CS 110
Computer Architecture

Course Summary

Instructor:
Sören Schwertfeger

https://robotics.shanghaitech.edu.cn/courses/ca

School of Information Science and Technology SIST

ShanghaiTech University

Slides based on UC Berkley's CS61C

Meltdown and Spectre

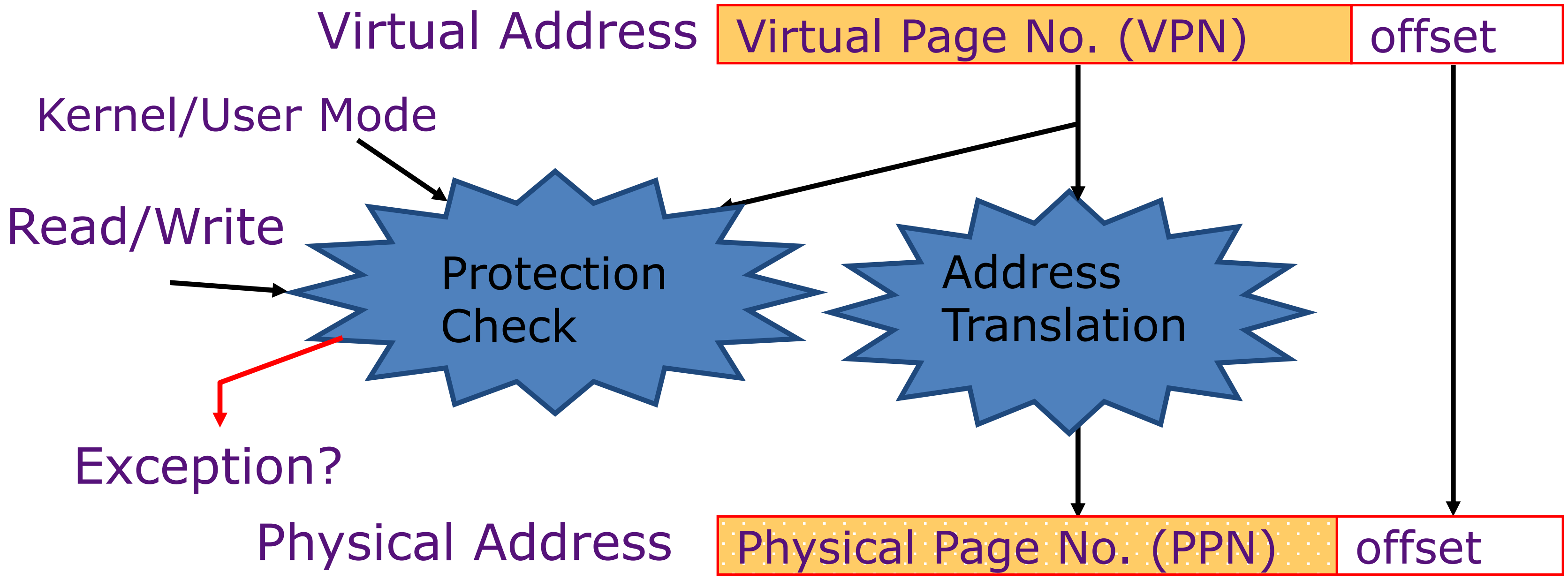
- Hardware vulnerability
- Affecting Intel x86 microprocessors, IBM POWER processors, and some ARM-based microprocessors
- All Operating Systems effected!
- They are considered "catastrophic" by security analysts!
- Allow to read all memory (e.g. from other process or other Virtual Machines (e.g. other users data on Amazon cloud service!))
- Towards the end of this CA course you can understand the basics of how Meltdown and Spectre work. Keywords:
  - Virtual Memory; Protection Levels; Instruction Pipelining; Speculative Execution; CPU Caching;

Meltdown & Spectre

	Meltdown	Spectre
Allows kernel memory read	Yes	No
Was patched with KAISER/KPTI	Yes	No
Leaks arbitrary user memory	Yes	Yes
Could be executed remotely	Sometimes	Definitely
Most likely to impact	Kernel integrity	Browser memory
Practical attacks against	Intel	Intel, AMD, ARM

- KAISER = KPTI: Kernel page-table isolation
- Disclaimer: Most details that follow are oversimplified!!!

VM: Address Translation & Protection

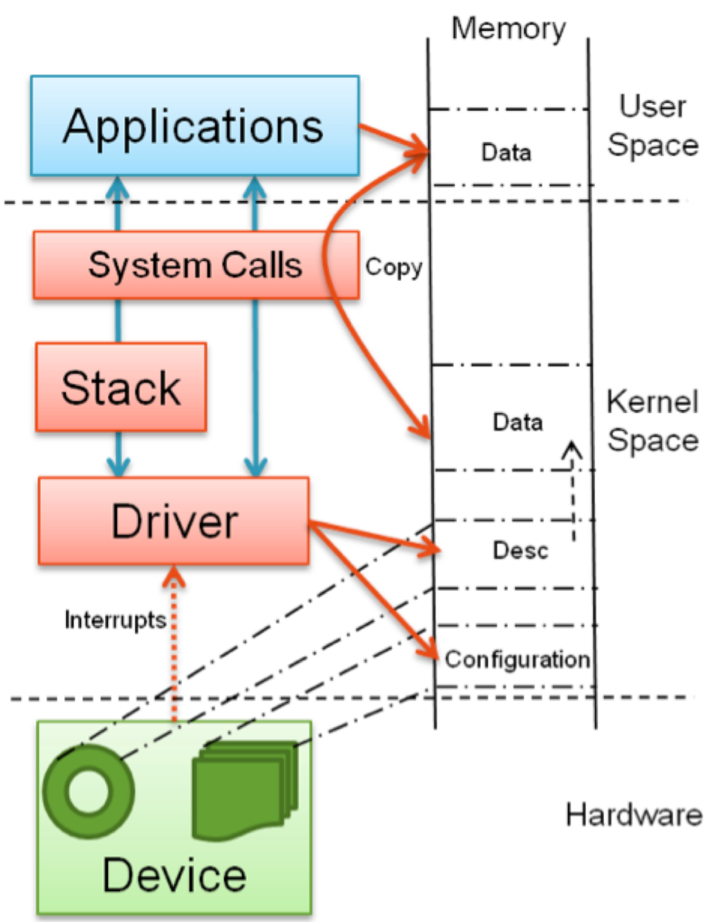


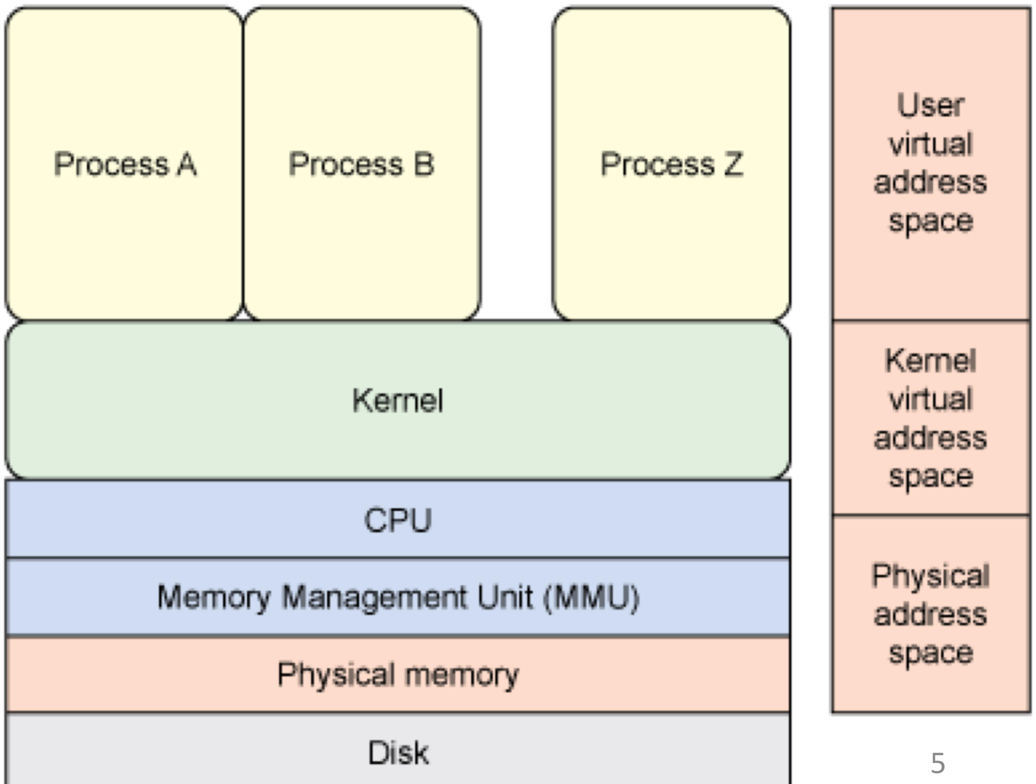
- Every instruction and data access needs address translation and protection checks

A good VM design needs to be fast (~ one cycle) and space efficient

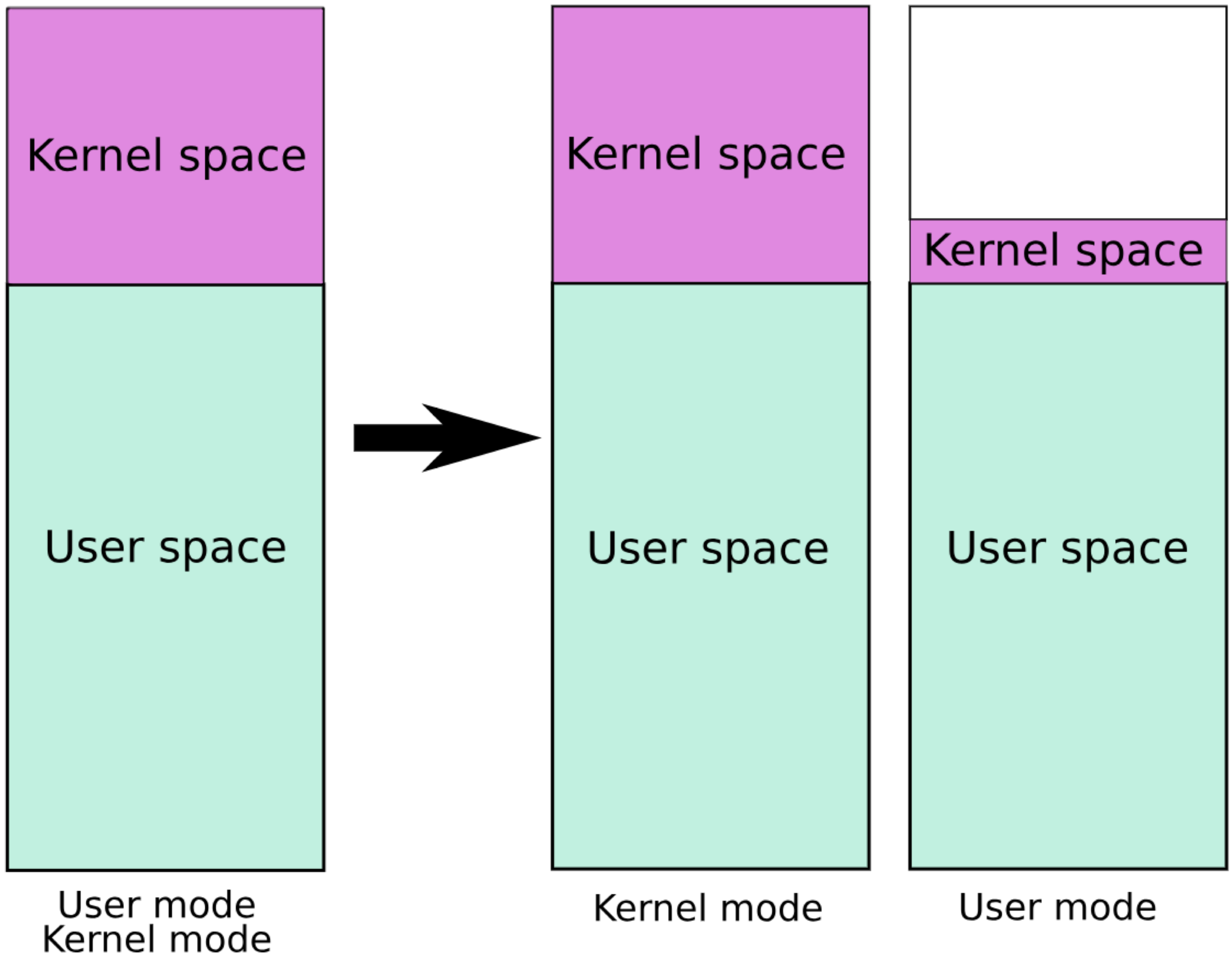
OS: Kernel Memory Space

- User processes have memory pages in the kernel space (managed by kernel, but with user data, e.g. network package received)





Kernel page-table isolation



KPTI: Meltdown only!

