

# Real-Time Embedded Systems

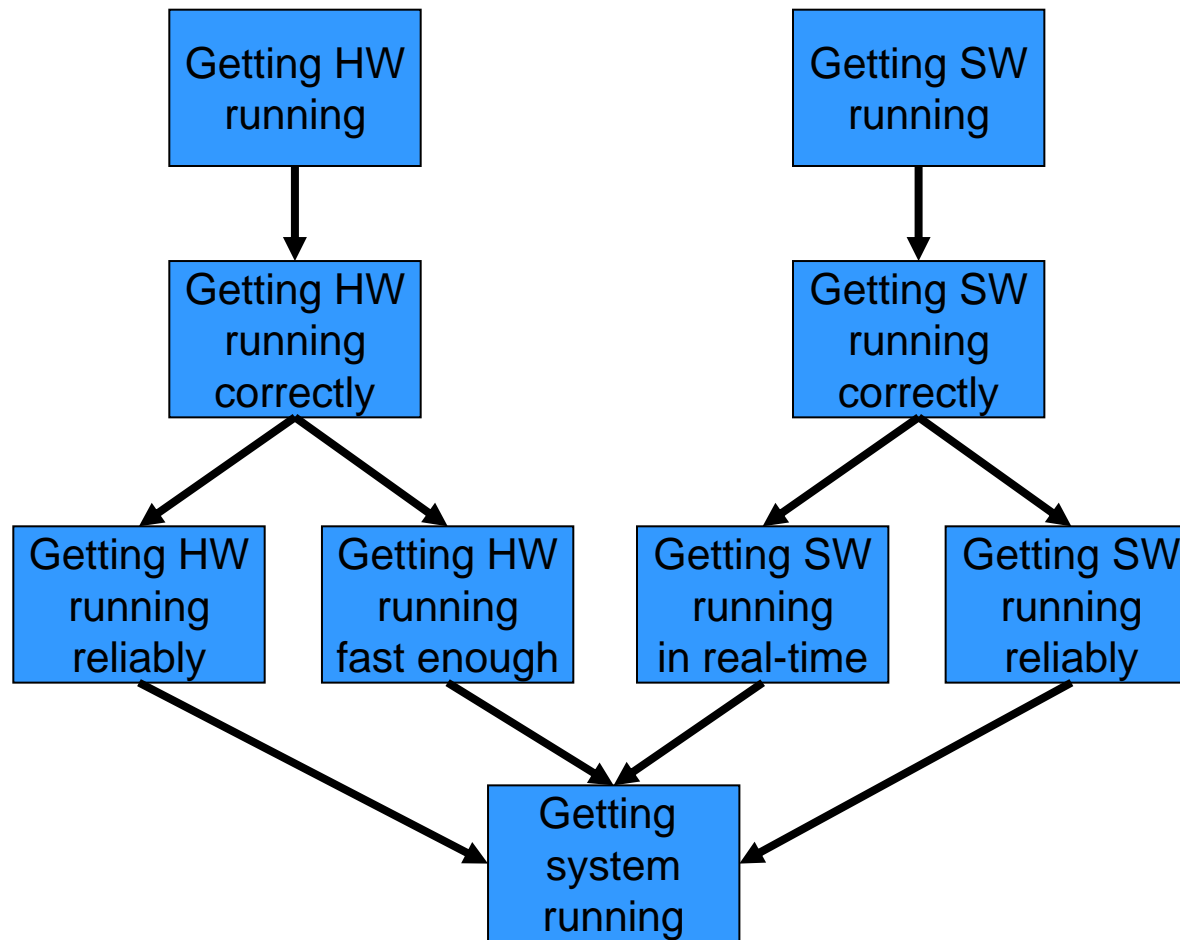
## CpE-450 Spring 07

Class 7

Bruce McNair

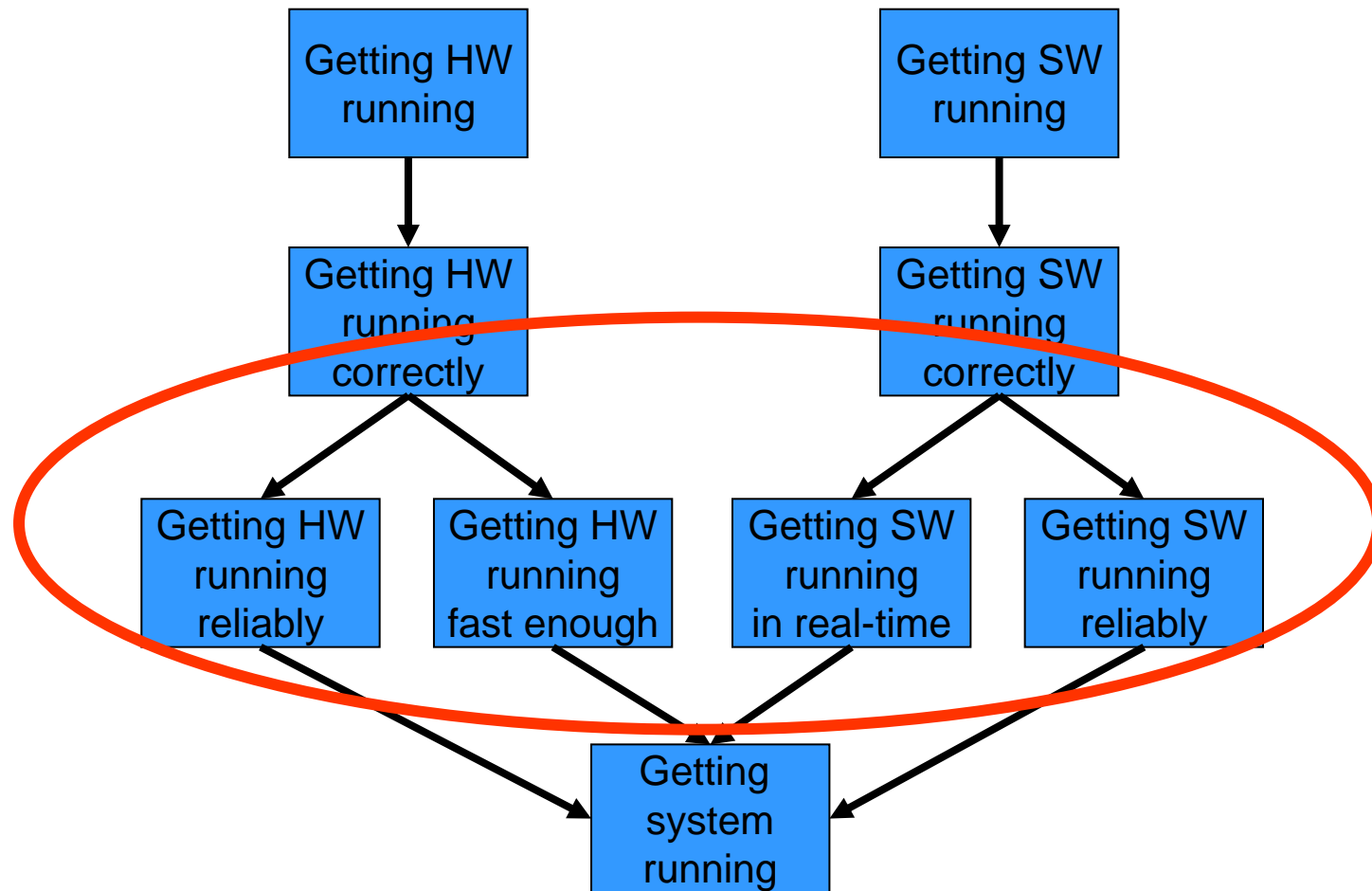
[bmcnair@stevens.edu](mailto:bmcnair@stevens.edu)

# Getting a Real-Time Embedded System Running



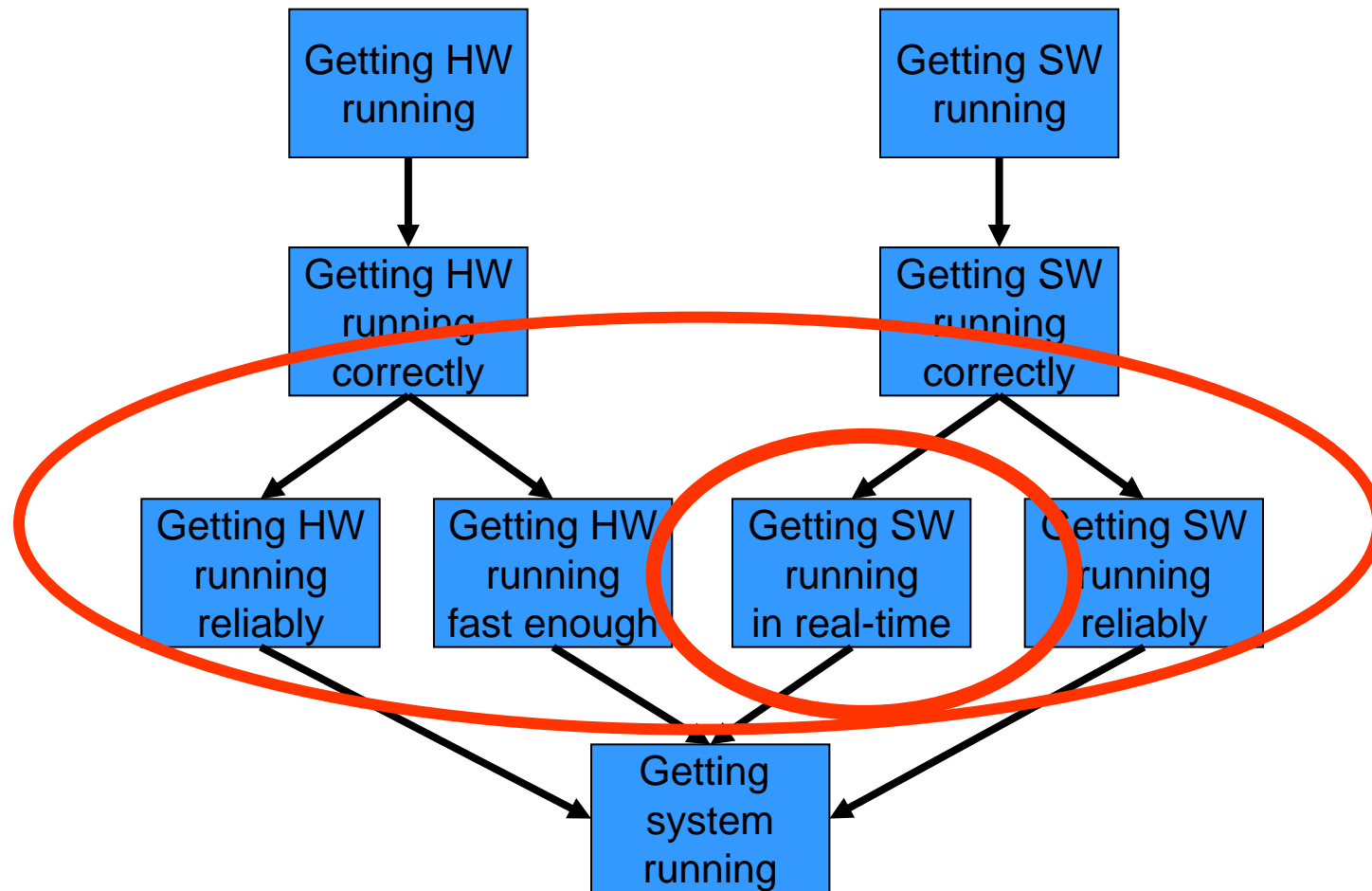
# Getting a Real-Time Embedded System Running

- Evaluating performance and correctness



# Getting a Real-Time Embedded System Running

- Evaluating performance and correctness



# Assessing Real-Time Performance

- Polling

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        wait_for_next_frame( );
    }
}
```

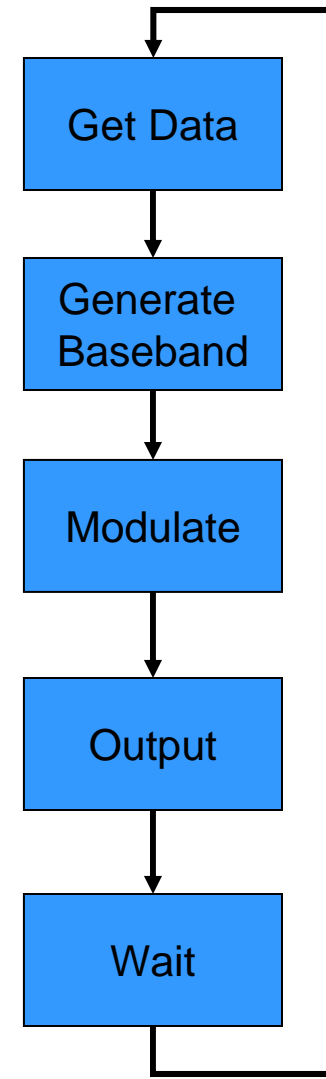
```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
{
    int i;
    static float phi=0;
    #define twopi=6.28318531
    float deltaPHI;
    deltaPHI=twopi*deltaT*fc;
    int phase;
    for(i=0; i<N, i++)
    {
        phase = (int) phi*SIZE/twopi;
        M[i] = S[i].real*cosine_table[phase] +
              S[i].imag*sine_table[phase];
        phi += deltaPHI;
        if (phi>twopi)
        {
            phi -= twopi;
        }
    }
}
```

# Assessing Real-Time Performance

- Polling

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        wait_for_next_frame( );
    }
}
```



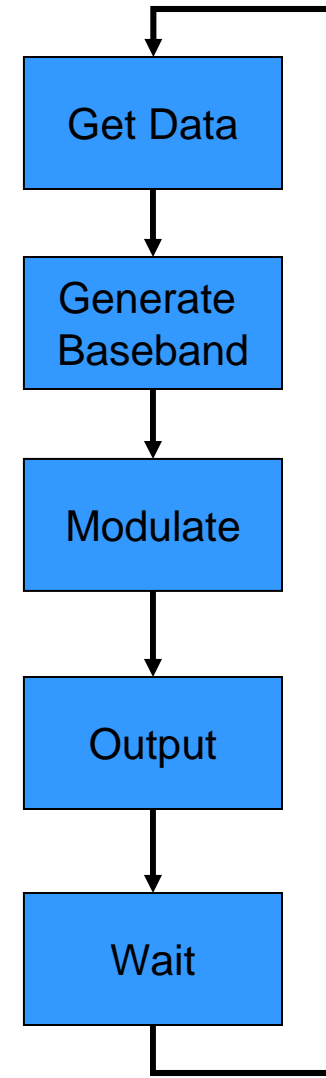
# Assessing Real-Time Performance

- Polling

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        wait_for_next_frame( );
    }
}
```

How can you tell if system is running in real-time?

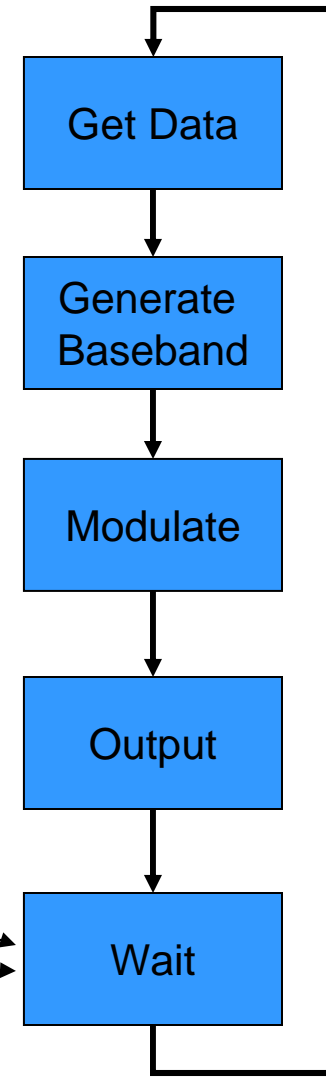


# Assessing Real-Time Performance

- Polling

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        wait_for_next_frame( );
    }
}
```



Real-time:  $wait > 0$

Not Real-time: " $wait < 0$ "

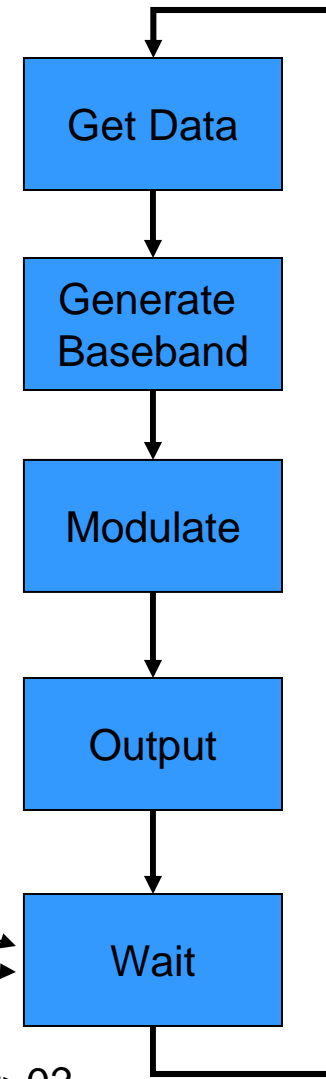
How can you tell if system is running in real-time?

