LITMUS^{RT}: A Status Report*

Created Fall 2019

Before - After - During

Sections

LITMUS^{RT}: A Status Report written in **2007**

- ➔ **Before 2007**
  Fundamentals
- ➔ **After 2007**
  Status of LITMUS^{RT} 2007 - 2019
- ➔ **During 2007**
  LITMUS^{RT}: A Status Report

SECTION 1 - BEFORE 2007

Background Before LITMUS^{RT}

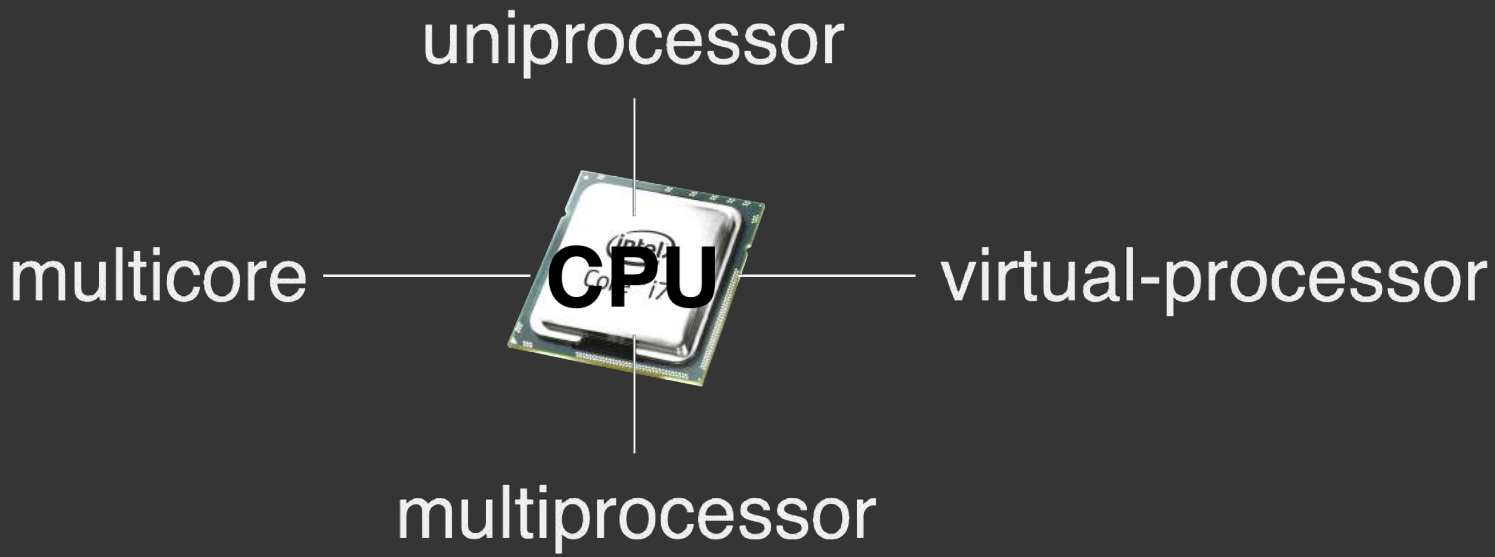
LITMUS^{RT}

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

— SECTION - BEFORE 2007

“Many … real-time Linux variants under development will be deployable on multicore & multiprocessor platforms” ~ LITMUS^{RT}: A Status Report

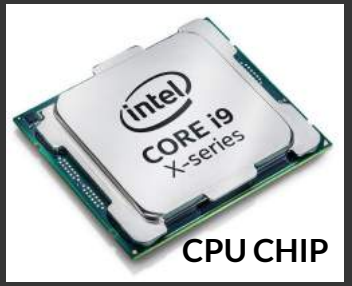
Multicore vs Multiprocessor



Multicore vs Multiprocessor

- **Uniprocessor**
  - 1 CPU with 1 execution-unit
- **Multicore**
  - 1 CPU with >1 execution-units sharing cache
- **Virtual/Logical Processor** (Hyper-Threading)
  - turns each physical execution-unit into 2 virtual units
- **Multiprocessor**
  - multiple CPUs sharing main memory

