# CS3210 Tut 2

## Programming Models & Performance of Parallel Systems

Slides yeeted from Sriram

# Question 1: Task Parallelism

# Question 1(a): Task Dependence Graphs

Fragment 1

```
parbegin
  do
    parbegin
      do
        parbegin
          do
            A
            C
          end
          parallel
          B
        parend
        F
      end
      parallel
      D
    parend
    G
  end
  parallel
  E
parend
H
```

What on earth is this about?

- `hx q1.c`
- `./q1`

# Question 1(a): Task Dependence Graphs

Fragment 1

```
parbegin
  do
    parbegin
      do
        parbegin
          do
            A
            C
          end
          parallel
          B
        parend
        F
      end
      parallel
      D
    parend
    G
  end
  parallel
  E
parend
H
```
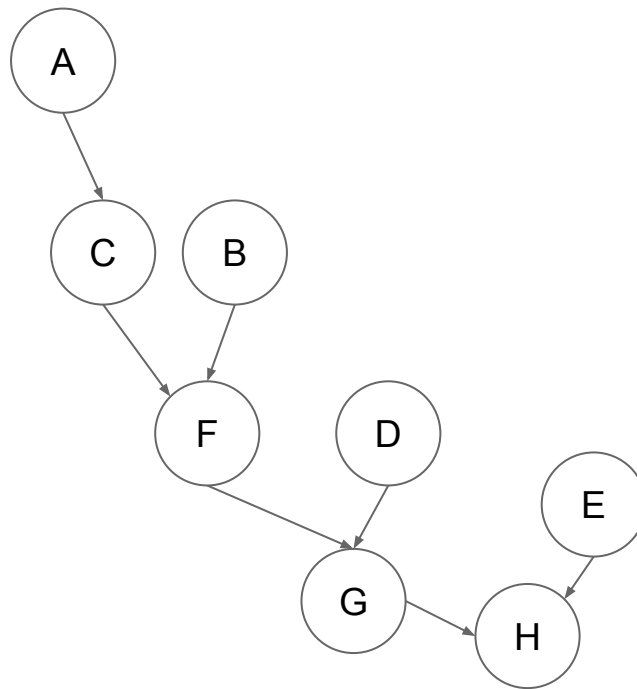
Let's find the dependencies!

Rules:

- **X**
  **Y**
  → X and Y are sequentially executed

- **X**
  **parallel**
  **Y**
  → X and Y are executed in parallel

- **parend**
  → All tasks must complete before moving on

# Question 1(a): Task Dependence Graphs

Fragment 1

```
parbegin
  do
    parbegin
      do
        parbegin
          do
            A
            C
          end
          parallel
          B
        parend
        F
      end
      parallel
      D
    parend
    G
  end
  parallel
  E
parend
H
```

Let's find the dependencies!

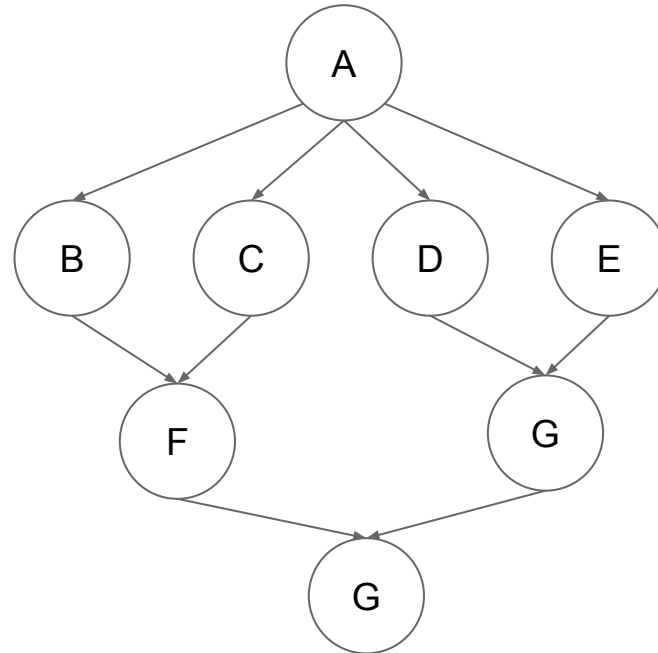# Question 1(a): Task Dependence Graphs

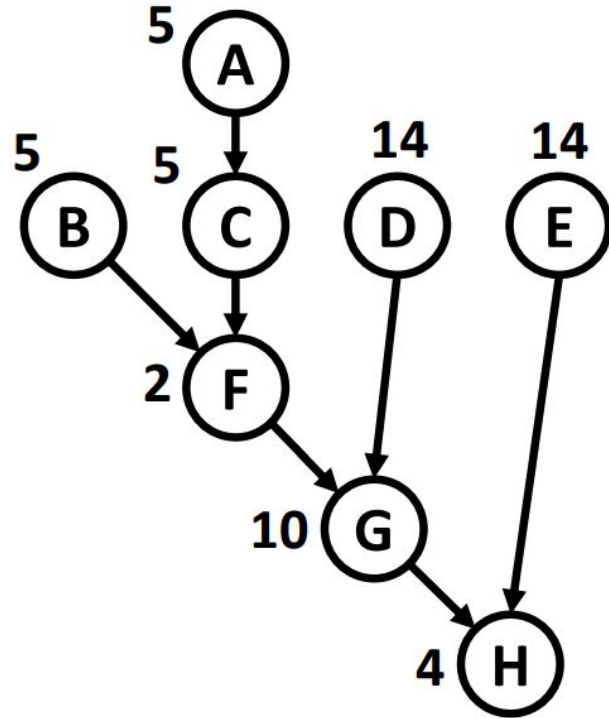Fragment 2

```
A
parbegin
  do
    parbegin
      B
    parallel
      C
    parend
    F
  end
  parallel
  do
    parbegin
      D
    parallel
      E
    parend
    G
  end
parend
H
```
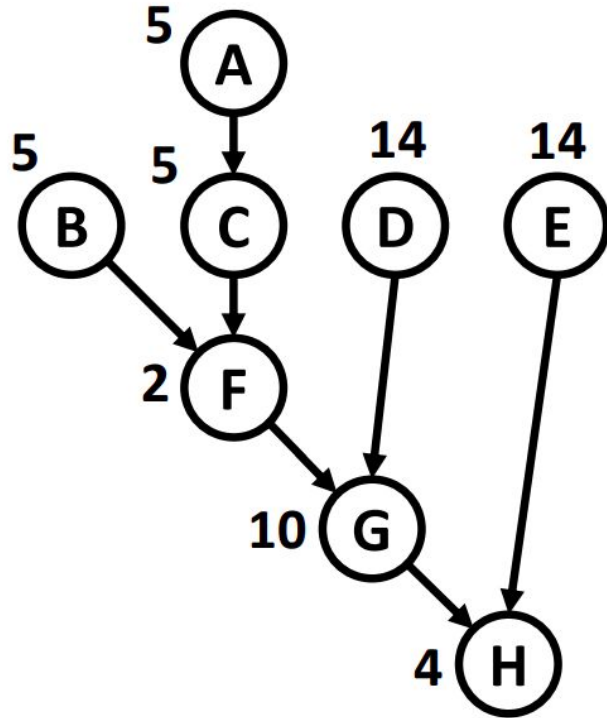
Fragment 2 Dependencies:

# Q1 (b): Amount of Concurrency

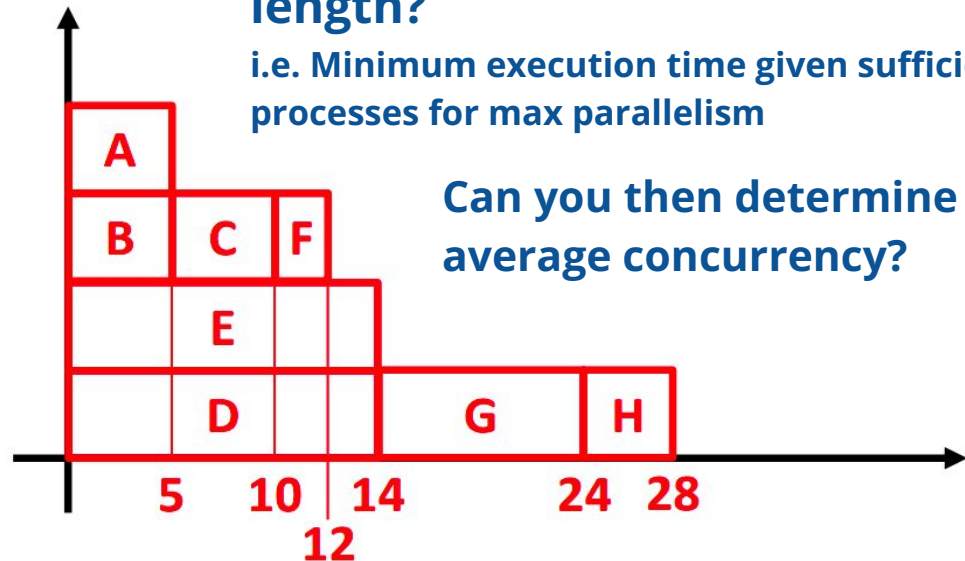

Plot the concurrency for Fragment 1

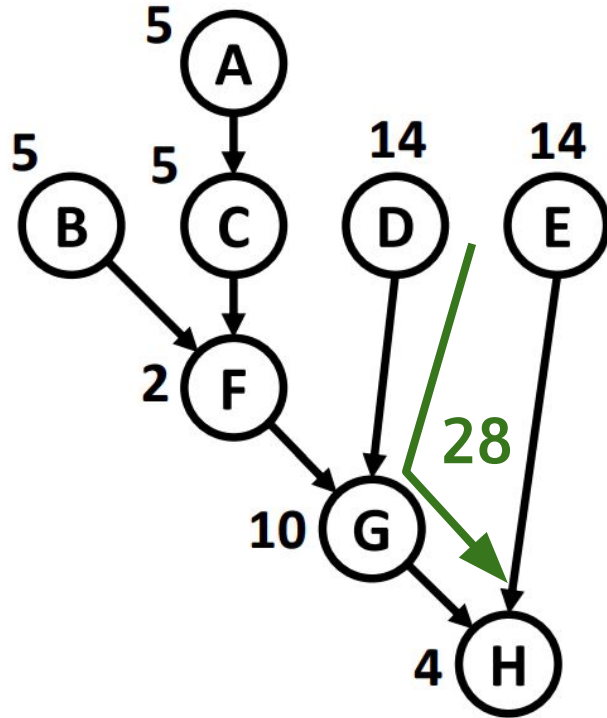# Q1 (b): Amount of Concurrency



Plot the concurrency for Fragment 1

**Can you determine the critical path length?**

i.e. Minimum execution time given sufficient processes for max parallelism

**Can you then determine the average concurrency?**

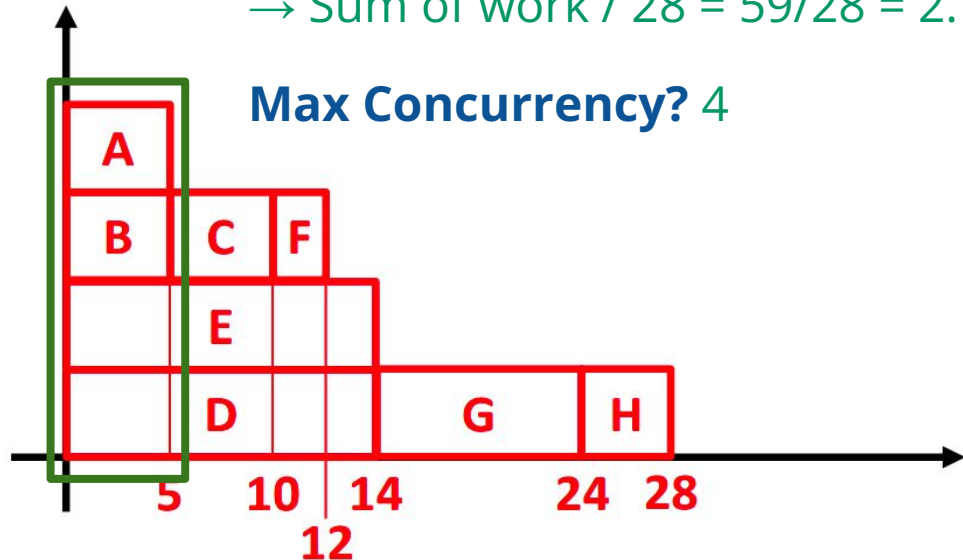# Q1 (b): Amount of Concurrency



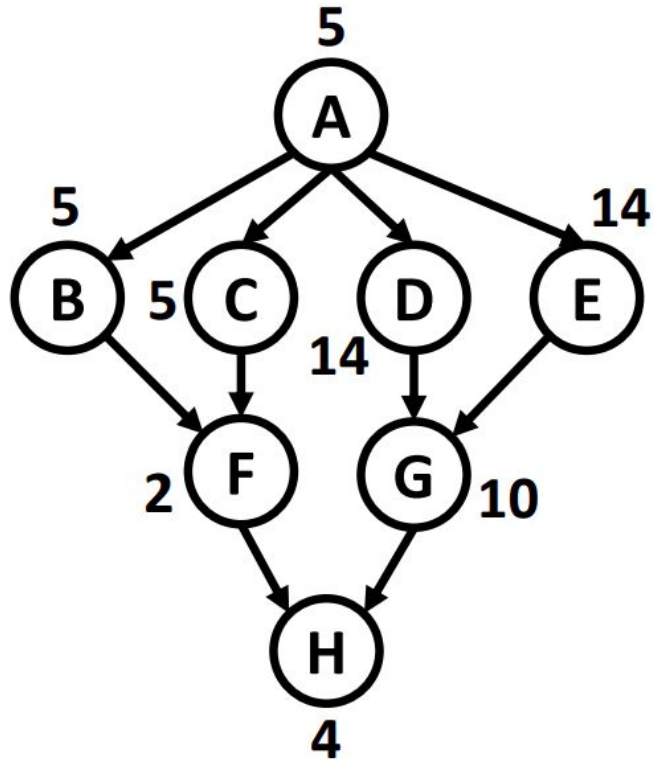**Critical path length?** 28

**Average Concurrency?**
→ Sum of work / 28 = 59/28 = 2.10

