Async programming: GCD

The four horsemen of asynchronicity: Sync, Async, Serial and Concurrent

by Fernando Olivares

Agenda

1-hour session

- Author Introduction
- Also why we're here: What is GCD?
- Live coding
  - The 4 horsemen of Asynchronicity
- Interesting tools: Semaphores and Groups
- Conclusion & Where to go from here
- Q&A

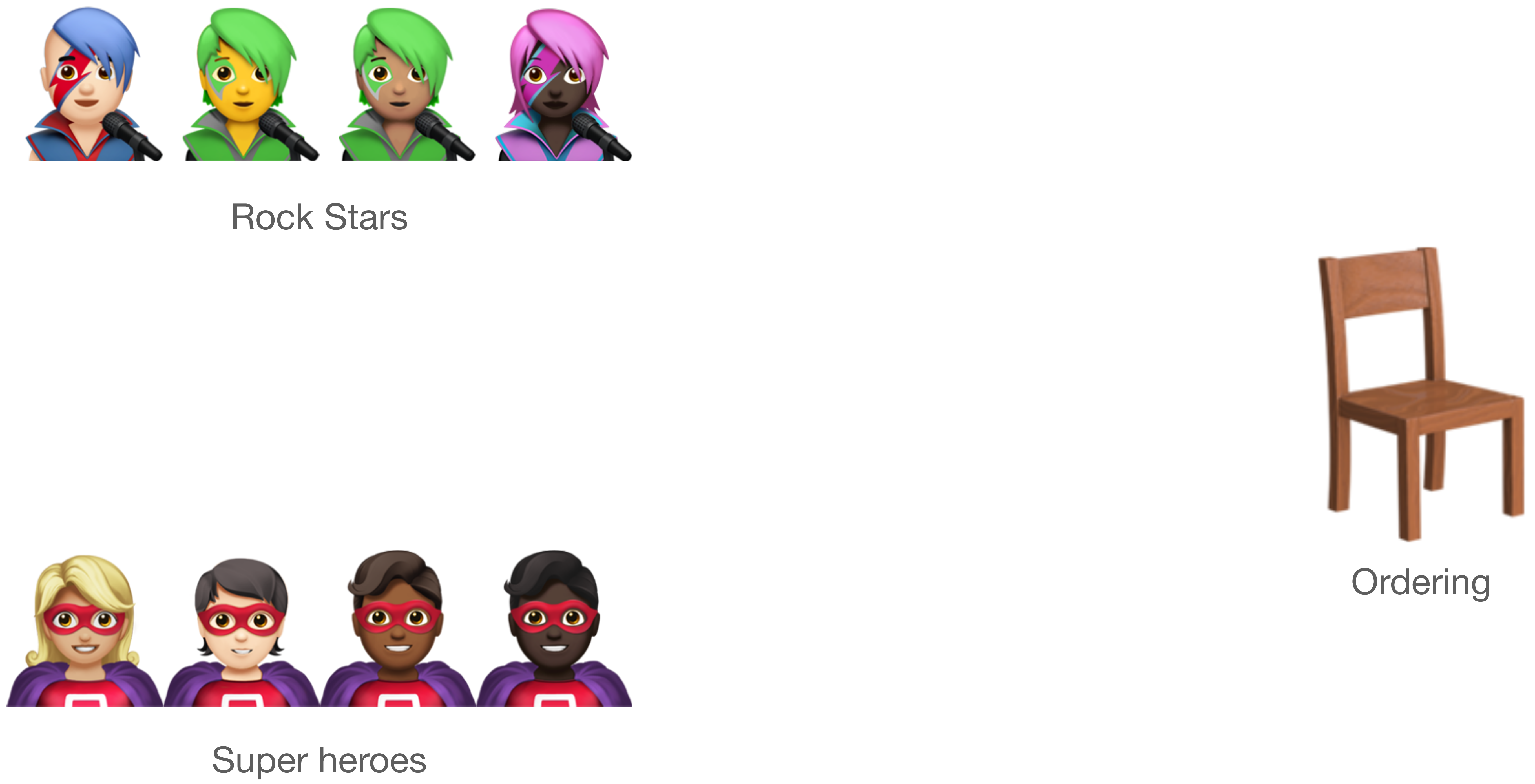
Author Introduction

Fernando

- ~10 years of experience
- Worked at small startups (1SecondEveryday) to publicly traded companies (J2 Global Inc.)
- Instructor at Big Nerd Ranch, bloc.io, Lambda School
- Won a few awards: The Storyteller Within (Apple), ERA Accelerator Top 10 (ERA NY)
- Product and Project experience
- iOS-only
- @fromJrToSr

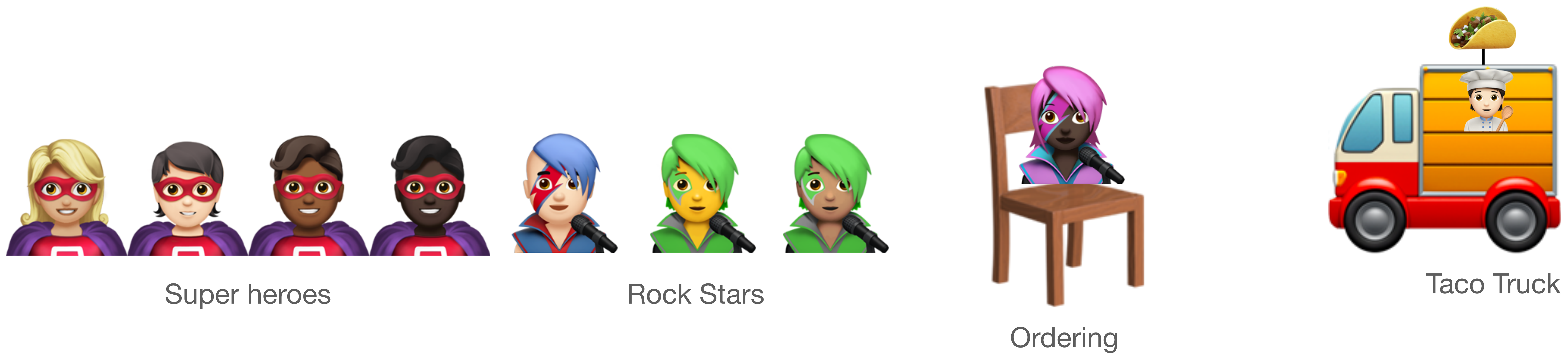
Synchronous programming

A long time ago... in a taco truck far, far away.



Synchronous programming

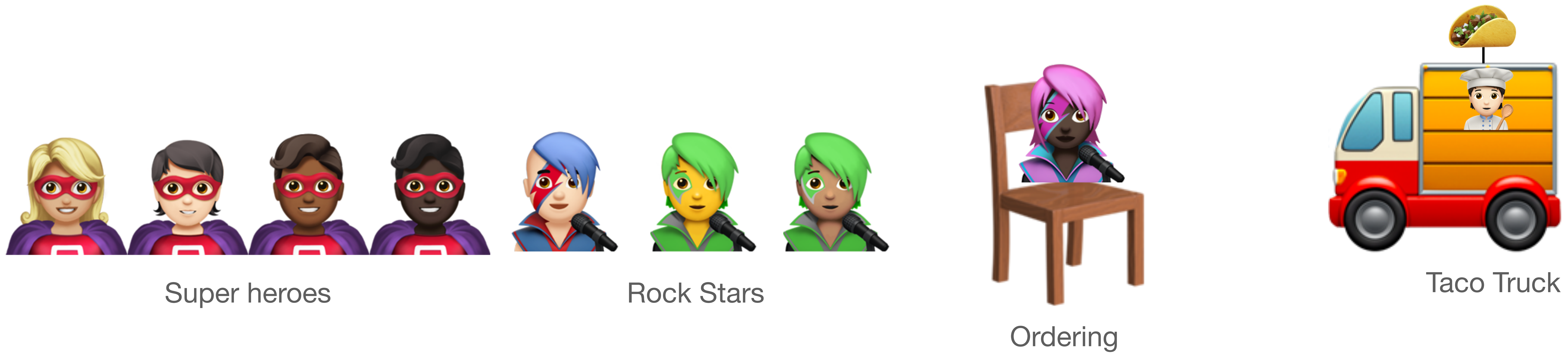
First come, first serve



Synchronous programming

Fist come, first serve

- Disadvantages:
    - We only have one chef.



Synchronous programming

Fist come, first serve

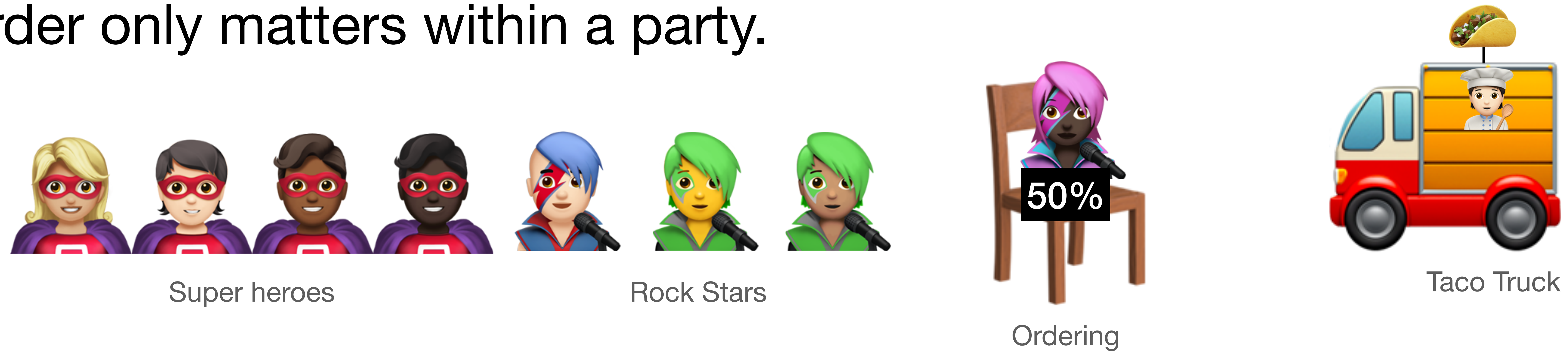
- Disadvantages:
  - We only have one chef.
  - If a someone takes too long, the rest of the line suffers.

Super heroes
Rock Stars
Ordering
Taco Truck

Synchronous programming

Fist come, first serve

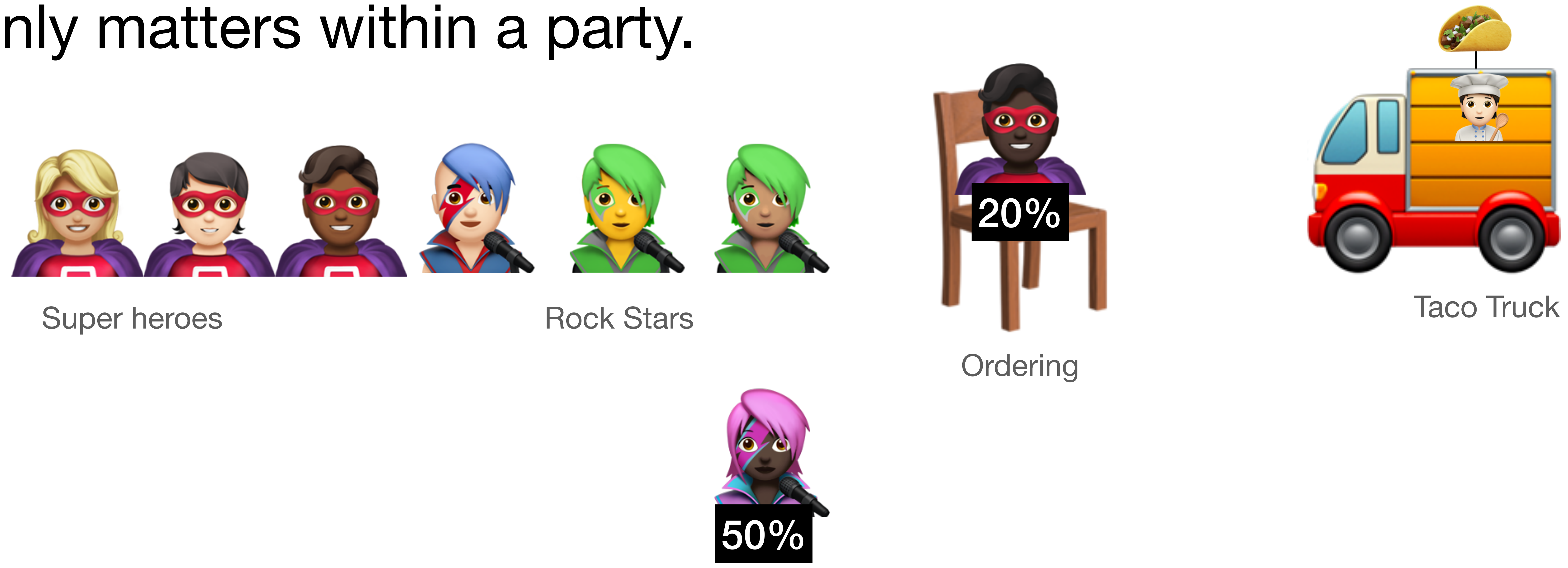
- Possible solution:
  - Prepare the dishes partially.
  - Order only matters within a party.



Synchronous programming

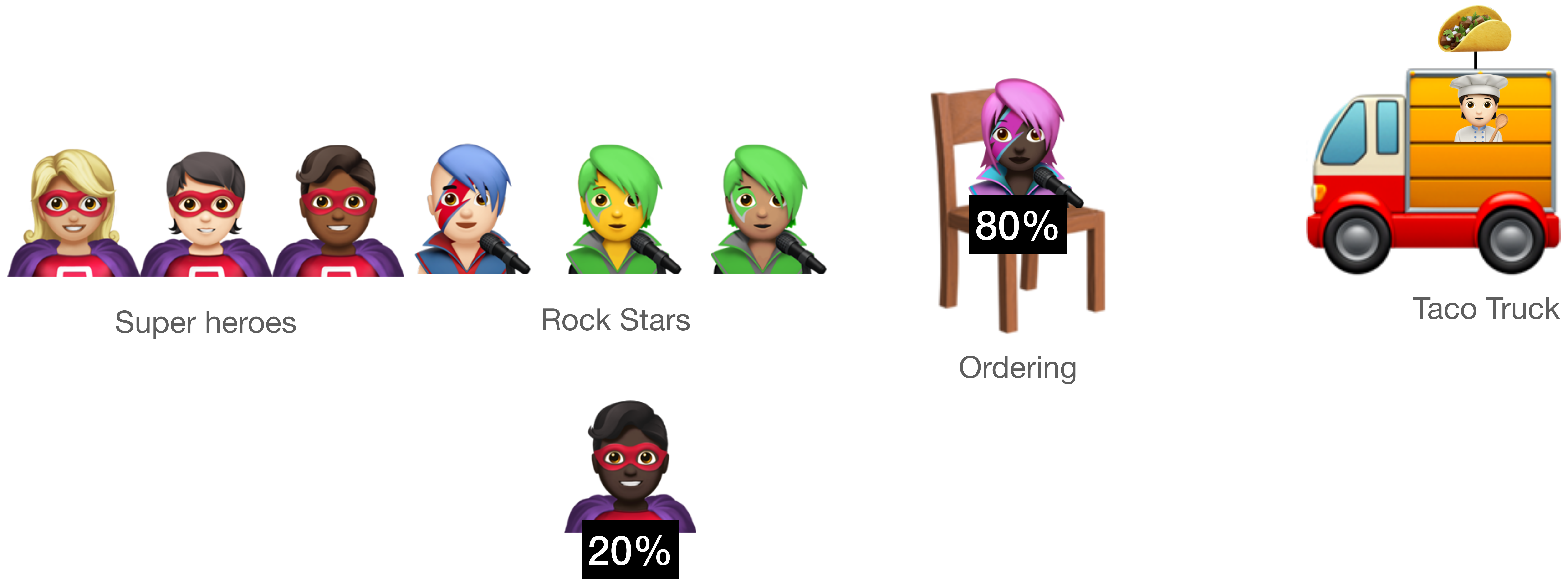
Fist come, first serve

- Possible solution:
  - Prepare the dishes partially.
  - Order only matters within a party.



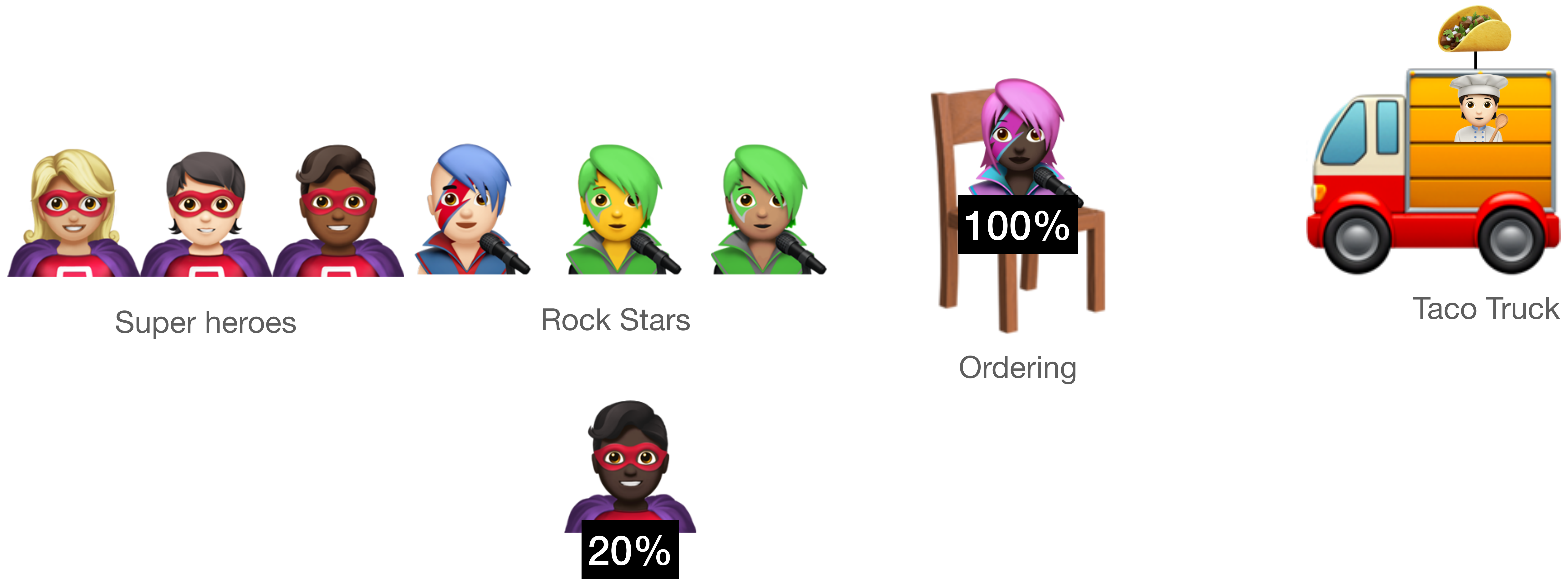
Synchronous programming

Illusion of one chef serving different parties.



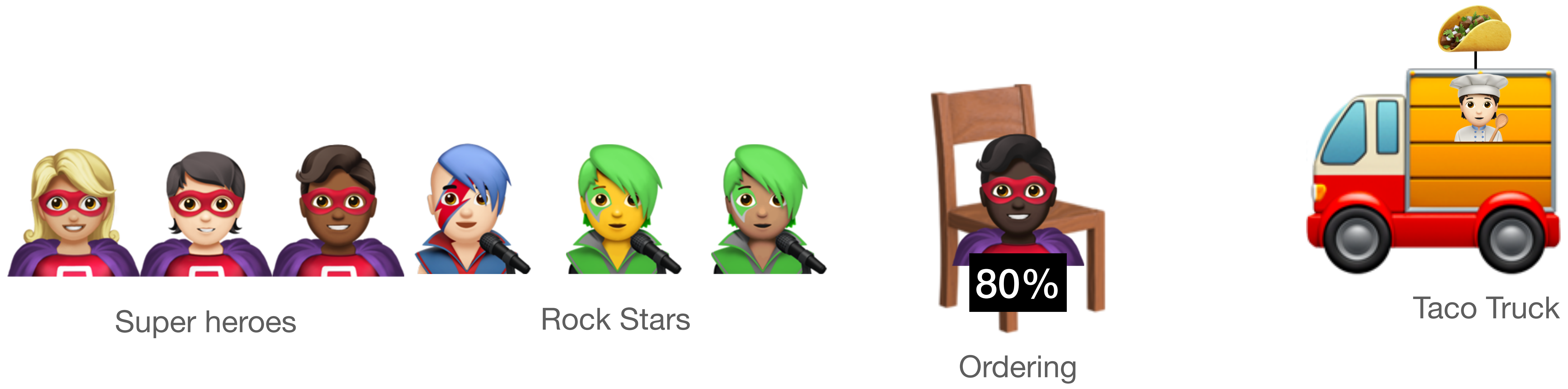
Synchronous programming

Illusion of one chef serving different parties.



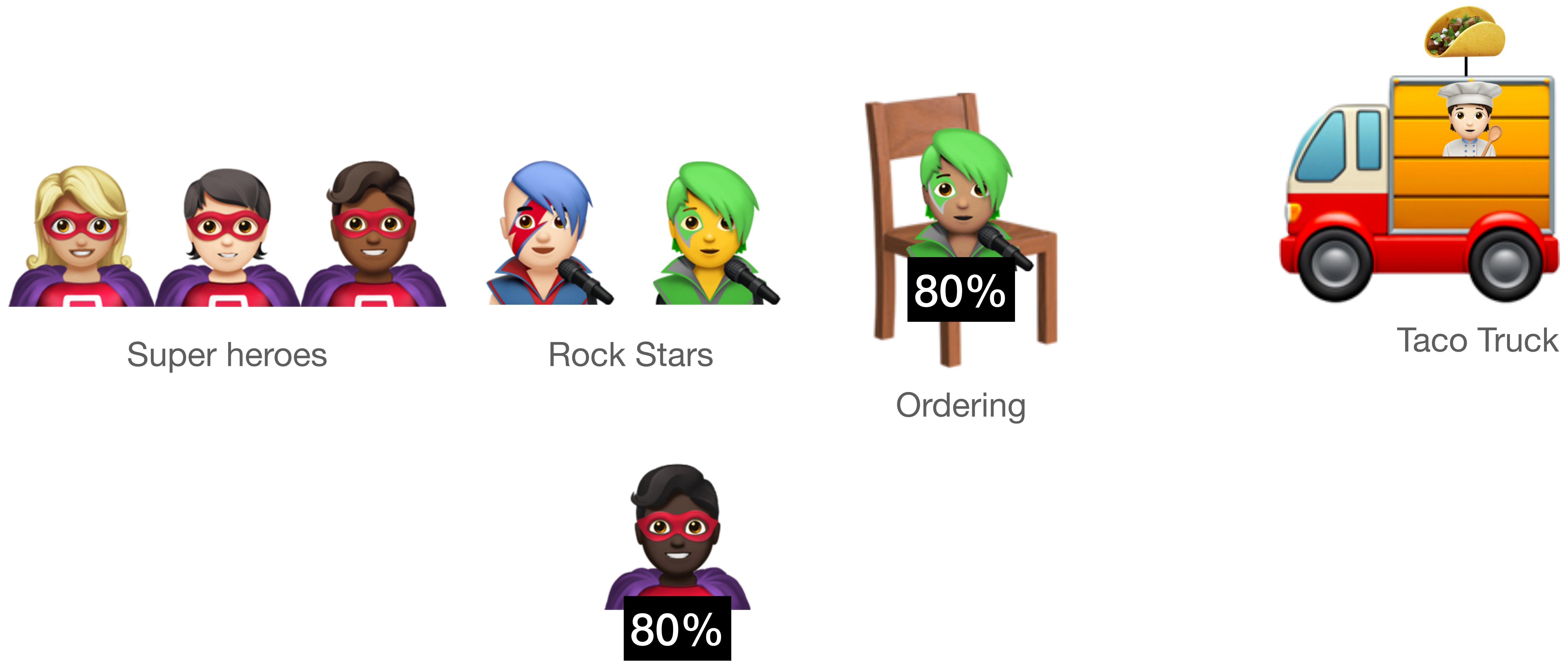
Synchronous programming

Illusion of one chef serving different parties.



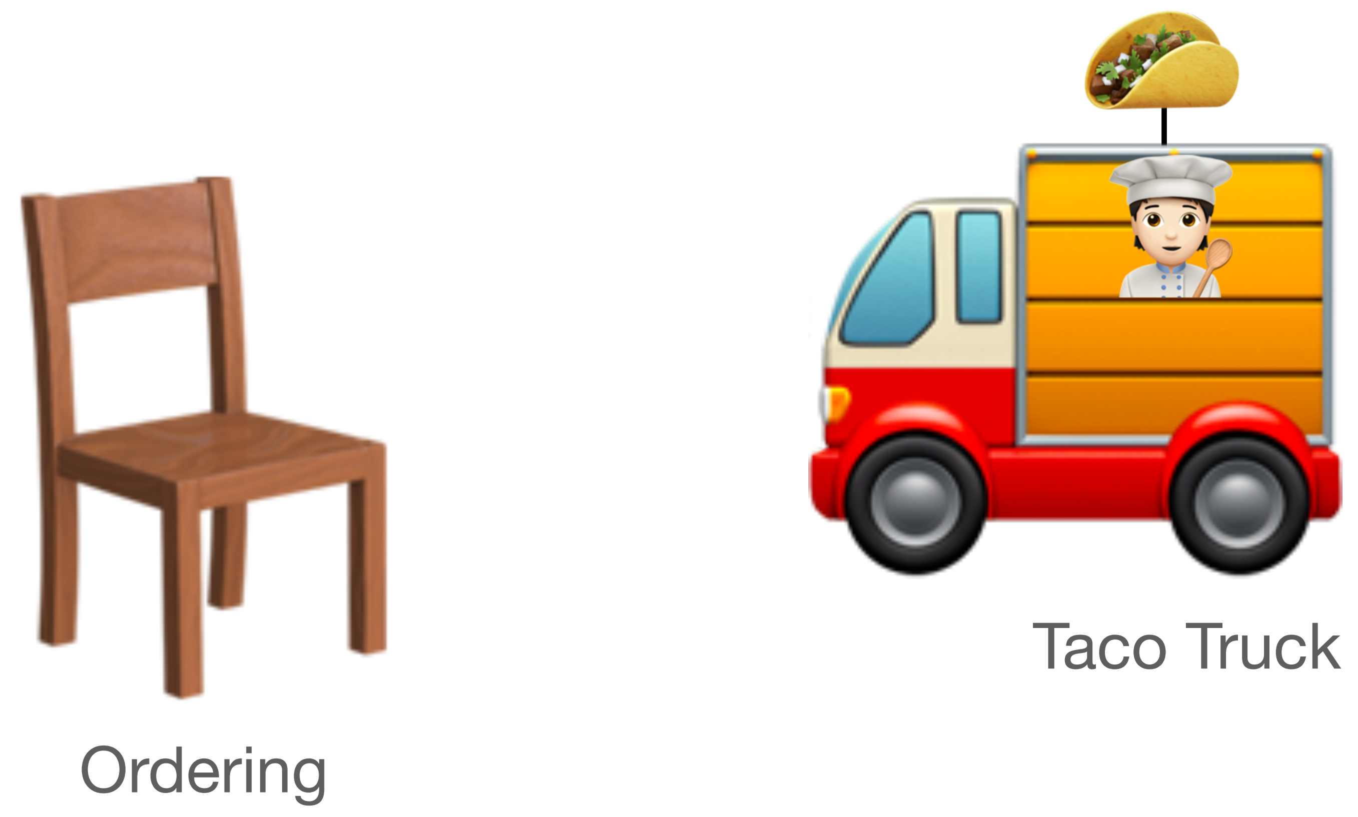
Synchronous programming

Illusion of one chef serving different parties.



Synchronous programming

Everyone ate tacos and left.

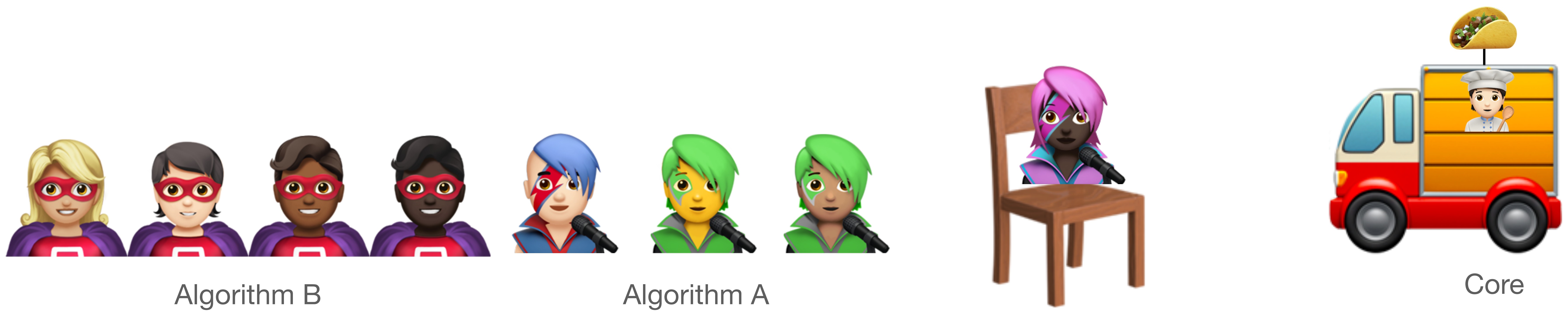


Synchronous programming

A long time ago... in a single-core computer.

Synchronous programming

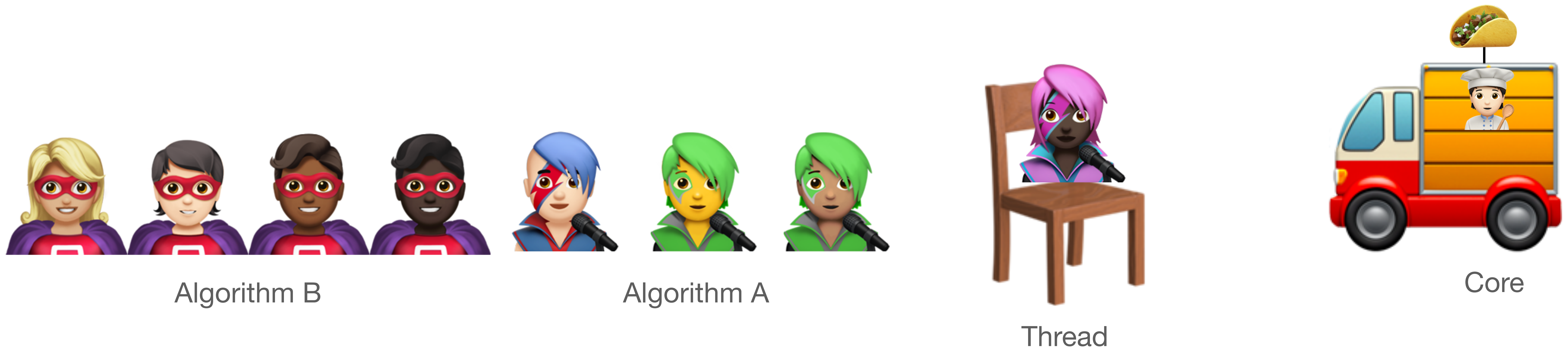
FIFO - First in, First out



Synchronous programming

FIFO - First in, First out

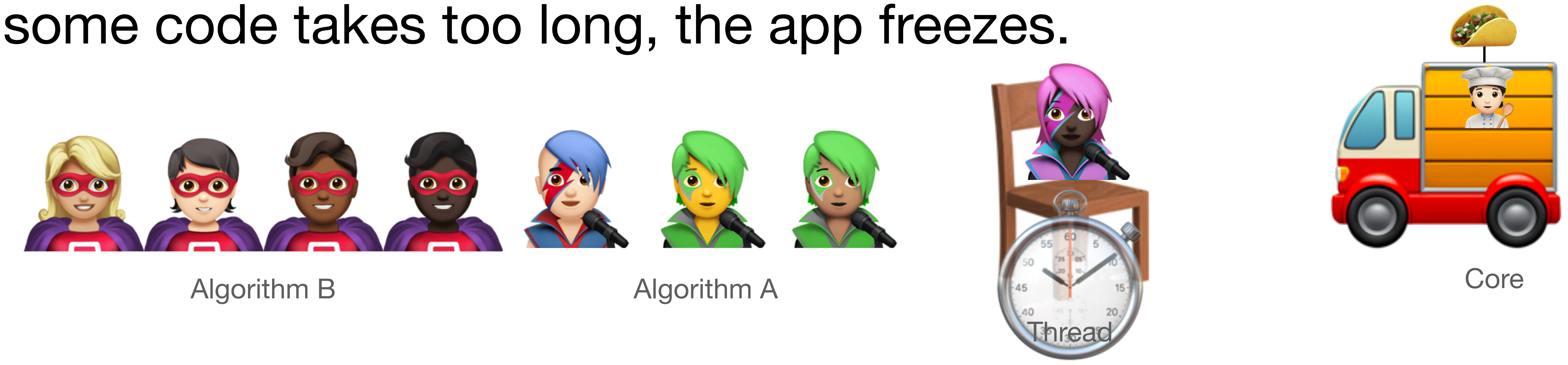
- Disadvantages:
    - We only have one core.



Synchronous programming

FIFO - First in, First out

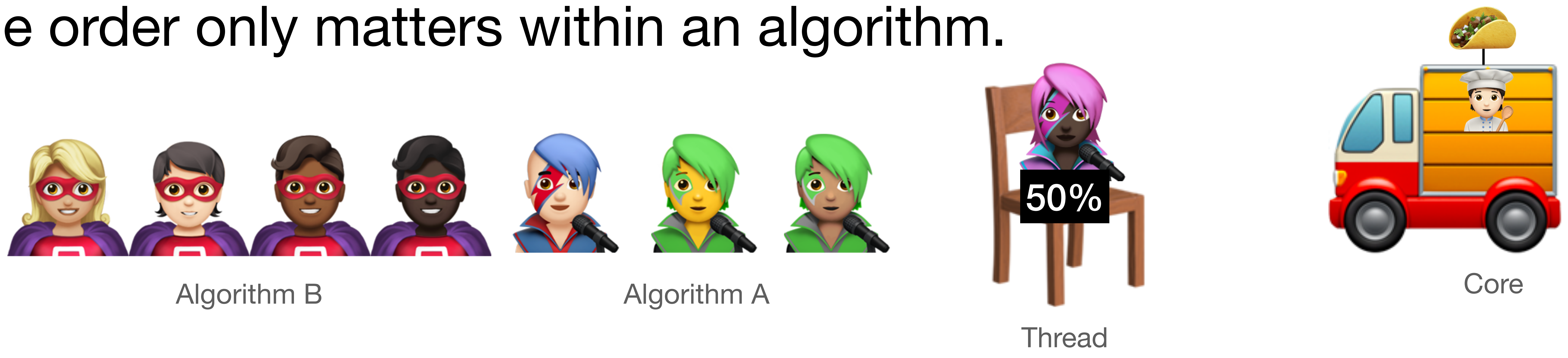
- Disadvantages:
    - We only have one core.
    - If some code takes too long, the app freezes.



Synchronous programming

FIFO - First in, First out

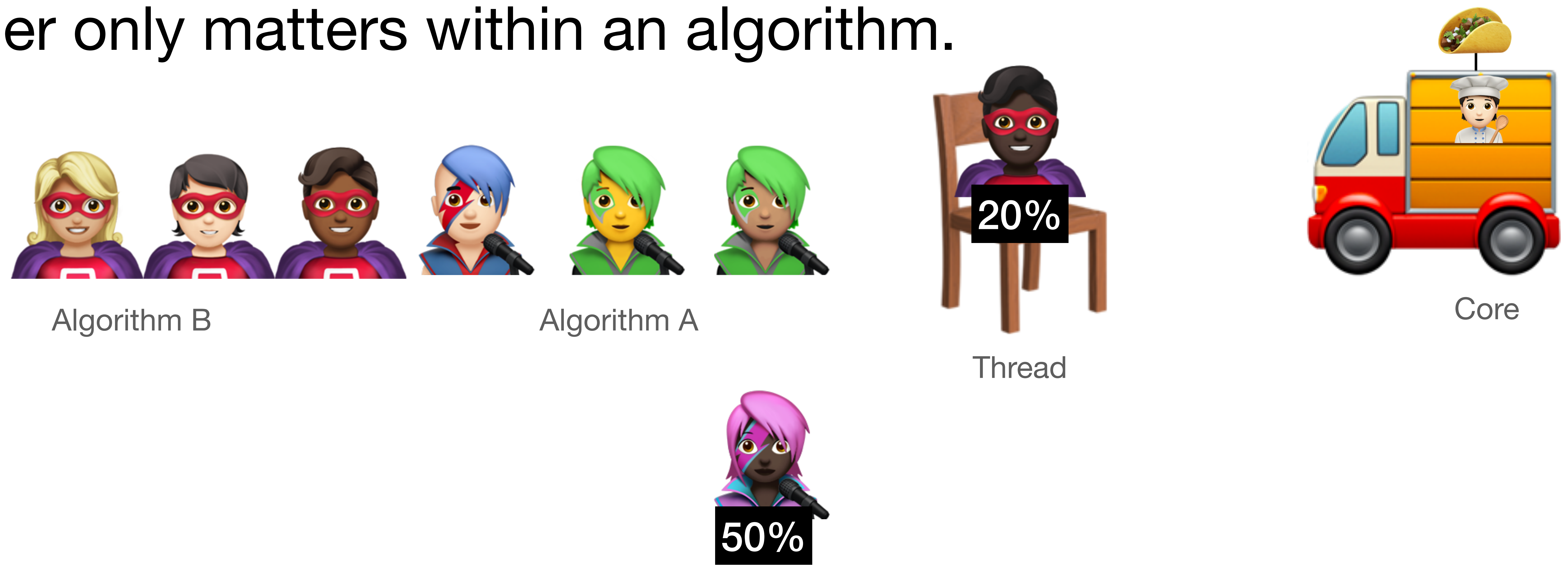
- Possible solution:
  - Execute parts of the code partially. (Pseudoparallelism)
  - The order only matters within an algorithm.



Synchronous programming

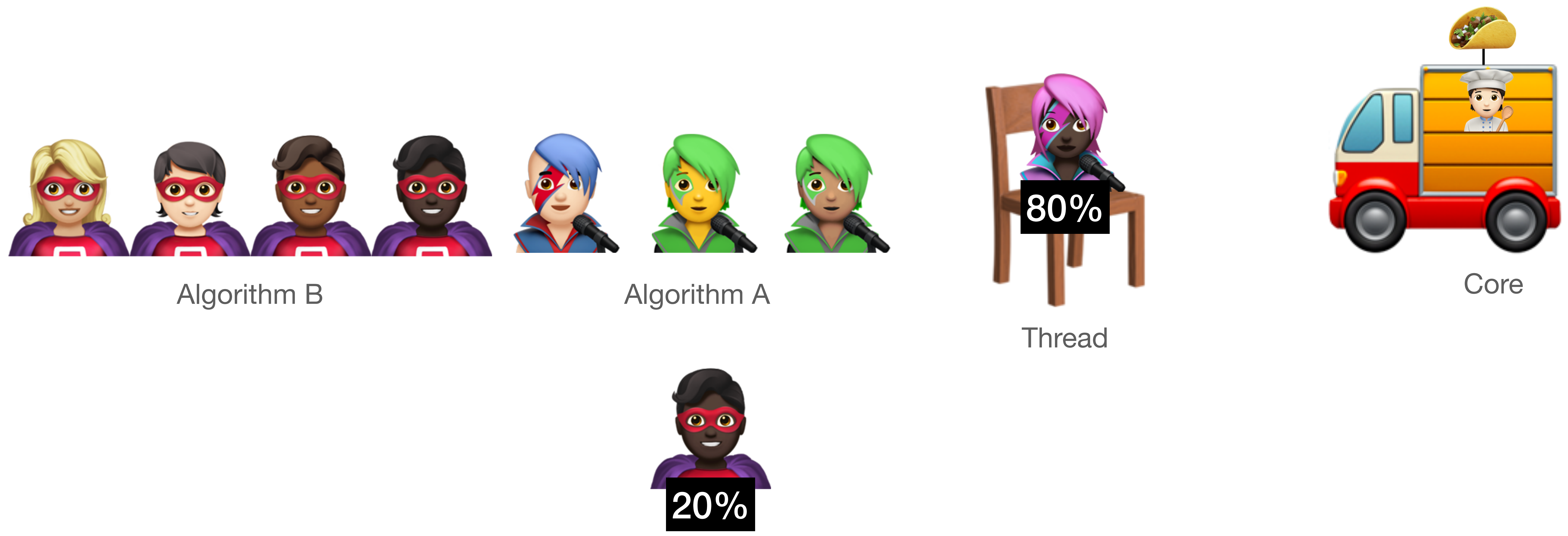
FIFO - First in, First out

- Possible solution:
  - Execute parts of the code partially. (Pseudoparallelism)
  - The order only matters within an algorithm.



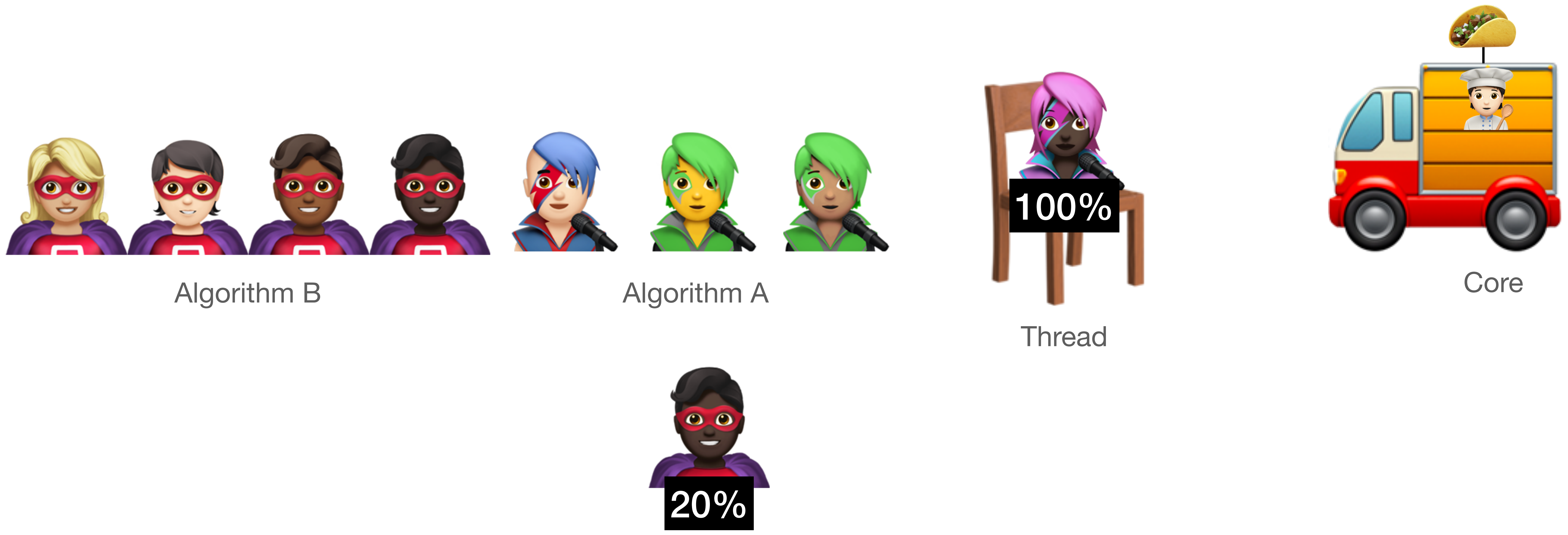
Synchronous programming

Illusion of one core performing several algorithms.



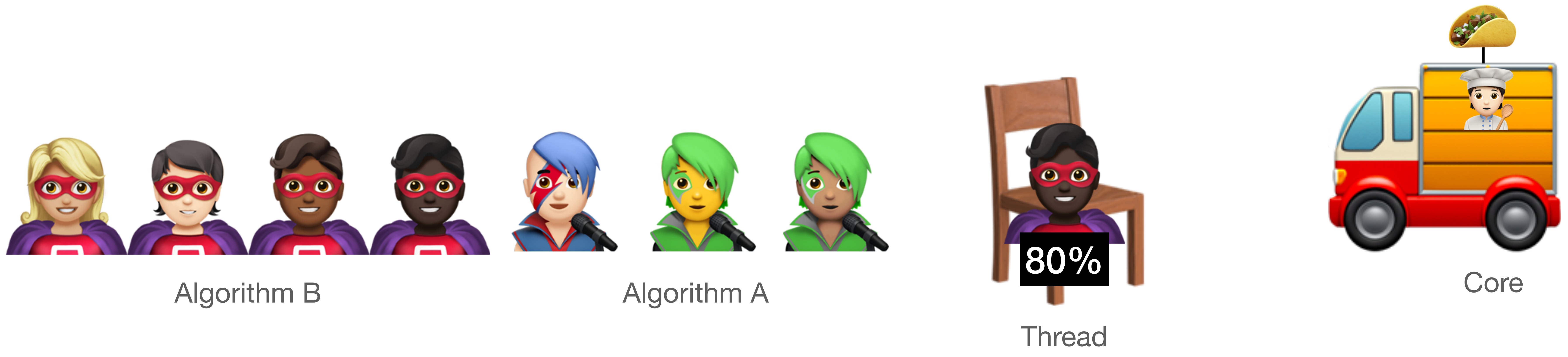
Synchronous programming

Illusion of one core performing several algorithms.



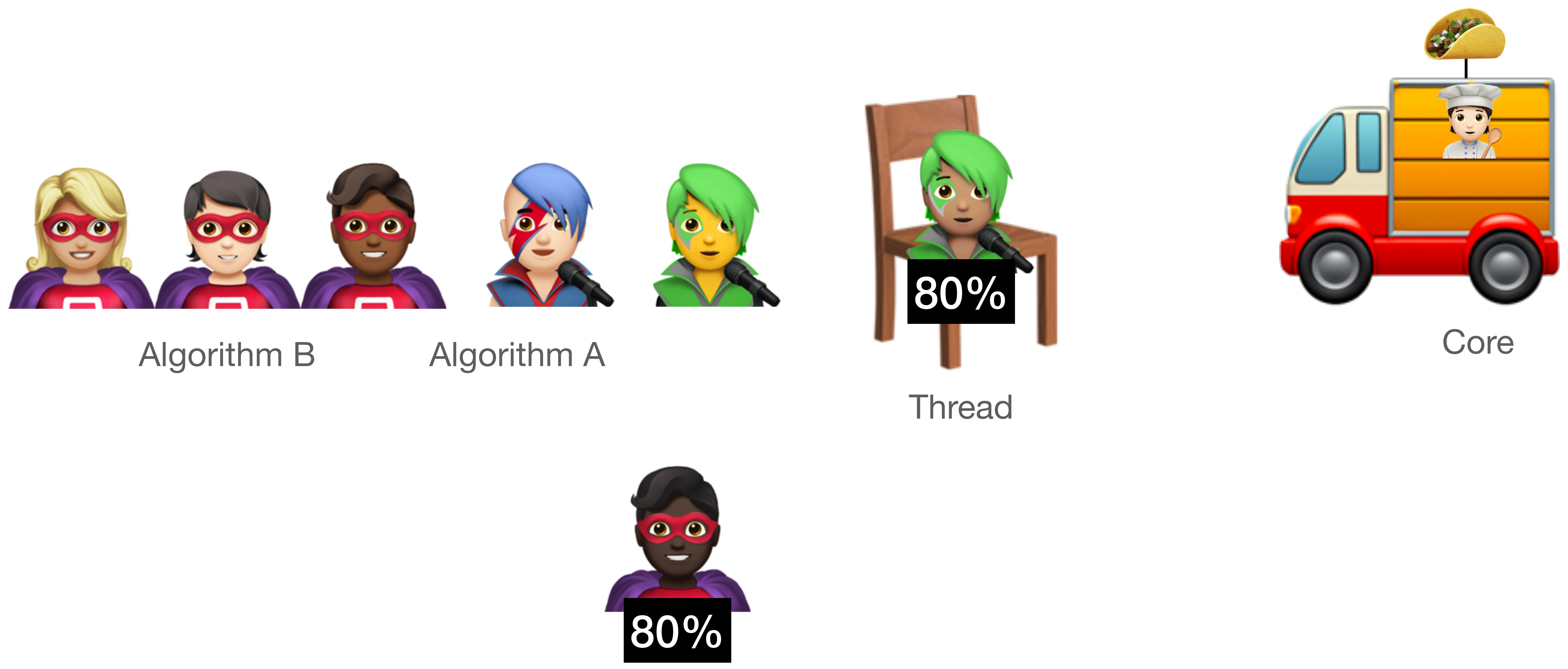
Synchronous programming

Illusion of one core performing several algorithms.



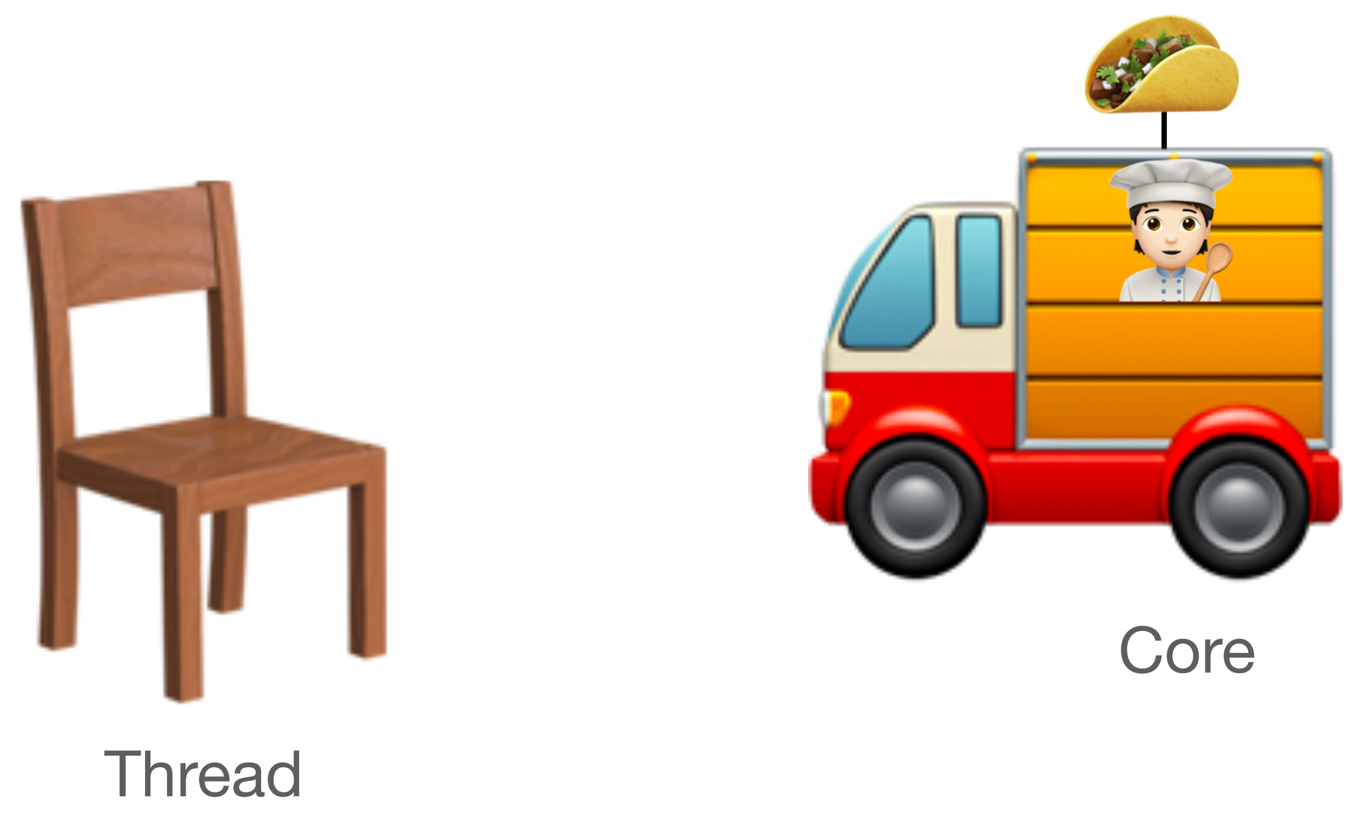
Synchronous programming

Illusion of one core performing several algorithms.



Synchronous programming

All code has been executed.



What is concurrency?

In theory, concurrency is having a smooth UI.

