

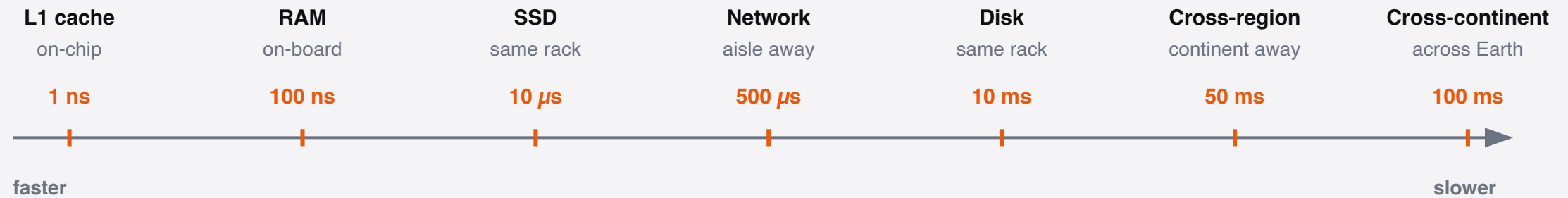
M2

Data Centers: The Backbone of Modern Databases

Latency in a modern data center spans eight orders of magnitude, from a nanosecond on-chip to a hundred milliseconds across the planet.

One data center: seven orders of magnitude

The same byte is 1 nanosecond or 100 milliseconds away, depending on its tier.



End to end is 100,000,000x. Read from the wrong tier and one access costs up to 10,000,000x more.

Seven tiers in one data center, fastest to slowest, from L1 cache at 1 ns to a cross-continent round trip at 100 ms. End to end is 100,000,000x.

PLAY IN CLASS

[Inside Microsoft's data centers \(video\)](#) ↗

M2

What Is a System?

A system is components chosen to work with each other.

Pick one from each bin

The goal: a car that will do 150 mph on a public road.

THE PARTS YOU CAN PICK

 **Engine**

E1	Motorbike	130 mph	\$5,000
E2	Turbo four	150 mph	\$12,000
E3	Jet	400 mph	\$2M

 **Chassis**

C1	Lego	2 mph	\$50
C2	Steel	150 mph	\$10,000
C3	Carbon	250 mph	\$5M

 **Wheels**

W1	Road	150 mph	\$600
W2	Sport	200 mph	\$2,000
W3	Slick	250 mph	\$50,000

WHAT YOU CAN BUILD FROM THEM

Mismatched

E3 + C1 + W3

2 mph

\$2.05M · C1 is the bottleneck

Oversized

E3 + C3 + W3

250 mph

\$7.05M · 100 mph past the goal

Right-sized

E2 + C2 + W1

150 mph

\$22,600 · hits the goal exactly

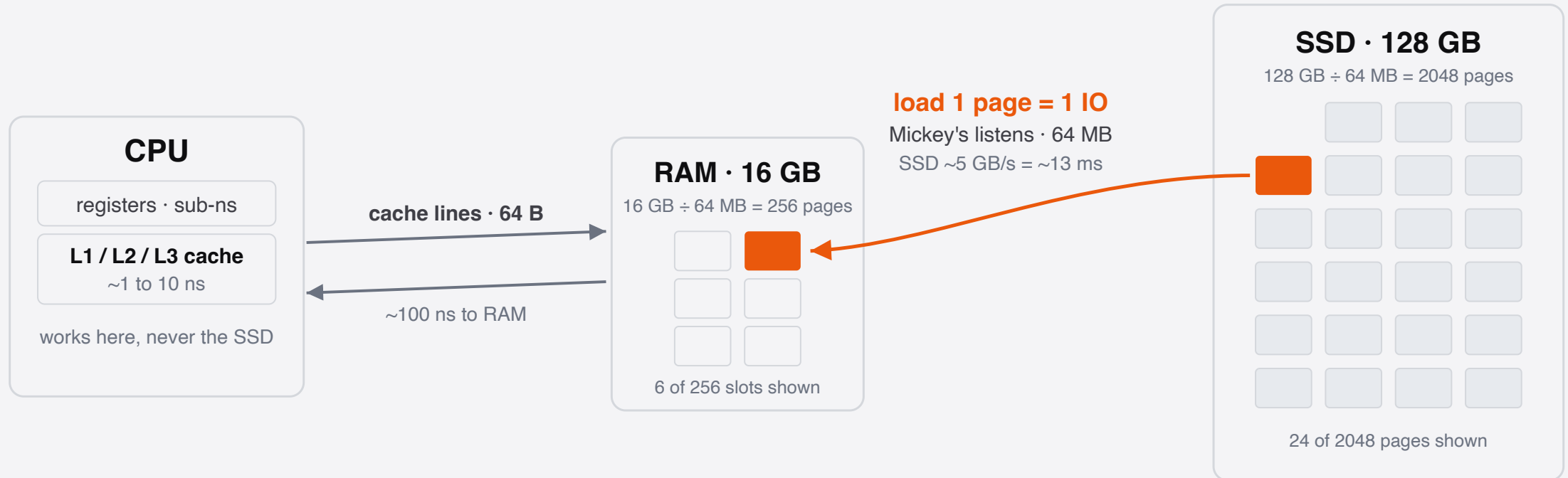
The goal is a car that will do 150 mph on a road. Pick one row from each bin; the car then runs at its slowest part and costs every part in it. E3+C1+W3 spends \$2.05M and travels at walking pace: C1 is the bottleneck. E3+C3+W3 has no bottleneck, but it is oversized: it runs 100 mph past the goal, at \$7.05M. E2+C2+W1 hits 150 on all three parts at once for \$22,600.

