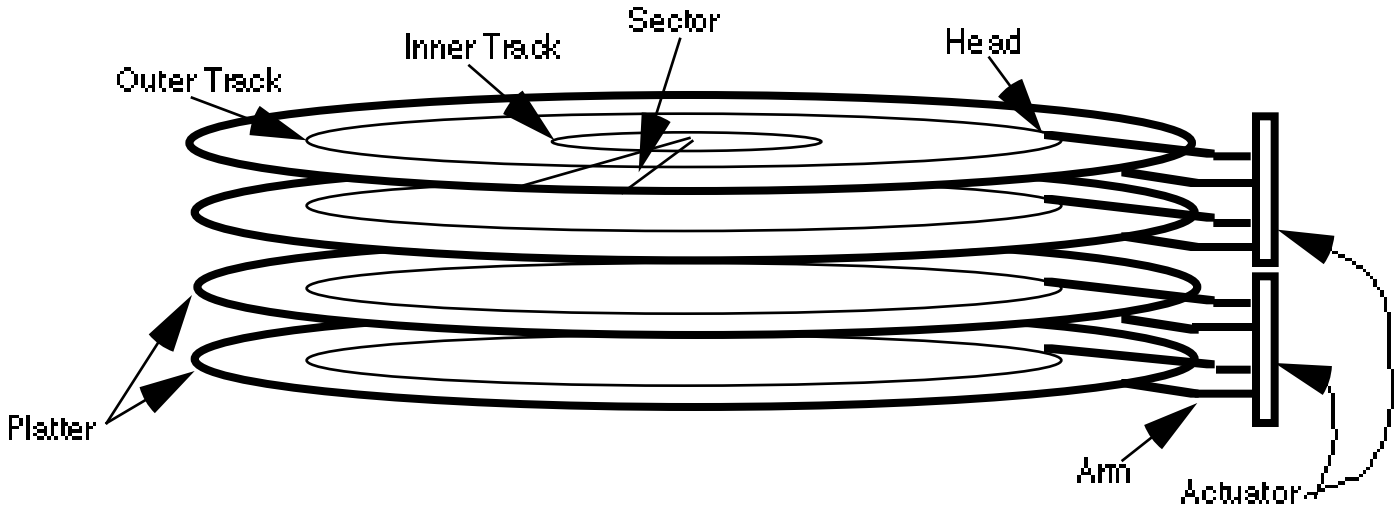
Lecture 12: I/O: Metrics, A Little Queuing Theory, and Busses

Professor David A. Patterson

Computer Science 252

Fall 1996

Review: Disk Device Terminology



Disk Latency = Queuing Time + Seek Time + Rotation Time + Xfer Time

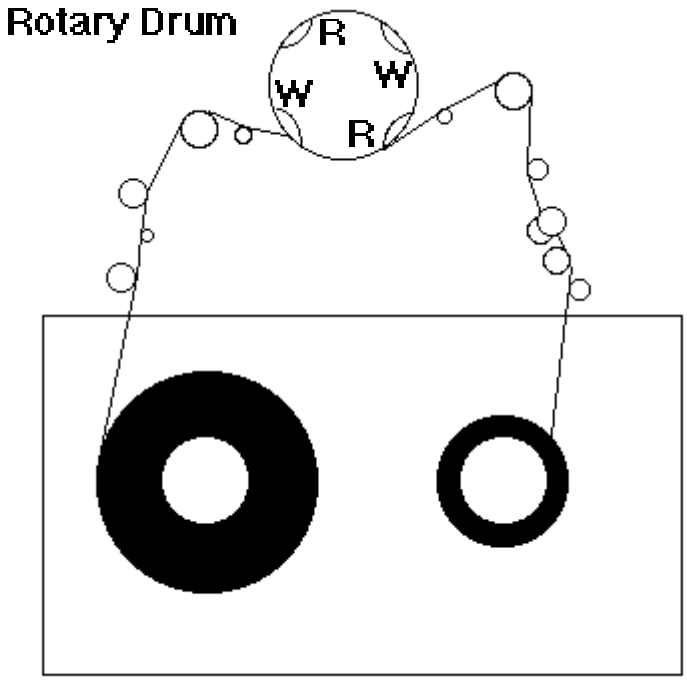
Order of magnitude times for 4K byte transfers:

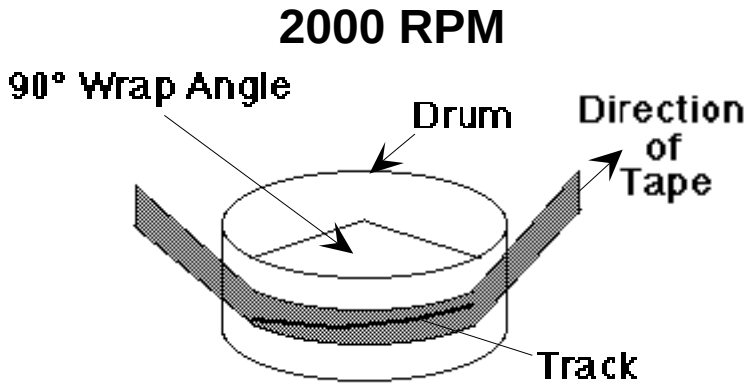
Seek: 12 ms or less

Rotate: 4.2 ms @ 7200 rpm (8.3 ms @ 3600 rpm)

Xfer: 1 ms @ 7200 rpm (2 ms @ 3600 rpm)

Review: R-DAT Technology





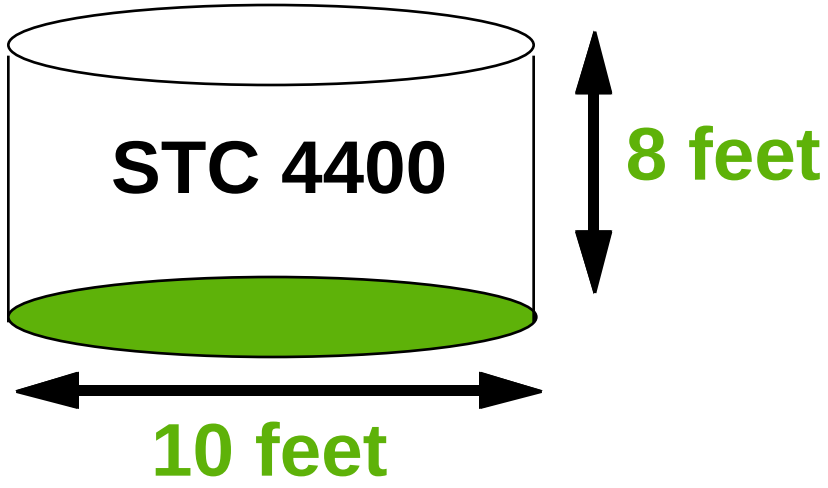
Four Head Recording

Tracks Recorded ±20° w/o guard band

Read After Write Verify

Helical Recording Scheme

Review: Automated Cartridge System



6000 x 0.8 GB 3490 tapes = 5 TBytes in 1992
$500,000 O.E.M. Price

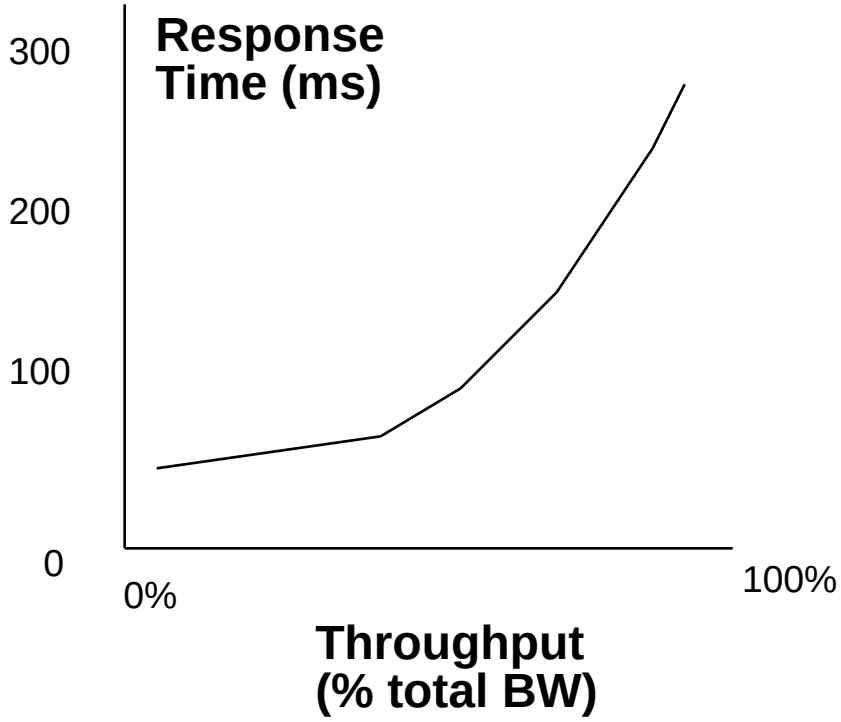
6000 x 20 GB D3 tapes = 120 TBytes in 1994
1 Petabyte (1024 TBytes) in 2000

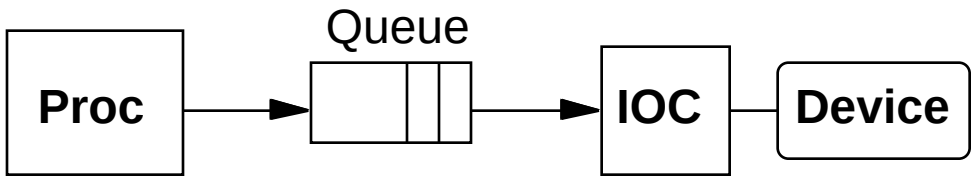
Review: Storage System Issues

- Historical Context of Storage I/O
- Secondary and Tertiary Storage Devices
- Storage I/O Performance Measures
- A Little Queuing Theory
- Processor Interface Issues
- I/O Buses
- Redundant Arrarys of Inexpensive Disks (RAID)
- ABCs of UNIX File Systems
- I/O Benchmarks
- Comparing UNIX File System Performance

Disk I/O Performance

Metrics:
- **Response Time**
- **Throughput**





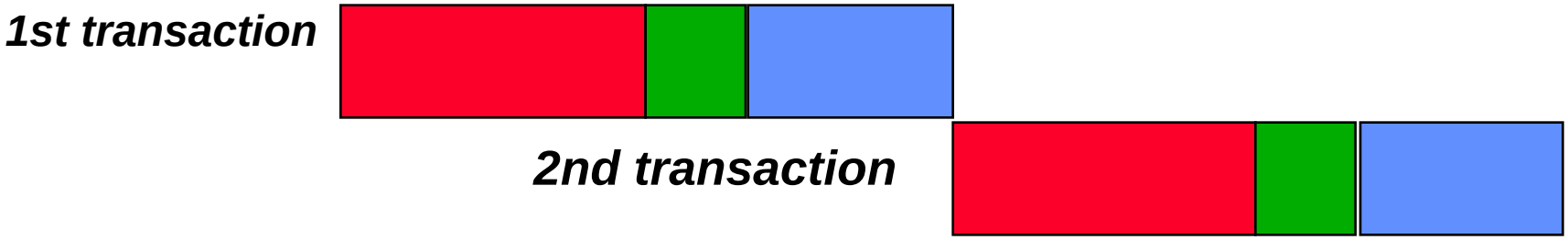
Response time = Queue + Device Service time