- Official definition: "concurrency is the ability of different parts or units of a program, algorithm, or problem to be executed out-of-order or in partial order, **without affecting the final outcome**"[1]

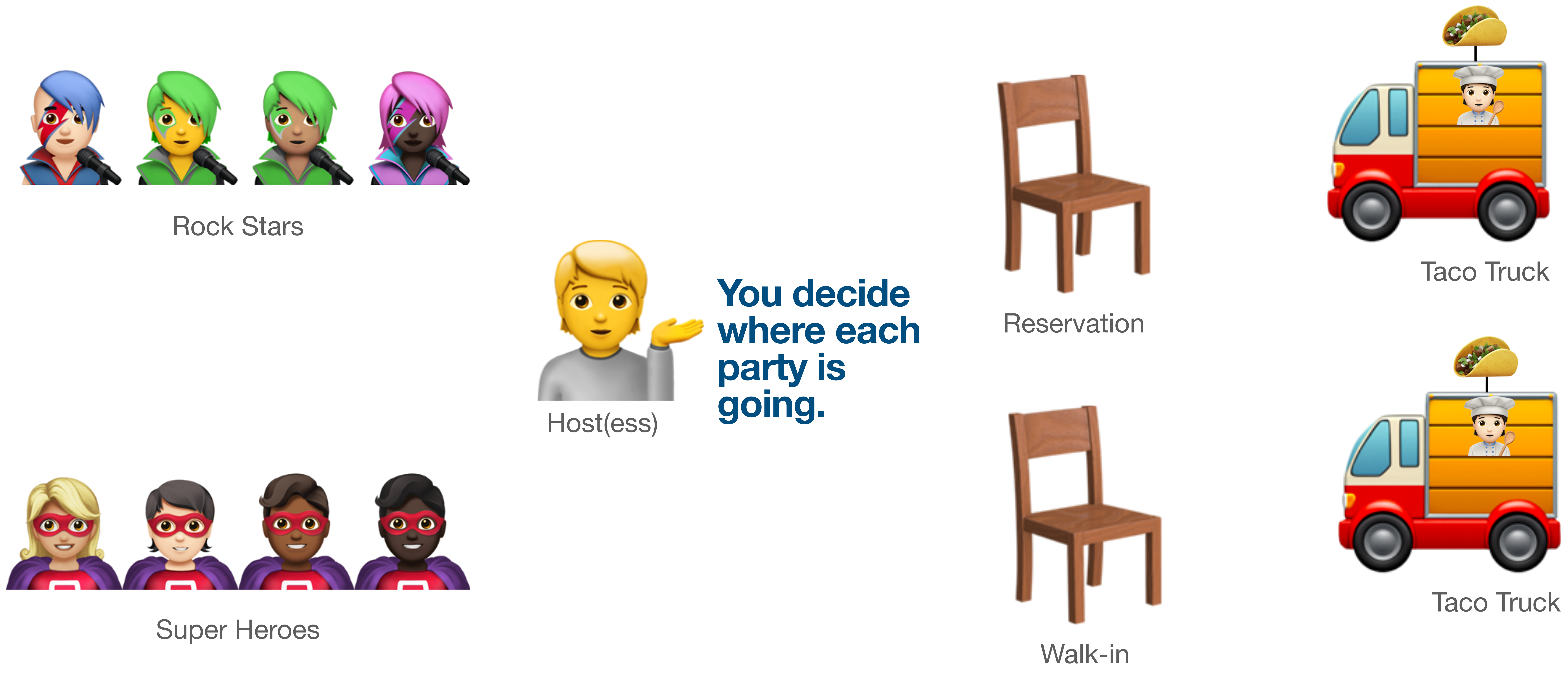
- Concurrency is achieved using threads, which are "the smallest sequence of programmed instructions that can be managed independently by a scheduler".

- Darwin (core OS) is a threaded platform. Threading allows us to execute long-running tasks without blocking execution of other tasks.

[1] - https://en.wikipedia.org/wiki/Concurrency_(computer_science)

Asynchronous programming

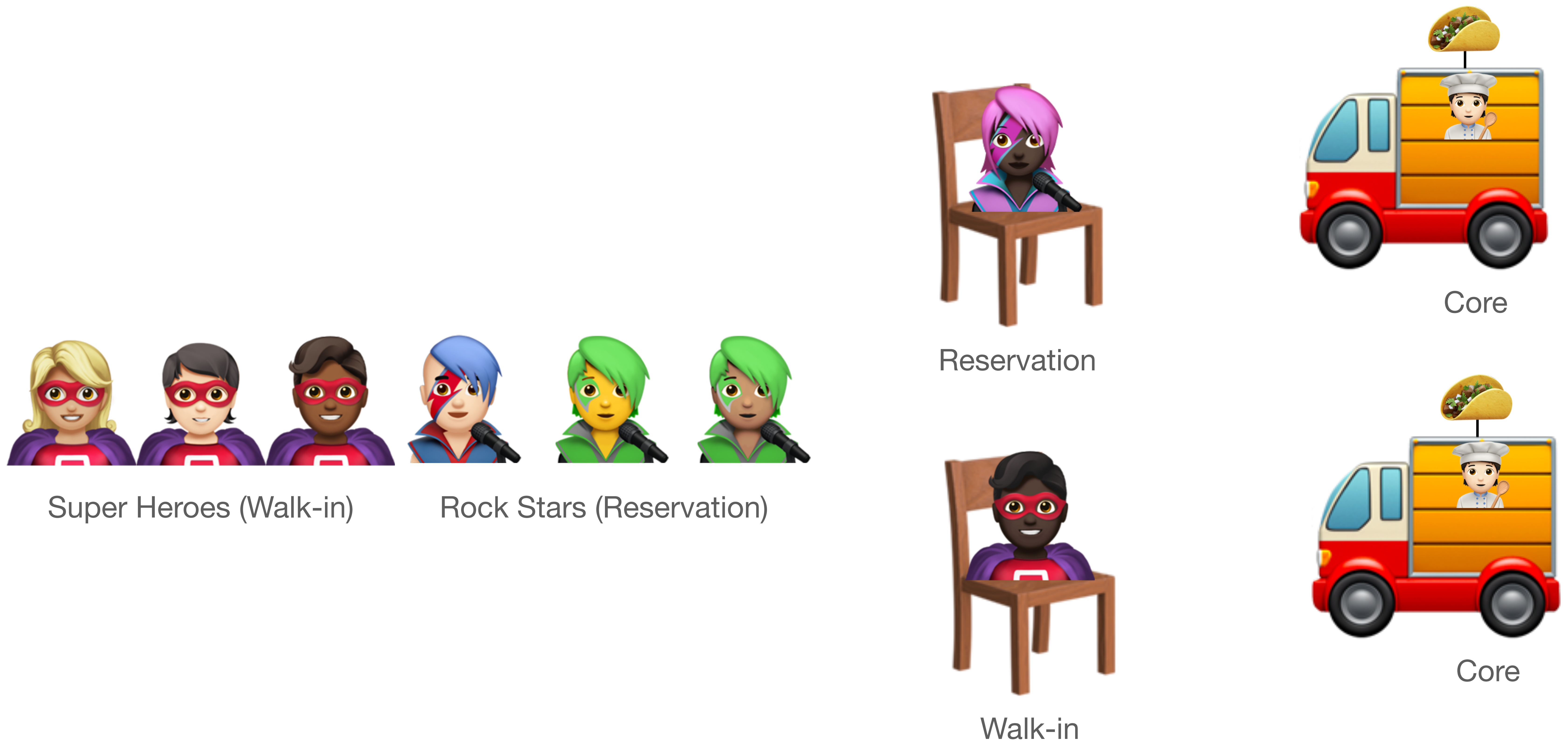
Tacos today.



You decide where each party is going.

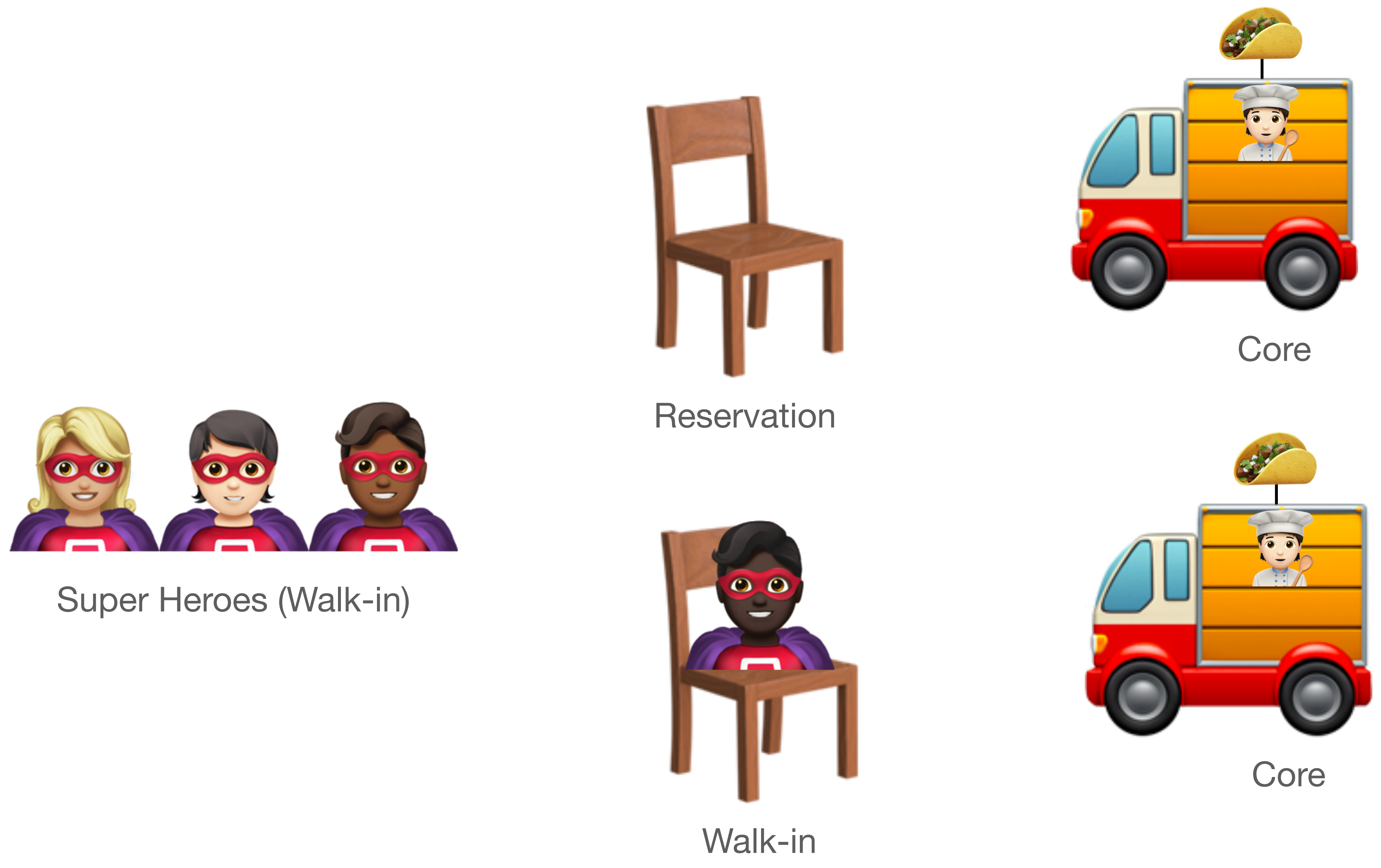
Asynchronous programming

First come, first serve



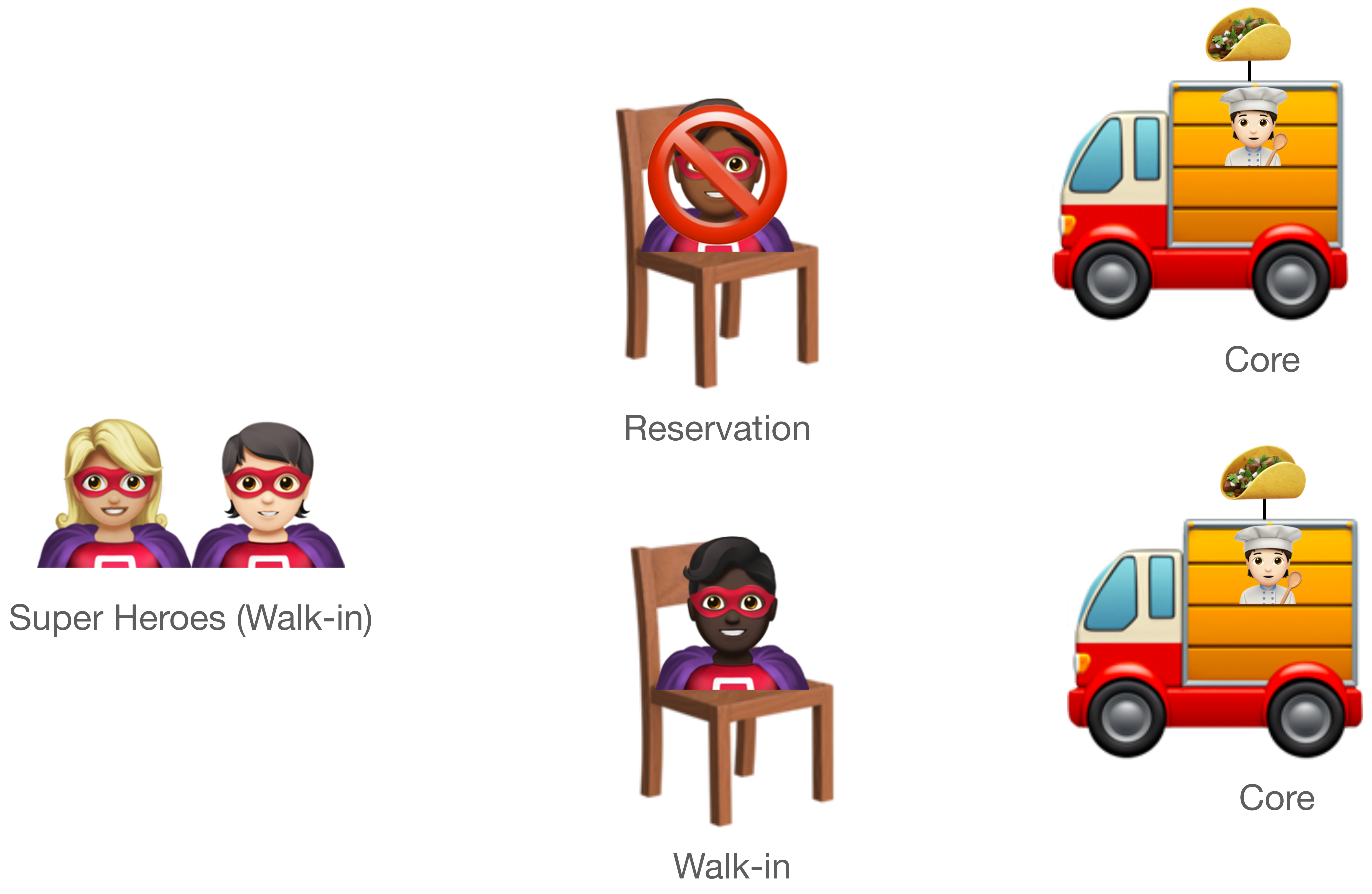
Asynchronous programming

First come, first serve



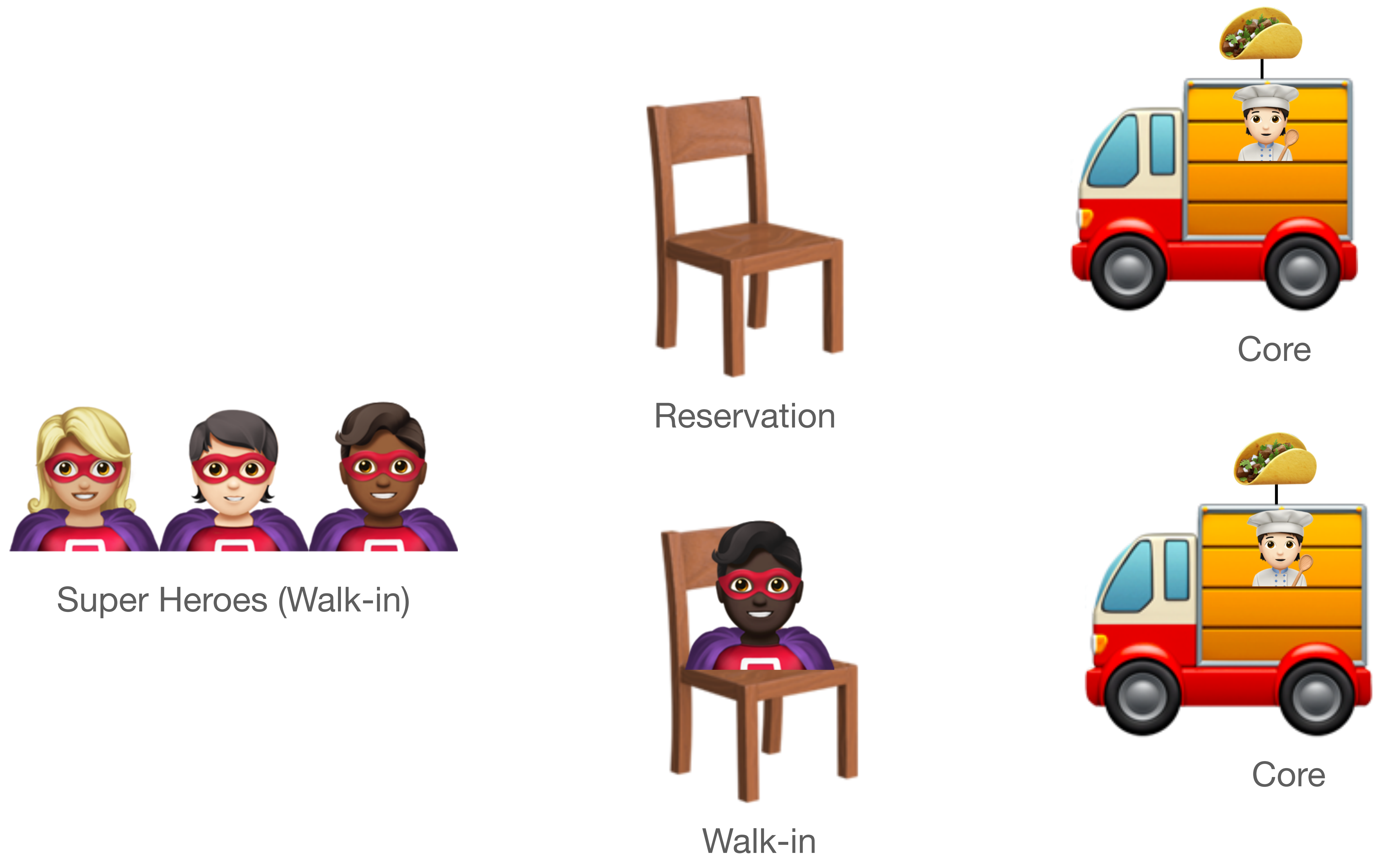
Asynchronous programming

First come, first serve



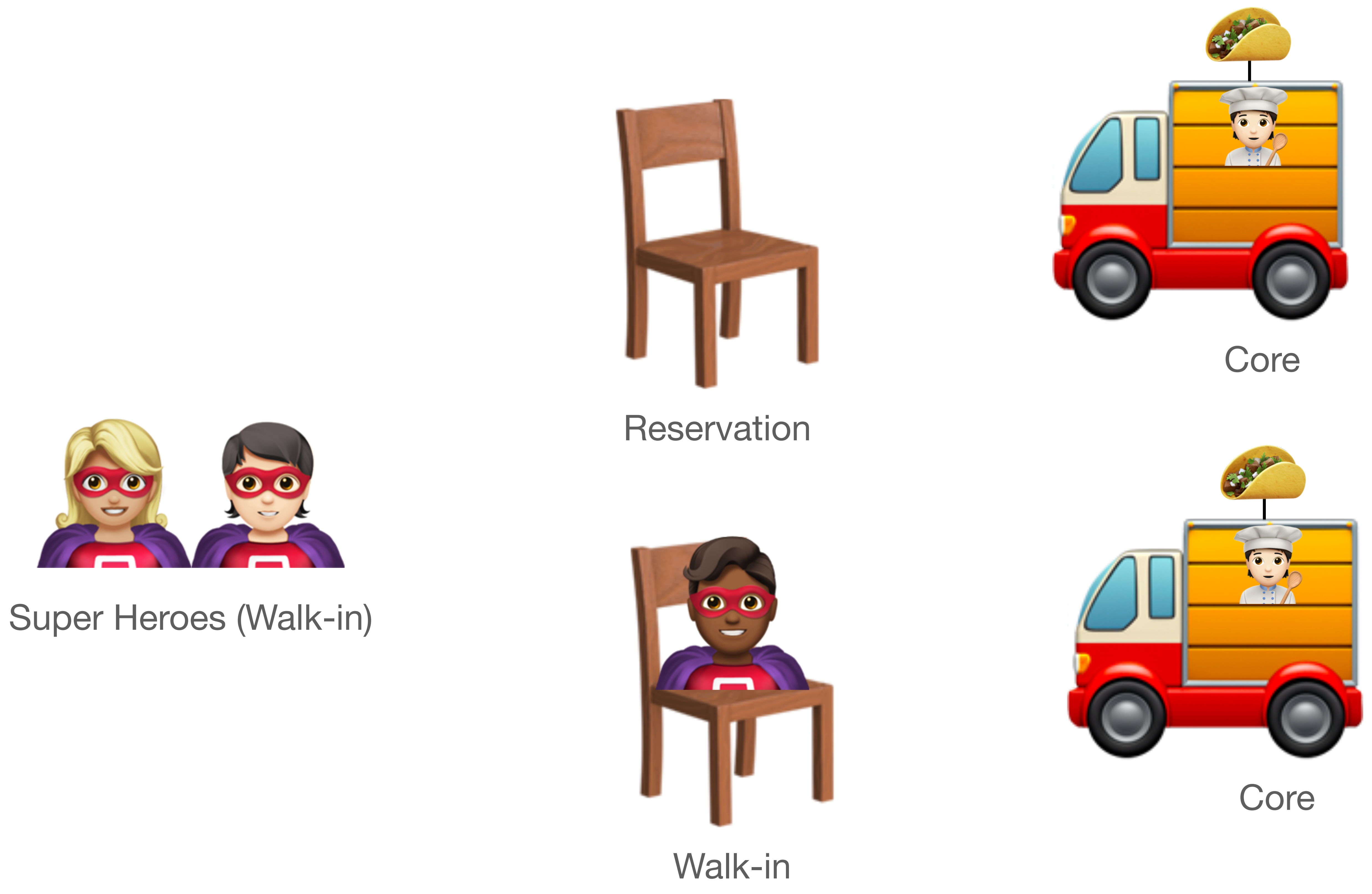
Asynchronous programming

First come, first serve



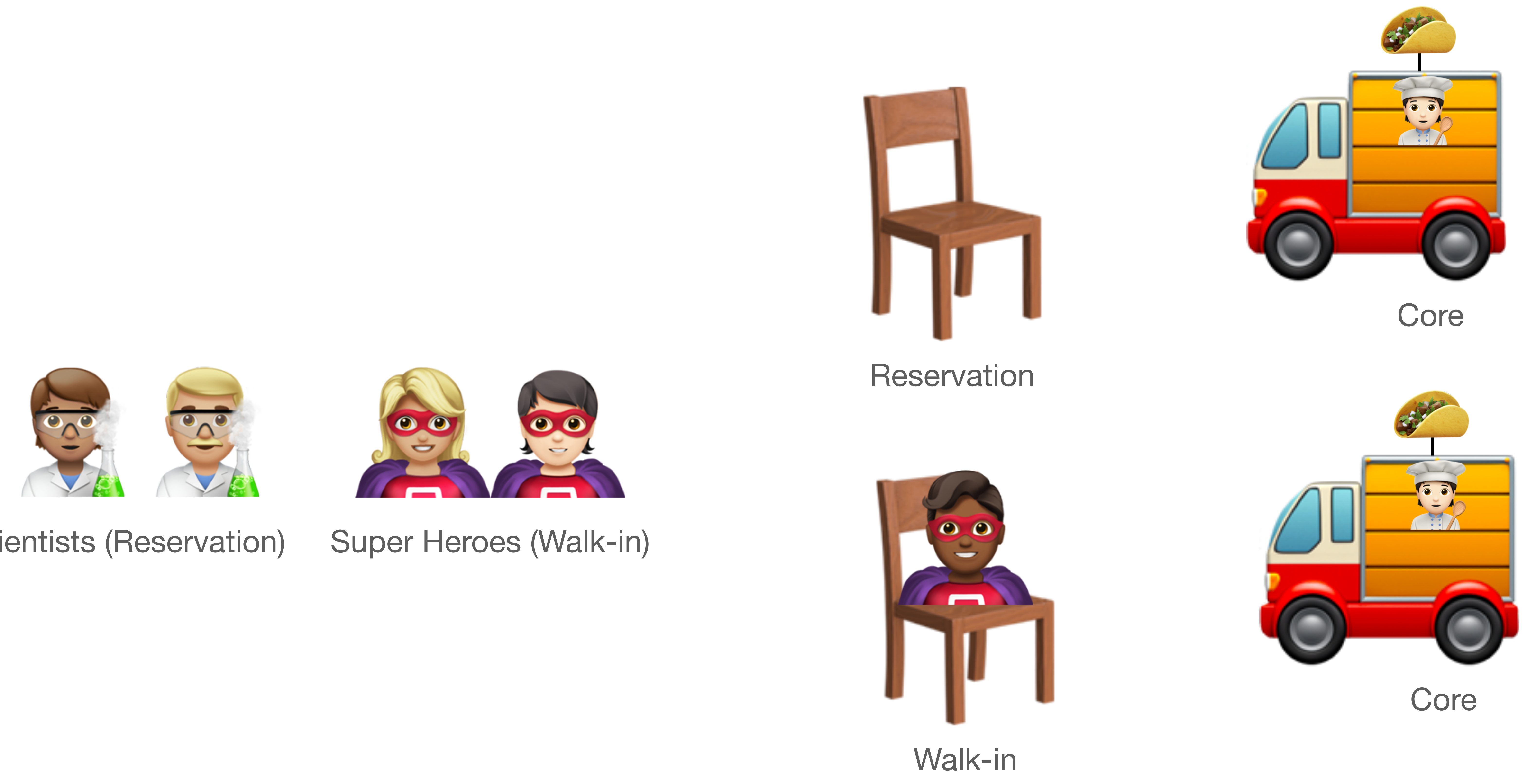
Asynchronous programming

First come, first serve



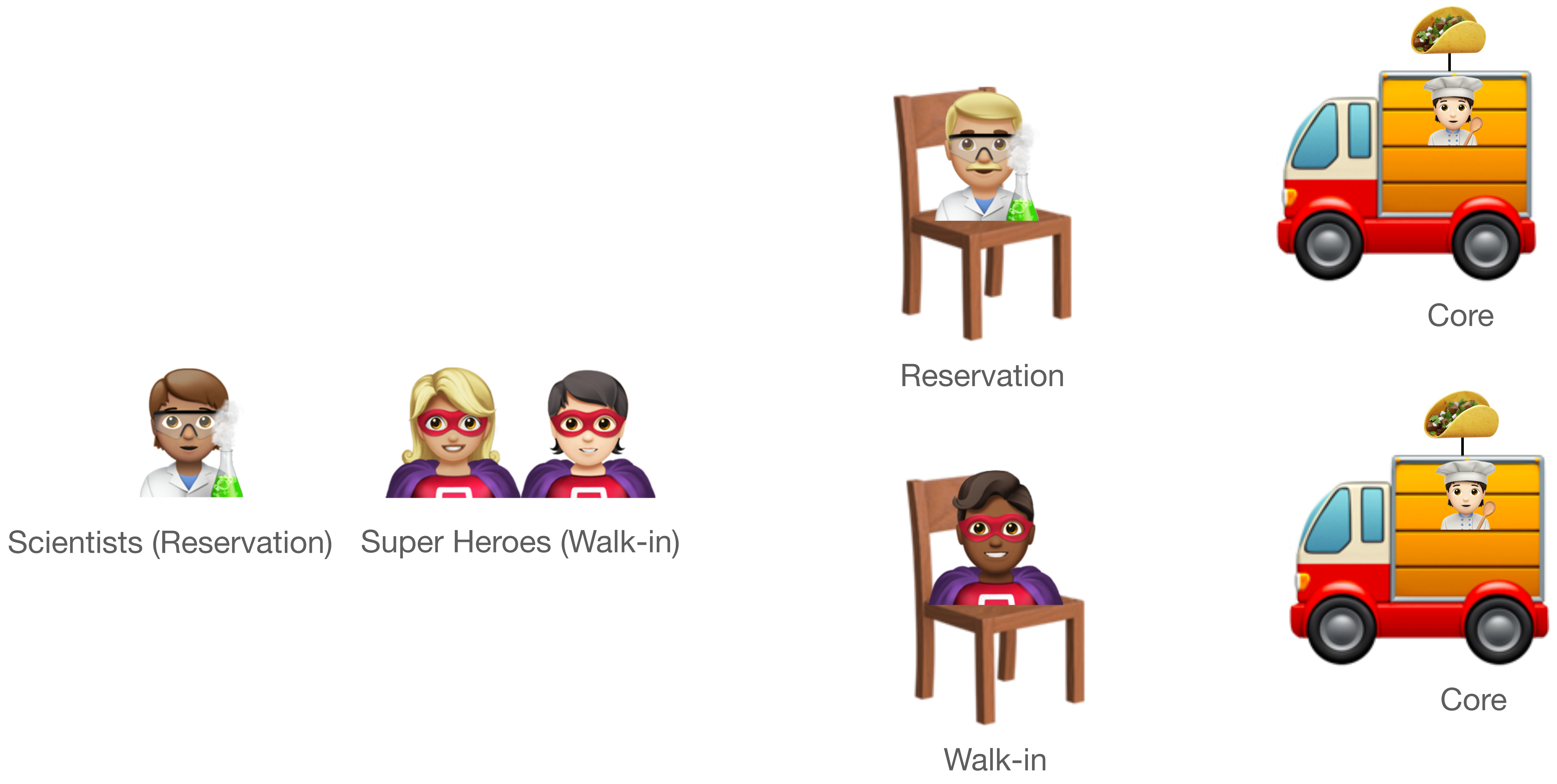
Asynchronous programming

First come, first serve



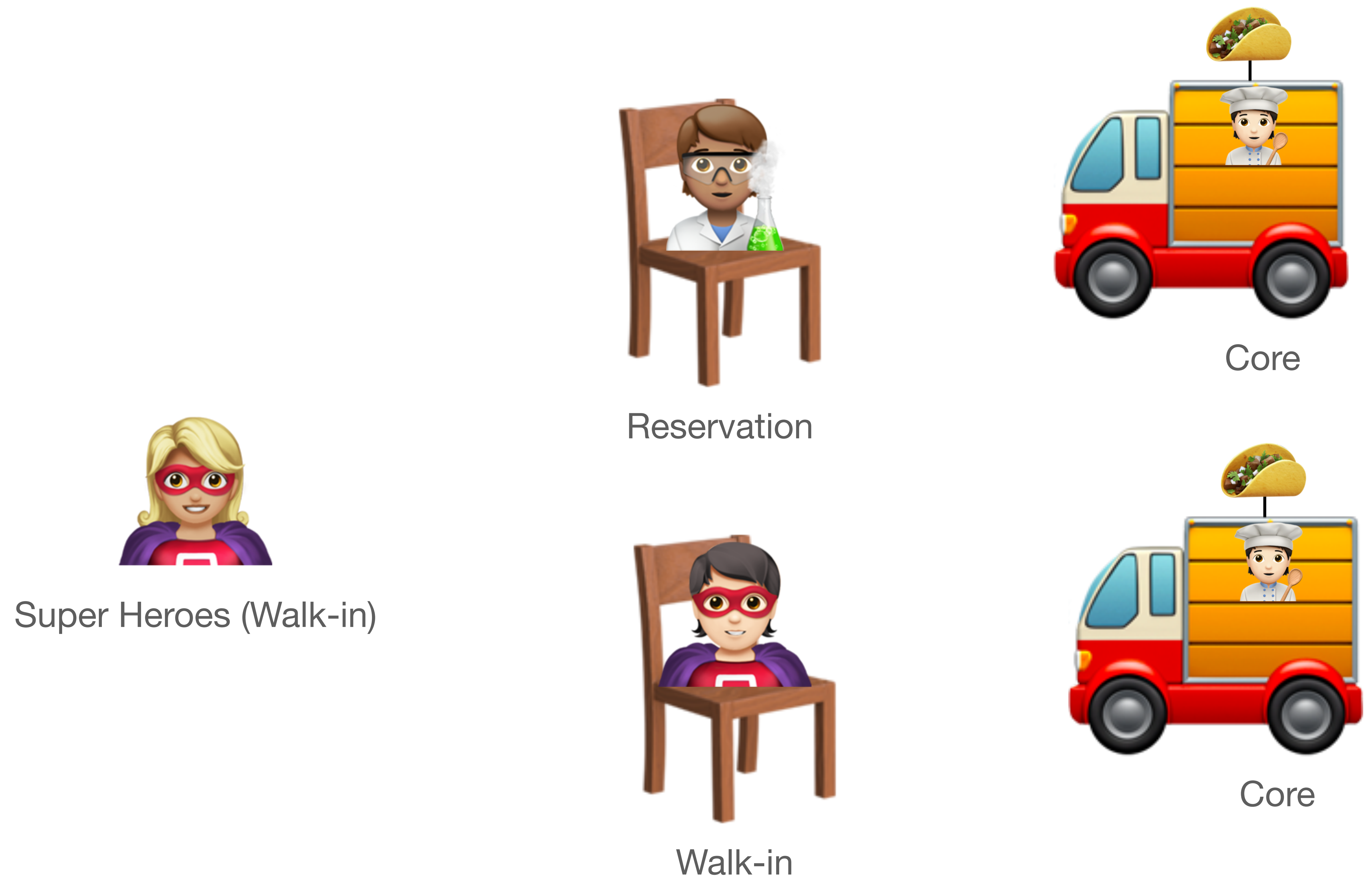
Asynchronous programming

First come, first serve



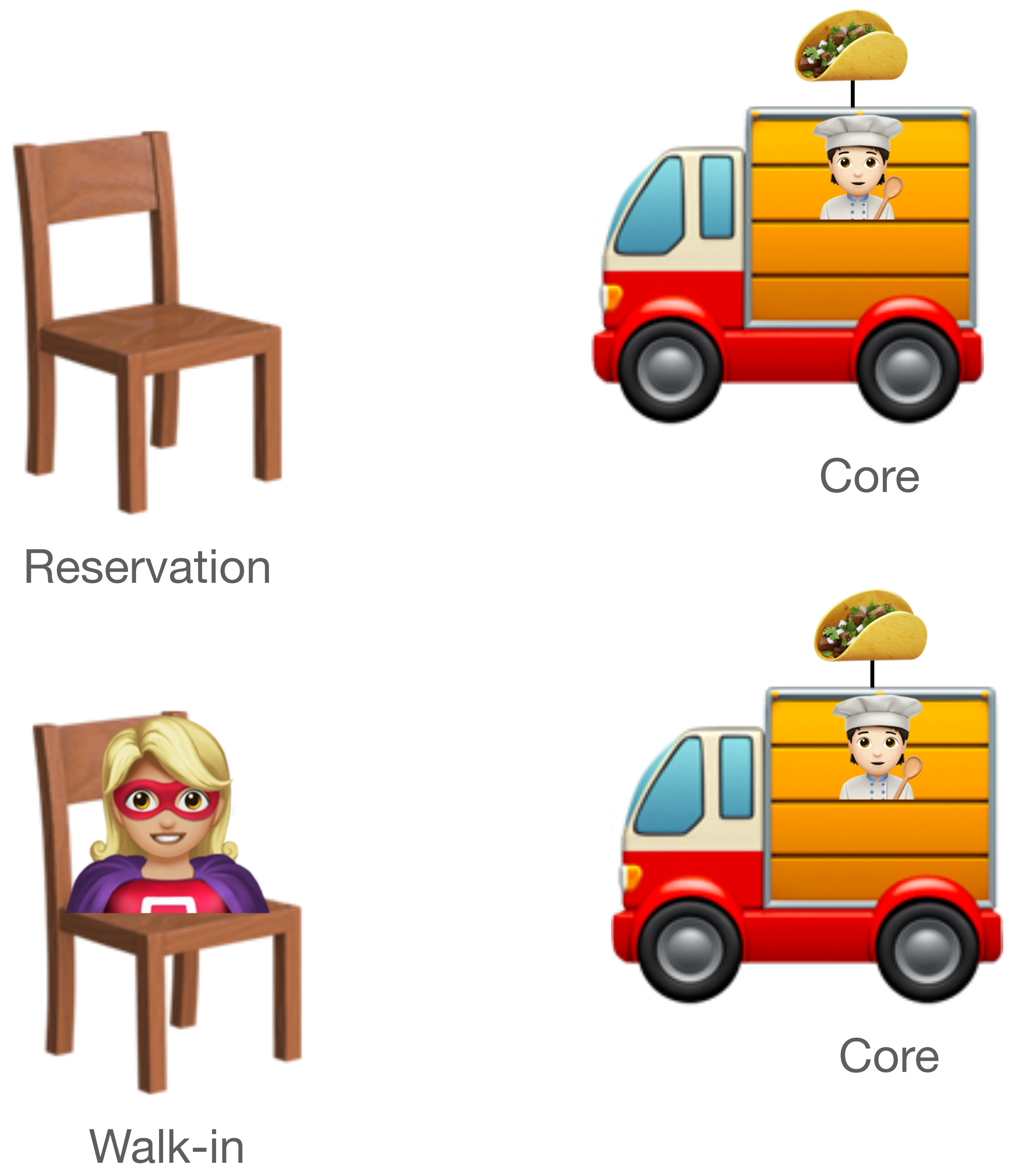
Asynchronous programming

First come, first serve



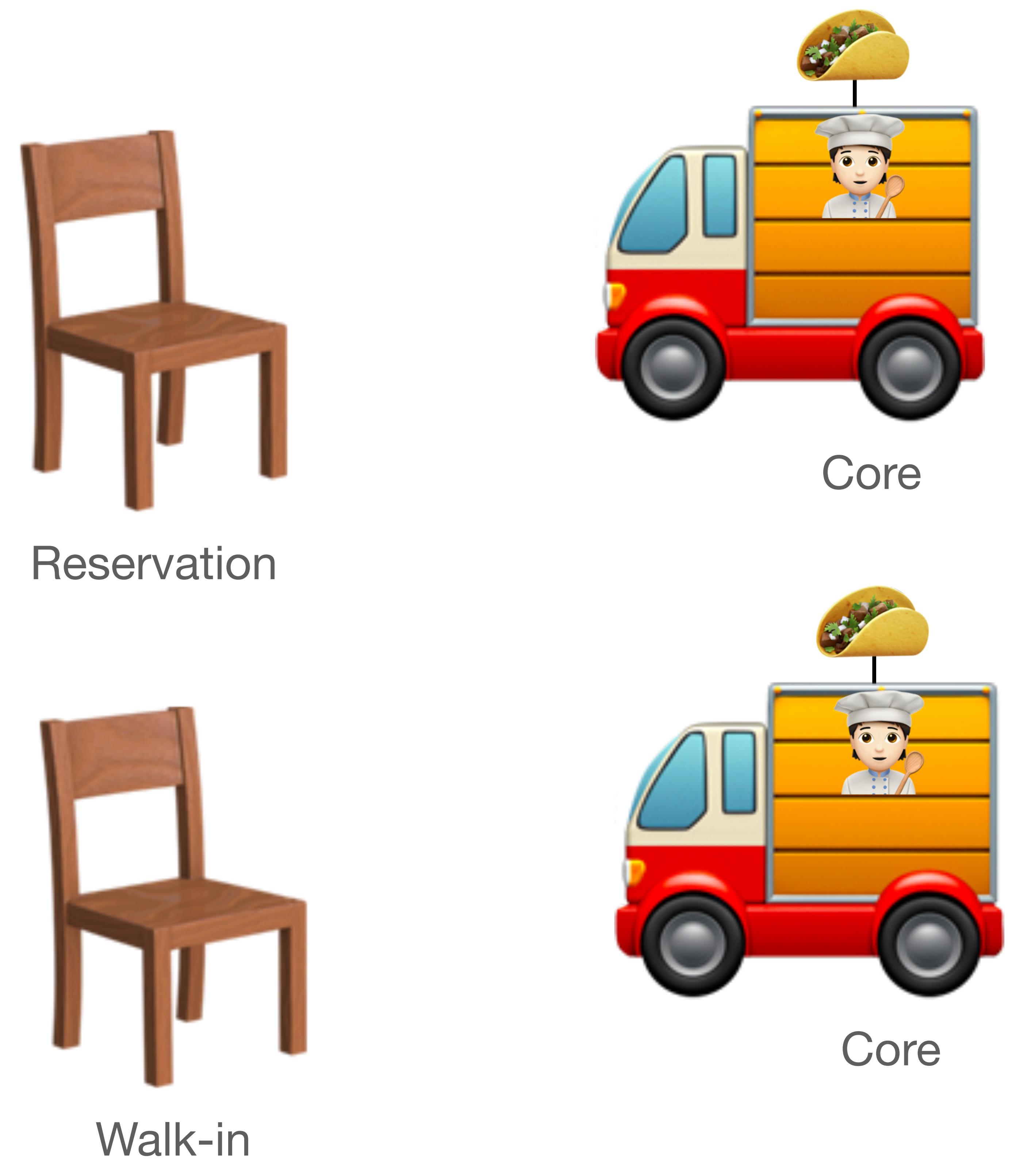
Asynchronous programming

First come, first serve



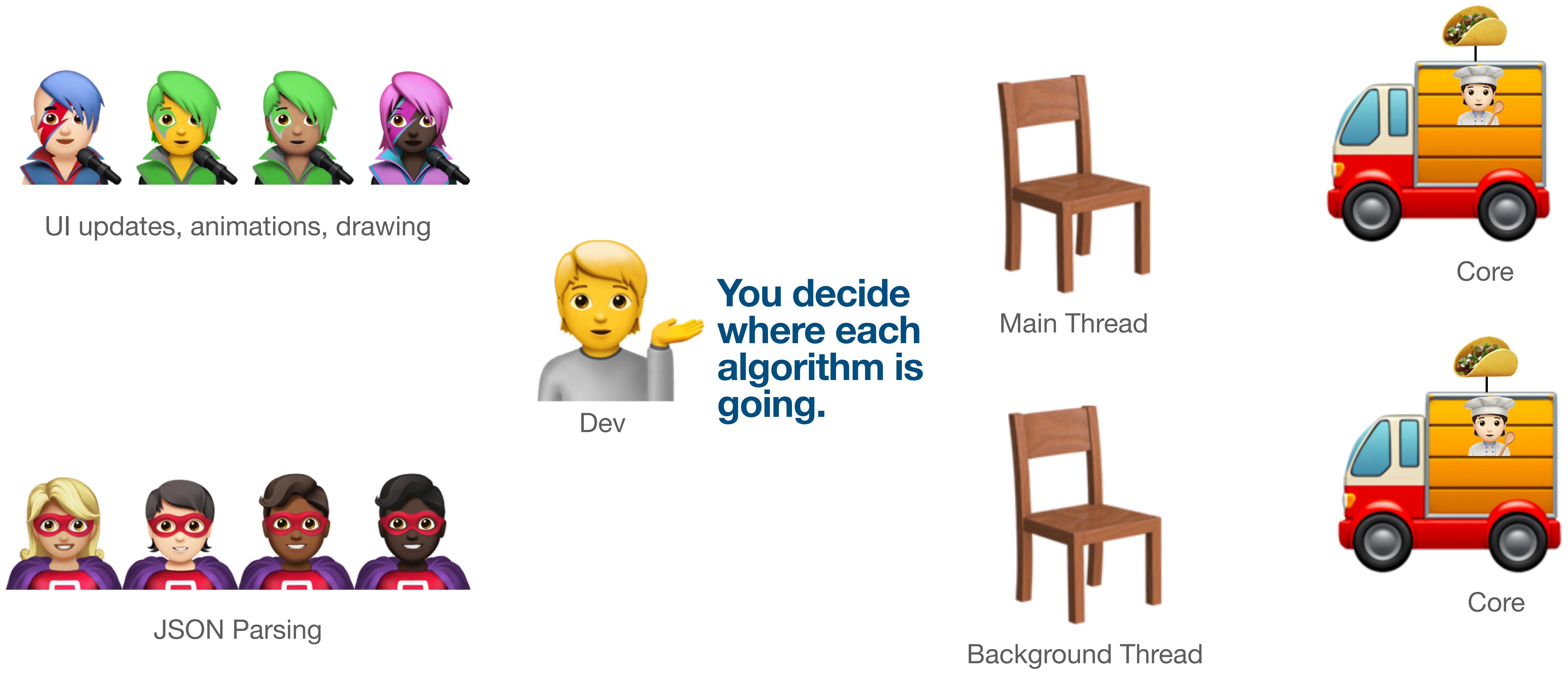
Asynchronous programming

All done.



Asynchronous programming

Threading today.



You decide where each algorithm is going.

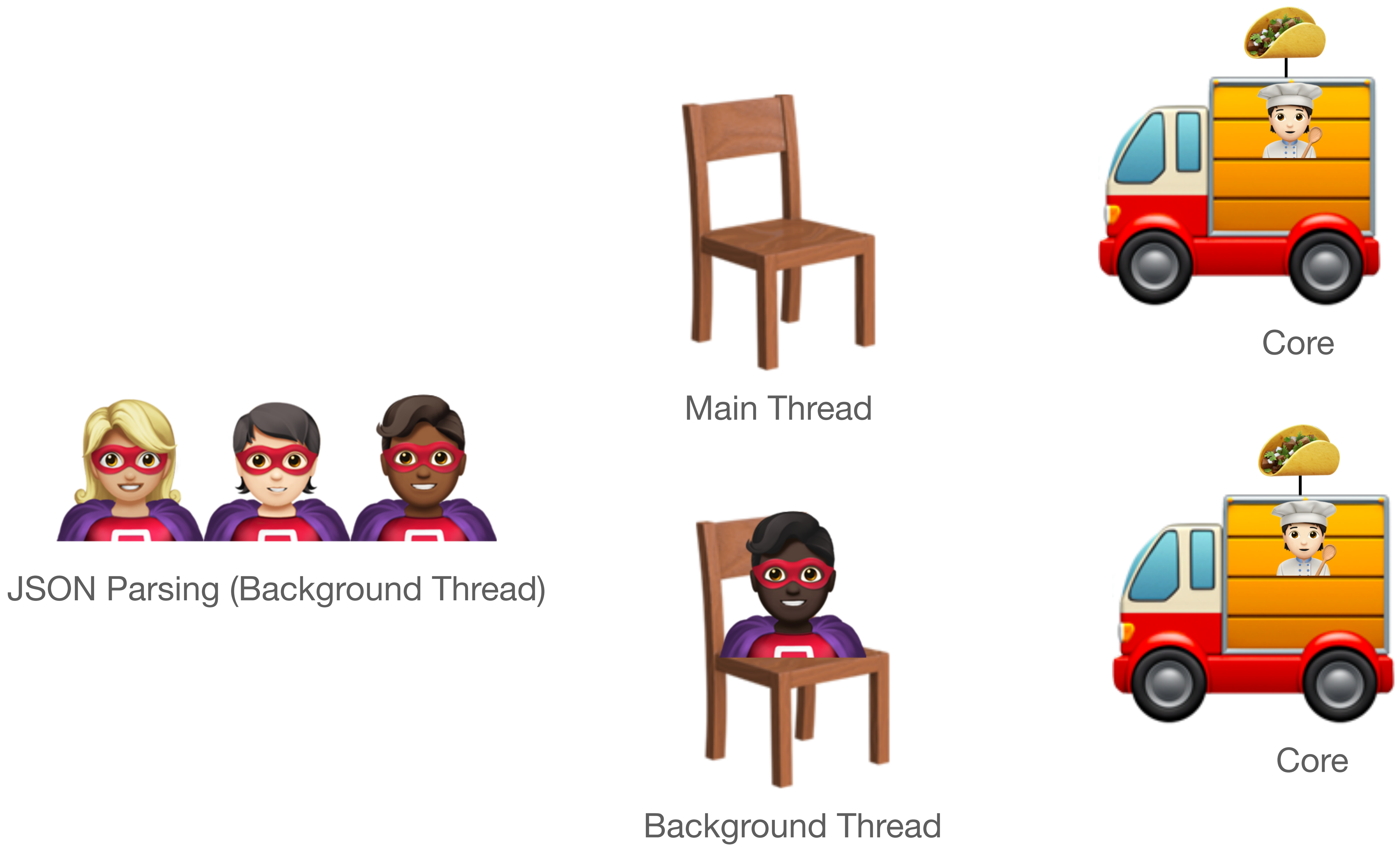
Asynchronous programming

The UI must always be updated on the main thread.

JSON Parsing (Background thread)
UI (Main Thread)
Main Thread
Background Thread
Core
Core

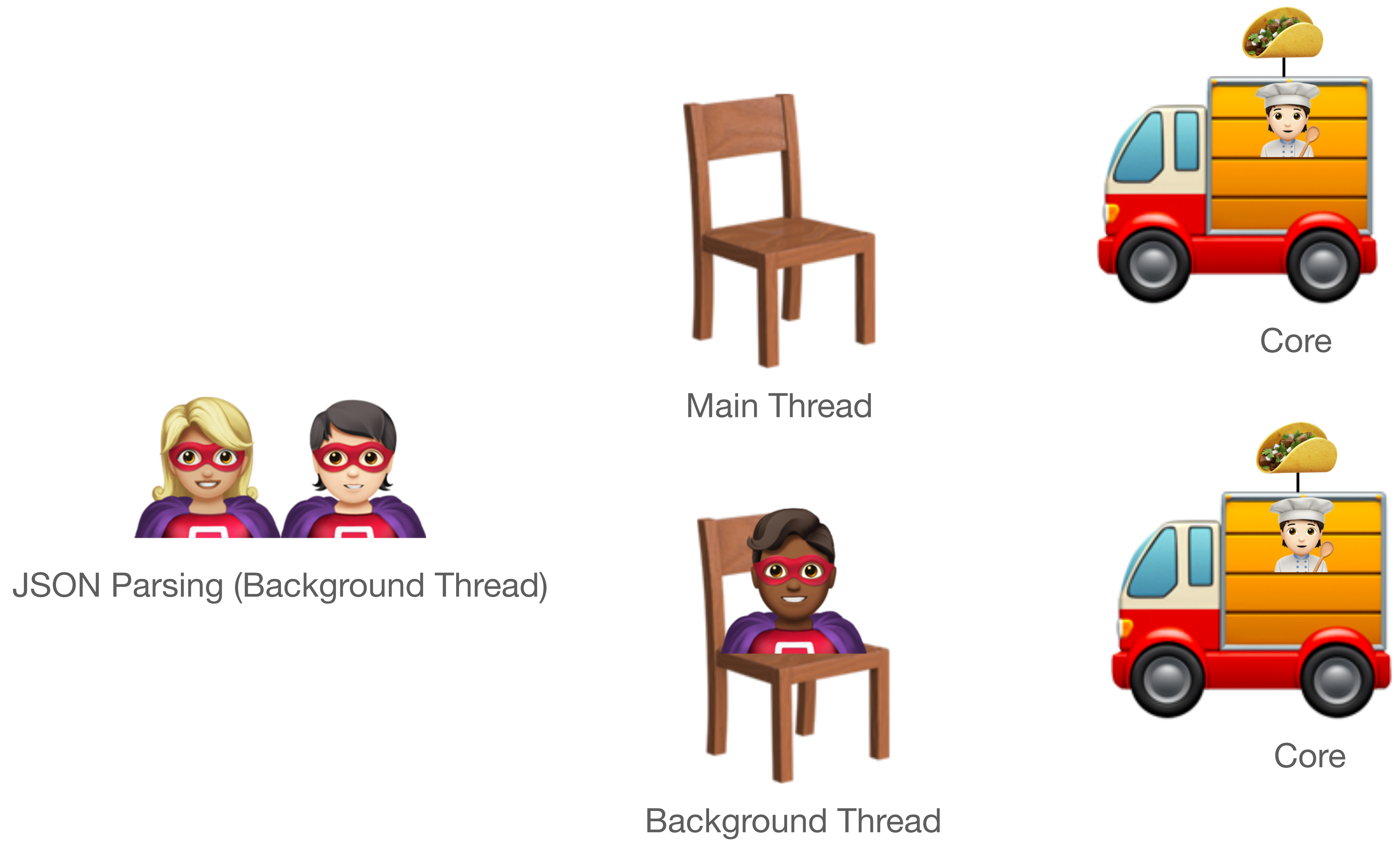
Asynchronous programming

The UI must always be updated on the main thread.



