Signale

PD Dr. Reinhard Bündgen
buendgen@de.ibm.com

Signale in Unix (POSIX)

- Kommunikation mit Prozessen
  - Prozess zu Prozess
  - Kern zu Prozess
  - wenig Information: kl. Zahl: 1- 64 (etwas Verursacherinfo)
- Signale
  - siehe `<asm/signal.h>`
  - `man signal.h`, or `man 7 signal` or `/usr/include/`*`architecture`*`/bits/signum.h`
  - z.B. HUP, INT, KILL, SEGV
  - reguläre Signale, RT Signale
- Signalbehandlung
  - asynchron
  - durch Kern (ignore, stop, continue, terminate, dump)
  - durch Prozess: Signalbehandlungsroutine

SIGNAL User Space Interface

- Kommando: `kill` sendet Signal
- API
  - `kill, tkill(), tgkill()`
    - sende Signal an Prozess (thread group), Thread
  - `sigaction(), signal()`
    - installiere Signalbehandlungsroutine
  - `sigprocmask()`
    - ändere Menge der zu blockierenden Signale
  - `sigpending(), sigsuspend()`
    - teste, warte auf Signal
  - `sigqueue()`
    - Sendet RT Signal
  - `rt_sigaction, rt_sigpending, rt_sigprocmask,...`
    - RT Varianten

Signaleigenschaften

- Signalbehandlung
  - Defaulthandler – vom Kern festgelegt
    - Terminate, dump, ignore, stop, continue
  - vom User Space abfangbar/überschreibbar
    - durch User Space Handler oder
    - SIG_IGN
- blockierbar: temporäre Maskierung
- maskierbar: während Signalbehandlung
- Signalwarteschlangen (queueing) für Signale des gleichen Typs nur für RT Signale (>31) möglich
- benötigt Autorisierung (capabilities)
- Anzahl ausstehender Signale/Prozess begrenzt

Signale für multithreaded Prozesse nach POSIX 1003.1

- Signal Handler gelten prozessweit (shared)
- Jeder Thread hat eigene Masken für pending und blockierte Signale
- Kern, kill() und sigqueue() senden Signale an einen Prozess, Behandlung durch einen beliebigen das Signal nicht blockierenden Thread
- fatale Signale terminieren alle Threads eines Prozesses
  - Fatale Signale: `SIGKILL` oder `SIG_DFL` = Terminate (Dump)

- private Signale: an einzelnen Prozess gesendet
- shared Signale: an Prozessgruppe gesendet

Kernel Aufgaben

- Signal-Erzeugung
  - markiere eine Task, dass sie ein Signal erhalten hat
  - speichere signal pending info in `task_struct`
  - teste ob Signal blockiert ist, wenn nein
    - setze `TIF_SIGPENDING` flag in `thread_info->flags`
    - evtl. wecke Task aus `TASK_INTERRUPTIBLE` oder `TASK_STOPPED` Status auf.
- Signal-Auslieferung
  - Bei jedem Übergang vom Kern zum User Space
  - lasse Task das Signal behandeln
    - durch ignorieren
    - durch default Handler
    - durch User Space Handler

