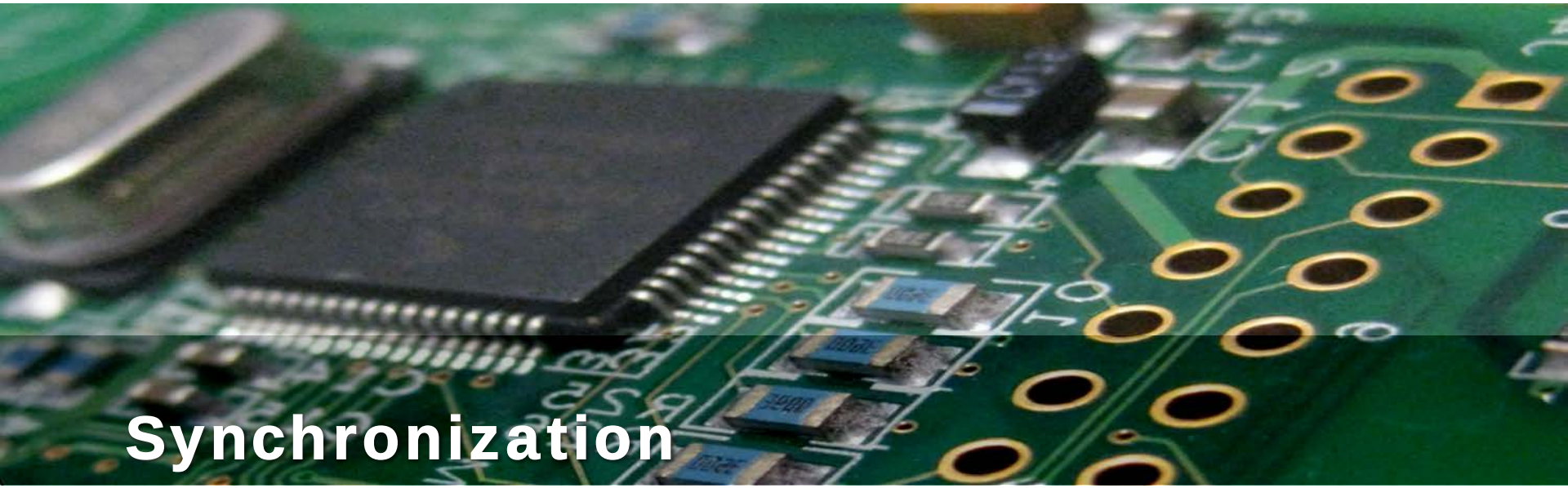
Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur



Synchronization

“We will have so many things in our system. How to tie them together?”

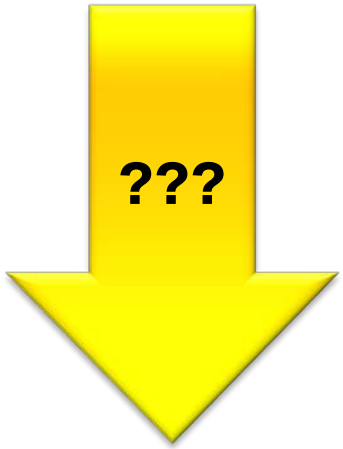
Prof. Erich Styger
erich.styger@hslu.ch
+41 41 349 33 01

