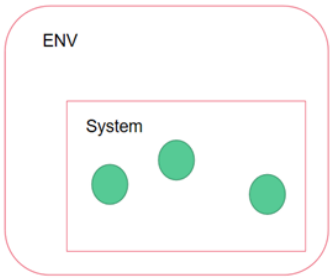
【计算机系统工程】绪论1: Coping with Complexity

@credits Yubin Xia, IPADS

1. Coping with Complexity

1. System & Complexity

什么是system？这是一个问题。



System =

– Interacting set of components with a specified behavior at the interface with its environment

Examples

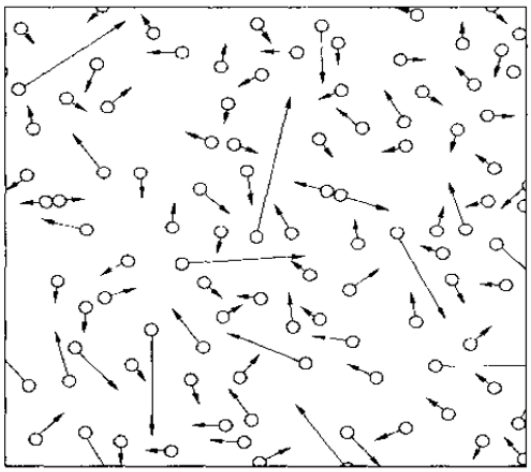
– A web server, an Android phone, file systems, Linux, …

我们发现了很矛盾的地方，系统可以不断增加，但是人的大脑对复杂性的理解是很有限的。如果我们需要搞定每个component的工作机制才能去理解（比如linux kernal，几千万行！），那么大脑真的hold不住。

如何掌握一个复杂系统？

An Example: The Gas System

A systematical way to understand a complex system is needed



当我们研究一个分子的时候，我们可以得到什么属性？

它的质量、方向，然后得到动量。

如果我们有一瓶气体，我们还是用研究一个分子的方式，研究一瓶气体，那显然是不对的，因为它太复杂。

而一瓶气体产生了新的属性——温度！温度体现出了气体分子的平均运动速度！对于单个分子，研究其温度是没有什么意义的。

Compare with the Computer Systems

Programming / Data Structure

– LoC (Lines of Code): From hundreds to millions

Operating System / Architecture

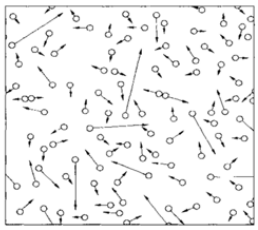
– CPU cores: from one to hundreds

Network

– Nodes: from two to millions

Web Service

– Clients: from tens to millions



我们来类比一下，当代码量增长非常多倍的时候，当我们的cpu核心数目非常非常多的时候，全世界的互联网（相比于两个节点连接而成的网路），会不会产生一些新的类似于温度的property？

2. 14 Properties

我们这门课两个轴：x轴，就是这些property；y轴就是不同的system。

我们这门课关心的就是y轴和x轴有什么关系。

14 Properties of Computer Systems

1. Correctness
2. Latency
3. Throughput
4. Scalability
5. Utilization
6. Performance Isolation
7. Energy Efficiency
8. Consistency
9. Fault Tolerance
10. Security
11. Privacy
12. Trust
13. Compatibility
14. Usability

2.1 Correctness

大家觉得最直观的特性。但其实很难判断，比如：

> Bug or feature?

- windows11双击F11，性能飙升到400%。

A Piece of Code, Seems Normal

```
char *buf = ...;
char *buf_end = ...;
unsigned int len = ...;
if (buf + len >= buf_end)
    return;   /* len too large */
if (buf + len < buf)
    return;   /* overflow, buf+len wrapped around */
/* write to buf[0..len-1] */
```

在C手册中写了，指针+len<指针其实是个undefined behavior。它的实现——depends。gcc的实现是——把这行代码删了。

gcc表示，这是你的代码的问题，我的编译器没问题。

Is This Compilers' Bug?

