

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

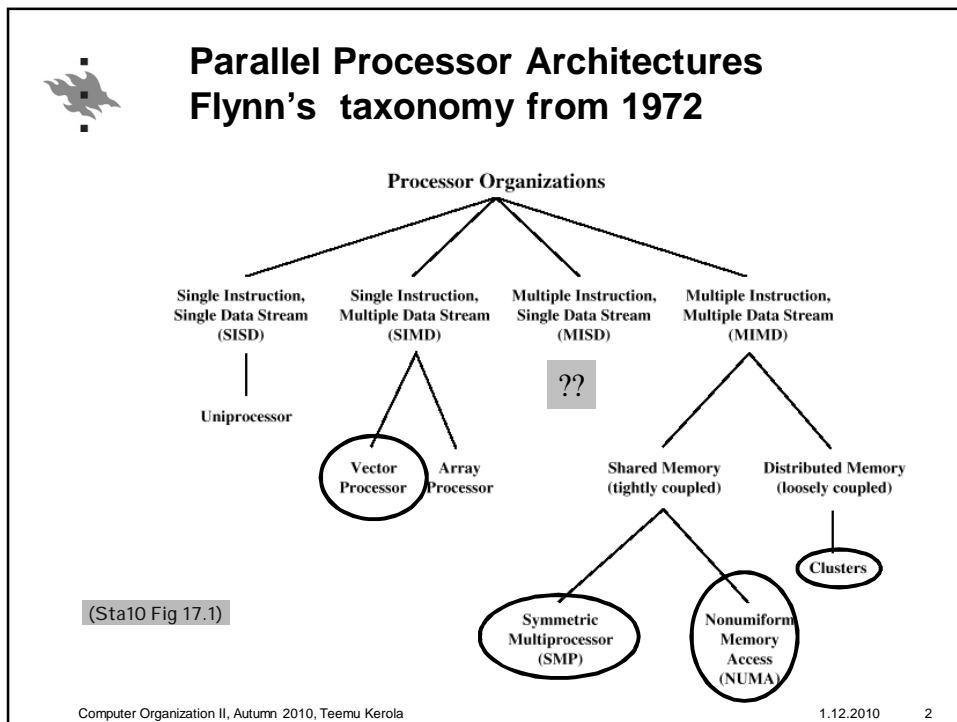
Lecture 11

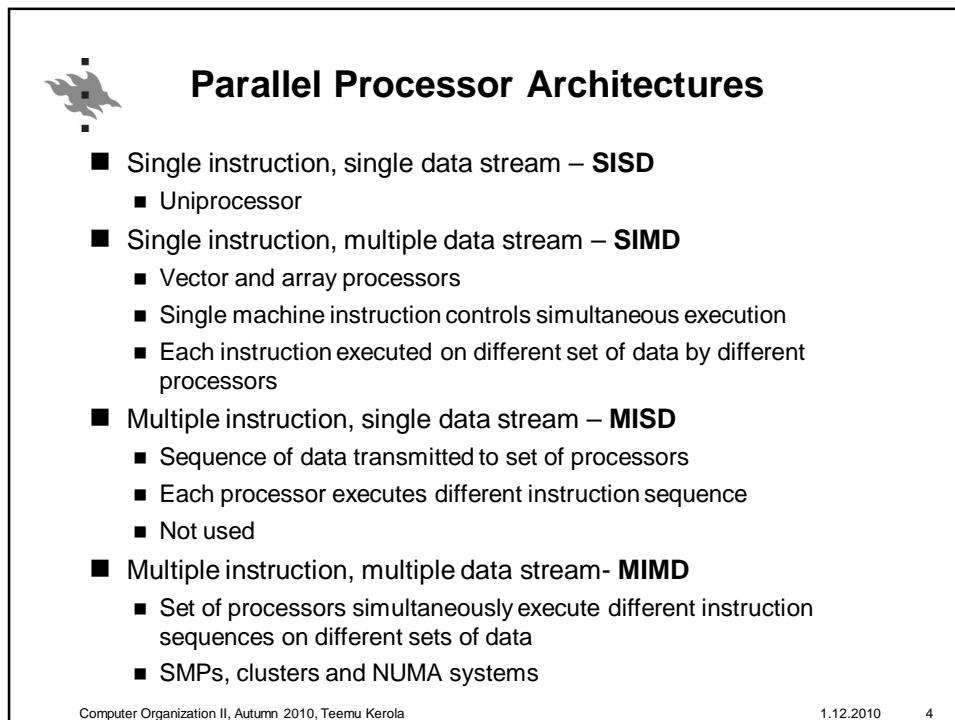
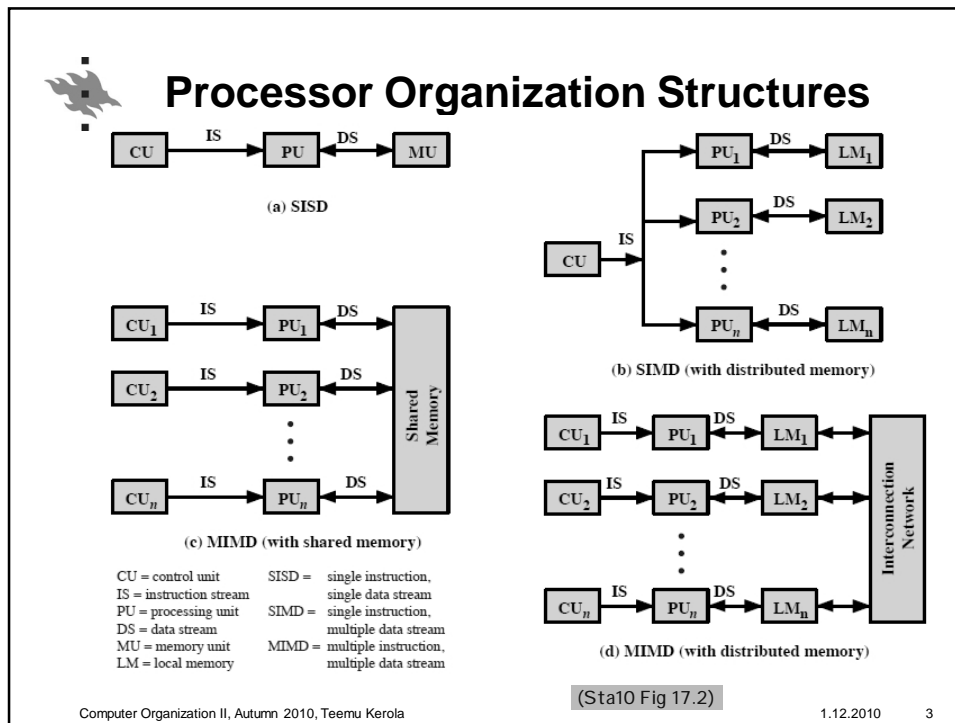
Parallel Processing

