FreeBSD 勉強会

Jail 機構と資源制御

佐藤　広生 <hrs@FreeBSD.org>

東京工業大学/ FreeBSD Project

2012/6/8

講師紹介

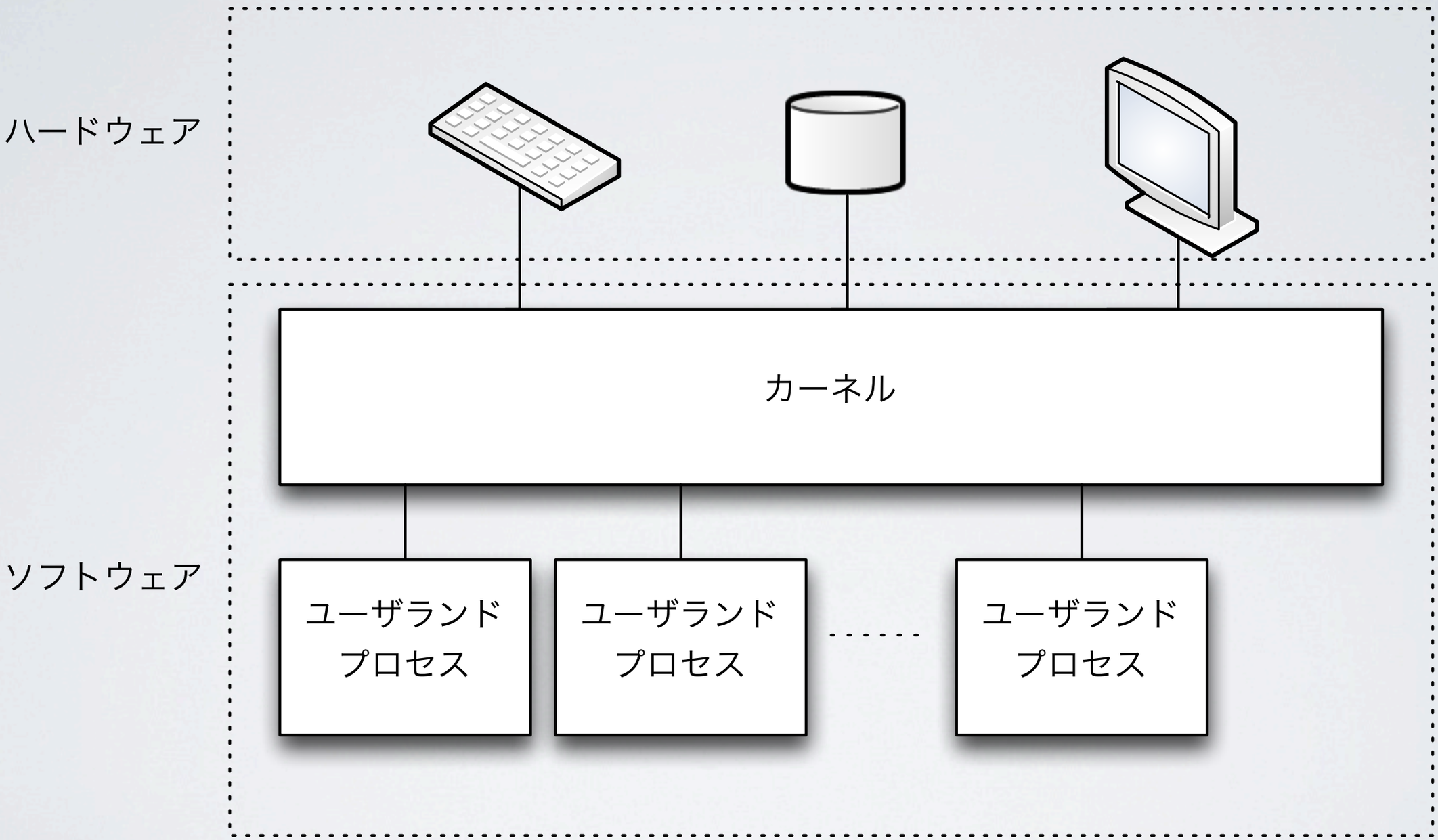
佐藤 広生 <hrs@FreeBSD.org>

- *BSD関連のプロジェクトで10年くらい色々やってます
- カーネル開発・ユーザランド開発・文書翻訳・サーバ提供などなど
- FreeBSD コアチームメンバ(3期目)、リリースエンジニア (commit 比率は src/ports/doc で 1042/957/1179)
- AsiaBSDCon 主宰
- 技術的なご相談や講演・執筆依頼は hrs@allbsd.org まで

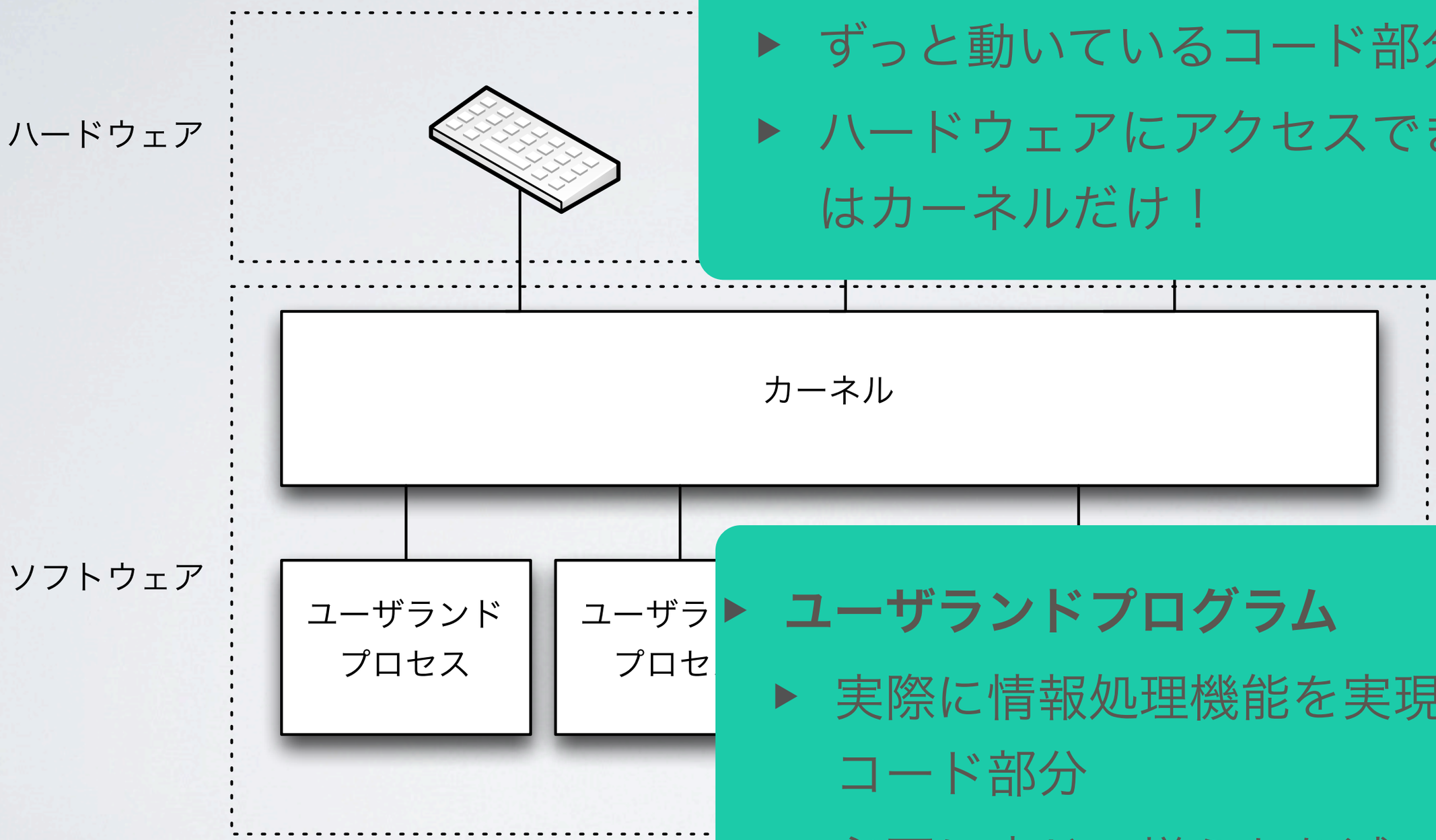
お話すること

- **基礎知識**
- **資源の使用量監視の枠組み**
  - UNIX系OSが備えているもの
- **資源の分離**
  - 原理と使える道具
  - Jailとは何ぞや
- **資源の分離と使用量制御のフレームワーク**
  - RACCT / RCTL

