Aufgabe 1 – Entwicklung einer Virtuellen Maschine

Rainer Müller

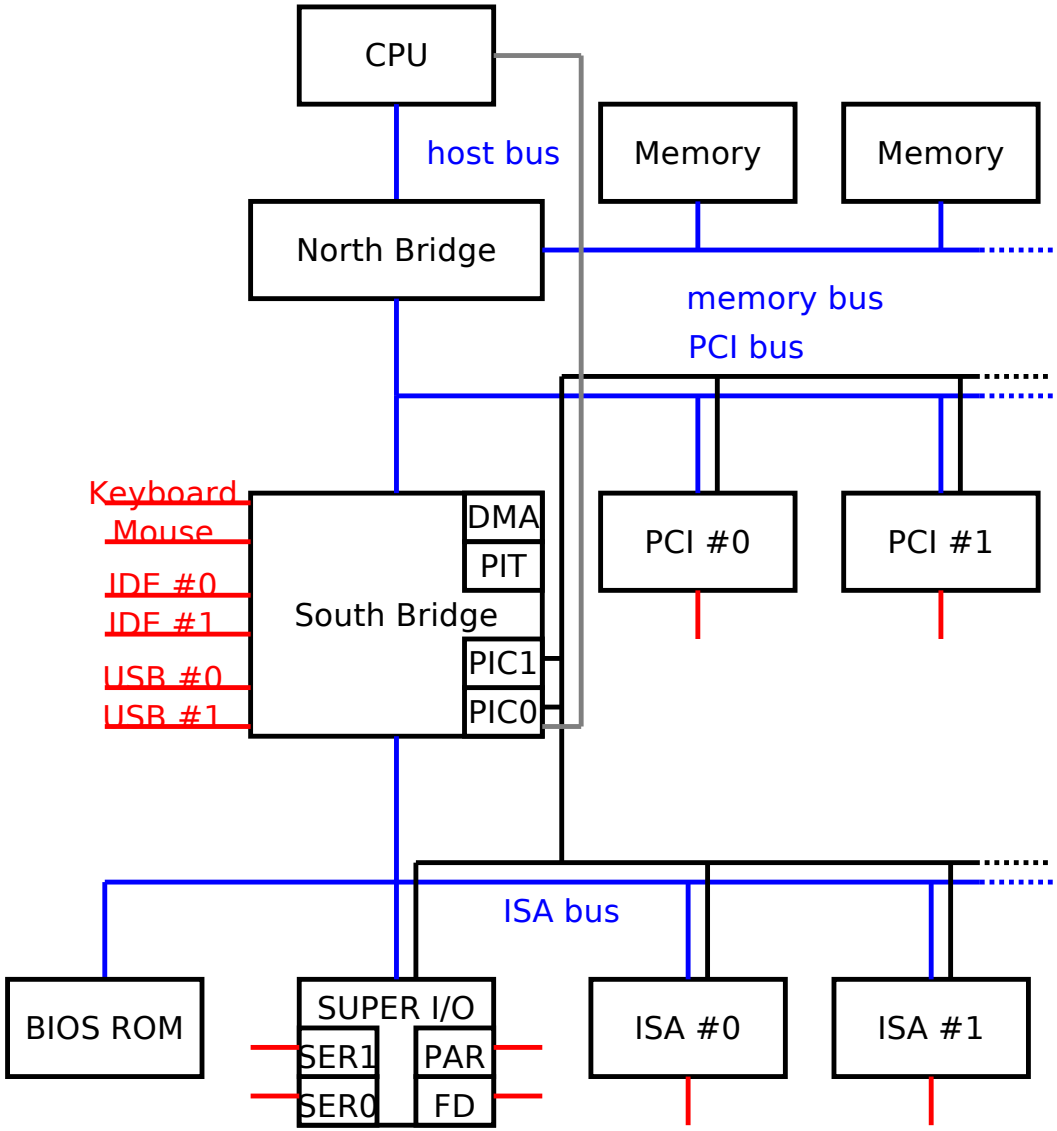
Department Informatik 4
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg

WS 2014/2015

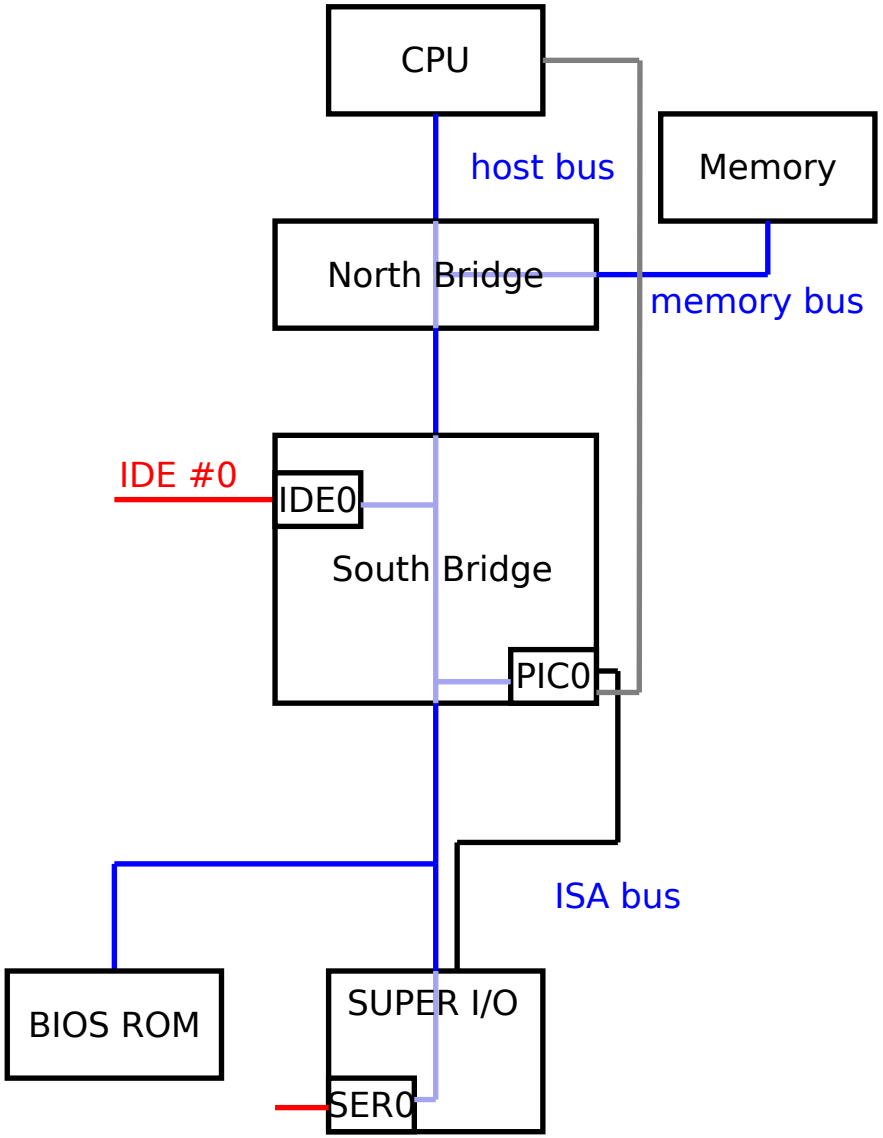
Einführung

Entwickelt werden soll ein virtueller (vereinfachter) PC.

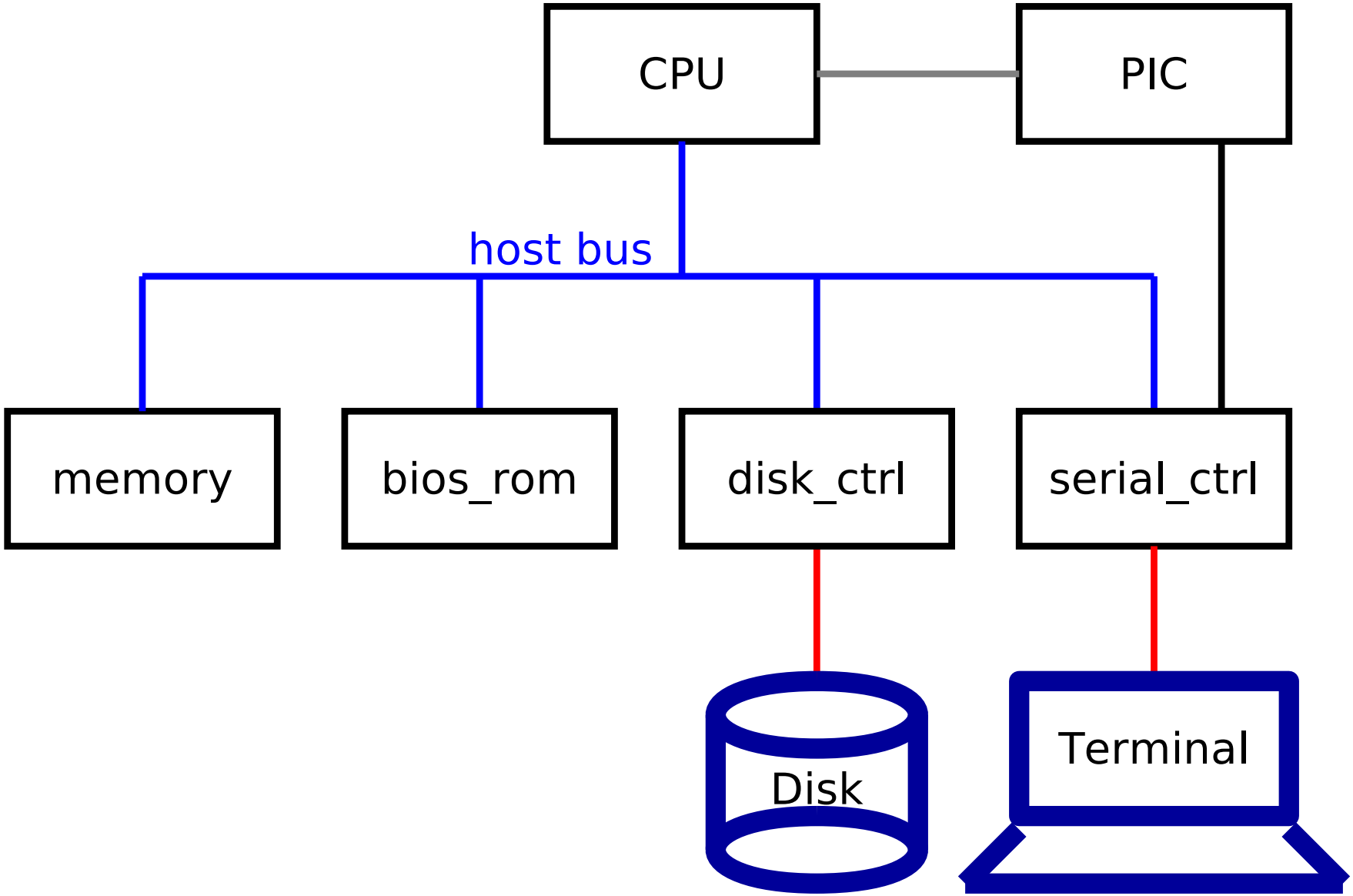
PC – Aufbau


CPU
host bus
Memory
Memory
North Bridge
memory bus
PCI bus
Keyboard
Mouse
IDE #0
IDE #1
USB #0
USB #1
South Bridge
DMA
PIT
PIC1
PIC0
PCI #0
PCI #1
ISA bus
BIOS ROM
SUPER I/O
SER1
SER0
PAR
FD
ISA #0
ISA #1

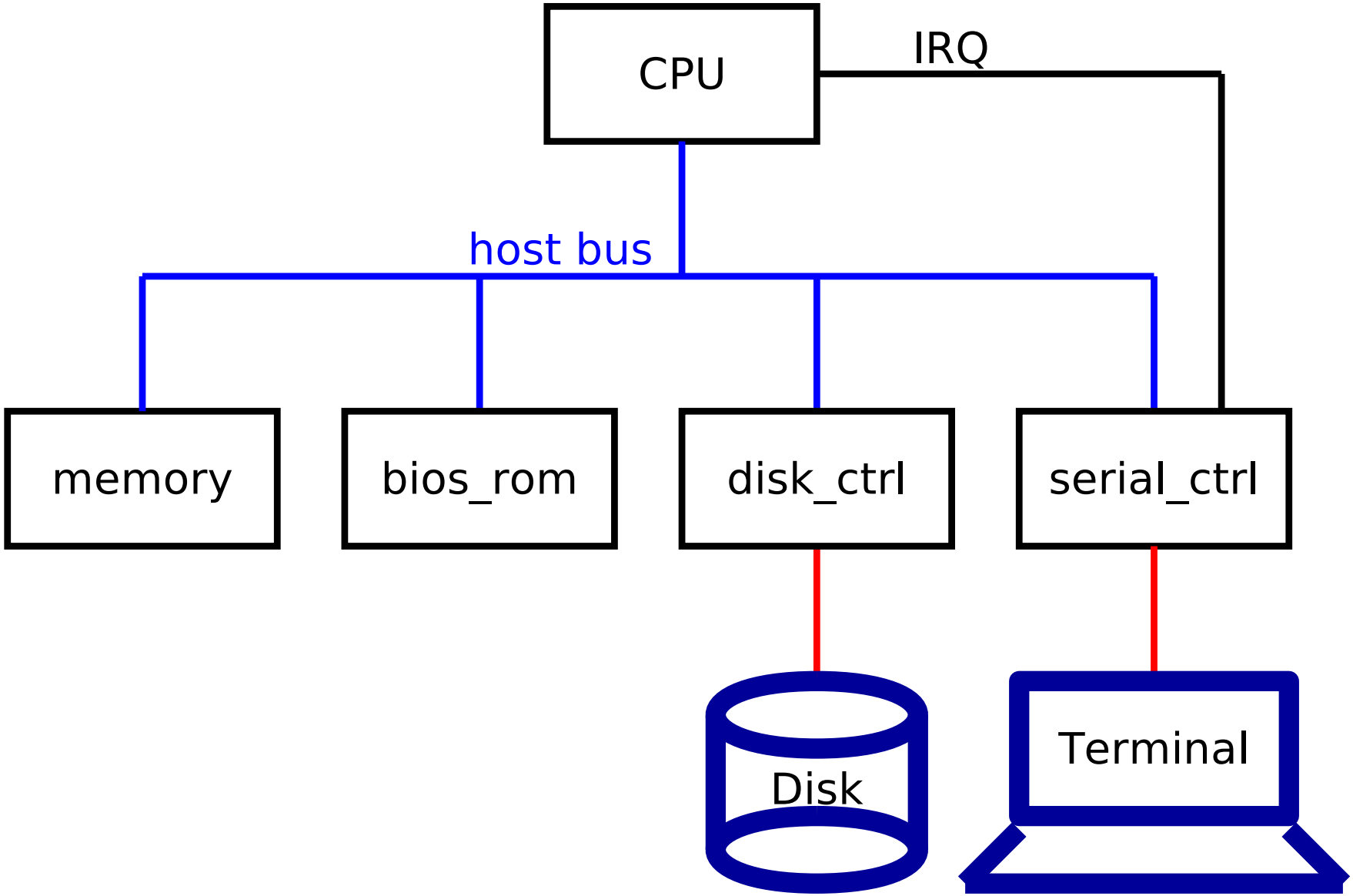
PC – Aufbau (vereinfacht)



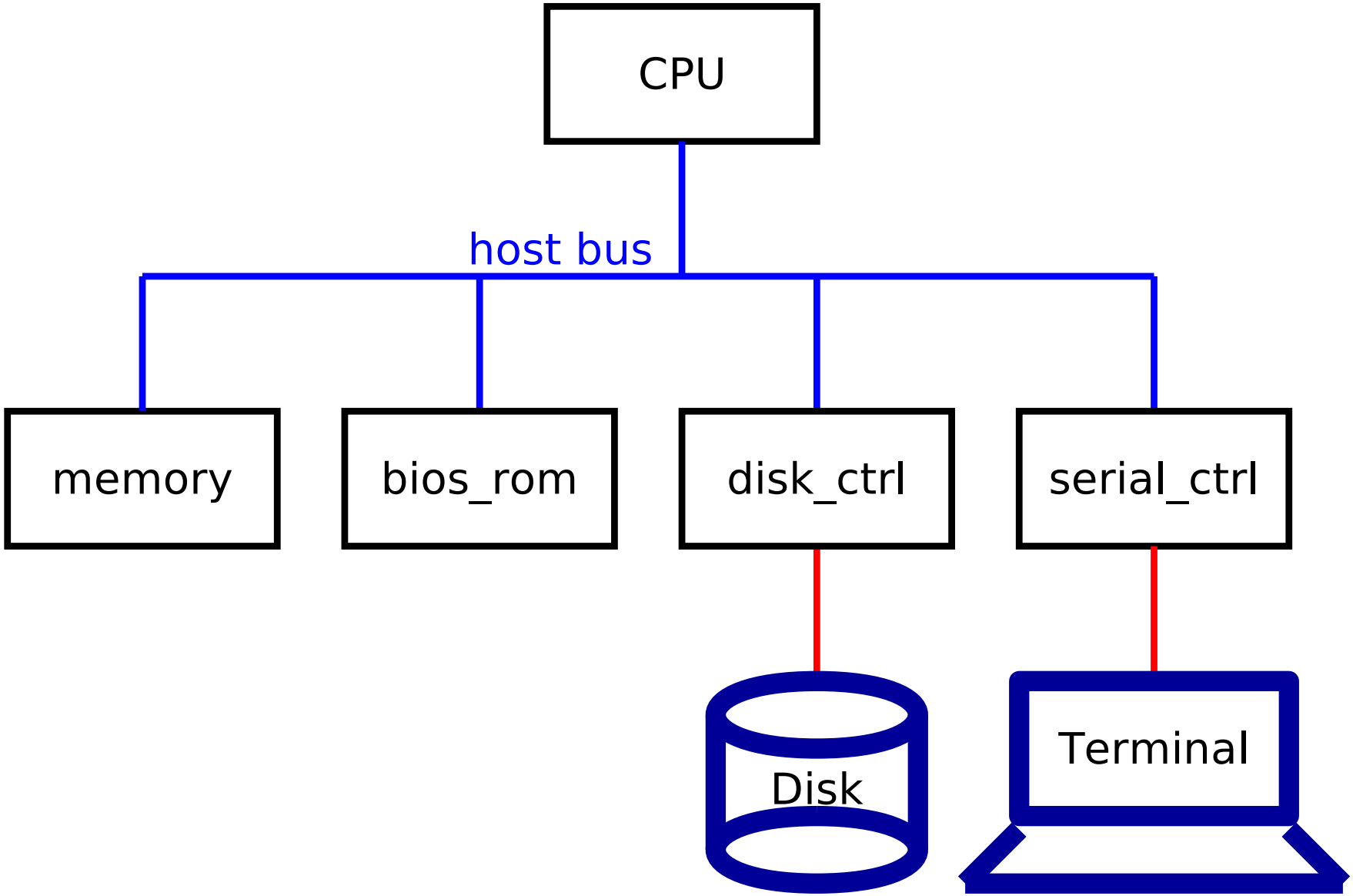
PC – Aufbau (vereinfacht)



PC – Aufbau (vereinfacht)



PC – Aufbau (vereinfacht)



Übersicht

- stark vereinfachter PC-Aufbau
- Komponenten direkt an `host_bus` angebunden
- `host_bus` begrenzt auf 8 Bit Datenleitungen
- `host_bus` unterscheidet nicht zwischen I/O- und Speicherzugriffen

CPU

Zunächst:

- Befehlssatz: Untermenge der 80x86-Architektur
- beschränkt auf Protected Mode/32-Bit Mode
- keine Exceptions
- keine Interrupts
- Reset: `EIP` ist Basisadresse des ROMS (`0xE000`)
- keine *FPU*, kein *MMX* oder andere Erweiterungen

- Dokumentation: „Intel 64 and IA-32 Architectures Software Developer's Manual“
  - Volume 1: Basic Architecture
  - Volume 2A, 2B, 2C: Instruction Set Architecture
  - Volume 3A, 3B, 3C: System Programming Guide
  - `http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html`

CPU

Exkurs: Assembler-Programmierung

Festplattencontroller / Festplattensimulator

- Basisadresse: `0xD000`
- ausgewertete Adressleitungen: `10`
- Adresse `0x000`–`0x003`: Register für Blocknummer (big-endian), *rw*
- Adresse `0x007`: Fehlerregister, *rw*
- Adresse `0x00b`: Block Lesen (0), Block Schreiben (1), *w*
- Adresse `0x200`–`0x3FF`: Block Puffer, *rw*
- andere: Fest verdrahtet mit Masse, Schreiben wird ignoriert.

Festplattencontroller / Festplattensimulator – Beispiel

```
# %eax: Block-Nummer (0-65535)
# %edi: Buffer-Adresse

read_block:
      movb $0, 0xd000+0     # Blocknummer setzen
      movb $0, 0xd000+1
      movb %ah, 0xd000+2
      movb %al, 0xd000+3
      movb $0, 0xd000+0xb   # Lesen starten
      movl $0, %eax         # Bytes kopieren
loop: movb 0xd000+0x200(%eax), %dl
      movb %dl, (%edi, %eax, 1)
      addl $1, %eax
      cmpl $512, %eax
      jne loop
      ret                   # fertig
```

Serielle Schnittstelle / Debug-Schnittstelle

- minimale Variante
- Basisadresse: `0xD800`
- ausgewertete Adressleitungen: `4`
- Adresse `0x0`: Schreiben gibt das Datum auf `stdout` aus, *w*
- andere: reserviert

Serielle Schnittstelle / Debug-Schnittstelle – Beispiel

```
# %eax: Pointer auf Zeichenkette

print_string:
      movb (%eax), %dl    # Hole Zeichen
      cmpb $'\0', %dl     # Ende, wenn Zeichen = '\0'
      je done
      movb %dl, 0xd800+0 # Gib Zeichen aus
      addl $1, %eax       # naechstes Zeichen
      jmp print_string
done:
      ret                 # fertig
```

Speicher

- 32 KiB Memory:
  - fest verdrahtet ab Adresse `0x0000`
  - Größe: 32 KiB
  - Tipp: sinnvoll initialisieren
- 4 KiB Bios ROM:
  - fest verdrahtet ab Adresse `0xE000`
  - Größe: 4 KiB
  - soll Inhalt aus `.rom`-Datei lesen können
  - wenn `.rom`-Datei zu klein, Rest sinnvoll initialisieren

Gestellte Dateien

- `Makefile`: kleines Makefile zum Bauen
- `setup.*`: Erzeugt/beendet die VM
- `bus/sig_host_bus.*`: Implementierung des Host-Busses
- `comp/serial-ctrl.*`: Minmal-Implementierung der seriellen (Debug-)Schnittstelle
- `comp/roms/*.S`: Assembler-Quellen für Test-Programme:
  - `bios_print_simple.S`: Einfachste „Hallo Welt“-Variante
  - `bios_print.S`: nochmal „Hallo Welt“ als Schleife
  - `bios_boot.S`: minimaler Boot-Loader

Sämtliche Dateien dürfen verändert werden!