Understanding Threading and Concurrency

David Flynn
SOFTWARE ENGINEER, FLYNN IT LTD

Introduction

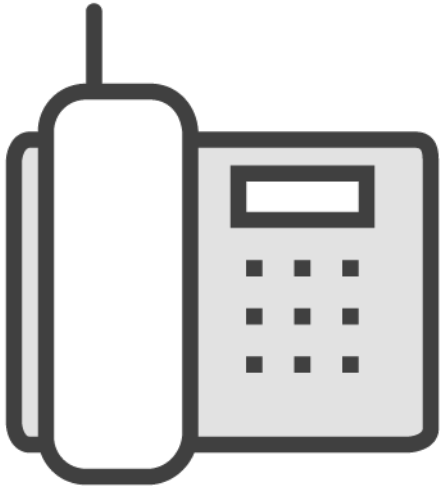
Section One

Concurrency, Multitasking and Multithreading

Section Two

Basic cache theory

Scenario ‘Cool Tech’ Interview



Technical Interview

Simple questions to start

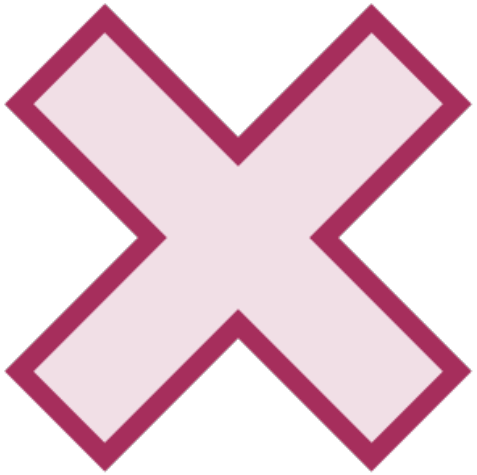
- Definitions

Common in phone screening

The ‘Cool Tech’ Interview

What do you understand
by concurrency?

A Not so Good Answer



“Concurrency is running several things at the same time.”

A bit vague

Would probably invite follow up questions

Multitasking Comparison

Multitasking

Running several processes on the same machine

Each could make progress without any user help

Non-multitasking environment

Only one process can be run

Only the active process can make progress

Examples of Processes

Programs and applications

System tools

Operating system processes

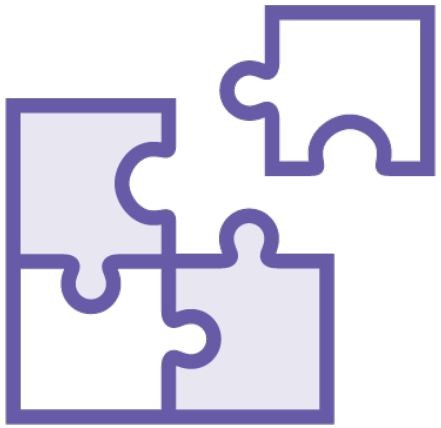
Process Characteristics

Usually standalone

Often no knowledge of each other

Own memory allocation (and usually segmentation)

