A Dozen Things about the z/Architecture

Josef “Jeff” Sipek

<jeffpc@josefsipek.net>

What's z/Architecture?

What's z/Architecture?

What's z/Architecture?

- IBM System z mainframes (2006)

What's z/Architecture?

- IBM System z mainframes (2006)
- formerly zSeries (2000)

What's z/Architecture?

- IBM System z mainframes (2006)
- formerly zSeries (2000)
- Descendent of ESA/390 (1990)

What's z/Architecture?

- IBM System z mainframes (2006)
- formerly zSeries (2000)
- Descendent of ESA/390 (1990)
- Descendent of ESA/370 (1988)

What's z/Architecture?

- IBM System z mainframes (2006)
- formerly zSeries (2000)
- Descendent of ESA/390 (1990)
- Descendent of ESA/370 (1988)
- Descendent of 370/XA (1983)

What's z/Architecture?

- IBM System z mainframes (2006)
- formerly zSeries (2000)
- Descendent of ESA/390 (1990)
- Descendent of ESA/370 (1988)
- Descendent of 370/XA (1983)
- Descendent of System/370 (1970)

What's z/Architecture?

- IBM System z mainframes (2006)
- formerly zSeries (2000)
- Descendent of ESA/390 (1990)
- Descendent of ESA/370 (1988)
- Descendent of 370/XA (1983)
- Descendent of System/370 (1970)
- Descendent of System/360 (1964)

0. Aren’t mainframes dead?

0. Aren't mainframes dead?

No!

0. Aren’t mainframes dead?

No!

Ok, this one was a freebie...

1. Random Trivia

- “Storage” is RAM, *not* disk

1. Random Trivia

- “Storage” is RAM, *not* disk
- Big endian machine

1. Random Trivia

- “Storage” is RAM, *not* disk
- Big endian machine
- MSB is bit 0

1. Random Trivia

- “Storage” is RAM, *not* disk
- Big endian machine
- MSB is bit 0
- Instruction retry

1. Random Trivia

- “Storage” is RAM, *not* disk
- Big endian machine
- MSB is bit 0
- Instruction retry
- This presentation is too short to summarize >1200 pages of documentation

2. Backwards Compatibility

- Full *application* backwards compatibility

2. Backwards Compatibility

- Full *application* backwards compatibility
  - Take a binary from 1960’s

2. Backwards Compatibility

- Full *application* backwards compatibility
  - Take a binary from 1960's
  - Run unmodified on zSeries

2. Backwards Compatibility

- Full *application* backwards compatibility
  - Take a binary from 1960's
  - Run unmodified on zSeries
  - Same output, but faster!

2. Backwards Compatibility

- Full *application* backwards compatibility
  - Take a binary from 1960's
  - Run unmodified on zSeries
  - Same output, but faster!
  - Cannot remove unprivileged instructions

2. Backwards Compatibility

- Full *application* backwards compatibility
  - Take a binary from 1960's
  - Run unmodified on zSeries
  - Same output, but faster!
  - Cannot remove unprivileged instructions
- OS should...

2. Backwards Compatibility

- Full *application* backwards compatibility
  - Take a binary from 1960's
  - Run unmodified on zSeries
  - Same output, but faster!
  - Cannot remove unprivileged instructions
- OS should...
  - Use new facilities

2. Backwards Compatibility

- Full *application* backwards compatibility
  - Take a binary from 1960's
  - Run unmodified on zSeries
  - Same output, but faster!
  - Cannot remove unprivileged instructions
- OS should...
  - Use new facilities
  - Hide the differences from applications

3. Specifications

- 6 types of Processor Units (PUs)

3. Specifications

- 6 types of Processor Units (PUs)
  - CP: generic PU

3. Specifications

- 6 types of Processor Units (PUs)
  - CP: generic PU
  - IFL: like CP, lacking $\sim$2 instructions, perfect for Linux

3. Specifications

- 6 types of Processor Units (PUs)
  - CP: generic PU
  - IFL: like CP, lacking $\sim$2 instructions, perfect for Linux
  - ICF: (runs special firmware)

3. Specifications

- 6 types of Processor Units (PUs)
  - CP: generic PU
  - IFL: like CP, lacking $\sim$2 instructions, perfect for Linux
  - ICF: (runs special firmware)
  - zAAP: for Java workloads

3. Specifications

- 6 types of Processor Units (PUs)
  - CP: generic PU
  - IFL: like CP, lacking $\sim$2 instructions, perfect for Linux
  - ICF: (runs special firmware)
  - zAAP: for Java workloads
  - zIIP: for DB workloads

3. Specifications

- 6 types of Processor Units (PUs)
  - CP: generic PU
  - IFL: like CP, lacking $\sim$2 instructions, perfect for Linux
  - ICF: (runs special firmware)
  - zAAP: for Java workloads
  - zIIP: for DB workloads
  - SAP: manages I/O (runs special firmware)

3. Specifications

- 6 types of Processor Units (PUs)
  - CP: generic PU
  - IFL: like CP, lacking $\sim$2 instructions, perfect for Linux
  - ICF: (runs special firmware)
  - zAAP: for Java workloads
  - zIIP: for DB workloads
  - SAP: manages I/O (runs special firmware)
- 1 to 54 PUs

3. Specifications

- 6 types of Processor Units (PUs)
  - CP: generic PU
  - IFL: like CP, lacking $\sim$2 instructions, perfect for Linux
  - ICF: (runs special firmware)
  - zAAP: for Java workloads
  - zIIP: for DB workloads
  - SAP: manages I/O (runs special firmware)
- 1 to 54 PUs
- 16 to 512 GB of storage

3. Specifications

- 6 types of Processor Units (PUs)
  - CP: generic PU
  - IFL: like CP, lacking $\sim$2 instructions, perfect for Linux
  - ICF: (runs special firmware)
  - zAAP: for Java workloads
  - zIIP: for DB workloads
  - SAP: manages I/O (runs special firmware)
- 1 to 54 PUs
- 16 to 512 GB of storage
- 1212 to 2003 kg (2672 to 4407 lbs)

3. Specifications

- 6 types of Processor Units (PUs)
  - CP: generic PU
  - IFL: like CP, lacking $\sim$2 instructions, perfect for Linux
  - ICF: (runs special firmware)
  - zAAP: for Java workloads
  - zIIP: for DB workloads
  - SAP: manages I/O (runs special firmware)
- 1 to 54 PUs
- 16 to 512 GB of storage
- 1212 to 2003 kg (2672 to 4407 lbs)
- 6.3 to 18.3 kW

3. Specifications

- 6 types of Processor Units (PUs)
  - CP: generic PU
  - IFL: like CP, lacking $\sim$2 instructions, perfect for Linux
  - ICF: (runs special firmware)
  - zAAP: for Java workloads
  - zIIP: for DB workloads
  - SAP: manages I/O (runs special firmware)
- 1 to 54 PUs
- 16 to 512 GB of storage
- 1212 to 2003 kg (2672 to 4407 lbs)
- 6.3 to 18.3 kW
- 21.5 to 62.4 kBTU/hr

4. Architecture Modes

- 2 supported modes

4. Architecture Modes

- 2 supported modes
- ESA/390

4. Architecture Modes

- 2 supported modes
- ESA/390
  - 31-bit addressing

4. Architecture Modes

- 2 supported modes
- ESA/390
  - 31-bit addressing
  - 32-bit arithmetic

4. Architecture Modes

- 2 supported modes
- ESA/390
  - 31-bit addressing
  - 32-bit arithmetic
- z/Architecture

4. Architecture Modes

- 2 supported modes
- ESA/390
  - 31-bit addressing
  - 32-bit arithmetic
- z/Architecture
  - Superset of ESA/390

4. Architecture Modes

- 2 supported modes
- ESA/390
  - 31-bit addressing
  - 32-bit arithmetic
- z/Architecture
  - Superset of ESA/390
  - 64-bit addressing

4. Architecture Modes

- 2 supported modes
- ESA/390
  - 31-bit addressing
  - 32-bit arithmetic
- z/Architecture
  - Superset of ESA/390
  - 64-bit addressing
  - 64-bit arithmetic

4. Architecture Modes

- 2 supported modes
- ESA/390
  - 31-bit addressing
  - 32-bit arithmetic
- z/Architecture
  - Superset of ESA/390
  - 64-bit addressing
  - 64-bit arithmetic
  - 32-bit arithmetic instructions still available

4. Architecture Modes

- 2 supported modes
- ESA/390
  - 31-bit addressing
  - 32-bit arithmetic
- z/Architecture
  - Superset of ESA/390
  - 64-bit addressing
  - 64-bit arithmetic
  - 32-bit arithmetic instructions still available
- Switch between modes at run time

5. Storage

- “Storage-reference-happy architecture”

5. Storage

- “Storage-reference-happy architecture”
- Dynamic Address Translation (with a TLB)

5. Storage

- “Storage-reference-happy architecture”
- Dynamic Address Translation (with a TLB)
  - 3 translation modes

5. Storage

- “Storage-reference-happy architecture”
- Dynamic Address Translation (with a TLB)
  - 3 translation modes
  - Up to 15 address spaces can be used at any time

5. Storage

- “Storage-reference-happy architecture”
- Dynamic Address Translation (with a TLB)
  - 3 translation modes
  - Up to 15 address spaces can be used at any time
- Page protection

