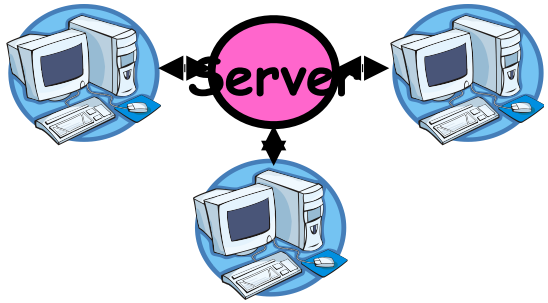
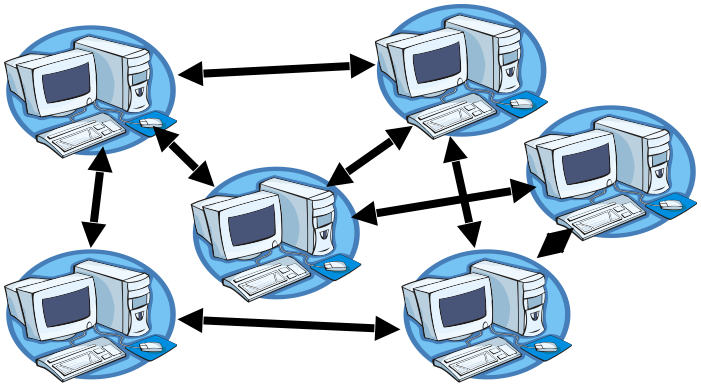
Centralized vs Distributed Systems



Client/Server Model



Peer-to-Peer Model

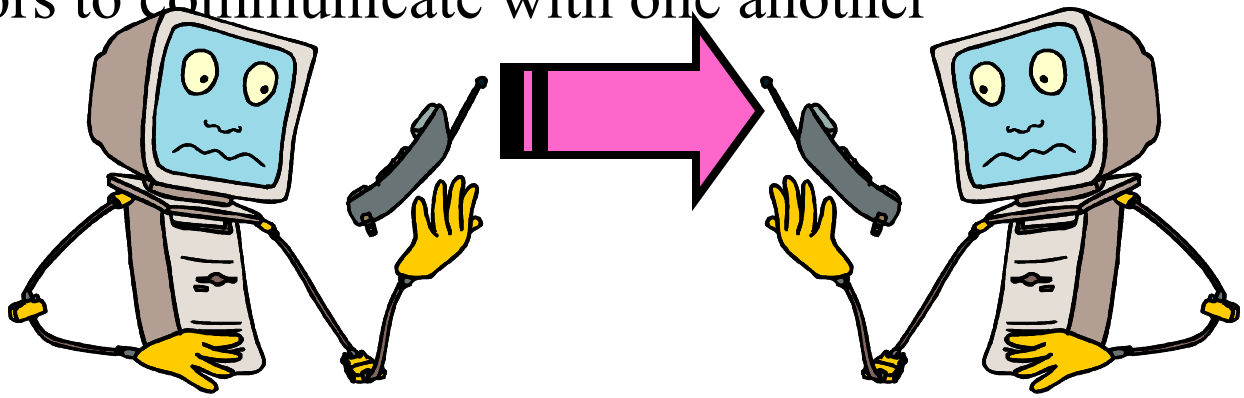
- Centralized System: System in which major functions are performed by a single physical computer
  - Originally, everything on single computer
  - Later: client/server model
- Distributed System: physically separate computers working together on some task
  - Early model: multiple servers working together
    - Probably in the same room or building
    - Often called a "cluster"
  - Later models: peer-to-peer/wide-spread collaboration

Distributed Systems: Motivation/Issues

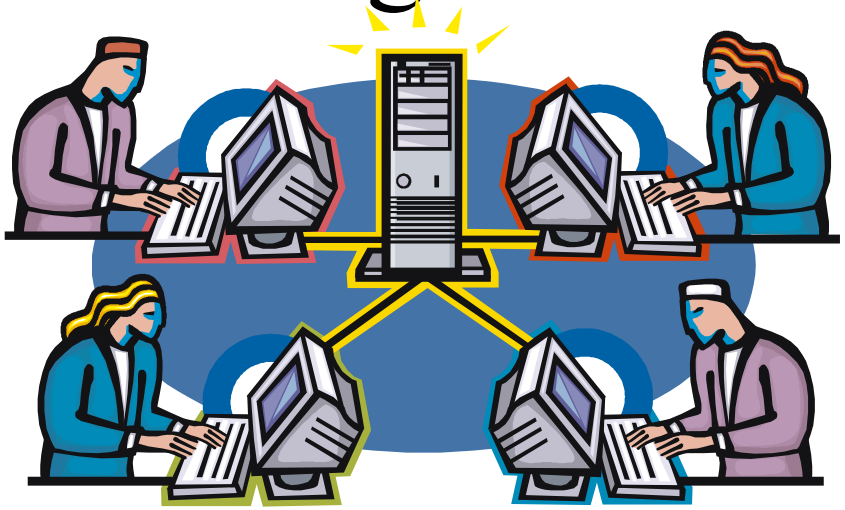
- Why do we want distributed systems?
  - Cheaper and easier to build lots of simple computers
  - Easier to add power incrementally
  - Users can have complete control over some components
  - Collaboration: Much easier for users to collaborate through network resources (such as network file systems)
- The *promise* of distributed systems:
  - Higher availability: one machine goes down, use another
  - Better durability: store data in multiple locations
  - More security: each piece easier to make secure
- Reality has been disappointing
  - Worse availability: depend on every machine being up
    - Lamport: "a distributed system is one where I can't do work because some machine I've never heard of isn't working!"
  - Worse reliability: can lose data if any machine crashes
  - Worse security: anyone in world can break into system
- Coordination is more difficult
  - Must coordinate multiple copies of shared state information (using only a network)
  - What would be easy in a centralized system becomes a lot more difficult

Distributed Systems: Goals/Requirements

- Transparency: the ability of the system to mask its complexity behind a simple interface
- Possible transparencies:
  - Location: Can't tell where resources are located
  - Migration: Resources may move without the user knowing
  - Replication: Can't tell how many copies of resource exist
  - Concurrency: Can't tell how many users there are
  - Parallelism: System may speed up large jobs by spliting them into smaller pieces
  - Fault Tolerance: System may hide varoius things that go wrong in the system
- Transparency and collaboration require some way for different processors to communicate with one another



Networking Definitions



- Network: physical connection that allows two computers to communicate
- Packet: unit of transfer, sequence of bits carried over the network
  - Network carries packets from one CPU to another
  - Destination gets interrupt when packet arrives
- Protocol: agreement between two parties as to how information is to be transmitted

网络与分布式操作系统

由于网络操作系统与分布式操作系统所采用的技术大多是相通的，将它们放在一起介绍。

计算机网络

一、网络的概念：

计算机网络是利用通信设备和通信线路，将地理上分散而且有相对独立功能的多个

计算机系统，按照某种原则相互连接在一起构成的计算机体系，它是计算机技术和通信技术相结合的产物。

二、网络组成：

1、组成：独立计算机、通信处理机、通信线路。

2.结点**:**网络中的主机及所附带的外部设备**,**也叫站点。

三、网络分类:

（一）按网络覆盖的地理范围，可将网络分为局域网和广域网、城域网。

（二）按照入网计算机的统一性分为：同构网络和异构网络

1、同构网络：在分布式操作系统中常采用同构网络

因为进程的动态迁移要求迁出站点与迁入站点具有相同或兼容的硬件环境。

2、异构网络：由不同类型的机器所构成的计算机网络。

在大型网络操作系统中，常采用异构网络，因为它对入网机器的类型没有任何限制。

四、网络的拓扑:

网络系统中的各个站点在物理上的联结方式。每种拓扑结构各有优点、缺点，对拓扑结构的评估常用以下标准:

1、基本成本：将系统中各站点联结起来所花费的代 价。

2、通信成本：把一个信息由站点**A**传送到站点**B**的距离

3、可靠性：如果一个通信链或站点失效，是否影响基余站点之间的通信。

（一）全联通拓朴结构：每个站点都直接与其它站点相连，这种结构的代价是昂贵的，因系统中任两个站点之间必须有直接的通信链。

*基本成本高，按站点数成平方地增长。

*传送速度快，因任两站点间的信息传送仅涉及一条通信链

*可靠性高，因仅当所有通信链都失效时，系统割裂。

（二）部分互联结构:

1、仅在一部分站点之间存在通信链，因而基本成本较低。

2、通信速度慢，由消息的传递可能要经过几个中间站点。

3、可靠性较差。

（三）层次结构:

除根站点外，每个站点均有唯一的父节点和若干个子节点

1、基本成本低

2、通信时往往要涉及几个节点

3、除叶站点外，任何一个站点的失效将导致不连通

（四）星型结构:

1、基本成本与站点数成线性比例关系

2、通信成本较低因一个站点与另一个站点之间的通信至多仅需两步。

3、可靠性差:一方面中心站点可能成为系统的瓶颈，另一方面，中心站点一旦失效，网络瘫痪。

（五）环形结构

（六）总线结构

通信与协议

OSI中，各层协议主要功能：

（1）物理层：负责两个站点之间字位流的传输。

（2）链路层，负责提供传输错误的恢复功能

（3）网络层：负责将消息分解为传输单位，并选择路径。

（4）传输层：负责站点之间的消息传送。

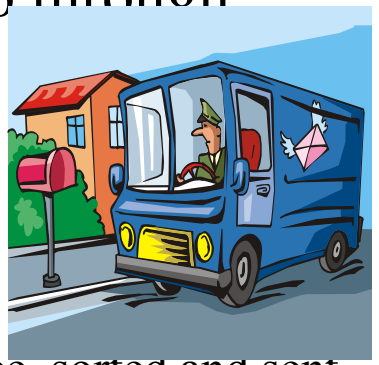
（5）会话层：负责进程间通信。

（6）表示层：负责数据转换。

（7）应用层：负责提供用户界面。

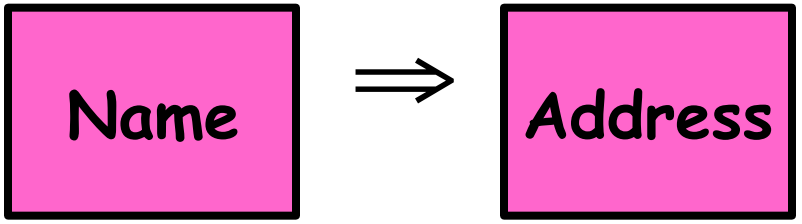
Routing

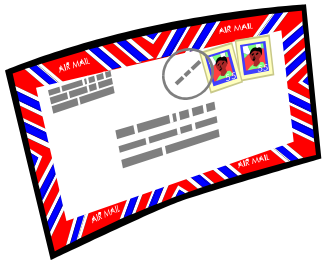
- Routing: the process of forwarding packets hop-by-hop through routers to reach their destination
  - Need more than just a destination address!
    - Need a path
  - Post Office Analogy:
    - Destination address on each letter is not sufficient to get it to the destination
    - To get a letter from here to Florida, must route to local post office, sorted and sent on plane to somewhere in Florida, be routed to post office, sorted and sent with carrier who knows where street and house is...



