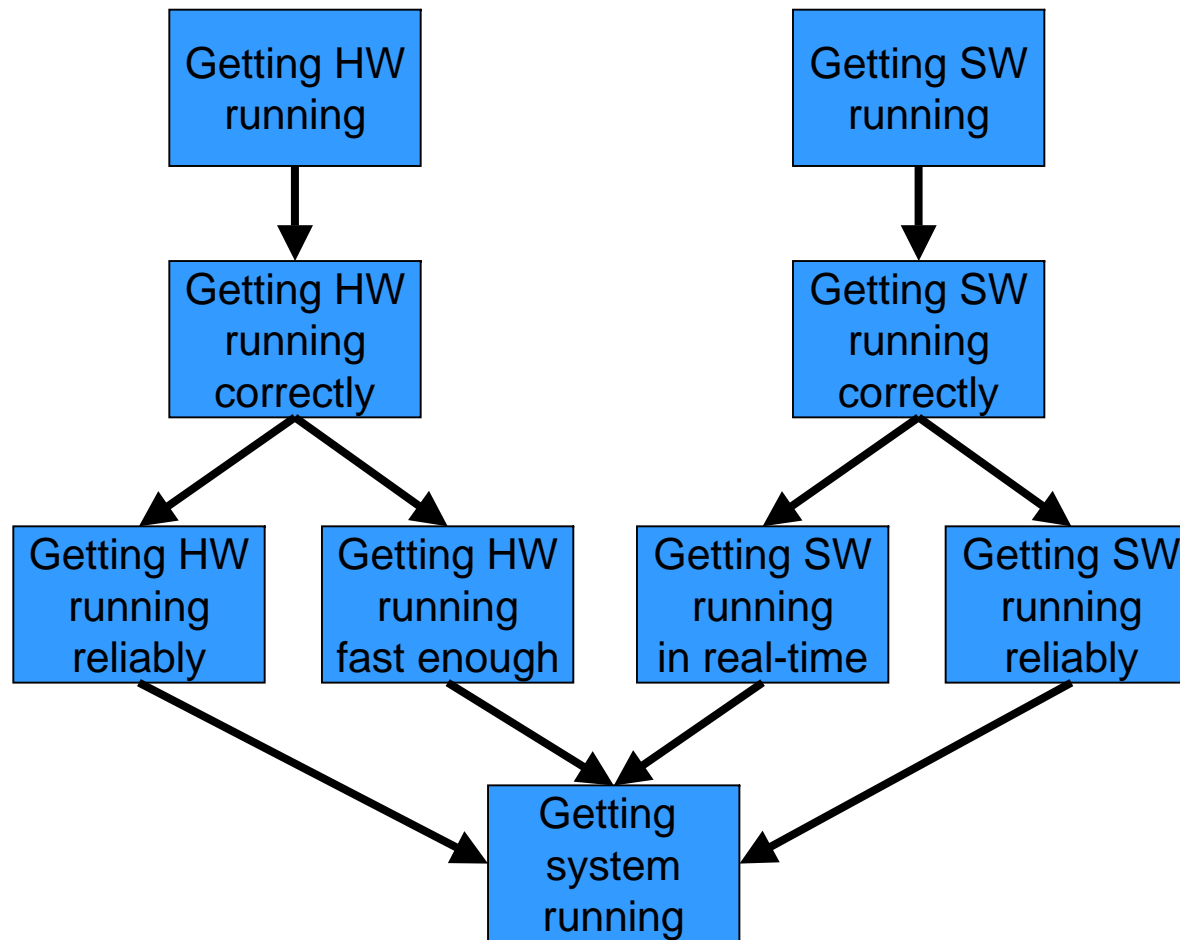


Architecture, Design and Implementation of Embedded Systems for Real-Time Applications

