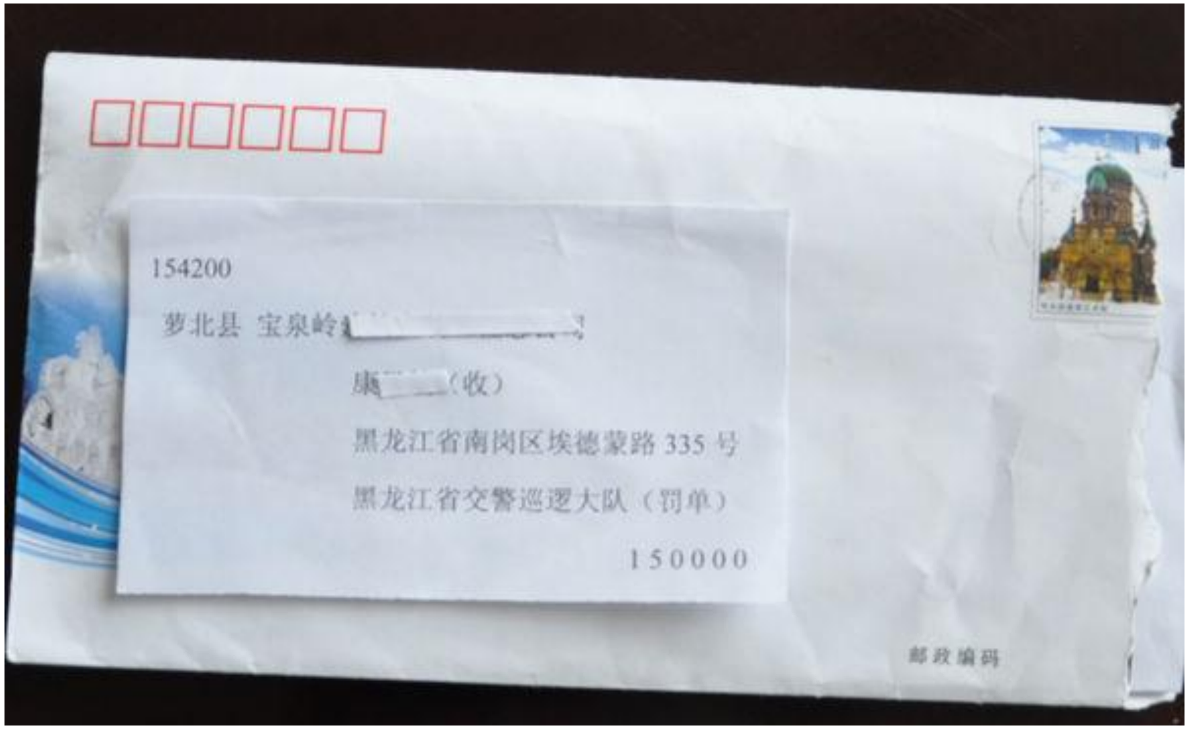
Message passing

- Message passing
- SMP/multi-core system
- Message passing in SMP

- Message passing

Message passing

- in the real world



154200

萝北县 宝泉岭[redacted]

康[redacted]（收）

黑龙江省南岗区埃德蒙路 335 号

黑龙江省交警巡逻大队（罚单）

150000

邮政编码

Abstraction

- message
- sender
- receiver
- send
- receive

Message

- a structure
  - sender
  - receiver
  - type
  - content
- How to interpret the type and content is up to the sender and receiver.

Sender & Receiver

- the entities who manipulate the message
  - in Nanos, they are all threads/processes
- sender:
  - fill the message
  - send it
- receiver:
  - receive the message
  - do something according to the content of the message

Send & Receive

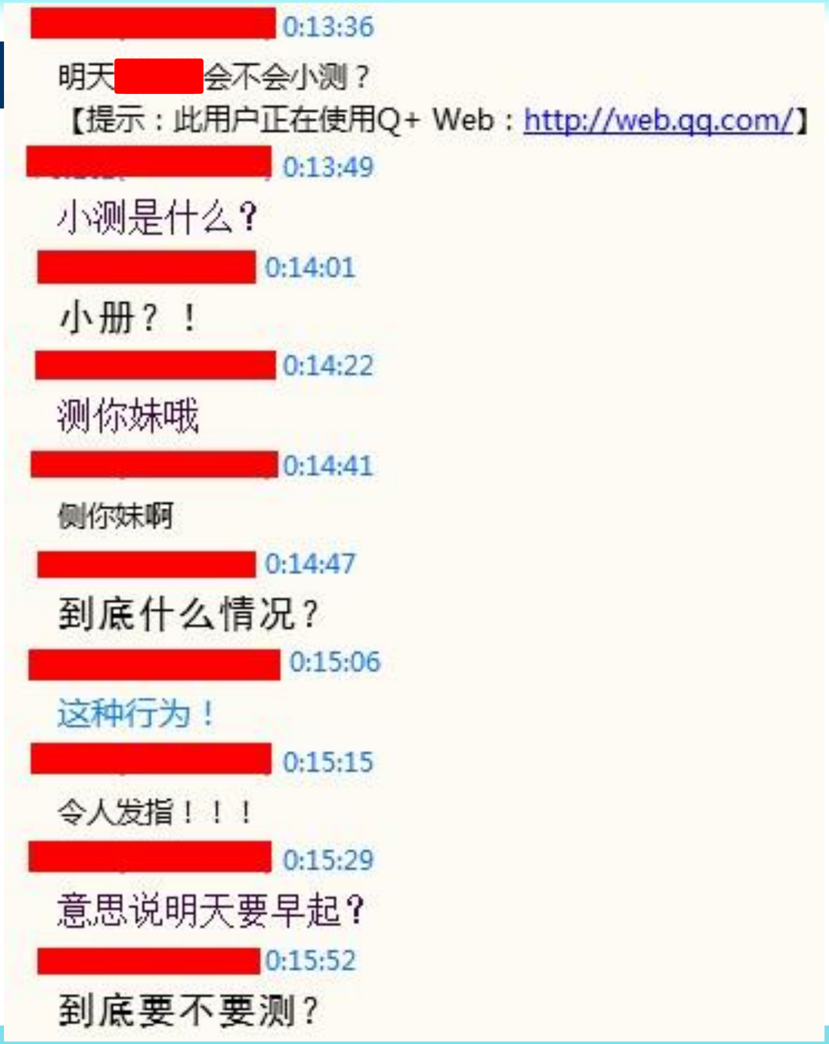
- send(dest, m):
  - deliver the message m to dest
    - every process has a "mail box"
    - put the message into receiver's "mail box"
    - always succeed, return immediately
- receive(src, m):
  - pick a message from src from the "mail box"
    - src can be anyone
  - if no message wanted is available, wait until one comes

The power

- communication
- notification
- synchronization
- multi-value returning
- decoupling

Communication

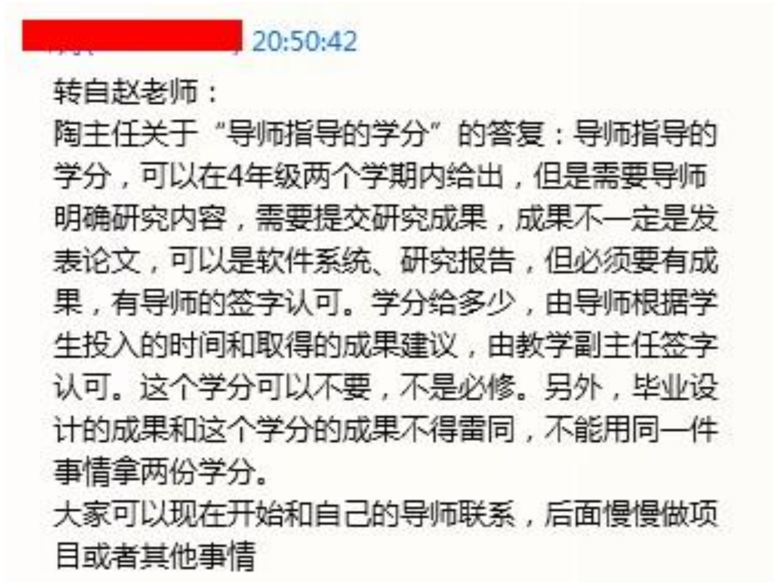
- send some data to other
  - the natural function



[redacted] 0:13:36
明天[redacted]会不会小测？
【提示：此用户正在使用Q+ Web：http://web.qq.com/】
[redacted] 0:13:49
小测是什么？
[redacted] 0:14:01
小册？！
[redacted] 0:14:22
测你妹哦
[redacted] 0:14:41
侧你妹啊
[redacted] 0:14:47
到底什么情况？
[redacted] 0:15:06
这种行为！
[redacted] 0:15:15
令人发指！！！
[redacted] 0:15:29
意思说明天要早起？
[redacted] 0:15:52
到底要不要测？

Notification

- send a message for occurence of some events
  - this is a special case of communication
  - usually to tell the receiver to handle the events



[redacted] 20:50:42

转自赵老师：
陶主任关于“导师指导的学分”的答复：导师指导的学分，可以在4年级两个学期内给出，但是需要导师明确研究内容，需要提交研究成果，成果不一定是发表论文，可以是软件系统、研究报告，但必须要有成果，有导师的签字认可。学分给多少，由导师根据学生投入的时间和取得的成果建议，由教学副主任签字认可。这个学分可以不要，不是必修。另外，毕业设计的成果和这个学分的成果不得雷同，不能用同一件事情拿两份学分。
大家可以现在开始和自己的导师联系，后面慢慢做项目或者其他事情

Synchronization

Synchronization (cont.)

- implement the relationship of “happen after”
  - if A should happen after B, then block the process until B finishes
  - receive() can exactly do the same thing!
- running

```
void runner(int id) {
        relax_himself();
        receive(referee, “ready!”);
        ready();
        receive(referee, “go!”);
        run();
}
```

Communication & Synchronization

- communication + synchronization = network
- client
  - send requests (communication)
  - wait for reply (synchronization)
- server
  - wait for request (synchronization)
  - process request
  - send reply (communication)

Multi-value returning

- When replying, a message can carry more than one return value.
  - more flexible than the function calls

Decoupling

- Processes communicate only by message passing.
- Other information is private.
  - invisible for other processes
  - fewer critical regions!
- This is exactly the core idea of micro-kernel.

Implementation

- Every “mail box” is a critical area!
  - Different processes may send messages to a certain process simultaneously.
- Receiver cannot receive message when the “mail box” is empty!
- a problem with mutex and synchronization

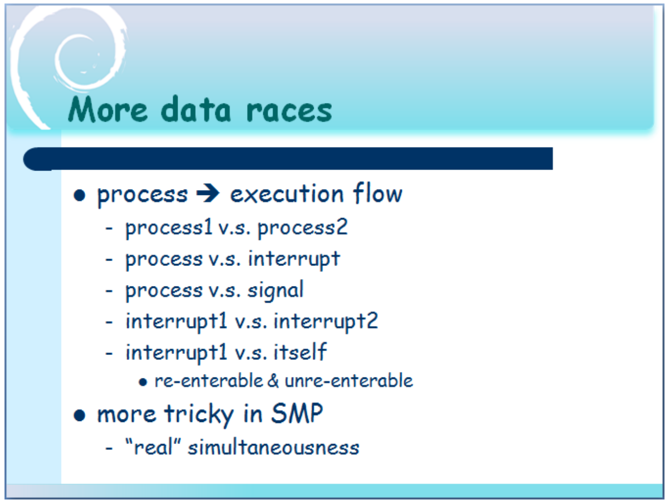
Implementation (cont.)

- In Nanos, send() is asynchronous.
  - send() always secceeds without being blocked
  - because we will use send in the interrupt handlers
  - What does this mean?
- low-level API avaliable
  - semaphore
  - locking
- How to implement message passing with them?

