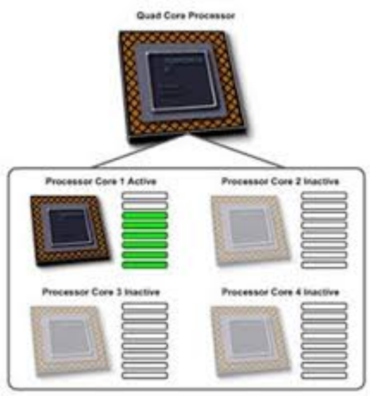
Bolt: Faster Reconfiguration in Operating Systems

Sankaralingam Panneerselvam
Michael M. Swift
Nam Sung Kim

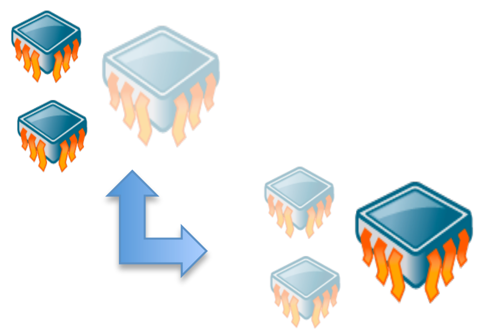
University of Wisconsin, Madison, WI

Benefits of Processor Reconfiguration



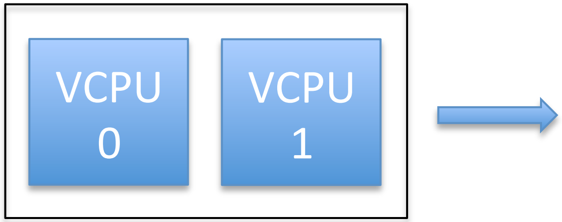
Energy Savings

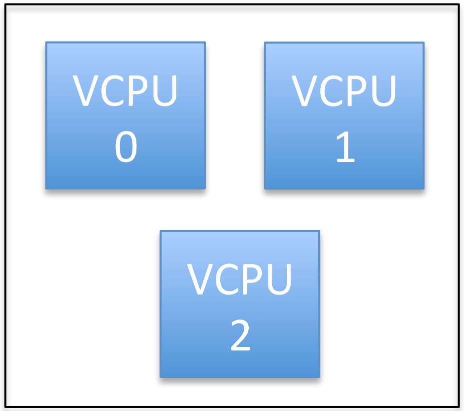
Performance Inflation



Power Constraints

Disaggregated Server Architectures





VM Scaling (e.g. IBM DLPAR)

Support for Reconfiguration

- Linux: Hotplug mechanism
  - Designed as a
    - Reliability mechanism to remove faulty cores
    - Testing mechanism to vary number of cores
  - Uncommon operation
  - Latency in the order of tens or hundreds of milliseconds

Faster Reconfiguration

- Processors enter/exit sleep state in 10 -100μs
  - Major overhead is contributed by the software
- Faster reconfiguration can enable high performance and energy efficient systems

How to achieve faster reconfiguration?

We built Bolt to reconfigure up to 20x faster than Hotplug.