# Assessing Real-Time Performance

- Polling

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        wait_for_next_frame( );
    }
}
```



Real-time:  $wait > 0$

Not Real-time: " $wait < 0$ "

How can you tell if system is running in real-time?

How do you know if  $wait > 0$ ?

# Assessing Real-Time Performance

- ISR driven

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

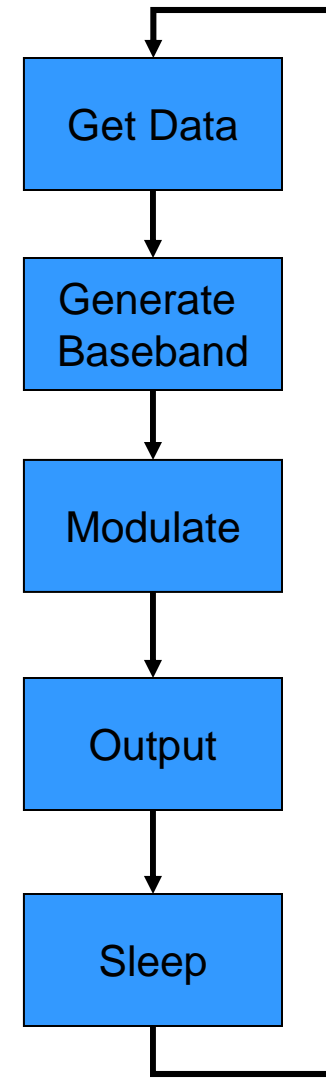
void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    set_up_interrupts(INTERRUPT_ON_DATA_PRESENT);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        sleep( );
    }
}

void interrupt_handler_DATA_PRESENT(void)
{
    /* wake up main( ) when data arrives */
}
```

CpE450 }  
3/5/2007

Copyright ©2005-2006  
Stevens Institute of Technology - All rights reserved

7-10/60



# Assessing Real-Time Performance

- ISR driven

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};
```

How can you tell if system is running in real-time?

How do you know if sleeptime > 0?

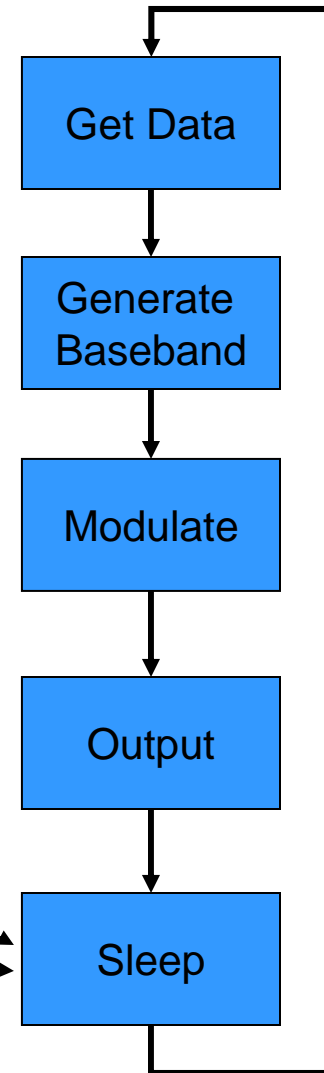
```
void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    set_up_interrupts(INTERRUPT_ON_DATA_PRESENT);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        sleep( );
    }
}

void interrupt_handler_DATA_PRESENT(void)
{
    /* wake up main( ) when data arrives */
}
```

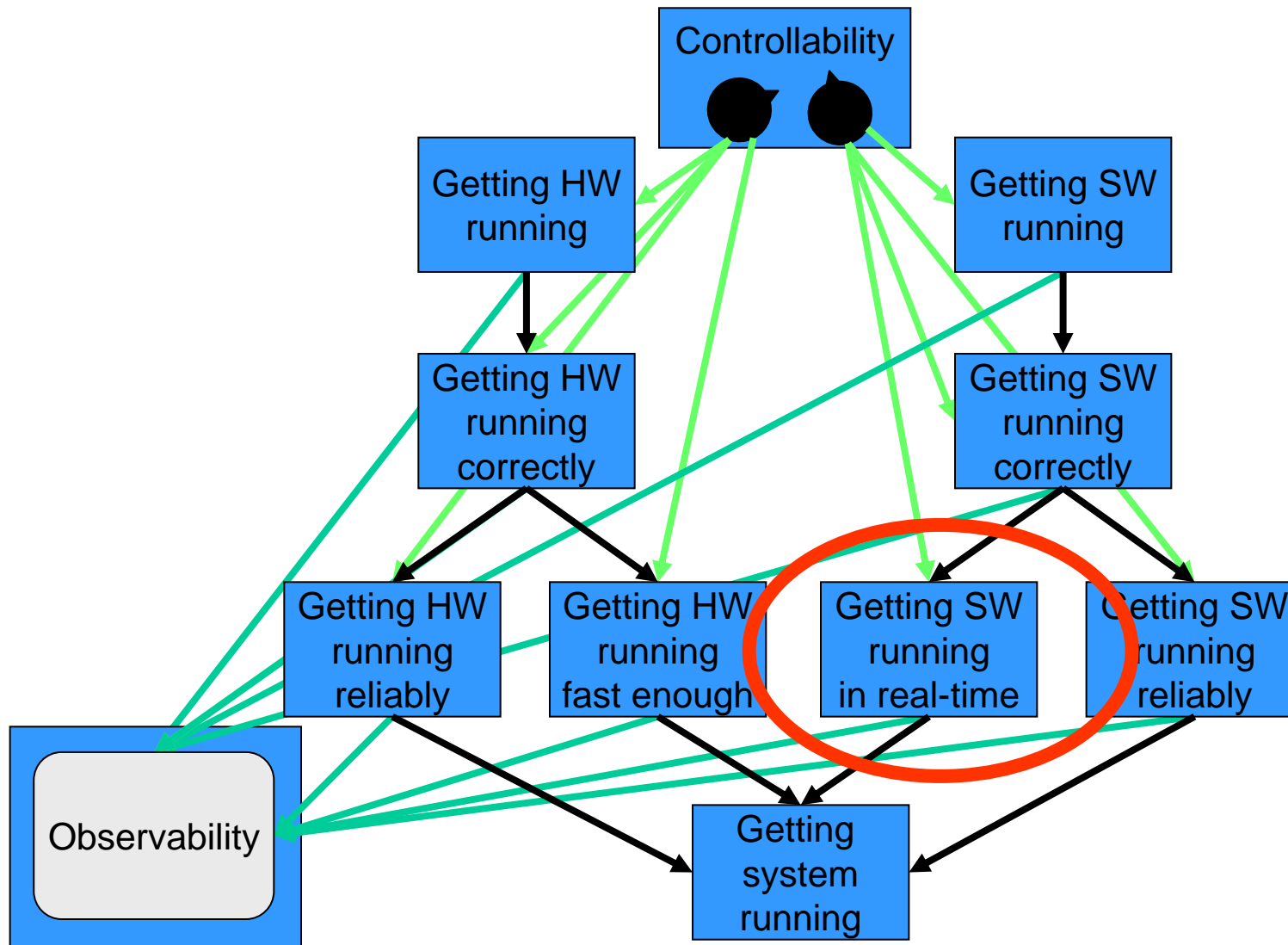
sleep( );

Real-time: sleeptime > 0

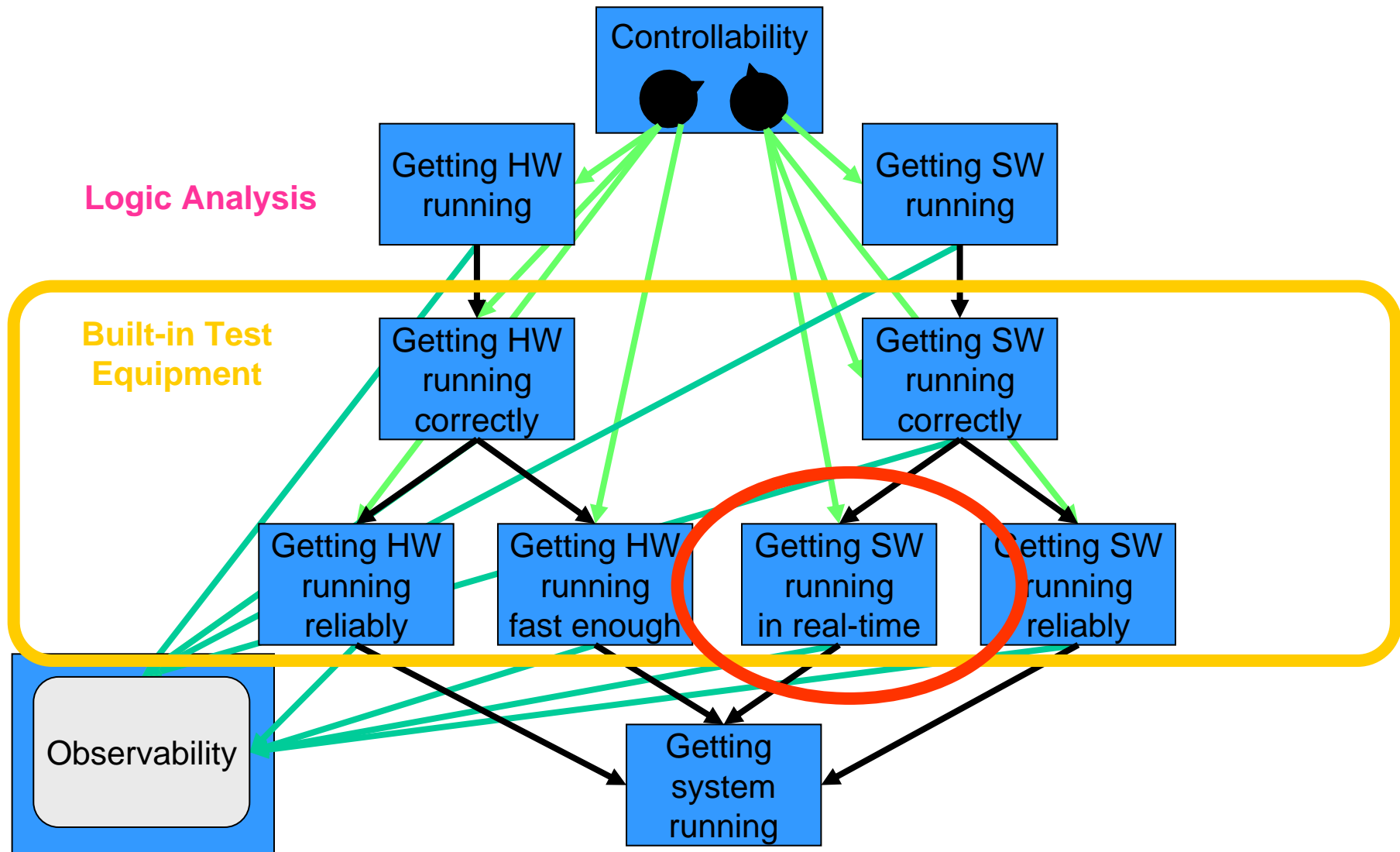
Not Real-time: "sleeptime < 0"



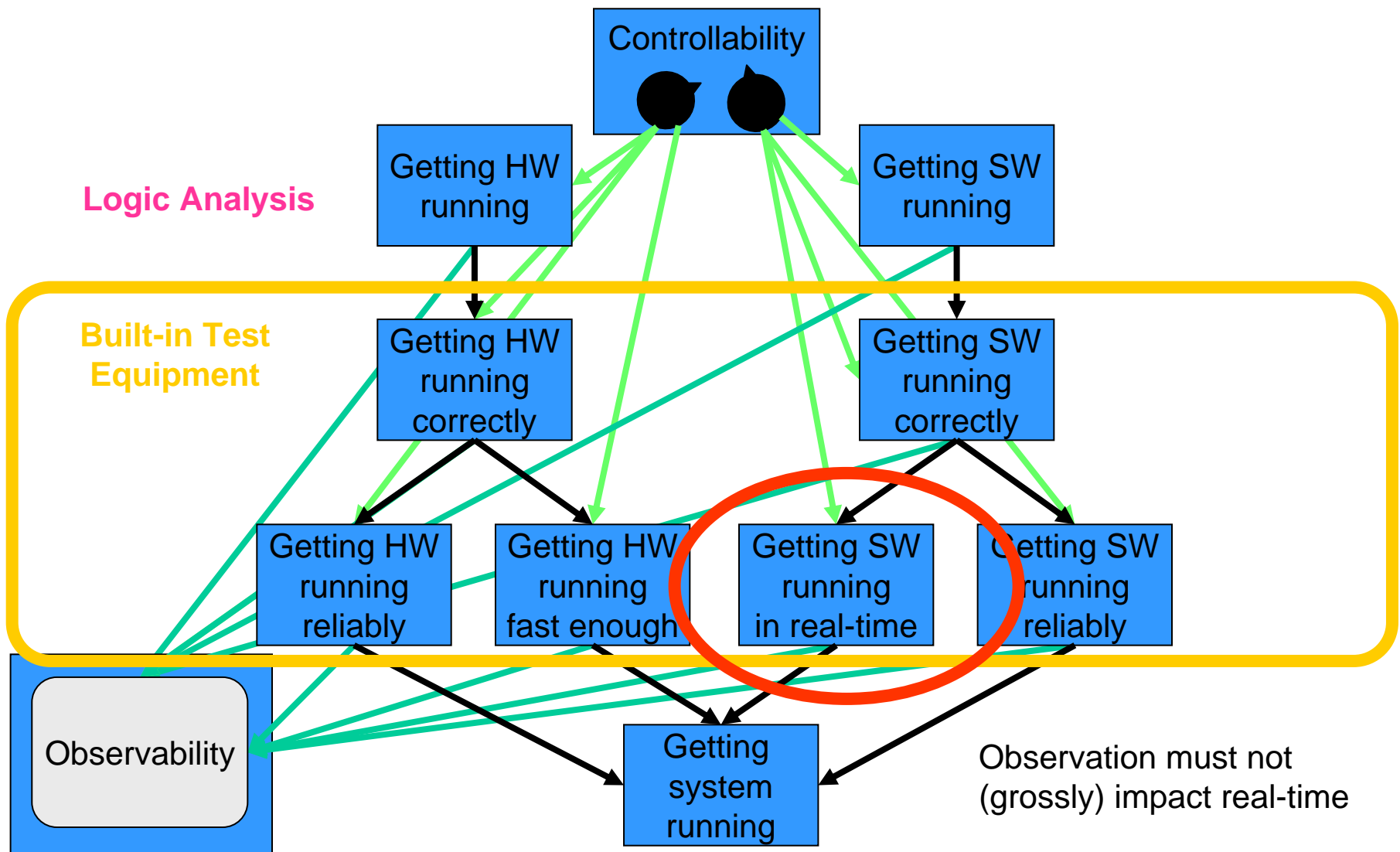
# Getting a Real-Time Embedded System Running



# Getting a Real-Time Embedded System Running



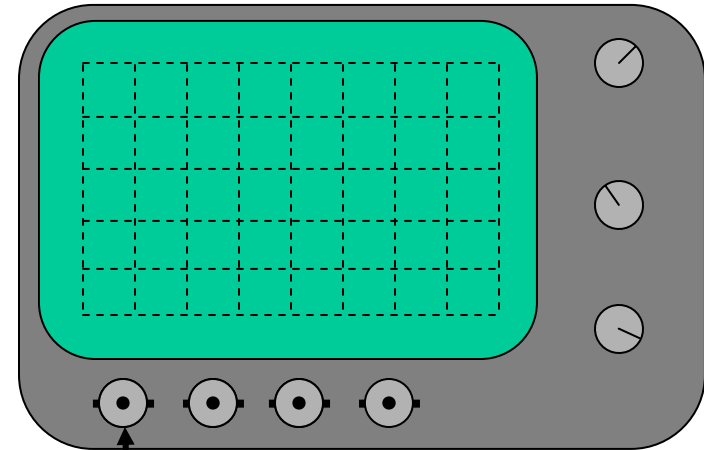
# Getting a Real-Time Embedded System Running



# Real-time Observability

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

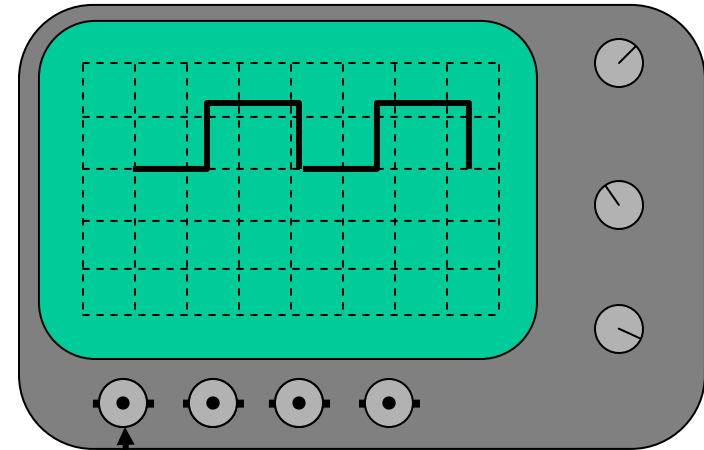
void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        set_output_flag(1);
        wait_for_next_frame( );
        set_output_flag(0);
    }
}
```



# Real-time Observability

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

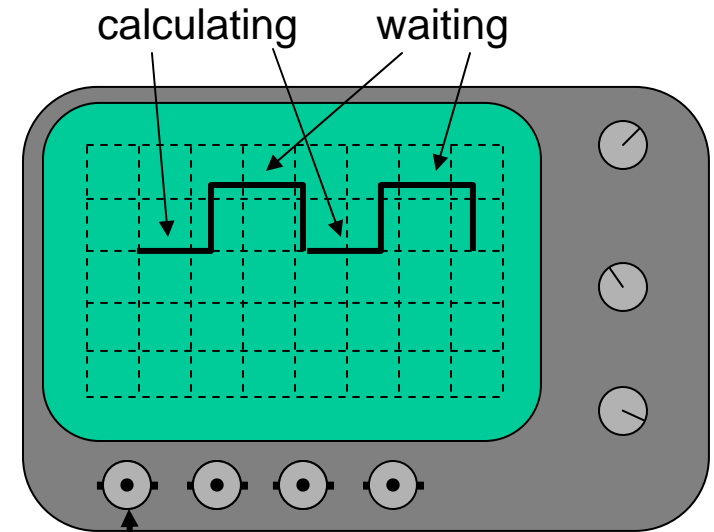
void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        set_output_flag(1);
        wait_for_next_frame( );
        set_output_flag(0);
    }
}
```



# Real-time Observability

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        set_output_flag(1);
        wait_for_next_frame( );
        set_output_flag(0);
    }
}
```

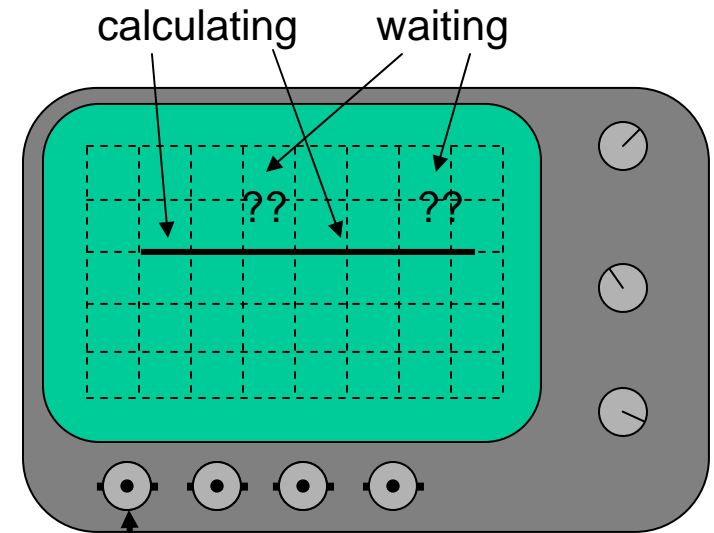


Running faster than real-time

# Real-time Observability

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        set_output_flag(1);
        wait_for_next_frame( );
        set_output_flag(0);
    }
}
```



Running slower than real-time

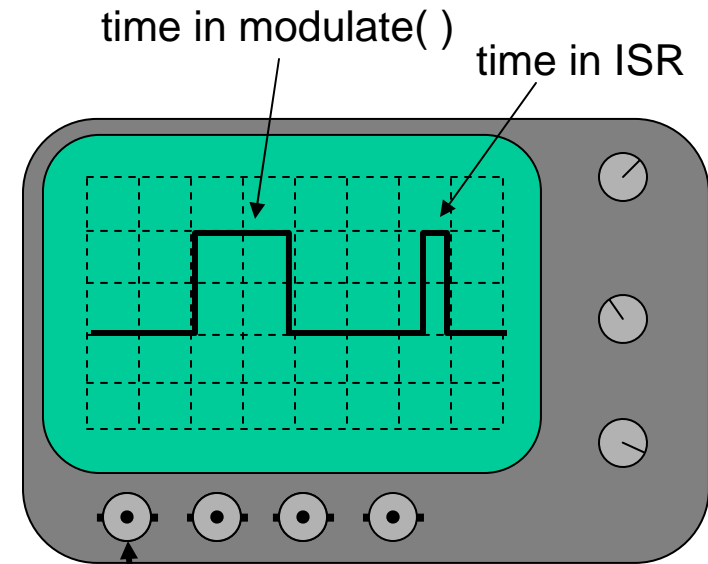
# Real-time Observability

- Observing time spent in ISR, other functions

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    set_up_interrupts(INTERRUPT_ON_DATA_PRESENT);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        set_output_flag(1);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        set_output_flag(0);
        output_mod(mod);
        sleep( );
    }
}

void interrupt_handler_DATA_PRESENT(void)
{
    set_output_flag(1);
    /* wake up main( ) when data arrives */
    set_output_flag(0);
}
```



# Real-time Tuning

- We now know how much time is spent in a routine
- We can tell how changes influence performance

# Real-time Tuning

- We now know how much time is spent in a routine
  - We can tell how changes influence performance
- **Where do we spend our efforts improving performance?**

# Real-time Performance

1. System is not running in real-time
2. System is running in real-time, but there isn't enough safety margin
3. System is running in real-time, but it is not yet complete – other features need to be added
4. System is running in real-time, but other features are likely to be added
5. System is running in real-time, but it would be desirable to improve performance allowing a slower (cheaper, lower power, ???) processor
6. .
7. .
8. .
9. Name your reason why you want to improve real-time performance.

# Pareto Principle

- Economics:
  - 80% of the resources are consumed by 20% of the population

# Pareto Principle

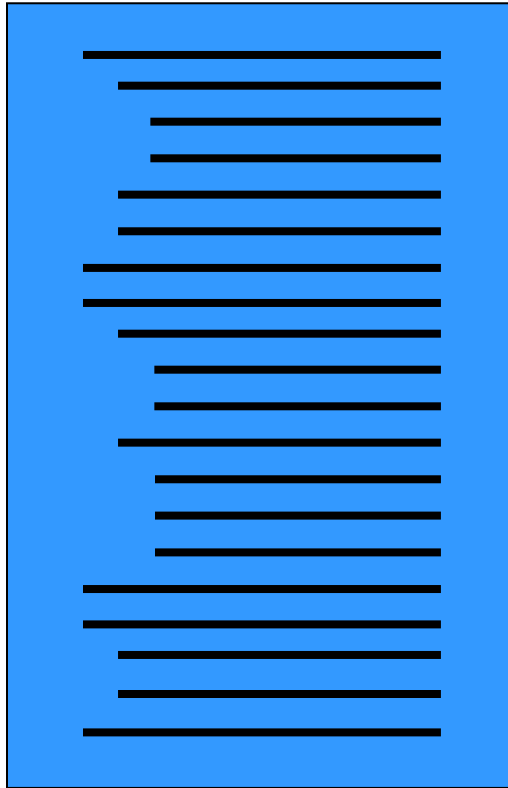
- Economics:
  - 80% of the resources are consumed by 20% of the population
- Reliability
  - 80% of the system failures are consumed by 20% of the components

# Pareto Principle

- Economics:
  - 80% of the resources are consumed by 20% of the population
- Reliability
  - 80% of the system failures are consumed by 20% of the components
- Real-time programming
  - 80% of the real-time is consumed by 20% of the code

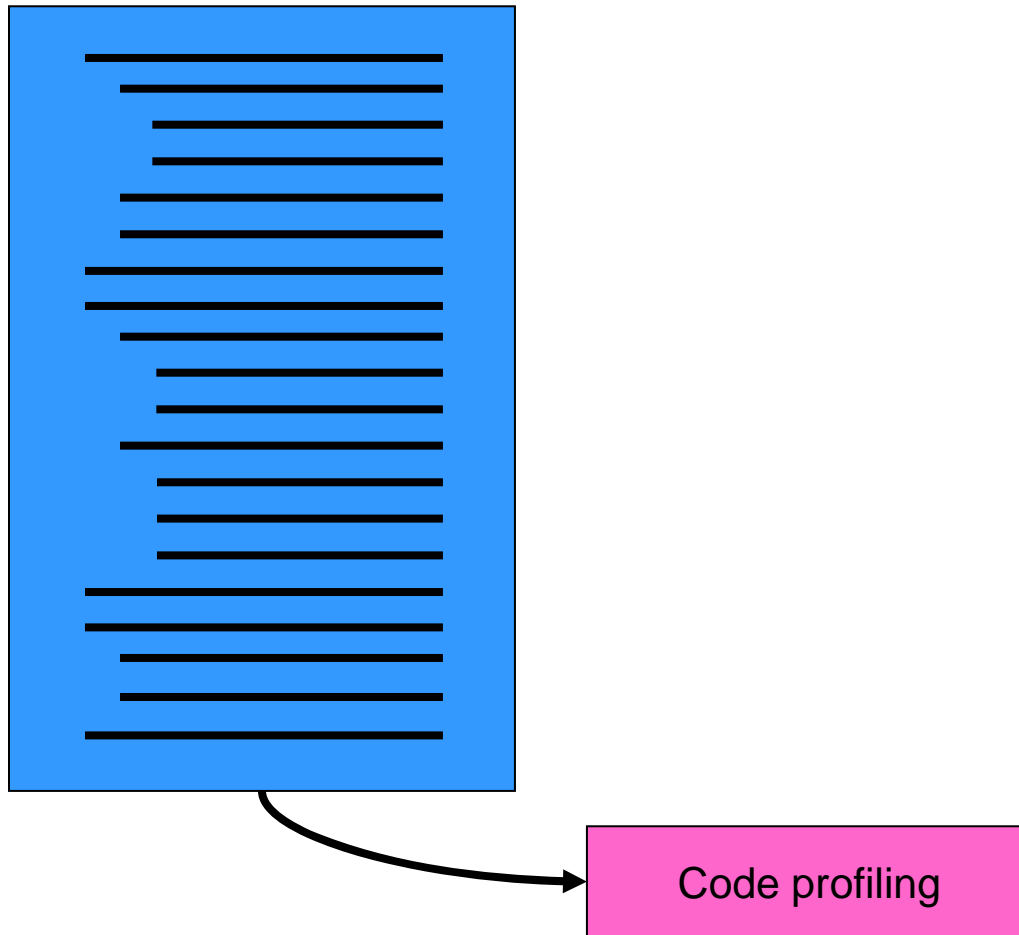
# Applying Pareto Principle to Software

- Consider a generic software module:



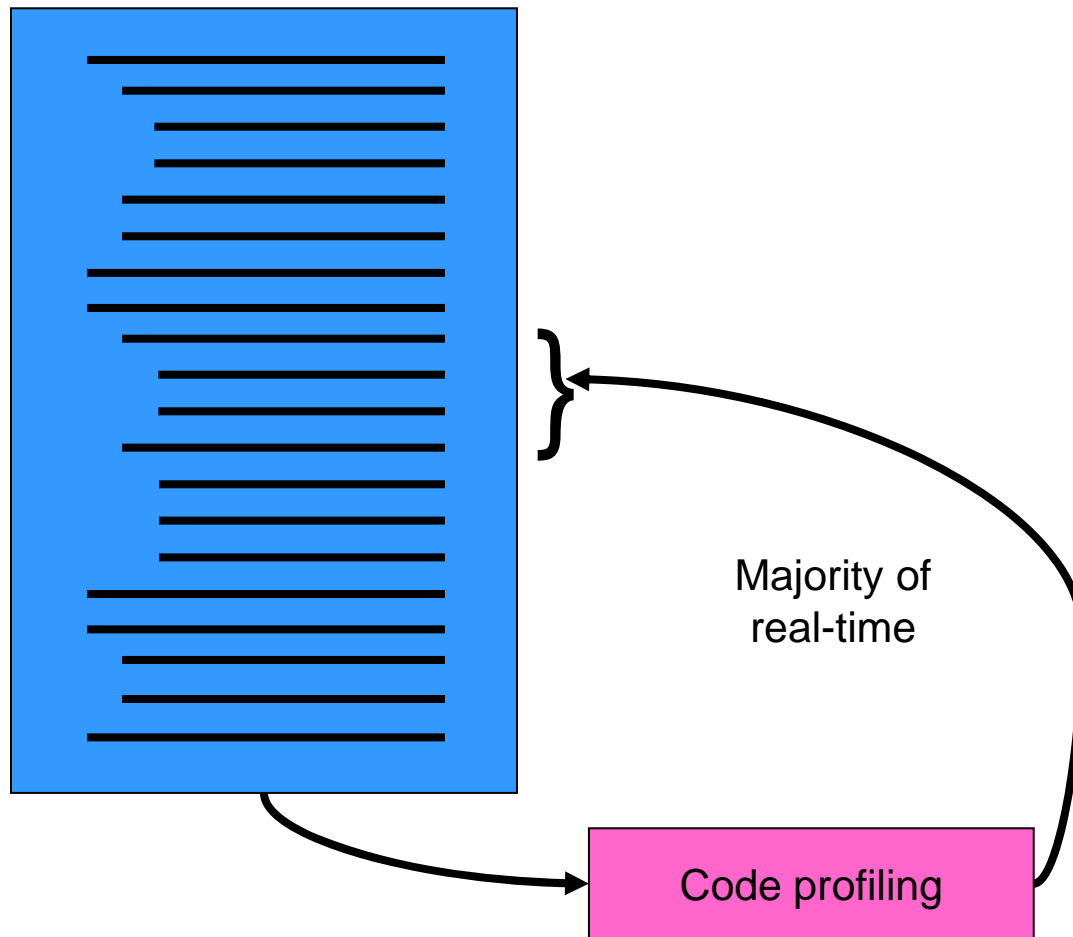
# Applying Pareto Principle to Software

- Consider a generic software module:



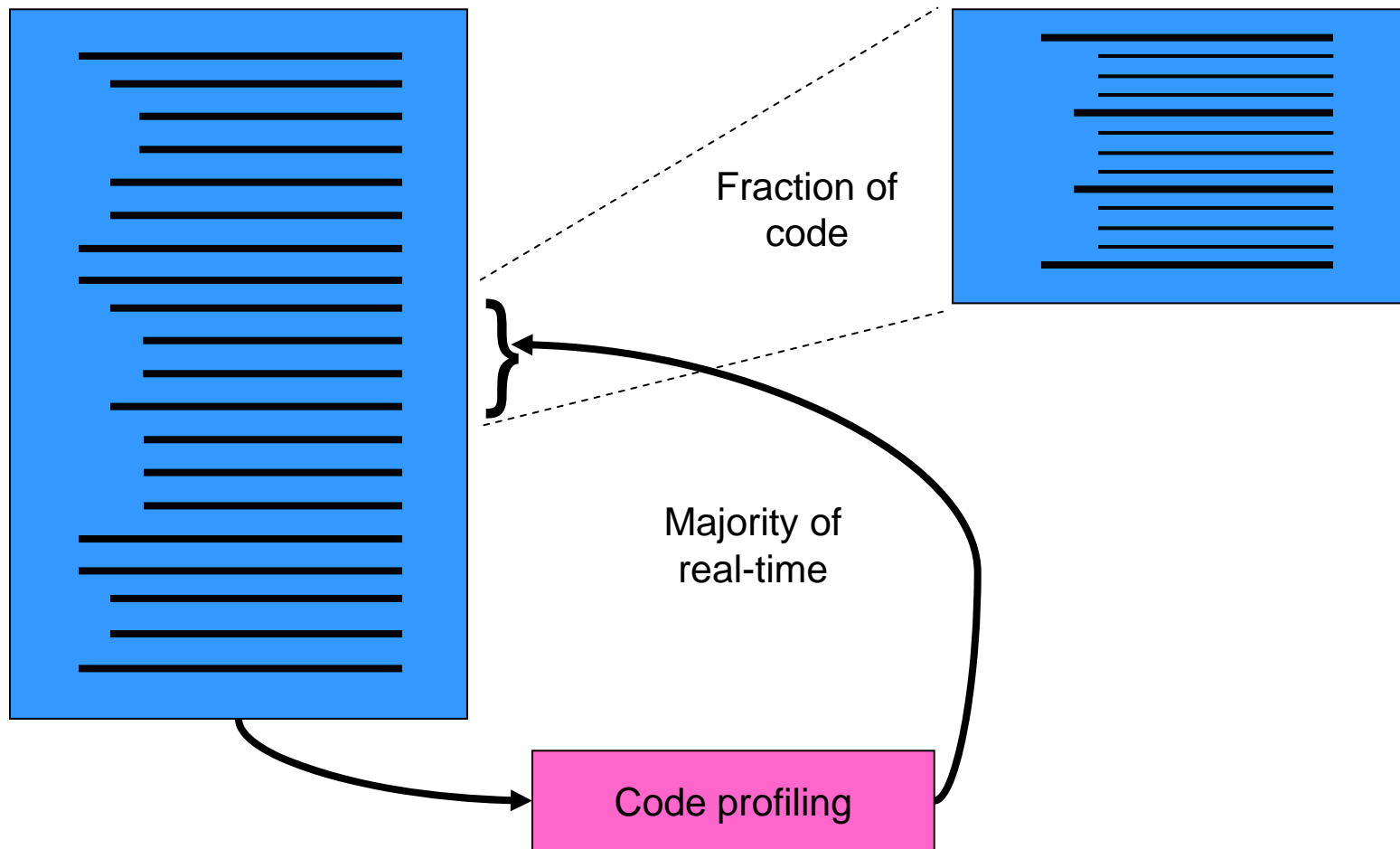
# Applying Pareto Principle to Software

- Consider a generic software module:

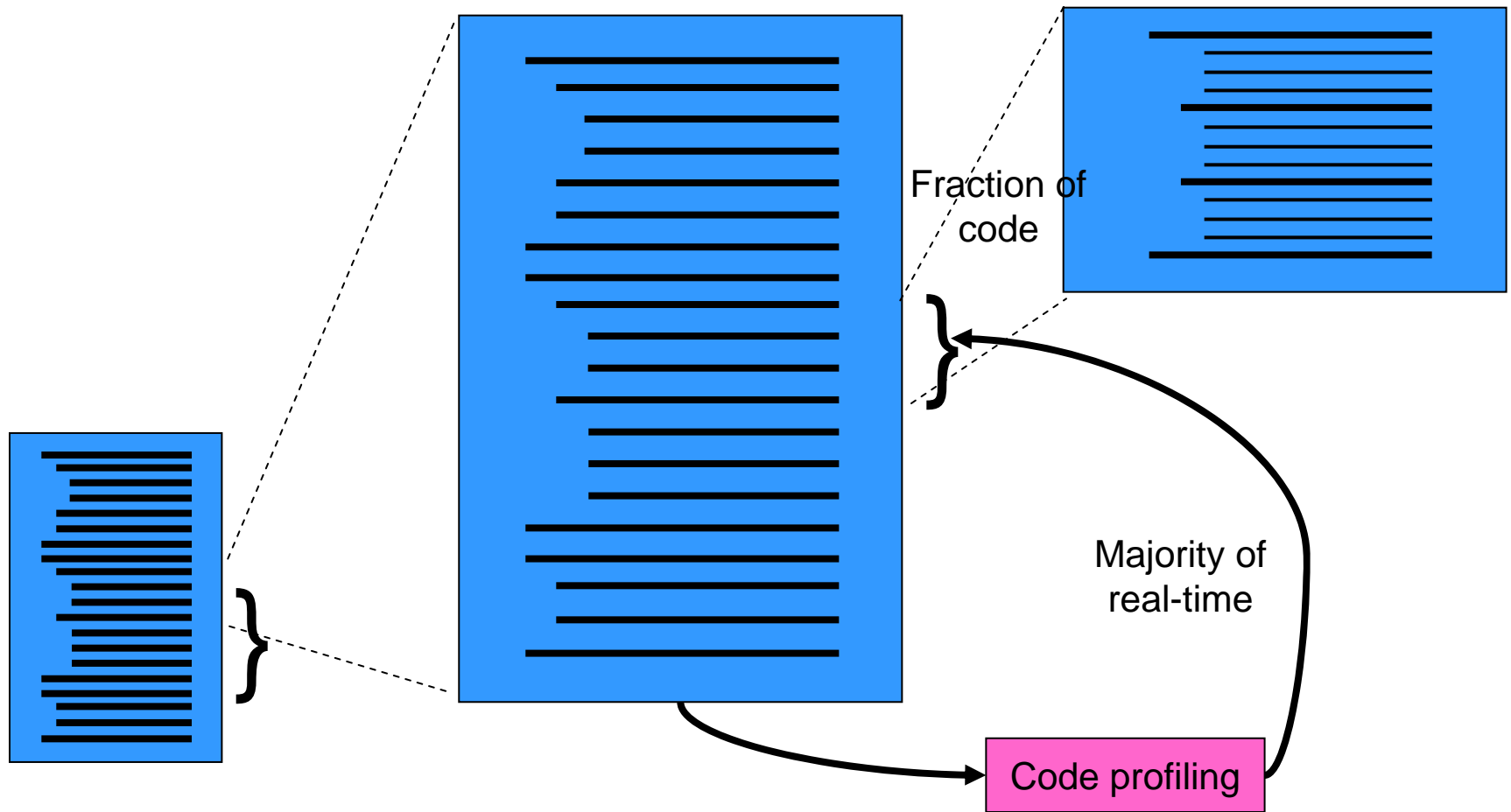


# Applying Pareto Principle to Software

- Consider a generic software module:

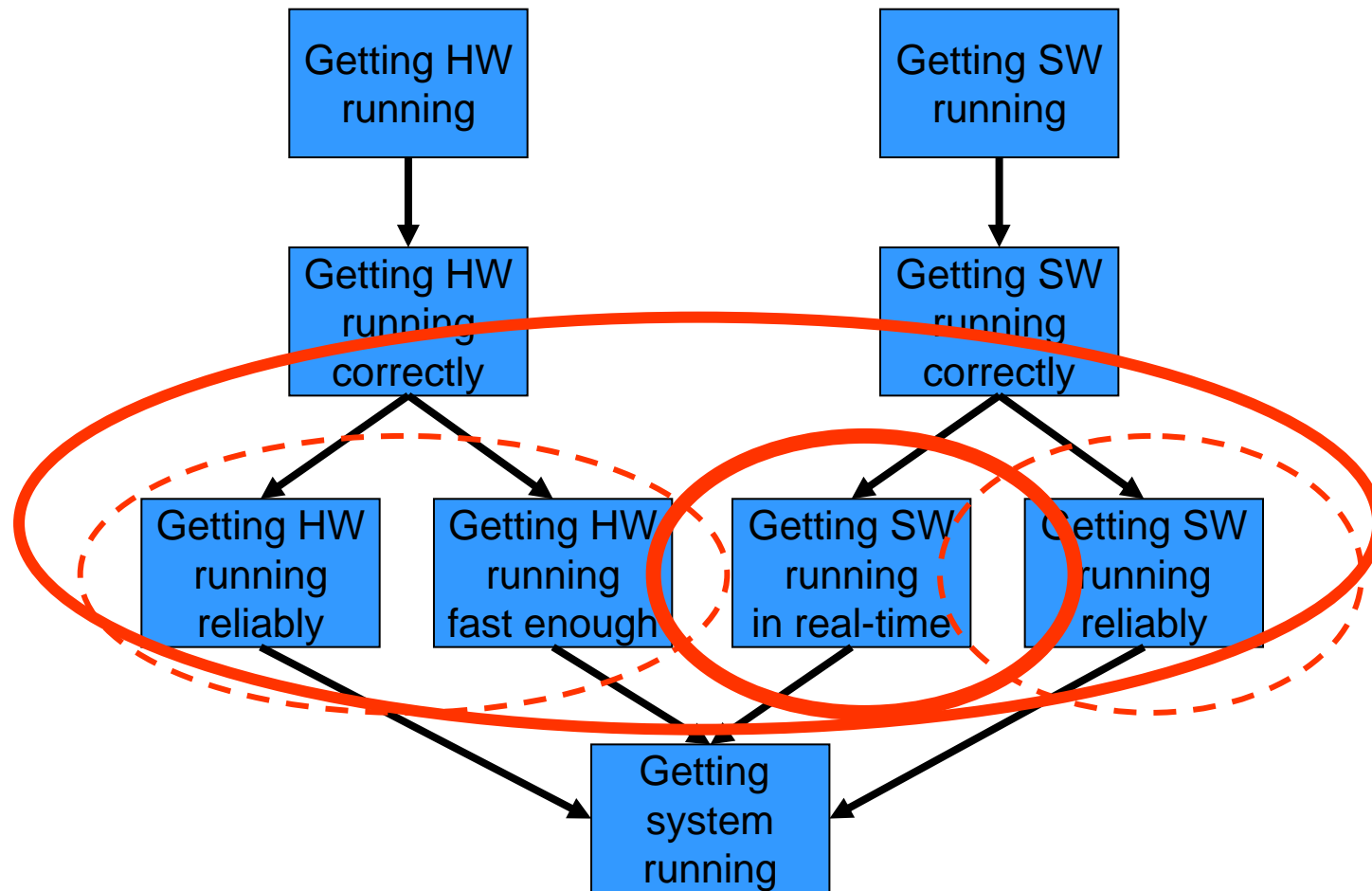


# Continuing the Process



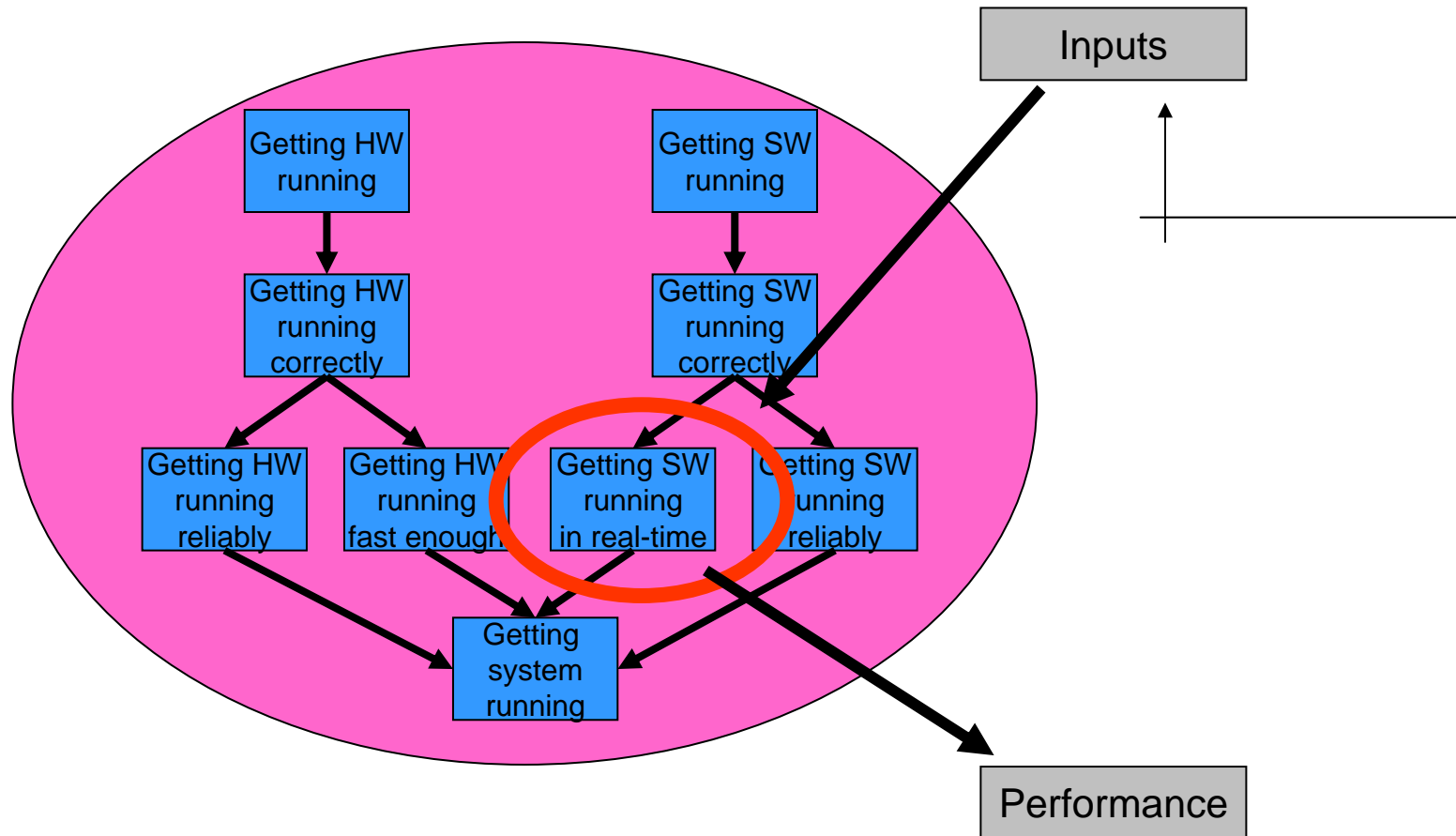
# Real-time Performance

- And other items



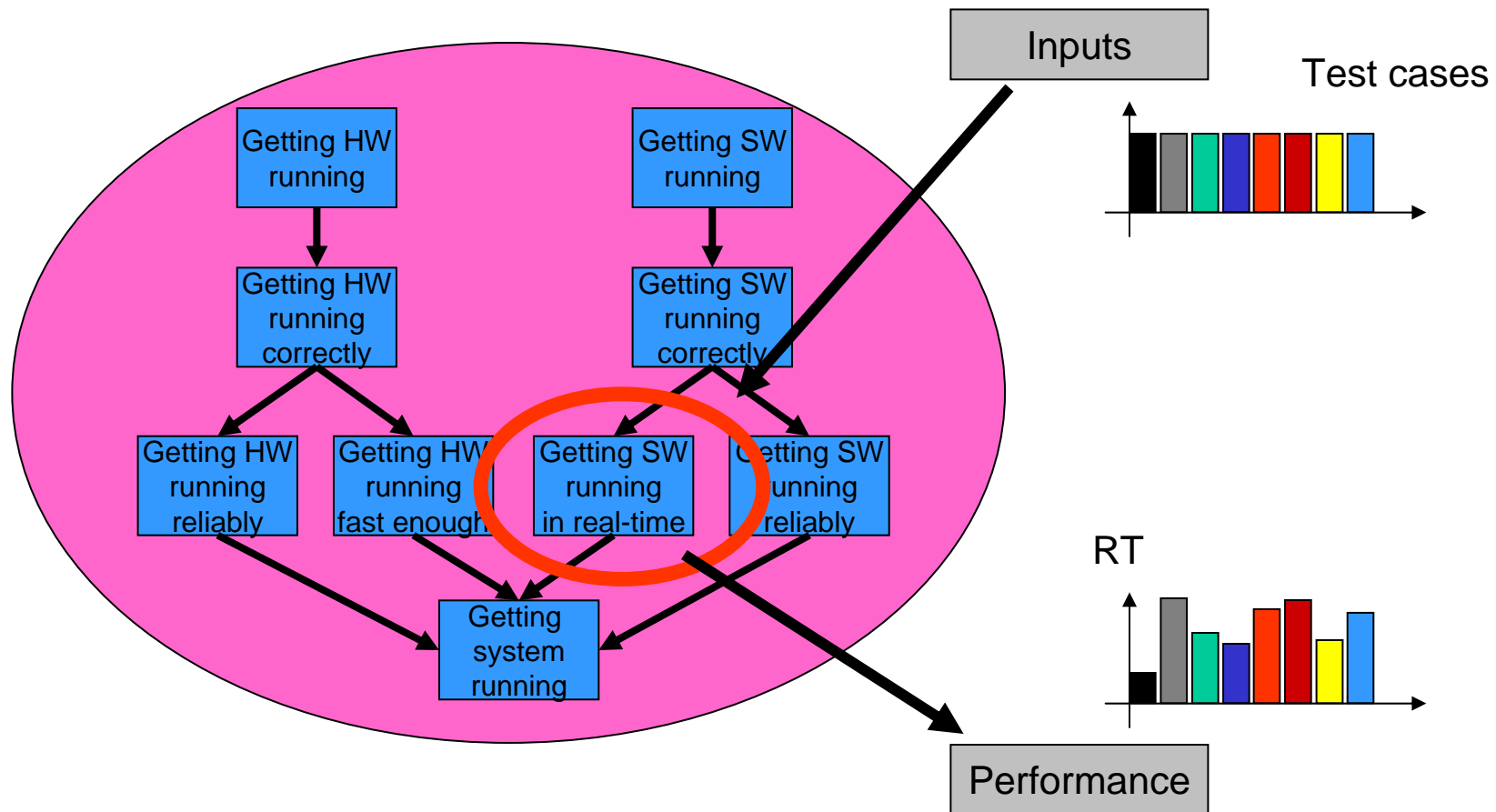
# Real-Time Performance

- Evaluating performance



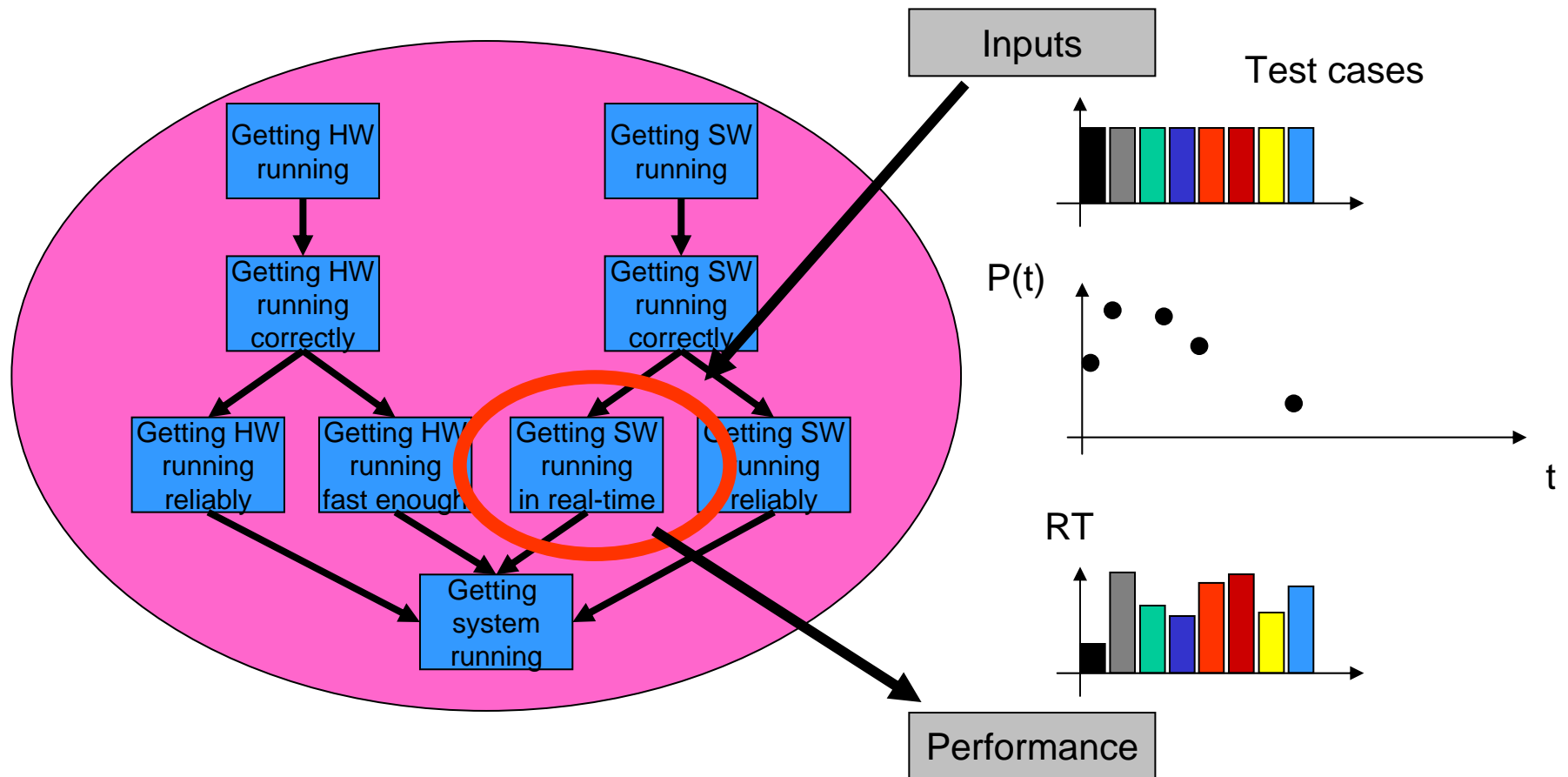
# Real-Time Performance

- Evaluating performance



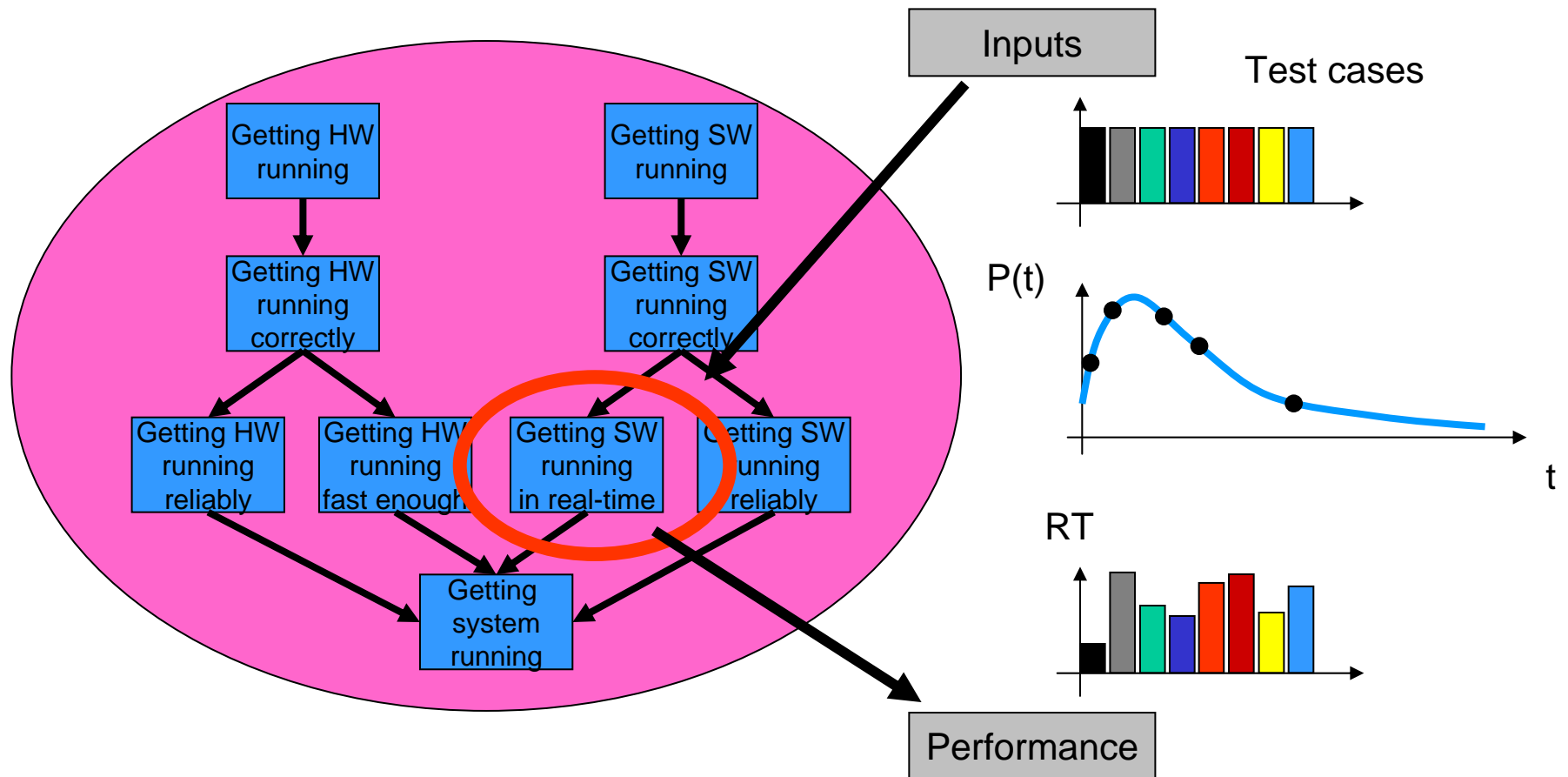
# Real-Time Performance

- Evaluating performance – a dynamically changing parameter