- SMP/multi-core system

Recap – critical region

- In a uni-processor system, only process-level critical region exists.
  - physical limit
  - only one execution flow can run at any point of time
- the root cause:
  - interrupts
- an effective solution:
  - disable interrupt



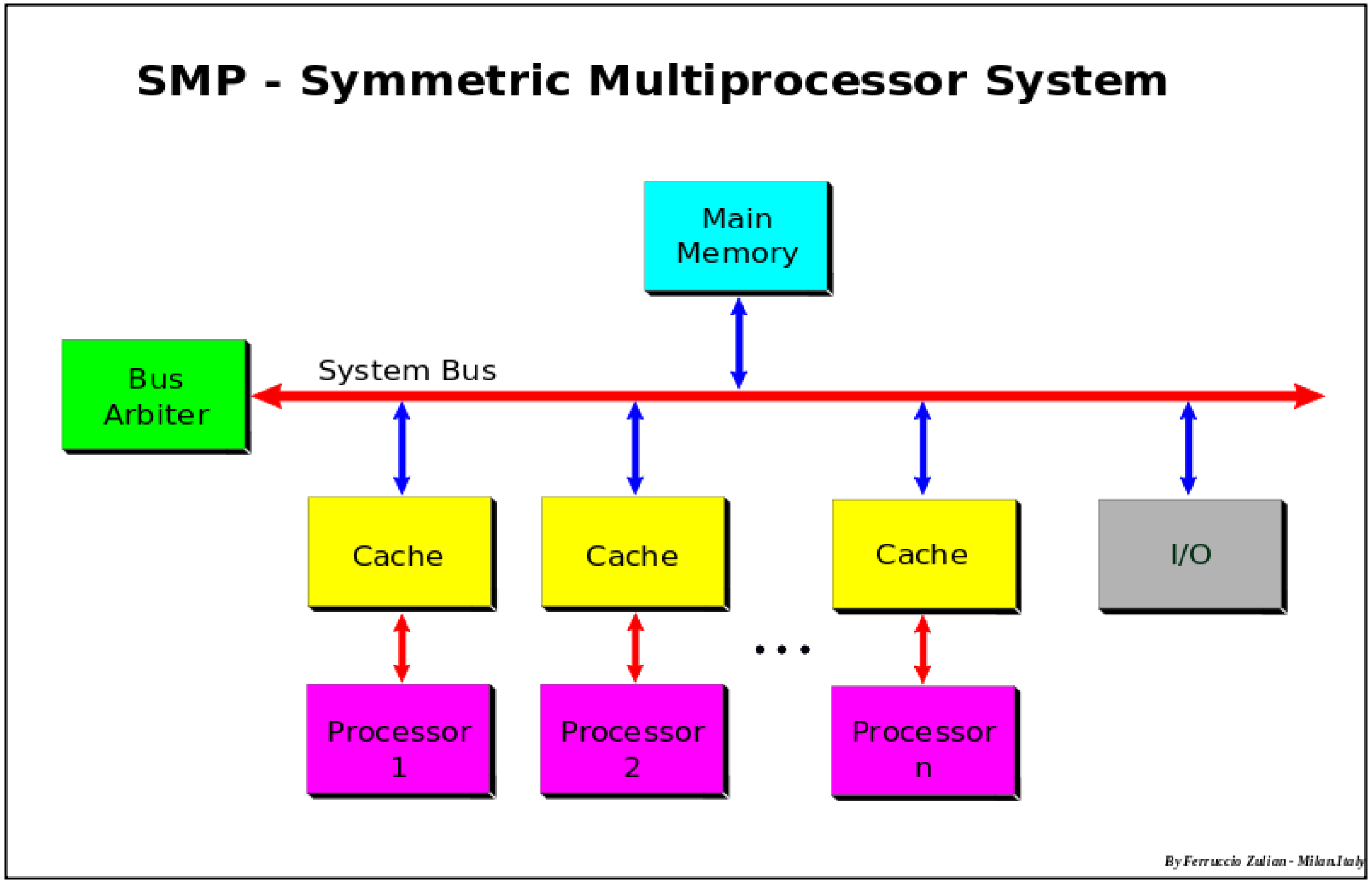
More data races

- process → execution flow
  - process1 v.s. process2
  - process v.s. interrupt
  - process v.s. signal
  - interrupt1 v.s. interrupt2
  - interrupt1 v.s. itself
    - re-enterable & unre-enterable
- more tricky in SMP
  - "real" simultaneousness

But in SMP...

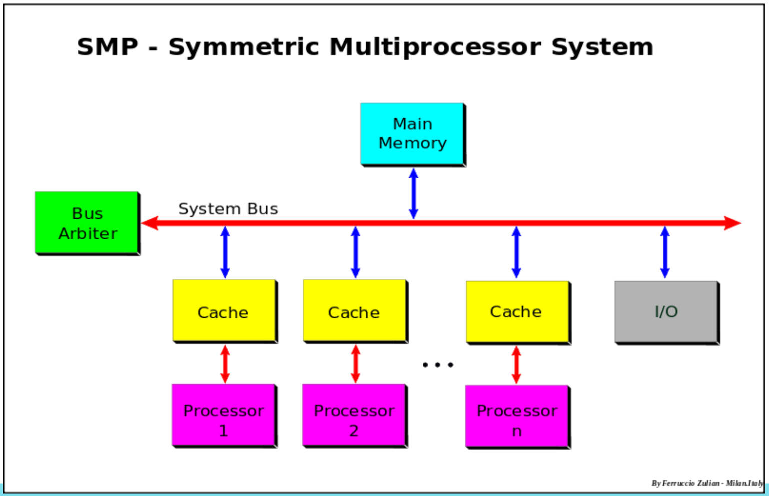
- Everything becomes tricky.
- the root cause
  - parallelism
  - not only concurrency
- But first you should know how SMP works.

SMP


SMP - Symmetric Multiprocessor System
Main Memory
Bus Arbiter
System Bus
Cache
Cache
Cache
I/O
Processor 1
Processor 2
...
Processor n
By Ferruccio Zulian - Milan.Italy

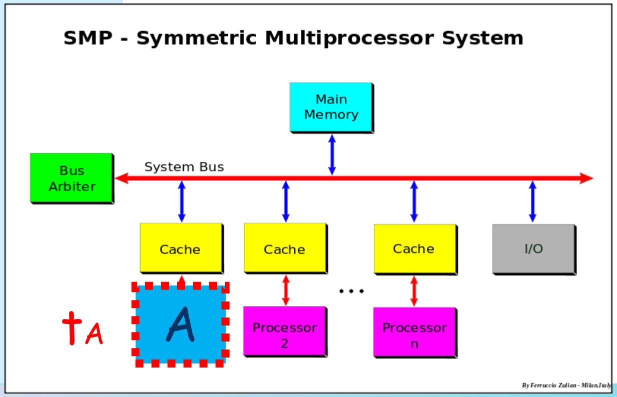
SMP

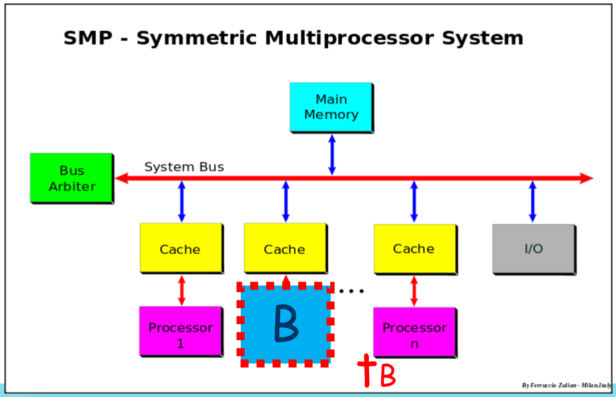
- Processors connect with each other via system bus.
- Each processor has its own L1 cache.
- Each processor can perform computation individually.

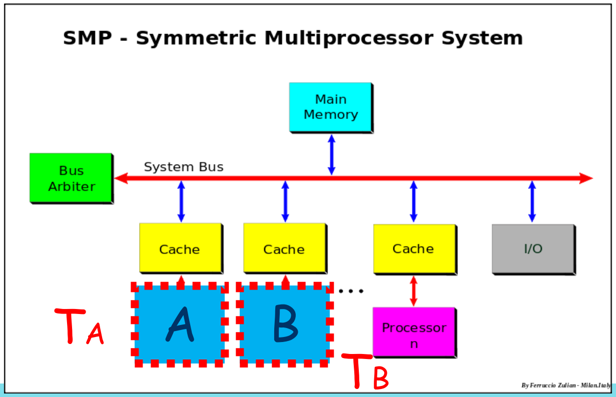


How much can we benefit from SMP?

- For two task A & B without data dependency
  - in theorey: $t_A = T_A$, $t_B = T_B$
  - in fact: $t_A < T_A$, $t_B < T_B$
    - or even $t_A + t_B < \min(T_A, T_B)$







Amdahl’s Law

http://en.wikipedia.org/wiki/Amdahl's_law

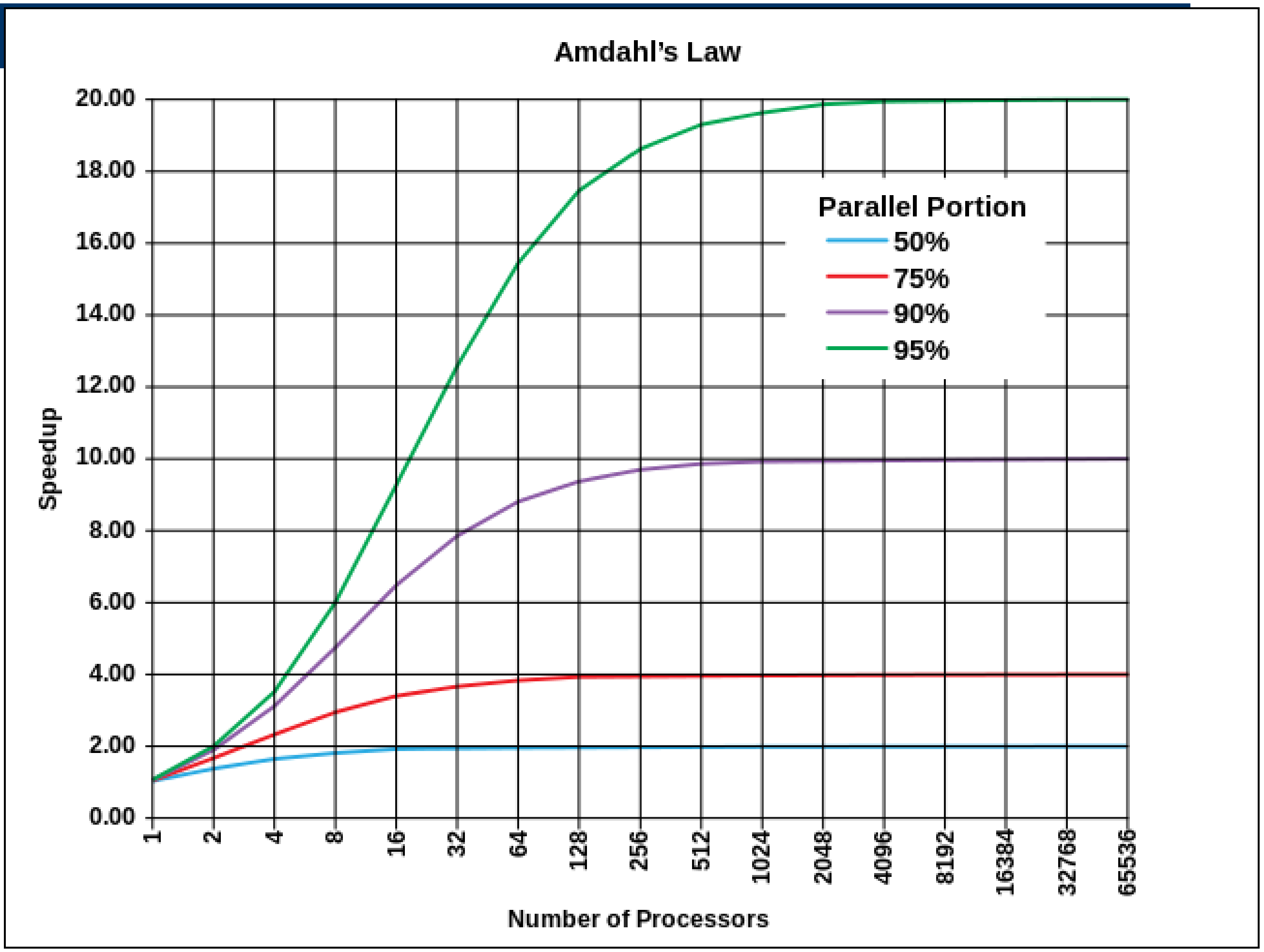
- For one task wants to benefit from SMP by multi-threading:
  - B = the fraction of the task can only be serialized
  - T(n) = the time to finish the task with n threads

$$T(n) = T(1)\left(B + \frac{1}{n}(1 - B)\right)$$

  - example: T(1) = 20s, B = 5%
    - then T(n) > T(infinity) = 1s ➔ 20x faster
  - T(n) is the lower bound in theory
    - even worse in practise

