Mikro-Kernel
kann man das essen ?

Marius
– Informatiker auf halbem Wege zur Erleuchtung

WTF

Ein Konzept zur Strukturierung und Organisation von Betriebssystemen.

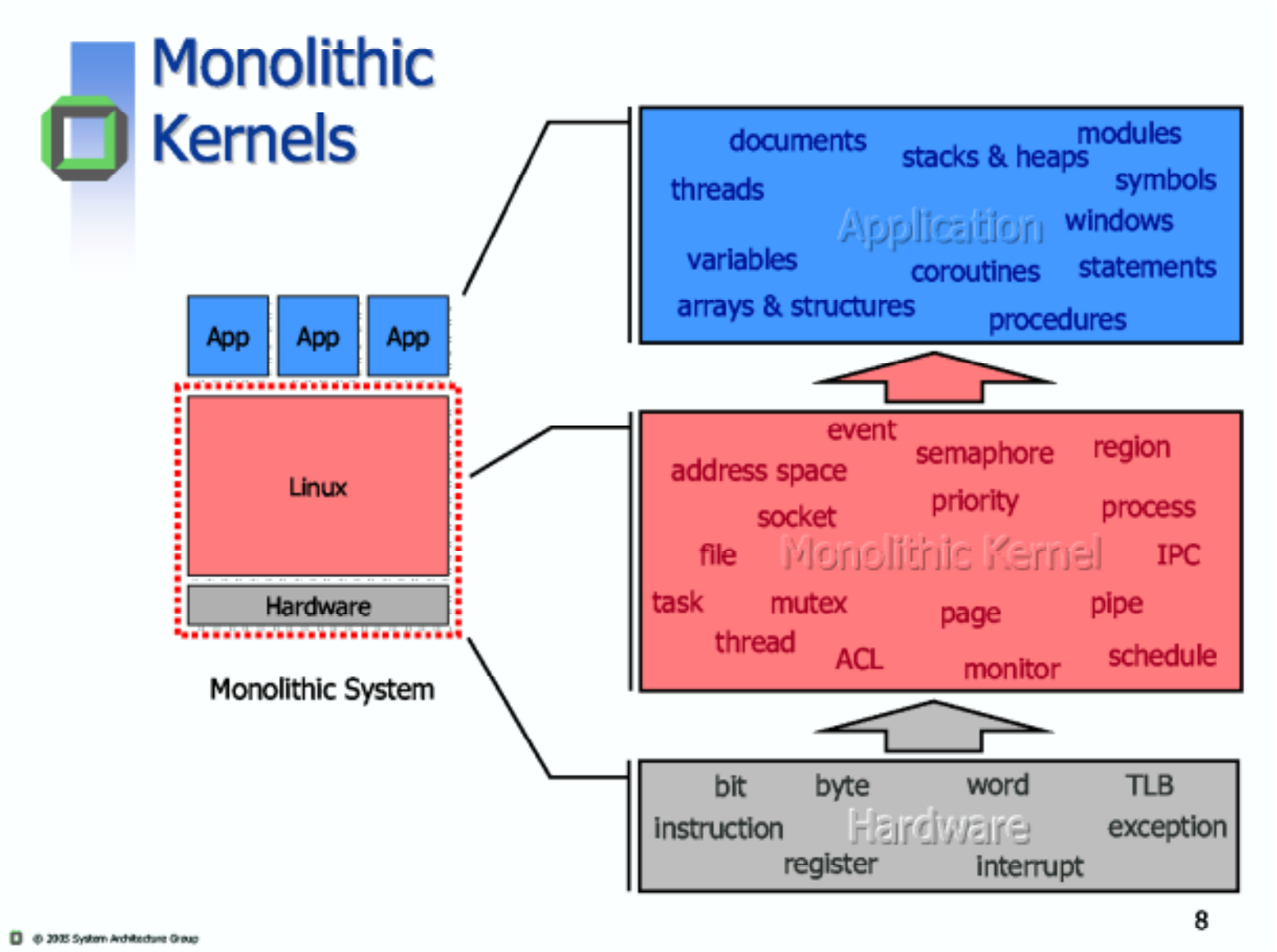
"Erfunden" ~ 1990

state-of-the-art in system architecture

Motivation

- Wer benutzt MacOS X ?
- Wer kennt Mach, wer Hurd ?
- Debatte Tanenbaum/Torvalds zu frühen Linux-Zeiten

