Universität Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

Universitätsstraße 38
D-70569 Stuttgart

Mobile Computing Lab
Assignment 1

Implementation and Evaluation of Multiplex Mechanisms for Wireless Media

Frank Dürr, Christoph Dibak, Ahmad Slo

Outline

- Motivation and goal
- Implementation of multiplex mechanisms
  - Task 1: TDMA
  - Task 2: CDMA
- Organizational issues

Motivation and Goal

Motivation:

- **Characteristics of wireless radio communication**
  - Shared media
  - ➔ Need multiplex mechanisms to implement more than one channel

Goal:

- Implement **multiplex mechanisms** for a wireless system
  - Time-division multiple access (TDMA)
  - Code-division multiple access (CDMA)

Universität Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

Universitätsstraße 38
D-70569 Stuttgart

Task 1
Time Division Multiple Access (TDMA)

Time Division Multiplex (TDM)

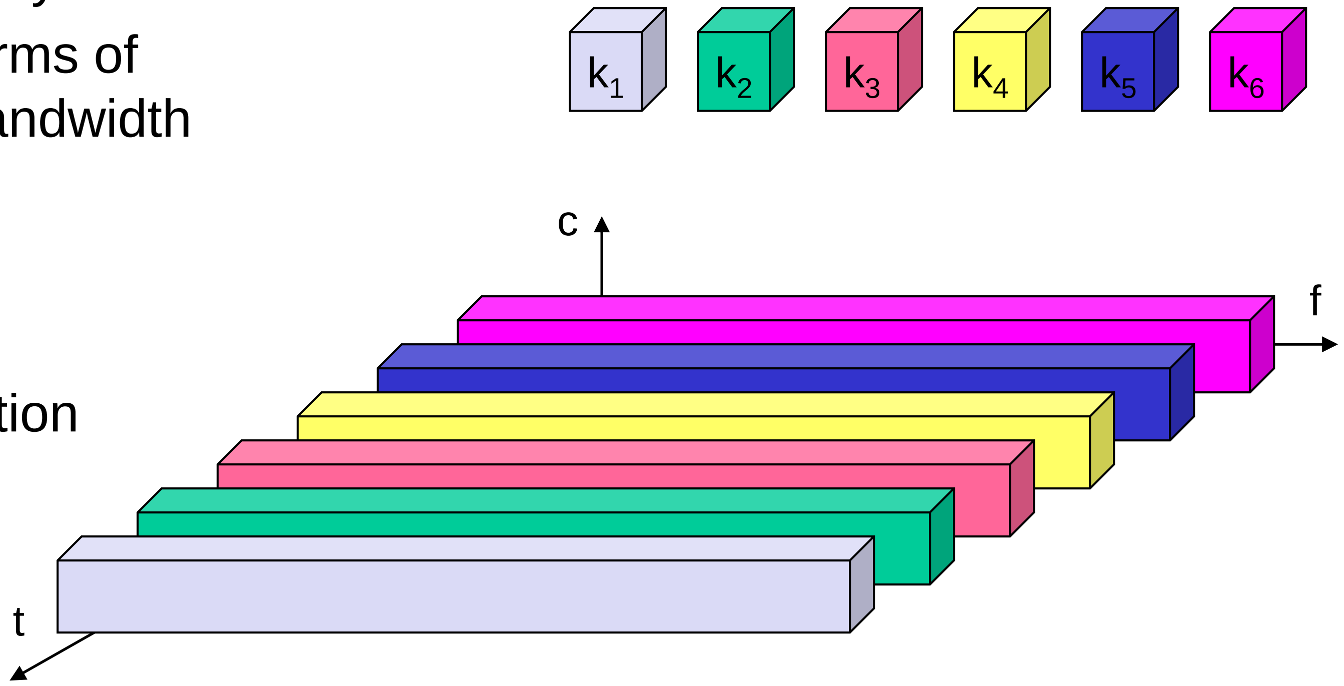
A channel gets the whole spectrum for a **certain amount of time**

Advantages:

- only one carrier in the medium at any time
- flexible in terms of assigning bandwidth