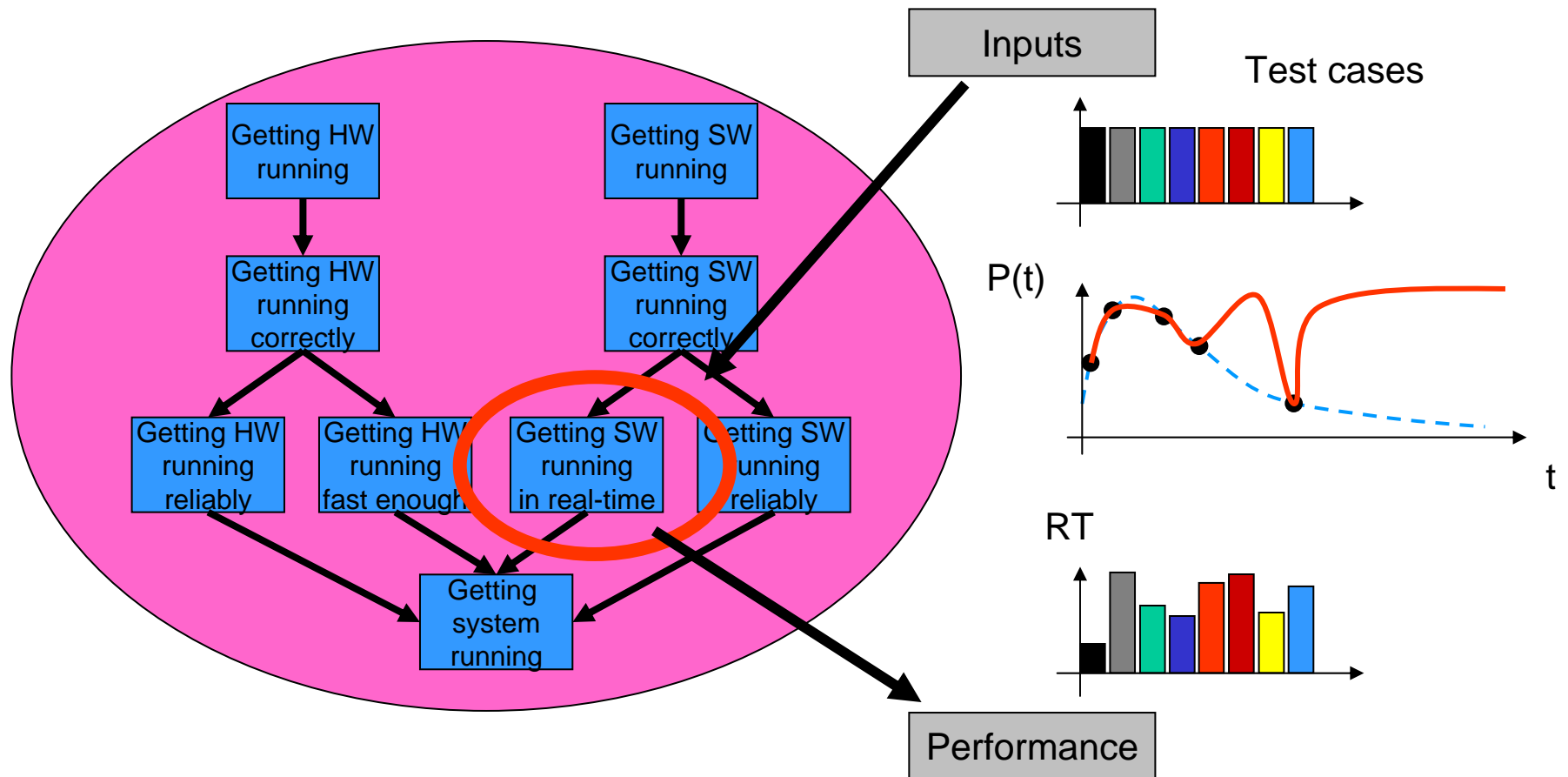
# Real-Time Performance

- Evaluating performance – a dynamically changing parameter

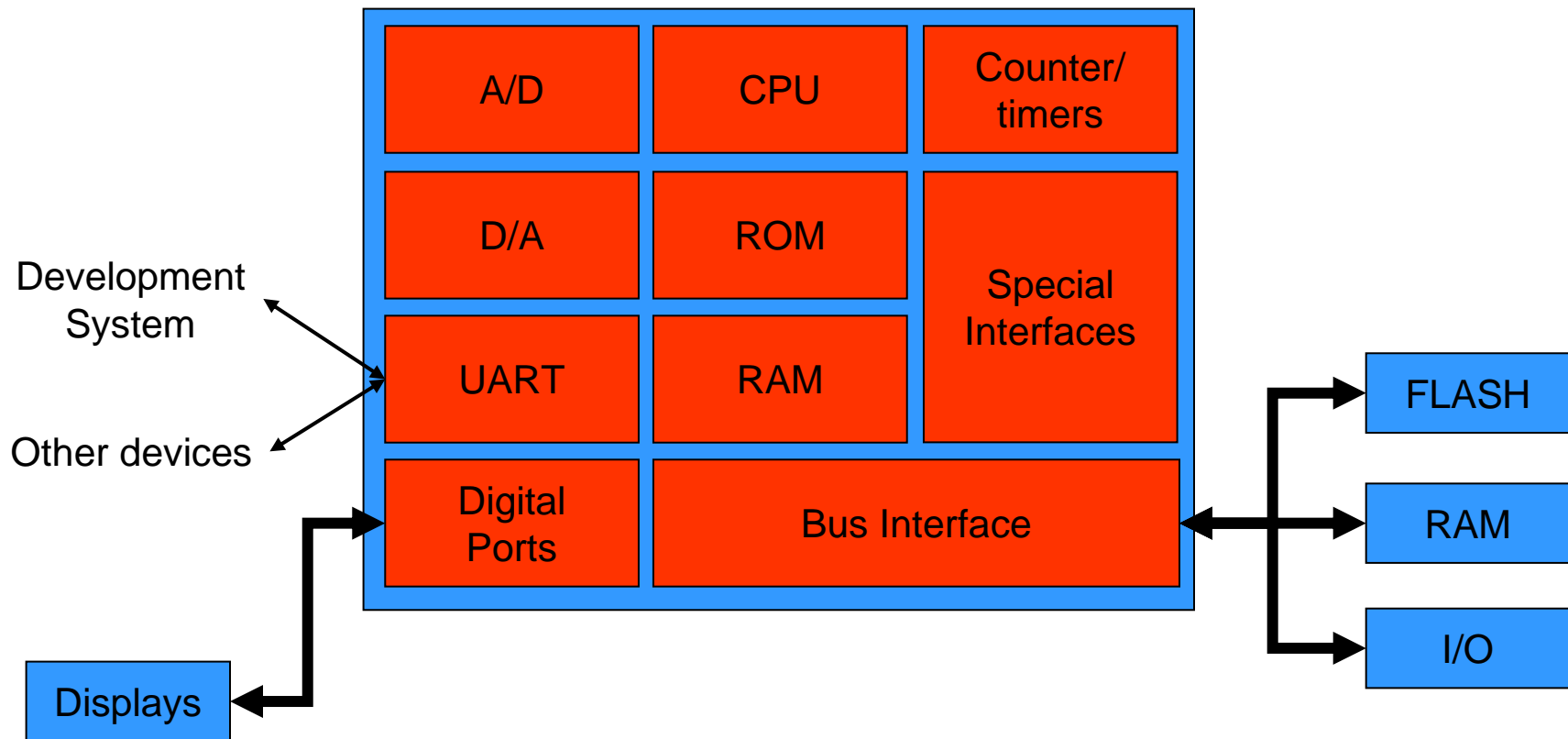


# Real-Time Performance

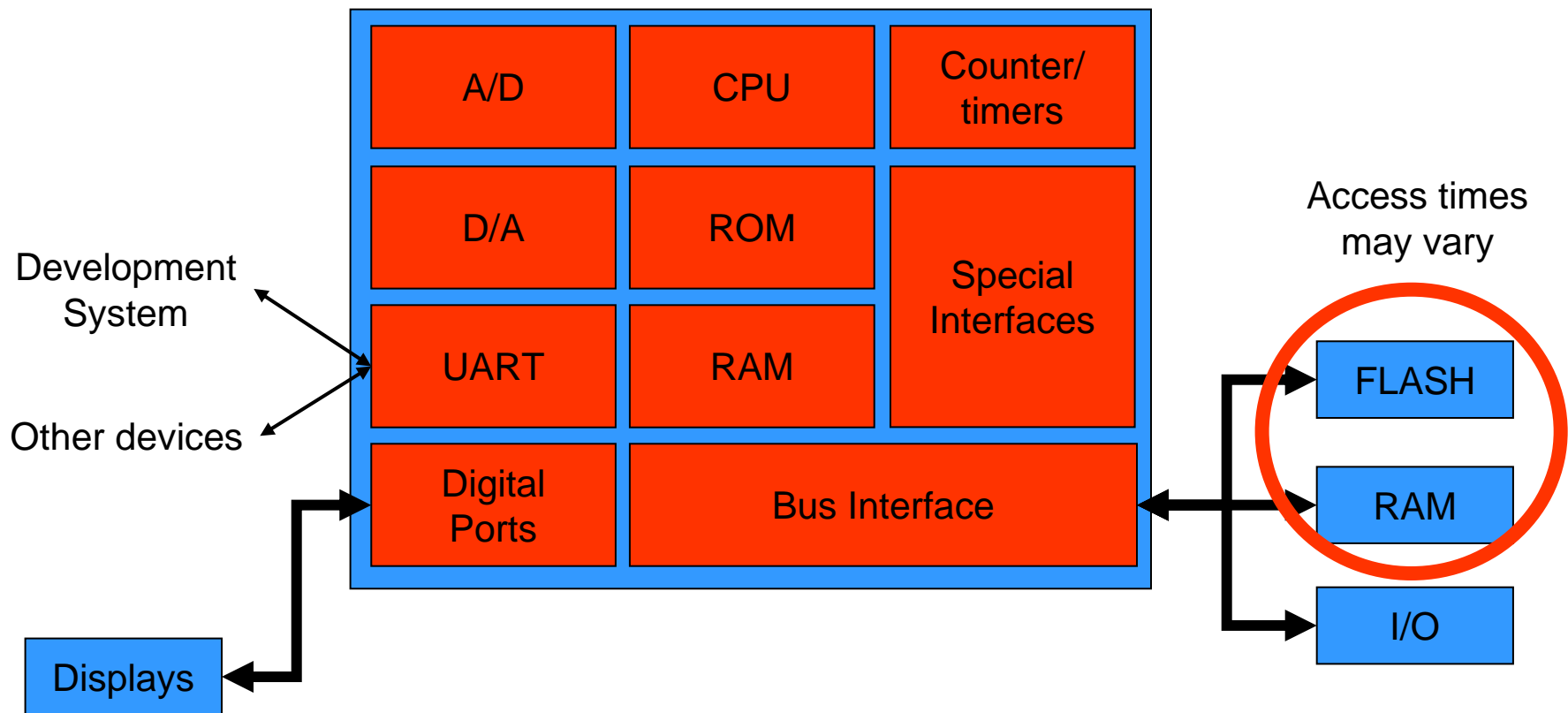
- Evaluating performance – a dynamically changing parameter



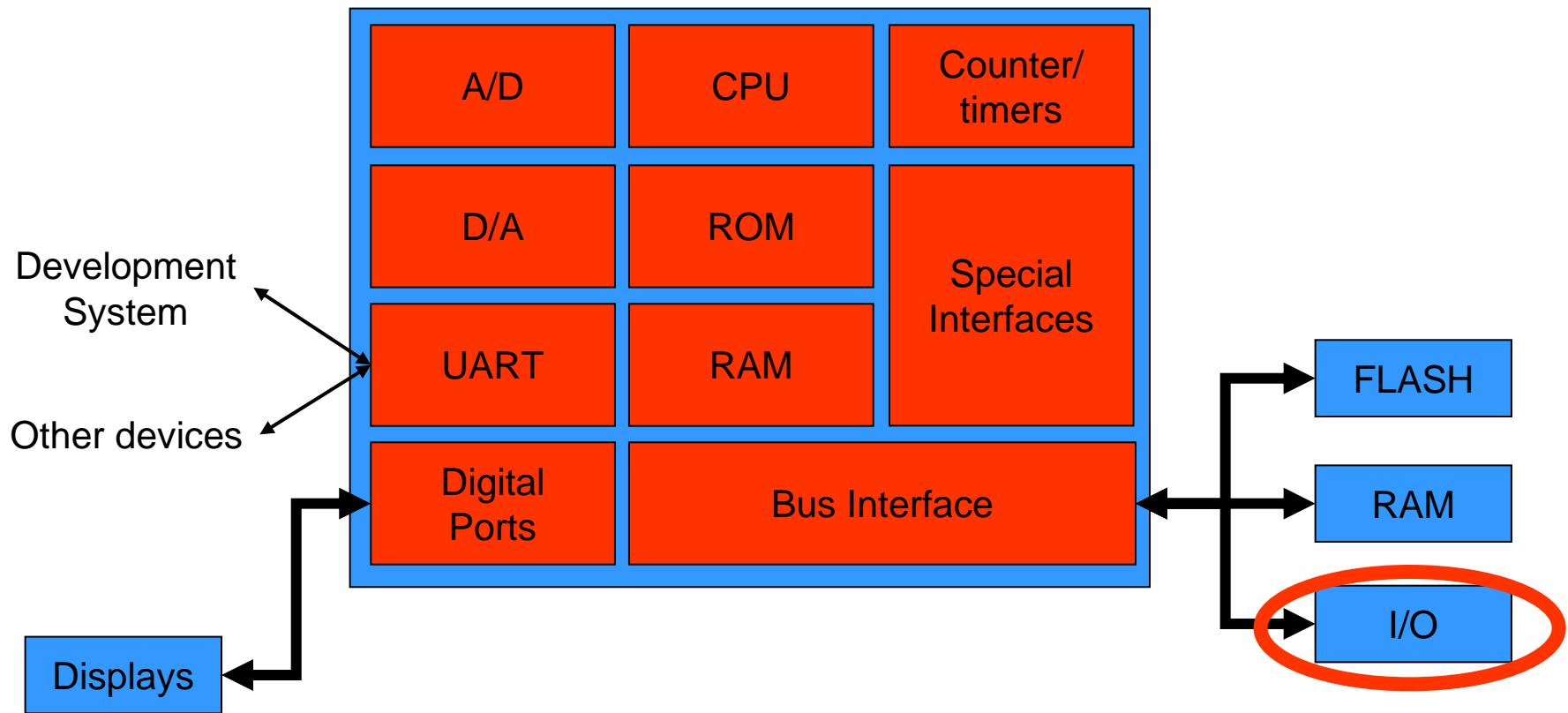
# A Representative Embedded System



# A Representative Embedded System



# A Representative Embedded System



Is I/O simulated for testing representative?