— SECTION - BEFORE 2007



HT = with Hyper-Threading

UNIPROCESSOR CHIP

execution unit

cache

UNIPROCESSOR & HT

execution unit

cache

MULTICORE CHIP

execution units

cache

MULTICORE & HT

execution units

cache

MULTIPROCESSOR WITH MULTICORE CPUs

MULTICORE CHIP

execution units

cache

MULTICORE CHIP

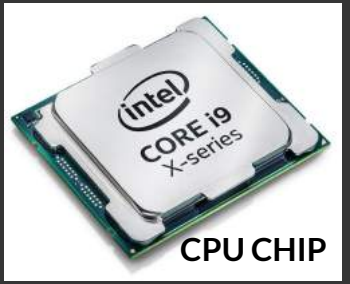
execution units

cache

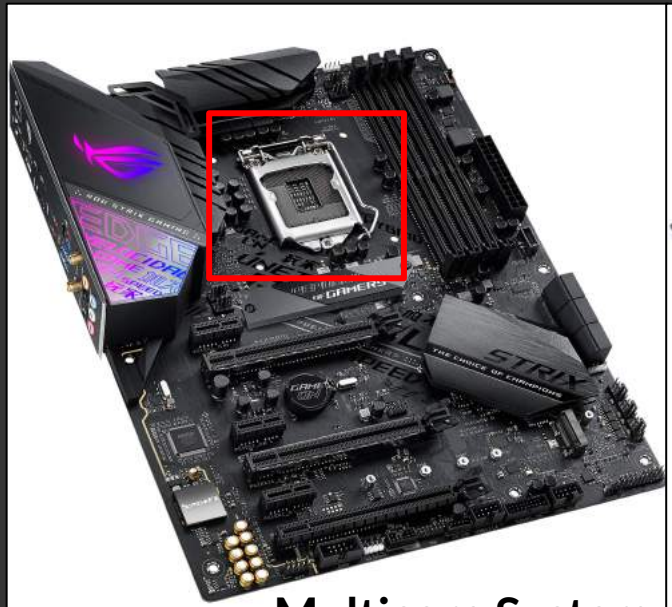
main memory

motherboard

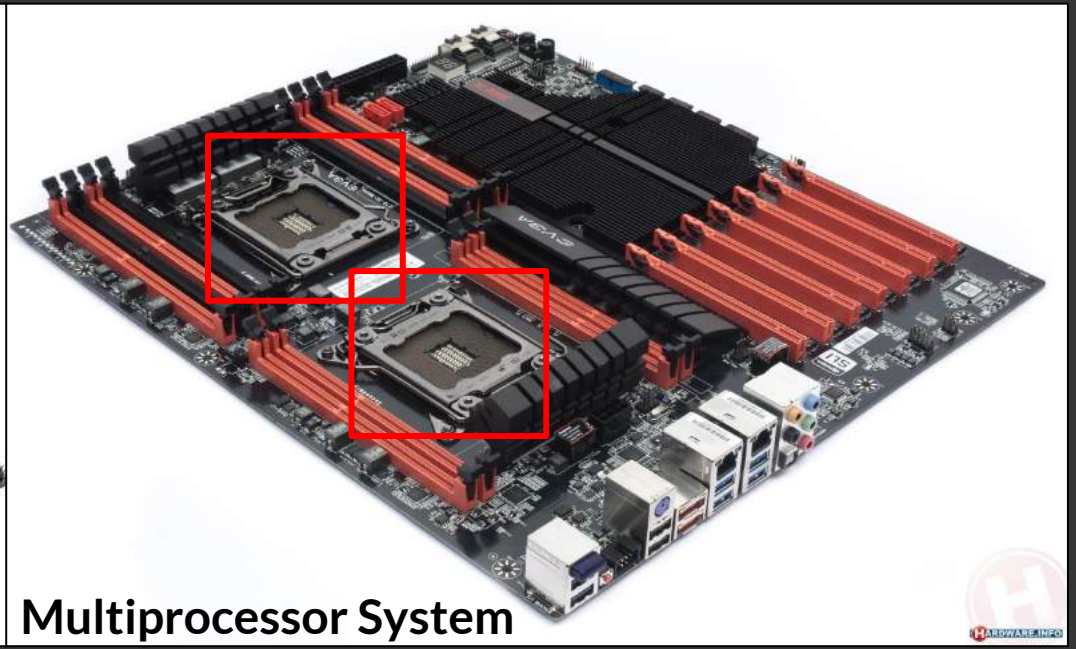
Single vs Multi Socket



CPU CHIP



Multicore System


Multiprocessor System

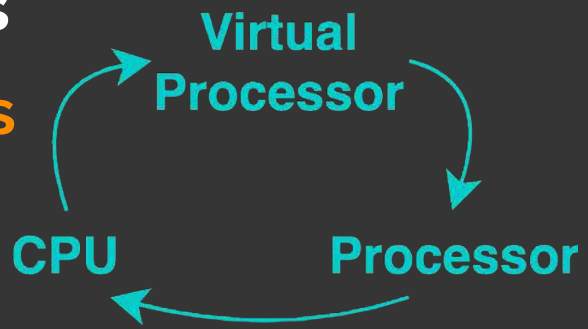
Terms Are Interchangeable

Sometimes:

- **Multicores** are called **Multiprocessors**
- **Virtual Processors** are just called **Processors**
- **Processors** are called **CPUs**
- **CPUs** are called **Virtual/Logical CPUs**

In Real-Time Systems, we use **processors**

to denote **virtual processors**



SECTION - BEFORE 2007

PUTTING IT ALL TOGETHER

linux command:

lscpu

Socket(s) ≈ # of CPU

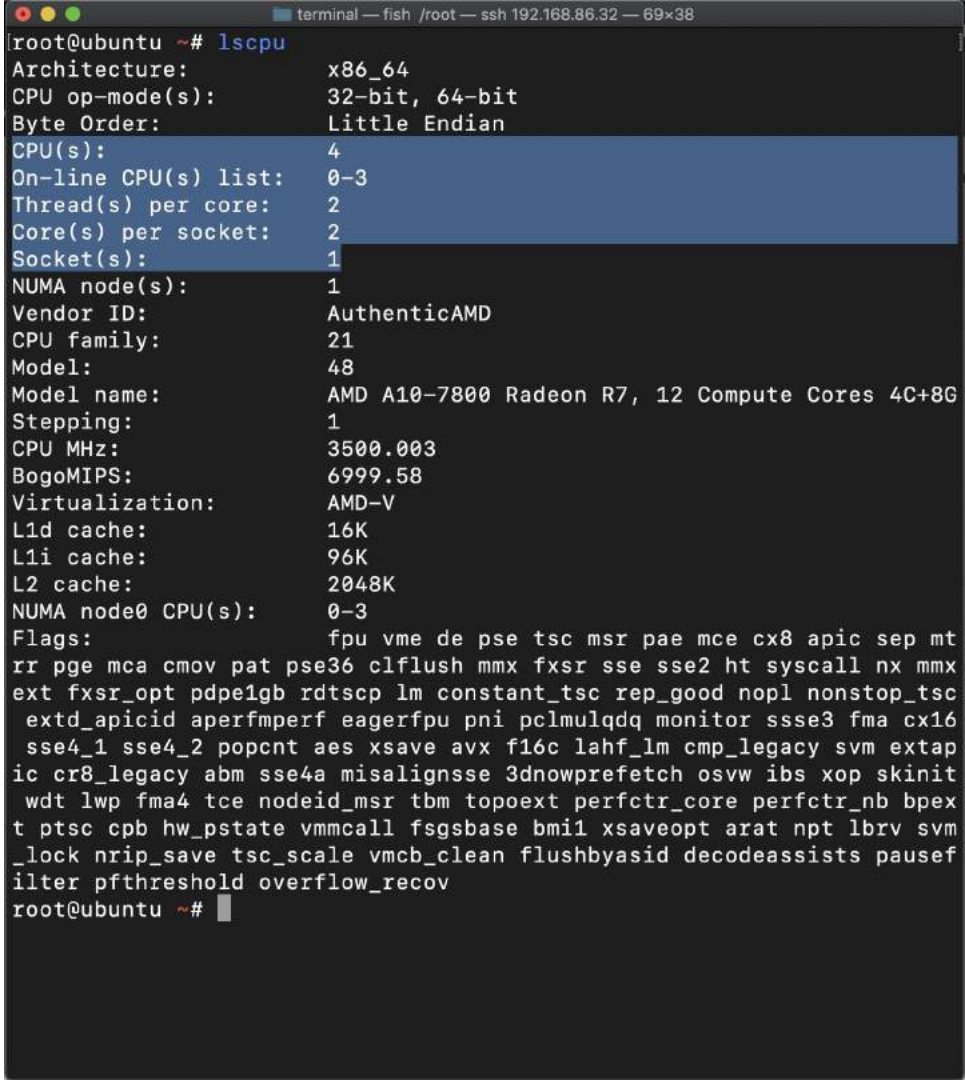
Core(s) ≈ # of cores per chip

Thread(s) ≈ Hyper-Threading

CPU(s) ≈ # of Virtual Units

LITMUS^{RT} command:

rtspin -v 1 1000 1 -p 0



Process vs Threads

Both are sequences-of-instructions that can be executed within a processor

Difference

- **Process** - processes do not share memory-stack
- **Threads** - threads share memory

In real-time systems, processes and threads are called jobs

SECTION 2 - AFTER 2007

