

Real-Time Embedded Systems

CpE-450 Spring 07

Class 4

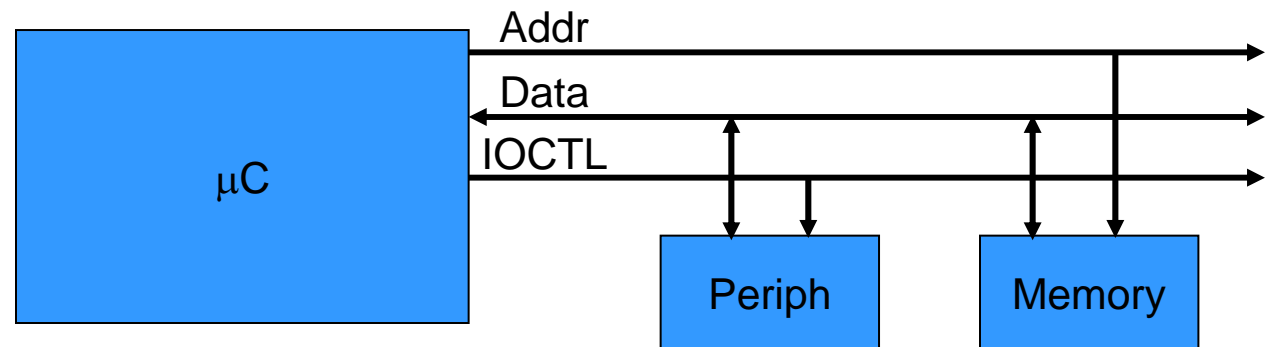
Bruce McNair

bmcnair@stevens.edu

Memory Mapped I/O vs. I/O Ports

- I/O Ports

LDA ADDR :Get data at ADDR to A
OUT PORT :Output data from A reg
.
.
IN PORT :Load port to A
STA ADDR :Store data



Memory Mapped I/O vs. I/O Ports

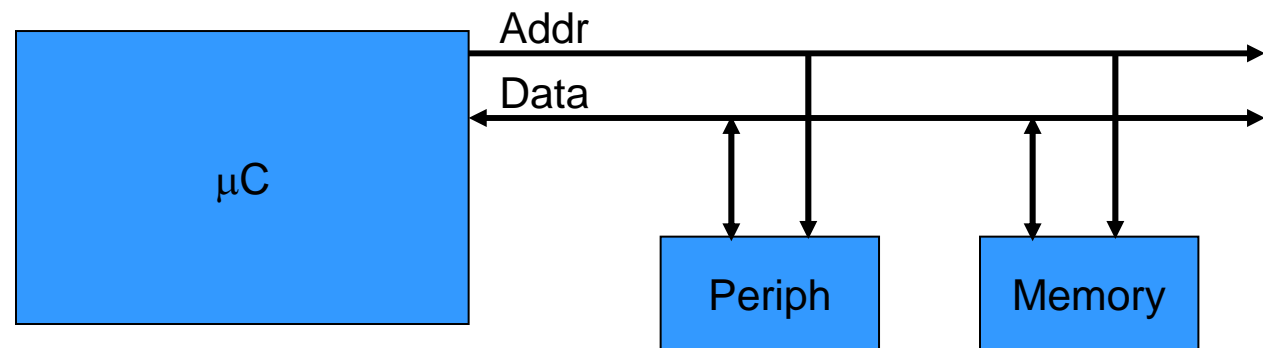
- Memory Mapped I/O

LDA ADDR :Load A from ADDR

.

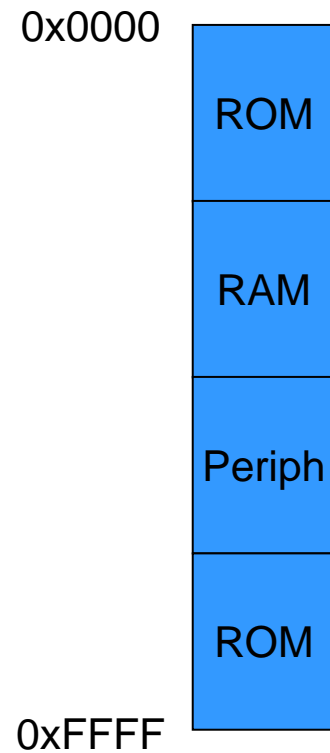
.

STA ADDR :Write A to ADDR



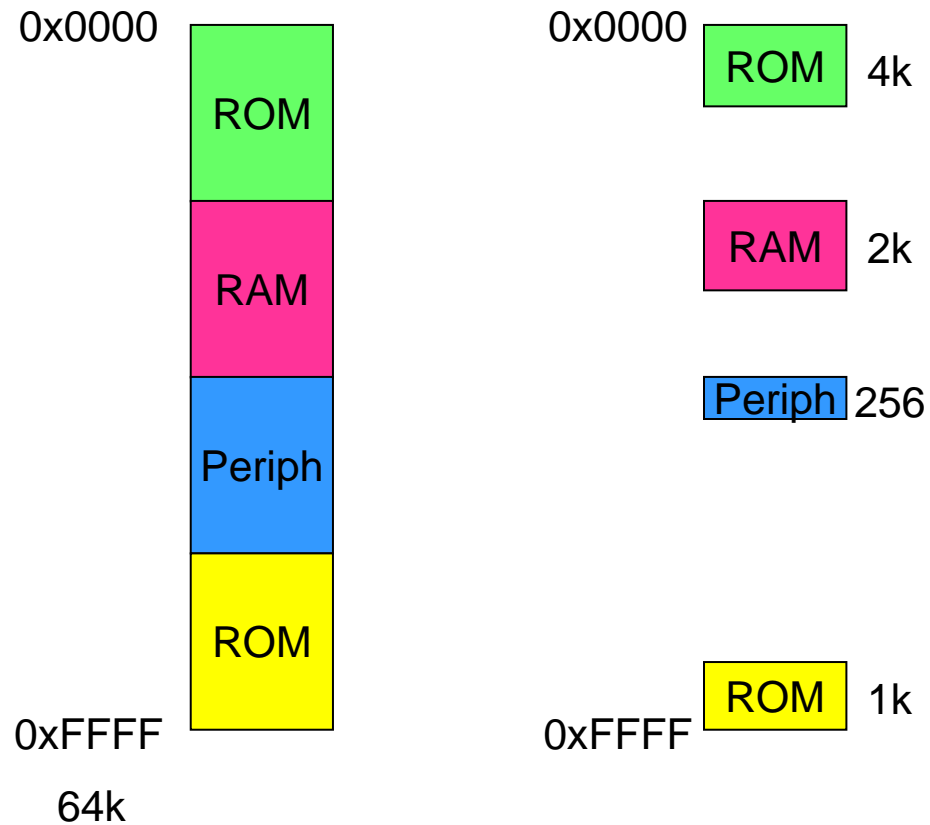
Architectural Implications of Limited Hardware

- Example memory map:



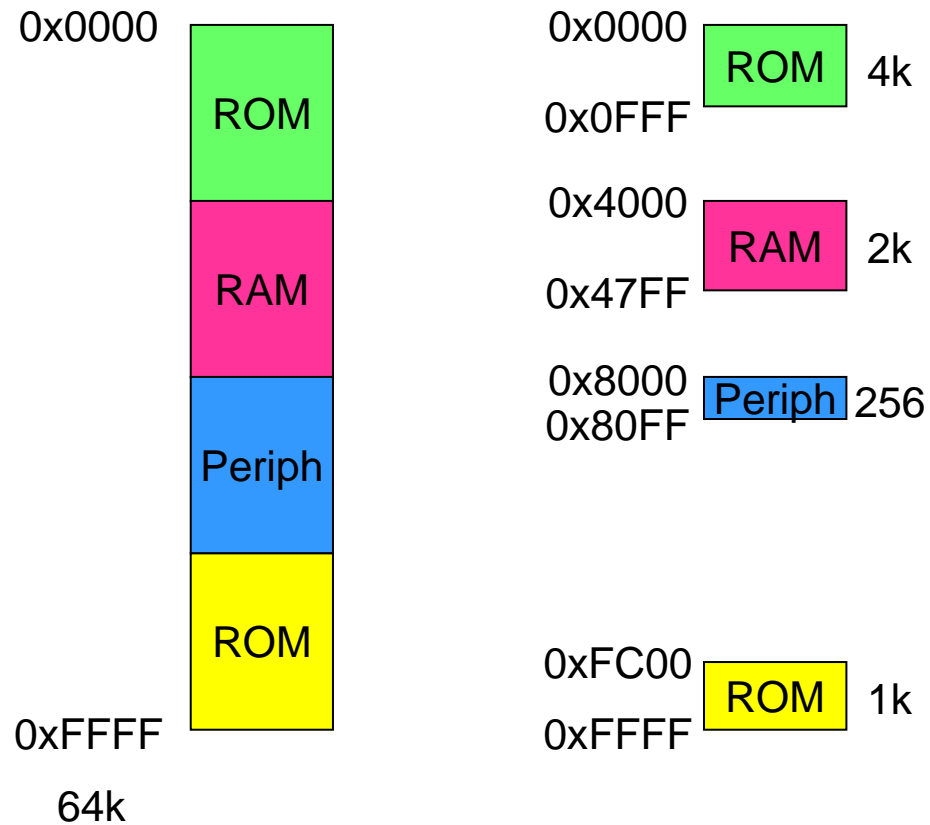
Architectural Implications of Limited Hardware

- Example memory map:



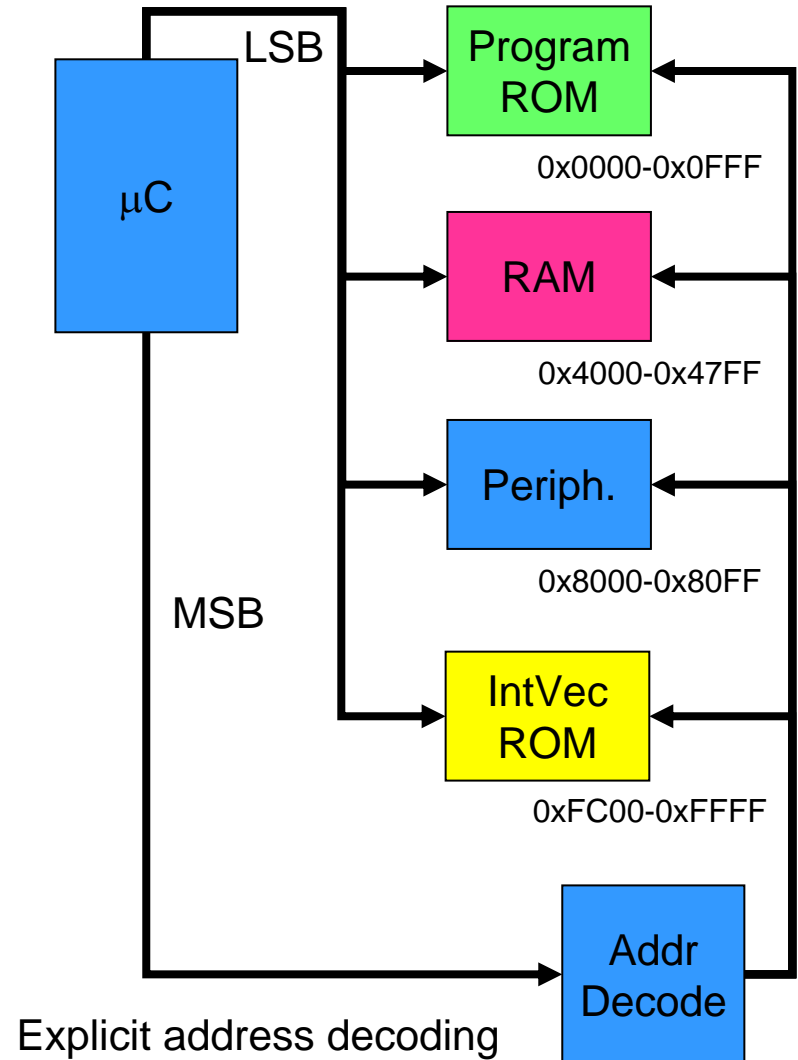
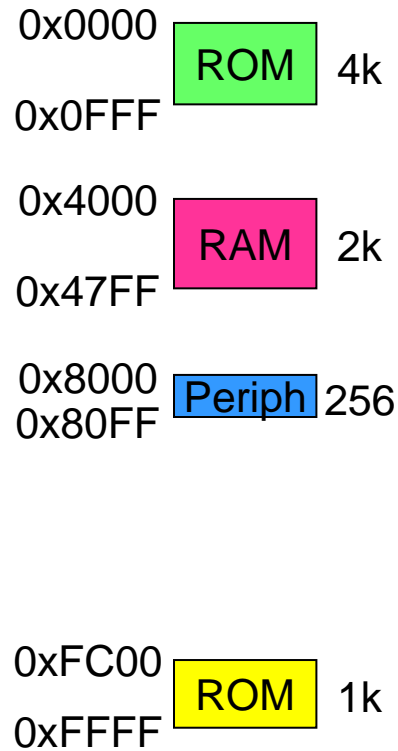
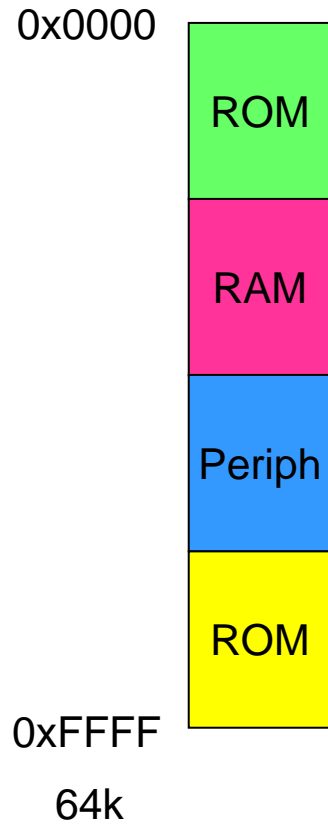
Architectural Implications of Limited Hardware

- Example memory map:



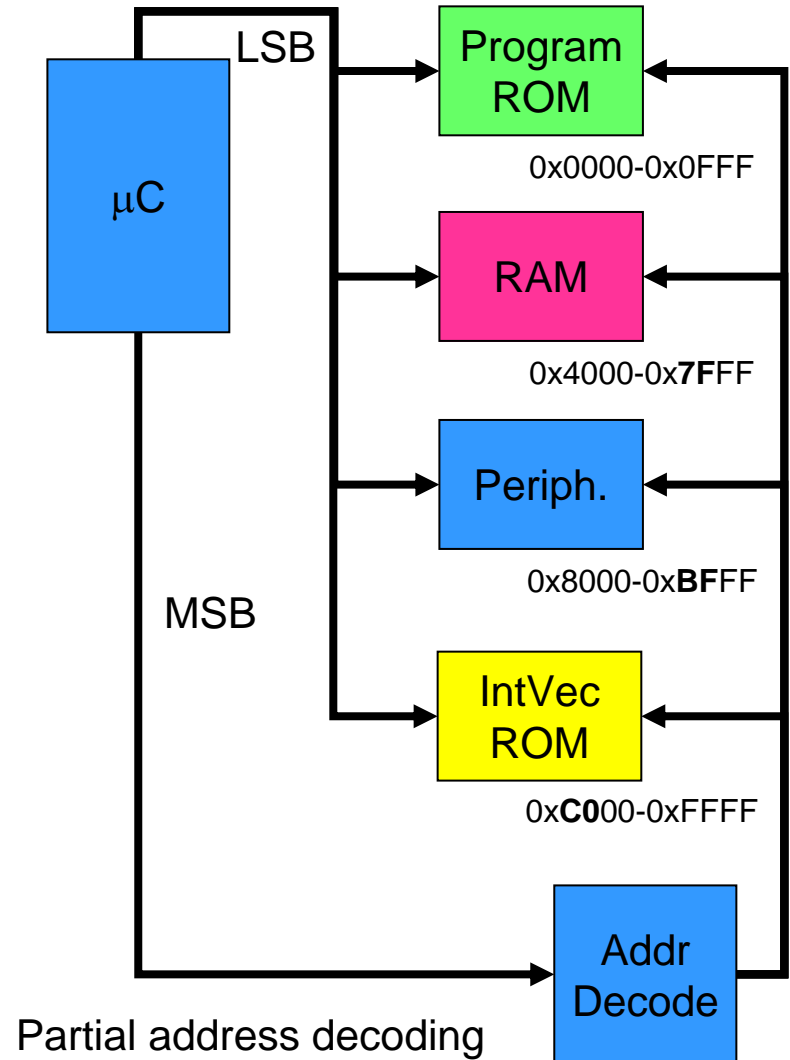
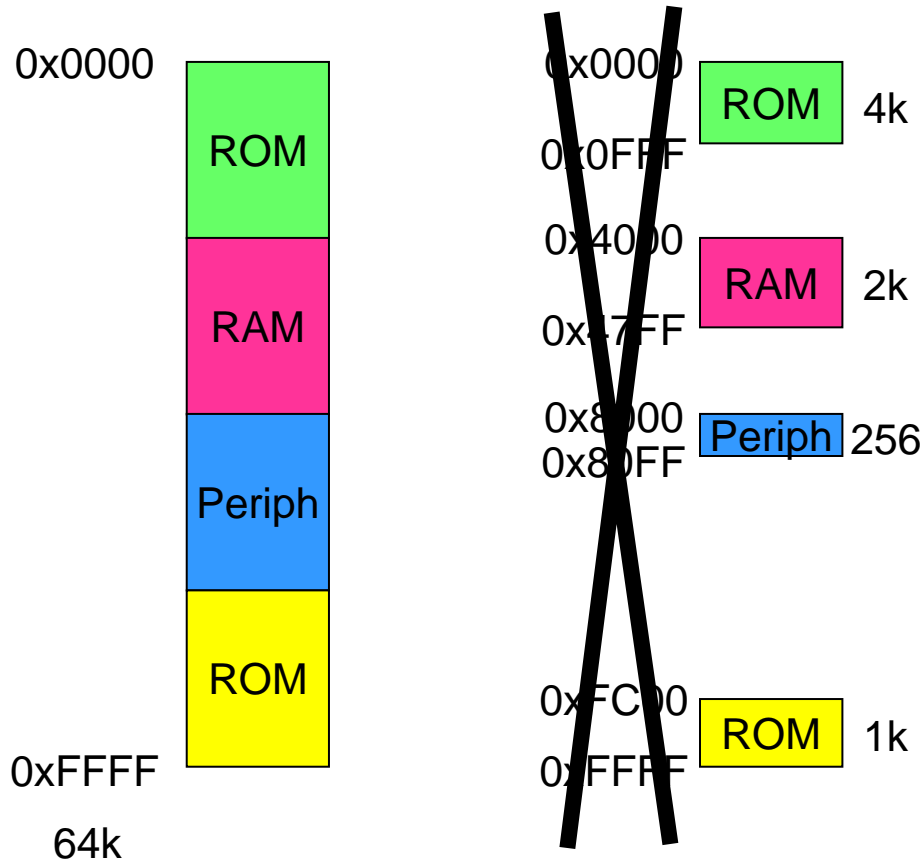
Architectural Implications of Limited Hardware

