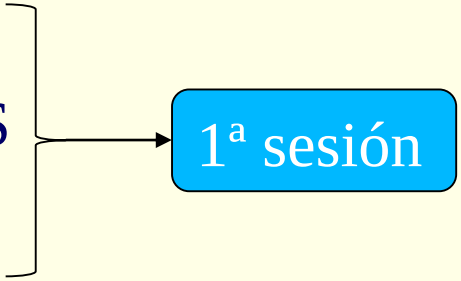
S. empotrados y ubicuos

Programación de dispositivos
(1ª sesión)

Fernando Pérez Costoya
fperez@fi.upm.es

Contenido

- **Introducción**
- Repaso de aspectos básicos del sistema de E/S
- El hardware de E/S visto desde el software
- Aspectos generales de la programación de dispositivos
- Programación de manejadores de dispositivos
  - Caso práctico: programación de manejadores en Linux

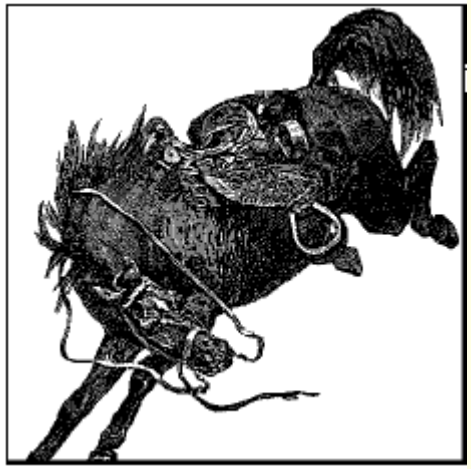


Introducción

- Computador incluye dispositivos de E/S para:
  - Almacenamiento de información
  - Interacción con mundo exterior
    - Usuarios, componentes físicos (sensores, actuadores,…)
  - Comunicación con otros equipos
- Software de E/S muy complejo y heterogéneo
- Precisamente por eso surgieron los SS.OO.
  - Ofrecen interfaz uniforme para todos los dispositivos
  - Manejadores (*drivers*) ocultan complejidad y heterogeneidad
- Programación de manejadores infrecuente en sist. propósito gral.

Introducción

- Sistema empotrado controla un sistema externo
  - Gran interacción con componentes físicos
  - Con frecuencia mediante hardware *ad hoc*
  - Necesidad de programar este hardware
- Menos habitual desarrollo de manejadores de E/S para:
  - Almacenamiento, interacción con usuarios o comunicación
- Progr. de dispositivos muy compleja
  - Hay que domar a “la bestia”



***Linux Device Drivers*, 3ª Edición. O'Reilly, 2005**

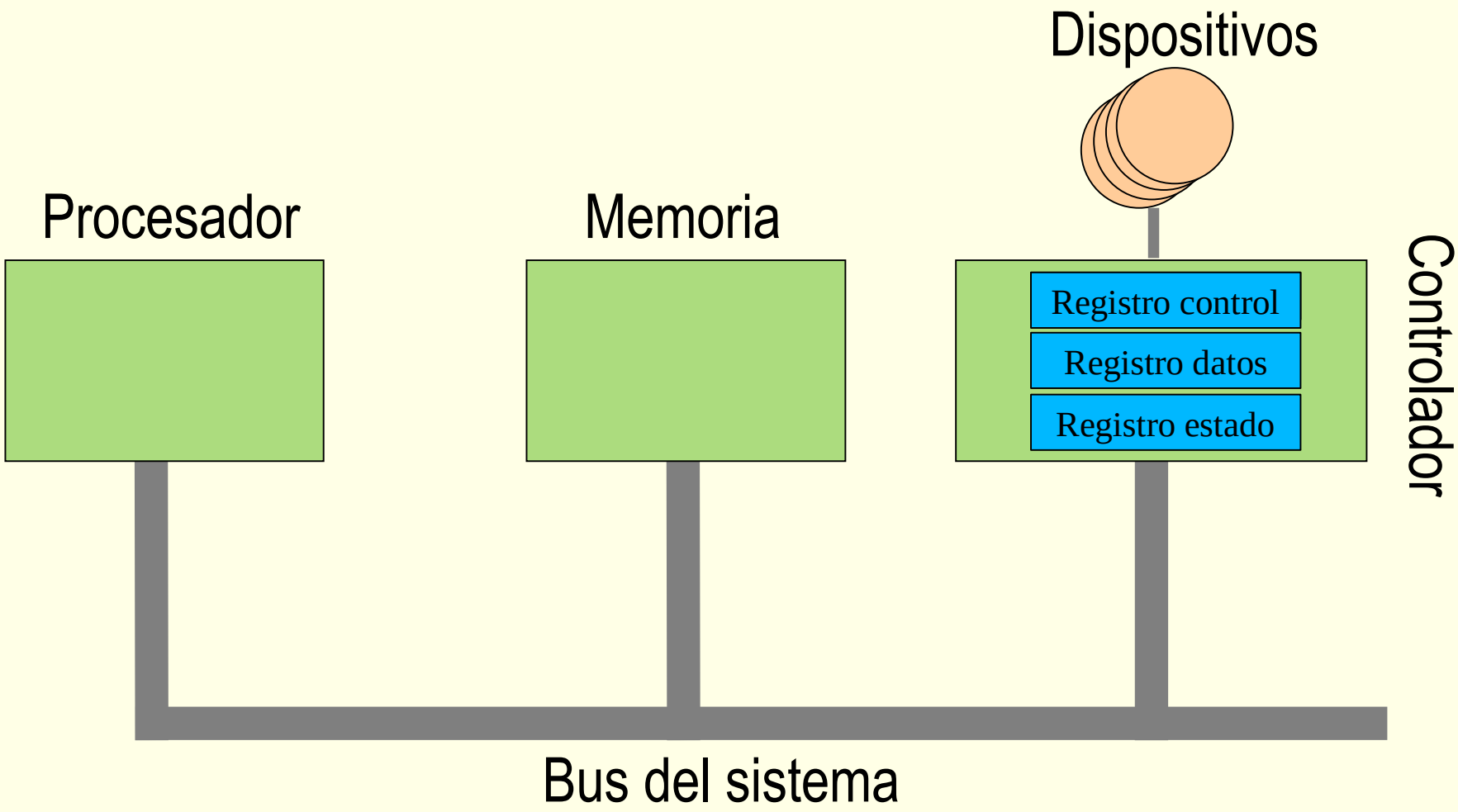
Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman

https://lwn.net/Kernel/LDD3/ http://www.makelinux.net/ldd3

Contenido

- Introducción
- **Repaso de aspectos básicos del sistema de E/S**
- El hardware de E/S visto desde el software
- Aspectos generales de la programación de dispositivos
- Programación de manejadores de dispositivos
  - Caso práctico: programación de manejadores en Linux

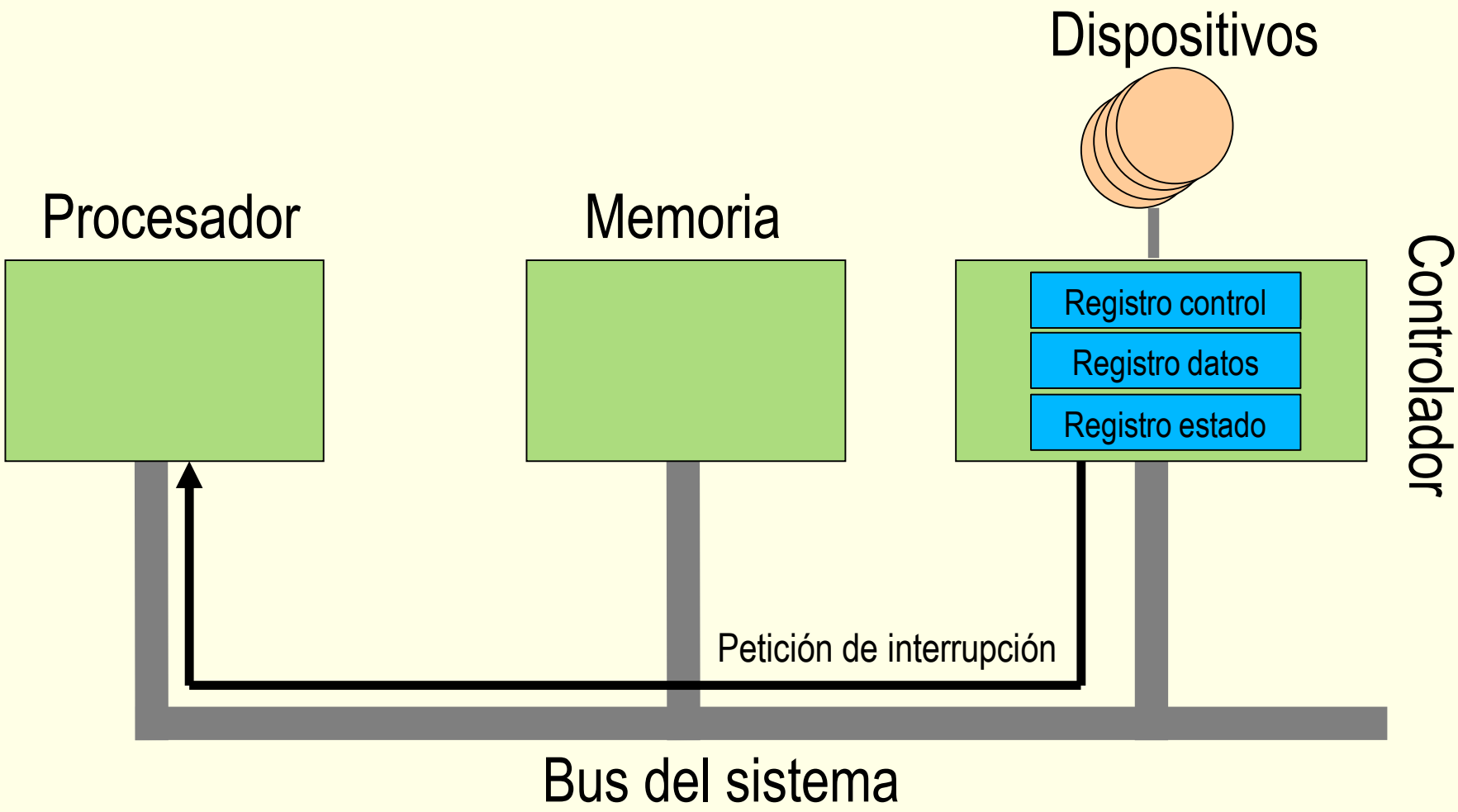
Modelo simplificado de dispositivo básico



E/S programada

- Esquema básico de operación lectura de N datos. Repetir N veces:
  1. Escritura en **reg. control** de código de operación de lectura
  2. Lectura repetida de **reg. estado** hasta fin de operación o error
  3. Si fin de operación → lectura de **reg. datos** del dato leído
- HW sencillo pero UCP monopolizada por la operación
- Aceptable sólo en sistema dedicado
- En otros sistemas, UCP debe ocuparse también de otras labores
- Posible opción: lectura periódica de r. estado. Repetir N veces:
  1. Escritura en **reg. control** de código de operación de lectura
  2. Arranca temporizador y pasa a otras labores
  3. Se cumple temporizador: Lectura de **reg. estado**
  4. Si fin operación → lectura **r. datos** del dato leído y vuelve a 1
  5. Si no → arranca temporizador y pasa a otras labores
- Mejor opción: uso de interrupciones

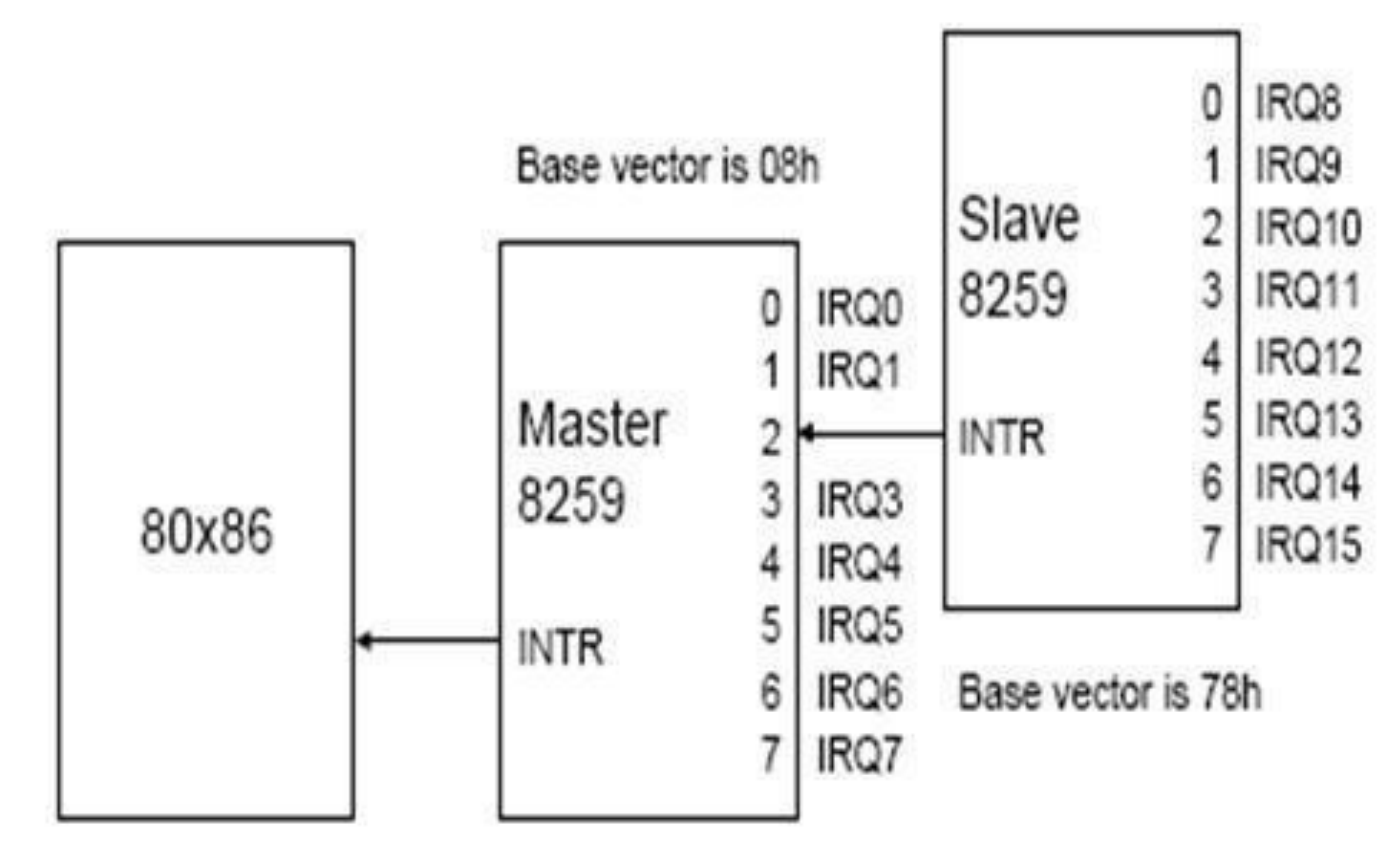
Modelo simplificado de dispo. con interrupciones



E/S con interrupciones

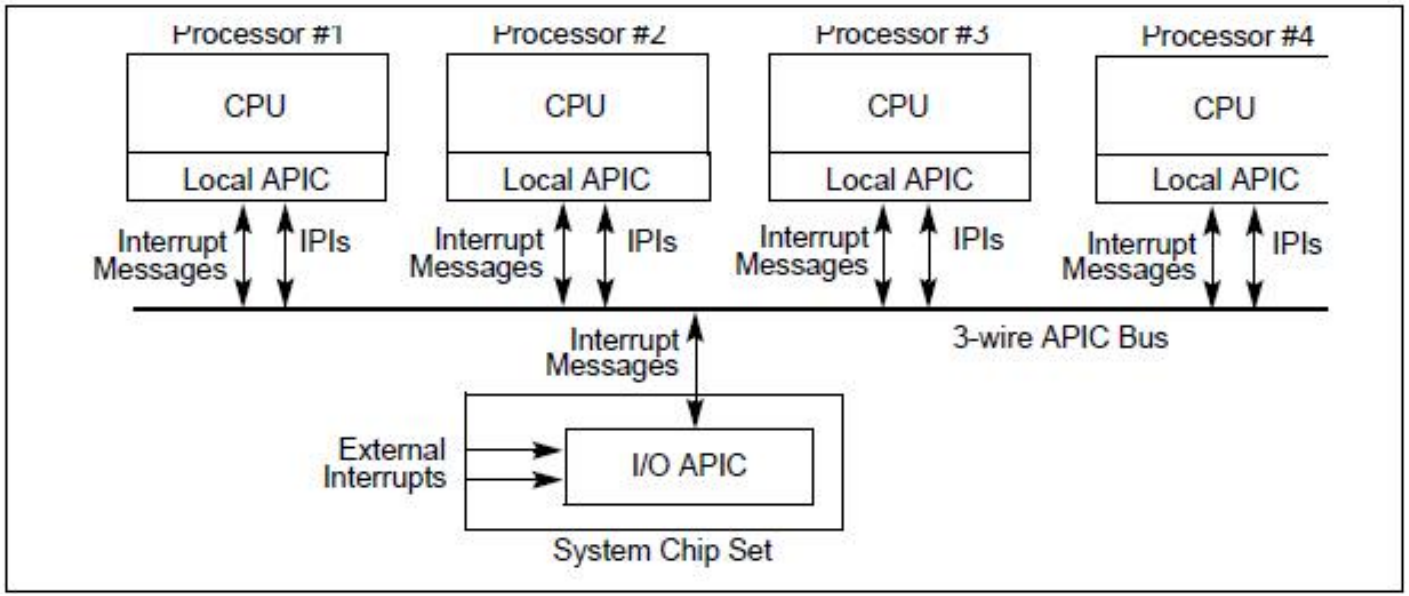
- Controlador de dispo. genera interrupción al completar operación
  - Proporciona un ID (vector) cuando UCP reconoce interrupción
  - Se activa rutina de interrupción correspondiente a vector
- HW más complejo pero solución más eficiente
  - UCP involucrada sólo cuando es necesario
- Esquema básico de operación lectura de N datos. Repetir N veces:
  1. Escritura en **reg. control** de código de operación de lectura
  2. Pasa a otras labores
  3. Se produce interrupción: Lectura de **reg. estado**
  4. Si no error $\rightarrow$ lectura **reg. datos** del dato leído
  5. Vuelve a 1
- UCP involucrada en obtener cada dato
  - Problema si muchos datos y/o dispositivo de alta velocidad
- Mejor opción: uso de DMA

Esquema de interrupciones “tradicional” (i8259)



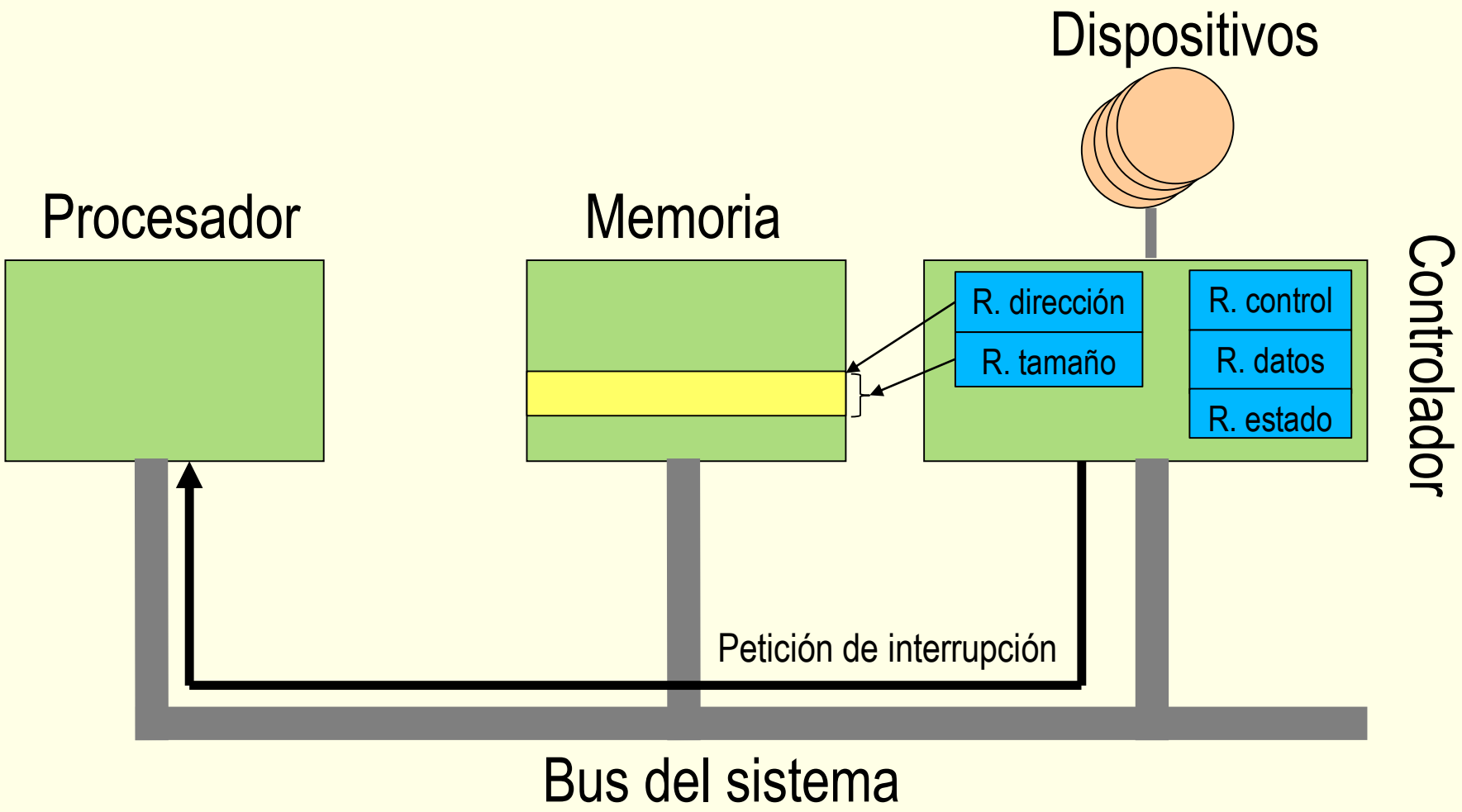
https://masherz.wordpress.com/2010/08/15/pic-8259/

Esquema de interrupciones “moderno” (APIC)



http://zmengchi.is-programmer.com/categories/10630/posts

Modelo simplificado de dispositivo con DMA



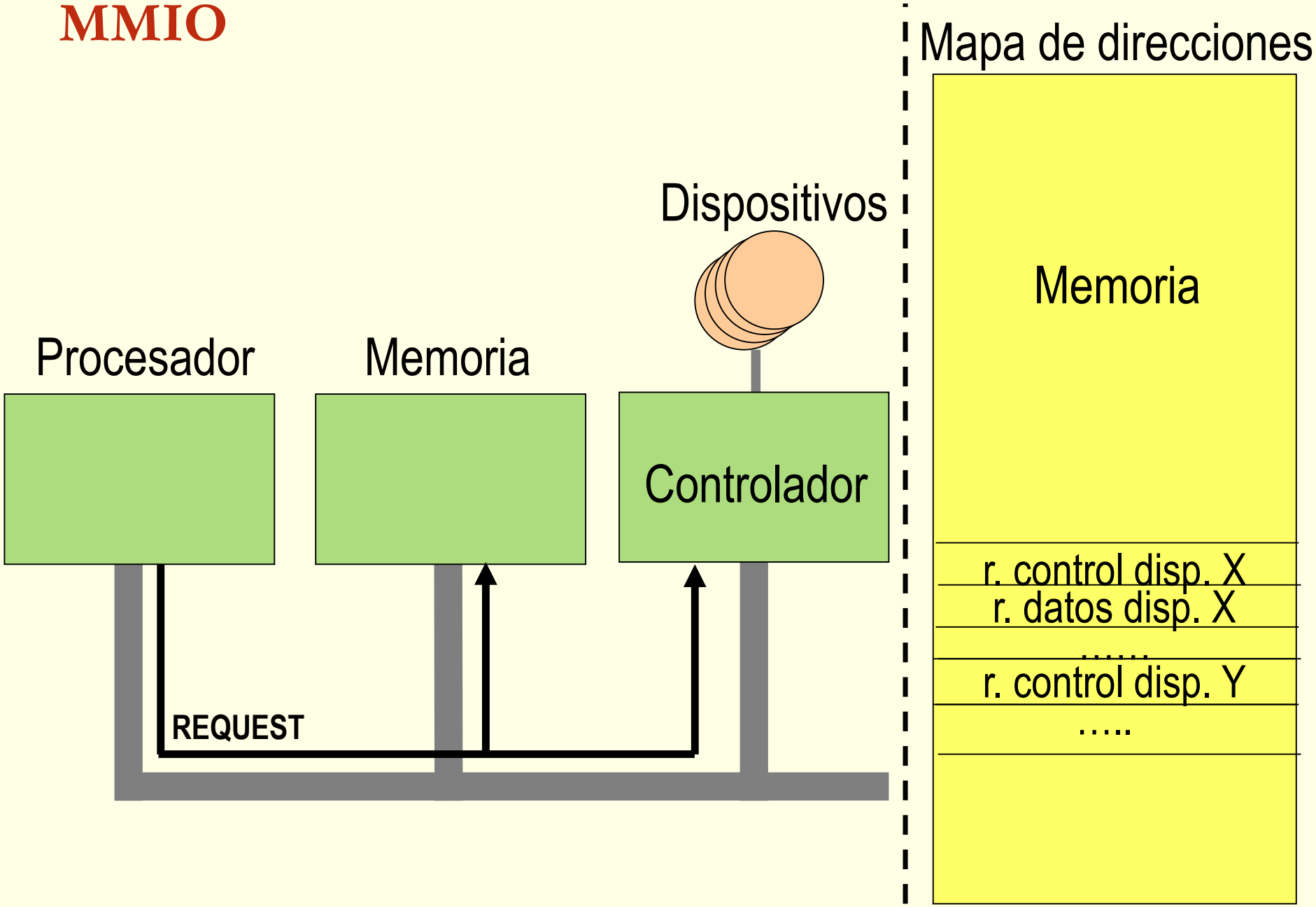
E/S con DMA

- Controlador puede copiar datos de dispo. a memoria y viceversa
  - Sin intervención de la UCP
  - Dispositivo envía interrupción al completar **toda** la operación
- HW más complejo pero solución más eficiente
  - UCP sólo involucrada al inicio para programar OP de DMA
  - y al final para tratar la interrupción
- Esquema básico de operación lectura de N datos. Sólo una vez:
  1. Escritura en **reg. tamaño** del valor N
  2. Escritura en **r. dirección** de dir. de mem. donde dejar los datos
  3. Escritura en **reg. control** de código de operación de lectura
  4. Pasa a otras labores
  5. Se produce interrupción: Operación total completada
     - Lectura de **reg. estado** para comprobar si ha habido un error

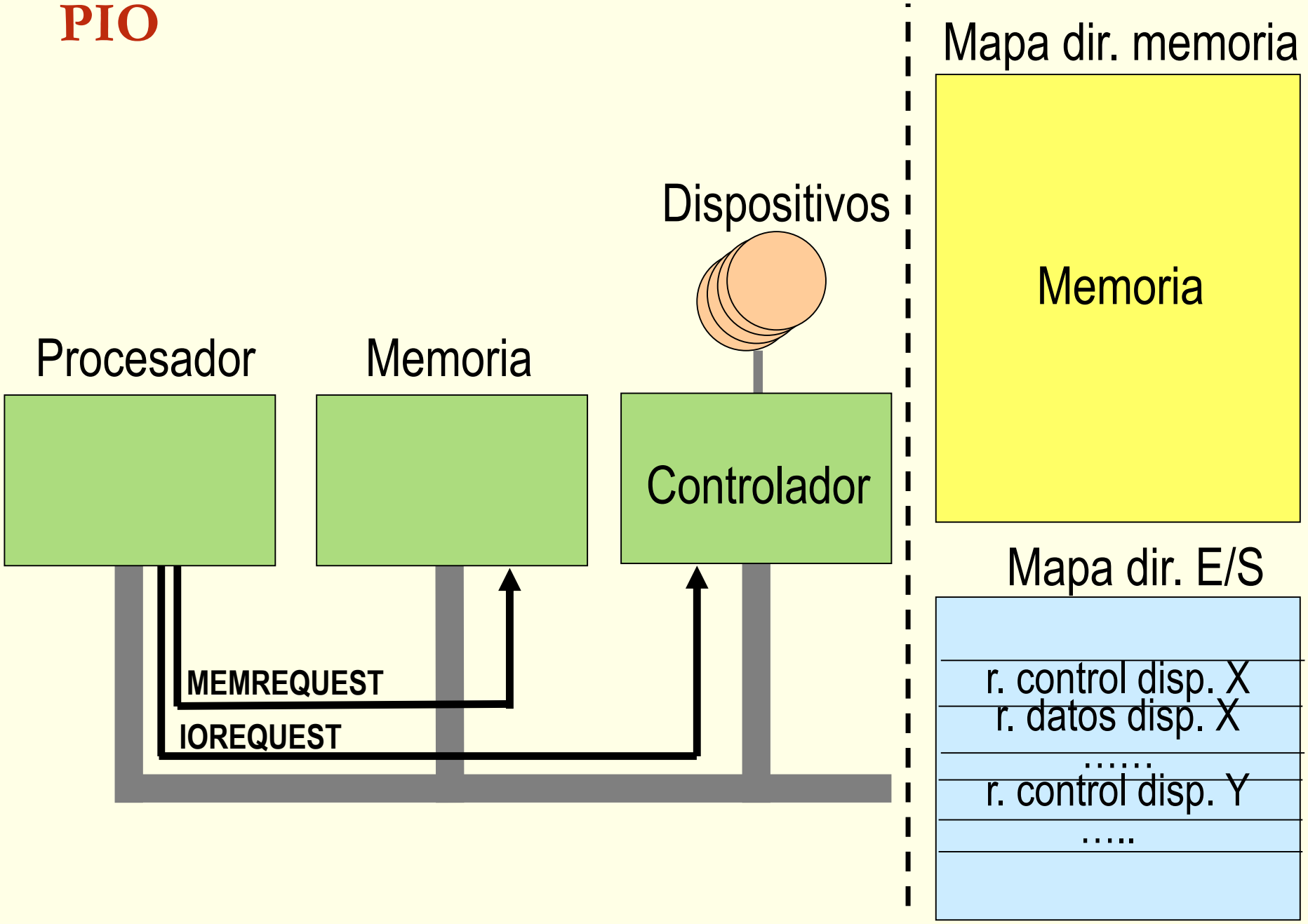
Memory-mapped I/O (MMIO) vs. *Port I/O* (PIO)

- MMIO: Mismo espacio de dir. e instrucciones para mem. y E/S
  - Instrucciones acceso a memoria convencionales: LOAD/STORE
- PIO: Distintos espacios de dir. e instrucciones para mem. y E/S
  - Bus incluye señal para discriminarlos (MEMREQ vs IOREQ)
  - Instrucciones de acceso específicas: IN/OUT
- Ventajas de MMIO en la programación de dispositivos
  - Procesador más sencillo
  - Puede usar cualquier tipo de direccionamiento en acceso a dispo
  - No requiere ensamblador
- Ventajas de PIO en la programación de dispositivos
  - Menos problemas de coherencia
- PIO no habitual excepto familia x86
  - Aunque también usa MMIO
  - Linux: ficheros /proc/ioports y /proc/iomem

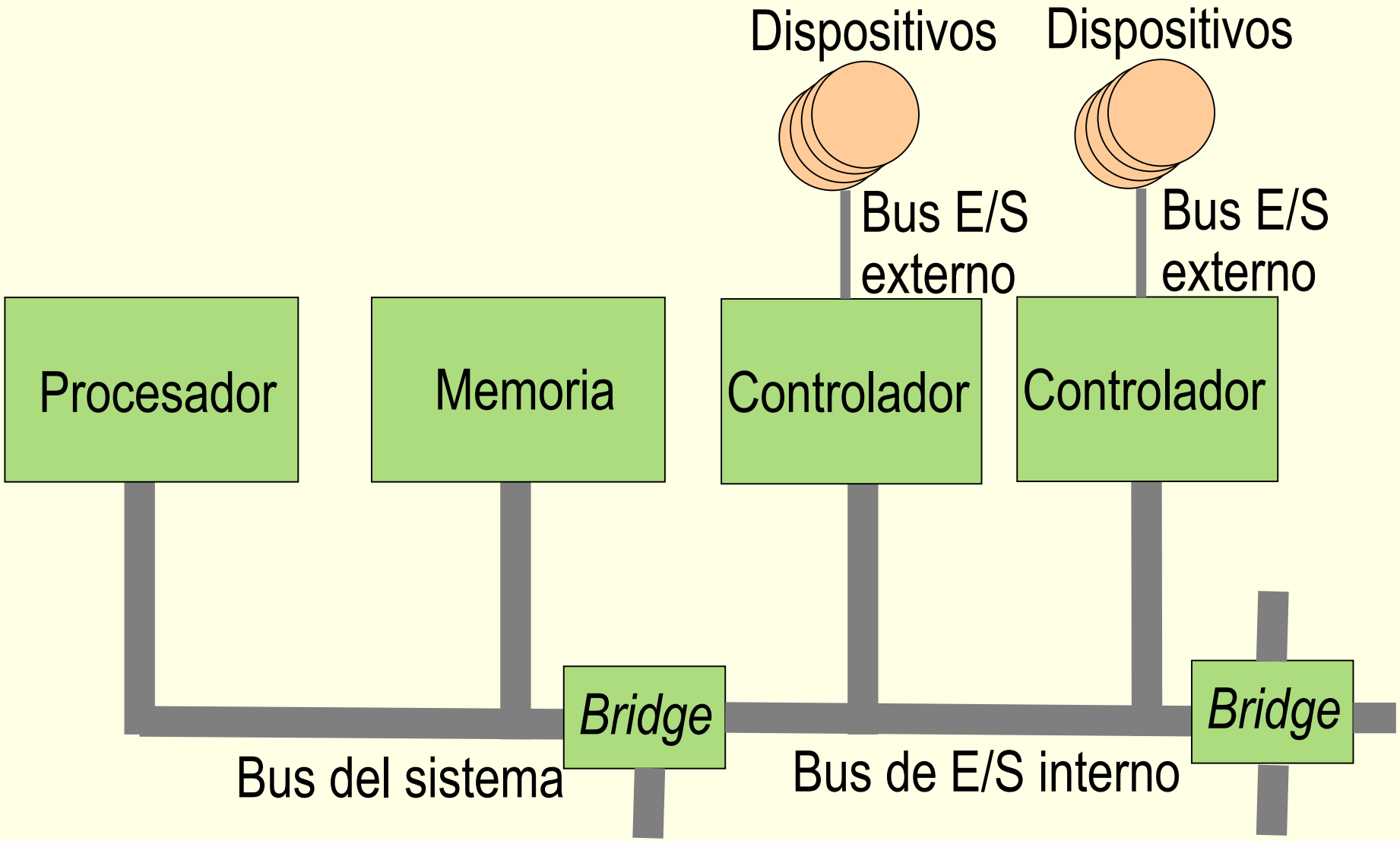
MMIO


Mapa de direcciones
Memoria
Dispositivos
Procesador
Memoria
Controlador
r. control disp. X
r. datos disp. X
……
r. control disp. Y
…..
REQUEST

PIO


Mapa dir. memoria
Memoria
Dispositivos
Procesador
Memoria
Controlador
MEMREQUEST
IOREQUEST
Mapa dir. E/S
r. control disp. X
r. datos disp. X
……
r. control disp. Y
…..

Topología simplificada de buses



Contenido

- Introducción
- Repaso de aspectos básicos del sistema de E/S
- **El hardware de E/S visto desde el software**
- Aspectos generales de la programación de dispositivos
- Programación de manejadores de dispositivos
  - Caso práctico: programación de manejadores en Linux

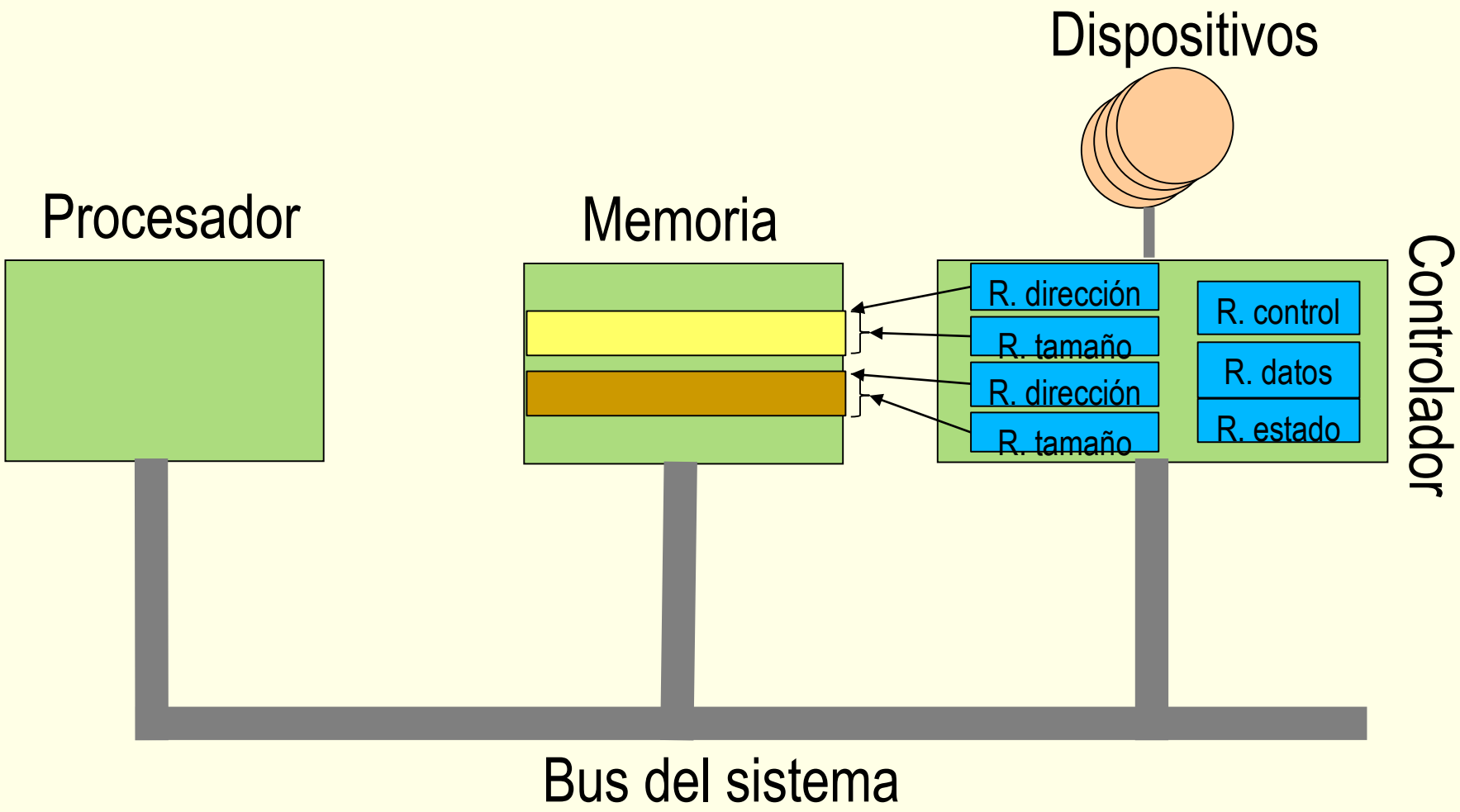
Interrupciones

- Distintas alternativas de diseño hardware visibles desde software
  - Por flanco (*edge-triggered*) o por nivel (*level-triggered*)
  - Basadas en líneas de interrupción vs *message-signaled* (MSI)
    - MSI (*in band*): interrupción→escritura de valor en cierta dirección
  - Interrupciones compartidas: Múltiples dispositivos/línea
    - Dificultad para compartir interrupciones
      - Por flanco: se pueden mezclar o perder pulsos
    - ¿Cómo identificar dispositivo que interrumpe?
      - Por SW: Comprobar estado de todos los dispositivos de la línea
      - Por HW: *Daisy-chain*
  - Interrupciones en multiprocesador
    - Modalidad de distribución de interrupciones entre procesadores
      - Fija; *Broadcast*; *multicast*;
      - Turno rotatorio; por prioridad; a UCP más reciente; ...
  - Linux: fichero /proc/interrupts

DMA

- Distintas alternativas de diseño visibles desde el software
  - Controlador con o sin *scatter-gather DMA (vectored I/O)*
    - Controlador con múltiples pares (reg. dirección; reg. tamaño)
  - Controlador con o sin coherencia entre caché y memoria
    - *Non-coherent DMA*: transferencias DMA no afectan a la caché
  - Uso de IO-MMU (*aka* virtual DMA)
    - Controlador ve memoria con direcciones ≠ UCP (p.e. SPARC)
    - Posibilita ver como contiguo buffer no contiguo en m. física.
    - Facilita virtualización de la E/S
  - Limitaciones en rango acceso a memoria desde dispositivos
    - ISA sólo primeros 16MB
    - PAE y dispositivos con DMA de 32 bits: no más allá de 4GB
  - /proc/dma: canales DMA de bus ISA

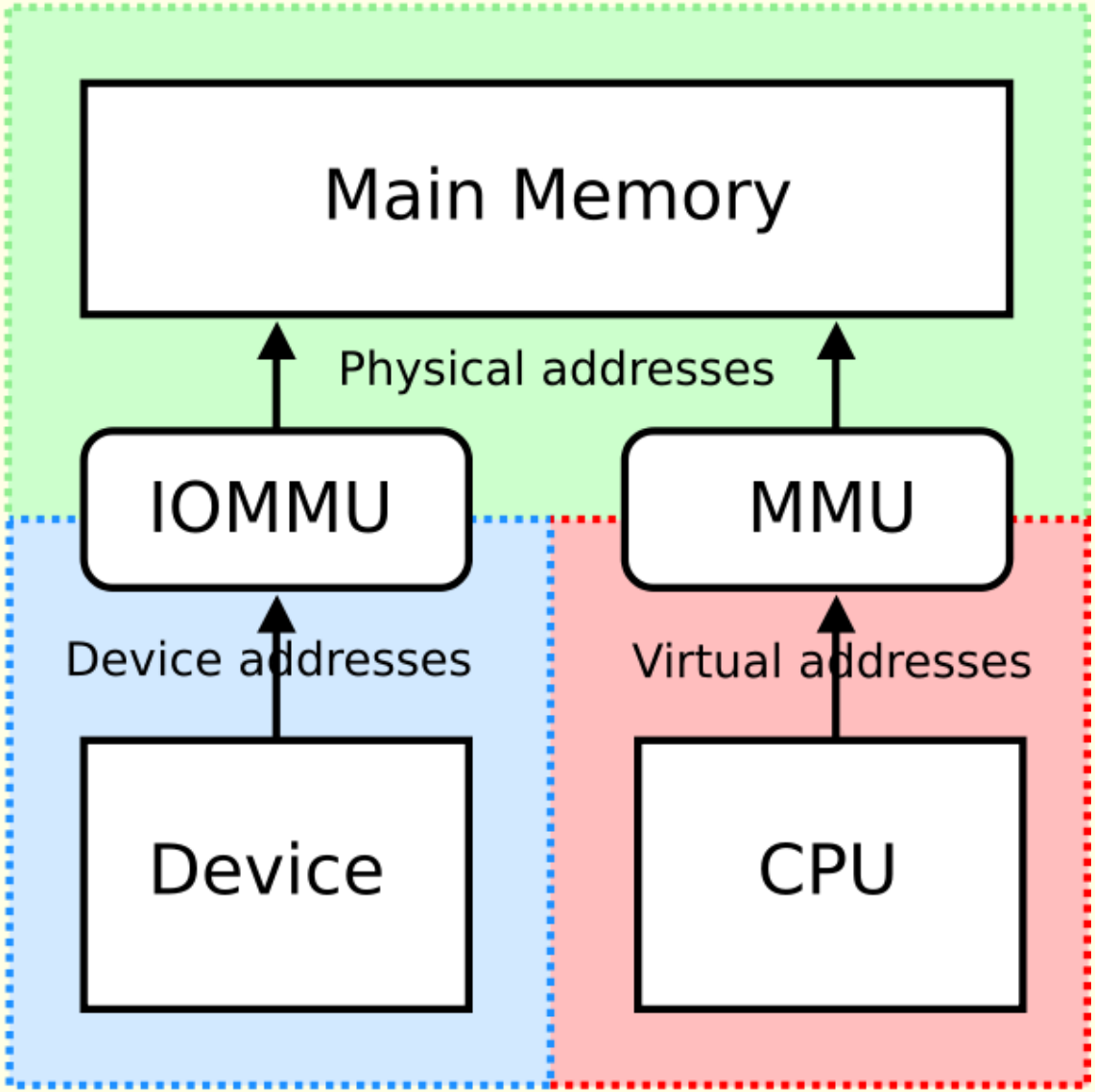
Controlador con *scatter-gather* DMA



Non-coherent DMA (wikipedia)

Operación de lectura

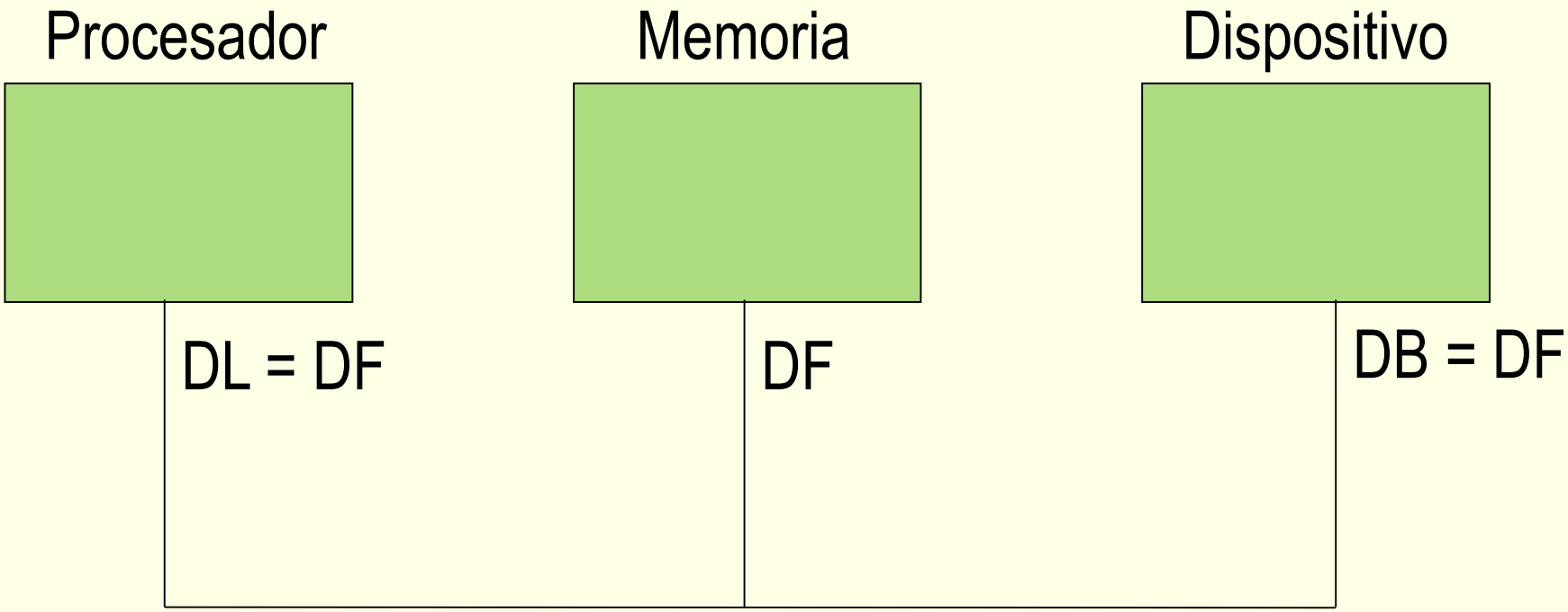
IO-MMU (wikipedia)



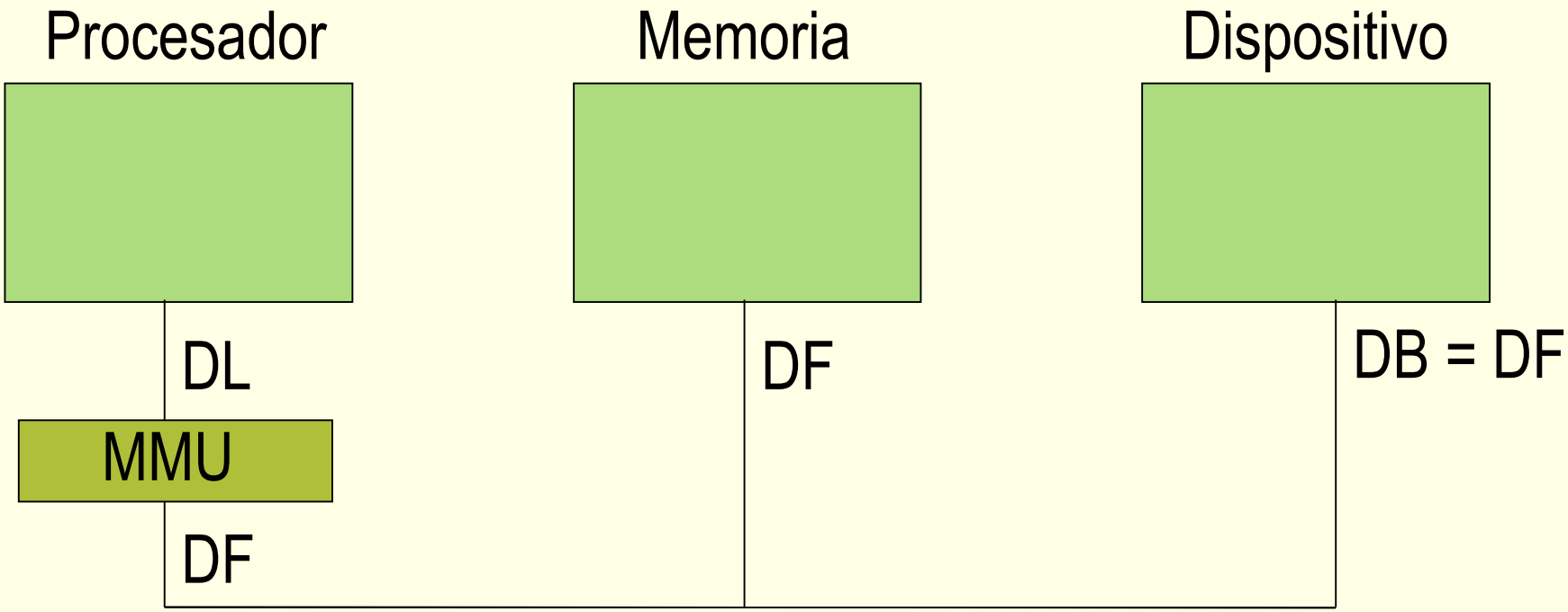
Tipos de direcciones

- Tres tipos:
  - Lógicas (DL) (o virtual): usadas por procesador
    - Si UCP distingue modos de ejecución: DL usuario o DL sistema
  - Físicas (DF): las que llegan a la memoria
  - De bus (DB): usadas por un dispositivo
- En sistema sin MMU ni IO-MMU
  - DL = DF = DB
- En sistema con MMU pero sin IO-MMU
  - DL ≠ DF = DB
- En sistema con MMU e IO-MMU
  - DL ≠ DF ≠ DB
- ***http://www.makelinux.net/ldd3/chp-15-sect-1***

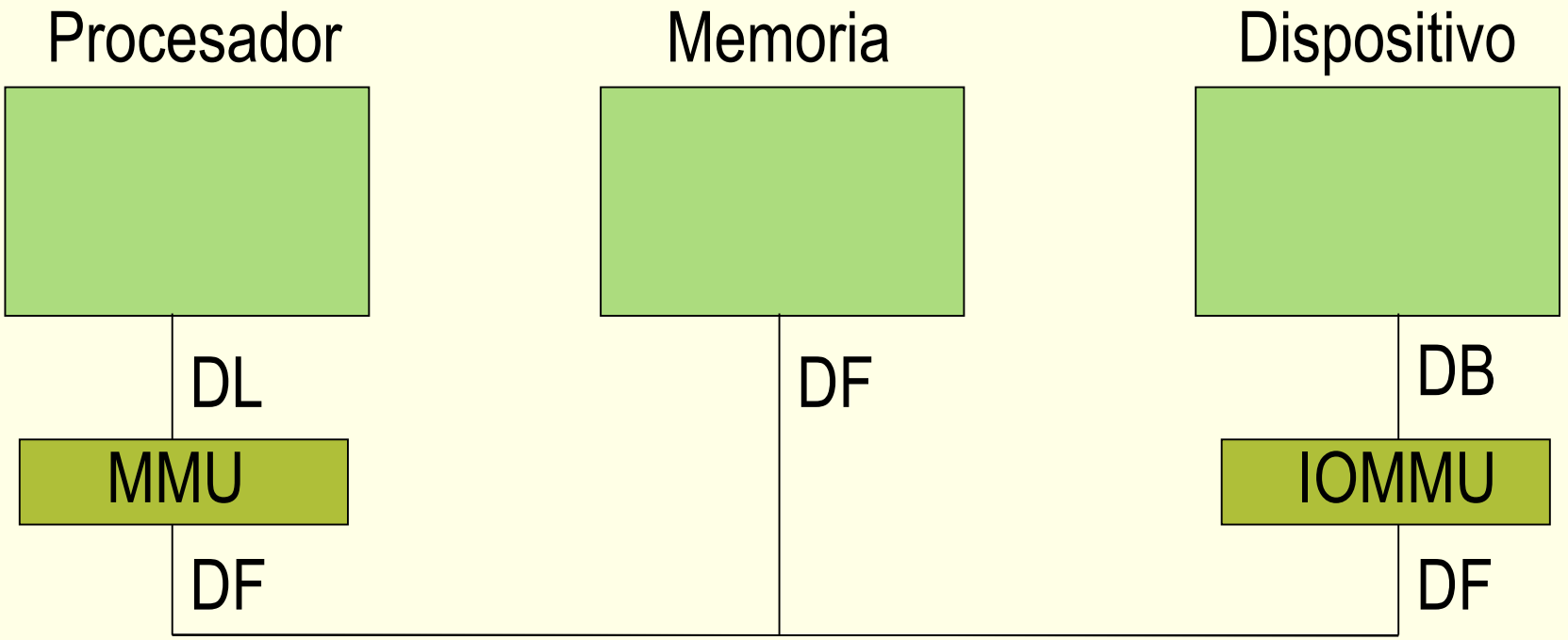
Sistema sin MMU ni IO-MMU



Sistema con MMU pero sin IO-MMU



Sistema con MMU e IO-MMU



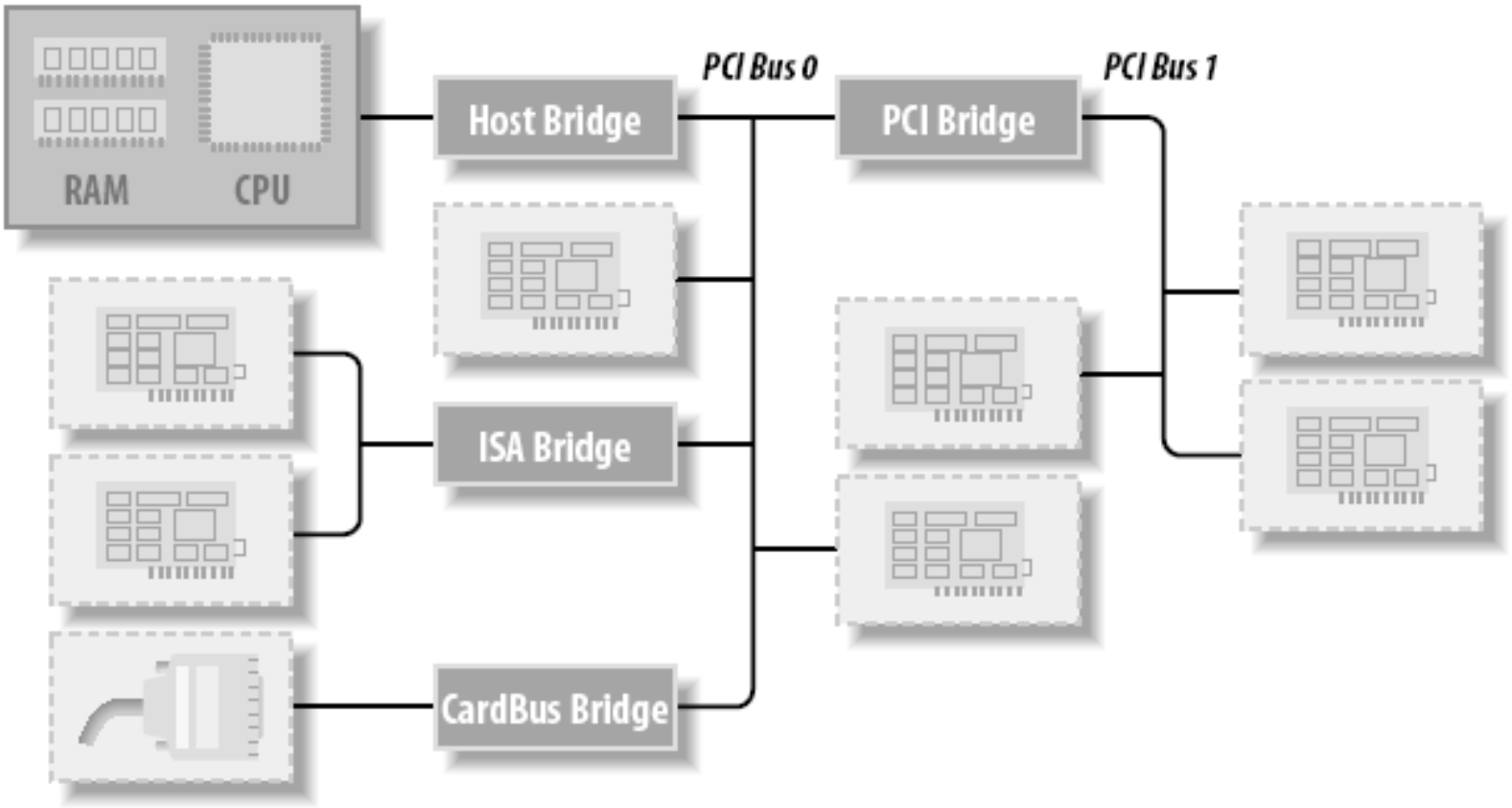
Configuración de dispositivos

- Cada dispositivo usa diversos recursos:
  - Rango dir E/S (puertos y/o memoria) + línea(s) de interrup.
- Asignación de recursos estática ("de fábrica") → colisiones
- Asignación mediante *jumpers* → poco flexible
- Deben ser configurables por software
- ¿Cómo configurar dispositivo si no tiene dirección asignada?
  - Uso direccionamiento geográfico → posición (*slot*) en el bus
- Deseable *plug* & *play* (*P&P*) y *hot-plugging* (*H-P*)
    - *P&P*: Configuración automática de dispositivos en arranque
    - *H-P*: Conexión de dispositivo con el sistema arrancado
  - Requiere reconocer de qué tipo de dispositivo se trata
    - Para configurar y cargar su manejador en tiempo de ejecución
    - Dispo. ofrece información en sus registros de configuración
      - ID único, vendedor, clase, nº dir. E/S e interrupciones requeridas,…
      - Incluso en algunos, niveles de consumo de energía disponibles

Jerarquía de buses de E/S: buses internos

- Variedad de dispositivos de muy diversas características
  - Conveniencia de jerarquía de buses
- Buses de E/S internos (PCI, ISA, ...)
  - Dispositivos directamente accesibles mediante PIO o MMIO
  - Jerarquía por limitaciones, rendimiento, compatibilidad, ...
    - Puentes (*Bridges*) permiten su interconexión
  - Proceso de enumeración de dispositivos (si el bus lo permite)
    - "Descubrimiento" mediante direccionamiento geográfico
      - Accede sucesivamente a cada posición del bus
      - atravesando los *Bridges* encontrados
    - Configuración de dirs. E/S (PIO o MMIO) e IRQs
    - Obtención de info. de dispositivo (vendedor, ID, clase, ...)

Jerarquía de buses internos



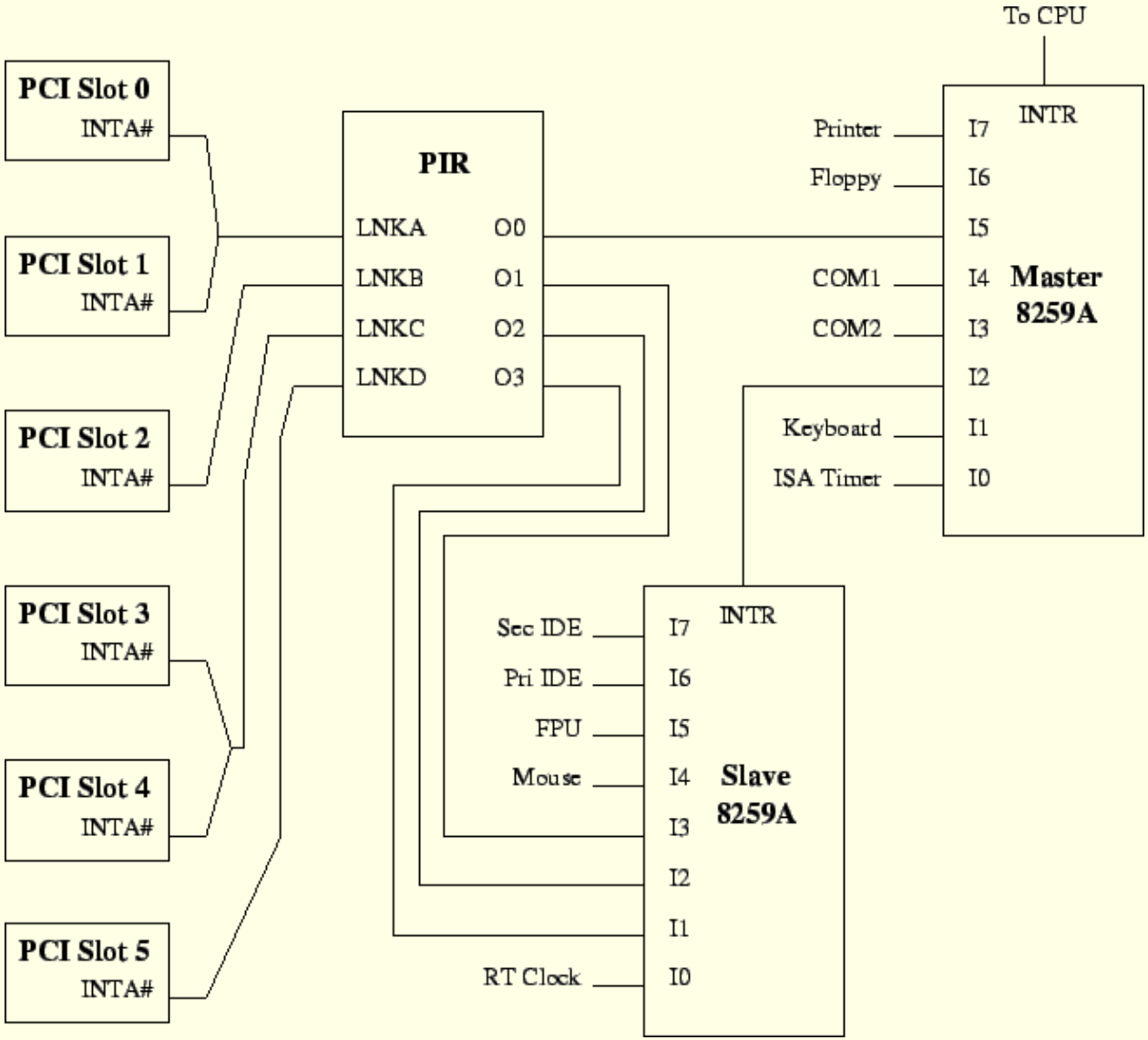
***Linux Device Drivers*, 3ª Edición**

Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman

Bus PCI

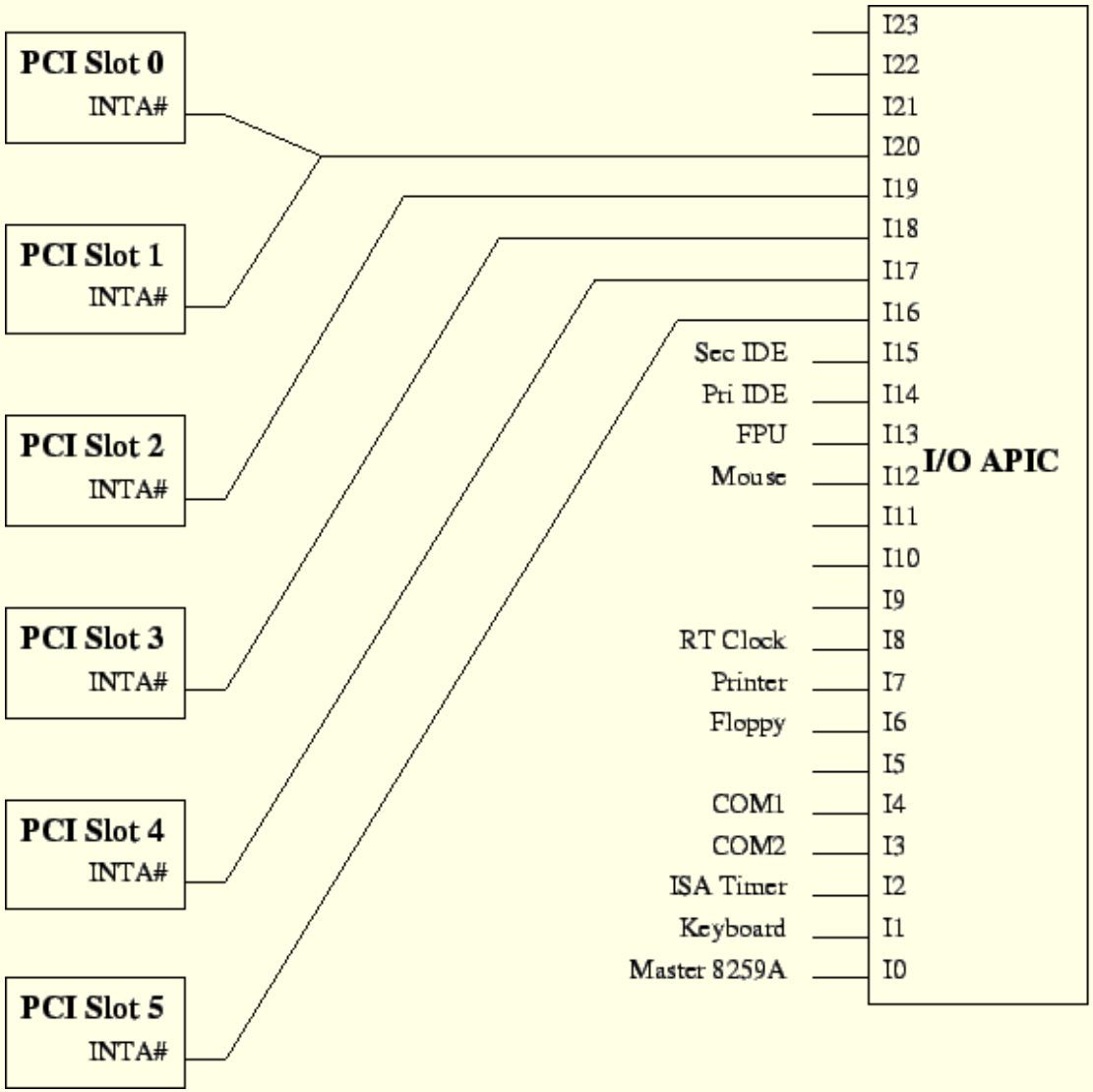
- Sustituto de ISA (desde *desktops* a grandes servidores)
  - Mucho más rápido, permite autoconfiguración y es "neutral"
  - PCI convencional, PCI-X, PCI Express (bus serie)
- Cada *slot* del bus: "dispositivo" con múltiples "funciones"
  - Realmente, cada función es un dispositivo
- Bus jerárquico mediante *PCI-PCI Bridges* (*PPB*) (Linux `lspci -tv`)
  - 256 buses (8 bits), 32 *slots*/bus (5 bits), 8 funciones/*slot* (3 bits)
  - CPU dialoga con *Host Bridge* (*HB*)
    - Puede haber varios *HBs*: terminología Linux, múltiples dominios
- Cada *slot* tiene 4 *pines* de interrupción (A-D)
  - Cada función puede usar uno (PCI usa interrup. compartidas)
  - Conectados de forma entrelazada (*pin* A *slot* 0→*pin* B *slot* 1,..)
  - Ajeno a PCI:conexión *pines* a líneas controlador interrupciones
    - Puede ser fija o programable

Ejemplo de asignación de interrupciones en i8259



https://people.freebsd.org/~jhb/papers/bsdcan/2007/article/node4.html

Ejemplo de asignación de interrupciones en APIC

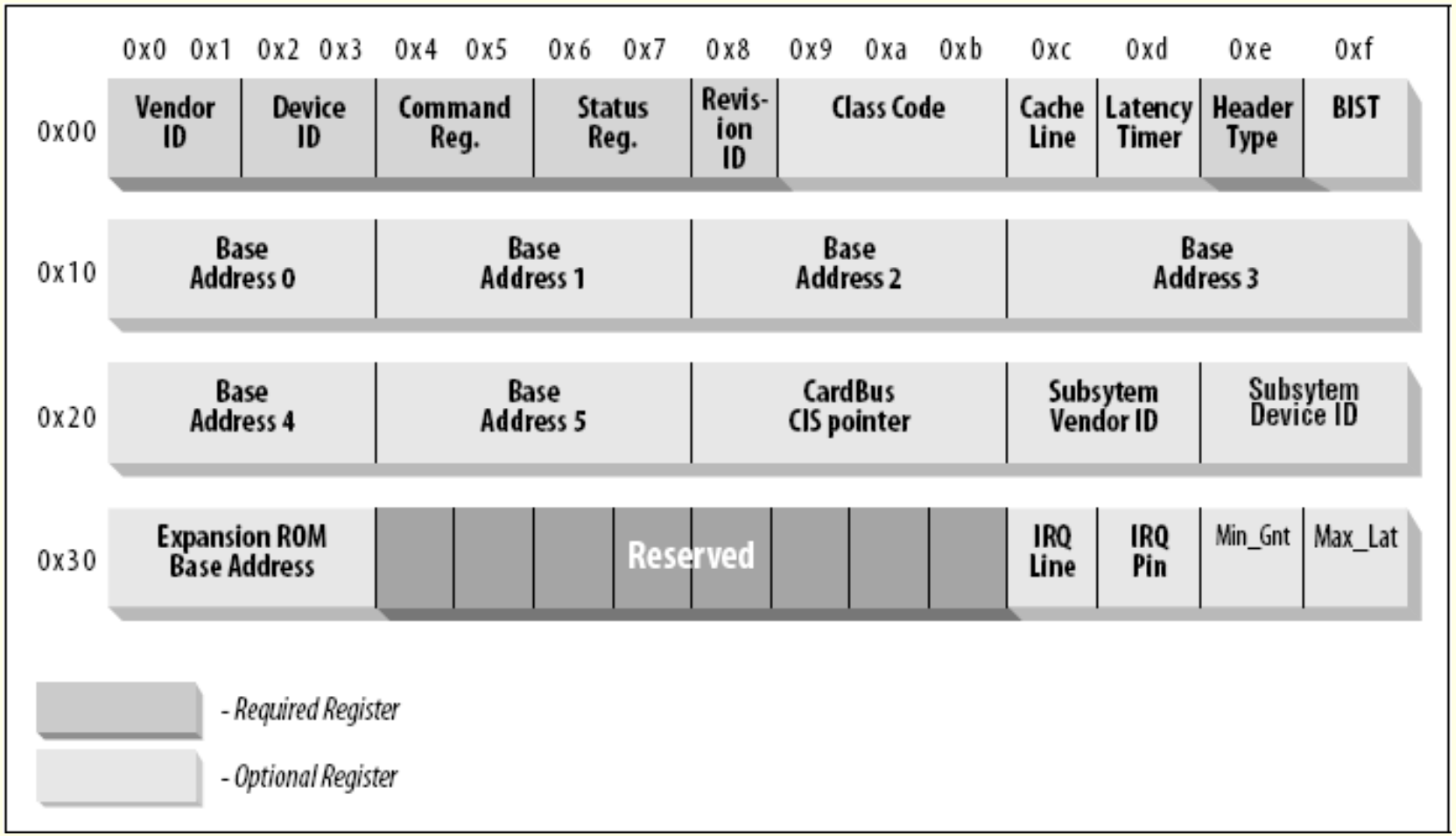


https://people.freebsd.org/~jhb/papers/bsdcan/2007/article/node4.html

Acceso al bus PCI

- Dispositivo PCI provee 2 tipos de accesos:
  - Configuración: acceso RW geográfico por posición *slot* en bus
    - Cada "función" proporciona registros de configuración (256B)
    - Acceso de configuración lee o escribe un registro
  - Una vez configurado: acceso convencional MMIO/PIO
- SW no puede realizar accesos de configuración
  - Solución habitual: *HB* proporciona dos puertos PIO
    - SW especifica dirección de acceso en CONFIG_ADDRESS (0xCF8)
      - nº bus + nº *slot* + nº función + nº r. configuración
    - SW lee/escribe valor r. configuración en CONFIG_DATA (0xCFC)
  - ¿Cómo HW realiza acceso geográfico? Solución habitual:
    - Si dispo. accedido en bus 0, *HB* genera acceso directo al mismo
      - Cada bit del bus direcciones a IDSEL de un *slot* (acceso tipo 0)
    - Si en otro, *HB* propaga dirección a *PPBs* y afectado repite proceso
      - Acceso de tipo 1

Info. de configuración de dispositivo en PCI

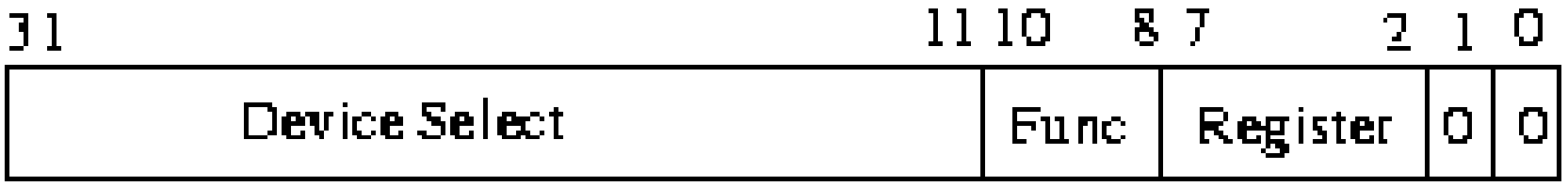


***Linux Device Drivers*, 3ª Edición**

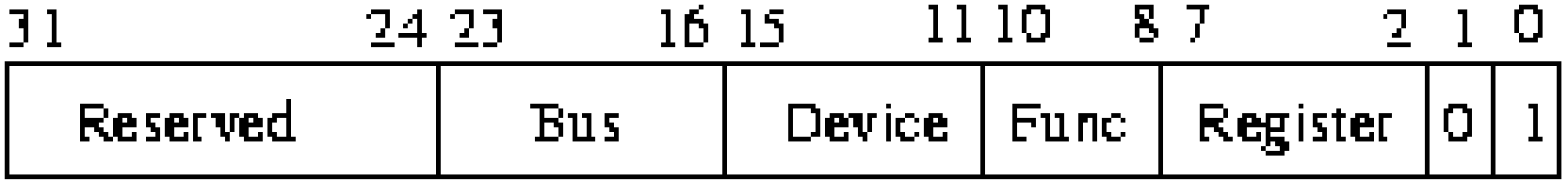
Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman

Tipos de ciclos de configuración

Fuente: TLDP



Tipo 0

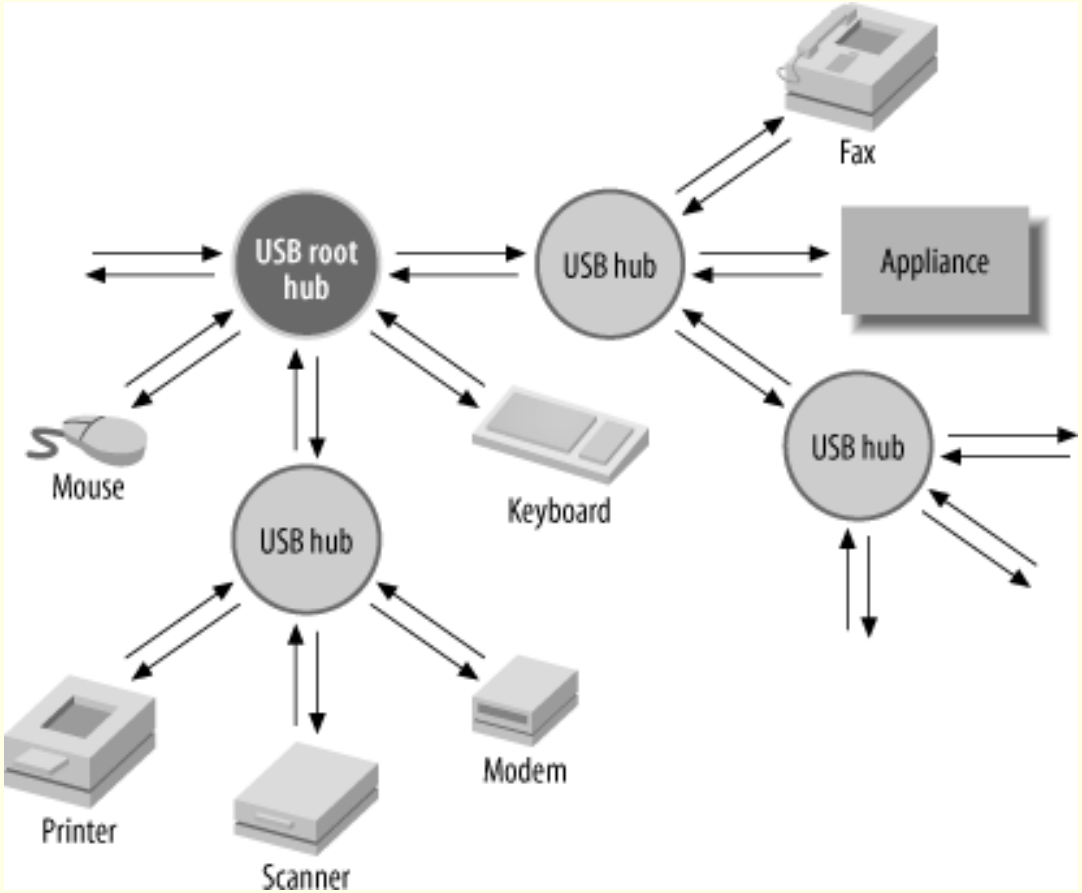


Tipo 1

Buses de E/S externos (SCSI, USB, ...)

- Conectados a internos mediante controlador (*host controller*)
  - Controlador descubierto/config. en enumeración de b. internos
- Dispositivos no directamente accesibles mediante PIO/MMIO
- SW interacciona (PIO/MMIO) con controlador de bus
  - Controlador interacciona con dispositivos conectados al bus
- Enumeración de bus externo (si lo permite, como USB)
  - Descubrimiento de dispositivos
  - Configuración de dispositivo (no de dirs. E/S ni de IRQs)
    - Puede asignar una dirección interna (de 7 bits en USB)
    - Puede obtenerse info. de dispositivo (vendedor, ID, clase, ...)
      - Incluso en algunos niveles de consumo de energía disponibles
  - Linux: mandato `lsusb -tv`

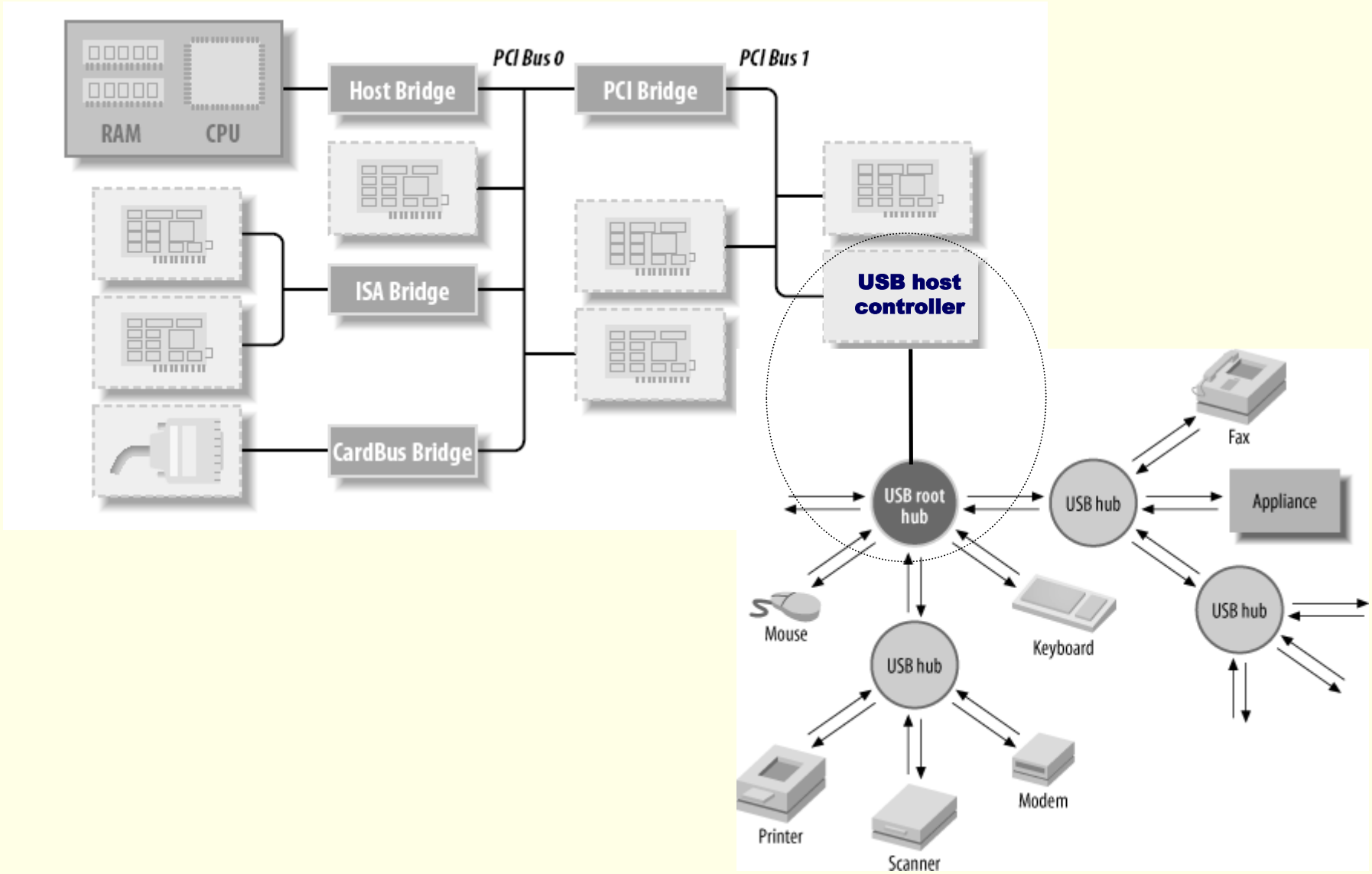
Bus externo USB



Designing Embedded Hardware

John Catsoulis

Jerarquía de buses de E/S



Control de consumo de energía de dispositivos

- Algunos dispositivos permiten distintos niveles consumo energía
- Estándar *Advanced Configuration and Power Interface* (ACPI)
  - Estados del sistema: S0 (*fully on*) a S5 (*fully off*)
  - Estados del dispositivo
    - D0: dispositivo completamente operativo (consumo máximo)
    - D3: dispositivo apagado
    - D1, D2: estados intermedios dependientes del dispositivo
- En "descubrimiento" de dispo. se averiguan niveles disponibles
- Cuando condiciones del sistema lo requieran →
  - Disminución alimentación de energía en el sistema
  - Aspectos operacionales (p.e. cambio de "misión")
  - Solicitud explícita de usuario
- → Se baja nivel de energía de dispositivos que lo permitan
  - De forma ordenada: 1º apagar dispos., después controlador bus