# Introduction

- Reading
  - Papers
- Reminder: Project due Sept 29th (see ext on web page)
  - Submit via email to hollings (mime attached tar file)
  - Sample data output now on web page

# Cache Coherency (write through)

- Read-only data cached
- Writeable values can be cached by one processor
  - a processor needs to gain write access
    - must force invalidation of other cached copies
  - all writes go back to main memory
  - reads can be served from cache for processor with write access
- Performance
  - good for
    - updates and reads by same processor
  - bad for
    - multiple updates by the same processor (many bus writes)

# How to Manage Caches

- Snooping
  - each cache controller watches bus for “interesting” info
  - may result in cache lines being invalidated if write seen
    - i.e. a write through cache
  - limited by speed of cache controllers to watch the bus
    - must see everything to maintain correctness
- Directories
  - memory stores information about cached copies
  - does not require each cache controller to snoop
  - permits more scaleable interconnect networks

# Sun Fireplane

- Two level coherency protocol
  - Snooping within 24 processors (snooping domain)
  - Point-to-point between snooping domains
- Address bus (snooping)
  - Bi-directional address repeater ASICs
  - Connected in tree structure
- Memory tags (point-to-point)
  - Stored in 8 byte ECC tags for every 64 bytes of memory
  - States
    - Global Modified, Global Shared, Global Invalid

# Directory Based Cache Controllers

- Requires additional circuits to maintain directories
- directories must be updated when a processors
  - starts caching a value
  - stops caching a value
  - changes from read to write caching (or back)
- each cache line has a directory entry
  - can use sparse schemes that only have entries for actively cached items
- can have several memory controllers in a machine
  - each manages a region of physical memory
  - bit vectors (one bit per processor)
  - addresses (several $\log_2 n$ entries)

# Representing Directories

- bit vectors
  - one bit per processor
  - uses lots of space for a large machine
  - permits each processor to cache a value
- addresses
  - several entries for PE id (each entry is $\log_2 n$ bits)
  - what happens if a processor wishes to cache, and all entries are full?
    - use a linked list of directories (SCI uses this approach)
    - use a "wildcard" and force a broadcast to invalidate

# Stanford Dash

- Structure
  – collection of bus based multi-processors
  – interconnect network and cache controller connect nodes
- Cache System
  – snoopy protocol within a single SMP node
  – directory based cache controller between nodes
    - misses on local cluster go to home cluster of memory "owner"
    - owner may have current copy or could be cached on another cluster
- Processors
  – 4 MIPS R3000 (33 Mhz) per node
- Interconnect
  – 2 dimensional mesh

# Stanford Dash (cont.)

- Performance
  - level 0 cache (1 clock)
  - remote cluster load (132 clocks)
- Later Directions
  - FLASH
  - use a full micro-processor for the cache controller
    - permits customization of cache protocols
    - makes the hardware simpler

# SGI Origin Servers

- Commercialization of Stanford DASH
  - SMP nodes
  - directory based cache controller
- Changes
  - processors are R10000 (250 Mhz)
  - only 2 nodes per bus
    - slightly cheaper bus than DASH
    - faster processors require more bus bandwidth
  - interconnection network
    - hypercube (to 32 nodes)
    - re-configurable routers beyond
  - Directory only based (I.e. no snoopy within the nodes)

# SGI Origin Structure

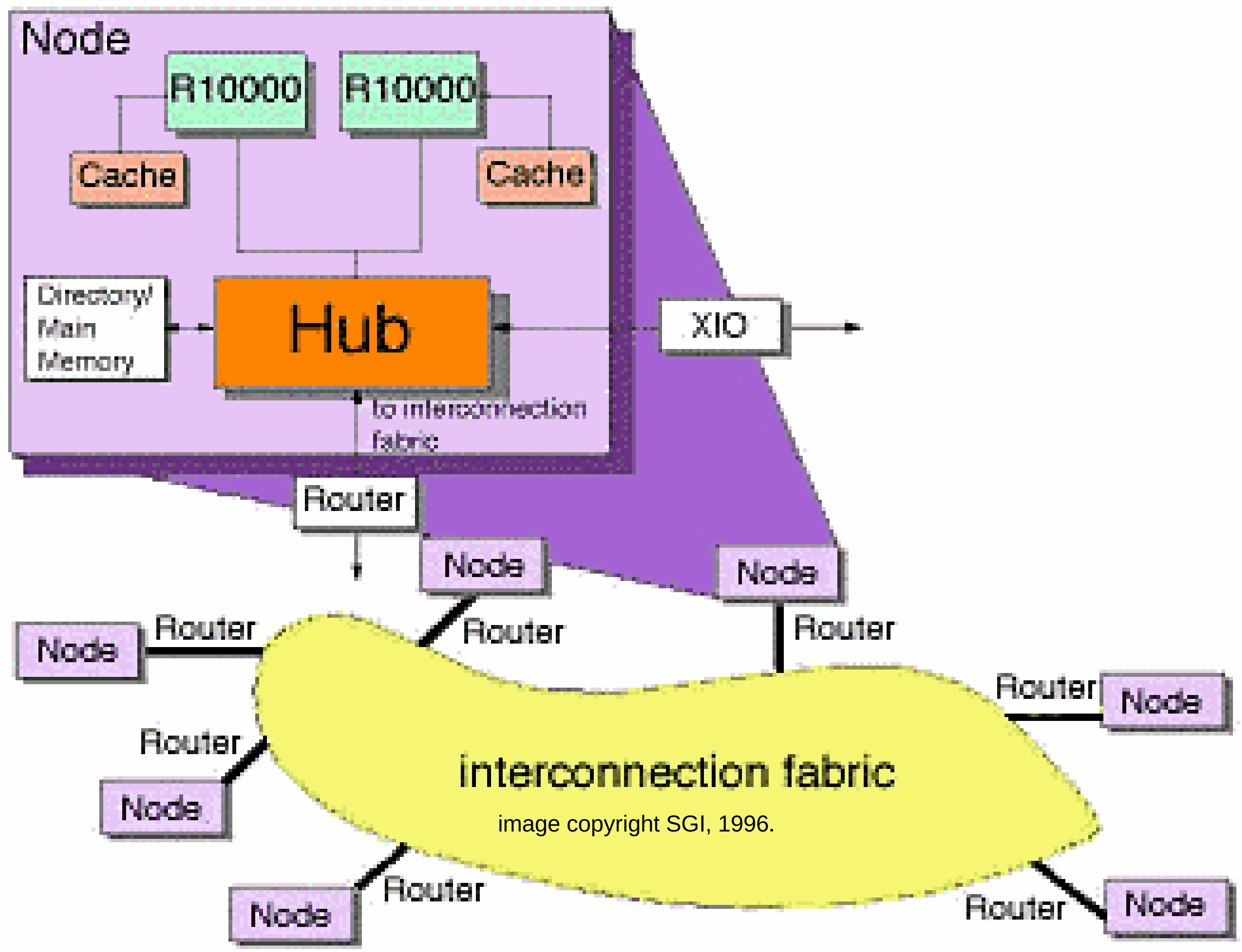


image copyright SGI, 1996.