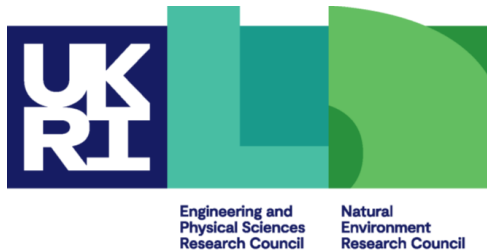


# Advanced Message-Passing Programming

---

Parallel Filesystems and Lustre



# Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

<http://creativecommons.org/licenses/by-nc-sa/4.0/>

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

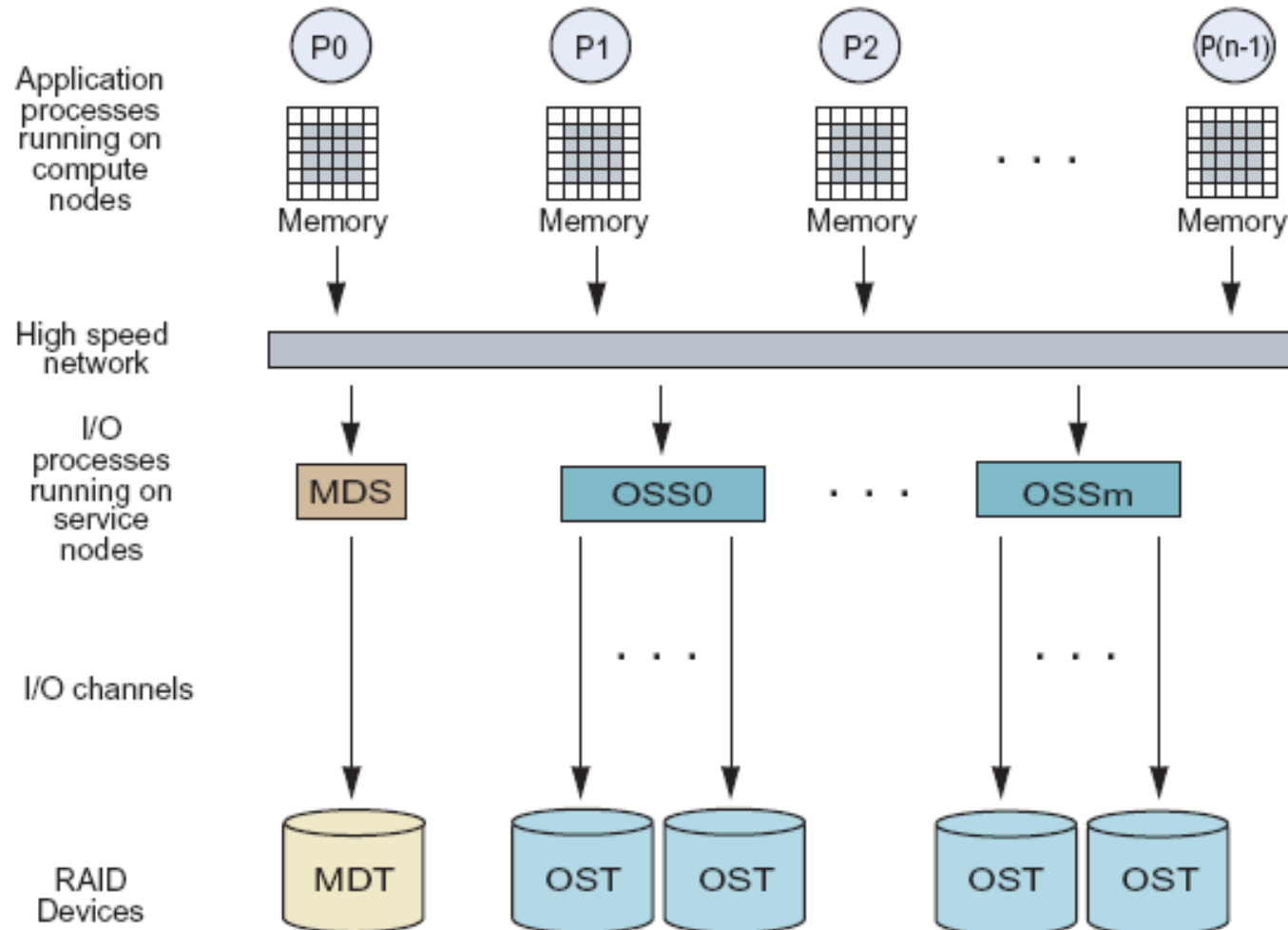
# Overview

- Lecture will cover
  - Parallel Filesystems
  - Lustre Filesystem
  - Striping
  - Simple Lustre commands
  - Bottlenecks