Scriptum:
Synchronization

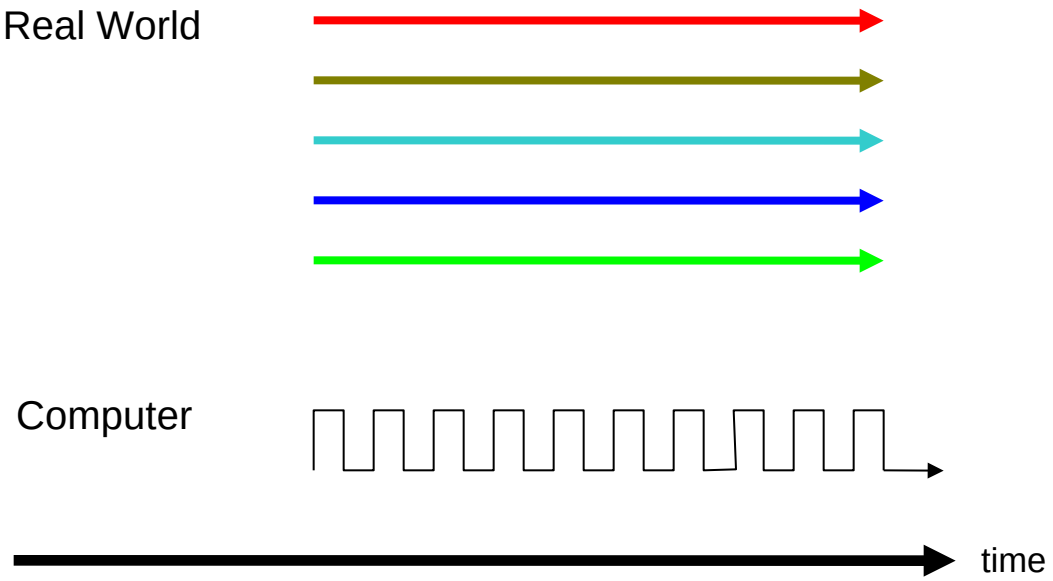
Learning Goals

- Problem: Why synchronization?

- Different kinds of synchronization
  - Real Time
  - Polling/Gadfly
  - Interrupts
- Interrupts & execution speed
- Data and interrupts
- Reentrancy
- Priorities
- Implementation



Comparison Computer vs. Real World



- Real World
  - concurrency
  - continuous
- Computer World
  - sequential
  - discrete

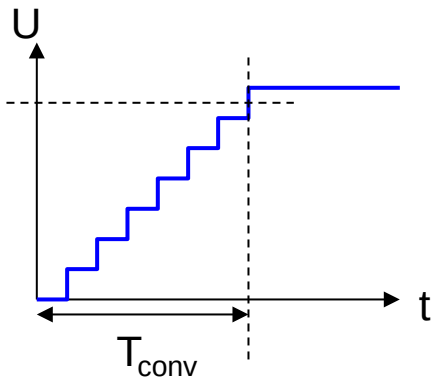
Real Time Systems: Need for synchronization