復習：UNIX系OSの基本構造



復習：UNIX系OSの基本構造

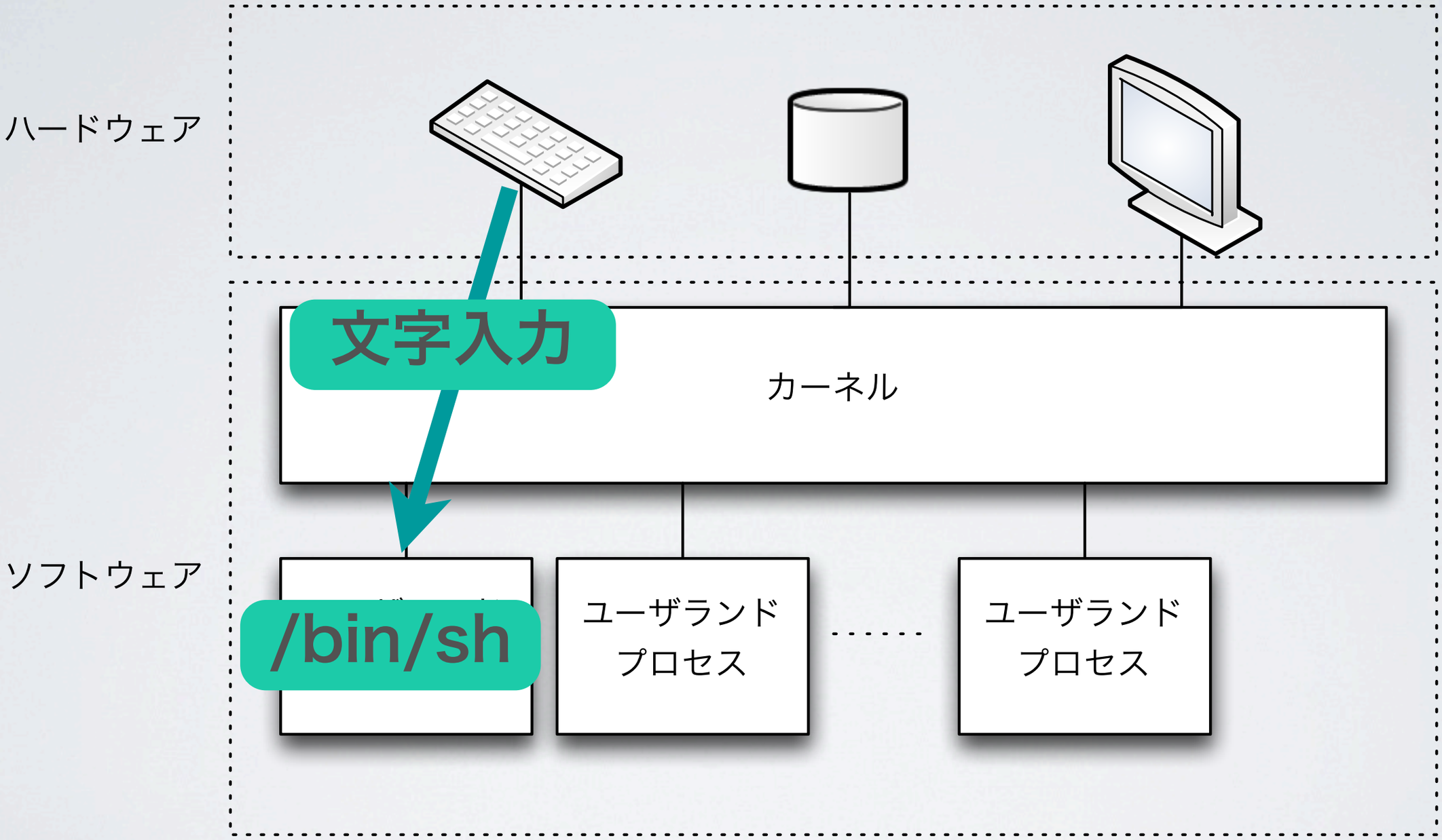


- **カーネルプログラム**
  - ずっと動いているコード部分。
  - ハードウェアにアクセスできるのはカーネルだけ！

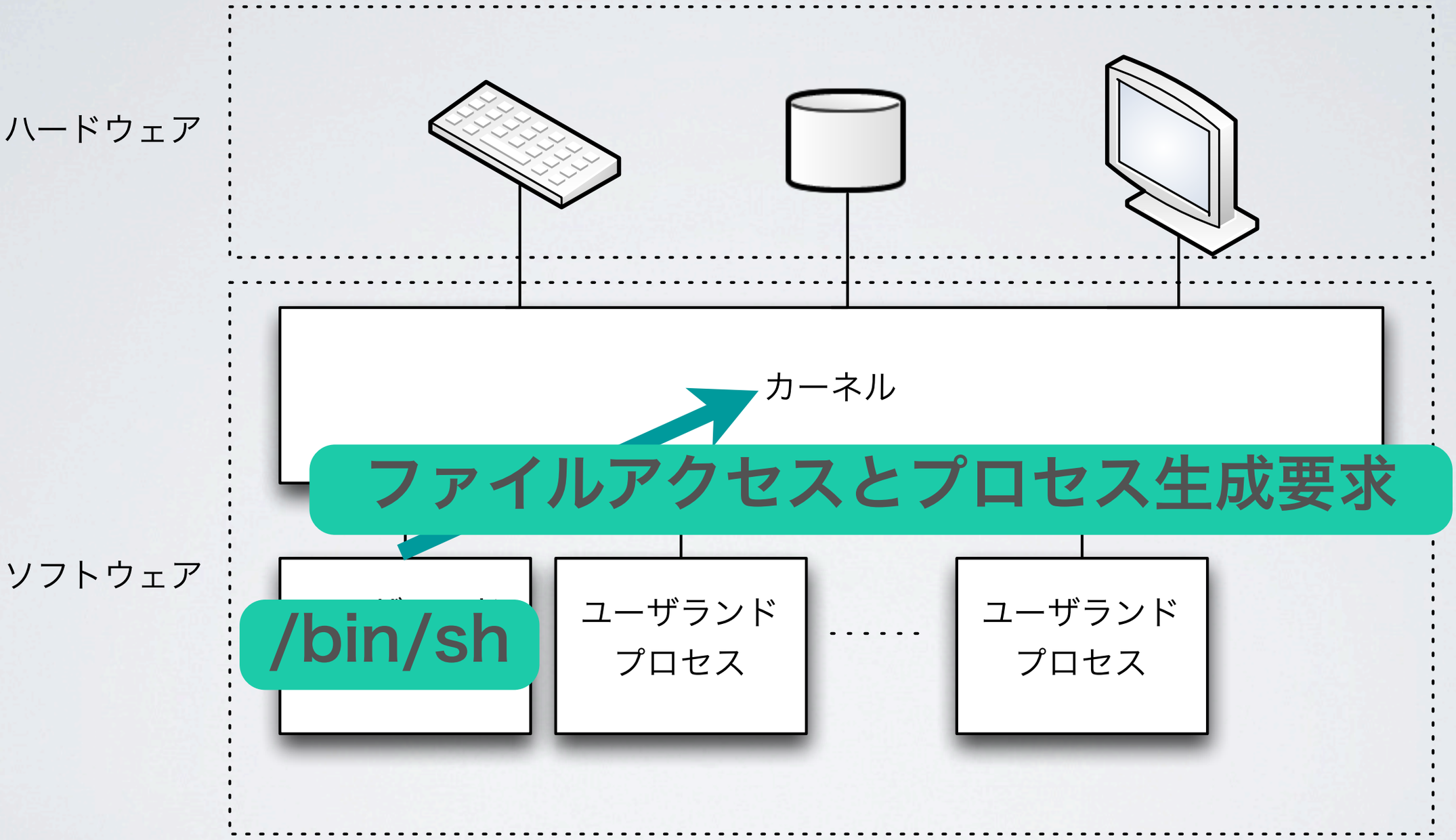
- **ユーザランドプログラム**
  - 実際に情報処理機能を実現するコード部分
  - 必要に応じて増えたり減ったり

