Message Passing and Microkernels

CSE 536 Spring 2024

jedimaestro@asu.edu

Message passing *vs.* RPC

- Message passing
  - Procedure not directly invoked by name
  - Can be asynchronous, and typically is
  - Multicast and broadcast are pretty natural to the abstraction
  - Models of message passing (like π calculus) incorporate physics, are decentralized
- RPC
  - Have to specify name of procedure you're calling
  - Can be non-blocking, *i.e.*, asynchronous, but typically is not
  - Not really a way to multicast or broadcast
  - Need a central database of [type, instance] pairs

Two cautionary notes

- Some people consider RPC to be a specific case of message passing
  - Message passing libraries often offer RPC API built on top of message passing
- Message passing means a lot of different things to a lot of different people
  - Object oriented people … It’s about modularity
  - Distributed computing people … It’s a beautiful model of concurrency
  - Microkernel people … It’s about robustness

https://en.wikipedia.org/wiki/Message_passing

- “In computer science, message passing is a technique for invoking behavior (*i.e.*, running a program) on a computer.”
- “The invoking program sends a message to a process (which may be an actor or object) and relies on that process and its supporting infrastructure to then select and run some appropriate code.”
- “Message passing differs from conventional programming where a process, subroutine, or function is directly invoked by name.”

Synchronous message passing

- When two objects are running at the same time, *e.g.*, in Java or Smalltalk
- “Synchronous messaging is analogous to a synchronous function call; just as the function caller waits until the function completes, the sending process waits until the receiving process completes.”
- *E.g.*, Circle, Square, and Rectangle are subclasses of Shape, send any Shape a message to calculate its own area

Isn’t that just object oriented programming with polymorphism?

Asynchronous message passing

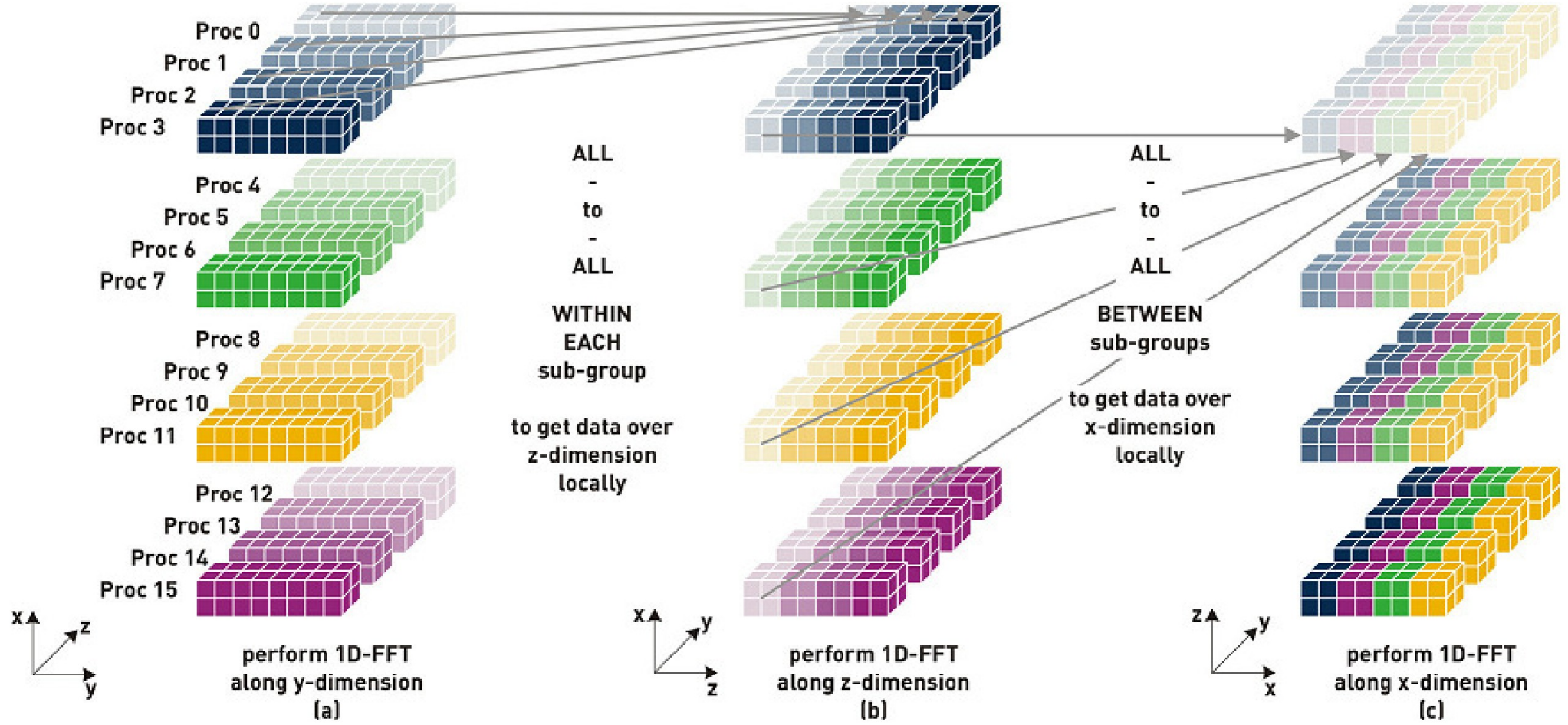
- “With asynchronous message passing the receiving object can be down or busy when the requesting object sends the message.”
- “Continuing the function call analogy, it is like a function call that returns immediately, without waiting for the called function to complete.”
- Requires storing and retransmitting data
- Buffer gets full?
  - Block (can deadlock), *--or--*
  - Drop messages

