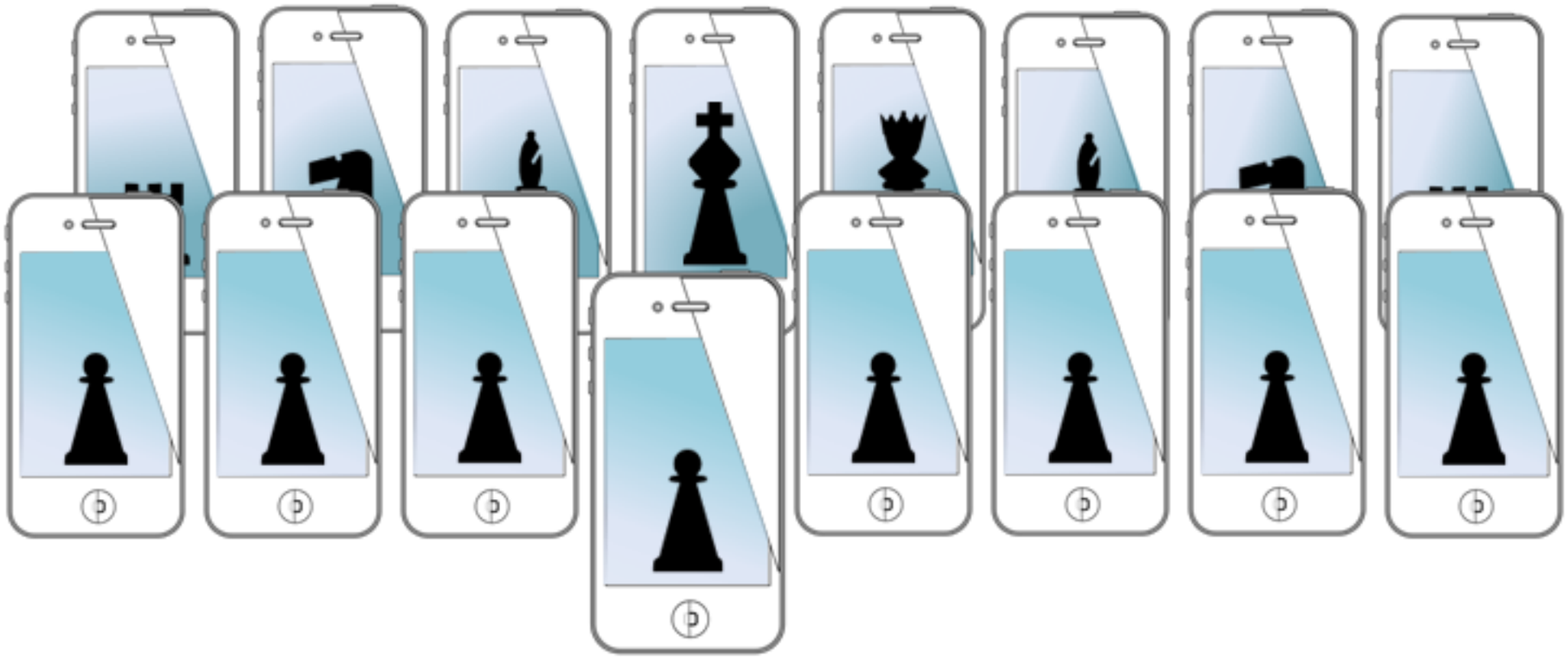
MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture fourteen: microcontroller basics

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

course logistics

- A3 grades have been up for most of the break
- A4 is due this Friday!

agenda

- microcontrollers
  - architecture
  - peripherals
  - common usage
- Arduino sketches
  - the Atmega328

embedded systems

- cameras
- microcontrollers
- FPGAs
- internet of things
- many, many, things

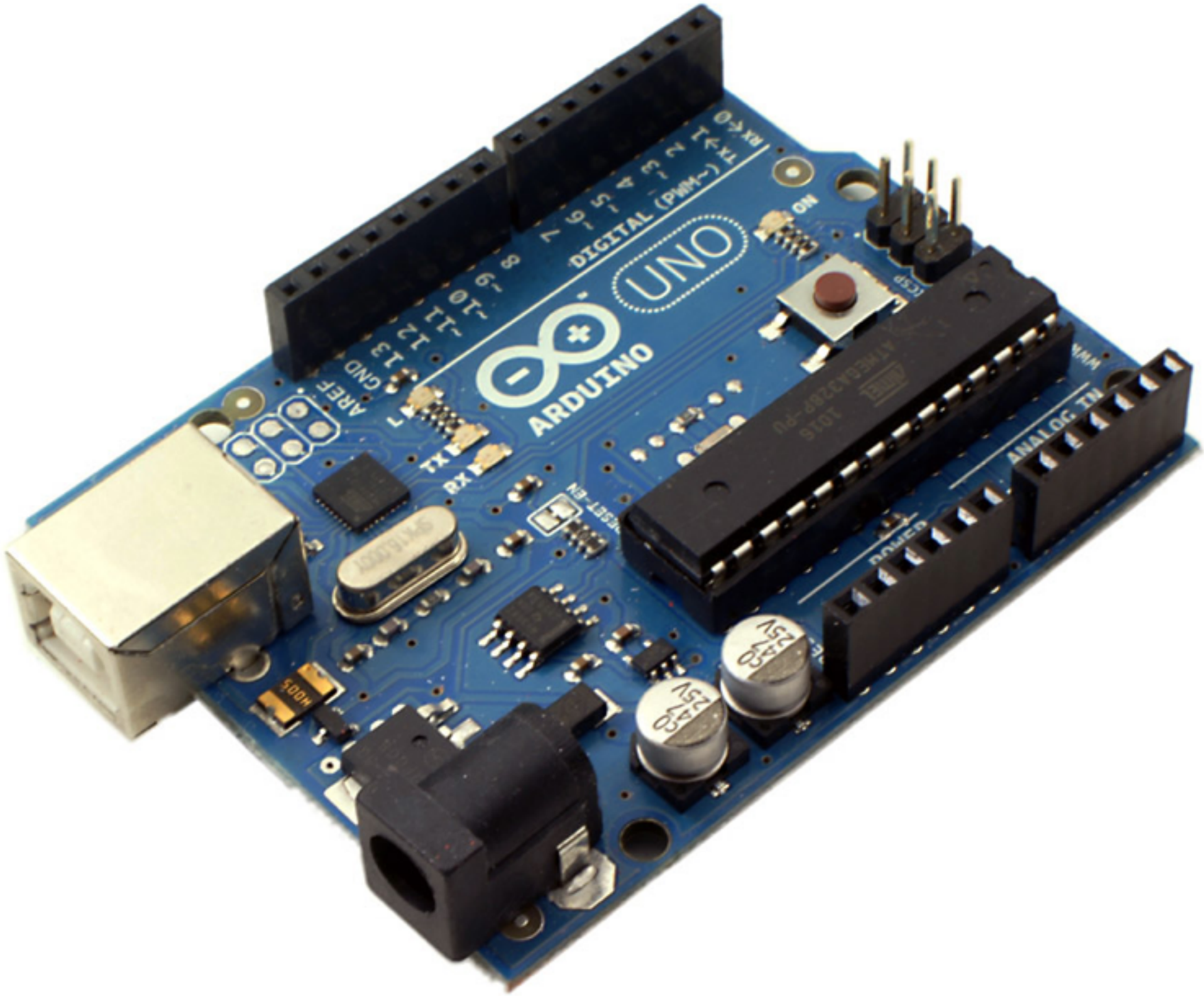
embedded system: a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints


OK
PWR
FDX
LNK
10M
ultra II
2.0GB


nest
75
IN 20 MIN
72

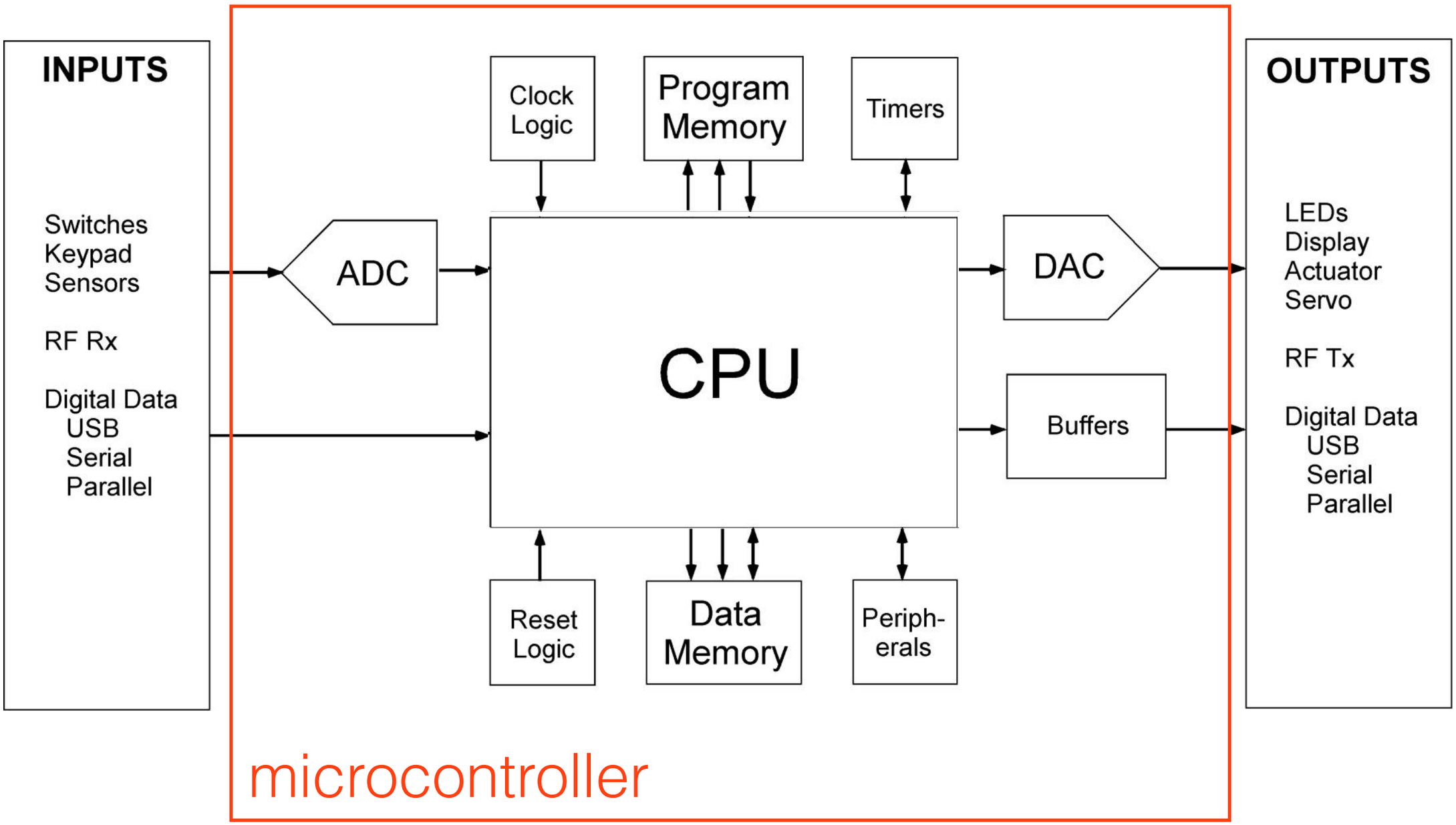

nest
75
IN 20 MIN
72


BELKIN

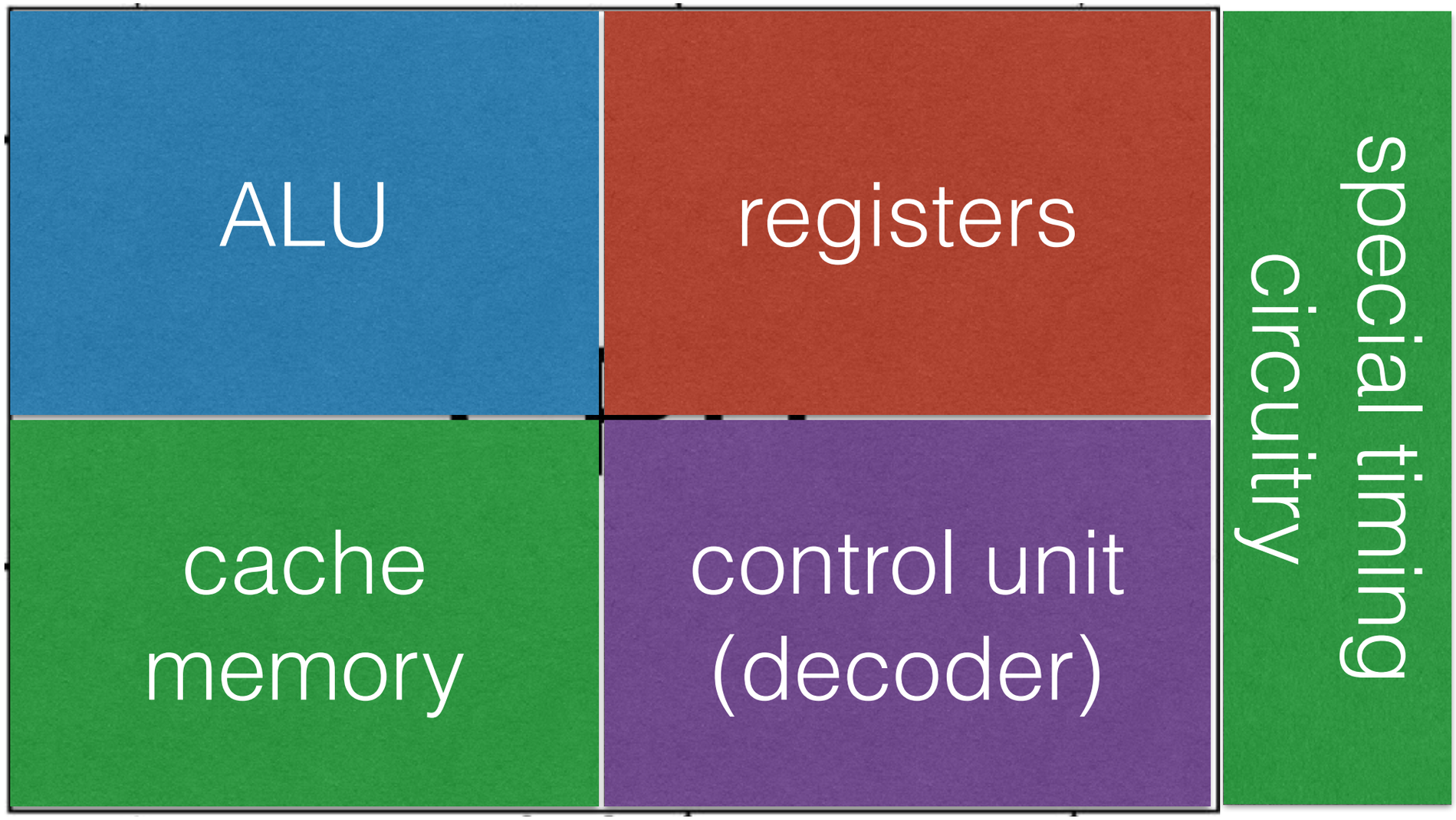

DIGITAL (PWM~)
UNO
ARDUINO
ON
POWER
ANALOG IN
TX
RX
AREF
GND


No SIM
9:11 PM
Messages
Tuesday
24
Calendar
Photos
Camera
Weather
Clock
Maps
Videos
Notes
Reminders
Stocks
142
Game Center
Newsstand
iTunes Store
App Store
Passbook
Compass
Settings
Phone
368
Mail
Safari
Music
iMore

embedded systems



microcontroller



MCU

microcontroller

- Atmega328
  - 16MHz
  - 16 bit ALU
  - 2KB SRAM
  - 32KB flash
  - 1KB EEPROM
  - 14 GPIO (6 PWM)
  - 6 analog inputs
  - hardware serial, TTL
  - hardware SPI

excellent tutorials:
http://www.adafruit.com/category/17

arduino sketch

- write in c
- use arduino functions

Blink

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

called on startup or reset

run forever

arduino serial comm

- really, really simple
- the Rx and Tx are on pins 0 and 1

```
Serial.begin(9600);

float resistance;

Serial.print(resistance);
Serial.print(" kohms\n");
```