復習：UNIX系OSの基本構造

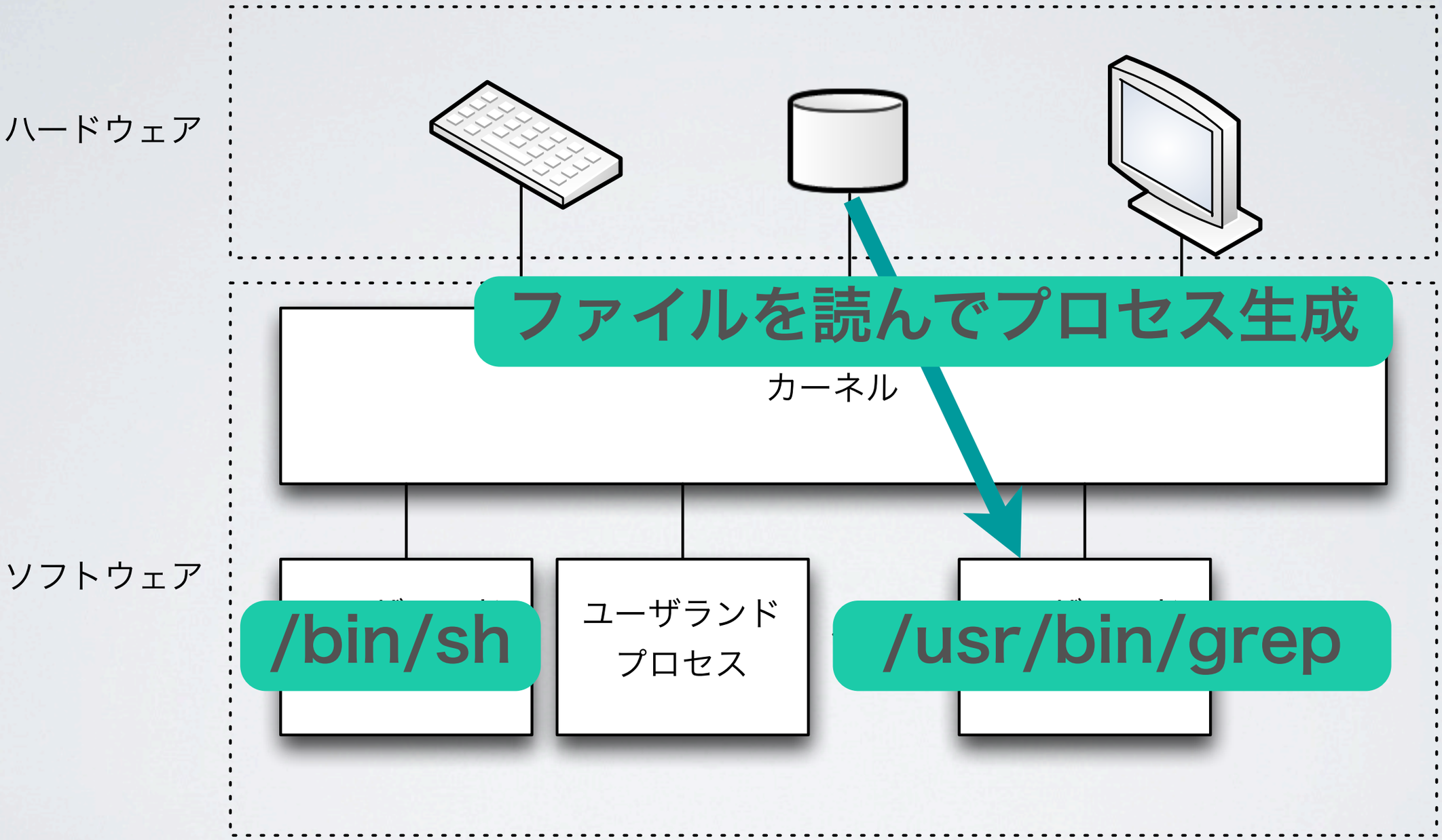
復習：UNIX系OSの基本構造



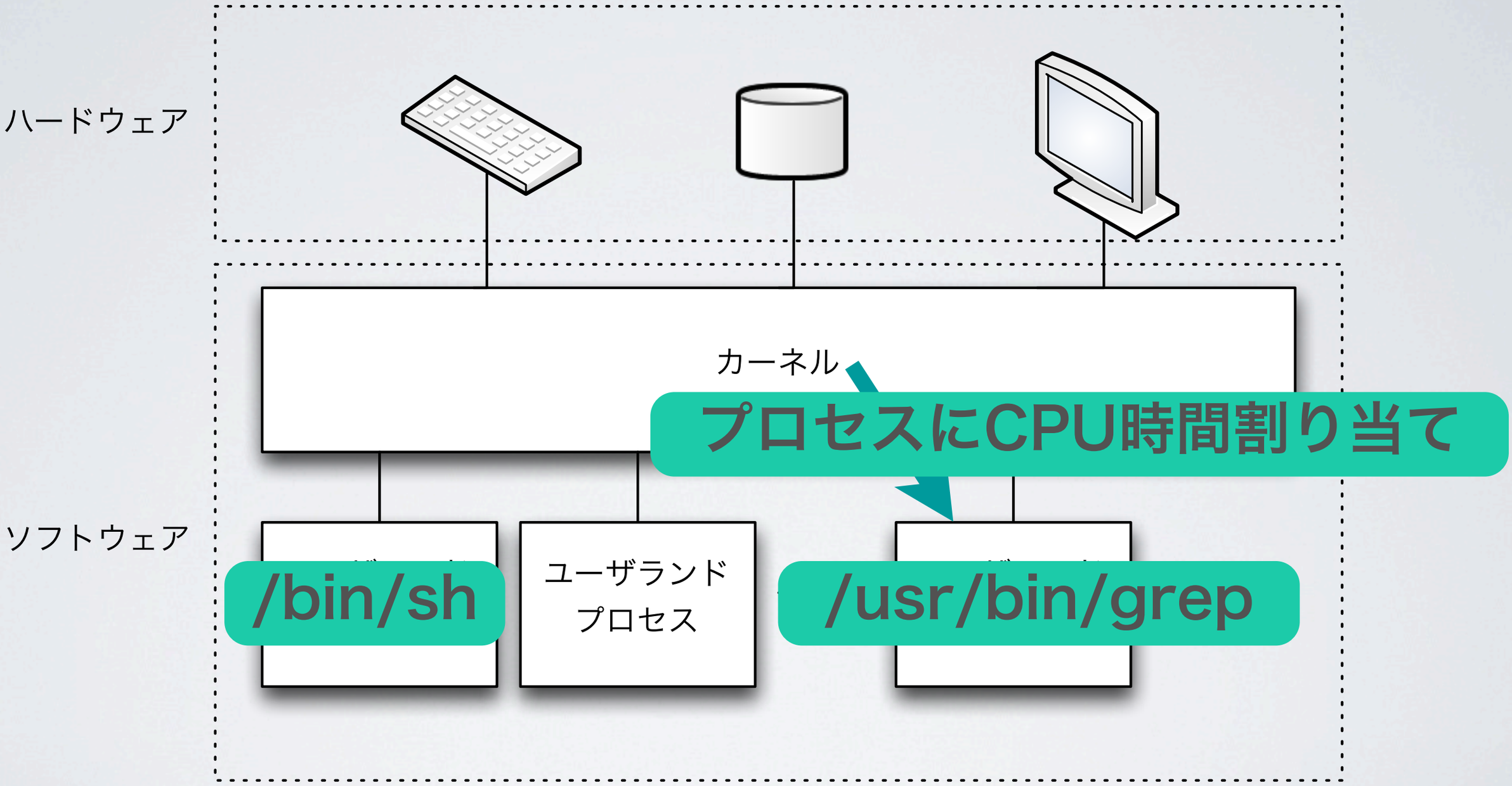
復習：UNIX系OSの基本構造



復習：UNIX系OSの基本構造



復習：UNIX系OSの基本構造



復習：UNIX系OSの基本構造

- **カーネルプログラム**
  - ユーザランドプログラムの監視役
  - マシンの資源（ハードウェア等）を管理

復習：UNIX系OSの基本構造

- **カーネルプログラム**
  - ユーザランドプログラムの監視役
  - マシンの資源（ハードウェア等）を管理
- **ユーザランドプログラム**
  - 実際に情報処理機能を実現するコード部分
  - カーネルが時間を公平に調整して、
    たくさんのプログラムをちょっとずつ実行
  - マシンの資源にアクセスしたい時は
    カーネルにお願い（システムコール）
  - 何やっても自由。やっちゃいけないことをやろうとする
    と、カーネルが実行を止める

復習：UNIX系OSの基本構造

- **マシン資源とは**
  - UNIXにおける資源は、「ファイル」が基本
    - ディスクに対応する /dev/da*
    - 文字入出力に対応する /dev/tty*
    - ...
  - ファイルじゃないのもたくさんある
    - CPU時間、メモリ、IPC メッセージキュー、...

復習：UNIX系OSの基本構造

- マシン資源とは
  - UNIXにおける資源は、「ファイル」が基本
    - ディスクに対応する /dev/da*
    - 文字入出力に対応する /dev/tty*
    - ...
  - ファイルじゃないのもたくさんある
    - CPU時間、メモリ、IPC メッセージキュー、...

基本的にはカーネルが全部把握していて、
ユーザランドプログラムに使用権を与えている

復習：UNIX系OSの基本構造

- **一台の物理マシンを共用しよう**
  - UNIX＝でかいマシンをみんなで共用できるとこが利点
  - マシンの資源をどうやって共用する？