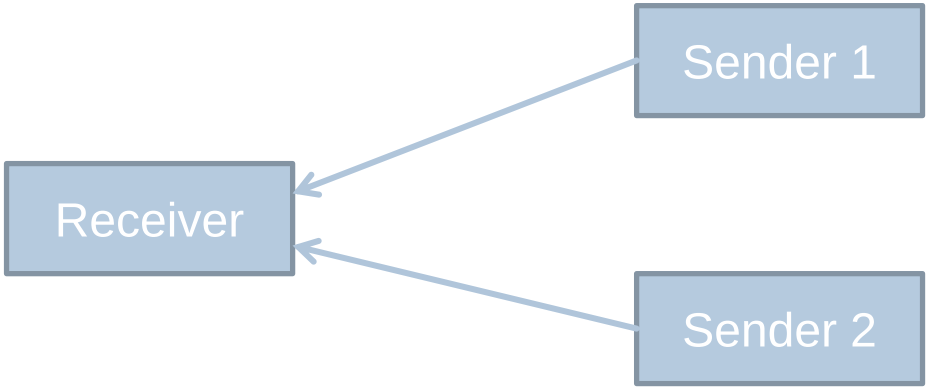
Disadvantages:

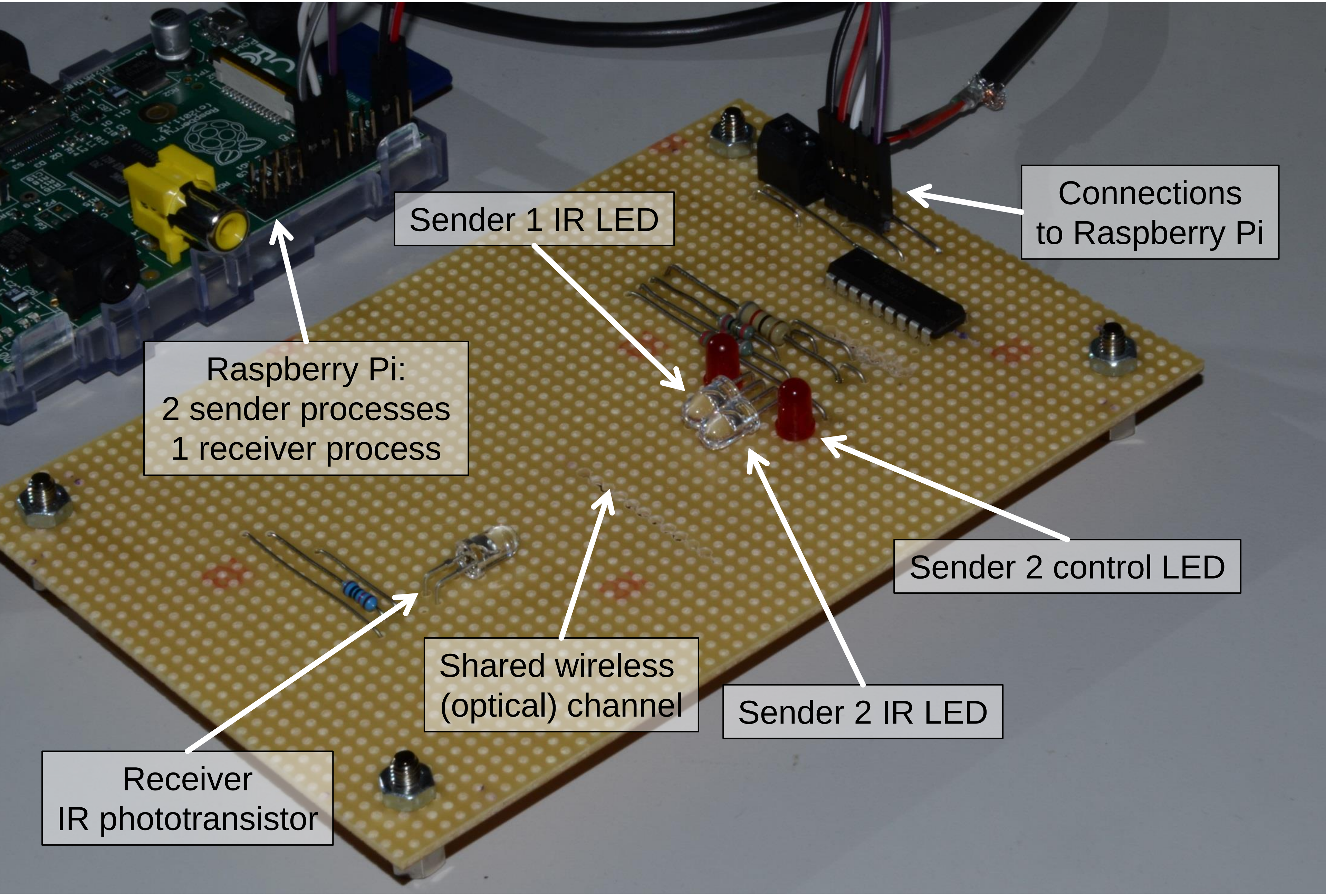
- precise synchronization necessary



Experiment Setup

- 2 senders
- 1 receiver
- Shared wireless channel
  - Optical: infrared (850 nm wave length)
- Unidirectional communication