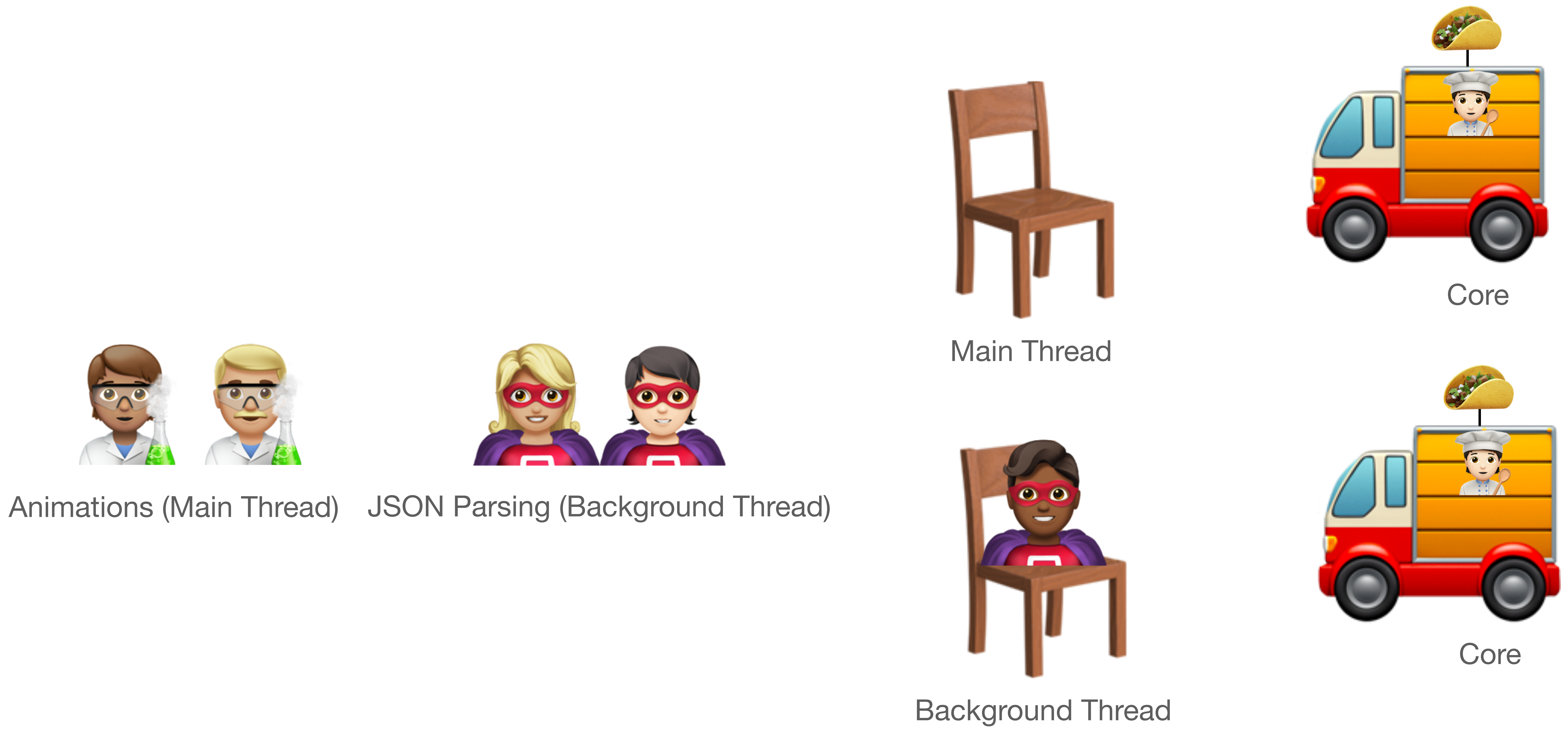
Asynchronous programming

The UI must always be updated on the main thread.



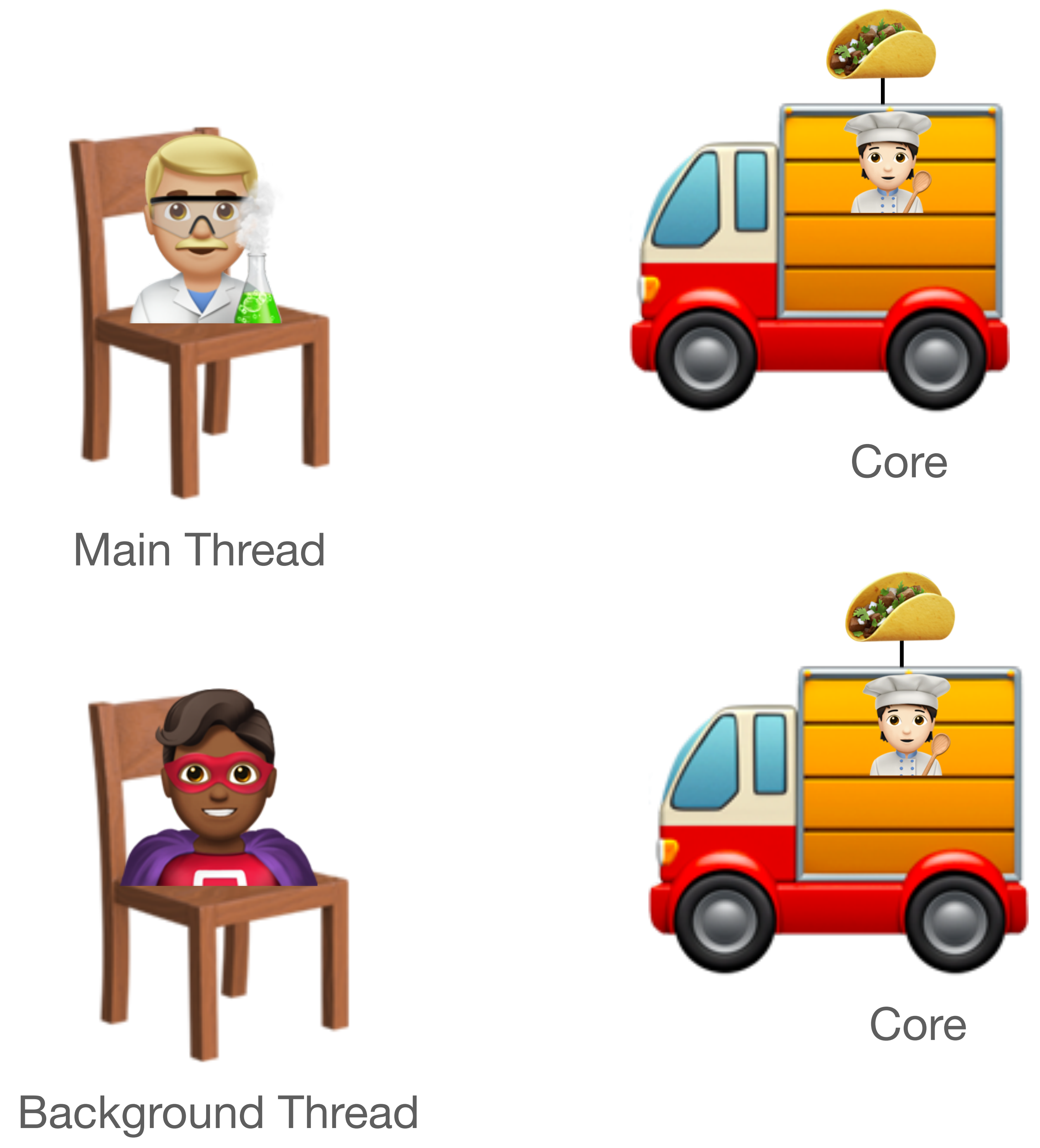
Asynchronous programming

The UI must always be updated on the main thread.



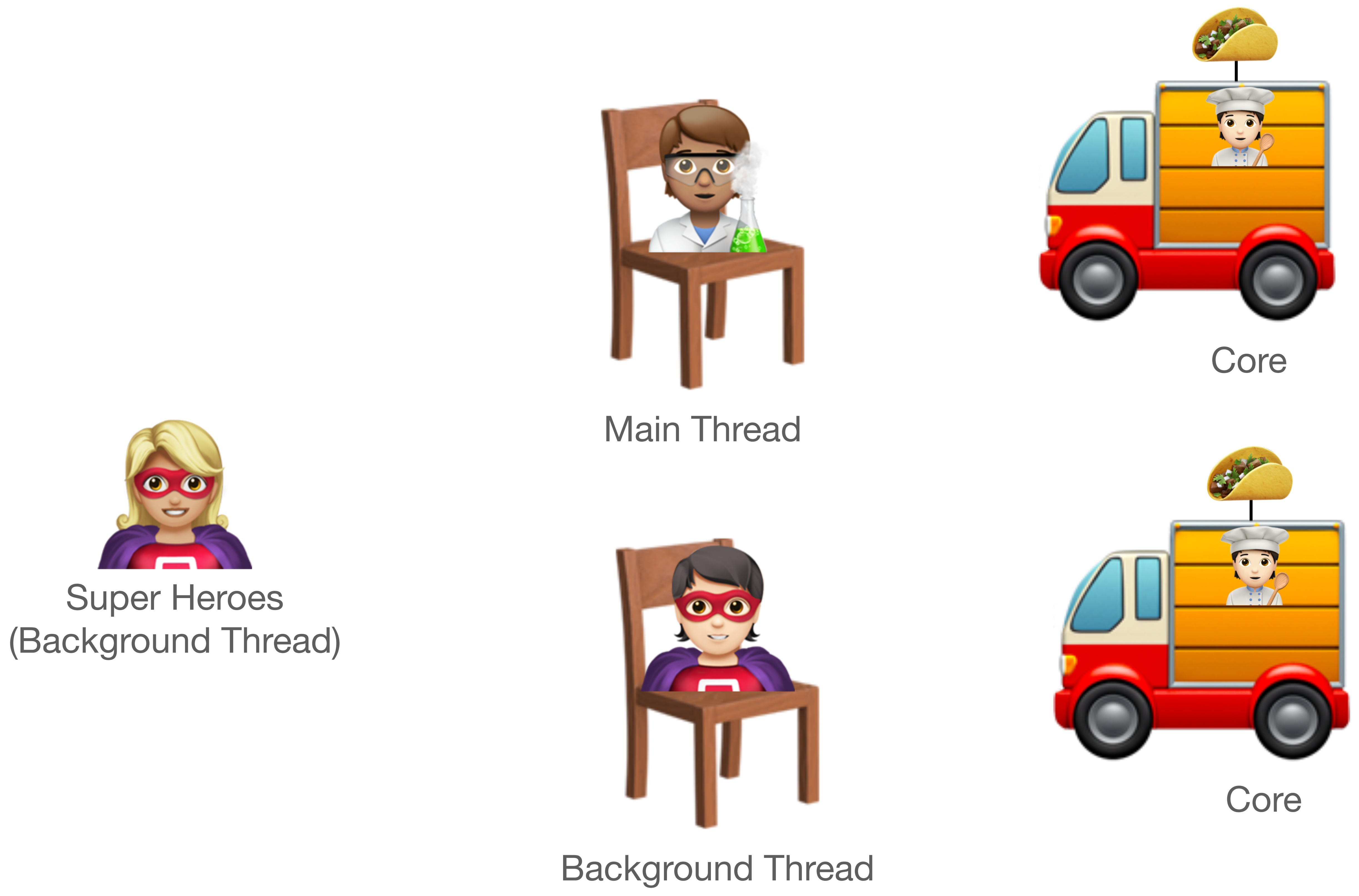
Asynchronous programming

The UI must always be updated on the main thread.



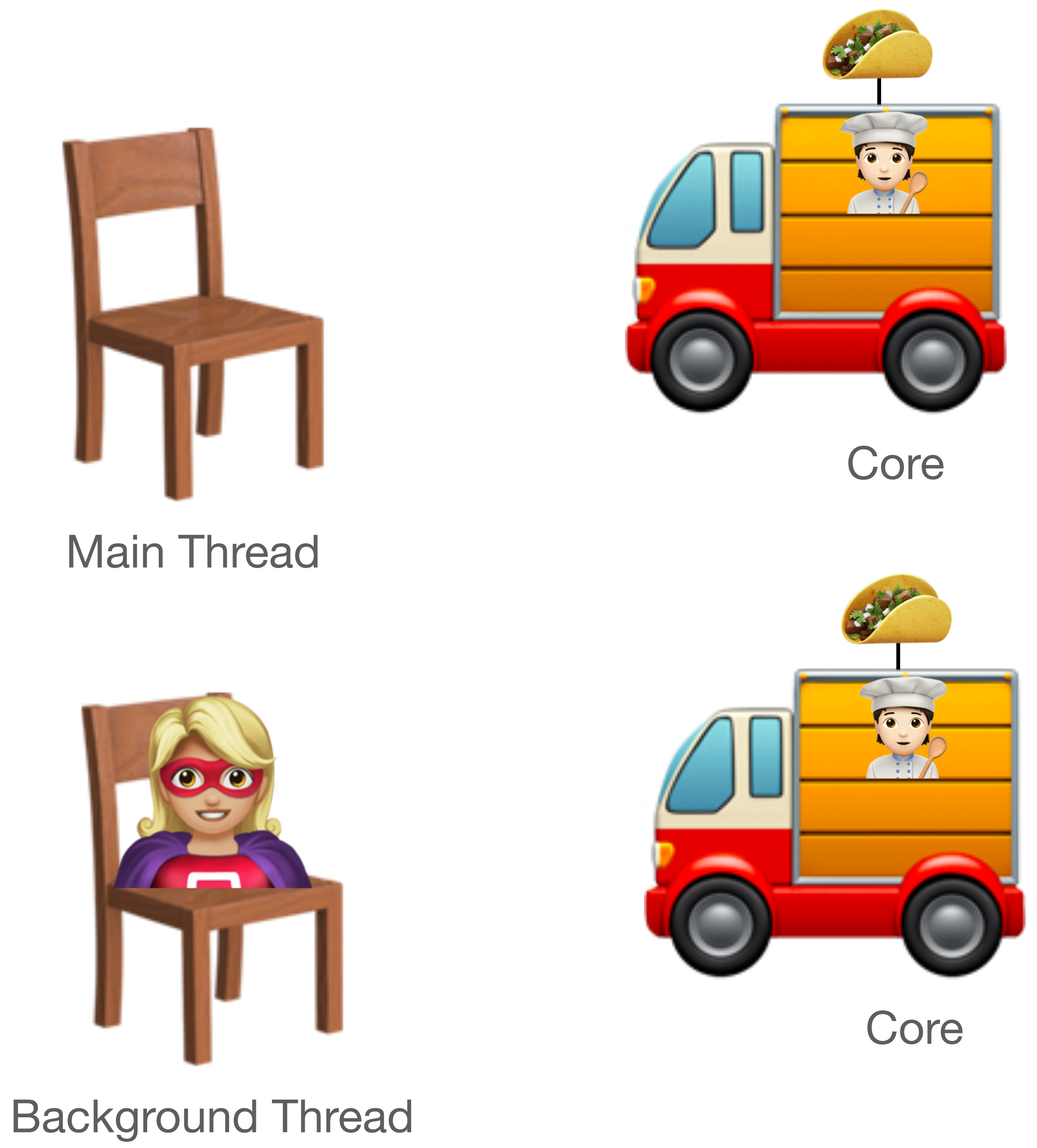
Asynchronous programming

The UI must always be updated on the main thread.



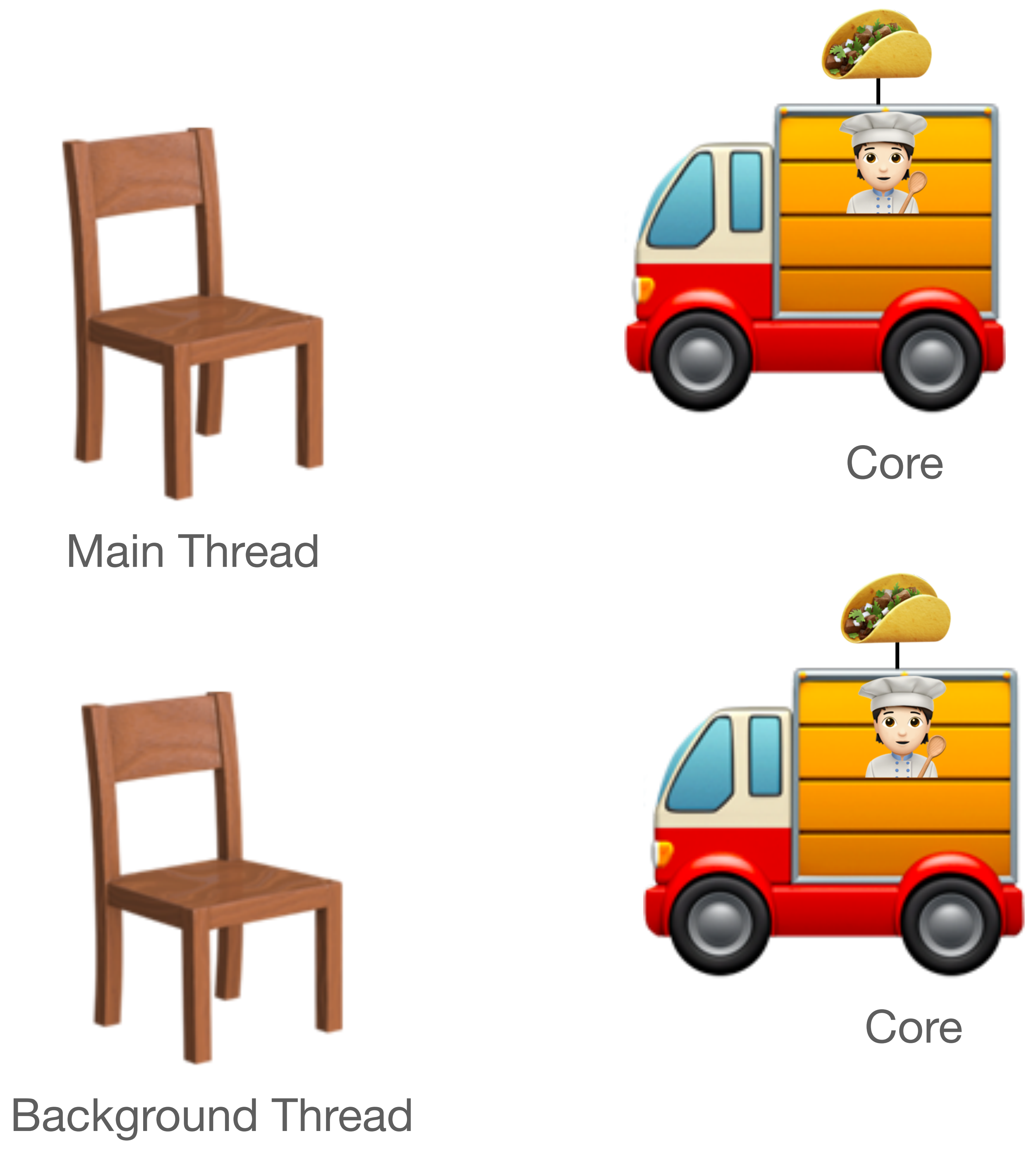
Asynchronous programming

The UI must always be updated on the main thread.



Asynchronous programming

The UI must always be updated on the main thread.



What is concurrency?

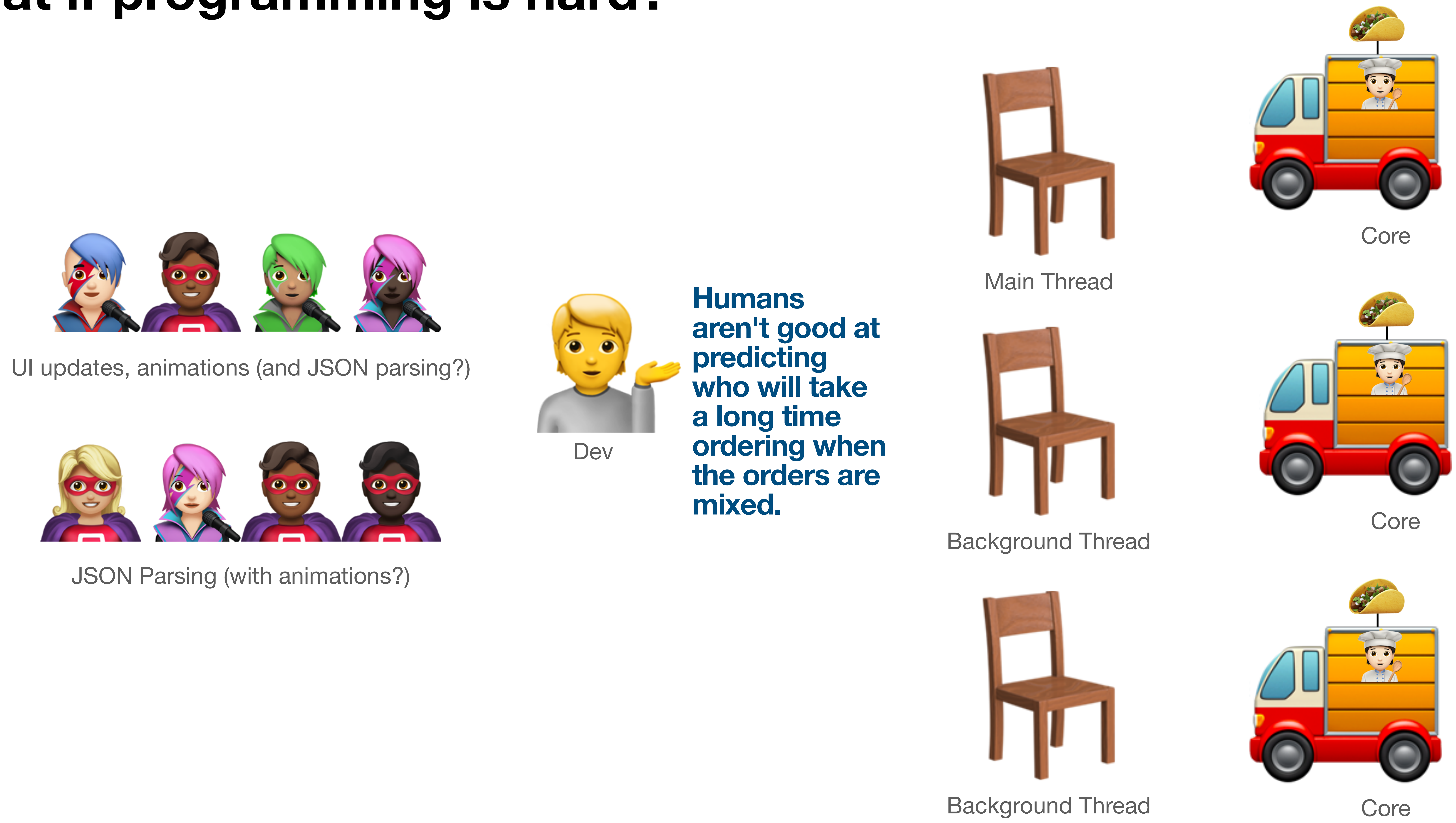
In practice, concurrency is introducing bugs without freezing the UI.

- Concurrency is very easy when you have tasks that are completely independent from each other (e.g. adding all numbers from 1-1000 in 2 threads).
- Concurrency is much more complicated when tasks are not completely independent (e.g. update one thread with info from another thread).
- New bugs to worry about: race conditions, deadlocking, livelocking, zombielocking, starvation, non-deterministic bugs.
- Concurrency should always be your last resort.

1 - https://en.wikipedia.org/wiki/Concurrency_(computer_science)

Asynchronous programming

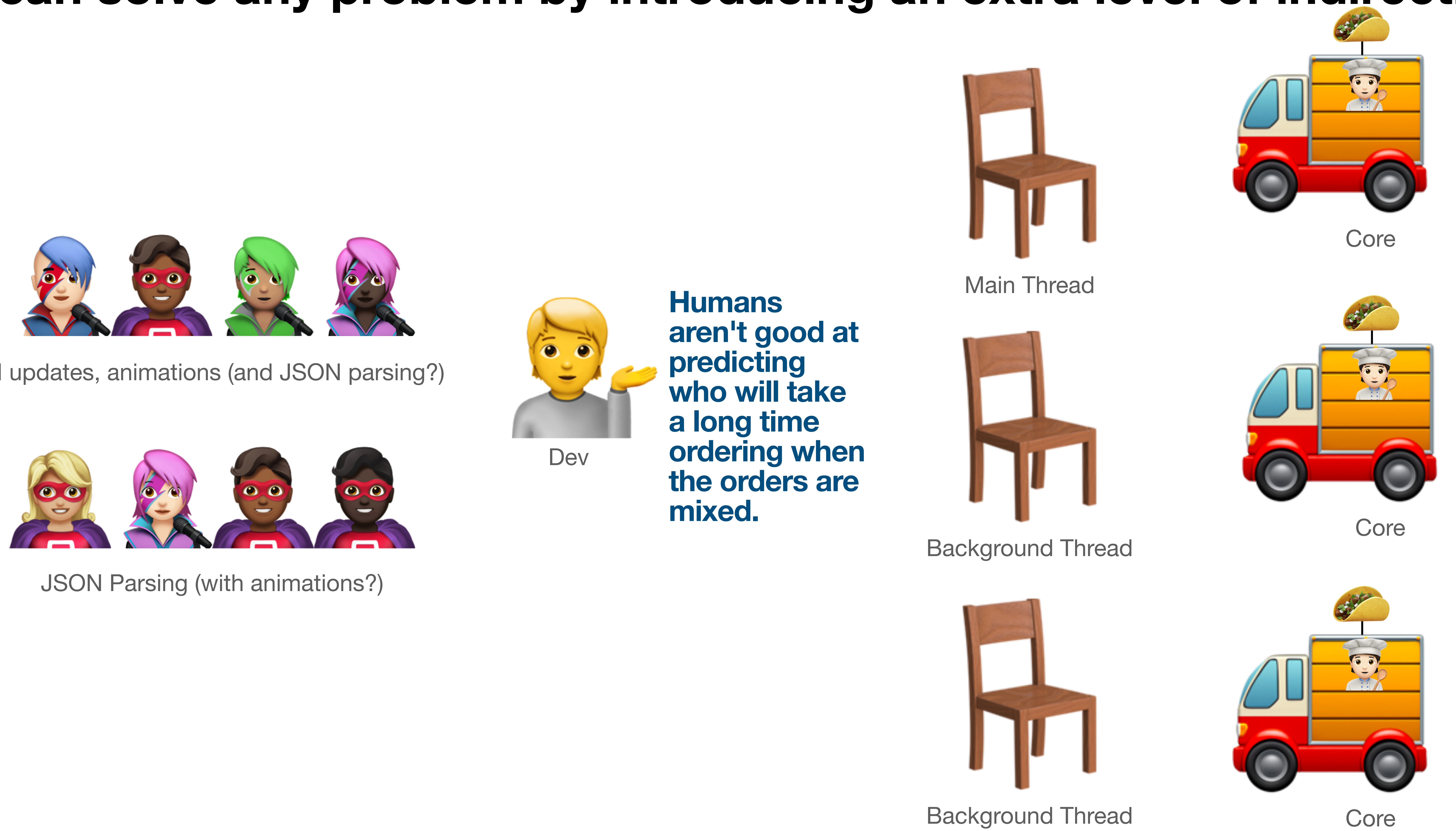
What if programming is hard?


UI updates, animations (and JSON parsing?)
JSON Parsing (with animations?)
Dev
Main Thread
Background Thread
Background Thread
Core
Core
Core

Humans aren't good at predicting who will take a long time ordering when the orders are mixed.

Asynchronous programming

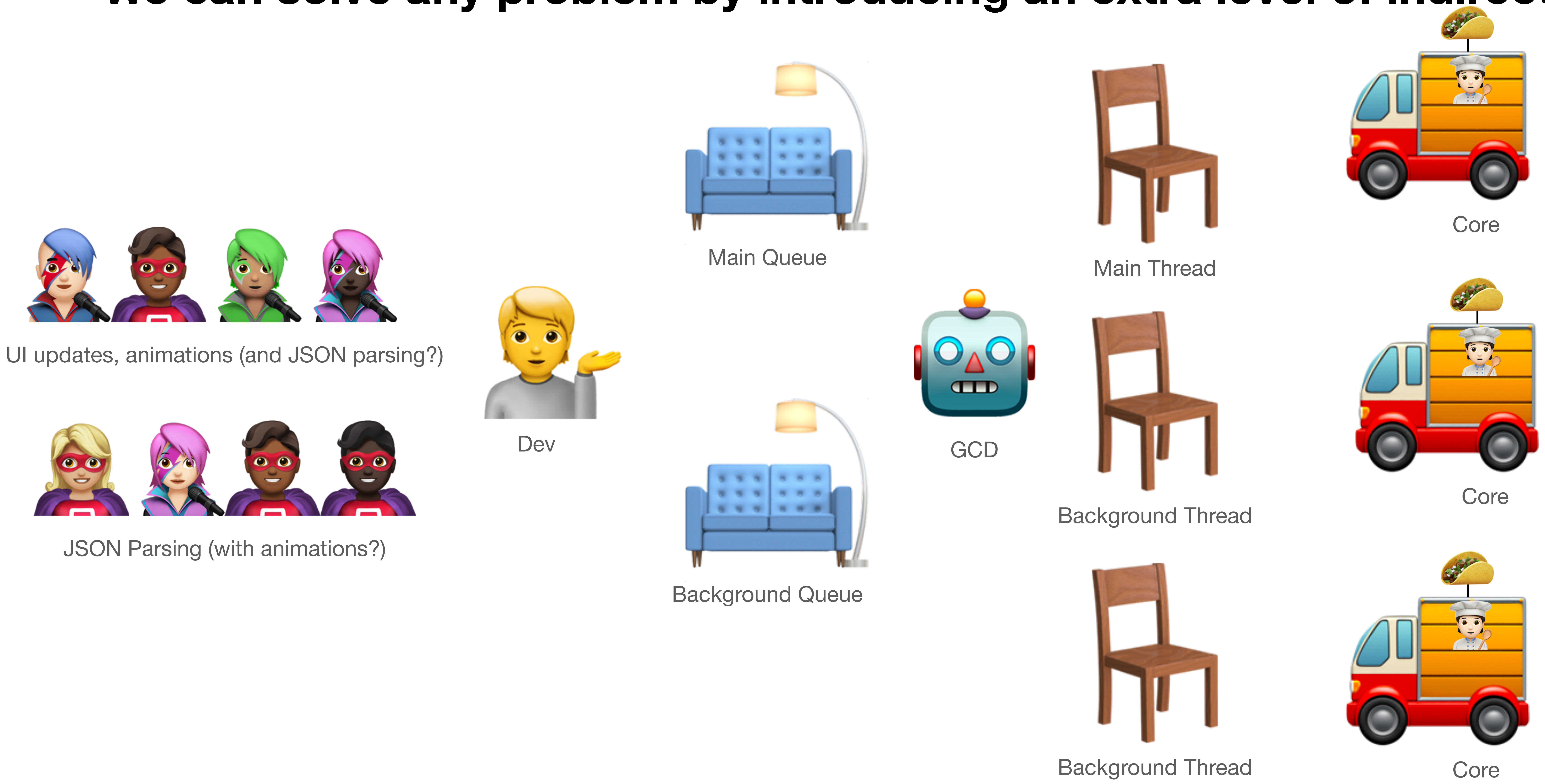
"We can solve any problem by introducing an extra level of indirection."


UI updates, animations (and JSON parsing?)
JSON Parsing (with animations?)
Dev
Main Thread
Background Thread
Background Thread
Core
Core
Core

Humans aren't good at predicting who will take a long time ordering when the orders are mixed.

Asynchronous programming

"We can solve any problem by introducing an extra level of indirection."



What is GCD?

In theory, it's easy concurrency

- Official definition: "provides comprehensive support for concurrent code execution on multicore hardware."[1]
- Handles tasks (i.e. closures) and passes them along to queues that execute them in synchronous or asynchronous order.
- Queues can be serial or concurrent.
- Used extensively for long-running tasks that would block the main queue (responsible for drawing).

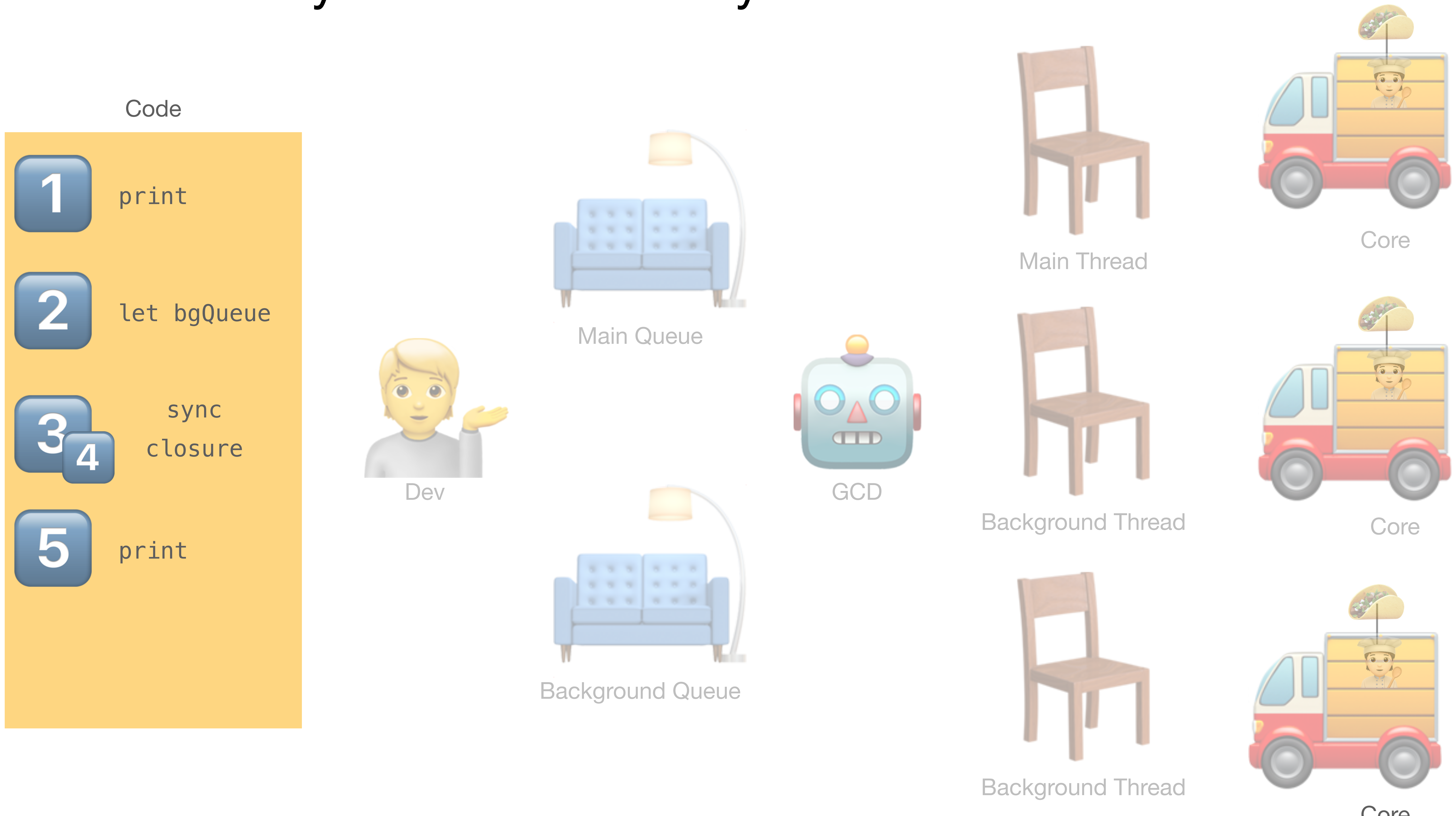
[1] - https://apple.github.io/swift-corelibs-libdispatch/

