仮想化 Linux 環境 Docker について

前田 俊行
千葉工業大学 人工知能・ソフトウェア技術研究センター

STAIR Lab

Docker ってなに?

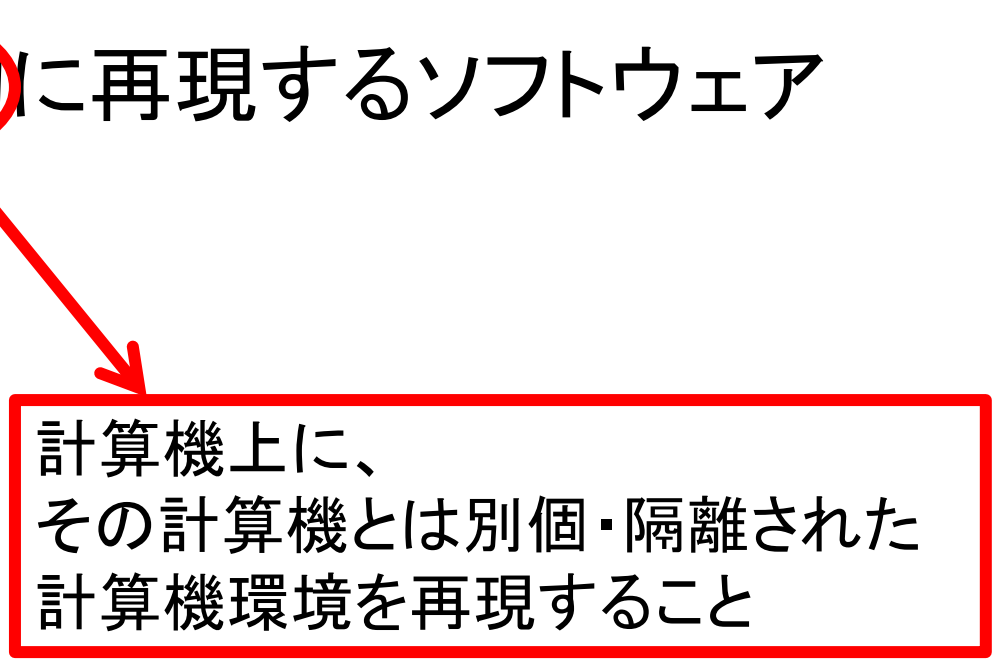
- Linux 環境を仮想的に再現するソフトウェア

Docker ってなに?

- Linux 環境を仮想的に再現するソフトウェア

Docker ってなに?

- Linux 環境を仮想的に再現するソフトウェア



仮想環境? 他にもいっぱいあるじゃん?

- KVM: https://www.linux-kvm.org/
- VMware: http://www.vmware.com/
- QEMU: http://www.qemu-project.org/
- VirtualBox: https://www.virtualbox.org/
- Hyper-V: http://www.microsoft.com/hyper-v
- LXC: https://linuxcontainers.org/
- Xen: https://www.xenproject.org/

…

Docker の特徴

- とても簡単
- とても軽い

Docker の特徴

- とても簡単
- とても軽い

Docker を使ってみよう

Docker をインストールしよう

https://store.docker.com/search?type=edition&offering=community

- 上記ページからダウンロード、
  もしくはインストール手順が確認できます
  - Linux (Ubuntu, Debian, Fedra, CentOS, etc.) 版も
    macOS 版も
    Windows 版もあります

Docker コンテナを実行してみよう

```
% docker run ubuntu cat /etc/lsb-release
```

Docker コンテナを実行してみよう

```
% docker run ubuntu cat /etc/lsb-release
```

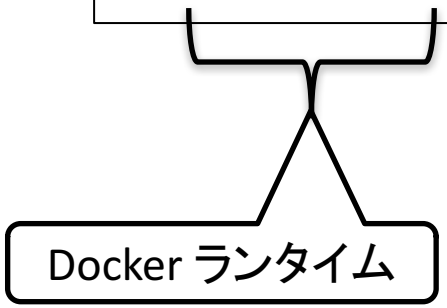
```
…
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04.2 LTS"
```

Docker コンテナを実行してみよう

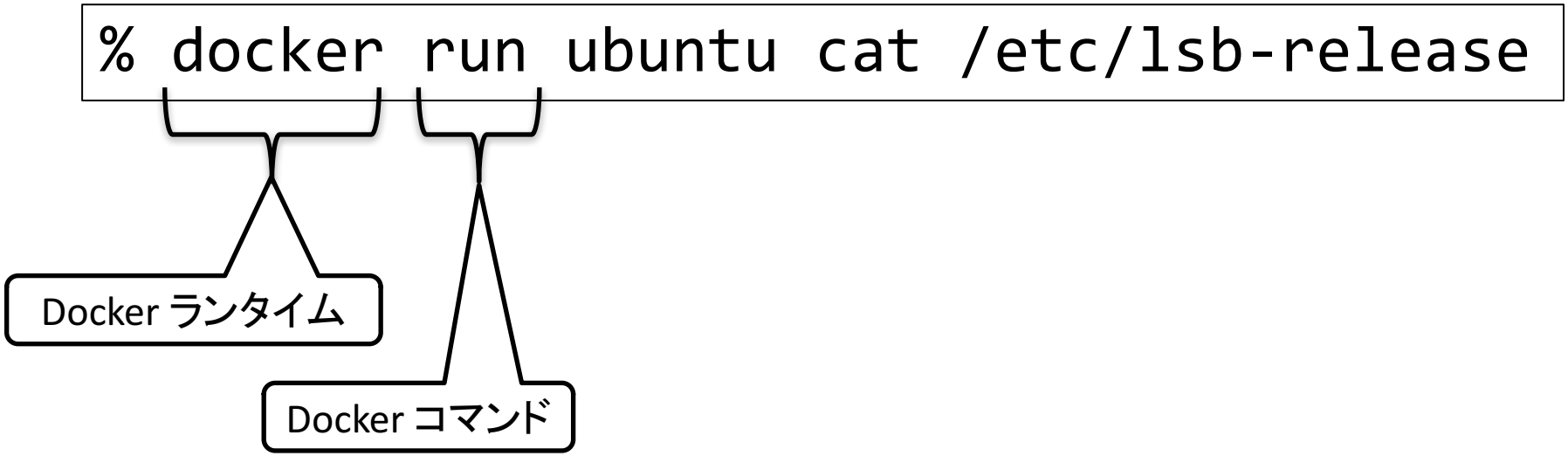
```
% docker run ubuntu cat /etc/lsb-release
```

Docker コンテナを実行してみよう

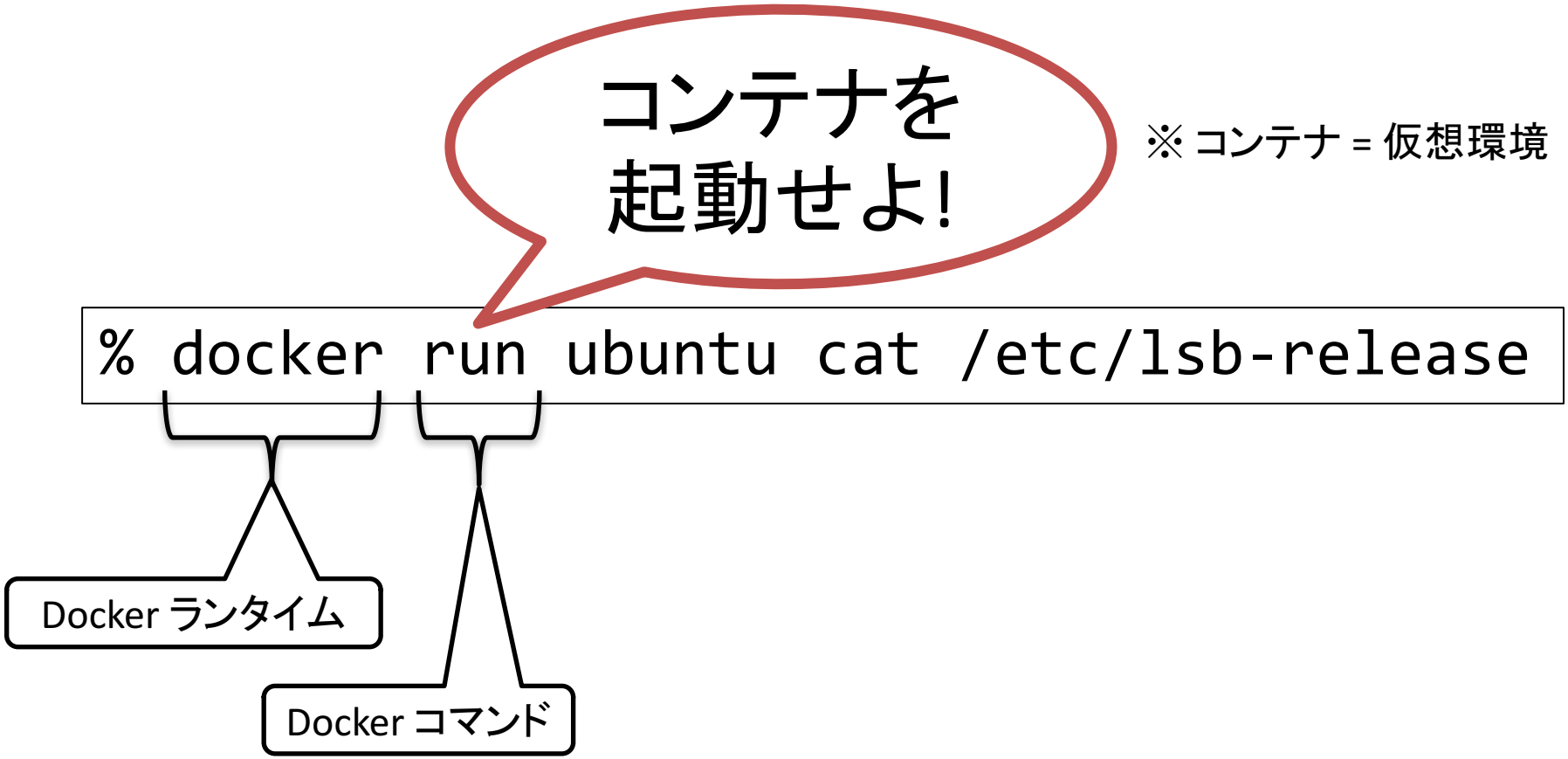
```
% docker run ubuntu cat /etc/lsb-release
```



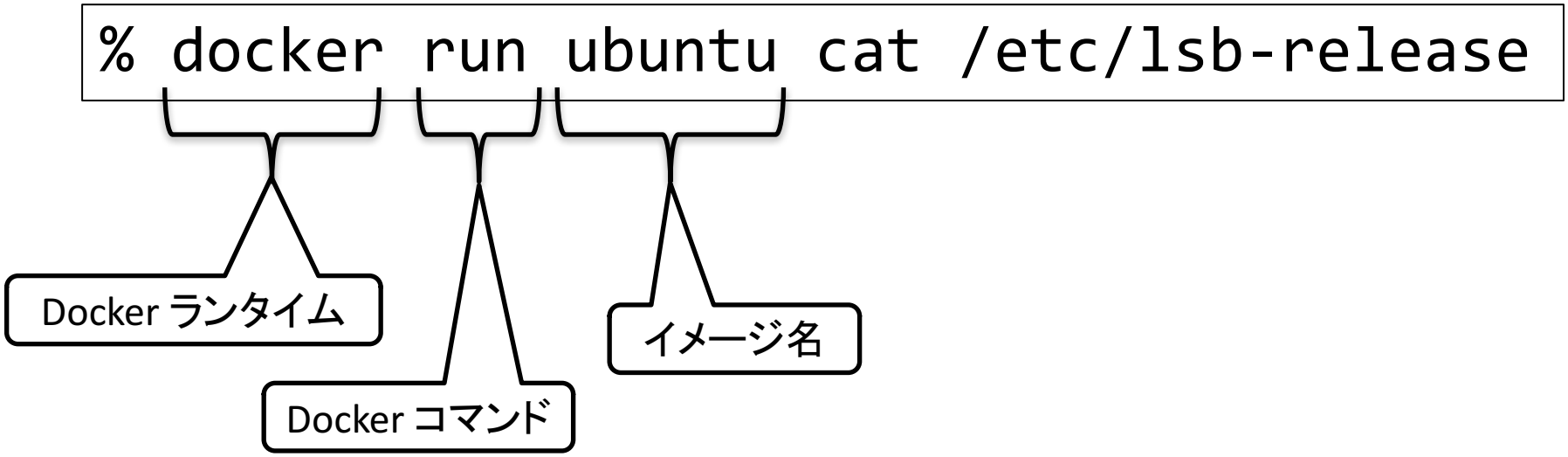
Docker コンテナを実行してみよう



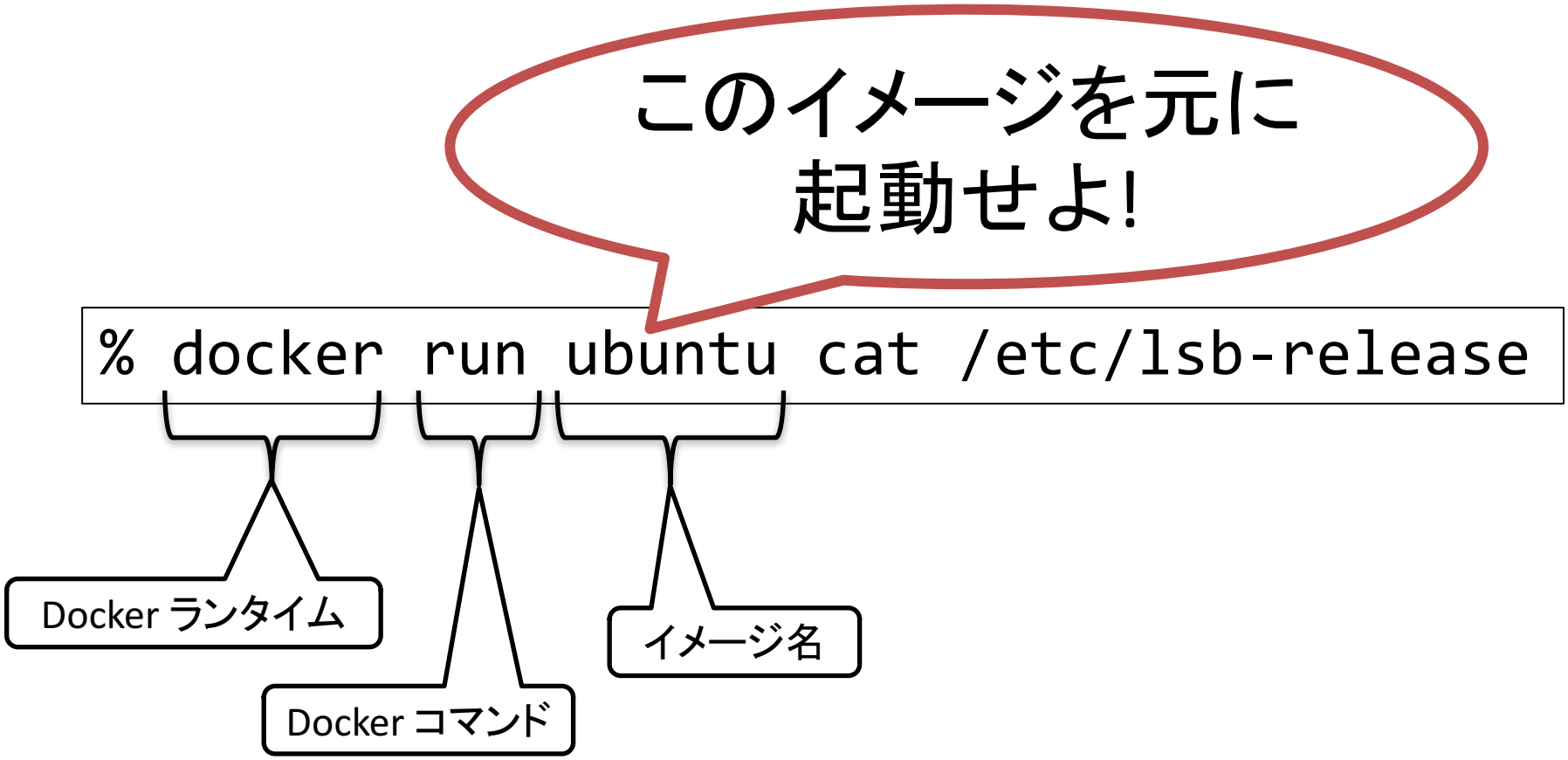
Docker コンテナを実行してみよう



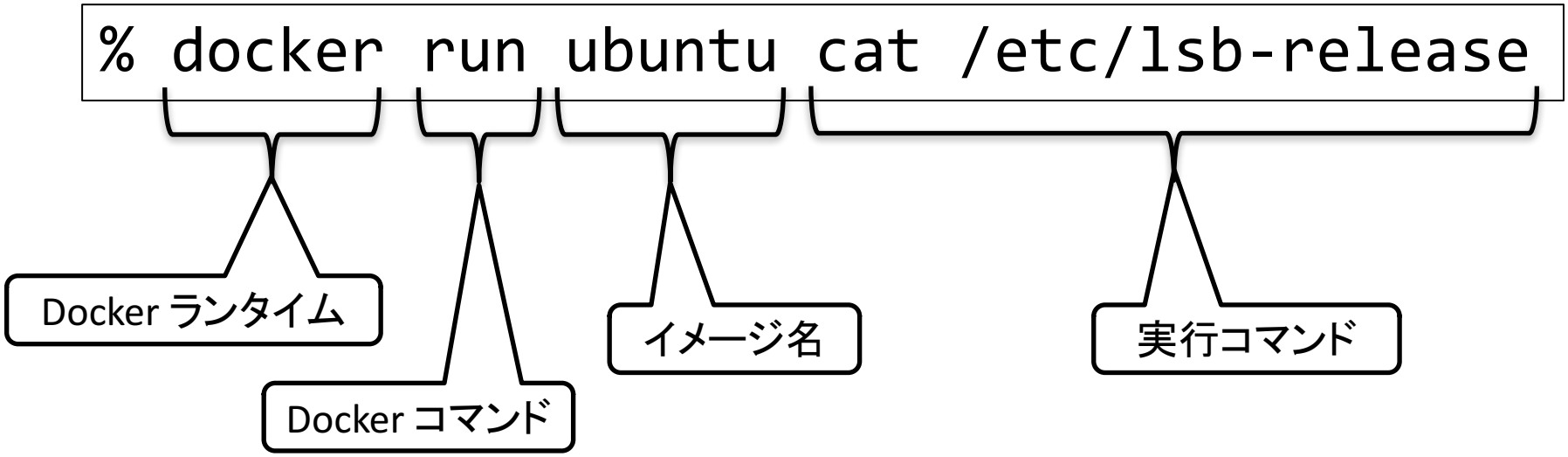
Docker コンテナを実行してみよう



Docker コンテナを実行してみよう



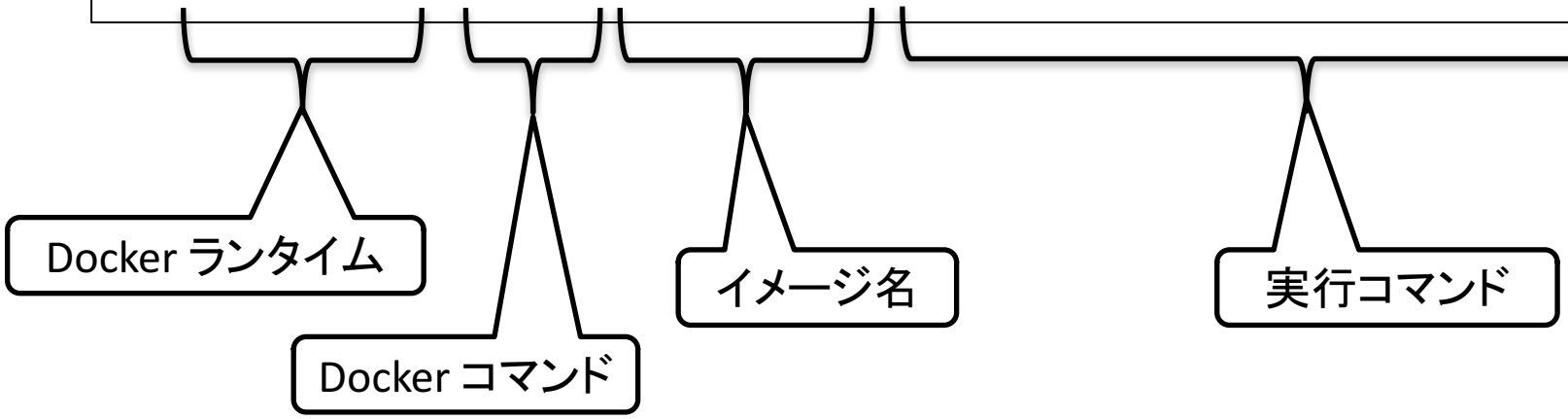
Docker コンテナを実行してみよう



Docker コンテナを実行してみよう

起動したコンテナ内で
このコマンドを実行せよ!

```
% docker run ubuntu cat /etc/lsb-release
```



Docker コンテナを実行してみよう

```
% docker run ubuntu cat /etc/lsb-release
```

Docker コンテナを実行してみよう

```
% docker run ubuntu cat /etc/lsb-release
```

```
…
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04.2 LTS"
```

イメージを探してみよう

```
% docker search centos
```

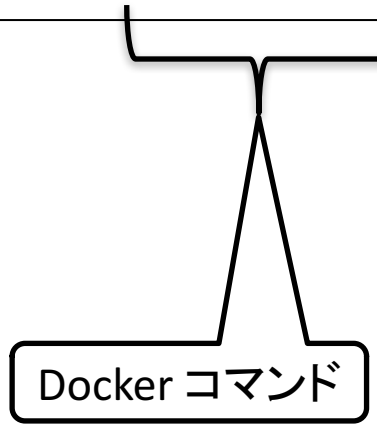
イメージを探してみよう

```
% docker search centos
```