CpE-450 - Spring 05
Class 17

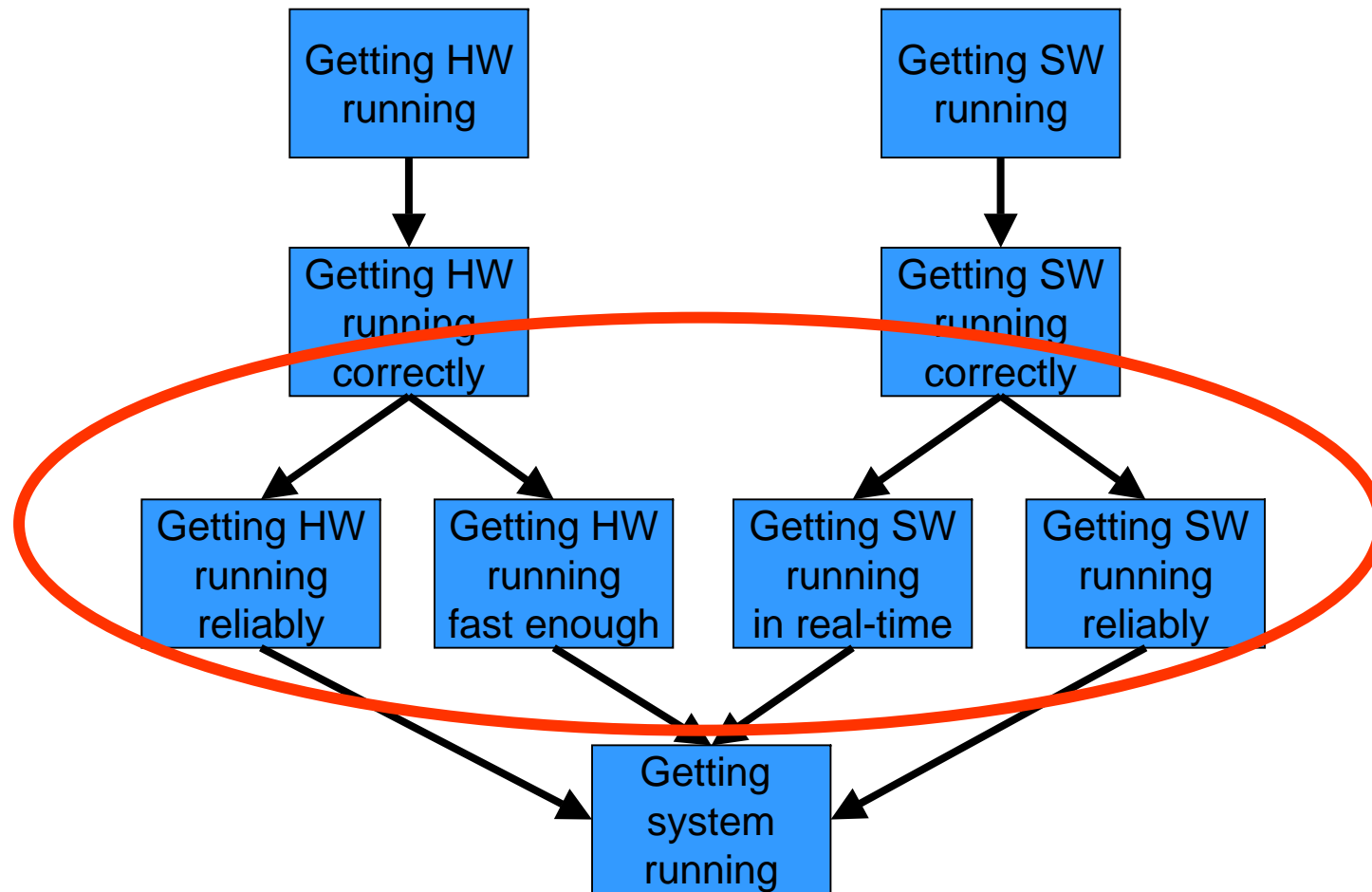
Bruce McNair
bmcnair@stevens.edu