Another advantage of message passing is multicast and broadcast…

```
//scatter rows of first matrix to different processes
MPI_Scatter(a, N*N/size, MPI_INT, aa, N*N/size, MPI_INT,0,MPI_COMM_WORLD);

//broadcast second matrix to all processes
MPI_Bcast(b, N*N, MPI_INT, 0, MPI_COMM_WORLD);

MPI_Barrier(MPI_COMM_WORLD);

//perform vector multiplication by all processes
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
    sum = sum + aa[j] * b[j][i];
  }
  cc[i] = sum;
  sum = 0;
}

MPI_Gather(cc, N*N/size, MPI_INT, c, N*N/size, MPI_INT, 0, MPI_COMM_WORLD);

MPI_Barrier(MPI_COMM_WORLD);
MPI_Finalize();
```

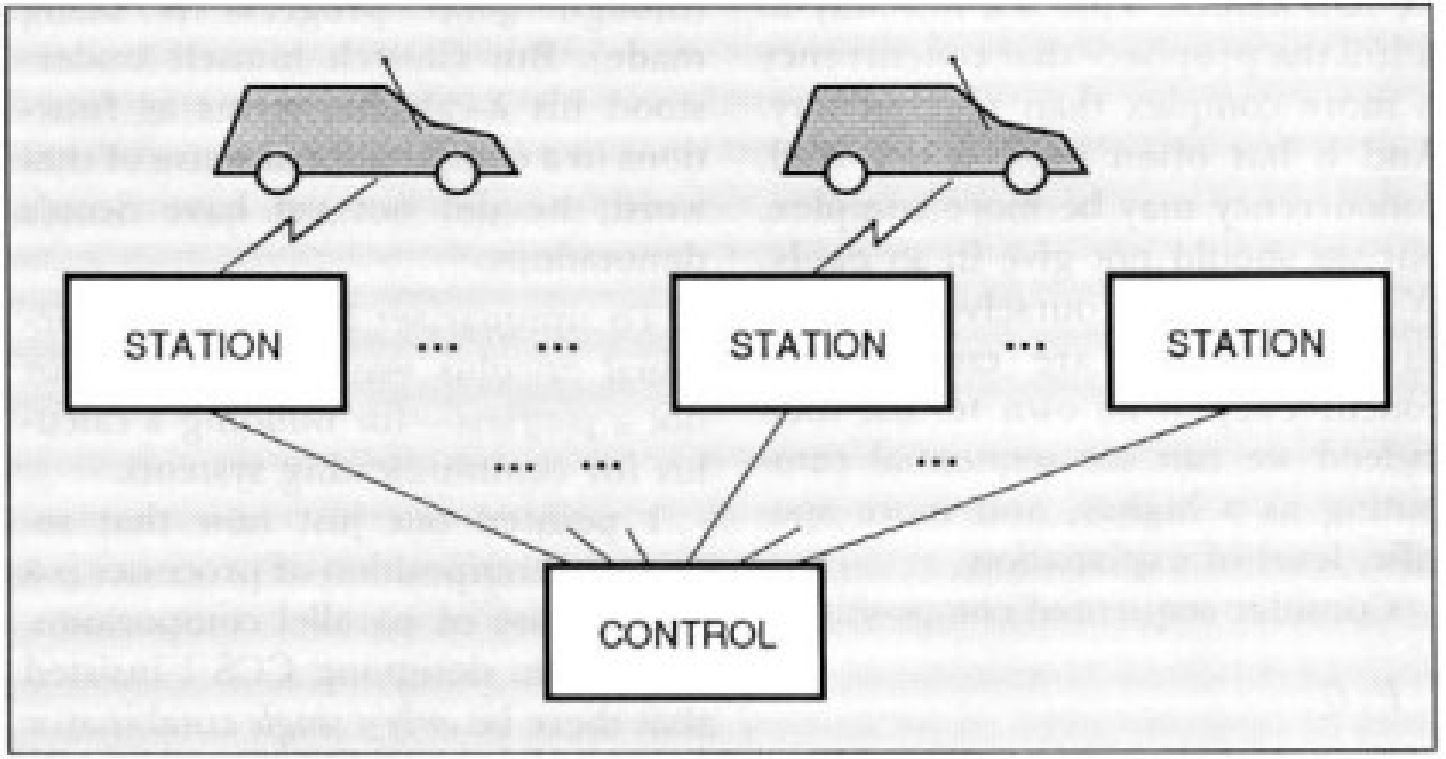
https://stackoverflow.com/questions/41575243/matrix-multiplication-using-mpi-scatter-and-mpi-gather

