

- Advantages of write-allocate (= disadvantage of no write allocate)
  - Faster when you are modifying multiple values in a cache line.
  - Matches read behavior
- Advantages of no write-allocate (= disadvantage of write allocate)
  -
  -

Write Allocation Tradeoffs

- Advantages of write-allocate (= disadvantage of no write allocate)
  - Faster when you are modifying multiple values in a cache line.
  - Matches read behavior
- Advantages of no write-allocate (= disadvantage of write allocate)
  - Avoids cluttering the cache with write-only data
  - Avoids consuming a lot of cache space for random writes

Wrapping Up

Sidenote:
X86 has non-temporal move instructions that bypass the cache!

- If writes go all the way through to the data source, we call that a write-through cache.
- If writes can stay in the cache, we call the written cache blocks dirty (and the cache must be write-back).
- If write misses result in a cache line being allocated in the cache, we call that write-allocate.
- If we bypass the cache for write misses, we call that no write-allocate.
- Rule of thumb: write-back and write-allocate typically go together as do write-through and no write allocate.

Now it is your turn

Takeaways

- Amdahl's Law gives us a precise and principled way to evaluate and compare the performance of two systems (where a system can be a program running under specific conditions, a machine, etc).
- Writes introduce new set of policy decisions:
  - Write through versus writeback
  - Write allocate versus no write allocate