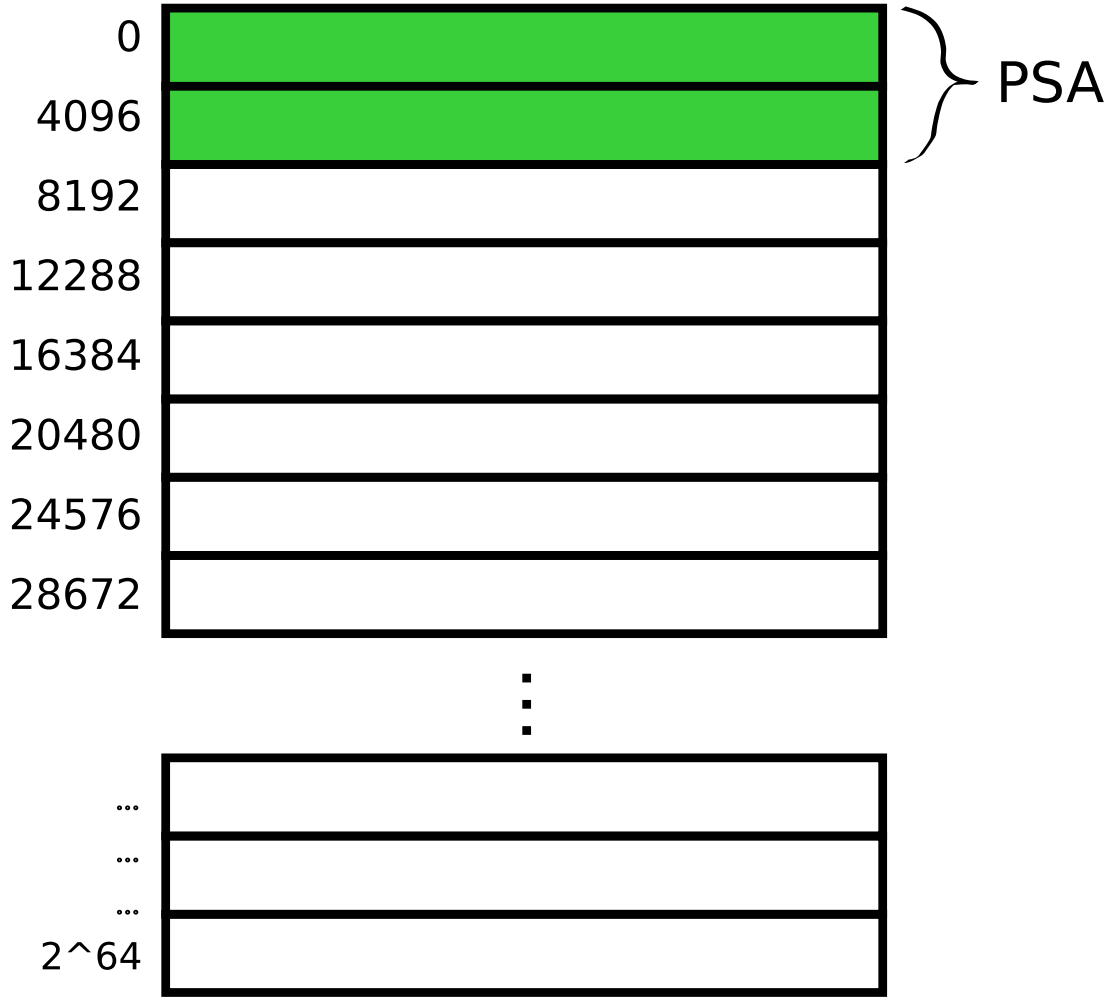
5. Storage

- “Storage-reference-happy architecture”
- Dynamic Address Translation (with a TLB)
  - 3 translation modes
  - Up to 15 address spaces can be used at any time
- Page protection
- Storage keys

5. Storage

- “Storage-reference-happy architecture”
- Dynamic Address Translation (with a TLB)
  - 3 translation modes
  - Up to 15 address spaces can be used at any time
- Page protection
- Storage keys
- Low-address protection

5. Storage



6. Addressing Modes

- *Addressing* and *architecture* modes are independently set

6. Addressing Modes

- *Addressing* and *architecture* modes are independently set
- When in ESA/390 *arch* mode

6. Addressing Modes

- *Addressing* and *architecture* modes are independently set
- When in ESA/390 *arch* mode
  - 24-bit, or 31-bit addressing

6. Addressing Modes

- *Addressing* and *architecture* modes are independently set
- When in ESA/390 *arch* mode
  - 24-bit, or 31-bit addressing
    - 16MB or 2GB of addressable storage

6. Addressing Modes

- *Addressing* and *architecture* modes are independently set
- When in ESA/390 *arch* mode
  - 24-bit, or 31-bit addressing
    - 16MB or 2GB of addressable storage
  - Bit 32 (Basic Addressing – BA) in PSW

6. Addressing Modes

- *Addressing* and *architecture* modes are independently set
- When in ESA/390 *arch* mode
  - 24-bit, or 31-bit addressing
    - 16MB or 2GB of addressable storage
  - Bit 32 (Basic Addressing – BA) in PSW
- When in z/Architecture *arch* mode

6. Addressing Modes

- *Addressing* and *architecture* modes are independently set
- When in ESA/390 *arch* mode
  - 24-bit, or 31-bit addressing
    - 16MB or 2GB of addressable storage
  - Bit 32 (Basic Addressing – BA) in PSW
- When in z/Architecture *arch* mode
  - 24-bit, 31-bit, or 64-bit addressing

6. Addressing Modes

- *Addressing* and *architecture* modes are independently set
- When in ESA/390 *arch* mode
  - 24-bit, or 31-bit addressing
    - 16MB or 2GB of addressable storage
  - Bit 32 (Basic Addressing – BA) in PSW
- When in z/Architecture *arch* mode
  - 24-bit, 31-bit, or 64-bit addressing
  - Bits 31 (Extended Addressing – EA) & 32 (BA) in PSW must be 1

6. Addressing Modes

- *Addressing* and *architecture* modes are independently set
- When in ESA/390 *arch* mode
  - 24-bit, or 31-bit addressing
    - 16MB or 2GB of addressable storage
  - Bit 32 (Basic Addressing – BA) in PSW
- When in z/Architecture *arch* mode
  - 24-bit, 31-bit, or 64-bit addressing
  - Bits 31 (Extended Addressing – EA) & 32 (BA) in PSW must be 1
    - 16MB, 2GB, or 16EB of addressable storage

7. Registers

- 16 General purpose registers (64/32-bit)

7. Registers

- 16 General purpose registers (64/32-bit)
- 16 Floating point registers (64-bit)

7. Registers

- 16 General purpose registers (64/32-bit)
- 16 Floating point registers (64-bit)
- 16 Access registers (32-bit)

7. Registers

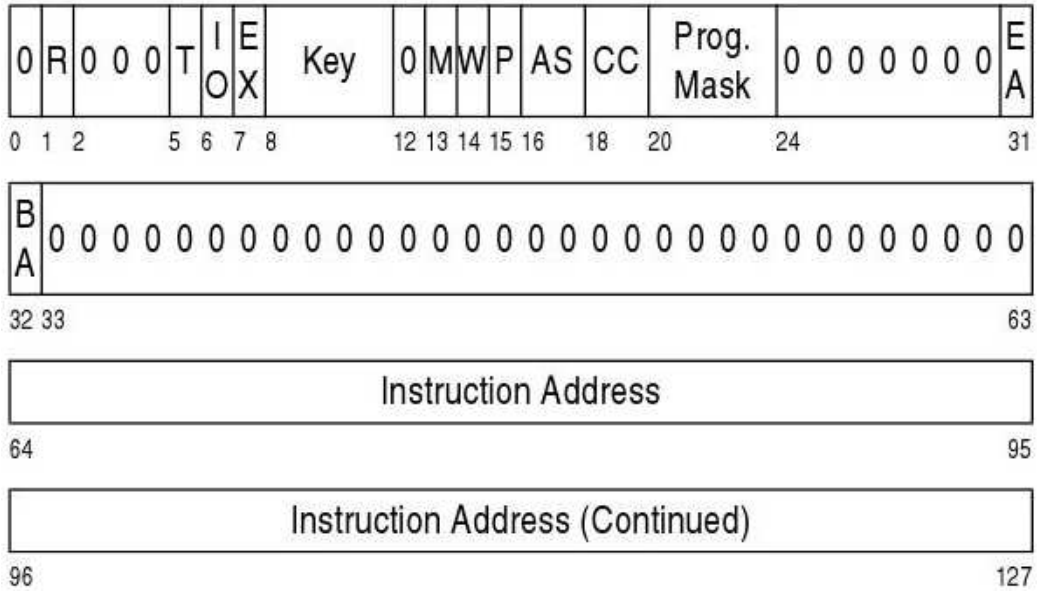
- 16 General purpose registers (64/32-bit)
- 16 Floating point registers (64-bit)
- 16 Access registers (32-bit)
- 16 Control registers (64/32-bit)

7. Registers

- 16 General purpose registers (64/32-bit)
- 16 Floating point registers (64-bit)
- 16 Access registers (32-bit)
- 16 Control registers (64/32-bit)
- Program-Status Word (PSW)

7. Registers

- 16 General purpose registers (64/32-bit)
- 16 Floating point registers (64-bit)
- 16 Access registers (32-bit)
- 16 Control registers (64/32-bit)
- Program-Status Word (PSW)



8. Instruction Set

- CISC architecture

8. Instruction Set

- CISC architecture
  - Complex Instruction Set Computer

8. Instruction Set

- CISC architecture
  - Complex Instruction Set Computer
  - System/360 (November 1970) had 143 instructions

8. Instruction Set

- CISC architecture
  - Complex Instruction Set Computer
  - System/360 (November 1970) had 143 instructions
  - z9 (September 2005) describes 689 instructions

8. Instruction Set

- CISC architecture
  - Complex Instruction Set Computer
  - System/360 (November 1970) had 143 instructions
  - z9 (September 2005) describes 689 instructions
- Instructions are always...

