Concurrency

Introduction to Computer Systems, Fall 2024

Instructors: Joel Ramirez Travis McGaha

Head TAs: Adam Gorka Daniel Gearhardt
Ash Fujiyama Emily Shen

TAs:

Ahmed Abdellah
Angie Cao
August Fu
Caroline Begg
Cathy Cao
Claire Lu
Eric Sungwon Lee
Ethan Weisberg
Garrett O'Malley Kirsch
Hassan Rizwan
Iain Li
Jerry Wang
Juan Lopez
Keith Mathe
Maya Huizar
Meghana Vasireddy
Perrie Quek
Sidharth Roy
Sydnie-Shea Cohen
Vivi Li
Yousef AlRabiah

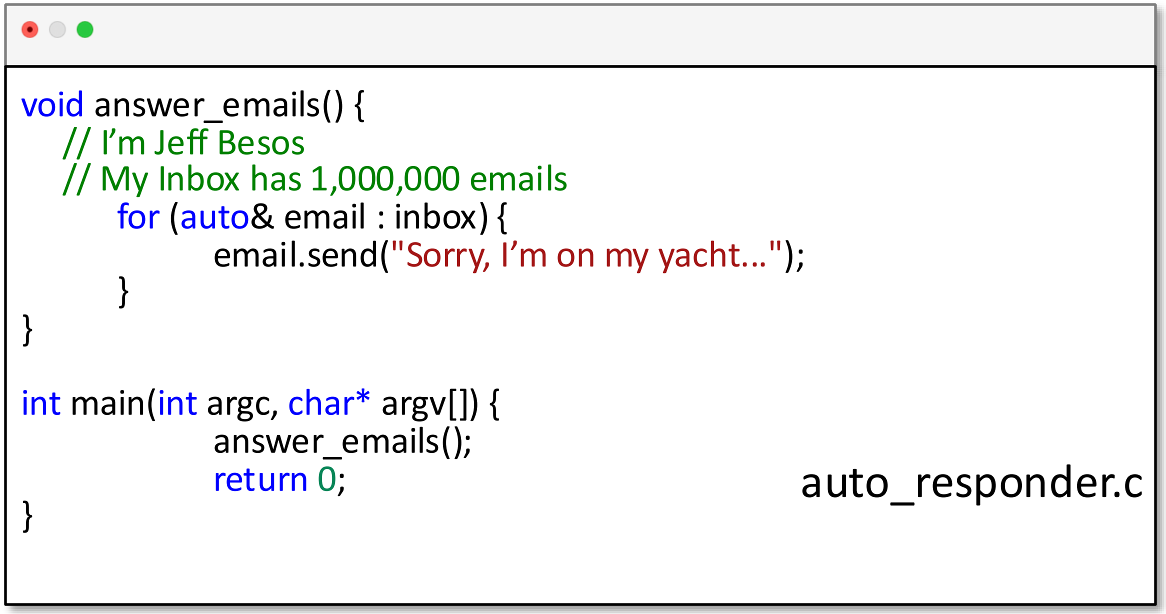
Poll Everywhere

pollev.com/cis2400

- I hope you were able to enjoy your breaks!
- Anything exciting happen? Any really good food made?

What is a Program?

- A "program" is a set of instructions, essentially a ***static*** file containing code.



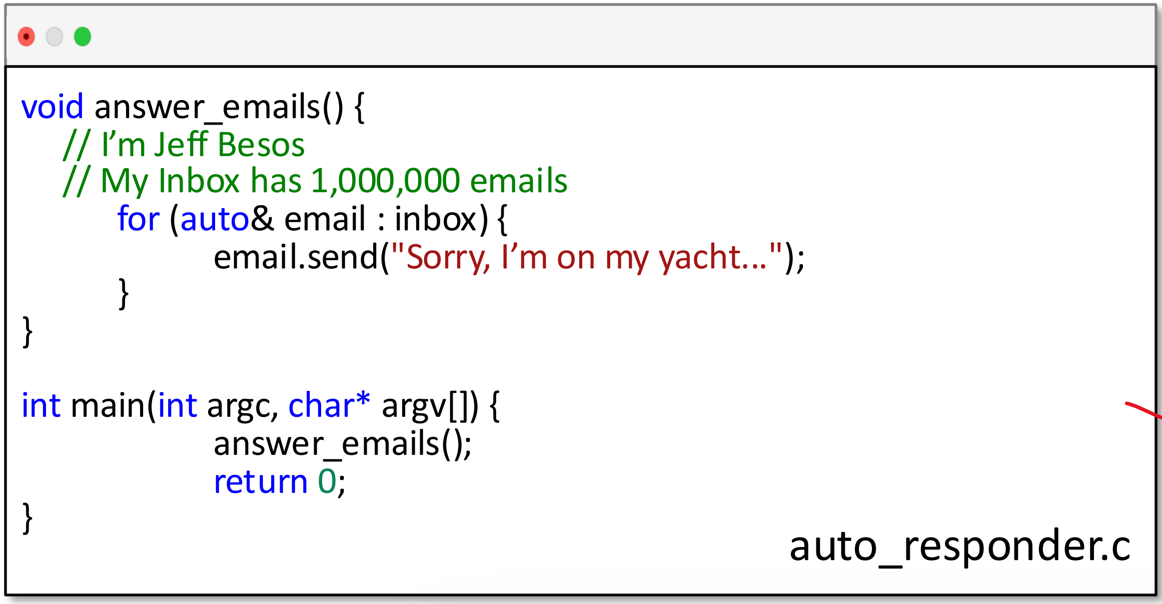
```
void answer_emails() {
    // I’m Jeff Besos
    // My Inbox has 1,000,000 emails
        for (auto& email : inbox) {
                email.send("Sorry, I’m on my yacht...");
        }
}

int main(int argc, char* argv[]) {
                answer_emails();
                return 0;
}
```

auto_responder.c

It’s just text…
nothing special about it.

What is a Program *in execution*?

- A "program" in ***execution*** is called a ***process***.



```
void answer_emails() {
    // I'm Jeff Besos
    // My Inbox has 1,000,000 emails
        for (auto& email : inbox) {
                email.send("Sorry, I'm on my yacht...");
        }
}

int main(int argc, char* argv[]) {
                answer_emails();
                return 0;
}
```

auto_responder.c

The compiled Instructions executed on the processor

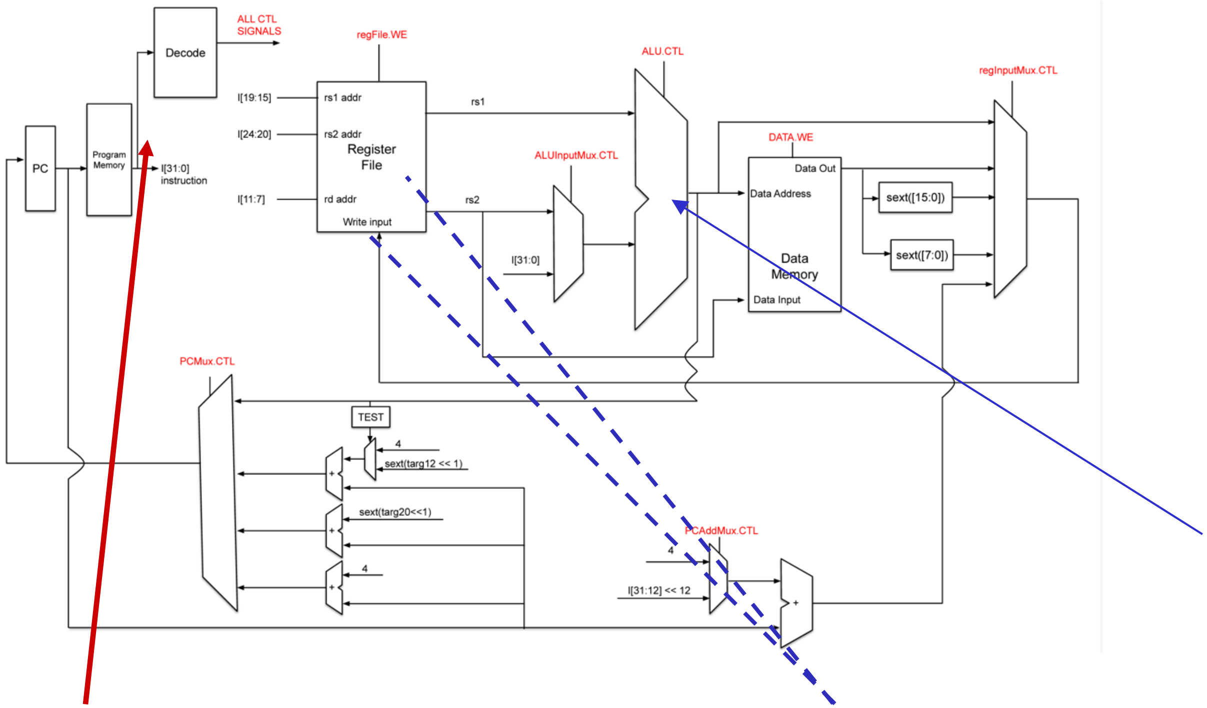
What does the process consist of?

memory, instructions, registers, and other state involved in program execution.

What is *necessary* to run a process?

- You need a CPU with at least one core!
- What's a core?

ITS THIS! (ESSENTIALLY)



Execution Unit (ALU)

Fetch/Decode Instructions

Register State/Memory State

What is *necessary* to run a process?

- You need a CPU with a single core!
- Single Core
  - Fetch/Decode, Register/Memory, Execution Unit (ALU)
  - *Fundamental unit of systems hardware*

Truth: most things aren't singly cored anymore

- Does anyone know the # of cores in an Intel i9 CPU?
  - (the ceo just quit yesterday btw. Company isn't doing well I hear)
  - We'll leave that to the Wharton people

*memory in these slides does not refer to RAM but rather the Cache

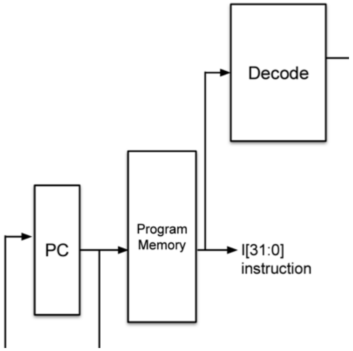
What is *necessary* to run a process?

- You need a CPU with a single core!
- Single Core
  - Fetch/Decode, Register/Memory, Execution Unit (ALU)
  - *Fundamental unit of systems hardware*

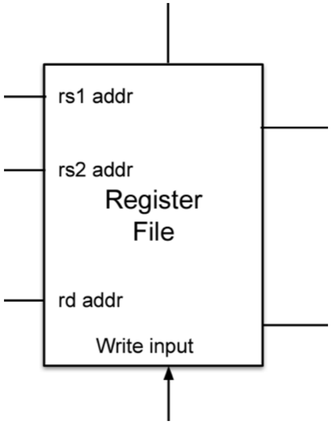
Truth: most things aren't singly cored anymore

- Does anyone know the # of cores in an Intel i9 CPU?
  - Trick question; ***depends on the model***.
  - Intel® Core™ i9-7900X X-series Processor
    - ***10 Cores!***
  - ***Highest model has 18 cores!***

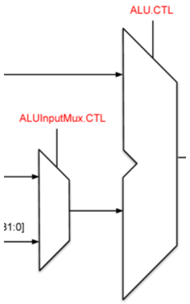
A Single Core



Fetch and decode instructions.



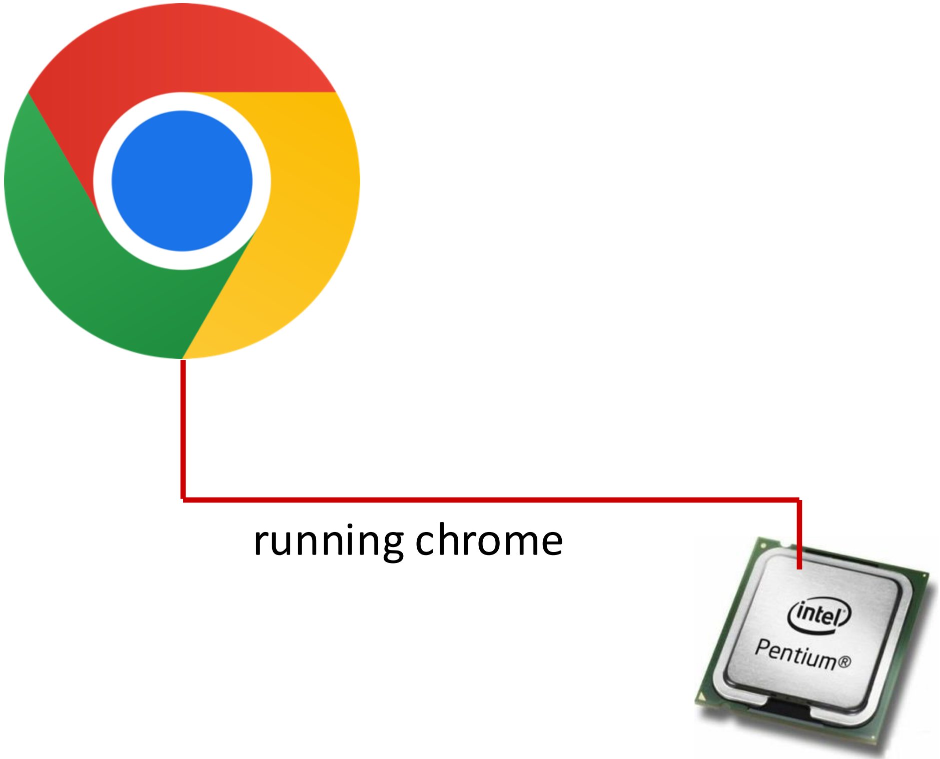
Register Set and Memory State



Execution Unit (ALU)

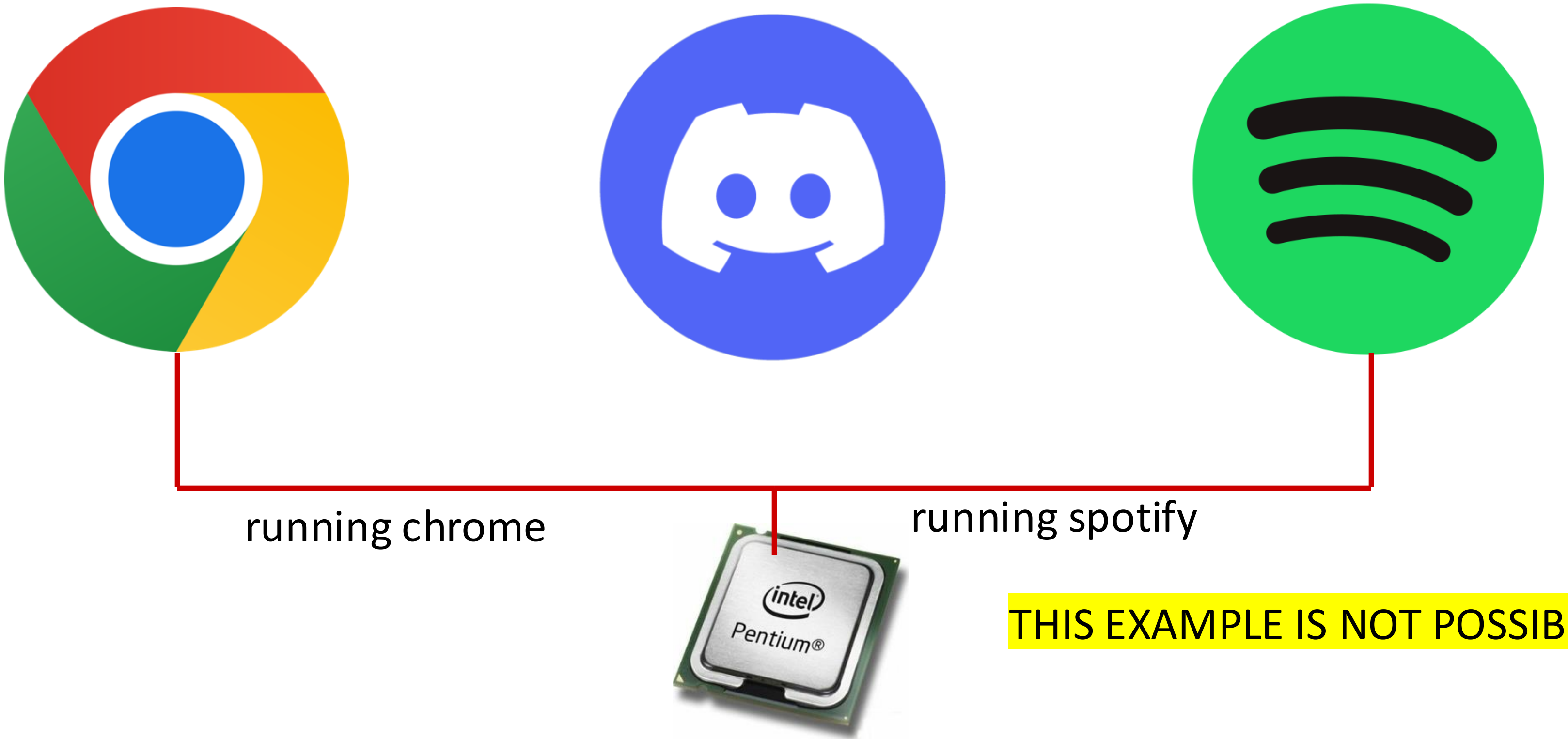
Let’s focus on just one core for now

- With one core, we can run one process!
  - **let’s open up chrome**
- **Question: what if you want to open 2 more applications?**



Let’s focus on just one core for now

- We want to run 3 things on a single core processor.
- **Things can not run *via parallelism*** *(simultaneously).*
  - *Why? We only have ONE CORE.* ***Only one ALU to go around.***



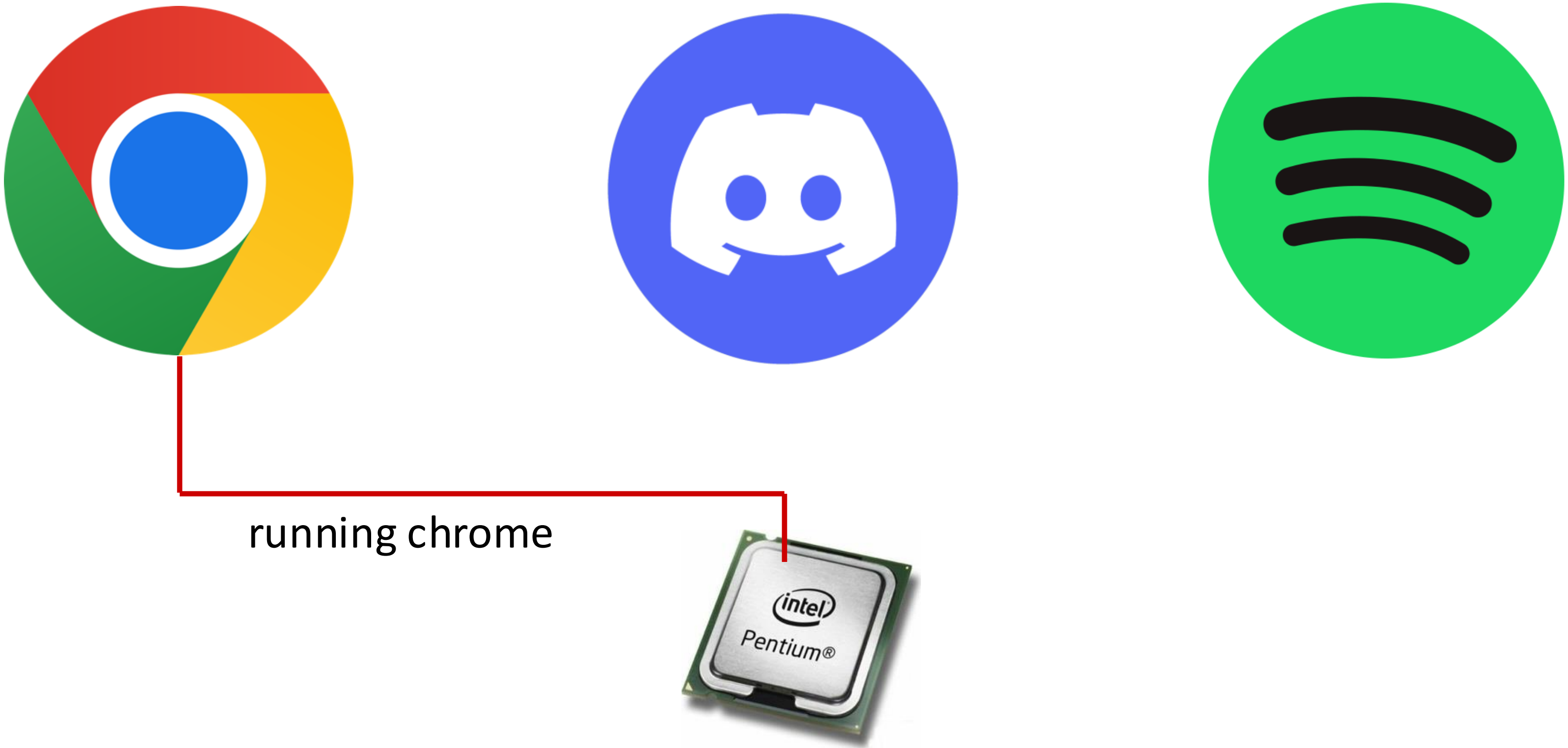
running chrome

running spotify

THIS EXAMPLE IS NOT POSSIBLE

Let’s focus on just one core for now

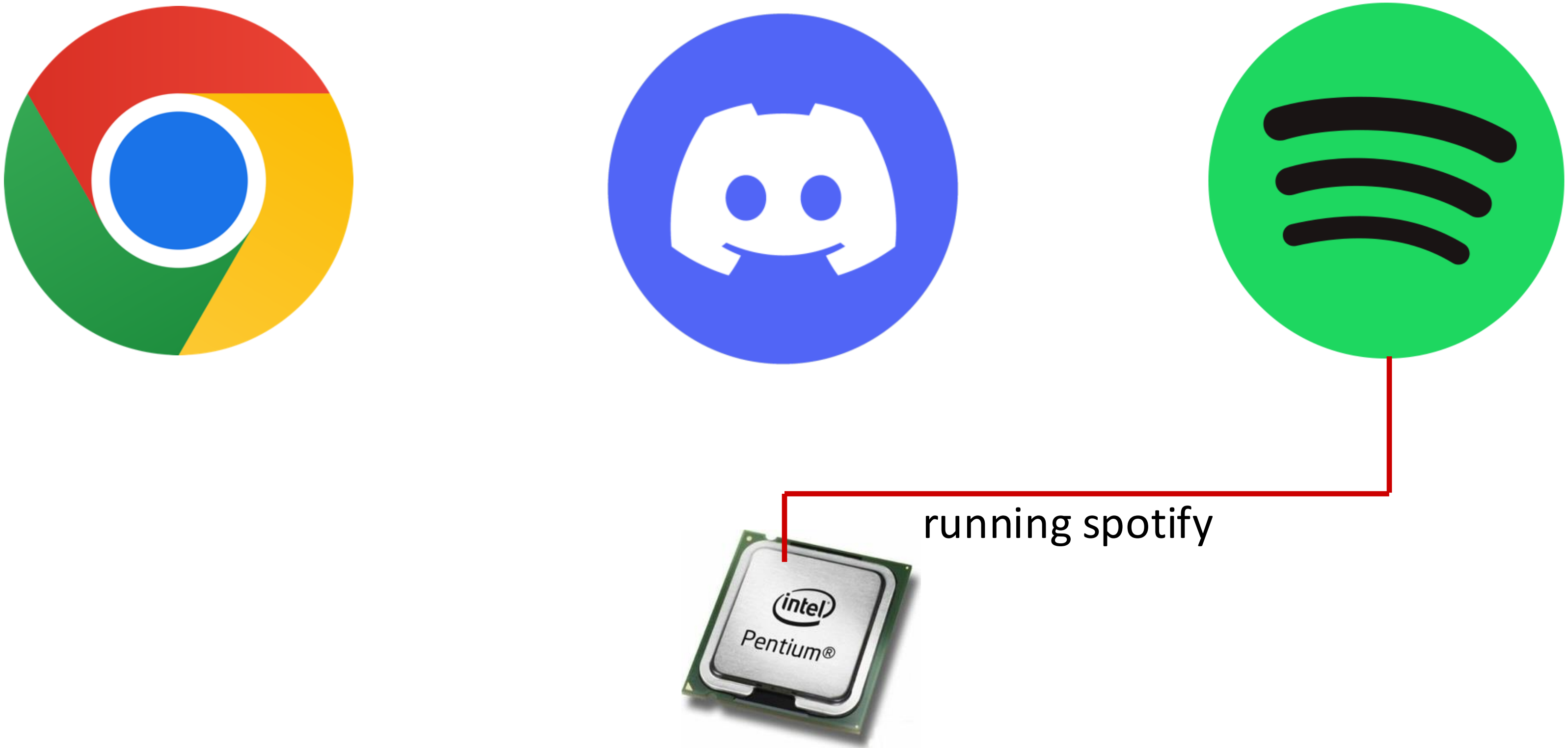
- We want to run 3 things on a single core processor.
- **Our Solution: Concurrency**
  - **we can switch between different processes**



running chrome

Let’s focus on just one core for now

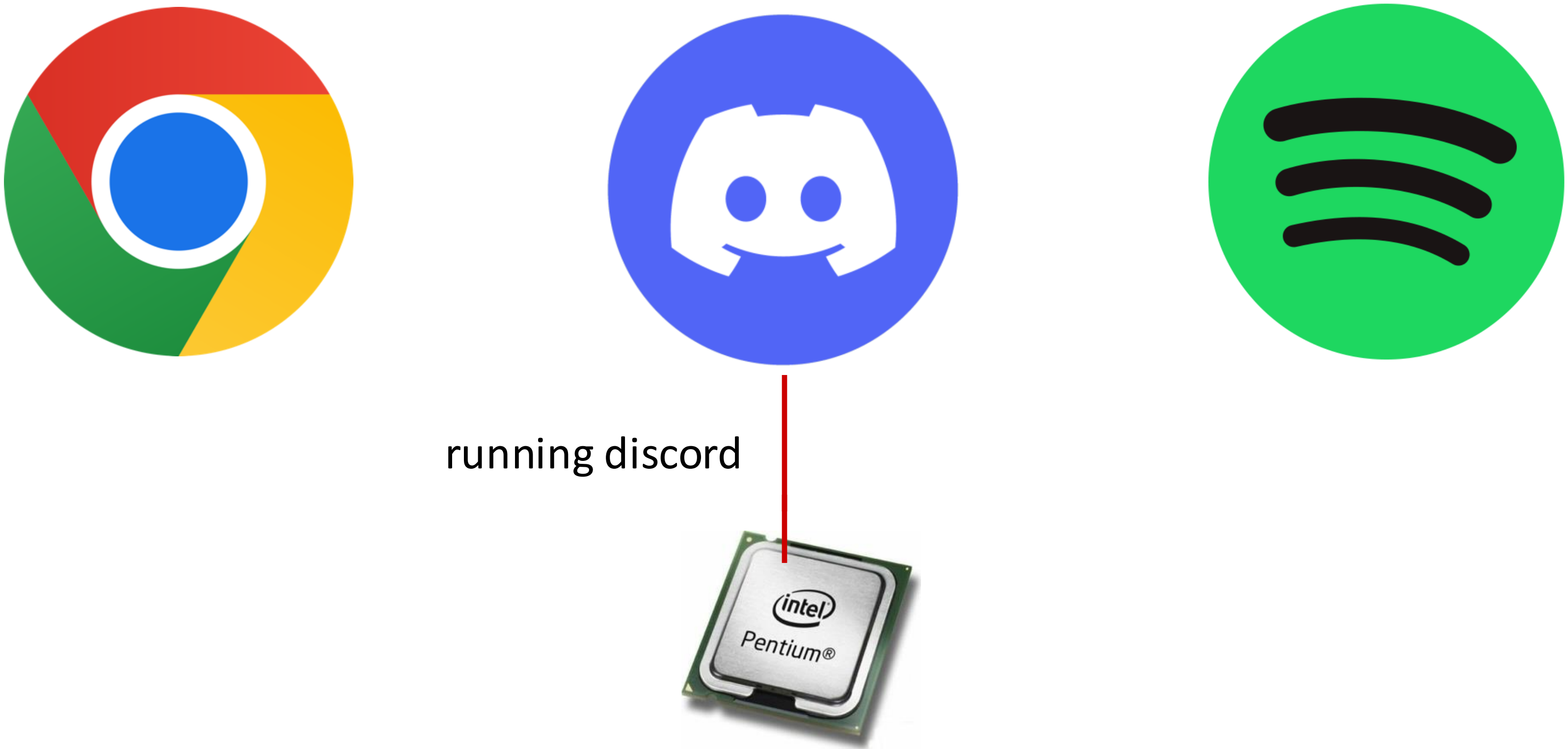
- We want to run 3 things on a single core processor.
- **Our Solution: Concurrency**
  - **we can switch between different processes**



running spotify

Let’s focus on just one core for now

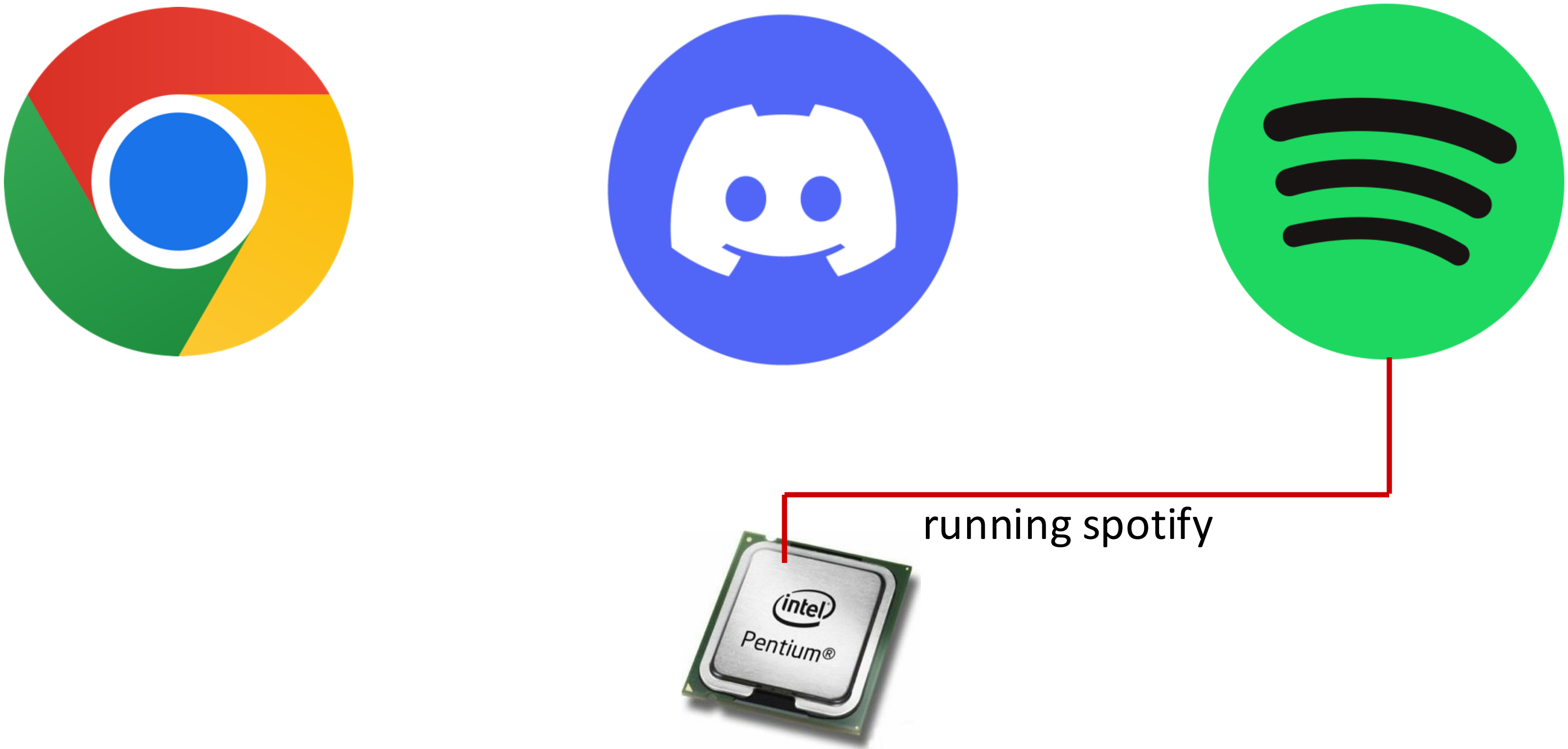
- We want to run 3 things on a single core processor.
- **Our Solution: Concurrency**
  - **we can switch between different processes**



running discord

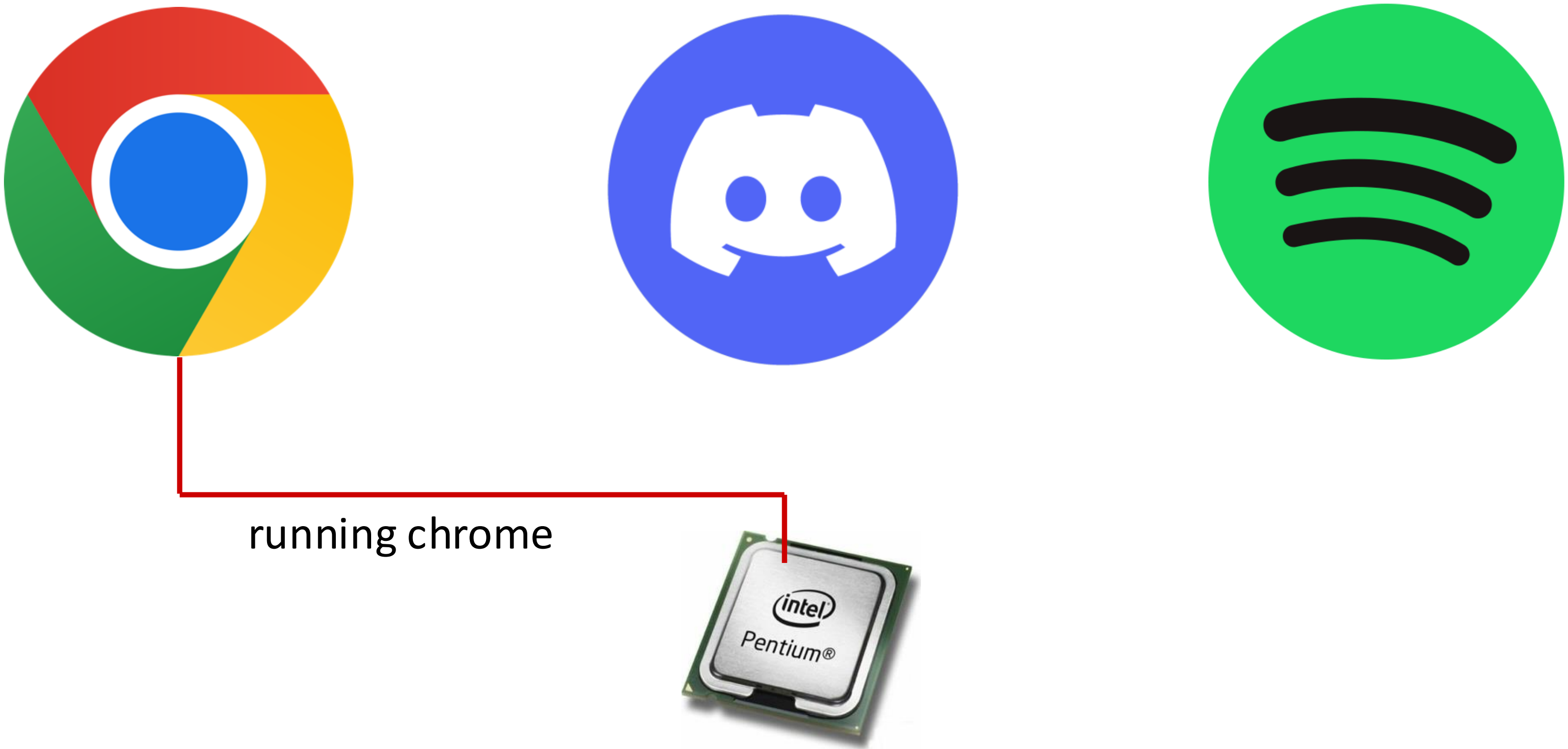
Let’s focus on just one core for now

- We want to run 3 things on a single core processor.
- **Our Solution: Concurrency**
  - **Make it seem like things are running *simultaneously***



Let’s focus on just one core for now

- We want to run 3 things on a single core processor.
- **Our Solution: Concurrency**
  - **Make it seem like things are running *simultaneously***

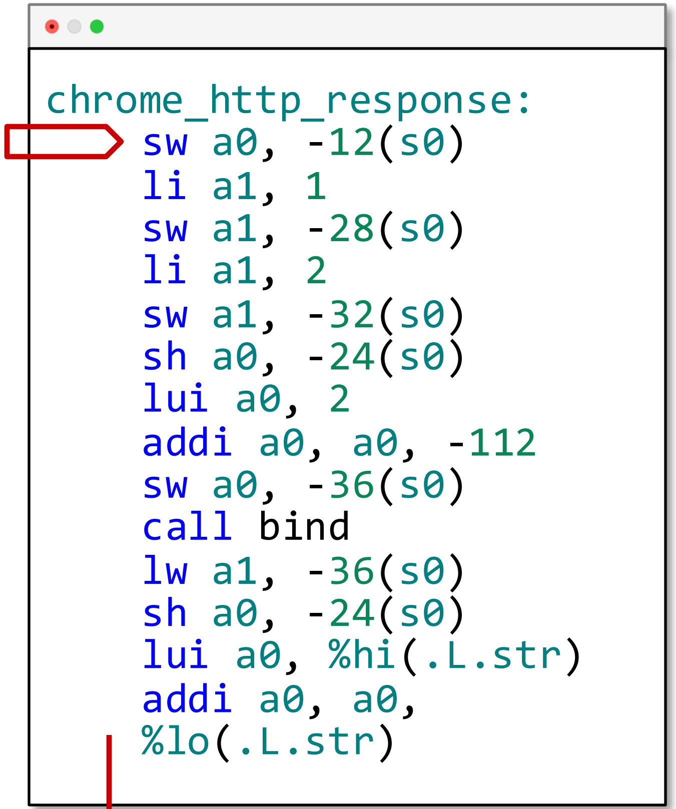


running chrome

What does concurrency look like?