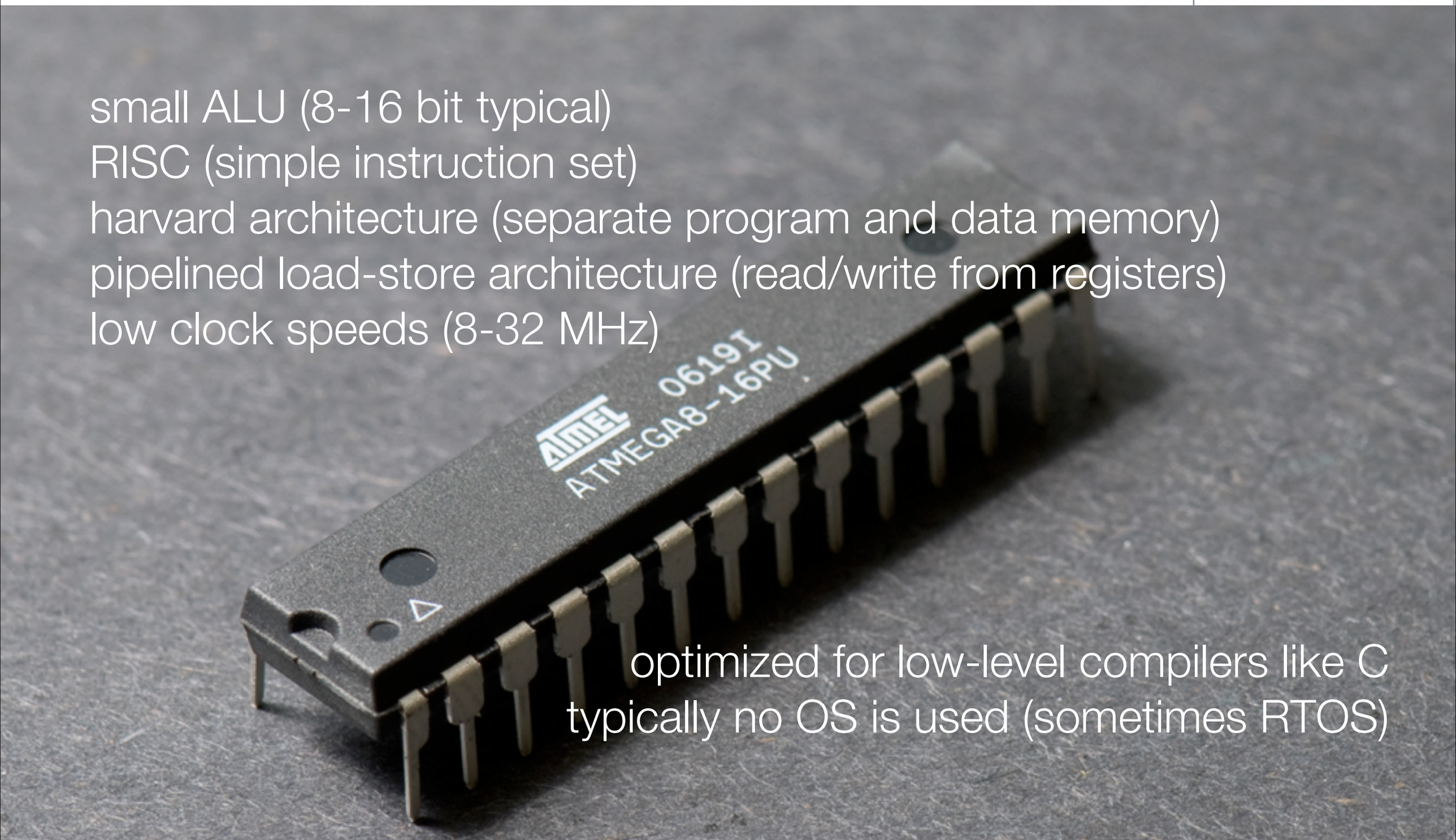
baud rate
9600 pulses/sec

send a float!

send a string!

atmega CPU

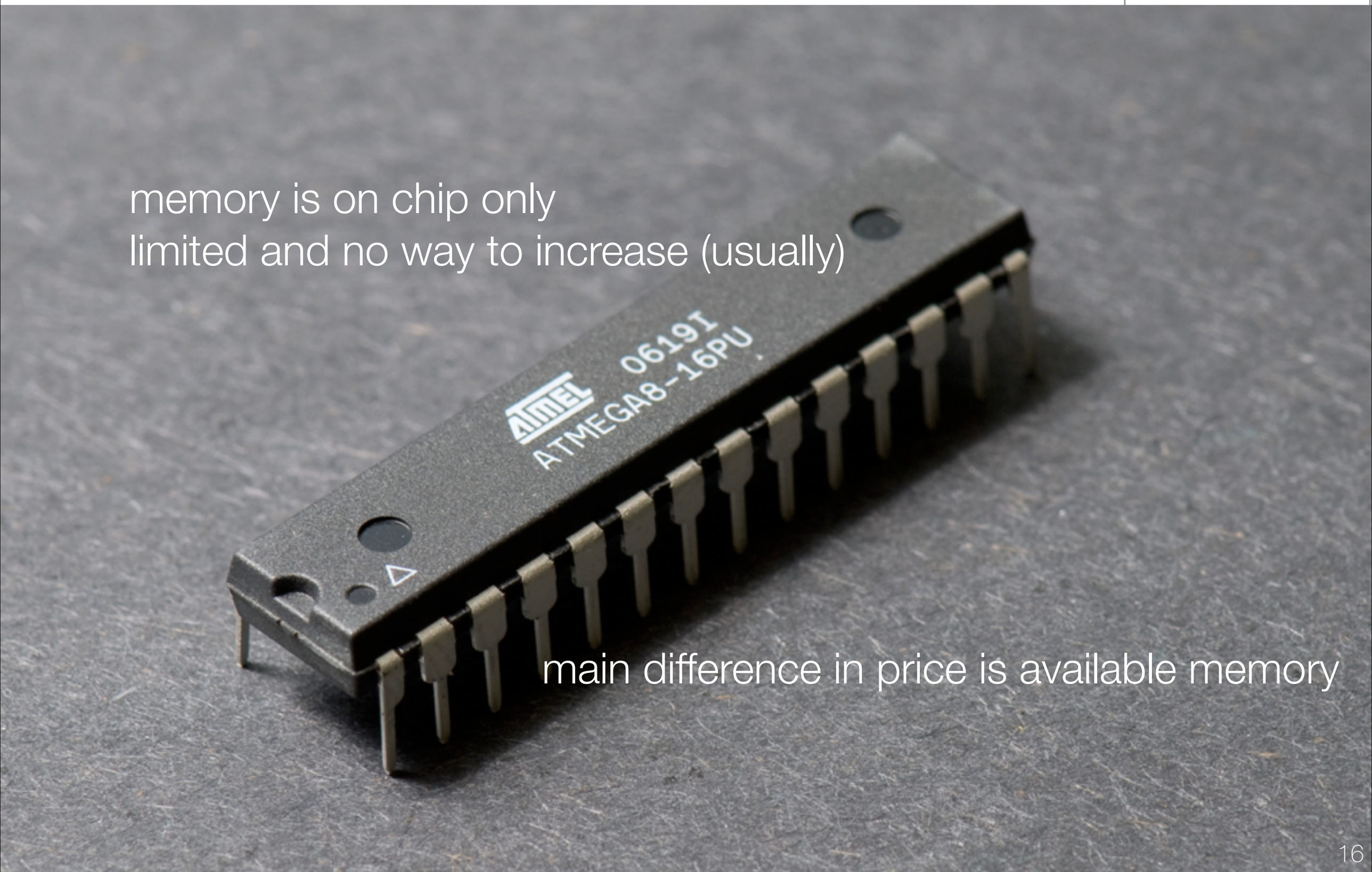
small ALU (8-16 bit typical)
RISC (simple instruction set)
harvard architecture (separate program and data memory)
pipelined load-store architecture (read/write from registers)
low clock speeds (8-32 MHz)



optimized for low-level compilers like C
typically no OS is used (sometimes RTOS)

atmega memory

memory is on chip only
limited and no way to increase (usually)