Live Demo

The first horseman: Sync

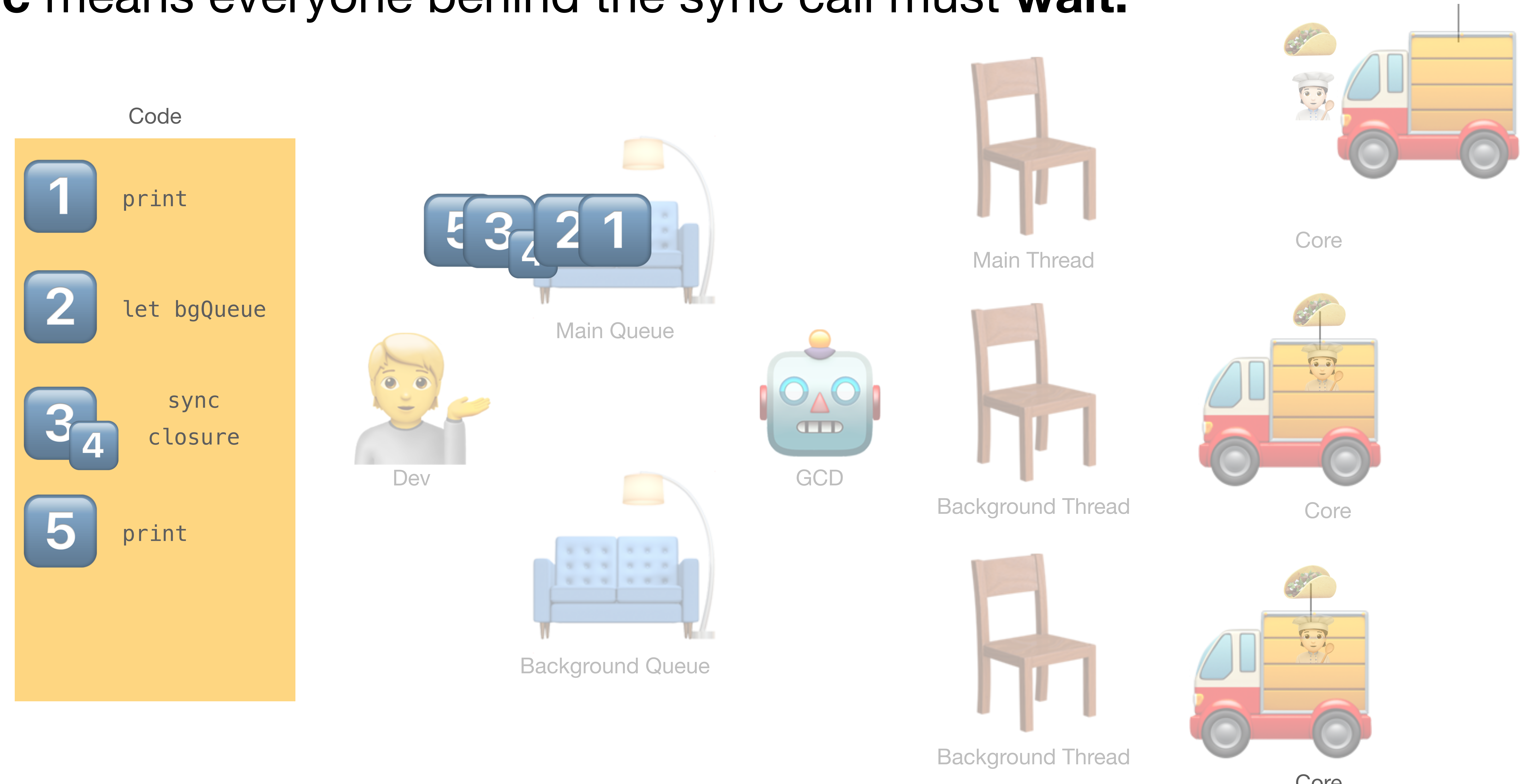
The Rules of GCD

Sync means everyone behind the sync call must **wait.**



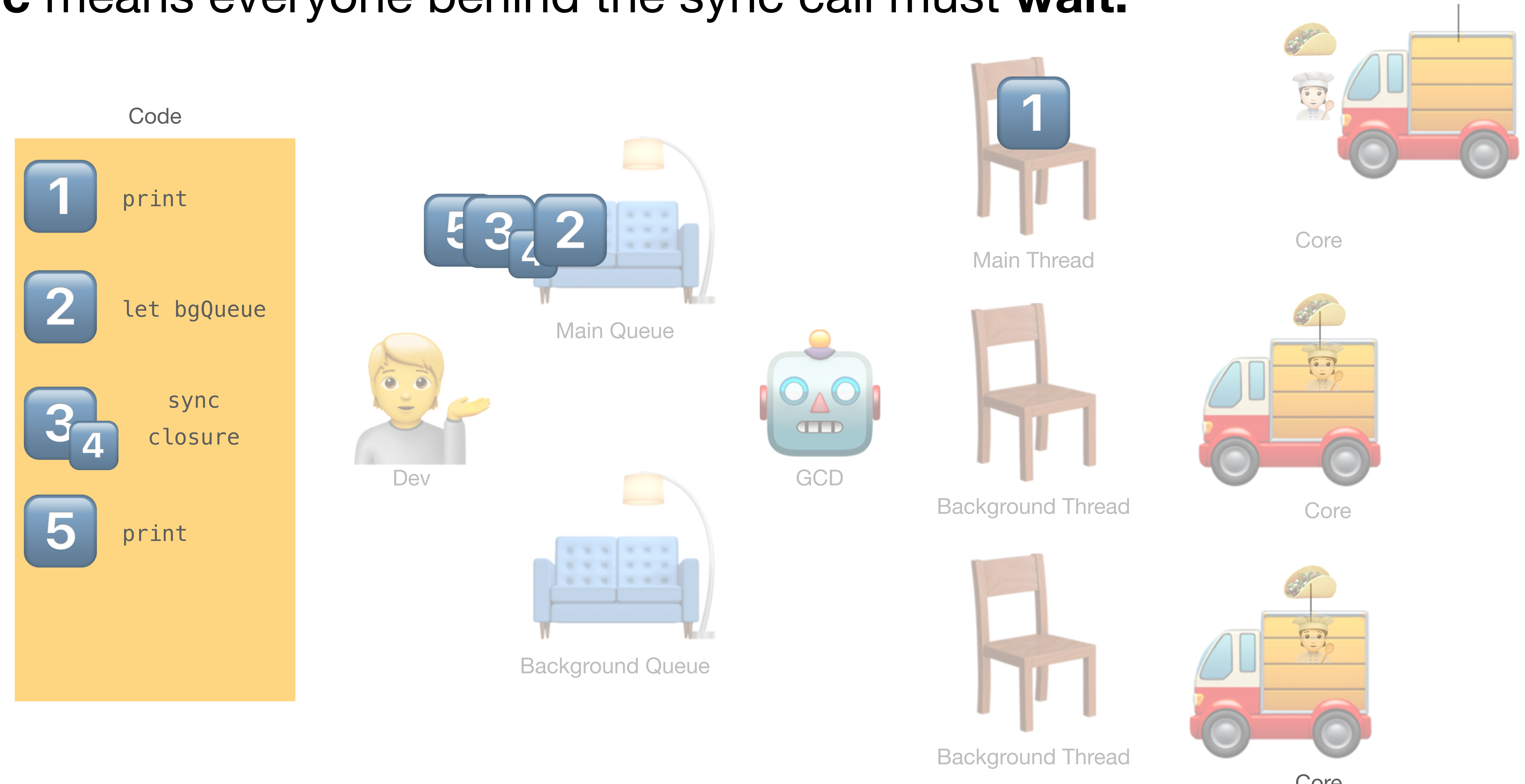
The Rules of GCD

Sync means everyone behind the sync call must **wait.**



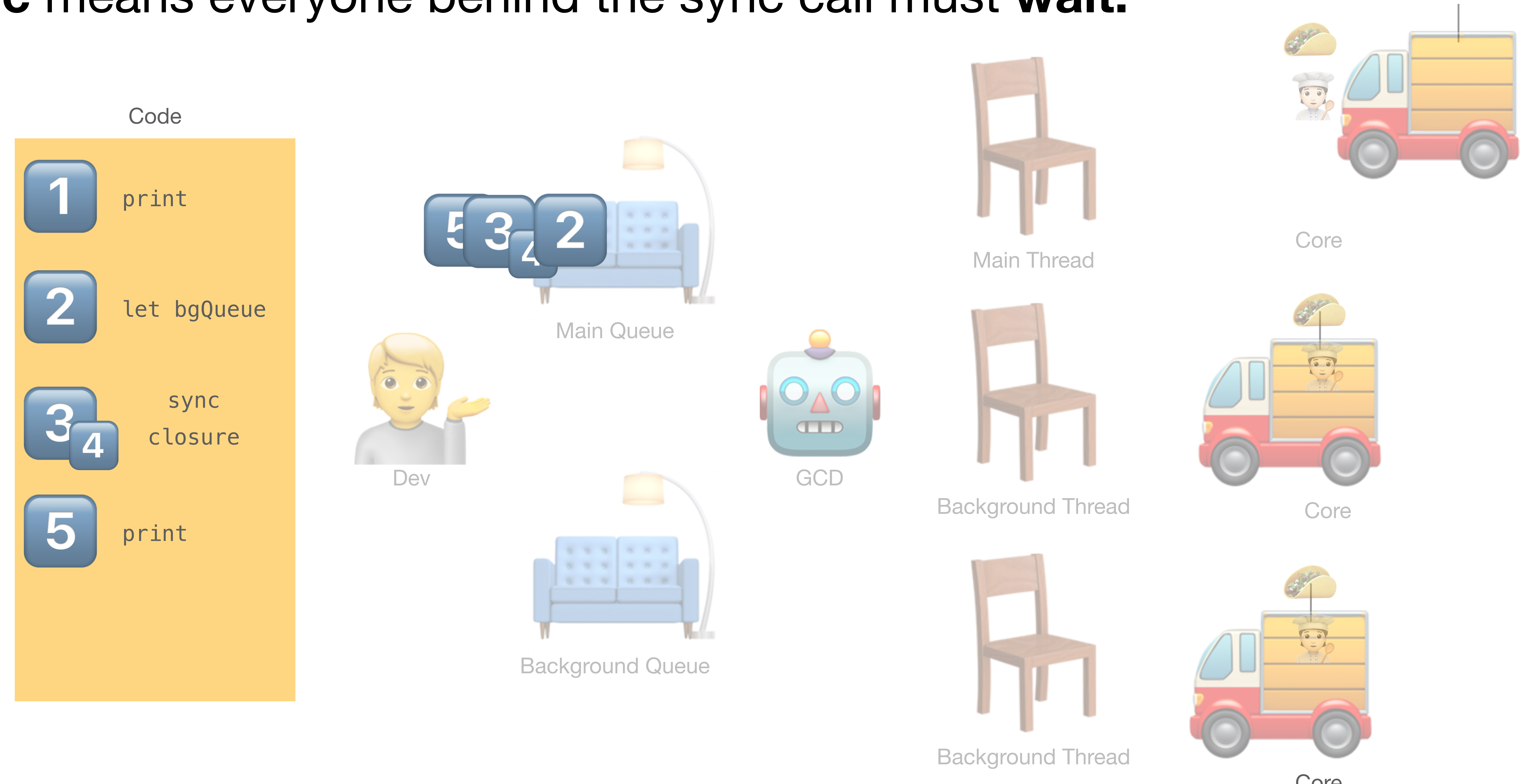
The Rules of GCD

Sync means everyone behind the sync call must **wait.**



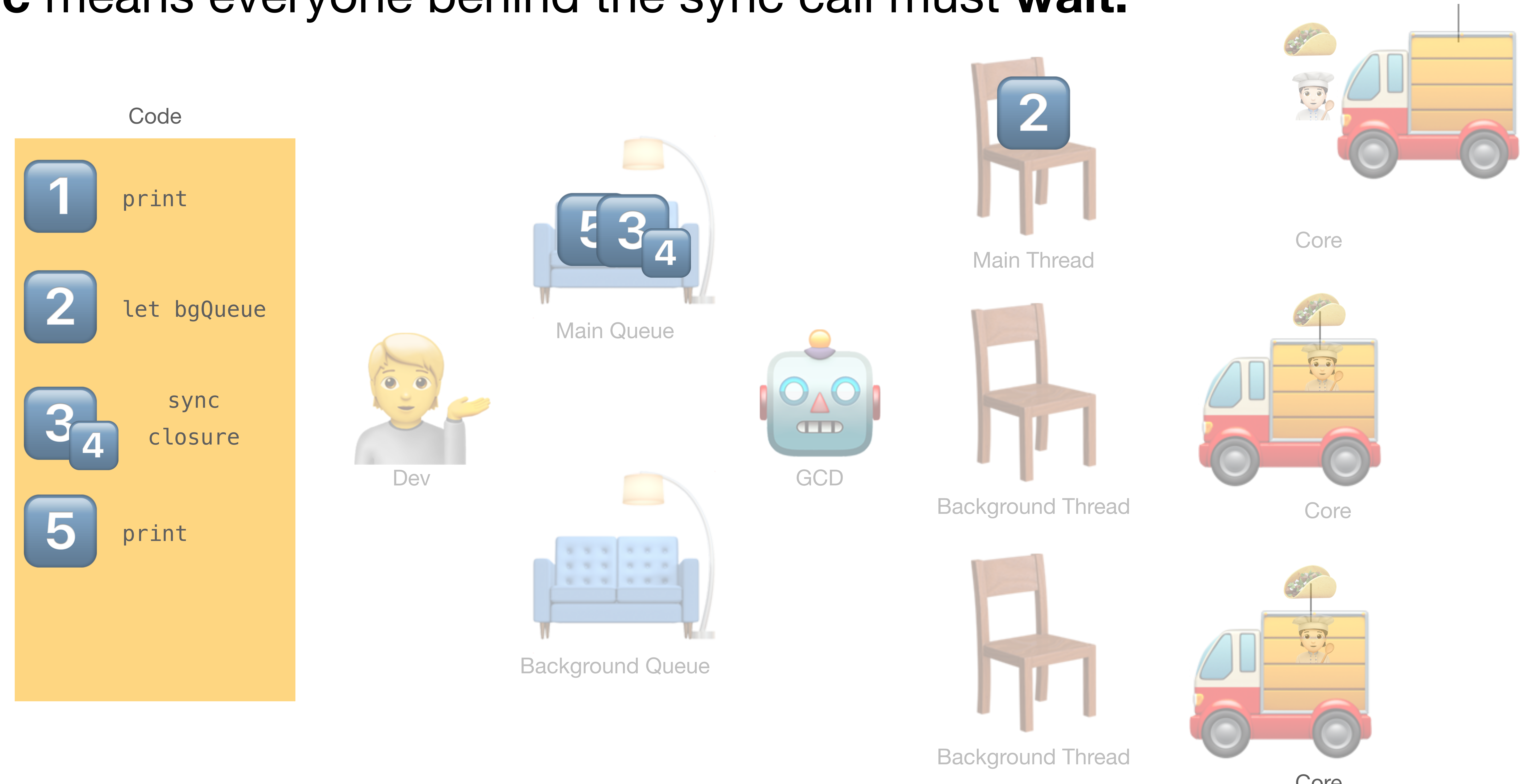
The Rules of GCD

Sync means everyone behind the sync call must **wait.**



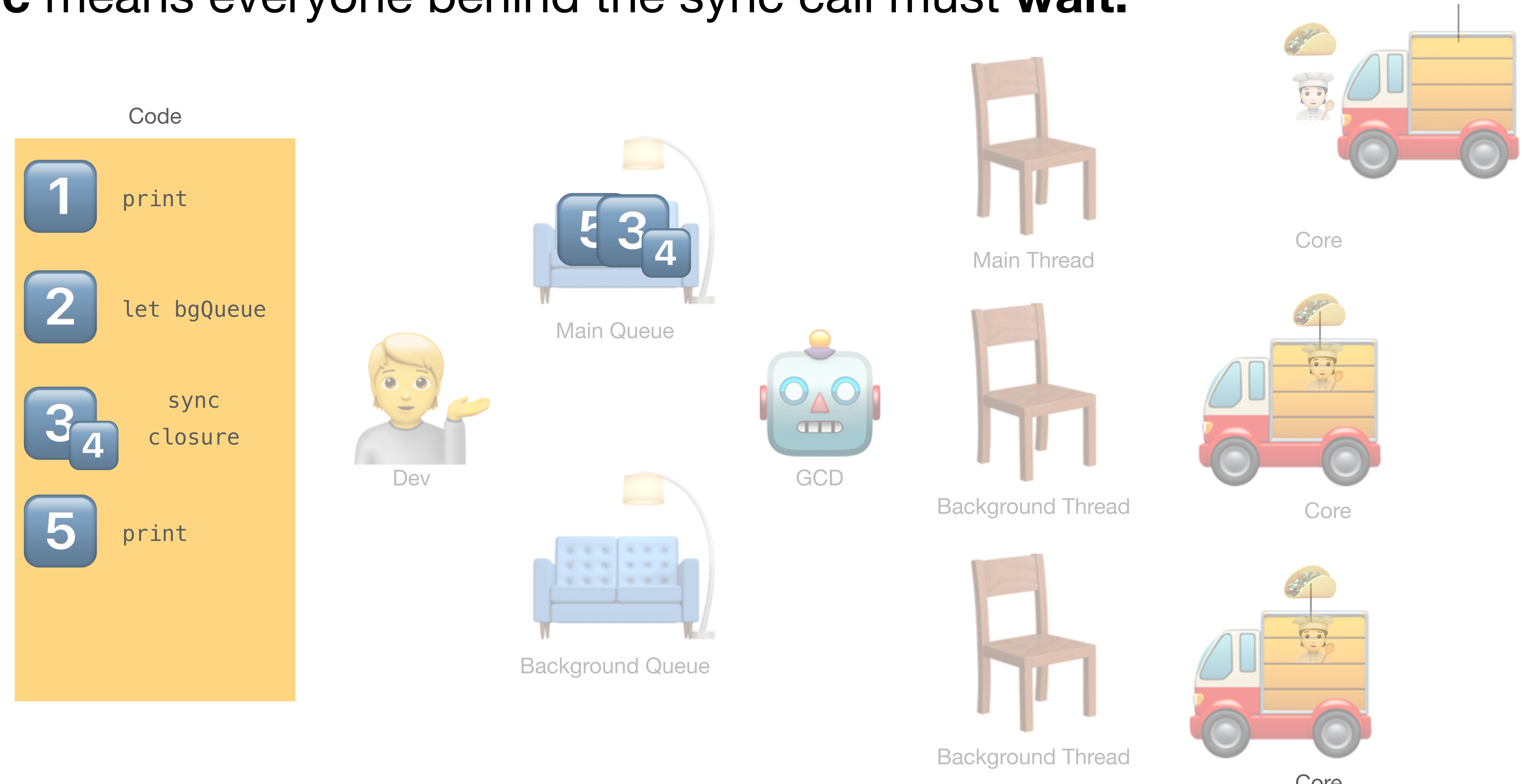
The Rules of GCD

Sync means everyone behind the sync call must **wait.**



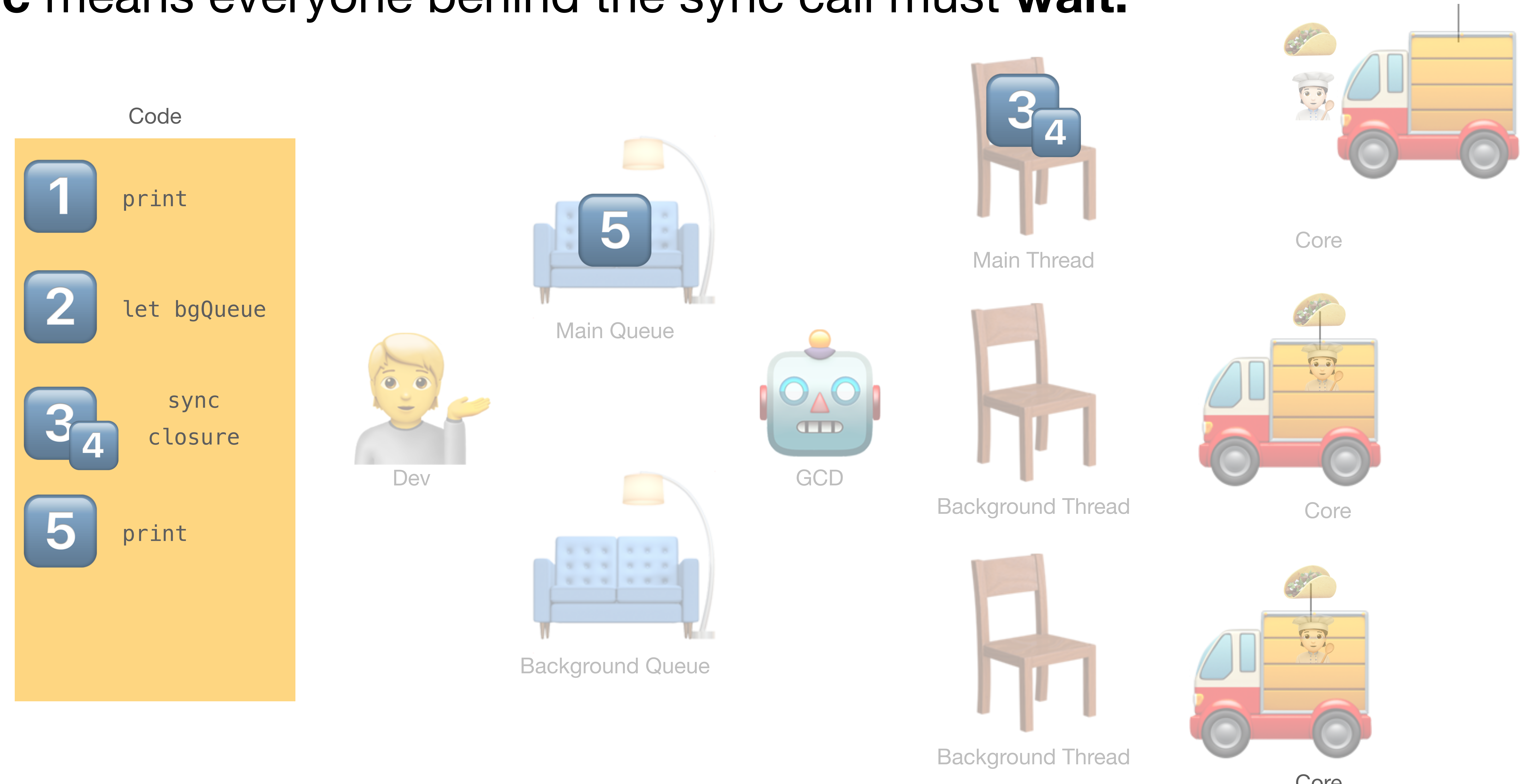
The Rules of GCD

Sync means everyone behind the sync call must **wait.**



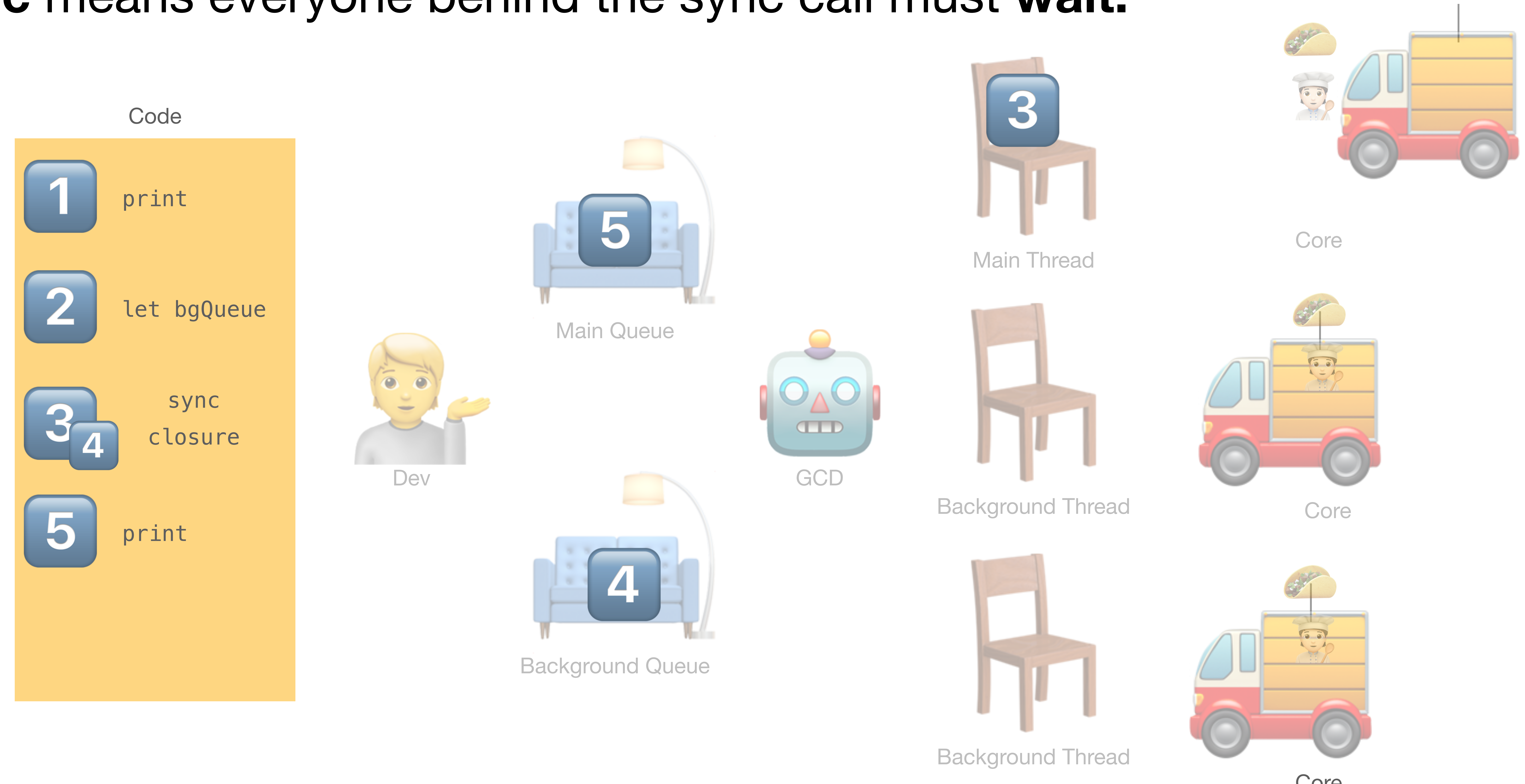
The Rules of GCD

Sync means everyone behind the sync call must **wait.**



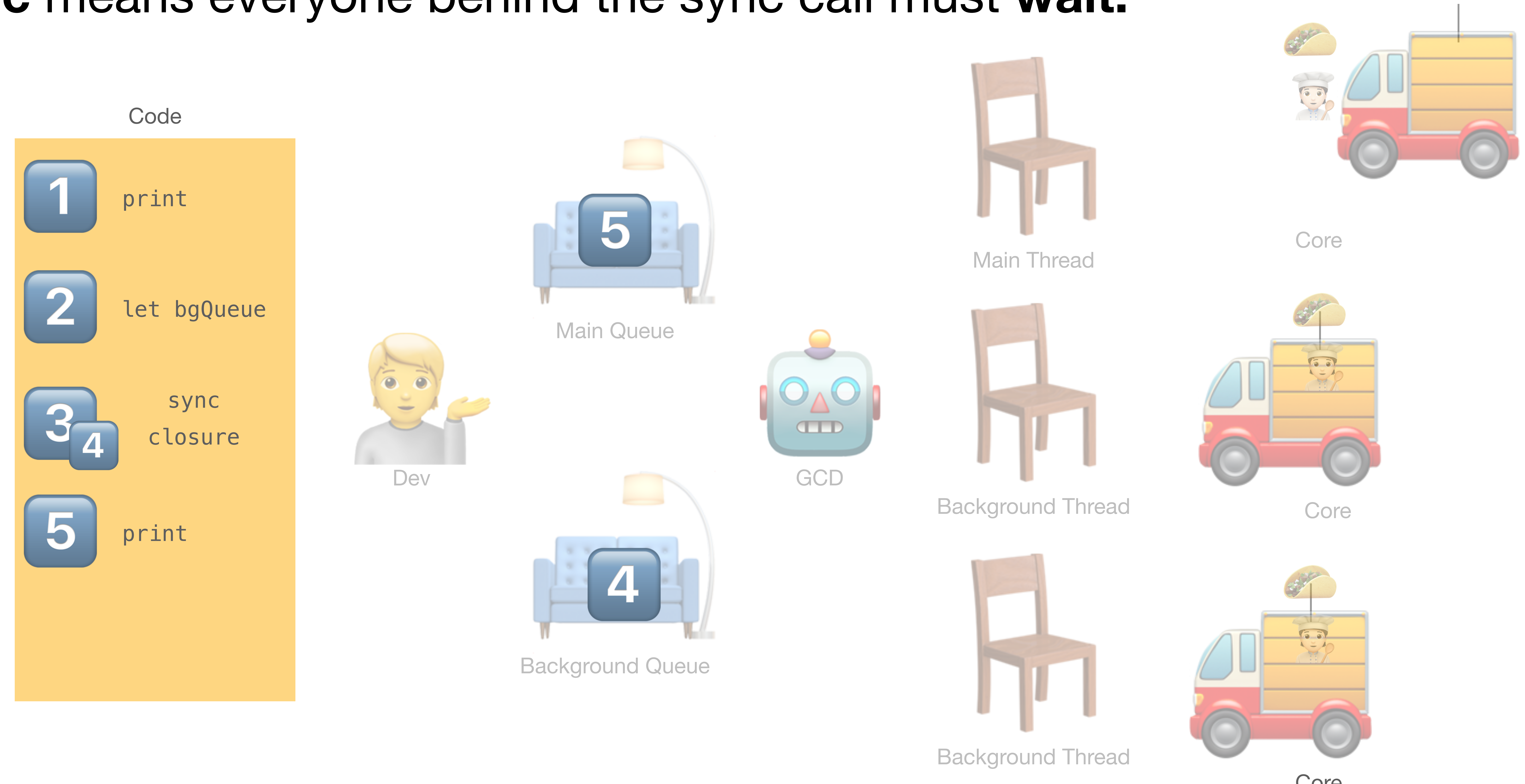
The Rules of GCD

Sync means everyone behind the sync call must **wait.**



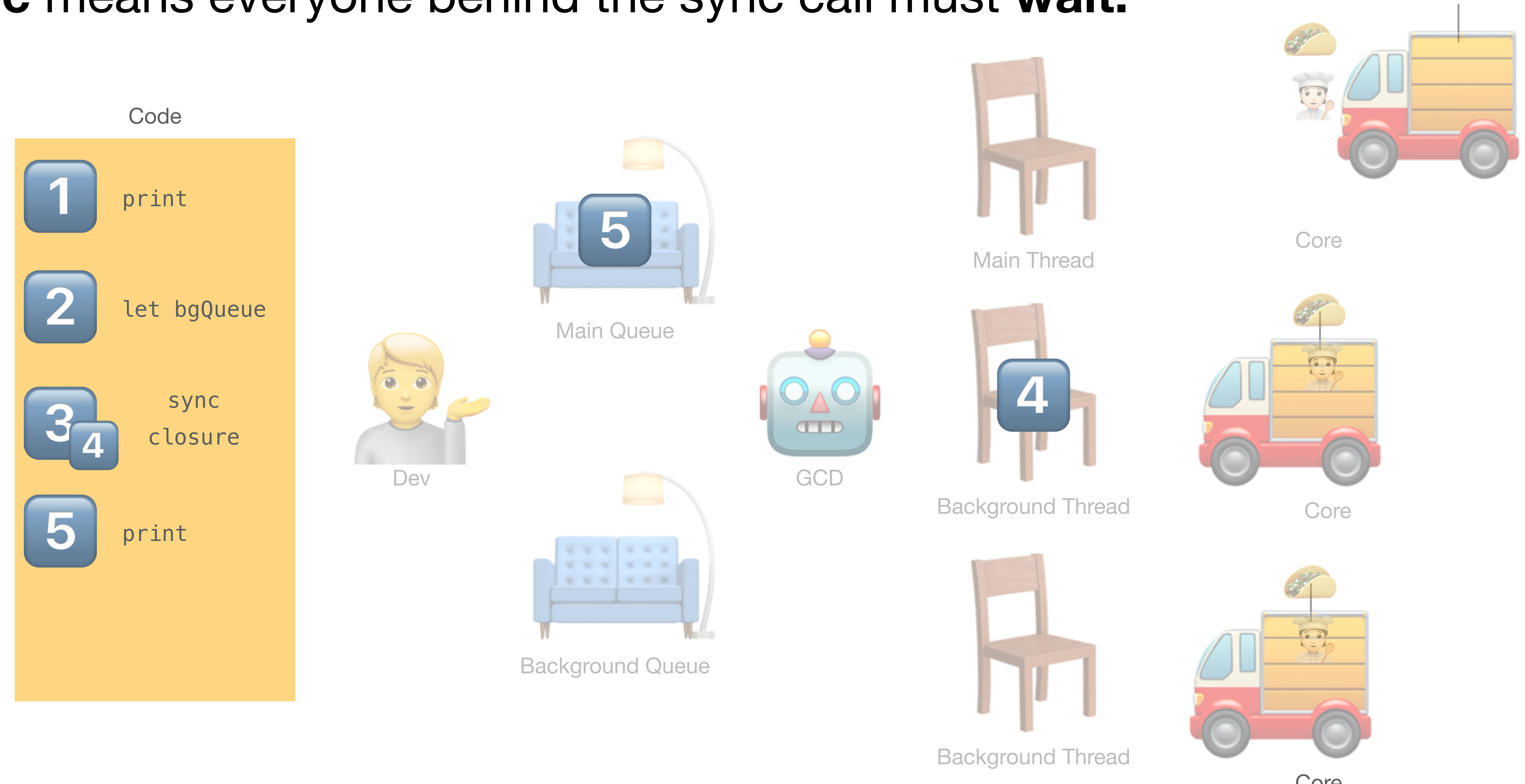
The Rules of GCD

Sync means everyone behind the sync call must **wait.**



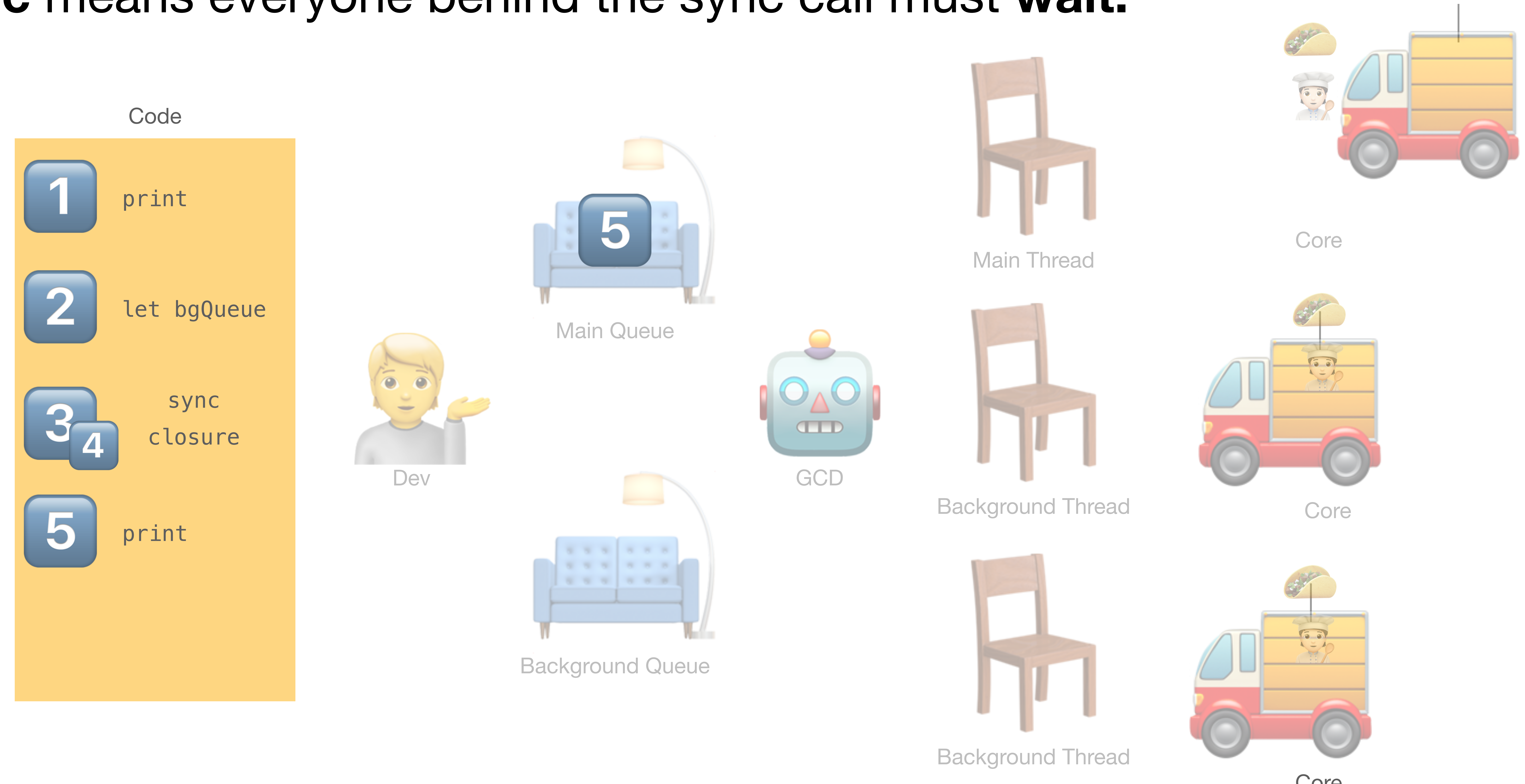
The Rules of GCD

Sync means everyone behind the sync call must **wait.**



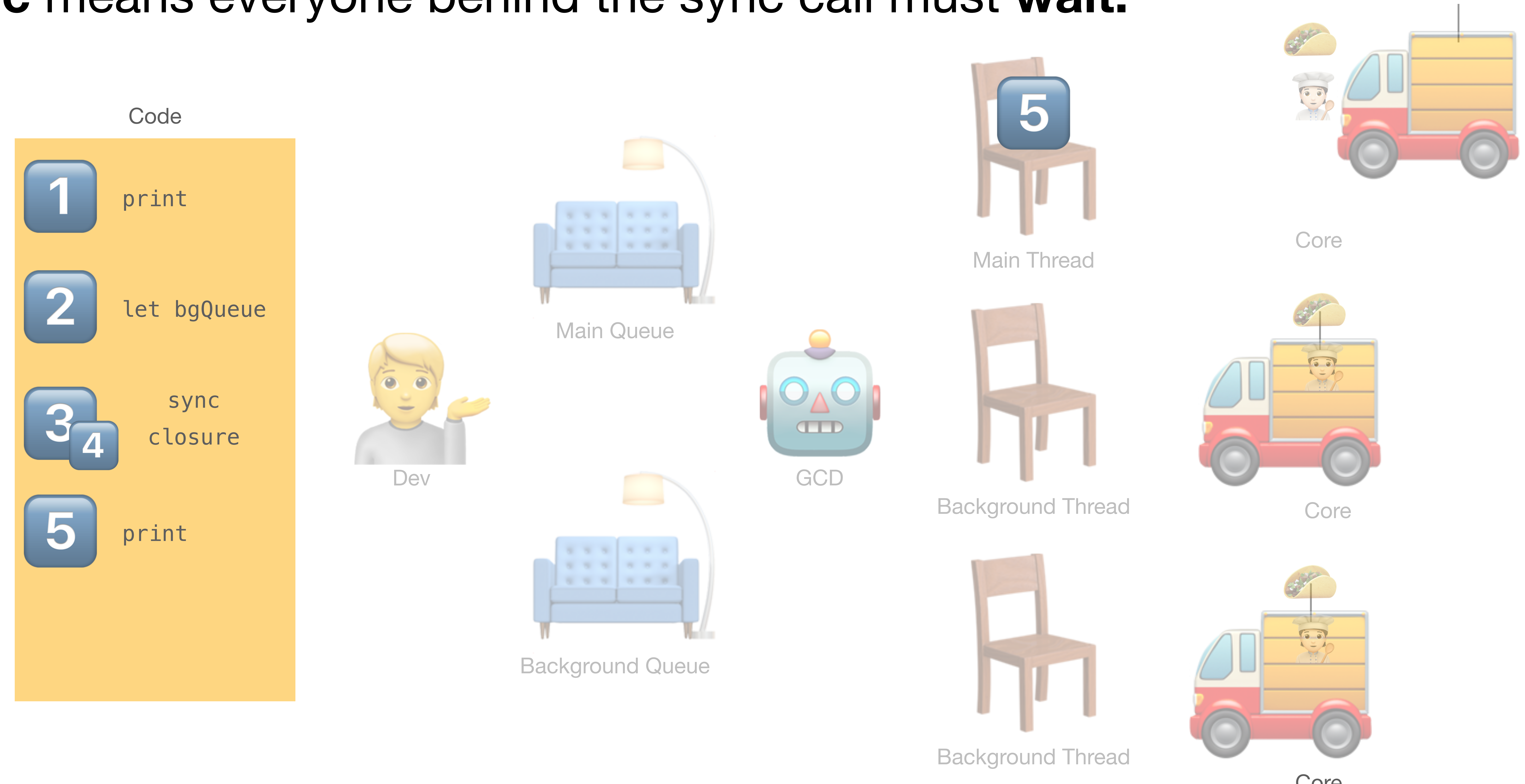
The Rules of GCD

Sync means everyone behind the sync call must **wait.**



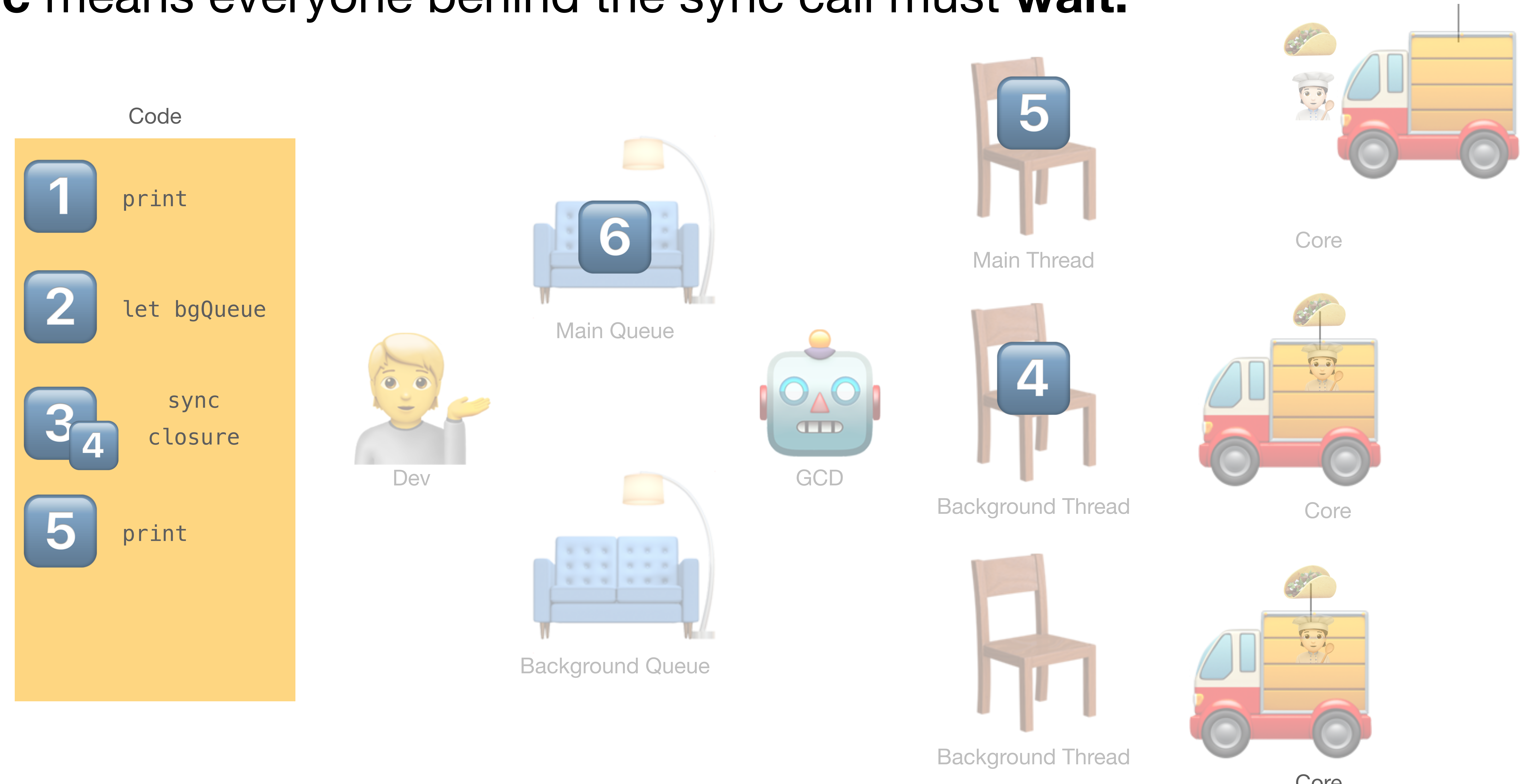
The Rules of GCD

Sync means everyone behind the sync call must **wait.**



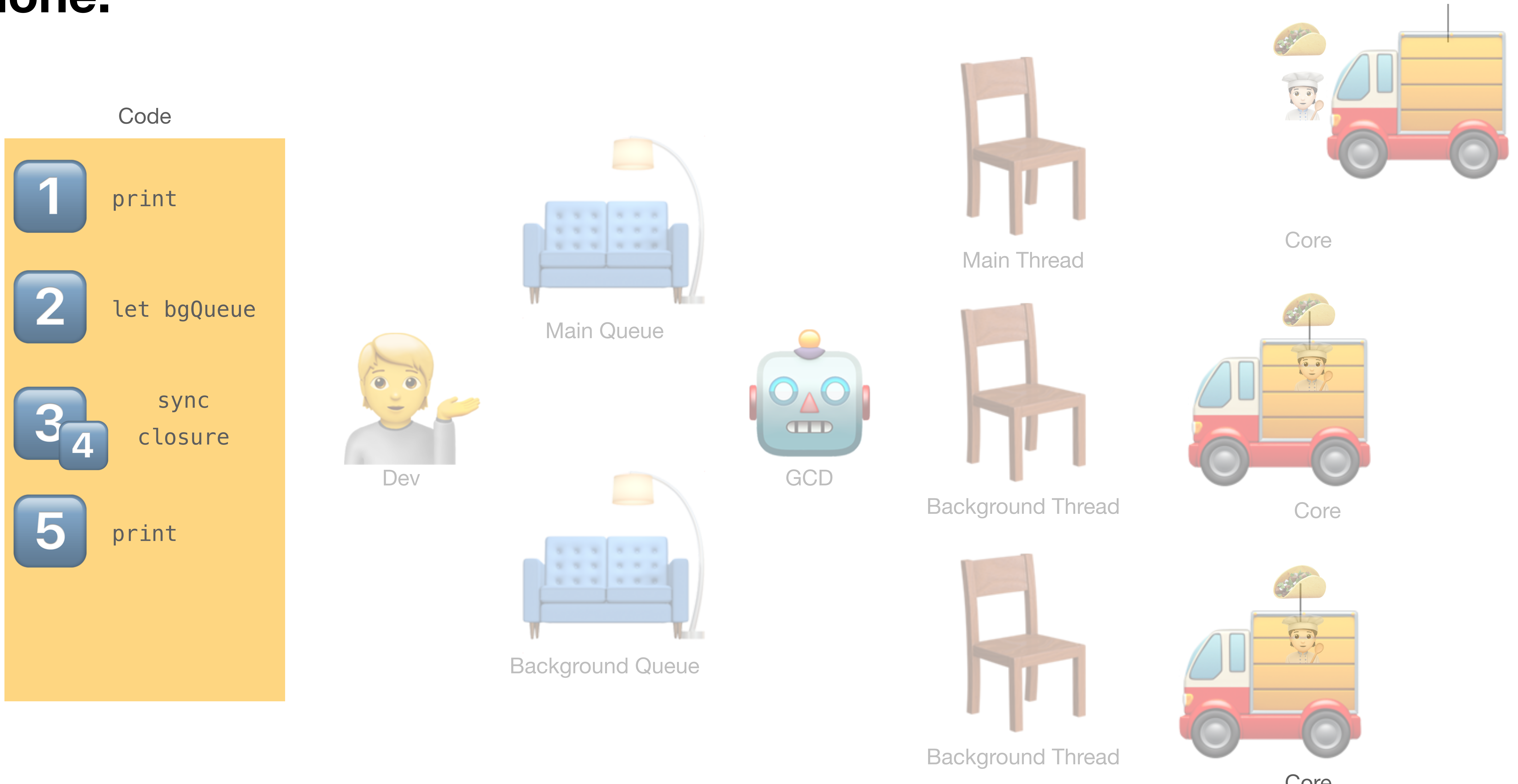
The Rules of GCD

Sync means everyone behind the sync call must **wait.**



The Rules of GCD

All done.

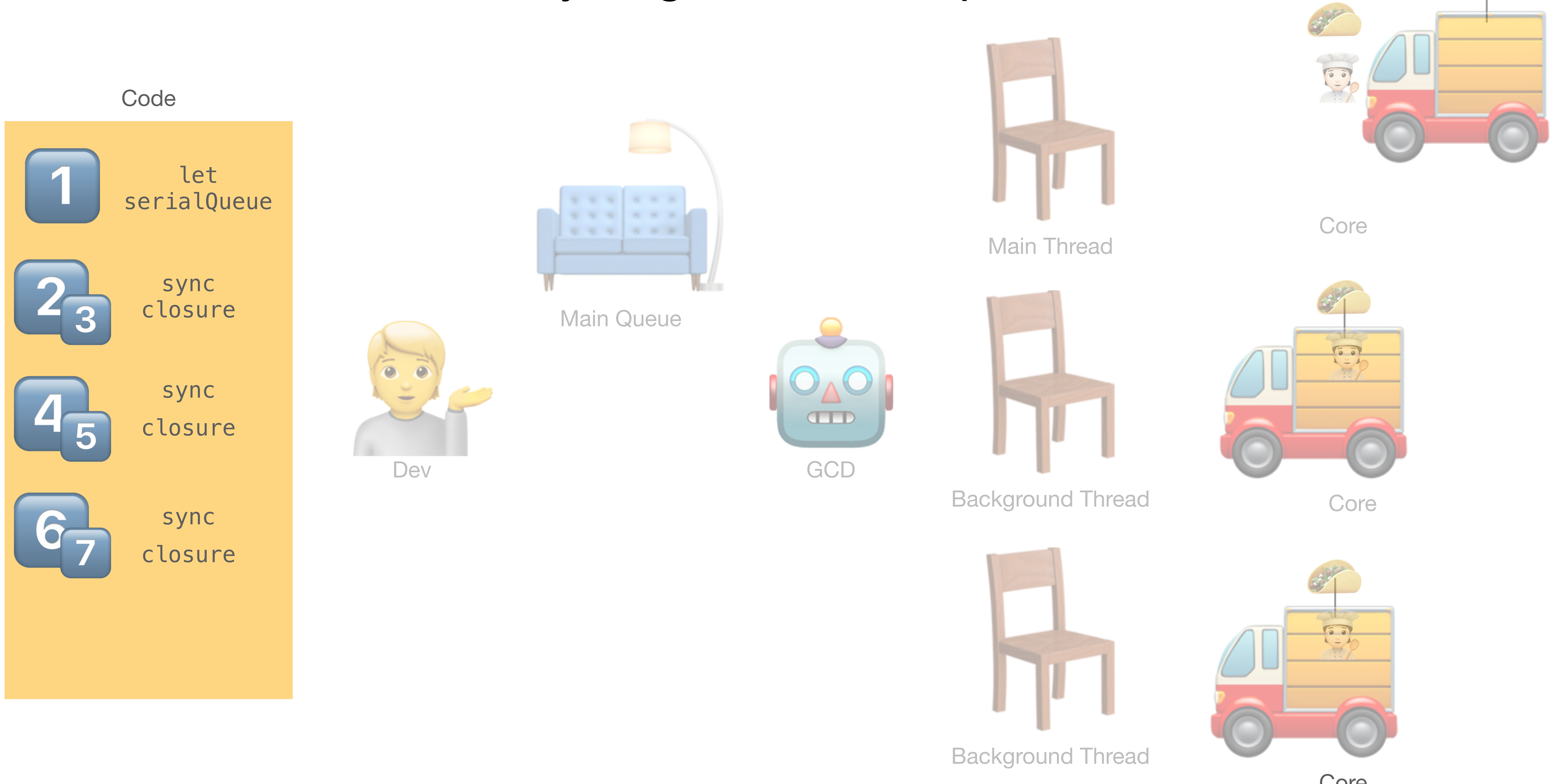


Live Demo

The second horseman: Serial

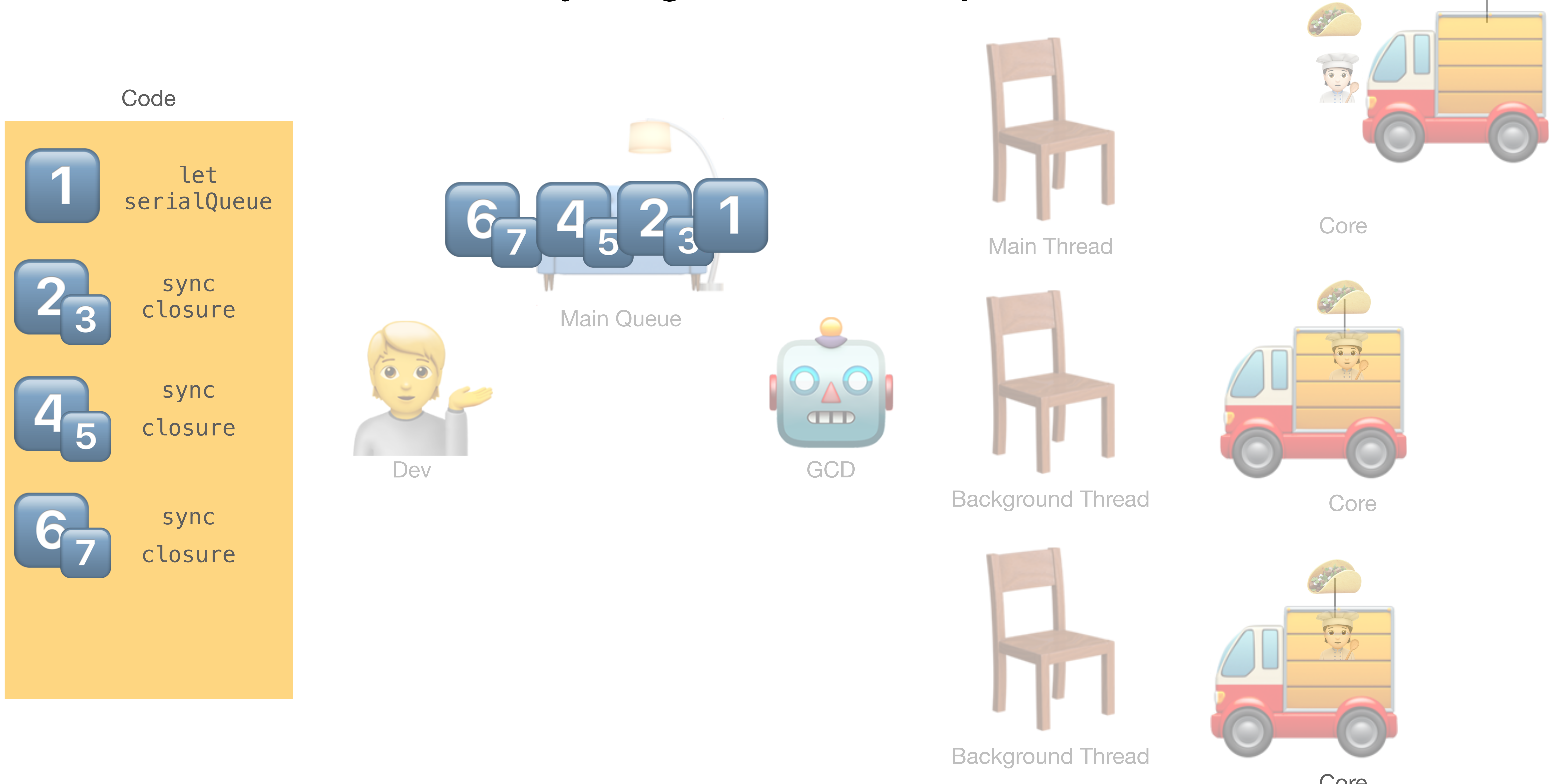
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



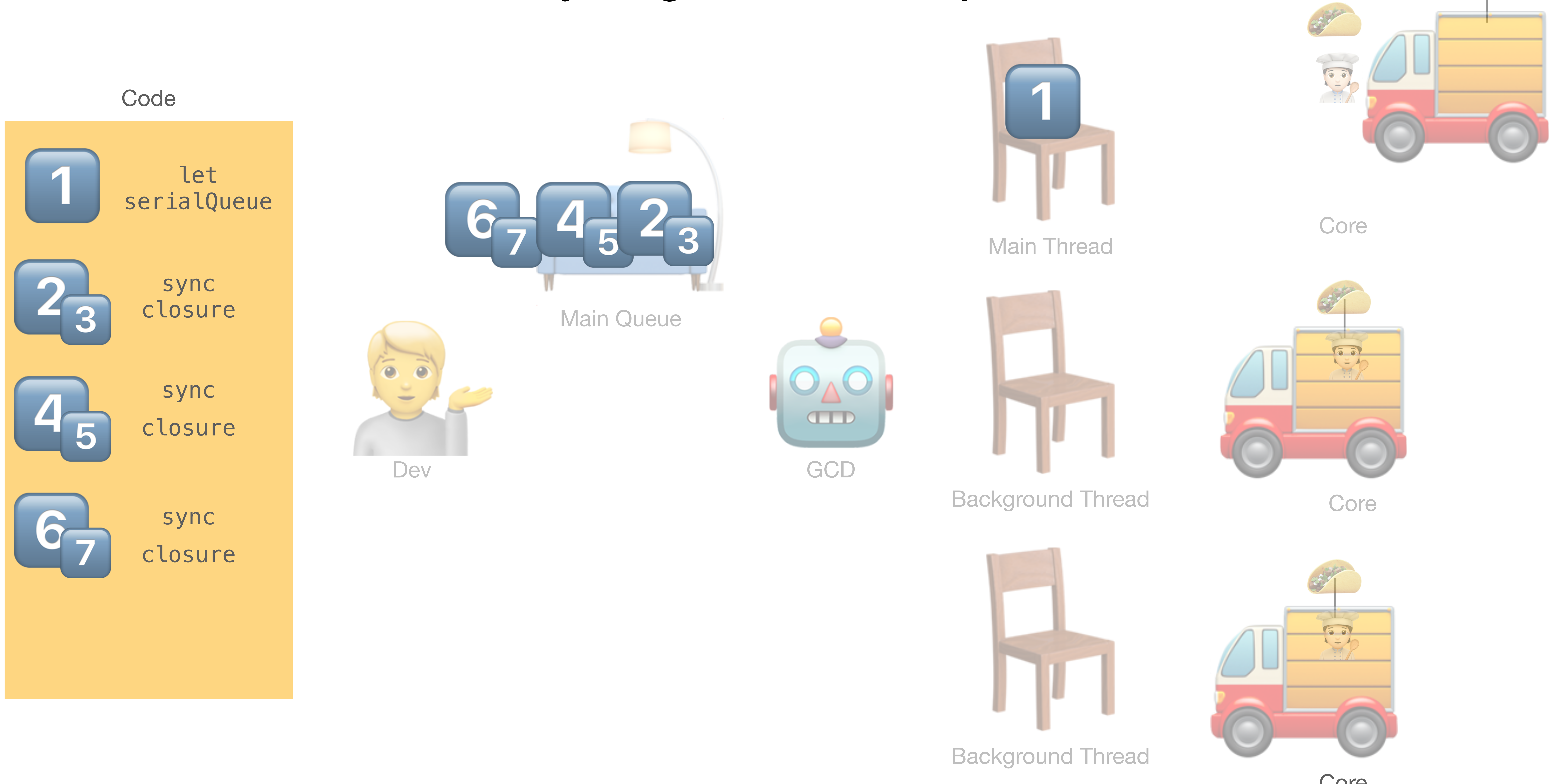
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



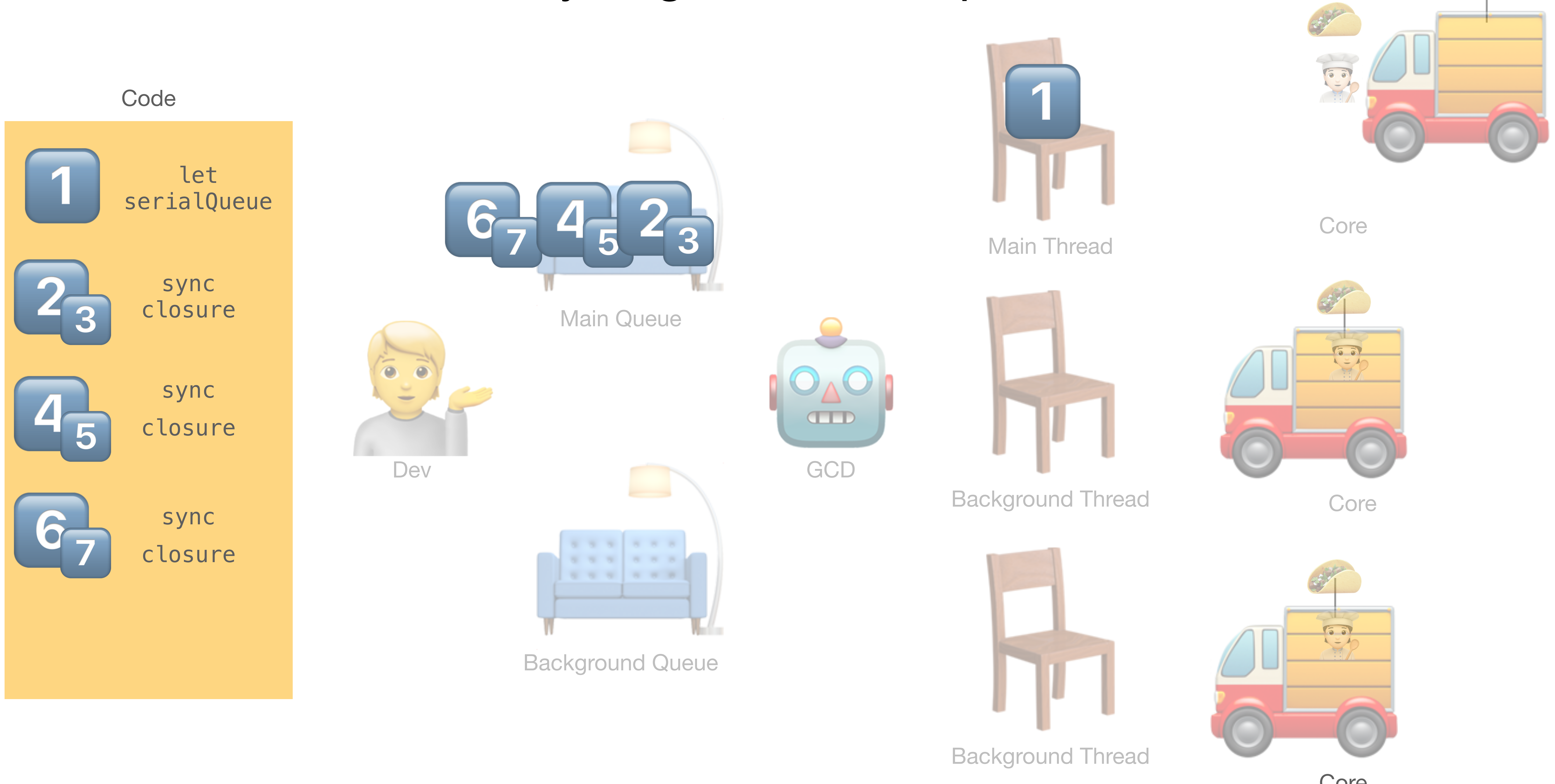
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



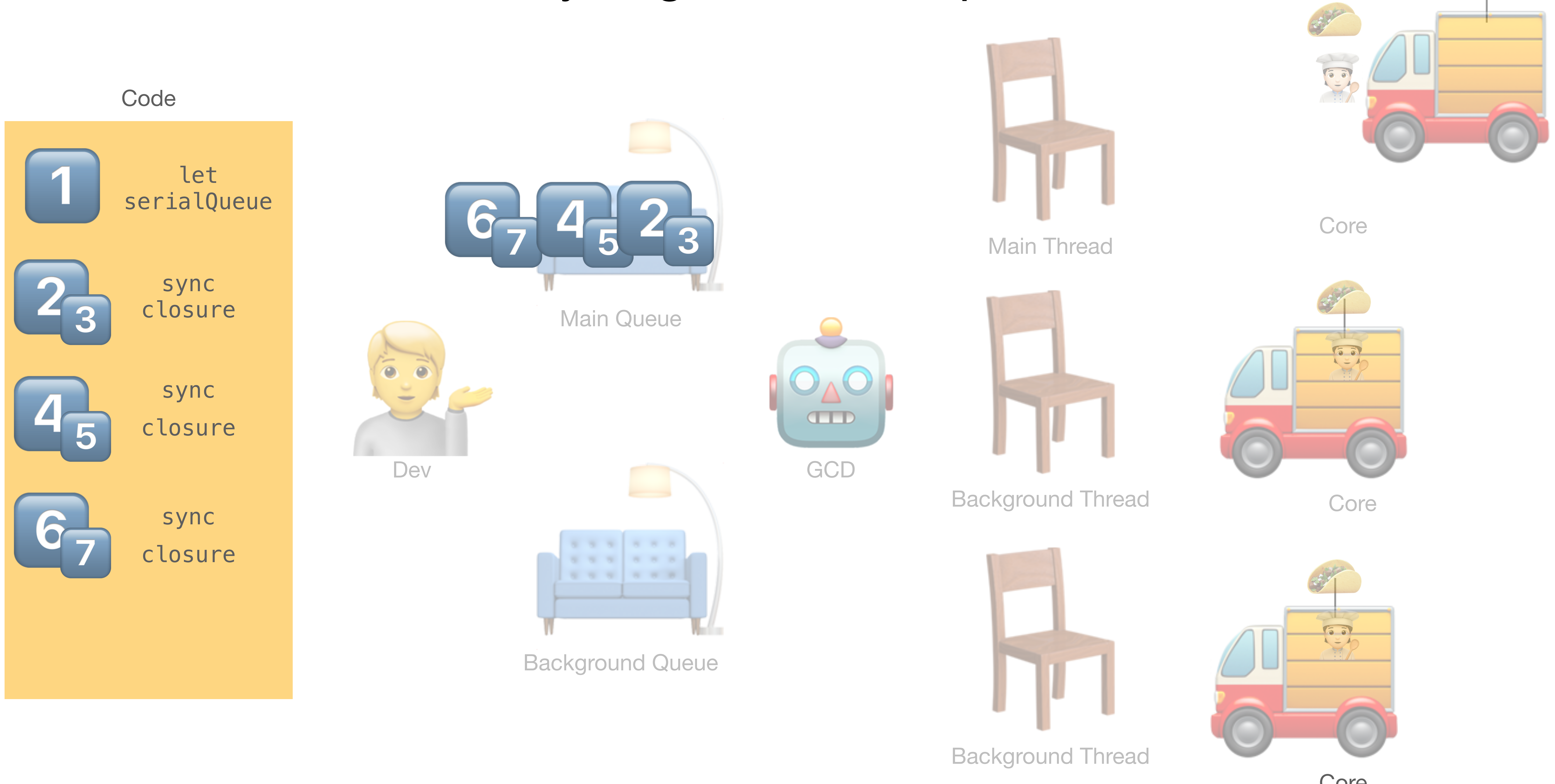
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



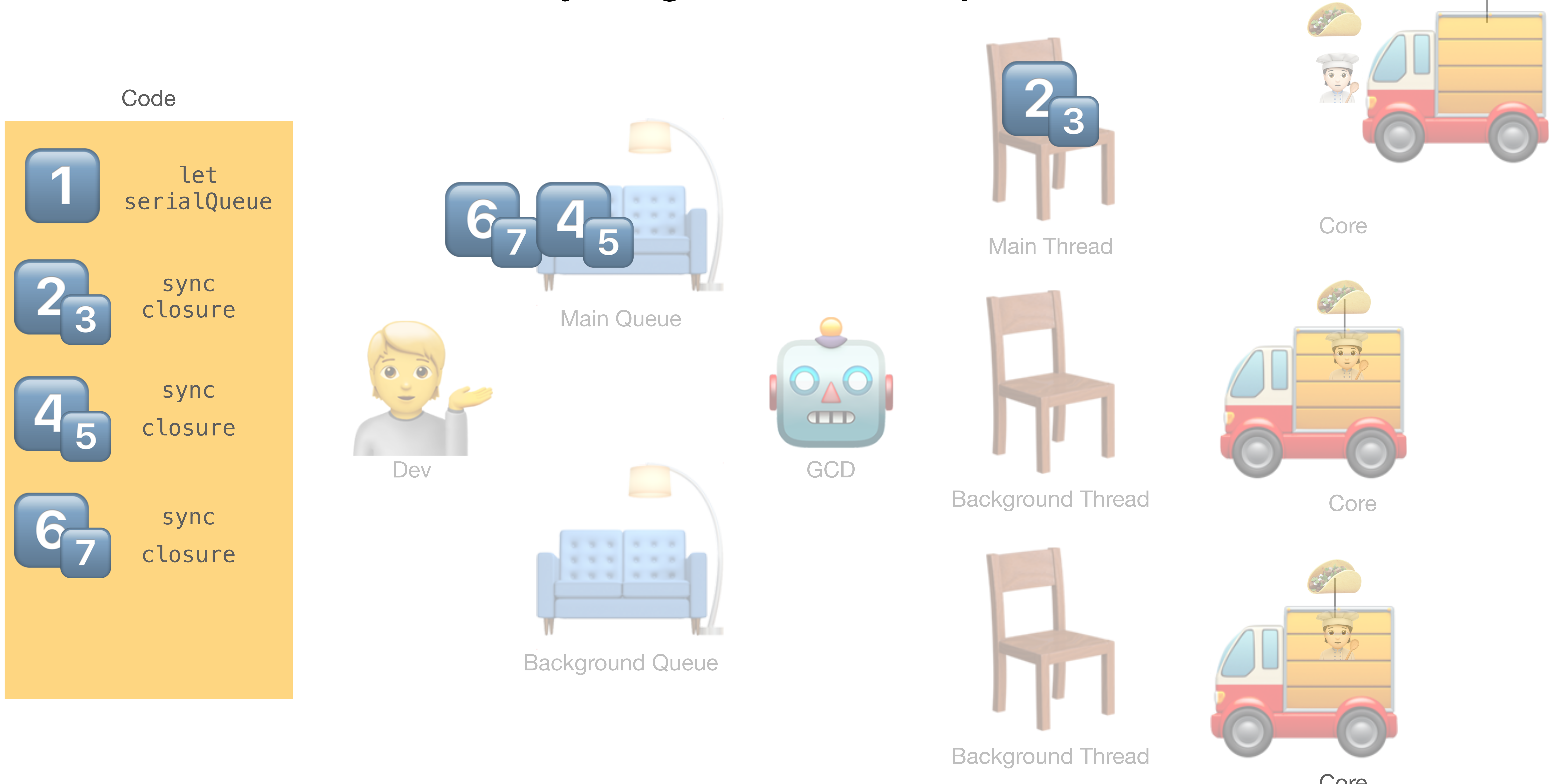
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



