Technology Roadmap, Parallel Computing & Datacenters

CS for Lawyers & Policy Workers

Babak Falsafi

ecocloud.ch

Slide credits: Babak Falsafi, Nir Shavit, Maurice Herlihy, Partha Ranganathan, Adam Wierman, Alex Landau of EPFL, MIT, Brown, Google, CalTech and IBM

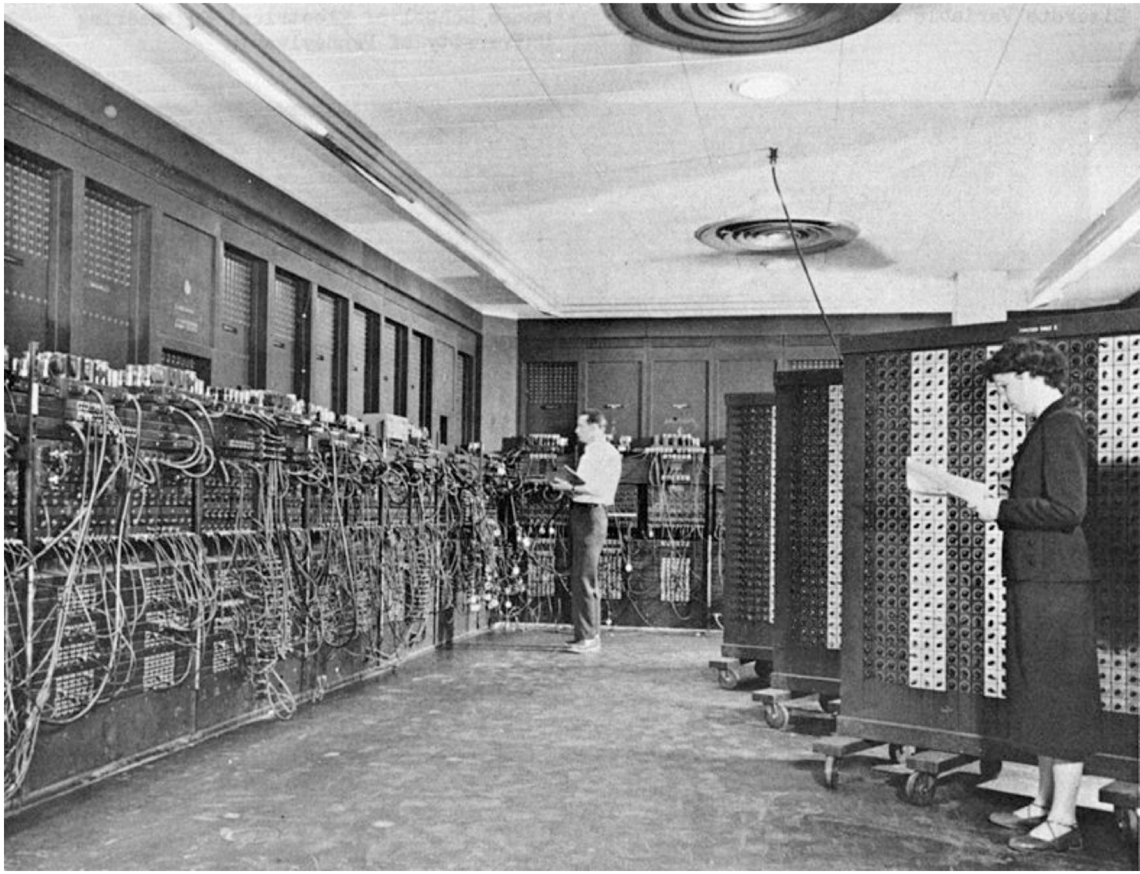
Roadmap

- Technology
  - Moore's Law
  - Parallelism
- Datacenters & Centralization
  - Economies of scale
  - Metrics
- Service-Oriented Computing
  - Cloud
  - Virtualization

Where did it all start? ENIAC

[picture from Wikipedia]

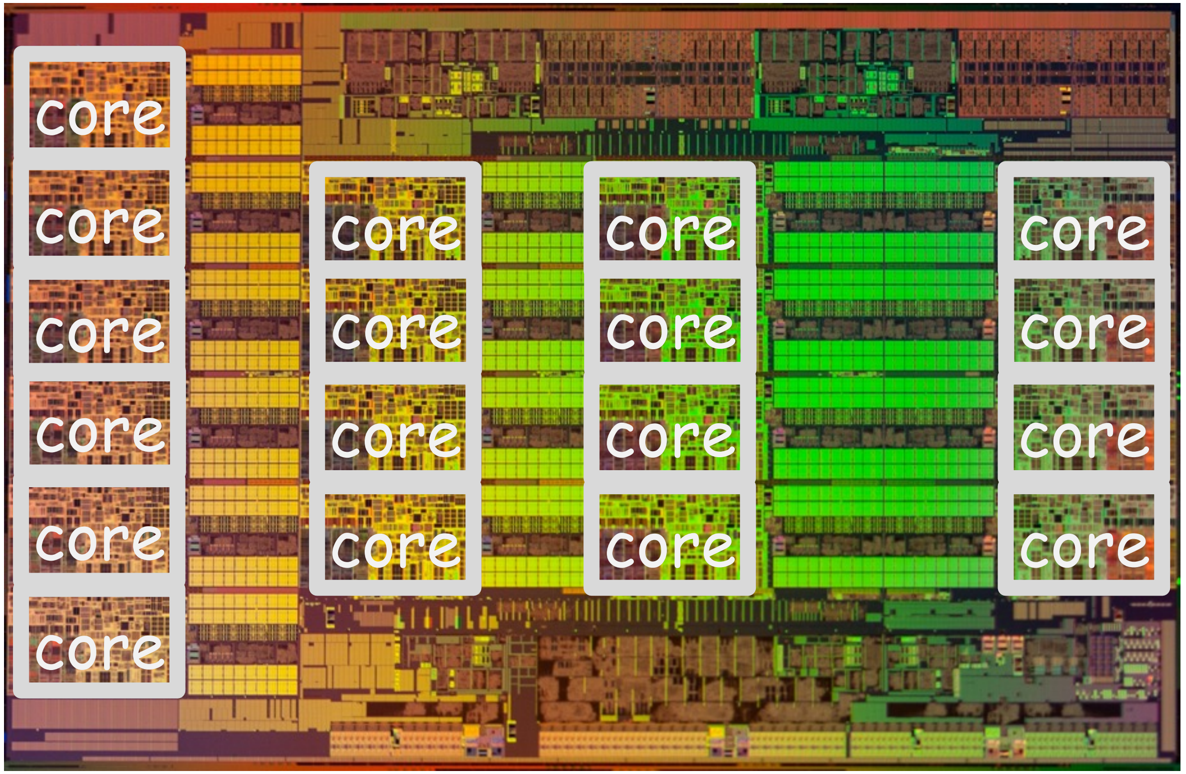
- At Penn
- Lt Gillon, Eckert and Mauchley
- Cost $486,804.22, in 1946
- 5000 ops/second
- 19K vacuum tubes
- Power = 200K Watts



67 m^3

Where are we today? Intel Xeon Broadwell-E5

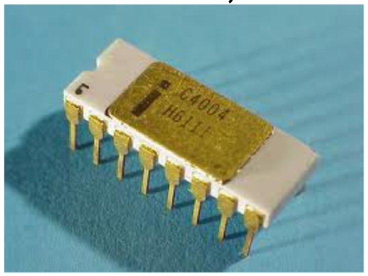
- 5.5+ billion transistors
- 18 cores
- 45 MB L3 cache
- 2.3 GHz
  - Turbo 3.6 GHz
- Roughly 145W



456 mm^2

Information Technology (IT): Four Decades of Exponential Growth

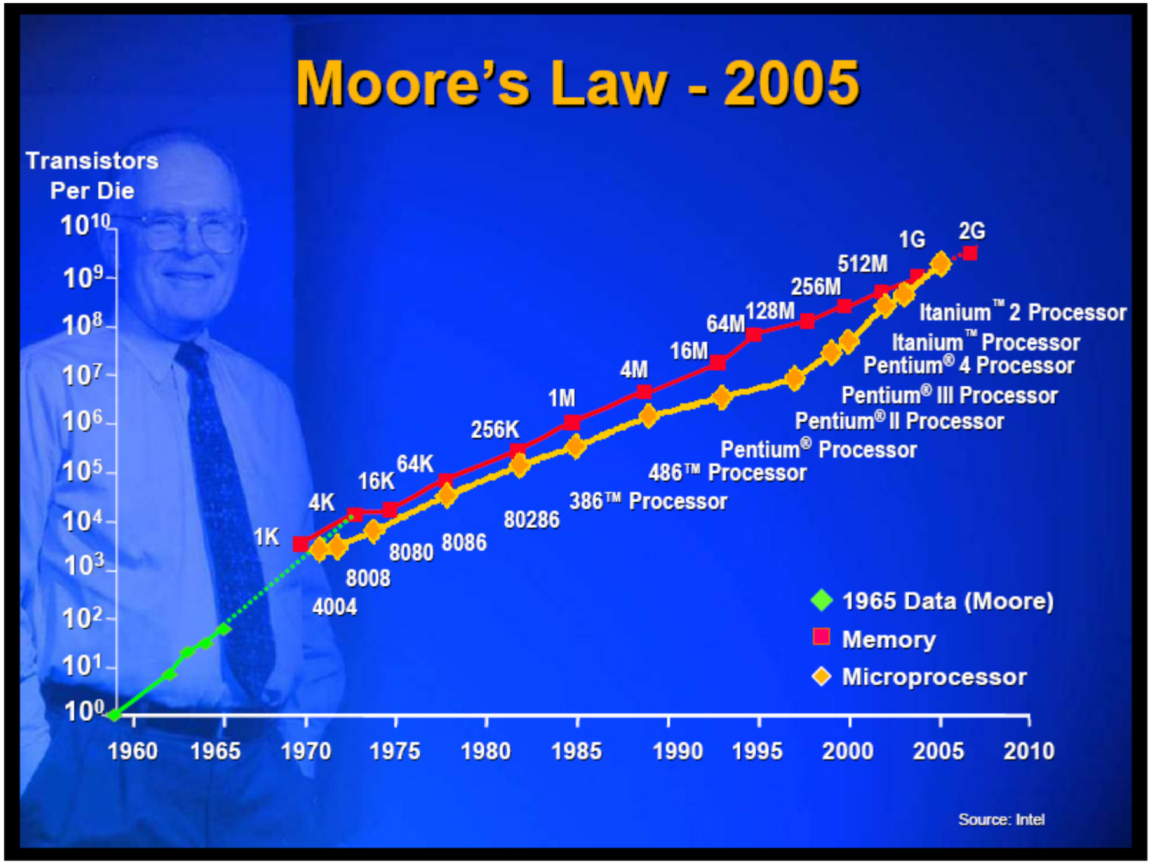
Intel 4004, 1971



92,000 ops/sec

Intel Xeon, 20117

166,000,000,000 ops/sec



IT is at the core everything we do & has become an indispensable pillar for a modern day society!

What Does this All Mean?

Microprocessor performance growth in perspective:

"Unmatched by any other industry"

[John Crawford, Intel Fellow, 1993]

Doubling every 18 months (1998-2008): roughly 100X

- Cars travel at 20,000 KM/H; get 50 ml/100 KM
- Air travel: Porto to Talinn in 1.5 min (MACH 100)
- Wheat yield: 10,000 bushels per acre

For four decades 2x transistors every 2 years

Moore's Law:

- More trans, faster CPU's
- Clocks from 1 MHz to 1GHz
- Parallel microarchitecture: superscalar + pipelining
- Perf: 2.5x per 2 years

Lowered chip power with lower voltages

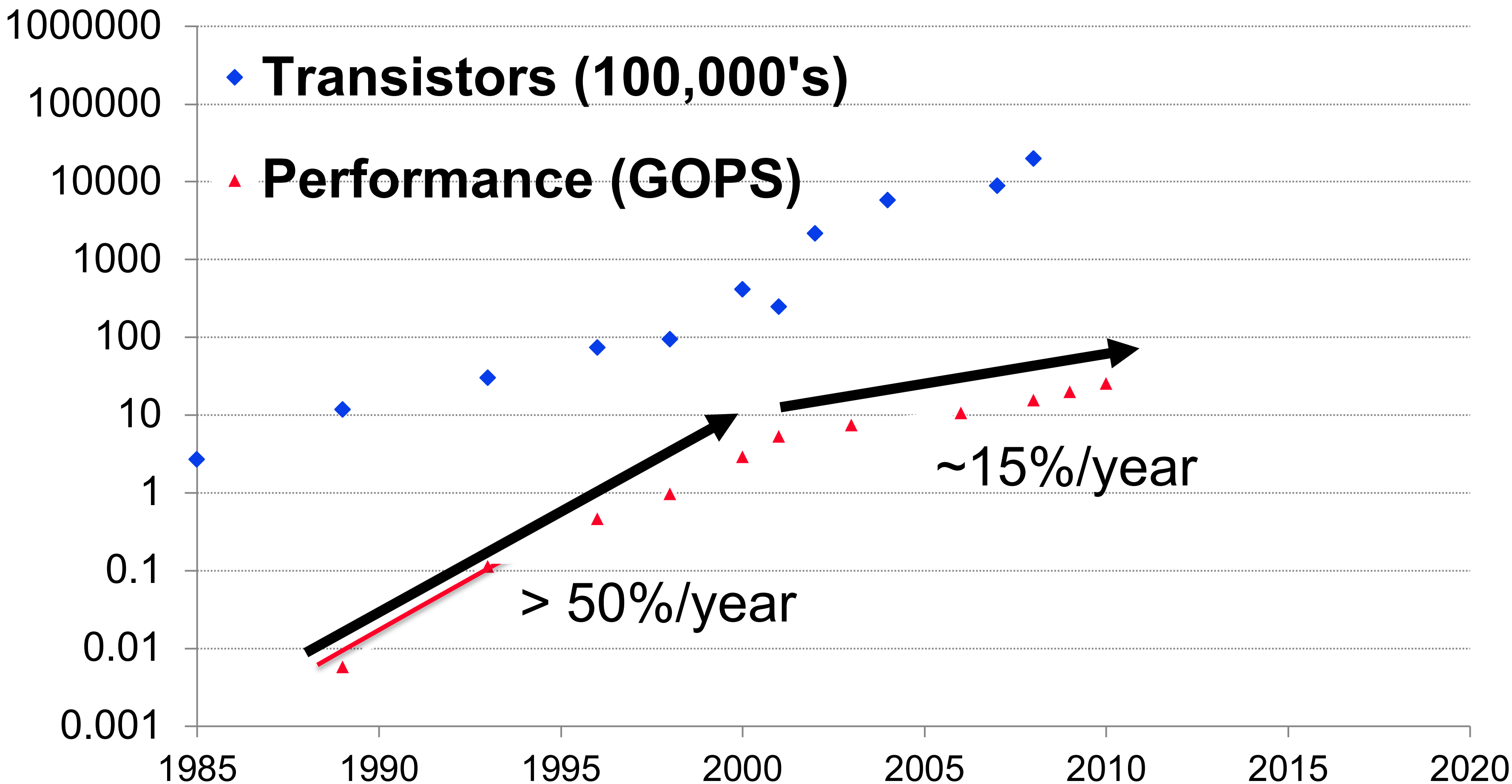
Ran sequential code, one thread/program

CPU in 1980

CPU in 1990

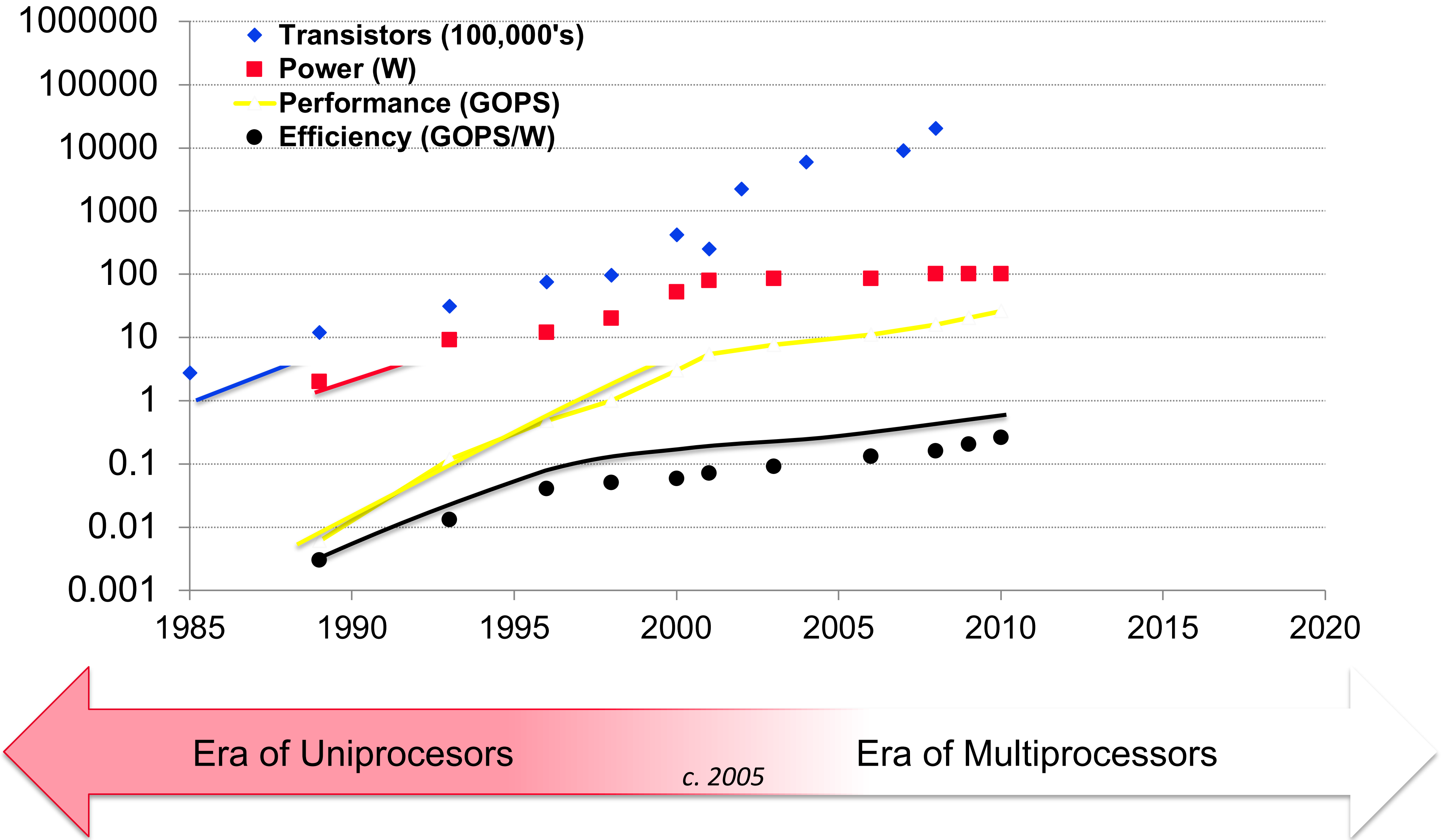
CPU in 2000

The chips started getting bigger not faster



[source: Prof. Tom Wenisch, Michigan]

Why? Must keep power at ~100W!

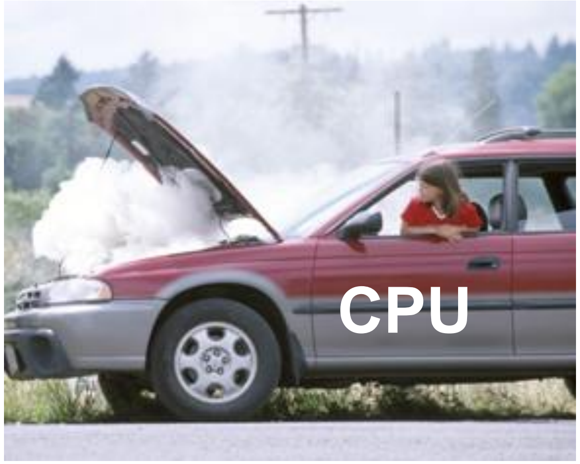


What is Power?

Power $\propto V^2F$

V = Operating voltages

F = Clock frequency

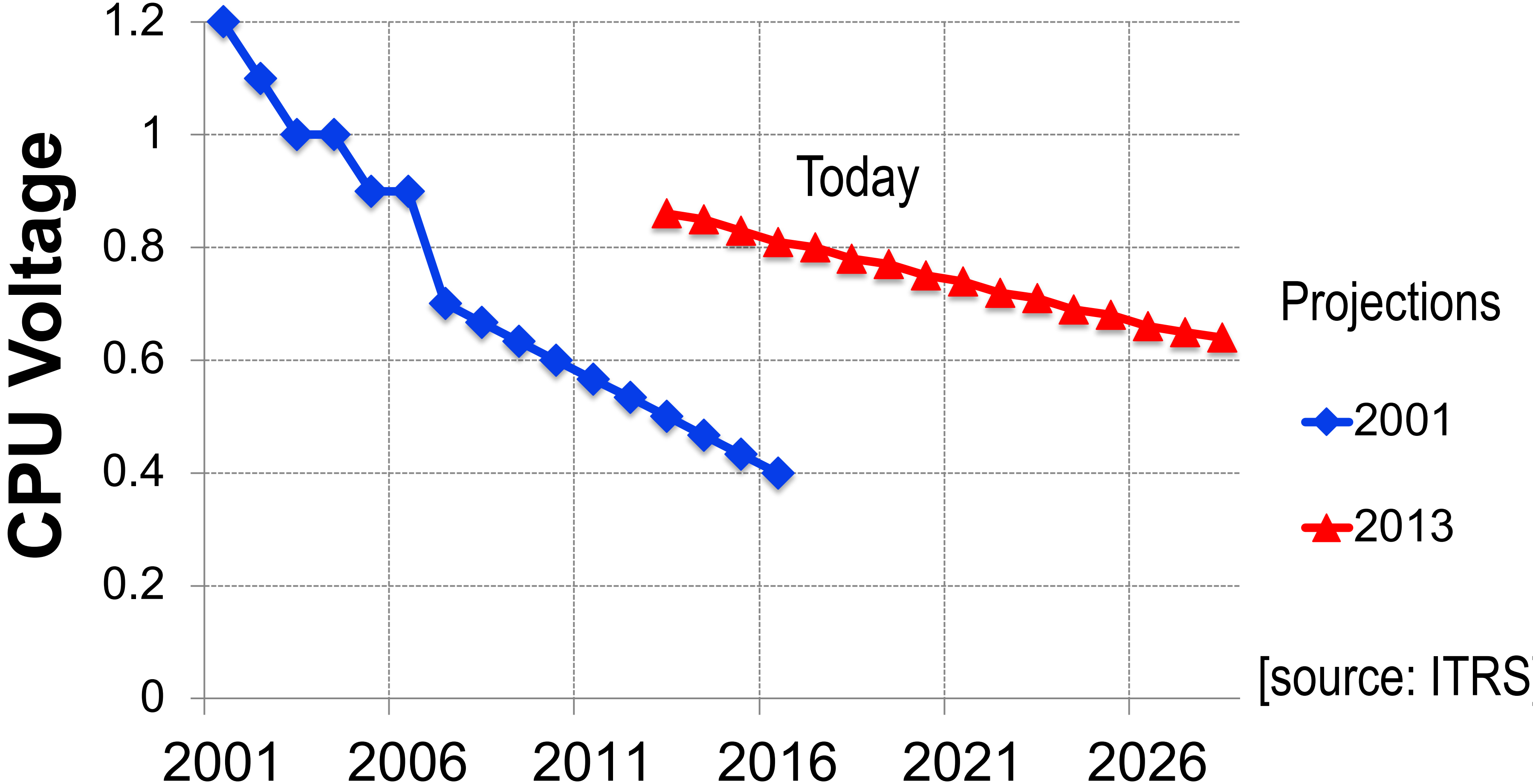


CPU type	Power	Constraint
Mobile	< a few W	Battery usage
Laptop	< 10s of W	Battery + heat
Desktop/Server	~ 100 W	Cooling
Supercomputer	~ 300 W	Cooling + electricity

What happened to Power?

- Voltages used to go down
  - From 5v (1970's) to 1v (2000's)
  - Power $\propto V^2F$
  - Power went down
  - But, voltage is squared!
  - Gave us enough room to increase clock frequency

Voltages stop going down!



[source: ITRS]

Voltages stopped going down

With more transistors:

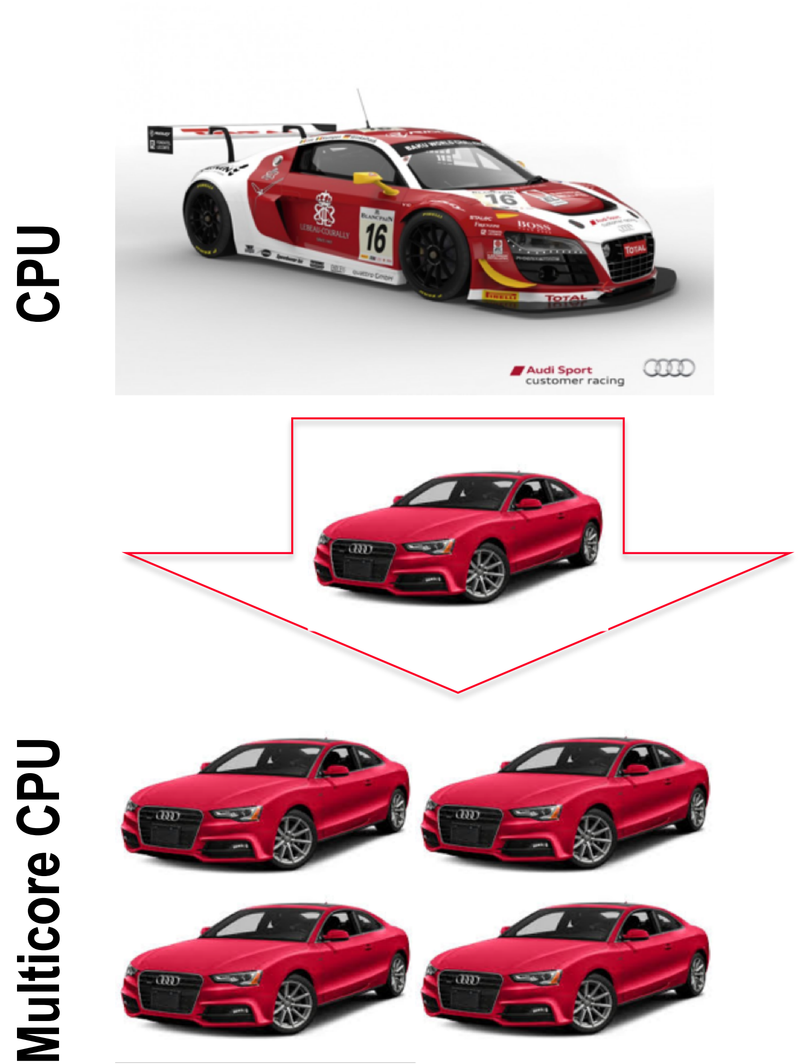
- Scale back core complexity
- Use cores of yesteryear
- Each core ➔ fewer joules/op

Analogy:

- Not quite a race car
- Handles nearly all cases

But, now……

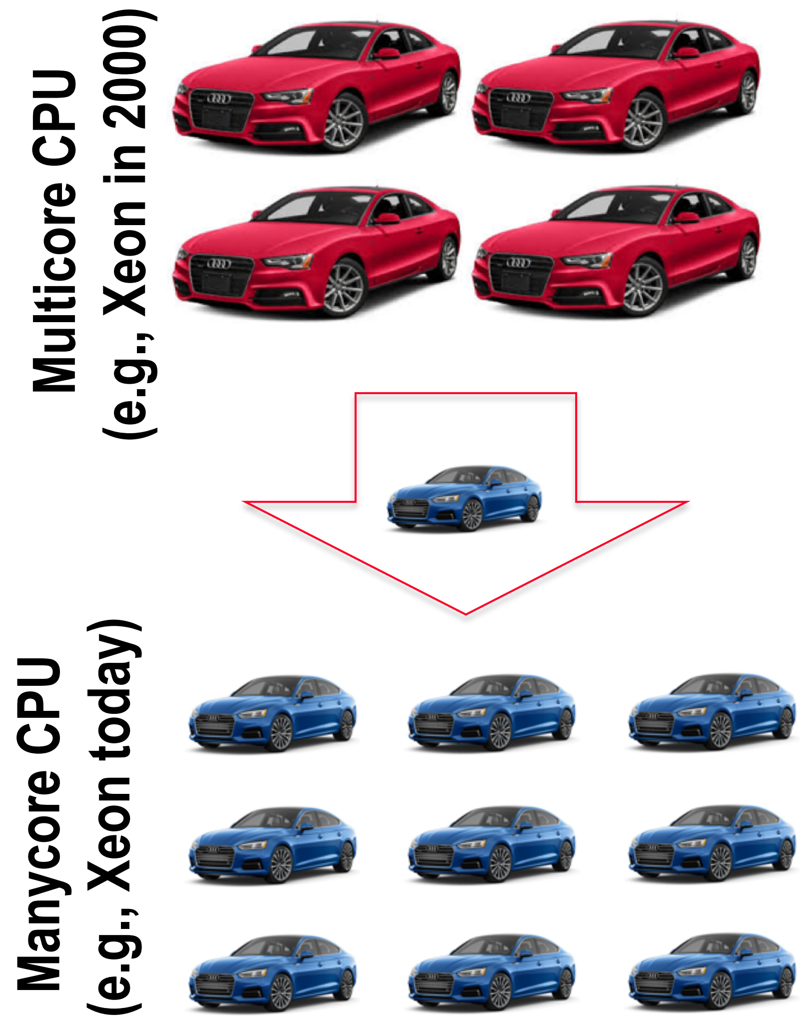
- Need parallel software!



From Multicore to Eco-Mode

Allow for adjustable frequency & voltage:

- More of the same core
- More parallelism on slower cores



Analogy:

- Audi in ”eco/blue” mode
- Uses less gas
- Not as spiffy

Turboboost

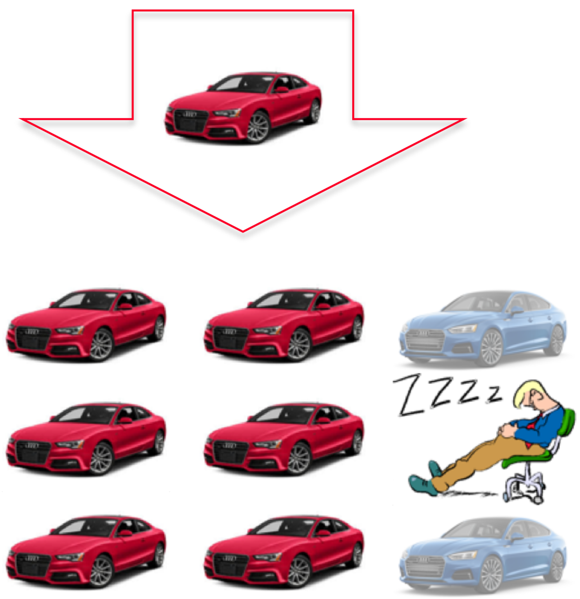
Allow for adjustable speed:

- More of the same core
- Less parallelism on faster cores

Analogy:

- Audi in ”sport” mode
- But uses more gas
- Can run fewer cores faster together

Manycore CPU (e.g., Xeon today)



Manycore CPU (e.g., Turboboost in Xeon)

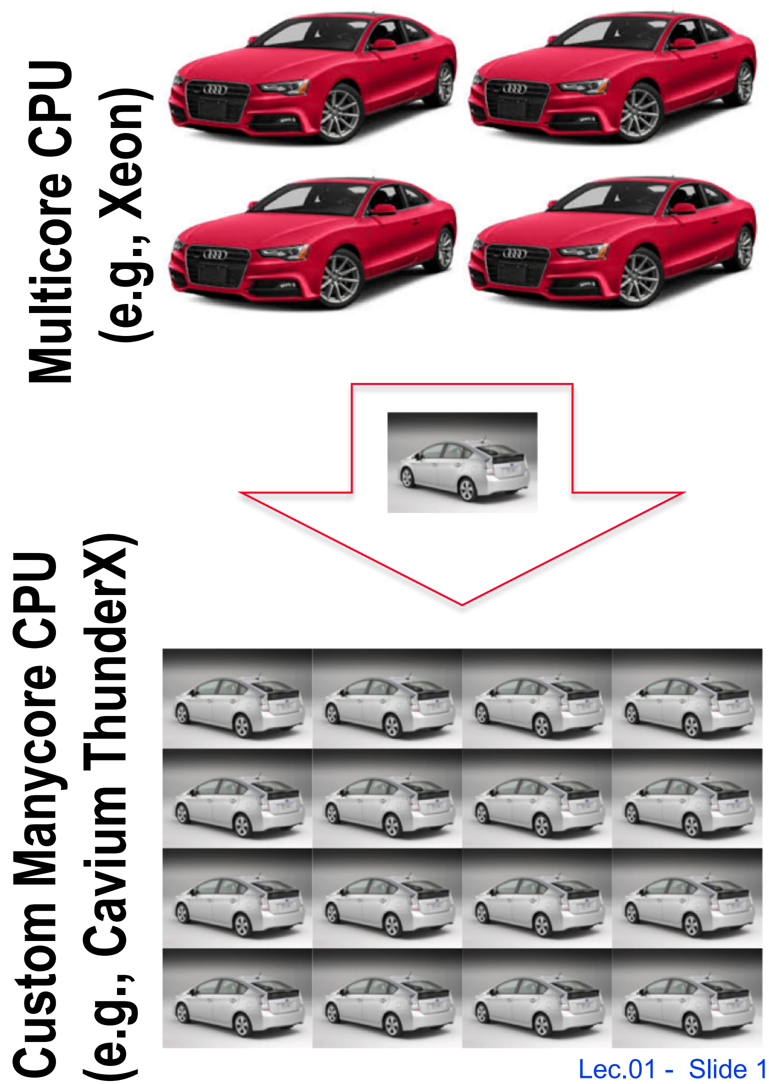
Custom Manycore

Can no longer afford the general-purpose/high-perf

- Custom cores
- Reduced complexity
- Mobile efficiency

Analogy:

- Prius can handle city well
- Lot more efficient
- But, limited at speed
- More work from much higher parallelism



Exercise: Cores vs. Clock

Power $\propto V^2F$

Assuming that voltages remain the same, to quadruple the number of cores while keeping the power the same, how much slower should the cores run at (slower clock rate)?