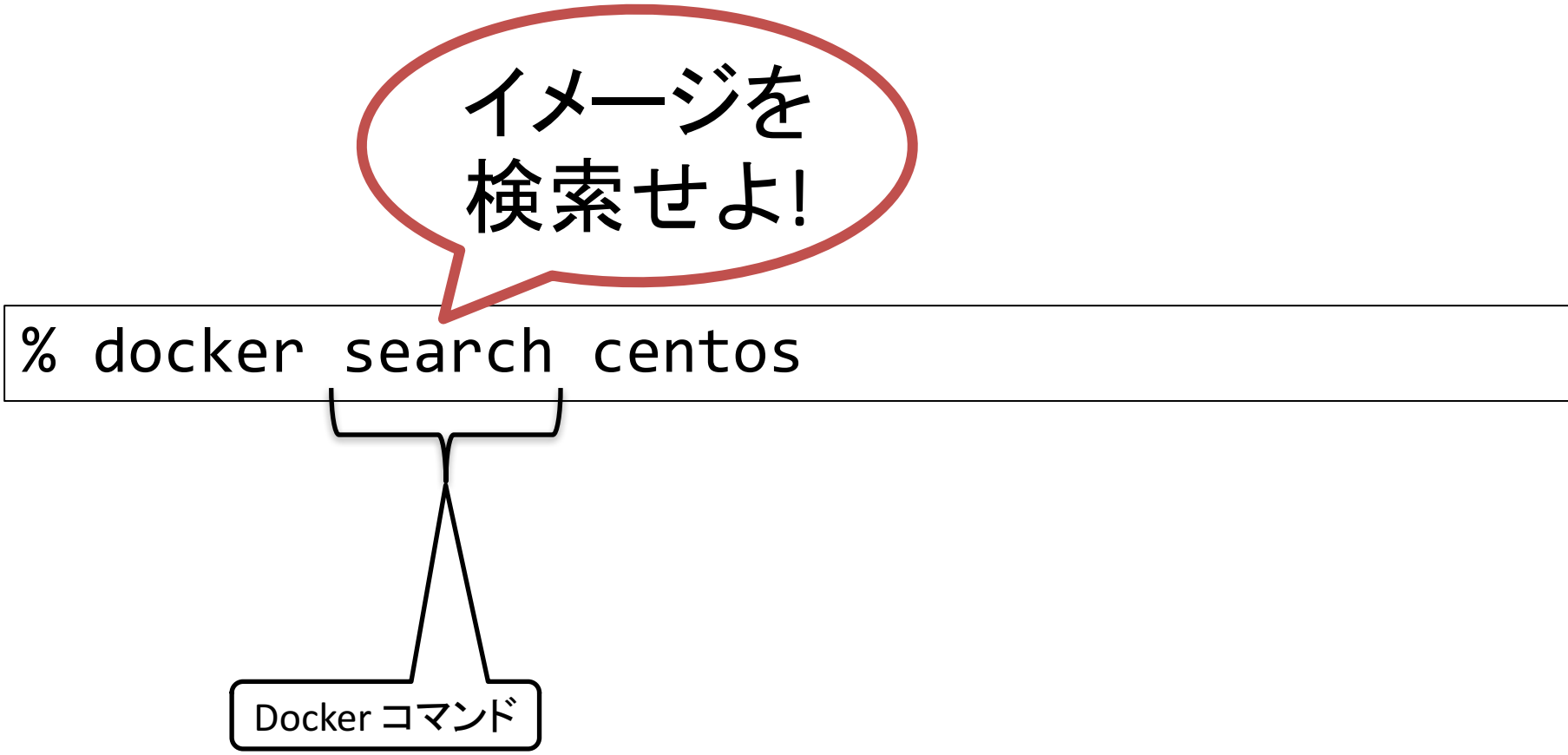
```
DNAME                                DESCRIPTION
centos                               The official b
jdeathe/centos-ssh                   CentOS-6 6.8 x
jdeathe/centos-ssh-apache-php        CentOS-6 6.8 x
consol/centos-xfce-vnc               Centos contair
nimmis/java-centos                   This is docker
…
```

イメージを探してみよう

```
% docker search centos
```

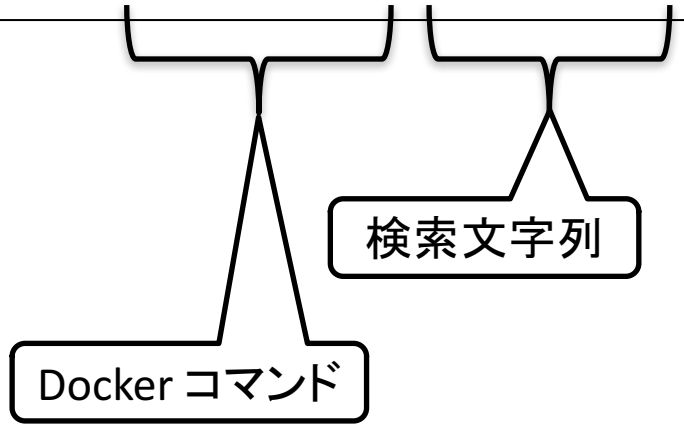


イメージを探してみよう

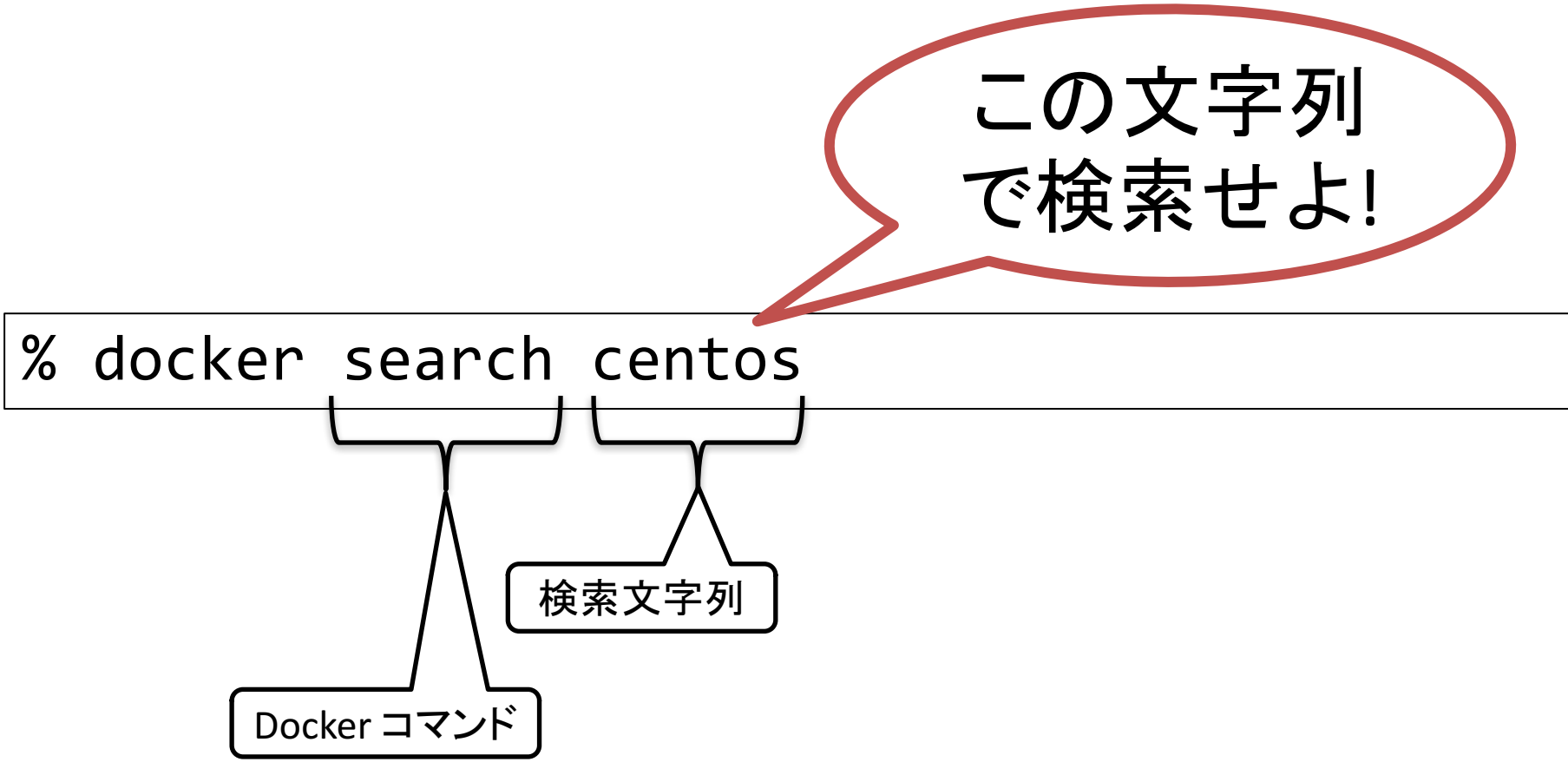


イメージを探してみよう

```
% docker search centos
```



イメージを探してみよう



イメージを探してみよう

```
% docker search centos
```

イメージを探してみよう

```
% docker search centos
```

```
DNAME                                DESCRIPTION
centos                               The official b
jdeathe/centos-ssh                   CentOS-6 6.8 x
jdeathe/centos-ssh-apache-php        CentOS-6 6.8 x
consol/centos-xfce-vnc               Centos contair
nimmis/java-centos                   This is docker
…
```

見つかったイメージを実行してみよう

```
% docker run centos cat /etc/redhat-release
```

見つかったイメージを実行してみよう

```
% docker run centos cat /etc/redhat-release
```

```
…
CentOS Linux release 7.3.1611 (Core)
```

裏で何が起きているのか?

https://hub.docker.com

検索

ネットワーク

結果

```
% docker search centos …
```

Cf. apt of Debian/Ubuntu and yum of CentOS/Fedora

裏で何が起きているのか?

https://hub.docker.com

イメージ
リクエスト
(pull)

ネットワーク

ダウンロード

```
% docker run centos …
```

Cf. apt of Debian/Ubuntu and yum of CentOS/Fedora

Docker Hub: https://hub.docker.com

- 様々な Docker イメージが登録されている
  レジストリ
- 欲しいイメージは大抵すでに存在している
  - ただし「誰でも登録できる」レジストリのため、
    オフィシャルイメージや信頼できるイメージ以外は
    実行しない方がよい

コンテナを実行してみよう (その2)
イメージをダウンロードする

```
% docker pull nginx
```

コンテナを実行してみよう(その2)
イメージをダウンロードする

```
% docker pull nginx
```

```
Using default tag: latest
latest: Pulling from library/nginx
693502eb7dfb: Downloading [=====> ] 50.86 MB/
6decb850d2bc: Download complete
c3e19f087ed6: Download complete
```

コンテナを実行してみよう(その2)
イメージをダウンロードする

```
% docker pull nginx
```

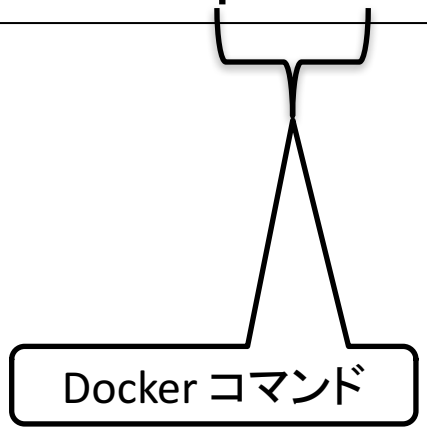
```
Using default tag: latest
latest: Pulling from library/nginx
693502eb7dfb: Pull complete
6decb850d2bc: Pull complete
c3e19f087ed6: Pull complete
Digest: sha256:52a189e49c0c797cfc5cbfe578c68
Status: Downloaded newer image for nginx:lat
```

コンテナを実行してみよう (その2)
イメージをダウンロードする

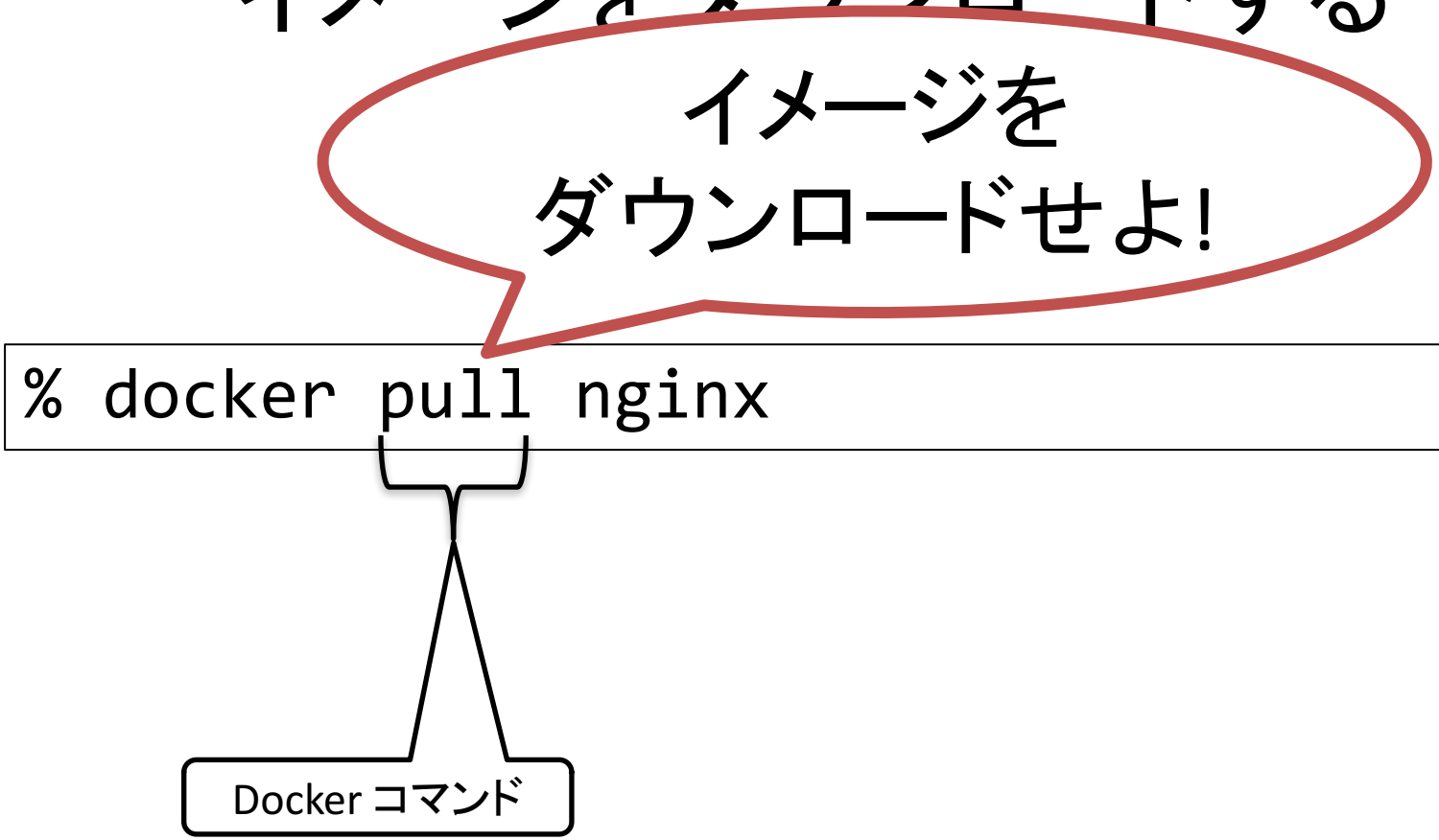
```
% docker pull nginx
```

コンテナを実行してみよう (その2)
イメージをダウンロードする

```
% docker pull nginx
```

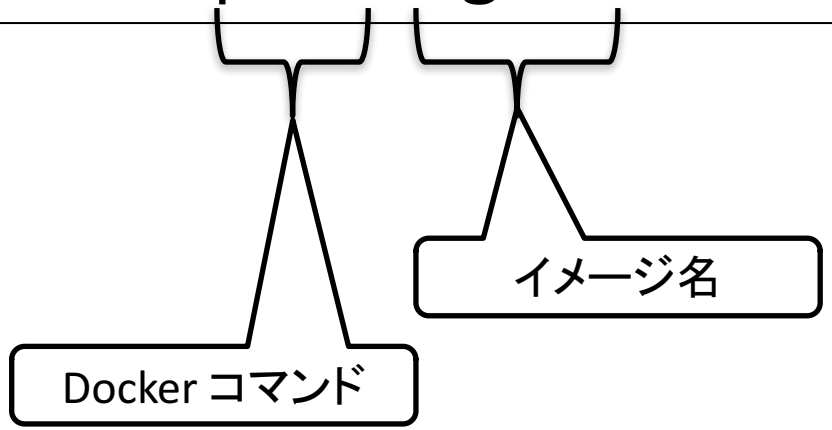


コンテナを実行してみよう (その2)
イメージをダウンロードする



コンテナを実行してみよう (その2)
イメージをダウンロードする

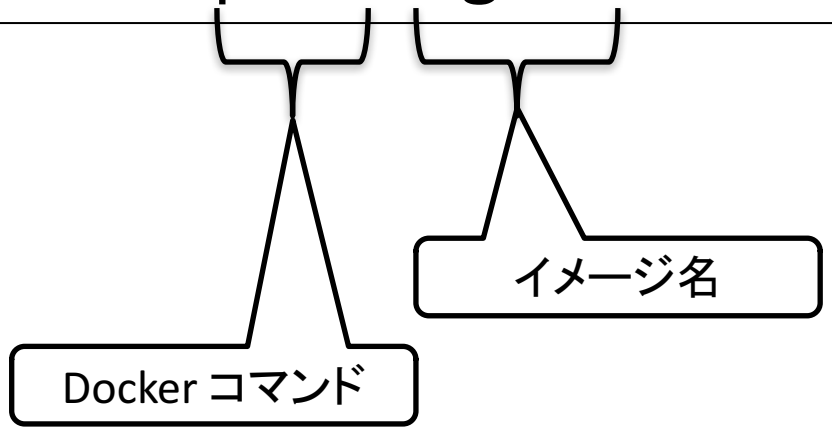
```
% docker pull nginx
```



コンテナを実行してみよう (その2)
イメージをダウンロードする

この名前のイメージを
ダウンロードせよ!

```
% docker pull nginx
```



コンテナを実行してみよう(その2)
イメージを確認する

```
% docker images
```

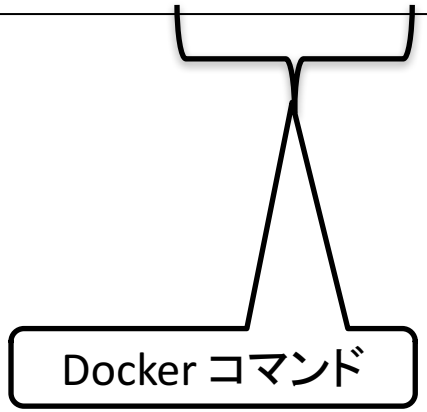
コンテナを実行してみよう(その2)
イメージを確認する

```
% docker images
```

```
REPOSITORY  TAG      IMAGE ID       CREATED
centos      latest   98d35105a391   4 days ago
nginx       latest   6b914bbcb89e   2 weeks ag
ubuntu      latest   0ef2e08ed3fa   2 weeks ag
```

コンテナを実行してみよう (その2)
イメージを確認する

```
% docker images
```



コンテナを実行してみよう (その2)
イメージを確認する

ローカルに存在する
イメージを列挙せよ!

```
% docker images
```

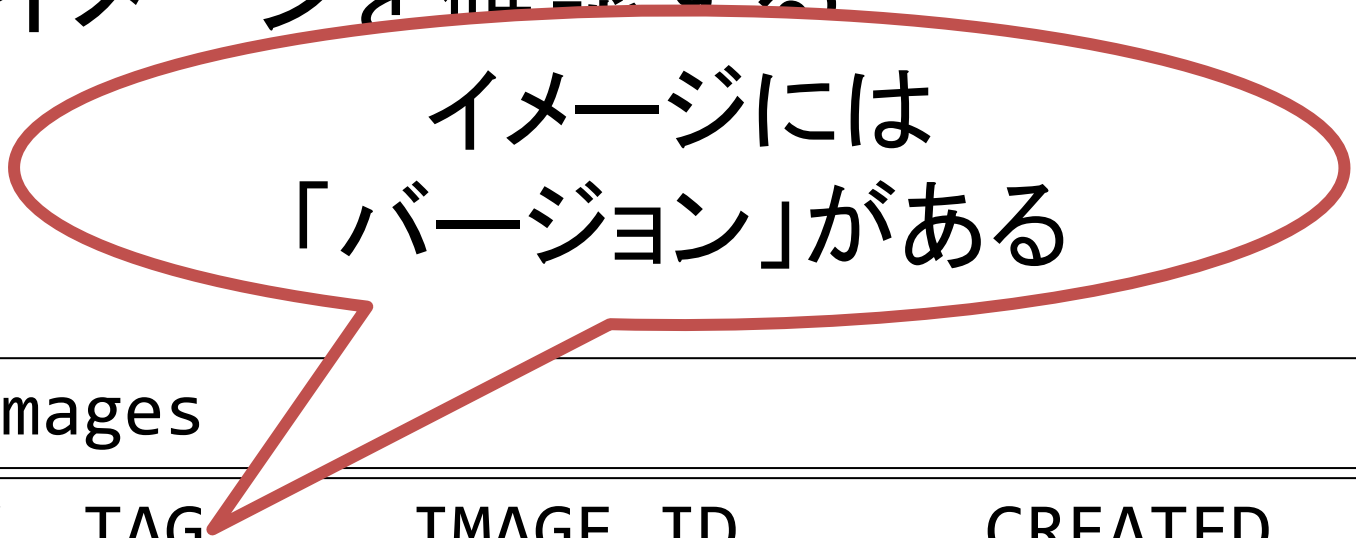
Docker コマンド

コンテナを実行してみよう(その2)
イメージを確認する

```
% docker images
```

```
REPOSITORY  TAG      IMAGE ID       CREATED
centos      latest   98d35105a391   4 days ago
nginx       latest   6b914bbcb89e   2 weeks ag
ubuntu      latest   0ef2e08ed3fa   2 weeks ag
```

コンテナを実行してみよう(その2)
イメージを確認する




```
% docker images
```

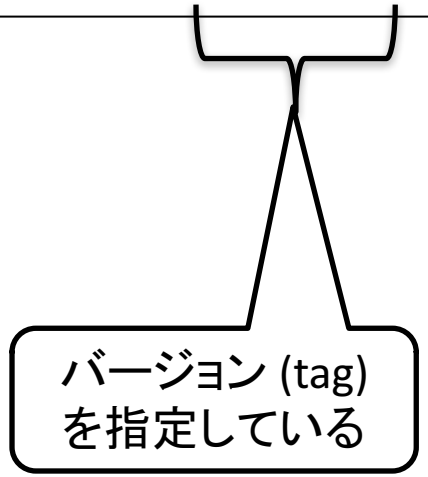
```
REPOSITORY  TAG      IMAGE ID       CREATED
centos      latest   98d35105a391   4 days ago
nginx       latest   6b914bbcb89e   2 weeks ag
ubuntu      latest   0ef2e08ed3fa   2 weeks ag
```

コンテナを実行してみよう(その2)
【ちょっと脱線】
バージョンを指定してみよう

```
% docker run ubuntu:17.04 cat /etc/lsb-release
```

コンテナを実行してみよう(その2)
【ちょっと脱線】
バージョンを指定してみよう

```
% docker run ubuntu:17.04 cat /etc/lsb-release
```