Chrome
Instructions

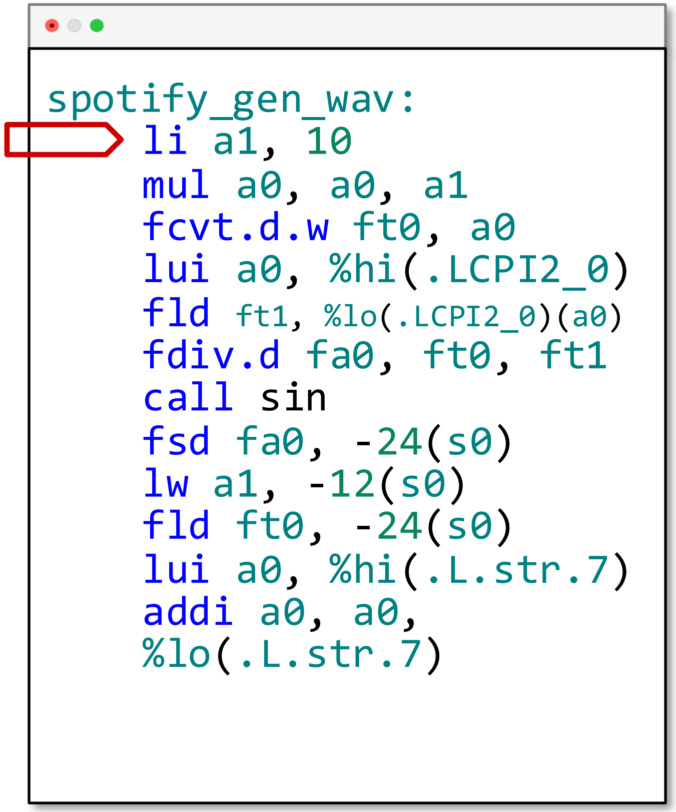
pc



```
chrome_http_response:
    sw a0, -12(s0)
    li a1, 1
    sw a1, -28(s0)
    li a1, 2
    sw a1, -32(s0)
    sh a0, -24(s0)
    lui a0, 2
    addi a0, a0, -112
    sw a0, -36(s0)
    call bind
    lw a1, -36(s0)
    sh a0, -24(s0)
    lui a0, %hi(.L.str)
    addi a0, a0,
    %lo(.L.str)
```

Spotify
Instructions

pc



```
spotify_gen_wav:
    li a1, 10
    mul a0, a0, a1
    fcvt.d.w ft0, a0
    lui a0, %hi(.LCPI2_0)
    fld ft1, %lo(.LCPI2_0)(a0)
    fdiv.d fa0, ft0, ft1
    call sin
    fsd fa0, -24(s0)
    lw a1, -12(s0)
    fld ft0, -24(s0)
    lui a0, %hi(.L.str.7)
    addi a0, a0,
    %lo(.L.str.7)
```

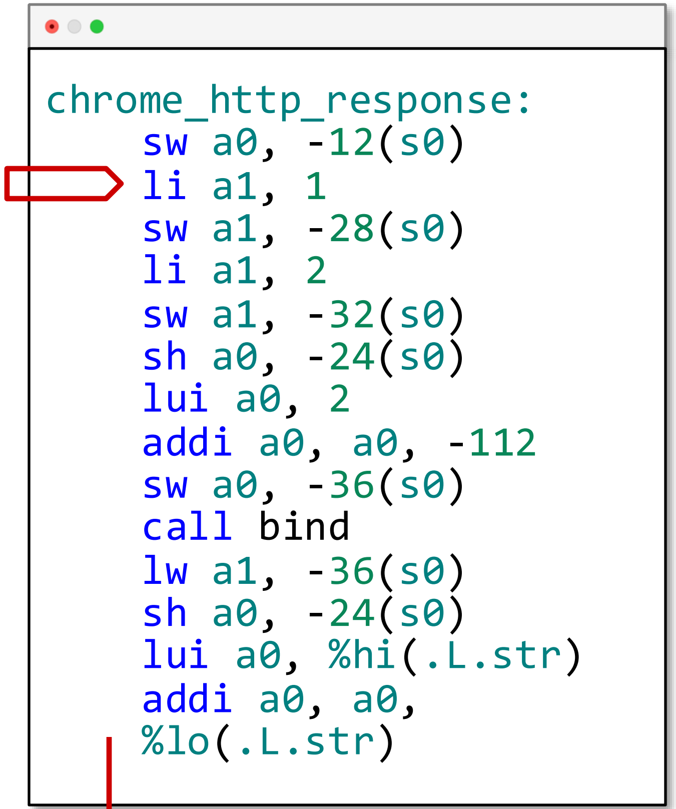
running chrome

*pc designates the nxt inst. to exec here.

What does concurrency look like?

Chrome
Instructions

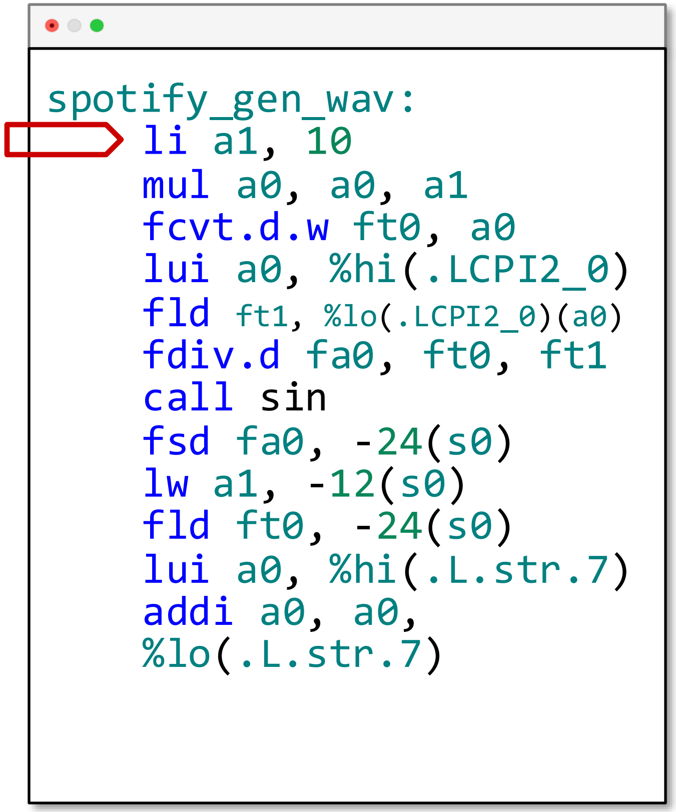
pc



```
chrome_http_response:
    sw a0, -12(s0)
    li a1, 1
    sw a1, -28(s0)
    li a1, 2
    sw a1, -32(s0)
    sh a0, -24(s0)
    lui a0, 2
    addi a0, a0, -112
    sw a0, -36(s0)
    call bind
    lw a1, -36(s0)
    sh a0, -24(s0)
    lui a0, %hi(.L.str)
    addi a0, a0,
    %lo(.L.str)
```

Spotify
Instructions

pc



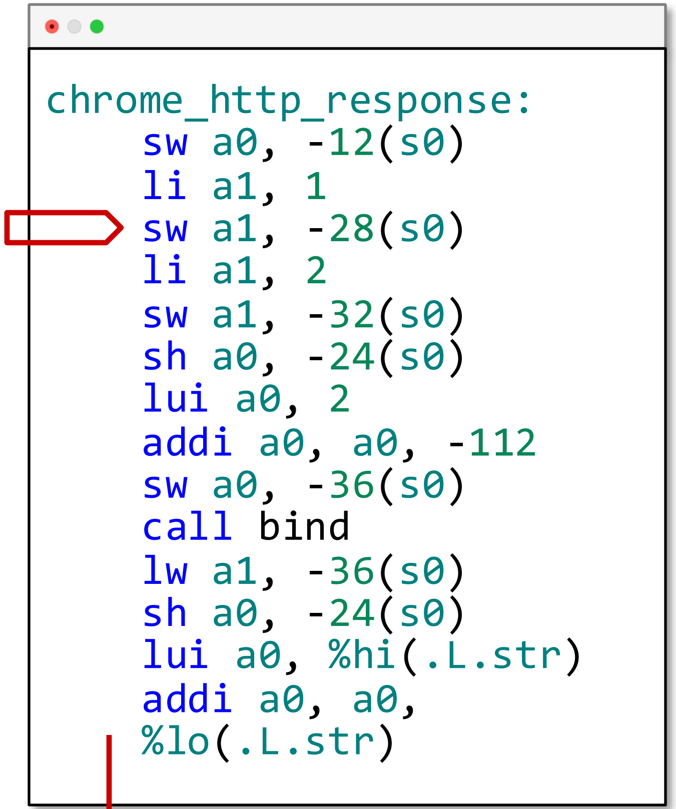
```
spotify_gen_wav:
    li a1, 10
    mul a0, a0, a1
    fcvt.d.w ft0, a0
    lui a0, %hi(.LCPI2_0)
    fld ft1, %lo(.LCPI2_0)(a0)
    fdiv.d fa0, ft0, ft1
    call sin
    fsd fa0, -24(s0)
    lw a1, -12(s0)
    fld ft0, -24(s0)
    lui a0, %hi(.L.str.7)
    addi a0, a0,
    %lo(.L.str.7)
```

running chrome

*pc designates the nxt inst. to exec here.

What does concurrency look like?

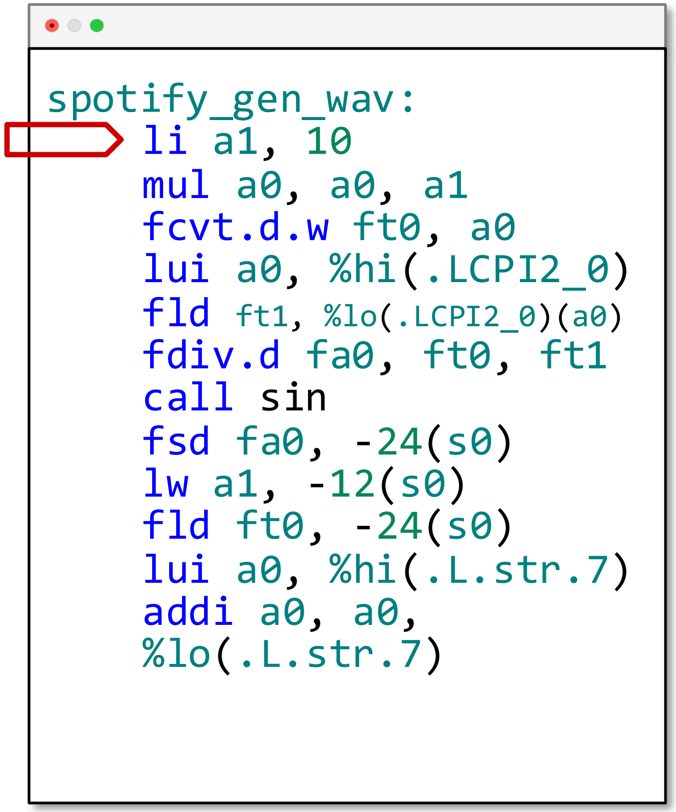
Chrome
Instructions



pc

```
chrome_http_response:
    sw a0, -12(s0)
    li a1, 1
    sw a1, -28(s0)
    li a1, 2
    sw a1, -32(s0)
    sh a0, -24(s0)
    lui a0, 2
    addi a0, a0, -112
    sw a0, -36(s0)
    call bind
    lw a1, -36(s0)
    sh a0, -24(s0)
    lui a0, %hi(.L.str)
    addi a0, a0,
    %lo(.L.str)
```

Spotify
Instructions



pc

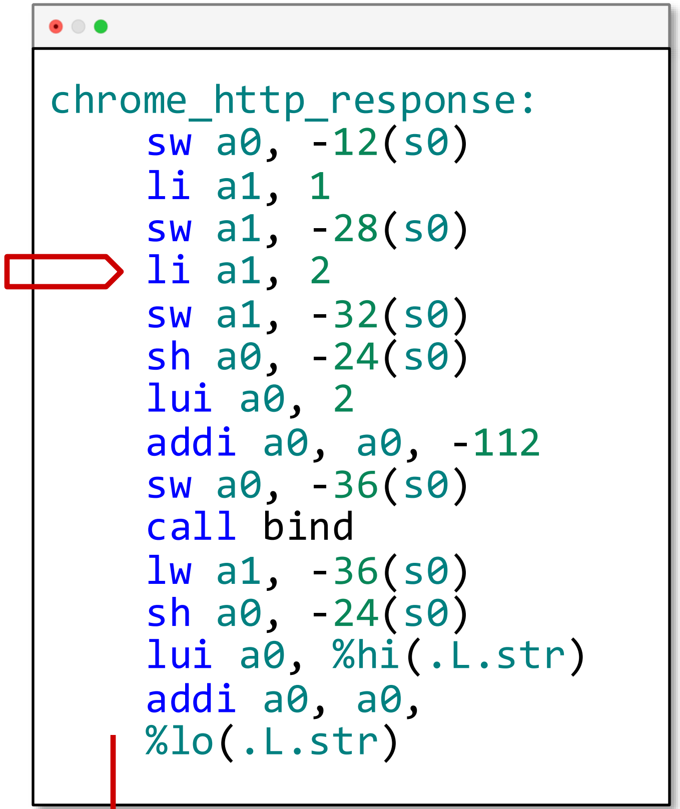
```
spotify_gen_wav:
    li a1, 10
    mul a0, a0, a1
    fcvt.d.w ft0, a0
    lui a0, %hi(.LCPI2_0)
    fld ft1, %lo(.LCPI2_0)(a0)
    fdiv.d fa0, ft0, ft1
    call sin
    fsd fa0, -24(s0)
    lw a1, -12(s0)
    fld ft0, -24(s0)
    lui a0, %hi(.L.str.7)
    addi a0, a0,
    %lo(.L.str.7)
```

running chrome

*pc designates the nxt inst. to exec here.

What does concurrency look like?

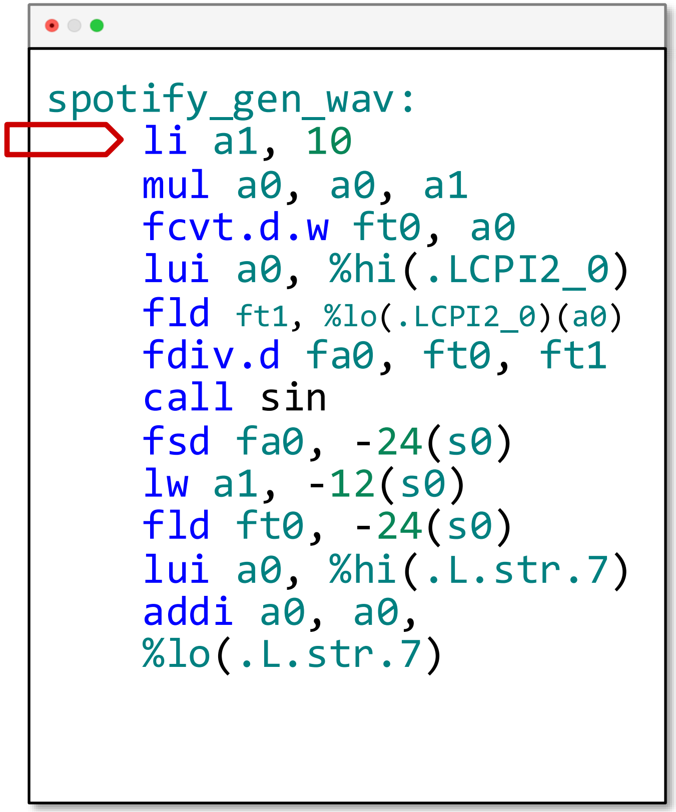
Chrome
Instructions



```
chrome_http_response:
    sw a0, -12(s0)
    li a1, 1
    sw a1, -28(s0)
    li a1, 2
    sw a1, -32(s0)
    sh a0, -24(s0)
    lui a0, 2
    addi a0, a0, -112
    sw a0, -36(s0)
    call bind
    lw a1, -36(s0)
    sh a0, -24(s0)
    lui a0, %hi(.L.str)
    addi a0, a0,
    %lo(.L.str)
```

pc

Spotify
Instructions



```
spotify_gen_wav:
    li a1, 10
    mul a0, a0, a1
    fcvt.d.w ft0, a0
    lui a0, %hi(.LCPI2_0)
    fld ft1, %lo(.LCPI2_0)(a0)
    fdiv.d fa0, ft0, ft1
    call sin
    fsd fa0, -24(s0)
    lw a1, -12(s0)
    fld ft0, -24(s0)
    lui a0, %hi(.L.str.7)
    addi a0, a0,
    %lo(.L.str.7)
```

pc

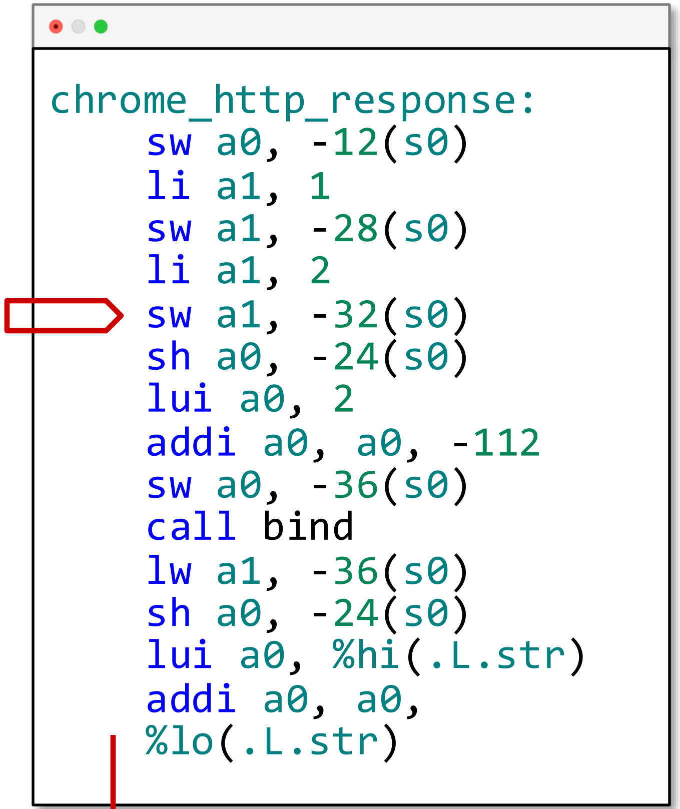
running chrome

*pc designates the nxt inst. to exec here.

What does concurrency look like?

Chrome
Instructions

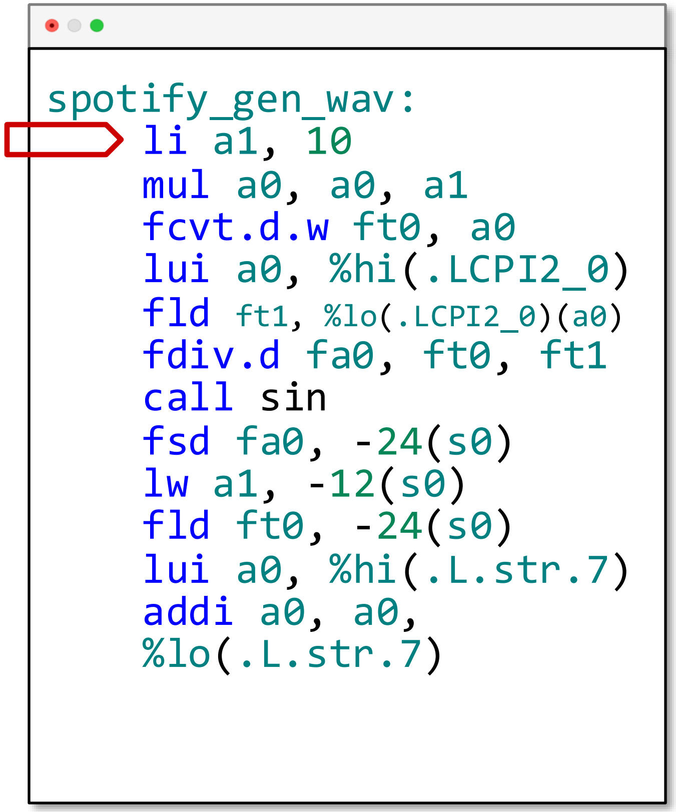
pc



```
chrome_http_response:
    sw a0, -12(s0)
    li a1, 1
    sw a1, -28(s0)
    li a1, 2
    sw a1, -32(s0)
    sh a0, -24(s0)
    lui a0, 2
    addi a0, a0, -112
    sw a0, -36(s0)
    call bind
    lw a1, -36(s0)
    sh a0, -24(s0)
    lui a0, %hi(.L.str)
    addi a0, a0,
    %lo(.L.str)
```

Spotify
Instructions

pc



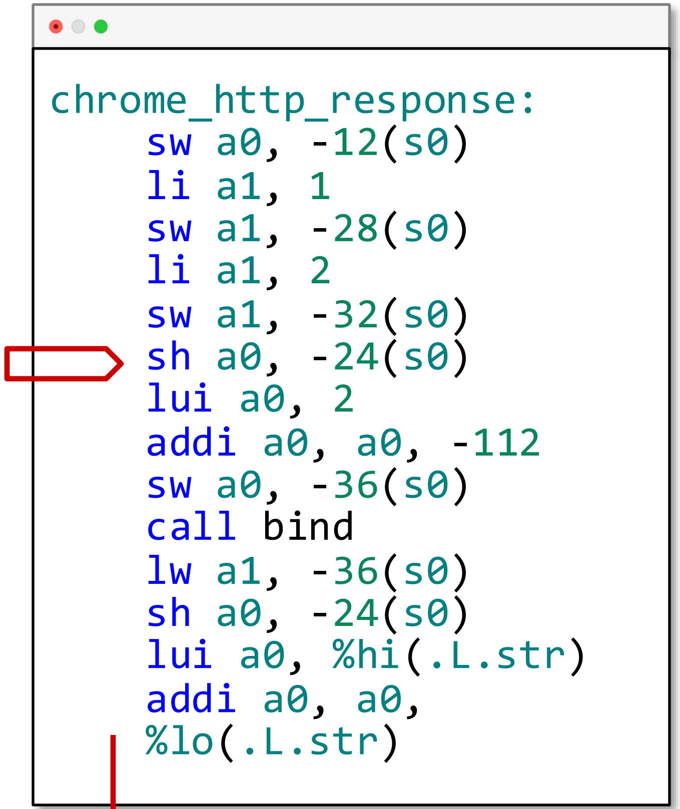
```
spotify_gen_wav:
    li a1, 10
    mul a0, a0, a1
    fcvt.d.w ft0, a0
    lui a0, %hi(.LCPI2_0)
    fld ft1, %lo(.LCPI2_0)(a0)
    fdiv.d fa0, ft0, ft1
    call sin
    fsd fa0, -24(s0)
    lw a1, -12(s0)
    fld ft0, -24(s0)
    lui a0, %hi(.L.str.7)
    addi a0, a0,
    %lo(.L.str.7)
```

running chrome

*pc designates the nxt inst. to exec here.

What does concurrency look like?

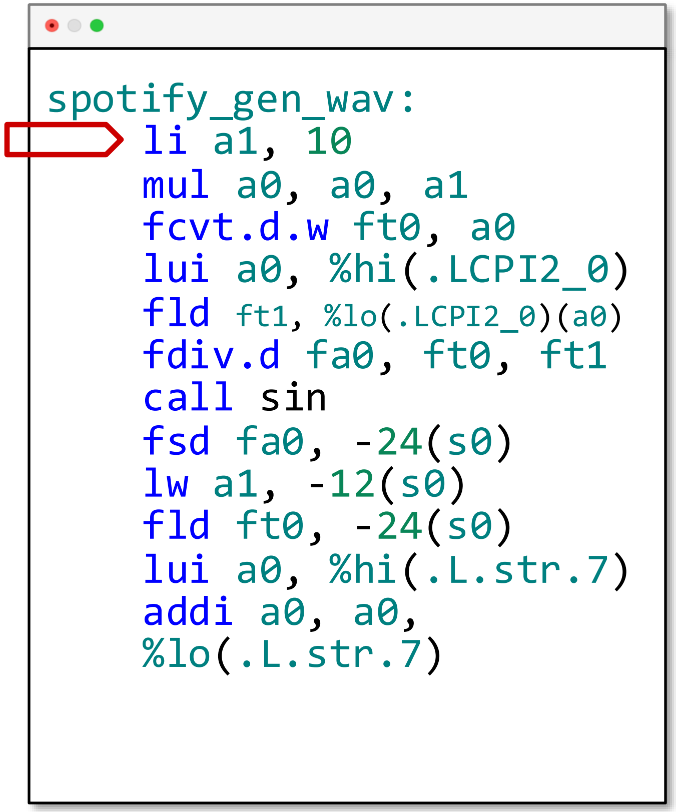
Chrome
Instructions



pc

```
chrome_http_response:
    sw a0, -12(s0)
    li a1, 1
    sw a1, -28(s0)
    li a1, 2
    sw a1, -32(s0)
    sh a0, -24(s0)
    lui a0, 2
    addi a0, a0, -112
    sw a0, -36(s0)
    call bind
    lw a1, -36(s0)
    sh a0, -24(s0)
    lui a0, %hi(.L.str)
    addi a0, a0,
    %lo(.L.str)
```

Spotify
Instructions



pc

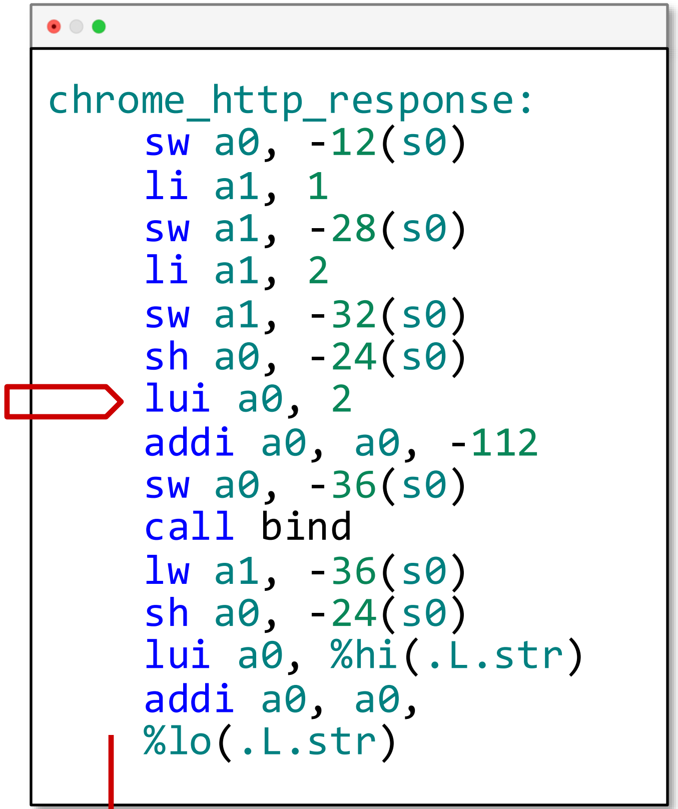
```
spotify_gen_wav:
    li a1, 10
    mul a0, a0, a1
    fcvt.d.w ft0, a0
    lui a0, %hi(.LCPI2_0)
    fld ft1, %lo(.LCPI2_0)(a0)
    fdiv.d fa0, ft0, ft1
    call sin
    fsd fa0, -24(s0)
    lw a1, -12(s0)
    fld ft0, -24(s0)
    lui a0, %hi(.L.str.7)
    addi a0, a0,
    %lo(.L.str.7)
```

running chrome

*pc designates the nxt inst. to exec here.

What does concurrency look like?