Four cores, same power, clock = F/4
With original F = 2 GHz, new F = 500 MHz

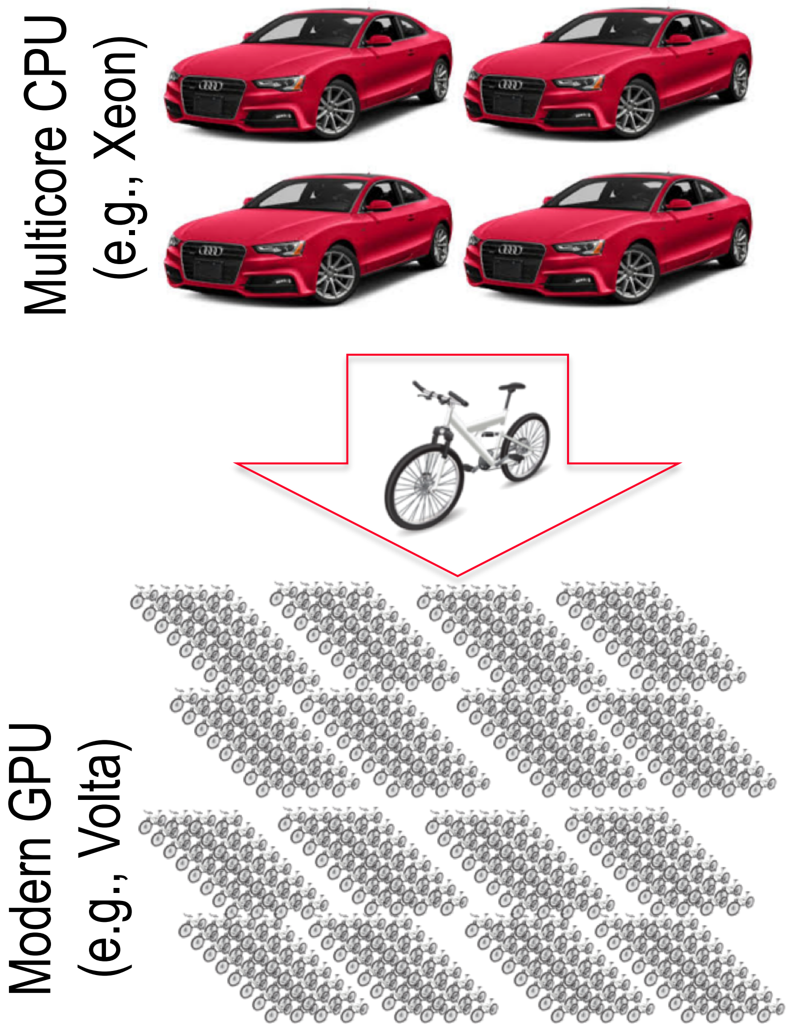
GPU's are Massively Parallel Manycores

Further reduction in complexity:

- Minimal core (EPFL 2nd year)
- Maximize number of cores
- Optimized for arithmetic density per silicon area
- Same program runs on independent data

Analogy:

- Groups of "spinners" tuned to music
- All spin at the same speed
- 1000's of spinners in parallel



Custom Computing

Reconfigurable (FPGA)

- Program in HDL
- Irregular parallelism
- Microsoft's Catapult

Accelerator

- Program in DSL
- App-specific, min. work
- Google's TPU

Roadmap

- Technology
  - Moore’s Law
  - Parallelism
- Datacenters & Centralization
  - Economies of scale
  - Metrics
- Service-Oriented Computing
  - Virtualization
  - Cloud

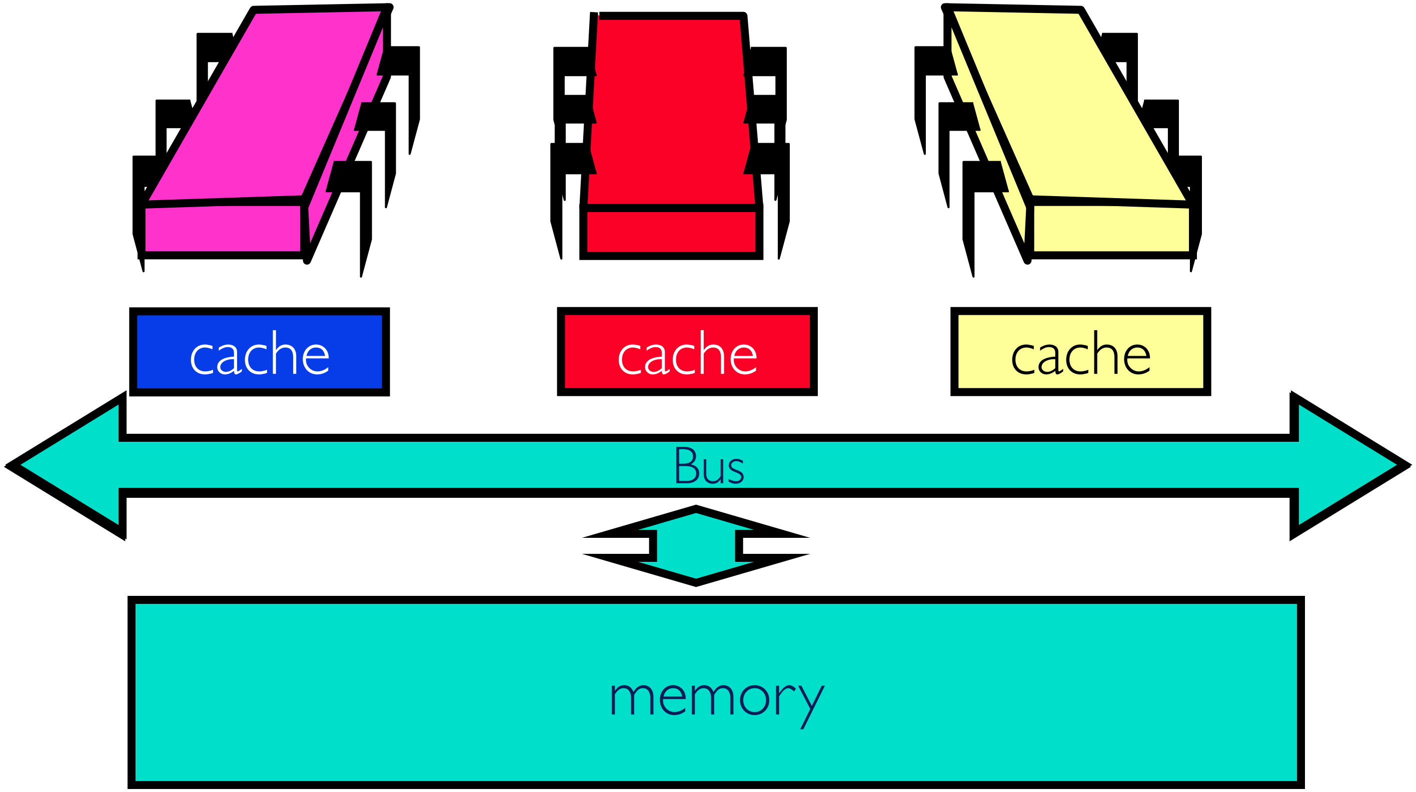
Parallel or Multiprocessor Architecture

- Abstract models are (mostly) OK to understand algorithm correctness and progress
- To understand how concurrent algorithms actually perform
- You need to understand something about multiprocessor architectures
- Detailed nuts & bolts? EPFL courses

Pieces

- Processors (also called CPU or cores)
- Threads
- Interconnect
- Memory
- Caches

Simple Multiprocessor



Old School vs. New School

Before 1990's:

- ▷ Processors on different chips
- ▷ Nearby processors share memory resources

After 1990's:

- ▷ On-chip processors (called Multicore/Manycore) and off-chip
- ▷ Nearby cores shared memory resources

Understanding the Pieces

- Lets try to understand what the pieces that make the multiprocessor machine are
- And how they fit together

Processors

- Cycle:
  - Fetch and execute one instruction
- Cycle times change
  - 1980: 10 million cycles/sec
  - 2018: 2,000 million cycles/sec

Computer Architecture

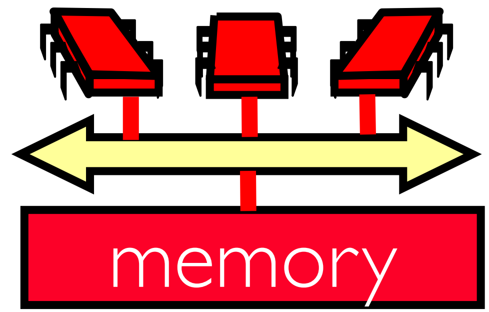
- Measure time in cycles
  - Absolute cycle times change
- Memory access: ~100s of cycles
  - Changes slowly
  - Mostly gets worse

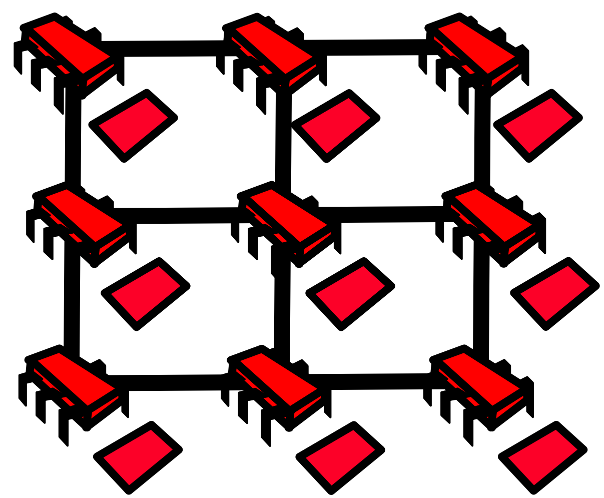
Threads

- Execution of a sequential program
- Software, not hardware
- A processor can run a thread
- Put it aside
  - Thread does I/O (talks to disk and network)
  - Thread runs out of time
- Run another thread

Interconnect

- Bus
  - Broadcast medium
  - Connects
    - Processors to memory
    - Processors to processors
- Network
  - Switch-based fabric
  - Connects
    - One node to another
    - Each node with its own processor, memory, …

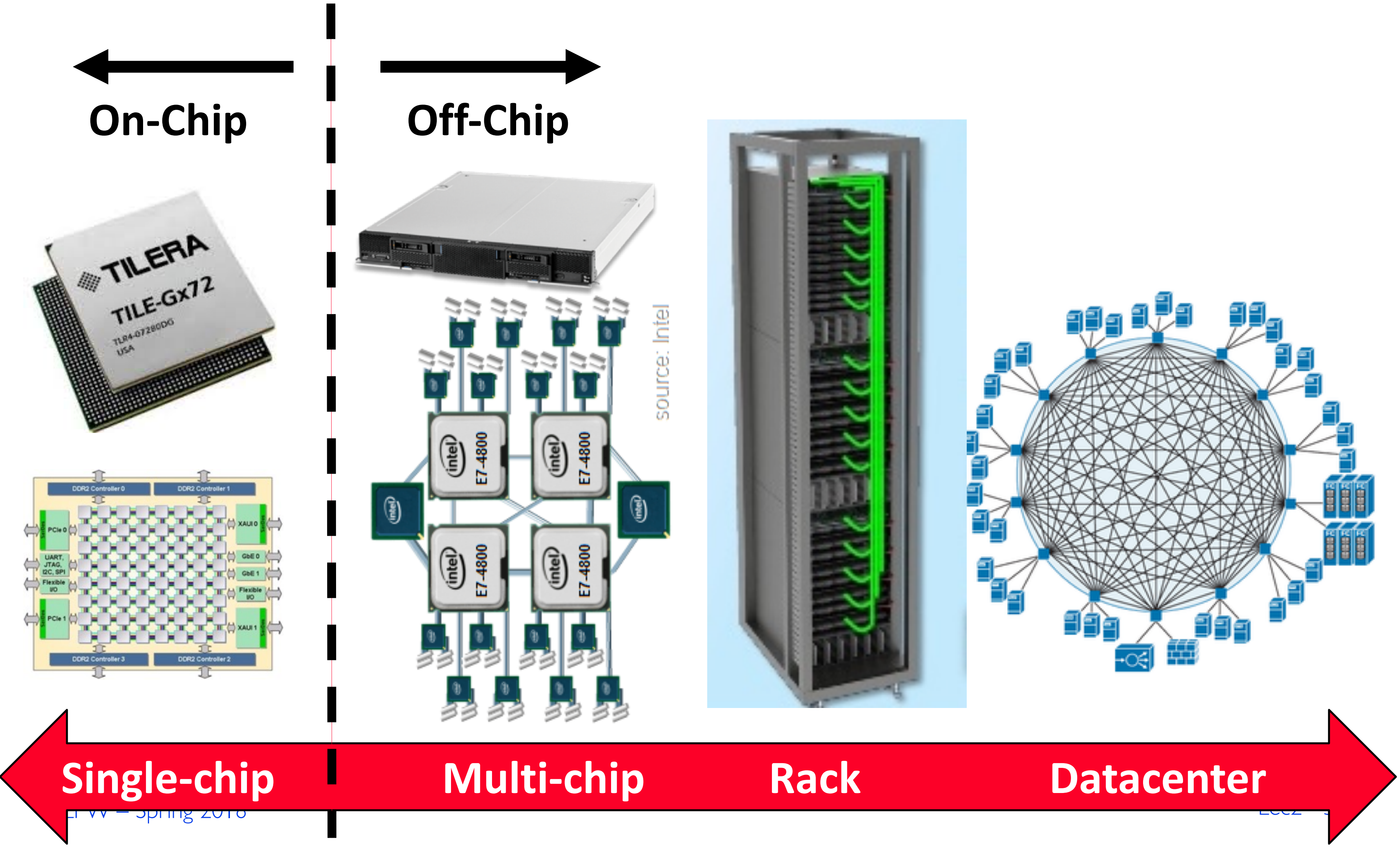




Interconnect (also called network)

- Interconnect is a finite resource
- Processors can be delayed if others are consuming too much
- Avoid algorithms that use too much bandwidth

Interconnect/Network for multiprocessors



Communication Models

- Shared Memory
  - Communication is unstructured, implicit in loads and stores
  - Easier to program, but harder to scale
  - Single node: mobiles, laptop/desktop, single server
- Message Passing
  - Structure all communication as messages
  - Harder to program, but easier to scale
  - Multiple nodes: rack of servers
- Data Parallel
  - Structure computation over groups of independent data
  - Single node: GPU
  - Multiple nodes: rack of servers (CPU or GPU)

Software layering

Hadoop, Spark, Pregel, GraphLab, FaRM, TensorFlow… — High-level frameworks

Cilk, OpenMP, TBBs, Pthreads, CUDA, MPI … — Libraries / Language extensions

C/C++, Java, Scala, Python, … — Programming languages

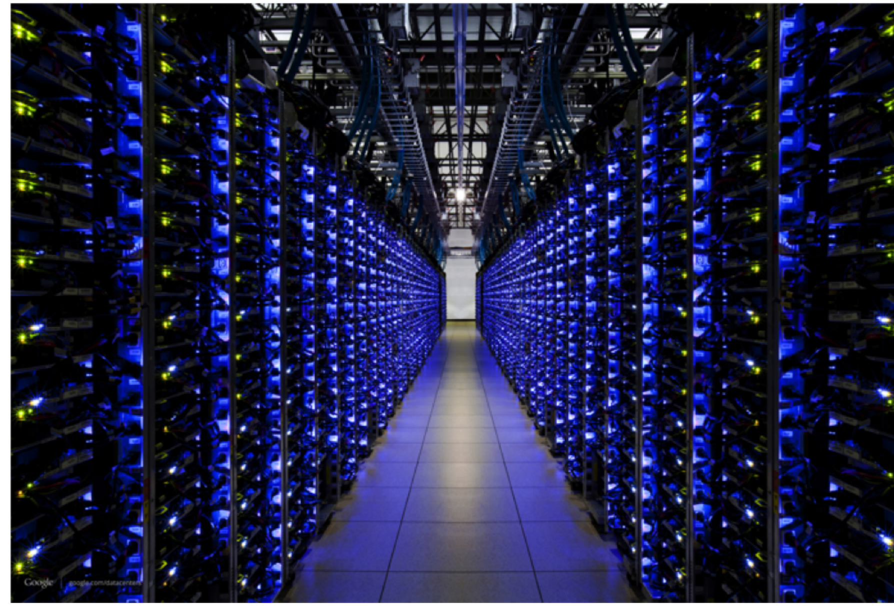
Single node: multicore, GPU | Multinode: cluster, datacenter, supercomputer — Hardware

Example platforms: Mobile

- E.g., iPhone, Surface, Smartwatch
- Small devices used in everyday life
- Heterogeneous CPU/GPU devices
  - Shared memory CPU
  - Data Parallel GPU
- Four low-power cores
  - 0.25 to 1 Watt per core
- Total power < 15 W
- Price < 1000 CHF

Example platforms: Datacenters

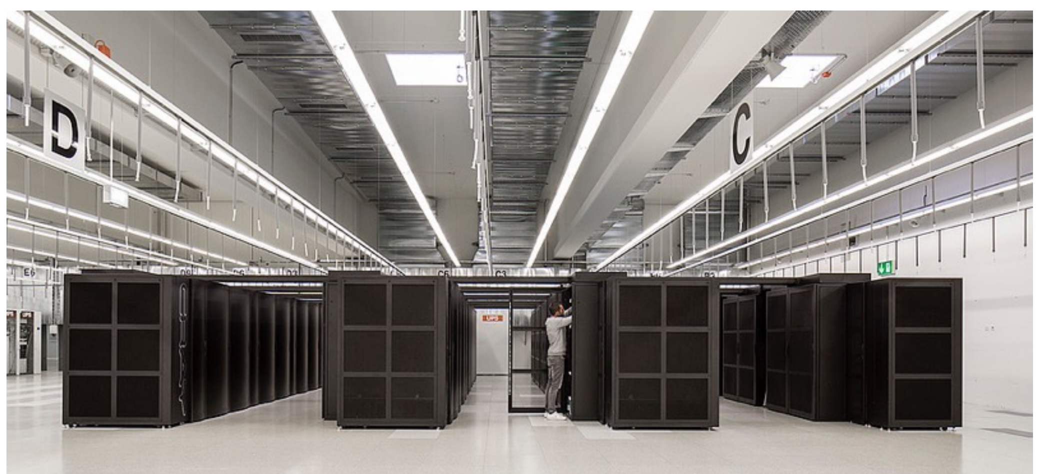
- E.g., Amazon, Facebook, Google, Microsoft
- Run all online services: search, social media, e-commerce
- Shared memory within server
- Message passing across
- Dozen cores/server ~100 W
- Datacenter power ~20 MW
- Price ~ 3 billion CHF



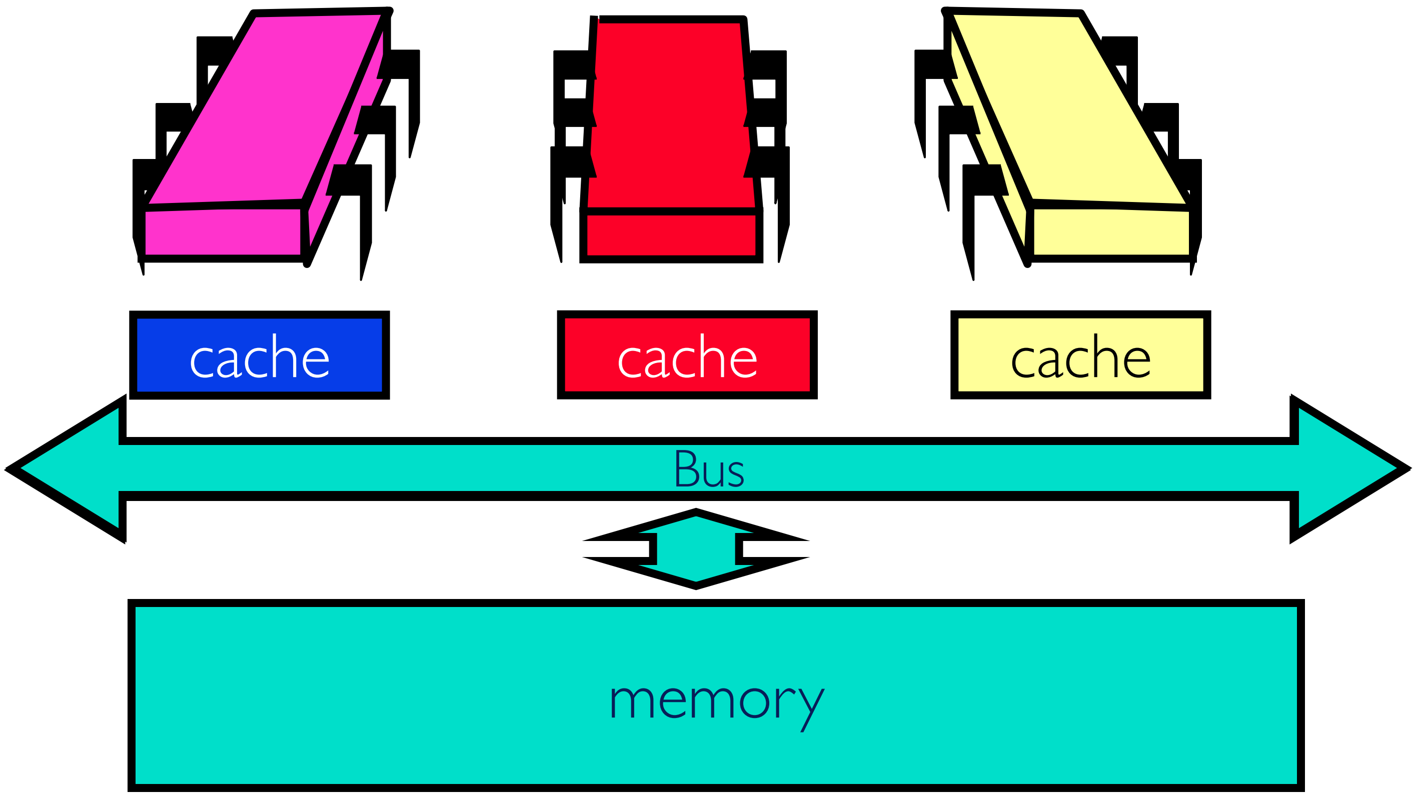
Example platforms: Supercomputers

- Popular software library: MPI (message passing interface)
- Sharing memory is too expensive at massive scale
  - Can connect commodity systems together to form large parallel machine
- E.g., "Piz Daint" comprised of ~6K independent systems
  - Using MPI, can program all of them as one

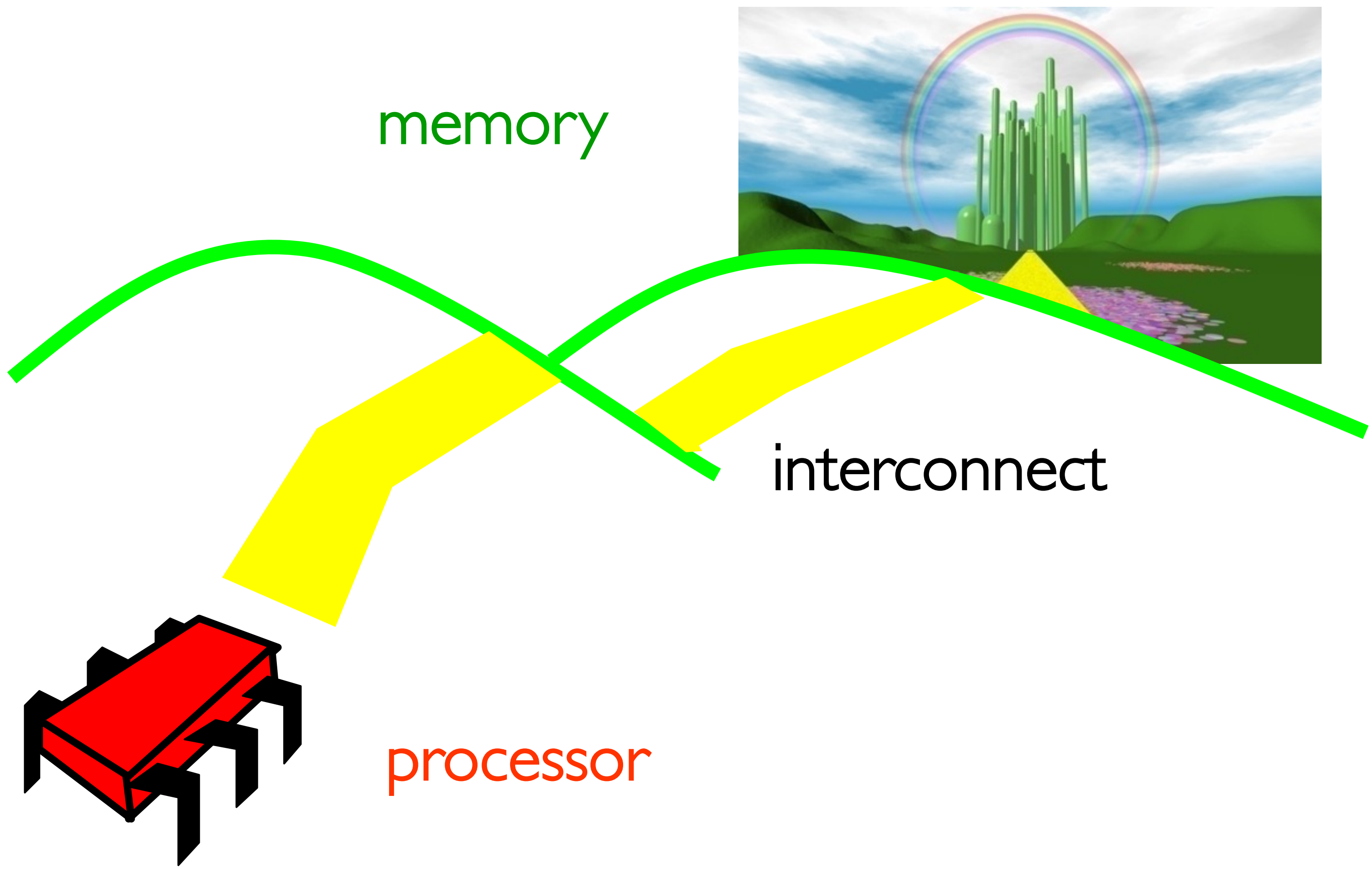
CSCS



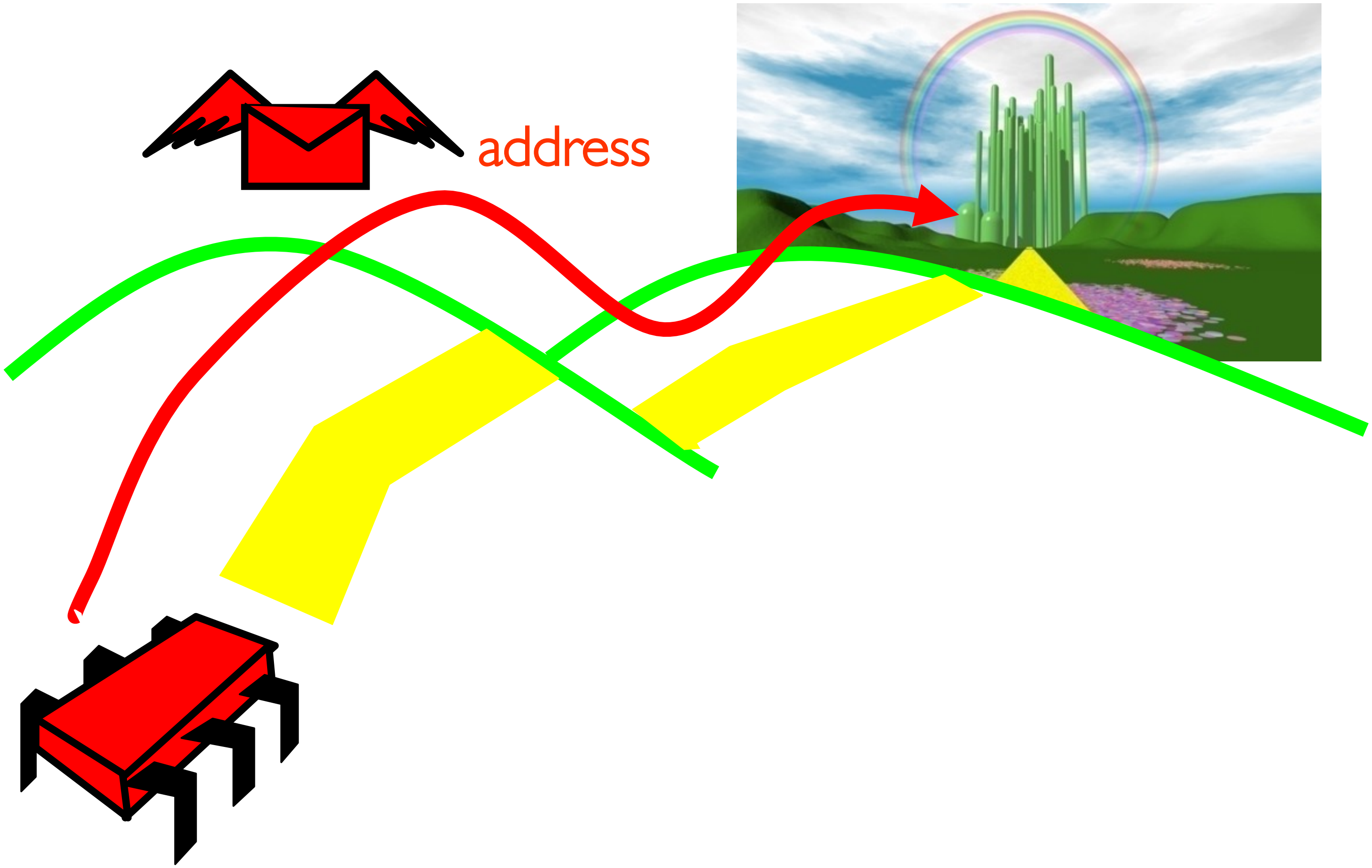
Back to Simple Multiprocessor: Shared Memory



