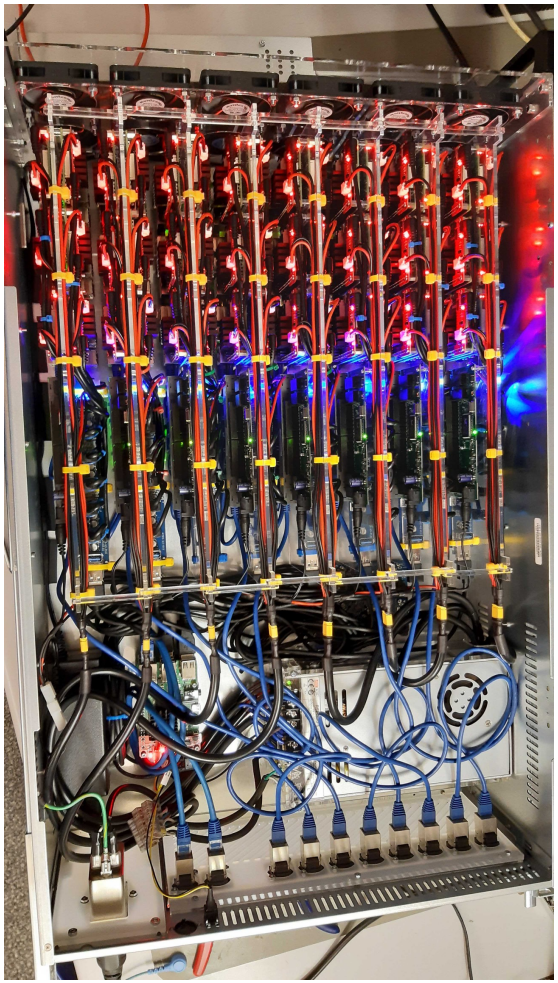
# Kernels and Tracing

Lecture 2, Part 3: Our lab environment

Prof. Robert N. M. Watson

2022-2023

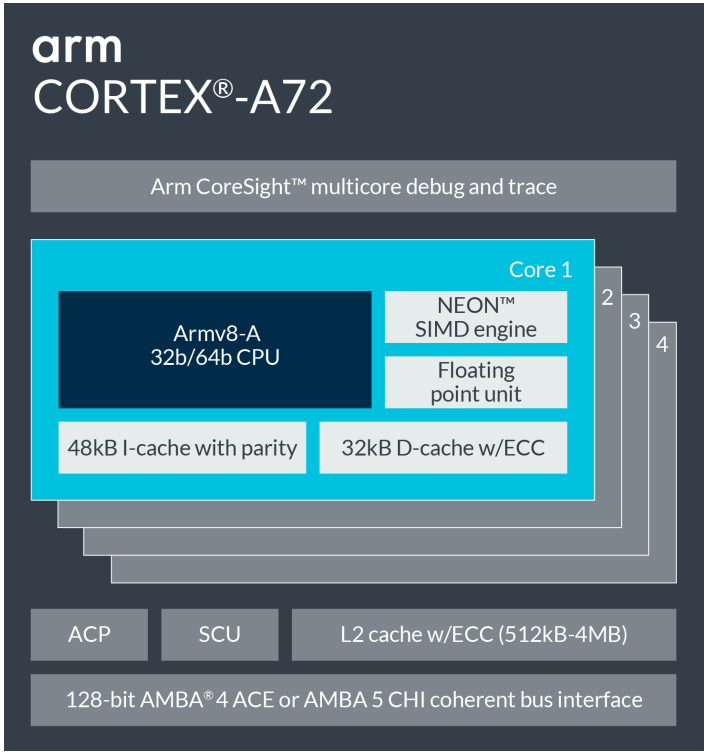
# Our lab platform: RPi4s + FreeBSD 13.x

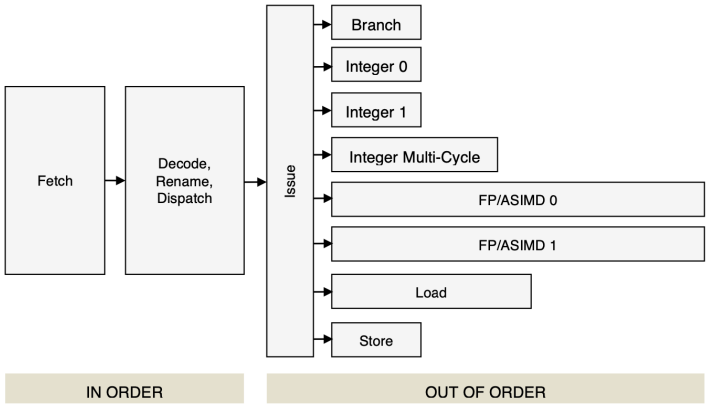


- 50x Raspberry Pi 4 boards
  - Broadcom BCM2711 SoC
  - 4x 64-bit A72 ARMv8-A cores
  - 8GB DRAM, 64G SD Card
- FreeBSD 13-STABLE
  - DTrace tracing tool
  - HWPMC counter framework
  - Bespoke benchmarks motivating OS and microarchitectural analysis
  - JupyterLab Notebook environment
- Access remotely via SSH + port forwarding for JupyterLab interface

# High-density Cortex A-72 slide

## (Some of this information will be useful only for later labs)




*** Our benchmarks use only the first core to simplify analysis**

The L1 memory system consists of separate instruction and data caches.