A true story: GCC bug #30475 (2007/01/15)

A GCC user:

– *"This will create MAJOR SECURITY ISSUES in ALL MANNER OF CODE. I don't care if your language lawyers tell you gcc is right. . . . FIX THIS! NOW!"*

A GCC developer:

– *"I am not joking, the C standard explicitly says signed integer overflow is undefined behavior. . . . GCC is not going to change."*

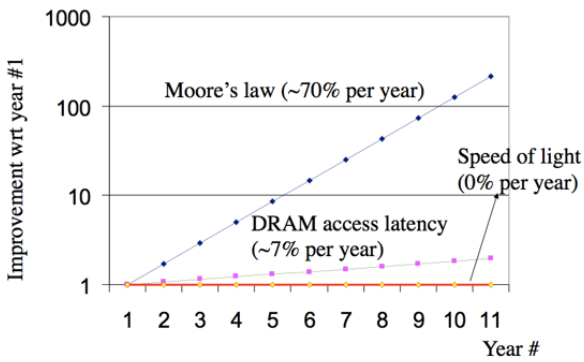
http://gcc.gnu.org/bugzilla/show_bug.cgi?id=30475

我们的自动驾驶代码就是用gcc编译出来的。你真的会去一行行比对你写的c和编译出来的汇编吗？

2.2 Latency

latency很难去optimize。吞吐量不够，我们第一反应是去买服务器，但是缩短每个人访问的latency是很难的。

Latency is hard to optimize, it has physical limitations



“内存墙”：我们发现差距越来越大，内存慢的要死。

2.3 Troughput

Throughput: a number of transactions / transferred-data per second

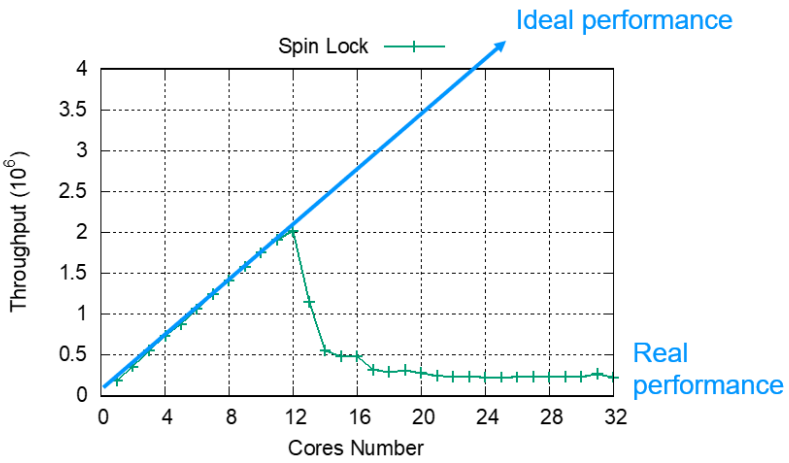
7nm → 5nm → 3nm → 2nm... what's next?

Once stop getting smaller, it has to go bigger

> **CES 2019: Moore's Law is dead, says Nvidia's CEO**
>
> The long-held notion that the processing power of computers increases exponentially every couple of years has hit its limit, according to Jensen Huang.

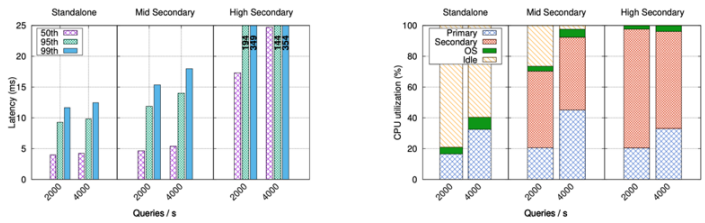
一旦我们的芯片密度不能越来越高，它势必会越来越大。一旦物理达到极限以后，软件的作用就越来越大。

2.4 Scalability



这是真实运行在鲲鹏上的代码。我们理想是核越多，性能越好。但是实际上12核的时候性能暴跌，因为所有的时间都用来争抢锁，在做cache coherence。

2.5 Performance Isolation



(a) Query response latency (b) CPU utilization

Figure 4: Single machine run of IndexServe standalone (no colocation) vs. colocated with an unrestricted secondary. A *mid* secondary increases the 99th percentile query latency by up to 42%, while a *high* increases same by up to 29×.