# Parallel File Systems

- Parallel computer
  - constructed of many standard processors, each not particularly fast
  - performance comes from using many processors at once
  - requires manual distribution of data and calculation across processors
- Parallel file systems
  - constructed from many standard disks, each not particularly fast
  - performance comes from reading / writing to many disks at once
  - requires many *clients* to read / write to different disks
    - each HPC *node* appears as a separate IO client
  - data from a single file can be *striped* across many disks
- Must appear as a single file system to user
  - typically have a single *MetaData* Server (MDS)

# Parallel File Systems: Lustre



# ARCHER's (not ARCHER2) Cray Sonexion Storage



***SSU: Scalable Storage Unit***

Multiple OSS's each with multiple OSTs



**Multiple SSUs are combined to form  
storage racks**

# Terminology

- Lustre has many different levels and virtualisations
  - e.g. one Object Storage Server has multiple Object Storage Targets
  - a single OST has many physical disks in a RAID array
- I will refer to the following parts of Lustre
  - Meta Data Server (MDS)
    - the database that contains information on, e.g., where a file is stored
  - Object Storage Target
    - the physical device that stores your data
    - I may also call this a “disk” (although it contains multiple hard drives)
- The MDS and the OSTs are what a user interacts with

# ARCHER2 hardware

- Three separate /work filesystems (work1, work2 & work3)
  - each has 12 OSTs and one MDS
  - consortia assigned to different partitions to share the load
  - multiple filesystems means the MDS is less likely to be overloaded
- One filesystem with Solid State (SSD) not spinning disks
  - we have access for this course
  - expect better latency, e.g. good for small I/O transactions
    - may also yield better bandwidth
    - possibly more reproducible in terms of performance due to fewer users
- Each has around 3.3 PiB of storage
  - total of 13.2 PiB (which is 14.5 PB !)

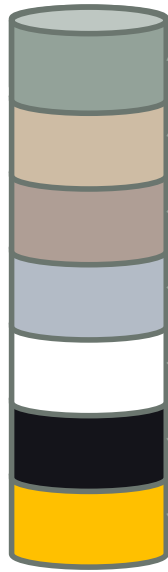
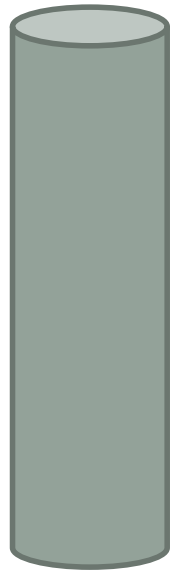


# Default Configuration

- By default, each file is stored on a single OST
  - assigned when the file is created
  - automatically distributed across available OSTs to balance the load
    - each OST is actually a separate Linux filesystem
- This is called an “unstriped” file
- Reading and writing multiple files from multiple nodes can benefit from multiple OSTs
- Access to a single file will not benefit from the parallel nature of the filesystem

# Lustre data striping

Parallel performance comes from striping single files over multiple OSTs



OS/file-system  
divides the file into  
stripes *if requested*  
*by the user*

Stripes read/written to/from  
their assigned OST

Single logical user  
file e.g.  
/work/q01/q01/user  
/bigfile.dat

# Striping

- Allow multiple IO processes to access same file
  - increases bandwidth as you are accessing multiple OSTs
- Typically optimised for bandwidth, not for latency
  - e.g. reading/writing small amounts of data is very inefficient
- This is called striping
  - striping of a file is fixed when it is created, under control of the user
  - fundamental parameters are the number of stripes and stripe size
- For example, if a file is created with a stripe count of 4
  - Lustre assigns four OSTs: OST1, OST2, OST3, OST4
  - first MiB is stored on OST1, second on OST2, third on OST3, fourth on OST4, fifth on OST1, sixth on OST2, ....
  - i.e. round-robin with default stripe size of 1MiB

