Hard disks, SSDs, and the I/O subsystem

Tyler Bletsch
Duke University

Slides include material from Vince Freeh (NCSU)
Hard Disk Drives (HDD)
History

- First: IBM 350 (1956)
  - 50 platters (100 surfaces)
  - 100 tracks per surface (10,000 tracks)
  - 500 characters per track
  - 5 million characters
  - 24” disks, 20” high
Overview

- Record data by magnetizing ferromagnetic material
- Read data by detecting magnetization
- Typical design
  - 1 or more platters on a spindle
  - Platter of non-magnetic material (glass or aluminum), coated with ferromagnetic material
  - Platters rotate past read/write heads
  - Heads ‘float’ on a cushion of air
  - Landing zones for parking heads
Basic schematic

a. side view.

b. top view.
Generic hard drive

- Spindle
- Head
- Actuator Arm
- Actuator Axis
- Power Connector
- Jumper Block
- Data Connector

^ (these aren’t common any more)
Types and connectivity (legacy)

- **SCSI (Small Computer System Interface):**
  - Pronounced “Scuzzy”
  - One of the earliest small drive protocols
  - Many revisions to standard – many types of connectors!
  - The Standard That Will Not Die: the drives are gone, but most enterprise gear still speaks the SCSI protocol

- **Fibre Channel (FC):**
  - Used in some Fibre Channel SANs
  - Speaks SCSI on the wire
  - Modern Fibre Channel SANs can use any drives: back-end ≠ front-end

- **IDE / ATA:**
  - Older standard for consumer drives
  - Obsoleted by SATA in 2003
Types and connectivity (modern)

• SATA (Serial ATA):
  • Current consumer standard
  • Series of backward-compatible revisions
    SATA 1 = 1.5 Gbit/s, SATA 2 = 3 Gbit/s,
    SATA 3 = 6.0 Gbit/s, SATA 3.2 = 16 Gbit/s
  • Data and power connectors are hot-swap ready
  • Extensions for external drives/enclosures (eSATA),
    small all-flash boards (mSATA, M.2),
    multi-connection cables (SFF-8484), more
  • Usually in 2.5” and 3.5” form factors

• SAS (Serial-Attached-SCSI)
  • SCSI protocol over SATA-style wires
  • (Almost) same connector
  • Can use SATA drives on SAS controller, not vice versa
Inside hard drive
Read/write head
Head close-up
Arm
Video of hard disk in operation

https://www.youtube.com/watch?v=sG2sGd5XxM4

From: http://www.metacafe.com/watch/1971051/hard_disk_operation/
Hard drive capacity

Seeking

• Steps
  • Speedup
  • Coast
  • Slowdown
  • Settle

• Very short seeks (2-4 tracks): dominated by settle time

• Short seeks (<200-400 tracks):
  • Almost all time in constant acceleration phase
  • Time proportional to square root of distance

• Long seeks:
  • Most time in constant speed (coast)
  • Time proportional to distance
Average seek time

• What is the “average” seek? If
  1. Seeks are fully independent and
  2. All tracks are populated:
     ➔ average seek = 1/3 full stroke

• But seeks are not independent
• Short seeks are common

• Using an average seek time for all seeks yields a poor model
Track following

- Fine tuning the head position
  - At end of seek
  - Switching between last sector one track to first on another
  - Switching between head (irregularities in platters) [*]

- Time for full settle
  - 2-4ms; 0.24-0.48 revolutions
  - \((7200\text{RPM} \rightarrow 0.12 \text{ revolutions/ms})\)

- Time for *
  - \(1/3-1/2\) settle time
  - 0.5-1.5 ms (0.06-0.18 revolutions @ 7200RPM)
Optimistic head settling on read

• Start reading when close
• Let error correcting come into play if not aligned
• Not feasible for writes
Zoning

• Note
  • More linear distance at edges than at center
  • Bits/track $\sim R$ (circumference = $2\pi R$)
  • To maximize density, bits/inch should be the same

• How many bits per track?
  • Same number for all $\Rightarrow$ simplicity; lowest capacity
  • Different number for each $\Rightarrow$ very complex; greatest capacity

