# EECS 159A/CSE 181A
# Task Planning

# Outline

- Statement of Work
  - Divide up project into tasks and subtasks
  - Assign tasks to responsible individuals
  - Define **deliverables**
- Gantt Chart
  - Dependency among subtasks and across tasks
  - Estimate task effort, schedule tasks by week
  - Define milestones for synchronization

# Statement of Work

- Outline
  - Tasks & subtasks
- Details on the tasks & subtasks
  - Expanded outline of the task & subtasks
  - Can be complete sentences or paragraphs
- Deliverables
  - Associated with each task

# Steps in Generating Task Outline

- Decompose work into tasks & subtasks
  - by system architecture (horizontal)
  - by layer of abstraction (vertical)
- Cover all Stages of Development
  - design, implement, integrate, test, improve
  - Figure out dependency between tasks, cut unnecessary dependencies

# Divide and Conquer

- Divide:  more intuitive
  - Spatial: horizontal vs. vertical decomposition
  - Temporal:  stages of development
- Conquer: often neglected or underestimated
  - not automatic!  always takes more time and effort
  - must be modular to enable independent testing
  - try to start integrating early
  - should not wait until integration to do first test

# Tasks & Subtasks vs. Milestones
# Divide and Conquer

- Tasks: "Divide"
  - Horizontally: by Subsystems ("block diagram")
  - Vertically: Level of abstraction (HW, SW, comm)
  - Nature of work (technical, presentation, …)
- Milestones: "conquer"
  - Intermediate goals along the way to completion
  - Project or subproject level, can cut across tasks
  - Potential decision points to switch to Plan B

# Horizontal vs Vertical Decomposition

- Horizontal: by subsystems
  - Divide task by subsystem
  - Each person may need to work on multiple layers of abstraction

| | scanner | gateway | server |
|---|---|---|---|
| Person1 | v | | |
| Person2 | | v | |
| Person3 | | | v |

- Vertical: by layer of abstraction
  - e.g., hardware, HAL, firmware, software, protocol stack, app
  - Each person may need to work across multiple systems

| | PersonA | PersonB | PersonC |
|---|---|---|---|
| **application** | | | v |
| **firmware** | | v | |
| **hardware** | v | | |

# Example Decomposition 1: Mostly Horizontal

Tasks

1. Scanner Subsystem

telemetry
board
sensors
enclosure

2. Gateway Subsystem

Control & processing

Data Logging

Power

3. Server Subsystem

UI
MVC
Comm.

This means you can (probably) develop in parallel, but you may spend a lot more time in integration.

# Example Task Outline 1: Horizontal Decomposition

1. Scanner Subsystem
    1.1. Component research
    1.2. Interface definition
    1.3. Enclosure design
    1.4. Telemetry design
    1.5. Presentation
2. Gateway Subsystem
    2.1. Data logging subsystem
    2.2. Bluetooth driver
    2.3. Power subsystem
    2.4. Sensor interfacing
    2.5. Presentation
3. Server Subsystem
    3.1. Define Database Schema
    3.2. Access Control Policy
    3.3. Python Coding
    3.4. Presentation
4. Integration
    4.1. Integrating Scanner & Gateway
    4.2. Integrating Gateway & Server
    4.3. Integrating all subsystems
    4.4. Demo video
    4.5. Final report

# Issues with Horizontal Decomposition

- Advantages:
  - Each person is responsible for own subsystem
  - Can develop in parallel, minimal dependency - till later
- Issues
  - Each person needs to know several layers of abstraction
  - e.g., both Person2 & Person3 would need to know hardware, firmware, etc
  - They might start talking to each other too late!

| | scanner | gateway | server |
|---|---|---|---|
| Person1 | application v<br>firmware v<br>hardware v | | |
| Person2 | | application v<br>firmware v<br>hardware v | |
| Person3 | | | v |