復習：UNIX系OSの基本構造

- **一台の物理マシンを共用しよう**
  - UNIX＝でかいマシンをみんなで共用できるとこが利点
  - マシンの資源をどうやって共用する？
  - 管理単位は「ユーザ」「グループ」
  - 異なるユーザは、互いに資源の使用権が分離されている
    例：他のユーザのファイルは勝手に消したりできない！

復習：UNIX系OSの基本構造

- **一台の物理マシンを共用しよう**
  - UNIX＝でかいマシンをみんなで共用できるとこが利点
  - マシンの資源をどうやって共用する？
  - 管理単位は「ユーザ」「グループ」
    - 異なるユーザは、互いに資源の使用権が分離されている
      例：他のユーザのファイルは勝手に消したりできない！
- 資源とUID, GIDの組み合わせでアクセス権限をルール化
  - 資源＝ファイル
  - ファイルの許可属性を使おう

復習：UNIX系OSの基本構造

- **ユーザ単位で分けられている資源**
  - ユーザランドプロセス空間
  - 使用中のハードウェア資源（≒ /dev/* のファイル）
- 互いに独立していて、相互干渉しない

復習：UNIX系OSの基本構造

- **ユーザ単位で分けられている資源**
  - ユーザランドプロセス空間
  - 使用中のハードウェア資源（≒ /dev/* のファイル）
- 互いに独立していて、相互干渉しない

- **カーネルの管理戦略**
  - **資源は、ルールと限界が許す限り早い者勝ち**で分配
    - ディスク容量、メモリ、CPU時間、ディスクI/O、ネットワーク送受信、etc
  - **ムダが少なくなるように**がんばる（不要資源回収など）

復習：UNIX系OSの基本構造

- **ユーザ単位で分けられているもの**
  - ユーザランドプロセス空間
  - ハードウェア資源（≒ /dev/* のファイル）
- 互いに独立していて、相互干渉しない
- **カーネルの管理戦略**
  - **資源は、ルールと限界が許す限り早い者勝ち**で分配
    - ディスク容量、メモリ、CPU時間、ディスクI/O、ネットワーク送受信、etc
  - **ムダが少なくなるように**がんばる（不要資源回収など）