... man sollte davon mal gehört haben



Struktur im Detail

Applikation Applikation

glibc Compiler ...

user-level

---

kernel-level vfs tcp

vmm ext2 reiserfs ipv4

psmouse ide scsi tulip

MMU I/O-Hardware

kernel-level

- Läuft im privilegierten Modus der CPU, wegen:

  - Zugriff auf MMU für virtuellen Speicher
  - Maßnahmen beim context-switch zwischen Prozeßen
  - Interrupt-Handler im priv-mode

kernel-level(2)

Danger !!!!!!
Darf alles !!!!!

Rechnerabsturz bei **kernel panic**,
im schlimmsten Falle
Datenverlust.

Ganz schlimm: Pleite / Absturz

kernel-level(3)

Kernel ist ein "blob",
keine Grenzen zwischen Treibern

- Bug in maustreiber überschreibt FS-Daten
- böses Modul ändert uids
- "krepierter" Pointer kann potentiell **alles** beeinflussen
- Seiteneffekte teils gewollt ...

kernel-level(4)

Kernel laufen lassen => Vertrauen haben.
binary-only-Module ?!
Wer versteht 3Mio Codezeilen ?!

Bsp Windows: GUI im Kernel.
Server stirbt wg. Bug im Start-Icon

Unix: X11 im user-level,
worst-case: Grafik "bunt", aber System noch da

Konsequenz

Alles raus aus dem Kernel !!

Was braucht man dort:

- virtueller Speicher (MMU)
- Prozeß-Verwaltung (contextswitch)
- Interrupt-Handling

