

Introduction

Performance Engineering: Theory & Practice

Definition

“Software Performance Engineering (SPE)
represents the entire collection of software
engineering activities and related analyses used
throughout the software development cycle,
which are directed to meeting performance
requirements.”

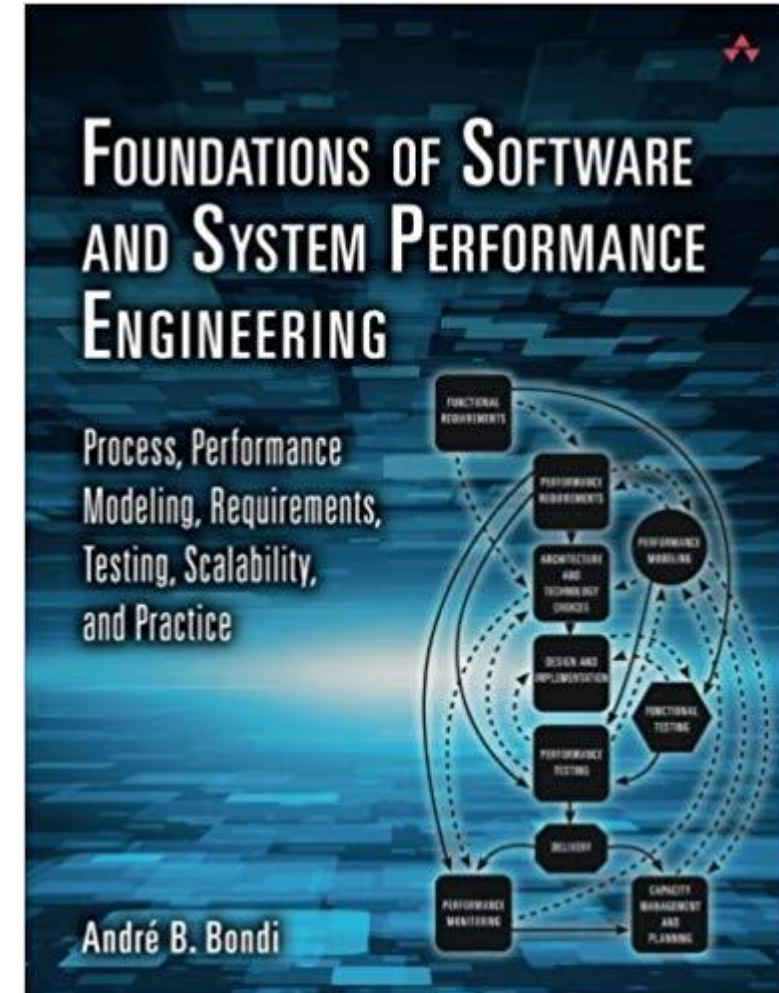
Woodside, et.al., “The Future of Software Performance Engineering,” 2007

Housekeeping

- PowerPoint versions of the slides require installation of the **Barmeno ExtraBold** font to be viewed properly.
- Poll Everywhere software ([link](#))
- Contact me:
 - e-mail: markf@demandtech.com or mbfried@uw.edu
 - call or message my cell: (425) 949-2302
 - office hours: CSE 264, 4-6 pm or TBA
 - my blog: <https://performancebydesign.blogspot.com/>
- Contact Sam: samgao365@gmail.com
- See <https://courses.cs.washington.edu/courses/csep590a/18au/> & follow the **Canvas** link for slides, readings, assignments, etc.
- Distance classroom logistics
- No final exam: Special projects instead

Your first reading assignment

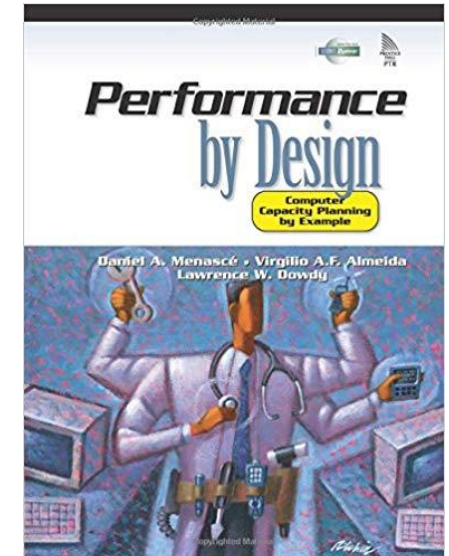
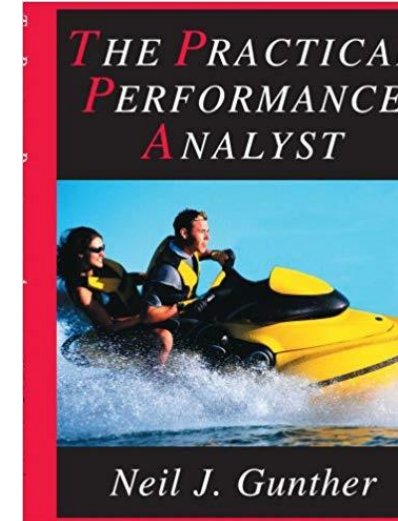
- Read Bondi, ch. 1-3



Your first reading assignment (Optional)

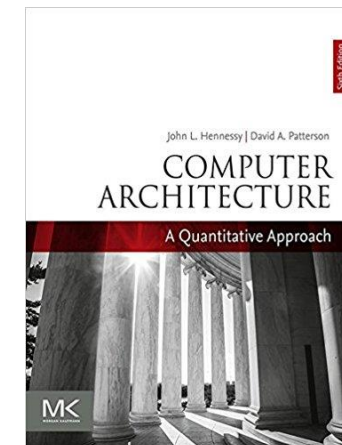
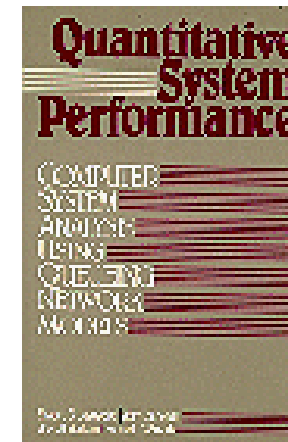
- **Additional Readings:**

- Gunther, *Practical Performance Analyst*, ch. 1-3
- Menasce, *Performance by Design*
- Friedman, "Performance Concepts," unpublished chapter



- **Background Reading:**

- Friedman, *Windows Server 2003 Performance Guide*, ch. 1.
- Hennessy & Patterson, *Computer Architecture*
- Lazowska, et. al., *QSP*



About me

- **Professional software developer: 40+ years**
 - **Master's in CS ~1980**
 - **specialized in performance tool development, beginning around 1984**
 - **Landmark's The Monitor for MVS™ (1989)**
 - **Performance SeNTry, aka NTSMF (1997)**
 - **Architect, Microsoft Developer Division, 2006-2010**

About me

- **Industry analyst and technology entrepreneur**
- **Author and Instructor:**
 - **two books on Windows performance (published 2002, 2005)**
 - **blog**
 - **numerous technical articles published in journals and magazines**
 - **professional seminars, mainly on performance topics**



Is “there is no room in the refrigerator for my stuff” a roommate **performance** problem or a **capacity** problem?

Friedman’s 3rd Law of Storage Management: “No one cleans up their hard drive until it is full.”

Performance Engineering: Theory and Practice

- **Theory**
 - **Algorithms and Complexity (e.g., NP-completeness)**
 - **Queuing Theory**
 - **Resource scheduling**
 - **Probability, Statistics and Data mining: exploratory data analysis**
 - **Feedback & control engineering**

Performance Engineering: Theory and Practice

- **Practice**
 - **Instrumentation & Measurement**
 - **Integration into the software development Life Cycle**
 - **Performance Testing: benchmarking and load testing**
 - **Parallel programming & Concurrency**
 - **Caching**
 - **Web applications**
- **Lots of practical examples, mostly from Windows, using C# & .NET**
- **Guest lecturers**

Performance Engineering: Theory and Practice

- Attendance ([link](#)):
- Grading:
 - Class participation (15%)
 - Homework (25%)
 - emphasis is on your thinking process, not on your getting the right answer, because there is unlikely to be a single right answer
 - Individual or Group project (60%)
 - Challenge yourself!

Definition

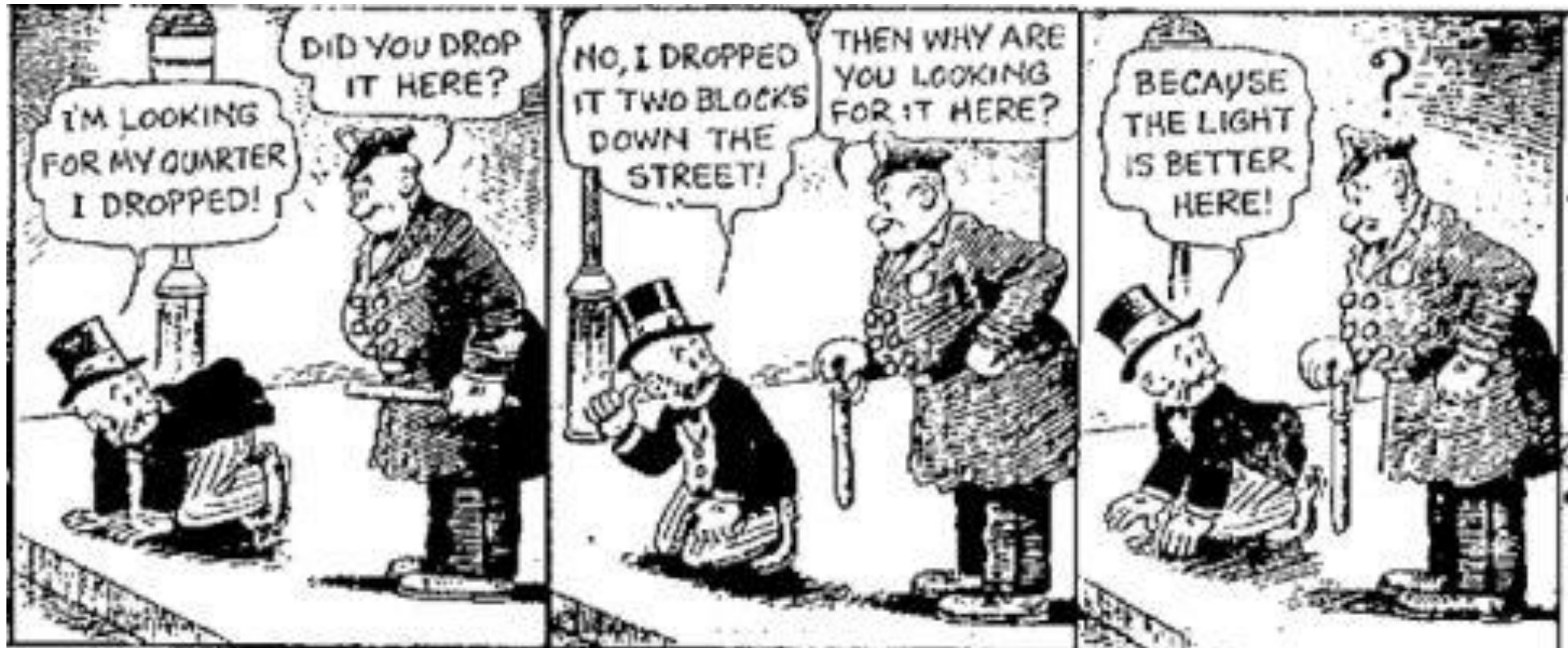
“Software Performance Engineering (SPE)
represents the entire collection of software
engineering activities and related analyses used
throughout the software development cycle,
which are directed to meeting performance
requirements.”

Woodside, et.al., “The Future of Software Performance Engineering,” 2007

About the discipline

- **Performance Engineering: Theory and Practice**
- **The “streetlight effect”:**
 - **based on an old joke about the drunk who is searching for his lost keys:**
 - **“Late at night, a police officer finds a drunk man crawling around on his hands and knees under a streetlight. The drunk man tells the officer he’s looking for his keys. When the officer asks if he’s sure this is where he dropped the keys, the man replies that he thinks he more likely dropped it across the street. Then why are you looking over here? the befuddled officer asks.”**

- **Because the light's better here, explains the drunk man. 😊**



Performance Engineering: Theory and Practice

- **Impact of the “streetlight effect” on performance investigations should be fairly obvious:**
 - **Gaps in the measurement data: leaving important aspects of performance opaque due to a dearth of instrumentation**
 - **Challenge, but also an opportunity**
 - **We may need to adopt tools that were not originally intended for this purpose**
 - **Performance tools typically lag current technology by one or more releases**
 - **Until we have an opportunity to build applications that exploit the new technology, we do not understand what performance tools are required**

About You

- **Class survey**
 - **How many classes in the PMP have you taken?**
 - **How many are professional software developers?**
 - **What programming languages do you use?**
 - **C++, C#, Java, JavaScript, Python, Ruby, R, assembler, etc.**
 - **What platform do the applications you work on target?**
 - **LAMP, MacOS, iPhone, Android, ASP.NET, etc.?**
 - **What performance tools are you familiar with?**

About You

- (Optional)
- Your first assignment is to tell me a little about you and why you are taking this class (other than I need to take **something** in order to graduate), including:
 - What do you hope to gain from the class?
 - Do you have any background or experience in the subject matter?
 - Are there topics/subject areas that you are particularly interested that the class should cover?
- 500 words or less
- e-mail me at markf@demandtech.com or mbfried@uwa.edu

Performance Engineering Overview

My highly personal view of the discipline

What is system or application performance?

- **Responsiveness**

- Real-time control applications often have hard limits on how long they can wait before they must react to current conditions
- Consistent response times establish a pattern for human-computer interaction

- **Throughput**

- Real world business requirements generate a workload

- **Scalability**

- How many customers for an e-business application?
- How many devices for an IoT application?

What is system or application performance?

Consider an e-business application...



- **Responsiveness**
 - How long does it take to respond to a customer request?
- **Throughput**
 - How many requests/second does the application need to process?
- **Scalability**
 - How many concurrent customers?

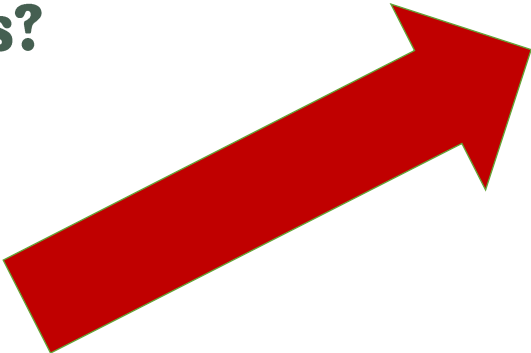
What is system or application performance?

Consider some mobile applications...

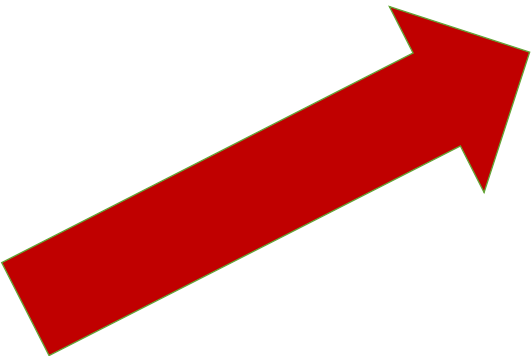


- **How long is a customer willing to wait...**
 - **to debit or credit a checking account?**
 - **to receive notification about the movement of a stock price?**
 - **to update a work order in order to proceed to the next queued work item?**

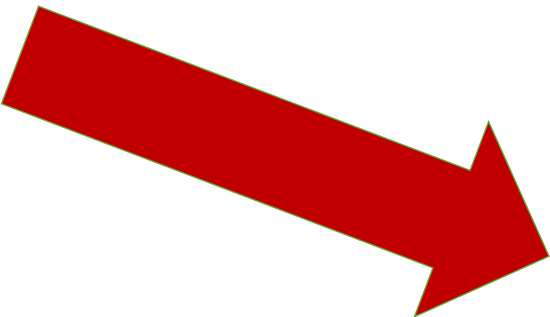
What is system or application performance?

- **What if?**
 - **You are a service bureau and your customers are banks?**
 - **You are a bank and you are losing customers to a rival that has a superior mobile app?**
 - **How long is a customer willing to wait...**
 - **to debit or credit a checking account?**
 - **to receive notification about the movement of a stock price?**
 - **to update a work order in order to proceed to the next queued work item?**
- 

What is system or application performance?