コンテナを実行してみよう(その2)
【ちょっと脱線】
バージョンを指定してみよう

```
% docker run ubuntu:17.04 cat /etc/lsb-release
```

コンテナを実行してみよう(その2)
【ちょっと脱線】
バージョンを指定してみよう

```
% docker run ubuntu:17.04 cat /etc/lsb-release
```

```
…
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=17.04
DISTRIB_CODENAME=zesty
DISTRIB_DESCRIPTION="Ubuntu Zesty Zapus (develo
```

コンテナを実行してみよう(その2)
サーバを起動してみる

```
% docker run -d -p (好きな番号):80 nginx
```

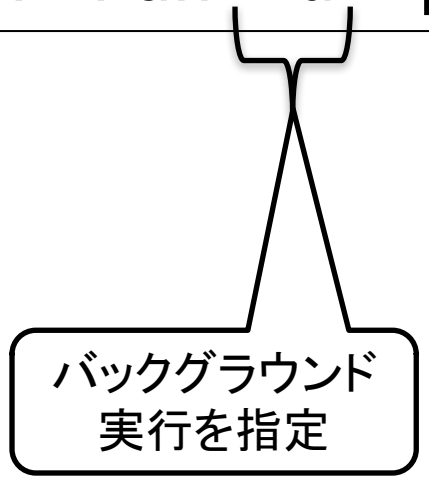
コンテナを実行してみよう(その2)
サーバを起動してみる

```
% docker run -d -p (好きな番号):80 nginx
```

```
ddc41a326fc28951546e3959d7dd552e03be8e82e1a9
```

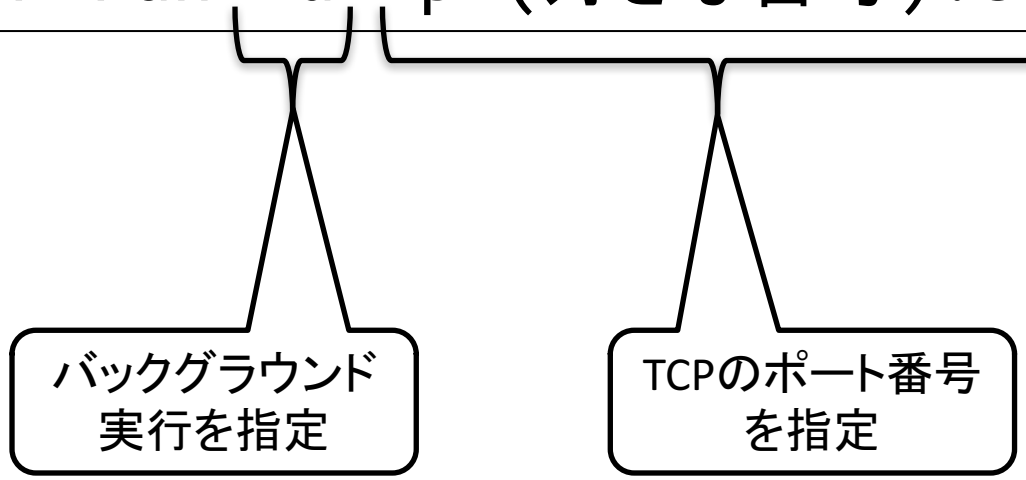
コンテナを実行してみよう(その2)
サーバを起動してみる

```
% docker run -d -p (好きな番号):80 nginx
```



コンテナを実行してみよう(その2)
サーバを起動してみる

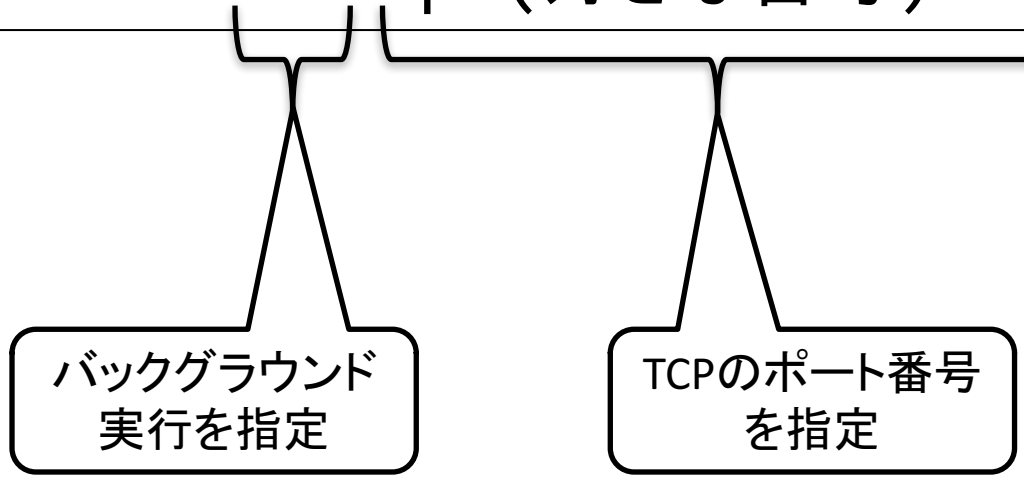
```
% docker run -d -p (好きな番号):80 nginx
```



コンテナを実行してみよう (その2)

ホスト側の (好きな番号) 番ポートを
コンテナの 80 番ポートに接続する

```
% docker run -d -p (好きな番号):80 nginx
```



コンテナを実行してみよう(その2)
サーバを起動してみる

```
% docker run -d -p (好きな番号):80 nginx
```

```
19ecb956ab5d1e5362fe57701ec3a7667cdf1a9e9ba2
```

コンテナを実行してみよう (その2)
サーバが起動しているか確認する

```
% docker ps
```

コンテナを実行してみよう(その2)
サーバが起動しているか確認する

```
% docker ps
```

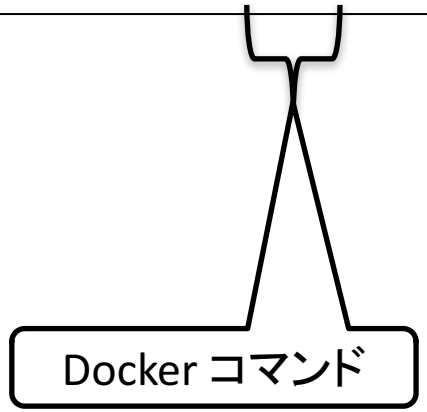
```
CONTAINER ID   IMAGE    COMMAND
19ecb956ab5d   nginx    “nginx -g ‘daemon ...”
```

コンテナを実行してみよう (その2)
サーバが起動しているか確認する

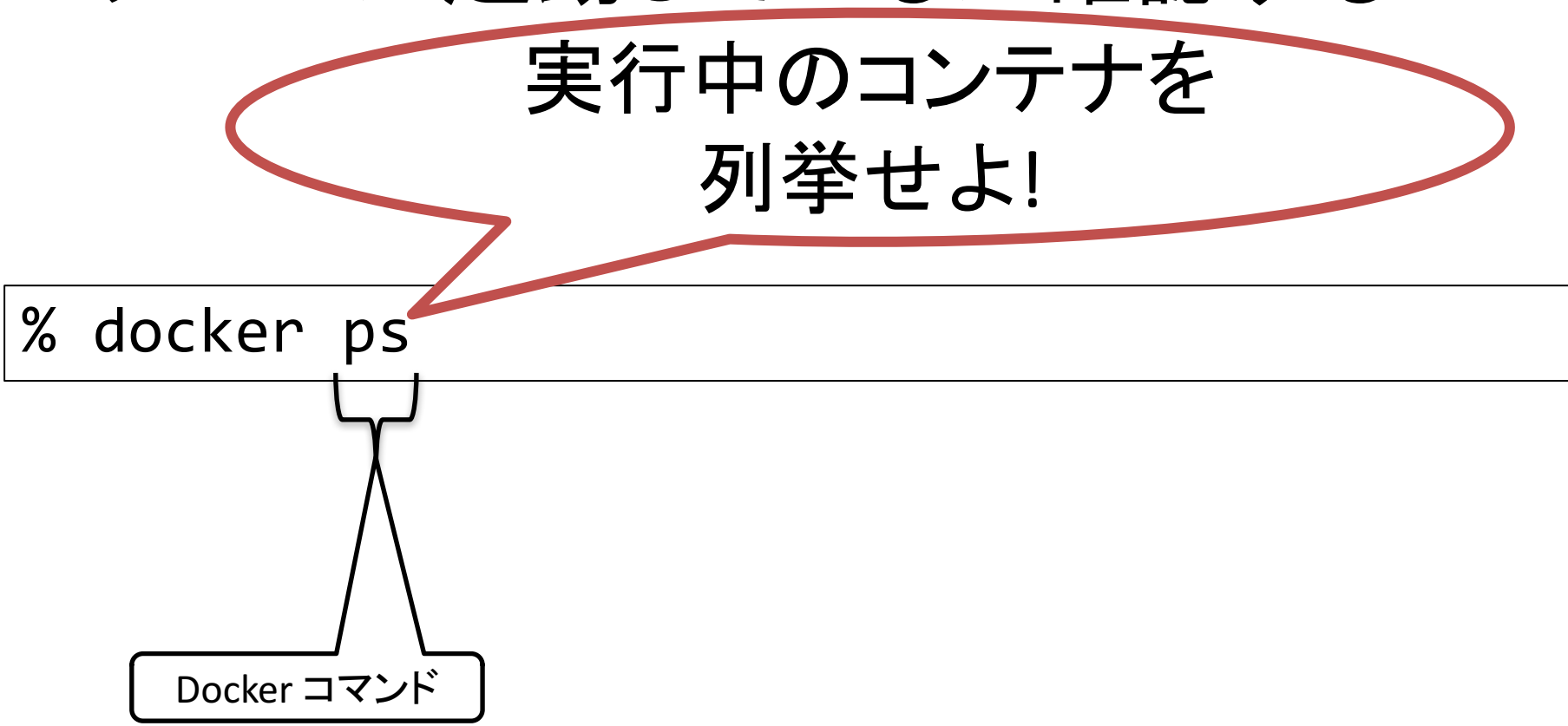
```
% docker ps
```

コンテナを実行してみよう (その2)
サーバが起動しているか確認する

```
% docker ps
```



コンテナを実行してみよう (その2)
サーバが起動しているか確認する



コンテナを実行してみよう (その2)
サーバが起動しているか確認する

```
% docker ps
```

コンテナを実行してみよう(その2)
サーバが起動しているか確認する

```
% docker ps
```

```
CONTAINER ID   IMAGE    COMMAND
19ecb956ab5d   nginx    “nginx -g ‘daemon ...”
```

コンテナを実行してみよう (その2)
サーバを停止してみる

```
% docker stop 19ecb
```

コンテナを実行してみよう (その2)
サーバを停止してみる

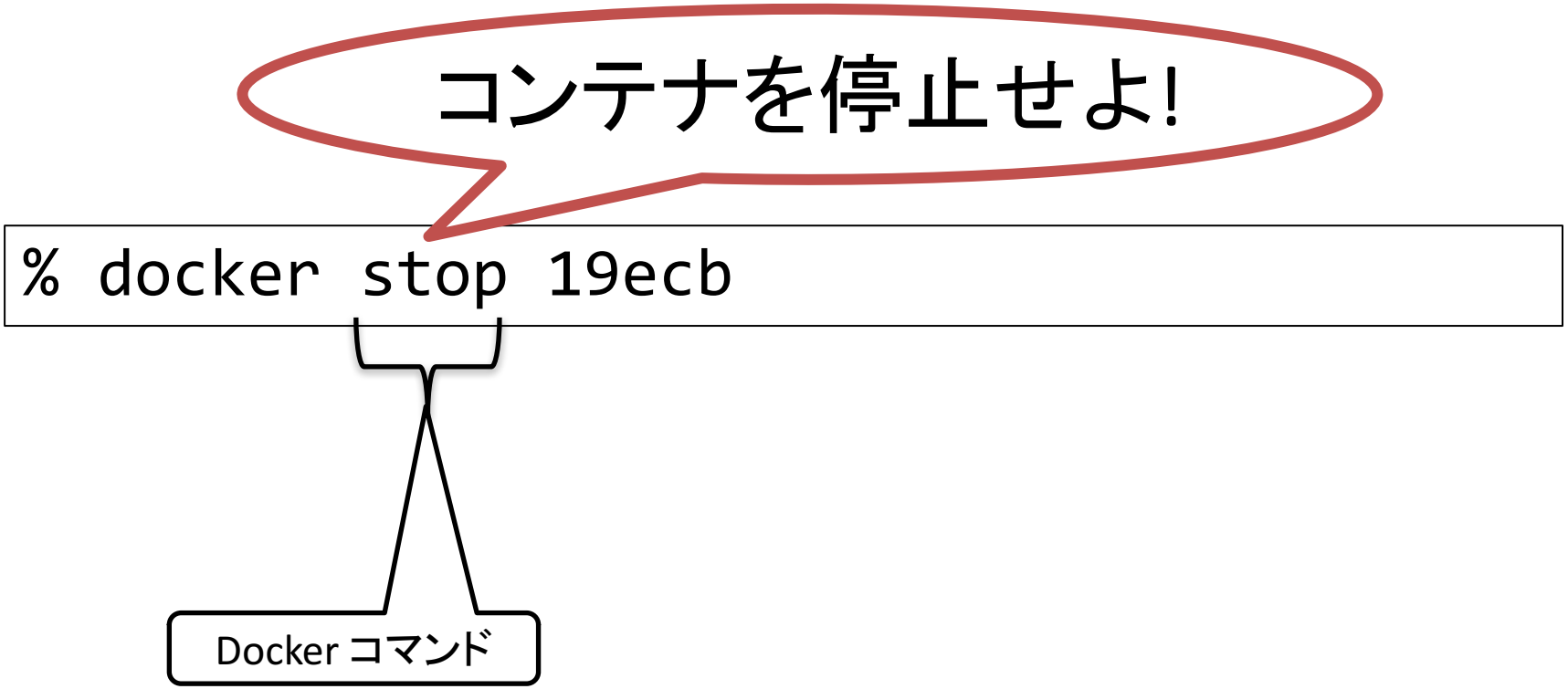
```
% docker stop 19ecb
```

```
19ecb
```

コンテナを実行してみよう (その2)
サーバを停止してみる

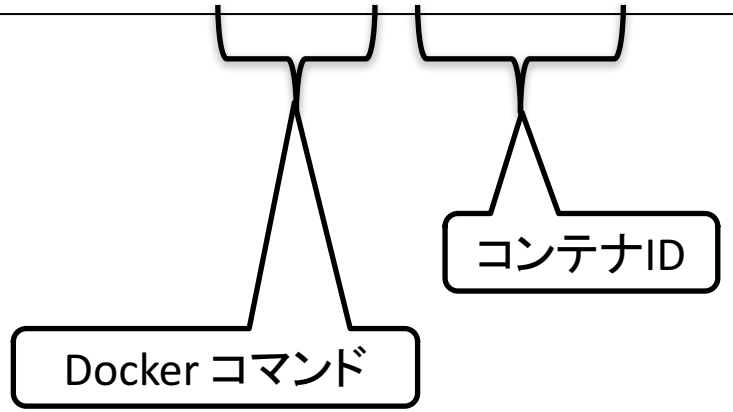
```
% docker stop 19ecb
```

コンテナを実行してみよう (その2)
サーバを停止してみる

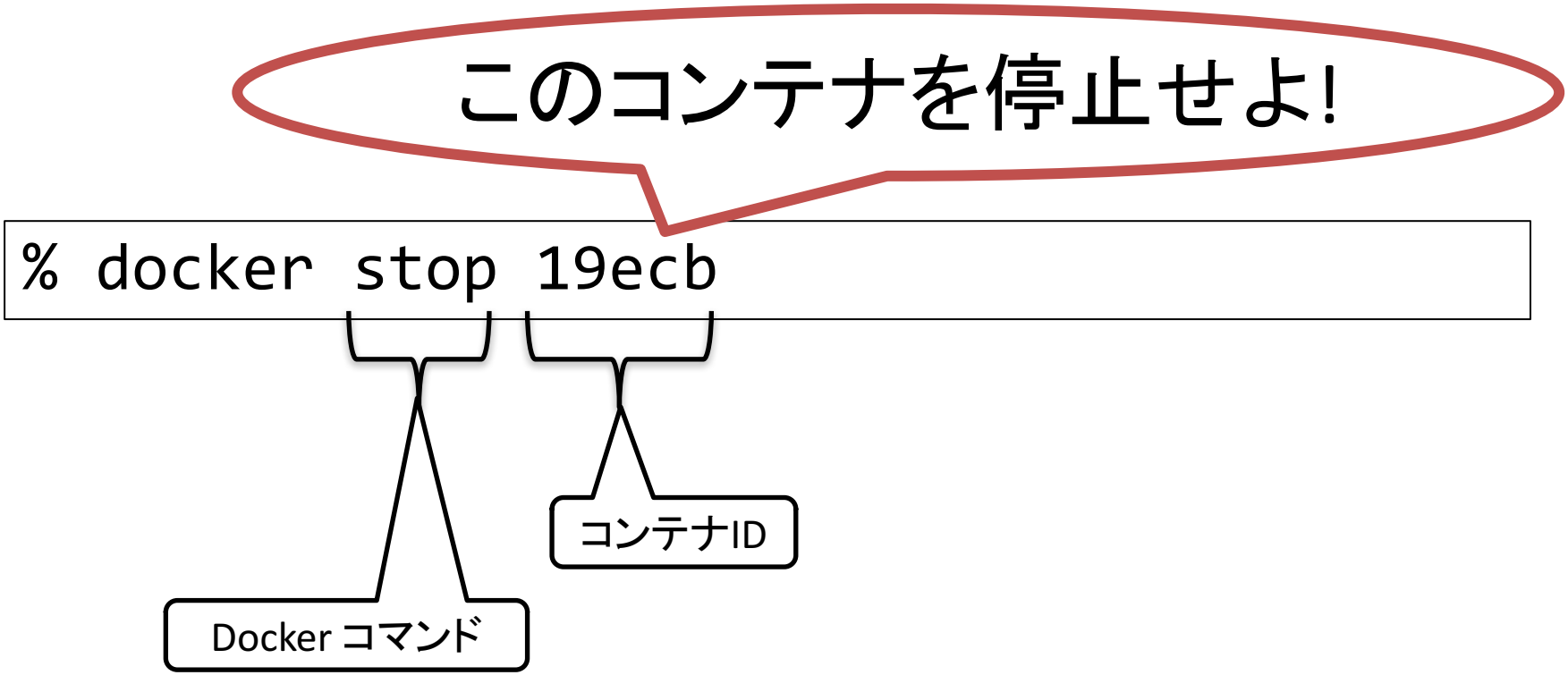


コンテナを実行してみよう (その2)
サーバを停止してみる

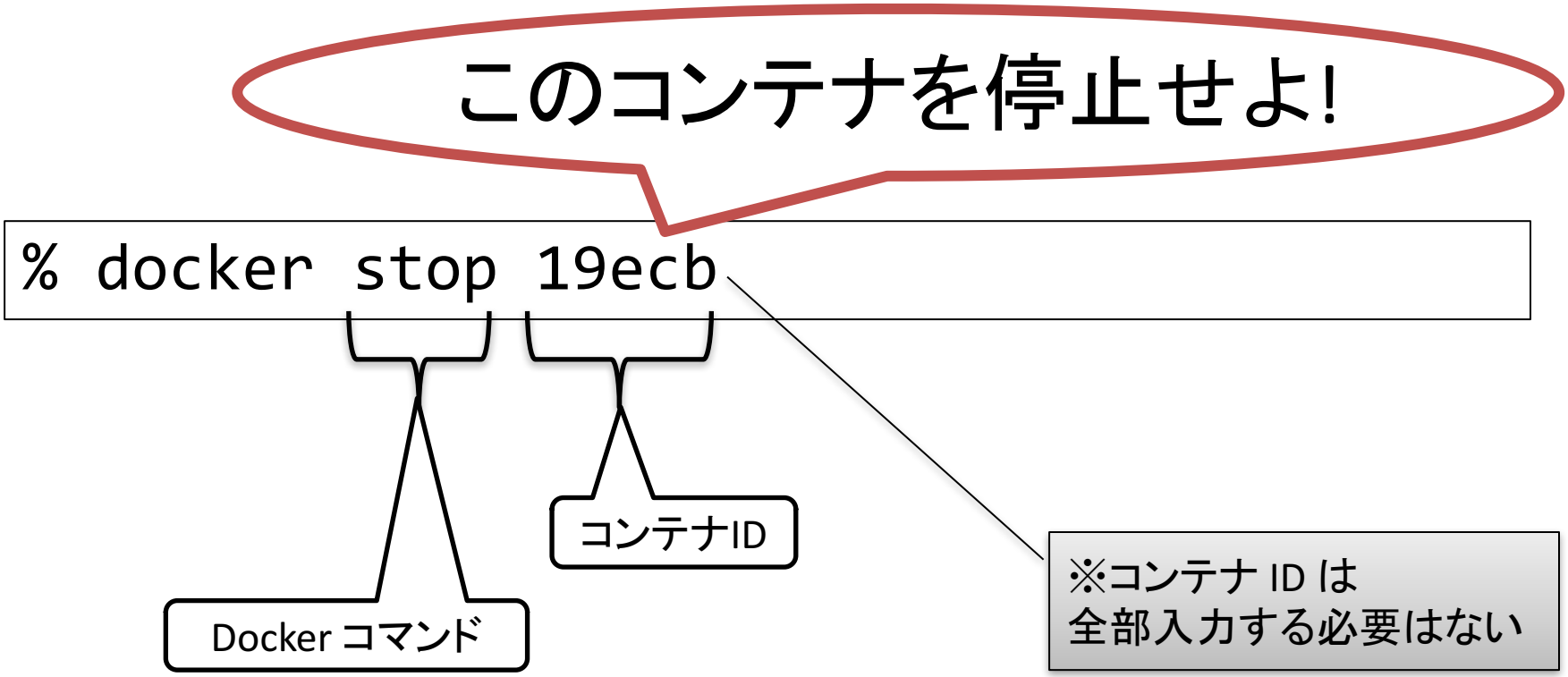
```
% docker stop 19ecb
```



コンテナを実行してみよう (その2)
サーバを停止してみる



コンテナを実行してみよう (その2)
サーバを停止してみる



コンテナを実行してみよう (その2)
サーバを停止してみる

```
% docker stop 19ecb
```

コンテナを実行してみよう (その2)
サーバを停止してみる

```
% docker stop 19ecb
```

```
19ecb
```

コンテナを実行してみよう (その2)
サーバが停止したか確認する

```
% docker ps
```

コンテナを実行してみよう(その2)
サーバが停止したか確認する

```
% docker ps
```

```
CONTAINER ID   IMAGE    COMMAND
```