Ch 17 [Sta10]

Taxonomy

- SMP
- Cache Coherence – MESI Protocol
- NUMA and CC-NUMA
- Vector Computation







Multiple instruction, multiple data stream- MIMD

- Differences in processor communication
- Symmetric Multiprocessor (SMP)
 - Tightly coupled – communication via shared memory
 - Share single memory or pool, shared bus to access memory
 - Memory **access time** of a given memory location is **approximately the same** for each processor
- Non-uniform memory access (NUMA)
 - Tightly coupled – communication via shared memory
 - **Access times** to different regions of memory may **differ**
- Clusters
 - Loosely coupled – no shared memory
 - Communication via fixed path or network connections
 - Collection of independent uniprocessors or SMPs

Computer Organization II, Autumn 2010, Teemu Kerola

1.12.2010 5




SMP – Symmetric Multiprocessor

- Two or more similar processors of comparable capacity
- All processors can perform the same functions (hence symmetric)
- Connected by a bus or other internal connection
- Share same memory and I/O
- I/O access to same devices through same or different channels
- Memory access time is approximately the same for each processor
- System controlled by integrated operating system
 - providing interaction between processors
 - Interaction at job, task, file and data element levels

Computer Organization II, Autumn 2010, Teemu Kerola


1.12.2010 6



SMP – Advantages

- Performance
 - Only if some work can be done in parallel
- Availability
 - More processors to do the same functions
 - Failure of a single processor does not halt the system
- Incremental growth
 - Increase performance by adding additional processors
 - Is there a limit on this? Bus becomes serious bottleneck?
- Scaling
 - Different computers can have different number of processors
 - Vendors can offer range of products based on number of processors

Computer Organization II, Autumn 2010, Teemu Kerola 1.12.2010 7



Multiprogramming vs multiprocessing (Moniajo)

Time →

Multi-programming

Process 1: [Running] [Blocked] [Running] [Blocked] [Running] [Blocked] [Running]

Process 2: [Blocked] [Running] [Blocked] [Running] [Blocked] [Running] [Blocked]

Process 3: [Blocked] [Running] [Blocked] [Running] [Blocked] [Running] [Blocked]

(a) Interleaving (multiprogramming, one processor)

Multi-processing

Process 1: [Running] [Blocked] [Running] [Blocked] [Running] [Blocked] [Running]

Process 2: [Running] [Blocked] [Running] [Blocked] [Running] [Blocked] [Running]

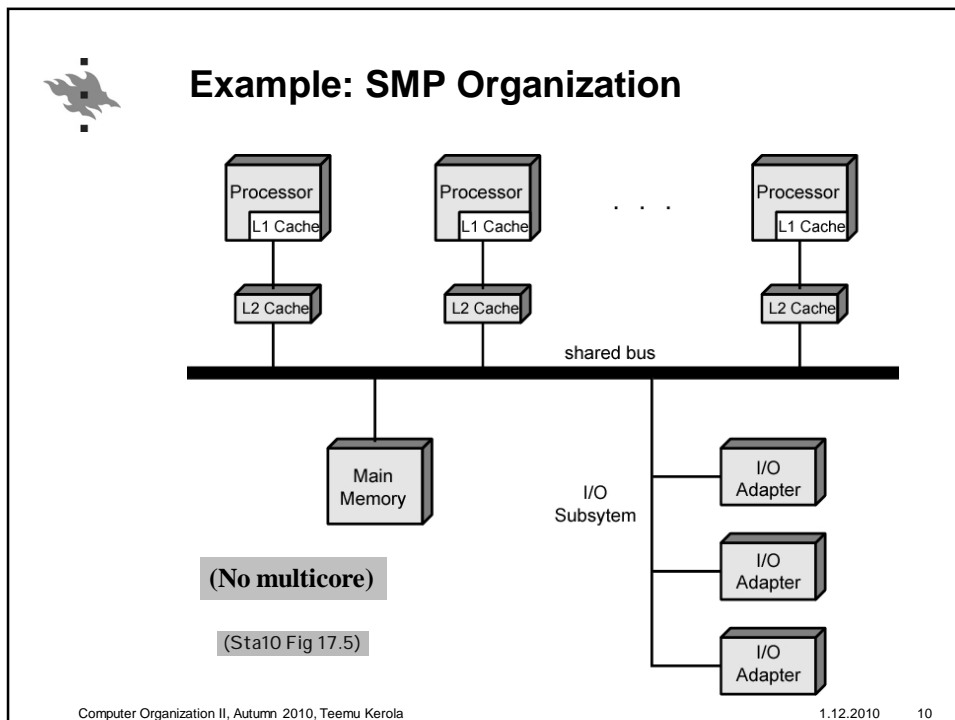
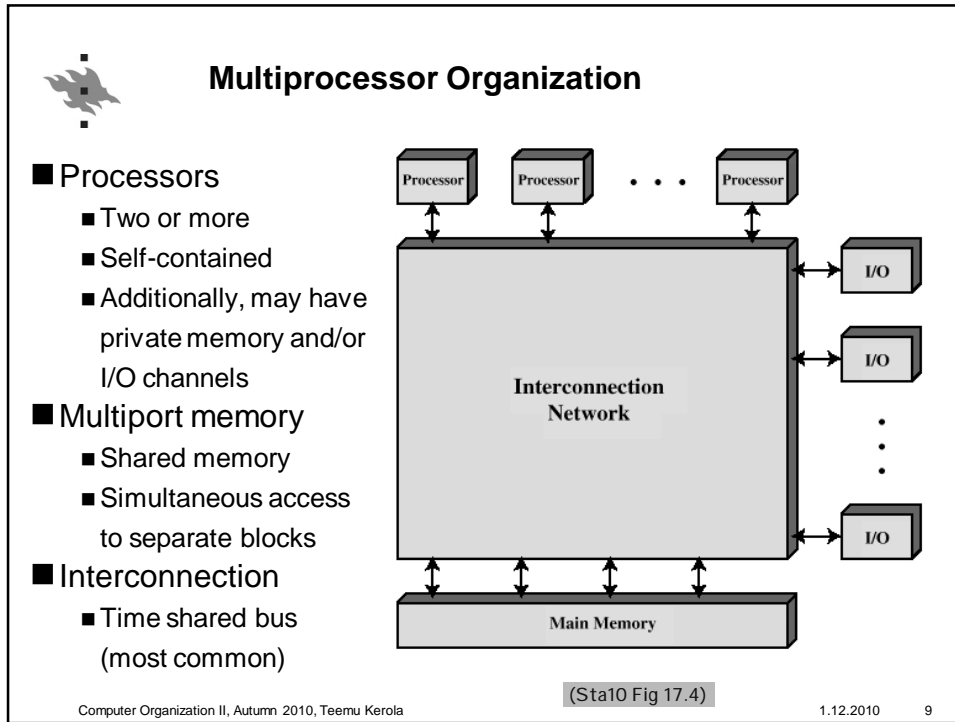
Process 3: [Running] [Blocked] [Running] [Blocked] [Running] [Blocked] [Running]