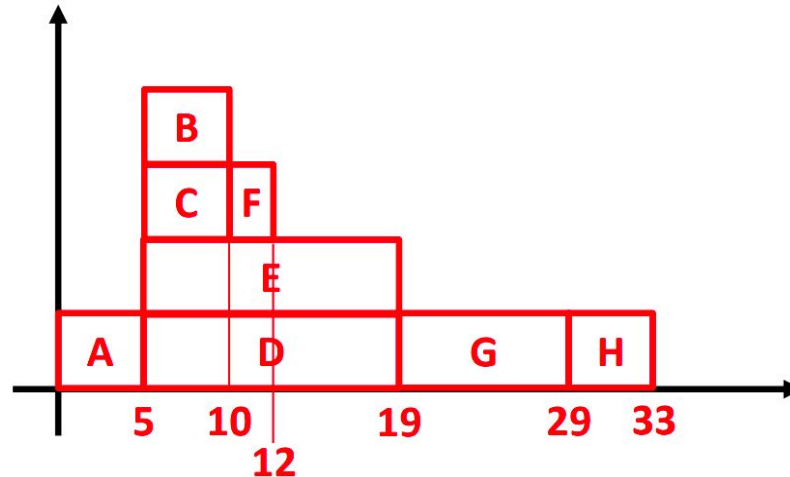**Max Concurrency?** 4

# Q1 (b): Amount of Concurrency



**Critical path length?**

**Average Concurrency?**

**Max Concurrency?**

# Q1 (b): Amount of Concurrency
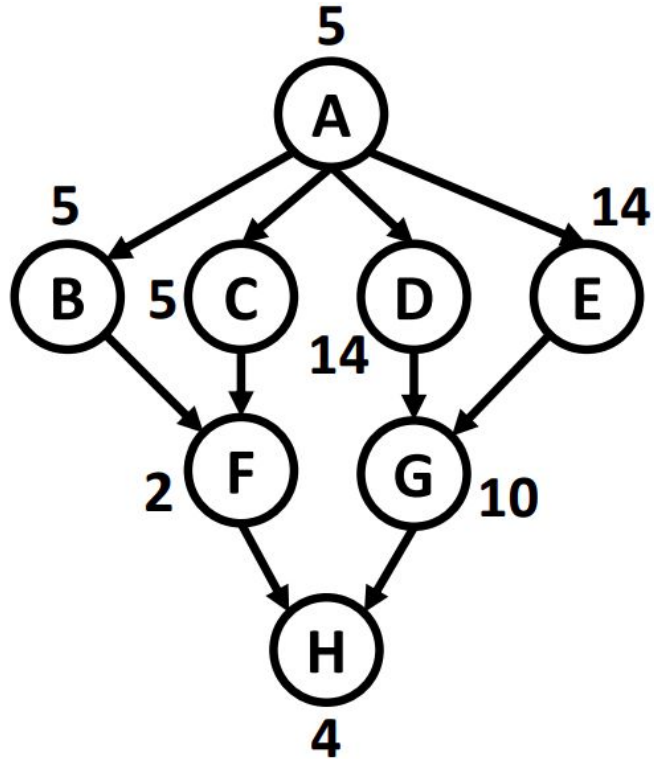


**Critical path length?** 33

**Average Concurrency?**
→ Sum of work / 33 = 59/33 = 1.78

**Max Concurrency?** 4
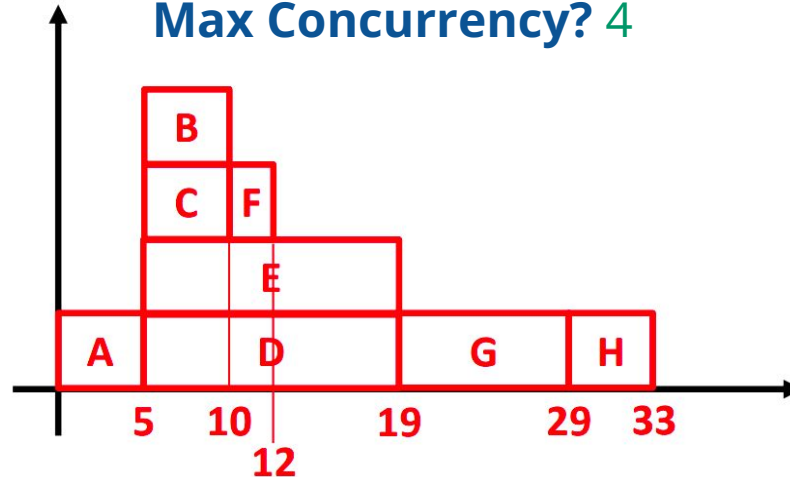
# Q1 (c): Speedup
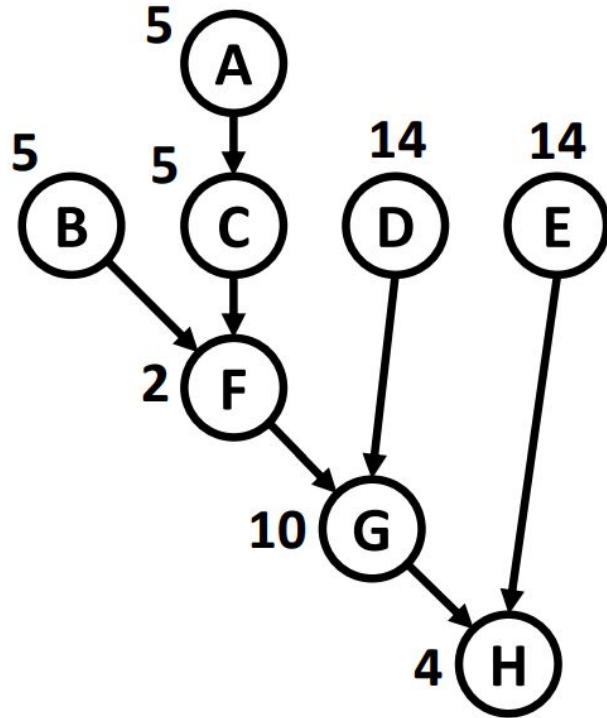


What is the speedup of this program given **infinite resources**?

# Q1 (c): Speedup
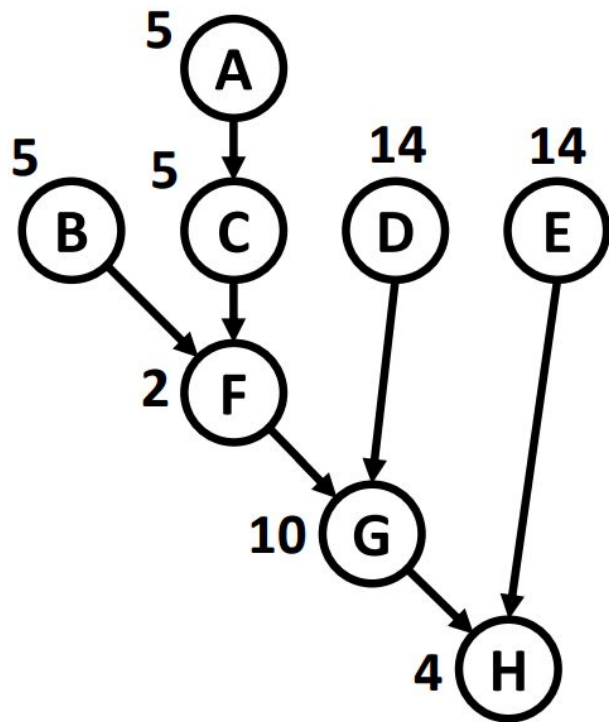


What is the speedup of this program given **infinite resources**?

