CONCURRENCY AND THREADING

George Porter
May 8, 2025

UC San Diego

ATTRIBUTION

- Content from “Java in a Nutshell”, Ivan Vazquez, and Rick Snodgrass

CONCURRENCY VS PARALLELISM

- Both deal with doing a lot at once, but aren't the same thing
  - Given set of tasks $\{T_1, T_2, \ldots, T_n\}$
- Concurrency:
  - Progress of multiple elements of the set *overlap in time*
- Parallelism:
  - Progress on elements of the set occur *at the same time*

CONCURRENCY

- Might be parallel, might not be parallel

- A single thread of execution can time slice a set of tasks to make partial progress over time
  - Time 0: Work on first 25% of Task 0
  - Time 1: Work on first 25% of Task 1
  - Time 2: Work on first 25% of Task 2
  - Time 3: Work on first 25% of Task 3
  - Time 4: Work on second 25% of Task 0
  - Time 5: Work on second 25% of Task 1
  - …

PARALLELISM

Multiple execution units enable progress to be made simultaneously

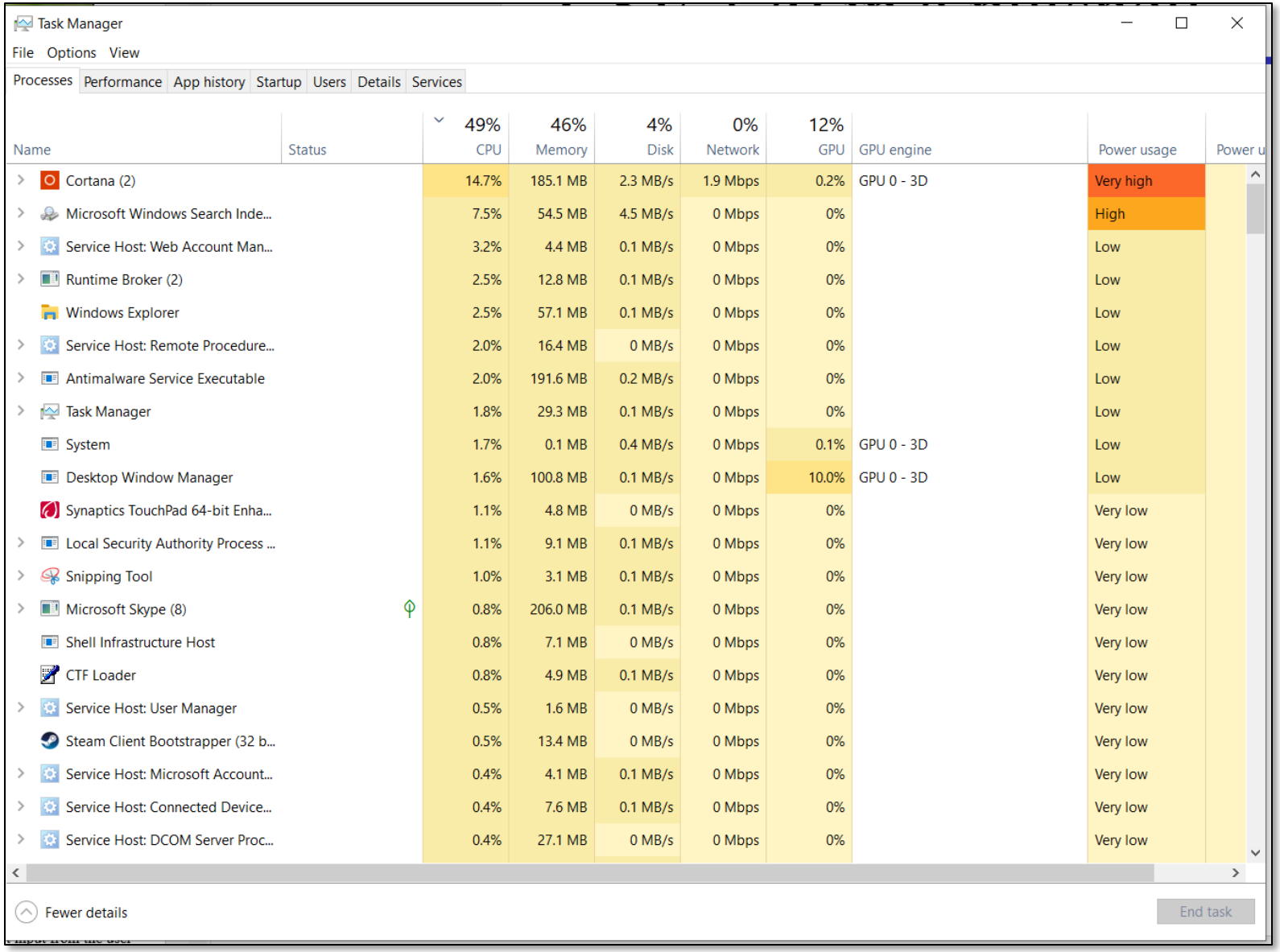
Processor 1

- Time 0: 1st 25% of Task1
- Time 1: 2nd 25% of Task1
- Time 2: 3rd 25% of Task1
- Time 3: 4th 25% of Task1
- Time 4: 1st 25% of Task3

Processor 2

- Time 0: 1st 25% of Task2
- Time 1: 2nd 25% of Task2
- Time 2: 3rd 25% of Task2
- Time 3: 4th 25% of Task2
- Time 4: 1st 25% of Task4

