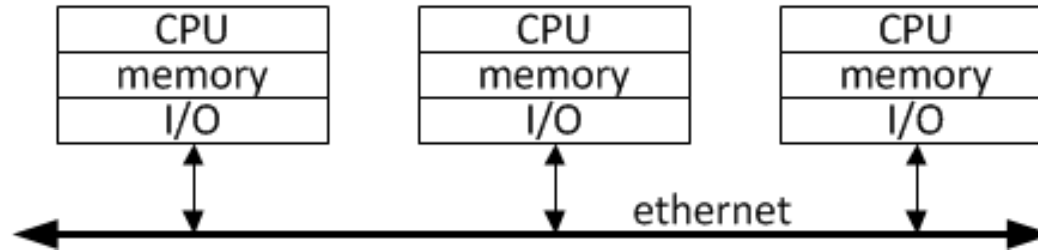


Multiprocessors

Loosely coupled [**Multi-computer**]



- each CPU has its own memory, I/O facilities and OS
- CPUs **DO NOT** share physical memory
- [IITAC Cluster](#) [in Lloyd building]

346 x IBM e326 compute node each with 2 x 2.4GHz 64bit AMD Opteron 250 CPUs, 4GB RAM, ...
80GB SATA scratch disk, 2 x Broadcom BCM5704 Gbit Ethernet and PCI-X 10Gb InfiniBand connect
2 x Voltaire 288 port InfiniBand switches + 512 port Gbit switch

Debian [**Linux**]

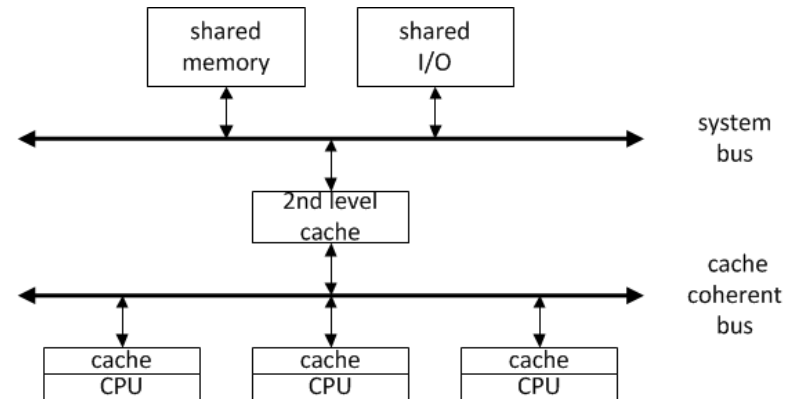
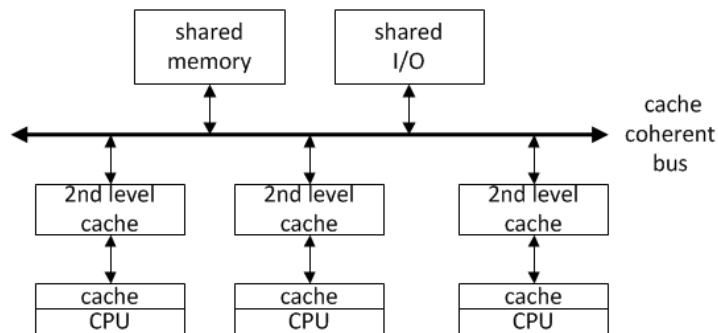
Theoretical peak performance 3.4TFlops [**Linpac 2.7Tflops**]

ranked 345th most powerful supercomputer [**in 2006**]

- a distributed system, cloud, ...

Multiprocessors

Tightly coupled [Multiprocessor]

