CS 110
Computer Architecture
Review for Midterm I

Instructor:

Sören Schwertfeger

http://shtech.org/courses/ca/

School of Information Science and Technology SIST

ShanghaiTech University

Slides based on UC Berkley's CS61C

Midterm I

- Date: Thursday, Apr. 19
- Time: 10:15 - 12:15 (normal lecture slot)
  - Be punctual – we start 10:15 sharp!
- Venue: Teaching Center 301 + 302
- One empty seat between students
- Closed book:
  - You can bring **one** A4 page with notes (both sides; English preferred; Chinese is OK): Write your Chinese and **Pinyin** name on the top! **Handwritten** by you!
  - You will be provided with the MIPS ”green sheet”
  - No other material allowed!

Midterm I

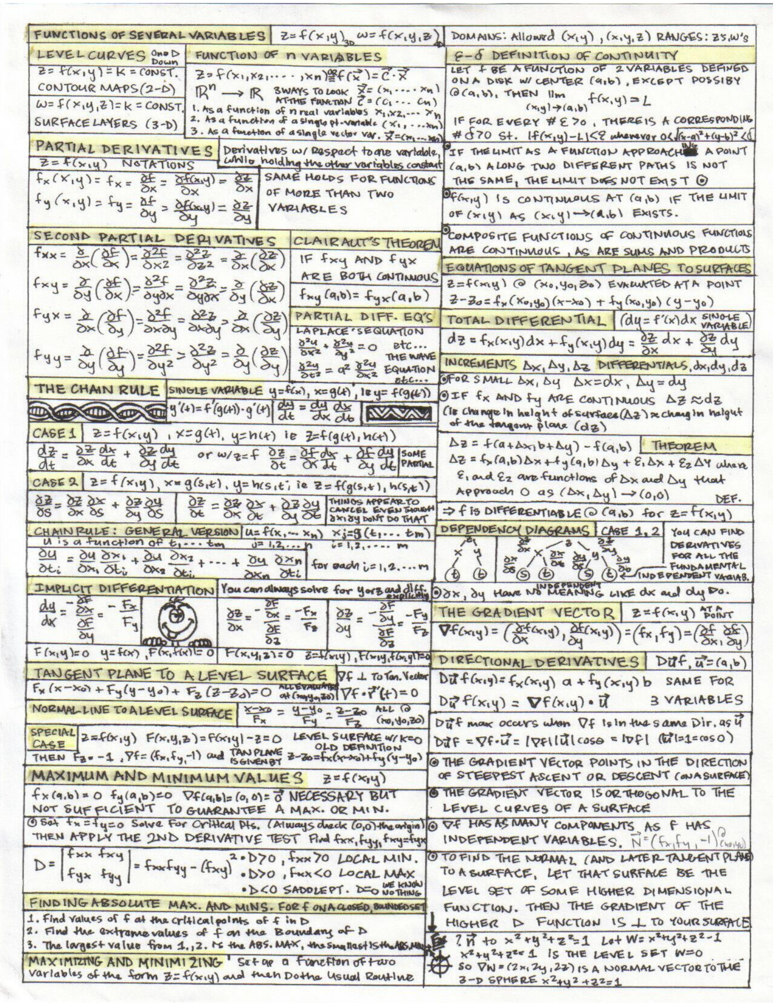
- Switch cell phones **off**! (not silent mode – off!)
  - Put them in your bags.
- Bags under the table. Nothing except paper, pen, 1 drink, 1 snack on the table!
- No other electronic devices are allowed!
  - No ear plugs, music, smartwatch…
- Anybody touching any electronic device will **FAIL** the course!
- Anybody found cheating (copy your neighbors answers, additional material, ...) will **FAIL** the course!

CASIO
JF-120TM
TWO WAY POWER
12 DIGITS
F CUT 5/4
4 3 2 1 0 A
TAX–
TAX RATE
TAX+
RATE SET %
GT
MRC
M–
M+
+/–
▶
C
ON
AC
0
00
·
1
2
3
4
5
6
7
8
9
×
–
+
=

08:08
Thu, Sep 1
HUAWEI

Image hosted by WallpapersHome.com
Tue
9

COMPUTER ORGANIZATION AND DESIGN
THE HARDWARE/SOFTWARE INTERFACE
FIFTH EDITION
DAVID A. PATTERSON
JOHN L. HENNESSY
MK
MORGAN KAUFMANN



FUNCTIONS OF SEVERAL VARIABLES $z=f(x,y)_{3D}$ $w=f(x,y,z)_{4D}$

LEVEL CURVES One D Down

$z=f(x,y)=k=\text{CONST.}$

CONTOUR MAPS (2-D)

$w=f(x,y,z)=k=\text{CONST.}$

SURFACE LAYERS (3-D)

FUNCTION OF n VARIABLES

$z=f(x_1,x_2,\cdots,x_n)$ or $f(\vec{x})=\vec{C}\cdot\vec{x}$

$\mathbb{R}^n \longrightarrow \mathbb{R}$ 3 WAYS TO LOOK AT THE FUNCTION $\vec{x}=(x_1\cdots x_n)$ $\vec{C}=(C_1\cdots C_n)$

1. As a function of n real variables $x_1,x_2,\cdots x_n$
2. As a function of a single pt-variable $(x_1,\dots x_n)$
3. As a function of a single vector var. $\vec{x}=(x_1,\dots x_n)$

PARTIAL DERIVATIVES Derivatives w/ Respect to one variable, while holding the other variables constant

$z=f(x,y)$ NOTATIONS

$f_x(x,y)=f_x=\frac{\partial f}{\partial x}=\frac{\partial f(x,y)}{\partial x}=\frac{\partial z}{\partial x}$

$f_y(x,y)=f_y=\frac{\partial f}{\partial y}=\frac{\partial f(x,y)}{\partial y}=\frac{\partial z}{\partial y}$

SAME HOLDS FOR FUNCTIONS OF MORE THAN TWO VARIABLES

SECOND PARTIAL DERIVATIVES

$f_{xx}=\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial x}\right)=\frac{\partial^2 f}{\partial x^2}=\frac{\partial^2 z}{\partial z^2}=\frac{\partial}{\partial x}\left(\frac{\partial z}{\partial x}\right)$

$f_{xy}=\frac{\partial}{\partial y}\left(\frac{\partial f}{\partial x}\right)=\frac{\partial^2 f}{\partial y\partial x}=\frac{\partial^2 z}{\partial y\partial x}=\frac{\partial}{\partial y}\left(\frac{\partial z}{\partial x}\right)$

$f_{yx}=\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial y}\right)=\frac{\partial^2 f}{\partial x\partial y}=\frac{\partial^2 z}{\partial x\partial y}=\frac{\partial}{\partial x}\left(\frac{\partial z}{\partial y}\right)$

$f_{yy}=\frac{\partial}{\partial y}\left(\frac{\partial f}{\partial y}\right)=\frac{\partial^2 f}{\partial y^2}=\frac{\partial^2 z}{\partial y^2}=\frac{\partial}{\partial y}\left(\frac{\partial z}{\partial y}\right)$

CLAIRAUT'S THEOREM

IF f_{xy} AND f_{yx} ARE BOTH CONTINUOUS

$f_{xy}(a,b)=f_{yx}(a,b)$

PARTIAL DIFF. EQ'S

LAPLACE'S EQUATION $\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0$ etc...

THE WAVE EQUATION $\frac{\partial^2 u}{\partial t^2}=a^2\frac{\partial^2 u}{\partial x^2}$ etc...

THE CHAIN RULE SINGLE VARIABLE $y=f(x)$, $x=g(t)$, ie $y=f(g(t))$

$y'(t)=f'(g(t))\cdot g'(t)$ | $\frac{dy}{dt}=\frac{dy}{dx}\frac{dx}{dt}$

CASE 1 $z=f(x,y)$, $x=g(t)$, $y=h(t)$ ie $z=f(g(t),h(t))$

$\frac{dz}{dt}=\frac{\partial z}{\partial x}\frac{dx}{dt}+\frac{\partial z}{\partial y}\frac{dy}{dt}$ or w/ $z=f$ $\frac{\partial z}{\partial t}=\frac{\partial f}{\partial x}\frac{dx}{dt}+\frac{\partial f}{\partial y}\frac{dy}{dt}$ SOME PARTIAL

CASE 2 $z=f(x,y)$, $x=g(s,t)$, $y=h(s,t)$ ie $z=f(g(s,t),h(s,t))$

$\frac{\partial z}{\partial s}=\frac{\partial z}{\partial x}\frac{\partial x}{\partial s}+\frac{\partial z}{\partial y}\frac{\partial y}{\partial s}$ | $\frac{\partial z}{\partial t}=\frac{\partial z}{\partial x}\frac{\partial x}{\partial t}+\frac{\partial z}{\partial y}\frac{\partial y}{\partial t}$ | THINGS APPEAR TO CANCEL EVEN THOUGH $\partial x, \partial y$ DON'T DO THAT

CHAIN RULE: GENERAL VERSION $u=f(x_1,\cdots x_n)$ $x_j=g(t_1,\dots t_m)$

u is a function of $t_1,\cdots t_m$ $j=1,2,\dots n$ $i=1,2,\dots m$

$\frac{\partial u}{\partial t_i}=\frac{\partial u}{\partial x_1}\frac{\partial x_1}{\partial t_i}+\frac{\partial u}{\partial x_2}\frac{\partial x_2}{\partial t_i}+\cdots+\frac{\partial u}{\partial x_n}\frac{\partial x_n}{\partial t_i}$ for each $i=1,2,\dots m$

IMPLICIT DIFFERENTIATION You can always solve for y or z and diff. explicitly

$\frac{dy}{dx}=-\frac{\frac{\partial F}{\partial x}}{\frac{\partial F}{\partial y}}=-\frac{F_x}{F_y}$ | $\frac{\partial z}{\partial x}=-\frac{\frac{\partial F}{\partial x}}{\frac{\partial F}{\partial z}}=\frac{-F_x}{F_z}$ | $\frac{\partial z}{\partial y}=-\frac{\frac{\partial F}{\partial y}}{\frac{\partial F}{\partial z}}=\frac{-F_y}{F_z}$

$F(x,y)=0$ $y=f(x)$, $F(x,f(x))=0$ | $F(x,y,z)=0$ $z=f(x,y)$, $F(x,y,f(x,y))=0$

TANGENT PLANE TO A LEVEL SURFACE $\nabla f \perp$ To Tan. Vector

$F_x(x-x_0)+F_y(y-y_0)+F_z(z-z_0)=0$ ALL EVALUATED at (x_0,y_0,z_0) | $\nabla f\cdot\vec{r}'(t)=0$

NORMAL LINE TO A LEVEL SURFACE $\frac{x-x_0}{F_x}=\frac{y-y_0}{F_y}=\frac{z-z_0}{F_z}$ ALL @ (x_0,y_0,z_0)

SPECIAL CASE $z=f(x,y)$ $F(x,y,z)=f(x,y)-z=0$ LEVEL SURFACE w/ k=0

THEN $F_z=-1$, $\nabla f=(f_x,f_y,-1)$ and TAN PLANE IS GIVEN BY OLD DEFINITION $z-z_0=f_x(x-x_0)+f_y(y-y_0)$

MAXIMUM AND MINIMUM VALUES $z=f(x,y)$

$f_x(a,b)=0$ $f_y(a,b)=0$ $\nabla f(a,b)=(0,0)=\vec{0}$ NECESSARY BUT NOT SUFFICIENT TO GUARANTEE A MAX. OR MIN.

⊙ Set $f_x=f_y=0$ Solve for Critical Pts. (Always check (0,0) the origin) THEN APPLY THE 2ND DERIVATIVE TEST Find $f_{xx}, f_{yy}, f_{xy}=f_{yx}$

$D=\begin{vmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy}\end{vmatrix}=f_{xx}f_{yy}-(f_{xy})^2$

- $D>0$, $f_{xx}>0$ LOCAL MIN.
- $D>0$, $f_{xx}<0$ LOCAL MAX
- $D<0$ SADDLE PT. $D=0$ WE KNOW NOTHING

FINDING ABSOLUTE MAX. AND MINS. FOR f ON A CLOSED, BOUNDED SET

1. Find values of f at the critical points of f in D
2. Find the extreme values of f on the Boundary of D
3. The largest value from 1., 2. is the ABS. MAX, the smallest is the ABS. MIN.

MAXIMIZING AND MINIMIZING Set up a function of two variables of the form $z=f(x,y)$ and then Do the Usual Routine

DOMAINS: Allowed (x,y), (x,y,z) RANGES: z's, w's

ε-δ DEFINITION OF CONTINUITY

LET f BE A FUNCTION OF 2 VARIABLES DEFINED ON A DISK W/ CENTER (a,b), EXCEPT POSSIBY @ (a,b). THEN $\lim_{(x,y)\to(a,b)} f(x,y)=L$

IF FOR EVERY # ε>0, THERE IS A CORRESPONDING # δ>0 St. $|f(x,y)-L|<\varepsilon$ whenever $0<\sqrt{(x-a)^2+(y-b)^2}<\delta$

⊙ IF THE LIMIT AS A FUNCTION APPROACHING A POINT (a,b) ALONG TWO DIFFERENT PATHS IS NOT THE SAME, THE LIMIT DOES NOT EXIST ⊙

⊙ f(x,y) IS CONTINUOUS AT (a,b) IF THE LIMIT OF (x,y) AS (x,y)→(a,b) EXISTS.

⊙ COMPOSITE FUNCTIONS OF CONTINUOUS FUNCTIONS ARE CONTINUOUS, AS ARE SUMS AND PRODUCTS

EQUATIONS OF TANGENT PLANES TO SURFACES

$z=f(x,y)$ @ (x_0,y_0,z_0) EVALUATED AT A POINT

$z-z_0=f_x(x_0,y_0)(x-x_0)+f_y(x_0,y_0)(y-y_0)$

TOTAL DIFFERENTIAL ($dy=f'(x)dx$ SINGLE VARIABLE)

$dz=f_x(x,y)dx+f_y(x,y)dy=\frac{\partial z}{\partial x}dx+\frac{\partial z}{\partial y}dy$

INCREMENTS $\Delta x, \Delta y, \Delta z$ **DIFFERENTIALS**, dx, dy, dz

⊙ FOR SMALL $\Delta x, \Delta y$ $\Delta x=dx$, $\Delta y=dy$

⊙ IF f_x AND f_y ARE CONTINUOUS $\Delta z\approx dz$ (Is change in height of surface (Δz) ≈ change in height of the tangent plane (dz))

$\Delta z=f(a+\Delta x,b+\Delta y)-f(a,b)$ **THEOREM**

$\Delta z=f_x(a,b)\Delta x+f_y(a,b)\Delta y+\varepsilon_1\Delta x+\varepsilon_2\Delta y$ where ε_1 and ε_2 are functions of Δx and Δy that Approach 0 as $(\Delta x,\Delta y)\to(0,0)$ DEF.

⇒ f is DIFFERENTIABLE @ (a,b) for $z=f(x,y)$

DEPENDENCY DIAGRAMS CASE 1, 2

$$\begin{tikzcd}[column sep=small] & z \arrow[dl, dash] \arrow[dr, dash] & \\ x \arrow[d, dash] & & y \arrow[d, dash] \\ t & & t \end{tikzcd}$$

$$\begin{tikzcd}[column sep=small] & & z \arrow[dl, dash, "\frac{\partial z}{\partial x}"'] \arrow[dr, dash, "\frac{\partial z}{\partial y}"] & & \\ & x \arrow[dl, dash, "\frac{\partial x}{\partial s}"'] \arrow[d, dash, "\frac{\partial x}{\partial t}"] & & y \arrow[d, dash, "\frac{\partial y}{\partial s}"'] \arrow[dr, dash, "\frac{\partial y}{\partial t}"] & \\ s & t & & s & t \end{tikzcd}$$

You CAN FIND DERIVATIVES FOR ALL THE FUNDAMENTAL INDEPENDENT VARIAB.

⊙ ∂x, ∂y Have NO INDEPENDENT MEANING LIKE dx and dy Do.