Almost NO linear benefit

http://en.wikipedia.org/wiki/Amdahl's_law



In theory

- conclusion from Amdahl’s Law
  - How much a task can be parallelized depends the task itself.
- parallelized without any serialization
  - vector addition
- parallelized with centain serialization
  - vector summation
- hard to parallelize at all
  - Hanoi tower

In practise

- What makes us far from the lower bound?
  - data race
  - data dependency
  - the cost from multi-threading
  - the cost from SMP/multi-core system
- even sometime perform worse than the serialization version on uni-core system
- better parallel algorithm
  - how to compute π?

1. Cost from multi-threading

- Thread creation/destruction has its own cost.
  - more than you do in OS lab

```
#define N 10000000
#define NR_THREAD N
int a[N], b[N], c[N];
void add(int tid) { c[tid] = a[tid] + b[tid]; }
```

- Amdahl's Law + cost from multi-threading
  - small scale parallelization (2-8 threads)

2. Data race

- This is exactly the “critical region” problem.
- goal: provide exclusive accessing
- Disabling interrupt does not work in SMP/multi-core system.

- solution: atomic instructions
- atomic inst. ➔ locking ➔high-level mutex

Atomic instructions

- provide exclusive accessing at ISA level
- RMW (read-modify-write)
  - Test&Set
  - Compare&Swap
  - load-link / store-condition
- take producer-consumer problem as an example

Test&Set

Computer Architecture, Princeton University

- blocking mutex

```
P:      Test&Set (mutex),$R_{temp}$
        if ($R_{temp}$!=0) goto P
        Load $R_{head}$, (head)
spin:   Load $R_{tail}$, (tail)
        if $R_{head}$==$R_{tail}$ goto spin
        Load R, ($R_{head}$)
        $R_{head}$=$R_{head}$+1
        Store $R_{head}$, (head)
V:      Store 0, (mutex)
        process(R)
```

Critical Section

```
Test&Set(addr, reg):
    R[reg] = M[addr];
    if (R[reg] == 0)
        M[addr] = 1;
```

Compare&Swap

Computer Architecture, Princeton University

- non-blocking mutex
- "ABA" problem

```
Compare&Swap (addr, rt, rs):
    if(R[rt] == M[addr])
        M[addr] = R[rs];
        R[rs] = R[rt];
        status = success;
    else
        status = fail;
```

```
try:    Load R_head, (head)
spin:   Load R_tail, (tail)
        if R_head==R_tail goto spin
        Load R, (R_head)
        R_newhead = R_head+1
        Compare&Swap(head), R_head, R_newhead
        if (status==fail) goto try
        process(R)
```

Implement atomic instructions

- lock the system bus
  - when the system bus is locked, no other memory accessing is allowed
- This may slow down the performance in SMP/multi-core system.
- Any better idea?

Load-link / Store-condition

Computer Architecture, Princeton University

```
try:    Load-link $R_{head}$, (head)
spin:   Load $R_{tail}$, (tail)
        if $R_{head}$==$R_{tail}$ goto spin
        Load R, ($R_{head}$)
        $R_{head}$ = $R_{head}$ + 1
        Store-conditional $R_{head}$, (head)
        if (status==fail) goto try
        process(R)
```

```
Store-condition(addr, reg):
    if(<flag, m> == <1, addr>)
        cancel other processors' reservation on addr;
        M[addr] = R[reg];
        status = success;
    else
        status = fail;
```

```
Load-link(reg, addr):
    <flag, m> = <1, addr>;
    R[reg] = M[addr];
```

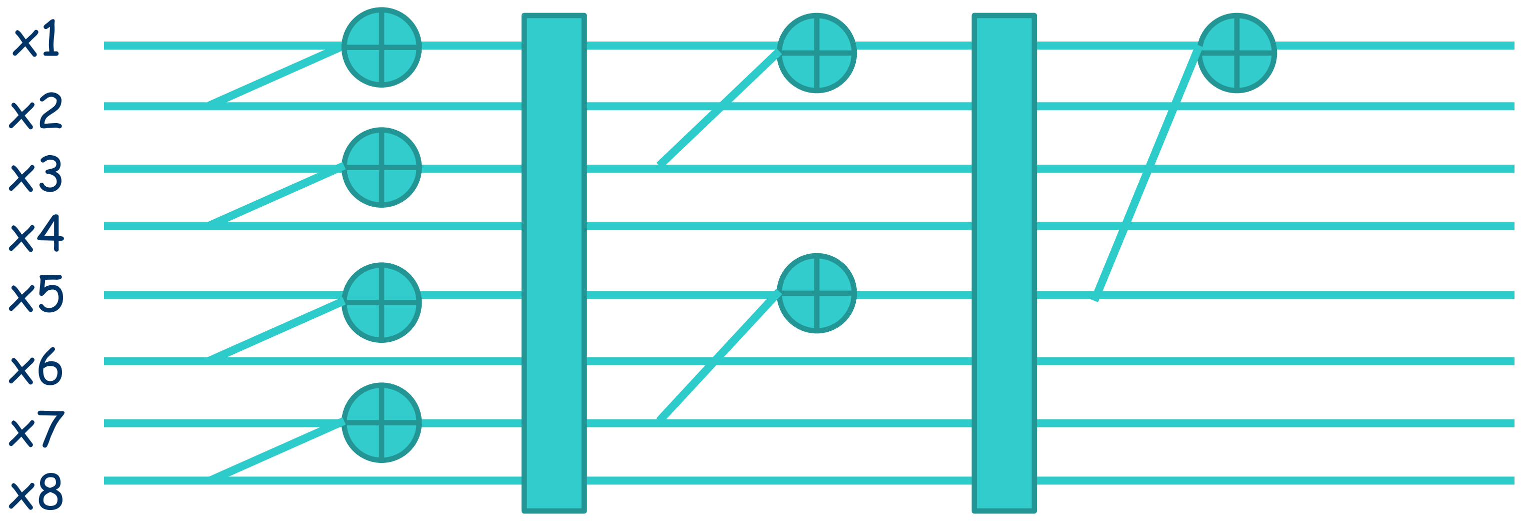
Load-link / Store-condition

- non-blocking mutex
- avoid “ABA” problem
- modification can be arbitrary
  - however, more complex modification -> higher probability to fail for store-condition

- see “MIPS32 instruction set” for more details

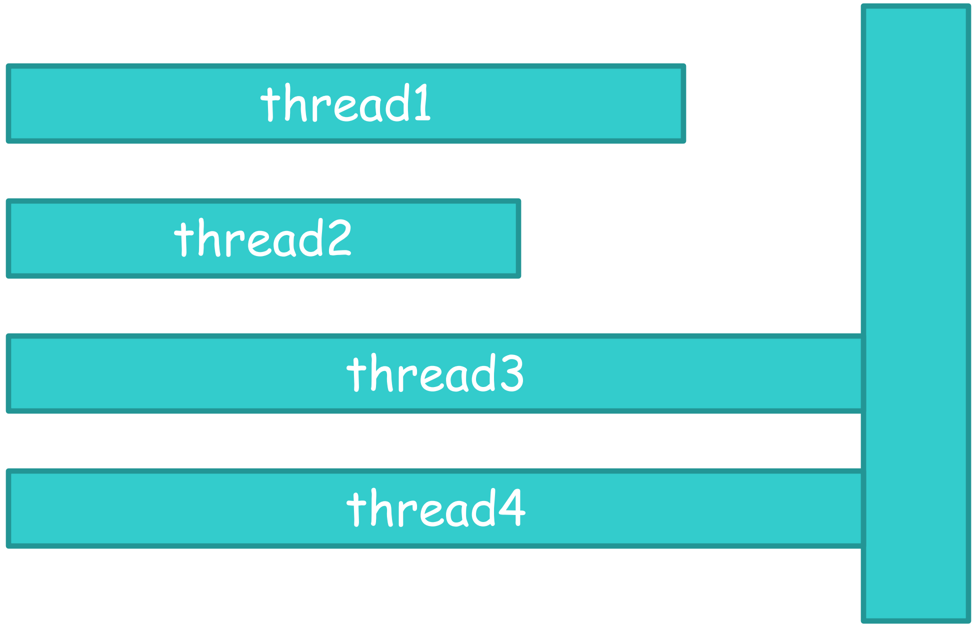
3. Data dependency

- Thread A needs the result from thread B to perform computation.
- Synchronization is needed.



