15-410

"...Two heads are better than one..."

Project 4 - "SMP_4"
Apr. 23, 2012

Dave Eckhardt

Acknowledgments

Leadership

- Jonah Sherman, Ryan Pearl, Garth Gibson

Base-code upgrades

- Ryan Pearl, Michael Sullivan, Andrew Bresticker

Feasibility-study implementations

- Paul Dagnelie, Alex Crichton

Handout reader

- Caroline Buckey

Knee-surgery recipient

- Eric Faust

Synchronization

P3extra reminder

- If your P3 isn't done, P3extra is *not optional!*
  - Filling out the form incorrectly doesn't make it optional

Deadline reminders

- *Please verify* that your group's p3extra xor p4 directory has been created
- P3extra deadline != P4 deadline
  - This is for real, not some oversight
- Don't forget about the book report...
  - Hand-in directories have been created

Outline

P4

- Symmetric multiprocessing!

The basic idea

- "A picture is worth a thousand words!"

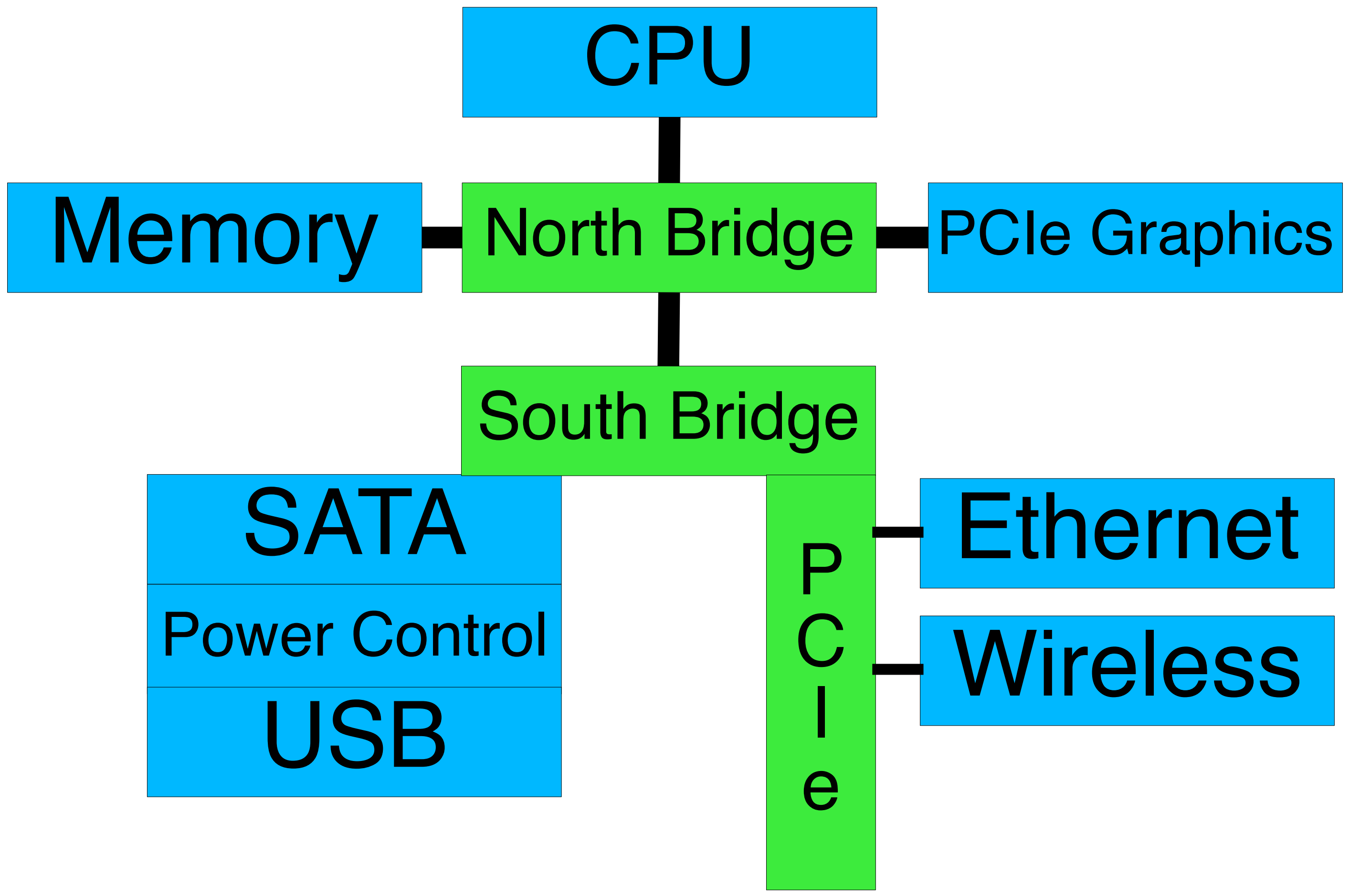
What you get from us

What you need to do

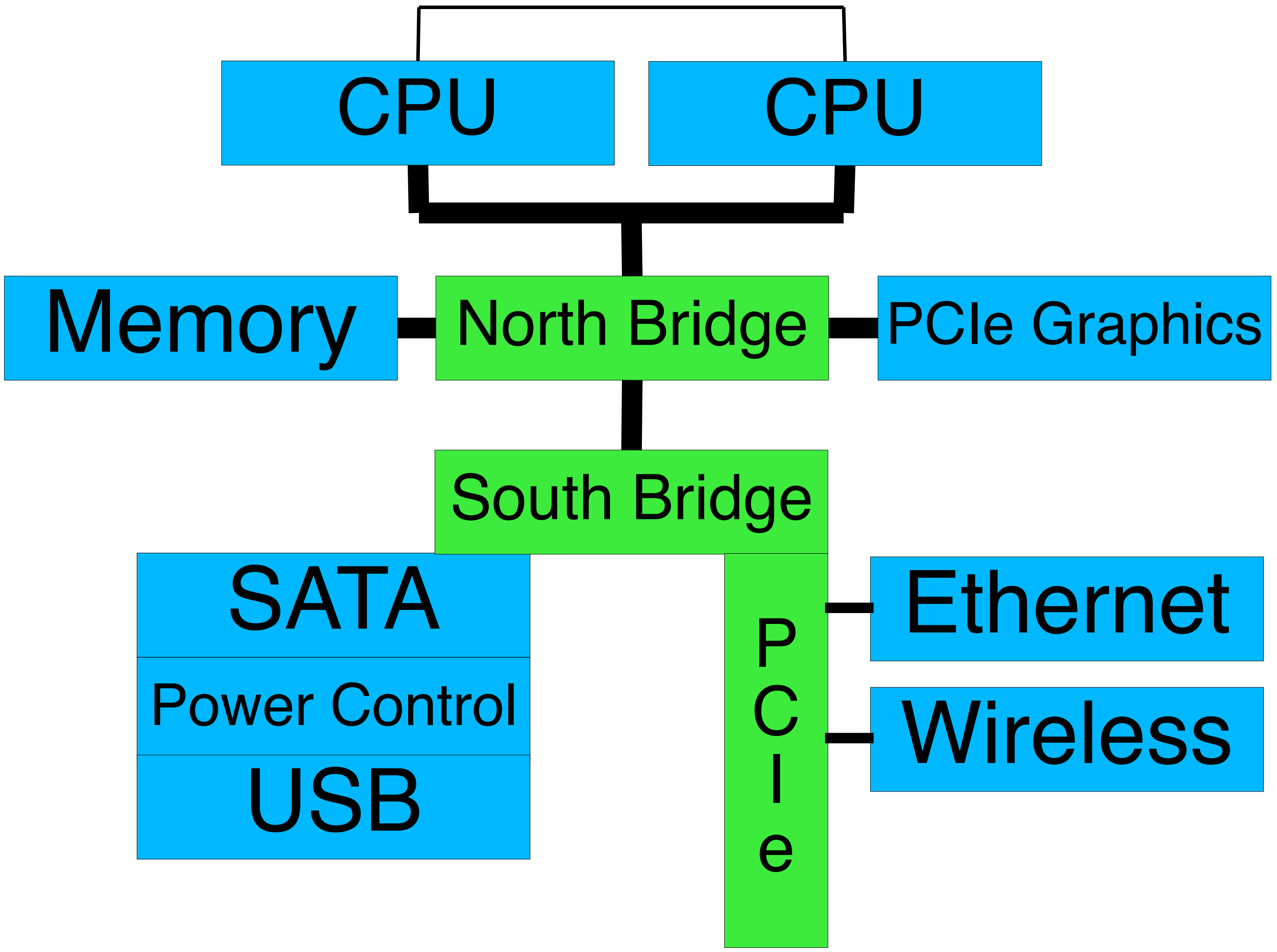
Considerations/strategy

Grading hints

The Basic Idea (Before)

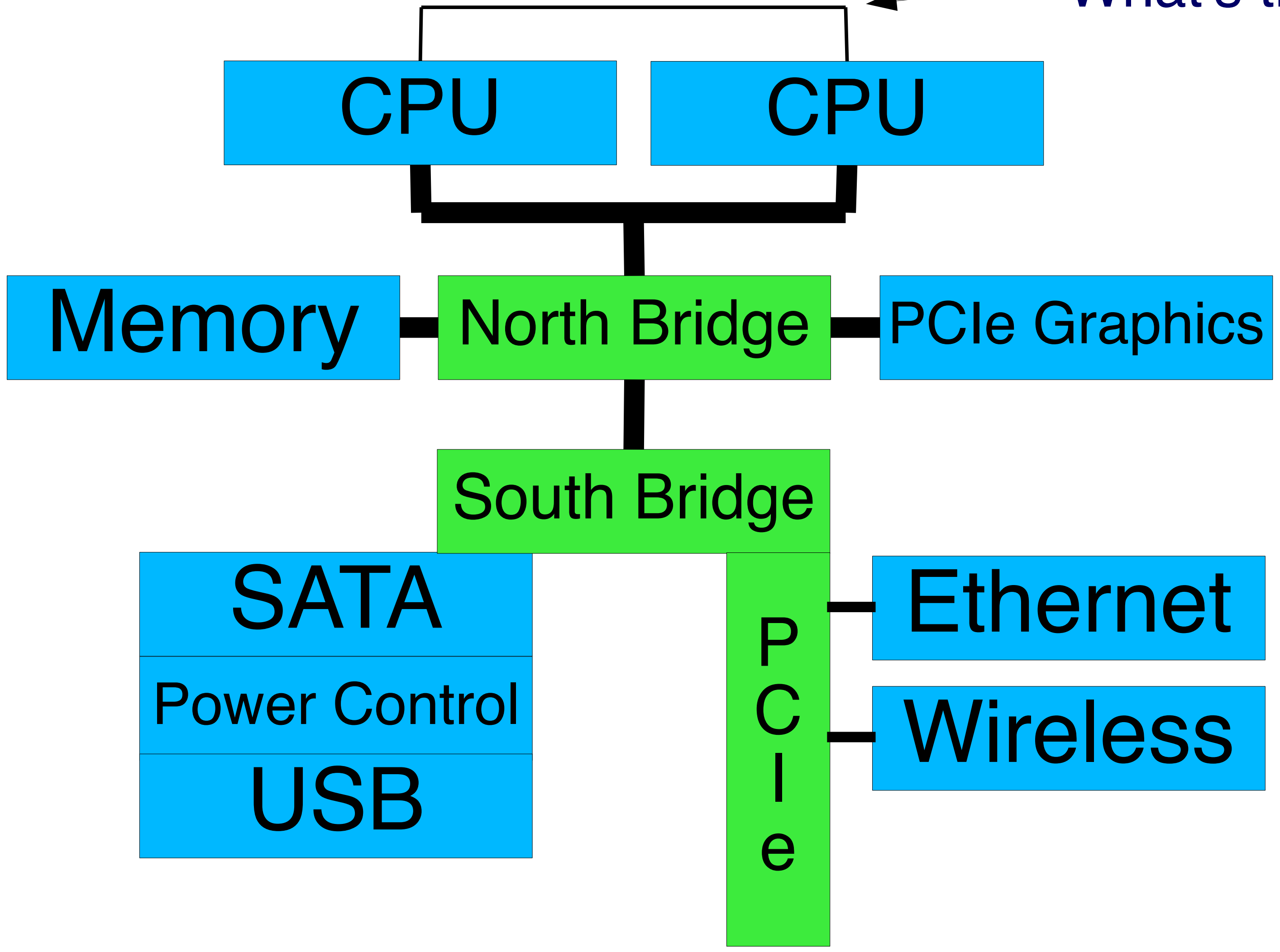


The Basic Idea (After)