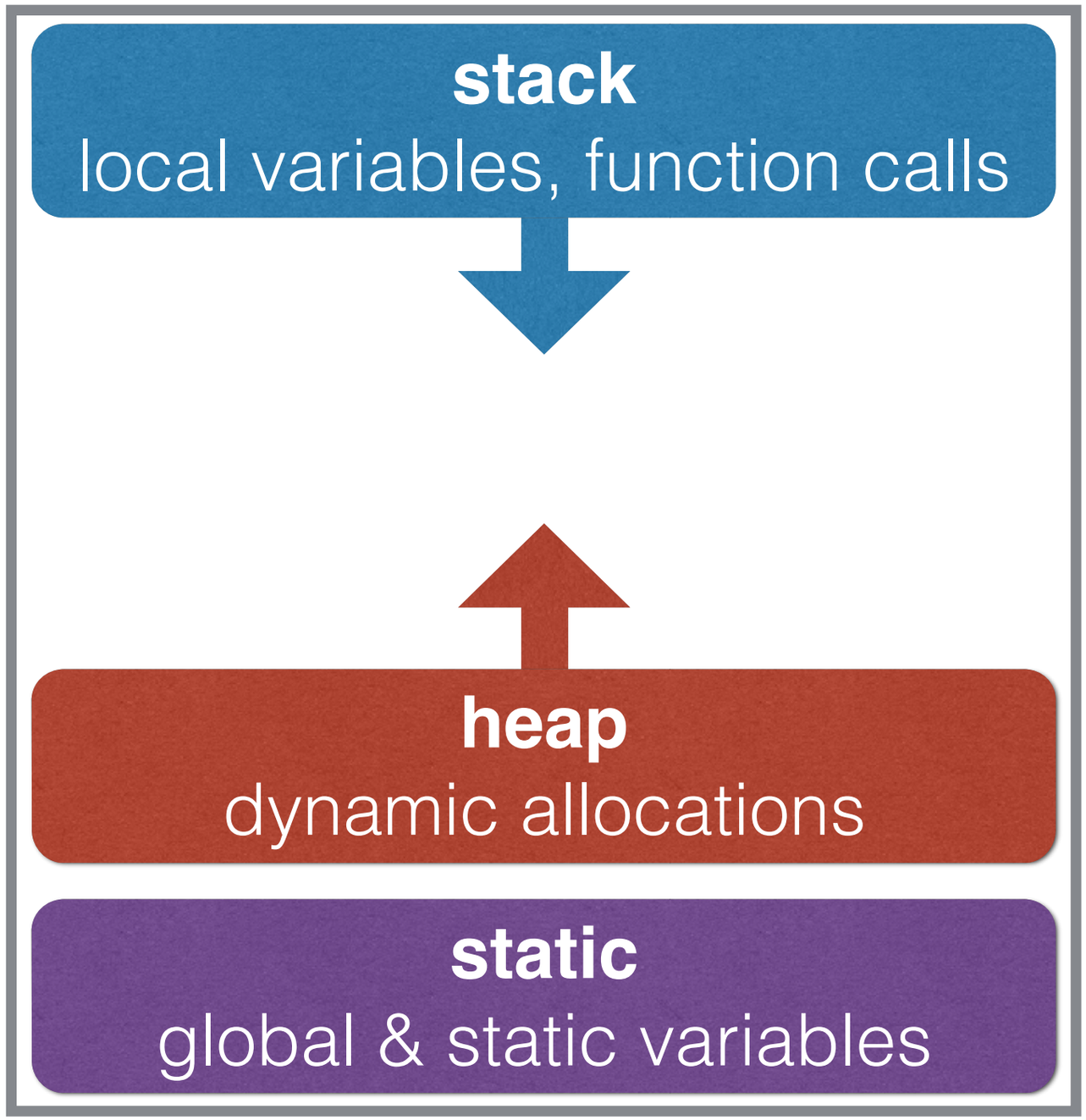
main difference in price is available memory

atmega memory

- three memory spaces
- volatile
  - SRAM
- non-volatile
  - EEPROM
- flash (program memory)

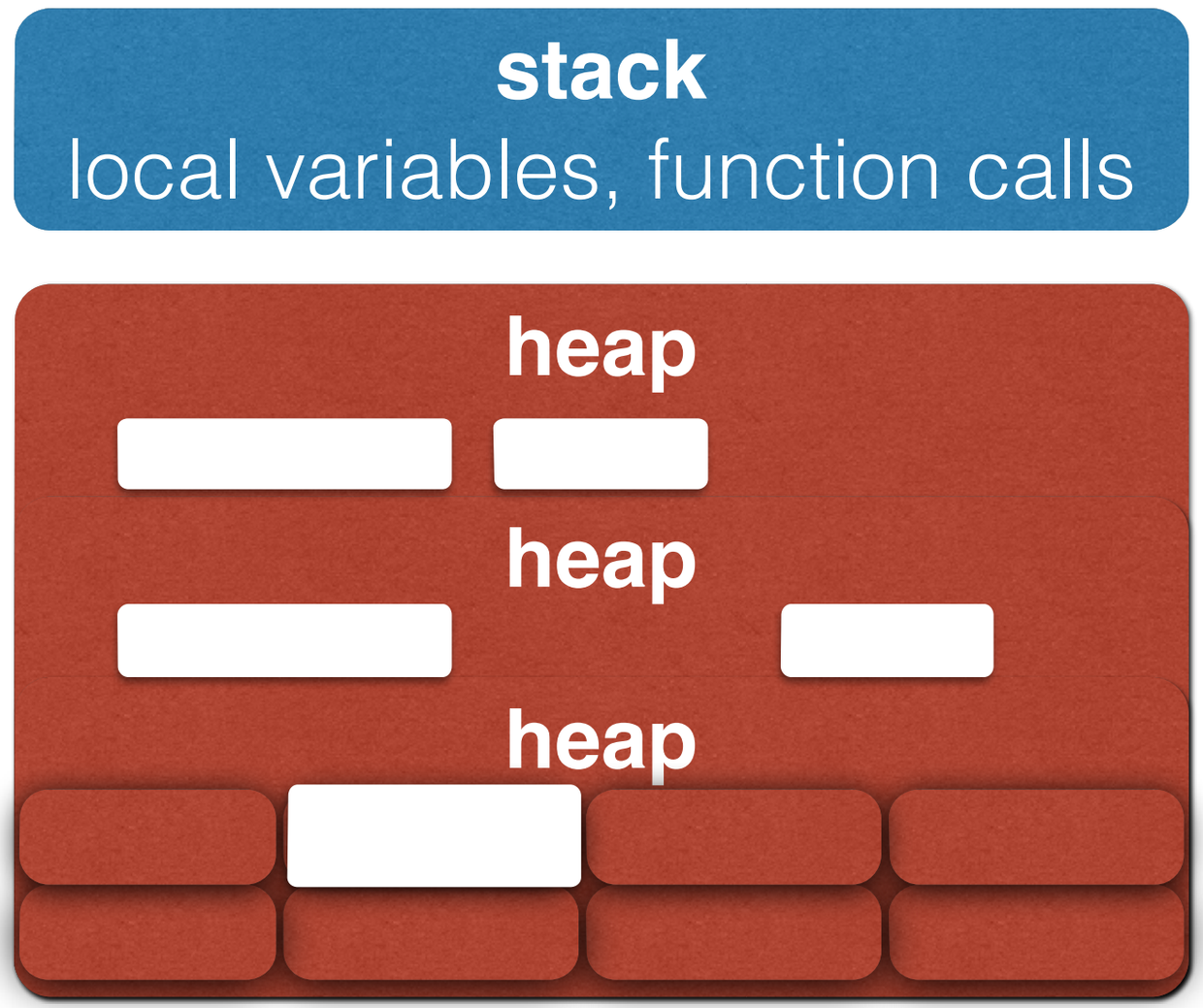
atmega SRAM

- SRAM divided into three areas



easy to do

- fragmented memory in the heap
  - it not an OS, garbage collection is VERY primitive



atmega SRAM

- guidelines to avoid fragmentation
  -
  -
  -
  -

```
int tmpSensorData[512];
int *moreSensorData;

// the setup routine runs once when you press reset:
void setup() {
      moreSensorData = (int*)malloc(512);
}

// the loop routine runs over and over again forever:
void loop() {
  int f = analogRead(1);

}
```

atmega non-volatile

- EEPROM
  - electronically erasable programmable memory
  - stays in memory when power turns off
  - good for saving a “state”
  - infinite number of reads
  - ~100,000 writes / erases
  - one write takes ~3.3ms, per byte

atmega non-volatile

eeprom_write §

```
 * turned off and may be retrieved later by another sketch.
 */

#include <EEPROM.h>
int addr = 0;
void setup()
{
}
void loop()
{
  // value from 0 to 255.
  int val = analogRead(0) / 4;

  // write the value to the appropriate byte of the EEPROM.
  EEPROM.write(addr, val);

  // advance to the next address.  there are 512 bytes in
  // the EEPROM, so go back to 0 when we hit 512.
  addr = addr + 1;
  if (addr == 512)
    addr = 0;

  delay(100);

  byte value = EEPROM.read(addr);
}
```

write a byte to “address”, 0-511

read a byte from “address”

atmega non-volatile

- program memory (flash)
  - where your code instructions are saved
  - flash memory
  - great for variables that do not change
    - like strings!
  - not stored in the SRAM until copied over

```
#include <avr/pgmspace.h>


  // save some unsigned ints
  PROGMEM  prog_uint16_t charSet[]  = { 8400, 77, 42};

  // save some chars
  PROGMEM prog_uchar signMessage[]  = {"PROGRAM MEMORY STRING"};

  unsigned int displayInt;
  int k;    // counter variable
  char myChar;

  // read back a 2-byte int
   displayInt = pgm_read_word_near(charSet + k)

  // read back a char
  myChar =  pgm_read_byte_near(signMessage + k);

  for (int i = 0; i < 6; i++)
  {
    strcpy_P(buffer, (char*)pgm_read_word(&(string_table[i]))); // copy
    Serial.println( buffer );
    delay( 500 );
  }
```