Iorgulescu, Călin, et al. "Perfiso: Performance isolation for commercial latency-sensitive services." *USENIX ATC'18*.

如何做性能的隔离？一台机器跑了benchmark：

- 情况1：一个人自己跑

- 情况2：背后还跑了一个while(1)，加上这个noisy neighbour，影响是对于99%的latency会增加42%，非常非常noisy的情况，latency会增加29倍。

虽然CPU可以运行多个程序，但是neighbour的情况影响是很大的。

2.6 Utilization

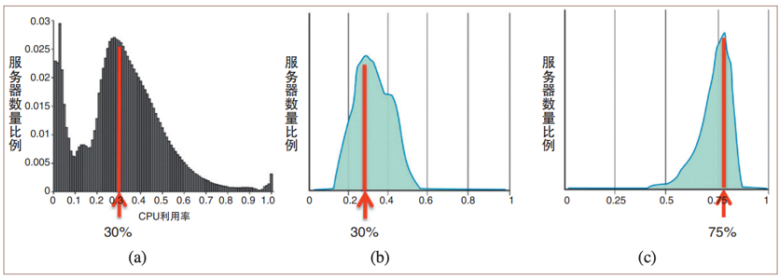


图2　谷歌数据中心CPU利用率：(a)2006年5000台在线应用服务器；(b)2013年2万台在线应用服务器；(c)2013年2万台批处理应用服务器[4]

数据来源："数据中心成本与利用率现状分析"，中国计算机学会通讯，2015.

对于云来说，Google数据中心的CPU利用率是30%。对于要求快速响应的服务器，需要预留很多资源，所以云端的服务器都开着虽然耗着电，但是利用率并没有拉满。

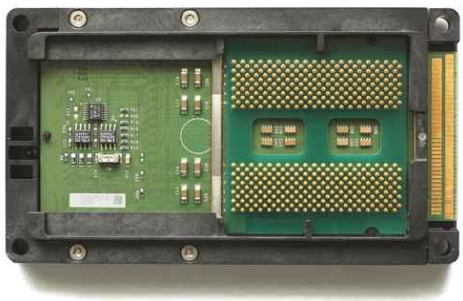
2.7 Energy Efficiency

我们的手机对于energy来说节能是非常重要的；对于台式机和服务器则没有那么重要。

2.8 Compaibility

兼容性。系统会不断进化，但如果进化之后不在兼容，可能产生严重的后果。

Intel's Itanium, 64-bit, not compatible to x86, died.



Itanium Process, from Wikipedia

安腾处理器是inter从32转64的重大选择；本来是想取代奔腾的（安腾的理念确实非常先进）。导致之前所有的x86代码都需要重新编译才能跑。

最后安腾失败了，amd成功了，所以现在我们比如下载东西的时候，后缀可能是amd64。amd就很保守，提出的64位方案是兼容以前的。以前英特尔一直嘲讽amd，现在英特尔不由得follow了这个标准。英特尔：我们叫它x86-64好不好~

还有一个经典的例子是google的tensorflow，被pytorch侵占了大片的份额。

2.9 Usability

左边是windows phone。微软基于desktop的思路（键盘鼠标的交互思路），给手机做触控笔。

2.10 Consistency

在12306买到相同高铁票怎么办?

2019-06-10 17:12

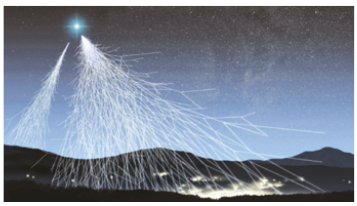
据时间视频消息，6月8日，北京南站，庞女士和丈夫乘坐G497列车，上车后发现票面上的座位坐了人。庞女士和在座位上的乘客交涉后，发现除了身份证和姓名不一样之外，她们的票和自己手上的票一模一样。两人均是在窗口购票，双方购票相差时间三分钟左右。双方在找到列车长之后，庞女士被安排到了另一个车厢。