「資源使用量を細かく管理しよう」という考え方は乏しい

資源の使用量管理

資源の使用量管理 =

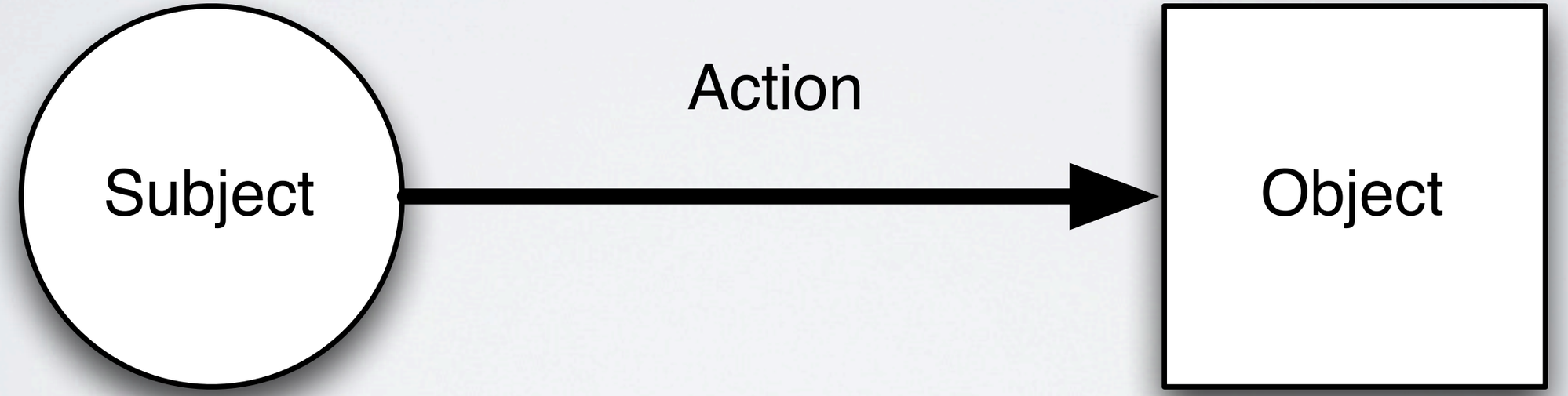
（1）資源へのアクセス権限管理

（2）資源の使用量監視

... が必要

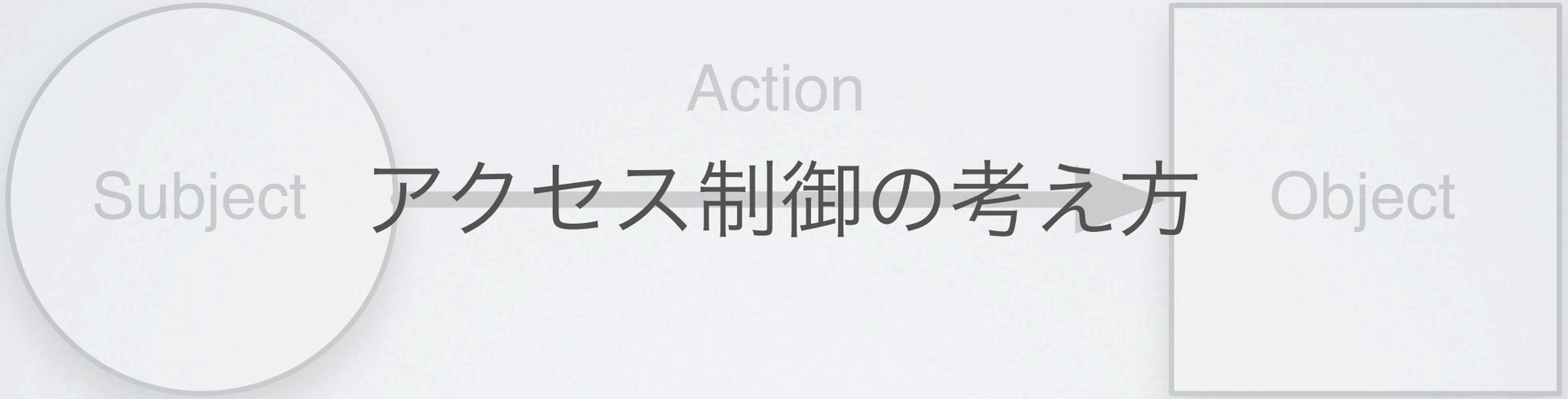
資源の使用量管理

- (1) 資源へのアクセス権限の管理



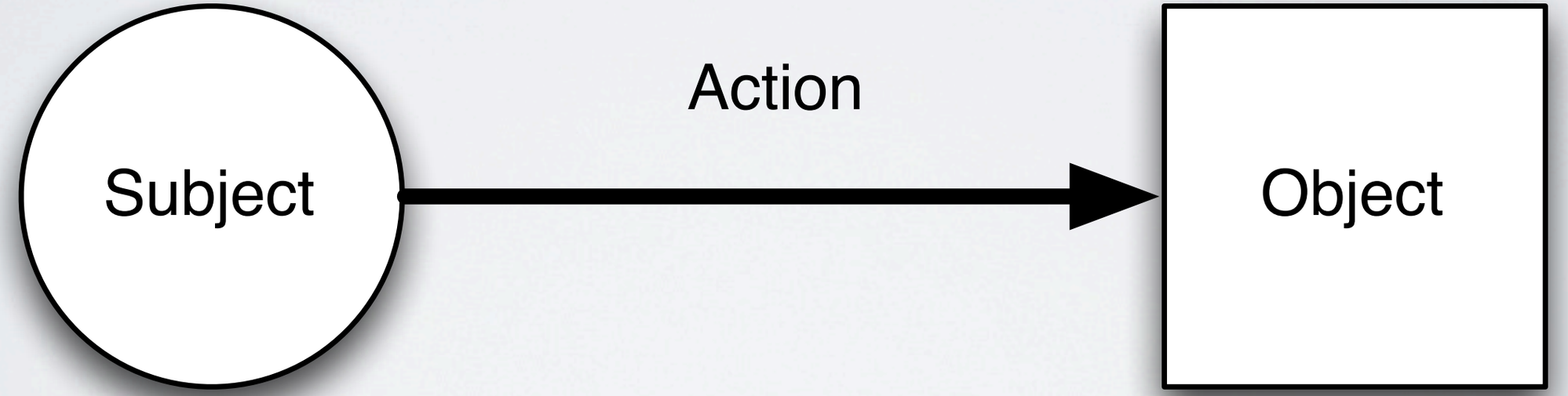
資源の使用量管理

- (1) 資源へのアクセス権限の管理



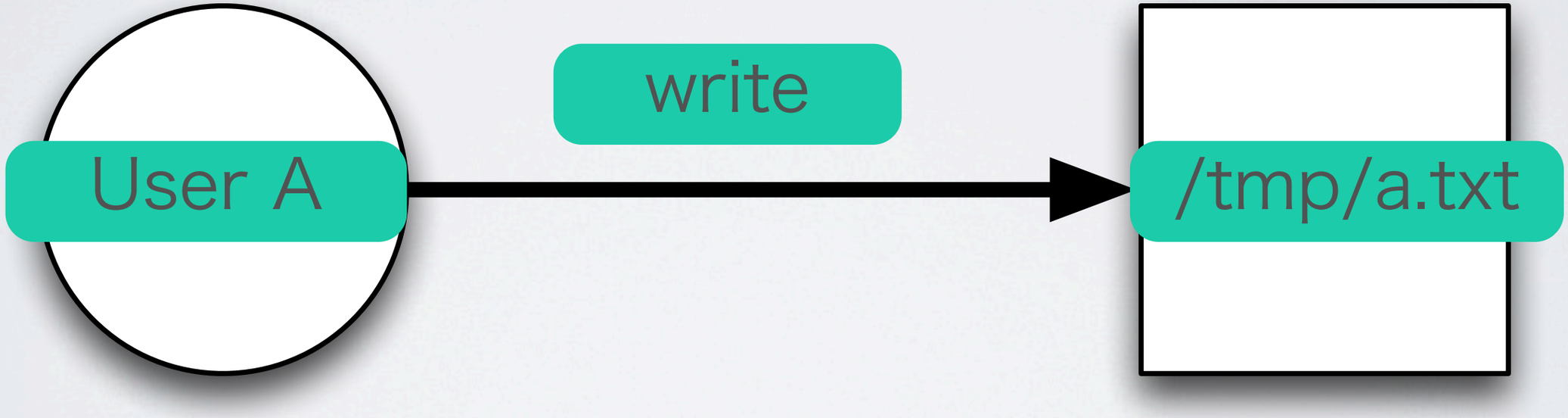
資源の使用量管理

- **(1) 資源へのアクセス権限の管理**
  - 単純なアクセス制御リスト(ACL)を使う



資源の使用量管理

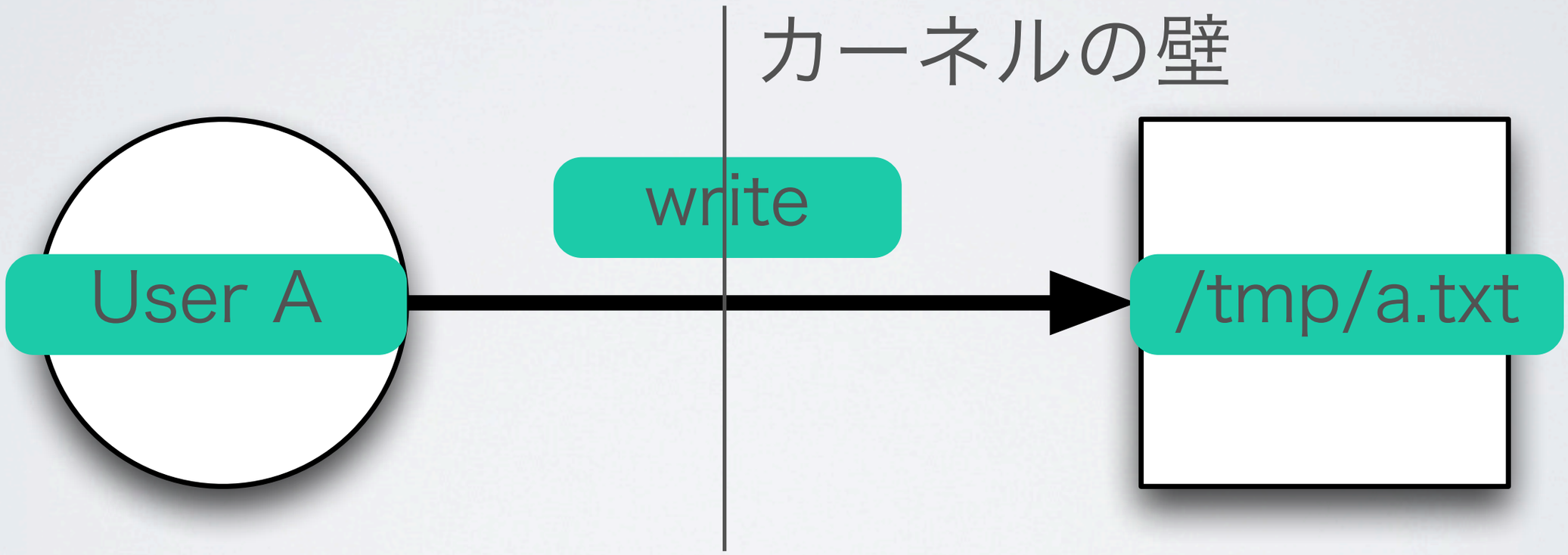
- **(1) 資源へのアクセス権限の管理**
  - 単純なアクセス制御リスト(ACL)を使う



