Lecture 14 – Architecture and Performance

Prof. Brendan Kochunas

NERS/ENGR 570 - Methods and Practice of Scientific Computing (F22)

COLLEGE OF ENGINEERING
NUCLEAR ENGINEERING & RADIOLOGICAL SCIENCES
UNIVERSITY OF MICHIGAN

Outline

- Motivating example--Understanding results of Lab 4 (and Lab 5)
- Basic notions of performance (for serial execution)
- Computer Architecture
  - Idealized Models
  - Really what’s going on
  - Practical Mental Models of Architecture
- A simple performance model

Learning Objectives: By the end of Today's Lecture you should be able to

- (*Knowledge*) explain the results observed in lab 4
- (*Knowledge*) define some performance metrics
- (*Knowledge*) describe the memory hierarchy of a single core computer architecture

Motivation

Matrix-matrix Multiply

- Lab 4
  - Write your own matrix-matrix multiply
  - Compare to `dgemm` from BLAS
    - For system BLAS library and OpenBLAS
    - Compare to NumPy

$$\mathbf{AB} = \mathbf{C} \rightarrow c_{i,j} = \sum_k a_{i,k} b_{k,j}$$

Matrix-matrix Multiply

- Lab 4
  - Write your own matrix-matrix multiply
  - Compare to `dgemm` from BLAS
    - For system BLAS library and OpenBLAS
    - Compare to NumPy

$$\mathbf{AB} = \mathbf{C} \rightarrow c_{i,j} = \sum_k a_{i,k} b_{k,j}$$

```
do i=1,n
  do j=1,n
    do k=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
    enddo
  enddo
enddo
```

```
do k=1,n
  do j=1,n
    do i=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
    enddo
  enddo
enddo
```

```
c=MATMUL(A,B)
```

Fortran one-liner

Results for Lab 4

	Using -Ofast						
	Naïve	I know fortran	I'm lazy	Blocked	System Blas	OpenBlas	MKL
20	0	0	0	0	0	0	0.002
100	0.001	0	0.001	0	0	0.001	0.002
500	0.155	0.061	0.01	0.053	0.077	0.007	0.007
1000	2.003	1.645	0.068	0.436	1.409	0.045	0.035
2000	31.465	10.425	0.5	3.562	8.158	0.263	0.242
4000	364.135	87.972	3.997	29.305	46.706	1.869	1.843
	i,j,k	k,j,i (stride-1)	MATMULT	blkSize=256	Reference	Optimized	Optimized

Results obtained on Great Lakes circa 2020(?)

Clearly something is going on here...