(b) Interleaving and overlapping (multiprocessing; multiple processors)

(Sta10 Fig 17.3)

Blocked
 Running

Computer Organization II, Autumn 2010, Teemu Kerola 1.12.2010 8






Time-shared bus

Advantages	Disadvantages
<ul style="list-style-type: none"> ■ Simplicity <ul style="list-style-type: none"> ■ Addressing, arbitration and time-sharing logic same as in uniprocessor system ■ Flexibility <ul style="list-style-type: none"> ■ Expand by attaching more processors to the bus ■ Reliability <ul style="list-style-type: none"> ■ Bus is passive, failure of attached device should not cause failure of the whole 	<ul style="list-style-type: none"> ■ Performance limited by bus cycle time ■ Each processor should have local cache <ul style="list-style-type: none"> ■ Reduce number of bus accesses ■ Leads to problems with <u>cache coherence</u> <ul style="list-style-type: none"> ■ Solved in hardware - see later

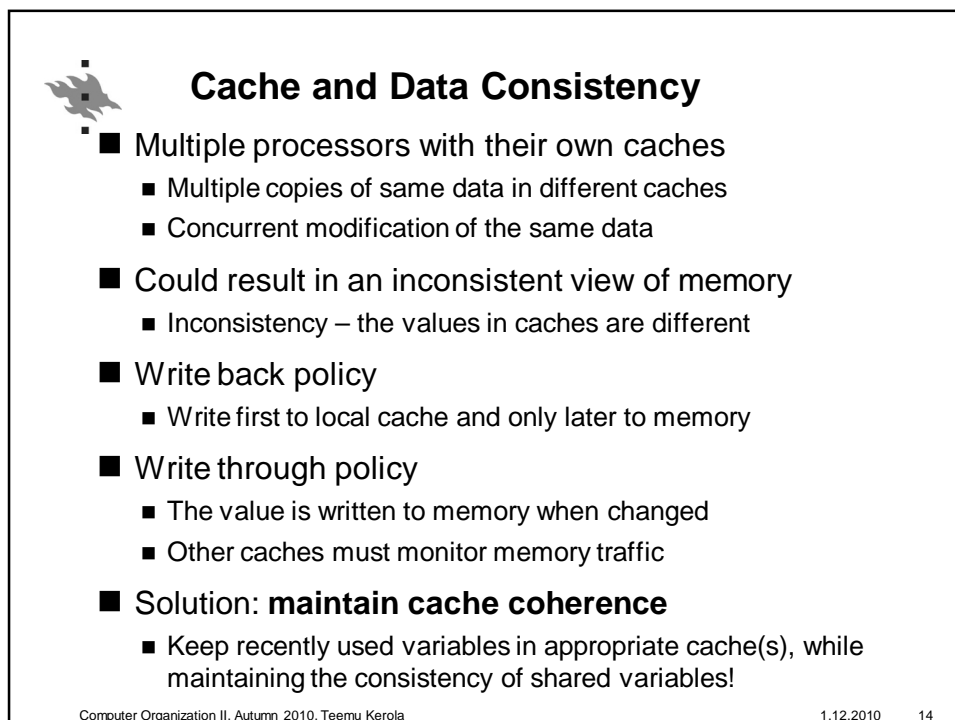
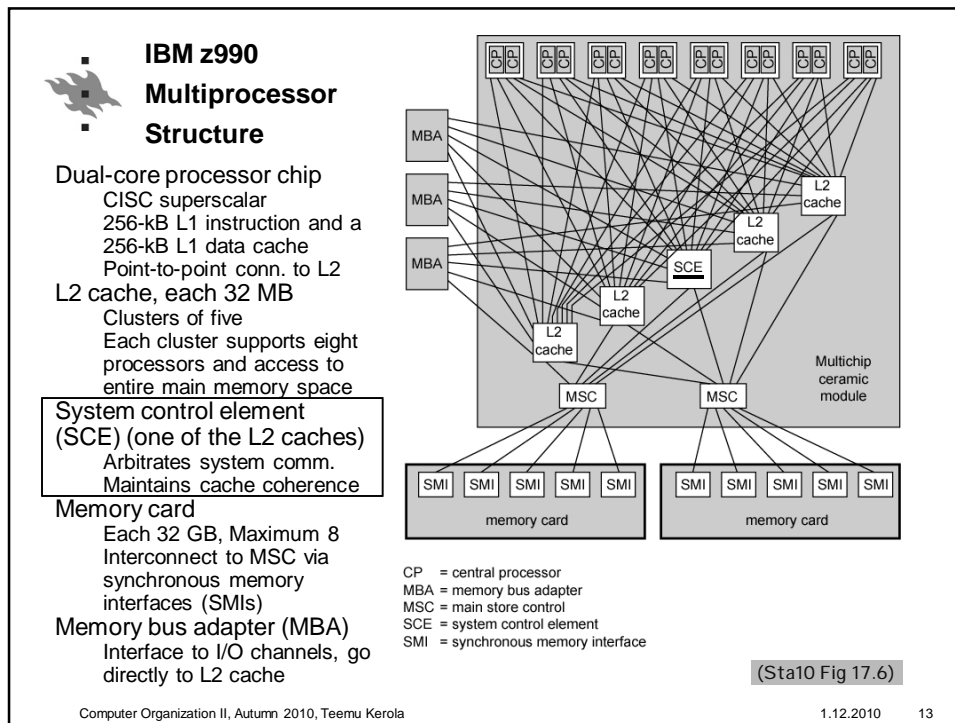
Computer Organization II, Autumn 2010, Teemu Kerola 1.12.2010 11