**Per-Core: L1 I-Cache: 48K**

The L1 instruction memory system has the following features:

- 48KB 3-way set-associative instruction cache.
- Fixed line length of 64 bytes.
- Parity protection per 16 bits.
- Instruction cache that behaves as Physically-indexed and physically-tagged (PIPT).
- Least Recently Used (LRU) cache replacement policy.
- MBIST support.

**Per-Core: L1 D-Cache: 32K**

The L1 data memory system has the following features:

- 32KB 2-way set-associative data cache.
- Fixed line length of 64 bytes.
- ECC protection per 32 bits.
- Data cache that is PIPT.
- Out-of-order, speculative, non-blocking load requests to Normal memory and non-speculative, non-blocking load requests to Device memory.
- LRU cache replacement policy.
- Hardware prefetcher that generates prefetches targeting both the L1 data cache and the L2 cache.
- MBIST support.

**Shared: L2 Cache: 1M**

The features of the L2 memory system include:

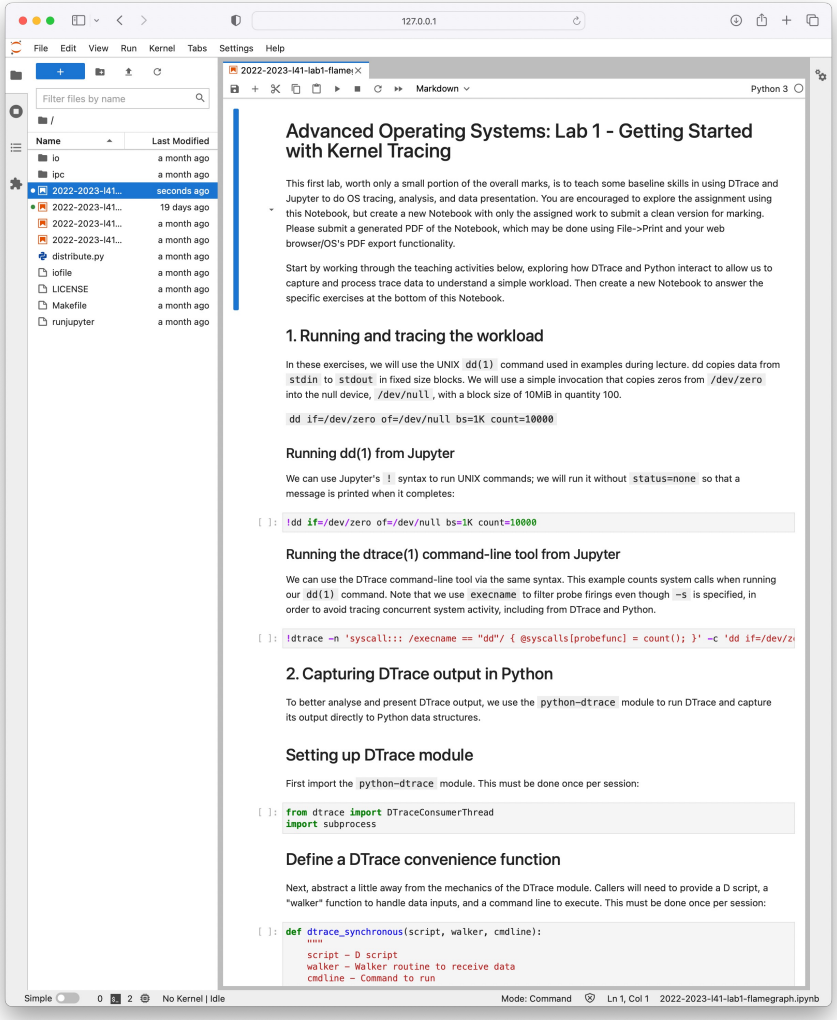
- Configurable L2 cache size of 512KB, 1MB, 2MB and 4MB.
- Fixed line length of 64 bytes.
- Physically indexed and tagged cache.
- 16-way set-associative cache structure.

**Per-Core: MMU**
**I-TLB: 48, D-TLB: 32, L2-TLB: 1024**

The MMU has the following features:

- 48-entry fully-associative L1 instruction TLB.
- 32-entry fully-associative L1 data TLB for data load and store pipeline
- 4-way set-associative 1024-entry L2 TLB in each processor.
- Intermediate table walk caches.
- The TLB entries contain a global indicator or an Address Space Identifier (ASID) to permit context switches without TLB flushes.
- The TLB entries contain a Virtual Machine Identifier (VMID) to permit virtual machine switches without TLB flushes.