コンテナを実行してみよう (その2)
停止したコンテナを確認する

```
% docker ps -a
```

コンテナを実行してみよう(その2)
停止したコンテナを確認する

```
% docker ps -a
```

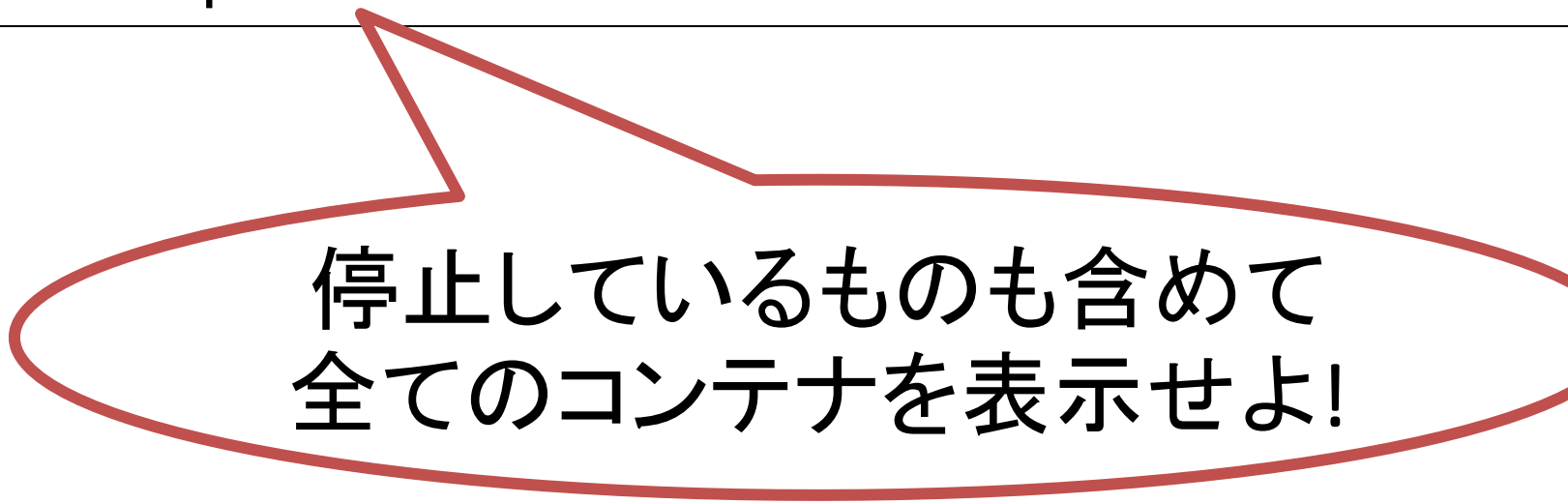
```
CONTAINER ID  IMAGE           COMMAND
19ecb956ab5d  nginx           "nginx -g 'daemo
a844e44f4790  ubuntu:17.04    "cat /etc/lsb-re
533c0f506a00  centos          "cat /etc/redhat
6b73015537c0  ubuntu          "cat /etc/lsb-re
```

コンテナを実行してみよう (その2)
停止したコンテナを確認する

```
% docker ps -a
```

コンテナを実行してみよう(その2)
停止したコンテナを確認する

```
% docker ps -a
```



コンテナを実行してみよう (その2)
停止したコンテナを確認する

```
% docker ps -a
```

コンテナを実行してみよう(その2)
停止したコンテナを確認する

```
% docker ps -a
```

```
CONTAINER ID  IMAGE           COMMAND
19ecb956ab5d  nginx           "nginx -g 'daemo
a844e44f4790  ubuntu:17.04    "cat /etc/lsb-re
533c0f506a00  centos          "cat /etc/redhat
6b73015537c0  ubuntu          "cat /etc/lsb-re
```

コンテナを実行してみよう(その2)
不要なコンテナを削除しよう

```
% docker rm 19ec a844 533c 6b73
```

コンテナを実行してみよう(その2)
不要なコンテナを削除しよう

```
% docker rm 19ec a844 533c 6b73
```

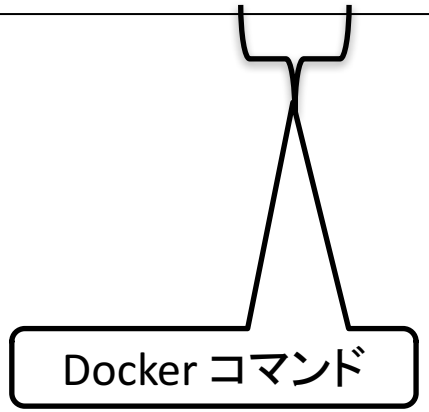
```
19ec
a844
533c
6b73
```

コンテナを実行してみよう(その2)
不要なコンテナを削除しよう

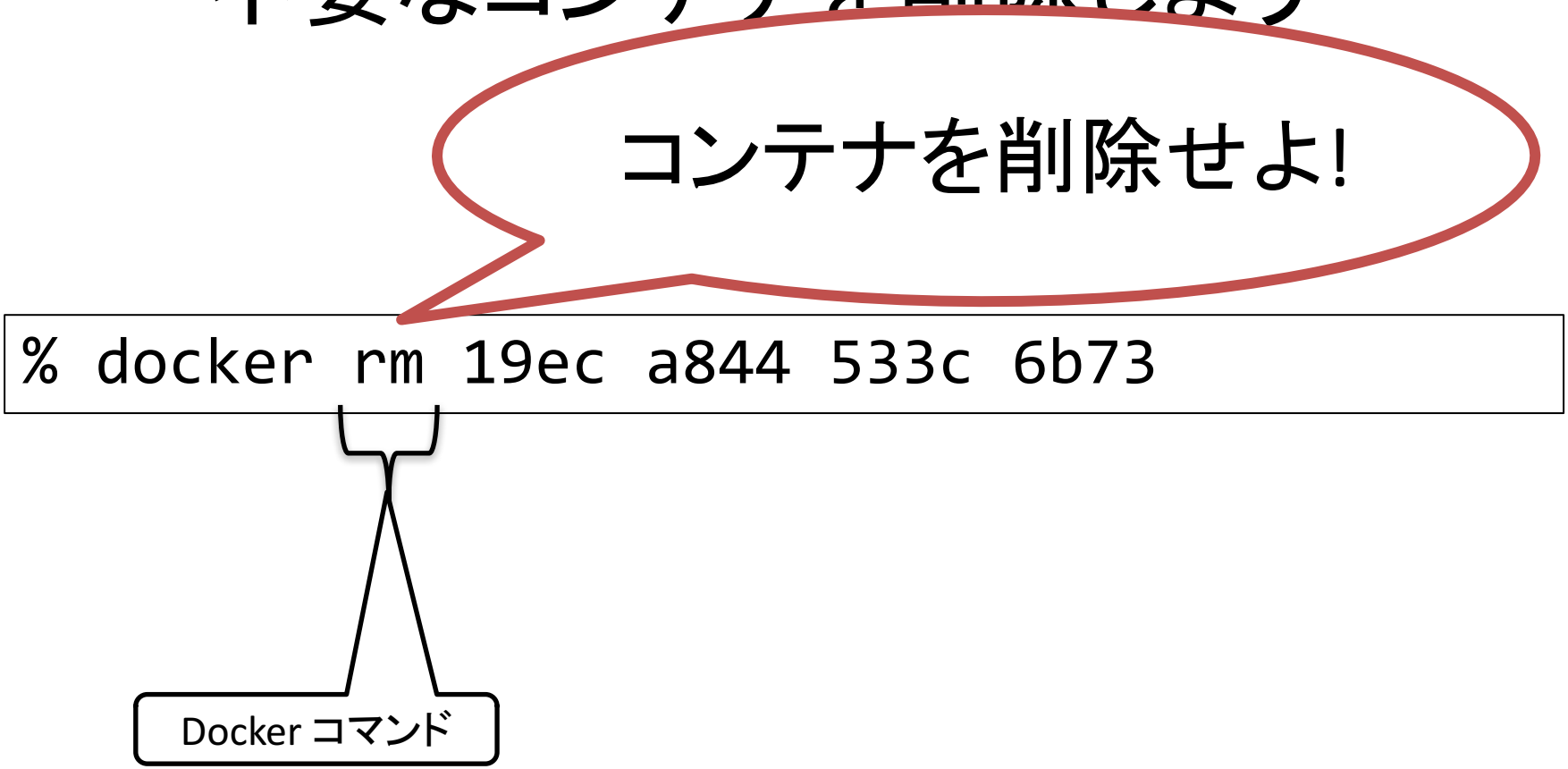
```
% docker rm 19ec a844 533c 6b73
```

コンテナを実行してみよう (その2)
不要なコンテナを削除しよう

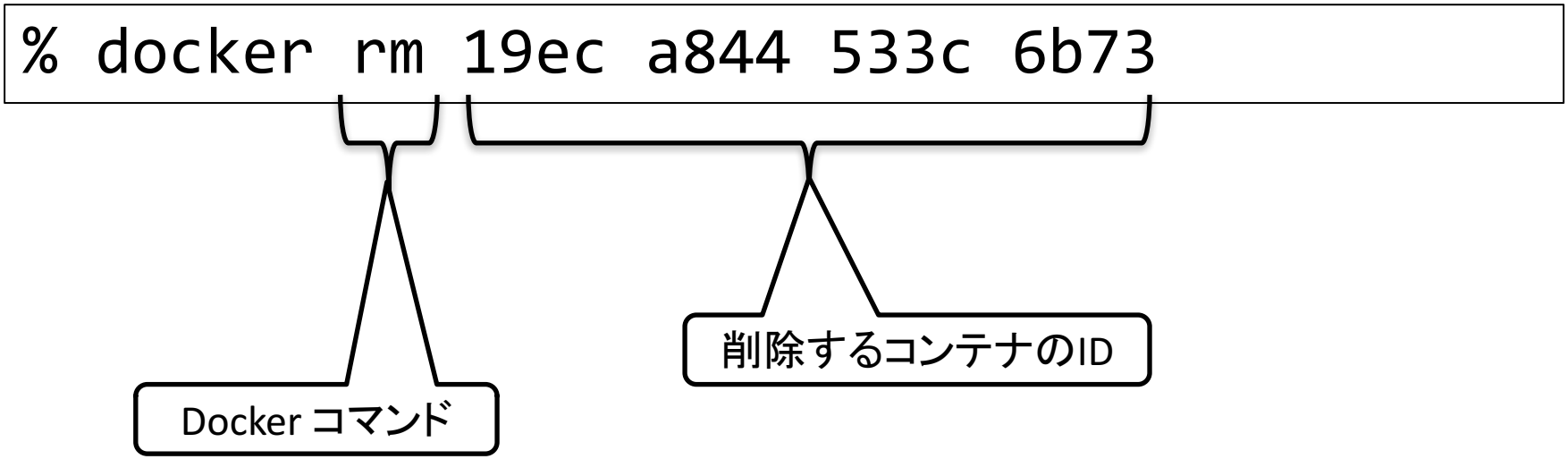
```
% docker rm 19ec a844 533c 6b73
```



コンテナを実行してみよう (その2)
不要なコンテナを削除しよう



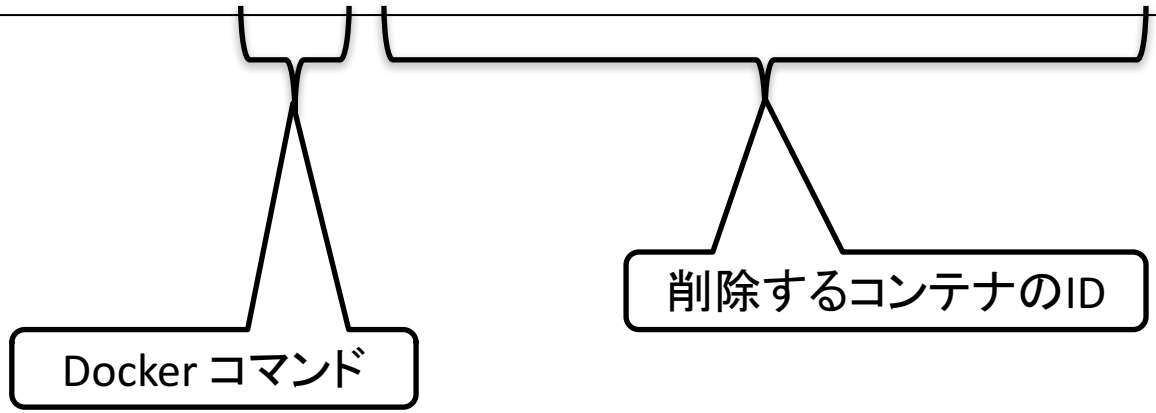
コンテナを実行してみよう(その2)
不要なコンテナを削除しよう



コンテナを実行してみよう (その2)
不要なコンテナを削除しよう

これらのコンテナを
削除せよ!

```
% docker rm 19ec a844 533c 6b73
```



コンテナを実行してみよう(その2)
不要なコンテナを削除しよう

```
% docker rm 19ec a844 533c 6b73
```

コンテナを実行してみよう(その2)
不要なコンテナを削除しよう

```
% docker rm 19ec a844 533c 6b73
```

```
19ec
a844
533c
6b73
```

コンテナを実行してみよう(その2)
コンテナを削除できたか確認する

```
% docker ps -a
```

コンテナを実行してみよう(その2)
コンテナを削除できたか確認する

```
% docker ps -a
```

```
CONTAINER ID   IMAGE    COMMAND
```

コンテナ内でシェルを起動してみる

```
% docker run –i –t ubuntu /bin/bash
```

コンテナ内でシェルを起動してみる

```
% docker run -i -t ubuntu /bin/bash
```

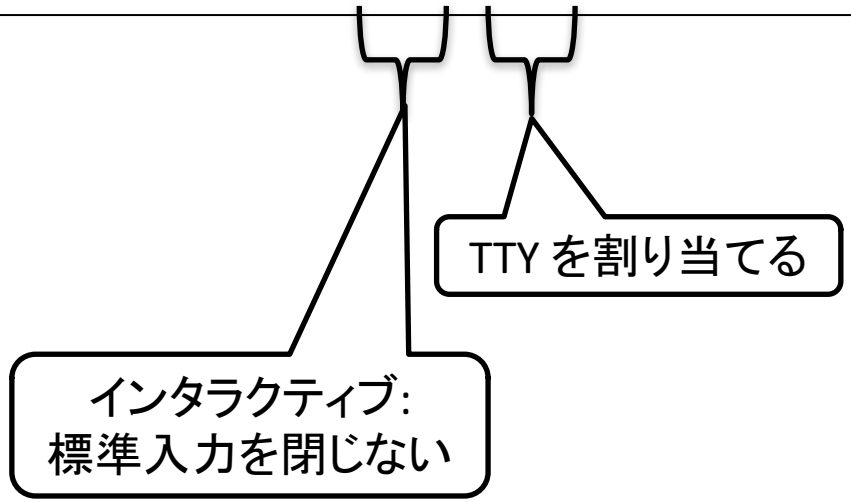
```
root@d2db426e0aa9:/#
```

コンテナ内でシェルを起動してみる

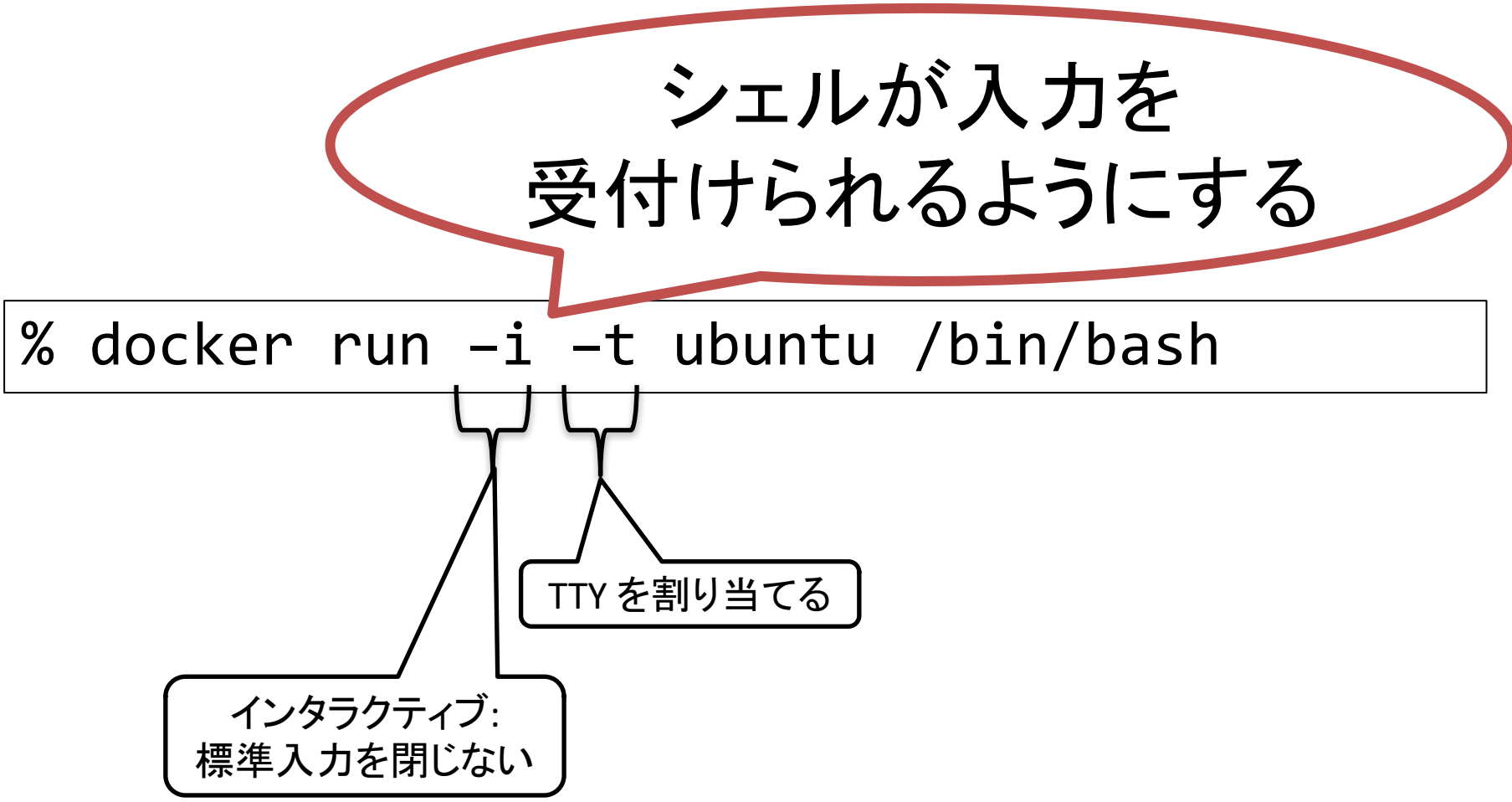
```
% docker run –i –t ubuntu /bin/bash
```

コンテナ内でシェルを起動してみる

```
% docker run -i -t ubuntu /bin/bash
```



コンテナ内でシェルを起動してみる



コンテナ内でシェルを起動してみる

```
% docker run –i –t ubuntu /bin/bash
```

コンテナ内でシェルを起動してみる

```
% docker run -i -t ubuntu /bin/bash
```

```
root@d2db426e0aa9:/#
```

コンテナ内でシェルを終了する

```
% docker run –i –t ubuntu /bin/bash
```

```
root@d2db426e0aa9:/# exit
```

停止したコンテナを再起動する

```
% docker restart d2db4
```

停止したコンテナを再起動する

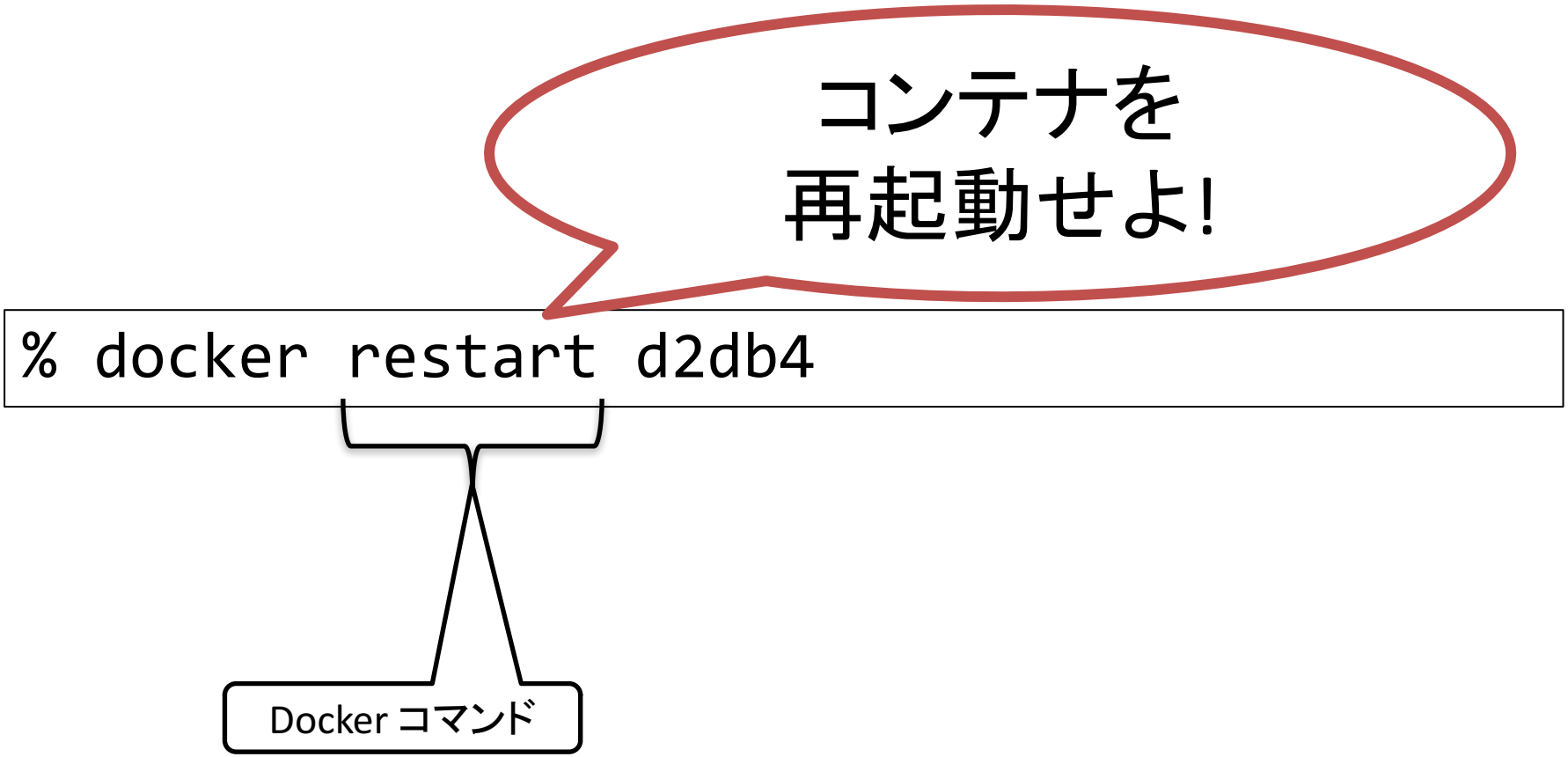
```
% docker restart d2db4
```

```
d2db4
```

停止したコンテナを再起動する

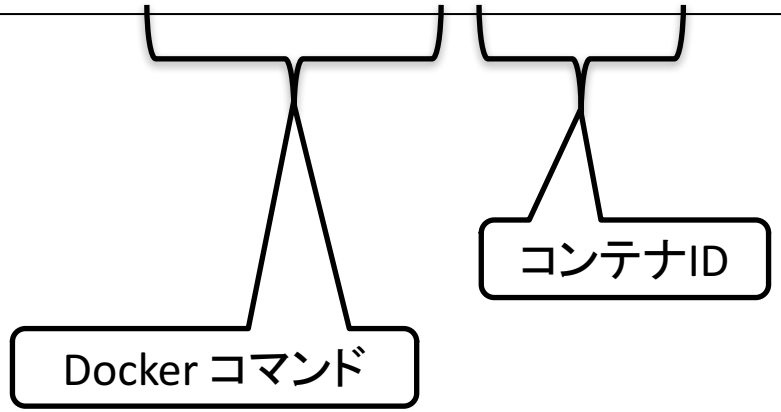
```
% docker restart d2db4
```

停止したコンテナを再起動する

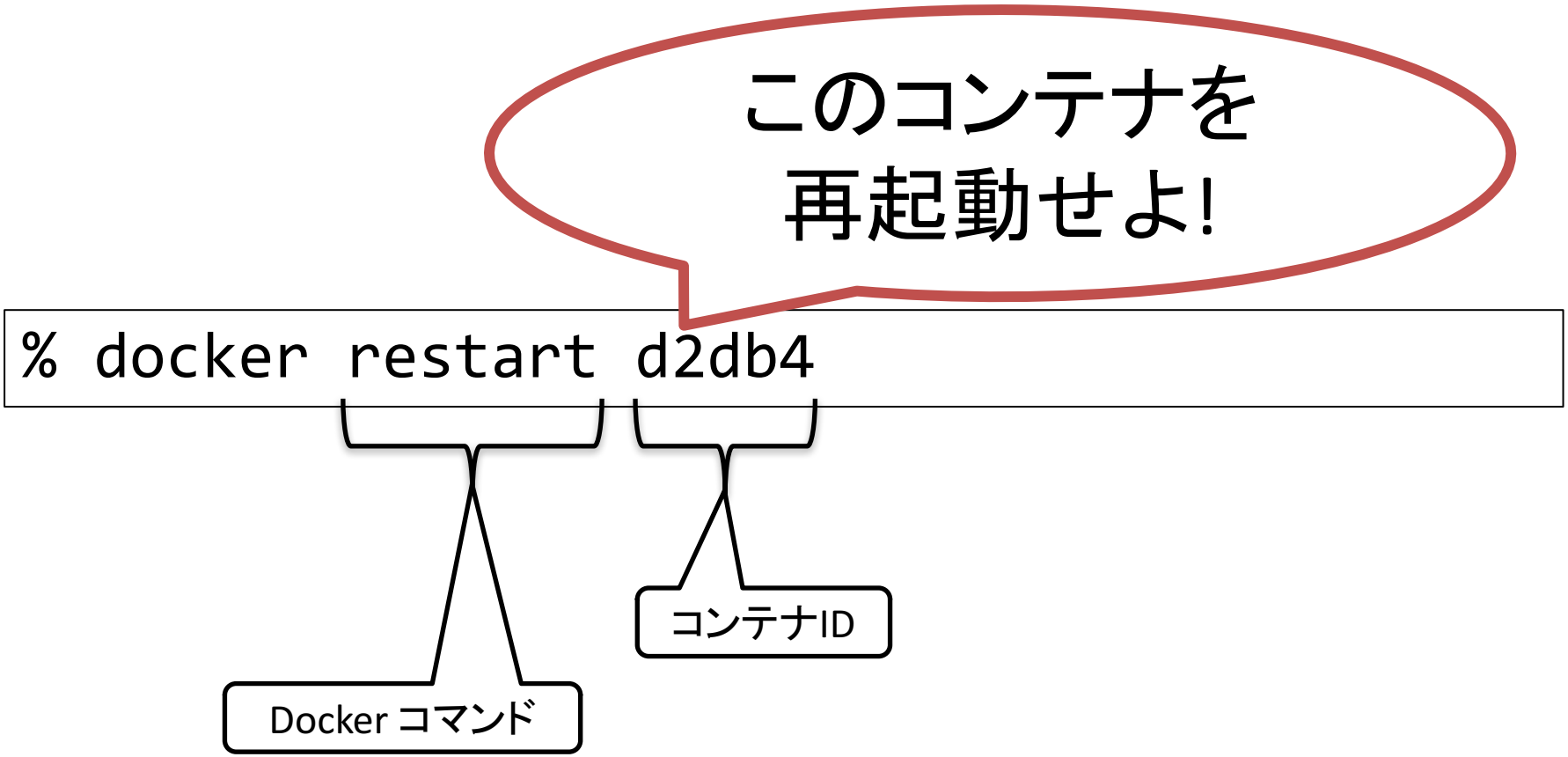


停止したコンテナを再起動する

```
% docker restart d2db4
```



停止したコンテナを再起動する



停止したコンテナを再起動する

```
% docker restart d2db4
```

停止したコンテナを再起動する

```
% docker restart d2db4
```

```
d2db4
```

再起動したコンテナに接続する

```
% docker attach d2db4
```

再起動したコンテナに接続する

```
% docker attach d2db4
```

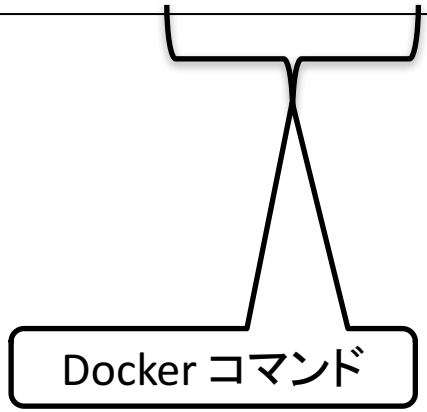
```
root@d2db426e0aa9:/#
```

再起動したコンテナに接続する

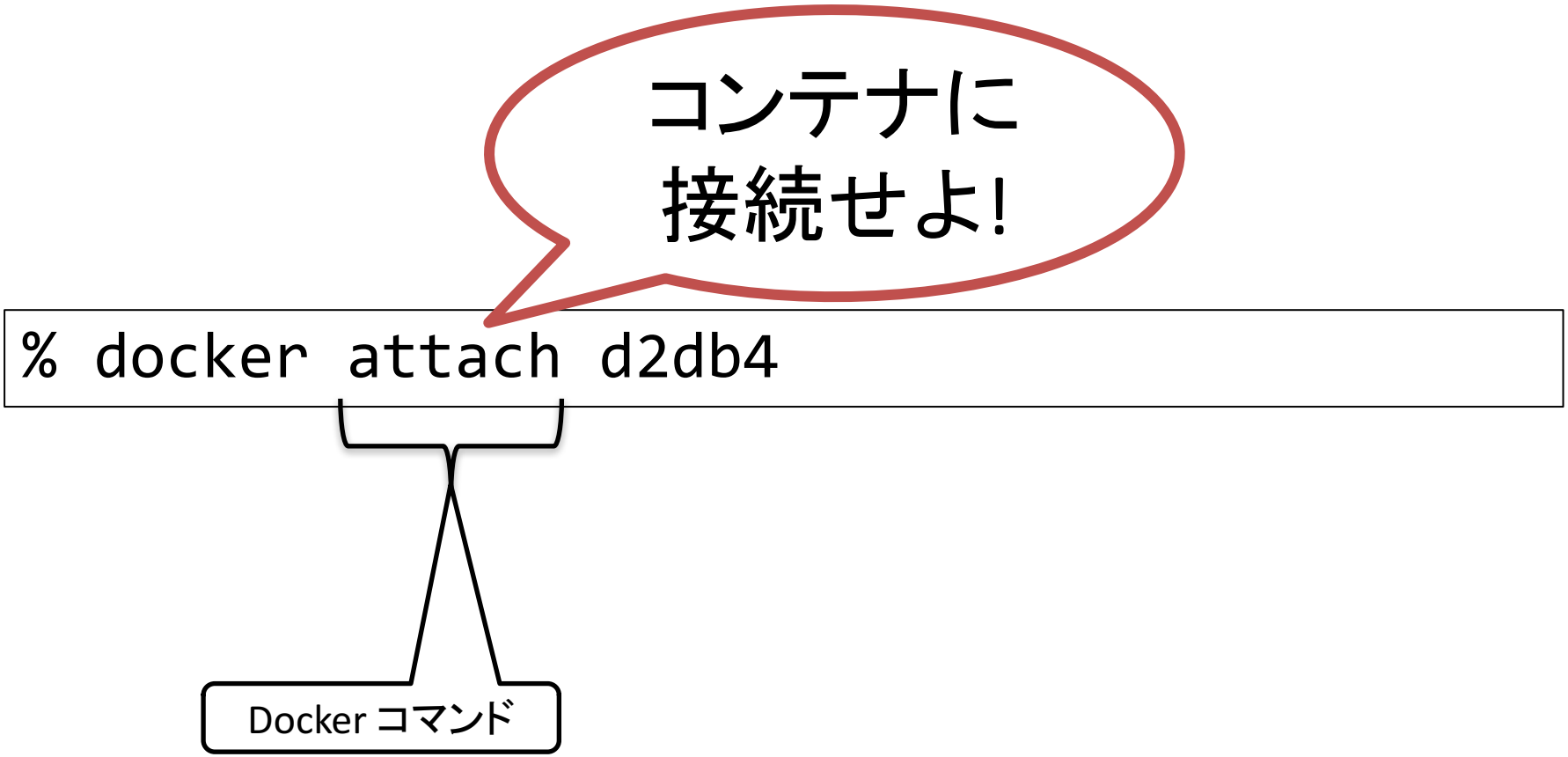
```
% docker attach d2db4
```

再起動したコンテナに接続する

```
% docker attach d2db4
```

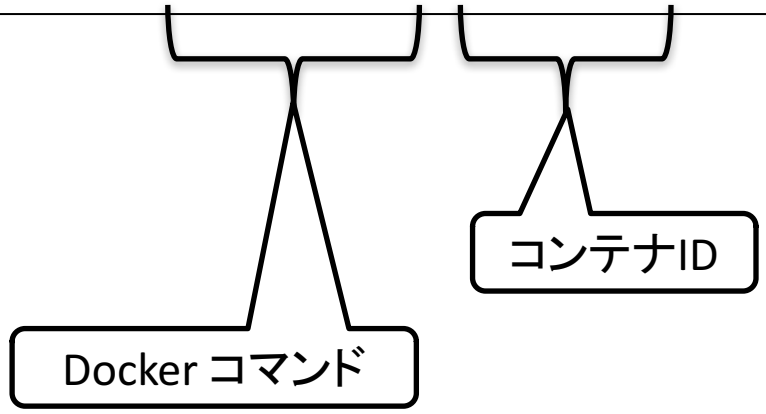


再起動したコンテナに接続する

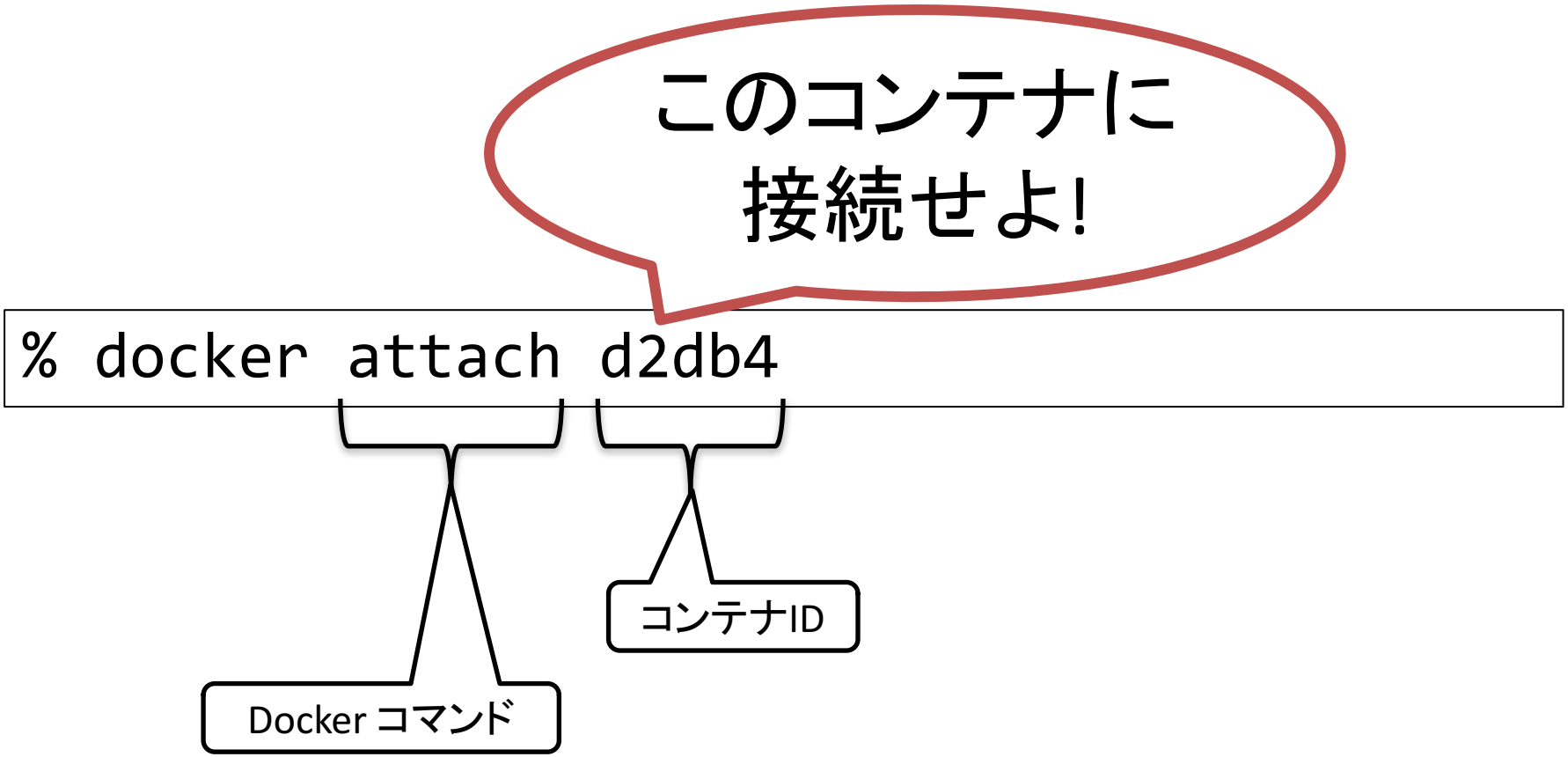


再起動したコンテナに接続する

```
% docker attach d2db4
```



再起動したコンテナに接続する



再起動したコンテナに接続する

```
% docker attach d2db4
```

再起動したコンテナに接続する

```
% docker attach d2db4
```

```
root@d2db426e0aa9:/#
```

実行中のコンテナ内でシェルを起動する

```
% docker run -d -p (好きな番号):80 nginx
```

実行中のコンテナ内でシェルを起動する

```
% docker run -d -p (好きな番号):80 nginx
```

```
30d198bf0ad3c03535f2d292153a9f7ae003b71ca0ec
```

実行中のコンテナ内でシェルを起動する

```
% docker exec –i –t 30d19 /bin/bash
```

実行中のコンテナ内でシェルを起動する

```
% docker exec -i -t 30d19 /bin/bash
```

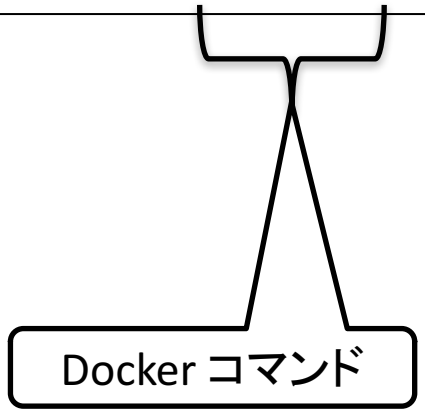
```
root@30d198bf0ad3:/#
```

実行中のコンテナ内でシェルを起動する

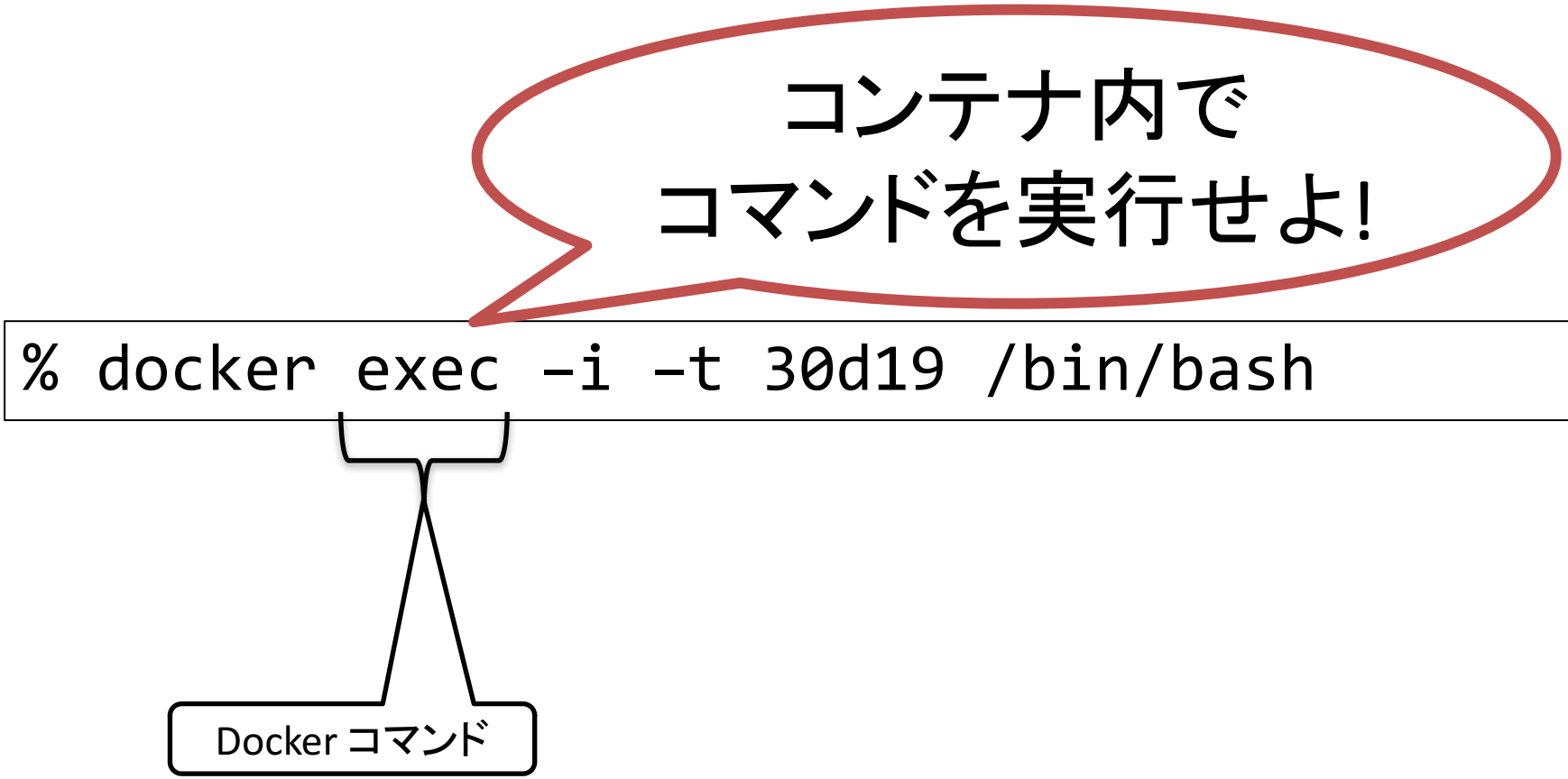
```
% docker exec –i –t 30d19 /bin/bash
```