- Internet routing mechanism: routing tables
  - Each router does table lookup to decide which link to use to get packet closer to destination
  - Don't need 4 billion entries in table: routing is by subnet
  - Could packets be sent in a loop? Yes, if tables incorrect
- Routing table contains:
  - Destination address range → output link closer to destination
  - Default entry (for subnets without explicit entries)

Naming in the Internet



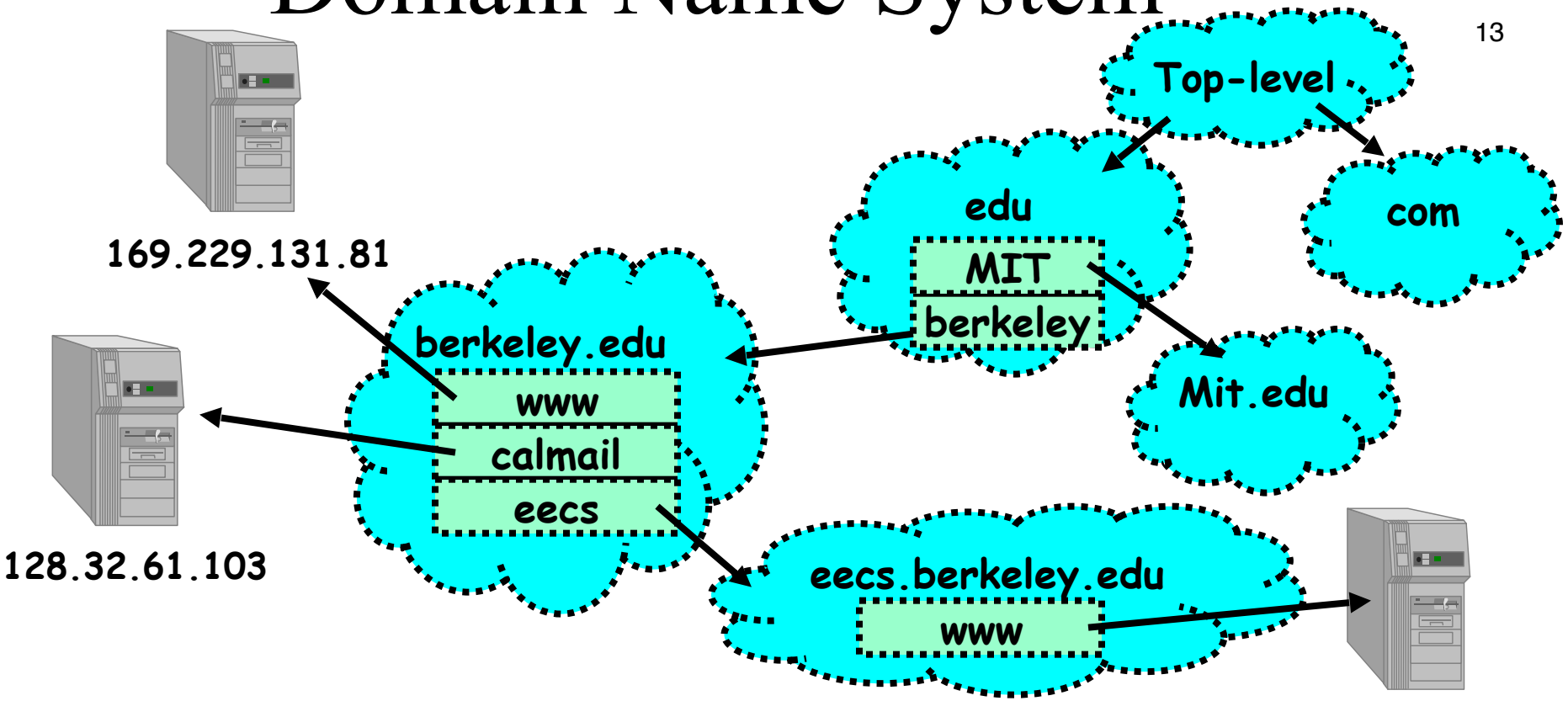


- How to map human-readable names to IP addresses?
  - E.g. www.berkeley.edu ⇒ 128.32.139.48
  - E.g. www.google.com ⇒ different addresses depending on location, and load
- Why is this necessary?
  - IP addresses are hard to remember
  - IP addresses change:
    - Say, Server 1 crashes gets replaced by Server 2
    - Or – google.com handled by different servers
- Mechanism: Domain Naming System (DNS)

Setting up Routing Tables

- How do you set up routing tables?
  - Internet has no centralized state!
    - No single machine knows entire topology
    - Topology constantly changing (faults, reconfiguration, etc)
  - Need dynamic algorithm that acquires routing tables
    - Ideally, have one entry per subnet or portion of address
    - Could have "default" routes that send packets for unknown subnets to a different router that has more information
- Possible algorithm for acquiring routing table
  - Routing table has "cost" for each entry
    - Includes number of hops to destination, congestion, etc.
    - Entries for unknown subnets have infinite cost
  - Neighbors periodically exchange routing tables
    - If neighbor knows cheaper route to a subnet, replace your entry with neighbors entry (+1 for hop to neighbor)
- In reality:
  - Internet has networks of many different scales
  - Different algorithms run at different scales

Domain Name System



- DNS is a hierarchical mechanism for naming
  - Name divided in domains, right to left: www.eecs.berkeley.edu
- Each domain owned by a particular organization
  - Top level handled by ICANN (Internet Corporation for Assigned Numbers and Names)
  - Subsequent levels owned by organizations
- Resolution: series of queries to successive servers
- Caching: queries take time, so results cached for period of time

How Important is Correct Resolution?

- If attacker manages to give incorrect mapping:
  - Can get someone to route to server, thinking that they are routing to a different server
    - Get them to log into "bank" - give up username and password
- Is DNS Secure?
  - Definitely a weak link
    - What if "response" returned from different server than original query?
    - Get person to use incorrect IP address!
  - Attempt to avoid substitution attacks:
    - Query includes random number which must be returned
- July 2008 hole in DNS security located!
  - Dan Kaminsky (security researcher) discovered an attack that broke DNS globally
    - One person in an ISP convinced to load particular web page, then *all* users of that ISP end up pointing at wrong address
  - High profile, highly advertised need for patching DNS
    - Big press release, lots of mystery
    - Security researchers told no speculation until patches applied

计算机模型

一、数据迁移:

当处于站点**A**的用户想要存取驻留于站点**B**的数据（文件）时，系统有两种方式:

（**1**）将整个文件都传送给站点**A,**此后对文件的访问便成为局部的了，当用户**A**不再需要该文件时，它便被送回到站点**B**。

（**2**）仅将文件的一部分传送到**A**，如果以后还需要，再传送另一部分，当站点**A**用户不再需要该文件时，　　将其修改部分传送回站点**B**。

二、计算迁移

在某些环境中，迁移计算比迁移数据效果更好例如：设站点**A**处的进程**P**　要使用站点**B**处的文件，它不是将**B**处的文件取过来，而是执行一个远程过程调用，以　　调用一个在**B**点已定义好的过程，该过程可对**P**所需的文件进行适当计算，然后　将结果发送给进程**P**。

另一种方法是：进程**P**发一个消息到站点**B**，然后由站点**B**处的操作系统　创建一个代理进程**Q**，其功能是执行**P**所指定的任务，当**Q**完成使命后，通过消　息将结　果送给

三、作业迁移

当一个作业到达时，它可以全部或部分地在不同站点处执行，其优点是：

1、负载平衡：作业或作业步可以在网上分布以均衡工作负载，

2、计算加速：如果一个作业可以划分为若干子作业，这些子作业可以在不同站点处并行执行，则整个作业的处理时间能被缩短。

3、硬件优选：有些作业可能只适合于在专用处理机上运行，例如矩际求逆。

4、软件优选：有的作业可能需要某些站点处的特别软件，而该软件不适合迁移，或迁移开销比作业开销大。

四、进程迁移：

进程迁移是将正运行于某站点处理的进程迁移到另一站点，由于在迁移时刻进程已经在原站点运行了一段时间，迁移时不仅需要迁移其代码和　数据，还应迁移与进程有关的数据结构，即进程控制块，进程迁移的目　的是实现负载平衡。

互斥

为了解决网络和分布式系统中互斥问题，必须提供类似信号灯的同步机构。为简单起见，这里只讨论二值信号灯的实现（相当于锁），由于网络和分布式系统中的互斥所涉及的进程可能位于不同站点，它们之间没有公共内存，因此比较复杂，这里假设共有**n**个处理机，所有处理机依次编号为**1—N**，每个处理机中仅有一个进程，且进程与处理机具有相同编号。

一、集中方式：

1、基本思想：系统中有一个进程负责协调对于临界区的进入。每一个要求进入临界区的进程都必须发送一个请求给协调者进程，仅当收到协调者进程的回答信号后，它才能进入自己的临界区；当一个进程退出临界区时，也需发送一个释放信号给协调者进程，然后继续执行。

当收到一个请求消息时，协调者进程需考查是否有某些进程正在其临界区内，若无，协调者进程发送一个回答消息给请求进程，否则请求进程需排队等待，若协调者进程收到一个释放消息，则它给等待队列中的某一进程发送回答信号允许它进入其临界区。

2、特点：

（**1**）无死锁

二、分布方式:

1、方法：当一个进程**P**要进入其临界区时。它产生一个新的时间邮戳**TS**，并发送一个**Request(P,TS)**给所有其它进程，当某个进程接收到此消息时它可能立即回答（如果它当前不在其临界区内）；也可能延迟回答（如果它当前正在其临界区内）。一个收到系统中所有进程回答信号的进程可以进入它的临界区，当一个进程退出其临界区后，它需要给所有向它发来请求消息的进程发送回答消息。

2、进程作出立即回答或延迟回答的决定因素:

（**1**）如果进程正在它的临界区内，延迟回答

（**2**）如果一个进程不想进入特的临界区，立即回答**; (2)**如果一个进程想进入但尚未进入它的临界区，该进程考查所有保存的请求表，此表用于保存该进程已收到但尚未回答的消息，并将当前收到的**REQUEST(P,TS)**消息中的**TS**与该表中所有消息的**TS**作比较，如果这个**TS**比表中所有消息的**TS**都小,则立即回答进程**P**，否则**REQUEST**被加到等待表中

3、上述算法特性:

（**1**）可实现互斥

（**2**）确保无死锁

（**3**）无“饿死“情况

DISTRIBUTED COORDINATION

FULLY DISTRIBUTED APPROACH

Approach due to Lamport. These are the general properties for the method:

a) The general mechanism is for a process **P[i]** to send a request (with ID and time stamp) to **all** other processes.

b) When a process **P[j]** receives such a request, it may reply immediately or it may defer sending a reply back.

c) When responses are received from all processes, then **P[i]** can enter its Critical Section.

d) When **P[i]** exits its critical section, the process sends reply messages to all its deferred requests.

DISTRIBUTED COORDINATION

FULLY DISTRIBUTED APPROACH

The general rules for reply for processes receiving a request:

a) If **P[j]** receives a request, and **P[j]** process is in its critical section, defer (hold off) the response to **P[i].**

b) If **P[j]** receives a request,, and not in critical section, and doesn't want to get in, then reply immediately to **P[i].**

c) If **P[j]** wants to enter its critical section but has not yet entered it, then it compares its own timestamp **TS[j]** with the timestamp **TS[i]** from **T[i].**

d) If **TS[j] > TS[i],** then it sends a reply immediately to **P[i]. P[i]** asked first.

e) Otherwise the reply is deferred until after **P[j]** finishes its critical section.