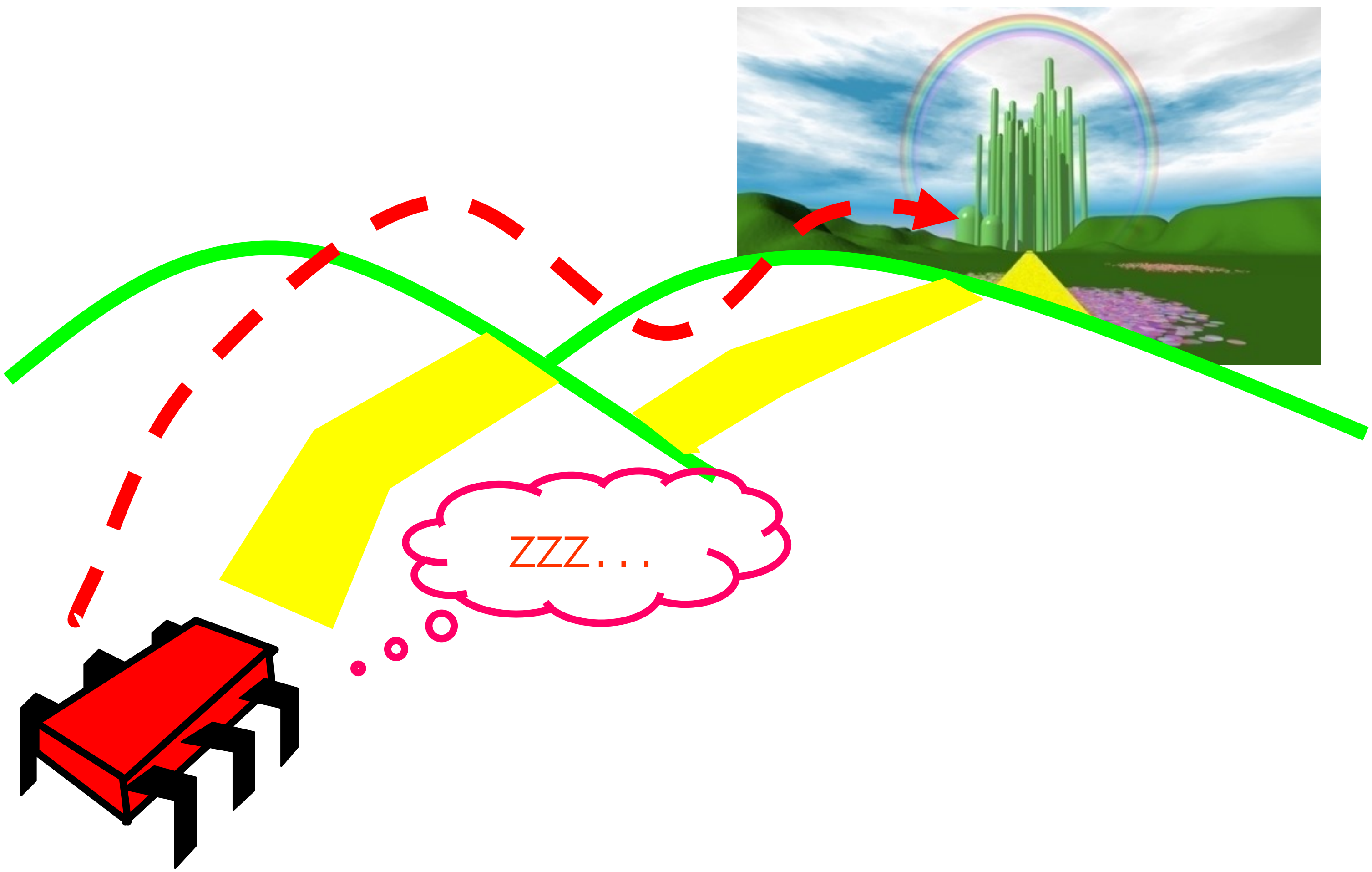
Processor and Memory are Far Apart



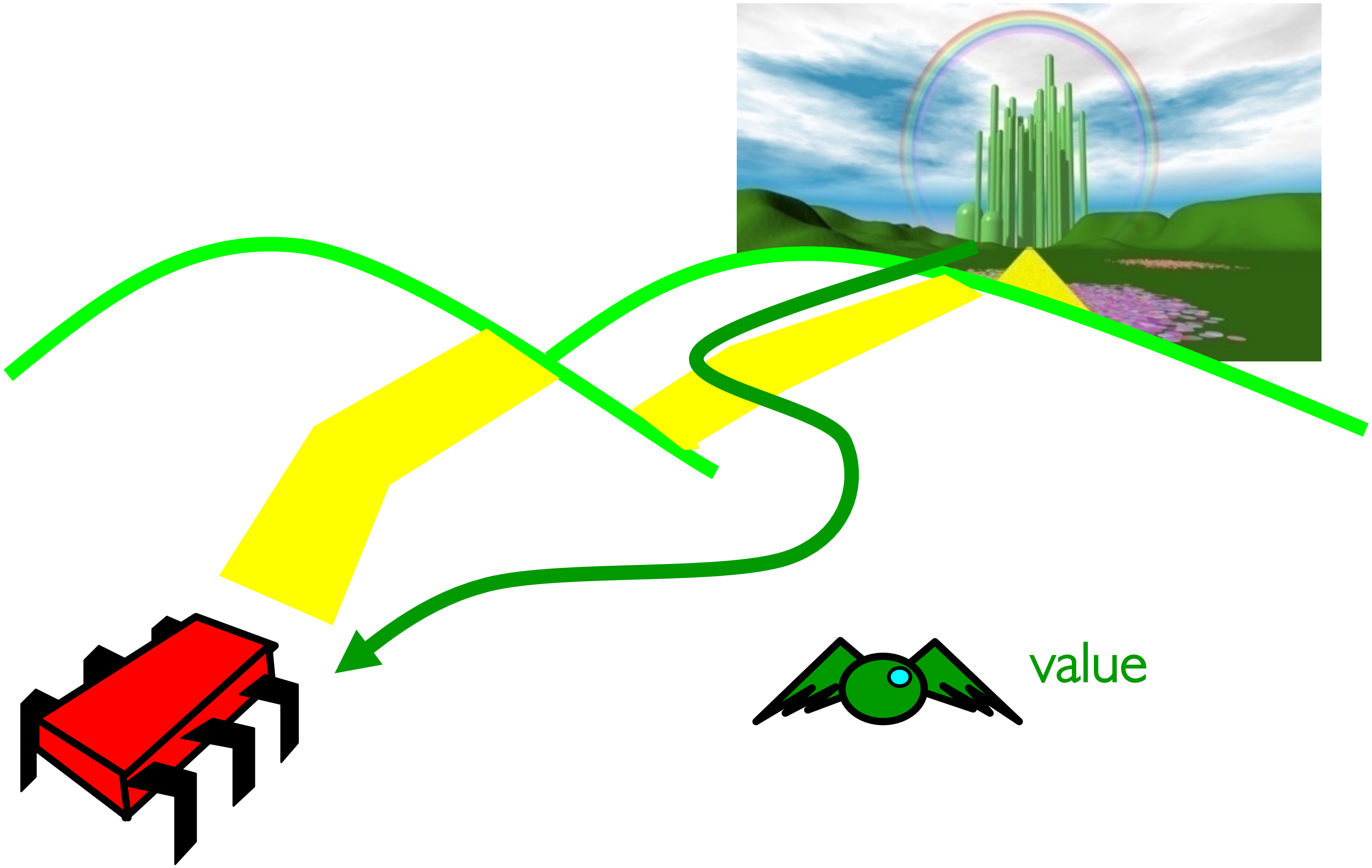
Reading from Memory



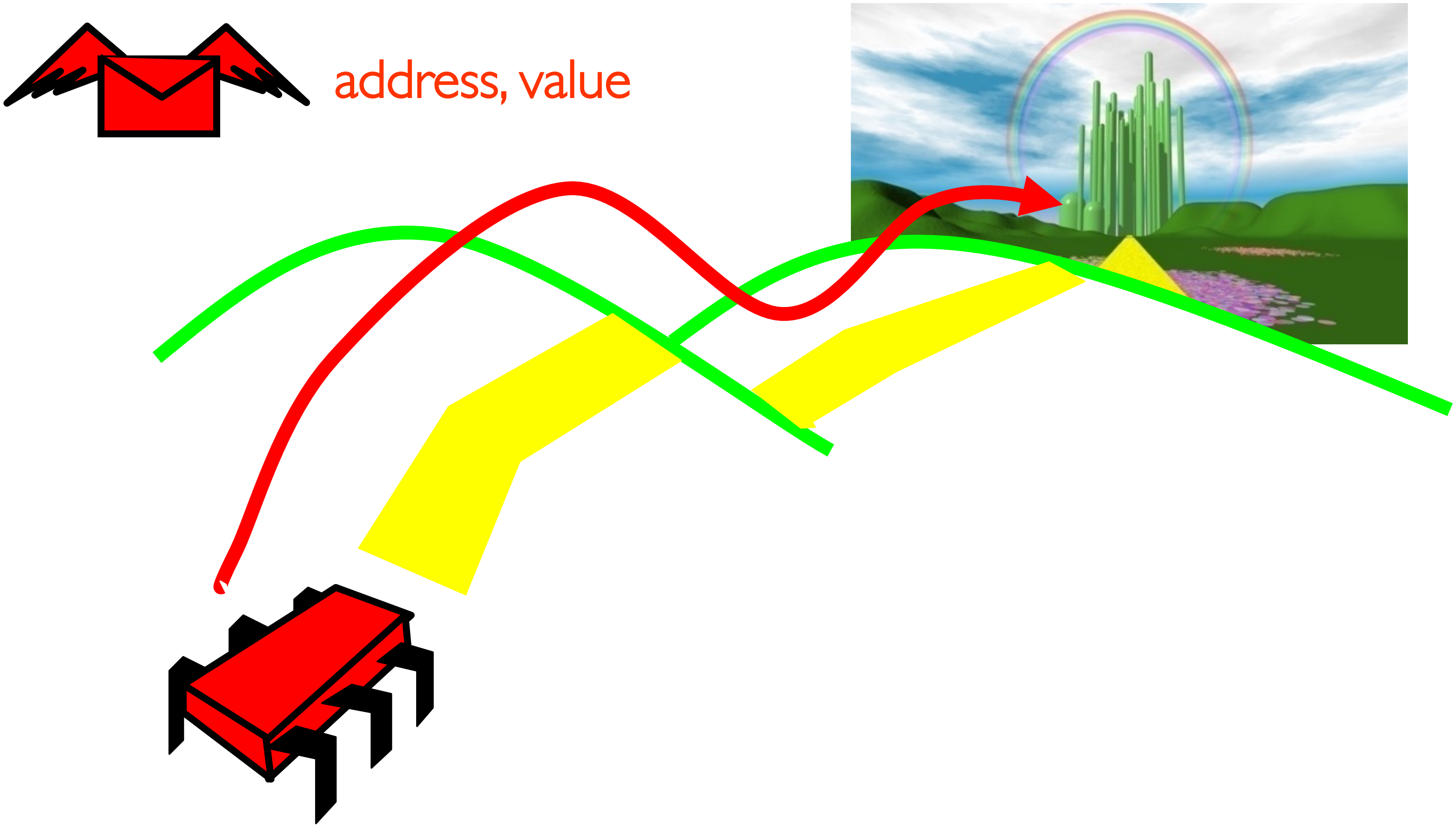
Reading from Memory



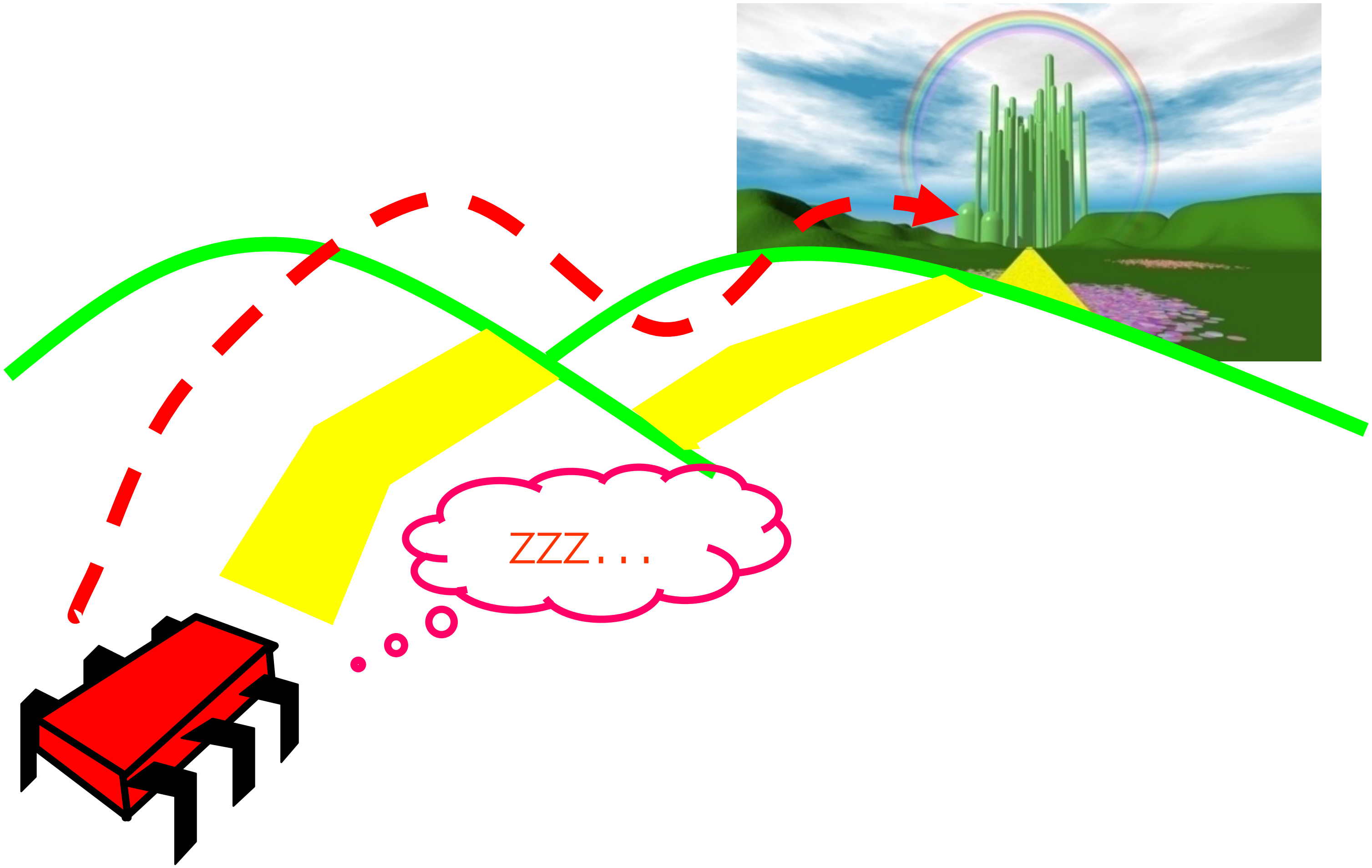
Reading from Memory



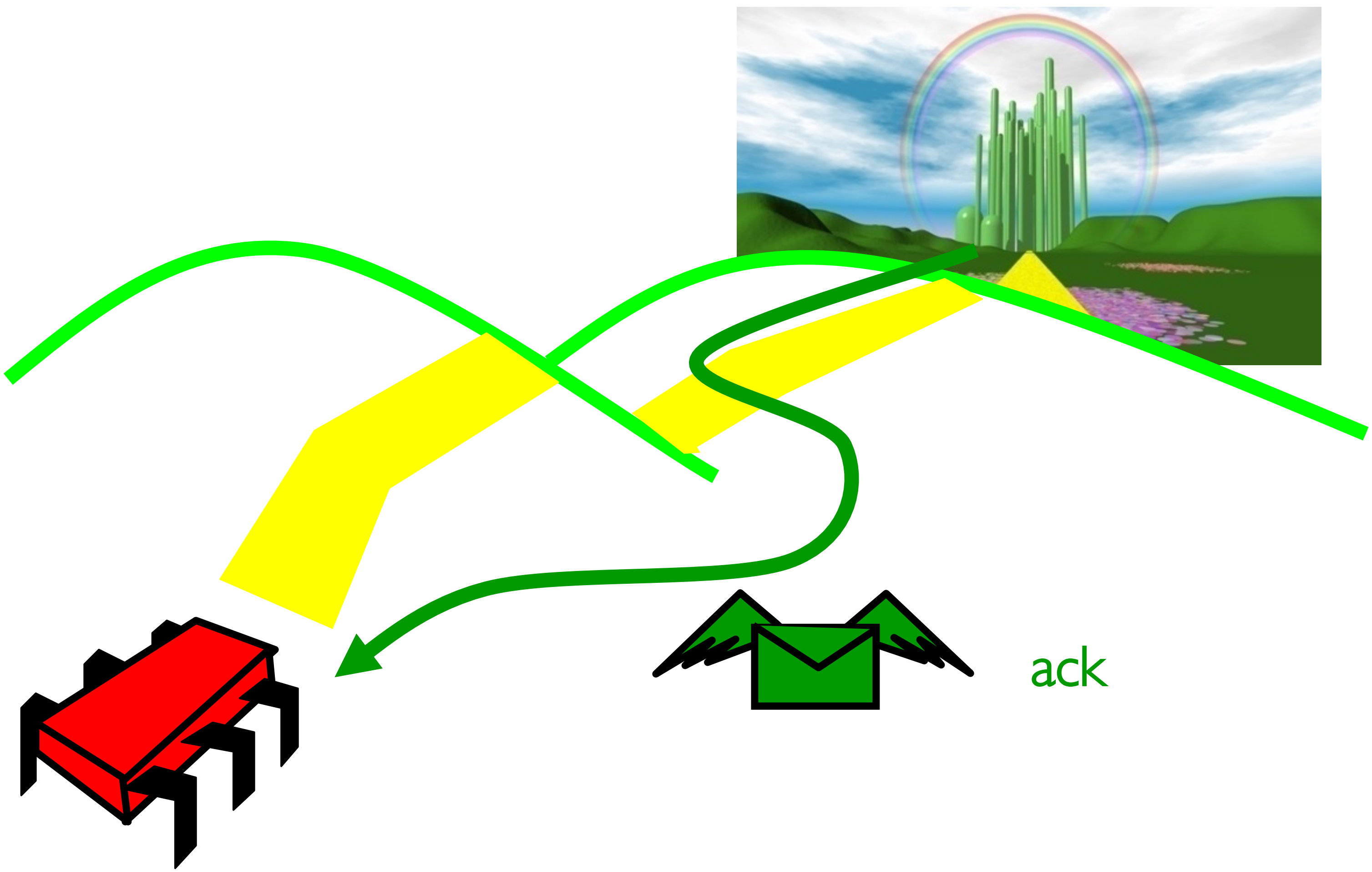
Writing to Memory



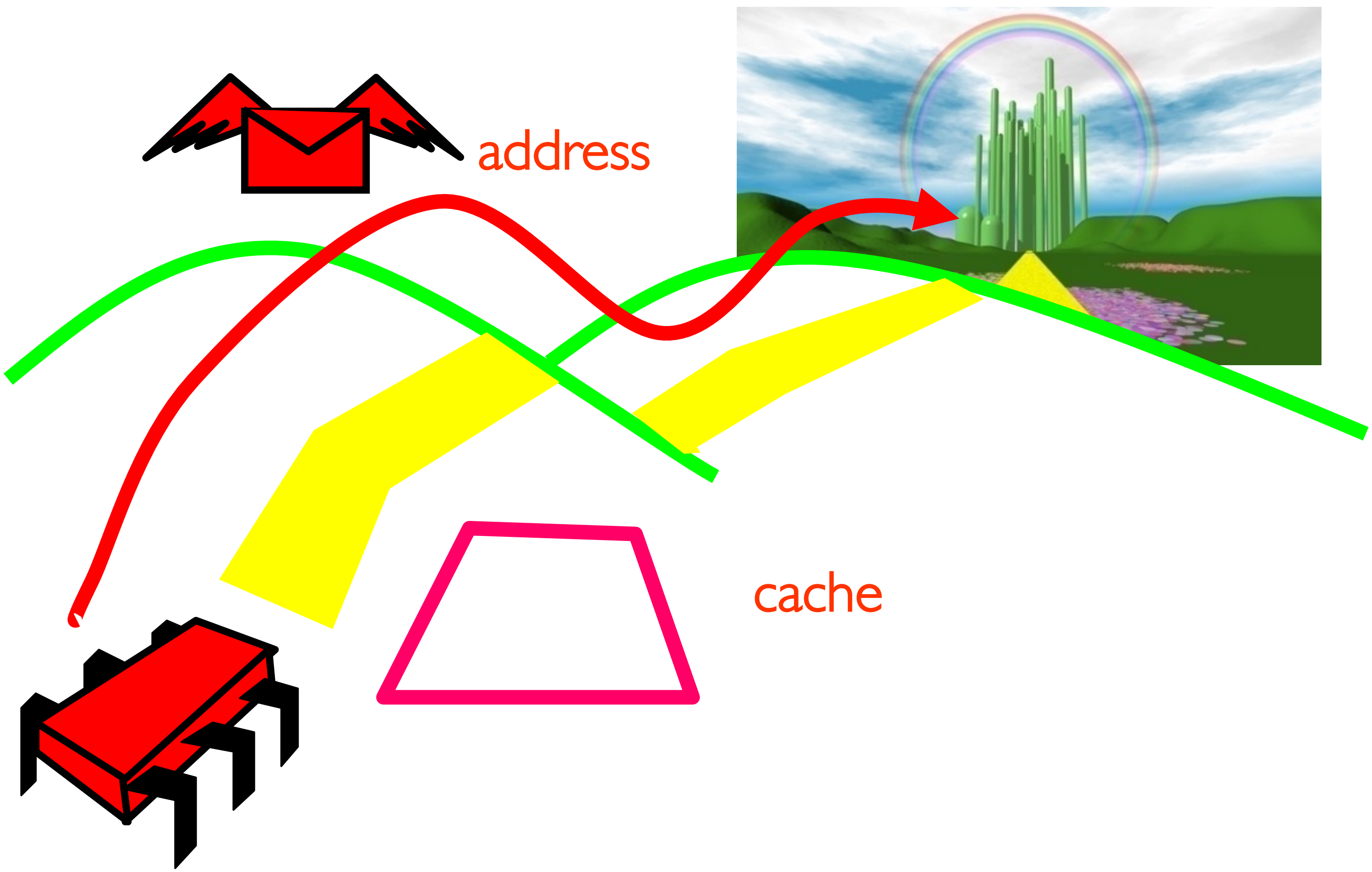
Writing to Memory



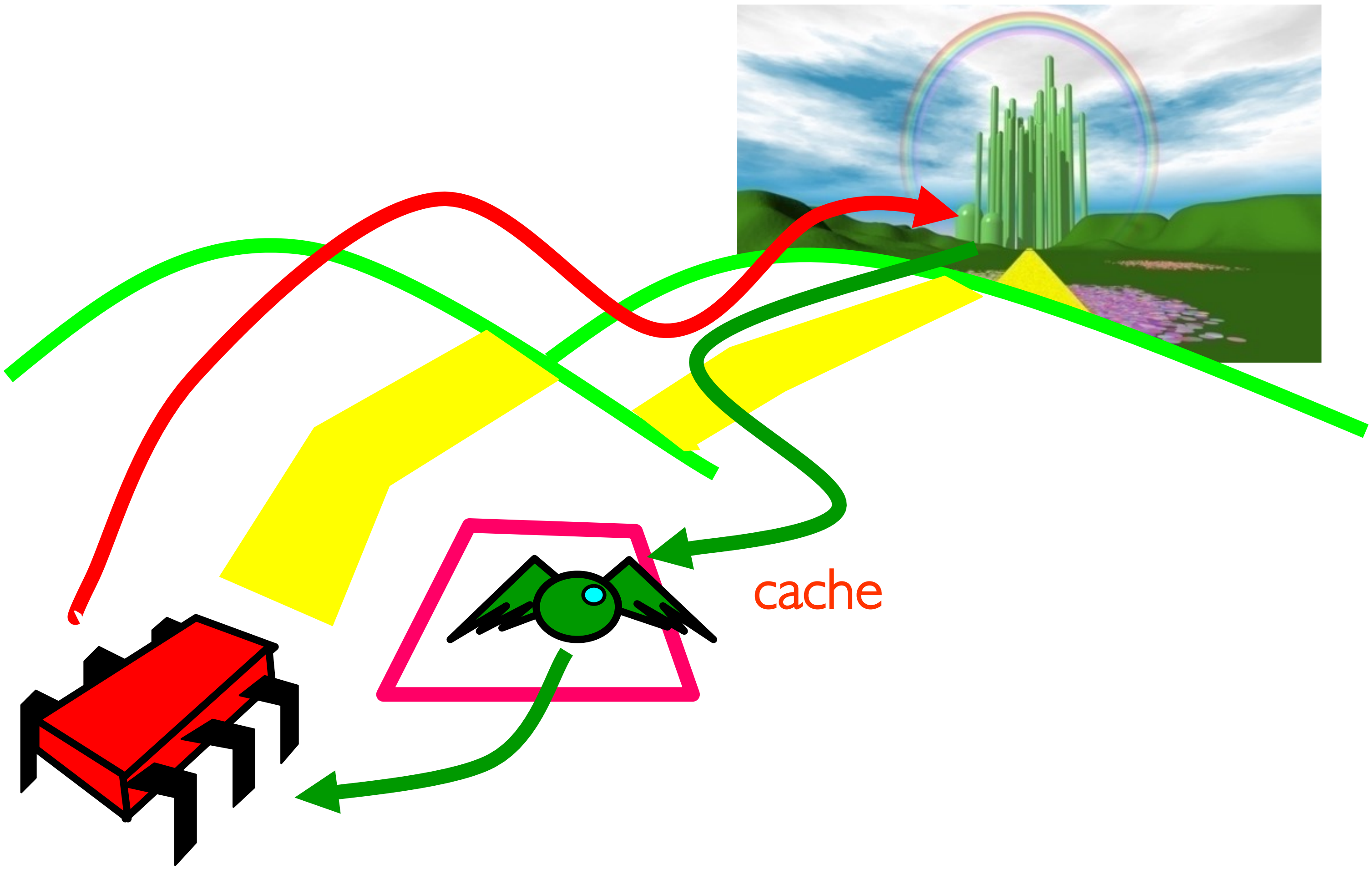
Writing to Memory



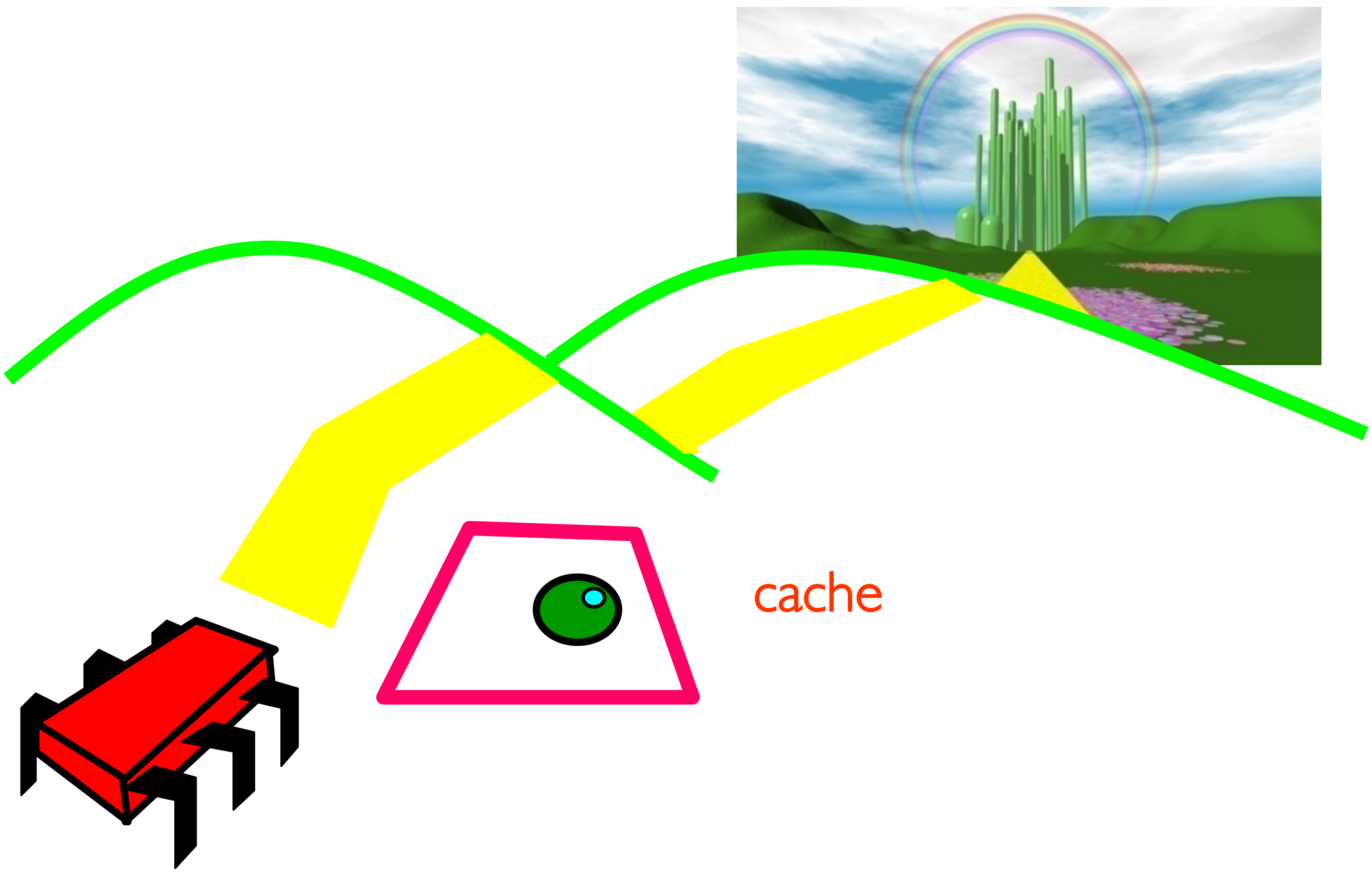
Cache: Reading from Memory



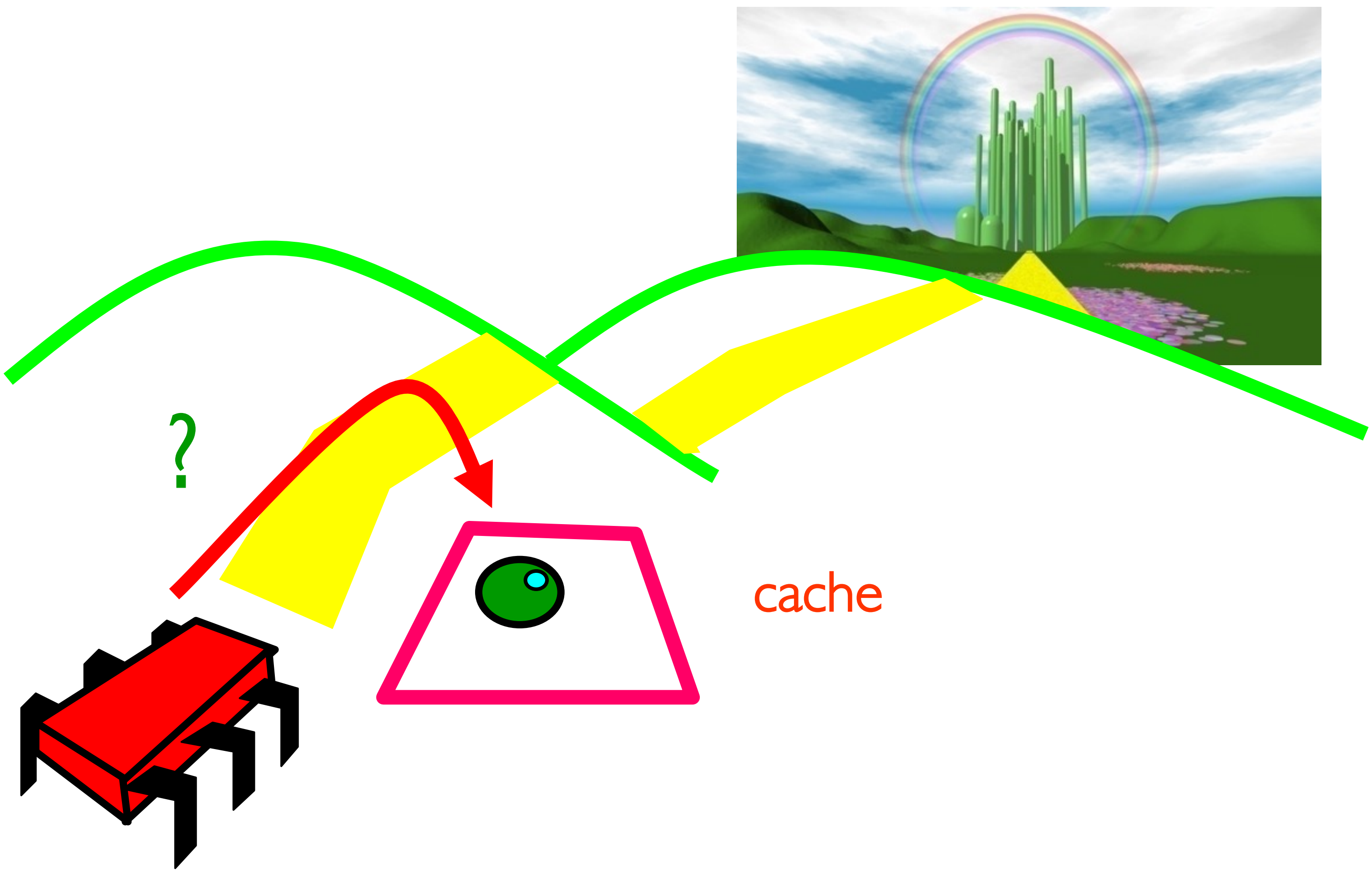
Cache: Reading from Memory



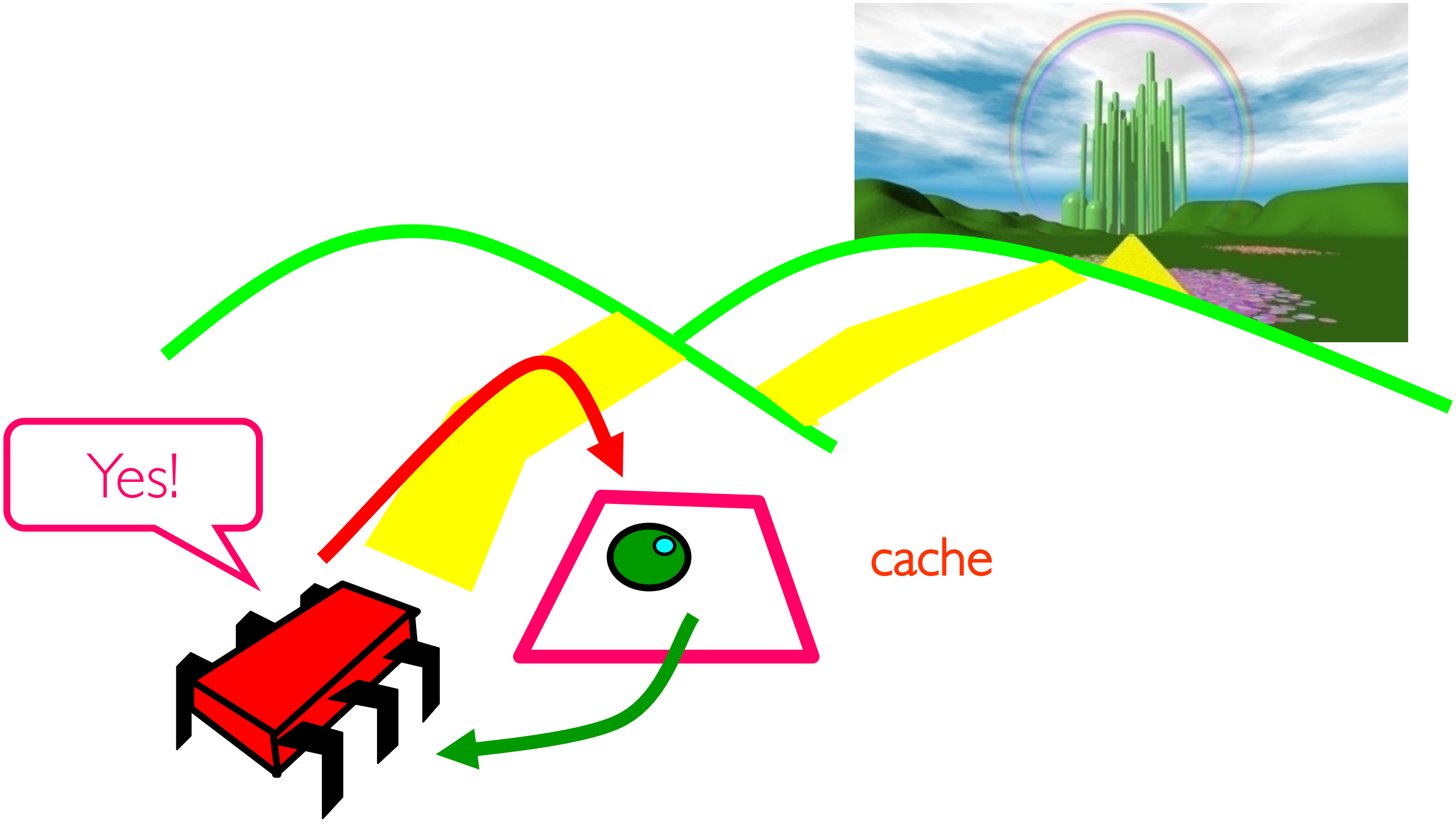
Cache: Reading from Memory



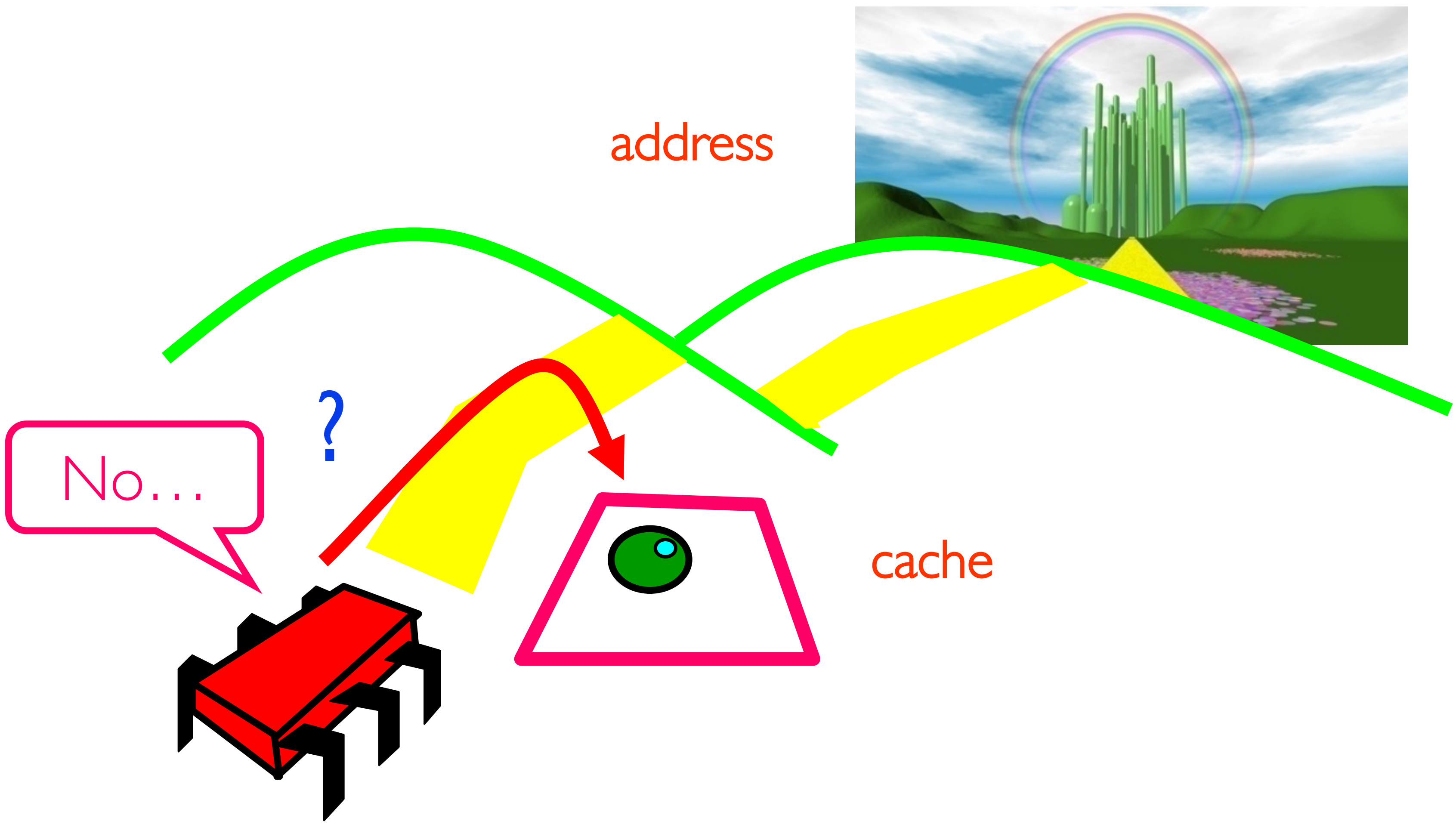
Cache Hit



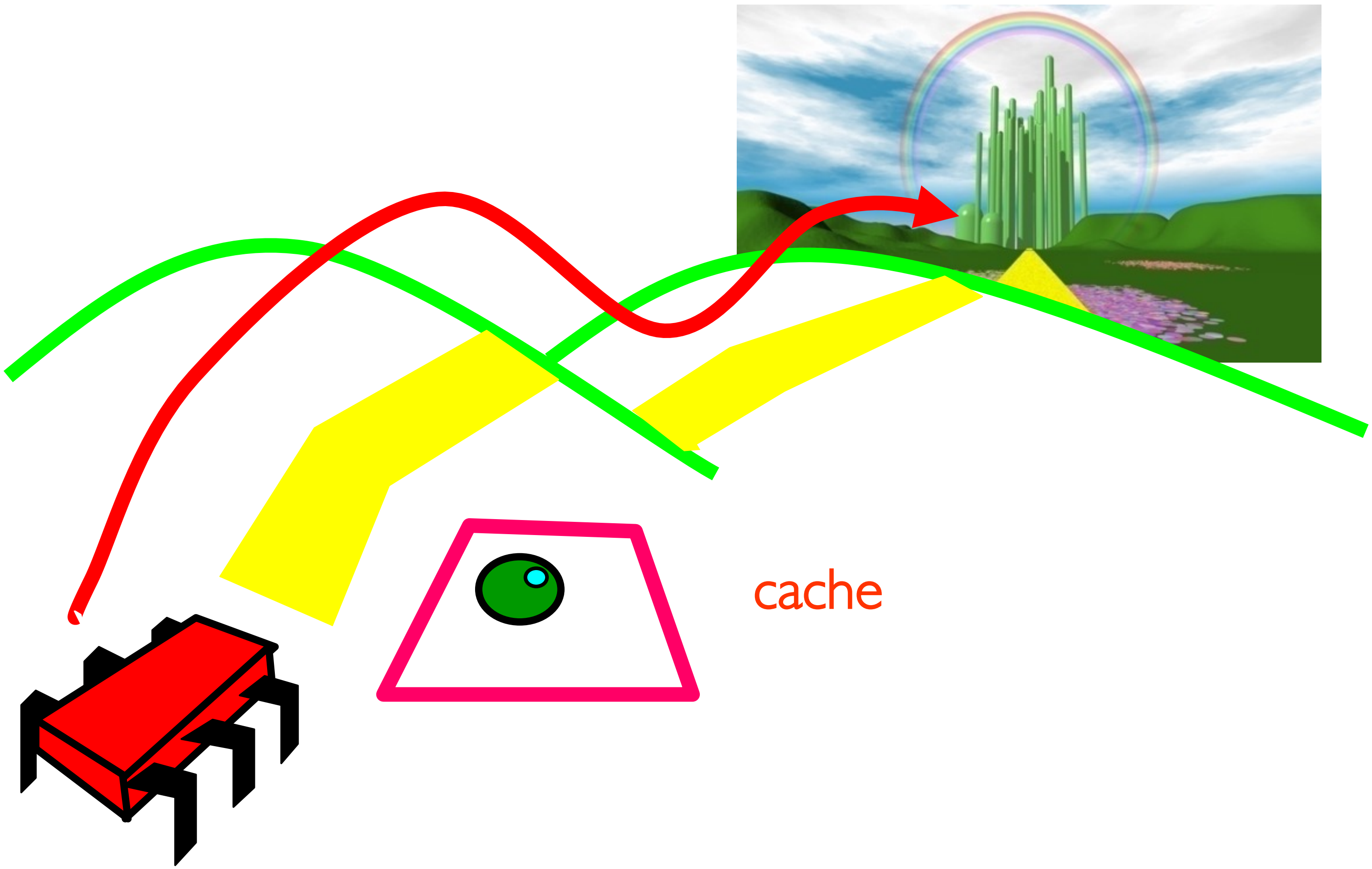
Cache Hit



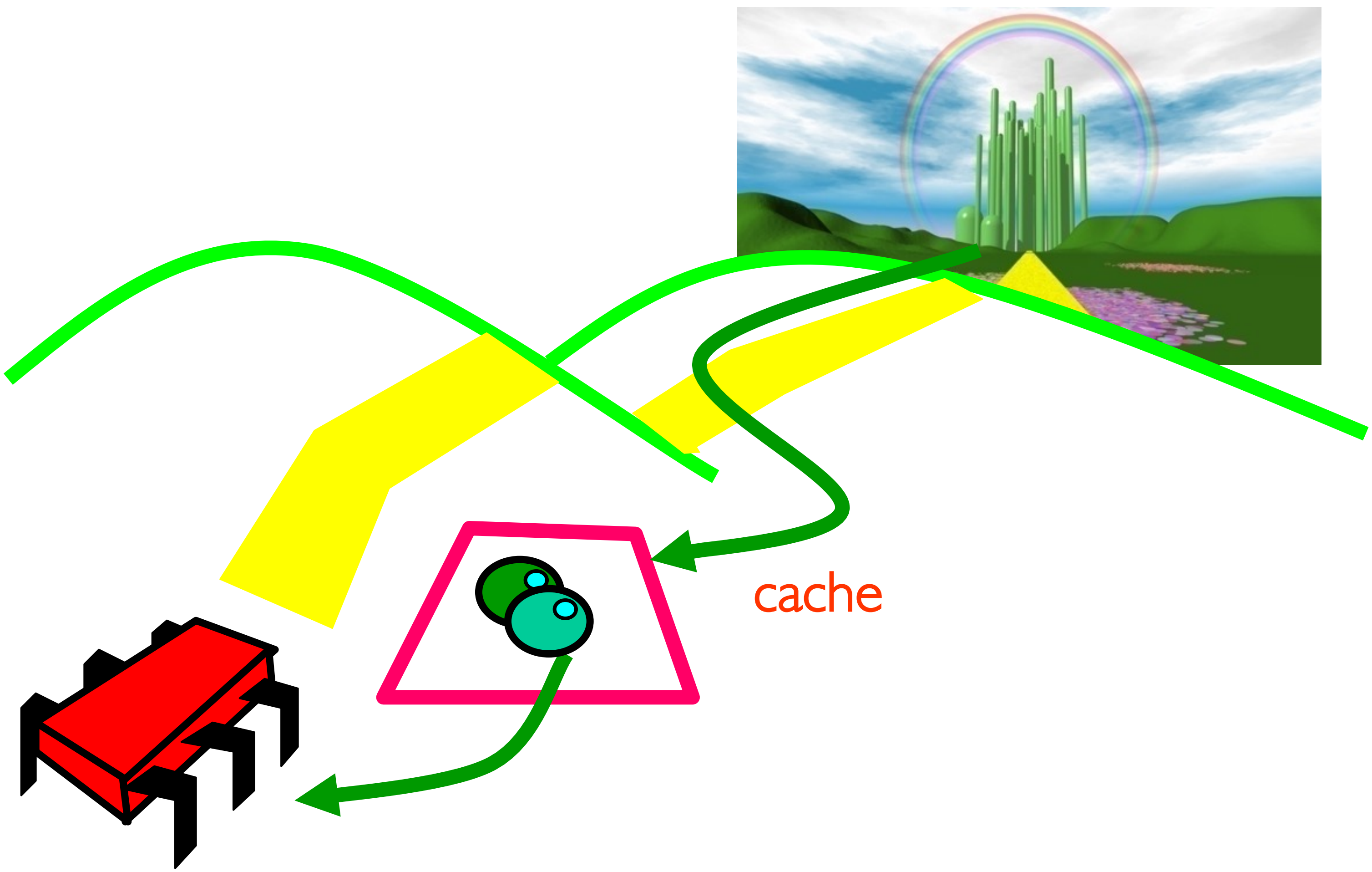
Cache Miss



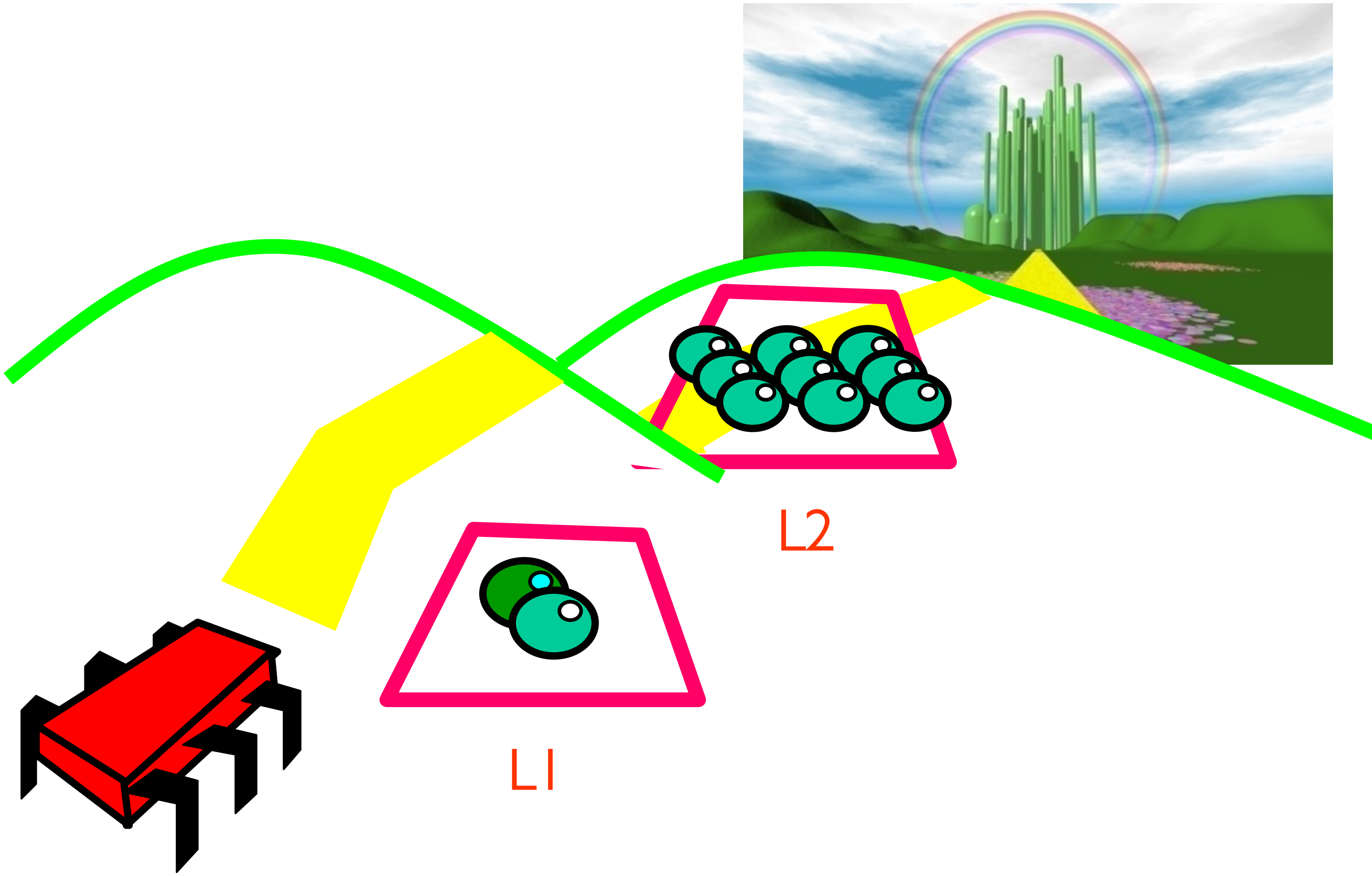
Cache Miss



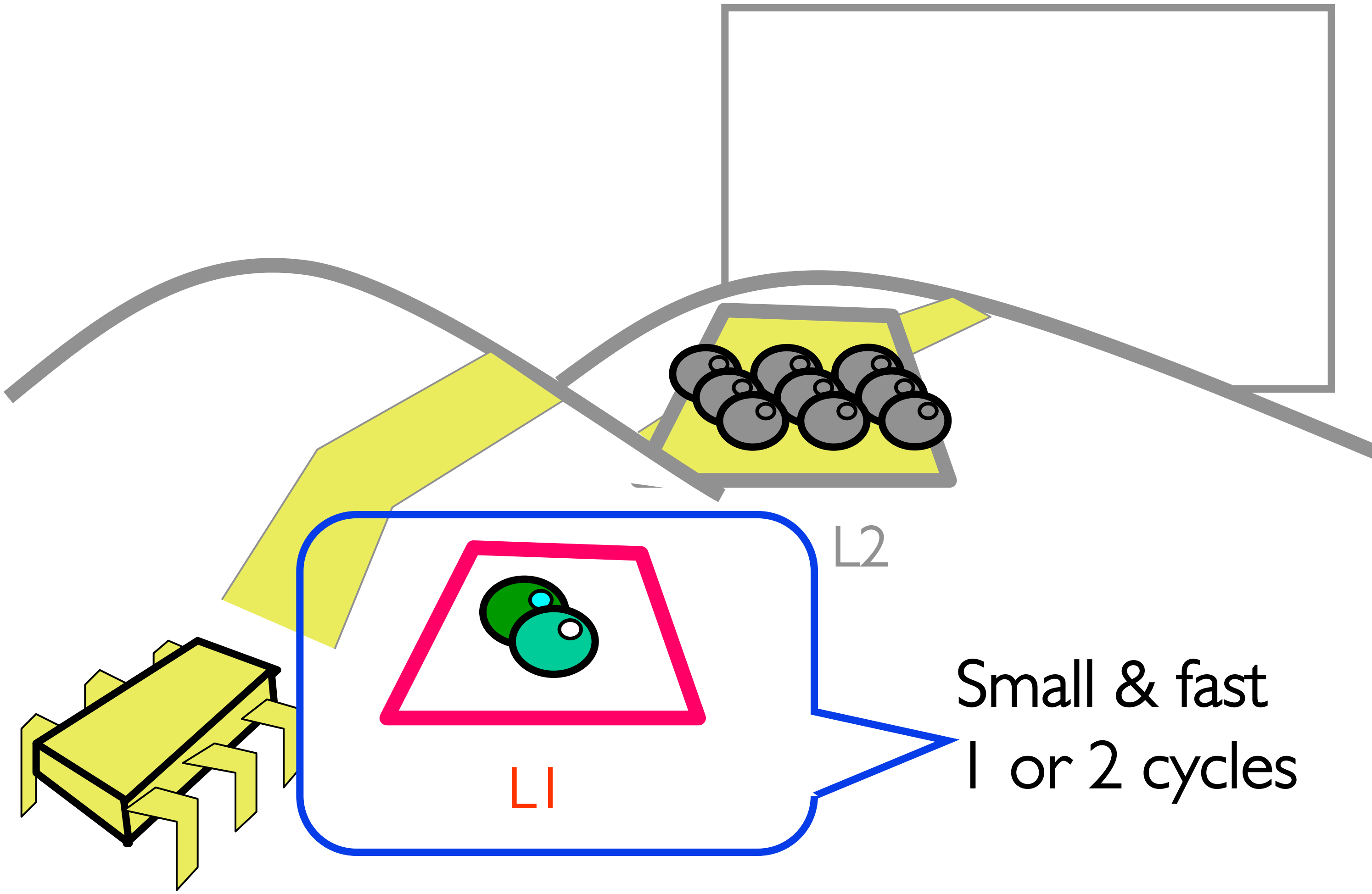
Cache Miss



L1 and L2 Caches

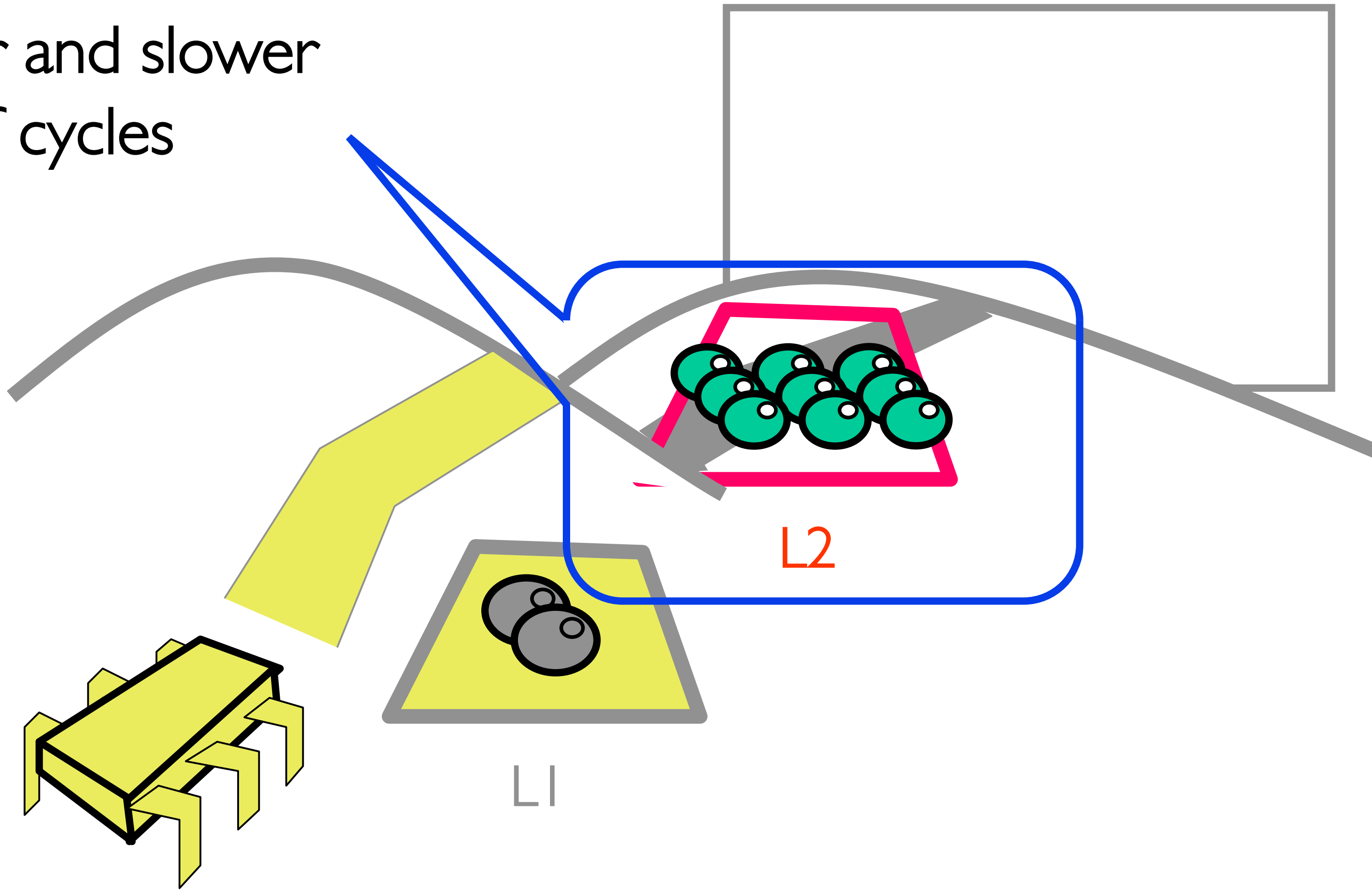


L1 and L2 Caches



L1 and L2 Caches

Larger and slower
10s of cycles

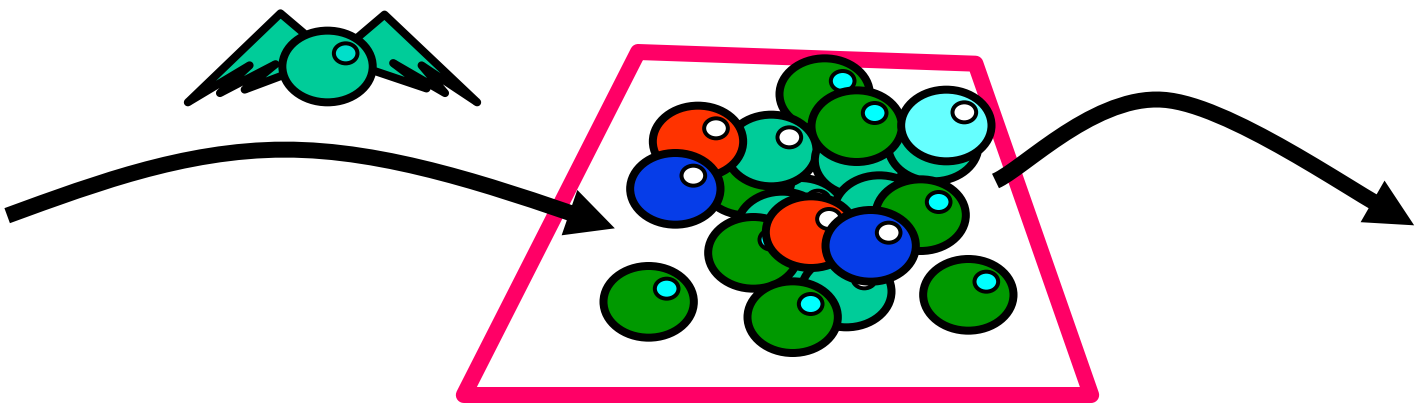


When a Cache Becomes Full…

- Need to make room for new entry
- By evicting an existing entry
- Need a replacement policy
  - Usually some kind of least recently used heuristic

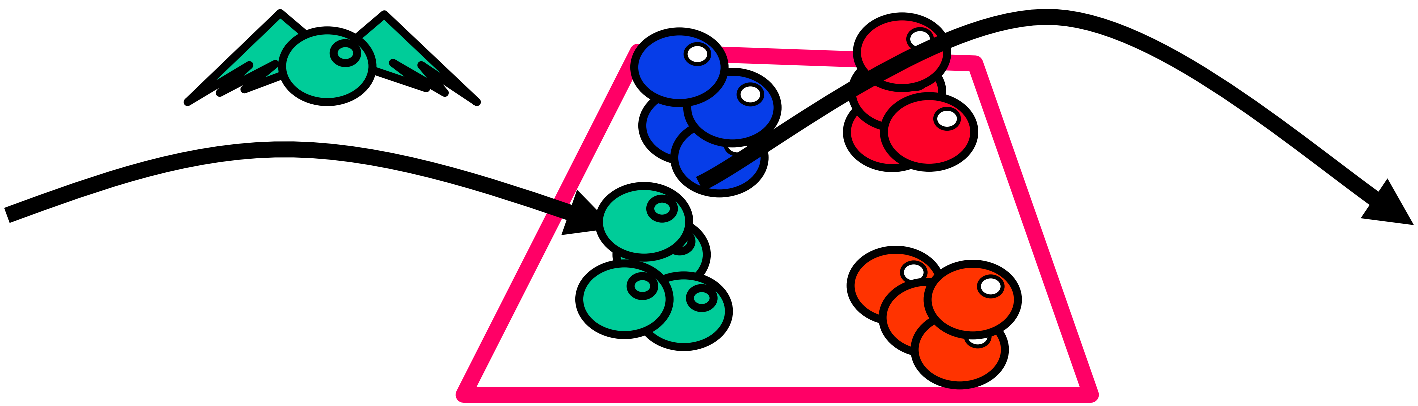
Fully Associative Cache

- Any line can be anywhere in the cache
  - Advantage: can replace any line
  - Disadvantage: hard to find lines



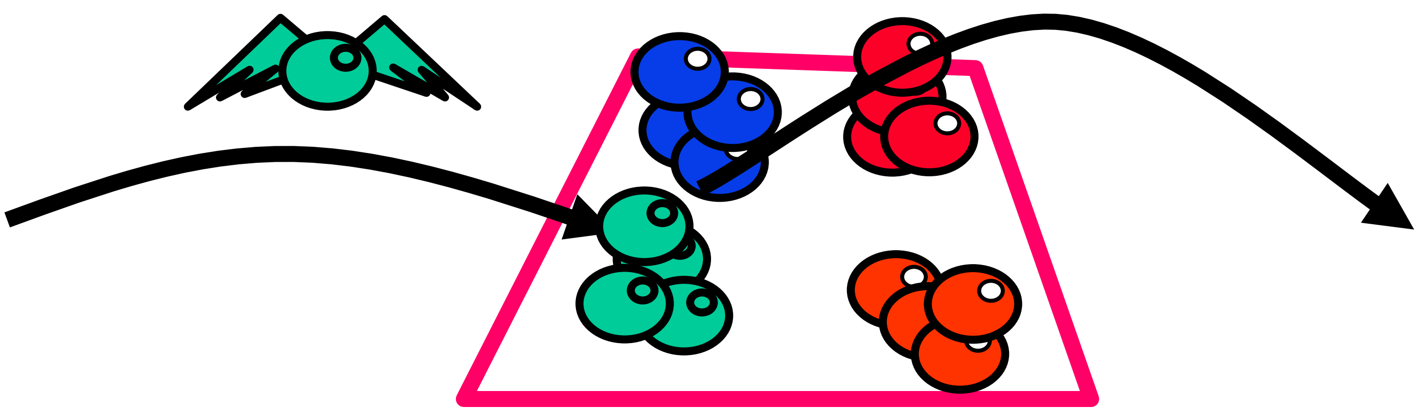
K-way Set Associative Cache

- Each slot holds k lines
  - Advantage: pretty easy to find a line
  - Advantage: some choice in replacing line



Multicore Set Associativity

- L2, lower levels can be more associative (e.g., > 16 ways)
  - Why? Because cores share sets
  - Threads cut effective size if accessing different data