- Sequental implementation = 59 units
- Parallel: finishes in **28 time units**
  - **Length of the critical path!!**
- Speedup = 59/28 = **2.10**
  - (== avg deg of concurrency)
- Fragment 2's speedup is **1.78** == its avg deg of concurrency

# Q1 (d): Speedup
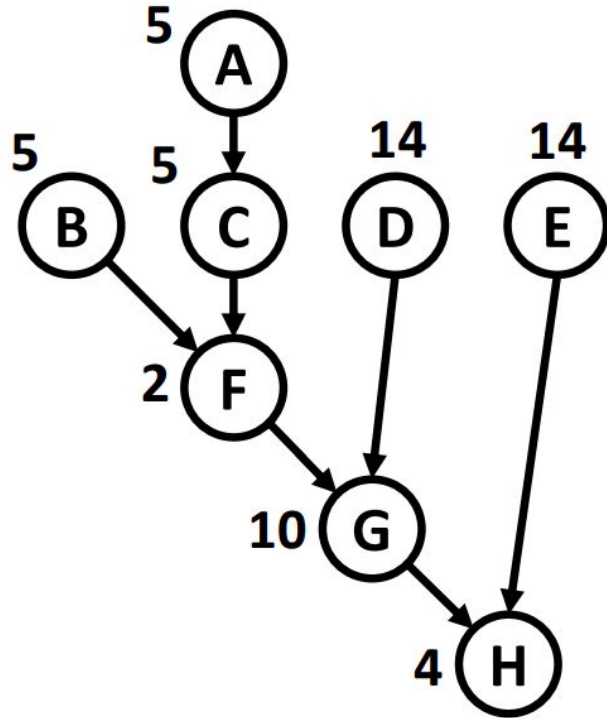


What if we have only 2 processing units? What is the speedup?

# Q1 (d): Speedup
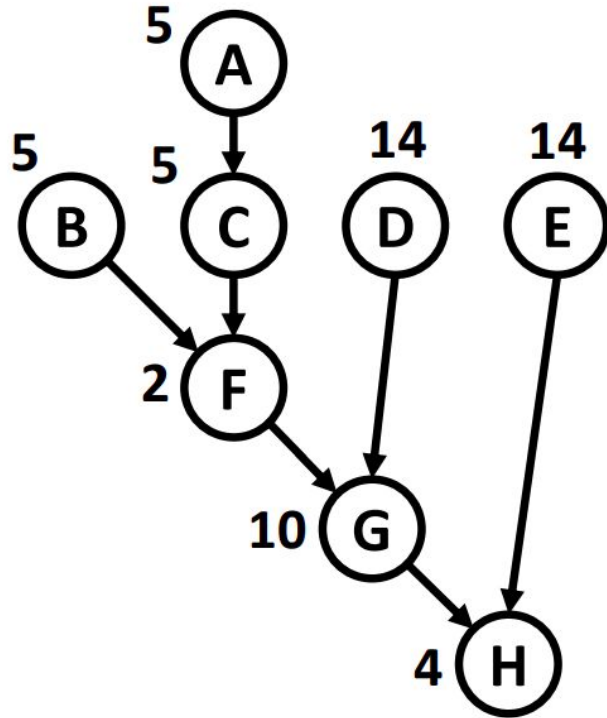


What if we have only 2 processing units? What is the speedup?

Have to draw it out, no easy solution

**Maximum speedup** $= \dfrac{59}{32} < \dfrac{59}{28}$

# Q1 (d): Speedup



What if we have only 2 processing units? What is the speedup?

Have to draw it out, no easy solution

Maximum speedup = $\frac{59}{35} < \frac{59}{33}$
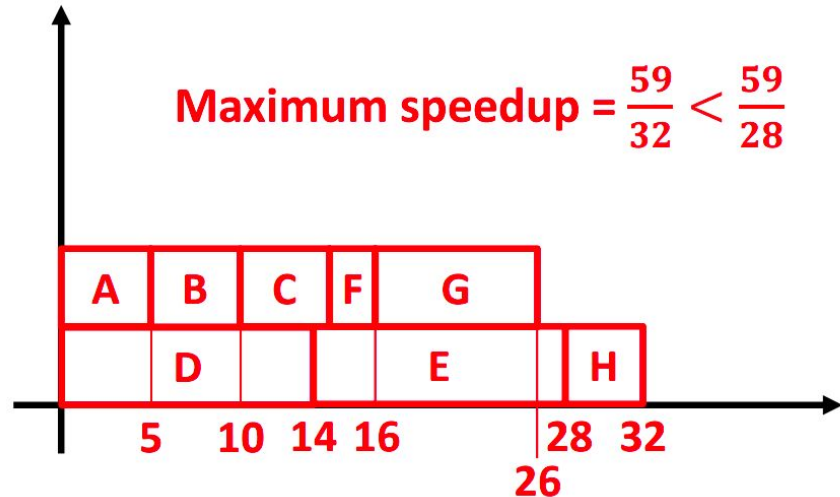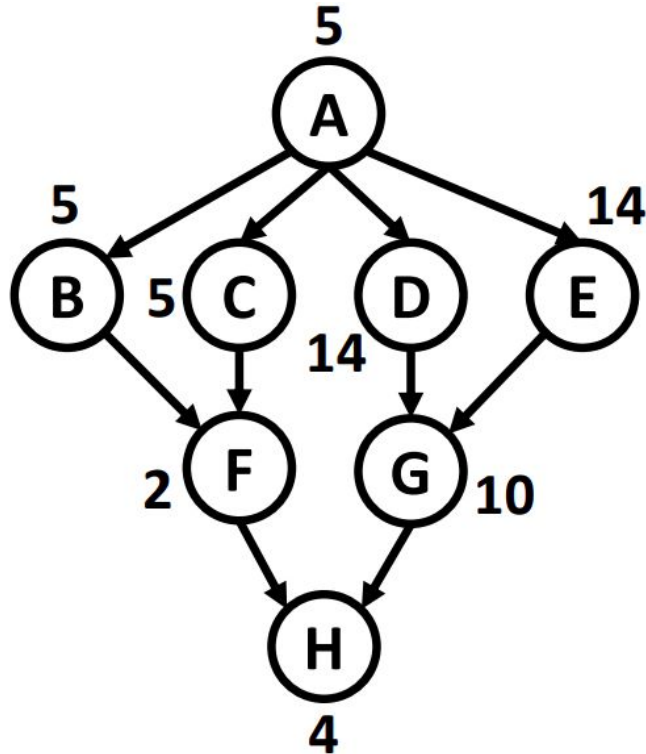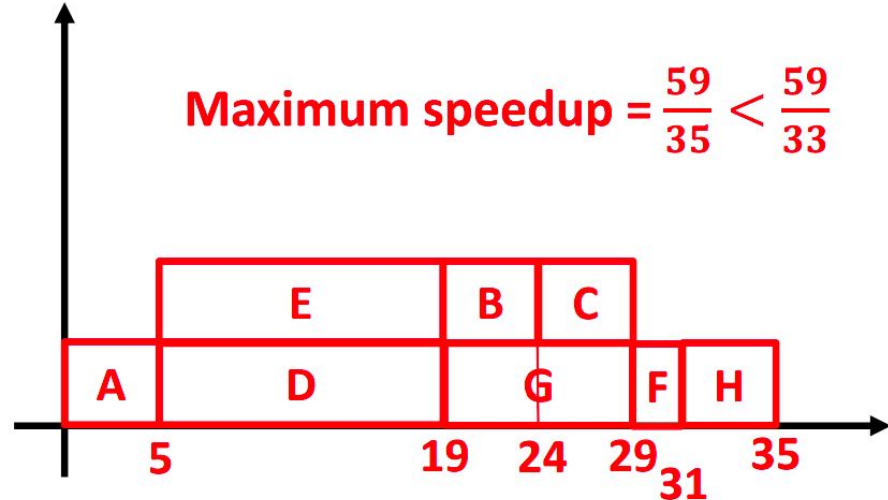
# Why does this question matter?

