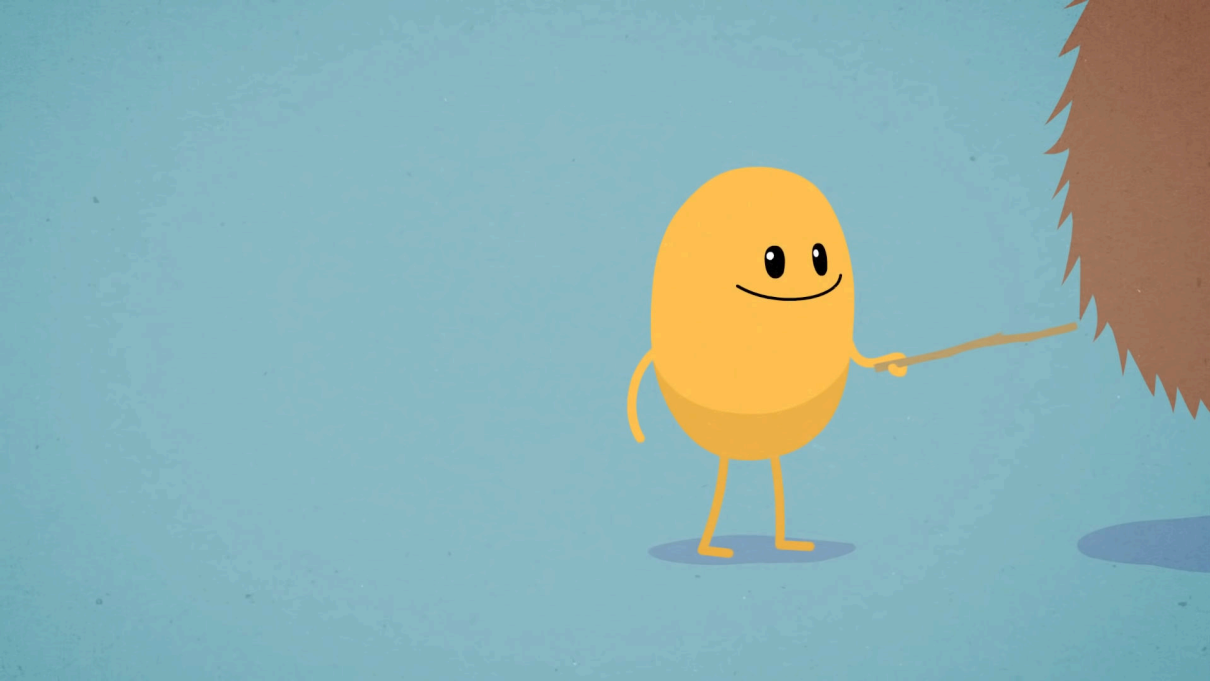
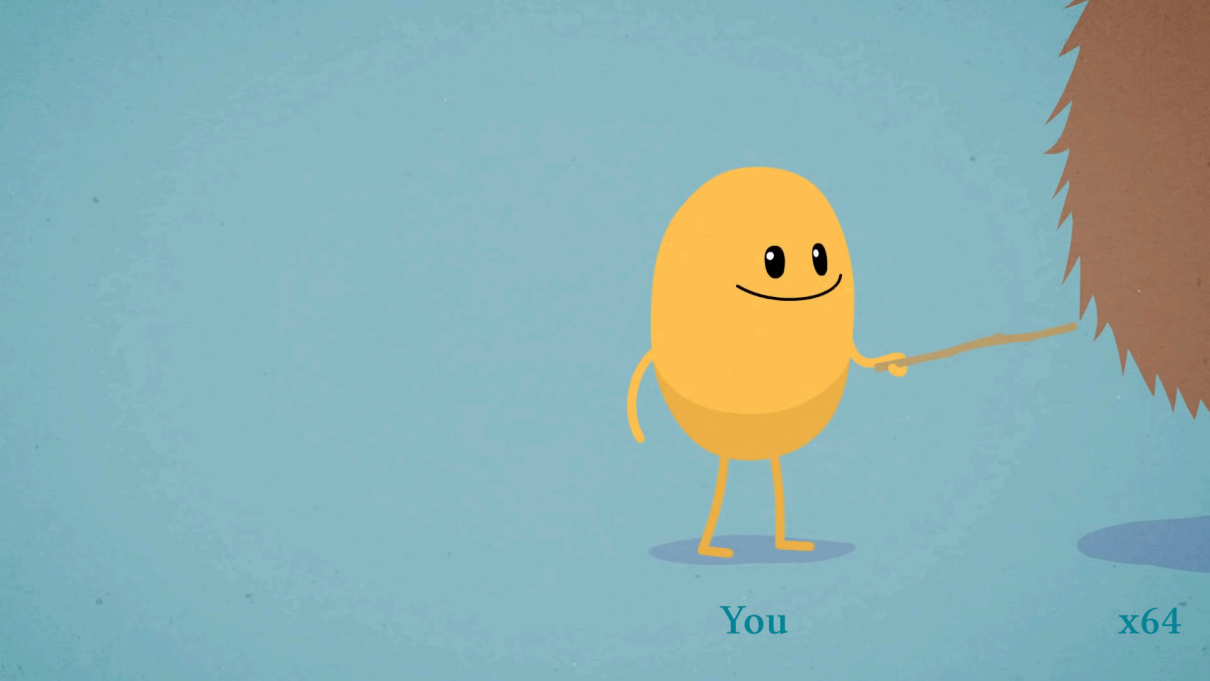