Response Time vs. Productivity

- **Interactive environments:**

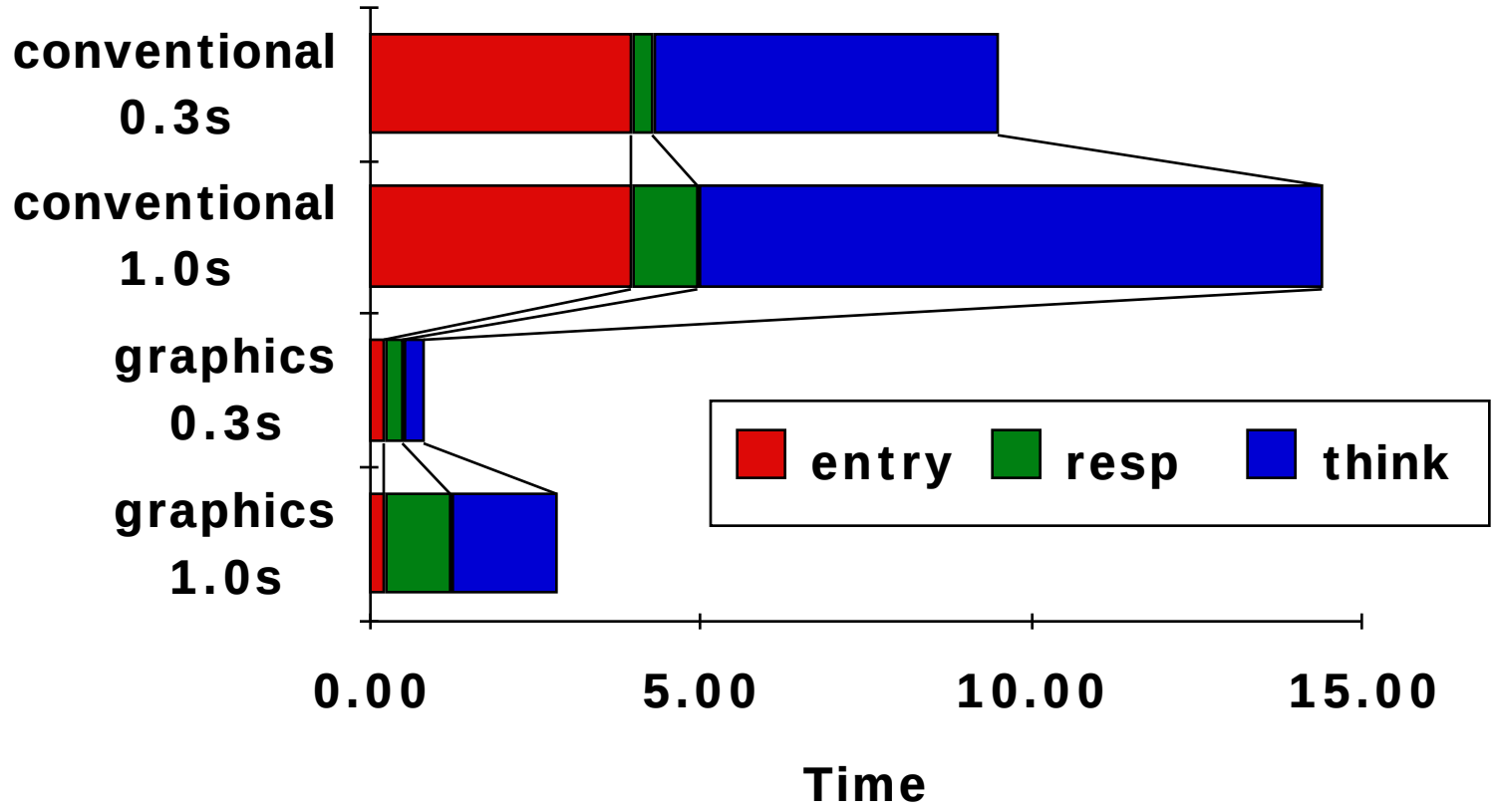
  Each interaction or *transaction* has 3 parts:

  - ***Entry Time*: time for user to enter command**
  - ***System Response Time*: time between user entry & system replies**
  - ***Think Time*: Time from response until user begins next command**



- **What happens to transaction time as shrink system response time from 1.0 sec to 0.3 sec?**
  - **With Keyboard: 4.0 sec entry, 9.4 sec think time**
  - **With Graphics: 0.25 sec entry, 1.6 sec think time**

Response Time & Productivity



- 0.7sec off response saves 4.9 sec (34%) and 2.0 sec (70%) total time per transaction => greater productivity
- Another study: everyone gets more done with faster response, but novice with fast response = expert with slow

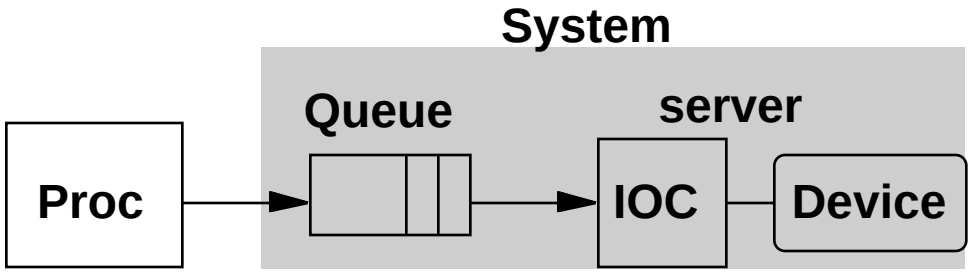
Disk Time Example

- **Disk Parameters:**
  - **Transfer size is 8K bytes**
  - **Advertised average seek is 12 ms**
  - **Disk spins at 7200 RPM**
  - **Transfer rate is 8 MB/sec**
- **Controller overhead is 1 ms**
- **Assume that disk is idle so no queuing delay**
- **What is Average Disk Access Time for a Sector?**
  - **Ave seek + ave rot delay + transfer time + controller overhead**
  - **12 ms + 0.5/(7200 RPM/60) + 8 KB/8 MB/s + 1 ms**
  - **12 + 4.15 + 1 + 1 = 18 ms**
- **Advertised seek time assumes no locality: typically 1/4 to 1/3 advertised seek time: 18 ms => 10 ms**

Introduction to Queueing Theory

- More interested in long term, steady state than in startup => Arrivals = Departures
- Little's Law: Mean number tasks in system = arrival rate x mean reponse time
- Applies to any system in equilibrium, as long as nothing in black box is creating or destroying tasks

A Little Queuing Theory: Notation

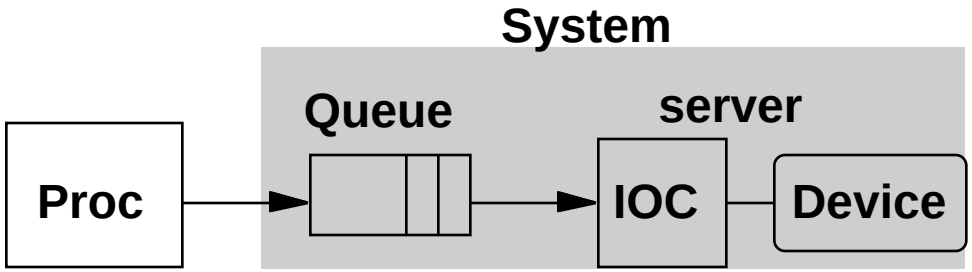


- **Queuing models assume state of equilibrium: input rate = output rate**
- **Notation:**

r	average number of arriving customers/second
T_{ser}	average time to service a customer (tradtionally $\mu = 1/ T_{ser}$)
u	server utilization (0..1): $u = r \times T_{ser}$
T_q	average time/customer in queue
T_{sys}	average time/customer in system: $T_{sys} = T_q + T_{ser}$
L_q	average length of queue: $L_q = r \times T_q$
L_{sys}	average length of system : $L_{sys} = r \times T_{sys}$

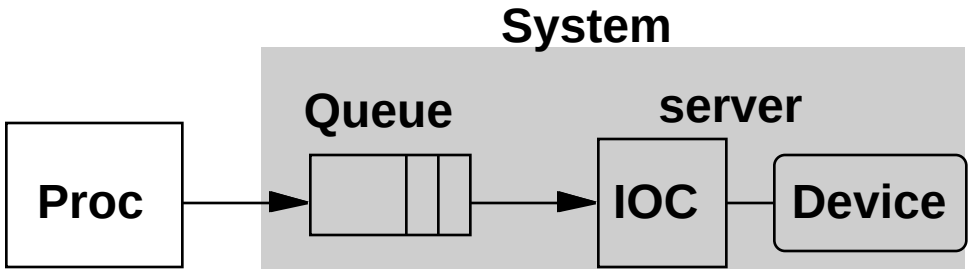
- **Little's Law:** $Length_{system} = rate \times Time_{system}$
  (Mean number customers = arrival rate x mean service time)

A Little Queuing Theory



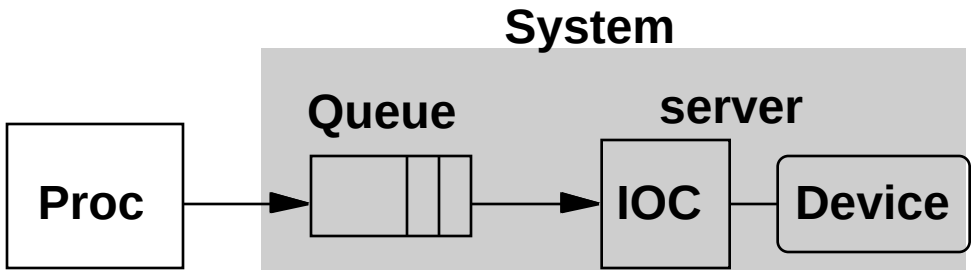
- **Service time completions vs. waiting time for a busy server when randomly arriving event joins a queue of arbitrary length when server is busy, otherwise serviced immediately**
- **A *single server queue*: combination of a servicing facility that accomodates 1 customer at a time (*server*) + waiting area (*queue*): together called a *system***
- **Server spends a variable amount of time with customers; *how do you characterize variability?***
  - **Distribution of a random variable: histogram? curve?**

A Little Queuing Theory



- **Server spends a variable amount of time with customers**
  - **Weighted mean *m1* = (f1 x T1 + f2 x T2 +...+ fn x Tn)/F (F=f1 + f2...)**
  - ***variance =* (f1 x $T1^2$ + f2 x $T2^2$ +...+ fn x Tn^2)/F – $m1^2$**
    - » **Changes depending on unit of measure (100 ms vs. 0.1 s)**
  - ***Squared coefficient of variance*: *C = variance/$m1^2$***
- **Exponential distribution *C = 1* : most short relative to average, few others long; 90% < 2.3 x average, 63% < average**
- **Hypoexponential distribution *C < 1* : most close to average, C=0.5 => 90% < 2.0 x average, only 57% < average**
- **Hyperexponential distribution *C > 1* : further from average C=2.0 => 90% < 2.8 x average, 69% < average**

A Little Queuing Theory: Variable Service Time



- **Server spends a variable amount of time with customers**
  - **Weighted mean *m1* = (f1xT1 + f2xT2 +...+ fnXTn)/F (F=f1+f2+...)**
  - **Squared coefficient of variance *C***
- **Disk response times *C ≈ 1.5* (majority seeks < average)**
- **Yet usually pick *C = 1.0* for simplicity**
- **Another useful value is average time must wait for server to complete task: *m1(z)***
  - **Not just 1/2 x m1 because doesn't capture variance**
  - **Can derive *m1(z) = 1/2 x m1 x (1 + C)***
  - ***No variance => C= 0 => m1(z) = 1/2 x m1***