Synchronization

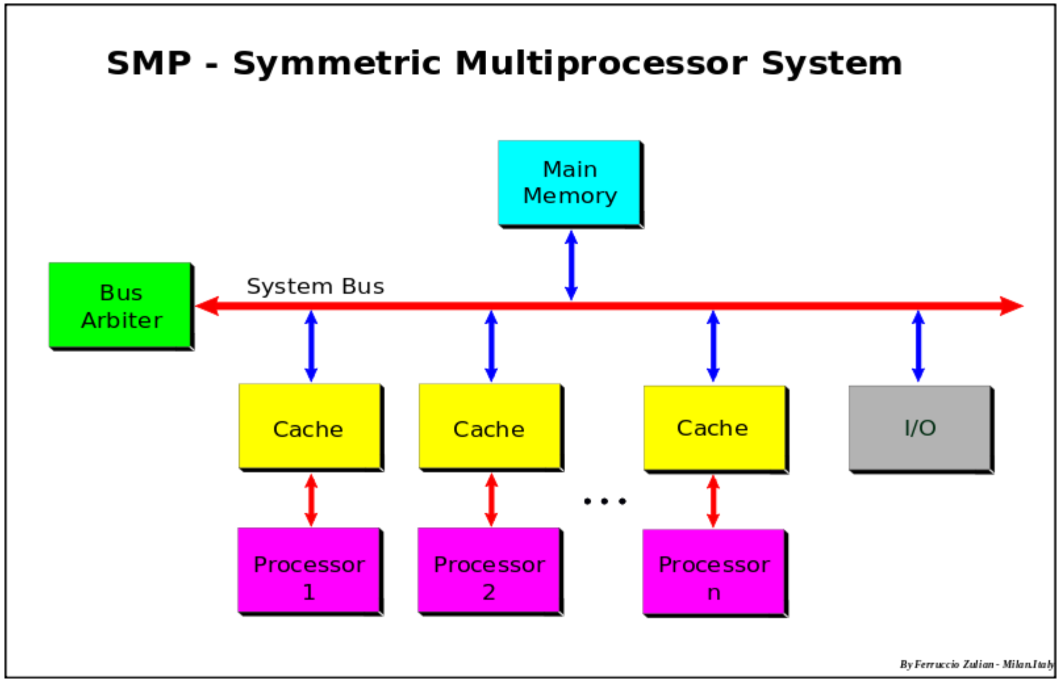
- wait until other threads are ready



- over-synchronization

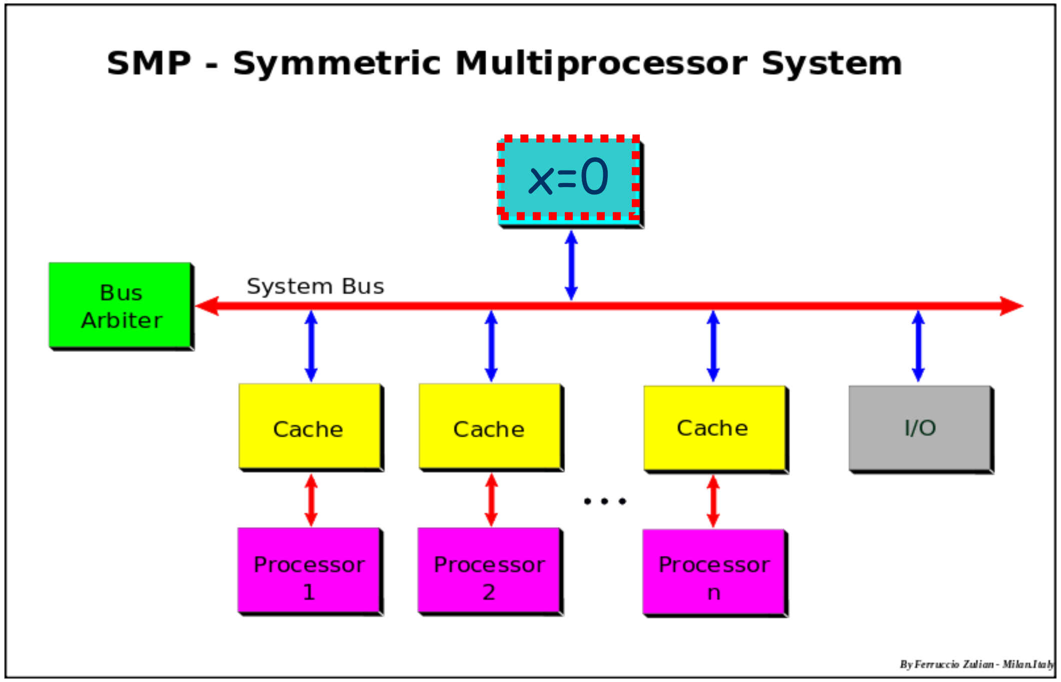
4. Cost from SMP/multi-core system

- Any problem?



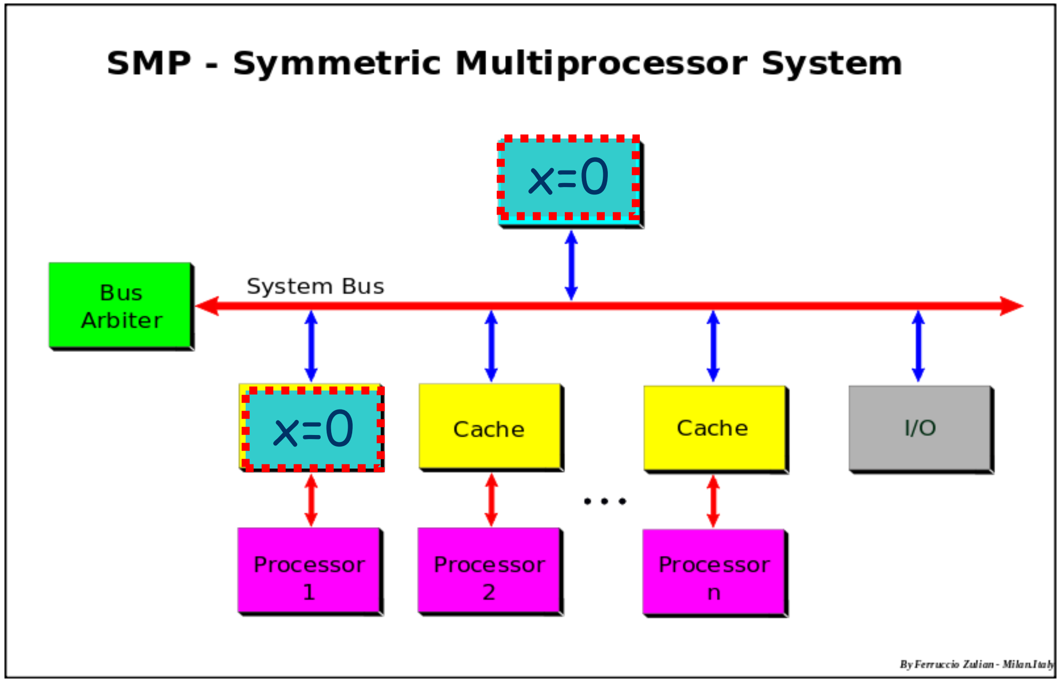
4. Cost from SMP/multi-core system

- Any problem?
  - initially x = 0 in memory



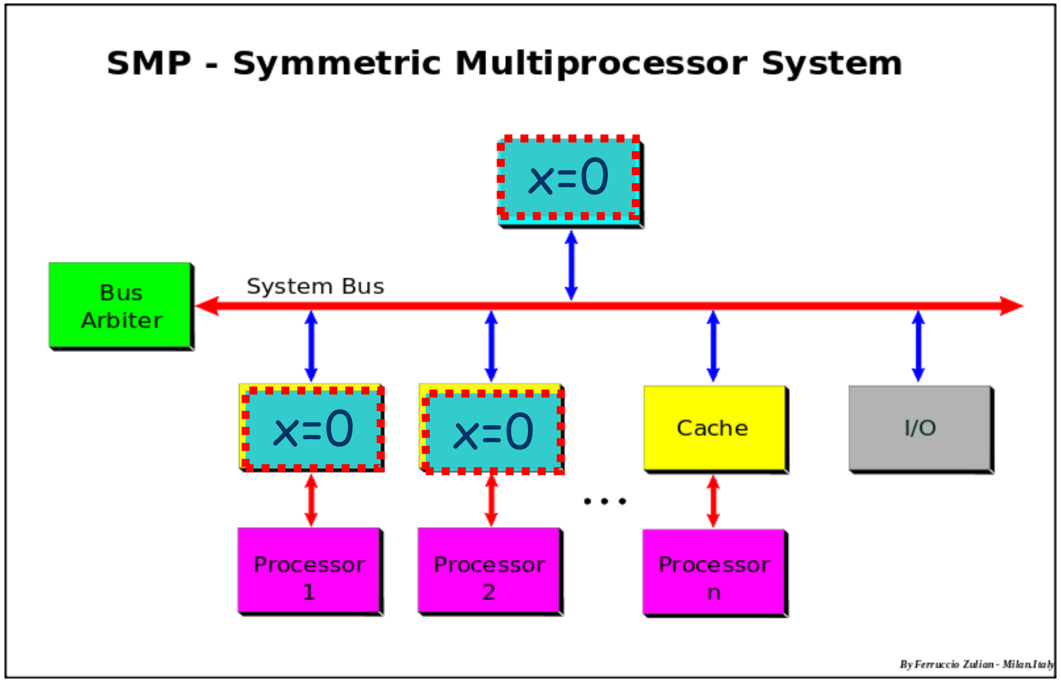
4. Cost from SMP/multi-core system

- Any problem?
  - initially x = 0 in memory
  - P1: read x



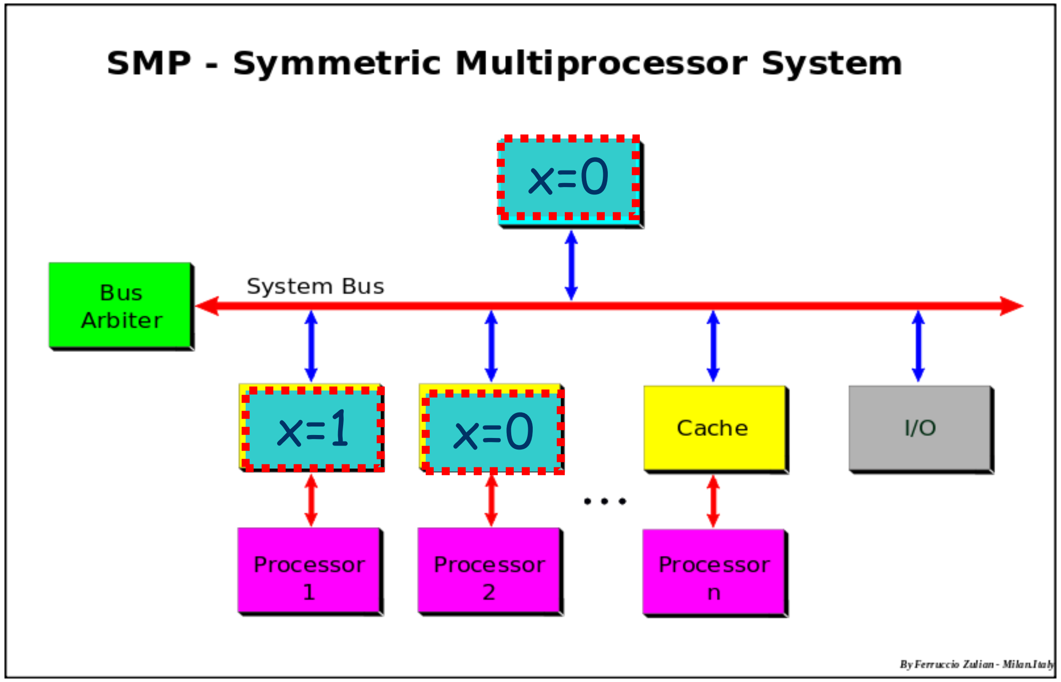
4. Cost from SMP/multi-core system

- Any problem?
  - initially x = 0 in memory
  - P1: read x
  - P2: read x



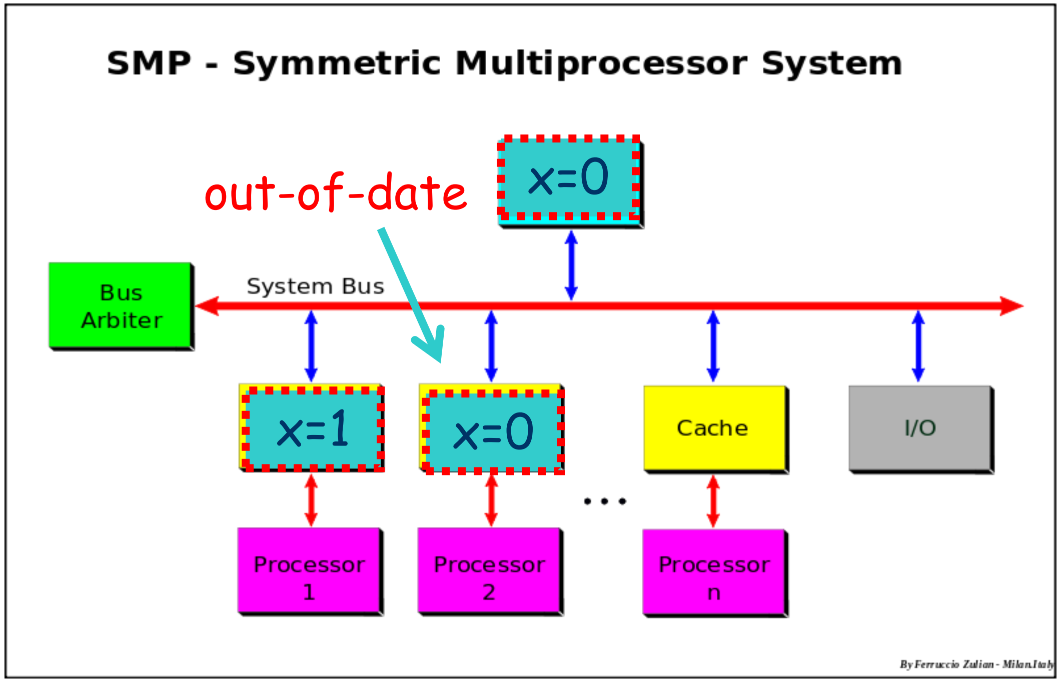
4. Cost from SMP/multi-core system

- Any problem?
  - initially x = 0 in memory
  - P1: read x
  - P2: read x
  - P1: write x = 1



