ECE 455 – Module 2

Worst-case Execution Time

Spring 2023

Murray Dunne – mdunne@uwaterloo.ca

Slides Acknowledgement

Some material in these slides is based on slides from:

- Prof. Sebastian Fischmeister
- Prof. Carlos Moreno

Other material used with citations

Motivation

- A real-time system has to interact with the world in real time
  - How long something takes has physical consequences

- Need to know best-case and worst-case execution times
  - In practice: best-case is usually pointless
    - Useful for calculating jitter in network scenarios

Disaster of the Day – Radio Bricked Mazdas (2022)

Formal definition

- The **Worst-case Execution Time** T of some code *P* with input *x* in environment *w* is

$$T = \max_{x,w} f_{P(x,w)}$$

  where

$$f_{P(x,w)}$$

  is the duration of *P* run on *x* in *w* as measured in physical units of time.

- The **Best-case Execution Time** B of some code *P* with input *x* in environment *w* is

$$B = \min_{x,w} f_{P(x,w)}$$

Practical definition

- An estimate T' of the **Worst-case Execution Time** of some code *P* with input *x* in environment *w* is

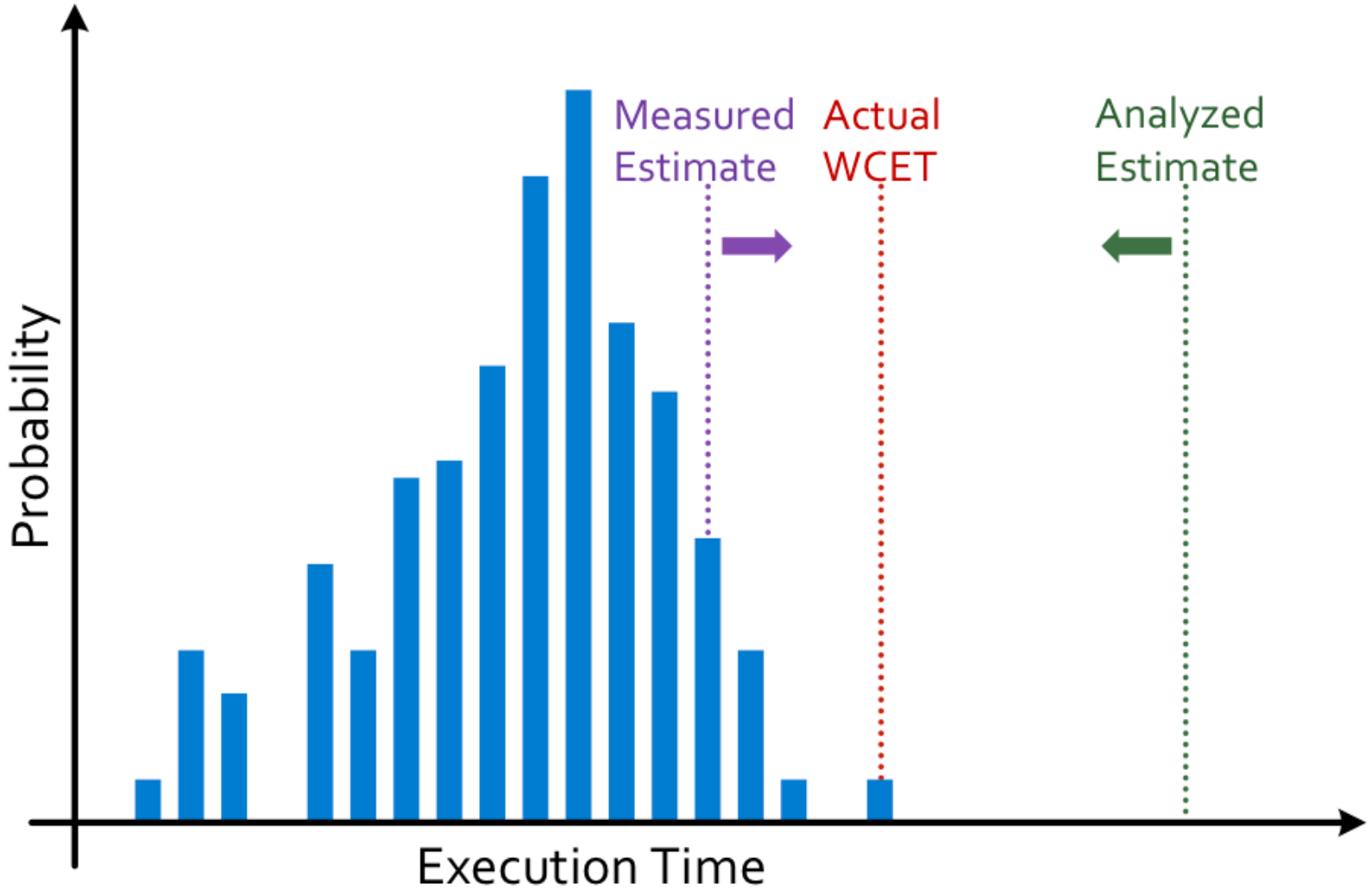
$$\forall x, w : f_{P(x,w)} \leq T'$$

- An estimate B' of the **Best-case Execution Time** of some code *P* with input *x* in environment *w* is

$$\forall x, w : f_{P(x,w)} \geq B'$$

- Necessarily we have $T \leq T'$ and $B \geq B'$

Practicalities of Measuring WCET



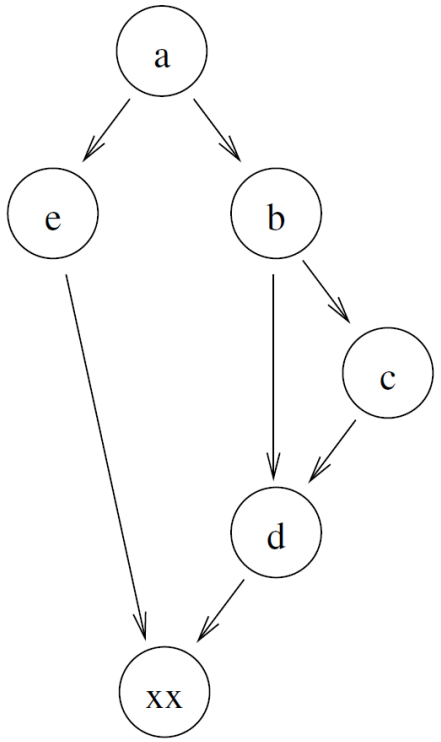
Path-based WCET Estimation

- Based on the longest possible path through the code
- Enumerate/explore the path space of the code
  - Analyze **Control Flow Graph**

Control Flow Graphs

- A graph $G = (V, E)$ where
  - V comprises the **basic blocks** of the program
  - E indicates the flow of control between those basic blocks

```
if (a) {
     if (b) {
          c;
     }
     d;
} else {
     e;
}
```



Control Flow Graphs

- Obtained from source code
- Or from machine code through:
  - **Static Analysis**
    - Look at the static machine code generated by the compiler
    - Weaknesses?
  - **Dynamic Analysis**
    - Record branch instructions and how they connect blocks at runtime
    - Weaknesses?

Loop Bounds

- How many iterations will a loop take to terminate?
  - Real-time systems must have terminating loops
    - Cannot make execution guarantees without them
  - Static analysis cannot find all possible loop bounds for Turing machines
    - Halting problem
  - Requires very simple loops
    - Or explicit developer annotation

```
for ( int i=0; i<=10; i++ ) {
  a++
}
```

```
void myfun (int x) {
  int i=0;
  while ((i*1.5) < (x+i))
     i=i+2;
}
```

Path Space

- The space of all execution paths in a program
  - Grows exponentially with conditionals in the program

```
#define MAXSIZE 100
int Array[MAXSIZE][MAXSIZE];
int Ptotal, Pcnt, Ntotal, Ncnt;
...
void count() {
  int Outer, Inner;
  for (Outer = 0; Outer < MAXSIZE; Outer++) {
    for (Inner = 0; Inner < MAXSIZE; Inner++) {
      if (Array[Outer][Inner] >= 0) {
        Ptotal += Array[Outer][Inner];
        Pcnt++;
      } else {
        Ntotal += Array[Inner][Outer];
        Ncnt++;
      }
}}}
```

Path Feasibility

- Not all paths can actually be taken
  - We only care about **Feasible Paths**
    - Reduces number of paths to consider
    - Can be reduced to the satisfiability problem
      - NP-complete
      - Very good tooling support for reasonable cases!