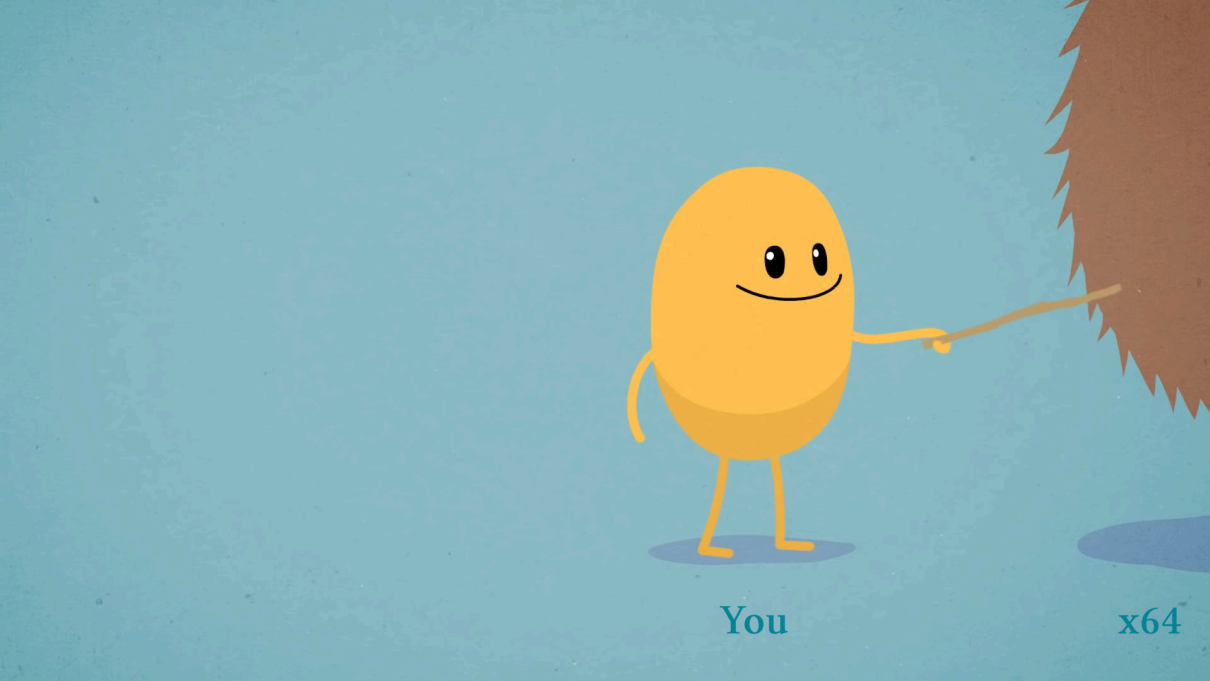