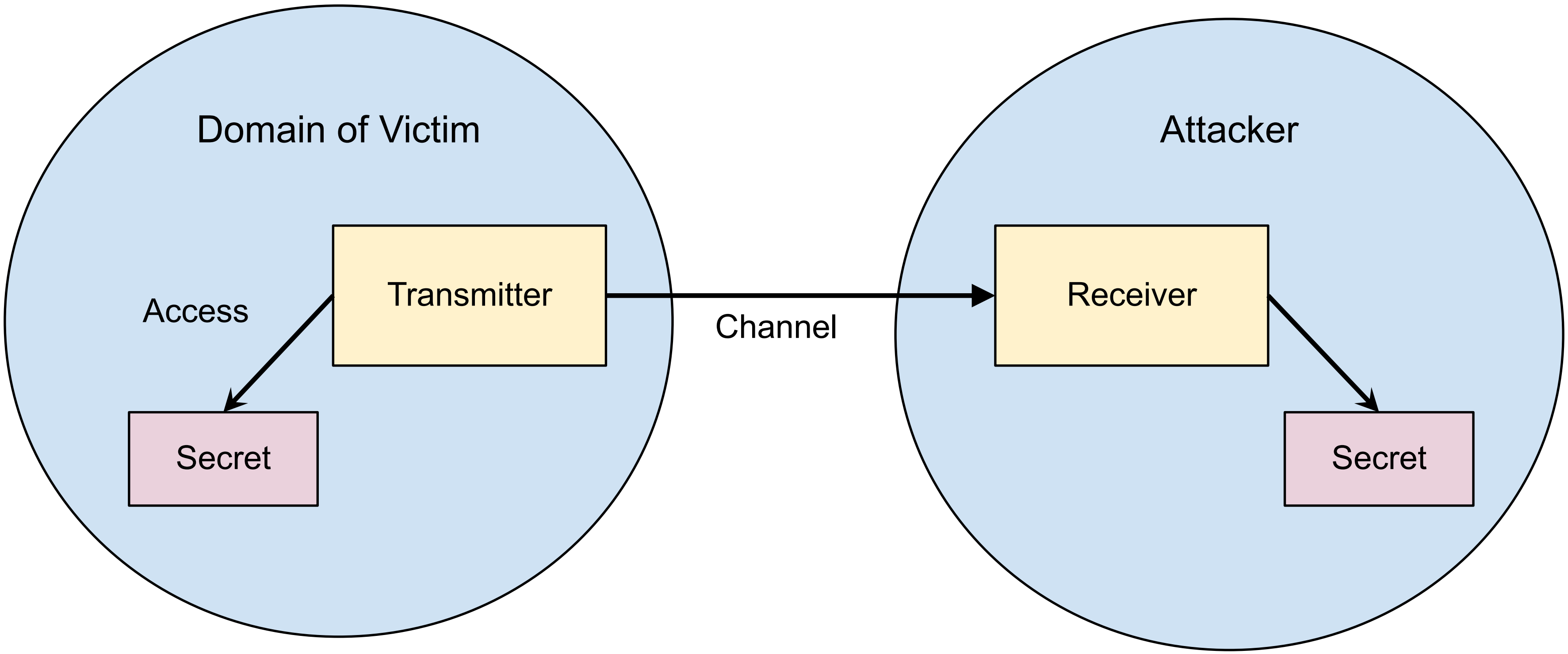
- Without KPTI:
  - Executing user-space code (applications), Linux keeps entire kernel memory mapped in page tables (but protected from access)
  - Advantage: System call into the kernel or Interrupt: kernel page tables are always present => most context-switching overheads (TLB flush, page-table swapping, etc.) can be avoided!
- With KPTI: 5% - 30% slower (depending on workload: more syscalls (e.g. Databases) slower)

Three Cve’s

- Common Vulnerabilities and Exposures (**CVE**) system provides a reference-method for publicly known information-security vulnerabilities and exposures

- CVE-2017-5715 - aka Spectre, branch target injection
- CVE-2017-5753 - aka Spectre, bounds check bypass
- CVE-2017-5754 - aka Meltdown, rogue data cache load, memory access permission check performed after kernel memory read

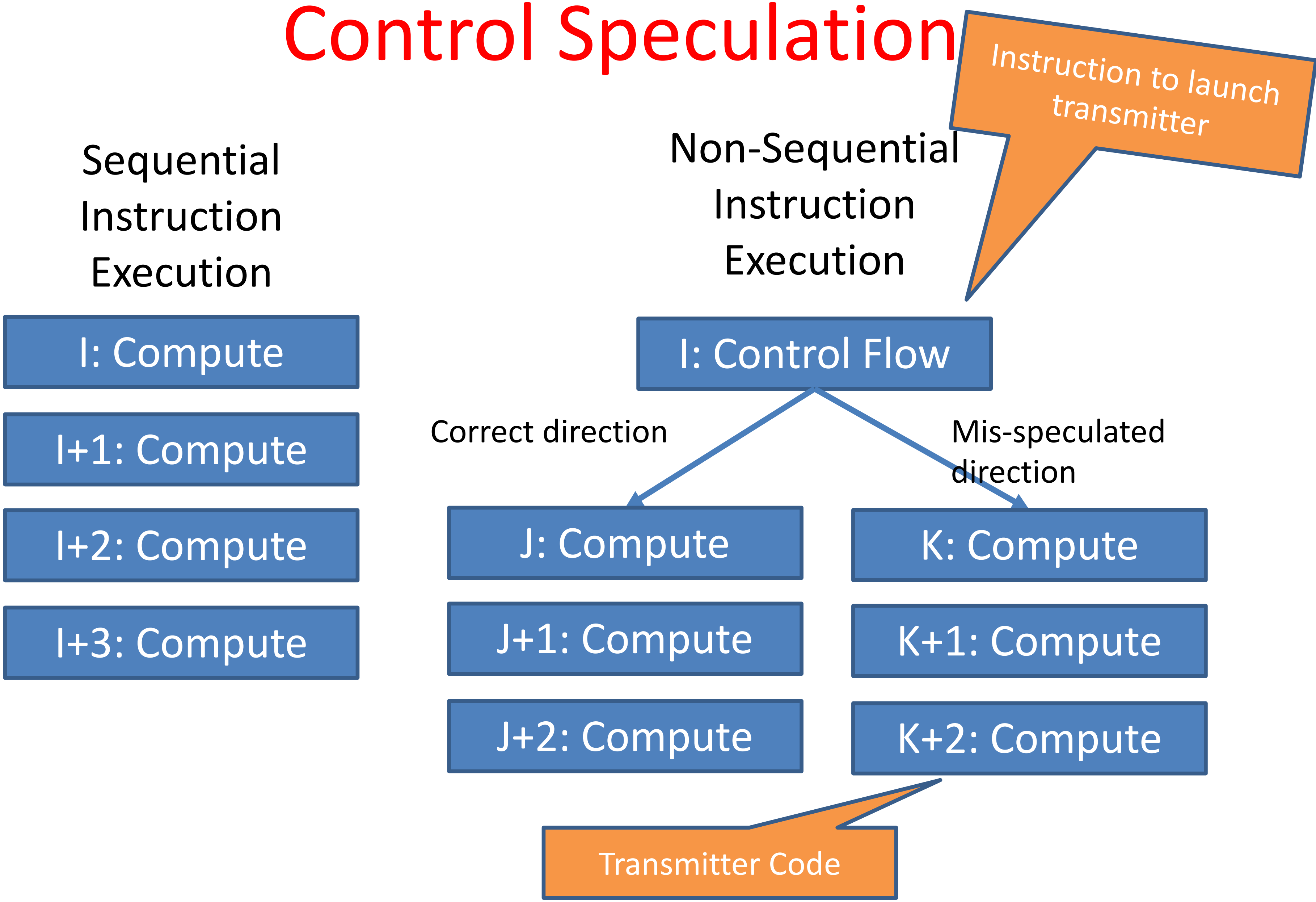
Attack Schema



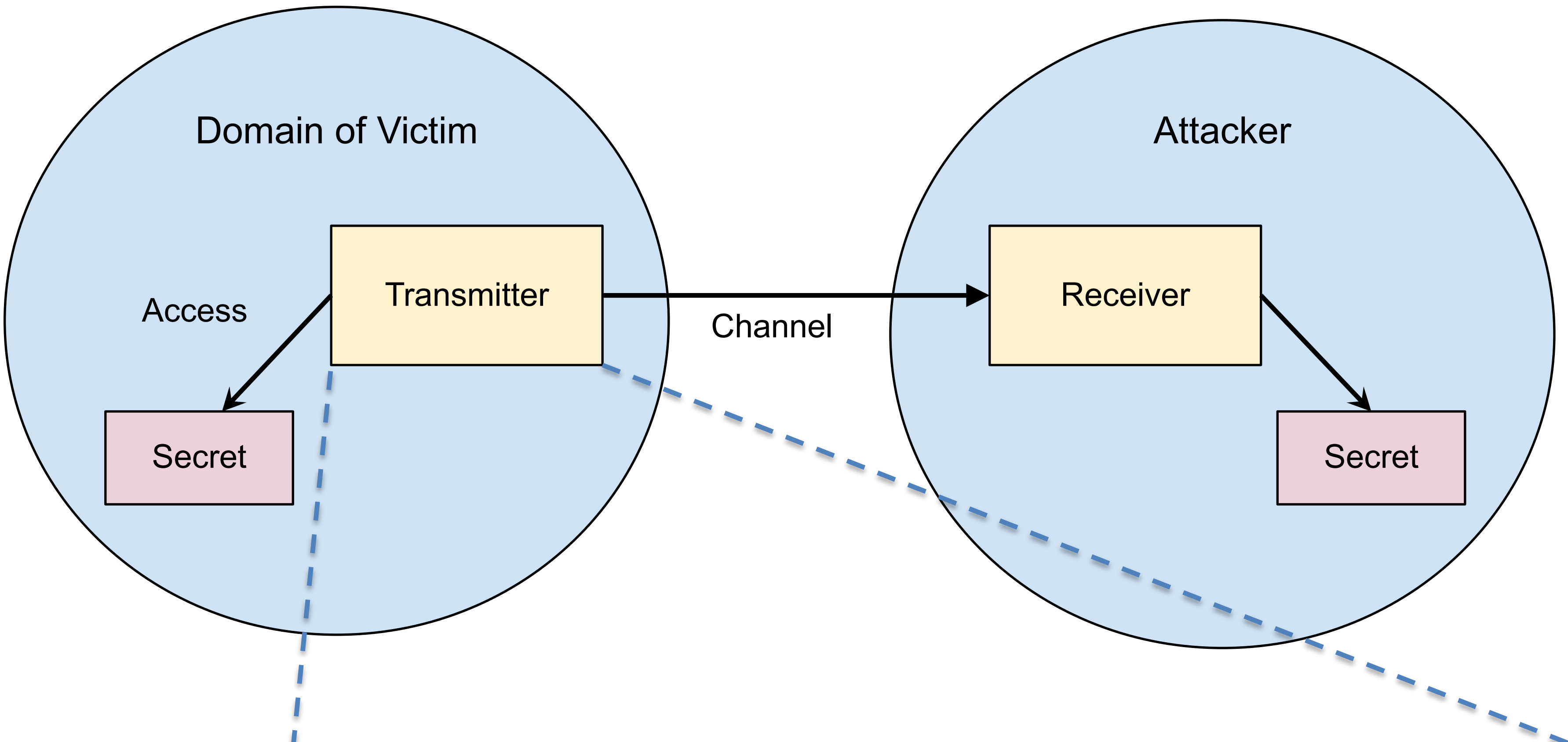
1. Create a channel
2. Create the transmitter
3. Launch the transmitter
4. Access the secret

Material from MIT: Adam Belay, Srini Devadas, Joel Emer

Control Speculation



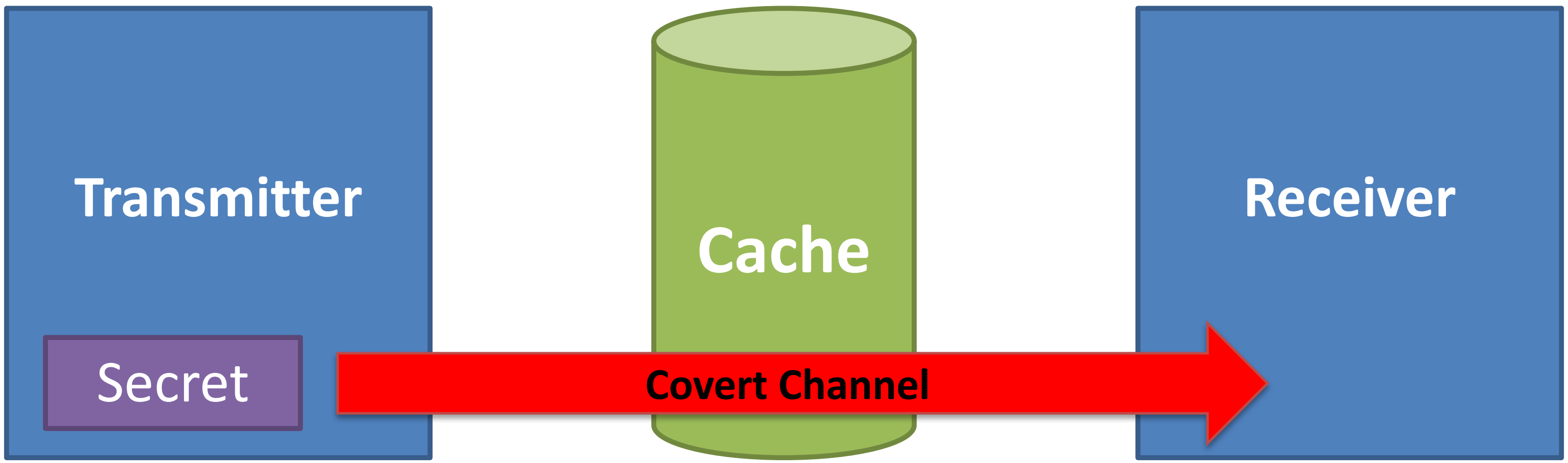
Building a Transmitter



Pre-existing (RSA example)
Written by attacker (Meltdown)
Synthesized out of existing victim code by attacker (Spectre style)

Meltdown and Spectre Attack Examples

Attack: Mis-speculation exfiltrates secrets through cache



Speculative Execution

Side Channel

Normal Execution

Meltdown

Problem: Attacker can influence speculative control flow

Bug: Speculative execution not subject to page permission checks

Attack: User code can read kernel data (secret)

Three steps:

1. Setup: flush the cache
2. Transmit: force speculation that depends on secret
3. Receive: measure cache timings

Meltdown example

Setup:

```
clflush(timing_ptr[guess]);
```

Transmit:

```
timing_ptr[*kernel_addr];
```

Pointing to secret data!

Page Fault

May still read *kernel_addr (speculatively)

Receive:

```
mfence();
s = rdtsc(); *timing_ptr[guess];
e = rdtscp();
if (e - s < CACHE_MISS_THRESHOLD)
  printf("guess was right!\n");
```

Code explained

- clflush(ptr): Cache Line Flush (remove from $)
- mfence(): in out-of-order processors ensure that all prior memory operations have been finished
- X86: Time Stamp Counter (TSC) 64-bit register: number of clock cycles since reset
  - rdtsc(): read TSC
  - rdtscp(): read TSC NOW (without out-of-order re-ordering)

Spectre

- Problem: Attacker can influence speculative control flow (same as before)
- Attack: Exfiltrate secrets within a process address space (e.g. a web browser). Can also be used to attack the kernel.
- Could use attacker provided code (JIT) or could co-opt existing program code
- Same three steps! Different setup and transmitters.