4. Cost from SMP/multi-core system

- Any problem?
  - initially x = 0 in memory
  - P1: read x
  - P2: read x
  - P1: write x = 1
  - P2: read x

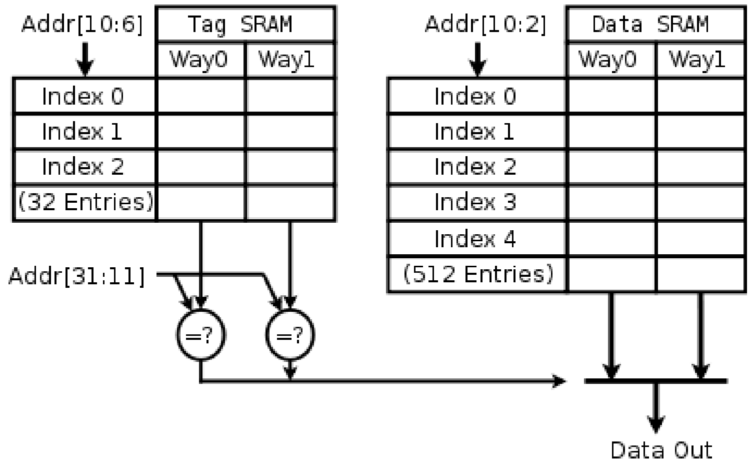


Cache coherence

- Cache of one processor is invisible for other processors.
  - data may be out-of-date
  - where the valid locates is not known
- Mechanism is needed for different caches to communicate.
  - at least notify others about updating
  - snooping (broadcast)

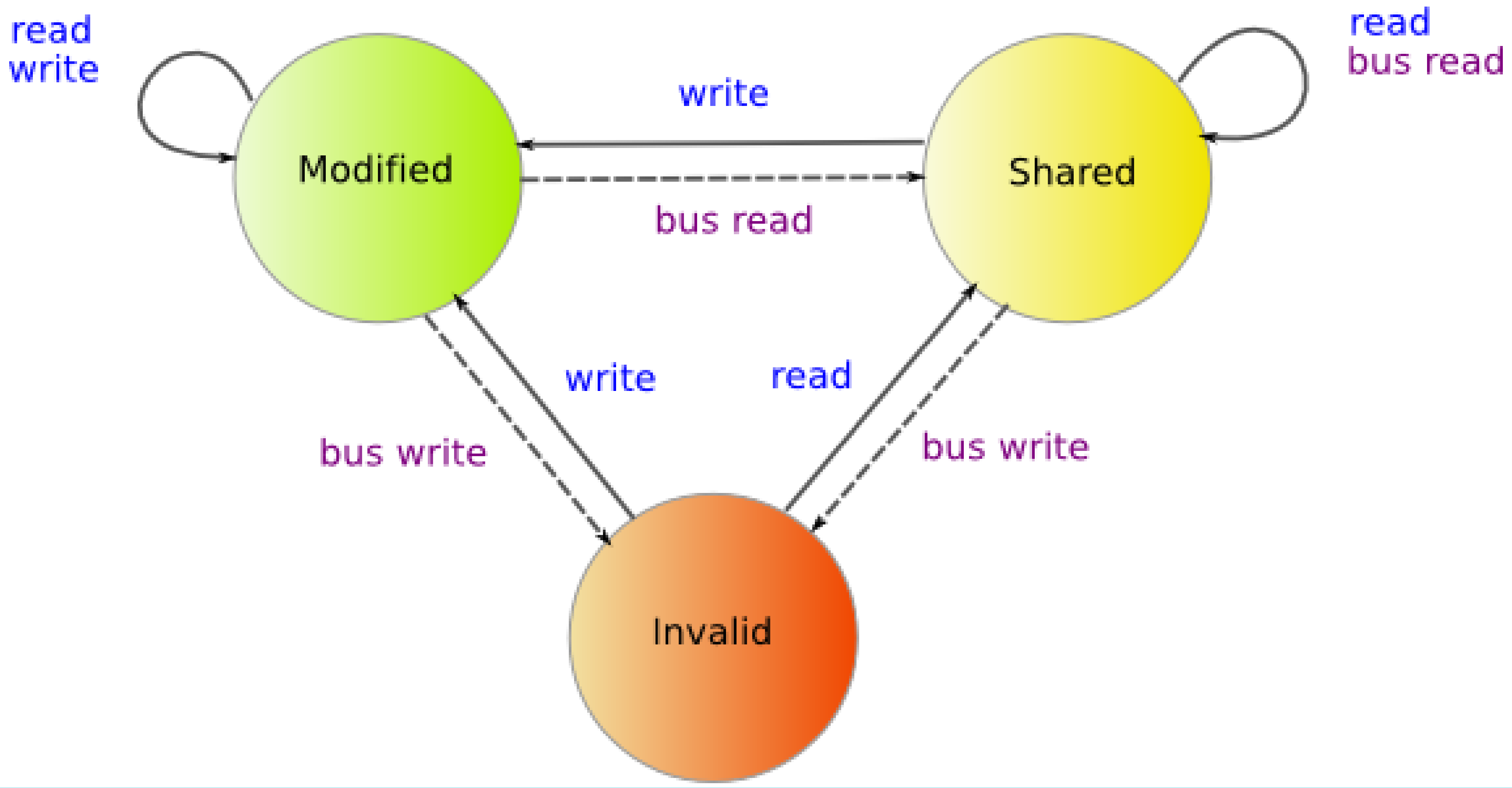
Cache coherence protocol - MSI

- For each cache line, maintain its state bits
  - M(modified): the data is dirty, needs writing back in the future
  - S(shared): the data is clean, and is shared by other caches
  - I(invalid): the data is invalid



4KB, 2-way set-associative 64B line cache read path

Cache coherence protocol - MSI



Performance

- To guarantee cache conherence, threads in SMP/multi-core system may perfrom badly.
  - Share data misses often dominate cache behavior even though they may only be 10% to 40% of the data accesses.
- Shared memory has little locality.
  - Why?

Memory ordering

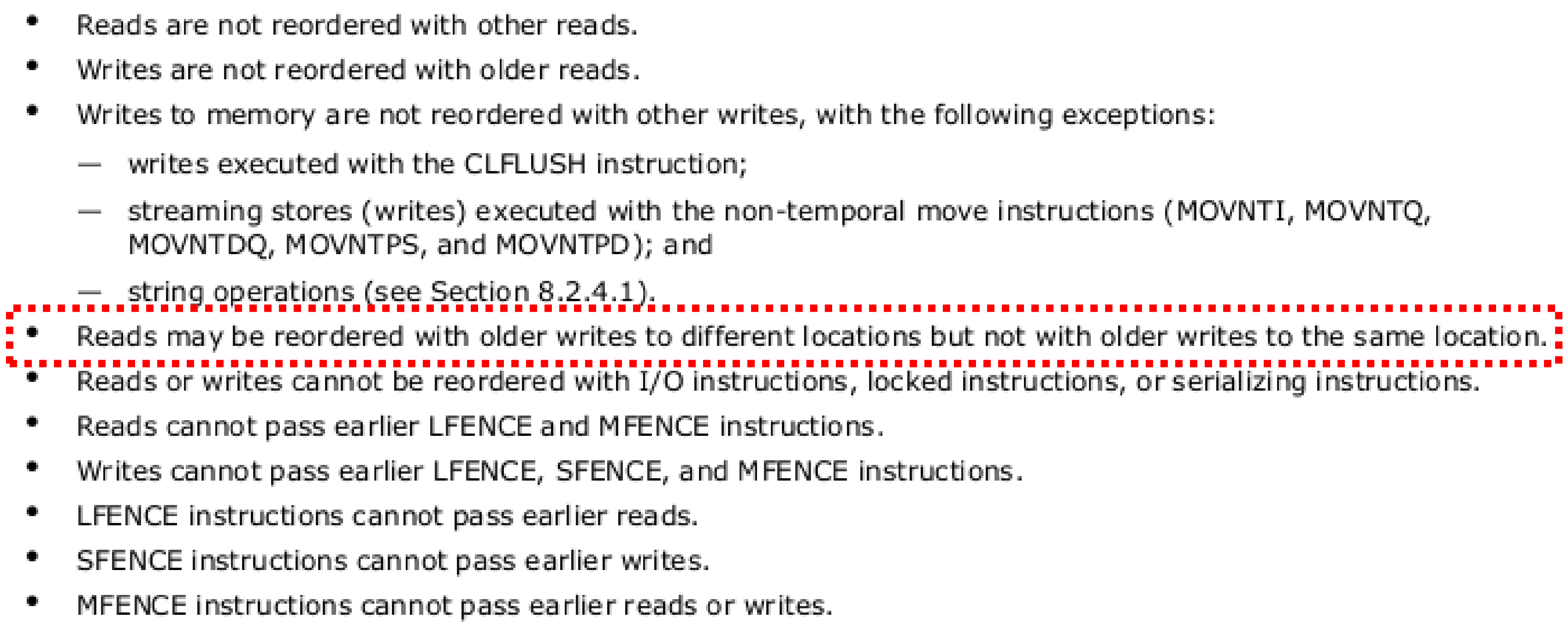
- let x = y = 0 initially

Processor 1	Processor 2
movl $1, (x)	movl $1, (y)
movl (y), %eax	movl (x), %ebx

- What will the result be?
- Is "%eax = %ebx = 0" possible?

Memory ordering

Intel® 64 and IA-32 Architectures Software Developer's Manual



- Reads are not reordered with other reads.
- Writes are not reordered with older reads.
- Writes to memory are not reordered with other writes, with the following exceptions:
  - — writes executed with the CLFLUSH instruction;
  - — streaming stores (writes) executed with the non-temporal move instructions (MOVNTI, MOVNTQ, MOVNTDQ, MOVNTPS, and MOVNTPD); and
  - — string operations (see Section 8.2.4.1).
- Reads may be reordered with older writes to different locations but not with older writes to the same location.
- Reads or writes cannot be reordered with I/O instructions, locked instructions, or serializing instructions.
- Reads cannot pass earlier LFENCE and MFENCE instructions.
- Writes cannot pass earlier LFENCE, SFENCE, and MFENCE instructions.
- LFENCE instructions cannot pass earlier reads.
- SFENCE instructions cannot pass earlier writes.
- MFENCE instructions cannot pass earlier reads or writes.

Memory ordering

Processor 1	Processor 2
movl $1, (x)	movl $1, (y)
movl (y), %eax	movl (x), %ebx

⇩

Processor 1	Processor 2
movl $1, (x)	movl (x), %ebx
movl (y), %eax	movl $1, (y)

- How to fix?

