リモート管理の活用法と
システムトラブル対応の基礎

佐藤 広生 <hrs@FreeBSD.org>

東京理科大学/FreeBSD Project

お話すること

- リモート管理の基礎知識
  - OSが動いてる場合
  - OSが動いてない（異常な状態）の場合
  - 利用可能な機能とツール群
- シリアルコンソールの設定と使い方
- 応用：障害発生時のアプローチ

リモート管理の必要性

- サーバ用途で使うマシン
  - 運用を開始すると物理的に触れる機会が少ない。
  - データセンタや機械室に置かれている
    ＝直接触ることが困難。
- 物理アクセス制限は、セキュリティ確保の第一歩。触れなくすることは重要。
- 日常のメンテナンス作業
  → 何らかの方法で遠隔地から行う。

リモート管理の必要性

- 物理的アクセスが制限されているマシンとは？
  - キーボードがない。
  - ディスプレイがない。
  - 電源スイッチやリセットスイッチの操作に鍵が必要。
- サーバ用途に設計されたマシン
  → デスクトップ用途のものに比べて物理アクセスの手段が簡略化
  → そもそも必要ないから

リモート管理の必要性

- 例：1U ラックマウント型サーバ
- 省スペースが重要
- 物理アクセスの
  効率は度外視

リモート管理の必要性

- 例：普通のデスクトップ用タワー型マシン
- 別にこういうので
  サーバを構築しても
  問題はない。
- デスクトップ用途で
  設計されているので、
  物理アクセスは簡単
- 場所をとる

リモート管理の必要性

- 物理的アクセスできないマシンを、どうやって管理するのか？
- 管理＝ソフトウェアの操作と
ハードウェアの操作
- 大きく分けて2種類
  - OSに頼る方法
  - OSに頼らない方法

リモート管理の必要性

- ハードウェアの操作
  - 電源操作（オン／オフ／リセット）
  - 構成変更
    - 起動方法
    - 不良コンポーネント（メモリやCPU）の診断や切り離し
    - などなど

リモート管理の必要性

- ソフトウェアの操作
  - OSへのログイン
  - OSの強制停止
  - 稼働状態のモニタリング

今回のお話のターゲット

- ハードウェアの操作
  - 電源まわり
- ソフトウェアの操作
  - OSへのログイン
  - OSの強制停止

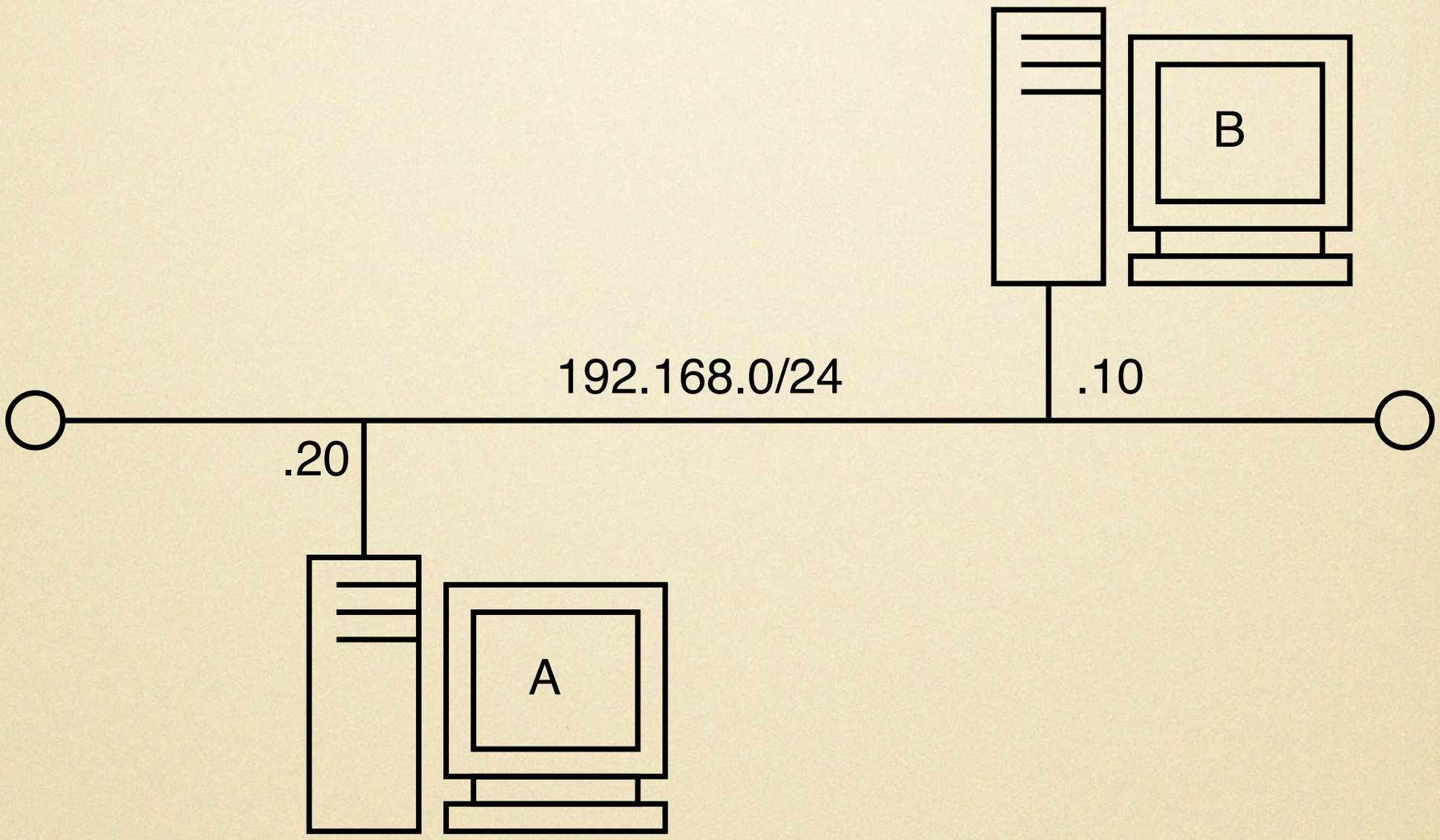
リモート管理で困ること

- OSをインストールした後に、キーボード・ディスプレイを外して運用
- ちゃんと動いているだろうか？
  どうやって知る？
- 動かなくなった時、どうする？
  - 電源スイッチで再起動してみる
  - 再起動してもサービスが元に戻らない...
- OSをアップグレードを
  リモートでやるのは怖い。

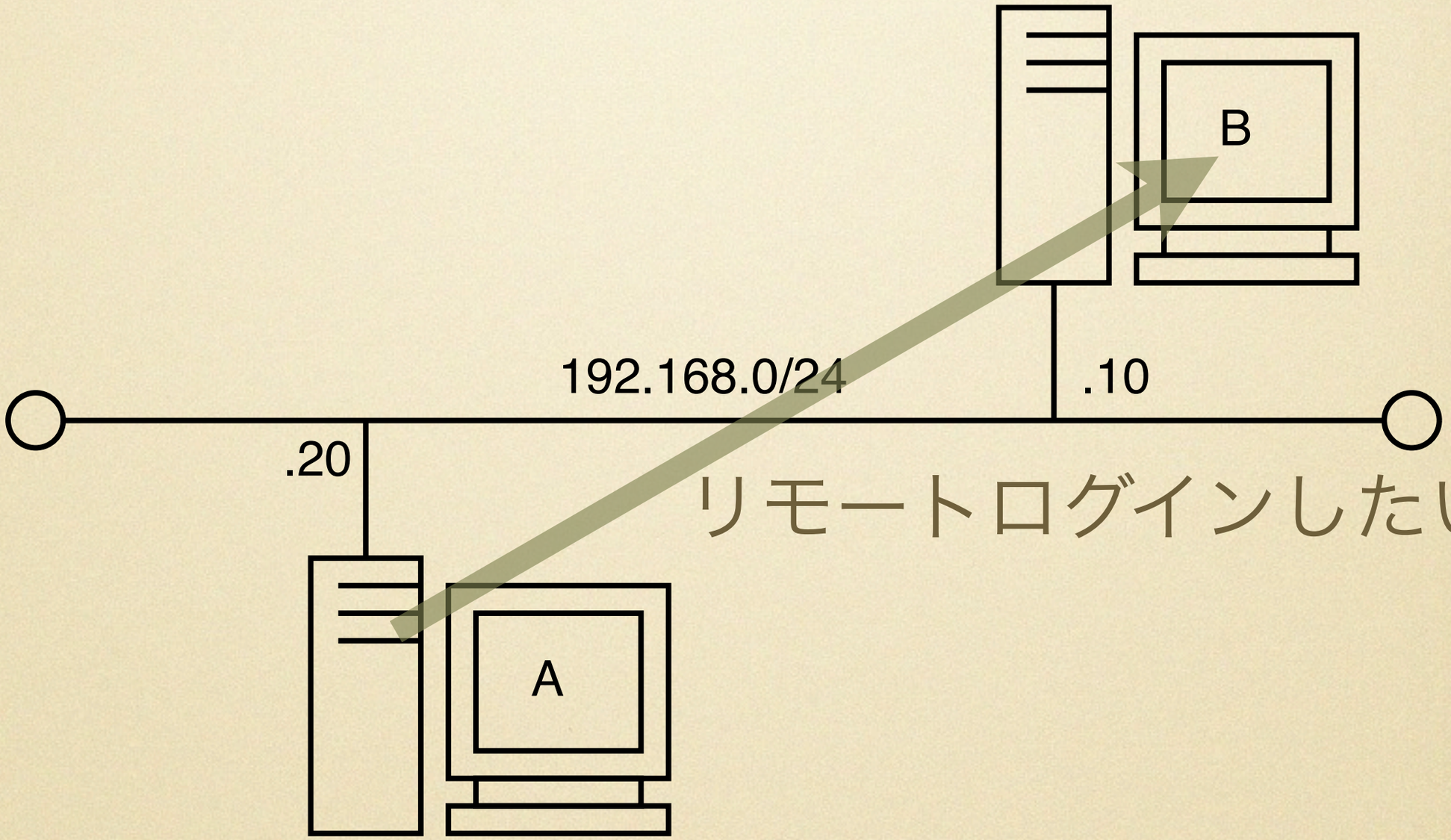
OSに頼る方法

- まずはOSが正常に動作している時に使える方法
  - SSHによるリモートログイン
  - screenを使ったセッション管理
- 基本中の基本。
- TCP/IPが正常に機能している必要がある。