Outline

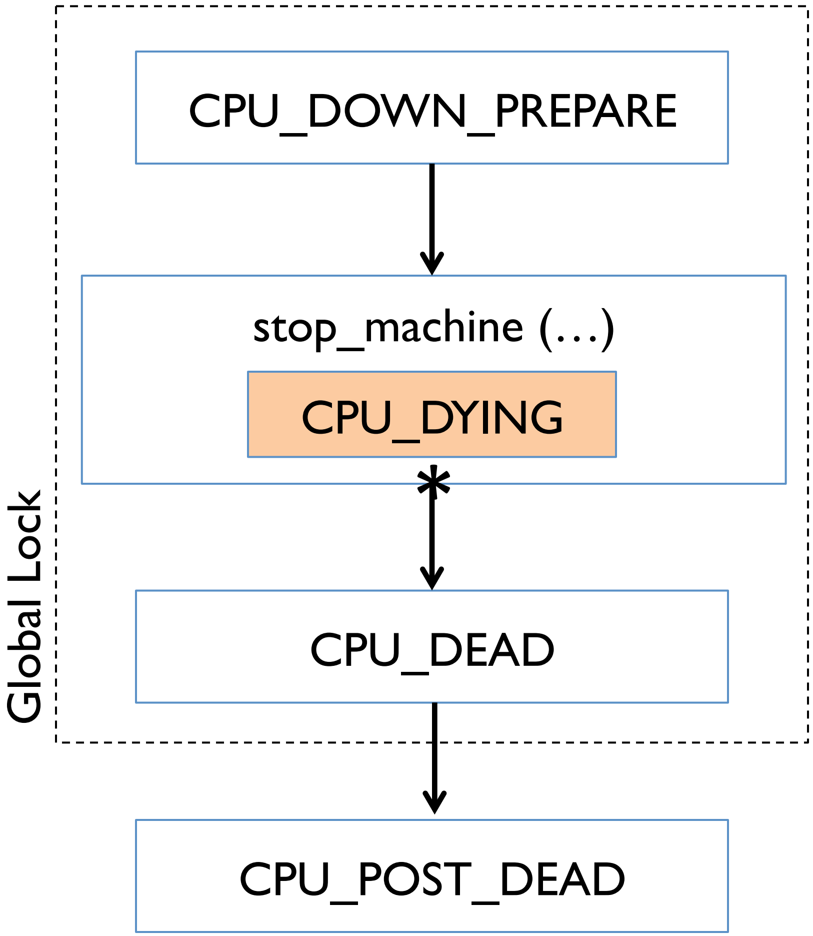
- Background on Hotplug
- Design of Bolt
- Evaluation Results

Hotplug Overview

- OS subsystems (like scheduler, watchdog …) subscribe for CPU reconfiguration events
  - 50 different subscriptions from various kernel subsystems

- Notifications sent during reconfiguration event (Addition-Online/Removal-Offline)
  - Thread migration
  - Watchdog enable/disable

CPU Offline

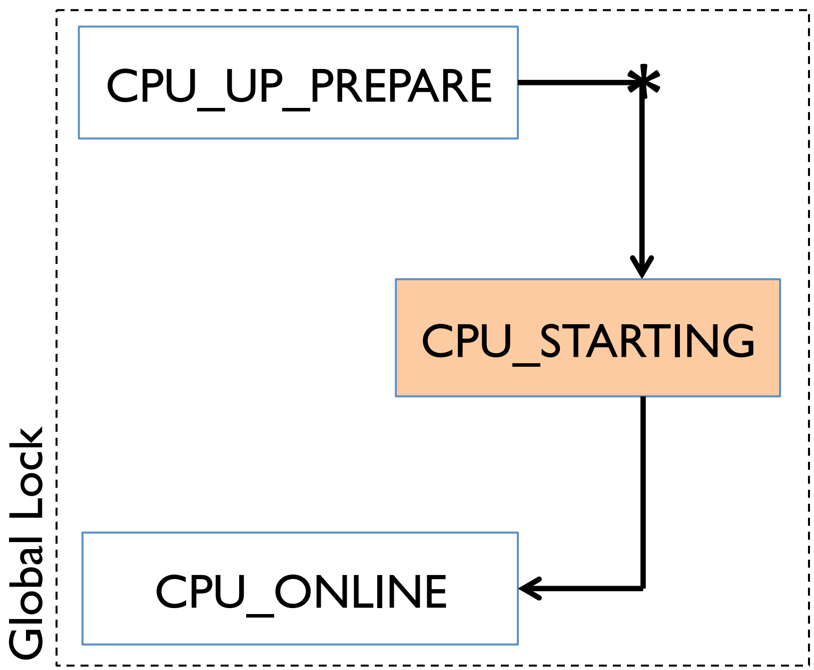


1. Notifications sent at Hotplug stages
2. Cleanup done during notifications
3. stop_machine halts the entire system
4. Global Hotplug lock serializes reconfiguration

Run in the context of CPU to be removed

* CPU is moved to offline state after the DYING notification is sent

CPU Online



1. Different message notifications are sent at various stages of Hotplug

2. Notifications allow OS to initialize software structures for the new CPU

3. Global Hotplug lock is held and thus all reconfiguration events are serialized

Run in the context of CPU to be added

* Startup Interrupt is sent to wake up the CPU

What makes Hotplug slow?

