Warehouse-Scale Computers

datacenter pictures

Google Council Bluffs, Iowa datacenter

datacenter pictures: servers racks

Facebook datacenter, Prineville, Oregon; via OregonLive

datacenter pictures: servers

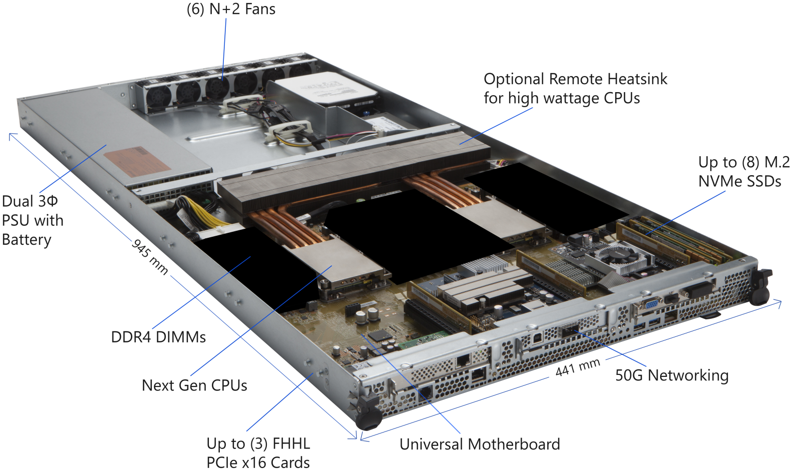
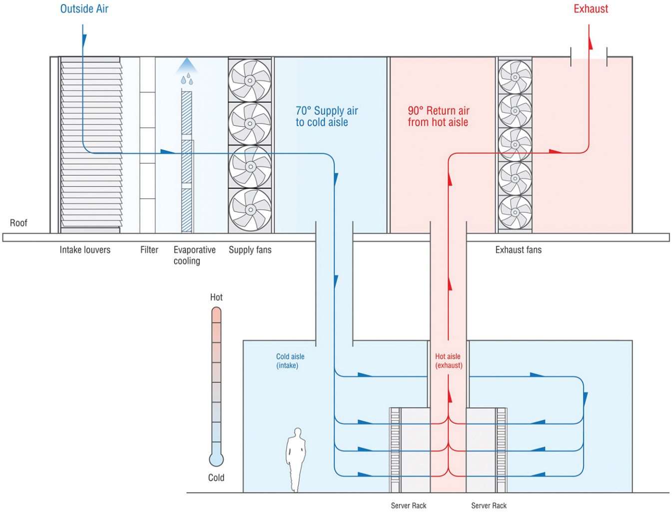


image: Open Compute Project (proposed 2016)

datacenter pictures: cooling



Sasser, "A Look at Data Center Cooling Technologies"

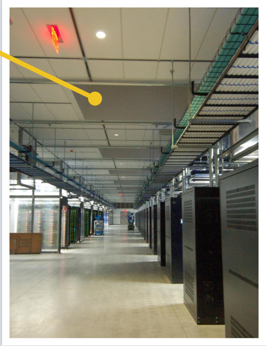
Mechanical Penthouse

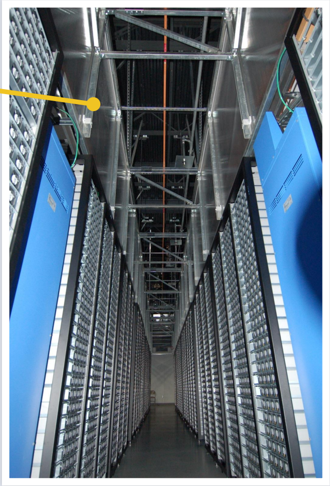
Air mixing section - Return air / out side air / filter corridor

Evaporative cooling / humidification corridor

Data Suite

- Hot aisle containment – ductless return
- Cold aisle pressurization – ductless supply





Facebook/Open Compute Project slide

datacenter pictures: backup power

Facebook datacenter, Prineville, Oregon; via TechCrunch

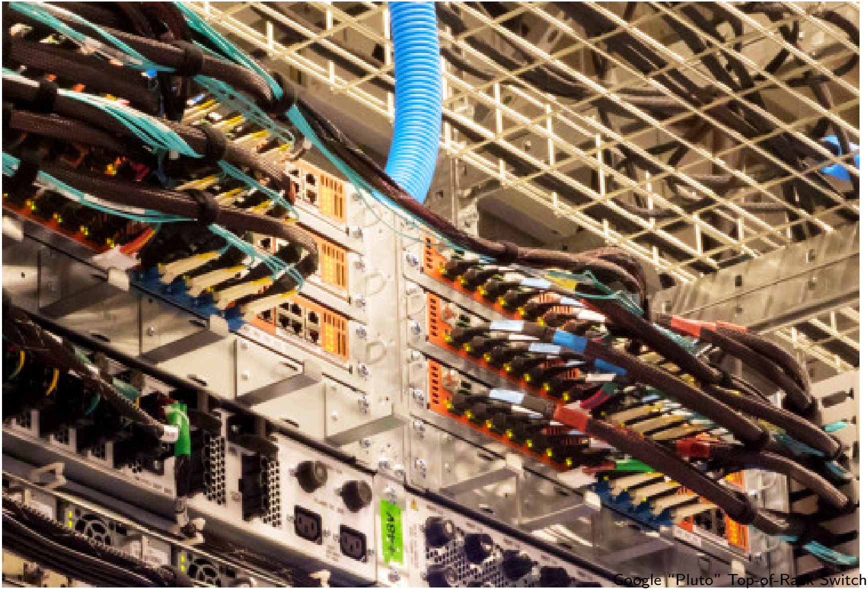
datacenter pictures: battery room

image: NOAA Center for Weather and Climate Prediction (at University of Maryland)

datacenter pictures: battery cabinet

Image: Facebook

datacenter pictures: TOR switch



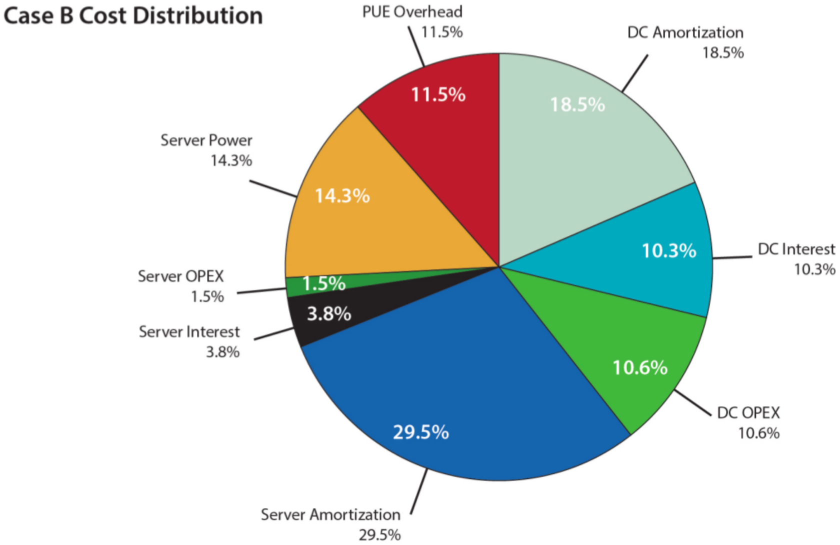
Google "Pluto" Top-of-Rack Switch

datacenter sizes

tens to hundreds of megawatts

tens of thousands to hundreds of thousands of servers

Money, Money, Money



Kinds of cost

Operational:

power — e.g. cheap hydroelectric
maintainence — replacement equipment, etc.
people — sysadmins

Capital

buying/renting building + cooling + backup power
buying servers and replacing them when they become outdated

Common metric — cost per Watt

Datacenter Applications

"the web"/interactive:

latency matters

reliability matters

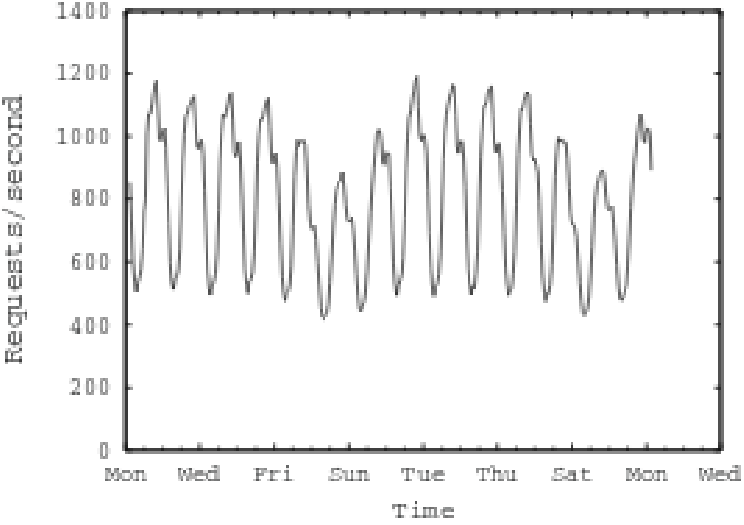
"free" parallelism — independent (mostly) requests

"batch":

throughput matters

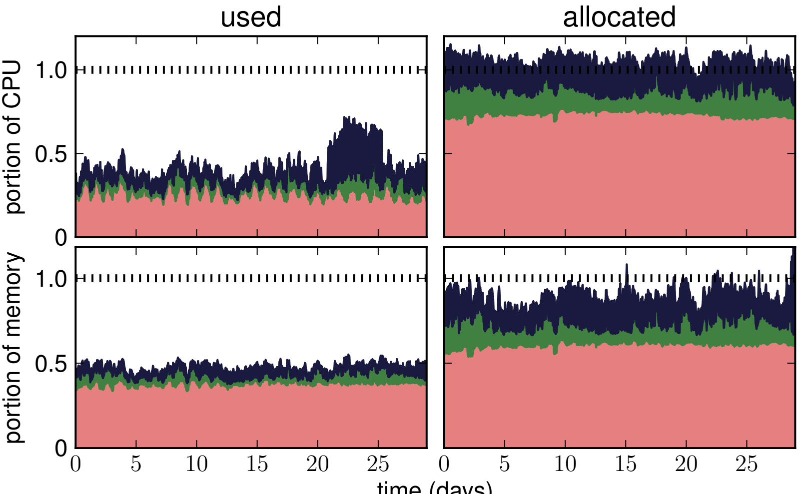
use 'spare' capacity from interactive stuff

Varying demand



Urdaneta et al, "Wikipedia workload analysis for decentralized hosting"

Datacenter applications: consolidation/unpredictability



Datacenter versus Supercomputer

both purpose-built

different kinds of applications

datacenters tend to be more continuously upgraded

DC v SC: Goals

datacenter: focus on cost-performance

scale-out: more servers, not bigger machine

bigger individual machines are less efficient per dollar

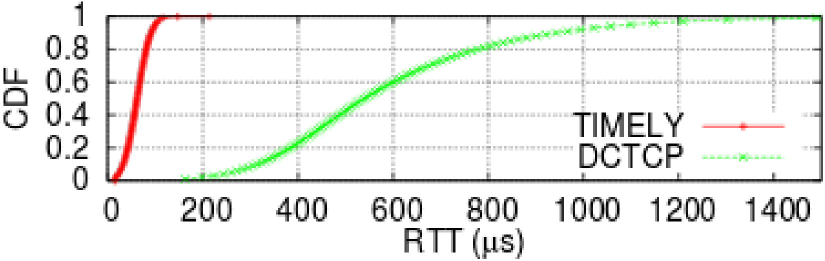
want to use most mass-produced hardware

consolidation — run multiple applications together

more software modifications to use worse servers

DC v SC: Network

highly optimized datacenter network latency:



supercomputer network latency:
often less than ten microseconds round-trip

Datacenter Topology: historical

traditional datacenter topology:

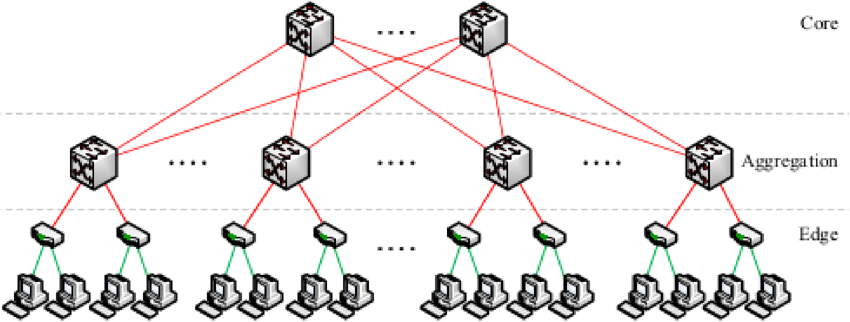


image: Al Fares et al, "A Scalable, Commodity Data Center Network Architecture"

Datacenter Topology: four-post

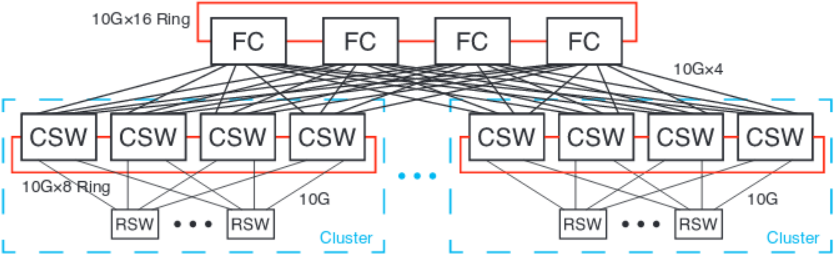


image: Farrington and Andreyev, “Facebook’s data center network architecture”

Datacenter topology: Clos (1)

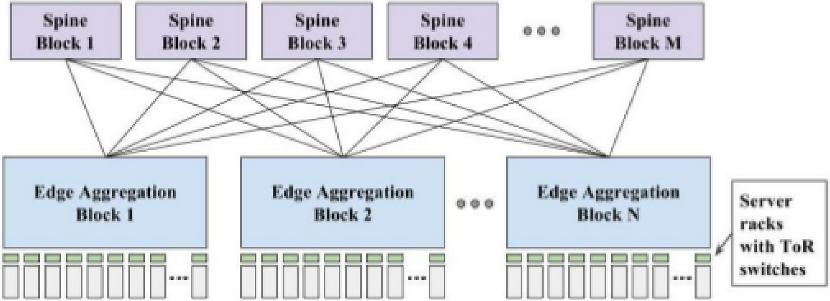


image: Singh et al, "Jupiter Rising: A Decade of Clos Topologies and Centralized Contorl in Google's Datacenter Network

Datacenter topology: Clos (2)

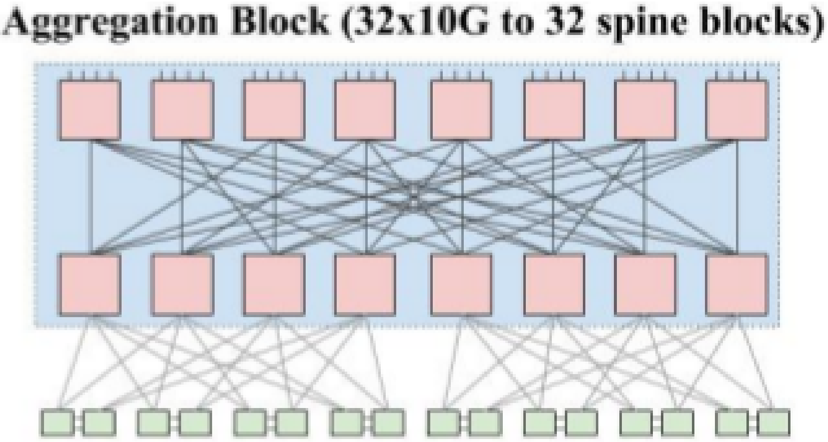


image: Singh et al, "Jupiter Rising: A Decade of Clos Topologies and Centralized Contorl in Google's Datacenter Network

Datacenter Topology: Clos (3)

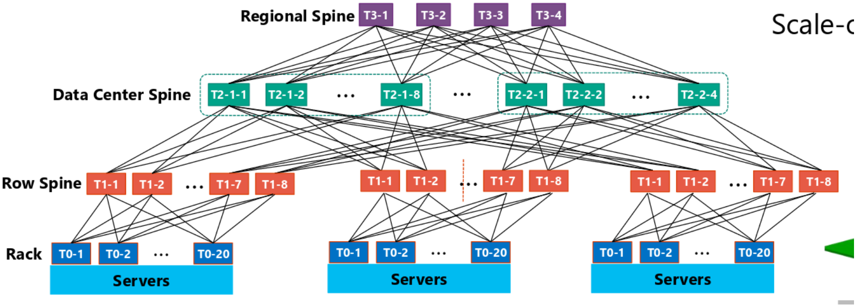


image: Greenberg, "SDN for the Cloud" (Microsoft)

DC v SC: Servers

very similar!

mass-produced, usually superscalar processors

usually high-power CPUs

… but not the most expensive

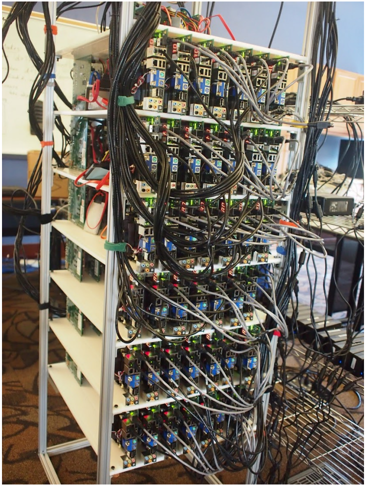
Server Balance

want to maximally use all server resources

balance CPU, memory, storage (disk or SSD)

depends on what applications you run

“wimpy” servers



another proposal: cheap, low-power servers at much higher density

DC v SC: Storage

storage on normal servers

- less networking required
- computations use local (fast) storage

seperate storage racks

- flat storage hierarchy, more convenient to program

DC v SC: Reliability

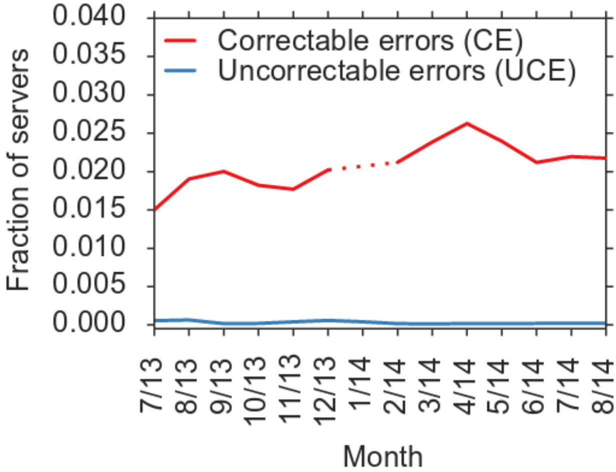
supercomputer: usually more reliable/expensive components

supercomputer: failures — reboot it all

datacenter: expect failures

datacenter: failures — work around broken component

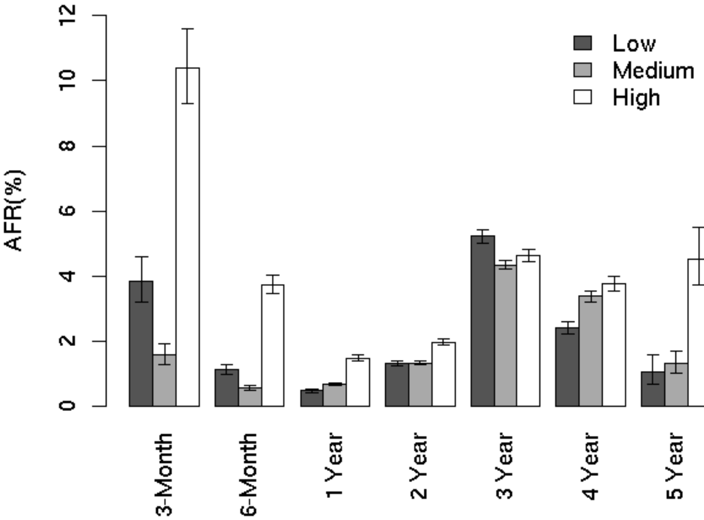
DRAM errors



uncorrectable: approx $.03\%$ of servers per month

Meza et al, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of Trends in the field"

Hard Drive failures



Pinheiro et al, “Failure Trends in a Large Disk Drive Population”

trading for software complexity

redundancy — handle failures

means having backup copies of everything

lots of applications per server — scheduling

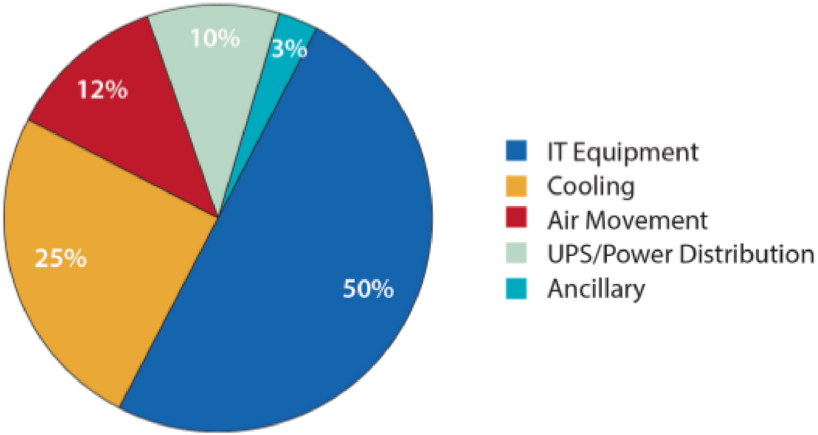
slower network — compute close to data

energy efficiency

also a problem for supercomputers, etc.

but optimized much more heavily in the datacenter era

old datacenter efficieny



PUE

$$\text{PUE} — \frac{\text{total power}}{\text{IT equipment power}}$$

servers and networking equipment

modern large datacenter: < 1.2

before attention to this problem, PUEs of 2 or more were common

Achieving high non-IT efficiency

airflow — don't mix hot/cold air

increased ambient temperature

cooling efficiency

- evaporative cooling
- better climates

power: increased electrical effeicency, e.g.:

- avoid AC/DC conversions
- distributed UPS
- get server power supplies that accept utility voltage

server efficiency

not especially well studied

similar losses from in-server power supplies, etc.

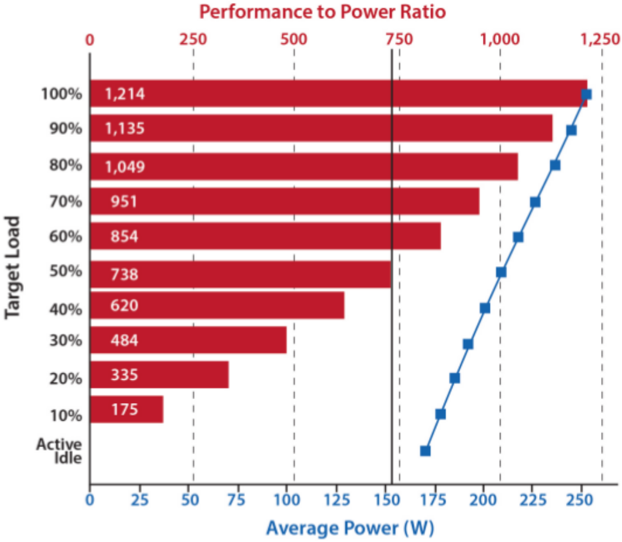
energy efficiency of components varies a lot

power-capping

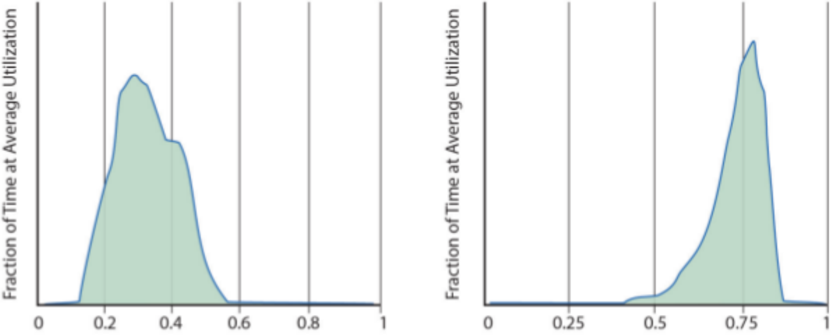
underprovision cooling, power distribution, etc.

limit what runs on servers to stay under actual maximum

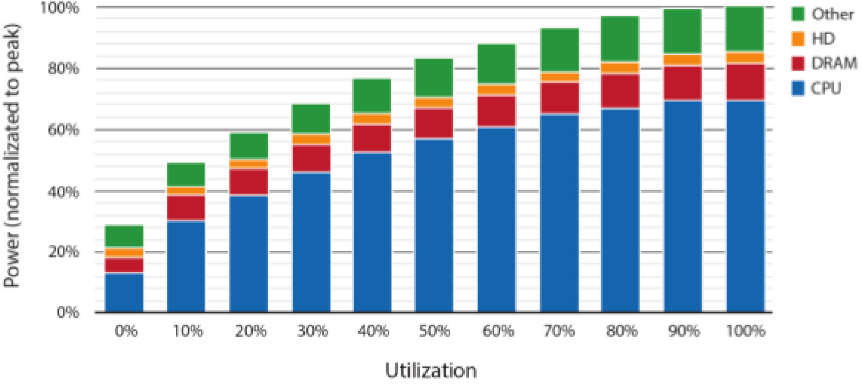
power proportionality problem (1)



power proportionality problem (2)



power proportionality problem (3)



power-saving modes (1)

what about "sleep" modes?

- save a lot of power
- take milliseconds to seconds to start/end

servers need to be available continuously (e.g. stored data)

10% utilized server might be doing some work in every second

not enough time to really save power

power-saving modes (2)

processors have lower frequency/voltage modes

problem: doesn't save power in proprtion to performance lost

problem: things other than processors use power

whack-a-mole in power usage

keep finding things which keep machine from sleeping for long times

keep finding components that use power continuously

tedious engineering problem

the datacenter for rent

public clouds — selling datacenter resources

e.g. Amazon Web Services

one way to deal with lower utilization

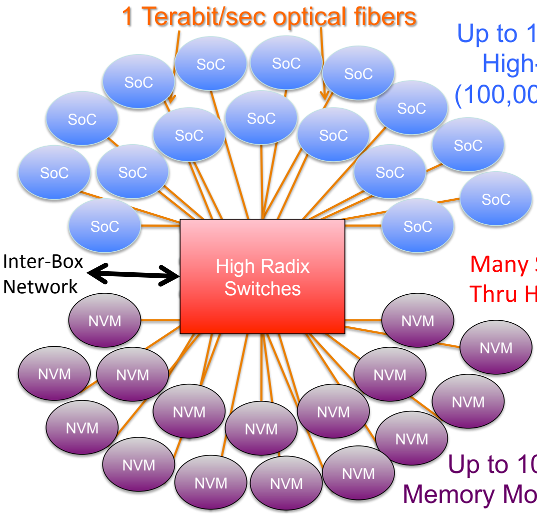
datacenter futures

started with: servers = desktop

trend now: beefier servers

(revisiting old ‘supercomputers’??)

FireBox Overview



Up to 1000 SoCs + High-BW Mem (100,000 core total)

Many Short Paths
Thru High-Radix Switches

Up to 1000 NonVolatile Memory Modules (100PB total)

datacenter futures

PCIe as a networking protocol within a rack?

fast, non-volatile RAM-like memories?

customized chips?

GPUs and FPGAs?

ASICs?

next time

general areas of HW security:

protect programs from each other — page tables, kernel mode, etc.

protect programs from adversaries — bounds checking, etc.

protect programs from people manipulating the hardware

paper: Smith and Weingart, "Building a high-performance, programmable secure coprocessor"

target audience: e.g. banks want to protect PINs

public key cryptography (1)

Smith and Weingart make extensive use of digital signatures

digital signatures use a public/private keypair

example use case: A wants to email B and have B know A wrote the email

public key-cryptography (2)

A generates keypair for communicating with B

public key: given to B; serves as identity/name

assumed known by/safe to tell everyone

private key: kept secret by A

assumed no one else has private key

public key cryptography (3)

two mathematical functions:

$$\textit{signature} = \mathbf{Sign}(\text{A's private key}, \text{message})$$

$$\textit{correct?} = \mathbf{Verify}(\text{A's public key}, \text{message}, \text{signature})$$

$\mathbf{Verify}$ will only say correct if private key was used

computationally infeasible to "forge" signature

A uses $\mathbf{Sign}$ operation, sends message and signature

B uses $\mathbf{Verify}$ operation; rejects if it says "not correct"

certificates

certificates are particular use of digital signature

example: A wants to help B communicate with C

certificate =
$\mathbf{Sign}(\text{A's private key}, \text{“C's public key is XXX”})$

certificate “proves” to B what C's public key is

if B trusts A enough

creating a certificate called “certifying”