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`, `/usr/include/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
  - `sigqeue()`
    - sende RT signal
  - `rt_sigaction, rt_sigpending, rt_sigprocmask,...`
    - RT Varianten

Signaleigenschaften

- Signalbehandlung
  - Defaulthandler – vom Kern pro Signal festgelegt
    - terminate, dump, ignore, stop, continue
  - vom User Space abfangbar/überschreibbar durch
    - User Space Signal Handler oder
    - SIG_IGN
  - SIGKILL, SIGSTOP können nicht ignoriert, blockiert oder abgefangen werden (ausser für Prozesse 0 und 1)
- blockierbar: temporäre Maskierung
- maskierbar während Signalbehandlung
- Signalqueueing nur für RT Signale (>31) möglich
- benötigt Authorisierung (capabilities)
- Anzahl ausstehendender Signale/Prozess begrenzt

Signale für multithreaded Prozesse nach POSIX 1003.1

- Signal Handler gelten prozessweit (shared)
- Jeder Thread hat eigene Masken für pending ond blockierte Signale
- Kern und kill() und sigqueue() senden Signale an einen Prozess, Behandlung durch einen einzigen, beliebigen das Signal nicht blockierenden Thread
- fatale Signale terminieren alle Threads eines Prozesses
  - fatale Sgnale: SIGKILL oder Handler ist SIG_DFL = Terminate,
- private signal: an einzelnen Prozess gesendet
- shared signal: 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 Kernel zum User Space
  - lasse Task nicht blockierte Signale behandeln
    - durch ignorieren
    - durch default Handler
    - durch User Space Handler