M2

IO & Storage

The CPU never touches the SSD

Pages move SSD → RAM, one page = one IO. The CPU works out of cache, filled from RAM.



When RAM fills, evict a page and load the next: **2048 SSD pages stream through 256 RAM slots, one at a time.**

The CPU computes only out of its registers and cache; it does not operate on the SSD directly. The SSD gives RAM a 64 MB page per IO, and RAM feeds the CPU 64-byte cache lines in nanoseconds. That page costs the same whether the query wants one row or the whole block, and at ~5 GB/s a 64 MB transfer takes ~13 ms, so cost is page count. ****Paging**** refers to the overall steps in moving the needed subset of pages in-n-out of RAM.

Why paging matters: the speed gap, in human time

If one L1 cache hit took 1 second, each tier would feel this long.



Rescale latency so an L1 cache hit takes one second. RAM becomes 1.5 minutes, an SSD read becomes 3 hours, and a hard disk read becomes 4 months. Paging keeps the working set in RAM. A query that reads from disk on every page pays the hard disk or SSD cost once per page.

OPTIONAL, RECOMMENDED

How does computer memory, cache, and storage work? (video) ↗

Small Data vs Big Data: Where the Cost Sits

- **Small data (< 10 GB): count CPU operations.** Everything fits in RAM, so we measure CPU costs.
- **Big data (> 100 GB): count IOs, pages read and written.** IO costs dominate; CPU costs are trivial by comparison.

IO COST

$$C_r \times \text{pages read} + C_w \times \text{pages written}$$

C_r time to read one page

C_w time to write one page

$P(R)$ number of pages in table R

$T(R) = n$ number of rows in R

Cost counts page I/Os, not CPU work. C_r and C_w are the time to read or write one page; $P(R)$ is the pages in table R and $T(R) = n$ its row count. Every algorithm's cost is $C_r \times \text{pages read} + C_w \times \text{pages written}$.

Three algorithms: count the pages

Example 1 · Basic read / write

Read and write every page once.

$$C_r \cdot P(R) + C_w \cdot P(R)$$

Example 2 · Multi-pass

Read every page 7 times, write 3 times.

$$C_r \cdot 7 \cdot P(R) + C_w \cdot 3 \cdot P(R)$$

Example 3 · Logarithmic

Read $\log P(R)$ passes; write $0.1 \cdot P(R)$ pages per row.

$$C_r \cdot \log P(R) \cdot P(R) + C_w \cdot 0.1 \cdot P(R) \cdot T(R)$$

Example 1 reads and writes every page once; Example 2 makes seven read passes and three write passes; Example 3 reads $\log P(R)$ passes and writes a tenth of the pages per row. The cost is the page count; the CPU work is negligible.

Calculating Read and Write Costs

The cost of reading one page, and of an algorithm:

$$C_r = \text{access time} + \frac{\text{page size}}{\text{scan speed}}$$

$$\text{IO cost} = C_r \times \text{pages read} + C_w \times \text{pages written}$$

Device (2025+)	Access time	Scan speed	C_r , one 64 MB page
RAM	100 ns	100 GB/s	0.00064 s
SSD	10 μ s	5 GB/s	0.01281 s
HDD	10 ms	100 MB/s	0.65 s
Network	1 μ s	10 GB/s	0.00640 s

Page size is 64 MB unless a problem says otherwise; units are decimal (1 MB = 1,000,000 bytes, 1 GB = 1,000,000,000 bytes). In practice a write can cost more than a read; unless a problem says otherwise, CS145 uses $C_w = C_r$.