ルール: {S,A,O} = 許可 or 拒否

資源の使用量管理

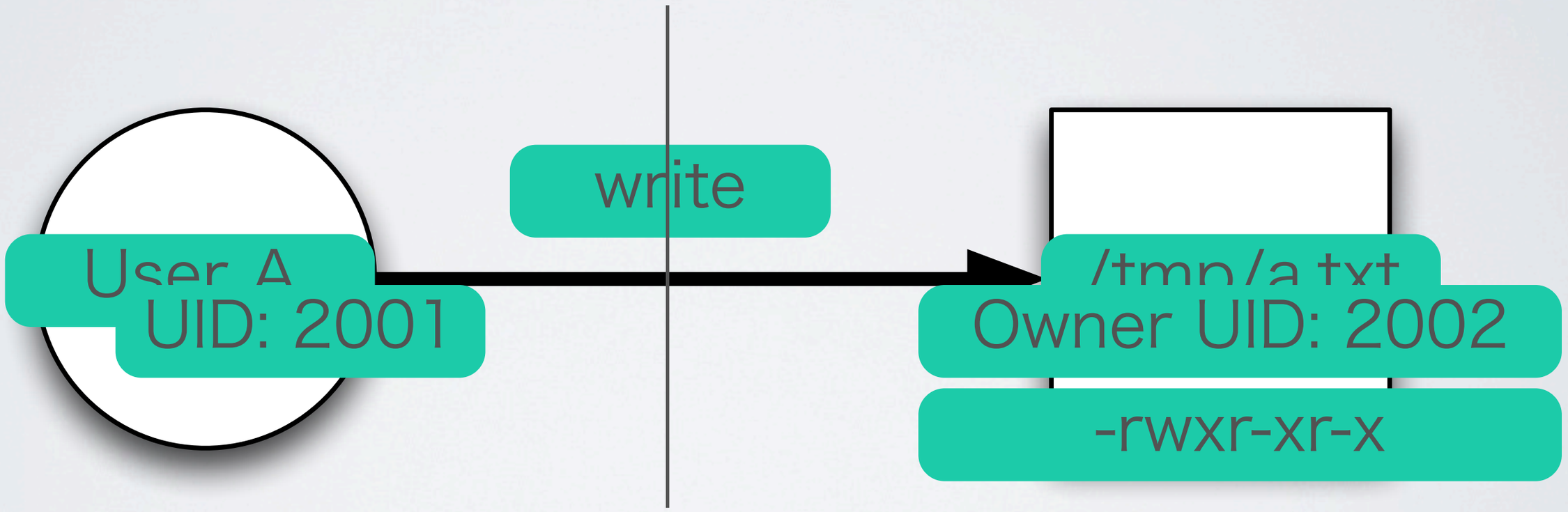
- **(1) 資源へのアクセス権限の管理**
  - 単純なアクセス制御リスト(ACL)を使う



ルール: {S,A,O} = 許可 or 拒否

資源の使用量管理

- **(1) 資源へのアクセス権限の管理**
  - 単純なアクセス制御リスト(ACL)を使う



カーネルは、{S, O} の UID とACLで判断

資源の使用量管理

- **(1) 資源へのアクセス権限の管理**
  - UID で管理されているものが対象
    - ファイル
    - プロセス
  - アクセス制御リストはどこに？
    - ファイルの許可属性
    - それ以外は「root かどうか(UID=0)」しか基本的に見ない

- 注：もっと強力な権限制御は、MACを使うと実現できる

資源の使用量管理

- **昔からある資源使用量管理**
  - Disc Quotas（使用可能ディスク容量の制限）
    - **S** = プロセスのUID
    - **O** = ファイルシステム上の使用量（ブロック・ファイル）
    - **A** = ファイルシステムへの書き込みアクセス
    - **ACL** = quotaファイル（edquota(8)で編集）
    - **資源監視** = カーネルがファイルシステムへのアクセスを監視
  - 4.2BSD でカーネルに追加された
  - デフォルトでは無効になっている

資源の使用量管理

- **昔からある資源使用量管理**
  - setrlimit(2)（プロセスが使用可能な共有資源の使用制限）
    - **S** = プロセスのID (PID)
    - **O** = 各種資源
    - **A** = 各種資源の取得（もちろんシステムコール経由）
    - **ACL** = カーネルのメモリ上（struct proc の中）
    - **資源監視** = カーネルが監視
  - これも4.2BSD でカーネルに追加された
  - POSIX にも採用されたのでだいたい使える

資源の使用量管理

setrlimit を使う簡単な方法＝ログインクラスで定義する

```
# cat /etc/login.conf
default:\
        :passwd_format=md5:\
        :copyright=/etc/COPYRIGHT:\
        :welcome=/etc/motd:\
        :setenv=MAIL=/var/mail/$,BLOCKSIZE=K,FTP_PASSIVE_MODE=YES:\
        :path=/sbin /bin /usr/sbin /usr/bin /usr/games /usr/local/sbin /usr/local/bin
~/bin:\
        :nologin=/var/run/nologin:\
        :cputime=unlimited:\
        :datasize=unlimited:\
        :stacksize=unlimited:\
        :memorylocked=unlimited:\
        :memoryuse=unlimited:\
        :filesize=unlimited:\
        :coredumpsize=unlimited:\
        :openfiles=unlimited:\
        :maxproc=unlimited:\
        :sbsize=unlimited:\
        :vmemoryuse=unlimited:\
        :swapuse=unlimited:\
        :pseudoterminals=unlimited:\
        :priority=0:\
        :ignoretime@:\
        :umask=022:
```

資源の使用量管理

setrlimit を使う簡単な方法＝ログインクラスで定義する

- setrlimit そのものは PID で効く
- 「ログインクラス」は、UID に対して
  ケーパビリティを定義する枠組み
- 「ケーパビリティ」＝アクセス制御に
  使う情報の一種
- この情報を元に、あるユーザのすべてのプロセスに
  setrlimit(2) が自動的に適用される

資源の使用量管理

今の制限値を調べる／変更する=limits(1)コマンド

```
# /usr/bin/limits
Resource limits (current):
  cputime              infinity secs
  filesize             infinity kB
  datasize             33554432 kB
  stacksize              524288 kB
  coredumpsize-cur            0 kB
  memoryuse            infinity kB
  memorylocked         infinity kB
  maxprocesses             5547
  openfiles               11095
  sbsize               infinity bytes
  vmemoryuse           infinity kB
  pseudo-terminals     infinity
  swapuse              infinity kB

# limits -u 1000000
Resource limits (current):
  maxprocesses          1000000
```

資源の使用量管理

シェル組み込みのlimit/ulimit でも調べられる

```
# limit
cputime      unlimited
filesize     unlimited
datasize     33554432 kbytes
stacksize    524288 kbytes
coredumpsize 0 kbytes
memoryuse    unlimited
vmemoryuse   unlimited
descriptors  11095
memorylocked unlimited
maxproc      5547
sbsize       unlimited
swapsize     unlimited
```

資源の使用量管理

- **欠点**
  - 制限は基本的に「ユーザ単位」でしか行えない
  - 「複数のユーザ／プロセスの集合に対して最大値を決めたい」とか複雑な設定は難しい
  - でもそんなの、どういう場合に使うの？

環境の分割

- 2台の物理マシンがやってることを、1台に集約したい
  - マシンは十分に高速／消費電力や設置体積を減らしたい
  - 単純に設定を混ぜてもダメ
    - 2台のユーザ構成は違うかも
    - root 権限を持っている人は両方の設定を操作できちゃう
    - 等々

話はがらりと変わって....

環境の分割

- **2台の物理マシンがやってることを、1台に集約したい**
  - マシンは十分に高速／消費電力や設置体積を減らしたい
  - 単純に設定を混ぜてもダメ
    - 2台のユーザ構成は違うかも
    - root 権限を持っている人は両方の設定を操作できちゃう
    - 等々

環境の分割

- 2台の物理マシンがやってることを、1台に集約したい
  - マシンは十分に高速／消費電力や設置体積を減らしたい
  - 単純に設定を混ぜてもダメ
    - 2台のユーザ構成は違うかも
    - root 権限を持っている人は両方の設定を操作できちゃう
    - 等々

実際は1台だけど、2台に見えるような

カラクリはないものか?

