# Multicore scheduling

Introduction to Scheduling in a Multicore environment

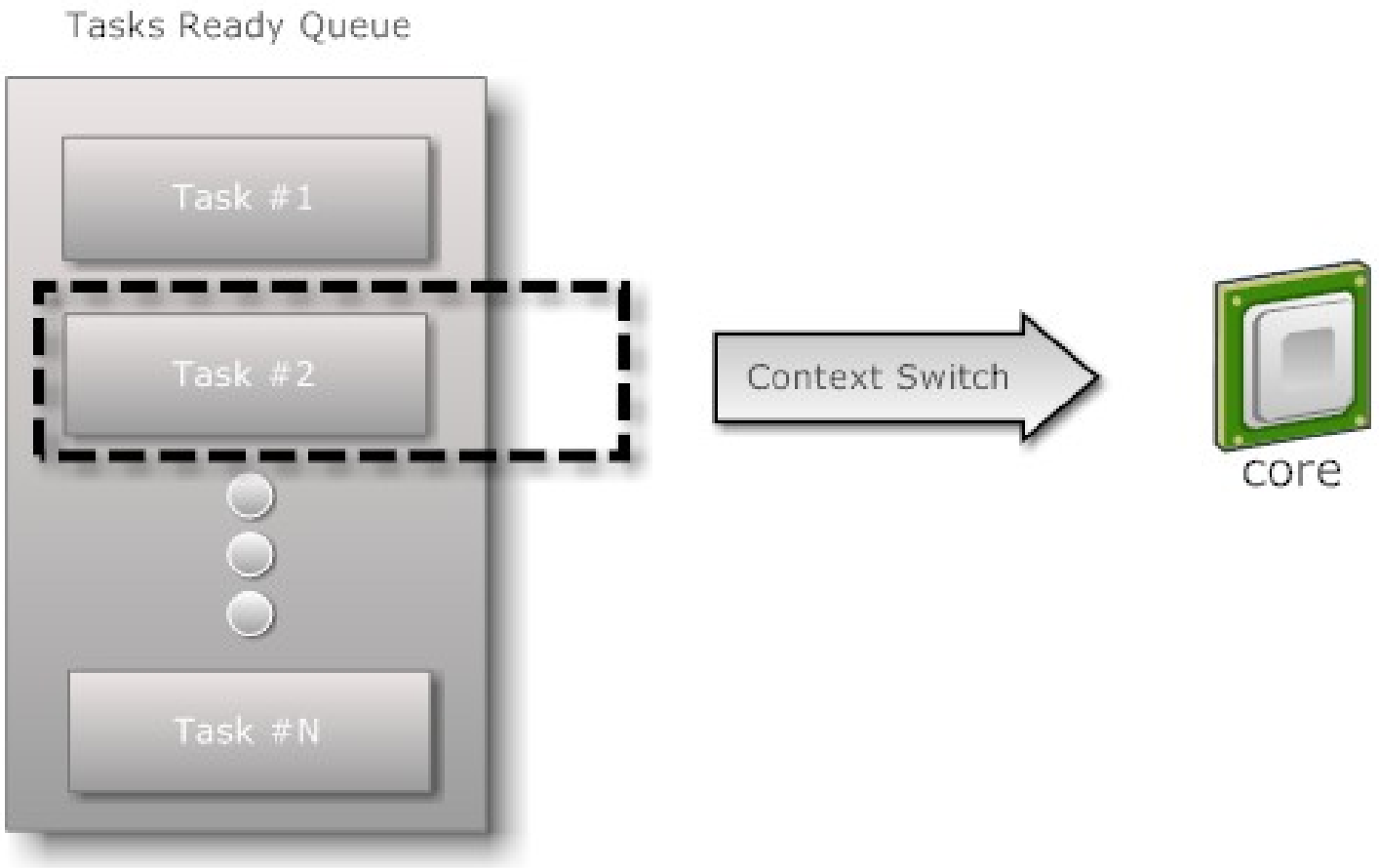
# Roadmap

- Brief introduction on scheduling for monocore system.
- Introduction on Multicore Scheduler.
- Linux Multicore Scheduler implementation.
- Toro Multicore Scheduler implementation.

# Monocore Scheduler Workflow

1. The scheduler is invoked
   - quantum time was reached.
   - A more privileged task is ready to be executed.
2. A new task is selected using some algorithm
   - RR, FIFO, etc
3. The Scheduler implements a task switch
4. The task starts or continues the execution

# Monocore Scheduler Workflow



# Multicore Scheduler

- In a multicore environment each core runs a scheduler.
- Each core has a queue of tasks.
- It is similar to monocore scheduler, BUT:

  it has to deal with:

  - **Task migration.**
  - **Performing balance**
    - *keep all cores as busy as possible*
  - **Protection for inter-core access.**

# Multicore Scheduler

This new feature brings **new problems** that **don't exist** in a **monocore system**.