A Little Queuing Theory: Average Wait Time

- **Calculating average wait time T_q**
  - **If something at server, it takes to complete on average *m1(z)***
  - **Chance server is busy = u; average delay is *u* x *m1(z)***
  - **All customers in line must complete; each avg T_{ser}**

$$T_q = u \times \underline{m1(z)} + L_q \times T_{s\,er} = 1/2 \times u \times T_{ser} \times (1 + C) + \underline{L_q} \times T_{s\,er}$$
$$T_q = 1/2 \times u \times T_{s\,er} \times (1 + C) + \underline{r} \times T_q \times \underline{T_{s\,er}}$$
$$T_q = 1/2 \times u \times T_{s\,er} \times (1 + C) + \underline{u \times T_q}$$
$$T_q \times \underline{(1 - u)} = T_{s\,er} \times u \times (1 + C)/2$$
$$T_q = T_{s\,er} \times u \times (1 + C) / (2 \times (1 - u))$$

- **Notation:**

r	average number of arriving customers/second
T_{ser}	average time to service a customer
u	server utilization (0..1): $u = r \times T_{ser}$
T_q	average time/customer in queue
L_q	average length of queue: $L_q = r \times T_q$

CS 252 Administrivia

- **Graded Midterm Quiz returned at end of class?**
- **Distribution of quiz scores**
  - **Mean**
  - **Median**
  - **Max**
  - **Min**
- **Enjoyed meeting everyone at LaVal's!**

A Little Queuing Theory: M/G/1 and M/M/1

- **Assumptions so far:**
  - **System in equilibrium**
  - **Time between two successive arrivals in line are random**
  - **Server can start on next customer immediately after prior finishes**
  - **No limit to the queue: works First-In-First-Out**
  - **Afterward, all customers in line must complete; each avg T_{ser}**
- **Described "memoryless" Markovian request arrival (M for C=1 exponentially random), General service distribution (no restrictions), 1 server: *M/G/1 queue***
- **When Service times have C = 1, *M/M/1 queue***

$$T_q = T_{ser} \times u \times (1 + C) / (2 \times (1 - u)) = T_{ser} \times u / (1 - u)$$

T_{ser} average time to service a customer
u server utilization (0..1): $u = r \times T_{ser}$
T_q average time/customer in queue

A Little Queuing Theory: An Example

- Suppose processor sends 10 x 8KB disk I/Os per second, requests exponentially distrib., disk service time = 20 ms
- On average, how utilized is the disk?
  - What is the number of requests in the queue?
  - What is the average time spent in the queue?
  - What is the average response time for a disk request?
- Notation:

r average number of arriving customers/second = 10

T_{ser} average time to service a customer = 20 ms

u server utilization (0..1): $u = r \times T_{ser}$ = 10/s x .02s = 0.2

T_q average time/customer in queue $= T_{ser} \times u / (1 - u)$
= 20 x 0.2/(1-0.2) = 20 x 0.25 = 5 ms

T_{sys} average time/customer in system: $T_{sys} = T_q + T_{ser}$ = 25 ms

L_q average length of queue: $L_q = r \times T_q$
= 10/s x .005s = 0.05 requests in wait line

L_{sys} average # tasks in system: $L_{sys} = r \times T_{sys}$ = 10/s x .025s = 0.25

A Little Queuing Theory: Another Example

- Suppose processor sends 20 x 8KB disk I/Os per sec, requests exponentially distrib., disk service time = 12 ms
- On average, how utilized is the disk?
  - What is the number of requests in the queue?
  - What is the average time a spent in the queue?
  - What is the average response time for a disk request?
- Notation:

r average number of arriving customers/second= ___

T_{ser} average time to service a customer= __

u server utilization (0..1): $u = r \times T_{ser}$= __/s x __s = __

T_q average time/customer in queue $= T_{ser} \times u / (1 - u)$
= __x __/(1-__) = _ x _ = _ ms

T_{sys} average time/customer in system: $T_{sys} = T_q + T_{ser}$= _ ms

L_q average length of queue: $L_q = r \times T_q$
= 20/s x .0038s = 0.016 requests in wait line

L_{sys} average # tasks in system : $L_{sys} = r \times T_{sys}$ = __/s x __s = __

A Little Queuing Theory: Yet Another Example

- Suppose processor sends 10 x 8KB disk I/Os per second, squared coef. var.(C) = 1.5, disk service time = 20 ms
- On average, how utilized is the disk?
  - What is the number of requests in the queue?
  - What is the average time a spent in the queue?
  - What is the average response time for a disk request?
- Notation:

r average number of arriving customers/second= 10

T_{ser} average time to service a customer= 20 ms

u server utilization (0..1): $u = r \times T_{ser}$= 10/s x .02s = 0.2

T_q average time/customer in queue $= T_{ser} \times u \times (1 + C) /(2 \times (1 - u))$
= 20 x 0.2(2.5)/2(1 – 0.2) = 20 x 0.32 = 6.25 ms

T_{sys} average time/customer in system: $T_{sys} = T_q + T_{ser}$= 26 ms

L_q average length of queue:$L_q = r \times T_q$
= 10/s x .006s = 0.06 requests in wait line

L_{sys} average # tasks in system :$L_{sys} = r \times T_{sys}$ = 10/s x .026s = 0.26

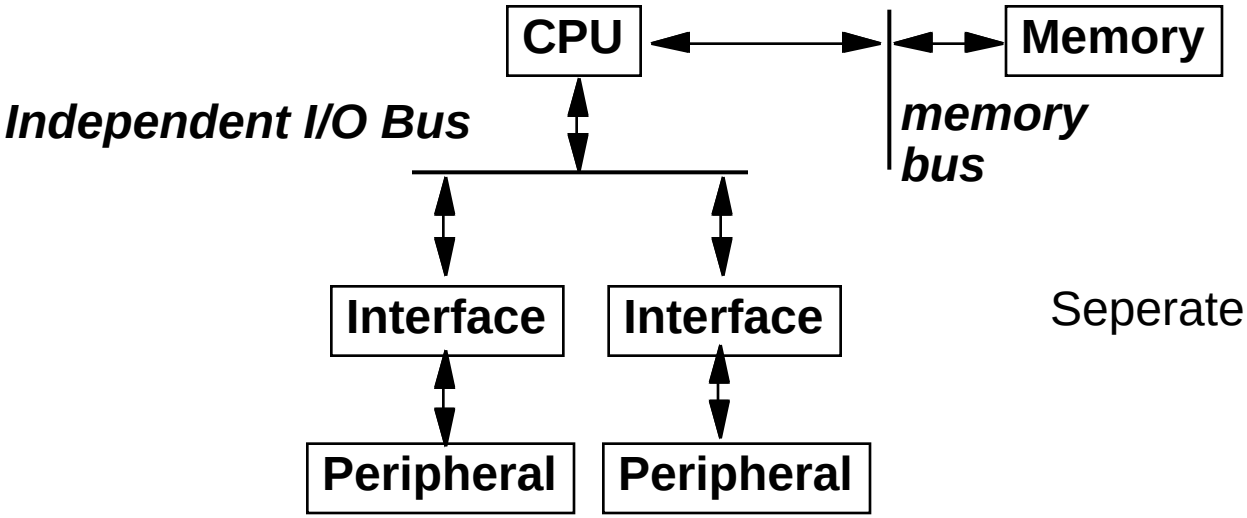
Storage System Issues

- Historical Context of Storage I/O
- Secondary and Tertiary Storage Devices
- Storage I/O Performance Measures
- A Little Queuing Theory
- Processor Interface Issues
- I/O Buses
- Redundant Arrarys of Inexpensive Disks (RAID)
- ABCs of UNIX File Systems
- I/O Benchmarks
- Comparing UNIX File System Performance

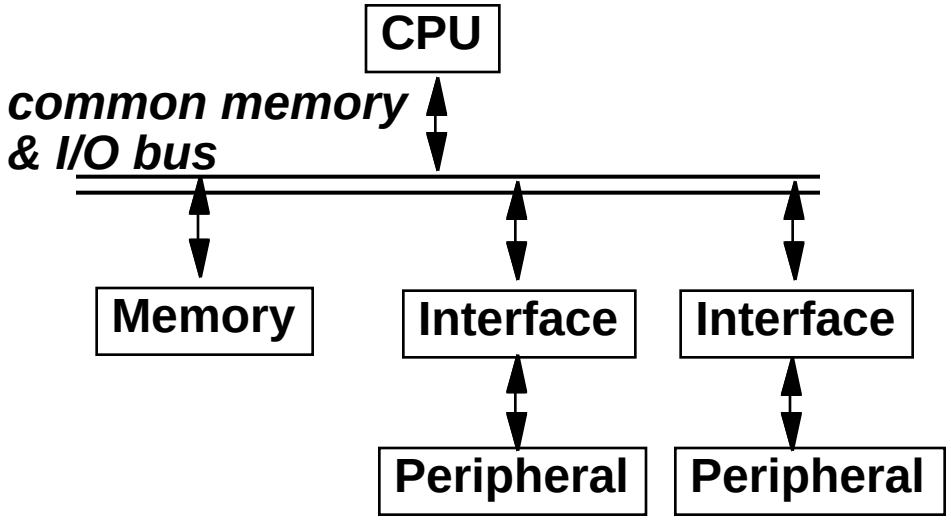
Processor Interface Issues

- **Interconnections**
  - **Busses**
- **Processor interface**
  - **Interrupts**
  - **Memory mapped I/O**
- **I/O Control Structures**
  - **Polling**
  - **Interrupts**
  - **DMA**
  - **I/O Controllers**
  - **I/O Processors**
- **Capacity, Access Time, Bandwidth**

I/O Interface



Seperate I/O instructions (in,out)