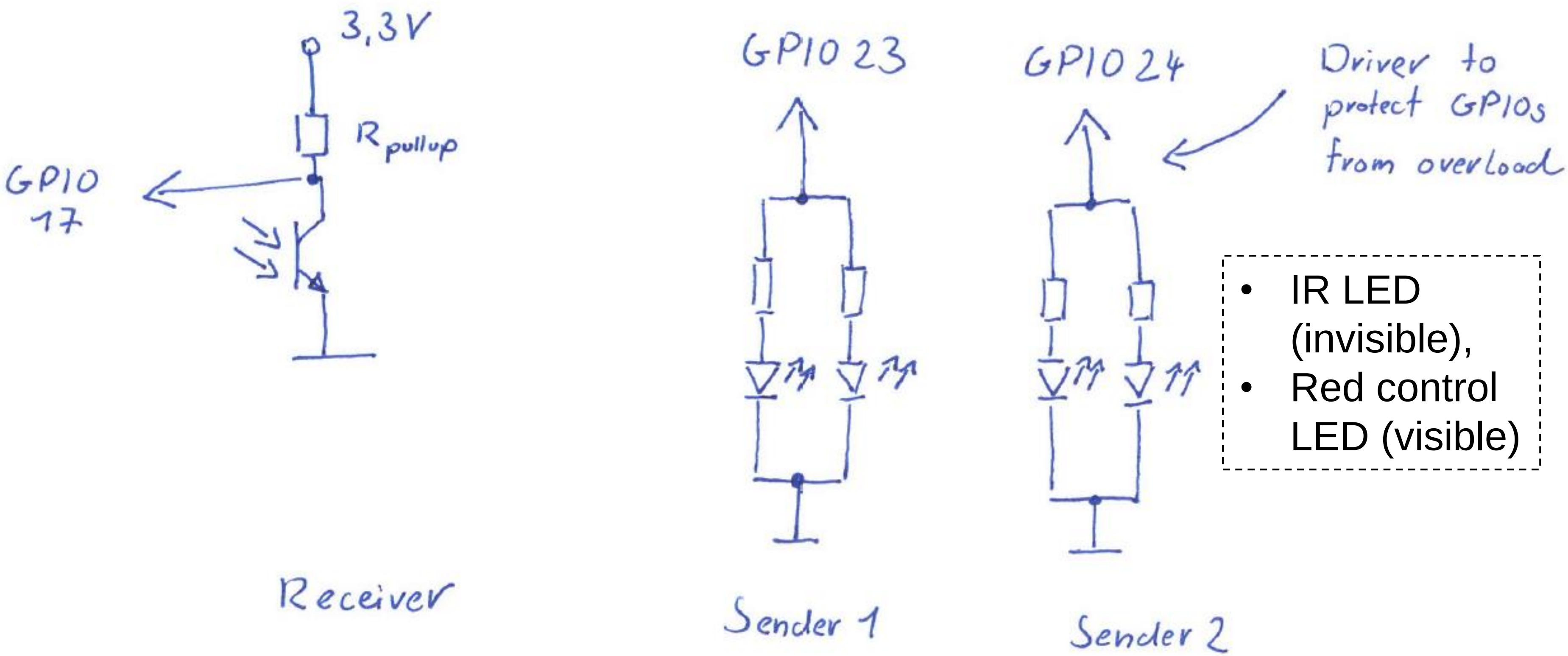



Connections to Raspberry Pi
Sender 1 IR LED
Raspberry Pi:
2 sender processes
1 receiver process
Sender 2 control LED
Shared wireless (optical) channel
Sender 2 IR LED
Receiver IR phototransistor

Experiment Setup

- Pullup resistor pulls GPIO high (3.3 V) when no IR signal is received
- Phototransistor pulls GPIO low (Gnd) when IR signal is received
- Inverted signal!

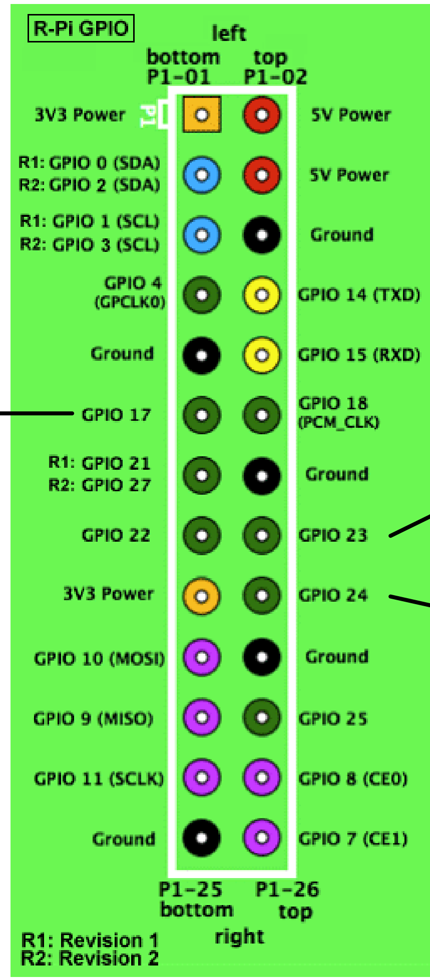
GPIO: General Purpose IO pins of Raspberry Pi
- Binary input (receiver) or output (sender)



- IR LED (invisible),
- Red control LED (visible)