• Zoning
  • Group tracks into zones, with same number of bits
  • Outer zones have more bits than inner zones
  • Compromise between simplicity and capacity
Example

IBM deskstar
40GV (ca. 2000)

<table>
<thead>
<tr>
<th>Zone</th>
<th>Tracks in Zone</th>
<th>Sectors Per Track</th>
<th>Data Transfer Rate (Mbits/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>624</td>
<td>792</td>
<td>372.0</td>
</tr>
<tr>
<td>1</td>
<td>1,424</td>
<td>780</td>
<td>366.4</td>
</tr>
<tr>
<td>2</td>
<td>1,680</td>
<td>760</td>
<td>357.0</td>
</tr>
<tr>
<td>3</td>
<td>1,616</td>
<td>740</td>
<td>347.6</td>
</tr>
<tr>
<td>4</td>
<td>2,752</td>
<td>720</td>
<td>338.2</td>
</tr>
<tr>
<td>5</td>
<td>2,880</td>
<td>680</td>
<td>319.4</td>
</tr>
<tr>
<td>6</td>
<td>1,904</td>
<td>660</td>
<td>310.0</td>
</tr>
<tr>
<td>7</td>
<td>2,384</td>
<td>630</td>
<td>295.9</td>
</tr>
<tr>
<td>8</td>
<td>3,328</td>
<td>600</td>
<td>281.8</td>
</tr>
<tr>
<td>9</td>
<td>4,432</td>
<td>540</td>
<td>253.6</td>
</tr>
<tr>
<td>10</td>
<td>4,528</td>
<td>480</td>
<td>225.5</td>
</tr>
<tr>
<td>11</td>
<td>2,192</td>
<td>440</td>
<td>206.7</td>
</tr>
<tr>
<td>12</td>
<td>1,600</td>
<td>420</td>
<td>197.3</td>
</tr>
<tr>
<td>13</td>
<td>1,168</td>
<td>400</td>
<td>187.9</td>
</tr>
<tr>
<td>14</td>
<td>18,15</td>
<td>370</td>
<td>173.8</td>
</tr>
</tbody>
</table>
Track skewing

Why:
- Imagine that sectors are numbered identically on each track, and we want to read all of two adjacent tracks (common!)
- When we finish the last sector of the first track, we seek to the next track.
- In that time, the platter has moved 0.24-0.48 revolutions
- We have to wait almost a full rotation to start reading sector 1! Bad!

What:
- Offset first sector a small amount on each track
- (Also offset it between platters due to head switch time)

Effect:
- Able to read data across tracks at full speed

Sparing

- Reserve some sectors in case of defects
- Two mechanisms
  - Mapping
  - Slipping
- Mapping
  - Table that maps requested sector \(\rightarrow\) actual sector
- Slipping
  - Skip over bad sector
- Combinations
  - Skip-track sparing at disk “low level” (factory) format
  - Remapping for defects found during operation
Caching and buffering

- Disks have caches
  - Caching (e.g., optimistic read-ahead)
  - Buffering (e.g., accommodate speed differences bus/disk)
- Buffering
  - Accept write from bus into buffer
  - Seek to sector
  - Write buffer
- Read-ahead caching
  - On demand read, fetch requested data and more
  - Upside: subsequent read may hit in cache
  - Downside: may delay next request; complex
Command queuing

- Send multiple commands (SCSI)
- Disk schedules commands
- Should be “better” because disk “knows” more

Questions

- How often are there multiple requests?
- How does OS maintain priorities with command queuing?
Time line

Read
- Host sends command
- Controller decodes it
- Controller disconnects from bus & starts seek
- SCSI bus data transfers to host
- Status message to host
- Disk mechanism
- Seek
- Rotation latency
- Data transfer off mechanism
- Head switch