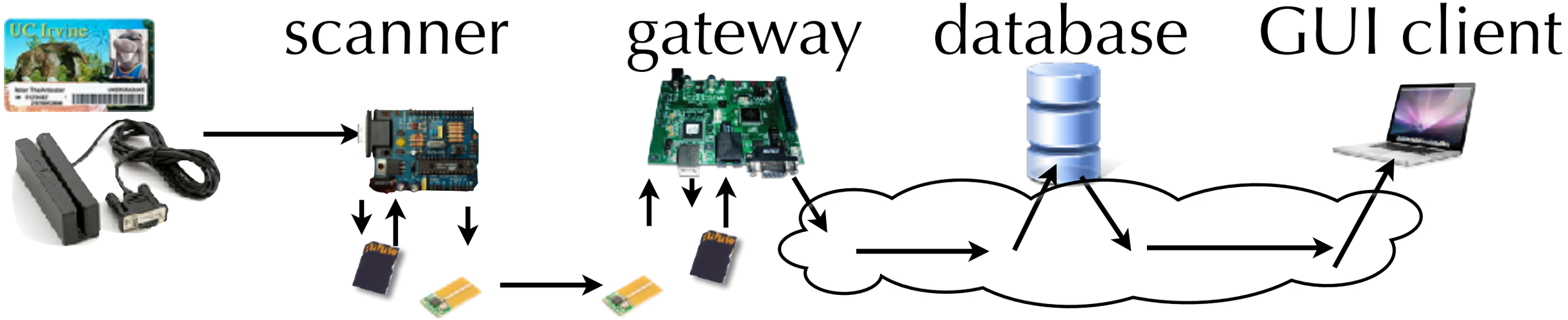
both are embedded systems
both contain hardware, firmware, device interfacing, …

one person must be able to build the whole scanner (hardware, firmware), another person must be able to build the whole gateway, …

# Example Task Outline 2: Vertical Decomposition

1. Hardware
   - 1.1. Component research
   - 1.2. Schematic for scanner, gateway
   - 1.3. Enclosure for scanner, gateway
   - 1.4. PCB layout and assembly for..
   - 1.5. Hardware Testing for …
2. Firmware
   - 2.1. Firmware architecture for scanner, gateway
   - 2.2. Device Drivers and HAL for scanner, gateway
   - 2.3. Bluetooth Communication between scanner, gateway
   - 2.4. Power Management for scanner, gateway
   - 2.5. Firmware testing
3. Application
   - 3.1. Application layer for scanner
   - 3.2. Application layer for gateway
   - 3.3. Application layer for server
4. Integration & Presentation
   - 4.1. Poster 1
   - 4.2. Testing Scanner, Gateway, Server
   - 4.3. Demo video
   - 4.4. Final report

# Example Decomposition 2: Mostly Vertical




3. App

1. Firmware

2. Hardware

| application | application | application | application |
|---|---|---|---|
| driver | power mgmt.<br>protocol stack<br>driver | protocol stack | protocol stack |
| HAL | HAL | server hardware | |
| schematic | schematic | | |
| board layout | board layout | | |
| telemetry | telemetry | | |
| enclosure | enclosure | | |

This means the same "domain expert" may have to work across multiple subsystems!

# Issues with Vertical Decomposition

- "Advantages"
  - Each person just needs to be an expert on their own area (e.g., hardware, HAL, firmware, software, protocol stack, app)
  - No need to learn another field from scratch
- Issues
  - e.g., Each person may need to work across multiple subsystems
  - Logistically may be more difficult: might need multiple units of the same MCU boards, debuggers, etc to develop in parallel

| | PersonA | PersonB | PersonC |
|---|---|---|---|
| **application** | | | **scanner** v, **gateway** v, **server** v |
| **firmware** | | **scanner** v, **gateway** v, **server** | |
| **hardware** | **scanner** v, **gateway** v, **server** | | |

# So, should you assign tasks vertically or horizontally?

- If your team's skill set is…
  - from different majors => vertical may be better
  - different majors may contribute at different levels
- If your system organization is..
  - mostly networked => horizontal may be better
  - can proceed in parallel, minimal dependency, black-box testing
- Often mixed vertical and horizontal

# Common Hardware Tasks

- component evaluation
- component ordering
- schematic design
- schematic checking (inspection, simulation)
- layout & floor planning
- assembly
- testing
- allow time for another hardware iteration

# Refining Task into Detailed SOW

1. Hardware

1.1. Component research

- Find options for ID scanner (barcode, QRcode)
- **<u>Deliverable</u>**: ordered scanner