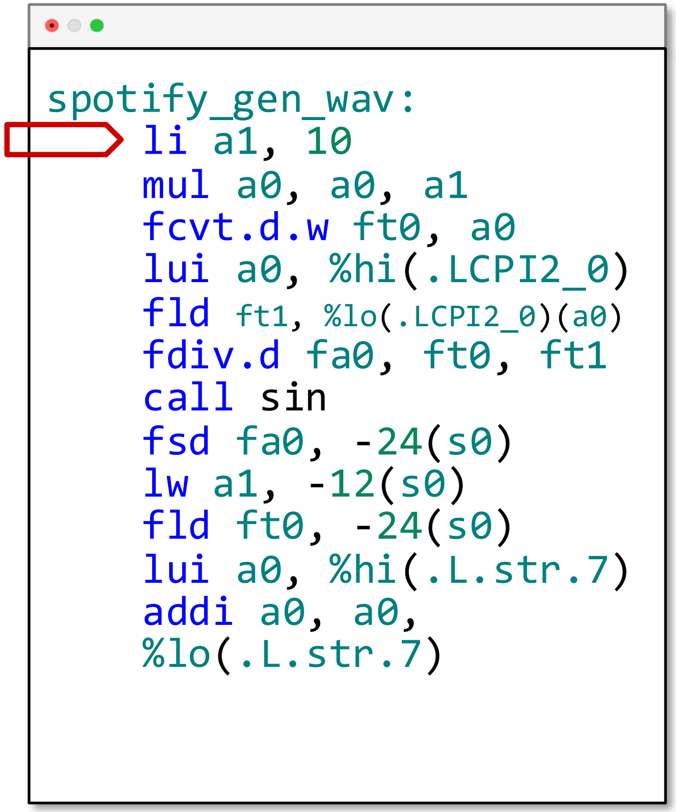
Chrome
Instructions



pc

```
chrome_http_response:
    sw a0, -12(s0)
    li a1, 1
    sw a1, -28(s0)
    li a1, 2
    sw a1, -32(s0)
    sh a0, -24(s0)
    lui a0, 2
    addi a0, a0, -112
    sw a0, -36(s0)
    call bind
    lw a1, -36(s0)
    sh a0, -24(s0)
    lui a0, %hi(.L.str)
    addi a0, a0,
    %lo(.L.str)
```

Spotify
Instructions



pc

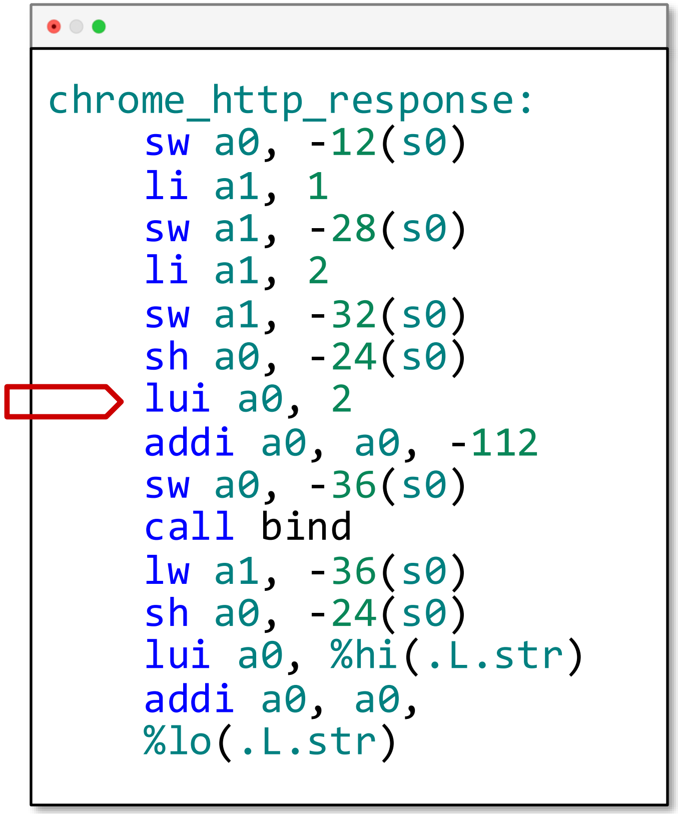
```
spotify_gen_wav:
    li a1, 10
    mul a0, a0, a1
    fcvt.d.w ft0, a0
    lui a0, %hi(.LCPI2_0)
    fld ft1, %lo(.LCPI2_0)(a0)
    fdiv.d fa0, ft0, ft1
    call sin
    fsd fa0, -24(s0)
    lw a1, -12(s0)
    fld ft0, -24(s0)
    lui a0, %hi(.L.str.7)
    addi a0, a0,
    %lo(.L.str.7)
```

running chrome

*pc designates the nxt inst. to exec here.

What does concurrency look like?

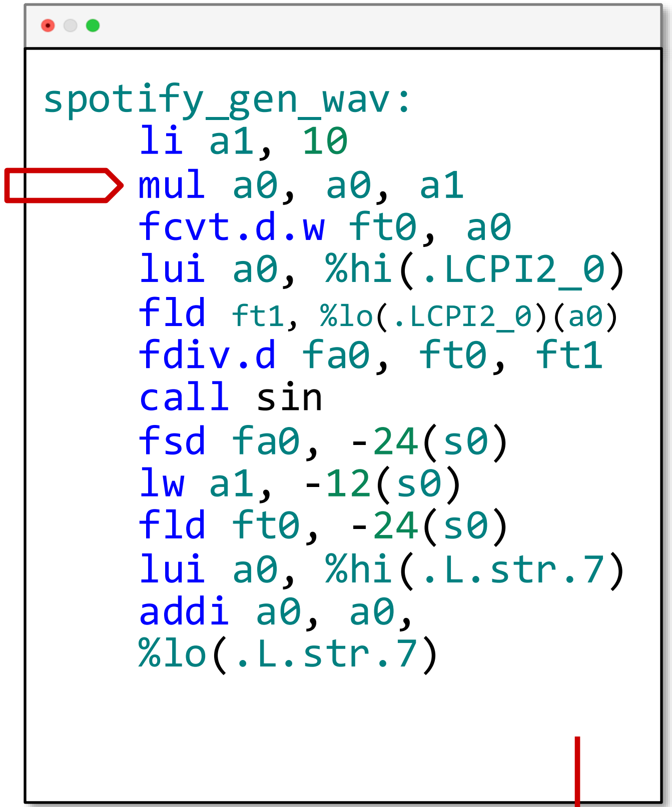
Chrome
Instructions



```
chrome_http_response:
    sw a0, -12(s0)
    li a1, 1
    sw a1, -28(s0)
    li a1, 2
    sw a1, -32(s0)
    sh a0, -24(s0)
    lui a0, 2
    addi a0, a0, -112
    sw a0, -36(s0)
    call bind
    lw a1, -36(s0)
    sh a0, -24(s0)
    lui a0, %hi(.L.str)
    addi a0, a0,
    %lo(.L.str)
```

pc

Spotify
Instructions



```
spotify_gen_wav:
    li a1, 10
    mul a0, a0, a1
    fcvt.d.w ft0, a0
    lui a0, %hi(.LCPI2_0)
    fld ft1, %lo(.LCPI2_0)(a0)
    fdiv.d fa0, ft0, ft1
    call sin
    fsd fa0, -24(s0)
    lw a1, -12(s0)
    fld ft0, -24(s0)
    lui a0, %hi(.L.str.7)
    addi a0, a0,
    %lo(.L.str.7)
```

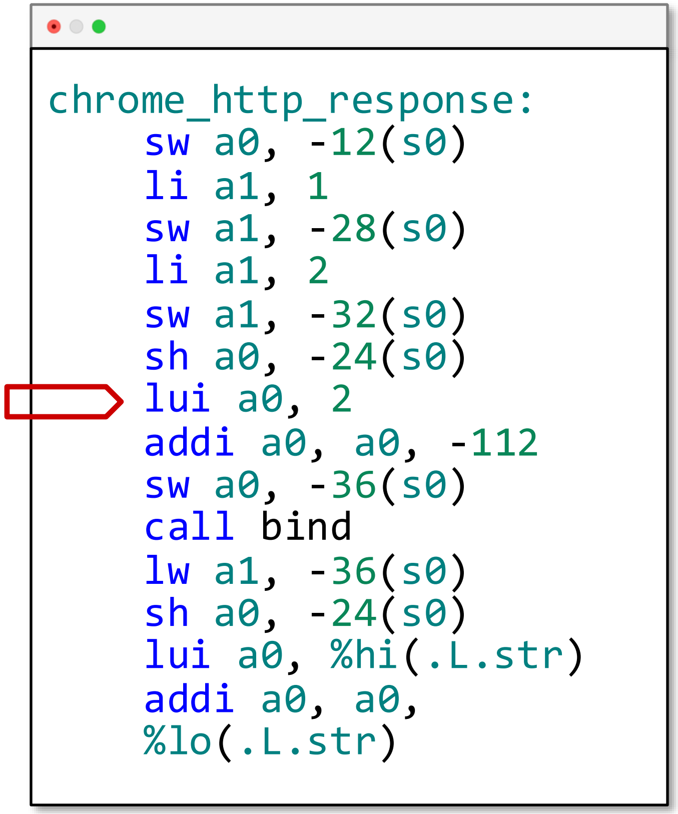
pc

running spotify

*pc designates the nxt inst. to exec here.

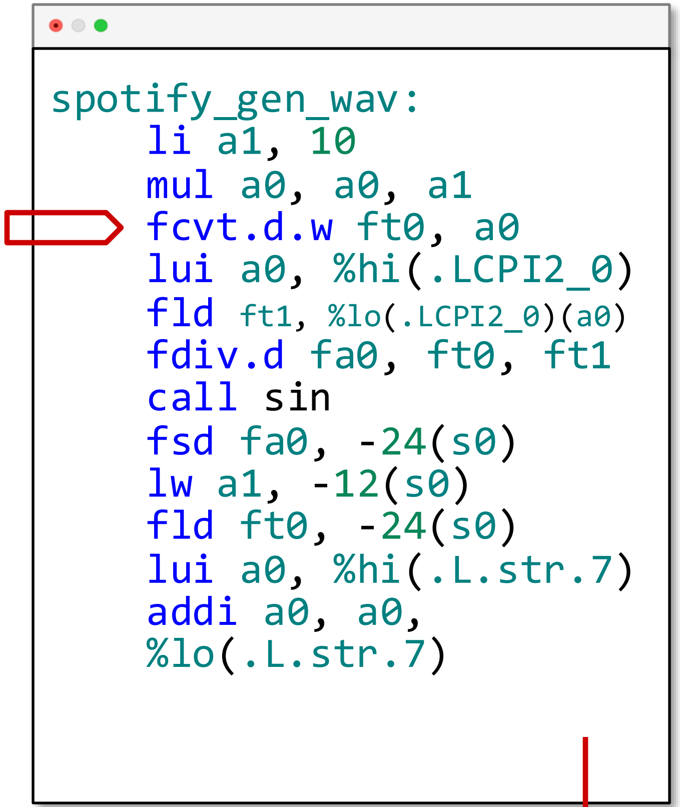
What does concurrency look like?

Chrome
Instructions



```
         chrome_http_response:
              sw a0, -12(s0)
              li a1, 1
              sw a1, -28(s0)
              li a1, 2
              sw a1, -32(s0)
              sh a0, -24(s0)
pc ⇨          lui a0, 2
              addi a0, a0, -112
              sw a0, -36(s0)
              call bind
              lw a1, -36(s0)
              sh a0, -24(s0)
              lui a0, %hi(.L.str)
              addi a0, a0,
              %lo(.L.str)
```

Spotify
Instructions



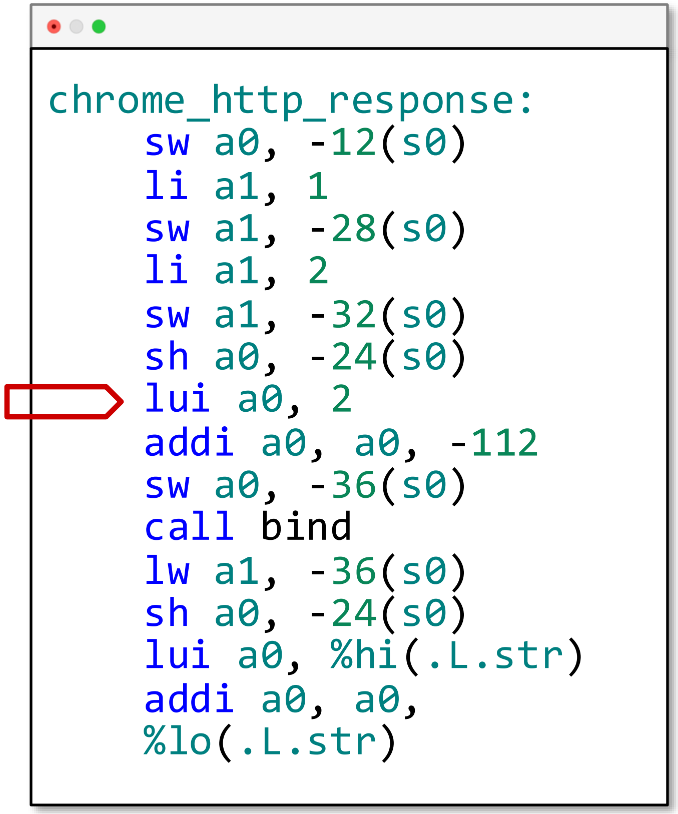
```
         spotify_gen_wav:
              li a1, 10
              mul a0, a0, a1
pc ⇨          fcvt.d.w ft0, a0
              lui a0, %hi(.LCPI2_0)
              fld ft1, %lo(.LCPI2_0)(a0)
              fdiv.d fa0, ft0, ft1
              call sin
              fsd fa0, -24(s0)
              lw a1, -12(s0)
              fld ft0, -24(s0)
              lui a0, %hi(.L.str.7)
              addi a0, a0,
              %lo(.L.str.7)
```

running spotify

*pc designates the nxt inst. to exec here.

What does concurrency look like?

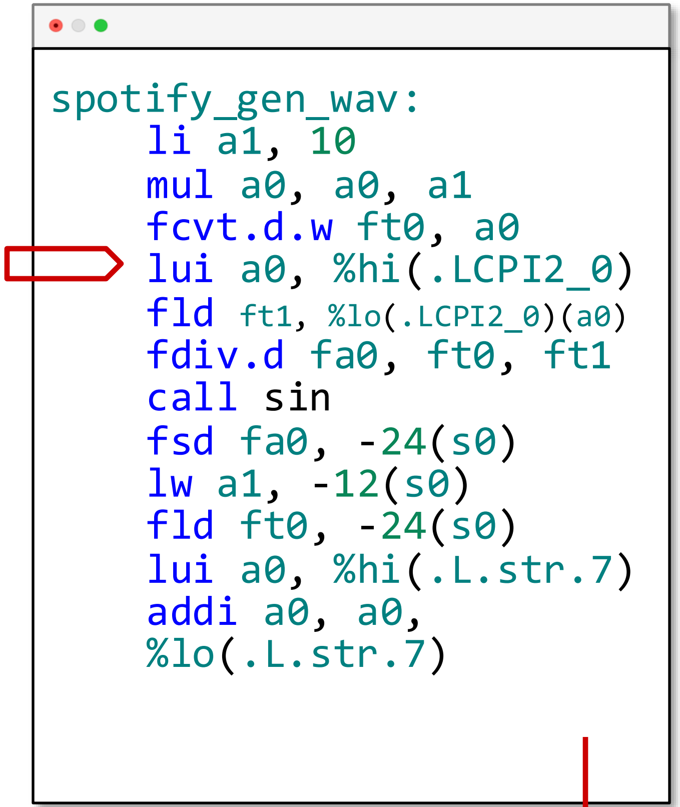
Chrome
Instructions



```
chrome_http_response:
    sw a0, -12(s0)
    li a1, 1
    sw a1, -28(s0)
    li a1, 2
    sw a1, -32(s0)
    sh a0, -24(s0)
    lui a0, 2
    addi a0, a0, -112
    sw a0, -36(s0)
    call bind
    lw a1, -36(s0)
    sh a0, -24(s0)
    lui a0, %hi(.L.str)
    addi a0, a0,
    %lo(.L.str)
```

pc

Spotify
Instructions



```
spotify_gen_wav:
    li a1, 10
    mul a0, a0, a1
    fcvt.d.w ft0, a0
    lui a0, %hi(.LCPI2_0)
    fld ft1, %lo(.LCPI2_0)(a0)
    fdiv.d fa0, ft0, ft1
    call sin
    fsd fa0, -24(s0)
    lw a1, -12(s0)
    fld ft0, -24(s0)
    lui a0, %hi(.L.str.7)
    addi a0, a0,
    %lo(.L.str.7)
```

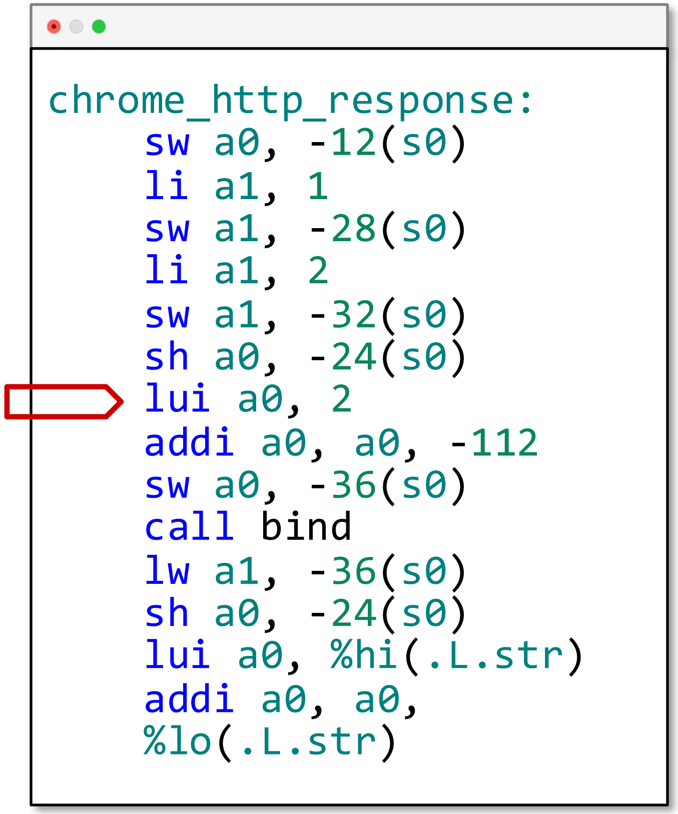
pc

running spotify

*pc designates the nxt inst. to exec here.

What does concurrency look like?

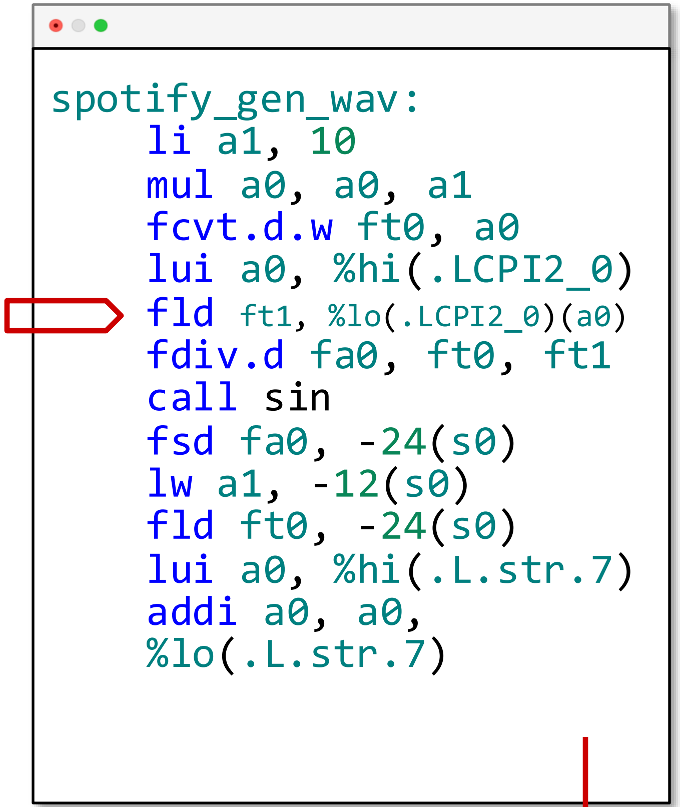
Chrome Instructions



```
chrome_http_response:
    sw a0, -12(s0)
    li a1, 1
    sw a1, -28(s0)
    li a1, 2
    sw a1, -32(s0)
    sh a0, -24(s0)
    lui a0, 2
    addi a0, a0, -112
    sw a0, -36(s0)
    call bind
    lw a1, -36(s0)
    sh a0, -24(s0)
    lui a0, %hi(.L.str)
    addi a0, a0,
    %lo(.L.str)
```

pc

Spotify Instructions



```
spotify_gen_wav:
    li a1, 10
    mul a0, a0, a1
    fcvt.d.w ft0, a0
    lui a0, %hi(.LCPI2_0)
    fld ft1, %lo(.LCPI2_0)(a0)
    fdiv.d fa0, ft0, ft1
    call sin
    fsd fa0, -24(s0)
    lw a1, -12(s0)
    fld ft0, -24(s0)
    lui a0, %hi(.L.str.7)
    addi a0, a0,
    %lo(.L.str.7)
```

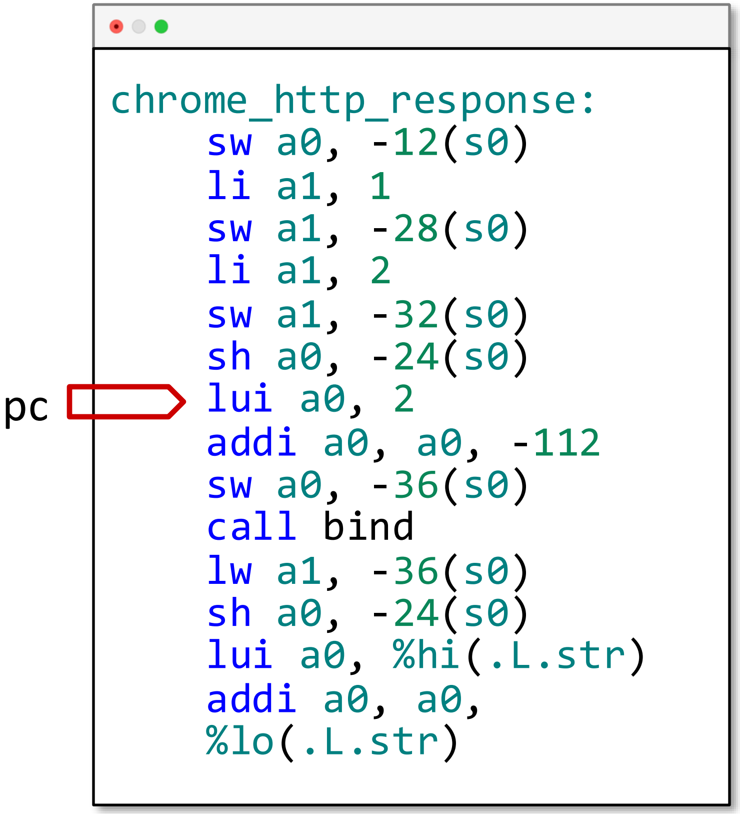
pc

running spotify

*pc designates the nxt inst. to exec here.

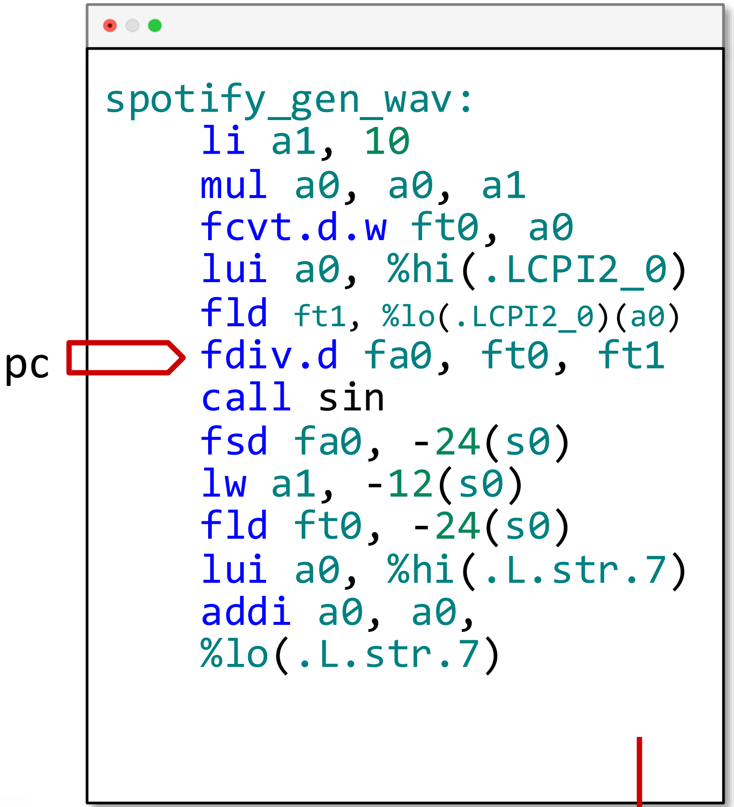
What does concurrency look like?

Chrome
Instructions



```
chrome_http_response:
    sw a0, -12(s0)
    li a1, 1
    sw a1, -28(s0)
    li a1, 2
    sw a1, -32(s0)
    sh a0, -24(s0)
    lui a0, 2
    addi a0, a0, -112
    sw a0, -36(s0)
    call bind
    lw a1, -36(s0)
    sh a0, -24(s0)
    lui a0, %hi(.L.str)
    addi a0, a0,
    %lo(.L.str)
```

Spotify
Instructions



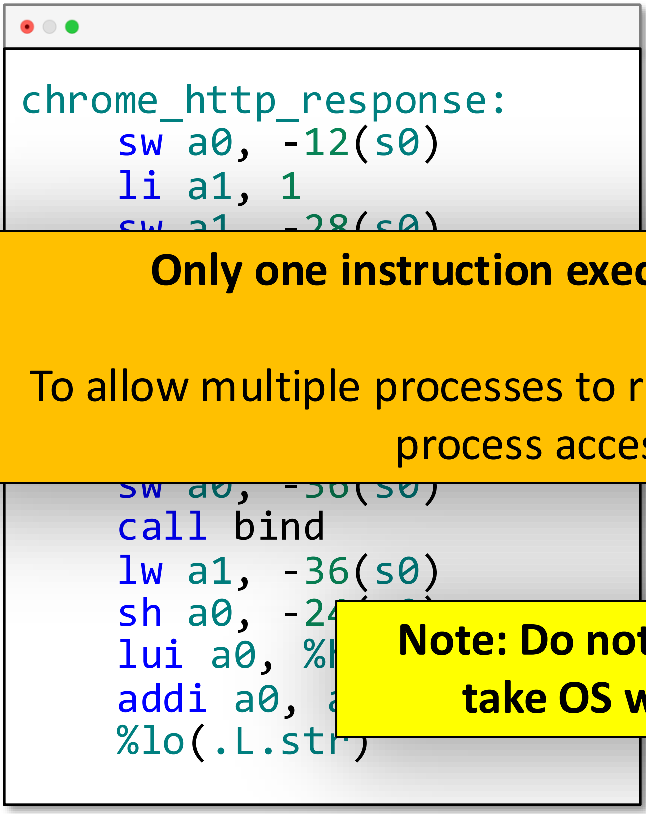
```
spotify_gen_wav:
    li a1, 10
    mul a0, a0, a1
    fcvt.d.w ft0, a0
    lui a0, %hi(.LCPI2_0)
    fld ft1, %lo(.LCPI2_0)(a0)
    fdiv.d fa0, ft0, ft1
    call sin
    fsd fa0, -24(s0)
    lw a1, -12(s0)
    fld ft0, -24(s0)
    lui a0, %hi(.L.str.7)
    addi a0, a0,
    %lo(.L.str.7)
```

running spotify

*pc designates the nxt inst. to exec here.

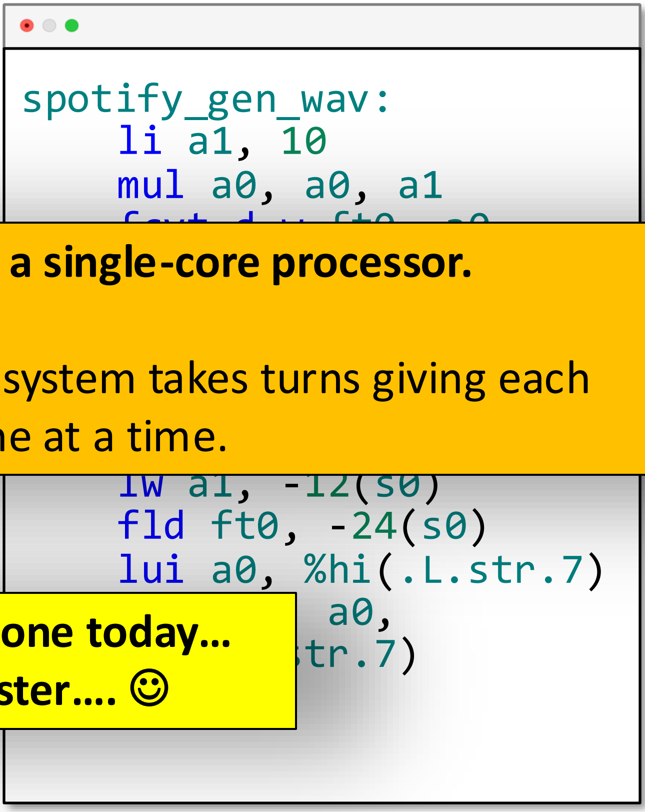
What does concurrency look like?

Chrome
Instructions



```
chrome_http_response:
    sw a0, -12(s0)
    li a1, 1
    sw a1, -28(s0)
    sw a0, -36(s0)
    call bind
    lw a1, -36(s0)
    sh a0, -24
    lui a0, %h
    addi a0, a
    %lo(.L.str)
```

Spotify
Instructions



```
spotify_gen_wav:
    li a1, 10
    mul a0, a0, a1
    fcvt.d.w ft0, a0
    lw a1, -12(s0)
    fld ft0, -24(s0)
    lui a0, %hi(.L.str.7)
              a0,
        str.7)
```

p

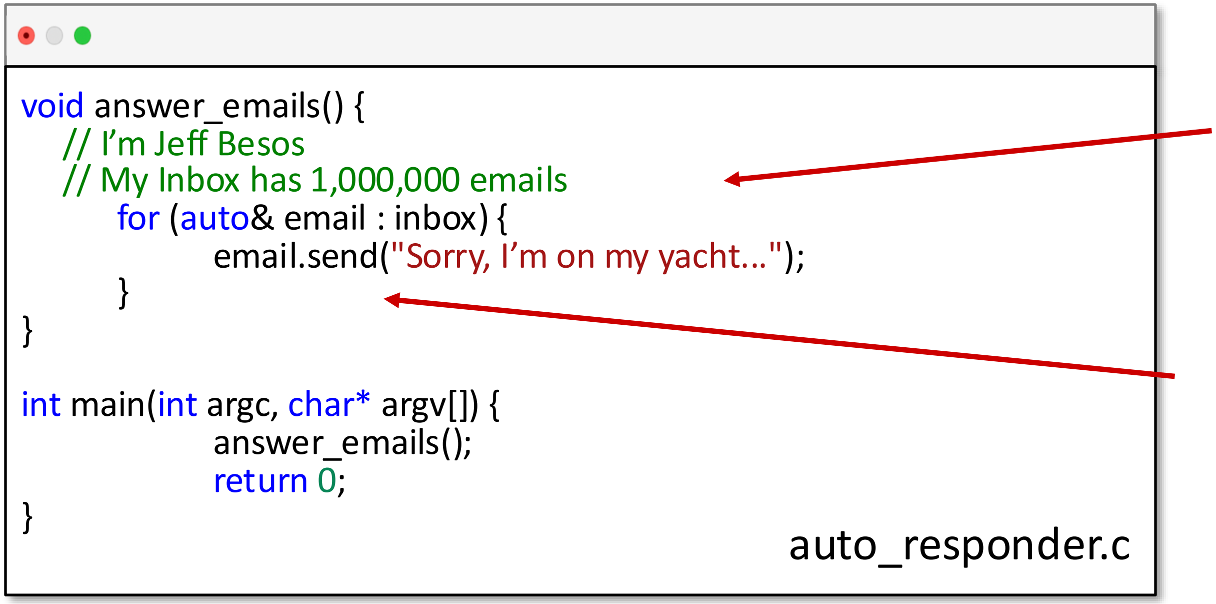
Only one instruction executes at a time on a single-core processor.

To allow multiple processes to run, the operating system takes turns giving each process access to the CORE, one at a time.

Note: Do not ask how this is done today…
take OS with us next semester…. ☺

*pc designates the nxt inst. to exec here.

Let’s bring back this code



```
void answer_emails() {
    // I’m Jeff Besos
    // My Inbox has 1,000,000 emails
        for (auto& email : inbox) {
                email.send("Sorry, I’m on my yacht...");
        }
}

int main(int argc, char* argv[]) {
                answer_emails();
                return 0;
}
```

auto_responder.c

Looping:
  keeping track of email
  loading email into mem

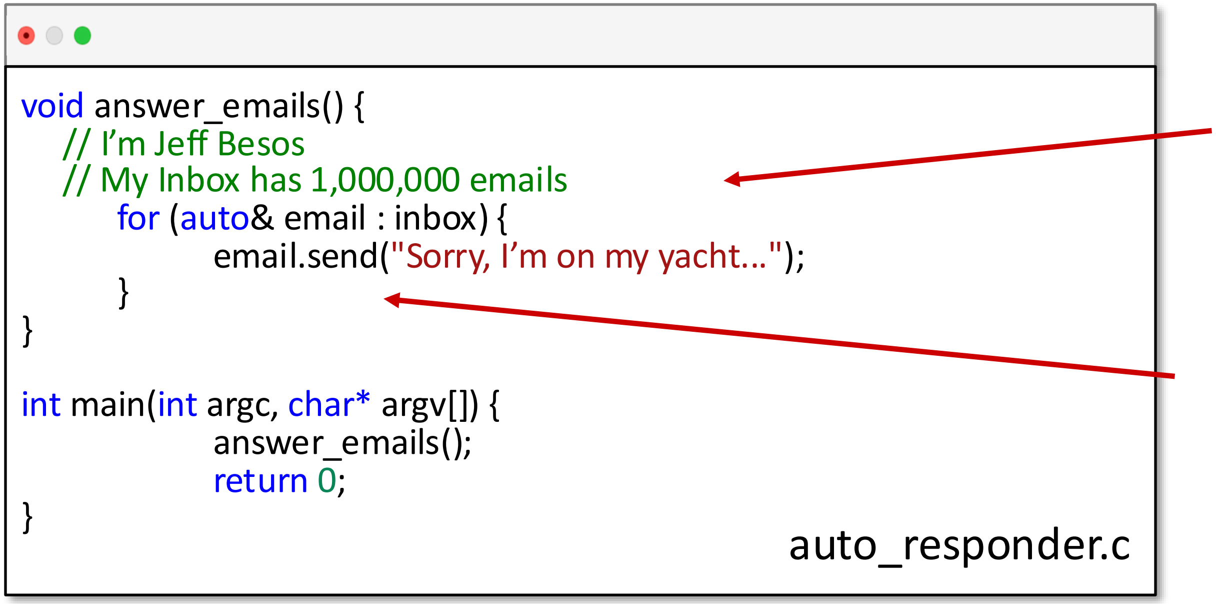
Send:
  creates email
  formats data for response
  connection to email server
  send over network!