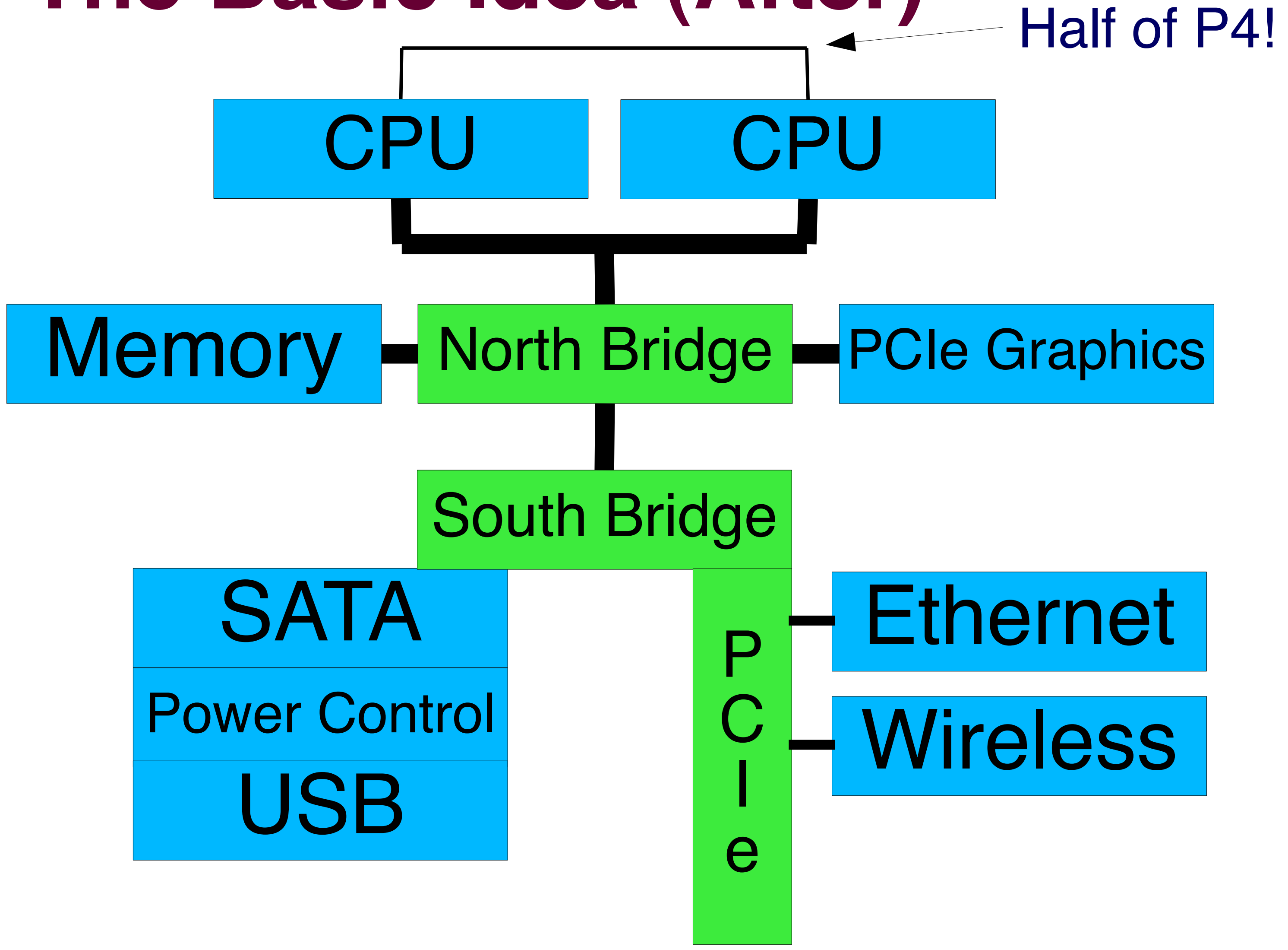


The Basic Idea (After)

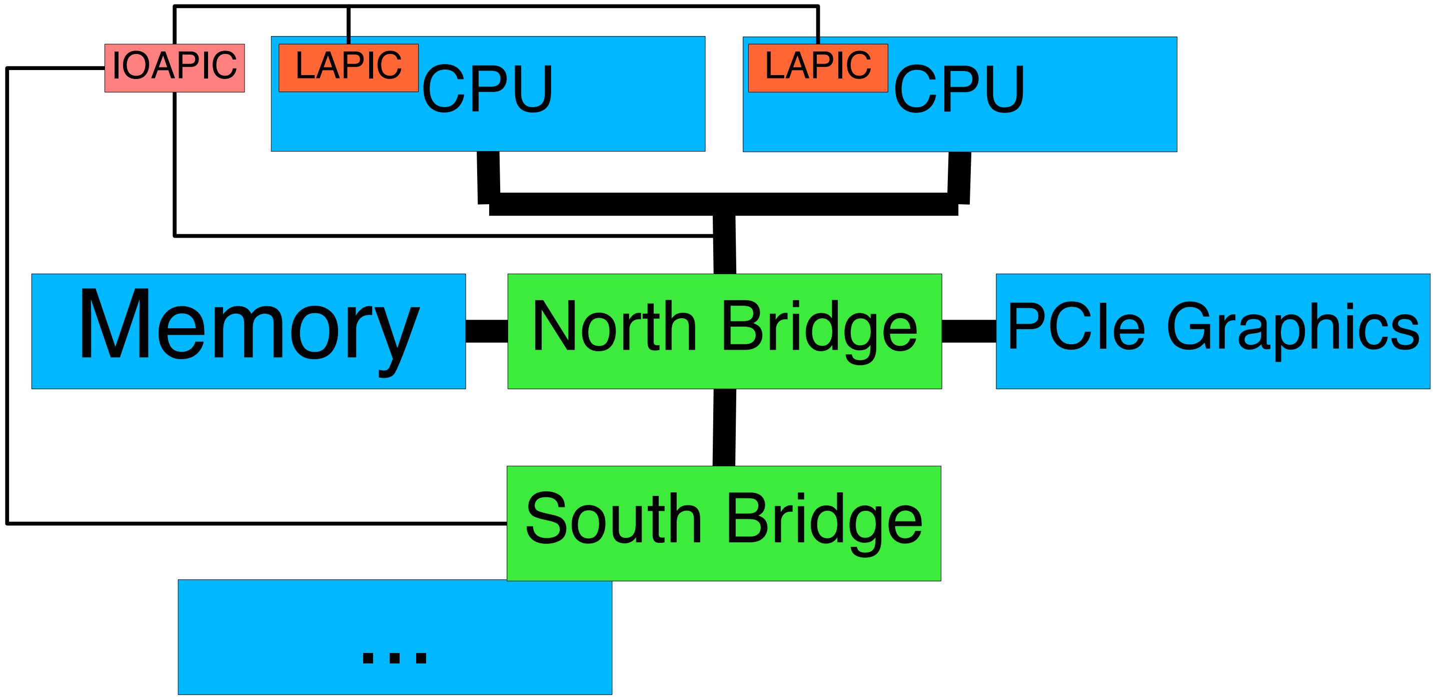
What's that thin line?



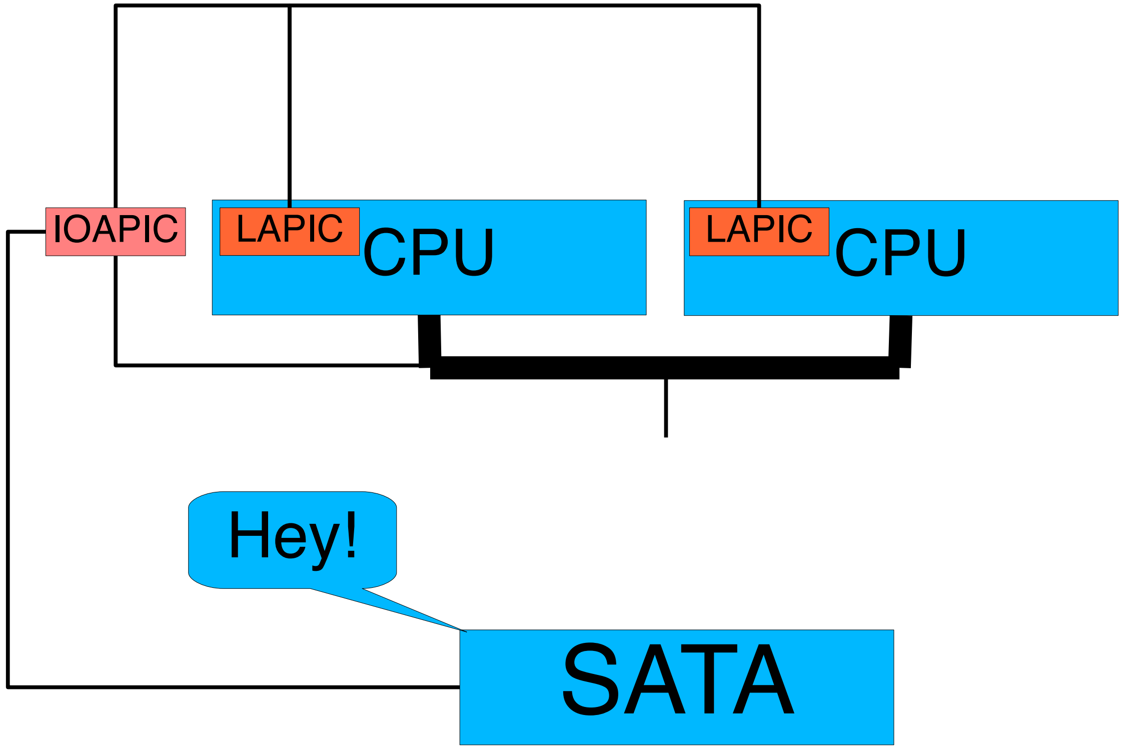
The Basic Idea (After)