Getting a Real-Time Embedded System Running



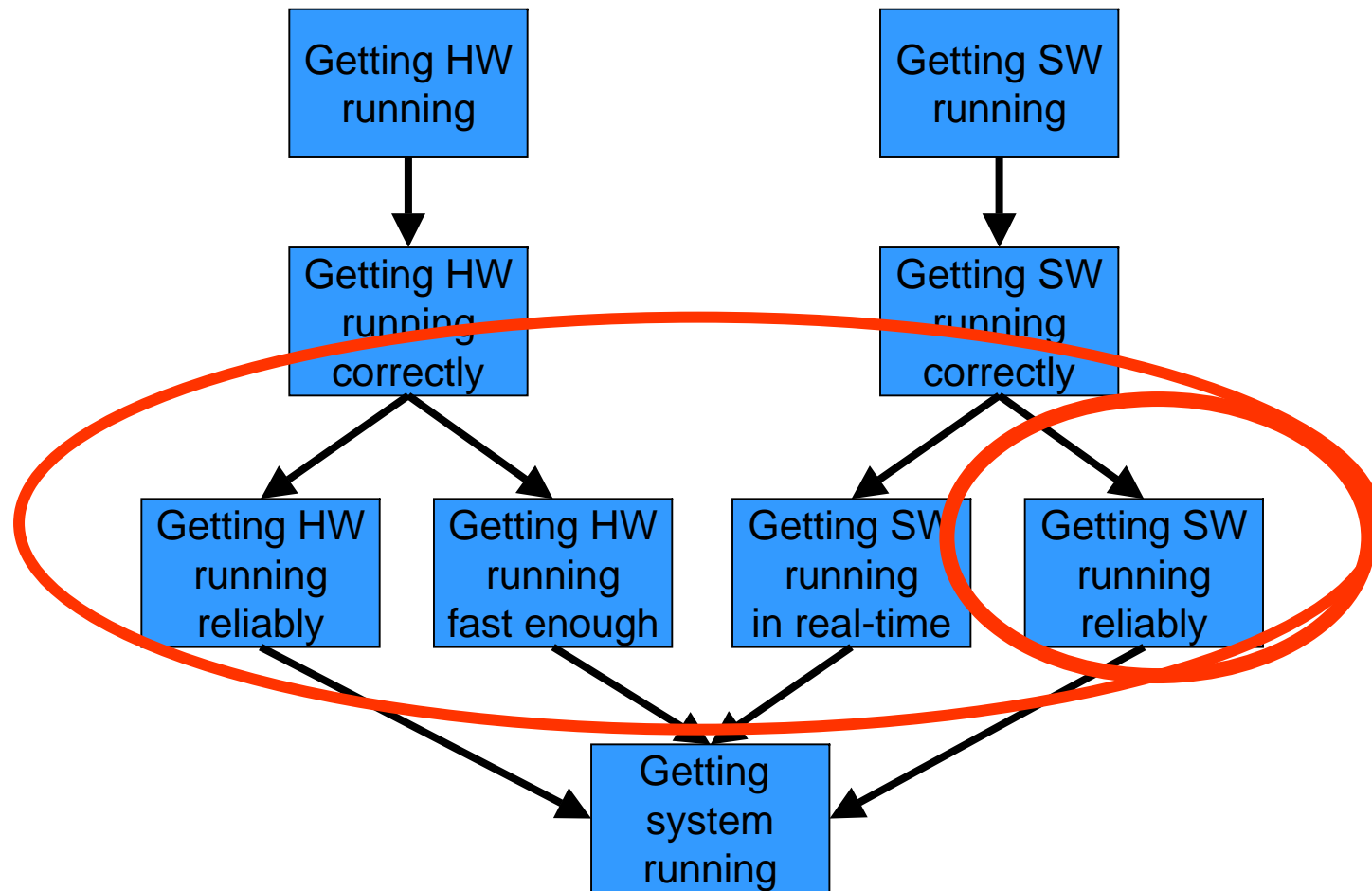
Getting a Real-Time Embedded System Running

