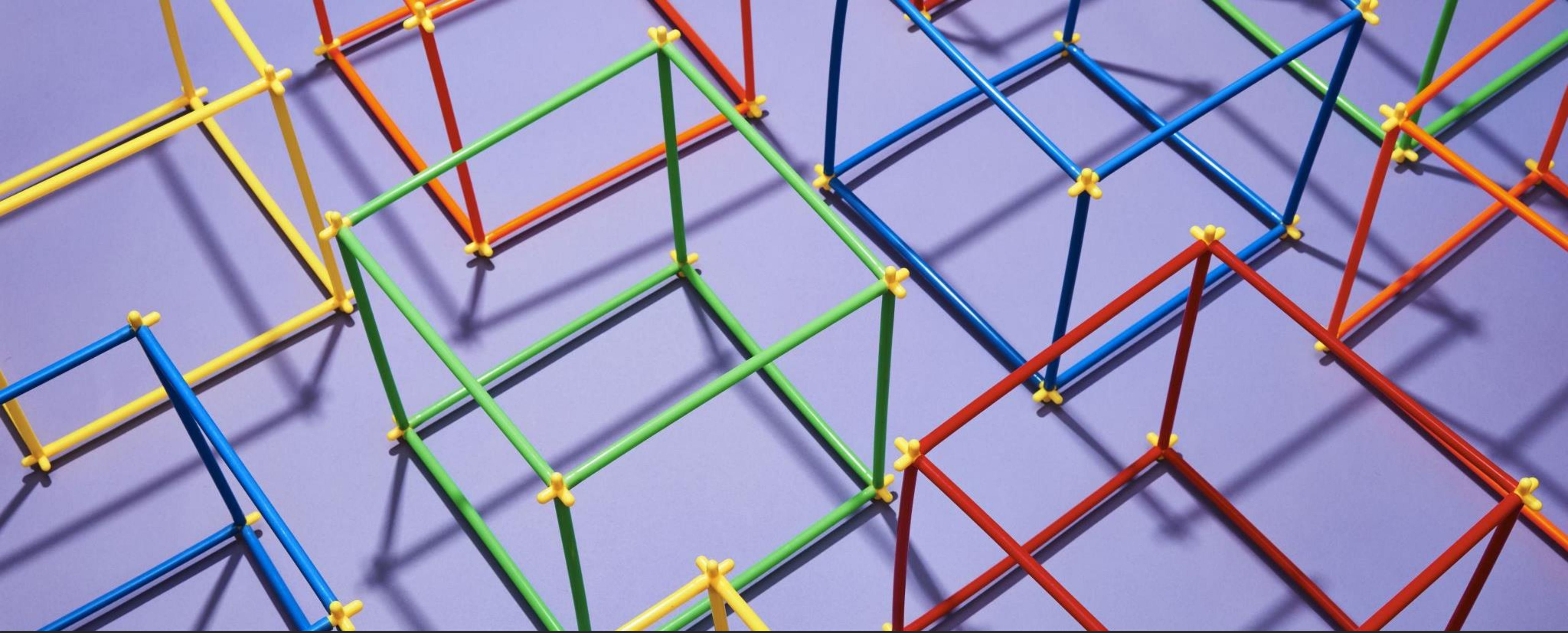


Applications of Data Structures

CSC 143
W2020
BRIAN CUI

The Best Data Structure?

- Array (+ArrayList)
- Linked List (+Doubly, Stack, Queue)
- Tree (+Binary, BST)
- Heap (+Priority Queue)
- HashMap
- Graph

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

bigocheatsheet.com

Humble Beginnings

```
// C (1973 - 2018+)
char[] mybuff = "Hello";
```

0	1	2	3	4	5
'H'	'e'	'l'	'l'	'o'	'\0'

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Before there were Strings, there were char arrays.

Extremely efficient, **zero overhead**

0	1	2	3	4	5
'H'	'e'	'l'	'l'	'o'	'\0'

1 char = 1 byte = 8 bits

→ 2^8 = 256 ASCII chars

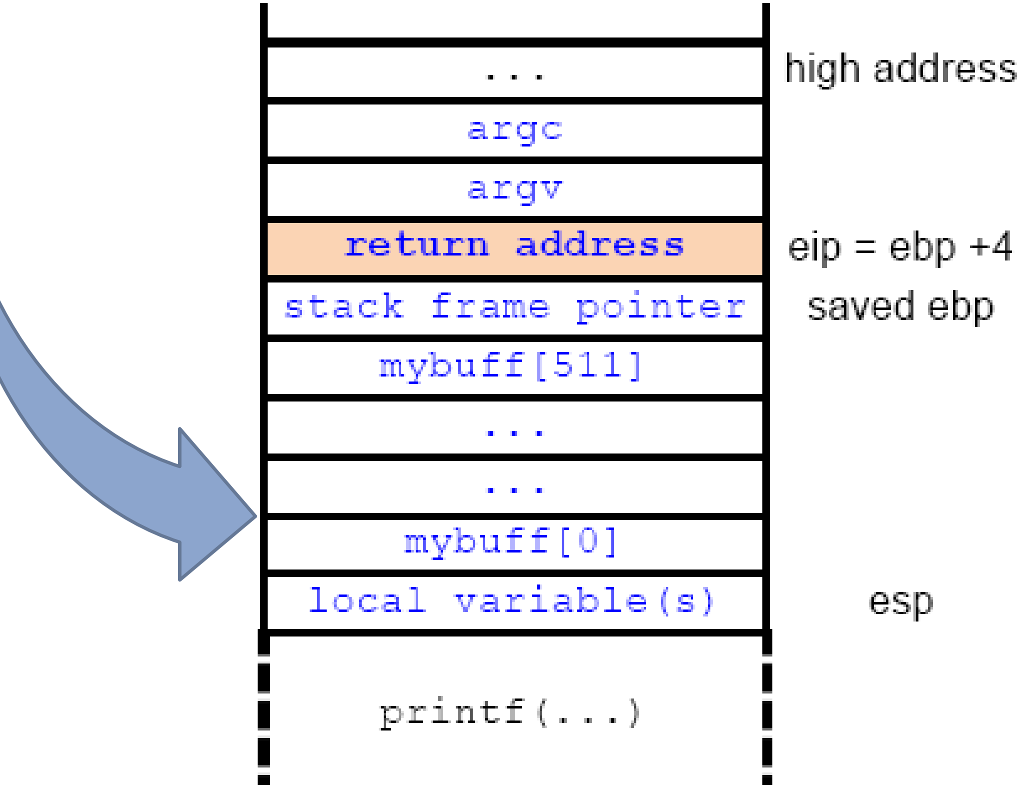
"Null Terminator" char indicates end of string

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040								96	60	140	`	`
1	1	001	SOH	(start of head	33	21	041								97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042								98	62	142	b	b
3	3	003	ETX	(end of text)		23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)			044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25		%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046			70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'		71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w

0	1	2	3	4	5
'H'	'e'	'l'	'l'	'o'	'\0'

1 char = 1 byte = 8 bits

→ $2^8 = 256$ ASCII chars



Raw Pointers are Dangerous!

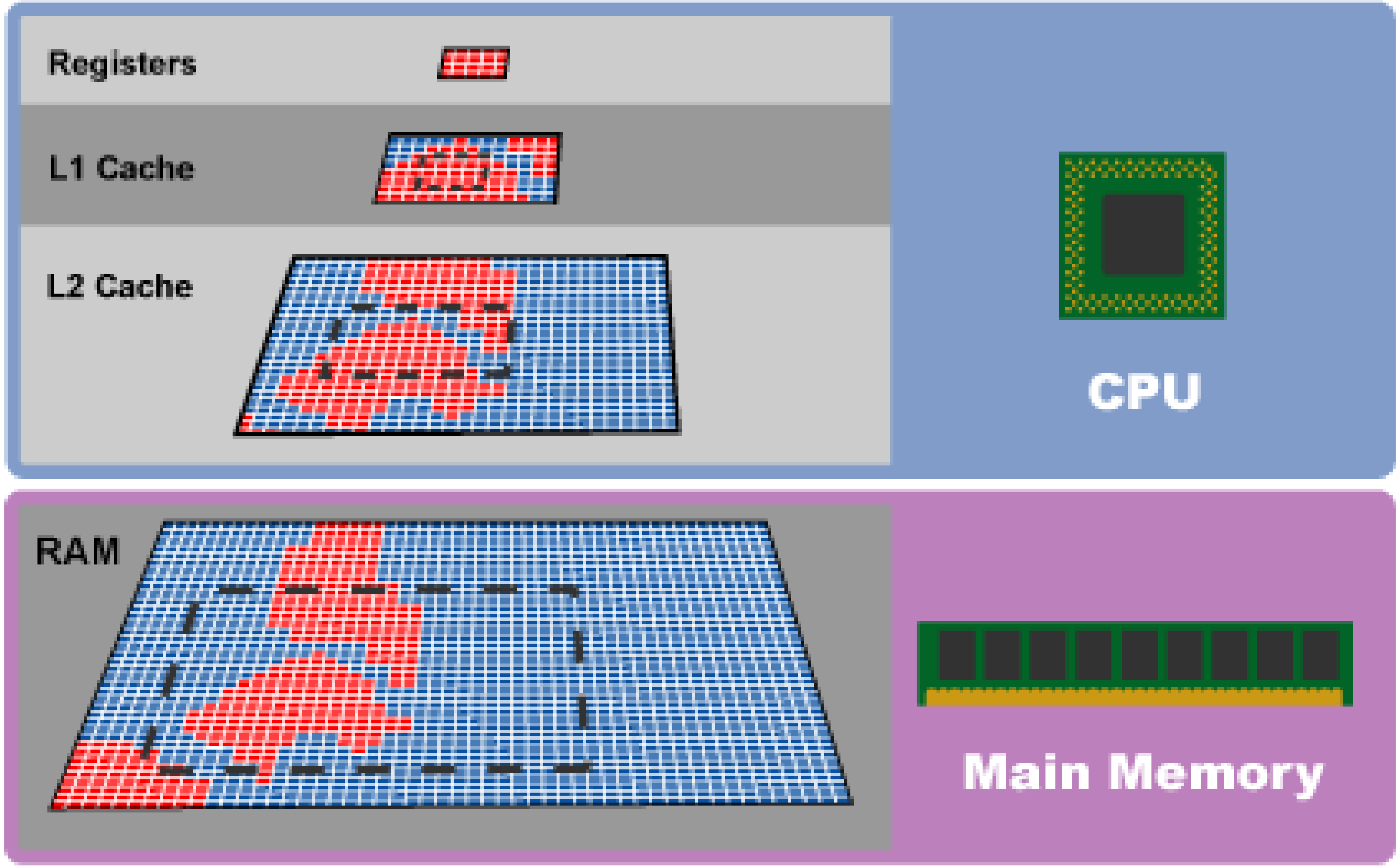
- Memory outside of String buffer is unprotected
  - Read/Write access
- Off-by-one errors overwrite important memory