- Example memory map:



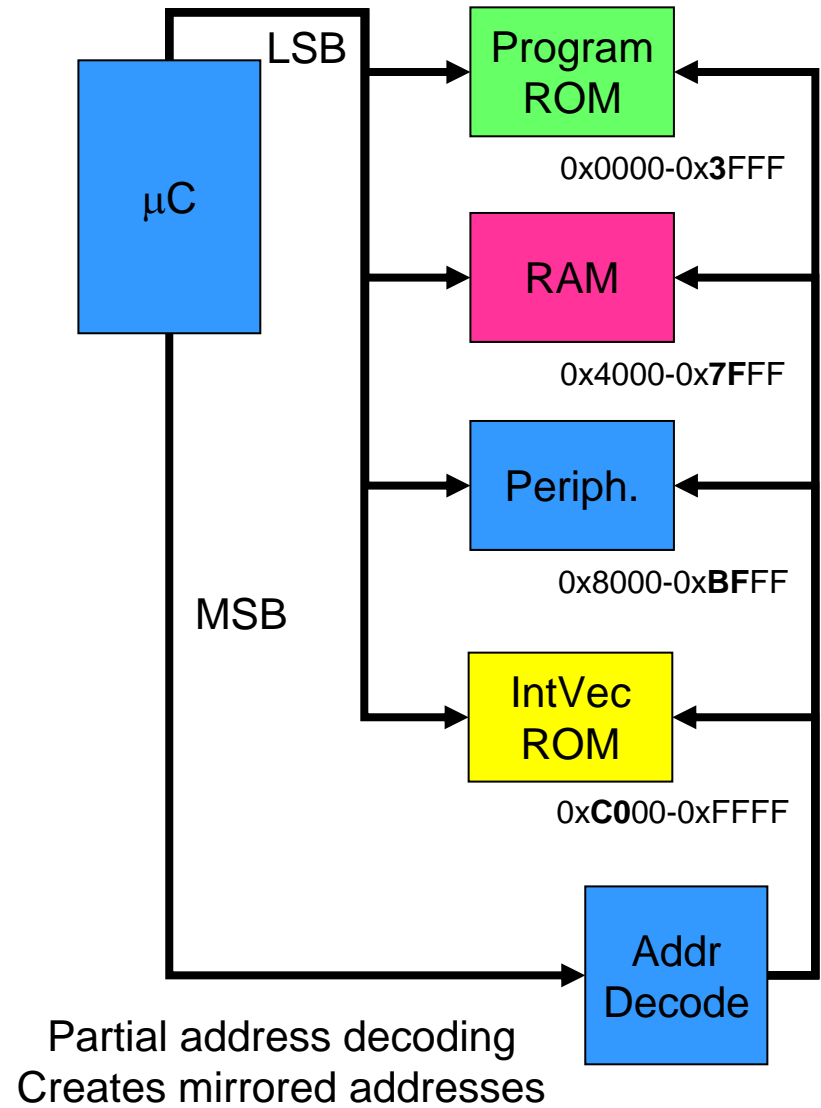
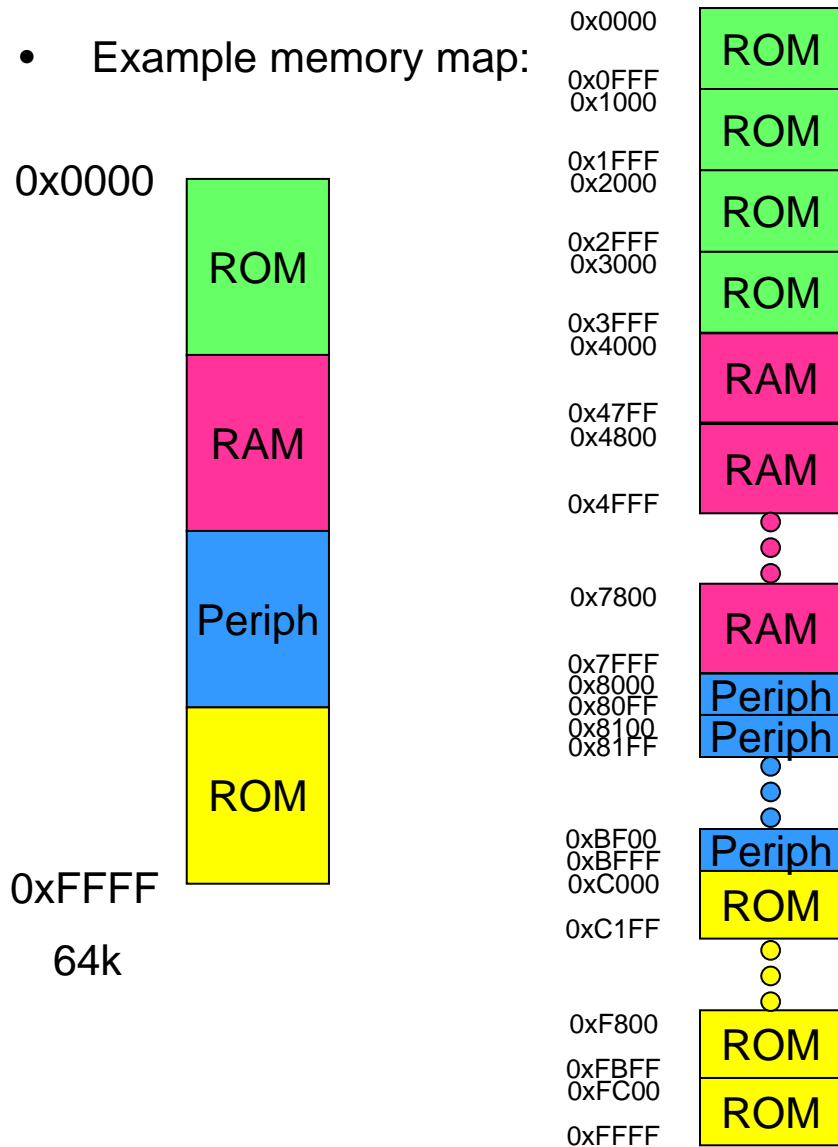
Architectural Implications of Limited Hardware

- Example memory map:

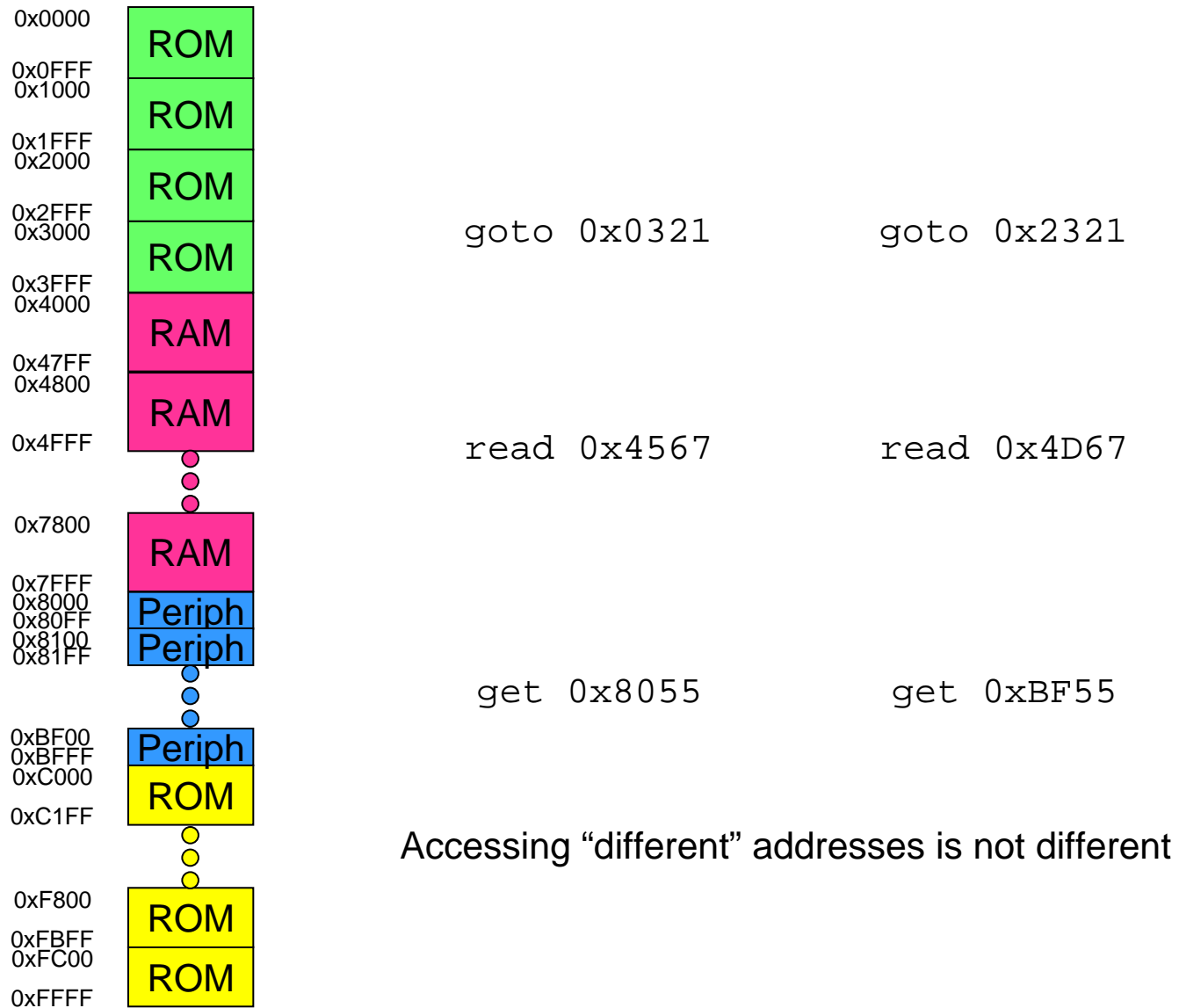


Architectural Implications of Limited Hardware

- Example memory map:

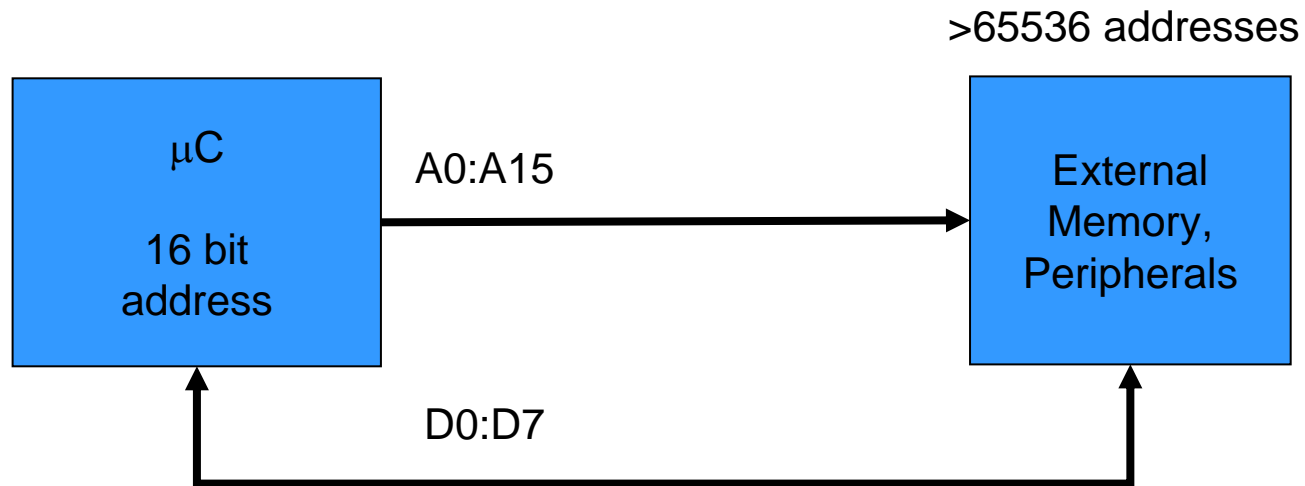


Effects of Mirrored Memory Addresses



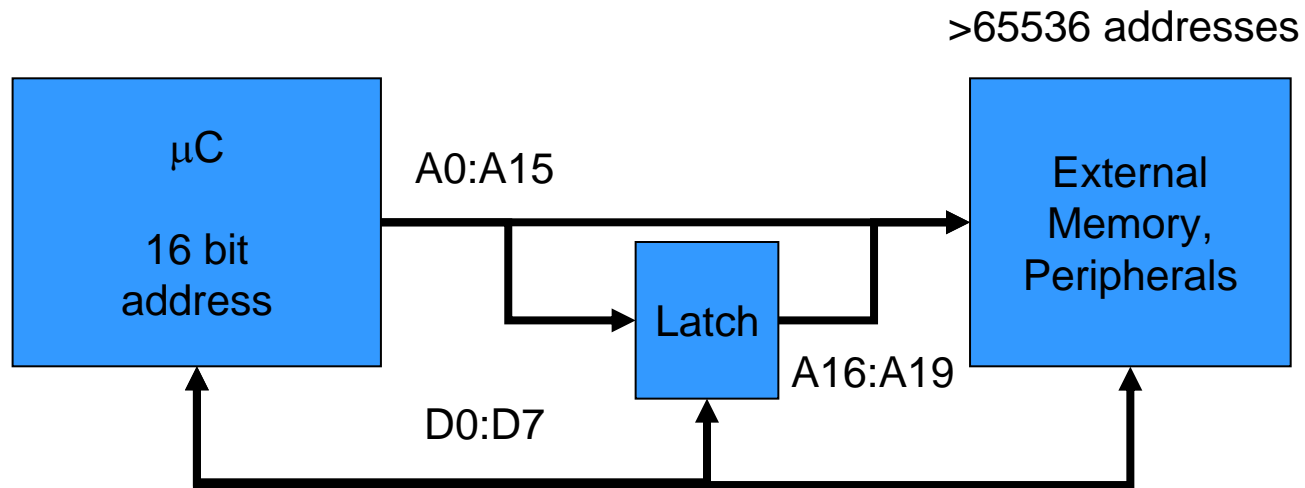
Dealing with Insufficient Address Space

- Programs, RAM, and peripherals may require additional address space



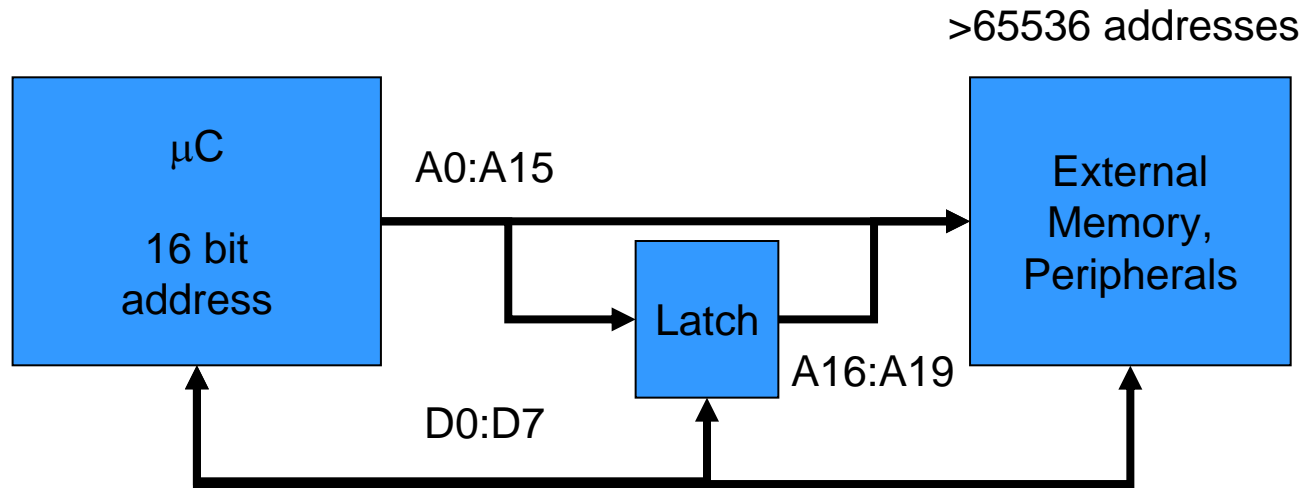
Dealing with Insufficient Address Space

- Programs, RAM, and peripherals may require additional address space
- Use bank selection