Spectre examples

Transmit - Bounds Check Bypass:

```
if (x < array1_size)
  array2[array1[x] * 256];
```

Spectre examples

Transmit - Bounds Check Bypass:

```
if (x < array1_size)
  array2[array1[x] * 256];
```

Transmit - Branch Target Injector:

```
fnptr_t foo = choose_function();
foo(bar);
```

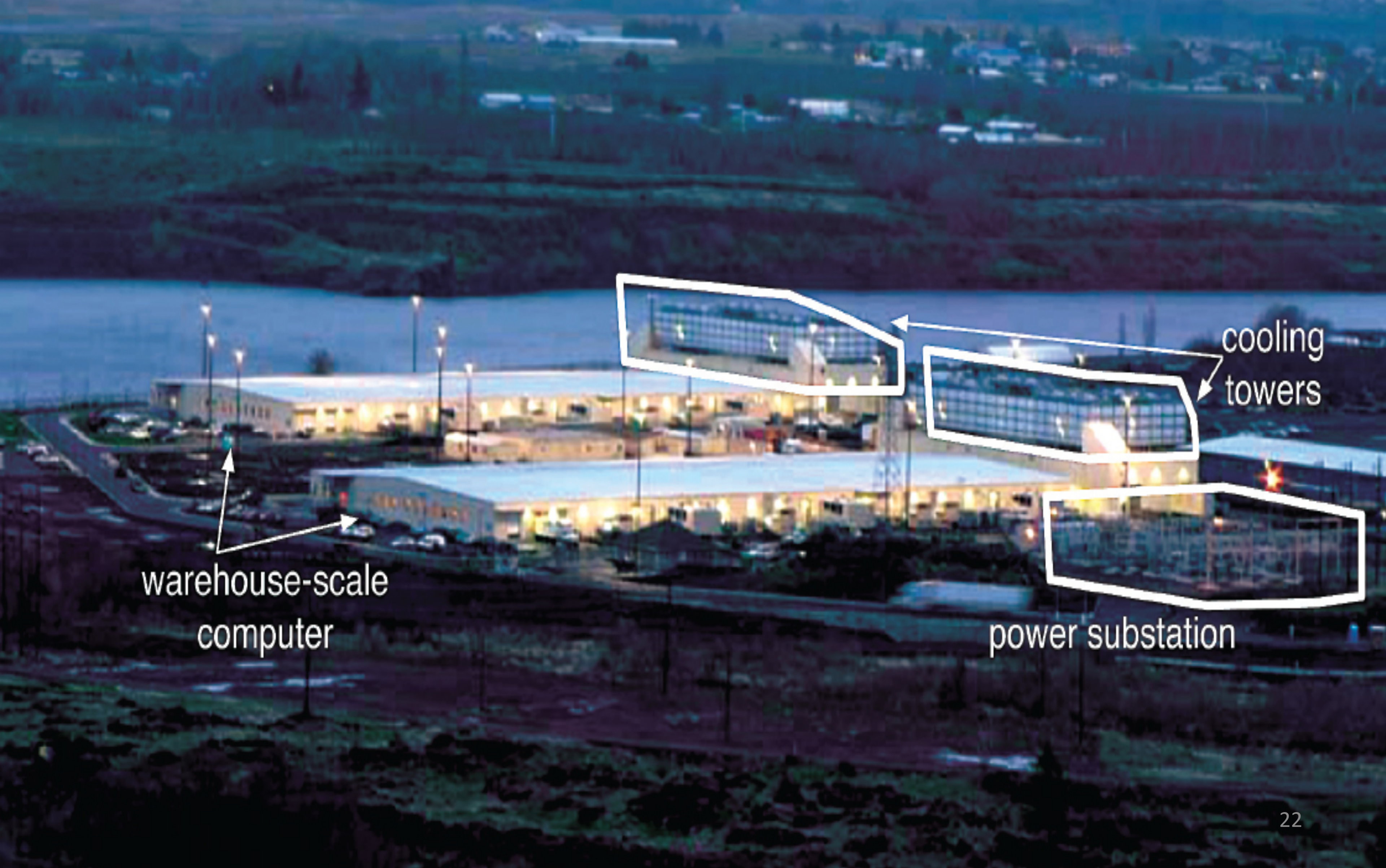
Fixing those bugs

- KPTI for meltdown (speed penalty!)
- Software: Serialize code (no out of order)
- Patches for Operating Systems
- BIOS patches:
  - Patch the firmware of the processors => different micro-code get's executed (microcode fixes)
  - Old processors without patches, e.g.:
  - Intel processors that will never get updates:
    - Bloomfield (2011), Bloomfield Xeon, Clarksfield (2012), Gulftown, Harpertown Xeon C0 and E0, Jasper Forest, Penryn/QC, SoFIA 3GR, Wolfdale (2011), Wolfdale Xeon, Yorkfield (2011), and Yorkfield Xeon.
- Wait for new hardware w/o those bugs…

New School Computer Architecture (1/3)

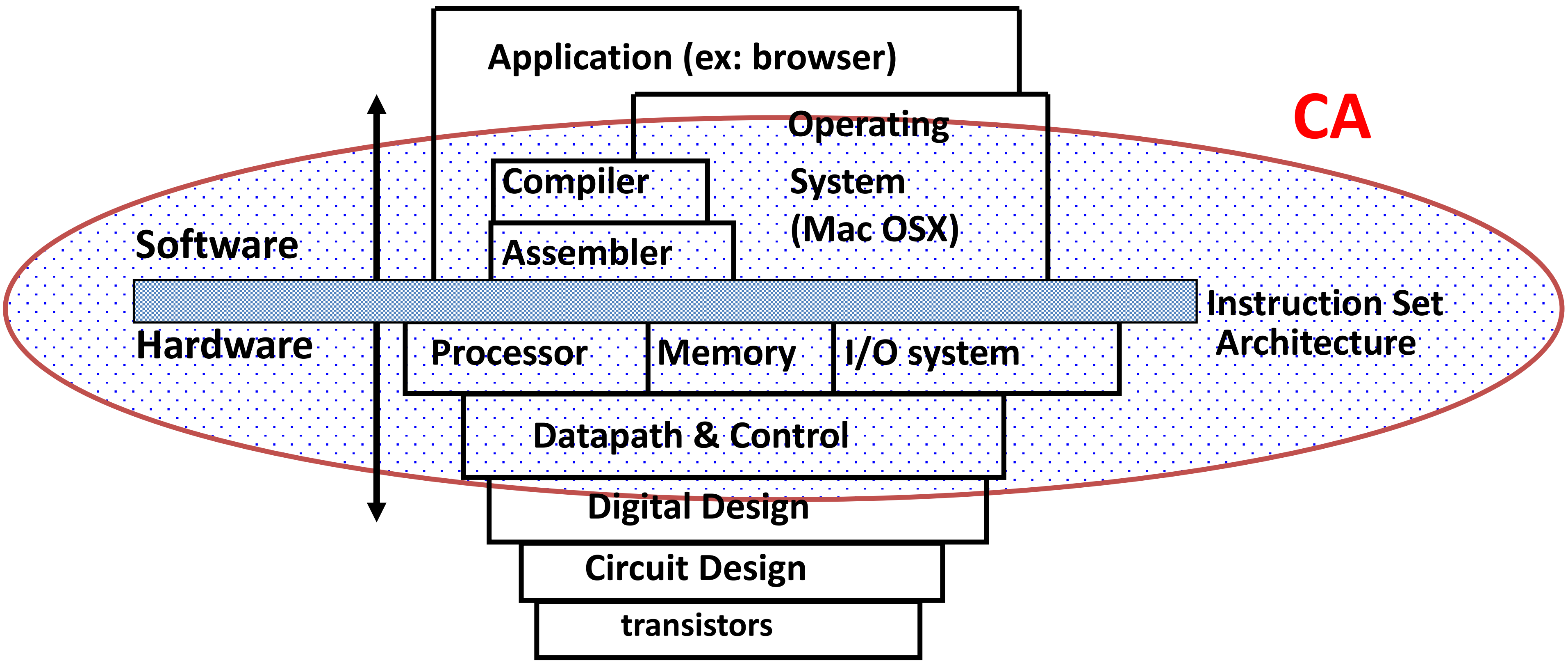
Personal
Mobile
Devices

New School Computer Architecture (2/3)



New School Computer Architecture (3/3)

Old Machine Structures

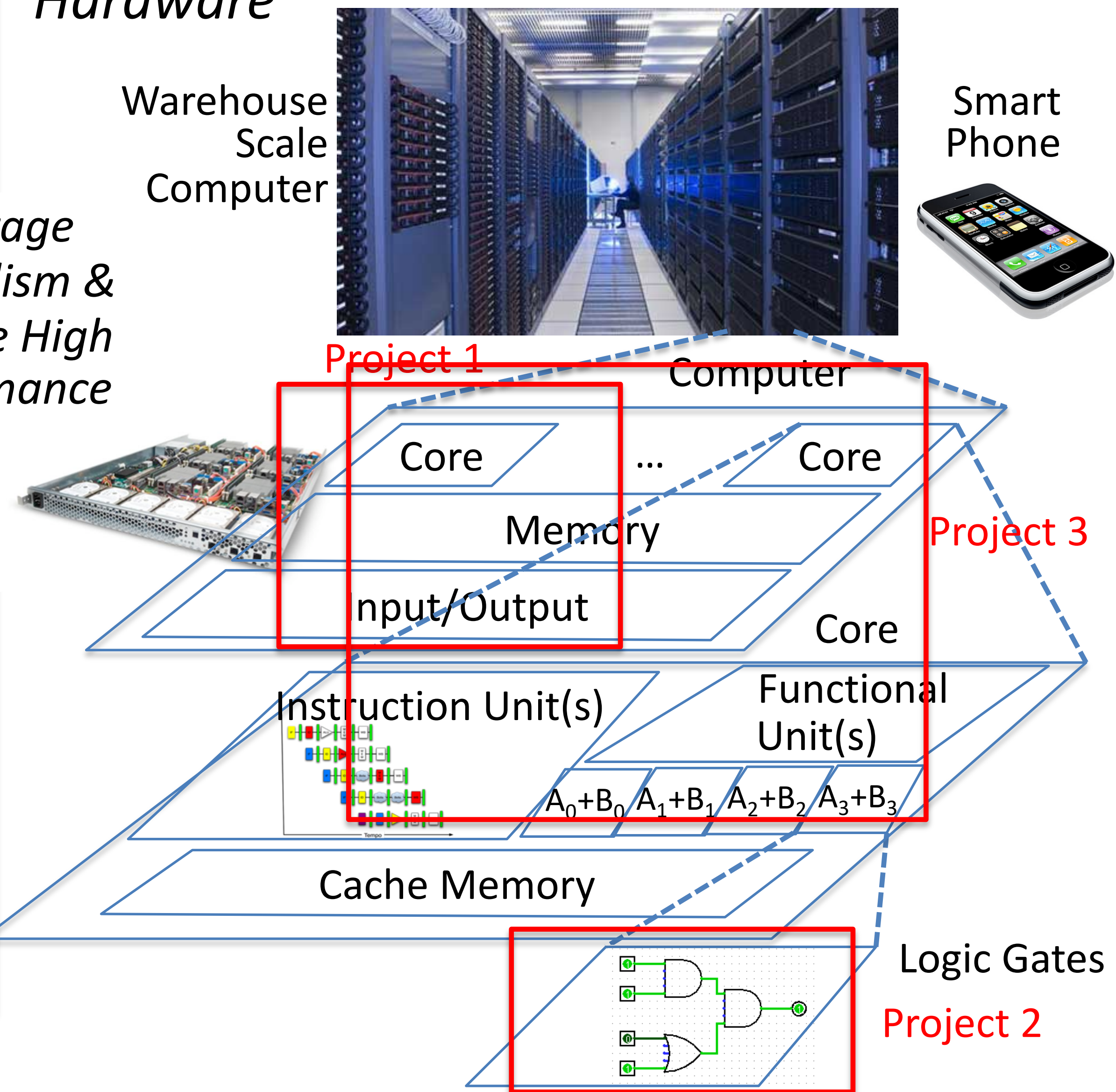


New-School Machine Structures (It’s a bit more complicated!)

Software | *Hardware*

- Parallel Requests
  Assigned to computer
  e.g., Search “Katz”
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates functioning in parallel at same time
- Programming Languages

Leverage Parallelism & Achieve High Performance



CA is NOT about C Programming

- It's about the hardware-software interface
  - What does the programmer need to know to achieve the highest possible performance
- Languages like C are closer to the underlying hardware, unlike languages like Python!
  - Allows us to talk about key hardware features in higher level terms
  - Allows programmer to explicitly harness underlying hardware parallelism for high performance: "programming for performance"

Great Ideas in Computer Architecture

1. Design for Moore's Law
2. Abstraction to Simplify Design
3. Make the Common Case Fast
4. Dependability via Redundancy
5. Memory Hierarchy
6. Performance via Parallelism/Pipelining/Prediction

Powers of Ten inspired CA Overview

- Going Top Down cover 3 Views

1. Architecture (when possible)
2. Physical Implementation of that architecture
3. Programming system for that architecture and implementation (when possible)