RAM + Cache Memory Layout

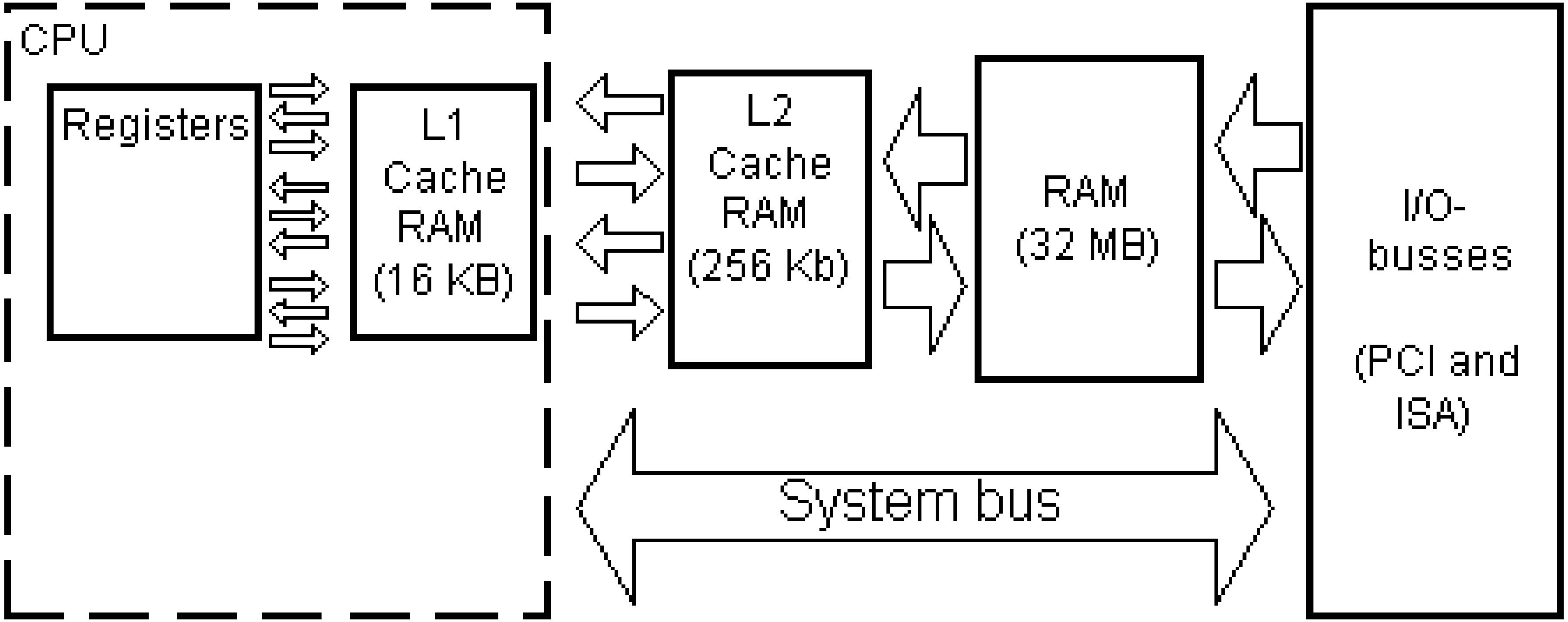
Arrays are *fast*.

Arrays sacrifice complexity for speed. Indexing is always O(1).

A significant contributor for array speed is *caching*.



0	1	2	3	4	5
'H'	'e'	'l'	'l'	'o'	'\0'



Speed vs. Storage

Physics (surface area) means smaller is faster

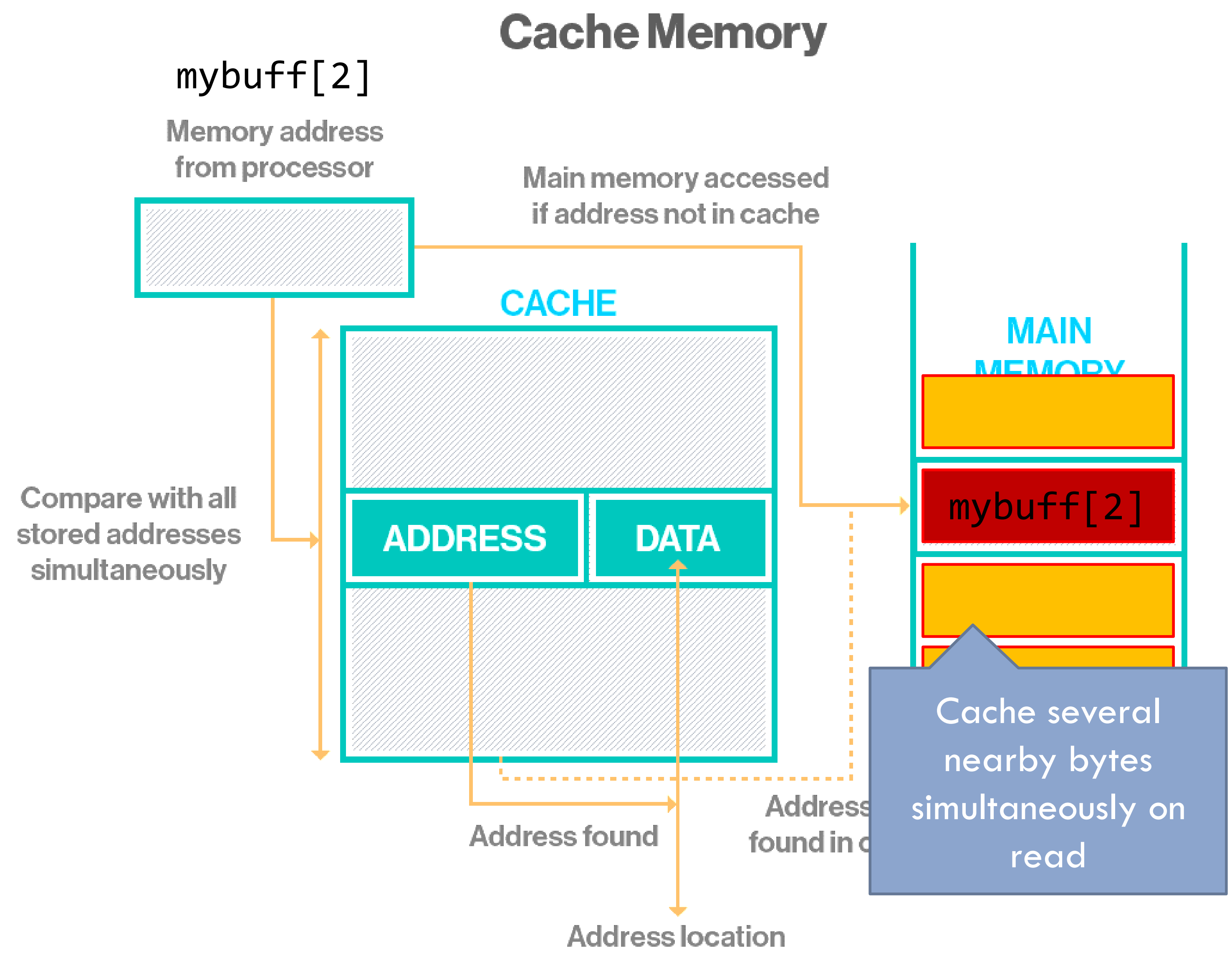
Layers of caches balance speed and size

Contiguous data (arrays) are easy to cache

Caching:
The Big Idea

As memory is read, copy large nearby contiguous blocks at once into cache.

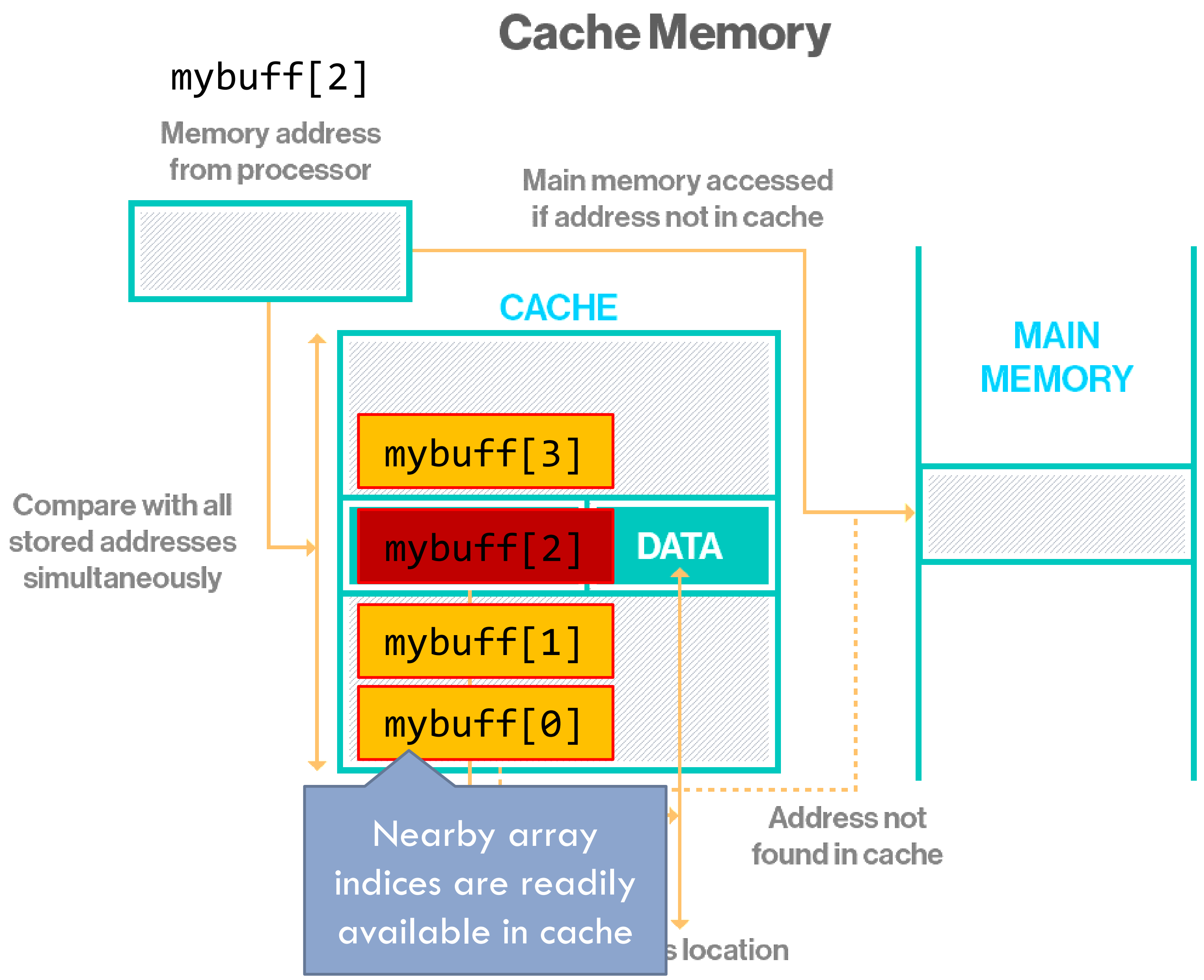
Arrays are by nature contiguous and therefore very cache friendly!



Caching:
The Big Idea

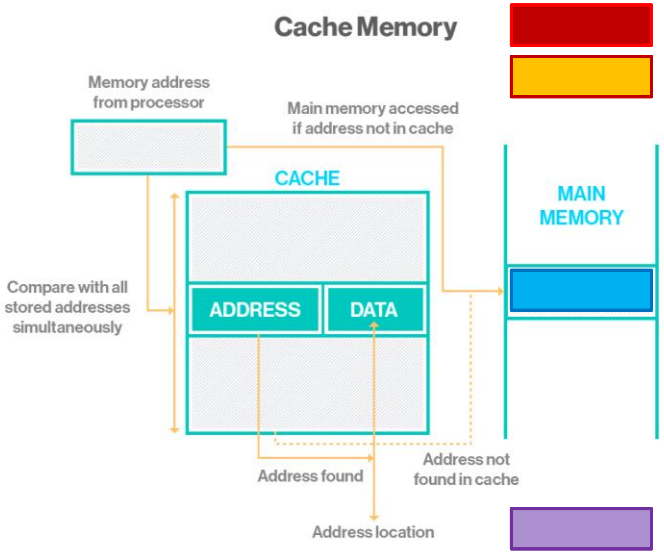
As memory is read, copy large nearby contiguous blocks at once into cache.