Most of these operations are relatively quick!

Except for one…

Connecting to an email server and sending an email can easily take 50+ ms.

Time Analysis



```
void answer_emails() {
    // I’m Jeff Besos
    // My Inbox has 1,000,000 emails
        for (auto& email : inbox) {
                email.send("Sorry, I’m on my yacht...");
        }
}

int main(int argc, char* argv[]) {
                answer_emails();
                return 0;
}
```

auto_responder.c

Looping:
  keeping track of email
  loading email into mem

Send:
  creates email
  formats data for response
  connection to email server
  send over network!

Total Elapsed Time

10 ms

Cal Current Email
Loading Email
invoking send()

150 ms

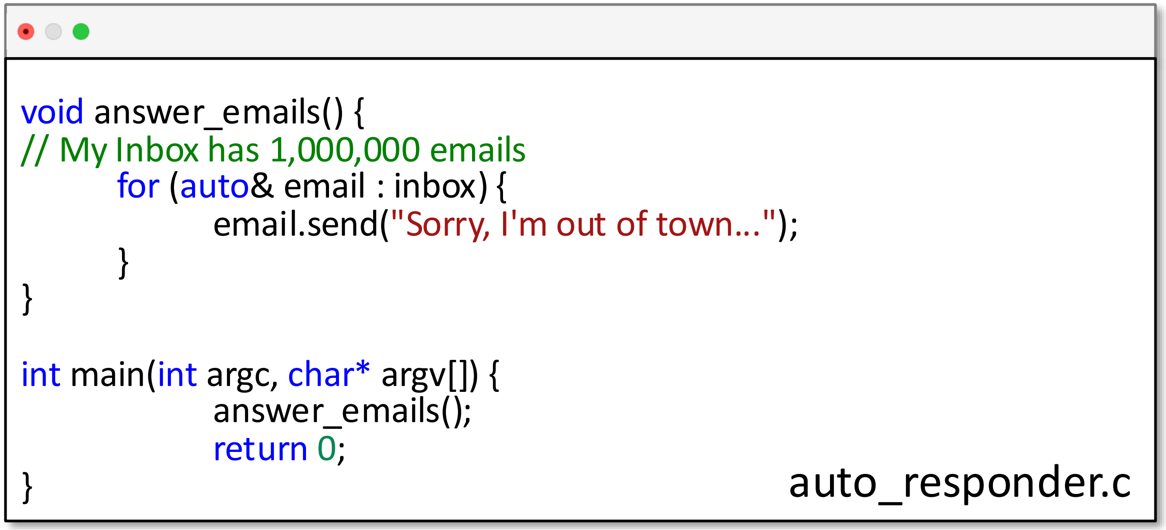
Time it takes to send the email over the network

This means ***calling the function***, not *doing the sending over the network*

Poll Everywhere

pollev.com/cis2400

What’s so wrong with this? Discuss!



```
void answer_emails() {
// My Inbox has 1,000,000 emails
        for (auto& email : inbox) {
                email.send("Sorry, I'm out of town…");
        }
}

int main(int argc, char* argv[]) {
                answer_emails();
                return 0;
}
```

auto_responder.c

10 ms

Cal Current Email
Loading Email
invoking send()

150 ms

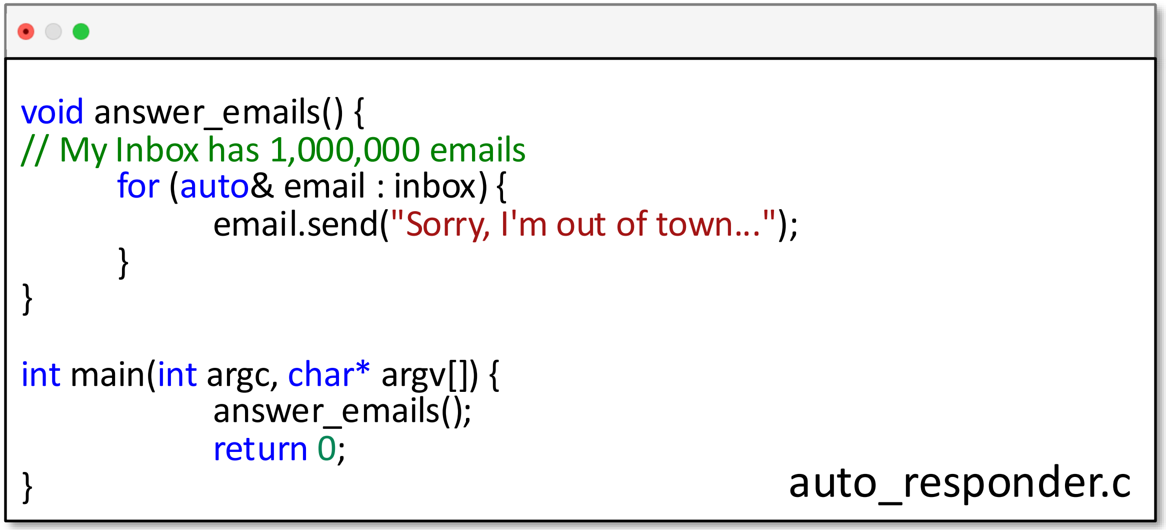
Time it takes to send the email over the network

Put your short responses into poll everywhere ☺

Poll Everywhere

pollev.com/cis2400

What do we spend most of our time doing?



```
void answer_emails() {
// My Inbox has 1,000,000 emails
        for (auto& email : inbox) {
                email.send("Sorry, I'm out of town...");
        }
}

int main(int argc, char* argv[]) {
                answer_emails();
                return 0;
}
```

auto_responder.c

10 ms

Cal Current Email

Loading Email

invoking send()

150 ms

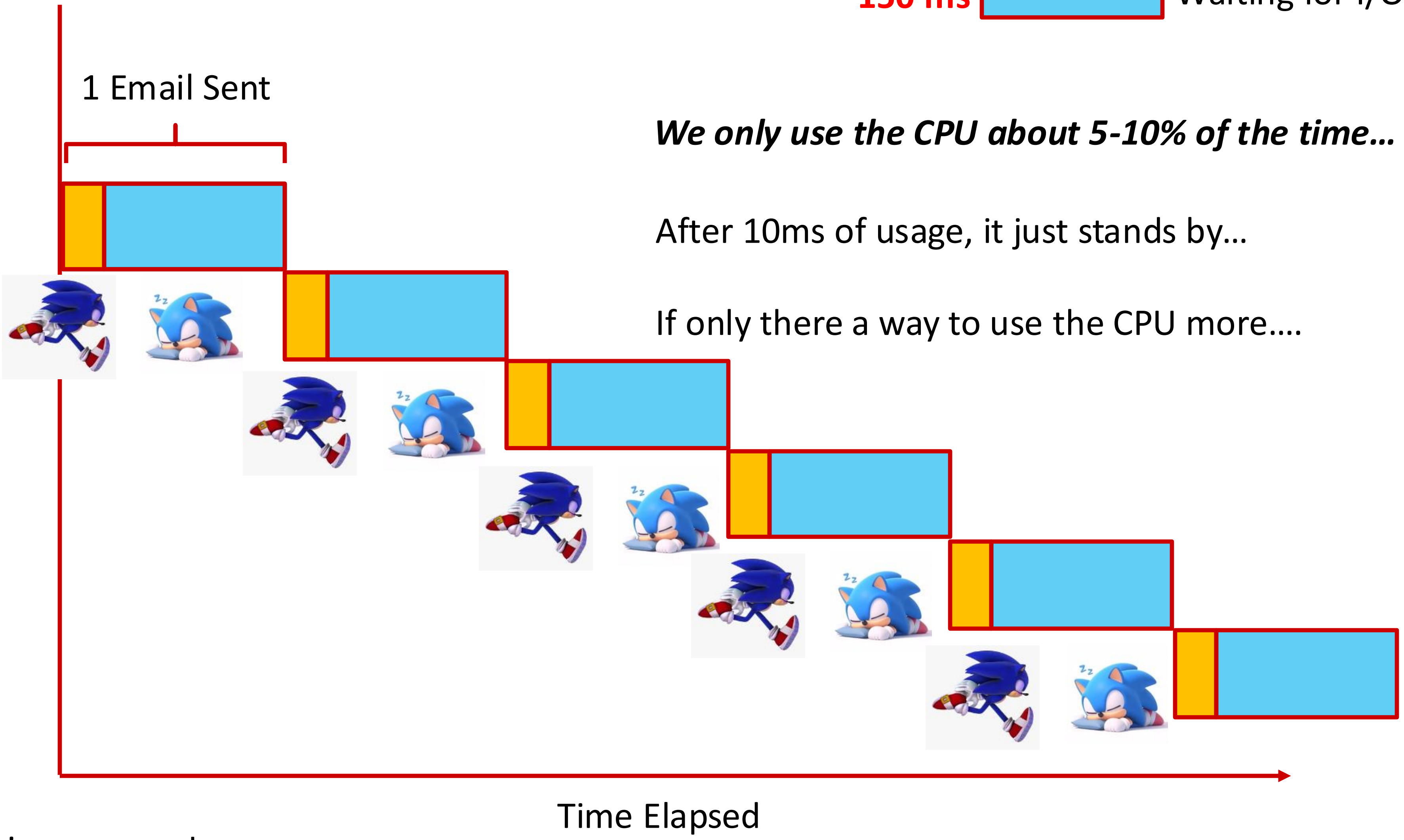
Time it takes to send the email over the network

- We spend more time in the I/O operations
  - Establishing Connection with Email Server
  - Sending Email Over Network

Let’s Visualize the Bottleneck

10 ms 🟨 Using CPU

150 ms 🟦 Waiting for I/O



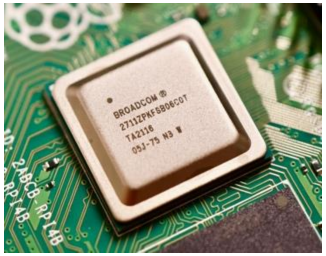
We only use the CPU about 5-10% of the time…

After 10ms of usage, it just stands by…

If only there a way to use the CPU more….

*yes it’s not to scale

CPU Utilization

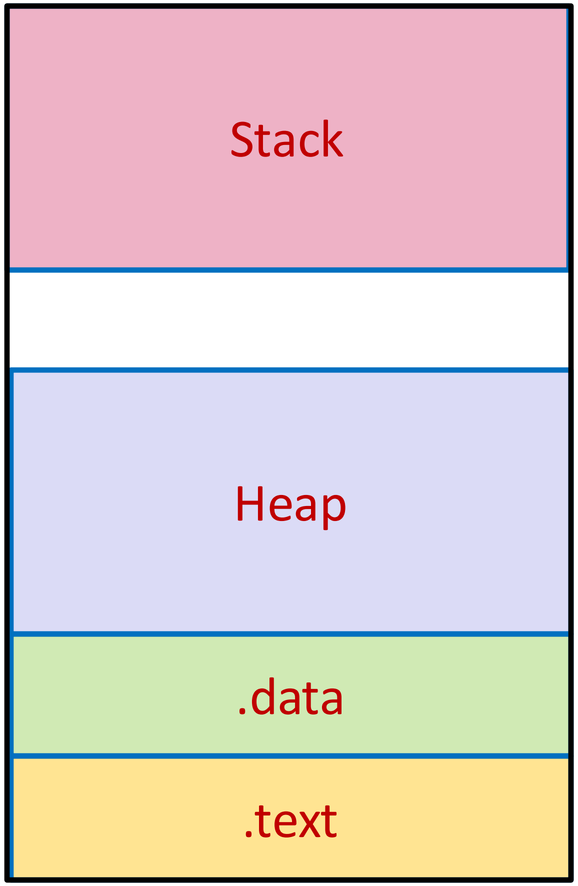


- When a process waits for I/O, the CPU remains idle, wasting valuable processing time.
- Our goal is to ***maximize CPU utilization*** and ensure it stays actively engaged in useful work.
- What if we could send the next email while waiting for the network I/O?

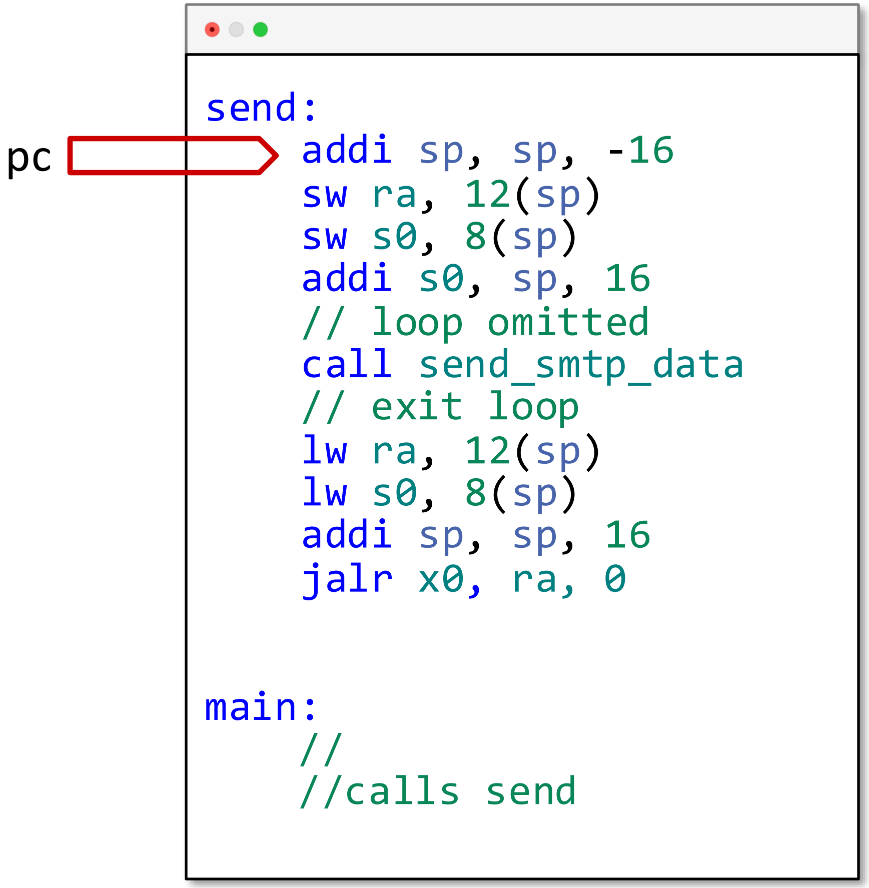
Before Threads, There Were Processes

- A "program" in ***execution*** is called a ***process***.

Memory Layout (State)



Instructions Executed

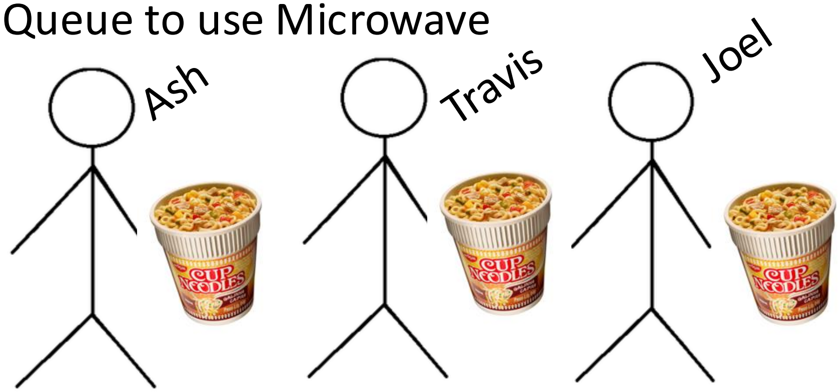


```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0


main:
    //
    //calls send
```

Sharing the CPU: It’s like a microwave

- When everyone wants to make cup of ramen at 2AM in your dorm, you probably *share* a microwave
  - (unless you all have a microwave in each of your rooms then this is an example of parallelism via multiple CPU Cores, don’t ask…)

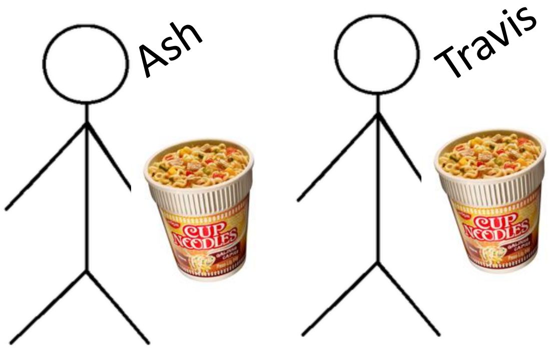
Queue to use Microwave



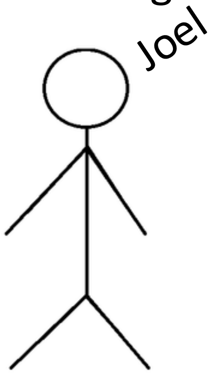
Sharing the CPU: It’s like a microwave

- When everyone wants to make cup of ramen at 2AM in your dorm, you probably *share* a microwave
  - (unless you all have a microwave in each of your rooms then this is an example of parallelism via multiple CPU Cores, don’t ask…)

Queue to use Microwave



Waiting For Microwave:

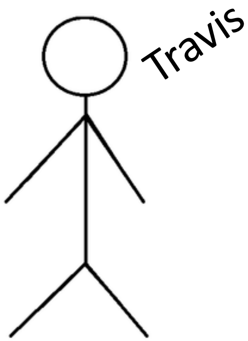


Sharing the CPU: It’s like a microwave

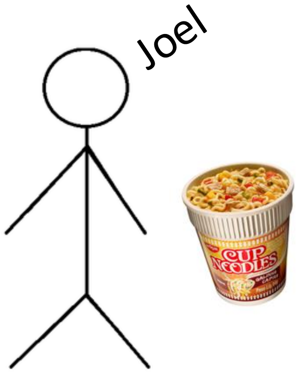
- When everyone wants to make cup of ramen at 2AM in your dorm, you probably *share* a microwave
  - (unless you all have a microwave in each of your rooms then this is an example of parallelism via multiple CPU Cores, don’t ask…)

Queue to use Microwave

Waiting For Microwave:



Done:

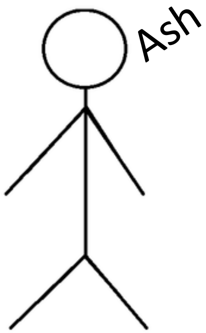


Sharing the CPU: It’s like a microwave

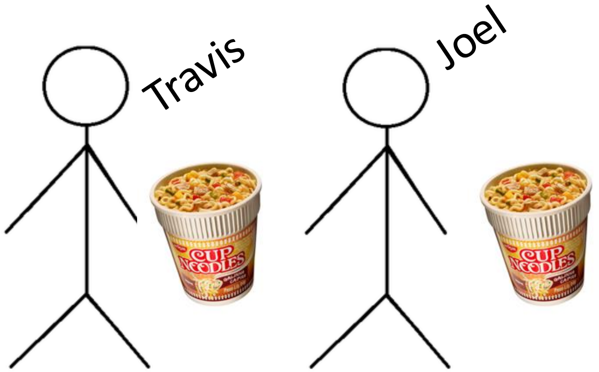
- When everyone wants to make cup of ramen at 2AM in your dorm, you probably *share* a microwave
  - (unless you all have a microwave in each of your rooms then this is an example of parallelism via multiple CPU Cores, don’t ask…)

Queue to use Microwave

Waiting For Microwave:



Done:



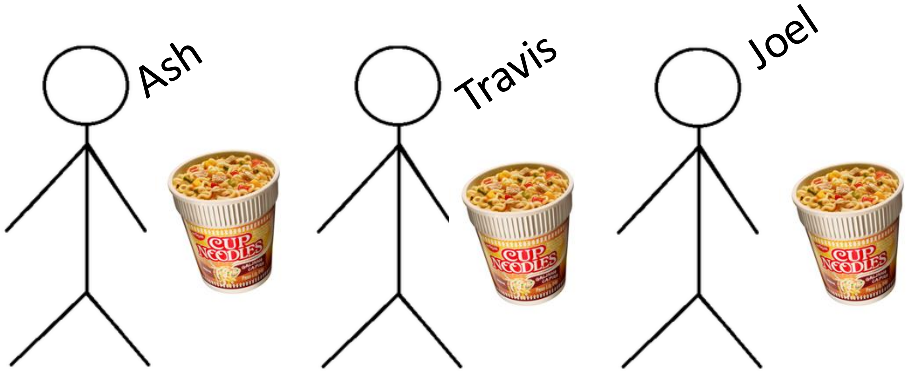
Sharing the CPU: It's like a microwave

- When everyone wants to make cup of ramen at 2AM in your dorm, you probably *share* a microwave
  - (unless you all have a microwave in each of your rooms then this is an example of parallelism via multiple CPU Cores, don't ask…)

Did you notice how the microwave was always being used?

We achieved 99% Microwave Utilization

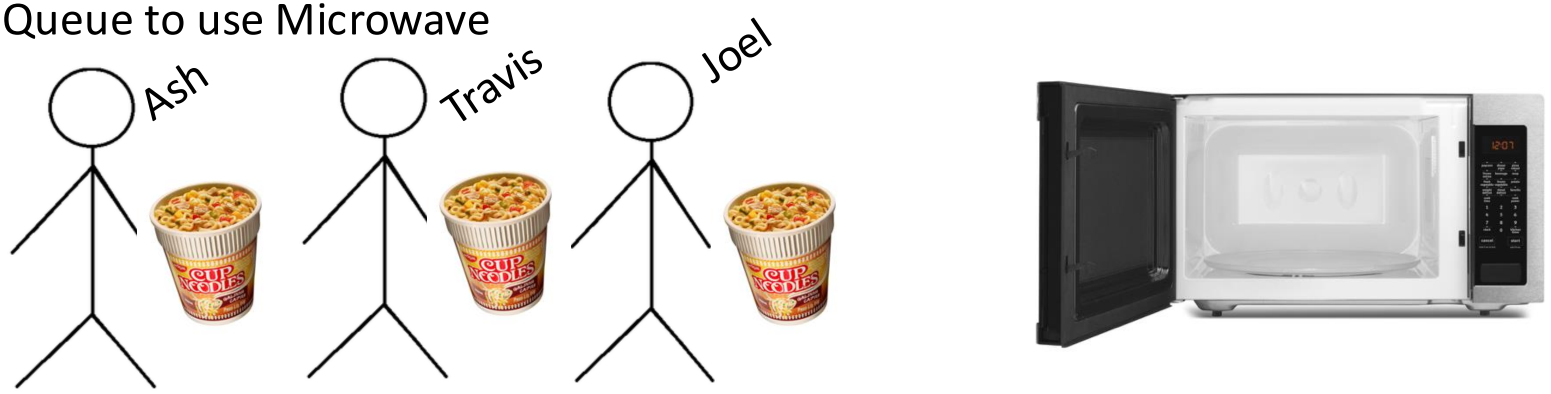
Done:



Sharing the Microwave Without Threads

- When everyone wants to make cup of ramen at 2AM in your dorm, you probably *share* a microwave
  - (unless you all have a microwave in each of your rooms then this is an example of parallelism via multiple CPU Cores, don’t ask…)

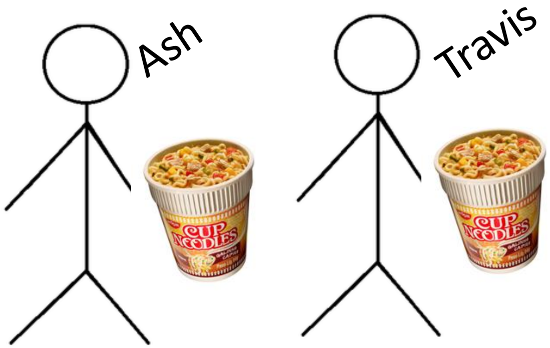
Queue to use Microwave



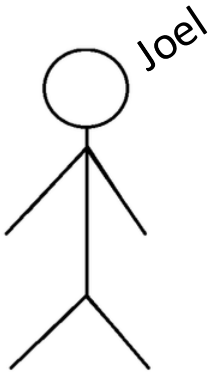
Sharing the Microwave Without Threads

- When everyone wants to make cup of ramen at 2AM in your dorm, you probably *share* a microwave
  - (unless you all have a microwave in each of your rooms then this is an example of parallelism via multiple CPU Cores, don’t ask…)

Queue to use Microwave



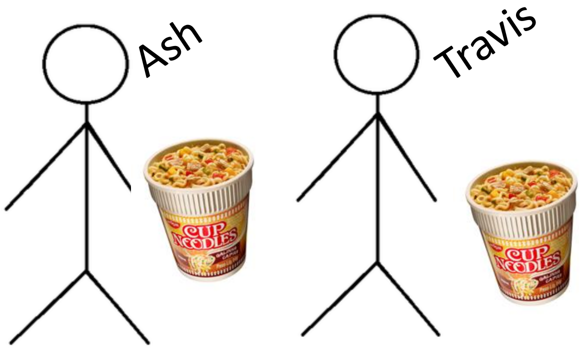
Waiting For Microwave:



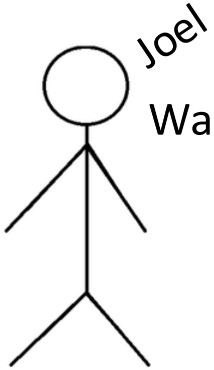
Sharing the Microwave Without Threads

- When everyone wants to make cup of ramen at 2AM in your dorm, you probably *share* a microwave
  - (unless you all have a microwave in each of your rooms then this is an example of parallelism via multiple CPU Cores, don't ask…)

Queue to use Microwave



Waiting For Microwave:

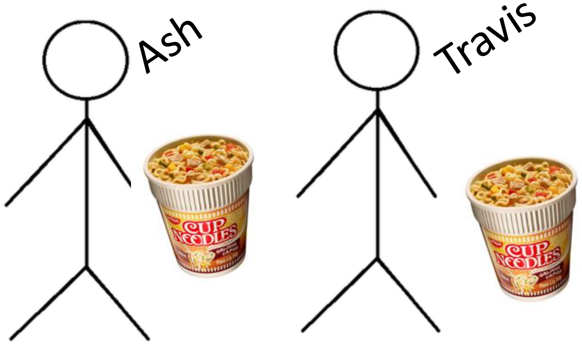


Wait!!!! You can't use it yet -- it has to cool down!

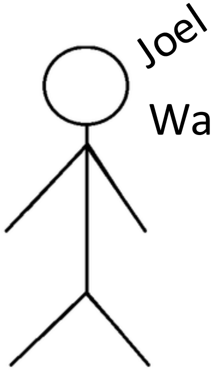
Sharing the Microwave Without Threads

- When everyone wants to make cup of ramen at 2AM in your dorm, you probably *share* a microwave
  - (unless you all have a microwave in each of your rooms then this is an example of parallelism via multiple CPU Cores, don't ask…)

Queue to use Microwave



Waiting For Microwave:



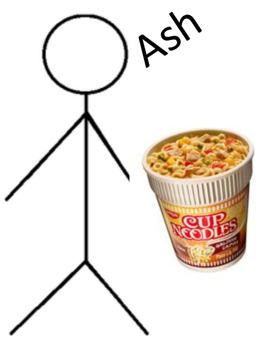
Wait!!!! You can't use it yet -- it has to cool down!

So even if the microwave isn't in use
the person takes ***ownership* of the microwave and hogs it…**

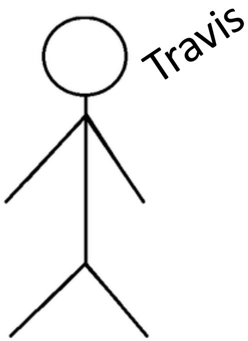
Sharing the Microwave Without Threads

- As you can see, this is incredibly inefficient…

Queue to use Microwave



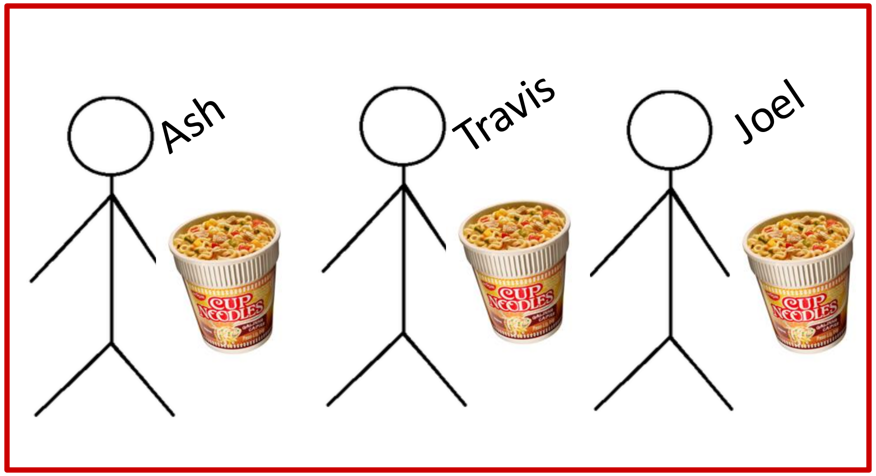
Waiting For Microwave:



Done:

Process vs Threads?

One Process: **`./cook_ramen_together`**



The Core

Process vs Threads?

One Process: `./cook_ramen_together`

```
cook_ramen_microwave();

cook_ramen_microwave();

cook_ramen_microwave();
```

The Core

Process vs Threads?

One Process: **`./cook_ramen_together`**

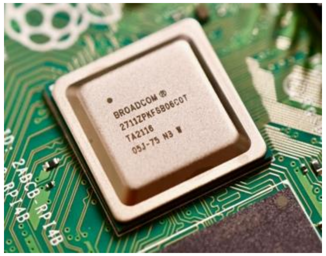
```
for(int i = 0; i < 3; i++){
   cook_ramen_microwave();
}
```

The Core

Process vs Threads?

One Process: **./cook_ramen_together**

```
for(int i = 0; i < 3; i++){
    cook_ramen_microwave();
}
```

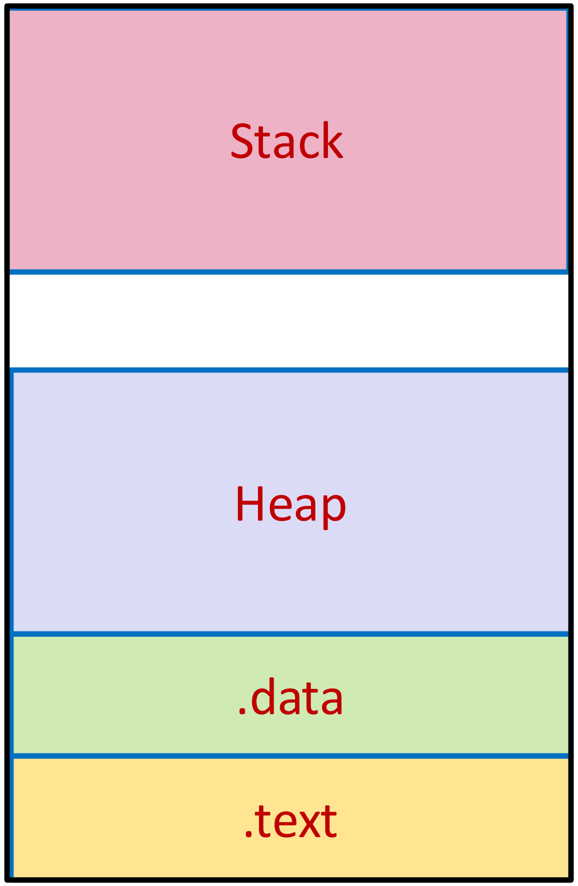


The Core

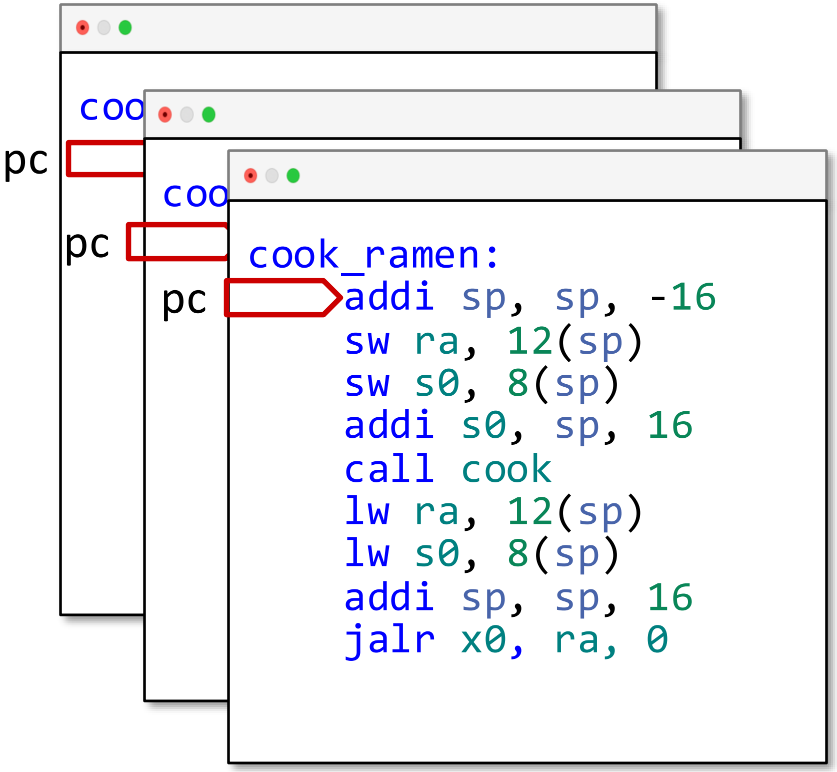
We all want to cook the ramen –
We just each need to run ***our own cook_ramen_mico()*** function.