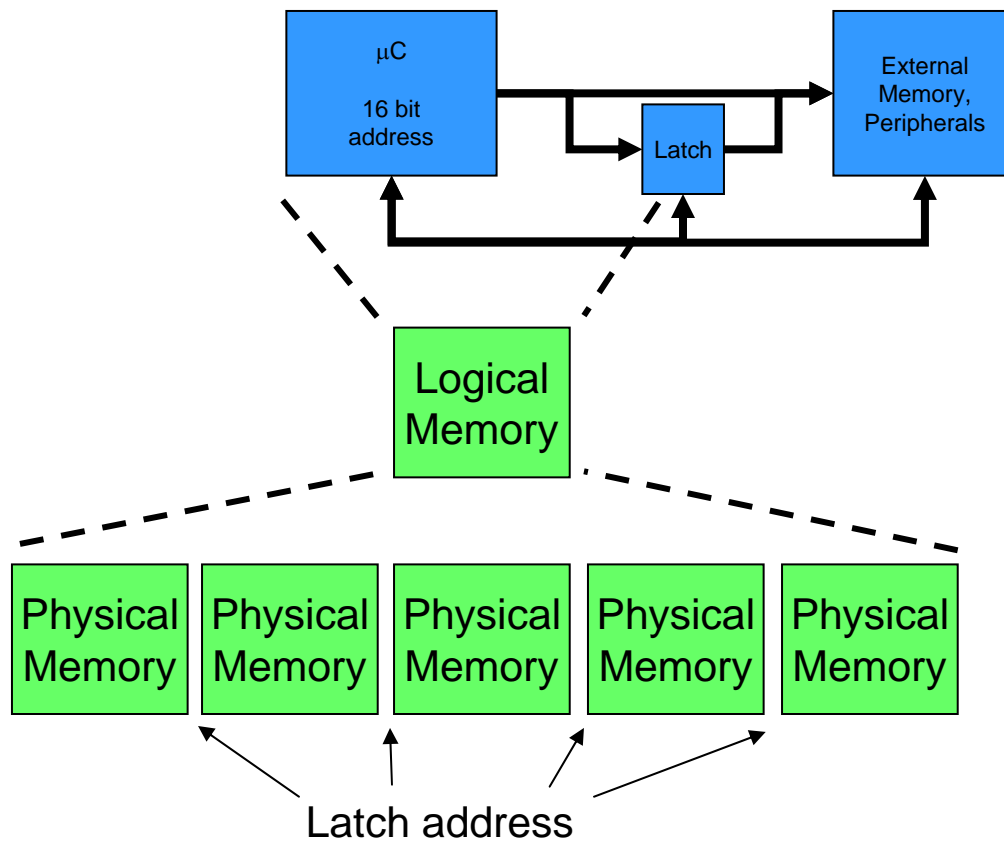
Dealing with Insufficient Address Space

- Programs, RAM, and peripherals may require additional address space
- Use bank selection

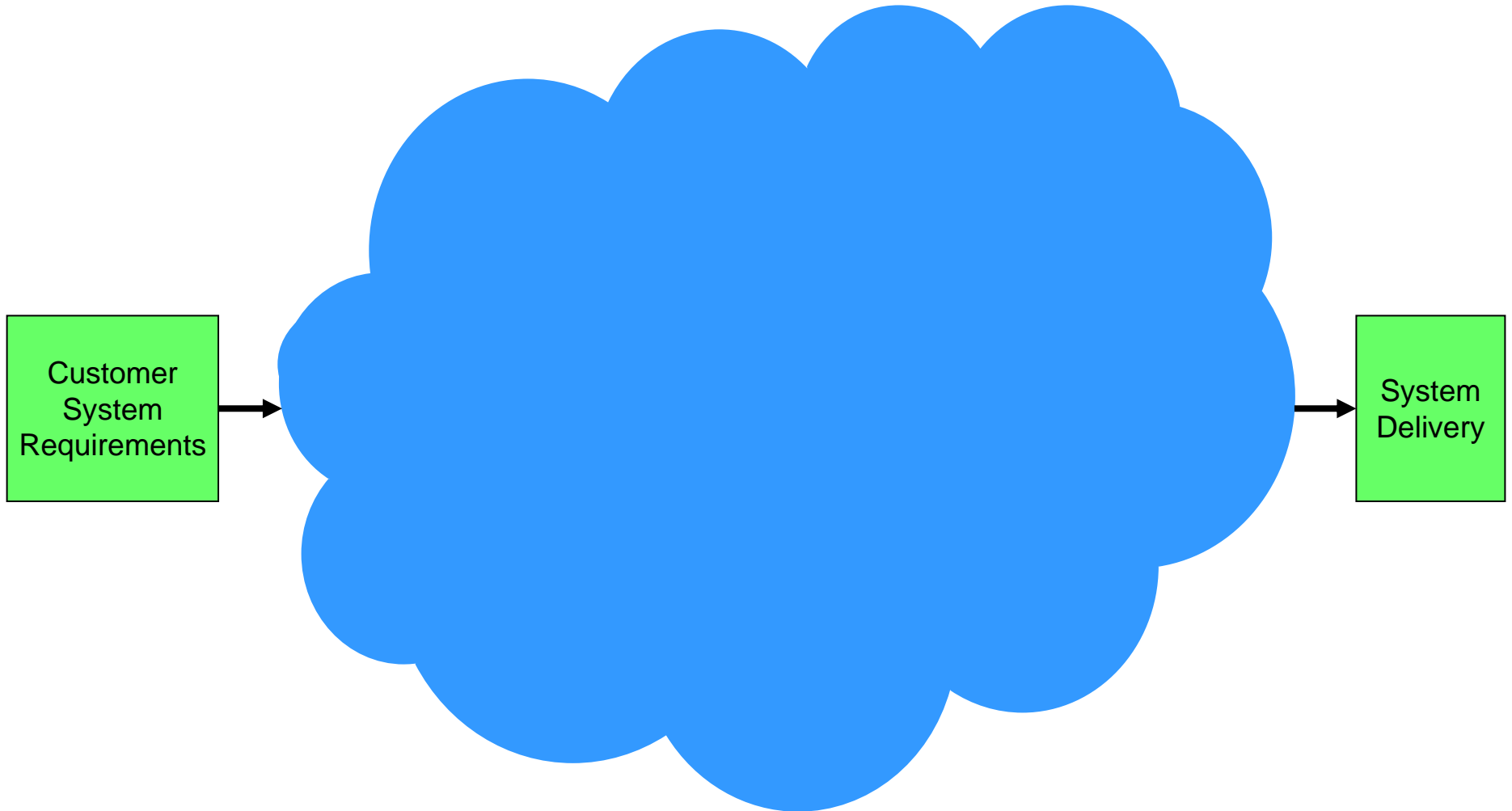


Latch occupies 1 or more address locations
Creates a larger physical address space