The official CS145 IO equations

CS 145 Equations – v1.3 (Updated 9/1/25)

Many CS algorithms assume data fits in RAM, and focus on a CPUcost model (e.g., $O(n \log n)$ for Sorting). “Big” data does not fit in RAM.
Our Goal: Model IOCosts for working with data in (slow) SSD/HDDs and RAM (versus fast CPUs)

[1] Basic IO Cost model for RAM, SSD, HDD

Assume: 1 (disk) Block = 1 (RAM) Page.

1. $numPages = \lceil \frac{numMBs}{pageSize} \rceil$
2. ReadCost per page (C_r) = 1 IO.
WriteCost per page (C_w): k IOs
($1 \leq k \leq 10$ – Based on hardware! Default: $k = 1$)
3. Time per IO (secs): $accessLatency + \frac{pageSize}{scanSpeed}$

Example

E.g., pageSize = 64 MBs, numMBs = 630 MBs
SSD: accessTime = 10 us, scanSpeed = 5 GBps

- $numPages = \lceil \frac{630MB}{64MBs} \rceil = 10$
- $C_r = 1IO = 10us + \frac{64MB}{5GBps}$ secs
- Time to read 10 pages = $10 * C_r$
- Time to read 100 pages on 20 machines = $5 * C_r$

[2] HP and BigSort ‘R’ on column X

For table ‘R’, $T(R) = \#$ of rows in R. $P(R) = \#$ of pages in R. Buffer (B) = # of RAM pages.

1. **HashPartition(R)**: IOCost = $(C_r + C_w) * P(R)$
2. **BigSort(R)**: Let $N = P(R)$. IOCost = $(C_r + C_w) * N * (1 + \lceil \log_B \frac{N}{2B} \rceil)$

Special case for BigSort(R) ($C_r, C_w = 1$):

- (Small data) $N \leq 2B$? IOcost = $2N$
- If $2B < N \leq 2B^2$? $4N$; If $2B^2 < N \leq 2B^3$? $6N$

Example – $C_r, C_w = 1$

- $P(R) = 1000, B = 100$
- $HP(R) = 2 * 1000 = 2000$ IOs
- $BigSort(R) = 2 * 1000 * (1 + \lceil \log_{100} \frac{1000}{2 * 100} \rceil) = 4000$ IOs

[3] Basic JOIN(R, S) on column X

Let OUT be the number of output pages for a JOIN depends on number of matching tuples and row size of JOIN result).

1. **BNLJ(R, S)**:
 $C_r * (P(R) + P(S) * \lceil \frac{P(R)}{B} \rceil) + C_w * OUT$
(Pick BNLJ(S, R), if cheaper – see Box[6])

Example – $C_r, C_w = 1$

- $P(R) = 1000, P(S) = 2000, B = 100$
- $BNLJ(R, S) = 1000 + 2000 * \lceil \frac{1000}{100} \rceil + OUT$
- $BNLJ(S, R) = 2000 + 1000 * \lceil \frac{2000}{100} \rceil + OUT$

[4] HPJ(R, S) on column X

1. **HPJ(R, S)**: $HP(R) + HP(S) + PJ(R, S) + C_w * OUT$
2. $HP(R) = (C_r + C_w) * P(R)$ // Ditto for $HP(S)$
3. $PJ(R, S) =$

- (“Ideal”-case) $C_r * (P(R) + P(S))$
- Skew? Try different hash.
- Many duplicates? Try different algo!

4. $HPJ(R, R)$. I.e., a self-join
 $HPJ(R, R) = HP(R) + C_r * P(R) + C_w * OUT$

Special case – $C_r, C_w = 1$

- $HPJ(R, S) = 3(P(R) + P(S)) + OUT$

Example – $C_r, C_w = 1$

- $P(R) = 1000, P(S) = 2000, B = 100$
- $HPJ(R, S) = 3 * (1000 + 2000) + OUT$
- $HPJ(R, R) = 3 * 1000 + OUT$

[5] SMJ(R, S) with JOIN on column X

1. **SMJ(R, S)**: $BigSort(R) + BigSort(S) + MergeJoin(R, S) + C_w * OUT$
2. MergeJoin(R, S) with JOIN on column X:
 - (a) (Best-case: No Backup) $C_r * (P(R) + P(S))$
 - (b) (Worst-case: All Backup) $C_r * P(R) * P(S)$
3. SMJ(R, R). I.e., a self-join
 - (a) (Best-case: No Backup) $BigSort(R) + C_r * P(R) + C_w * OUT$
 - (b) (Worst-case: All Backup) $BigSort(R) + C_r * P(R) * P(R) + C_w * OUT$
4. If R is pre-sorted, $BigSort(R) = 0$. (Same as S.)