Kernel Datenstrukturen

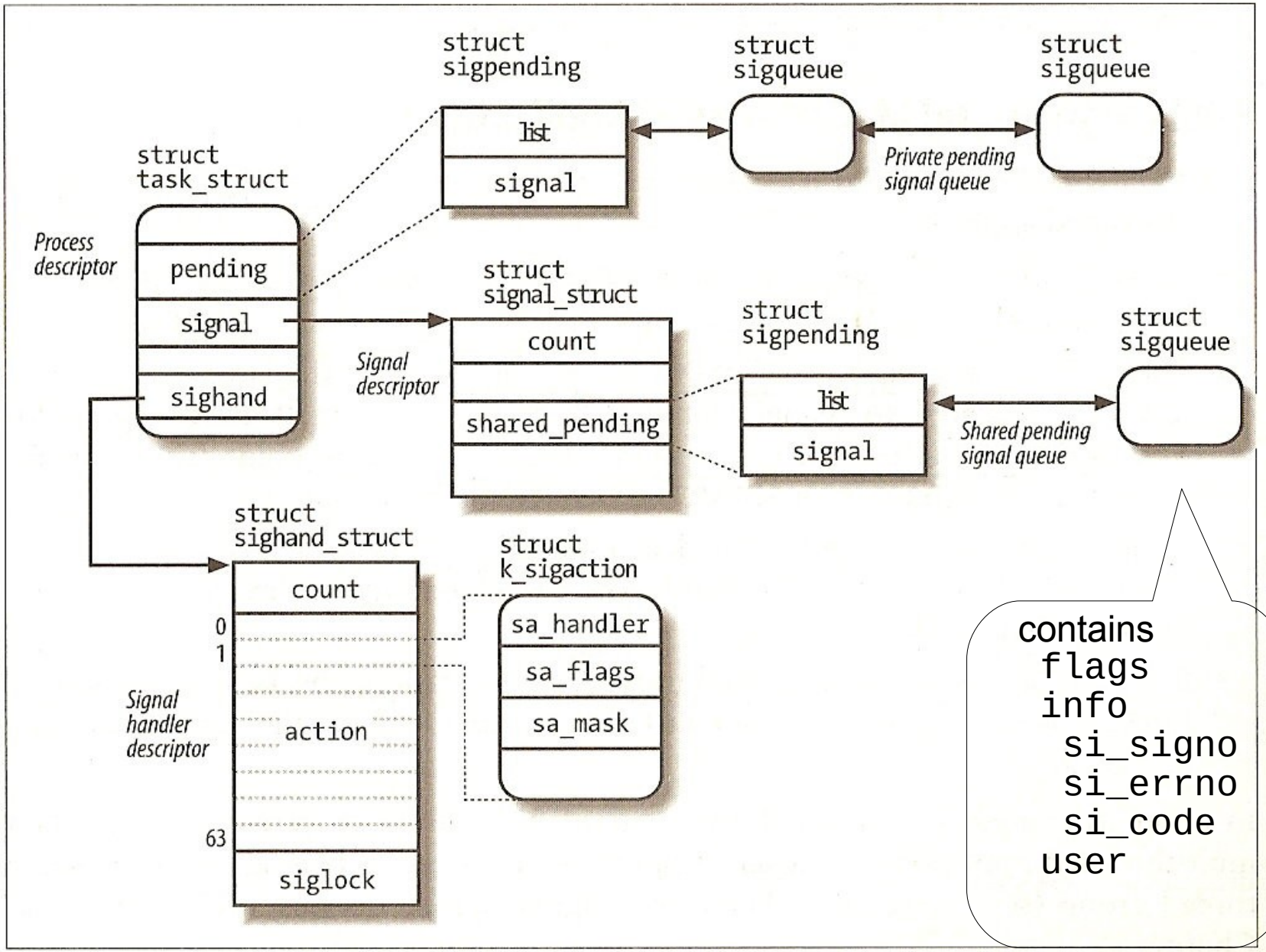
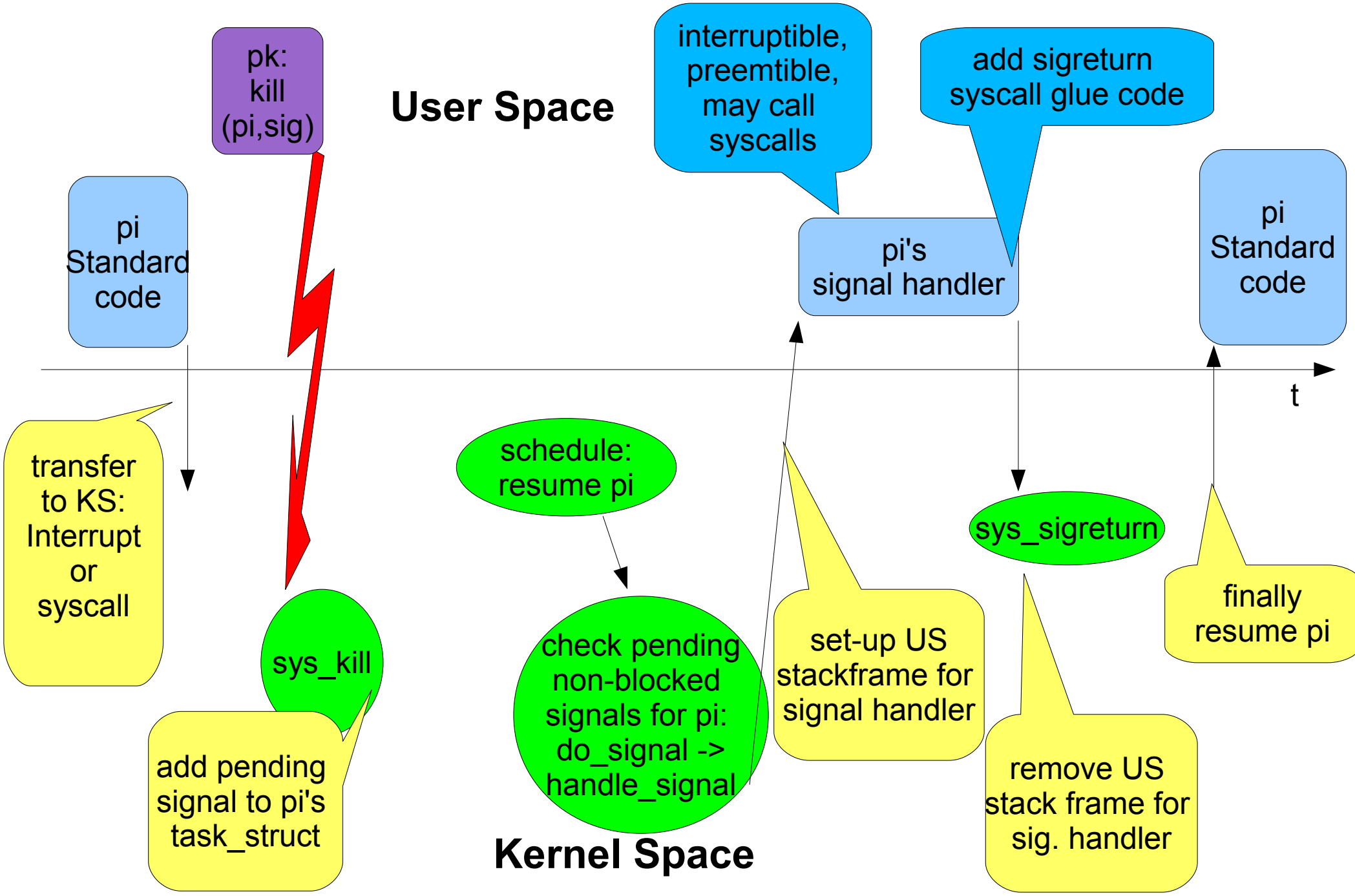


