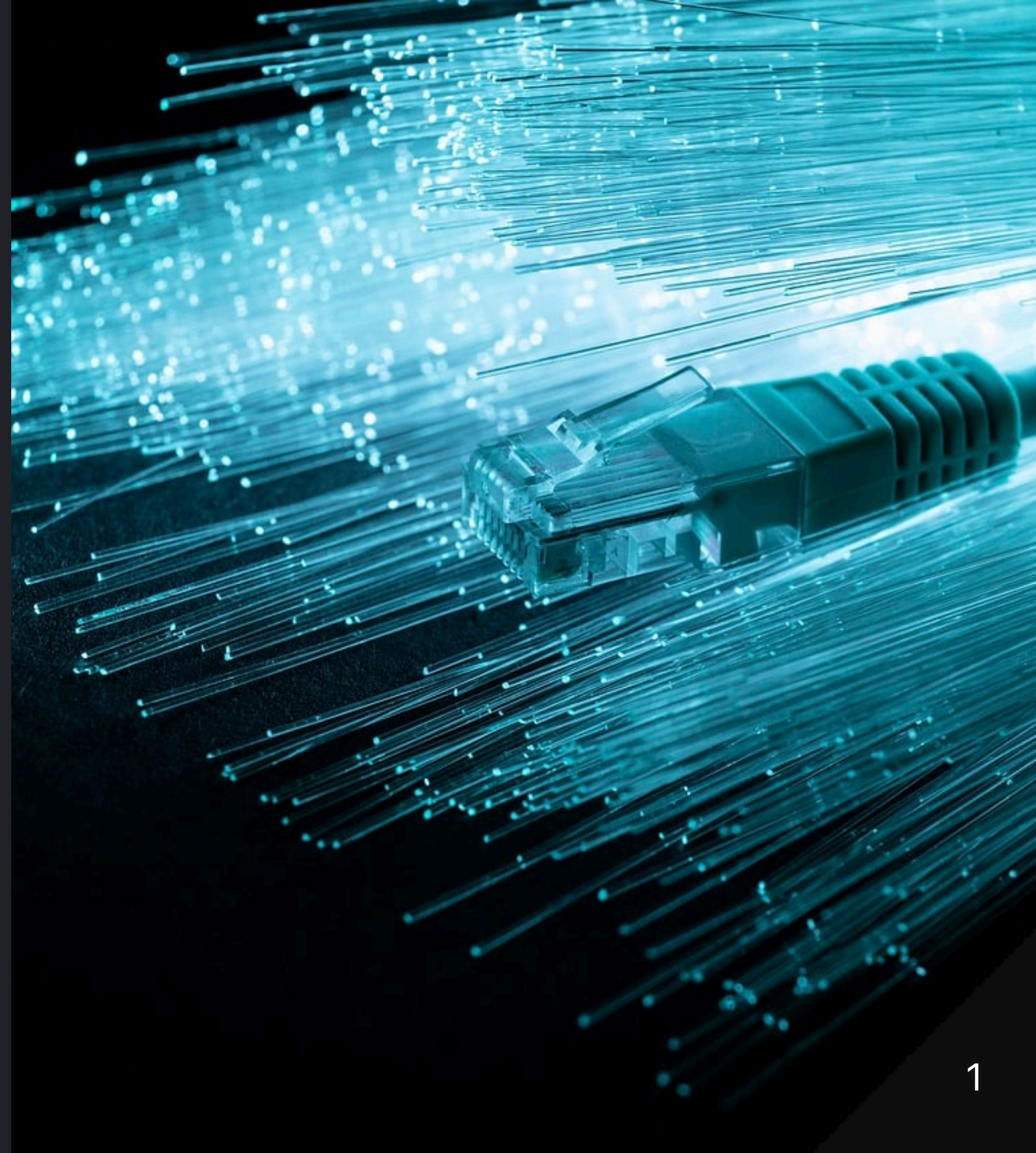