Segmentation



Processes can't access or modify other's data

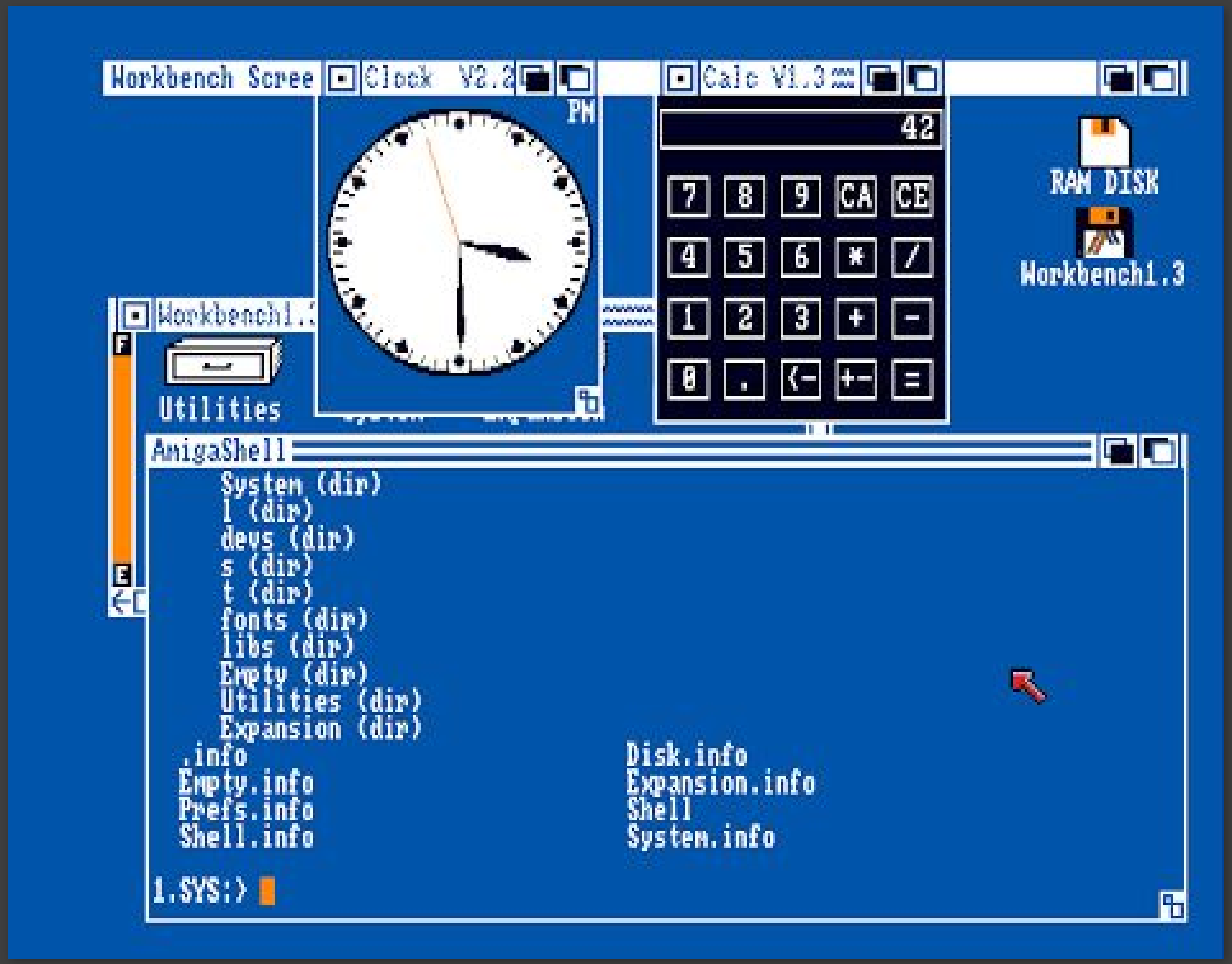
Any communication takes place via:

- File system
- Sockets
- Specifically designated shared memory

Tasks Are Heavyweight

Processes require a fair amount of 'accounting' data

Each has own memory allocation


Workbench Scree
Clock V2.2
PM
Calc V1.3
42
7 8 9 CA CE
4 5 6 * /
1 2 3 + -
0 . <- +- =
RAM DISK
Workbench1.3
Utilities
AmigaShell
System (dir)
l (dir)
devs (dir)
s (dir)
t (dir)
fonts (dir)
libs (dir)
Empty (dir)
Utilities (dir)
Expansion (dir)
.info
Empty.info
Prefs.info
Shell.info
Disk.info
Expansion.info
Shell
System.info
1.SYS:>

Time Slicing

Allocated a time slice on CPU

Interrupted when time is up

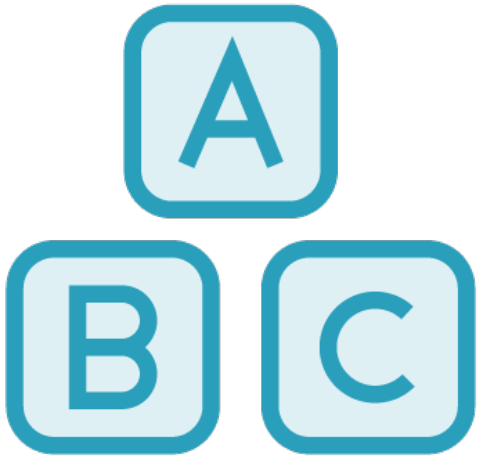
Task's state saved

Another's is restored and continued

Multithreading

Running several threads of computation at the same time in the same process

Threads



Share memory and resources

Likely to have lower overheads than processes

Part of the same program

Cannot exist outside of the process' lifetime

Threads are often seen as lightweight processes

Will Threads Run in Parallel?