DISTRIBUTED COORDINATION

The Fully Distributed Approach assures:

a) Mutual exclusion
b) Freedom from deadlock
c) Freedom from starvation, since entry to the critical section is scheduled according to the timestamp ordering. The timestamp ordering ensures that processes are served in a first-come, first-served order.
d) 2 X (n - 1) messages needed for each entry. This is the minimum number of required messages per critical-section entry when processes act independently and concurrently.

Problems with the method include:

a) Need to know identity of everyone in system.
b) Fails if anyone dies - must continually monitor the state of all processes.
c) Processes are always coming and going so it's hard to maintain current data.

死锁处理

传统系统中所用的死锁预防、避免，以及检测等算法的思想一般也适合于网络和分存式系统，只需做某些适当的修改。

例如：只要在系统事件之间定义一个全序，资源分配预防死锁技术就可用于网络和分布式环境中，也即，系统内所有资源被赋予一个唯一的编号，一个进程可以要求一个编号为I的资源，当且仅当它未占有编号比I更小的资源。只要确定系统中某个进程为银行家，由它保持执行银行家算法所必需的信息，并负责系统中资源的分配，则银行家算法也同样适合于网络和分布式系统

一、死锁预防：

(一)死锁产生的四个必要条件：互斥条件，请求和保持条件、不剥夺条件、环路等待条件

(二)网络与分布式系统中预防死锁的方法**:**通过剥夺资源以破坏循环等待条件。

方法：赋给每个进程一个唯一的优先数，这个优先数被用于决定一个进程 **Pi** 是否等待另外一个进程 **Pi**。例如：如果**Pi**具有更高的优先数，可以令**Pi**等待 **Pj**，否则**Pi**回退，即死掉。

缺陷：可能出现饥饿现象，因为有些低优先级的进程可能总是被回退，为此有人提出用时间邮戳的设想，系统中的每个进程，在其产生时被赋予一个唯一的时间邮戳，下面介绍两个方案：

方案一：死等：资源申请者回退：

此方案基于非剥夺技术，当一个进程**Pi**要求另外一个进程**Pj**保持的资源时，**Pi**被允许等待，仅当它具有比**Pj**更小的邮戳时间，即**Pi**是比**Pj**更老的，否则**Pi**回退。

方案二：剥夺式等待：资源占有者回退：

此法基于剥夺技术，是死等方案的改进，当进程**Pi**要求进程**Pj**当前占有的资源时，则**Pi**获准等待的条件是它具有比**Pj**更大的邮戳时间，即**Pi**比**Pj**更年轻，否则**Pj**回退。

二、死锁检测:

死锁预防技术可能剥夺一些资源或回退一些 进程，即使死锁不会出现。为了防止不必要的剥夺和回退，可以允许死锁发生，并在发生后将死锁检测出来，为此需要构造资源分配状态图，若此图出现环路，则死锁发生。

(一)资源等待图的保存方法：要求每个站点保持一个局部等待图，图中站点对应占有和申请局部于该站点资源的那些进程，这些进程可能是局部于本站点的，也可能是属于其它站点的。

(二)判断:

如果注意一个局部等待图中出现了环路，则死锁已经发生，但若每个站点的局部等待图均无环路，并不意味着没有死锁，为了确定没有思锁，必须证明所有局部等待图之“并”没有环路，两个或多个局部图之“并”已经出现了一个环路。

DISTRIBUTED COORDINATION

Reaching Agreement Between Processes

The problem here is how to get agreement with an unreliable mechanism. In order to do an election, as we just discussed, it would be necessary to work around the following problems.

UNRELIABLE COMMUNICATIONS

- Can have faulty links - can use a timeout to detect this.

FAULTY PROCESSES

- Can have faulty processes generating bad messages.
- Cannot guarantee agreement.

分布式文件系统

- Caching vs. Remote Access

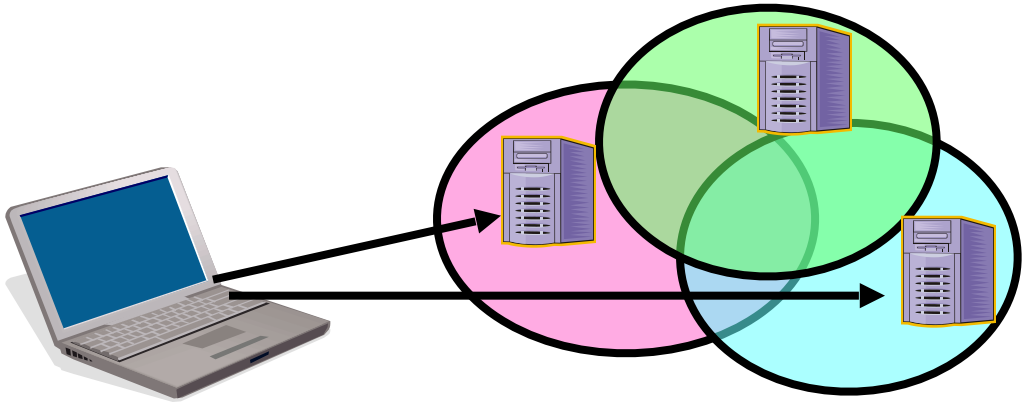
- 局部性原理：Many remote accesses can be handled by a local cache. There's a great deal of locality of reference in file accesses. Servers can be accessed only occasionally rather than for each access.

- 网络效率：Caching causes data to be moved in a few big chunks rather than in many smaller pieces; this leads to considerable efficiency for the network.

- 一致性：Cache consistency is the major problem with caching. When there are infrequent writes, caching is a win. In environments with many writes, the work required to maintain consistency overwhelms caching advantages.

- 复杂性：Caching requires a whole separate mechanism to support acquiring and storage of large amounts of data. Remote service merely does what's required for each call. As such, caching introduces an extra layer and mechanism and is more complicated than remote service.

How to perform Authorization for Distributed Systems?



- Issues: Are all user names in world unique?
  - No! They only have small number of characters
    - kubi@mit.edu → kubitron@lcs.mit.edu → kubitron@cs.berkeley.edu
    - However, someone thought their friend was kubi@mit.edu and I got very private email intended for someone else...
  - Need something better, more unique to identify person
- Suppose want to connect with any server at any time?
  - Need an account on every machine! (possibly with different user name for each account)
  - OR: Need to use something more universal as identity
    - Public Keys! (Called "Principles")
    - People *are* their public keys

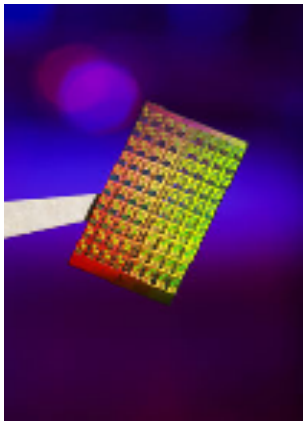
Impact of the Power Density Wall

- The real "Moore's Law" continues
  - i.e. # of transistors per chip continues to increase exponentially
- But thermal limitations prevent us from scaling up clock rates
  - otherwise the chips would start to melt, given practical heat sink technology

- How can we deliver more performance to individual applications?
  → increasing numbers of cores per chip

- Caveat:
  - in order for a given application to run faster, it must exploit parallelism

ManyCore Chips: The future is here



- Intel 80-core multicore chip (Feb 2007)
  - 80 simple cores
  - Two floating point engines /core
  - Mesh-like "network-on-a-chip"
  - 100 million transistors
  - 65nm feature size

- "ManyCore" refers to many processors/chip
  - 64? 128? Hard to say exact boundary
- Question: How can ManyCore change our view of OSs?
  - ManyCore is a challenge
    - Need to be able to take advantage of parallelism
    - Must utilize many processors somehow
  - ManyCore is an opportunity
    - Manufacturers are desperate to figure out how to program
    - Willing to change many things: hardware, software, etc.
  - Can we improve: security, responsiveness, programmability?

Examples from Apple's product line:

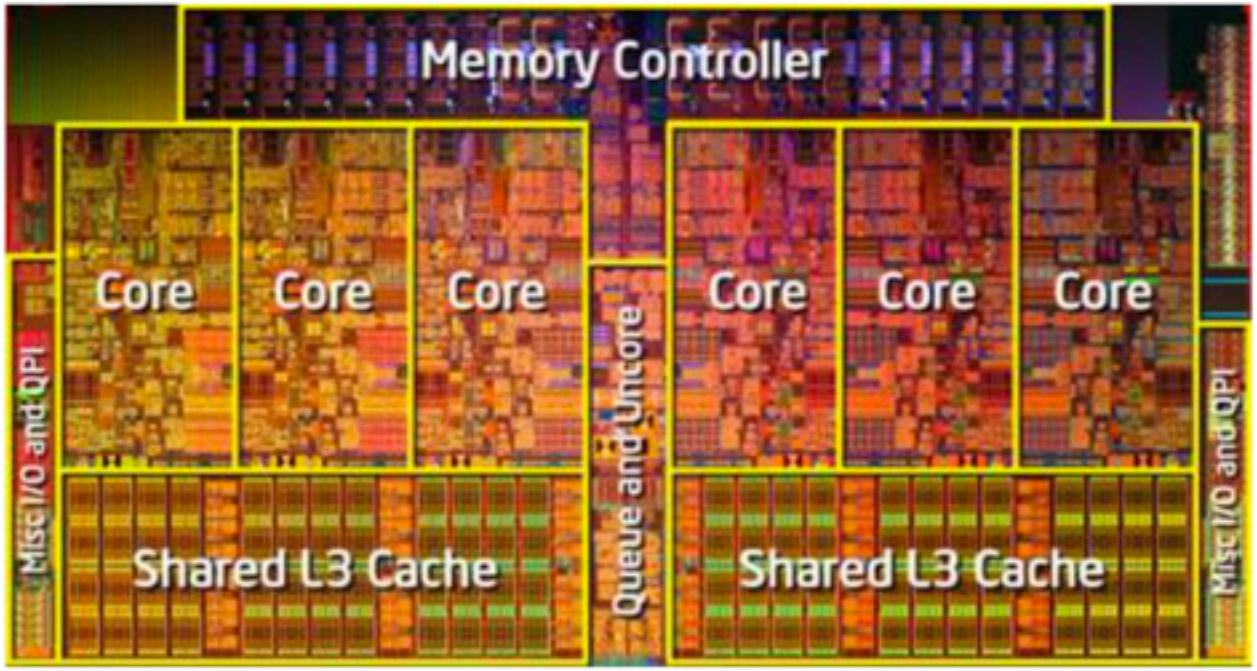
Mac Pro
12 Intel Xeon E5 cores

iMac
4 Intel Core i5 cores

MacBook Pro Retina 15"
4 Intel Core i7 cores

iPad Air 2
3 A8X cores

Example "Multicore" Processor: Intel Core i7

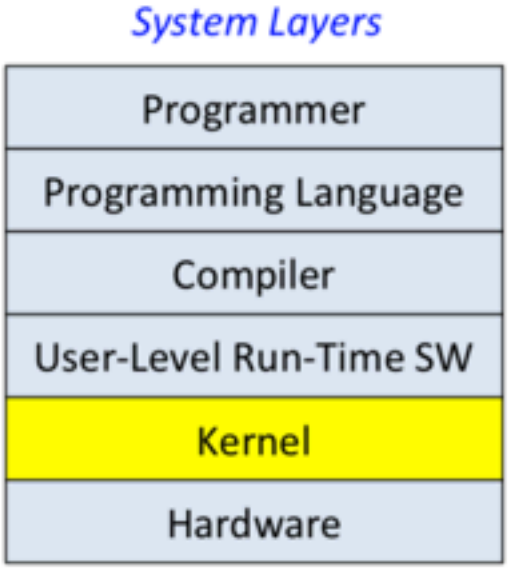


Intel Core i7-980X (6 cores)