http://www.hector.ac.uk/cse/distributedcse/reports/cp2k02/cp2k02/node8.html



Another advantage of message passing is mobility...

Can we use semaphores, mutexes, *etc.* for this?



https://dl.acm.org/doi/pdf/10.1145/151233.151240

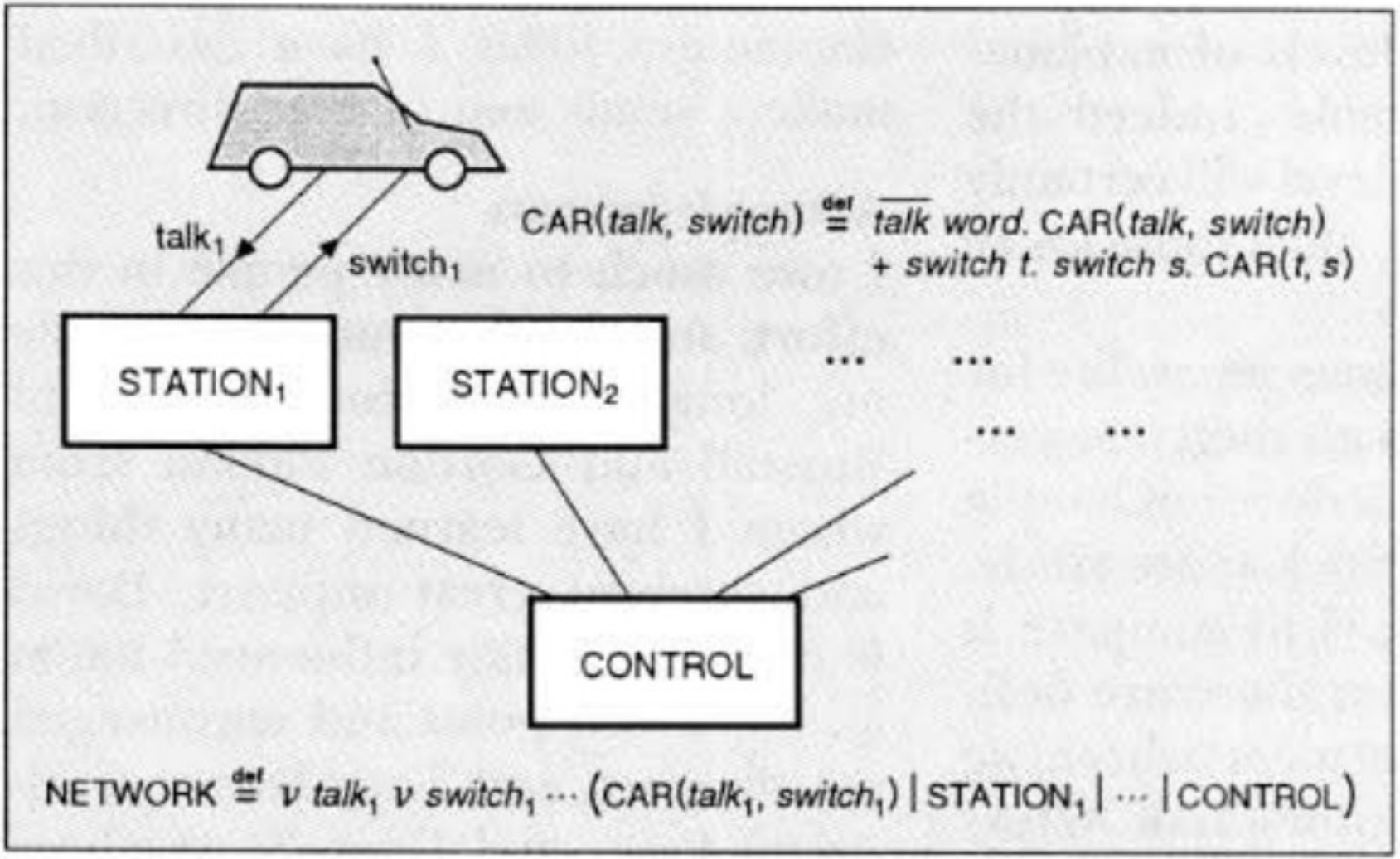
π calculus

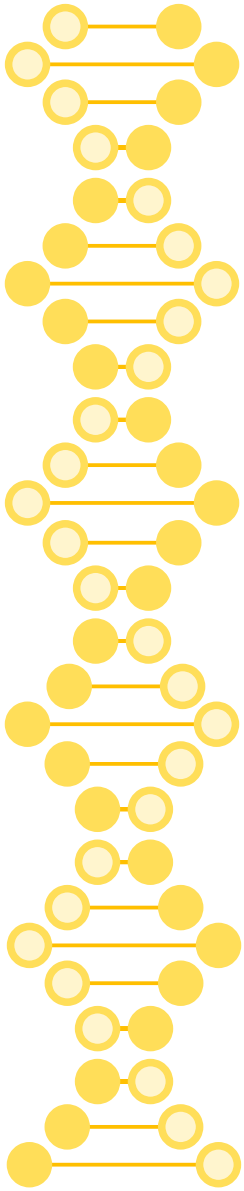
names $x, y, z, \ldots$

action terms	A ::=	$\bar{x}z.P$	send z along x
		$xy.P$	receive any y along x
terms	P ::=	$A_1 + \cdots + A_n$	alternative action ($n \geq 0$)
		$P_1 \mid P_2$	composition
		$\nu y P$	restriction
		$!P$	replication

(*the occurrences of y are binding*)

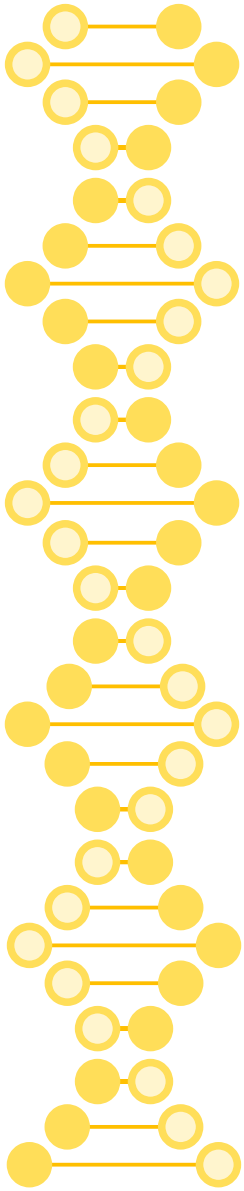
basic rule of computation $xy.P_1[y] \mid \bar{x}z.P_2 \rightarrow P_1[z] \mid P_2$


talk1
switch1
CAR(talk, switch) def= talk word. CAR(talk, switch)
+ switch t. switch s. CAR(t, s)
STATION1
STATION2
CONTROL
NETWORK def= ν talk1 ν switch1 ··· (CAR(talk1, switch1) | STATION1 | ··· | CONTROL)



In operating systems...

- Message passing is an Interprocess Communication (IPC) mechanism
- Special semantics and memory protection
  - Ordering of events is based on messages
    - No need for mutexes, shared memory and semaphores, *etc.*
  - Multicast and broadcast
  - Security and reliability benefits?