- Helps you to understand **how to design your parallel programs!**
  - Your program is only as strong as its weakest link!

- Parallel tasks should be as **independent as possible**

- They should have the **shortest critical path possible** (dependency chain should be minimized)
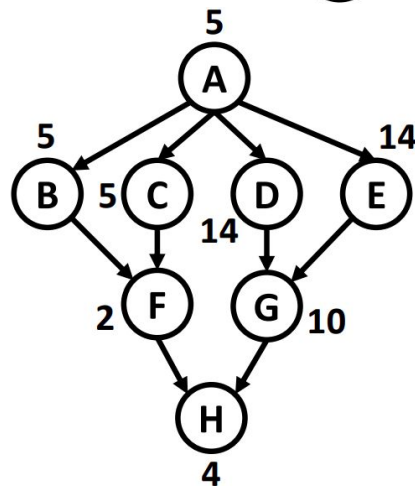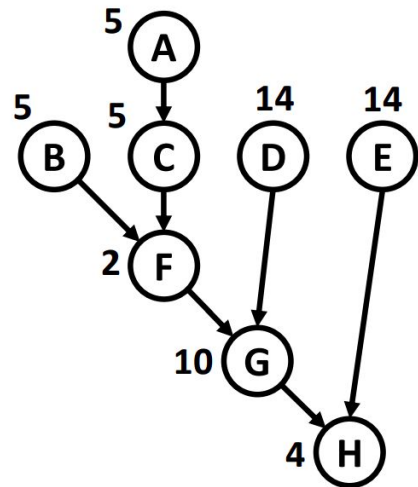
# Question 2: User CPU Time

# Question 2: Cycles Per Instruction

- Cycles per instruction (CPI) for each instr: **$I_1$ = 1, $I_2$ = 2, $I_3$ = 3**
- What is the **average** CPI for the translation of both programs?

| Translation | $I_1$ | $I_2$ | $I_3$ | Total instructions | $n_{cycles}$ |
|---|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 5 | 10 |
| 2 | 4 | 1 | 1 | 6 | 9 |

# Question 2: Cycles Per Instruction

- Cycles per instruction (CPI) for each instr: $I_1 = 1, I_2 = 2, I_3 = 3$
- What is the **average** CPI for the translation of both programs?

- Relatively trivial: no. of cycles / no. of instructions
  - 1st program: 10 / 5 = 2.0 (10 = 2*1 + 1*2 + 2*3)
  - 2nd program: 9 / 6 = 1.5

| Translation | $I_1$ | $I_2$ | $I_3$ | Total instructions | $n_{cycles}$ |
|---|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 5 | 10 |
| 2 | 4 | 1 | 1 | 6 | 9 |

# Why does this question matter?

- Not all instructions are created equal!
  - *i.e. no. of instructions is not a good comparative metric for time taken*

- Let's try...
  - `hx intadd.c`
  - `hx floatadd.c`
  - `perf stat -e cycles,instructions -- ./intadd`
  - `perf stat -e cycles,instructions -- ./floatadd`

- https://godbolt.org/z/3MobeMxbP

# Question 3: MIPS

# Question 3: MIPS

- Cycles per instruction (CPI) for each instr: $I_1 = 1, I_2 = 2, I_3 = 3$
- **2 GHz (2 x $10^9$) clock rate**
- **Different composition of instructions in each program**
- What is the relationship between time taken (to complete the program) and MIPS in this qn?

| Program | $I_1$ | $I_2$ | $I_3$ |
|---------|-------|-------|-------|
| $A_1$ | $5 \times 10^9$ | $1 \times 10^9$ | $1 \times 10^9$ |
| $A_2$ | $10 \times 10^9$ | $1 \times 10^9$ | $1 \times 10^9$ |

# Question 3: MIPS

- Cycles per instruction (CPI) for each instr: $I_1 = 1, I_2 = 2, I_3 = 3$
- **2 GHz ($2 \times 10^9$) clock rate**
- How long does each program take?
  - Time for $A_1$ = (Cycles / Clock speed) = $(5 + (1 \times 2) + (1 \times 3)) \times 10^9 / (2 \times 10^9)$     = 5s
  - Time for $A_2$ = (Cycles / Clock speed) = $(10 + (1 \times 2) + (1 \times 3)) \times 10^9 / (2 \times 10^9)$     = 7.5s
- What is the MIPS for each program?
  - MIPS for $A_1$ = (MillionInstrs / Time)     = $((5 + 1 + 1) \times 10^9 / 106) / 5$          = 1400 MIPS
  - MIPS for $A_2$ = (MillionInstrs / Time)     = $((10 + 1 + 1) \times 10^9 / 106) / 7.5$          = 1600 MIPS
- What can we conclude?

| Program | $I_1$ | $I_2$ | $I_3$ |
|---------|-------|-------|-------|
| $A_1$ | $5 \times 10^9$ | $1 \times 10^9$ | $1 \times 10^9$ |
| $A_2$ | $10 \times 10^9$ | $1 \times 10^9$ | $1 \times 10^9$ |

"MIPS is like a real estate agent determining **how much a building is worth** by measuring the **weight of a rubber chicken.**"

(Brendan, StackOverflow)

# Why does this question matter?

- **Don't be fooled by marketing!**

- You are now educated in parallel computing performance :)

# Question 4: Speedup

# Question 4: Speedup

- For these questions: we *could* use the laws directly
  (feel free in the exam, faster)

- But in this tutorial: let's **understand what we're doing from first principles!**

| **Sequential:** $N$ instructions | **Parallel:** $N^2$ instructions |
|---|---|

Each instruction takes 2 cycles on a 1 GHz ($10^9$ Hz) processor

# Question 4: Speedup

- For N = 100, what is the speedup for 10 and 100 processors?