New Requirements to Operating System

- **Simultaneous concurrent processes**
 - Reentrant OS routines (only local data, no conc. problems)
 - OS data structure synchronization – avoid deadlocks etc.
- **Scheduling**
 - On SMP any processor may execute scheduler at any time
- **Synchronization**
 - Controlled access to shared resources
- **Memory management**
 - Use parallel access options
- **Reliability and fault tolerance**
 - Graceful degradation in the face of single processor failure

Computer Organization II, Autumn 2010, Teemu Kerola 1.12.2010 12





Software Solutions for Cache Coherence

- Compiler and operating system deal with problem
- Overhead transferred to compile time
- Design complexity transferred from hardware to software
- However, software tends to make conservative decisions
 - Inefficient cache utilization - do not cache shared variables
- Analyze code to determine safe periods for caching shared variables



Hardware Solutions for Cache Coherence

- Dynamic recognition of potential problems at run time
- More efficient use of cache, transparent to programmer
- Directory protocols
 - Collect and maintain information about copies of data in cache
 - Directory stored in main memory
 - Requests are checked against directory
 - Creates central bottleneck
 - Effective in large scale systems with complex interconnections
- Snoopy protocols
 - Distribute cache coherence responsibility to all cache controllers
 - Cache recognizes that a line is shared
 - Updates announced to other caches
 - Suited to bus based multiprocessor



Snoopy Cache Protocols

- Write-Invalidate
 - Multiple readers, one writer
 - Write request invalidates that line in all other caches
 - Writing processor gains exclusive (cheap) access until line required by another processor
 - Used in Pentium II and PowerPC systems
 - State of every line marked as **modified**, **exclusive**, **shared** or **invalid** (MESI)
- Write-Update
 - Multiple readers and writers
 - Updated word is distributed to all other processors
- Some systems use an adaptive mixture of both solutions

Computer Organization II, Autumn 2010, Teemu Kerola

1.12.2010 17



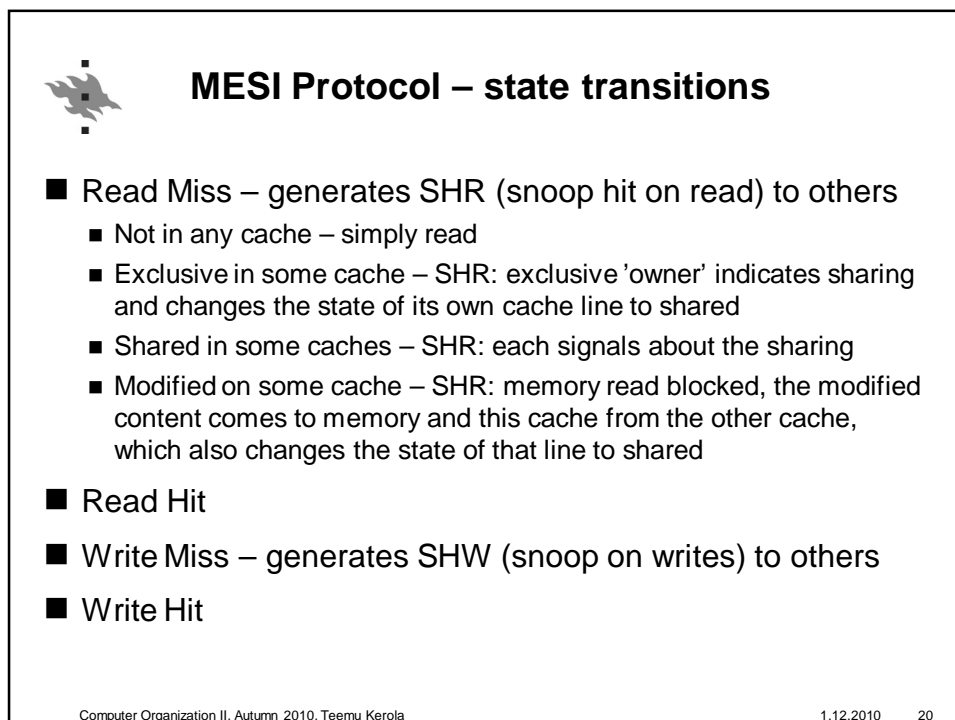
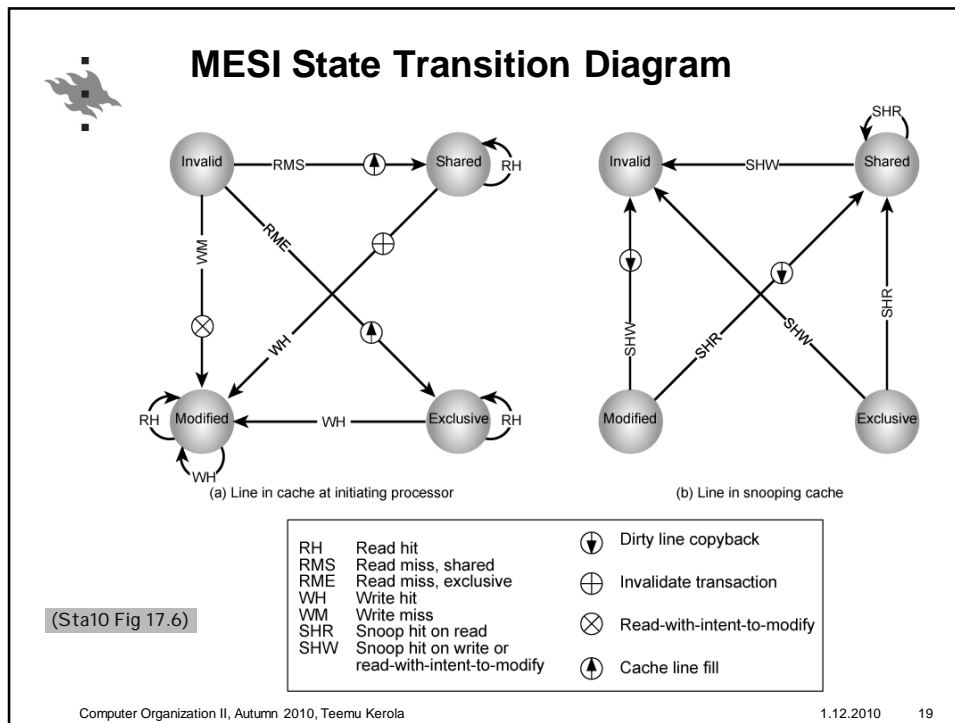
MESI Protocol