THE GRADIENT VECTOR $z=f(x,y)$ AT A POINT

$\nabla f(x,y)=\left(\frac{\partial f(x,y)}{\partial x},\frac{\partial f(x,y)}{\partial y}\right)=(f_x,f_y)=\left(\frac{\partial f}{\partial x},\frac{\partial f}{\partial y}\right)$

DIRECTIONAL DERIVATIVES $D_{\vec{u}}f$, $\vec{u}=(a,b)$

$D_{\vec{u}}f(x,y)=f_x(x,y)\,a+f_y(x,y)\,b$ SAME FOR

$D_{\vec{u}}f(x,y)=\nabla f(x,y)\cdot\vec{u}$ 3 VARIABLES

$D_{\vec{u}}f$ max occurs when ∇f is in the same Dir. as $\vec{u}$

$D_{\vec{u}}F=\nabla f\cdot\vec{u}=|\nabla f||\vec{u}|\cos\theta=|\nabla f|$ ($|\vec{u}|=1=\cos 0$)

⊙ THE GRADIENT VECTOR POINTS IN THE DIRECTION OF STEEPEST ASCENT OR DESCENT (ON A SURFACE)

⊙ THE GRADIENT VECTOR IS ORTHOGONAL TO THE LEVEL CURVES OF A SURFACE

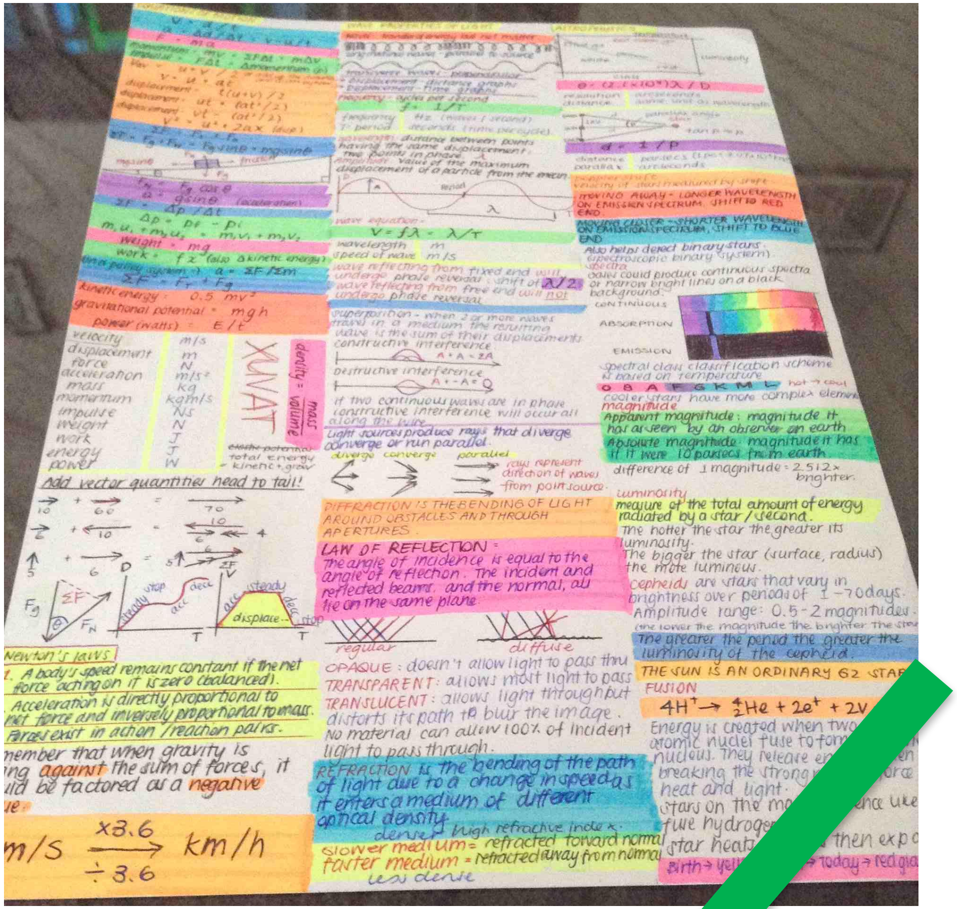
⊙ ∇f HAS AS MANY COMPONENTS AS f HAS INDEPENDENT VARIABLES. $\vec{N}=(f_x,f_y,-1)$ @ (x_0,y_0)

⊙ TO FIND THE NORMAL (AND LATER TANGENT PLANE) TO A SURFACE, LET THAT SURFACE BE THE LEVEL SET OF SOME HIGHER DIMENSIONAL FUNCTION. THEN THE GRADIENT OF THE HIGHER D FUNCTION IS ⊥ TO YOUR SURFACE

Ex ? $\vec{n}$ to $x^2+y^2+z^2=1$ Let $W=x^2+y^2+z^2-1$

$x^2+y^2+z^2=1$ IS THE LEVEL SET $W=0$

SO $\nabla W=(2x,2y,2z)$ IS A NORMAL VECTOR TO THE 3-D SPHERE $x^2+y^2+z^2=1$



$v = d/t$

$a = \Delta d/\Delta t = v-u/t$

$F = ma$

momentum = mv = ΣFΔt = mΔv

impulse = FΔt = Δmomentum (p)

$V_{av} = u+v / 2$

$v = u + at$

displacement = $t(u+v)/2$

displacement = $ut + (at^2/2)$

displacement = $vt - (at^2/2)$

$v^2 = u^2 + 2ax$

$\Sigma F = F_g + F_N = F_g\sin\theta = mg\sin\theta$

mg sinθ
friction
F_N
F_g

$F_N = F_g\cos\theta$

$a = g\sin\theta$ (acceleration)

$\Sigma F = \Delta p/\Delta t$

$\Delta p = p_f - p_i$

$m_1u_1 + m_2u_2 = m_1v_1 + m_2v_2$

weight = mg

work = fx (also Δkinetic energy)

pulley system → $a = \Sigma F/\Sigma m$

$\Sigma F = F_T + F_g$

kinetic energy = $0.5\,mv^2$

gravitational potential = mgh

power (watts) = E/t

velocity	m/s
displacement	m
force	N
acceleration	m/s²
mass	kg
momentum	kgm/s
impulse	Ns
weight	N
work	J
energy	J
power	W

XUVAT

density = $\frac{\text{mass}}{\text{volume}}$

~~elastic potential~~ total energy = kinetic + grav

Add vector quantities head to tail!

Newton's laws

1. A body's speed remains constant if the net force acting on it is zero (balanced).

Acceleration is directly proportional to net force and inversely proportional to mass.

Forces exist in action / reaction pairs.

member that when gravity is ing against the sum of forces, it uld be factored as a negative ue.

$m/s \xrightarrow[\div 3.6]{\times 3.6} km/h$

WAVE PROPERTIES OF LIGHT

wave - transfer of energy but not matter

longitudinal waves - parallel to source

transverse waves - perpendicular

→ displacement - distance graphs

→ displacement - time graphs

frequency - cycles per second

$f = 1/T$

f frequency	Hz (waves / second)
T period	seconds (time per cycle)

wavelength: distance between points having the same displacement: two points in phase λ

amplitude: value of the maximum displacement of a particle from the mean

wave equation:

$V = f\lambda = \lambda/T$

wavelength	m
speed of wave	m/s

wave reflecting from fixed end will undergo phase reversal: shift of λ/2

wave reflecting from free end will not undergo phase reversal

superposition - when 2 or more waves travel in a medium the resulting wave is the sum of their displacements

constructive interference

A + A = 2A

destructive interference

A + -A = 0

if two continuous waves are in phase constructive interference will occur all along the wave

Light sources produce rays that diverge converge or run parallel.

diverge converge parallel

rays represent direction of waves from point source.

DIFFRACTION IS THE BENDING OF LIGHT AROUND OBSTACLES AND THROUGH APERTURES.

LAW OF REFLECTION = The angle of incidence is equal to the angle of reflection. The incident and reflected beams, and the normal, all lie on the same plane.

regular diffuse

OPAQUE: doesn't allow light to pass thru

TRANSPARENT: allows most light to pass

TRANSLUCENT: allows light through but distorts its path to blur the image.

No material can allow 100% of incident light to pass through.

REFRACTION is the bending of the path of light due to a change in speed as it enters a medium of different optical density.

denser - high refractive index.

slower medium = refracted toward normal

faster medium = refracted away from normal

less dense

ASTROPHYSICS

$\theta = (2.1\times10^{5})\lambda / D$

resolution distance	arcseconds same unit as wavelength

$\tan p \approx p$

$d = 1/p$

distance	parsecs
parallax	arcseconds

Doppler shift velocity of stars measured by shift.

MOVING AWAY - LONGER WAVELENGTH ON EMISSION SPECTRUM, SHIFT TO RED END.

MOVING CLOSER - SHORTER WAVELENGTH ON EMISSION SPECTRUM, SHIFT TO BLUE END

Also helps detect binary stars. (spectroscopic binary system)

spectra

gases could produce continuous spectra or narrow bright lines on a black background.

CONTINUOUS

ABSORPTION

EMISSION

spectral class classification scheme is based on temperature

O B A F G K M L hot → cool

cooler stars have more complex elements

magnitude

Apparent magnitude: magnitude it has as seen by an observer on earth

Absolute magnitude: magnitude it has if it were 10 parsecs from earth

difference of 1 magnitude = 2.512× brighter.

luminosity

measure of the total amount of energy radiated by a star / second.

The hotter the star the greater its luminosity.

The bigger the star (surface, radius) the more luminous.

cepheids are stars that vary in brightness over periods of 1 - 70 days.

Amplitude range: 0.5 - 2 magnitudes.

(the lower the magnitude the brighter the star)

The greater the period the greater the luminosity of the cepheid.

THE SUN IS AN ORDINARY G2 STAR

FUSION

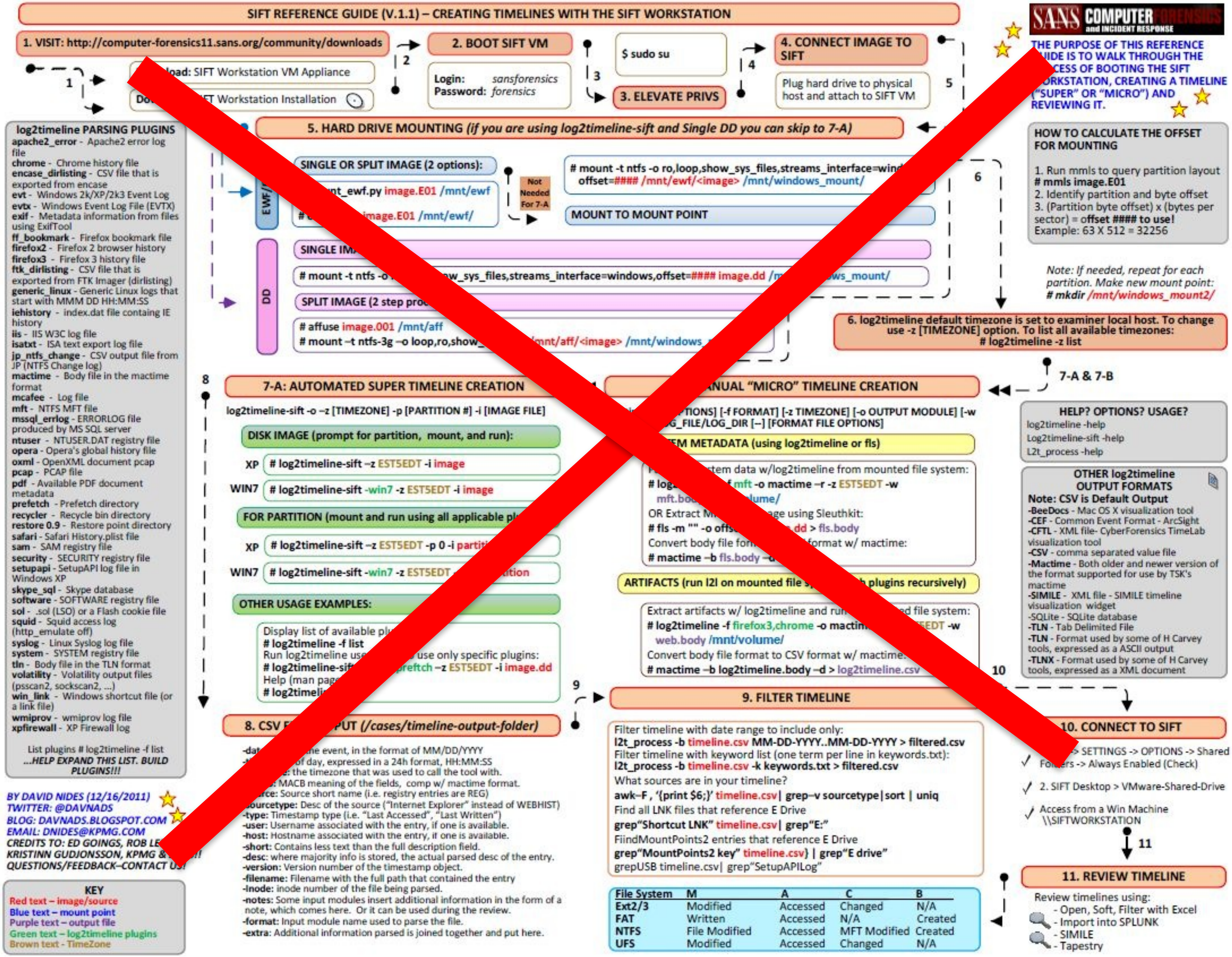
$4H^{+} \rightarrow {}^{4}_{2}He + 2e^{+} + 2V$

Energy is created when two atomic nuclei fuse to form nucleus. They release en breaking the strong force heat and light.

stars on the m ence like fuse hydrogen star heats then exp

Birth → yel → today → red gia

✓



SIFT REFERENCE GUIDE (V.1.1) – CREATING TIMELINES WITH THE SIFT WORKSTATION

THE PURPOSE OF THIS REFERENCE UIDE IS TO WALK THROUGH THE CESS OF BOOTING THE SIFT ORKSTATION, CREATING A TIMELINE ("SUPER" OR "MICRO") AND REVIEWING IT.

1. VISIT: http://computer-forensics11.sans.org/community/downloads

load: SIFT Workstation VM Appliance

Do FT Workstation Installation

2. BOOT SIFT VM

Login: *sansforensics*
Password: *forensics*

```
$ sudo su
```

3. ELEVATE PRIVS

4. CONNECT IMAGE TO SIFT

Plug hard drive to physical host and attach to SIFT VM

log2timeline PARSING PLUGINS

apache2_error - Apache2 error log file
chrome - Chrome history file
encase_dirlisting - CSV file that is exported from encase
evt - Windows 2k/XP/2k3 Event Log
evtx - Windows Event Log File (EVTX)
exif - Metadata information from files using ExifTool
ff_bookmark - Firefox bookmark file
firefox2 - Firefox 2 browser history
firefox3 - Firefox 3 history file
ftk_dirlisting - CSV file that is exported from FTK Imager (dirlisting)
generic_linux - Generic Linux logs that start with MMM DD HH:MM:SS
iehistory - index.dat file containg IE history
iis - IIS W3C log file
isatxt - ISA text export log file
jp_ntfs_change - CSV output file from JP (NTFS Change log)
mactime - Body file in the mactime format
mcafee - Log file
mft - NTFS MFT file
mssql_errlog - ERRORLOG file produced by MS SQL server
ntuser - NTUSER.DAT registry file
opera - Opera's global history file
oxml - OpenXML document pcap
pcap - PCAP file
pdf - Available PDF document metadata
prefetch - Prefetch directory
recycler - Recycle bin directory
restore 0.9 - Restore point directory
safari - Safari History.plist file
sam - SAM registry file
security - SECURITY registry file
setupapi - SetupAPI log file in Windows XP
skype_sql - Skype database
software - SOFTWARE registry file
sol - .sol (LSO) or a Flash cookie file
squid - Squid access log (http_emulate off)
syslog - Linux Syslog log file
system - SYSTEM registry file
tln - Body file in the TLN format
volatility - Volatility output files (psscan2, sockscan2, ...)
win_link - Windows shortcut file (or a link file)
wmiprov - wmiprov log file
xpfirewall - XP Firewall log

List plugins # log2timeline -f list
...HELP EXPAND THIS LIST. BUILD PLUGINS!!!

5. HARD DRIVE MOUNTING *(if you are using log2timeline-sift and Single DD you can skip to 7-A)*

SINGLE OR SPLIT IMAGE (2 options):

```
nt_ewf.py image.E01 /mnt/ewf
# image.E01 /mnt/ewf/
```

```
# mount -t ntfs -o ro,loop,show_sys_files,streams_interface=wind
  offset=#### /mnt/ewf/<image> /mnt/windows_mount/
```

MOUNT TO MOUNT POINT

SINGLE IM

```
# mount -t ntfs -o w_sys_files,streams_interface=windows,offset=#### image.dd /m ws_mount/
```

SPLIT IMAGE (2 step pro

```
# affuse image.001 /mnt/aff
# mount –t ntfs-3g –o loop,ro,show_ mnt/aff/<image> /mnt/windows_
```

HOW TO CALCULATE THE OFFSET FOR MOUNTING

1. Run mmls to query partition layout
# mmls image.E01
2. Identify partition and byte offset
3. (Partition byte offset) x (bytes per sector) = **offset #### to use!**
Example: 63 X 512 = 32256

Note: If needed, repeat for each partition. Make new mount point:
# mkdir */mnt/windows_mount2/*

6. log2timeline default timezone is set to examiner local host. To change use -z [TIMEZONE] option. To list all available timezones: # log2timeline -z list

7-A: AUTOMATED SUPER TIMELINE CREATION

log2timeline-sift -o –z [TIMEZONE] -p [PARTITION #] -i [IMAGE FILE]

DISK IMAGE (prompt for partition, mount, and run):

XP
```
# log2timeline-sift –z EST5EDT -i image
```

WIN7
```
# log2timeline-sift -win7 -z EST5EDT -i image
```

FOR PARTITION (mount and run using all applicable pl

XP
```
# log2timeline-sift –z EST5EDT -p 0 -i partiti
```

WIN7
```
# log2timeline-sift -win7 -z EST5EDT ition
```

OTHER USAGE EXAMPLES:

Display list of available pl
# log2timeline -f list
Run log2timeline use use only specific plugins:
# log2timeline-sif preftch –z EST5EDT -i image.dd
Help (man pag
# log2timeli

NUAL "MICRO" TIMELINE CREATION

TIONS] [-f FORMAT] [-z TIMEZONE] [-o OUTPUT MODULE] [-w G_FILE/LOG_DIR [--] [FORMAT FILE OPTIONS]

EM METADATA (using log2timeline or fls)

tem data w/log2timeline from mounted file system:
```
# log f mft -o mactime –r -z EST5EDT -w
  mft.bo lume/
```
OR Extract M age using Sleuthkit:
```
# fls -m "" -o offs dd > fls.body
```
Convert body file for format w/ mactime:
```
# mactime –b fls.body –d
```

ARTIFACTS (run l2l on mounted file h plugins recursively)

Extract artifacts w/ log2timeline and run d file system:
```
# log2timeline -f firefox3,chrome -o mactim EDT -w
  web.body /mnt/volume/
```
Convert body file format to CSV format w/ mactime:
```
# mactime –b log2timeline.body –d > log2timeline.csv
```

HELP? OPTIONS? USAGE?

log2timeline -help
Log2timeline-sift -help
L2t_process -help

OTHER log2timeline OUTPUT FORMATS

Note: CSV is Default Output
-BeeDocs - Mac OS X visualization tool
-CEF - Common Event Format - ArcSight
-CFTL - XML file- CyberForensics TimeLab visualization tool
-CSV - comma separated value file
-Mactime - Both older and newer version of the format supported for use by TSK's mactime
-SIMILE - XML file - SIMILE timeline visualization widget
-SQLite - SQLite database
-TLN - Tab Delimited File
-TLN - Format used by some of H Carvey tools, expressed as a ASCII output
-TLNX - Format used by some of H Carvey tools, expressed as a XML document

8. CSV PUT *(/cases/timeline-output-folder)*

-dat he event, in the format of MM/DD/YYYY
of day, expressed in a 24h format, HH:MM:SS
: the timezone that was used to call the tool with.
: MACB meaning of the fields, comp w/ mactime format.
rce: Source short name (i.e. registry entries are REG)
ourcetype: Desc of the source ("Internet Explorer" instead of WEBHIST)
-type: Timestamp type (i.e. "Last Accessed", "Last Written")
-user: Username associated with the entry, if one is available.
-host: Hostname associated with the entry, if one is available.
-short: Contains less text than the full description field.
-desc: where majority info is stored, the actual parsed desc of the entry.
-version: Version number of the timestamp object.
-filename: Filename with the full path that contained the entry
-Inode: inode number of the file being parsed.
-notes: Some input modules insert additional information in the form of a note, which comes here. Or it can be used during the review.
-format: Input module name used to parse the file.
-extra: Additional information parsed is joined together and put here.