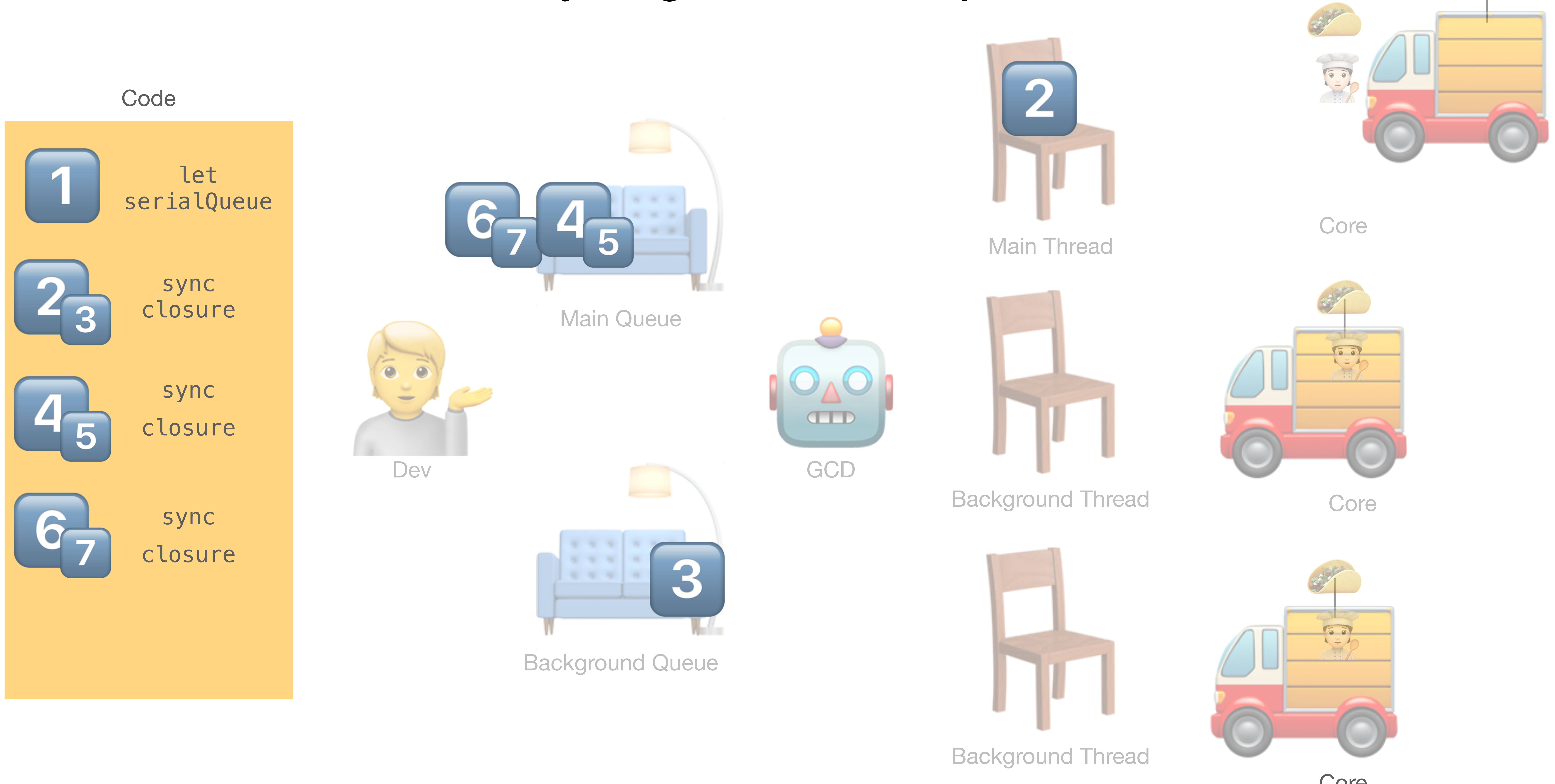
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



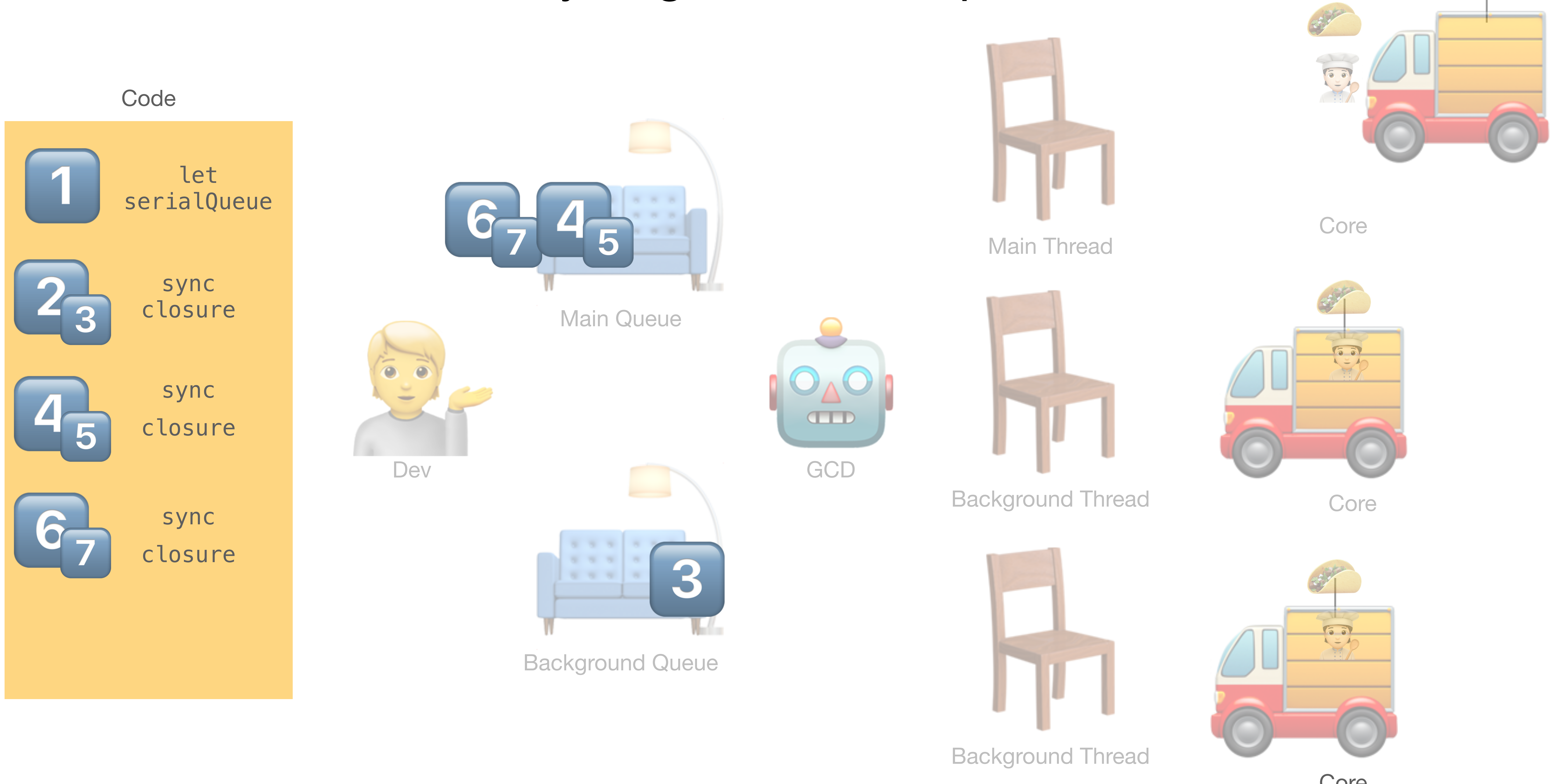
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



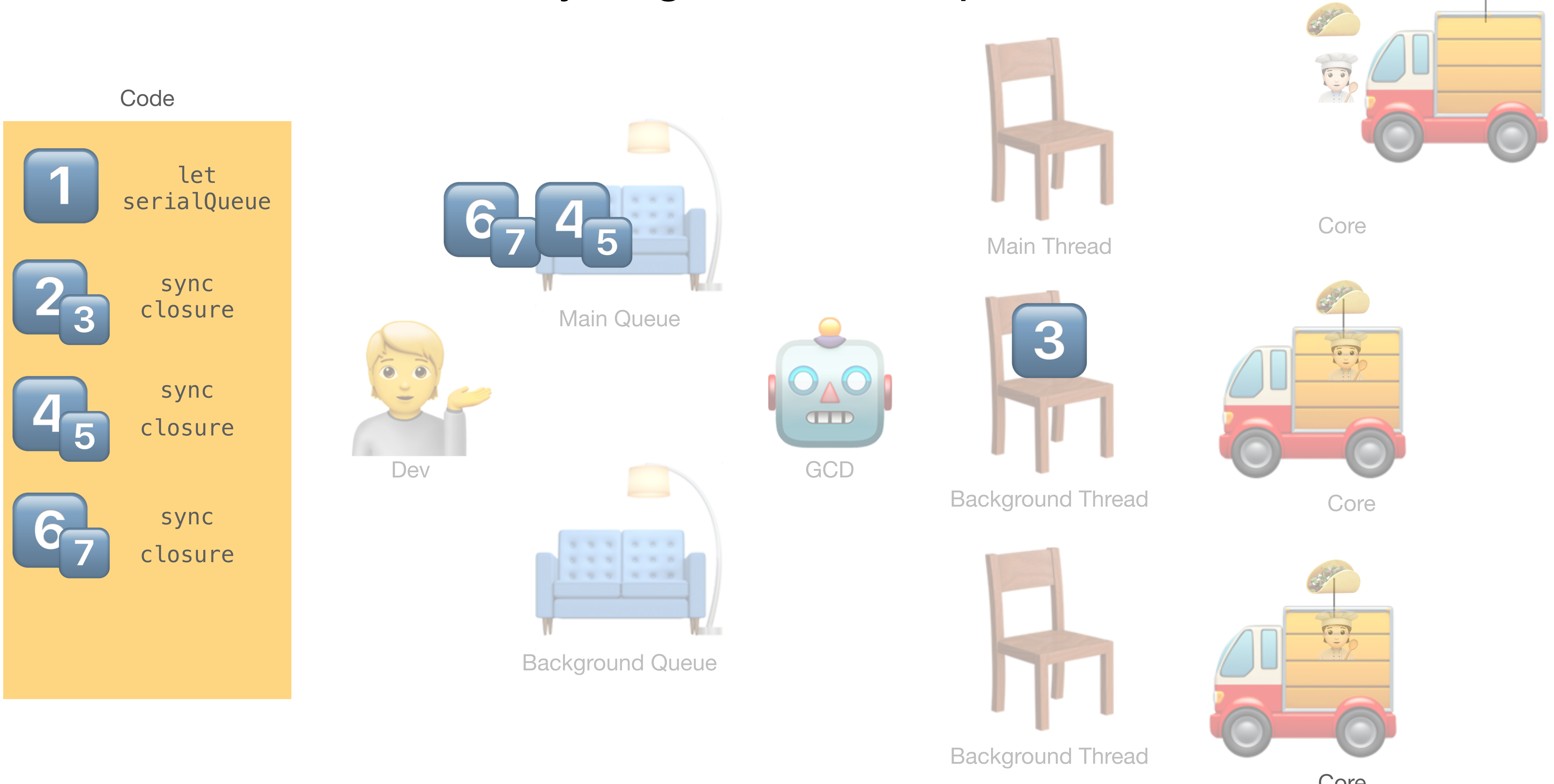
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



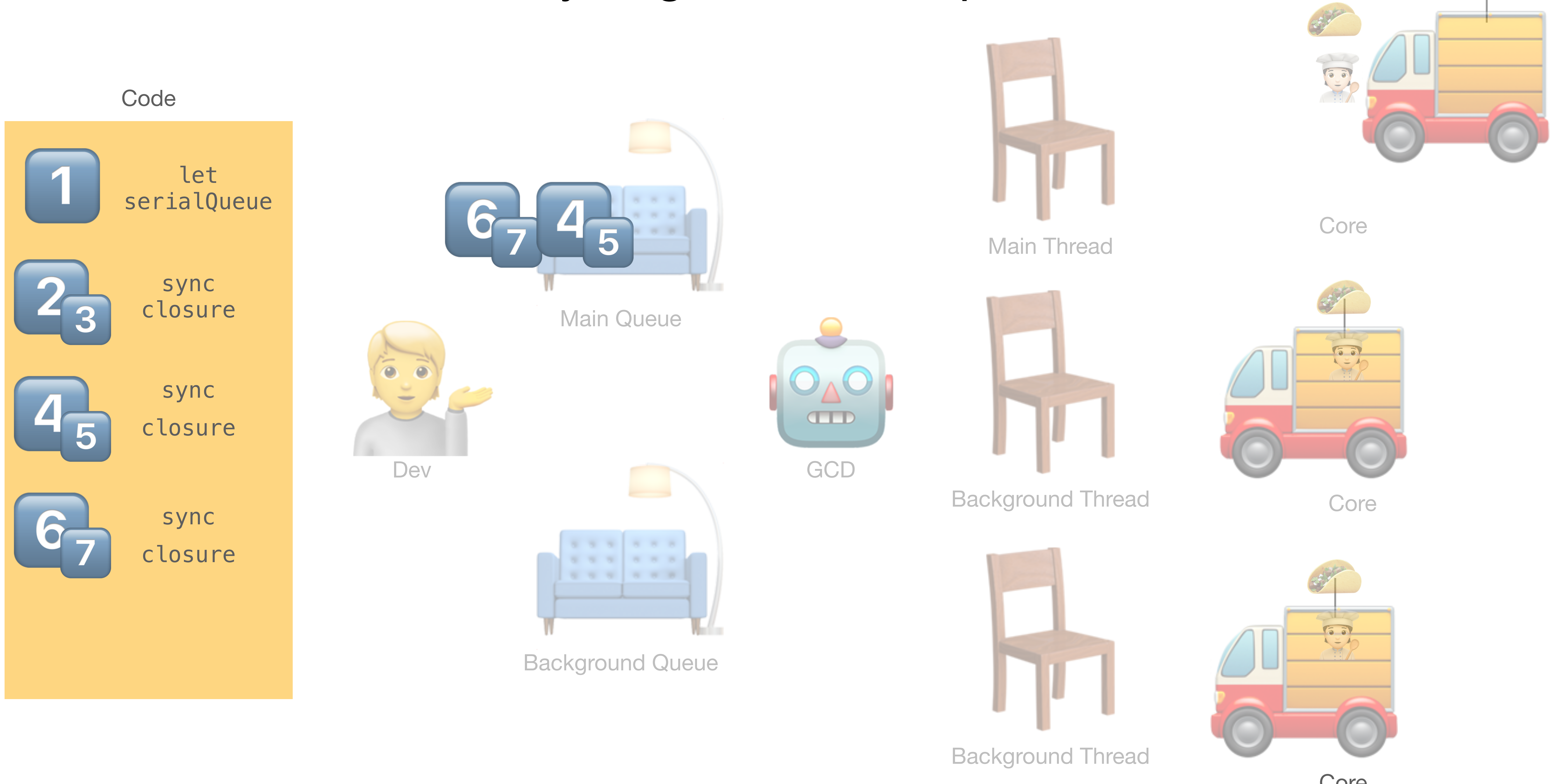
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



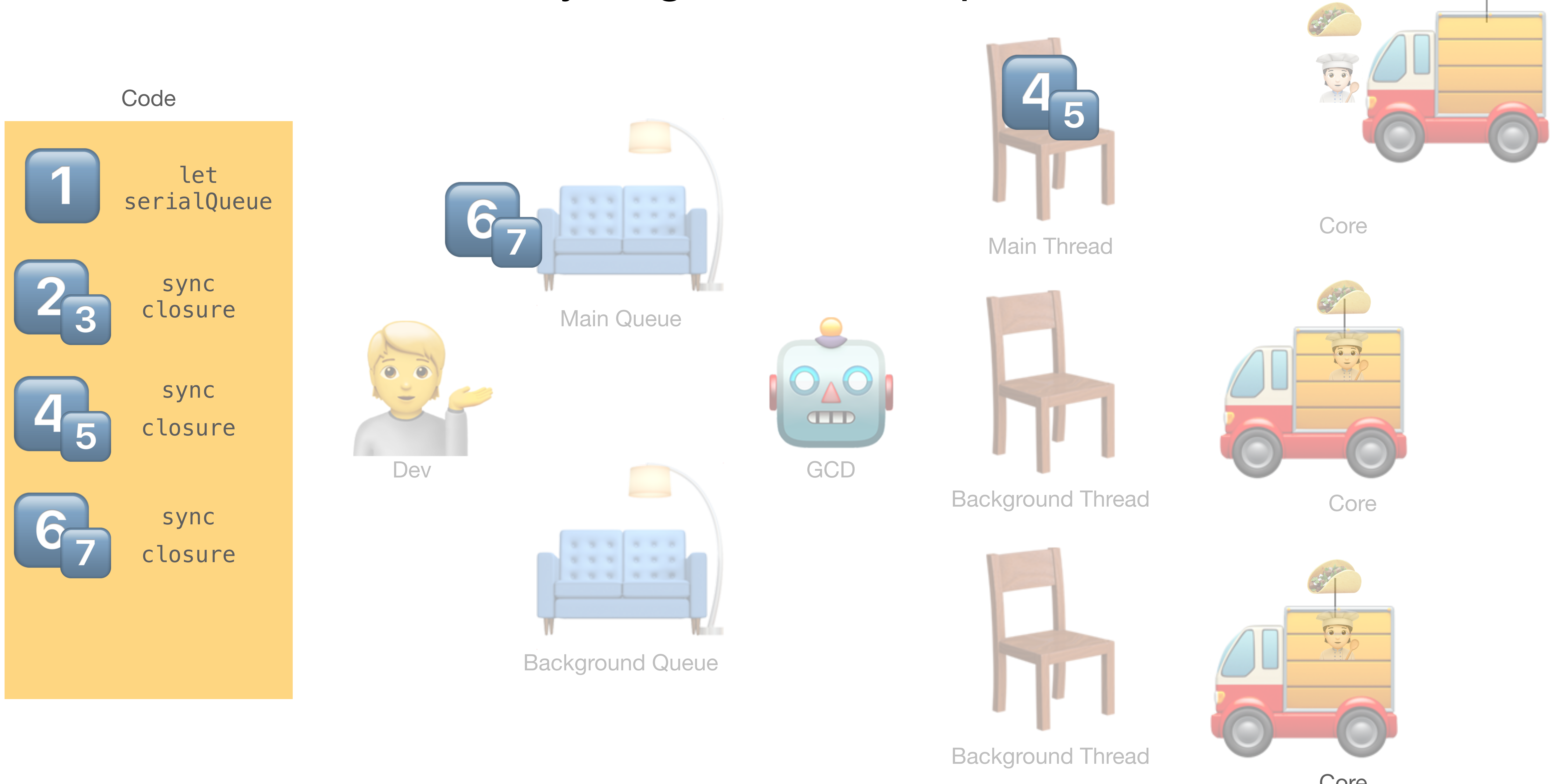
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



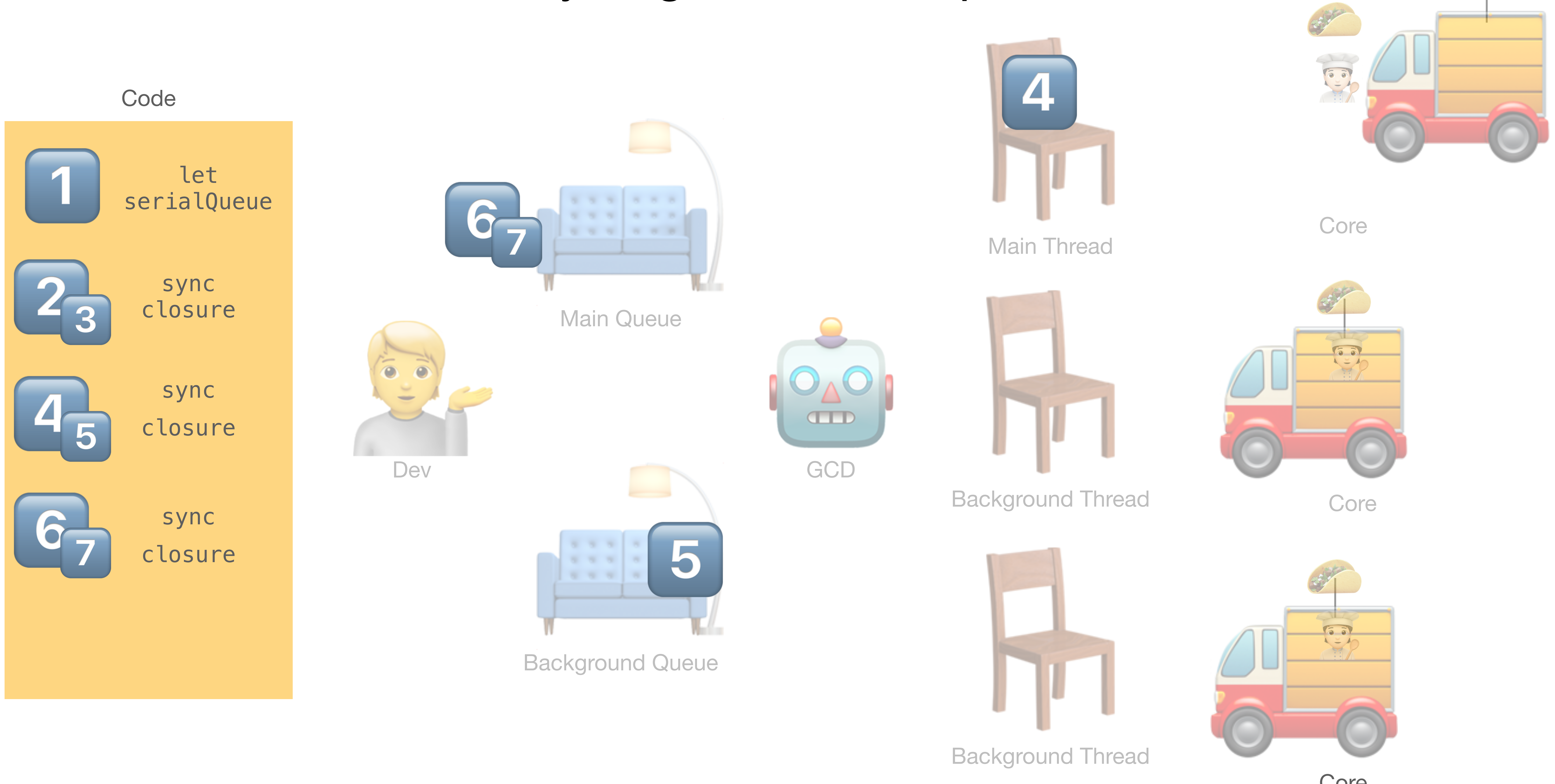
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



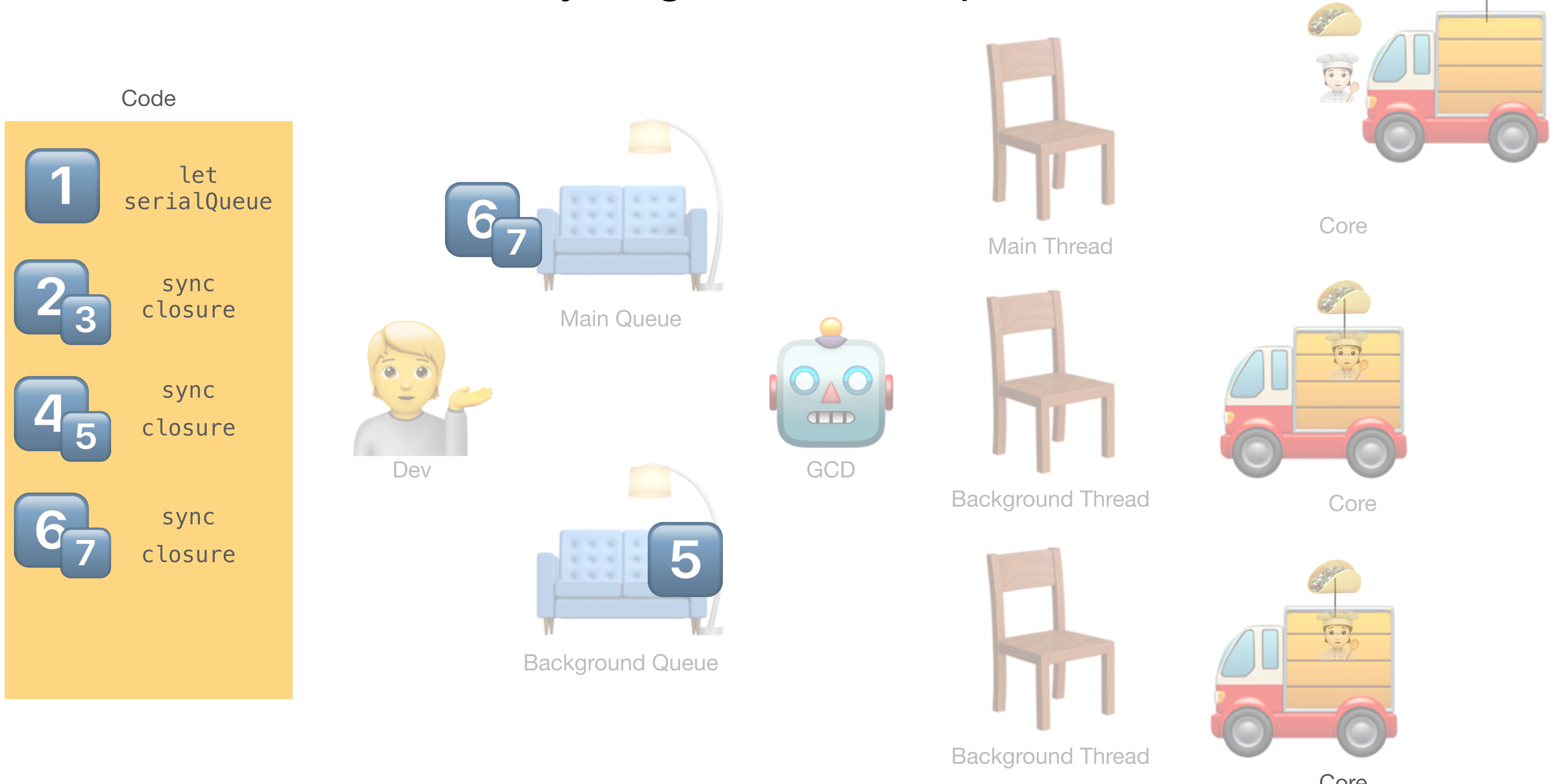
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



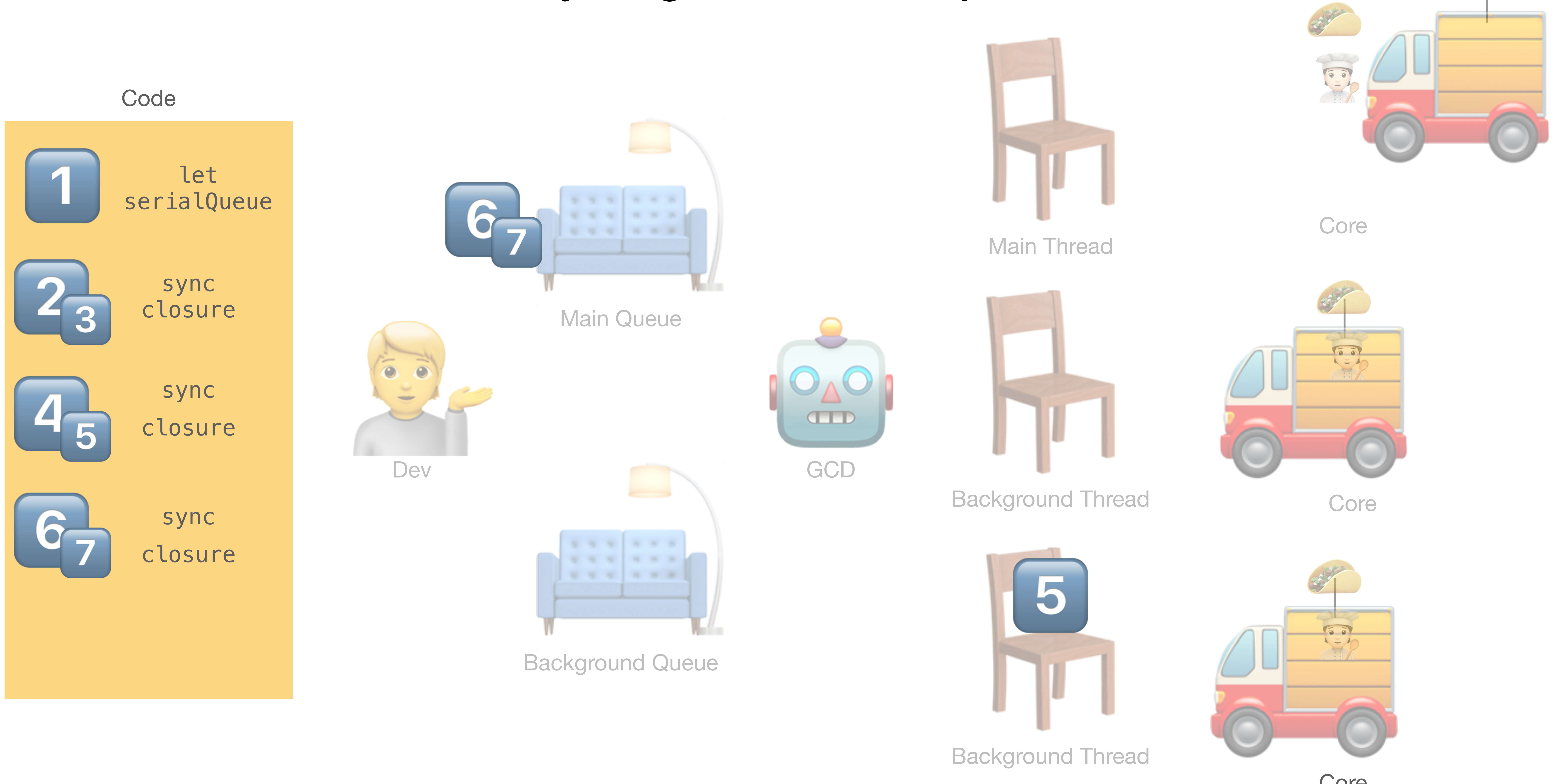
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



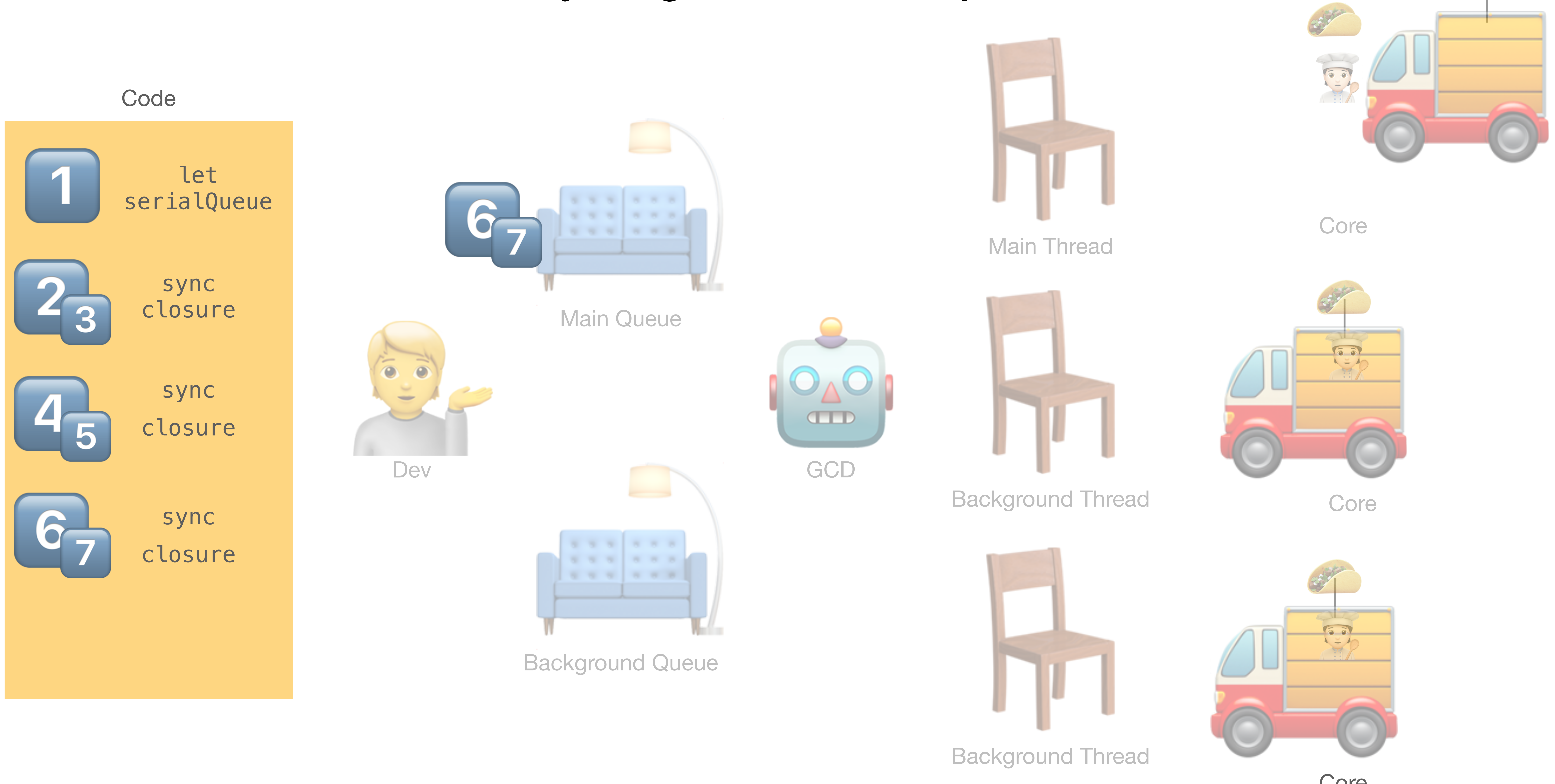
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



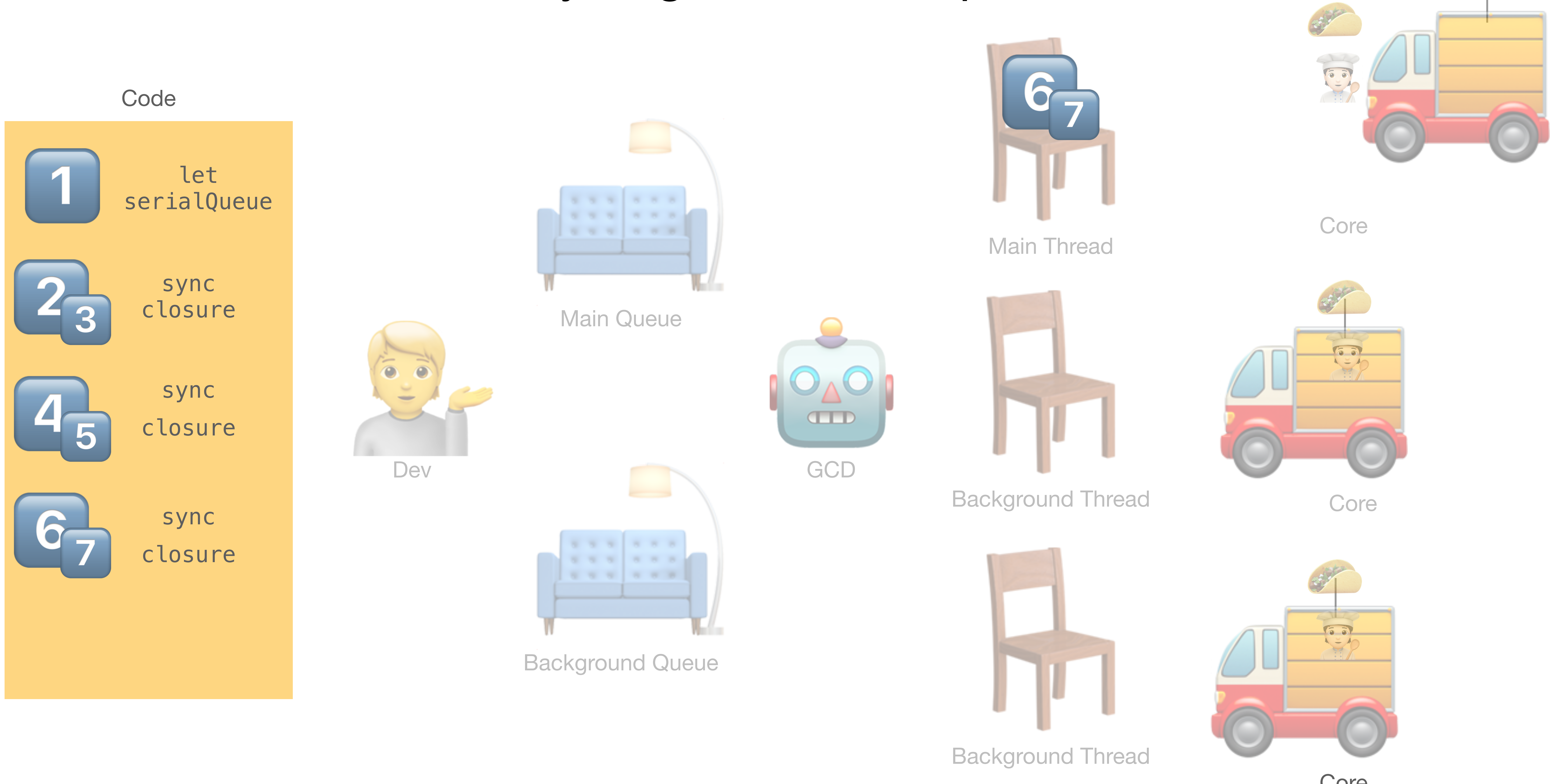
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



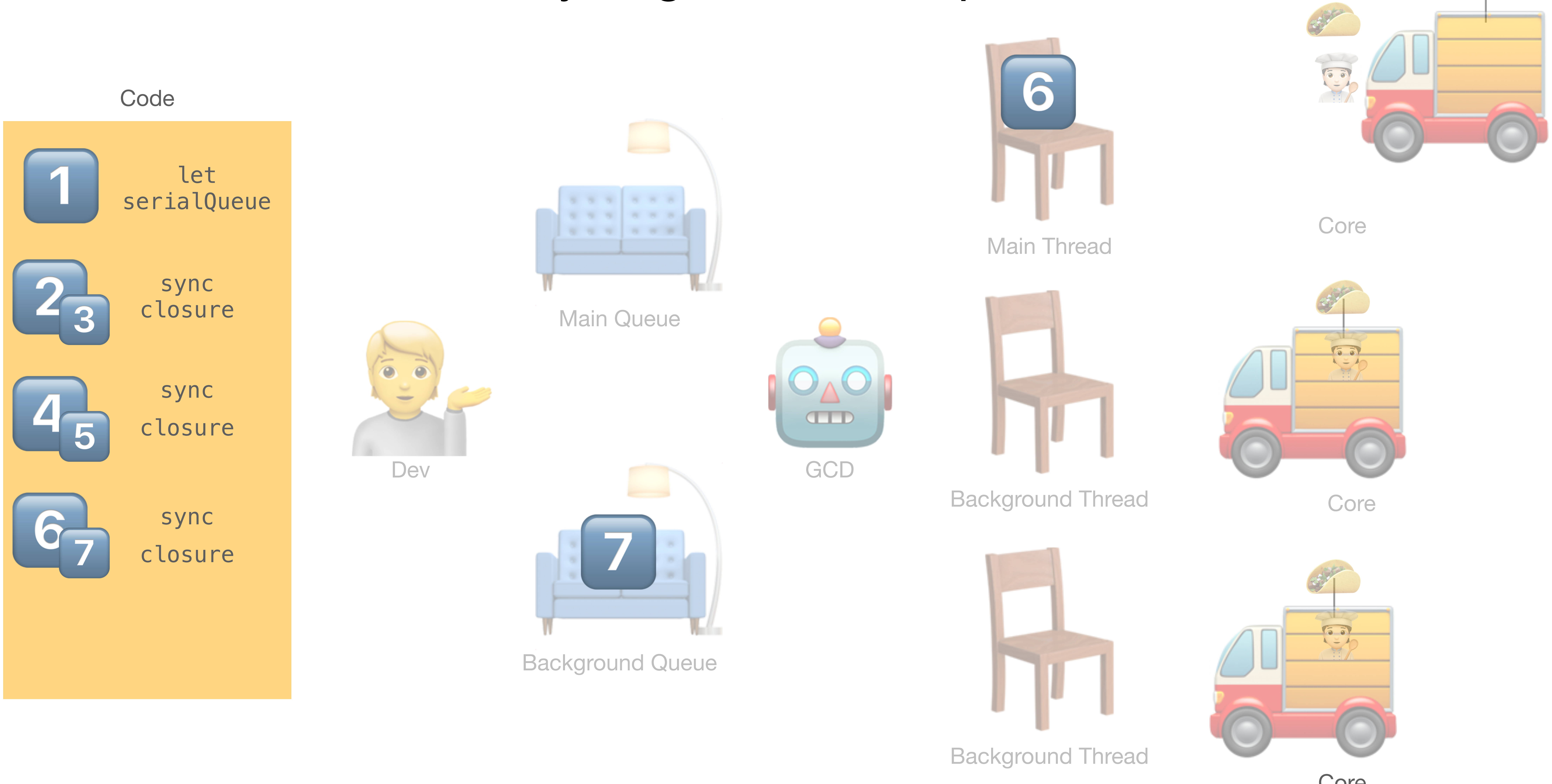
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



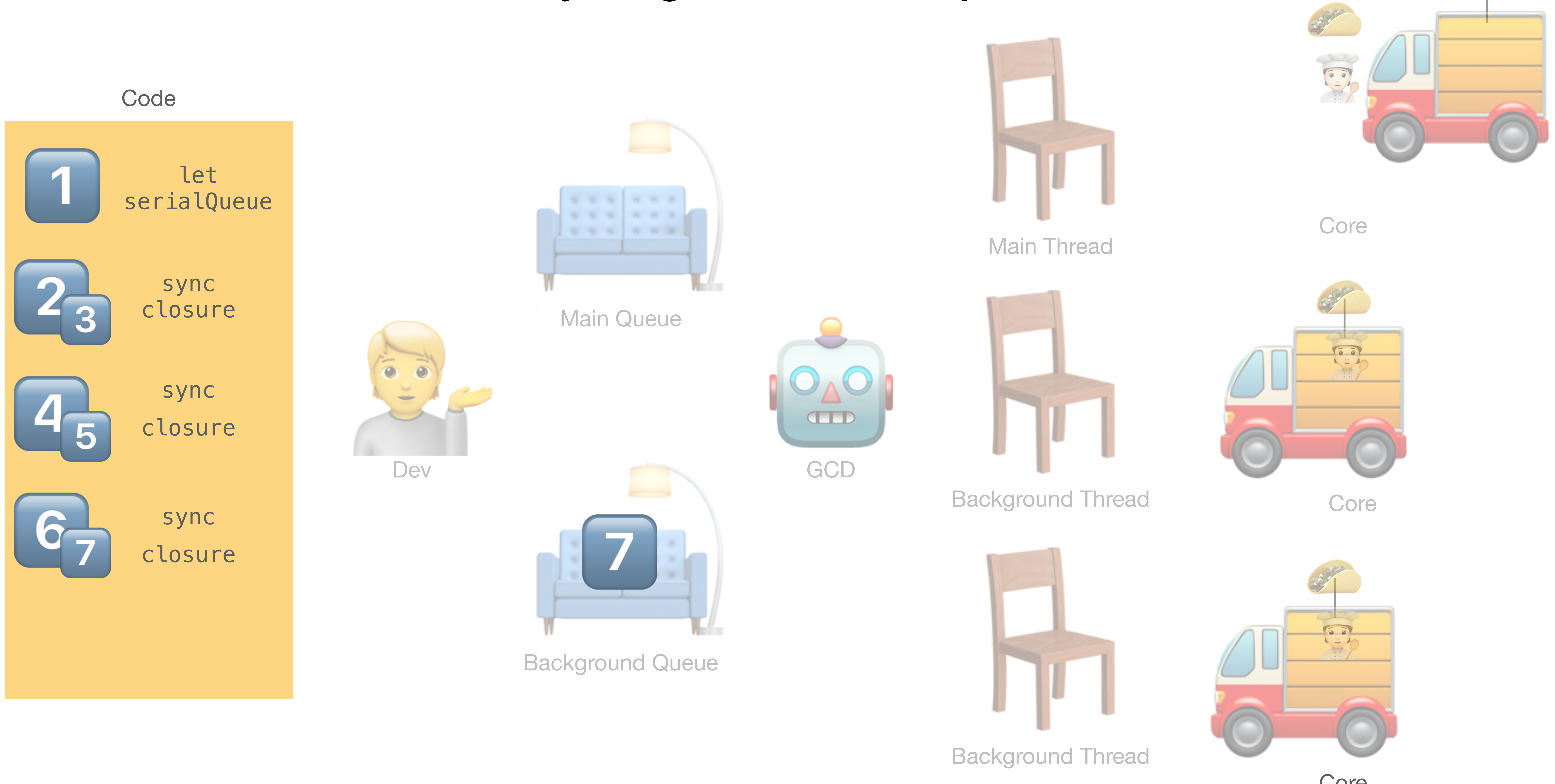
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



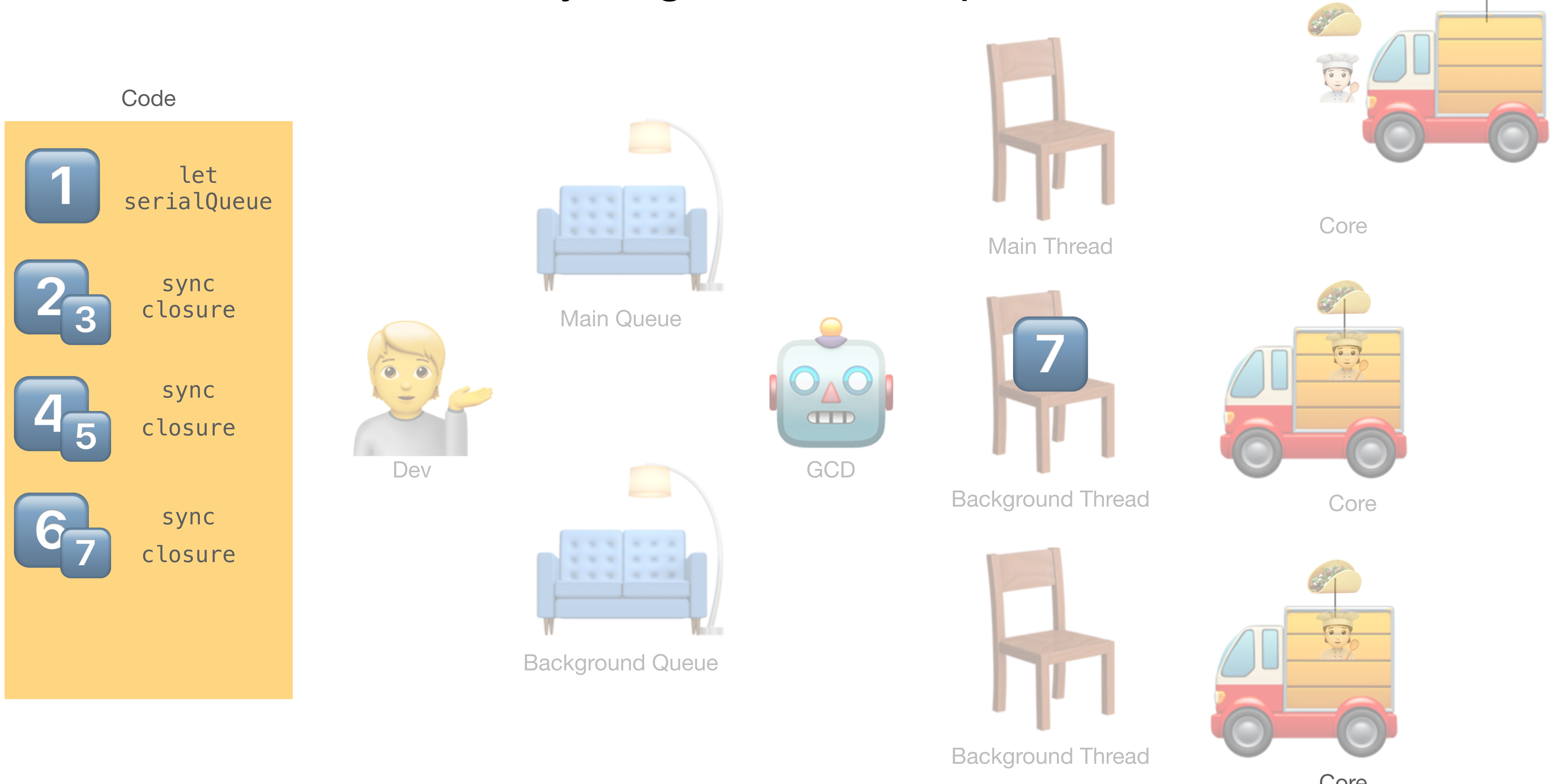
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



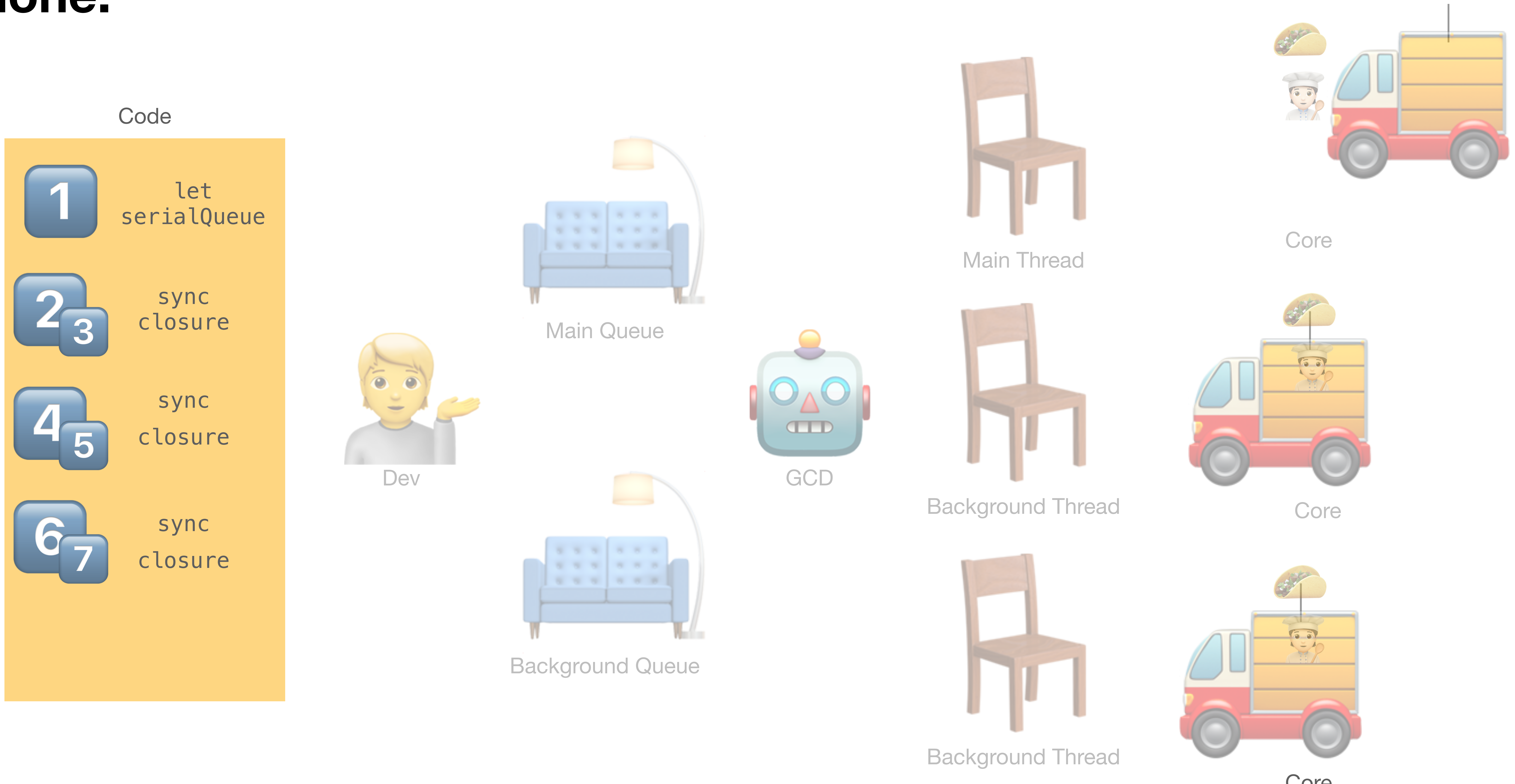
The Rules of GCD

Serial means a block can only begin when the previous block **finishes.**



The Rules of GCD

All done.