- Evaluating performance and correctness

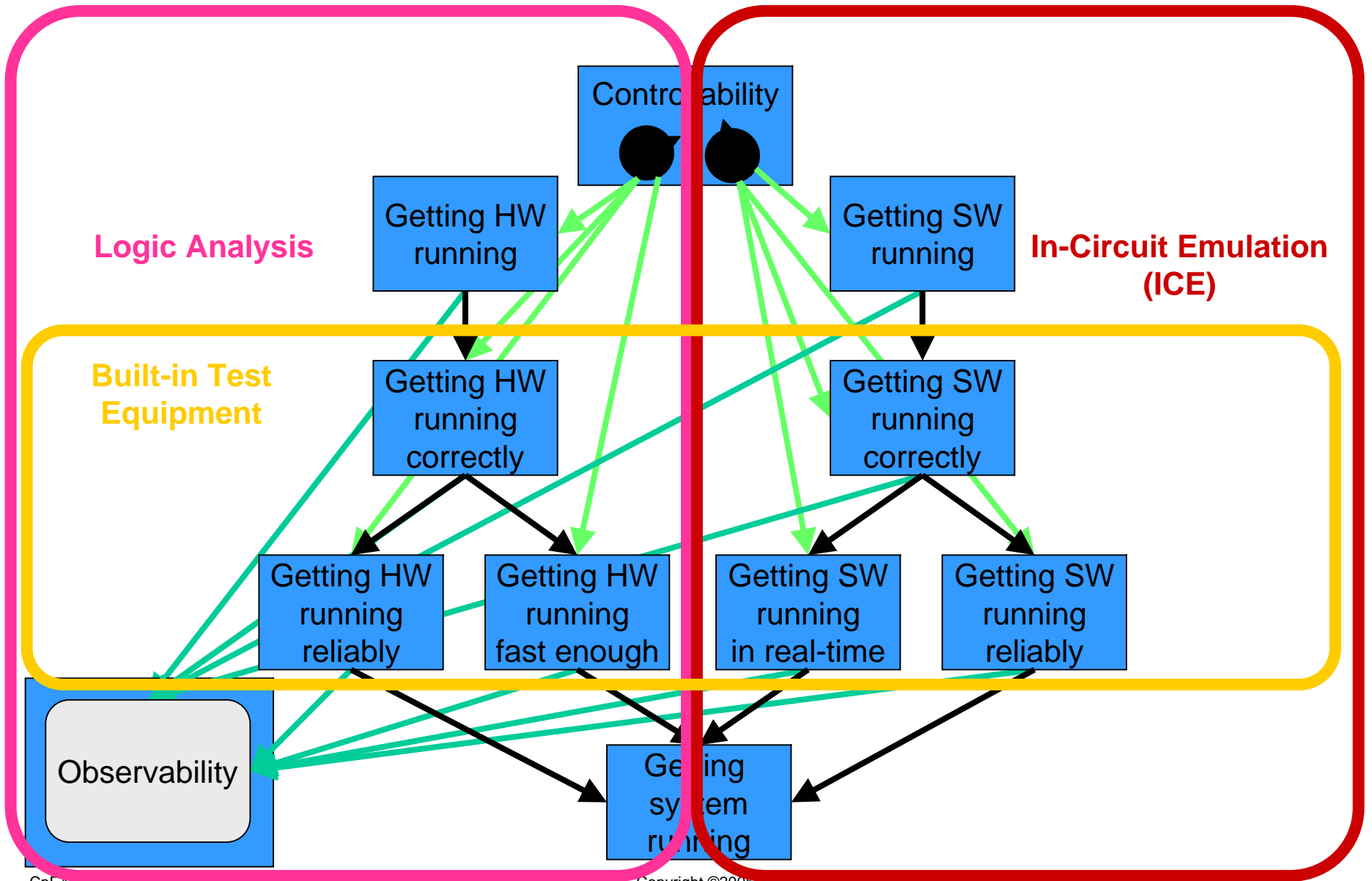


Getting a Real-Time Embedded System Running

- Evaluating performance and correctness



Getting a Real-Time Embedded System Running



Initial Conditions

- This code will probably work on your PC:

```
void main()
{
  int i;
  float sine(100);
  while (i++ != 100)
  {
    sine(i) = sin(i*2*pi/100)
  }
  do_something_with_sine_table(sine);
}
```

Initial Conditions

- This code will probably work on your PC:

```
void main()
{
  int i;
  float sine(100);
  while (i++ != 100)
  {
    sine(i) = sin(i*2*pi/100)
  }
  do_something_with_sine_table(sine);
}
```

- Results on an embedded processor:

Initial Conditions

- This code will probably work on your PC:

```
void main()
{
  int i;
  float sine(100);
  while (i++ != 100)
  {
    sine(i) = sin(i*2*pi/100)
  }
  do_something_with_sine_table(sine);
}
```

- Results on an embedded processor:
 - sine table size is 100

Initial Conditions

- This code will probably work on your PC:

```
void main()
{
    int i;
    float sine(100);
    while (i++ != 100)
    {
        sine(i) = sin(i*2*pi/100)
    }
    do_something_with_sine_table(sine);
}
```

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100

Initial Conditions

- This code will probably work on your PC:

```
void main()
{
  int i;
  float sine(100);
  while (i++ != 100)
  {
    sine(i) = sin(i*2*pi/100)
  }
  do_something_with_sine_table(sine);
}
```

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100

Initial Conditions

- This code will probably work on your PC:

```
void main()
{
  int i;
  float sine(100);
  while (i++ != 100)
  {
    sine(i) = sin(i*2*pi/100)
  }
  do_something_with_sine_table(sine);
}
```

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100
 - program runs forever

Initial Conditions