9. FILTER TIMELINE

Filter timeline with date range to include only:
l2t_process -b timeline.csv MM-DD-YYYY..MM-DD-YYYY > filtered.csv
Filter timeline with keyword list (one term per line in keywords.txt):
l2t_process -b timeline.csv -k keywords.txt > filtered.csv
What sources are in your timeline?
awk–F , '{print $6;}' timeline.csv| grep–v sourcetype|sort | uniq
Find all LNK files that reference E Drive
grep"Shortcut LNK" timeline.csv| grep"E:"
FiindMountPoints2 entries that reference E Drive
grep"MountPoints2 key" timeline.csv} | grep"E drive"
grepUSB timeline.csv| grep"SetupAPILog"

File System	M	A	C	B
Ext2/3	Modified	Accessed	Changed	N/A
FAT	Written	Accessed	N/A	Created
NTFS	File Modified	Accessed	MFT Modified	Created
UFS	Modified	Accessed	Changed	N/A

10. CONNECT TO SIFT

✓ > SETTINGS -> OPTIONS -> Shared Fo ers -> Always Enabled (Check)
✓ 2. SIFT Desktop > VMware-Shared-Drive
✓ Access from a Win Machine \\SIFTWORKSTATION

11. REVIEW TIMELINE

Review timelines using:
- Open, Soft, Filter with Excel
- Import into SPLUNK
- SIMILE
- Tapestry

BY DAVID NIDES (12/16/2011)
TWITTER: @DAVNADS
BLOG: DAVNADS.BLOGSPOT.COM
EMAIL: DNIDES@KPMG.COM
CREDITS TO: ED GOINGS, ROB LE
KRISTINN GUDJONSSON, KPMG & !!
QUESTIONS/FEEDBACK–CONTACT US!

KEY
Red text – image/source
Blue text – mount point
Purple text – output file
Green text – log2timeline plugins
Brown text - TimeZone

Midterm I

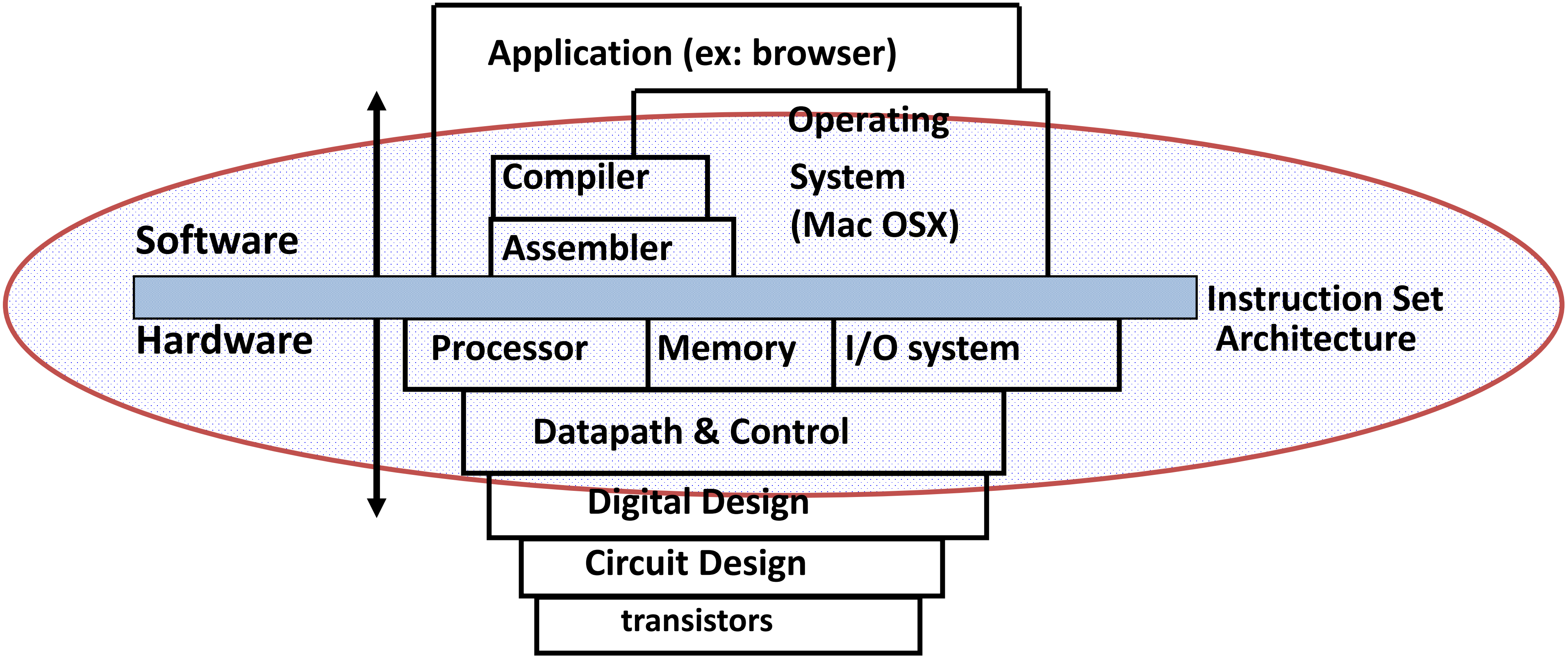
- Ask questions today!
- Discussion is Q&A session
  - Suggest topics for review in piazza!

- This review session does not/ can not cover all possible topics!

Content

- Main topics
  - Number representation
  - C
  - MIPS
- Plus general ”Computer Architecture” knowledge
- Everything till lecture 8 CALL – including lecture 8

Old School Machine Structures

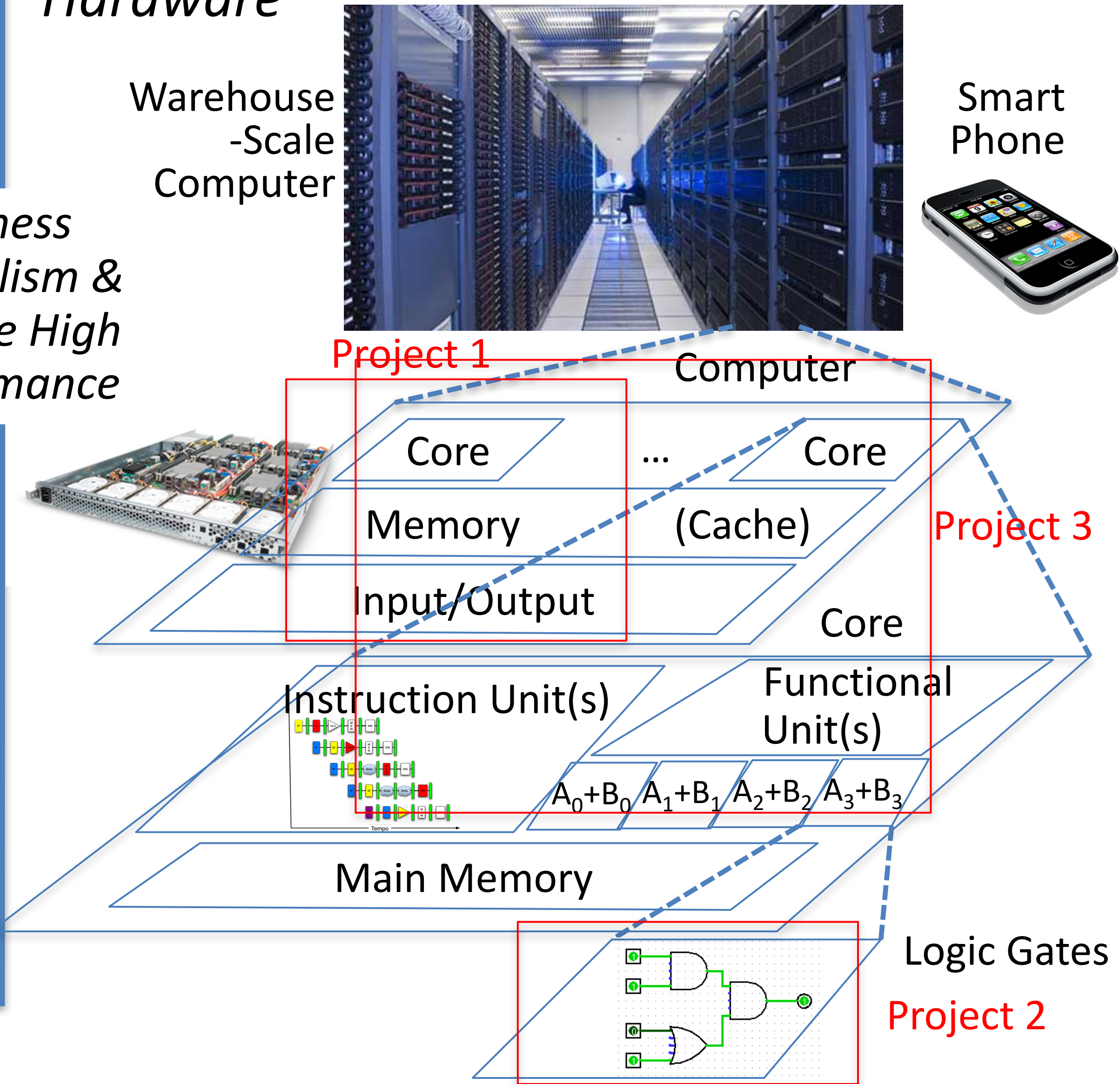


New-School Machine Structures (It’s a bit more complicated!)

Software | *Hardware*

- Parallel Requests
  Assigned to computer
  e.g., Search “cats”
- 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

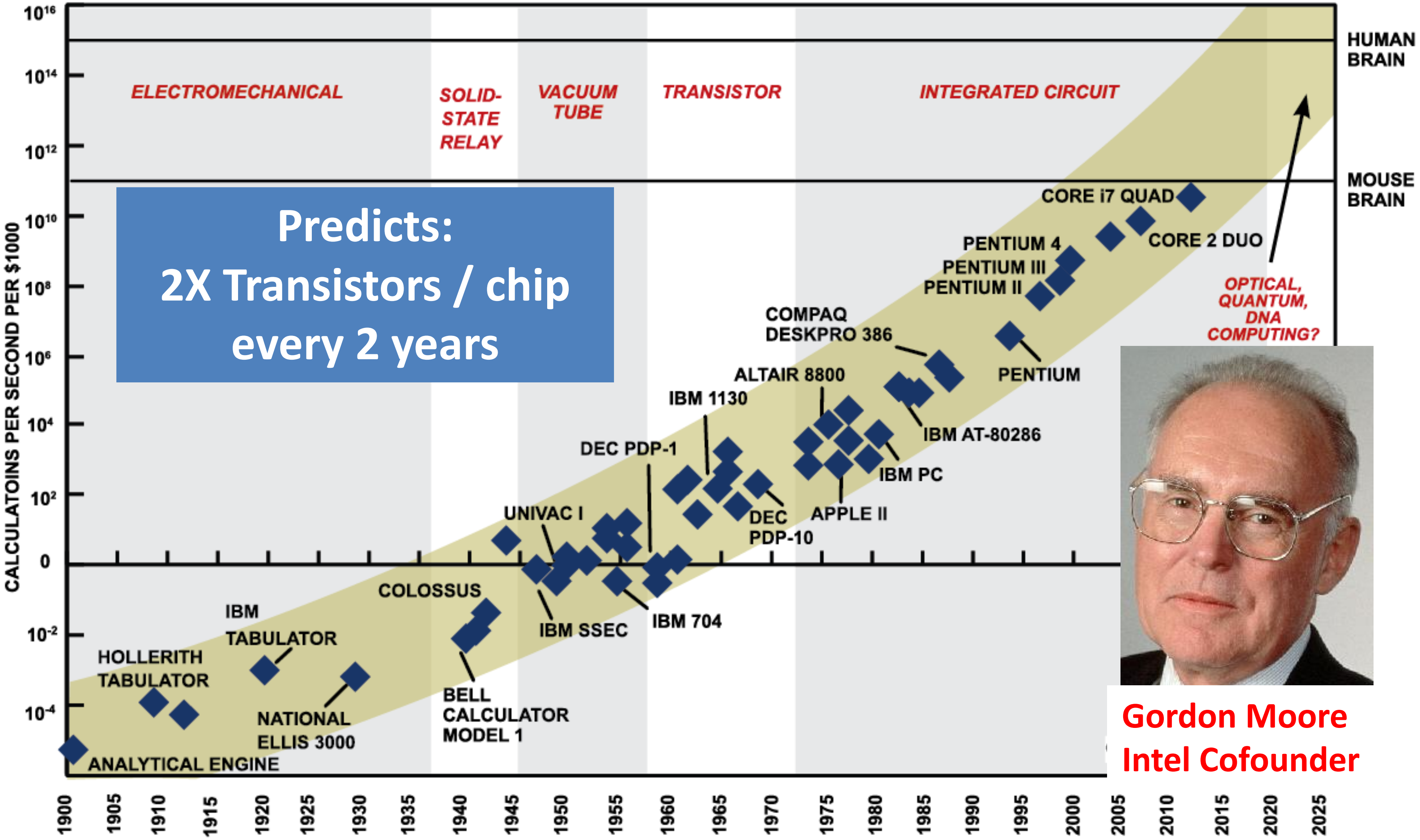
Harness Parallelism & Achieve High Performance



6 Great Ideas in Computer Architecture

1. Abstraction
   (Layers of Representation/Interpretation)
2. Moore’s Law (Designing through trends)
3. Principle of Locality (Memory Hierarchy)
4. Parallelism
5. Performance Measurement & Improvement
6. Dependability via Redundancy