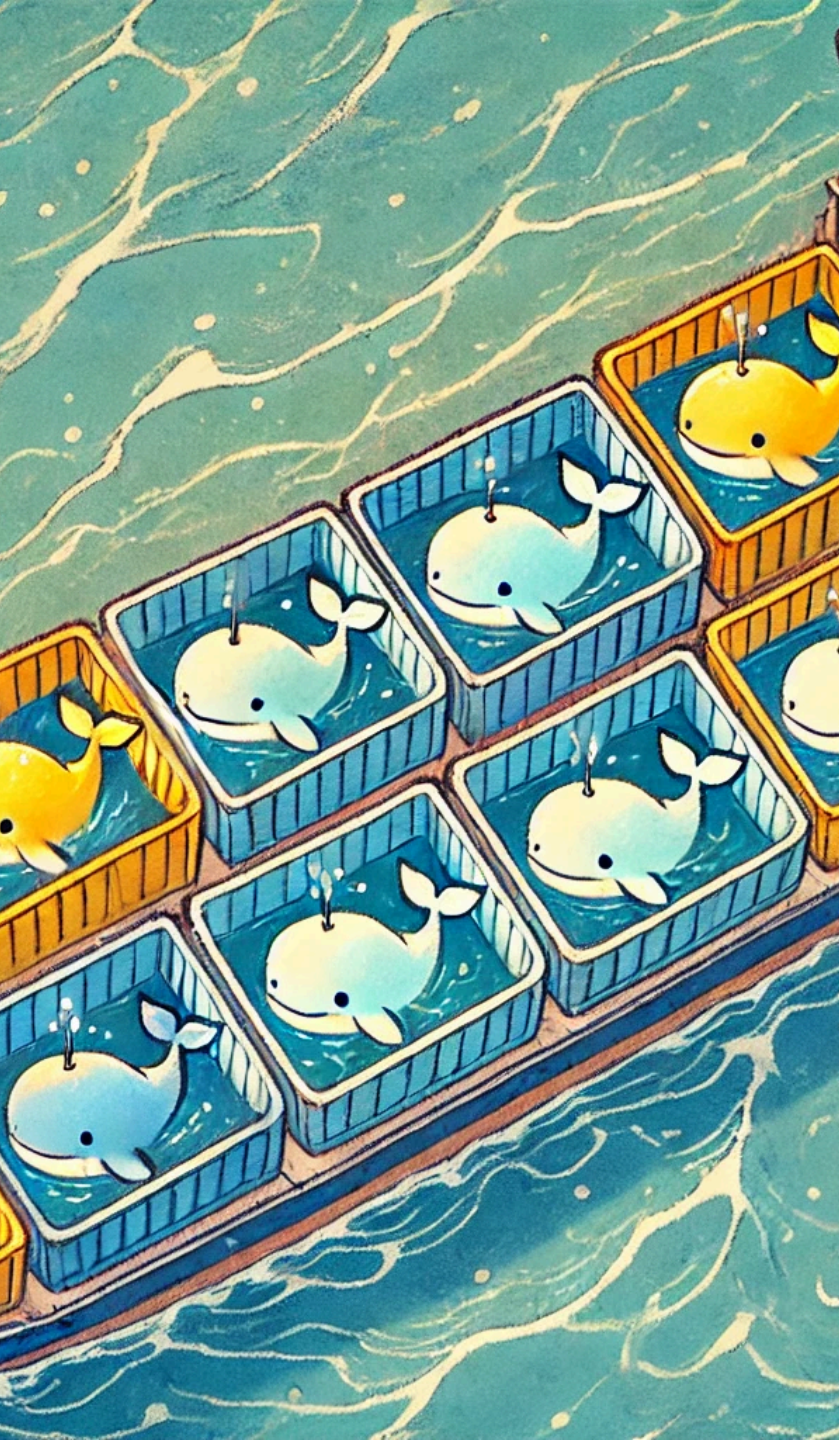