Synchronization

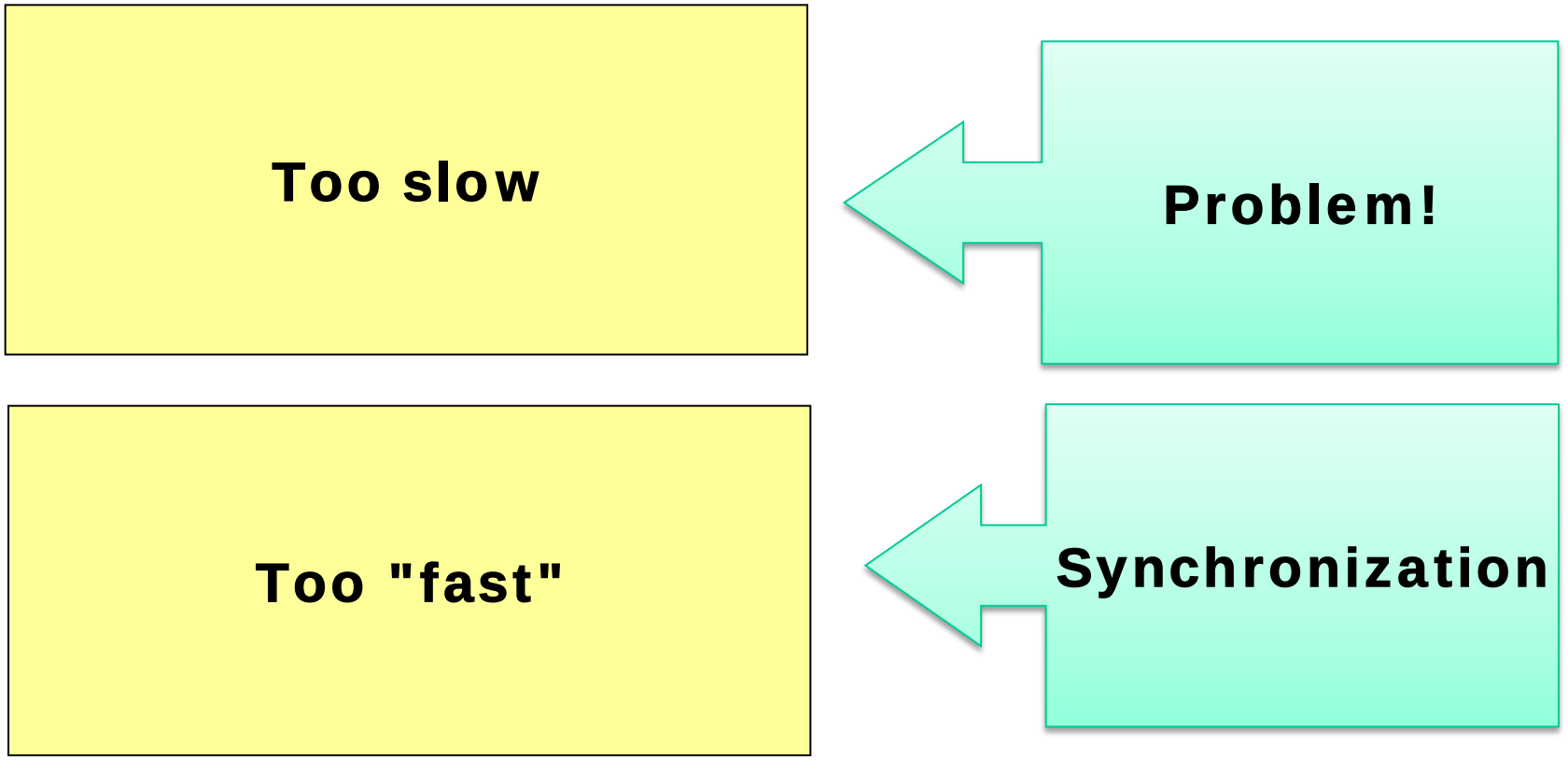
- Connection point between processes
- Computer has to attach to the process
- Computer operates in different time scale
  - If slower than reality
    - Result too late: incorrect
  - If faster than reality
    - Result too early: incorrect

The *correct* result at the *right* time

- Computer has to synchronize with (real world time-) process
- Examples
  - A/D converter
  - keyboard

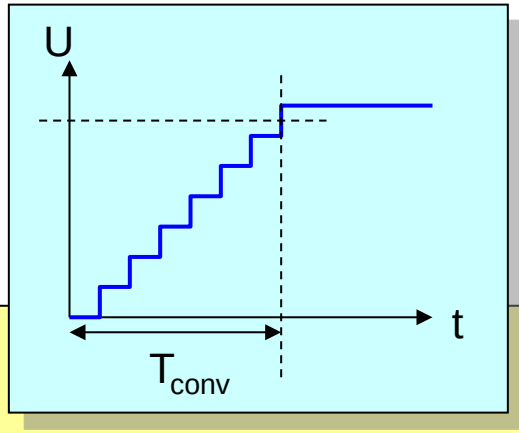


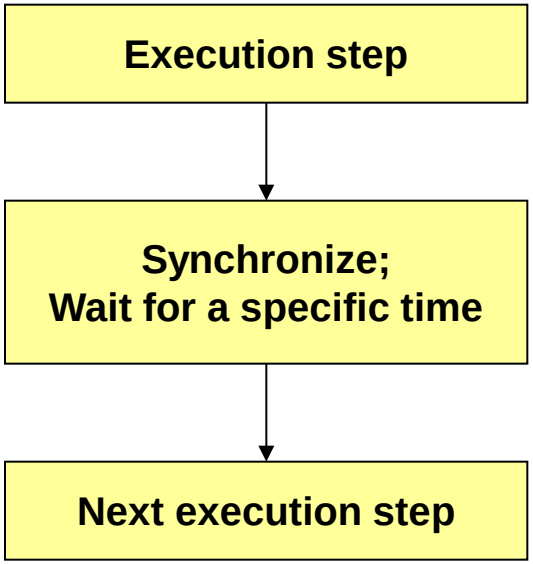
Computation Speed



A) Realtime Synchronization

Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur






```
ADCCONF = 0x37; /* setup A/D-Converter */
ADCCRL |= 0x80; /* start conversation */
for (i=0; i<2000; i++); /* wait for some time… */
Value = ADCDATA; /* read value */
```

- What is the needed waiting time?
- Inefficient
- Different Compiler (version)?
- Different computer, portability?
- Different (dynamic) clock rate?