#2: Moore's Law

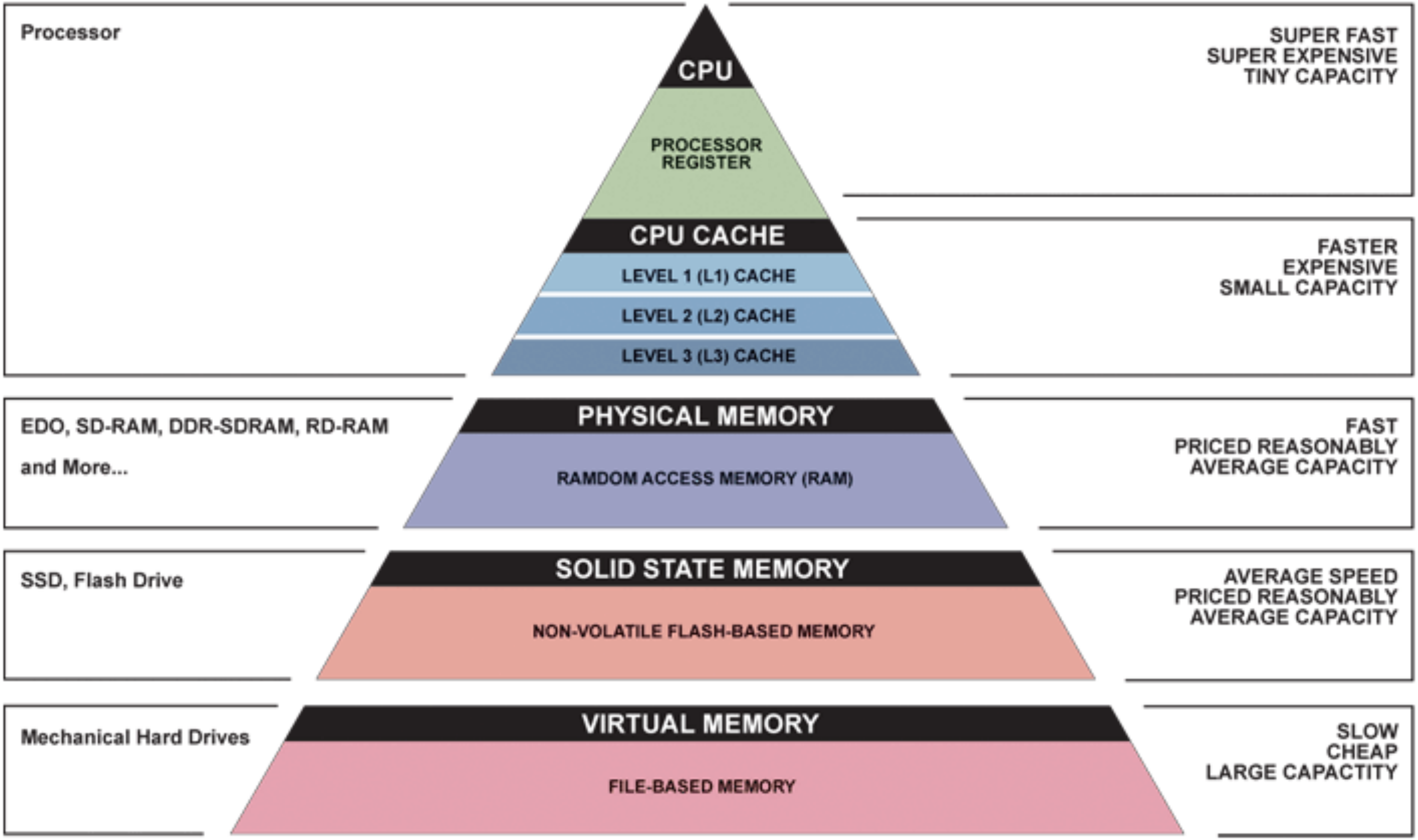


Predicts:
2X Transistors / chip
every 2 years

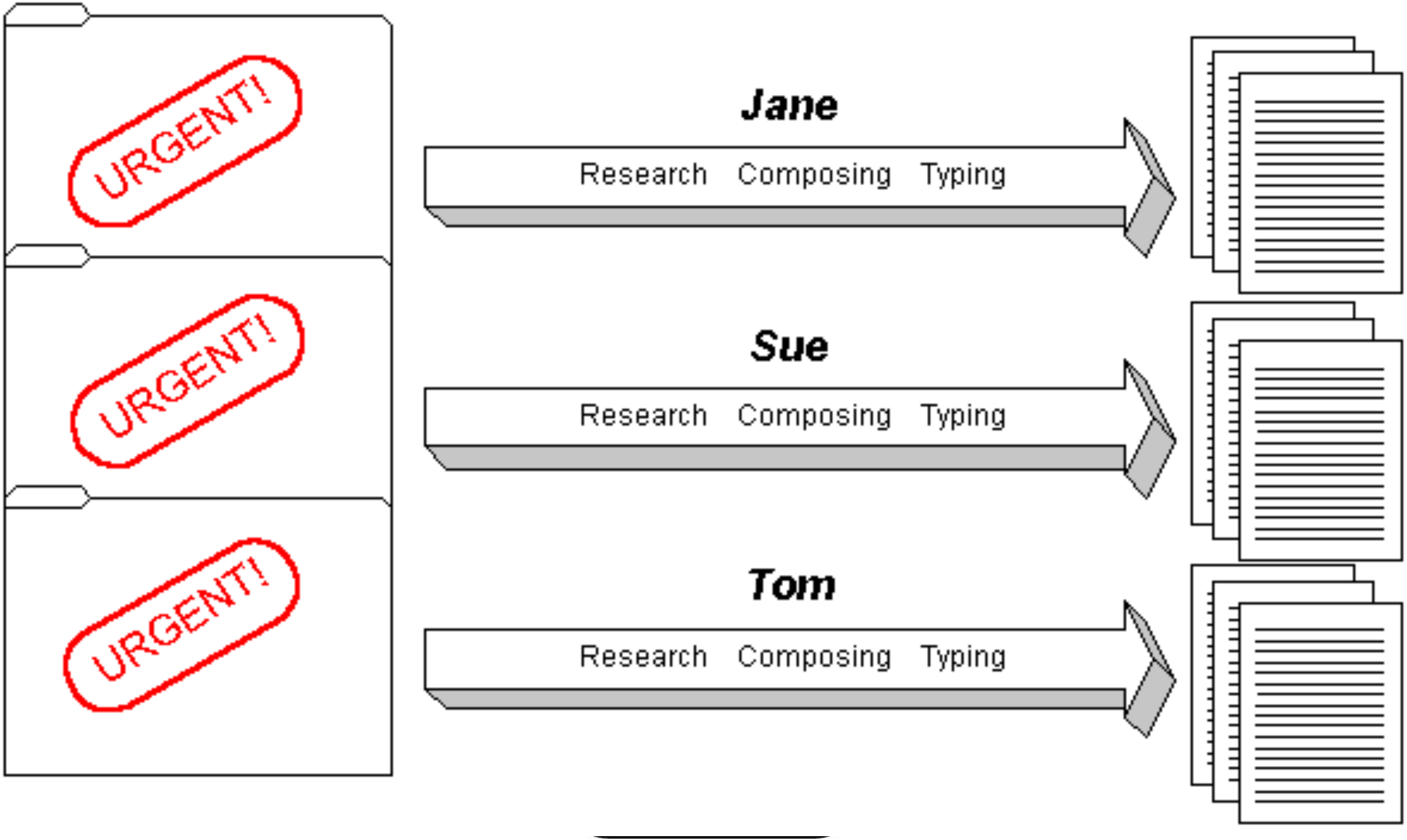
Gordon Moore
Intel Cofounder

SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, *THE VIKING PRESS*, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

Great Idea #3: Principle of Locality/ Memory Hierarchy



Great Idea #4: Parallelism

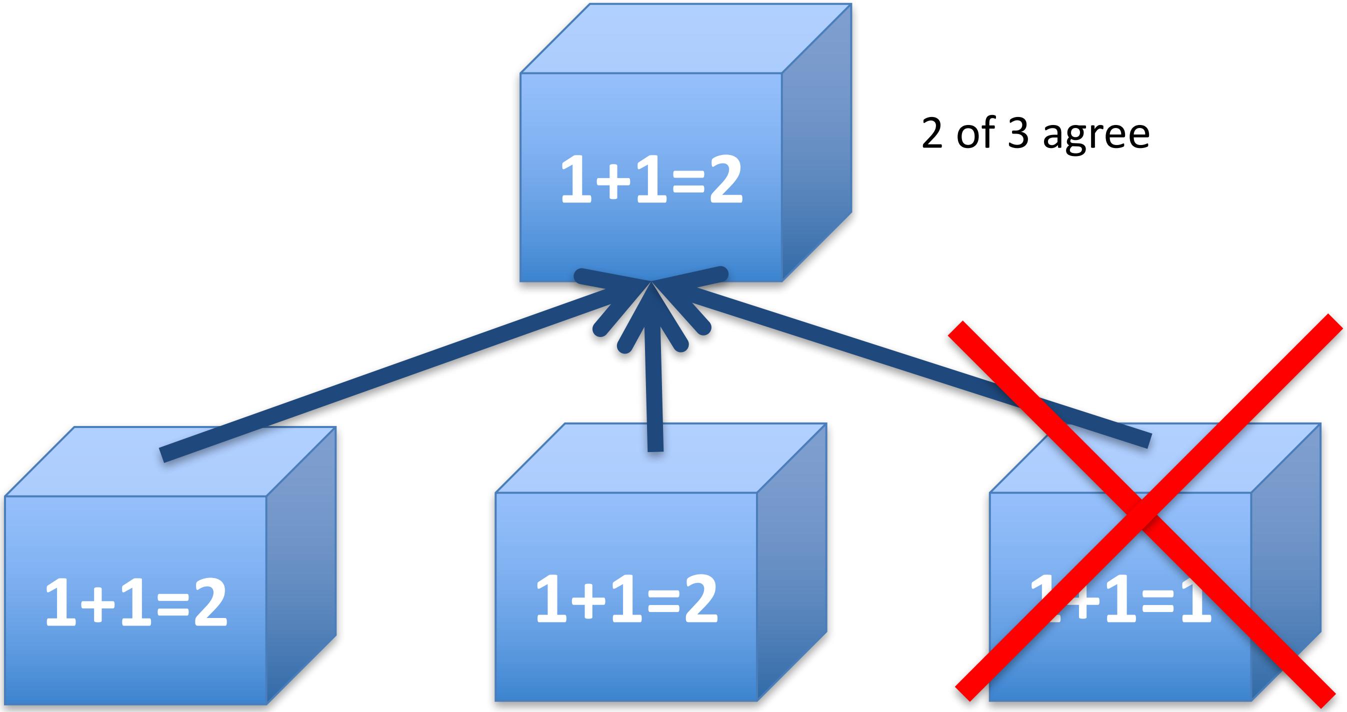

URGENT!
URGENT!
URGENT!
Jane
Research Composing Typing
Sue
Research Composing Typing
Tom
Research Composing Typing

Great Idea #5: Performance Measurement and Improvement

- Tuning application to underlying hardware to exploit:
  - Locality
  - Parallelism
  - Special hardware features, like specialized instructions (e.g., matrix manipulation)
- Latency
  - How long to set the problem up
  - How much faster does it execute once it gets going
  - It is all about *time to finish*

Great Idea #6:
Dependability via Redundancy

- Redundancy so that a failing piece doesn't make the whole system fail



2 of 3 agree

FAIL!

Increasing transistor density reduces the cost of redundancy

Key Concepts

- Inside computers, everything is a number
- But numbers usually stored with a fixed size
  - 8-bit bytes, 16-bit half words, 32-bit words, 64-bit double words, …
- Integer and floating-point operations can lead to results too big/small to store within their representations: *overflow/underflow*

Number Representation

Number Representation

- Value of i-th digit is $d \times Base^i$ where i starts at 0 and increases from right to left:
- $123_{10} = 1_{10} \text{ x } {10_{10}}^2 + 2_{10} \text{ x } {10_{10}}^1 + 3_{10} \text{ x } {10_{10}}^0$

  $= 1\text{x}100_{10} + 2\text{x}10_{10} + 3\text{x}1_{10}$

  $= 100_{10} + 20_{10} + 3_{10}$

  $= 123_{10}$
- Binary (Base 2), Hexadecimal (Base 16), Decimal (Base 10) different ways to represent an integer
  - We use 1_{two}, 5_{ten}, 10_{hex} to be clearer

    (vs. 1_2, 4_8, 5_{10}, 10_{16})

Number Representation

- Hexadecimal digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- $\text{FFF}_{\text{hex}} = 15_{\text{ten}} \text{x } 16_{\text{ten}}{}^{2} + 15_{\text{ten}} \text{x } 16_{\text{ten}}{}^{1} + 15_{\text{ten}} \text{x } 16_{\text{ten}}{}^{0}$
  $= 3840_{\text{ten}} + 240_{\text{ten}} + 15_{\text{ten}}$
  $= 4095_{\text{ten}}$
- $1111\ 1111\ 1111_{\text{two}} = \text{FFF}_{\text{hex}} = 4095_{\text{ten}}$
- May put blanks every group of binary, octal, or hexadecimal digits to make it easier to parse, like commas in decimal

Signed Integers and Two's-Complement Representation

- Signed integers in C; want ½ numbers <0, want ½ numbers >0, and want one 0
- *Two's complement* treats 0 as positive, so 32-bit word represents 2^{32} integers from -2^{31} (–2,147,483,648) to $2^{31}-1$ (2,147,483,647)
  - Note: one negative number with no positive version
  - Book lists some other options, all of which are worse
  - Every computer uses two's complement today
- *Most-significant bit* (leftmost) is the *sign bit,* since 0 means positive (including 0), 1 means negative
  - Bit 31 is most significant, bit 0 is least significant

Two's-Complement Integers

Sign Bit

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{\text{two}} = 0_{\text{ten}}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{\text{two}} = 1_{\text{ten}}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{\text{two}} = 2_{\text{ten}}$

... ...

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{\text{two}} = 2{,}147{,}483{,}645_{\text{ten}}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{\text{two}} = 2{,}147{,}483{,}646_{\text{ten}}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{\text{two}} = 2{,}147{,}483{,}647_{\text{ten}}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{\text{two}} = -2{,}147{,}483{,}648_{\text{ten}}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{\text{two}} = -2{,}147{,}483{,}647_{\text{ten}}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{\text{two}} = -2{,}147{,}483{,}646_{\text{ten}}$

... ...

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{\text{two}} = -3_{\text{ten}}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{\text{two}} = -2_{\text{ten}}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{\text{two}} = -1_{\text{ten}}$

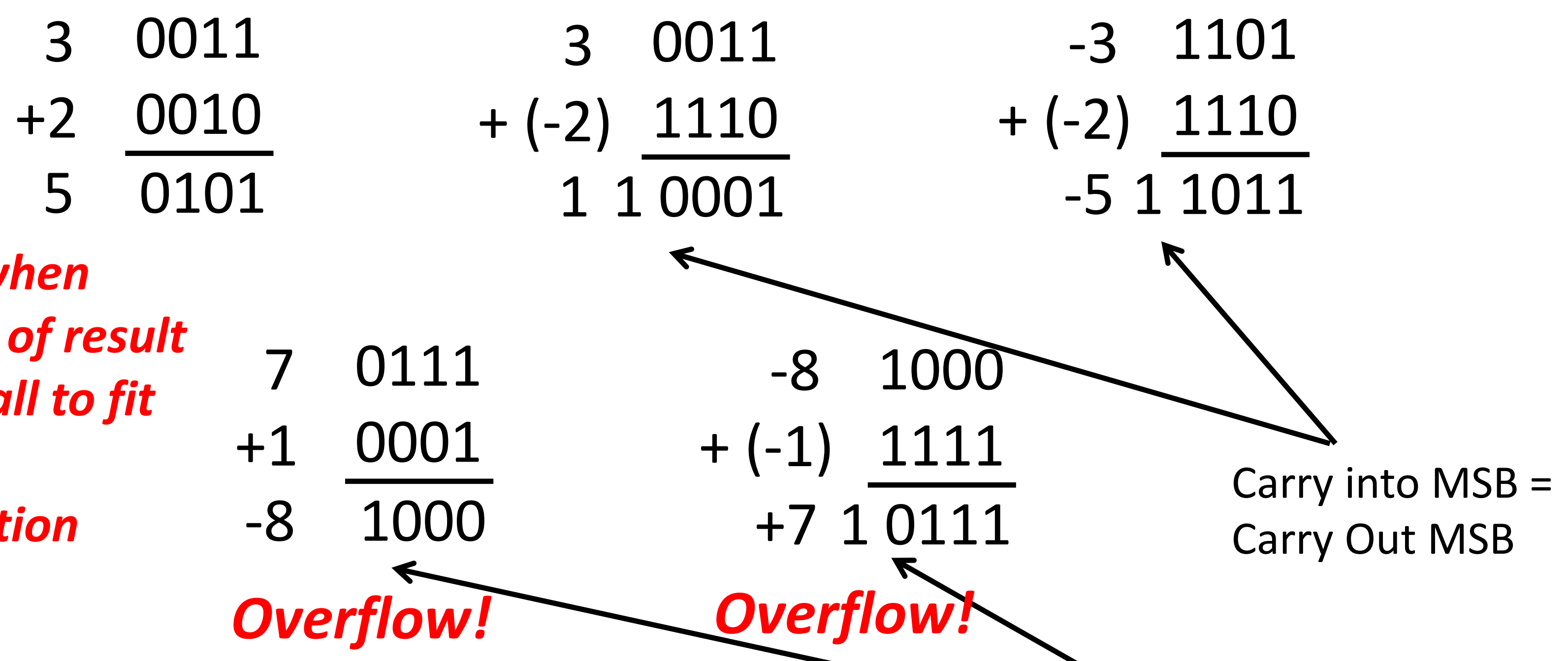
Ways to Make Two's Complement

- For N-bit word, complement to $2_{ten}{}^{N}$
  - For 4 bit number $3_{ten}=0011_{two}$, two's complement (i.e. -3_{ten}) would be $16_{ten}-3_{ten}=13_{ten}$ or $10000_{two} – 0011_{two} = 1101_{two}$
- Here is an easier way:
  - Invert all bits and add 1
  - Computers actually do it like this, too

$$\begin{array}{lr} 3_{ten} & 0011_{two} \\ \text{Bitwise complement} & 1100_{two} \\ & +\quad 1_{two} \\ \hline -3_{ten} & 1101_{two} \end{array}$$

Two's-Complement Examples

- Assume for simplicity 4 bit width, -8 to +7 represented



$$\begin{array}{rr} 3 & 0011 \\ +2 & \underline{0010} \\ 5 & 0101 \end{array} \qquad \begin{array}{rr} 3 & 0011 \\ +(-2) & \underline{1110} \\ 1 & 1\,0001 \end{array} \qquad \begin{array}{rr} -3 & 1101 \\ +(-2) & \underline{1110} \\ -5 & 1\,1011 \end{array}$$

$$\begin{array}{rr} 7 & 0111 \\ +1 & \underline{0001} \\ -8 & 1000 \end{array} \qquad \begin{array}{rr} -8 & 1000 \\ +(-1) & \underline{1111} \\ +7 & 1\,0111 \end{array}$$

Overflow when magnitude of result too big small to fit into result representation

Overflow! ***Overflow!***

Carry into MSB = Carry Out MSB

Carry into MSB ≠ Carry Out MSB

Carry in = carry from less significant bits

Carry out = carry to more significant bits

Suppose we had a 5-bit word. What integers can be represented in two's complement?

□ -32 to +31

□ 0 to +31

□ -16 to +15

□ -15 to +16

Suppose we had a 5-bit word. What integers can be represented in two's complement?