- Path-based WCET estimation finds the *longest feasible path*

Feasible Paths Example

```
#define PPRZ_MODE_AUTO2 2
#define PPRZ_MODE_HOME 3
#define VERTICAL_MODE_AUTO_ALT 3
#define CLIMB_MAX 1.0
...
void altitude_control_task(void) {
  if (pprz_mode == PPRZ_MODE_AUTO2 || pprz_mode == PPRZ_MODE_HOME) {
    if (vertical_mode == VERTICAL_MODE_AUTO_ALT) {
      float err = estimator_z - desired_altitude;
      desired_climb = pre_climb + altitude_pgain * err;
      if (desired_climb < -CLIMB_MAX) {
        desired_climb = -CLIMB_MAX;
      }
      if (desired_climb > CLIMB_MAX) {
        desired_climb = CLIMB_MAX;
      }
}}}
```

Disaster of the Day – Airbus A350 (2017)

Wikimedia Leo067 - https://commons.wikimedia.org/wiki/File:A350_Berlin_ILA_18.jpg

Path-based WCET Estimation Demo

- (maybe, time permitting)

Disaster of the Day – Spirit (2004)

Environmental Effects

- Memory hierarchy
  - What if stuff is in a cache? Which cache?
    - Assume all accesses are cache misses
    - Accurately predicting caching behavior is very hard

- Architecture effects
  - Pipelining
    - Details of stalls in your processor’s manual
  - Bus contention
    - Bus implementation details in your processor’s manual

Environmental Effects Continued

- Interrupts
  - Interrupt overhead time
  - Maskability
    - Which interrupts can fire in the context of the code you're running?
  - Sum of interference time
    - Minimum interarrival time of interrupt i is a_i
    - WCET of interrupt i's code (including overhead) t_i
    - WCET estimate of code you're considering before counting interrupts C'
    - Add in interrupt time to get updated estimate T'

$$T' = C' + \sum_i t_i \left\lceil \frac{C'}{a_i} \right\rceil$$

Environmental Effects Continued 2

- Wait, there's a problem!
  - Adding interrupts to the time adds total time
    - Which means more possible interrupts in that time!
    - This problem is recursive
- Replace C' in sum with T'_{n-1}
  - Minimum interarrival time of interrupt i is a_i
  - WCET of interrupt i's code (including overhead) t_i
  - WCET estimate of code you're considering before counting interrupts C'
  - Last recursion value is T'_{n-1} with initial estimate $T'_0 = C'$
  - Add in interrupt time to get updated estimate T'_n

$$T'_n = C' + \sum_i t_i \left\lceil \frac{T'_{n-1}}{a_i} \right\rceil$$

  - Stop when $T'_n = T'_{n-1}$
- Start at highest priority interrupt
  - Then repeat for lower priority, then lower still, until finally non-interrupt code

Implicit Path Enumeration (IPET)

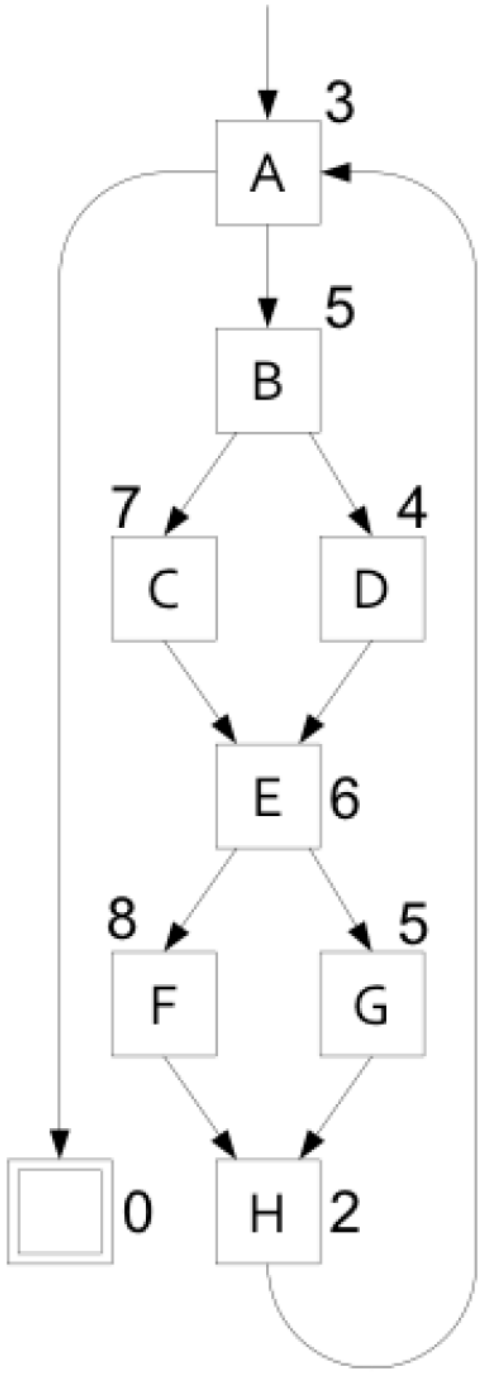
- Formulate path-based WCET estimation as Integer Linear Program (ILP)
  - Assume unique source and destination node
  - Variables x_i indicate execution count of block i
  - $x_1 = 1$ and $x_n = 1$
    - Source and destination only run once
  - $x_i = \sum_{j \in P_i} \chi_{ji} = \sum_{j \in S_i} \chi_{ij}$
    - P_i are the predecessor basic blocks of x_i
    - S_i are the successor basic blocks of x_i
  - Solve

$$\max_{x_i, 1 \leq i \leq n} \sum_{i=1}^{n} w_i x_i$$

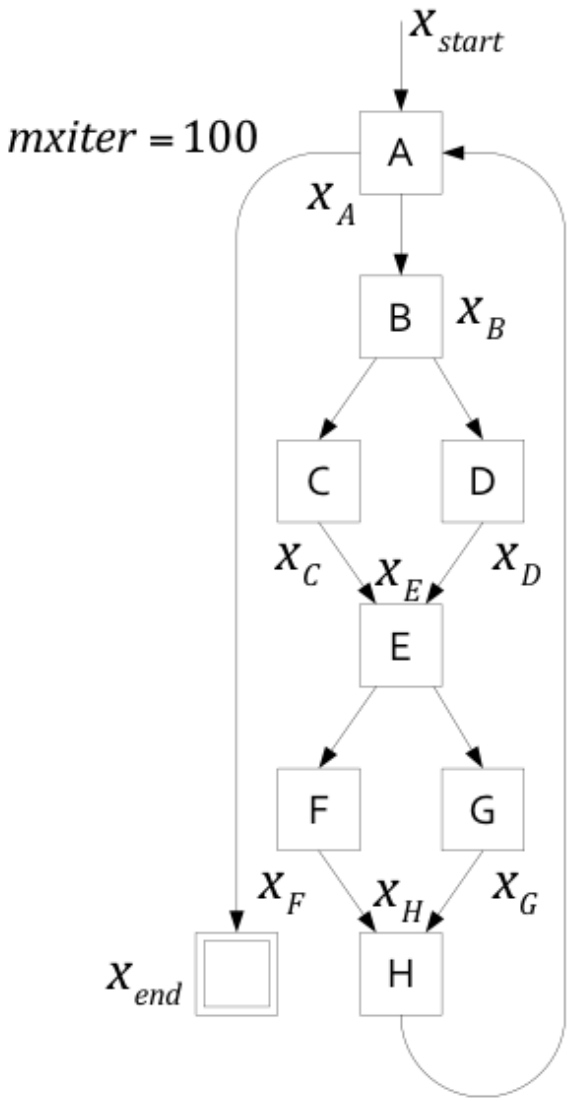
    - Don’t forget about environment w_i