SSH

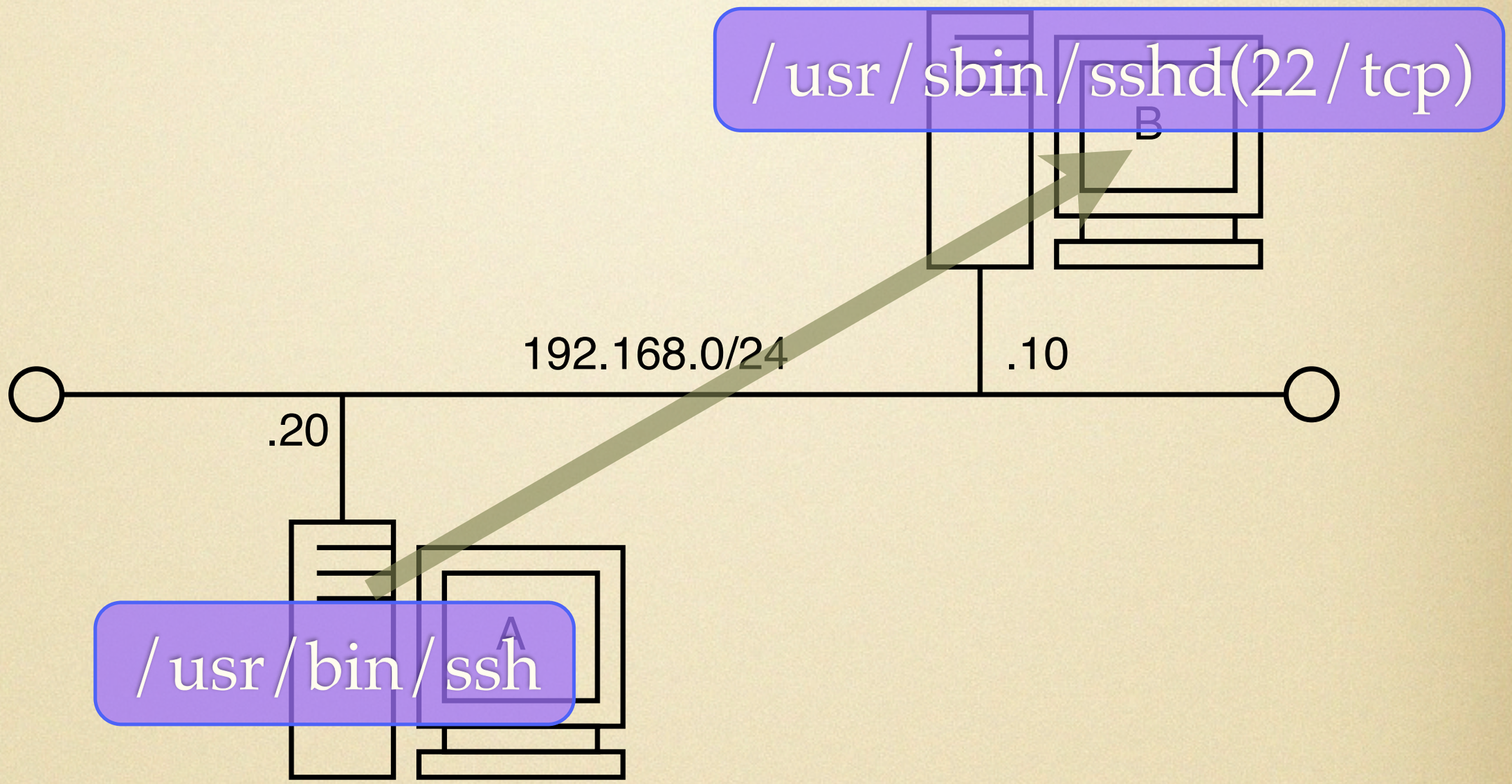


SSH

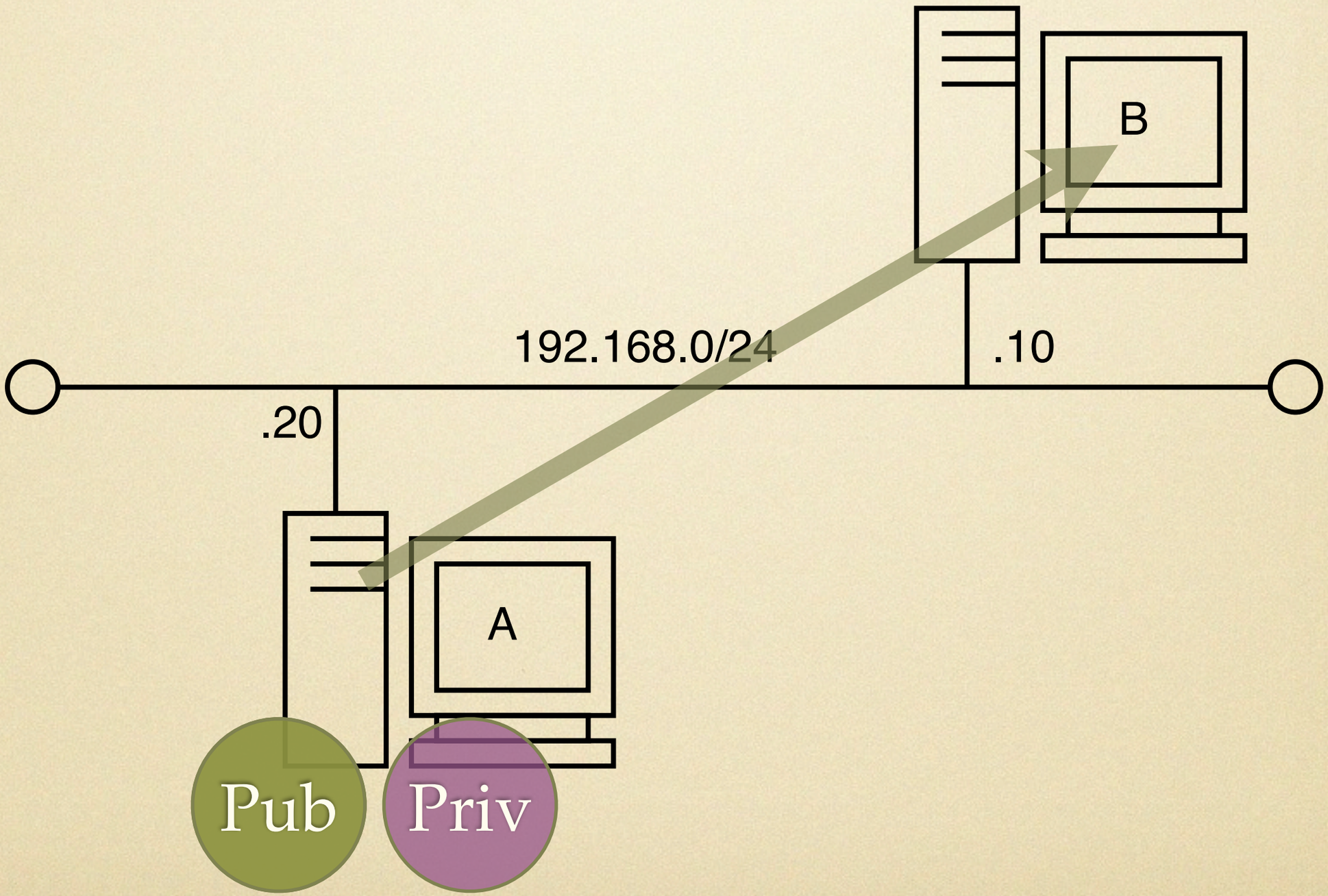


リモートログインしたい

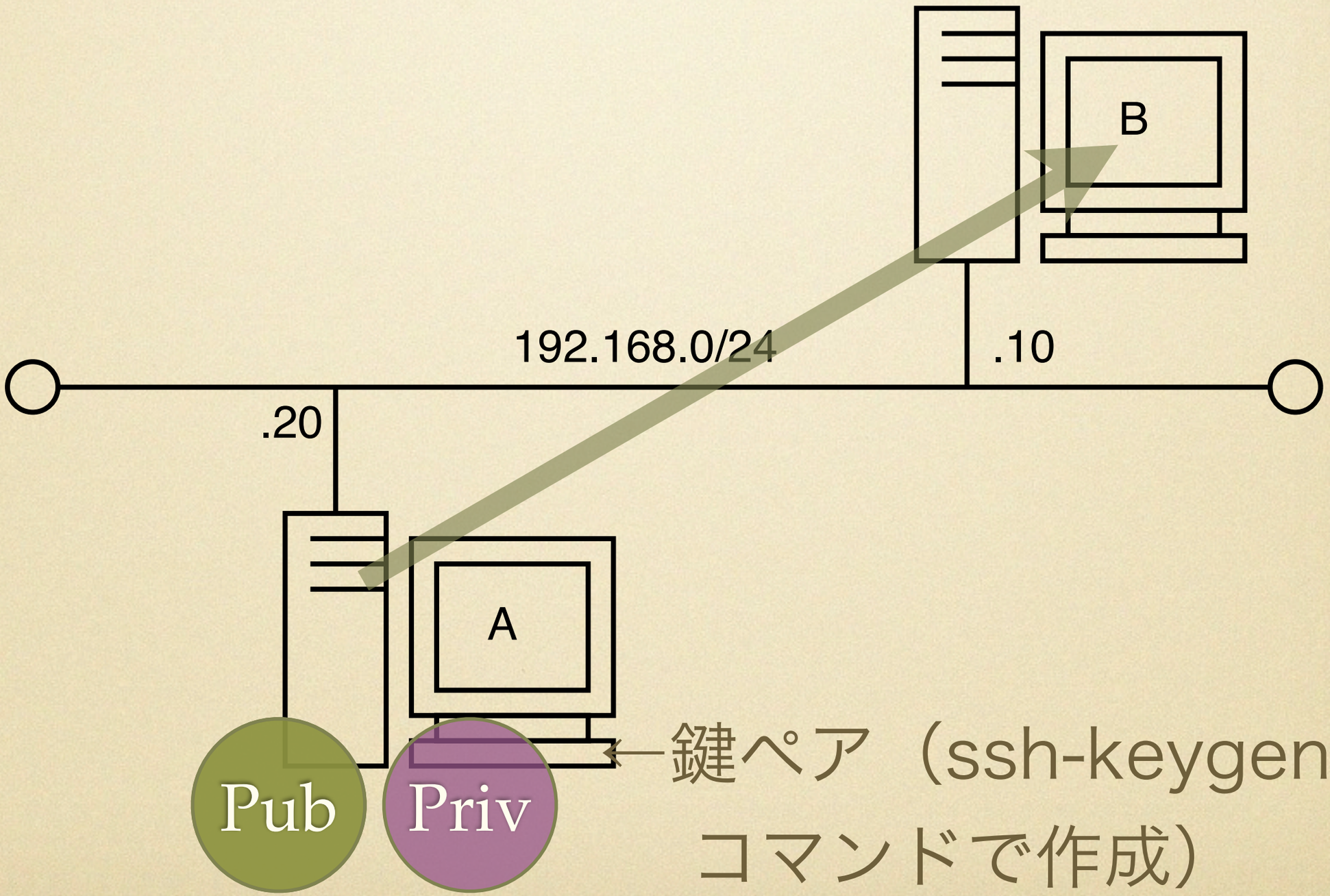
SSH



SSH



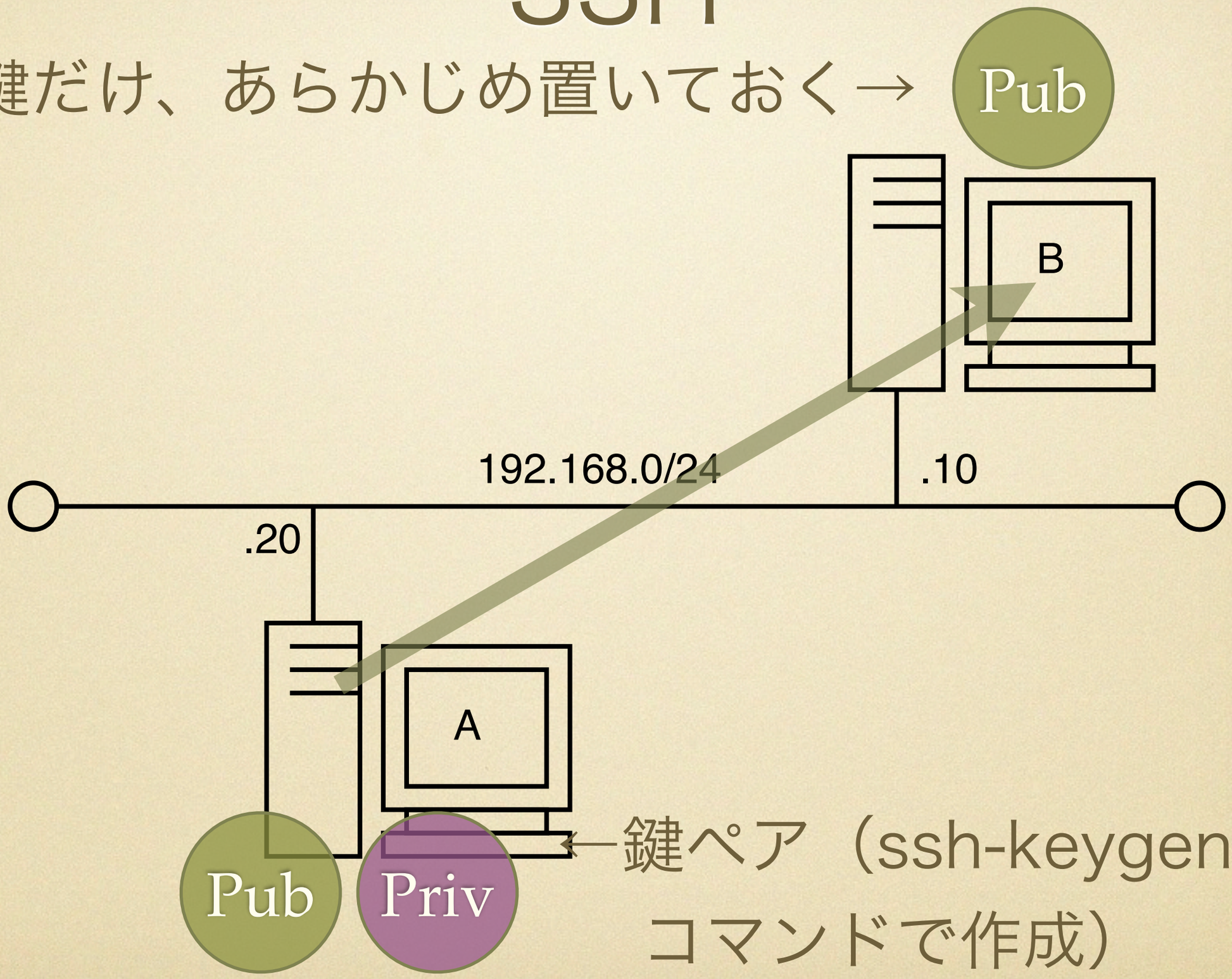
SSH



←鍵ペア（ssh-keygen
コマンドで作成）

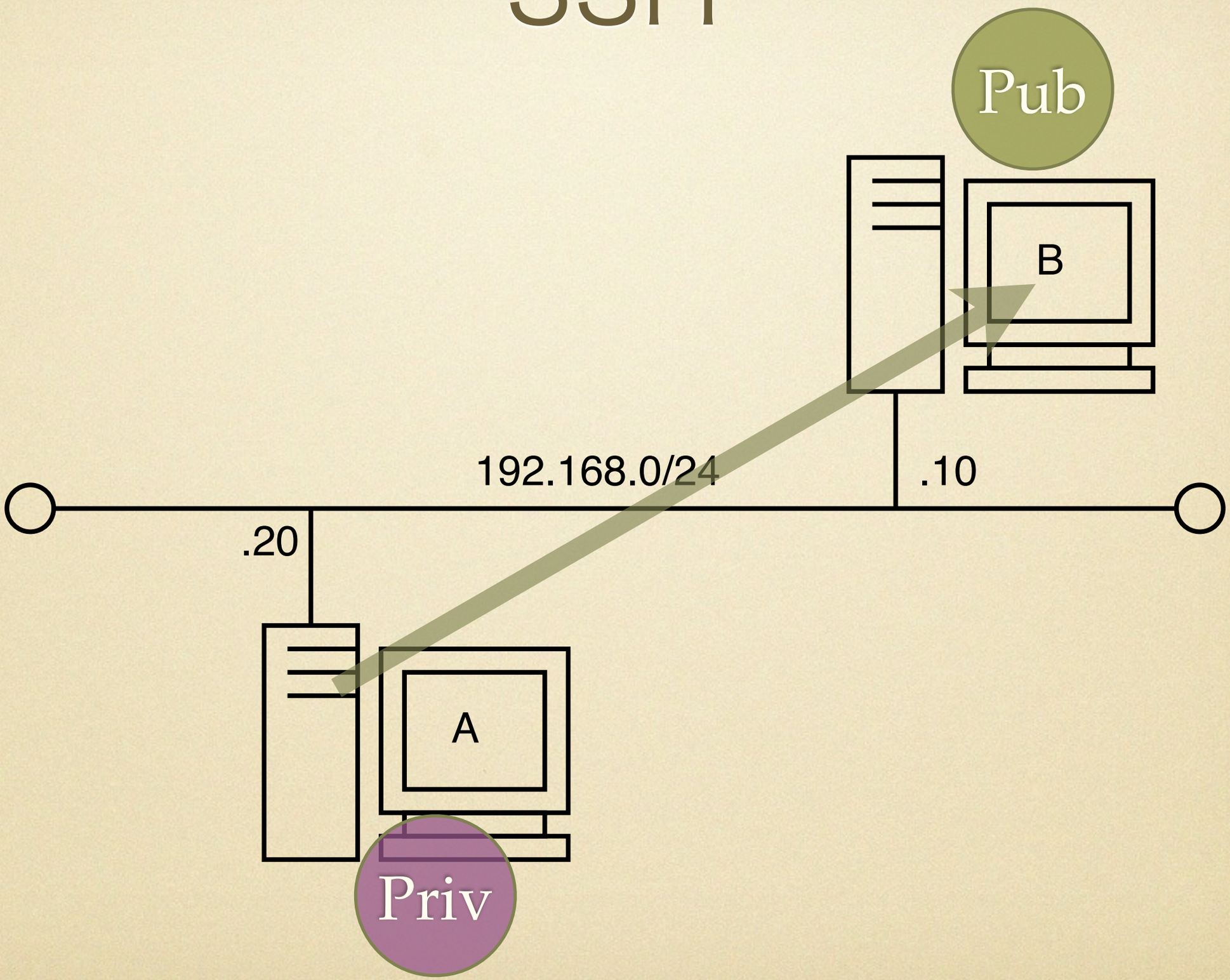
SSH