1.2. Enclosure design

- Draw enclosure 3D model in Solidworks
- Print sample enclosure for fitting PCB and for look and feel
- Revise 3D model to meet constraint
- **<u>Deliverable</u>**: 2 units of 3D-printed enclosures

1.3. …

# Common Mechanical Design Tasks

- conceptualization
- CAD modeling
- 3D printing
- fitting, post processing
- assembling
- testing
- design revision, fine tuning

# Common Software Tasks

- software block diagrams
- defining API and data structures
- writing header files
- coding
- testing
- debugging
- documentation

# Common Management Tasks

- settle on tools & method
- problem statement
- list requirements
- evaluate solutions
- budgeting
- make purchases
- document work

# Organize project tasks

- Given a problem
  - Identify what is ready solution vs. work that needs to be done
- Organize tasks by category
- Figure out dependencies
  - True dependency vs. pseudo-dependency
  - Anticipate delays outside your control

# Task Assignment and Scheduling

- Now that we have SOW
- Need to assign tasks to team members
  - Workload should be balanced
  - Tasks should match the person's skill set
- Need to schedule tasks on a timeline
  - Figure out dependencies among tasks
  - May need to work backwards from deadline!

# Team Organization

- Choose a "balanced" organization
  - individual: between technical & nontechnical tasks
  - team: equitable workload among members
- Hierarchy
  - Flat (1 level) preferred; no more than 2 levels
  - Assign responsibility to individuals
    => that person makes decision in that aspect

# Project Planning

- Input: tasks and subtasks
  - Estimate the amount of time required
  - figure out dependencies between tasks
- Place subtasks onto Gantt Chart
  - Optionally add arrows to show dependencies
- Identify milestones
  - Goals that cut across tasks
  - Potential points for fall-back plans (Plan A, B, …)

# Considerations for Task Organization

- Responsibilities vs. execution (in a small team)
  - The most-qualified person should be responsible
  - Most people should manage themselves
- Integration Task
  - Could be its own task but involves everyone
  - Could integrate subsystems before entire system
  - Integration almost always takes longer than expected
- Static vs. dynamic tasks
  - Allow enough slack for unexpected tasks to arise

# Task Assignment

- 1 top-level task per one person
  - Top-level task includes all its subtasks
  - Assignee = coordinator, likely also doer
- Check entries for ownership
  - put an [x] in rows within your own column
- Idea: letter encoding for status
  - need people, waiting on decision, completed, …

| Task | Peter | Ann | Jose |
|---|---|---|---|
| 1. Hardware | X | | |
| 1.1 Components | x | | |
| 1.2 PCB layout | x | | |
| 1.3 Enclosure | | | x |
| 1.4 … | x | | |
| 2. Firmware | | X | |
| 2.1 Architecture | | x | |
| 2.2 Driver & HAL | | x | |

# Timeline

- Time Granularity and Range
  - Usually good to plan by 1-week or 2-weeks
  - Day may be too fine-grained
- Reference real calendar

