# Course Projects

These projects are designed to *cover the whole course* (measurement → concurrency/scheduling → observability → containers/cgroups → K8s resource management → network/storage tail latency → security → LLM workloads → agent runtime), while still being feasible on a laptop + Ubuntu VM.

## Format (common to all projects)

- **Team size:** 1–2 students
- **Duration:** Entire Semester
- **Effort expectation:** ~1–2 hours/week (average). Some weeks will be heavier (setup, milestone weeks).
- **Compute environment:** laptop + Ubuntu 22.04/24.04 VM (VirtualBox). No kernel hacking required.
- **Deliverables:**
  - **Technical report (60%)**
  - **Oral report / demo (40%)**

### Recommended milestones

- **Week 4 Select the project**
- **Week 6 Mechanism checkpoint**: Submit 1 page project plan and progress report
- **Week 8 Midterm Progress Check**: 1 page progress report
- **Week 10 Final freeze:** finalized experiments + draft report
- **Week 12 Defense:** presentation + reproduction checklist

### MS baseline (completion bar): evidence contract (VM-friendly)

This is a graduate OS course. **MS baseline is the minimum completion bar** across all projects.

For each major experiment (interference/resource-bound) and each mitigation, your writeup must include:

1. **Two independent evidence sources (cross-layer observations)**
   - "signal" here means **evidence / observable metrics**.
   - "independent" means **from different layers / subsystems**, e.g.:
     - app/user metrics (p50/p95/p99, TTFT, tokens/s, success rate)
     - plus OS/resource-control evidence (cgroup v2 stats, PSI, `/proc`, `pidstat` / `vmstat` / `iostat`, K8s events)
2. **One exclusion check (controlled-variable counterexample)**
   - Provide at least one piece of evidence to rule out a plausible alternative cause.
   - Examples: `iostat` stable → not IO; `cpu.stat` shows no throttling → not CPU quota.
3. **Before/after percentiles + one mechanism metric moved**
   - Report p50/p95/p99 (or at least p50/p99)
   - plus one mechanism-level metric that changes consistently with your explanation (e.g., `throttled_usec`, PSI, major faults, OOM/eviction events).

**VM note:** In VirtualBox/VMware Ubuntu VMs, many perf PMU hardware events are unavailable. Prefer VM-friendly signals (cgroup/PSI/ `/proc` /sysstat tools). Do not depend on hardware cache events.

### What "good" looks like (grading philosophy)

A strong project is not a feature build. It is a **system case study** with:

1. a clear metric (latency/throughput/error rate),
2. a reproducible workload,
3. a hypothesis-driven debugging loop,
4. evidence from OS observability (even if perf PMU is limited in VMs), and
5. a mechanism-level explanation.

## Project menu

### Recommended (v3): 4 integrative capstone projects (handouts)

- **Project 01 — Red/Blue Oncall Game: Incident-in-a-Box × Observability Stack**
- **Project 02 — Multi-hop Service on Mini-K8s: Cgroups/QoS and Tail Latency**
- **Project 03 — Systematic Profiling of an LLM Inference Server (CPU-first)**
- **Project 04 — Complete Profile of SWE-agent (Agent Runtime as a System)**

---

## Report & oral presentation requirements (applies to all projects)

### Technical report (60%)

A complete report should include:

- Abstract + problem statement
- System model + assumptions
- Experimental setup (hardware/VM config) and reproducibility checklist
- Metrics and methodology
- Results (plots/tables) + interpretation
- Mechanism explanation (OS-level)
- Intervention/mitigation and validation
- Related work (brief)
- Limitations

### Oral report / demo (40%)

- 15 minute talk + 5 minute Q&A (adjustable)
- Must include:
  - the symptom/metric,
  - the evidence trail,
  - the mechanism explanation,
  - and at least one live or recorded reproduction.