IPET WCET Example

```
for (int i=0; i<100; i++) {
  if (B)
    C;
  else
    D;
  if (E)
    F;
  else
    G;

  H;
}
```



IPET WCET Example



$$t_A = 3$$
$$t_B = 5$$
$$t_C = 7$$
$$t_D = 4$$
$$t_E = 6$$
$$t_F = 8$$
$$t_G = 5$$
$$t_H = 2$$

$$x_{start} = 1$$
$$x_{end} = 1$$

$$x_{startA} = x_{start}$$
$$x_A = x_{startA} + x_{HA} = x_{Aend} + x_{AB}$$
$$x_B = x_{AB} = x_{BC} + x_{BD}$$
$$x_C = x_{BC} = x_{CE}$$
$$x_D = x_{BD} = x_{DE}$$
$$\vdots$$
$$x_H = x_{FH} + x_{GH} = x_{HA}$$

$$x_A \leq 101$$

$$T' = \max\left(x_A t_A + x_B t_B + \ldots + x_H t_H\right) = 3103$$

Disaster of the Day – Lauda Flight 004 (1989)

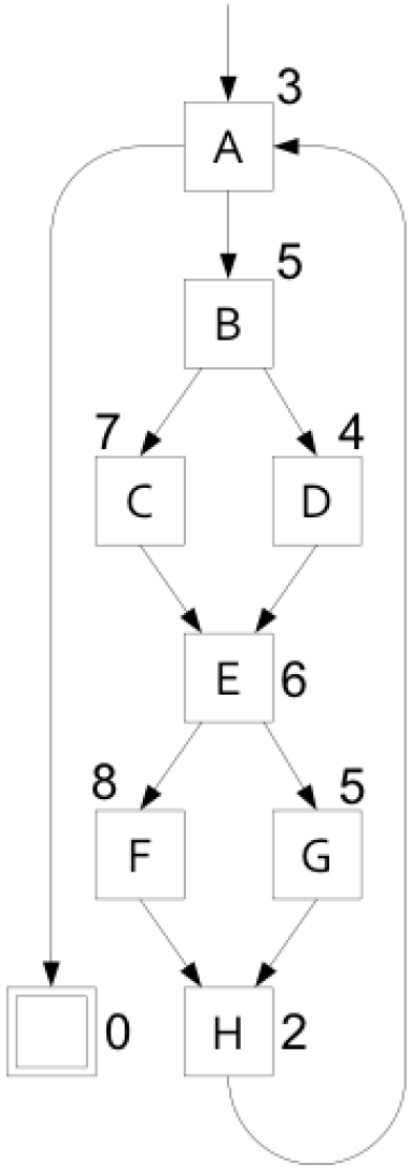
Disaster of the Day – Lauda Flight 004 (1989)

Structure-based WCET Estimation

- Composite basic blocks structures
- Build up WCET estimate with composite of composites

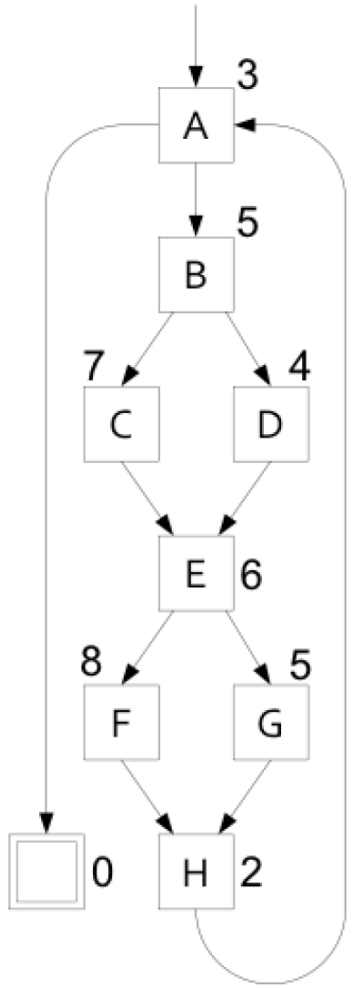
Structure-based WCET Example

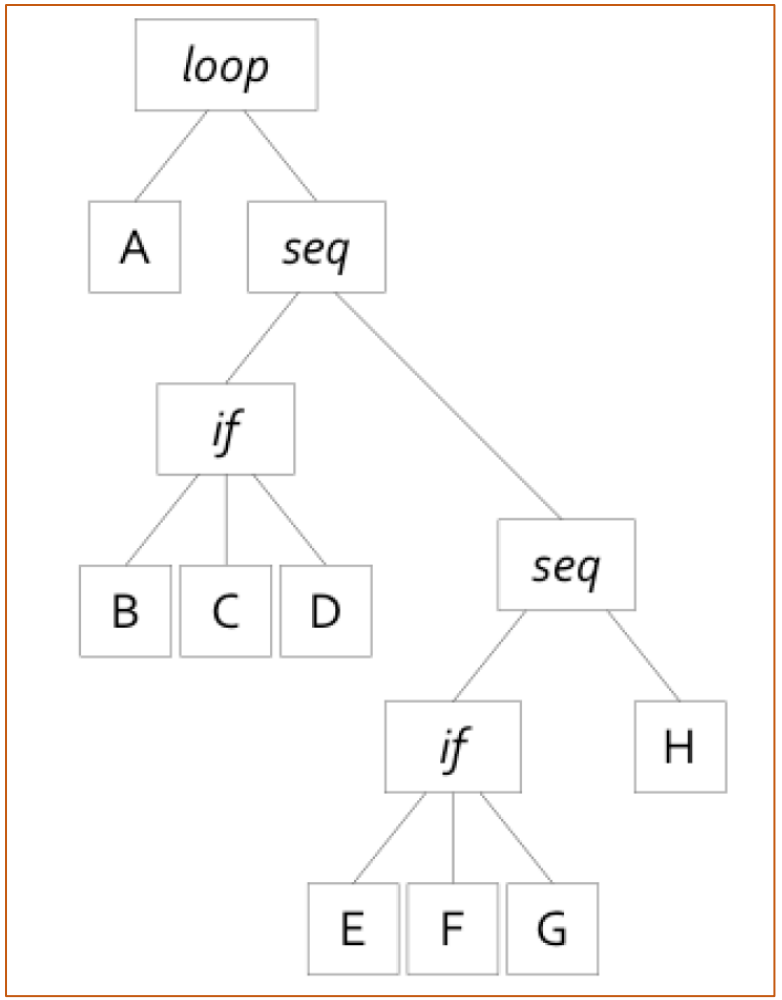
```
for (int i=0; i<100; i++) {
  if (B)
    C;
  else
    D;
  if (E)
    F;
  else
    G;

  H;
}
```