# Making Gantt Chart

- X-axis: Time (by week or 2 weeks)
- Y-axis: Tasks and subtasks

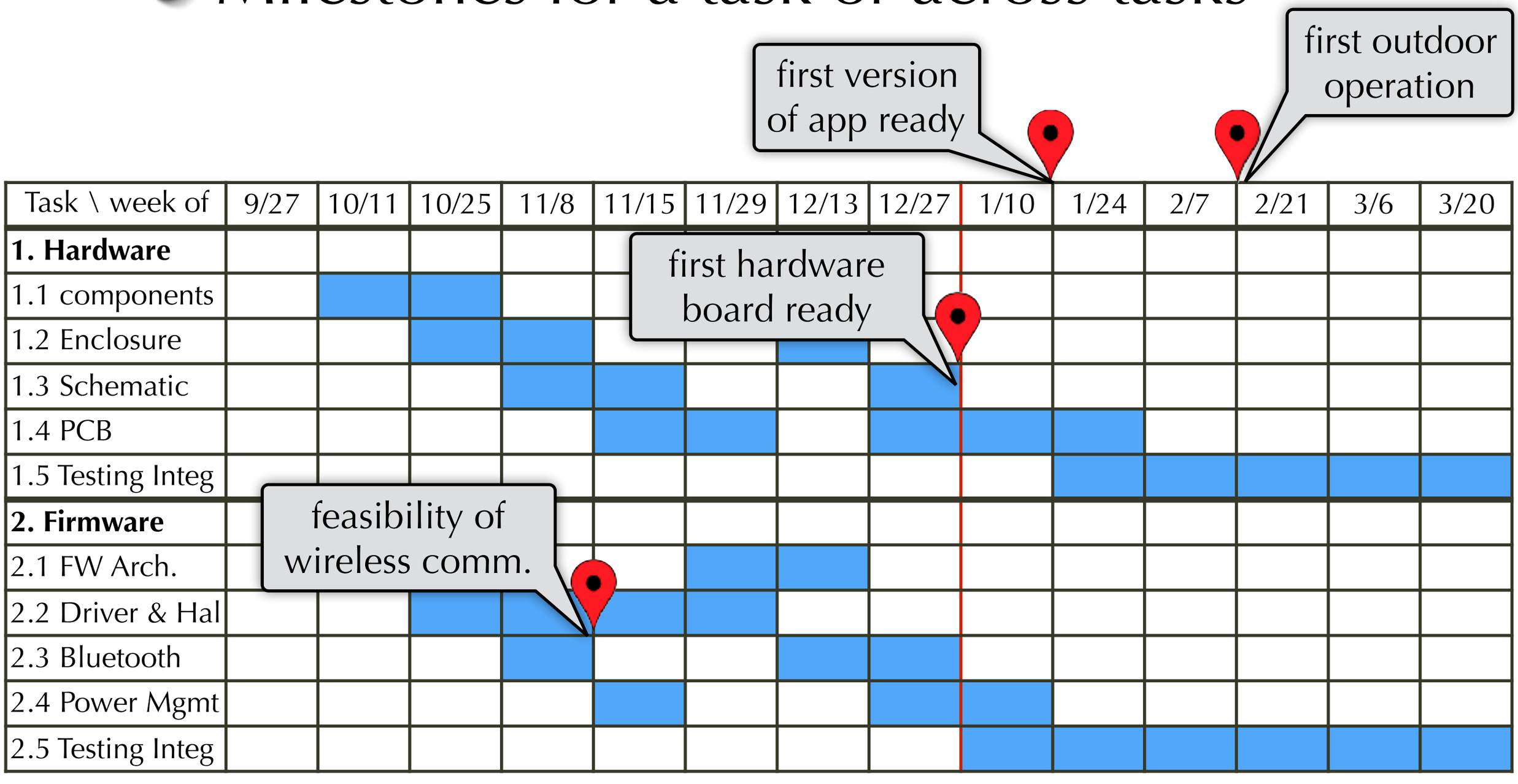
| Task \ week of | 9/27 | 10/11 | 10/25 | 11/8 | 11/15 | 11/29 | 12/13 | 12/27 | 1/10 | 1/24 | 2/7 | 2/21 | 3/6 | 3/20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1. Hardware** | | | | | | | | | | | | | | |
| 1.1 components | | | | | | | | | | | | | | |
| 1.2 Enclosure | | | | | | | | | | | | | | |
| 1.3 Schematic | | | | | | | | | | | | | | |
| 1.4 PCB | | | | | | | | | | | | | | |
| 1.5 Testing Integ | | | | | | | | | | | | | | |
| **2. Firmware** | | | | | | | | | | | | | | |
| 2.1 FW Arch. | | | | | | | | | | | | | | |
| 2.2 Driver & Hal | | | | | | | | | | | | | | |
| 2.3 Bluetooth | | | | | | | | | | | | | | |
| 2.4 Power Mgmt | | | | | | | | | | | | | | |
| 2.5 Testing Integ | | | | | | | | | | | | | | |

# Milestones

- Intermediate goals
  - Important achievements before completion
  - Cuts across tasks. Enabler for next (sub)phase of work
  - Potential decision points to switch to Plan B
- Examples
  - Feasibility of wireless data transfer
  - Hardware board ready
  - First version of app ready for testing with board
  - First outdoor operation of sensors without enclosure