□ -32 to +31

□ 0 to +31

[✓ □ -16 to +15]

□ -15 to +16

C Programming

Quiz: Pointers

```
void foo(int *x, int *y)
{   int t;
    if ( *x > *y ) { t = *y; *y = *x; *x = t; }
}
int a=3, b=2, c=1;
foo(&a, &b);
foo(&b, &c);
foo(&a, &b);
printf("a=%d b=%d c=%d\n", a, b, c);
```

Result is:

A: **a=3 b=2 c=1**

B: **a=1 b=2 c=3**

C: **a=1 b=3 c=2**

D: **a=3 b=3 c=3**

E: **a=1 b=1 c=1**

Arrays and Pointers

```
int
foo(int array[],
    unsigned int size)
{
   …
   printf(“%d\n”, sizeof(array));
}

int
main(void)
{
   int a[10], b[5];
   int c[] = {1, 3, 2, 5, 6};
   … foo(a, 10)… foo(c, 5)  …
   printf(“%d\n”, sizeof(c));
}
```

What does this print (64bit)? **8**

... because **`array`** is really a pointer (and a pointer is architecture dependent, but likely to be 8 on modern machines!)

What does this print? **40**

Quiz:

```
int x[] = { 2, 4, 6, 8, 10 };
int *p = x;
int **pp = &p;
(*pp)++;
(*(*pp))++;
printf("%d\n", *p);
```

Result is:

A: 2

B: 3

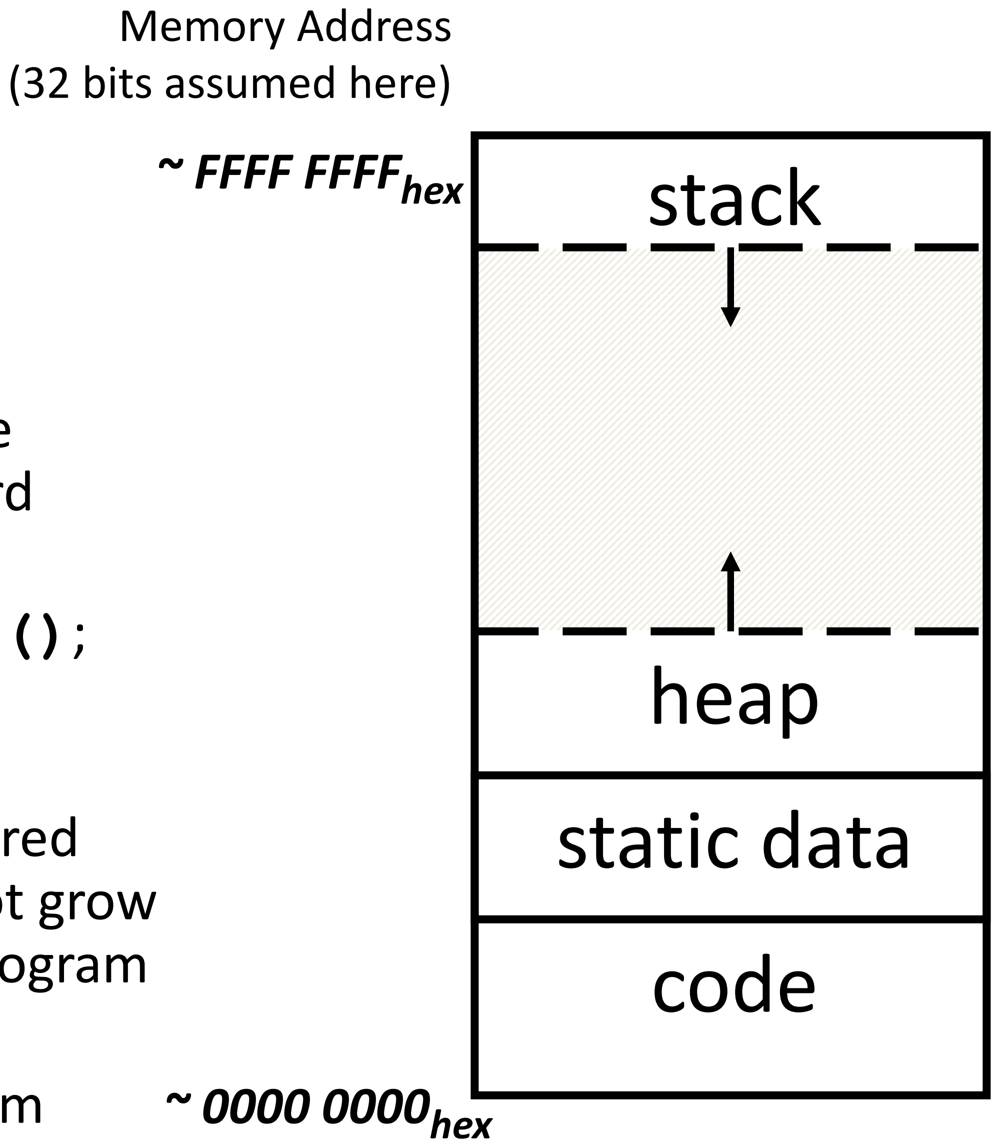
C: 4

D: 5

E: None of the above

C Memory Management

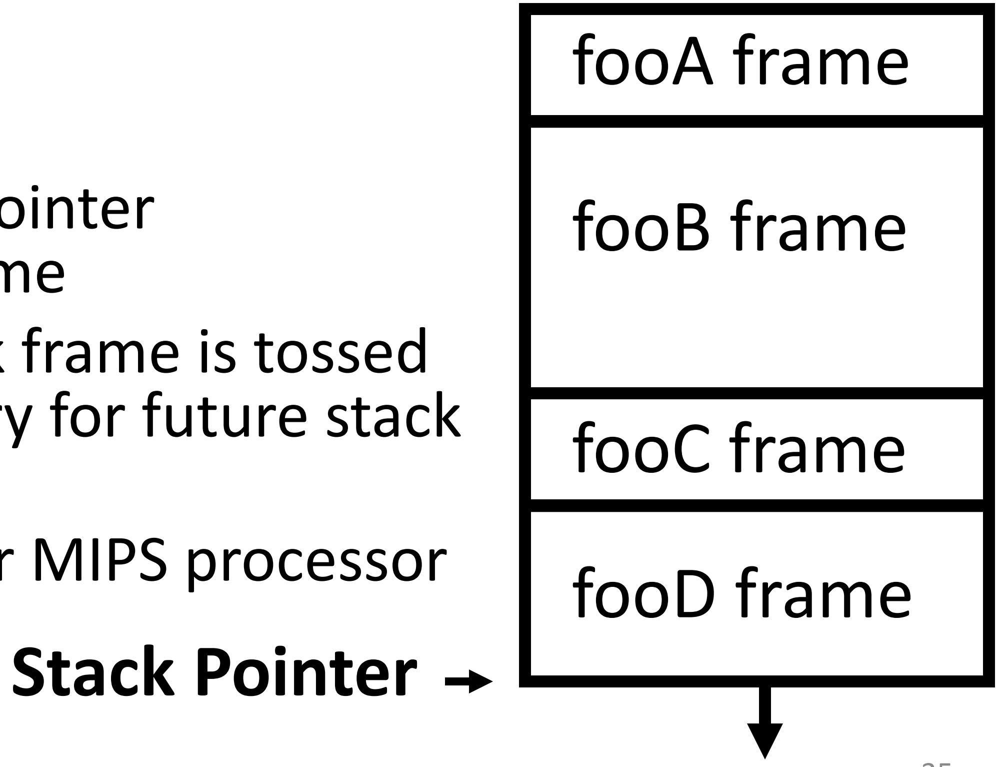
- Program's *address space* contains 4 regions:
  - stack: local variables inside functions, grows downward
  - heap: space requested for dynamic data via **`malloc()`**; resizes dynamically, grows upward
  - static data: variables declared outside functions, does not grow or shrink. Loaded when program starts, can be modified.
  - code: loaded when program starts, does not change



The Stack

- Every time a function is called, a new frame is allocated on the stack
- Stack frame includes:
  - Return address (who called me?)
  - Arguments
  - Space for local variables
- Stack frames contiguous blocks of memory; stack pointer indicates start of stack frame
- When function ends, stack frame is tossed off the stack; frees memory for future stack frames
- We'll cover details later for MIPS processor

```
fooA() { fooB(); }
fooB() { fooC(); }
fooC() { fooD(); }
```



Faulty Heap Management

- What is wrong with this code?
- Memory leak!

```
int foo() {
   int *value = malloc(sizeof(int));
   *value = 42;
   return *value;
}
```

Using Memory You Don’t Own

- What is wrong with this code?

```
int* init_array(int *ptr, int new_size) {
  ptr = realloc(ptr, new_size*sizeof(int));
  memset(ptr, 0, new_size*sizeof(int));
  return ptr;
}

int* fill_fibonacci(int *fib, int size) {
  int i;
  init_array(fib, size);
  /* fib[0] = 0; */ fib[1] = 1;
  for (i=2; i<size; i++)
   fib[i] = fib[i-1] + fib[i-2];
  return fib;
}
```

Using Memory You Don’t Own

- Improper matched usage of mem handles

```
int* init_array(int *ptr, int new_size) {
   ptr = realloc(ptr, new_size*sizeof(int));
   memset(ptr, 0, new_size*sizeof(int));
   return ptr;
}
```

Remember: `realloc` may move entire block

```
int* fill_fibonacci(int *fib, int size) {
   int i;
   /* oops, forgot: fib = */ init_array(fib, size);
   /* fib[0] = 0; */ fib[1] = 1;
   for (i=2; i<size; i++)
    fib[i] = fib[i-1] + fib[i-2];
   return fib;
}
```

What if array is moved to new location?

And In Conclusion, …

- Pointers are an abstraction of machine memory addresses
- Pointer variables are held in memory, and pointer values are just numbers that can be manipulated by software
- In C, close relationship between array names and pointers
- Pointers know the type of the object they point to (except void *)
- Pointers are powerful but potentially dangerous

And In Conclusion, …

- C has three main memory segments in which to allocate data:
  - Static Data: Variables outside functions
  - Stack: Variables local to function
  - Heap: Objects explicitly malloc-ed/free-d.
- Heap data is biggest source of bugs in C code

MIPS

Addition and Subtraction of Integers Example 1

- How to do the following C statement?

  a = b + c + d - e; a = ((b + c) + d) - e;

  b → $s1; c → $s2; d → $s3; e → $s4; a → $s0

- Break into multiple instructions

```
add $t0, $s1, $s2 # temp = b + c
add $t0, $t0, $s3 # temp = temp + d
sub $s0, $t0, $s4 # a = temp - e
```

- A single line of C may break up into several lines of MIPS.
- Notice the use of temporary registers – don’t want to modify the variable registers `$s`
- Everything after the hash mark on each line is ignored (comments)

Overflow handling in MIPS

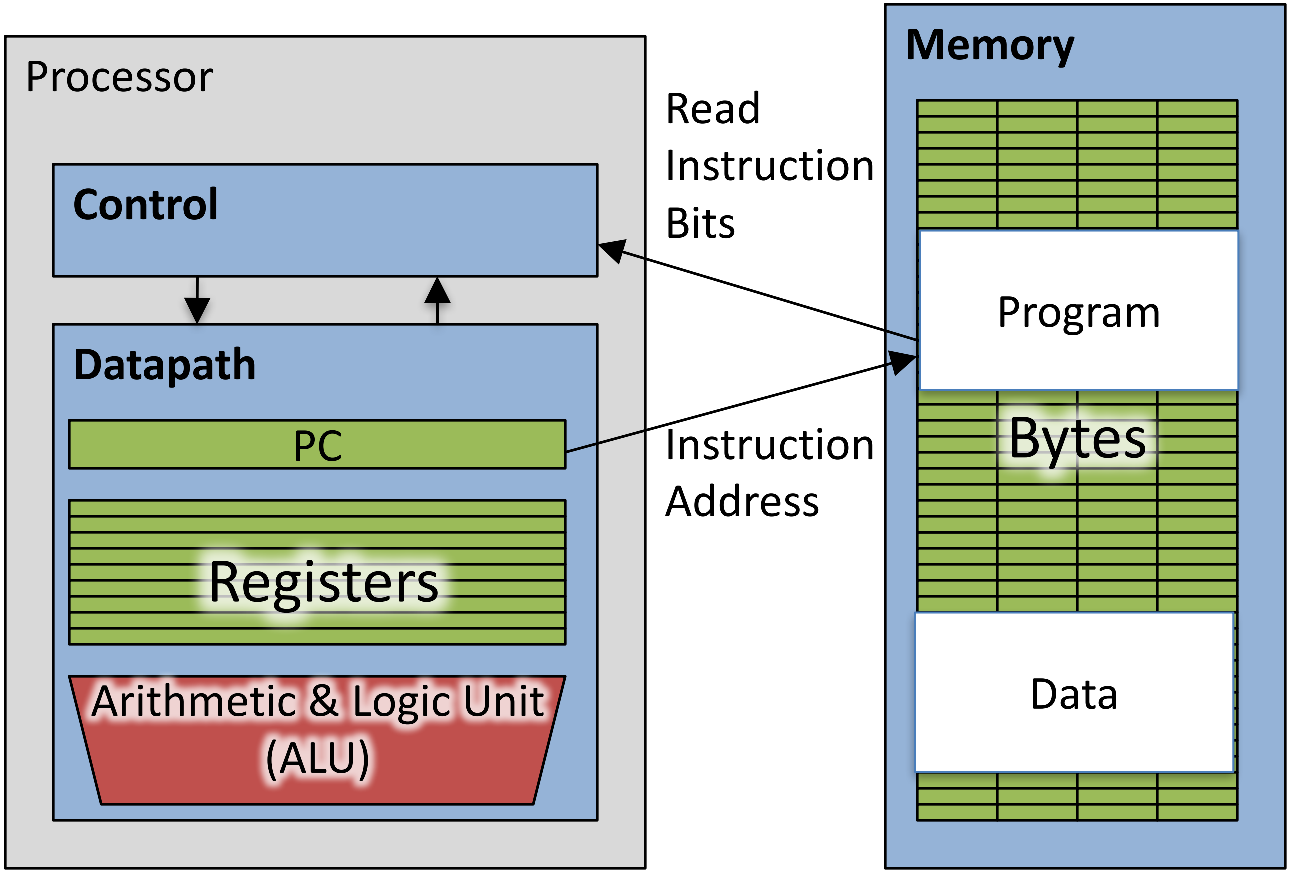
- Some languages detect overflow (Ada), some don't (most C implementations)
- MIPS solution is 2 kinds of arithmetic instructions:
  - These cause overflow to be detected
    - add (add)
    - add immediate (addi)
    - subtract (sub)
  - These do not cause overflow detection
    - add unsigned (addu)
    - add immediate unsigned (addiu)
    - subtract unsigned (subu)
- Compiler selects appropriate arithmetic
  - MIPS C compilers produce addu, addiu, subu

Question:

We want to translate `*x = *y +1` into MIPS
(x, y int pointers stored in: `$s0 $s1`)

```
A:    addi $s0,$s1,1

B:    lw   $s0,1($s1)
      sw   $s1,0($s0)

C:    lw   $t0,0($s1)
      addi $t0,$t0,1
      sw   $t0,0($s0)


D:    sw   $t0,0($s1)
      addi $t0,$t0,1
      lw   $t0,0($s0)

E:    lw   $s0,1($t0)
      sw   $s1,0($t0)
```

Executing a Program



- The PC (program counter) is internal register inside processor holding byte address of next instruction to be executed.
- Instruction is fetched from memory, then control unit executes instruction using datapath and memory system, and updates program counter (default is add +4 bytes to PC, to move to next sequential instruction)

Question!

```
        addi $s0,$zero,0
Start:  slt  $t0,$s0,$s1
        beq  $t0,$zero,Exit
        sll  $t1,$s0,2
        addu $t1,$t1,$s5
        lw   $t1,0($t1)
        add  $s4,$s4,$t1
        addi $s0,$s0,1
        j Start
Exit:
```

What is the code above?

A: while loop

B: do … while loop

C: for loop

D: A or C

E: Not a loop

MIPS Function Call Conventions

- Registers faster than memory, so use them
- `$a0–$a3`: four *argument* registers to pass parameters ($4 - $7)
- `$v0,$v1`: two *value* registers to return values ($2,$3)
- `$ra`: one *return address* register to return to the point of origin ($31)

Instruction Support for Functions (1/4)

C

```
... sum(a,b);... /* a,b:$s0,$s1 */
 }
 int sum(int x, int y) {
   return x+y;
 }
```

MIPS

```
address (shown in decimal)
 1000
 1004
 1008
 1012
 1016
 …
 2000
 2004
```

In MIPS, all instructions are 4 bytes, and stored in memory just like data. So here we show the addresses of where the programs are stored.

Instruction Support for Functions (2/4)

C

```
... sum(a,b);... /* a,b:$s0,$s1 */
 }
 int sum(int x, int y) {
   return x+y;
 }
```

MIPS

```
address (shown in decimal)
 1000 add  $a0,$s0,$zero   # x = a
 1004 add  $a1,$s1,$zero   # y = b
 1008 addi $ra,$zero,1016  # $ra=1016
 1012 j    sum             # jump to sum
 1016 …                    # next instruction
 …
 2000 sum: add $v0,$a0,$a1
 2004 jr   $ra    # new instr. “jump register”
```

Instruction Support for Functions (3/4)

C

```
... sum(a,b);... /* a,b:$s0,$s1 */
 }
 int sum(int x, int y) {
   return x+y;
 }
```

- Question: Why use `jr` here? Why not use `j`?
- Answer: `sum` might be called by many places, so we can’t return to a fixed place. The calling proc to `sum` must be able to say “return here” somehow.