# Réseau



# Un conteneur, et le reste du monde



# Un conteneur a son propre réseau

- Comme il a son propre nom de machine et ses propres processus
- Vous l'avez vu au chapitre 01 : isolé, il ne joint plus personne
- Mais Docker lui branche de quoi communiquer, dès le premier `docker run`

# Les quatre questions du chapitre

- Comment un conteneur accède-t-il à internet ?
- Comment deux conteneurs se parlent-ils ?
- Comment les joint-on depuis l'extérieur ?
- Comment empêche-t-on les autres de se parler ?

# Ce que Docker branche tout seul



# Trois réseaux livrés d'office

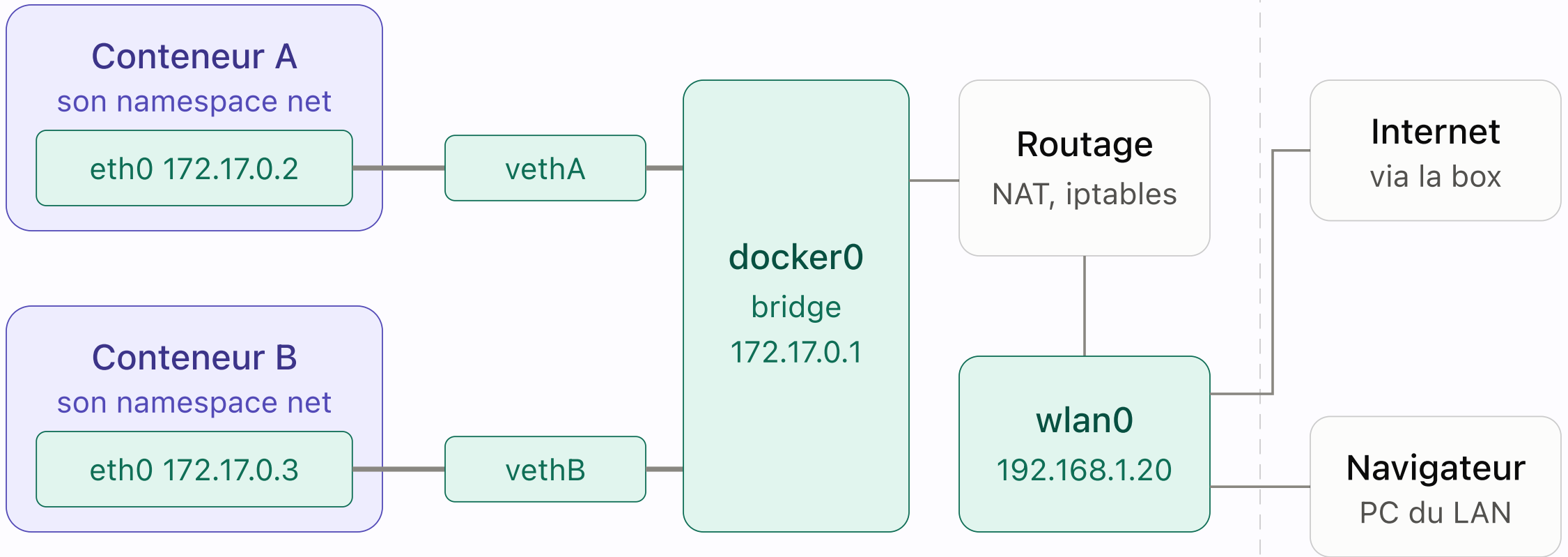
- `bridge` — celui que tout conteneur reçoit par défaut
- `host` — le conteneur utilise le réseau de la machine
- `none` — pas de réseau du tout

```
docker network ls
```

# Le réseau par défaut

- Un réseau privé, interne à la machine, géré par Docker
- Tout conteneur lancé sans rien préciser s'y retrouve
- Il y reçoit une adresse, choisie par Docker

## Hôte Linux



Interfaces réseau



Namespace net d'un conteneur



Paire veth (câble)

# Sortir vers internet : rien à faire

- Un conteneur joint l'extérieur dès sa création
- Il passe par la machine, qui relaie pour lui
- Comme vos appareils passent par votre box

“ Vu de l'extérieur, c'est la machine hôte qui parle, pas le conteneur. ”

# Entre conteneurs : par adresse

- Sur ce réseau, les conteneurs peuvent se joindre par leur **adresse**
- Leur nom, lui, ne veut rien dire
- Et cette adresse change à chaque recreation du conteneur

# Donc : à éviter

- Tout le monde y est, personne n'y est vraiment isolé
- Les adresses y sont instables
- Docker lui-même recommande de ne pas s'en servir