- Cores: six 3.33 GHz Nahelem processors (with 2-way "Hyper-Threading")
- Caches: 64KB L1 (private), 256KB L2 (private), 12MB L3 (shared)

Impact of Parallel Processing on the Kernel (vs. Other Layers)

- Kernel itself becomes a parallel program
  - avoid bottlenecks when accessing data structures
    - lock contention, communication, load balancing, etc.
  - use all of the standard parallel programming tricks
- Thread scheduling gets more complicated
  - parallel programmers usually assume:
    - all threads running *simultaneously*
      - load balancing, avoiding synchronization problems
    - threads *don't move* between processors
      - for optimizing communication and cache locality
- Primitives for naming, communicating, and coordinating need to be *fast*
  - Shared Address Space: virtual memory management across threads
  - Message Passing: low-latency send/receive primitives



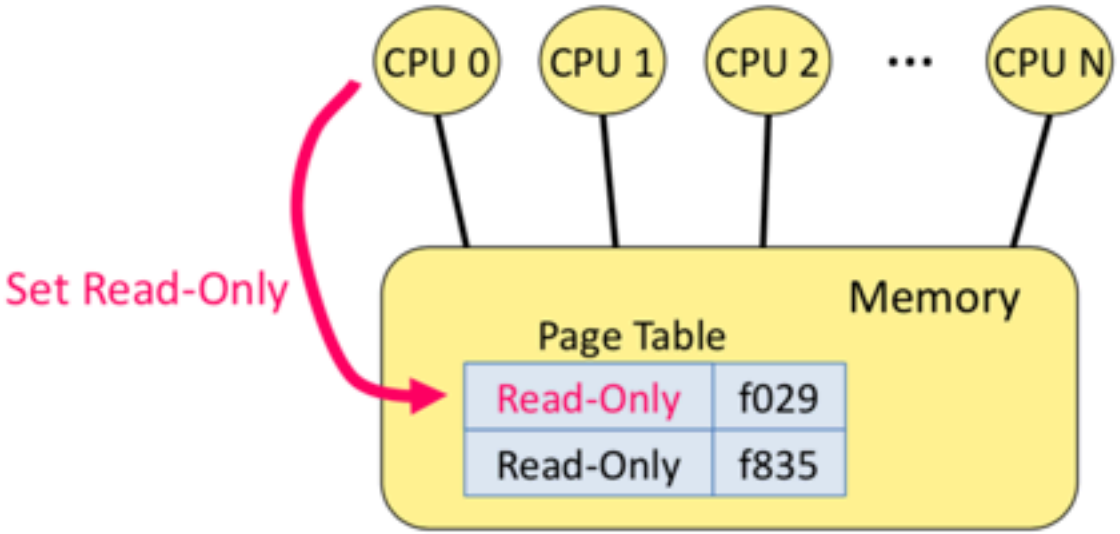
- One important role of the OS:
  - provide *protection* so that buggy processes don't corrupt other processes

- Shared Address Space:
  - access permissions for virtual memory pages
    - e.g., set pages to read-only during copy-on-write optimization

- Message Passing:
  - ensure that target thread of send() message is a valid recipient

How Do We Propagate Changes to Access Permissions?

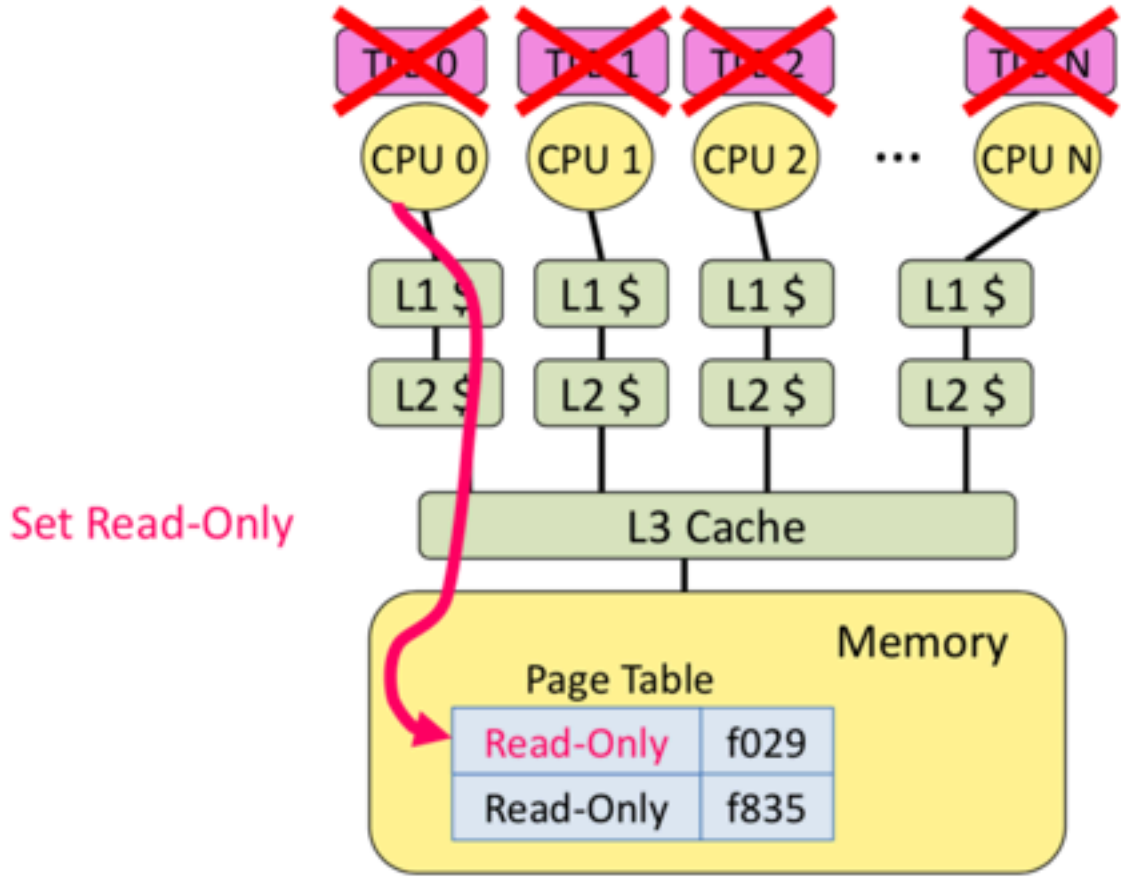
- What if parallel machines (and VM management) simply looked like this:
  - *(assume that this is a parallel program, deliberately sharing pages)*



(No caches or TLBs.)

- Updates to the page tables (in shared memory) could be read by other threads
- But this would be a very slow machine!
  - Why?

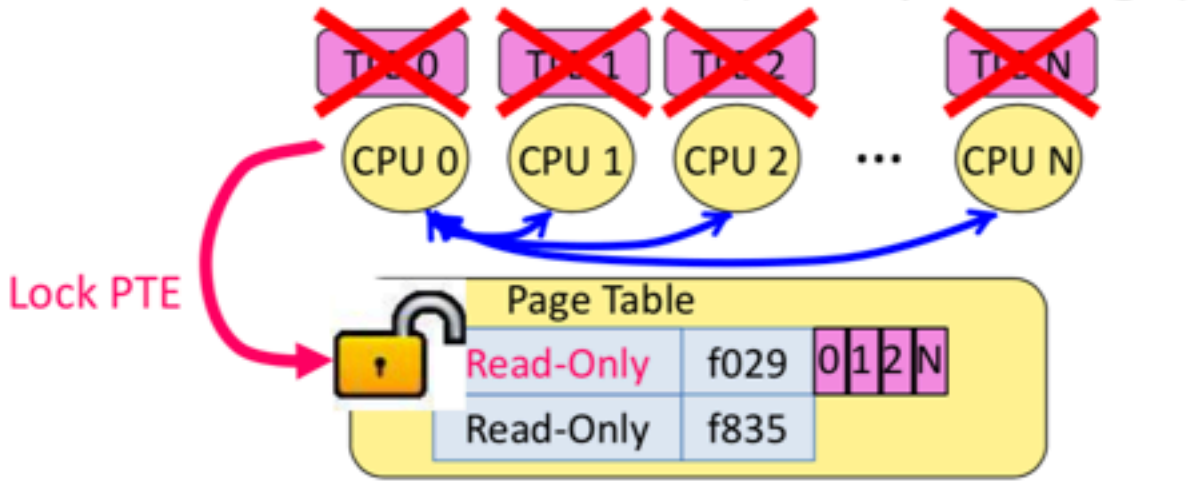
VM Management in a Multicore Processor



"TLB Shootdown":

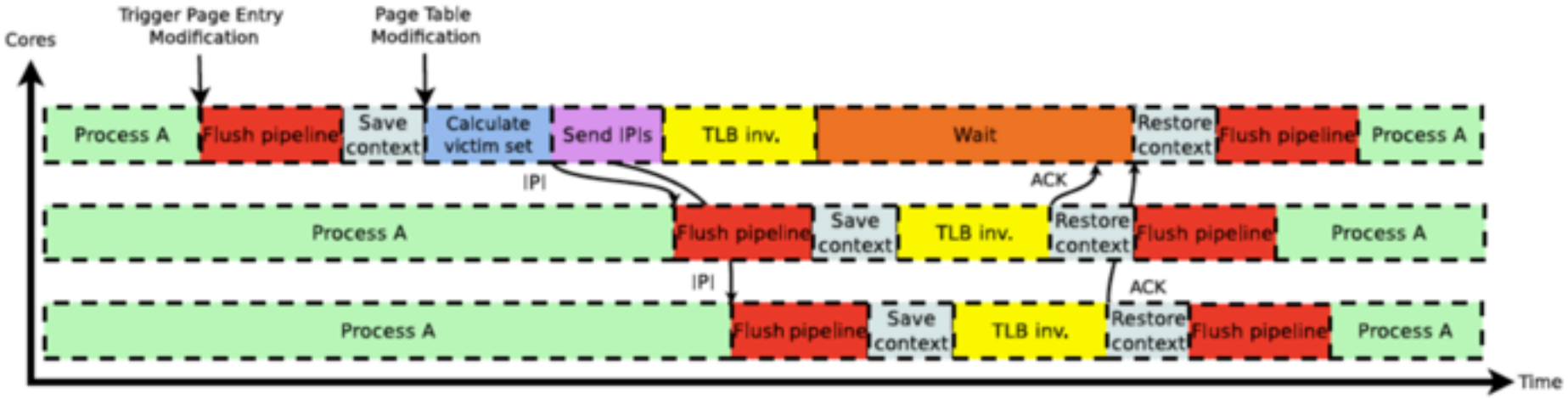
- relevant entries in the TLBs of other processor cores need to be flushed

TLB Shootdown (Example Design)



1. Initiating core triggers OS to lock the corresponding Page Table Entry (PTE)
2. OS generates a list of cores that may be using this PTE (erring conservatively)
3. Initiating core sends an Inter-Processor Interrupt (IPI) to those other cores
   - requesting that they invalidate their corresponding TLB entries
4. Initiating core invalidates local TLB entry; waits for acknowledgements
5. Other cores receive interrupts, execute interrupt handler which invalidates TLBs
   - send an acknowledgement back to the initiating core
6. Once initiating core receives all acknowledgements, it unlocks the PTE

TLB Shootdown Timeline



- Image from: Villavieja *et al*, "DiDi: Mitigating the Performance Impact of TLB Shootdowns Using a Shared TLB Directory." In *Proceedings of PACT 2011*.