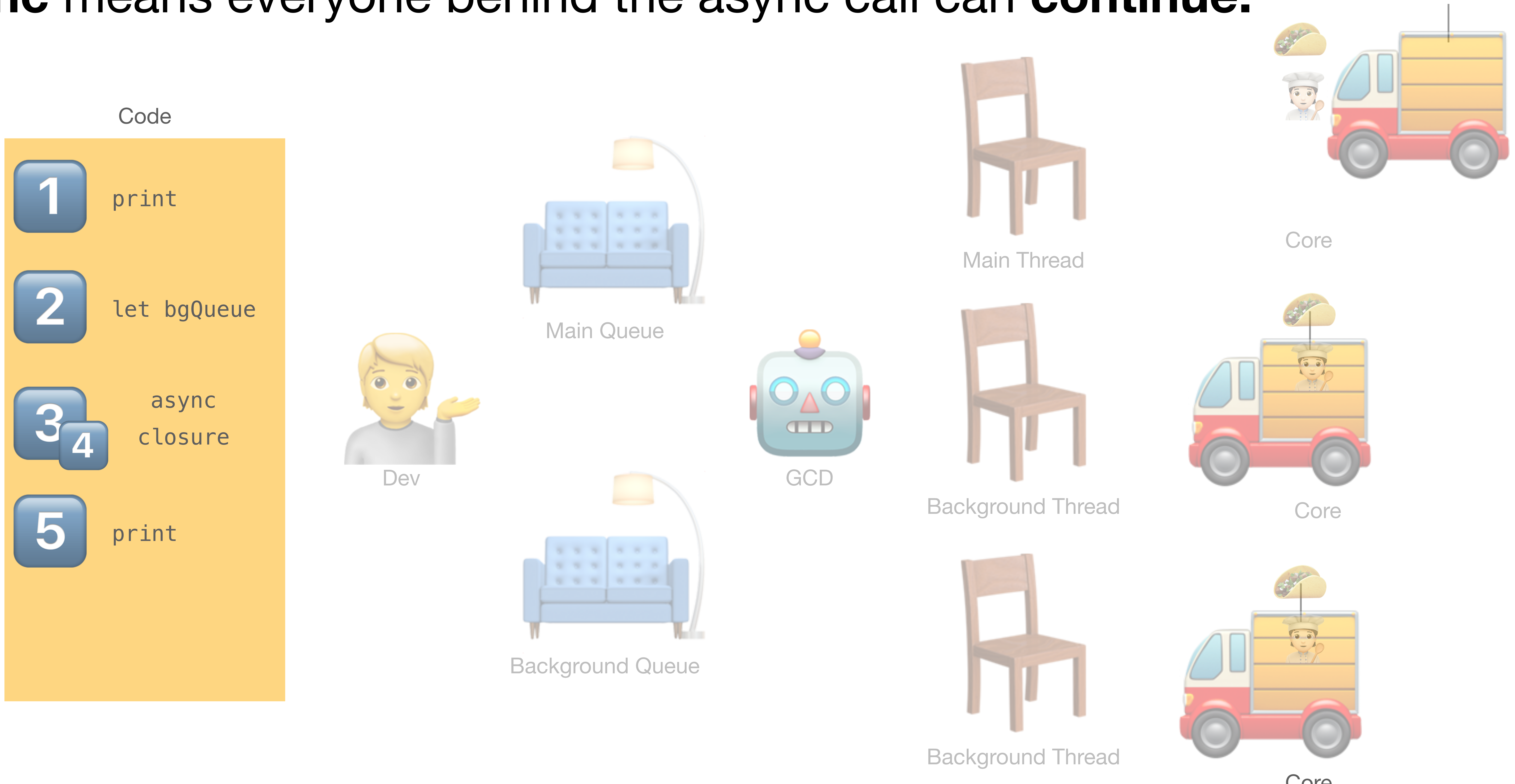


Live Demo

The third horseman: Async

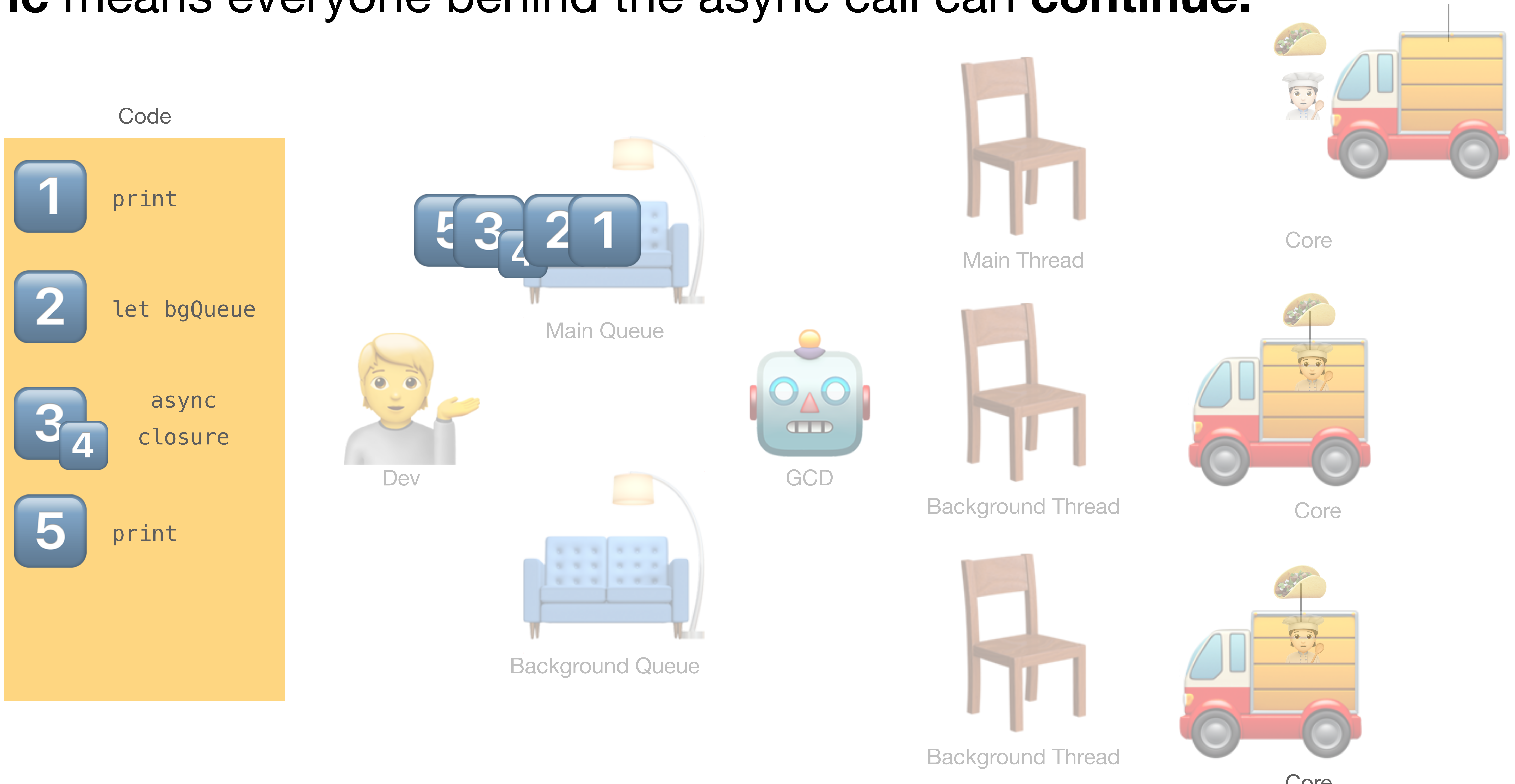
The Rules of GCD

Async means everyone behind the async call can **continue.**



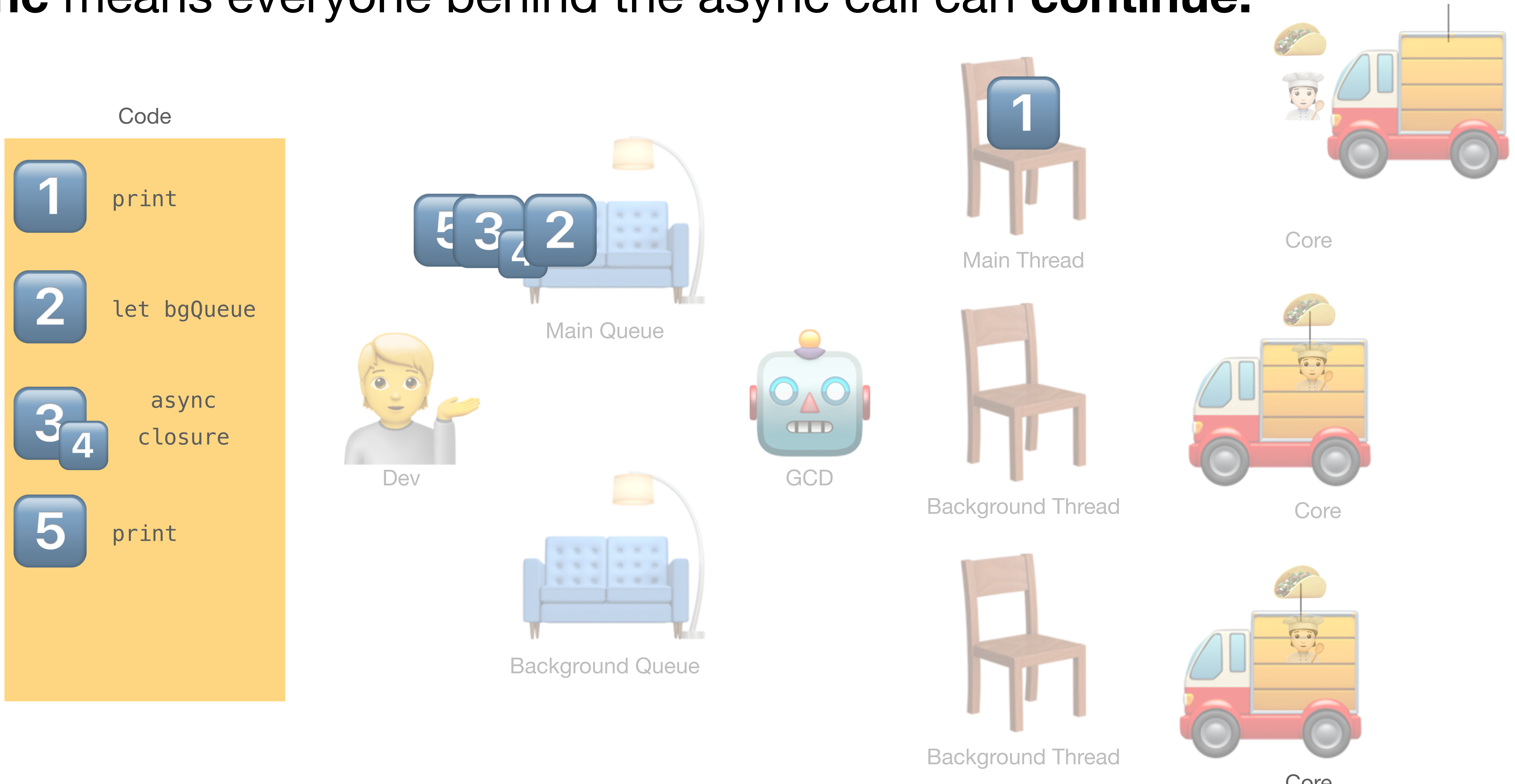
The Rules of GCD

Async means everyone behind the async call can **continue.**



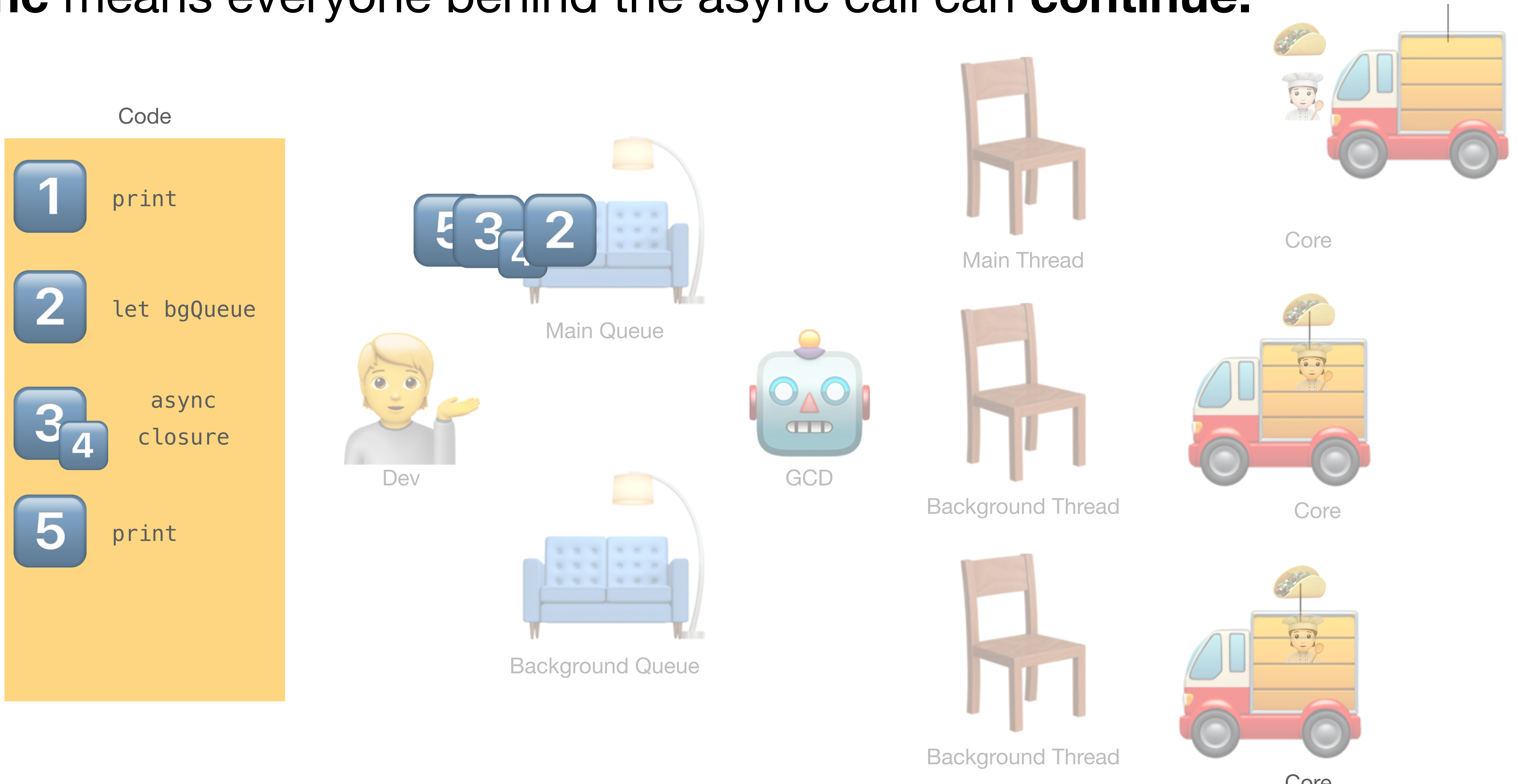
The Rules of GCD

Async means everyone behind the async call can **continue.**



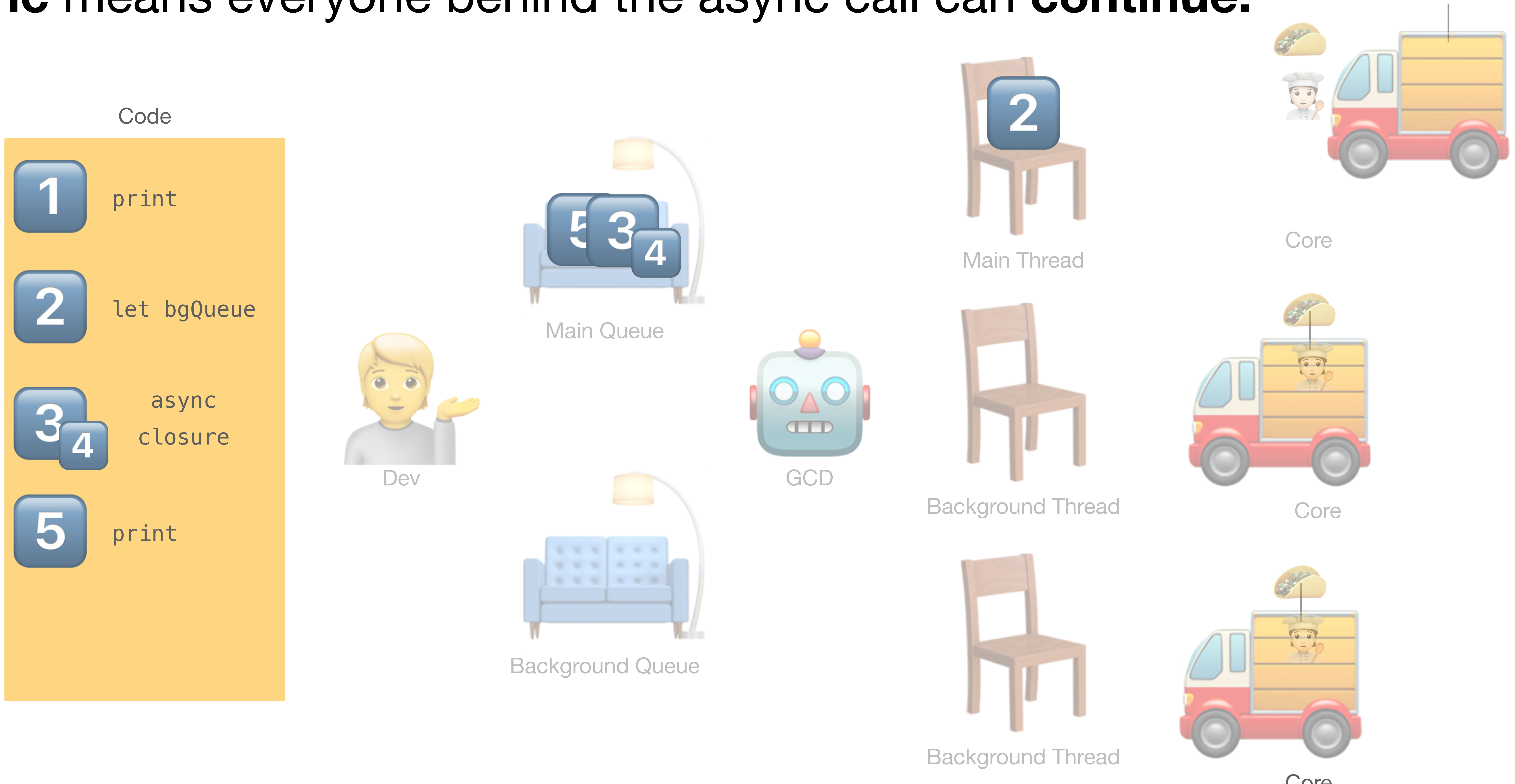
The Rules of GCD

Async means everyone behind the async call can **continue.**



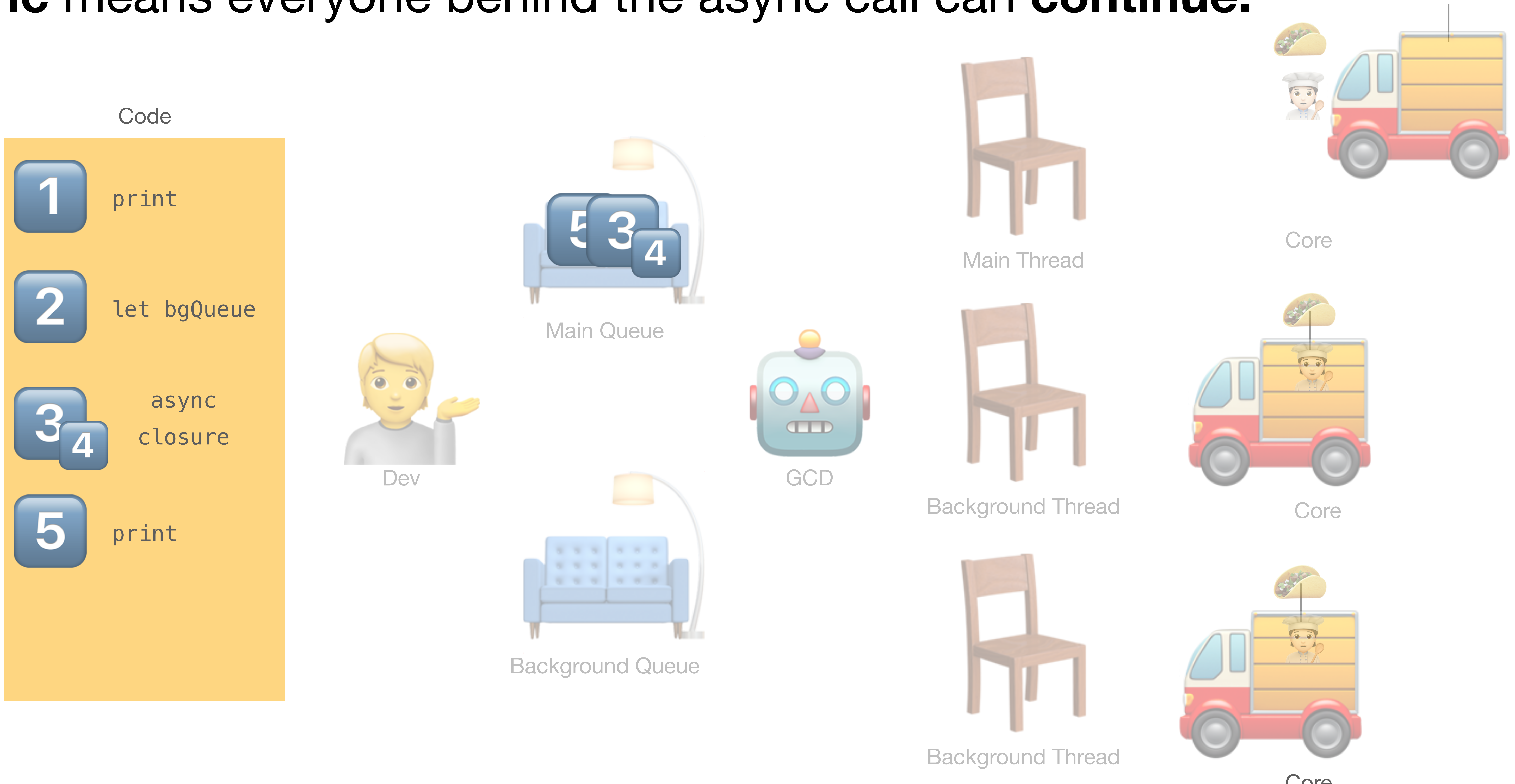
The Rules of GCD

Async means everyone behind the async call can **continue.**



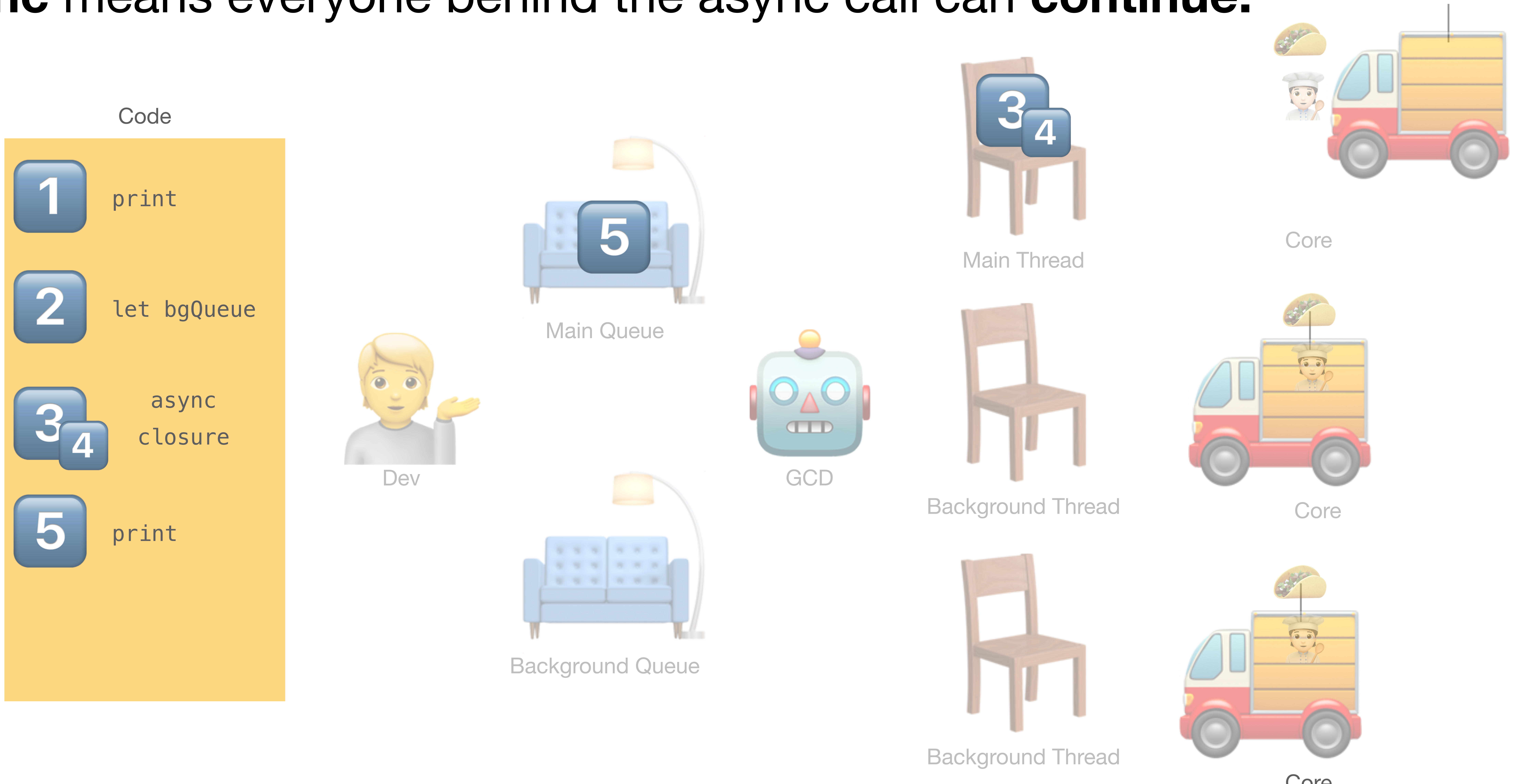
The Rules of GCD

Async means everyone behind the async call can **continue.**



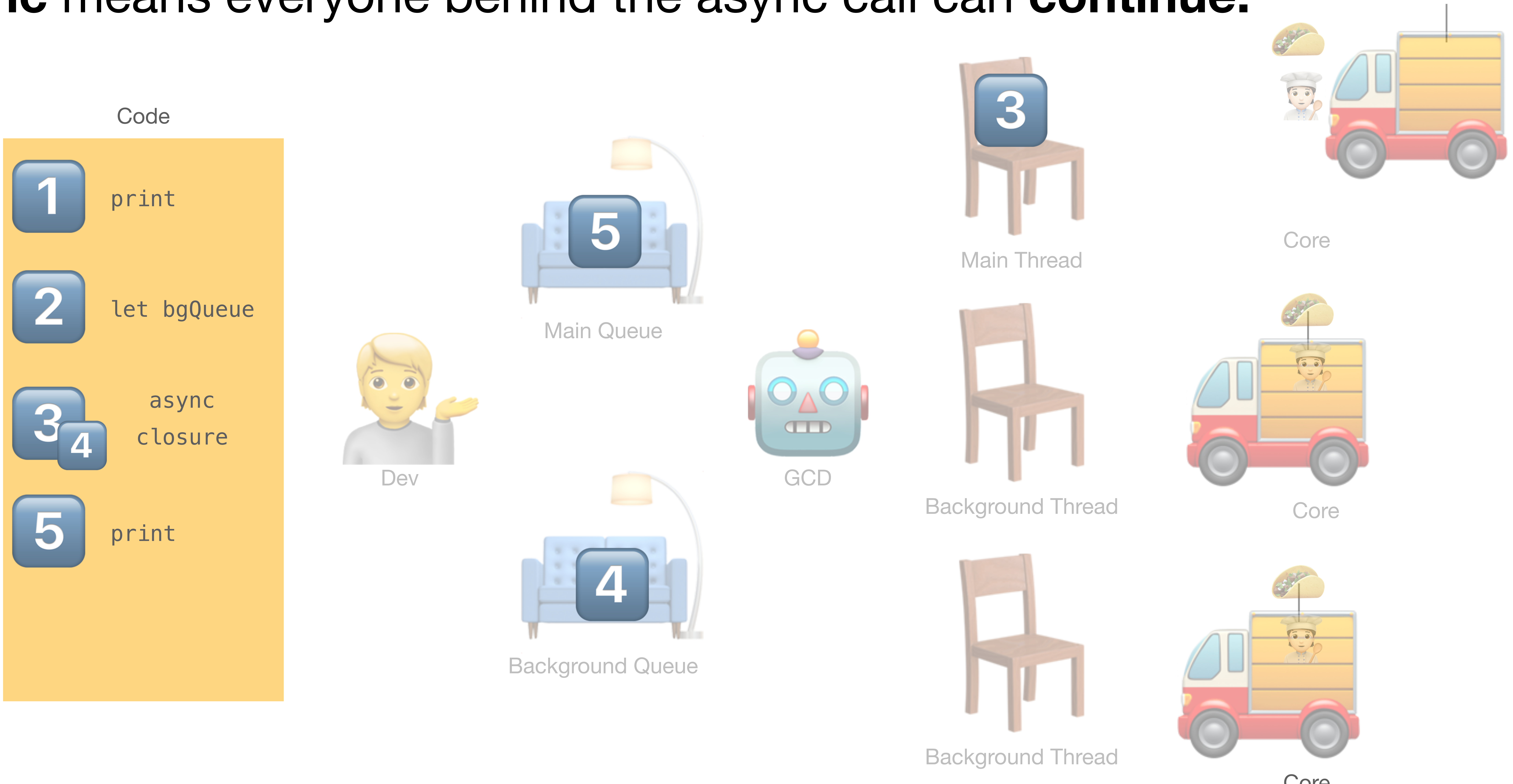
The Rules of GCD

Async means everyone behind the async call can **continue.**



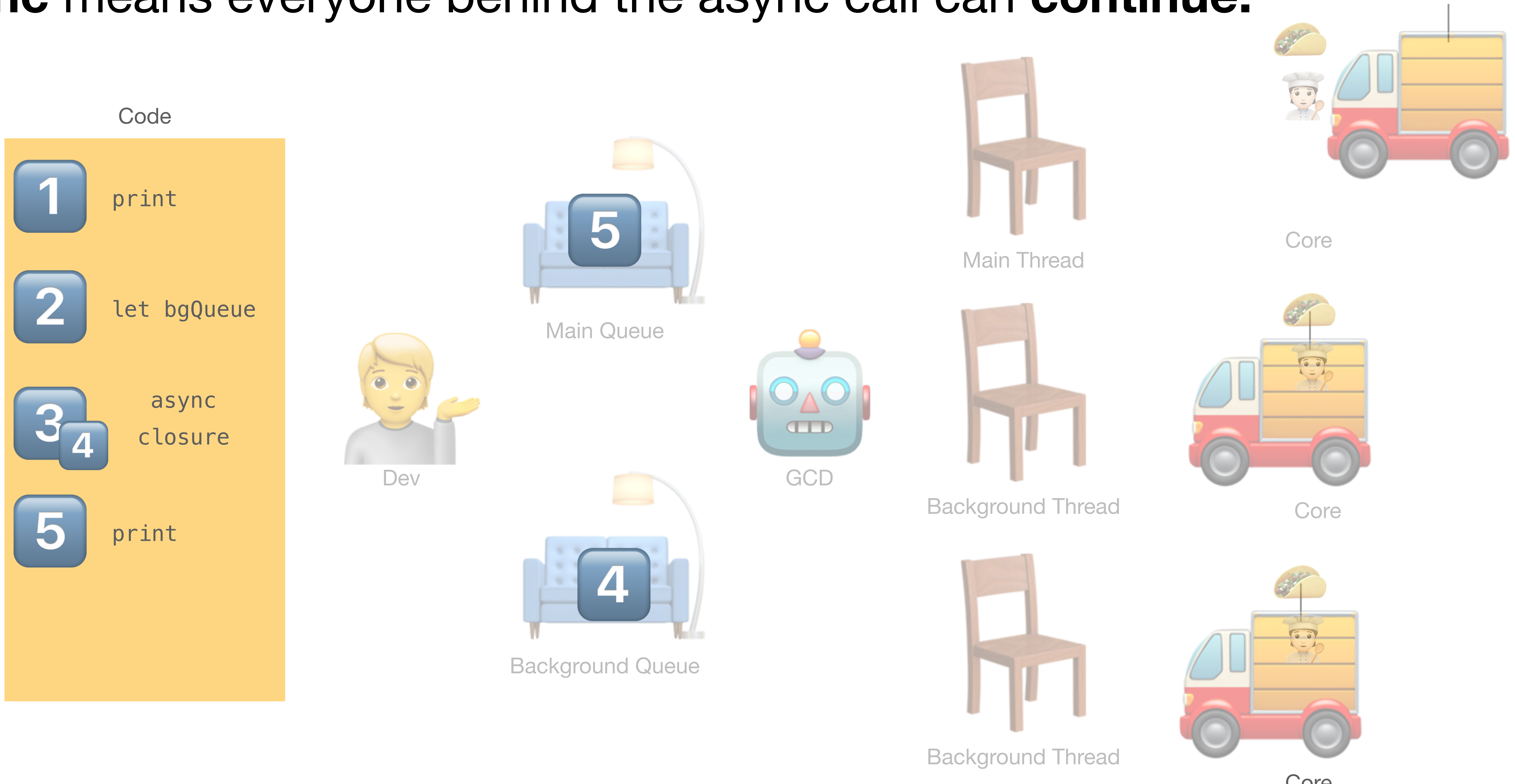
The Rules of GCD

Async means everyone behind the async call can **continue.**



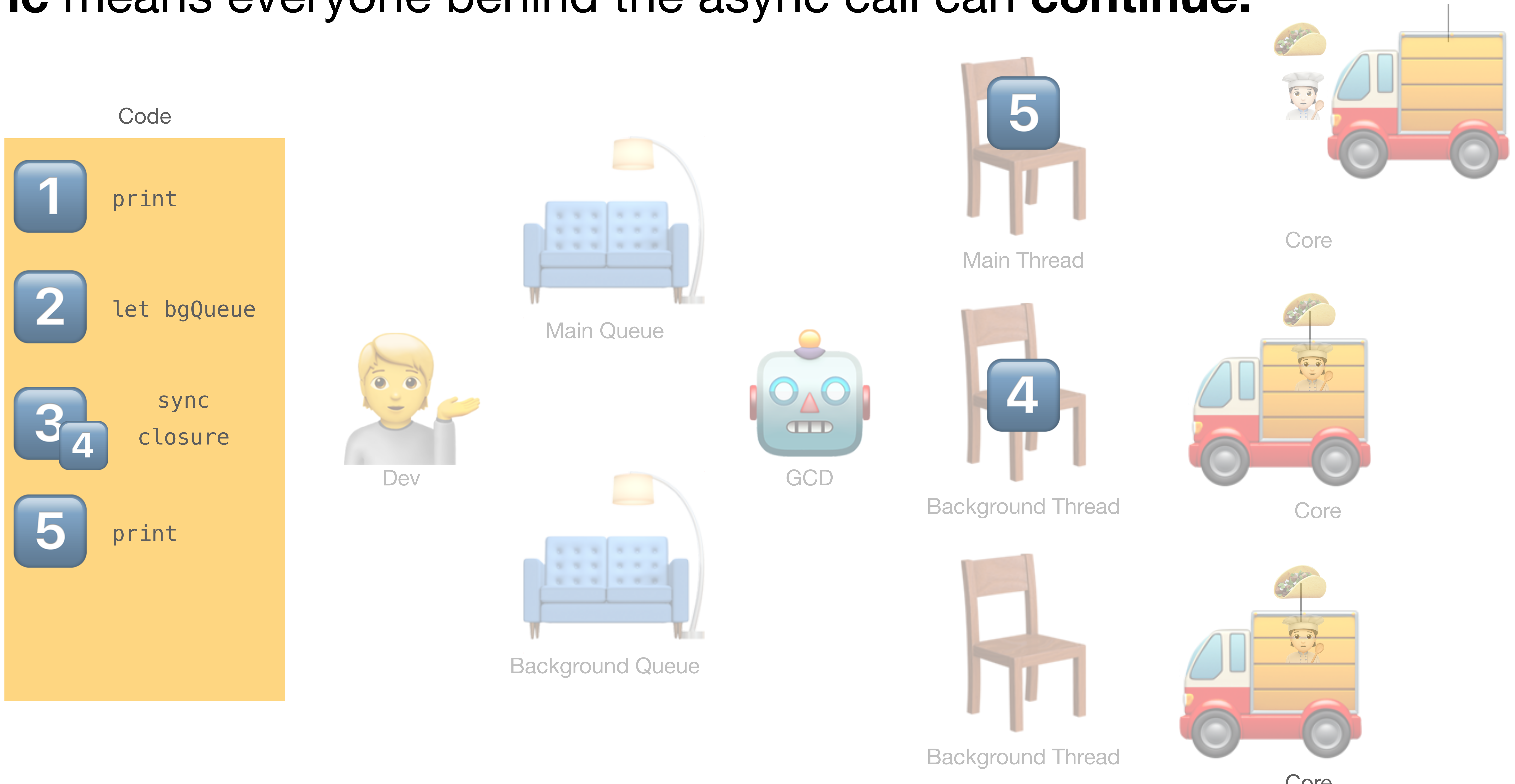
The Rules of GCD

Async means everyone behind the async call can **continue.**



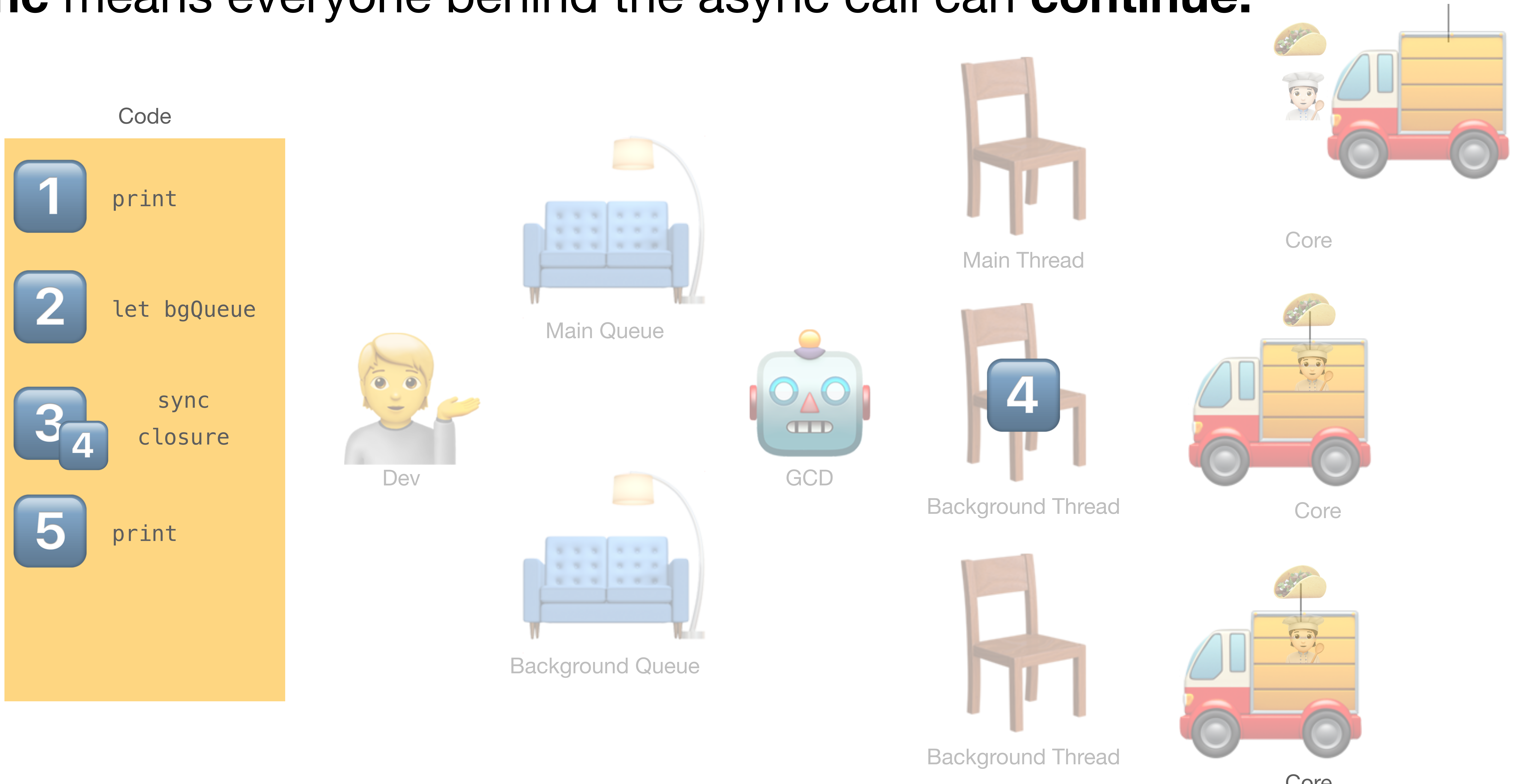
The Rules of GCD

Async means everyone behind the async call can **continue.**



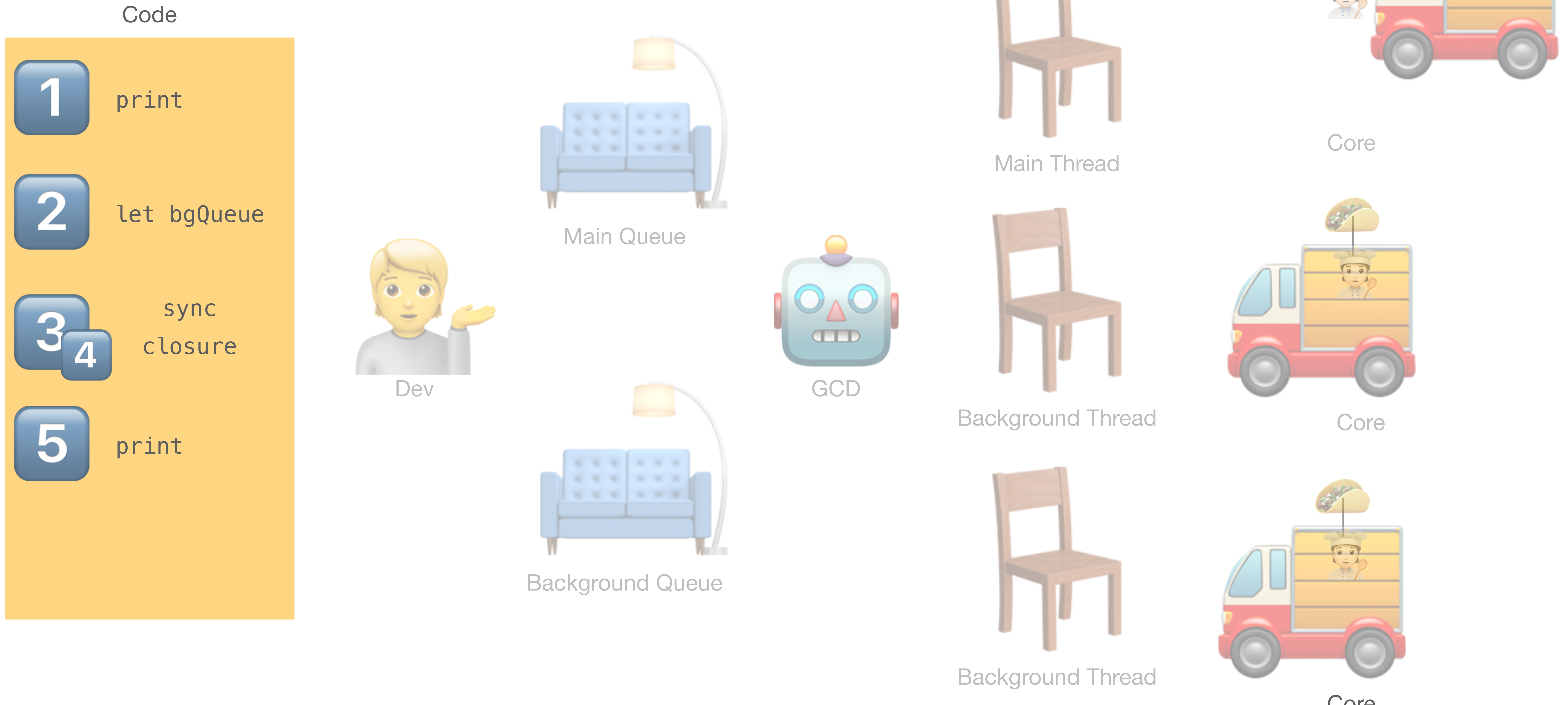
The Rules of GCD

Async means everyone behind the async call can **continue.**



The Rules of GCD

All done.

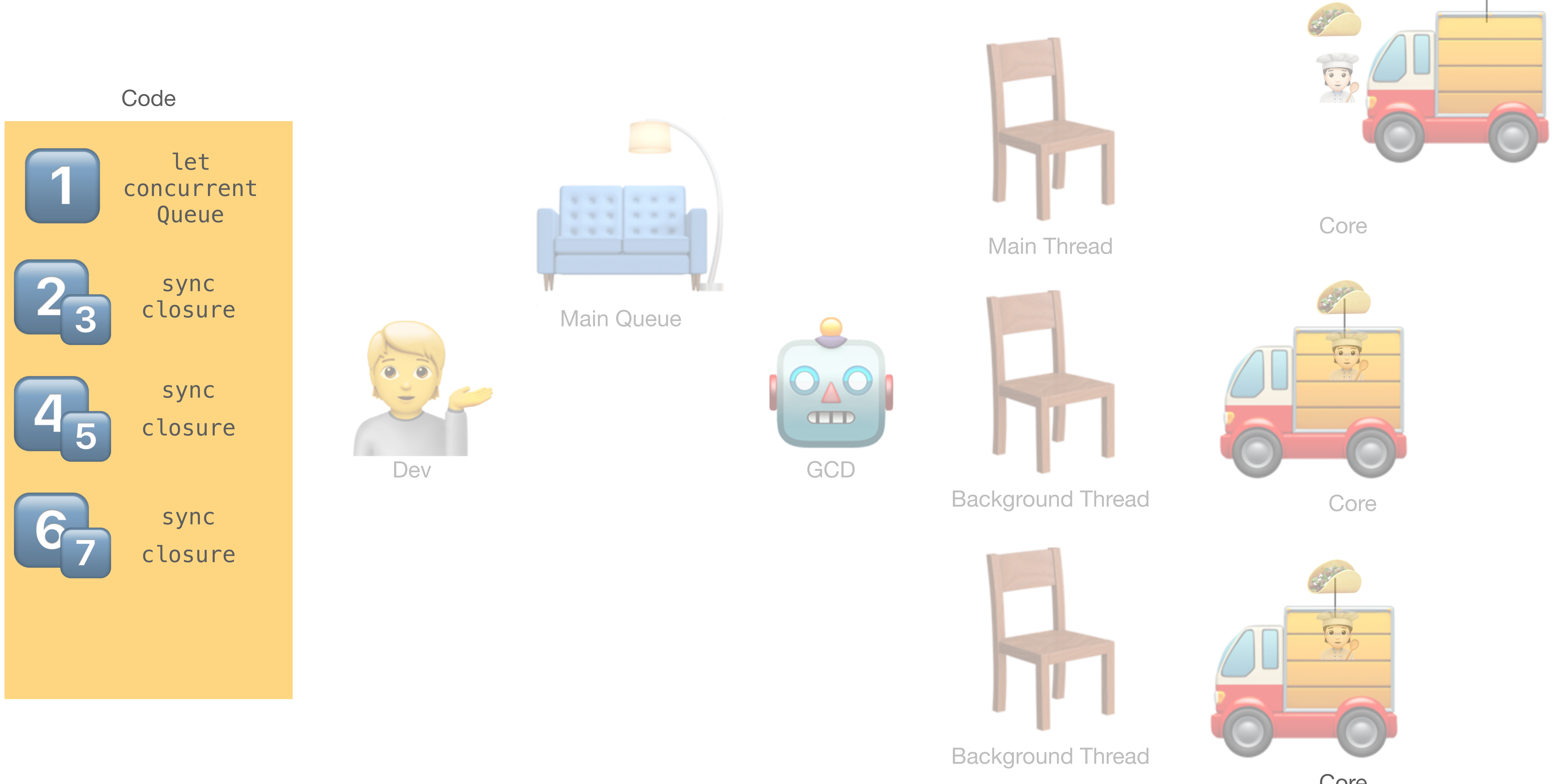


Live Demo

The fourth horseman: Concurrent

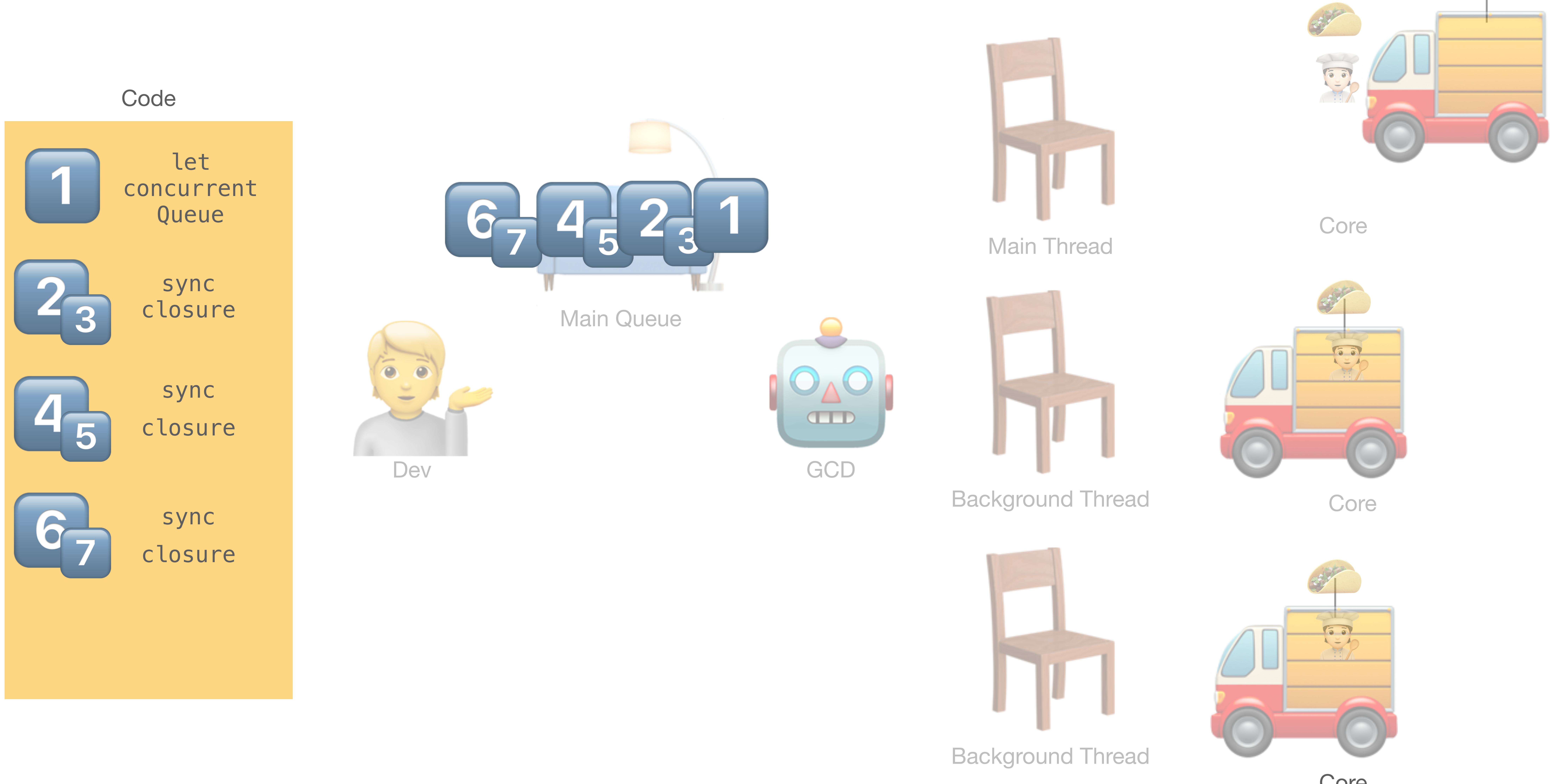
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