Delay Loop (1)

```
void delay(void) {
  unsigned char i;

  for(i=0;i<100;i++);
}
```

ARM Cortex-M4:

```
  sub  sp, sp, #12
  add  r7, sp, #0
  movs  r3, #0
  strb  r3, [r7, #7]
  b .L2
.L3:
  ldrb  r3, [r7, #7]@zero_ext2
  adds  r3, r3, #1
  strb  r3, [r7, #7]
.L2:
  .loc 1 107 0 discriminator 1
  ldrb  r3, [r7, #7]@zero_ext2
  cmp r3, #99
  bls .L3
```

Delay Loop (2)

```
void delay(void) {
  unsigned char i;

  for(i=0;i<100;i++);
}
```

ARM Cortex-M4 (-O3):

```
  bx  lr
```

Delay Loop (3)

```
void delay(void) {
  volatile unsigned char i;

  for(i=0;i<100;i++);
}
```

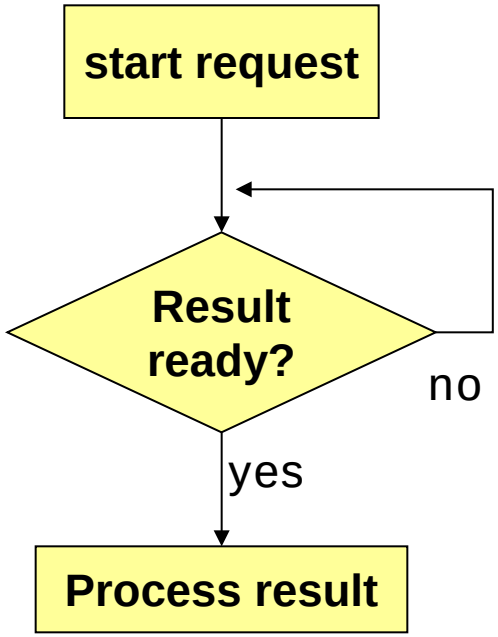
ARM Cortex-M4 (-O3):

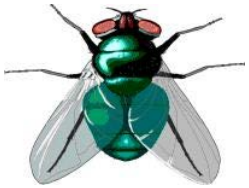
```
  sub  sp, sp, #8
  movs r3, #0
  strb r3, [sp, #7]
  ldrb r3, [sp, #7]@zero_extend
  cmp  r3, #99
  bhi  .L1
.L4:
  ldrb r3, [sp, #7]@zero_extend
  adds r3, r3, #1
  uxtb r3, r3
  strb r3, [sp, #7]
  ldrb r3, [sp, #7]@zero_extend
  cmp  r3, #99
  bls  .L4
.L1:
  add  sp, sp, #8
  bx   lr
```

Delay Loop (4)

```
__attribute__((naked, no_instrument_function))
 void Wait10Cycles(void)
{
 /* This function will wait 10 CPU cycles
  *(including call overhead).
  * Cortex-M0 and M4 have 1 cycle for a NOP */
__asm (
   /* bl Wai10Cycles() to here: [4] */
   "nop   \n\t" /* [1] */
   "nop   \n\t" /* [1] */
   "nop   \n\t" /* [1] */
   "bx lr \n\t" /* [3] */
  );
}
```