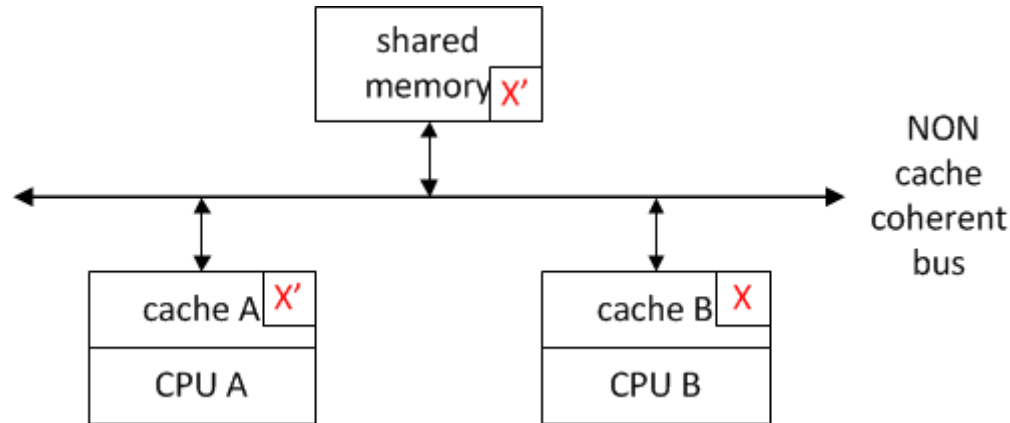


- CPUs physically share memory and I/O
- inter-processor communicate via shared memory
- symmetric multiprocessing possible [i.e. single shared copy of operating system]
- critical sections protected by locks/semaphores
- processes can easily migrate between CPUs

Multiprocessors

Multiprocessor Cache Coherency

- what's the problem? must guarantee that a CPU always reads the most up to date value of a *memory location*



- 1) CPU A reads location X, copy of X stored in cache A
- 2) CPU B reads location X, copy of X stored in cache B
- 3) CPU A writes to location X, X in cache and main memory updated [**write-through**]
- 4) CPU B reads X, BUT gets out of date [**stale**] copy from its cache B

Multiprocessors

Multiprocessor Cache Coherency...

- solutions based on *bus watching* or *snoopy* caches [yet again]
- *snoopy* caches watch all bus transactions and update their internal state according to a pre-determined cache coherency protocol
- cache coherency protocols given names eg. Dragon, **Firefly**, **MESI**, Berkeley, Illinois, **Write-Once** and **Write-Through** [will discuss protocols in bold]

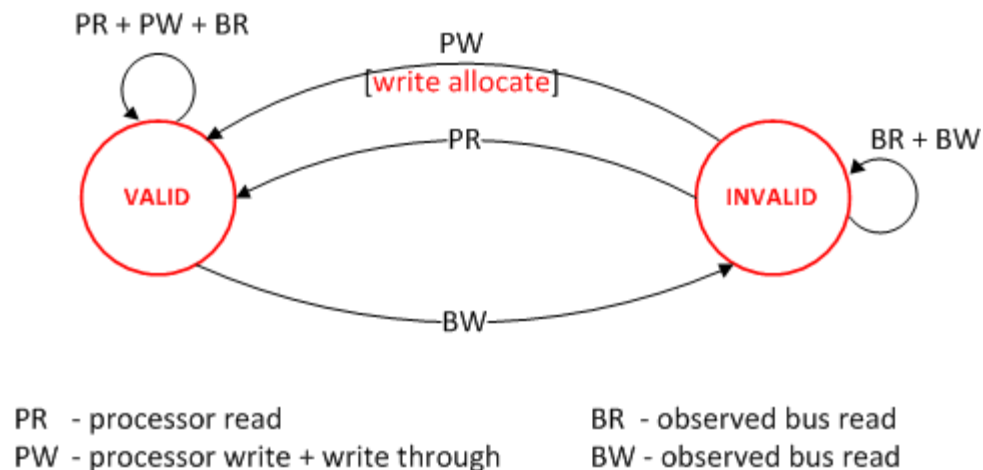
Multiprocessors

Write-Through Protocol

- key ideas
 - uses write-through caches
 - when a cache observes a bus write to a memory location it has a copy of, it simply invalidates the cache line [a write-invalidate protocol]
 - the next time the CPU accesses the same memory location/cache line, the data will be fetched from memory [hence getting the most up to date value]

Multiprocessors

Write-Through State Transition Diagram



- each cache line is in either the **VALID** or **INVALID** state [if an address NOT in the cache, it is captured by the **INVALID** state]
- $PW \equiv$ processor write and write through to memory
- bus traffic $\approx \Sigma$ writes [**writes $\approx 20\%$ of memory accesses**]
- straightforward to implement and effective for small scale parallelism [**< 6 CPUs**]

Multiprocessors

Write-Once Protocol

- uses write-back caches [to reduce bus traffic]

BUT still uses a write-invalidate protocol

- each cache line can be in one of 4 states:
 - **INVALID**
 - **VALID**: cache line in one or more caches, cache copies \equiv memory
 - **RESERVED**: cache line in one cache ONLY, cache copy \equiv memory
 - **DIRTY**: cache line in one cache ONLY and it is the ONLY up to date copy [memory copy out of date / stale]

Multiprocessors

Write-Once Protocol...

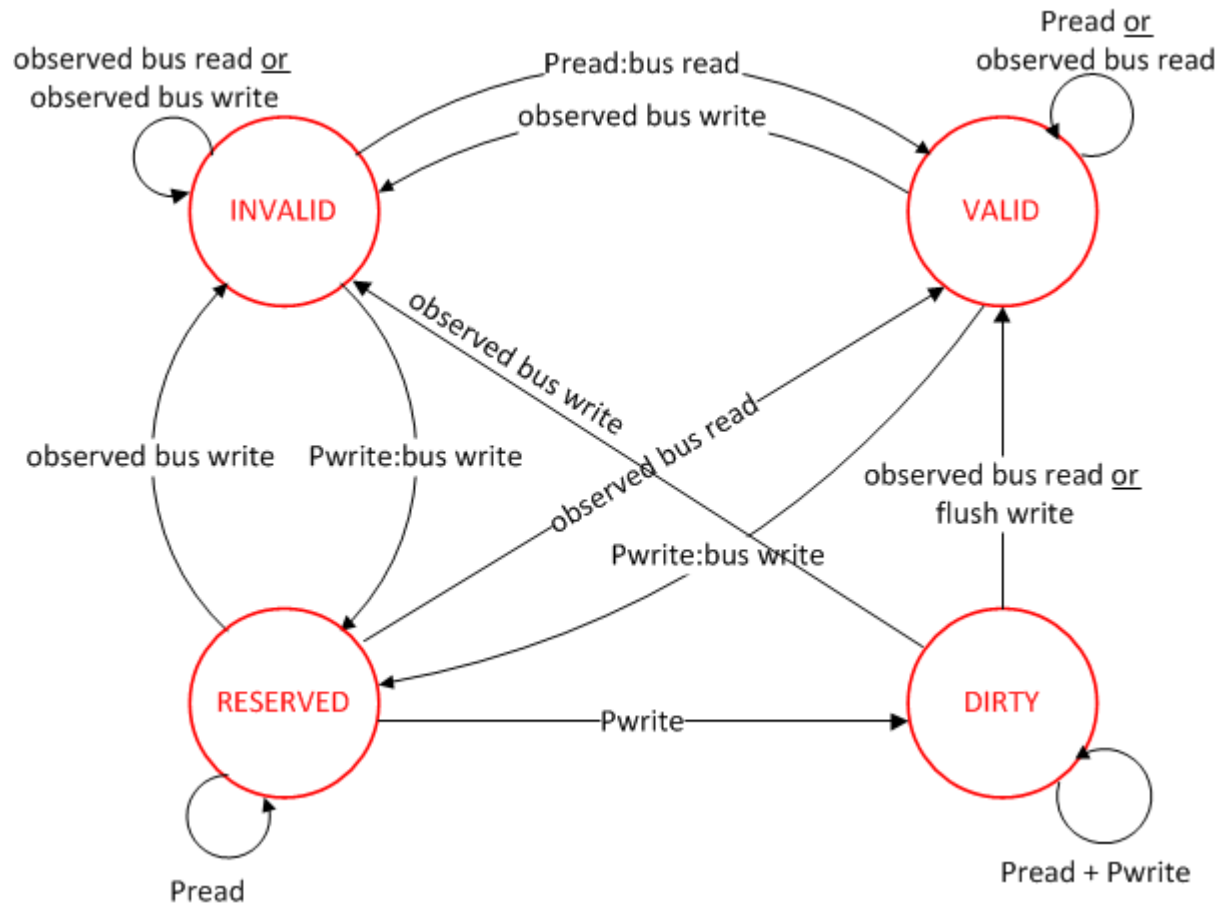
- key ideas
 - write-through to VALID cache lines
 - write back (local write) to RESERVED/DIRTY cache lines [as cache line in one cache ONLY]
- when a memory location is read initially, it enters the cache in the VALID state
- a cache line in the VALID state changes to the RESERVED state when written for the first time [hence the name write-once]
- a write which causes a transition to the RESERVED state is written through to memory so that all other caches can observe the transaction and invalidate their copies if present [cache line now in one cache ONLY]

Write-Once Protocol...

- subsequent writes to a RESERVED cache line will use write-back cycles and the cache line marked as DIRTY as it is now the only up to date copy in the system [**memory copy out of date**]
- caches must monitor bus for any reads or writes to its RESERVED and DIRTY cache lines
 - if a cache observes a read from one of its RESERVED cache lines, it simply changes its state to VALID
 - if a cache observes a read from one of its DIRTY cache lines, the cache must intervene [**intervention cycle**] and supply the data onto the bus to the requesting cache, the data is also written to memory and the state of both cache lines are changed to VALID
 - NB: behaviour on an observed write [**DIRTY => INVALID**]

Multiprocessors

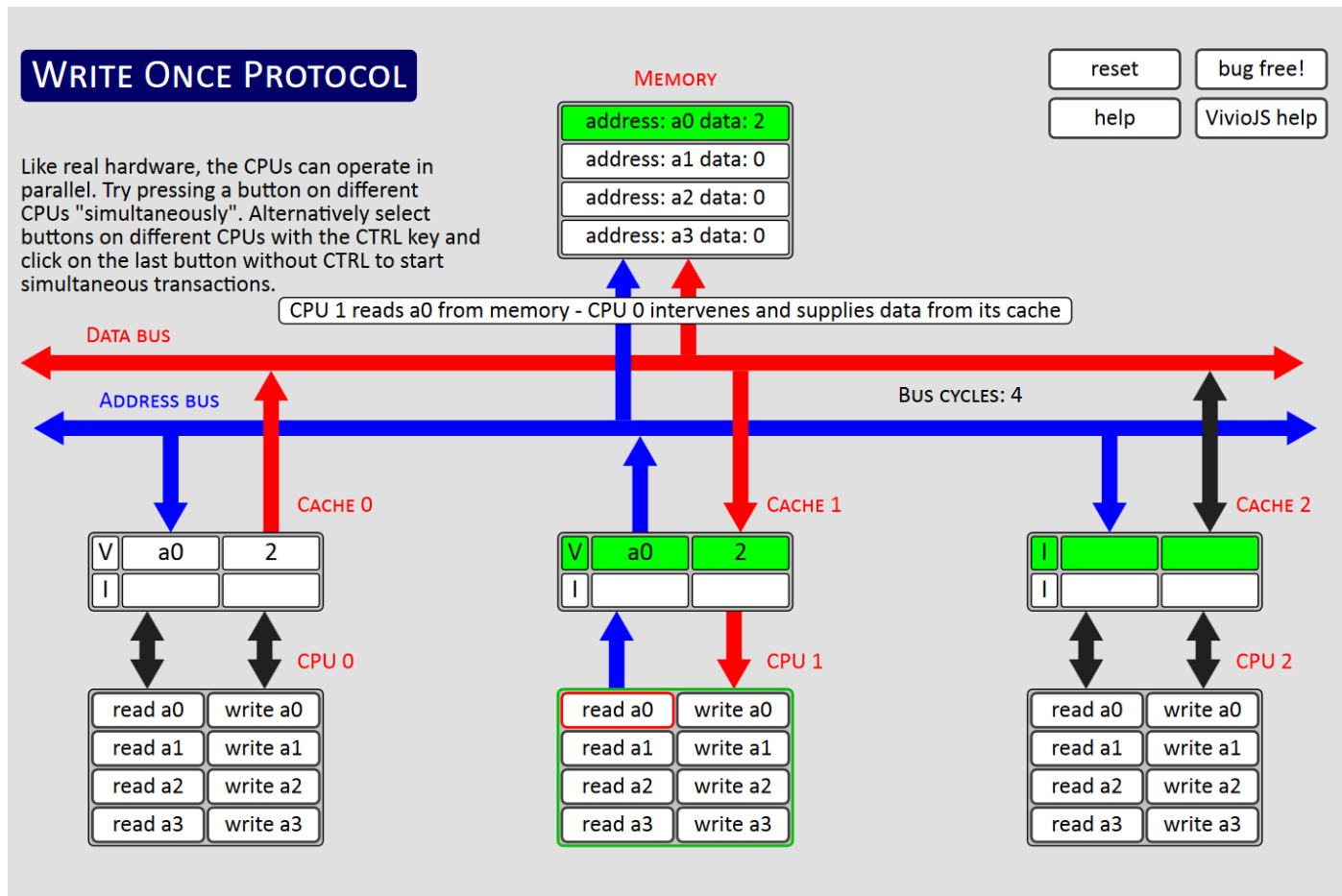
Write-Once State Transition Diagram



Multiprocessors

Write-Once Protocol...

- try the Vivio cache coherency animation



Multiprocessors

Firefly Protocol

- used in the experimental Firefly DEC workstation

DEC \equiv Digital Equipment Corporation

- each cache line can be in one of 4 states:
 - \sim Shared & \sim Dirty [Exclusive & Clean]
 - \sim Shared & Dirty [Exclusive & Dirty]
 - Shared & \sim Dirty [Shared & Clean]
 - Shared & Dirty [Shared & Dirty]
- NB: there is NO INVALID state

Multiprocessors

Firefly Protocol...

- key ideas
 - a cache knows if its cache lines are shared with other caches [**may not actually be shared but that is OK**]
 - when a cache line is read into the cache the other caches will assert a common SHARED bus line if they contain the same cache line
 - writes to exclusive cache line are write-back
 - writes to shared cache lines are write-through and the other caches which contain the same cache lines are updated together with memory [**write-update protocol**]
 - when a cache line ceases to be shared, it needs an extra write-through cycle to find out that the cache line is no longer shared [**SHARED signal will not be asserted**]
 - sharing may be illusory e.g. (i) processor may no longer be using a *shared* cache line
(ii) process migration between CPUs [**sharing with an old copy of itself**]

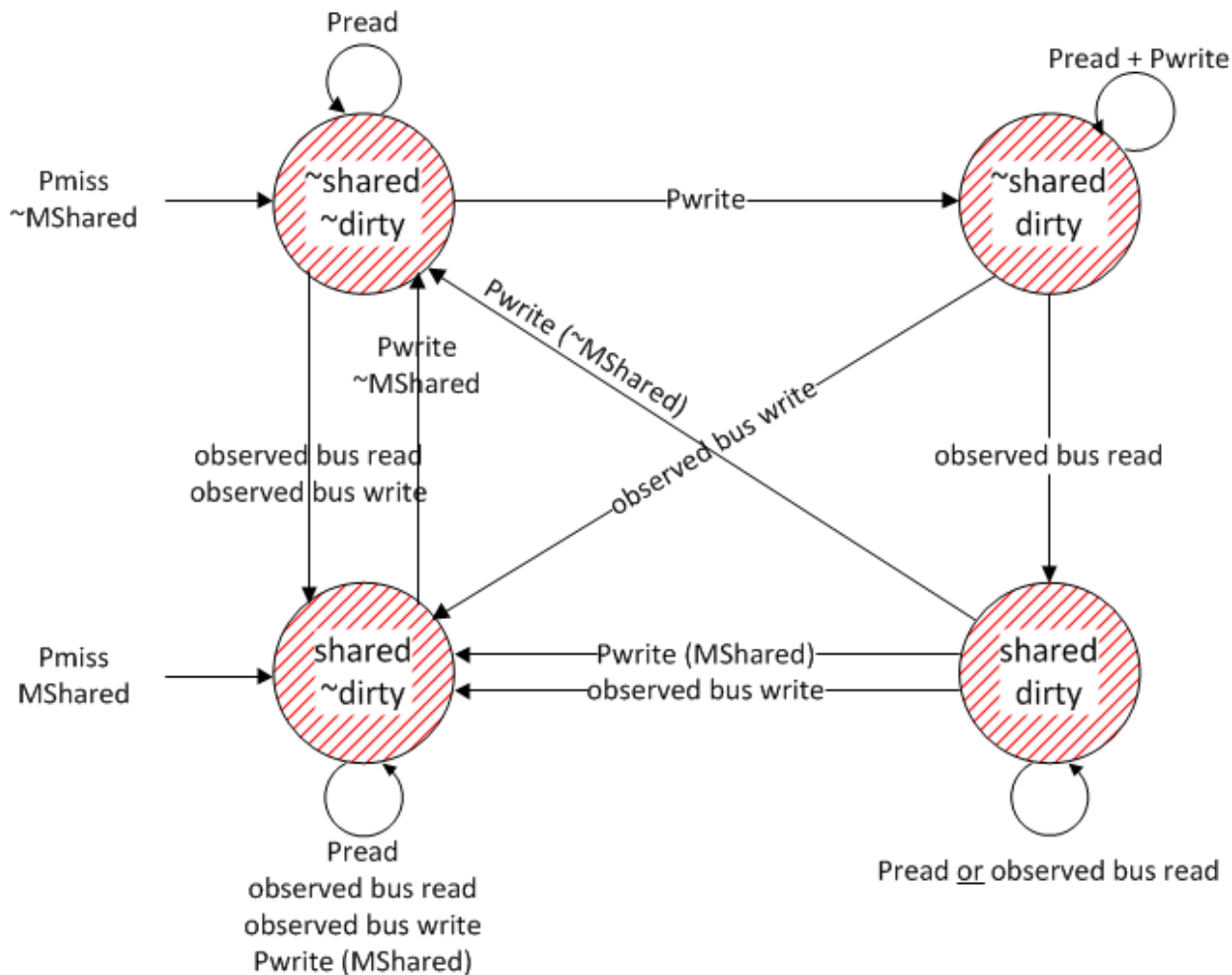
Multiprocessors

Firefly Protocol...

- as there is NO INVALID state
 - at reset the cache is placed in a *miss* mode and a bootstrap program fills cache with a sequence of addresses making it consistent with memory [**VALID state**]
 - during normal operation a location can be displaced if the cache line is needed to satisfy a miss, BUT the protocol never needs to invalidate cache line
- how is the Shared & Dirty state entered?
 - asymmetrical behaviour with regard to updating memory [**avoids changing memory cycle from a read cycle to a write cycle mid cycle**]
- try the vivio animation

Multiprocessors

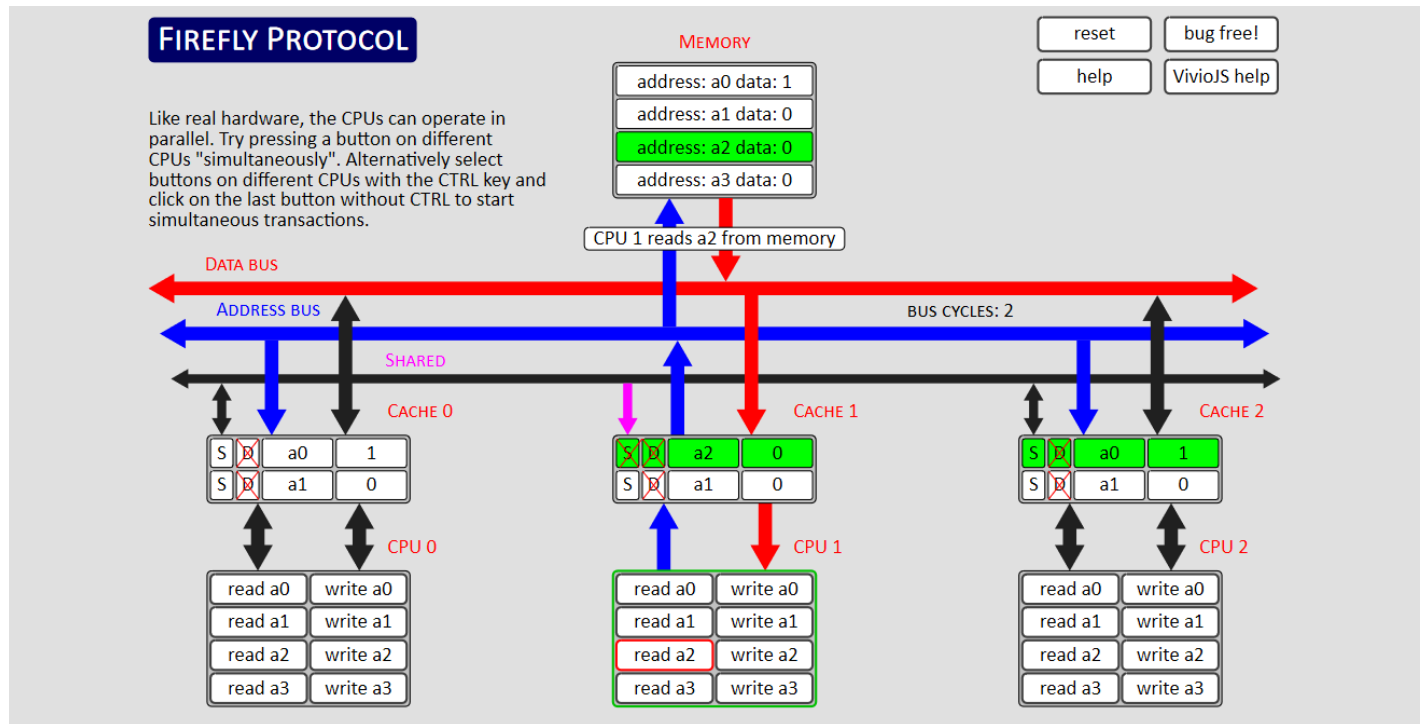
Firefly State Transition Diagram



Multiprocessors

Firefly Protocol...

- try the Vivio cache coherency animation



MESI Cache Coherency Protocol

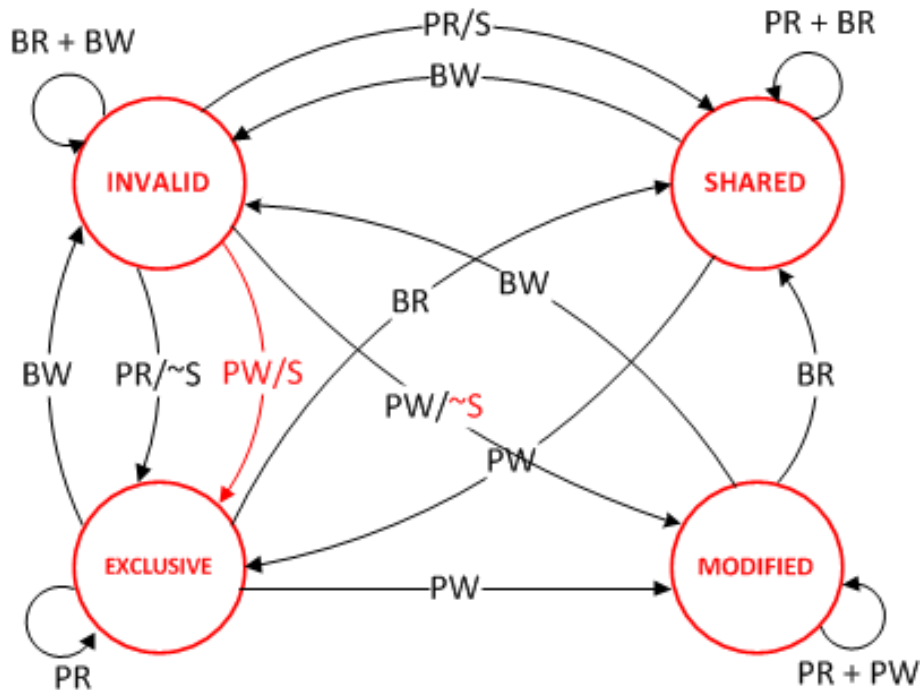
- used by Intel CPUs
- very similar to write-once, BUT uses a *shared* bus signal [**like Firefly**] so cache line can enter cache and be placed directly in the shared or exclusive state
- a write-invalidate protocol

MESI states compared with Write-Once states

- Modified \equiv dirty
 - Exclusive \equiv reserved
 - Shared \equiv valid
 - Invalid \equiv invalid
-
- what is the difference between the MESI and Write-Once protocols?
 - cache line may enter cache and be placed directly in the Exclusive [**reserved**] state
 - "write-once" write-through cycles no longer necessary if cache line is Exclusive
 - try the Vivio animation