- **What if?**
 - **You are a company of professional stock traders?**
 - **How long is a customer willing to wait...**
 - **to debit or credit a checking account?**
 - **to receive notification about the movement of a stock price?**
 - **to update a work order in order to proceed to the next queued work item?**
- 

What is system or application performance?

- **What if?**
 - **a slow application affects the productivity of your workers, which impacts the company's bottom line?**
 - **How long is a customer willing to wait...**
 - **to debit or credit a checking account?**
 - **to receive notification from a stock trade?**
 - **to update a work order in order to proceed to the next queued work item?**
- 

What is system or application performance?

- Performance is a very important and often critical aspect of **software quality!**
 - Performance engineering is a discipline within software engineering
- Performance is often highly correlated with User satisfaction
 - Poor and/or erratic response times are a huge dissatisfier
- Performance (or **service levels**) can be measured; many other aspects of software quality are not so readily quantified

Why do so many large scale software development projects **fail when it comes to meeting their performance requirements?**

Performance engineering is a hard problem!

- **Complex software development is risky!**
- **Development projects can **fail** when it comes to meeting their performance requirements due to many reasons:**
 - **Scalability requirements are not understood or well-defined**
 - **Performance is not emphasized early enough in the development process**
 - **Time constraints and budgetary considerations**
 - **Hardware constraints**
 - **Organizational emphasis on fire-fighting, instead of more pro-active approaches**

Performance and the Development Life Cycle

- **Why aren't performance concerns incorporated into the fashionable software development methodologies?**
 - **Performance is considered a “non-functional” requirement of an application**
 - **Yet, meeting performance requirements is often a key success factor.**
 - **Still...**
 - **no mention in Design Patterns**
 - **seldom considered when people discuss Use cases**
 - **Rarely comes up when people are teaching Agile**
 - **even though it make good sense to annotate a scenario with its performance requirements**

Performance and the Development Life Cycle

- **Concerns about “premature optimization” that defers tuning efforts until the code base is stable are valid – up to a point**
- **Senior technical staff are heroes that parachute into investigations to fix performance problems in the latter stages of a project**
- **But if there is a fundamental design flaw that was baked in early...**
 - **More expensive to fix it in the later stages**
 - **a serious enough “flaw” can delay (or even torpedo) the entire release**

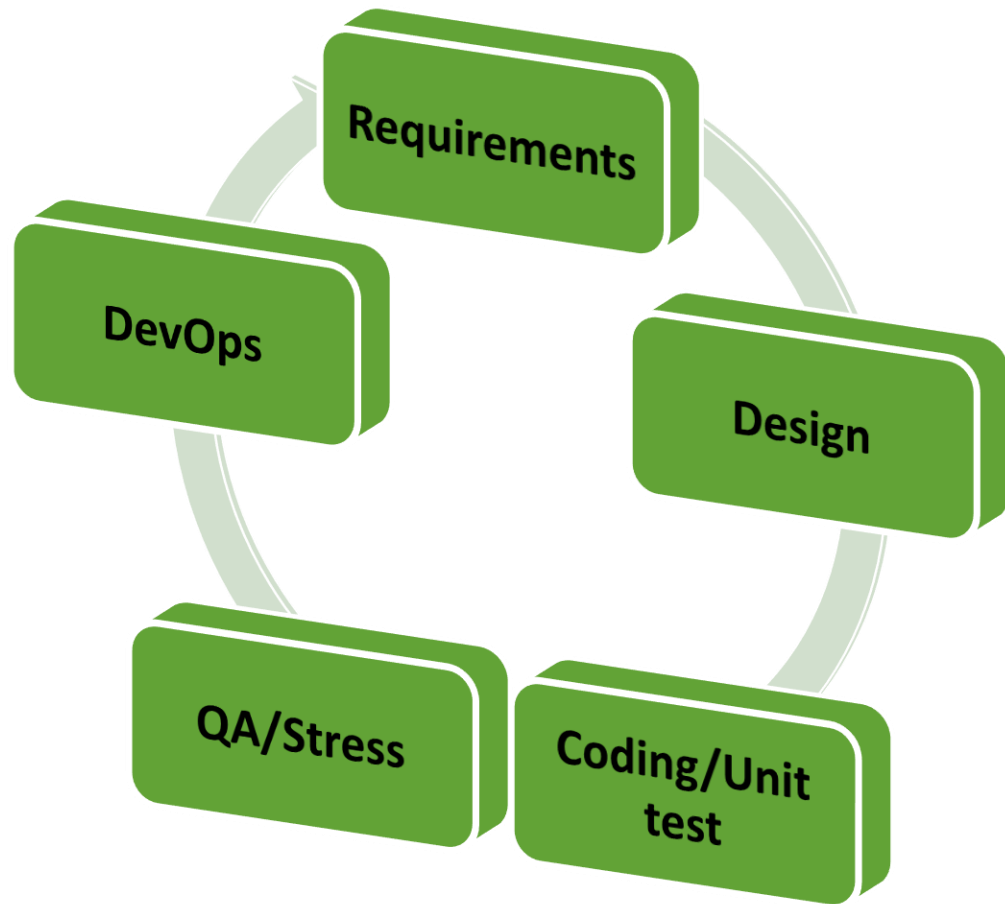
Performance and the Development Life Cycle

- Here are some things that have been tried:
 - Performance “anti-patterns” approach
 - single-lane bridge
 - long path
 - resource bottlenecks
 - e.g., Resource ***R*** is a candidate bottleneck if:
 1. it is used by the majority of scenarios,
 2. many scenarios that use it are too slow,
 3. it is near saturation (>80% of its units are busy),
 4. resources that are acquired earlier and released later than ***R*** are also near saturation
 - layered software bottlenecks

Performance and the Development Life Cycle

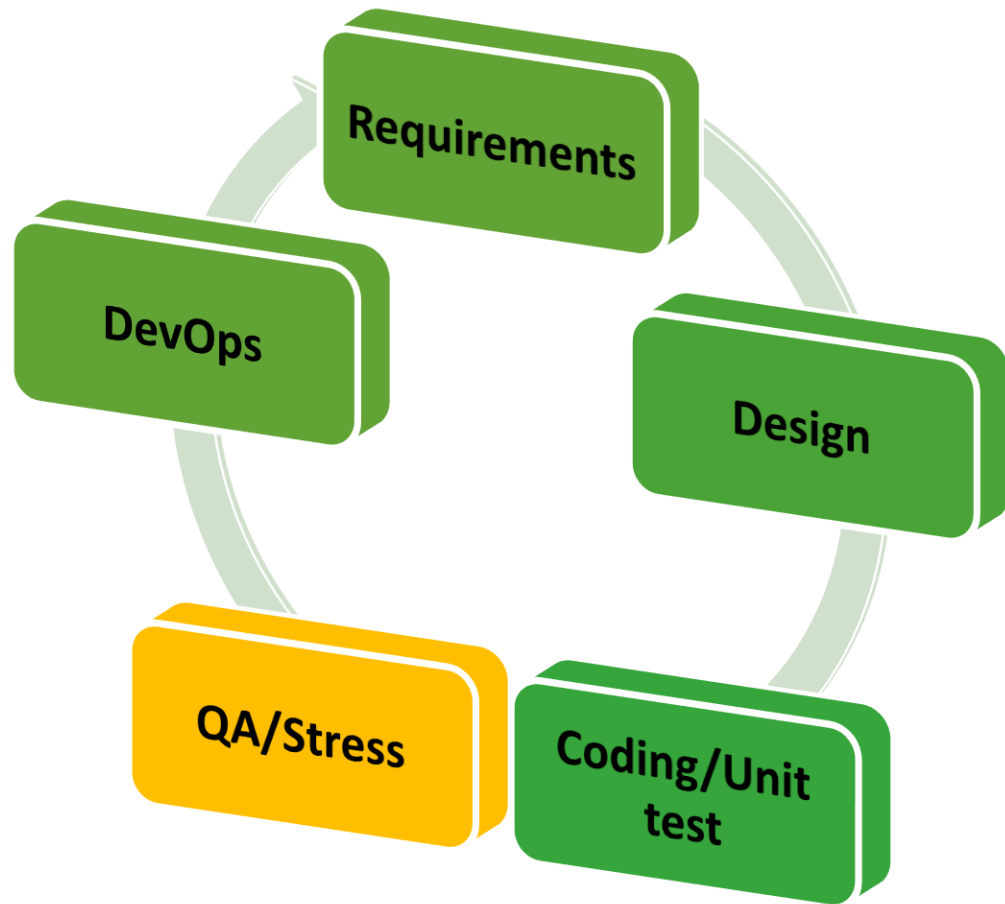
- **Here are some things that have been tried:**
 - **Generate a model from the specification**
 - **Markov models of sequence, to queueing models**
 - **annotated UML \Rightarrow Queueing model**
 - **Issues:**
 - **Validation**
 - **Use static analysis to parameterize a model rather than wait for actual run-time measurements**

Performance and the Development Life Cycle



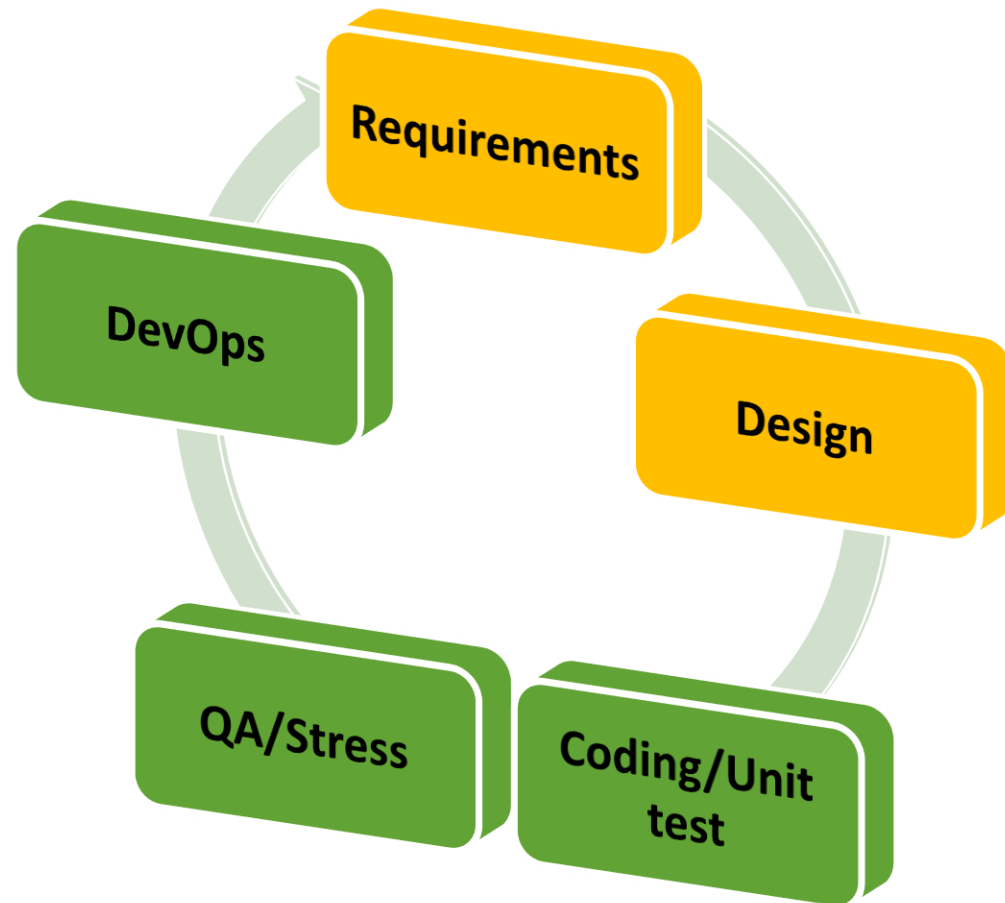
- **Day of Reckoning** when a major release misses its performance objectives by a wide margin
 - Can lead to adding a performance stress test step to assess risk prior to release
- ***Pro-active*** performance management
 - Monitor and report on progress/risk against performance objectives throughout the life cycle

Performance and the Development Life Cycle



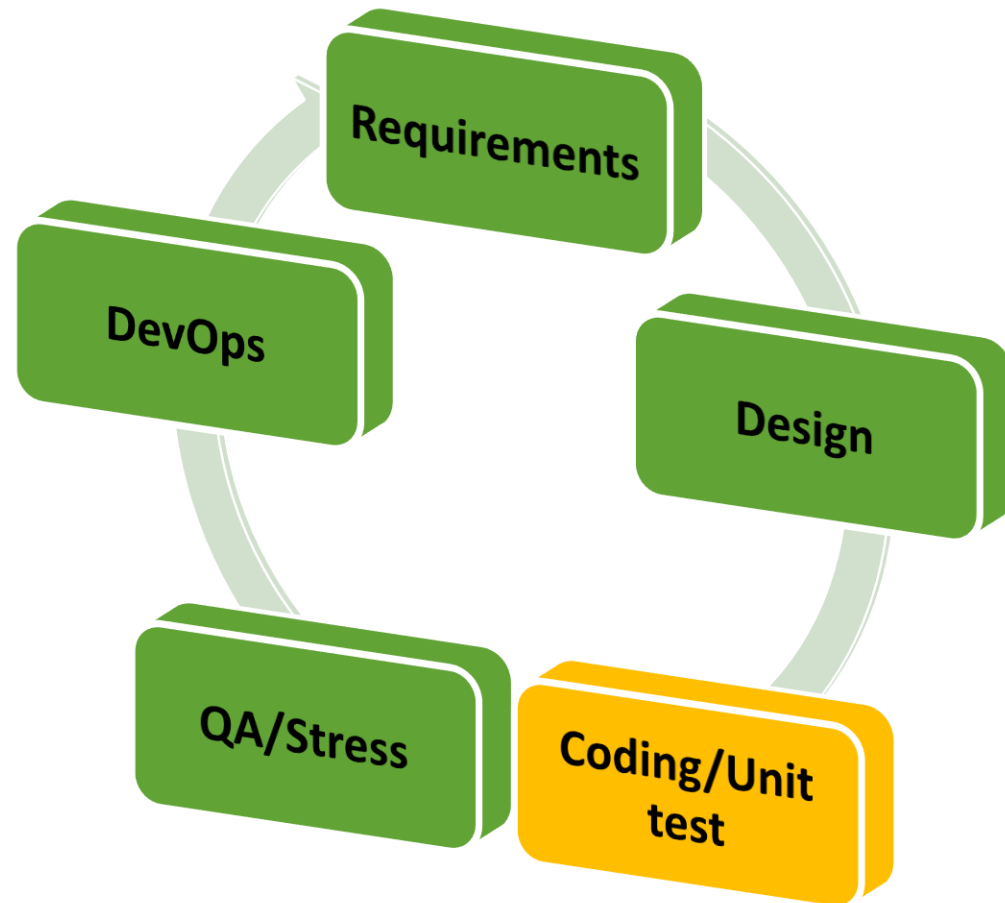
- **Preventative** performance management
 - Sometimes following a particularly “heroic” tuning effort that finally brings the performance of the release into an acceptable range
- **Continuous improvement model**
 - Monitor and report on progress/risk against performance objectives throughout the life cycle

Performance and the Development Life Cycle



- **Continuous improvement model**
 - Set *achievable* Performance goals initially based on requirements
 - So they can inform design decisions (and early stage scouting)