M
I
P
S

```
2000 sum: add $v0,$a0,$a1
2004 jr   $ra # new instr. “jump register”
```

Instruction Support for Functions (4/4)

- Single instruction to jump and save return address: jump and link (`jal`)
- Before:

```
1008 addi $ra,$zero,1016  # $ra=1016
1012 j sum                # goto sum
```

- After:

```
1008 jal sum  # $ra=1012,goto sum
```

- Why have a `jal`?
  - Make the common case fast: function calls very common.
  - Don’t have to know where code is in memory with `jal`!

Question

- Which statement is FALSE?

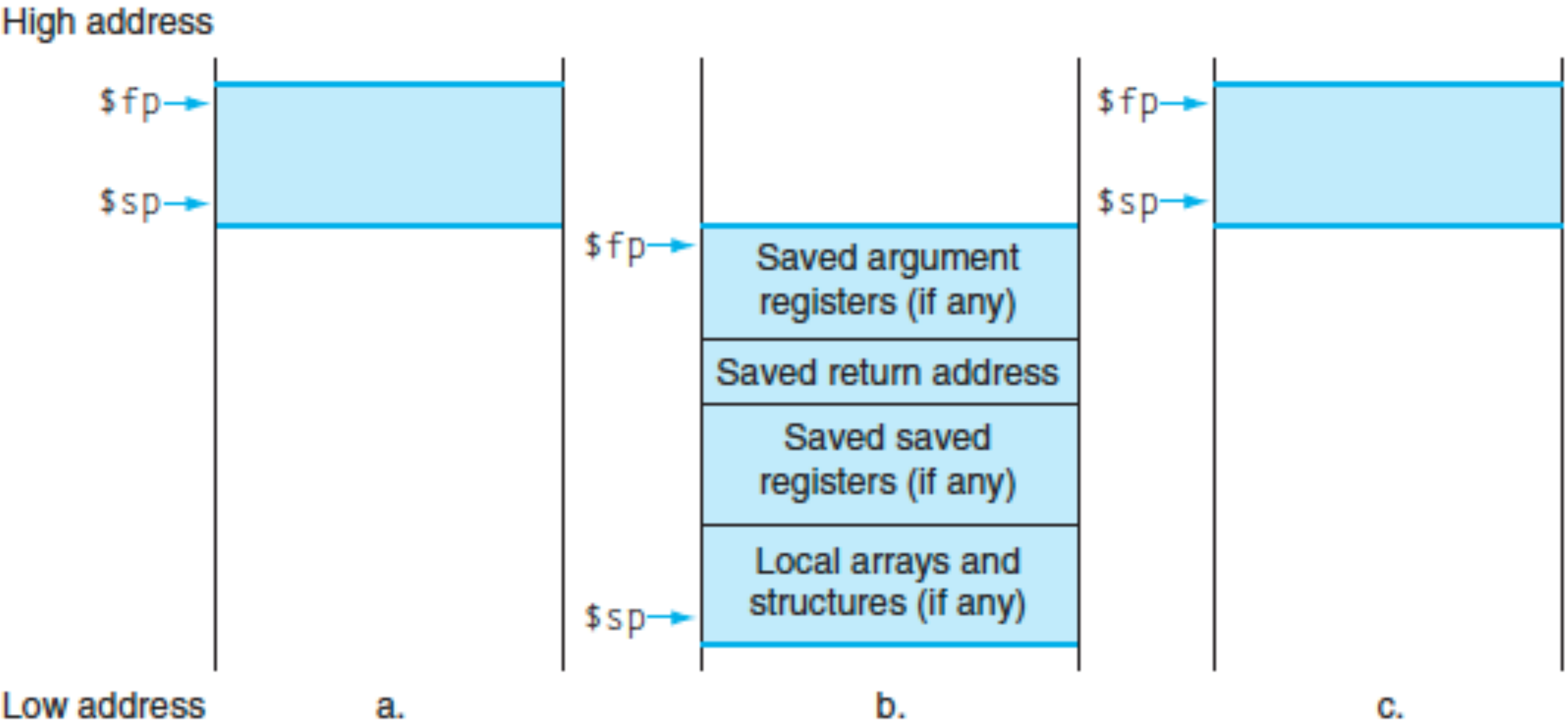
A: MIPS uses `jal` to invoke a function and jr to return from a function

B: `jal` saves PC+1 in `$ra`

C: The callee can use temporary registers (`$ti`) without saving and restoring them

D: The caller can rely on save registers (`$si`) without fear of callee changing them

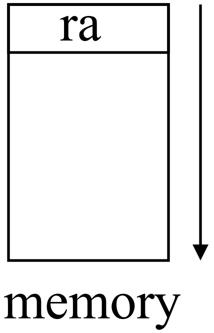
Stack Before, During, After Call


High address
$fp
$sp
$fp
Saved argument registers (if any)
Saved return address
Saved saved registers (if any)
Local arrays and structures (if any)
$sp
$fp
$sp
Low address
a.
b.
c.

Basic Structure of a Function

Prologue

```
entry_label:
addi $sp,$sp, -framesize
sw $ra, framesize-4($sp)   # save $ra
save other regs if need be
```



Body ...   **(call other functions…)**

Epilogue

```
restore other regs if need be
lw $ra, framesize-4($sp)   # restore $ra
addi $sp,$sp, framesize
jr $ra
```

Instruction Formats

- I-format: used for instructions with immediates, `lw` and `sw` (since offset counts as an immediate), and branches (`beq` and `bne`)
  - (but not the shift instructions; later)
- J-format: used for `j` and `jal`
- R-format: used for all other instructions
- It will soon become clear why the instructions have been partitioned in this way

R-Format Instructions (1/5)

- Define “fields” of the following number of bits each: 6 + 5 + 5 + 5 + 5 + 6 = 32

6	5	5	5	5	6

- For simplicity, each field has a name:

`opcode`	`rs`	`rt`	`rd`	`shamt`	`funct`

- Important: On these slides and in book, each field is viewed as a 5- or 6-bit unsigned integer, not as part of a 32-bit integer
  - Consequence: 5-bit fields can represent any number 0-31, while 6-bit fields can represent any number 0-63

I-Format Instructions (2/4)

- Define “fields” of the following number of bits each: 6 + 5 + 5 + 16 = 32 bits

6	5	5	16

  - Again, each field has a name:

`opcode`	`rs`	`rt`	`immediate`

  - Key Concept: Only one field is inconsistent with R-format. Most importantly, **`opcode`** is still in same location.

I-Format Example (2/2)

- MIPS Instruction:

```
addi    $21,$22,-50
```

Decimal/field representation:

8	22	21	-50

Binary/field representation:

001000	10110	10101	1111111111001110

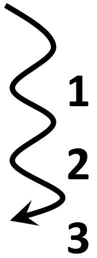
hexadecimal representation: **`22D5 FFCE`**$_{\textbf{hex}}$

Branch Example (1/2)

- MIPS Code:

```
Loop: beq   $9,$0,End
      addu  $8,$8,$10
      addiu $9,$9,-1
      j     Loop
End:
```

Start counting from instruction AFTER the branch



- I-Format fields:

  `opcode` = 4 (look up on Green Sheet)

  `rs` = 9 (first operand)

  `rt` = 0 (second operand)

  `immediate` = 3

Branch Example (2/2)

- MIPS Code:

```
Loop: beq   $9,$0,End
      addu  $8,$8,$10
      addiu $9,$9,-1
      j     Loop
End:
```

Field representation (decimal):

31 0

4	9	0	3

Field representation (binary):

31 0

000100	01001	00000	0000000000000011

J-Format Instructions (2/4)

- Define two “fields” of these bit widths:

31	0
6	26

- As usual, each field has a name:

31	0
`opcode`	`target address`

- **Key Concepts:**
  - Keep `opcode` field identical to R-Format and I-Format for consistency
  - Collapse all other fields to make room for large target address

Summary

- I-Format: instructions with immediates, `lw/sw` (offset is immediate), and `beq/bne`
  - But not the shift instructions
  - Branches use PC-relative addressing

I:	opcode	rs	rt	immediate

- J-Format: `j` and `jal` (but not `jr`)
  - Jumps use absolute addressing

J:	opcode	target address

- R-Format: all other instructions

R:	opcode	rs	rt	rd	shamt	funct

Assembler Pseudo-Instructions

- Certain C statements are implemented unintuitively in MIPS
  - e.g. assignment (`a=b`) via add $zero
- MIPS has a set of “pseudo-instructions” to make programming easier
  - More intuitive to read, but get translated into actual instructions later
- Example:

```
   move dst,src
```

translated into

```
   addi dst,src,0
```

Multiply and Divide

- Example pseudo-instruction:

  `mul $rd,$rs,$rt`

  - Consists of mult which stores the output in special hi and lo registers, and a move from these registers to $rd

  ```
  mult $rs,$rt
  mflo $rd
  ```

- **`mult`** and **`div`** have nothing important in the **`rd`** field since the destination registers are **`hi`** and **`lo`**
- **`mfhi`** and **`mflo`** have nothing important in the **`rs`** and **`rt`** fields since the source is determined by the instruction (see COD)

Question

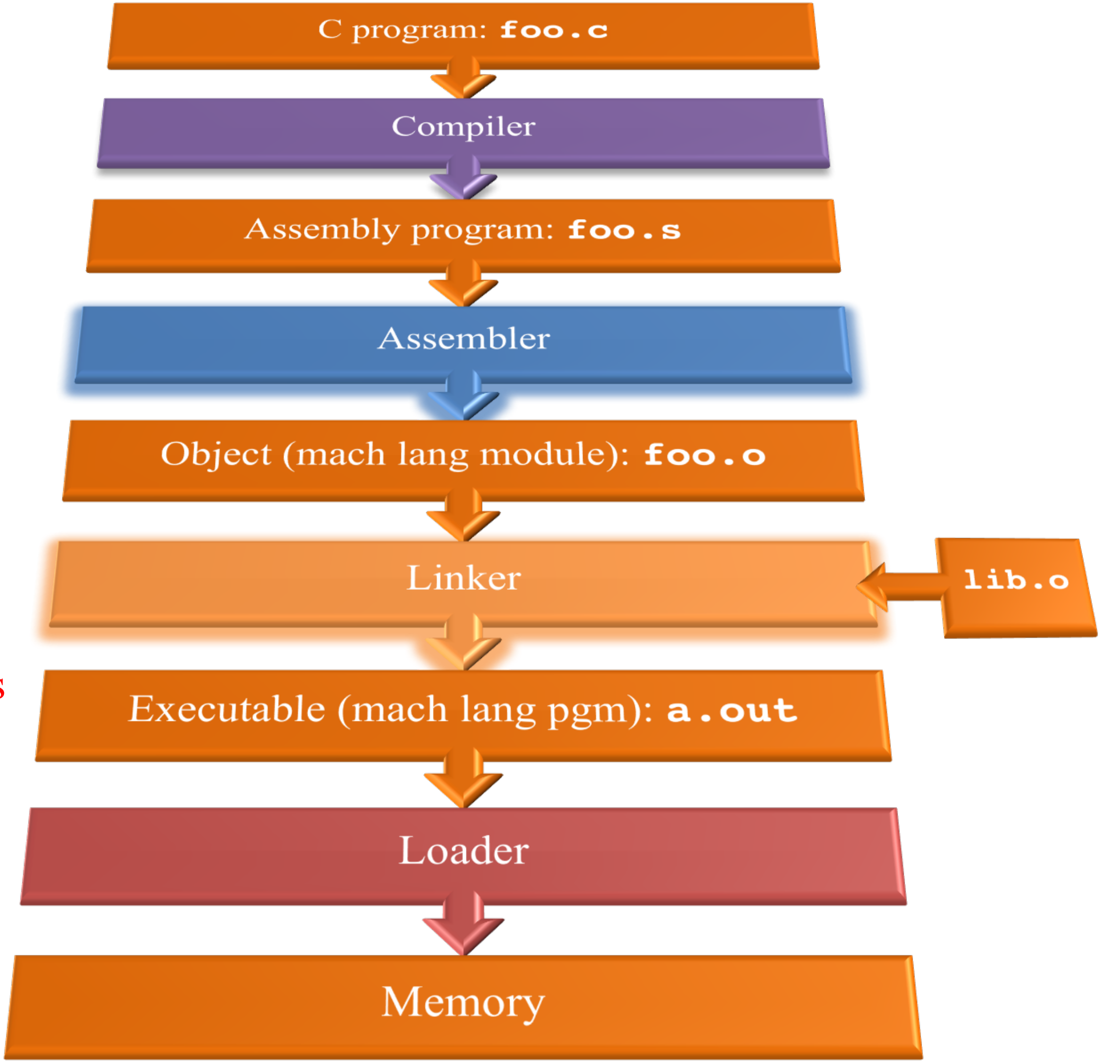
Which of the following place the address of LOOP in $v0?

```
1) la $t1, LOOP
   lw $v0, 0($t1)


2) jal LOOP
   LOOP: addu $v0, $ra, $zero


3) la $v0, LOOP
```

	1	2	3
A)	T,	T,	T
B)	T,	T,	F
C)	F,	T,	T
D)	F,	T,	F
E)	F,	F,	T

Steps in compiling a C program

- Compiler converts a single HLL file into a single assembly language file.
- Assembler removes pseudo-instructions, converts what it can to machine language, and creates a checklist for the linker (relocation table). A `.s` file becomes a `.o` file.
  - Does 2 passes to resolve addresses, handling internal forward references
- Linker combines several `.o` files and resolves absolute addresses.
  - Enables separate compilation, libraries that need not be compiled, and resolves remaining addresses
- Loader loads executable into memory and begins execution.



Pseudo-instruction Replacement

- Assembler treats convenient variations of machine language instructions as if real instructions

```
Pseudo:                 Real:
subu $sp,$sp,32         addiu $sp,$sp,-32
sd $a0, 32($sp)         sw $a0, 32($sp)
                        sw $a1, 36($sp)
mul $t7,$t6,$t5         mult $t6,$t5
                        mflo $t7
addu $t0,$t6,1          addiu $t0,$t6,1
ble $t0,100,loop        slti $at,$t0,101
                        bne $at,$0,loop
la $a0, str             lui $at,left(str)
                        ori $a0,$at,right(str)
```

Question

At what point in process are all the machine code bits generated for the following assembly instructions:

1) `addu $6, $7, $8`

2) `jal fprintf`

A: 1) & 2) After compilation

B: 1) After compilation, 2) After assembly

C: 1) After assembly, 2) After linking

D: 1) After assembly, 2) After loading

E: 1) After compilation, 2) After linking

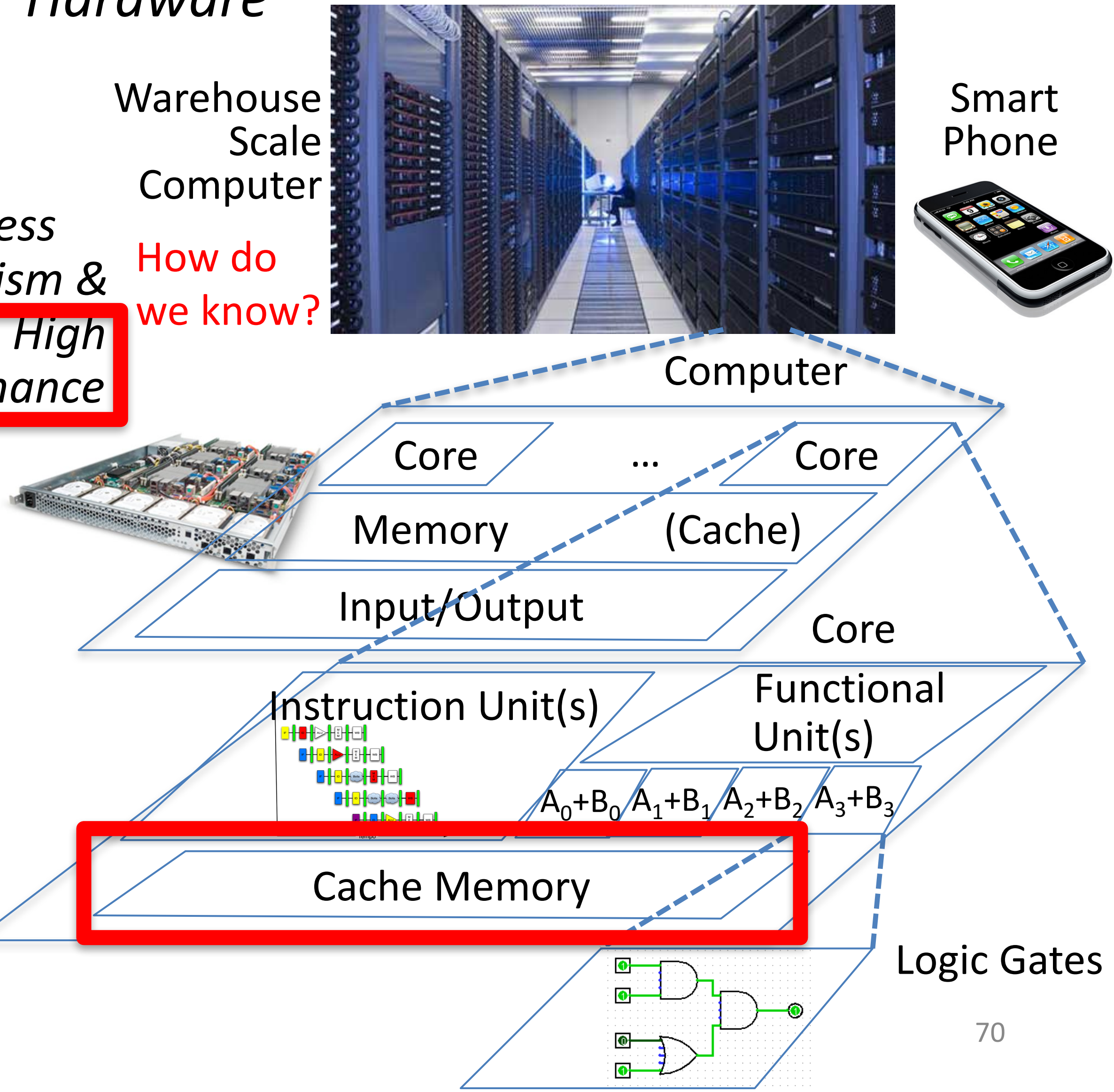
INTRO TO CACHES

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 @ one time
- Programming Languages

Harness Parallelism & Achieve High Performance



Warehouse Scale Computer

How do we know?