http://arduino.cc/en/Reference/PROGMEM#.UyKZIdztJg0

atmega osc & reset

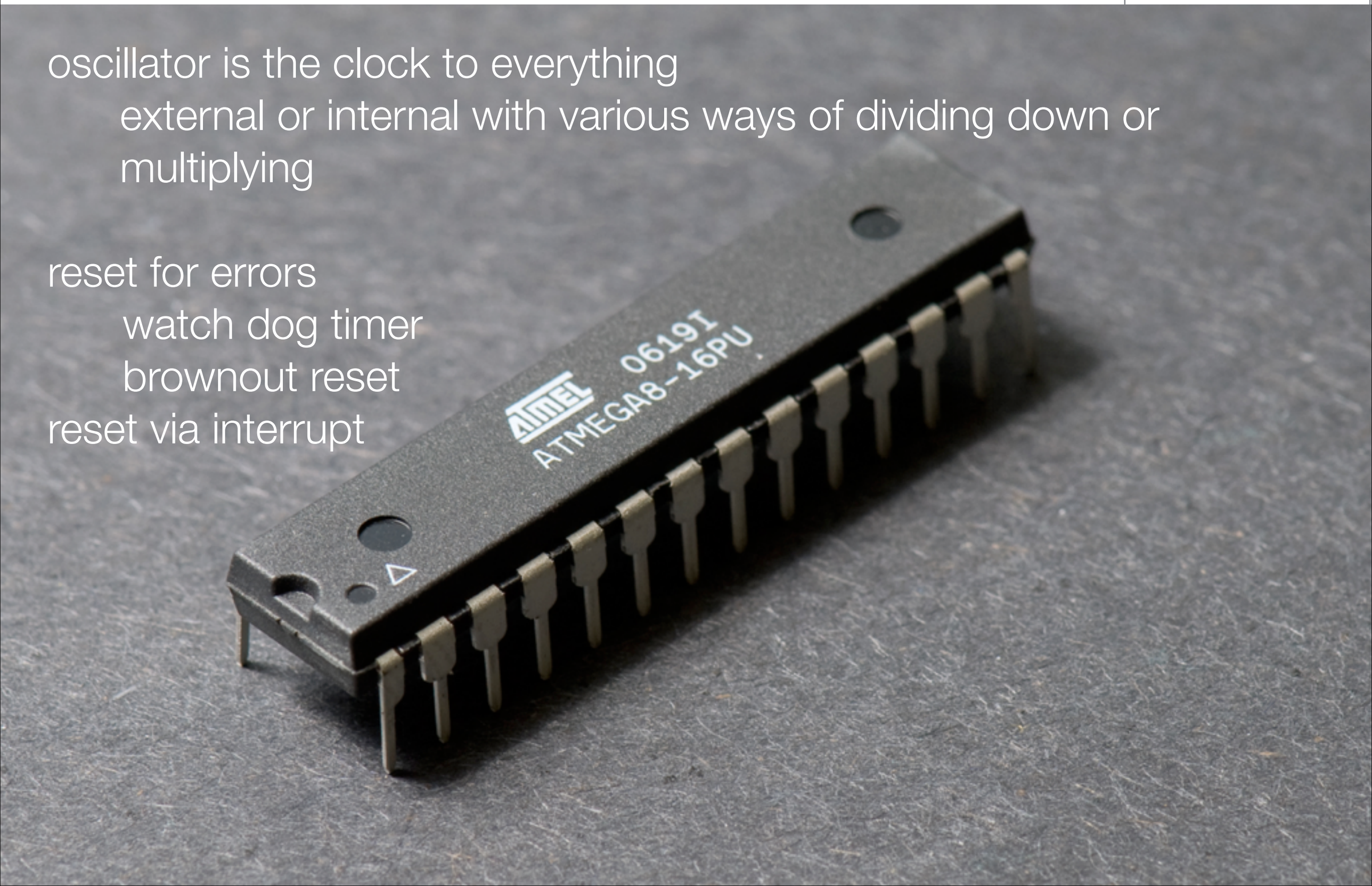
oscillator is the clock to everything

- external or internal with various ways of dividing down or multiplying

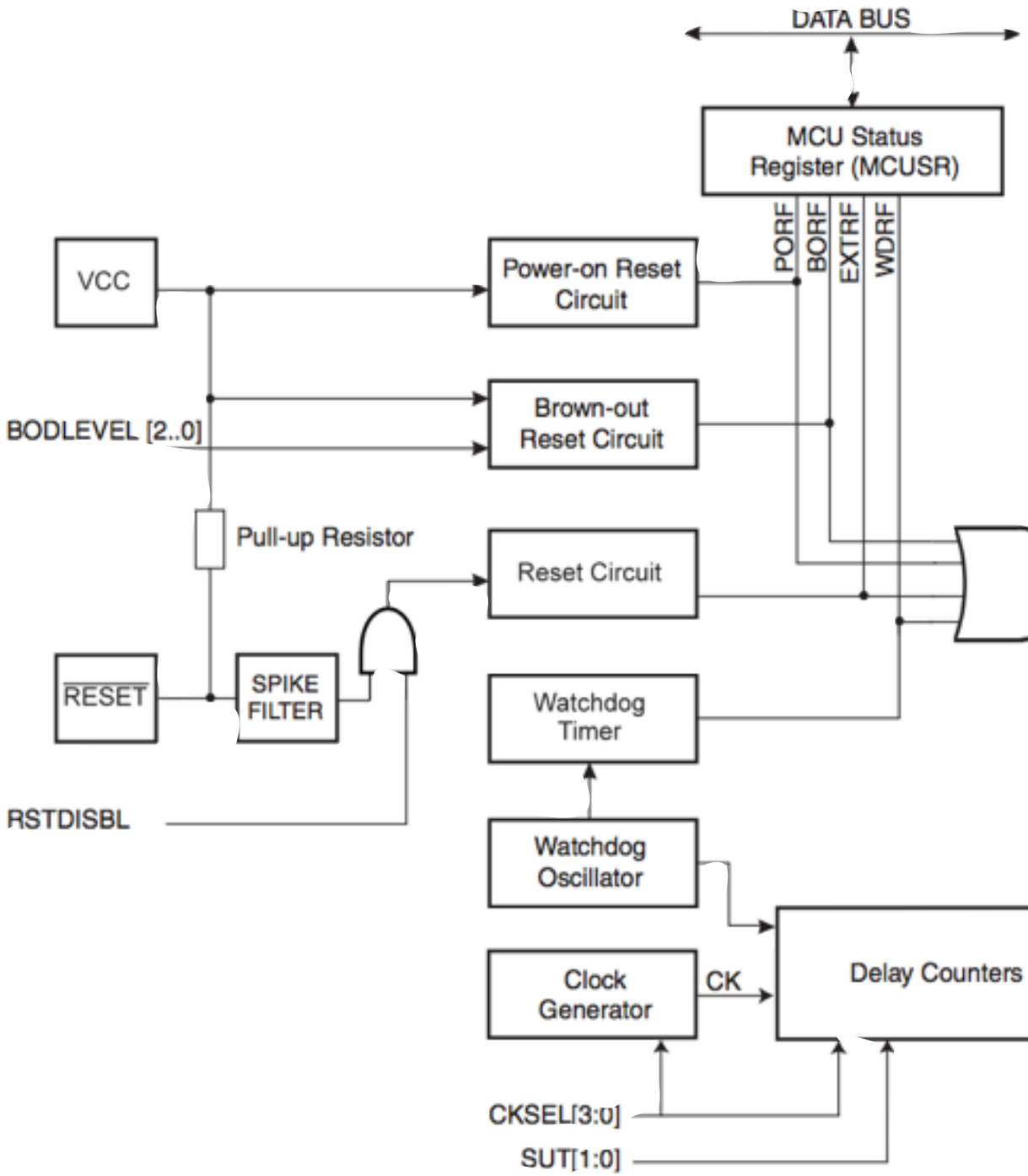
reset for errors

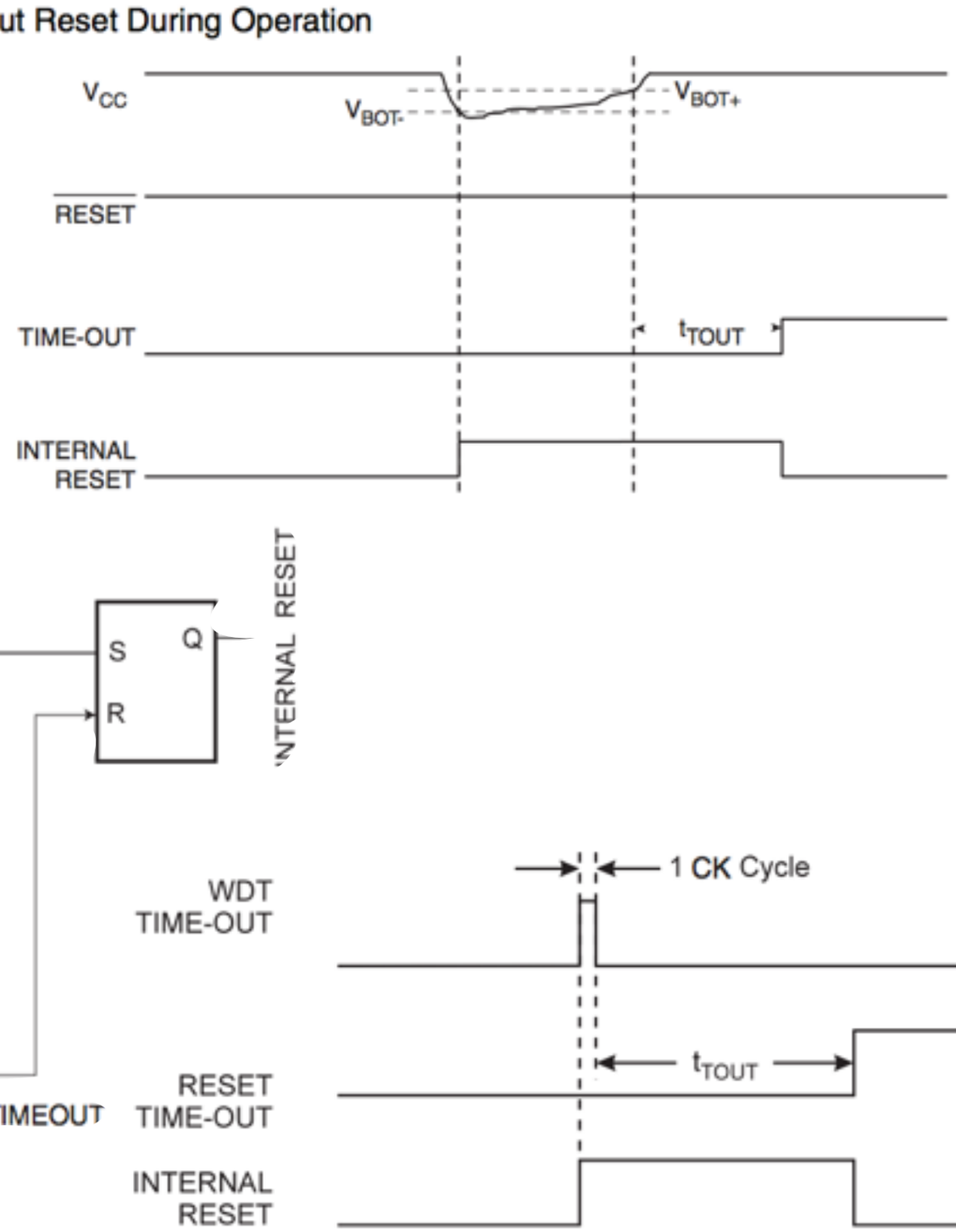
- watch dog timer
- brownout reset

reset via interrupt



atmega reset





watch dog timer

- enable or disable

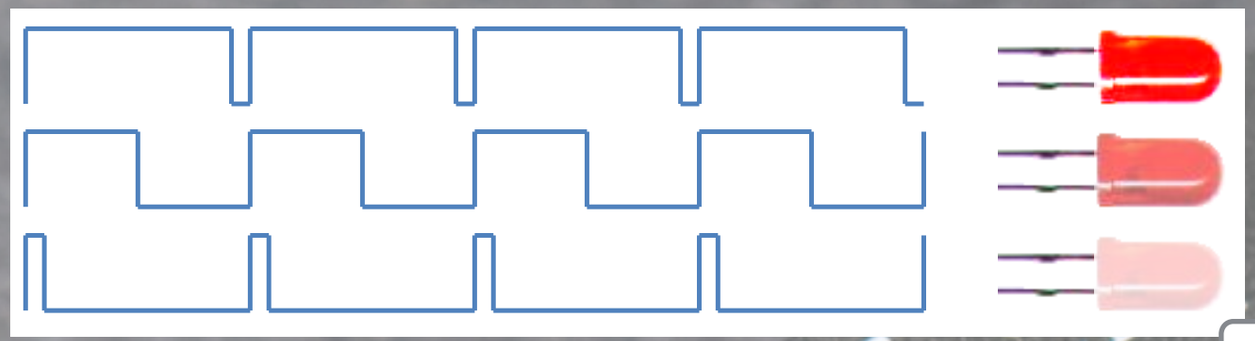
```
if(USE_WDT==1)
  wdt_enable(WDTO_8S); // watch dog set at 8 seconds, don't go much lower
else
  wdt_disable();
```

- must set back to zero periodically or reset occurs

```
void SW_ISR() {