Performance of TLB Shootdown

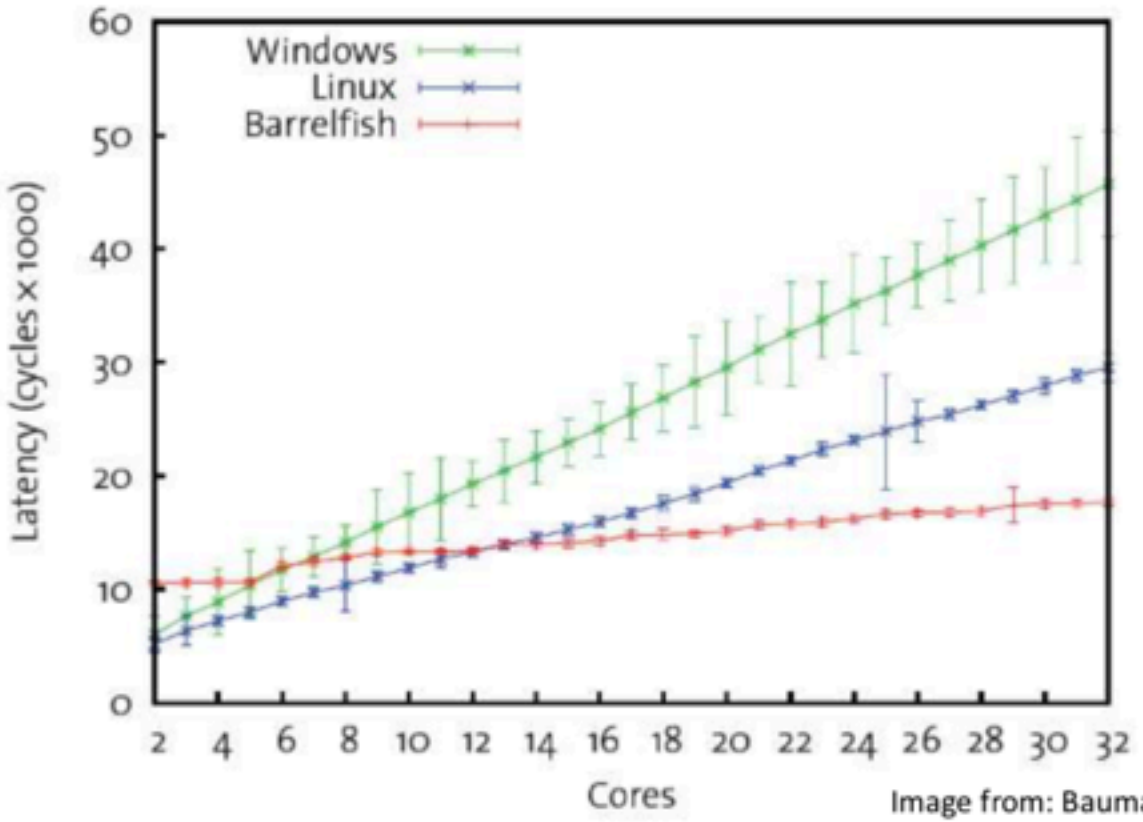
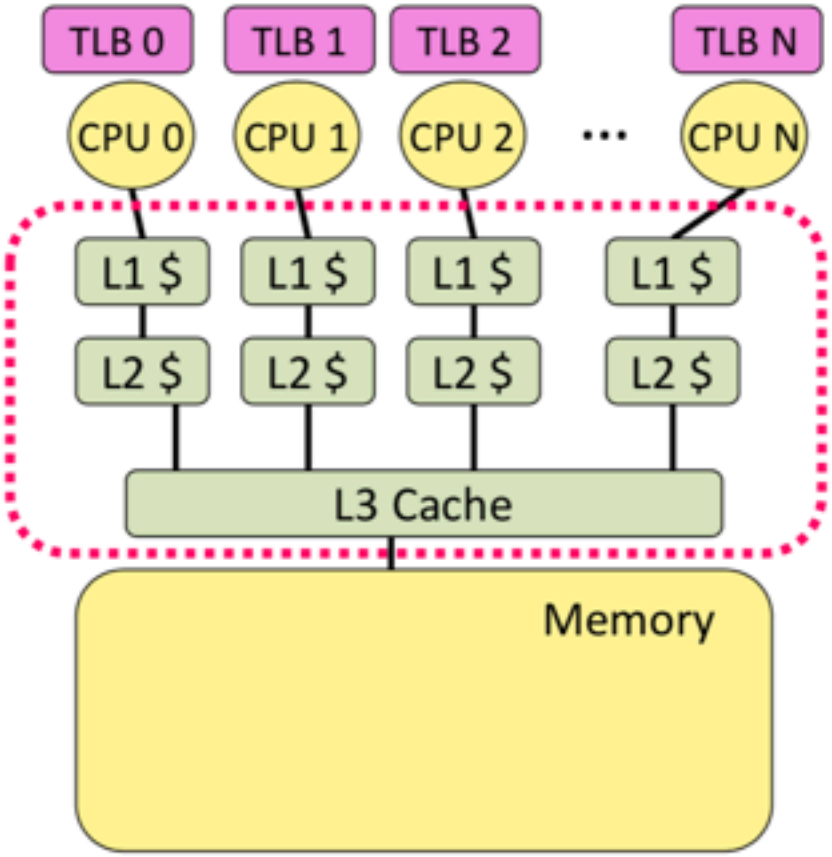


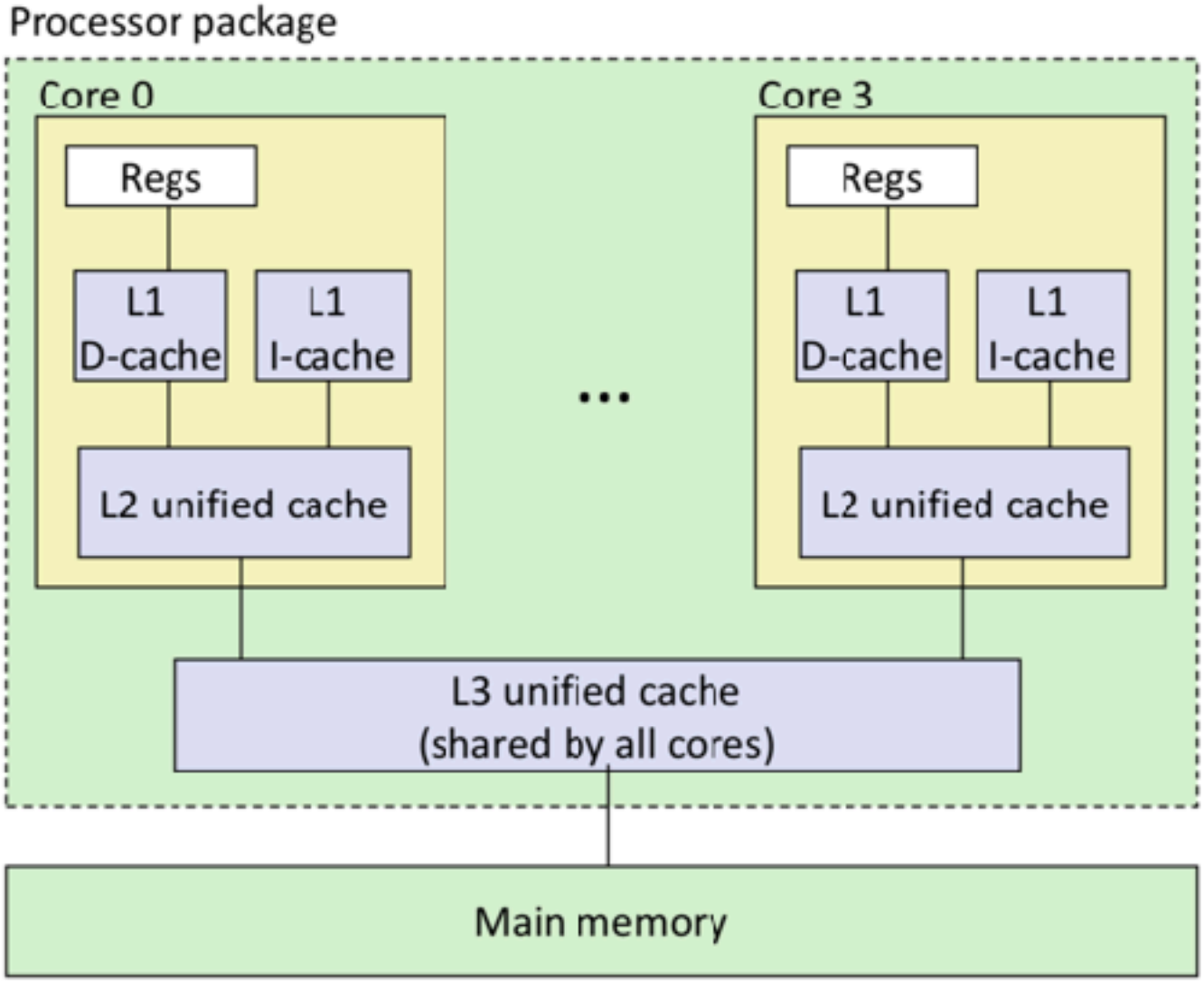
Image from: Baumann *et al*,"The Multikernel: a New OS Architecture for Scalable Multicore Systems." In Proceedings of SOSP '09.

- Expensive operation
  - e.g., over 10,000 cycles on 8 or more cores
- Gets more expensive with increasing numbers of cores


TLB 0
TLB 1
TLB 2
TLB N
CPU 0
CPU 1
CPU 2
...
CPU N
L1 $
L1 $
L1 $
L1 $
L2 $
L2 $
L2 $
L2 $
L3 Cache
Memory

- Ideally, memory would be arbitrarily fast, large, and cheap
  - unfortunately, you can't have all three (e.g., fast → small, large → slow)
  - cache hierarchies are a hybrid approach
    - if all goes well, they will behave as though they are both fast and large
- Cache hierarchies work due to *locality*
  - temporal locality → even relatively small caches may have high hit rates
  - spatial locality → move data in *blocks* (e.g., 64 bytes)
- Locating the data:
  - Main memory: directly (geographically) addressed
    - we know exactly where to look:
      - at the unique location corresponding to the address
  - Cache: may or may not be somewhere in the cache
    - need *tags* to identify the data
    - may need to check multiple locations, depending on the degree of associativity

Intel Quad Core i7 Cache Hierarchy



L1 I-cache and D-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 11 cycles

L3 unified cache:
8 MB, 16-way,
Access: 30-40 cycles

Block size: 64 bytes for all caches.