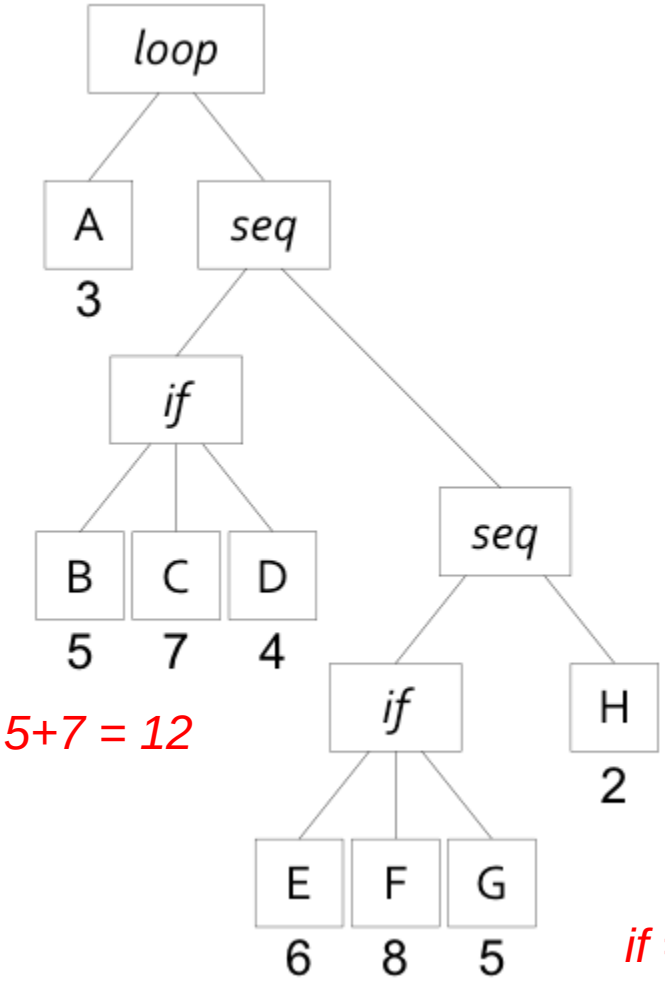
Structure-based WCET Example

```
for (int i=0; i<100; i++) {
  if (B)
    C;
  else
    D;
  if (E)
    F;
  else
    G;

  H;
}
```