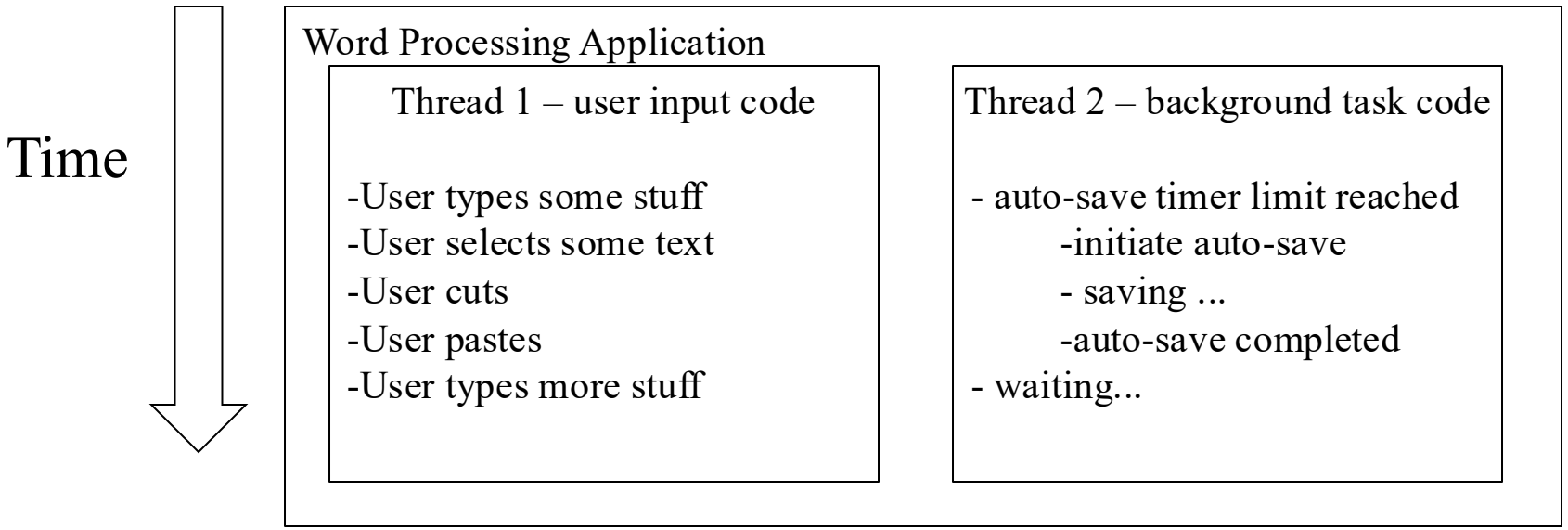
Many processes running at a time



Name	Status	49% CPU	46% Memory	4% Disk	0% Network	12% GPU	GPU engine	Power usage	Power u
Cortana (2)		14.7%	185.1 MB	2.3 MB/s	1.9 Mbps	0.2%	GPU 0 - 3D	Very high	
Microsoft Windows Search Inde...		7.5%	54.5 MB	4.5 MB/s	0 Mbps	0%		High	
Service Host: Web Account Man...		3.2%	4.4 MB	0.1 MB/s	0 Mbps	0%		Low	
Runtime Broker (2)		2.5%	12.8 MB	0.1 MB/s	0 Mbps	0%		Low	
Windows Explorer		2.5%	57.1 MB	0.1 MB/s	0 Mbps	0%		Low	
Service Host: Remote Procedure...		2.0%	16.4 MB	0 MB/s	0 Mbps	0%		Low	
Antimalware Service Executable		2.0%	191.6 MB	0.2 MB/s	0 Mbps	0%		Low	
Task Manager		1.8%	29.3 MB	0.1 MB/s	0 Mbps	0%		Low	
System		1.7%	0.1 MB	0.4 MB/s	0 Mbps	0.1%	GPU 0 - 3D	Low	
Desktop Window Manager		1.6%	100.8 MB	0.1 MB/s	0 Mbps	10.0%	GPU 0 - 3D	Low	
Synaptics TouchPad 64-bit Enha...		1.1%	4.8 MB	0 MB/s	0 Mbps	0%		Very low	
Local Security Authority Process ...		1.1%	9.1 MB	0.1 MB/s	0 Mbps	0%		Very low	
Snipping Tool		1.0%	3.1 MB	0.1 MB/s	0 Mbps	0%		Very low	
Microsoft Skype (8)		0.8%	206.0 MB	0.1 MB/s	0 Mbps	0%		Very low	
Shell Infrastructure Host		0.8%	7.1 MB	0 MB/s	0 Mbps	0%		Very low	
CTF Loader		0.8%	4.9 MB	0.1 MB/s	0 Mbps	0%		Very low	
Service Host: User Manager		0.5%	1.6 MB	0 MB/s	0 Mbps	0%		Very low	
Steam Client Bootstrapper (32 b...		0.5%	13.4 MB	0 MB/s	0 Mbps	0%		Very low	
Service Host: Microsoft Account...		0.4%	4.1 MB	0.1 MB/s	0 Mbps	0%		Very low	
Service Host: Connected Device...		0.4%	7.6 MB	0.1 MB/s	0 Mbps	0%		Very low	
Service Host: DCOM Server Proc...		0.4%	27.1 MB	0 MB/s	0 Mbps	0%		Very low	

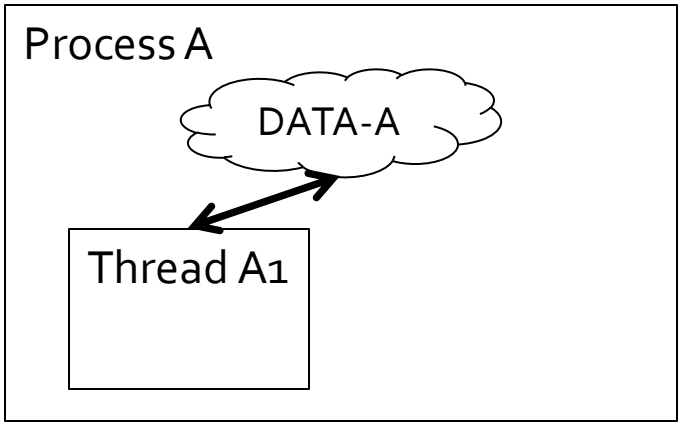
What Are Threads?