- This code will probably work on your PC:

```
void main()
{
  int i;
  float sine(100);
  while (i++ != 100)
  {
    sine(i) = sin(i*2*pi/100)
  }
  do_something_with_sine_table(sine);
}
```

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100
 - program runs forever
 - program crashed unexpectedly

Initial Conditions

- This code will probably work on your PC:

```
void main()
{
  int i;
  float sine(100);
  while (i++ != 100)
  {
    sine(i) = sin(i*2*pi/100)
  }
  do_something_with_sine_table(sine);
}
```

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100
 - program runs forever
 - program crashed unexpectedly
 - first, second and Nth run of program give different results

Initial Conditions

- This code will probably work on your PC:

```
void main()
{
  int i;
  float sine(100);
  while (i++ != 100)
  {
    sine(i) = sin(i*2*pi/100)
  }
  do_something_with_sine_table(sine);
}
```

What is the initial value of i?

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100
 - program runs forever
 - program crashed unexpectedly
 - first, second and Nth run of program give different results

Initial Conditions

- This code will probably work on your PC:

```
void main()
{
  int i;
  float sine(100);
  while (i++ != 100)
  {
    sine(i) = sin(i*2*pi/100)
  }
  do_something_with_sine_table(sine);
}
```

What is the initial
value of i?

PC's OS probably initialized
memory to 0
Embedded system does whatever
you programmed

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100
 - program runs forever
 - program crashed unexpectedly
 - first, second and Nth run of program give different results

Initial Conditions

- This code will probably work on your PC:

```
void main()
{
  int i;
  float sine(100);
  while (i++ != 100)
  {
    sine(i) = sin(i*2*pi/100)
  }
  do_something_with_sine_table(sine);
}
```

What is the initial value of i?