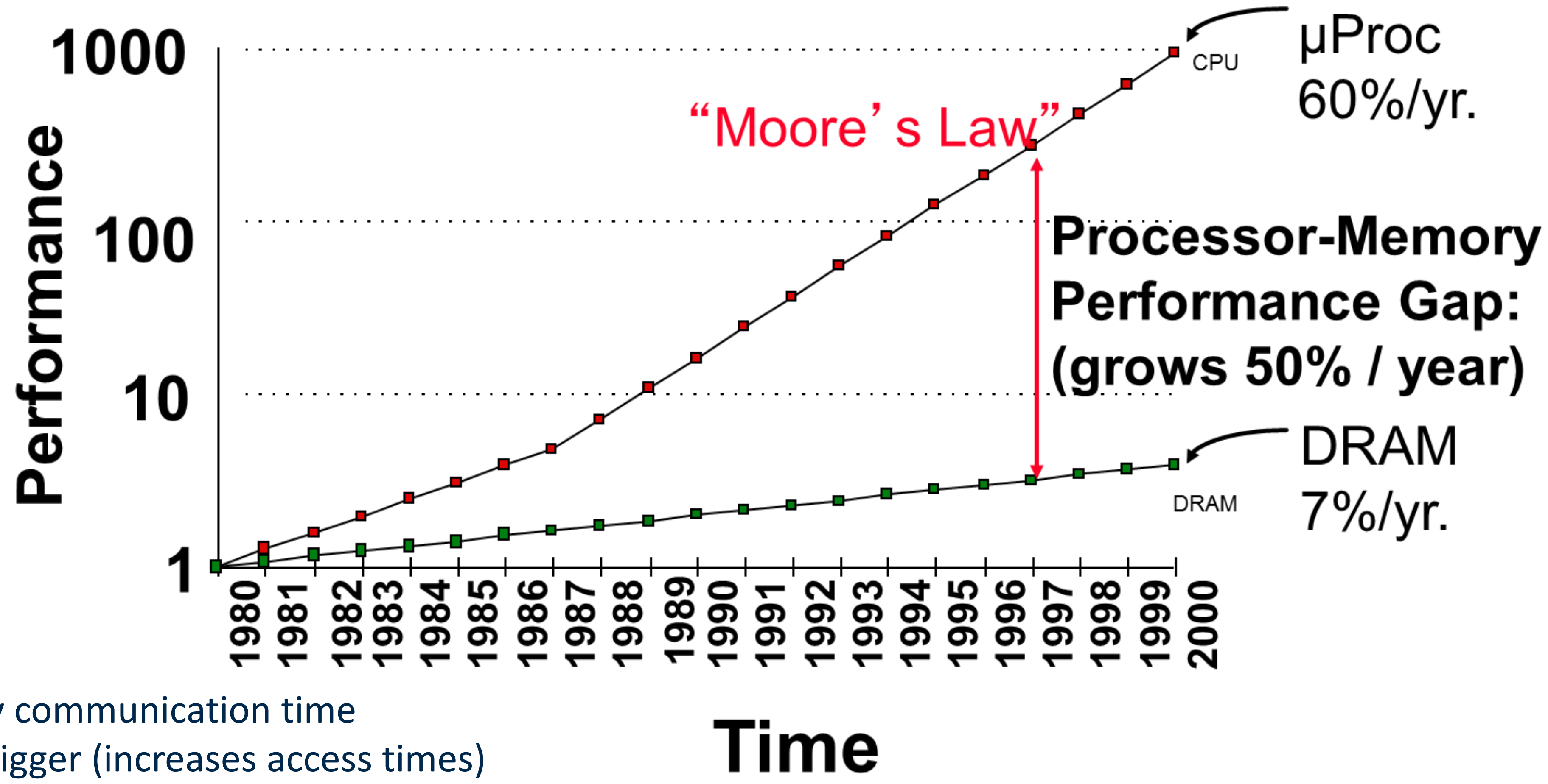
What is NumPy doing?

```
$ module load python3.7-anaconda
$ python
>>> import numpy
>>> numpy.__config__.show()
blas_mkl_info:
    libraries = ['mkl_rt', 'pthread']
    library_dirs = ['/sw/arcts/centos7/python3.7-anaconda/2020.02/lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['/sw/arcts/centos7/python3.7-anaconda/2020.02/include']
blas_opt_info:
    libraries = ['mkl_rt', 'pthread']
    library_dirs = ['/sw/arcts/centos7/python3.7-anaconda/2020.02/lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['/sw/arcts/centos7/python3.7-anaconda/2020.02/include']
lapack_mkl_info:
    libraries = ['mkl_rt', 'pthread']
    library_dirs = ['/sw/arcts/centos7/python3.7-anaconda/2020.02/lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['/sw/arcts/centos7/python3.7-anaconda/2020.02/include']
lapack_opt_info:
    libraries = ['mkl_rt', 'pthread']
    library_dirs = ['/sw/arcts/centos7/python3.7-anaconda/2020.02/lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['/sw/arcts/centos7/python3.7-anaconda/2020.02/include']
```

- Turns out…
  - NumPy uses Intel MKL!
- Anaconda (and probably Canopy) distributions of NumPy also include Intel MKL libraries!
- MKL has a free community edition library.

Fundamental Performance Model Concept

Processor-DRAM Gap (latency)



Main delay is mostly communication time
Memory is getting bigger (increases access times)

Performance Basics

What is performance?

What does it mean for a code to be fast?