# Le cas classique

# Un réseau, une commande

```
docker network create appli  
docker run --network appli --name web nginx
```

- Seuls les conteneurs qu'on y attache s'y trouvent
- C'est le réseau d'une application

# Le nom suffit

- Sur un réseau qu'on a créé, le **nom du conteneur** se résout via un DNS interne
- Docker tient l'annuaire, et le met à jour tout seul

```
ping web
```

# Pourquoi ça change tout

- L'adresse d'un conteneur n'est pas stable, son nom l'est
- On le recrée, on le remplace, on le redémarre : le nom suit
- Les configurations s'écrivent donc avec des noms

# Les alias

- `--network-alias` ajoute un nom supplémentaire
- Utile quand le service a un nom et le conteneur un autre
- Plusieurs conteneurs peuvent porter le même alias

# Attacher, détacher

- `docker network connect` — brancher un conteneur en marche
- `docker network disconnect` — le débrancher
- Un conteneur peut être sur plusieurs réseaux à la fois

# Deux réseaux ne se parlent pas

- Un conteneur de l'un ne joint pas un conteneur de l'autre
- Ni par son nom, ni par son adresse
- C'est une brique d'isolation essentielle



# Joindre un conteneur de l'extérieur

# Le problème, dans l'autre sens

- Un conteneur sort, mais rien n'entre
- Votre navigateur ne sait pas atteindre ce réseau privé
- Il faut lui ouvrir une porte sur la machine

# **-p** , la porte d'entrée

```
-p 8080:80
```

- À gauche le port de la **machine**, à droite celui du **conteneur**
- Ce qui arrive sur **8080** est transmis au conteneur
- La porte se referme quand le conteneur disparaît

# Deux formes utiles

```
-p 8080:80  
-p 127.0.0.1:8080:80
```

- Sans adresse, la porte est ouverte à tout le réseau local
- Avec `127.0.0.1`, seule votre machine peut entrer

# Un port de la machine ne sert qu'une fois

- Deux conteneurs ne peuvent pas publier sur le même 8080
- Le second refuse de démarrer
- Côté conteneur, en revanche, le même port se répète sans gêne

# Voir ce qui est publié

- La colonne `PORTS` de `docker ps` liste les portes ouvertes
- `docker port` répond pour un conteneur donné

```
docker port mon-conteneur
```

# Publier, c'est ouvrir

- Une porte publiée l'est pour tout le réseau local
- On ne publie que ce qui doit vraiment être joignable
- Tout le reste se joint de l'intérieur, par son nom

“

Une base de données ne se publie pas.

”