Memory ordering

- memory barrier (memory fency)
  - force serialization before and after the barrier instructions
  - lfence, sfence, mfence
- Before any memory operation starts after the barrier instruction, all memory operations before it must finish.
- cost
  - more instruction dependency, less performance

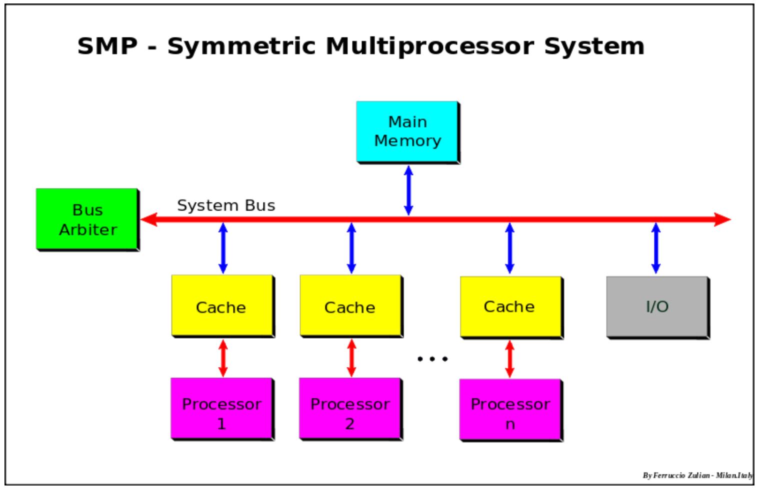
Summary

- SMP/multi-core system sounds attractive.
- But in practise, some performance is lost to guarantee the correctness.
  - atomic instructions, cache coherence, memory barrier...
  - more than you expect!
- In computer science, trade-off is everywhere.
- How we can benefit more from SMP/multi-core system is still in research.

- Message passing in SMP

Shared memory

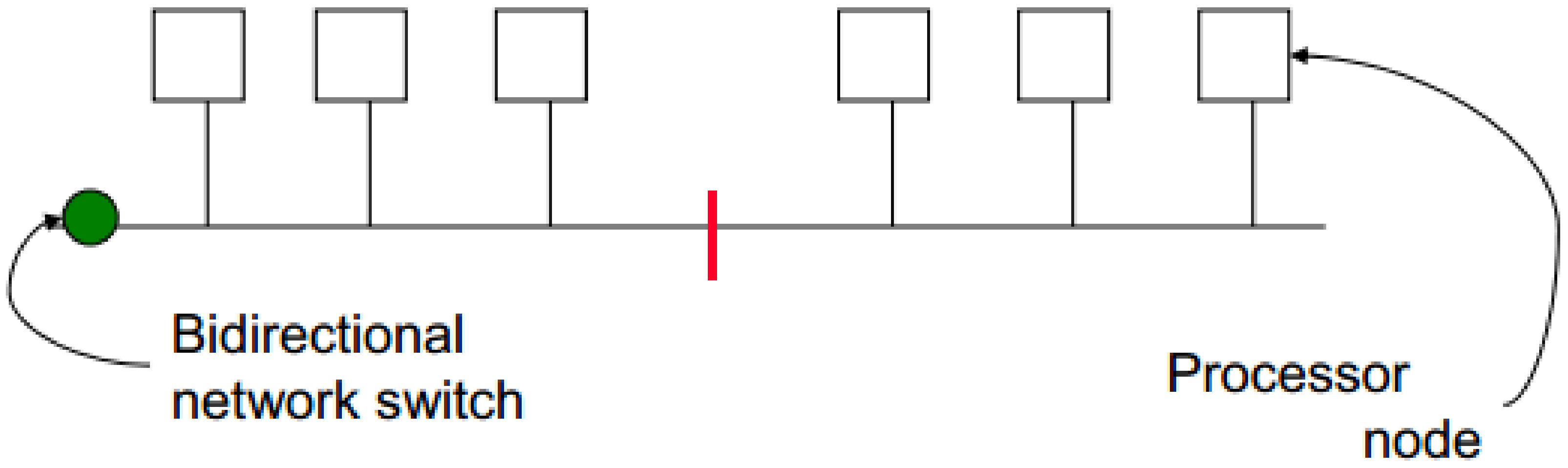
- This is the architecture we use in PC.
  - processors connected by a bus
  - a global memory shared by all processors
    - also connected on a bus
- All communications must go through the bus.
  - more processors ➔ bottleneck



Architecture based on message passing

- Processors connected to switches.
- To communicate, send messages to switches.
- Switches perform forwarding & routing.
- Each processor has its local memory.
  - usually no global memory
  - can use as unified/separate address space(s)
- Yes! This is a network!
  - Interconnection Network
  - NoC (Network on Chip)

Bus IN



Network on Chip

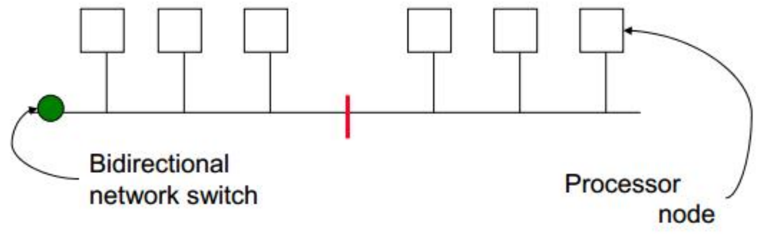
- Now the topic turns to network...
  - How to increase
    - throughput, reliability?
  - How to decrease
    - latency, bottleneck, congestion, design cost, energy cost?

- Trade-off is everywhere.

Topology

[Lecture notes based on slides created by Mary Jane Irwin, and Patterson & Hennessy “Computer Organization and Design”]

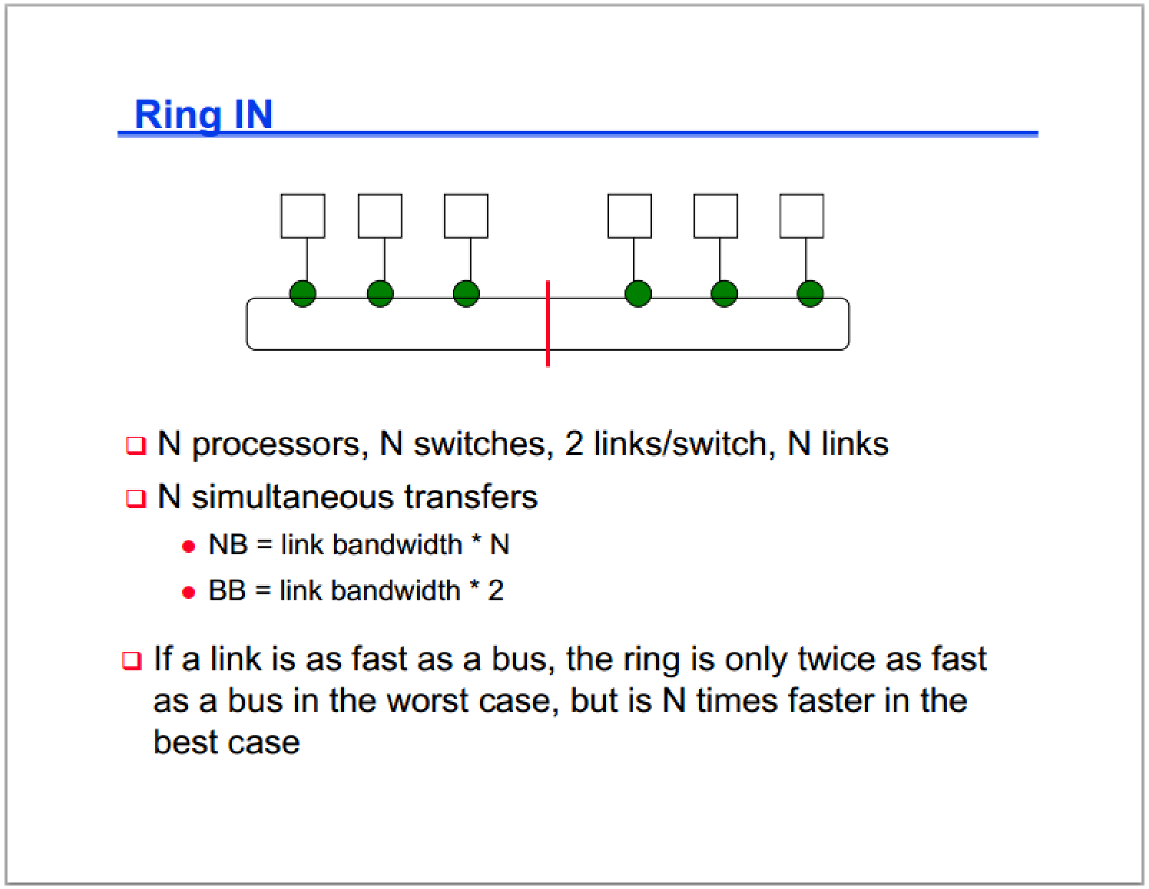
Bus IN



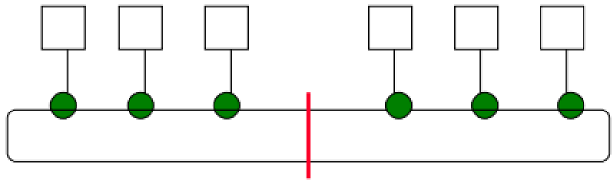
- N processors, 1 switch (●), 1 link (the bus)
- Only 1 simultaneous transfer at a time
  - NB = link (bus) bandwidth * 1
  - BB = link (bus) bandwidth * 1

Topology

[Lecture notes based on slides created by Mary Jane Irwin, and Patterson & Hennessy "Computer Organization and Design"]