B) Gadfly Synchronization

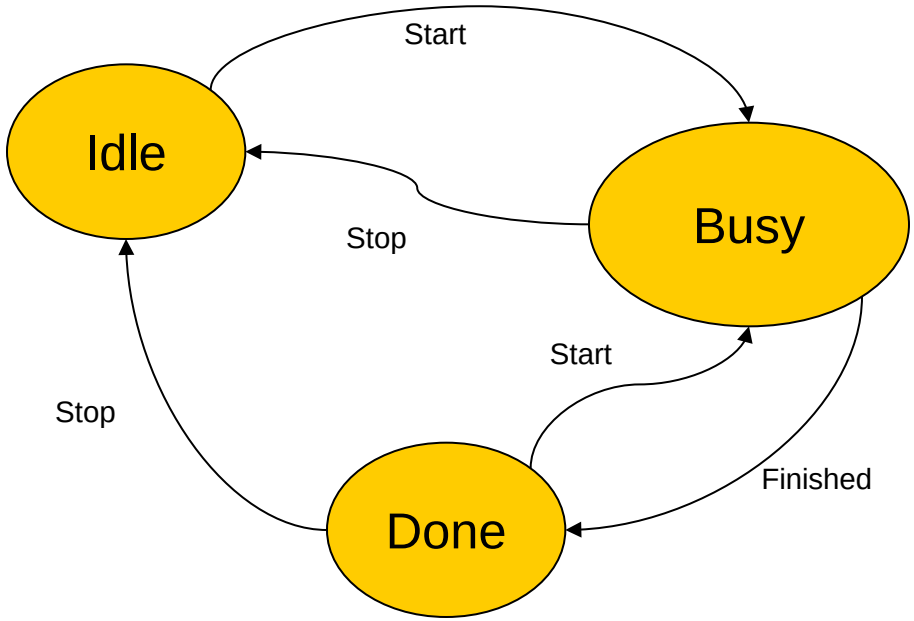




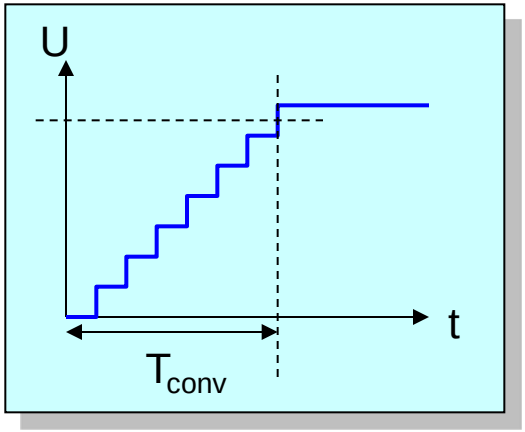
Source: Google Picture Library

- Active waiting/polling
- Processing power needed
- Blocks further execution

Gadfly Synchronization: Hardware support



- Examples
  - SCI
  - AD/DA



Using volatile

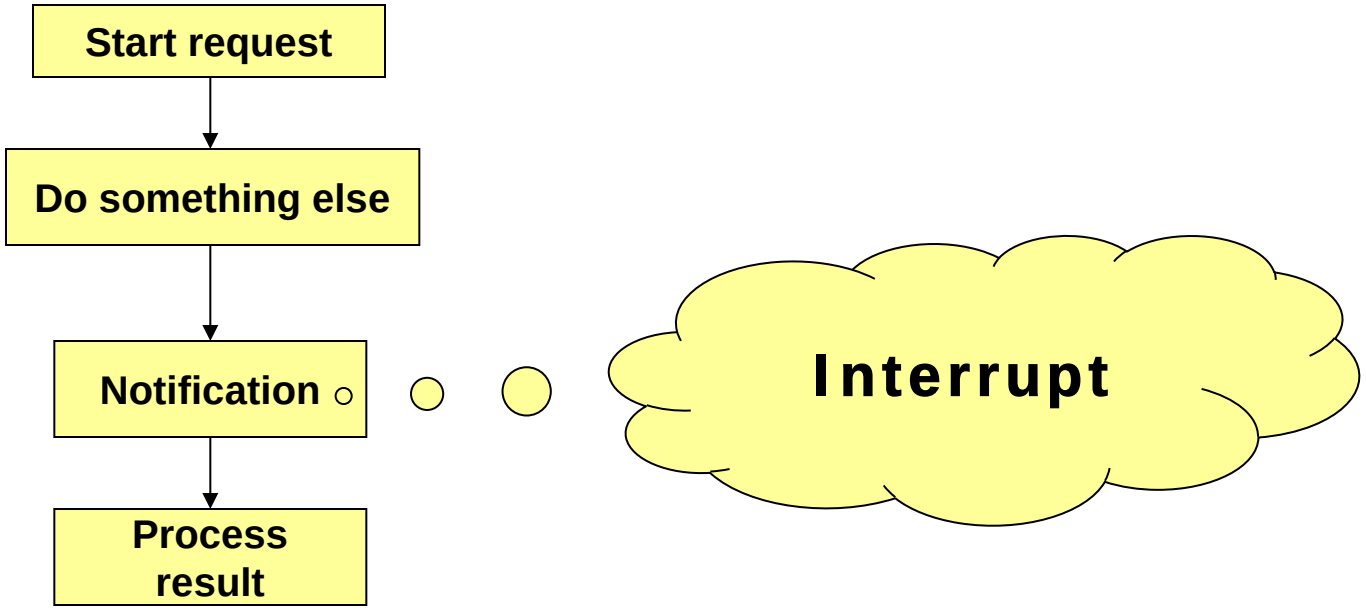
- Prevent compiler optimization
- Forces code to reload value

```
extern volatile int ISR_Flag;
```

```
void UART_ISR(void) {
  …
  ISR_Flag = 1;
  …
}
```