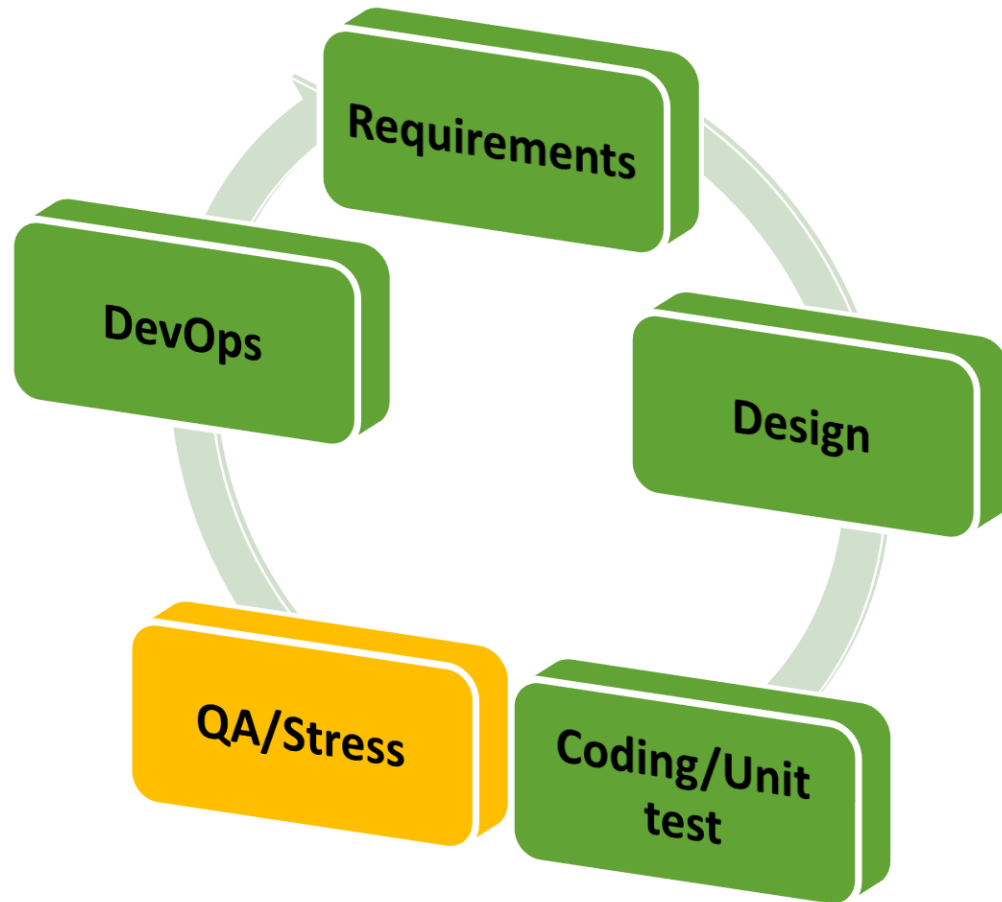
Performance and the Development Life Cycle



- **Continuous improvement model**

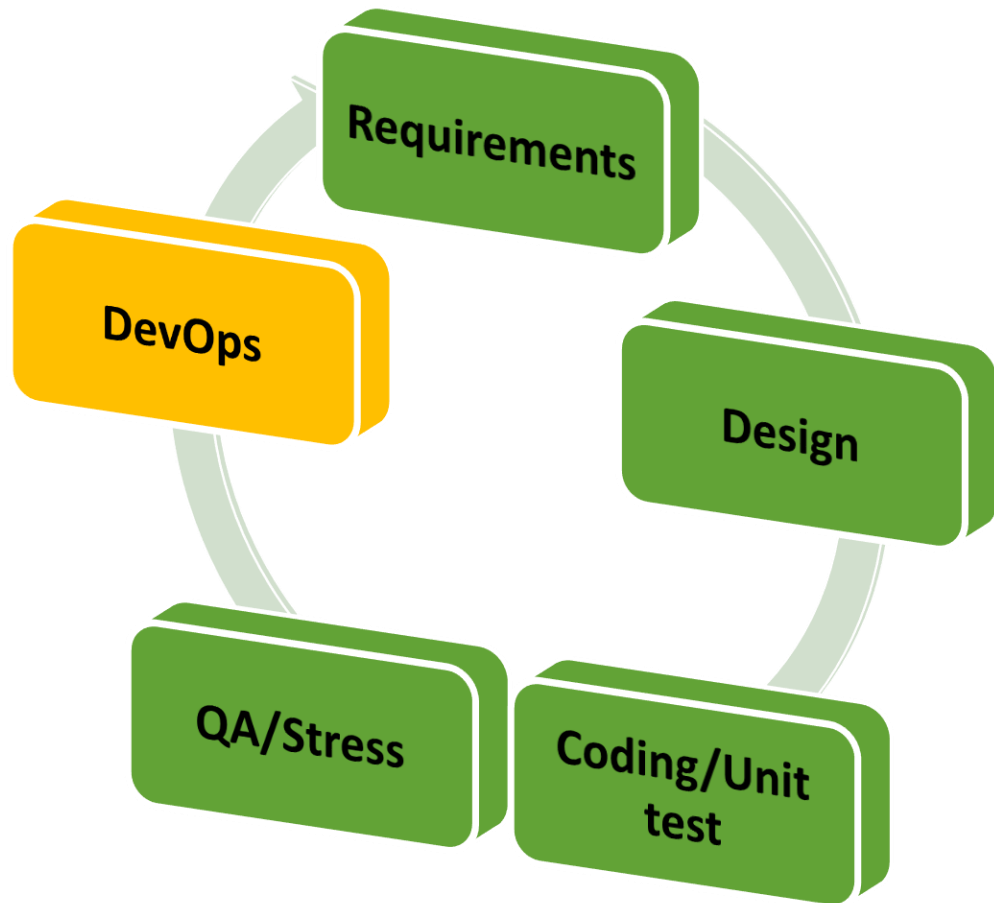
- **Performance tests performed early and often to monitor progress against goals**
- **Automated performance testing**
 - **Instrument early and test often**
 - **Every unit test can also be a Timing test!**
- **Performance Quality gates to detect problems **prior to integration****

Performance and the Development Life Cycle



- **Continuous improvement model**
 - **Full scale load/stress testing**
 - **Evaluate the cost of embedded instrumentation**

Performance and the Development Life Cycle



- **Continuous improvement model**
 - **Service level reporting**
 - **Management by Exception**
 - **Statistical Quality Control techniques**
 - **Embedded instrumentation**
 - **Diagnostic tools to drill into problems on demand**

Fundamentals of software performance engineering

- **Problems of Scale**

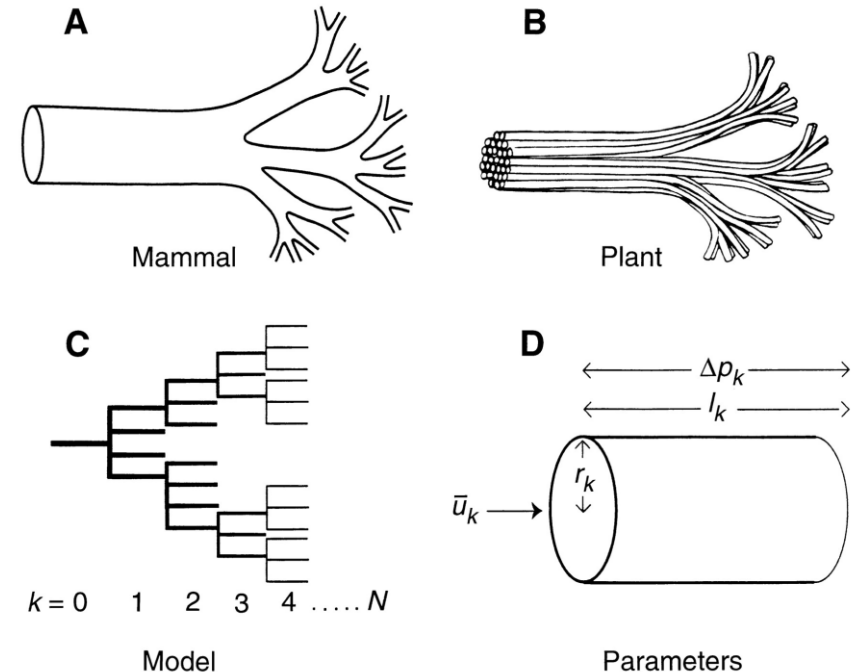
- **Algorithms and Complexity**

- **Hardware capacity limitations**

- network latency
 - parallel programming
 - resource sharing & queueing
 - n-tiers, clustering

- **Cost/Performance trade-offs**

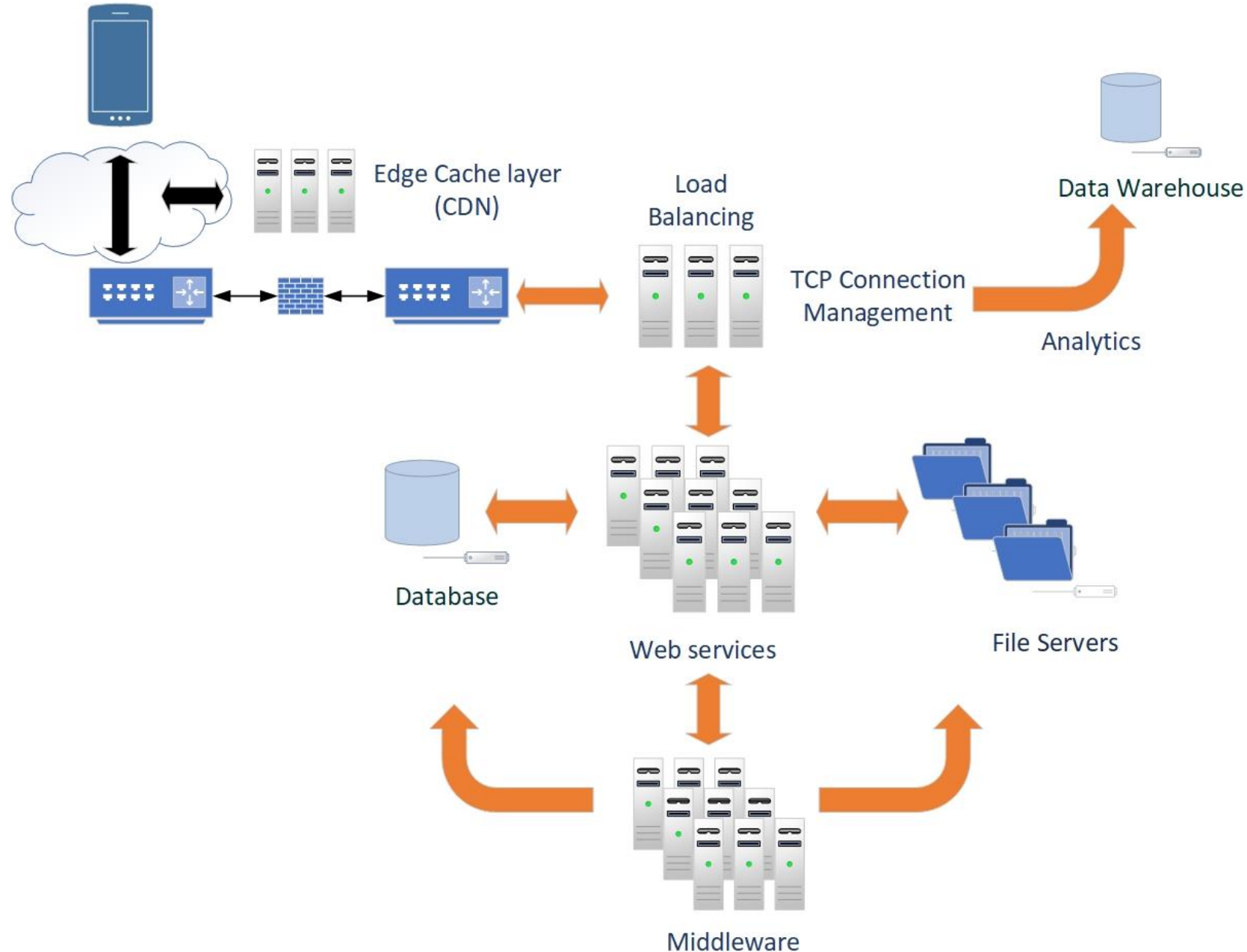
- e.g., virtual memory, Caches
 - energy & power consumption



from Geoffrey B. West, et. al., "A General Model for the Origin of Allometric Scaling Laws in Biology," *Science*, April 1997

Web Application Scalability

- **Architecture**
- **Management**
- **Measurement**
- **Cost/Performance trade-offs**
- **Growth; capacity planning**
- **Continuous improvement**



IoT Scalability

- Hardware capacity & performance
- Network performance
- Analytics

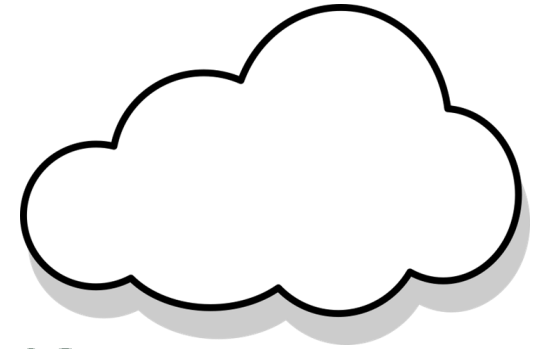


Datacenter provisioning

- **Cloud computing**
 - both on and off premises
- **Virtualization**
 - servers
 - SANs
 - software-defined Networks
 - fungible resources:
 - capacity planning replaced by “provisioning”
- **Containers and microservices**
- **DevOps**



Cloud computing



- **Either,**
 - **3rd party data centers where massive resources are available on demand**
 - **on-premises cloud option when very sensitive data is involved**
- **Performance characteristics of workloads readily shifted to run off premises in the “cloud”**
 - **customer-facing web services**
 - **geographically distributed**
 - **highly elastic demand (e.g., FIFA World Cup video streaming)**
 - **new applications that anticipate rapid scale up (e.g., IoT)**

Prospects for Automating Computer Performance

How do we scale performance management to meet these challenges?

Automating performance management

- **Automated optimization & tuning approaches**
 - **Manual tuning does not scale to the quantity of computer resources that need to be managed today**
- **What are the prospects for autonomic computing?**
 - **Machine Learning**
 - **Anomaly detection**
 - **Bottleneck detection needs to be informed by analytic models**
 - **Feedback and control engineering**
 - **success factors:**
 - **clear, unambiguous measurements**
 - **resources that can be provisioned dynamically**

- **Bill Gates, Elon Musk, Jeff Bezos, Eric Schmidt, and John Hennessy meet at Davos and decide to pool their considerable wealth and intellect to build the world's most powerful supercomputer to answer the question,**



“What is the meaning of life?”

“What is the meaning of life?”



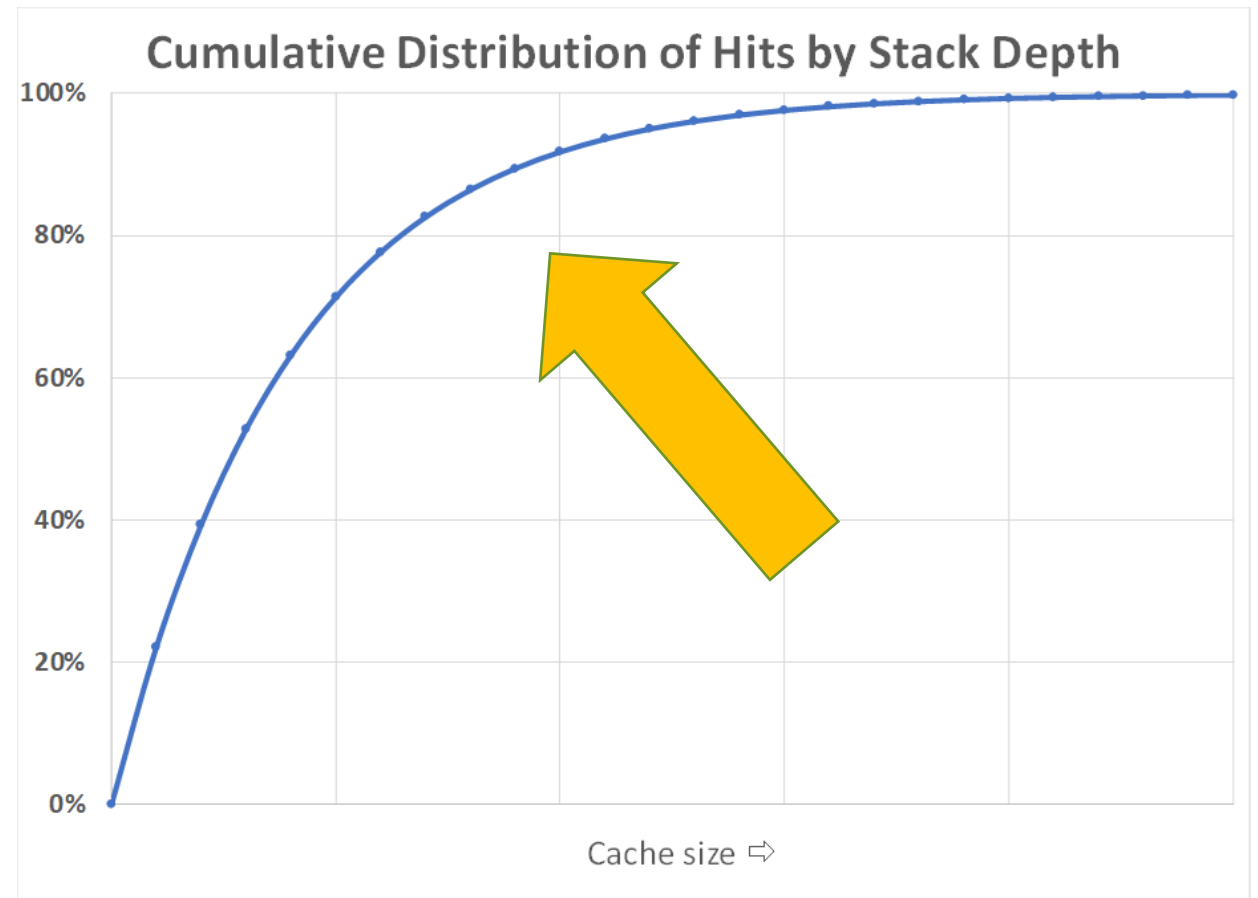
**Let me tell
you a
story...**

So, that reminds me of a story...

- In the mid-1990s, I decided to investigate a claim in the 1st edition of the book, *Inside Windows NT*, by Helen Custer, that the file cache feature in the OS was “self-tuning.”
- Microsoft Windows NT was a new OS, created from scratch by a team led by David Cutler, formerly the developer Lead for Digital’s VAX/VMS operating system
- In 1996, I was beginning to invest in building performance monitoring software for the Windows NT platform

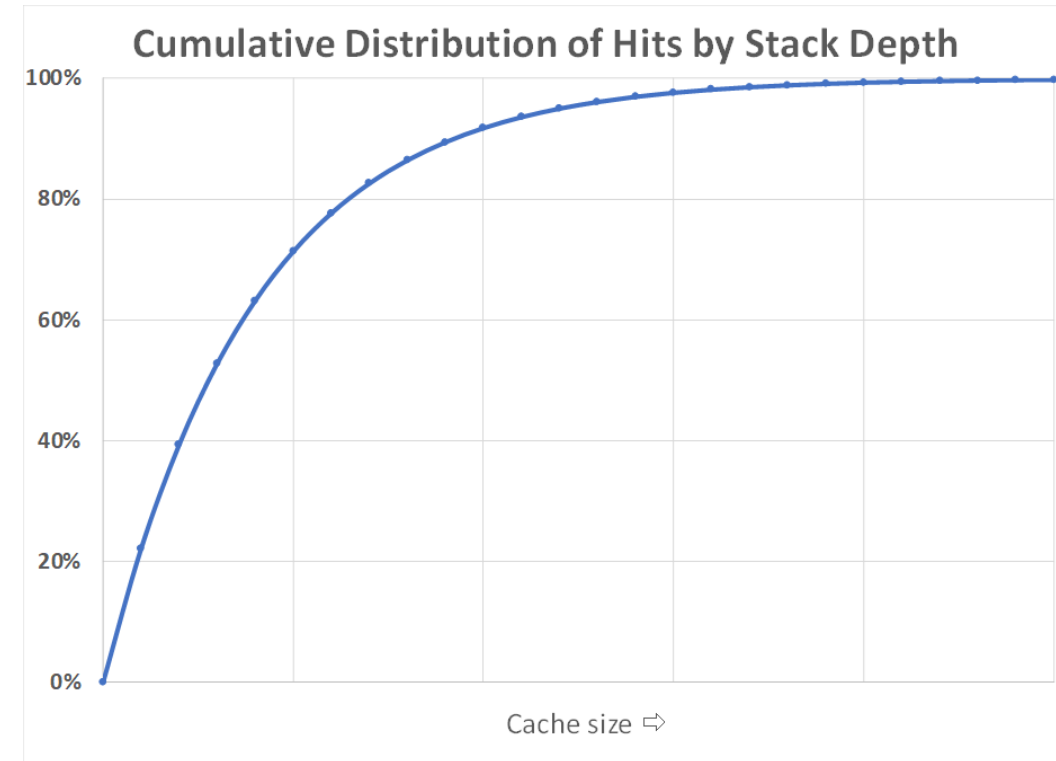
Is the Windows NT file cache “self-tuning” ?

- In principle, an intelligent cache could size itself dynamically
- by calculating the shape of its distribution of the cache hit ratio : cache size distribution
- and making appropriate adjustments



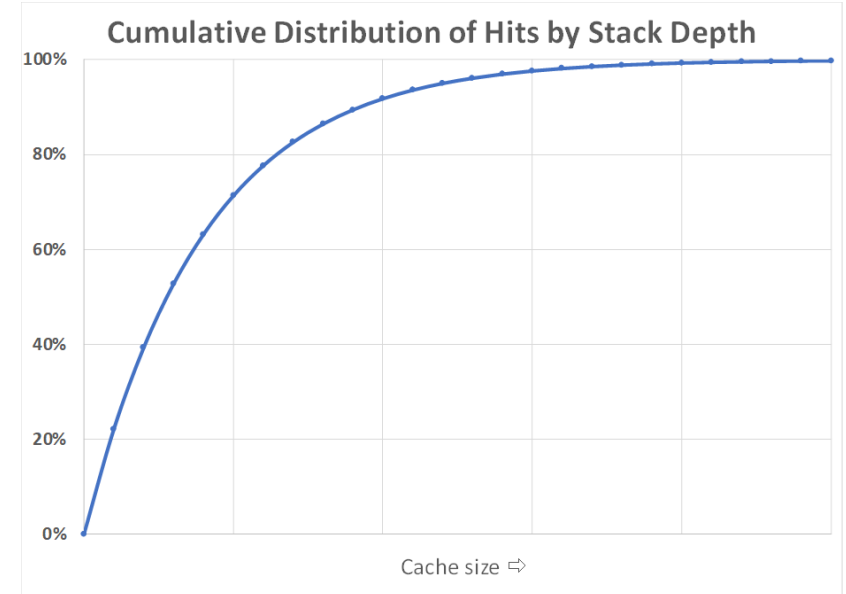
Disk cache and me

- In the early 1980s, while working for a hardware vendor, I had the opportunity to investigate the effectiveness of emerging commercial disk cache technology.
- See Alan Jay Smith, “Disk Cache”, ACM TOCS, 1985.



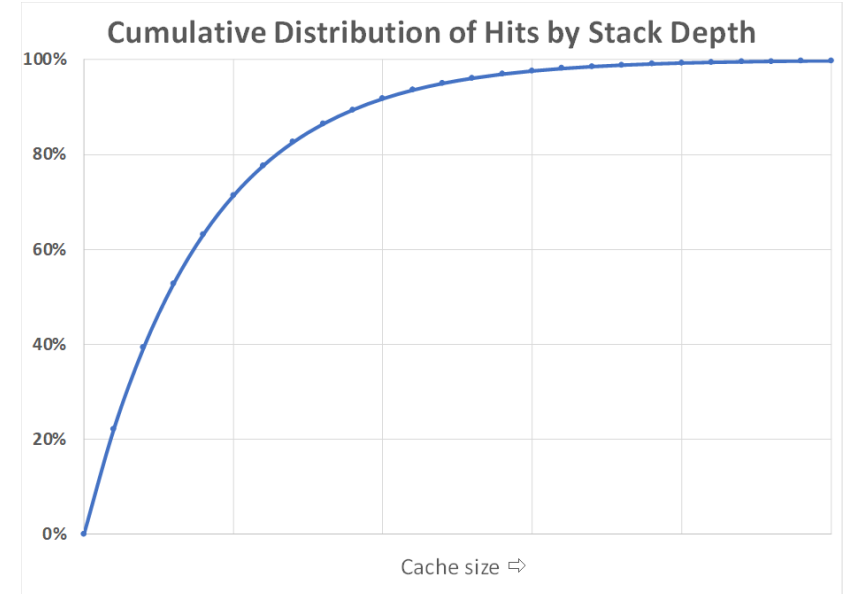
My trace-driven, disk cache simulator program

- **Initially, I gathered IO traces from several customers**
- **LRU Stack depth (hit ratio by cache size) distributions for disk cache were the same shape as virtual memory and processor caches**
- **These simulations demonstrated that disk cache was a very effective IO accelerator for commercial, transaction-processing workloads**



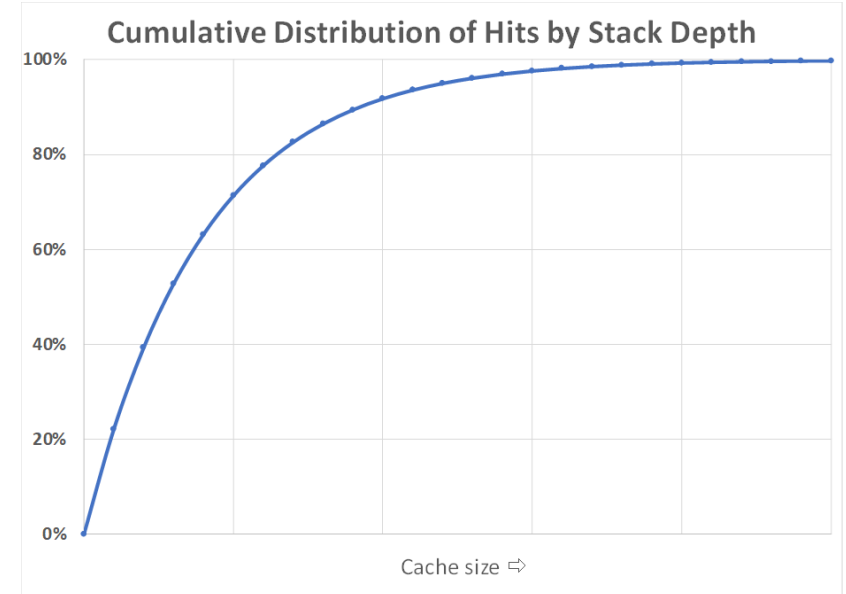
My trace-driven, disk cache simulator program

- **Counter-intuitive result that required further investigation:**
 - the back-store consisted of rotating magnetic disks known as **Direct Access Storage Devices (DASD)**
 - hardware support for direct Seeks to a random disk location on the disk platter
 - to simplify modeling, an average hardware Seek was defined as $\frac{1}{3}$ the maximum Seek distance
 - Performance tools that used data from IO traces to optimize file placement on disk to **minimize seek distance** were in wide used



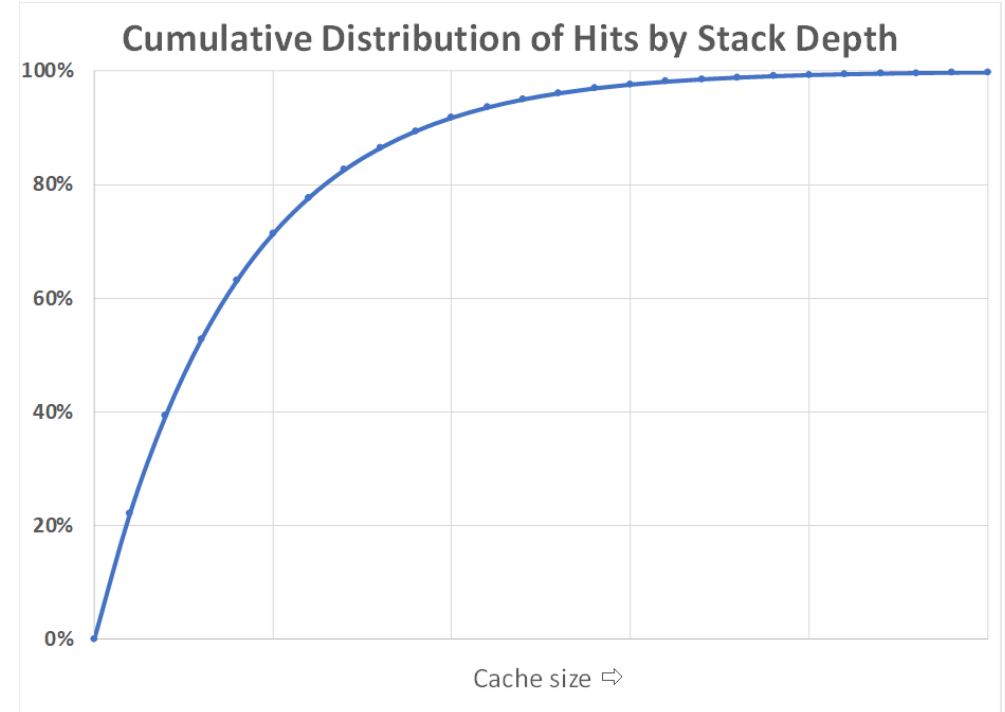
My trace-driven, disk cache simulator program

- Of course, the use virtual memory management was also controversial when it was first introduced!
 - see [Denning](#)
- I published results from my disk cache simulation program in 1983, at which time I also had data that confirmed these speculations empirically from several customers



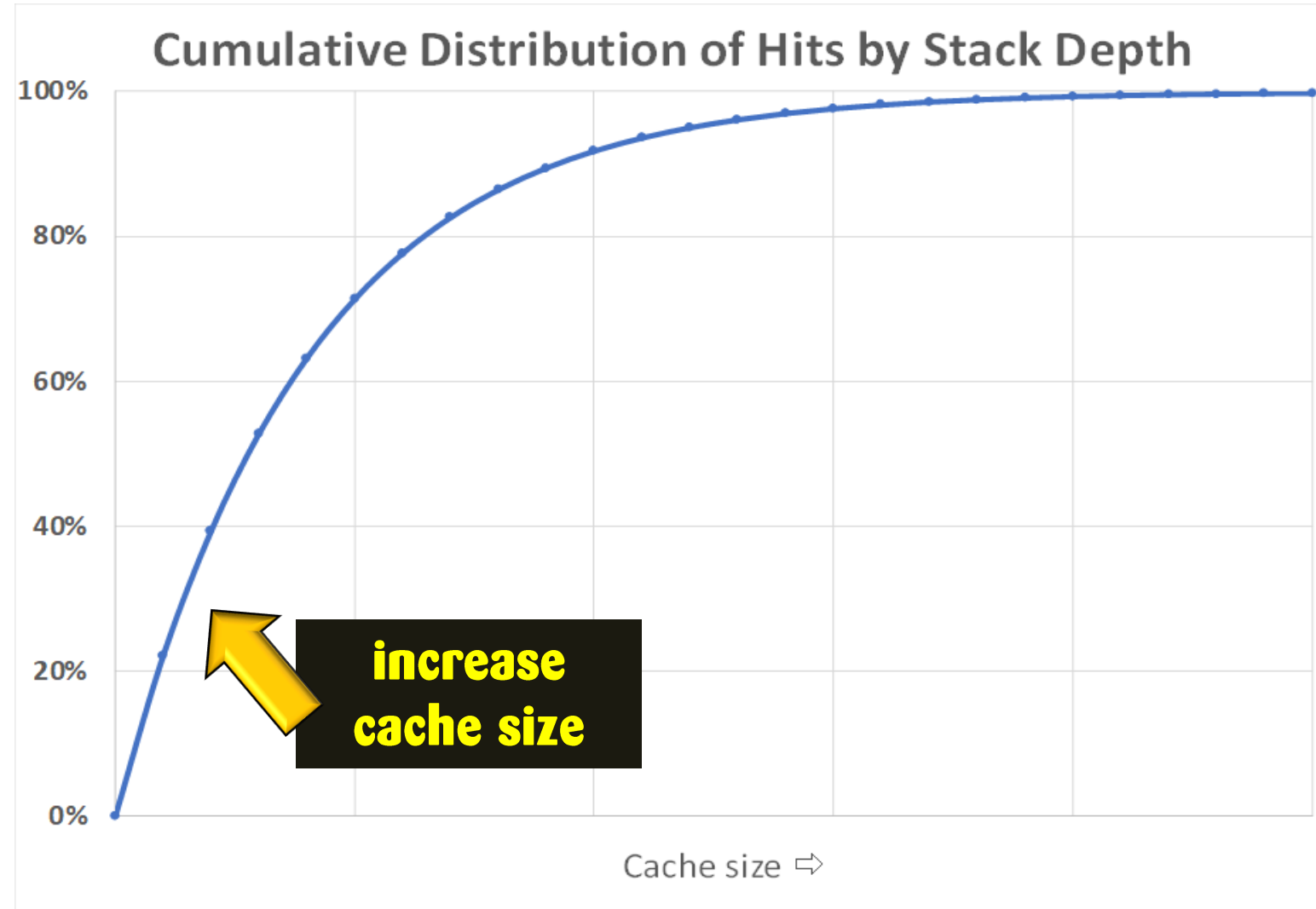
“self-tuning” dynamic cache sizing

- By 1990, there were four well-understood examples of cache technology in wide use
 - virtual memory (**Denning**)
 - working set model of program behavior
 - CPU cache (**Smith**)
 - disk cache
 - relational database memory buffering
- If cache sizes adjustments were possible, decision-support for good dynamic adjustments in real-time seemed achievable



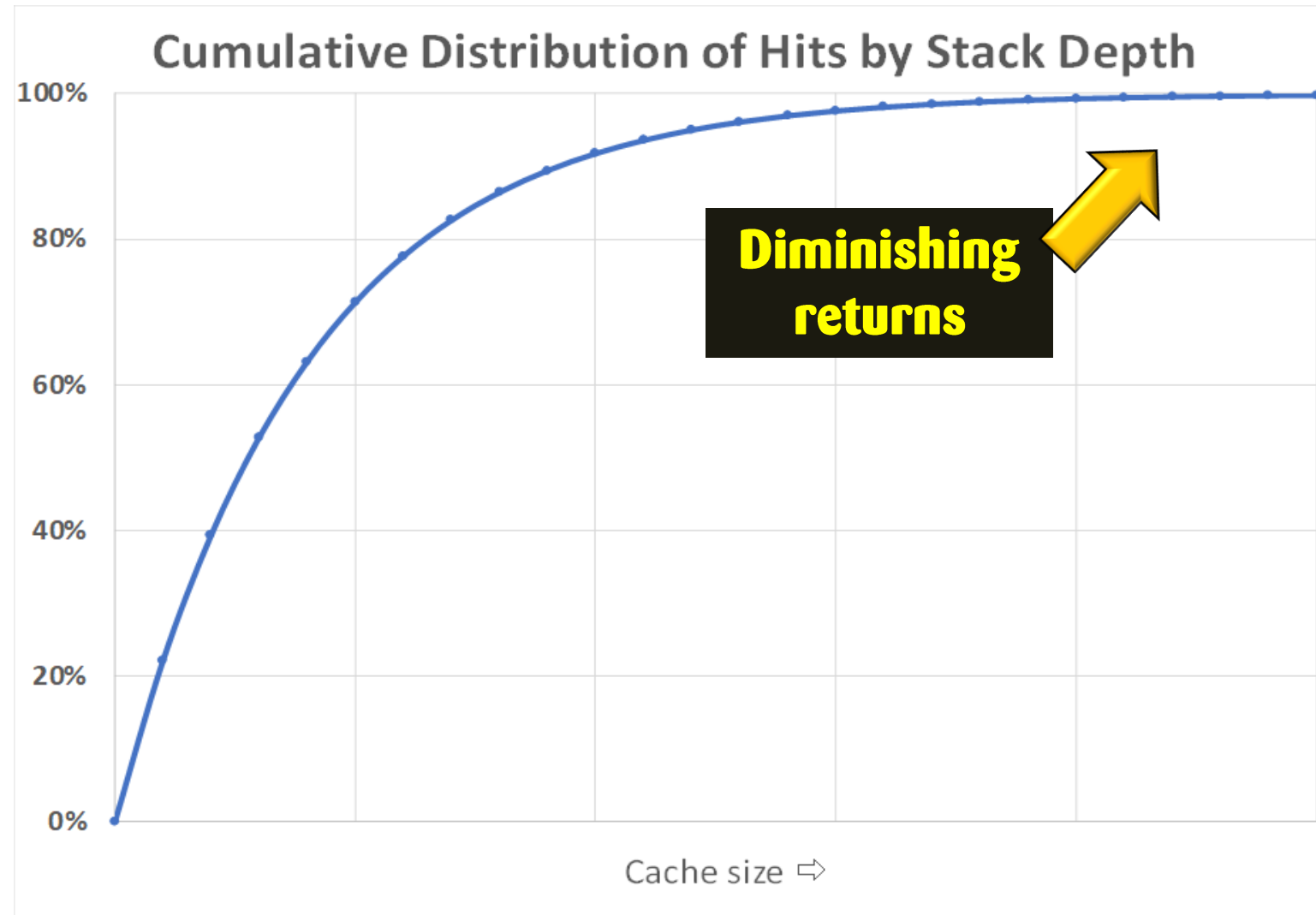
The shape of the cache hit : cache size distribution

- When **small**, incremental changes in cache size cause **large** fluctuations in cache hit ratios, then the cache is too small
- slope $\gg 1$



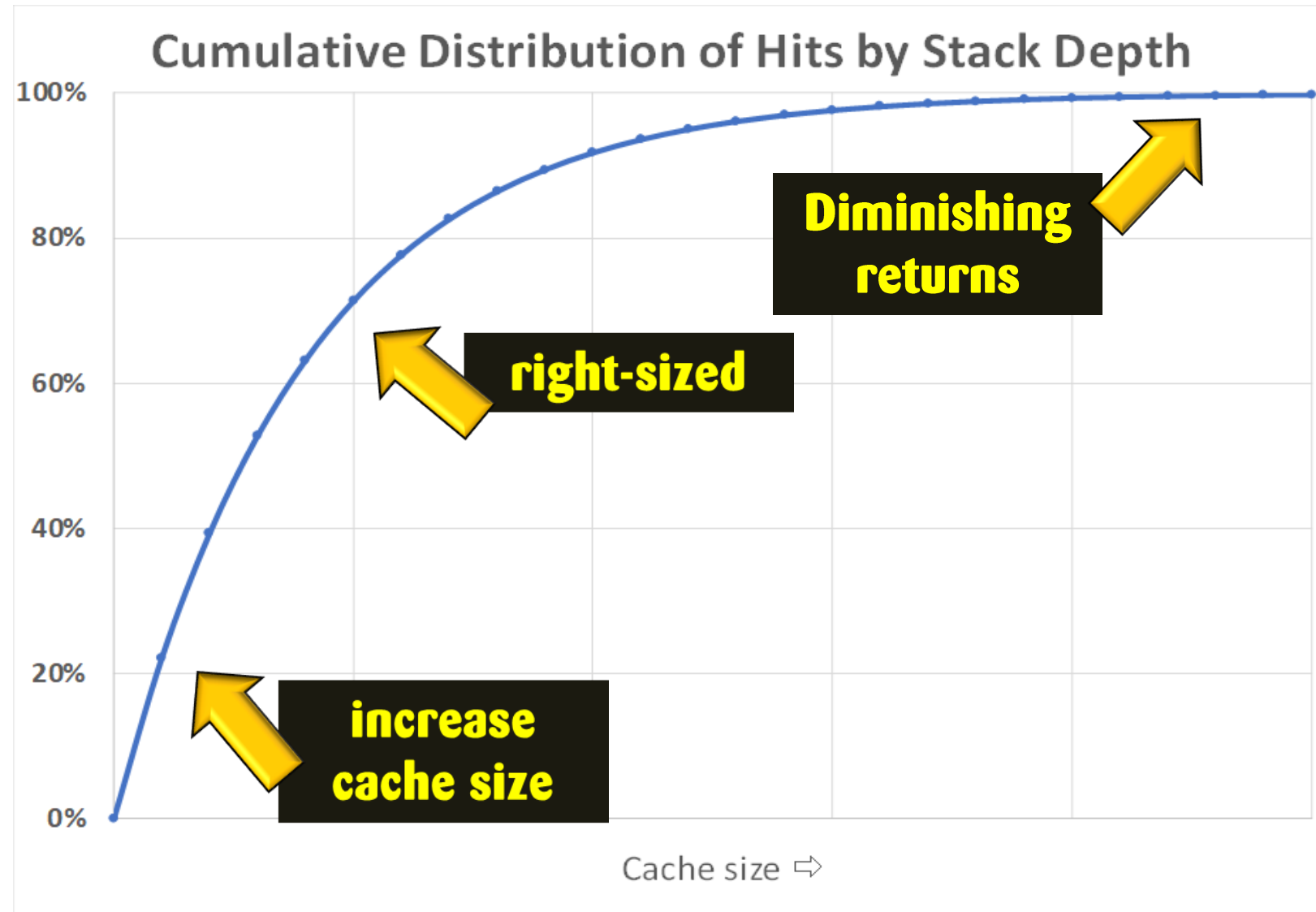
The shape of the cache hit : cache size distribution

- When large, incremental changes in cache size cause only minor fluctuations in cache hit ratios, then the cache is larger than necessary
- slope $\Rightarrow 0$



The shape of the cache hit : cache size distribution

Cache is right-sized
when slope $\Rightarrow 1$

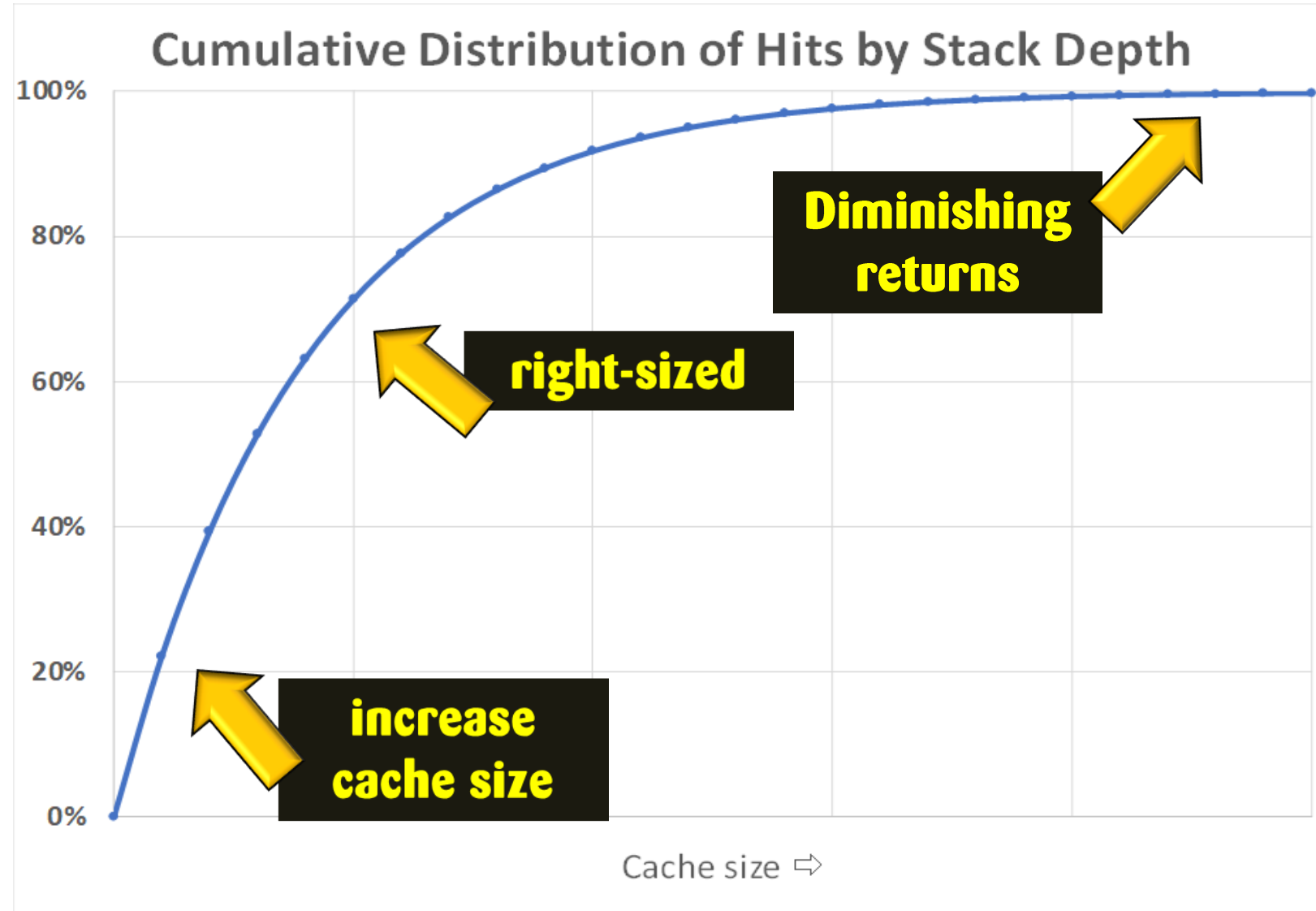


The shape of the cache hit : cache size distribution

A dynamic approach to regulating the size of the cache:

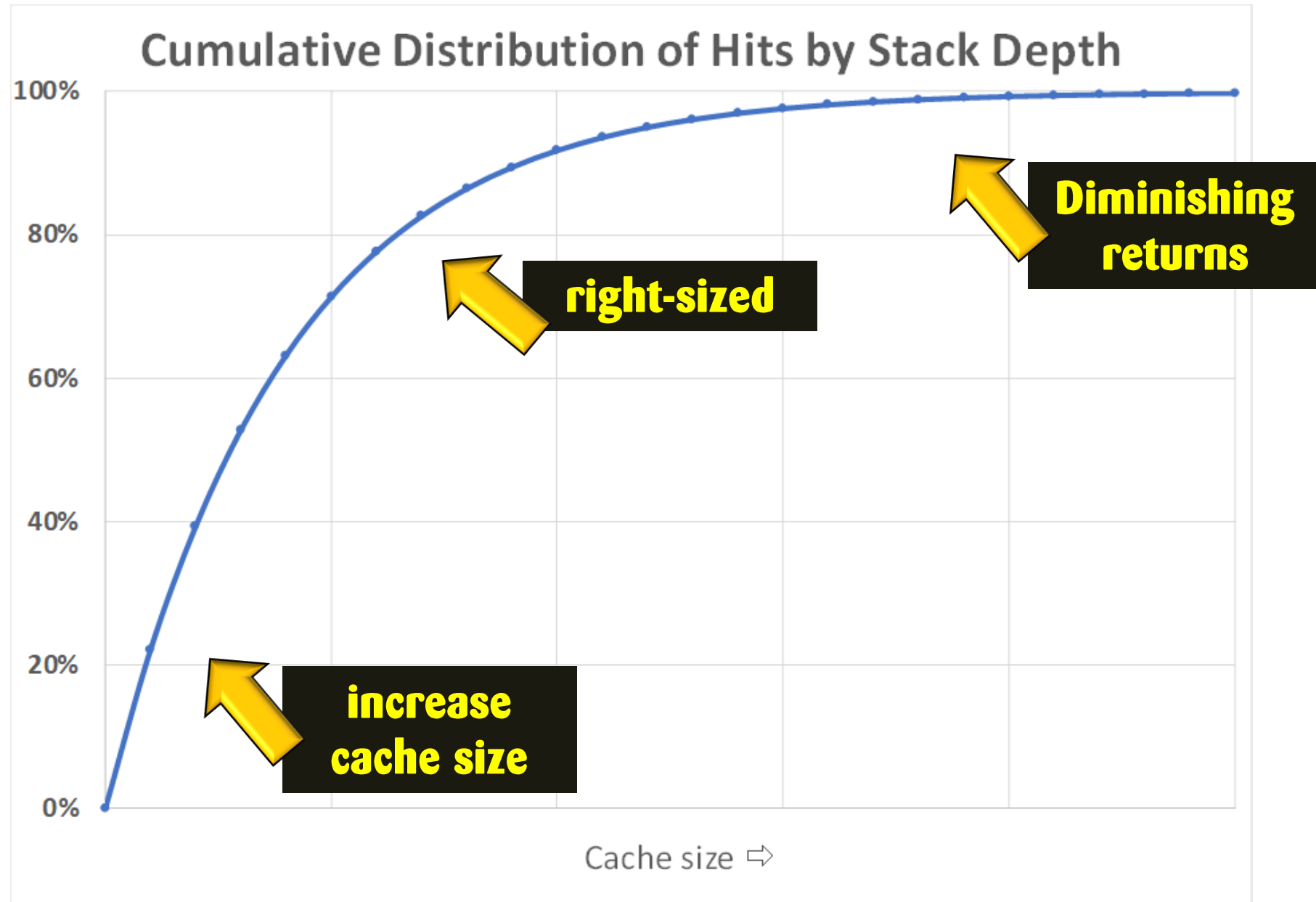
- retain cache miss vs. size recent history
- attempt to derive the full distribution
- track a gradient from the current size to an optimal size

seems feasible (or even desirable)



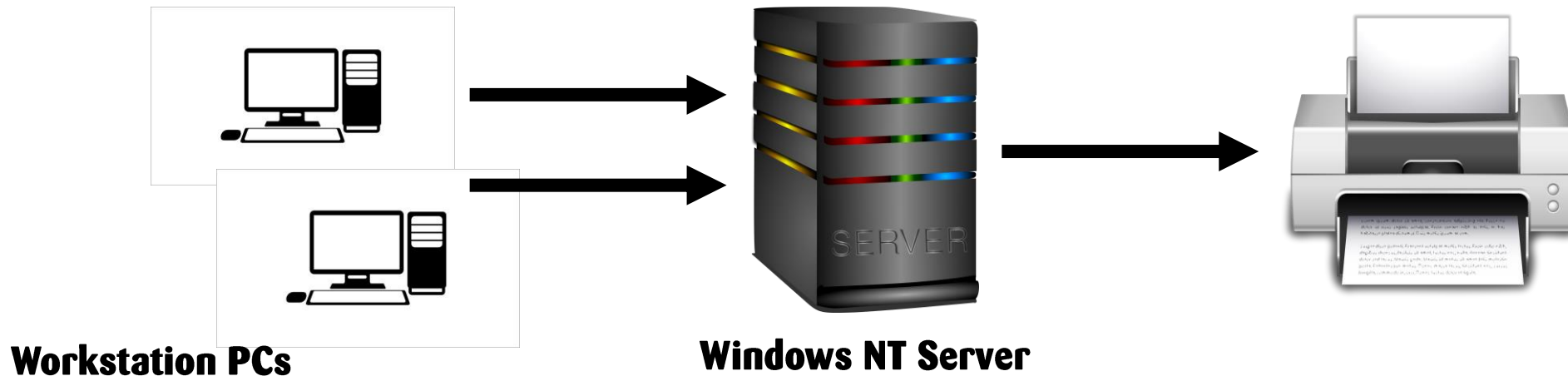
Was this what the “intelligent” file cache management function inside Windows NT was doing?

- If so, IMHO, it would have been a considerable achievement!

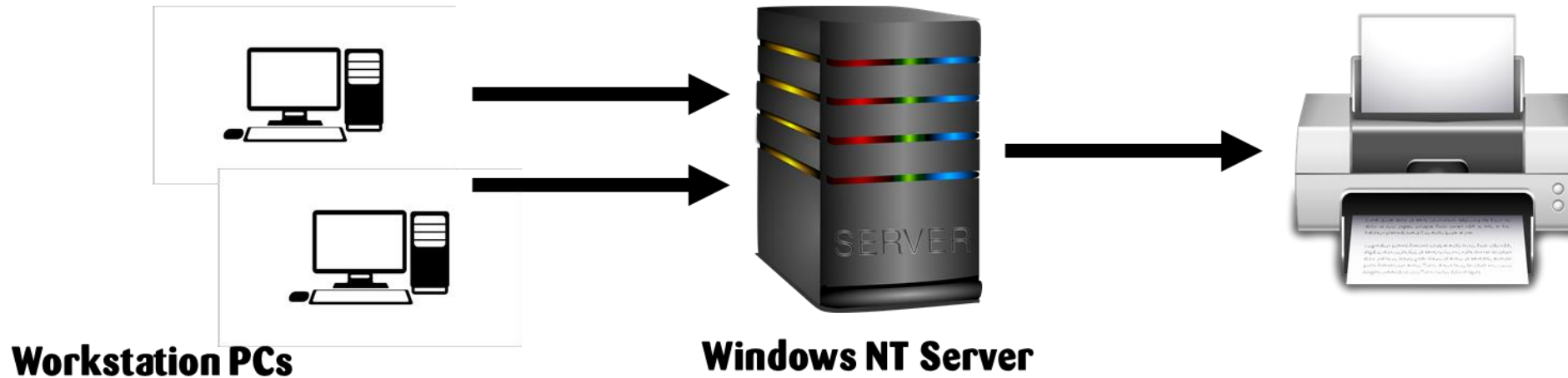


Windows NT file system cache

- **An important use case for the initial versions of Windows NT was to function as a multiple-user File and Print Server**
 - **Cached disk controller cards were not yet available for PCs, so disk access was relatively slow**
 - **With semi-conductor DRAM prices decreasing steadily (Moore's Law), speed up access to disk files by using some portion of the machine's physical memory to cache them**



Windows NT file system cache



Problem: how much memory should be devoted to use for the disk cache?

- **Note: caching disk files in memory speeds up access to files only when there is **Read** access to shared files by **multiple** users concurrently!**

**How would you determine if the
Windows NT File cache was a
breakthrough in intelligent, adaptive
behavior?**

Evaluating the Windows NT File cache

- Feed it a “worst-case” workload designed to defeat an LRU-based caching scheme
- See if the Windows NT File cache can recognize this workload and adapt to it by limiting the memory used by the File cache
 - Does it do sequential limiting?
- Feed it a file cache workload analogous to the **PageALot** program, a performance **anti-pattern** that is designed to defeat LRU-based virtual memory management schemes

Evaluating the Windows NT File cache

- **I fed it a file to Read sequentially that was considerably larger than the *sizeof*(RAM) so it would overflow the memory-resident cache and watched what happened using Perfmon counters**
 - **Memory counters**
 - **Cache counters**
- **On Windows NT Server, the OS expanded the size of the file cache until it consumed almost all of physical memory**
 - **The File cache used pre-fetching during sequential Read operations, so file cache hit % measurements indicated quite good performance**
 - **But the machine itself was unusable, if you tried to run any other workload concurrently, due to a physical memory shortage & related excess paging**

Was Windows NT self-tuning?

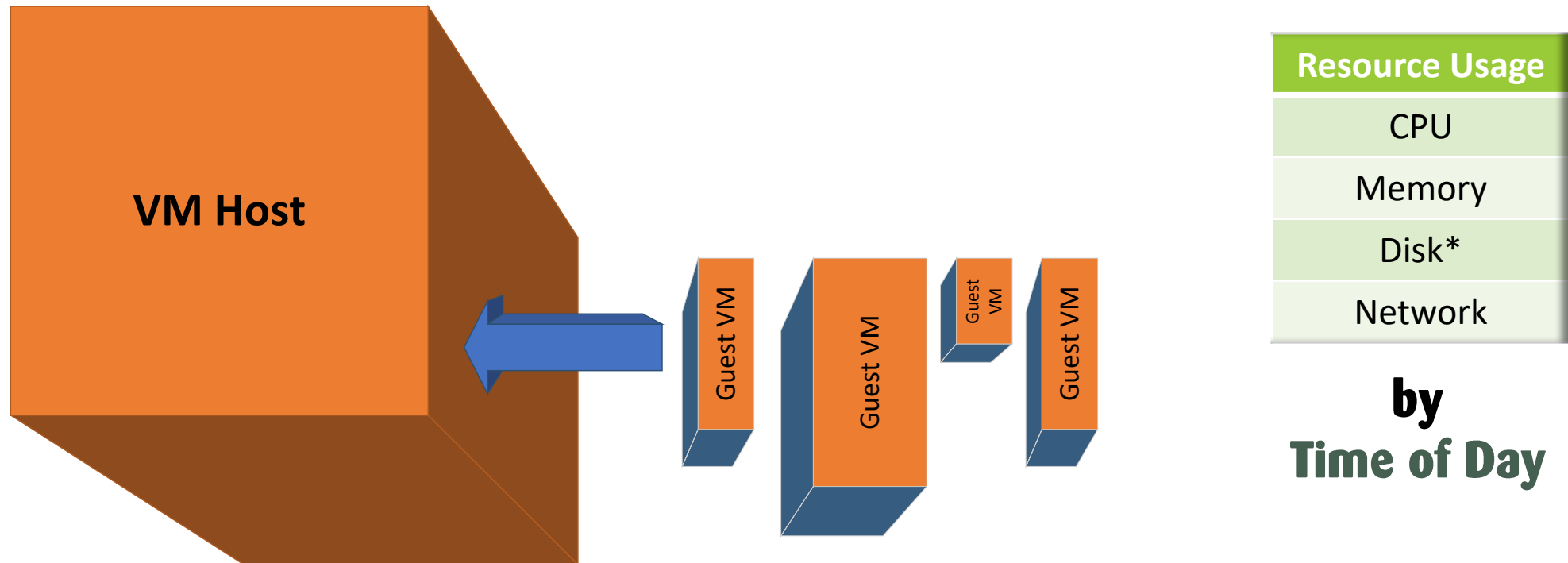
- **Claims in the 1st “Inside Windows NT” book about the OS being “self-tuning” were greatly exaggerated!**
 - **The OS had a large number performance parameters that were hidden from the customer, but available in the Registry under various keys**
 - **see, e.g.,** `HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management`
 - **Some of these parameters were set at boot time, automatically based on memory size, processor speed**
 - **Some of these parameters were set differently, depending on whether you were running Server edition or Workstation**
 - **Eventually, the Registry settings were documented**
 - **Subsequently, David Solomon published a much better version of “Inside Windows NT” that detailed how the OS worked!**

What are the prospects for autonomic computing?

- **Manual tuning does not scale to the quantity of computer resources that need to be managed today**
- **Traditional capacity planning has given way to “provisioning”**
 - **Virtualization, cloud services, containerization, micro-services, serverless computing**
- **Can this provisioning be automated?**
 - **QoS**
 - **Machine Learning**
 - **Anomaly detection & alerting**
 - **Bottleneck detection needs to be informed by analytic models**
 - **Feedback and control engineering**
 - **Requires:**
 - **clear, unambiguous measurements**
 - **resources that can be provisioned dynamically**

The Challenge of Virtualization

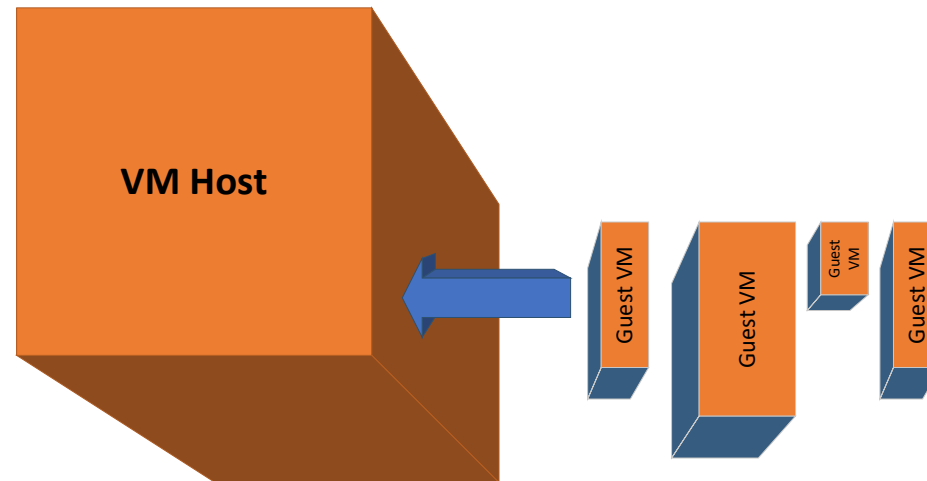
- Initial VM Host machine sizing *appears* relatively easy
 - Stack 5-dimensional shapes efficiently into a 5-dimensional container



- being careful not to ever exceed the *capacity* of the container in any dimension

The Challenge of Virtualization

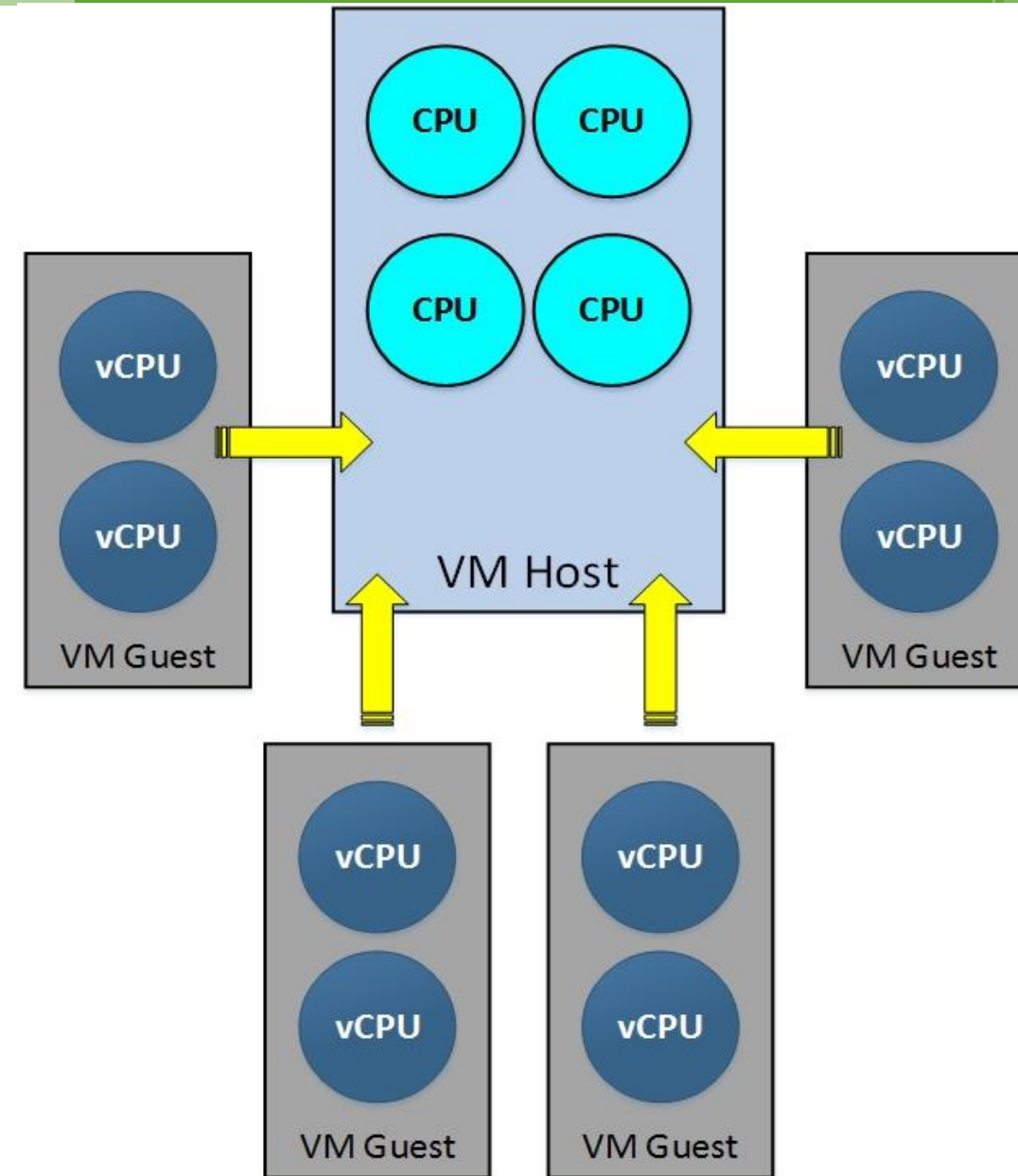
- Note: the capacity of the container is *static*, but usage behavior of the guest machines is *dynamic*



- Post-virtualization, it becomes much more difficult to assess how much physical resources guest machines actually require
 - e.g., Physical memory requirements are especially difficult to assess*

The Challenge of Virtualization

- **Balance** more efficient use of the hardware against the performance risks of **over-commitment**
- Leverage configuration options that are not available in physical hardware:
 - **Is three CPUs enough?**
 - **RAM partition sizes that are not available in hardware configurations**
- Virtualized SAN disks that cut across the physical machines, representing an independent virtualization layer



Virtualization capacity planning challenges

- ***Guest machine physical memory requirements are especially difficult to assess**
 - **Memory management is very dynamic**
 - **virtual memory management tends to allocate all the RAM that is available**
 - **reclaims “older” memory areas on demand when there is contention**
 - **applications like SQL Server that rely on memory-resident caching immediately allocate all the RAM that is available on the guest machine**
 - **well-behaved Windows server apps respond to Lo/Hi memory notifications issued by the OS**
 - **SQL Server**
 - **.NET Framework applications (including ASP.NET Web Server apps)**
- **Justifies *over-committing* physical memory on the VM Host**

The Long View:

- Managing a large, virtualized computing infrastructure mainly involves *load-balancing* of the hardware and rapid *provisioning* of new guest machines that execute in an application *cluster* when they begin to encounter constraints.
- This mode of operation is *reactive*, rather than *proactive*, which flies in the face of 40 years of effective data center capacity planning.
 - Note: the mega-datacenters that are devoted to servicing a small number of huge, monolithic application suites do not face this problem
 - e.g., Google, Facebook, AWS, Microsoft Azure
 - But the traditional corporate IT datacenters, trying to support a *heterogeneous* mix of applications, do!

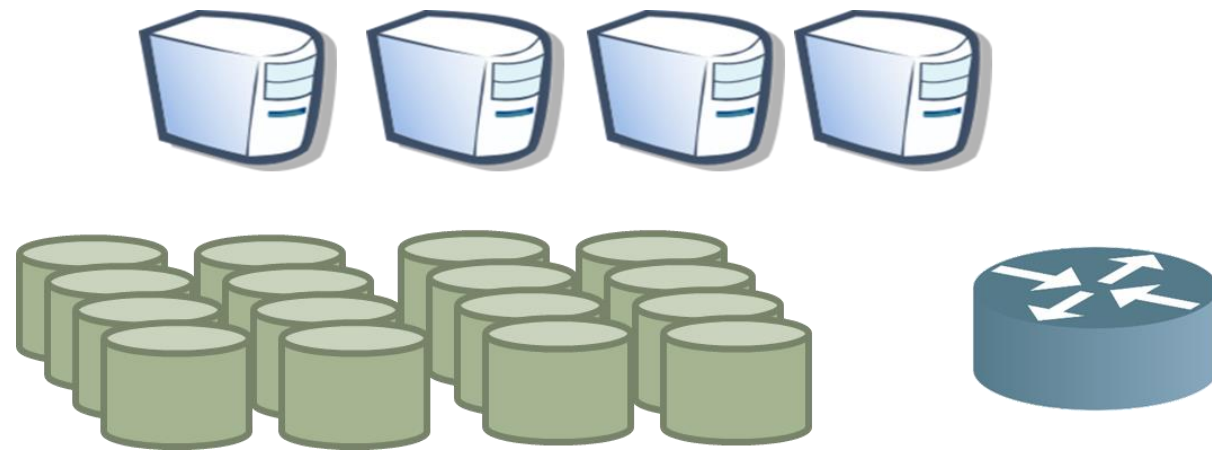
The Challenge of Virtualization

- **Virtualized infrastructure in the corporate IT datacenter introduces resource sharing, amid complex, heterogeneous configurations**



- **Unfortunately, no single view of the infrastructure is adequate or complete**

- **Shared storage layer**
- **Shared networking infrastructure**
- **VM Host clusters**
- **Guest machines (often clustered)**
- **N-tier layered applications**



The Performance Monitoring challenge:

- **No single view of the infrastructure topology is adequate or complete:**

NAS/SAN	Networking	VM Hosts	Guest VMs	App monitoring (n-tiered)	RUM
<ul style="list-style-type: none">• Physical Disk and controller utilization• storage hierarchy• Cache	<ul style="list-style-type: none">• Routers• Load balancers• Cache	<ul style="list-style-type: none">• CPUs• RAM• VM scheduling• SLAT	<ul style="list-style-type: none">• Processes• Virtual memory (includes GC)• Virtual Device service times	<ul style="list-style-type: none">• Service levels• Delays• Component Response Times• HA Clustering	<ul style="list-style-type: none">• Includes the network Round Trip time (RTT)

- **Consequences:**
 - **Absence of accurate measurement data limits the effectiveness of automatic feedback and control mechanisms**
 - **Hypervisor provides familiar Load Balancing, priority scheduling and QoS reservations tuning options**

Virtualization capacity planning challenge:

- Virtualized infrastructure presents significant challenges to *traditional* data center capacity planning practices
 - Virtualization has only a minor impact on guest machine performance so long as the resources of a massively over-provisioned VM Host machine are not *over-committed*
 - But, when *over-commitment* occurs, the performance impact can be severe
 - as a consequence of the *black box* approach
 - Plus, untangling the root cause of the performance problems is difficult
 - due to the complexity of the environment and the limited vision of the tools

Virtualization capacity planning challenge:

- Virtualized infrastructure presents significant challenges to **traditional** data center capacity planning practices
 - The potential for resource contention is minimized when the VM Host machine's resources are **underutilized**, but that sacrifices **efficiency**
 - Goal: run hardware systems that are **balanced** and guest machines that are **right-sized**
 - Note that dynamic load balancing (*e.g.*, VMWare **vMotion**) is potentially disruptive

Virtualization capacity planning challenge:

- Virtualized infrastructure presents significant challenges to *traditional* data center capacity planning practices
 - Partitioning introduces the potential for resource contention, especially in the case when the VM Host is *over-subscribed*
 - Round-robin (fair) + priority-based scheduling mechanisms at the Processor
 - Dynamic memory management, including inflating a guest machine physical memory balloon
 - By design, *paravirtualization* treats the guest machine as a *black box*
 - With the exception of a few, targeted Windows Hyper-V “*enlightenments*,” there is no ability to feed-forward guest machine *service level* measurements into the physical resource scheduling algorithms

When does Guerilla Capacity Planning work?

- Massive computing resources devoted to large-scale, monolithic web properties tends to create predictably **stable** configurations
 - Relatively easy to load balance using simple, round-robin Request scheduling
 - Once they reach a critical mass, forecasting **incremental** application growth is also straight-forward
 - Predictive analytic modeling techniques can also be applied
- Option to divert applications with very variable resource requirements to on-demand, pay-for-play, public Cloud Computing resources

Virtualization capacity planning challenge:

- Virtualized infrastructure presents significant challenges to **traditional** data center capacity planning practices
 - Many current industry Best Practices are based on experience with very large scale, **monolithic** web sites & services
 - However, in most corporate data centers, the IT department must manage a **diverse** portfolio of application workloads
 - Result: the VMs residing on a given VM Host represent a **complex, heterogeneous, and combustible** mixture
 - With many different server applications running on each VM Host and sharing its physical resources

Virtualization capacity planning challenge summary

- Virtualized infrastructure presents significant challenges to *traditional* data center capacity planning practices
 - Guest machine performance suffers when
 - the guest machine is *under-provisioned*
-- or --
 - the VM Host machine is *over-committed*
 - Plus, configuring more resources than the guest requires can impact other resident guest machines
 - Virtualization of clock interrupts interferes with the quality of guest machine performance from internal measurements

Provisioning VM Host machines

Condition	Who suffers a performance penalty
Over-committed VM Host	<i>All resident guest machines suffer</i>
Efficiently provisioned VM Host	No resident guest machines suffer
Over-provisioned VM Host	No guest machines suffer, but hardware cost is higher than necessary
Under-provisioned Guest	<i>Guest machine suffers</i>

- and “**right-sizing**” the guest machines

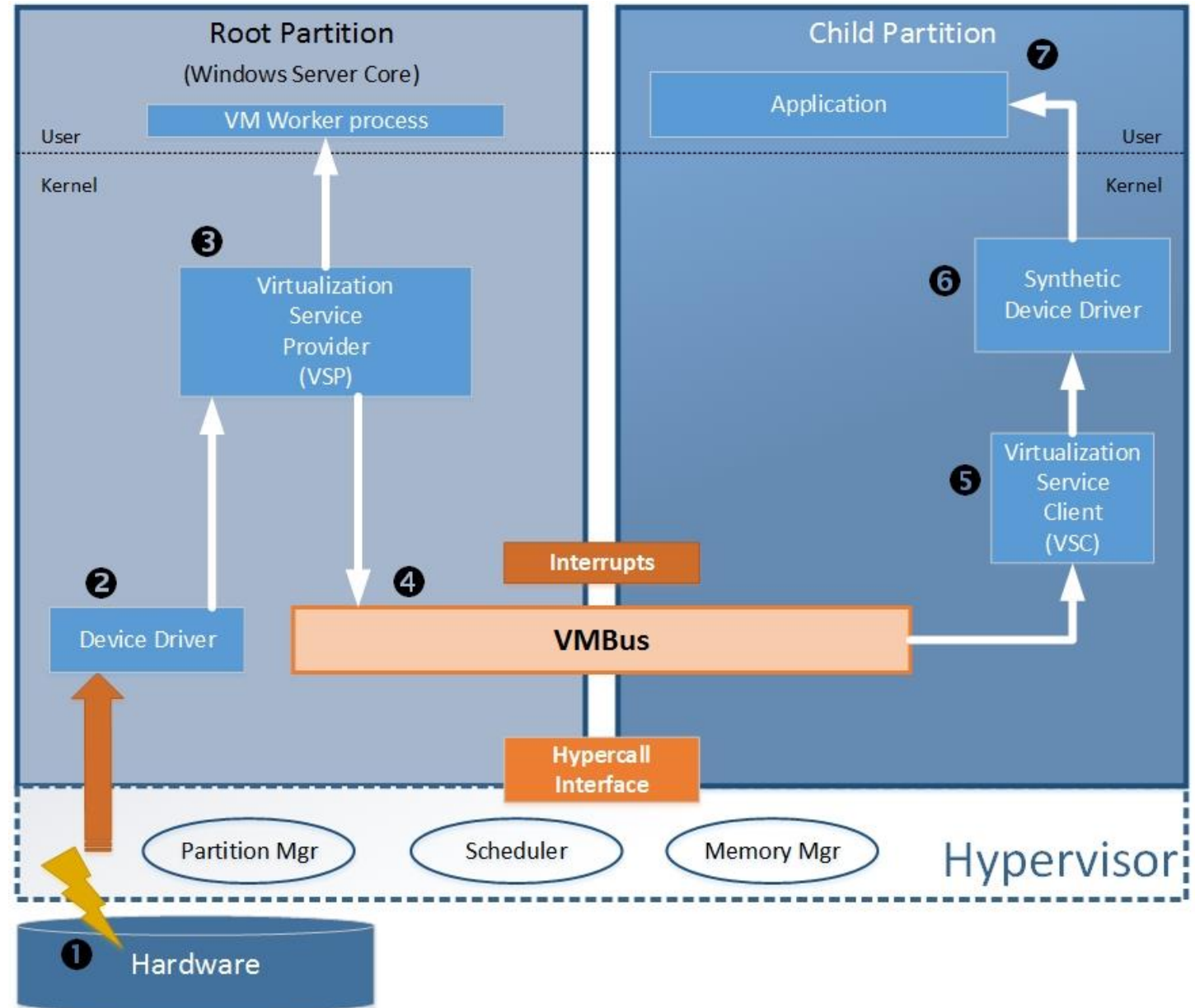
Virtualization Configuration strategies

Partitioned	<ul style="list-style-type: none">• Very Large scale hardware• a few large scale guest machines (e.g., large database servers)• Guest machine right-sized to underlying physical hardware• e.g., 15 vCPUs outperforms 16 vCPUs on a physical machine with 15 physical CPUs/core
Over-provisioned	<ul style="list-style-type: none">• $\text{vCPUs} \leq \text{Physical CPUs}$• $\sum \text{virtual RAM} \leq \text{Machine memory}$
Efficiently provisioned	<ul style="list-style-type: none">• large number of smaller guests• heterogeneous workloads• variable demand• $\text{vCPUs} > \text{Physical CPUs}$• $\sum \text{virtual RAM} > \text{Machine memory}$
Over-committed (over-subscribed)	<ul style="list-style-type: none">• large number of smaller guests• heterogeneous workloads• variable demand• $\text{vCPUs} \gg \text{Physical CPUs}$• $\sum \text{virtual RAM} \gg \text{Machine memory}$

Hypervisor Architecture

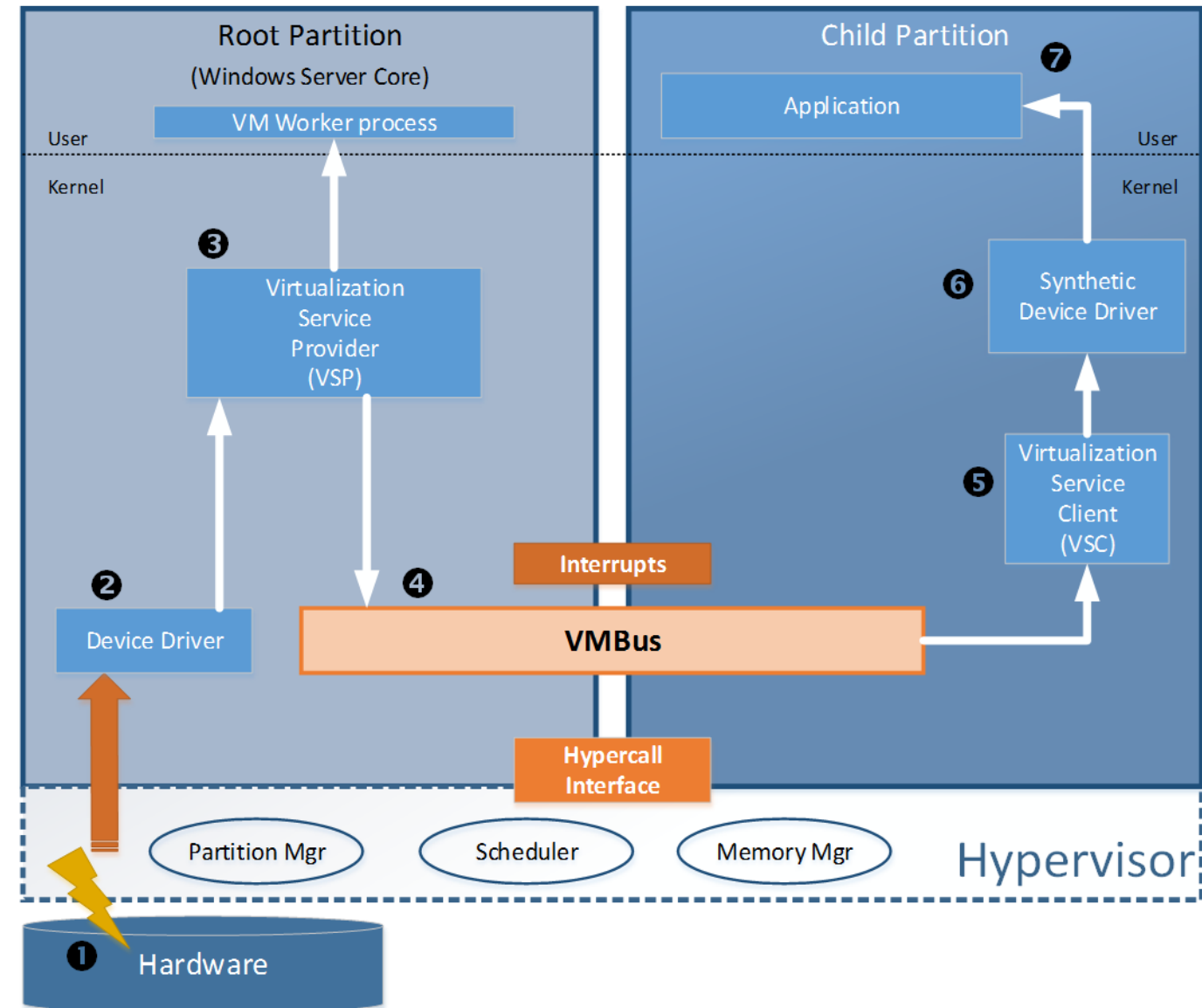
- **Interrupt processing**

1. hardware interrupt
2. native device driver
3. virtual device routing & translation
4. transfer to guest machine
5. virtual hardware interrupt processing
6. synthetic device driver
7. application scheduling



Hypervisor Architecture

- **Performance impacts**
 - **increased code path**
 - mitigated somewhat by “enlightened” device driver software
 - **Pending interrupt time accumulates if an available guest machine Logical Processor cannot be dispatched immediately**
- **Hardware clock (*rdtsc*) instructions and timers are also subject to virtualization (with similar delays)**



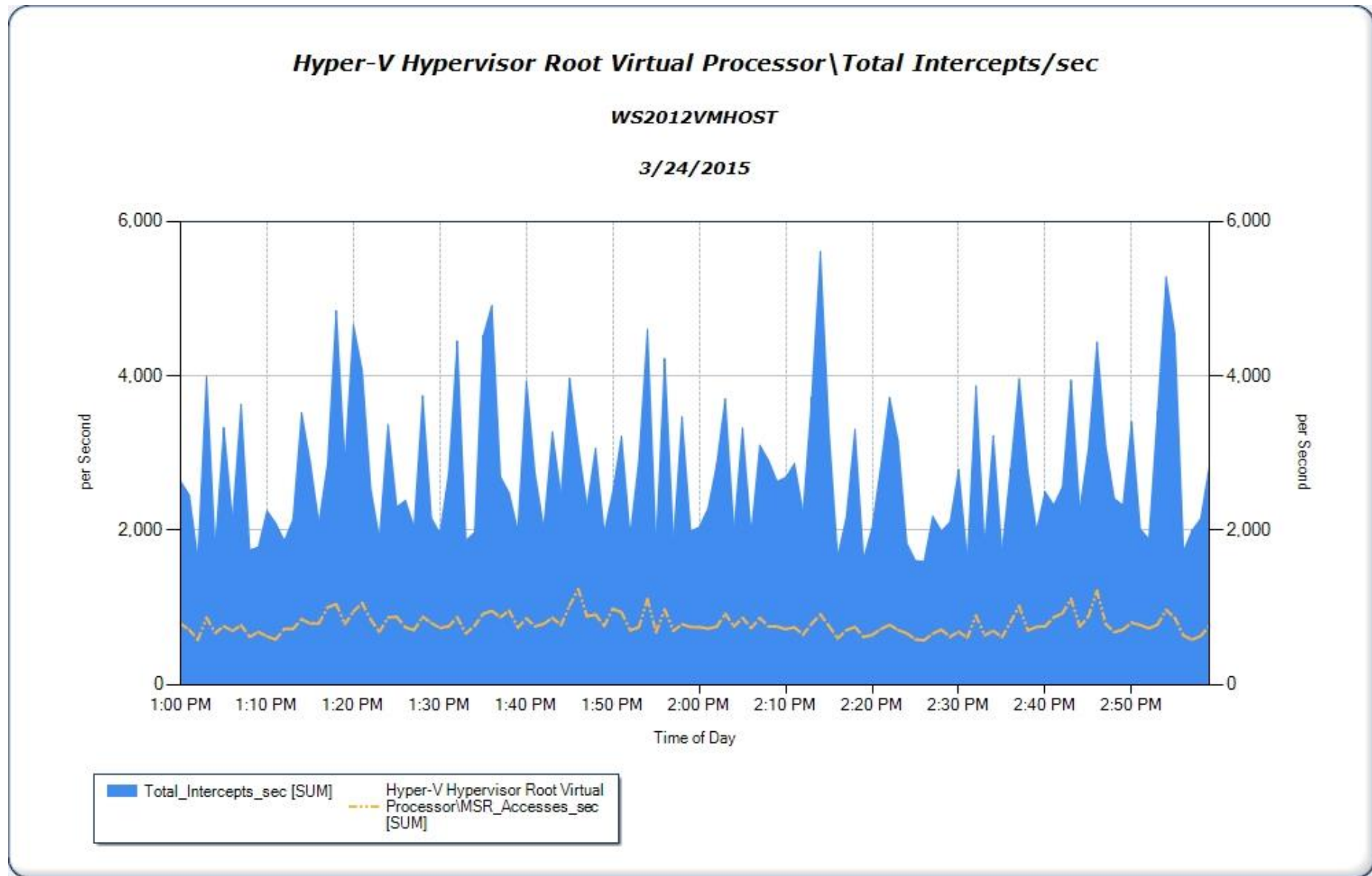
Performance impact of virtualization “overheads”

- **Minor performance impact so long as the VM Host is not over-committed**
 - **5-15% stretch factor due to:**
 - **Instruction emulation**
 - **Guest VM Scheduler overheads**
 - **Virtual interrupt processing**
- **However, expect a major performance impact when the VM Host machine is **over-committed****
 - **e.g., Guest Machine Memory ballooning**

Performance impact of virtualization “overheads”

- **Instruction emulation**
 - **Whenever the guest machine (usually the guest OS) executes restricted instructions that must be trapped by the VM Host layer and then emulated**
 - **CPUID**
 - **OS accessing MSRs**
 - **accessing IO ports**
 - **invalid operations (page faults, attempts to divide by zero)**
 - **rdtsc**

e.g., Hyper-V intercepts



Benchmarking results (Hyper-V)

Configuration	# of machines	CPUs per machine	elapsed time (minutes)	stretch factor	Thruput	Hyper-V % Run Time
Native machine	1	4	90	...	1	...
Root Partition	1	4	100	1.11	1	6%
Guest machine	1	4	105	1.17	1	8%
Under-provisioned Guest machine	1	2	147	1.63	1	4%
2 Guest machines	2	2	178	1.98	2	6%
4 Guest machines	4	2	370	4.08	4	6%



Benchmarking results (Hyper-V)

- **Timing test executes 10-17% longer, compared to Native baseline**
- **Under-provisioned guest machine pays a significant penalty**
 - **stretch factor = 1.6**
- **Scalability improvements can mediate the performance impact**
 - **stretch factor = 2.0; throughput = 2x**
- **Over-committed VM Hosts can cause significant degradation**
 - **Setting guest machine Priority or making a QoS capacity reservation will protect a cherished workload**

Benchmarking results (Hyper-V)

- **Over-committed VM Hosts can cause significant degradation**
 - **Setting guest machine Priority or making a QoS capacity reservation will protect a cherished workload**

Configuration	# guest machines	CPUs per machine	Best case elapsed time	stretch factor
Native machine	...	4	90	...
4 Guest machines (no priority)	4	2	370	4.08
4 Guest machines with Relative Weights	4	2	230	2.56
4 Guest machines with Reservations	4	2	270	3.00

Recognizing under-provisioned guest machines

- **Hypervisor does not have direct access to internal performance counters**
 - With one notable exception of an “enlightenment” used by the Hyper-V Memory Manager
- **Manual tuning knobs are provided**
 - **Not enough CPUs defined to the guest**
 - VMware ESX (relaxed) chained processor scheduling discourages over-provisioning the guest VM
 - Evaluate the System\Processor Queue Length counter
 - **Not enough RAM provisioned for the guest**
 - Chronic shortage of Memory\Available Bytes
 - High rates of hard paging to disk (Memory\Pages input/sec)



Recognizing over-committed VM Host machines

- **Over-commitment** has the potential to impact **every** resident guest machine
 - Without some degree of over-commitment, however, the Host machine hardware will be under-utilized!
 - $\sum_1^n \# \text{Virtual Processors}_{guesti} > \text{Host Machine \#CPUs}$
 - Guest machine CPU Ready (*milliseconds*)
 - $\sum_1^n \text{sizeof}(RAM)_{guesti} > \text{Host Machine sizeof}(RAM)$
 - Guest machine **Balloon Memory**
 - **Over-subscribed** is more apt term than **Over-committed**
 - **Note:** Shared disk and networking hardware can also be over-subscribed

Over-committed VM Host machines

- Over-commitment has the potential to impact **every** resident guest machine
- Automatic load balancing using active migration of guest VMs
 - e.g., vMotion
 - But, without an understanding of the guest machine **application state**, a feature like vMotion is potentially disruptive, and
 - Hypervisor does not have direct access to internal performance counters to assist in its decision-making
- So, **manual** tuning knobs are provided
 - Load balancing at the VM Host level
 - Scheduling Priority settings
 - QoS Reservations and Limits

Over-committed VM Host machines

- Hypervisor does not have direct access to guest machine internal performance indicators
 - With one notable exception of a proprietary “enlightenment” used by the Hyper-V Memory Manager
- Manual tuning knobs are provided
 - Scheduling priority settings
 - QoS reservations and limits
 - Crude controls that are difficult to implement (trial & error)
- Given the size and complexity of the configurations SysAdmins must manage, these tuning options are poor alternatives to goal-oriented **control systems** that have access to guest machine **feedback**

Over-committed memory in Hyper-V

- Hyper-V attempts to equalize **Memory Pressure** across all Windows VMs with the same dynamic memory allocation priority
 - an “enlightenment” used by the Hyper-V Memory Manager
- Pressure is a Memory contention index (V/R):

$$\frac{\text{guest machine Committed Bytes} * 100}{\text{current machine memory allocation}}$$

- guest machine **paging** increases as **Memory Pressure** $\gg 100$
- interfaces with the “hardware” hot memory Add/Remove facility
- memory priority creates “bands” of machines based on **Memory Pressure**

Addendum

- **Answer to most performance questions: It depends!**
 - **Measure, measure, measure**
 - **(Remember the “lighthouse effect”)**
- **Performance **anti-patterns** and **worst-case** performance**
 - **best case performance**
 - **average performance**
- **Open source benchmarking projects: ([link](#))**

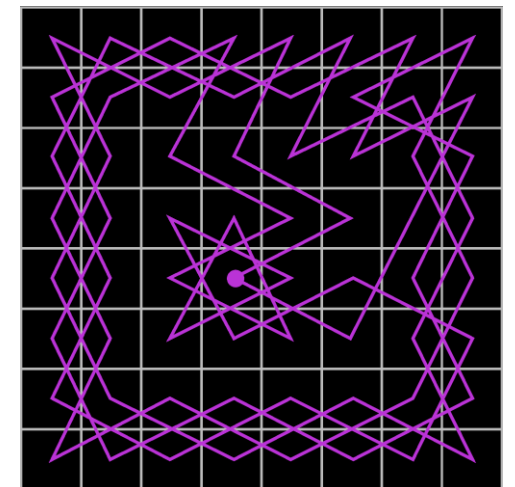
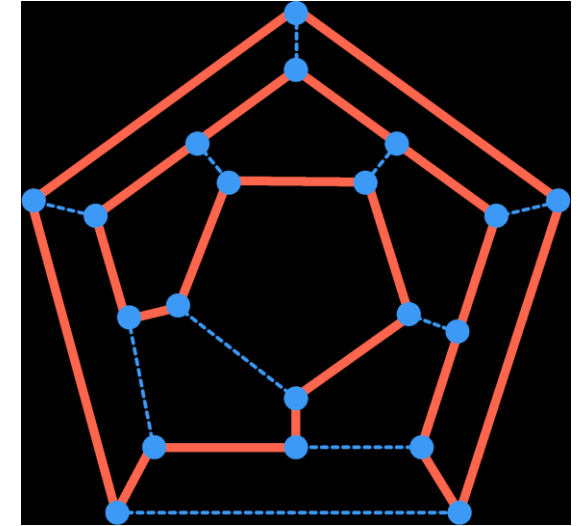
Addendum

- **Algorithms and complexity**
 - **Problems of scale:**
 - what happens when n grows large?
 - **Data structures: Arrays, Lists, trees, etc.**
 - **Search**
 - linear: $O(n)$
 - binary: $O(\log n)$
 - **Sort**
 - Bubble: $O(n^2)$
 - QuickSort, Heap: $O(n \log n)$
 - **SortedBinaryTree.Insert()**
- **What happens to scalability when $O \Rightarrow n^n$?**

NP-complete

- e.g. Hamiltonian Path

- Is there a path through all the nodes that touches each node only once?
- **$n!$** possible paths
 - Calculate the shortest path?
 - that returns to the origin (Hamiltonian cycle)
- practical examples:
 - Chess Knight's traversal
 - shortest path (traveling salesman)
 - optimize placement of components on a computer chip
 - static analysis of all possible code paths through a library routine
 - optimize computer resources efficiently over a set of geographically-distributed network nodes



Insight from Number theory

- **Project to establish Mathematics as a formal system (based on a set of axioms, like Geometry), using first-order Logic (predicate calculus) and Set theory (Cantor, Zermolo-Fraenkle)**
 - **e.g., What is a number?**
 - **Russell & Whitehead: using Set theory and logical types**
 - e.g., The number 3 is the set of all things that have that number of items in them.
- **Demolished forever by Gödel's Incompleteness Theorem, namely**
 - **for any computable axiomatic system powerful enough to describe the arithmetic of the natural numbers (e.g., Peano)**
 - **If a (logical or axiomatic formal) system is **consistent**, it cannot be **complete**.**
 - **The consistency of axioms cannot be proved within their own system.**

Insight from Number theory

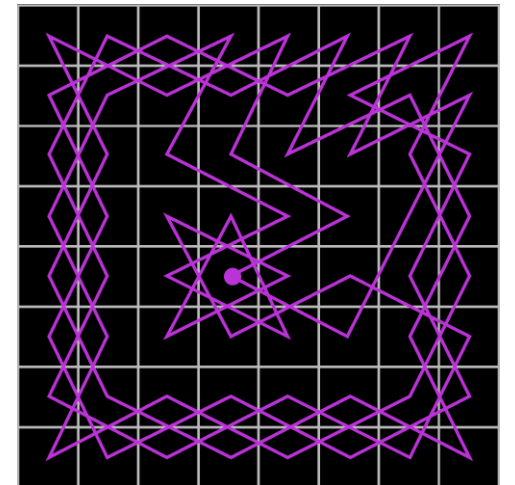
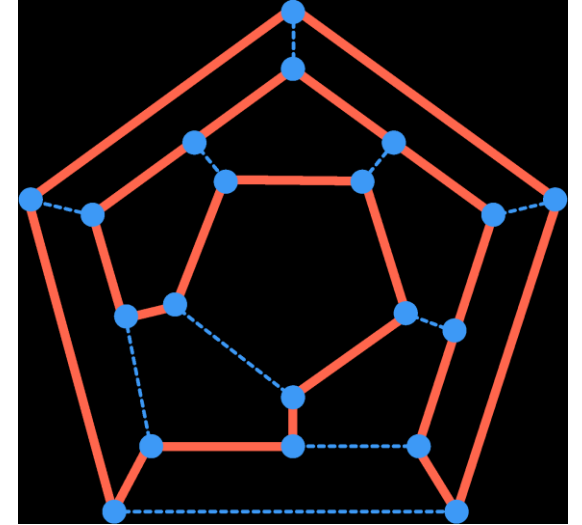
- **Turing's Halting problem:**
 - Is there a deterministic procedure for deciding whether a given program will execute to completion, computing the correct answer?
 - No, it is **undecidable** for Turing machines (note: Church's thesis)
- **Informal proof:**
 - suppose a computer program is executing an algorithm that is NP-complete
 - There is no provably correct method that can accurately decide whether
 - the program is defective, i.e., it is in an infinite loop, and should be canceled, or
 - the program is making program progress towards a solution

Computationally intractable problems

- The argument that computers are **finite-state machines** does not help:
 - A machine with finite memory has a finite number of states,
 - so, any deterministic program on it must eventually either halt or repeat a previous state
 - but how many states are there?
 - a machine with 1 Million bits $\Rightarrow 2^{1,000,000}$ possible states
- Undecidable becomes intractable
- Conclusion: beware of NP-complete algorithms
 - e.g., optimizing computer resources efficiently over a set of geographically distributed network nodes, based on the workload

NP-complete

- **practical examples:**
 - **shortest path (traveling salesman)**
 - **optimize placement of components on a computer chip**
 - **static analysis of all possible code paths through a library routine**
 - **optimize computer resources efficiently over a set of geographically-distributed network nodes**



Questions



References

- Woodside, et.al., “The Future of Software Performance Engineering,” *Proceedings Future of Software Engineering*, IEEE, 2007.
- Friedman, ["Virtual memory management in VMware"](#)
- Friedman, ["Hyper-V performance"](#)