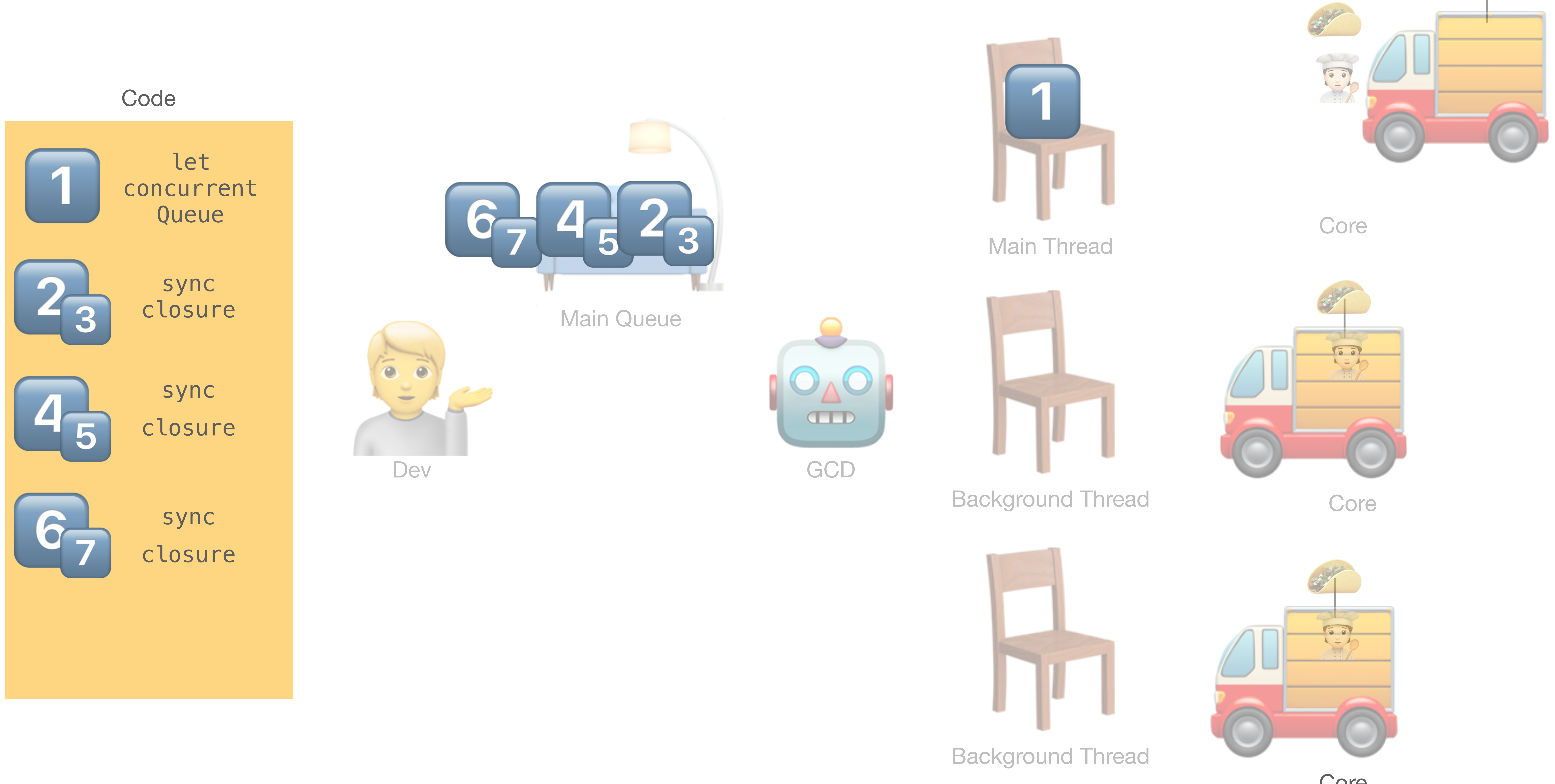
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



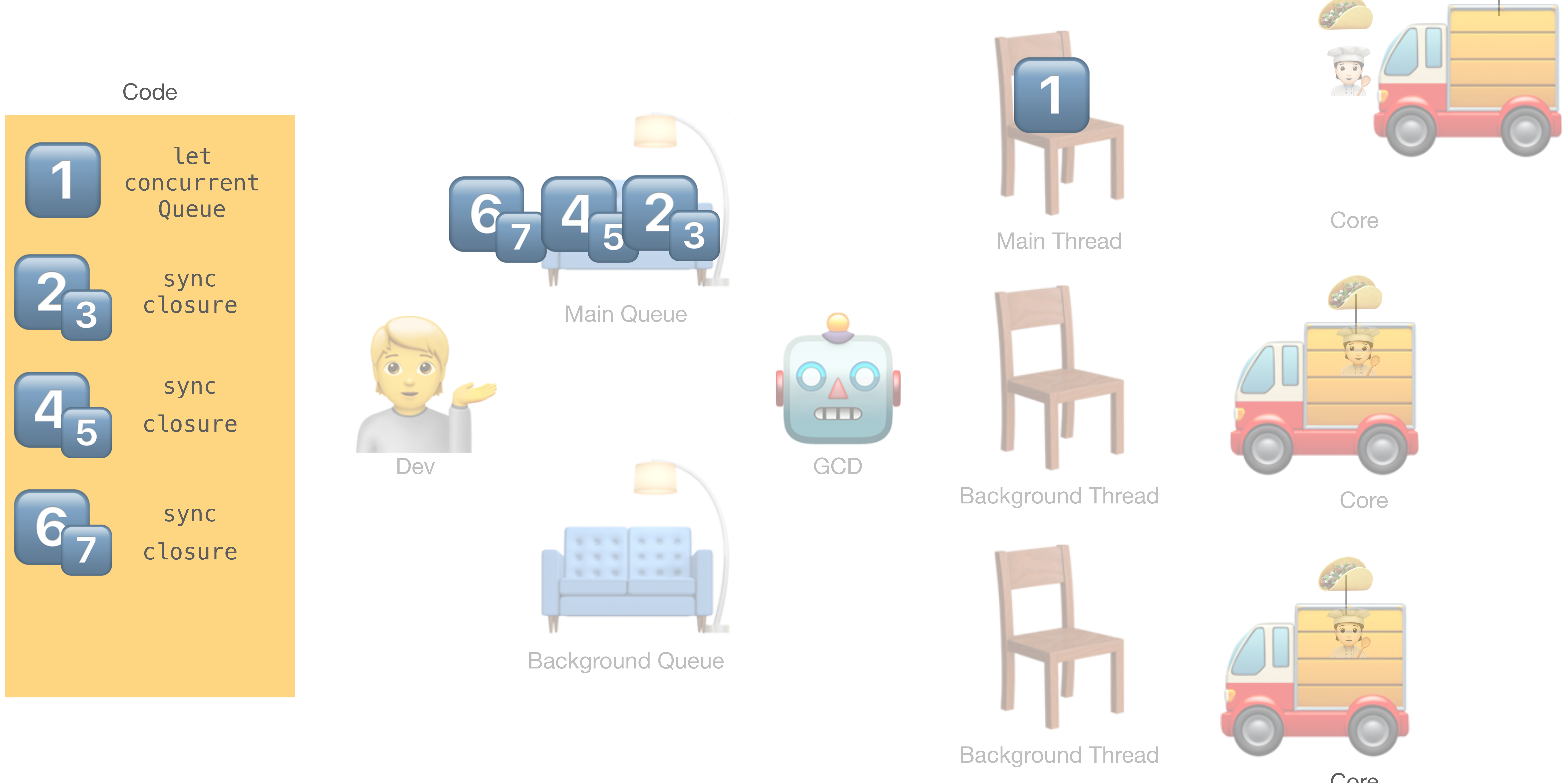
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



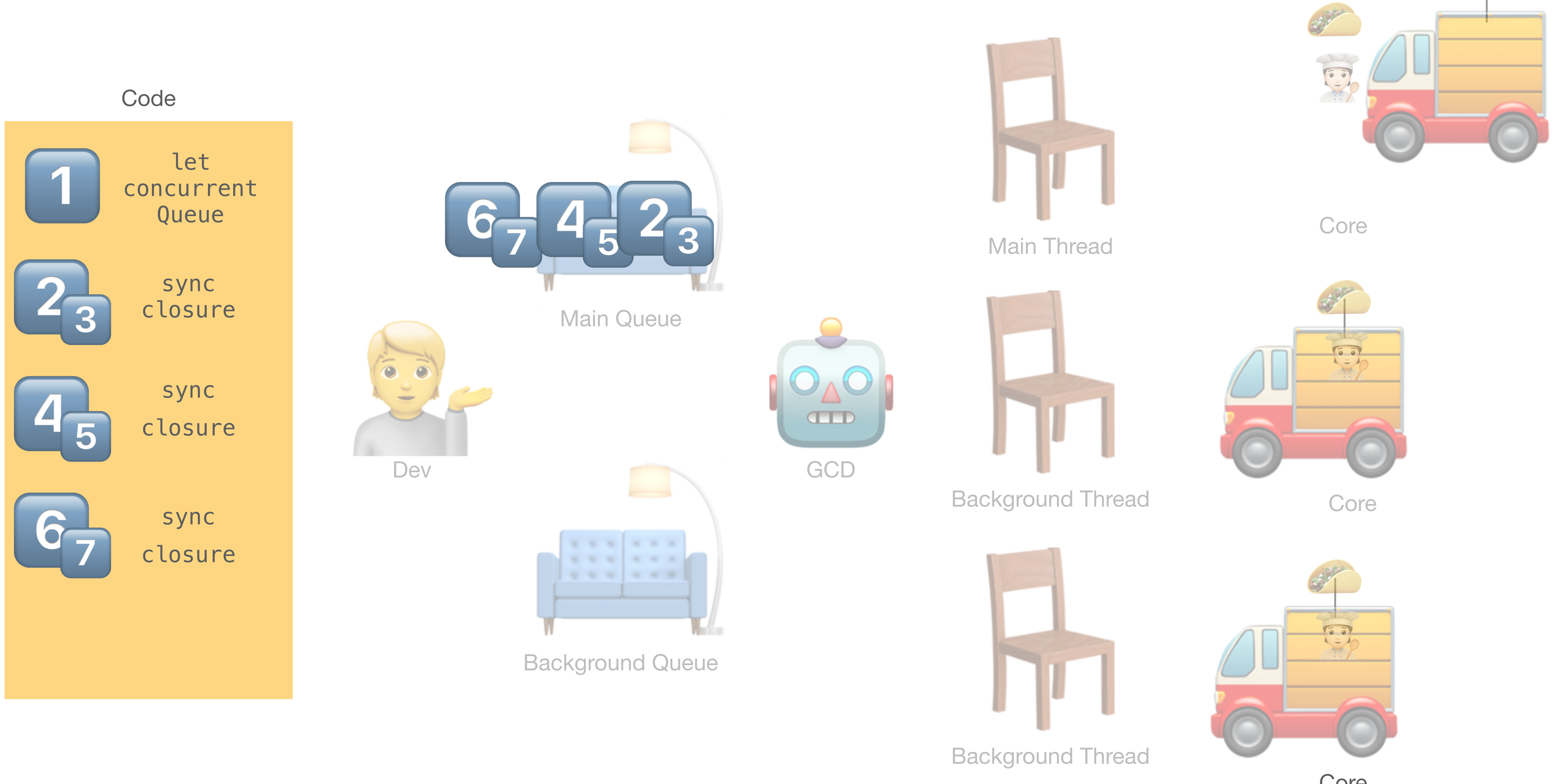
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



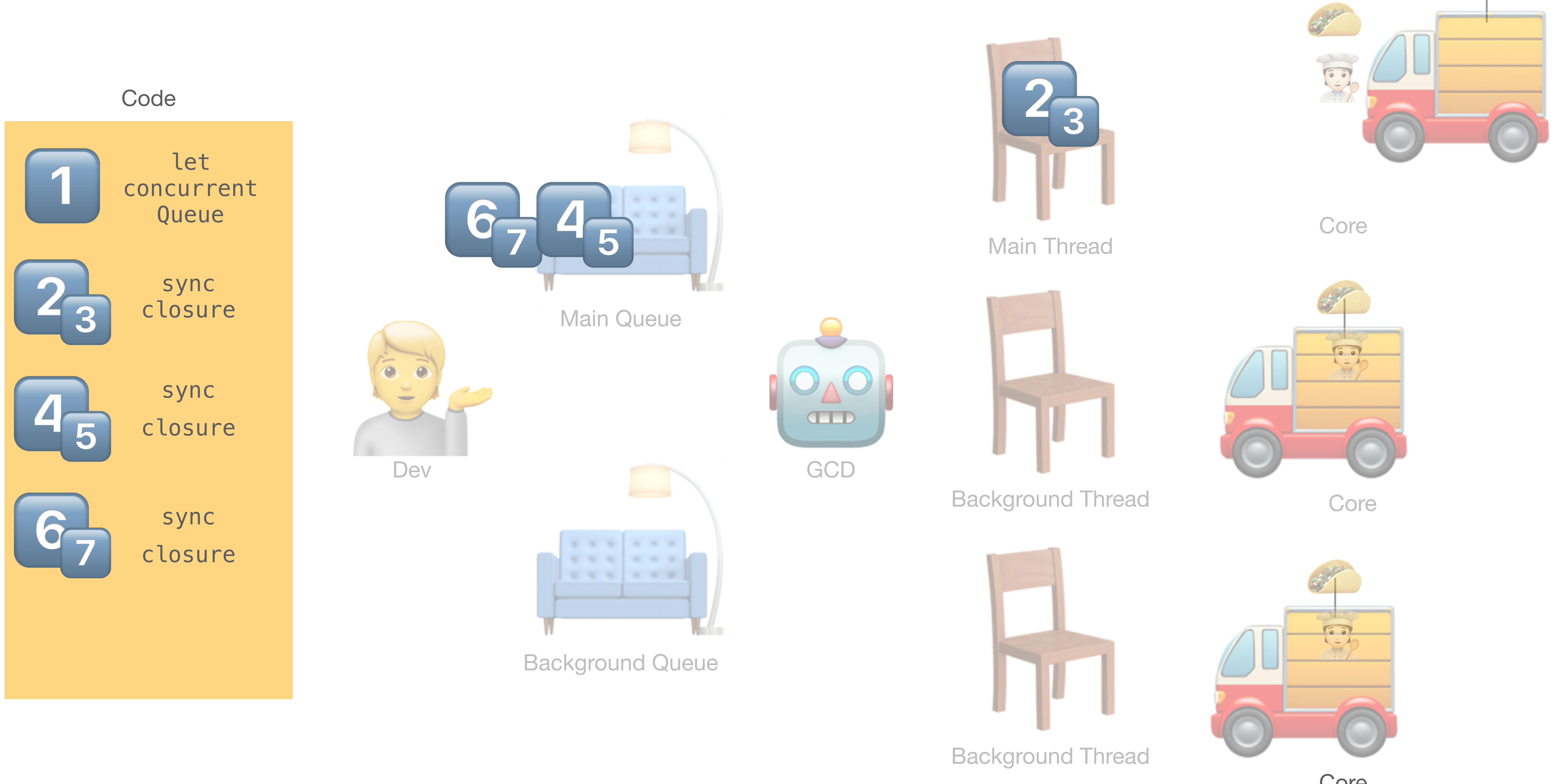
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



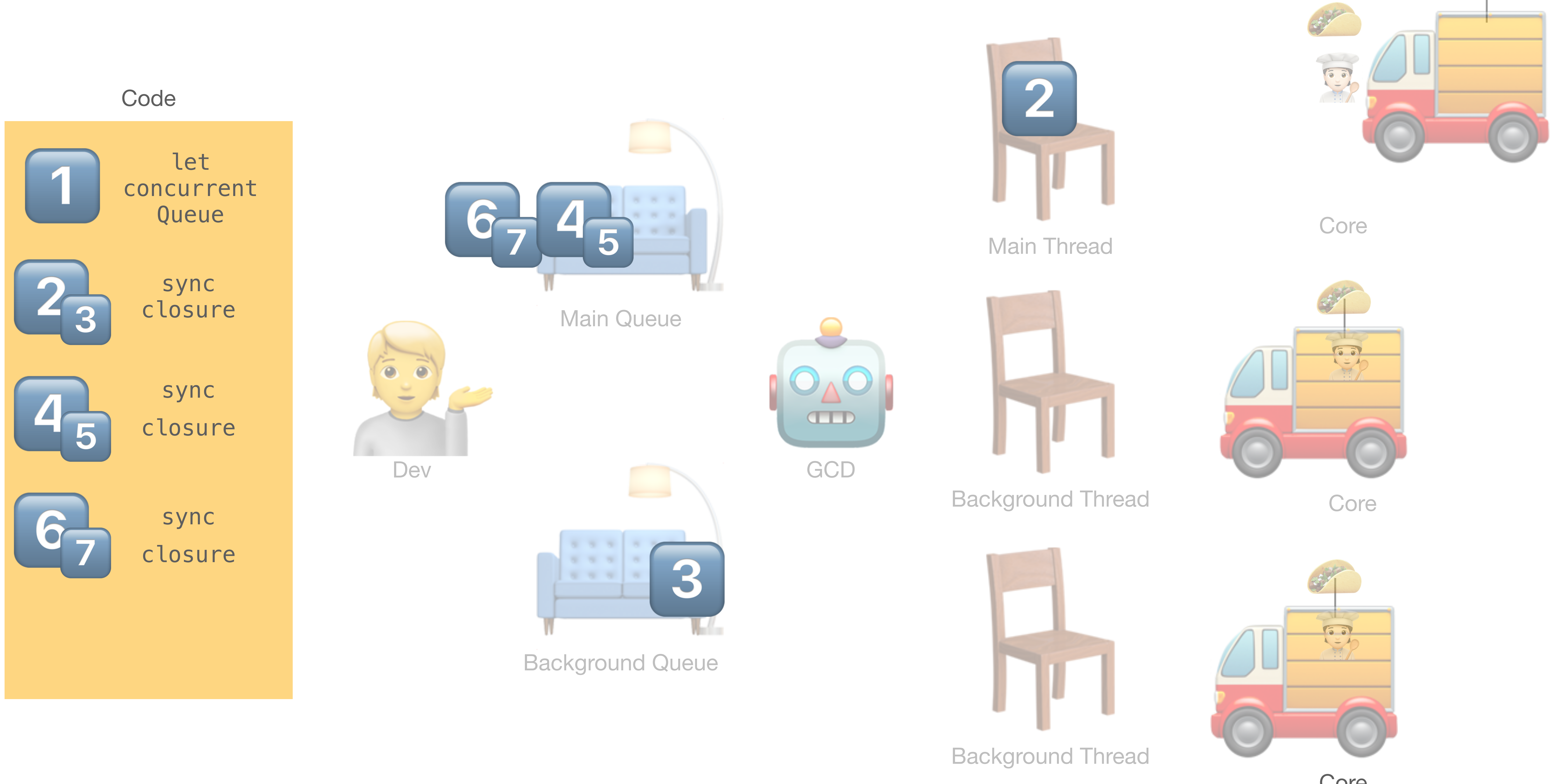
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



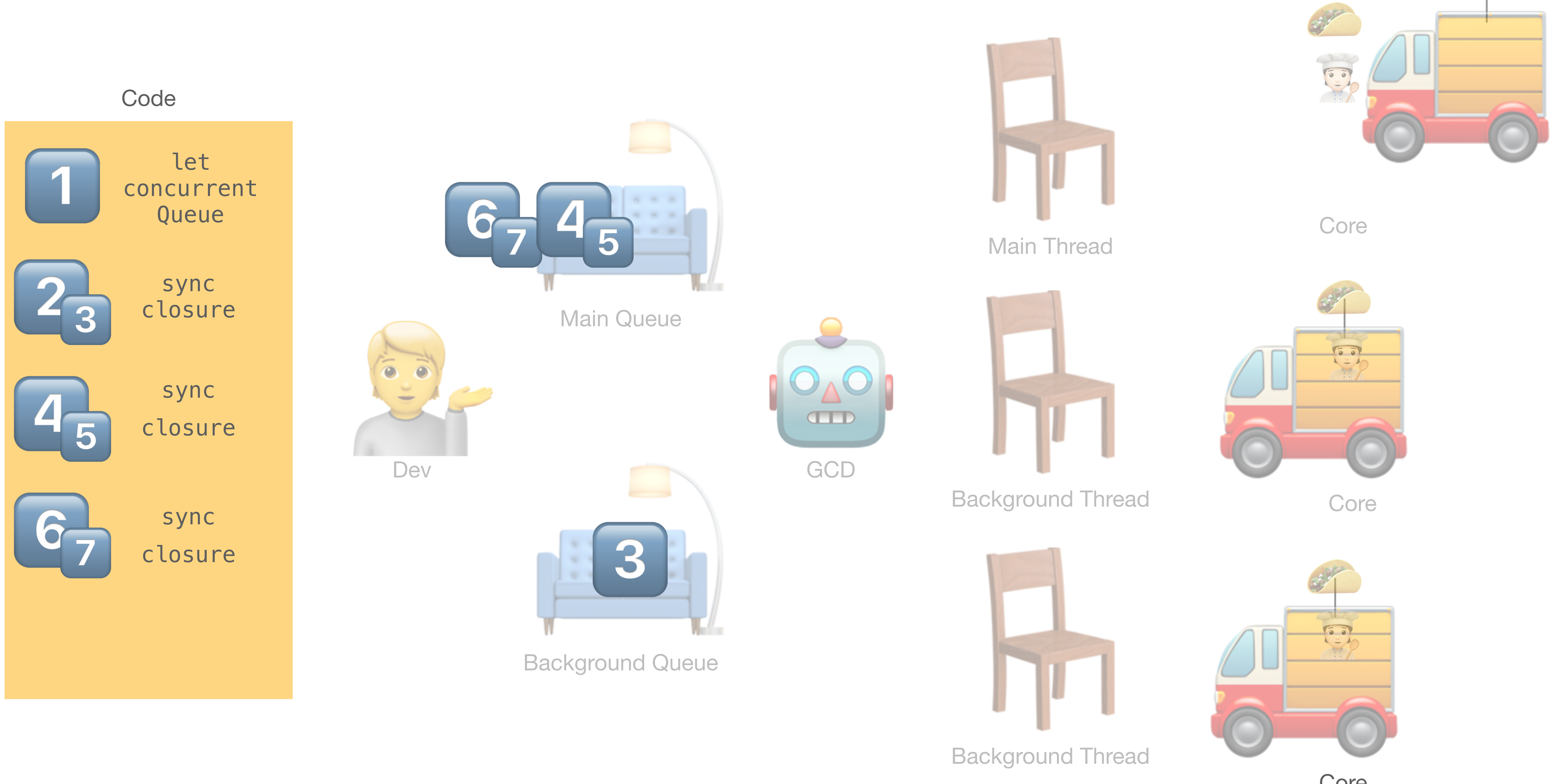
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



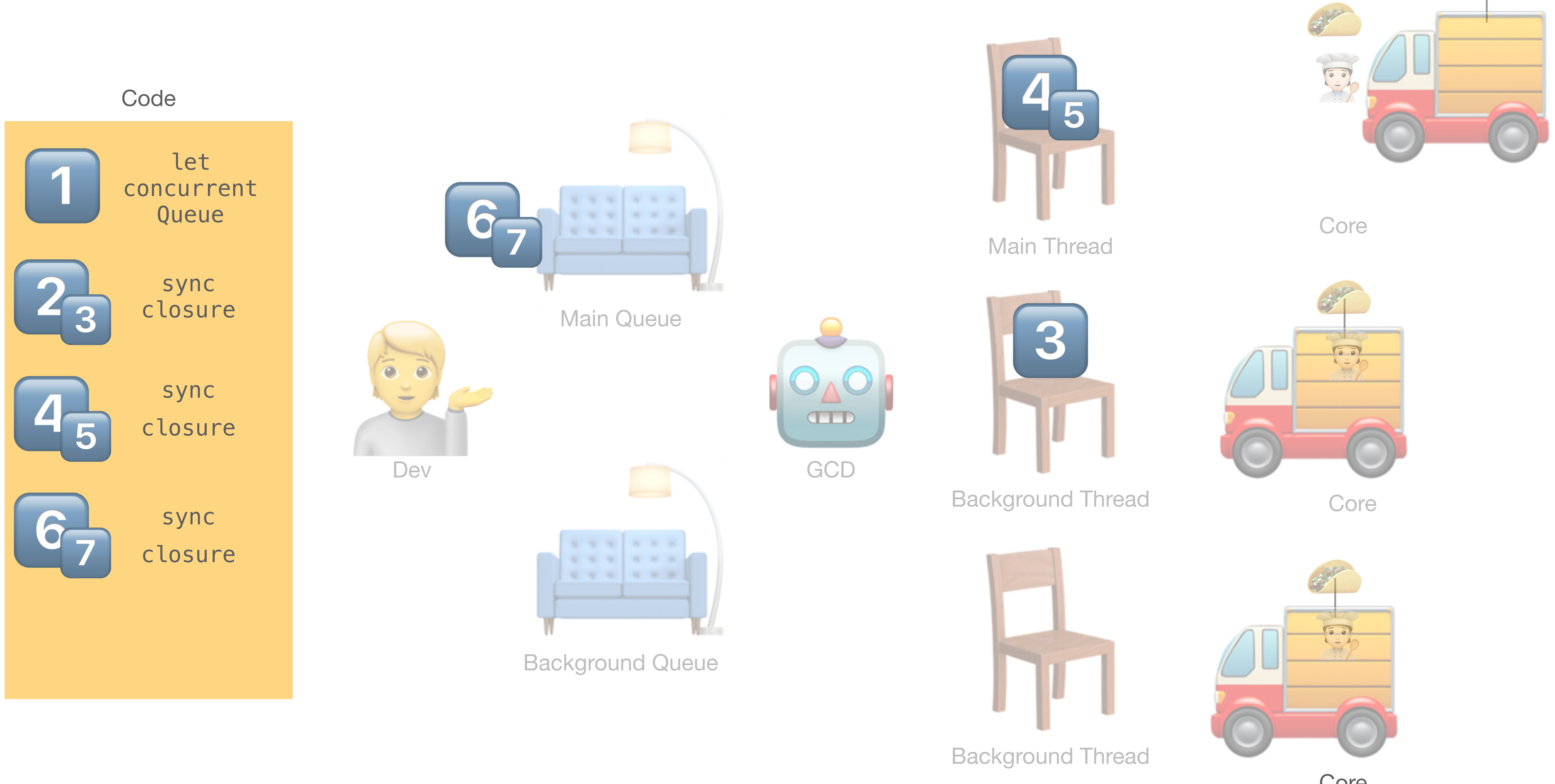
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



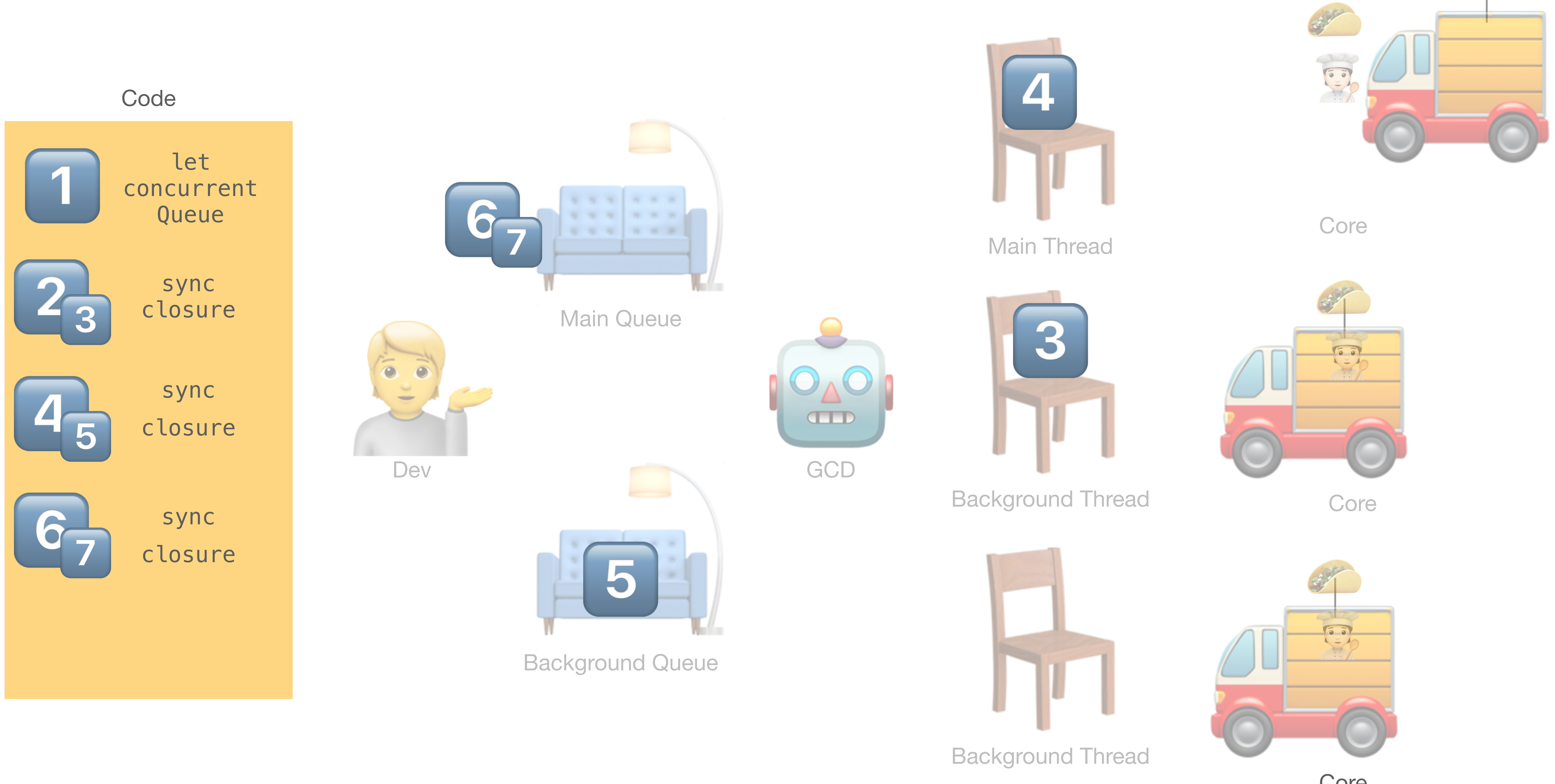
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



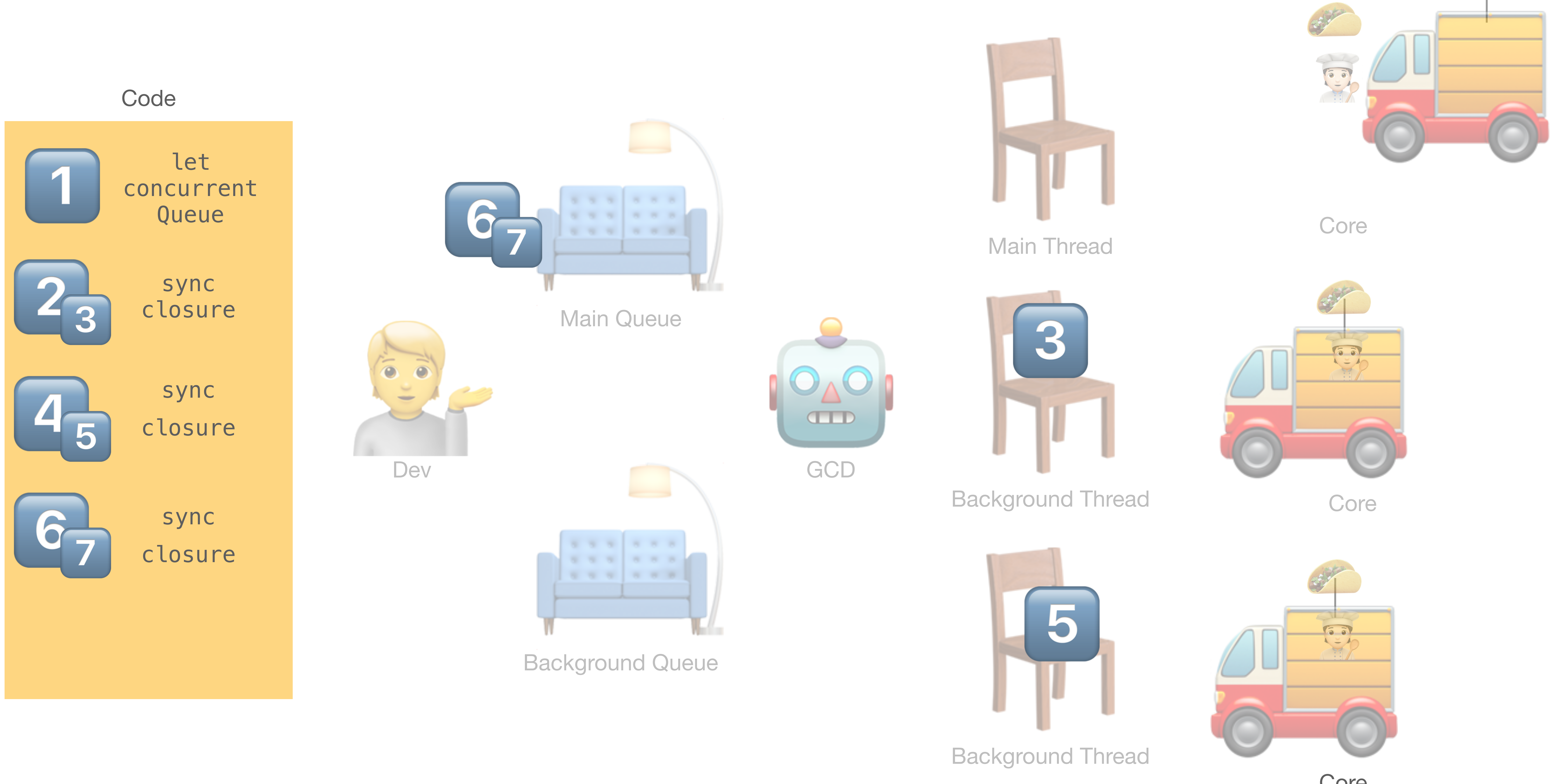
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



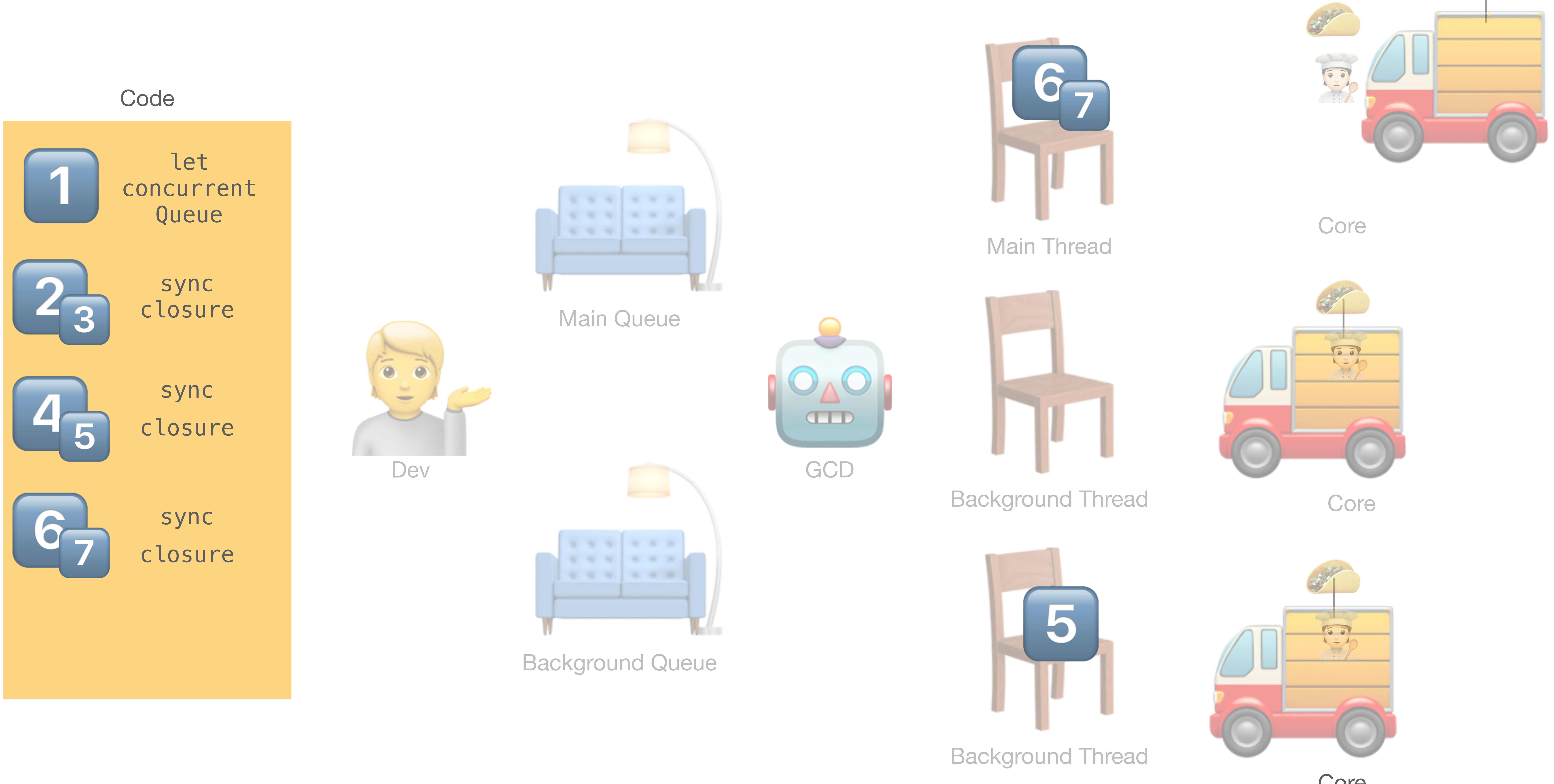
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



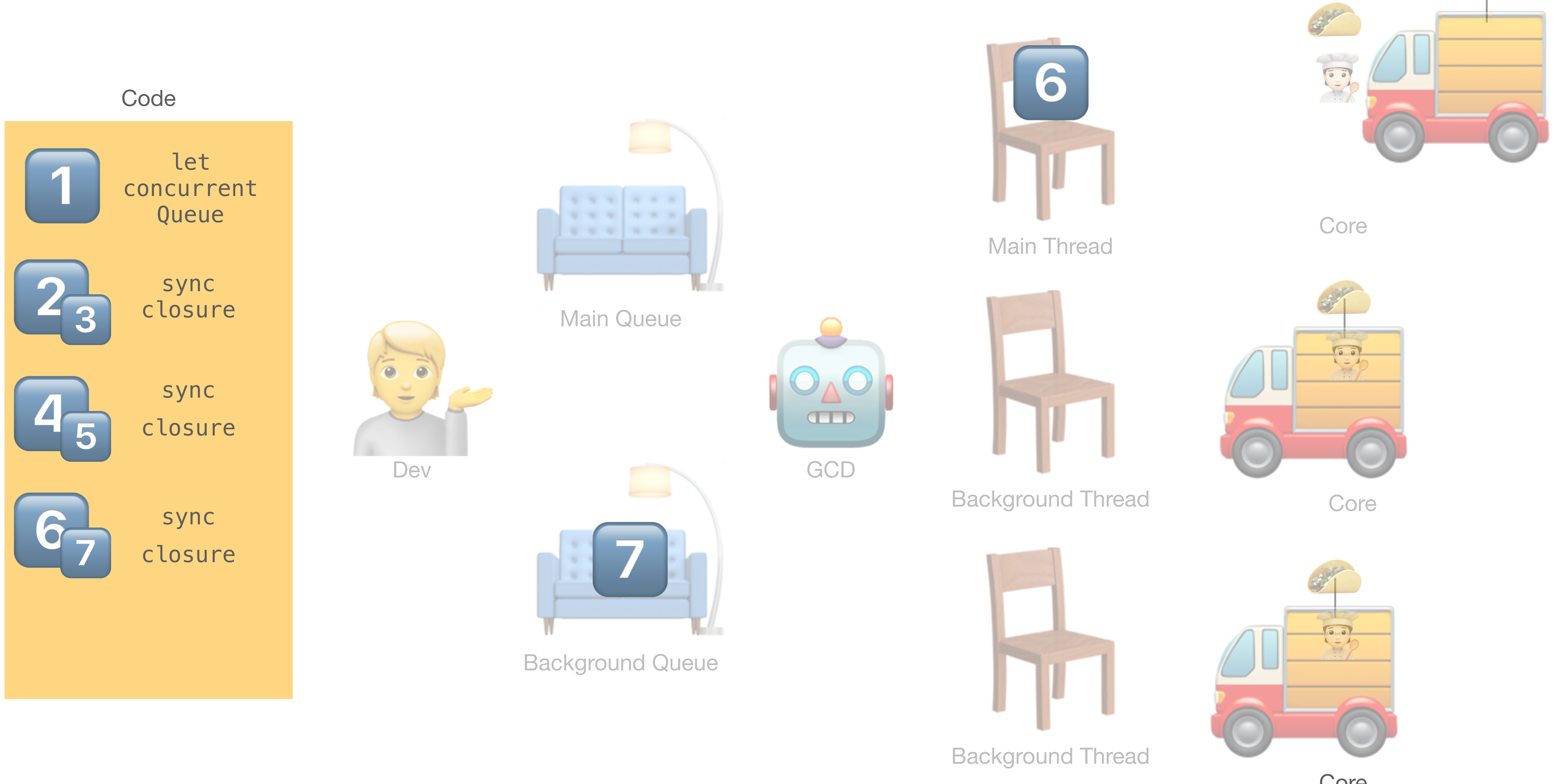
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



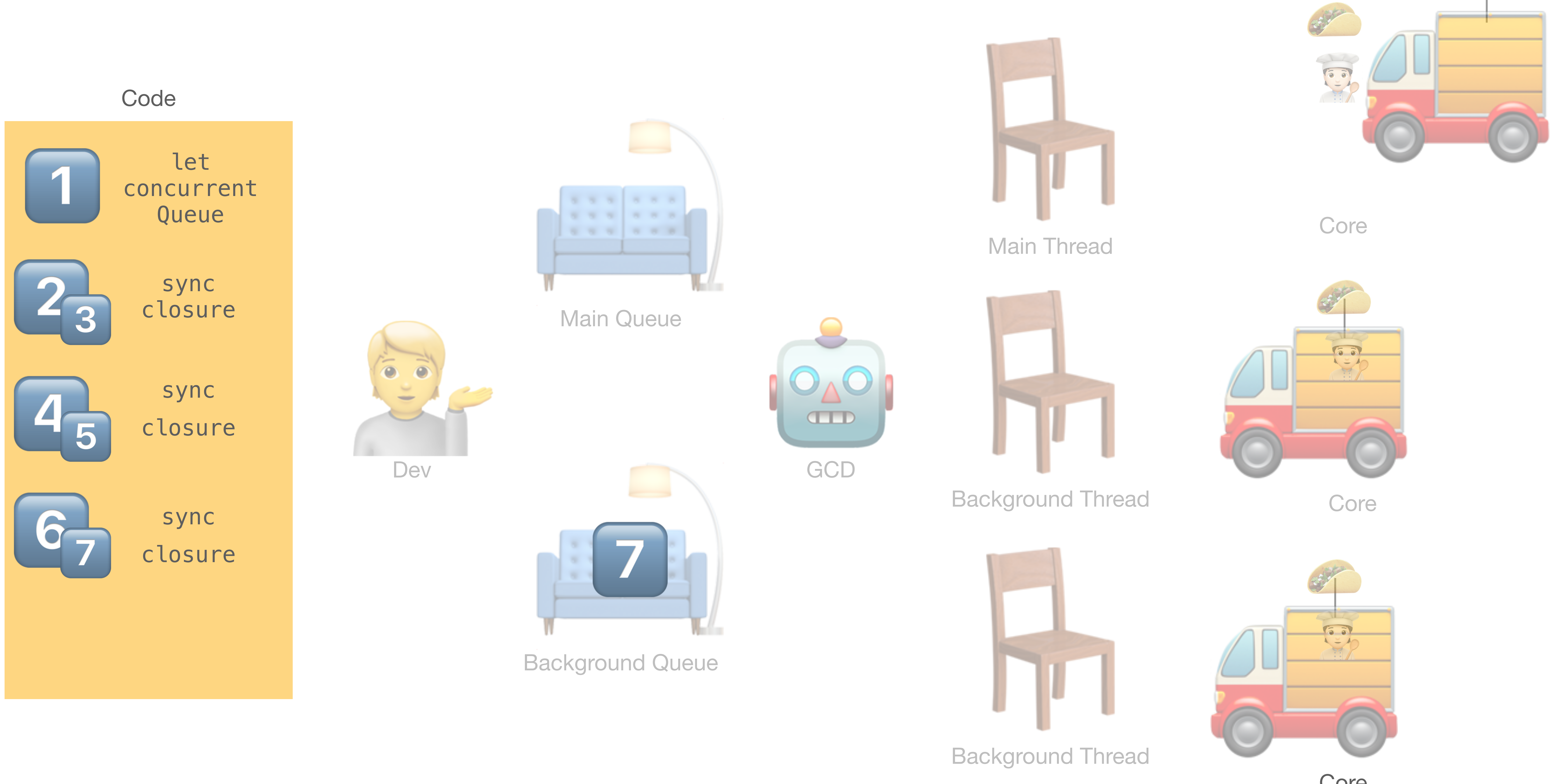
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



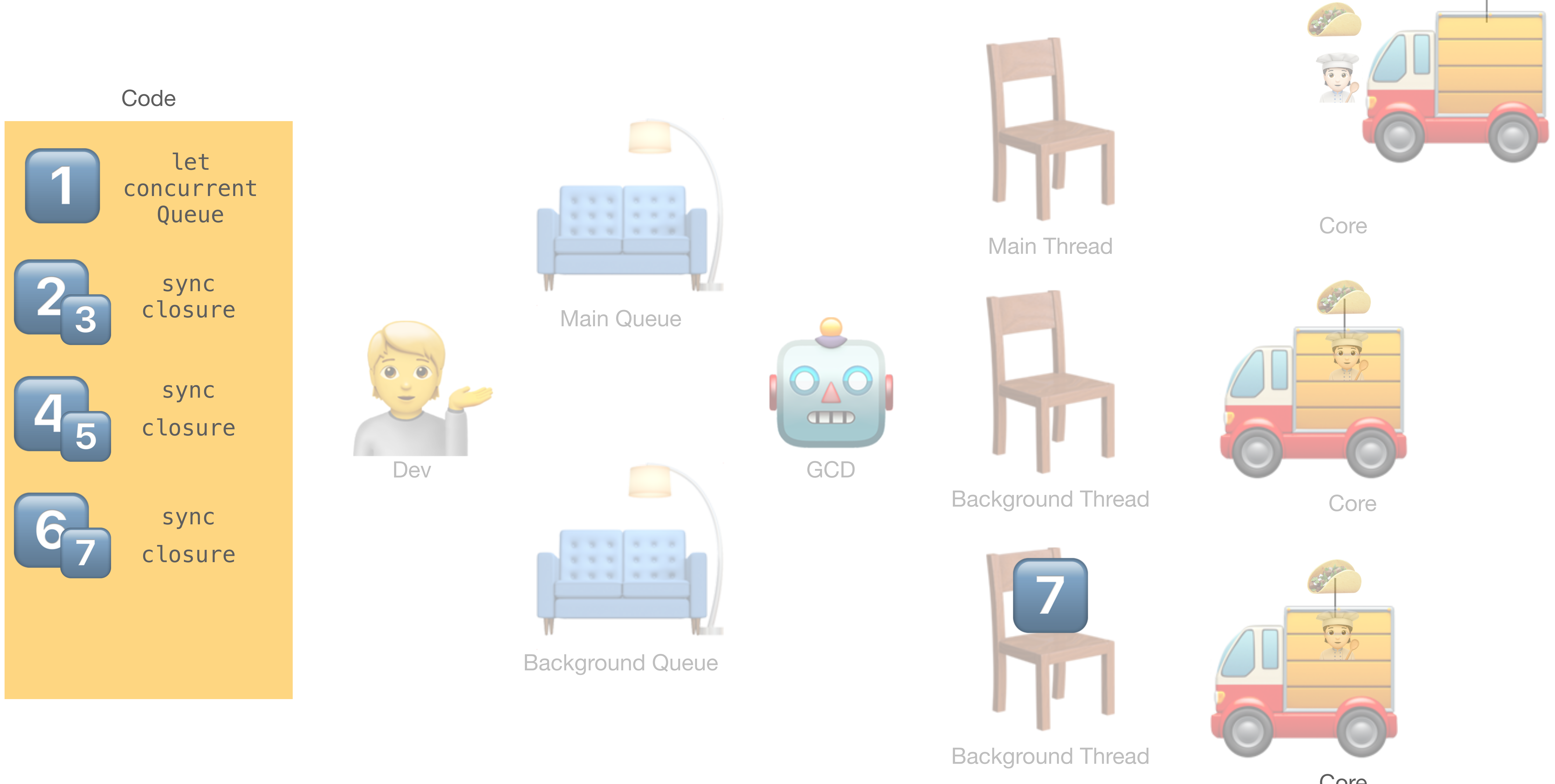
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



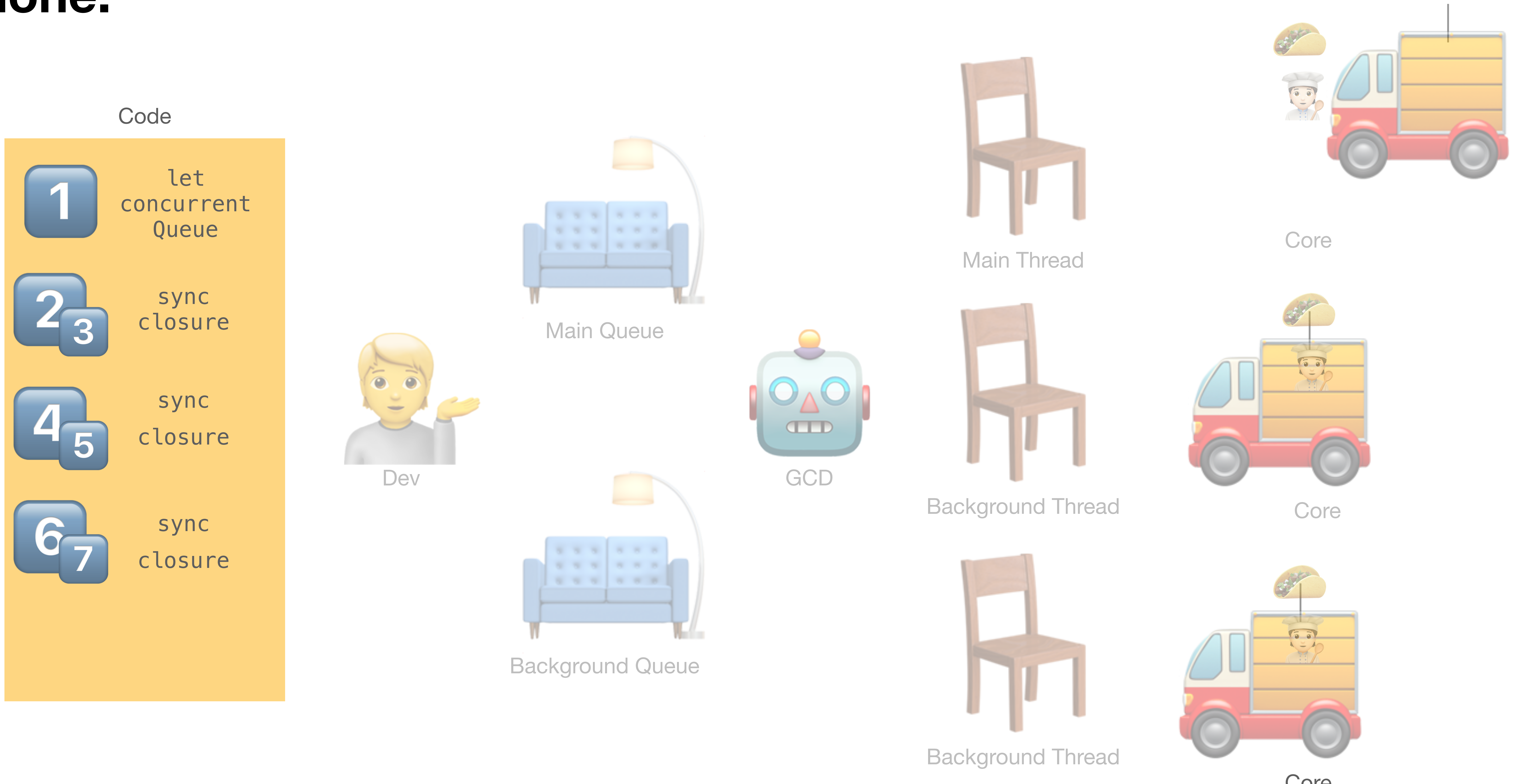
The Rules of GCD

Concurrent means a block can only begin when the previous block **begins.**



The Rules of GCD

All done.



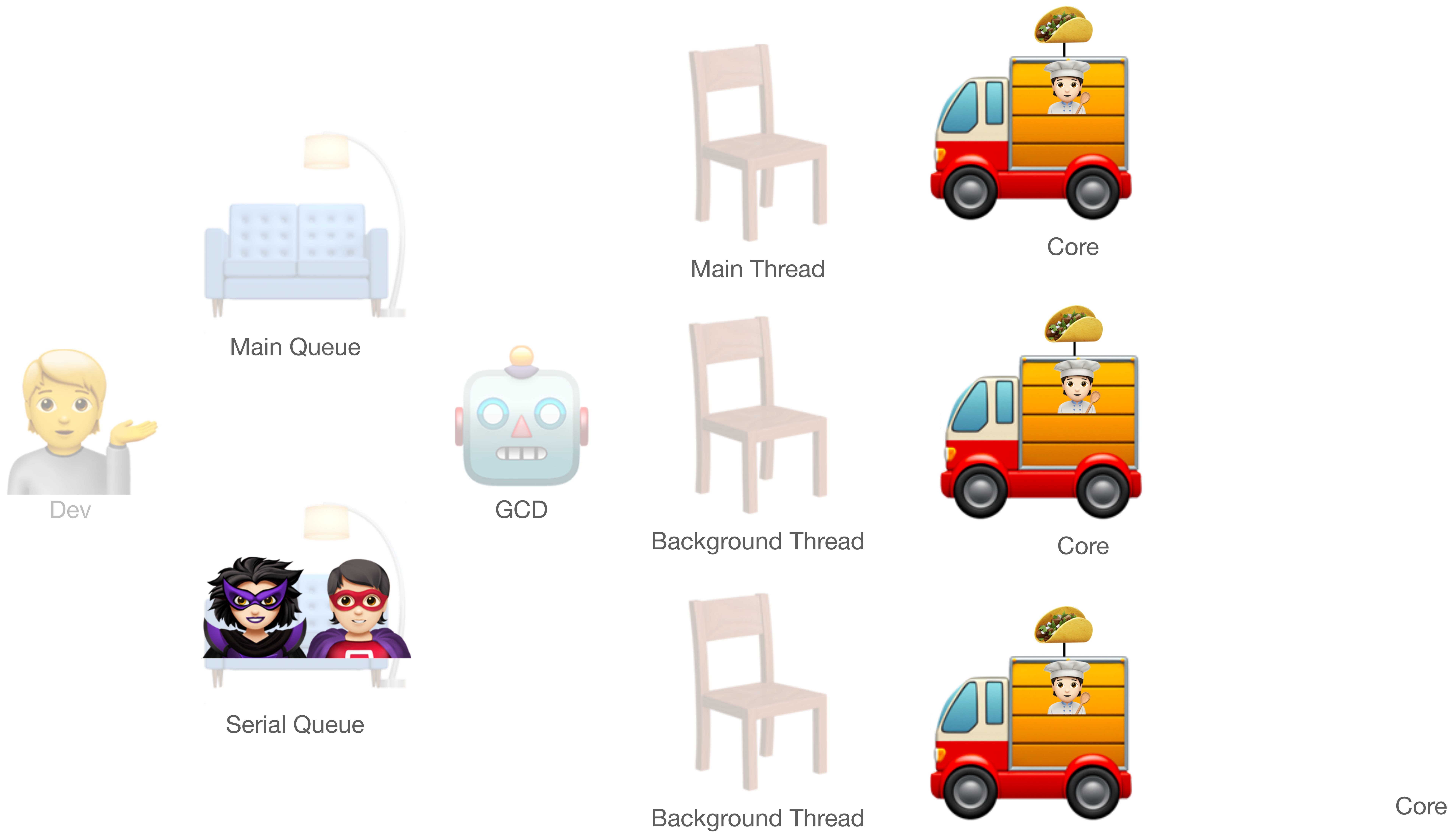
What is GCD?