# Lustre commands

- To set the striping on a directory or file

```
lfs setstripe -c nstripe <dir/file>
```

- nstripe = -1 is full striping (12 on ARCHER2's three disk filesystems)

- Stripe size: 

```
lfs setstripe -S 4m <dir/file>
```

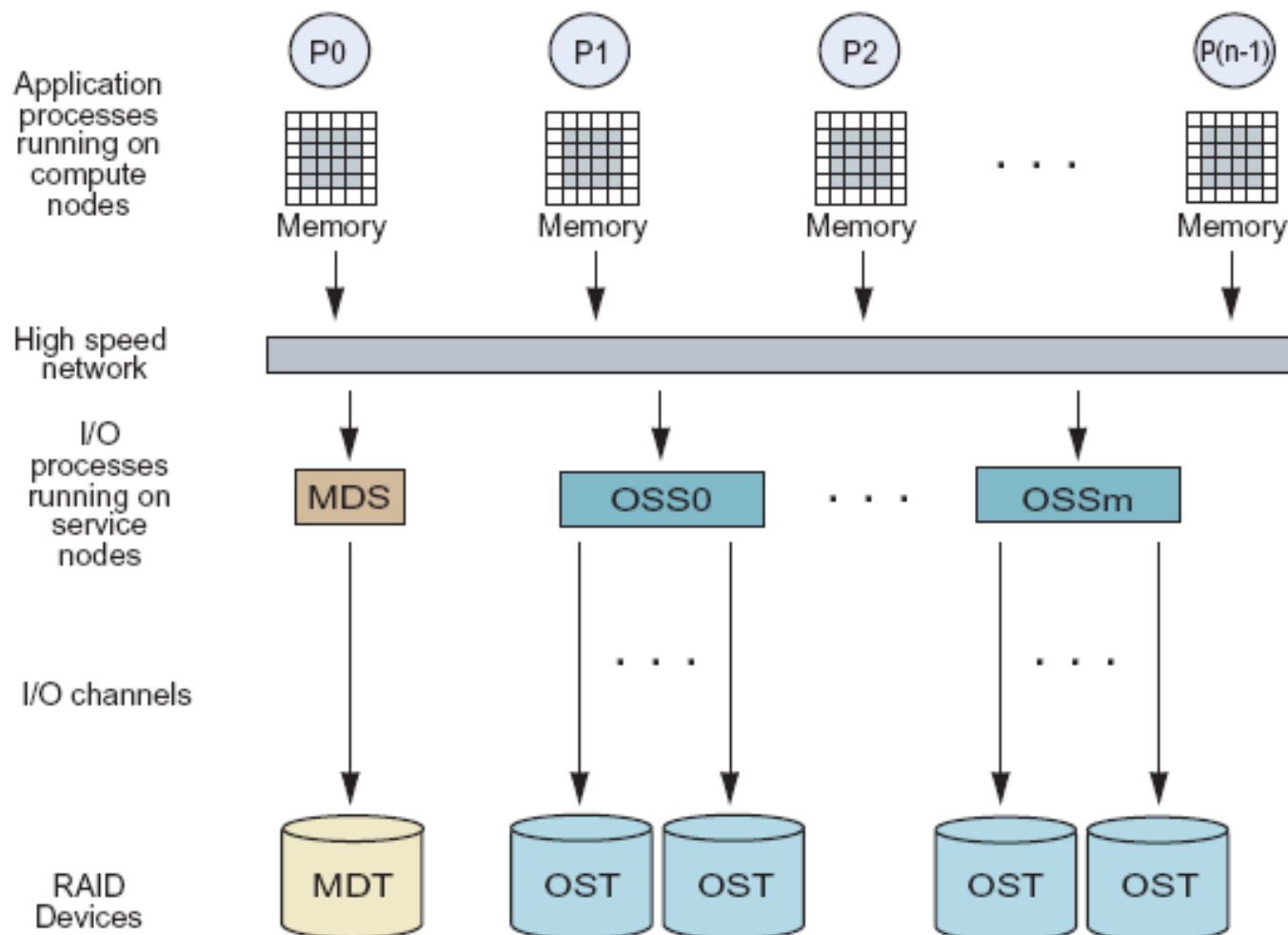
- Does **not** alter striping for existing files: that requires a copy
- I always use setstripe on directories
  - all files subsequently created in directory will have the same striping
- To enquire: 

```
lfs getstripe <dir/file>
```

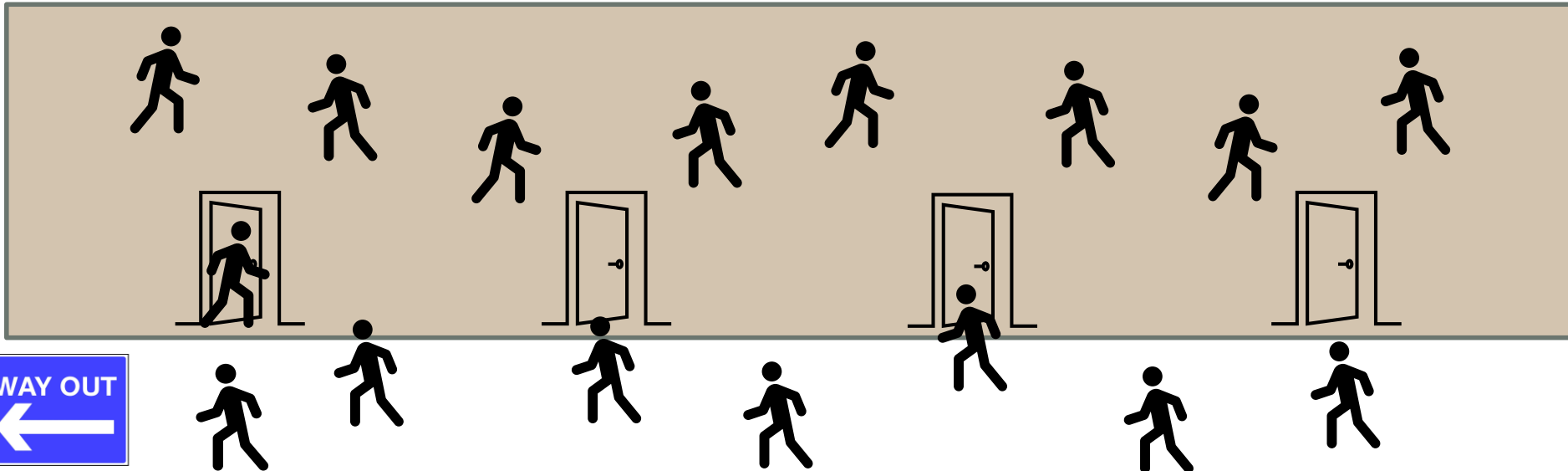
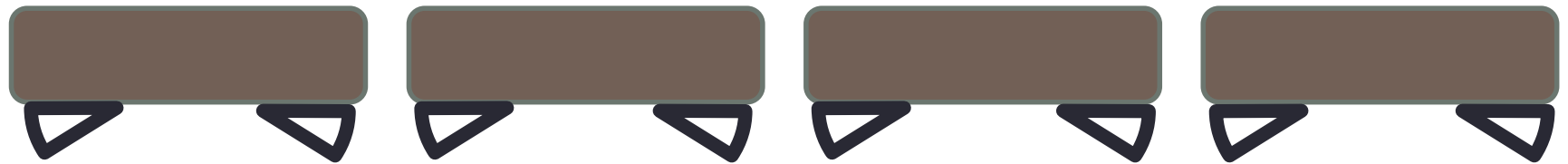
# Parallel IO to a striped file

- Very complicated in practice!
  - where in the file does the local data need to be written?
  - which OSTs are the stripes located on?
  - are there write conflicts coming from different processes?
- Need to use a parallel IO library

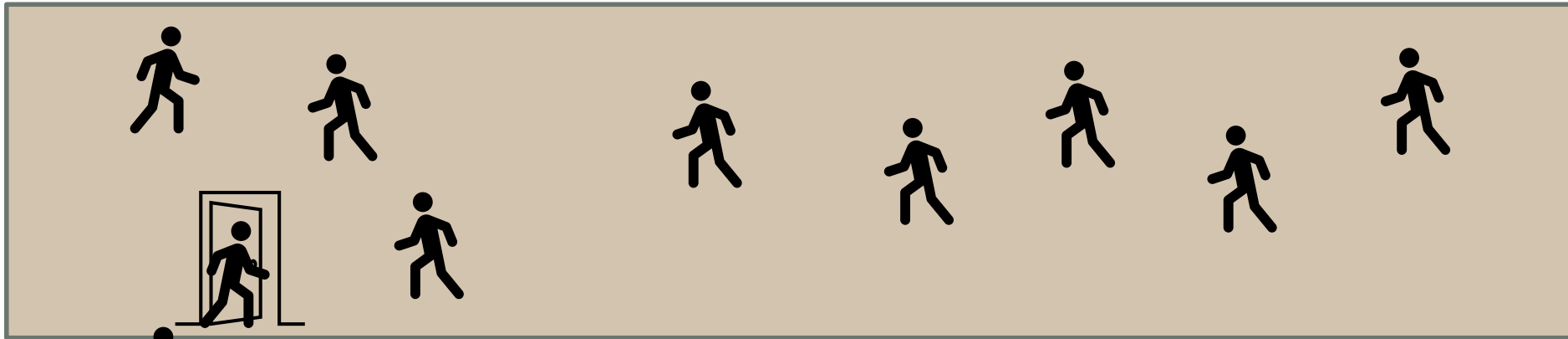
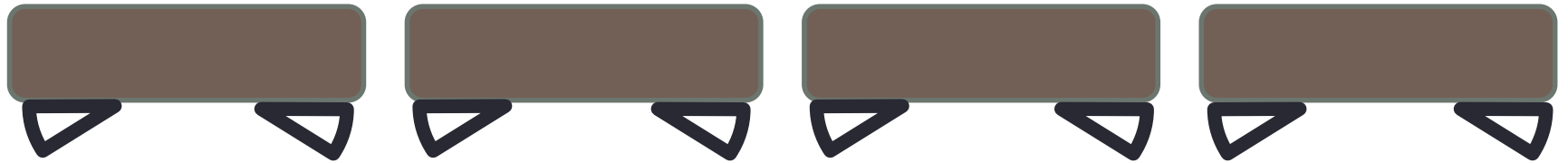
# Lustre: where are the bottlenecks?



# Train Analogy

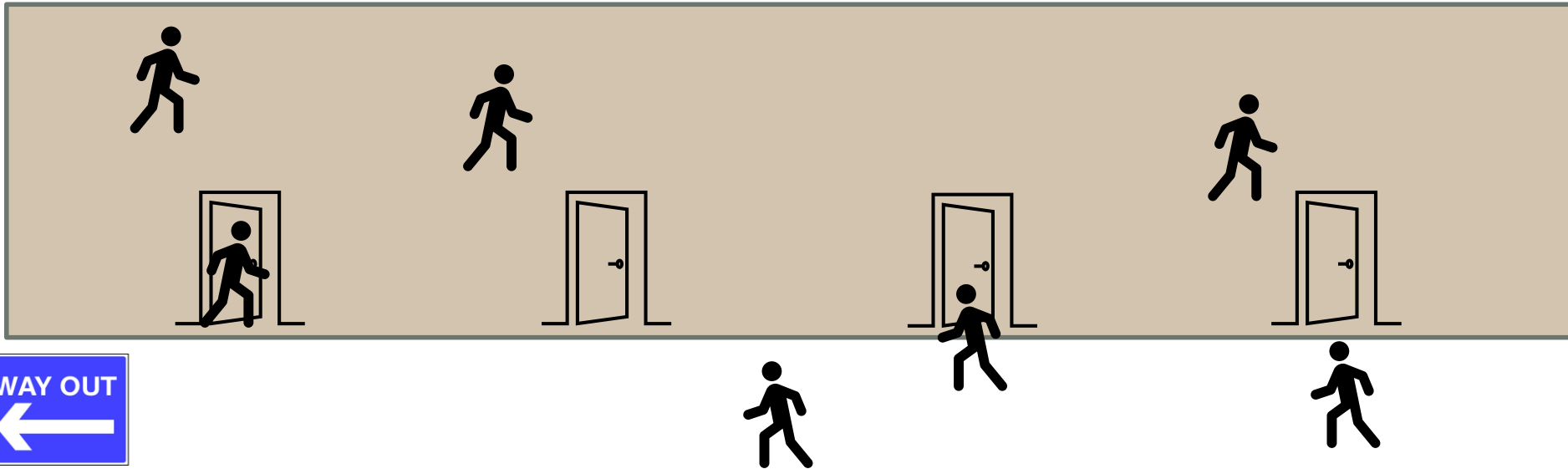
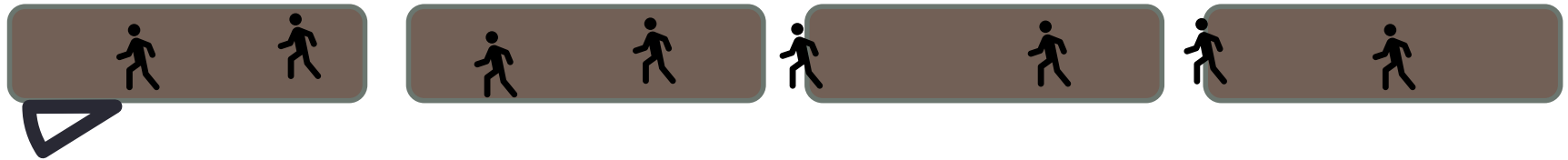


# Serial Exit Door

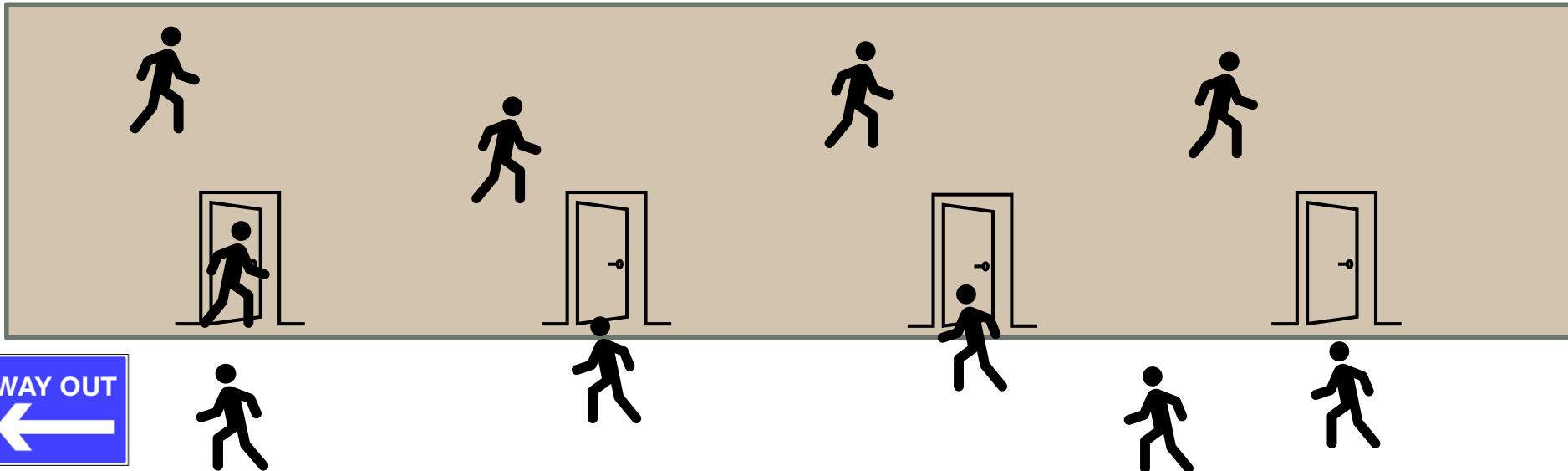
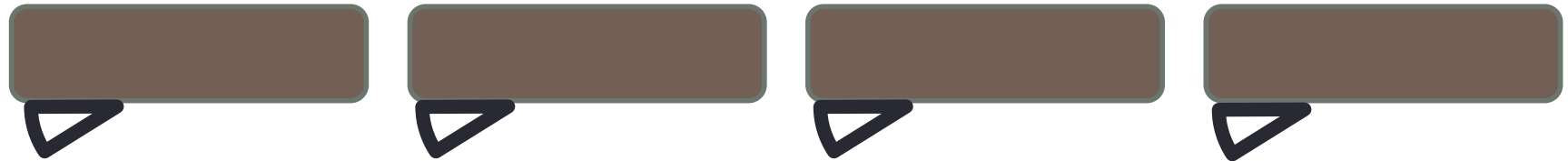




# Serial Train Door



# Reduced Train Doors



# Benchio benchmark

- Obvious questions:
  - does the MDS become overloaded for large numbers of files?
  - what is the maximum performance of a single OST?
  - can one process saturate an OST? can a node saturate an OST?
    - or is the network the limiting factor?
  - how well do different IO libraries work with Lustre?
  - what are the best stripe count (and size) settings?
  - ....
- I wrote a simple benchmark to help investigate Lustre performance characteristics and bottlenecks
  - we will use benchio for the practical examples
  - writes a large distributed 3D array of double precision numbers
  - <https://github.com/davidhenty/benchio/>.

# Cellular Automaton Model

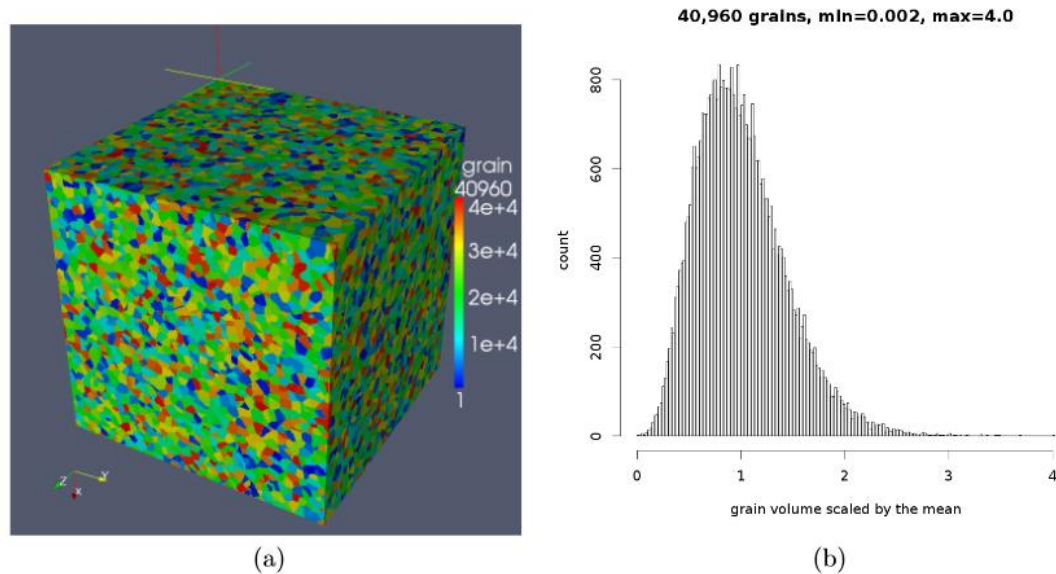


Figure 1: A  $4.1 \times 10^9$  cell, 40,960 grain equiaxed microstructure model, showing (a) grain arrangement with colour denoting orientation; (b) grain size size (volume) histogram.

- *Fortran coarray library for 3D cellular automata microstructure simulation*, Anton Shterenlikht, proceedings of 7<sup>th</sup> International Conference on PGAS Programming Models, 3-4 October 2013, Edinburgh, UK.

# Performance variability

- Share HPC resources with all users on the machine
  - network bandwidth
- Share IO resources with all users on same filesystem
  - access to OSTs
  - access to MDS
- Any of these can suffer from contention
  - IO can sometimes be very slow due to congestion

# Summary

- A Lustre filesystem has multiple OSTs
  - I think of these as being multiple disks
- By default on ARCHER2, each file stored on a single OST
  - i.e. an unstriped file with a stripe count of 1
  - increased performance for multiple files
    - a single user writing many files
    - multiple users each writing a single file
- Improving performance for a single file requires striping
  - fully under control of the user
  - expect parallel IO libraries to take advantage of striping

# Sample results

- Single process (“serial”): 1.2 GiB/s
- Multiple processes (“proc”) with 128 per node:
  - 1 node 15 GiB/s
  - 2 nodes 25 GiB/s
  - 4 nodes 50 GiB/s
  - 8 nodes 75 GiB/s
  - 16 nodes 100 GiB/s
  - results on higher node counts very variable
- I assume:
  - single proc: 1.2 GiB/s max
  - single node: 15 GiB/s max
  - single OST: 10 GiB/s max (i.e around 100 GiB/s for all 12 OSTs)