- Assumption: Reconfiguration events are uncommon

- Three limitations
  - Synchronous: All work done on critical path
  - Pessimistic: Clear/Initialize s/w structures
  - Serial: No reconfiguration of multiple cores

Outline

- Background on Hotplug
- Design of Bolt
- Evaluation Results

Bolt Overview

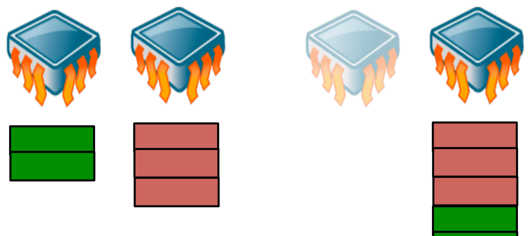
- Change in assumption
  - Online ← → Offline state changes are frequent
- Bolt handles Hotplug limitations
  - Asynchronous: Defer work from the critical path
  - Optimistic: Re-use software structures
  - Parallel: Bulk Interface to reconfigure multiple core at once

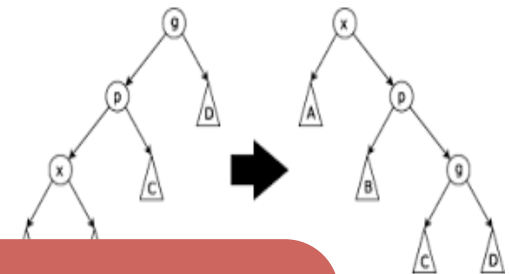
Bolt

- Bolt is built on existing Hotplug infrastructure
  - Retains subscription/notification mechanism

- Classify subsystems notification handler into
  - Critical: Need to be executed immediately for correctness (E.g. Migrating threads)
  - Non-Critical: Immediate action is not necessary (E.g. Freeing a memory structure)

Categorization of Hotplug Operations





(Thre

tion

Classify ~50 Hotplug callbacks into one of these categories

Hardware management
(Microcode update)

Software structures
cleanup/initialize
(per-cpu structures)

Global Bitmask
Updates

Critical Operations

- Handled immediately for correctness

Category	Examples
State Migration	threads in runqueue, softirqs in per-cpu queue
Hardware Management	microcode update, updating MTRR registers
Bitmask updates	global structures like cpu_online_mask
Re-organization	scheduler domain structures

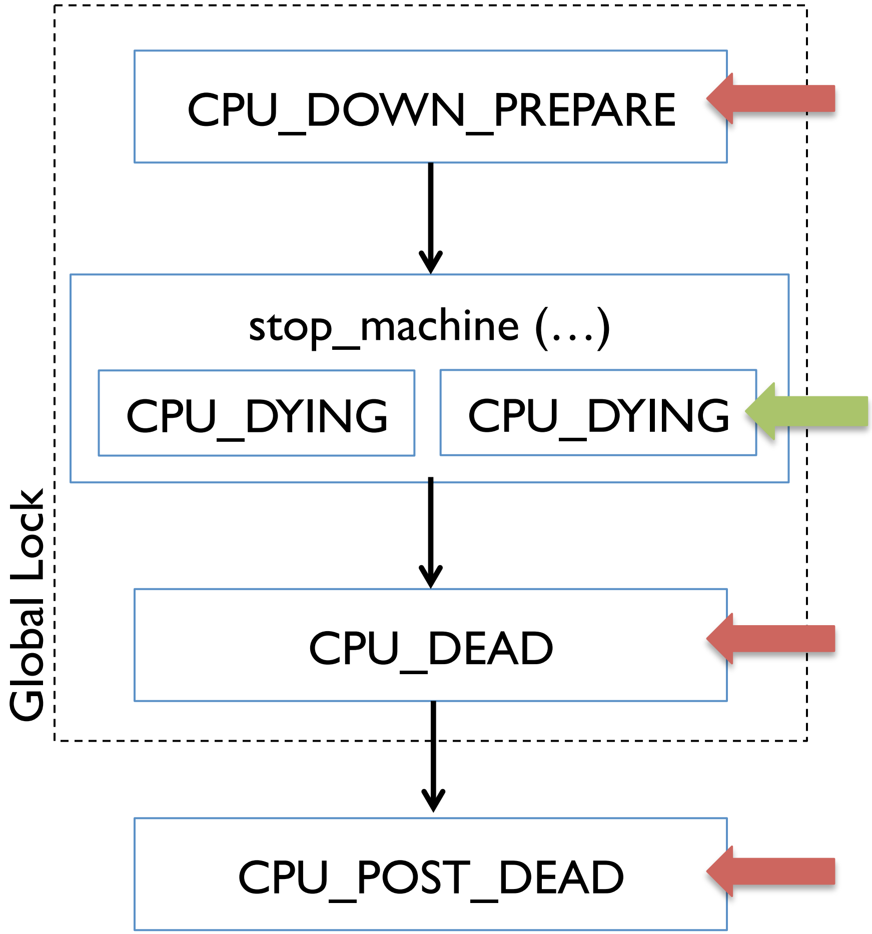
- Critical operations are handled synchronously