针对此事，12306客服回应称这种情况很少见，可能是由于机器故障所致。

一致性非常重要。为了容错，我们需要多个备份，但是备份之间就会出现数据不一致这个问题，对于支付宝来说，不能接受数据不一致的情况。比如用不同的服务器一笔钱花了两次或者一个订单扣了两次钱。为了性能一定会有多个副本在多种cache中，所以一定会有这个问题。

2.11 Fault Tolerance

A bug of Cisco router is caused by cosmic radiation



Cisco Bug: CSCuz62750 - Incremental drops on counter DROP_FRM_CRC_ERR_SGMII0 causes traffic loss

Last Modified
Sep 20, 2016

Product
Cisco ASR 9000 Series Aggregation Services Routers

Known Affected Releases
all

Description (partial)
Symptom:
Partial data traffic loss can be observed. Below list counters need to focus to determine the issue:-

"show controller np counters all loc <location>" could indicate FRM_FRM or CRC drop counts.

1082	DROP_FRM_CRC_ERR_SGMII0	1429516994	0
1083	DROP_FRM_FRM_ERR_SGMII0	74455703	0
1084	DROP_FRM_CRC_ERR_SGMII1	1427728248	0
1085	DROP_FRM_FRM_ERR_SGMII1	74942820	0
1086	DROP_FRM_CRC_ERR_SGMII2	1431171380	0
1087	DROP_FRM_FRM_ERR_SGMII2		
1088	DROP_FRM_CRC_ERR_SGMII3		
1089	DROP_FRM_FRM_ERR_SGMII3		

Cosmic radiation

It is also theoretically possible for data corruption to happen after the CRC check, which should then be detected at the higher protocol layers.

Conditions:
Problem observed on operational network. Possible trigger is cosmic radiation causing SEU soft errors.

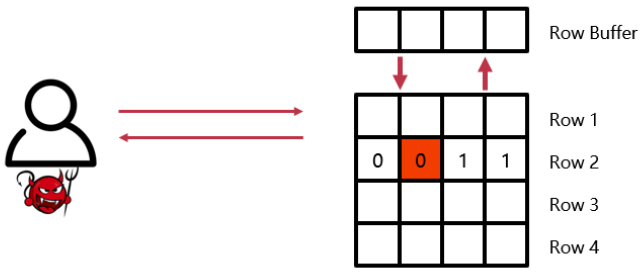
This issue can affect "Juggernaut" LCs (A9K-1X100GE-SE/TR, A9K-2X100GE-SE/TR)

思科的路由：被宇宙射线照射了，导致bit翻转。

其实这个bug并没有那么罕见。在我们的内存中，存储模式是先定位行再定位列，内存需要为这一行进行充电操作，在充电的时候，临近两行的电就会弱一点点。如果我们频繁地读row1和row3会导致row2的电量下降，下降到一定程度的时候，row2的1就会变成0。

Property-11: Fault Tolerance

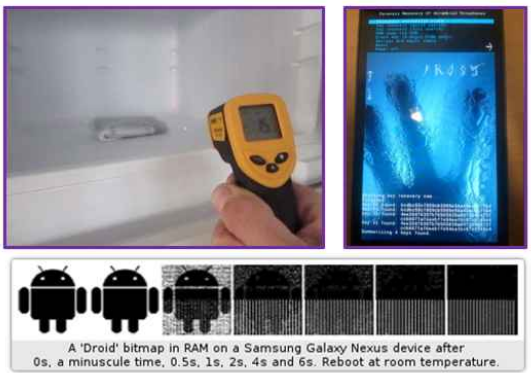
CPU Architecture	Errors	Access-Rate
Intel Haswell (2013)	22.9K	12.3M/sec
Intel Ivy Bridge (2012)	20.7K	11.7M/sec
Intel Sandy Bridge (2011)	16.1K	11.6M/sec
AMD Piledriver (2012)	59	6.1M/sec



Rowhammer Attack: flip memory bits!