環境の分割

- **環境 = ユーザプロセスの集合体**
- カーネルは要求に応じて資源を分配するだけ
  - 分配に必要な情報はカーネルの外にあることがほとんど
  - 考えよう：/etc/passwd はどこにある？誰がチェックしてる？

環境の分割

- **昔からある方法: chroot jail**
  - UNIX = 資源をファイルで管理している
  - 見えるファイルを分ければいいのでは！

環境の分割

- **昔からある方法: chroot jail**
  - UNIX = 資源をファイルで管理している
  - 見えるファイルを分ければいいのでは！
  - chroot システムコールを発行すると、
    - そのプロセスから見えるルートディレクトリが変わる
    - ファイルシステム上でアクセスできる範囲が限定される
- **この種の環境の分割　＝「名前空間の分離」**
  - 実際の資源はまとまって管理されているけれど、
    それにアクセスする方法（名前）が分離されている

資源の分割

- **やってみよう**
  - 今のシステムのコピーをつくってみる

```
# mkdir -p /a/jail /a/jail/dev /a/jail/tmp
# cp -Rp /etc /lib /bin /sbin /usr /var /a/jail
# cd /
# mount -t devfs devfs /a/jail/dev
# chroot /a/jail /bin/sh
```

これだけでおしまい！

- make installworld でも作れる（/etc とかの設定は用意しないとダメ）

```
# mkdir -p /a/jail
# cd /usr/src && make DESTDIR=/a/jail installworld
```

資源の分離

- **chroot jail でできること**
  - カーネルが共通で、見えるルートディレクトリが異なる
  - 複数のマシンのルートディレクトリをまるまる
    あるディレクトリにコピーして、chroot でシェルを
    起動すると、ほぼ独立した環境になる
  - サブディレクトリの外には出られない

- セキュリティ対策として使われることがしばしば
  （ただし抜け道はいくつか存在する）

資源の分離

- **できないこと**
  - ファイルとして管理されていない資源は分離されてないので、垣根を超えることができてしまう
  - ネットワーク設定、プロセスに対する kill など

資源の分離

- **もっと分離したい！**
  - FreeBSD は chroot を拡張して、
    他の資源も分離できるものをつくった ＝ “FreeBSD Jail”
  - **中身は chroot + α**
  - 名前空間が分離されている
    - sysctl MIB
    - proc list
    - PID 空間（PID + Jail ID で管理）
  - 注意：UID は分離されていない
- **資源は同じカーネルが一括管理しているが、**
  Jail 環境はその外の資源が見えない

FreeBSD Jail

- **やってみよう**
  - 今のシステムのコピーをつくってみる

```
# mkdir -p /a/jail /a/jail/dev /a/jail/tmp
# cp -Rp /etc /lib /bin /sbin /usr /var /a/jail
# cd /
# mount -t devfs devfs /a/jail/dev
# jail -c name=j1 host.hostname=j1.example.com path=/a/jail command=/bin/sh
```

これだけでおしまい！

- 違うのは jail コマンドを使うところ
- ネットワークの設定等も同時にできる

FreeBSD Jail

- **やってみよう**
  - 今のシステムのコピーをつくってみる

```
# mkdir -p /a/jail /a/jail/dev /a/jail/tmp
# cp -Rp /etc /lib /bin /sbin /usr /var /a/jail
# cd /
# mount -t devfs devfs /a/jail/dev
# jail -c name=j1 host.hostname=j1.example.com path=/a/jail command=/bin/sh
# ps auwwx
USER   PID  %CPU %MEM    VSZ    RSS  TT  STAT STARTED    TIME COMMAND
0    47173   0.0  0.2  14636   2168   3  SJ    6:08AM 0:00.01 /bin/sh
0    47178   0.0  0.1  14328   1524   3  R+J   6:11AM 0:00.00 ps auwwx
#
```

FreeBSD Jail

- **やってみよう**
  - ちなみに chroot の時は... （たくさん出てくる）

```
# mkdir -p /a/jail /a/jail/dev /a/jail/tmp
# cp -Rp /etc /lib /bin /sbin /usr /var /a/jail
# cd /
# mount -t devfs devfs /a/jail/dev
# chroot /a/jail/bin/sh
# ps auwwx
USER    PID  %CPU %MEM    VSZ    RSS  TT  STAT STARTED       TIME COMMAND
0        11 199.9  0.0      0     32  ??  RL    7:59AM 2611:43.88 [idle]
0         0   0.0  0.0      0    160  ??  DLs   7:59AM    0:00.06 [kernel]
   :
   :
   :
   :
   :
   :
```

FreeBSD Jail

- **やってみよう**
  - jail(8) で起動したプロセスは、jailed process になる

```
# mkdir -p /a/jail /a/jail/dev /a/jail/tmp
# cp -Rp /etc /lib /bin /sbin /usr /var /a/jail
# cd /
# mount -t devfs devfs /a/jail/dev
# jail -c name=j1 host.hostname=j1.example.com path=/a/jail command=/bin/sh
# ps auwwx
USER   PID  %CPU %MEM    VSZ    RSS  TT  STAT STARTED    TIME COMMAND
0    47173   0.0  0.2  14636   2168   3  SJ    6:08AM 0:00.01 /bin/sh
0    47178   0.0  0.1  14328   1524   3  R+J   6:11AM 0:00.00 ps auwwx
```

FreeBSD Jail

- **FreeBSD Jail の環境**
  - root 以外の UID からのファイルに対する操作は特に変わらず
  - root 権限が限定的
    - ファイルシステムに対しては通常通り（ACL を使う）
    - sysctl や jail 環境外部に影響を与えるシステムコールは拒否
      （raw socket, mount, sysvipc など）
  - 名前空間が分かれているので、Jail の外のプロセスは見えない
  - ネットワークインタフェースも名前空間が分離されている

FreeBSD Jail

- **起動時に自動的につくるには**
  - rc.conf(5) に書く

```
jail_enable="YES"
jail_list="j1"
jail_j1_rootdir="/a/basejail"
jail_j1_hostname="j1.example.com”
jail_j1_interface=""
jail_j1_ip="127.0.0.1"
jail_j1="/bin/sh /etc/rc"
```

```
# /etc/rc.d/jail start
```

- 作成や維持管理には、
  FreeBSD のインストールイメージを展開して Jail をつくる
  sysutils/ezjail というツールが人気

FreeBSD Jail

- **実装（難しい話）**
  - jail(2) システムコール(struct jail を受け取る)
    - JID を設定してプロセス生成
  - jail_attach(2) でプロセスを JID に割り当てる
  - ABI emulation (Linux, solaris) は親プロセスを引き継ぐ

```
struct jail {
        u_int32_t       version;
        char            *path;
        char            *hostname;
        char            *jailname;
        unsigned int    ip4s;
        unsigned int    ip6s;
        struct in_addr  *ip4;
        struct in6_addr *ip6;
};
```

FreeBSD Jail

- **大技**
  - CentOS のユーザランドを動かす　→　＜デモ＞

```
% df
Filesystem           1K-blocks      Used    Avail Capacity  Mounted on
/dev/da0s1a            1012974    419890   512048    45%    /
devfs                        1         1        0   100%    /dev
/dev/da0s1d            4058062      1708  3731710     0%    /tmp
/dev/da0s1f           30462636   2865782 25159844    10%    /usr
/dev/da0s1e           20308398   1202680 17481048     6%    /var
/dev/da0s1g          175586848  78326048 83213854    48%    /a
devfs                        1         1        0   100%    /a/altroot/a/centos/4.8/dev
linprocfs                    4         4        0   100%    /a/altroot/a/centos/4.8/proc
192.168.200.10:/home 203114302 178650720  8214438    96%    /a/altroot/a/centos/4.8/home
```