Non-Critical Operations

Category	How handled?
Software structures	Memory cleanup deferred to a master thread
	sysfs file access is checked during open(…) invocation
Thread operations	Parking/Unparking made asynchronous
	Thread re-use without destroying them

- ~30 out of 50 callbacks were identified as non-critical operations
- Bolt removes them from critical path but retaining the interface

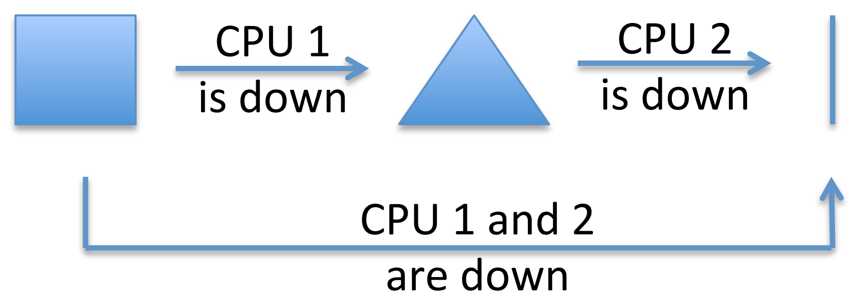
Bulk Interface



Offline

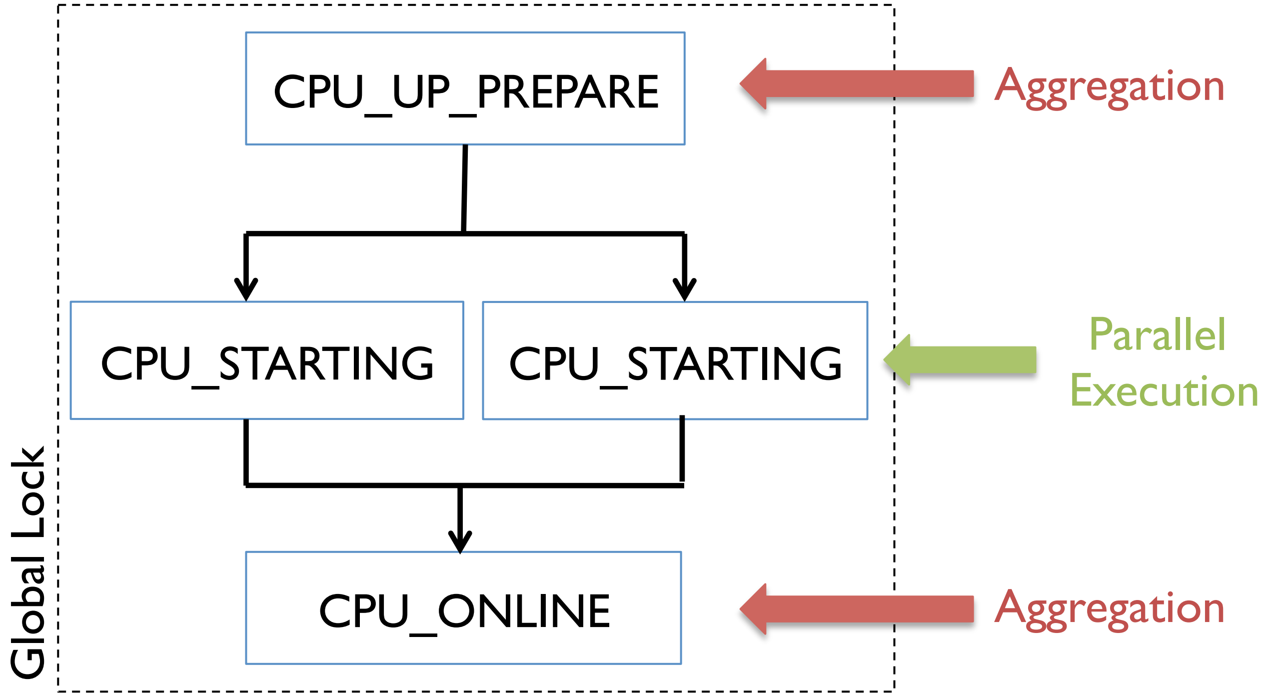
1. **Aggregation** - Avoid redundant execution

cpumask passed to down_prepare, dead and post_dead messages



2. **Parallel execution** – DYING message is handled in parallel by all cores in cpumask

