Example System Cases (Final Project Ideas)

> *These are example "system cases" to show students what a good final project looks like. Present 2-3 of these in Week 1 to help students form their own ideas.*

Case 1: PostgreSQL p99 Latency Spike Under Memory Pressure

Problem Statement

A PostgreSQL instance shows stable p50 latency (~2ms) but p99 spikes to 50ms+ under memory pressure. The DBA sees "nothing unusual" in PostgreSQL logs.

Investigation Approach

1. Reproduce with controlled memory pressure (cgroup memory.max or stress-ng)
2. Use perf to identify hotspots during spike
3. Use eBPF to trace page faults and reclaim events
4. Correlate memory.stat (pgmajfault, pgscan) with latency histogram

Expected Findings

- p99 spikes correlate with major page faults
- When dirty pages exceed threshold, synchronous reclaim kicks in
- Tuning vm.dirty_ratio or increasing memory budget reduces spikes

Skills Demonstrated

- cgroup memory control
- perf profiling
- eBPF tracing
- Causal explanation with data

Case 2: Redis Latency with Noisy Neighbor

Problem Statement

Redis container shows increased latency when a CPU-intensive workload runs on the same host, even though Redis has "dedicated" CPU cores via cpuset.

Investigation Approach

1. Set up two containers: Redis (cpuset=0-1) and CPU-burner (cpuset=2-3)
2. Measure Redis latency baseline
3. Start CPU-burner and remeasure
4. Use perf to examine cache behavior (LLC misses)
5. Use eBPF to trace scheduler decisions

Expected Findings

- Latency increase despite cpuset isolation
- LLC (last-level cache) contention is the culprit
- CPU-burner evicts Redis's hot data from shared cache

Skills Demonstrated

- Container CPU isolation (cpuset)
- Hardware performance counters (cache)
- Understanding of shared resources beyond CPU time

Case 3: Container OOM But Host Has Memory

Problem Statement

A containerized application gets OOM-killed, but `free -h` on the host shows plenty of available memory.

Investigation Approach

1. Reproduce by setting memory.max and running memory-hungry workload
2. Examine cgroup memory.events (oom, oom_kill)
3. Trace memory.stat over time
4. Understand difference between container limit and host memory

Expected Findings

- OOM is enforced by cgroup, not host-level OOM killer
- Application's memory footprint exceeded container's memory.max
- Page cache within container contributed to memory pressure

Skills Demonstrated

- cgroup v2 memory controller
- Understanding of OOM mechanisms (cgroup vs kernel)
- Memory accounting in containers

Case 4: llama.cpp Inference Timeout Under Concurrency

Problem Statement

A llama.cpp HTTP server handles single requests fine (~500ms), but under 10 concurrent requests, many time out (>5s).

Investigation Approach

1. Profile single vs concurrent requests with perf
2. Measure p50/p95/p99 latency distribution
3. Identify bottleneck: CPU saturation? lock contention? memory bandwidth?
4. Use cgroup to limit concurrent threads and observe effect

Expected Findings

- Memory bandwidth saturation (large model doesn't fit in cache)
- Thread contention in tokenization or sampling
- Batching improves throughput but increases individual latency

Skills Demonstrated

- Profiling ML inference workload
- Concurrency analysis
- Resource contention identification

Case 5: Sidecar Proxy Doubles p99 Latency

Problem Statement

After adding a sidecar proxy (e.g., Envoy) for observability, service p99 latency increases from 5ms to 12ms.

Investigation Approach

1. Measure latency with and without sidecar
2. Use eBPF to trace time spent in kernel (TCP stack) vs userspace (proxy)
3. Identify where the extra 7ms goes
4. Experiment with different proxy configurations

Expected Findings

- Extra TCP connection (service → proxy → upstream) adds latency
- Proxy's event loop introduces scheduling delays under load
- eBPF-based alternatives (e.g., Cilium) reduce overhead

Skills Demonstrated

- Network latency decomposition
- eBPF for network tracing
- Understanding of proxy architecture overhead

Case 6: fsync-Heavy Workload Tail Latency

Problem Statement

A logging service writes to disk with fsync after each batch. p50 is 1ms, but p99 is 50ms.

Investigation Approach

1. Profile with blktrace or eBPF block layer tracing
2. Identify when fsync takes long (device queue depth, journal commit)
3. Experiment with batching (fsync every N writes vs every write)
4. Measure impact of filesystem (ext4 vs xfs) and mount options

Expected Findings

- p99 correlates with journal commit or device flush
- Batching reduces fsync frequency, improves p99
- SSD vs HDD makes significant difference

Skills Demonstrated

- Storage I/O tracing
- Filesystem semantics (journaling, fsync)
- Tail latency analysis

Presentation Tips for Instructors

When presenting these cases in Week 1:

1. **Show the "before" state** — confused user, mysterious symptom

2. **Walk through investigation** — what tools, what hypotheses
3. **Show the "aha" moment** — the data that revealed the cause
4. **Emphasize reproducibility** — "you could do this experiment"

Don't make it look too polished — students should see that investigation involves false starts and iteration.