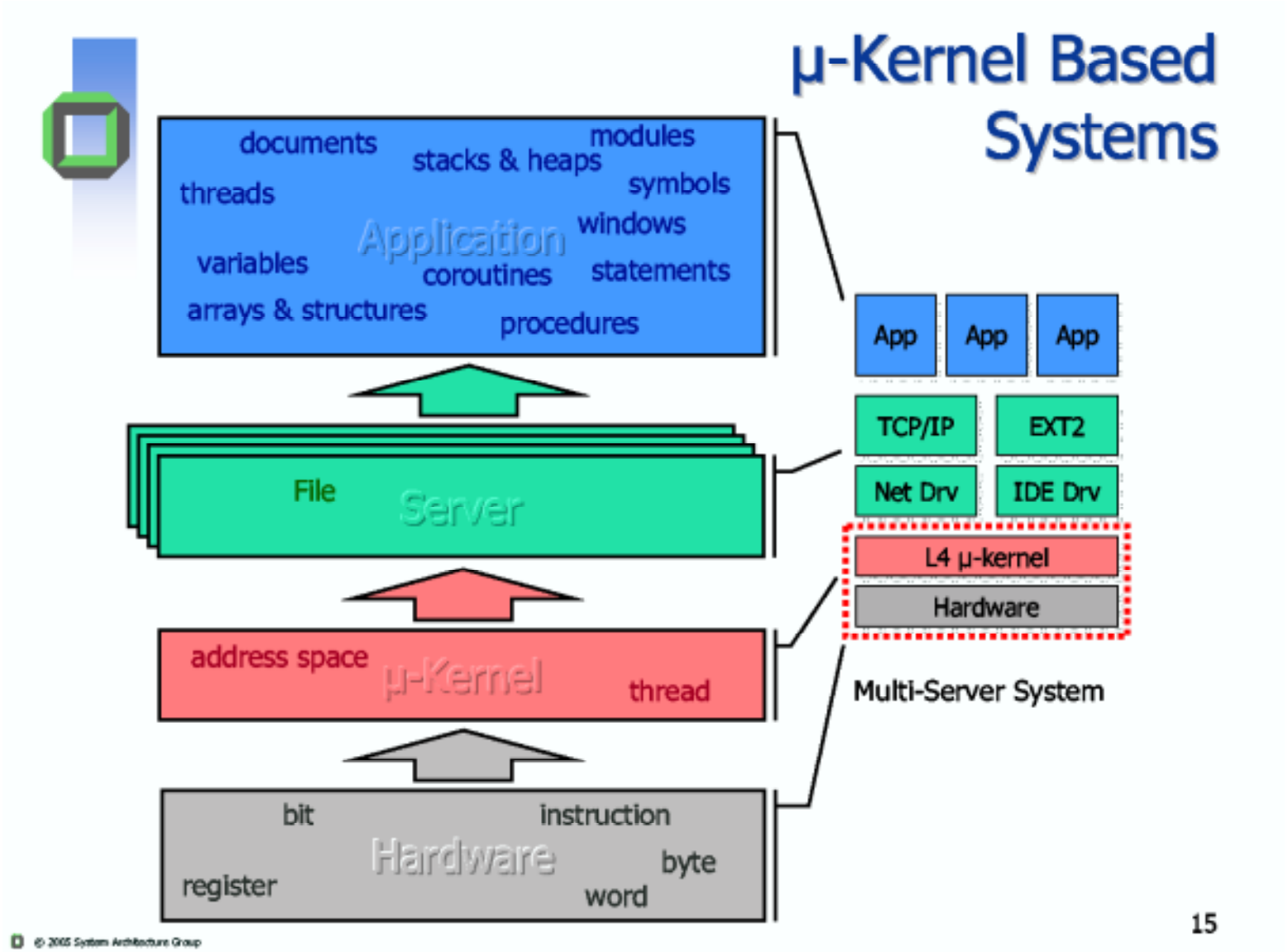
Alles andere im user-level

Mikrokernel-Konzept

µKernel bietet:

- (virtuelle) Adreßräume
- Threads (mit Kontext)
- IPC (Komm. zwischen Threads)

Das wars !!



Virtueller Speicher

Kernel verwaltet die Adreßräume, pager im user-level definieren sie.

Bsp Pagefault eines Threads:

- Kernel sendet IPC an pager
- pager erkennt: PF war im Code-Bereich
- pager lädt page vom FS-Server in eigenen Adreßraum
- spezieller IPC mappt den Bereich vom pager-AS in den AS des unterbrochenen Threads
- applikation kann weiterlaufen

Interrupts

Bsp:

- HW setzt interrupt-flag
- Kernel landet im interrupt-handler
- "interrupt-thread" sendet IPC an registrierten driver-thread
- dieser schreibt nächstes byte in LPT-buffer
- antwortet ACK per IPC an interrupt-thread
- kernel erzeugt Interrupt-ACK für HW

Struktur des Systems

- Einzelne Module (n threads in m ASs) kooperieren
- haben jeweils definierte Aufgabe
- fest definierte Schnittstelle
- keine Seiteneffekte
- einzeln austauschbar
- einfacher zu pflegen

Bsp: tcp, ip, ethernet; vfs, ext2, disk

Pro/Contra

- Schutz durch getrennte ASs
- leichtere Code-Pflege
- µK als trusted-base **klein**
- HW-Fummelei schön abstrahiert

- Kommunikation statt prozedur-aufruf

Mach

Mikrokernel 1. Generation – Konzept nur ansatzweise umgesetzt.
Kommunikation kriecht
t(IPC) > 100µs

- Nur ein Pager für alle Threads
- Treiber noch im Kernel
- Code ästhetisch statt schnell

"Damals"

Als OS für GNU-Projekt angedacht:
Hurd auf Mach-Basis (hurd of servers)
mit aktuellem Konzept

Minix als Lehrbetriebssystem
ebenfalls µK-basiert

Linux Bastelei eines Studenten mit
monolithischem Ansatz

Darwin

- Kernel von MacOS X
- Basiert auf Mach

Offensichtlich "schnell genug"

Schneller !!!

IPCs verrichten keine Arbeit
=> ideal wäre t(IPC) = 0

µKernel hat wenig Funktionalität
=> wenig Code

IPCs unterbrechen Code-Fluss
=> idealerweise wenig
cache-pollution (auch TLB)

L4 (l4ka.org)

siehe Papers Prof. Liedtke

Ergebnis ist 2nd-Generation µK L4

Aktuelle Implementierung:
L4Ka::Pistacchio in 10.000 Zeilen C++/asm
auf alpha, amd64, arm, ia32, ia64, ppc32/64, mips64

Was tun mit L4

Hurd wird auf L4 portiert

Diverse VM-Geschichten

Praktikum in meinem Studium

siehe l4ka.org und l4hq.org