No guarantee

The scheduler uses time slicing

JVM manages threads in the scheduler

Threads Are Lightweight, but Not Free

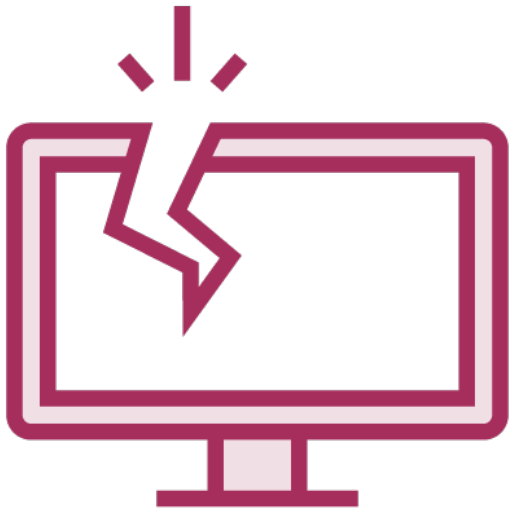
Need their own stack

Have some memory overhead

Creating and destroying takes time

Scheduler swapping one to the next takes time

Too Many Threads



Run the machine out of memory

Prevent other threads getting enough time on the CPU

This is starvation

Concurrency

When a task is broken down into smaller pieces and run simultaneously

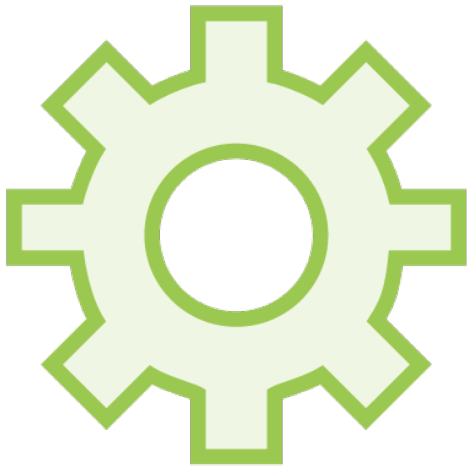
Parallelism

Multithreading and multitasking do not require it

So neither does concurrency

It would be useful to run concurrent parts in parallel

Even Without Parallelism...



Some parts might block

During which other parts can make progress

Don’t have to worry about how to make sure all parts make progress

Other Uses of Concurrency: GUIs

Keep GUI responsive

Special GUI thread for drawing and reporting user events

Don't want GUI to lock up while tasks carried out

Other Uses of Concurrency: Actors

Threads take on roles

Work simultaneously

Interact

Optimum Number of Threads

Keep all available cores busy

- Maximise throughput
- Minimise time taken to complete

Need to determine optimum thread number

- Make it tweakable

Determining Optimum Number

Estimate

Use models (Amdahl's law)

Experiment on [copy of] production system

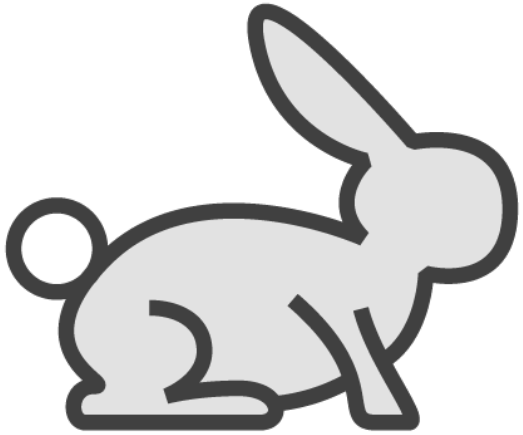
What do you understand
by concurrency?

Task is broken down into smaller pieces

Run simultaneously

- Tasks (processes) in multitasking
- Threads in multithreading
- Distribution of work across a grid or the cloud

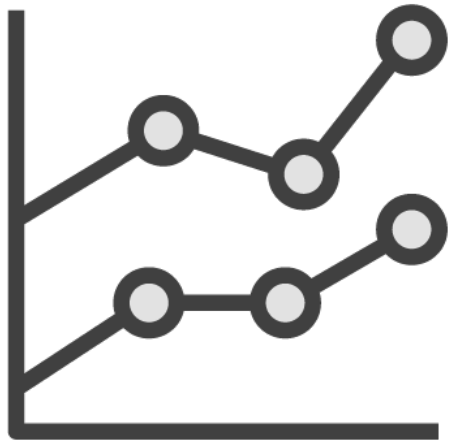
And what benefits does that bring?



Allows problem to be solved far quicker

Allows problems to be solved that would take too long otherwise

How does that differ
from multithreading?



Multithreading: Running several threads of computation at the same time in the same process

- Threads share memory and resources plus they can interact

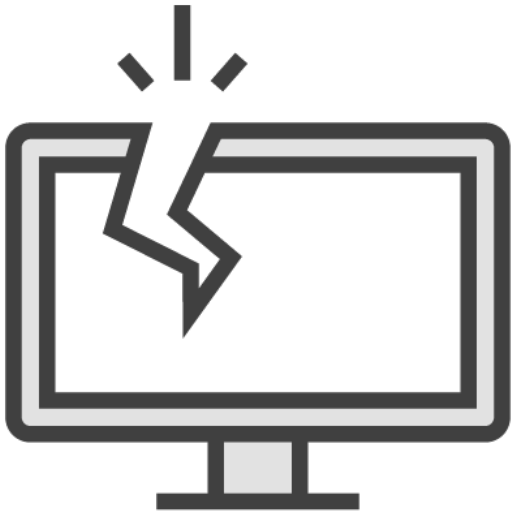
One way of executing concurrent parts of the program

Does multithreading require the ability to run in parallel?