What is Correct Behavior for a Parallel Memory Hierarchy?

- Note: side-effects of writes are only observable when *reads* occur
  - so we will focus on the values returned by reads

- Intuitive answer:
  - reading a location should return the latest value written (by any thread)

- Hmm… what does "latest" mean exactly?
  - within a thread, it can be defined by program order
  - but what about across threads?
    - the most recent write in physical time?
      - hopefully not, because there is no way that the hardware can pull that off
        - » e.g., if it takes >10 cycles to communicate between processors, there is no way that processor 0 can know what processor 1 did 2 clock ticks ago

Solid State Drive

Solid-State Disks (SSD)

What is “solid state” storage?

- RAM backed by a battery!
- “NOR flash”
- “NAND flash”
- Newer things

Solid-State Disks (SSD)

What is "solid state" storage?

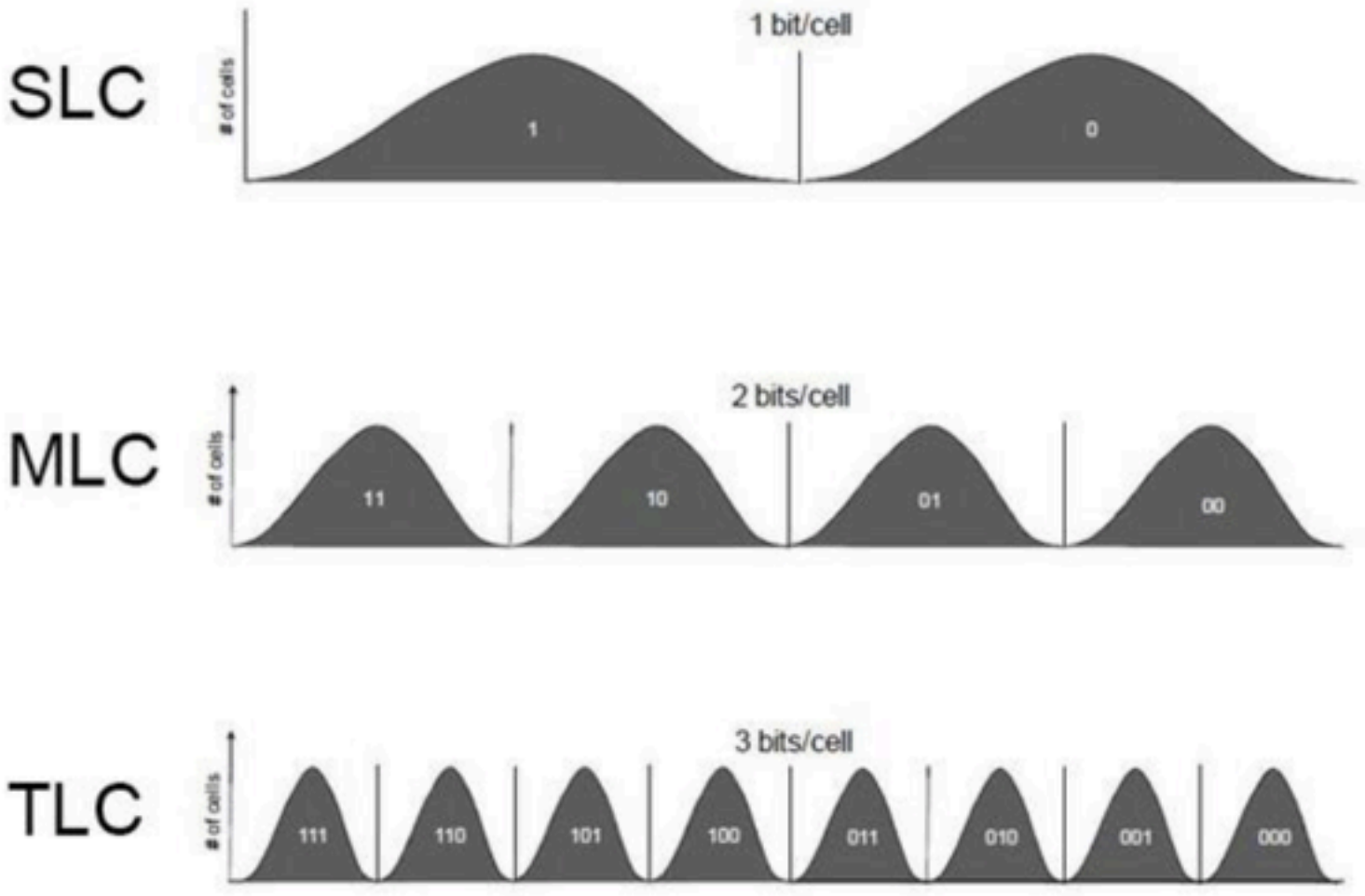
- RAM backed by a battery!
  - Fast
  - Legato "Prestoserve", 1989 ($8,000 for 1 MB)
  - Allowed NFS servers to complete write RPCs without waiting for disk
- "NOR flash"
  - Word-accessible
  - Writes are slow, density is low
  - Used to boot embedded devices, store configuration
- "NAND flash"
  - Read/write "pages" (512 B), erase "blocks" (16 KB)
  - Most SSDs today are NAND flash
- Newer things
  - "Phase-change" memory (melting), magnetic RAM, "Memristor" memory

Newer Things

What is “solid state” storage?

- “Phase-change” memory (melting)
- Magnetic RAM
- “Memristor” memory
- Intel's new “3D XPoint” / “Optane”
  - How it works isn't widely known
  - Characteristics
    - Word addressable (small random accesses are fast)
    - Slower than RAM, faster than NAND flash
    - Less power than RAM, more power than NAND flash
    - Doesn't have write amplification
    - Wear is less of a threat
    - Price is a multiple of NAND flash
  - Initially packaged as “Optane” SSD
  - Expected to be packaged later as DIMMs
    - Exact usage model unclear

NAND


SLC
1 bit/cell
# of cells
1
0
MLC
2 bits/cell
# of cells
11
10
01
00
TLC
3 bits/cell
# of cells
111
110
101
100
011
010
001
000

Solid-State Disks (SSD)

Architectural features of NAND flash

- No moving parts means no “seek time” / “rotational delay”
- Read is faster than write
- Write and “erase” are different
  - A blank page can be written to (once)
  - A written page must be erased before rewriting
  - But pages can't be individually erased!
    - “Erase” works on multi-page *blocks* (16 KB)
    - “Erase” is very slow
    - “Erase” *damages the block* each time

Implications

- “Write amplification”
- “Wear leveling”

Advantages

- Reliability in portable environments and no noise
  - No moving parts
- Faster start up
  - Does not need spin up
- Extremely low read latency
  - No seek time (25 us per page/4KB)
- Deterministic read performance
  - The performance does not depends on the location of data

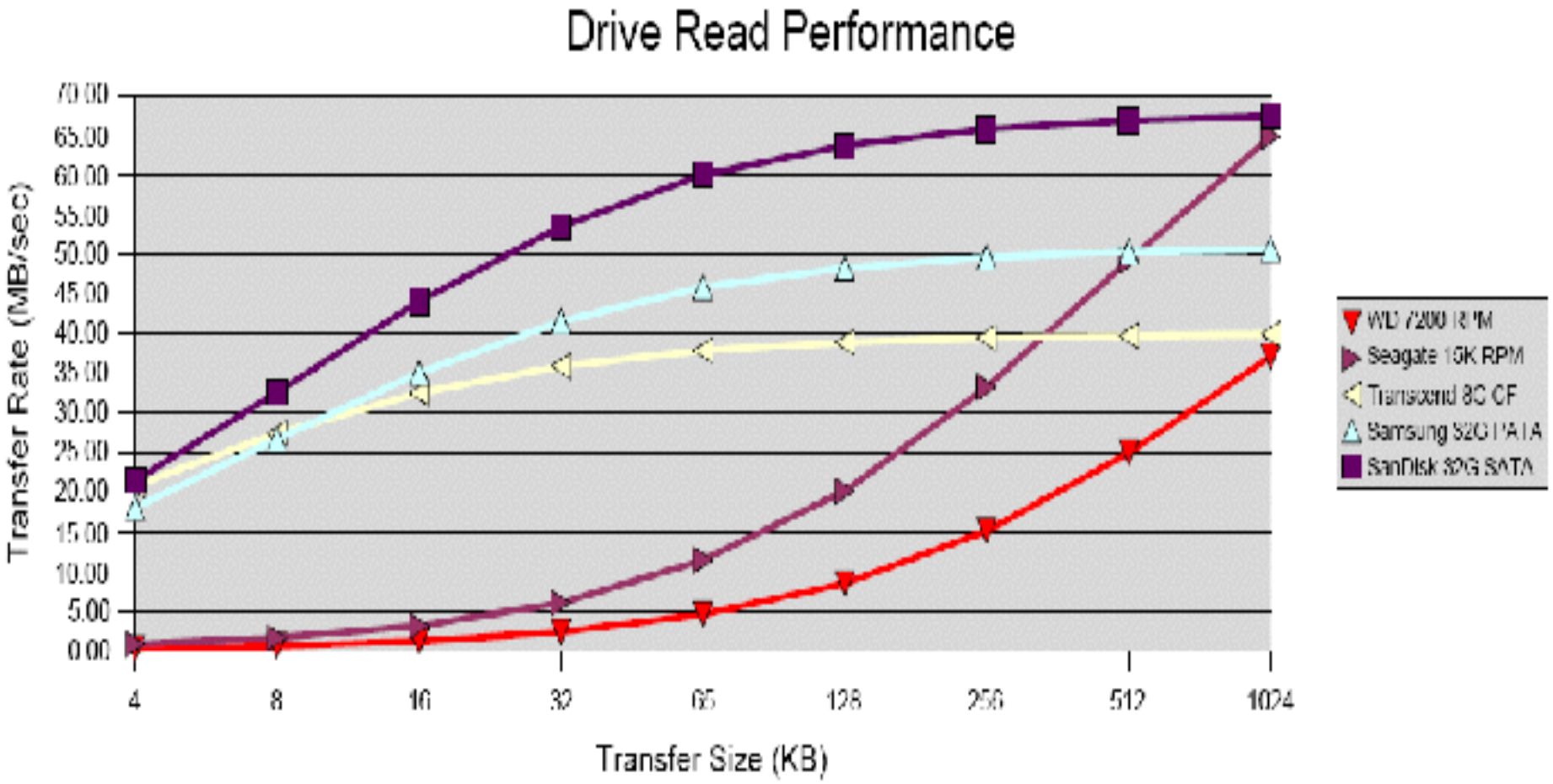
Disadvantage

- Cost significantly more per unit capacity
  - 3$/GB vs. 0.15$/GB
- Limited write erase time
  - 100000 writes for SLC (MLC is even fewer)
  - high endurance cells may have an 1-5 million
  - But some files still need more
  - Weaver leaving to spread writes all over the disk
- Slower write speeds because of the erase blocks are becoming larger and larger(1.5 ms per erase)
- For low capacity flash SSDs, low power consumption and heat production when in active use. High capacity SSDs may have significant higher power requirements