**host et  
none**

# **host** : le réseau de la machine

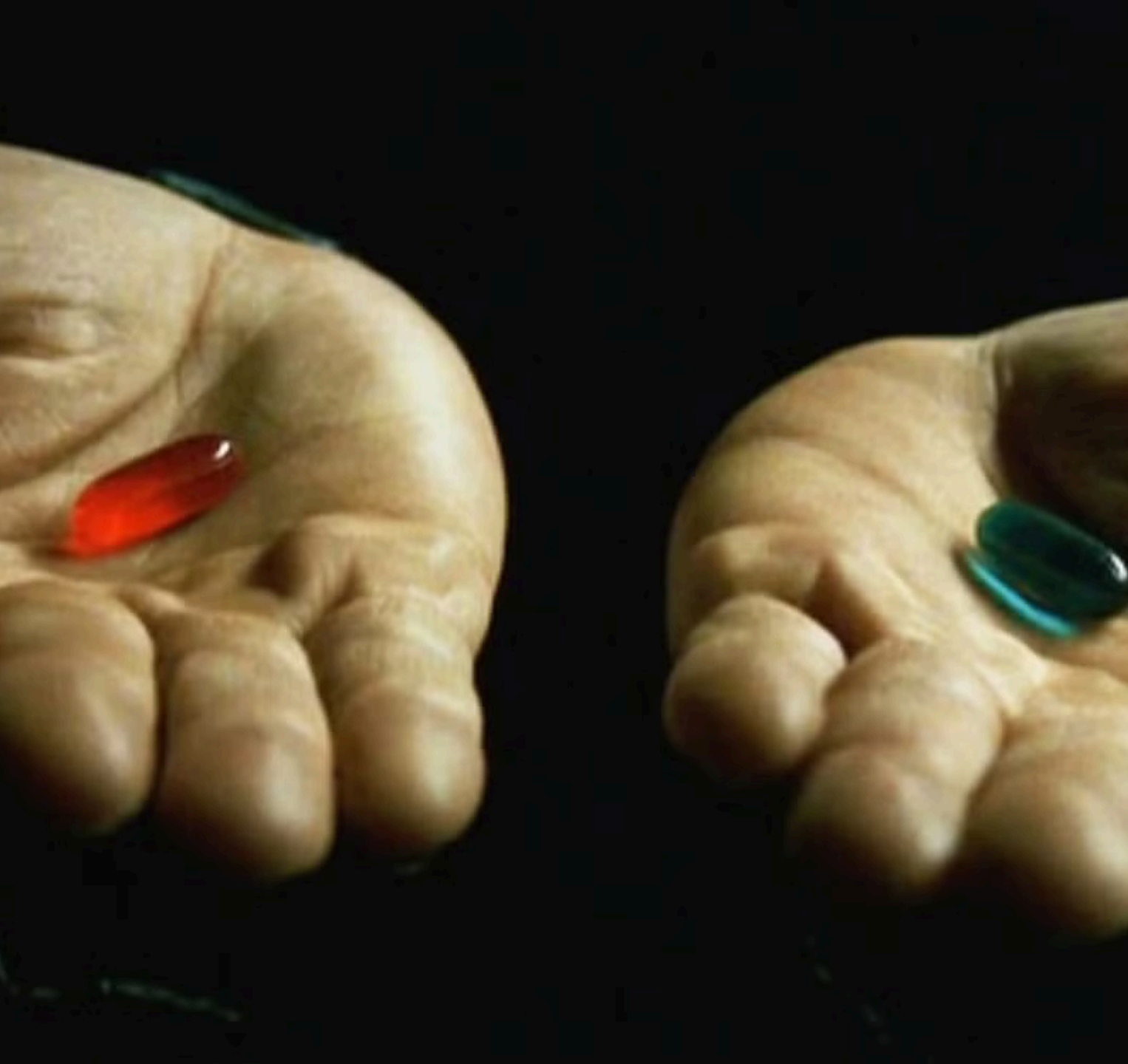
- Le conteneur n'a plus de réseau à lui
- Il utilise directement celui de la machine
- Ses ports sont ceux de la machine, et **-p** ne sert plus à rien

# Ce qu'on y perd

- Le cloisonnement : le conteneur voit le réseau comme la machine
- Le choix des ports : ils sont pris tels quels, sans translation
- La portabilité : sur Mac et Windows, ce n'est pas votre réseau

# **none** : pas de réseau

- Le conteneur n'a aucun moyen de communiquer
- Il ne sort pas, et personne ne le joint
- Il lit et écrit normalement ce qu'on lui monte



# Choisir

# Un réseau à soi : par défaut

- Dès que deux conteneurs doivent se parler
- Dès qu'on veut séparer des services
- Se comporte partout de la même façon

“

C'est le choix dans neuf cas sur dix.

”

# Le réseau par défaut : dépannage

- Un conteneur seul, un essai de trente secondes
- Rien à nommer, rien à isoler
- Dès que ça devient sérieux, on crée un réseau

# **host** : le cas particulier

- Quand le conteneur doit voir le réseau de la machine
- Supervision, découverte d'équipements, trop de ports à publier
- Choix rare, qui se justifie : on y perd le cloisonnement