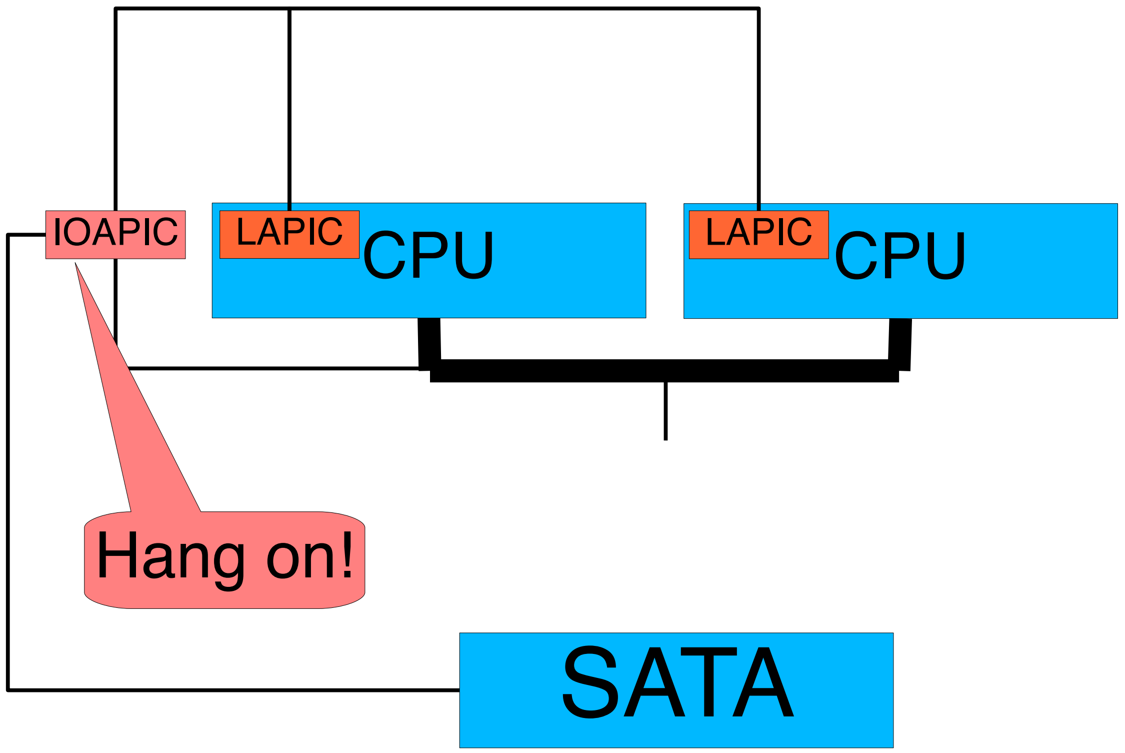
The Basic Idea (APICs!)