Typical read and write rates

	Drive Model	Description	Seek Time			Latency	Read XFR Rate		Write XFR Rate	
			Track to Track	Average	Full Stroke		Outer Tracks	Inner Tracks	Outer Tracks	Inner Tracks
Hard Drives	Western Digital WD7500AYYS	7200 RPM 3.5" SATA	0.6 ms	8.9 ms	12.0 ms	4.2 ms	85 MB/sec	60 MB/sec*	85 MB/sec	60 MB/sec*
	Seagate ST9367515S	15K RPM 2.5" SAS	0.2 ms	2.9 ms	5.0 ms*	2.0 ms	112 MB/sec	79 MB/sec	112 MB/sec	79 MB/sec
Flash SSDs	Transcend TS8GCF266	8GB 266x CF Card	0.09ms				40 MB/sec		32 MB/sec	
	Samsung MCAQE32G5APP	32G 2.5" PATA	0.14ms				51 MB/sec		28 MB/sec	
	Sandisk SATA5000	32G 2.5" SATA	0.125ms				68 MB/sec		40 MB/sec	

Drive read performance



Mixed writes and reads

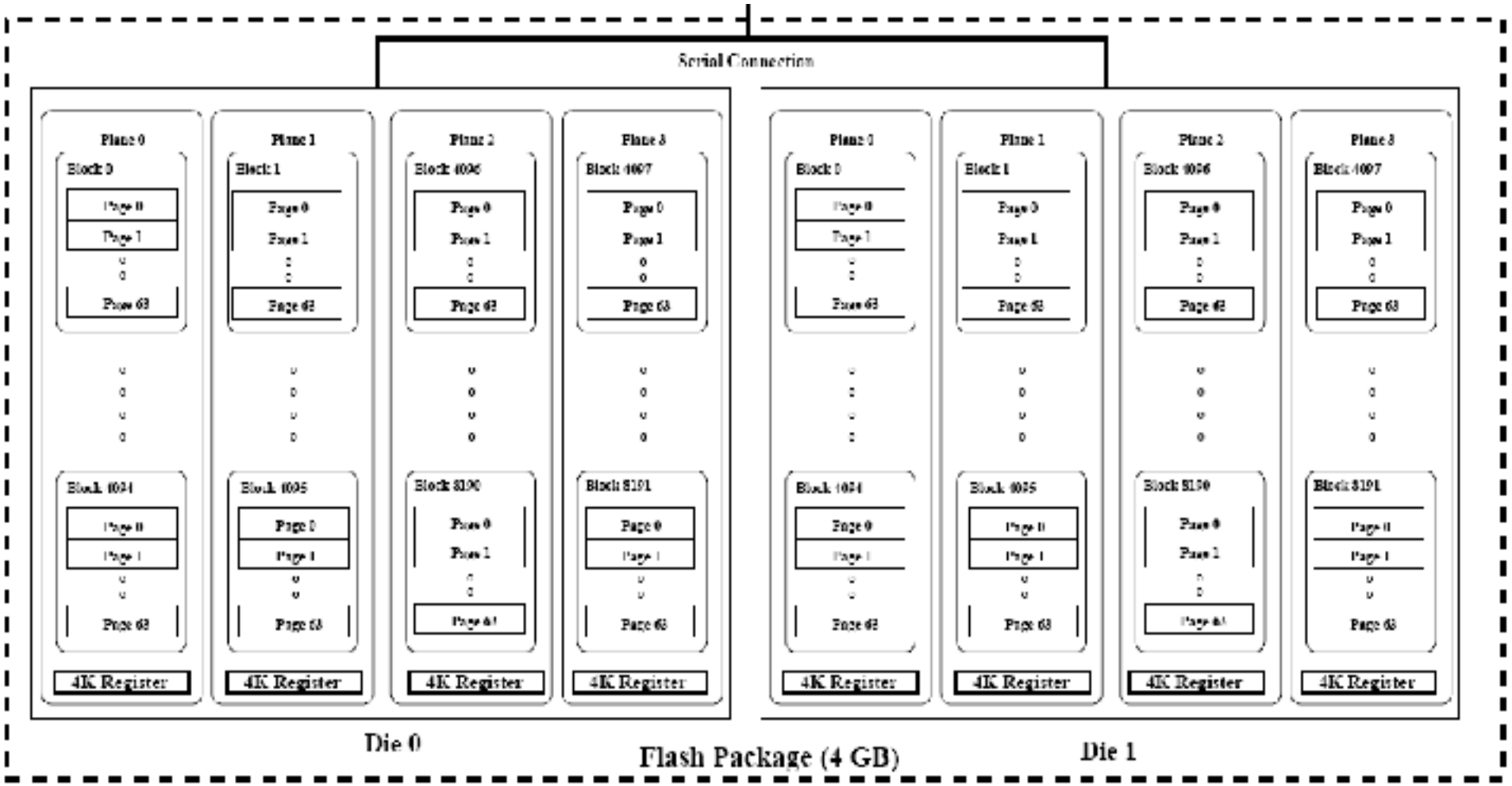
% Writes	Total IOPS	Performance vs 15K SAS Hard Drive
0%	5400	20x better
5%	252	1.25x better
10%	130	1.5x worse
20%	65	3x worse
50%	26	8x worse
100%	13	16x worse

	Sequential		Random 4K	
	Read	Write	Read	Write
USB	11.7 MB/sec	4.3 MB/sec	150/sec	<20/sec
MTron	100 MB/sec	80 MB/sec	11K/sec	130/sec
Zeus	200 MB/sec	100 MB/sec	52K/sec	11K/sec
FusionIO	700 MB/sec	600 MB/sec	87K/sec	Not avail

Power consumption

Device	Approximate power consumption
DRAM DIMM module (1 GB)	5W
15,000-RPM drive (300 GB)	17.2W
7200-RPM drive (750 GB)	12.6W
High-performance flash SSD (128 GB)	2W

Samsug flash internals



Bandwidth and interleave

- Without interleaving
  - For read: 25+100 us per page
    - 8000 reads/s = 32MB/s
  - For write: 200+100 us per page
    - 3330 writes/s = 13 MB/s
- With interleaving
  - For read
    - 10000 reads/s = 40MB/s
  - For write
    - 5000 writes/s = 20 MB/s

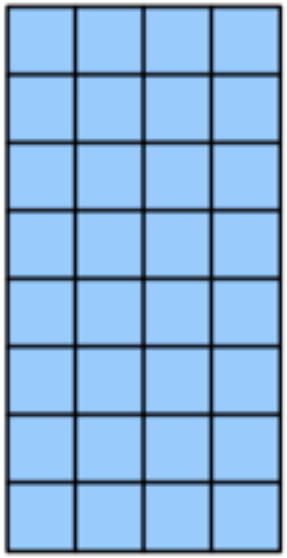
An example of SSD

- 1 die = 4 planes
- 1 plane = 2048 blocks
- 1 block = 64 pages
- 1 page = 4KB
- Dies can operate independently
- Reading and programming is performed on a page basis, erasure can only be performed on a block basis.

- Read
  - 25μs from page to data register
  - 100μs transfer in the serial line
- Write
  - Page granularity
  - Sequentially with in a block
  - Block must be erased before writing
  - 200μs from register into flash cells

“Write Amplification”

Goal: update 8 pages (4 KB) in a block (16 KB)

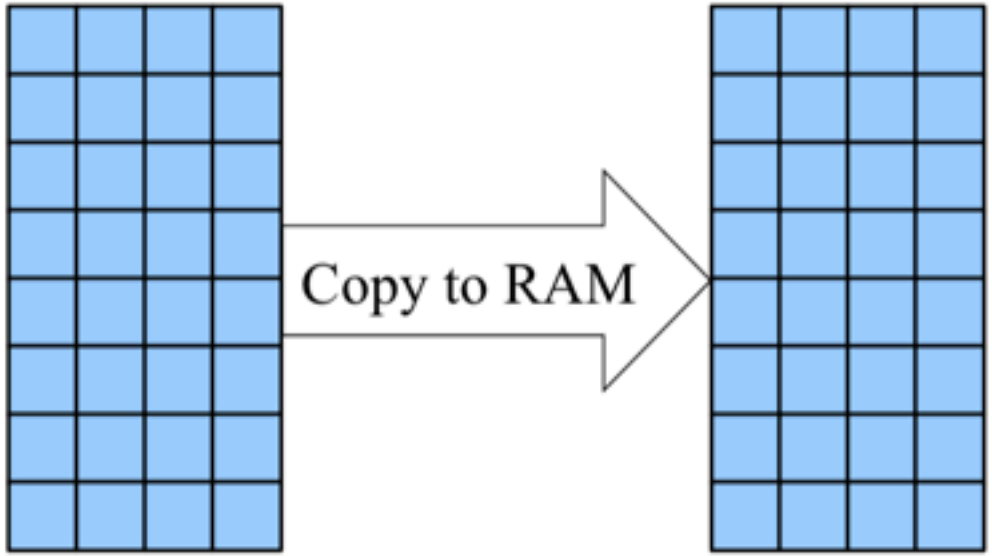


“Write Amplification”

Goal: update 8 pages (4 KB) in a block (16 KB)

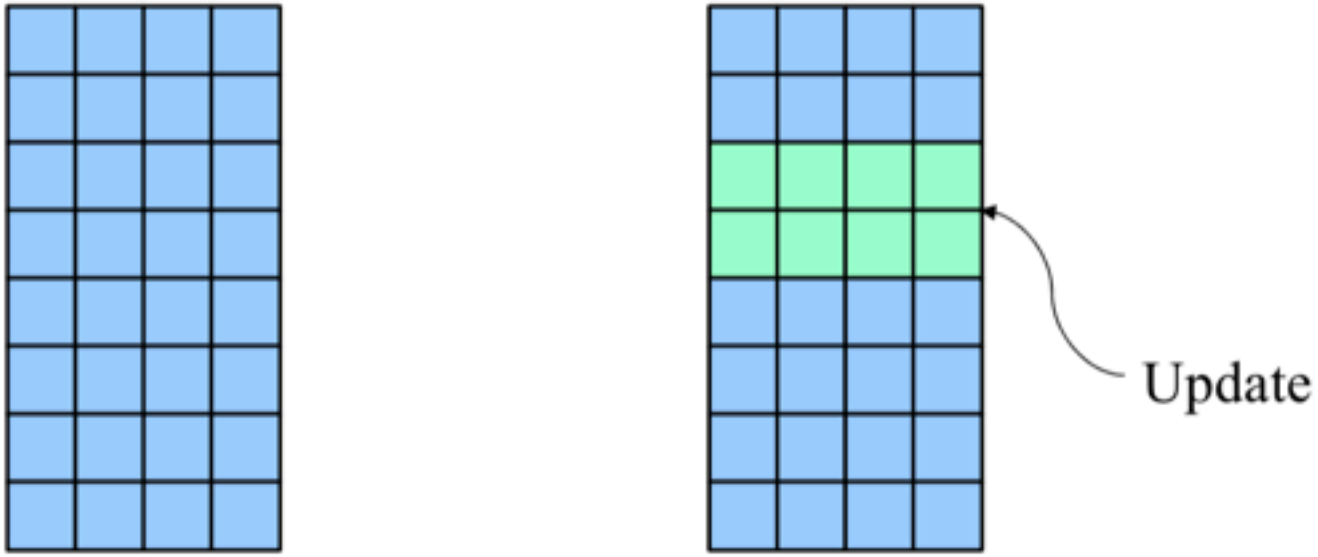
“Write Amplification”