PC's OS probably initialized memory to 0

Embedded system does whatever you programmed

Next execution of program picks up where previous execution left off.

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100
 - program runs forever
 - program crashed unexpectedly
 - first, second and Nth run of program give different results

System Initialization

- Embedded programmer must initialize:
 - All variables that are declared
 - All peripheral controls
 - Stack location
 - Initial stack pointer
 - .
 - .
 - .

Memory Leakage

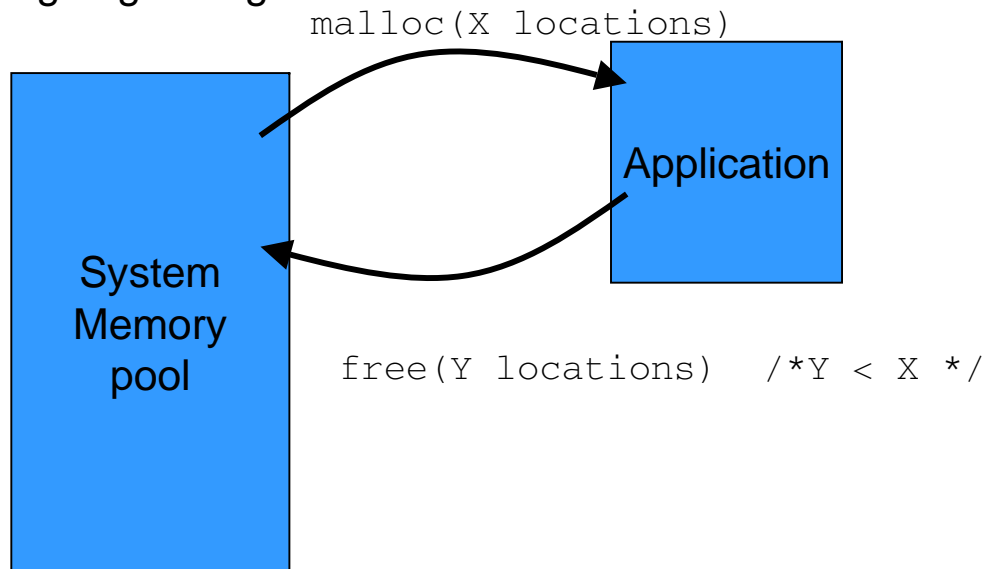
- Try this at home:
 1. Open N instances of your favorite application (Netscape, Explorer, Matlab, Word, etc.)
 2. Close them all
 3. Repeat until Windows complains about being short of memory, handles, or other resource.

Memory Leakage

- Try this at home:
 1. Open N instances of your favorite application (Netscape, Explorer, Matlab, Word, etc.)
 2. Close them all
 3. Repeat until Windows complains about being short of memory, handles, or other resource.
- What is going wrong?