Write
- Host sends command
- Controller decodes it
- SCSI bus data transfer from host
- Status message to host
- Disk mechanism
- Seek
- Rotation latency
- Data transfer to mechanism
- Head switch
## Disk Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Diameter</td>
<td>3.5”</td>
<td>2.5”</td>
<td>1.8”</td>
</tr>
<tr>
<td>Capacity</td>
<td>6 TB</td>
<td>73 GB</td>
<td>10 GB</td>
</tr>
<tr>
<td>RPM</td>
<td>7200 RPM</td>
<td>10000 RPM</td>
<td>4200 RPM</td>
</tr>
<tr>
<td>Cache</td>
<td>128 MB</td>
<td>8 MB</td>
<td>512 KB</td>
</tr>
<tr>
<td>Platters</td>
<td>~6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Average Seek</td>
<td>4.16 ms</td>
<td>4.5 ms</td>
<td>7 ms</td>
</tr>
<tr>
<td>Sustained Data Rate</td>
<td>216 MB/s</td>
<td>94 MB/s</td>
<td>16 MB/s</td>
</tr>
<tr>
<td>Interface</td>
<td>SAS/SATA</td>
<td>SCSI</td>
<td>ATA</td>
</tr>
<tr>
<td>Use</td>
<td>Desktop</td>
<td>Laptop</td>
<td>Ancient iPod</td>
</tr>
</tbody>
</table>
Disk Read/Write Latency

- Disk read/write latency has four components
  - **Seek delay** \(t_{\text{seek}}\): head seeks to right track
  - **Rotational delay** \(t_{\text{rotation}}\): right sector rotates under head
    - On average: time to go halfway around disk
  - **Transfer time** \(t_{\text{transfer}}\): data actually being transferred
  - **Controller delay** \(t_{\text{controller}}\): controller overhead (on either side)

- Example: time to read a 4KB page assuming...
  - 128 sectors/track, 512 B/sector, 6000 RPM, 10 ms \(t_{\text{seek}}\), 1 ms \(t_{\text{controller}}\)

\[
\begin{align*}
6000 \text{ RPM} & \rightarrow 100 \text{ R/s} \rightarrow 10 \text{ ms/R} \rightarrow t_{\text{rotation}} = 10 \text{ ms} / 2 = 5 \text{ ms} \\
4 \text{ KB page} & \rightarrow 8 \text{ sectors} \rightarrow t_{\text{transfer}} = 10 \text{ ms} \times 8/128 = 0.6 \text{ ms} \\
\sum t_{\text{disk}} & = t_{\text{seek}} + t_{\text{rotation}} + t_{\text{transfer}} + t_{\text{controller}} \\
& = 10 + 5 + 0.6 + 1 = 16.6 \text{ ms}
\end{align*}
\]
Solid State Disks (SSD)
Introduction

- Solid state drive (SSD)
  - Storage drives with no mechanical component
  - Available up to 4TB capacity (as of 2017)
  - Usually 2.5” form factor

Source: wikipedia
Evolution of SSDs

- PROM – programmed once, non erasable
- EPROM – erased by UV lighting*, then reprogrammed
- EEPROM – electrically erase entire chip, then reprogram
- Flash – electrically erase and rerecord a single memory cell
- SSD - flash with a block interface emulating controller

* Obsolete, but totally awesome looking because they had a little window:
Flash memory primer

- Types: NAND and NOR
  - NOR allows bit level access
  - NAND allows block level access
    - For SSD, NAND is mostly used, NOR going out of favor
- Flash memory is an array of columns and rows
  - Each intersection contains a memory cell
    - Memory cell = floating gate + control gate
    - 1 cell = 1 bit
## Memory cells of NAND flash

<table>
<thead>
<tr>
<th>Single-level cell (SLC)</th>
<th>Multi-level cell (MLC)</th>
<th>Triple-level cell (TLC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single (bit) level cell</td>
<td>Two (bit) level cell</td>
<td>Three (bit) level cell</td>
</tr>
<tr>
<td>Fast: 25us read/100-300us write</td>
<td>Reasonably fast: 50us read, 600-900us write</td>
<td>Decently fast: 75us read, 900-1350us write</td>
</tr>
<tr>
<td>Write endurance - 100,000 cycles</td>
<td>Write endurance – 10000 cycles</td>
<td>Write endurance – 5000 cycles</td>
</tr>
<tr>
<td>Expensive</td>
<td>Less expensive</td>
<td>Least expensive</td>
</tr>
</tbody>
</table>
SSD internals