実行中のコンテナ内でシェルを起動する

```
% docker exec -i -t 30d19 /bin/bash
```

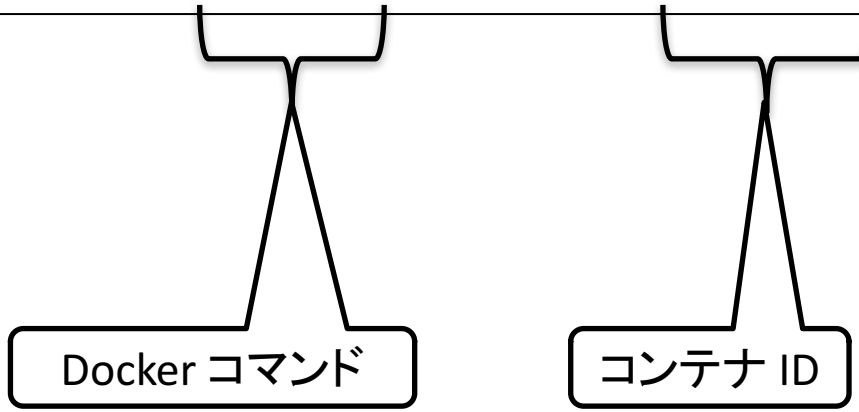


実行中のコンテナ内でシェルを起動する

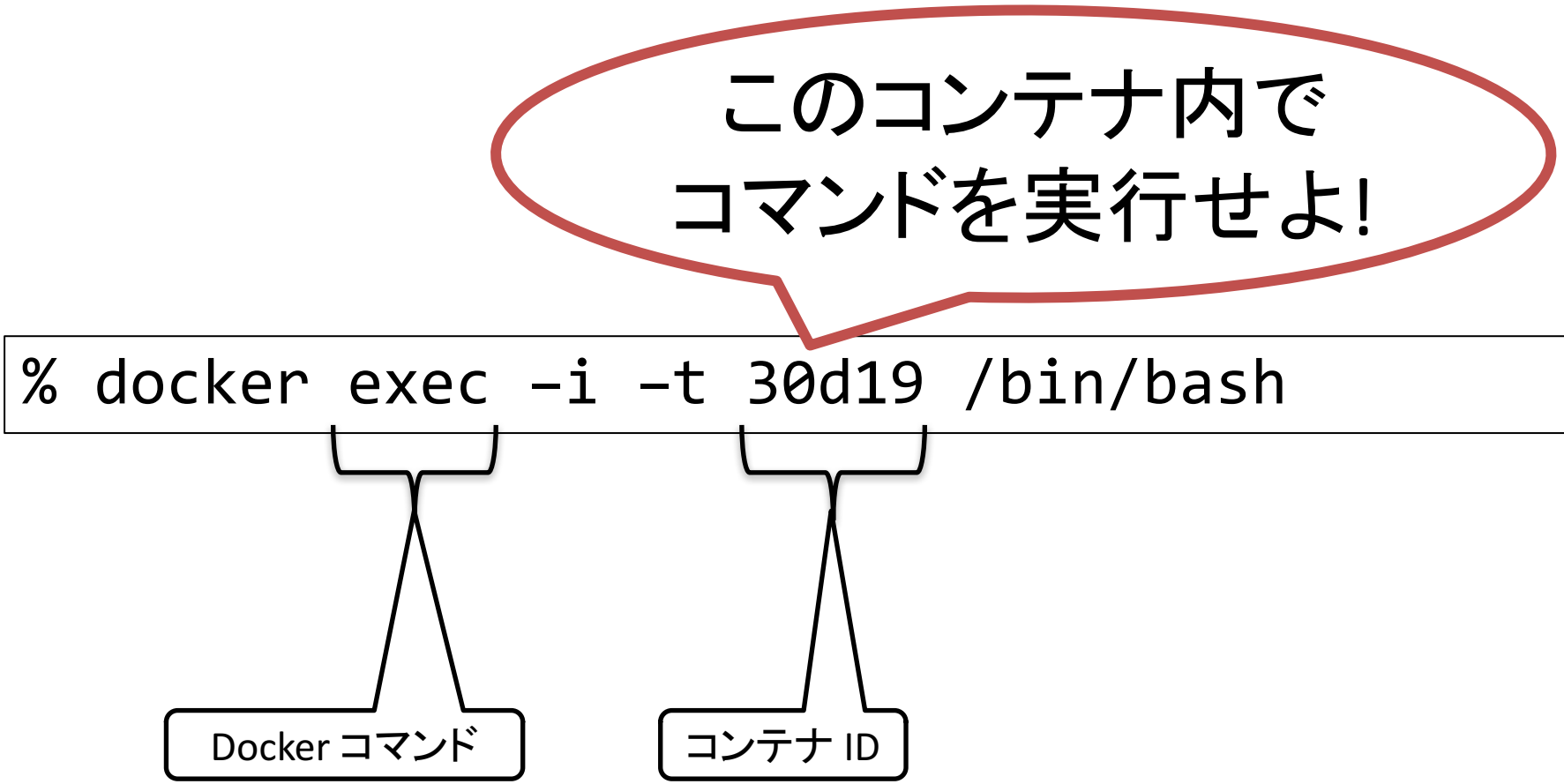


実行中のコンテナ内でシェルを起動する

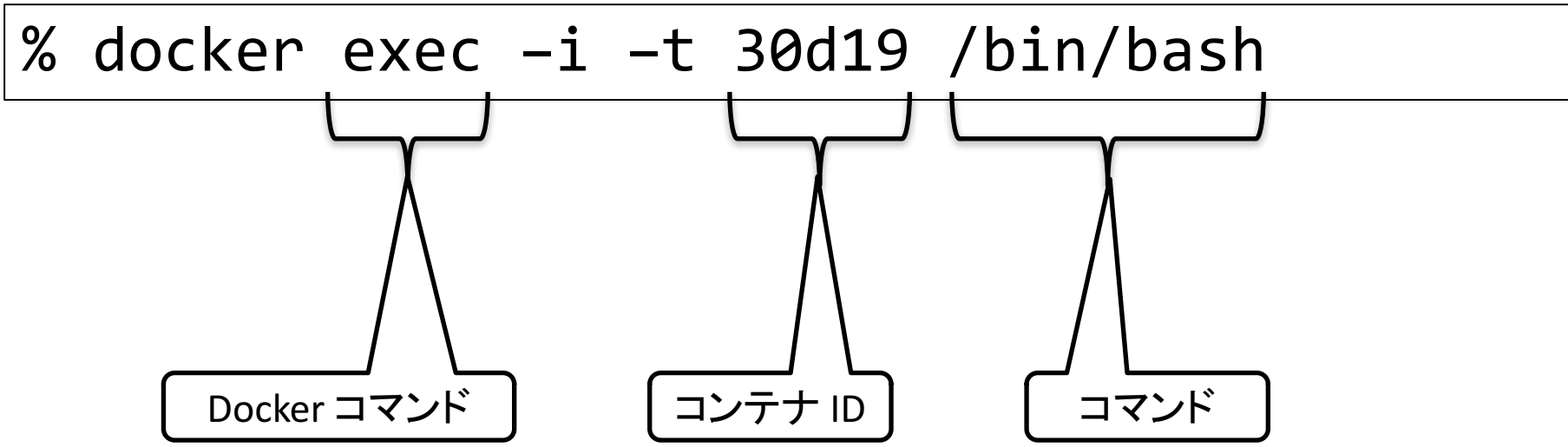
```
% docker exec -i -t 30d19 /bin/bash
```



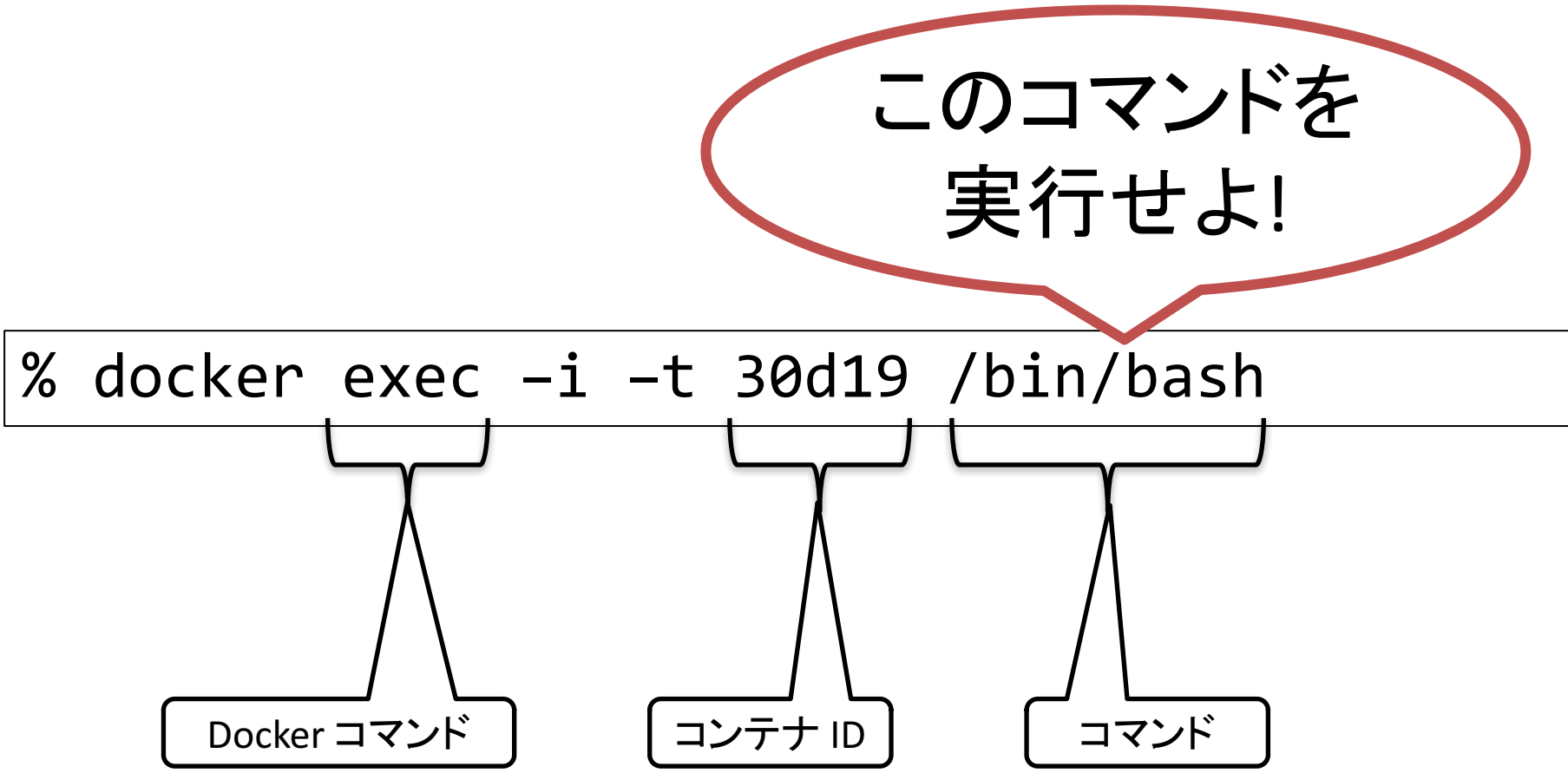
実行中のコンテナ内でシェルを起動する



実行中のコンテナ内でシェルを起動する



実行中のコンテナ内でシェルを起動する



実行中のコンテナ内でシェルを起動する

```
% docker exec –i –t 30d19 /bin/bash
```

実行中のコンテナ内でシェルを起動する

```
% docker exec -i -t 30d19 /bin/bash
```

```
root@30d198bf0ad3:/#
```

コンテナからイメージを作成する

```
% docker stop 30d19
```

コンテナからイメージを作成する

```
% docker stop 30d19
```

```
30d19
```

コンテナからイメージを作成する

```
% docker commit 30d19 (好きな名前)
```

コンテナからイメージを作成する

```
% docker commit 30d19 (好きな名前)
```

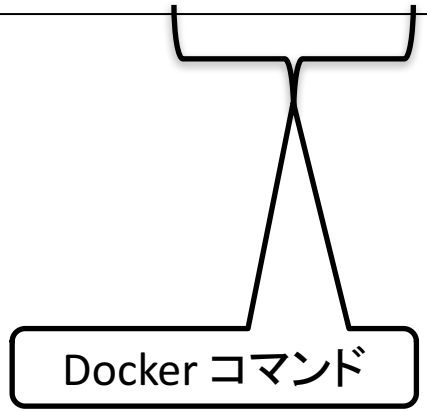
```
sha256:96475fbbe6901fcf946502eebb317869d791a
```

コンテナからイメージを作成する

```
% docker commit 30d19 (好きな名前)
```

コンテナからイメージを作成する

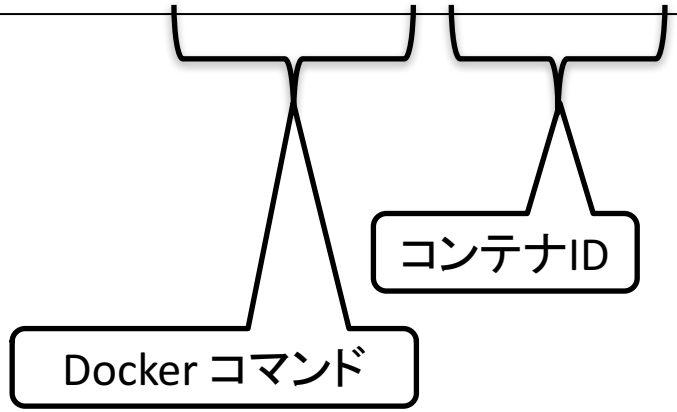
```
% docker commit 30d19 (好きな名前)
```



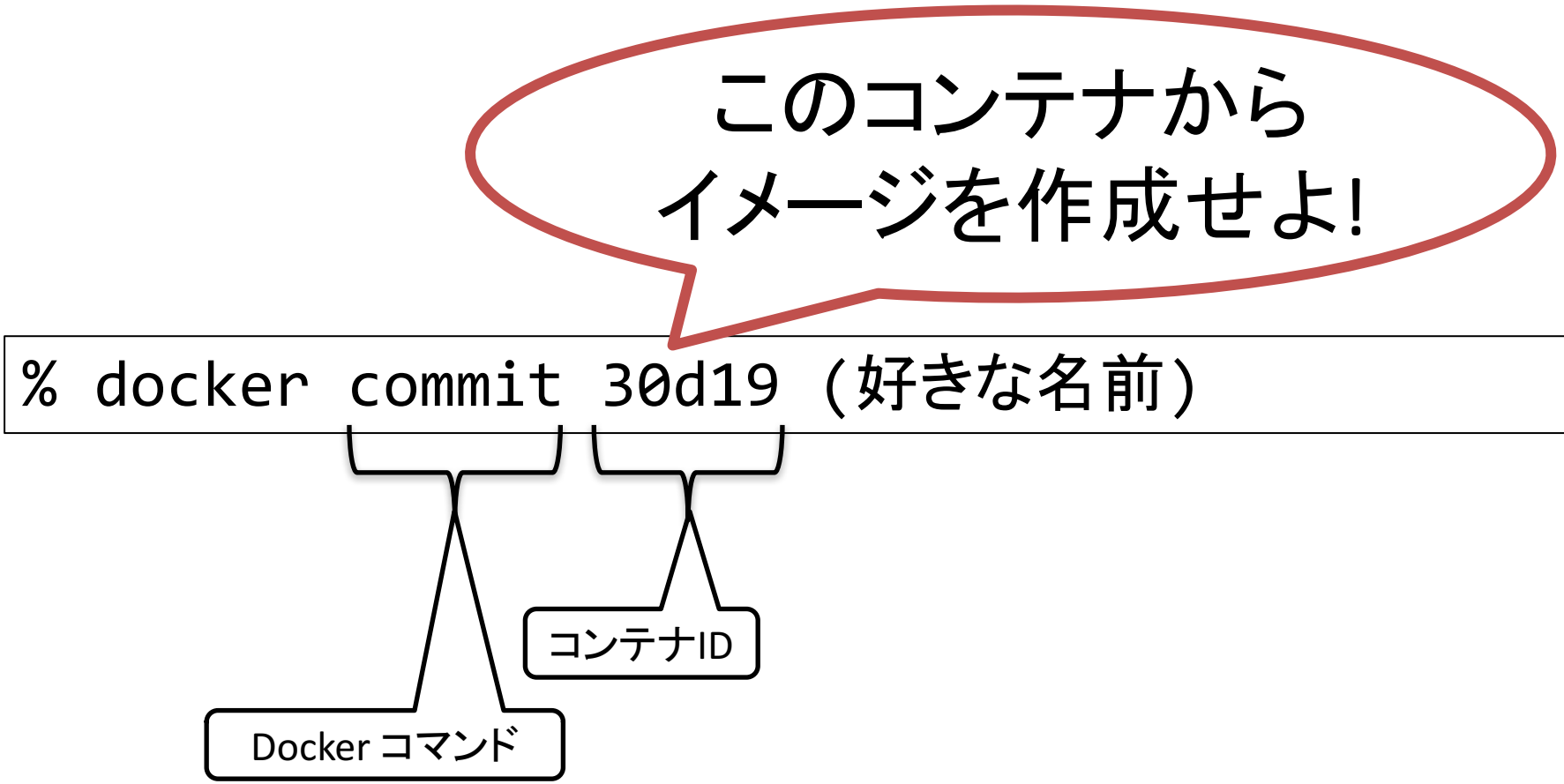
コンテナからイメージを作成する

コンテナからイメージを作成する

```
% docker commit 30d19 (好きな名前)
```

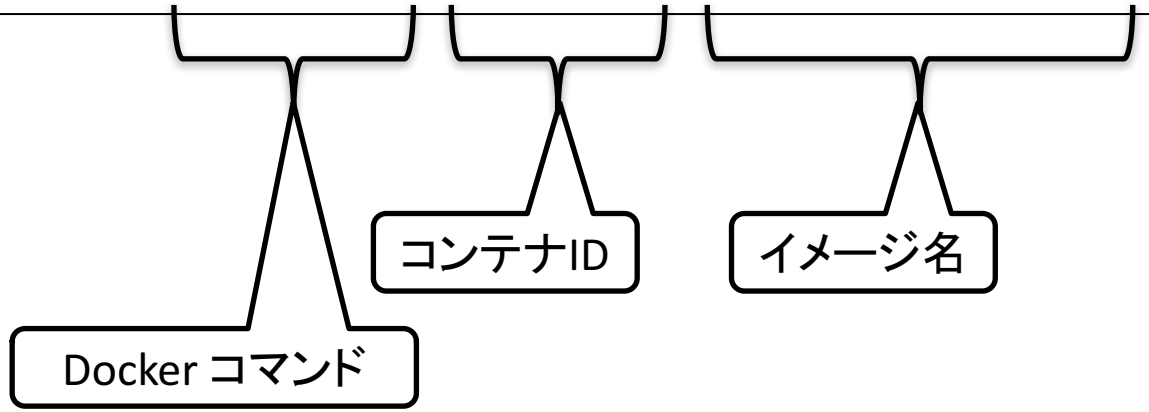


コンテナからイメージを作成する



コンテナからイメージを作成する

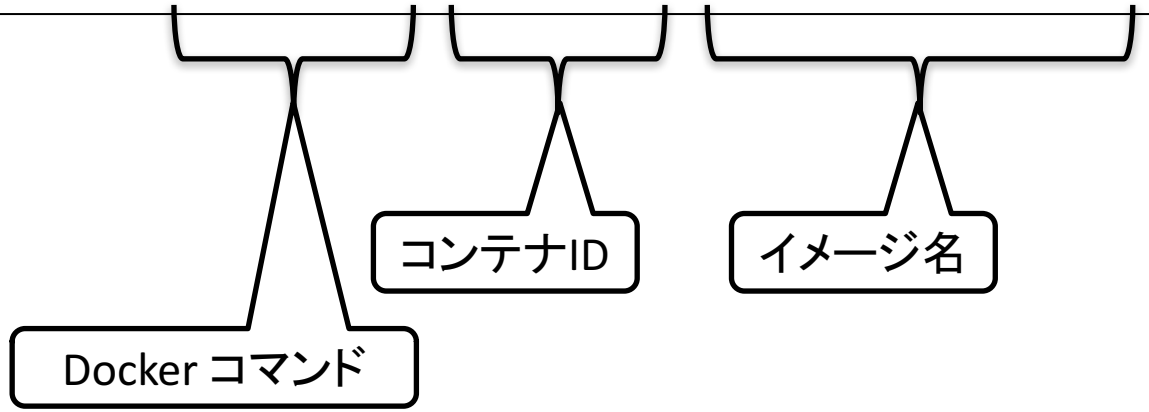
```
% docker commit 30d19 (好きな名前)
```



コンテナからイメージを作成する

このイメージ名で
イメージを作成せよ!

```
% docker commit 30d19 (好きな名前)
```



コンテナからイメージを作成する

```
% docker commit 30d19 (好きな名前)
```

コンテナからイメージを作成する

```
% docker commit 30d19 (好きな名前)
```

```
sha256:96475fbbe6901fcf946502eebb317869d791a
```

作成したイメージを確認する

```
% docker images
```

作成したイメージを確認する

```
% docker images
```

```
REPOSITORY  TAG      IMAGE ID       CREATED
(好きな名前) latest  96475fbbe690  32 seconds
centos      latest   98d35105a391  4 days ago
nginx       latest   6b914bbcb89e  2 weeks ag
ubuntu      latest   0ef2e08ed3fa  2 weeks ag
```

作成したイメージでコンテナを実行する

```
% docker run -d -p (好きな番号):80 (好きな名前)
```

作成したイメージでコンテナを実行する

```
% docker run -d -p (好きな番号):80 (好きな名前)
```

```
172e159d9e814d15a7c579a06eabfc050106811b6bfcad1
```

ホストとコンテナでファイルを共有する

```
% mkdir nginx-files
% docker run -d -p (好きな番号):80 \
   –v $PWD/nginx-files:/usr/share/nginx/html \
   (好きな名前)
```

ホストとコンテナでファイルを共有する

```
% mkdir nginx-files
% docker run -d -p (好きな番号):80 \
   –v $PWD/nginx-files:/usr/share/nginx/html \
   (好きな名前)
```

```
87ee8e79f2139f658cbdbdd77667b64b0b5d0dae2cff
```

ホストとコンテナでファイルを共有する

```
% mkdir nginx-files
% docker run -d -p (好きな番号):80 \
   –v $PWD/nginx-files:/usr/share/nginx/html \
   (好きな名前)
```

ホストとコンテナでファイルを共有する

```
% mkdir nginx-files
% docker run -d -p (好きな番号):80 \
   -v $PWD/nginx-files:/usr/share/nginx/html \
   (好きな名前)
```

ホスト上のディレクトリ

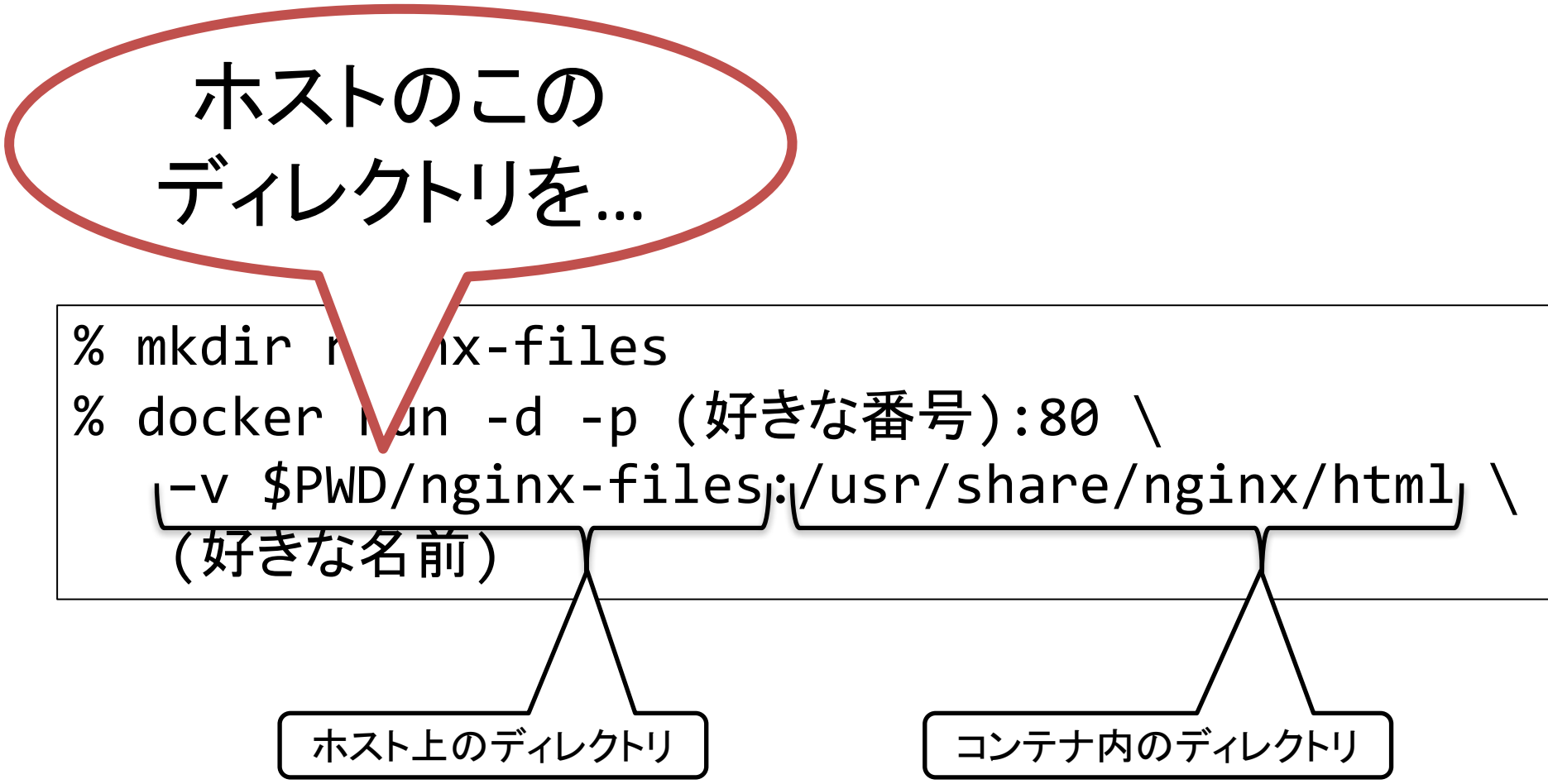
ホストとコンテナでファイルを共有する

```
% mkdir nginx-files
% docker run -d -p (好きな番号):80 \
   -v $PWD/nginx-files:/usr/share/nginx/html \
   (好きな名前)
```