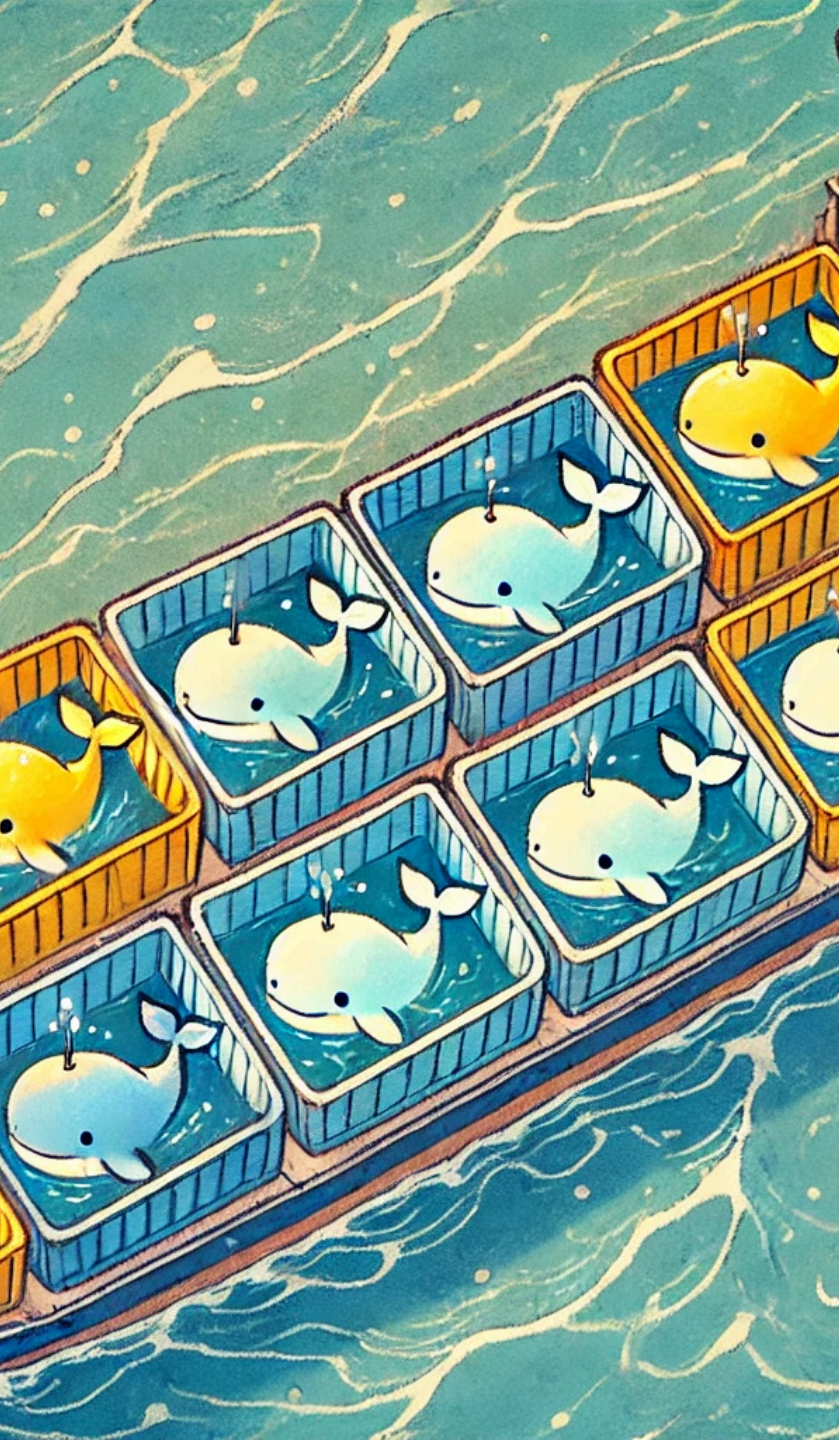
# **none** : le bac à sable

- Un traitement qui n'a aucune raison de communiquer
- Un programme qu'on ne connaît pas et qu'on préfère couper du monde
- On lui monte ses fichiers, et c'est tout

# Le réflexe

“ Dans le doute, un réseau à soi. `host` et `none` se justifient, le réseau par défaut se subit. ”