Pub鍵だけ、あらかじめ置いておく→

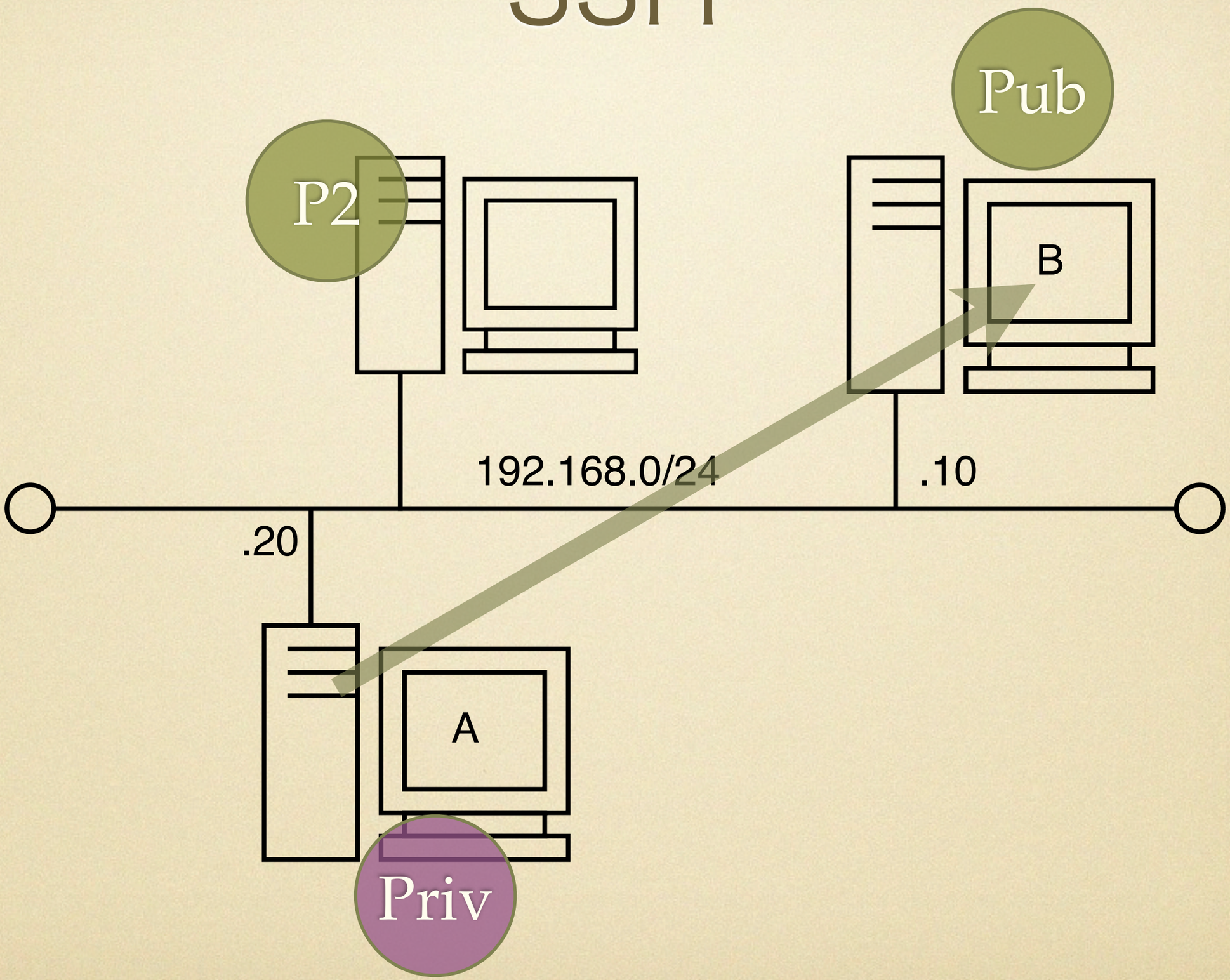


←鍵ペア（ssh-keygen コマンドで作成）

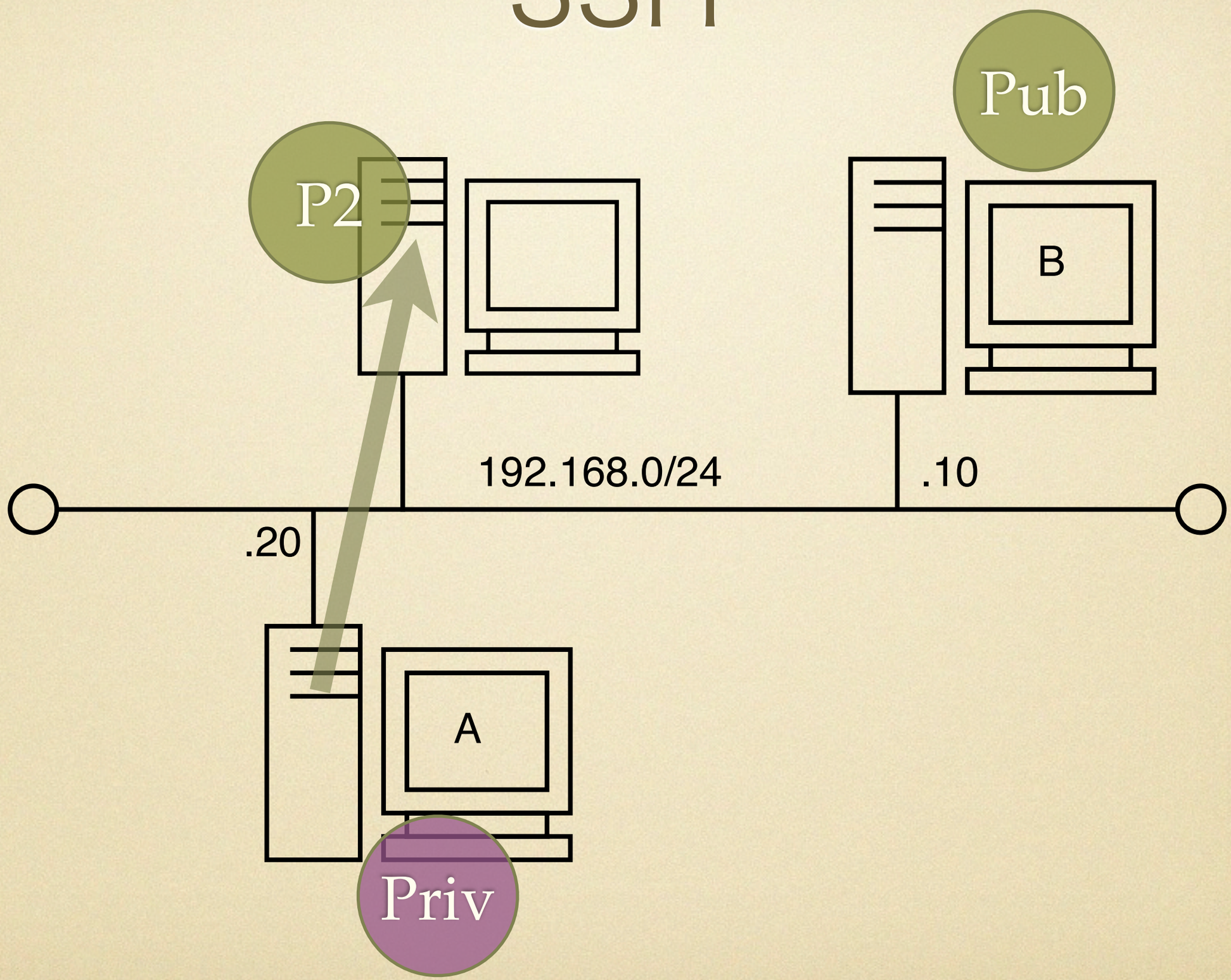
SSH



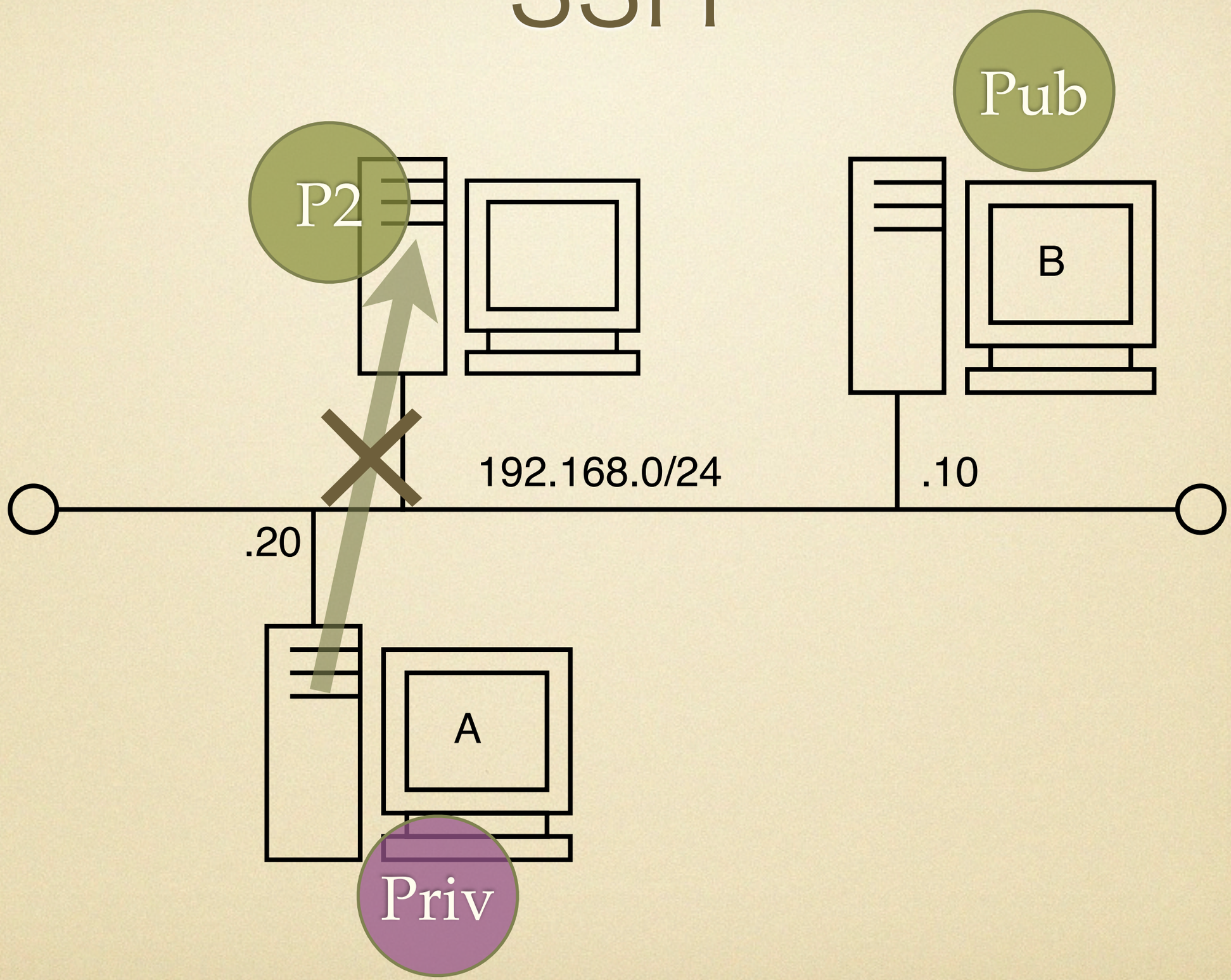
SSH



SSH



SSH



動作の流れ（超省略版）

- C: サーバの sshd にアクセス(22/tcp)
- S: sshd がアクセスを受理
- S: sshd がホスト情報を ssh に送る
- C: sshがホスト情報を受け取って、検証
  → 初めての接続時や、以前の接続と
  異なっていたら警告が出る
- C: 認証情報を送る
- S: 認証できるかどうかチェック。問題なければログインセッション開始

SSHを有効にするには

- サーバ側のsshdを有効にする
  → sshd_enable=”YES” in rc.conf
- クライアント側はsshコマンド
  → % ssh 192.168.0.10
- 認証は
  - パスワード（できるかぎり避けること）
  - 公開鍵認証（Pub/Privペアを使う）
  - 最初にホスト認証がかかる