Example – $C_r = C_w = 1$

- $P(R) = 1000, P(S) = 2000, B = 100$
- $SMJ(R, S) = 4000 + 8000 + 3000 + OUT$ IOs
 - $BigSort(R) = 2 * 1000 * (1 + \lceil \log_{100} \frac{1000}{2 * 100} \rceil) = 4000$
 - $BigSort(S) = 2 * 2000 * (1 + \lceil \log_{100} \frac{2000}{2 * 100} \rceil) = 8000$
 - MergeJoin(R, S) = 3000 IOs, no backup
 - MergeJoin(R, S) = 1000 * 2000 IOs, full backup

[6] Indexing: Hash Index and B+ Trees

- n : Number of data rows, N : Number of data pages
- d : Data row size, k : Key size, p : Pointer size
- M : Page size, h : index height

Data Pages: $N = \lceil \frac{n * d}{M} \rceil$

Index Fanout: $f = \lceil \frac{M}{k + p} \rceil$

Hash Index (Unsorted): $P(index) = \lceil \frac{n}{f} \rceil$

Hash Index (Sorted): $P(index) = \lceil \frac{N}{f} \rceil$

B+ Tree Height (Unsorted): $h \approx \lceil \log_f(n) \rceil$

B+ Tree Height (Sorted): $h \approx \lceil \log_f(N) \rceil$

Query IOs: (index IO + data IO) $\approx h + 1$

Example Values:

- $M = 64MB, k = 8 bytes, p = 8 bytes, n = 10^{12}, d = 1024 bytes$
- $f = \lceil \frac{64 * 1024 * 1024}{8 + 8} \rceil = 4,194,304$
- $N = \lceil \frac{10^{12} * 1024}{64 * 1024 * 1024} \rceil = 15258790$
- B+ Tree height (unsorted): $\lceil \log_{4194304}(10^9) \rceil = 2$
- B+ Tree height (sorted): $\lceil \log_{4194304}(15258790) \rceil = 2$

M2

Problems: IO Cost Model

Problem 1: Table Size Calculation

The Spotify Songs table has 500 million rows with an average row size of 1024 bytes. Calculate:

1. Total size in MB
2. Number of 64 MB pages needed

▼ Show Solution

Total size

500,000,000 rows × 1024 bytes = **512,000 MB = 512 GB**

Number of pages

512,000 MB ÷ 64 MB = **8,000 pages**

[1] Basic IO Cost model for RAM, SSD, HDD

Assume: 1 (disk) Block = 1 (RAM) Page.

1. $numPages = \lceil \frac{numMBs}{pageSize} \rceil$
2. ReadCost per page (C_r) = 1 IO.
WriteCost per page (C_w): k IOs
($1 \leq k \leq 10$ – Based on hardware! Default: $k = 1$)
3. Time per IO (secs): $accessLatency + \frac{pageSize}{scanSpeed}$

Example

E.g., $pageSize = 64$ MBs, $numMBs = 630$ MBs

SSD: $accessTime = 10$ us, $scanSpeed = 5$ GBps

- $numPages = \lceil \frac{630MB}{64MBs} \rceil = 10$
- $C_r = 1IO = 10us + \frac{64MB}{5GBps}$ secs
- Time to read 10 pages = $10 * C_r$
- Time to read 100 pages on 20 machines = $5 * C_r$

Problem 2: Read Cost Comparison

Calculate the time (in seconds) to read 100 pages from different storage devices.

▼ Show Solution

RAM

$$100 \times (0.0000001 \text{ s} + 0.00064 \text{ s}) = \mathbf{0.064 \text{ s}}$$

SSD

$$100 \times (0.00001 \text{ s} + 0.0128 \text{ s}) = \mathbf{1.281 \text{ s}}$$

HDD

$$100 \times (0.01 \text{ s} + 0.64 \text{ s}) = \mathbf{65 \text{ s}}$$
 (about 1,000× slower than RAM)

[1] Basic IO Cost model for RAM, SSD, HDD

Assume: 1 (disk) Block = 1 (RAM) Page.