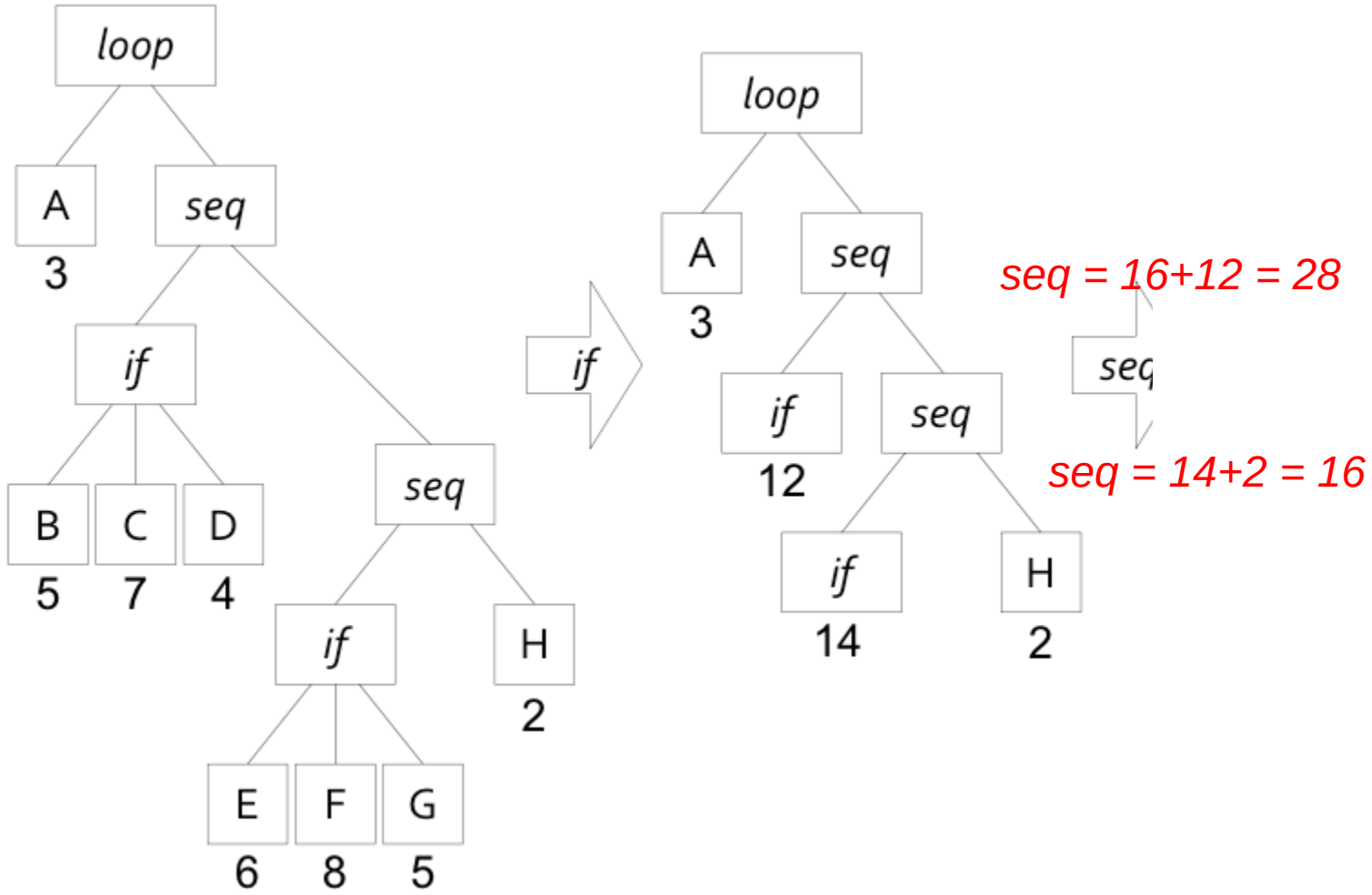
Structure-based WCET Example



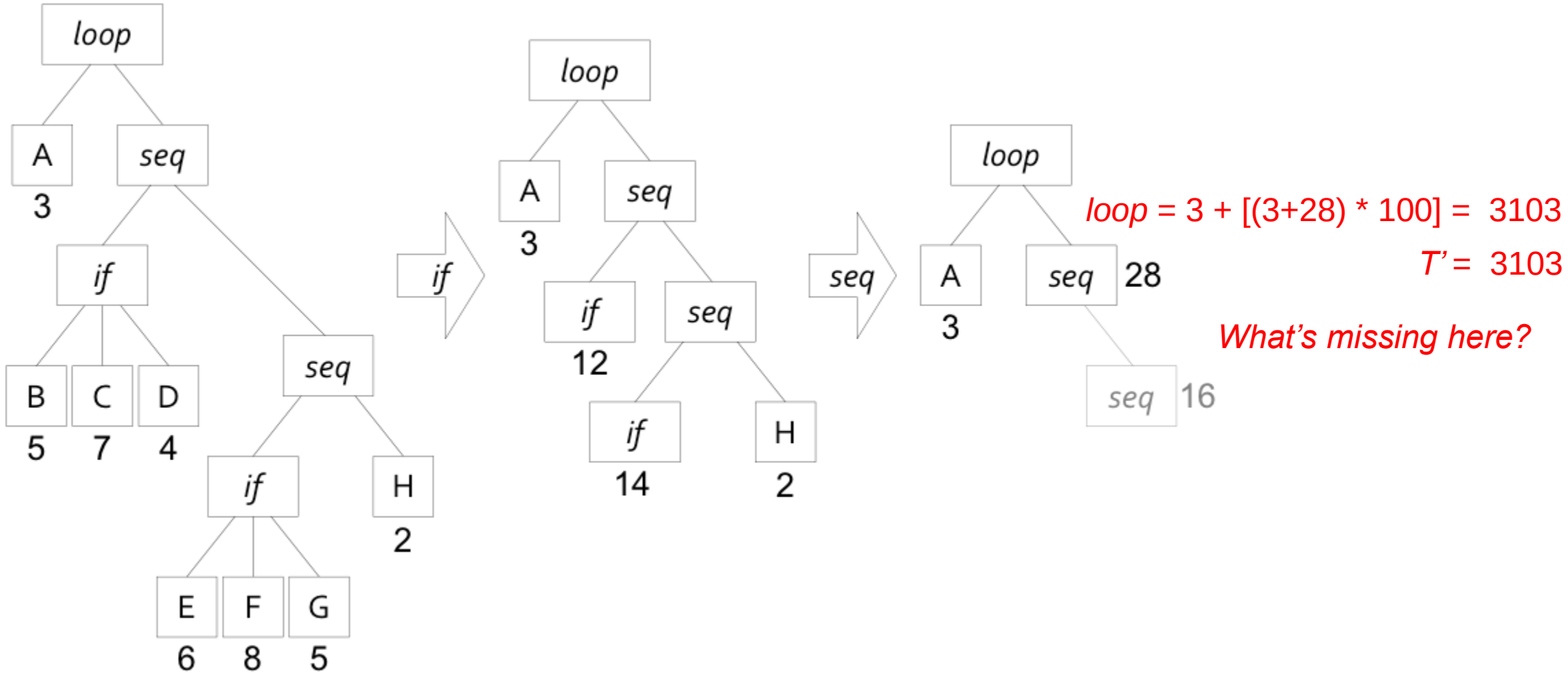
if = 5+7 = 12

if = 6+8 = 14

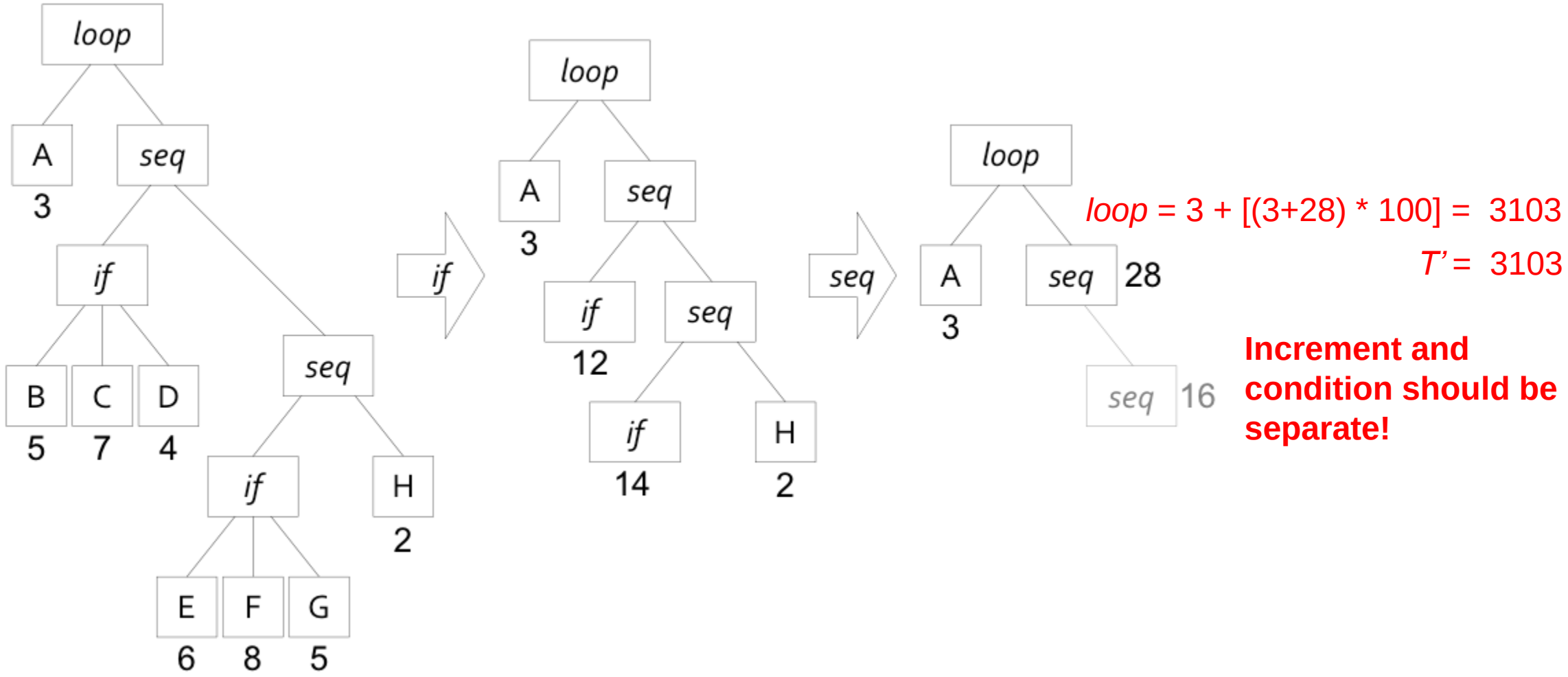
Structure-based WCET Example



Structure-based WCET Example



Structure-based WCET Example



Structure-based WCET Structures

- Sequence *seq*
  - Just add them
- If statement *if*
  - Take the condition plus the longer branch
- Loop *loop*
  - Initializer + condition + count * (condition + increment + body)
    - Condition runs one extra time
- No formal specification of structures
  - They should be obvious
  - If they’re not obvious, don’t use them in your code

Compiler Interference

- Optimizations will affect the generated code
  - Actually check what the compiler did
- Loop unrolling
  - May remove loop counter and replace with fixed values
  - May group iterations into batches
- Enable and disable optimizations individually
  - Instead of just –O2

Measurement-based WCET Verification

- Don’t use measurement to calculate WCET!
  - Use it to double-check your calculated WCET

- Methods
  - CPU Clock
    - Checking takes time
    - Drift
  - Processor cycle counter
    - Checking takes time
    - Drift
    - Easier to compare to WCET
  - GPIO + Logic analyzer
    - I/O Latency

Disaster of the Day – Lufthansa Flight 2904 (1993)