Example: Average Memory latency

w/o cache, all accesses go to memory:

Average Memory Access Time = memory latency

with one level cache:

Average Memory Access Time =

cache hit time x (1 – cache miss rate) +

memory latency x cache miss rate

- With a cache has a miss rate of 10%, hit time of 2 cycles and a memory latency off 100 cycles, what is AMAT?

AMAT = (2 x 90%) + (100 x 10%) = 11.8 cycles

Example: Average Memory latency

Assume the following params:

L1 hit time = 2 cycles, L1 miss rate = 5%

L2 hit time = 10 cycles, L2 miss rate = 2%

Memory latency = 200 cycles

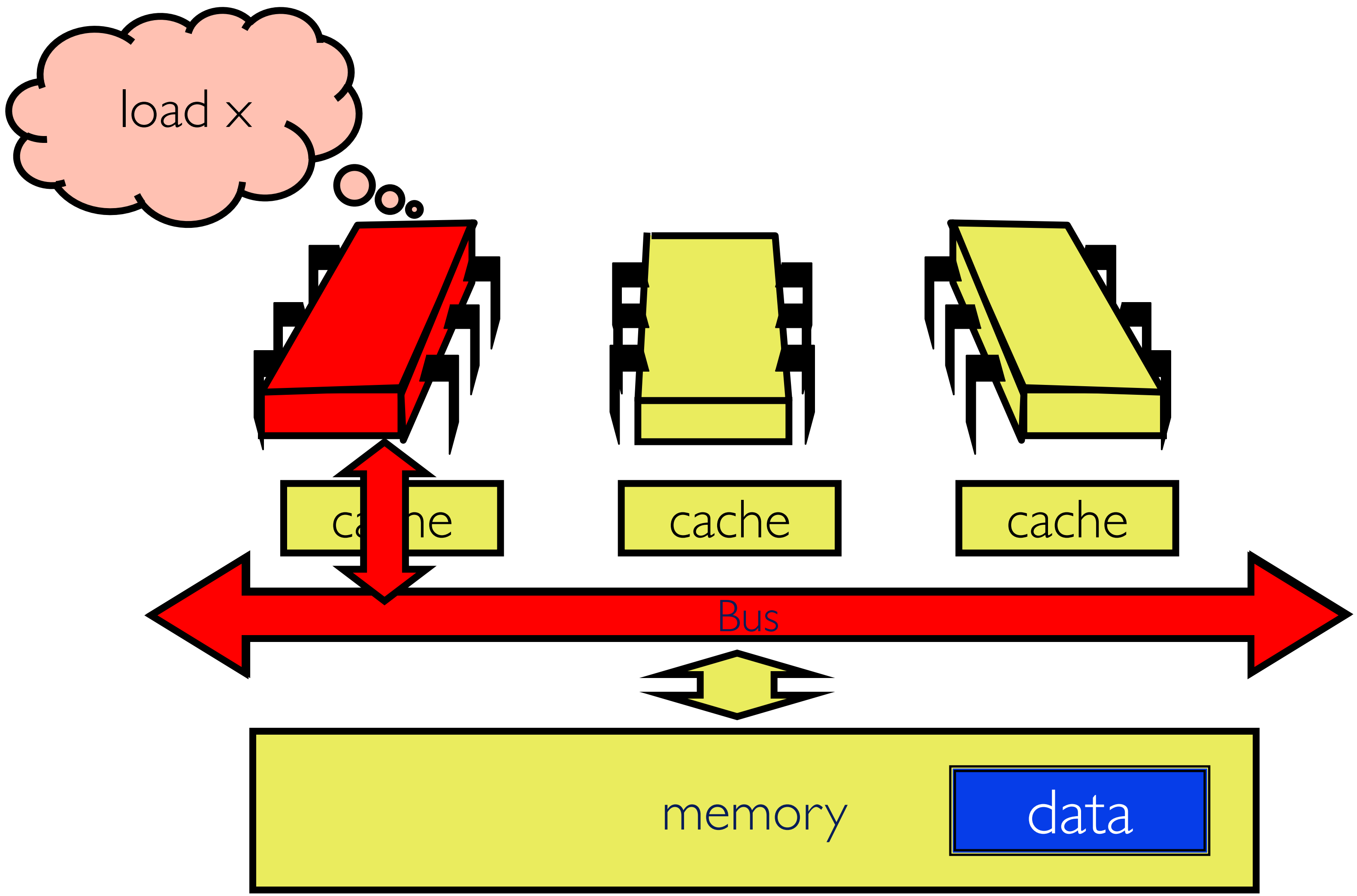
- What is AMAT?

$$AMAT = (2 \times 95\%) + (10 \times 5\%) + (200 \times 5\% \times 2\%)$$
$$= 1.9 + 0.5 + 0.2 = 2.6 \text{ cycles}$$

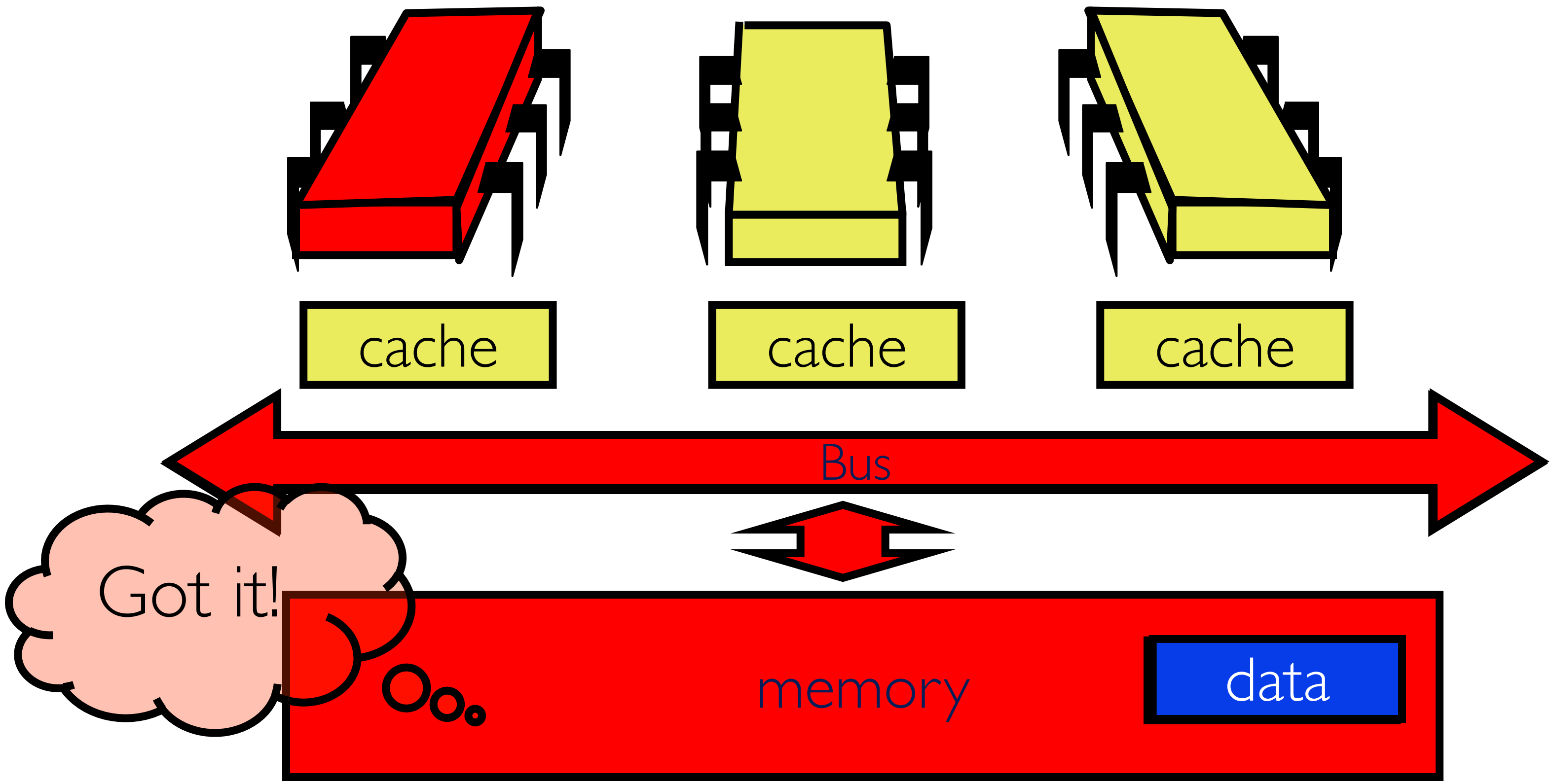
Cache Coherence

- A and B both cache address x
- A writes to x
  - Updates cache
- How does B find out?
- Many coherence protocols in products

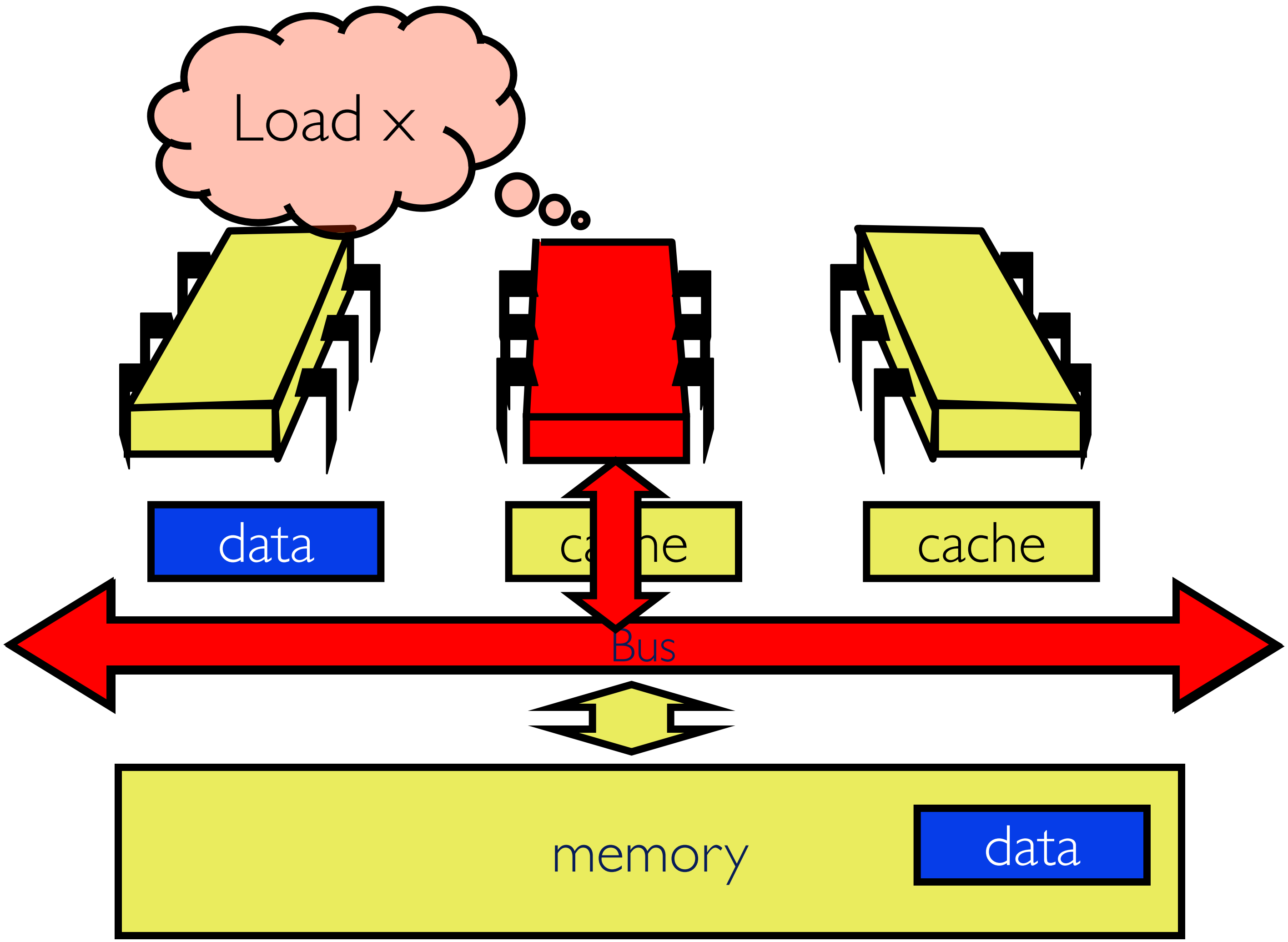
Processor Issues Load Request



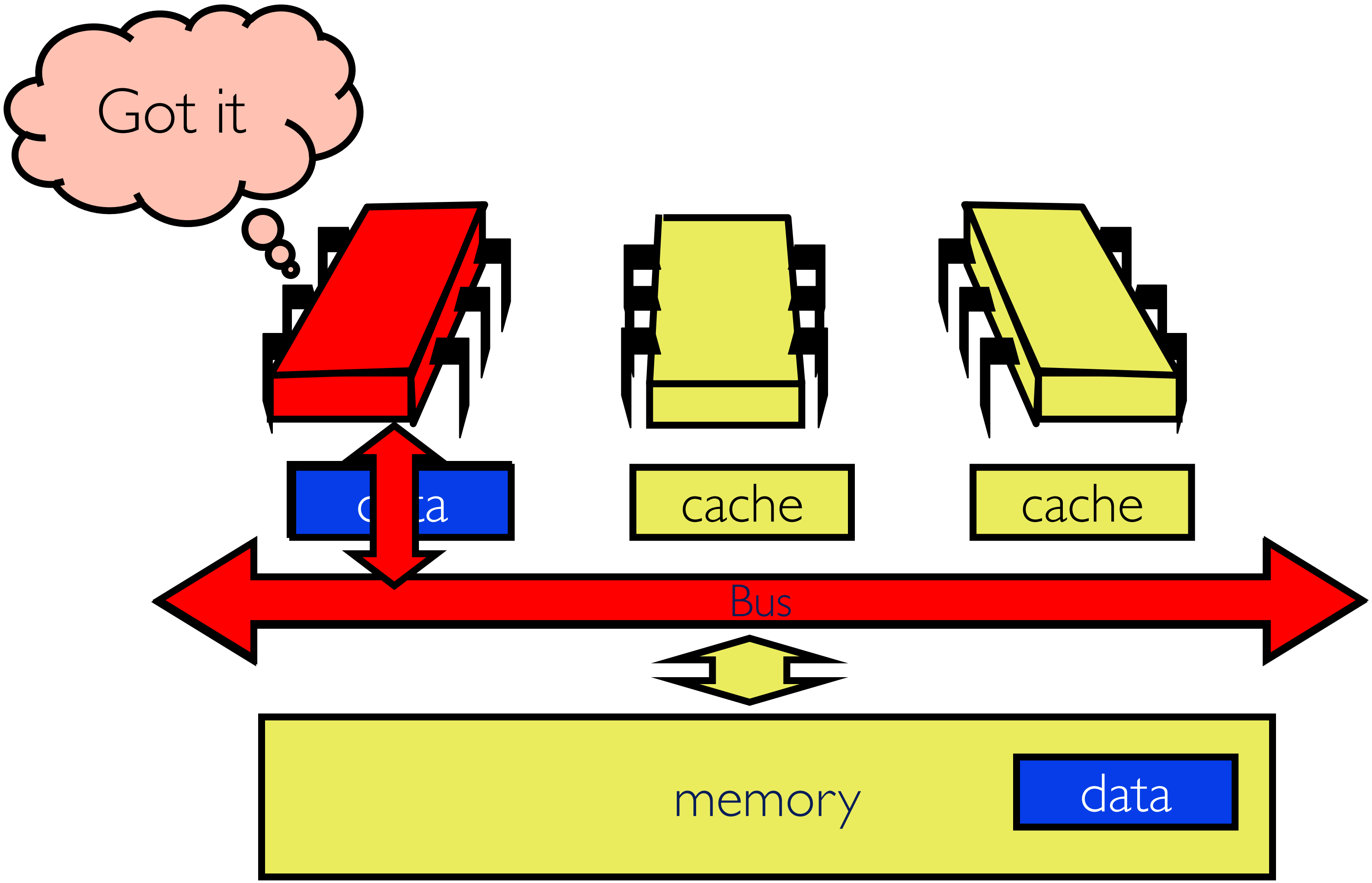
Memory Responds



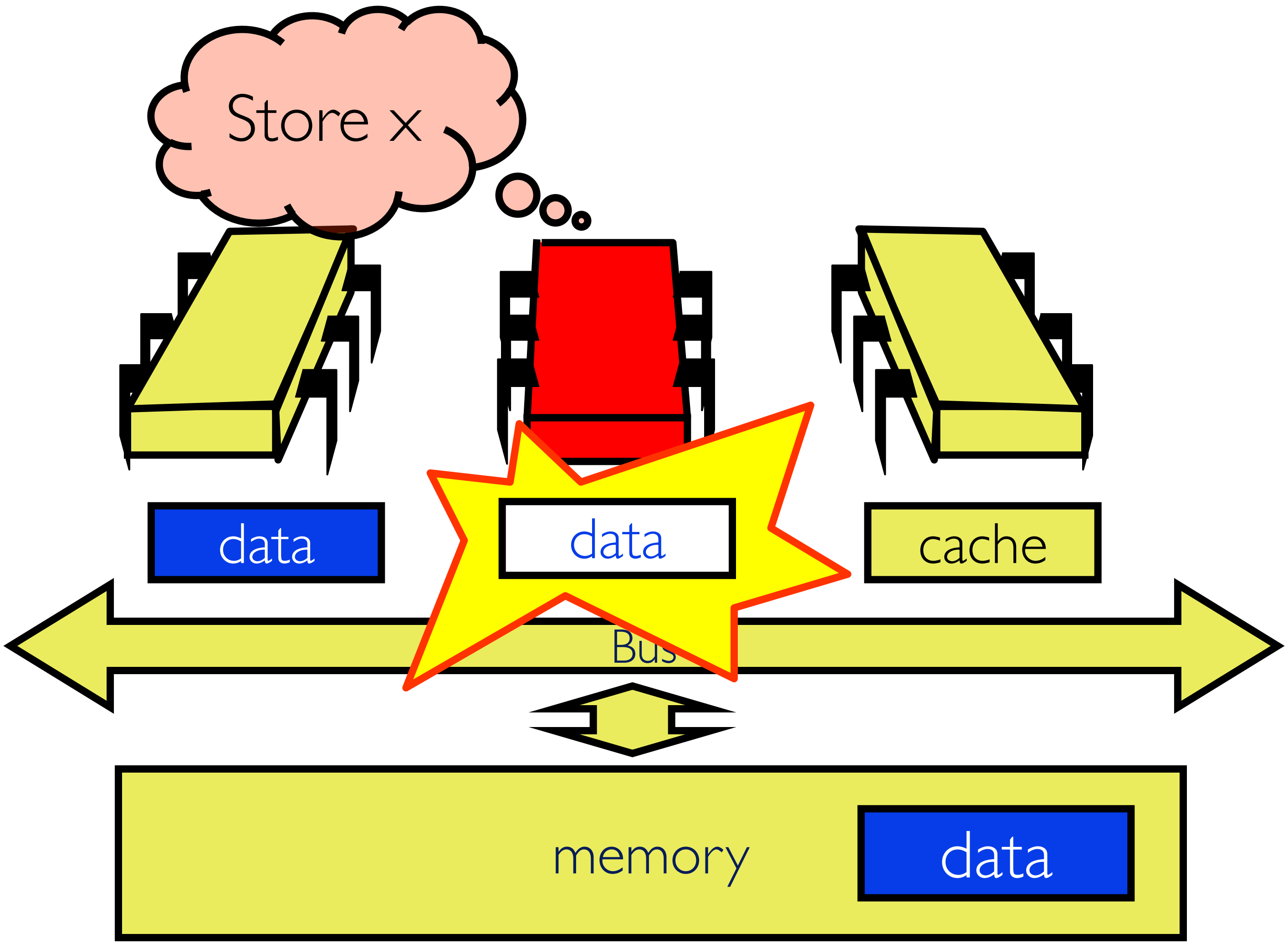
Processor Issues Load Request



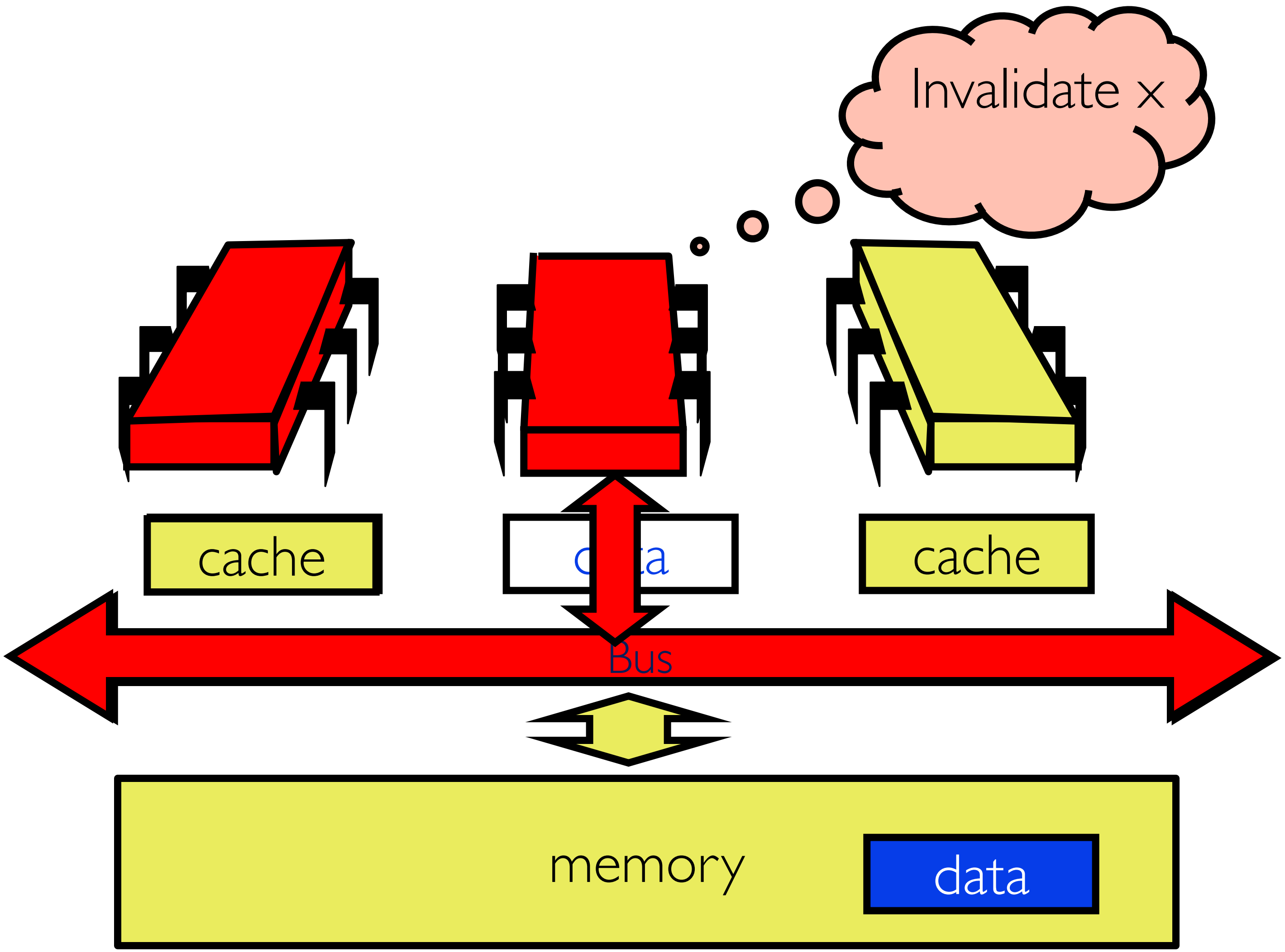
Other Processor Responds



Modify Cached Data



Invalidate



Coherence Misses

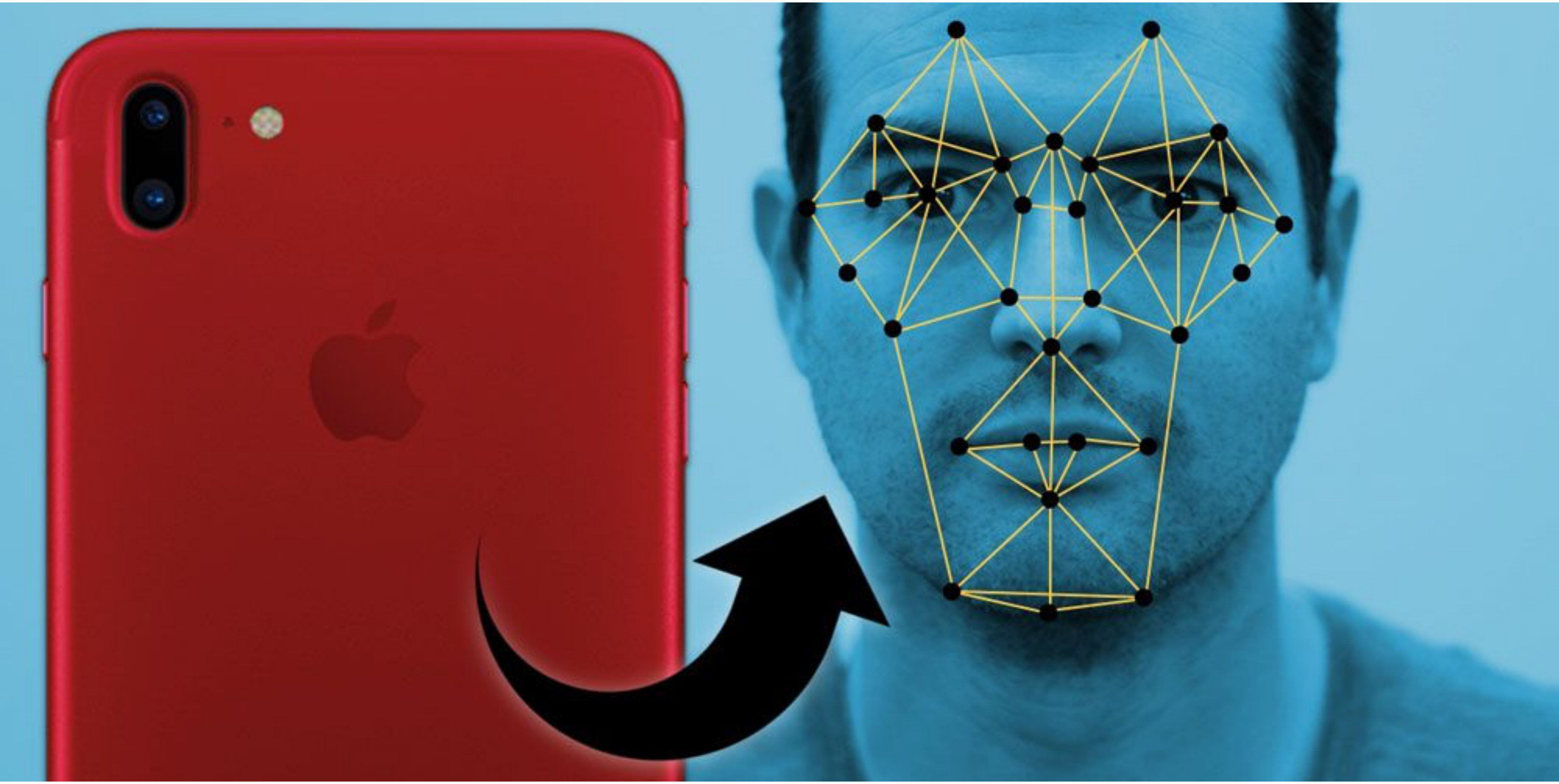
- Sometimes necessary, called "True Sharing"
  - When two processors read/write same variable "x"
  - Communicate a new value of "x" from one thread to another
  - Inherent to the computation
- Sometimes not necessary, called "False Sharing"
  - Reading and writing two distinct variables "x1" and "x2"
  - Happen to reside in the same cache block
  - E.g., x1 and x2 are single words, but a 64-byte block can hold 8 words, and contains both "x1" and "x2"
- Can restructure to reduce coherence misses