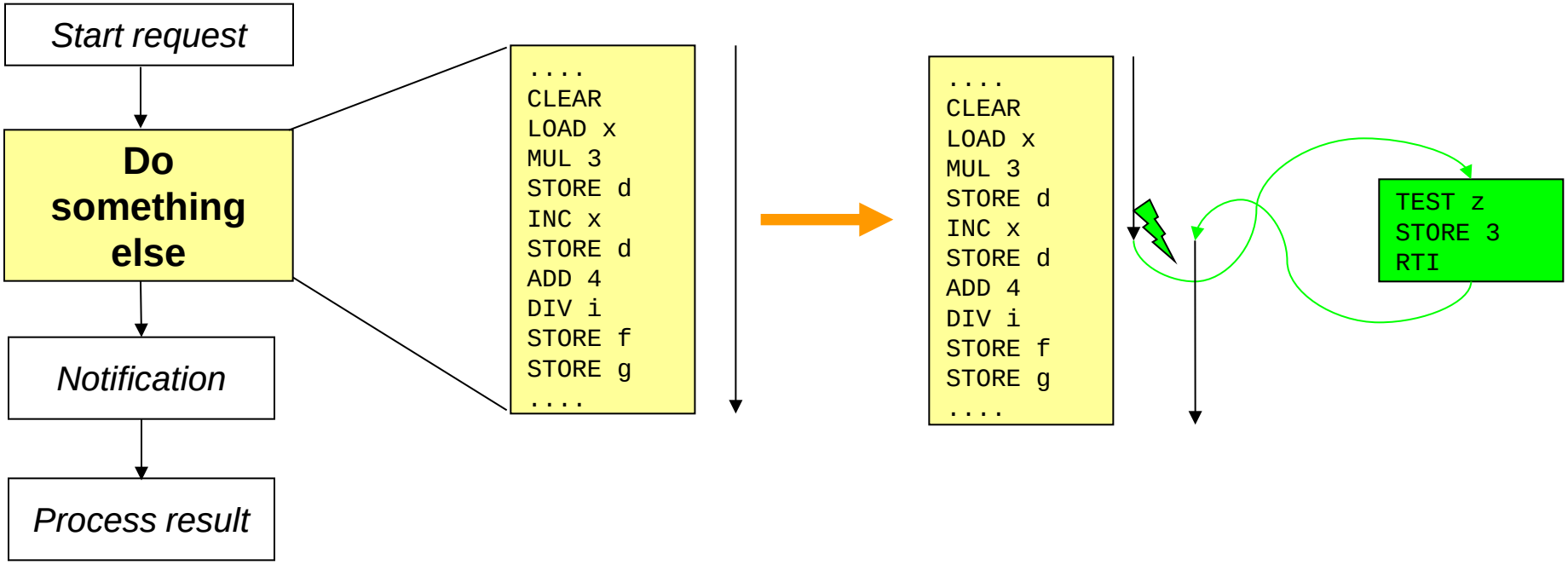
```
void main(void) {
  …
  ISR_Flag = 0; /* reset */
  UART_Send('h'); /* will trigger interrupt */
  /* wait for interrupt flag… */
  while(!ISR_Flag); /* wait for ISR */
  …
}
```