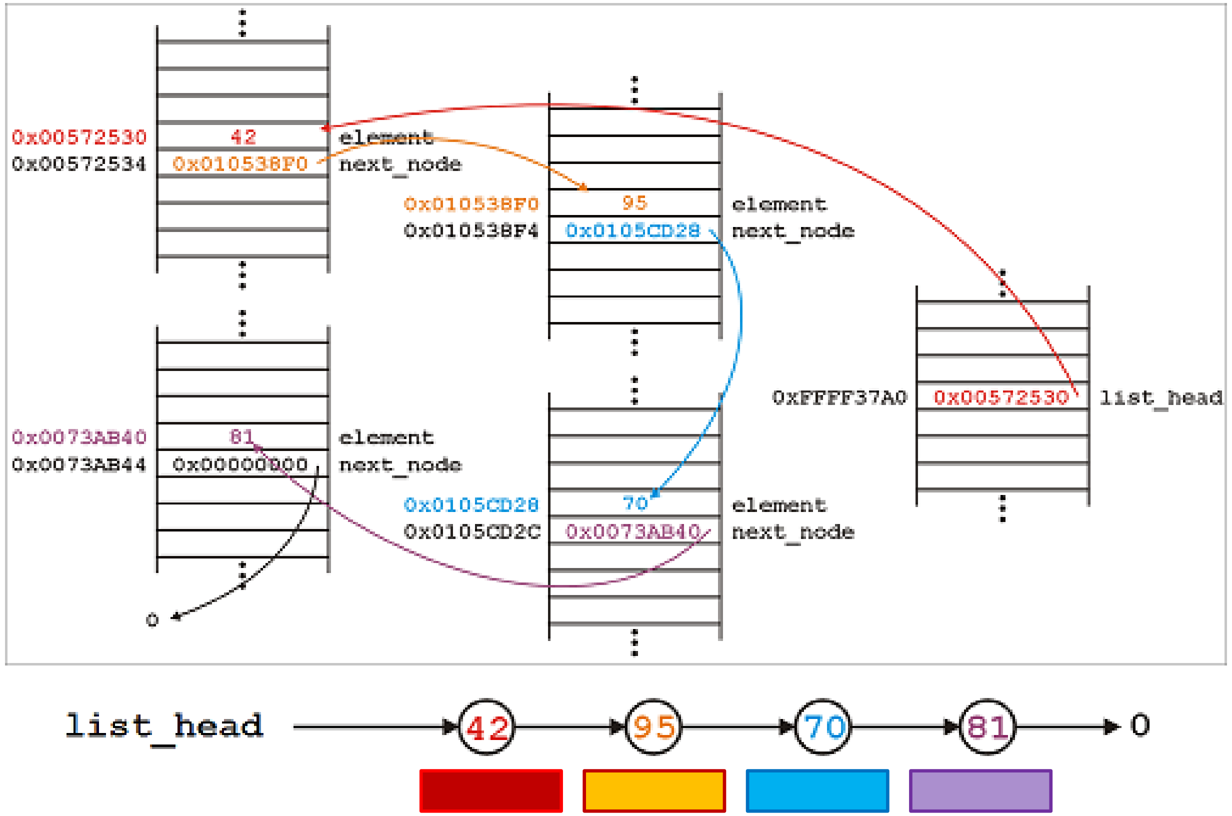
Arrays are by nature contiguous and therefore very cache friendly!



Caching + Linked List: Not So Fast

Linked List nodes can be stored physically far apart and are harder to cache

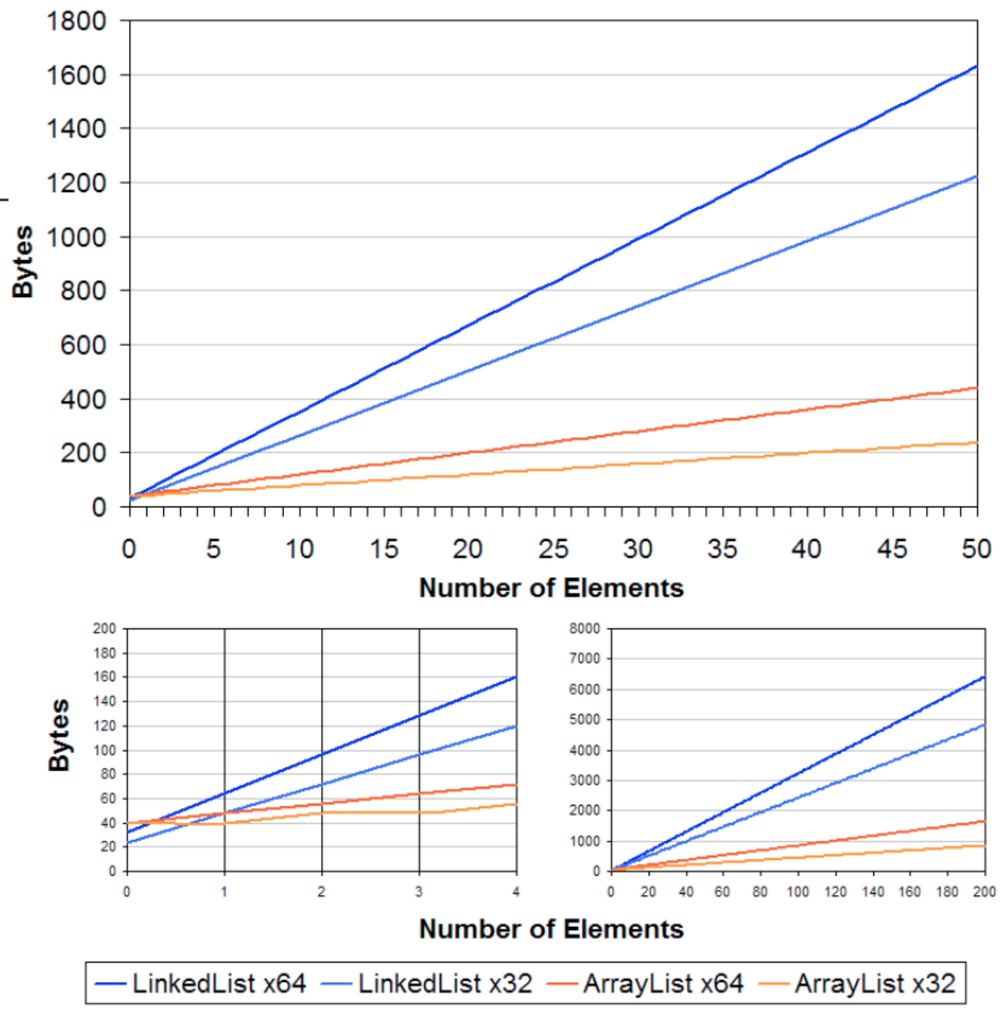




Linked List
Considered Harmful?

Linked Lists are not only cache-unfriendly, but they incur overhead:

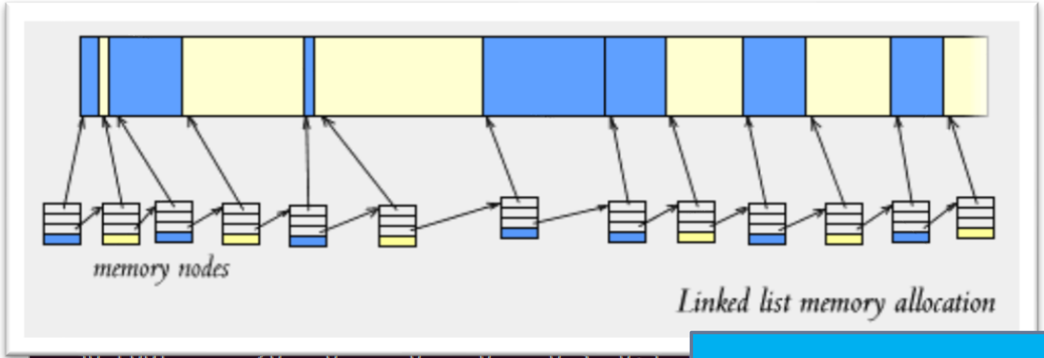
- **Object** overhead: header for **every** node contains references to class + methods
- **Allocation** overhead: every insertion requires scanning the Heap for free space

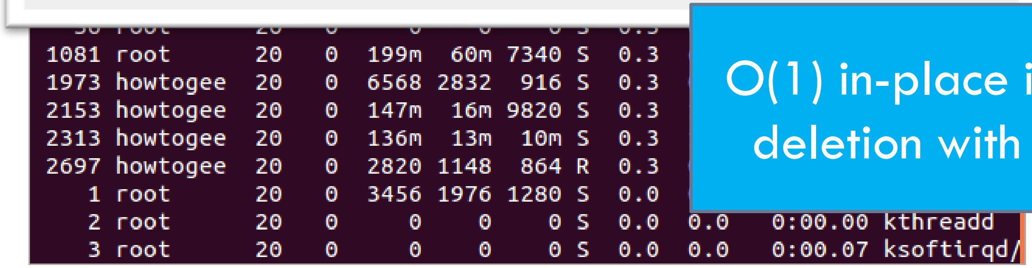


Source: [StackOverflow LinkedList vs. ArrayList Reading]

Linked Lists in the OS

Linked Lists are excellent for connecting a sequence of objects that are by nature separately stored.