Summary

- Most multiprocessors use shared memory
- We will assume an SMP, simple multiprocessor model
- Must know how to platform works to construct software
- Must understand the bottlenecks
- Now, we will see how to think parallel

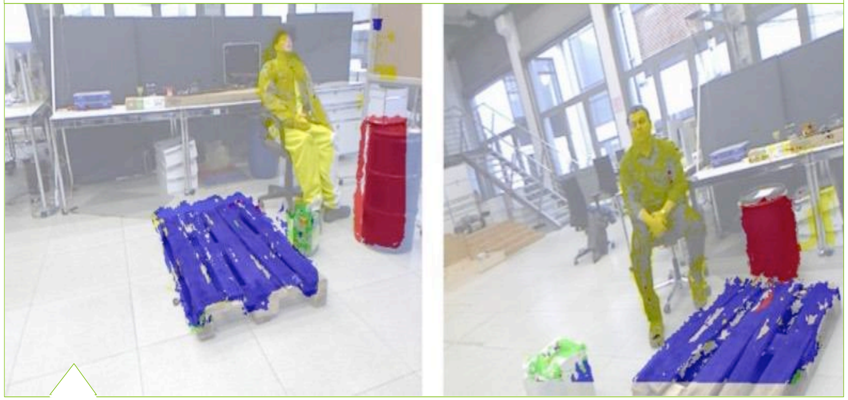
To design well-balanced parallel software we need to think about how a problem can be solved in parallel, divide the work evenly among threads, maximize the parallelism and reduce overhead

Example App: Apple X Face Unlock

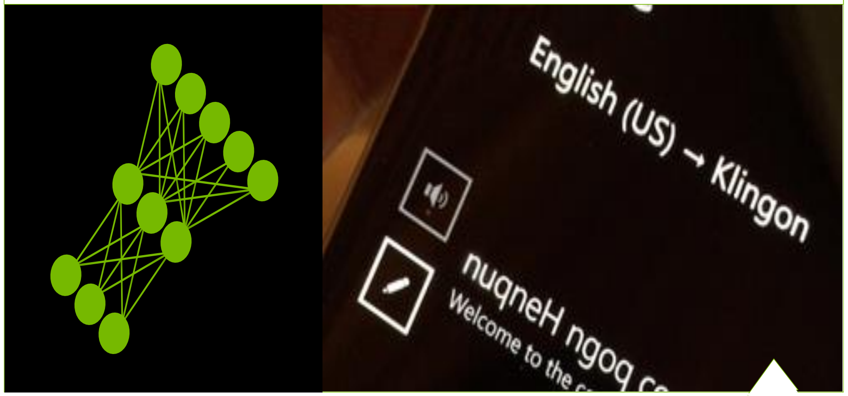


DEEP LEARNING EVERYWHERE

Image Classification, Object Detection, Localization, Action Recognition



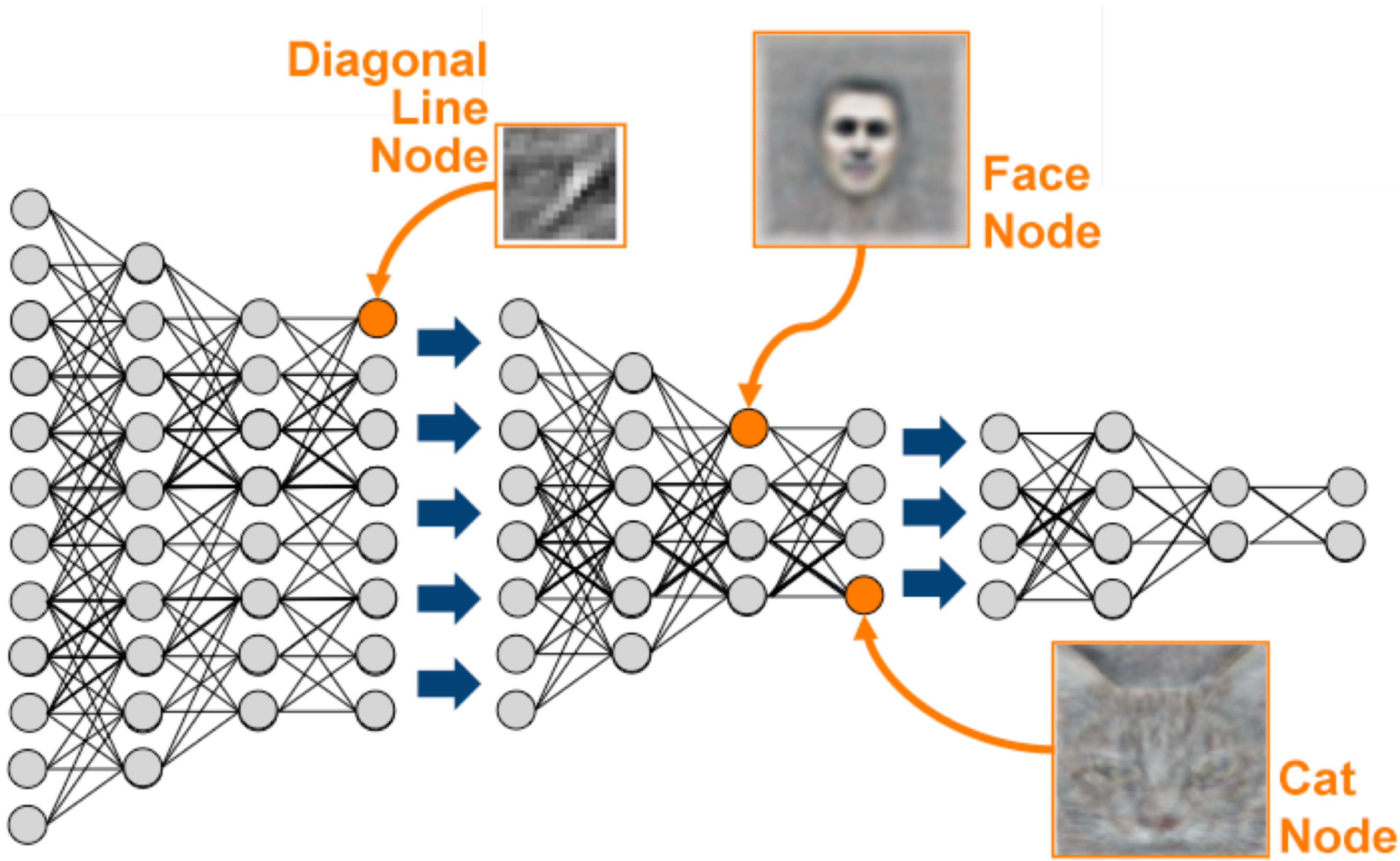
Speech Recognition, Speech Translation, Natural Language Processing



Pedestrian Detection, Lane Detection, Traffic Sign Recognition

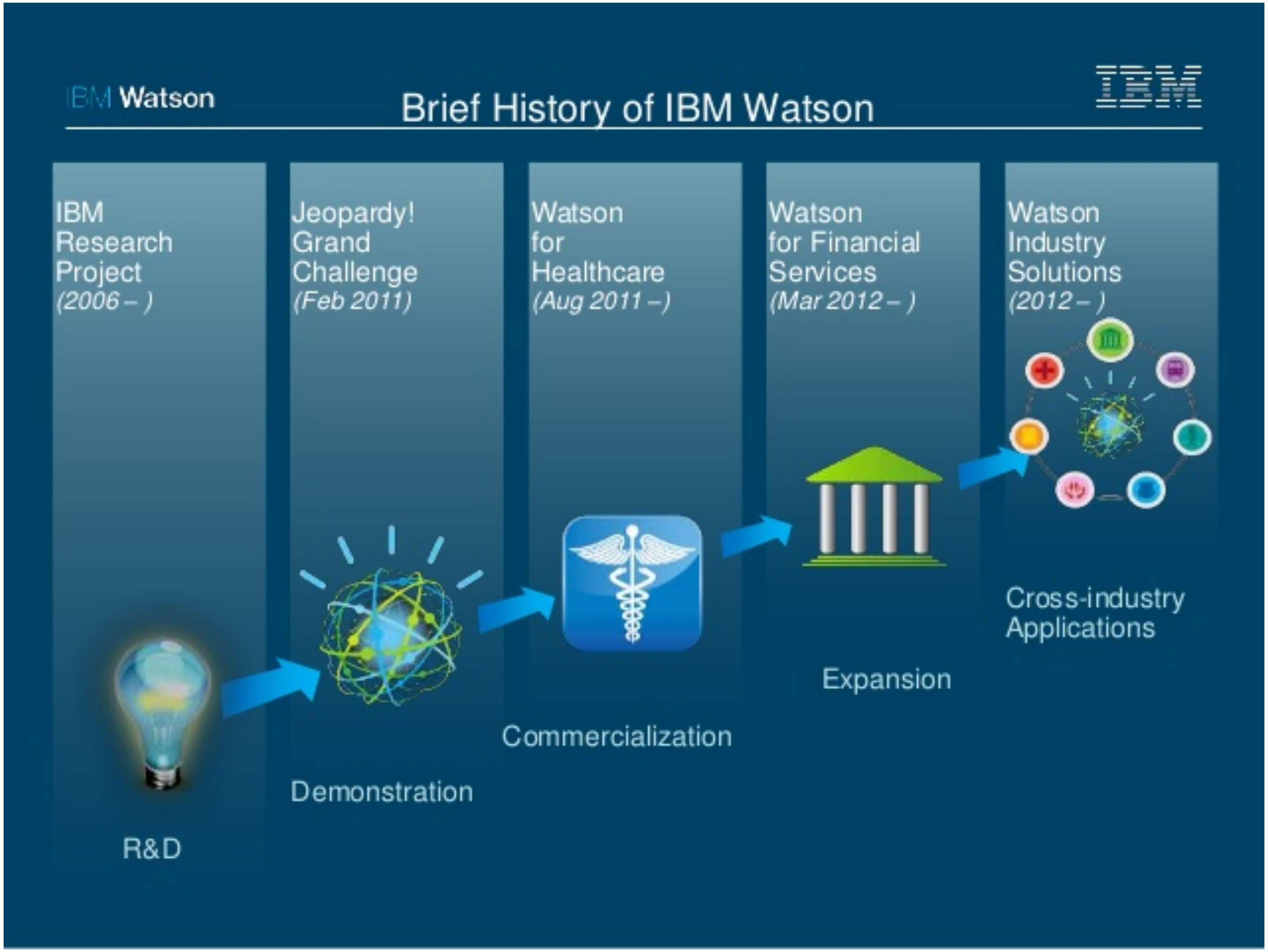
Breast Cancer Cell Mitosis Detection, Volumetric Brain Image Segmentation

Example App: Deep Learning



Used in search, machine translation, face recognition, investment banking,..

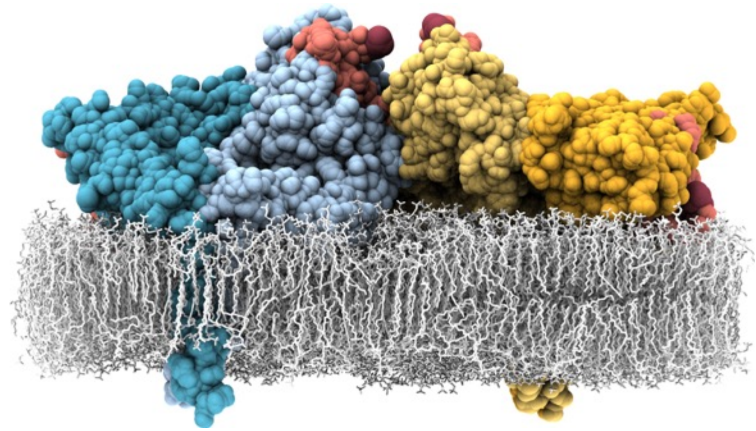
Example App: IBM Watson



Example Apps: Science and Engineering

- Examples
  - Weather prediction
  - Drug development
  - Oil reservoir simulation
  - Automobile crash tests
  - ….

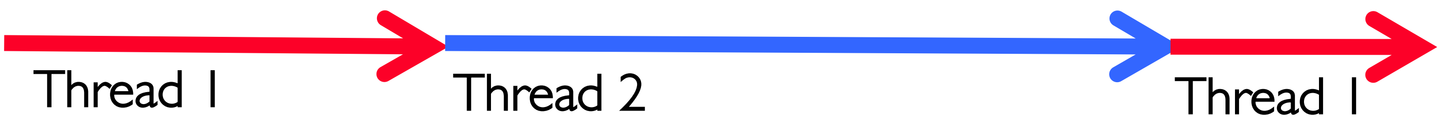
Molecular dynamics used in drug discovery



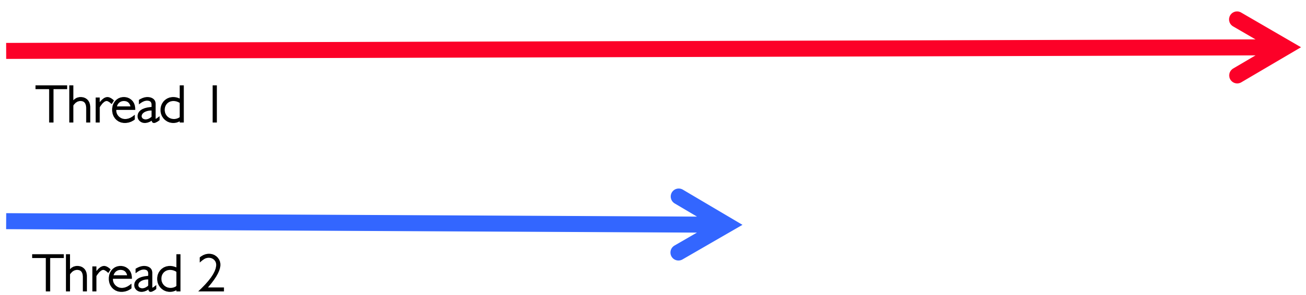
- Typically model physical systems or phenomena
- Problems are 2D or 3D
- Usually requires heavy arithmetic

Technical distinction: Concurrency vs. Parallelism

- Concurrency: two or more threads march together



- Parallelism: two or more threads execute at the same time
  - All parallel threads are concurrent, but not vice versa



- Roughly how many threads vs. how many cores

Terminology

- A Task is a piece of work
  - iPhone X Face Unlock: compute a grid point on image
- Task grain
  - small → fewer operations (less work) per task
  - large → more operations (more work) per task
- Threads performs tasks
  - Threads execute on cores

Forms of Parallelism

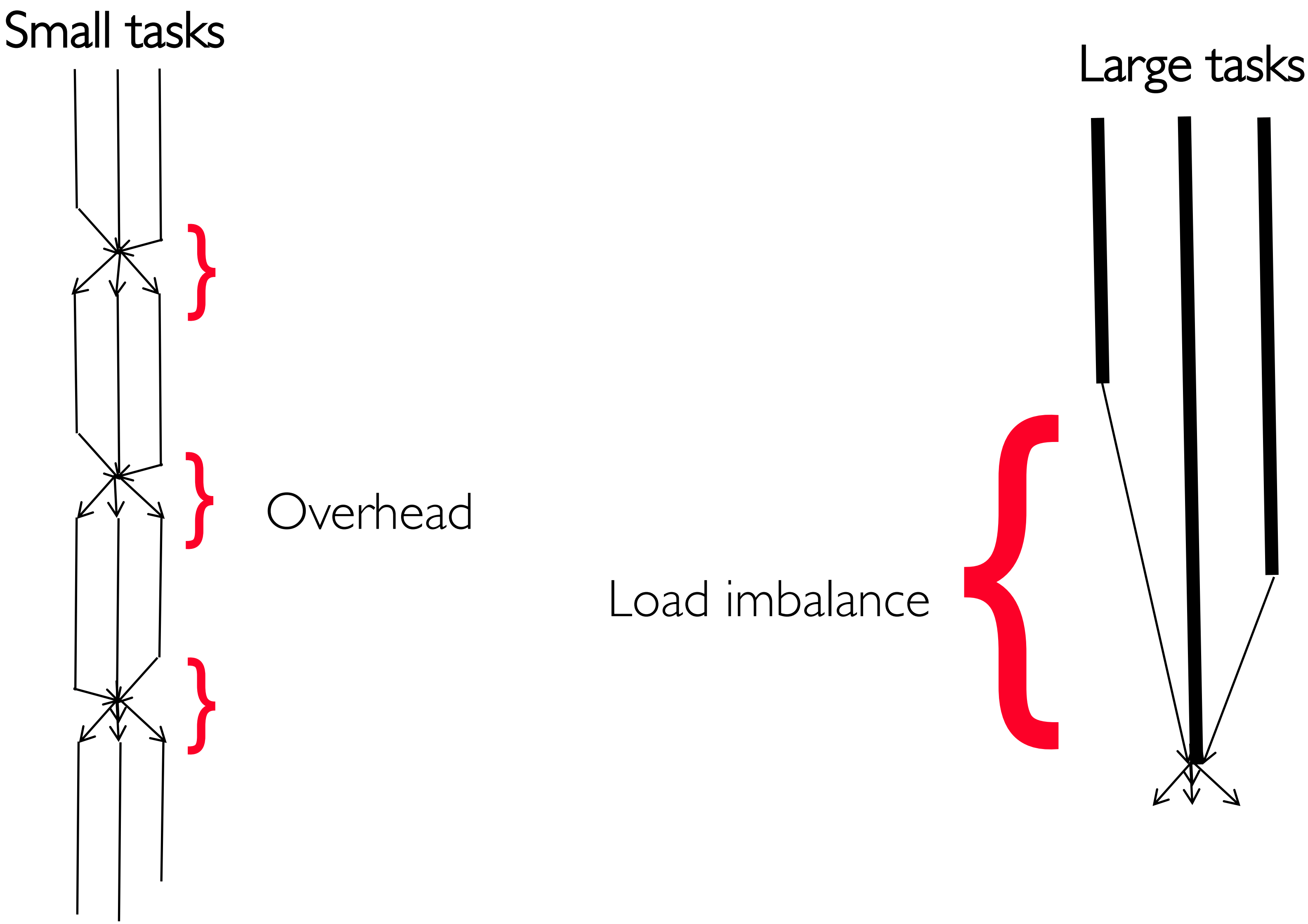
- Throughput parallelism
  - Perform many (identical) sequential tasks at the same time
  - E.g., Google search, ATM (bank) transactions
- Functional or task parallelism
  - Perform tasks that are functionally different in parallel
  - E.g., iPhoto (face recognition with slide show)
- Pipeline parallelism
  - Perform tasks that are different in a particular order
  - E.g., speech (signal, phonemes, words, conversation)
- Data parallelism
  - Perform the same task on different data
  - E.g., Data analytics, image processing

} Reduce time for one job

Division of Work: It's about Performance

- Balance workload
  - Give each parallel task the same rough amount of work
- Reduce communication
  - Balance computation time with communication time
  - Computation → useful work, Communication → overhead
- Reduce extra work
  - Creating a thread, assigning work
  - Scheduling threads on processors, OS, etc.
- These are at odds with each other

Example: Division of Work



Matrix Multiplication

$$\left(\boldsymbol{C}\right) = \left(\boldsymbol{A}\right) \bullet \left(\boldsymbol{B}\right)$$

Matrix Multiplication

$$\left(C_{n\times n}\right) = \left(A_{n\times n}\right) \times \left(B_{n\times n}\right)$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

Where

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Matrix Multiplication

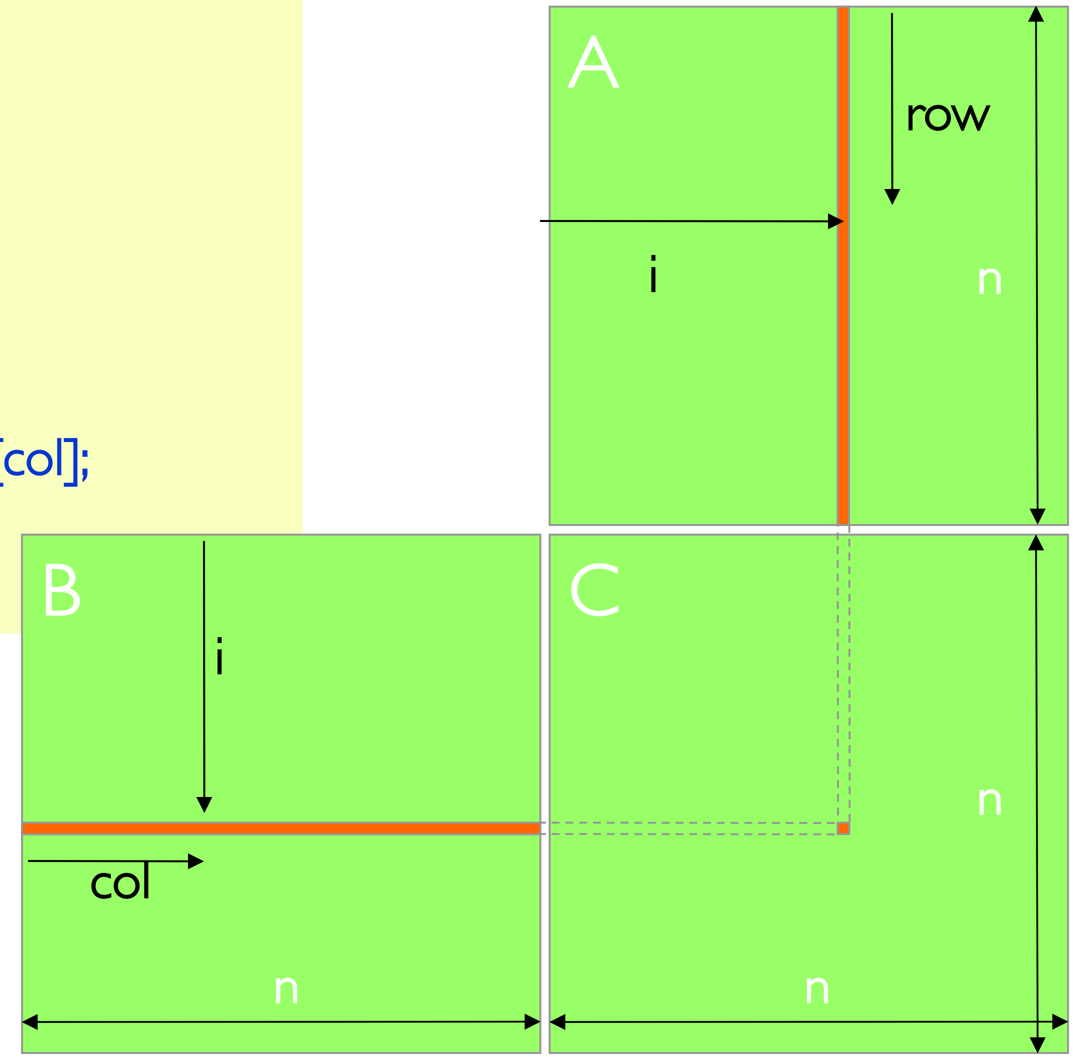
$$\left(C_{n\times n}\right)=\left(A_{n\times n}\right)\times\left(B_{n\times n}\right)$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

For each *i* and *j*:
Multiply the entries a_{ik} by the entries b_{kj} for $k = 1, 2, \ldots, n$
and summing the results over k

Matrix Multiplication in Java

```
class Worker extends Thread {
  int row, col;
  Worker(int row, int col) {
    this.row = row; this.col = col;
  }
  public void run() {
    double dotProduct = 0.0;
    for (int i = 0; i < n; i++)
      dotProduct += A[row][i] * B[i][col];
    C[row][col] = dotProduct;
  }}}
```



Matrix Multiplication

```
class Worker extends Thread {
  int row, col;
  Worker(int row, int col) {
    this.row = row; this.col = col;
  }
  public void run() {
    double dotProduct = 0.0;
    for (int i = 0; i < n; i++)
      dotProduct += A[row][i] * B[i][col];
    C[row][col] = dotProduct;
  }}}
```

a thread

Matrix Multiplication

```
class Worker extends Thread {
  int row, col;
  Worker(int row, int col) {
    this.row = row; this.col = col;
  }
  public void run() {
    double dotProduct = 0.0;
    for (int i = 0; i < n; i++)
      dotProduct += A[row][i] * B[i][col];
    C[row][col] = dotProduct;
  }}}
```

Which matrix entry to compute

Matrix Multiplication

```
class Worker extends Thread {
  int row, col;
  Worker(int row, int col) {
    this.row = row; this.col = col;
  }
  public void run() {
    double dotProduct = 0.0;
    for (int i = 0; i < n; i++)
      dotProduct += A[row][i] * B[i][col];
    C[row][col] = dotProduct;
}}}
```

Actual computation

Matrix Multiplication

```
void multiply() {
  Worker[][] worker = new Worker[n][n];
  for (int row …)
    for (int col …)
      worker[row][col] = new Worker(row,col);
  for (int row …)
    for (int col …)
      worker[row][col].start();
  for (int row …)
    for (int col …)
      worker[row][col].join();
}
```

Matrix Multiplication

```
void multiply() {
 Worker[][] worker = new Worker[n][n];
  for (int row …)
   for (int col …)
    worker[row][col] = new Worker(row,col);
  for (int row …)
   for (int col …)
    worker[row][col].start();
  for (int row …)
   for (int col …)
    worker[row][col].join();
}
```

Create n x n threads

Matrix Multiplication

```
void multiply() {
 Worker[][] worker = new Worker[n][n];
  for (int row …)
    for (int col …)
      worker[row][col] = new Worker(row,col);
  for (int row …)
    for (int col …)
      worker[row][col].start();
  for (int row …)
    for (int col …)
      worker[row][col].join();
}
```

Start them

Matrix Multiplication

```
void multiply() {
  Worker[][] worker = new Worker[n][n];
  for (int row …)
    for (int col …)
      worker[row][col] = new Worker(row,col);
  for (int row …)
    for (int col …)
      worker[row][col].start();
  for (int row …)
    for (int col …)
      worker[row][col].join();
}
```

Start them

Wait for them to finish

Matrix Multiplication

```
void multiply() {
 Worker[][] worker = new Worker[n][n];
  for (int row …)
   for (int col …)
    worker[row][col] = new Worker(row,col);
  for (int row …)
   for (int col …)
    worker[row][col].start();
  for (int row …)
   for (int col …)
    worker[row][col].join();
}
```

Start them

Wait for them to finish

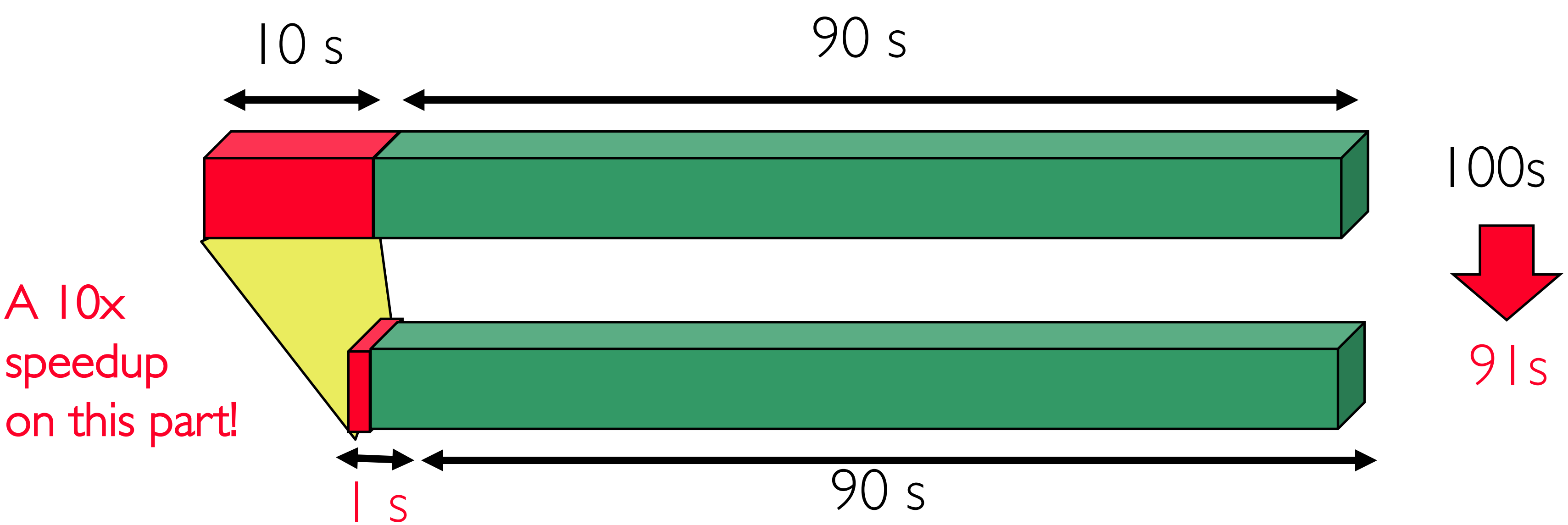
What's wrong with this picture?

Thread Overhead

- One thread per task
  - One dot product task
- Threads Require resources
  - State:
    - Memory for stacks
    - A copy of the register file
    - Program state: Program Counter, Stack pointer,….
  - Setup, teardown
  - Scheduler overhead
- Short-lived threads
  - Bad ratio of work versus overhead

One More "Big" Performance-Related Axiom

- Amdahl's Law
  - In English: if you speed up only a small fraction of the execution time of a program or a computation, the speedup you achieve on the whole application is limited
- Example



Amdahl's Law

Parallel fraction

$$\text{Speedup} = \frac{1}{\frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} + (1 - \text{Fraction}_{\text{enhanced}})}$$

Amdahl's Law

Sequential fraction

$$\text{Speedup} = \frac{1}{\frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} + (1 - \text{Fraction}_{\text{enhanced}})}$$

Amdahl's Law

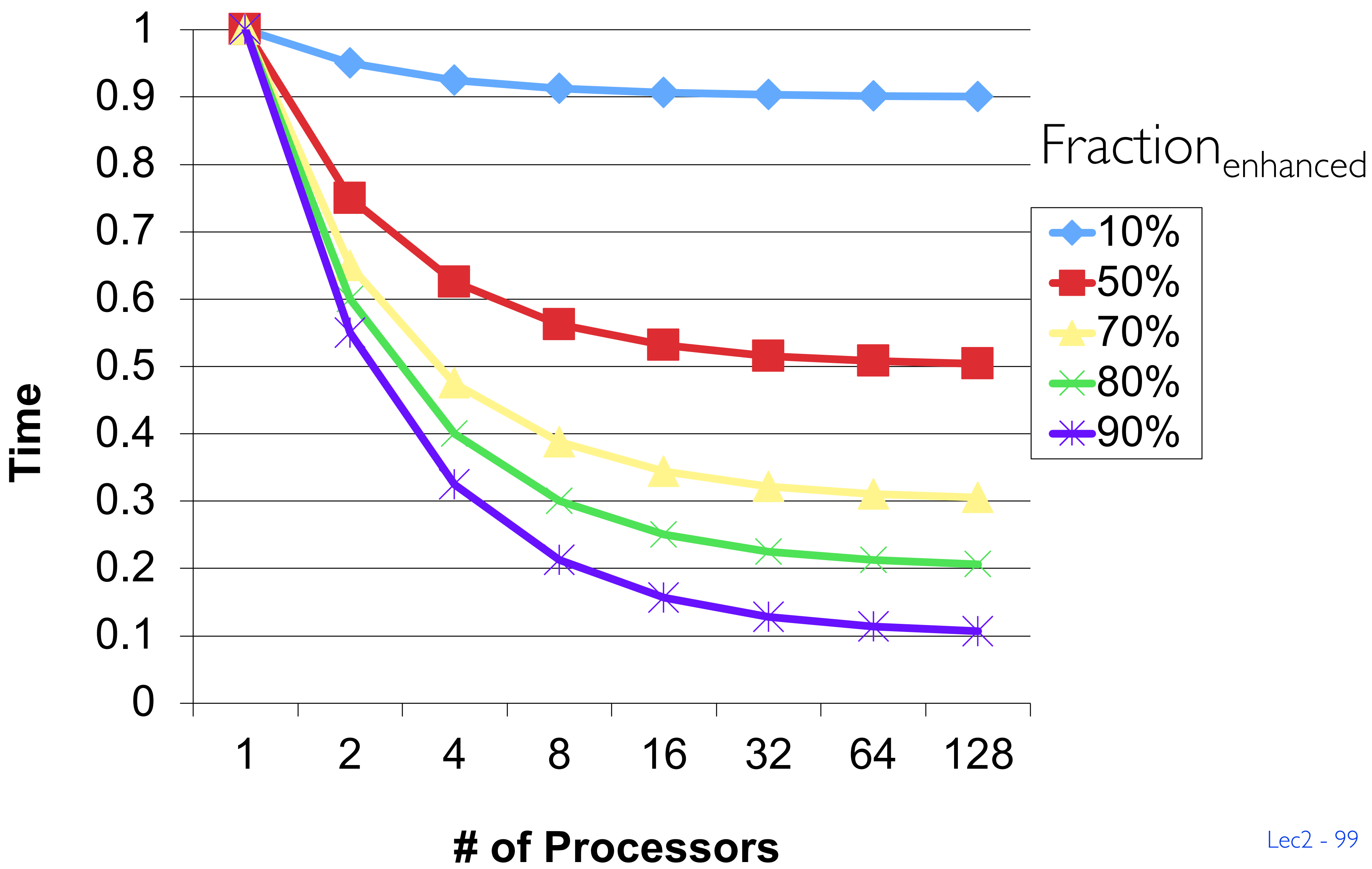
$$\text{Speedup} = \frac{1}{\frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} + (1 - \text{Fraction}_{\text{enhanced}})}$$

Simple example:

- Program runs for 100 seconds on a uniprocessor
- 10% of the program can be parallelized on a multiprocessor
- Assume an ideal multiprocessor with 10 processors

$$\text{Speedup} = \frac{1}{\frac{0.1}{10} + (1-0.1)} = \frac{1}{0.01 + 0.9} = \frac{1}{0.91} = 1.1$$

Implications of Amdahl's Law



Parallel execution is not ideal

- 10 processors rarely get a speedup of 10
  - Load imbalance
  - Thread start/join overhead
  - Communication overhead
- Even if $\text{Fraction}_{\text{enhanced}}$ is close to 100%
  - $\text{Speedup}_{\text{enhanced}} << p$ for p processors
  - Our goal is to get it as close as possible to p

Amdahl's Law (in practice)

Back to Matrix Multiplication

Sequential

$(1 - \text{Fraction}_{\text{enhanced}})$

```
void multiply() {
  Worker[][] worker = new Worker[n][n];
  for (int row …)
    for (int col …)
      worker[row][col] = new Worker(row,col);
  for (int row …)
    for (int col …)
      worker[row][col].start();
  for (int row …)
    for (int col …)
      worker[row][col].join();
}
```

Matrix Multiplication

```
class Worker extends Thread {
  int row, col;
  Worker(int row, int col) {
    this.row = row; this.col = col;
  }
  public void run() {
    double dotProduct = 0.0;
    for (int i = 0; i < n; i++)
      dotProduct += A[row][i] * B[i][col];
    C[row][col] = dotProduct;
}}}
```

Parallel ($Fraction_{enhanced}$)

Example: n = 16

- How many threads will our Matrix Multiplication create?

16 x 16 = 256 threads

- How many of these threads are concurrent (i.e., degree of concurrency)?

All threads

Example: Assume there are 4 processors

- How many threads per processor?

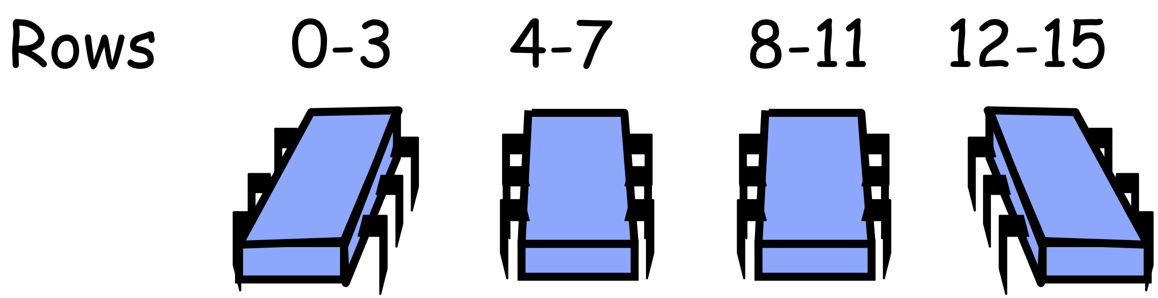
256 threads / 4 = 64 threads

- How many threads run in parallel (i.e., degree of parallelism)?