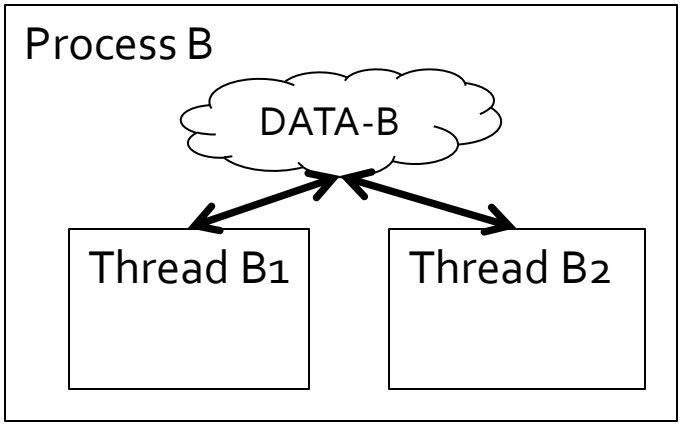
- As an example program using threads, a word processor should be able to accept input from the user and at the same time, auto-save the document.
- The word processing application contains two threads:
  - One to handle user-input
  - Another to process background tasks (like auto-saving).



Programming Perspective

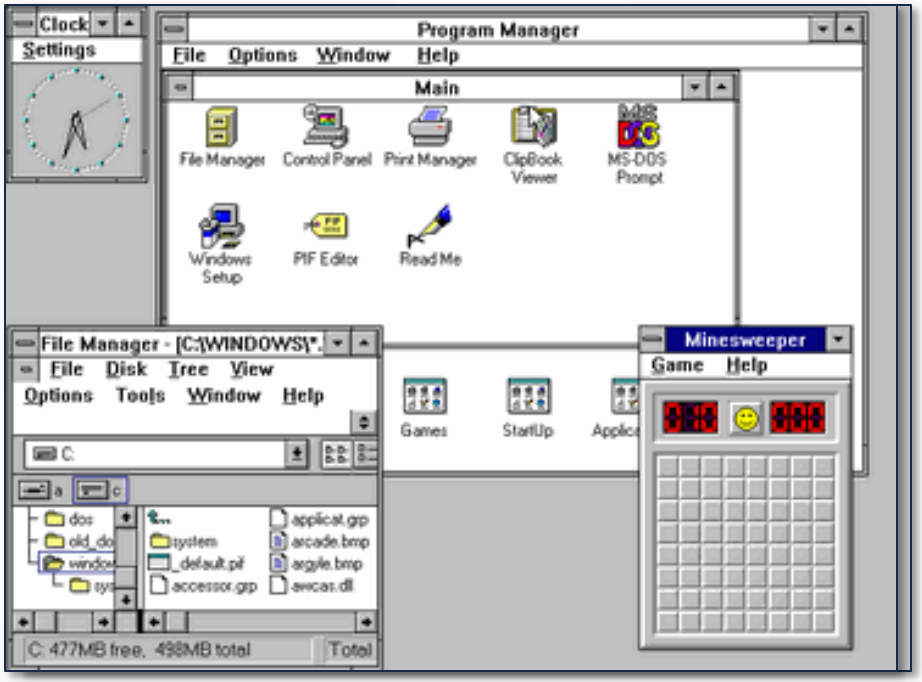
- Every process on your server/machine has:
  - A virtual memory address space
    - Your program's heap, code, global variables
  - One or more "threads of control" (or just "threads")
    - Each one consists of:
      - Its own local program counter
      - Its own local stack
- Threads run on a CPU (or CPU "core")
- The OS schedules threads (puts them on the CPU)
  - And deschedules them (takes them off the CPU)
- Your "main" function runs in a thread
  - You've already been programming using threads!

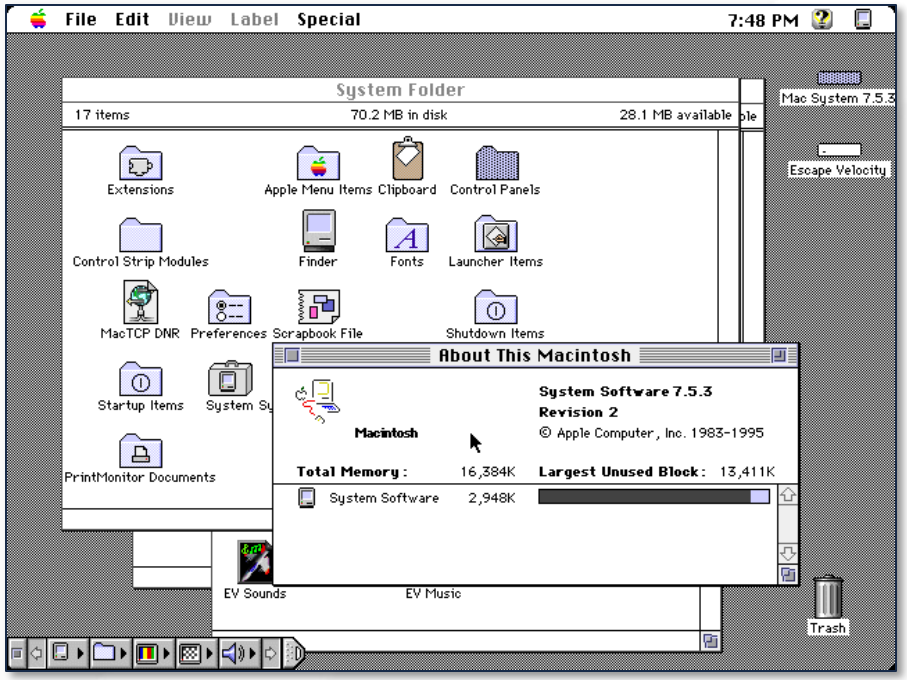




CONCURRENCY VS PARALLELISM

- Concurrency is an abstraction implemented via the OS and/or programming runtime
  - The hardware determines if that concurrency is parallel or not
  - Earlier OSes were single CPU, single core, but had multitasking!





GO AND GOROUTINES

- Go supports concurrency via *goroutines*
- Put the keyword **go** in front of a function call
  - The runtime will run it in its own goroutine
    - Concurrently with `main` and any other goroutines
  - Goroutines can't return anything
    - Well they can but the caller throws away that result
    - Other methods are needed to retrieve results, as we'll see in a minute

WHAT ABOUT NETWORKED SERVICES?