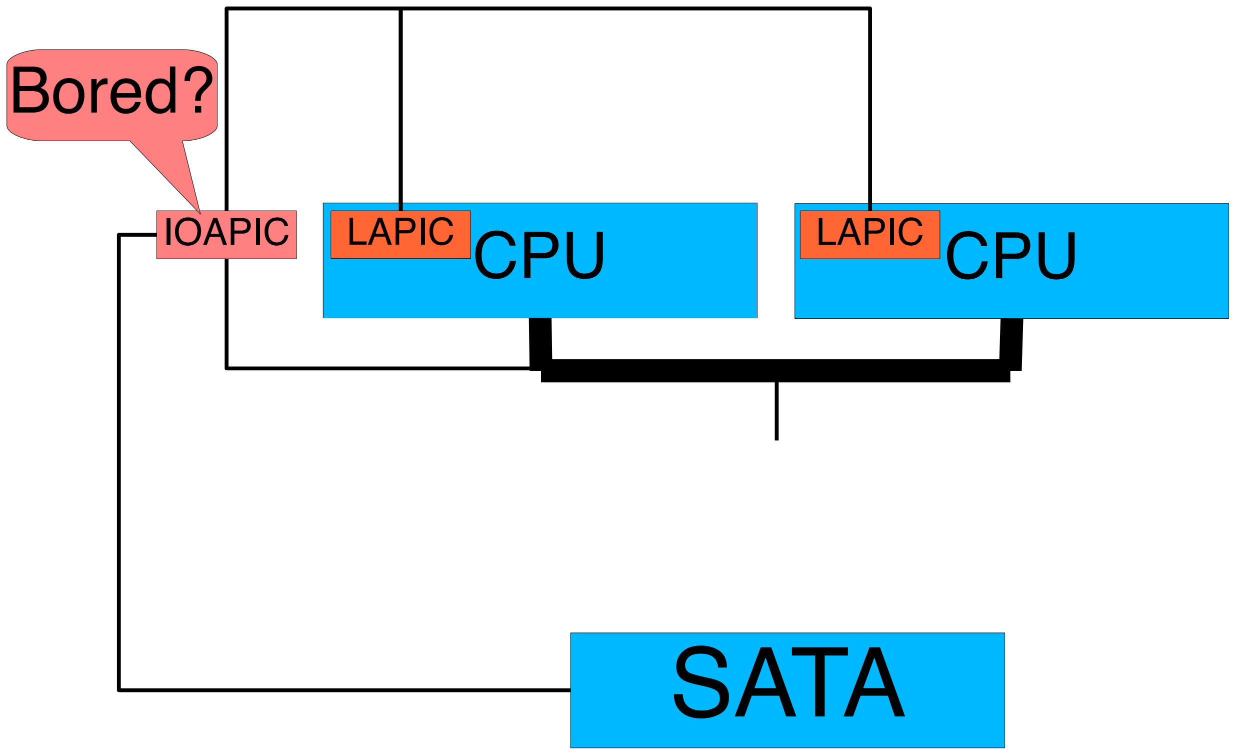
APICs At Work



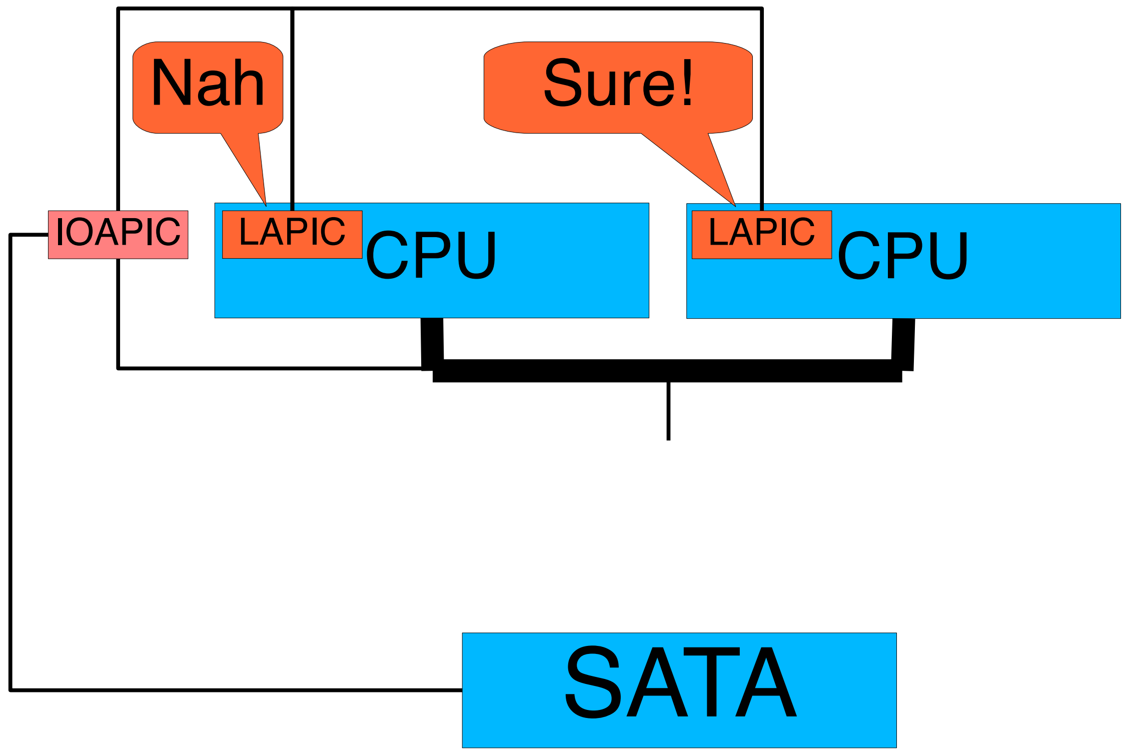
APICs At Work



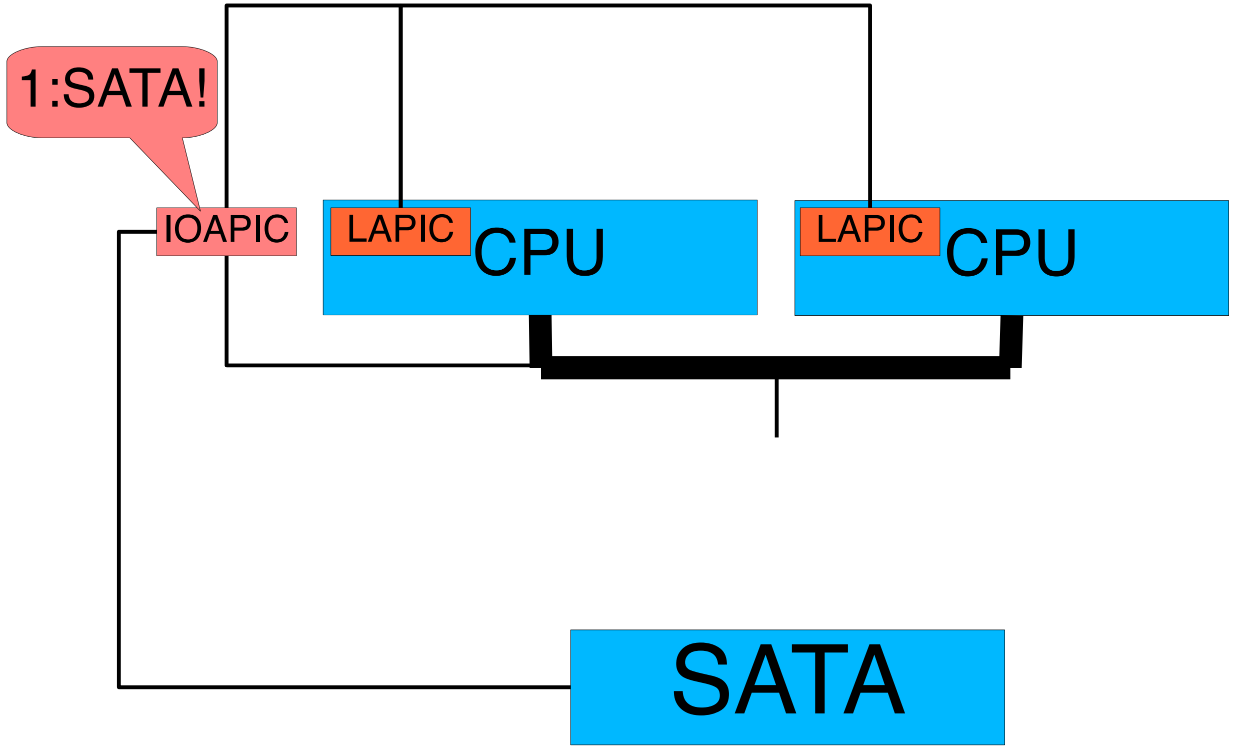
APICs At Work