Process list: every process has a unique PID

O(1) in-place insertion and deletion with no copying

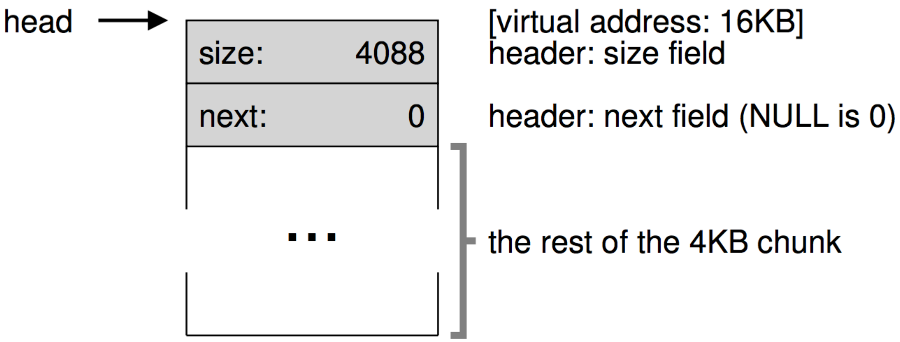


Figure 17.3: **A Heap With One Free Chunk**

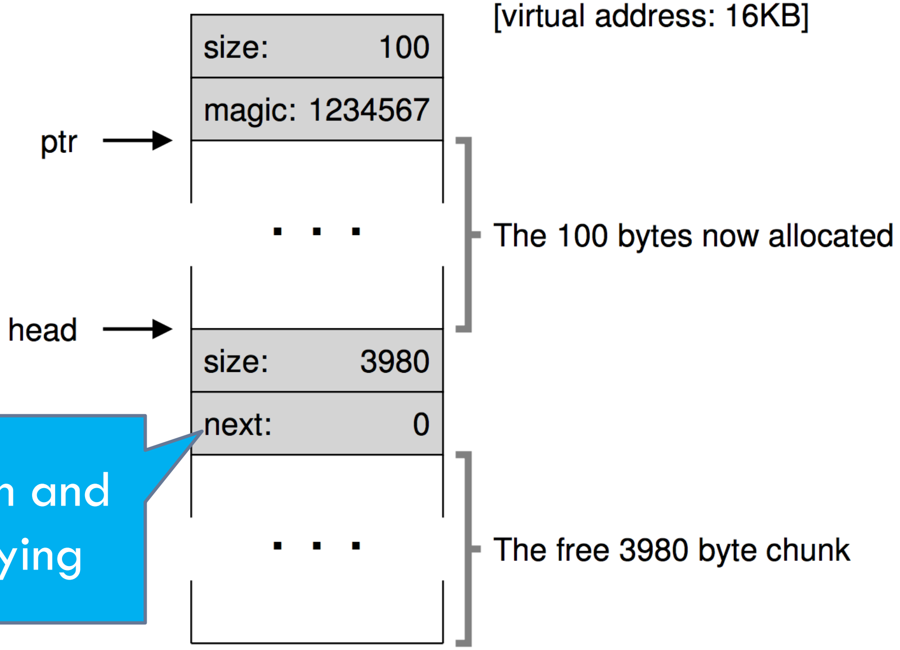


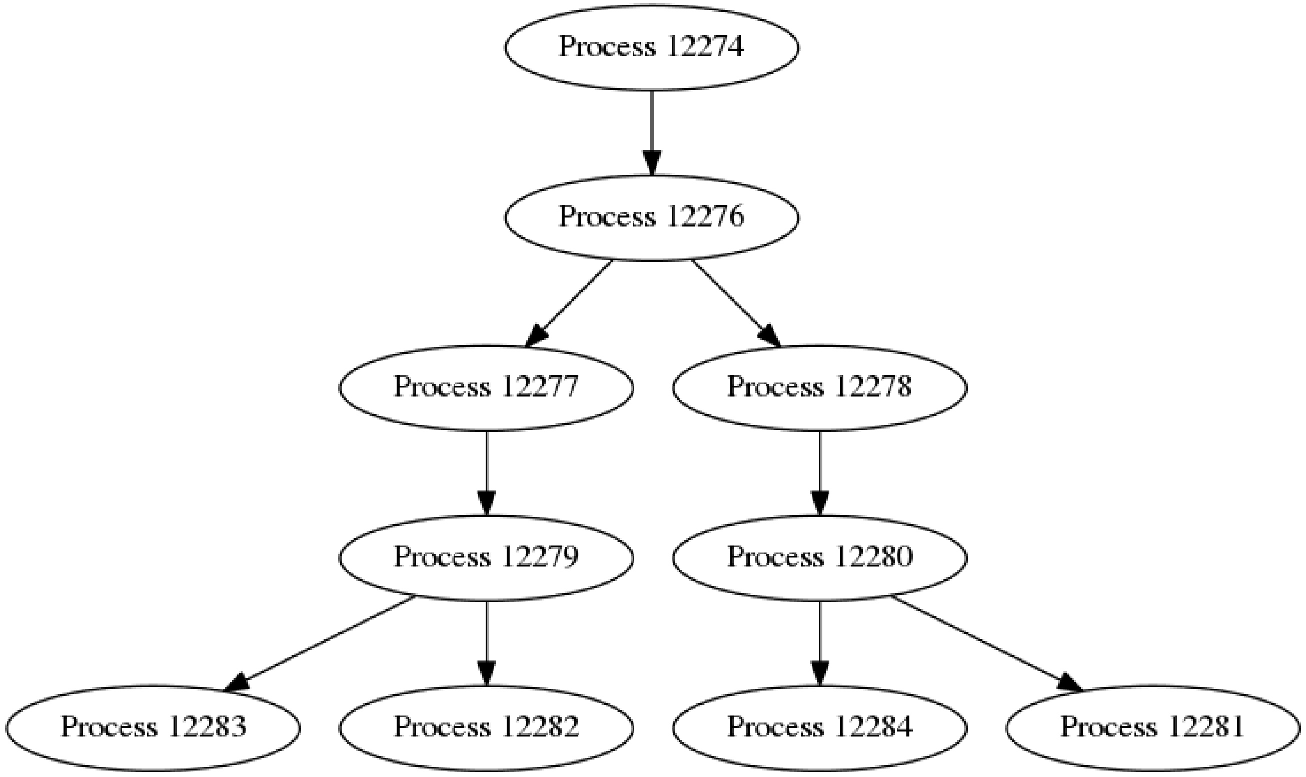
Figure 17.4: **A Heap: After One Allocation**

pstree in Unix

OS Multitasking: Processes as Trees

- Each OS process can spawn *child processes.*
  - *In turn,* they may spawn child processes of their own.
  - Processes have their own main thread and memory.

- Parents may kill their children (!)
- Parents may abandon their children (!)



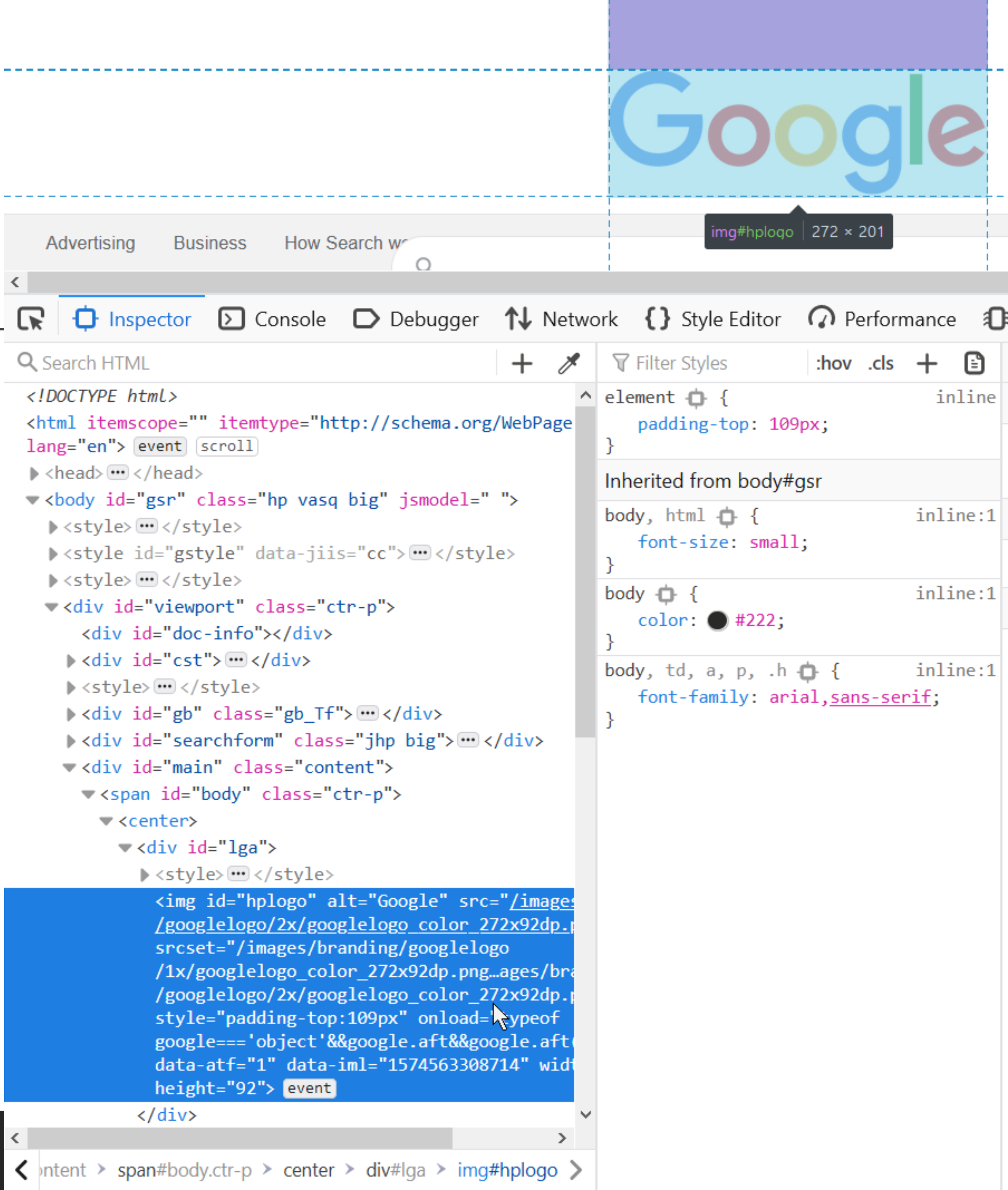
Trees as Websites

HTML (Hyper-Text Markup Language) describes the structure of webpages

- Leaf nodes (text, images) provide content
- Parent nodes provide context to children (**styling**, positioning, links)

Observe: nodes may have any number of children, to any depth

- Nodes can be parents of their own type
- **Path to leaf** gives context to leaf
- Allows for reusability and extensibility, e.g. a **bold link** described by parent chain <b> <a>



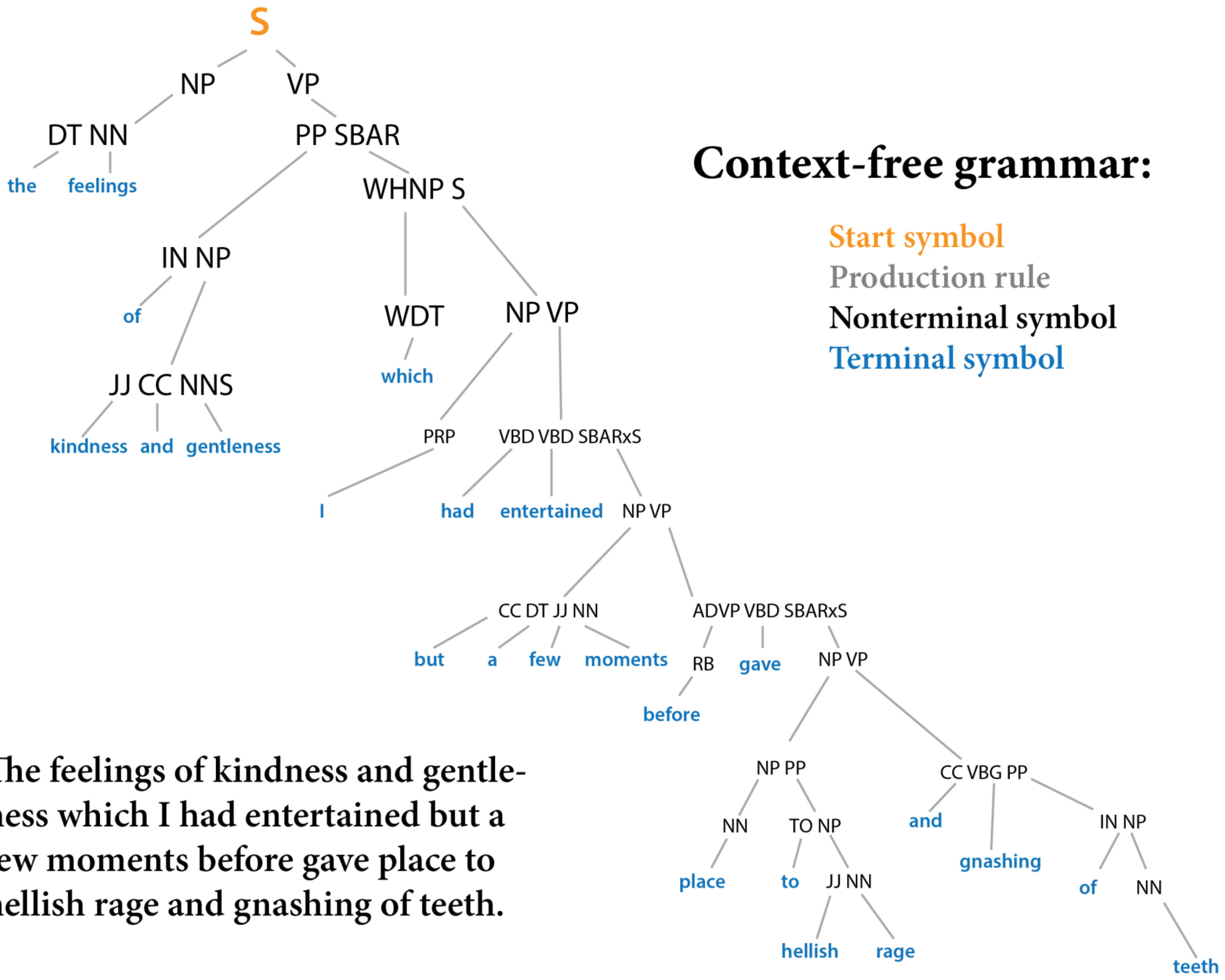
Trees as Languages

English is a Tree

- Nodes are entities (subject, object, verb)
- Leaves are words

English is recursive

- Verb phrase can have verb phrase, …
- Sentences can contain sentences?