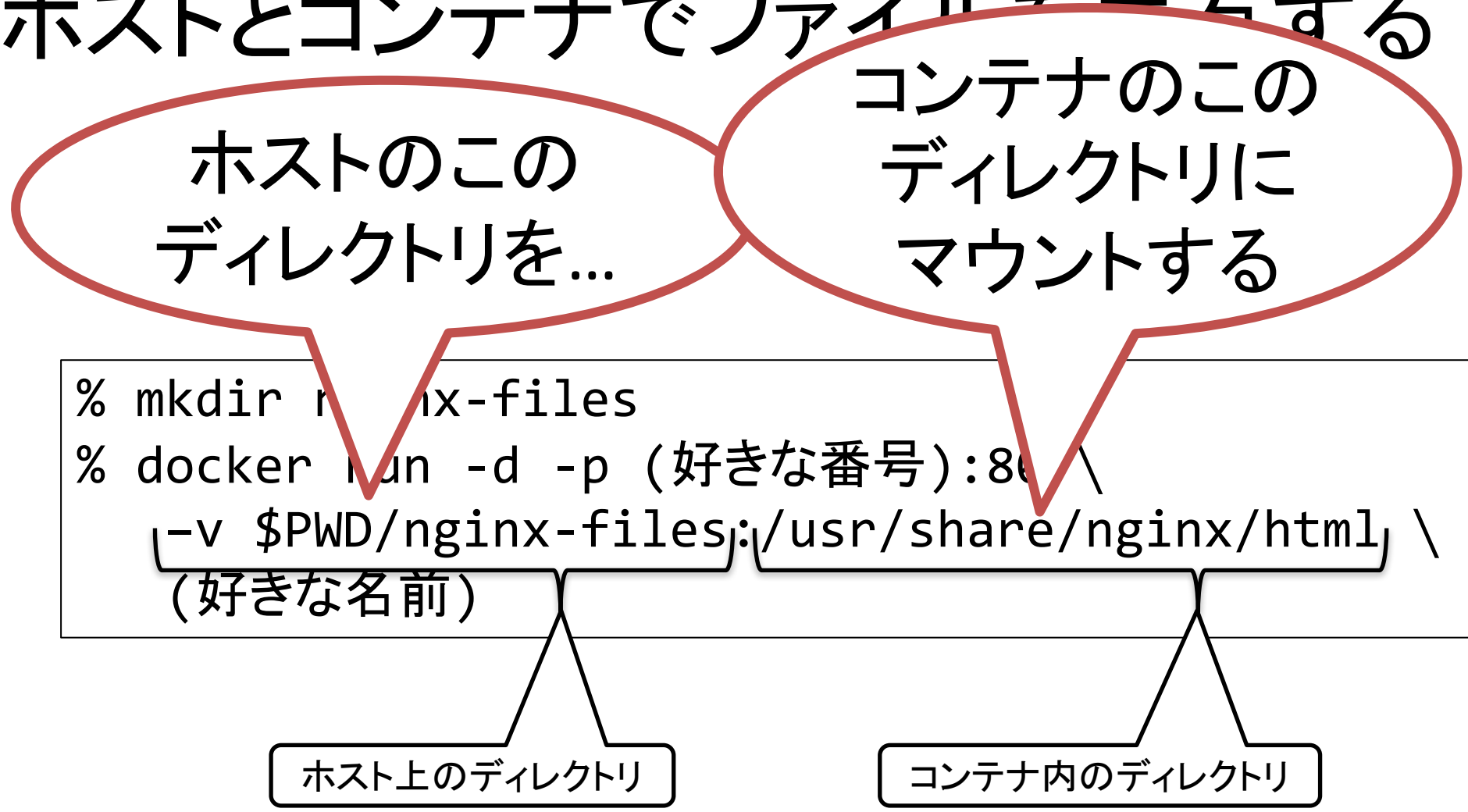
ホスト上のディレクトリ

コンテナ内のディレクトリ

ホストとコンテナでファイルを共有する



ホストとコンテナでファイルを共有する



ホストとコンテナでファイルを共有する

```
% mkdir nginx-files
% docker run -d -p (好きな番号):80 \
   –v $PWD/nginx-files:/usr/share/nginx/html \
   (好きな名前)
```

このように共有されたディレクトリをボリュームと呼ぶ

ホストとコンテナでファイルを共有する

```
% mkdir nginx-files
% docker run -d -p (好きな番号):80 \
   –v $PWD/nginx-files:/usr/share/nginx/html \
   (好きな名前)
```

ホストとコンテナでファイルを共有する

```
% mkdir nginx-files
% docker run -d -p (好きな番号):80 \
   –v $PWD/nginx-files:/usr/share/nginx/html \
   (好きな名前)
```

```
87ee8e79f2139f658cbdbdd77667b64b0b5d0dae2cff
```

ホストとコンテナでファイルを共有する: ホストからファイルを編集する

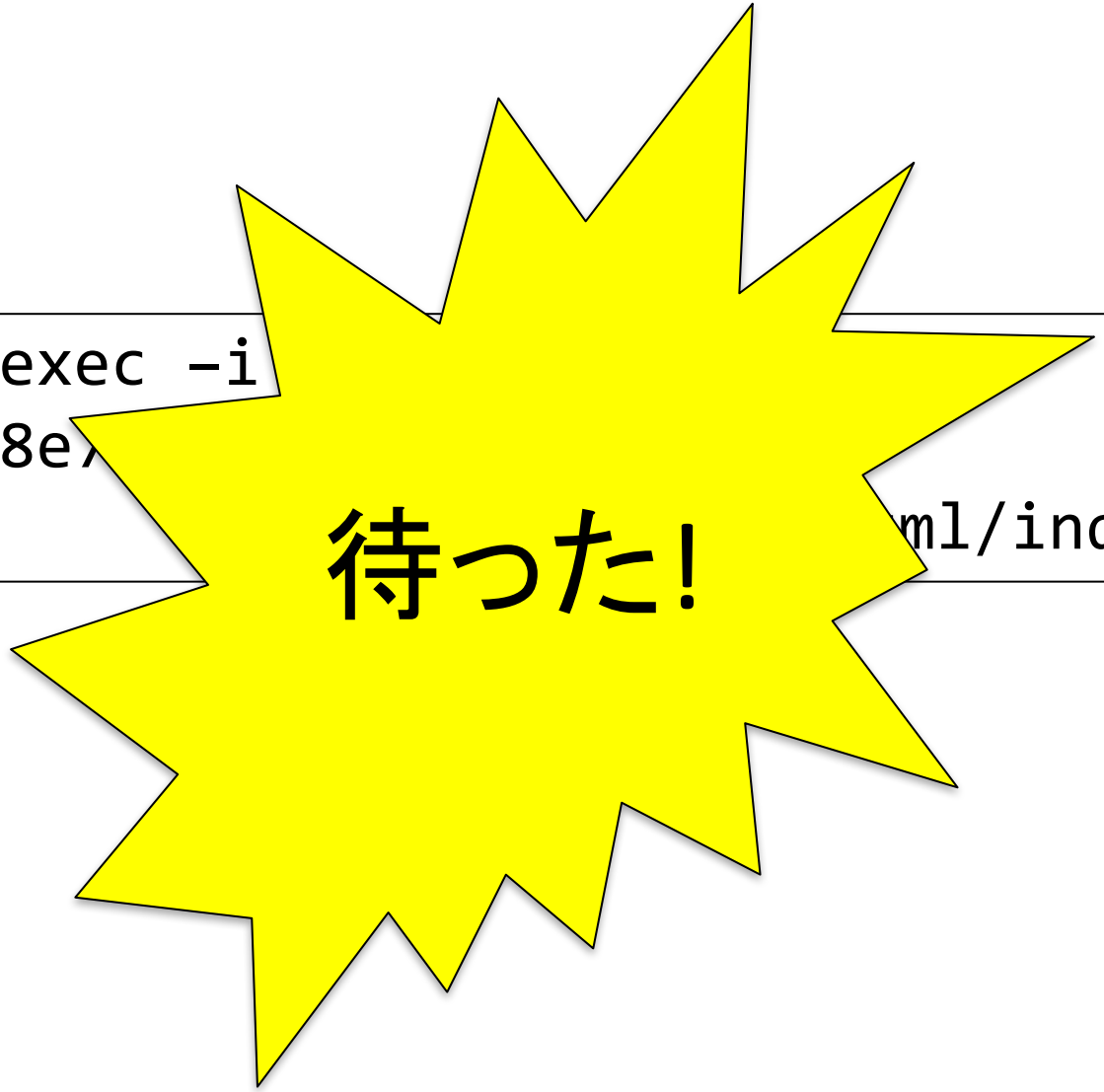
```
% vi nginx-files/index.html
```

ホストとコンテナでファイルを共有する: コンテナからファイルを編集する

```
% docker exec –i –t 87ee8 /bin/bash
root@87ee8e79f213:/# vim.tiny \
                /usr/share/nginx/html/index.html
```

ホストとコンテナでファイルを共有する: コンテナからファイルを編集する

```
% docker exec -i
root@87ee8e
                                    ml/index.html
```



コンテナからボリューム上のファイルをさわるときは注意が必要

```
root@87ee8e79f213:/# cd /usr/share/nginx/html
root@87ee8e79f213:/# touch index2.html
root@87ee8e79f213:/# exit
% ls –l nginx-files
```

コンテナからボリューム上のファイルをさわるときは注意が必要

```
root@87ee8e79f213:/# cd /usr/share/nginx/html
root@87ee8e79f213:/# touch index2.html
root@87ee8e79f213:/# exit
% ls –l nginx-files
```

```
total 0
-rw-r--r-- 1 ubuntu ubuntu 51 Mar 22 02:55 index.html
-rw-rw-r-- 1 root   root    0 Mar 22 03:00 index2.htm
```

コンテナからボリューム上のファイルをさわるときは注意が必要

```
root@87ee8e79f213:/# cd /usr/share/nginx/html
root@87ee8e79f213:/# touch index2.html
root@87ee8e79f213:/# exit
% ls –l nginx-files
```

```
total 0
-rw-r--r-- 1 ubuntu ubuntu 51 Mar 22 02:55 index.html
-rw-rw-r-- 1 root   root    0 Mar 22 03:00 index2.htm
```

コンテナ ボリ ファイルを

root 権限で
ファイルが
作れてしまった!

```
root@
root@
r
% ls -l ngi ies
```

```
total 0
-rw-r--r-- 1 ubuntu ubuntu 51 Mar 22 02:55 index.html
-rw-rw-r-- 1 root   root    0 Mar 22 03:00 index2.htm
```

コンテナからボリューム上のファイルをさわるときは注意が必要

```
root@680e0e43f517:/# cd /usr/share/nginx/html
root@680e0e43f517:/# touch index2.html
root@680e0e43f517:/# exit
% ls –l nginx-files
```

```
total 8
-rw-r--r-- 1 tosh  staff  51  3 22 14:30 index.html
-rw-rw-r-- 1 tosh  staff   0  3 22 14:35 index2.html
```

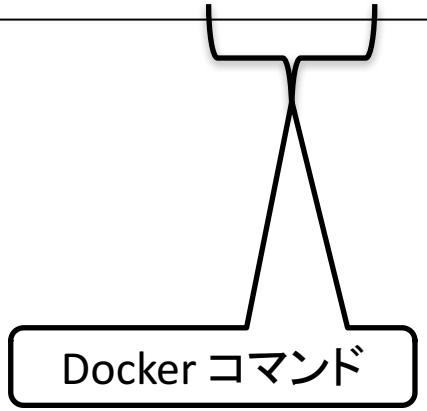
※ そうならない環境もあります (e.g., macOS)

イメージをファイルに保存する

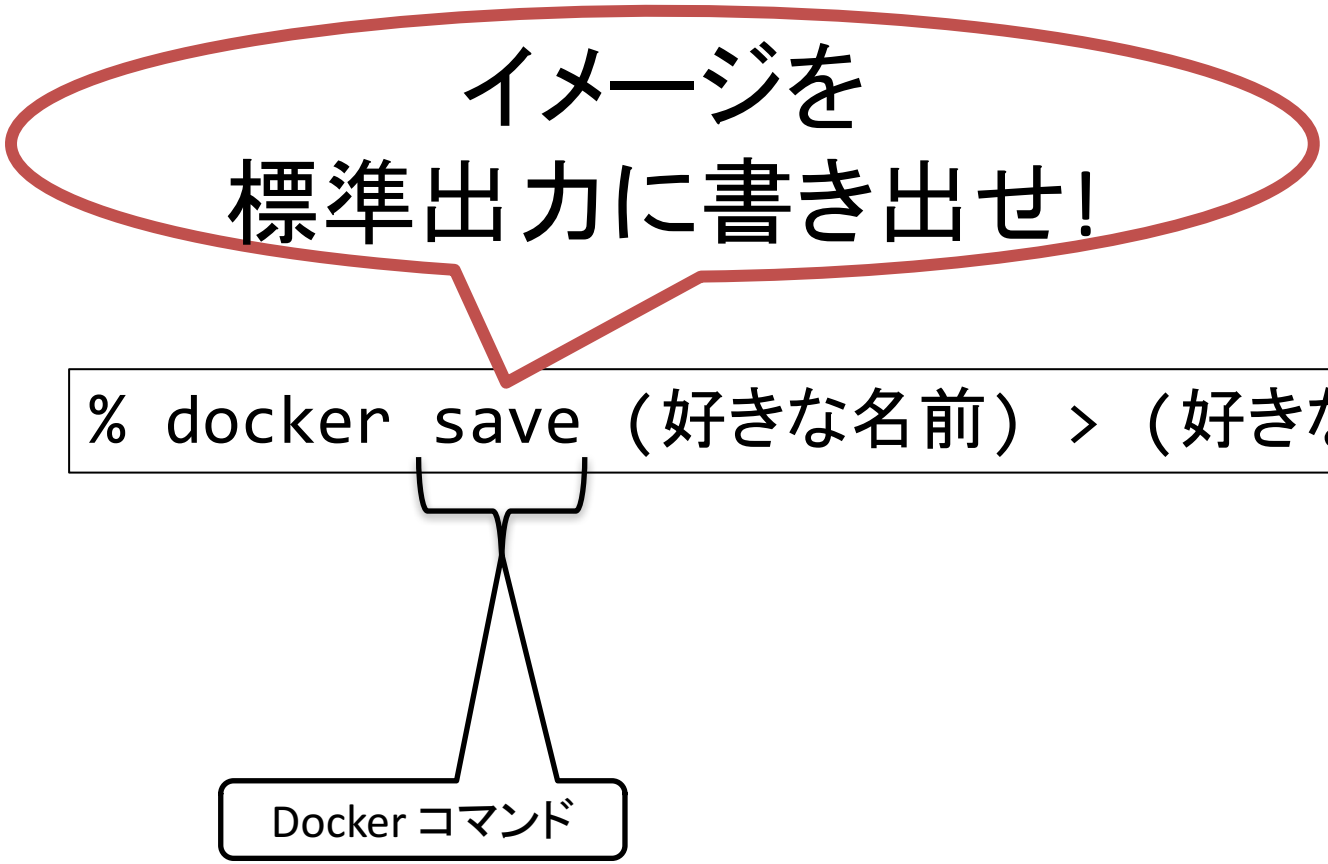
```
% docker save（好きな名前）>（好きなファイル名）.tar
```

イメージをファイルに保存する

```
% docker save（好きな名前）>（好きなファイル名）.tar
```

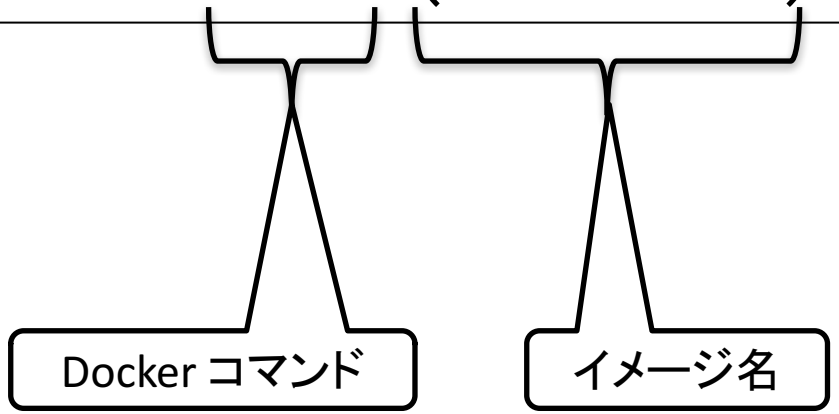


イメージをファイルに保存する

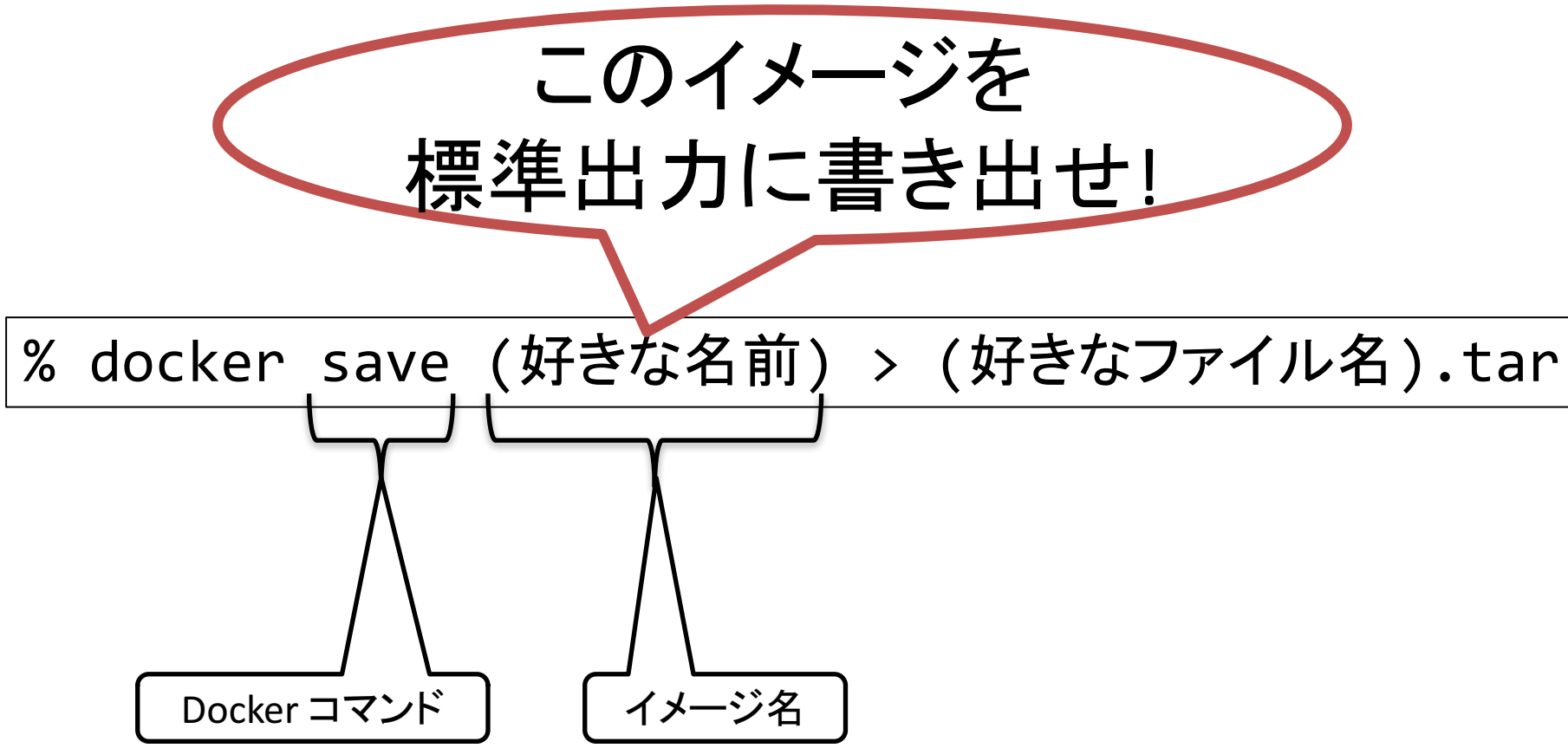


イメージをファイルに保存する

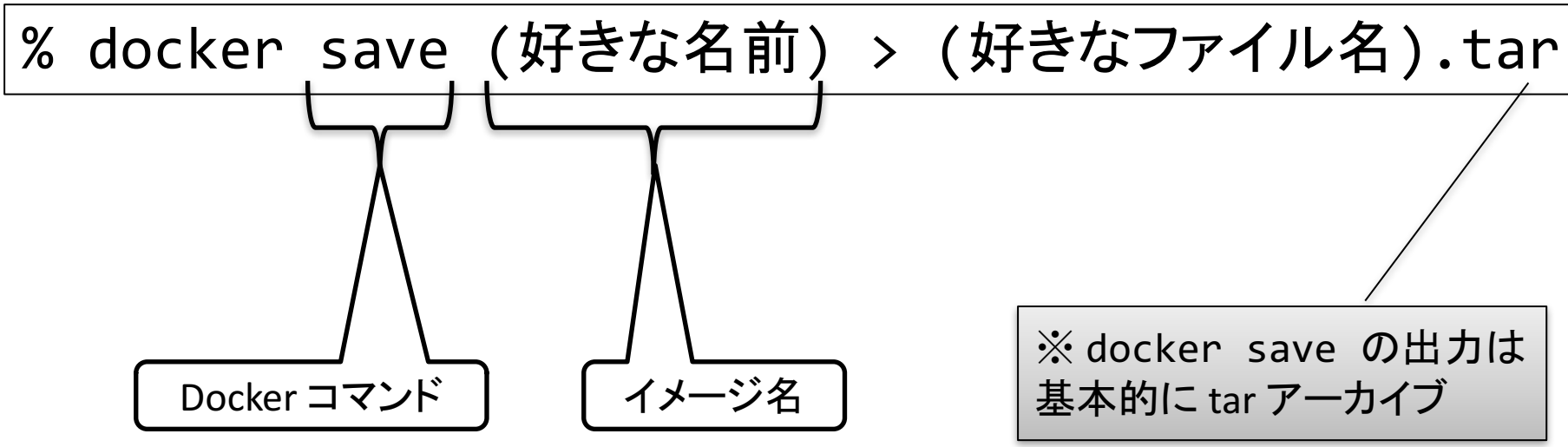
```
% docker save (好きな名前) > (好きなファイル名).tar
```