A little MacOS history...

https://en.wikipedia.org/wiki/Altair_8800

(1974)

https://en.wikipedia.org/wiki/Apple_I

(1976)

Because the Apple I did not include a case, customers needed to supply their own.

https://en.wikipedia.org/wiki/Apple_II

(1977)

https://en.wikipedia.org/wiki/Apple_Lisa
(1983)

MEXICO'S EARTHQUAKE TRAGEDY

Newsweek

A WHIZ KID'S FALL

How Apple Computer Dumped Its Chairman

Shoot-out in Silicon Valley

An Exclusive Interview With Steve Jobs

STEVE JOBS: OUT FOR REVENGE

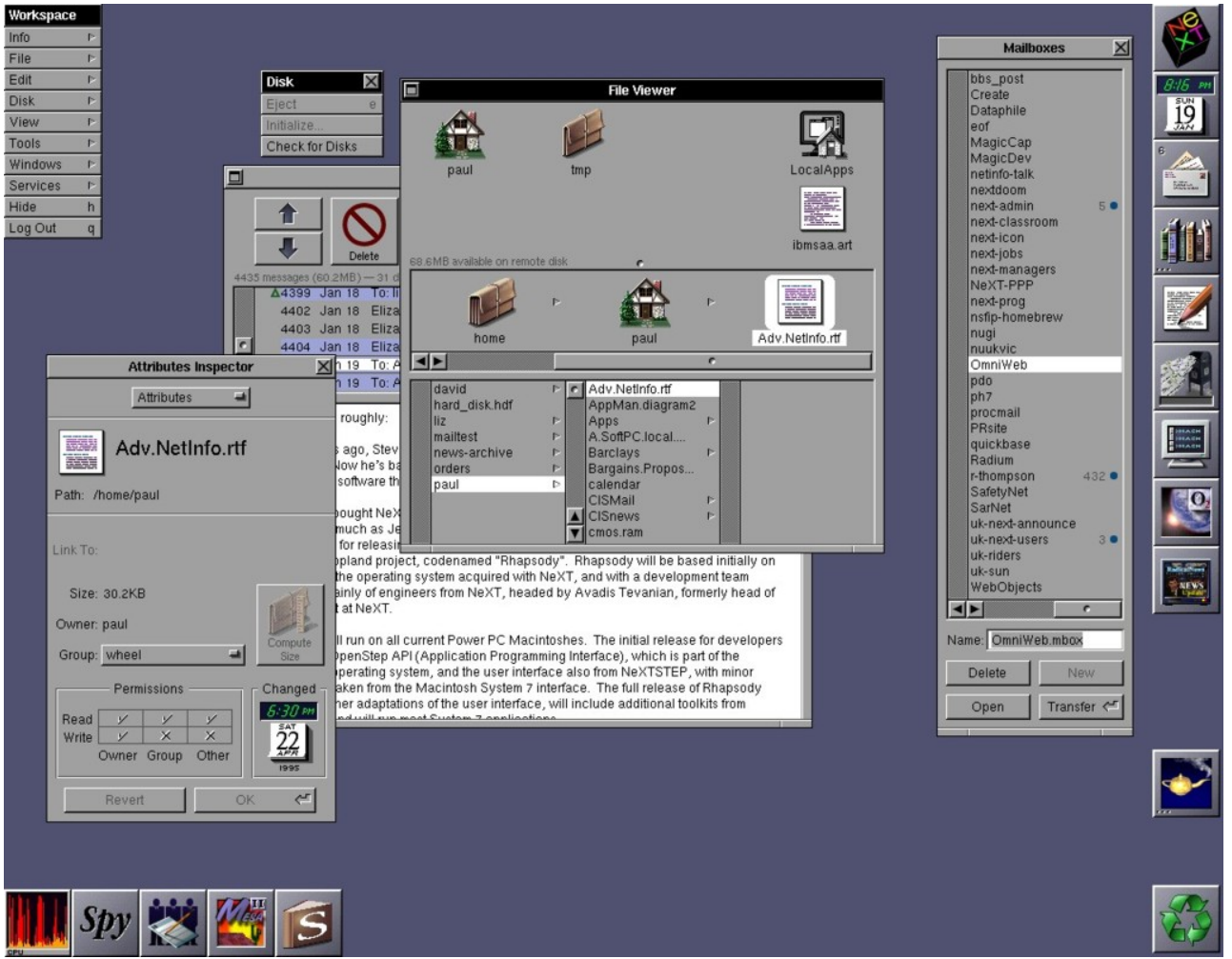
By Phil Patton

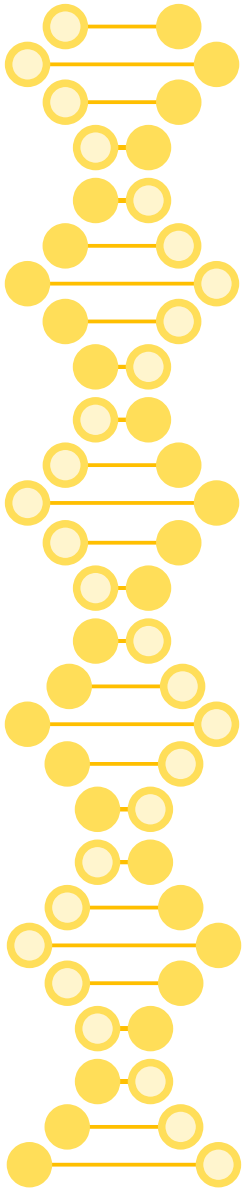