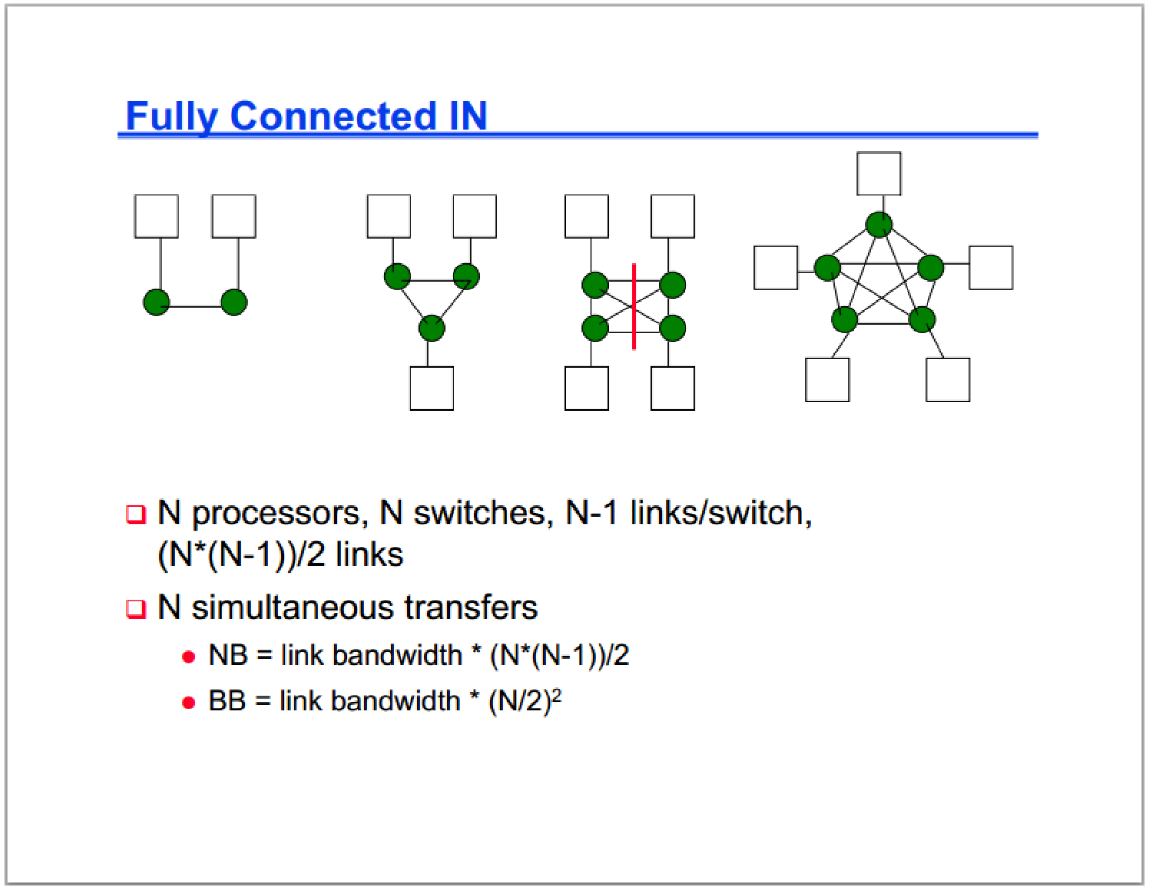
Ring IN



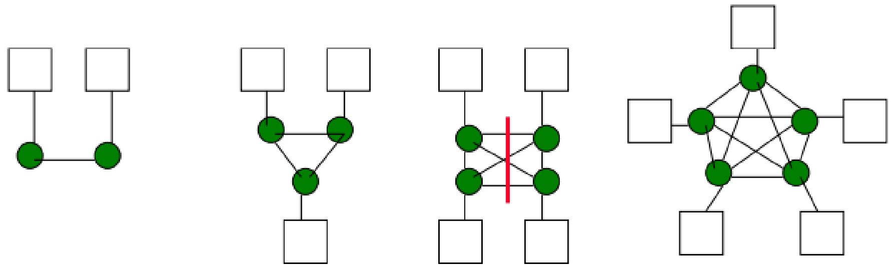
- N processors, N switches, 2 links/switch, N links
- N simultaneous transfers
  - NB = link bandwidth * N
  - BB = link bandwidth * 2
- If a link is as fast as a bus, the ring is only twice as fast as a bus in the worst case, but is N times faster in the best case

Topology

[Lecture notes based on slides created by Mary Jane Irwin, and Patterson & Hennessy "Computer Organization and Design"]



Fully Connected IN

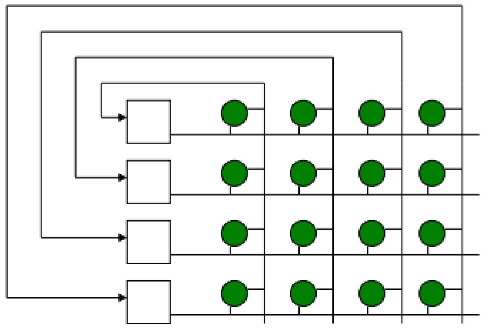


- N processors, N switches, N-1 links/switch, (N*(N-1))/2 links
- N simultaneous transfers
  - NB = link bandwidth * (N*(N-1))/2
  - BB = link bandwidth * $(N/2)^2$

Topology

[Lecture notes based on slides created by Mary Jane Irwin, and Patterson & Hennessy "Computer Organization and Design"]

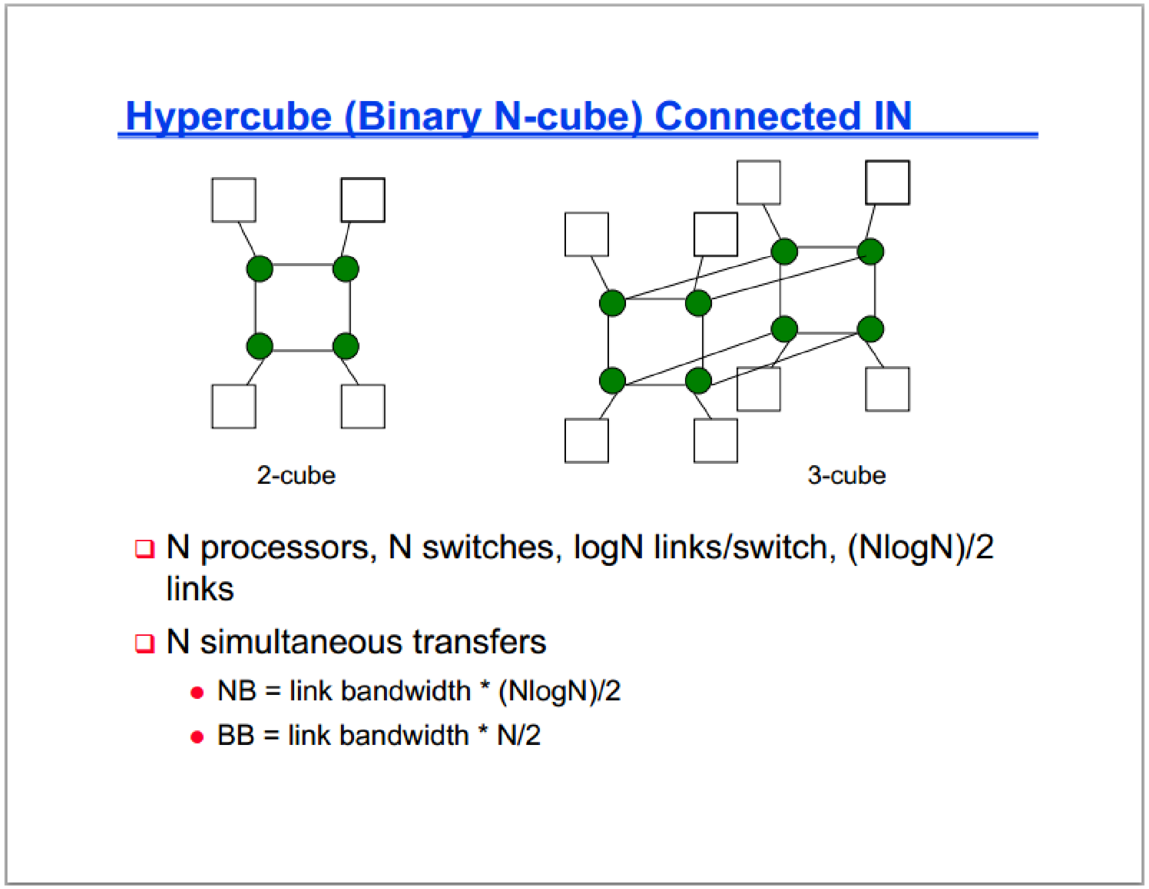
Crossbar (Xbar) Connected IN



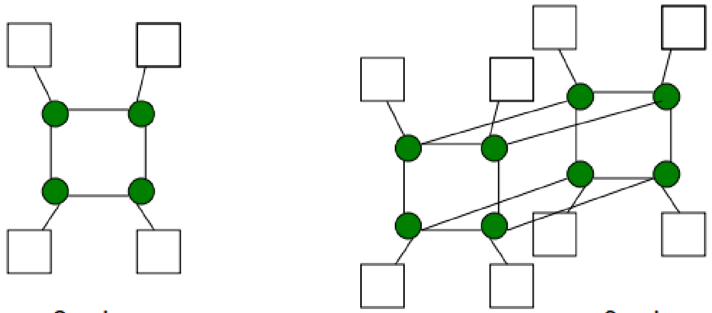
- N processors, N^2 switches (unidirectional),2 links/switch, N^2 links
- N simultaneous transfers
  - NB = link bandwidth * N
  - BB = link bandwidth * N/2

Topology

[Lecture notes based on slides created by Mary Jane Irwin, and Patterson & Hennessy "Computer Organization and Design"]

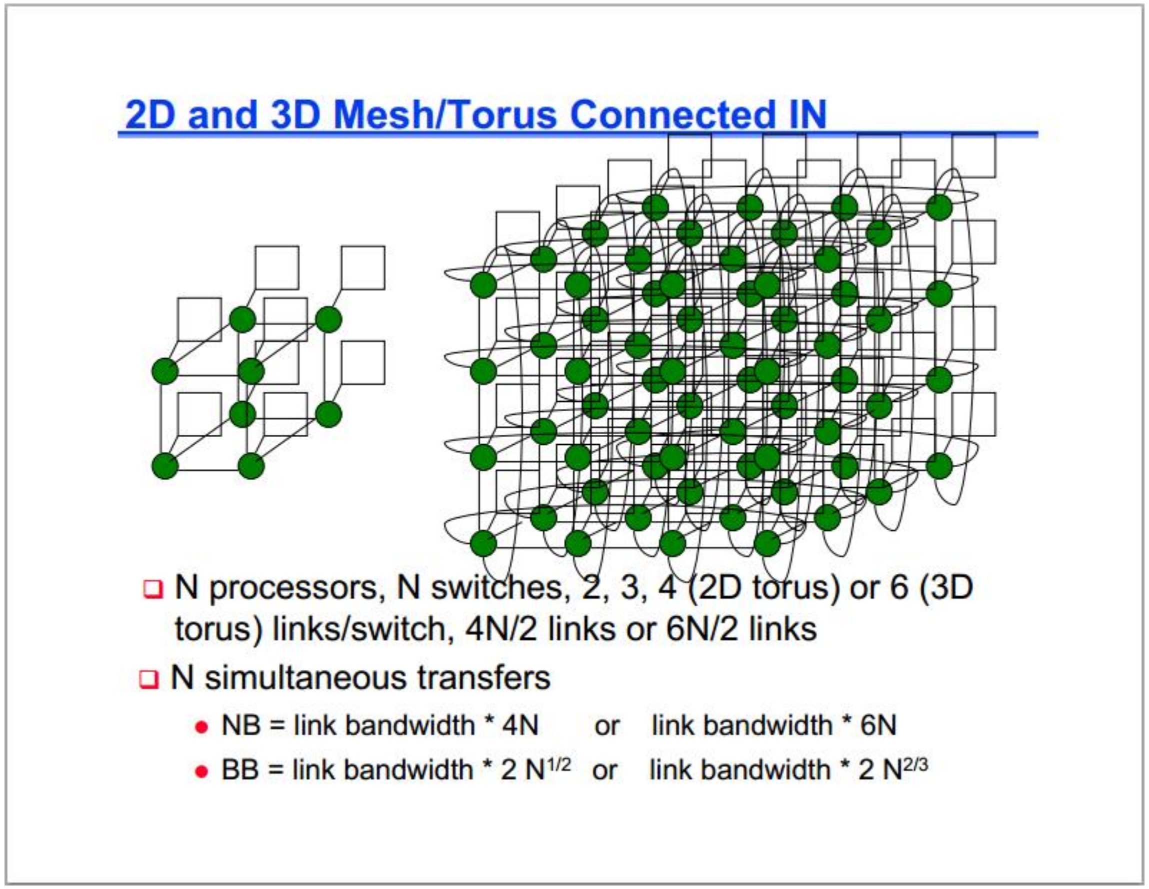


Hypercube (Binary N-cube) Connected IN

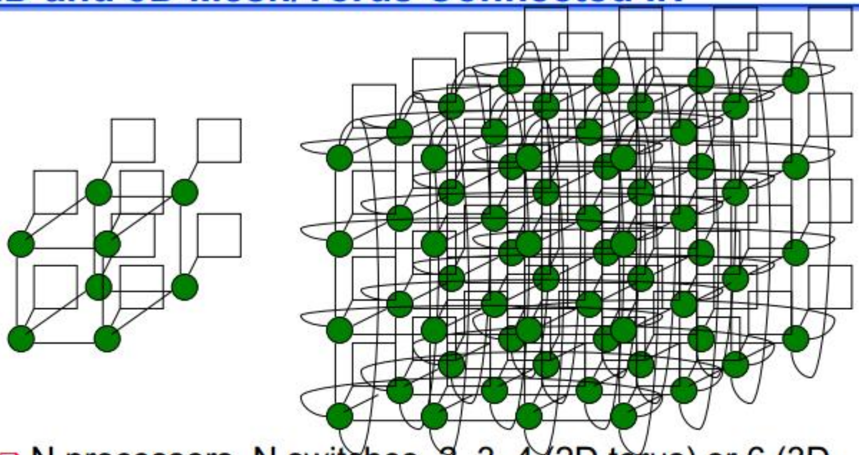


- N processors, N switches, logN links/switch, (NlogN)/2 links
- N simultaneous transfers
  - NB = link bandwidth * (NlogN)/2
  - BB = link bandwidth * N/2