イメージをファイルに保存する



イメージをファイルに保存する



イメージをファイルに保存する

```
% docker save（好きな名前）>（好きなファイル名）.tar
```

不要なイメージを削除する

```
% docker rmi (好きな名前)
```

不要なイメージを削除する

```
% docker rmi (好きな名前)
```

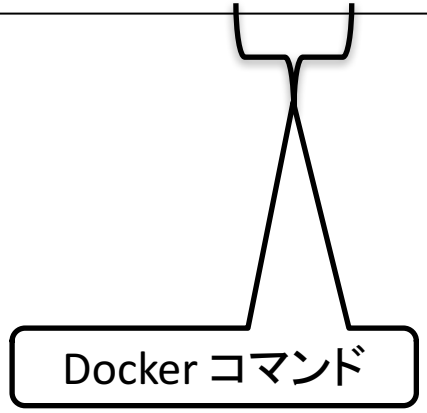
```
Deleted: sha256:96475fbbe6901fcf946502eebb31786
Deleted: sha256:be5c26bf4ce7314097263ed9f418939
```

不要なイメージを削除する

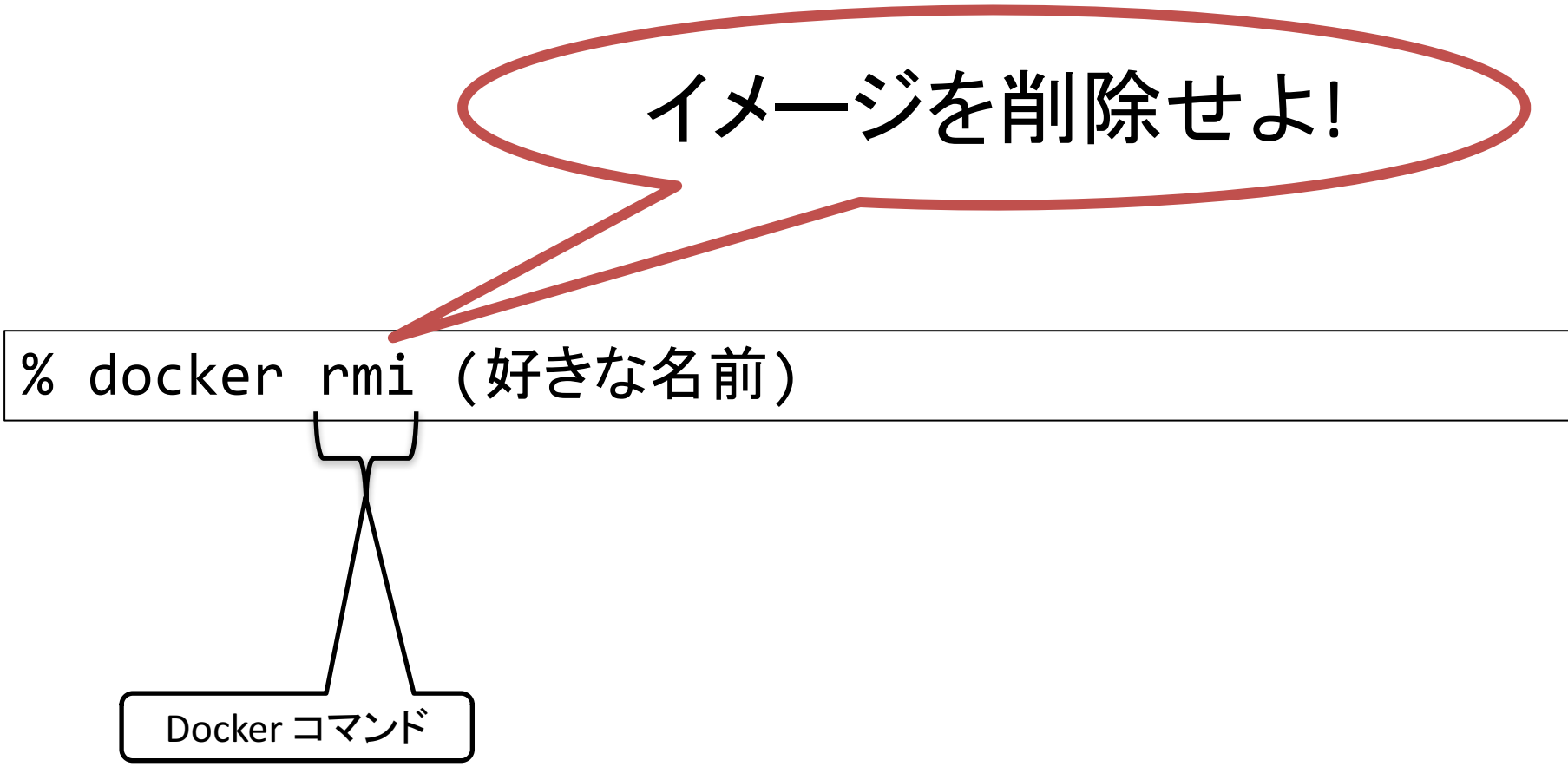
```
% docker rmi (好きな名前)
```

不要なイメージを削除する

```
% docker rmi (好きな名前)
```

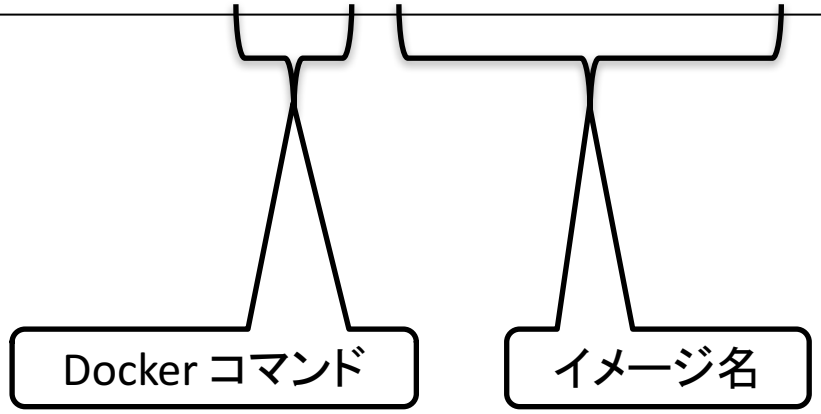


不要なイメージを削除する

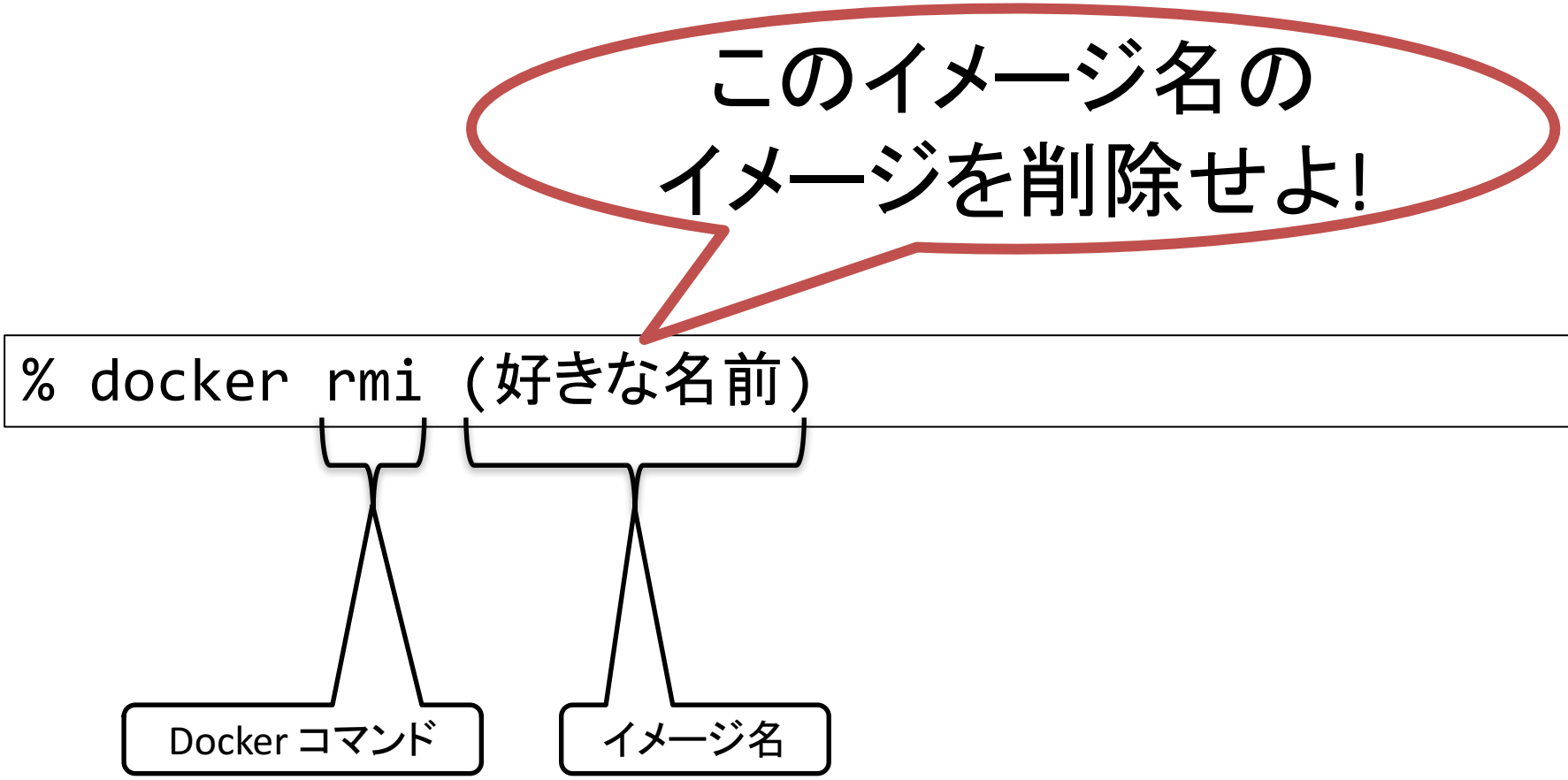


不要なイメージを削除する

```
% docker rmi (好きな名前)
```



不要なイメージを削除する



不要なイメージを削除する

```
% docker rmi (好きな名前)
```

不要なイメージを削除する

エラーが出ますか?

```
% docker rmi (好きな名前)
```

不要なイメージを削除する

エラーが出ますか?

```
% docker rmi (好きな名前)
```

そのイメージを使ったコンテナが存在していないか

`docker ps -a`

で確認して、

`docker stop` からの `docker rm`

で削除してください

不要なイメージを削除する

```
% docker rmi (好きな名前)
```

```
Deleted: sha256:96475fbbe6901fcf946502eebb31786
Deleted: sha256:be5c26bf4ce7314097263ed9f418939
```

ファイルからイメージをロードする

```
% docker load < (好きなファイル名).tar
```

ファイルからイメージをロードする

```
% docker load < (好きなファイル名).tar
```

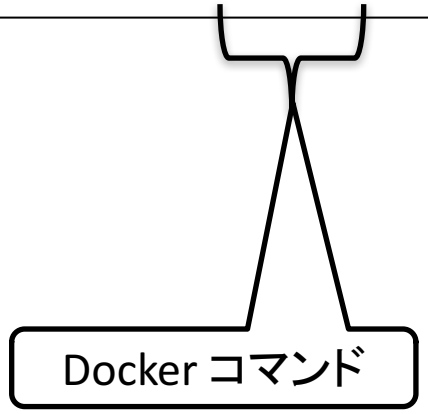
```
d17d48b2382a: Loading layer [==========>] 128.
cf0d2468e726: Loading layer [==========>]  60.
e3a86ffb1c45: Loading layer [==========>] 3.58
…
Loaded image (好きな名前):latest
```

ファイルからイメージをロードする

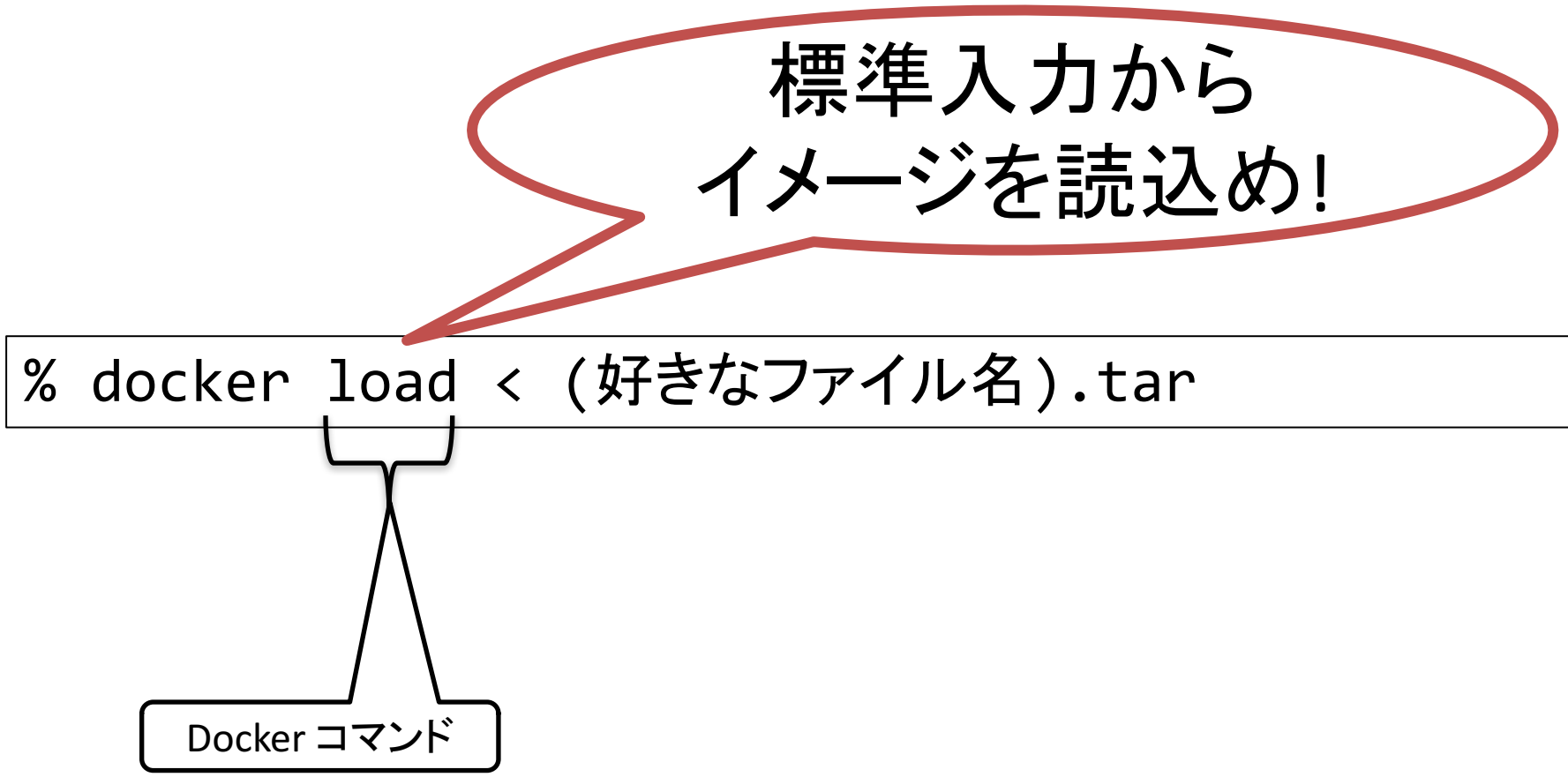
```
% docker load < (好きなファイル名).tar
```

ファイルからイメージをロードする

```
% docker load < (好きなファイル名).tar
```



ファイルからイメージをロードする



ファイルからイメージをロードする

```
% docker load < (好きなファイル名).tar
```

ファイルからイメージをロードする

```
% docker load < (好きなファイル名).tar
```

```
d17d48b2382a: Loading layer [==========>] 128.
cf0d2468e726: Loading layer [==========>]  60.
e3a86ffb1c45: Loading layer [==========>] 3.58
…
Loaded image (好きな名前):latest
```

出てきたコマンドの一覧

- Docker run: コンテナを実行する
  - Docker run -i -t シェルが標準入力を取れるように実行する
  - Docker run -d バックグラウンドで実行する
  - Docker run -p ポート番号をマップする
  - Docker run -v ボリュームをマウントする
- Docker stop コンテナを停止する
- Docker restart コンテナを再起動する
- Docker search イメージを検索する
- Docker pull イメージをダウンロードする
- Docker images イメージの一覧を得る
- Docker ps 実行中のコンテナの一覧を得る
  - Docker ps –a (停止中のものも含め) 全てのコンテナの一覧を得る
- Docker rm コンテナを削除する
- Docker rmi イメージを削除する
- Docker attach 実行中のコンテナに接続する
- Docker exec 実行中のコンテナ内でコマンドを実行する
- Docker commit コンテナからイメージを作成する
- Docker save イメージを標準出力へ書き出す
- Docker load 標準入力からイメージをロードする

Docker の特徴

- とても簡単
- とても軽い

Docker の特徴

- とても簡単
- とても軽い

Docker の性能は?

- Linux 版は何もしないで実行しても
  ほとんどネイティブ実行と変わらないか、
  少し悪いくらい
  - CPU: ほとんど変わらない
  - ファイル I/O: ほとんど変わらないか若干悪化
  - ネットワーク: 若干悪化
    - デフォルトの設定だと NAT 処理がはさまるため(?)

- macOS 版や Windows 版は xhyve や Hyper-V のオーバーヘッドのため(?)若干性能が落ちる
  - xhyve: macOS の仮想化ハイパーバイザ
  - Hyper-V: Windows の仮想化ハイパーバイザ

Docker の性能は?

- ちょっと工夫すると
  ネイティブ実行と区別つかないくらいになる
  - ファイル I/O:
    - ボリュームを使ってしまえばよい (i.e., -v オプションを使う)
      → ネイティブとほぼ同性能が出る
  - ネットワーク I/O:
    - ホスト側のネットワークをそのまま使ってしまえばよい
      → ネイティブとほぼ同性能がでる
      - コンテナ起動時に `--net=host` オプションを指定すると
        コンテナがホストのネットワークスタックを直接叩くようになる

Docker の性能は?

- ちょっと特殊なデバイスも使える
  - デバイスファイルを `--device` オプションで
    コンテナ内にマウントできる
  - NVDIA:
    - https://github.com/NVIDIA/nvidia-docker
  - Infiniband:
    - /dev/infiniband 下のデバイスファイルを
      コンテナ内にマウントすれば OK (らしい)

Docker のセキュリティは?

Docker のセキュリティは?

Docker のセキュリティは?

- 事実上、
  「docker コマンドを叩ける」
  ==「root 権限を持っている」

- 率直に言えば、現時点では
  生の Docker を一般ユーザに
  そのまま使わせるのは
  管理者の視点からは危険すぎる

Docker のセキュリティ、ここがやばい (その1)

- ボリューム機能がセキュリティについて何も考えてなさすぎ

```
% docker run –v /:/foo –i –t ubuntu /bin/bash
root@9ab5647d7695:/# cat /foo/etc/shadow
root@9ab5647d7695:/# cp /bin/sh /foo/
root@9ab5647d7695:/# chown root:root /foo/sh
root@9ab5647d7695:/# chmod a+s /foo/sh
```

- 真面目に解決するには SELinux などの設定が必要

Docker のセキュリティ、ここがやばい (その2)

- ユーザ間の区別がないに等しい
  - 誰のイメージでもアクセス・実行・削除等が可能
  - 誰のコンテナでもアクセス・実行・削除等が可能

現在よく取られている現実的な解決策

- Docker とは別にユーザ隔離できる仮想環境を
  ユーザごとに用意して
  その中で Docker を使わせる

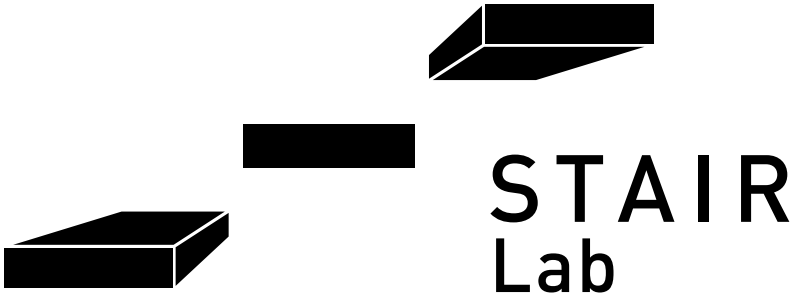
  – 例: Amazon ECS, Google Container Engine, …

Docker にはまだまだ
いろいろな機能・ツールがいっぱい

- 今日の内容は基礎中の基礎だけ

Docker にはまだまだ いろいろな機能・ツールがいっぱい

- Dockerfile
  - Makefile の Docker 版と考えればよい
    - イメージをビルドする手順を指定する
- docker-compose
  - 複数のコンテナを組み合わせた環境を構築・実行するのに便利
- docker-machine
  - AWS や GCE や SoftLayer 等の上でコンテナを実行させるのに便利
- docker swarm
  - Docker が動作する複数のホストをまとめて管理するのに便利
- docker network
  - 仮想的なネットワークを構築する
    - 物理的に異なる複数ホスト間で
      仮想的なネットワークを構築することもできる
- …


STAIR
Lab