1. $numPages = \lceil \frac{numMBs}{pageSize} \rceil$
2. ReadCost per page (C_r) = 1 IO.
WriteCost per page (C_w): k IOs
($1 \leq k \leq 10$ – Based on hardware! Default: $k = 1$)
3. Time per IO (secs): $accessLatency + \frac{pageSize}{scanSpeed}$

Example

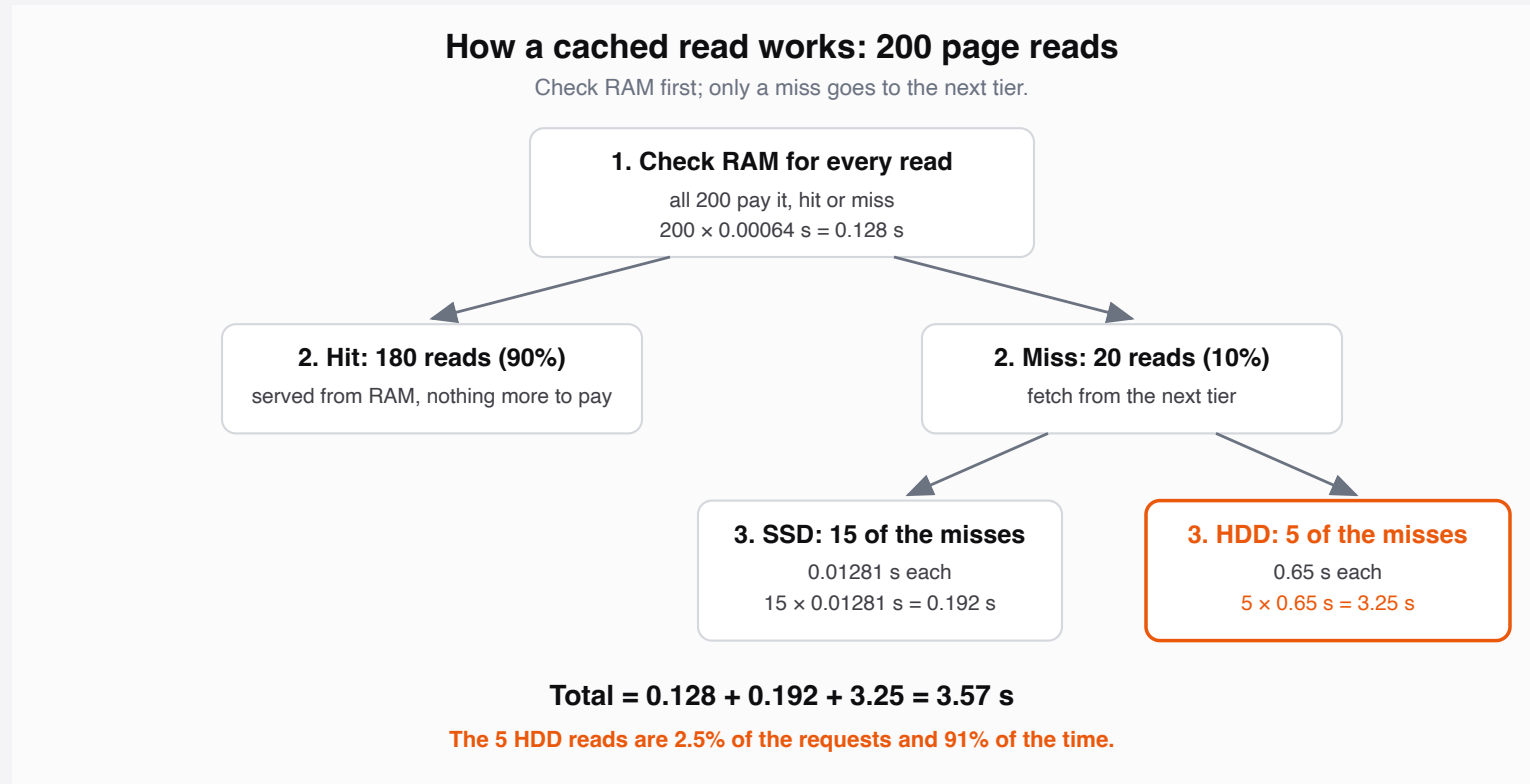
E.g., $pageSize = 64 \text{ MBs}$, $numMBs = 630 \text{ MBs}$
SSD: $accessTime = 10 \text{ us}$, $scanSpeed = 5 \text{ GBps}$