No:

- Time slicing allows threads to run simultaneously
- No guarantee that threads will ever run in parallel

Can you have almost
unlimited numbers
of threads?



Yes, but...

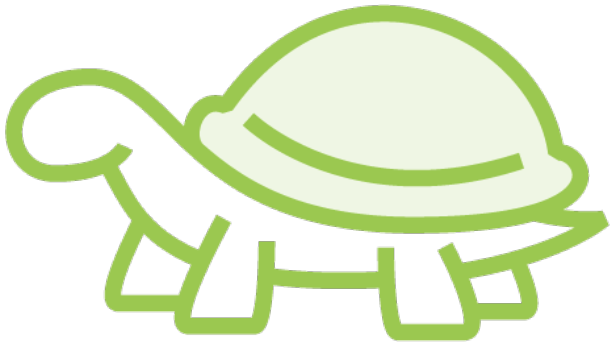
Each requires
- Own stack
- Memory to manage

Smaller number of cores
- Time slicing

Too many threads will either run the machine out of memory or cause starvation

Suggest issues that might occur in a multithreaded system with respect to caching?

Memory Accesses Are Slow

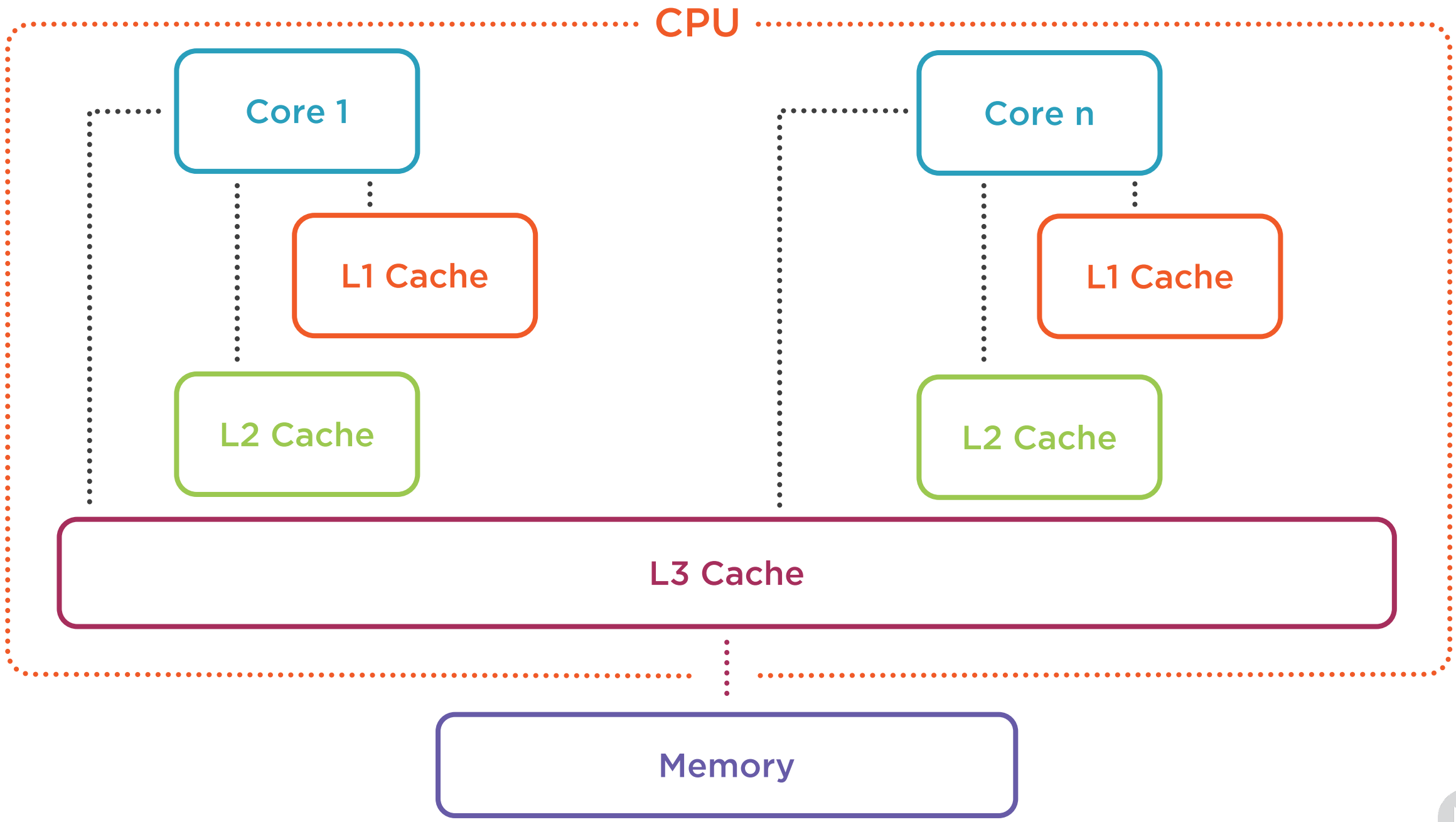


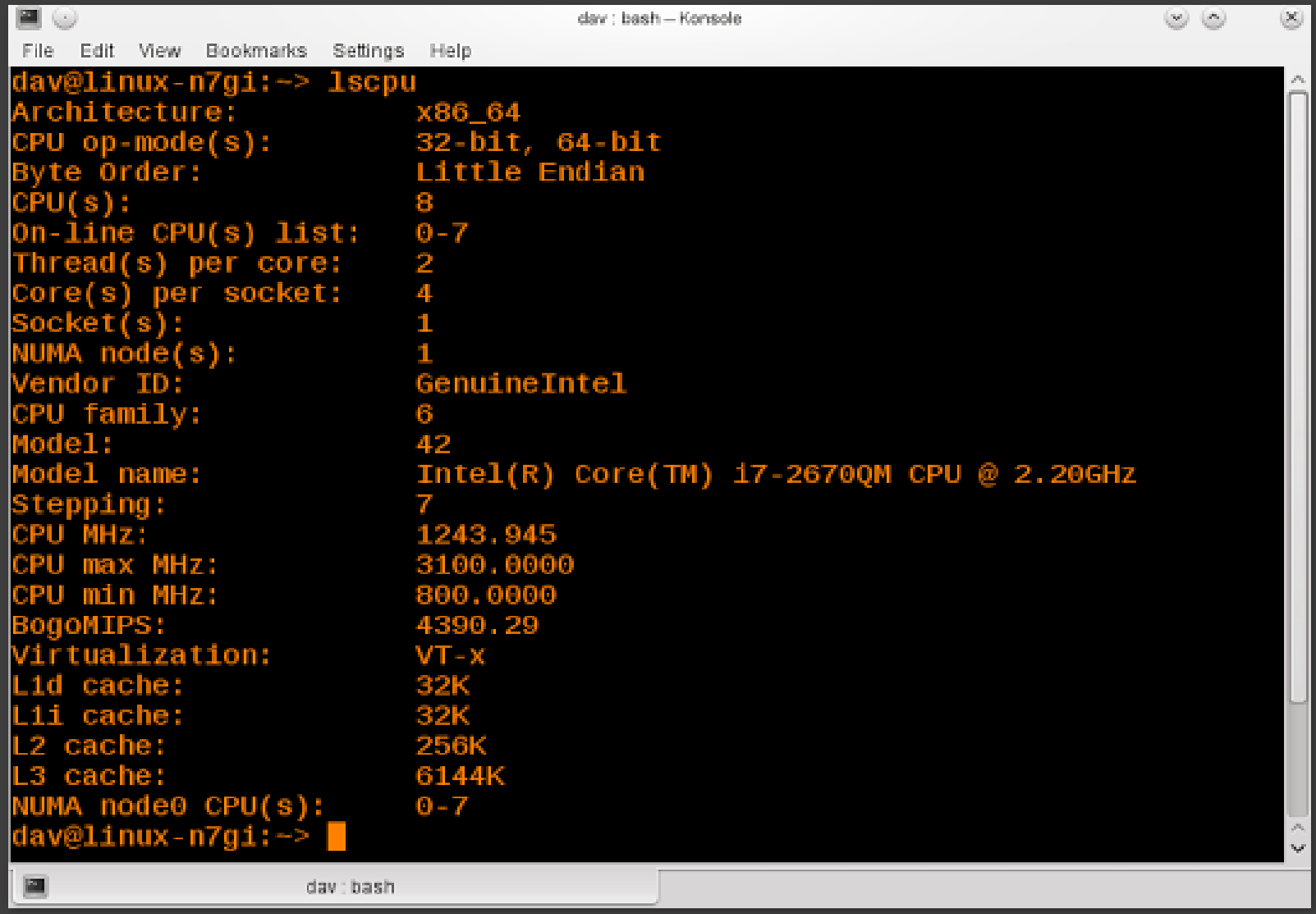
Distance between CPU and memory

- Small, but not insignificant

Exclusive access to bus may take many clock cycles

Therefore cache memory inside CPU


CPU
Core 1
L1 Cache
L2 Cache
Core n
L1 Cache
L2 Cache
L3 Cache
Memory

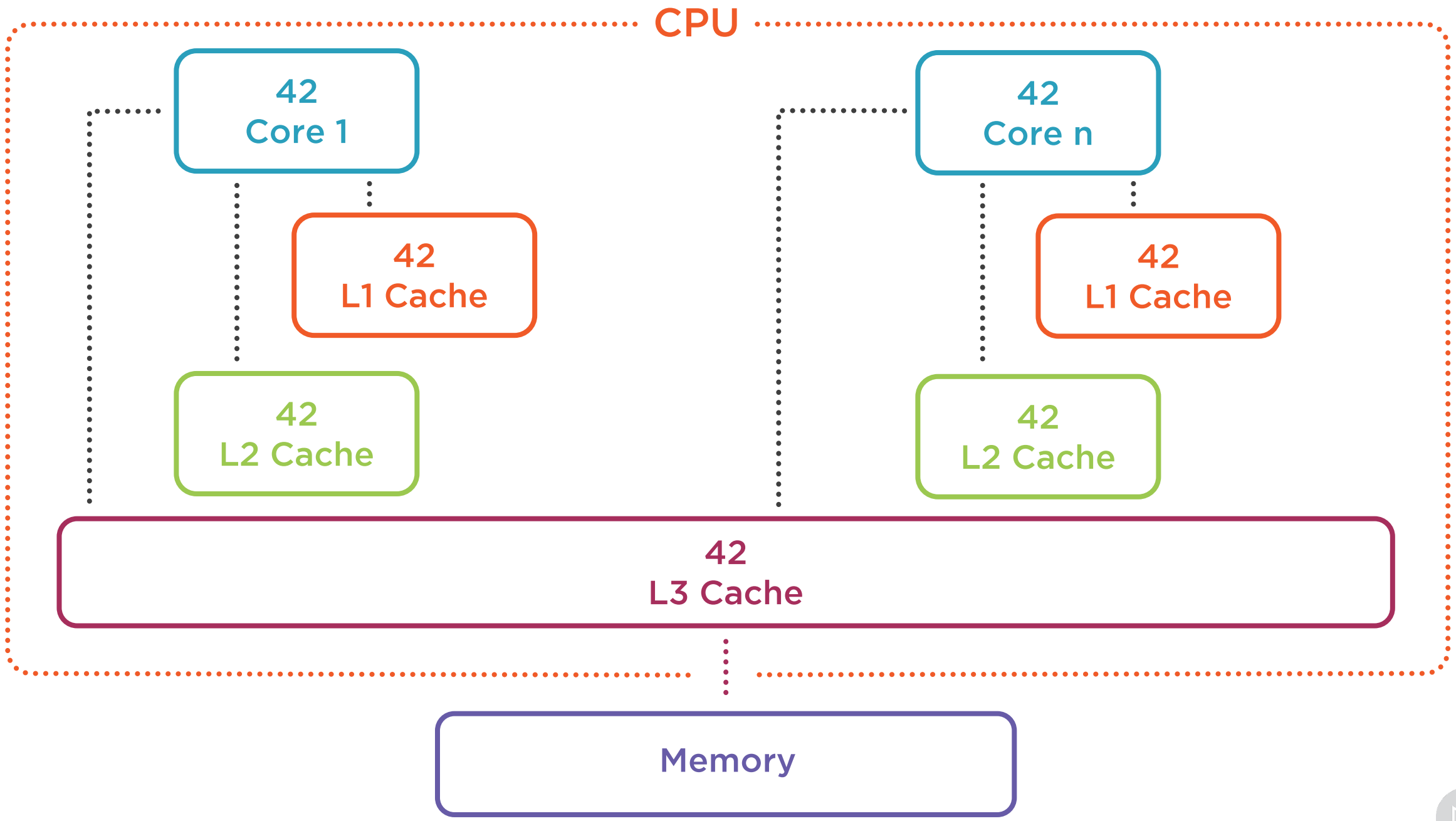

dav : bash — Konsole
File Edit View Bookmarks Settings Help
dav@linux-n7gi:~> lscpu
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 8
On-line CPU(s) list: 0-7
Thread(s) per core: 2
Core(s) per socket: 4
Socket(s): 1
NUMA node(s): 1
Vendor ID: GenuineIntel
CPU family: 6
Model: 42
Model name: Intel(R) Core(TM) i7-2670QM CPU @ 2.20GHz
Stepping: 7
CPU MHz: 1243.945
CPU max MHz: 3100.0000
CPU min MHz: 800.0000
BogoMIPS: 4390.29
Virtualization: VT-x
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 6144K
NUMA node0 CPU(s): 0-7
dav@linux-n7gi:~>
dav : bash