Process vs Threads?

One Process: `./cook_ramen_together`



Three threads

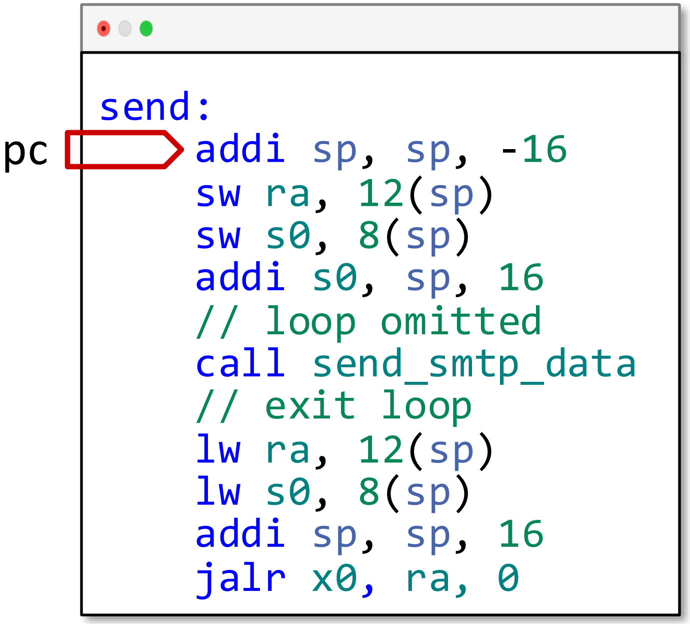


```
cook_ramen:
     addi sp, sp, -16
     sw ra, 12(sp)
     sw s0, 8(sp)
     addi s0, sp, 16
     call cook
     lw ra, 12(sp)
     lw s0, 8(sp)
     addi sp, sp, 16
     jalr x0, ra, 0
```

Multi-Threaded Process

Thread One

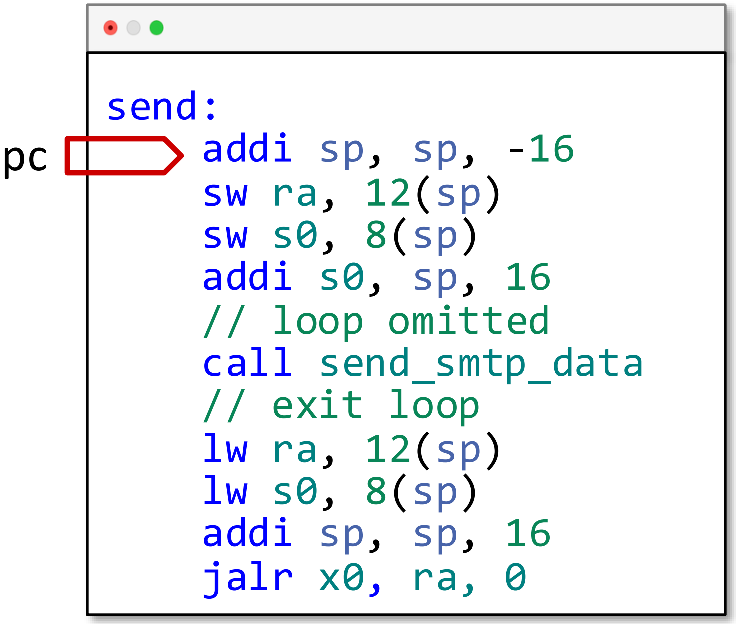
Instructions



```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

Thread Two

Instructions



```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

Wait, ***two program counters in the same process***? Yup! (Don’t worry about how this is possible)

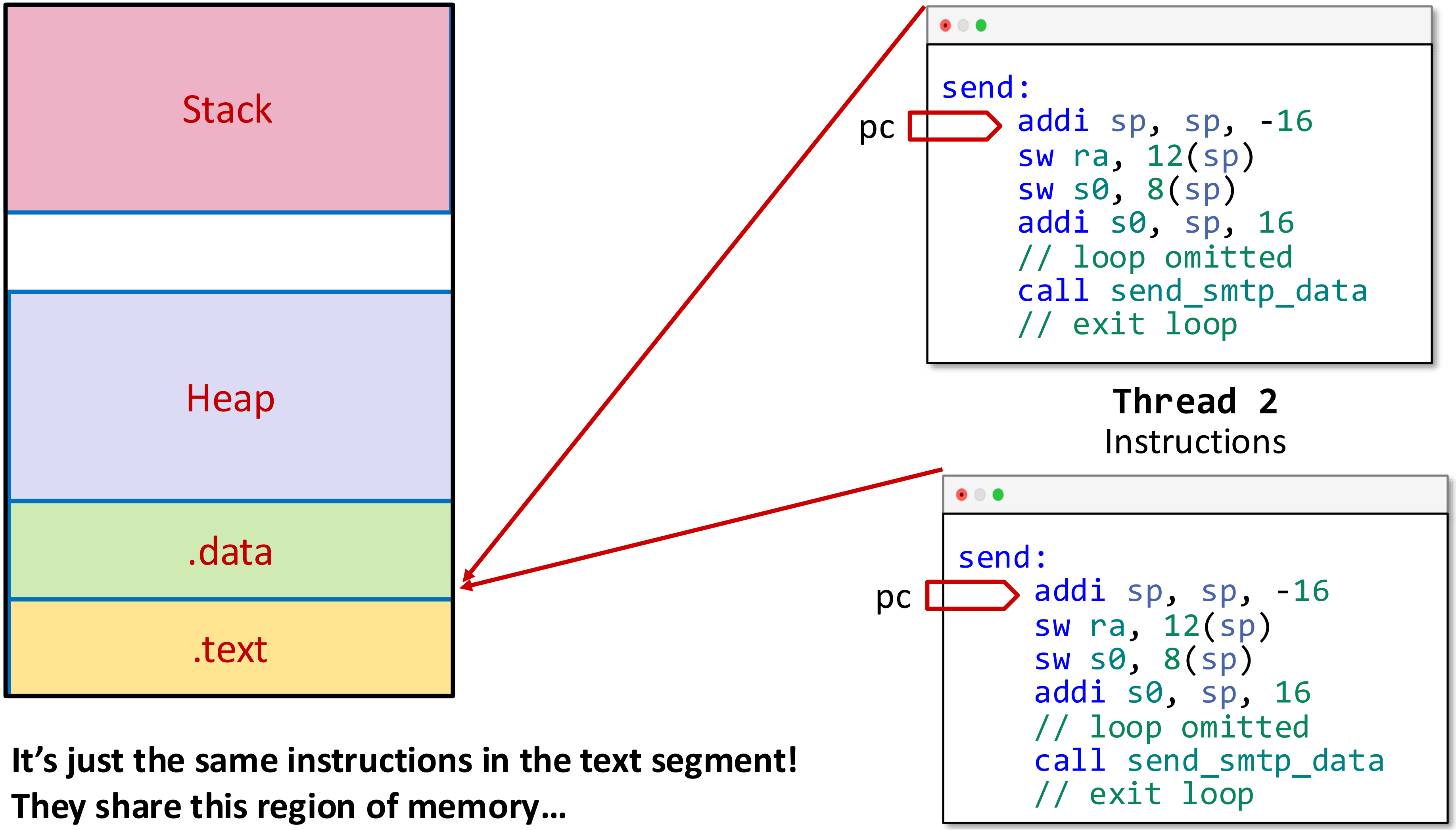
Wait, ***two copies of the same instructions***? No! (They share this region…)

*important to know that these threads are running in the same *process*

Multi-Threaded Program

Memory Layout (State)

Thread Two
Instructions



```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
```

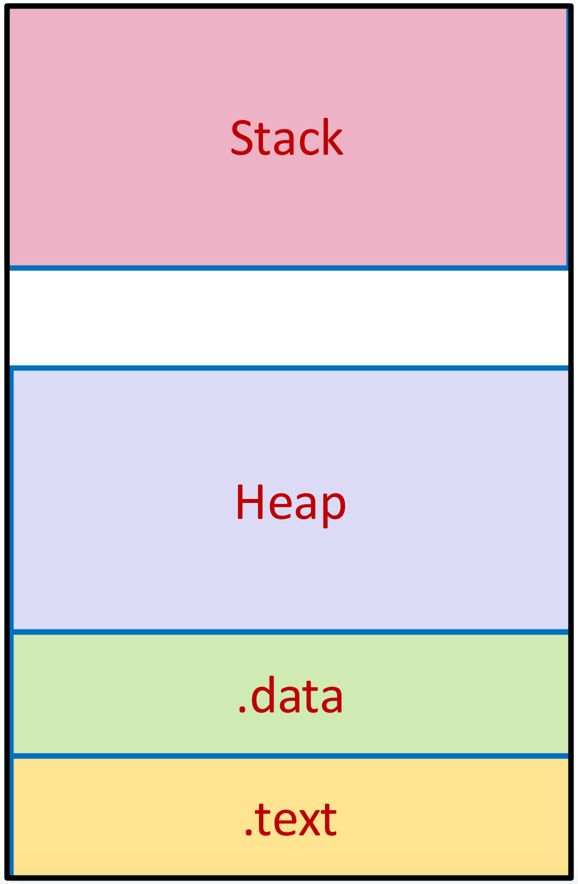
Thread 2
Instructions

```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
```

It’s just the same instructions in the text segment!
They share this region of memory…

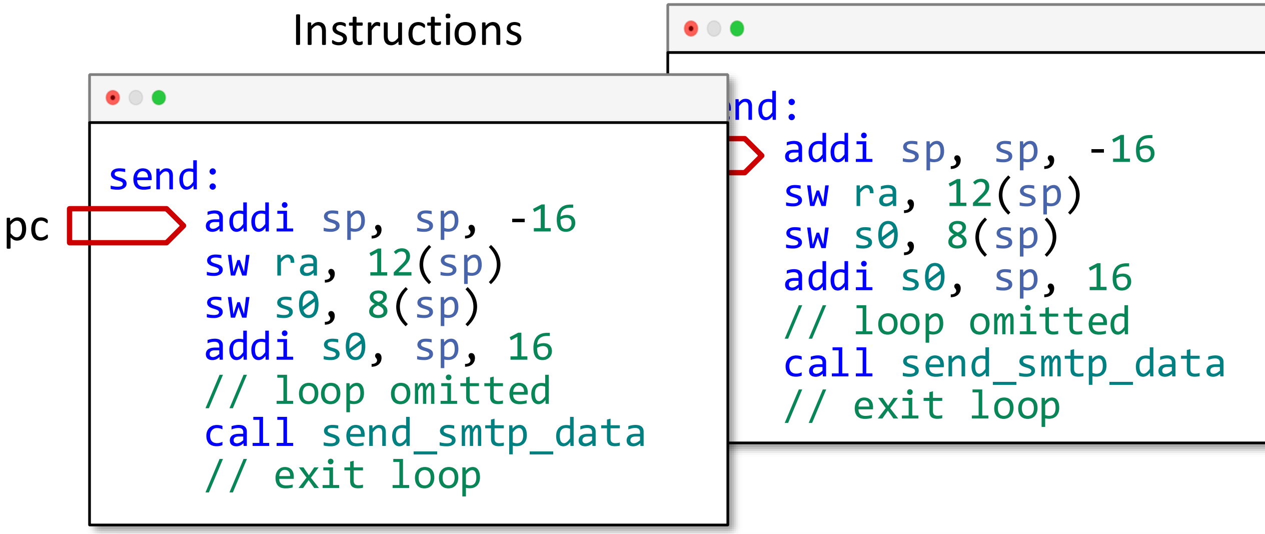
Multi-Threaded Program

Memory Layout (State)



Thread Two
Instructions

Thread 2
Instructions



```
send:
     addi sp, sp, -16
     sw ra, 12(sp)
     sw s0, 8(sp)
     addi s0, sp, 16
     // loop omitted
     call send_smtp_data
     // exit loop
```

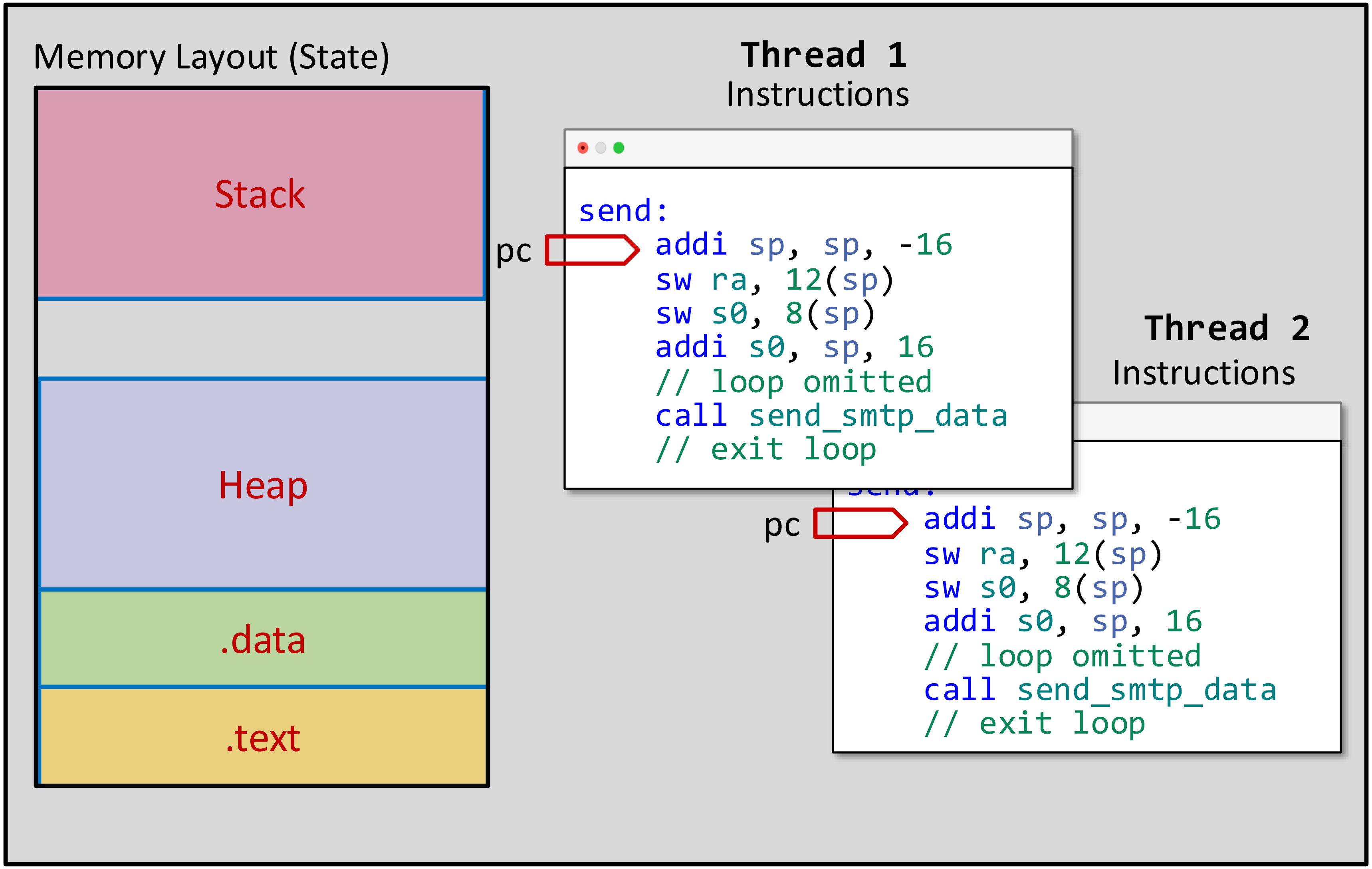
```
nd:
  addi sp, sp, -16
  sw ra, 12(sp)
  sw s0, 8(sp)
  addi s0, sp, 16
  // loop omitted
  call send_smtp_data
  // exit loop
```

These two threads also share the processor!

These two threads share the entire memory space!

The stack, the heap, data (global vars), and the text!

One Process with Two Threads



```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
```

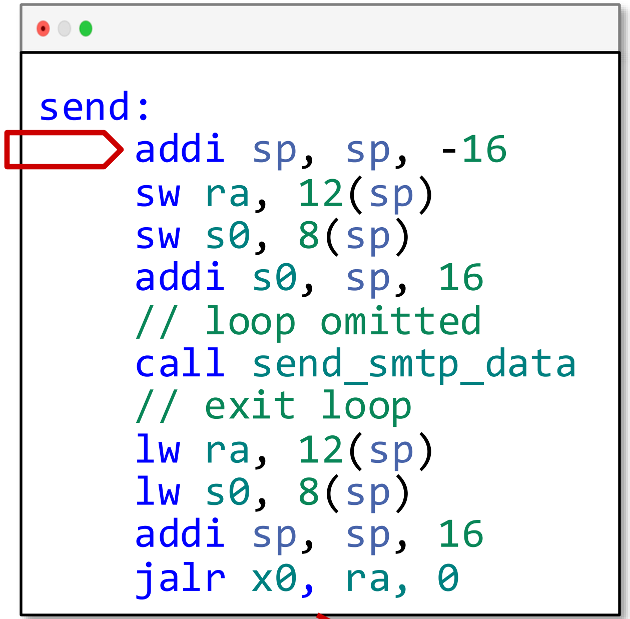
```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
```

Concurrency: Processor Sharing

Thread One

Instructions

pc

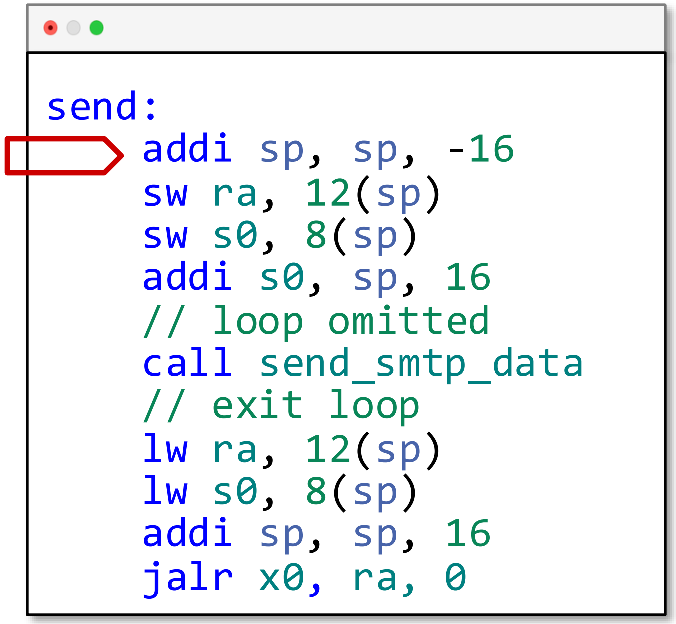


```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

Thread Two

Instructions

pc



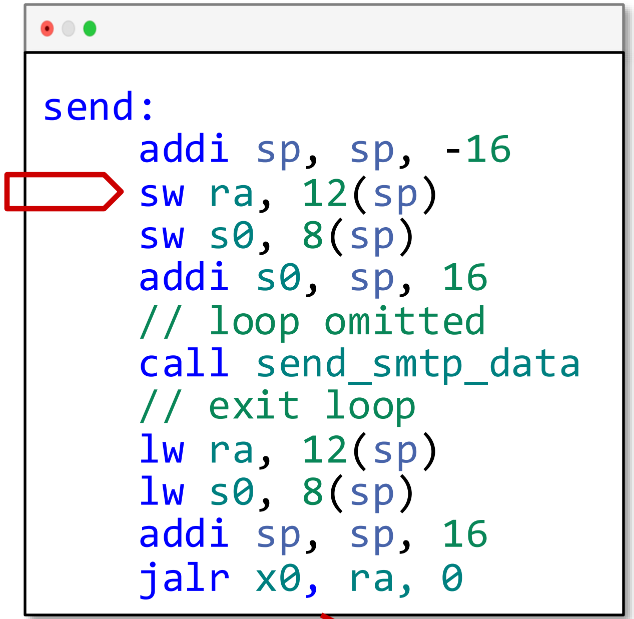
```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

This thread is waiting for CPU…

Concurrency: Processor Sharing

Thread One

Instructions

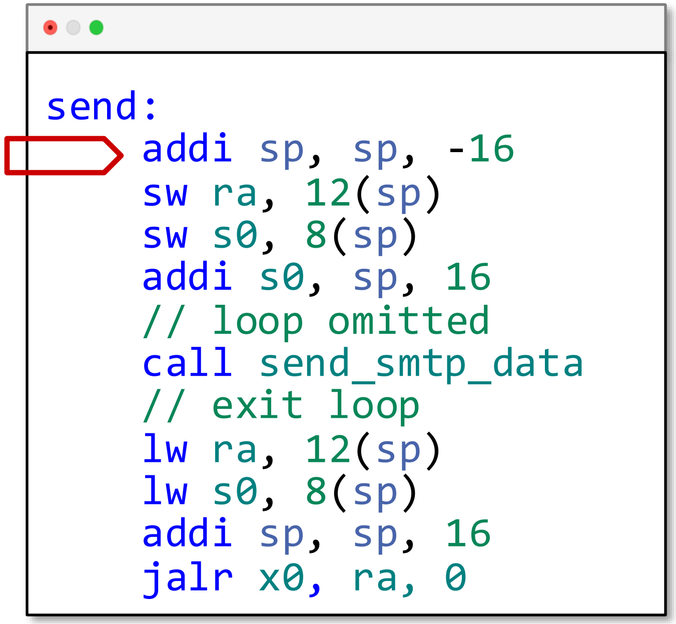


pc

```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

Thread Two

Instructions



pc

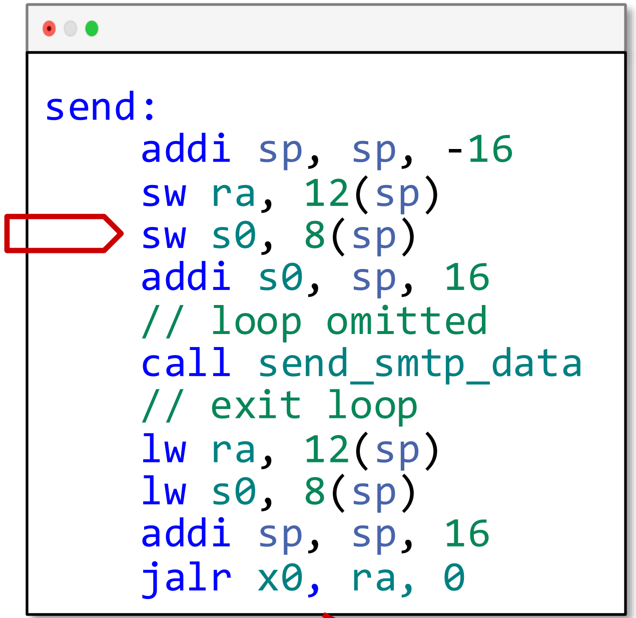
```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

This thread is waiting for CPU…

Concurrency: Processor Sharing

Thread One

Instructions

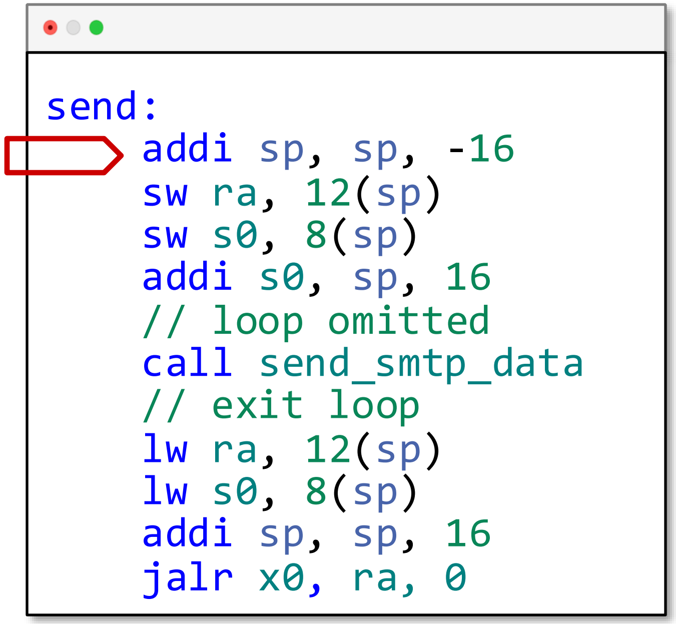


pc

```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

Thread Two

Instructions



pc

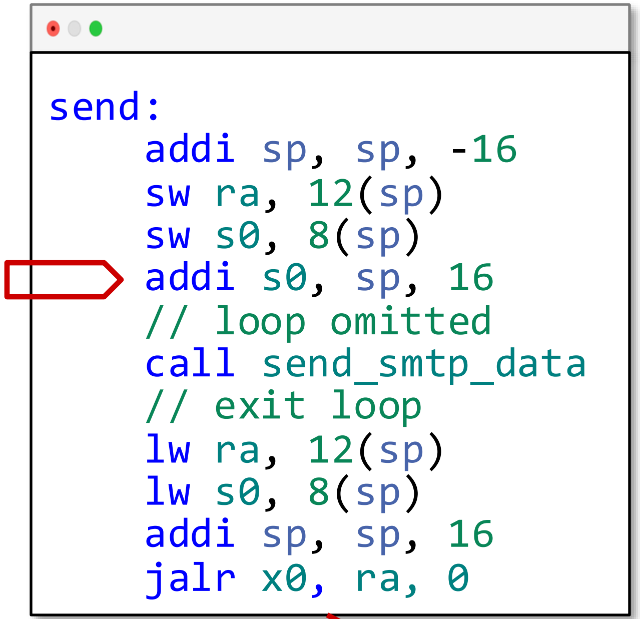
```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

This thread is waiting for CPU…

Concurrency: Processor Sharing

Thread One

Instructions

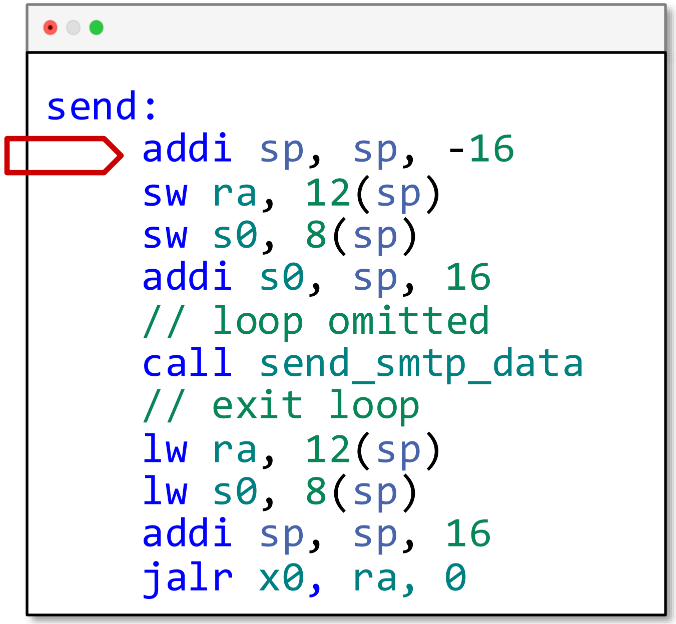


pc

```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

Thread Two

Instructions



pc

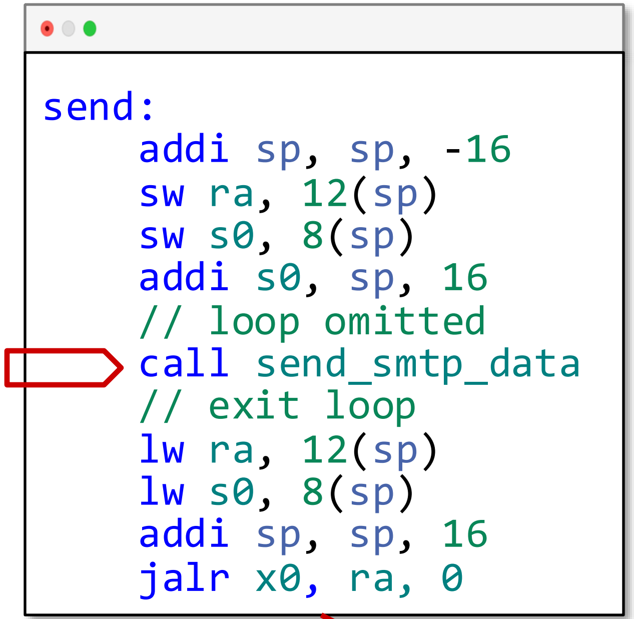
```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

This thread is waiting for CPU…

Concurrency: Processor Sharing

Thread One

Instructions

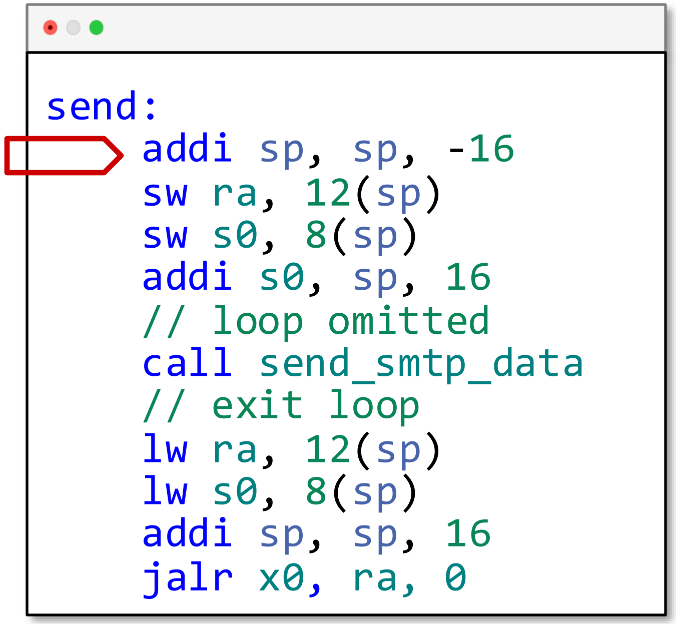


pc

```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

Thread Two

Instructions



pc

```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

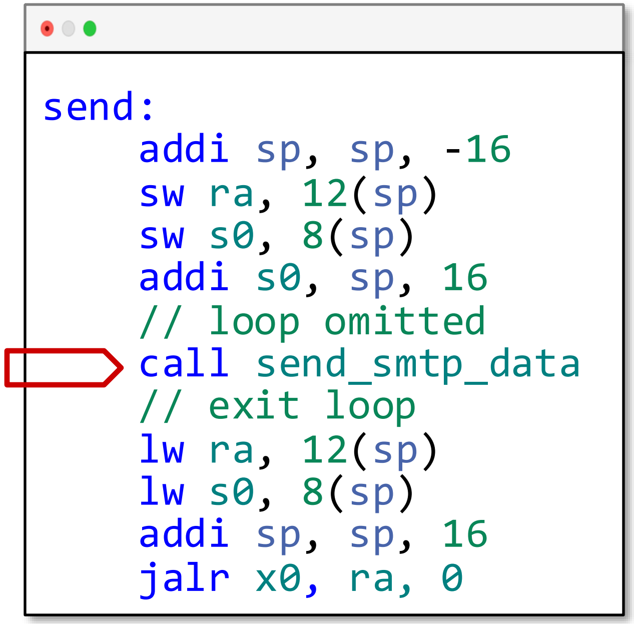
This thread is waiting for CPU…

Concurrency: Processor Sharing

Thread One

Instructions

pc

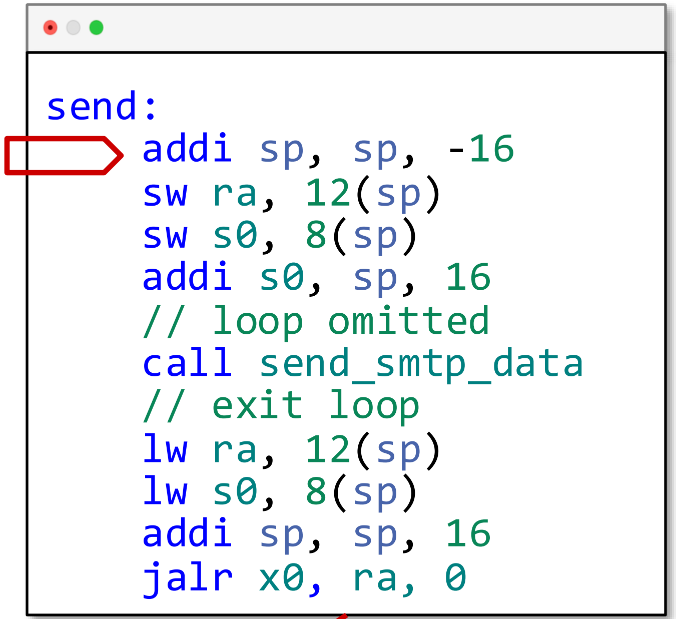


```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

Thread Two

Instructions

pc



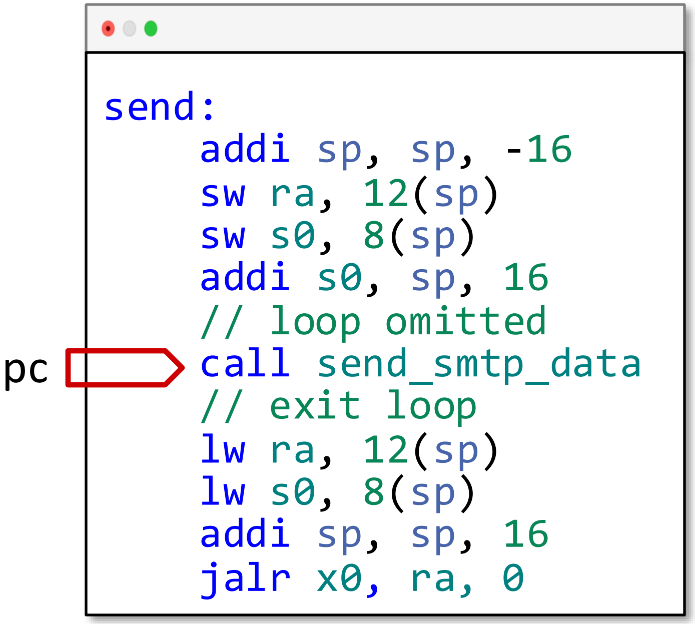
```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