Ousted from Apple, the computer legend is back on the fast track with a $10,000 PC loaded with power.

https://en.wikipedia.org/wiki/NeXTcube

(1990)

https://en.wikipedia.org/wiki/NeXTSTEP

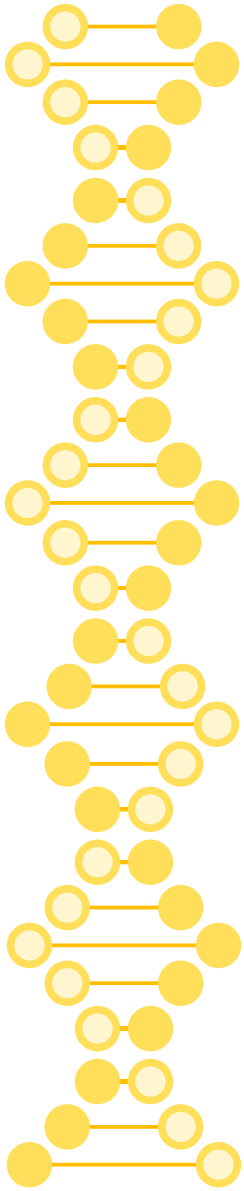




Have you heard of any of these?

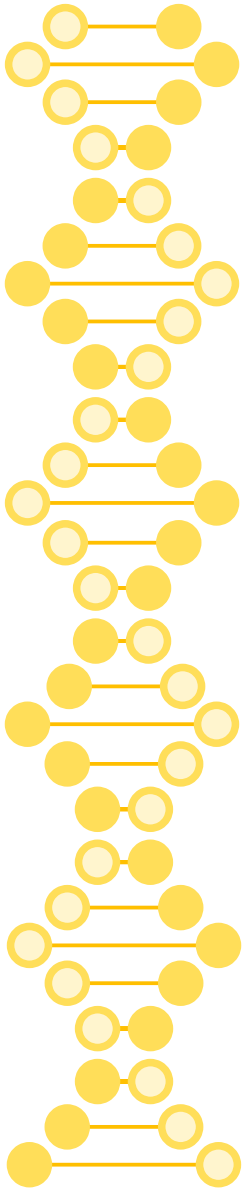
- Pixar
- Doom
- Quake
- The World Wide Web


30
AMMO
100%
HEALTH
2 3 4
5 6
ARMS
88%
ARMOR
BULL
88 / 200
30 / 50
6 / 50
260 / 300



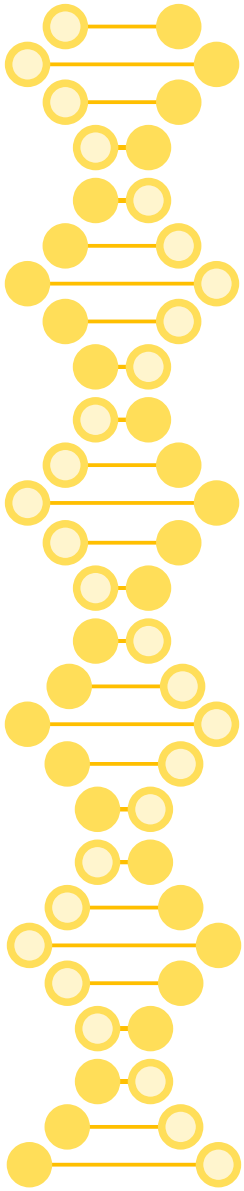
NextSTEP

- Combination of Mach and FreeBSD
- Objective-C
- An object oriented application layer known as “kits”
- The “Dock” in the GUI
- First app store



https://en.wikipedia.org/wiki/Mach_(kernel) (1985 to 1994)