The **problems** with the ***migration*** are:

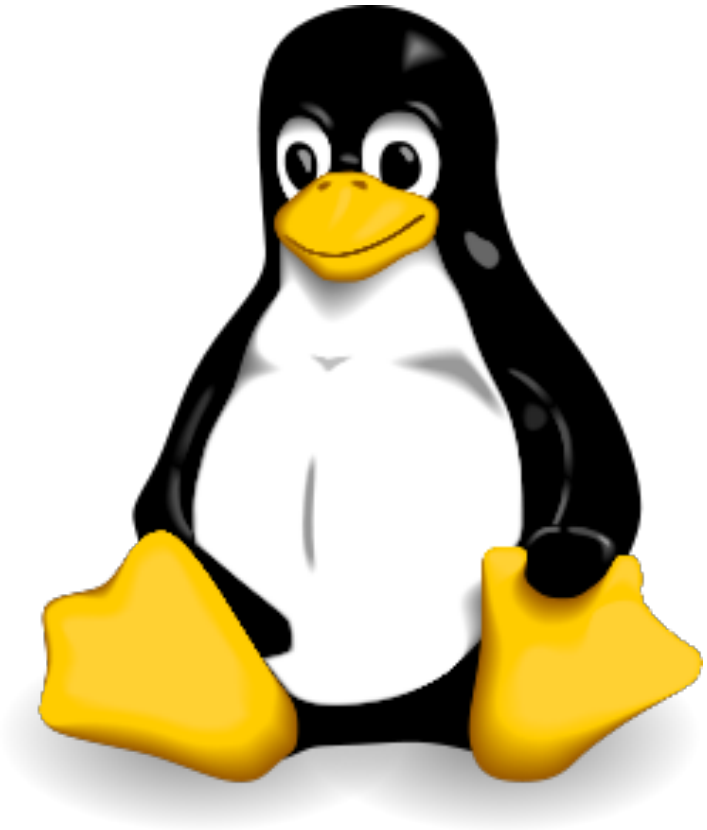
- It needs **communication** between cores.
  - in many core systems the communication might become not scalable.
- '***cold cache***' problem.
  - when a task is migrated, at the beginning of the execution  the cache is *'cold'.*

# Multicore Scheduler

Those **problems** causes **unpredictable variations** in the **execution times**.

Note: It could be critical for  real time purpose

# Linux Multicore Scheduler Implementation



# Linux Multicore Scheduler

- When tasks are created, they are placed on a given CPU.
- We will not know where it will run
  - initial allocation of tasks to CPUs is likely suboptimal.
- How does it solve?
  - Every 200ms balancing is performing

# Linux balancing Workflow

1. Select the busiest runqueue.

2. Select a task suitable to be migrated.
   a. it must not be running
   b. it should not be a "hot cache".

3. Locks both queue
   a. The number of task that will be moved at a time is set by *sched_nr_migrate*.
   b. Interruptions must be disabled in this operation.
      i. latencies for real-time app

# Linux Multicore Scheduler

*Migrations* in linux works but we still have the following problems:

- Cold cache.
- Migration is not a scalable procedure
  - It is implemented using SoftIRQ
  - it becomes a bottleneck in many core systems.

# TORO Multicore Scheduler implementation

# TORO brief description

- Toro is oriented to dedicated purposes.
  - hardware is dedicated to a fixed task.
  - It is a non general purpose OS.
- It must be scalable.
  - It must support many cores in the near future.
- The main idea is to give the whole control to the user.

# TORO brief description

- The Algorithm is Cooperative Tasking.
  - non preemptive
  - The task has to invoke the scheduler
  - Similar than Real Time FIFO in Linux
  - Non priorities
  - Very easy to do modification or implement other algorithm
- The user has to choose in which core a task will be created.
  - It is a parameter when the task is created.

# TORO Multicore Scheduler

- Created tasks cannot be migrated.

- Migration just happens when the scheduler is invoked.
  - In a lock-free way
  - The whole queue is migrated

# TORO Multicore Scheduler

The **Advantages** of this implementation are:

- A task is fixed to a core avoiding '*cold cache*'
- Migration is scalable
- We reduce the number of times the interruptions are disabled.

The **drawbacks** are:

- unpredictable time for migration.
- the program becomes more complex.

# Conclusion

"**Balancing** in general **is an art**. **Lots of considerations** must be taken into account. **Cache lines, NUMA and more.**

This is true with **general processes** which expect high through put and **migration can be done in batch**.

But when it comes to **RT tasks**, we really need to **put them off** to a **CPU** that they can **run on as soon as possible**. **Even** if it means a **bit of cache line flushing**.

**Right now** an **RT task can wait several milliseconds** before it gets scheduled **to run**. And perhaps even longer. The **migration thread is not fast enough** to take care of RT tasks."

Steven Rostedt, Red Hat developer, kerneltrap.org

# Thanks!

Matias E. Vara
matiasevara@gmail.com