This thread is waiting for I/O.
Let’s switch to the other thread…

Concurrency: Processor Sharing

Thread One

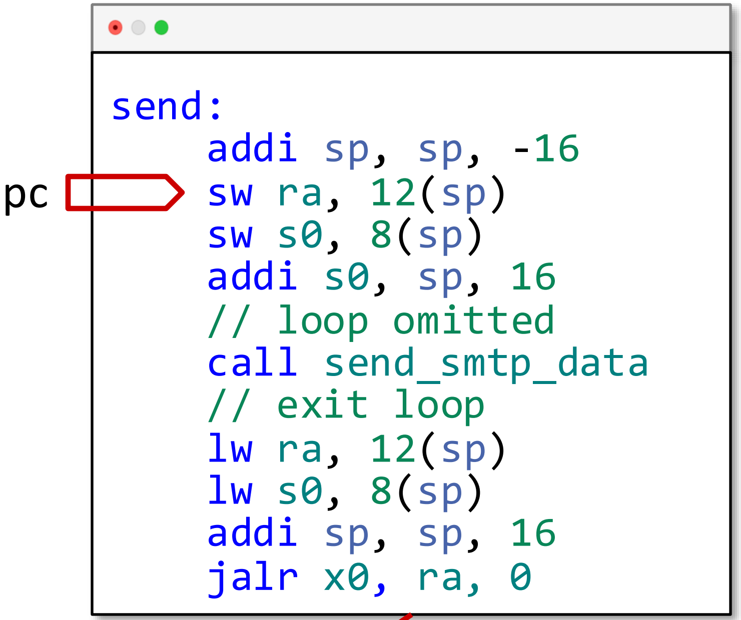
Instructions



```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

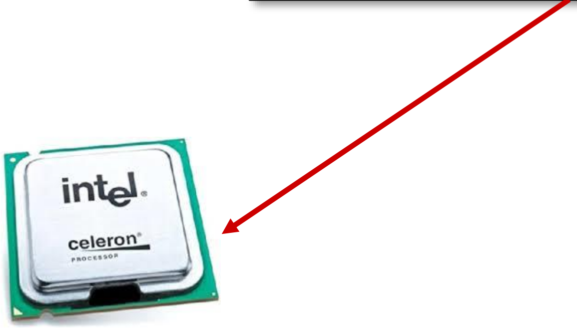
Thread Two

Instructions



```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

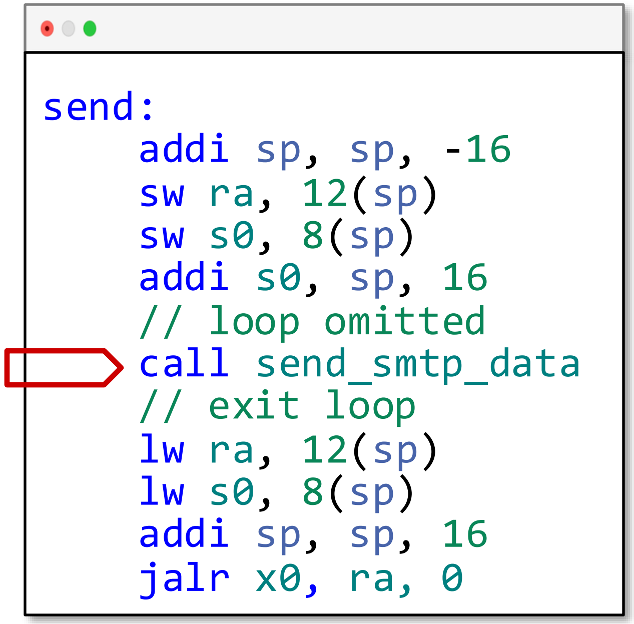
This thread is waiting for I/O.
Let's switch to the other thread...



Concurrency: Processor Sharing

Thread One

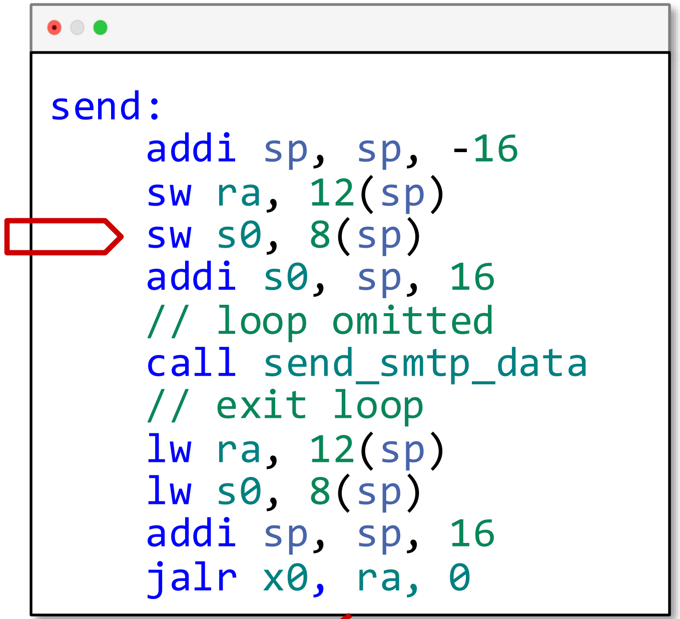
Instructions



```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

Thread Two

Instructions



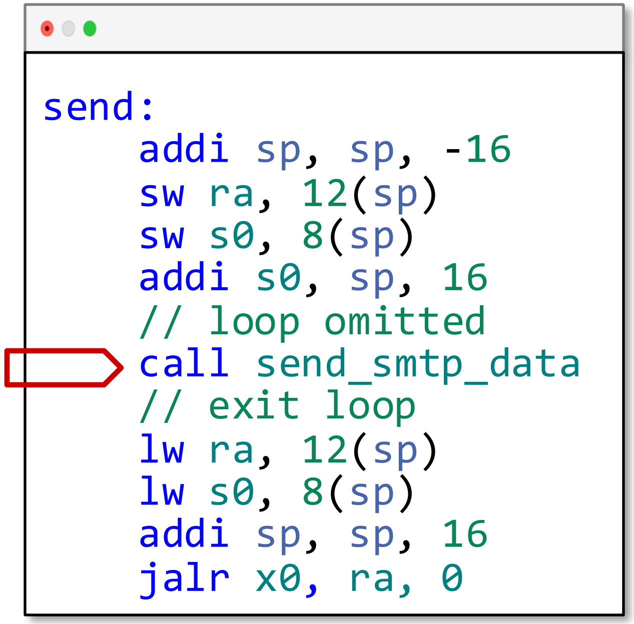
```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

This thread is waiting for I/O.
Let’s switch to the other thread…

Concurrency: Processor Sharing

Thread One

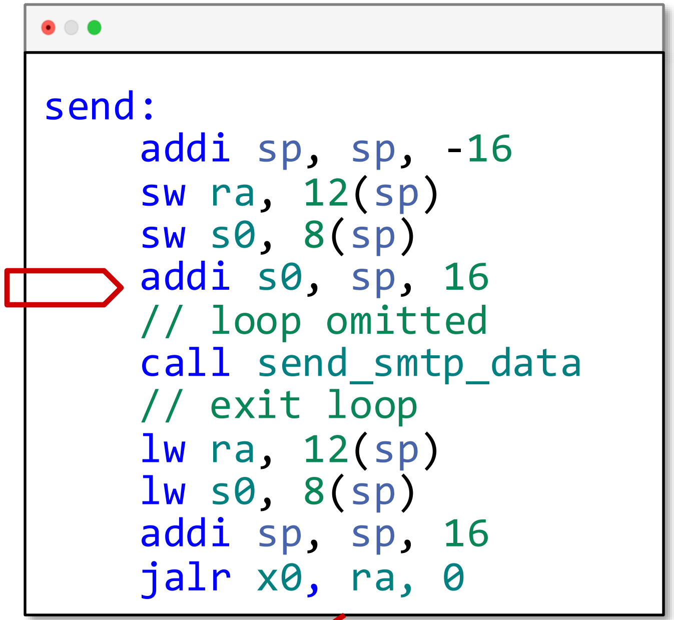
Instructions



```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

Thread Two

Instructions



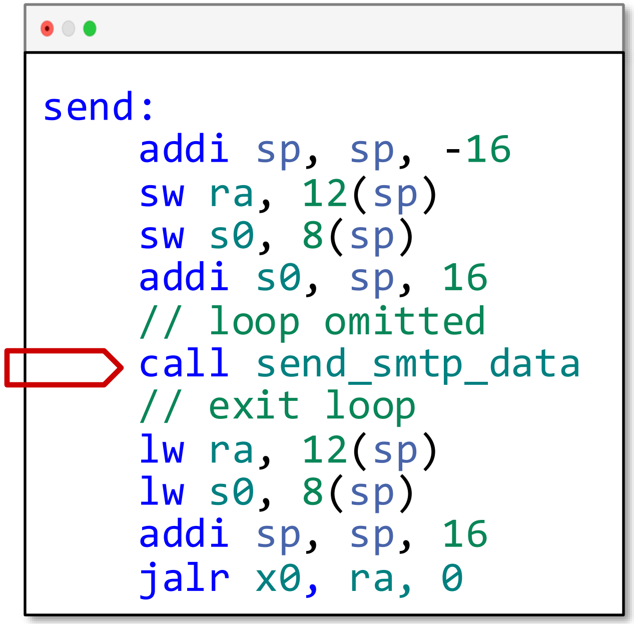
```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

This thread is waiting for I/O.
Let’s switch to the other thread…

Concurrency: Processor Sharing

Thread One

Instructions

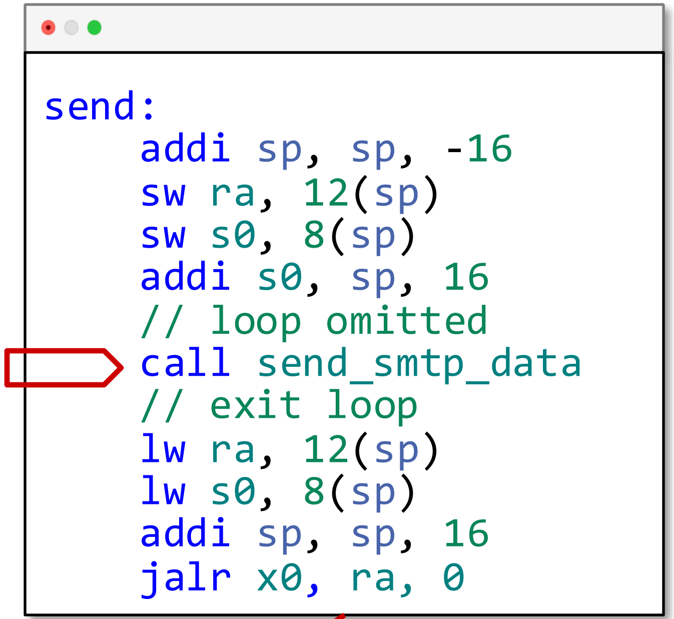


```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

pc

Thread Two

Instructions



```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

pc

This thread is ***waiting for CPU now..***

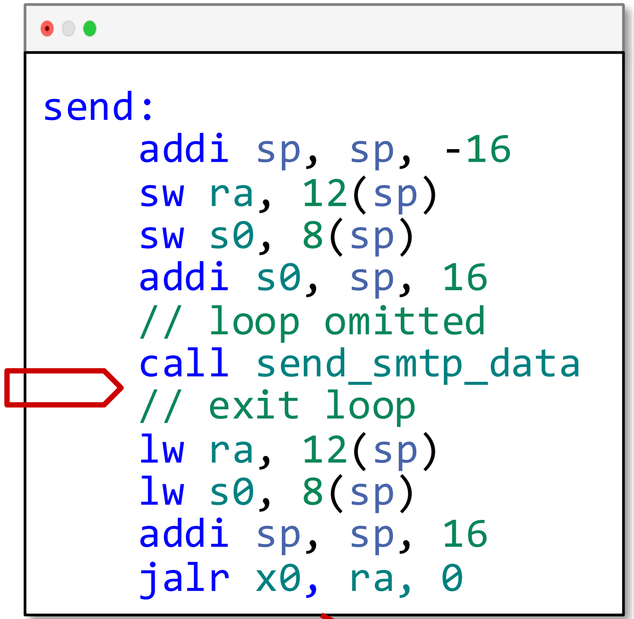
This thread is waiting for I/O.

Let's switch to **thread 1** and see if the I/O is done

Concurrency: Processor Sharing

Thread One

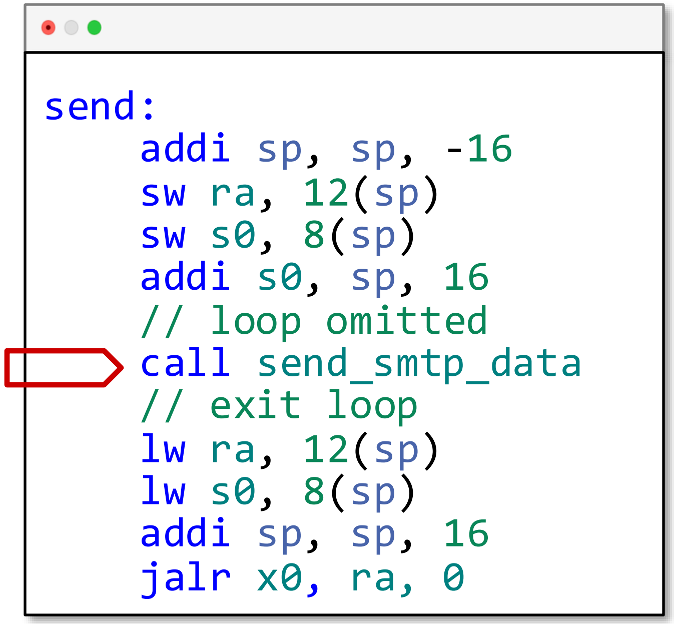
Instructions



```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

Thread Two

Instructions