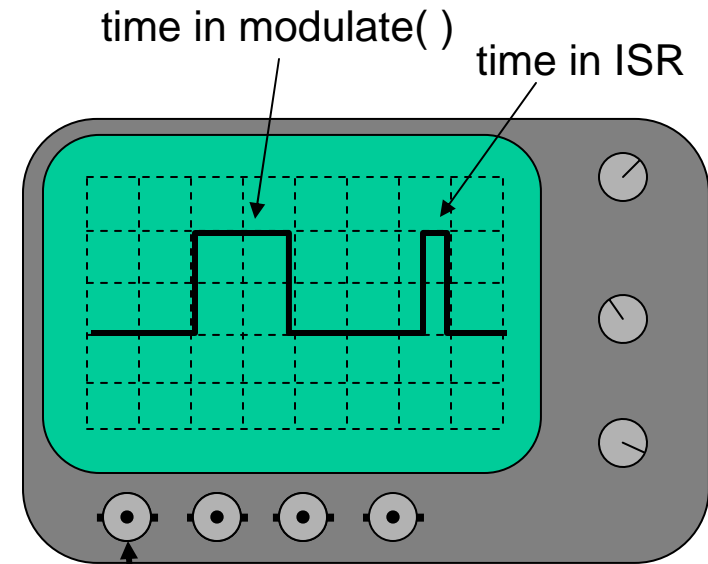
# Real-time Observability

- Observing time spent in ISR, other functions

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    set_up_interrupts(INTERRUPT_ON_DATA_PRESENT);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        set_output_flag(1);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        set_output_flag(0);
        output_mod(mod);
        sleep( );
    }
}

void interrupt_handler_DATA_PRESENT(void)
{
    set_output_flag(1);
    /* wake up main( ) when data arrives */
    set_output_flag(0);
}
```



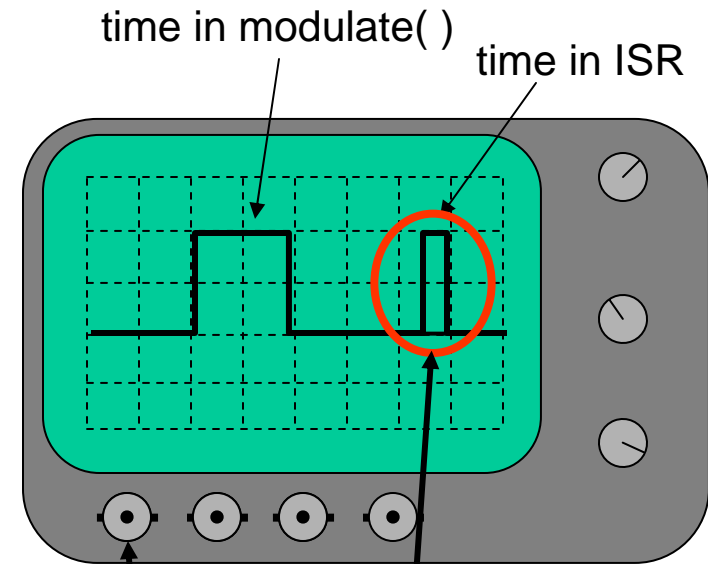
# Real-time Observability

- Observing time spent in ISR, other functions

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

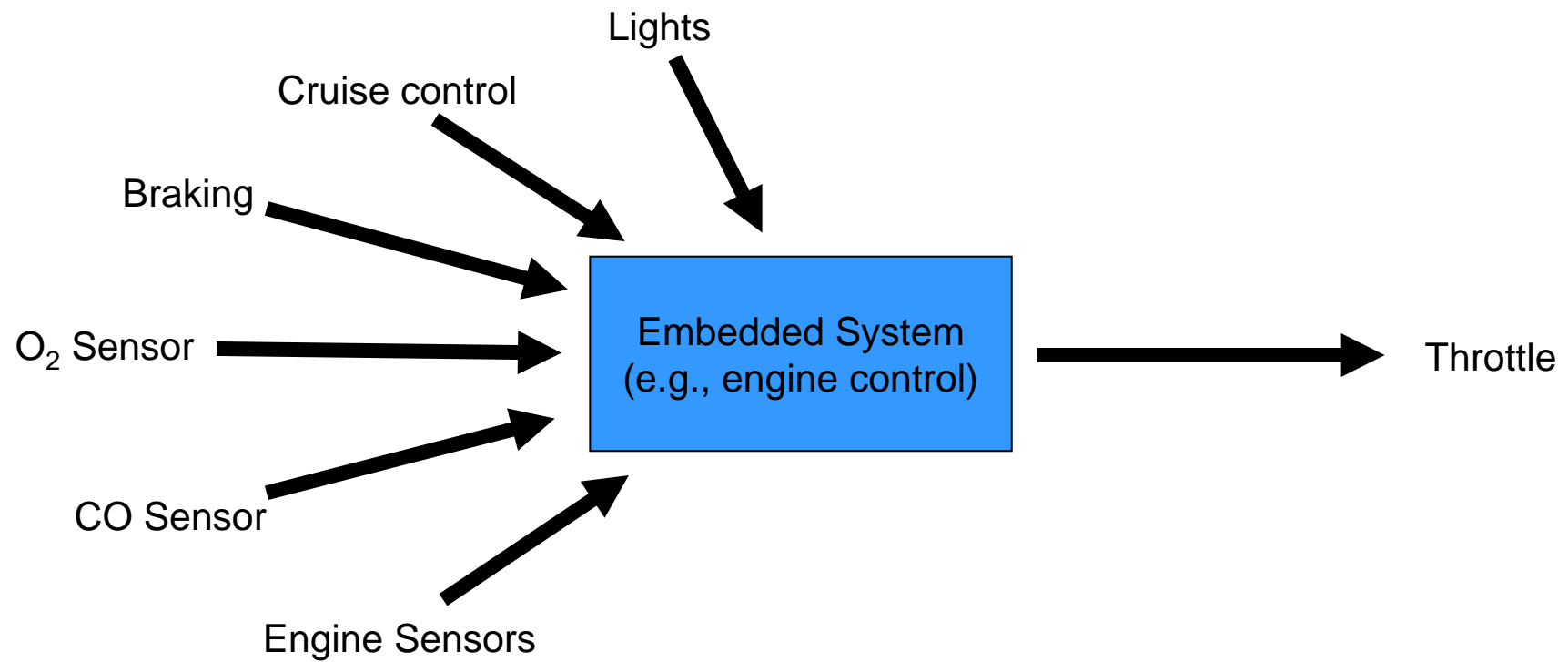
void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    set_up_interrupts(INTERRUPT_ON_DATA_PRESENT);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        set_output_flag(1);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        set_output_flag(0);
        output_mod(mod);
        sleep( );
    }
}

void interrupt_handler_DATA_PRESENT(void)
{
    set_output_flag(1);
    /* wake up main( ) when data arrives */
    set_output_flag(0);
}
```

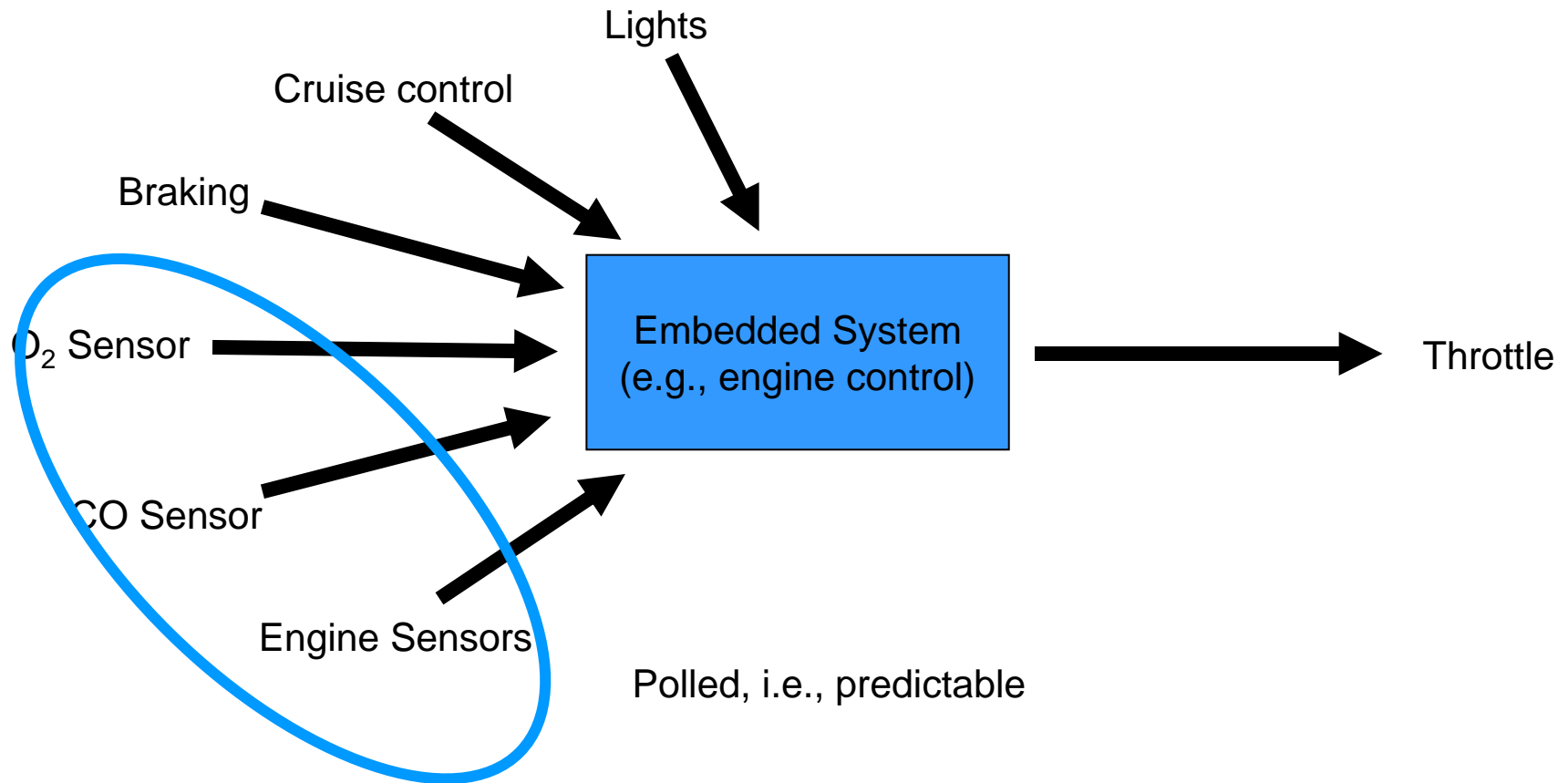


What if ISR is not called frequently?