- Four states (two bits per tag)
 - **Modified**: modified cache line (only in this cache)
 - **Exclusive**: only in this cache, but the same as memory
 - **Shared**: same as memory, may be in other caches
 - **Invalid**: line does not contain valid data

	M Modified	E Exclusive	S Shared	I Invalid
This cache line valid?	Yes	Yes	Yes	No
The memory copy is...	out of date	valid	valid	—
Copies exist in other caches?	No	No	Maybe	Maybe
A write to this line...	does not go to bus	does not go to bus	goes to bus and updates cache	goes directly to bus

Computer Organization II, Autumn 2010, Teemu Kerola

1.12.2010 18






Multithreading

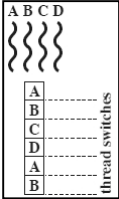
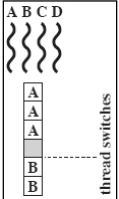
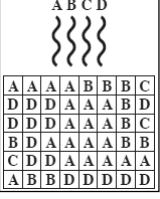
- Process
 - Resources
 - Scheduling
 - Process switch
 - 1 or more executable threads
 - Private stack, shared resources (e.g., memory)
 - Thread switch
 - Kernel level threads are OS controlled
 - User level threads are process controlled
 - Thread level parallelism

Computer Organization II, Autumn 2010, Teemu Kerola
1.12.2010 21

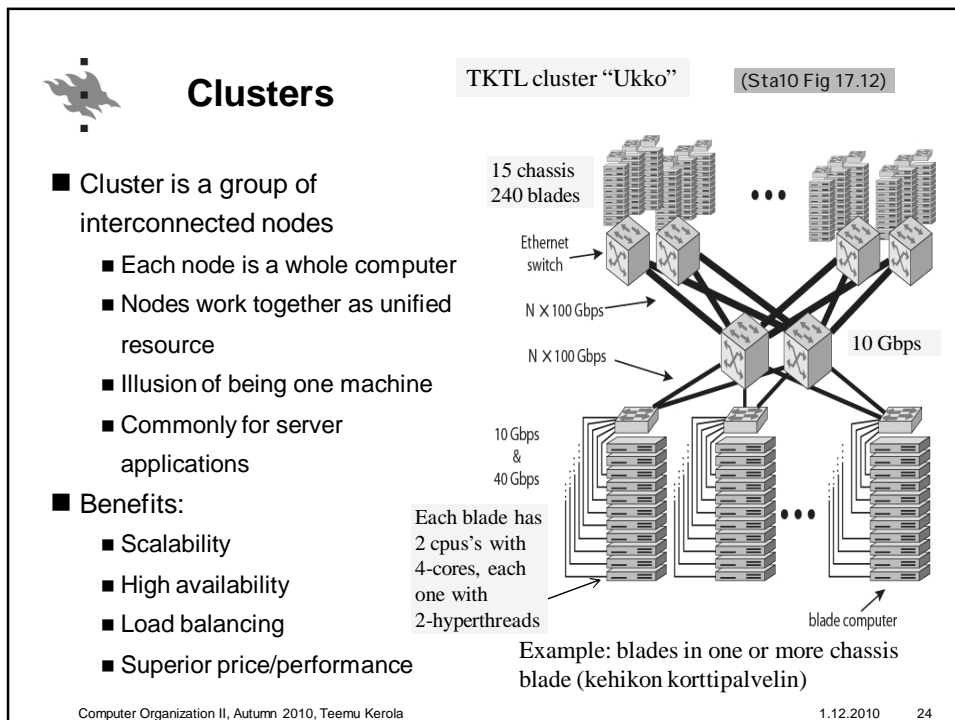
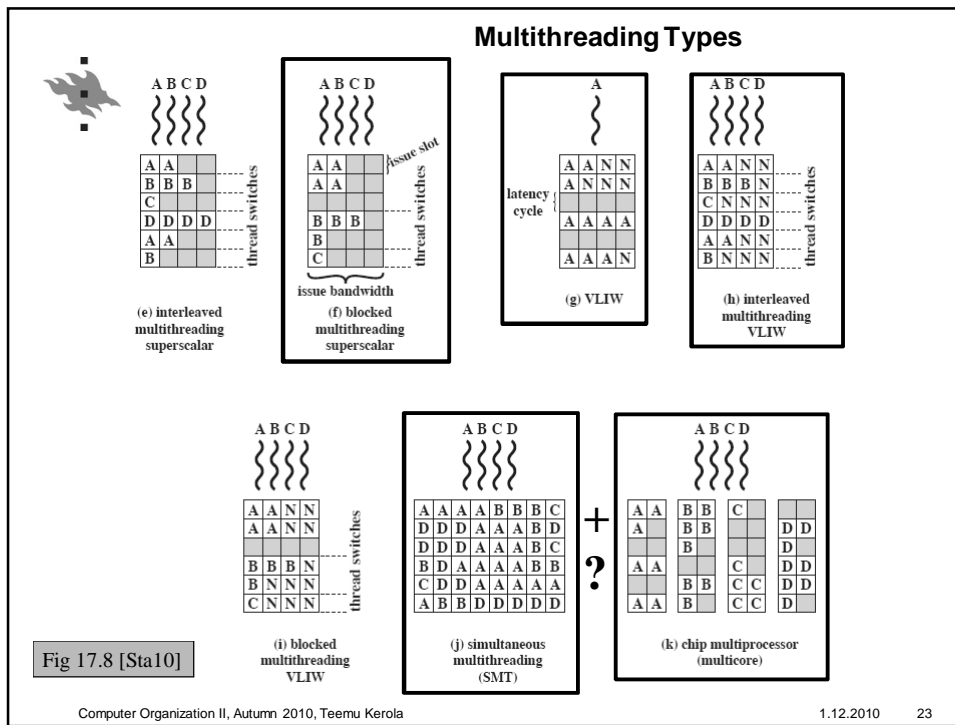