2.12 Security

Cold-boot attack: physically attack that can scan memory to get any plaintext

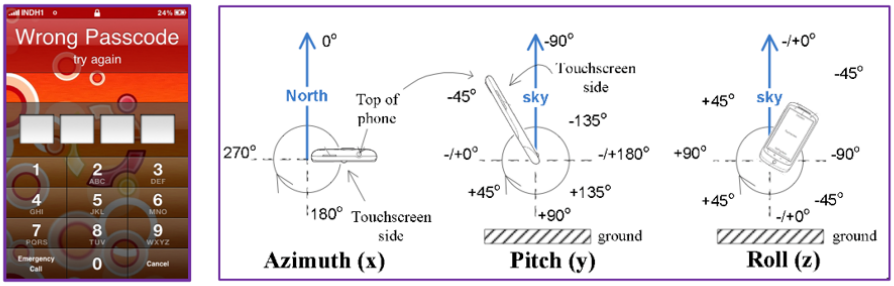


A 'Droid' bitmap in RAM on a Samsung Galaxy Nexus device after 0s, a minuscule time, 0.5s, 1s, 2s, 4s and 6s. Reboot at room temperature.

Müller, Tilo, and Michael Spreitzenbarth. "Frost." *Applied Cryptography and Network Security*. Springer Berlin Heidelberg, 2013.

以手机为例，里面有一堆关键的数据和账号。内存在掉电之后，数据不会马上丢失。放在非常冷的环境下，可以延缓这个过程。手机内存芯片其实就是一个SSD，可以直接读只是数据加密了。

数据加密的密钥放在DDM。内存里是明文，加密到硬盘里是密文。现在保护数据基本上是保护硬盘上的数据，而不保护内存上的数据。

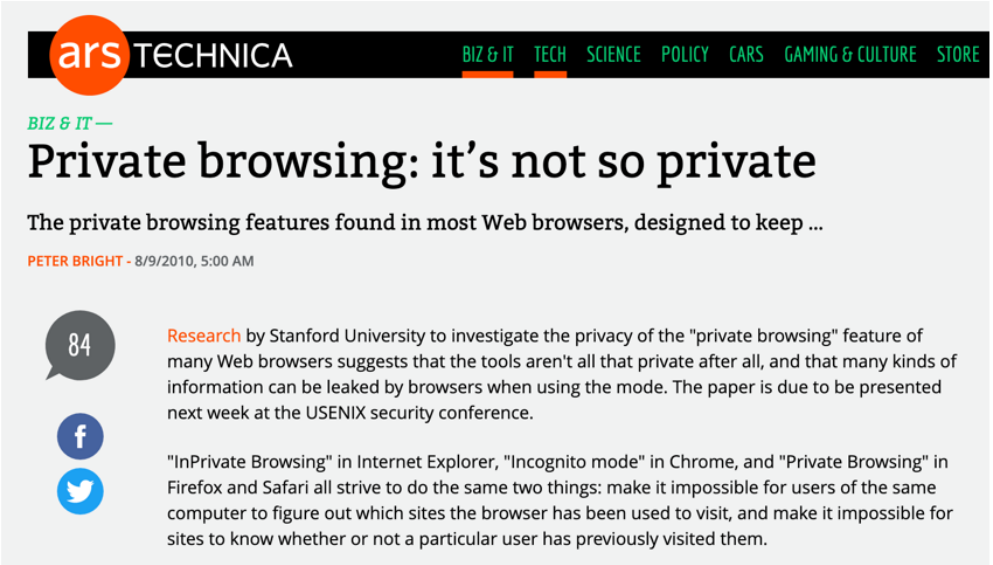


Side-Channel: infer PIN code through motion-detector

Cai, Liang, and Hao Chen. "TouchLogger: Inferring Keystrokes on Touch Screen from Smartphone motion." *HotSec* 2011.