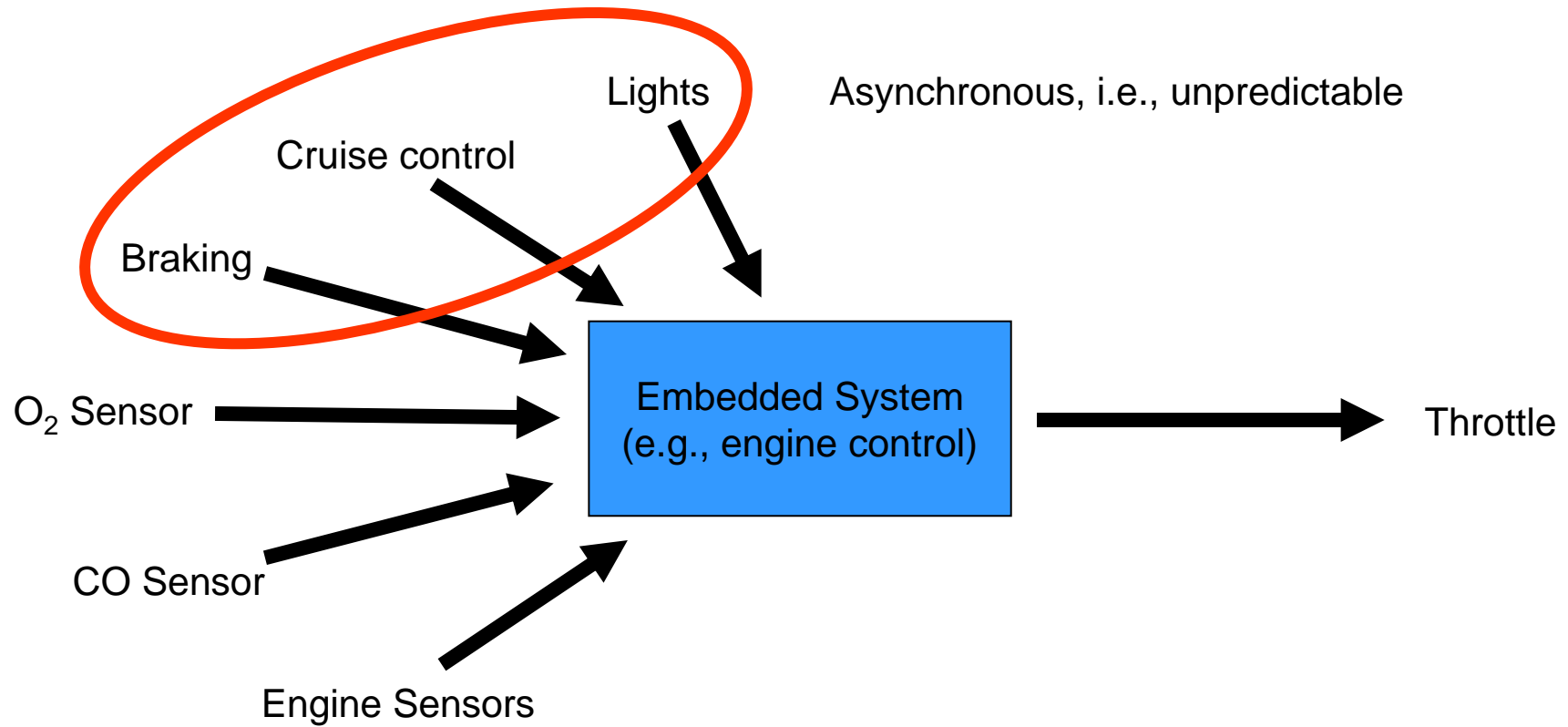
# Unpredictability of data, events



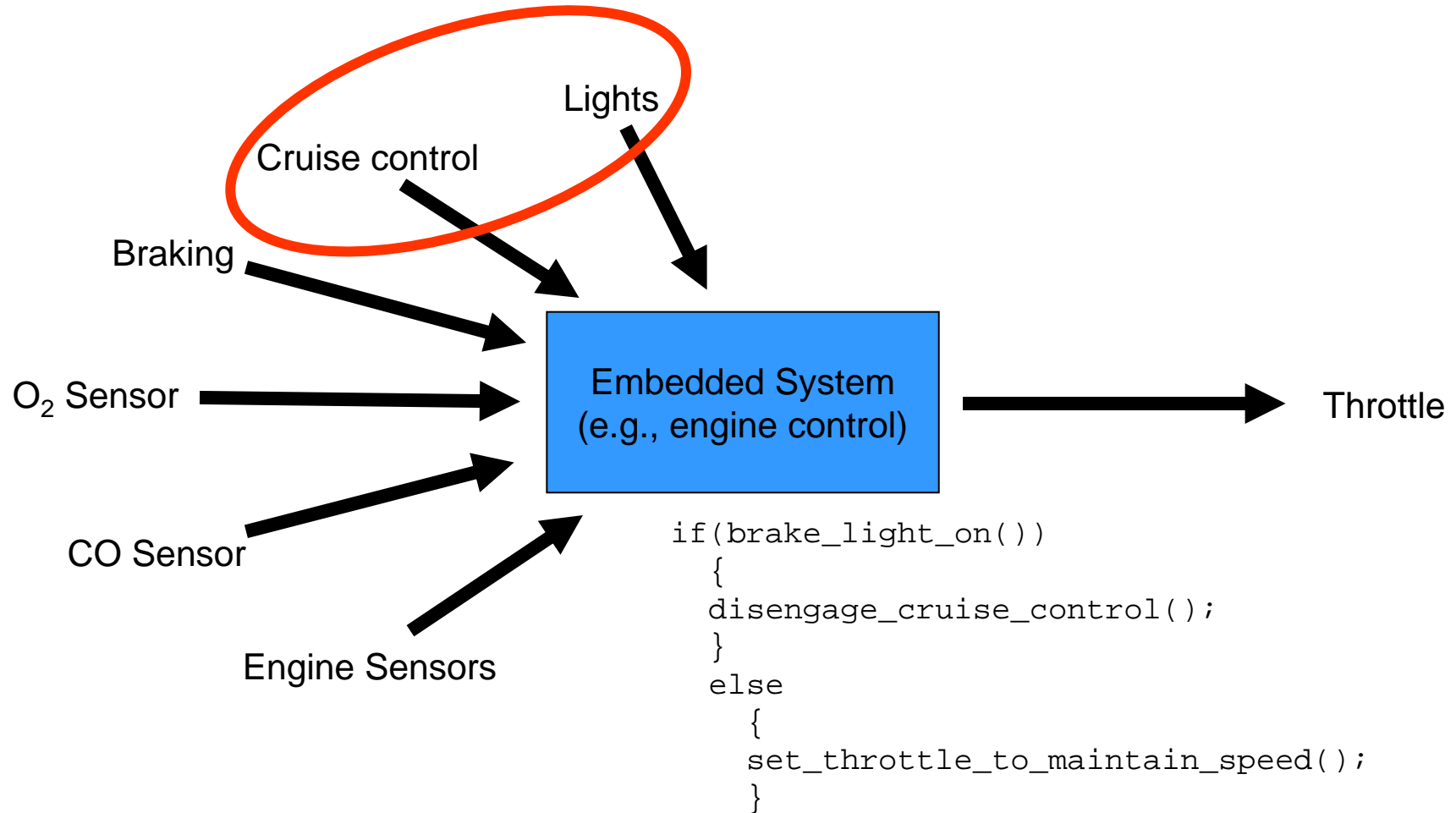
# Unpredictability of data, events



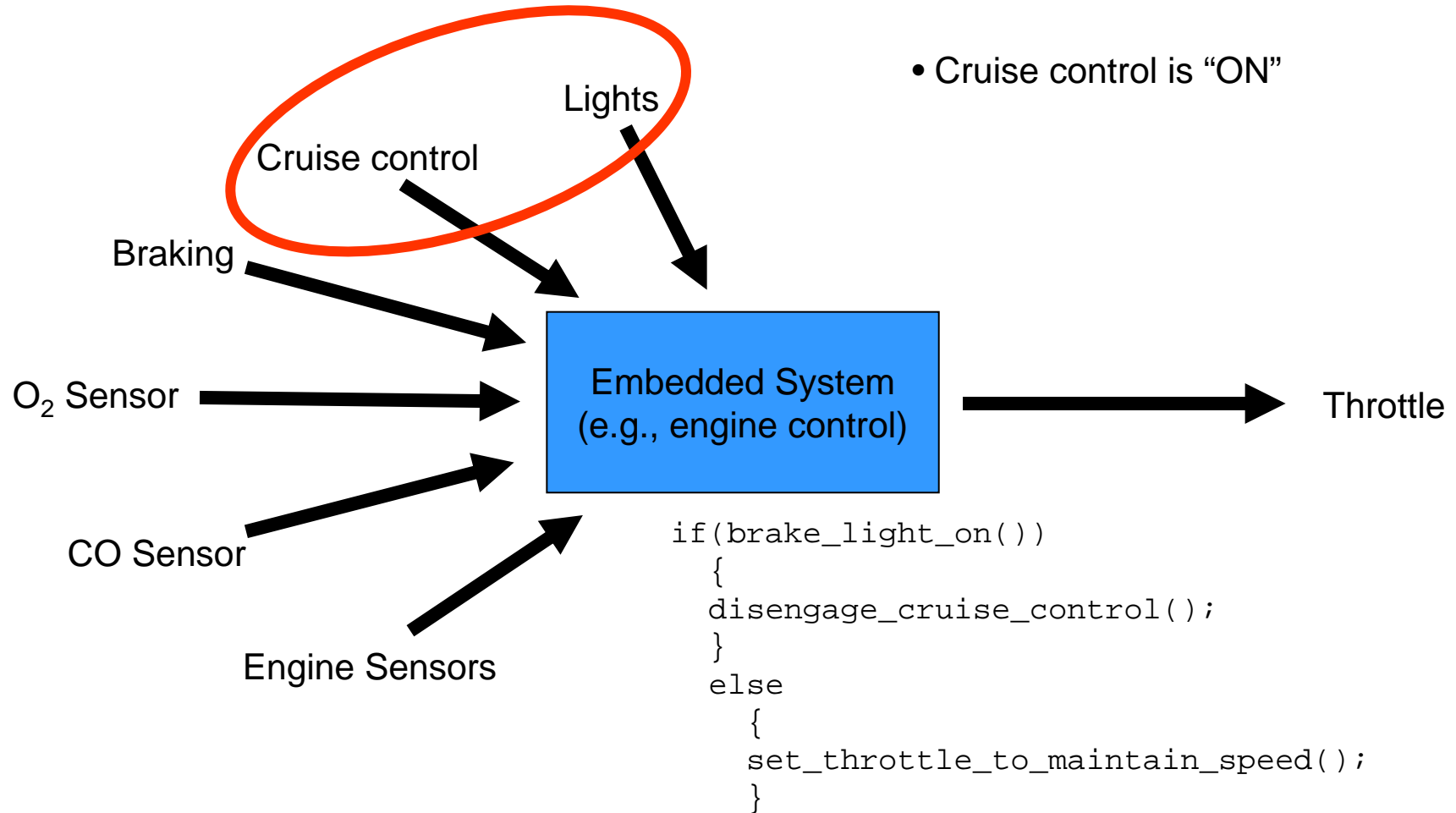
# Unpredictability of data, events



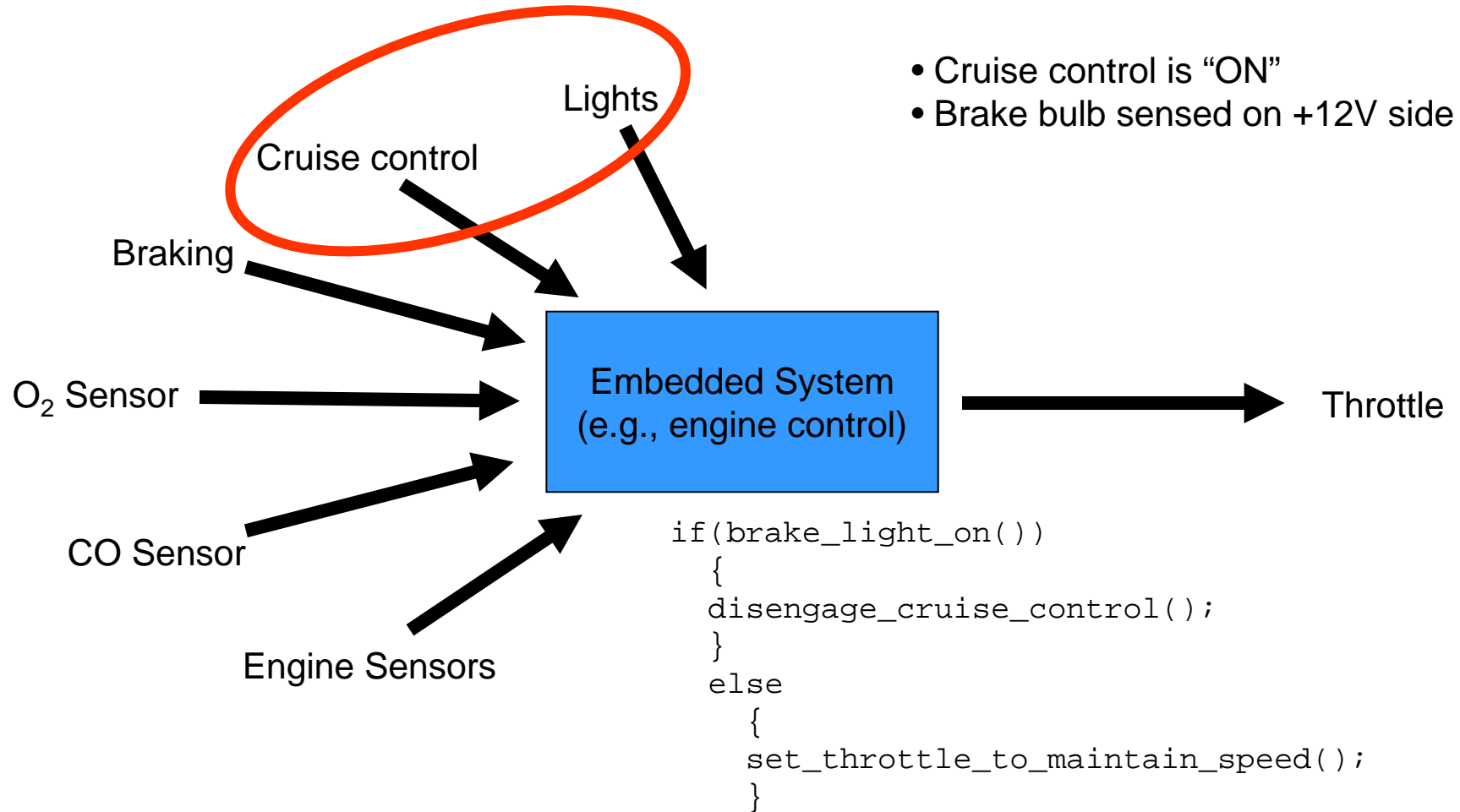
# Unanticipated interaction of data, events



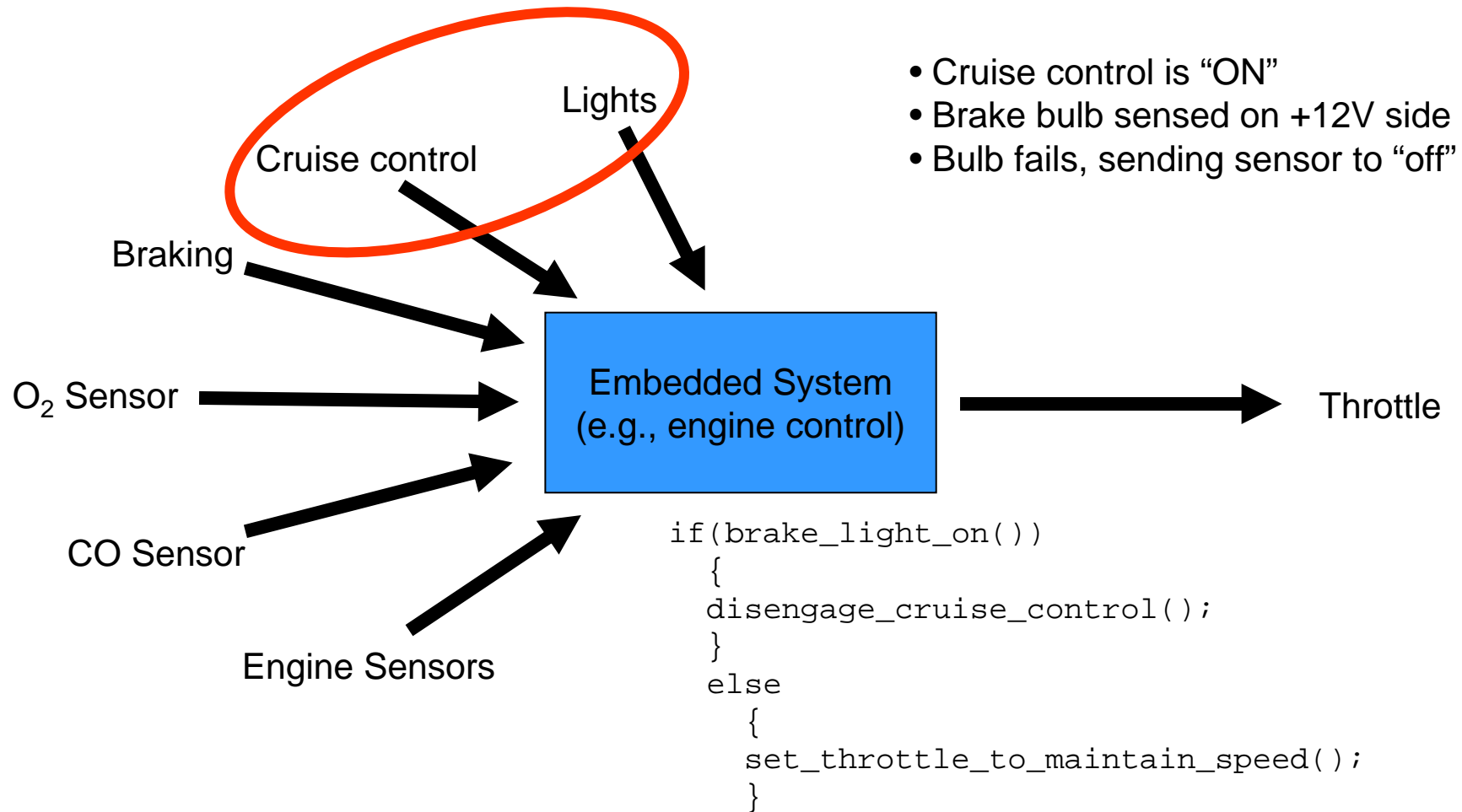
# Unanticipated interaction of data, events



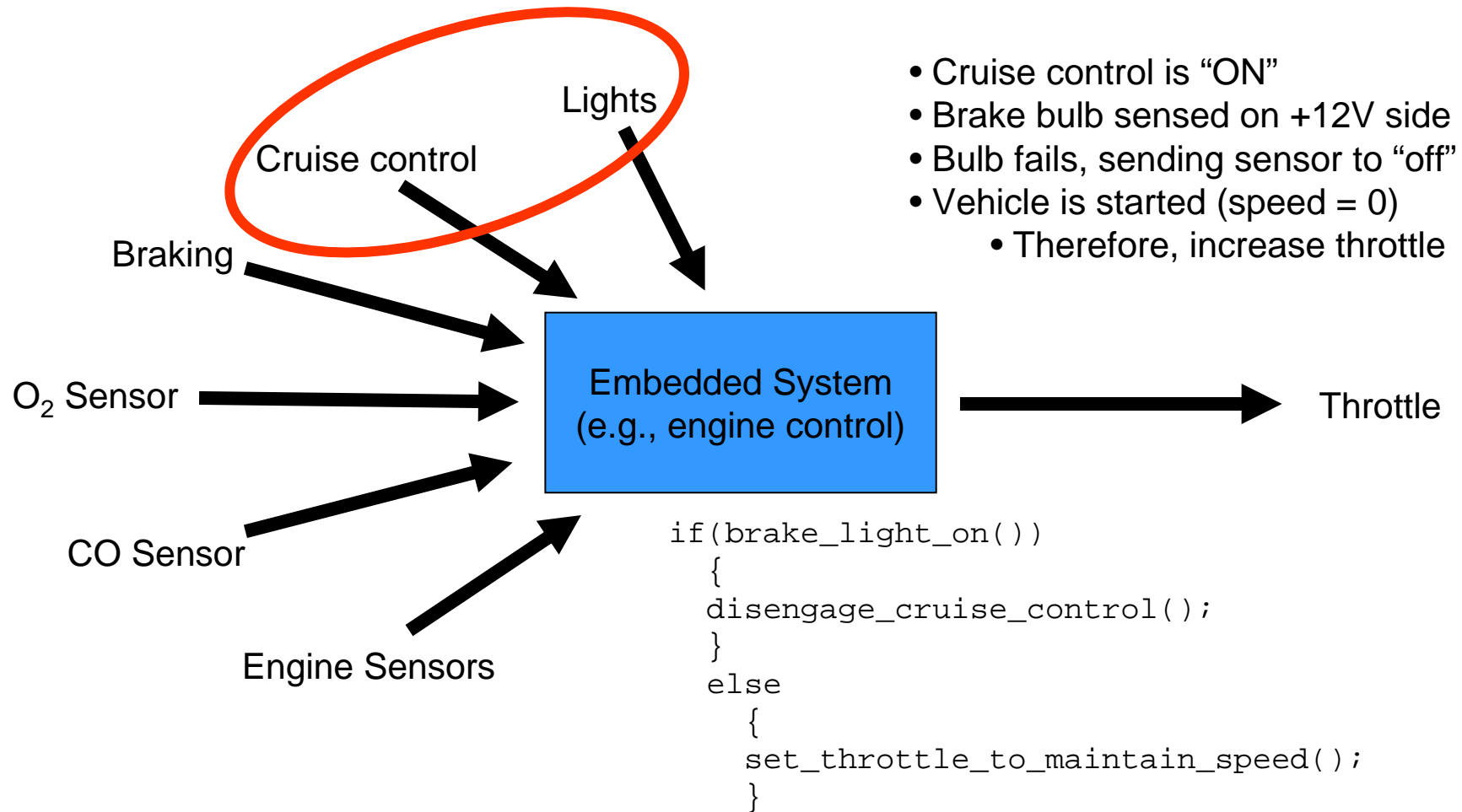
# Unanticipated interaction of data, events



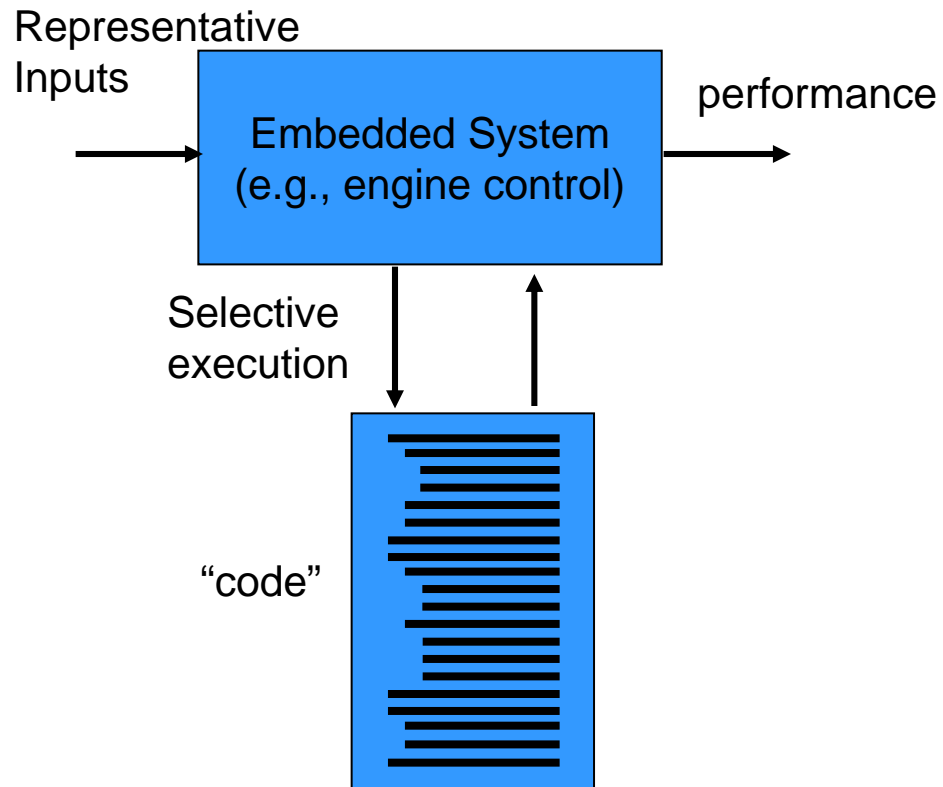
# Unanticipated interaction of data, events