You
x64


You
x64


You
x64

Paging

You

x64


You
x64

(so many)

DUMB WAYS TO DIE

You

x64

(so many)

DUMB WAYS TO DIE – *eine Post-mortem-Analyse*

Grafiken von Dumb Ways to Die™ © Metro Trains Melbourne — youtu.be/IJNR2EpS0jw

Der Page Frame Allocator

- Informationen aus Multiboot und bekannte Adressen

Der Page Frame Allocator

- Informationen aus Multiboot und bekannte Adressen
- Verwaltung über (zwei) verkettete Listen
  - mittels dynamische Speicherverwaltung
  - eine Liste für Kernel, eine für Userspace

Der Page Frame Allocator

- Informationen aus Multiboot und bekannte Adressen
- Verwaltung über (zwei) verkettete Listen
  - mittels dynamische Speicherverwaltung
  - eine Liste für Kernel, eine für Userspace
- zuerst verfügbare Bereiche aufnehmen
  - nur vollständig umschlossene Seitenrahmen aufnehmen
  - überlappende Bereiche in Multiboot sind möglich

Der Page Frame Allocator

- Informationen aus Multiboot und bekannte Adressen
- Verwaltung über (zwei) verkettete Listen
  - mittels dynamische Speicherverwaltung
  - eine Liste für Kernel, eine für Userspace
- zuerst verfügbare Bereiche aufnehmen
  - nur vollständig umschlossene Seitenrahmen aufnehmen
  - überlappende Bereiche in Multiboot sind möglich
- danach belegte Bereiche entfernen
  - sobald ein Seitenrahmen auch nur *gestriffen* wird
  - vier Fälle möglich

Der Page Frame Allocator

- Informationen aus Multiboot und bekannte Adressen
- Verwaltung über (zwei) verkettete Listen
  - mittels dynamische Speicherverwaltung
  - eine Liste für Kernel, eine für Userspace
- zuerst verfügbare Bereiche aufnehmen
  - nur vollständig umschlossene Seitenrahmen aufnehmen
  - überlappende Bereiche in Multiboot sind möglich
- danach belegte Bereiche entfernen
  - sobald ein Seitenrahmen auch nur *gestriffen* wird
  - vier Fälle möglich
- Funktionen um einen freien Seitenrahmen zu bekommen und wieder zurückzugeben
  - in $\mathcal{O}(1)$ (keine Schleife!)
  - kein Zusammenfügen notwendig

Das Paging

- Grundfunktionen zum Speicher einblenden (`map`) und wieder ausblenden (`unmap`)

Das Paging

- Grundfunktionen zum Speicher einblenden (`map`) und wieder ausblenden (`unmap`)
- zuerst Datenstrukturen für vollständiges korrektes Kernelmapping aufsetzen, dann `cr3` umschalten

Das Paging

- Grundfunktionen zum Speicher einblenden (`map`) und wieder ausblenden (`unmap`)
- zuerst Datenstrukturen für vollständiges korrektes Kernelmapping aufsetzen, dann `cr3` umschalten
- *present*-Bit von unbenutzte Einträge auf `0` setzen
  - sonst passieren wilde Dinge
  - Seiten vom Page Frame Allocator können beliebigen Inhalt haben
    → meist nur auf Hardware (Qemu/KVM mit genulltem Speicher)

Das Paging

- Grundfunktionen zum Speicher einblenden (`map`) und wieder ausblenden (`unmap`)
- zuerst Datenstrukturen für vollständiges korrektes Kernelmapping aufsetzen, dann `cr3` umschalten
- *present*-Bit von unbenutzte Einträge auf `0` setzen
  - sonst passieren wilde Dinge
  - Seiten vom Page Frame Allocator können beliebigen Inhalt haben
    → meist nur auf Hardware (Qemu/KVM mit genulltem Speicher)
- weitere relevante Bits: *user mode* und *writeable*
  - Steuerung nur über unterste Ebene (Page Table)
  - permissive Einstellung in PML4, PDP & PD wählen (`1`)