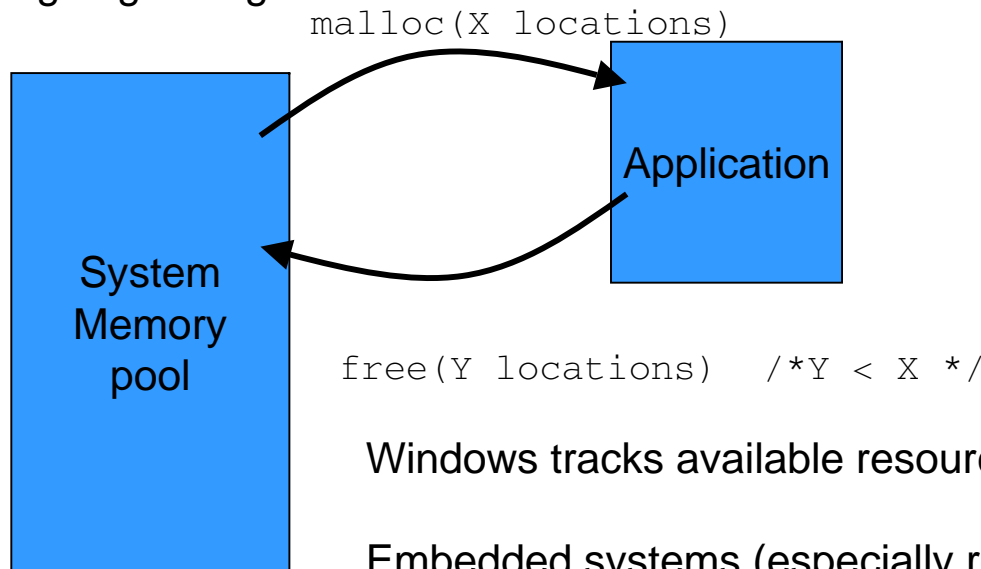
Memory Leakage

- Try this at home:
 1. Open N instances of your favorite application (Netscape, Explorer, Matlab, Word, etc.)
 2. Close them all
 3. Repeat until Windows complains about being short of memory, handles, or other resource.
- What is going wrong?



Memory Leakage

- Try this at home:
 1. Open N instances of your favorite application (Netscape, Explorer, Matlab, Word, etc.)
 2. Close them all
 3. Repeat until Windows complains about being short of memory, handles, or other resource.
- What is going wrong?



Windows tracks available resources and still gets into trouble

Embedded systems (especially real-time systems)
don't track resources

Embedded System Constraints

- Recursion: a simple way to write iterative code

```
long int factorial(int i)
{
  if(i == 1)
  {
    return(1);
  }
  else
  {
    return(i*factorial(i-1));
  }
}
```

Embedded System Constraints

- Recursion: a simple way to write iterative code

```
long int factorial(int i)
{
  if(i == 1)
  {
    return(1);
  }
  else
  {
    return(i*factorial(i-1));
  }
}
```

- What does this require from system?

Embedded System Constraints – Stack Overflow

- Recursion: a simple way to write iterative code

```
long int factorial(int i)
{
    if(i == 1)
    {
        return(1);
    }
    else
    {
        return(i*factorial(i-1));
    }
}
```

- What does this require from system?
 - At each call of factorial(), return address and machine state are pushed on stack
 - There is no protected memory, so stack grows without restriction