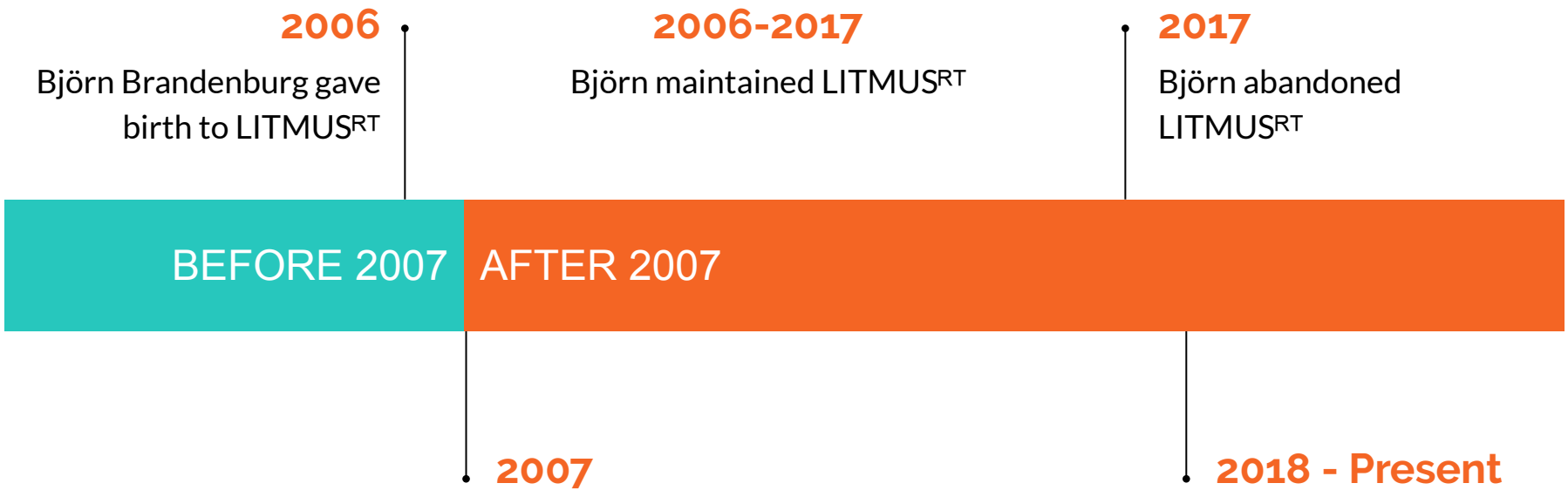
Status of LITMUS^{RT} 2007 - 2019

LITMUS^{RT}

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

SECTION - AFTER 2007

LITMUS^{RT}: A Status Report - UPDATED



SECTION 3 - DURING 2007

LITMUS^{RT}: A Status Report - 2007

LITMUS^{RT}

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

marcuschiu.com/rt.html

LITMUS^{RT}: A Status Report

SUB-SECTIONS

- ➔ **Basics**
  Fundamentals of Real-Time Systems
- ➔ **Implementations**
  TODO
- ➔ **TODO**
  TODO
- ➔ **TODO**
  TODO