装APP的时候，有些索要的权限很关键。有些看起来不关键的就比如陀螺仪。陀螺仪可以用来判断到底是哪几位密码。

2.13 Privacy



privacy和security还有点不一样。security是有坏人想偷你东西。privacy出发点不一定是坏的，但是可能留下一些日后被用来干坏事的东西。

当我们打开页面的时候，有一个模式是隐私模式。比如我们用携程的时候，担心大数据宰熟。所以我们打开privacy页面。我们的期望是我们打开privacy之后，浏览这个网站，再关掉，就像我们从来没去过这个网站一样。但事实上并不是，有一些数据依旧保存了下来。但倒也不是别人故意做这件事情。

虚拟内存里面有个概念——换页，内存不够了，换到disk上。在访问网页的时候，OS dump数据，当物理内存不够的时候，会把内存文件swap到硬盘上，它不会care上面是隐私文件还是别的文件，因此privacy其实也会留下一些印记。

2.14 Trust

Fighting Club, 1999

社会的正常运转是建立在trust上的。比如你相信明天你的钱不会突然贬值，虽然钱只是服务器上的一个数字。Fighting Club里面最后炸了银行的服务器，这件事情看上去离我们很远；但是事实上，银行真的不会保存这么多纸钞啊！它只是几个服务器上面有些数字而已。

去中心化的货币就是在trust依赖上的进一步减少，不会依赖某一个人某一个国家每一个军队。

我们后面也会讲到bit coin，看它为什么能这么火。

Conflict between these Properties

Usability VS. Privacy

Performance VS. Security

Performance VS. Energy

Fault Tolerance VS. Consistency

...

All these lead to more complexity

3. System Complexity

系统的complexity有一些共性，我们重点强调：

Emergent properties (**surprise!**)

- The properties that are not considered at design time

Propagation of effects (**butterfly effort**)

- Small change -> big effect

Incommensurate scaling (**growing pains**)

- Design for small model may not scale

Trade-offs (**waterbed effect**)

- You cannot sell the cow and drink the milk

3.1 Emergent properties

你永远不知道系统会出什么问题，直到你build出来真正地run它。人类的认知是有限的。

The Millennium Bridge

- For pedestrians over the River Thames in London
- Pedestrians synchronize their footsteps when the bridge sways, causing it to sway even more

- It had to be closed after only a few days

伦敦千禧桥事件。

工程师：“再大的风我们都考虑到了，但是人怎么会做这么非理性的事情！”

Surprise!~

3.2 Propagation of effects

蝴蝶效应。

2. Propagation of Effects [Cole'69]

WHO: tried control malaria in North Borneo

- Sprayed villages with DDT
- Wiped out mosquitoes, but ….
- Roaches collected DDT in tissue
- Lizards ate roaches and became slower
- Easy target for cats
- Cats didn't deal with DDT well and died
- Forest rats moved into villages
- Rats carried the bacillus for the plague

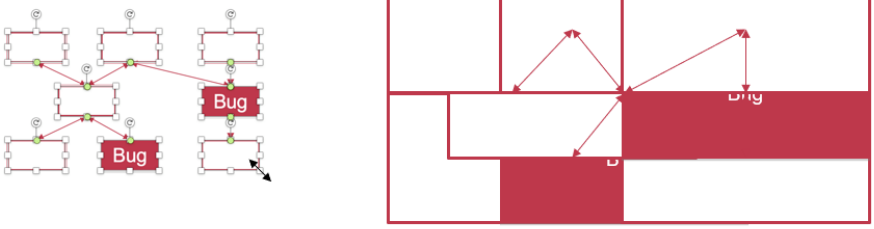
WHO just replaced malaria with the plague

50

- 例子：主机要升级，买CPU发现针脚不支持，然后换主板、换内存，最终只能把整套东西都换掉。
- 例子：屎山，你动了一行代码……

3.3 Incommensurate scaling

Example: Scaling Figures in PPT



Original　　A mess after scaling

一个系统十倍的时候是不一样的。从数学角度来看，一个维度变化的时候，不是系统中的所有组件在以相同比例进行变化。

- 一个班有十个同学，一百个同学，五百个同学，上课是完全不同的上法。

3.4 Trade-offs

General models

- Limited amount of goodness
- Maximize the goodness
- Avoid wasting
- Allocate where helps most

Waterbed effect

- Pushing down on a problem at one point
- Causes another problem to pop up somewhere else