Topology
[Lecture notes based on slides created by Mary Jane Irwin, and Patterson & Hennessy "Computer Organization and Design"]



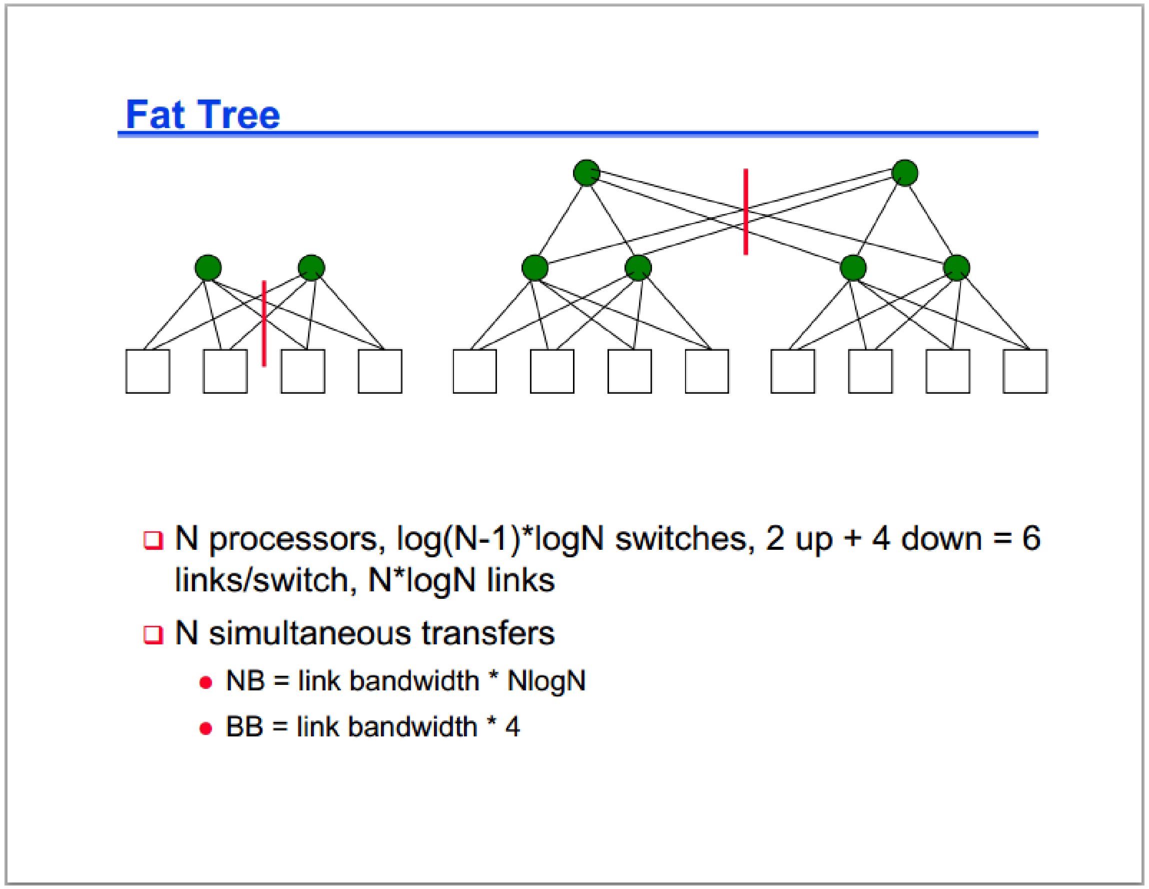
2D and 3D Mesh/Torus Connected IN



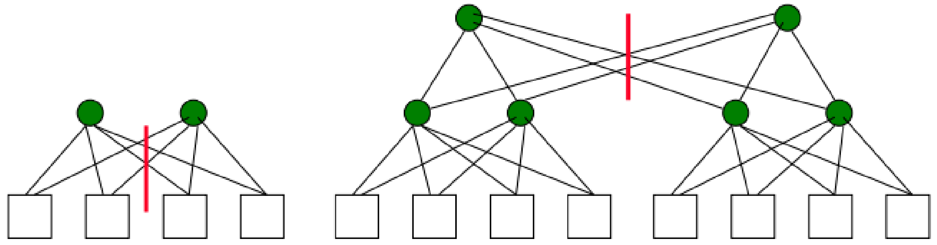
- N processors, N switches, 2, 3, 4 (2D torus) or 6 (3D torus) links/switch, 4N/2 links or 6N/2 links
- N simultaneous transfers
  - NB = link bandwidth * 4N or link bandwidth * 6N
  - BB = link bandwidth * 2 $N^{1/2}$ or link bandwidth * 2 $N^{2/3}$

Topology

[Lecture notes based on slides created by Mary Jane Irwin, and Patterson & Hennessy "Computer Organization and Design"]



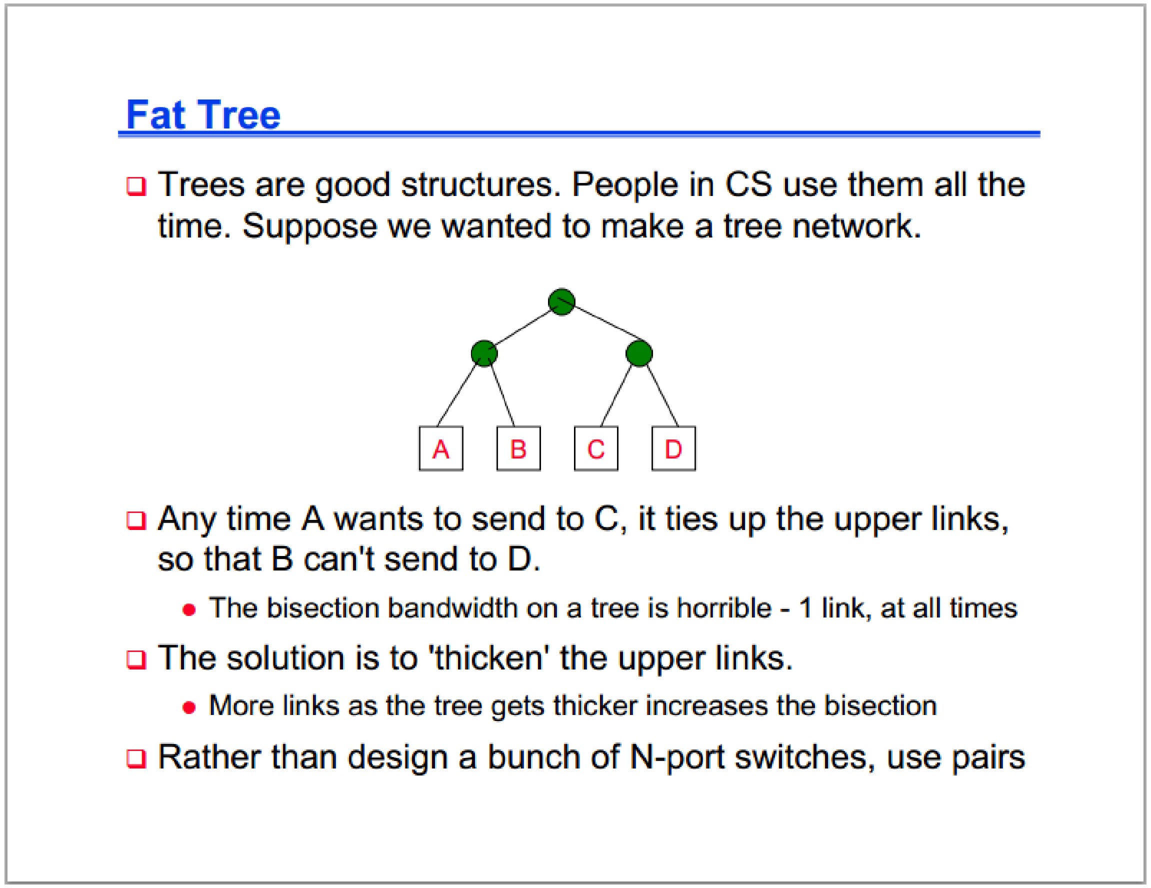
Fat Tree



- N processors, log(N-1)*logN switches, 2 up + 4 down = 6 links/switch, N*logN links
- N simultaneous transfers
  - NB = link bandwidth * NlogN
  - BB = link bandwidth * 4

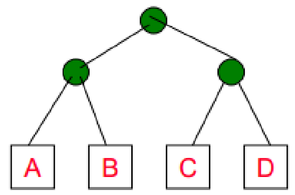
Topology

[Lecture notes based on slides created by Mary Jane Irwin, and Patterson & Hennessy “Computer Organization and Design”]



Fat Tree

- Trees are good structures. People in CS use them all the time. Suppose we wanted to make a tree network.

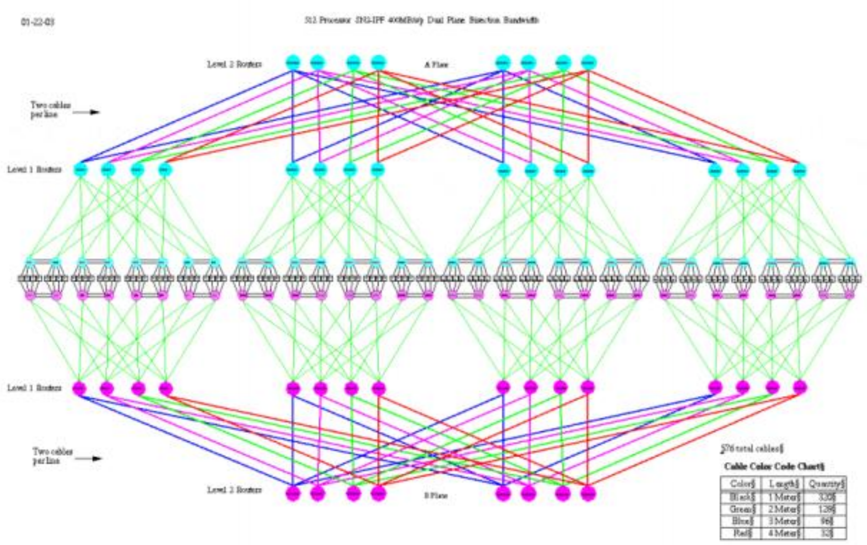


- Any time A wants to send to C, it ties up the upper links, so that B can't send to D.
  - The bisection bandwidth on a tree is horrible - 1 link, at all times
- The solution is to 'thicken' the upper links.
  - More links as the tree gets thicker increases the bisection
- Rather than design a bunch of N-port switches, use pairs

Topology

[Lecture notes based on slides created by Mary Jane Irwin, and Patterson & Hennessy “Computer Organization and Design”]

SGI NUMAlink Fat Tree



576 total cables

Cable Color Code Chart

Color	Length	Quantity
Black	1 Meter	320
Green	2 Meter	128
Blue	3 Meter	96
Red	4 Meter	32

Figure 2. 512-Processor Dual “Fat-Tree” Interconnect Topology

www.embedded-computing.com/articles/woodacre

Cache coherence

- For unified memory address space, we should still maintain cache coherence.
- Protocol based on snooping does not work.
  - Broadcast has a very high cost.
- directory-base protocols
  - each entry of the directory presents the state of a memory block
    - who has the copy, whether it is dirty…
  - ask the directory control for information

Shared memory v.s. message passing

- shared memory
  - memory is shared
  - communicate with load & store
  - has issue of critical region
  - synchronization with explict instruction
- message passing
  - memory is privade
  - communicate with send & receive
  - no issue of critial region
  - synchronization is implict