- **Total number of cycles if no parallelism:** $(N + N^2)$ x 2 cycles/instr.

$$= (100 + 100^2) \text{ x } 2$$

| **Sequential:** $N$ instructions | **Parallel:** $N^2$ instructions |
|---|---|

Each instruction takes 2 cycles on a 1 GHz ($10^9$ Hz) processor

# Question 4: Speedup

- For N = 100, what is the speedup for 10 and 100 processors?
- **Strategy (using basic logic)**: find sequential time, find parallel times
- **Sequential time:** $((100 + 100^2) \times 2) / 10^9$ = 20200 ns
- **Par time (p=10):** $((100 + 100^2/10) \times 2) / 10^9$ = 2200 ns (9.18x speedup)
- **Par time (p=100):** $((100 + 100^2/100) \times 2) / 10^9$ = 400 ns (50.6x speedup)
- If we have infinite resources, the parallel section can be approximated to 0 instructions running serially
- Par time (p=∞): $((100 + 0) \times 2) / 10^9$ = 200 ns (101x speedup)

| **Sequential:** $N$ instructions | **Parallel:** $N^2$ instructions |
|---|---|

Each instruction takes 2 cycles on a 1 GHz ($10^9$ Hz) processor

# Question 4: Speedup

Note: This is identical from direct application of Amhdal's law, but the basics are simpler to understand for now!

$$f = \frac{I_{seq}}{I_{total}} = \frac{N}{N + N^2} = \frac{1}{1 + N} = \frac{1}{101}$$

$$S_{10}(100) = \frac{1}{\frac{1}{101} + \frac{100}{101(10)}} = 9.181$$

$$S_{100}(100) = \frac{1}{\frac{1}{101} + \frac{100}{101(100)}} = 50.5$$

$$S_{\infty}(100) = \frac{1}{\frac{1}{101} + 0} = \frac{1}{f} = 101$$

**Recap:**
f = ratio of sequential execution time to total execution time
$S_{10}$ = Speedup achievable with 10 processors

| **Sequential:** $N$ instructions | **Parallel:** $N^2$ instructions |
|---|---|

Each instruction takes 2 cycles on a 1 GHz ($10^9$ Hz) processor

# Question 4: Speedup

- Fixed time: N that can be solved with 10/100 processors?
- Sequential time (p=1):  $((100 + 100^2) \times 2) / 109$  = 20200 ns
- For **10** processors, find N:
  - $((N + N^2/10) \times 2) / 10^9$ = 20200 ns  $\Rightarrow$ N = 312
  - "Gustafson speedup": (Sequential time for N = 312 / Parallel time for N = 312)
- For **100** processors, find N:
  - $((N + N^2/100) \times 2) / 10^9$ = 20200 ns  $\Rightarrow$ N = 956
- **You don't need to "stick to the law" to solve these problems!**

| Sequential: $N$ instructions | Parallel: $N^2$ instructions |
| --- | --- |

Each instruction takes 2 cycles on a 1 GHz ($10^9$ Hz) processor

# Why does this question matter?

- If you have a fixed-sized problem to solve OR you have a constant sequential fraction with increasing problem size
  - **Amdahl's law applies**: speedup limited by the sequential fraction

- If you have a problem size that can be varied AND your sequential fraction does not scale as much with the problem
  - **Gustafson's law applies**: you can solve larger problems with more speedup!

- **Amdahl's Law — Strong Scaling**
  - Fixed Problem Size
  - How much does parallelism reduce the execution time of a problem?

- **Gustafson's Law — Weak Scaling**
  - Fixed Execution Time
  - How much longer does it take for the problem without parallelism?

# Question 5: Parallel Programming Models

# Question 5: Parallel Programming Models
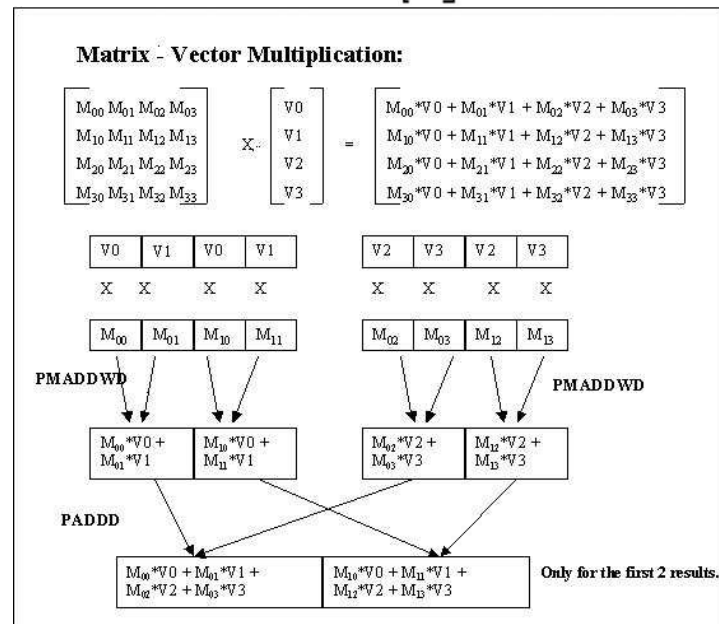
Data or task parallelism?

$$A \times x = y$$

$$
\begin{bmatrix} \phantom{xxxxx} \end{bmatrix} \times \begin{bmatrix} \phantom{x} \end{bmatrix} = \begin{bmatrix} \phantom{x} \end{bmatrix}
$$

m x n matrix
(m rows,
n columns)

n x 1 matrix
(n-dimensional
vector)

m-dimensional
vector

# Question 5: Parallel Programming Models

- Data Parallelism

- We want to use SIMD (Flynn's Taxonomy) for this!

  - Yes, MIMD is superset of SIMD
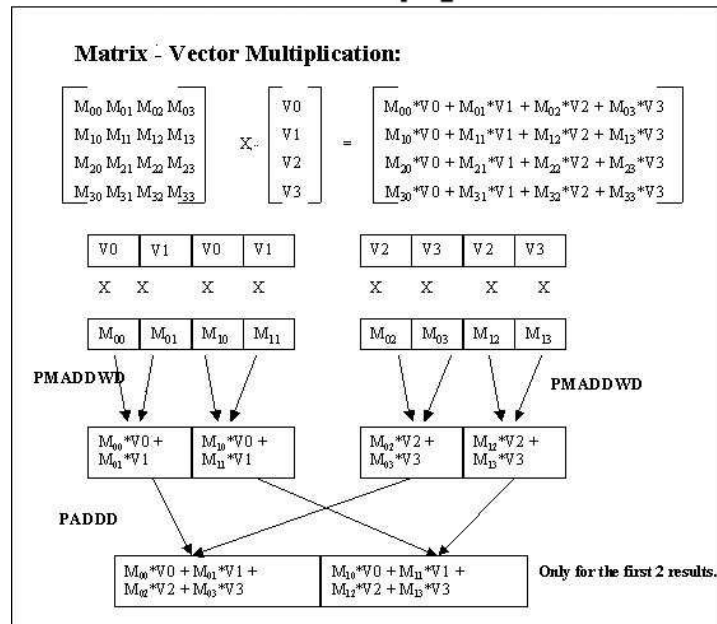  - Teaching point here is the idea of SIMD instructions

# Question 5: Parallel Programming Models

What kind of parallel programming pattern (disregarding SIMD) would be best for this?