# Integrating Milestones into Gantt Chart

- Milestones for a task or across tasks




| Task \ week of | 9/27 | 10/11 | 10/25 | 11/8 | 11/15 | 11/29 | 12/13 | 12/27 | 1/10 | 1/24 | 2/7 | 2/21 | 3/6 | 3/20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1. Hardware** | | | | | | | | | | | | | | |
| 1.1 components | | | | | | | | | | | | | | |
| 1.2 Enclosure | | | | | | | | | | | | | | |
| 1.3 Schematic | | | | | | | | | | | | | | |
| 1.4 PCB | | | | | | | | | | | | | | |
| 1.5 Testing Integ | | | | | | | | | | | | | | |
| **2. Firmware** | | | | | | | | | | | | | | |
| 2.1 FW Arch. | | | | | | | | | | | | | | |
| 2.2 Driver & Hal | | | | | | | | | | | | | | |
| 2.3 Bluetooth | | | | | | | | | | | | | | |
| 2.4 Power Mgmt | | | | | | | | | | | | | | |
| 2.5 Testing Integ | | | | | | | | | | | | | | |

# Plan B

- Needed to combat uncertainty
  - Allows some deviation from ideal goal
  - But still showcase most of the proposed functions
- e.g.: want to custom-make PCB, but doesn't work
  - Plan B:  Use eval. board in place of custom board
- e.g.: power management causes noisy sensor data
  - Plan B:  Disable power management
  - make sure the system functions correctly, manage power later

# Do's & Don't's on Plan B

- Identify your priorities
  - What is crucial, and what's bonus?
- Avoid doing a radically different Plan B
  - Encapsulate Plan B in statement of work
  - Better to build in enough slack into schedule
- The rest of plan should remain stable
  - Try to isolate impact of Plan B on the rest of project
  - Keep the same milestones
  - Make Gantt chart appear "unconditional"

# Task Dependencies

- Need to do Task A before Task B
- Example: hardware tasks
  - order components, make PCB first, before you can solder components onto PCB
- Example: software tasks
  - define API, write the code, before you can test the code.
- Sounds kind of obvious, but…

# Pseudo-dependencies

- Some dependencies are not real!
- Example: want to make a custom board
  - Can't start writing code until board is ready
    => really? Think again!
- Example: Testing Scanner Communication
  - Can't test scanner until gateway is ready
    => really? Think again

# Resource Dependencies

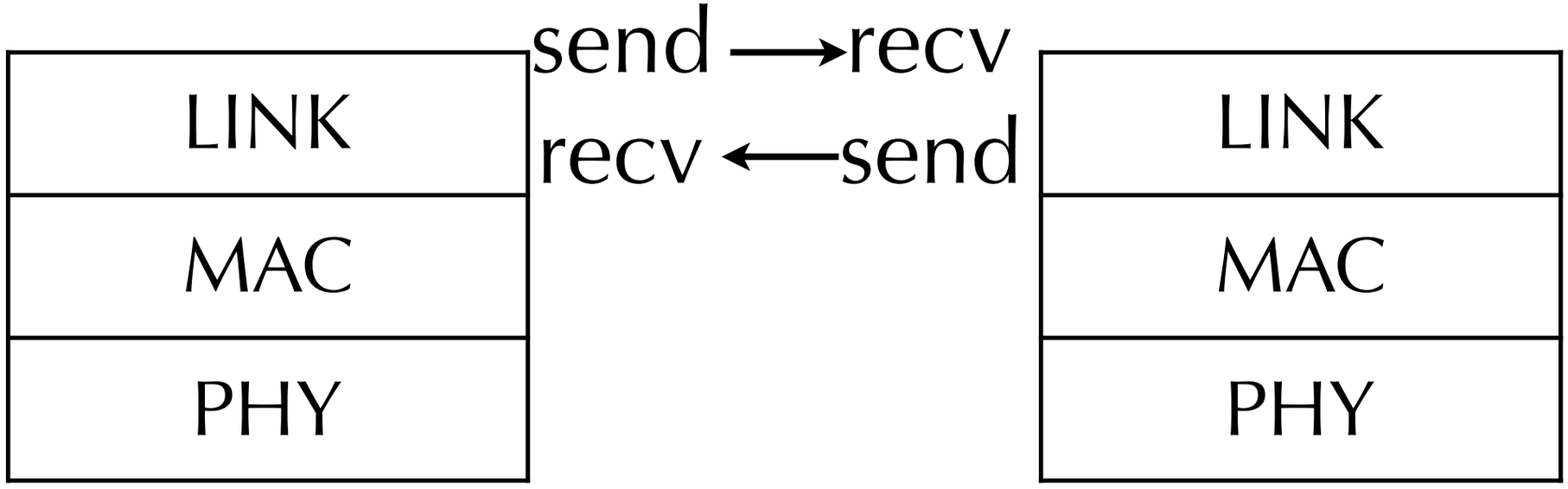
- Example Resources
  - hardware board, compiler (limited license)
- Solution: duplicate resources
  - one board not enough for parallel development
    => buy more boards!
- Solution: stand-in
  - Before custom board is ready, buy existing evaluation board for the same processor to run code
  - No excuse for software and hardware people to blame each other for stalling their progress!