Multiprocessors

MESI State Transition Diagram



PR = processor read
PW = processor write

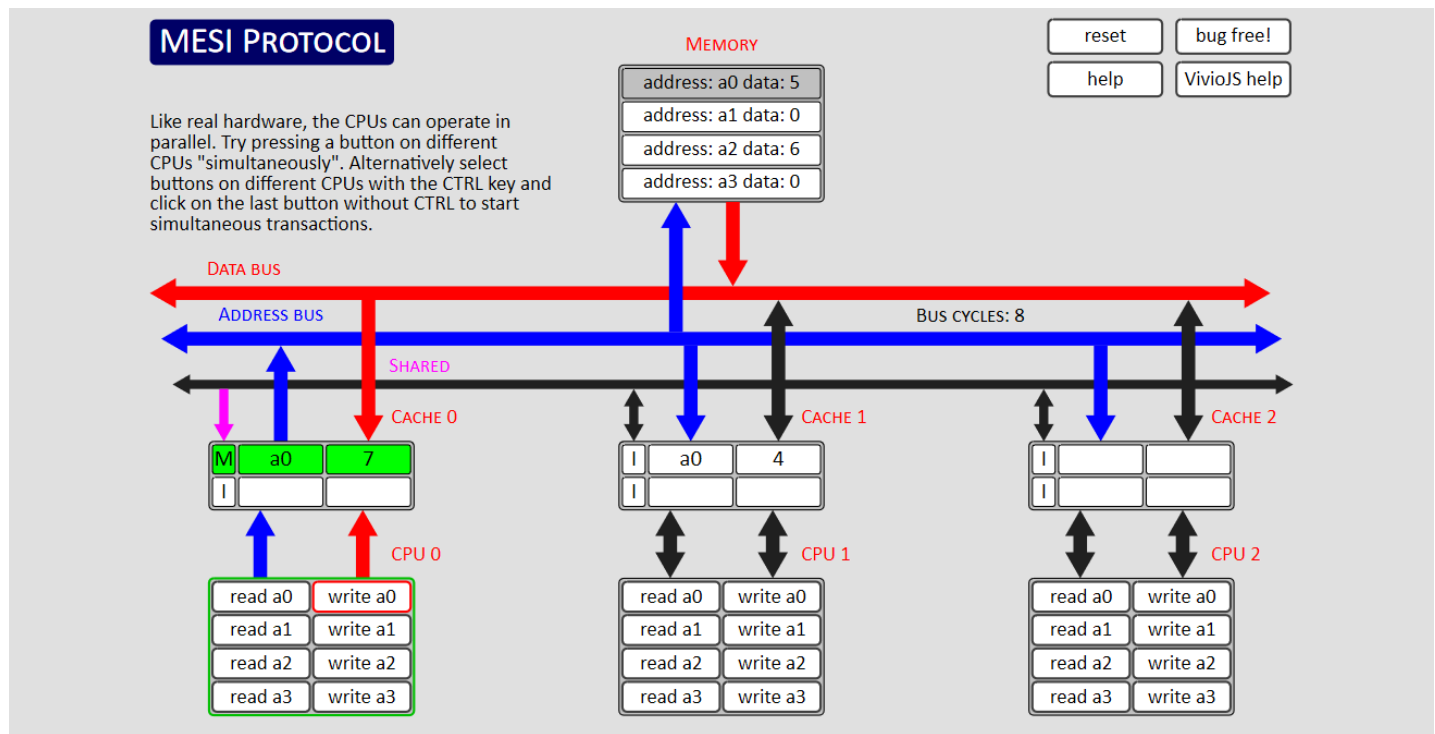
BR = observed bus read
BW = observed bus read

S/~S = shared/NOT shared

Multiprocessors

MESI Protocol...

- try the Vivio cache coherency animation



Multiprocessors

Summary

- you are now able to:
 - explain the cache coherency problem
 - describe the operation of the write-through, write-once, Firefly and MESI cache coherency protocols
 - predict the cache behaviour and bus traffic for a sequence of CPU "memory" accesses in a multiprocessor with a cache coherent bus
 - evaluate the cost updating shared data