  // if you do not interrupt the WDT, then the board will reset
  wdt_reset();
}
```

timers

WDT, real-time clock,
pulse width modulation: square wave output



bright

dim

controls duty cycle

clock

register value

comparator

output enable

divider

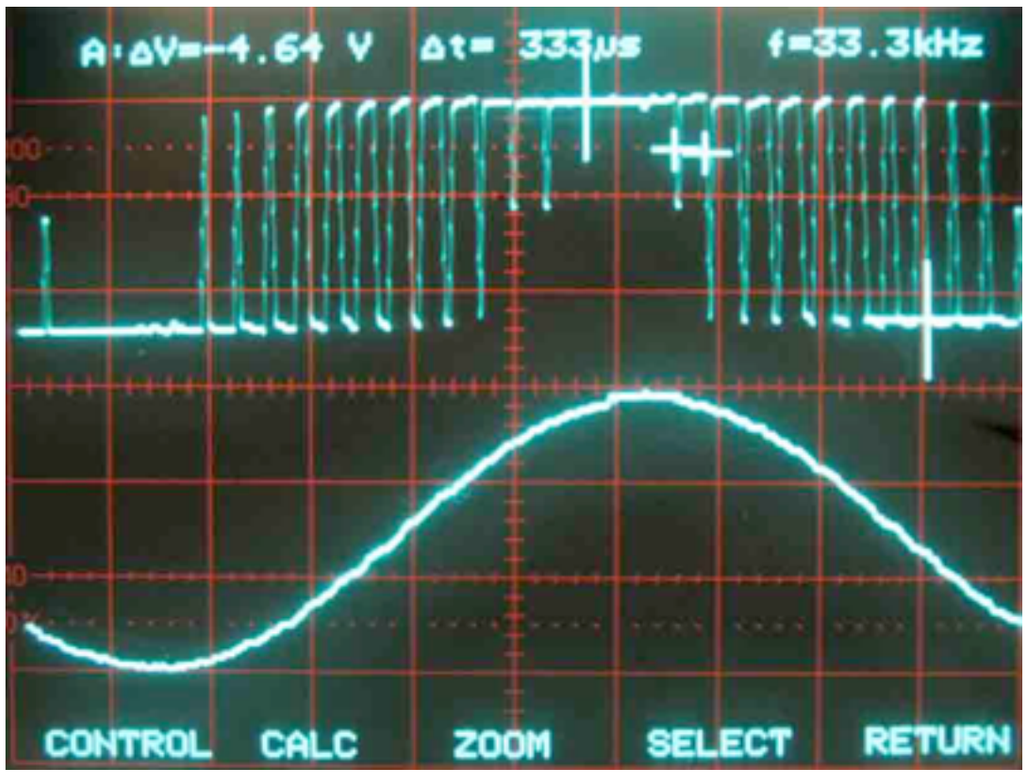
counter

also controls frequency of wave

max value controls frequency of wave

PWM

- servo control and motor control
  - width of pulse matters
- digital to analog conversion
  - smooth a square wave
- LED brightness control
  - “duty cycle” matters




```
ledBrightness += delta;     // change brightness setting
analogWrite(LED_PWM, ledBrightness); // set new brightness
```

pin to write to
(must be PWM)

value to write (0-255)