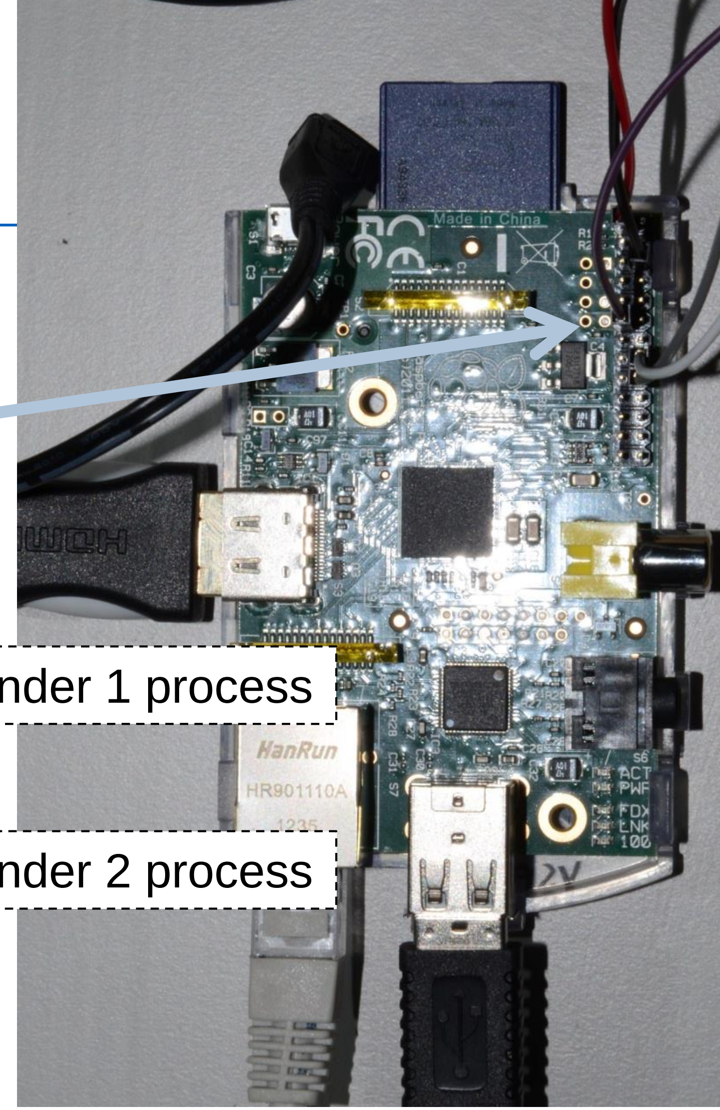
GPIOs of Raspberry Pi

Receiver process



Sender 1 process

Sender 2 process



Controlling GPIOs with Python

```
#!/usr/bin/python

import RPi.GPIO as GPIO
import time

# Use BCM numbering
GPIO.setmode(GPIO.BCM)

GPIO.setup(23, GPIO.OUT)

while True:
    GPIO.output(23, GPIO.HIGH)
    time.sleep(1)
    GPIO.output(23, GPIO.LOW)
    time.sleep(1)
```

Controlling GPIOs with Python

```
#!/usr/bin/python

import RPi.GPIO as GPIO
import time

# Use BCM numbering
GPIO.setmode(GPIO.BCM)

GPIO.setup(17, GPIO.IN)

while True:
    value = GPIO.input(17)
    print value
    time.sleep(1)
```

Never ever configure GPIO 17 as output!

- No protection against overloading the output!

Task 1.1: Sending with One Sender

- Implement a Python program that sends the text “Hello World!“ from one sender
  - Encode text into bit pattern (e.g., ASCII code)
  - Send bit pattern by turning LED on and off via GPIO pin 23
  - Send text in an endless loop
- Implement a Python program that receives the text
  - Decode binary signals from GPIO 17
  - Translate binary signal to text
  - Output text on console
- Run both processes on the same host (Raspberry Pi)
- Hints:
  - Sender and receiver processes share the same (perfectly synchronized) clock → can be used to synchronize sender and receiver
  - Longer bit times make it easier to synchronize

Task 1.2: Multiple Senders with TDMA

- Implement a Time Division Multiple Access (TDMA) scheme
  - 2 sender processes
  - 1 receiver process
  - Run all three processes on the same host (Raspberry Pi)
- Send text “HELLO FROM SENDER 1” and “hello from sender 2” in endless loops
- Output text received by receiver on console
  - To keep things simple: one stream of interleaved characters: “HELLhellO FROMo from…”
- Try to send as fast as possible (evaluate maximum possible bit rate)
- Hint: Processes share the same (perfectly synchronized) clock
  - Can be used to synchronize senders