Tasks vs Jobs - Scheduler

Tasks ≠ Jobs

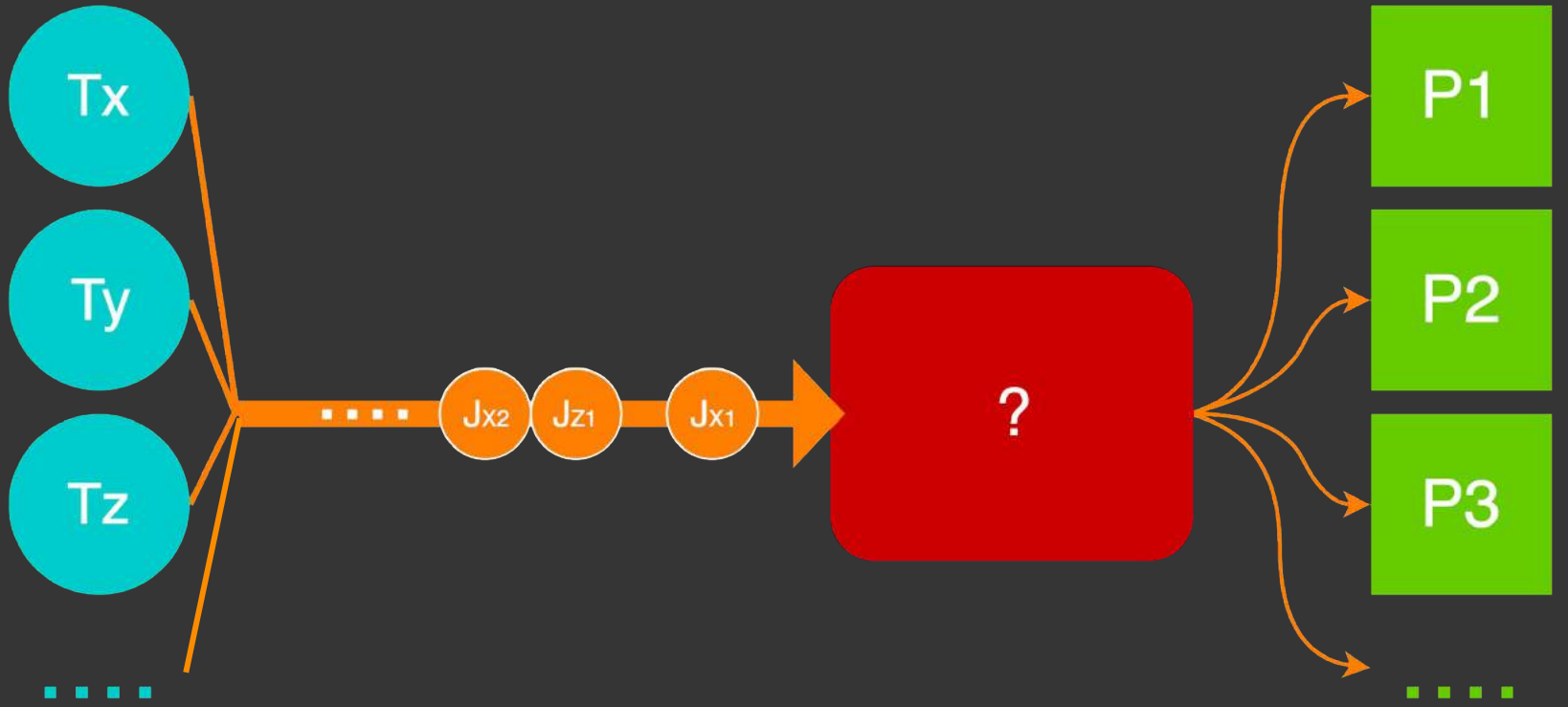
Difference

- **Tasks** - generates jobs
- **Jobs** - are scheduled, then executed

Jobs are processes. In other cases jobs can be threads

Scheduler - schedules jobs to be executed on processor(s)