PWM

change in duty cycle

15% 35% 75% 95% 100%

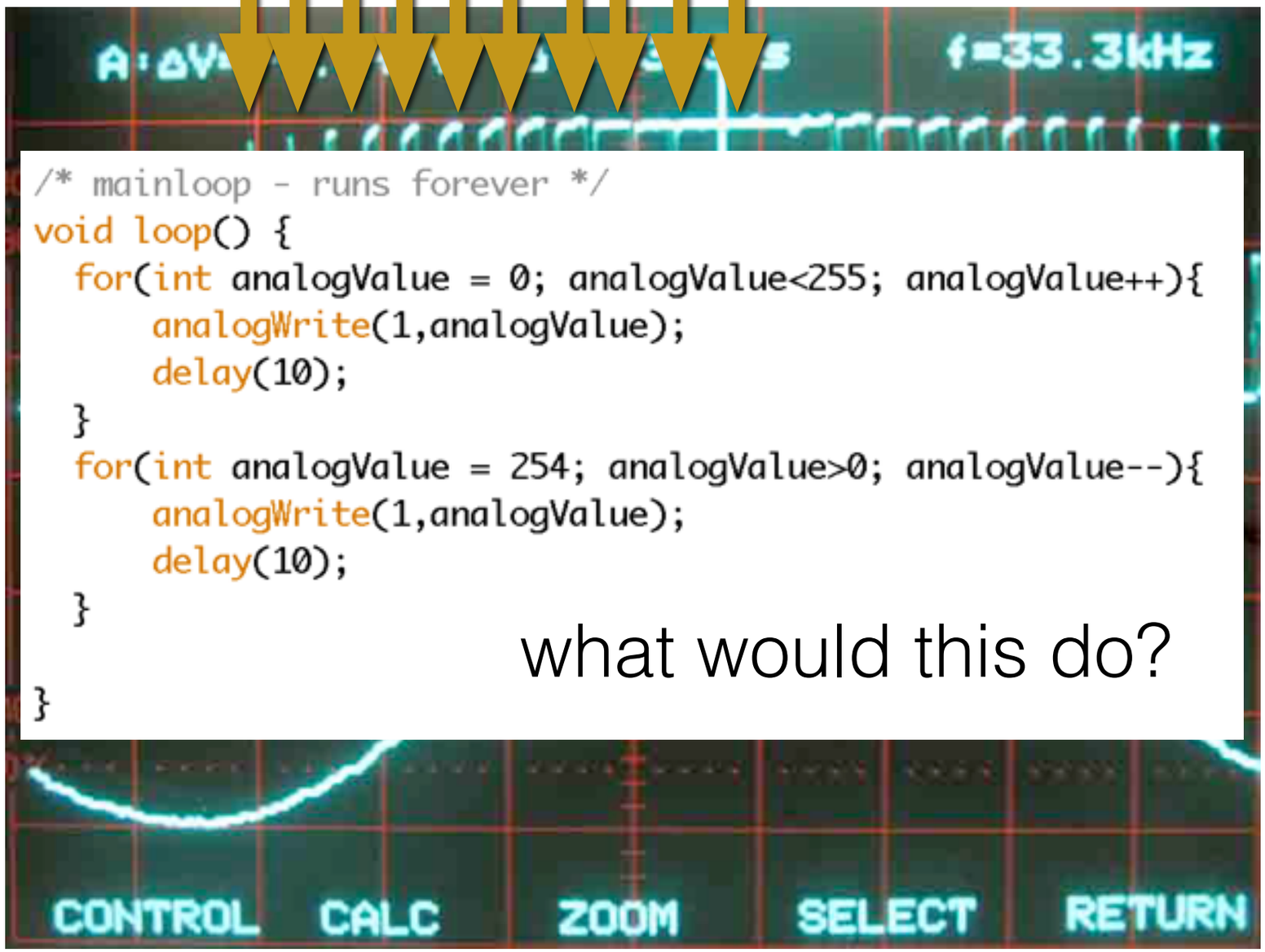
10% 25% 50% 85% 99%

PWM

after low pass filter

```
/* mainloop - runs forever */
void loop() {
  for(int analogValue = 0; analogValue<255; analogValue++){
      analogWrite(1,analogValue);
      delay(10);
  }
  for(int analogValue = 254; analogValue>0; analogValue--){
      analogWrite(1,analogValue);
      delay(10);
  }
}
```

what would this do?



normal PWM: constant duty cycle

this example: time varying duty cycle

PWM with registers

- registers control most peripherals on a uC

```
pinMode(3, OUTPUT);
pinMode(11, OUTPUT);
TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
TCCR2B = _BV(CS22);
OCR2A = 200;
OCR2B = 120;
```

in this register

set bit with this name

TCCR2A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR2B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	TCCR2B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

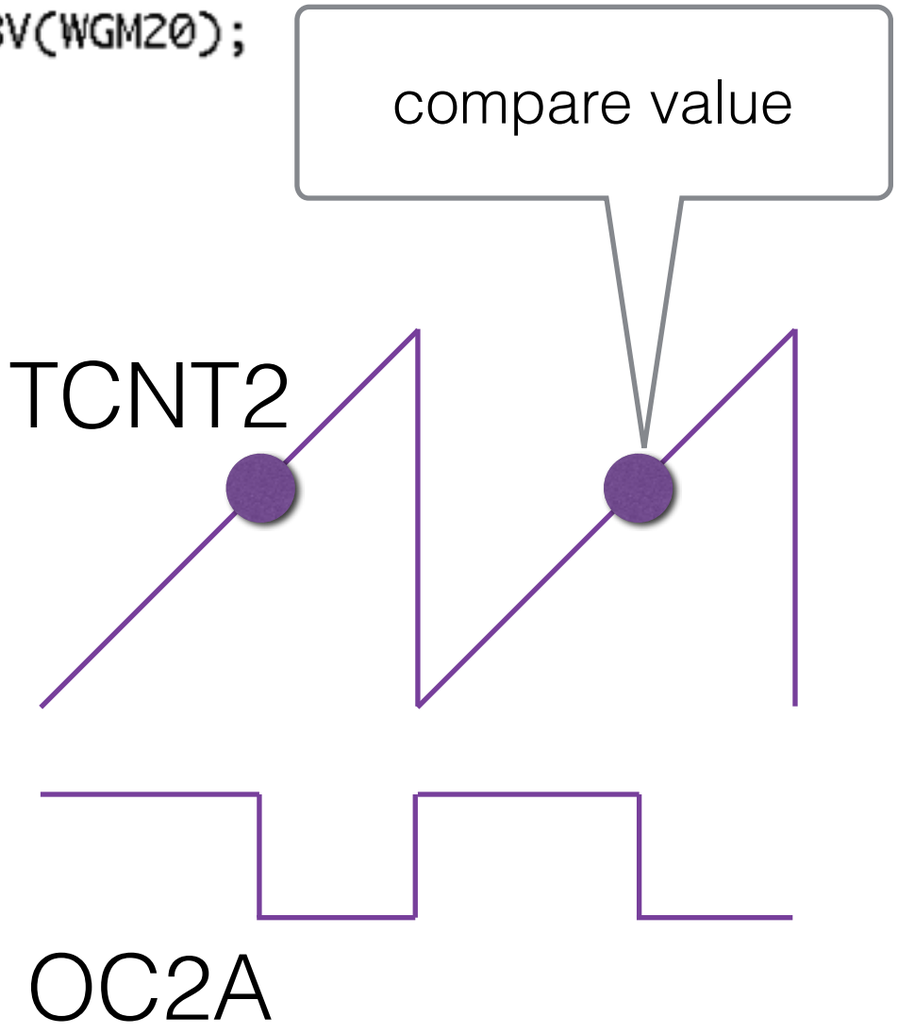
PWM with registers

- registers control most peripherals on a uC

```
pinMode(3, OUTPUT);
pinMode(11, OUTPUT);
TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
TCCR2B = _BV(CS22);
OCR2A = 200;
OCR2B = 120;
```

Table 18-4. Compare Output Mode, Phase Correct PWM Mode[(1)]

COM2A1	COM2A0	Description
0	1	WGM22 = 0: Normal Port Operation, OC2A Disconnected. WGM22 = 1: Toggle OC2A on Compare Match.
1	0	Clear OC2A on Compare Match when up-counting. Set OC2A on Compare Match when down-counting.
1	1	Set OC2A on Compare Match when up-counting. Clear OC2A on Compare Match when down-counting.