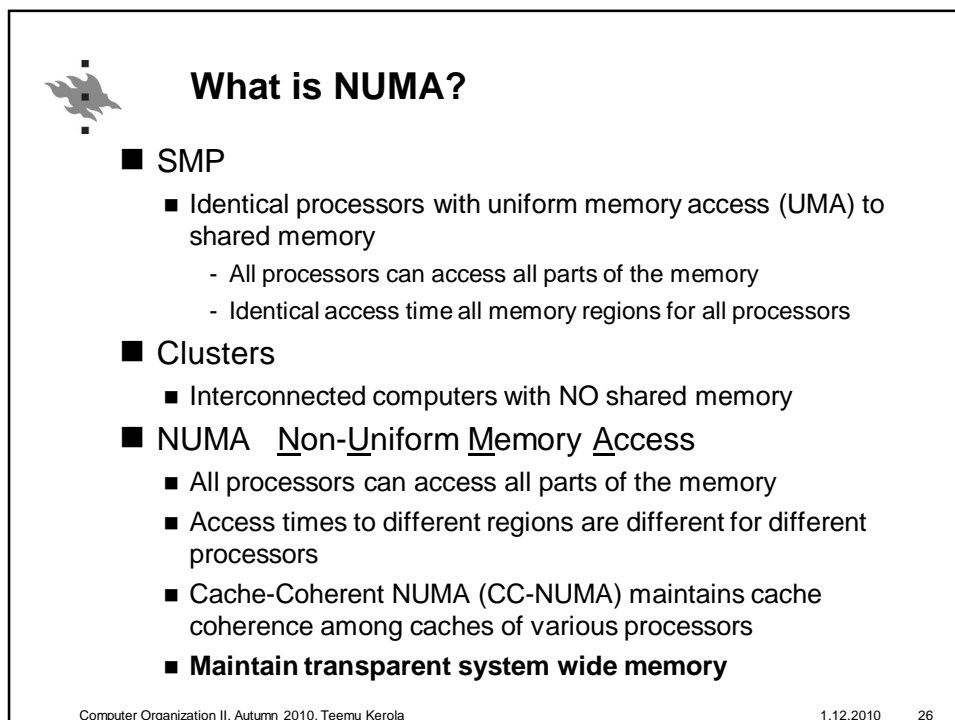
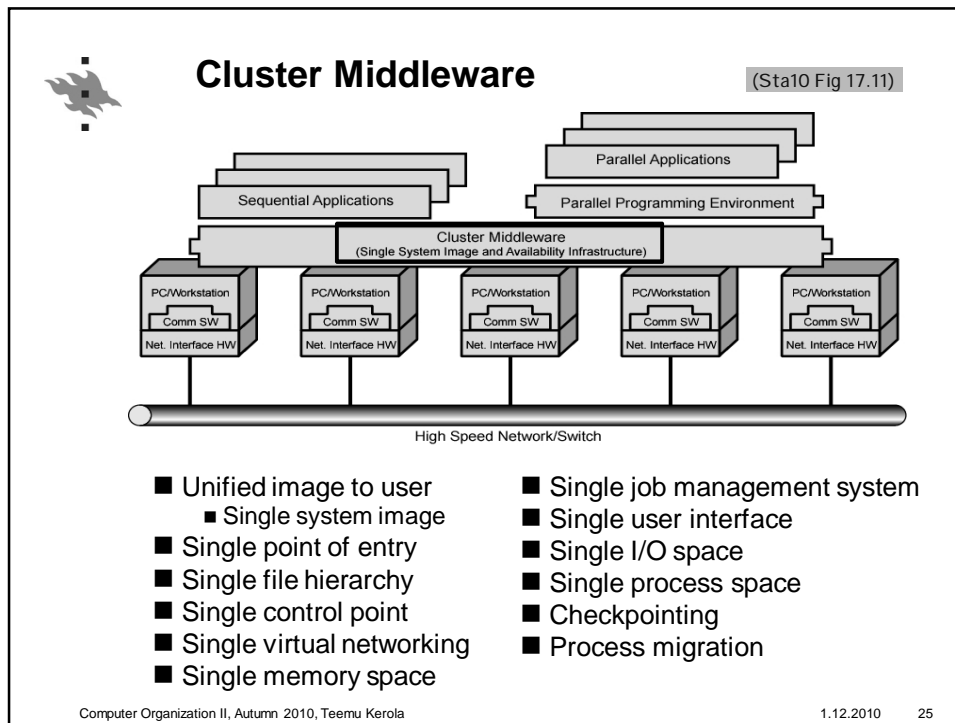
Multithreading Types