- Recall the ‘goroutine per client’ model from week 2’s TA discussion section:

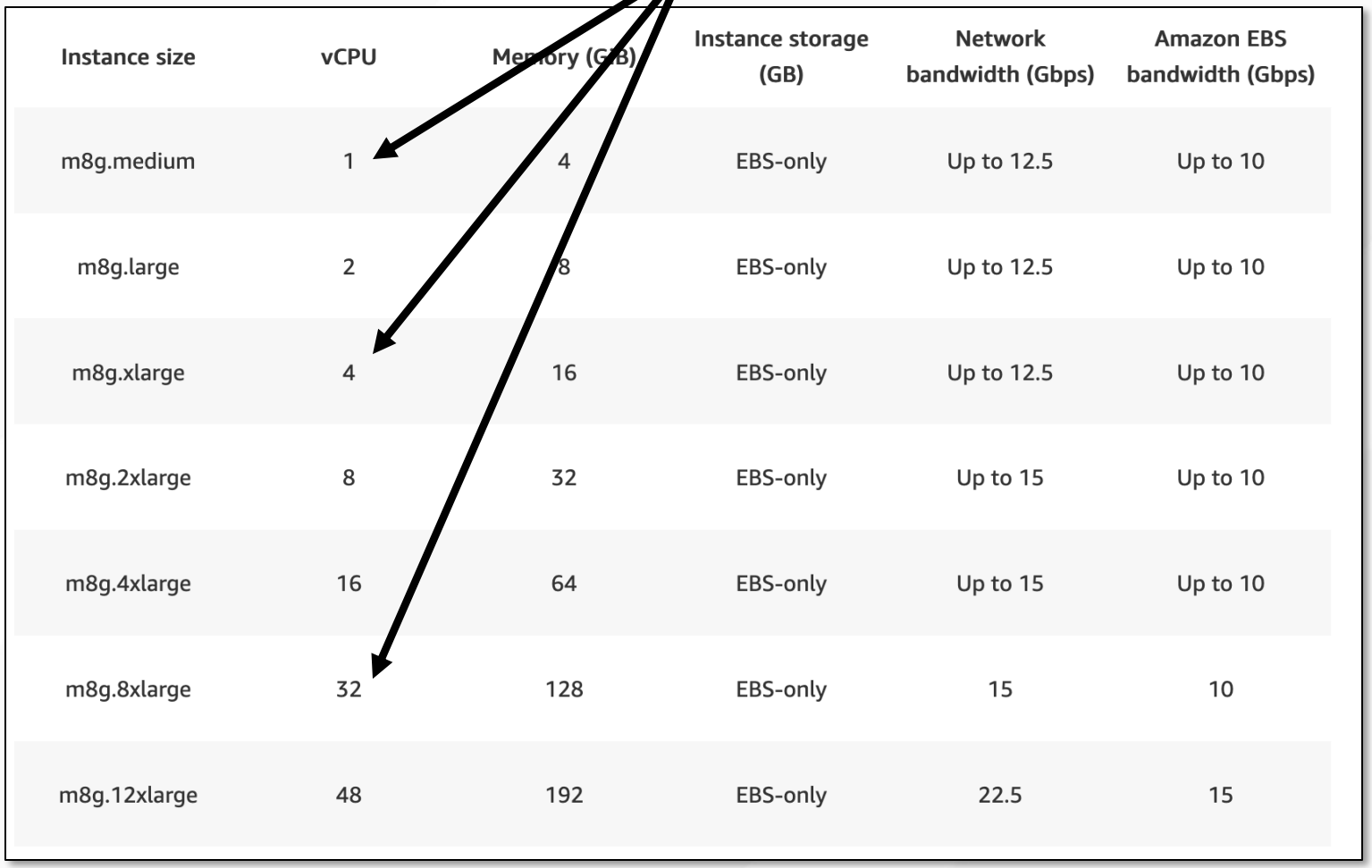
```go
for {
    // Accept incoming client connections
    conn, err := listener.Accept()
    if err != nil {
        fmt.Println( a...: "Error accepting connection:", err)
        continue
    }
    fmt.Printf( format: "Client connected from: %s\n", conn.RemoteAddr().String())

    // Handle each client in a separate goroutine
    go handleClient(conn)
}
```

THREAD/GOROUTINE PER CLIENT

- Each per-client **`net.Conn`** is passed to a goroutine which handles that connection
- Simple model suitable for smaller projects and lower level of *scale*
- But what happens when:
  - each goroutine runs a long time?
  - many many clients arrive close in time to each other?

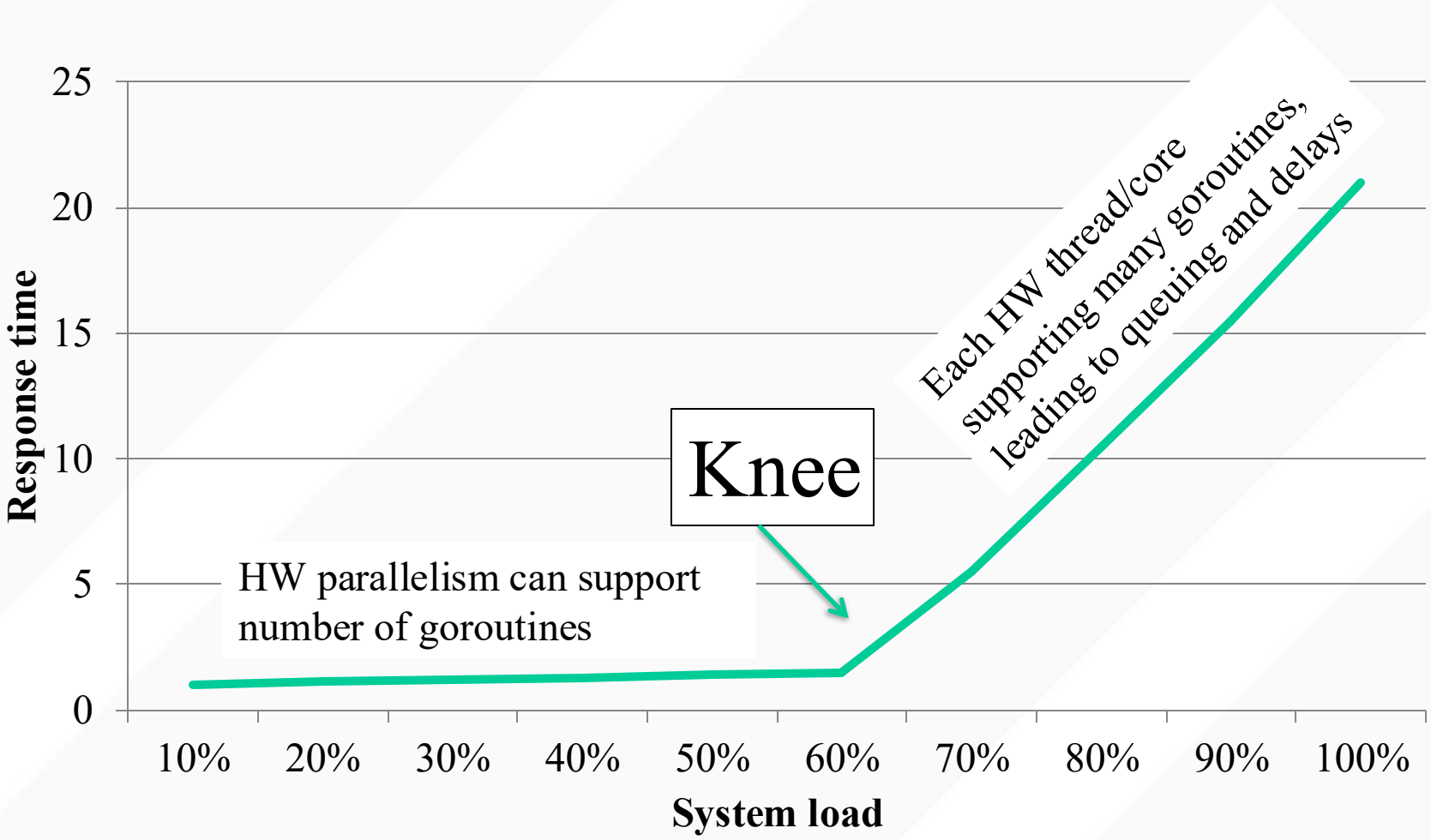
WHEN HARDWARE PARALLELISM MATTERS

- No direct connection between the number of goroutines supported by the runtime and the hardware capability to do parallelism



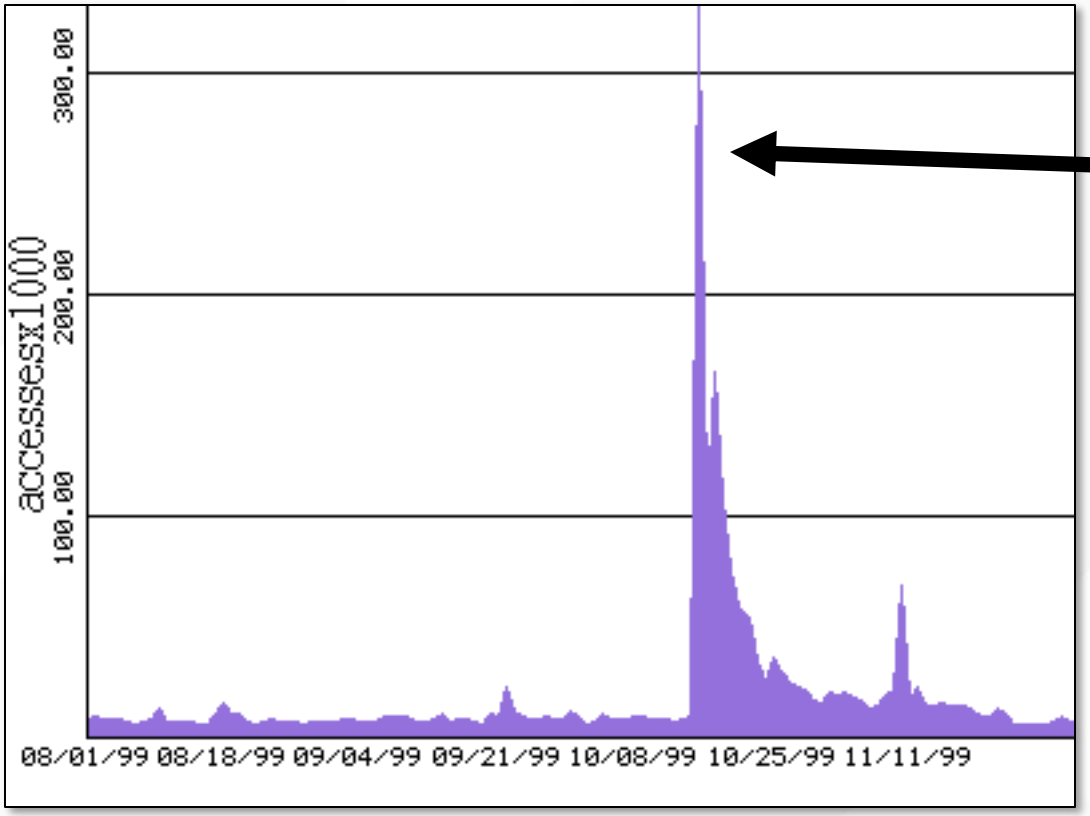
Instance size	vCPU	Memory (GiB)	Instance storage (GB)	Network bandwidth (Gbps)	Amazon EBS bandwidth (Gbps)
m8g.medium	1	4	EBS-only	Up to 12.5	Up to 10
m8g.large	2	8	EBS-only	Up to 12.5	Up to 10
m8g.xlarge	4	16	EBS-only	Up to 12.5	Up to 10
m8g.2xlarge	8	32	EBS-only	Up to 15	Up to 10
m8g.4xlarge	16	64	EBS-only	Up to 15	Up to 10
m8g.8xlarge	32	128	EBS-only	15	10
m8g.12xlarge	48	192	EBS-only	22.5	15