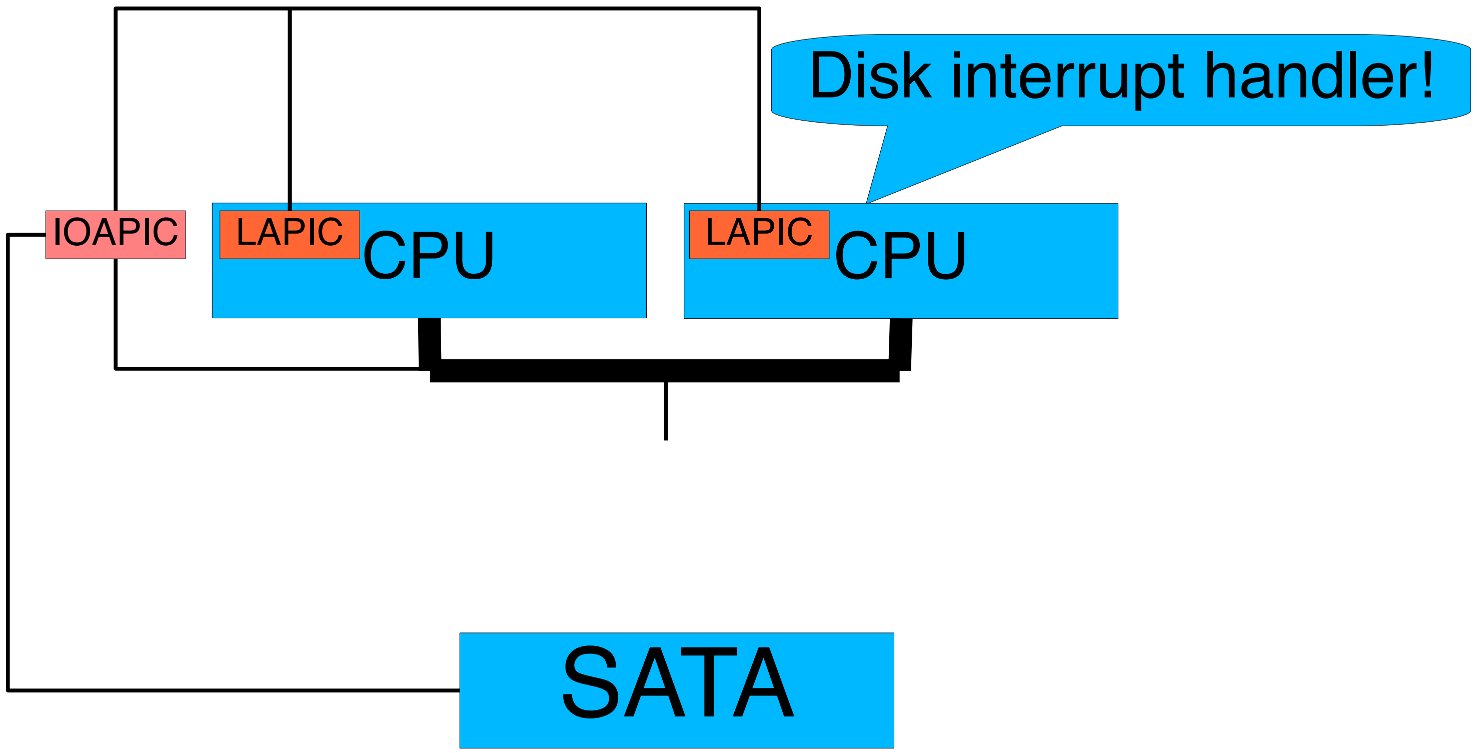
APICs At Work



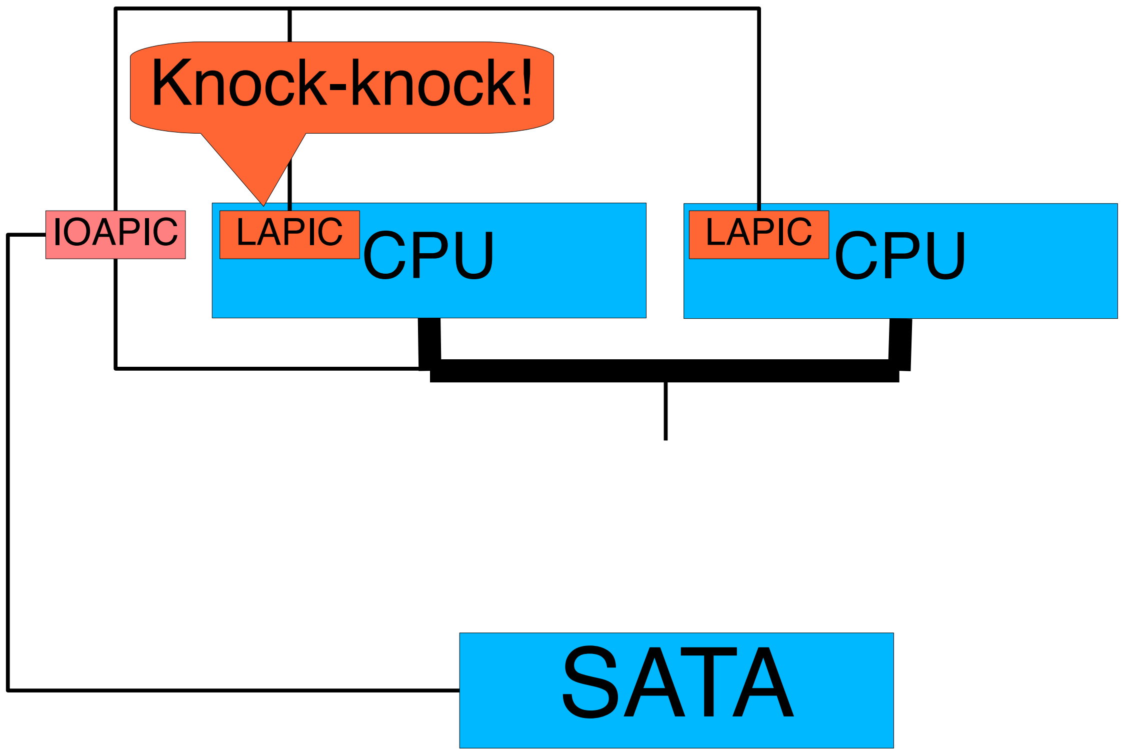
APICs At Work



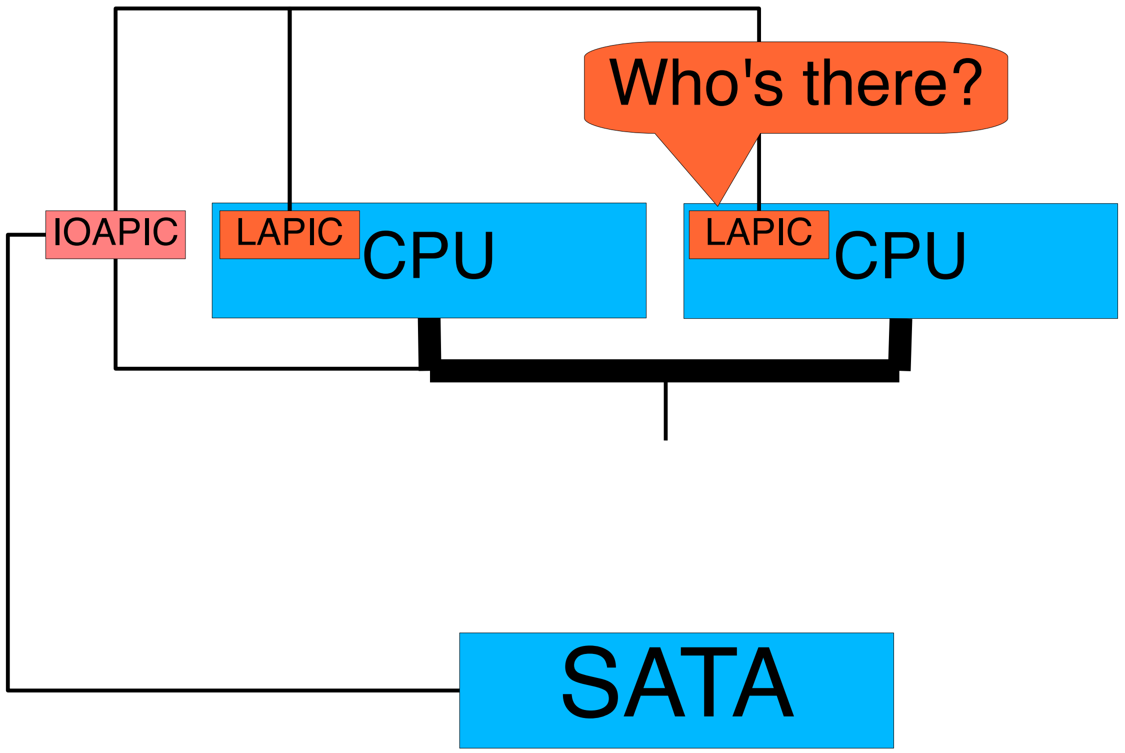
APICs At Work



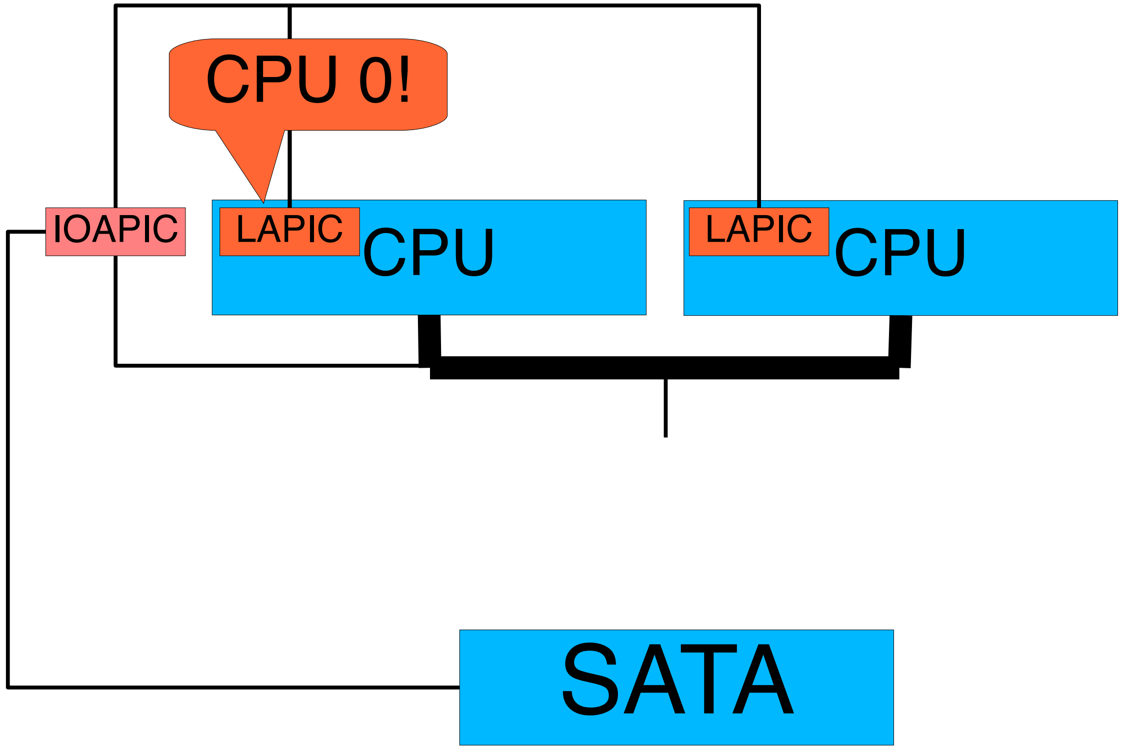