```
send:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    addi s0, sp, 16
    // loop omitted
    call send_smtp_data
    // exit loop
    lw ra, 12(sp)
    lw s0, 8(sp)
    addi sp, sp, 16
    jalr x0, ra, 0
```

This thread is waiting for I/O.

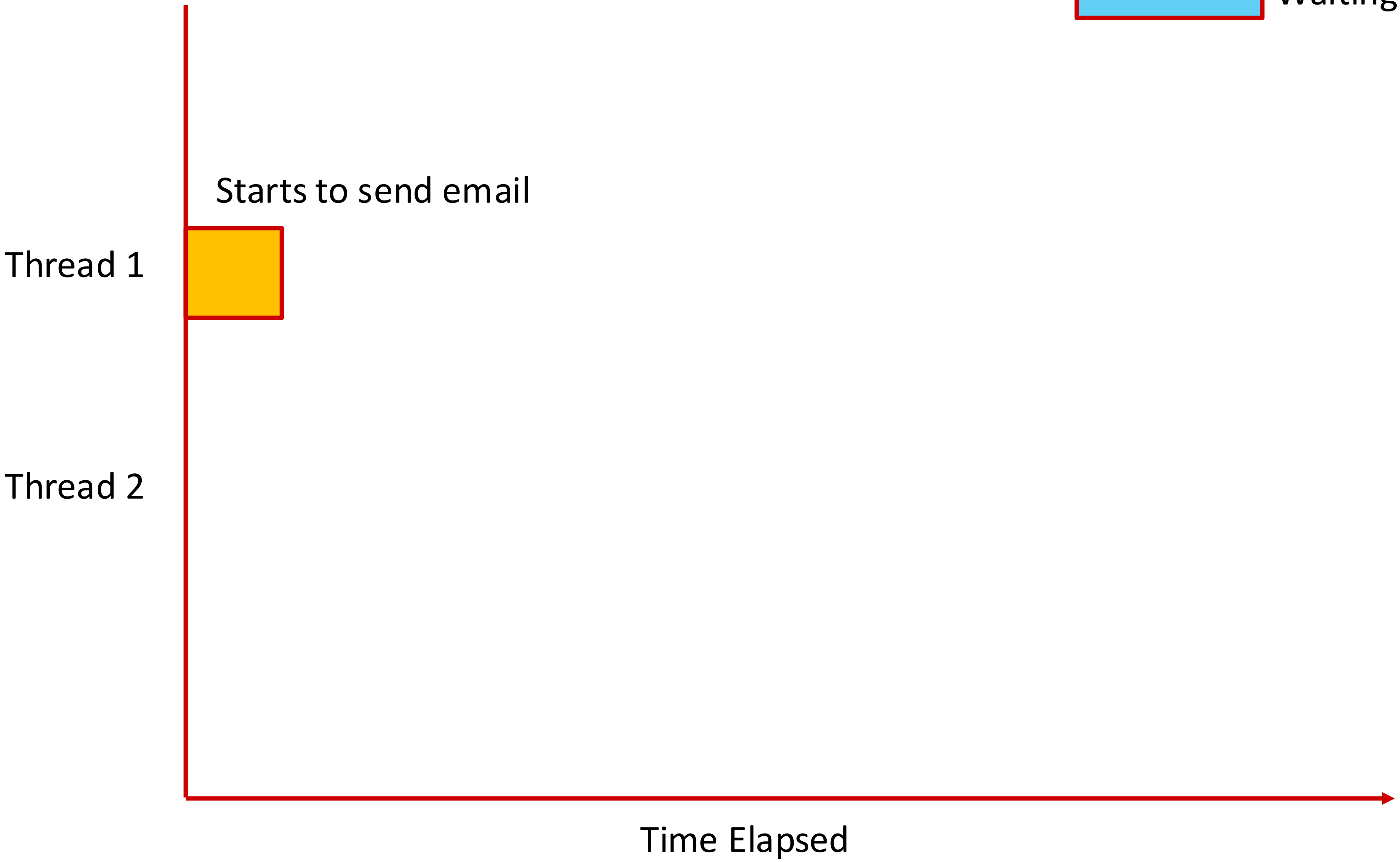
Now this thread continues! It does the set up to send another email!

*you might notice this looks familiar

Now With Two Threads

🟨 Using CPU

🟦 Waiting for I/O



Starts to send email

Thread 1

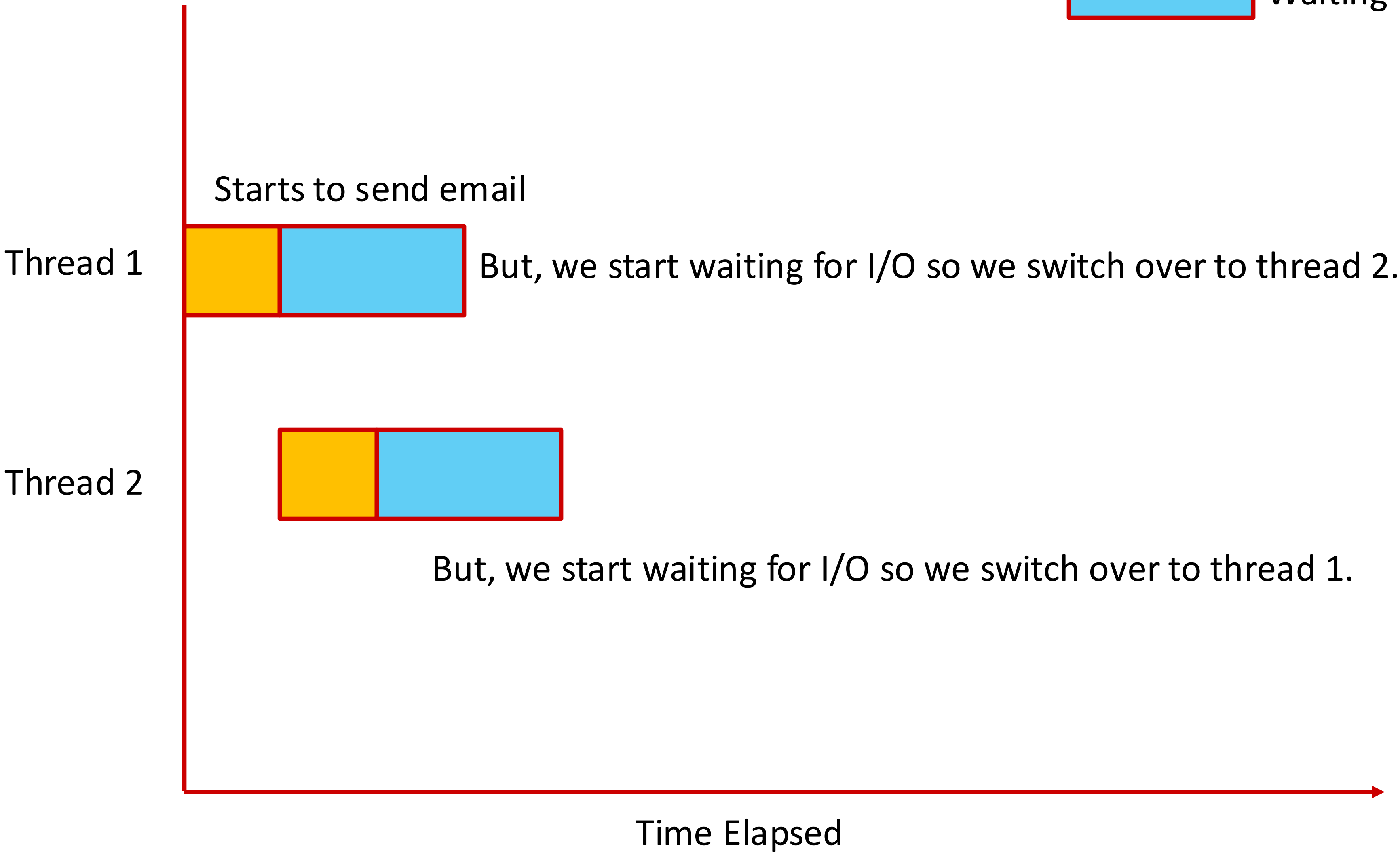
Thread 2

Time Elapsed

Now With Two Threads

🟧 Using CPU

🟦 Waiting for I/O

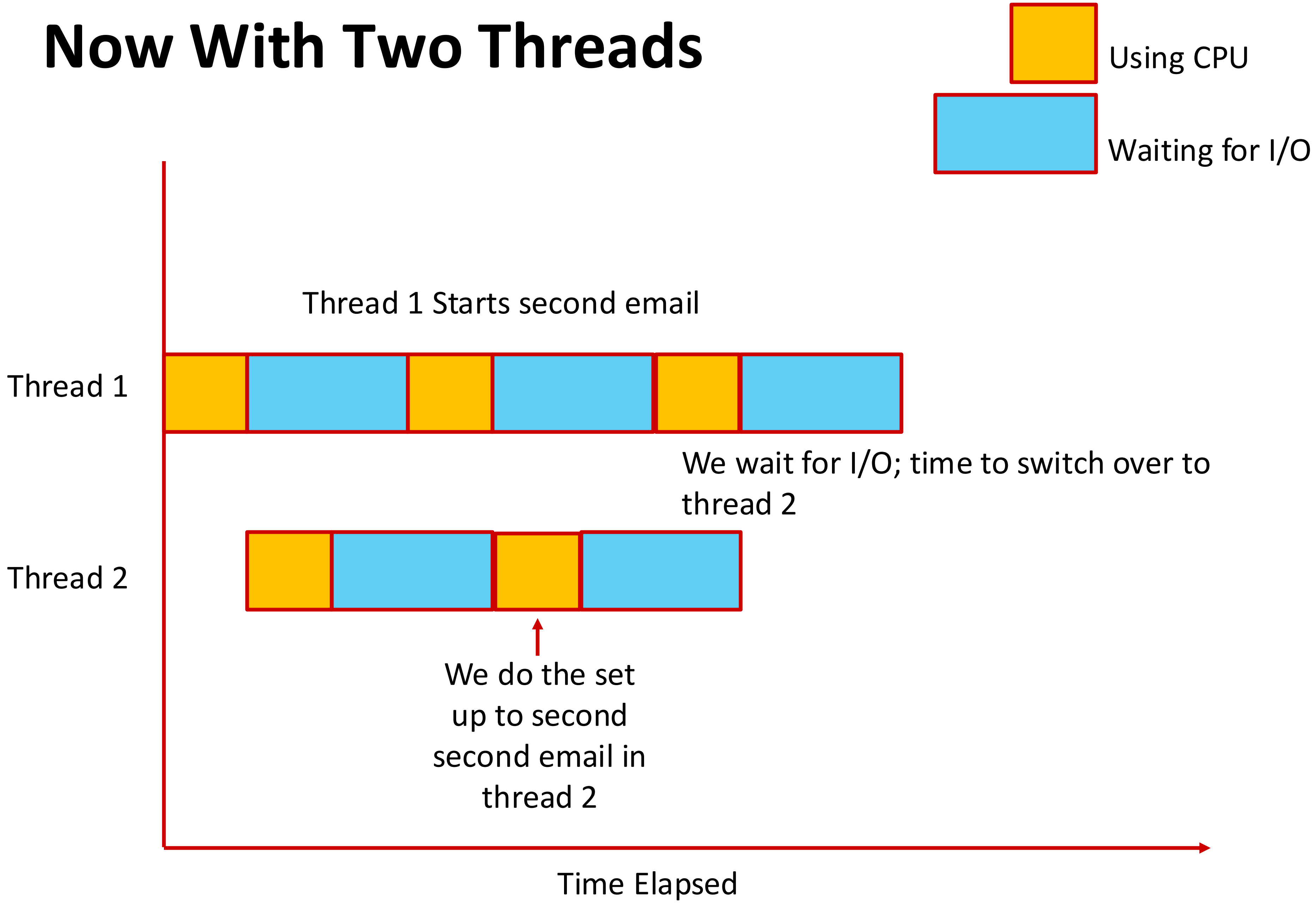


Starts to send email

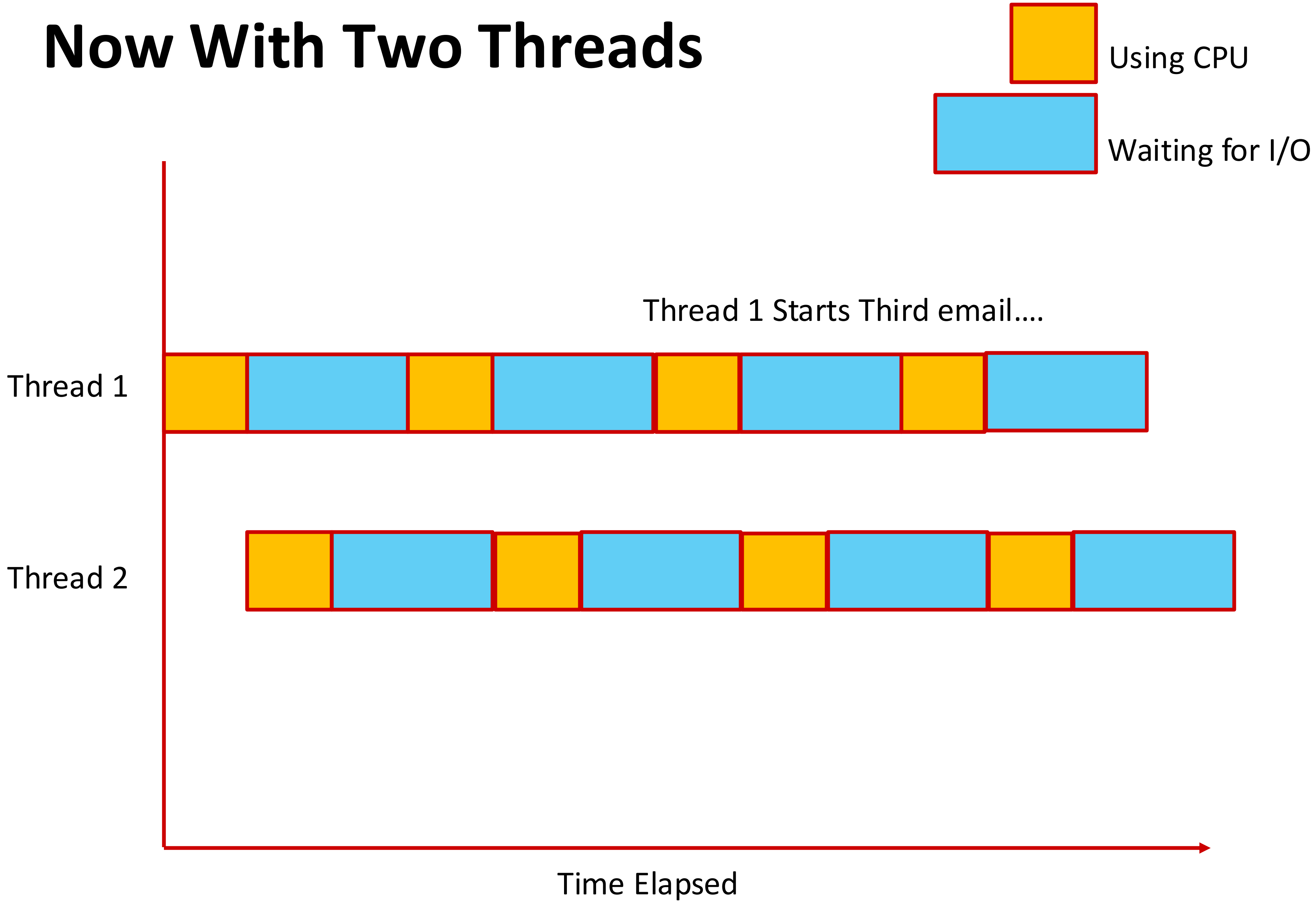
But, we start waiting for I/O so we switch over to thread 2.

But, we start waiting for I/O so we switch over to thread 1.

Now With Two Threads

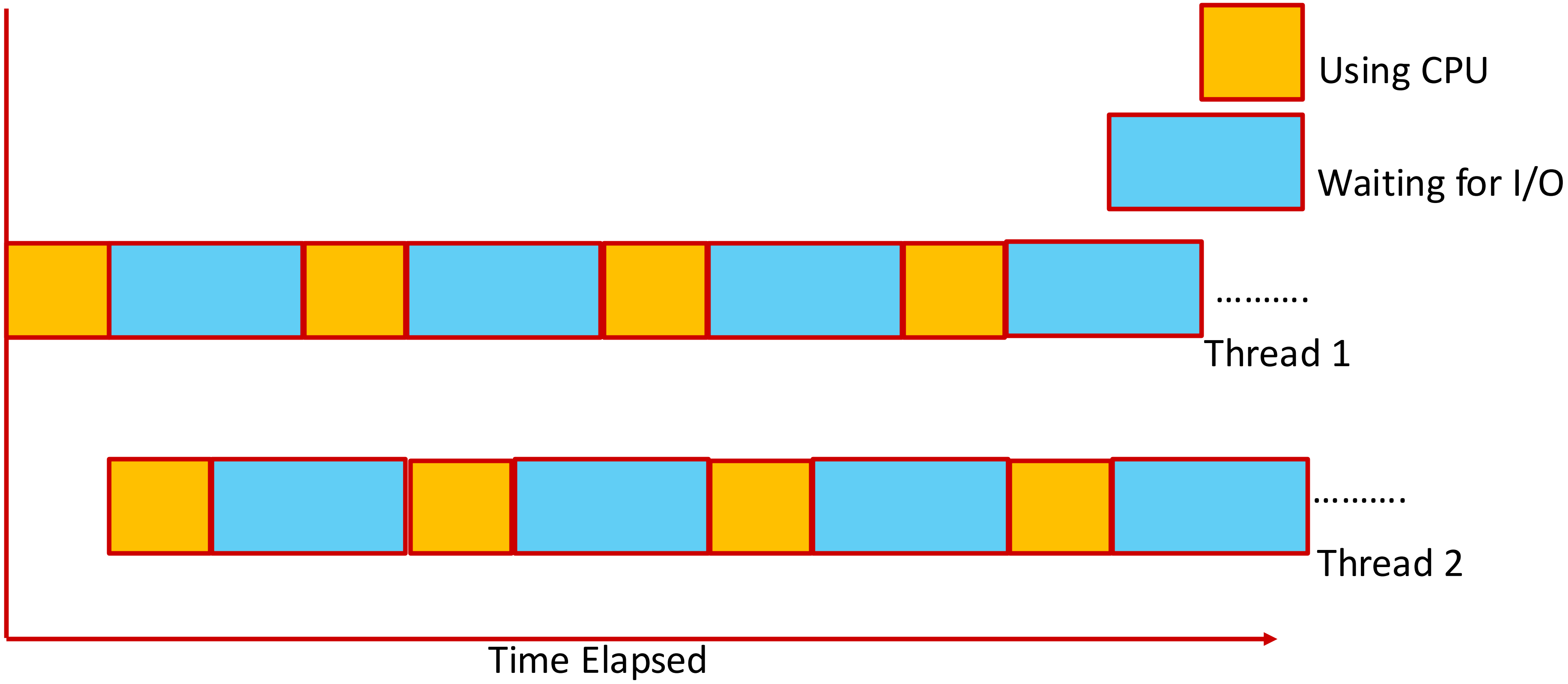

Using CPU
Waiting for I/O
Thread 1 Starts second email
Thread 1
We wait for I/O; time to switch over to thread 2
Thread 2
We do the set up to second second email in thread 2
Time Elapsed

Now With Two Threads



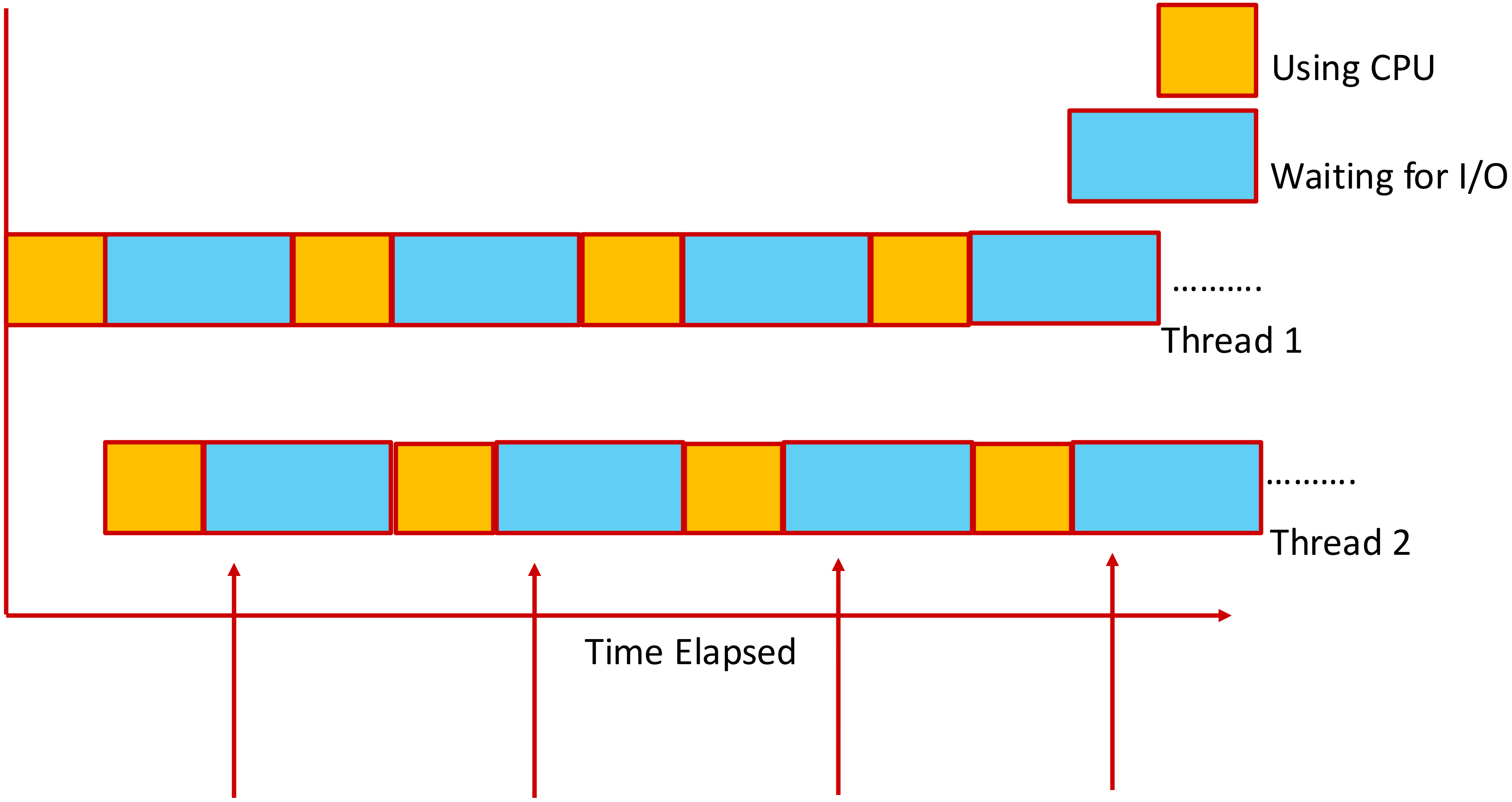
Poll Everywhere pollev.com/cis2400

Is the CPU always utilized? Discuss!



Poll Everywhere pollev.com/cis2400

Is the CPU always utilized? No! :/

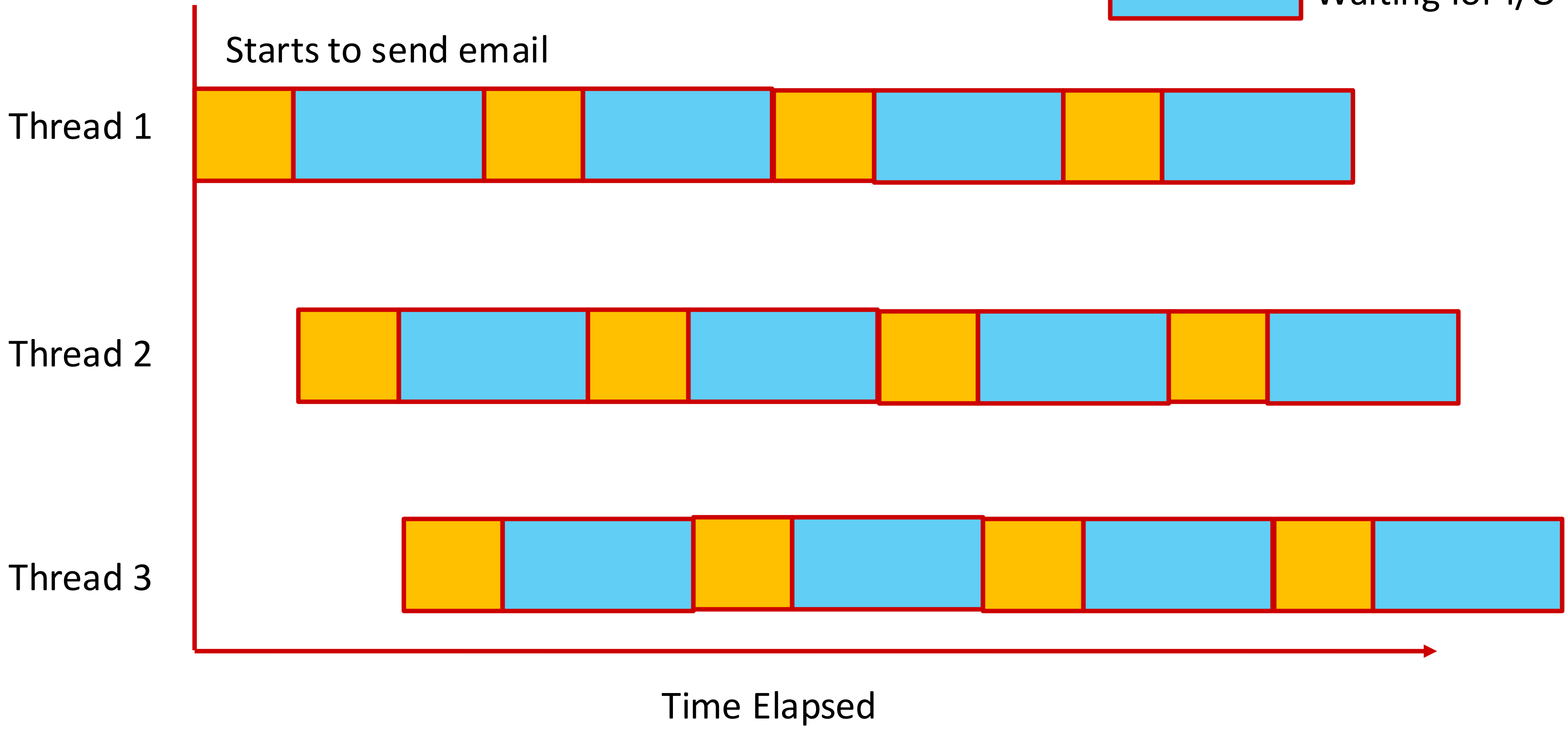


Here, there is only waiting for I/O until there's more work to do.

How can we do better? 3 Threads!

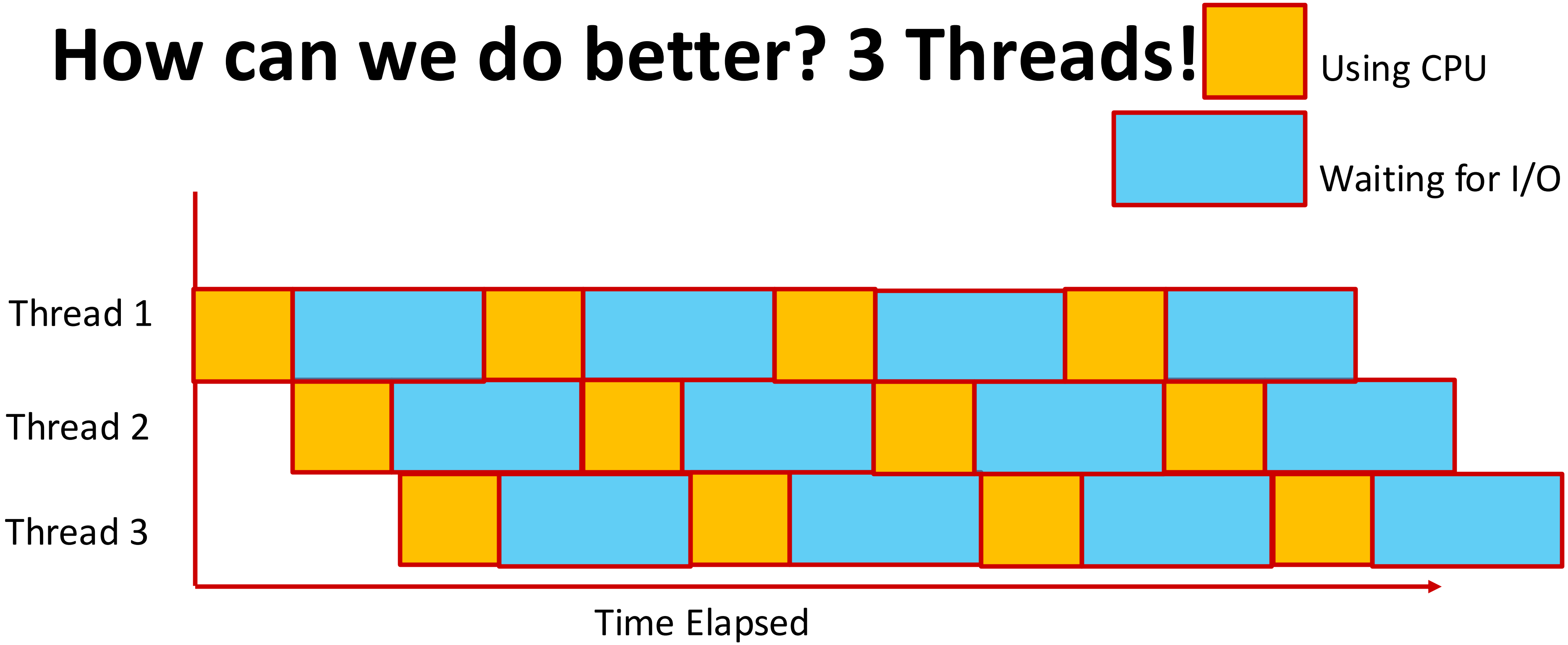
🟨 Using CPU

🟦 Waiting for I/O



Here, the CPU or core is consistently active, with no idle time spent waiting for additional tasks to process.

How can we do better? 3 Threads!



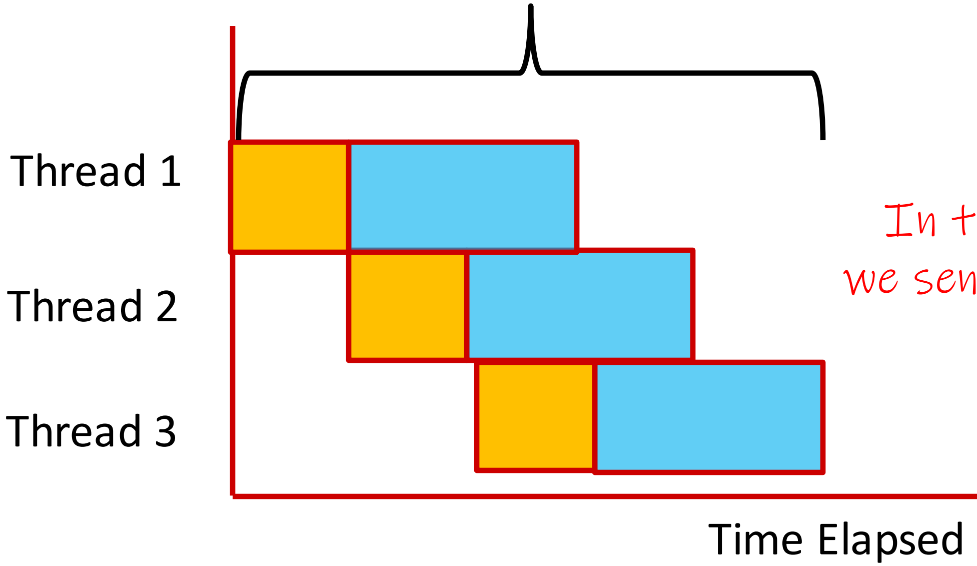
Here, the CPU or core is **consistently active**, with **no idle time** spent waiting for additional tasks to process.

As soon as ***one thread starts waiting for I/O*** there's always another waiting for the CPU.

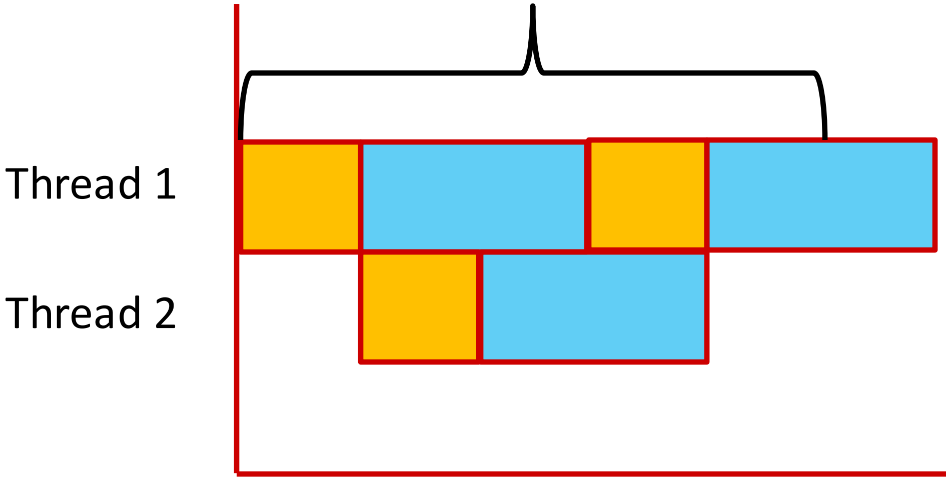
How is this better?

🟨 Using CPU

🟦 Waiting for I/O



In this Interval,
we send three emails!

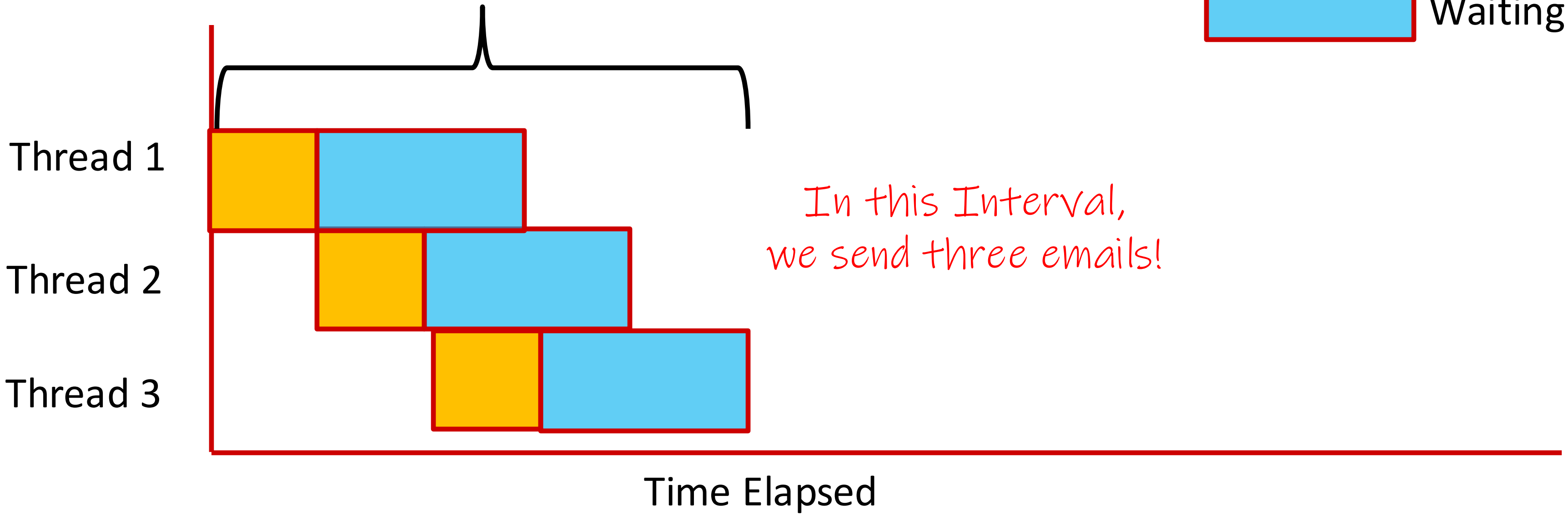


In the same Interval,
we send only two. emails!

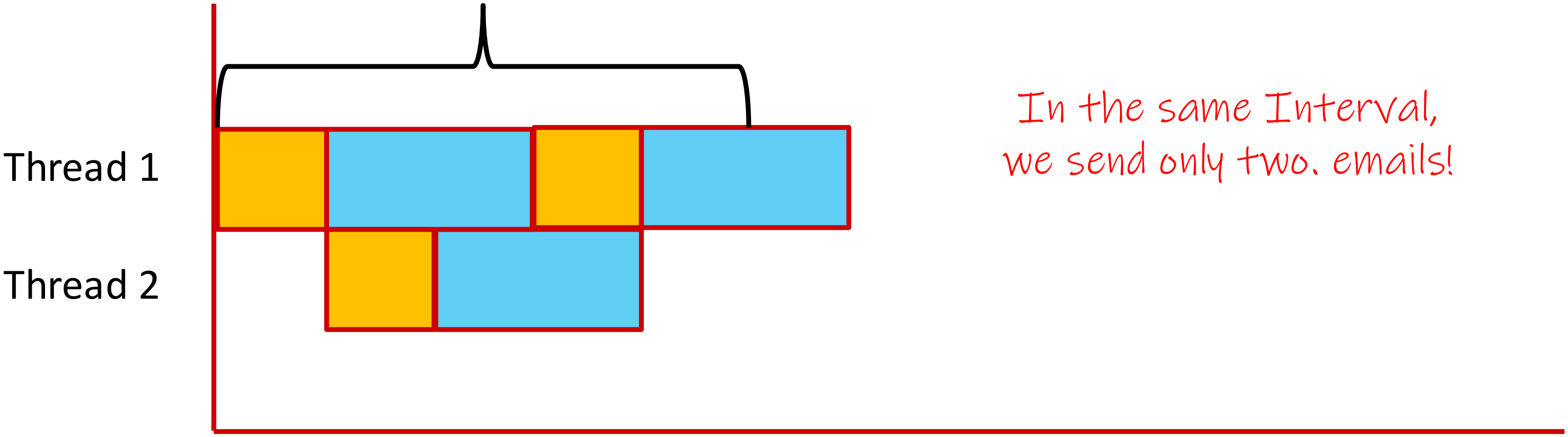
How is this better?

🟨 Using CPU

🟦 Waiting for I/O

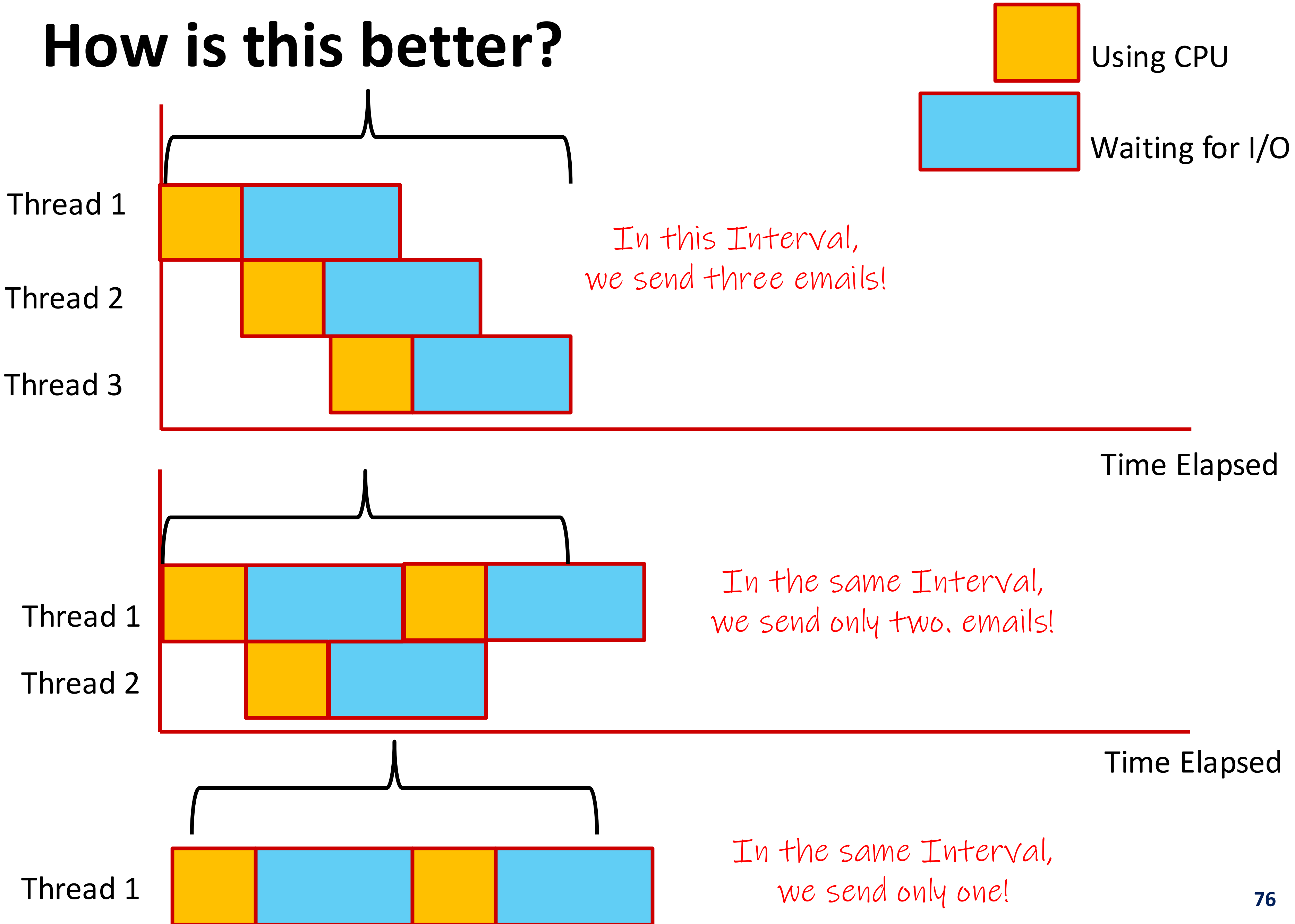


In this Interval,
we send three emails!



In the same Interval,
we send only two. emails!

How is this better?

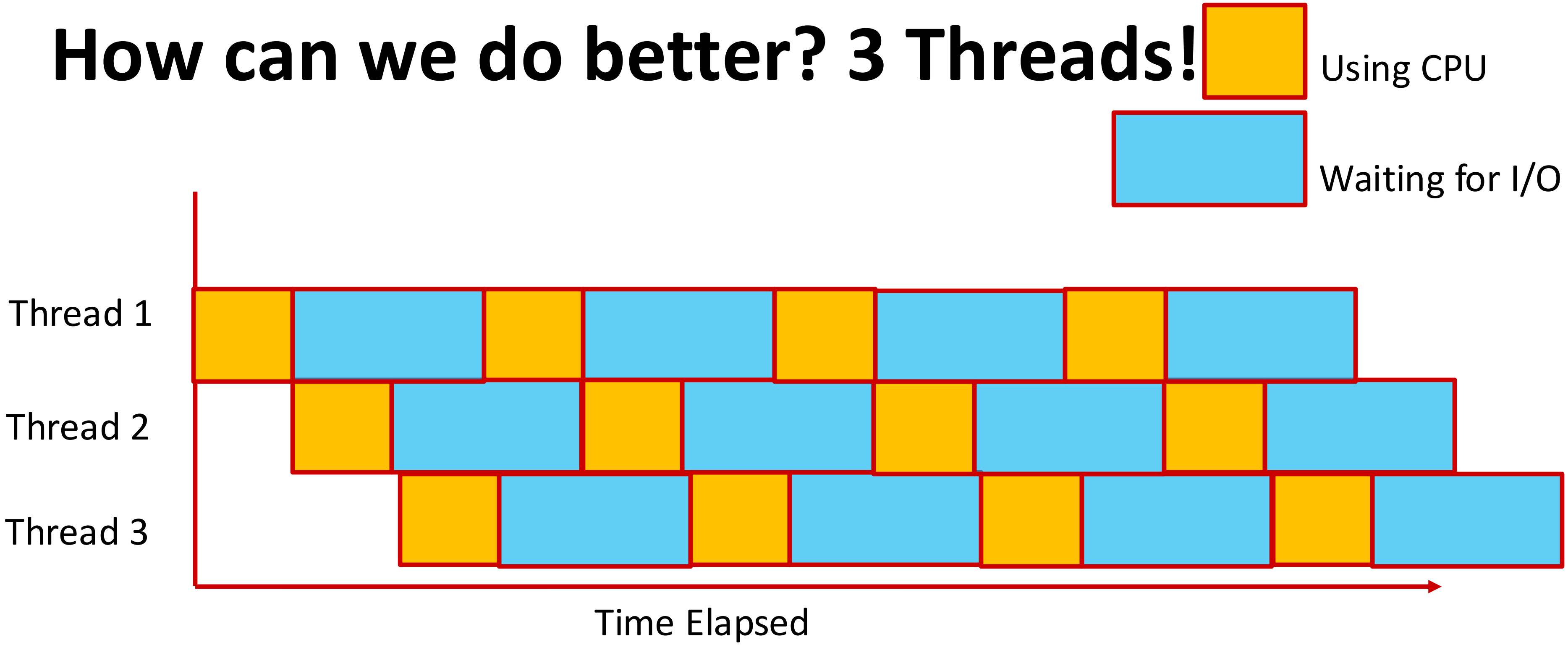


In this Interval,
we send three emails!

In the same Interval,
we send only two. emails!

In the same Interval,
we send only one!

How can we do better? 3 Threads!

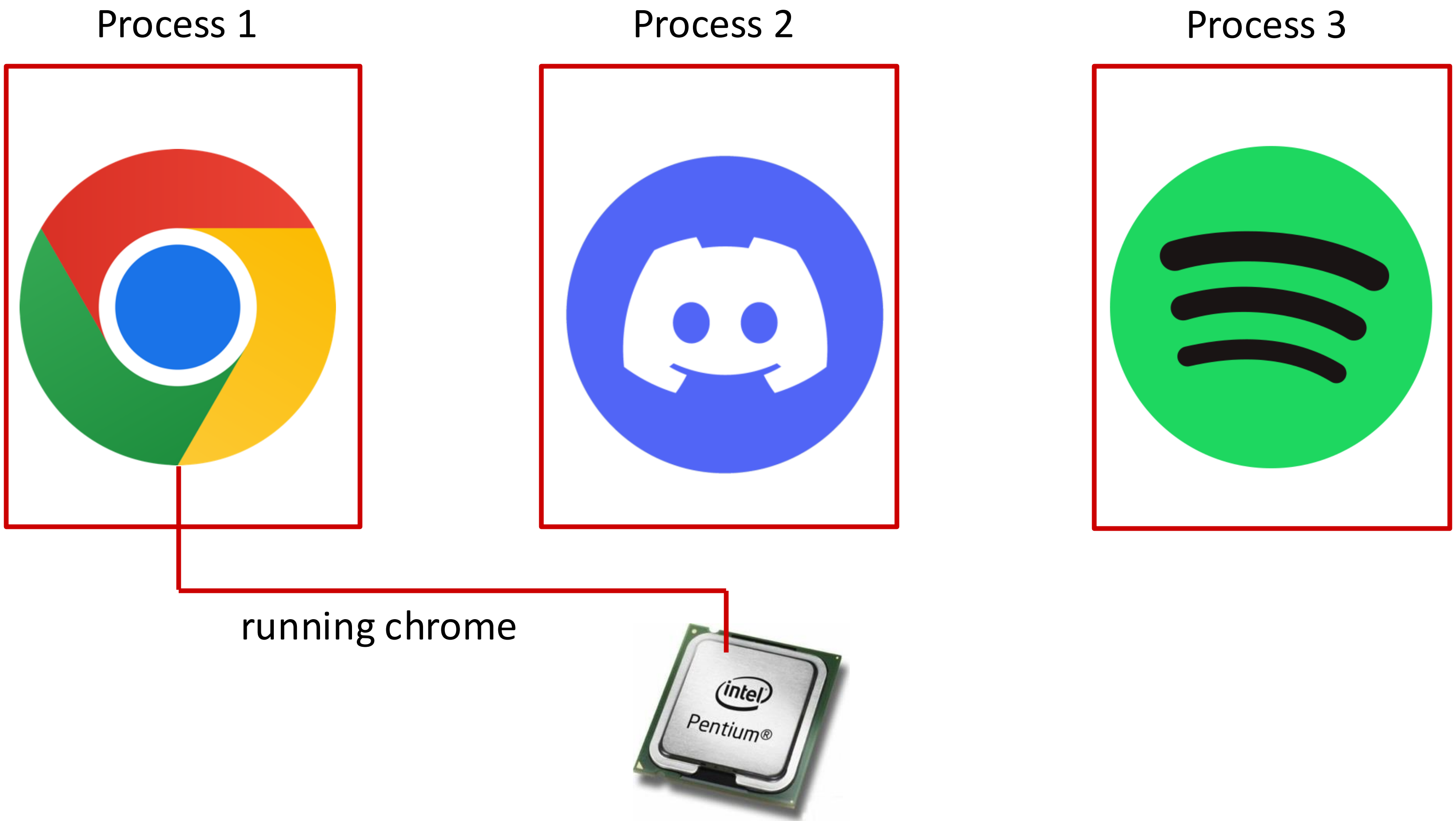


In general, using more threads allows multiple tasks to be handled simultaneously, which means more work gets done in the same amount of time.

*sometimes, adding more threads makes things slower actually.

Processes vs. Threads: What's the Difference?

- Each process has its own memory space—makes sense, right?
- ***Why should Spotify have access to Chrome's memory?***



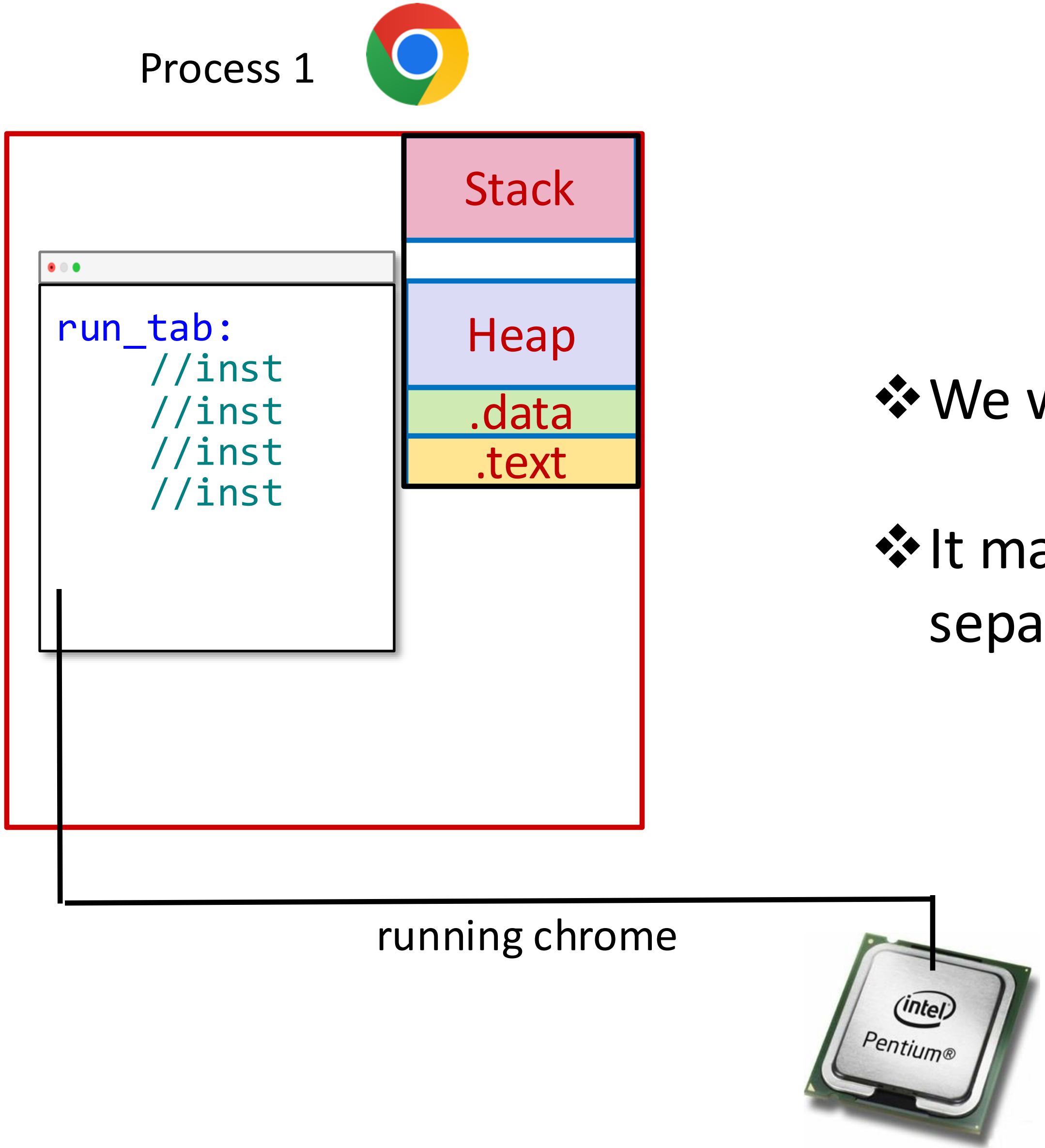
Process 1

Process 2

Process 3

running chrome

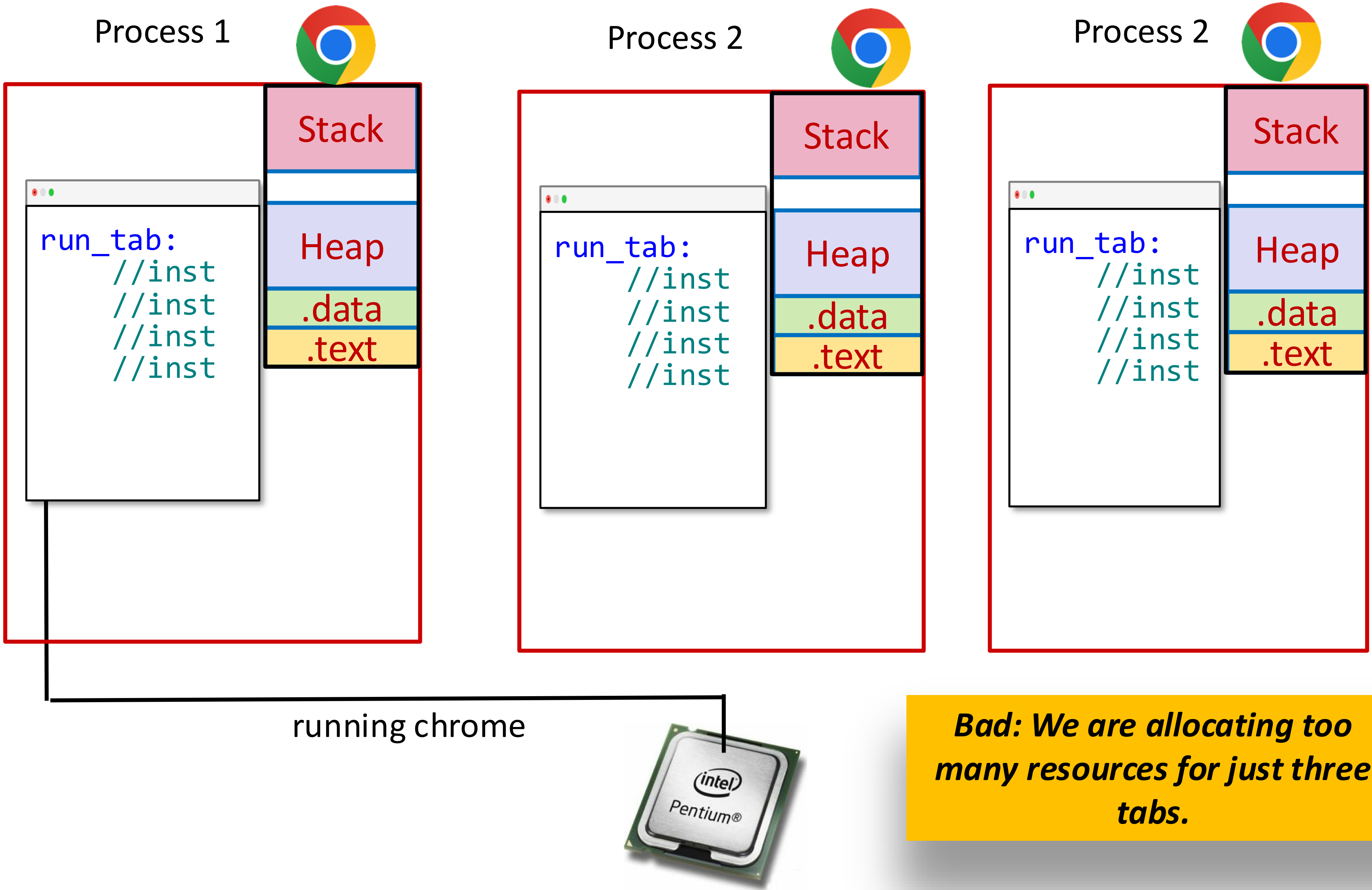
Processes vs. Threads: What's the Difference?