公開鍵認証の設定

- Pub/Priv 鍵ペアをつくる（クライアント）

```
hrs@phaleano-allbsd-org % ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/hrs/.ssh/id_rsa):
Created directory '/Users/hrs/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/hrs/.ssh/id_rsa.
Your public key has been saved in /Users/hrs/.ssh/id_rsa.pub.
The key fingerprint is:
7a:a2:46:9c:85:57:34:1e:de:81:6f:eb:d8:db:51:ad hrs@phaleano-
allbsd-org.local
hrs@phaleano-allbsd-org % ls -al ~/.ssh/
total 8
drwx------    4 hrs  staff   136 Jun 18 11:15 ./
drwxr-xr-x+ 227 hrs  staff  7718 Jun 18 11:15 ../
-rw-------    1 hrs  staff  1751 Jun 18 11:15 id_rsa
-rw-r--r--    1 hrs  staff   411 Jun 18 11:15 id_rsa.pub
```

公開鍵認証の設定

- サーバ側で受理したい鍵をサーバに設定
  → authorized_keys という名前にする。

```
hrs@phaleano-allbsd-org % cat ~/.ssh/id_rsa.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAtqlDoWVK8hsP2t0zocs5VdhnfRtnSGdcoO
8m+kQ2CyUsQXm6SWp3KC1pAZbbje7eEHZJ6ptwK+WCn/HWk2dyG/
KVgG0LXWguphjmXW53cIZN8hQiKSAWc6QPFBpUCVVwMzJsD8pIFllft2ev52wE
dCz4d05rWN30Ra7Hh0ZQ0NVP63L1iFma8ChqfETvL6Oi1dJ4EXOjlpGQf9UCYi
h6q0bjuHhac7KsXsyqbmlWMSVIoniAdeNYkbeVM3jLXtPAhSy1YKqzlKcG5lGK
j0pyq86suyUNdtVoNXJDpQqUcc9Zdq7vhw49/e/
xsZrfCbt9yOfxD2QEtxWMAqtljEA8mQ== hrs@phaleano-allbsd-
org.local

hrs@alph-allbsd-org % cat id_rsa.pub >> ~/.ssh/authorized_keys
```

公開鍵認証の設定

- こうすると
  - サーバに登録したPub鍵に対応するPriv鍵を持っているクライアントからのみ、アクセスを許可（ややこしい日本語）
  - Priv鍵を持ってないクライアントからはアクセスできない
  - Priv鍵にはパスフレーズを設定できる。設定しないで使うことも可能。
  - authorized_keys ファイルには、複数の鍵を登録可能。

注意！

- Priv鍵は可能な限り
  - ネットワークに流すな
  - 増やすな
- Priv鍵のパスフレーズは、
  Priv鍵の使用に使われる。
  ログインに使われているわけではない。
  サーバ上のauthorized_keysとは関係なく変更できる。
- /etc/hosts_allowでsshdを許可していないと、接続できない

注意！

- SSHのパスワード認証を狙ったアタックが頻繁に発生
  - インターネットからアクセス可能なホスト
    → パスワード認証を無効にしておく
  - /etc/ssh/sshd_config で

    PasswordAuthentication no
    ChallengeResponceAuthentication no

複数のサーバ

- たとえば、直接接続できない複数のサーバがあったら？
  - 1. 経由するサーバにSSHで接続
  - 2. そこからさらにSSHで接続
- 2. の段階で、経由するサーバ上にPub/Priv鍵ペアが必要?

複数のサーバ

- 大原則：Priv鍵は増やすな。
- 経由するサーバ上にPub/Priv鍵を作成した場合、Priv鍵が増えてしまう
- Priv鍵が盗まれる可能性が高まる
- 信頼できないマシン上に自分のPriv鍵を置いてはいけない。
  - 複数人で共用しているファイルサーバ上
  - 自分以外がrootになっているマシン上

複数のサーバ

- じゃあ、どうすれば？
- SSH agent forwarding を使う
  - マシンAからSSHでログインしたマシンBにおいて、さらにSSHで別のマシンCに接続したい時に、マシンA-B間で交わされた認証情報をマシンB-C間でも使えるようにする仕組み（ややこしい日本語）
- agent forwarding を使うと、Priv鍵を中間ホストに置かなくてもマシンCにたどり着ける。

SSH agent forwarding

- 手順
  - ssh-agent(1)を起動する
  - ssh-add(1)で、agentで使う鍵を登録
  - あとは普通にsshでアクセスするだけ。
- ssh-agent(1)を実行すると表示される環境変数を、シェルに合わせて設定すること。
- ssh, sshd は個々にagent を無効にすることができる。デフォルトでは有効になっている(AllowAgentForwarding を参照)

SSH まとめ

- OSが正常に動作している時に、ソフトウェアの操作を遠隔地から行うための方法のひとつ。基本中の基本。
- 22/tcp を使用
- 公開鍵を使った認証
- AESで暗号化された安全な通信路
- 鍵の扱いに注意！agentを駆使してPriv鍵は最小限にする。
- 欠点：sshdがコケると使えない。

おまけ：screen

- 端末のセッション管理をするユーティリティ
- 複数のサーバにログインした状態を
  維持できる。
- 日々のメンテナンスや監視に有用
- ports/misc/screen

おまけ：screen

- ^a c: 新しいスクリーン/^a k スクリーン削除
- ^a n: 次のスクリーン/^a p: 前のスクリーン
- ^a 0: 0 番のスクリーン
- ^a S: スクリーン上下分割（戻すには ^a Q)
- ^a TAB: スクリーン上下分割後の移動
- ^a d: セッション切り離し
- screen -r/-x: 再接続

sshdが使えない時は？

- sshdが動く→マルチユーザモードで
  ネットワークの設定がきちんと
  完了していることが前提
- トラブルが発生した時は、sshdが
  動かないことがしばしば。
- ハードウェアの操作はほとんどできない
- もっと確実なアクセス方法は？

リモートコンソールアクセス

- TCP/IPよりも低いレベルでのアクセス
- 基本はシリアルコンソール
  - シリアル(ANSI/TIA/EIA-232-F-1997)
  - シリアル over Ethernet
  - シリアル over FireWire(IEEE1394)
  - シリアル over IPMI over LAN
- 「キーボードの入力をリモートで行う」というイメージ

シリアルコンソール

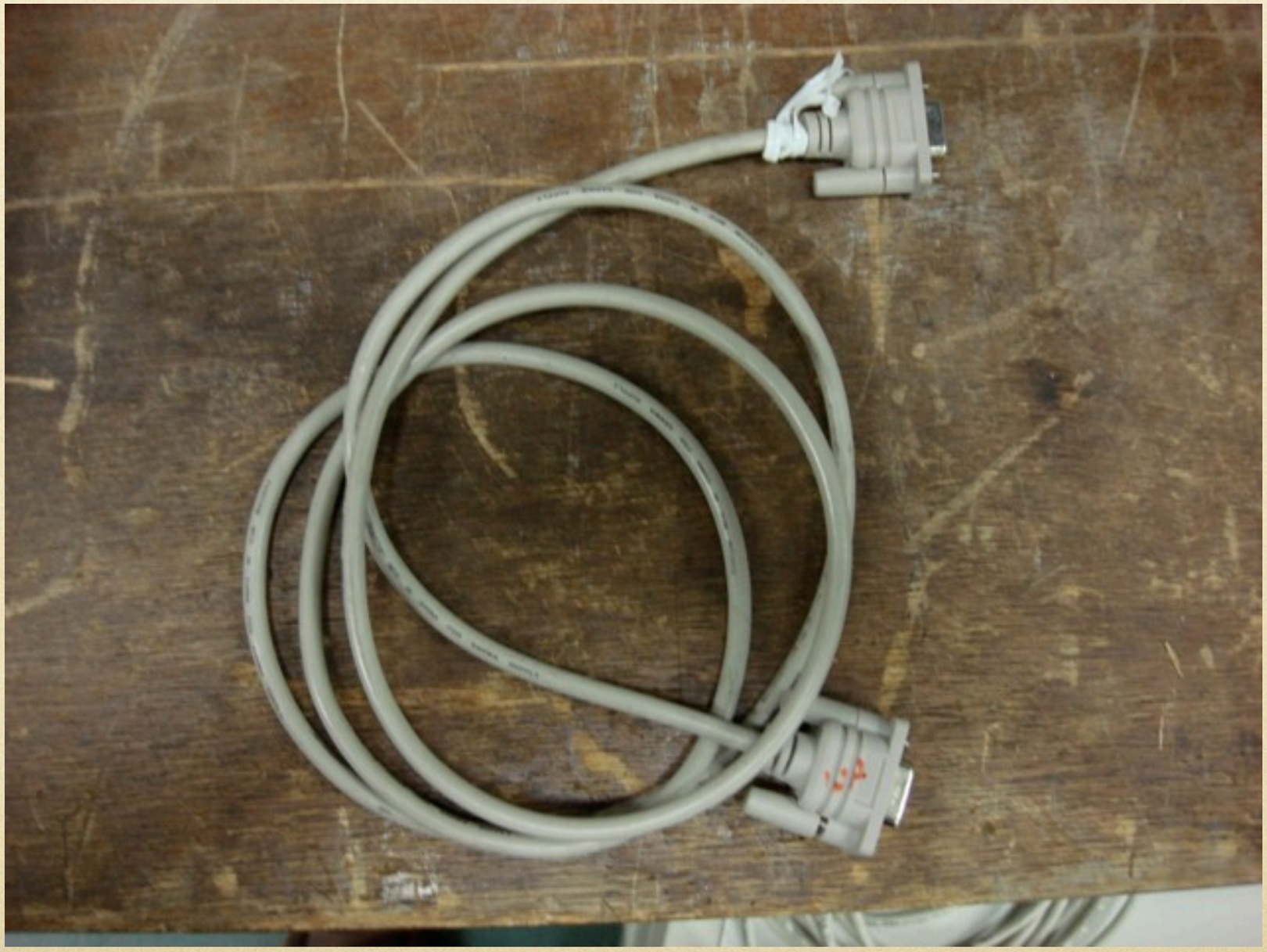
- 一番簡単な方法：
  シリアル端子にケーブルを接続

シリアルコンソール

- 一番簡単な方法：
  シリアル端子にケーブルを接続

シリアルコンソール

- DSUB9p のクロスケーブル



シリアルコンソール

- DSUB9p のクロスケーブル

シリアルコンソール

- 2台のマシンのシリアルインタフェースを、クロスケーブルで接続。
- インタフェースが2個ある場合は、クロスカップル接続しても良い。
  マシンAのCOM1をマシンBのCOM2へ、
  マシンBのCOM2をマシンAのCOM1へ。
  どっちがコケても対応できる
- アクセス先マシンの /etc/ttys を編集

```
ttyd0   "/usr/libexec/getty std.9600"   dialup  off secure
    ↓
ttyd0   "/usr/libexec/getty std.9600"   dialup  on secure
```