- 動機：業務で使っているソフトがCentOS4でしか動かなかった :(

FreeBSD Jail

- **大技**
  - CentOS のユーザランドを動かす

```
% uname -a
Linux cadmaster2.vlsi.ee.noda.tus.ac.jp 2.6.16 FreeBSD 8.2-STABLE #4: Thu Dec 29 09:05:09
JST 2011 i686 i686 i386 GNU/Linux
```

アイディアのおさらい：ユーザランド部分をまるまる複数持たせて
仮想環境をつくる！

- FreeBSD Jail は chroot よりも分離が強いので
  こういう使い方が十分に可能
- yum もほぼ動く（もちろん落とし穴も多いですが...）

FreeBSD Jail

- **FreeBSD Jail のさらなる拡張**
  - VIMAGE Jail (VNET)
  - 名前空間分離ではない、
    完全な独立ネットワークスタックを備えた Jail
  - 9.0 から”options VIMAGE”で利用可能（デフォルトは無効）
  - たとえば...
    3個jailをつくって、クライアント：ルータ：サーバのような
    ネットワーク環境を簡単に 1 台のマシンの中だけで
    作ることができる

FreeBSD Jail

- **ちなみに、KVM とかとどこが違うの？**
  - ハイパーバイザは存在せず、カーネルは共通
  - 根本的に別物です
  - 資源の見え方を変えただけなので、
    オーバヘッドは非常に小さい

さて資源制御だ

- FreeBSD Jail を使うと仮想環境がつくれるぜ
- 資源はどうなってる？
  - カーネルがひとつで資源の見え方が違うだけ＝制御されてない...

ようやく本題です

- setrlimit で戦えるのか
  - FreeBSD Jail の中はたくさんのUIDとPID...
  - そしてJailもたくさん...

- そもそも Jail 単位で資源がどれだけ使われているのか
  カーネルは知らない！

さて資源制御だ

- **FreeBSD Jail を使うと仮想環境がつくれるぜ**
- **資源はどうなってる？**
  - カーネルがひとつで資源の見え方が違うだけ＝制御されてない...
- **setrlimit で戦えるのか**
  - FreeBSD Jail の中はたくさんのUIDとPID...
  - そしてJailもたくさん...
- そもそも Jail 単位で資源がどれだけ使われているのか
  カーネルは知らない！

RACCT / RCTL

- ないなら作ってやろう
- setrlimit 実装の欠点
  - ユーザ単位での制限が難しい
  - 制限を超えた時にカーネルが起こすアクションが変えられない
  - 一度設定した後に、制限値を自由に変更できない

- RACCT
  - 使用資源量をカーネルが把握するためのフレームワーク
- RCTL
  - RACCT の結果を利用して、資源量制限を設定するフレームワーク
  - {S,A,O}のACL と似た形で制限資源量をルールで記述できる

RACCT / RCTL

- 9.0 から使えます
  - options RACCT
  - options RCTL
- コマンドは rctl(8)
- ルールファイルは /etc/rctl.conf

RACCT / RCTL

- **資源使用量のルール**
  - subject : subject-id : resource : action = amount/per

```
# rctl -hu user:hrs
cputime=4214
datasize=21M
stacksize=0
coredumpsize=0
memoryuse=33M
memorylocked=0
maxproc=12
openfiles=0
vmemoryuse=446M
pseudoterminals=0
swapuse=9620k
nthr=12
msgqqueued=0
msgqsize=0
nmsgq=0
nsem=0
nsemop=0
nshm=0
shmsize=0
wallclock=899k
```

  - -u オプションは、現在の制限値を表示
  - subject = user, process, loginclass, jail
  - ユーザ単位、プロセス単位、
    ログインクラス単位、jail 単位

RACCT / RCTL

- **資源使用量のルール**
  - subject : subject-id : resource : action = amount/per

```
# rctl -hu user:hrs
cputime=4214
datasize=21M
stacksize=0
coredumpsize=0
memoryuse=33M
memorylocked=0
maxproc=12
openfiles=0
vmemoryuse=446M
pseudoterminals=0
swapuse=9620k
nthr=12
msgqqueued=0
msgqsize=0
nmsgq=0
nsem=0
nsemop=0
nshm=0
shmsize=0
wallclock=899k
```

  - -u オプションは、現在の制限値を表示
  - resource = 左の項目

RACCT / RCTL

- **資源使用量のルール**
  - subject : subject-id : resource : action = amount/per

```
# rctl -hu user:hrs
cputime=4214
datasize=21M
stacksize=0
coredumpsize=0
memoryuse=33M
memorylocked=0
maxproc=12
openfiles=0
vmemoryuse=446M
pseudoterminals=0
swapuse=9620k
nthr=12
msgqqueued=0
msgqsize=0
nmsgq=0
nsem=0
nsemop=0
nshm=0
shmsize=0
wallclock=899k
```

  - -u オプションは、現在の制限値を表示
  - action = deny, log, devctl, sig*
  - 拒否、ログ記録、devctlイベント、シグナル発行

RACCT / RCTL

- **資源使用量のルール**
  - subject : subject-id : resource : action = amount/per

```
# rctl -hu user:hrs
cputime=4214
datasize=21M
stacksize=0
coredumpsize=0
memoryuse=33M
memorylocked=0
maxproc=12
openfiles=0
vmemoryuse=446M
pseudoterminals=0
swapuse=9620k
nthr=12
msgqqueued=0
msgqsize=0
nmsgq=0
nsem=0
nsemop=0
nshm=0
shmsize=0
wallclock=899k
```

  - -u オプションは、現在の制限値を表示
  - amount = 資源量
  - per = subject と同じ
    - 1g/user = ユーザ単位で 1G

RACCT / RCTL

- **設定例**　→　＜デモ＞

```
# rctl -a user:hrs:maxproc:deny=10
```

```
# rctl -a jail:j1:maxproc:log=10/user
```

- -a で追加
- -r で削除
- -u で現在使用量
- ~~ルールの一覧表示はできません~~

  オプションを付けないとルールが一覧表示されます

  「rctl ::maxproc:」のように、表示されるルールの限定も可能

RACCT / RCTL

- **設定例**

```
# rctl -a jail:j1:maxproc:devctl=5
```

```
# /etc/devd.conf
notify 0 {
        match "system"          "RCTL";
        match "rule"            "jail:j1:maxproc:.*";
        action                  "/etc/maxproc.sh restart";
};
```

- action = devctl を使うと、devd(8) 経由でいろいろできる
- devd.conf のサンプルは、action の最後にセミコロンが抜けてます。注意。

RACCT / RCTL

- **設定例**

```
# rctl -a user:70:swap:devctl=1g
```

```
# /etc/devd.conf
notify 0 {
        match "system"          "RCTL";
        match "rule"            "user:70:swap:.*";
        action                  "/usr/local/etc/rc.d/postgresql restart";
};
```

- サーバプロセスにUIDを割り当てて、
  swap 使用量を監視＝1GB を超えたら再起動！

RACCT / RCTL

- **設定例**

```
# /etc/rctl.conf
user:70:swap:devctl=1g
jail:j1:maxproc:devctl=5

# /etc/rc.conf
rctl_enable="YES"
```

- ルールファイルは列挙するだけ
- 起動時に設定される

まとめ

- **カーネルとユーザランドプログラムの区別**
  - カーネルは資源の守り神
  - ユーザランドプログラムはCPU時間やメモリを借りて動く
- **環境はユーザランドプログラムがつくる**
  - 複数用意してうまく分離すると、カーネルが共通でも独立環境が実現可能
  - chroot jail, FreeBSD Jail の基本的なアイディアは単純
- **資源の制限はどうやるの？**
  - 伝統的には setrlimit(2) や Quota がある。基本は UID 単位
  - 9.0R からは rctl(8) でルールベースの設定ができるようになった！
    - ユーザ、プロセス、ログインクラス、jail 単位で設定可能
    - アクションも自由に設定できる
    - 複数の FreeBSD Jail を動かしているマシンで活用できる

おしまい

- 質問はありますか？