Goal: update 8 pages (4 KB) in a block (16 KB)



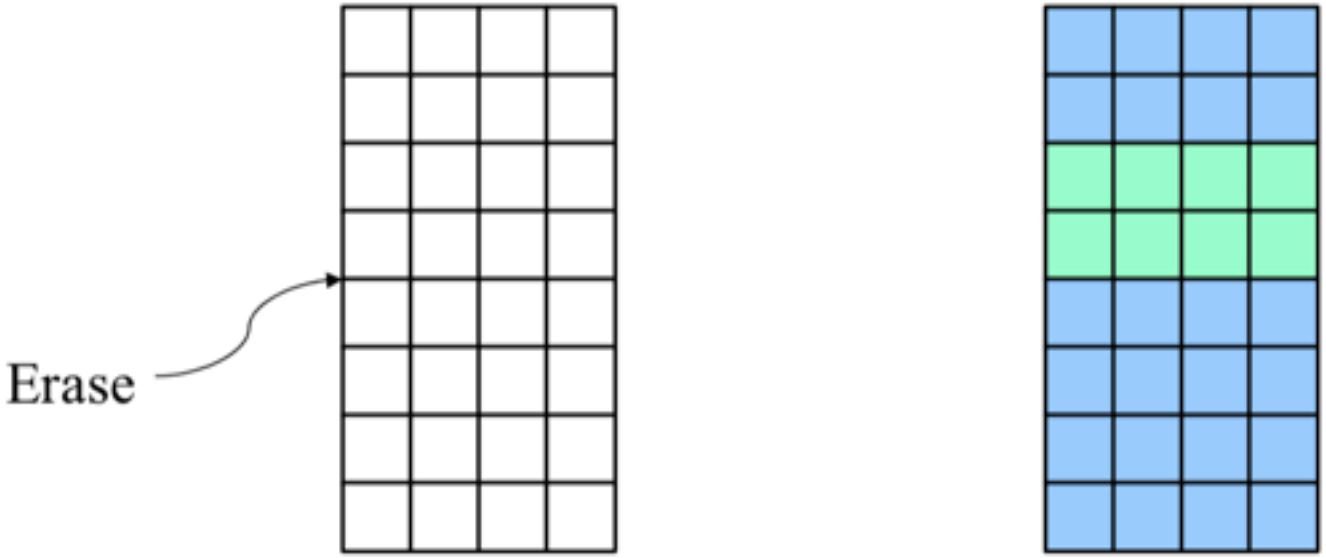
“Write Amplification”

Goal: update 8 pages (4 KB) in a block (16 KB)



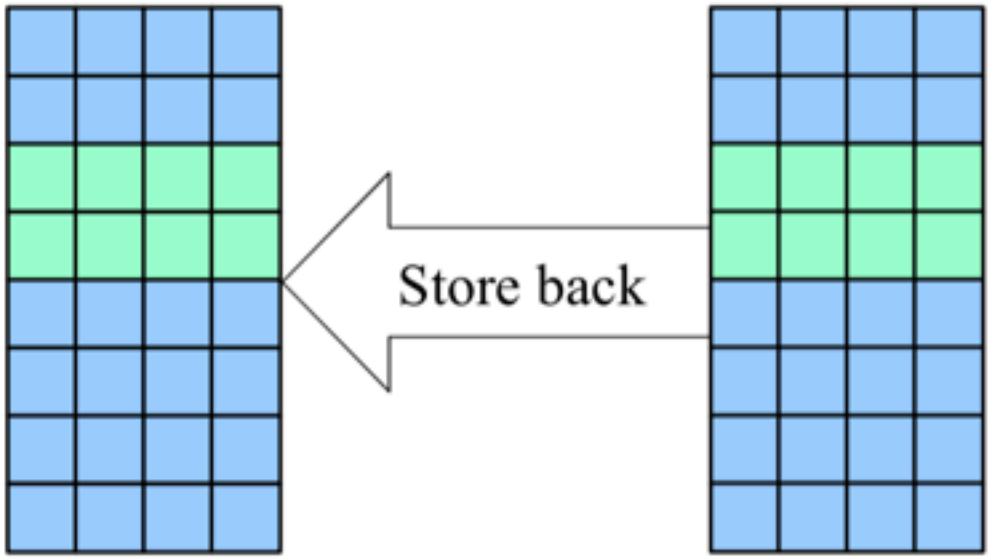
“Write Amplification”

Goal: update 8 pages (4 KB) in a block (16 KB)



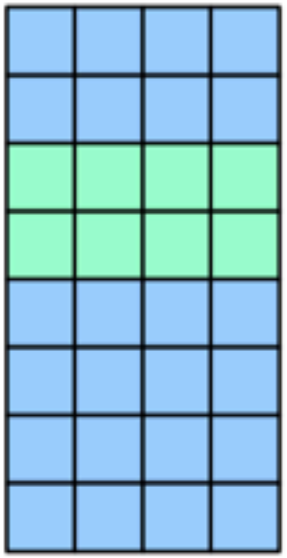
“Write Amplification”

Goal: update 8 pages (4 KB) in a block (16 KB)



"Write Amplification"

Goal: update 8 pages (4 KB) in a block (16 KB)



Result

- Logical: wrote 4 KB
- Physical: erased and write 16 KB
- "Amplification factor": 4
  - Why do we care? Device will wear out 4X faster!

Hot-Spot Wear

The bad case

- File systems like to write the same block repeatedly
- Erasing damages part of the flash
  - ~10,000 erases destroys a block

Strategy: ?

Managing - Wear Leveling

The bad case

- File systems like to write the same block repeatedly
- Erasing damages part of the flash
  - ~10,000 erases destroys a block

Strategy: lie to the OS!

- Host believes it is writing to specific “disk blocks” - LBA
- Store the information somewhere else!
  - Secretly re-map host address onto NAND address
  - FTL - “flash translation layer”

Managing - Wear Leveling

The bad case

- File systems like to write the same block repeatedly
- Erasing damages part of the flash
  - ~10,000 erases destroys a block

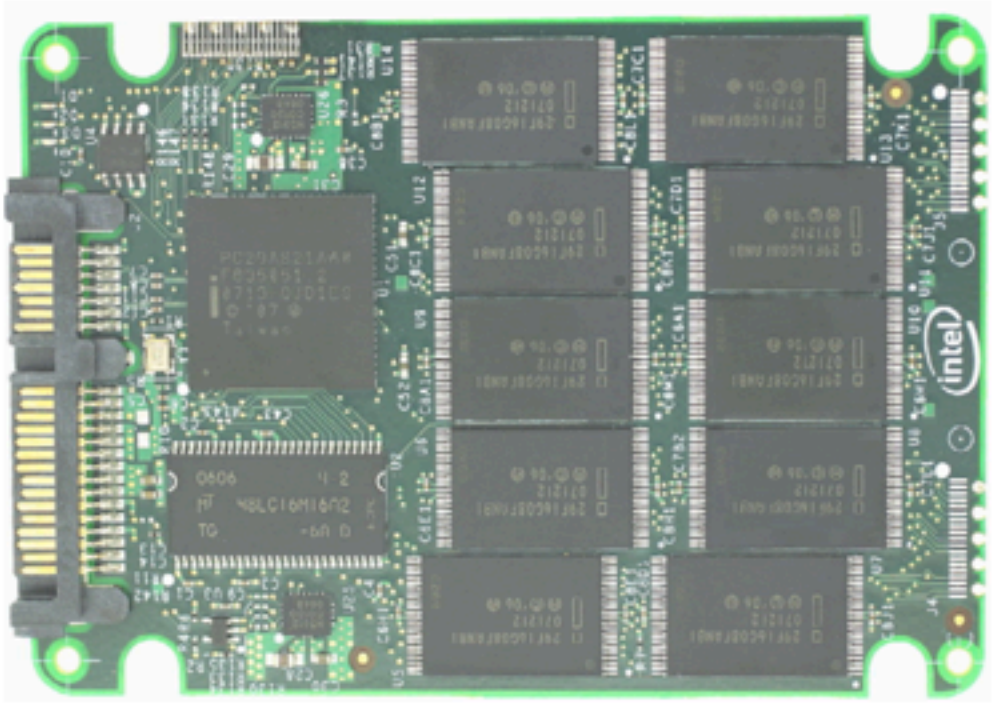
Strategy: lie to the OS!

- Host believes it is writing to specific “disk blocks” - LBA
- Store the information somewhere else!
  - Secretly re-map host address onto NAND address
  - FTL - “flash translation layer”
- Each part of the “disk” moves from one part of the flash to another over time
- “Over-provision”
  - Advertise less space than there really is
  - Use spare space to replace worn-out blocks
- Use up overprovisioning as blocks wear out

Wear Leveling - FTL

FTL is a *computer*

- **CPU, RAM**
- **Access to lots of flash for code & data structures & user data**



Managing - Write Amplification

The bad case

- Small random writes

Strategy: lie to the OS!

- Host believes it is writing to specific "disk blocks" - LBA
- Store the information somewhere else!
  - Secretly re-map host address onto NAND address
  - FTL - "flash translation layer"
- Group multiple small writes into full blocks
  - Write at sequential write rates
- To update a "disk block", store a new copy *somewhere else*
  - Leaves "holes" in other blocks (stale old block versions)
  - At some point, "clean out" the holes by reading a bunch of old blocks and writing back a smaller number of whole pages
- Rate of cleaning depends amount of unallocated space
  - Controller reserves X% hidden space (ie. 10, 20, 50%)

SSD Summary

SSD vs. disk

- ☺ SSD's implement "regular disk" model
  - LBA sectors
  - Write-sector, read-sector, "park heads", etc.
- ☺ Read operations are extremely fast (100X faster), no "seek time" or "rotational delay" (every sector is "nearby")
- ? Write operations "vary widely" (maybe 100X faster, maybe not faster at all)
- ☺ SSD's use less power than actual disks (~1/5?)
- ☺ SSD's are shock-resistant
- ☹ Writing to an SSD wears it out much faster than a disk
- ☹ SSD's are *expensive* (20X or more)

SSD Summary

Opportunity & threat

- "TRIM" command speeds up writes!
  - "Dear FTL, logically zero-fill these blocks"
- "Securely erase disk" may or may not be possible

The future?

- Lots more SSD's
- Lots more disks too
- Hybrid systems to take advantage of best features of both

What You Should Know

Storage is *slow*

- Whatever you want to do may take *milliseconds*

Storage lies

- You get some number of “disk blocks”
- There is no way to know where on the “disk” they are
- LBA is a faint approximation of proximity

Failure model

- Sometimes a read fails (sorry!)
- Writing to that block will cause the device to re-map
  - Both spinning-disk and SSD
- When re-map space is exhausted, device refuses to write

Security

- Actually erasing information from flash is uncertain