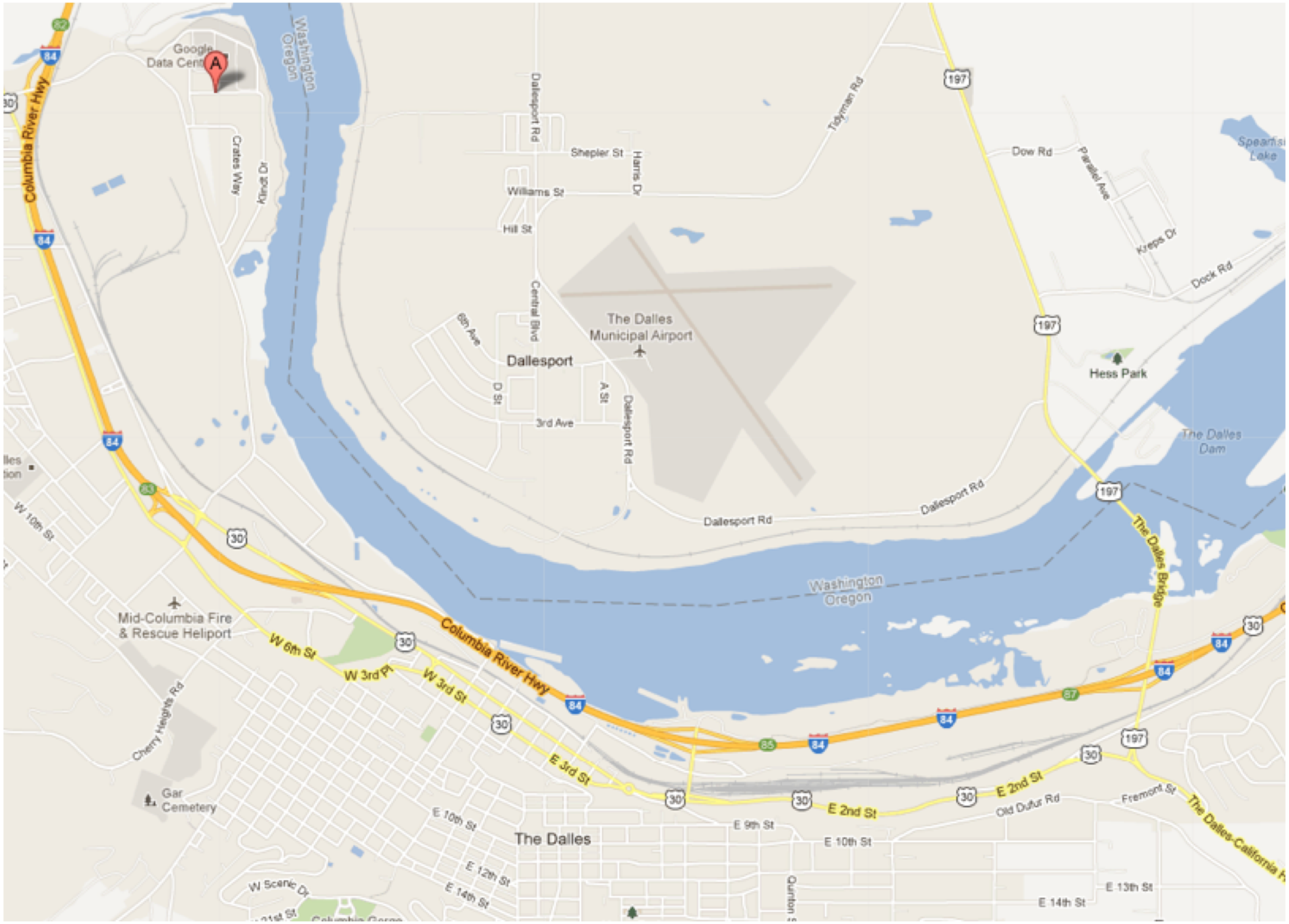
- See http://www.powersof10.com/film

Earth

10^7 meters

The Dalles, Oregon

10^4 meters



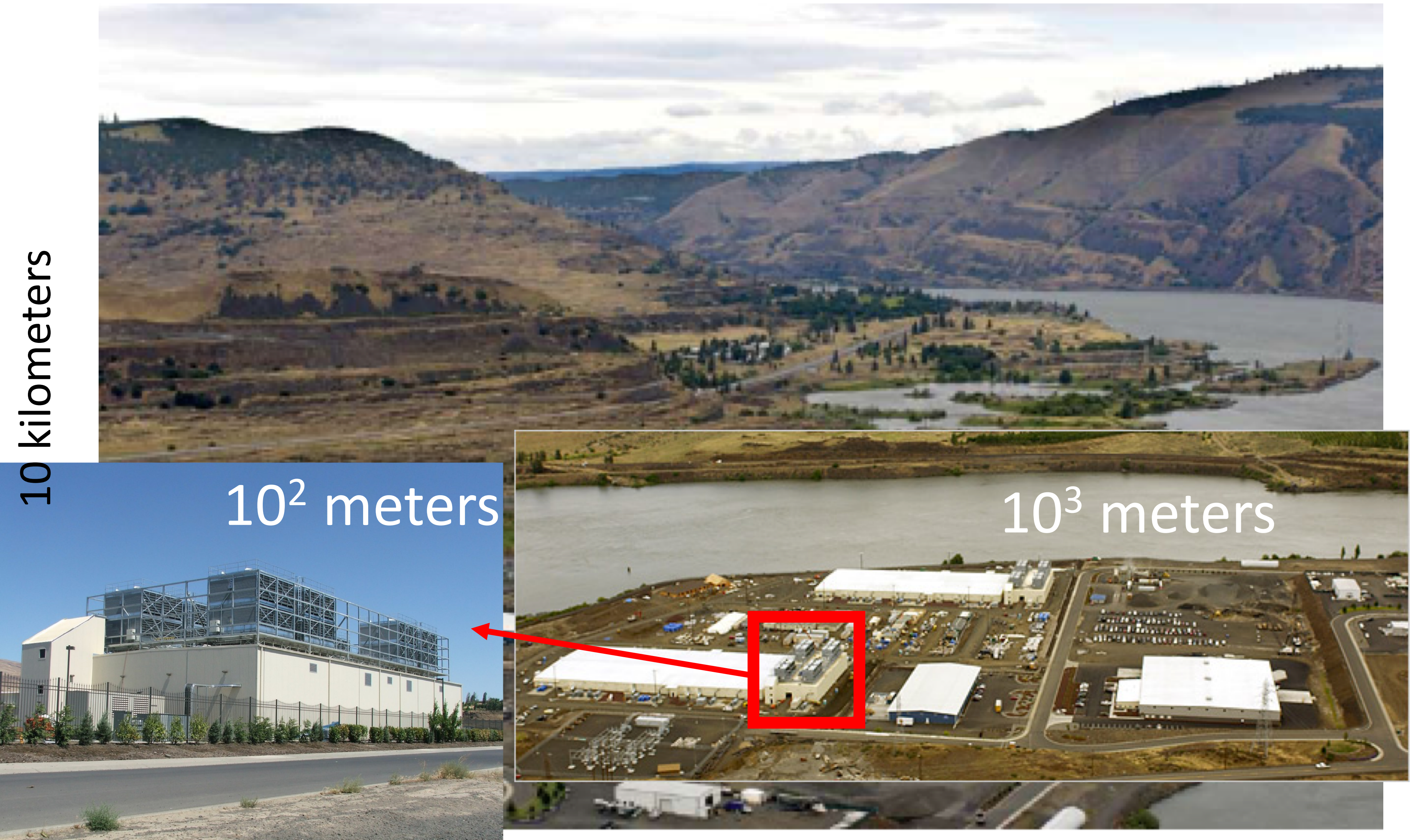
The Dalles, Oregon

10^4 meters

Google's Oregon WSC 10^3 meters

10^4 meters

Google's Oregon WSC



Google Warehouse

- 90 meters by 75 meters, 10 Megawatts
- Contains 40,000 servers, 190,000 disks
- Power Utilization Effectiveness: 1.23
  - 85% of 0.23 overhead goes to cooling losses
  - 15% of 0.23 overhead goes to power losses
- Contains 45, 40-foot long containers
  - 8 feet x 9.5 feet x 40 feet
- 30 stacked as double layer, 15 as single layer

Containers in WSCs

10^2 meters

100 meters

Google Container

10^1 meters

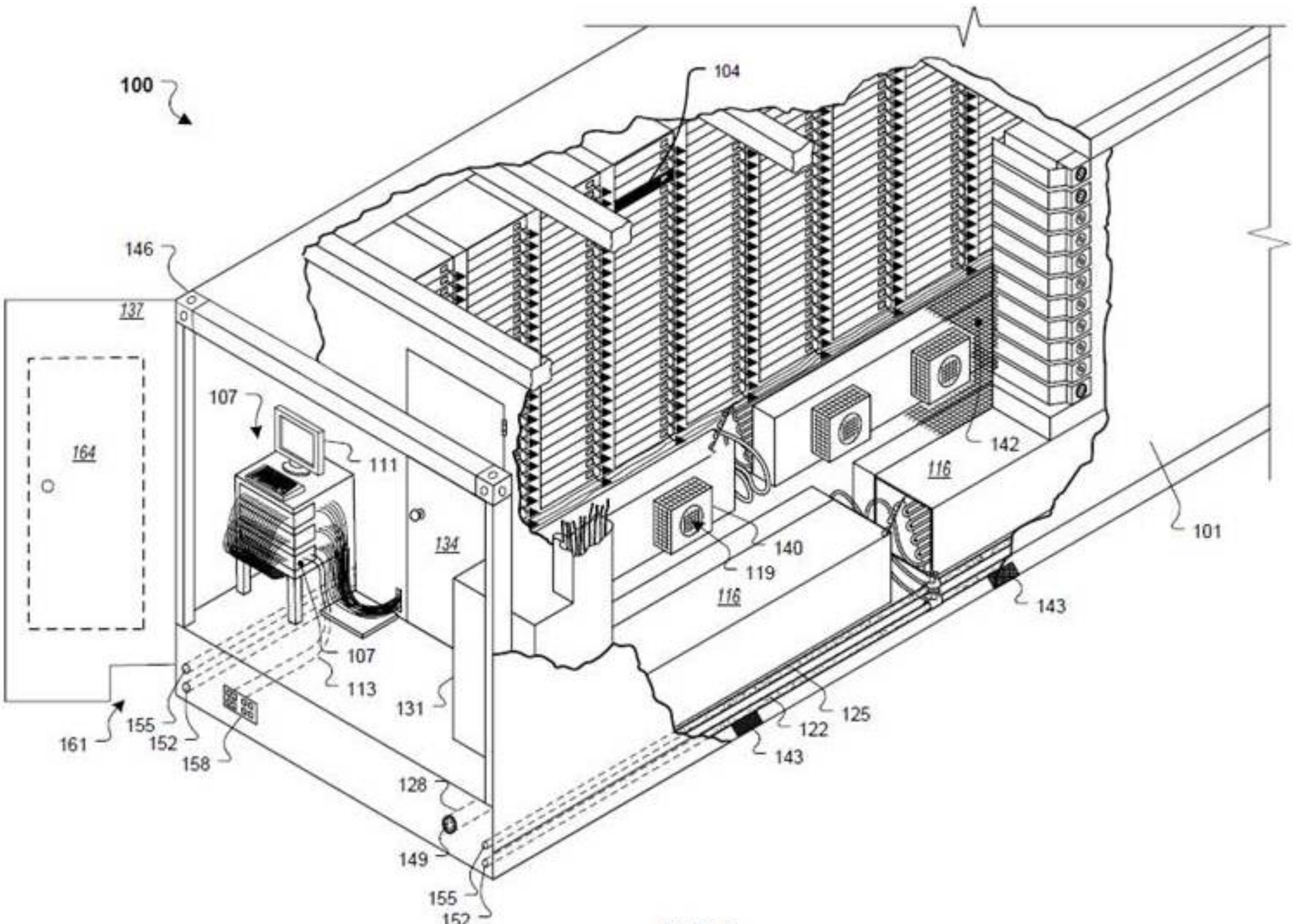
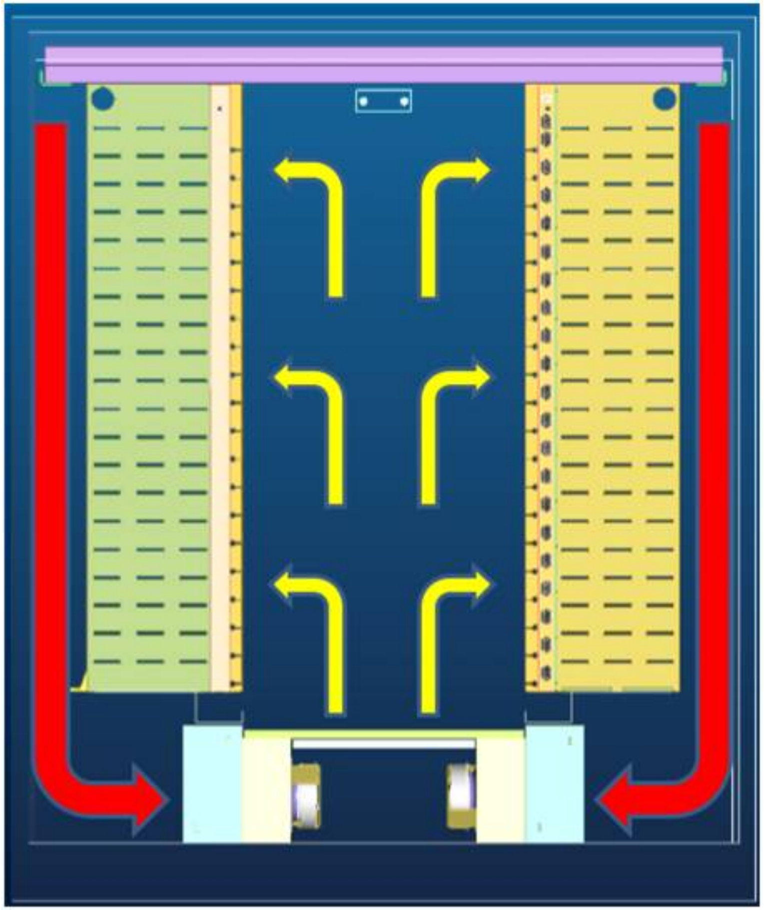


FIG. 1

Google Container

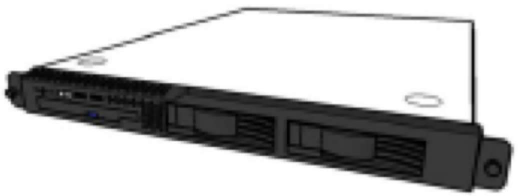
10^0 meters

10 meters



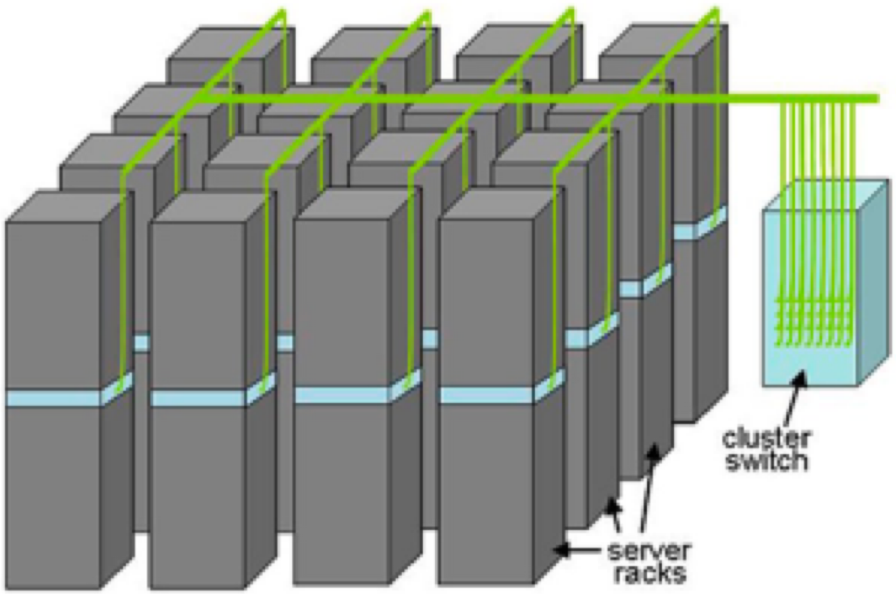
- 2 long rows, each with 29 racks
- Cooling below raised floor
- Hot air returned behind racks

Equipment Inside a Container



Server (in rack format):

7 foot Rack: servers + Ethernet local area network switch in middle ("rack switch")



Array (aka cluster): server racks + larger local area network switch ("array switch") 10X faster => cost 100X: cost $f(N^2)$

Google Rack

- Google rack with 20 servers + Network Switch in the middle
- 48-port 1 Gigabit/sec Ethernet switch every other rack
- Array switches connect to racks via multiple 1 Gbit/s links
- 2 datacenter routers connect to array switches over 10 Gbit/s links