One thread per core = 4 threads

Best efficiency: Concurrency ~ Parallelism

- All work is independent
- Max parallelism? 4
- Only need 4 threads
  - Reduce thread.start(), thread.join() overhead
  - Only 4 start() and 4 join()
  - Workers (each thread) should do more work
  - 16x16 dot products divided by 4 = 64 dot products per thread



Matrix Multiplication on “p” cores

```
void multiply() {
  BigWorker[] worker = new BigWorker[n];

  for (int row=0; row < n; row+=n/p)
       worker[row] = new BigWorker(row);

  for (int row=0; row < n; row+=n/p)
        worker[row].start();

  for (int row=0; row < n; row+=n/p)
        worker[row].join();
}
```

Matrix Multiplication: (n/c) x n per worker

```
void multiply() {
  BigWorker[] worker = new BigWorker[n];

  for (int row=0; row < n; row+=n/p)
      worker[row] = new Worker(row);

  for (int row=0; row < n; row+=n/p)
      worker[row].start();

  for (int row=0; row < n; row+=n/p)
      worker[row].join();
}
```

Create p threads

BigWorker: Each thread does (n/p) x n

```
class BigWorker extends Thread {
  int begin_row;
  BigWorker(int begin_row) {
    this.begin_row = begin_row;
  }
  public void run() {
    double dotProduct = 0.0;
    for (int row=begin_row; row < begin_row+n/p; row++)
          for (int col=0; col < n; col++)
              for (int i = 0; i < n; i++)
                  dotProduct += A[row][i] * B[i][col];
              C[row][col] = dotProduct;
}}}
```

BigWorker: Each thread does (n/p) x n

```
class Worker extends Thread {
  int begin_row;
  BigWorker(int begin_row) {
    this.begin_row = begin_row;
  }
  public void run() {
    double dotProduct = 0.0;
    for (int row=begin_row; row < begin_row+n/p; row++)
        for (int col=0; col < n; col++)
            for (int i = 0; i < n; i++)
                dotProduct += A[row][i] * B[i][col];
                C[row][col] = dotProduct;
}}}
```

Multiple rows
All columns

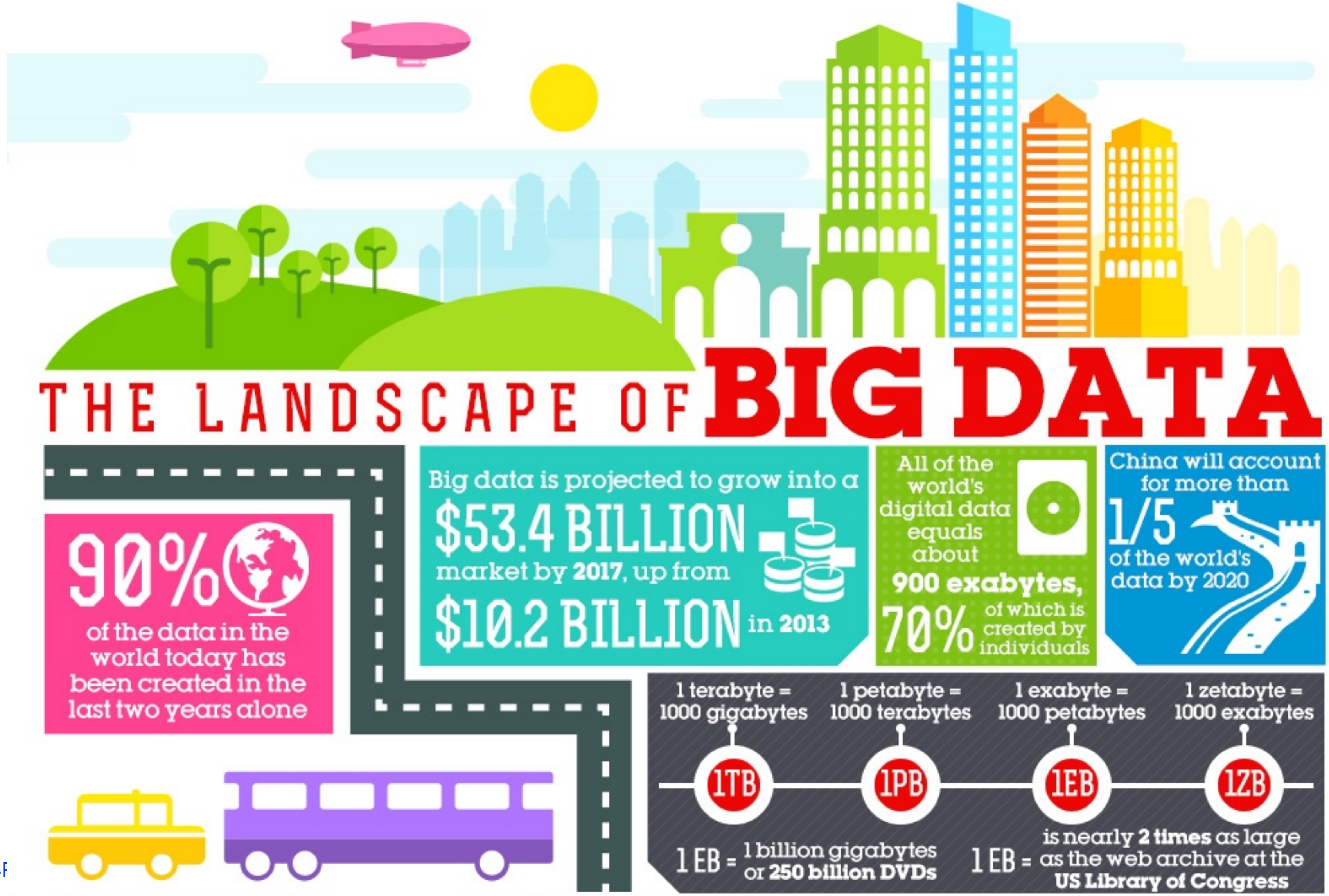
Summary

- Need to think parallel
  - Division of work
  - Lots of bottlenecks
- Don't forget Amdahl's Law

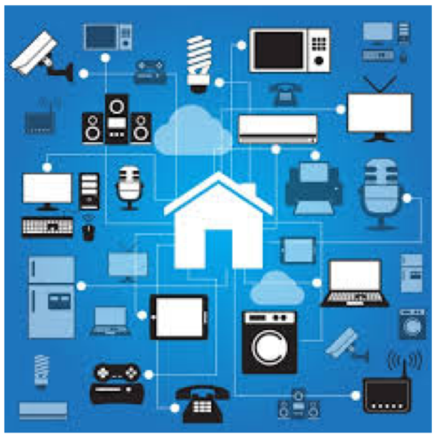
Roadmap

- Technology
  - Moore's Law
  - Parallelism
- Datacenters & Centralization
  - Economies of scale
  - Metrics
- Service-Oriented Computing
  - Cloud
  - Virtualization

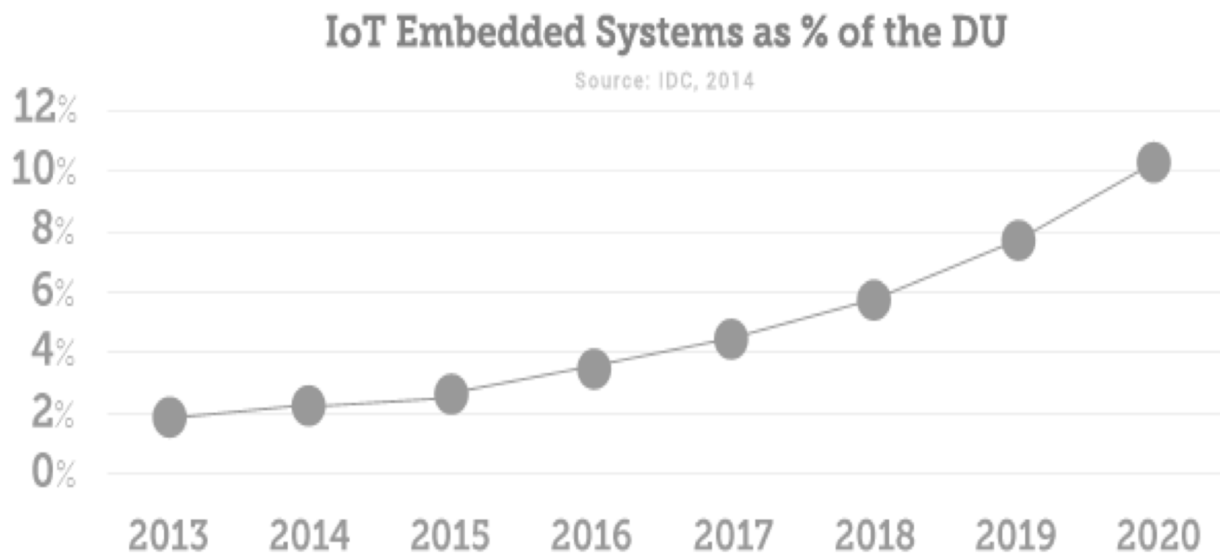
Data Economics



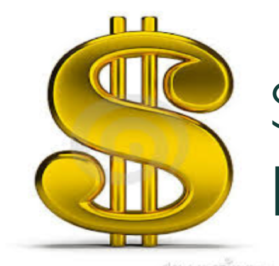
Internet-of-Things (IoT): Data in Flight



20 Billion Connected Devices



4 Zettabytes of Data, 10% of Digital Universe



$7 Trillion Market Revenue

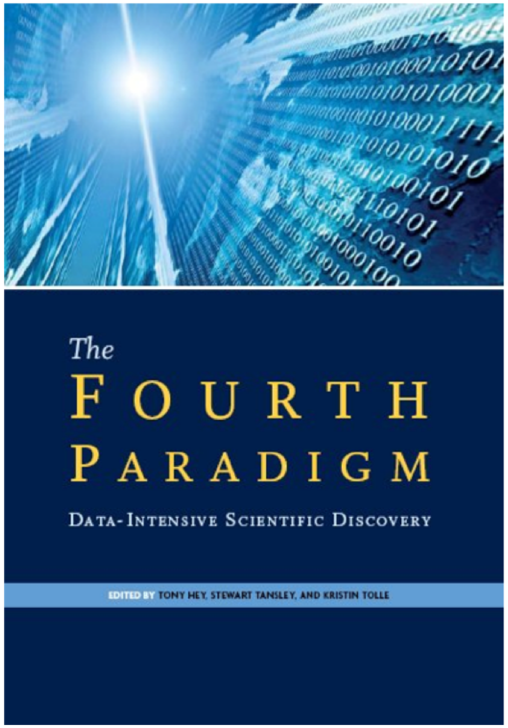
Source: IDC Worldwide and Regional IoT forecast, EMC Digital Universe with Research and Analysis by IDC

Data Shaping All Science & Technology

Science entering 4th paradigm

- Analytics using IT on
  - Instrument data
  - Simulation data
  - Sensor data
  - Human data
  - …

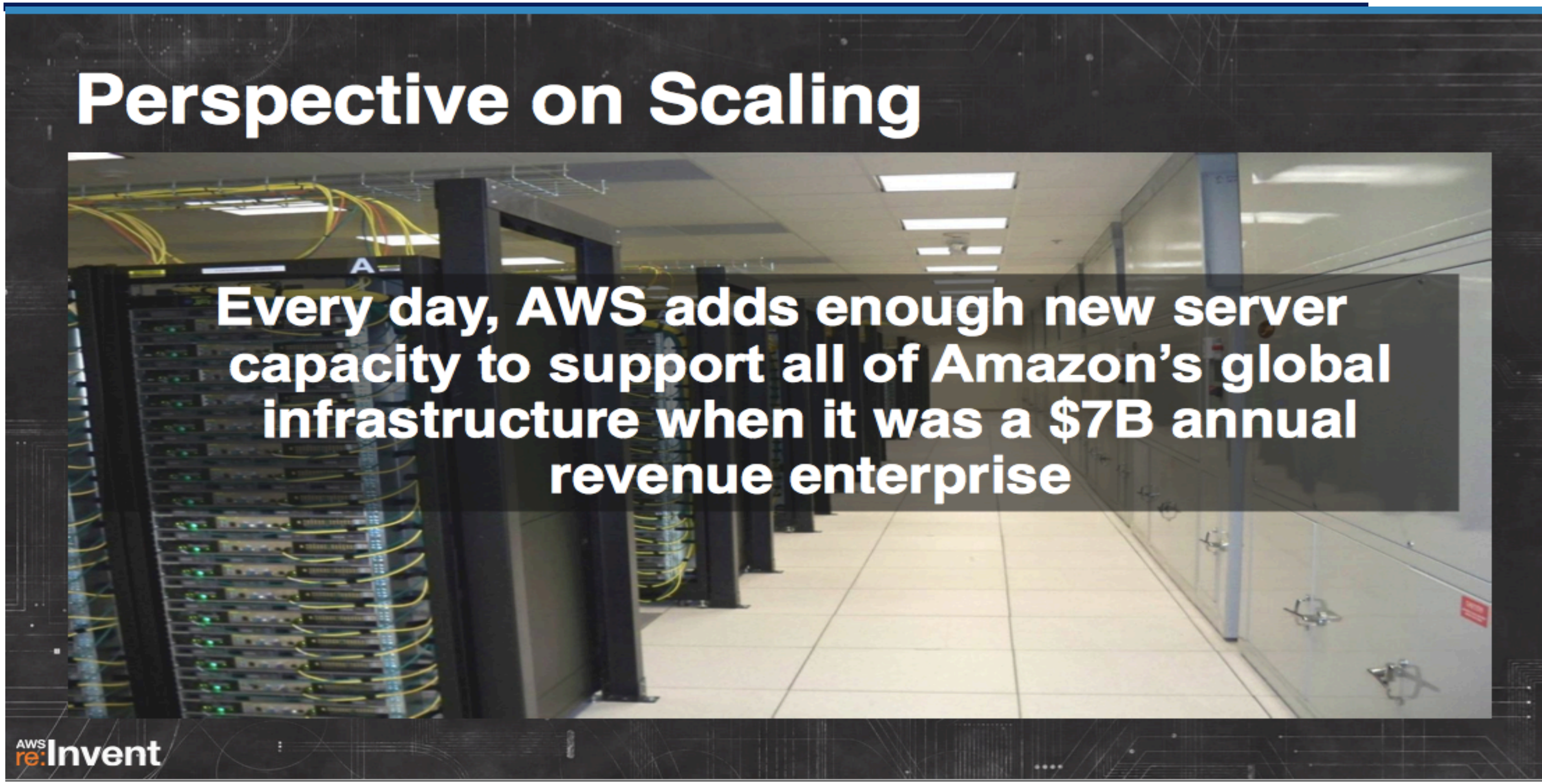
Complements theory, empirical science & simulation



Data-centric science key for innovation-based economies!

Source: James Hamilton, 2014

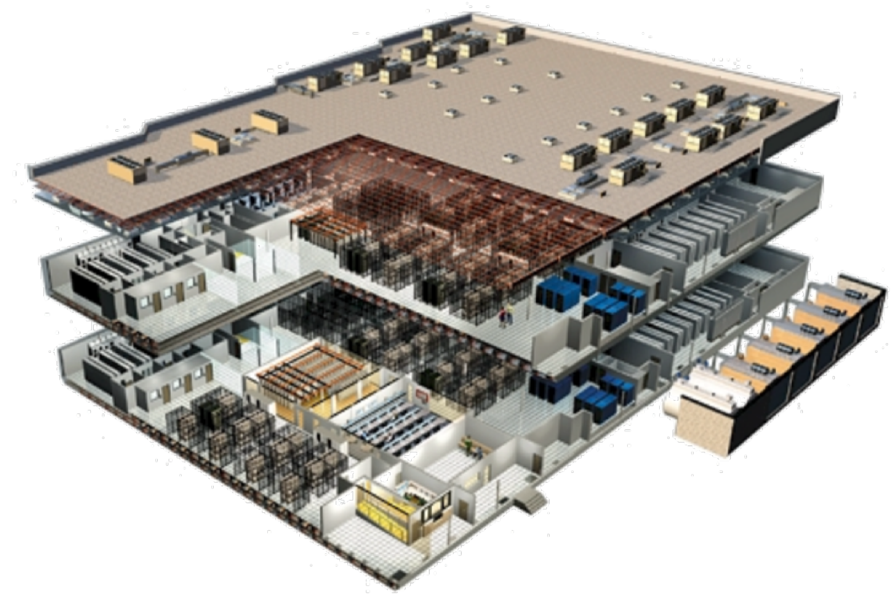
mvdirona.com/jrh/TalksAndPapers/JamesHamilton_Reinvent20131115.pdf



Daily IT growth in 2014 = All of AWS in 2004!

Modern Datacenters are Warehouse-Scale Computers

- Millions of interconnected home-brewed servers
- Centralization helps exploit economies of scale
- Network fabric provides micro-second connectivity
- At physical limits
- Need sources for
  - Electricity
  - Network
  - Cooling



30MW, 20x Football Field
$3 billion

Warning!
Datacenters are not Supercomputers

- Run heterogeneous data services at massive scale
- Driven for commercial use
- Fundamentally different design, operation, reliability, TCO
  - Density 10-25KW/rack as compared to 25-90KW/rack
  - Tier 3 (~2 hrs/downtime) vs. Tier 1 (upto 1 day/downtime)
  - ……and lots more

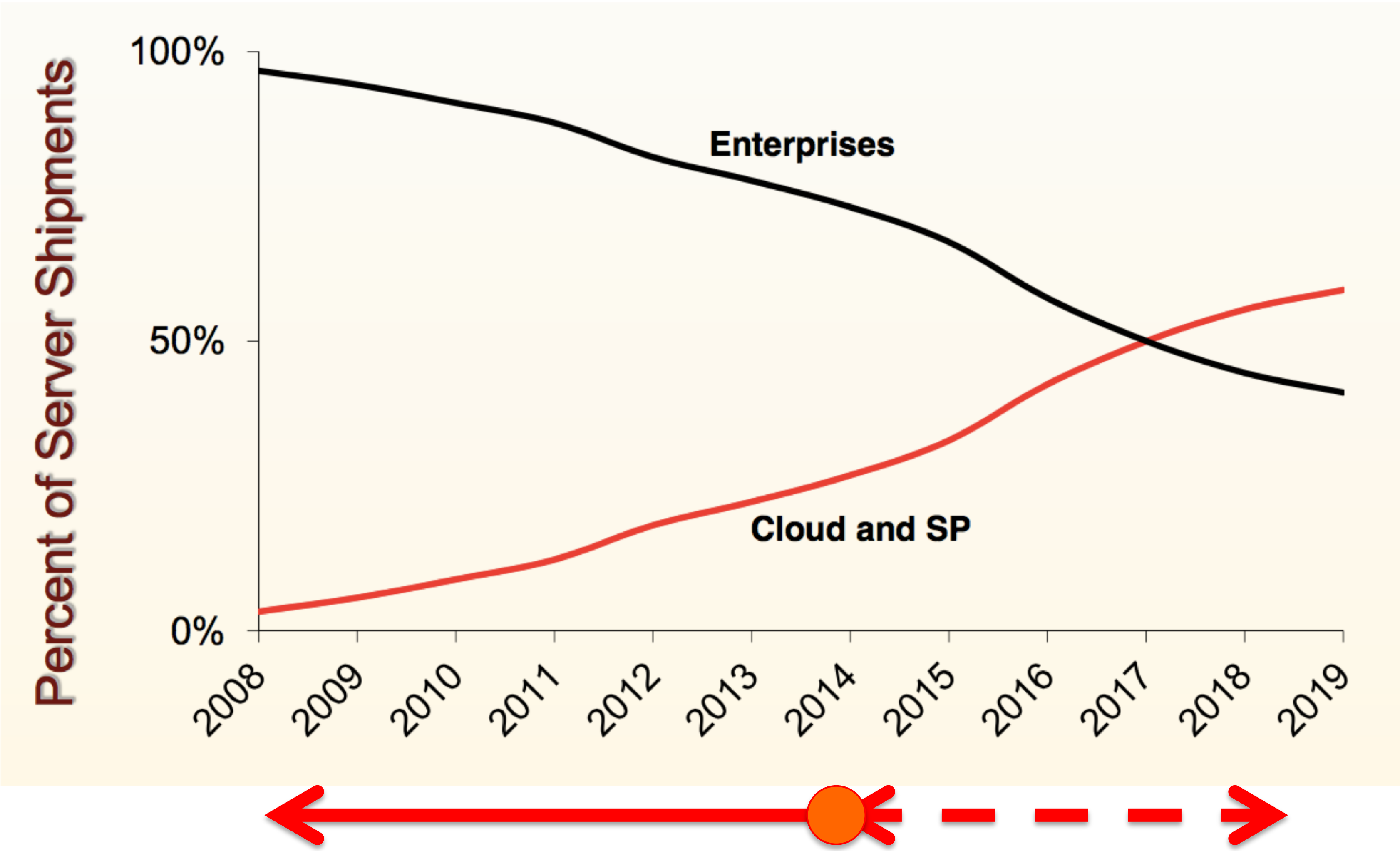
Datacenters are the IT utility plants of the future

Supercomputing

≠

Cloud Computing

Cloud Taking Over Enterprise



Source: Dell 'Oro 2Q15

Historically, what is a datacenter?

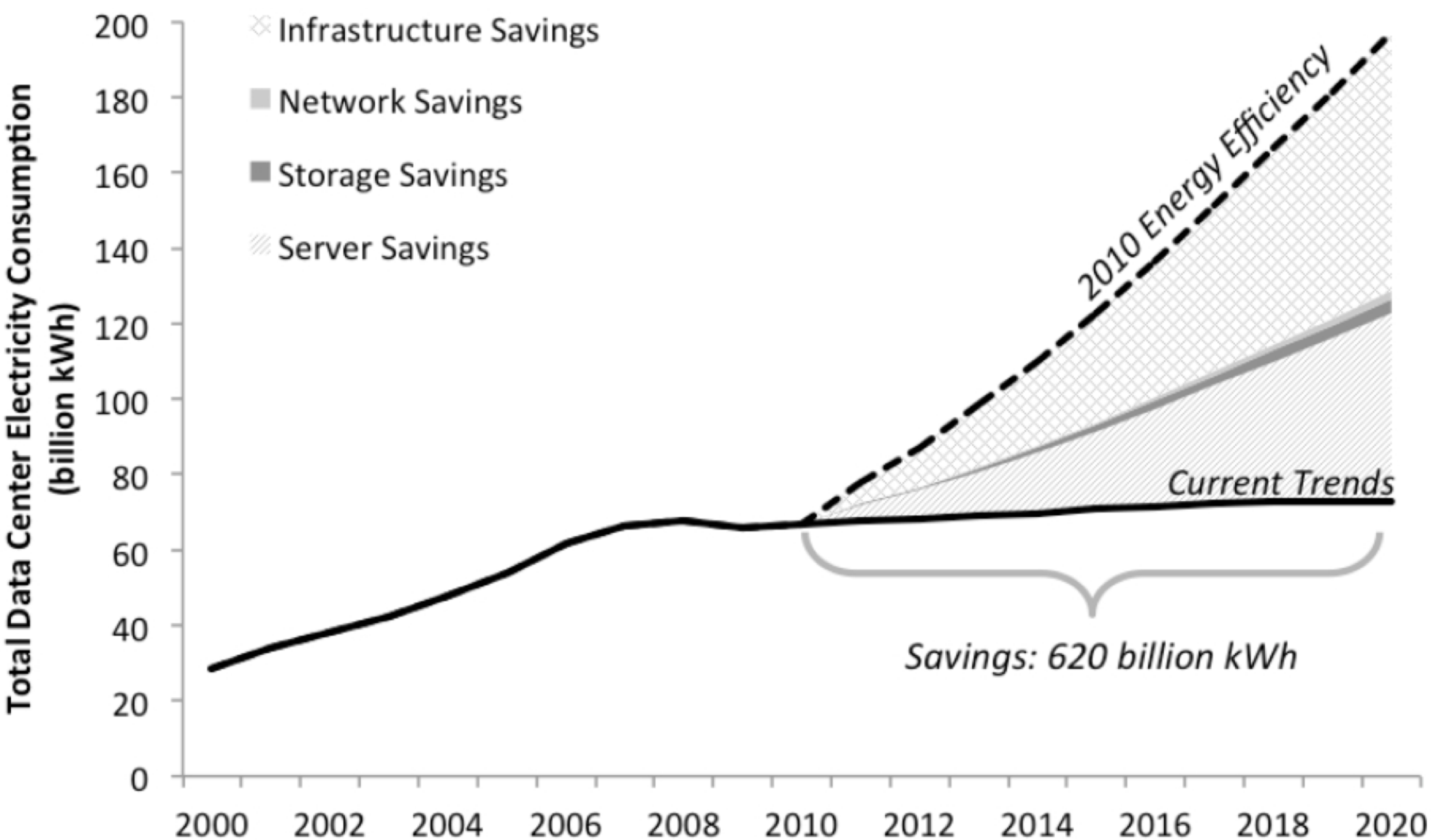
- Oxford dictionary defines "data center" as:

"A large group of networked computer servers typically used by organizations for the remote storage, processing, or distribution of large amounts of data."

- Term originated in the 1990s with the advent of client-server architecture
- Dot-com bubble ➔ internet data centers

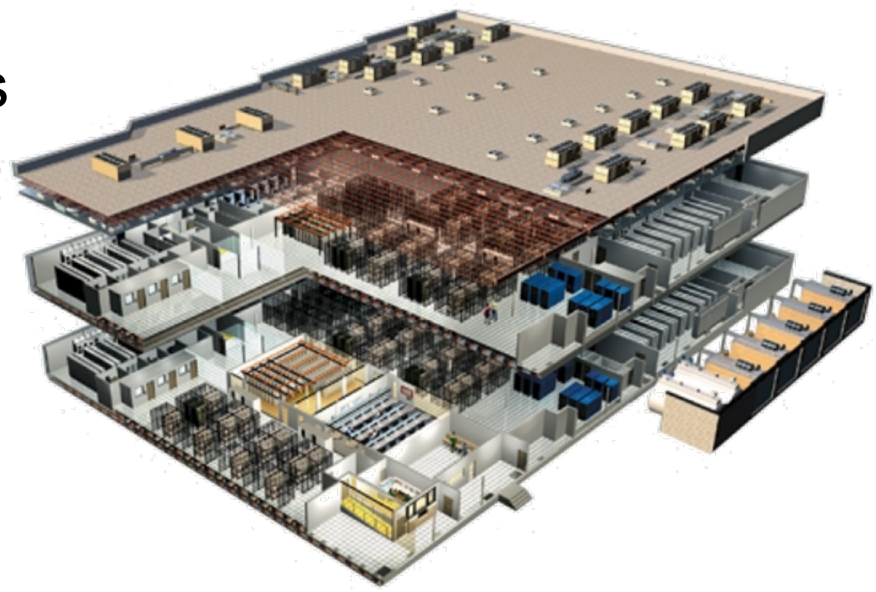
Datacenter Power

- 1.5% of worldwide electricity
  - 1.3% annual growth: energy-efficiency dramatically improved recently

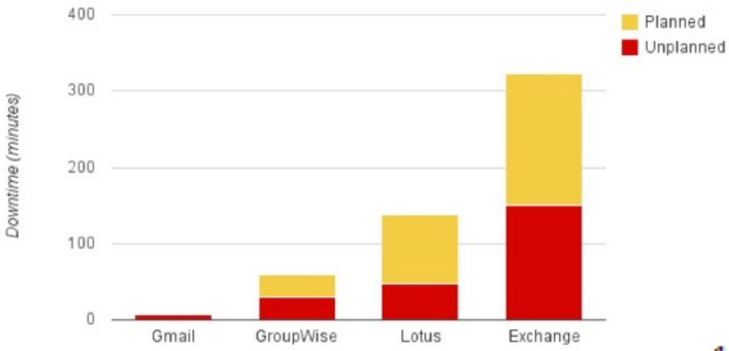


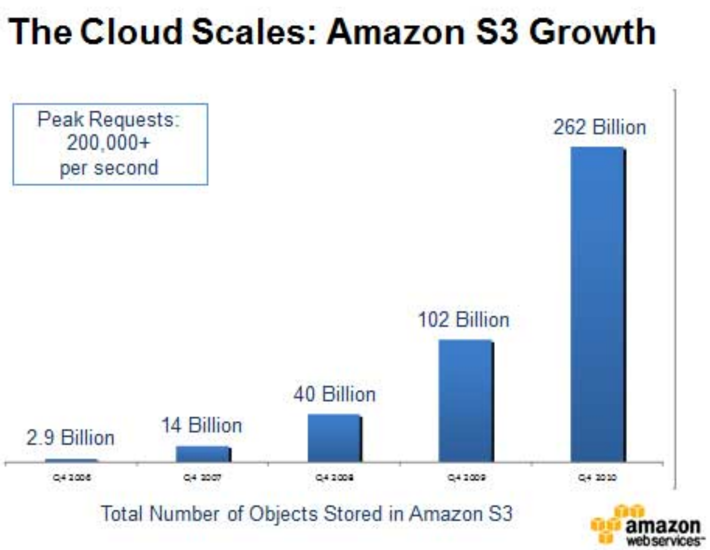
Warehouse-scale computing: The datacenter is the computer

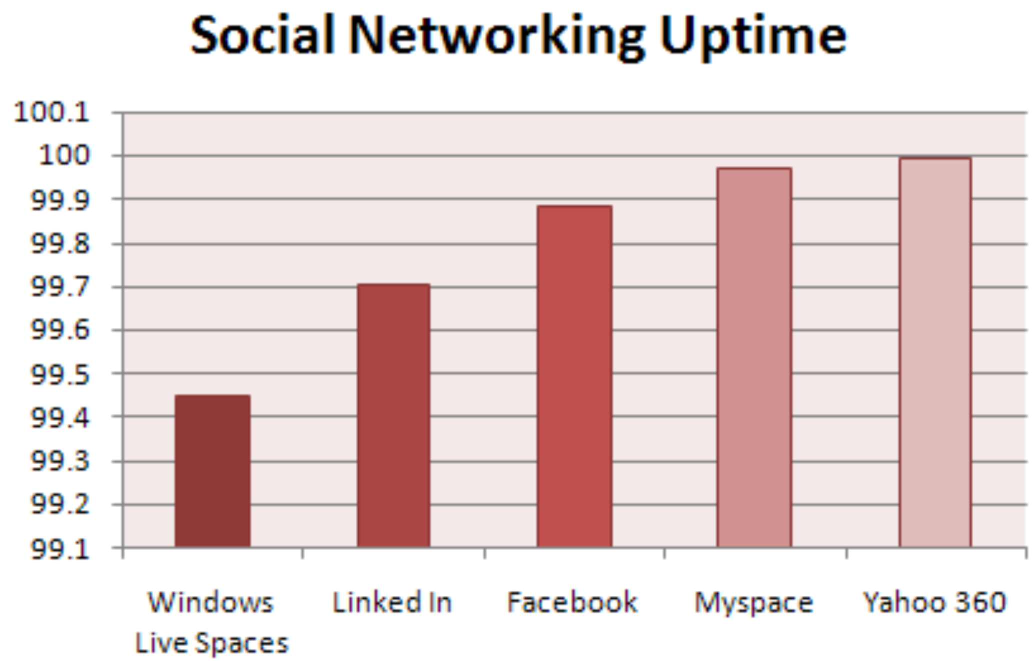
- **Program**
  - Internet service (e.g. web search, email, video streaming, maps, etc.)
- **Computer**
  - Thousands of individual computing nodes
  - Networking and storage subsystems
  - Power distribution and cooling system



Datacenter availability requirements



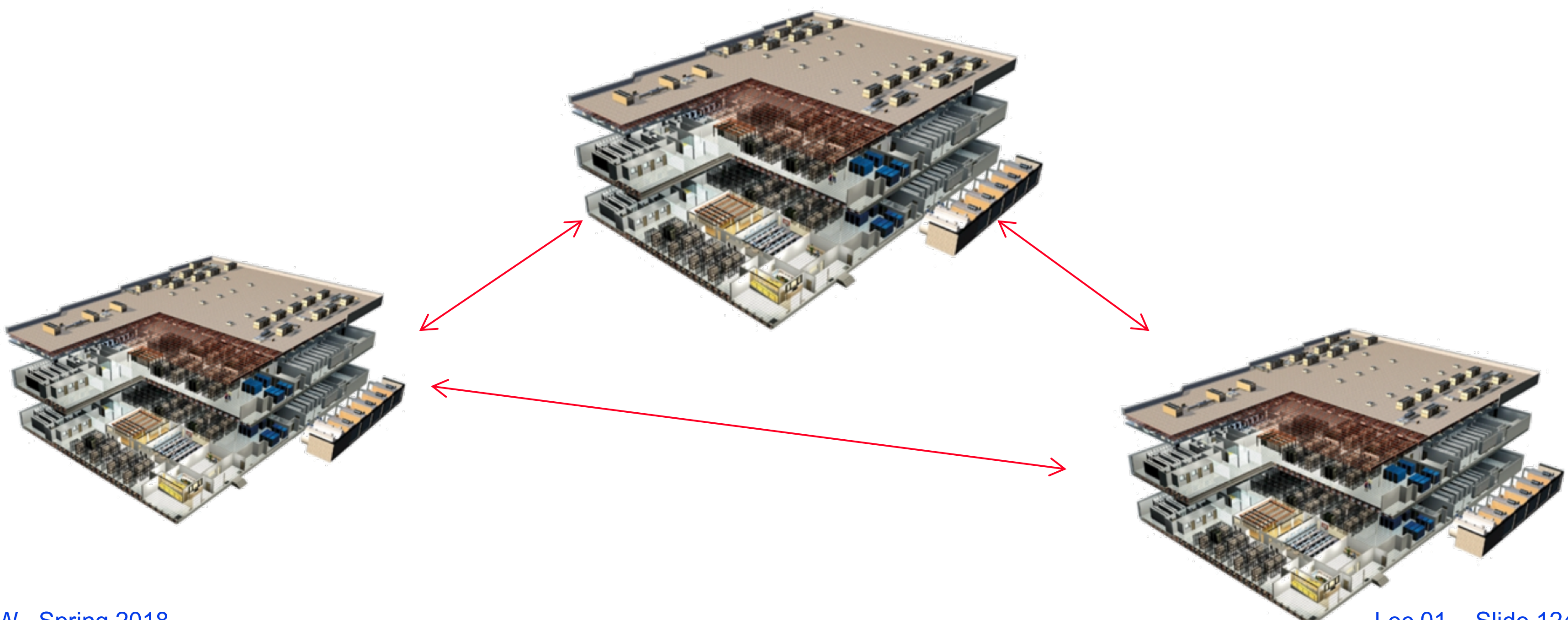




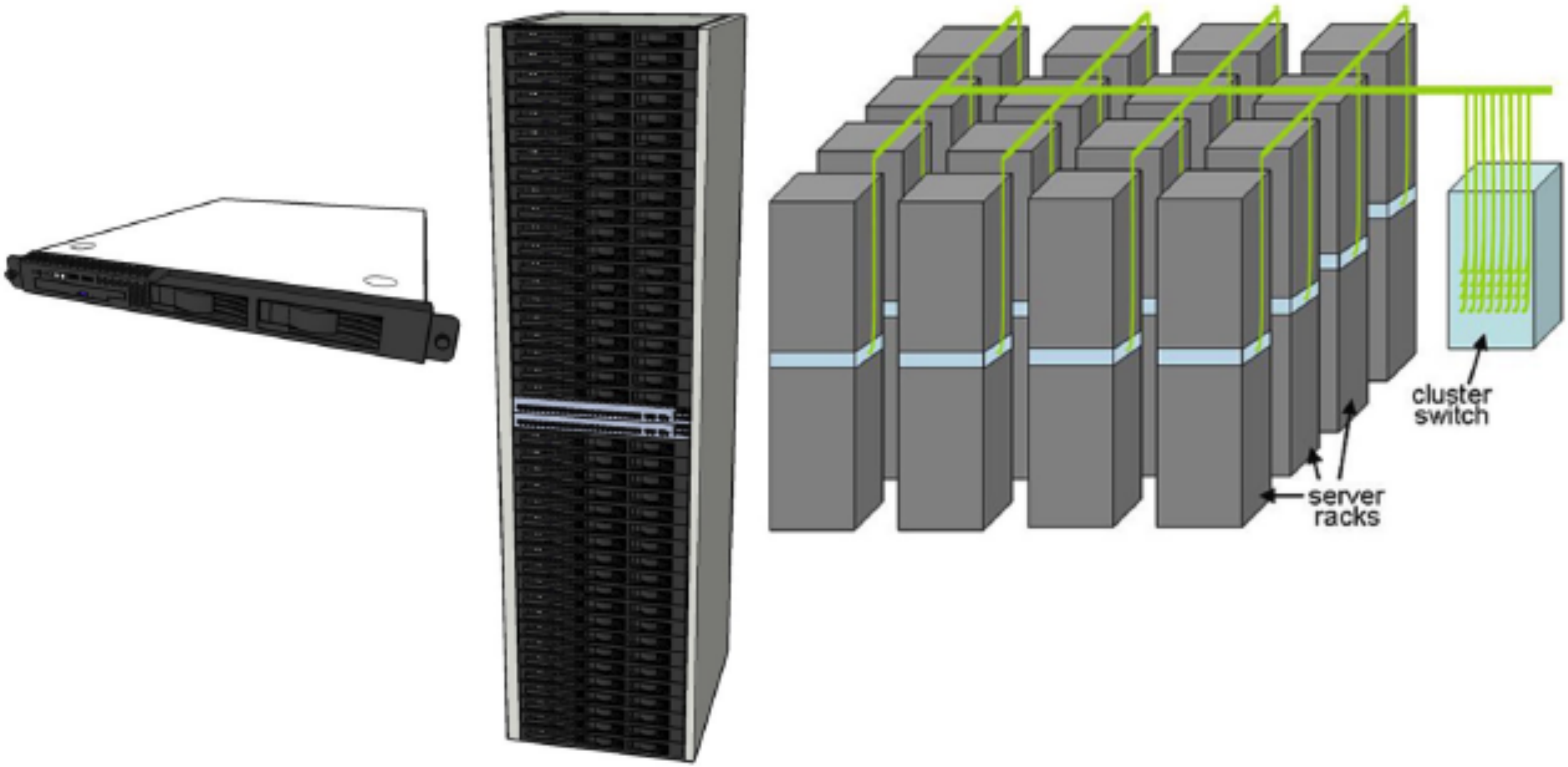
Cost and reliability are important for high availability!