- Package contains multiple dies (chips)
- Die segmented into multiple planes
- A plane with thousands (2048) of blocks + IO buffer pages
- A block is around 64 or 128 pages
- A page has a 2KB or 4KB data + ECC/additional information
SSD internals

• Logical pages striped over multiple packages
  • A flash memory package provides 40MB/s
  • SSDs use array of flash memory packages

• Interfacing:
  • Flash memory → Serial IO → SSD Controller → disk interface (SATA)

• SSD Controller implements Flash Translation Layer (FTL)
  • Emulates a hard disk
  • Exposes logical blocks to the upper level components
  • Performs additional functionality
SSD controller

- Differences in SSD is due to controller
  - Performance loss if controller not properly implemented
- Has CPU, RAM cache, and may have battery/supercapacitor
- Dynamic logical block mapping
  - LBA to PBA
    - Page level mapping (uses large RAM space ~512MB)
    - Block level mapping (expensive read/write/modify)
  - Most use hybrid
    - Block level with log sized page level mapping
Wear levelling

• SSDs wear out
  • Each memory cell has finite flips
  • All storage systems have finite flips even HDD
  • SSD finite flips < HDD
  • HDD failure modes are larger than SSD

• General method: over-provision unused blocks
  • Write on the unused block
  • Invalidate previous page
  • Remap new page
Dynamic wear leveling

- Only pool unused blocks
- Only non-static portion is wear leveled
- Controller implementation easy
- Example: SSD lifespan dependent on 25% of SSD

Source: micron
Static wear leveling

- Pool all blocks
- All blocks are wear leveled
- Controller complicated
  - needs to track cycle # of all blocks
- Static data moved to blocks with higher cycle #
- Example: SSD lifespan dependent on 100% of SSD

Source: micron
Preemptive erasure

- Preemptive movement of cold data
- Recycle invalidated pages
  - Performed by garbage collector
  - Background operation
  - Triggered when close to having no more unused blocks
SSD operations

- **Read**
  - Page level granularity
  - 25us (SLC) to 60us (MLC)

- **Write**
  - Page level granularity
  - 250us (SLC) to 900us (MLC)
  - 10 x slower than read

- **Erase**
  - Block level granularity, not page or word level
  - Erase must be done before writes
  - 3.5ms
  - 15 x slower than write
SSD TRIM! Sent from the OS

- **TRIM**
  - Command to notify SSD controller about deleted blocks
  - Sent by filesystem when a file is deleted
  - Avoids write amplification and improves SSD life
Using SSD (1)

- **Hybrid storage (tiering)**
  - Server flash
    - Client cache to backend shared storage
    - Accelerates applications
    - Boosts efficiency of backend storage (backend demand decreases by upto 50%)
  - Example: NetApp Flash Accel acts as cache to storage controller
    - Maintains data coherency between the cache and backend storage
    - Supports data persistent for reboots
Using SSD (2)

- Hybrid storage
  - Flash array as cache (PCI-e cards flash arrays)
    - Example: NetApp Flash Cache in storage controller
    - Cache for reads
  - SSDs as cache
    - Example: NetApp Flash Pool in storage controller
    - Hot data tiered between SSDs and HDD backend storage
    - Cache for read and write

- SSD as main storage device
  - NetApp “All Flash” storage controllers
  - 300,000 read IOPS
  - < 1 ms response time
  - > 6Gbps bandwidth
  - Cost: $big
  - Becoming increasingly common as SSD costs fall
NetApp flash cache

- Combined with HDD
- Upto 16 TB read cache
NetApp EF540 flash array

• 2U
• Target: transactional apps with high IOPS and low latency
• Equivalent to > 1000 15K RPM HDDs
• 95% reduction in space, power, and cooling
• Capacity: up to 38TB