Das Paging

- Grundfunktionen zum Speicher einblenden (`map`) und wieder ausblenden (`unmap`)
- zuerst Datenstrukturen für vollständiges korrektes Kernelmapping aufsetzen, dann `cr3` umschalten
- *present*-Bit von unbenutzte Einträge auf `0` setzen
  - sonst passieren wilde Dinge
  - Seiten vom Page Frame Allocator können beliebigen Inhalt haben
    → meist nur auf Hardware (Qemu/KVM mit genulltem Speicher)
- weitere relevante Bits: *user mode* und *writeable*
  - Steuerung nur über unterste Ebene (Page Table)
  - permissive Einstellung in PML4, PDP & PD wählen (`1`)
- unbenutzte Bits auf `0` setzen
  - idealerweise ganze Tabelle eingangs mit `memset` nullen

Virtuellen Speicher für jeden Thread

- **jeder** Thread hat einen eigenen Adressraum
  - keine Page Table zwischen Threads teilen!

Virtuellen Speicher für jeden Thread

- **jeder** Thread hat einen eigenen Adressraum
  - keine Page Table zwischen Threads teilen!
- nur Kernelspace gemappt…
  - erste 64 MiB exklusive erste Seite
  - zusätzlich IOAPIC und LAPIC
  - (in dieser Aufgabe noch) für Userspace zugreifbar
  - ggf. nur lesbar (abhängig von App)

Virtuellen Speicher für jeden Thread

- **jeder** Thread hat einen eigenen Adressraum
  - keine Page Table zwischen Threads teilen!
- nur Kernelspace gemappt…
  - erste 64 MiB exklusive erste Seite
  - zusätzlich IOAPIC und LAPIC
  - (in dieser Aufgabe noch) für Userspace zugreifbar
  - ggf. nur lesbar (abhängig von App)
- …sowie eine Seite für Userstack
  - bei allen Threads die gleiche (virtuelle) Adresse
  - schreibbar

Virtuellen Speicher für jeden Thread

- **jeder** Thread hat einen eigenen Adressraum
  - keine Page Table zwischen Threads teilen!
- nur Kernelspace gemappt…
  - erste 64 MiB exklusive erste Seite
  - zusätzlich IOAPIC und LAPIC
  - (in dieser Aufgabe noch) für Userspace zugreifbar
  - ggf. nur lesbar (abhängig von App)
- …sowie eine Seite für Userstack
  - bei allen Threads die gleiche (virtuelle) Adresse
  - schreibbar
- Umschaltung des Mappings bei Kontextwechsel

Virtuellen Speicher für jeden Thread

- **jeder** Thread hat einen eigenen Adressraum
  - keine Page Table zwischen Threads teilen!
- nur Kernelspace gemappt…
  - erste 64 MiB exklusive erste Seite
  - zusätzlich IOAPIC und LAPIC
  - (in dieser Aufgabe noch) für Userspace zugreifbar
  - ggf. nur lesbar (abhängig von App)
- …sowie eine Seite für Userstack
  - bei allen Threads die gleiche (virtuelle) Adresse
  - schreibbar
- Umschaltung des Mappings bei Kontextwechsel
- Threads dynamisch (zur Laufzeit) erstellen
  - keine Annahmen bzgl. Initialisierungsreihenfolge globaler Objekte

Das NX Bit (7.5 ECTS)

- überall außer bei Kernel-`.text` setzen
  (➔ *Source Code Reference* in Linkerskript einfügen)

Das NX Bit (7.5 ECTS)

- überall außer bei Kernel-`.text` setzen
  (➔ *Source Code Reference* in Linkerskript einfügen)
- wegen `.boot` nicht zwingend ausgerichtete Adresse