Inter-Processor Interrupts (IPIs)



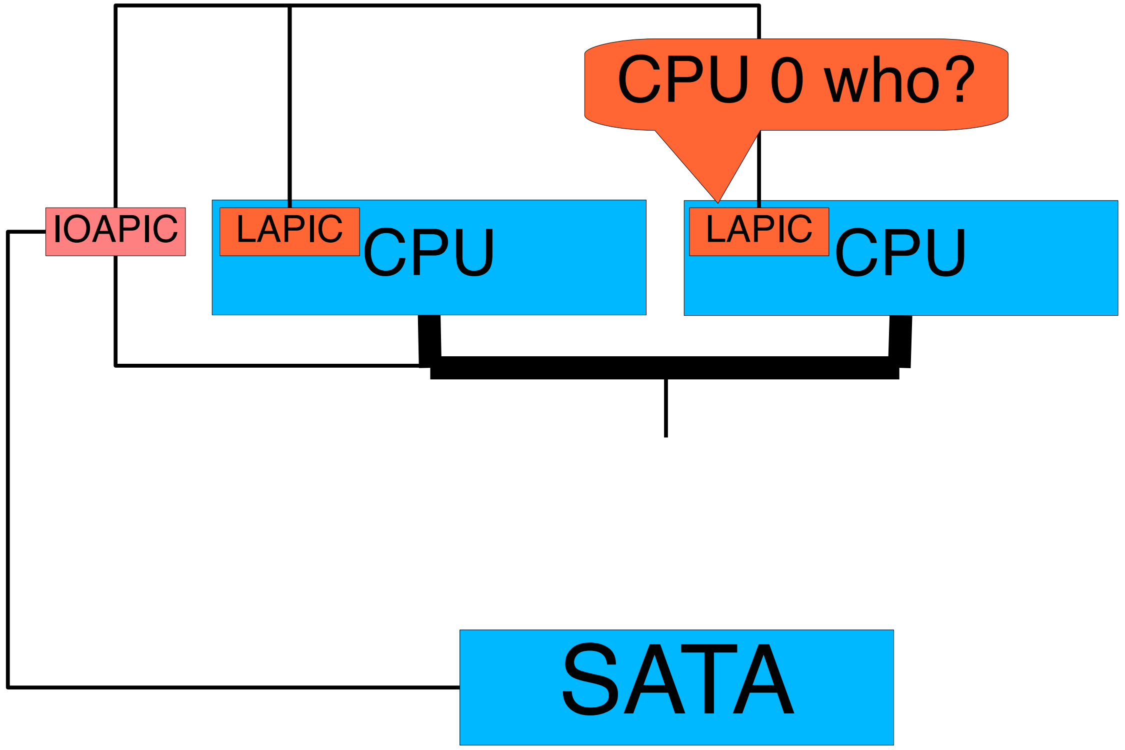
Inter-Processor Interrupts (IPIs)



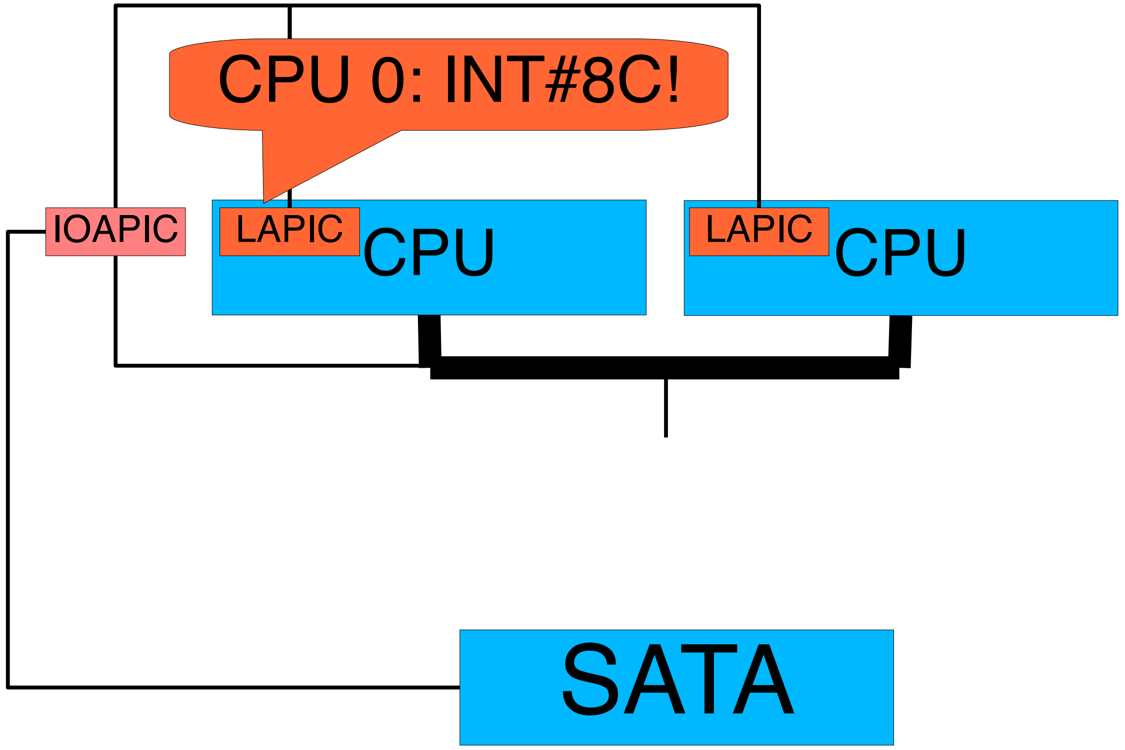
Inter-Processor Interrupts (IPIs)