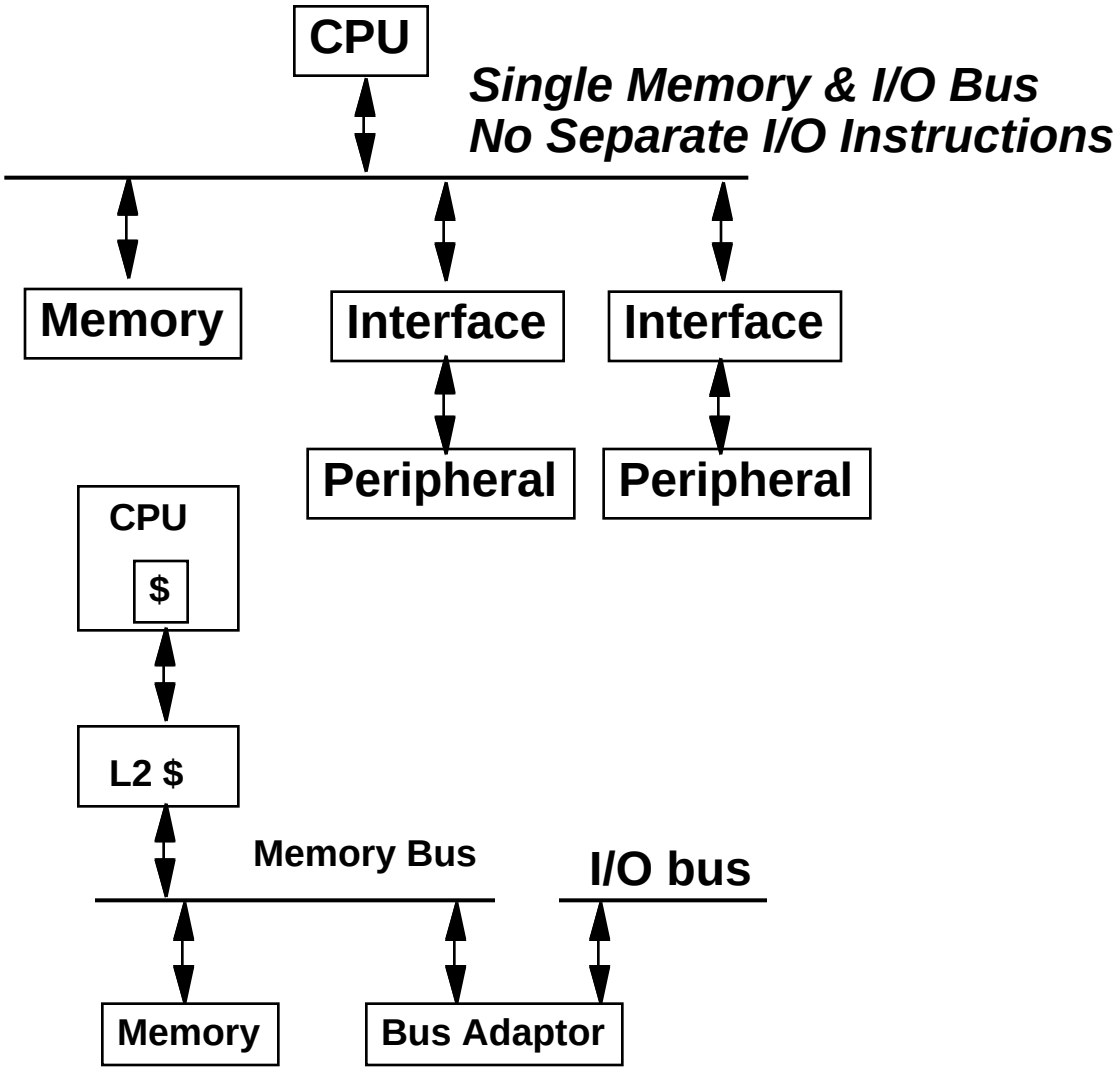
Lines distinguish between
I/O and memory transfers

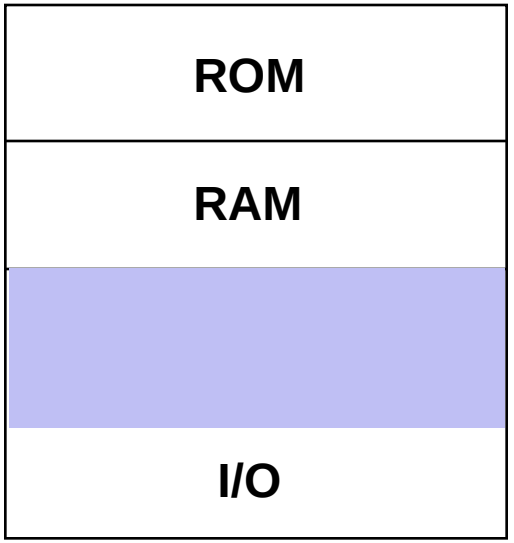
VME bus
Multibus-II
Nubus

40 Mbytes/sec
optimistically

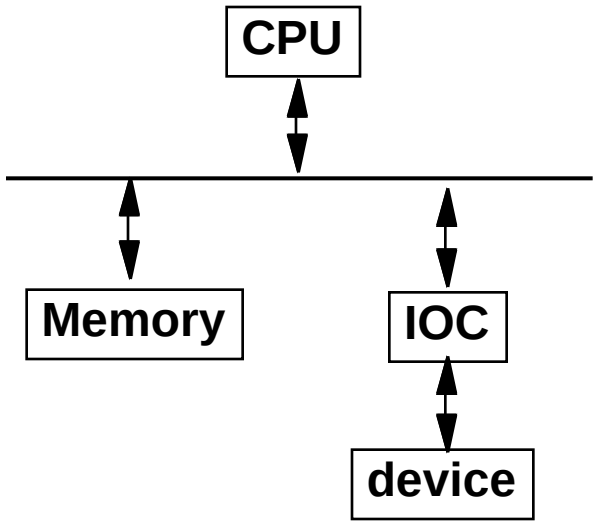
10 MIP processor
completely
saturates the bus!

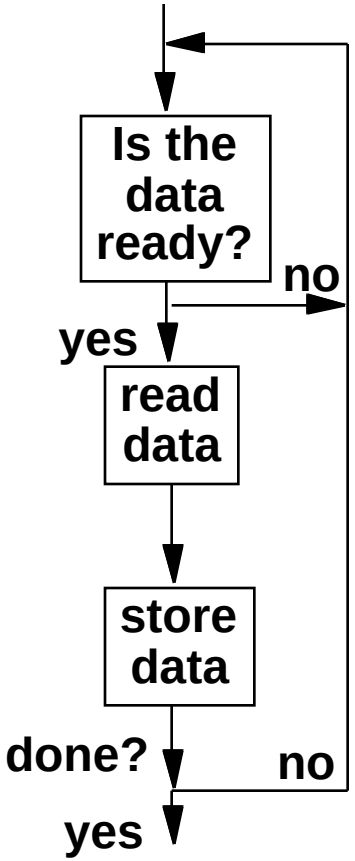
Memory Mapped I/O





Programmed I/O (Polling)

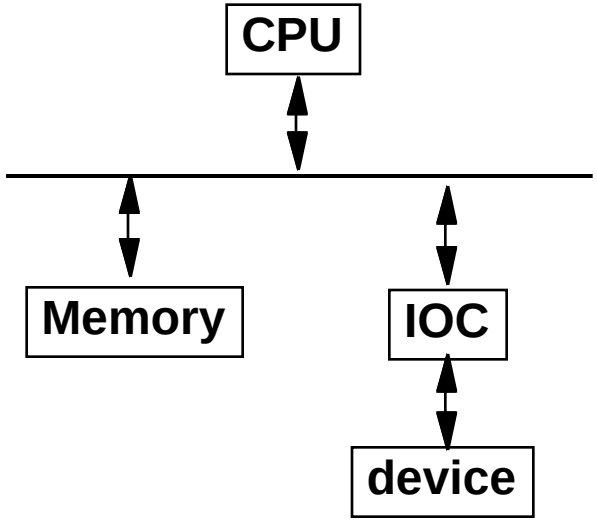


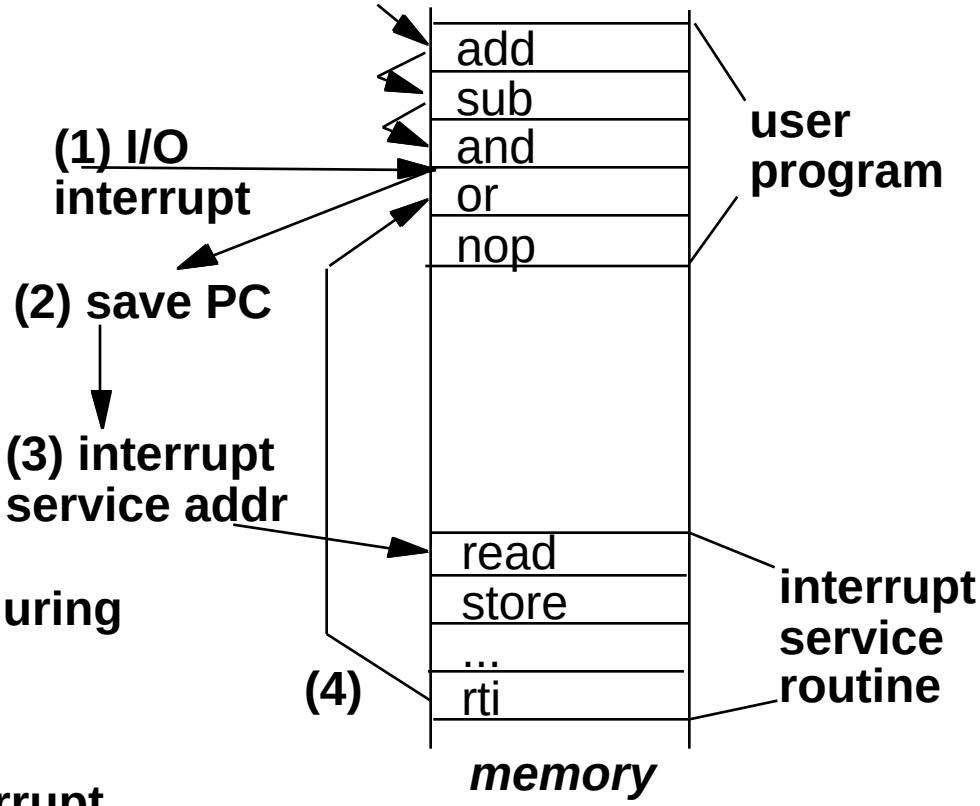


busy wait loop not an efficient way to use the CPU unless the device is very fast!

but checks for I/O completion can be dispersed among computationally intensive code

Interrupt Driven Data Transfer





User program progress only halted during actual transfer

1000 transfers at 1 ms each:
1000 interrupts @ 2 µsec per interrupt
1000 interrupt service @ 98 µsec each = 0.1 CPU seconds

Device xfer rate = 10 MBytes/sec => 0 .1 x 10^{-6} sec/byte => 0.1 µsec/byte
=> 1000 bytes = 100 µsec
1000 transfers x 100 µsecs = 100 ms = 0.1 CPU seconds

Still far from device transfer rate! 1/2 in interrupt overhead

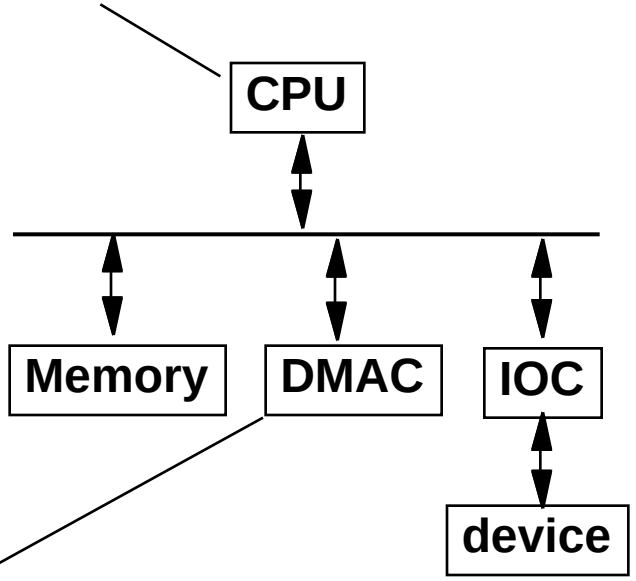
Direct Memory Access

Time to do 1000 xfers at 1 msec each:

1 DMA set-up sequence @ 50 µsec
1 interrupt @ 2 µsec
1 interrupt service sequence @ 48 µsec

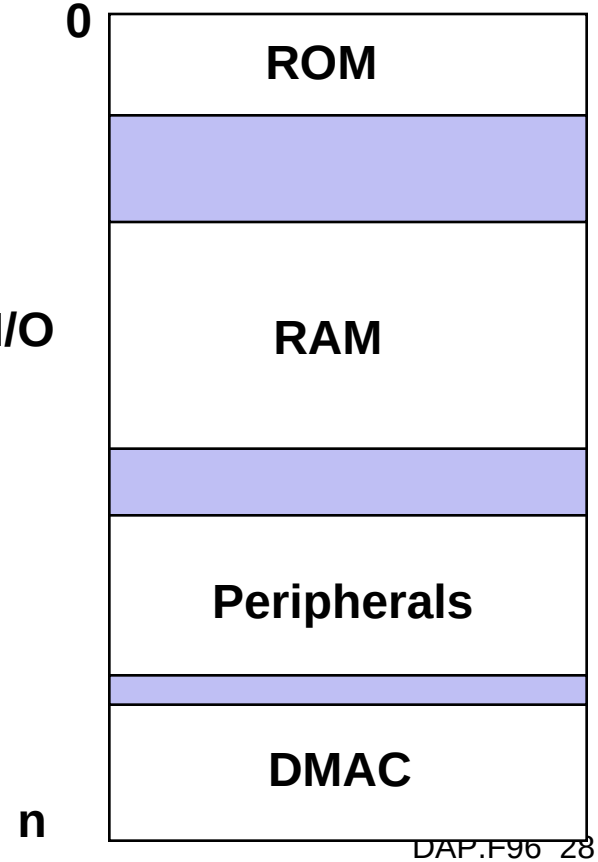
.0001 second of CPU time

CPU sends a starting address, direction, and length count to DMAC. Then issues "start".