In practice, it's easi-*er* concurrency

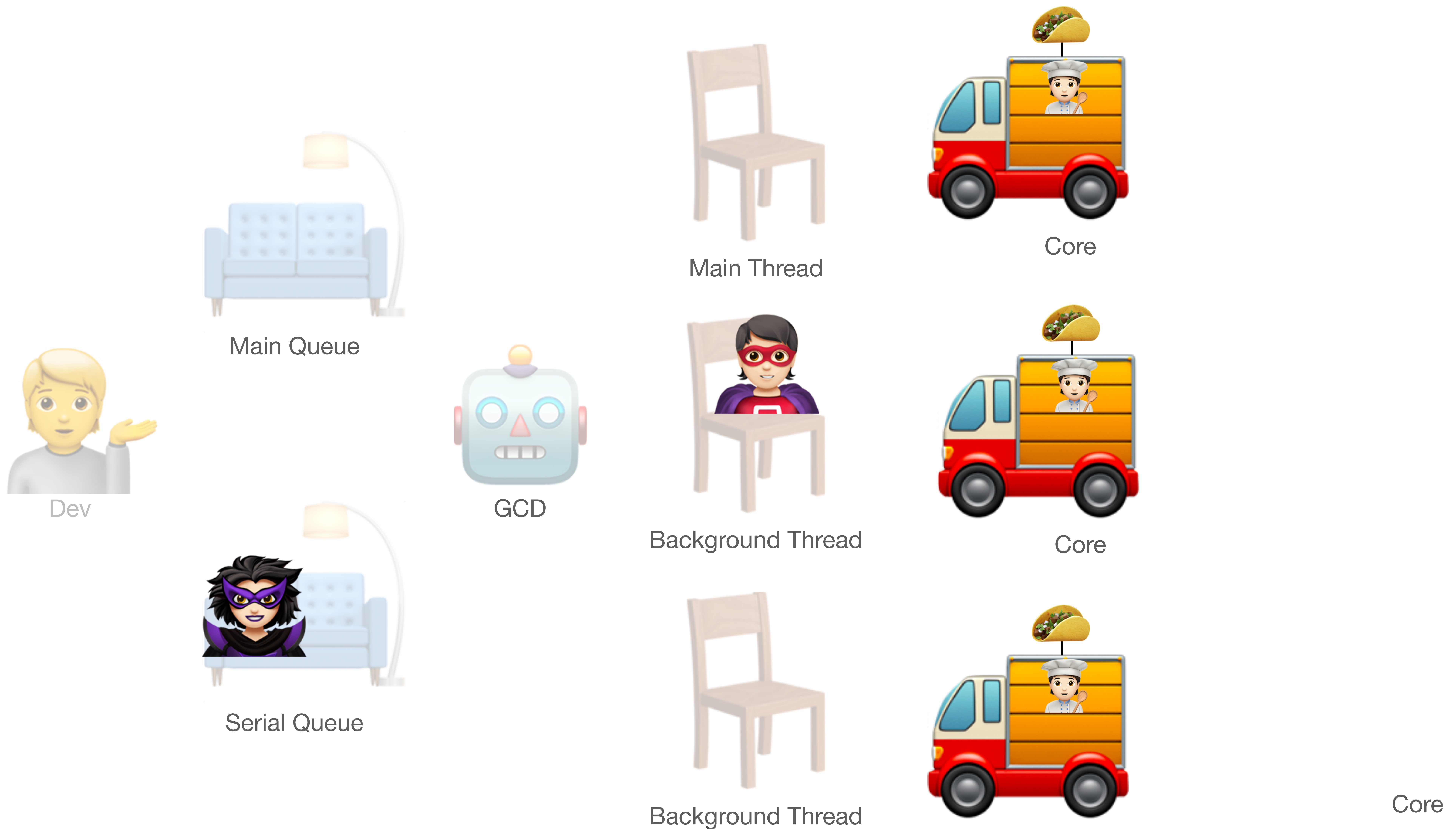
- Understanding concurrency is still a prerequisite. GCD will only help avoid the most common scenarios.
- GCD provides practical APIs that help you handle queues, not threads. You are not guaranteed a specific thread for your closure, *except for the main thread*.
- The bugs are still there: deadlocking, livelocking, resource starvation...
- Stay as far away from GCD as possible. We've known it since 1974 (and maybe earlier): "Premature optimization is the root of all evil (or at least most of it) in programming." - Donald Knuth

Quiz

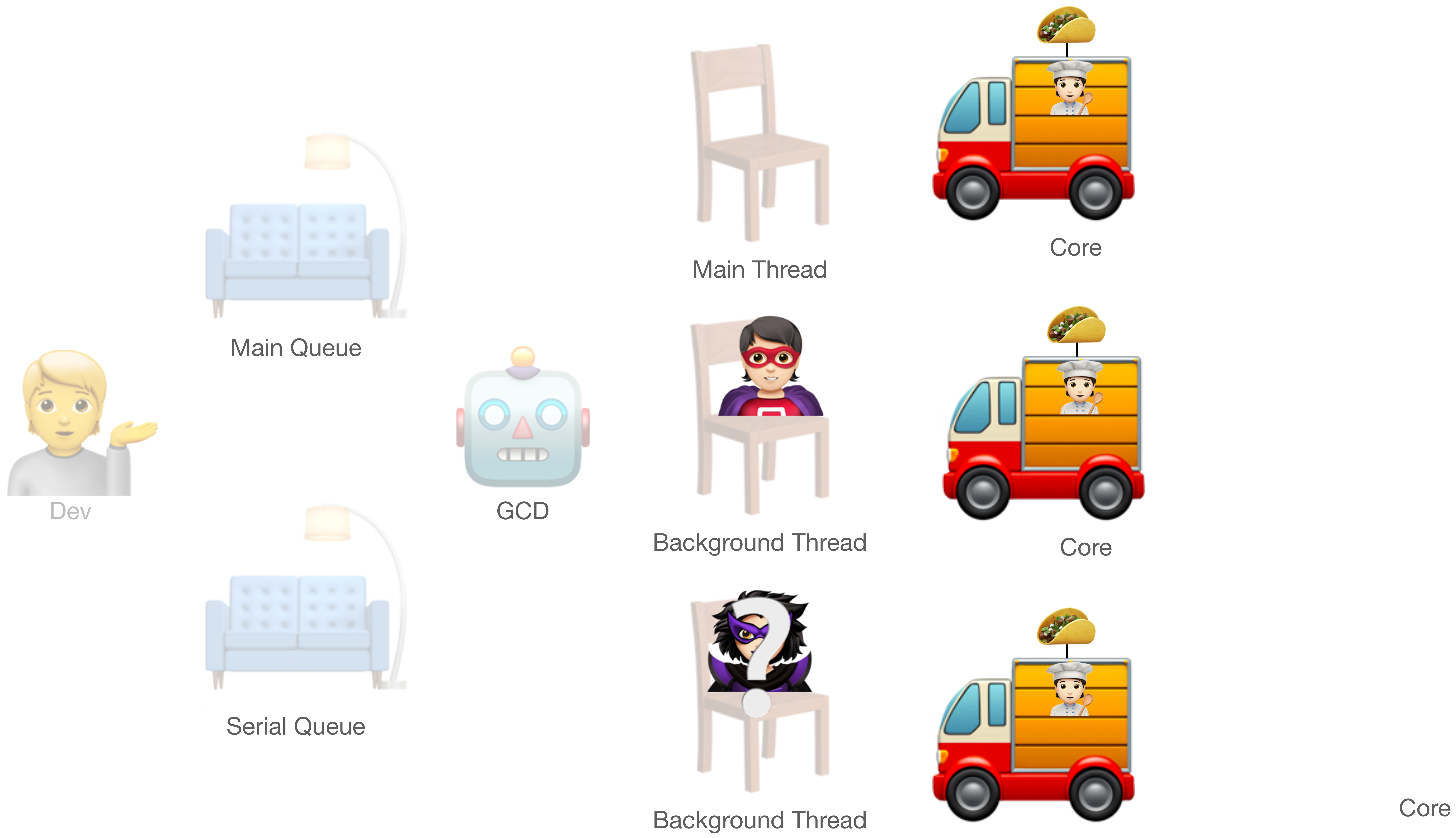
If 🦸 is sync, does 🦹 begin on the other thread?



If 🦸 is sync, does 🦹 begin on the other thread?

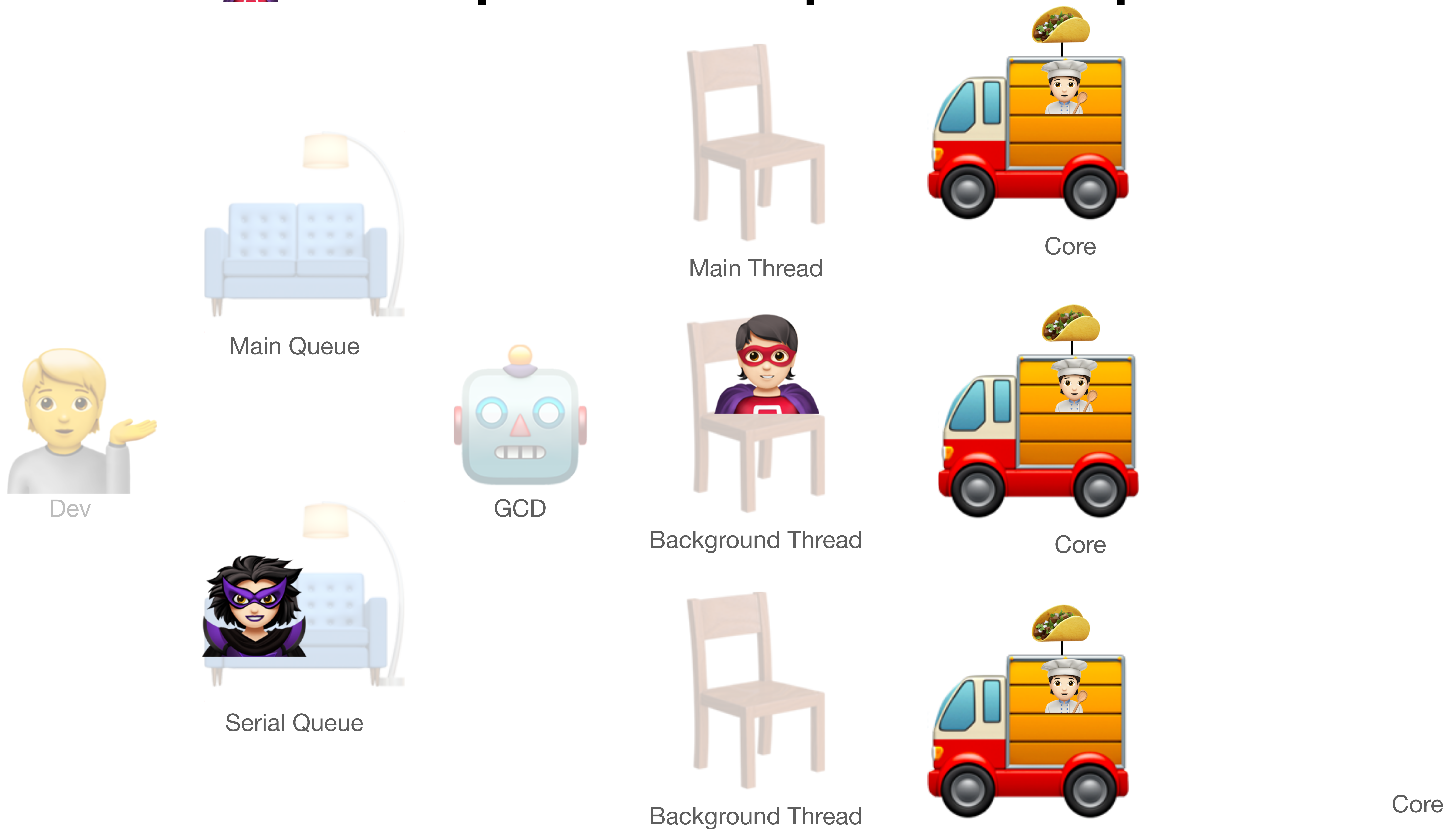


If 🦸 is sync, does 🦹 begin on the other thread?

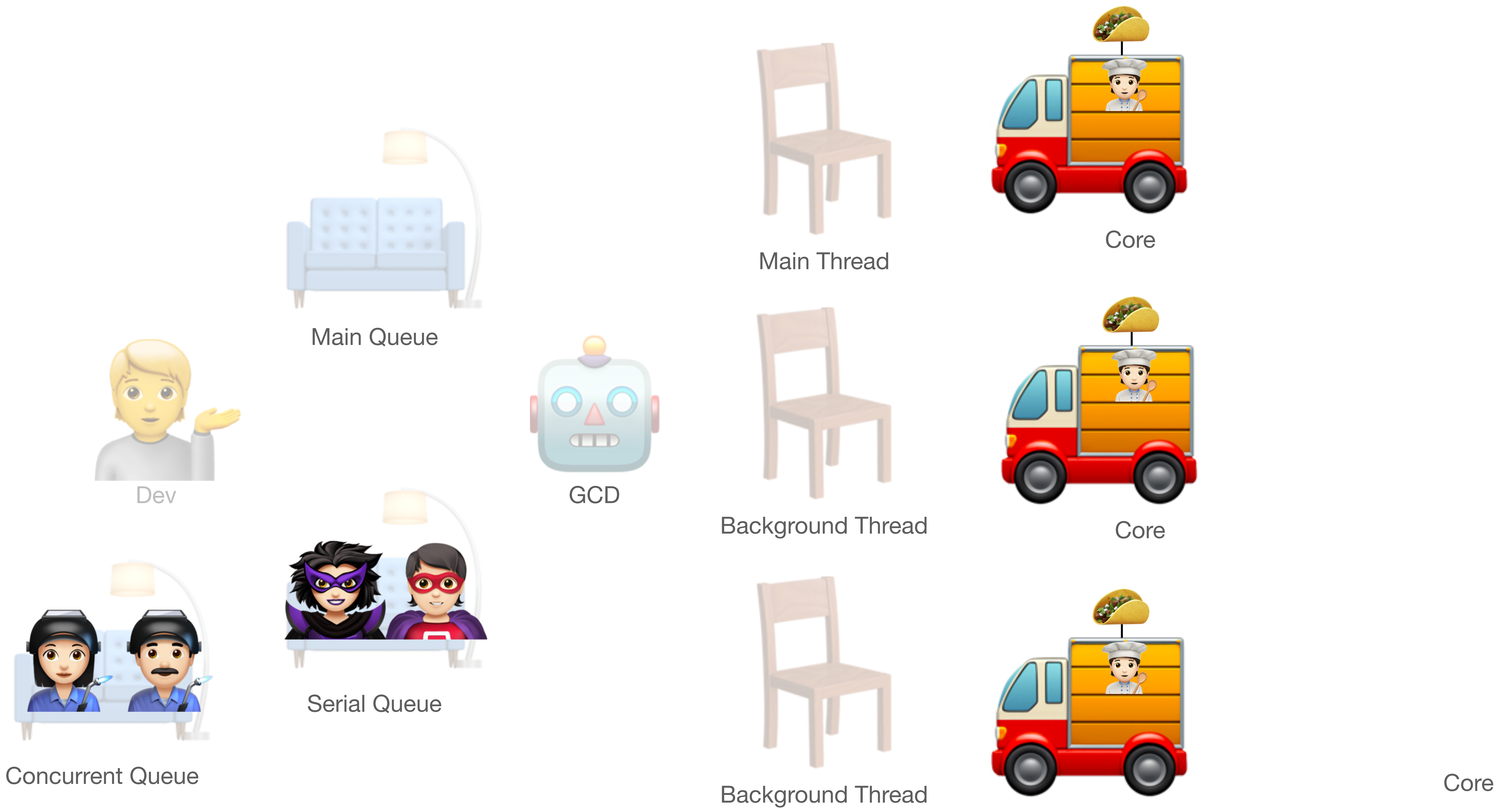


If 🦸 is sync, does 🦹 begin on the other thread?

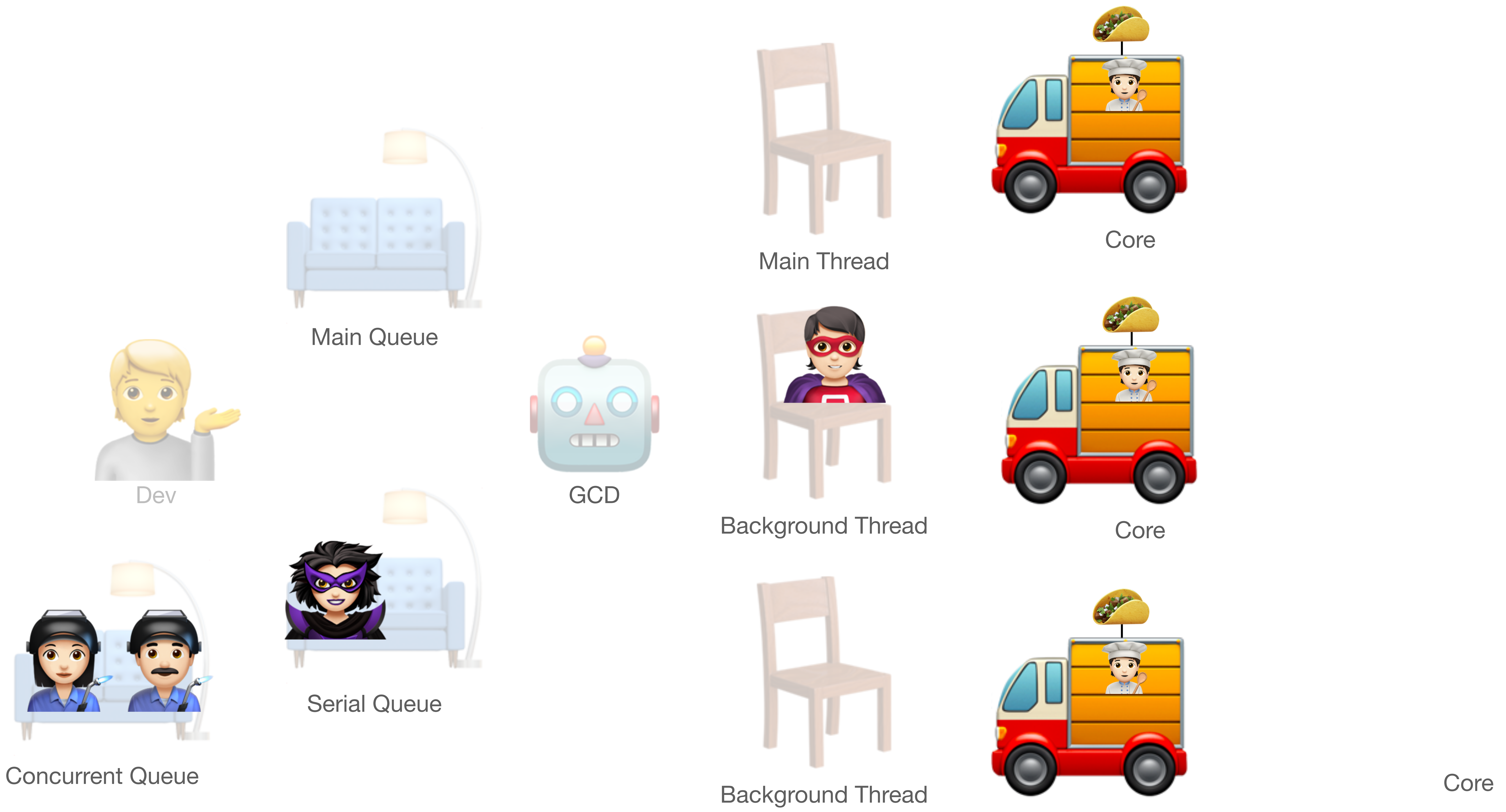
No, never, because 🦸 has requested the queue to stop.



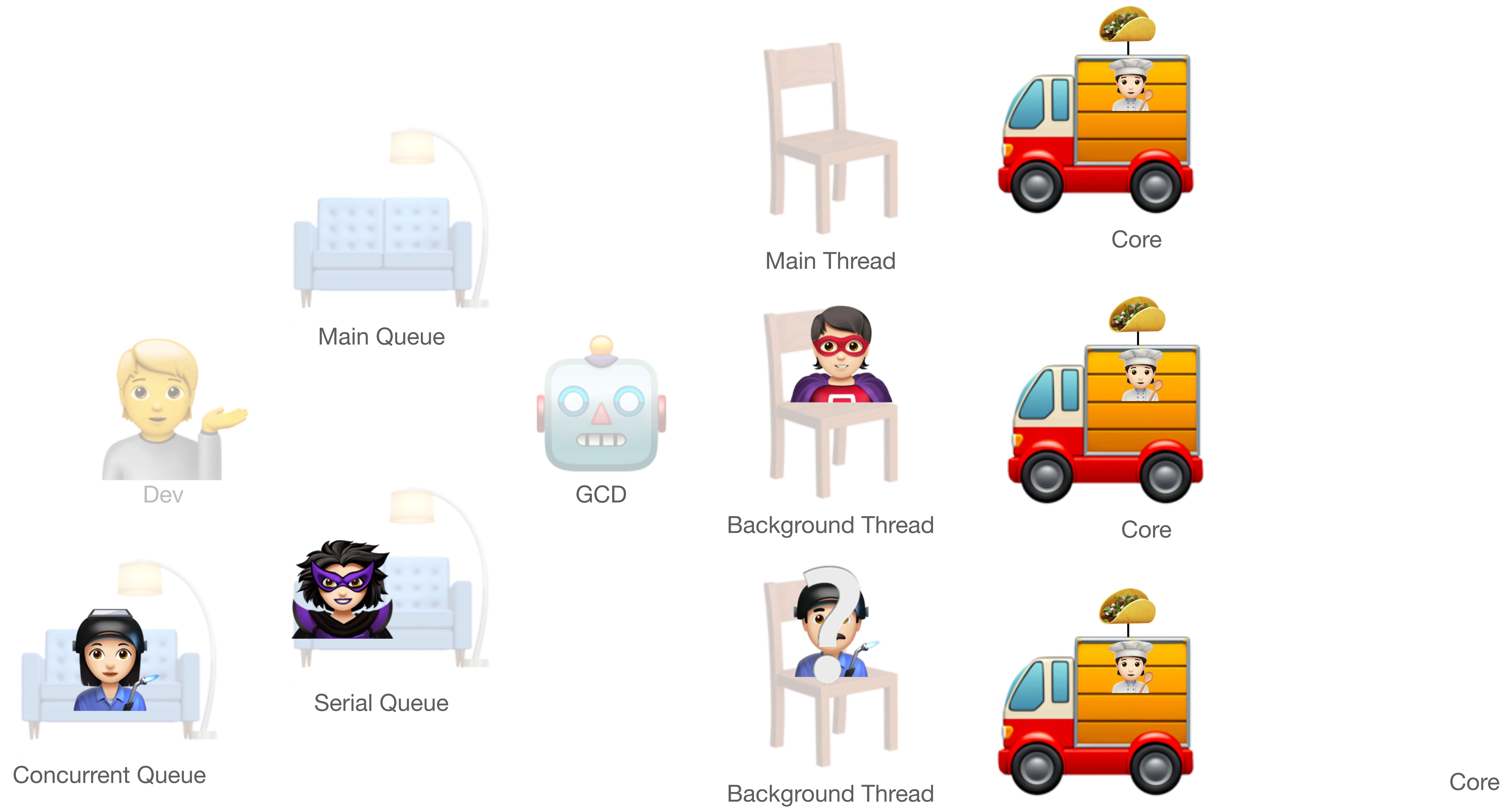
If 🦸 is sync, does 👨‍🏭 begin on the other thread?



If 🦸 is sync, does 👨‍🏭 begin on the other thread?

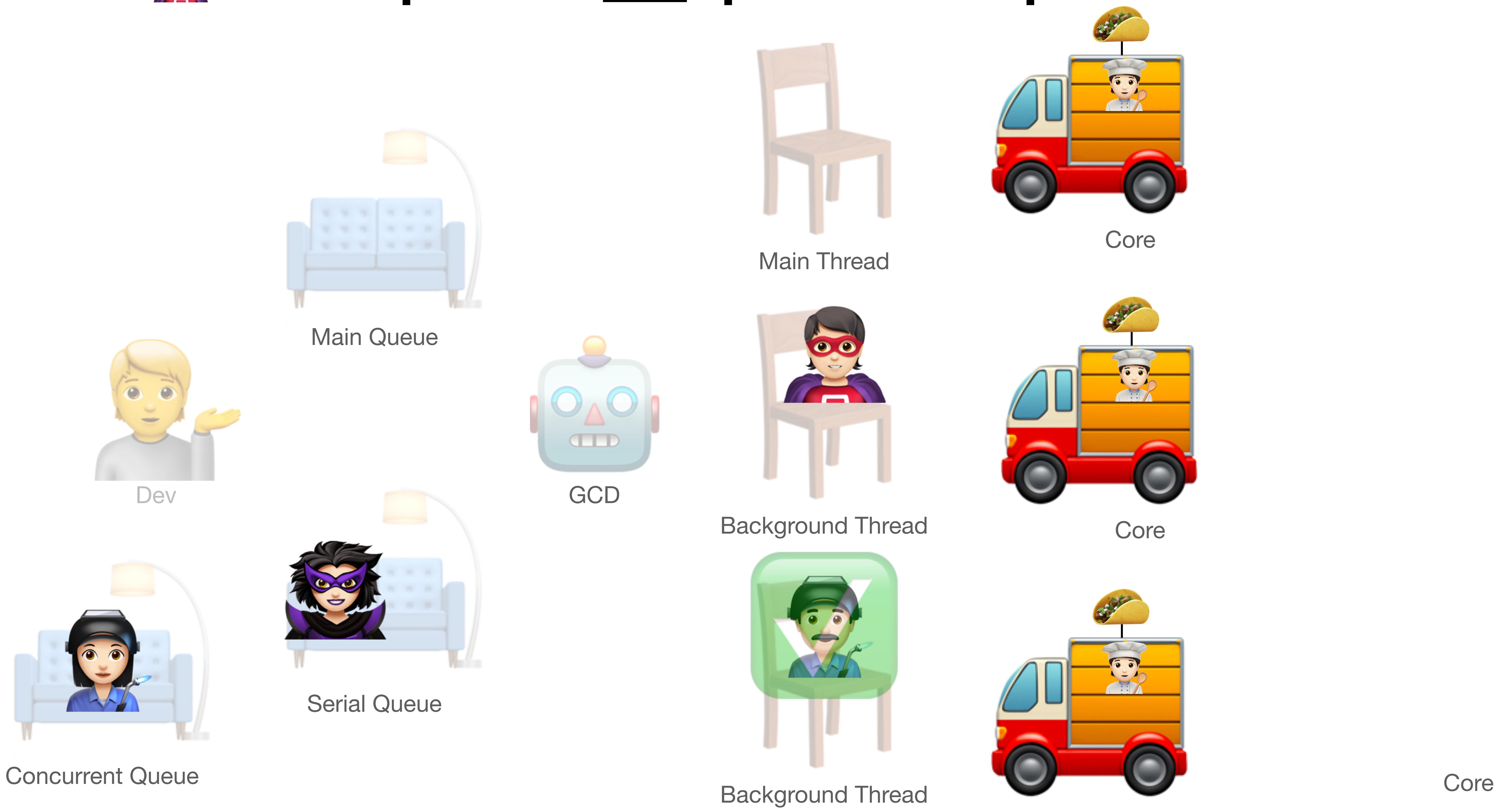


If 🦸 is sync, does 👨‍🏭 begin on the other thread?

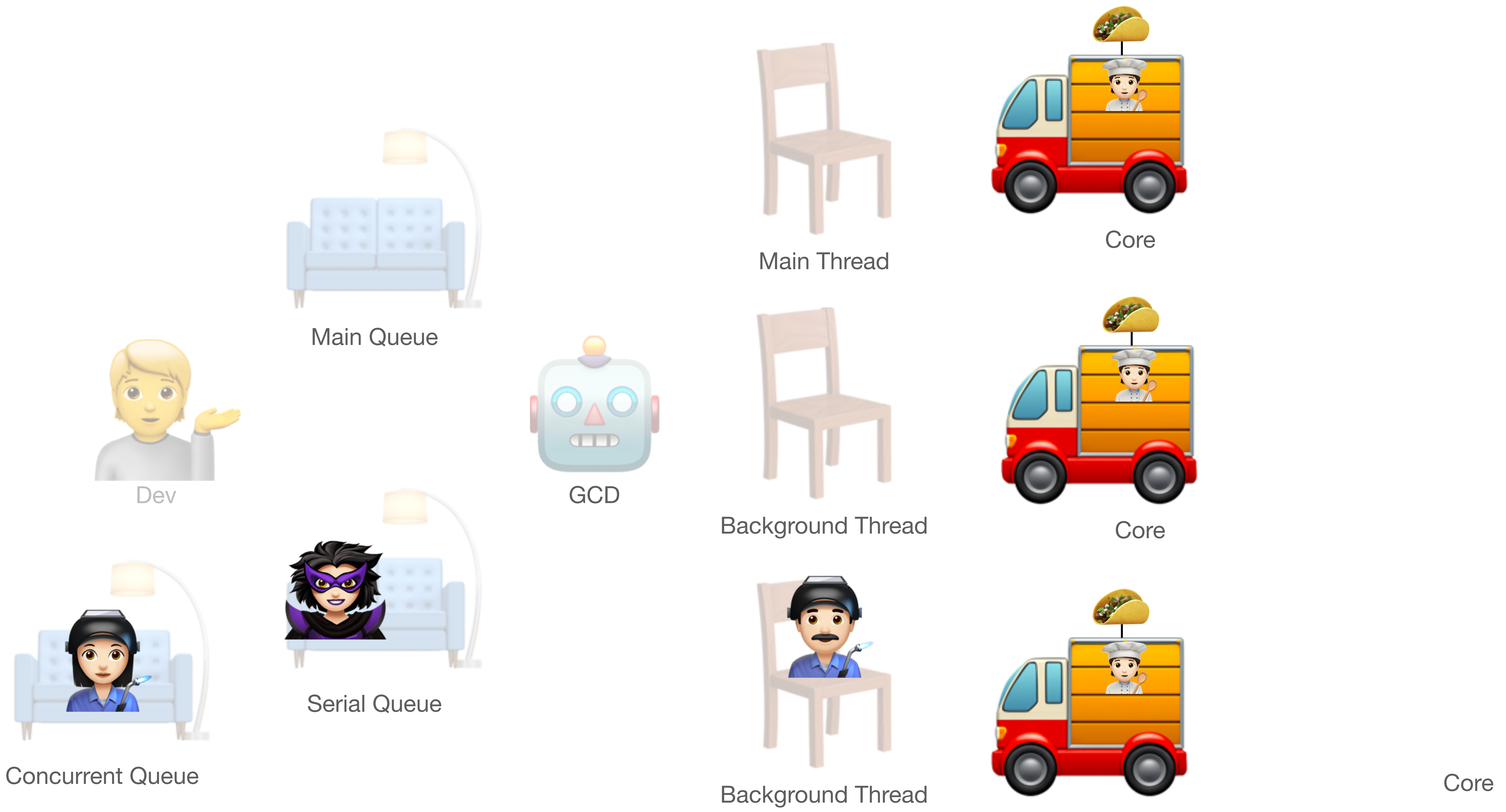


If 🦸 is sync, does 👨‍🏭 begin on the other thread?

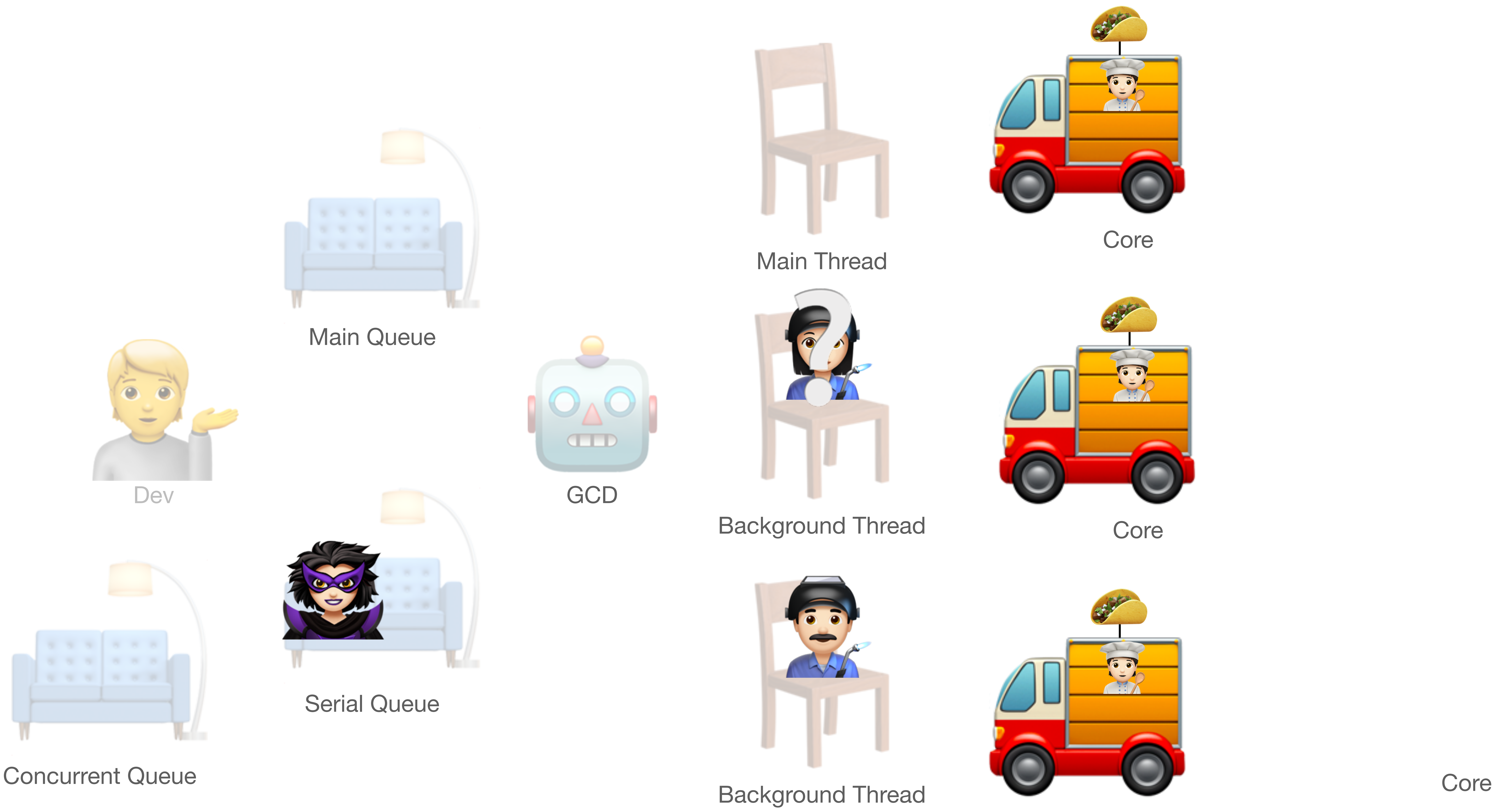
Yes, because 🦸 has requested <u>her</u> queue to stop.



If 🦸 is done, does 👩‍🏭 begin on the other thread?

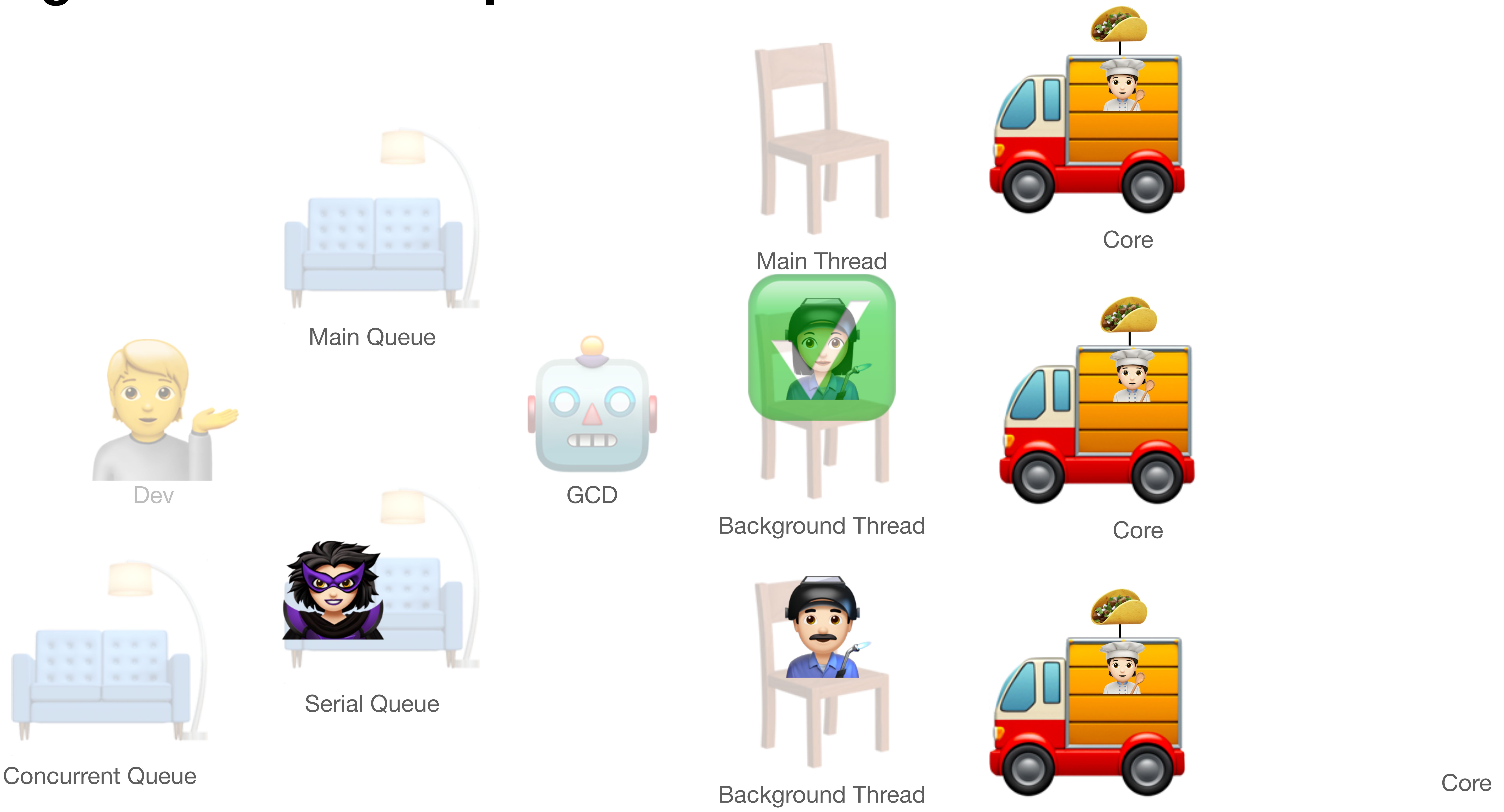


If 🦸 is done, does 👩‍🏭 begin on the other thread?

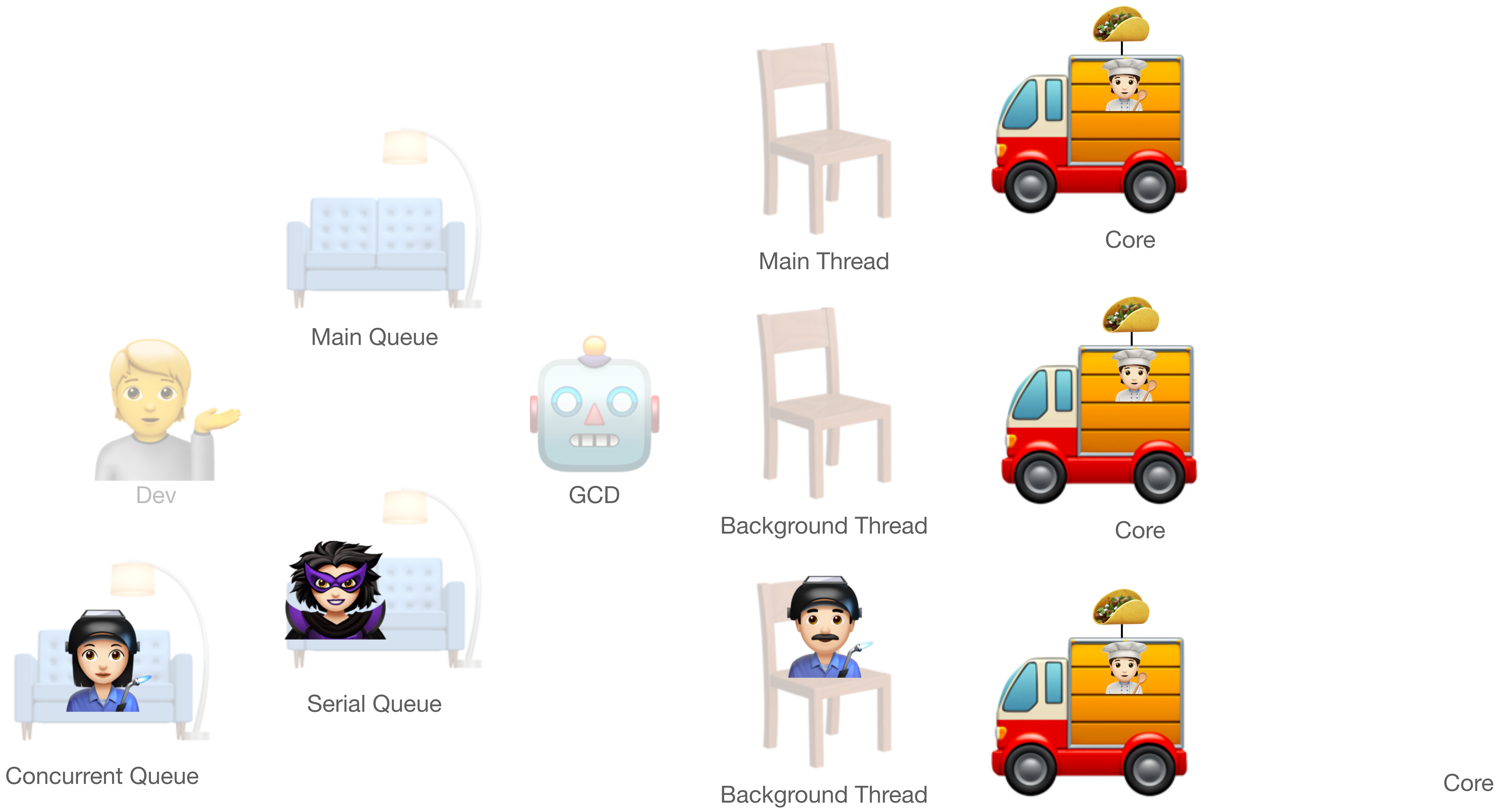


If 🦸 is done, does 👩‍🏭 begin on the other thread?

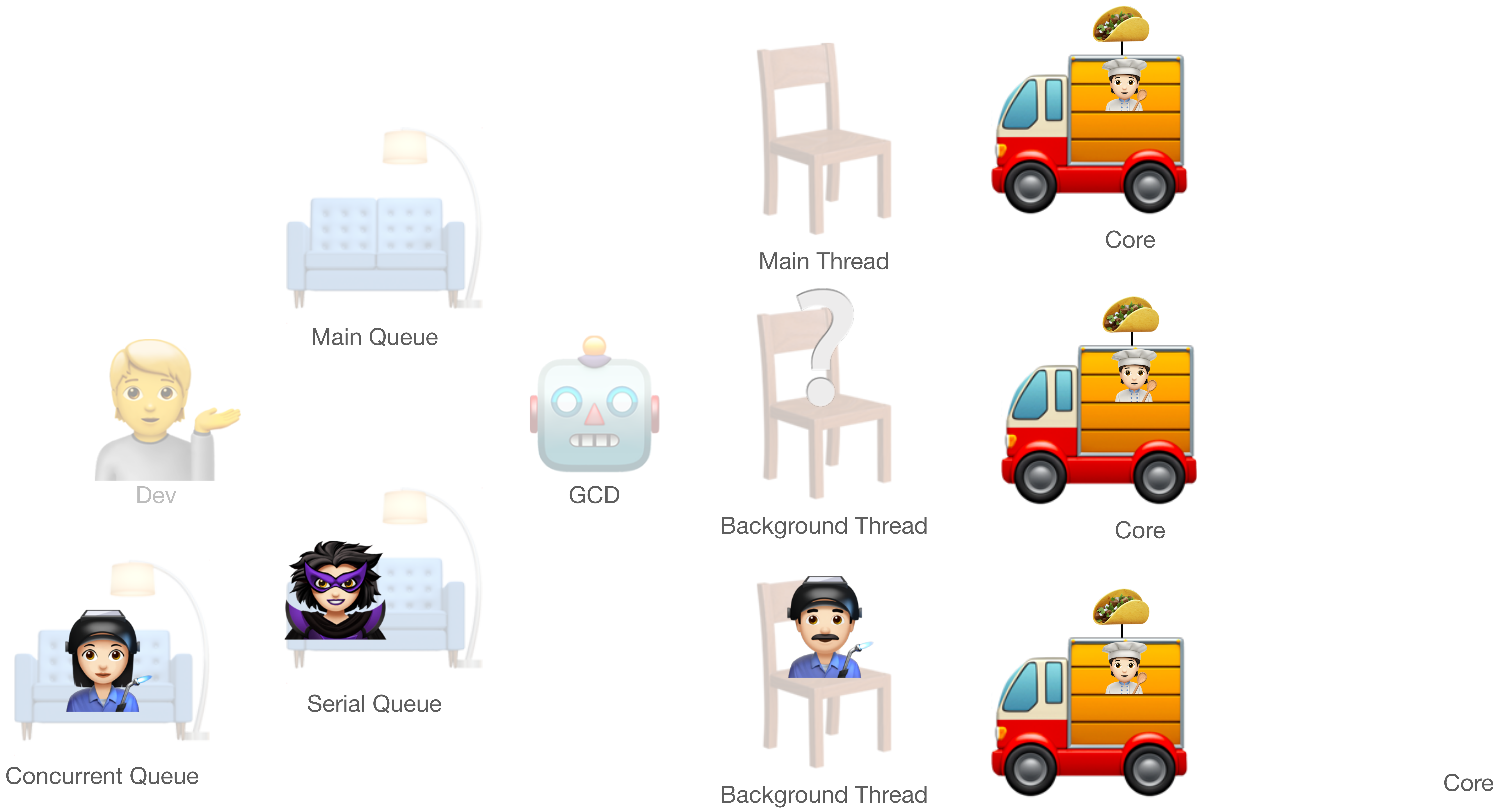
👩‍🏭 can begin because the queue is concurrent.



👨‍🏭 is async. Who should begin on the other thread, 👩‍🏭 or 🦹‍♀️?

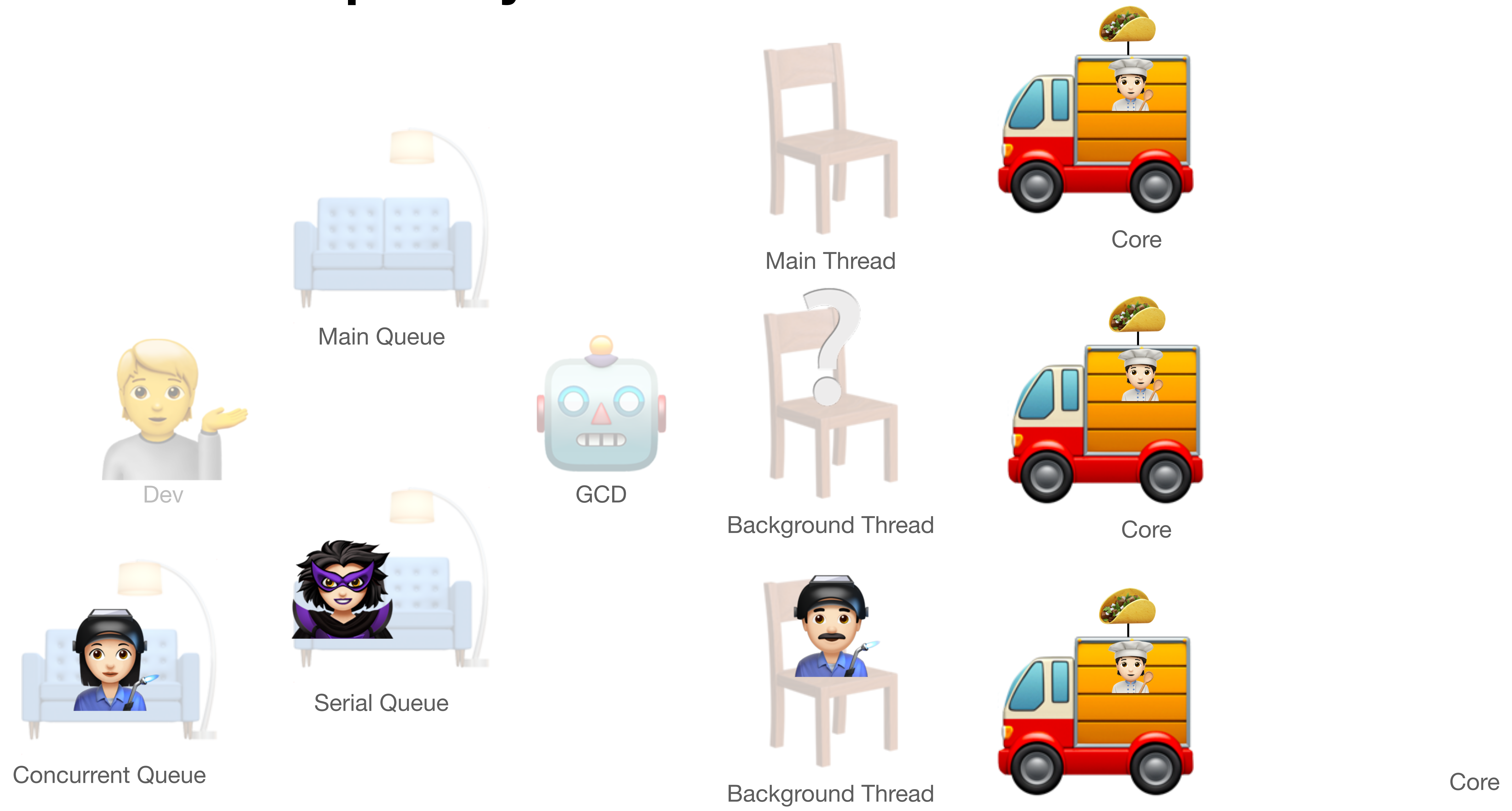


👨‍🏭 is async. Who should begin on the other thread, 👩‍🏭 or 🦹‍♀️?



Who should begin on the other thread, 👩‍🏭 or 🦹‍♀️?

🤖 decides based on priority.



GCD and more

Where to go from here?

- Will be uploaded to https://github.com/olivaresf/intro_to_closures
- You can reach me @fromJrToSr
- Additional info:
  - https://theswiftdev.com/ultimate-grand-central-dispatch-tutorial-in-swift/
  - https://www.raywenderlich.com/5370-grand-central-dispatch-tutorial-for-swift-4-part-1-2

Next Class

Core Data

- Official definition: "is an object graph and persistence framework provided by Apple in the macOS and iOS operating systems."[1]
- Allows a layer of abstraction to exist between the persistence medium (SQLite, Binary, etc) and the app code.
- Core Data has pagination, predicate filtering, context management and abstracts away reading/saving from a database.
- Can be used to persist objects.

[1] - https://developer.apple.com/documentation/coredata

Support Fernando

I need to eat

- Practice Swift weekly with a 15-minute exercise:

  https://mailchi.mp/hey/weekly-swift-exercise-signup

- Donations are welcome! They help keep classes free.

  https://paypal.me/fromjuniortosenior

Q&A