8. Instruction Set

- CISC architecture
  - Complex Instruction Set Computer
  - System/360 (November 1970) had 143 instructions
  - z9 (September 2005) describes 689 instructions
- Instructions are always...
  - 2, 4, or 6 bytes long

8. Instruction Set

- CISC architecture
  - Complex Instruction Set Computer
  - System/360 (November 1970) had 143 instructions
  - z9 (September 2005) describes 689 instructions
- Instructions are always...
  - 2, 4, or 6 bytes long
  - Aligned on 2-byte boundary

8. Instruction Set

- CISC architecture
  - Complex Instruction Set Computer
  - System/360 (November 1970) had 143 instructions
  - z9 (September 2005) describes 689 instructions
- Instructions are always...
  - 2, 4, or 6 bytes long
  - Aligned on 2-byte boundary
- Many instructions reference storage

9. ADD & CIPHER MESSAGE

- 43 variants of ADD

9. ADD & CIPHER MESSAGE

- 43 variants of ADD
  - A, AR, AY, AG, AGR, AGF, AGFR, AXBR, ADB, ADBR, ADTR, AEB, AEBR, AP, AH, AHY, AHI, AGHI, AFI, AGFI, and many more...

9. ADD & CIPHER MESSAGE

- 43 variants of ADD
  - A, AR, AY, AG, AGR, AGF, AGFR, AXBR, ADB, ADBR, ADTR, AEB, AEBR, AP, AH, AHY, AHI, AGHI, AFI, AGFI, and many more...
- CIPHER MESSAGE

9. ADD & CIPHER MESSAGE

- 43 variants of ADD
  - A, AR, AY, AG, AGR, AGF, AGFR, AXBR, ADB, ADBR, ADTR, AEB, AEBR, AP, AH, AHY, AHI, AGHI, AFI, AGFI, and many more...
- CIPHER MESSAGE
  - Encrypts a buffer

9. ADD & CIPHER MESSAGE

- 43 variants of ADD
  - A, AR, AY, AG, AGR, AGF, AGFR, AXBR, ADB, ADBR, ADTR, AEB, AEBR, AP, AH, AHY, AHI, AGHI, AFI, AGFI, and many more...
- CIPHER MESSAGE
  - Encrypts a buffer
  - Hashes a buffer

10. Interpretive-Execution Facility

- Virtualization the proper way

10. Interpretive-Execution Facility

- Virtualization the proper way
- Instruction — **`SIE`**

10. Interpretive-Execution Facility

- Virtualization the proper way
- Instruction — **`SIE`**
  - Runs virtually all instructions natively

10. Interpretive-Execution Facility

- Virtualization the proper way
- Instruction — **`SIE`**
  - Runs virtually all instructions natively
  - Hardware-speed most of the time

10. Interpretive-Execution Facility

- Virtualization the proper way
- Instruction — **`SIE`**
  - Runs virtually all instructions natively
  - Hardware-speed most of the time
  - Some instructions are intercepted, host must...

10. Interpretive-Execution Facility

- Virtualization the proper way
- Instruction — **`SIE`**
  - Runs virtually all instructions natively
  - Hardware-speed most of the time
  - Some instructions are intercepted, host must...
    - Emulate the instruction

10. Interpretive-Execution Facility

- Virtualization the proper way
- Instruction — **`SIE`**
  - Runs virtually all instructions natively
  - Hardware-speed most of the time
  - Some instructions are intercepted, host must...
    - Emulate the instruction
    - Reissue **`SIE`**

10. Interpretive-Execution Facility

- Virtualization the proper way
- Instruction — **SIE**
  - Runs virtually all instructions natively
  - Hardware-speed most of the time
  - Some instructions are intercepted, host must...
    - Emulate the instruction
    - Reissue **SIE**
- **SIE** uses a State Descriptor

10. Interpretive-Execution Facility

- Virtualization the proper way
- Instruction — **SIE**
  - Runs virtually all instructions natively
  - Hardware-speed most of the time
  - Some instructions are intercepted, host must...
    - Emulate the instruction
    - Reissue **SIE**
- **SIE** uses a State Descriptor
  - Guest PSW

10. Interpretive-Execution Facility

- Virtualization the proper way
- Instruction — **SIE**
  - Runs virtually all instructions natively
  - Hardware-speed most of the time
  - Some instructions are intercepted, host must...
    - Emulate the instruction
    - Reissue **SIE**
- **SIE** uses a State Descriptor
  - Guest PSW
  - Guest control registers

10. Interpretive-Execution Facility

- Virtualization the proper way
- Instruction — **SIE**
  - Runs virtually all instructions natively
  - Hardware-speed most of the time
  - Some instructions are intercepted, host must...
    - Emulate the instruction
    - Reissue **SIE**
- **SIE** uses a State Descriptor
  - Guest PSW
  - Guest control registers
  - Defines guest storage

10. Interpretive-Execution Facility

- Virtualization the proper way
- Instruction — **SIE**
  - Runs virtually all instructions natively
  - Hardware-speed most of the time
  - Some instructions are intercepted, host must...
    - Emulate the instruction
    - Reissue **SIE**
- **SIE** uses a State Descriptor
  - Guest PSW
  - Guest control registers
  - Defines guest storage
  - Interception controls

11. Interruptions

- 6 Classes of interruptions

11. Interruptions

- 6 Classes of interruptions
  - Restart

11. Interruptions

- 6 Classes of interruptions
  - Restart
  - External (e.g., timers)

11. Interruptions

- 6 Classes of interruptions
  - Restart
  - External (e.g., timers)
  - Supervisor-call (e.g., used for “system calls”)

11. Interruptions

- 6 Classes of interruptions
  - Restart
  - External (e.g., timers)
  - Supervisor-call (e.g., used for “system calls”)
  - Program (e.g., DAT faults, integer over/under-flows)

11. Interruptions

- 6 Classes of interruptions
  - Restart
  - External (e.g., timers)
  - Supervisor-call (e.g., used for “system calls”)
  - Program (e.g., DAT faults, integer over/under-flows)
  - Machine-check (e.g., hardware malfunctions)

11. Interruptions

- 6 Classes of interruptions
  - Restart
  - External (e.g., timers)
  - Supervisor-call (e.g., used for “system calls”)
  - Program (e.g., DAT faults, integer over/under-flows)
  - Machine-check (e.g., hardware malfunctions)
  - I/O (e.g., operation complete, device req. attention)

11. Interruptions

- 6 Classes of interruptions
  - Restart
  - External (e.g., timers)
  - Supervisor-call (e.g., used for “system calls”)
  - Program (e.g., DAT faults, integer over/under-flows)
  - Machine-check (e.g., hardware malfunctions)
  - I/O (e.g., operation complete, device req. attention)
- A number of them can be masked out

12. Channels

- CPUs are meant to run user code

12. Channels

- CPUs are meant to run user code
- Baby-sitting IO wastes time

12. Channels

- CPUs are meant to run user code
- Baby-sitting IO wastes time
- Prepare an IO operation on a CPU

12. Channels

- CPUs are meant to run user code
- Baby-sitting IO wastes time
- Prepare an IO operation on a CPU
- Let co-processors execute it

12. Channels

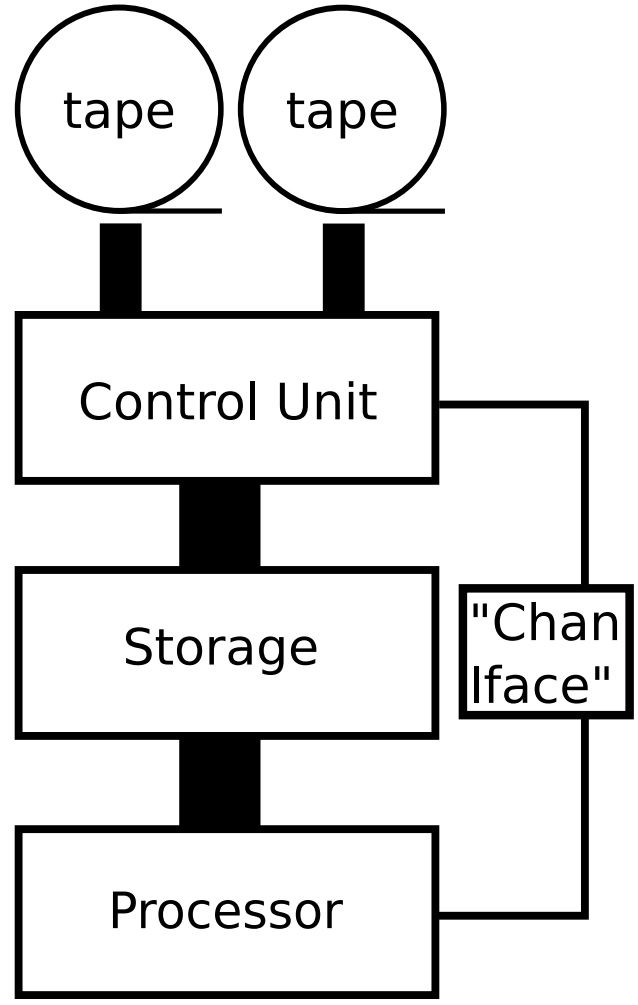
- CPUs are meant to run user code
- Baby-sitting IO wastes time
- Prepare an IO operation on a CPU
- Let co-processors execute it
- CPU can continue executing user applications

12. Channels

- CPUs are meant to run user code
- Baby-sitting IO wastes time
- Prepare an IO operation on a CPU
- Let co-processors execute it
- CPU can continue executing user applications
- Similar to DMA, but *way* more advanced

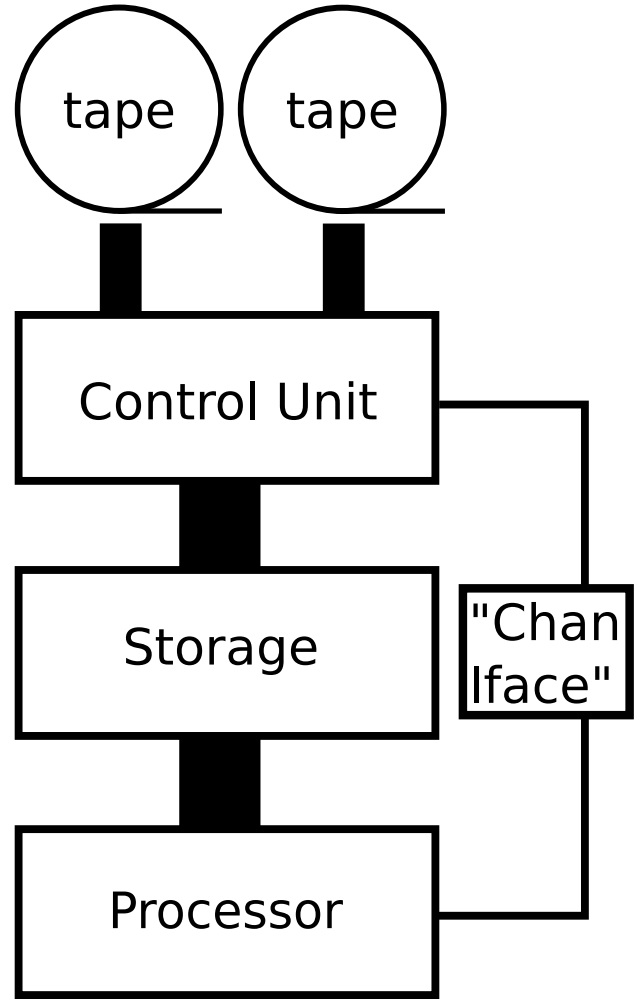
12. Channels - Connections

- A device is attached to a Control Unit



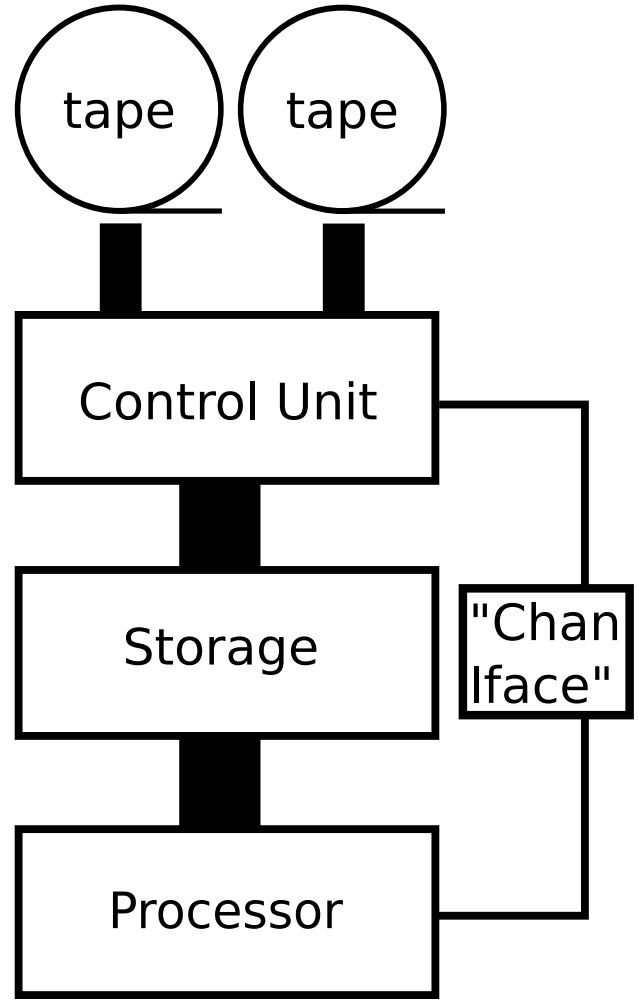
12. Channels - Connections

- A device is attached to a Control Unit
- Control Units are connected to...



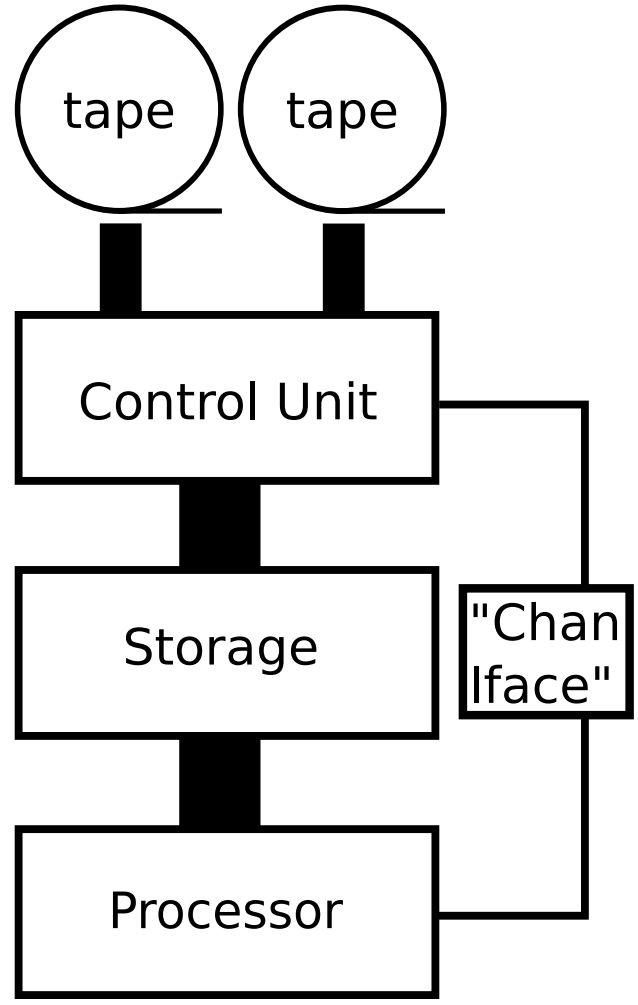
12. Channels - Connections

- A device is attached to a Control Unit
- Control Units are connected to...
  - Storage



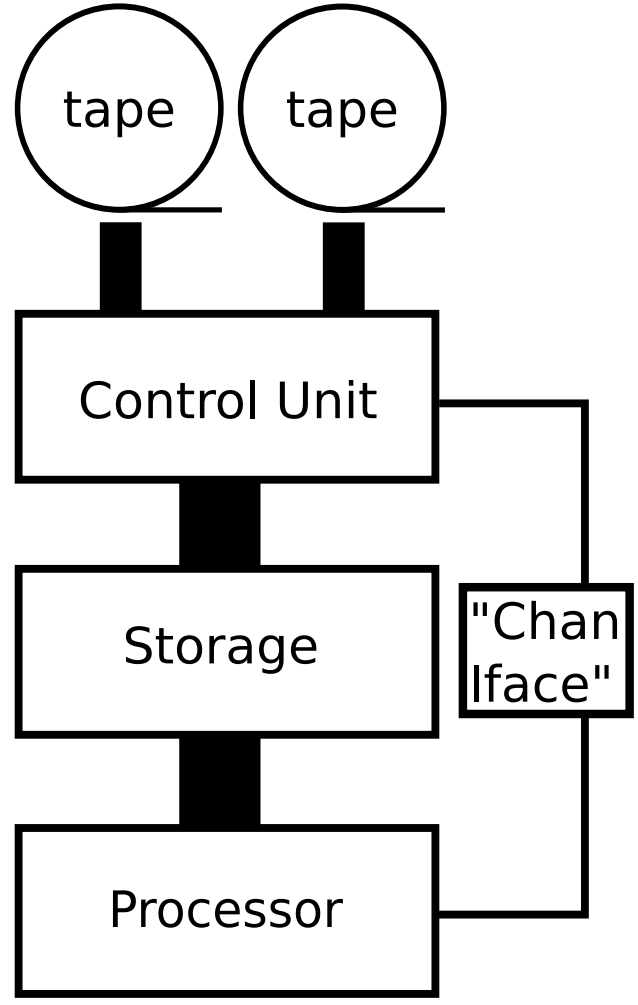
12. Channels - Connections

- A device is attached to a Control Unit
- Control Units are connected to...
  - Storage
  - “Channel interface” for a lack of a better term



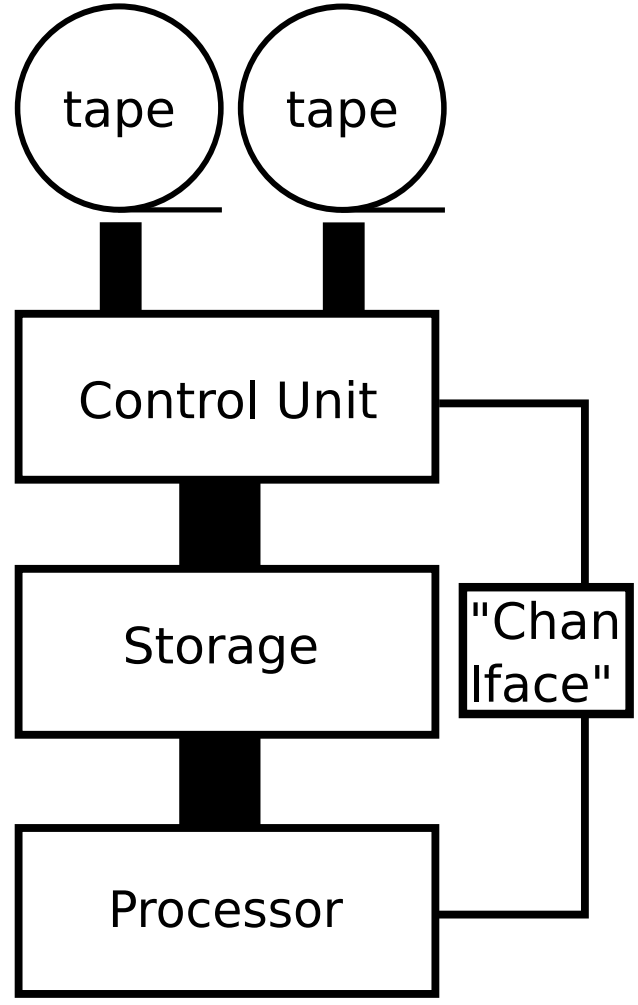
12. Channels - Connections

- A device is attached to a Control Unit
- Control Units are connected to...
  - Storage
  - “Channel interface” for a lack of a better term
- *Logical* link between a device and “channel interface” box is a *subchannel*



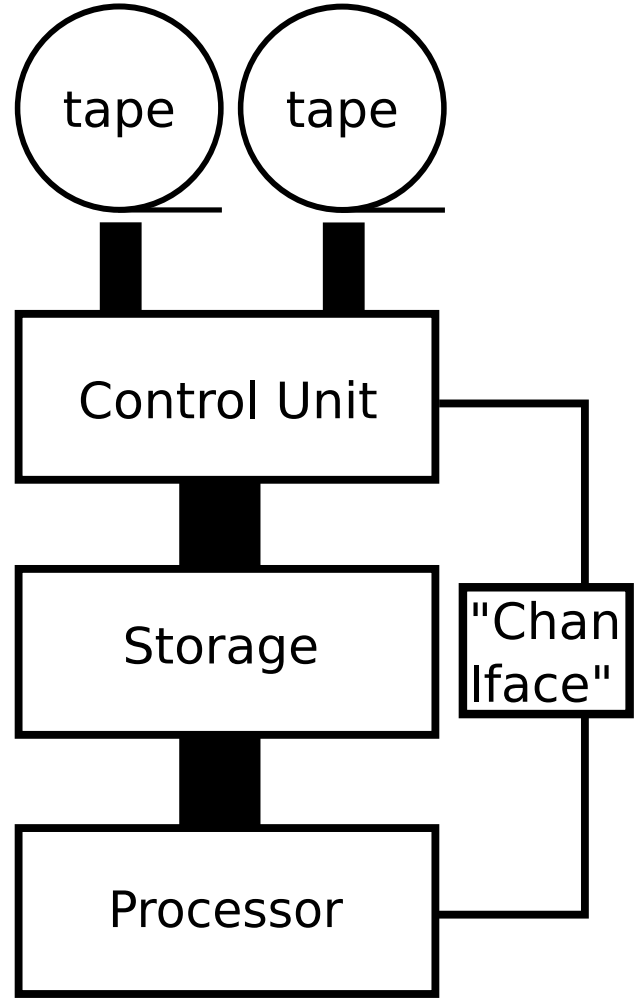
12. Channels - Connections

- A device is attached to a Control Unit
- Control Units are connected to...
  - Storage
  - “Channel interface” for a lack of a better term
- *Logical* link between a device and “channel interface” box is a *subchannel*
- *Physical* connections are a lot more complex



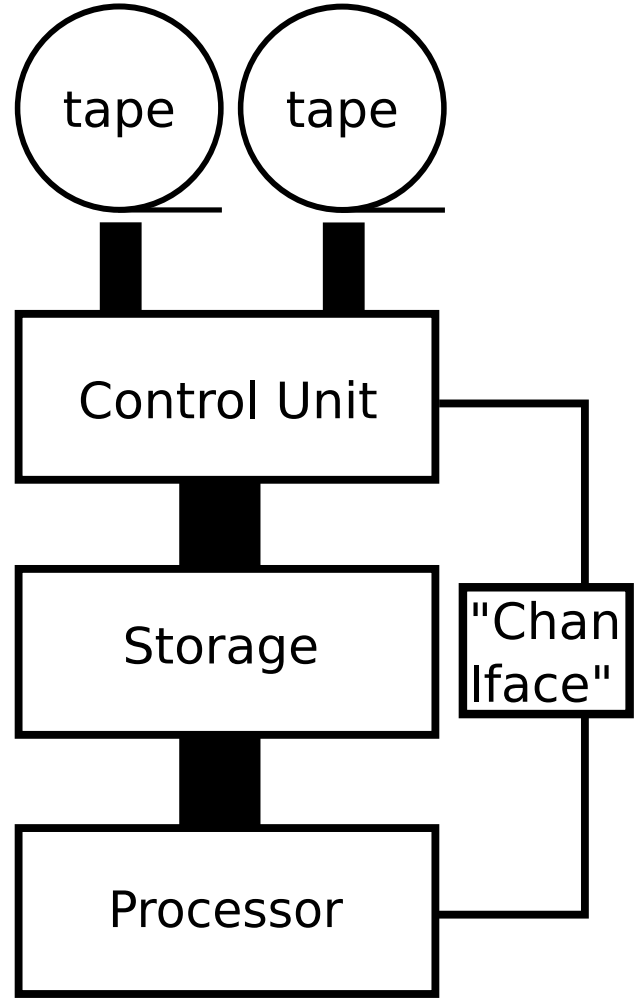
12. Channels - Issuing I/O

- Save Channel Command Words (CCWs) in storage



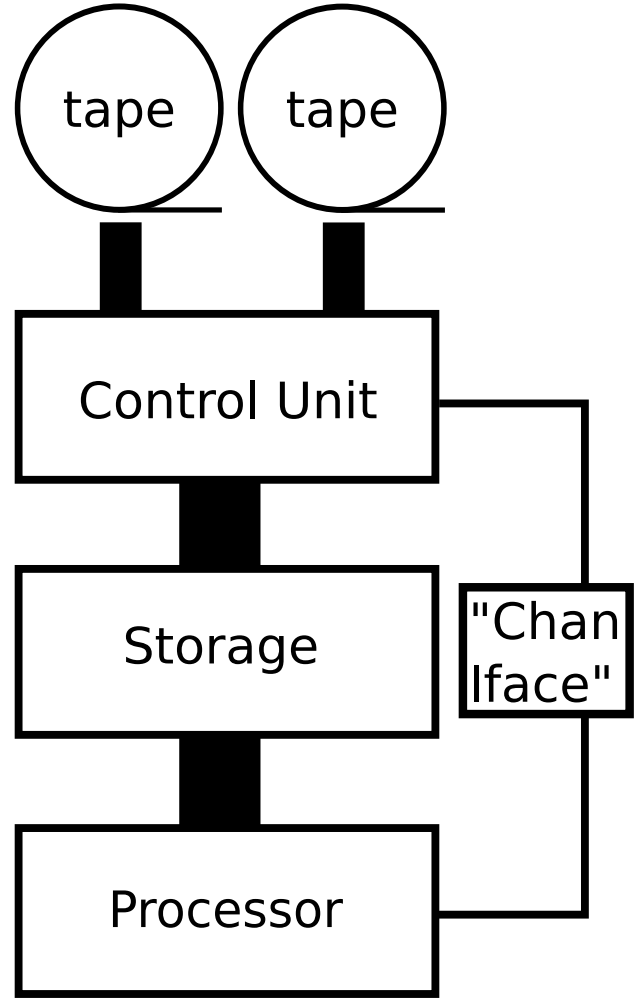
12. Channels - Issuing I/O

- Save Channel Command Words (CCWs) in storage
- Signal CU to execute stored commands



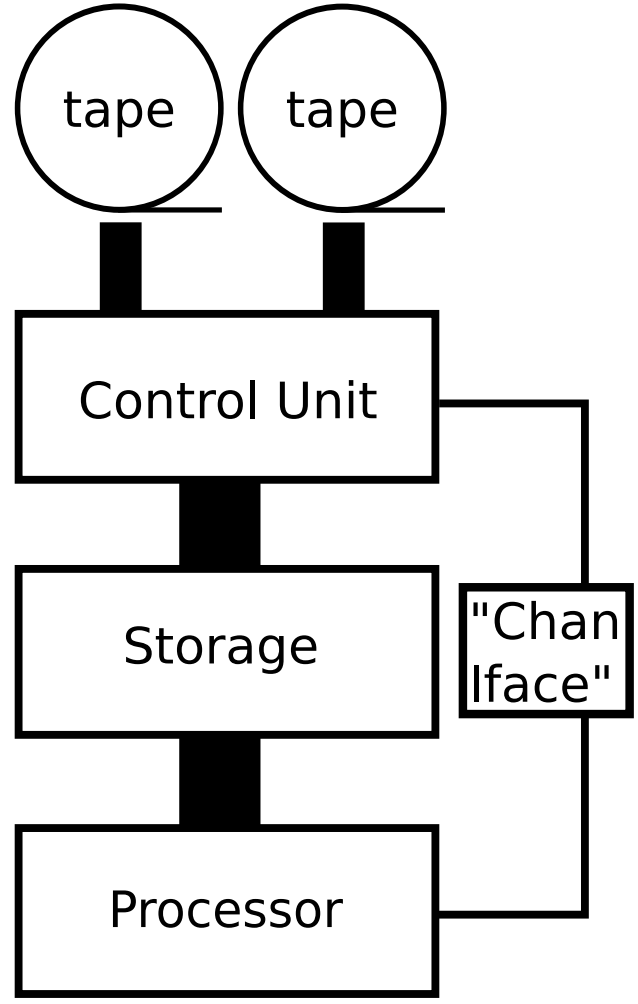
12. Channels - Issuing I/O

- Save Channel Command Words (CCWs) in storage
- Signal CU to execute stored commands
- CU generates an IO interrupt when...



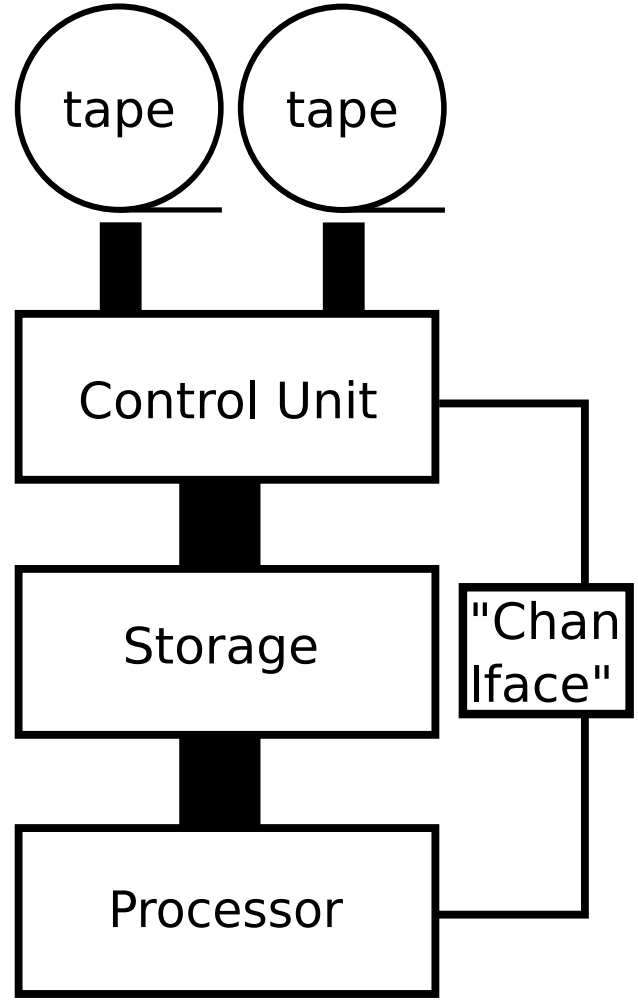
12. Channels - Issuing I/O

- Save Channel Command Words (CCWs) in storage
- Signal CU to execute stored commands
- CU generates an IO interrupt when...
  - IO completes

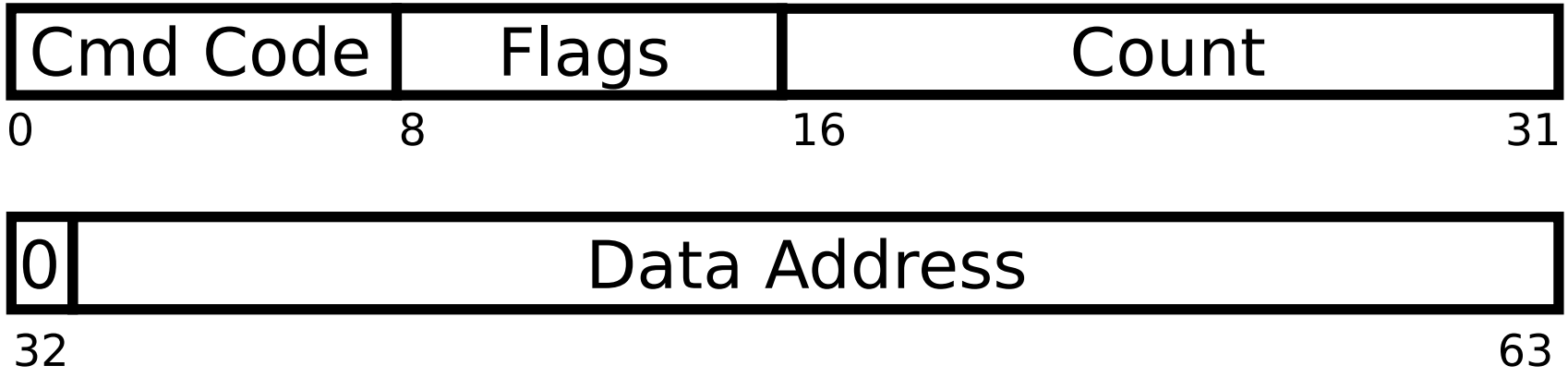


12. Channels - Issuing I/O

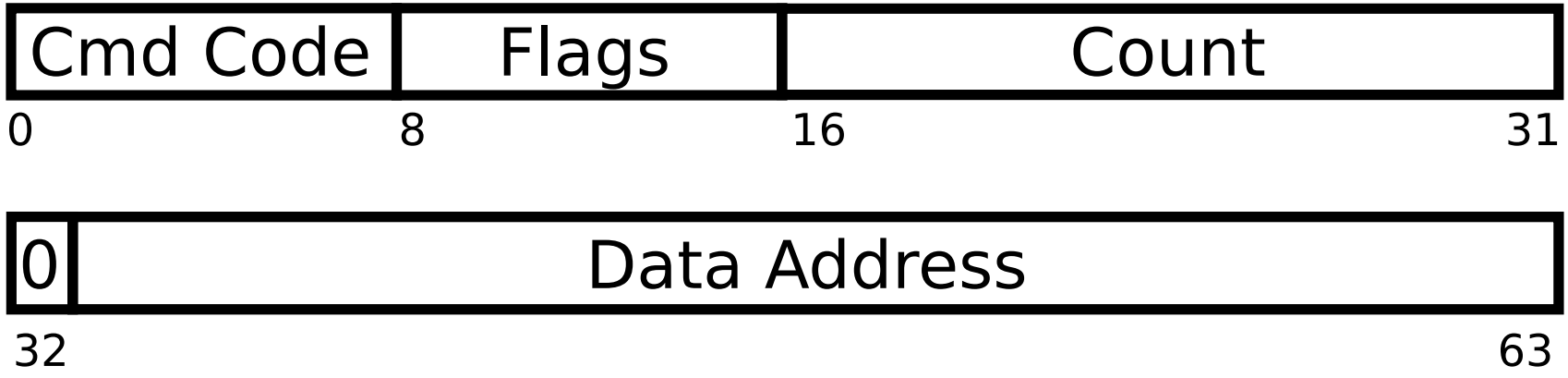
- Save Channel Command Words (CCWs) in storage
- Signal CU to execute stored commands
- CU generates an IO interrupt when...
  - IO completes
  - Error occurs



12. Channels - CCWs

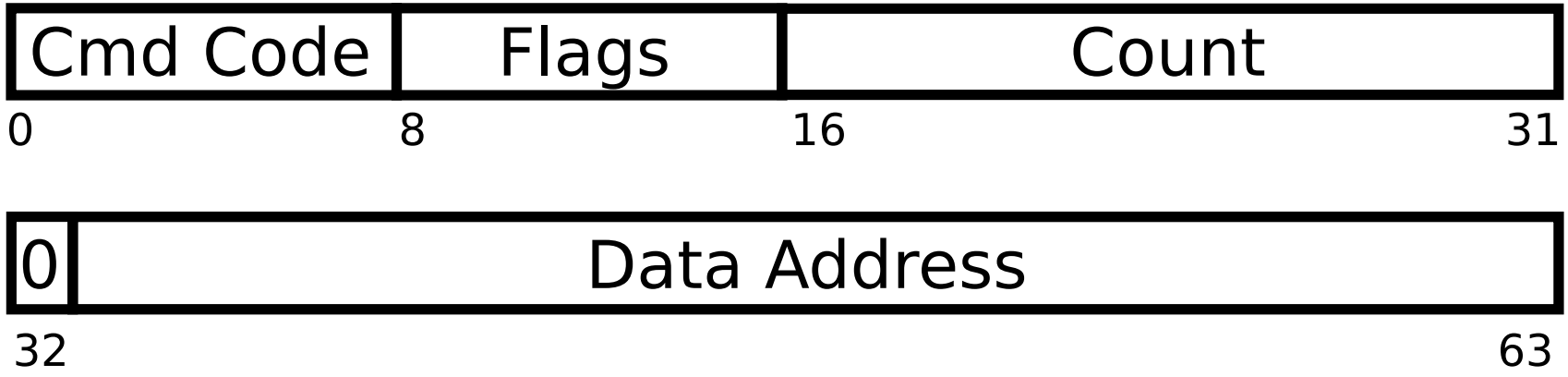


12. Channels - CCWs



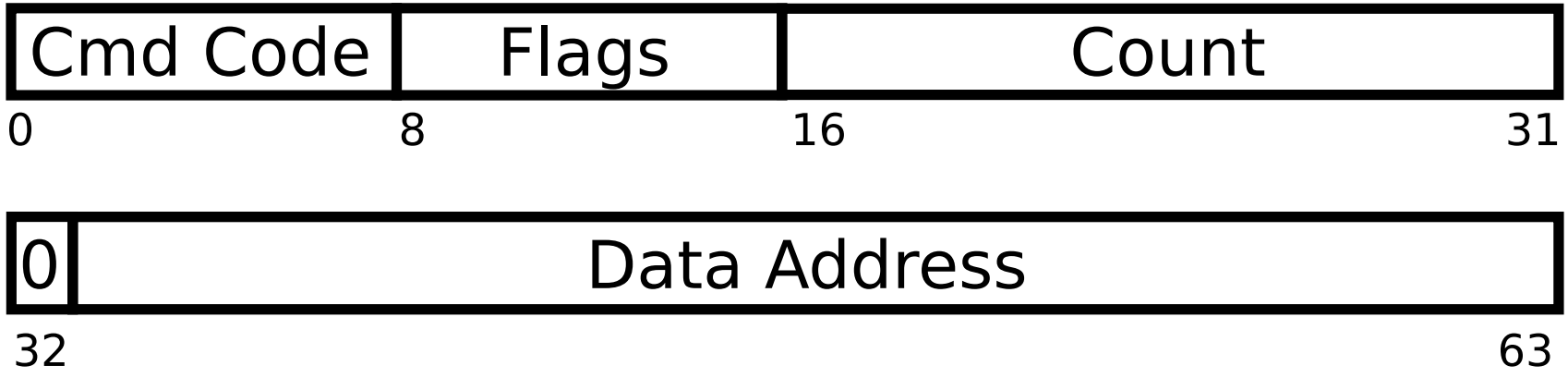
- Command

12. Channels - CCWs



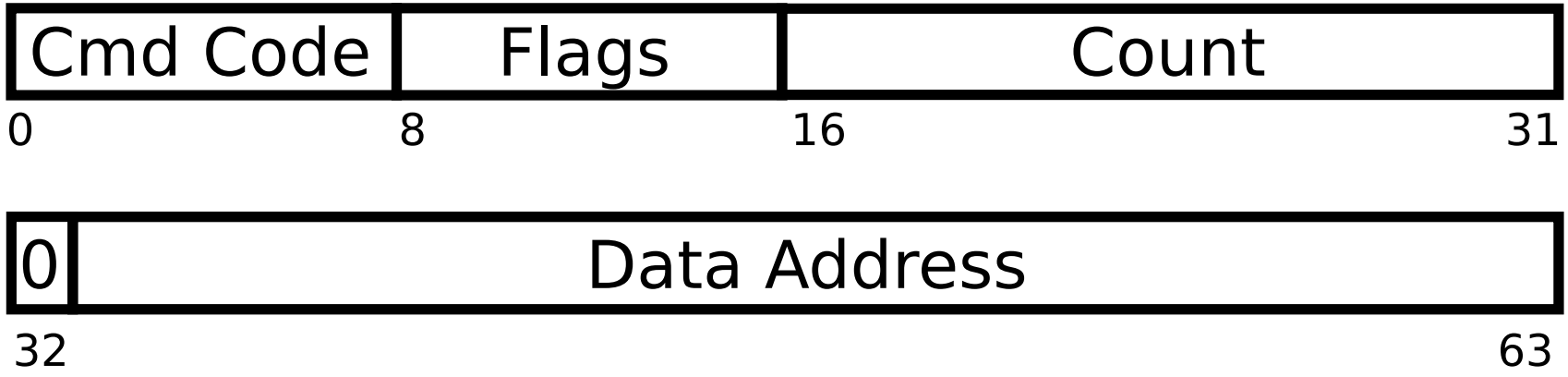
- Command
  - Write
  - Read
  - Read Backward
  - Control
  - Sense
  - Sense ID
  - Transfer in Control (branch!)

12. Channels - CCWs



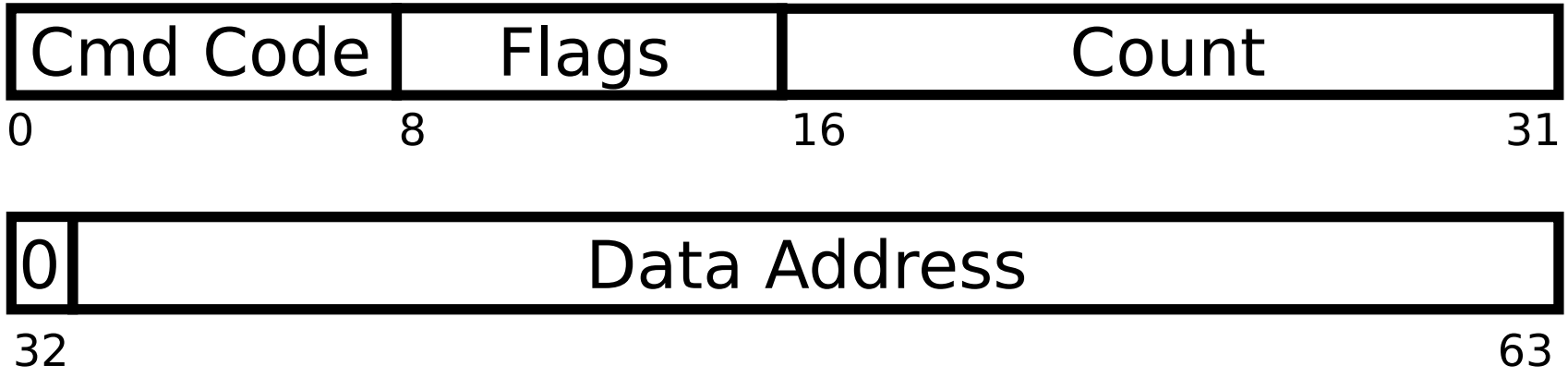
- Command
- Flags

12. Channels - CCWs



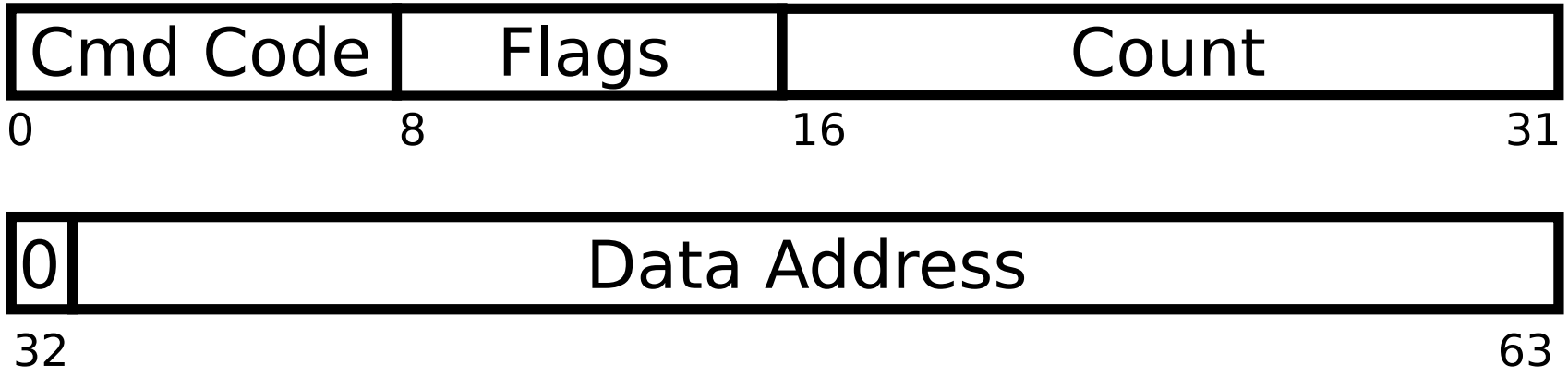
- Command
- Flags
  - CCWs chaining
  - Skip (do not read)
  - Indirect addressing (2 modes: IDA, MIDA)
  - Suspend execution
  - ...

12. Channels - CCWs



- Command
- Flags
- Byte count

12. Channels - CCWs



- Command
- Flags
- Byte count
- Buffer address

My thoughts exactly...

- “This architecture is awesome!”

My thoughts exactly...

- “This architecture is awesome!”
- “I wish I could play with one.”

My thoughts exactly...

- “This architecture is awesome!”
- “I wish I could play with one.”
  - Hercules: open source emulator

My thoughts exactly...

- “This architecture is awesome!”
- “I wish I could play with one.”
  - Hercules: open source emulator
- “I wish I could run Linux on it.”

My thoughts exactly...

- “This architecture is awesome!”
- “I wish I could play with one.”
  - Hercules: open source emulator
- “I wish I could run Linux on it.”
  - You can!

My thoughts exactly...

- “This architecture is awesome!”
- “I wish I could play with one.”
  - Hercules: open source emulator
- “I wish I could run Linux on it.”
  - You can!
- “I wish I could write an OS for it.”

My thoughts exactly...

- “This architecture is awesome!”
- “I wish I could play with one.”
  - Hercules: open source emulator
- “I wish I could run Linux on it.”
  - You can!
- “I wish I could write an OS for it.”
  - Funny you should ask...

HVF

- OS/Hypervisor

HVF

- OS/Hypervisor
- z/Architecture

HVF

- OS/Hypervisor
- z/Architecture
- Written from scratch

HVF

- OS/Hypervisor
- z/Architecture
- Written from scratch
- Mostly C, with few bits of assembly

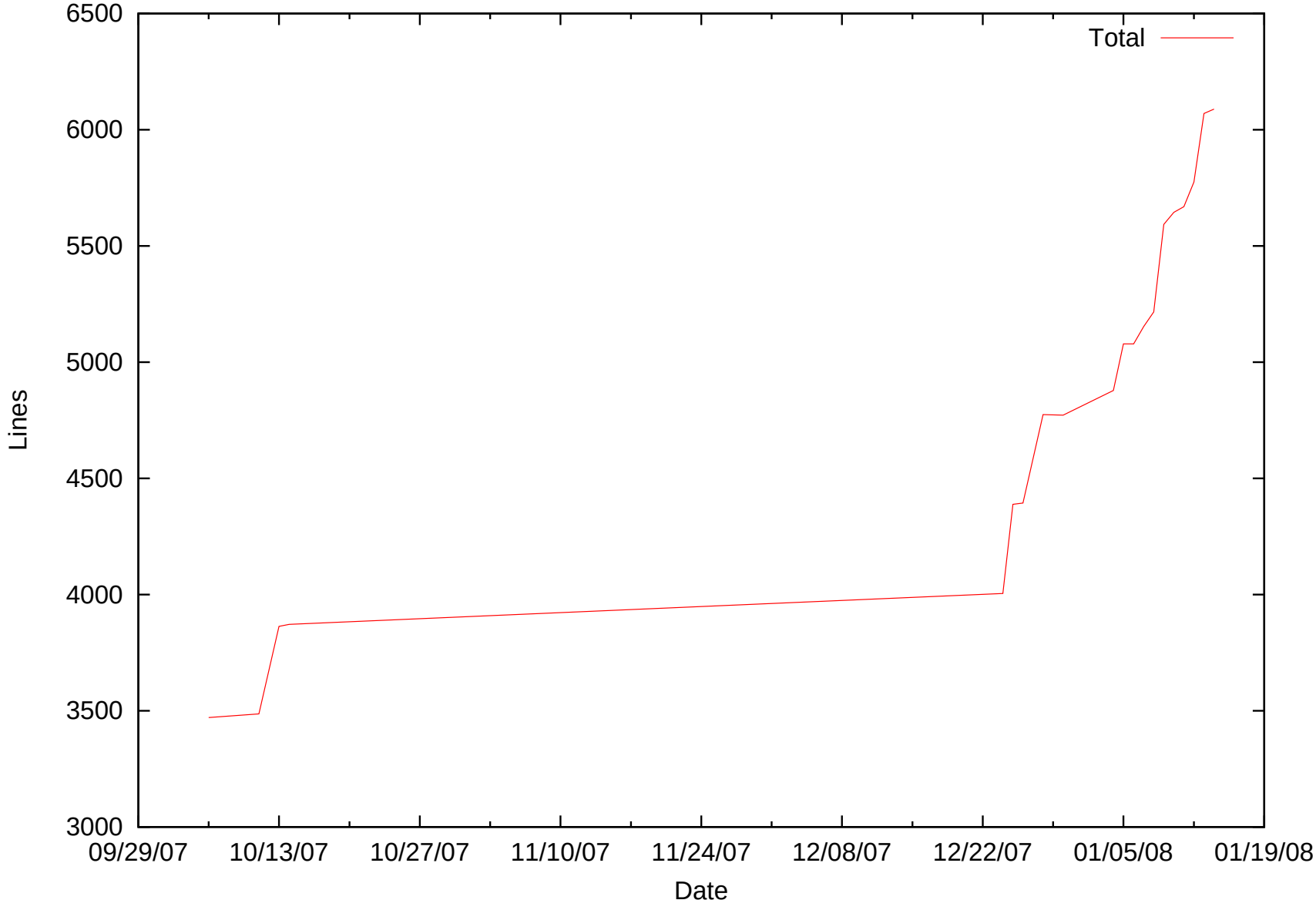
HVF

- OS/Hypervisor
- z/Architecture
- Written from scratch
- Mostly C, with few bits of assembly
- Announced January 11, 2008

HVF

- OS/Hypervisor
- z/Architecture
- Written from scratch
- Mostly C, with few bits of assembly
- Announced January 11, 2008
- GPLv2

HVF - Repository Size



HVF - Status & TODO

- What's done

HVF - Status & TODO

- What's done
  - Buddy & SLAB allocators

HVF - Status & TODO

- What's done
  - Buddy & SLAB allocators
  - Virtual memory

HVF - Status & TODO

- What's done
  - Buddy & SLAB allocators
  - Virtual memory
  - Basic console IO

HVF - Status & TODO

- What's done
  - Buddy & SLAB allocators
  - Virtual memory
  - Basic console IO
- What needs to be done

HVF - Status & TODO

- What's done
  - Buddy & SLAB allocators
  - Virtual memory
  - Basic console IO
- What needs to be done
  - $\sim$30 **`FIXME`**s of varying difficulty

HVF - Status & TODO

- What's done
  - Buddy & SLAB allocators
  - Virtual memory
  - Basic console IO
- What needs to be done
  - $\sim$30 **`FIXME`**s of varying difficulty
  - User directory (currently WIP)

HVF - Status & TODO

- What's done
  - Buddy & SLAB allocators
  - Virtual memory
  - Basic console IO
- What needs to be done
  - $\sim$30 **`FIXME`**s of varying difficulty
  - User directory (currently WIP)
  - Virtual devices/IO proxying

HVF - Status & TODO

- What's done
  - Buddy & SLAB allocators
  - Virtual memory
  - Basic console IO
- What needs to be done
  - ∼30 **`FIXME`**s of varying difficulty
  - User directory (currently WIP)
  - Virtual devices/IO proxying
  - Testing, and more testing

HVF - Status & TODO

- What's done
  - Buddy & SLAB allocators
  - Virtual memory
  - Basic console IO
- What needs to be done
  - ∼30 **`FIXME`**s of varying difficulty
  - User directory (currently WIP)
  - Virtual devices/IO proxying
  - Testing, and more testing
- Contact info

HVF - Status & TODO

- What's done
  - Buddy & SLAB allocators
  - Virtual memory
  - Basic console IO
- What needs to be done
  - ∼30 **`FIXME`**s of varying difficulty
  - User directory (currently WIP)
  - Virtual devices/IO proxying
  - Testing, and more testing
- Contact info
  - Mailing list: `lists.josefsipek.net/listinfo/hvf`

HVF - Status & TODO

- What's done
  - Buddy & SLAB allocators
  - Virtual memory
  - Basic console IO
- What needs to be done
  - ∼30 **`FIXME`**s of varying difficulty
  - User directory (currently WIP)
  - Virtual devices/IO proxying
  - Testing, and more testing
- Contact info
  - Mailing list: `lists.josefsipek.net/listinfo/hvf`
  - **`#hvf`** on OFTC

Questions?

References

- z/Architecture Principles of Operation (SA22-7832-05)
- System z Architecture Course
- System/370 XA — Interpretive Execution (SA22-7095-1)
- Hercules Emulator
  `www.hercules-390.org`
- Installing Debian under Hercules
  `www.josefsipek.net/docs/s390-linux/`

- Diagram of PSW taken from SA22-7832-05
- Image of z9 from infoworld.com's website