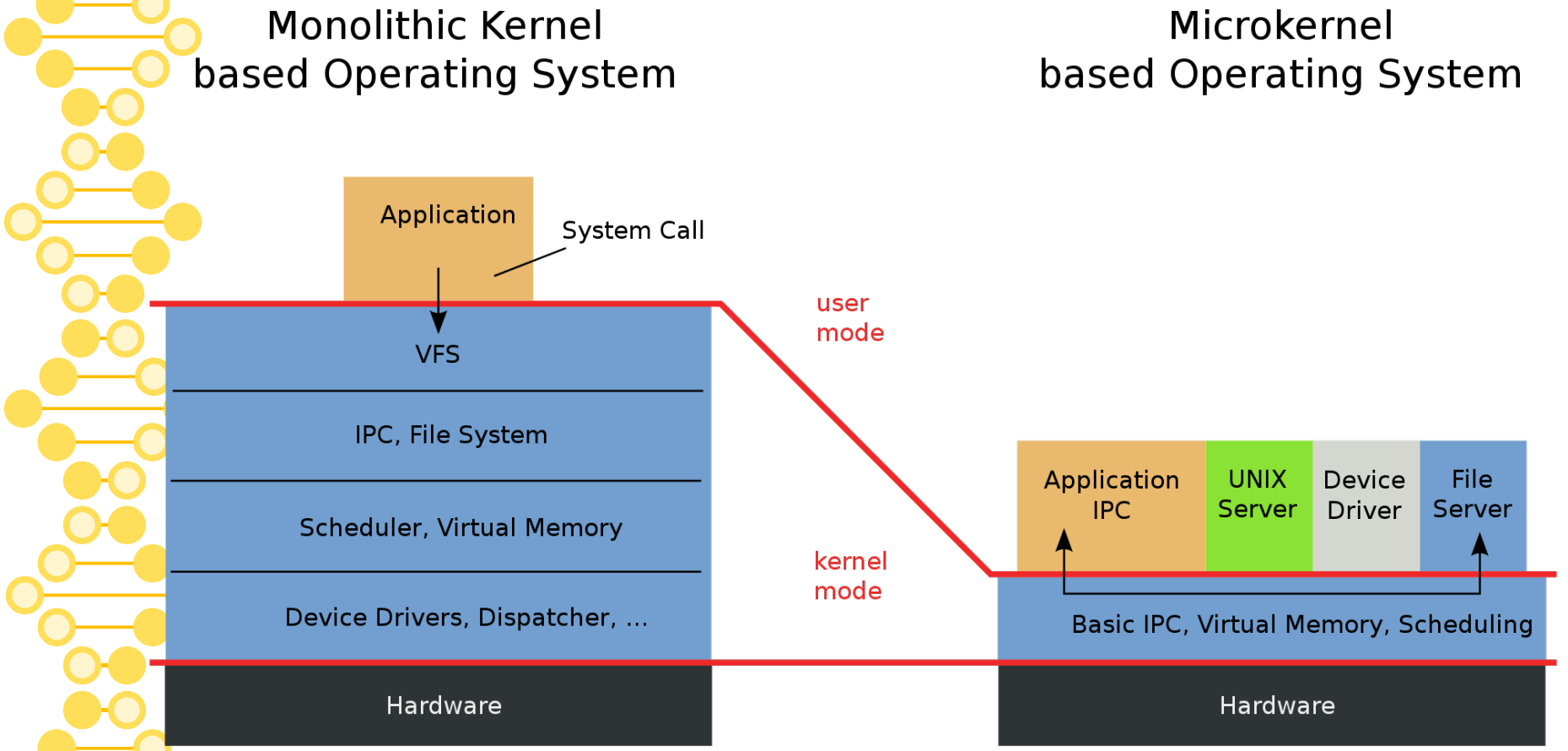
- Microkernel replacement for UNIX
- Started as a monolithic kernel and evolved into a microkernel
- Basis for…
  - GNU Hurd
  - XNU
    - macOS, iOS, iPadOS, tvOS, and watchOS