- 編集したら kill -HUP 1

シリアルコンソール

- アクセス元で次のコマンド：

  % tip com1

- ログインプロンプトが出る！
- 終わりたい時は “~.”を入れる。

シリアルコンソール

- 何をやってるのか
  - ttys の編集は、ログインプロンプトを出す先としてシリアルインタフェースを指定するのに必要
  - デフォルトでは、ディスプレイ出力以外無効になっている
- アクセス元からは、単にシリアル通信プログラムを起動しただけ。

シリアルコンソール

- キーボードで直接操作しているのと
  基本的に同じことが実現できる。
- ネットワークが停止してても問題なし。
- ケーブルの種類：
  - クロスケーブル（リバースケーブル）
    結線の種類がいくつかある。
    フロー制御(CD線等)まわり。
    3-wire (RX/TX入れ替え+GND)でも
    十分動作

シリアルコンソール

- ハードウェアの操作はどこまでできる？
  - 製品に依存する。
  - サーバ向けのものでは、
    電源制御ができたり、
    BIOS画面をシリアル側に表示することが
    できるものが存在
  - OSが起動してからの機能性（ソフトウェア
    の操作）は、製品に依らず同じ

シリアルコンソール

- ソフトウェアの操作はどこまでできる？
  - ログインができる（前述）
  - OSが完全に起動していなくても、起動ローダのレベルで操作ができる
    → FreeBSDの起動ローダがシリアル通信に対応しているため
  - OSを強制停止させて、カーネルデバッガの操作をすることもできる
    → 反応がなくなったり、panicが発生しても対応できる可能性あり

シリアルコンソール

- 起動ローダ等、すべてをシリアル経由で操作するための設定方法
  - /boot.config という名前で”-Dh”だけが含まれるファイルを作成
  - /boot/loader.conf に
    console=”comconsole”
    beastie_disable=”YES”
    を追加
  - これで、起動の時からの画面出力がすべてシリアル側に流れるようになる
  - 注: boot0 の段階は出てこない。出したければ/boot/boot0sioをインストールすること

シリアルコンソール

- 実際の例をみてみましょう。

シリアルコンソール

- いざという時に
  シリアルから強制的にOSを止めるには
- カーネルデバッガを入れて、あるキー操作で
  OSが止まるオプションを設定する。
- options DDB
  options KDB
  options GDB
  options ALT_BREAK_TO_DEBUGGER
  options KDB_UNATTENDED
- [CR] [~] [^b] でbreakがかかる
- panic すると ddb プロンプトに落ちる

シリアルコンソール

- 実際の例をみてみましょう。

シリアルコンソール

- OSのコンソール出力を全部シリアルに出す
- DDBを有効にして、いざという時にOSを止められるようにしておく
- キーボードとディスプレイがなくても、たいていの作業はできるようになる
- SSHと違うのは
  - OSのTCP/IP設定に頼らない
  - OSが完全に起動していることに頼らない（ただし、ソフトウェアには依存している）
- シリアル over FireWireも可能(今回は省略)

シリアルコンソール

- OSのコンソール出力を全部シリアルに出す
- DDBを有効にして、いざという時に
  OSを止められるようにしておく

注意：

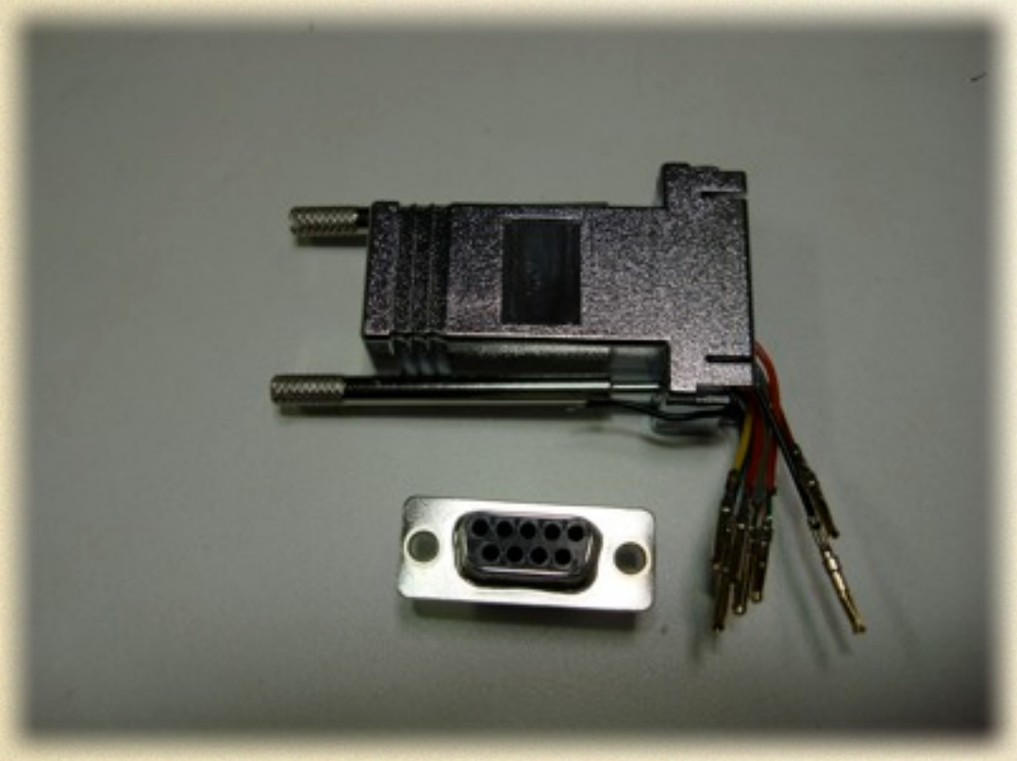
USB-シリアル変換など
OSが初期化しないと
動かないものは、
コンソール出力用途には
使えません。

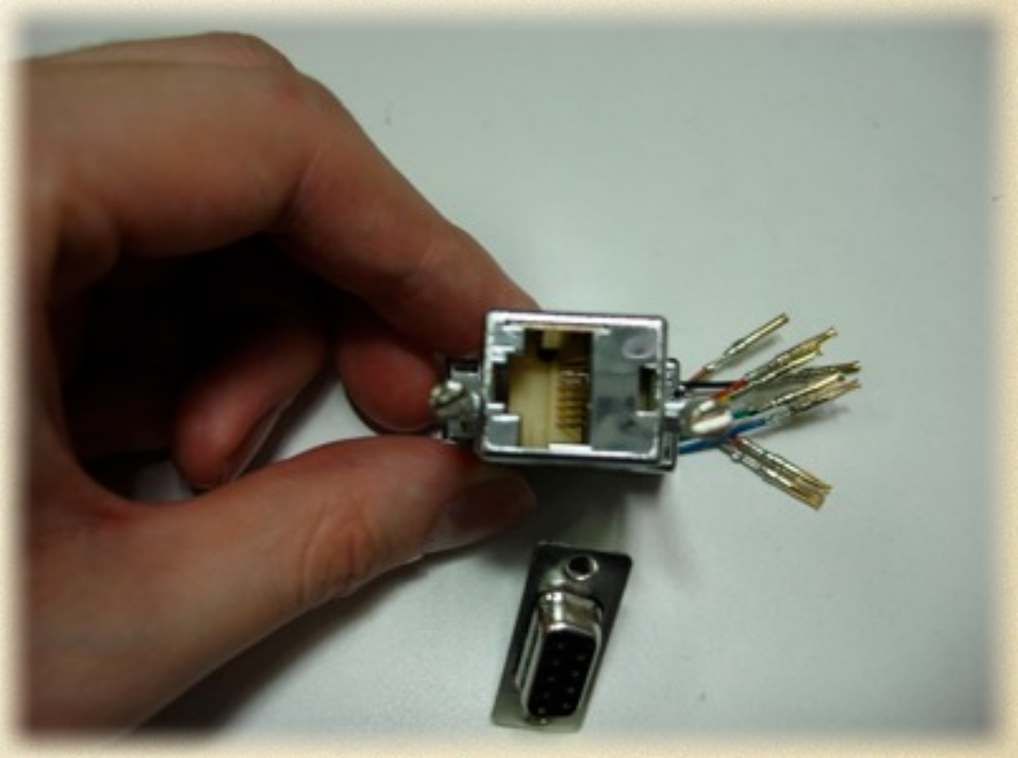
アクセスする側としては
使えます。

シリアルコンソール

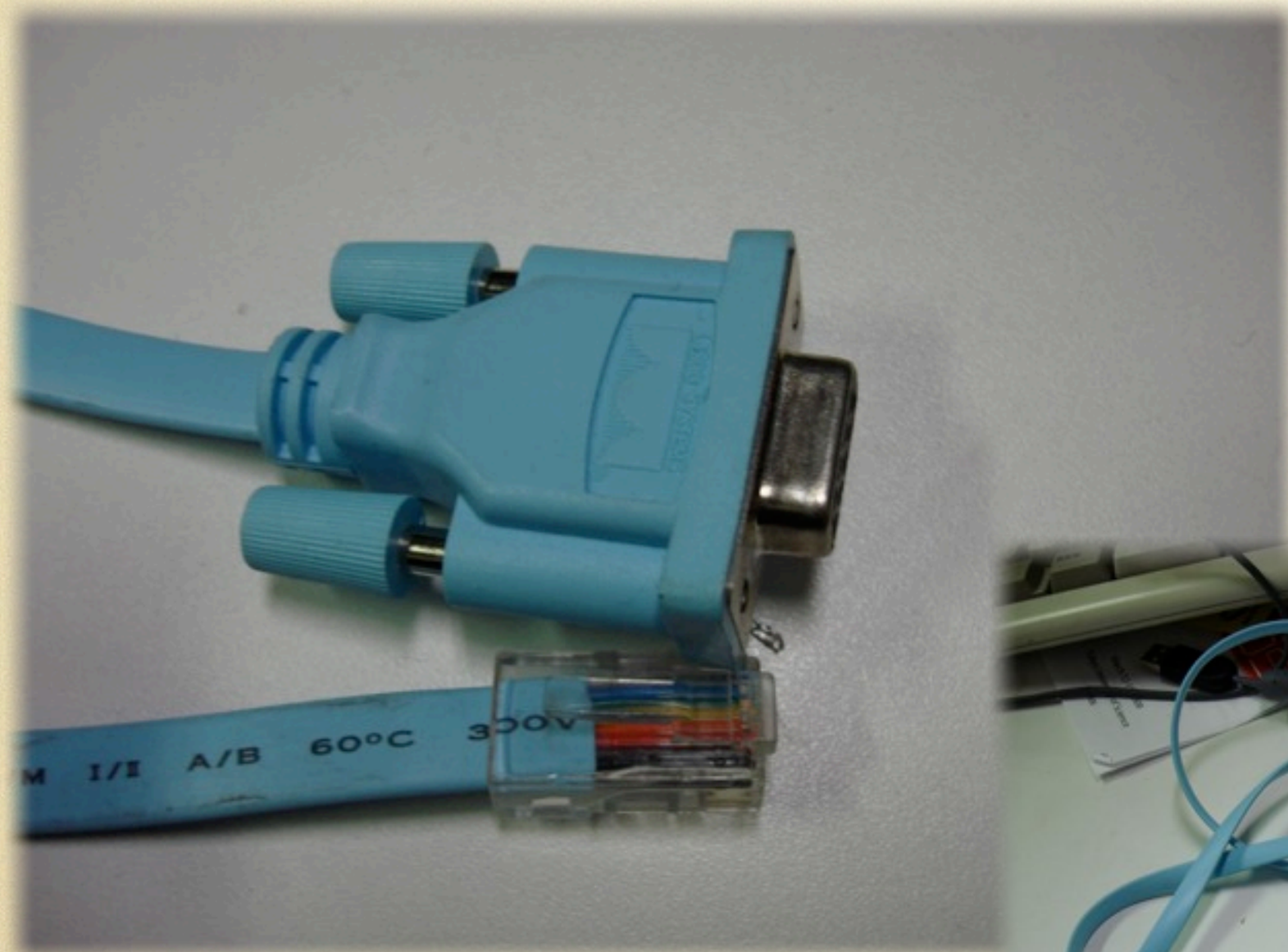
- ケーブルの管理
  - たくさんのマシンになってくると、管理が大変
  - クロスケーブルは重くて種類も少ない
  - RJ-45 to DSUB変換アダプタがおすすめ