Tasks Jobs Job Scheduler Processor(s)



Scheduler Properties

A Scheduler can be described with 3 properties:

1. Number of queues

   Global, Partitioned, or Clustered

2. Scheduling Order of each queue

   FIFO, EDF, Priority-Based, Round-Robin, etc

3. Whether jobs are PREEMPTIBLE or NON-PREEMPTIBLE

Scheduler - # of Queues

Global - one queue for all processors

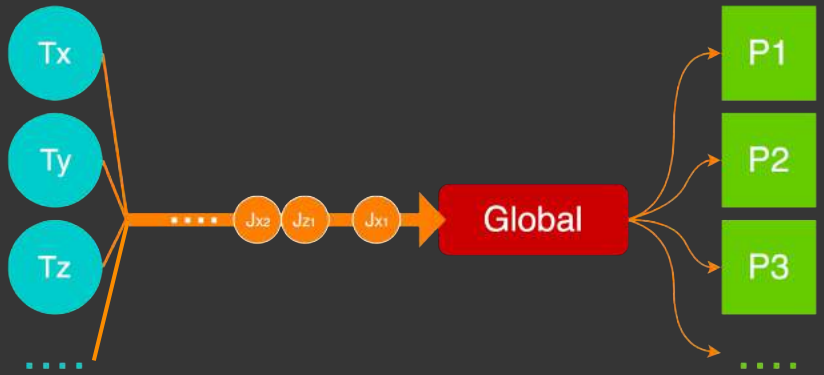
Partitioned - a queue for each processor