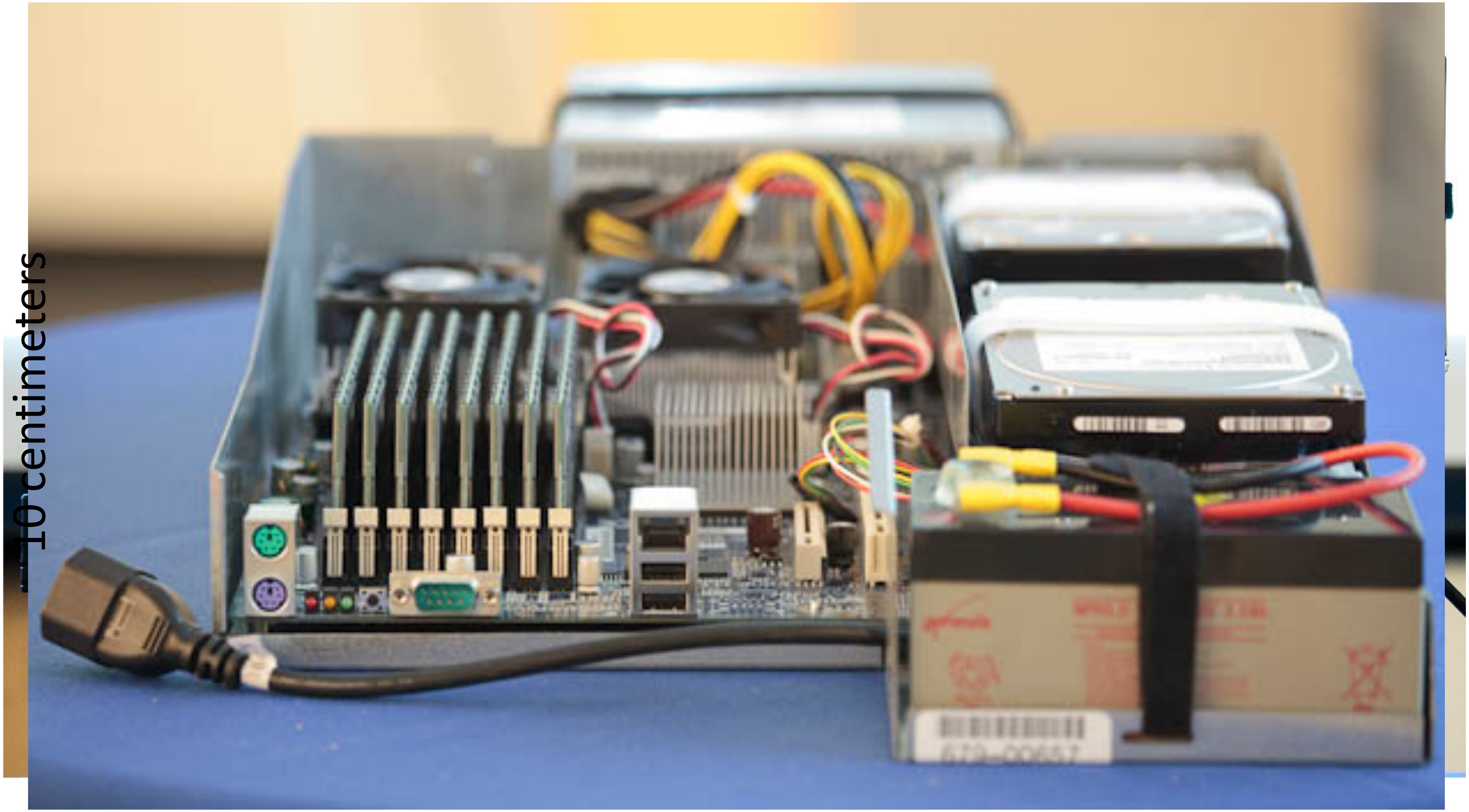
1 meter

Great Ideas in Computer Architecture

1. *Design for Moore's Law*
   -- WSC, Container, Rack
2. Abstraction to Simplify Design
3. Make the Common Case Fast
4. *Dependability via Redundancy*
   -- Multiple WSCs, Multiple Racks, Multiple Switches
5. Memory Hierarchy
6. *Performance via Parallelism/Pipelining/Prediction*
   -- Task level Parallelism, Data Level Parallelism

Google Server Internals

10^{-1} meters



Google Board Details

- Supplies only 12 volts
- Battery per board vs. large battery room
  - Improves PUE: 99.99% efficient local battery vs 94% for battery room
- 2 SATA Disk Drives
  - 1 Terabyte capacity each
  - 3.5 inch disk drive
  - 7200 RPM
- 2 AMD Opteron Microprocessors
  - Dual Core, 2.2 GHz
- 8 DIMMs
  - 8 GB DDR2 DRAM
- 1 Gbit/sec Ethernet Network Interface Card

Programming Multicore Microprocessor: OpenMP

```
#include <omp.h>
#include <stdio.h>
static long num_steps = 100000;
int value[num_steps];
int reduce()
{    int i;    int sum = 0;
#pragma omp parallel for private(x) reduction(+:sum)
     for (i=1; i<= num_steps; i++){
        sum = sum + value[i];
     }
}
```

Great Ideas in Computer Architecture

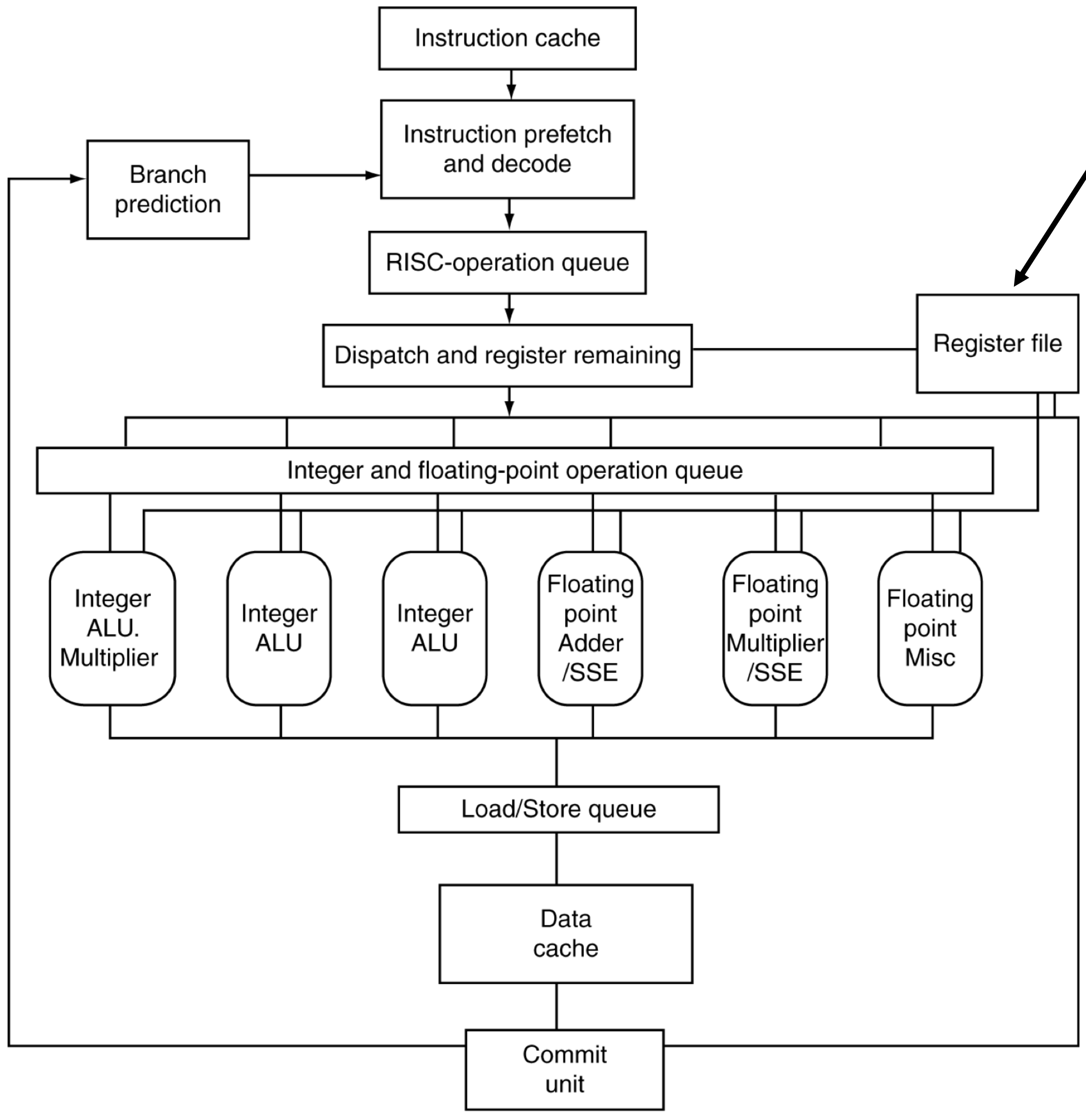
1. *Design for Moore's Law*
   -- More transistors = Multicore + SIMD
2. Abstraction to Simplify Design
3. Make the Common Case Fast
4. Dependability via Redundancy
5. *Memory Hierarchy*
   -- More transistors = Cache Memories
6. *Performance via Parallelism/Pipelining/ Prediction*
   -- Thread-level Parallelism