Source: NetApp
## Differences between SSD and HDD

<table>
<thead>
<tr>
<th>SSD</th>
<th>HDD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Uniform seek time</strong></td>
<td>Different seek time for different sectors</td>
</tr>
<tr>
<td><strong>Fast seek time – random read/writes as fast as sequential read/writes</strong></td>
<td>Seek time dependent upon the RPM</td>
</tr>
<tr>
<td><strong>Cost (Intel 530 Series 240GB – $209)</strong></td>
<td><strong>Cost (Seagate Constellation 1TB 7200rpm - $116)</strong></td>
</tr>
<tr>
<td>• Capacity – $0.87/GB</td>
<td>• Capacity – $0.11/GB</td>
</tr>
<tr>
<td>• Rate – $0.005/IOPS</td>
<td>• Rate – $0.55/IOPS</td>
</tr>
<tr>
<td>• Bandwidth - $0.38/Mbps</td>
<td>• Bandwidth - $0.99/Mbps</td>
</tr>
</tbody>
</table>

**Power:**
- **Active power:** 195mW – 2W
- **Idle power:** 125mW – 0.5 W
- Low power consumption, No sleep mode

**Power:**
- Average operating power: 5.4W
- Higher power consumption, sleep mode zero power, higher wake up cost
# Differences between SSD and HDD

<table>
<thead>
<tr>
<th>SSD</th>
<th>HDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 10,000 to &gt; 1million IOPS</td>
<td>Hundreds of IOPS</td>
</tr>
<tr>
<td>Read/write in microseconds</td>
<td>Read/write in milliseconds</td>
</tr>
<tr>
<td>No mechanical part – no wear and tear</td>
<td>Moving part – wear and tear</td>
</tr>
<tr>
<td>MTBF ~ 2 million hours</td>
<td>MTBF ~ 1.2 million hours</td>
</tr>
<tr>
<td>Faster wear of a memory cell when it is written multiple times</td>
<td>Slower wear of the magnetic bit recording</td>
</tr>
</tbody>
</table>
Intel X-25E - $345 (older)
SLC
32 GB
SATA II
170-250MB/s
Latency 75-85us

Intel 530 - $209 (new)
MLC
240GB
SATA III
up to 540MB/s
Latency 80-85us

Samsung 840 EVO - $499 (new)
TLC
1TB
SATA III
up to 540MB/s
Which is cheaper?

HDD? Yes!
Cheaper per gigabyte of capacity.

or

SSD? Yes!
Cheaper per IOPS (performance).

Tradeoff!
### Workloads

<table>
<thead>
<tr>
<th>Workloads</th>
<th>SSD</th>
<th>HDD</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>High write</td>
<td>Y</td>
<td></td>
<td>Wear for SSD</td>
</tr>
<tr>
<td>Sequential write</td>
<td>Y</td>
<td>Y</td>
<td>SSD: Seek time low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HDD: Lower seek time</td>
</tr>
<tr>
<td>Log files (small writes)</td>
<td>Y</td>
<td></td>
<td>Faster seek time</td>
</tr>
<tr>
<td>Database read queries</td>
<td>Y</td>
<td></td>
<td>Faster seek time</td>
</tr>
<tr>
<td>Database write queries</td>
<td>Y</td>
<td></td>
<td>Faster seek time</td>
</tr>
<tr>
<td>Analytics – HDFS</td>
<td>Y</td>
<td>Y</td>
<td>SSD – Append operation faster</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HDD – higher capacity</td>
</tr>
<tr>
<td>Operating systems</td>
<td>Y</td>
<td>Y</td>
<td>SSD: Less changing files</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HDD: More read only data</td>
</tr>
</tbody>
</table>
Other Flash technologies - NVDIMMS

- Revisiting NVRAM
- DDR3 DIMMS + NAND Flash
  - Speed of DIMMS
  - extensive read/write cycles for DIMMS
  - Non volatile nature of NAND Flash
- Support added by BIOS
  - Backup to NAND Flash
  - Triggered by HW SAVE signal
- Stored charge
  - Super capacitors
  - Battery packs

How It Works

If there is a power failure, the supercap module powers NVDIMM while it copies all data from the DDR-3 to on-module flash

When power is restored NVDIMM copies all data from flash to DDR-3 and normal operation resumes

(SNIA - NVDIMM Technical Brief)
In future - persistent memory

- NVM latency closer to DRAM
- Types
  - Battery-backed DRAM, NVM with caching, Next-gen NVM
- Attributes:
  - Bytes-addressable, LOAD/STORE access, memory-like, DMA
  - Data not persistent until flushed