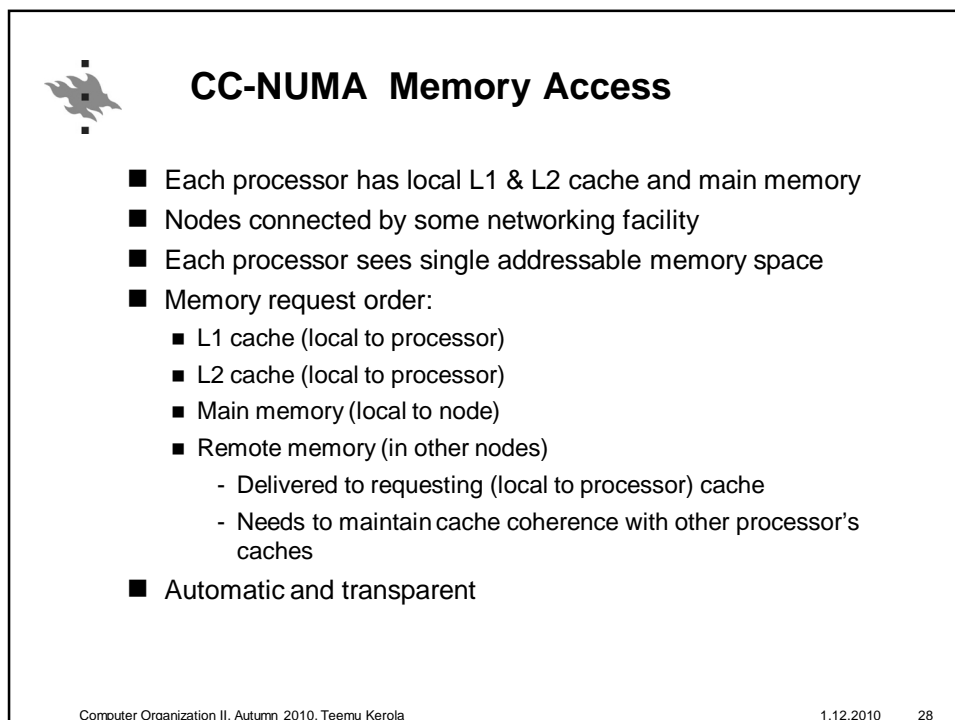
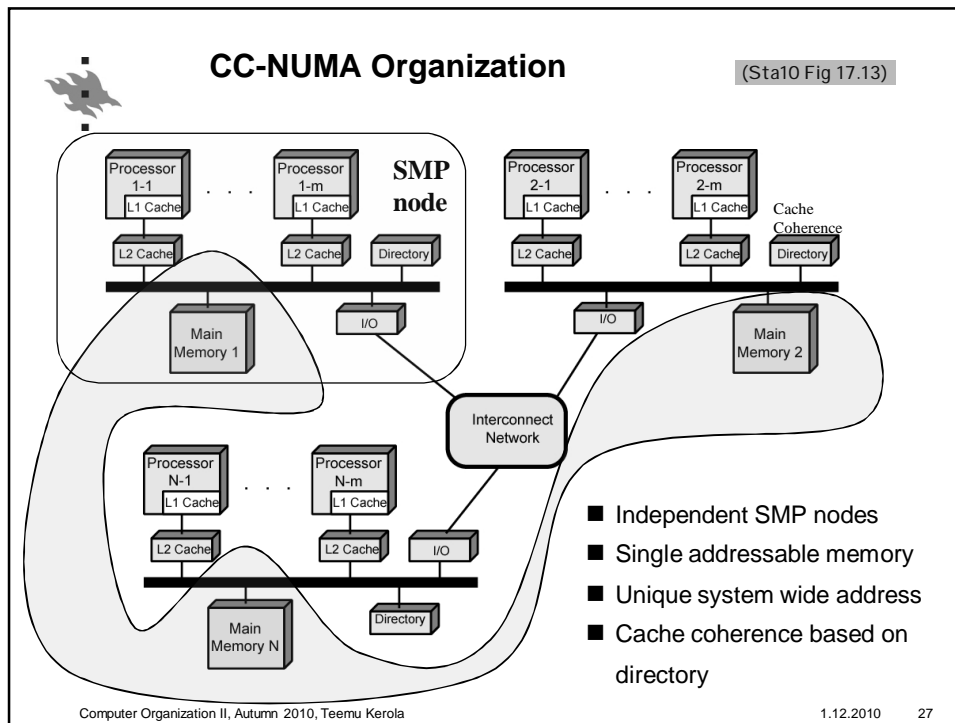
- Interleaved multithreading
 - “Fine-grained multithreading”,
 - Separate hw context (registers) for each thread
 - Alternate thread execution after each clock cycle
- Blocked multithreading
 - “Coarse grained multithreading”
 - Switch thread when previous one is blocked (e.g., cache miss)
- Simultaneous multithreading (SMT)
 - Intel “hyperthreading”
 - Instructions from multiple threads in superscalar processor
 - Many contexts (register sets) at use simultaneously’
- Chip multiprocessing
 - “Multicore”
 - Many complete cpu’s on one chip, not necessarily symmetric!
 - cpu’s (cores) may share some L2 or L3 cache

Computer Organization II, Autumn 2010, Teemu Kerola
1.12.2010 22









Vector Computation

- SIMD instructions
 - One dimensional data
- When needed?
- When used in HLL code?
 - Vector lengths determined by application
- How to implement in HW?
 - Vector length determined by HW

Computer Organization II, Autumn 2010, Teemu Kerola

1.12.2010 292929



Matrix Multiplication with Vectors

serial

Fortran

```

do 100 i = 1,n
  do 100 j = 1, n
    c(i,j) = 0.0
    do 100 k = 1,n
      c(i,j) = c(i,j) + a(i,k) + b(k,j)
    100 continue
  100 continue

```

(Sta10 Fig 17.15)

vector

```

do 100 i = 1,n
  c(i,j) = 0.0 (j=1,n)
  do 100 k = 1,n
    c(i,j) = c(i,j) + a(i,k) + b(k,j) (j=1,n)
  100 continue

```

- Matrix multiplication: $C = A \times B$
 - Compiler technology for vector machines is outside course objectives
 - Vector register length varies
 - 8 64-bit FP registers?