10^{-2} meters

AMD Opteron Microprocessor

centimeters

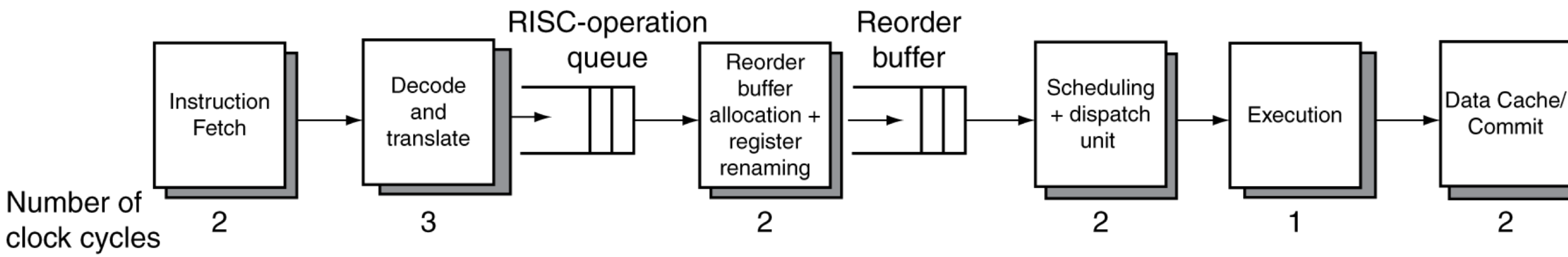
AMD Opteron Microarchitecture



72 physical registers

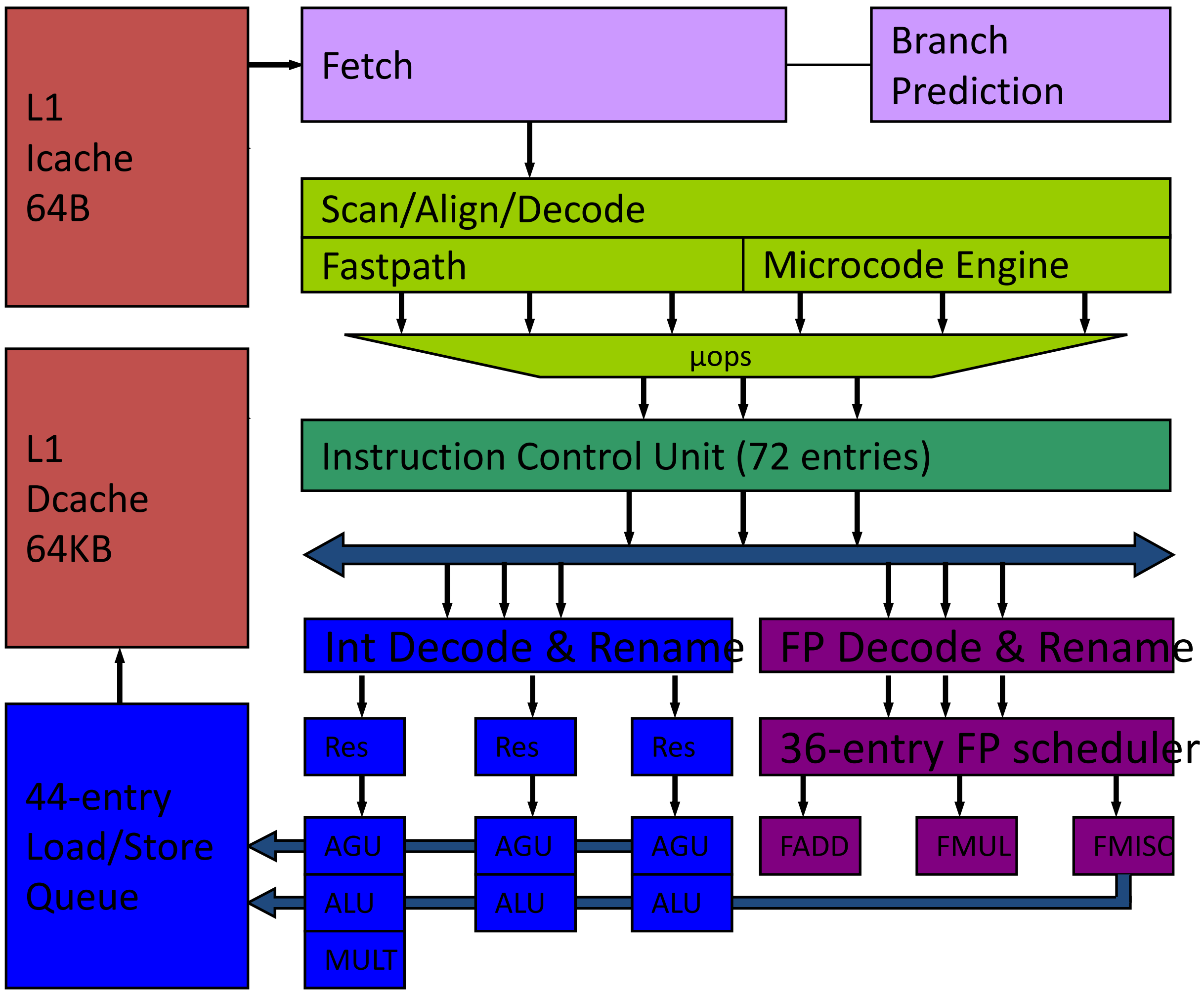
AMD Opteron Pipeline Flow

- For integer operations



  - 12 stages (Floating Point is 17 stages)
  - Up to 106 RISC-ops in progress

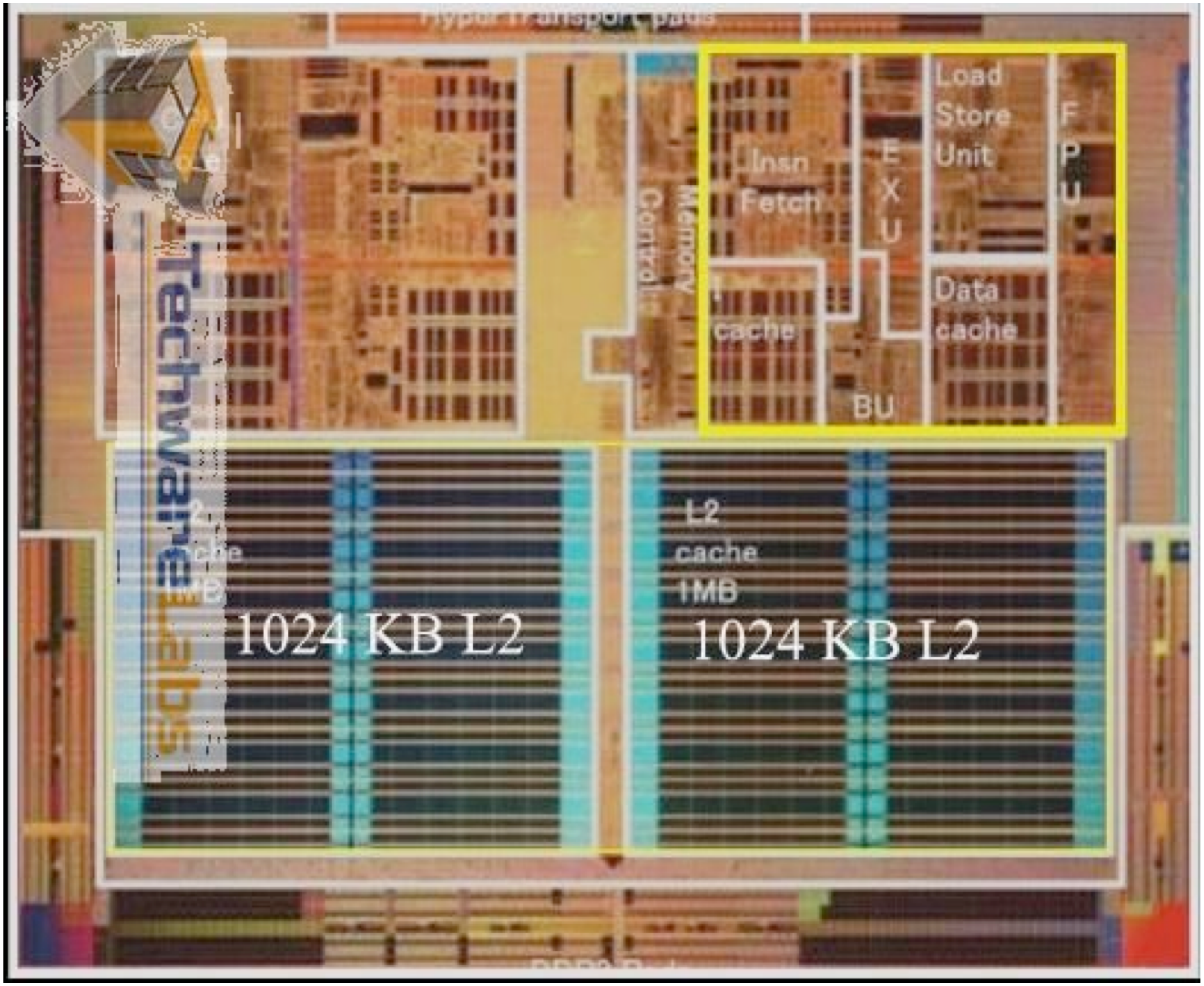
AMD Opteron Block Diagram



10^{-2} meters

AMD Opteron Microprocessor

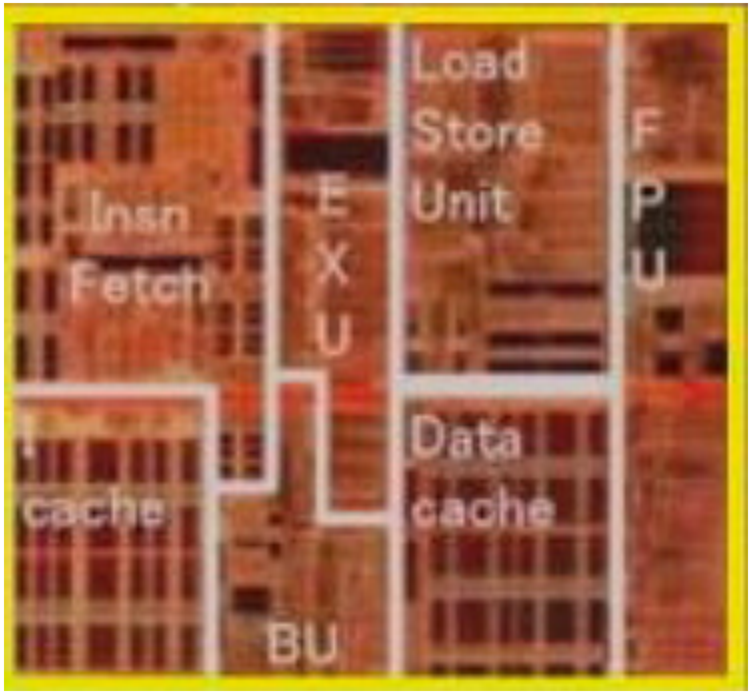
centimeters



10^{-3} meters

AMD Opteron Core

millimeters



Programming One Core: C with Intrinsics

```
void mmult(int n, float *A, float *B, float *C)
{
 for ( int i = 0; i < n; i+=4 )
  for ( int j = 0; j < n; j++ )
  {
   __m128 c0 = _mm_load_ps(C+i+j*n);
   for( int k = 0; k < n; k++ )
    c0 = _mm_add_ps(c0, _mm_mul_ps(_mm_load_ps(A+i+k*n),
                                   _mm_load1_ps(B+k+j*n)));
   _mm_store_ps(C+i+j*n, c0);
  }
}
```

Inner loop from gcc –O -S

Assembly snippet from innermost loop:

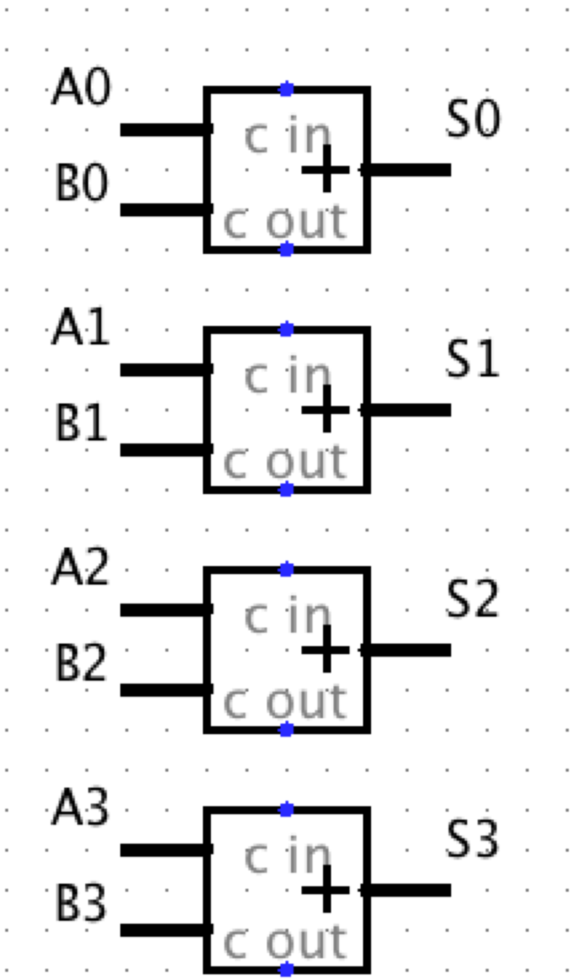
```
movaps  (%rax), %xmm9
mulps   %xmm0, %xmm9
addps   %xmm9, %xmm8
movaps  16(%rax), %xmm9
mulps   %xmm0, %xmm9
addps   %xmm9, %xmm7
movaps  32(%rax), %xmm9
mulps   %xmm0, %xmm9
addps   %xmm9, %xmm6
movaps  48(%rax), %xmm9
mulps   %xmm0, %xmm9
addps   %xmm9, %xmm5
```

Great Ideas in Computer Architecture

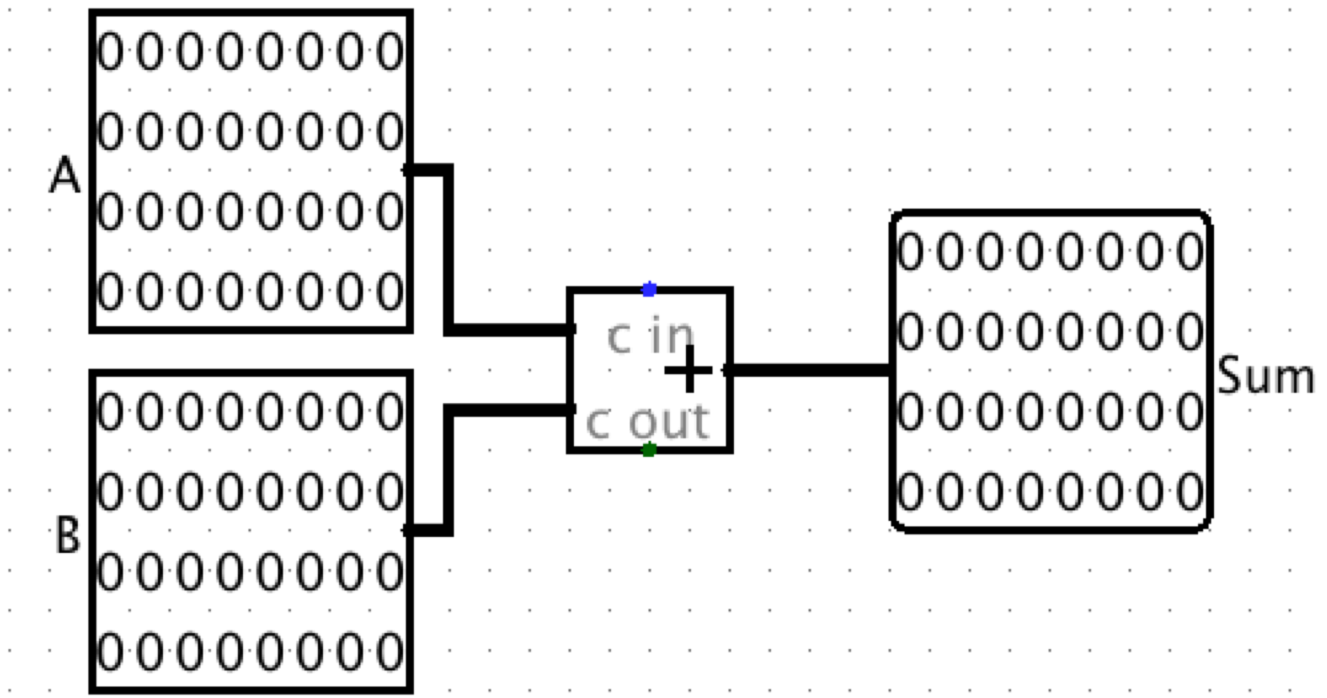
1. Design for Moore's Law
2. *Abstraction to Simplify Design*
   -- Instruction Set Architecture, Micro-operations
3. Make the Common Case Fast
4. Dependability via Redundancy
5. Memory Hierarchy
6. *Performance via Parallelism/Pipelining/Prediction*
   -- Instruction-level Parallelism (superscalar, pipelining)
   -- Data-level Parallelism

SIMD Adder

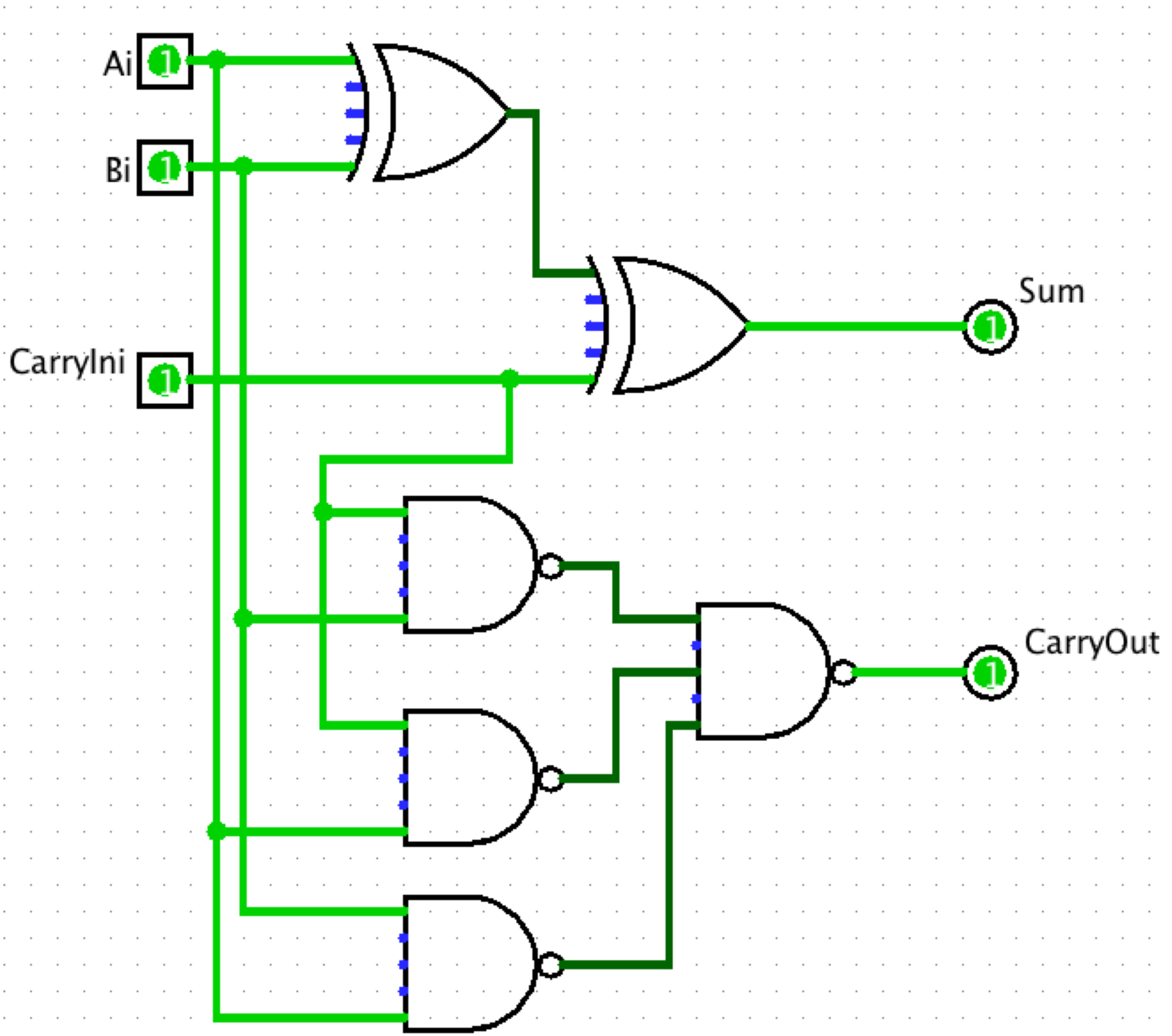
- Four 32-bit adders that operate in parallel
  - Data Level Parallelism