Context-free grammar:

Start symbol
Production rule
Nonterminal symbol
Terminal symbol

The feelings of kindness and gentleness which I had entertained but a few moments before gave place to hellish rage and gnashing of teeth.

Trees as Languages

Programming Languages are Trees

- *Statements* composed of *Expressions*
- *Expressions* composed of *Operators* + and *Operands* a, b
- *Operands* may be *Statements* (recursive)

Grammar	Languages	Automaton	Production rules (constraints)*	Examples[3]
Type-0	Recursively enumerable	Turing machine	$\alpha A\beta \rightarrow \gamma$	$L = \{w \mid w$ describes a terminating Turing machine$\}$
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A\beta \rightarrow \alpha\gamma\beta$	$L = \{a^n b^n c^n \mid n > 0\}$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \alpha$	$L = \{a^n b^n \mid n > 0\}$
Type-3	Regular	Finite state automaton	$A \rightarrow \mathrm{a}$ and $A \rightarrow \mathrm{a}B$	$L = \{a^n \mid n \geq 0\}$

* Meaning of symbols:

- a = terminal
- A, B = non-terminal
- α, β, γ = string of terminals and/or non-terminals
  - α, β = maybe empty
  - γ = never empty

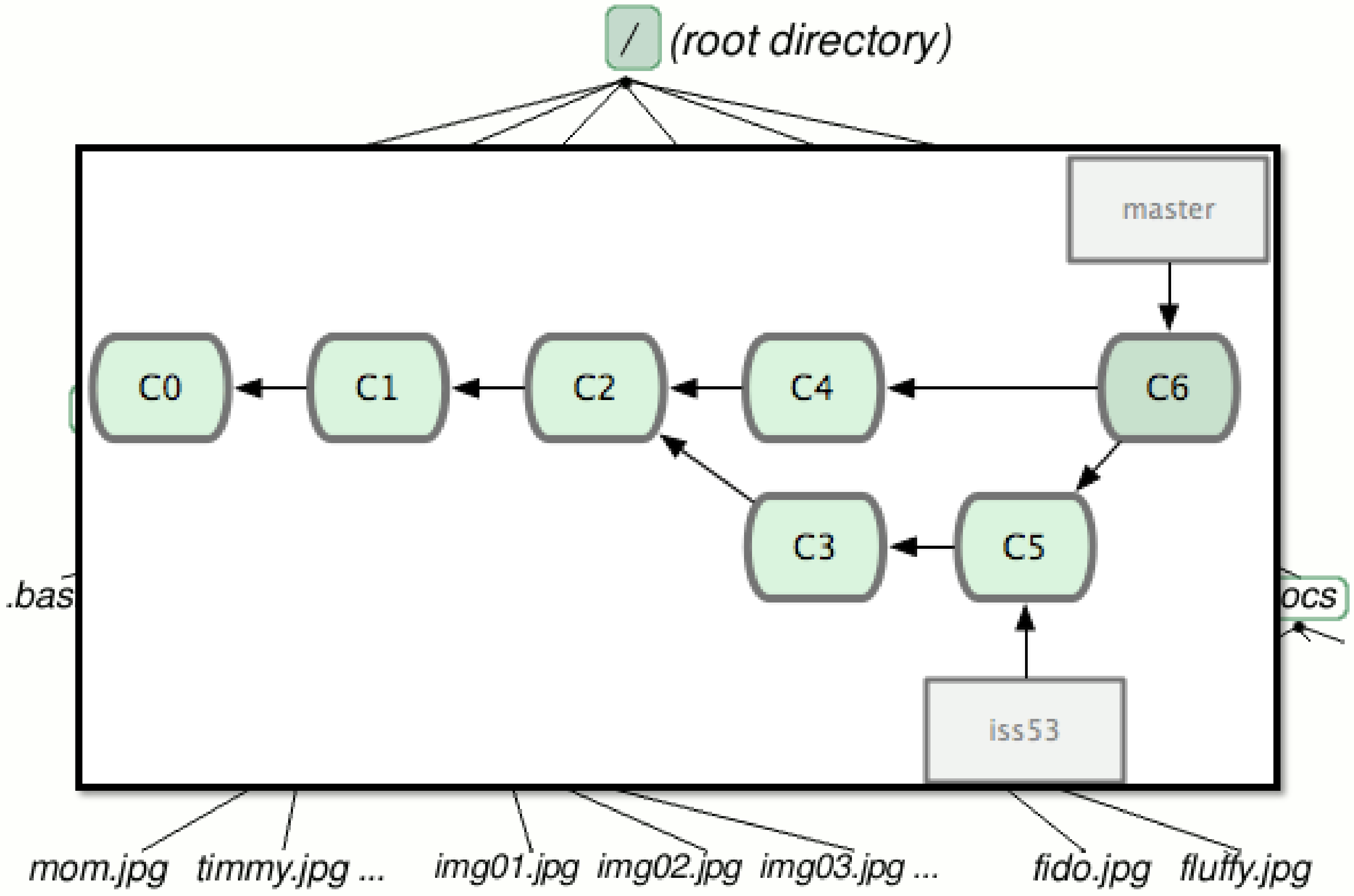
Trees as Filesystems

File Systems track directories and files

- *Directories are parent nodes*
  - May also be leaf nodes
- *Files are leaf nodes*