Universität Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

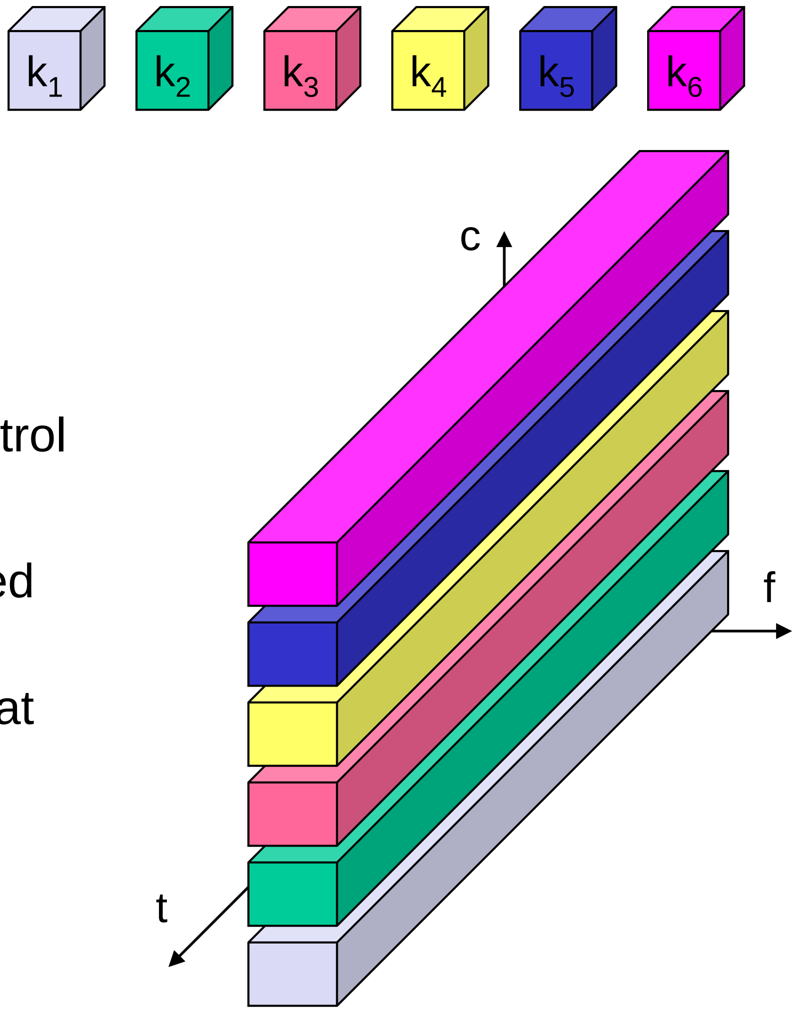
Universitätsstraße 38
D-70569 Stuttgart

Task 2
Code Division Multiple Access (CDMA)

Code Division Multiplex (CDM)

Each channel has **a unique code**

- All channels use the same spectrum at the same time



Advantages:

- bandwidth efficient
- no coordination in terms of access control

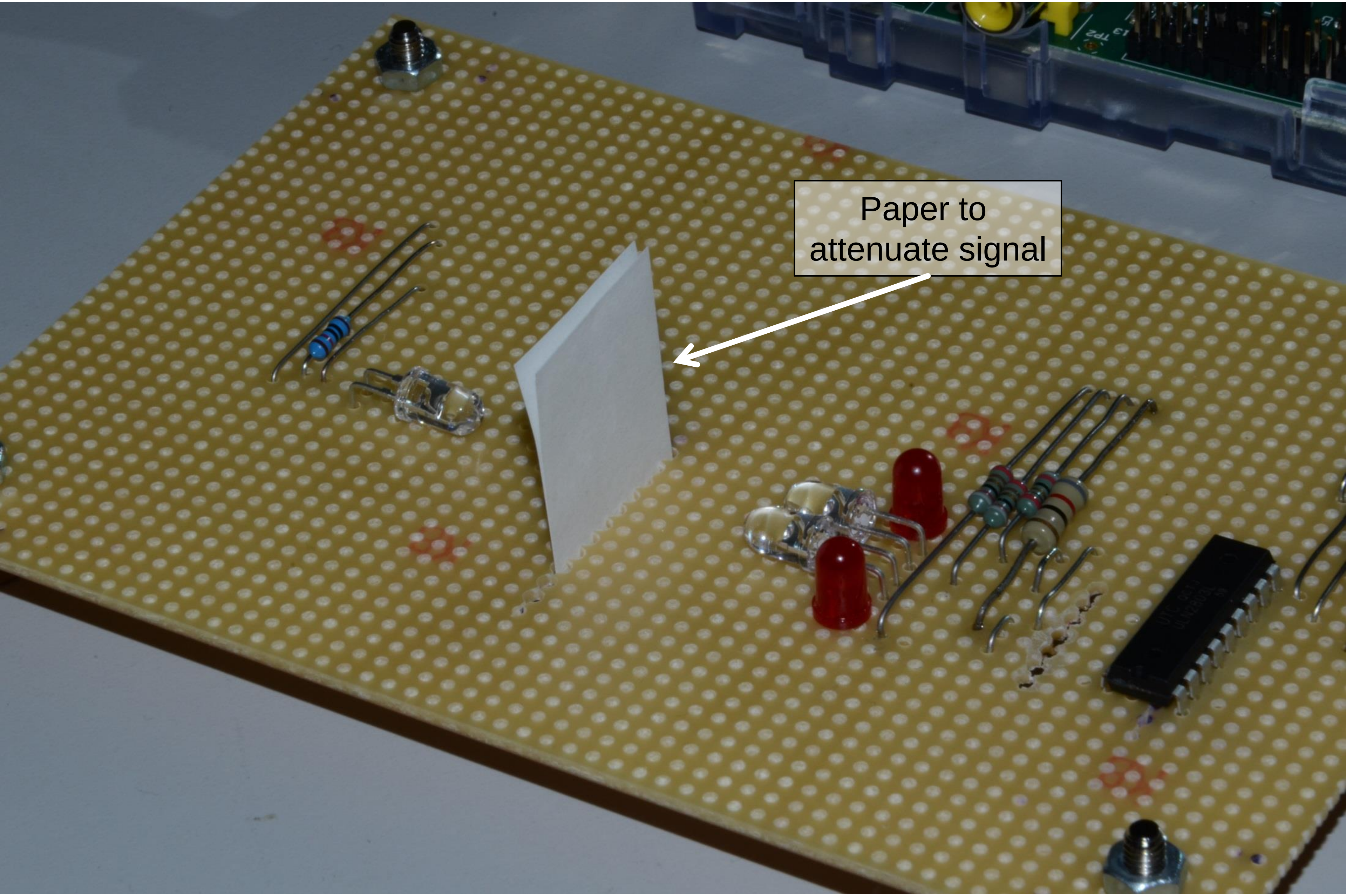
Disadvantages:

- receiver must be precisely synchronized with transmitter
- all signals should have same strength at receiver

Implemented using spread spectrum technology

Experiment Setup

- Binary receiver replaced by analog/digital (AD) converter
  - Can measure the signal strength on the channel
  - Can distinguish whether two senders are sending (at the same time) or only one sender, or no sender
- 10 bit AD converter implemented by Arduino
  - Converts voltage at phototransistor to value in range [0,1023]