Bulk Interface



Online

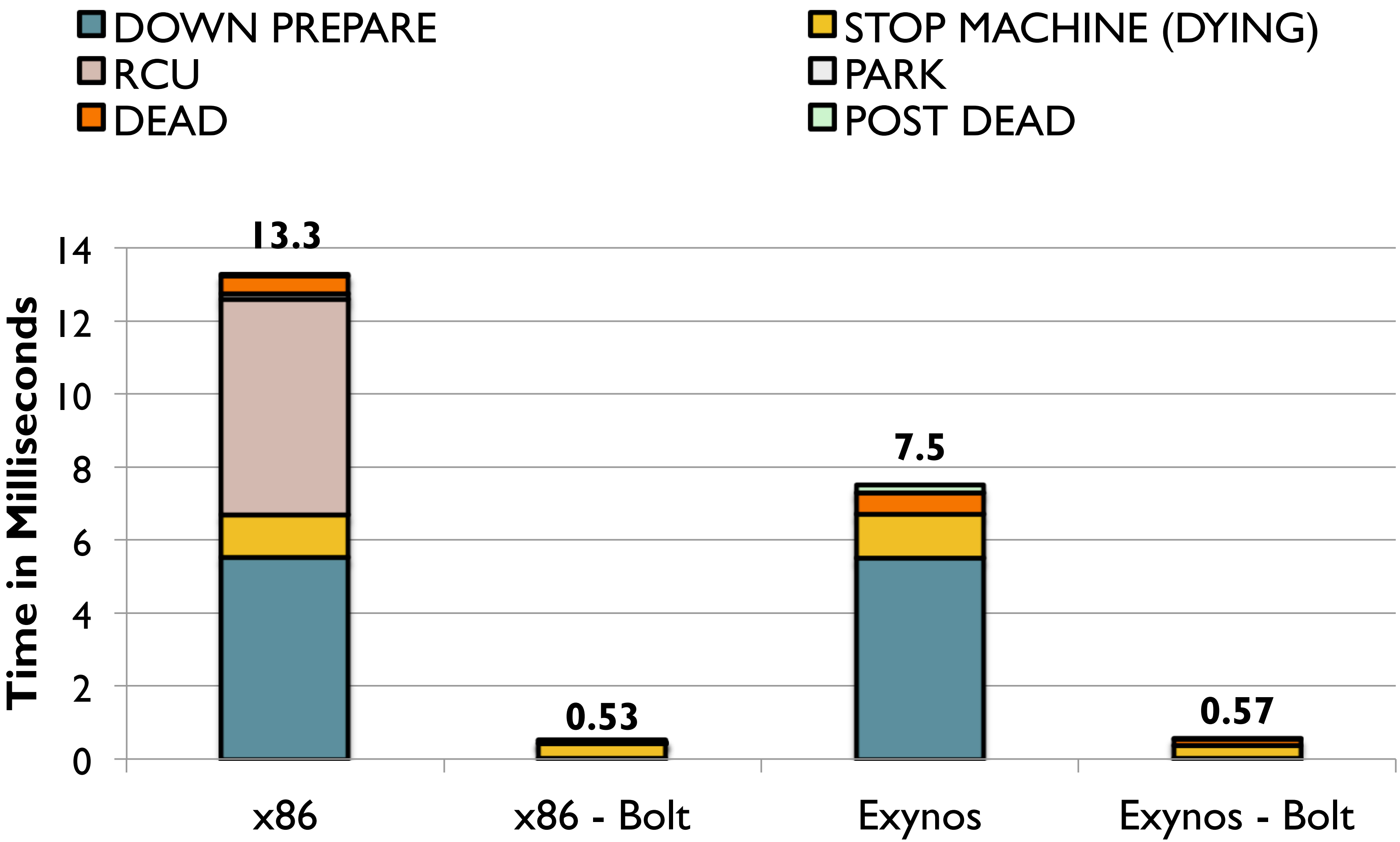
Outline

- Background on Hotplug
- Design of Bolt
- Evaluation Results

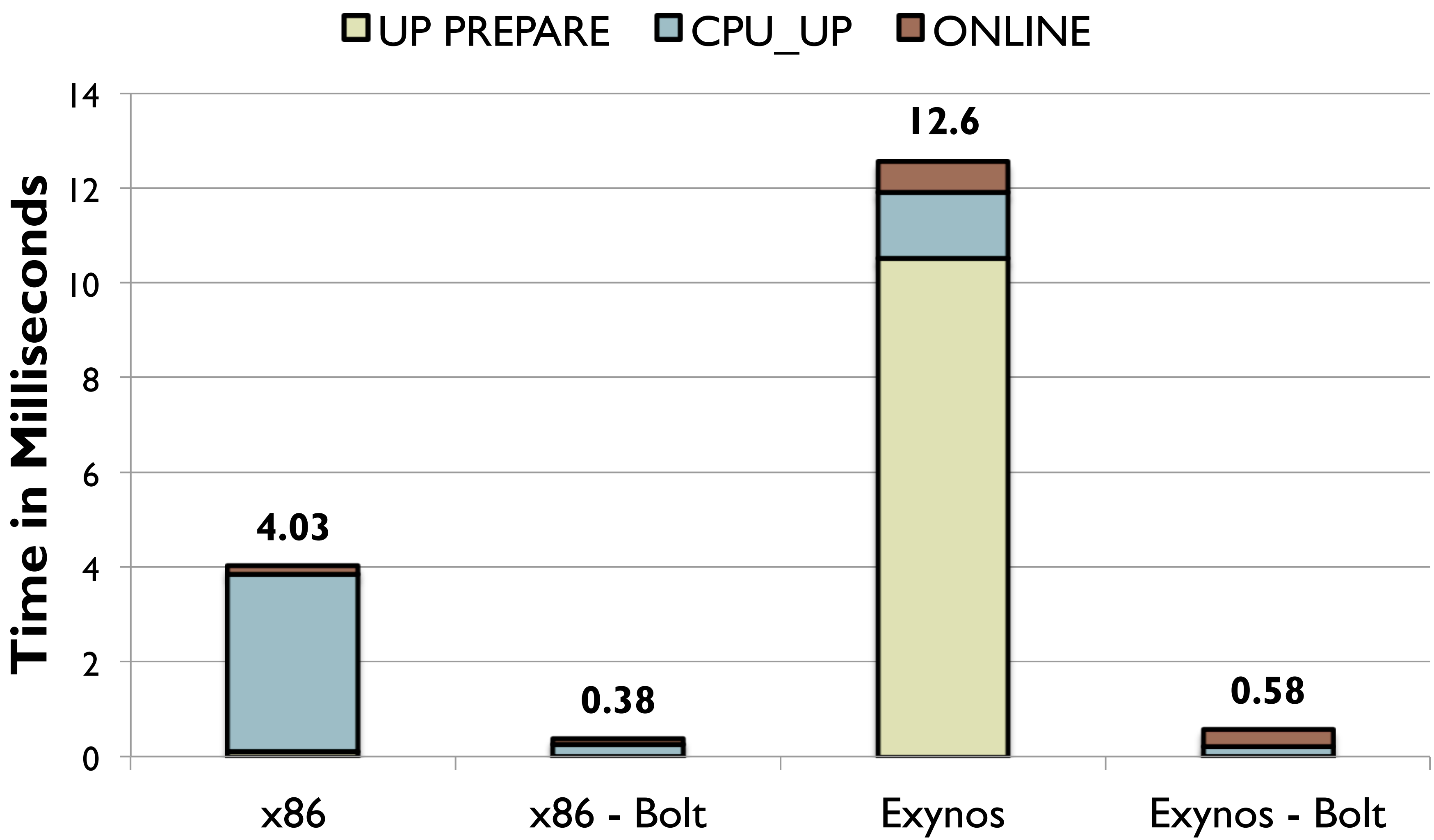
Experimental Platform

- Two platforms
  – x86: 4-core Intel i5-2500K (Sandybridge)
  – ARM: big.LITTLE Exynos 5410 (Odroid development board)
- Experiments: Latency measured for Hotplug operations
  – Done in an idle system at the highest processor frequency