# JupyterLab




- Web-based interactive Python(++) environment
  - Runs on the RPi4, with UI reached via your web browser tunneled over SSH
  - “Notebooks” blend code, text, data, and plots
- Data analysis + plotting is best done with JupyterLab
  - … but you might find the DTrace command-line client easier to work with for casual use

# JupyterLab – the UI

Markdown

Drop-down menu selects cell type

```
Buffer size:  4194304
Buffer size:  8388608
Buffer size:  16777216
'Benchmark run completed'
```

Executed cell (number)

## Plot the collected data

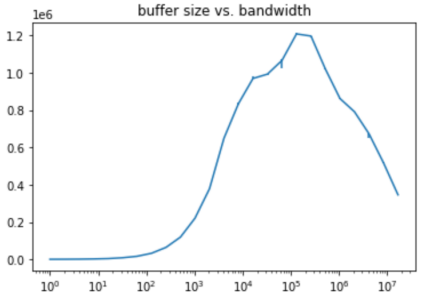
Markdown cell

Finally, we generate a plot using `matplotlib`, consisting of medians and error bars based on IQR:

[3]:
```
fig1, ax = plt.subplots()
ax.set_title("buffer size vs. bandwidth")

x_coords = []
y_coords = []
low_errs = []
high_errs = []

for x in [2**v for v in range(25)]:
    x_coords.append(x)
    y_coords.append(medians[x])
    low_errs.append(q1s[x])
    high_errs.append(q3s[x])

ax.set_xscale("log")
ax.errorbar(x_coords, y_coords, [low_errs, high_errs])
plt.show()
```

Current cell

Code cell




Cell output

Ctrl-Enter in a cell executes it

In execution cells show [*]

## Create an annotated plot

In analysing this plot, it is worth considering key inflection points: Points on the plot where there are behavioural changes, and what they reflect. We can directly annotate those points on the plot using `avxline`.

In the next plot, we've manually placed several vertical lines at points where the data you collect is likely to experience inflection points. If they don't line up, check that you are collecting data as expected.

Be sure to take note of the linear Y axis and exponential X axis, and consider its implications for data analysis.

Unexecuted cell (no number)

[ ]:
```
### This content the same as the above cell
fig1, ax = plt.subplots()
ax.set_title("buffer size vs. bandwidth")
```

# Connecting to your board

- You will be assigned a dedicated RPi4; its hostname will be in the form rpi4-0*XX*.advopsys.cl.cam.ac.uk
  - You will be contacted directly regarding your board assignment and how to collect login credentials
- The RPi4 nodes are accessible via SSH from within the CUDN (Cambridge University Data Network)
  - This will apply to most students working from colleges / the department
- If you are not directly connected, you can:
  - Use the UIS or CL VPNs
  - Hop using SSH via another system on the CUDN (e.g., slogin-serv)
- You will run all parts of the lab as the root user
  - Exercise care; we can re-image toasted boards, or assign you a replacement, but data you may have on the board will be lost
- **Please get in touch directly (and quickly) if you are having problems accessing your RPi4 board remotely; test this early**

# Web access over SSH

- In addition to logging in via SSH, you will use SSH to port forward the JupyterLab web interface:

```
ssh -L8080:127.0.0.1:8080  root@rpi4-0XX.advopsys.cl.cam.ac.uk
```

- This command allows software on your workstation to connect to 127.0.0.1:8080 and be transparently connected to the same port on the remote system
  - I.e., by connecting to http://127.0.0.1:8080
- Now run JupyterLab on your RPi4 using the following command (typically with a cwd of /data):

```
jupyter-lab --allow-root --no-browser --port=8080
```

- JupyterLab will print out the URL to use it starts
  - The URL includes a cookie specific to a session of JupyterLab; there is a new URL each time you run it

# Notes on the execution environment

- **/data** is where you should store notebooks, output, etc.
  - This is where we will look for it if we need to help you, or want to check your work during marking
- **/usr/src/sys** contains synchronized kernel source code
- You are running as root – please be careful not to hose the board you've been assigned
  - We can remotely re-image, but your data will be lost
- DTrace can have a significant impact on performance for some scripts – e.g., instrumenting ":::" (all probes)
  - Try not to render your board unresponsive, if possible
  - We can remotely reset, but it risks data loss
- Please back up your data to your personal machine

# How to contact us

- Preferred: Course slack
  - advopsys.slack.com
- Also possible: Email to the lecturer
  - robert.watson@cl.cam.ac.uk

# Wrapping up

- In this lecture, we have:
  - DTrace, the kernel tracing facility we will use
  - The *probe effect* and its impact
  - Our lab environment

- Our next lecture will explore:
  - The *process model*
  - The practical implications of the process model

- Readings for the next lecture:
  - McKusick, et al: Chapter 4 (Process Management)
  - Anderson, et al. 1992. (**L41 only**)