PERFORMANCE "HOCKEY STICK" GRAPH



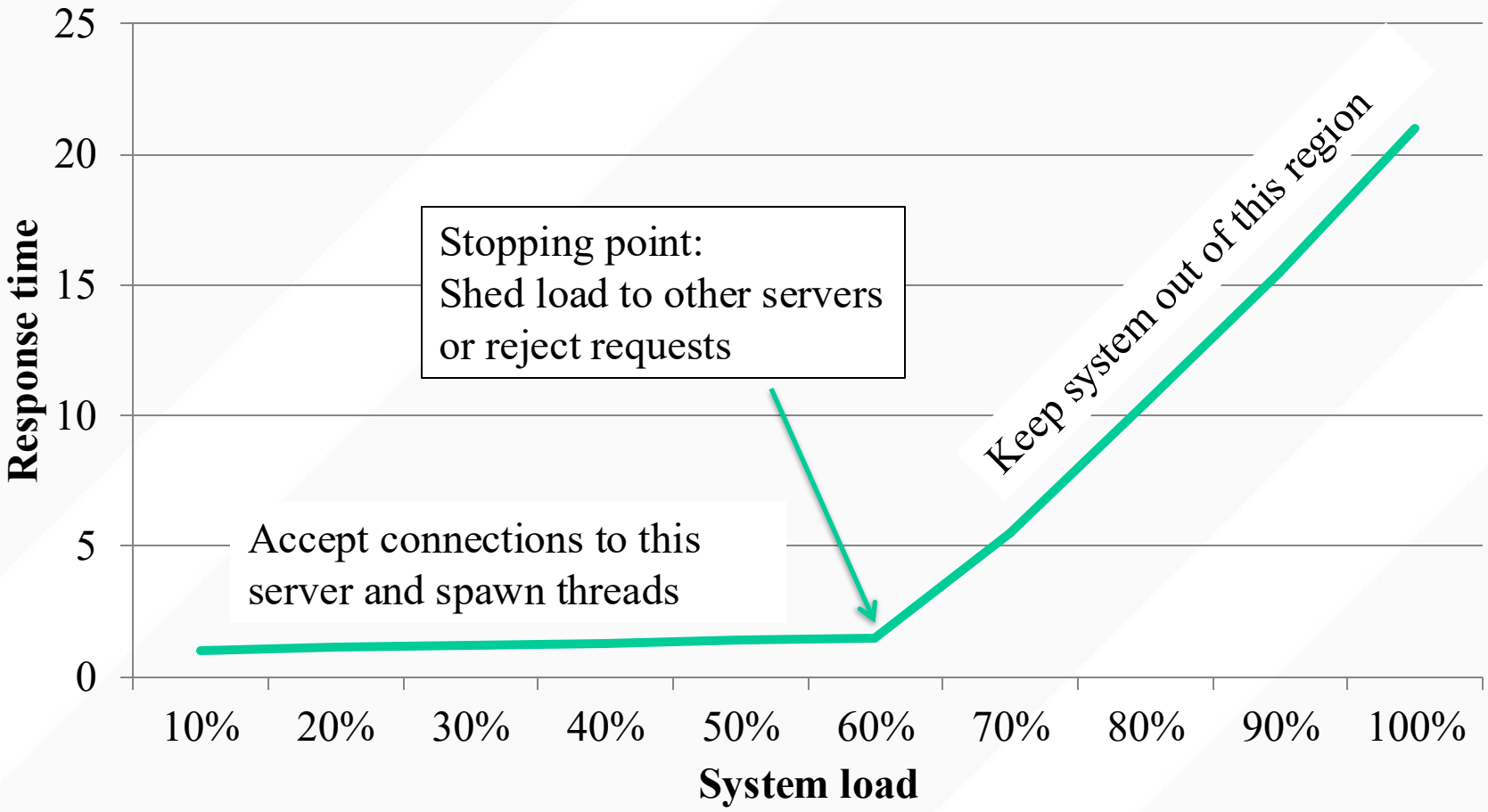
FLASH TRAFFIC

- USGS Pasadena, CA office Earthquake site
- Oct 16, 1999 earthquake



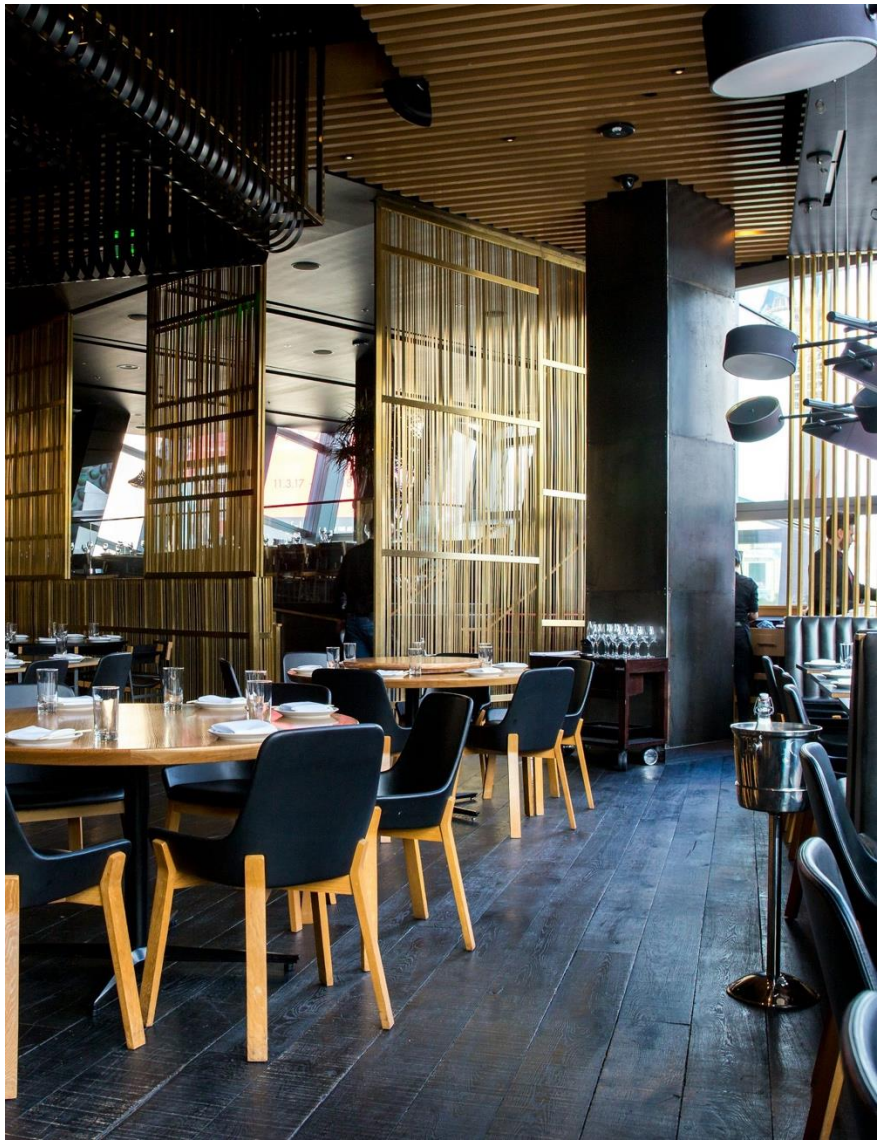
Not enough HW parallelism to support this level of software concurrency

SO HOW TO MANAGE HIGH SCALE?