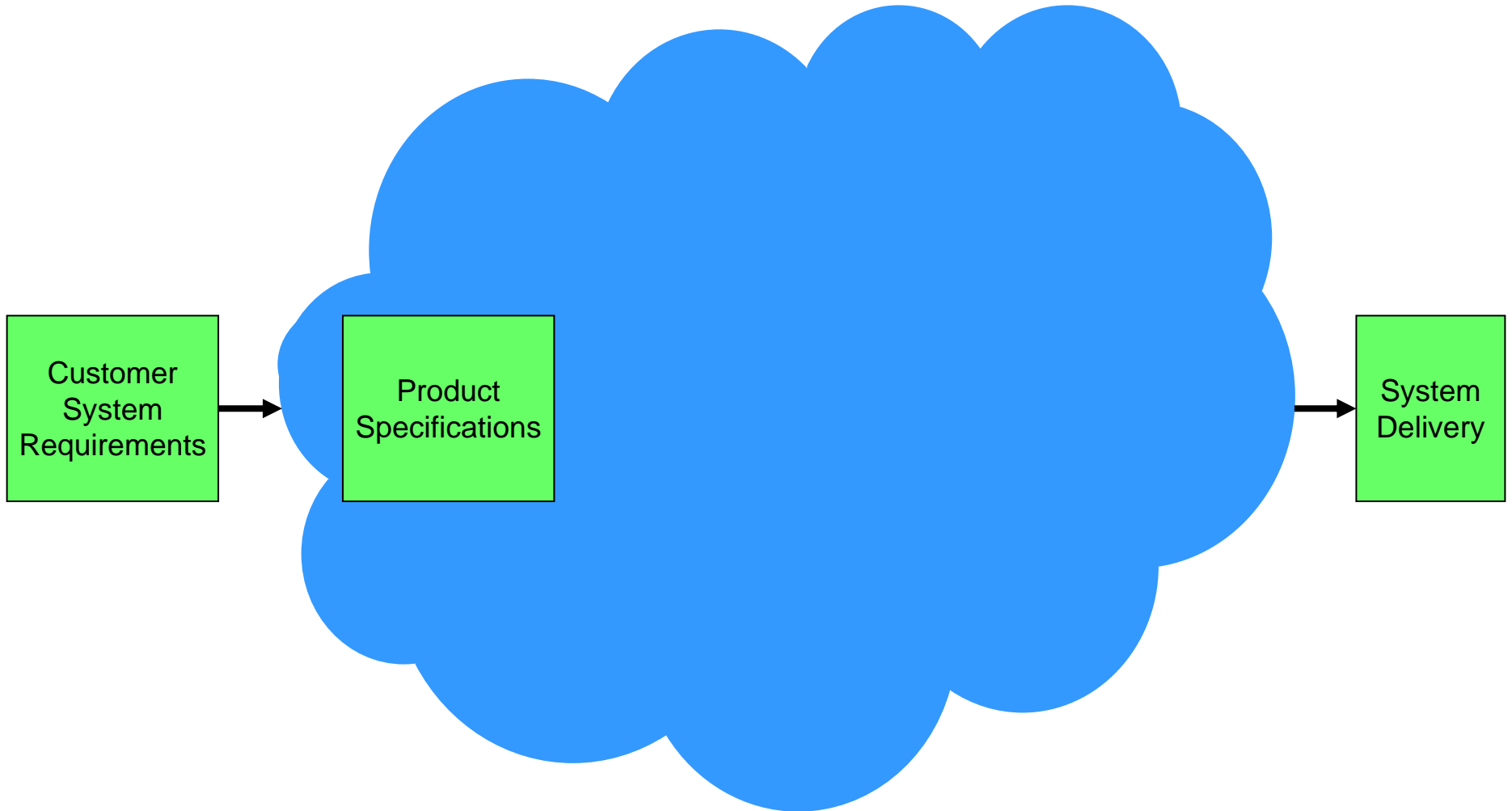
Memory Paging



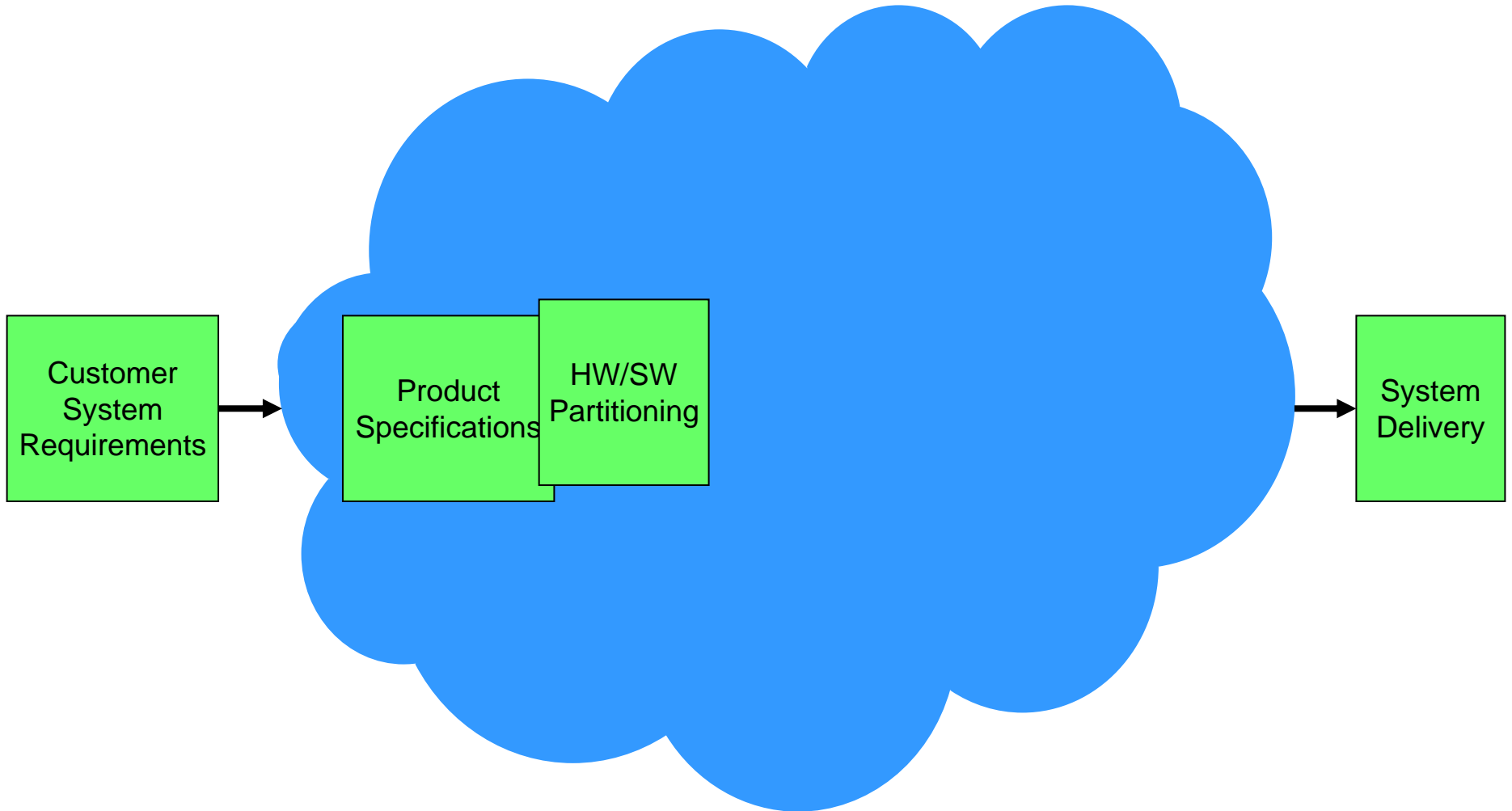
Embedded System Development



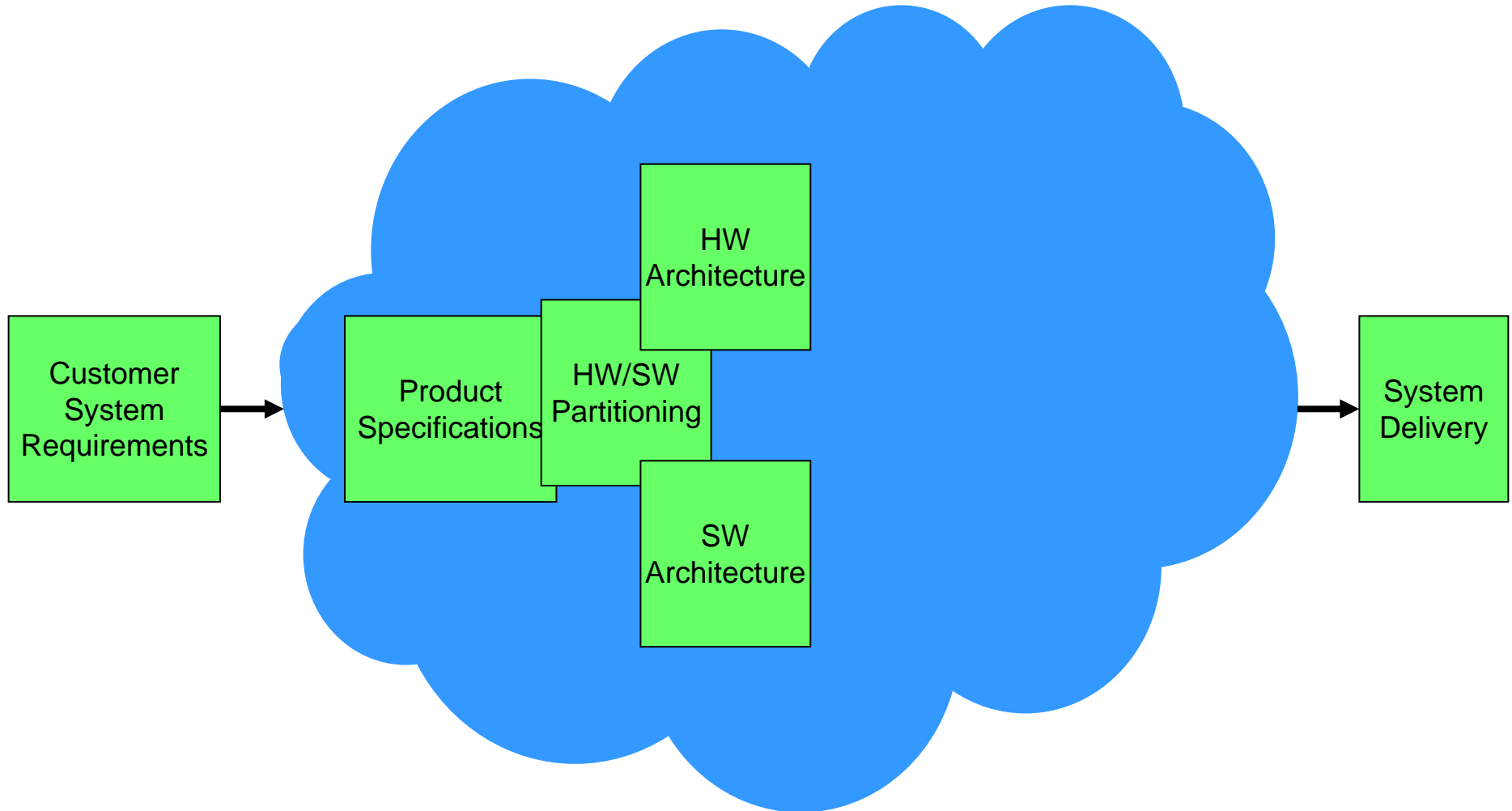
Embedded System Development



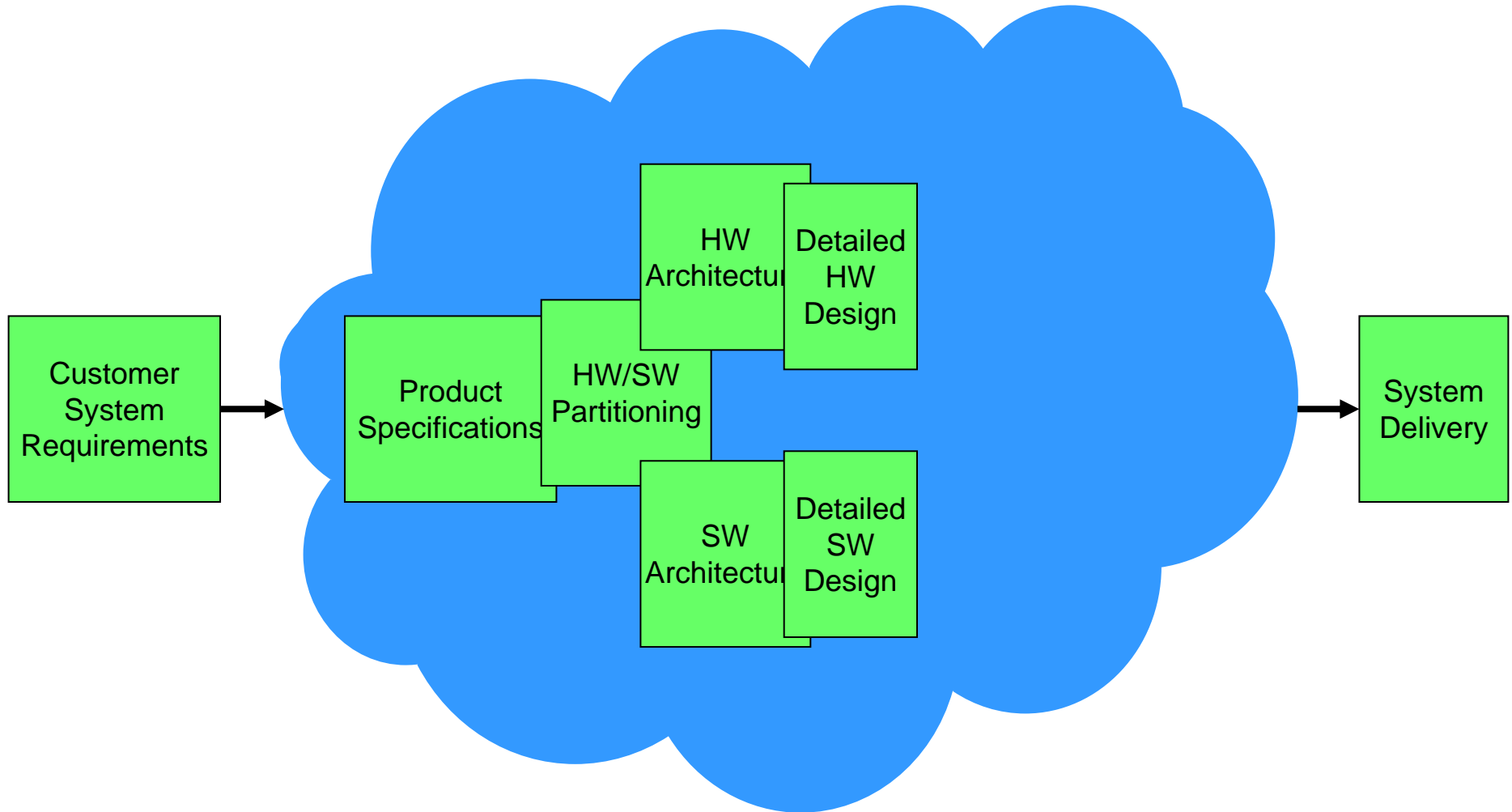
Embedded System Development



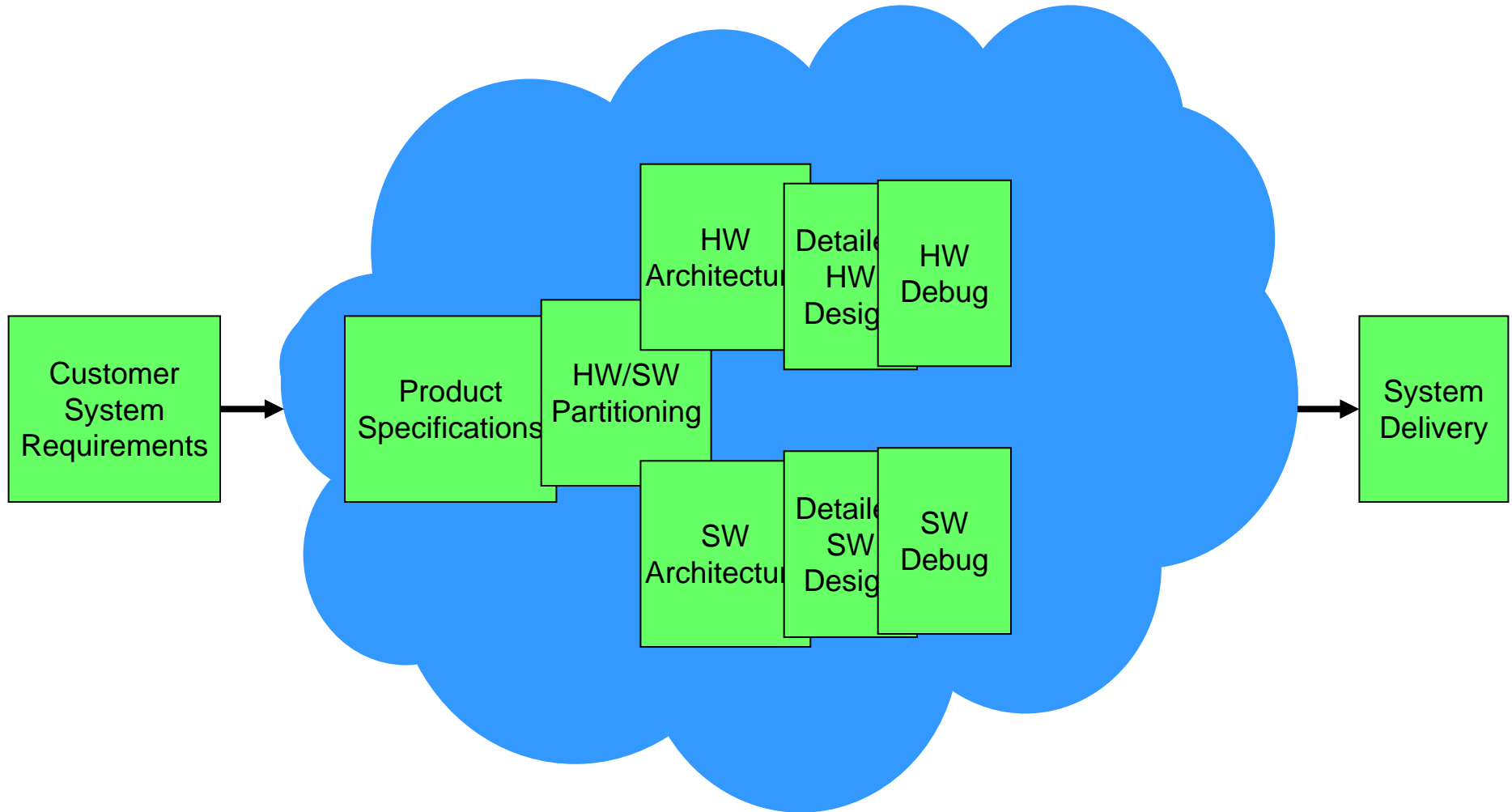
Embedded System Development



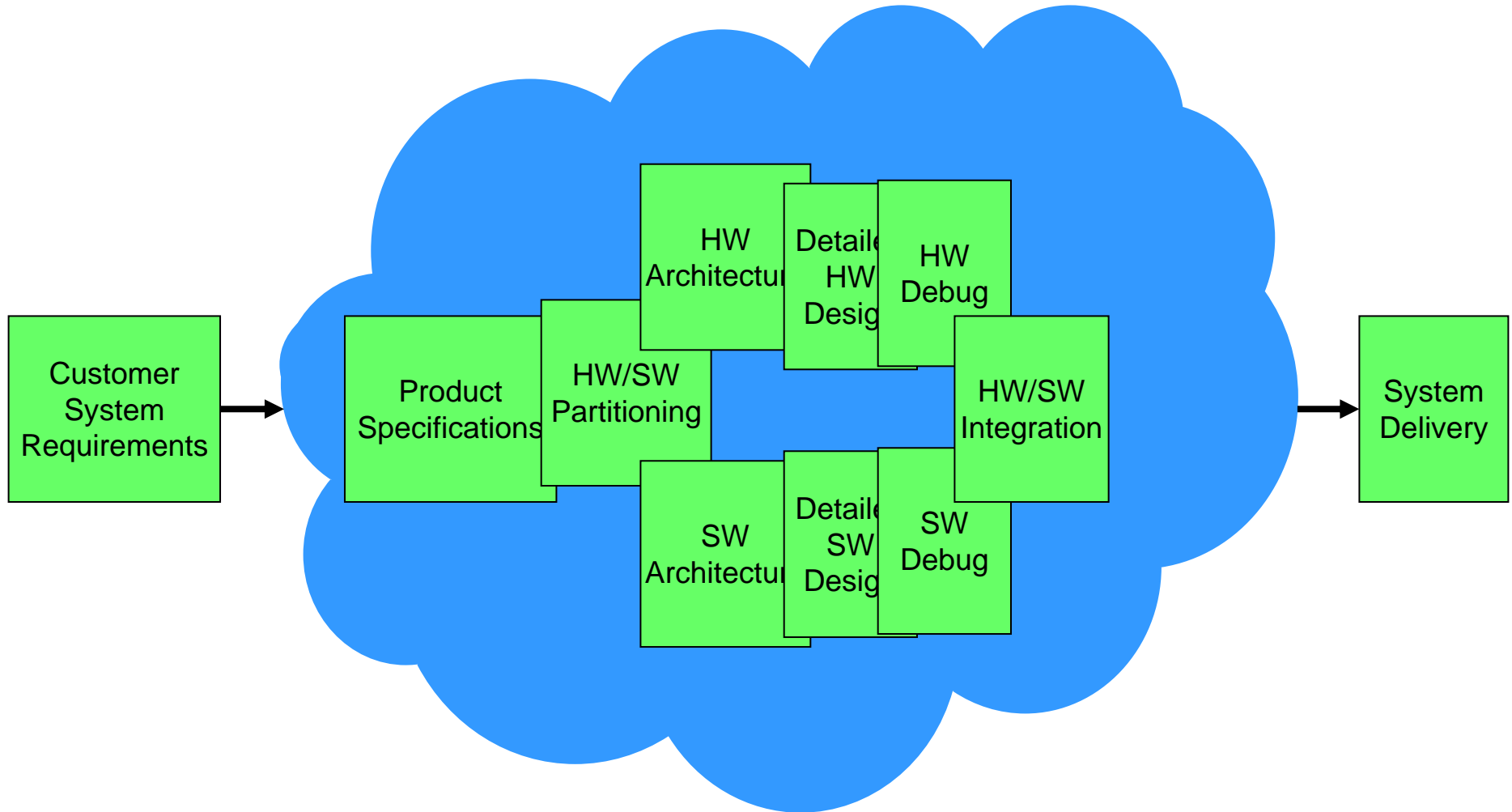
Embedded System Development



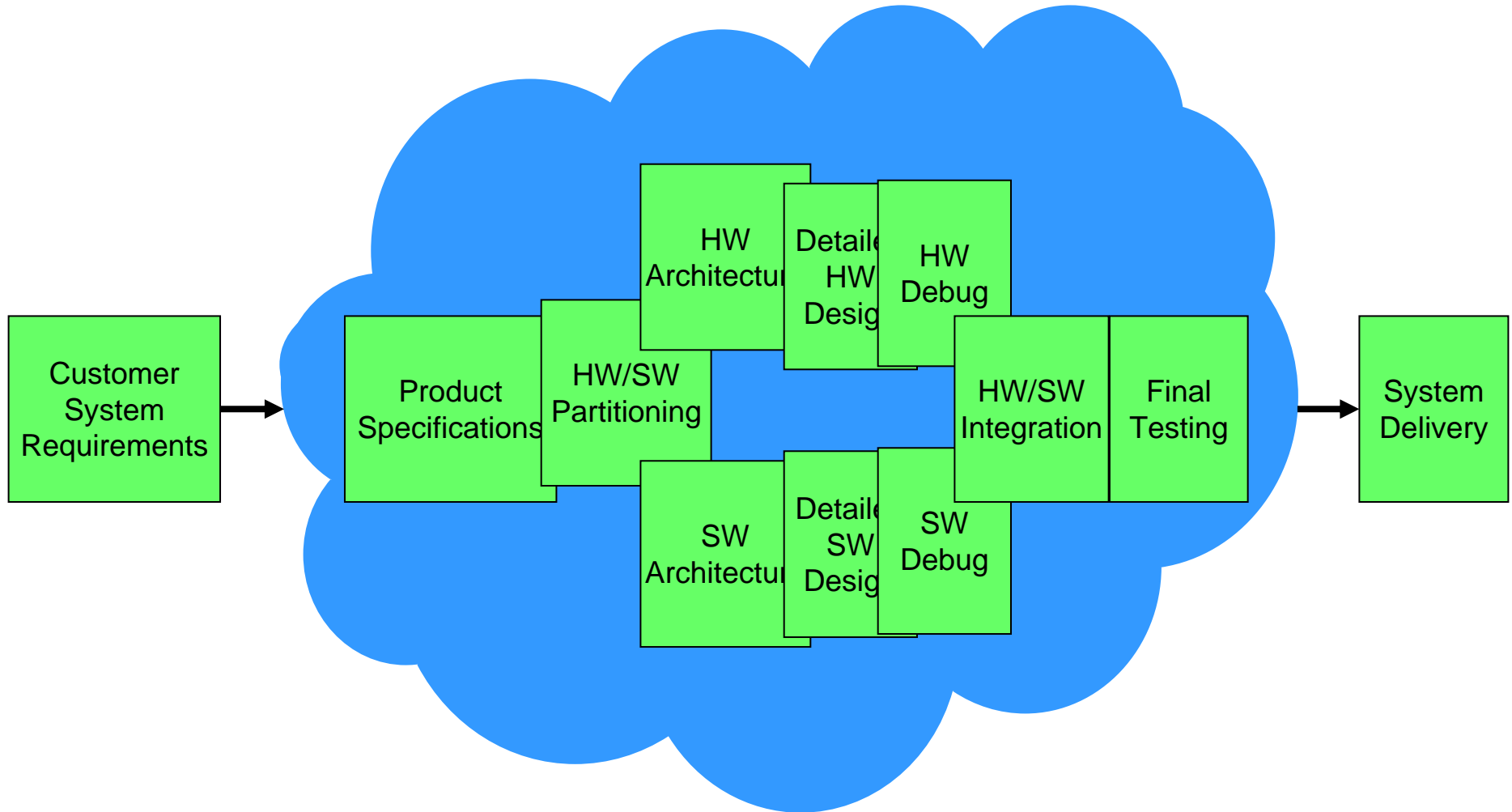
Embedded System Development



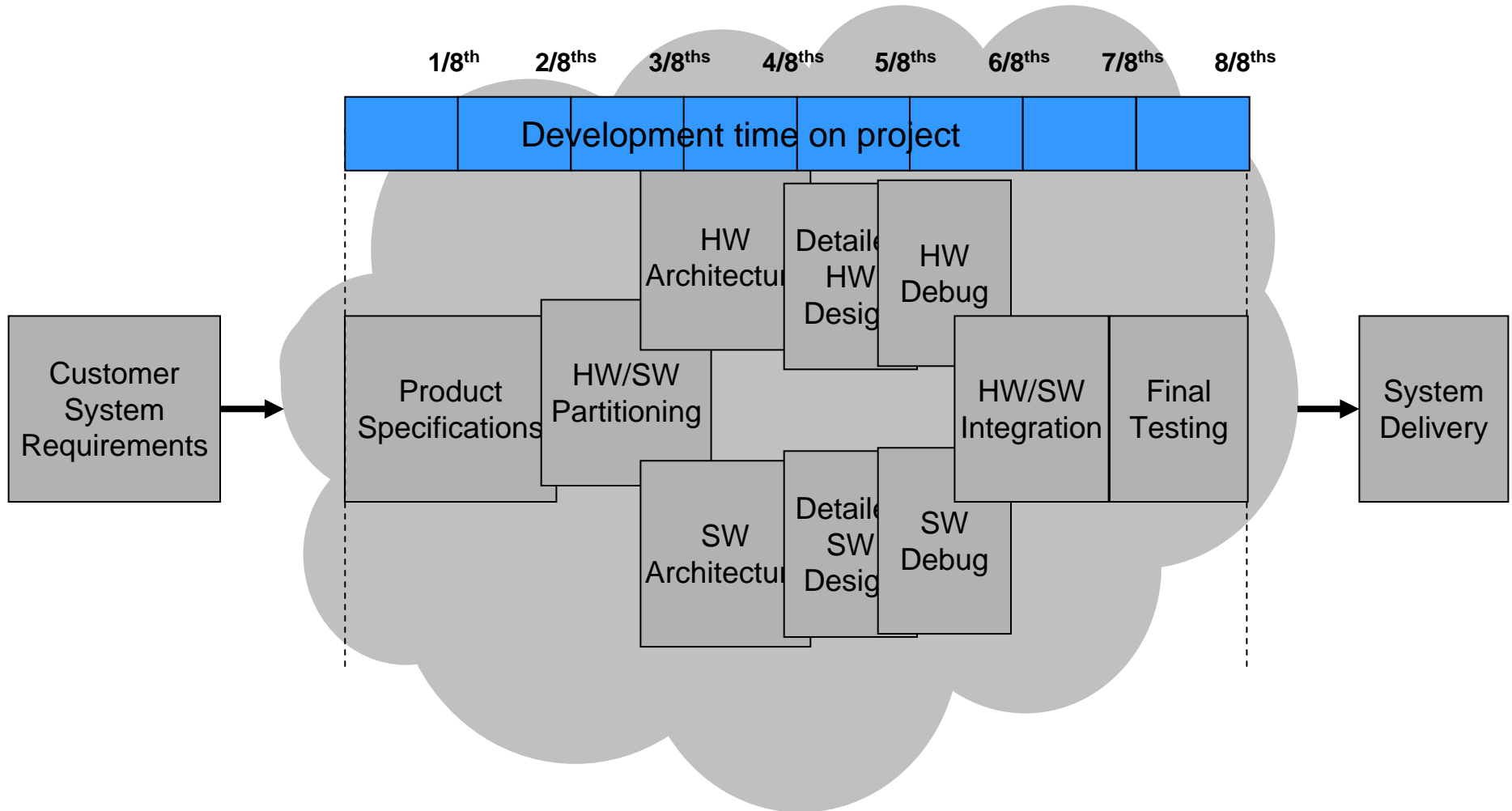
Embedded System Development



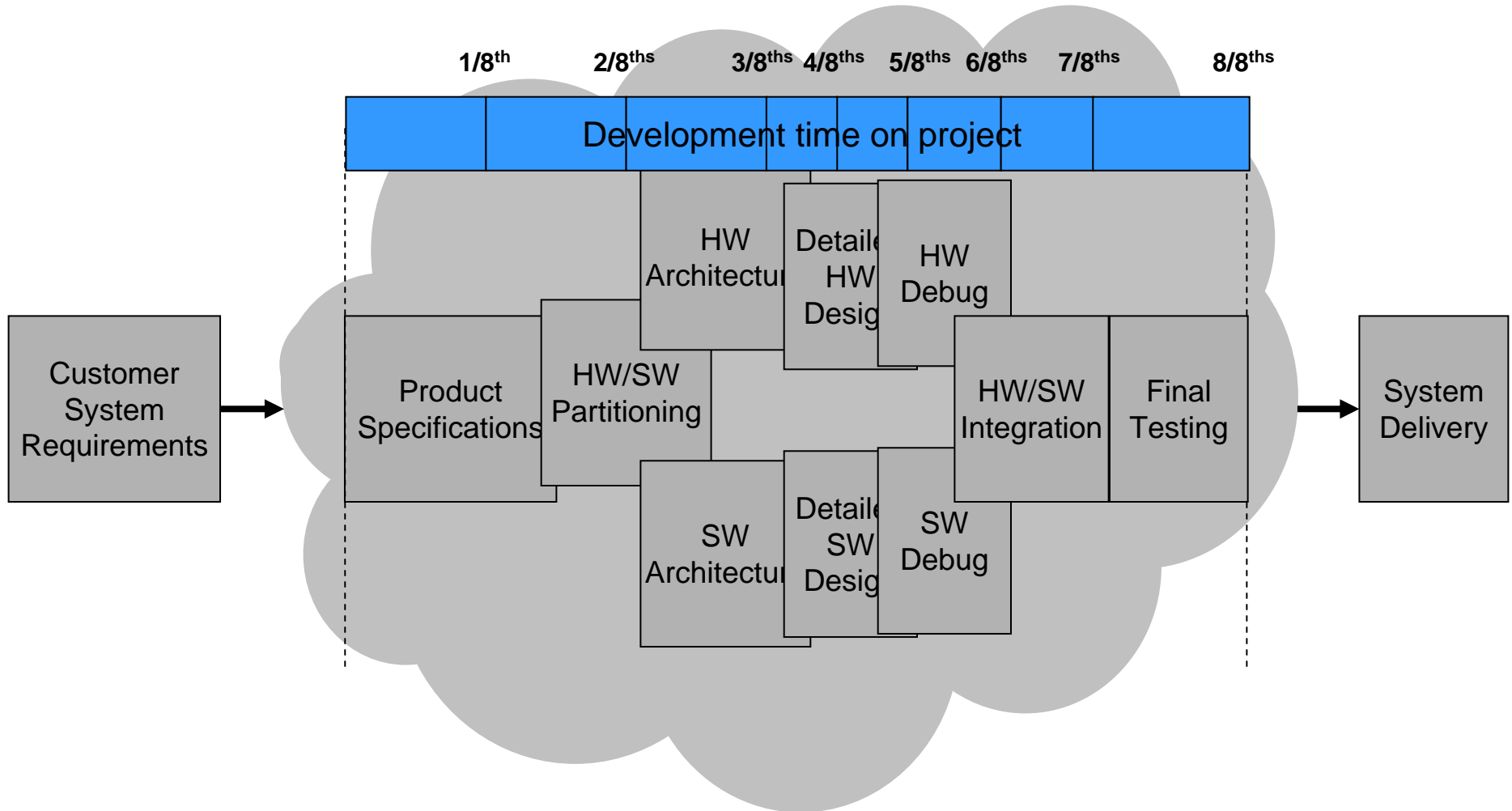
Embedded System Development



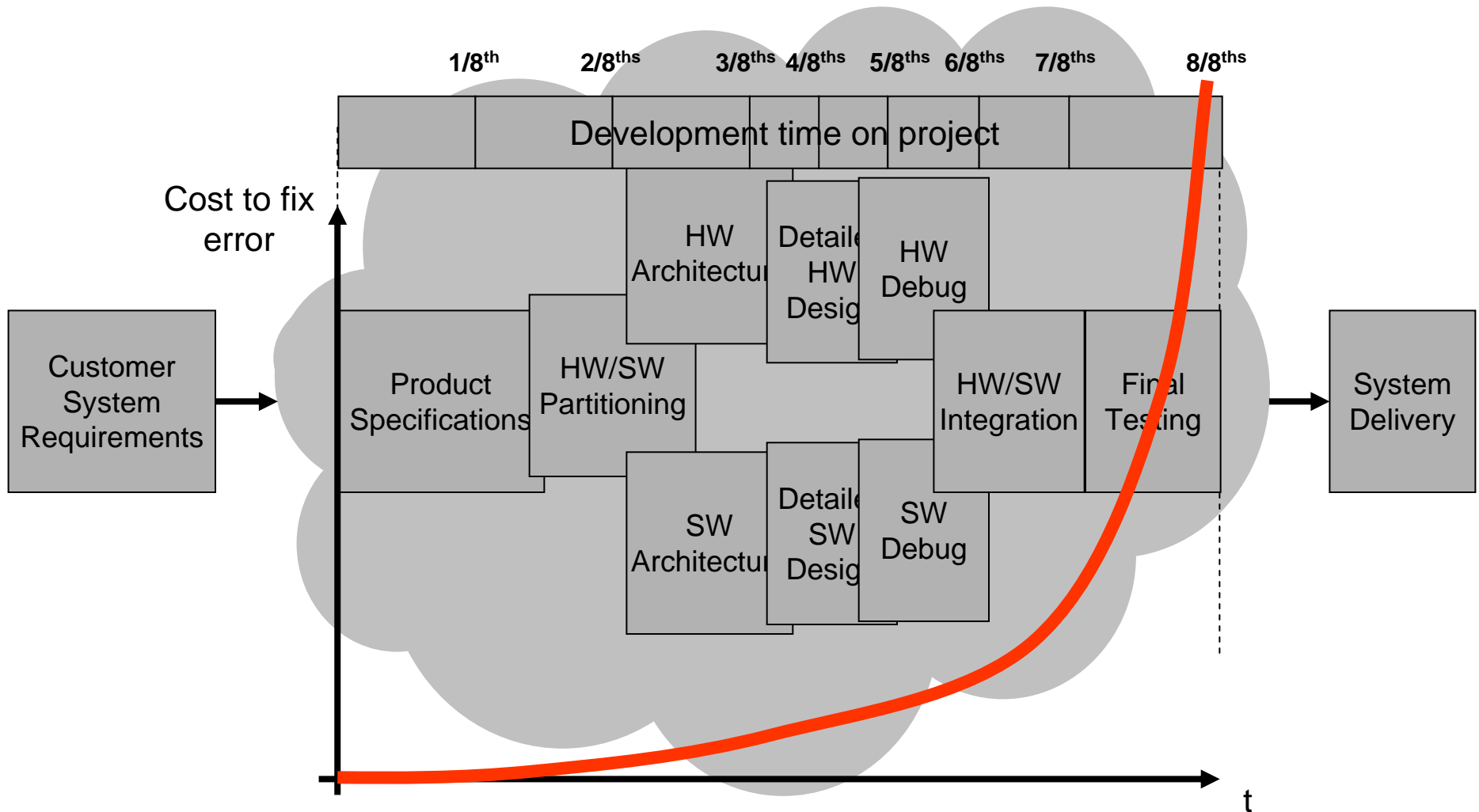
Embedded System Development



Embedded System Development



Embedded System Development



HW/SW Design Errors

- Consider the following program segment:

```
unsigned long int x;
x = 0x01234567;
y = first_byte(x);

/* what is the value of y? */

byte first_byte(unsigned long int in)
{
    return( (0xFF000000&in)>>24);
}
```

HW/SW Design Errors

- Consider the following program segment:

```
unsigned long int x;  
x = 0x01234567;  
y = first_byte(x);  
  
/* what is the value of y? */  
  
byte first_byte(unsigned long int in)  
{  
    return( (0xFF000000&in)>>24);  
}
```



y = 0x67

HW/SW Co-design Errors

- Consider the following program segment:
- Sun workstations and TI DSPs address memory differently than SGI workstations and Intel '86 family PCs

```

unsigned long int x;
x = 0x01234567;
y = first_byte(x);

/* what is the value of y? */

byte first_byte(unsigned long int in)
{
return( (0xFF000000&in)>>24);
}

```

“Big Endian vs. Little Endian” arithmetic

Depends on processor arithmetic architecture



y = 0x67



y = 0x01

More on HW/SW Co-design Errors

- Previous case was easy to debug. What about this one:

```
struct port
{
    byte status;
    byte data;
}
.
.
.

port.status = DATA_RATE | BITS_7 | NO_PARITY;
.
.
.
port.data = *x++;
```

Still More on HW/SW Co-design Errors

- First case was easy to debug. Second case was a little harder.

What about this one:

```
boolean PN_generator (long unsigned int *state)
{
    next_bit = parity(state & MASK);
    *state = *state>>1 | next_bit<<31;
}
```

```
boolean parity(long unsigned int x)
{
    boolean p=0;
    for(i=0;i<32;i++)
    {
        p ^= x&0x01;
        x = x>>1;
    }
    return(p);
}
```

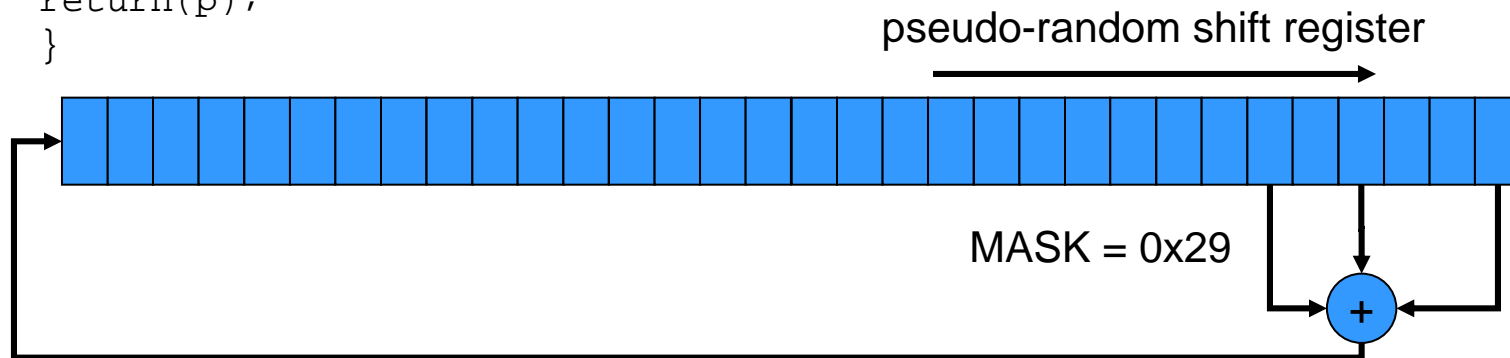
Still More on HW/SW Co-design Errors

- First case was easy to debug. Second case was a little harder.

What about this one:

```
boolean PN_generator (long unsigned int state)
{
    next_bit = parity(state & MASK);
    state = state>>1 | next_bit<<31;
}
```

```
boolean parity(long unsigned int x)
{
    boolean p=0;
    for(i=0;i<32;i++)
    {
        p ^= x&0x01;
        x = x>>1;
    }
    return(p);
}
```



Still More on HW/SW Co-design Errors

- First case was easy to debug. Second case was a little harder.

What about this one:

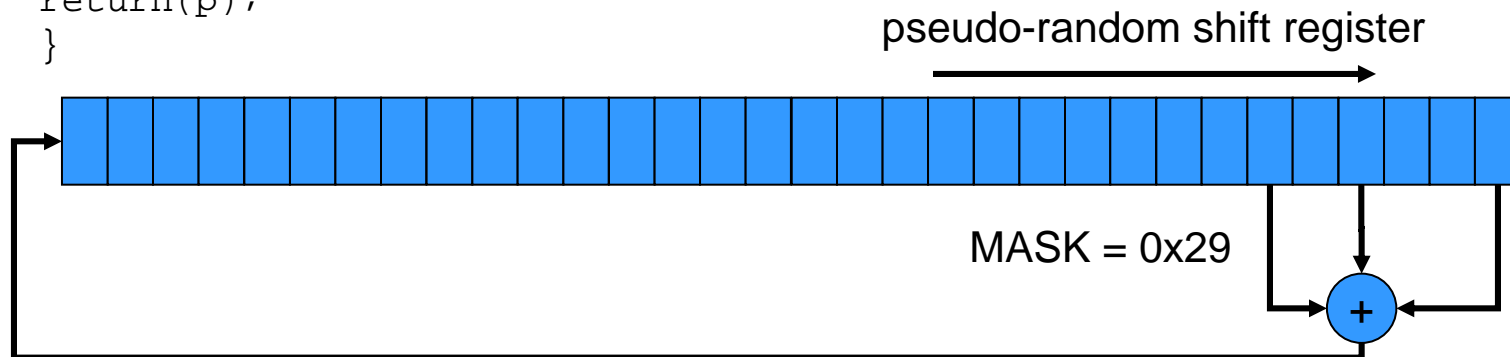
```
boolean PN_generator (long unsigned int state)
{
    next_bit = parity(state & MASK);
    state = state>>1 | next_bit<<31;
}
```

```
boolean parity(long unsigned int x)
{
    boolean p=0;
    for(i=0;i<32;i++)
    {
        p ^= x&0x01;
        x = x>>1;
    }
    return(p);
}
```

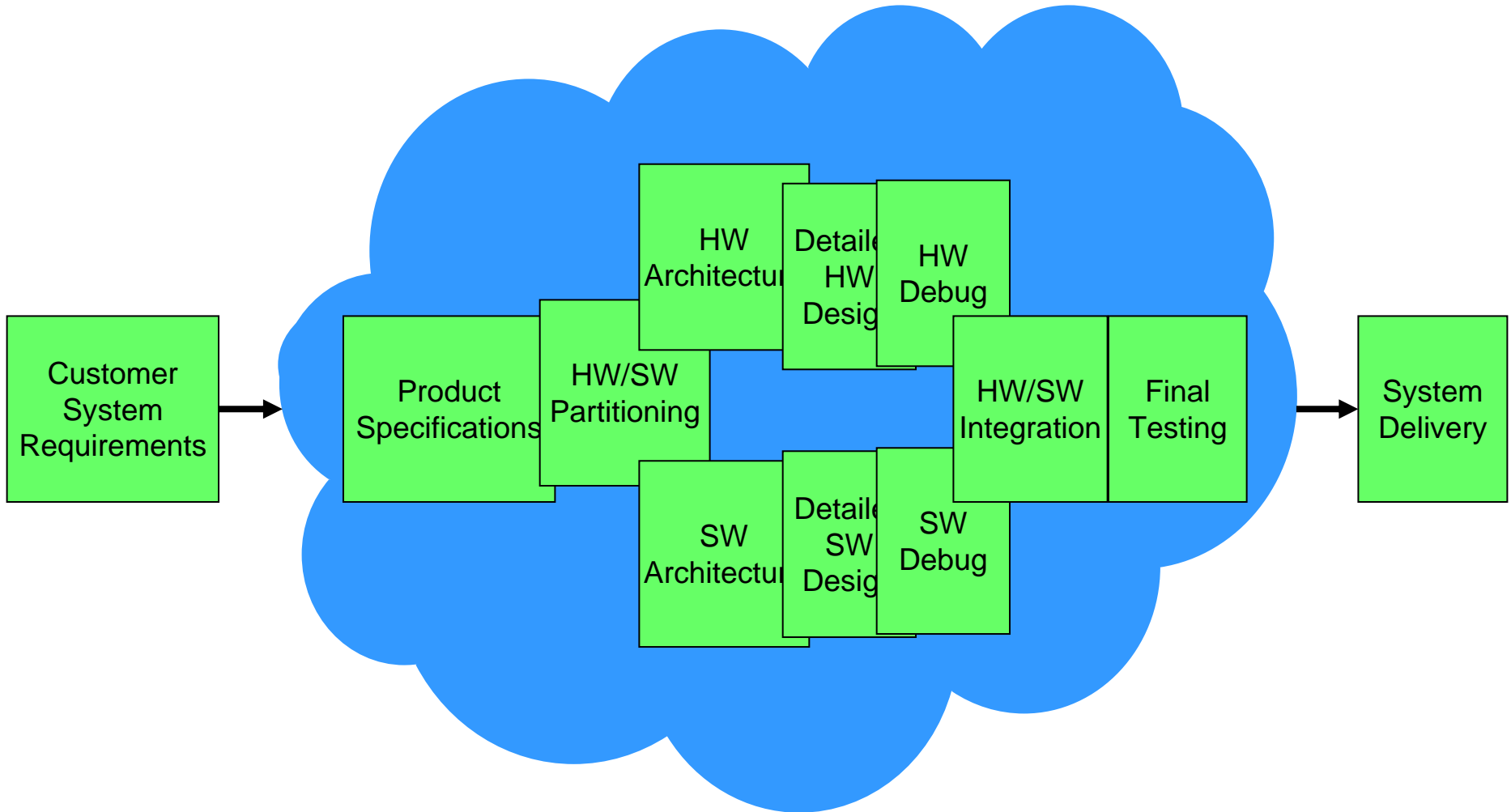
Or:

```
unsigned int rand(unsigned int x)
{
    return(mod(A*x+B,C));
}
```

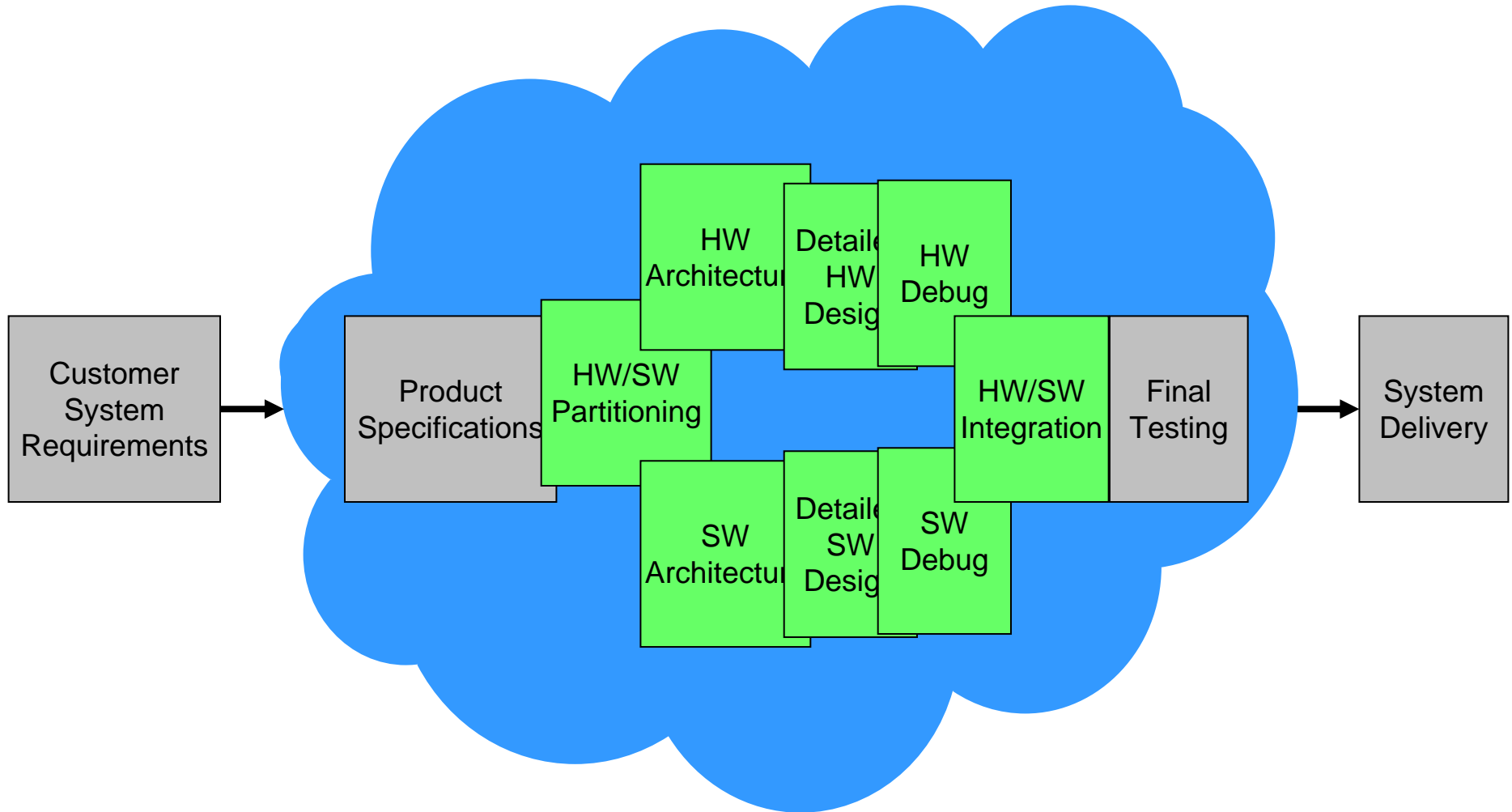
A, B, and C are constants chosen to generate maximal length random sequence (or do they?)



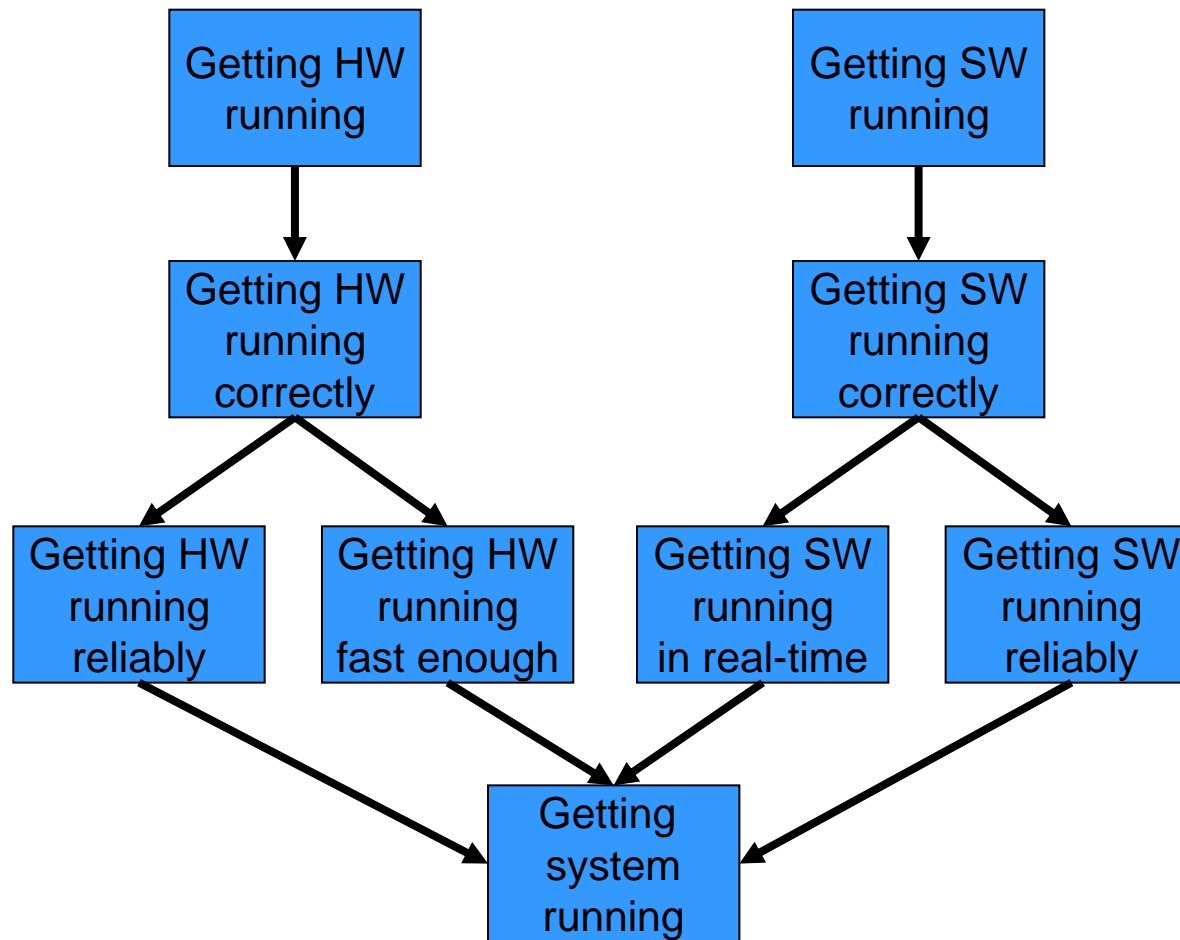
Embedded System Development



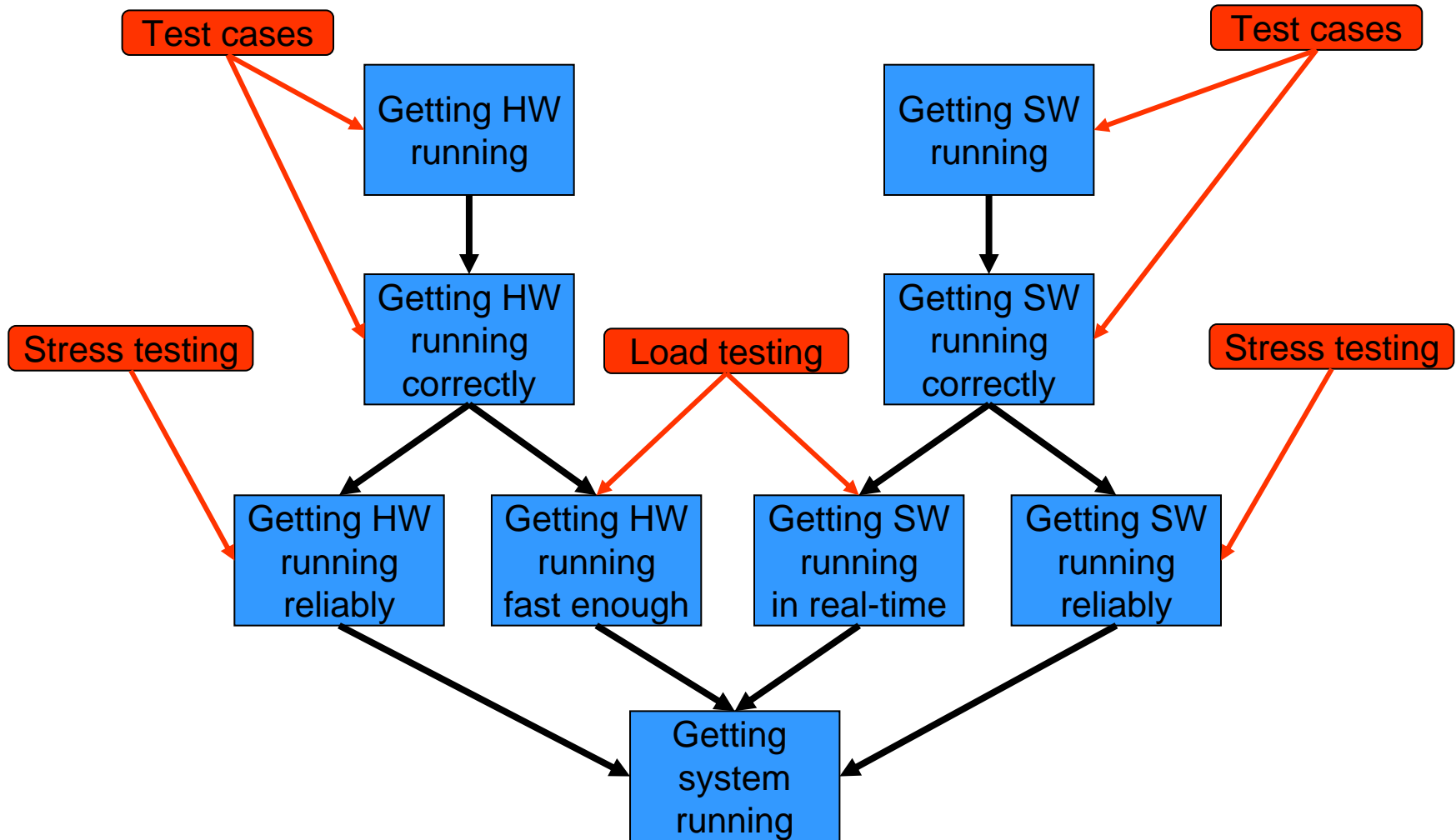
Embedded System Development



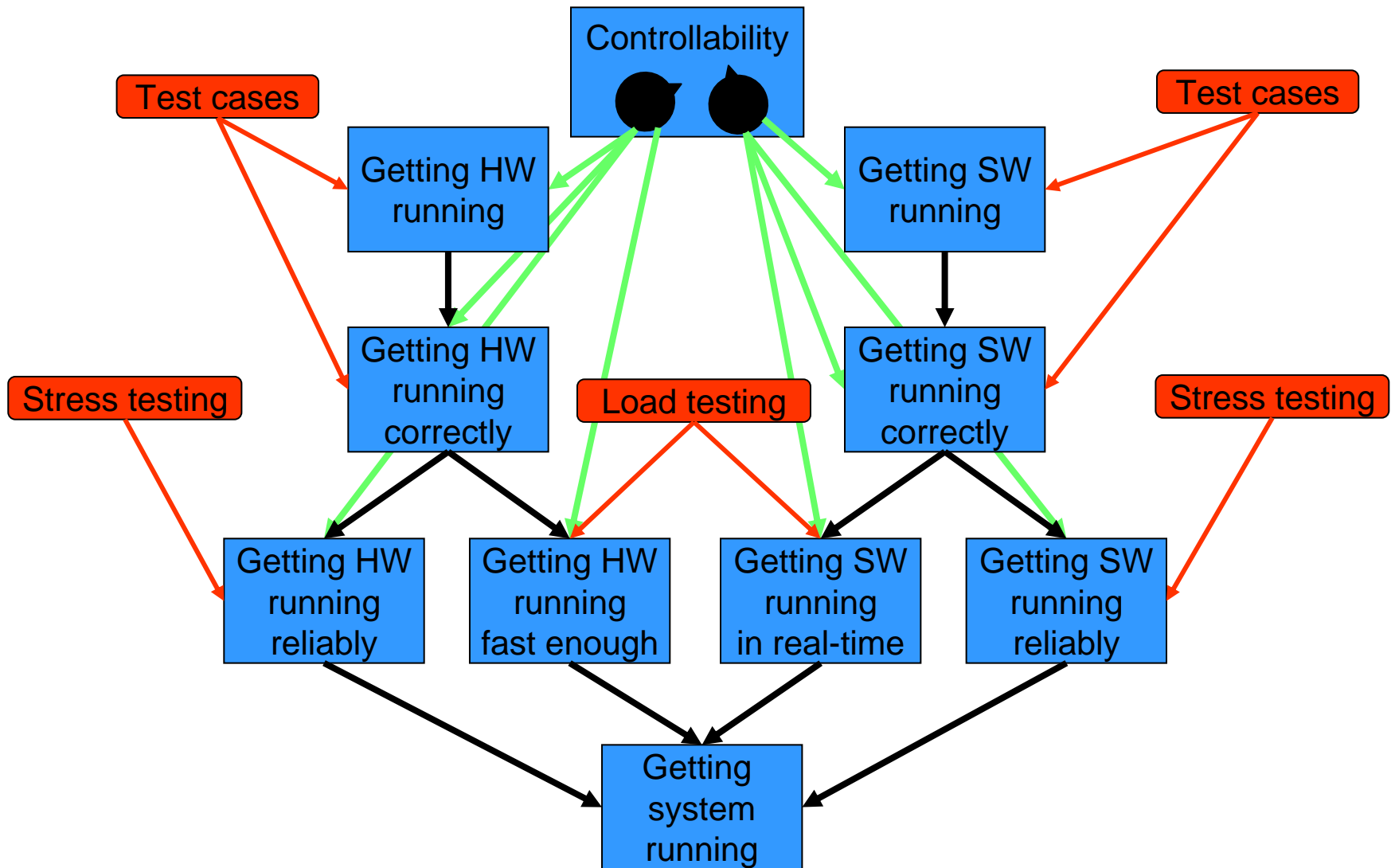
Getting a Real-Time Embedded System Running



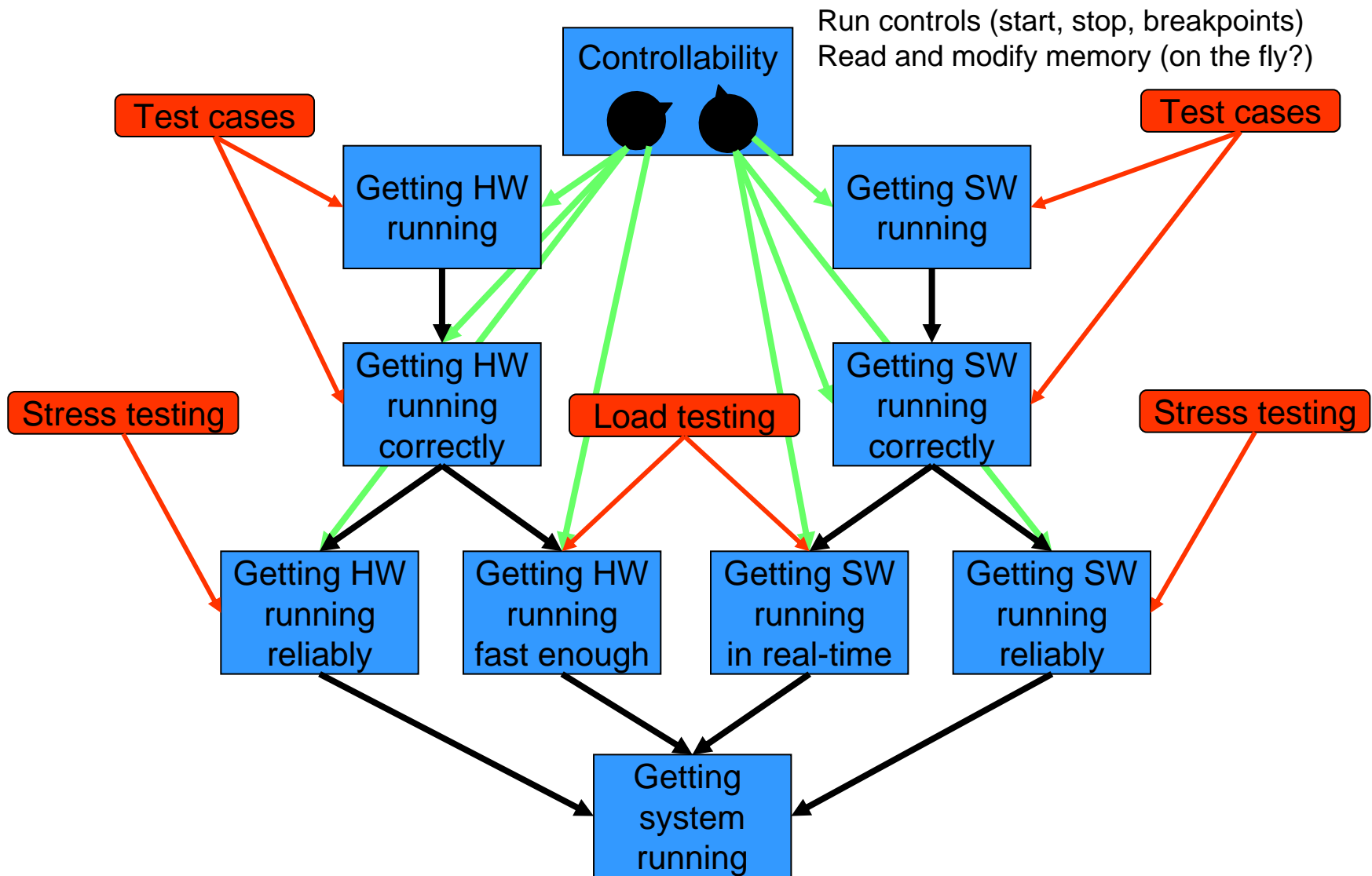
Getting a Real-Time Embedded System Running



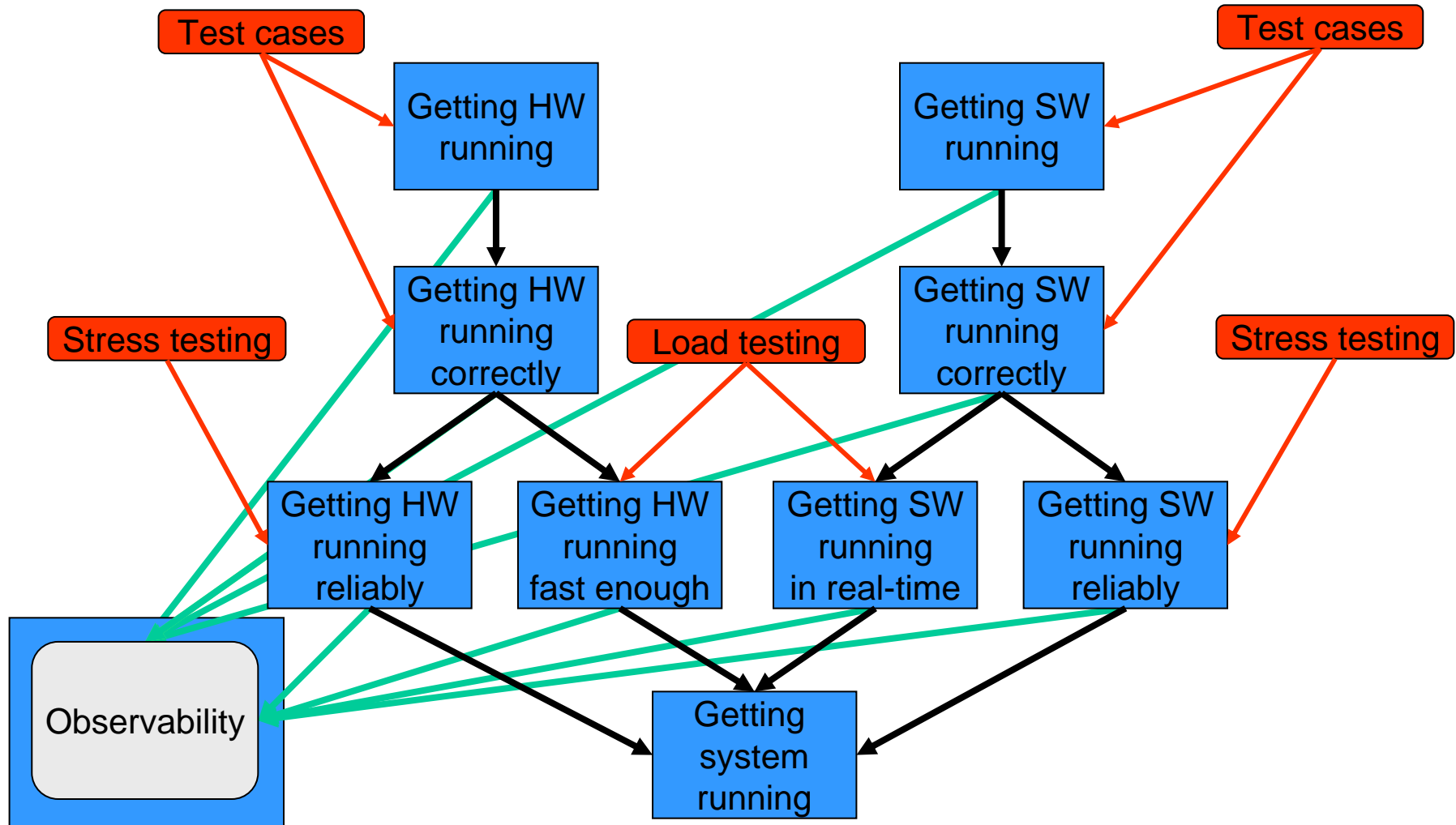
Getting a Real-Time Embedded System Running



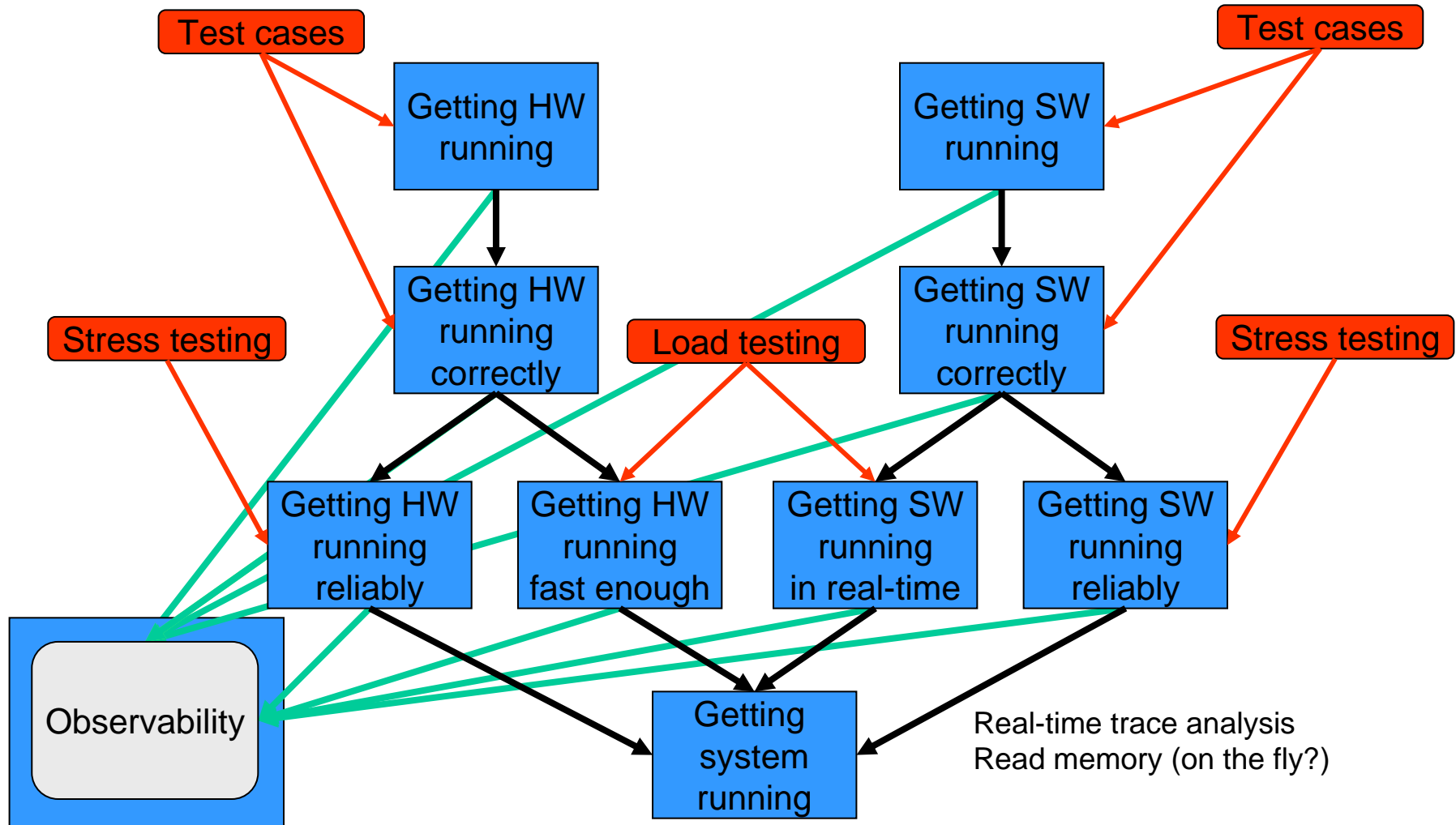
Getting a Real-Time Embedded System Running