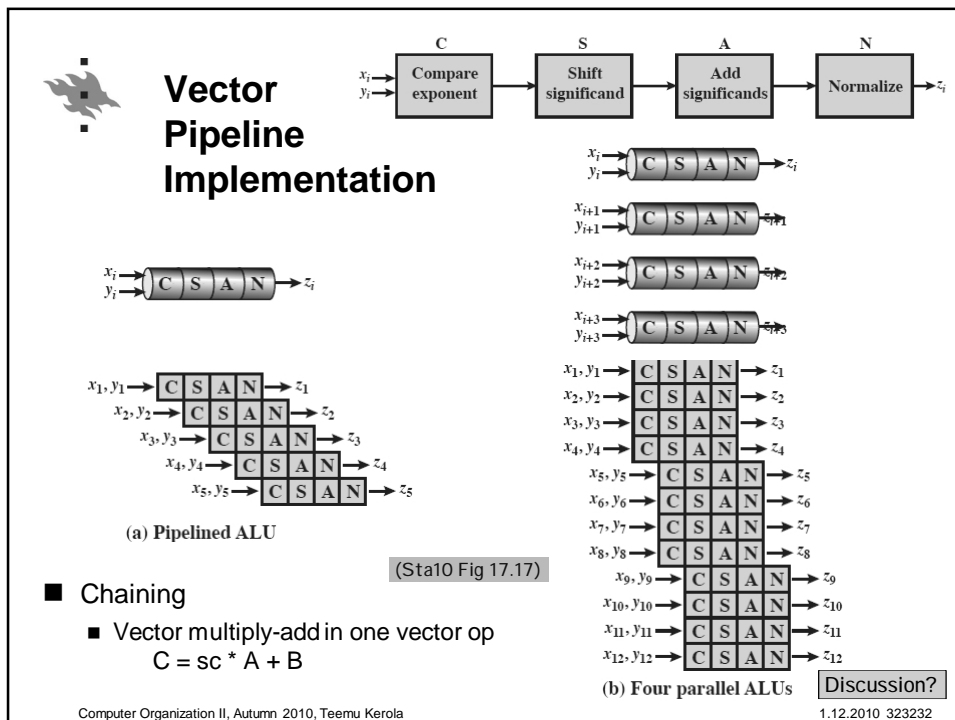
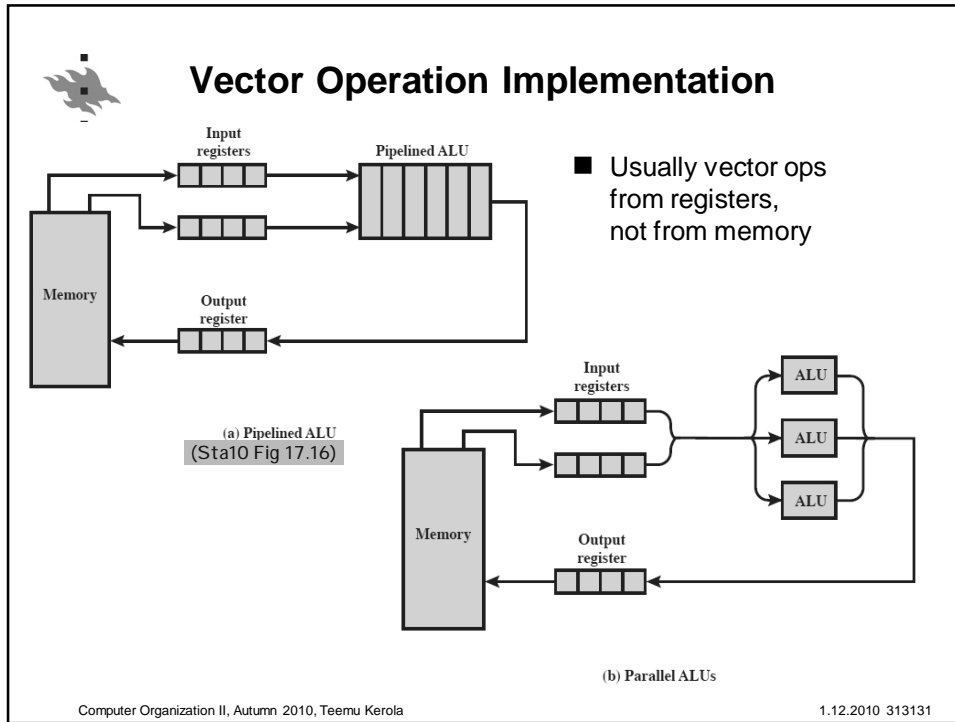
```

...
vload vr1, a[k] ; 8 values at a time?
vload vr2, b[k]
vadd vr3, vr1, vr2
....
vstore vr3, c[k]

```

Computer Organization II, Autumn 2010, Teemu Kerola

1.12.2010 303030





Parallel Processing Summary

- Parallel processing classification
- SMP, NUMA, CC-NUMA
- SMP: Shared memory becomes bottleneck with many processors
- CC-NUMA
 - Performance suffers if too much remote memory access
 - Need good temporal and spatial locality of software with
 - L1 & L2 cache design to reduce all memory access
 - Virtual memory management move pages to nodes that use them most
 - Not truly transparent memory access
 - Page allocation, process allocation and load balancing changes needed
- Vector instructions



Review Questions / Kertauskysymyksiä

- Cache coherence and MESI protocol
- Differences and similarities of SMP, NUMA and cluster
- When would you use vector operations?