Smart Phone

Computer

Core … Core

Memory (Cache)

Input/Output

Core

Instruction Unit(s)

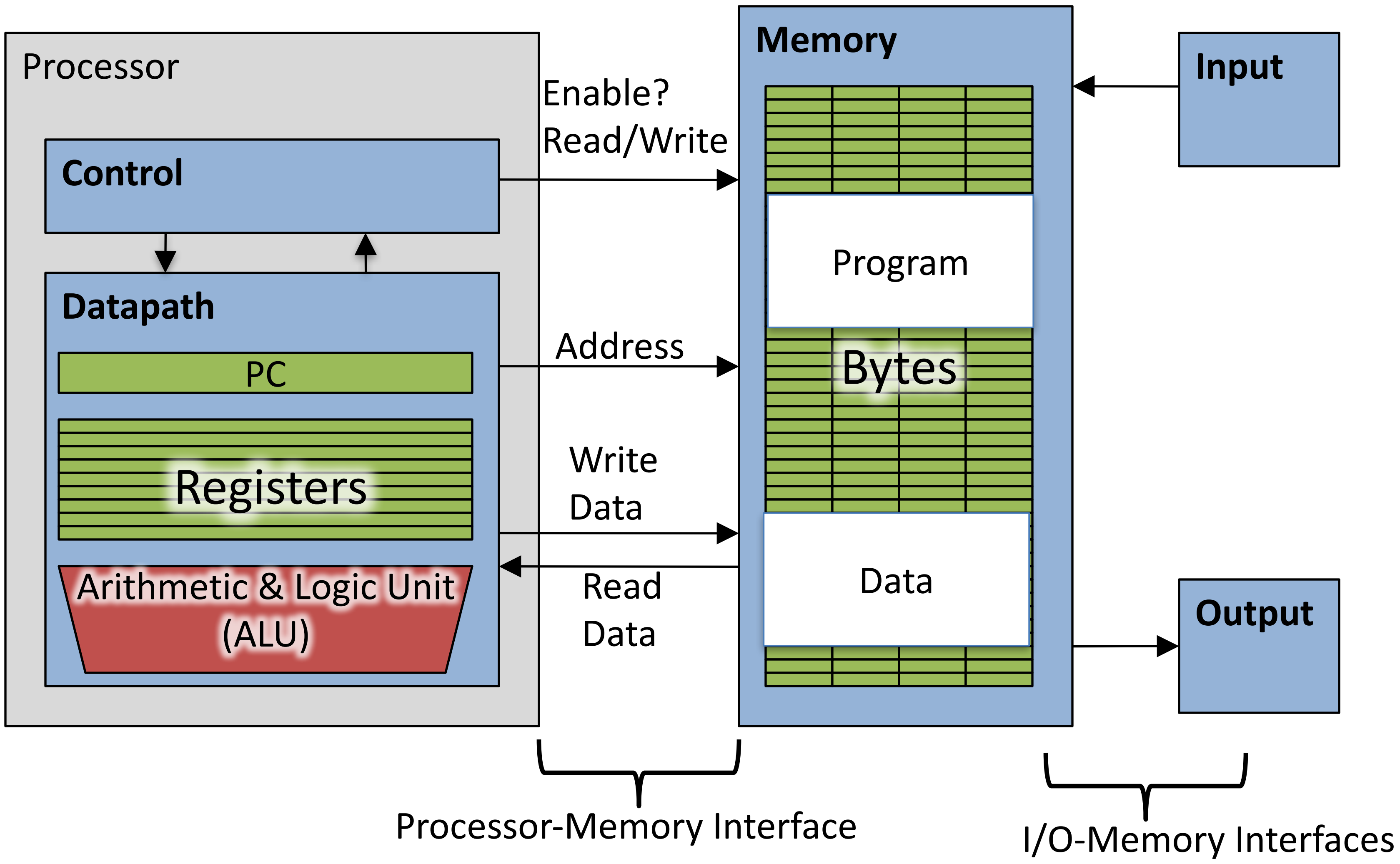
Functional Unit(s)

A_0+B_0 A_1+B_1 A_2+B_2 A_3+B_3

Cache Memory

Logic Gates

Components of a Computer

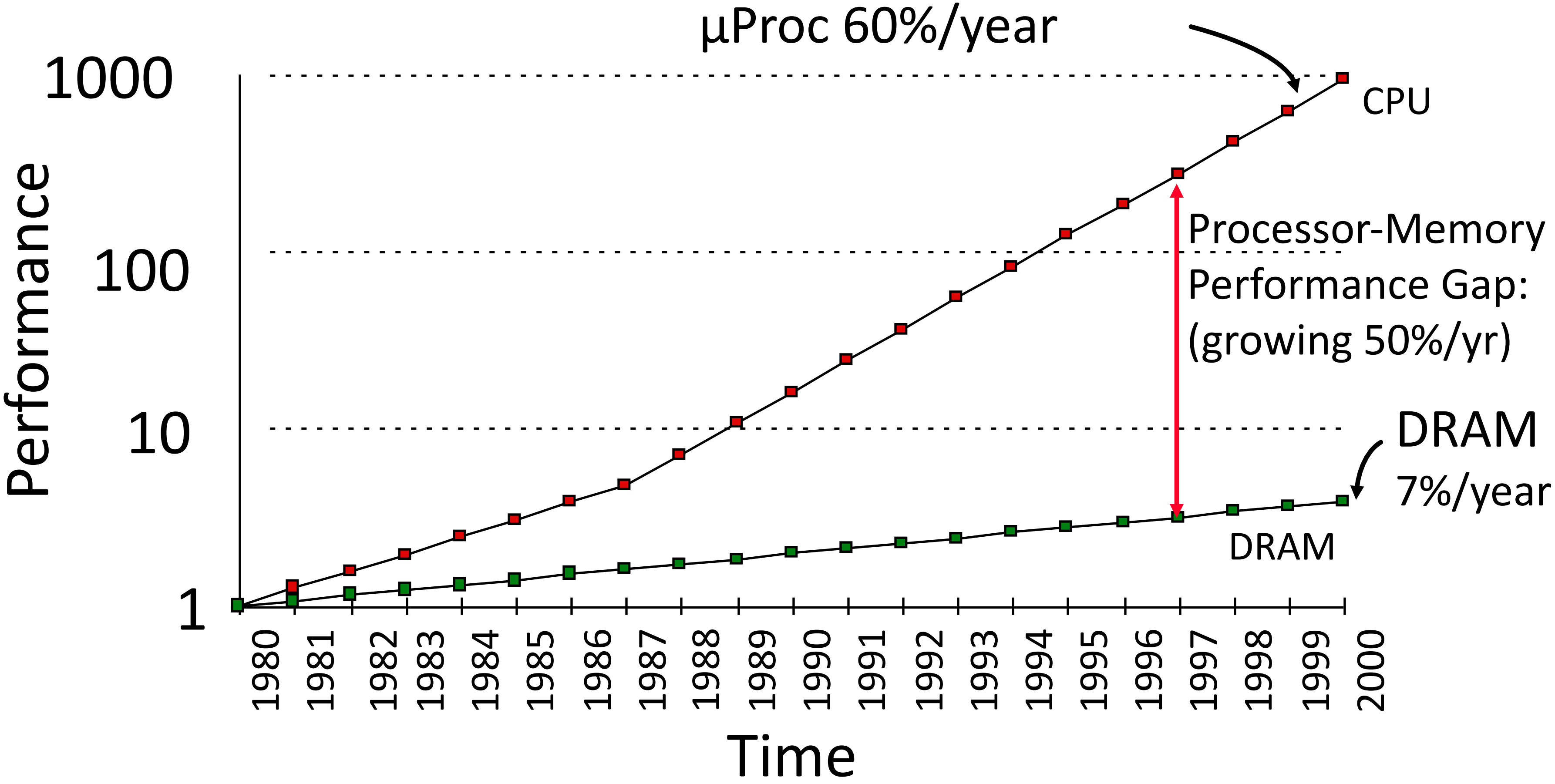


Problem: Large memories slow? Library Analogy

- Finding a book in a large library takes time
  - Takes time to search a large card catalog – (mapping title/author to index number)
  - Round-trip time to walk to the stacks and retrieve the desired book.
- Larger libraries makes both delays worse
- Electronic memories have the same issue, *plus* the technologies that we use to store an individual bit get slower as we increase density (SRAM versus DRAM versus Magnetic Disk)

However what we want is a large yet fast memory!

Processor-DRAM Gap (latency)



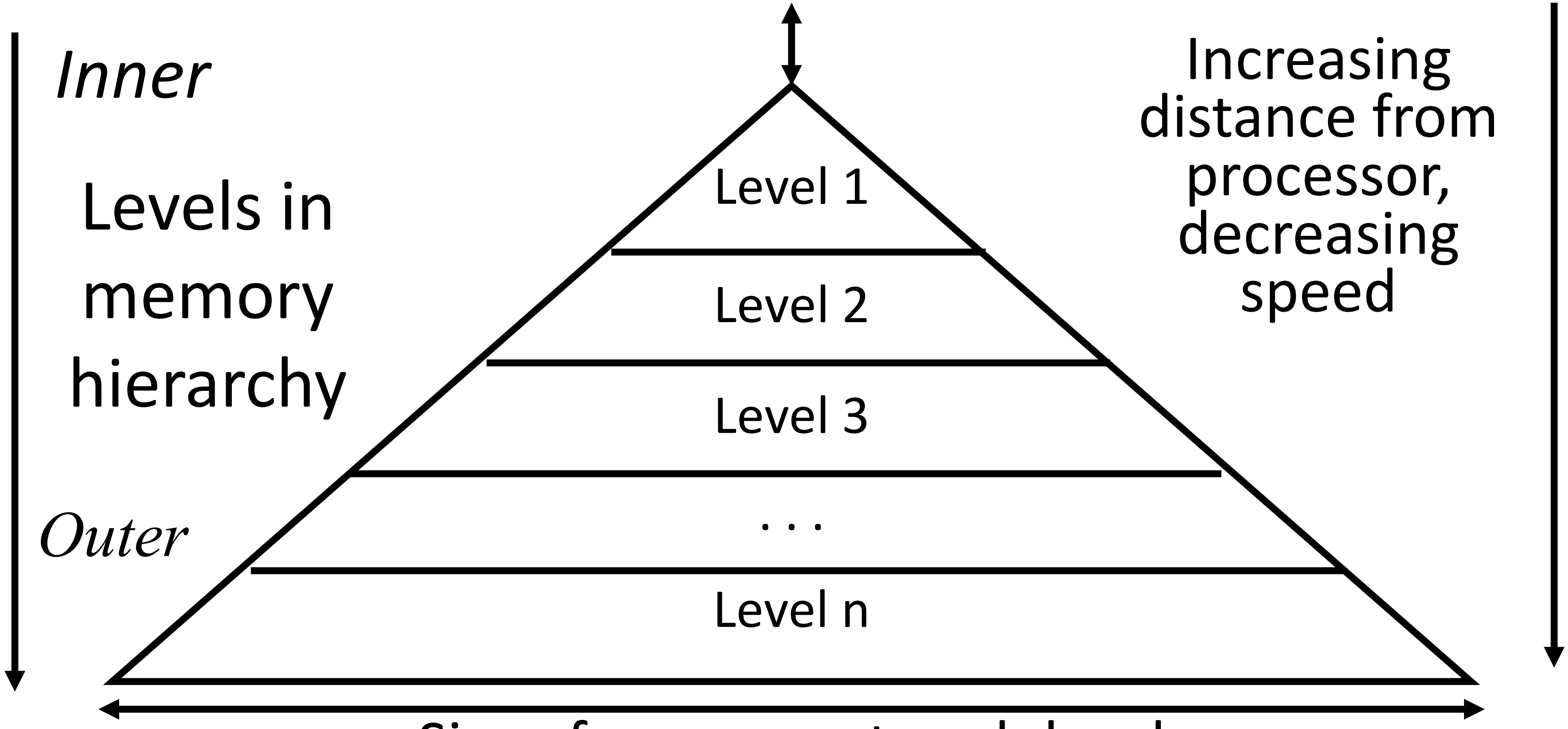
1980 microprocessor executes ~one instruction in same time as DRAM access

2015 microprocessor executes ~1000 instructions in same time as DRAM access

Slow DRAM access could have disastrous impact on CPU performance!

Big Idea: Memory Hierarchy

Processor



Inner

Levels in memory hierarchy

Outer

Level 1

Level 2

Level 3

. . .

Level n

Increasing distance from processor, decreasing speed

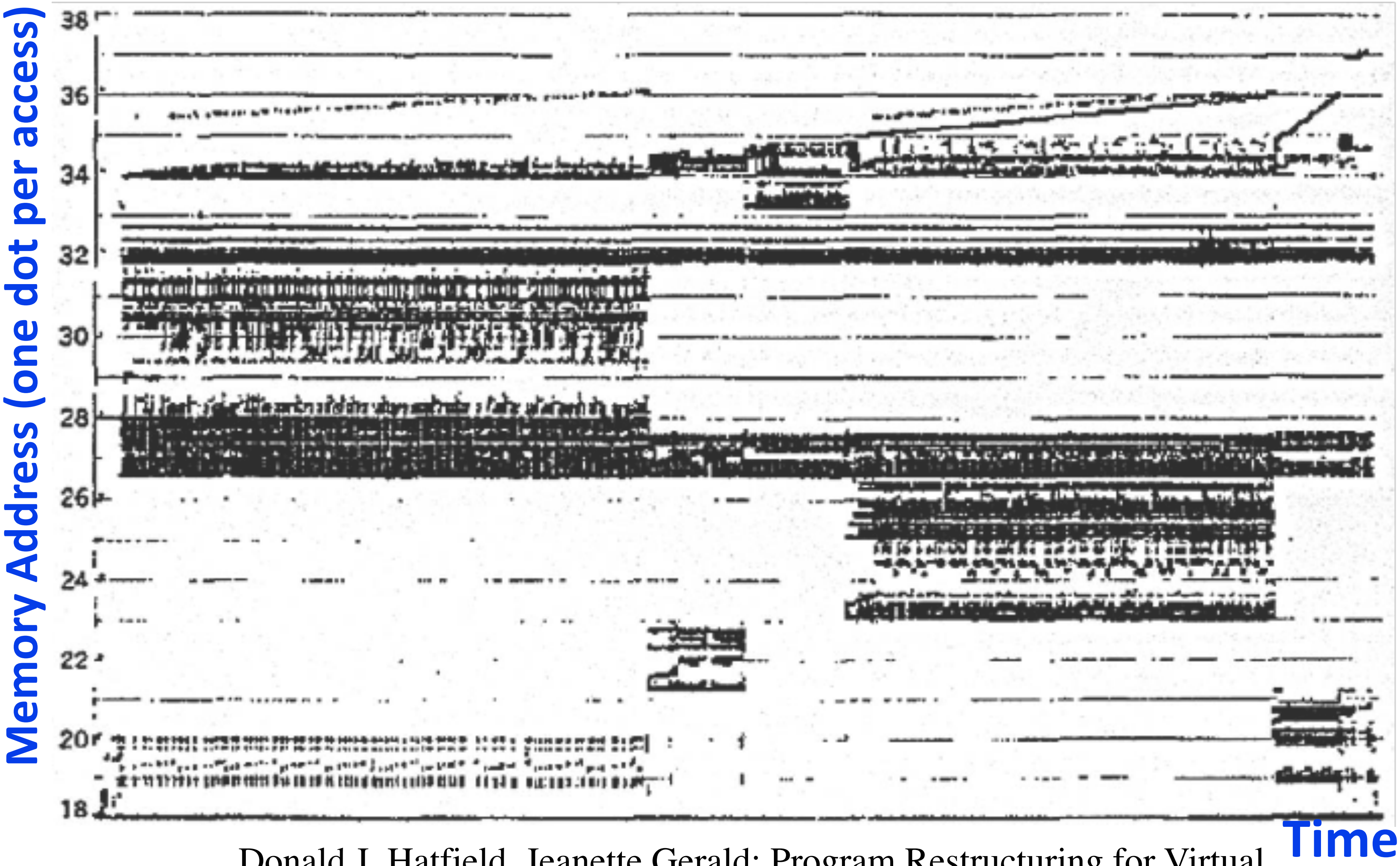
Size of memory at each level

As we move to outer levels the latency goes up and price per bit goes down. Why?