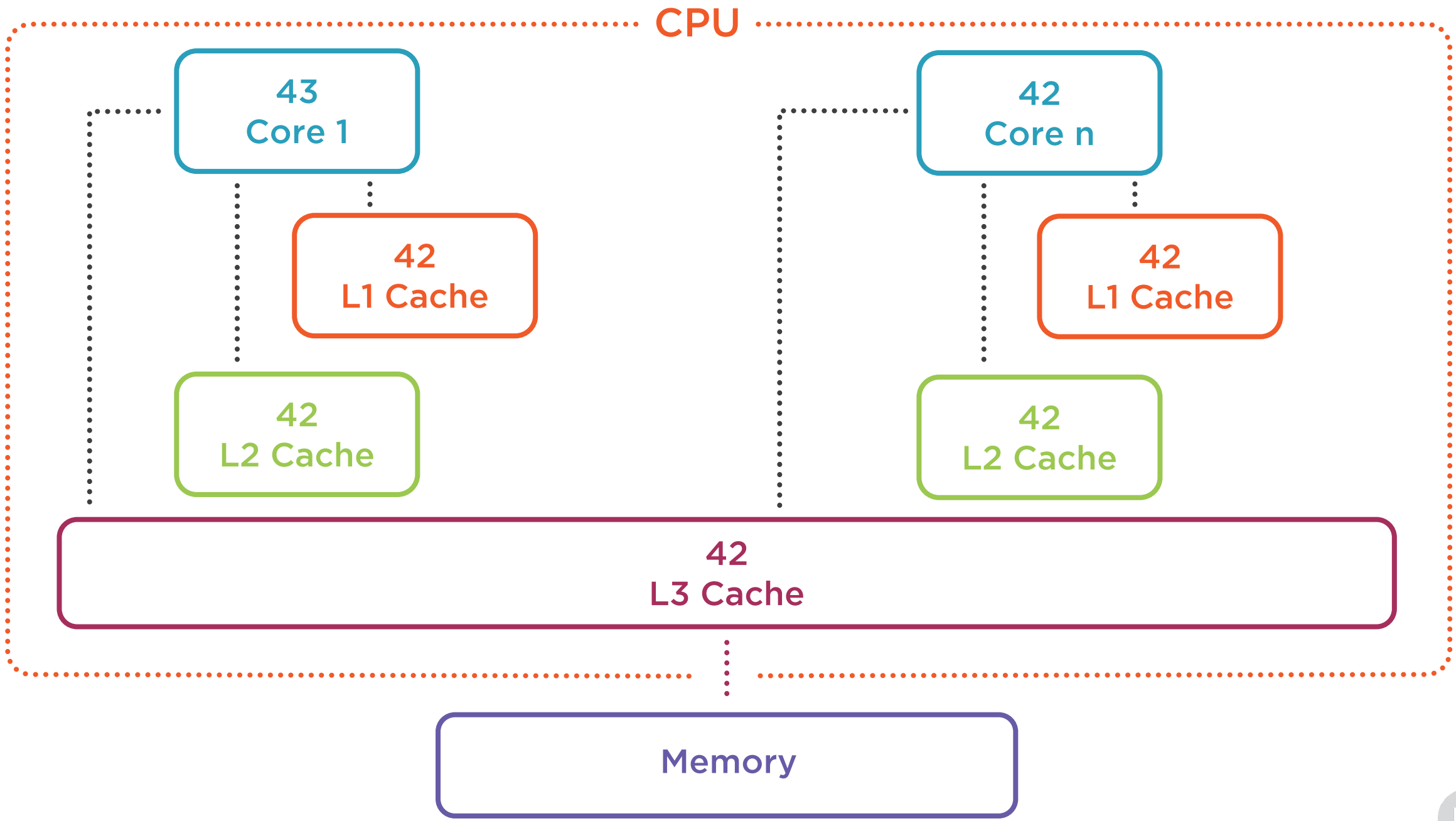
Cache Lines

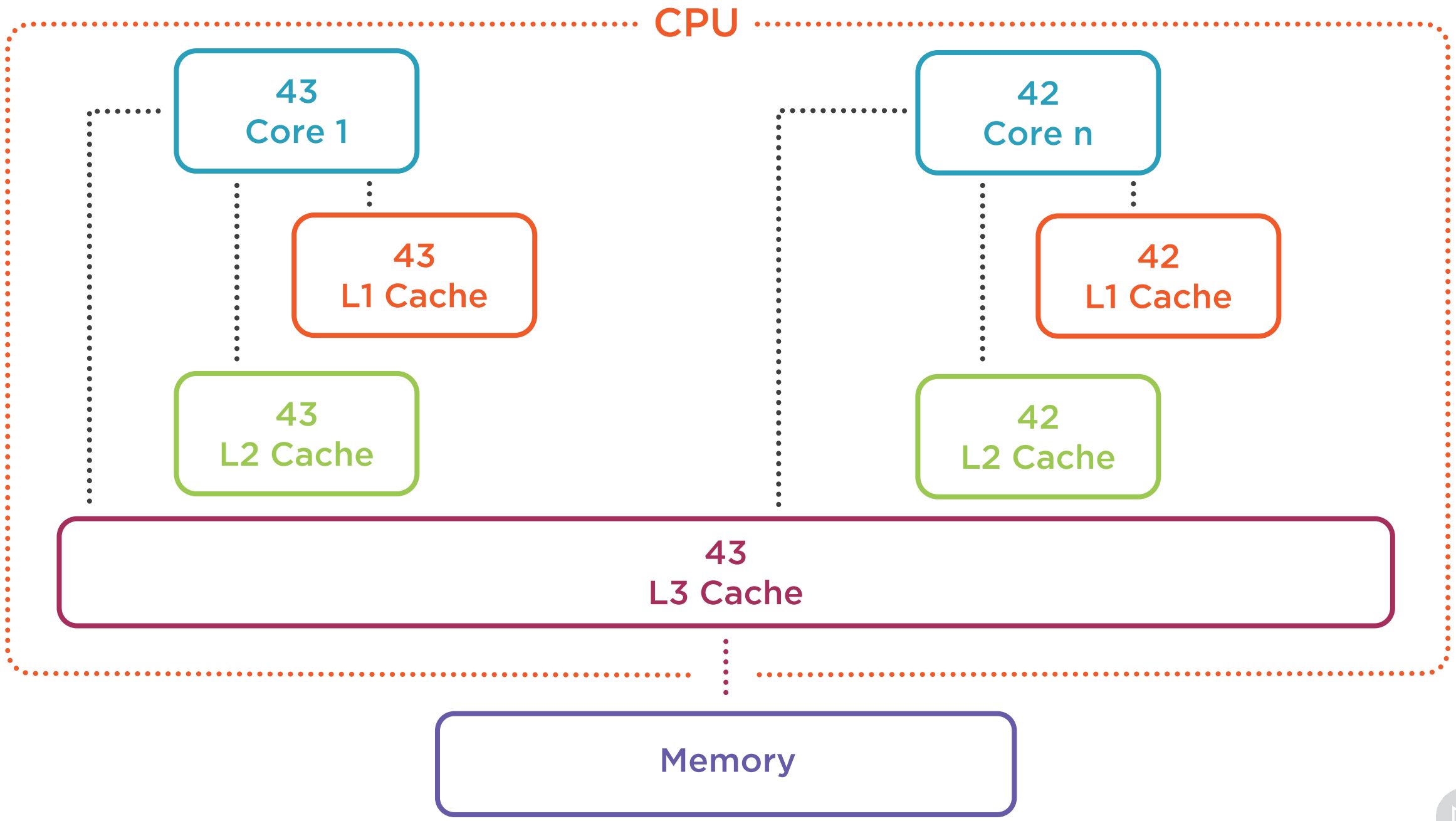
Often access contiguous memory

Word at a time transfer is slow

So transfer block into cache line in cache


CPU
42
Core 1
42
Core n
42
L1 Cache
42
L1 Cache
42
L2 Cache
42
L2 Cache
42
L3 Cache
Memory


CPU
43
Core 1
42
L1 Cache
42
L2 Cache
42
Core n
42
L1 Cache
42
L2 Cache
42
L3 Cache
Memory


CPU
43
Core 1
42
Core n
43
L1 Cache
42
L1 Cache
43
L2 Cache
42
L2 Cache
43
L3 Cache
Memory

Cache Synchronization

Synchronizing takes time

- Default: see value in own core's cache

Thread will see data it wrote out

- Assuming not modified in the meantime

Inconsistent Data

One thread updates while other reads

Data read with mix of latest and stale data

- Data structure appears inconsistent
- Breaks invariants

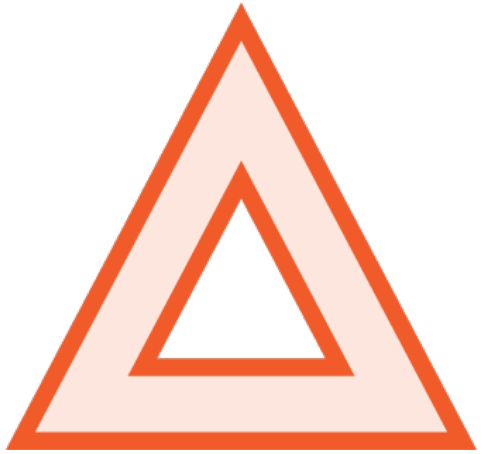
Preventing Inconsistent Data

Use monitors or locks

Or

Publish immutable copies to other threads

Suggest issues that might occur in a multithreaded system with respect to caching?



If one core writes to its local caches then another core reads, it will see stale values

- No guarantee when latest update visible to a core

Inconsistent data structures