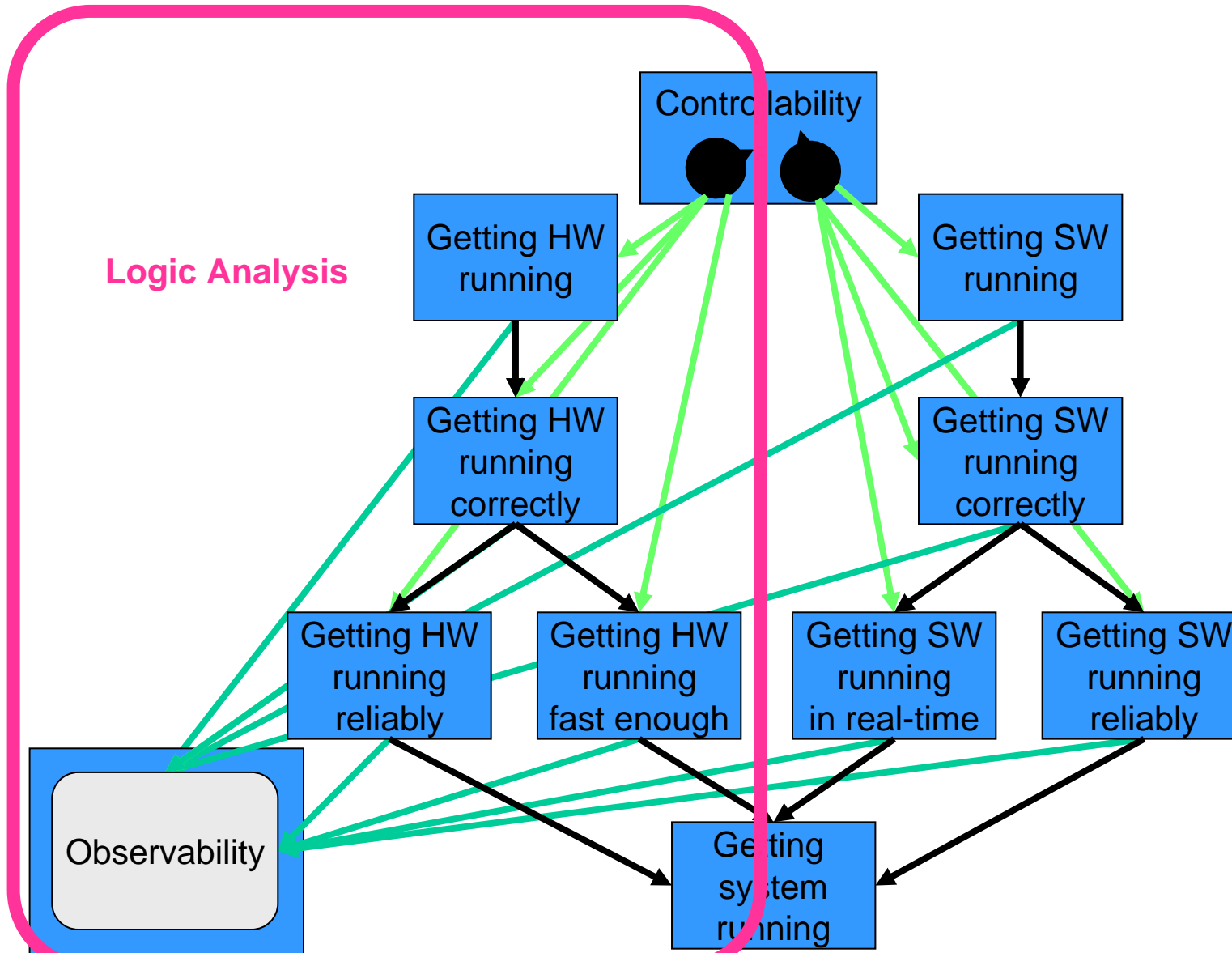
Getting a Real-Time Embedded System Running



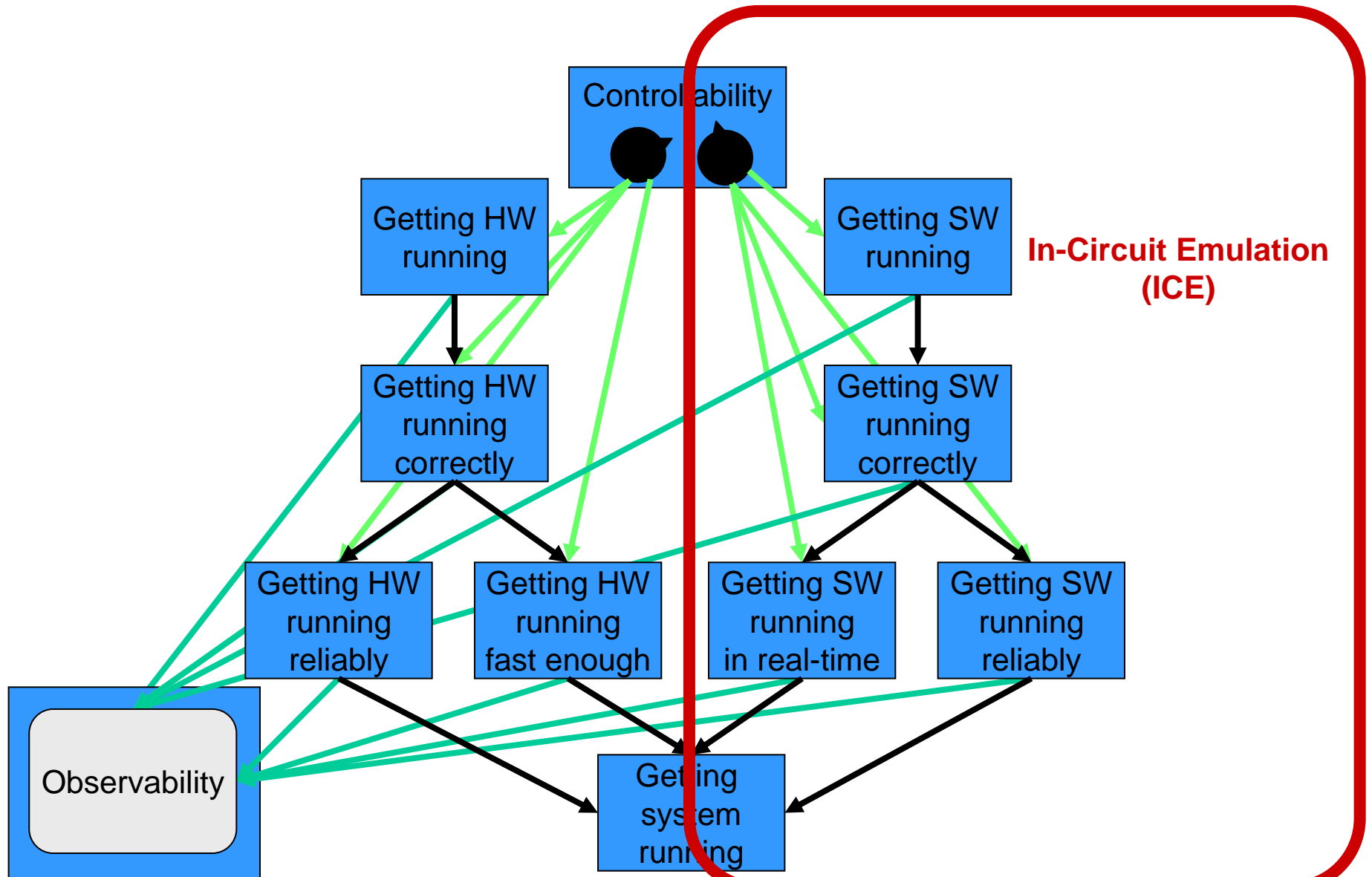
Getting a Real-Time Embedded System Running



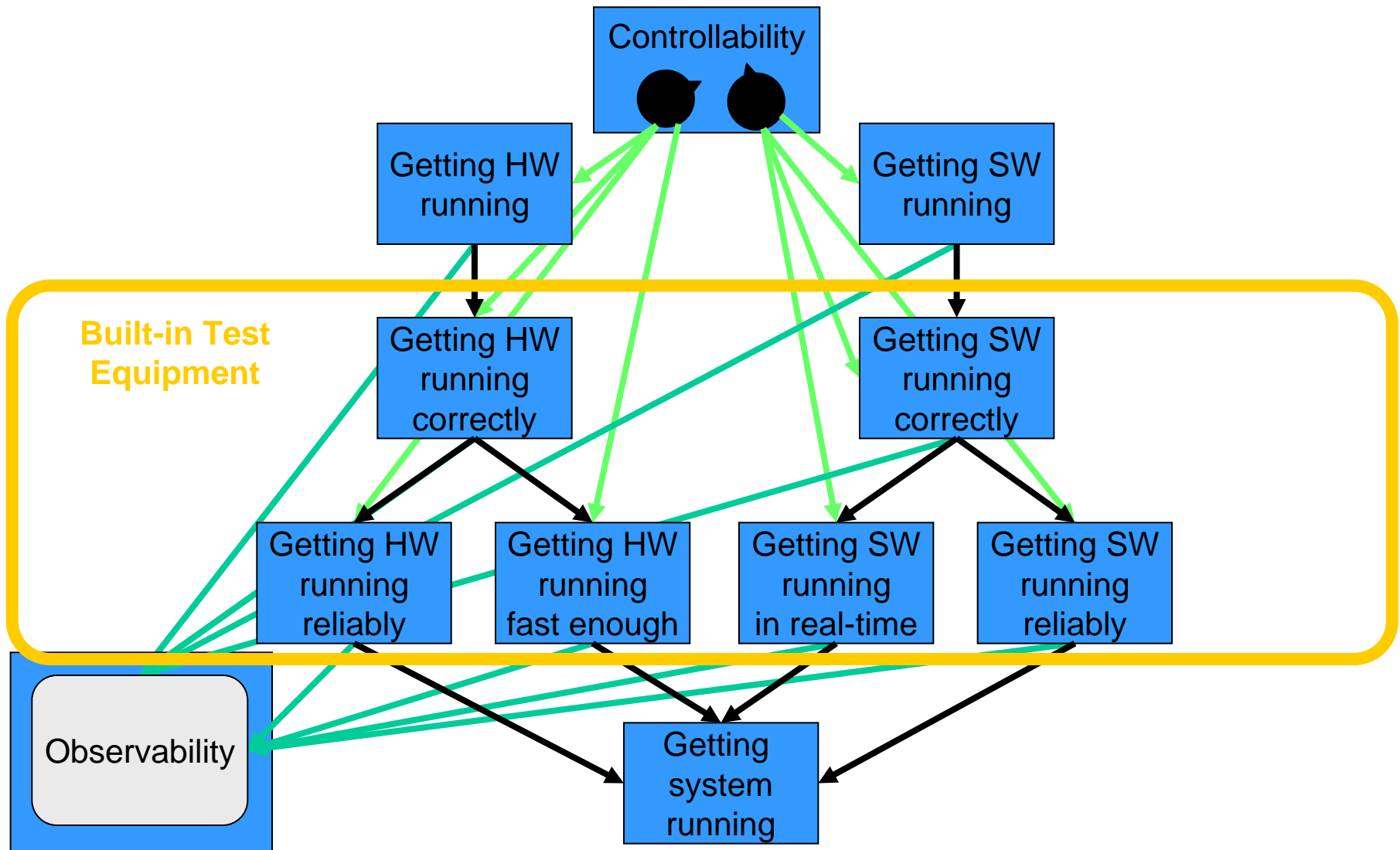
Getting a Real-Time Embedded System Running



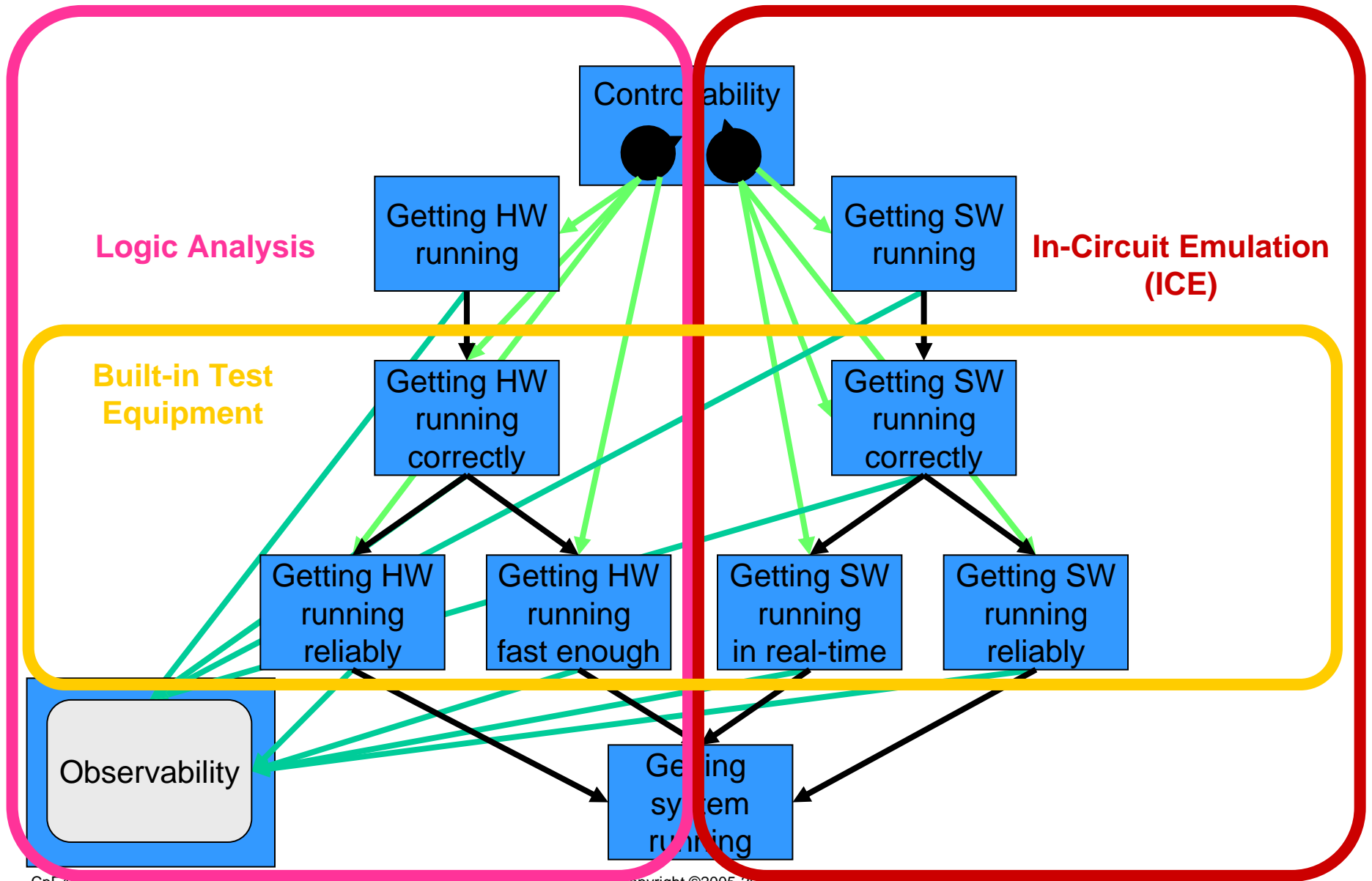
Getting a Real-Time Embedded System Running



Getting a Real-Time Embedded System Running



Getting a Real-Time Embedded System Running



Course Project

- Define the architecture of an embedded system with real-time requirements:
 - Create a high-level functional description of system (10%)
 - What does it do? what are the subsystems?
 - Define the input/output requirements for the system. (10%)
 - Sensors, actuators, indicators, etc.
 - Define the system's real-time requirements (10%)
 - Identify any safety requirements for the system and describe fail-soft design design elements ***
 - Suggest hardware vs. programmable controller tradeoffs (20%)
 - What is the best technology to realize the system?
 - Is design suited to one processor or are many needed? ***
 - How would you handle interprocessor communicaitons?
 - Estimate code size of programmable components (15%)
 - Is the code likely to fit within the constraints of available controllers?
 - Define software architecture for system
 - Estimate hardware complexity of design (15%)
 - Define how you will estimate system performance. How would you test it? (15%)
 - Describe how your system design can be extended if additional capabilities are needed (5%)
- A high-level paper design is required.
 - Any prototyping or simulation of the system you can perform will be given extra credit ***
 - A detailed design (hardware and/or software) will be given extra credit ***
- Fully cite all references
- A system with the design complexity of an automotive engine controller is the target. Harder or easier system designs will be graded appropriately

Assignment 4

- The Motorola 68000 and its descendents use a linear address space while the Intel 8086 and its descendents have used a segmented address space, requiring a page register and an offset register (within a page). Research the hardware and software advantages of these approaches.
- Investigate at least two of the following $\mu\text{P}/\mu\text{C}$'s. Do they use Big Endian or Little Endian numeric representation (note: it may be necessary to examine the instruction set data sheets looking at the effect of instructions like "Shift Right"):
 - Atmel AVR series
 - Intel 8051
 - Intel 80x86
 - Microchip PIC microcontrollers
 - Motorola 68HC12
 - Parallax Stamp
 - Zilog Z8