# Testing/Debugging tips

- How to test a scanner before the gateway is ready?
- Several options
  - API stub routines in scanner code
  - Computer + RF module (over UART) as stand-in gateway

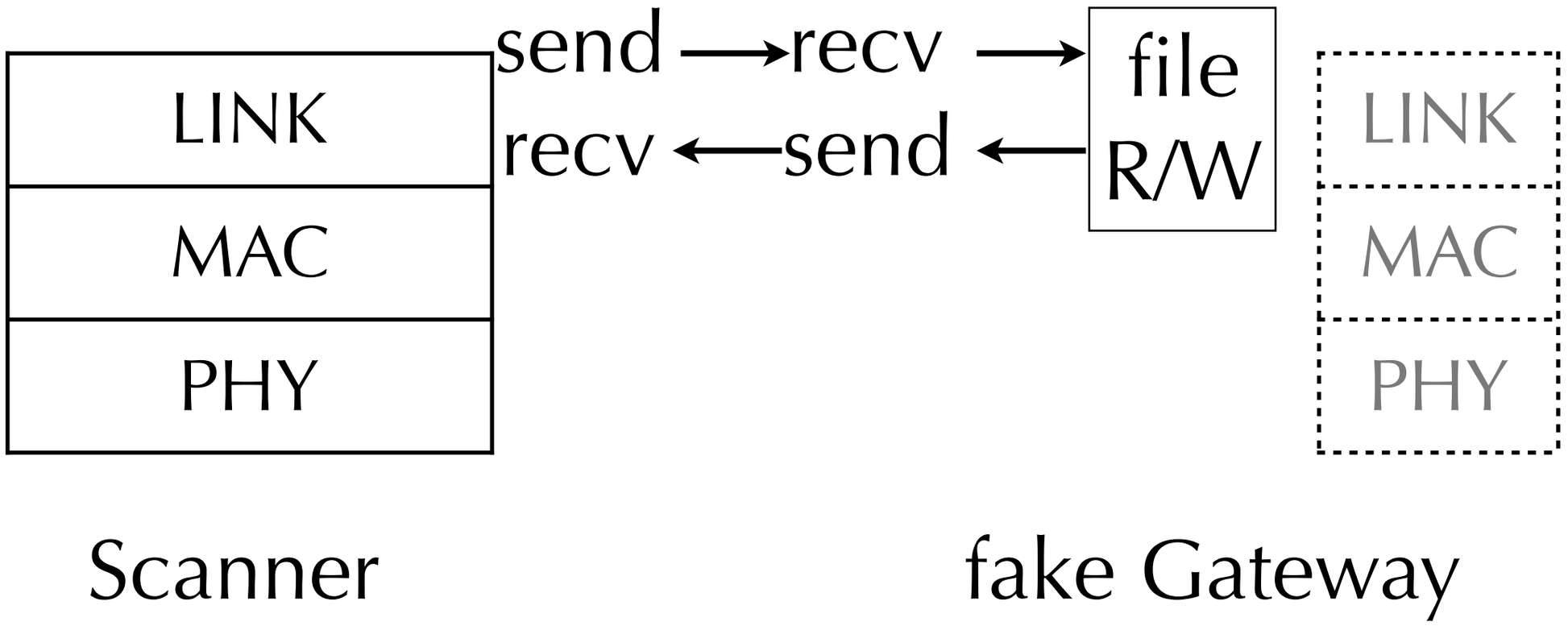
# Option 1: API Stubs

- At a given level of abstraction, there are corresponding routines on both sides



# API Stubs for testing

- When the other side doesn't exist, replace it with a file reading/writing (or GUI) stub!
- Same idea works for SD card, scanner, etc...