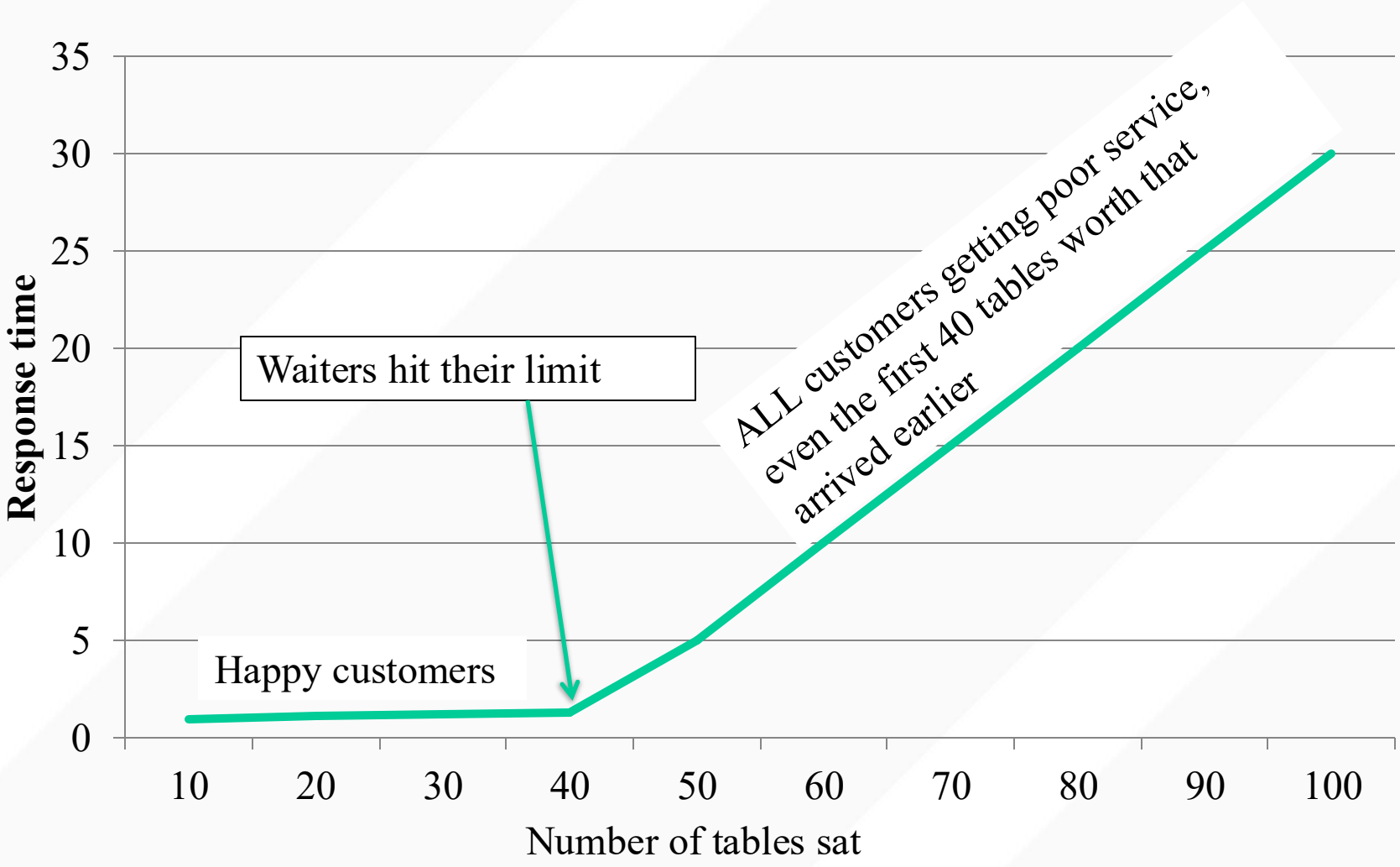


Example situation

- Imagine a restaurant with 100 tables
- Each waiter can handle 4 tables before getting overwhelmed
- Because the flu is going around, only 10 waiters came to work this morning

