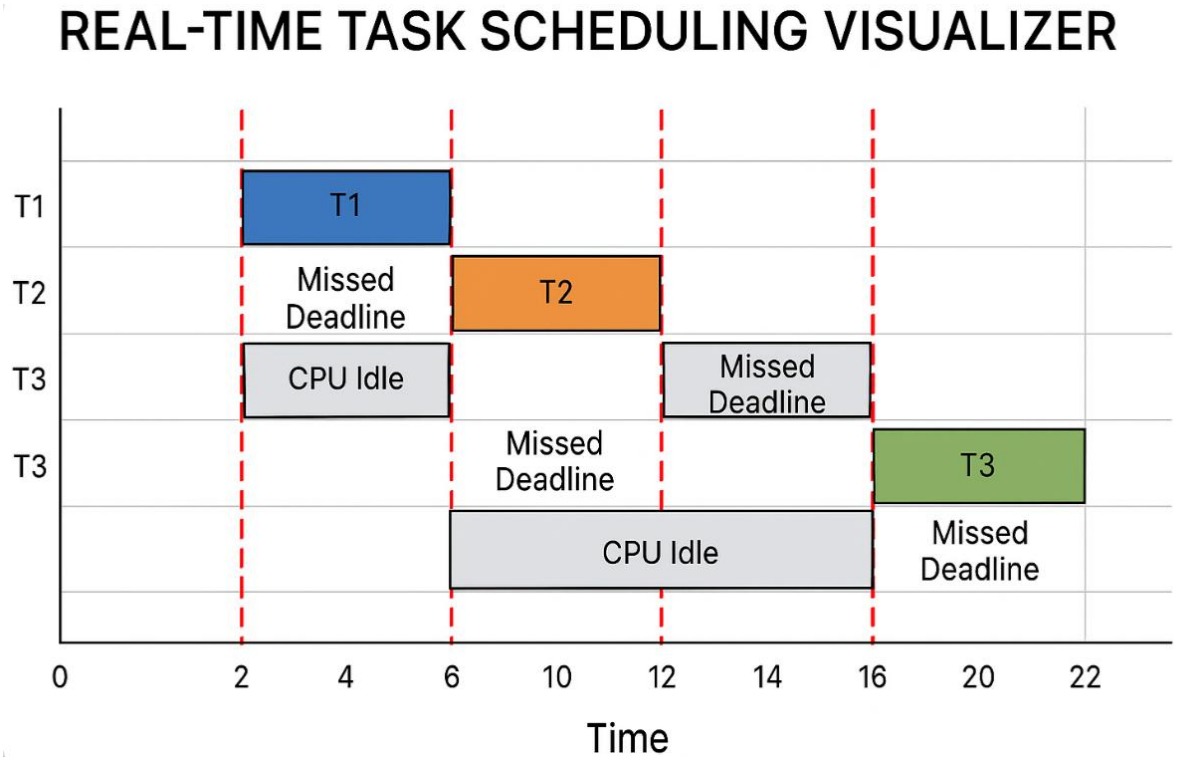
# Project Proposal: Real-Time Task Timing Visualizer (Simulator)

By- Dhruv Belawat and Saarthak Gupta

**1. Goal**

Build a simulator that models real-time task scheduling on a CPU. The simulator will run task traces (periodic, aperiodic, interrupts) and dynamically visualize scheduling decisions, deadlines, and missed deadlines. It will support multiple real-time scheduling algorithms, allow dynamic task arrivals, and provide interactive visualization.




---

**2. Features**

- **Dynamic Task Arrivals**
    - Tasks can arrive at runtime (random arrivals, scripted from JSON/CSV, or user-added via GUI).
- **Scheduling Algorithms**
    - Real-time: Rate Monotonic (RM), Earliest Deadline First (EDF), Deadline Monotonic (DM), Least Laxity First (LLF).
    - OS-style: FCFS, Shortest Job First (SJF/SRTF), Round Robin (RR), Fixed Priority.
- **Visualization**
    - Gantt-chart style timeline of task execution.
    - Deadline markers, with missed deadlines highlighted.

    - CPU idle periods shown distinctly.
- **Metrics**
    - CPU utilization, response times, missed deadlines count.
    - Export results to CSV.

---

**3. Extensions**

1. **Energy-aware scheduling**
    - Simulate CPU energy states (active vs idle).
    - Show estimated energy consumption under different schedulers.
2. **Multiprocessor Scheduling**
    - Simulate dual-core or multi-core scheduling (Global EDF, Partitioned RM).
3. **Task Dependencies**
    - Support Directed Acyclic Graphs (DAGs) to model precedence constraints.
4. **Priority Inversion Simulation**
    - Model resource locking and priority inversion.
    - Compare with Priority Inheritance protocol.
5. **Jitter Modeling**
    - Introduce variability in task release times and visualize its impact.

---

**4. Tools & Implementation Plan**

- **Language**: Python (preferred for fast prototyping and visualization).
- **Visualization**:
    - Matplotlib (Python desktop app).
    - Optional: D3.js (web-based interactive version).
- **Trace Input**:
    - Generated synthetic tasks.
    - User interactive input (add task at runtime).
    - Optional: load from CSV/JSON traces.
- **System Requirements**:
    - Runs on a general-purpose laptop (Windows/Linux/Mac).
    - No special hardware required.

---

**5. Project Phases**

1. **Task Model & Simulator Core**
    - Implement task objects (arrival, exec, deadline, priority).
    - Discrete time loop to simulate execution.
2. **Scheduler Implementations**
    - RM, EDF, DM, LLF → baseline.
    - Add OS schedulers for comparison (FCFS, RR, SJF).
3. **Visualization Layer**
    - Gantt chart with deadlines, idle times, and missed deadlines.
4. **Metrics & Reporting**
    - CPU utilization, response times, deadline misses → CSV export.
5. **Extensions**
    - Energy-aware, multiprocessor, priority inversion, jitter, DAG dependencies.

**6. Deliverables**

- **Simulator Software** (Python app).
- **Visualization Tool** (timeline plots, interactive GUI).
- **Documentation** (design decisions, how to run, sample traces).
- **Demo** showing multiple scheduling algorithms on the same workload.