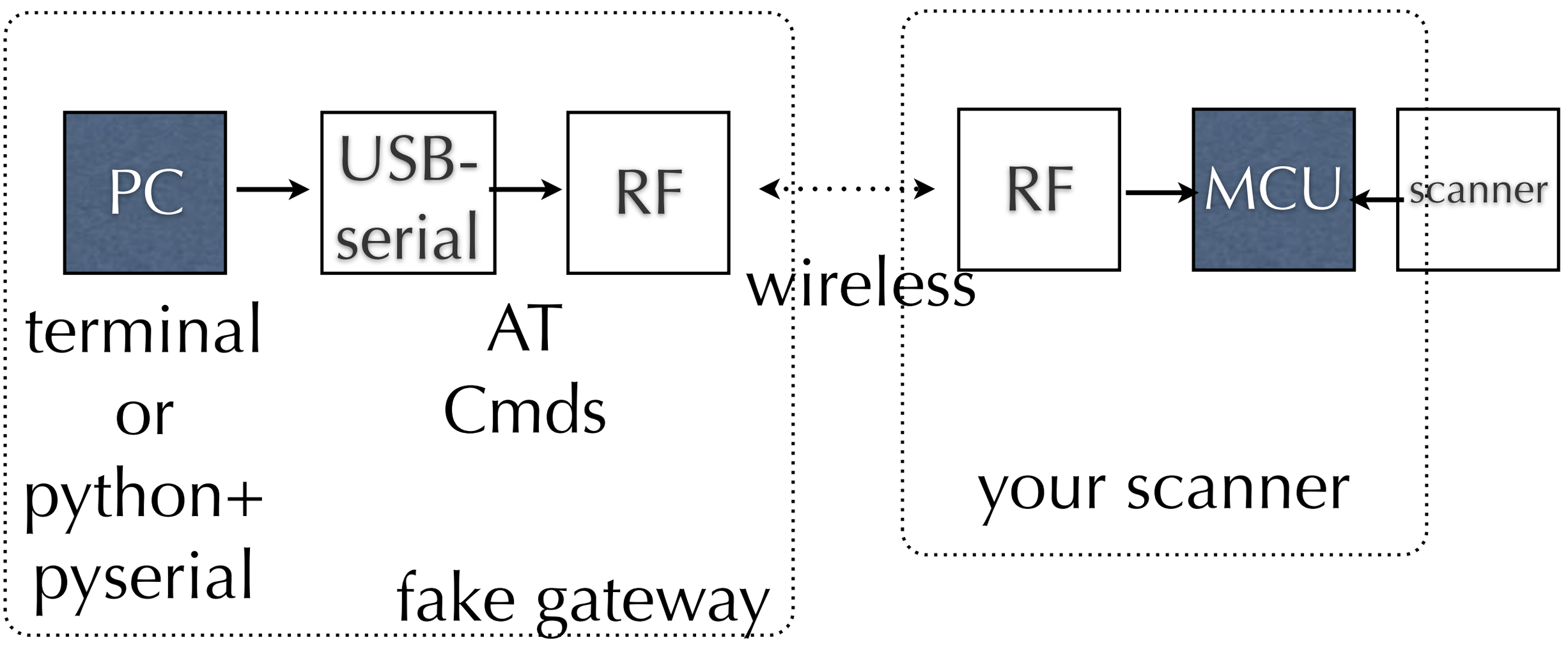


# Option 2: Computer+RF module

- Works for serial port, maybe USB too
  - e.g., XBee, which uses AT commands
  - works for serial scanner as well
- Can replace MCU board with PC
  - Hyperterminal to type in command
  - better option: Python+pyserial

# Option 2: Computer + RF module

- PC as a stand-in for an MCU



# Summary

- Divide up project into tasks and subtasks
- Assign tasks to responsible individuals
- Define deliverables
- Find dependency among subtasks and across tasks
  - Identify and eliminate pseudo-dependencies
- Estimate task effort, schedule tasks by week
- Define milestones for synchronization