```
run_tab:
    //inst
    //inst
    //inst
    //inst
```

- We want to run three tabs.
- It makes sense to run each tab separately!

Processes vs. Threads: What's the Difference?



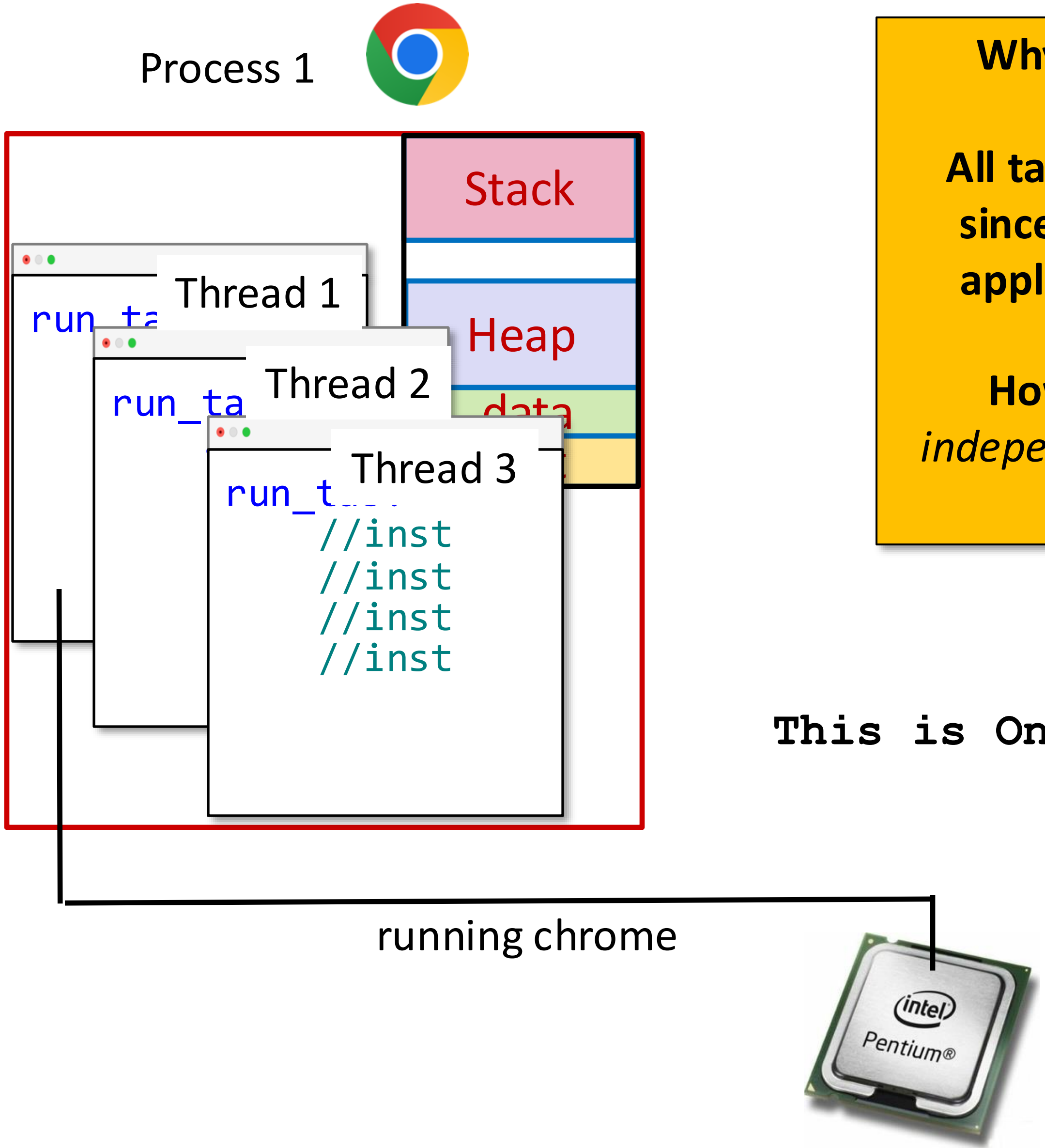
```
run_tab:
      //inst
      //inst
      //inst
      //inst
```

```
run_tab:
      //inst
      //inst
      //inst
      //inst
```

```
run_tab:
      //inst
      //inst
      //inst
      //inst
```

Bad: We are allocating too many resources for just three tabs.

Processes vs. Threads: What's the Difference?



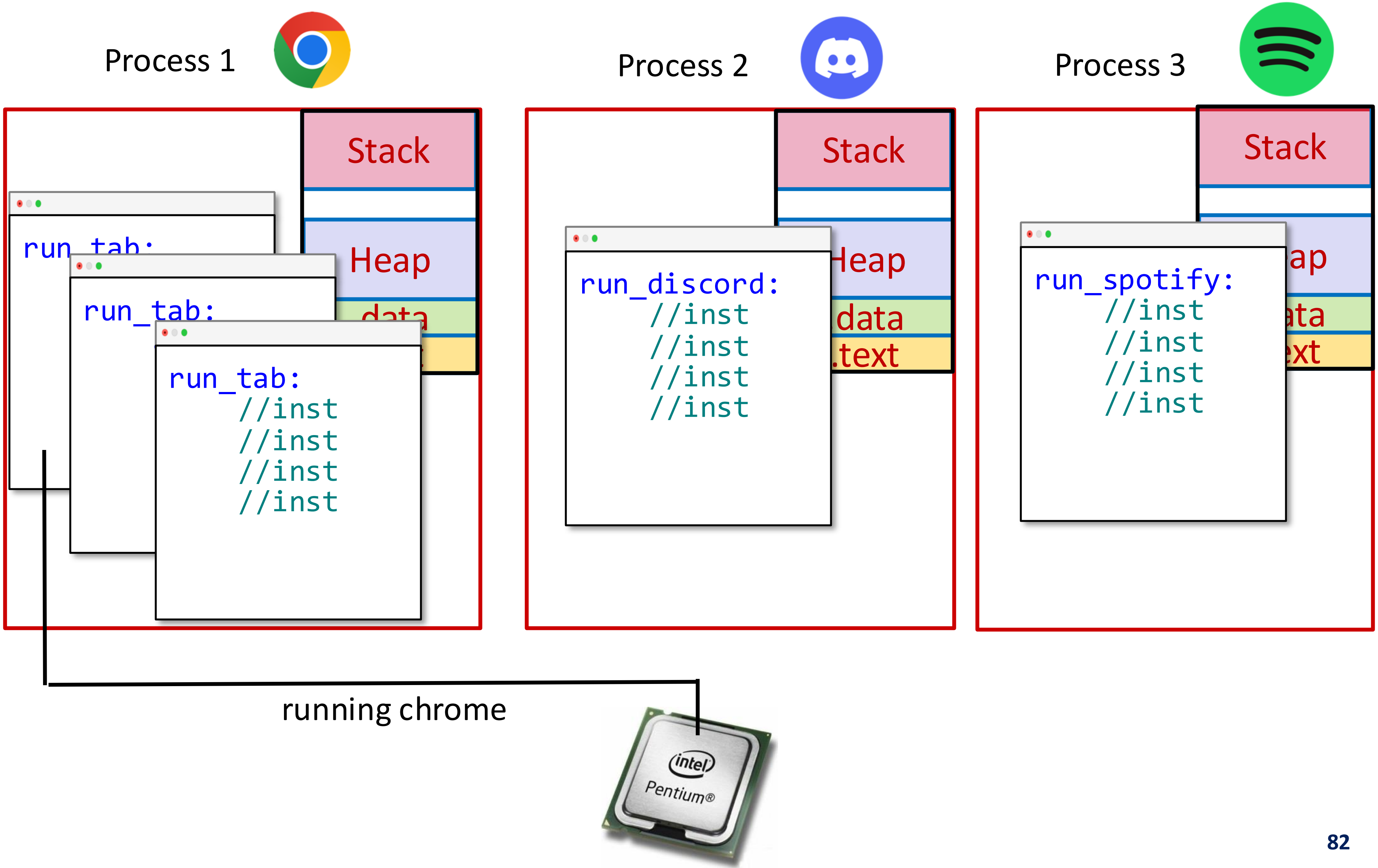
Why not something like this?

All tabs share the same memory since they're running the same application—this makes sense.

However, each tab operates *independently*, maintaining its own execution context.

This is One Process with Three Threads

Processes vs. Threads: What's the Difference?



```
run_tab:
```

```
run_tab:
```

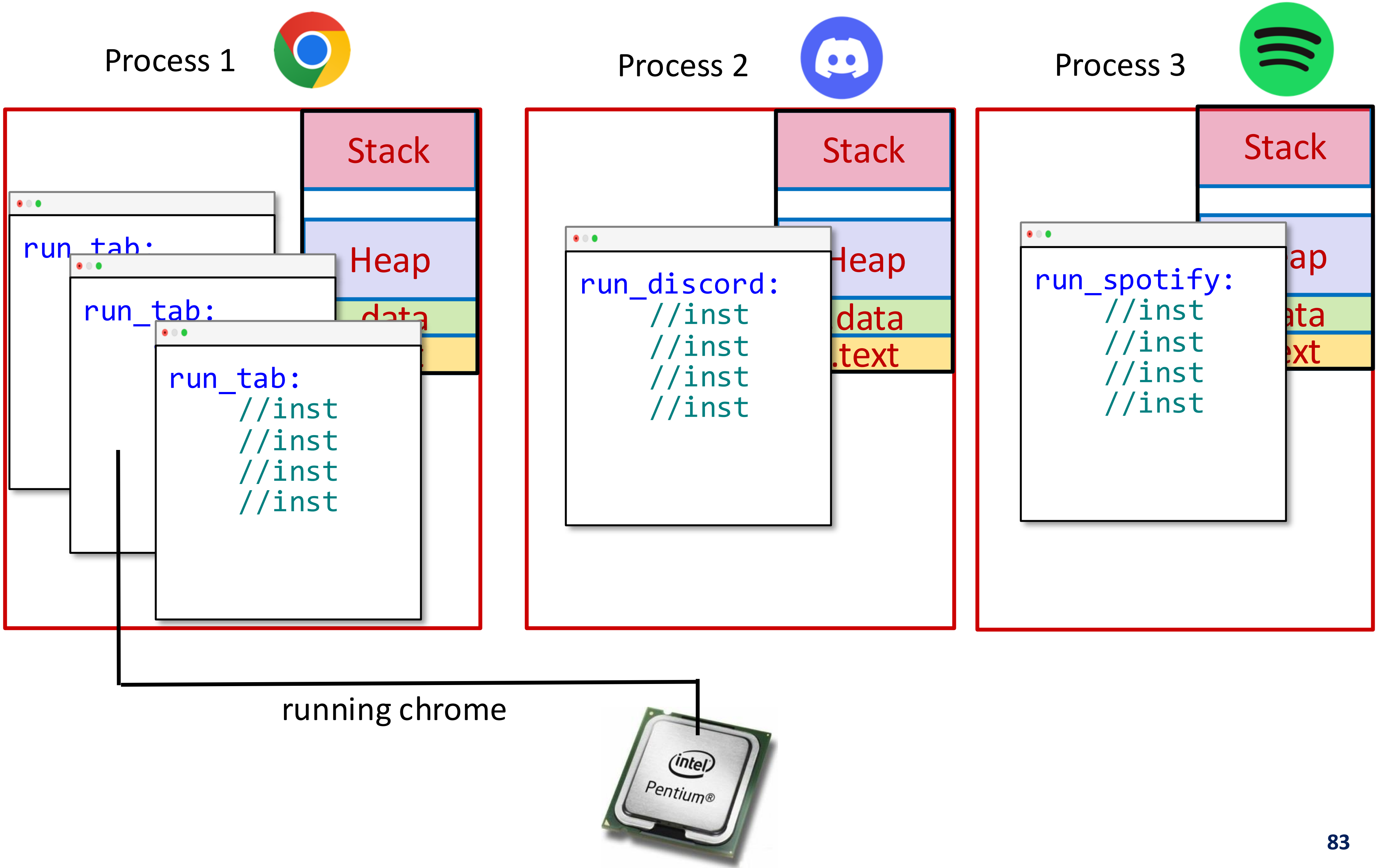
```
run_tab:
    //inst
    //inst
    //inst
    //inst
```

```
run_discord:
    //inst
    //inst
    //inst
    //inst
```

```
run_spotify:
    //inst
    //inst
    //inst
    //inst
```

running chrome

Processes vs. Threads: What's the Difference?



```
run_tab:
```

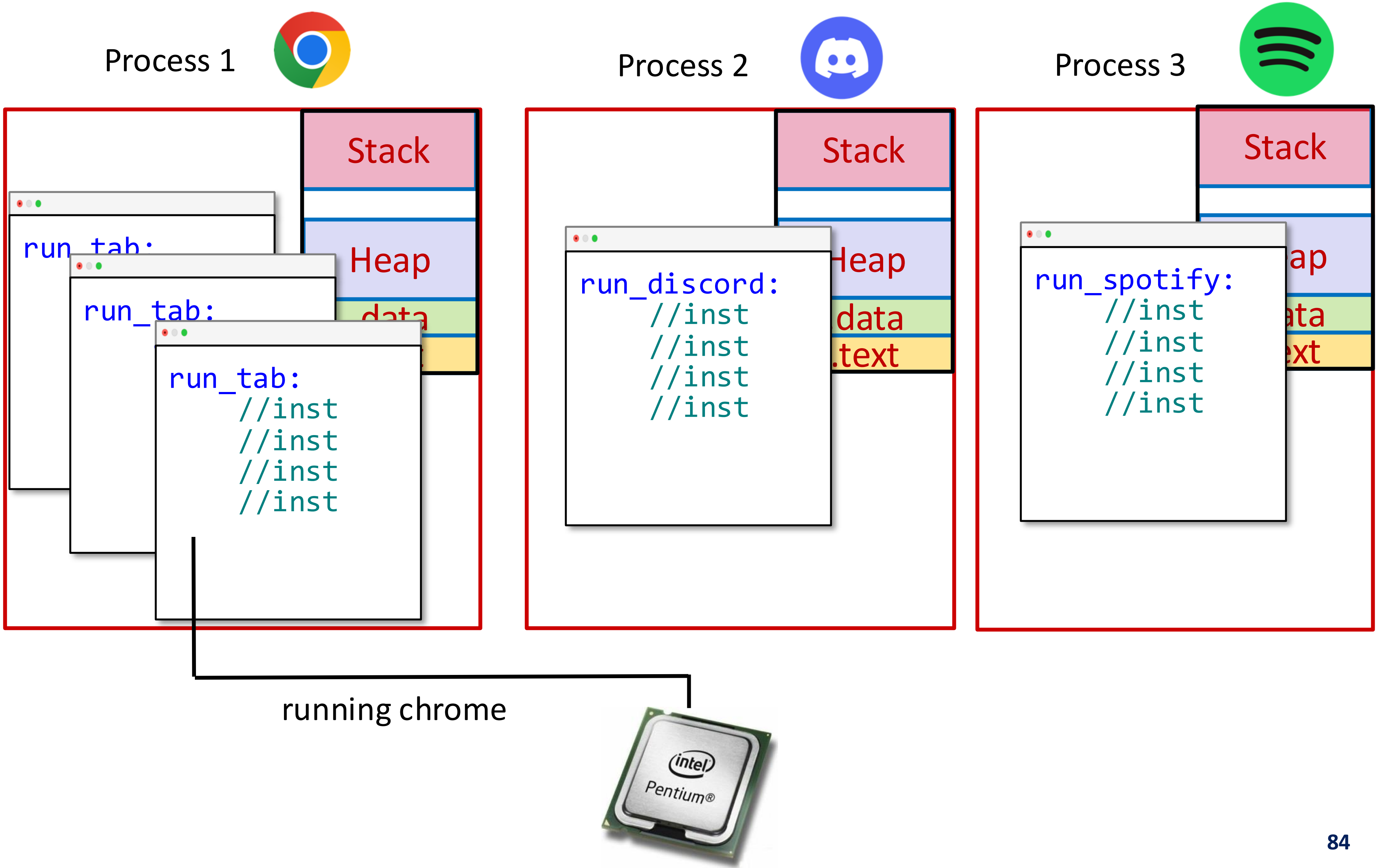
```
run_tab:
```

```
run_tab:
    //inst
    //inst
    //inst
    //inst
```

```
run_discord:
    //inst
    //inst
    //inst
    //inst
```

```
run_spotify:
    //inst
    //inst
    //inst
    //inst
```

Processes vs. Threads: What's the Difference?



```
run_tab:
```

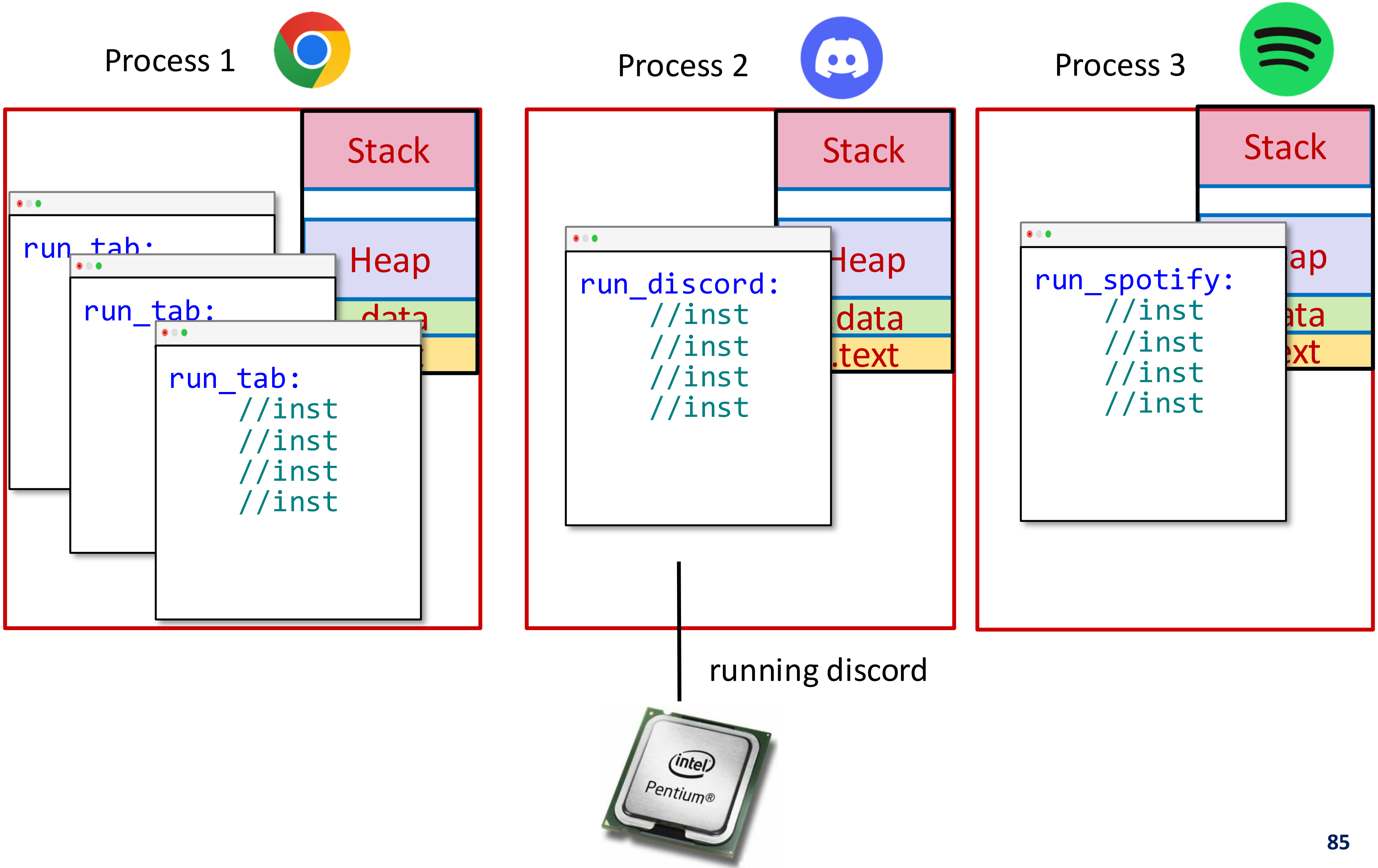
```
run_tab:
```

```
run_tab:
    //inst
    //inst
    //inst
    //inst
```

```
run_discord:
    //inst
    //inst
    //inst
    //inst
```

```
run_spotify:
    //inst
    //inst
    //inst
    //inst
```

Processes vs. Threads: What's the Difference?



```
run_tab:
```

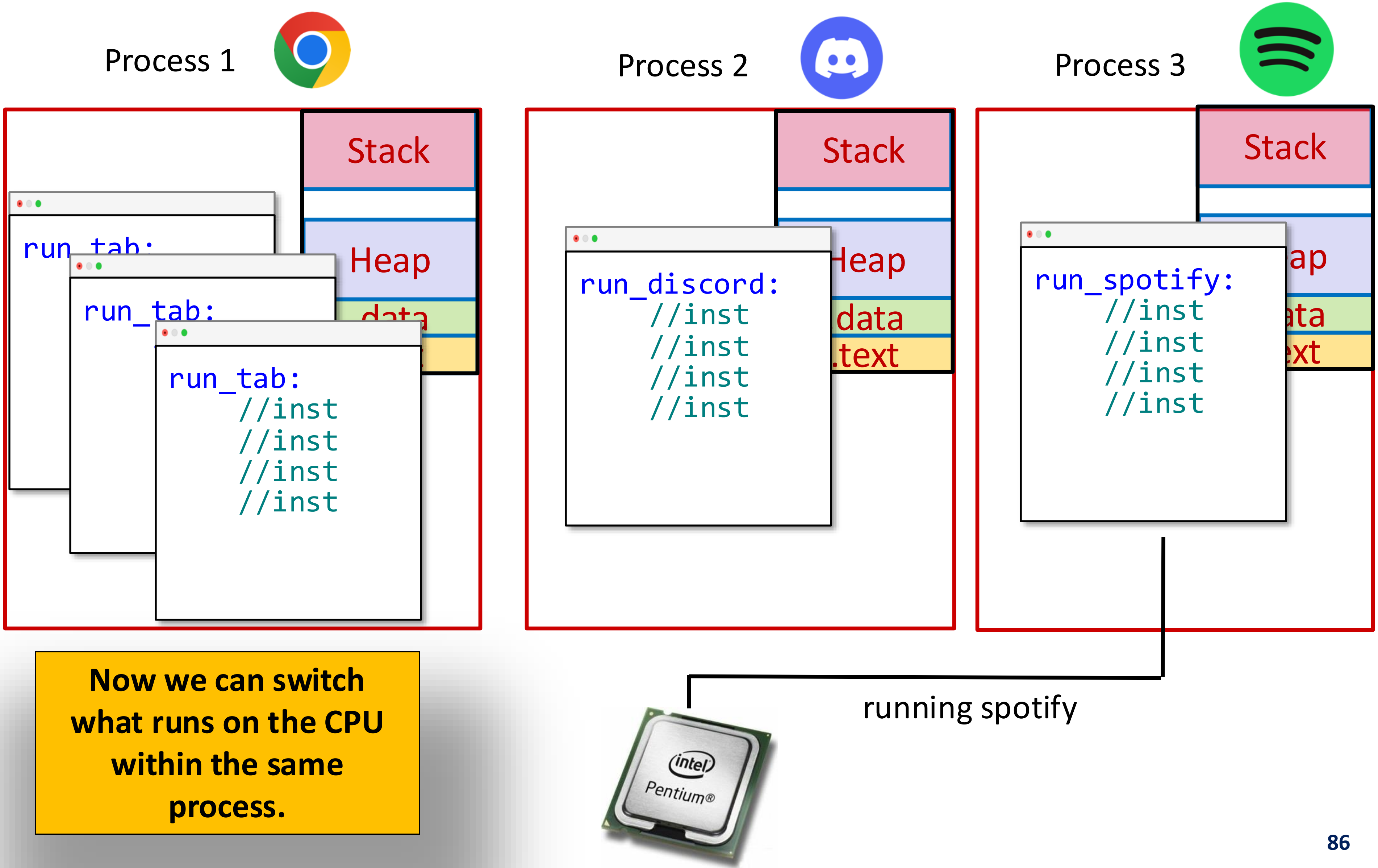
```
run_tab:
```

```
run_tab:
        //inst
        //inst
        //inst
        //inst
```

```
run_discord:
        //inst
        //inst
        //inst
        //inst
```

```
run_spotify:
        //inst
        //inst
        //inst
        //inst
```

Processes vs. Threads: What's the Difference?



```
run_tab:
```

```
run_tab:
```

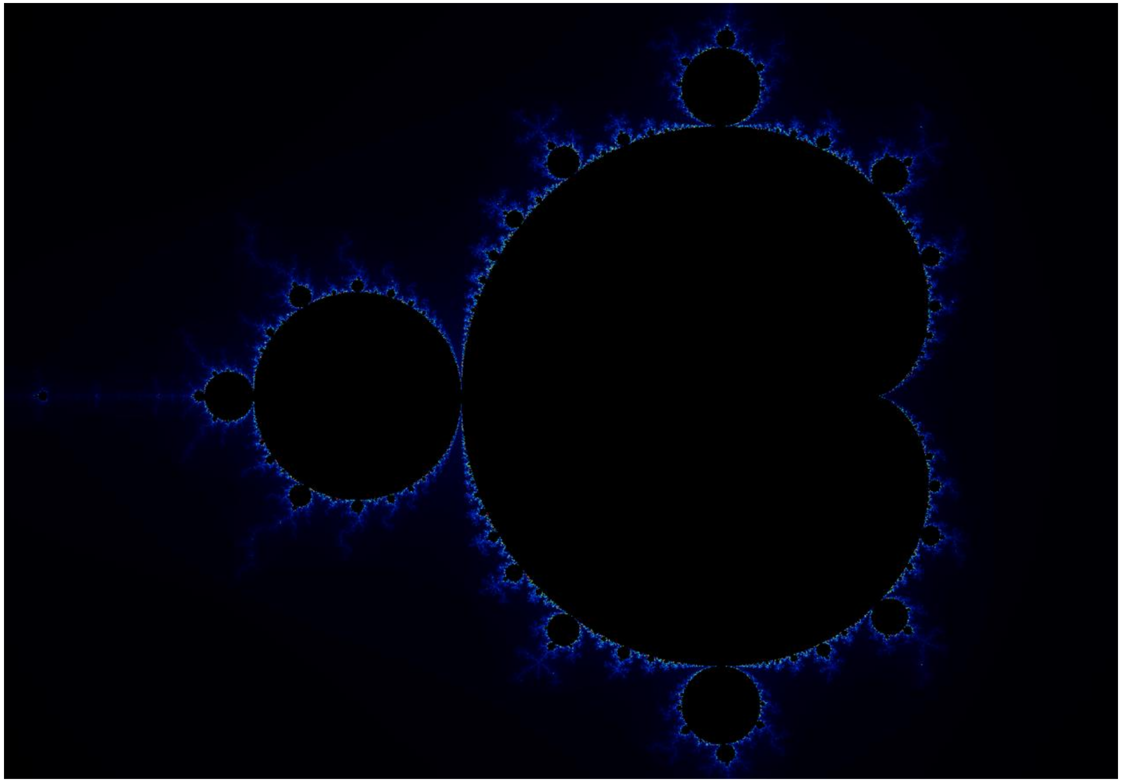
```
run_tab:
    //inst
    //inst
    //inst
    //inst
```

```
run_discord:
    //inst
    //inst
    //inst
    //inst
```

```
run_spotify:
    //inst
    //inst
    //inst
    //inst
```

Now we can switch what runs on the CPU within the same process.

Example: Visualizing the Mandlebrot Set



Compute Intensity

We need to compute for each pixel, if it belongs in the set in addition.

Currently: ~960000 values

Let’s compare how threads help us here:

Non-Threaded vs Fully Threaded Implementations

And that’s it! ☺