RJ-45-DSUB変換





RJ-45-DSUB変換



シリアルコンソール

- シリアルコンソールの集約
  - マシンがたくさんあると、アクセス元には、シリアルインタフェースがたくさん必要
  - USB-シリアルをたくさん付ける
  - PCI-シリアルとか付けてみる
  - あるいは、昔使われていたアナログモデム用ダイヤルアップアクセスサーバを使うという手もある

シリアルコンソール

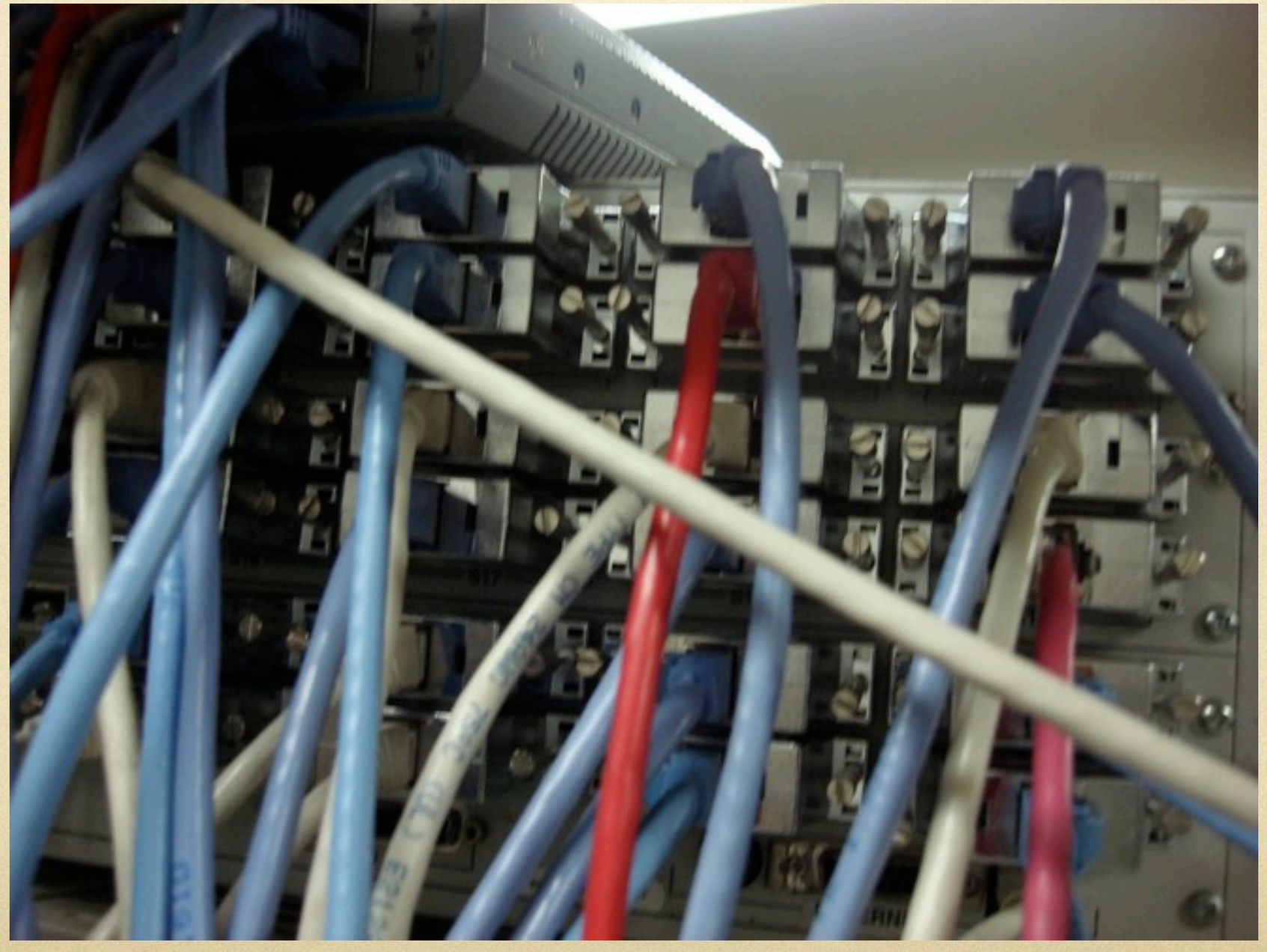
- ダイヤルアップアクセスサーバ：
  ダイヤルアップ接続を処理する専用箱
- モデム接続用のシリアルポートがいっぱい
- Ethernet でアクセス可能
- 普通はモデムをつないで、そこからのアクセスを受けるためのものだが、逆にEthernetからシリアルポートにつなぐこともできる。

シリアルコンソール

- PortMaster 2E (最大30ポート), 10BASE-T
- 3-4万円で買えます(www.portmasters.com)

シリアルコンソール

- 30台分アダプタ付きで配線するとこんな感じ



シリアルコンソール

- 10 ポート版。2Uです。

シリアルコンソール

- シリアルコンソールの集約
  - 物理的に接続できても、tip(1)は使いにくい
  - console(1)というツールが便利 (comms/conserver-com)
  - いろいろな接続を、一元化管理できる (後述)

リモートコンソールアクセス

- さて、DDBへのbreakも効かないくらい状況になったらどうしよう？
- breakは異常な状態でも効くようになっているが、OSが完全に死んでしまうとアウト
- ソフトウェアに頼らない方法はないのか
  - 専用ハードウェア
  - シリアル経由でマシン固有の操作ができるもの
  - シリアル over IPMI over LAN

専用ハードウェア

- リモート管理ソリューションとして販売されているものはたくさんある。
- リモートKVM:
  キーボードとディスプレイの信号をIP経由で遠隔地へ飛ばすもの。電源制御付きもあり。
  （写真はAvocent DSR1021）

専用ハードウェア

- リモート管理ソリューションとして販売されているものはたくさんある。
- シリアルコンソール箱：
前述のケーブルによるシリアル接続を
集約できる専用の機械。TCP/IPで接続できるものが多い。（写真はAvocent Cyclades CS8)

専用ハードウェア

- 電源制御箱：
  電源のオン／オフを制御できる専用の機械
- シリアルやTCP/IPで接続できるものが多い。
- 写真はAvocent Cyclades PM20

専用ハードウェア

- 残念ながら、概して高価(十万〜百万程度)
- KVMはWindowsサーバなど、
  GUIが必須なものでなければ必要ない
- シリアルコンソール箱と、電源制御箱は、
  あるとかなり便利
- サーバグレードのマシンは、そもそも
  似たような機能を持っているものも多い

専用ハードウェア

- おかしな機械もたくさんあります

  → リセット端子だけ制御する箱

専用ハードウェア

- おかしな機械もたくさんあります

  → リセット端子だけ制御する箱

どこに投資するか？

- 「安い、遠隔管理を考慮していないPC」と「高価な遠隔管理機器」の組み合わせは、結局高くつく？（用途や要求にも依る）
- マシンにその遠隔管理機能が載っているものを選ぶのが安心かも
- 違いは高々、3-4万円/台?

どこに投資するか？

- Sun
  - LOM, RSC, ALOM などの機能があればハードウェアが操作できる。
  - シリアルコンソールはOpenFirmwareが標準対応。

どこに投資するか？

- HP:
  - x86系：Integrated Lights-Out
    （いろいろ名前があってややこしい）
  - IA-64系(zx2000など)は EFI Shell が標準でハードウェア管理に対応している

どこに投資するか？

- Dell
  - x86系：DRAC
    (Dell Remote Access Controller)

ハードウェア管理機能の標準

- 近年のx86サーバは、独自方式に加えてIPMIに対応するものがほとんど
- Intelligent Platform Management Interface
- Dell, Intel, HP, NEC が提唱
- OSに依存しない、ハードウェア操作のためのフレームワーク

（x86以外は、また別にいろいろあります）

IPMI

- 何ができるの？
  - 電源制御
  - ハードウェア障害ログや
    各種センサ情報の取得
  - BIOS情報の取得・設定
  - LAN経由でのアクセス(IPMI over LAN)
  - SNMP経由でのアクセス
  - シリアルコンソールアクセス

IPMI

- 必ずしもすべての機能が使えるとは限らないので注意。IPMI対応と書いてあったら、何が使えるのかベンダに確認しましょう。
- IPMIの機能は、クライアントソフトウェアからアクセスします。
- アクセス方法は
  1. 直接該当マシン上から
  2. シリアルポートから
  3. LAN経由から

  あたりが良く使われます。

IPMI

- アクセスしてみよう！
- アクセス用のツール
  → ipmitool(sysutils/ipmitool)
- FreeBSD上から直接そのマシンにアクセスするには、ipmi(4)を組み込まないとダメ
  → # kldload ipmi
- 認識されれば、こんな表示が出ます

```
ipmi0: <IPMI System Interface> port 0xca2-0xca3 on acpi0
ipmi0: KCS mode found at io 0xca2 on acpi
```