Pipeline WCET Example

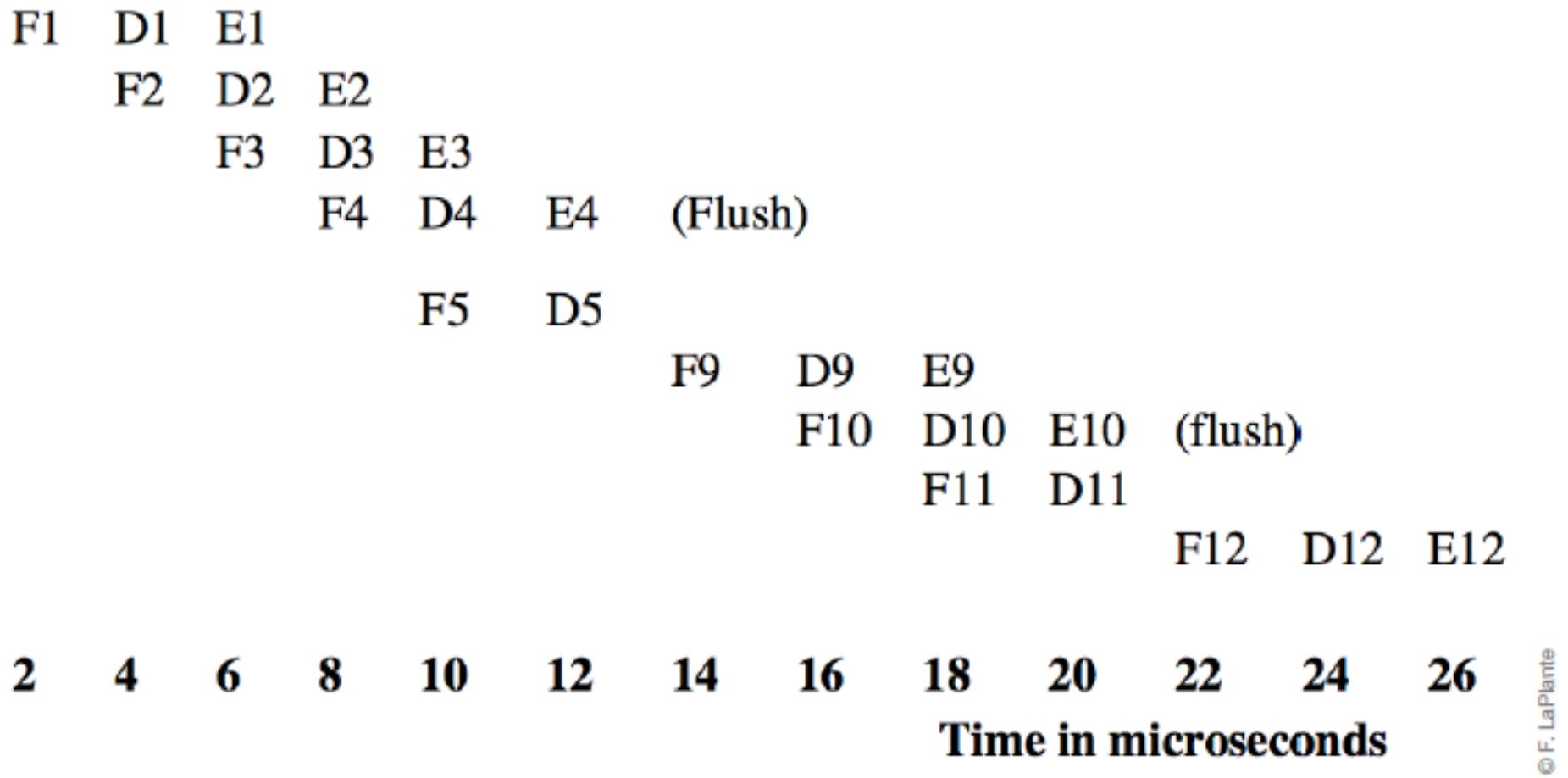
- Let's do an example, first we find the paths

```
1     LOAD R1, @a          ; R1 <-- contents of "a"
2     LOAD R2, @b          ; R2 <-- contents of "b"
3     TEST R1, R2          ; compare R1 and R2 , set condition code
4     JNE @L1              ; goto L1 if not equal
5     ADD R1, R2           ; R1 <-- R1 + R2
6     TEST R1,R2           ; compare R1 and R2 , set condition code
7     JGE @L2              ; goto L2 if R1 >= R2
8     JMP @END             ; goto END
9     @L1 ADD R1, R2       ; R1 <-- R1 + R2
10    JMP @END             ; goto END
11    @L2 ADD R1, R2       ; R1 <-- R1 + R2
12    @END SUB R2, R3      ; R2 <-- R2 - R3
```

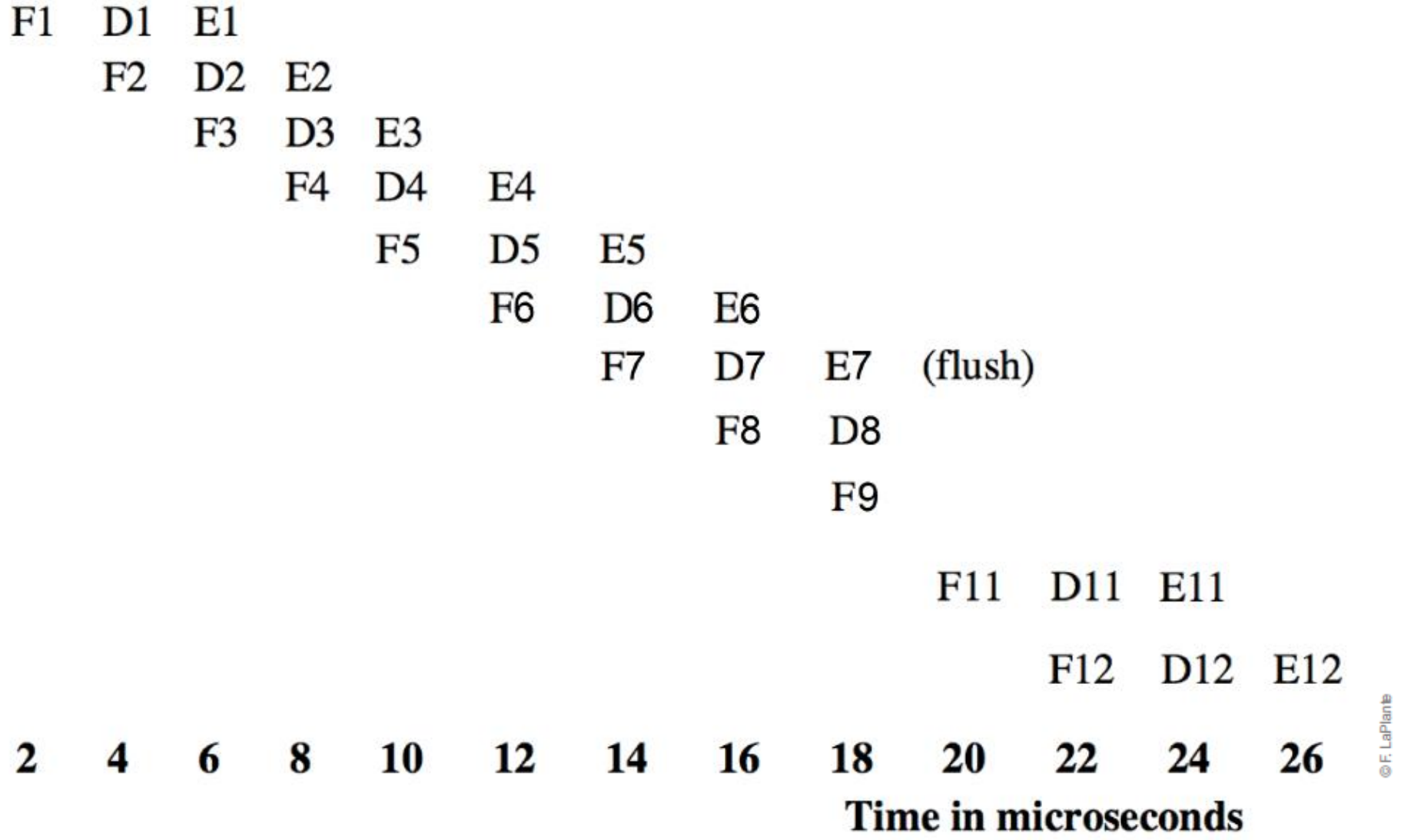
Pipeline WCET Example Cont.

- Let’s assume a naïve 6μs/instruction
- We have 3 paths:
  - Path 1: 7 instructions, 42μs
  - Path 2: 9 instructions, 54μs
  - Path 3: 9 instructions, 54μs
- Now what about a pipeline?
  - Let’s say it was a 3-stage pipeline with 2μs/cycle