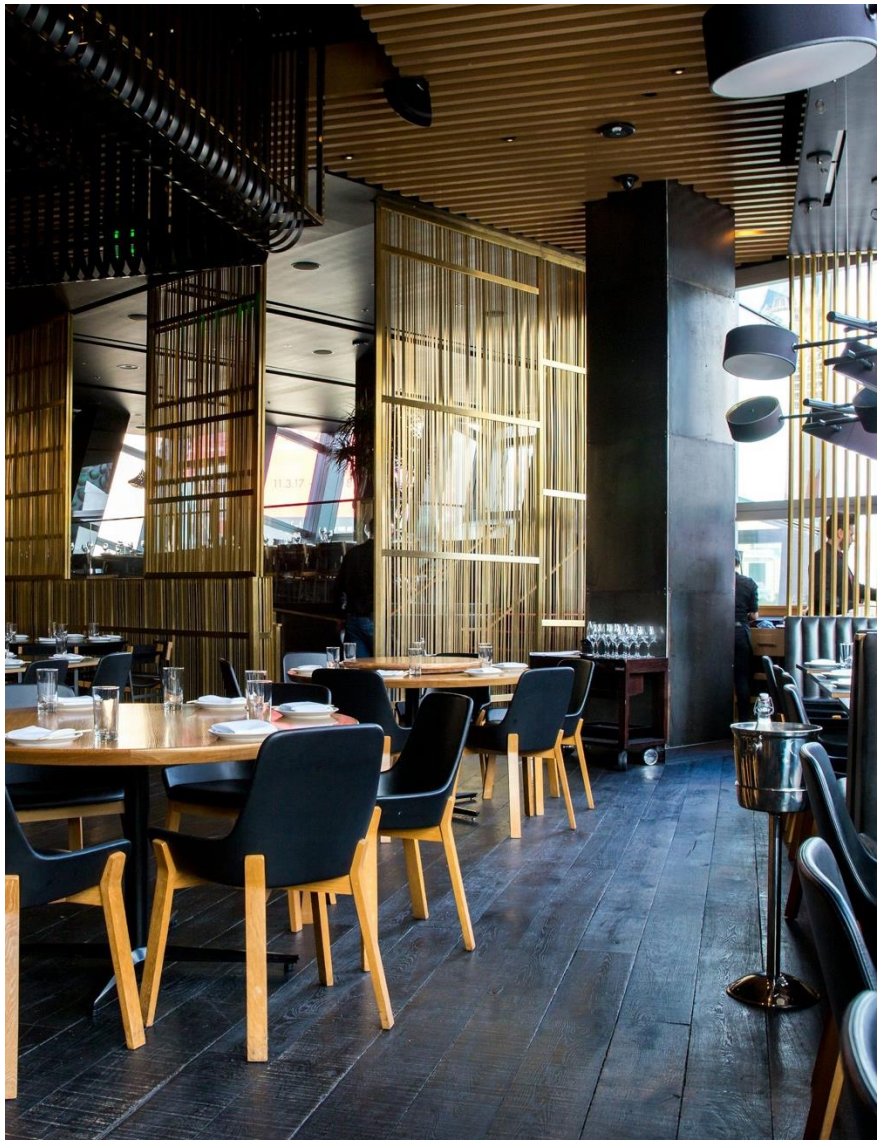


100 SEAT RESTAURANT WITH 10 WAITERS (4 TABLES PER WAITER)



Thread pool idea

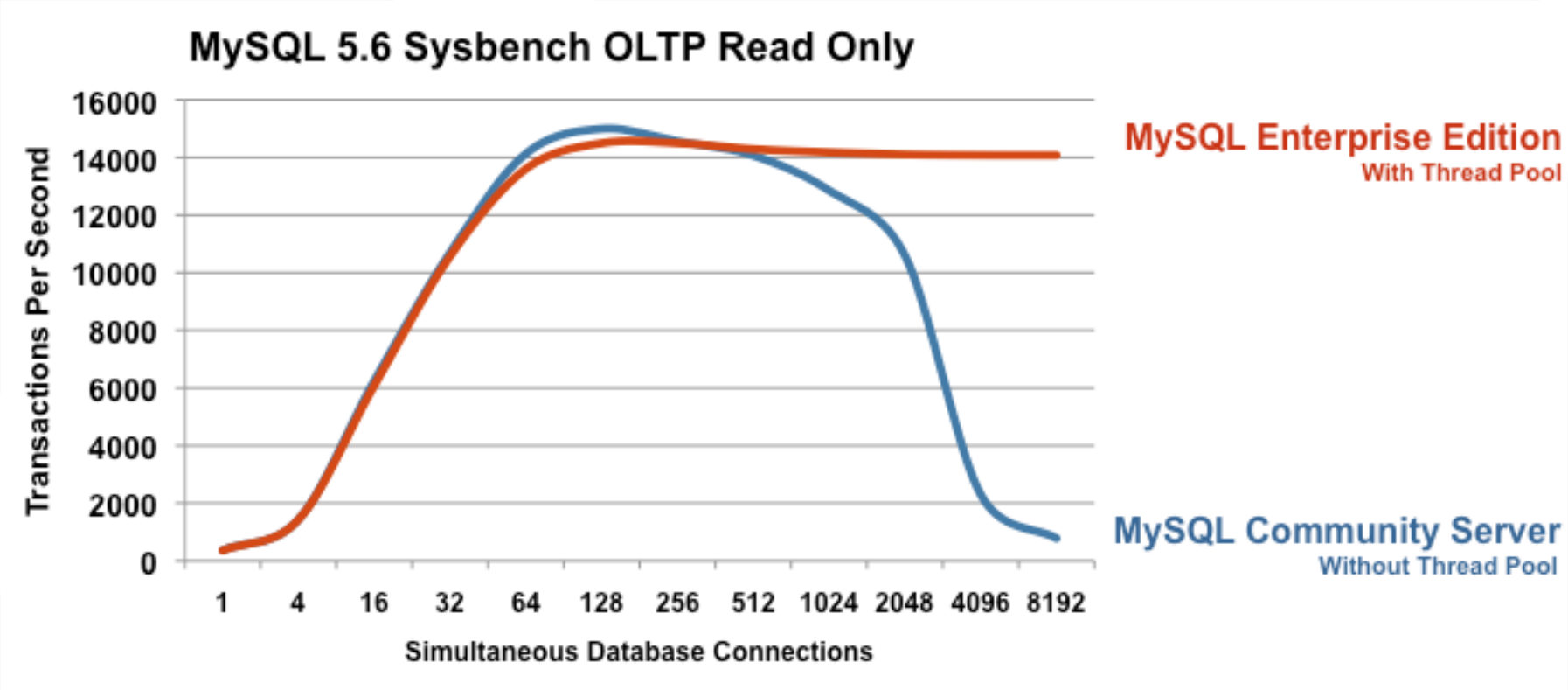
- Only sit 40 tables (the capacity of the waitstaff)
- When guests arrive past that point, have them wait outside or tell them the restaurant is full tonight
- Result: not all arriving customers get seated!
- BUT, those that do have sufficient waitstaff time/attention for a good experience



THREAD POOL EXAMPLE

THREAD POOLS IN ACTION

- Too much parallelism causes thrashing, excessive switching, lower performance



IN-CLASS DEMO

- Let’s see an example of goroutines and channels

UC San Diego