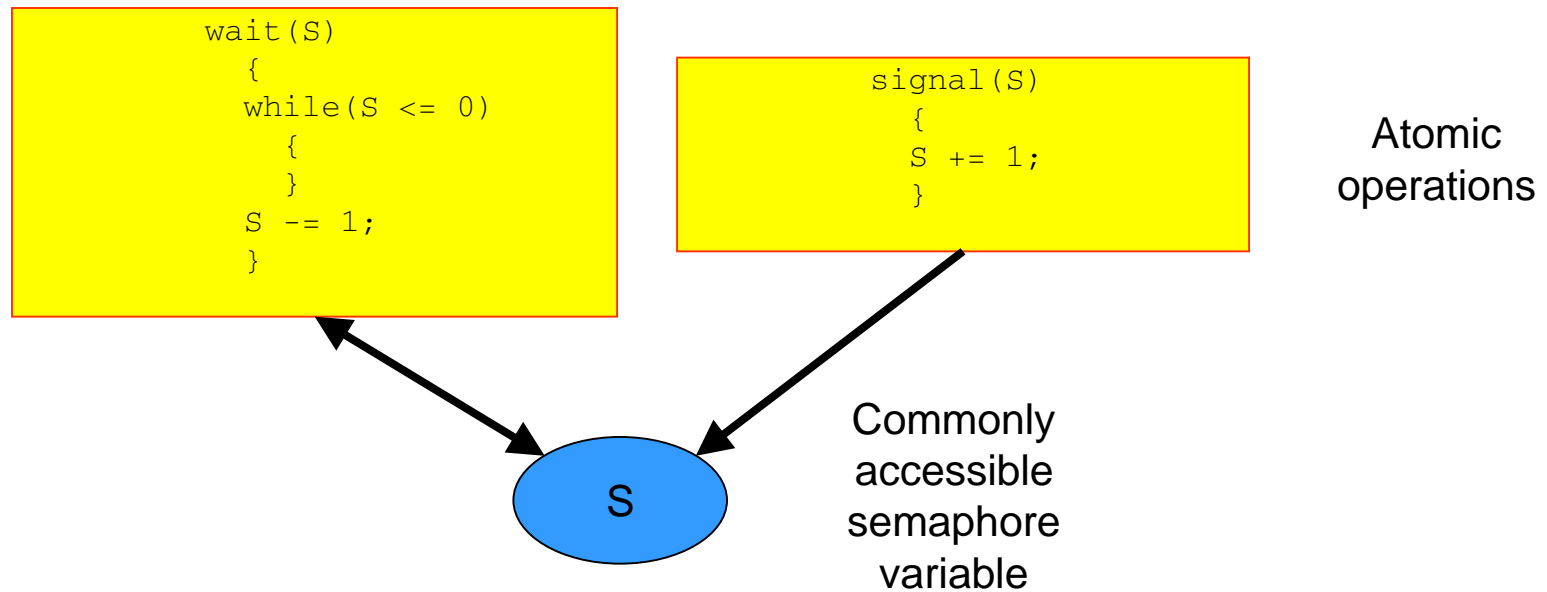
Embedded System Constraints – Stack Overflow

- Recursion: a simple way to write iterative code

```
long int factorial(int i)
{
    if(i == 1)
    {
        return(1);
    }
    else
    {
        return(i*factorial(i-1));
    }
}
```

- What does this require from system?
 - At each call of factorial(), return address and machine state are pushed on stack
 - There is no protected memory, so stack grows without restriction
 - Until it collides with data, program code, or peripheral

Semaphores for Interprocess Communications



Volatility

- Consider this code:

```
void main()
{
    int semaphore_S;
    do_something();
    wait(&semaphore_S);
    do_something_else();
}
```

```
void wait(int *S) /* pass the semaphore's address to wait() */
{
    while(*S <= 0)
    {
    }
    *S -= 1;
}
```

```
signal(S)
{
    S += 1;
}
```

Other process



Volatility

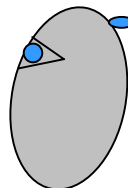
- Consider this code:

```
void main()
{
    int semaphore_S;
    do_something();
    wait(&semaphore_S);
    do_something_else();
}
```

```
void wait(int *S) /* pass address to wait() */
{
    while(*S <= 0)
    {
    }
    *S -= 1;
}
```

```
signal(S)
{
    S += 1;
}
```

S is not modified in while() loop. I can improve this!



Compiler

Volatility

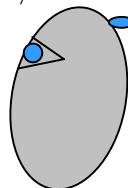
- Consider this code:

```
void main()
{
    int semaphore_S;
    do_something();
    wait(&semaphore_S);
    do_something_else();
}
```

```
void wait(int *S) /* pass address to wait() */
{
    /* while(*S <= 0) */
    if(*S <= 0)
    {
        halt();
    }
    *S -= 1;
}
```

```
signal(S)
{
    S += 1;
}
```

S is not modified in while() loop. I can improve this!



Compiler

Volatility

- Consider this code:

```
void main()
{
    int semaphore_S;
    do_something();
    wait(&semaphore_S);
    do_something_else();
}
```

```
signal(S)
{
    S += 1;
}
```

```
void wait(volatile int *S)    /* pass the semaphore's address to wait() */
{
    while(*S <= 0)
    {
    }
    *S -= 1;
}
```

Variable must be considered to be asynchronously modified

Assignment 7

- Enjoy Spring Break