C) Interrupt Synchronization



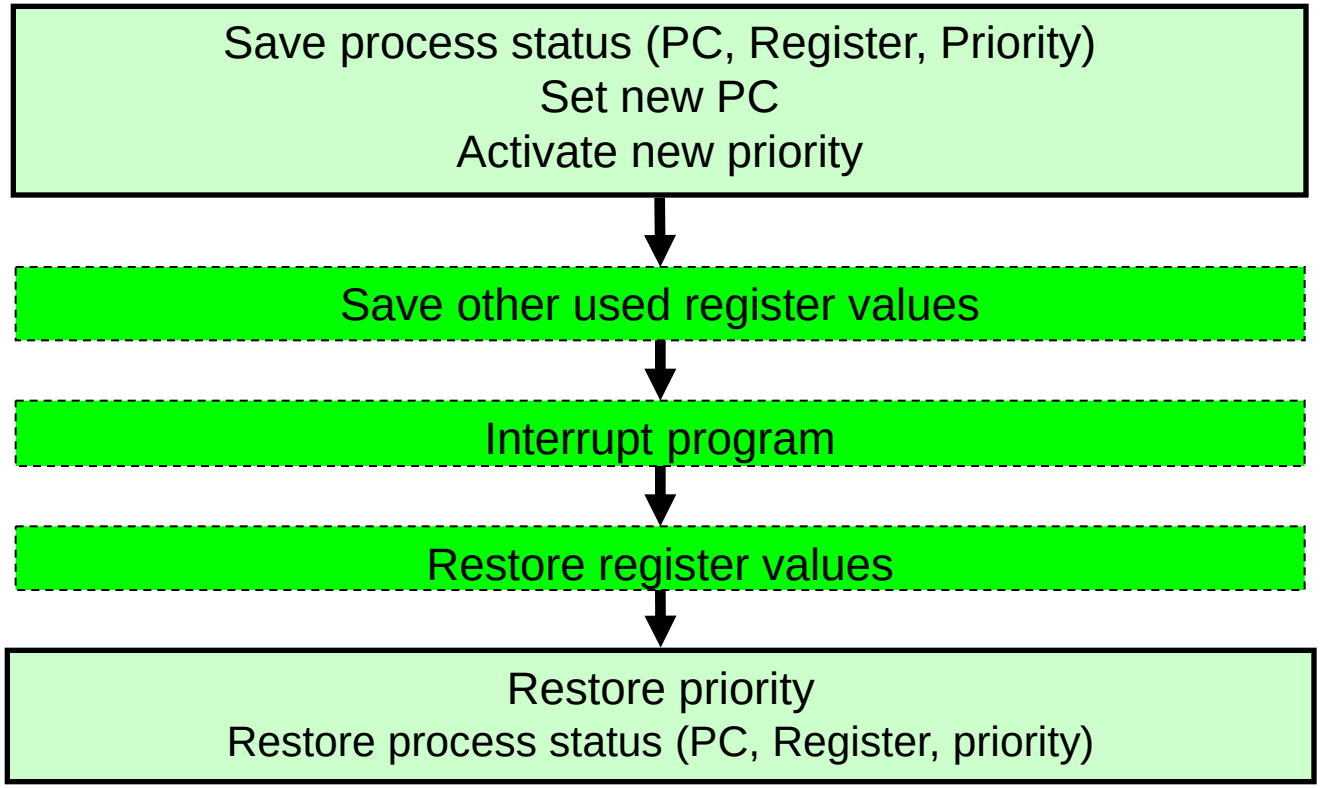
- No waiting
- Better performance (consider overhead!)

Interrupt Synchronization



- Program needs to be able to branch to somewhere else
- Main program: sequence as there would be no branching
- Need to save/restore program status!