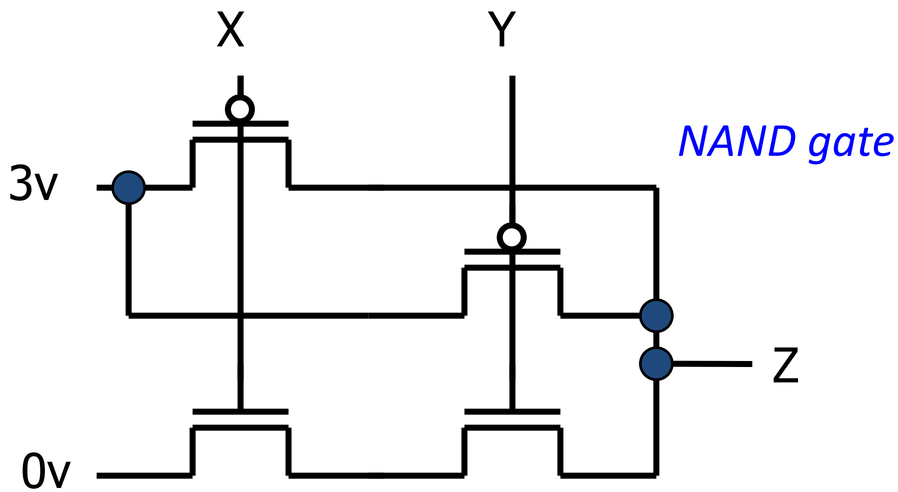
One 32-bit Adder



1 bit of 32-bit Adder



Complementary MOS Transistors (NMOS and PMOS) of NAND Gate

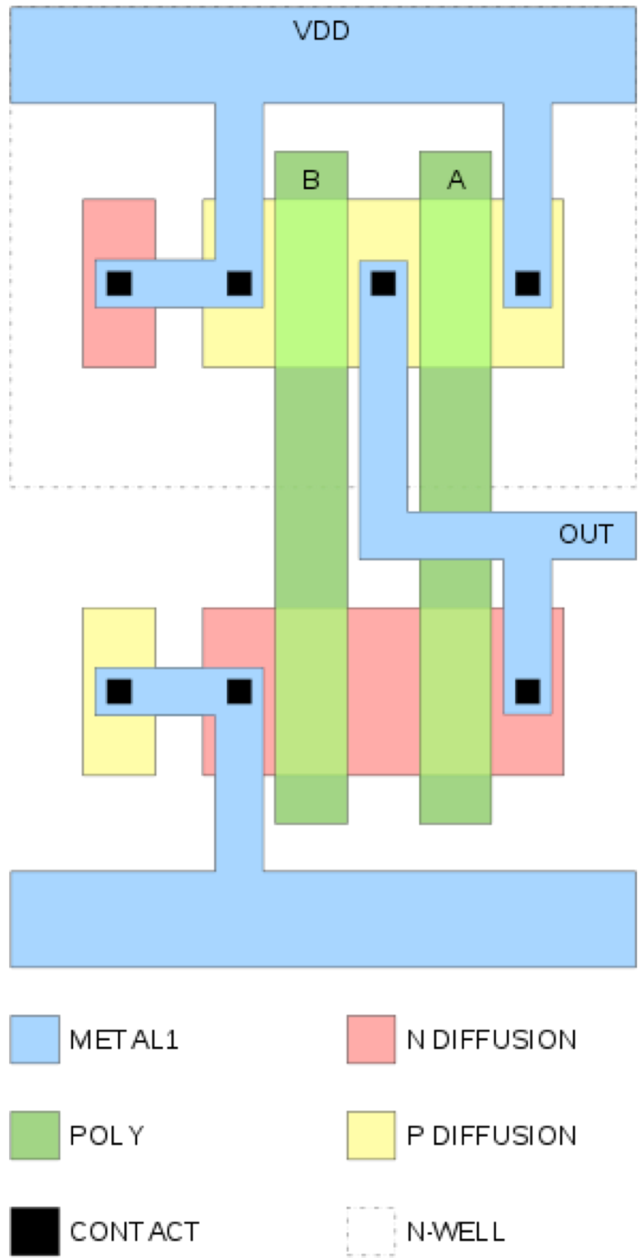


x	y	z
0 volts	0 volts	3 volts
0 volts	3 volts	3 volts
3 volts	0 volts	3 volts
3 volts	3 volts	0 volts