Multi-datacenter scenarios

- User queries may involve computation across multiple datacenters
- Inter-datacenter communications are of much poorer quality than intra-datacenter communications



Architectural overview



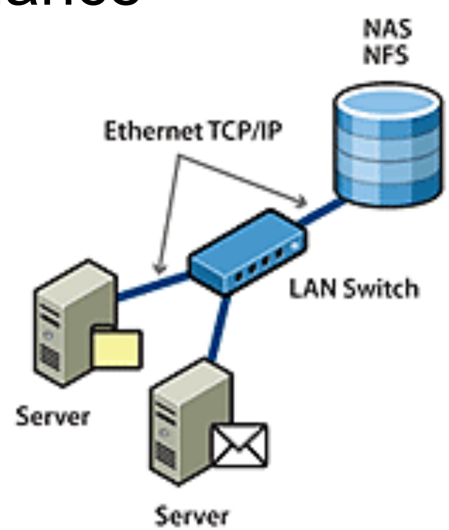
Storage management

Distributed File System

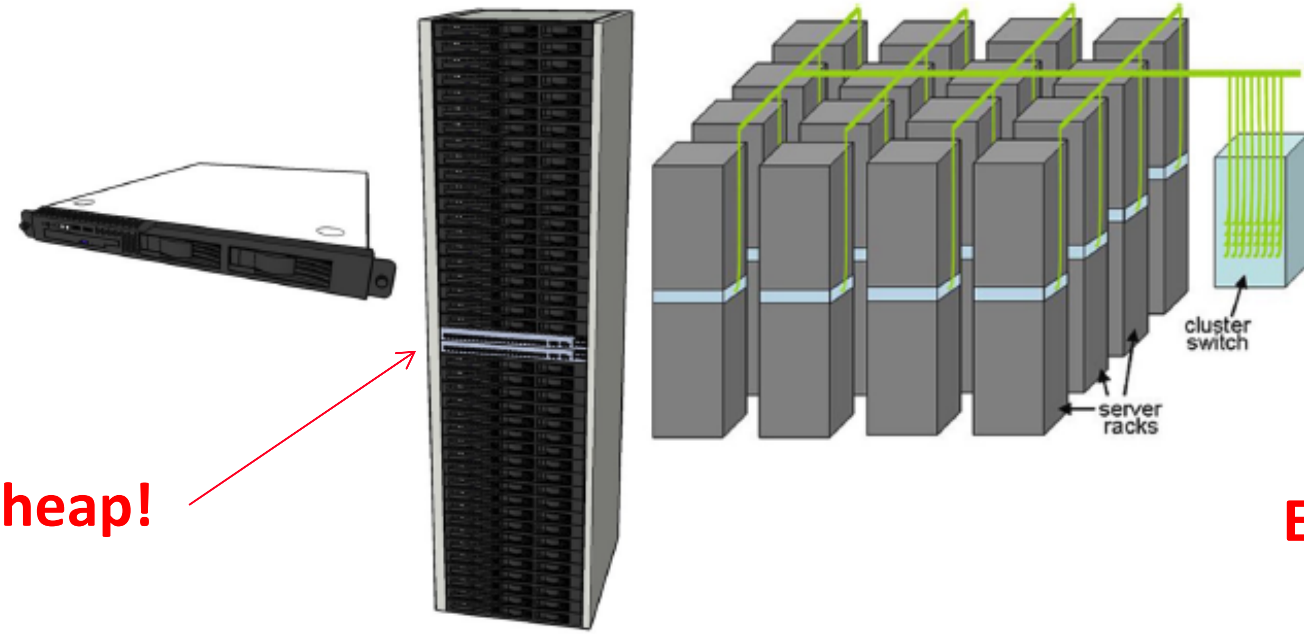
- Disk drives are directly attached to server nodes
- Replication across different machines
- Poorer write performance
- Higher read performance
- Can exploit data locality
- Google File System(GFS)

Network-Attached Storage

- Disk drives are connected to cluster-level switch
- Replication within each appliance



Network fabric

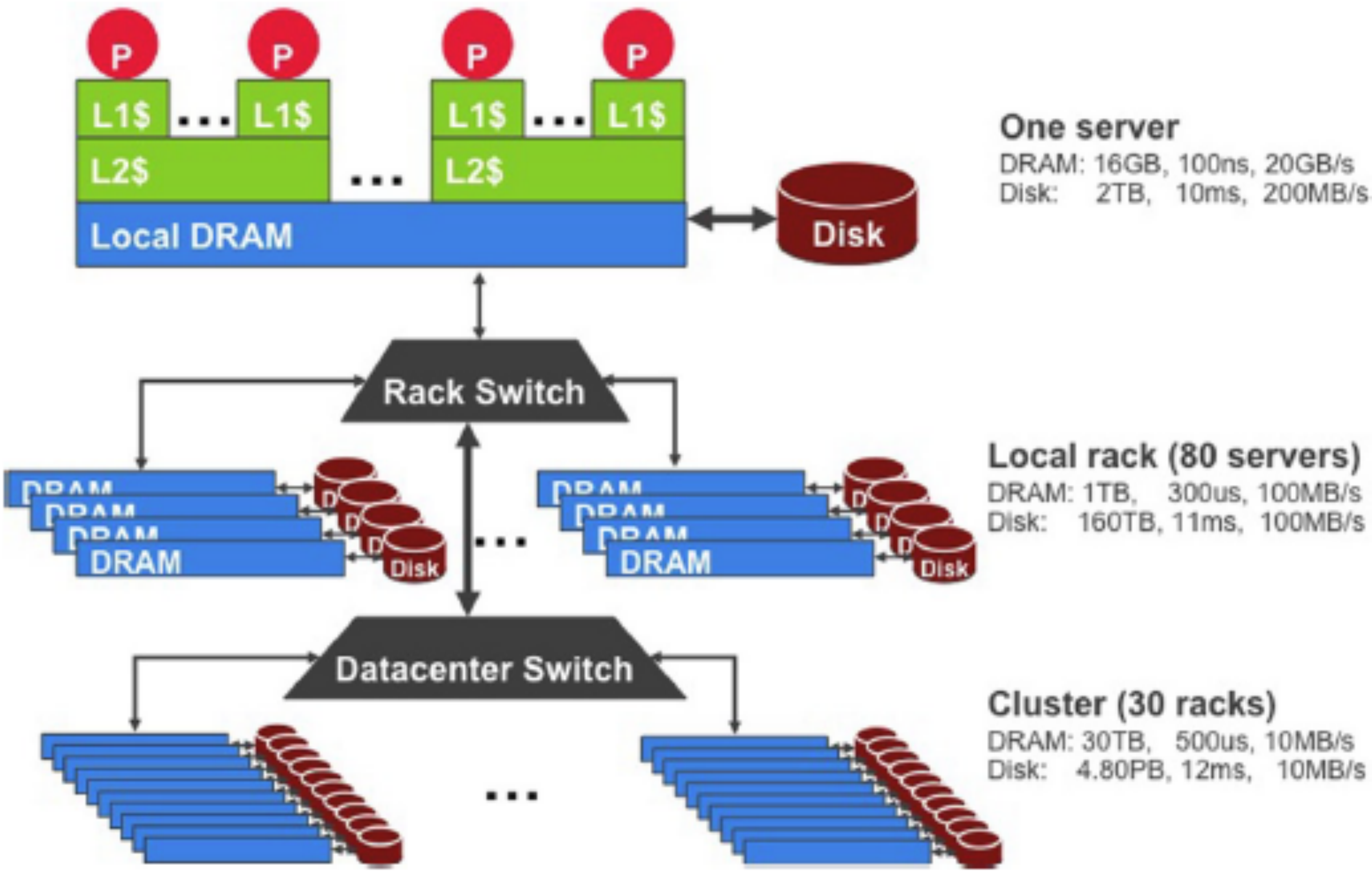


Cheap!

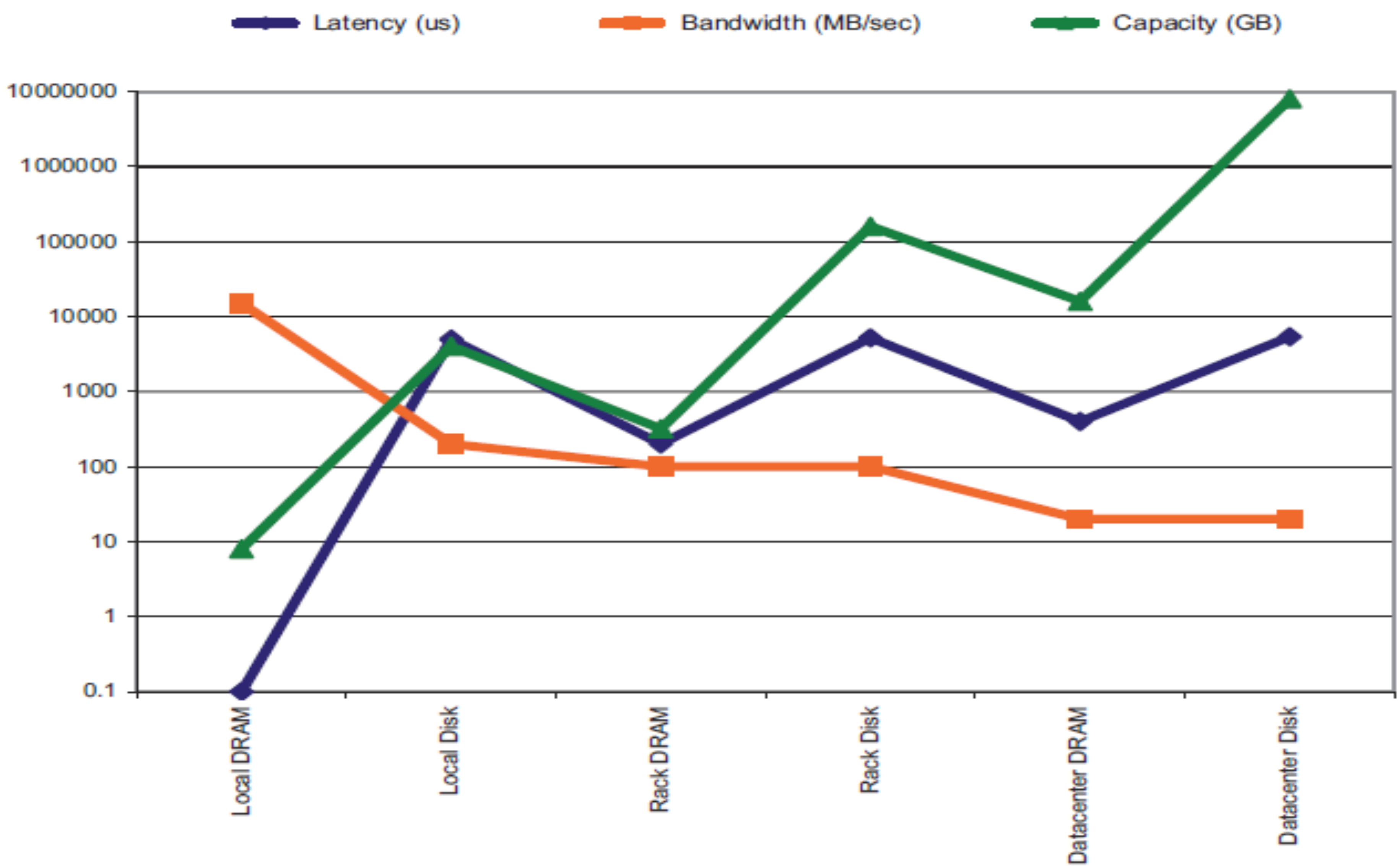
Expensive!

Software should exploit rack-level locality!

Storage hierarchy



Performance variations



Useful Numbers

Courtesy of Jeff Dean, Google

◆ L1 cache reference	0.5 ns
◆ L2 cache reference	7 ns
◆ Mutex lock/unlock	25 ns
◆ Main memory reference	100 ns
◆ Compress 1K bytes with Zippy	3,000 ns
◆ Send 2K bytes over 1 Gbps network	20,000 ns
◆ Read 1 MB sequentially from memory	250,000 ns
◆ Round trip within same datacenter	500,000 ns
◆ Disk seek	10,000,000 ns
◆ Read 1 MB sequentially from disk	20,000,000 ns
◆ Send packet CA→Europe→CA	150,000,000 ns

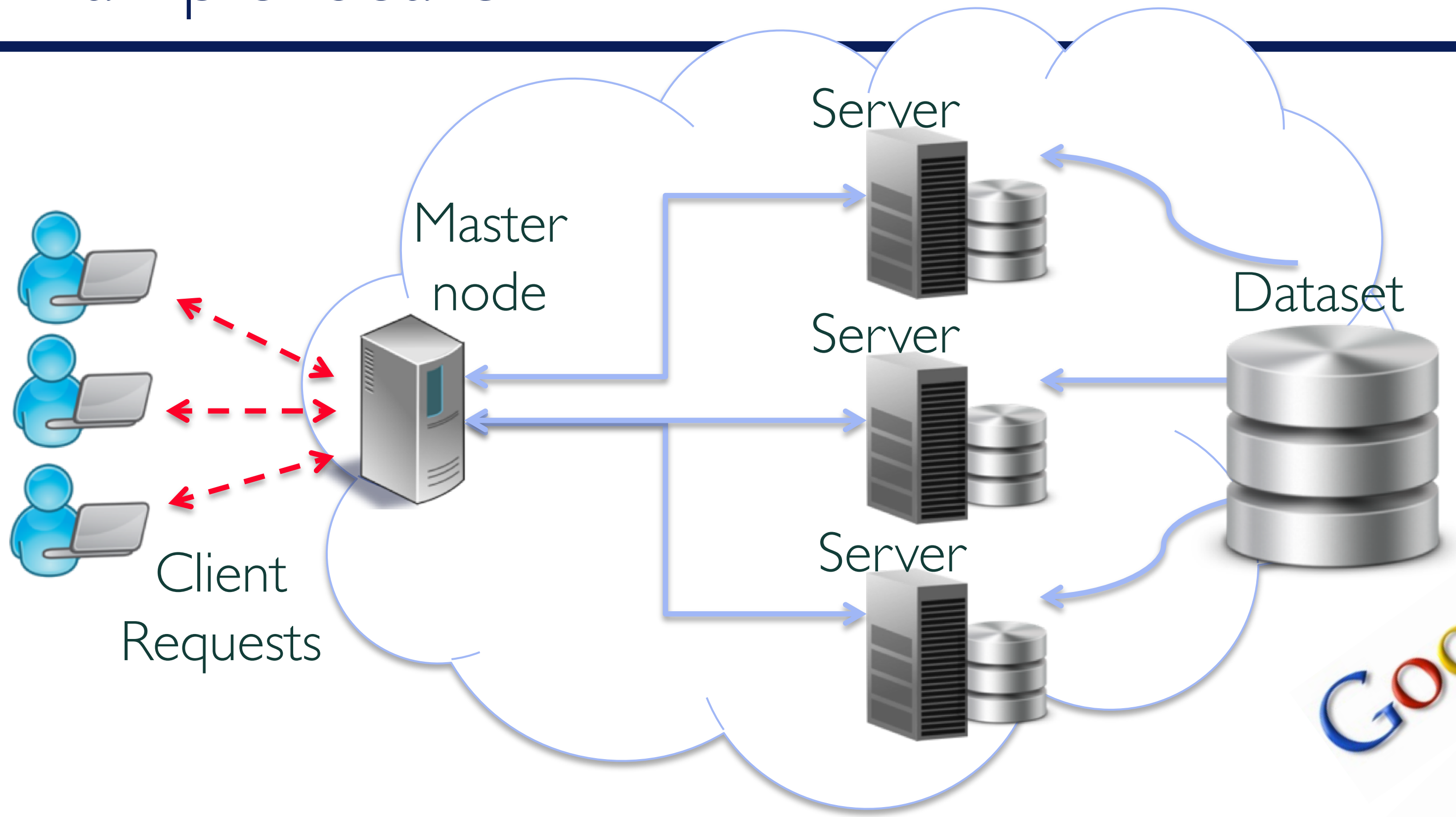
Software layers

- Platform-level software – common firmware, kernel, operating system distribution, and libraries
- Cluster-level infrastructure – distributed file systems, schedules, and remote procedure call (RPC) layers
- Application-level software – specific services
  - Online services: web search, email, maps
  - Offline services: building of web index

Datacenter software development

- Applications have inherent data parallelism or request parallelism
- Each platform generation has significant homogeneity
- Isolation of users from service implementation makes it much easier to deploy new software quickly
- Cluster-level software must deal with expected frequent hardware failure

Example: Search



- Many independent requests/tasks
- Huge dataset split into shards
- Use aggregate memory over network

Cluster-level infrastructure software

- Resource management
  - Maps user tasks to hardware resources
  - Enforces priorities and quotas
  - Users specify job requirements
    - e.g., CPU performance, memory capacity, bandwidth
- Hardware abstraction
- Deployment and maintenance
- Programming frameworks

Software-based fault tolerance

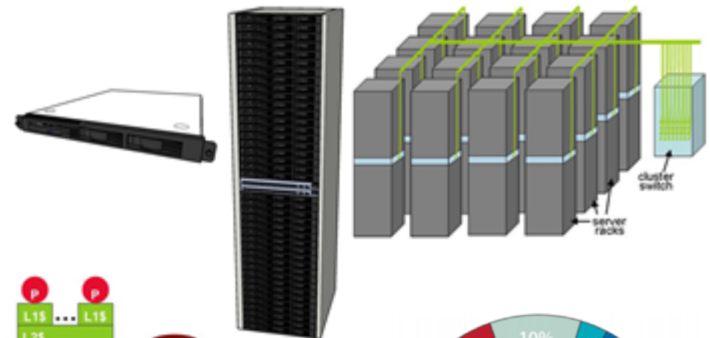
Fault tolerance:

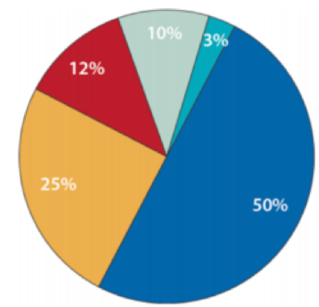
- Phones crash all the time (not safety critical)
- Planes, cars, hospital equipment, supercomputers can not crash often (use expensive hardware)
- Servers are cheap, crash less often but repaired in software and have a legal contract to customers

- Key idea: hardware faults are detected and reported to software in a timely manner, software takes appropriate action to manage the fault
- Hides complexity from application-level software

Data Center Power Consumption

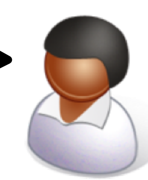
Computing/IT equipment





Over half of energy consumption

What else consumes power in a datacenter?



Power distribution units, cooling system, etc.

The Components of a Typical Datacenter

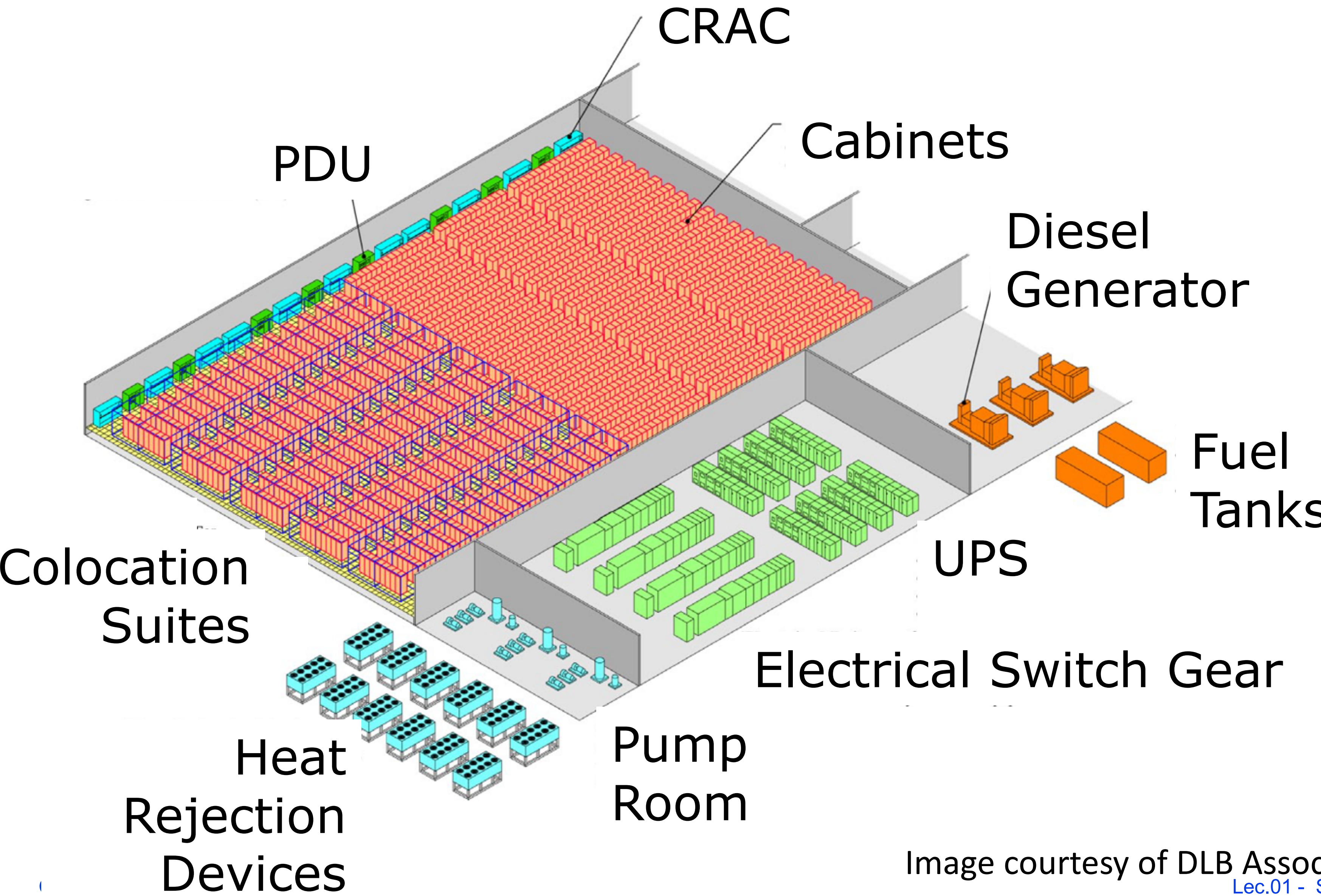


Image courtesy of DLB Associates

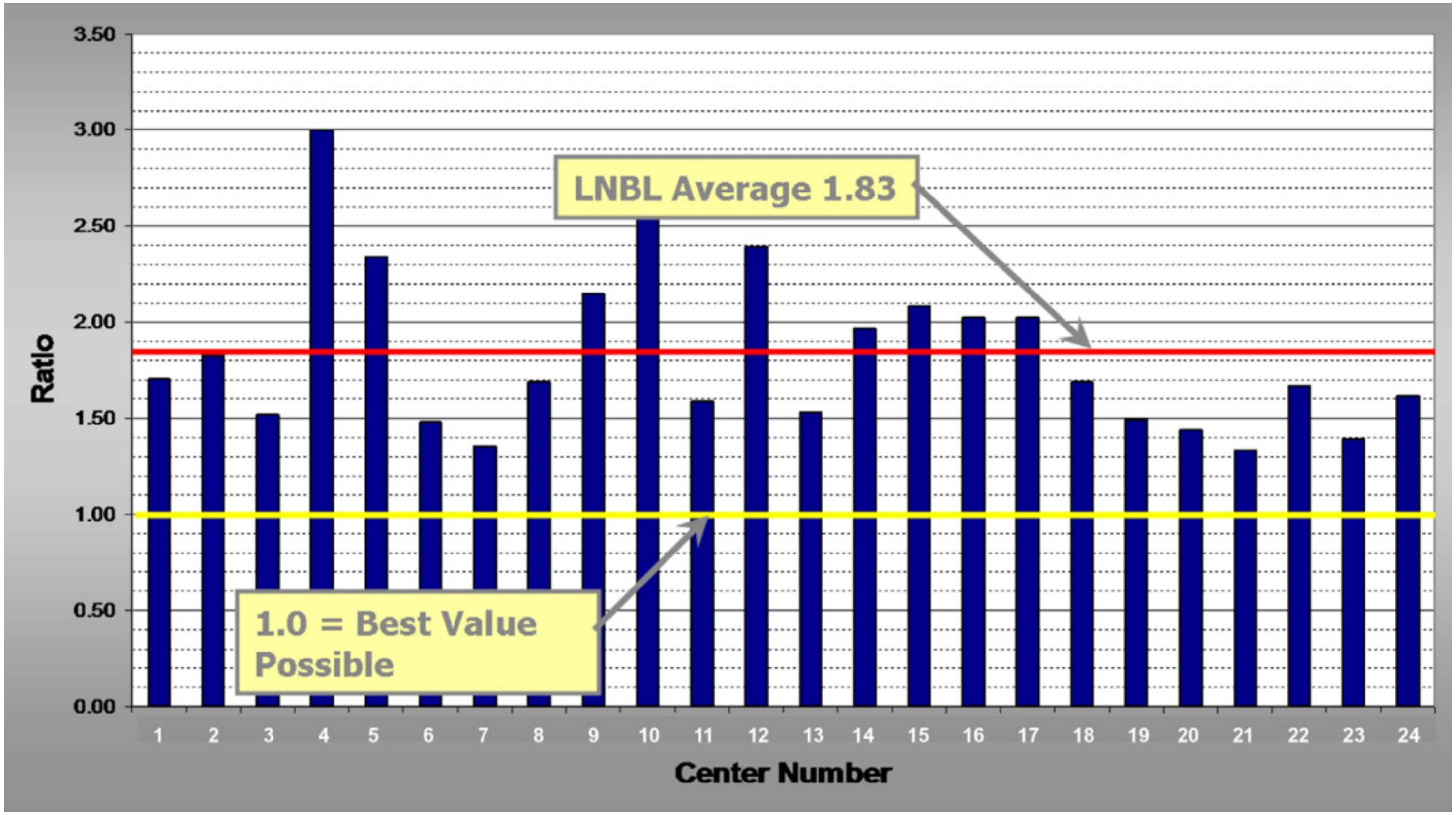
How to Measure Datacenter Energy Efficiency

- PUE = power usage effectiveness
  - Building power/power of IT (servers, network)
  - Current state of the art PUE = ~1.1
- SPUE = server power usage effectiveness
  - Server power/power for server components (e.g., CPU, disk)
  - State of the art SPUE = 1.1
- If PUE=SPUE=1.1 => 10% of energy is "wasted"

$$Efficiency = \frac{Computation}{Total\ Energy} = \left(\frac{1}{PUE}\right) \times \left(\frac{1}{SPUE}\right) \times \left(\frac{Computation}{Total\ Energy\ to\ Electronic\ Components}\right)$$

PUE Statistics

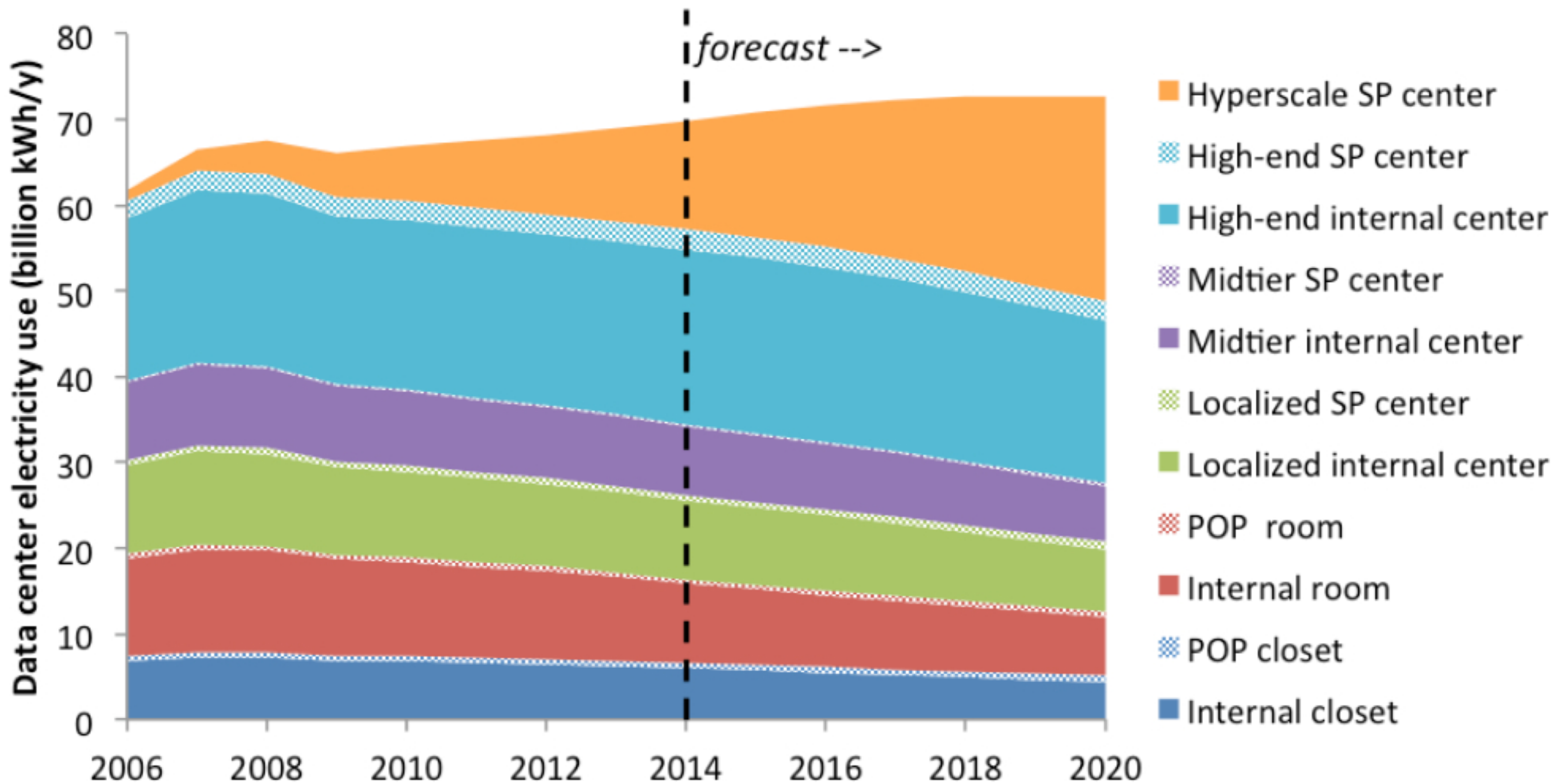
- [2006] About 85% of datacenters estimated to have PUE > 3.0
- [2006] Only about 5% of datacenters estimated to have PUE ≤ 2.0
- [2010] Average PUE of approximately 1.89



Update: LBNL survey of PUE of 24 datacenters, 2007 [Greenberg et al.]

Datacenter consolidation

Trend is to consolidate workloads in hyperscale datacenters

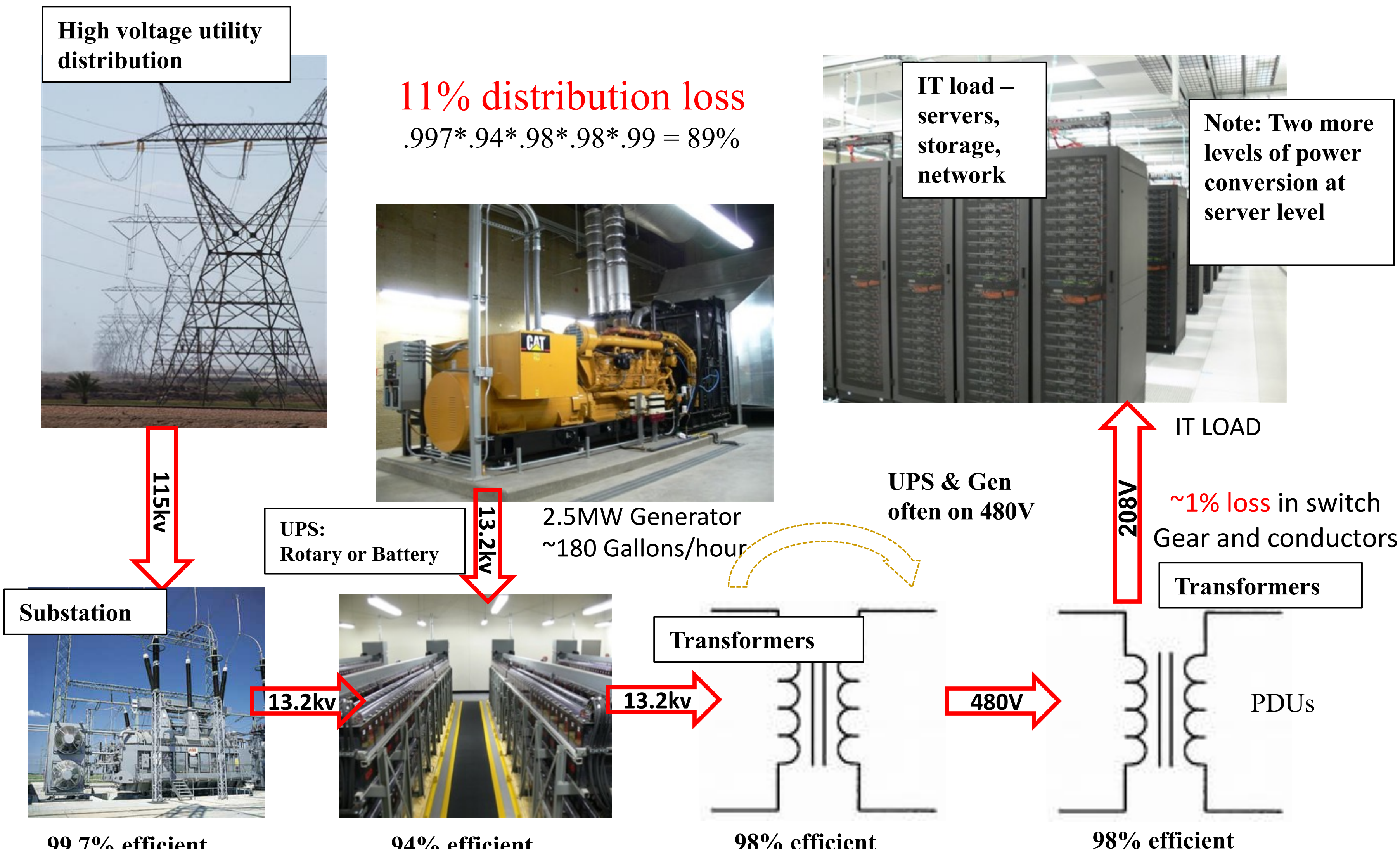


Hyperscale datacenters are more efficient

- Avg. PUE comparison:
  - Hyperscale datacenter: 1.1
  - Midtier datacenter: 1.6
- Reasons for increased efficiency:
  - Higher utilization
  - Redundancy control is cheaper at high scale

More hyperscale → more efficient datacenter computing

Sources of Efficiency Loss: Power Distribution (James Hamilton)



Old School Datacenter Cooling System

- Datacenter floor is raised 2-4ft above the concrete floor
- Under-floor area helps distribute cool air to the server racks
- Also used for routing power/network cables to the racks

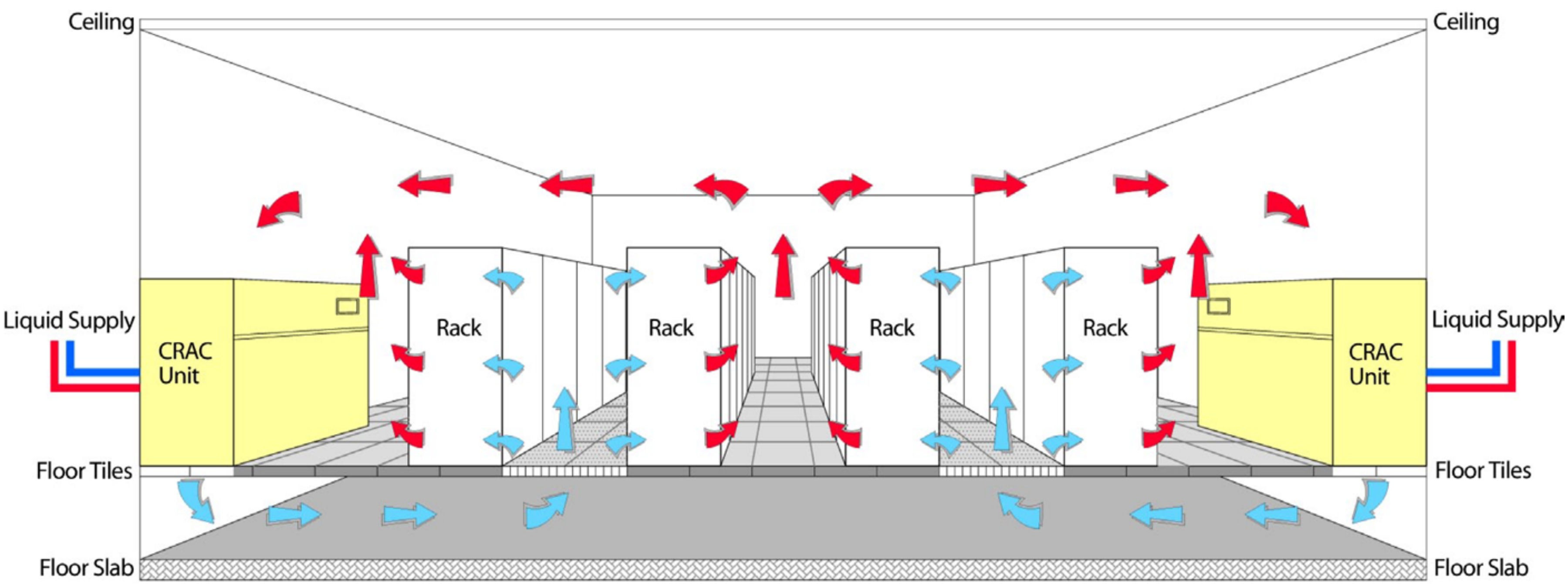
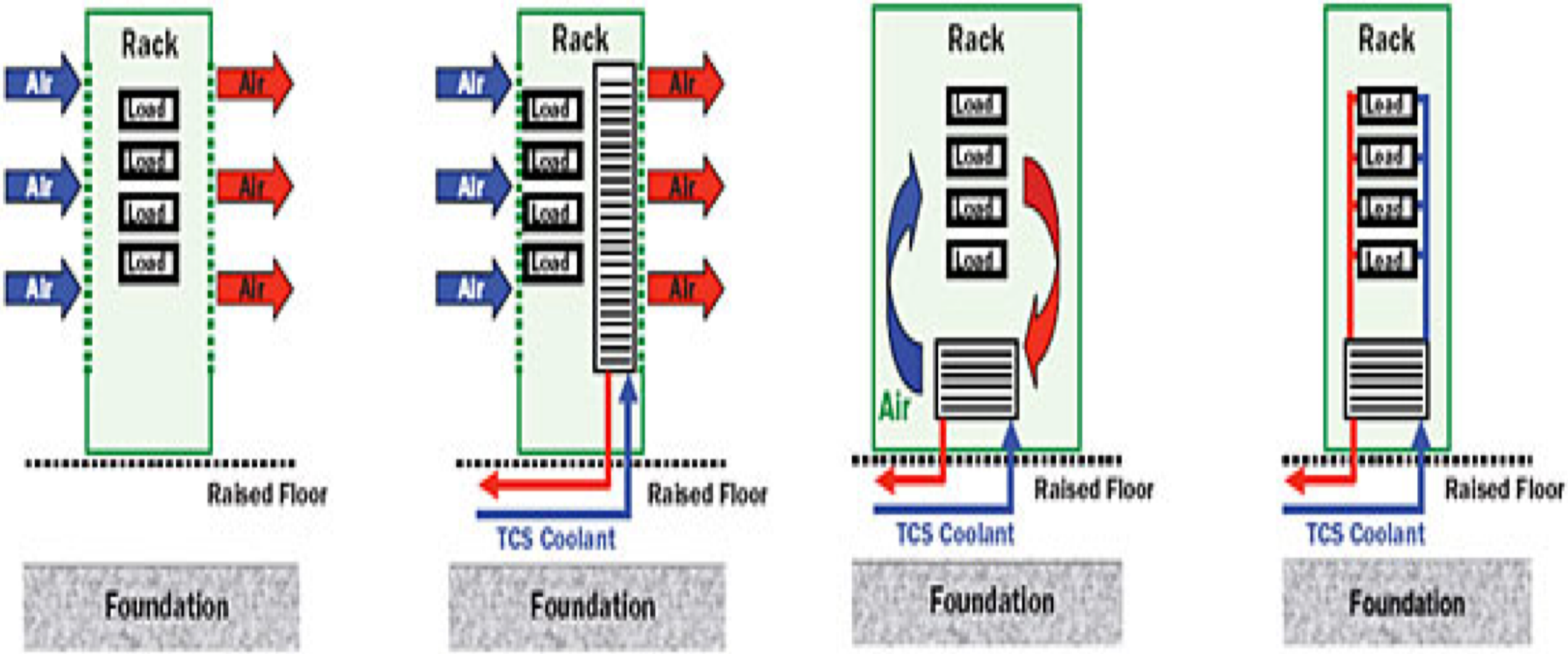