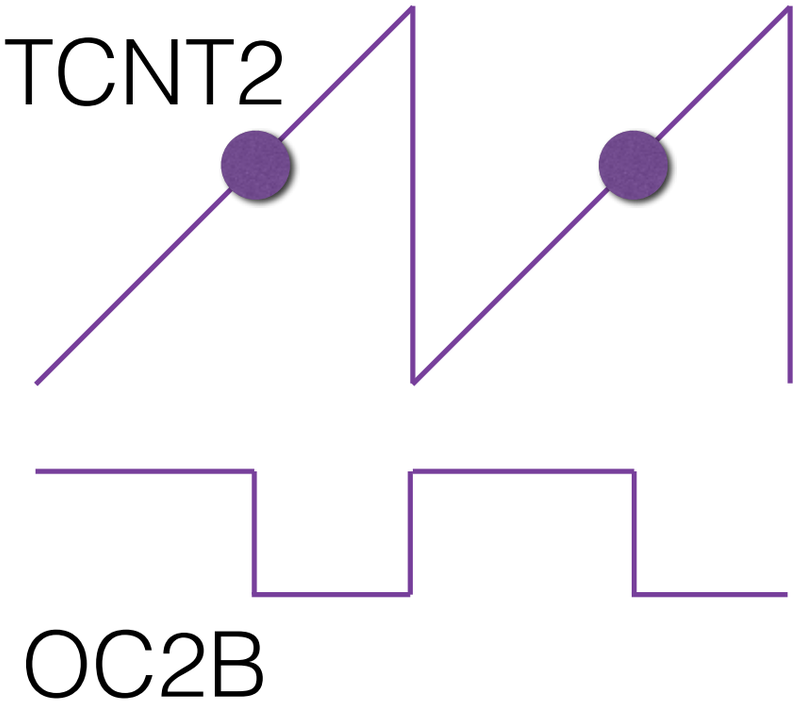
PWM with registers

- registers control most peripherals on a uC

```
pinMode(3, OUTPUT);
pinMode(11, OUTPUT);
TCCR2A = _BV(COM2A1)  _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
TCCR2B = _BV(CS22);
OCR2A = 200;
OCR2B = 120;
```

Table 18-6. Compare Output Mode, Fast PWM Mode(1)

COM2B1	COM2B0	Description
0	0	Normal port operation, OC2B disconnected.
0	1	Reserved
1	0	Clear OC2B on Compare Match, set OC2B at BOTTOM, (non-inverting mode).
1	1	Set OC2B on Compare Match, clear OC2B at BOTTOM, (inverting mode).



PWM with registers

- registers control most peripherals on a uC

```
pinMode(3, OUTPUT);
pinMode(11, OUTPUT);
TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
TCCR2B = _BV(CS22);
OCR2A = 200;
OCR2B = 120;
```

Table 18-8. Waveform Generation Mode Bit Description

Mode	WGM22	WGM21	WGM20	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

PWM with registers

- registers control most peripherals on a uC

```
pinMode(3, OUTPUT);
pinMode(11, OUTPUT);
TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
TCCR2B = _BV(CS22);
OCR2A = 200;
OCR2B = 120;
```

Table 18-9. Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)

two outputs at 16MHz / 64 / 256 = 976.5625Hz

PWM with registers

- registers control most peripherals on a uC

```
pinMode(3, OUTPUT);
pinMode(11, OUTPUT);
TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
TCCR2B = _BV(CS22);
OCR2A = 200;
OCR2B = 120;
```

OCR2A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
(0xB3)	OCR2A[7:0]								OCR2A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT2). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC2A pin.

OCR2B – Output Compare Register B

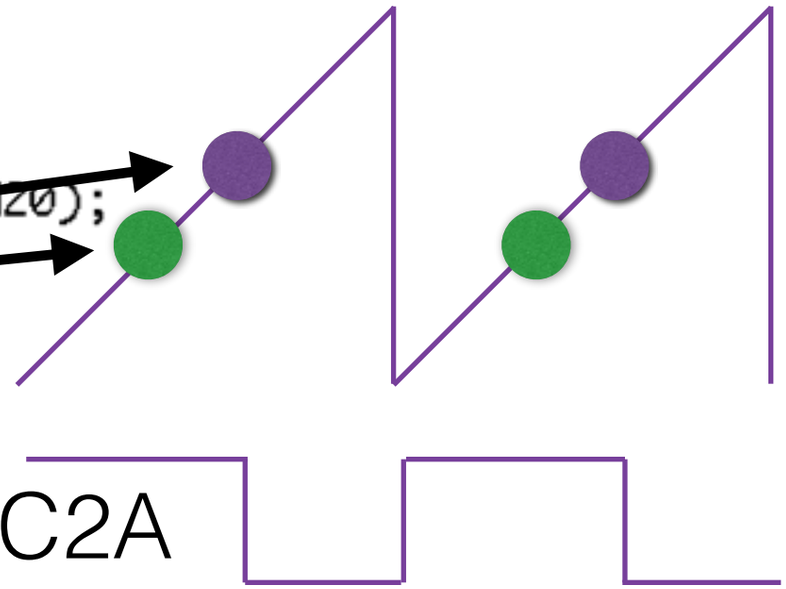
Bit	7	6	5	4	3	2	1	0	
(0xB4)	OCR2B[7:0]								OCR2B
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Output Compare Register B contains an 8-bit value that is continuously compared with the counter value (TCNT2). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC2B pin.

PWM with registers

- registers control most peripherals on a uC

```
pinMode(3, OUTPUT);
pinMode(11, OUTPUT);
TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
TCCR2B = _BV(CS22);
OCR2A = 200;
OCR2B = 120;
```



duty cycle of OC2A (pin 3) = 200/256 = 78%

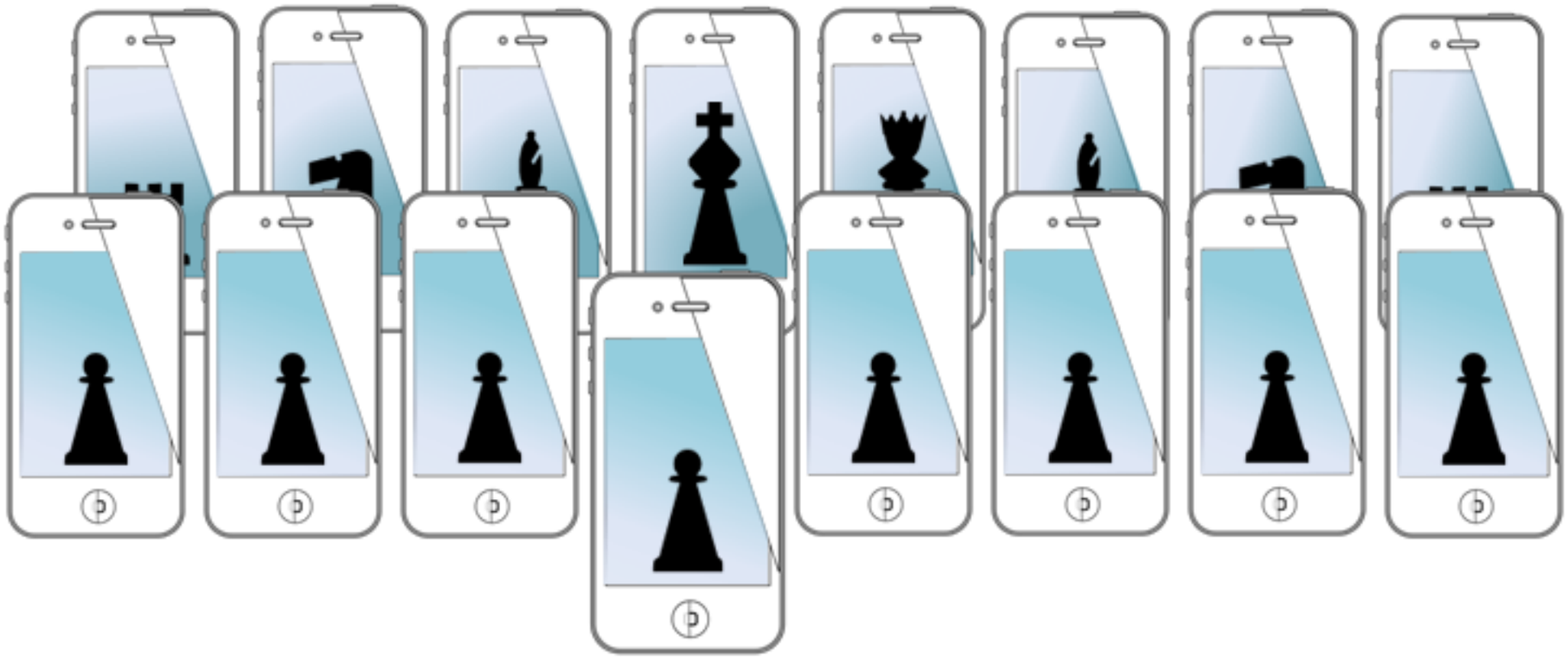
duty cycle of OC2B (pin 11) = 120/256 = 47%

two outputs at 16MHz / 64 / 256 = 976.5625Hz

for next time…

- sensors and more control
- arduino shields
- blue tooth communication

MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture fourteen: microcontroller basics

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University