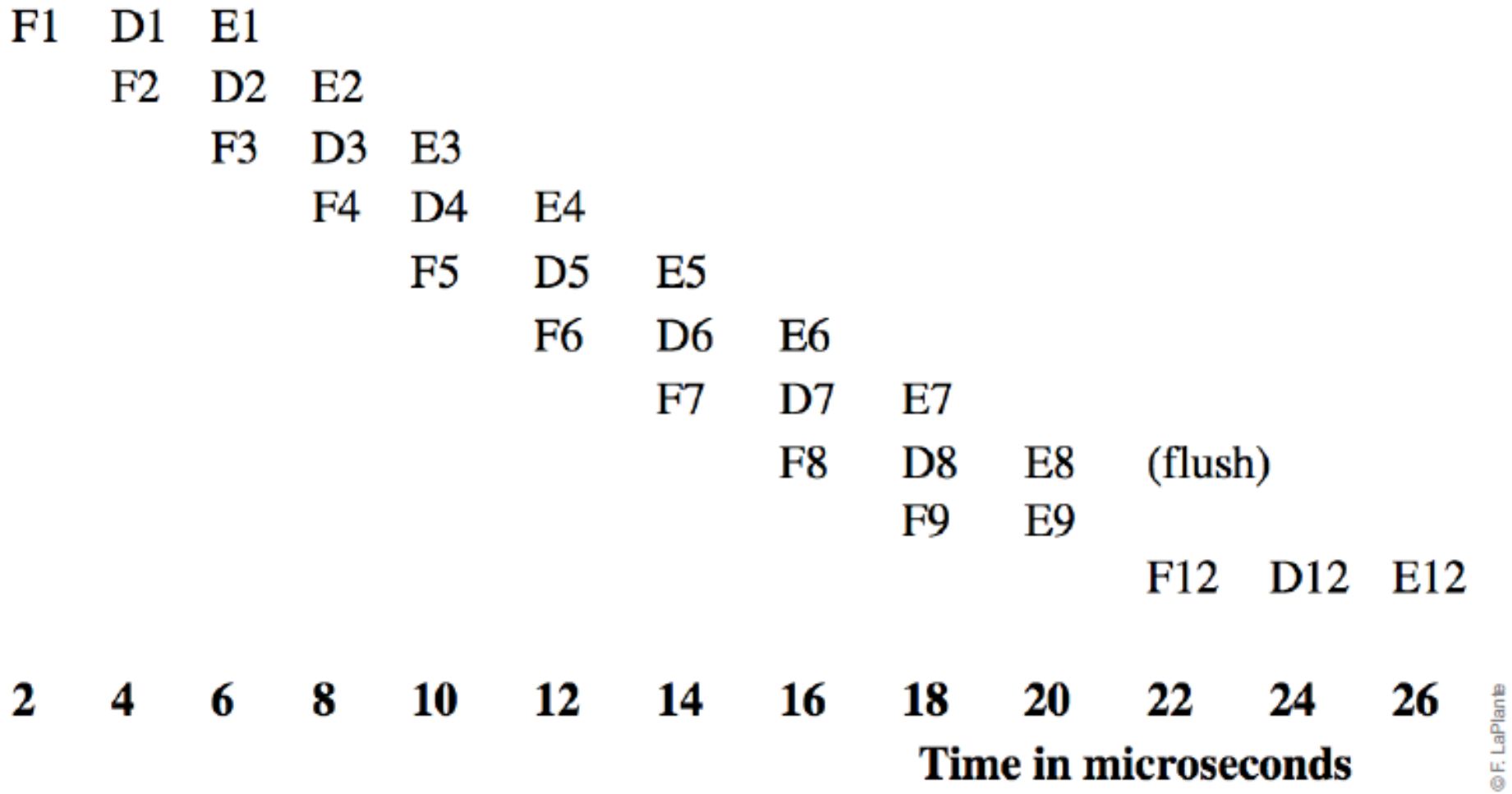
Pipeline WCET Example- Path 1



Pipeline WCET Example- Path 2



Pipeline WCET Example- Path 3



Pipeline WCET Example Cont. 2

- They were all 26µs!
  - This is just a coincidence
- This is very simplified!
  - No loops
  - Very simple pipeline
  - No caching
  - No interrupts

Throughput WCET Examples

Throughput examples from:

Lewis, Daniel Wesley. Fundamentals of Embedded Software: Where C and Assembly Meet with CDROM. Prentice Hall PTR, 2002. Chapter 6.

(with some small changes)

Throughput WCET Example - Polling

```
#define SERIAL_DATA_PORT 0x2F8
#define SERIAL_STATUS_PORT 0x2FD

/* Bits in the status port ... */
#define TX_READY (1 << 5) /* bit 5: 1= Transmitter holding register empty */
#define RX_READY (1 << 0) /* bit 0: 1= Input data ready ( available ) */

BYTE8 Serial_Input ( void ) {
      /* do nothing ( wait for data to arrive ) */
      while (( inportb ( SERIAL_STATUS_PORT ) & RX_READY ) == 0);
            return inportb ( SERIAL_DATA_PORT );
}

void Serial_Output ( BYTE8 ch) {
      /* do nothing ( wait for device to finish transmitting ) */
      while (( inportb ( SERIAL_STATUS_PORT ) & TX_READY ) == 0)
            ;
      outportb ( SERIAL_DATA_PORT , ch );
}
```

Throughput WCET Example - Polling

- Here are those same functions in assembly:

```
        MOV DX, 02FDh
SI1:    IN AL, DX
        TEST AL, 00000001B
        JZ SI1
        MOV DX, 02F8h
        IN AL, DX
        MOVZX EAX, AL
        RET
```

```
        MOV DX, 02FDh
S01:    IN AL, DX
        TEST AL, 00100000B
        JZ S01
        MOV AL, [ESP+4]
        MOV DX, 02F8h
        OUT DX, AL
        RET
```

- Memory delays for I/O likely the limiting factor
  - Let’s look closer

Throughput WCET Example - Polling

Assumptions: x86, 1 byte/instruction, 4 bytes per memory read, 60ns per memory access, 40MHz PCI bus for IO

```
        MOV DX, 02FDh
SI1:    IN AL, DX
        TEST AL, 00000001B
        JZ SI1
        MOV DX, 02F8h
        IN AL, DX
        MOVZX EAX, AL
        RET
```

	Instruction Bytes		Stack	I/O
	Opcode	Imm.	Bytes	Transfers
	1	2		
	1			1
	1	1		
	1	1		
	1	2		
	1			1
	1			
	1		4	
	8	6	4	2

Throughput WCET Example - Polling

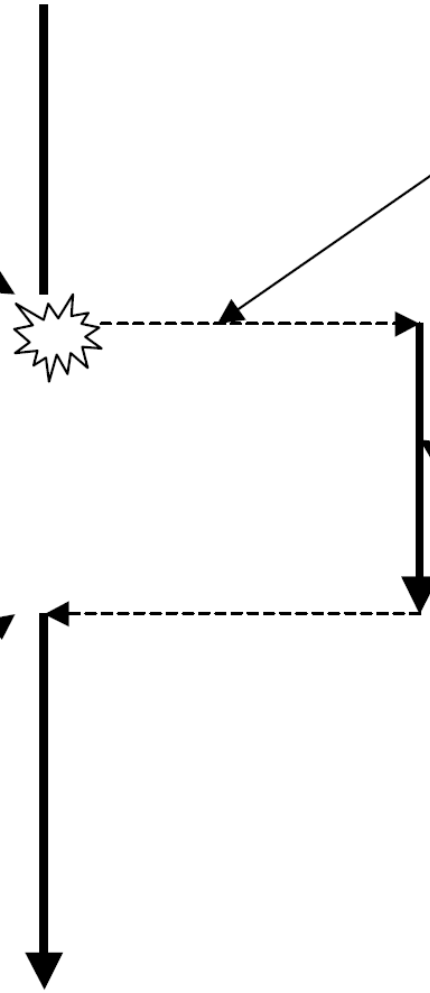
- 18 bytes accessed from memory (8+6 instruction, 4 stack)
  - $\left\lceil \frac{18}{4} \right\rceil$ = 5 memory accesses
- 5 x 60ns = 0.3µs for code/stack accesses
- 40MHz PCI for I/O gives 0.025µs per I/O read
  - 2x I/O reads give 0.05µs for the function
- 0.3µs + 0.05µs = 0.35µs per byte read
  - 2,857,142.86 B/s = **2.72 MB/s throughput**

- A common fast baud rate for serial devices is 115200 baud
  - That's 115200 bits/s = 14.06 kB/s = 0.01 MB/s < 2.72 MB/s
  - Our throughput is sufficient, but **latency is unknown**
    - When do we call this code?

Interrupt Handling (x86 Example)

Hardware interrupt Request occurs: CPU Finishes the current instruction and then initiates an interrupt response sequence.

Interrupt Response Sequences CPU pushes E-Flags and the return address (held in CS and EIP), disables interrupts, reads an interrupt type code from the requesting device, and transfers control to the corresponding Interrupt Service Routine.



Interrupt Complete: Interrupted code continues where it left off as if nothing happened.