    - Again: Inverted!
  - Sends value to Raspberry Pi via serial console `/dev/ttyACM0`
- Direct signal from one LED too strong
  - Already one active LED/sender alone would lead to value 0
  - Insert paper between sender and receiver to attenuate signal (see next photo)

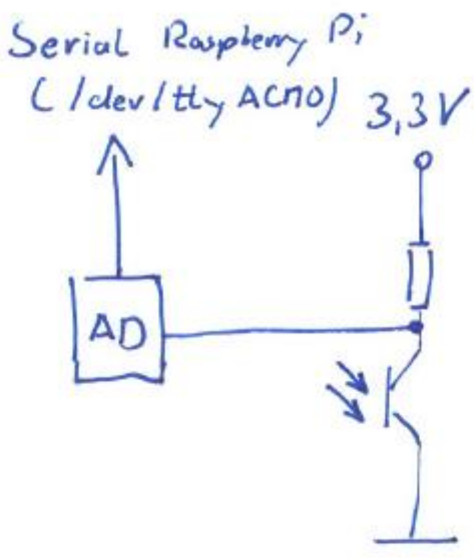

Paper to
attenuate signal

Experiment Setup

- Ambient light leads to noise on channel
- Put box over board to reduce noise – or only work at night ☺

AD Converter

- Sampling rate: 100 Hz
- Serial settings /dev/ttyACM0:
  115200 baud, 8N1 (8 bit, no parity, 1 stop bit)
- Text format: one value (ASCII characters) per line



Serial Interface in Python

```
#!/usr/bin/python

import serial

ser = serial.Serial(port='/dev/ttyACM0', baudrate=115200)

while True:
    # Receives analog readings froim Arduino at 100 Hz
    line = ser.readline()
    line = line.rstrip()
    print line

ser.close()
```

Task 2.1: Calibration

- Implement a serial receiver in Python
  - Reading values from /dev/ttyACM0
  - Writing values to console
- Implement an application that switches on LEDs periodically
  - LED 1=off, LED 2=off
  - LED 1=on, LED 2=off
  - LED 1=off, LED 2=on
  - LED 1=on, LED 2=on
- Record output of receiver for the different states of the LEDs
- Define value ranges to safely identify the following states:
  - Both LEDs off
  - 1 LED on, 1 LED off
  - Both LEDs on

Task 2.2: Multiple Senders with CDMA

- Implement a Code Division Multiple Access (CDMA) scheme
  - 2 sender processes (now sending simultaneously!)
  - 1 receiver process
  - Run all three processes on the same host (Raspberry Pi)
- Define chipping sequences for both senders (Walsh/Hadamard code)
- Implement senders sending text
  - Sender 1: “HELLO FROM SENDER 1”
  - Sender 2: “hello from sender 2”
  - Translate ASCII text to binary code
  - Send binary code using sender’s chipping sequence
    - Hint: You have to send multiple “chips” per bit (spreading)!
    - Hint: You have to synchronize both senders

Task 2.2: Multiple Senders with CDMA

Implement receiver

- Translate values from serial interface to the calibrated three states of Task 2.1
  - *x* LEDs on
- Translate the three states (*x* LEDs on) to bipolar notation
  - Hint: Assume that both senders are sending continuously, i.e., you do not have to distinguish an inactive sender from a sender sending a 0 chip
- Multiply bipolar sequence with bipolar chipping sequences of both senders
- Integrate (sum up) results and translate to binary code
- Translate binary code to ASCII code
- Output text of both senders
  - To keep things simple: one stream of interleaved characters: "HELLhellO FROMo from…"

Working Places

- Located in **Nexus Lab (0.353)**
  - When you enter the room, to the left at the window
- Can enter lab with your student id
- **Login information:**
  - One account per team on each Rpi
    - Home directories are not mirrored!
  - Username: **teamX**
  - Remote login (SSH)
    - Hostnames: vspi-mcl-01, vspi-mcl-02 and vspi-mcl-03
    - **Do not login if another team has the slot!**
  - You can use sshfs to mount the pi on your laptop
  - Can use multiple terminals (Ctrl-Alt-Fx), `screen`, `tmux`, or X

Resource Reservation

- Three identical working places
  - Three teams can work in parallel

- Teams can book time slots:
  - http://goo.gl/GdQhN4

- Each team must book at most one slot at a time!

Note about System Setup

- We will change the hardware setup after one week
  - April 20 – April 27: Task 1 setup
  - After April 27: Task 2 setup (including Arduino & ADC)

- Please try to finish Task 1 within the given period

Submission & Next Meeting

- Post questions on the forum
- You have **2 weeks time** to work on this assignment until the final date of submission!
  - Next assignment and ~~demonstration of your results~~ scheduled for **Wednesday May 4**
  - Demonstration of results on **Wednesday May 11**
- **Submit via Ilias** at least the night before the demonstration meeting
  - **Source code** and presentation (**slides**) of you evaluation results
  - Group submission!

Group presentation of assignment results

- Three or four groups will give a short presentation of their results
  - about **10 – 15 minutes of presentation**
  - **each group member should actively participate**
  - **Groups selected randomly**

- Presentation must contain…
  - Explanation of your solution
  - Performance results (numbers)
  - Experience with real world tests
    - E.g., encountered problems
  - Live demonstration of solution (if possible)
    - Proof of concept – does it work?

Group presentation of assignment results

- All other groups demonstrate their results individually
  - **each group member should actively participate**

- Schedule will be announced on ILIAS
  - time slot: Wednesdays, 15:45 to 19:30 in 0.353 (Nexus Lab)

Questions?