Source: Andy Rudoff, Intel
The I/O Subsystem
I/O Systems
I/O Interface

Independent I/O Bus

CPU

Memory

Interface

Peripheral

Interface

Peripheral

Lines distinguish between I/O and memory transfers

Separate I/O instructions (in, out)
Memory Mapped I/O

Single Memory & I/O Bus
No Separate I/O Instructions

CPU

Memory

Interface

Peripheral

Interface

Peripheral

CPU

L2

Memory Bus

I/O bus

Memory

Bus Adaptor

ROM

RAM

I/O
Programmed I/O (Polling)

- CPU
- Memory
- IO Controller
- device

Is the data ready?
- yes: read data
- no: busy wait loop

store data

done?
- yes
- no: not an efficient way to use the CPU unless the device is very fast!

but checks for I/O completion can be dispersed among computationally intensive code
Interrupt Driven Data Transfer

User program progress only halted during actual transfer
Direct Memory Access (DMA)

- Interrupts remove overhead of polling...
- But still requires OS to transfer data one word at a time
  - OK for low bandwidth I/O devices: mice, microphones, etc.
  - Bad for high bandwidth I/O devices: disks, monitors, etc.

- **Direct Memory Access (DMA)**
  - Transfer data between I/O and memory without processor control
  - Transfers entire blocks (e.g., pages, video frames) at a time
    - Can use bus “burst” transfer mode if available
  - Only interrupts processor when done (or if error occurs)
DMA Controllers

• To do DMA, I/O device attached to **DMA controller**
  • Multiple devices can be connected to one DMA controller
  • Controller itself seen as a memory mapped I/O device
    • Processor initializes start memory address, transfer size, etc.
  • DMA controller takes care of bus arbitration and transfer details
    • So that’s why buses support arbitration and multiple masters!
I/O Processors

• A DMA controller is a very simple component
  • May be as simple as a FSM with some local memory
• Some I/O requires complicated sequences of transfers
  • **I/O processor**: heavier DMA controller that executes instructions
    • Can be programmed to do complex transfers
    • E.g., programmable network card

```
<table>
<thead>
<tr>
<th>CPU ($)</th>
<th>DMA</th>
<th>DMA</th>
<th>IOP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Main Memory</td>
<td>Disk</td>
<td>display</td>
</tr>
</tbody>
</table>
```

Bus
Top questions to ask about any I/O system:

- **Storage device(s):**
  - What kind of device (SSD, HDD, etc.)?
  - Performance characteristics?

- **Topology:**
  - What’s connected to what (buses, IO controller(s), fan-out, etc.)?
  - What protocols in use (SAS, SATA, etc.)?
  - Where are the bottlenecks (PCI-E bus? SATA protocol limit? IO controller bandwidth limit?)
  - Protocol interaction: polled, interrupt, DMA?
Basics of IO Performance Measurement
Motivation and basic terminology

- We cover performance measurement in detail later in the semester, but you may need the basics for your project sooner than that...

- The short version:
  - Sequential workload: **MB/s**
    - Even an SSD does better sequential than random because of caching and other locality optimizations
  - Random workload: **IO/s** (commonly written IOPS)
    - You need to indicate the IO size, but it’s not part of the metric
  - Don’t forget: **latency (ms)**
Measurement methodology

• Basic test: do X amount of IO and divide by time T.
  • Both X and T may be specified or measured
  • Example:
    • Measure time to do 100,000 IOs (X given, T free)
    • Write to disk at max rate for 60 seconds, look at file size (T given, X free)

• Problem: **measurement variance**
Combating measurement variance (1)

- Measurement varying too much? Make sure your tests are long enough!
  - Otherwise you’re testing tiny random effects instead of the actual phenomenon under study...
• Measurement variance never goes away
  • Need to characterize it when presenting results, or you won’t be trusted!
  • How? Take multiple repetitions show average and standard deviation (or other variance metric)

• **ALL data requires variance to be characterized!**
  (not just in this course, but in your *life*)
  • For your projects, failure to characterize variance is likely an automatic request for resubmission!!

• How to present:
  • In tables, show variance next to average (e.g. “251.2 ± 11.6”)
  • In graphs, show variance with error bars, e.g.: