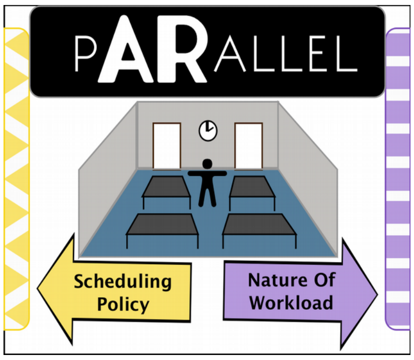
Suggested workflow for ParallelAR flashcards

For more information, visit http://parallel.auckland.ac.nz/education/parallelar

This document recommends a workflow to get started using the **ParallelAR** flashcards.

1. Overview

Select *Scheduling Policy*:
- Sequential
- Fully Parallel
- Static
- Dynamic

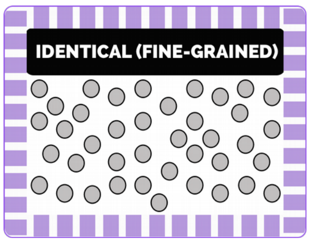


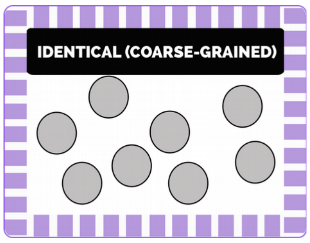
Select *Nature of Workload*:
- Identical (Coarse-grained)
- Identical (Fine-grained)
- Mixed

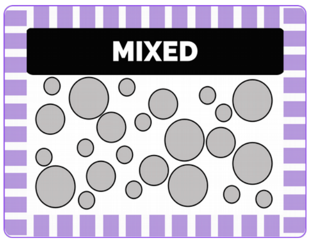
Assume there are four processor cores in the system (i.e. a quad-core processor). The general workflow involves selecting one of the ***Scheduling Policy*** flashcards, and one of the ***Nature of Workload*** flashcards. Based on the configuration, think of the following:

- How long will it take to complete all the tasks? *[Total time in seconds]*
- What do you think will be the utilization of each processor core? *[0-100%]*
- Relative to the number of tasks and their sizes, is there a large amount of overhead? When is this overhead present? What about context switching?
- Is the workload balanced fairly evenly among the worker threads?

2. Nature of Workload





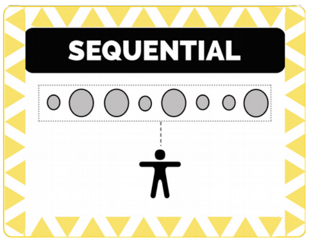


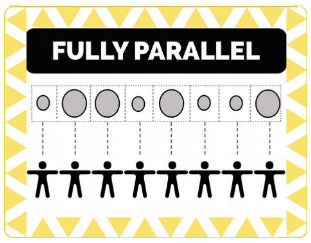
There are three types of computational workloads:

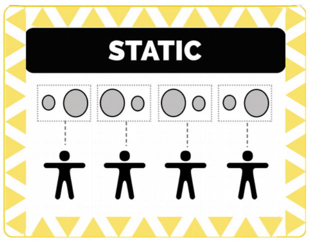
- **Identical (Coarse-grained)**
  - There are a total of **8 tasks**.
  - Each task is identical, and each task takes **4 seconds** to complete.
- **Identical (Fine-grained)**
  - There are a total of **40 tasks**.
  - Each task is identical, and each task takes **1 second** to complete.
- **Mixed**
  - There are a total of **24 tasks**.
  - There is a mixture of three types of tasks. Some take **1 second**, some take **3 seconds,** and some take **4 seconds** to complete. They are not ordered in any particular way.

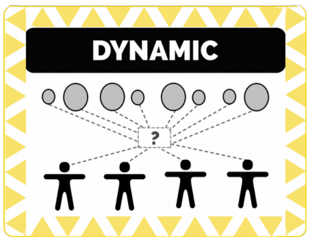
The number of circles on the flashcards corresponds to the number of computational tasks in the program.

3. Scheduling Policy









There are four types of scheduling policies:

- **Sequential**
  - There is only one thread (the main thread) that executes all the tasks.
- **Fully Parallel**
  - A new thread is created and assigned to each and every task.
  - The number of threads matches the total number of tasks.
- **Static**
  - There is one thread created per processor core (similar to Dynamic). The total number of threads is therefore four (quad-core system).
  - The main thread assigns an equal *number* of tasks to each thread. Only once the tasks have all been assigned, will the threads begin (i.e. static schedule).
- **Dynamic**
  - There is one thread created per processor core (similar to Static). The total number of threads is therefore four (quad-core system).
  - Unlike Static, the tasks are not allocated upfront. Instead, tasks are placed in a central location and each thread grabs a task when it is free. Once it completes that task, it returns to the central location and grabs another task.

4. Sample workflow, prompt questions, learning outcomes

1. To get the base performance for coarse-grained computations, start by combining the **Sequential** and **Identical (Coarse-grained)** flashcards.
   (a) How much overall time do you think it will take to complete all tasks?
   (b) What do you think the core utilization will be? Estimate a numerical value for the utilization of each core.
2. Now replace **Sequential** with **Fully Parallel** to see the impact of parallelization.
   (a) Do you think there will be an improvement in the overall time? How much?
   (b) Compare each core utilization in this round compared to the previous round.
   (c) ***Learning outcome:*** appreciate the potential of parallelization.
3. To get a base performance for fine-grained computations, select the **Sequential** and **Identical (Fine-grained)** flashcards.
   (a) How much overall time do you think it will take to complete all tasks?
   (b) What do you think the utilization will be for each core?
4. Now replace **Sequential** with **Fully Parallel**, keeping **Identical (Fine-grained)**.
   (a) How much overall time do you think it will take to complete all tasks?
   (b) How does this compare to the parallelization of the **Identical (Coarse-grained)** computations that we saw in steps 1 and 2?
   (c) Compare each core utilization in this round compared to the previous rounds.

   (d) Is the core utilization here more or less than when we used **Fully Parallel** with **Identical (Coarse-grained)**?
   (e) ***Learning outcome:*** appreciate the overhead of creating and scheduling a large number of small tasks. With an excessive number of threads, the system may struggle with context switching on the limited number of cores.
5. Now replace **Fully Parallel** with **Static**, keeping **Identical (Fine-grained)**.
   (a) Do you think there will be an improvement in the overall time? How much?
   (b) Compare each core utilization in this round compared to the previous rounds.
   (c) Why is there a change in overall time and overall core utilization?
   (d) ***Learning outcome:*** appreciate the value of reusing threads. By statically assigning tasks to threads upfront, this improves the core utilization (less overhead at runtime). This is important for fine-grained computations.
6. To get a base performance for mixed computations, select the **Sequential** and **Mixed** flashcards.
   (a) How much overall time do you think it will take to complete all tasks?
   (b) What do you think the utilization will be for each core?
7. Now replace **Sequential** with **Static**, keeping **Mixed**.
   (a) How much overall time do you think it will take to complete all tasks? Is there a worst-case or best-case scenario? Although each thread will be allocated the same number of tasks, you don't know exactly how the tasks are distributed.
   (b) Will all the threads finish at the same time? How does the performance of **Static** scheduling differ for **Mixed** versus **Identical (Fine-grained)** tasks?
   (c) ***Learning outcome:*** while static scheduling worked well when the tasks are identical, this does not work well for when the workload distribution is not balanced due to tasks having varying levels of computation.
8. Now replace **Static** with **Dynamic**, keeping **Mixed**.
   (a) Do you think there will be an improvement in the overall time?
   (b) Compare each core utilization in this round compared to the previous round.
   (c) Why is there a change in overall time and overall core utilization?
   (d) ***Learning outcome:*** appreciate the value of dynamically allocating tasks at runtime for unknown/unbalanced tasks. Although the core utilization might be sacrificed due to the overhead of runtime scheduling, this still results in an overall better-balanced workload.
9. There is a total of 12 different possible combinations to explore!