Clustered - a queue for a subset of processors

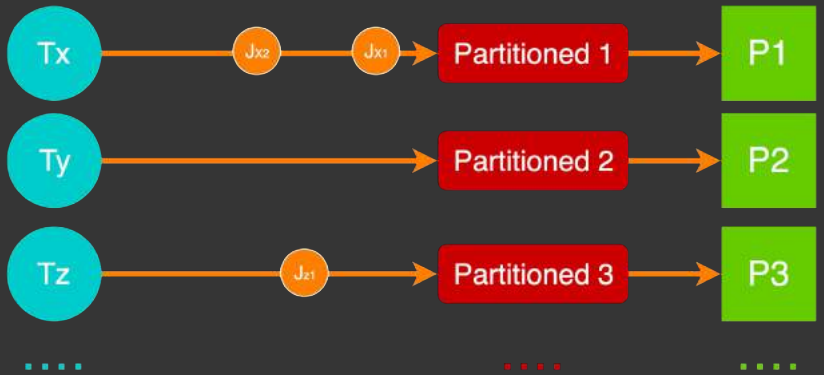
— SECTION - DURING 2007 - BASICS

marcuschiu.com/rt.html

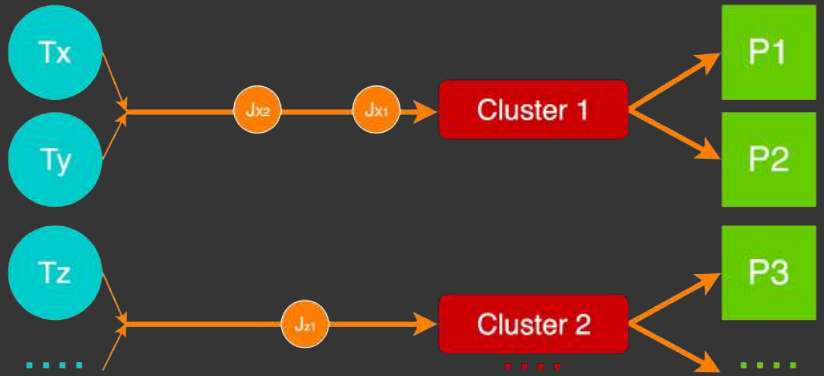
GLOBAL



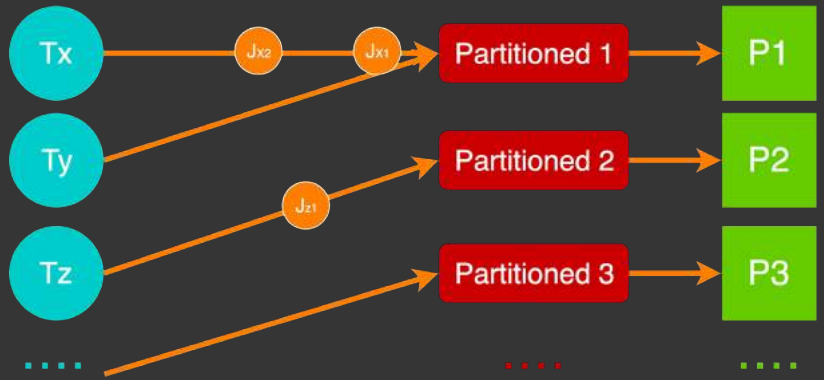
PARTITIONED Version 1



CLUSTERED



PARTITIONED Version 2



Scheduler - Scheduling Orders

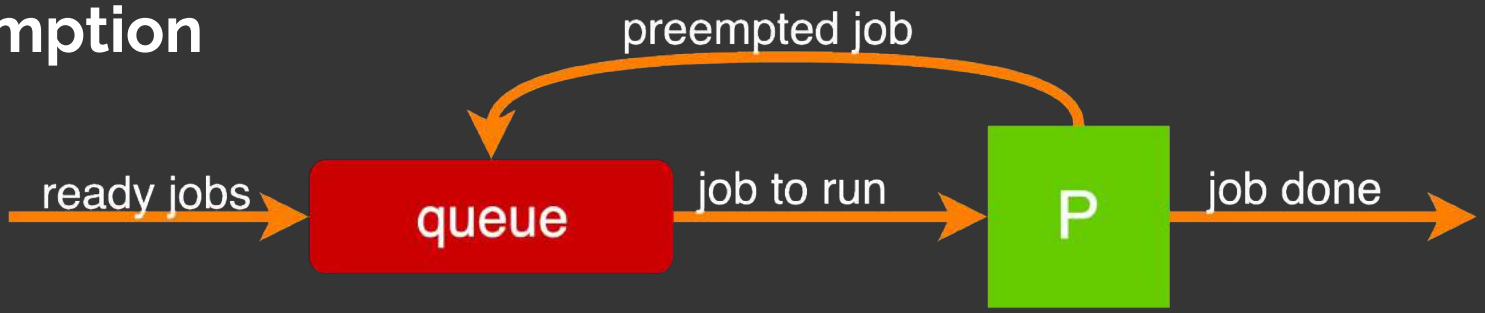
Each queue has a scheduling order

Common Scheduling Orders

- **FIFO** - First In First Out
- **EDF** - Earliest Deadline First
- **Priority Based** -
- **Round-Robin** - each job is allocated a time-slice (inherently preemptive)

Scheduler - Preemption & Non-Preemption

Preemption



Non-Preemption

Scheduling Model

Choose value for each scheduling property and you will get a **Scheduling Model**

Some Common Scheduling Models

- Partitioned EDF
- Preemptive Global EDF
- Non-Preemptive Global EDF
- etc

Pfair Scheduling & PD2

Motivation: We have learned that a Preemptive EDF scheduler for a SINGLE processor is always optimal. But is Preemptive EDF also optimal for a multiprocessor system?

NO

Pfair Scheduling - solves this by dividing each task into equal length subtasks. Subtasks are scheduled via EDF

An implementation of Pfair Algorithm is called PD2

Which Scheduling Model is the Best?

It really depends on the types of tasks at hand

"... for hard-deadline tasks, P-EDF and PD2 are usually preferable, while for soft-deadline tasks, G-EDF and G-NP-EDF are better" ~ LITMUS^{RT}: A Status Report

Thanks!

For more information check out
http://confluence.marcuschiu.com