Interrupt Service Routine:

1. Reenable higher priority interrupts.
2. Preserve CPU registers.
3. Transfer data (also clears the interrupt request).
4. Reenable lower priority interrupts.
5. Restore CPU registers.
6. Pop EIP, CS, and EFlags and return to interrupted code.

Throughput WCET Example - Interrupts

- What is before the software Interrupt Service Routine (ISR)?

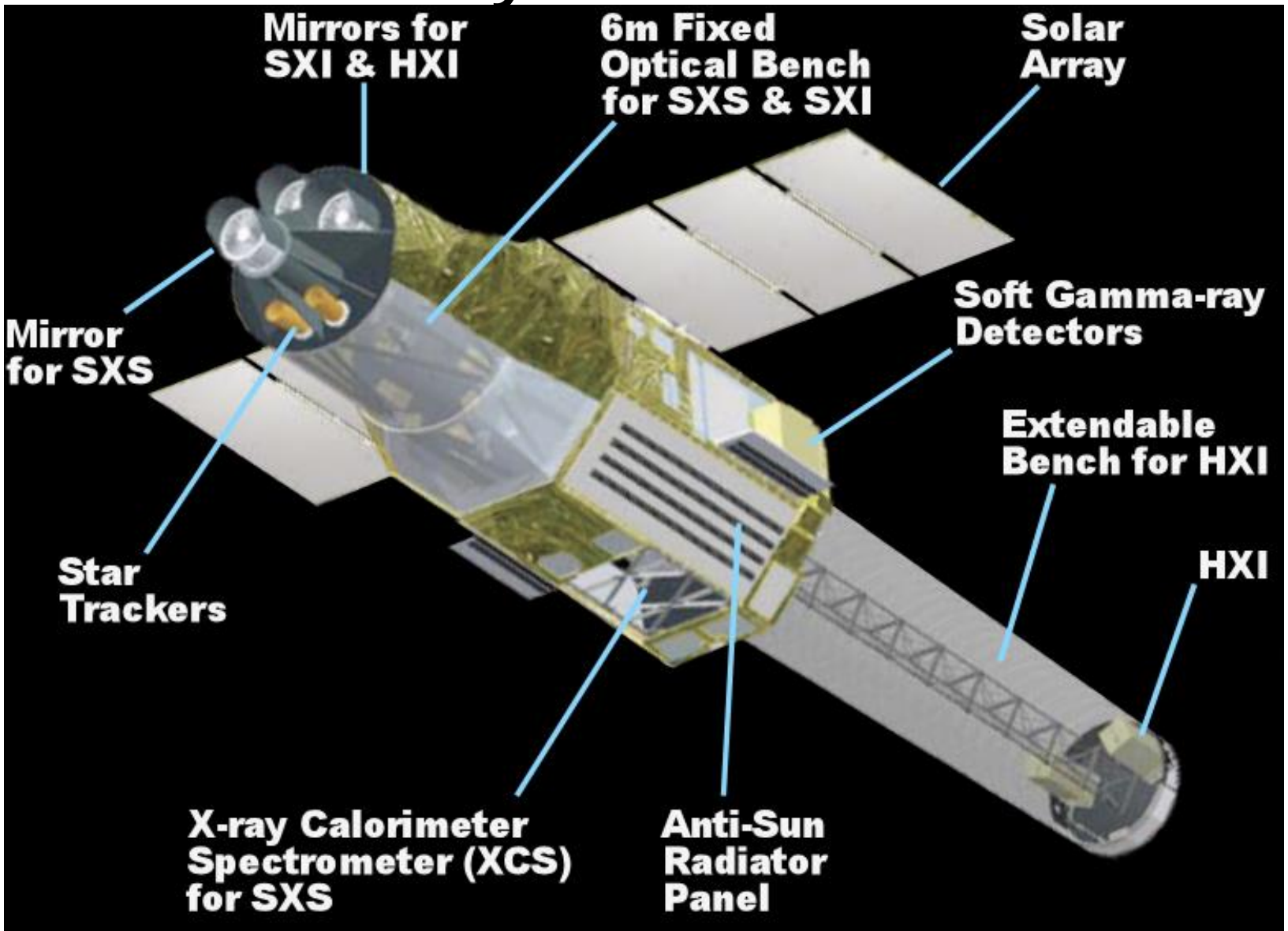
Step	Action	Bytes Transferred
1	Push EFlags register.	4 (stack write)
2	Disable interrupts.	n/a
3	Push return segment.	4 (stack write)
4	Push return offset.	4 (stack write)
5	Identify interrupt.	1 (I/O read)
6	Load CS and EIP	8 (IDT read) + 8 (GDT read)

- EIP = Program Counter
- EFLAGS = Program Status Register
- IDT = Interrupt Descriptor Table
- CS = Code Segment Register
- GDT = Global Descriptor Table

Throughput WCET Example - Interrupts

- 28 bytes total accessed from memory (12 stack, 8 IDT, 8 GDT)
  - $\left\lceil \frac{28}{4} \right\rceil$ = 7 memory accesses
- 7 x 60ns = 0.42μs for stack/IDT/GDT accesses
- There's 1 I/O transfer for the interrupt type code, 0.025μs
- Total 0.445μs interrupt delay??
- **WAIT! Interrupts can only happen on instruction boundaries!**
- The longest instruction (in this x86 processor) is PUSHA at 9 memory cycles = 0.54μs
- So worst-case interrupt delay is 0.985μs

Disaster of the Day – Hitomi Satellite (2016)



Throughput WCET Example - Interrupts

- Okay now lets look at the interrupt code itself:

			Instr.	Data	Stack	I/O
_Serial_Input_ISR:			Bytes	Bytes	Bytes	Transfers
STI		;Enable higher prior. Ints.	1			
PUSH	EAX	;Prepare	1		4	
PUSH	EDX	; of EAX and EDX.	1		4	
MOV	DX, 02FDH	;Retrieve the data and	3			
IN	AL, DX	; clear the request.	1			1
; Latency ends here... (I/O port has been read)						
MOV	[_serial_data], AL	;Save the data away.	5	1		
MOV	AL,00100000b	;Send EOI command to the	2			
OUT	20h, AL	; Prog. Interrupt Ctlr.	2			1
POP	EDX	;Restore original contents	1		4	
POP	EAX	; of the registers.	1		4	
IRET		;Restore CS, EIP and Eflags.	1	_	12+8(GDT)	_
			19	1	36	2

Throughput WCET Example - Interrupts

- 15 bytes total memory accesses before read (7 instrs, 8 stack)
  - $\left\lceil \frac{15}{4} \right\rceil$ = 4 memory accesses
- 4 x 60ns = 0.24µs for instr/stack accesses
- There's 1 I/O transfer for the interrupt type code, 0.025µs
- Total 0. 985µs + 0.24µs + 0.025µs = **1.25µs latency**
- Total across the whole interrupt: 56 memory, 2 I/O
- 14 x 60ns + 2 x 0.025µs = 0.89µs
- 0.985µs + 0.89µs = 1.875µs per byte read
  - 533,333.33 B/s = **0.509 MB/s throughput**
- A 115200baud serial device is 0.01 MB/s < 0.509 MB/s, still good!

Throughput WCET Example - DMA

- The DMA controller is an on-chip peripheral that can perform memory and I/O transfers asynchronously
  - No need for interrupts or polling
  - Can run at full bus speed
    - Minus anything else you might be using the bus for
    - At 60ns per 4-byte memory access that’s **63.58MB/s**

Throughput WCET Example - Conclusions

Method	Latency	Throughput
Polling	unknown	2.72MB/s
Interrupts	1.25µs	0.509MB/s
DMA	0.06µs (or config time)	63.58MB/s

- Takeaways:
  - The DMA is really good, use it!
  - Interrupts are your go-to when latency matters
    - They’re just more convenient than polling
      - But you really should be using the DMA for anything large
  - Polling has faster throughput than interrupts
    - But latency is hard to calculate and depends on other code
    - Use when data arrival is predictable
      - But you really should be using the DMA for large predictable transfers