Git is also tree-like

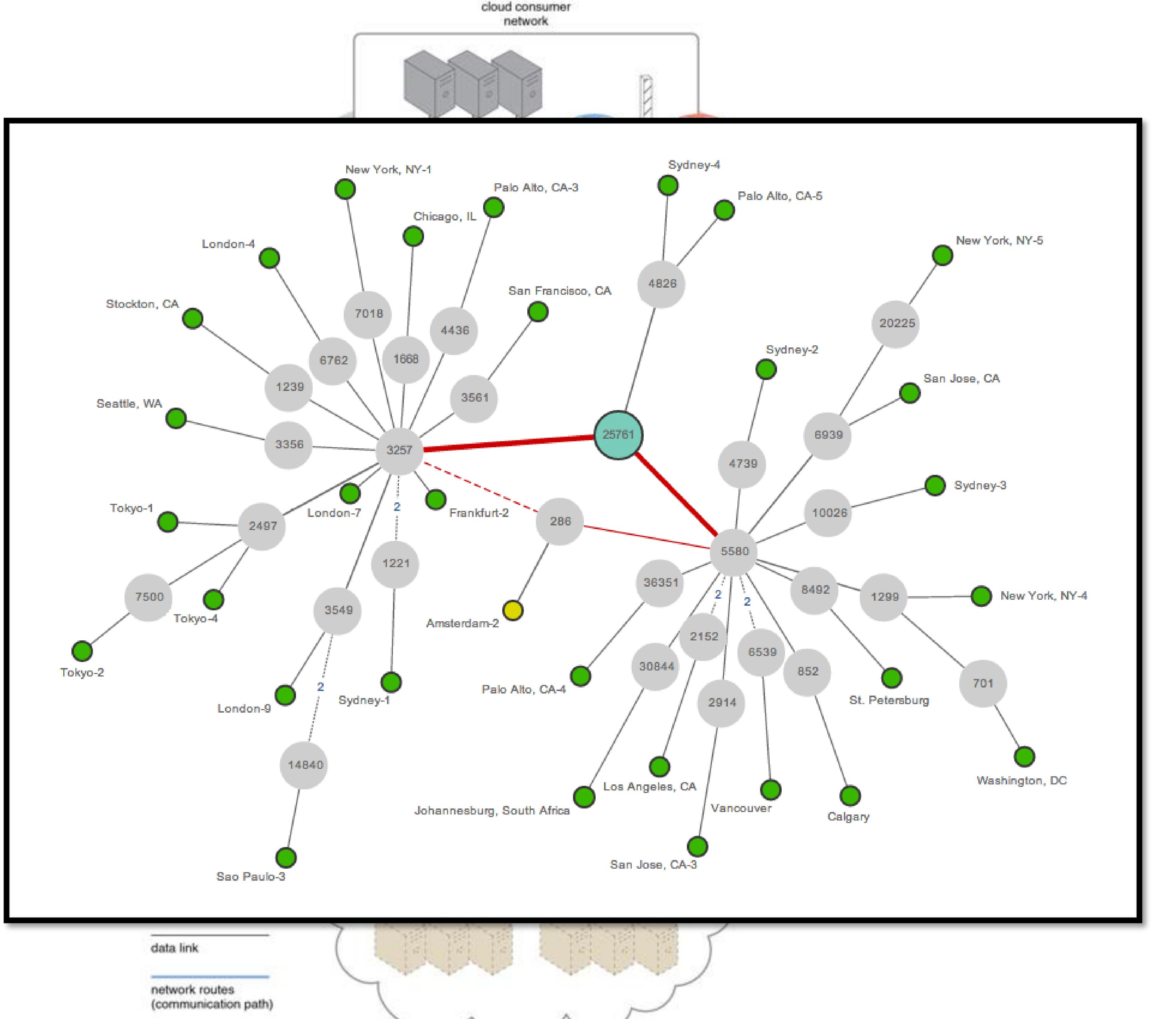
- Commits are nodes
- Actually, a graph



Graphs in Networking

The Internet is composed of several devices and routers

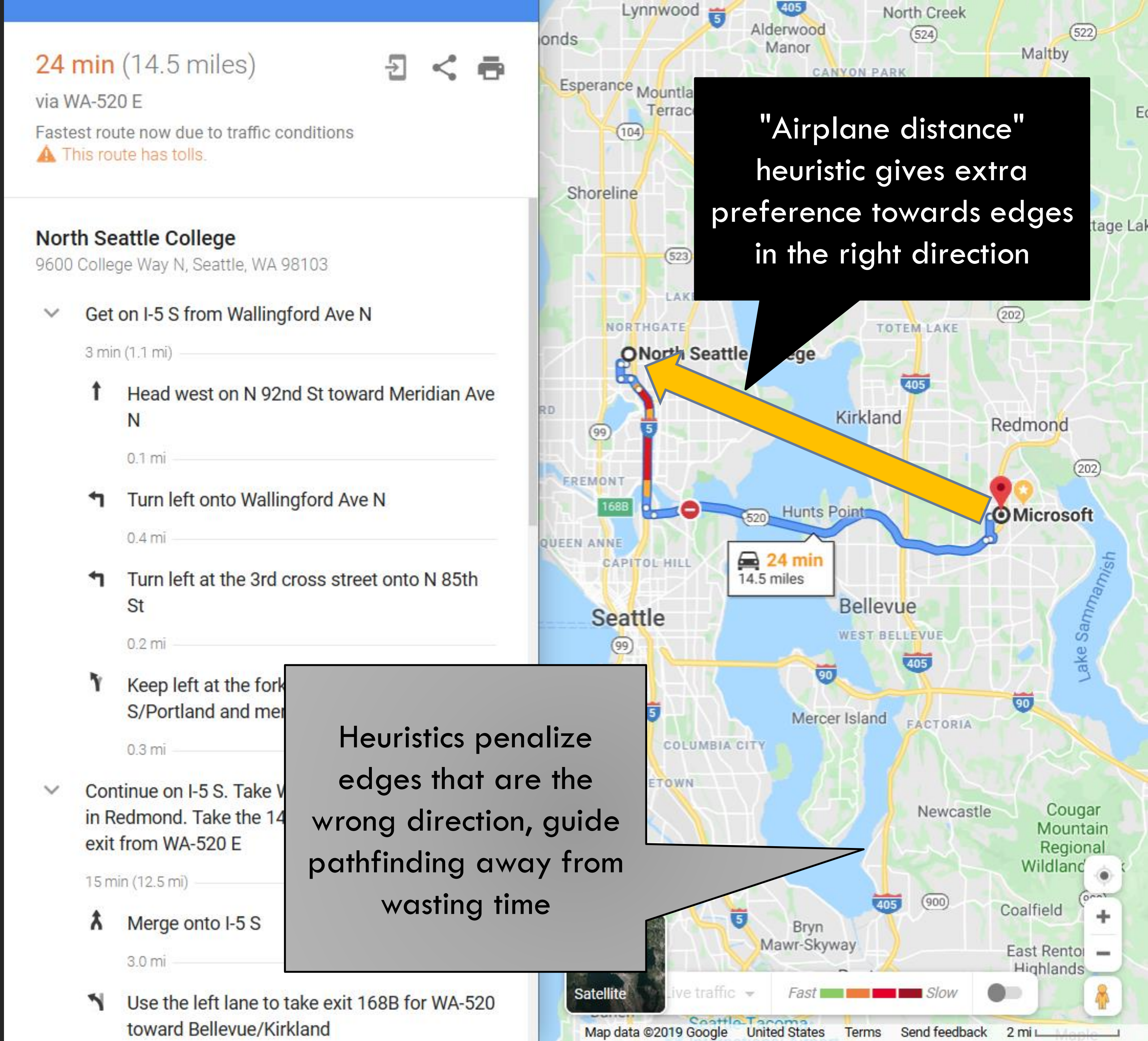
- Endpoint devices (phones, laptops, etc) are like tree leaves
- Middle "parent" nodes are routers that determine "one-hop" pathing



Graphs, IRL

Maps model the real world as graphs
- Destinations are nodes
- Streets are edges
- "Cost of Travel" (time) is the cost of each edge

Dijkstra's Algorithm, plus heuristics (A*)



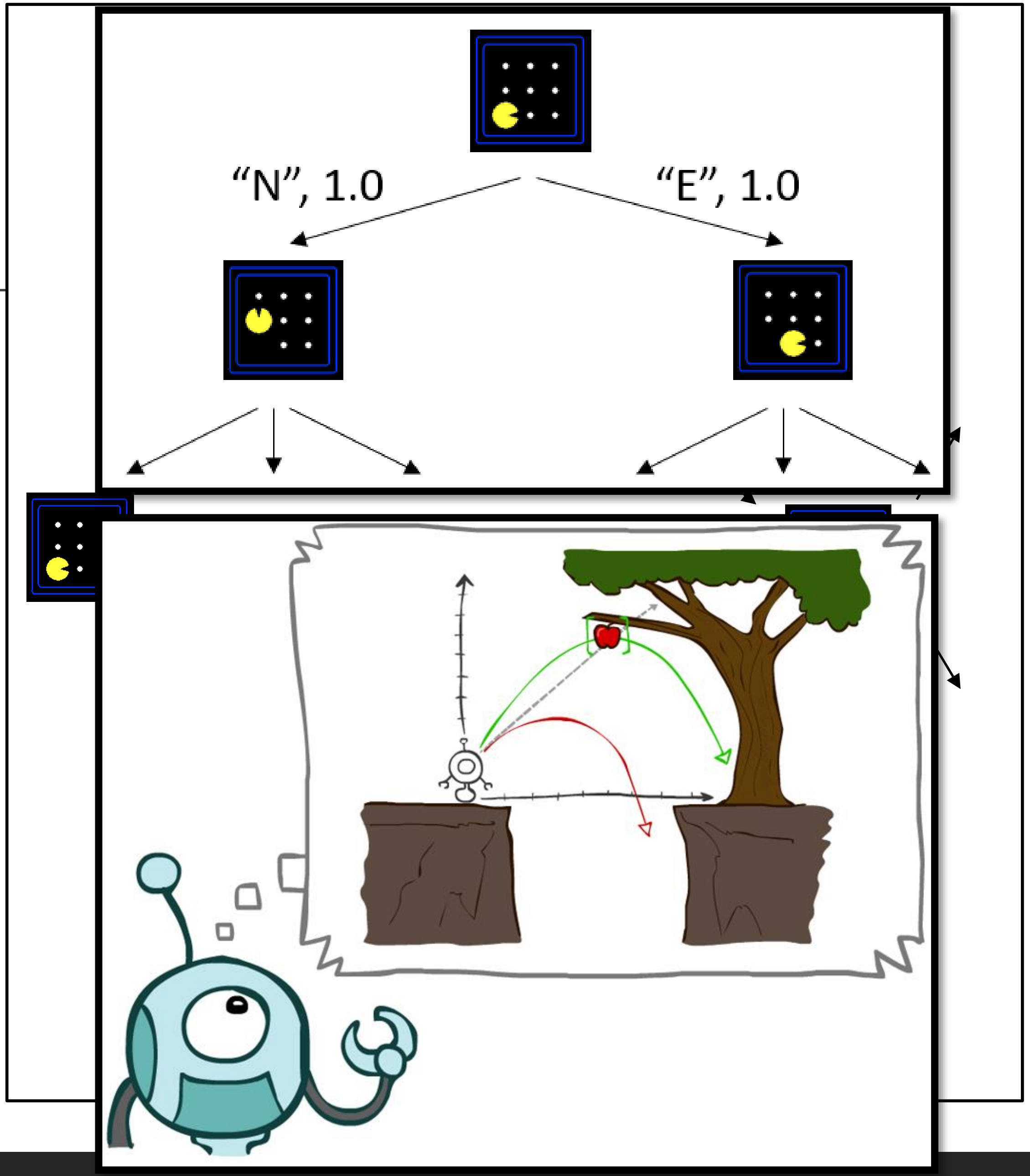
AI: State Graphs

Graphs can be used to model *state* and *transitions* for AI

- *State* is like variables in an Object
  - *The observed world are graph nodes*
- *Transitions* are like method invocations that change state
  - *The actions of a robot are edges*

Graph search algorithms allow robots to find **paths** to **goals**

- Remember `path(…)`?



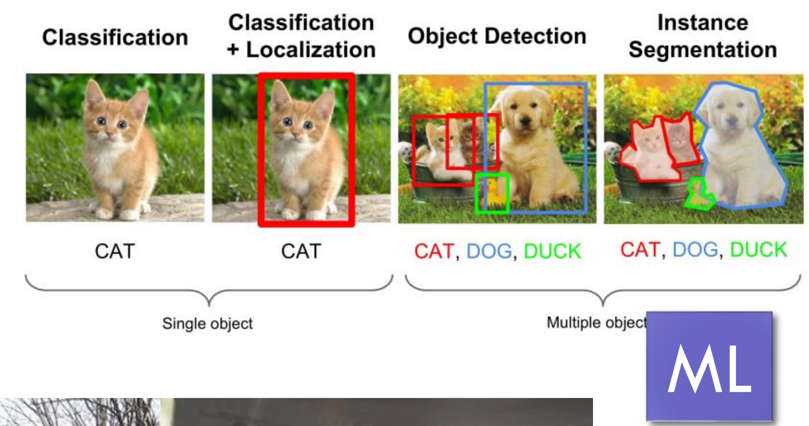
Aside: AI vs. Machine Learning (ML)

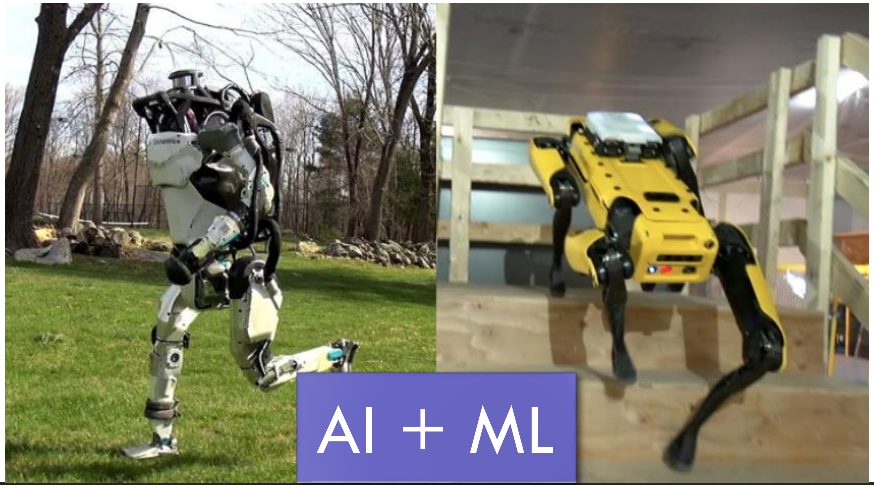
Contrary to what the media believes,

AI and **Machine Learning** are two different things

- AI: teaching computers to act rationally in a modeled world to achieve goals
- ML: teaching computers to make observations about the world

Machine learning and AI can coexist, but are not synonymous!

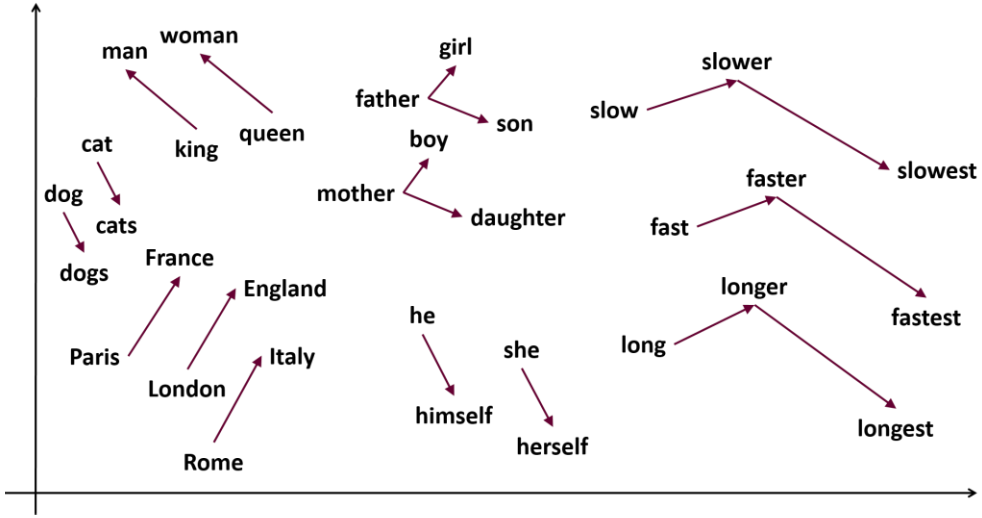




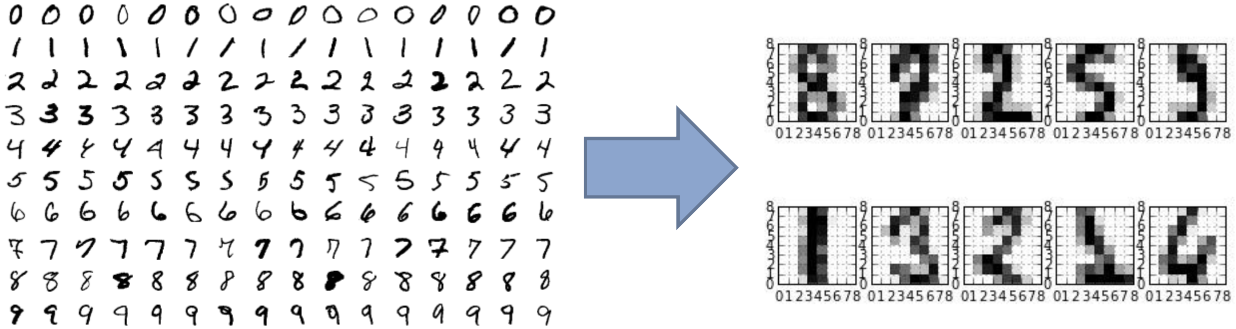
ML: Vectors & Matrices

Fundamental to ML are vectors and matrices (calculus, linear algebra)