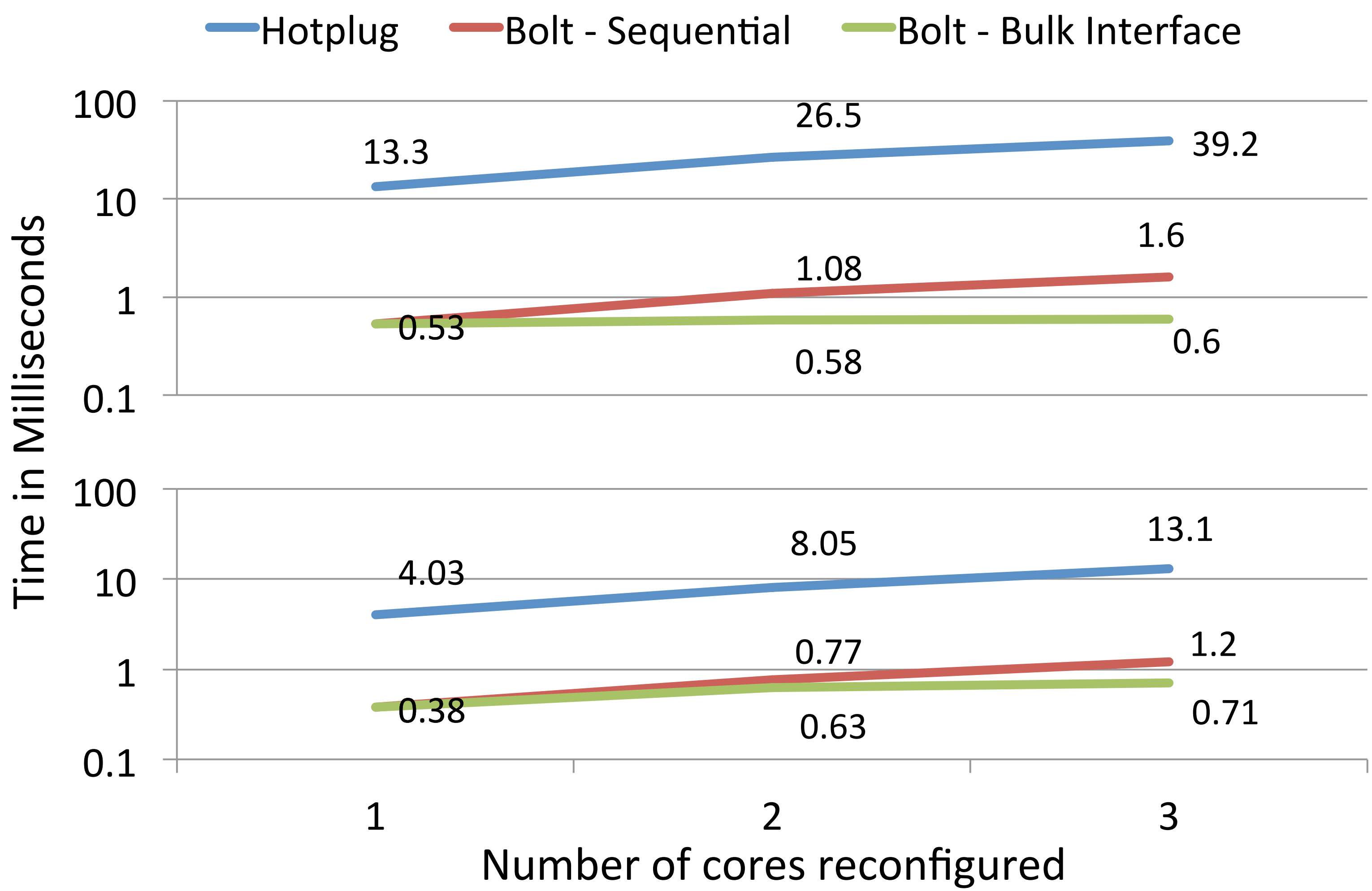
[CPU Offline] Bolt vs. Hotplug



[CPU Online] Bolt vs. Hotplug



Bulk Interface (x86)



Conclusion

- Hotplug is currently used as a reconfiguration mechanism
- Bolt reduces reconfiguration latency by separating critical from non-critical operations
- Bolt's principles can be applied to any reconfiguration mechanism

Questions