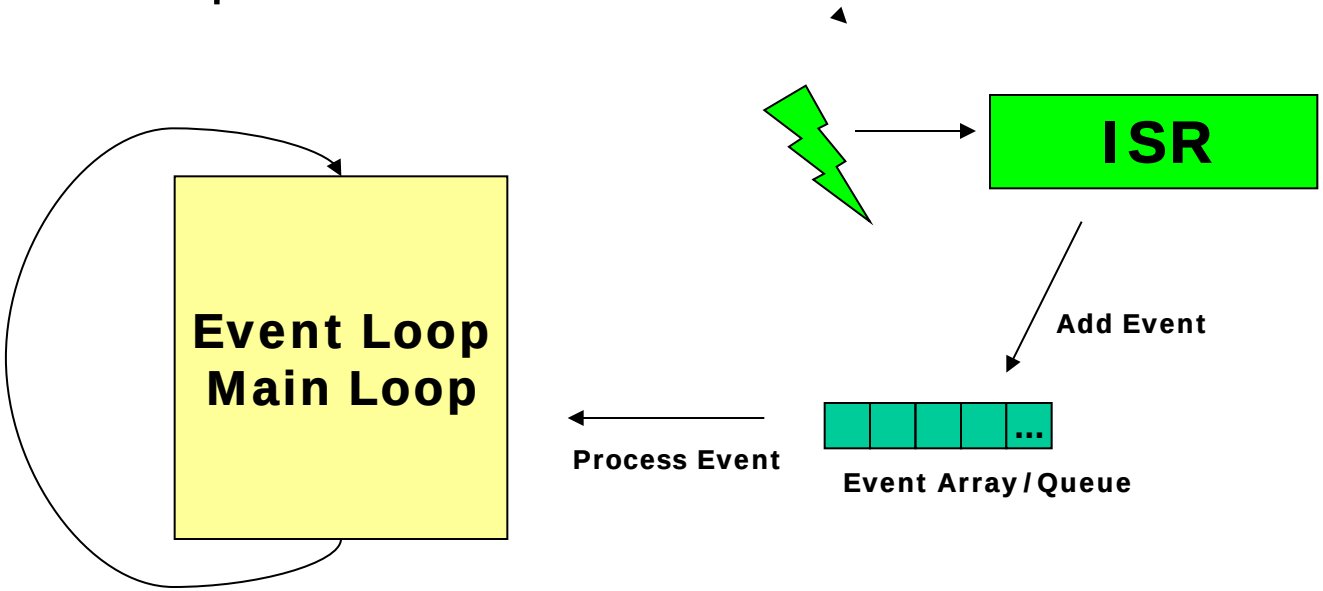
Interrupt Execution



Speed/performance for the context switch?
Speed of the Interrupt program?

Interrupt Execution Speed

- ISR: as efficient and straight forward as possible
- Possible approach: Event/main Loop/Handler
- Event Handler does the 'heavy' workload
- Interrupt Service Routines: Create/Add events



Summary: Synchronization Methods

```
ADCCRL |= 0x80; /* start conversation */
for (i=0; i<2000; i++); /* wait for some time… */
Value = ADCDATA; /* read value */
```

```
ADCCRL |= 0x80; /* start conversation */
while(!(ADCCRL&1)); /* wait until complete */
Value = ADCDATA; /* read value */
```

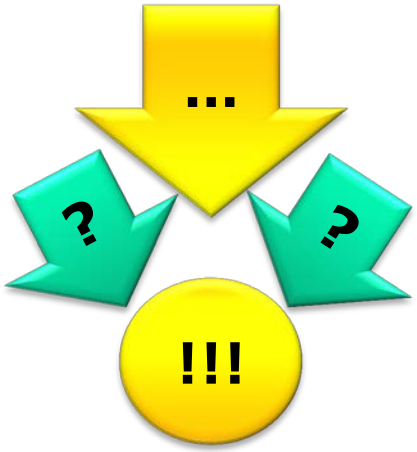
```
 ADCCRL |= 0x80; /* start conversation */
 …
}

interrupt void ADC_OnFinish(void) {
  Value = ADCDATA; /* read value */
}
```

Summary

- *Problem: Why synchronization?*

- Computer and sequential execution
- Synchronization
  - Realtime
  - Gadfly
  - Interrupts



Lab: Synchronization

- Add WAIT component to project
- Evaluate different synchronization methods