Big Idea: model complex information as a **vector** or **matrix** of integers



- Point, Point3D, **arrays, 2D arrays**
- Position of vector and proximity to others indicate how things are related
- *Almost like hashing, minus the random*

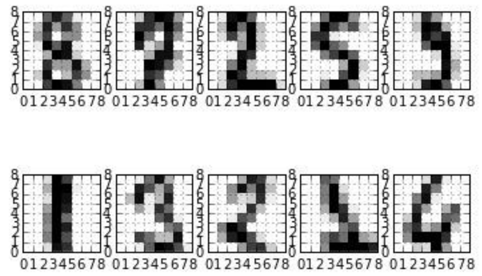


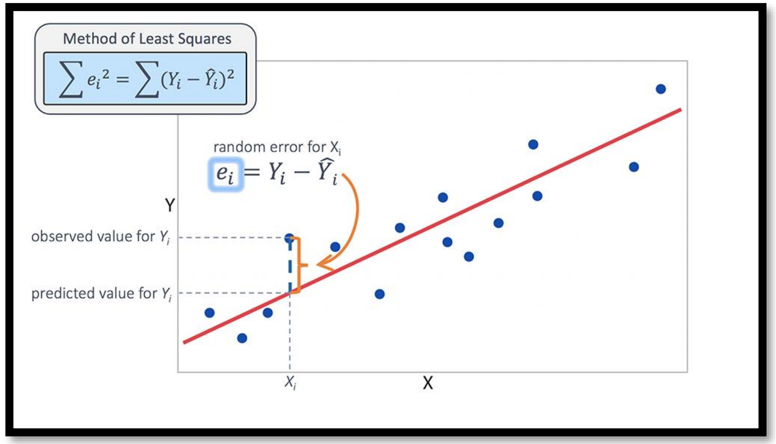
ML: Vectors & Matrices

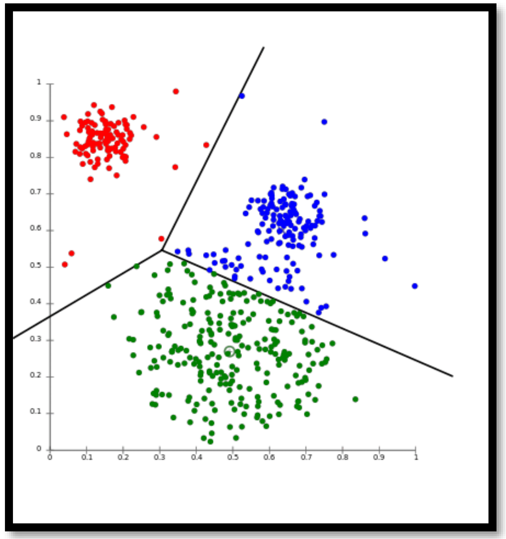
Big Idea: model complex information as a **vector** or **matrix** of integers

- Point, Point3D, **arrays, 2D arrays**
- Position of vector and proximity to others indicate how things are related

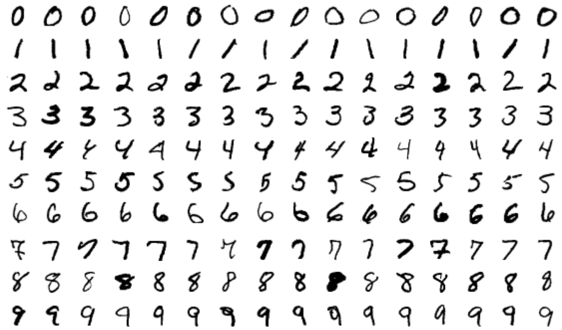
Then: perform a regression (statistical analysis) on data

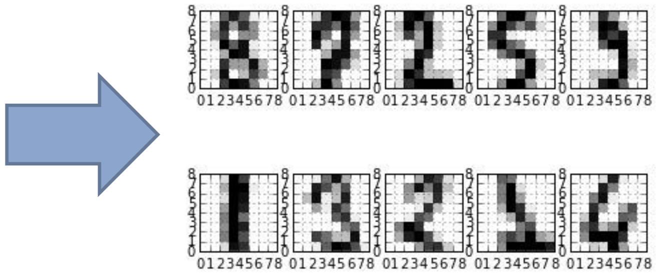






ML: Neural Nets+

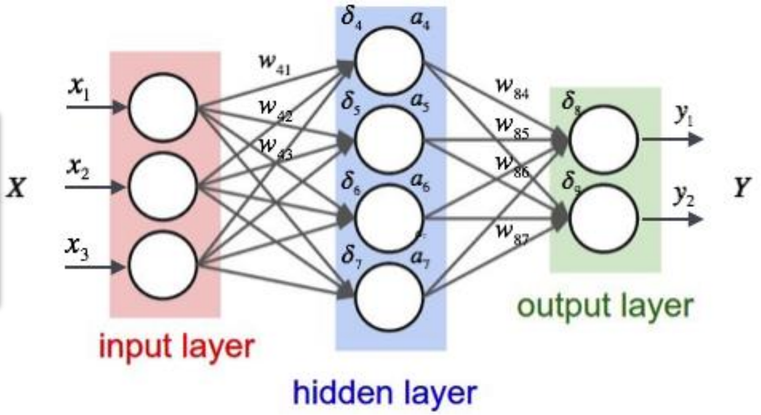




Big Idea: model complex information as a **vector** or **matrix** of integers

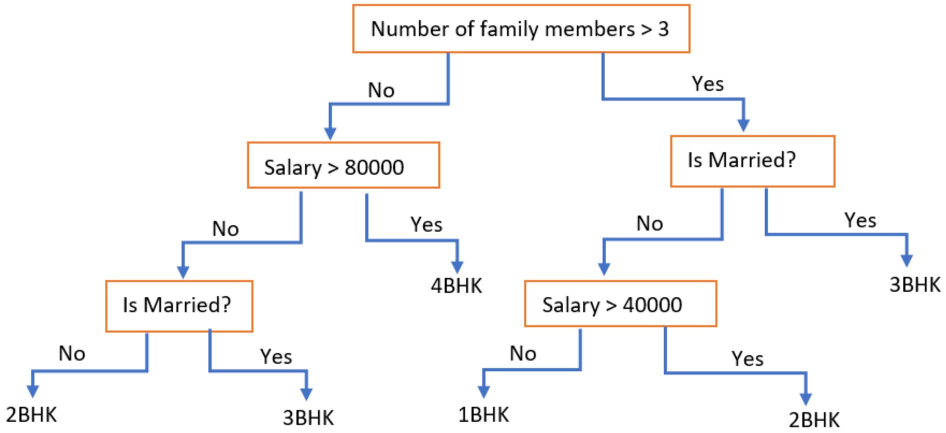
- Point, Point3D, **arrays, 2D arrays**
- Position of vector and proximity to others indicate how things are related

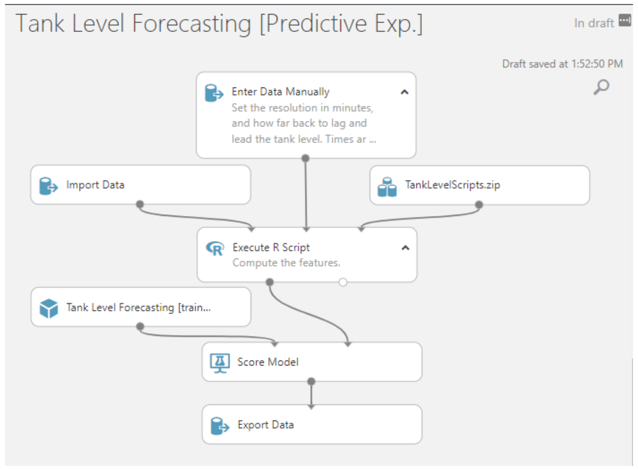
Neural nets **(graphs)** transform the data repeatedly until it becomes useful (training, gradient descent)