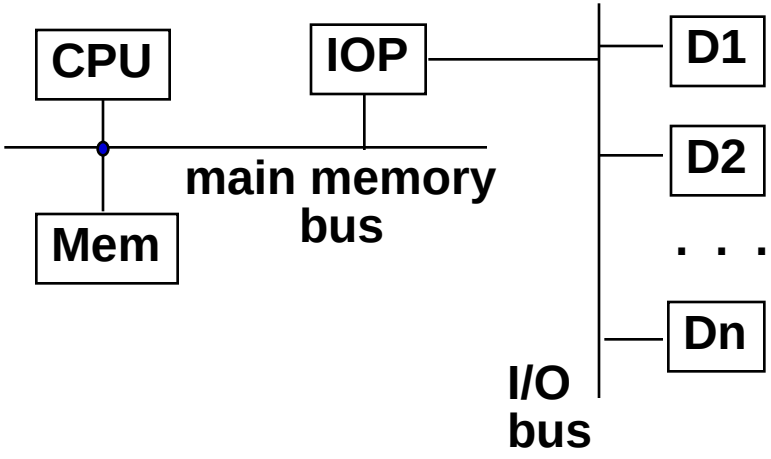


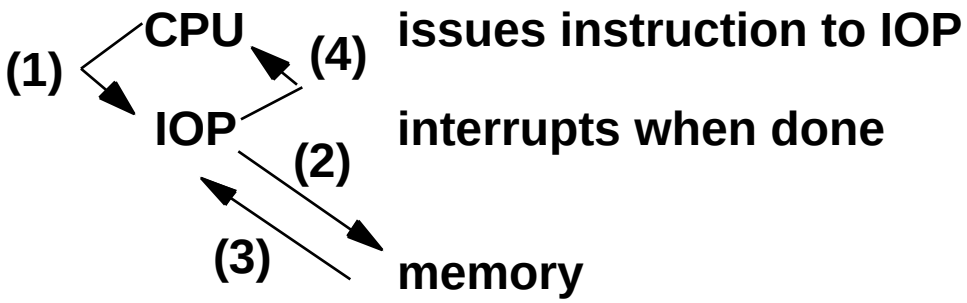
DMAC provides handshake signals for Peripheral Controller, and Memory Addresses and handshake signals for Memory.

Memory Mapped I/O



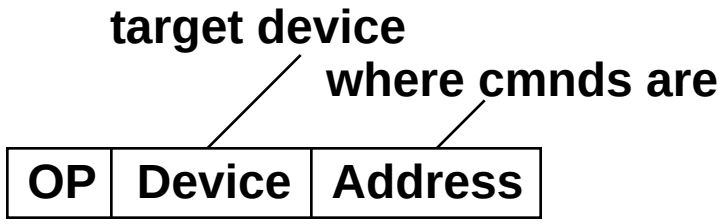
Input/Output Processors



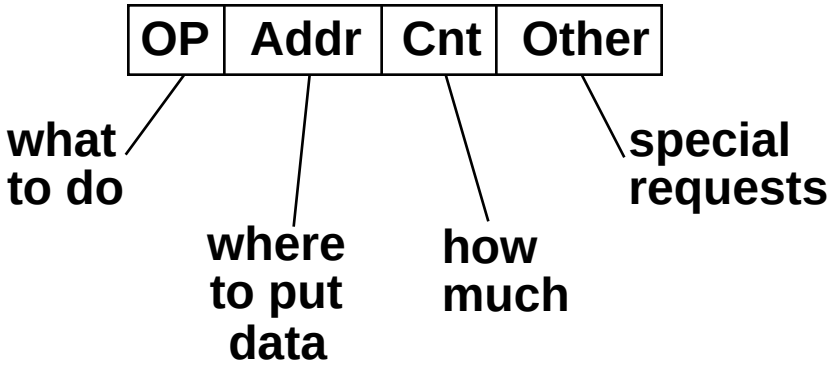


Device to/from memory transfers are controlled by the IOP directly.

IOP steals memory cycles.



looks in memory for commands



Relationship to Processor Architecture

- **I/O instructions have largely disappeared**
- **Interrupt vectors have been replaced by jump tables**
  PC <- M [IVA + interrupt number]
  PC <- IVA + interrupt number
- **Interrupts:**
  - **Stack replaced by shadow registers**
  - **Handler saves registers and re-enables higher priority int's**
  - **Interrupt types reduced in number; handler must query interrupt controller**

Relationship to Processor Architecture

- **Caches required for processor performance cause problems for I/O**
  - **Flushing is expensive, I/O polutes cache**
  - **Solution is borrowed from shared memory multiprocessors "snooping"**
- **Virtual memory frustrates DMA**
- **Load/store architecture at odds with atomic operations**
  - **load locked, store conditional**
- **Stateful processors hard to context switch**

5 minute Class Break

- **Lecture Format:**
  - ≈ 1 minute: review last time & motivate this lecture
  - ≈ 20 minute lecture
  - ≈ 3 minutes: discuss class manangement
  - ≈ 25 minutes: lecture
  - 5 minutes: break
  - ≈25 minutes: lecture
  - ≈1 minute: summary of today's important topics

Storage System Issues

- Historical Context of Storage I/O
- Secondary and Tertiary Storage Devices
- Storage I/O Performance Measures
- A Little Queuing Theory
- Processor Interface Issues
- I/O Buses
- Redundant Arrarys of Inexpensive Disks (RAID)
- ABCs of UNIX File Systems
- I/O Benchmarks
- Comparing UNIX File System Performance

Interconnect Trends

- **Interconnect = glue that interfaces computer system components**
- **High speed hardware interfaces + logical protocols**
- **Networks, channels, backplanes**

	Network	Channel	Backplane
Distance	>1000 m	10 - 100 m	1 m
Bandwidth	10 - 100 Mb/s	40 - 1000 Mb/s	320 - 1000+ Mb/s
Latency	high (>ms)	medium	low (<µs)
Reliability	low Extensive CRC	medium Byte Parity	high Byte Parity

message-based
narrow pathways
distributed arb

memory-mapped
wide pathways
centralized arb

Backplane Architectures

Metric	VME	FutureBus	MultiBus II	SCSI-I
Bus Width (signals)	128	96	96	25
Address/Data Multiplexed?	No	Yes	Yes	na
Data Width	16 - 32	32	32	8
Xfer Size	Single/Multiple	Single/Multiple	Single/Multiple	Single/Multiple
# of Bus Masters	Multiple	Multiple	Multiple	Multiple
Split Transactions	No	Optional	Optional	Optional
Clocking	Async	Async	Sync	Either
Bandwidth, Single Word (0 ns mem)	25	37	20	5, 1.5
Bandwidth, Single Word (150 ns mem)	12.9	15.5	10	5, 1.5
Bandwidth Multiple Word (0 ns mem)	27.9	95.2	40	5, 1.5
Bandwidth Multiple Word (150 ns mem)	13.6	20.8	13.3	5, 1.5
Max # of devices	21	20	21	7
Max Bus Length	.5 m	.5 m	.5 m	25 m
Standard	IEEE 1014	IEEE 896	ANSI/IEEE 1296	ANSI X3.131

Distinctions begin to blur:

SCSI channel is like a bus

FutureBus is like a channel (disconnect/reconnect)

HIPPI forms links in high speed switching fabrics

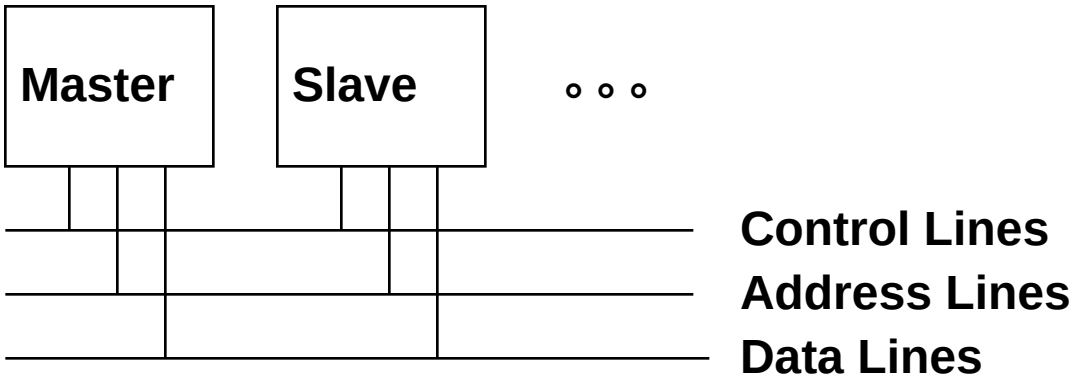
Bus-Based Interconnect

- **Bus: a shared communication link between subsystems**
  - **Low cost: a single set of wires is shared multiple ways**
  - **Versatility: Easy to add new devices & peripherals may even be ported between computers using common bus**
- **Disadvantage**
  - **A communication bottleneck, possibly limiting the maximum I/O throughput**
- **Bus speed is limited by physical factors**
  - **the bus length**
  - **the number of devices (and, hence, bus loading).**
  - **these physical limits prevent arbitrary bus speedup.**

Bus-Based Interconnect

- **Two generic types of busses:**
  - **I/O busses: lengthy, many types of devices connected, wide range in the data bandwidth), and follow a bus standard (sometimes called a *channel*)**
  - **CPU–memory buses: high speed, matched to the memory system to maximize memory–CPU bandwidth, single device (sometimes called a *backplane*)**
  - **To lower costs, low cost (older) systems combine together**
- **Bus transaction**
  - **Sending address & receiving or sending data**

Bus Protocols



Multibus: 20 address, 16 data, 5 control, 50ns Pause

Bus Master: has ability to control the bus, initiates transaction

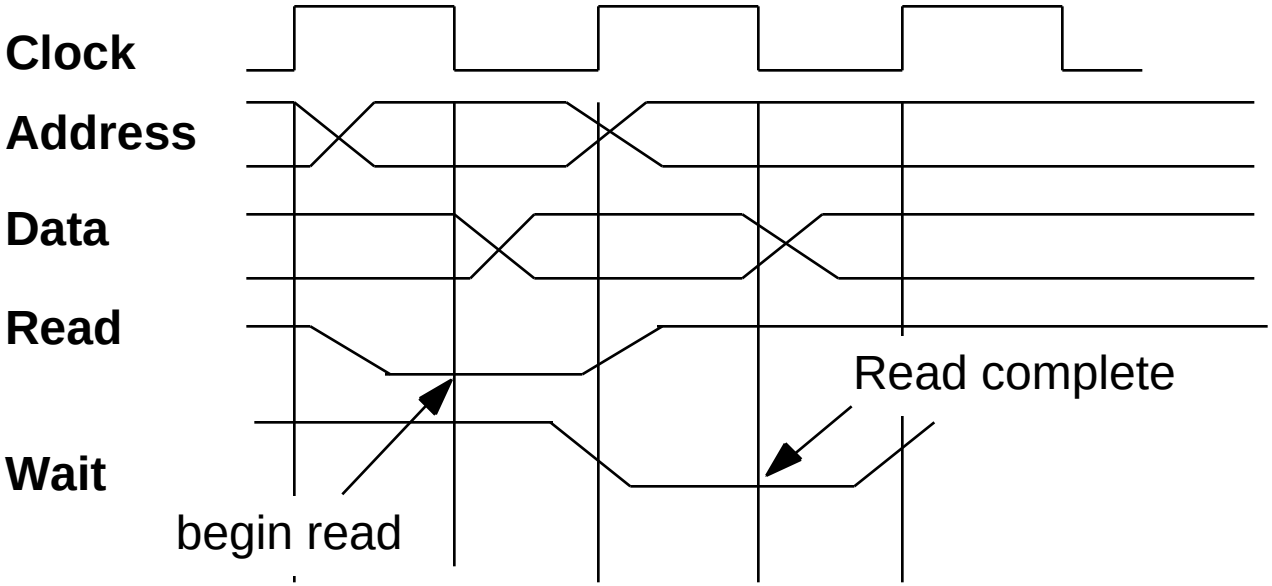
Bus Slave: module activated by the transaction

Bus Communication Protocol: specification of sequence of events and timing requirements in transferring information.

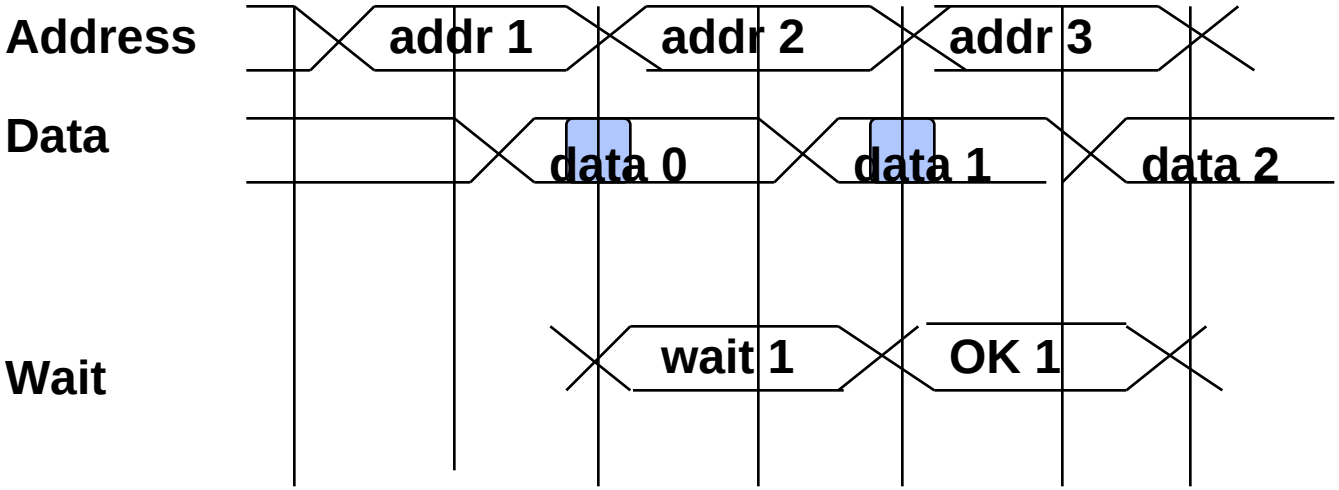
Asynchronous Bus Transfers: control lines (req., ack.) serve to orchestrate sequencing

Synchronous Bus Transfers: sequence relative to common clock

Synchronous Bus Protocols

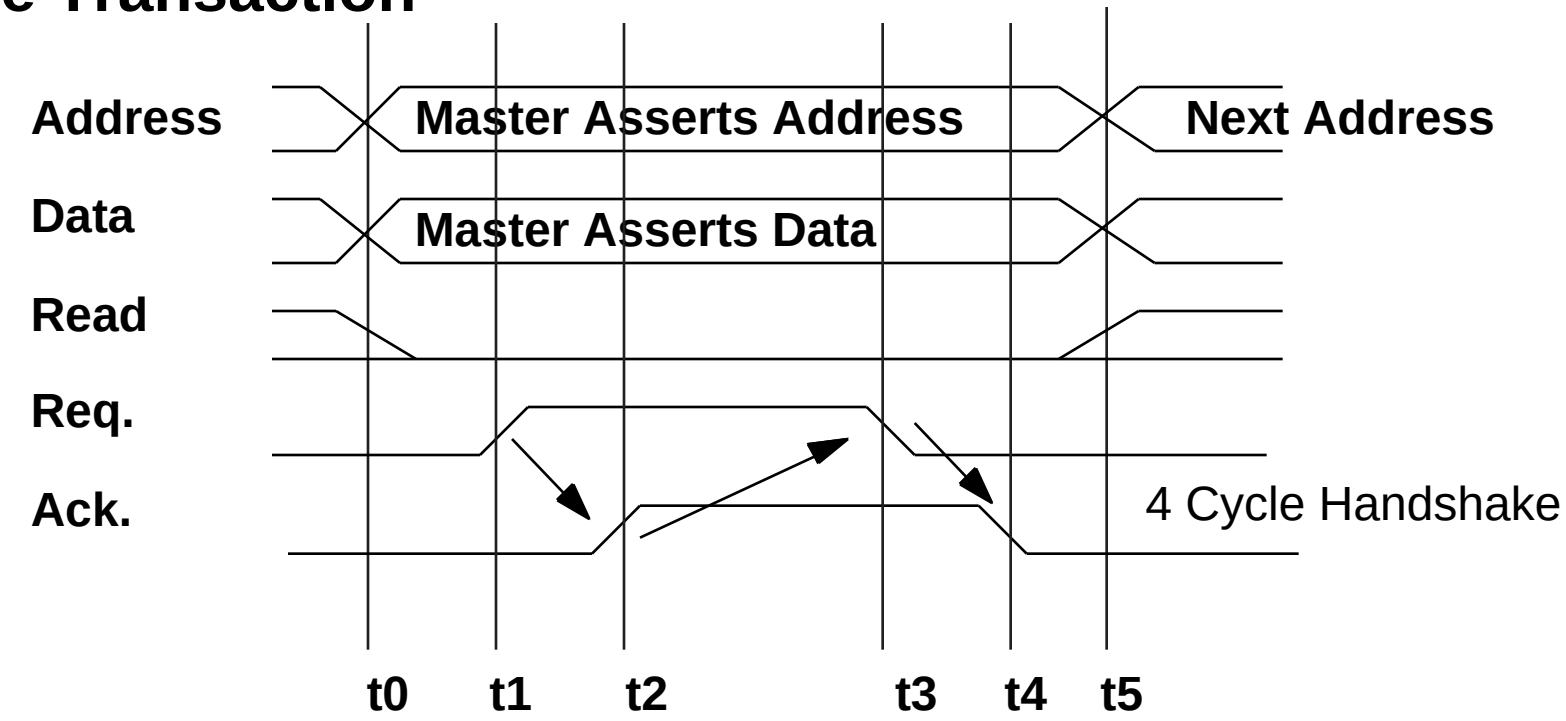


Pipelined/Split transaction Bus Protocol



Asynchronous Handshake

Write Transaction



t0 : Master has obtained control and asserts address, direction, data
Waits a specified amount of time for slaves to decode target

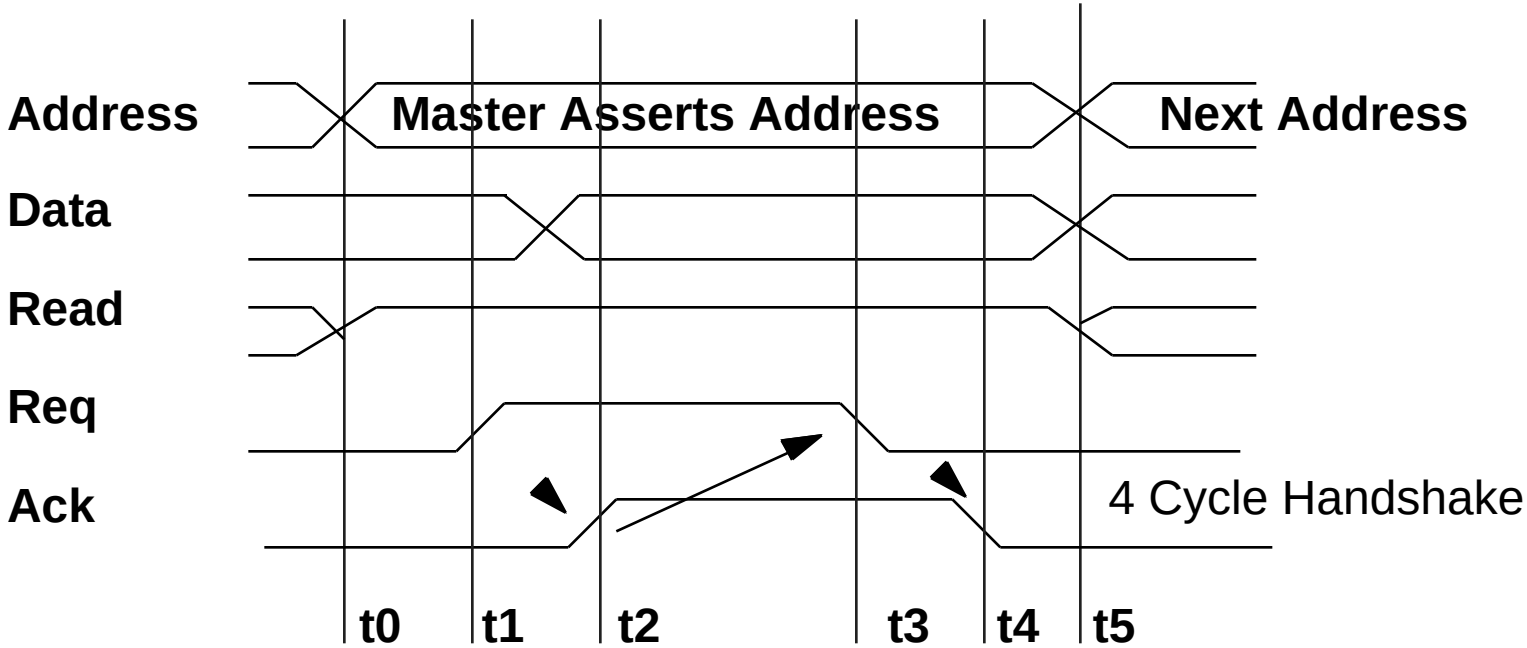
t1: Master asserts request line

t2: Slave asserts ack, indicating data received

t3: Master releases req

t4: Slave releases ack

Read Transaction



t0 : Master has obtained control and asserts address, direction, data
Waits a specified amount of time for slaves to decode target

t1: Master asserts request line

t2: Slave asserts ack, indicating ready to transmit data

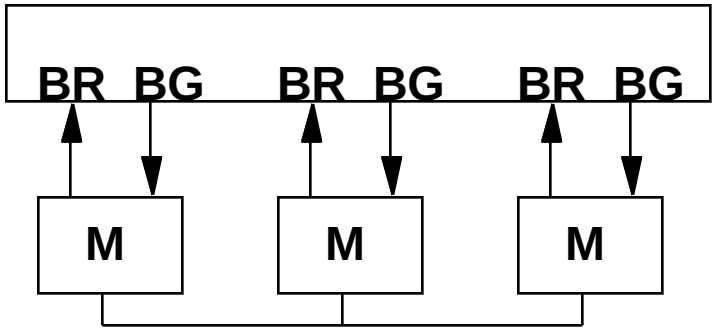
t3: Master releases req, data received

t4: Slave releases ack

Time Multiplexed Bus: address and data share lines

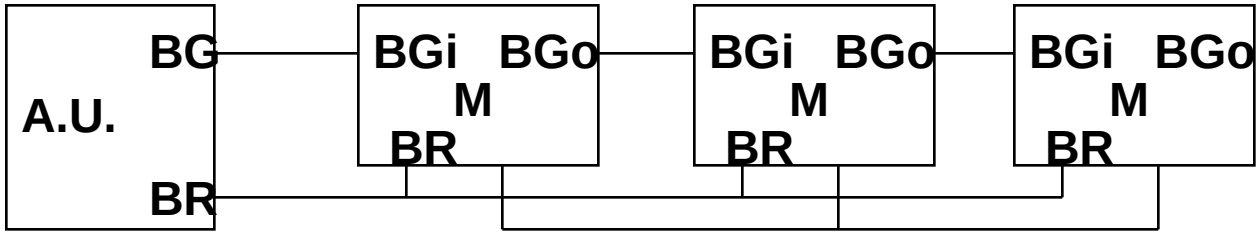
Bus Arbitration

Parallel (Centralized) Arbitration

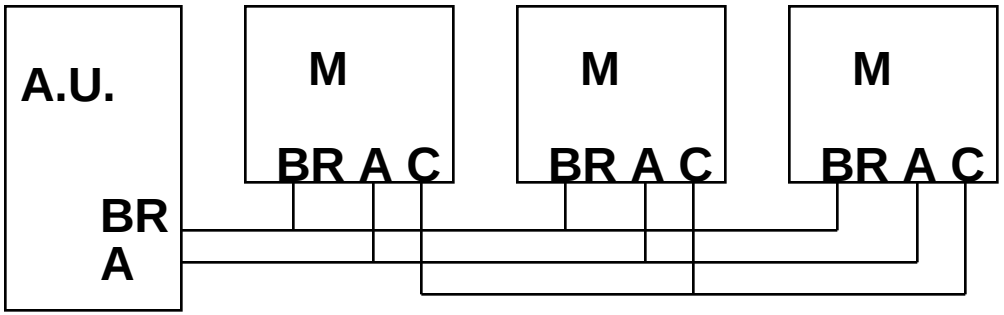


Bus Request
Bus Grant

Serial Arbitration (daisy chaining)



Polling



Bus Options

Option	*High performance*	*Low cost*
Bus width	Separate address & data lines	Multiplex address & data lines
Data width	Wider is faster (e.g., 32 bits)	Narrower is cheaper (e.g., 8 bits)
Transfer size	Multiple words has less bus overhead	Single-word transfer is simpler
Bus masters	Multiple (requires arbitration)	Single master (no arbitration)
Split transaction?	Yes—separate Request and Reply packets gets higher bandwidth (needs multiple masters)	No—continuous connection is cheaper and has lower latency
Clocking	Synchronous	Asynchronous

1990 Bus Survey (P&H, 1st Ed)

	VME	FutureBus	Multibus II	IPI	SCSI
Signals	128	96	96	16	8
Addr/Data mux	no	yes	yes	n/a	n/a
Data width	16 - 32	32	32	16	8
Masters	multi	multi	multi	single	multi
Clocking	Async	Async	Sync	Async	either
MB/s (0ns, word)	25	37	20	25	1.5 (asyn) 5 (sync)
150ns word	12.9	15.5	10	=	=
0ns block	27.9	95.2	40	=	=
150ns block	13.6	20.8	13.3	=	=
Max devices	21	20	21	8	7
Max meters	0.5	0.5	0.5	50	25
Standard	IEEE 1014	IEEE 896.1	ANSI/IEEE 1296	ANSI X3.129	ANSI X3.131

VME

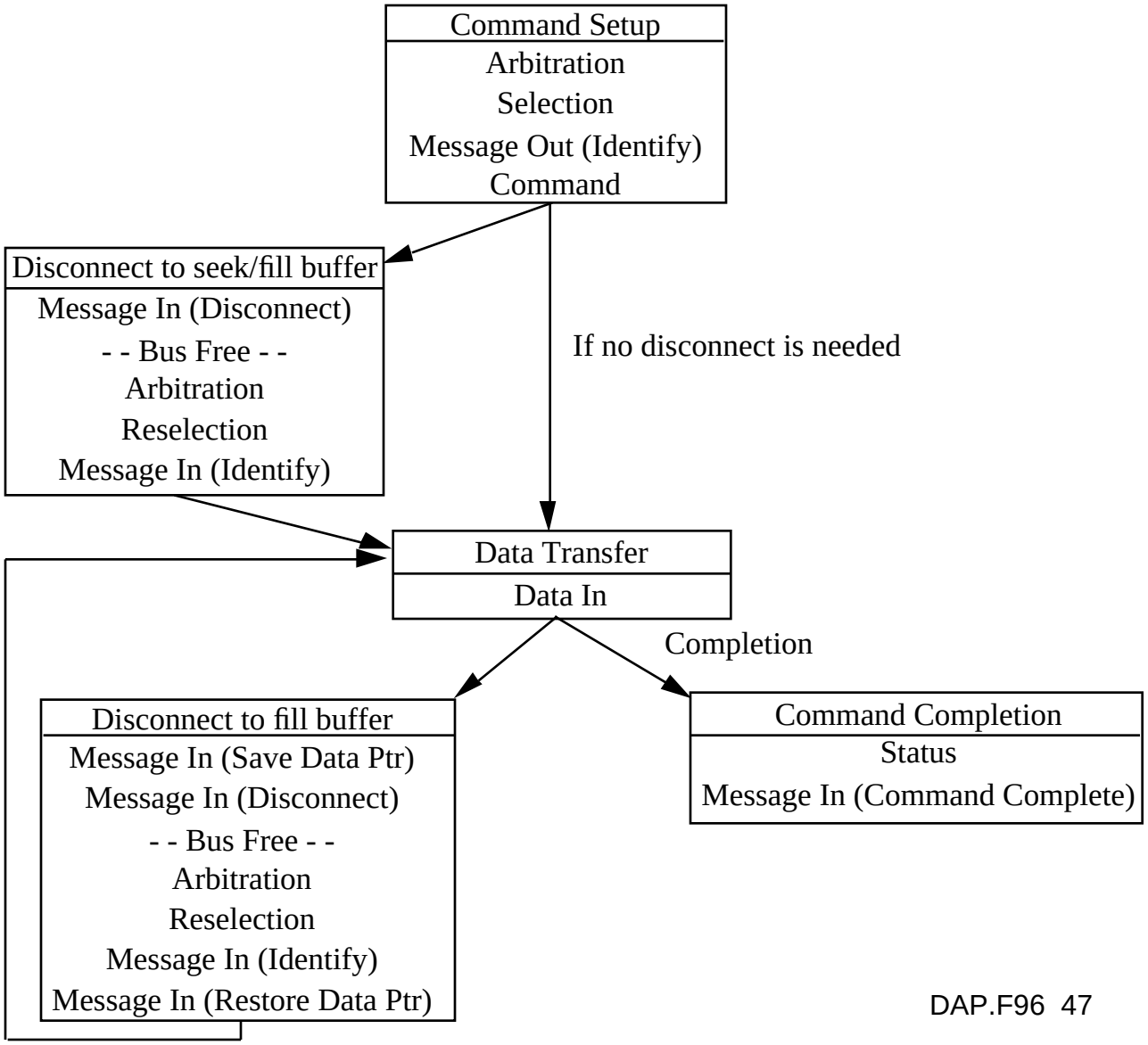
- 3 96-pin connectors
- 128 defined as standard, rest customer defined
  - 32 address
  - 32 data
  - 64 command & power/ground lines

SCSI: Small Computer System Interface

- **Clock rate: 5 MHz / 10 MHz (fast) / 20 MHz (ultra)**
- **Width: n = 8 bits / 16 bits (wide); up to n – 1 devices to communicate on a bus or "string"**
- **Devices can be slave ("target") or master("initiator")**
- **SCSI protocol: a series of "phases", during which specific actions are taken by the controller and the SCSI disks**
  - **Bus Free: No device is currently accessing the bus**
  - **Arbitration: When the SCSI bus goes free, multiple devices may request (arbitrate for) the bus; fixed priority by address**
  - **Selection: informs the target that it will participate (Reselection if disconnected)**
  - **Command: the initiator reads the SCSI command bytes from host memory and sends them to the target**
  - **Data Transfer: data in or out, initiator: target**
  - **Message Phase: message in or out, initiator: target (identify, save/restore data pointer, disconnect, command complete)**
  - **Status Phase: target, just before command complete**

SCSI “Bus”: Channel Architecture

peer-to-peer protocols
initiator/target
linear byte streams
disconnect/reconnect



1993 I/O Bus Survey (P&H, 2nd Ed)

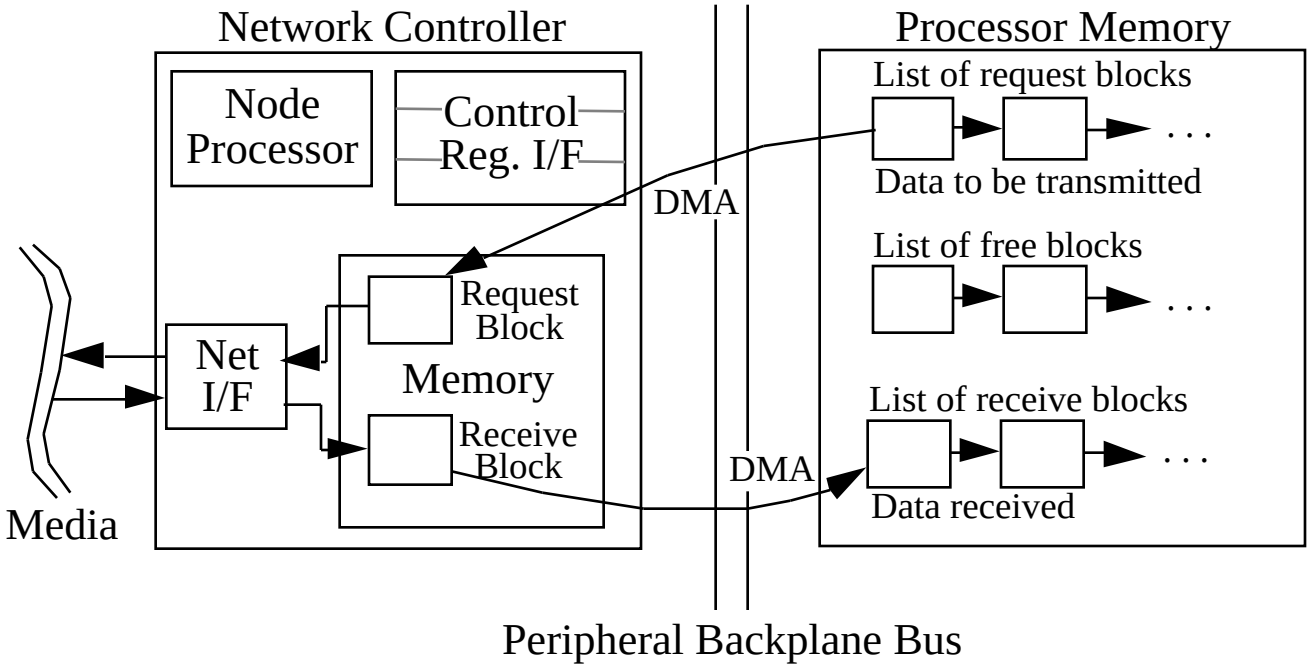
Bus	SBus	TurboChannel	MicroChannel	PCI
Originator	Sun	DEC	IBM	Intel
Clock Rate (MHz)	16-25	12.5-25	async	33
Addressing	Virtual	Physical	Physical	Physical
Data Sizes (bits)	8,16,32	8,16,24,32	8,16,24,32,64	8,16,24,32,64
Master	Multi	Single	Multi	Multi
Arbitration	Central	Central	Central	Central
32 bit read (MB/s)	33	25	20	33
Peak (MB/s)	89	84	75	111 (222)
Max Power (W)	16	26	13	25

1993 MP Server Memory Bus Survey

Bus	Summit	Challenge	XDBus
Originator	HP	SGI	Sun
Clock Rate (MHz)	60	48	66
Split transaction?	Yes	Yes	Yes?
Address lines	48	40	??
Data lines	128	256	144 (parity)
Data Sizes (bits)	512	1024	512
Clocks/transfer	4	5	4?
Peak (MB/s)	960	1200	1056
Master	Multi	Multi	Multi
Arbitration	Central	Central	Central
Addressing	Physical	Physical	Physical
Slots	16	9	10
Busses/system	1	1	2
Length	13 inches	12? inches	17 inches

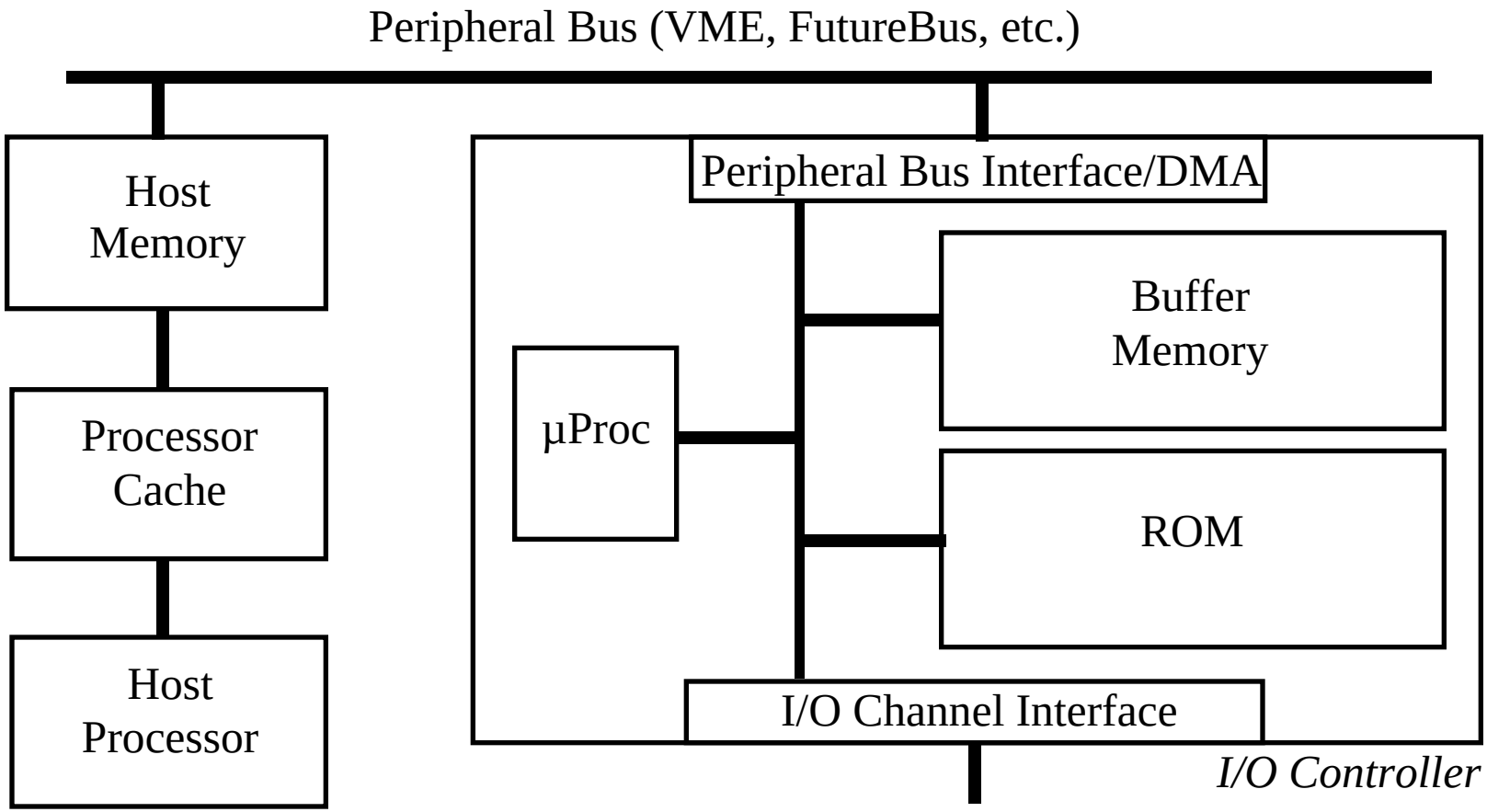
Communications Networks

Performance limiter is memory system, OS overhead



- **Send/receive queues in processor memories**
- **Network controller copies back and forth via DMA**
- **No host intervention needed**
- **Interrupt host when message sent or received**

I/O Controller Architecture

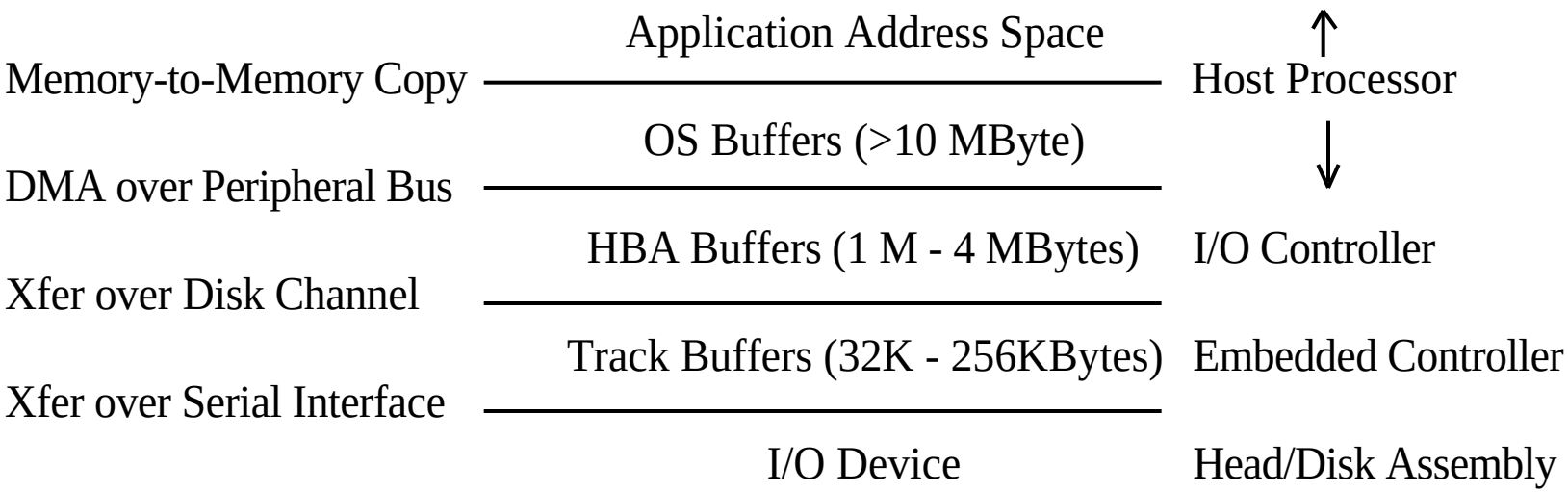


Request/response block interface

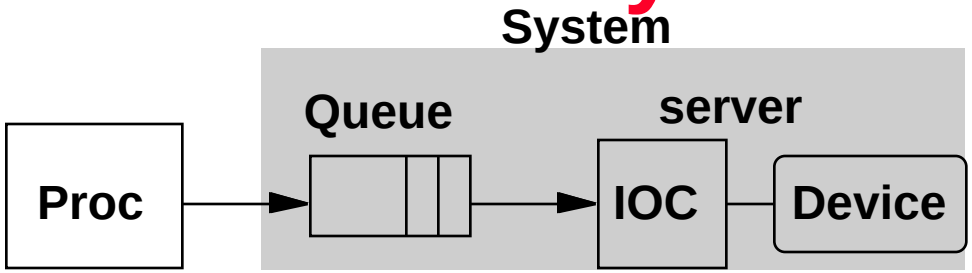
Backdoor access to host memory

I/O Data Flow

Impediment to high performance: multiple copies, complex hierarchy



Summary: A Little Queuing Theory



- **Queuing models assume state of equilibrium: input rate = output rate**
- **Notation:**

r	average number of arriving customers/second
T_{ser}	average time to service a customer (tradtionally $\mu = 1/ T_{ser}$)
u	server utilization (0..1): $u = r \times T_{ser}$
T_q	average time/customer in queue
T_{sys}	average time/customer in system: $T_{sys} = T_q + T_{ser}$
L_q	average length of queue: $L_q = r \times T_q$
L_{sys}	average length of system : $L_{sys} = r \times T_{sys}$

- **Little's Law: $Length_{system} = rate \times Time_{system}$**
  (Mean number customers = arrival rate x mean service time)

Summary: Processor Interface Issues

- **Processor interface**
  - **interrupts**
  - **memory mapped I/O**
- **I/O Control Structures**
  - **polling**
  - **interrupts**
  - **DMA**
  - **I/O Controllers**
  - **I/O Processors**

Summary: Relationship to Processor Architecture

- I/O instructions have disappeared
- Interrupt vectors have been replaced by jump tables
- Interrupt stack replaced by shadow registers
- Interrupt types reduced in number
- Caches required for processor performance cause problems for I/O
- Virtual memory frustrates DMA
- Load/store architecture at odds with atomic operations
- Stateful processors hard to context switch