What to do: Library Analogy

- Want to write a report using library books
- Go to library, look up relevant books, fetch from stacks, and place on desk in library
- If need more, check them out and keep on desk
  - But don't return earlier books since might need them
- You hope this collection of ~10 books on desk enough to write report, despite 10 being only a tiny fraction of books available

Real Memory Reference Patterns

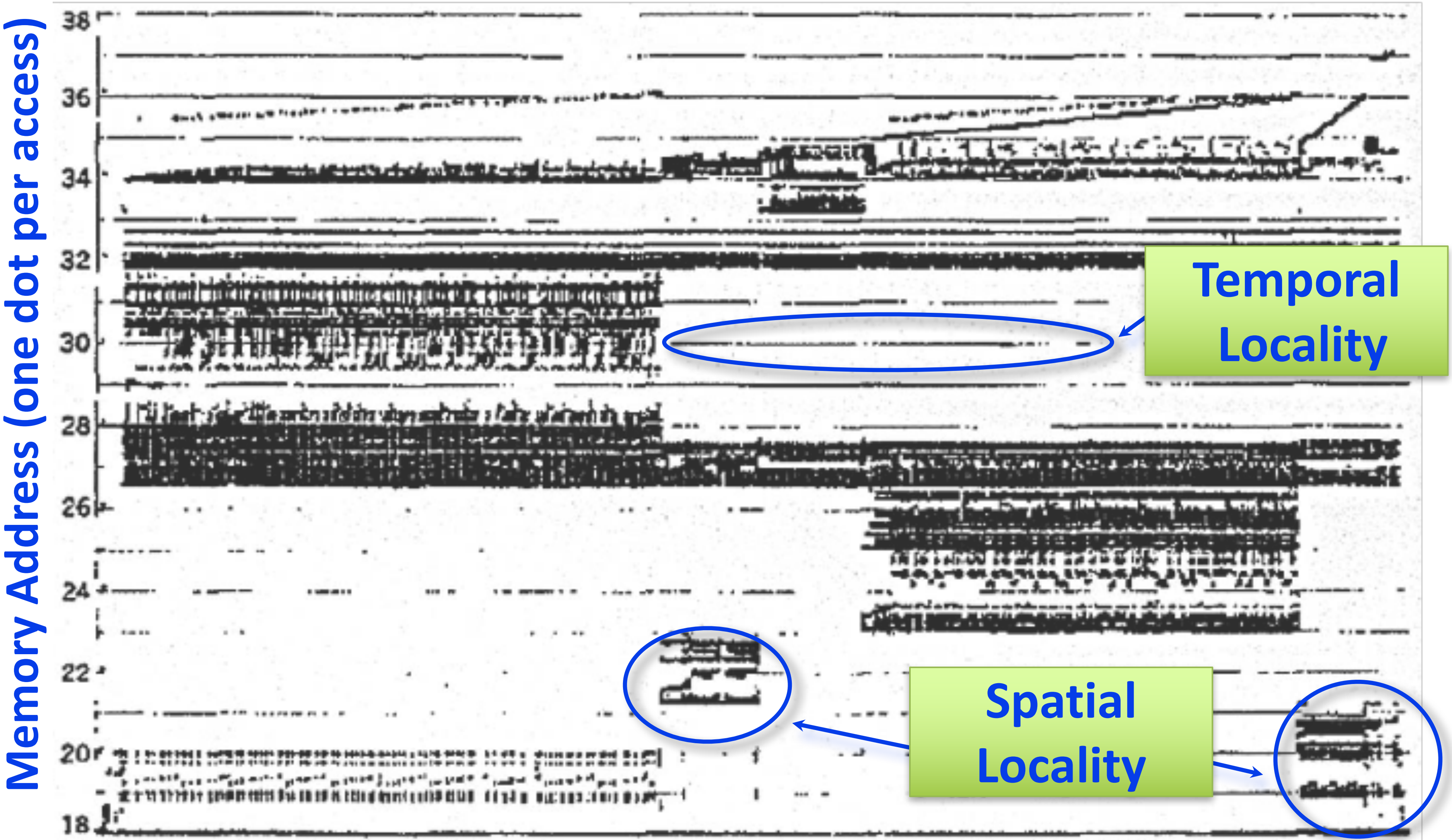


Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Big Idea: Locality

- *Temporal Locality* (locality in time)
  - Go back to same book on desktop multiple times
  - If a memory location is referenced, then it will tend to be referenced again soon
- *Spatial Locality* (locality in space)
  - When go to book shelf, pick up multiple books on J.D. Salinger since library stores related books together
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

Memory Reference Patterns



Temporal Locality

Spatial Locality

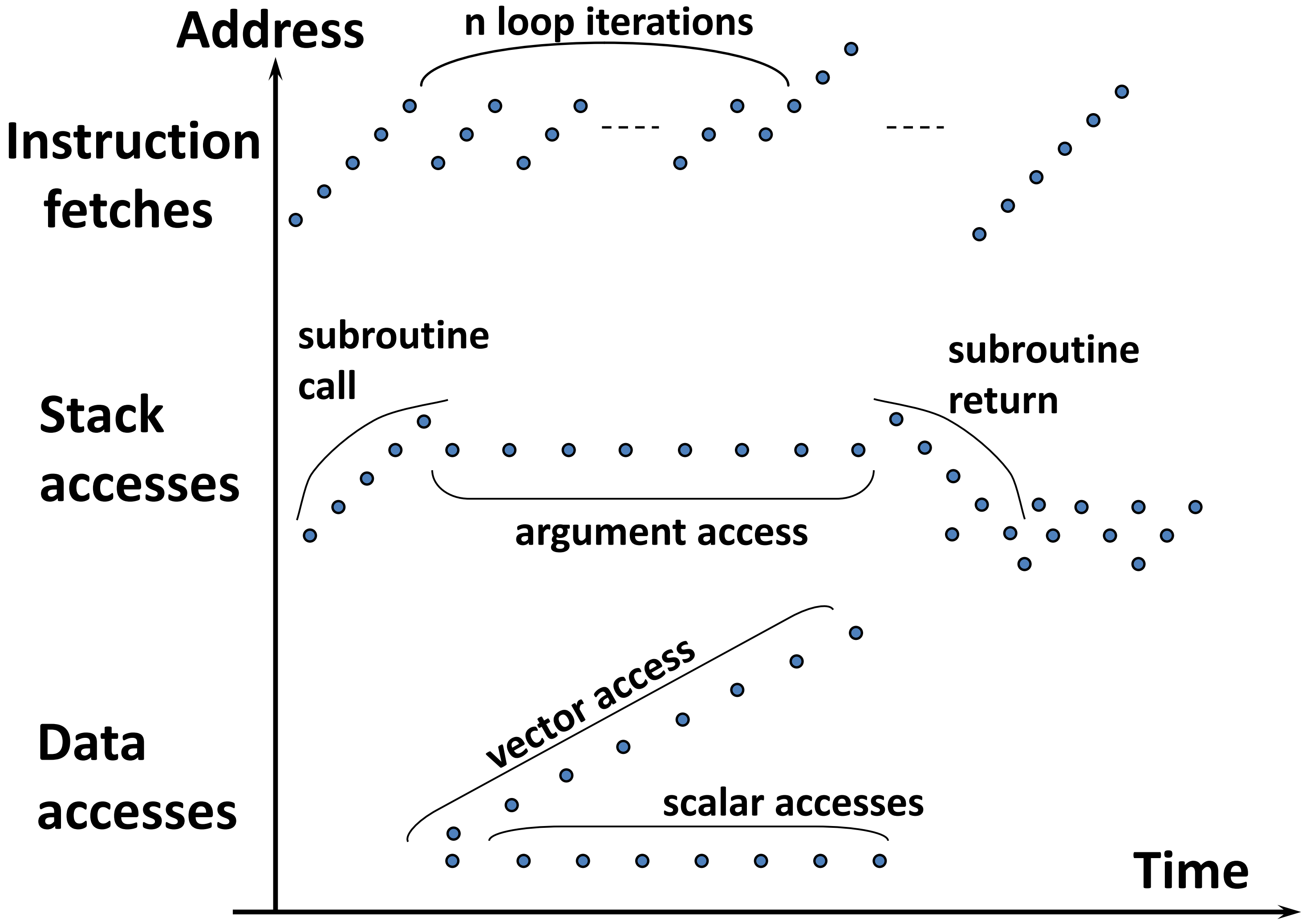
Time

Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Principle of Locality

- *Principle of Locality*: Programs access small portion of address space at any instant of time (spatial locality) and repeatedly access that portion (temporal locality)
- What program structures lead to temporal and spatial locality in **instruction** accesses?
- In **data** accesses?

Memory Reference Patterns



Cache Philosophy

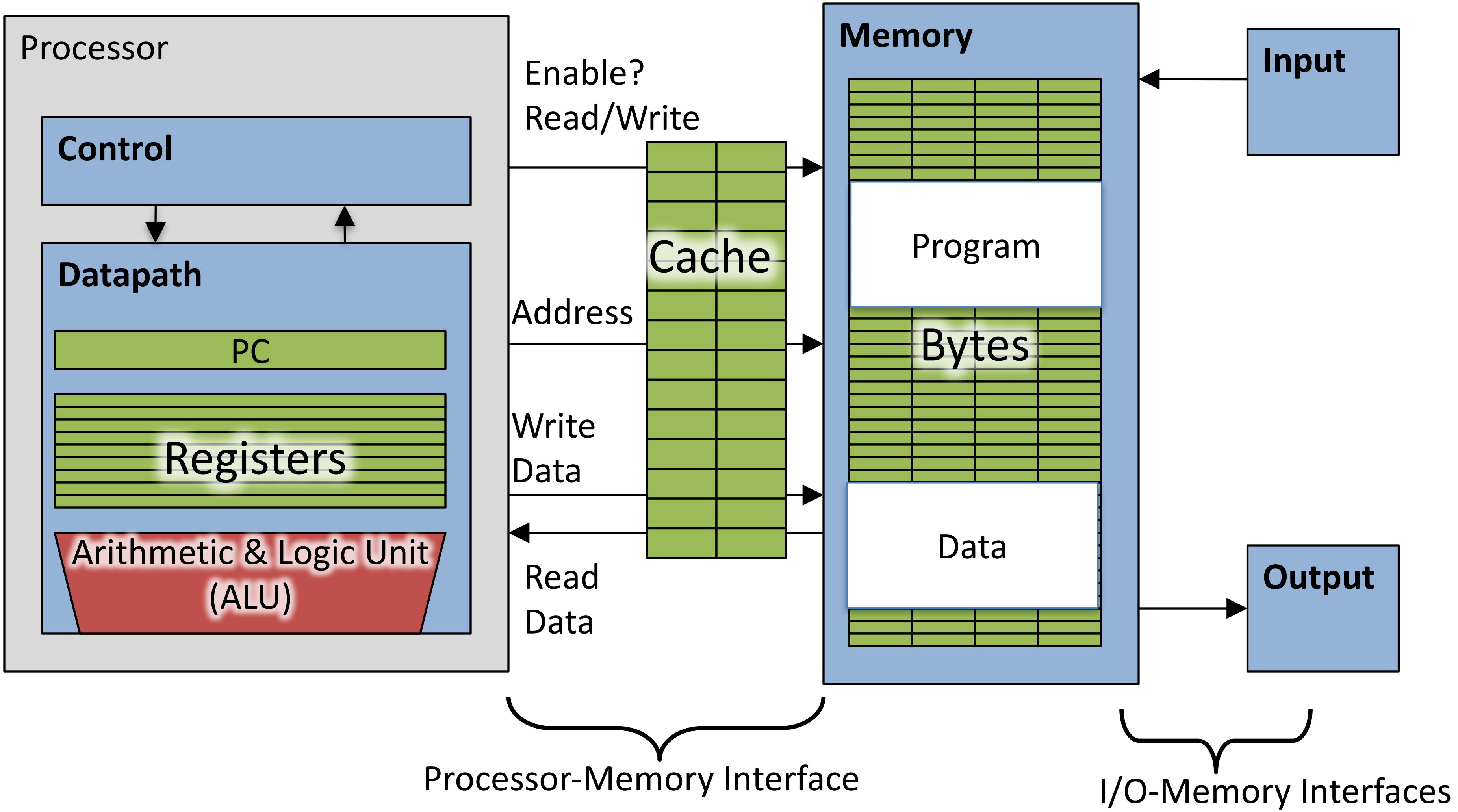
- Programmer-invisible hardware mechanism to give illusion of speed of fastest memory with size of largest memory
  - Works fine even if programmer has no idea what a cache is
  - However, performance-oriented programmers today sometimes “reverse engineer” cache design to design data structures to match cache

Memory Access without Cache

- Load word instruction: `lw $t0,0($t1)`
- $t1 contains 1022_{ten}, Memory[1022] = 99

1. Processor issues address 1022_{ten} to Memory
2. Memory reads word at address 1022_{ten} (99)
3. Memory sends 99 to Processor
4. Processor loads 99 into register $t0

Adding Cache to Computer



Memory Access with Cache

- Load word instruction: `lw $t0,0($t1)`
- $t1 contains $1022_{ten,}$ Memory[1022] = 99
- With cache: Processor issues address 1022_{ten} to Cache

1. Cache checks to see if has copy of data at address 1022_{ten}

   2a. If finds a match (Hit): cache reads 99, sends to processor

   2b. No match (Miss): cache sends address 1022 to Memory

      I. Memory reads 99 at address 1022_{ten}
      II. Memory sends 99 to Cache
      III. Cache replaces word with new 99
      IV. Cache sends 99 to processor

2. Processor loads 99 into register $t0

Cache "Tags"

- Need way to tell if have copy of location in memory so that can decide on hit or miss
- On cache miss, put memory address of block in "tag address" of cache block

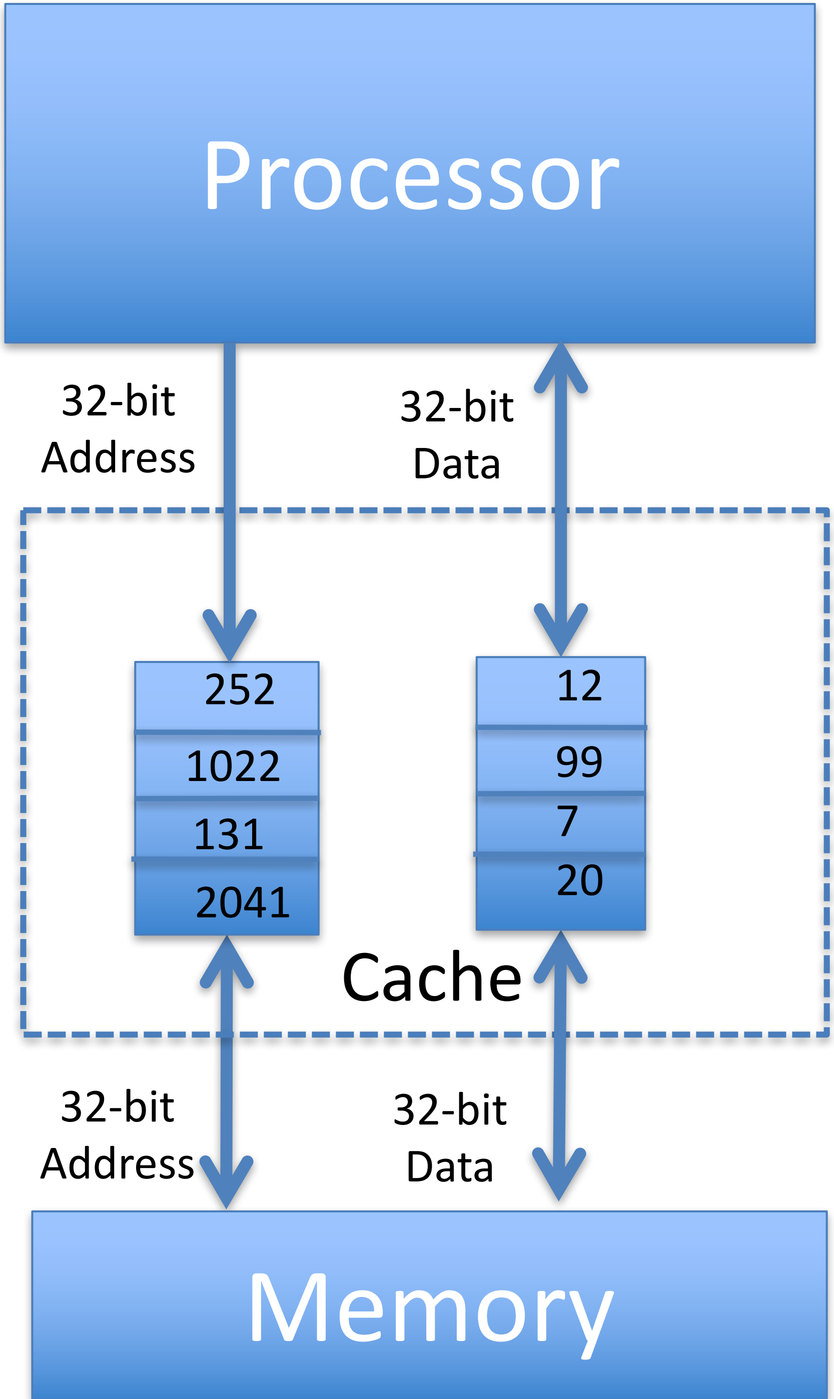
  1022 placed in tag next to data from memory (99)

Tag	Data
252	12
1022	99
131	7
2041	20

From earlier instructions

Anatomy of a 16 Byte Cache, 4 Byte Block

- Operations:
  1. Cache Hit
  2. Cache Miss
  3. Refill cache from memory
- Cache needs Address Tags to decide if Processor Address is a Cache Hit or Cache Miss
  – Compares all 4 tags


Processor
32-bit Address
32-bit Data
252
1022
131
2041
12
99
7
20
Cache
32-bit Address
32-bit Data
Memory