Das NX Bit (7.5 ECTS)

- überall außer bei Kernel-`.text` setzen
  (➔ *Source Code Reference* in Linkerskript einfügen)
- wegen `.boot` nicht zwingend ausgerichtete Adresse
- nur in PT, nicht jedoch bei PML4, PDP & PD gesetzt

Das NX Bit (7.5 ECTS)

- überall außer bei Kernel-`.text` setzen
  (➔ *Source Code Reference* in Linkerskript einfügen)
- wegen `.boot` nicht zwingend ausgerichtete Adresse
- nur in PT, nicht jedoch bei PML4, PDP & PD gesetzt
- muss via *Extended Feature Enable Register*
  (ein *Model-Specific Register*) erst aktiviert werden
  - in MPStuBSmI auf jedem Kern
  - und zwar nur, wenn die CPU das unterstützt
    (mittels `cpuid` prüfbar)

  ➔ StuBS5 ohne NX zum testen

Das NX Bit (7.5 ECTS)

- überall außer bei Kernel-`.text` setzen
  (➔ *Source Code Reference* in Linkerskript einfügen)
- wegen `.boot` nicht zwingend ausgerichtete Adresse
- nur in PT, nicht jedoch bei PML4, PDP & PD gesetzt
- muss via *Extended Feature Enable Register*
  (ein *Model-Specific Register*) erst aktiviert werden
  - in MPStuBSmI auf jedem Kern
  - und zwar nur, wenn die CPU das unterstützt
    (mittels `cpuid` prüfbar)
  - ➔ StuBS5 ohne NX zum testen
- **Fehler resultieren meist in Bootschleife**

Die ewigen Klassiker

- mehrfache Deallokation (*double free*)

Die ewigen Klassiker

- mehrfache Deallokation (*double free*)
- fehlende Referenzierung bei *Source Code Reference*
  - kein `&` bei `___KERNEL_START___` (aus Linkerskript)
    mit `extern "C" void * ___KERNEL_START___`

Die ewigen Klassiker

- mehrfache Deallokation (*double free*)
- fehlende Referenzierung bei *Source Code Reference*
  - kein `&` bei `___KERNEL_START___` (aus Linkerskript)
    mit `extern "C" void * ___KERNEL_START___`
- fehlerhafte Referenzierung einer Variable
  - `&idleThread` bei `PerCore<IdleThread*> idleThread`
  - `&userStack` bei `char * userStack`
    (geändert von vormals `char userStack[4096]`)

Die ewigen Klassiker

- mehrfache Deallokation (*double free*)
- fehlende Referenzierung bei *Source Code Reference*
  - kein `&` bei `___KERNEL_START___` (aus Linkerskript)
    mit `extern "C" void * ___KERNEL_START___`
- fehlerhafte Referenzierung einer Variable
  - `&idleThread` bei `PerCore<IdleThread*> idleThread`
  - `&userStack` bei `char * userStack`
    (geändert von vormals `char userStack[4096]`)
- versehentlich Bits gesetzt
  - Speicher nicht genullt (`memset`)
  - oder `reserved = 1` (statt 0)

Die ewigen Klassiker

- mehrfache Deallokation (*double free*)
- fehlende Referenzierung bei *Source Code Reference*
  - kein `&` bei `___KERNEL_START___` (aus Linkerskript)
    mit `extern "C" void * ___KERNEL_START___`
- fehlerhafte Referenzierung einer Variable
  - `&idleThread` bei `PerCore<IdleThread*> idleThread`
  - `&userStack` bei `char * userStack`
    (geändert von vormals `char userStack[4096]`)
- versehentlich Bits gesetzt
  - Speicher nicht genullt (`memset`)
  - oder `reserved = 1` (statt 0)
- zu neue Compiler
  - Gcc 12 (veröffentlicht nach Semesterbeginn Anfang Mai)

…und nun weiter mit Aufgabe 4

…und nun weiter mit Aufgabe 4
(wie viel schlimmer kann es denn noch werden?)