Image courtesy of DLB Associates

New School: Chilled-Water Cooling



Open air-cooled cabinet

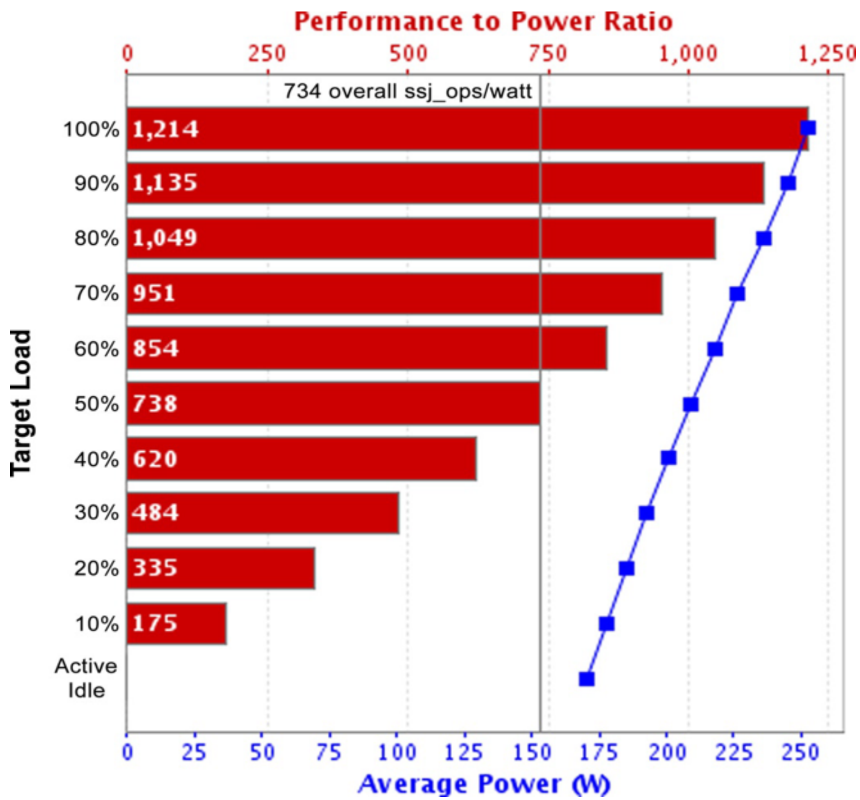
Open liquid-cooled cabinet (air-to-liquid heat exchanger)

Closed liquid-cooled cabinet (air-to-liquid heat exchanger)

Closed liquid-cooled cabinet (liquid-to-liquid heat exchanger)

Key Energy Usage Feature of Current Servers

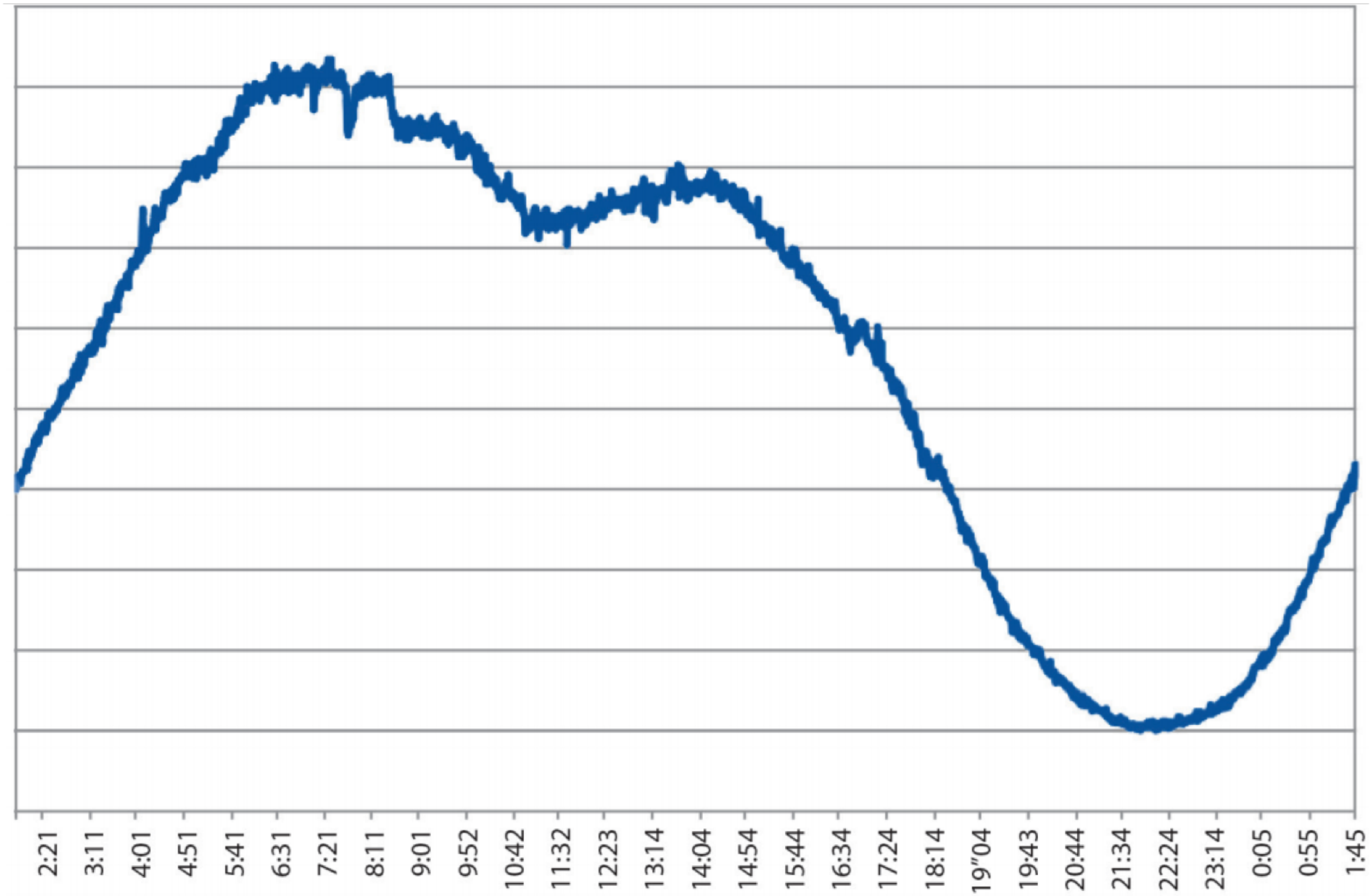
- Under low utilization, the inefficiency is significantly higher
  - Cause: Idle power consumption more than half of peak!!!
- Individual servers spend negligible time completely idle



An example benchmark result for SPECpower_ssj2008; energy efficiency is indicated by bars, whereas power consumption is indicated by the line. Both are plotted for a range of utilization levels, with the average metric corresponding to the vertical dark line. The system has a single-chip 2.83 GHz quad-core Intel Xeon processor, 4 GB of DRAM, and one 7.2 k RPM 3.5" SATA disk drive.

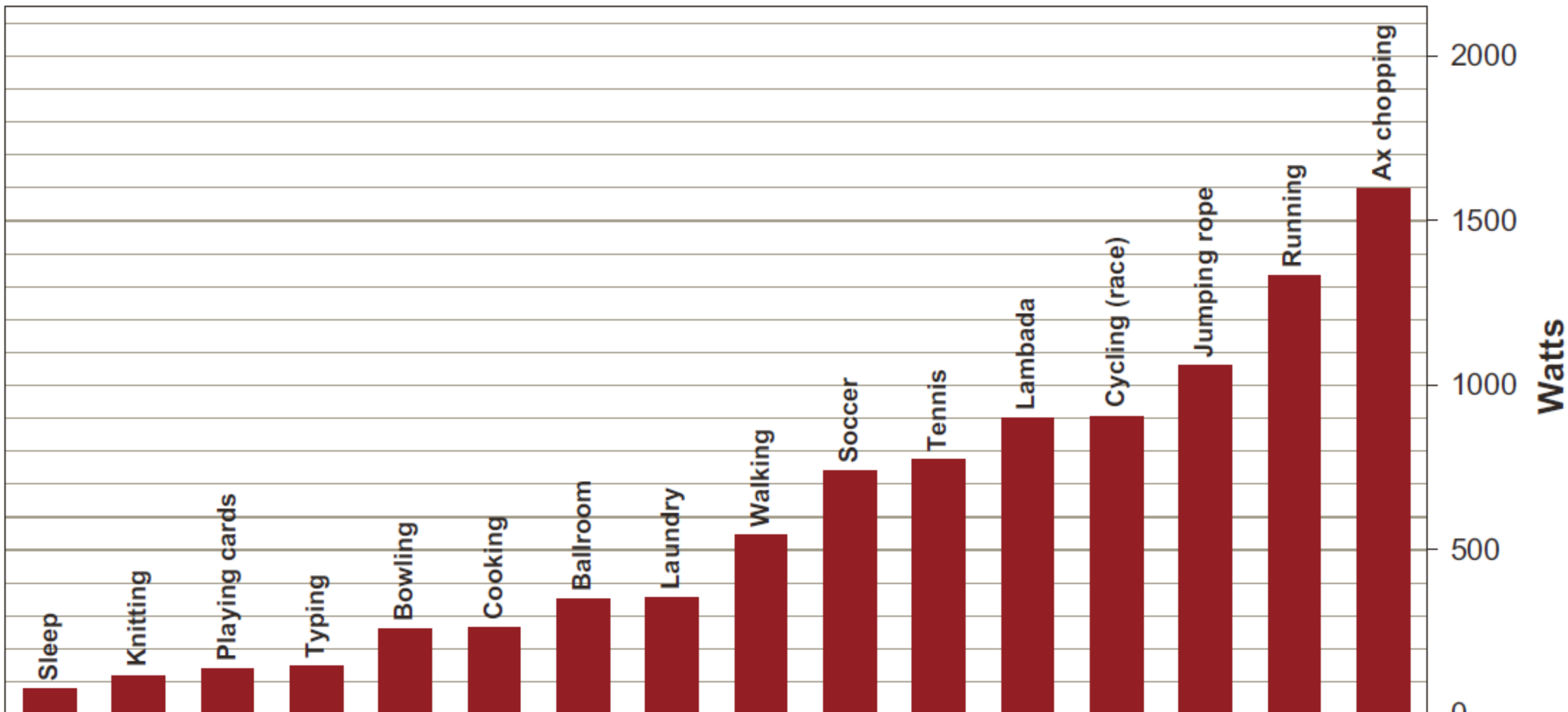
Load vs. Efficiency

- Off-peak traffic is 2x lower than peak period
  - Need to operate efficiently with low utilization



Dynamic Power Range

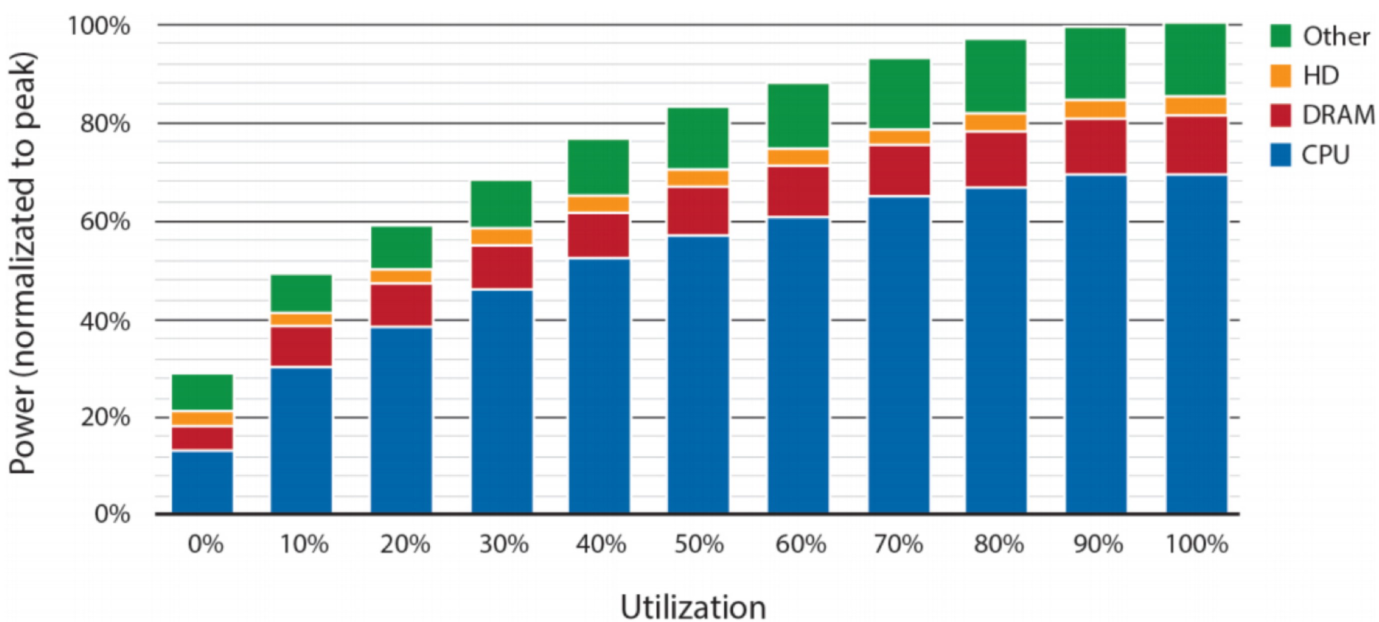
Energy-proportional machines would exhibit a wide dynamic power range – rare in computing equipment (merely 2x), but not unprecedented in other domains (e.g. Humans have a 20x factor)



Human energy usage vs. activity levels (adult male)

Causes of Poor Energy Proportionality

- CPUs are not necessarily the main culprit!
- Over the years, CPU designers have been more attentive to energy efficiency than their counterparts for other subsystems (e.g., switching to multicore vs. higher clock frequencies)
- Server-class CPUs have dynamic power range of 3.0x or more (compare with: 2.0x for memory, 1.3x for disks, less than 1.2x for networking switches)



Role of Software

- Clever software strategies can enhance energy-proportionality of the underlying hardware:
  - Intelligent use of power management features in existing hardware
  - Using low-overhead inactive or active low-power modes
  - Power-friendly scheduling of tasks

- Challenges
  - Encapsulation (avoid exposure to developers)
  - Robustness (avoid side-effects like increased variability of response-time)

Summary

- Datacenters basics
- Software
  - Parallel
  - Fault-tolerant
- Energy efficiency
  - Decrease PUE*SPUE
  - Energy proportional computing

Roadmap

- Technology
  - Moore's Law
  - Parallelism
- Datacenters & Centralization
  - Economies of scale
  - Metrics
- Service-Oriented Computing
  - Cloud
  - Virtualization

A Brief History of IT

Mobile Era

Consumer Era

1970s- 1980s 1990s Today+

Mainframes

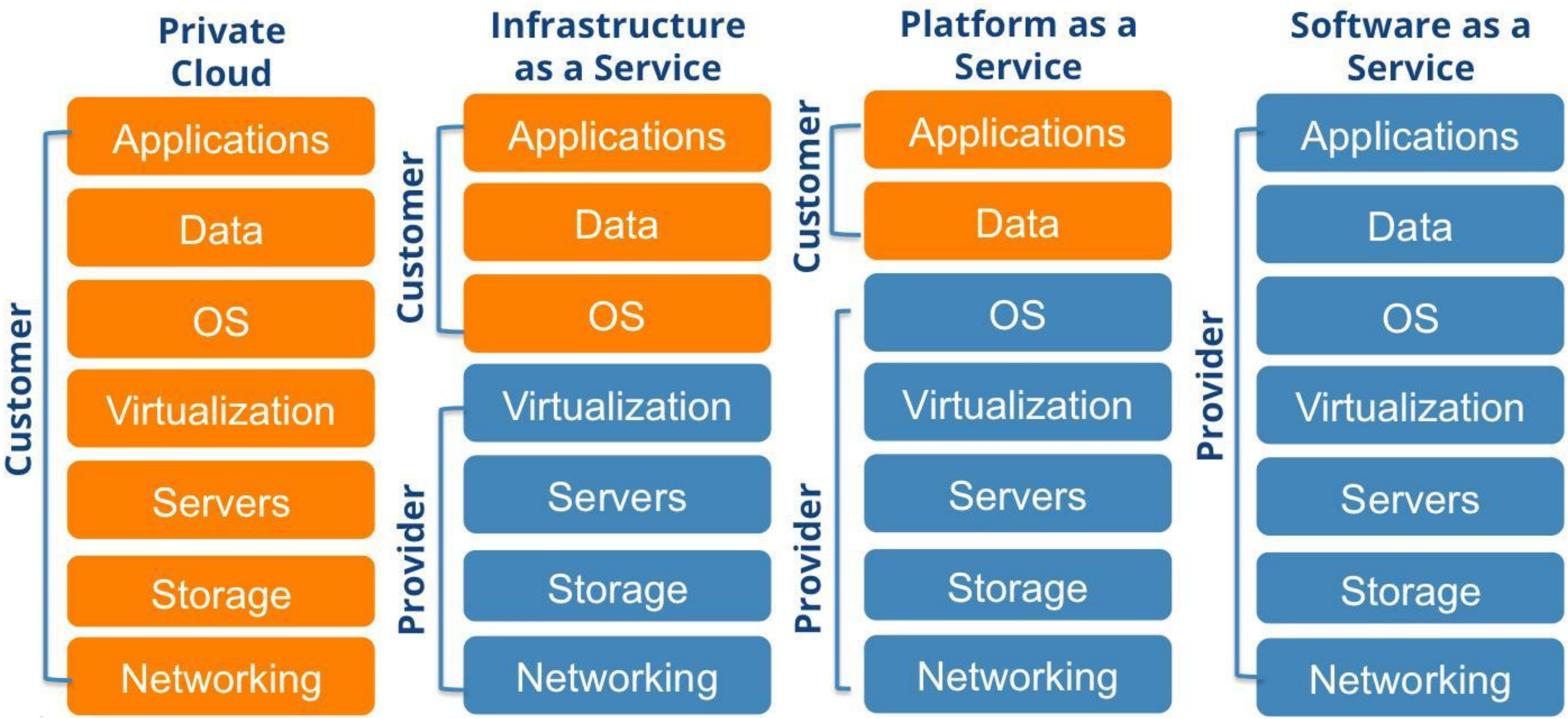
PC Era

- From computing-centric to data-centric
- Consumer Era: Internet-of-Things in the Cloud

Cloud Computing

- IT resources provided as a service
  - Compute, storage, databases, queues
- Clouds leverage economies of scale of commodity hardware
  - Cheap storage, high bandwidth networks & multicore processors
  - Geographically distributed data centers
- Focus on business, science, governance rather than IT maintenance
- Sign a contract to use a service
- Offerings from Microsoft, Amazon, Google, …

Cloud Service Models



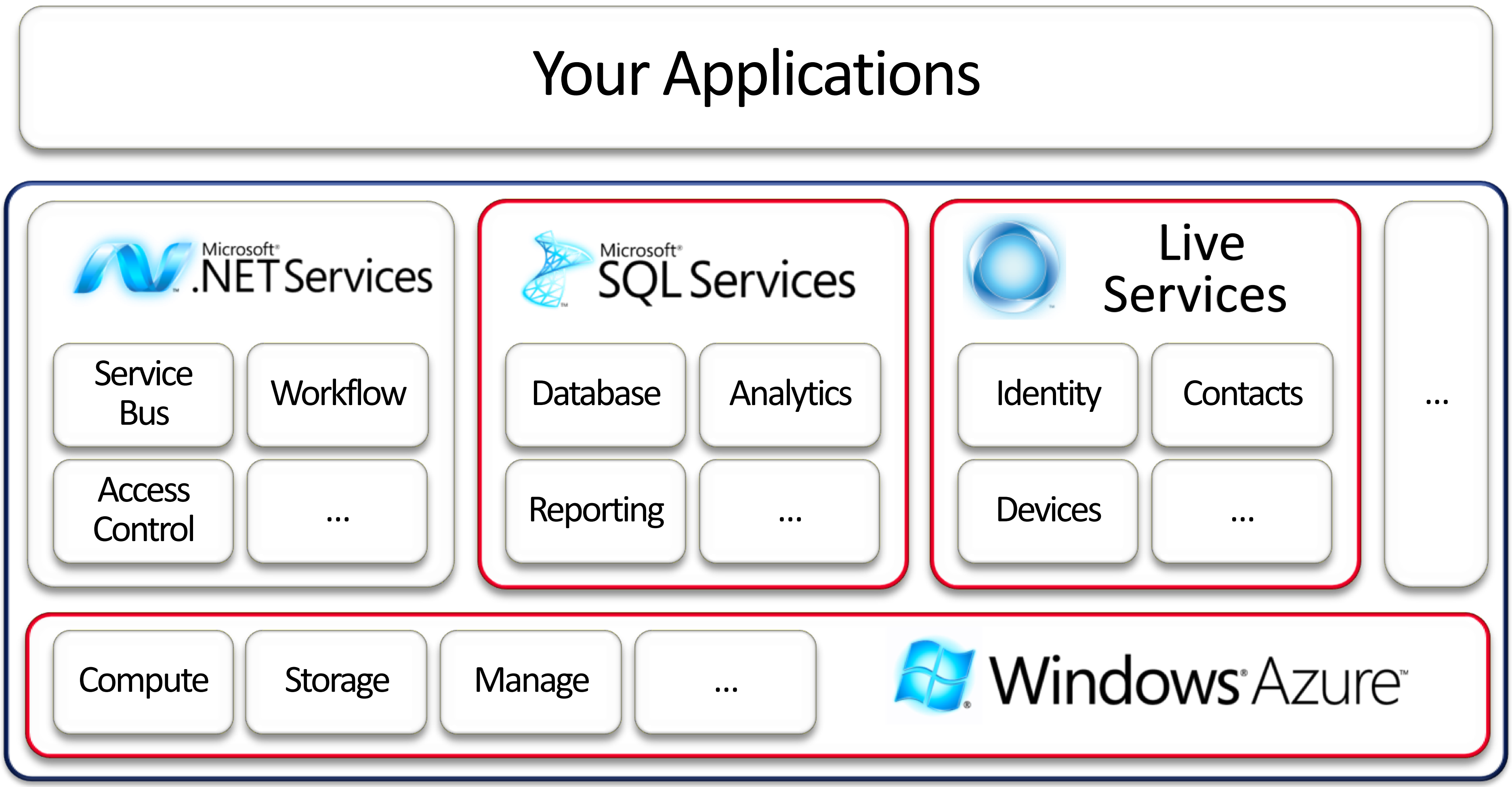
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
IC Cluster

amazon web services
rackspace HOSTING

App Engine
Windows Azure

salesforce CRM
LotusLive

Example: Microsoft Azure

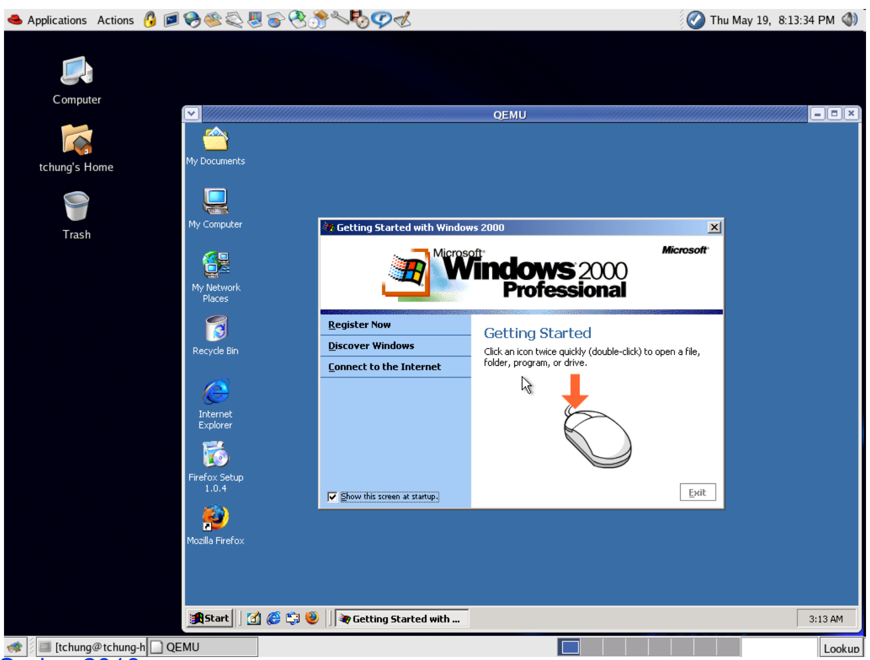


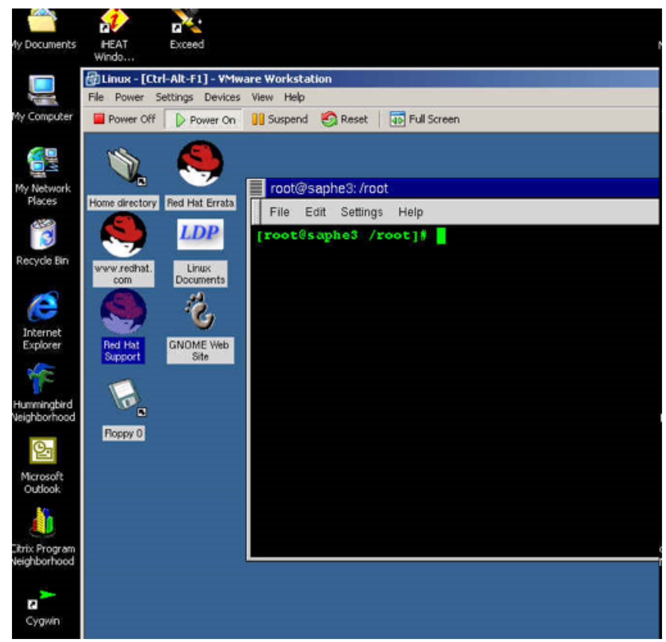
Cloud Model Service Layers [src: Mark Baker]

	Services	Description
Application Focused	Services	Services – Complete business services such as PayPal, OpenID, OAuth, Google Maps, Alexa
	Application	Application – Cloud based software that eliminates the need for local installation such as Google Apps, Microsoft Online
	Development	Development – Software development platforms used to build custom cloud based applications (PAAS & SAAS) such as SalesForce
Infrastructure Focused	Platform	Platform – Cloud based platforms, typically provided using virtualization, such as Amazon ECC, Sun Grid
	Storage	Storage – Data storage or cloud based NAS such as CTERA, iDisk, CloudNAS
	Hosting	Hosting – Physical data centers such as those run by IBM, HP, NaviSite, etc.

What is virtualization?

- Run **multiple operating systems** and **user applications** on the same hardware
  - E.g., run both Windows and Linux on the same laptop
- The OSes are completely **isolated** from each other
- Complete control over your own “guest” OS



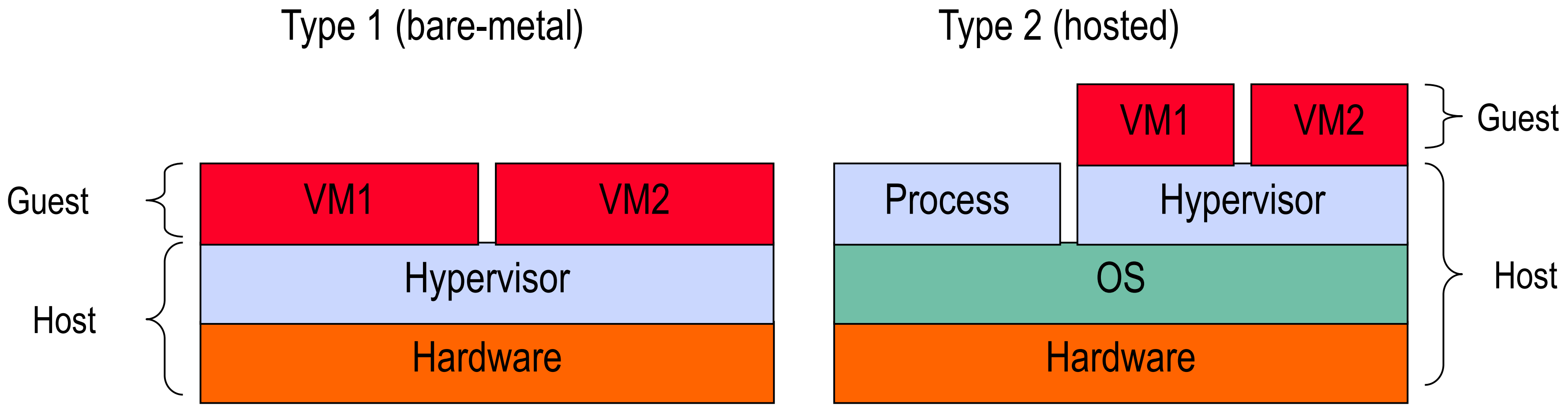


Uses of virtualization

- Server consolidation
  - Run a **web server** and a **mail server** on the **same physical server**
- Easier development
  - Develop critical **operating system components** (file system, disk driver) without affecting **computer stability**
- Quality Assurance
  - Testing a network product (e.g., a firewall) may require **tens of computers**
  - Try testing thoroughly a product at each pre-release milestone
- Cloud computing
  - Buy computing as a service
  - You pay for e.g., 2 CPU cores for 3 hours plus 10GB of network traffic

Two Types of Virtualization

- Definitions
  - **Hypervisor** (or **VMM** – Virtual Machine Monitor) is a software layer that allows several **virtual machines** to **run** on a **physical machine**
  - Physical OS and hardware are called the **Host**
  - Virtual machine OS and applications are called the **Guest**



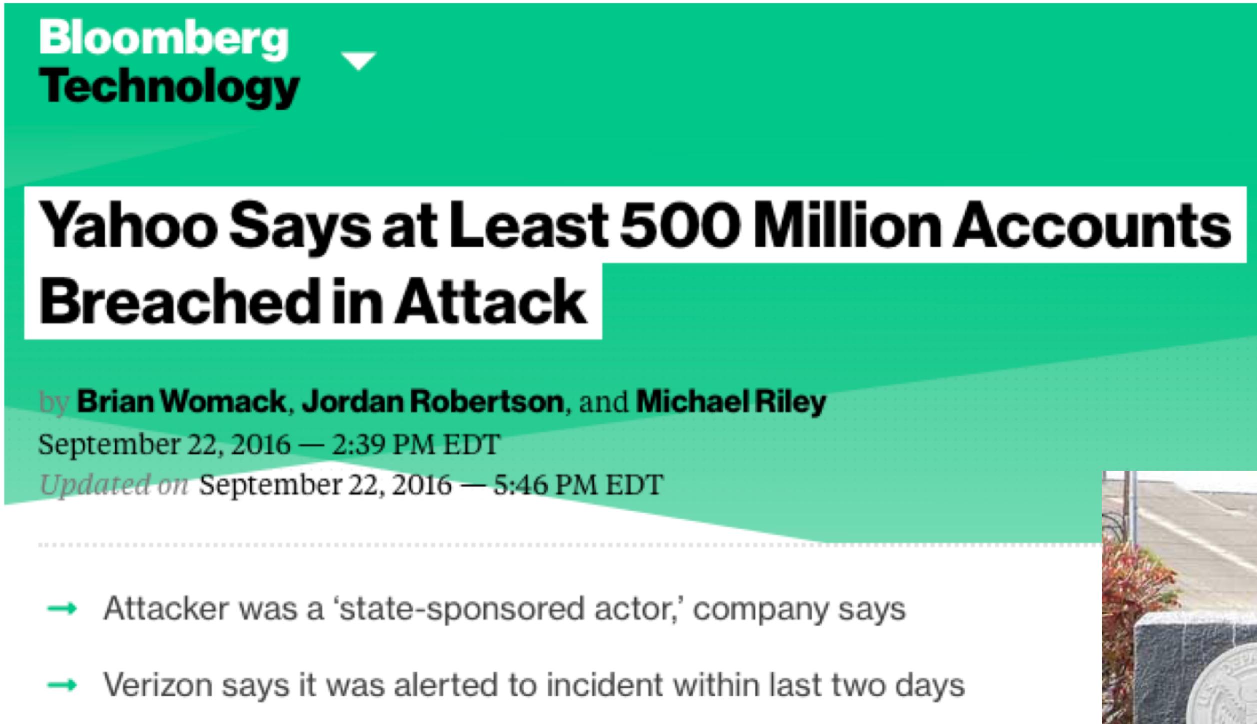
VMware ESX, Microsoft Hyper-V, Xen

VMware Workstation, Microsoft Virtual PC, Sun VirtualBox, QEMU, KVM

How to run a VM (also called a container)?

- Emulate
  - Interpret every guest instruction, real slow
  - E.g., BOCHS
- Dynamic binary translation
  - Most code will run native (like JAVA JIT)
  - Sensitive code will call the hypervisor (trapping into host)
  - E.g., Vmware, QEMU
- Paravirtualization
  - Modified guest OS works directly with the hypervisor
  - E.g., Xen

Hundreds of Data Breaches in 2016


Bloomberg Technology

Yahoo Says at Least 500 Million Accounts Breached in Attack

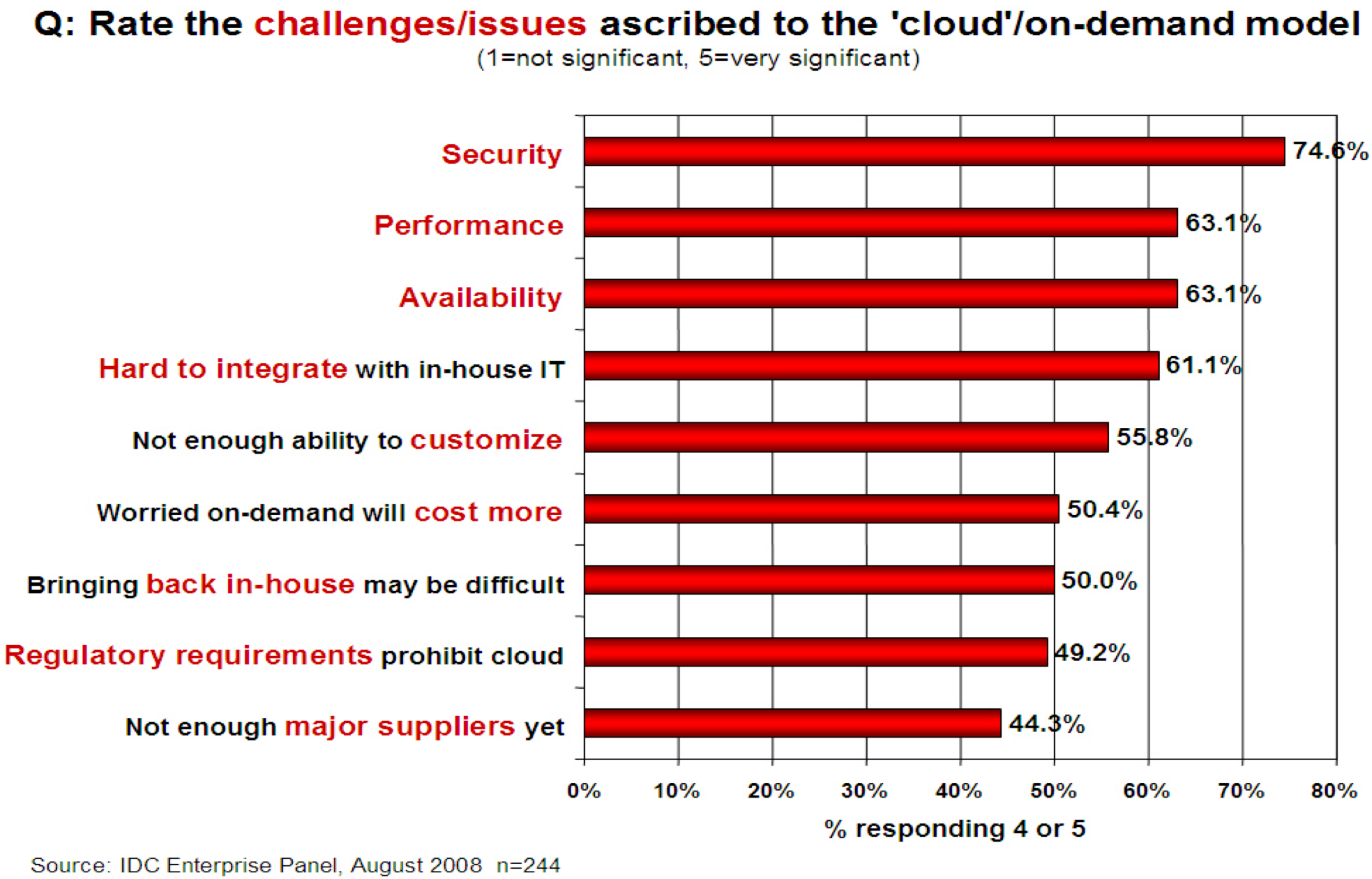
by **Brian Womack**, **Jordan Robertson**, and **Michael Riley**
September 22, 2016 — 2:39 PM EDT
Updated on September 22, 2016 — 5:46 PM EDT

- Attacker was a 'state-sponsored actor,' company says
- Verizon says it was alerted to incident within last two days

Seagate

verizon

Cloud Security



Advantages vs. Challenges

- ✓ Reduced exposure
- ✓ Auditing/testing
- ✓ Automatic management
- ✓ Redundancy
- ✓ Disaster recovery

- ✗ Trusting vendors
- ✗ Accountability
- ✗ Opaque technologies
- ✗ Loss of physical control

source: Peter Mell, Tim Grance, NIST, Information Technology Laboratory, 2009

www.nist.gov

End of Safe Harbor

U.S. ★ EU
SAFEHARBOR
U.S. DEPARTMENT OF COMMERCE

Administration fédérale > PFPDT

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Suite de l'arrêt concernant l'accord «Safe Harbor»: indications utiles pour la transmission de données aux États-Unis

Born: 2000
RIP: 6.10.2015

No longer a valid legal basis in CH

Need new legal frameworks for IT!

Digital Sovereignty

Microsoft Windows Server System

Yesterday: IT Products → **Today+: IT Services**

- ◆ Bought server & software
- ◆ Local usage (in office/building)
- ◆ Governed privately
- ✓ Digital Sovereignty

- ▪ Cloud services
- ▪ Global resources
- ▪ Governed by country
- ✗ Loss of Sovereignty

Technologies & legal frameworks to enable transition?

Implications for Switzerland

- #1 ICT spend per capita [World Bank, 2013]
  - 7.2% of GDP with strong consumer and enterprise spend

- 70% service-based economy
  - Top ten knowledge-based

- Living in private clouds with legacy technologies is a risk

Major stake holder in cloud technologies!

Summary

- We now live in a Digital Universe
- Technology roadmap ➔ Centralization
- Clouds & datacenters are the only path forward
  - Leverage massive data analytics
  - Benefit from economies of scale
- Challenges
  - Technologies to scale platforms
  - Frameworks to guarantee sovereignty