# Unanticipated interaction of data, events

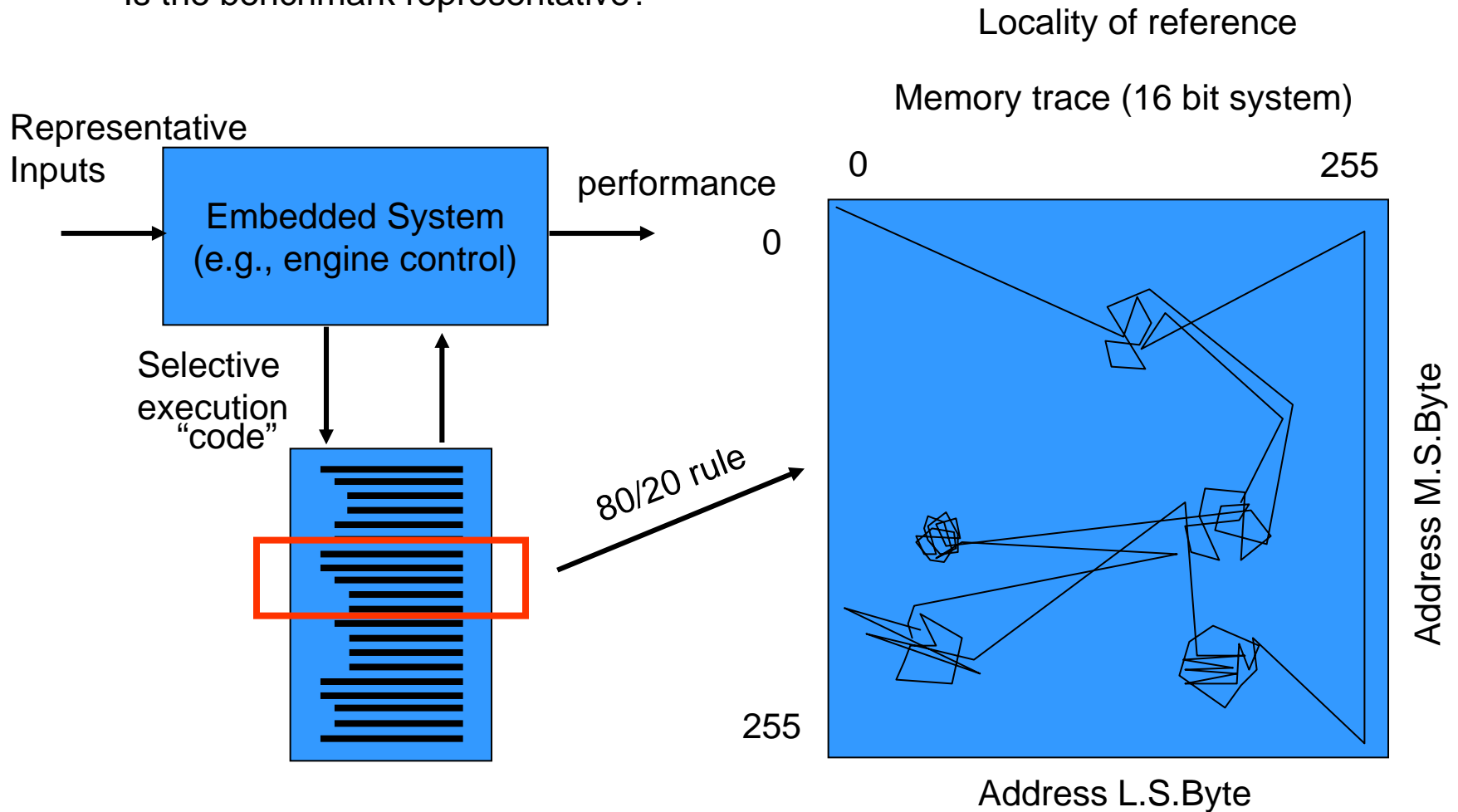


# Benchmarking performance

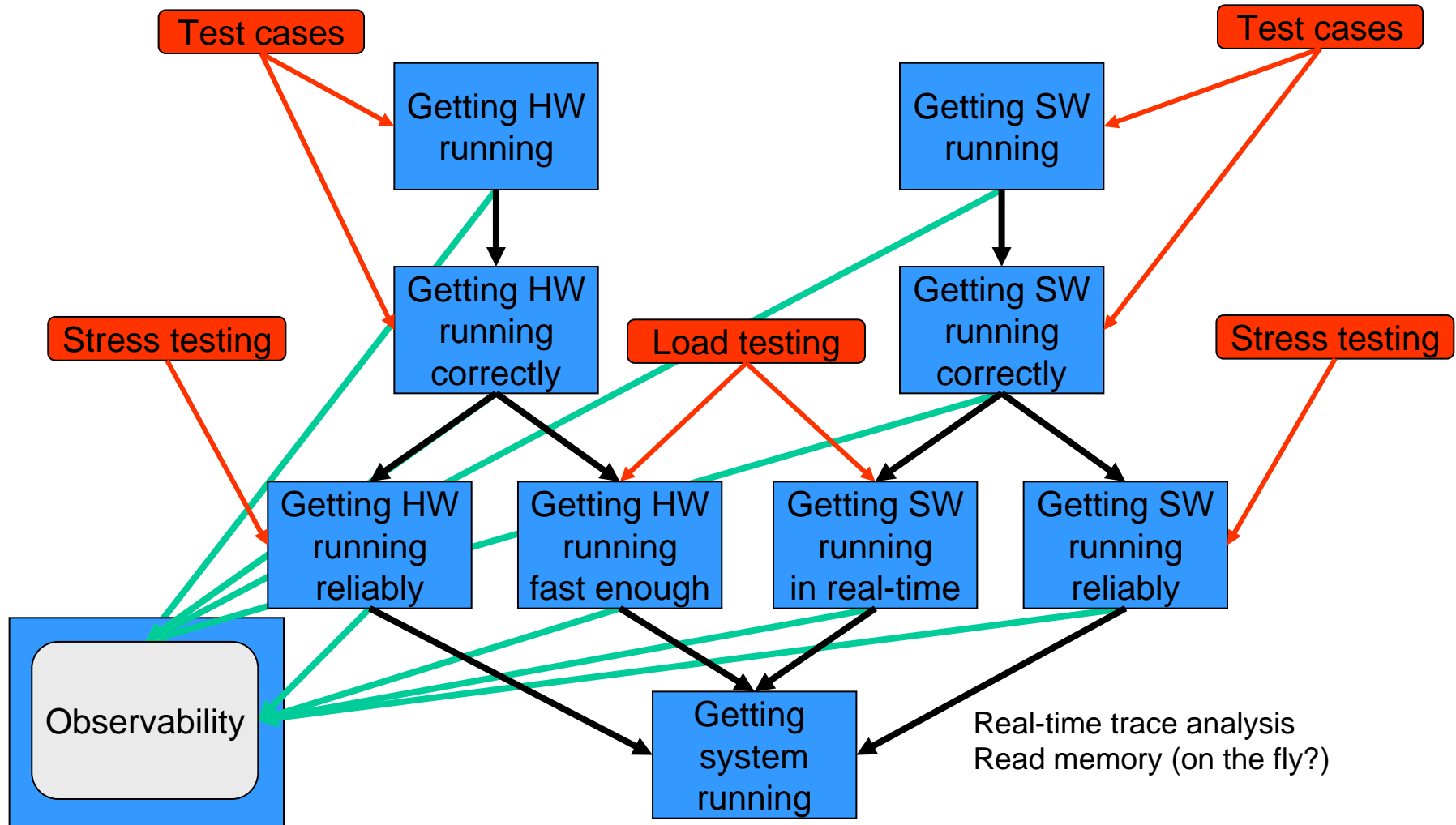


# Benchmarking performance

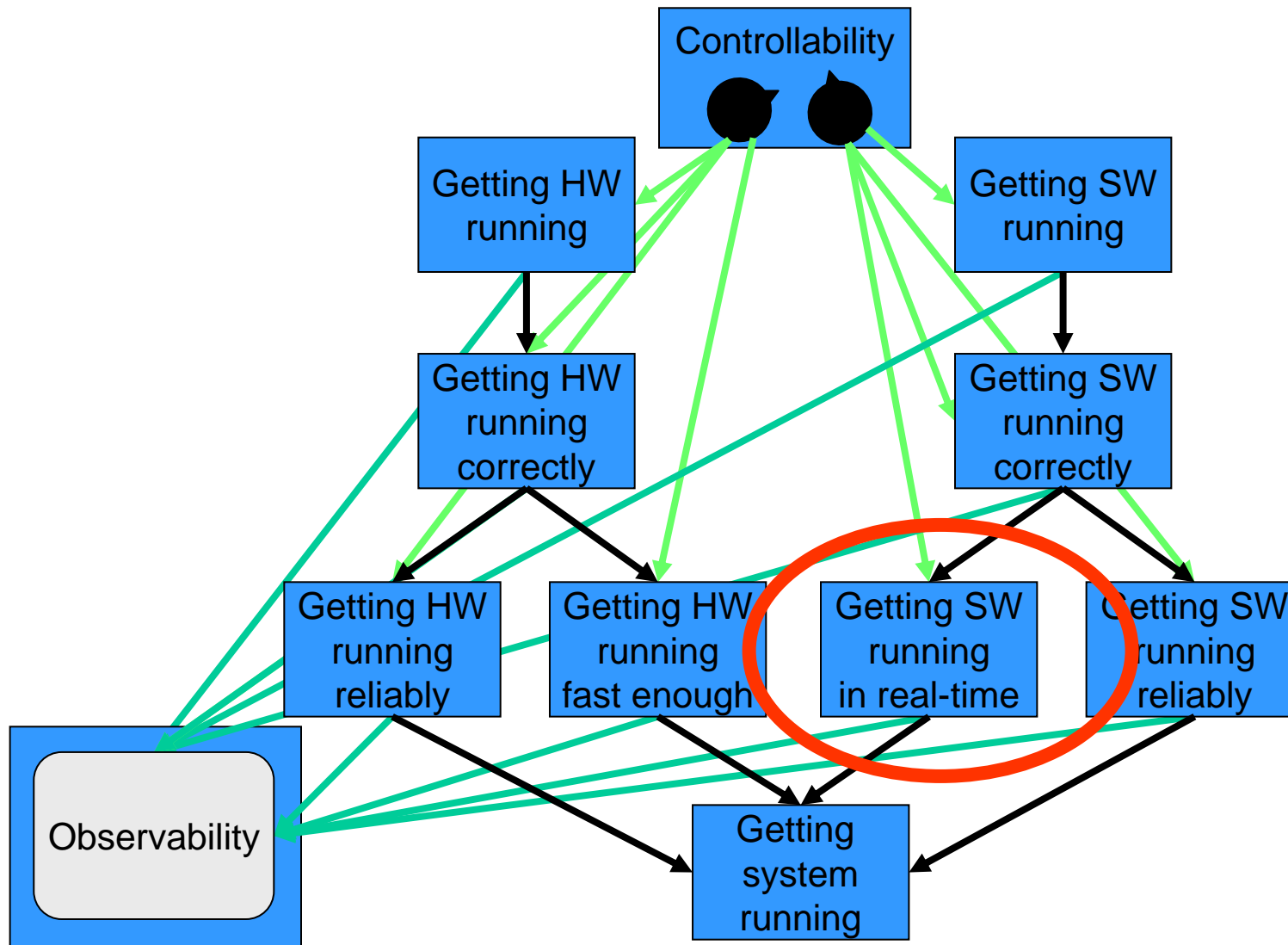
- Does everything get accessed?
- Is the benchmark representative?



# Getting a Real-Time Embedded System Running

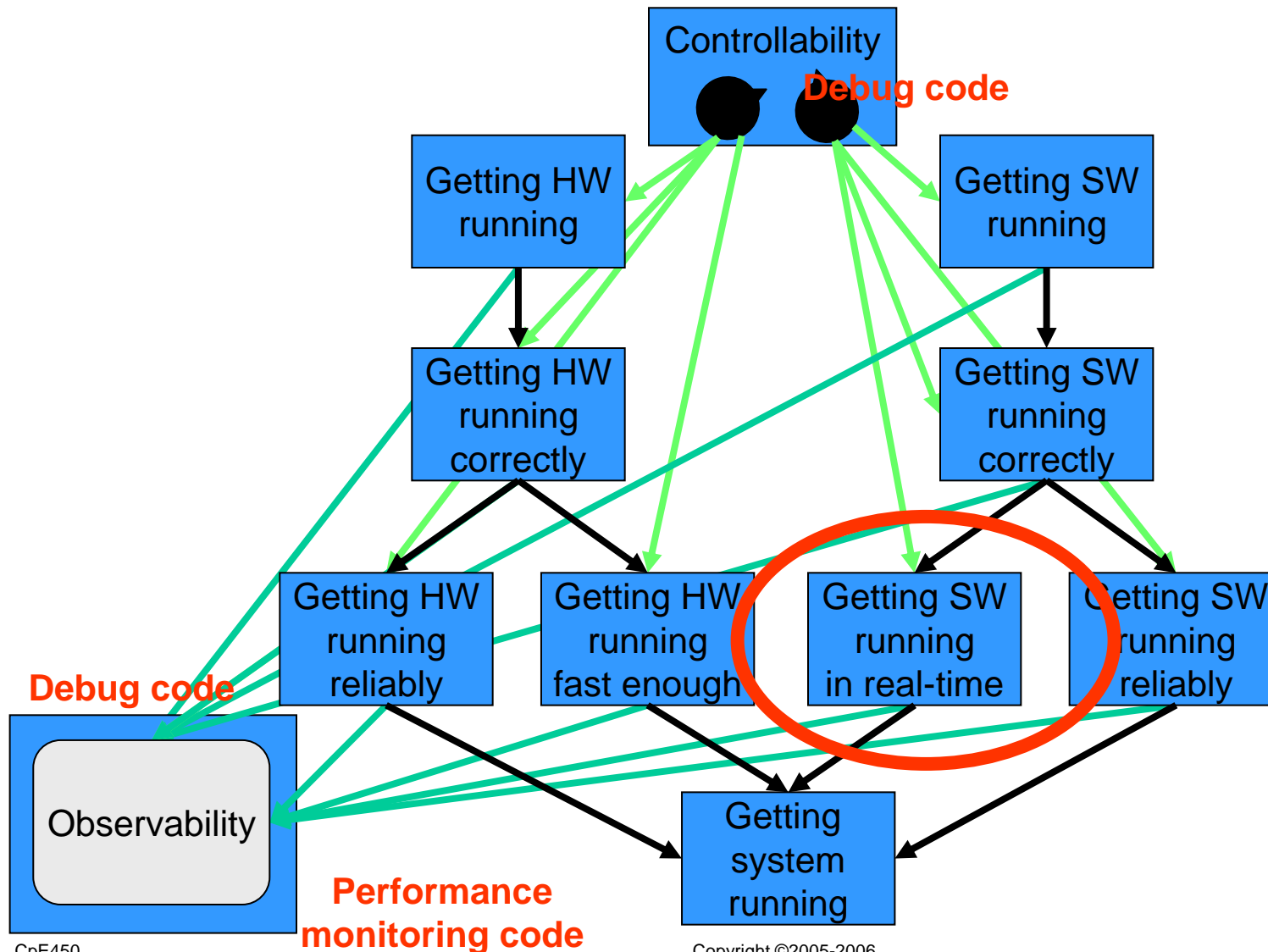


# Getting a Real-Time Embedded System Running

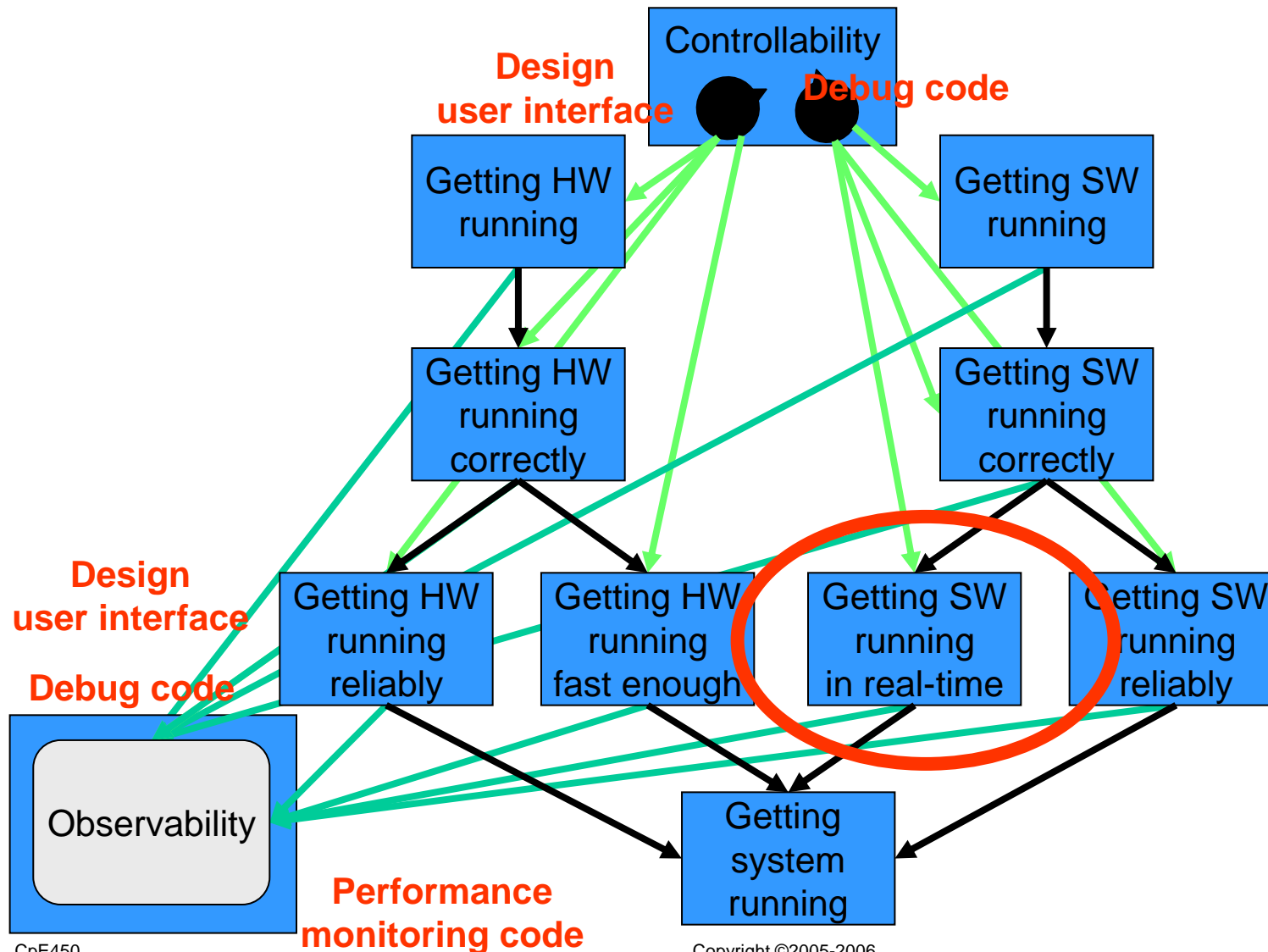




# Getting a Real-Time Embedded System Running



# Getting a Real-Time Embedded System Running



# Is Real-time *Monitoring* of Real-time *Performance* Necessary?

```
void main( )
{
  /* initialization code */
  while(1)
  {
    get_data(bits);
    generate_baseband(bits, sig);
    set_output_flag(1); ←
    modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
    set_output_flag(0); ←
    output_mod(mod);
    sleep( );
  }
}
```

Real-time observation

# Is Real-time *Monitoring* of Real-time *Performance* Necessary?

```
void main( )
{
  /* initialization code */
  start_clock();
  while(1)
  {
    t_get -= clock();
    get_data(bits);
    t_get += clock();
    t_generate -= clock();
    generate_baseband(bits, sig);
    t_generate += clock();
    /* set_output_flag(1); */
    t_mod -= clock
    modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
    t_mod += clock();
    /* set_output_flag(0); */
    t_out -= clock();
    output_mod(mod);
    t_out += clock();
    sleep( );
  }
}
```

# Is Real-time *Monitoring* of Real-time *Performance* Necessary?

```
void main( )
{
  /* initialization code */
  start_clock();
  while(1)
  {
    t_get -= clock();
    get_data(bits);
    t_get += clock();
    t_generate -= clock();
    generate_baseband(bits, sig);
    t_generate += clock();
    /* set_output_flag(1); */
    t_mod -= clock();
    modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
    t_mod += clock();
    /* set_output_flag(0); */
    t_out -= clock();
    output_mod(mod);
    t_out += clock();
    sleep( );
    /* periodically report the values of the t_ values */
  }
}
```

t\_get accumulates the total  
time spent in get\_data( )

# Assignment 6

- Using either the code presented in last class' slides or other code available to you (written in any suitable language):
  - Modify the code run on your PC. E.g., you would need a main( ) program to call the modulate routine
  - Set up the code to allow you to time execution of the various sections of the code, e.g., write a routine to monitor system time and determine the execution time of code segments.
  - Try to verify the effectiveness of the code profiling methods presented this week (i.e., the Pareto Principle), modifying the code to improve execution time. Note: Larger, more complex programs will make it easier to profile and optimize.