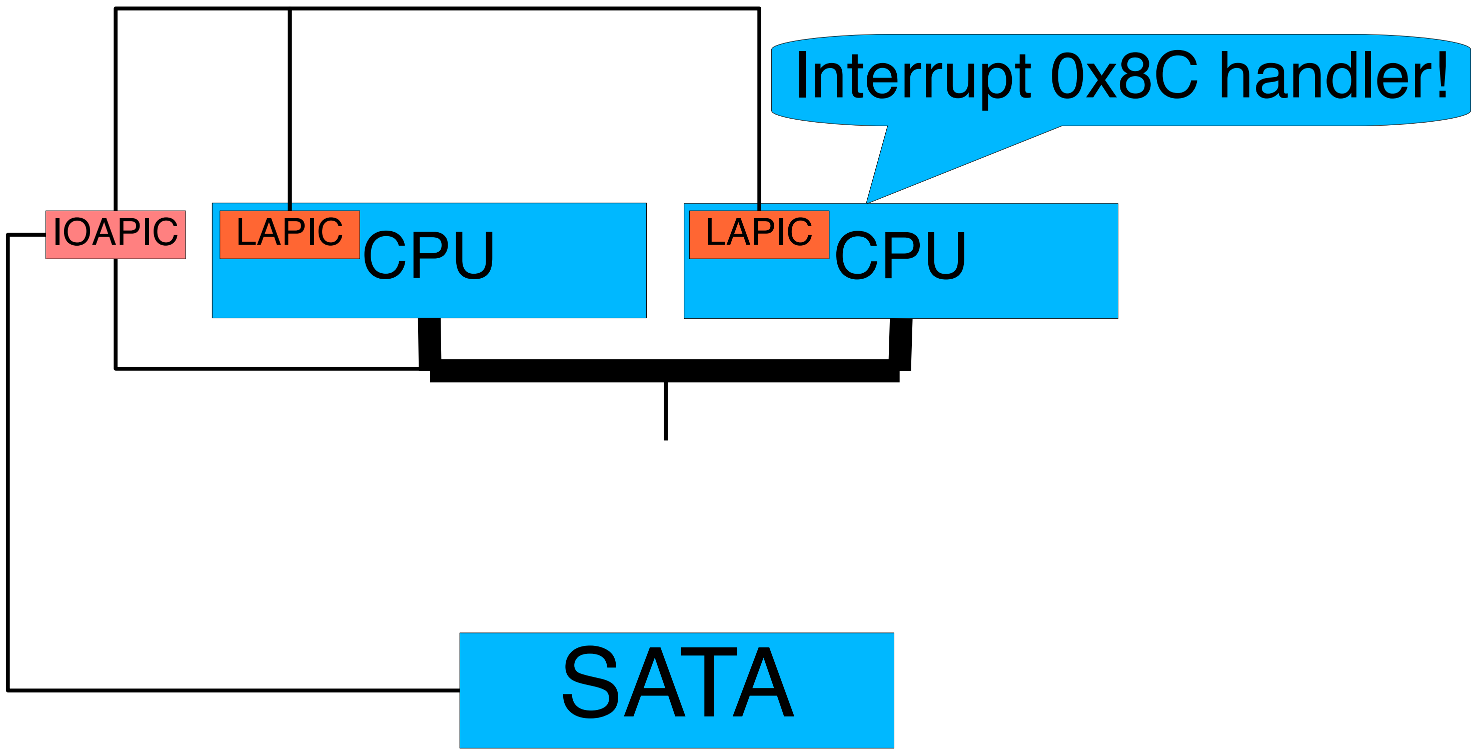
Inter-Processor Interrupts (IPIs)



Inter-Processor Interrupts (IPIs)



Inter-Processor Interrupts (IPIs)



APIC summary

I/O APIC

- One (or a few) per motherboard
- Tries to find somebody to serve each device interrupt
- Not really used in P4

Local APIC

- One per CPU
- Physically part of the CPU
- Accepts interrupts from I/O APIC and other Local APICs
- Swiss-Army knife of other functions
  - Programmable interrupt timer
  - Interrupt priority registers

APIC summary

I/O APIC

- One (or a few) per motherboard
- Tries to find somebody to serve each device interrupt
- Not really used in P4

Local APIC

- One per CPU
- Physically part of the CPU
- Accepts interrupts from I/O APIC and other Local APICs
- Swiss-Army knife of other functions
  - Programmable interrupt timer
  - Interrupt priority registers
  - Miscellaneous slow-ish registers
  - Tweezers, corkscrew

APIC summary

I/O APIC

- One (or a few) per motherboard
- Tries to find somebody to serve each device interrupt
- Not really used in P4

Local APIC

- One per CPU
- Physically part of the CPU
- Accepts interrupts from I/O APIC and other Local APICs
- Swiss-Army knife of other functions
  - Programmable interrupt timer - Used in P4
  - Interrupt priority registers - Not really used in P4
  - Miscellaneous slow-ish registers - Used by SMP base code
  - Tweezers, corkscrew - Not really used in P4

What You Get From Us

Handout

- Explains how the pieces should fit together
- Specific hints on many topics

libsmp.a

- mptable.c: parse Intel's “MP tables”
- apic.c: talk to Intel's local APICs, send IPIs
- smp.c, smp.h, smp_start.S: launch APs, misc.

Selected Intel documentation

- intel-mp.pdf – description of the Intel SMP environment
- intel-sys.pdf – detailed information about APICs

What You Need to Do

Design

- Some locking code “probably” needs work
- Scheduler “probably” needs work
  - Selected invariants have just varied
  - Need to have a serious chat with the locking code
- Remote TLBs are full of lies

Hacking

- Glue code to enable libsmp to do its job
- CPU startup code
- Locking
- Scheduler
- Run code without crashing
- Switch to APIC timers
- Run code without crashing

Considerations/Strategy

Design will be very important

- This is an opportunity to understand concurrency much better...
    - Be sure you start with a strong understanding (detailed review) of what your existing code does!
- Planning how to test code before you write it will be important
- You may gain a new level of respect for “the idle thread”

Hierarchy of partial solutions

- Some uni-processor code-base designs will be challenged by a multi-processor world
    - This happened to commercial kernels
    - “Temporary” workarounds exist (of varying quality), see handout

Approximate Grading

Weight	*Section*
5	Clean build
15	README (see handout)
5	Launch APs
20	Scheduler work
15	Locking work
10	VM work
20	Tests
10	Miscellaneous

Note: initial approximation!

Suggestions

Read the handout carefully

Carefully review existing locking code

Carefully review locking *clients*

- Is everybody using the right kind of lock?

Carefully review scheduler

Attack plan!

- Try to have some fun while you're at it

Outline

P4

- Symmetric multiprocessing!

The basic idea

- "A picture is worth a thousand words!"

What you get from us

What you need to do

Considerations/strategy

Grading hints