Then: perform a regression (statistical analysis) on data

Decision **trees** find meaningful ways to divide data based on conditions





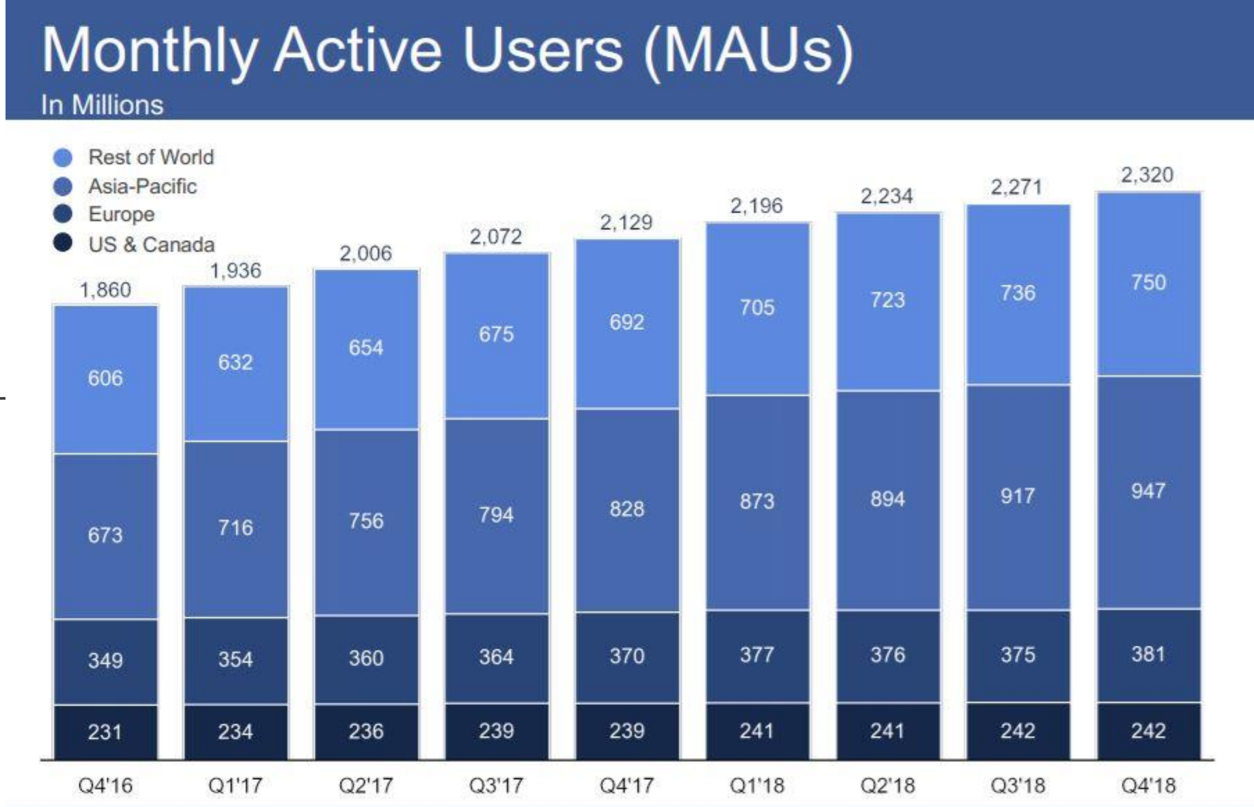
Speaking of Big Data…

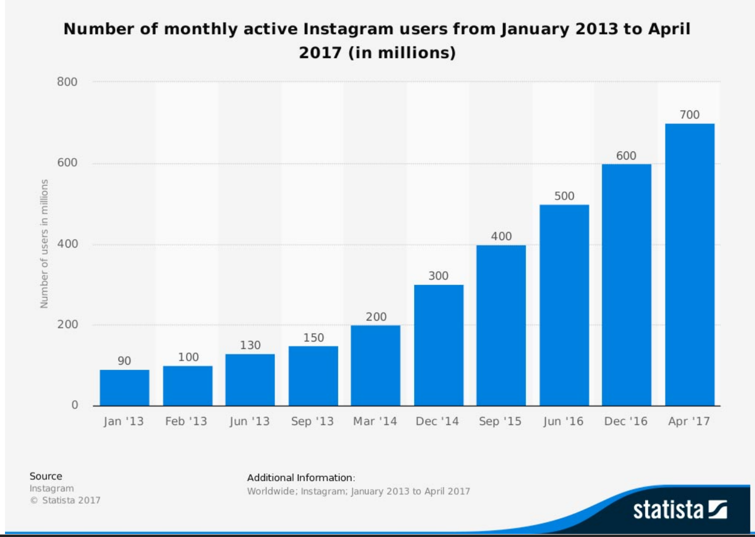
Companies like Facebook track millions of users each day

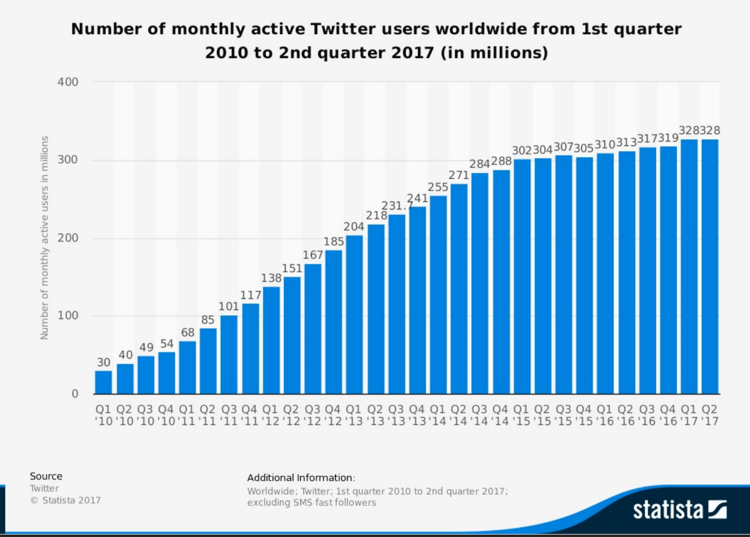
- Databases must associate names and information
- Users expect instant response

We've seen two structures with efficient lookup

- HashMap O(1)?
- Binary Search Tree O(logn)?







B+ Trees for Databases

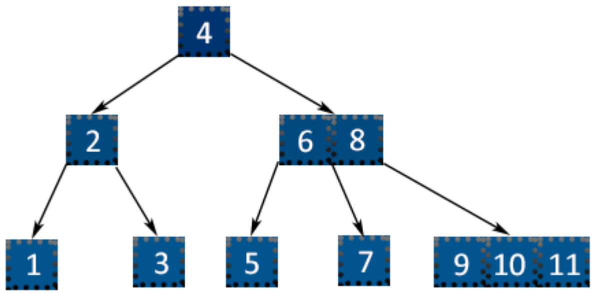
Databases like MySQL use "B+ trees" for fast lookup

- Like binary search trees:
  - left is lower, right is bigger, O(logn) lookup
- Unlike binary search trees:
  - A node can have any number of children
  - B-trees are perfectly balanced (as all things sho

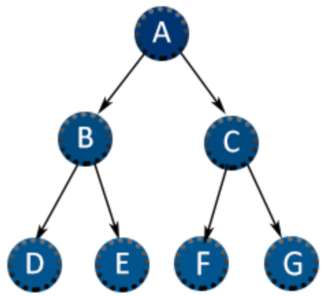
Why not HashMaps?

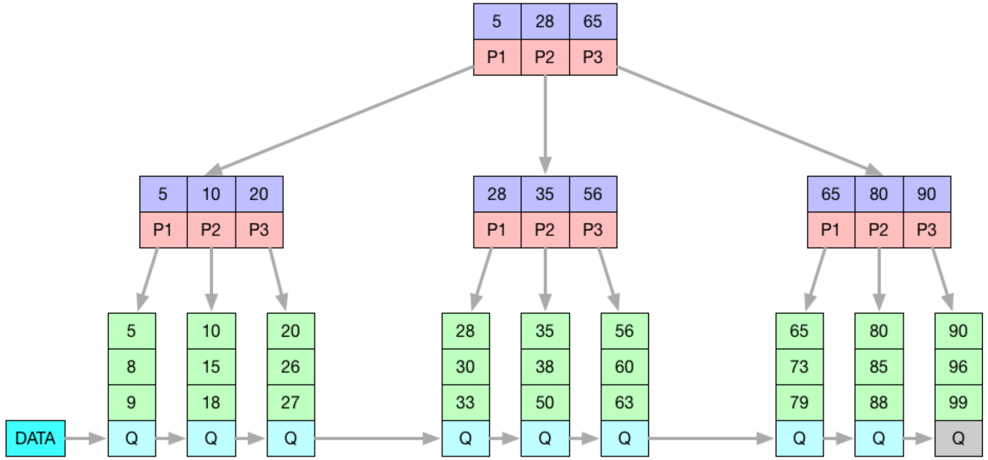
- No sequential ordering (why?)
- Wasted space (why?)

B-TREE



BINARY TREE





HashMap Strikes Back

HashMap is useful for **dynamic content** at "small scale"

- News articles mapped to links
- Zip codes mapped to cities
- Videos mapped to thumbnails
- Users mapped to messages

Both JavaScript and Python have HashMaps as **first-class citizens** of the language

- *Everything* is a HashMap underneath

Valid code in both **Python and JavaScript:**

```
map = {}
map["seattle"] = "WA"
map["portland"] = "OR"
```

Valid code in **JavaScript:**

```
console.log("Hello, World!");
console["log"]("Hello, World!");
```

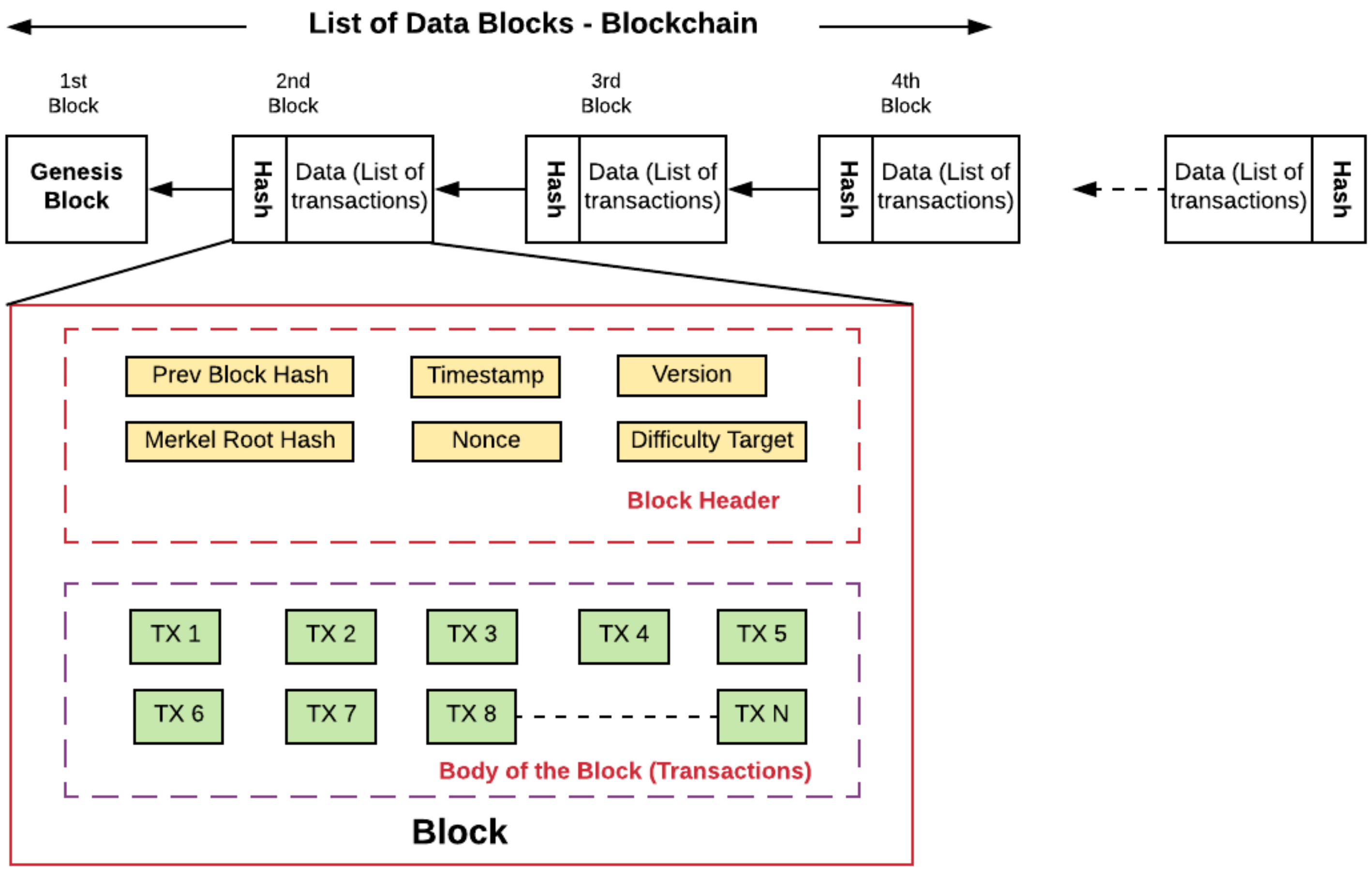
Valid code in **Python:**

```
def foo(): print("hello")
setattr(foo, "thing", "world")
print("hello " + getattr(foo, "thing"))
```

BONUS: Linked List as Blockchain?

Blockchain was all the rage of yesteryear

- Linked List + hashes of neighbor blocks
- Hashes "guarantee" immutability of chain



BONUS: Bitcoin – at what cost?

UNIVERSITY OF CAMBRIDGE Judge Business School | Cambridge **Centre for Alternative Finance** | Cambridge Bitcoin Electricity Consumption Index

- CBECI
- **COMPARISONS**
- METHODOLOGY
- FAQ
- CONTACT

Venezuela	Chile	Bitcoin	Philippines	Belgium
71.96 TWh per year	**73.22** TWh per year	**74.59** TWh per year	**78.30** TWh per year	**82.16** TWh per year

The Best Data Structure?

- Array (+ArrayList)
- Linked List (+Doubly, Stack, Queue)
- Tree (+Binary, BST)
- Heap (+Priority Queue)
- HashMap
- Graph

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)