IPMI

- # ipmitool sdr

  → /dev/ipmi0 経由でセンサ情報がとれます
- % ipmitool -I lan -U user -P pass -H ipaddr sdr

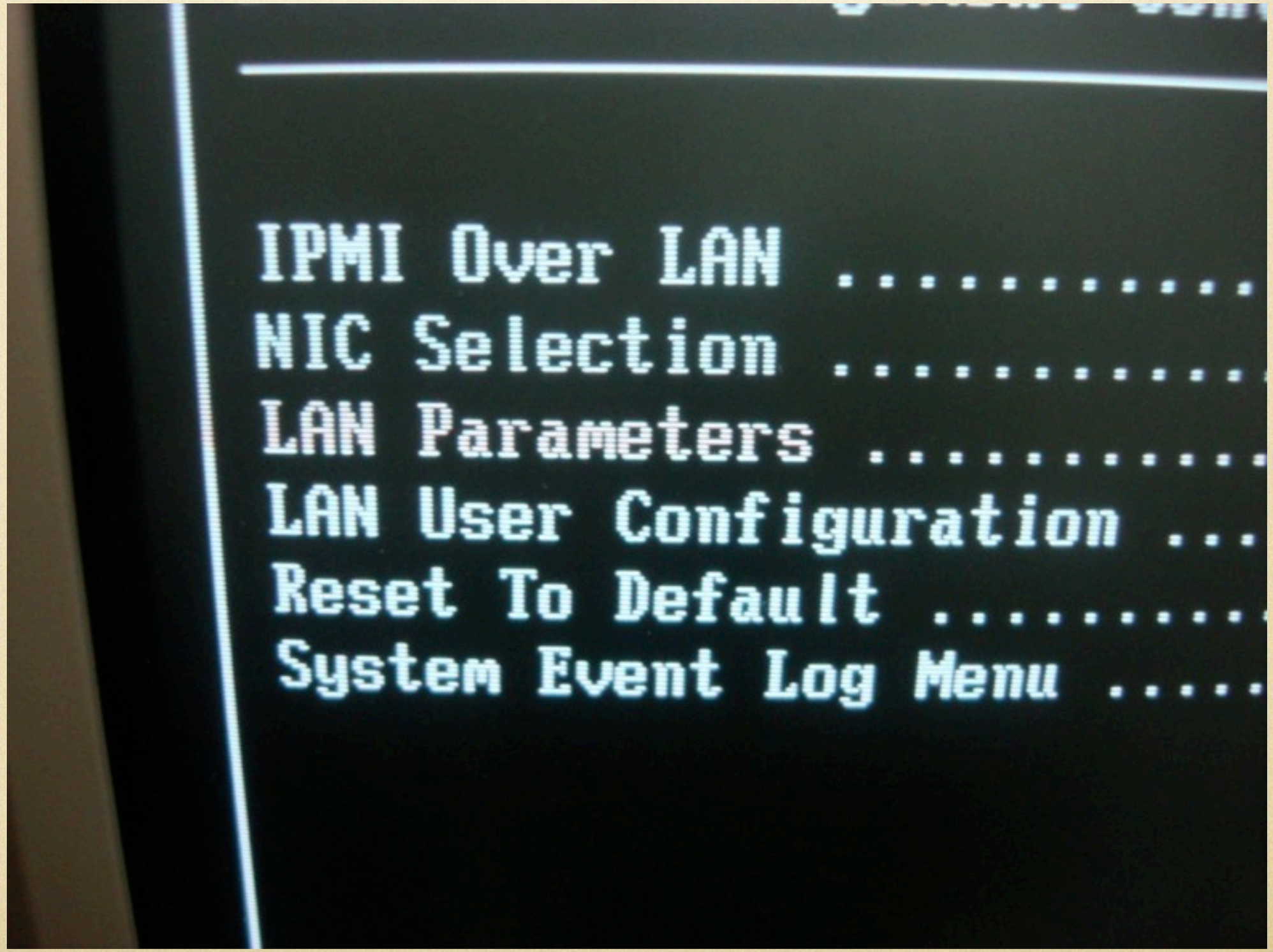
  → LAN経由でセンサ情報がとれます
- % ipmitool -I lan -U user -P pass -H ipaddr power on

  → LAN経由で電源がONできます

IPMI

- 使う前の基礎知識と準備
- IPMIは、BMC (Baseboard Management Controller) という、マシン上の制御部分と通信するための規格です
- LAN経由でも使えますが、ユーザ名、パスワード、IPアドレスを事前に設定しておかなければいけません。
- マシン上から/dev/ipmi0経由で設定できますが、BIOS画面で設定できることが多いです。

IPMI


IPMI Over LAN .....
NIC Selection .....
LAN Parameters .....
LAN User Configuration ...
Reset To Default .....
System Event Log Menu .....

IPMI

- どのNICをIPMI用につかうのかは、自分の環境とマシン構成で考えましょう
  - IPMIは623/udpを使います
- NICを普通の通信と共用することも、別個に専用のNICを持たせることも可能。別にするとセキュリティや信頼性を高めることができます。

IPMI

IPMI

- シリアル over IPMI
- # ipmitool -I lanplus -U u -P p -H ip sol activate
  → LAN経由でシリアルポートにアクセス
- シリアルケーブルを接続する必要なし！
- 電源制御もすべてLANでできる

コンソールアクセスの集約

- コンソールアクセスは便利だが、アクセス元を集約しないと管理が大変
- コンソールアクセスサーバをつくる
- comms/conserver-comが便利

コンソールアクセスの集約

- 基本的な使い方
  - /usr/local/etc/conserver.cf に、
    コンソール定義を列挙
  - /etc/rc.conf に conserver_enable=”YES”
  - /usr/local/etc/rc.d/conserver start
  - % console <名前>
    とすると、tip と同じようにアクセスできる

conserver.cf

```
access * {
        admin hrs;
        allowed 127.0.0.1;
}
config * {
        defaultaccess   rejected;
        daemonmode      yes;
        logfile         /var/log/conserver.log;
        passwdfile      /usr/local/etc/conserver.passwd;
        reinitcheck     1;
}
# ---------------------------------
default * {
        master 127.0.0.1;
        baud 9600;
        parity none;
        rw hrs;
        logfile "/var/log/conserver/&.log";
        logfilemax 5m;
}
console A {type device; device /dev/cuac00; }
```

conserver.cf

シリアルポートの場合

console A {type device; device /dev/cuac00; }

TCP接続の場合

console B {type host; protocol telnet; host 192.168.180.1; port 23; }

SSH接続の場合

console C {type exec; protocol raw; exec “/usr/bin/ssh -oGlobalKnownHostsFile=/var/conserver/known_hosts -oidentityFile=/usr/local/etc/conserverkey user@hostname”; }

IPMI接続の場合

console D {type exec; protocol raw; exec “/usr/local/bin/ipmitool -l lanplus -H hostname -U user -P pass sol activate”; }

→　どんな接続方法でも、console <名前> で接続できる。どこに何が接続されているのか意識する必要なし。

コンソールアクセスの集約

- conserverを使えば、
  コンソールのログも収集できるので、異常停止した時の状況が調べやすい。
- ^E c がエスケープ。^E c dで終了。
  ^E c ? でヘルプ。
- hubやルータなど、その他にCLIで設定できる機器へのアクセスも、全部集約できます。
- 実際の例をみてみましょう

まとめ

- SSHを有効にすると
  - リモートからログインしてシステム管理ができる
- シリアルコンソールを整備すると、
  - ネットワーク機能が死んでいてもログインできたり、breakをかけてOSを強制的に再起動させることができる
- IPMI対応のマシンを用意すると
  - 電源オフ状態からの起動や、BIOS操作などもできるようになる。
  - LAN経由(OSのネットワーク機能と独立)で管理可能。

まとめ

- シリアルコンソール＋電源制御
  - PXE起動を駆使すれば、リモート環境で新規インストールすることは難しくない。
  - stable storage が必要ないなら、ディスクレスでネットワーク起動させてサーバにするという運用もある。
    → 代替機の調達／起動が簡単
- 信頼できるネットワークを作ることを忘れずに
  - VLANを切る、経路を暗号化する
  - hub・スイッチの故障や電源断が発生するとコンソールアクセスが失われる

障害のパターンと対応

- OSに関わる障害は、リモートコンソールがあると対応できることが多い
- 典型的な4つのパターンを簡単に紹介

障害のパターンと対応

- シングルユーザモードで起動した場合は、リモートコンソールで対応できる
  - fstab の書き間違いや、ディスク装置の故障など、多くの場合にシングルユーザモードで止まる
    - fsck -y と mount -t ufs -a でディスクをマウント
    - /etc/netif start でネットワーク設定
  - 修正したら再起動させる
- シングルユーザモードは、カーネルが起動してから rc.d スクリプトが正常完了するまでの間のこと。カーネルの動作状態が特別なものになっているわけではない。

障害のパターンと対応

- システムそのものは動いているが、挙動がおかしい
  - たとえば、umount したら反応が返ってこなくなった、など。俗に「刺さる」と呼ぶことがある。周辺デバイスの異常動作による場合は、再起動しないと元に戻らないことが多い。
  - メモリを食いつぶすプログラムをつくって実行すると、swap処理でシステムが反応できなくなったりする。
  - reboot コマンドを実行しても再起動してくれない、という状況もあり得る。
- 早急に再起動が必要であれば、break をかけて強制再起動させてしまう。

障害のパターンと対応

- panic メッセージが出た
  - OSのバグ、あるいはroot権限での不適切なオペレーションで出ることがある。
  - panic は、OS内部で想定外の異常動作が検出されたことを意味する。なので、普通は出ない。
  - 特定操作でpanicが再現する場合は、なるべく開発者にフィードバックしてほしい。
    バグだった場合、レポートがないといつまでも直らない可能性がある。
- シリアルコンソールを使っているならば、panic 時のメッセージをコピーしてコミュニティに送ると良い。

障害のパターンと対応

- OSが何の反応もしなくなる
  - 頻繁にbreakも効かないような状態になる場合は、対応が困難です。
  - どうしても運用の継続が必要なら、watchdog timer を使う手があります。
  - IPMI の他、ichwd (Intel ICHファミリに搭載)を使うこともできます。
  - 詳しくはwatchdogd(8)のマニュアルを参照
  - 反応がなくなった時、一定時間後にハードウェアが自発的にリセットするようになります。

中途半端ですが

- このへんで終了です。
- 質問はありますか？