10^{-7} meters

Physical Layout of NAND Gate

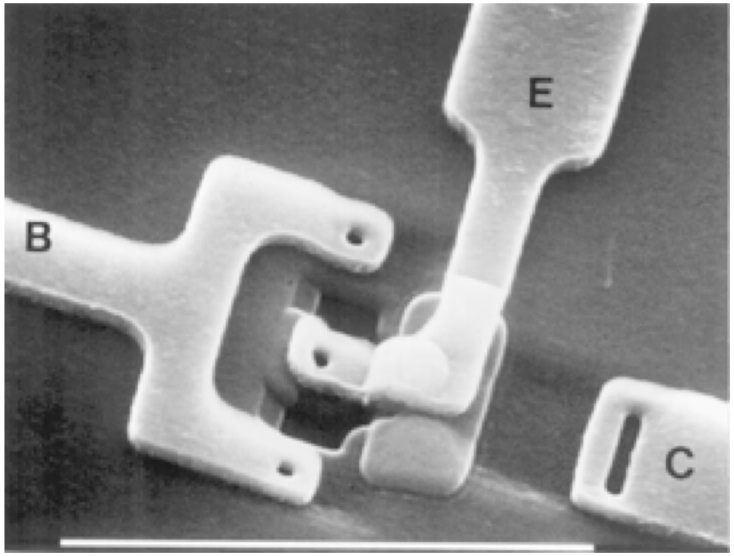
100 nanometers



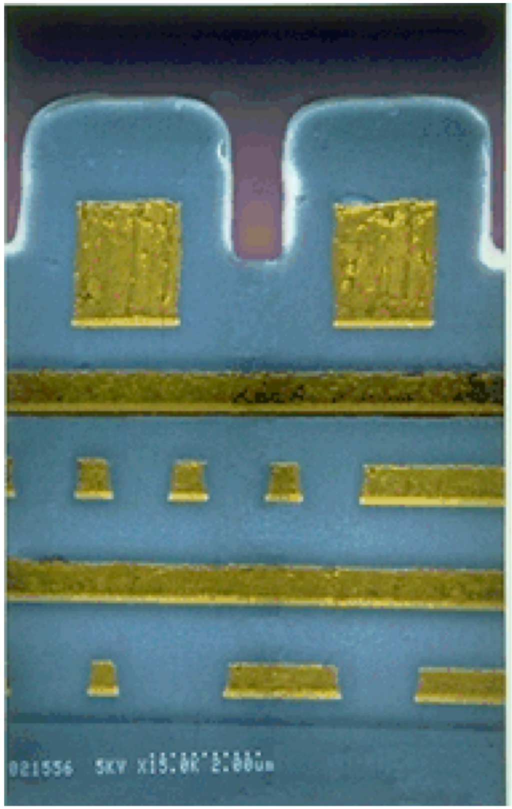
10^{-7} meters

Scanning Electron Microscope

100 nanometers



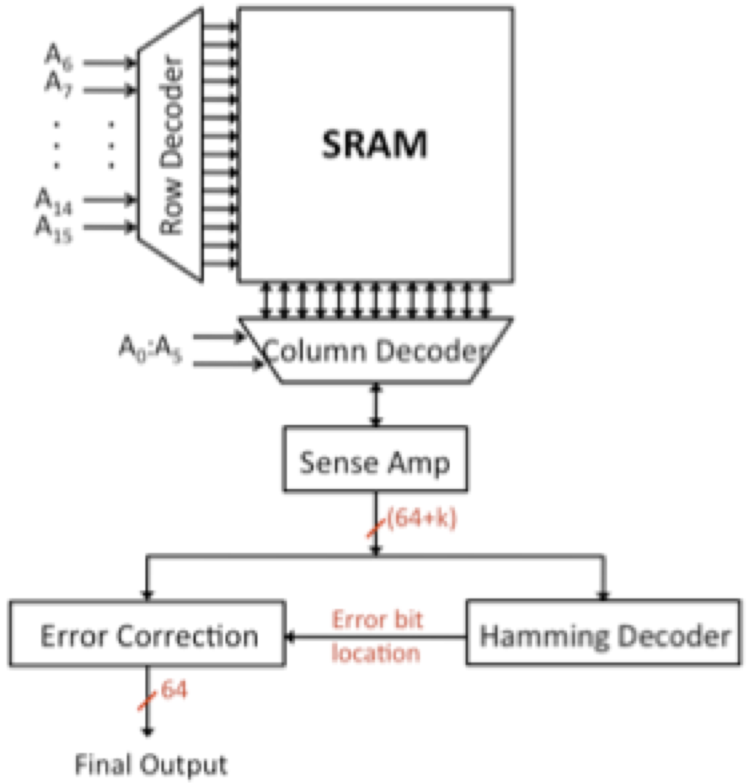
Top View



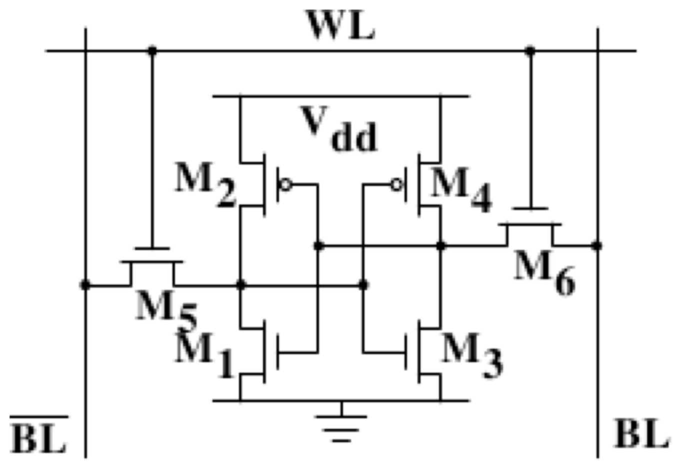
Cross Section

10^{-6} meters

Block Diagram of Static RAM



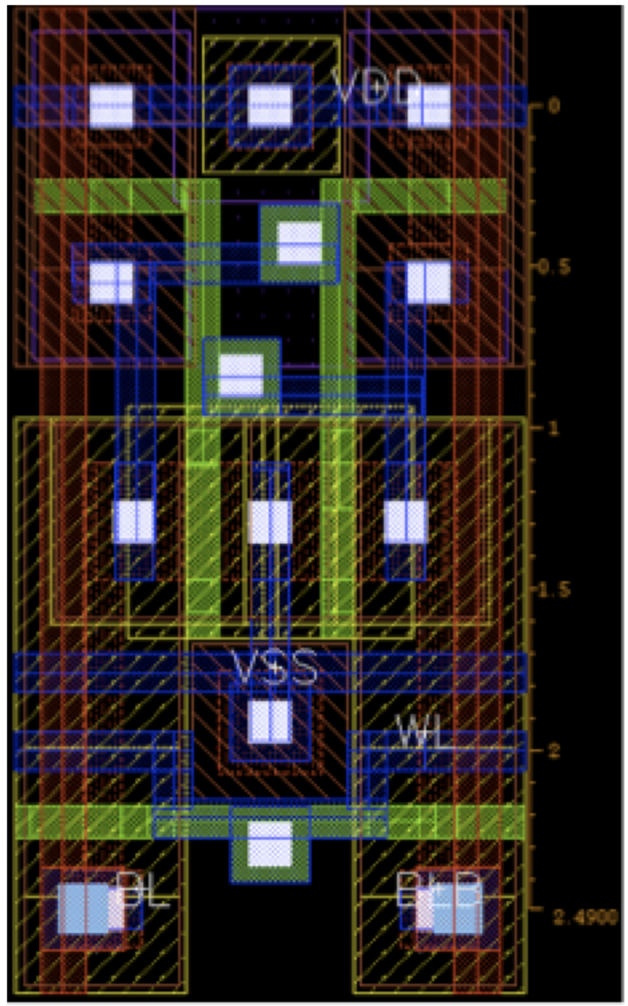
1 Bit SRAM in 6 Transistors



10^{-7} meters

Physical Layout of SRAM Bit

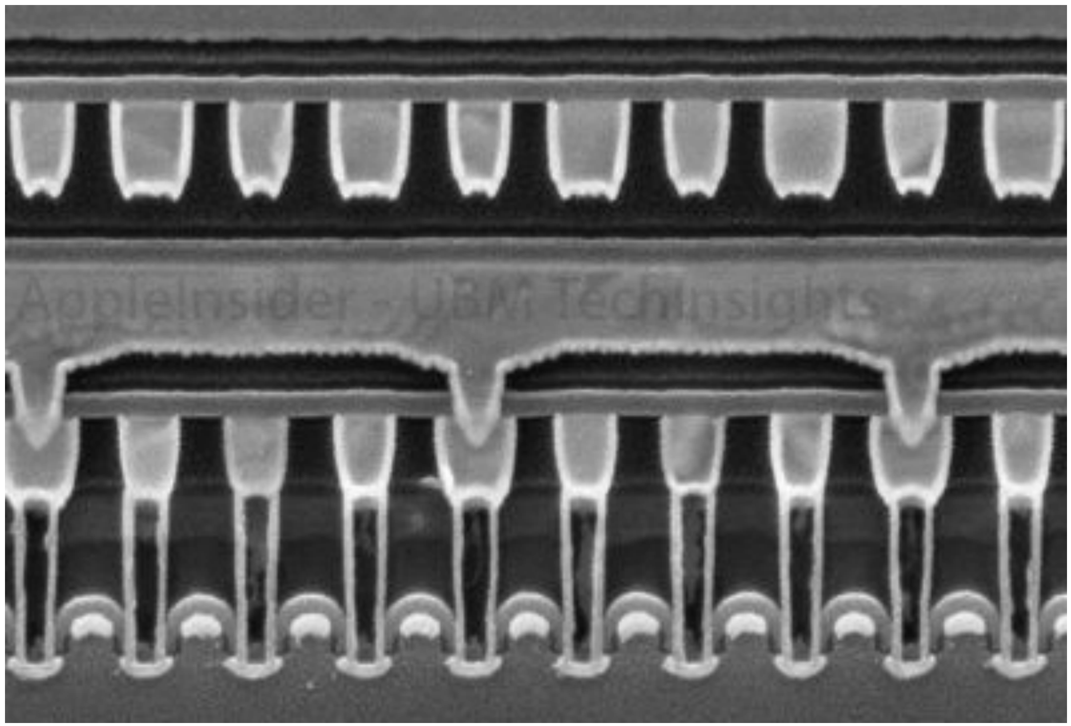
100 nanometers



10^{-7} meters

SRAM Cross Section

100 nanometers



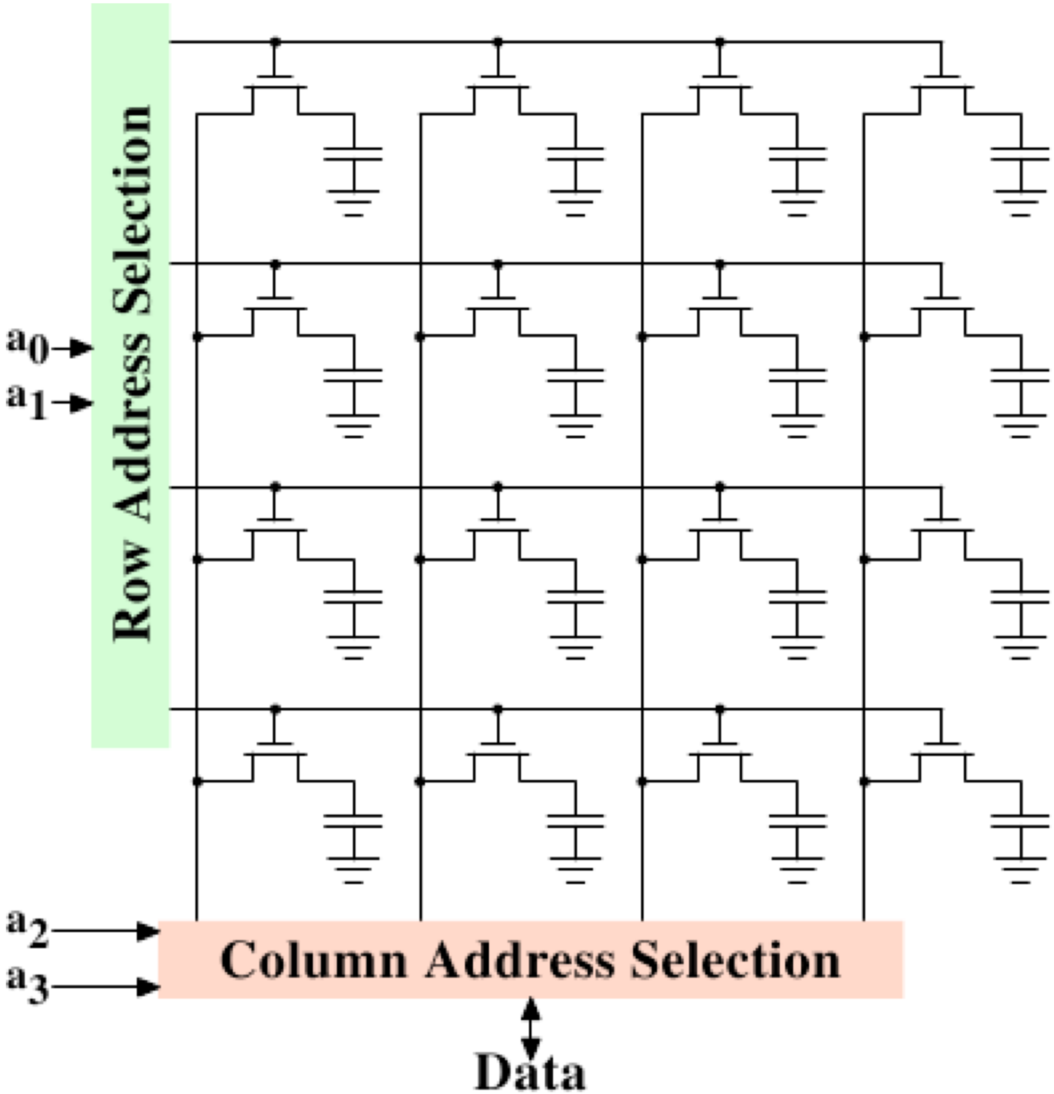
DIMM Module

- DDR = Double Data Rate
  - Transfers bits on Falling AND Rising Clock Edge
- Has Single Error Correcting, Double Error Detecting Redundancy (SEC/DED)
  - 72 bits to store 64 bits of data
  - Uses "Chip kill" organization so that if single DRAM chip fails can still detect failure
- Average server has 22,000 correctable errors and 1 uncorrectable error per year

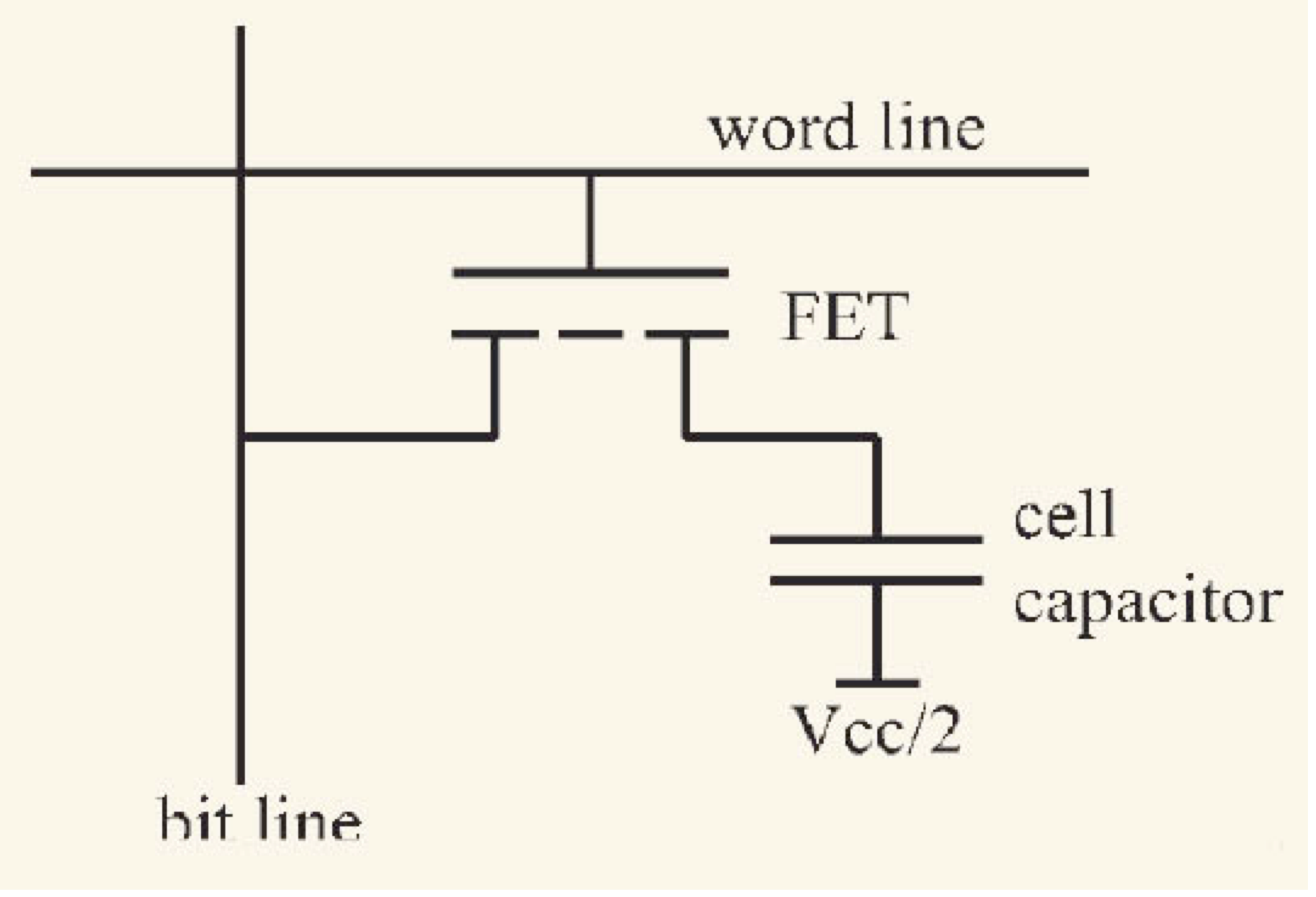
10^{-6} meters

DRAM Bits

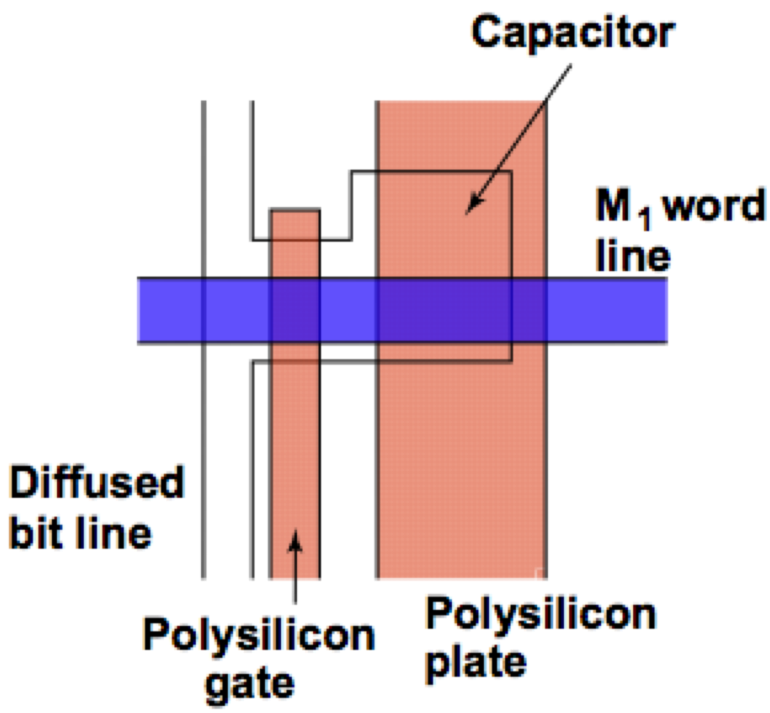
1 micron



DRAM Cell in Transistors



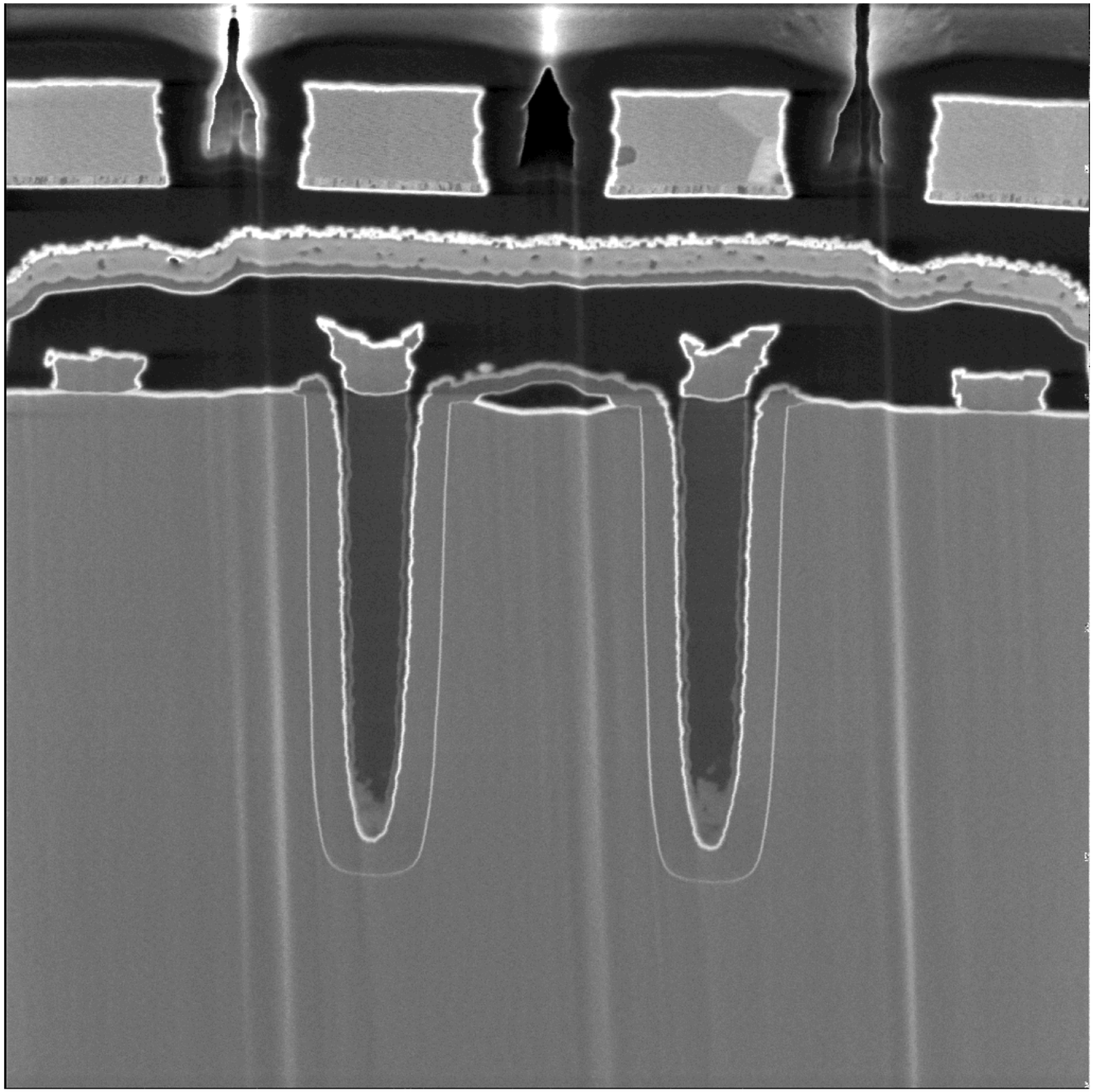
Physical Layout of DRAM Bit



10^{-7} meters

Cross Section of DRAM Bits

100 nanometers



AMD Opteron Dependability

- L1 cache data is SEC/DED protected
- L2 cache and tags are SEC/DED protected
- DRAM is SEC/DED protected with chipkill
- On-chip and off-chip ECC protected arrays include autonomous, background hardware scrubbers
- Remaining arrays are parity protected
  - Instruction cache, tags and TLBs
  - Data tags and TLBs
  - Generally read only data that can be recovered from lower levels

Programming Memory Hierarchy: Cache Blocked Algorithm

- The blocked version of the i-j-k algorithm is written simply as (A,B,C are submatricies of a, b, c)

```
for (i=0;i<N/r;i++)
  for (j=0;j<N/r;j++)
    for (k=0;k<N/r;k++)
      C[i][j] += A[i][k]*B[k][j]
```

  - r = block (sub-matrix) size (Assume r divides N)
  - X[i][j] = a sub-matrix of X, defined by block row i and block column j

Great Ideas in Computer Architecture

1. *Design for Moore's Law*

   -- Higher capacities caches and DRAM
2. Abstraction to Simplify Design
3. Make the Common Case Fast
4. *Dependability via Redundancy*

   -- Parity, SEC/DEC
5. *Memory Hierarchy*

   -- Caches, TLBs
6. *Performance via Parallelism/Pipelining/Prediction*

   -- Data-level Parallelism

Course Summary

- As the field changes, Computer Architecture courses change, too!
- It is still about the software-hardware interface
  - Programming for performance!
  - Parallelism: Task-, Thread-, Instruction-, and Data-MapReduce, OpenMP, C, SSE instrinsics
  - Understanding the memory hierarchy and its impact on application performance