- **Fork-join**
- **Parbegin-parend**
- **Master-worker**
- **Task pool**
- **Pipelining**
- **Producer-consumer**

$$A \times x = y$$

$$\begin{bmatrix} \phantom{xxx} \end{bmatrix} \times \begin{bmatrix} \phantom{x} \end{bmatrix} = \begin{bmatrix} \phantom{} \end{bmatrix}$$

**Matrix - Vector Multiplication:**

$$\begin{bmatrix} M_{00} & M_{01} & M_{02} & M_{03} \\ M_{10} & M_{11} & M_{12} & M_{13} \\ M_{20} & M_{21} & M_{22} & M_{23} \\ M_{30} & M_{31} & M_{32} & M_{33} \end{bmatrix} X \begin{bmatrix} V0 \\ V1 \\ V2 \\ V3 \end{bmatrix} = \begin{bmatrix} M_{00}*V0 + M_{01}*V1 + M_{02}*V2 + M_{03}*V3 \\ M_{10}*V0 + M_{11}*V1 + M_{12}*V2 + M_{13}*V3 \\ M_{20}*V0 + M_{21}*V1 + M_{22}*V2 + M_{23}*V3 \\ M_{30}*V0 + M_{31}*V1 + M_{32}*V2 + M_{33}*V3 \end{bmatrix}$$

| V0 | V1 | V0 | V1 |
|----|----|----|----|
| X | X | X | X |
| M_{00} | M_{01} | M_{10} | M_{11} |

PMADDWD

| V2 | V3 | V2 | V3 |
|----|----|----|----|
| X | X | X | X |
| M_{02} | M_{03} | M_{12} | M_{13} |

PMADDWD

| $M_{00}*V0 + M_{01}*V1$ | $M_{10}*V0 + M_{11}*V1$ |
|---|---|

| $M_{02}*V2 + M_{03}*V3$ | $M_{12}*V2 + M_{13}*V3$ |
|---|---|

PADDD

| $M_{00}*V0 + M_{01}*V1 + M_{02}*V2 + M_{03}*V3$ | $M_{10}*V0 + M_{11}*V1 + M_{12}*V2 + M_{13}*V3$ | Only for the first 2 results. |
|---|---|---|

# Question 5: Parallel Programming Models

- What kind of parallel programming pattern would be best for this?
- **Fork-join / Parbegin-parend**
  - We don't need the complexity of fork-join - leads to very spaghetti code
  - Parbegin/end is arguable since that is literally OpenMP
- **Master-worker**
  - Best here: we want relatively simple and homogeneous (same amount of work) worker threads, and a master thread to organize them
- **Task pool**
  - Not as good as master-worker here: tasks are usually for heterogeneous tasks or those that finish at very different times
- **Pipelining:** better for task parallelism, this is data parallel, single task type
- **Producer-consumer:** does not fit into this model as nothing is "produced"

# Why does this question matter?

- Thinking of the right pattern allows you to use the **right tools**

- **Non-data-parallel-like problem:** Don't use GPGPU computation

- **Parbegin/end:** good for OpenMP

- Many more!

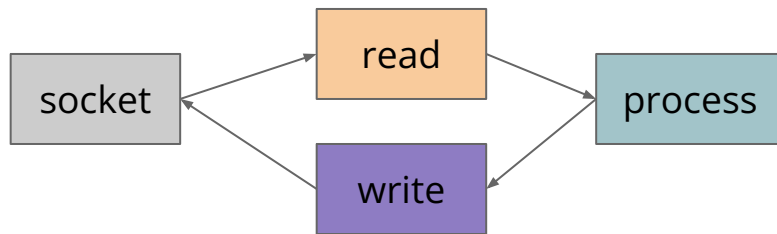# Question 6: 2020 Midterms Q1

# Question 6: 2020 Midterms Q1

- You are parallelizing a backend web server
- It runs on a <u>shared memory</u> machine

- **Discuss about**
  - **task vs data parallelism**
  - **what pattern to use**

- **Request for each client run in this order**
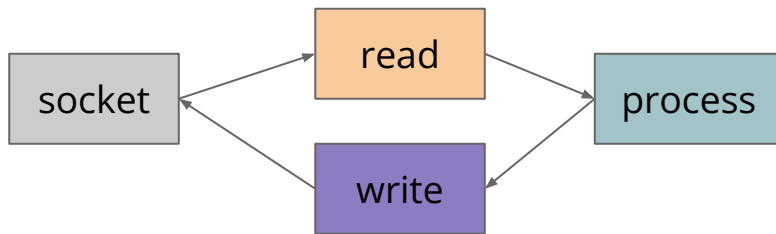
# Question 6: 2020 Midterms Q1
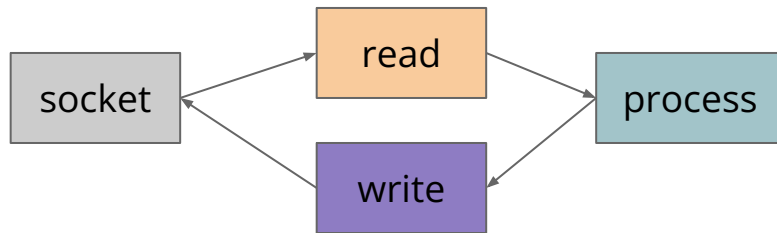
What kind of parallelism?
**[Task/Data parallelism]**



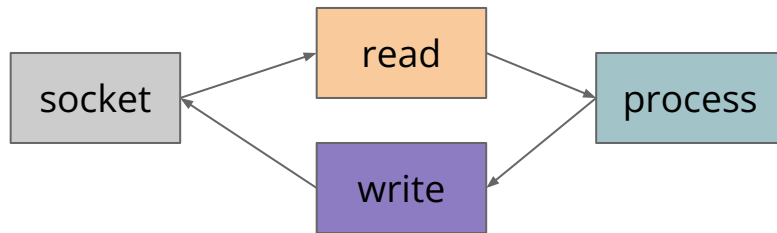**Assume that read/write/process time is *all very similar***

# Question 6: 2020 Midterms Q1

What kind of parallelism?

- **Task parallelism: each stage is a task**
- Data parallelism may be accepted if request = task AND requests assumed to arrive at same time
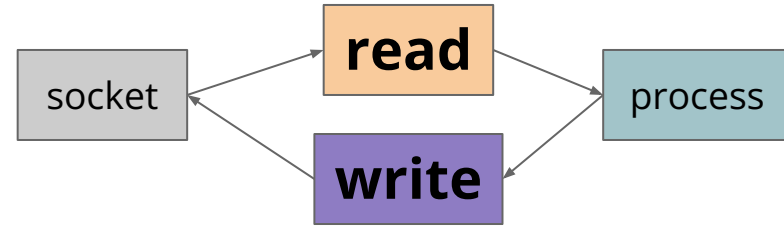
What pattern to use?



**Assume that read/write/process time is *all very similar***

# Question 6: 2020 Midterms Q1

What kind of parallelism?