- $numPages = \lceil \frac{630MB}{64MBs} \rceil = 10$
- $C_r = 1IO = 10us + \frac{64MB}{5GBps} \text{ secs}$
- Time to read 10 pages = $10 * C_r$
- Time to read 100 pages on 20 machines = $5 * C_r$

Problem 3: Cache Hit Rate Impact

Read 200 pages through a cache. RAM holds 90% of them. Of the pages RAM misses, 75% are on SSD and 25% on HDD. What is the total read time?

▼ Show Solution



Total time: 3.57 s. The RAM check is paid on every read, and the few reads that fall to HDD dominate.

Problem 4: Mixed Read/Write Operations

You need to:

- Read 30 pages
- Write 10 pages

Calculate total cost for each storage device.

▼ Show Solution

Cost formula

$$\text{Total} = 30 \times C_R + 10 \times C_W$$

Reads and writes cost the same on these devices, so this is $40 \times C_R$.

RAM

$$40 \times 0.00064 \text{ s} = \mathbf{0.026 \text{ s}}$$

SSD

$$40 \times 0.01281 \text{ s} = \mathbf{0.512 \text{ s}}$$

HDD

$$40 \times 0.65 \text{ s} = \mathbf{26 \text{ s}}$$

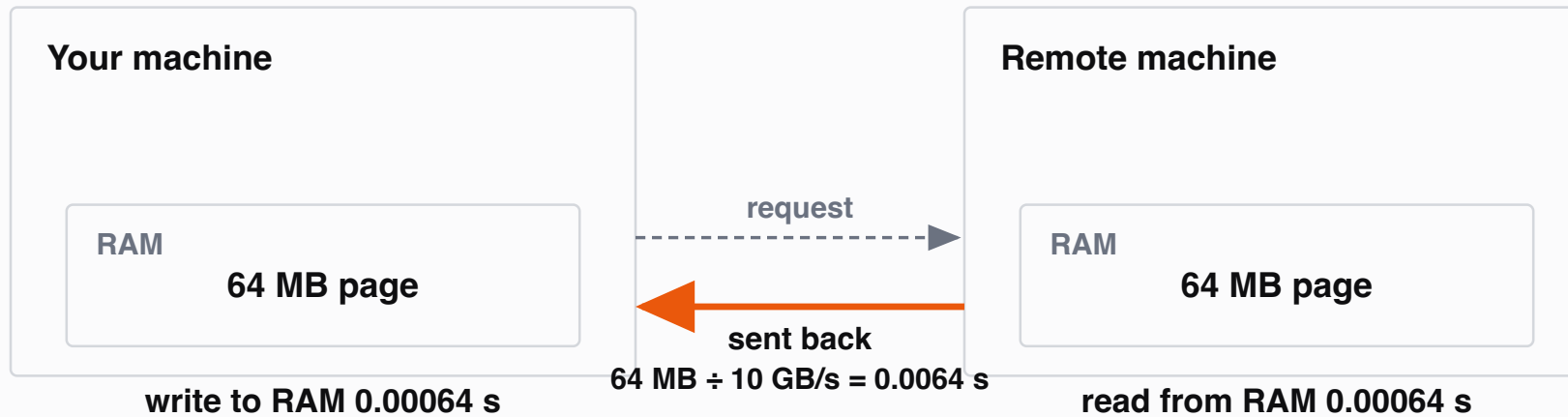
Problem 5: Network vs Local Storage

Compare reading 1 page from:

1. Local RAM
2. Local SSD
3. Local HDD
4. Network RAM (RAM on another machine)

Network RAM: the page lives on another machine

Request it, read it from remote RAM, send it back, write it into yours.



$$0.00064 + 0.0064 + 0.00064 = 0.0077 \text{ s}$$

Faster than the same page off a local SSD (0.0128 s).

Key Takeaways

1. **HDDs and SSDs are much slower than RAM** for random access
2. **Cache hit rates matter enormously** - even 5% HDD access can dominate
3. **Network RAM $\approx$ Local SSD** in performance
4. **Page size affects random access penalty** - larger pages amortize seek cost
5. **Moving forward:** We'll count IOs (1 page = 1 IO) to analyze algorithm complexity