4. Coping with Complexity: M.A.L.H

和传统工程不一样的地方：复杂性的控制没有太多的经验，当我们考虑控制复杂性的时候，经验还是不足的（传统工程：比如造桥，承重是多少的时候，你用什么材料，都是量化的，还有很多case）。我们出生的还是太早。一百年后软件工程的同学们就能利用我们踩的雷了。

为了填补这些空白，我们只能从case study中找到一些visible的特点抽取出来去看现有的系统的怎么做的。

我们需要注意的一点是，system是不能往下深挖的，到某一点我们就需要停下来——你再往下挖还有电路、电子、夸克……这就是控制complexity的维度，如果考虑太细，就导致不可控了。我们从代码深挖到CPU到流水线到电路板上的布局布线，我们考虑所有东西就会使得我们没有办法去思考。所以我们要控制住debug的边界，至少在大部分时候要相信编译器。

M.A.L.H

Modularity

- Split up system
- Consider separately

Abstraction

- Interface/Hiding
- Avoid propagation of effects

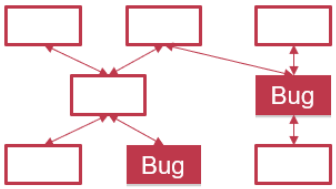
Layering

- Gradually build up capabilities

Hierarchy

- Reduce connections
- Divide-and-conquer

4.1 Modularity



$$BugCount \sim N$$
$$DebugTime \sim N \times BugCount \sim N^2$$

Original System

$$DebugTime \sim \left(\frac{N}{K}\right)^2 \times K \sim \frac{N^2}{K}$$

System with Modularity

Assumptions: consider the number of bugs is proportional to its size, and bugs are randomly distributed 68

当我们有很多模块的时候，就有两种堆法。后面再看。

4.2 Abstraction

Abstraction

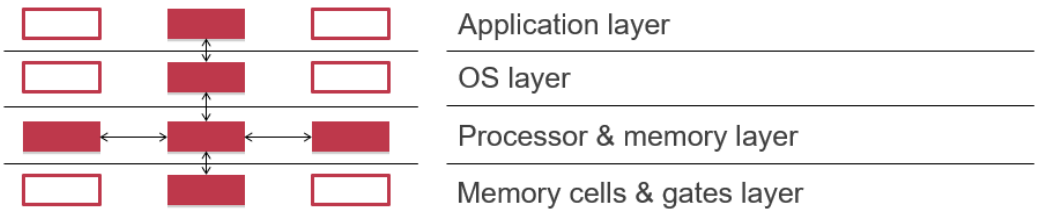
- Treat a module based on external specifications, no need for details inside

Principles to divide a module

- Follow natural or effective boundaries
- Fewer interactions among modules (Chap.4 & 5)
- Less propagation of effects

高内聚，低耦合。

4.3 Layering



House:

- Inner layer of studs, joist, rafter (shape & strength)
- Layer of sheathing and drywall (wind out)
- Layer of siding, flooring and roof tiles (watertight)
- Cosmetic layer of paint (looks good)

Algebra:

- integer, complex number, polynomials, polynomials with polynomial coefficients

模块可以和本层的、上下的层交流。网络就是一个经典的例子。

4.4 Hierarchy

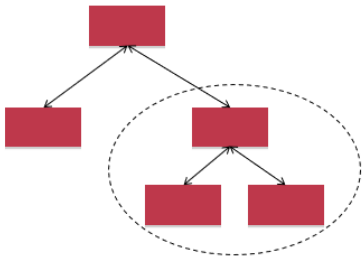
但是层次化本身还不够，我们就需要hierarchy。

Hierarchy: another module organization

- Start with a small group of modules
- Assemble them into a stable, self-contained subsystem with well defined interface
- Assemble a group of subsystems to a larger subsystem

Example

- 1 manager leads N employees
- 1 higher manager leads N lower managers



75

有很多模块的情况下，对外可能只是一个模块的形式存在。

比如我们要连亚马逊的服务器，中间要通过很多路由，比如先找到美国的路由器，让它再帮我们去找，否则我们要把所有的服务器和ip都记录下来。而这样我们就只需要记录到美国的一个地址就可以了。