- **Task parallelism: each stage is a task**
- Data parallelism may be accepted if request = task AND requests assumed to arrive at same time

What pattern to use?

- **Best:** pipelining - best in cases where **each stage** takes **similar time**



**Assume that read/write/process time is *all very similar***

# Question 6: 2020 Midterms Q1

What kind of parallelism?



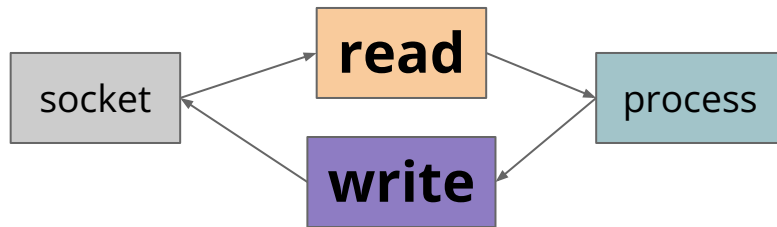**Assume that read/write >> compute time**

# Question 6: 2020 Midterms Q1

What kind of parallelism?

- **Task parallelism: each stage is a task**
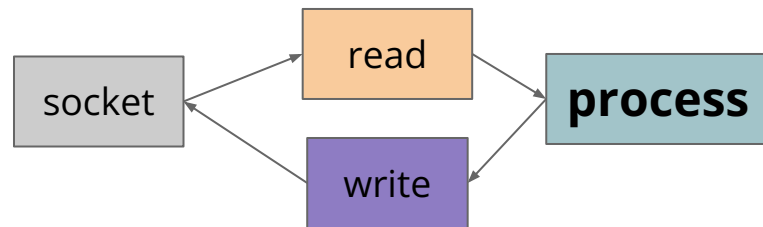
What pattern to use?

- **Producer-consumer:** producer **reads** from socket and **processes**, consumer **writes back** to the socket, **#producers == #consumers**
- **No pipelining: the stages are uneven!**

```
socket  →  read  →  process
socket  ←  write  ←  process
```

**Assume that read/write >> compute time**

# Question 6: 2020 Midterms Q1

Similar to previous part: except for producer/consumer, we need **more producers than consumers**



socket → read → **process**
socket → write → **process**

**Assume that compute time >> read/write**

# Question 6: 2020 Midterms Q1

- **Assume task parallelism**
- What pattern do we use?

- **Task pool is perfect here:** pool of tasks are ready for jobs and are assigned as necessary
- **If distributed memory:** same answers, but need explicit communication.
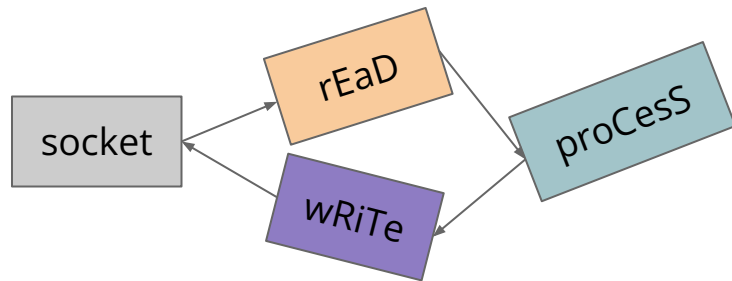


**What if we don't know how long anything takes? Or heavily varies?**

# Question 6: 2020 Midterms Q1

- **Distributed memory architecture?**

- Same answers, but need explicit communication.
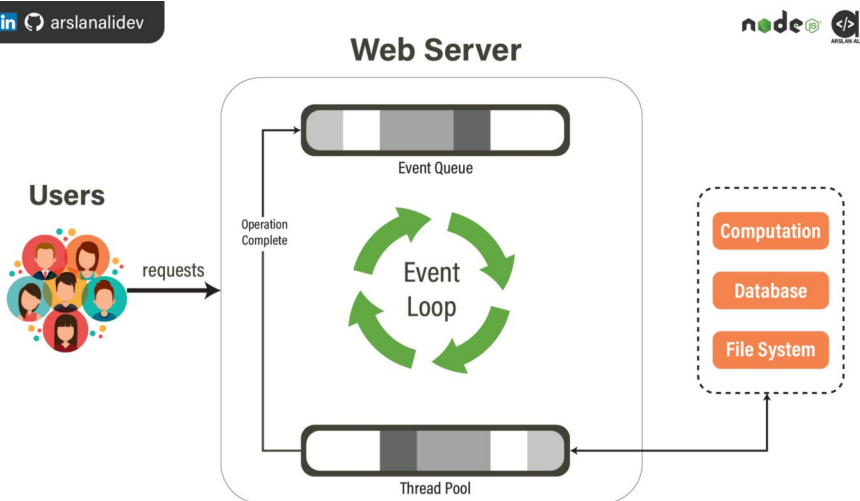- Pipelining not feasible
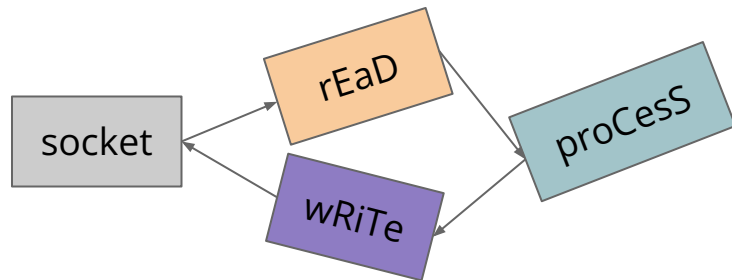


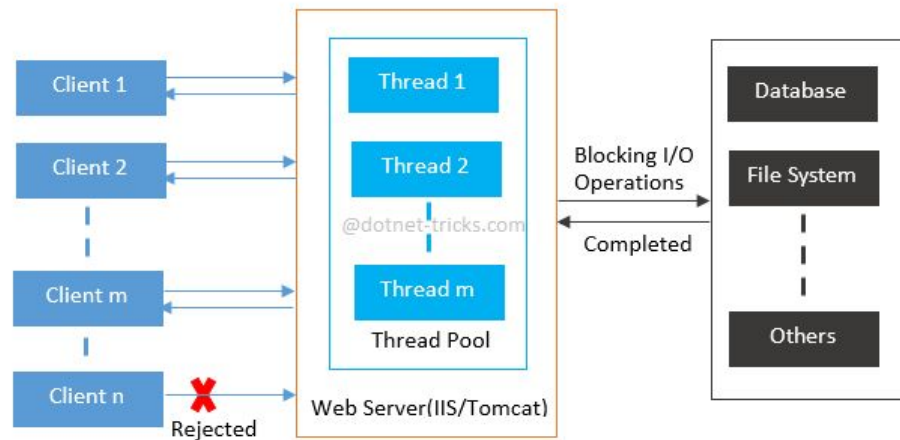**What if we don't know how long anything takes? Or heavily varies?**

# Why does this question matter?

- Affects your entire system architecture and downstream programming patterns!



Node JS Architecture

Multi-Threaded Web Server request processing

# End of Tut 2

**Anonymous Feedback:**

https://bit.ly/feedback-theodore

**Reminder: Submit your Lab 2 by Wednesday 2pm!**

# Attendance