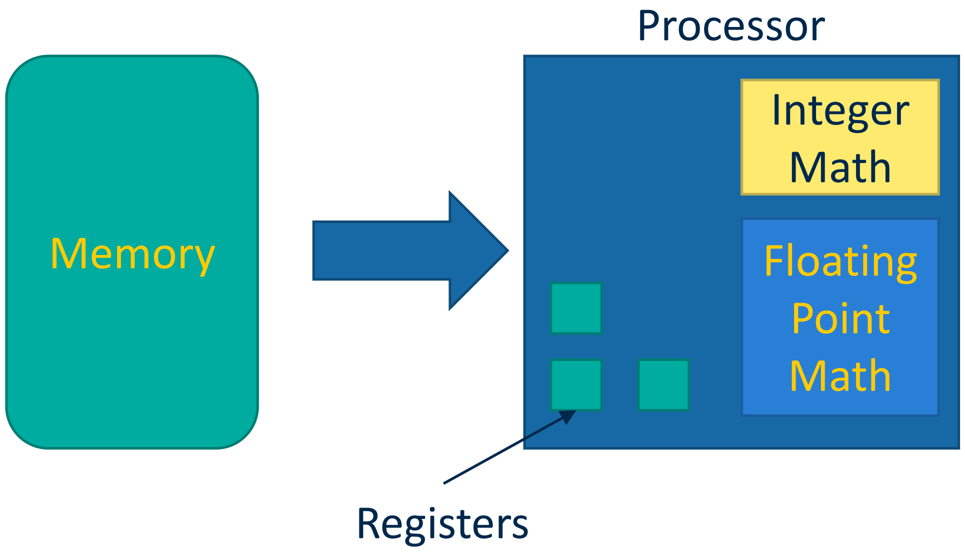
- The real metric: Time
- Derived metrics
  - *FLOPS* = FLoating Point OPerations per Second
  - *Bandwidth* = data per unit time (sort of like a flow rate)
  - *Latency* = Minimum time for data to travel from point A to point B
- Theoretical Peak Performance
  - *Very* difficult to achieve in practice
  - Can be computed from hardware specs
- Do things efficiently in time

Metrics (Schmetrics)

Computer Architecture

Idealized Processor Model

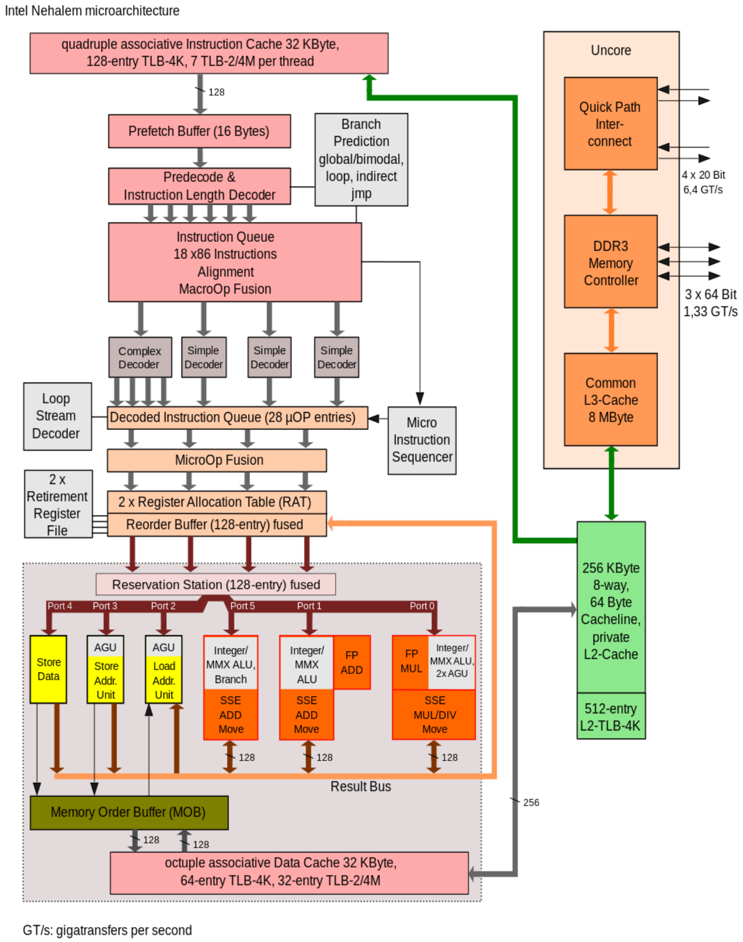
- Processor names bytes, words, etc. in its address space
  - These represent integers, floats, pointers, arrays, etc.
- Operations include
  - Read and write into very fast memory called registers
  - Arithmetic and other logical operations on register
- Order specified by program
  - Read and returns the most recently written data
  - Compiler and architecture translate high level expressions into “obvious” Lower level instructions
  - Hardware executes instructions in order specified by compiler
- Idealized Cost
  - Each operation has roughly the same cost (read, write, add, multiply, etc.)



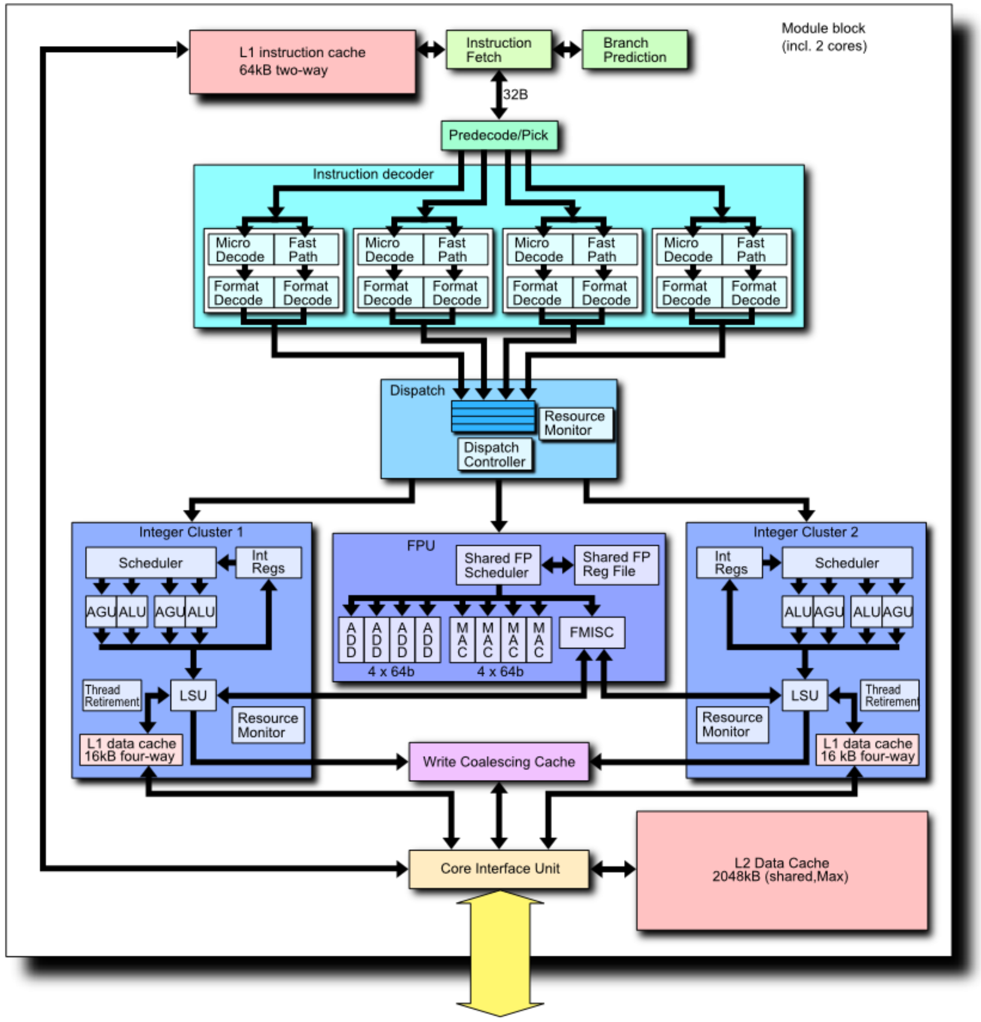
A+B=C →

Read address(A) into R1
Read address(B) into R2
R3 = R1 + R2
Write R3 to address(C)

Real World Processors



Intel "Nehalem"

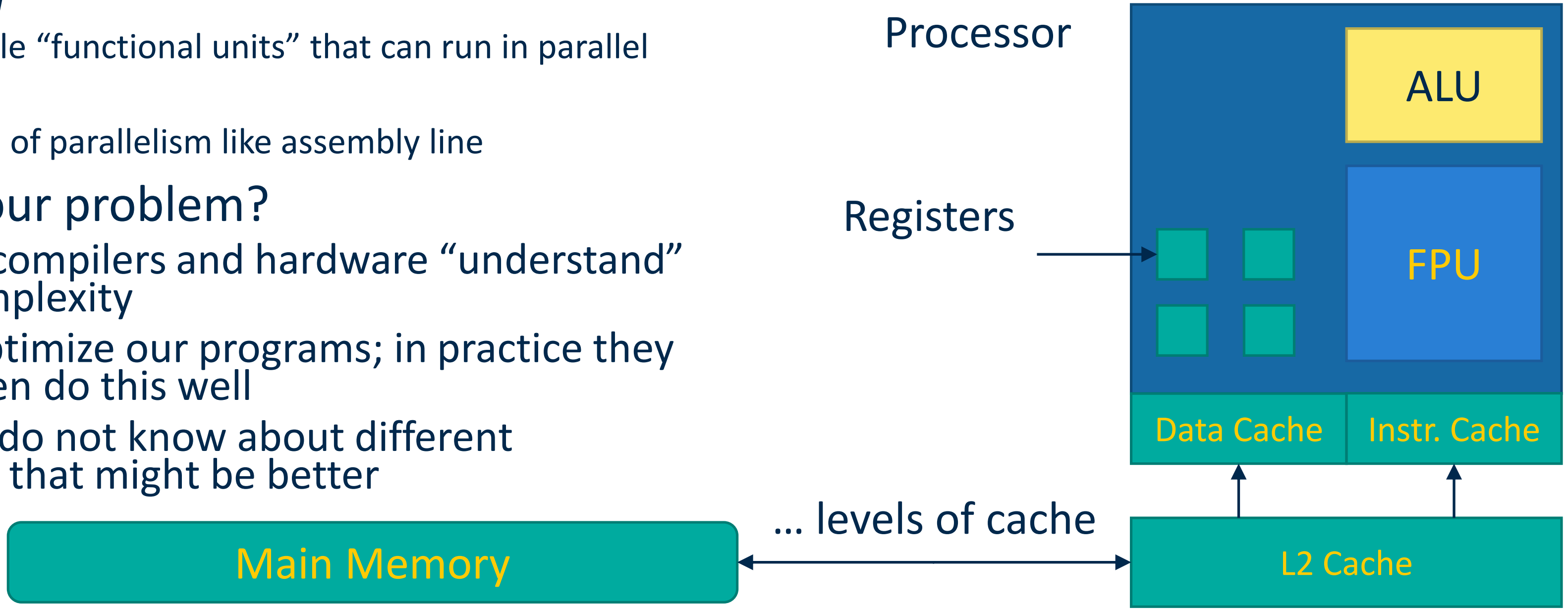


AMD "Bulldozer"

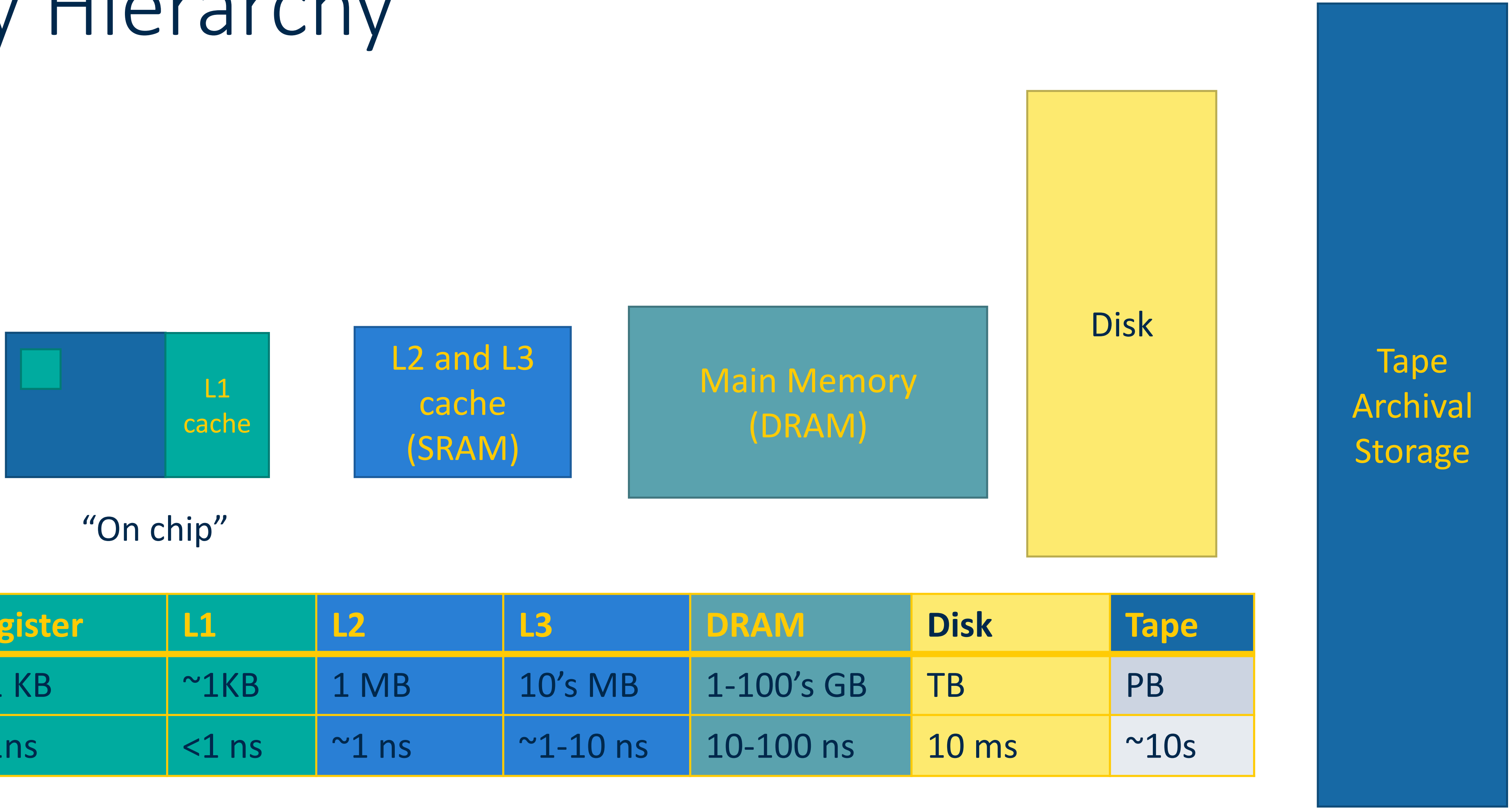
“Single” Processor Concept

- Real world processors have
  - *Registers and caches*
    - Small amounts of fast memory
    - Stores values of recently used data nearby
    - Different memory operations can have very different costs
  - *Parallelism*
    - Multiple “functional units” that can run in parallel
  - *Pipelining*
    - A form of parallelism like assembly line
- Why is this your problem?
  - In theory, compilers and hardware “understand” all this complexity
  - and can optimize our programs; in practice they do not often do this well
  - Compilers do not know about different algorithms that might be better

- We want to know the details to use processors effectively
- Don’t want to know all the details
- Don’t want to have an incomplete model.



Memory Hierarchy



	Register	L1	L2	L3	DRAM	Disk	Tape
Size	< 1 KB	~1KB	1 MB	10's MB	1-100's GB	TB	PB
Speed	< 1ns	<1 ns	~1 ns	~1-10 ns	10-100 ns	10 ms	~10s

Why have multiple levels of cache/memory?

- Most programs have a high degree of locality in their memory access patterns
  - ***Spatial Locality***: accessing data nearby previously accessed data
  - ***Temporal Locality***: access data and reuse that data a lot
  - A memory hierarchy attempts to exploit locality to improve overall average access time.
- Cache is small and fast (speed = $$$)
  - A large cache always has delays: time to check addresses is longer
  - There are other parts to memory hierarchy (TLB, pages, swap, etc.)
- Attempts to reconcile Processor/Memory Gap

A Simple Performance Model

Poll Question 1 – What fraction of peak performance do you expect SpMV to be able to achieve?

Simple Performance Model

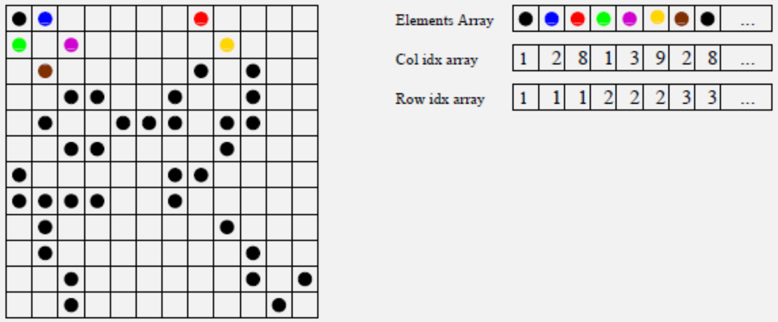
Calculating fraction of peak from our model

Matrix-Vector Multiply

```
{implements y = y + A*x}
for i = 1:n
          for j = 1:n
                y(i) = y(i) + A(i,j)*x(j)
```

SpMV Analysis

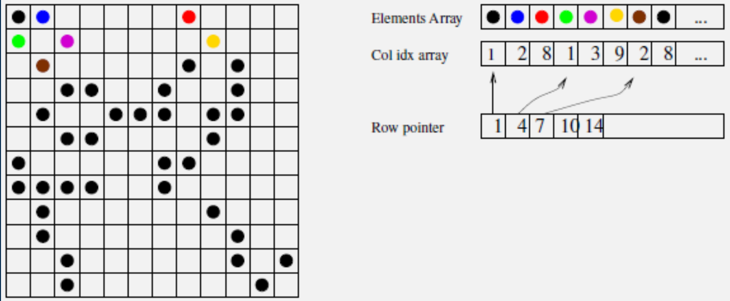
COO




```
do i=1,nz
   ir = ia(i)
   jc = ja(i)
   y(ir) = y(ir) + as(i)*x(jc)
enddo
```

Cost: 5 memory reads, 1 write and 2 flops per iteration.

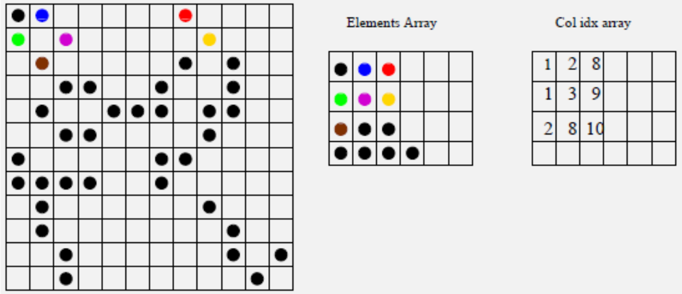
CRS




```
do i=1,m
   do j=ia(i), ia(i+1)-1
      y(i) = y(i) + as(j)*x(ja(j))
   enddo
enddo
```

Cost: 3 memory read and 1 write per outer iteration, 3 memory read and 2 flops per inner iteration.

ELLPACK




```
do i=1,m
   do j=1, maxnz
      y(i) = y(i) + as(i,j)*x(ja(i,j))
   enddo
enddo
```

Cost: 1 memory read and 1 write per outer iteration, 3 memory read and 2 flops per inner iteration (also, regular access pattern).

Poll question 2 -- Now what do you think?

Summary of Performance

Execution time = time to perform arithmetic + *time to move data*

$$T = Ft_f + Lt_m$$

In Lab we'll learn how to measure time memory latencies on a computer

In the next lecture we'll work on developing more complex performance models to understand how algorithmic choice and architecture influence performance metrics.