Figure 11-1. The most significant data structures related to signal handling

Signal Datenstrukturen

- `k_sigaction` (`<asm-generic/signal.h>`)
  - siehe auch `sigaction()`
  - `sa_handler`:
    - Funktionszeiger oder `SIG_DFL` oder `SIG_IGN`
  - `sa_flags`
    - z.B. `SA_NOCLDSTOP`, `SA_RESTART`, `SA_RESETHAND`
  - `sa_mask`
    - Signale, die während der Signalbehandlung maskiert sind
- `siginfo_t` (`<asm-generic/siginfo.h>`)
  - `si_signo`: Signalnummer
  - `si_errno`: Fehlernummer, falls Signal durch Fehler erzeugt
  - `si_code`: Absender
    - `SI_USER`, `SI_KERNEL`, `SI_QUEUE`, `SI_TIMER`, `SI_ASYNCIO`, `SI_TKILL, ...`
  - `_sifields`: signalabhängige Information

User Space Signal Handler



Unterbrechung von Systemrufen durch Signale

- Situation: Systemruf im Zustand TASK_INTERRUPTIBLE wird von Signal unterbrochen (d.h. geweckt)
  - Option 1: nach Beendigung des Signalhandlers endet der Systemruf mit EINTR
  - Option 2: nach Beendigung des Signalhandlers wird der Systemruf wiederholt
- Welche Option zutrifft hängt vom unterbrochenen Systemruf und vom SA_RESTART flag des Signal Handlers ab