Kernel Datenstrukturen

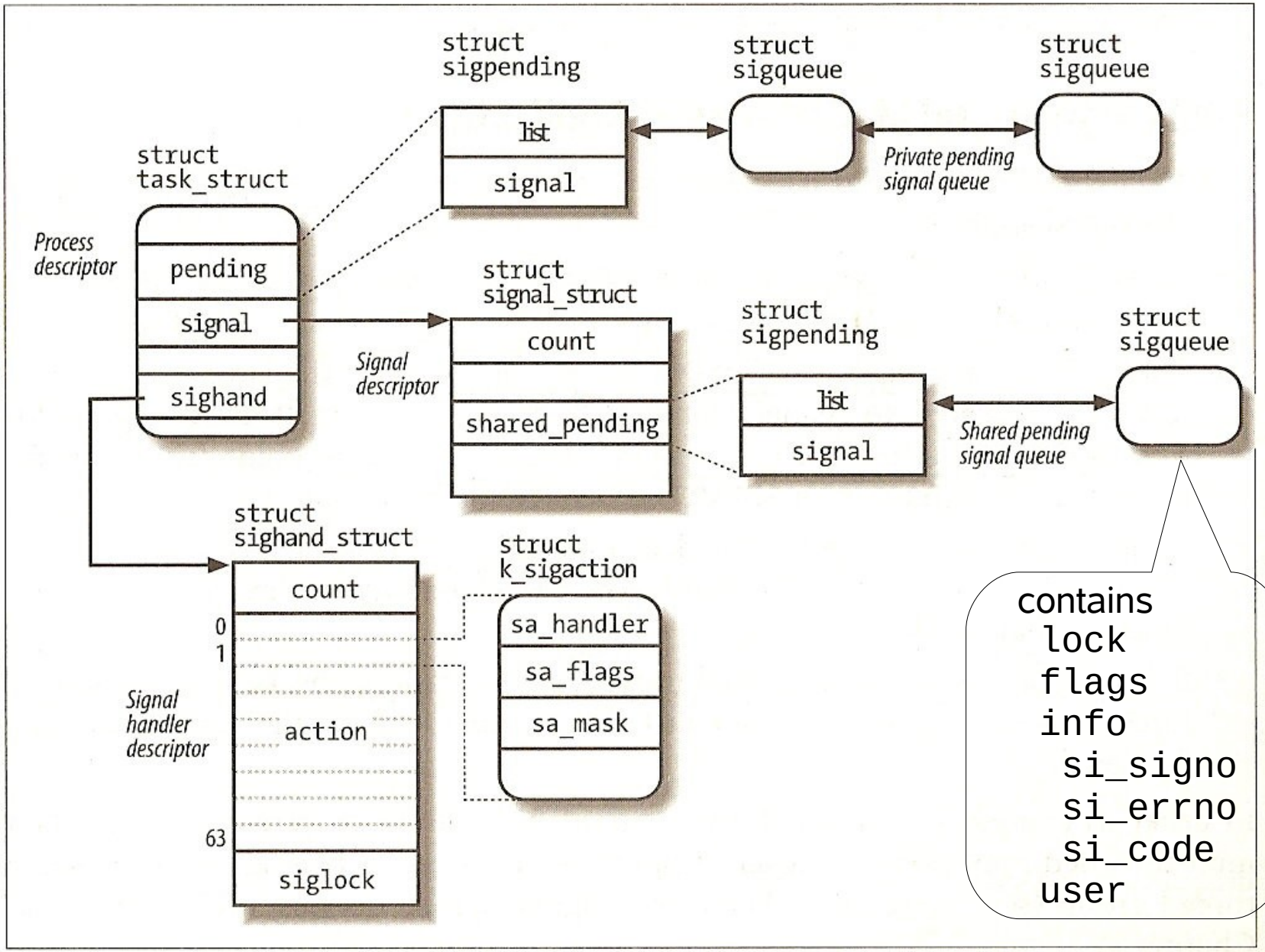
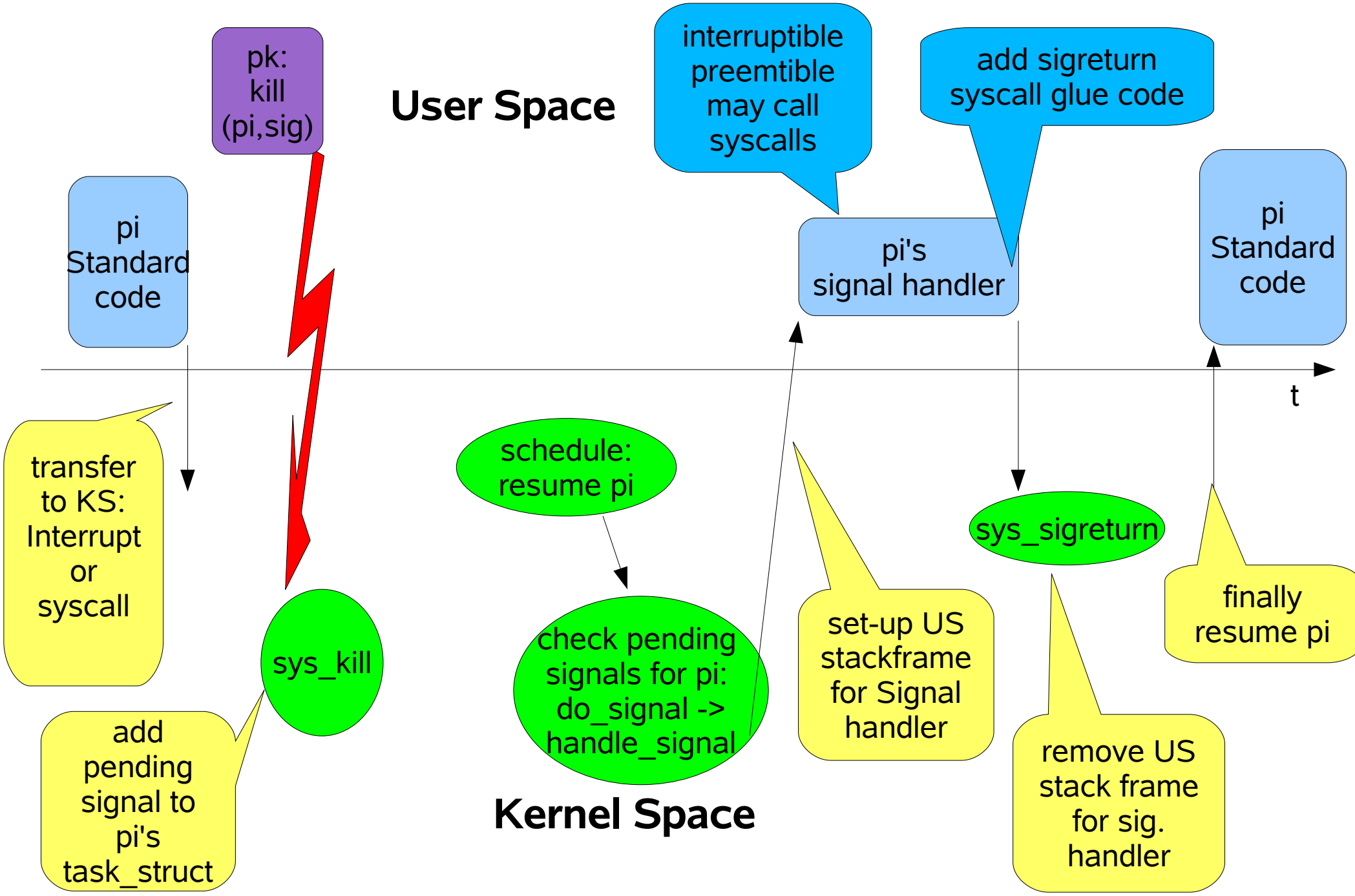


Figure 11-1. The most significant data structures related to signal handling

Signal Datenstrukturen

- `k_sigaction` (siehe auch `sigacton()`)
  - `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`
  - `si_signo:` Signalnummer
  - `si_errno:` Fehlernummer, Falls Signal durch Feler erzeugt
  - `si_code:` Absender
    - `SI_USER, SI_KERNEL, SI_QUEUE, SI_TIMER, SI_ASYNCIO, SI_TKILL`
  - `_sifields:` signalabhängige Information

User Space Signal Handler



Unterbrochene Systemrufe

- Task in Zustand TASK_INTERRUPTIBLE
  - wartet
  - kann durch Signal geweckt werden
- wenn Task wartet in Systemruf und wird durch Signal geweckt
  - unterbrochener Systemcall wird nicht zu Ende geführt, Kern kann
    - seine Ausführung wiederholen oder
    - EINTR zurückgeben