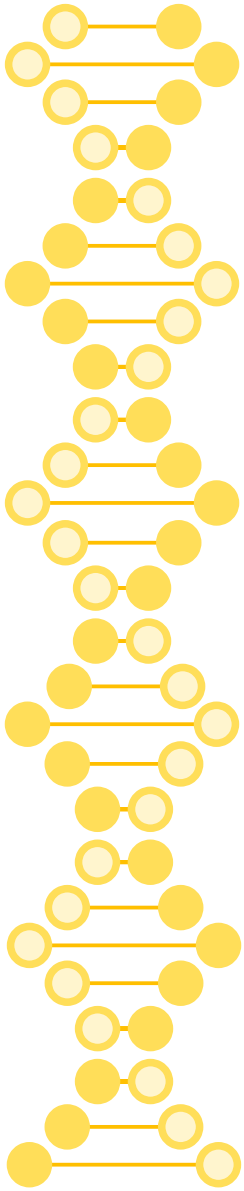


UNIX in the 1980s

- Everything is all about pipes
- Networking, device drivers, *etc.*
  - A lot of complexity being added
- Aleph kernel at Univ. of Rochester … OS is modular and communicates over pipes
  - Added shared memory
- Mach based on message passing

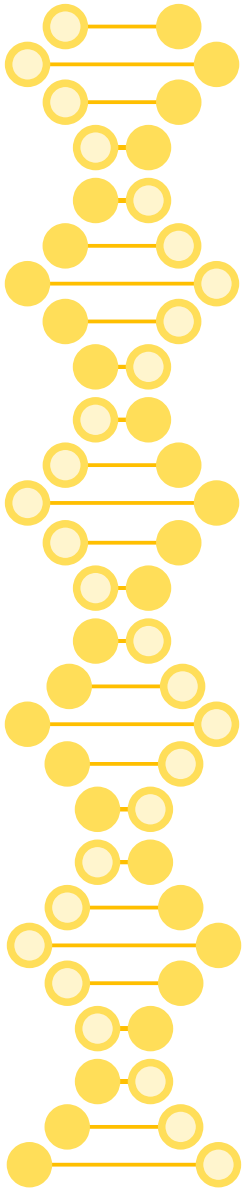
https://en.wikipedia.org/wiki/Microkernel





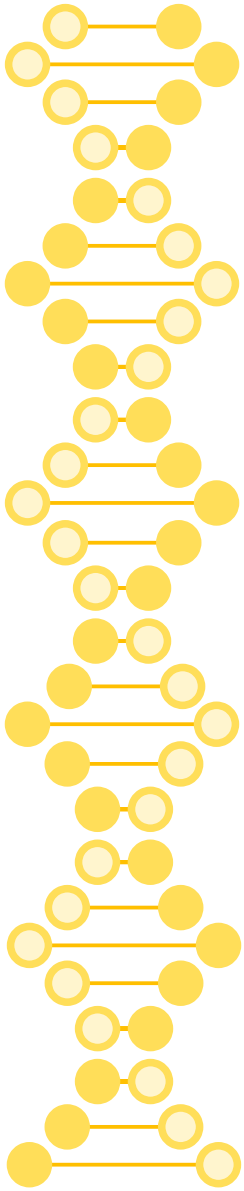
Supposed benefits of microkernels

- More modular
  - A crash in part of the OS doesn’t crash the system?
  - Plug and play parts of the OS?
- Makes muiltiple CPUs and distributed computing easier
  - Based on message passing
- More secure?
  - Filesystem, *etc.* can be in userspace
  - What about transitivity?



Truth about microkernels

- Context switches will always be expensive
  - TLB flushes, virtualization
- As far as I can tell, the Mach messaging layer in MacOS is there for historical reasons only
- The most impressive things about “message passing” on MacOS are object-oriented stuff happening in the GUI (AFAIK --- prove me wrong)



Message passing is still important

- Easy way to do IPC and concurrency
  - Advantages over…
    - Pipes?
    - Stream sockets?
    - Datagram sockets?
    - Shared memory?
- Beautiful academic theories if you need them
  - Mobility
- Supercomputing
  - RPC can’t broadcast and muilticast