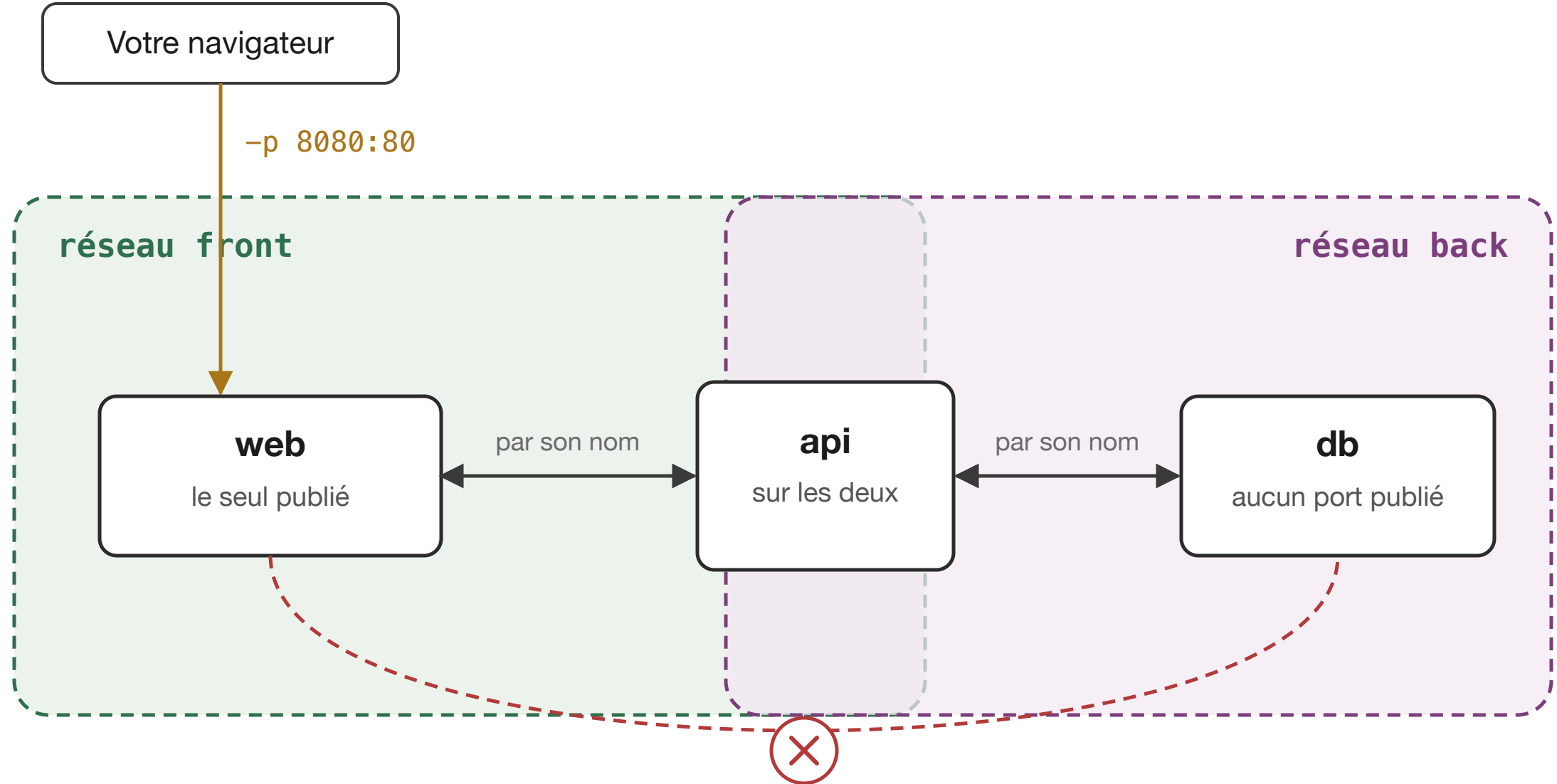
# Découper par réseaux



# Le motif classique

- Un réseau **front** : ce qui est joignable de l'extérieur
- Un réseau **back** : la base de données, les files, les caches
- Un seul service à cheval sur les deux

# Isoler par réseaux



web ne joint pas db, ni par son nom ni par son adresse

# Ce que ça garantit

- La base n'a aucune porte ouverte : l'extérieur ne la voit pas
- Le serveur web ne la voit pas non plus
- Seule l'API, attachée aux deux réseaux, y accède

# La règle de conception

- Un réseau par frontière de confiance, pas un par service
- On ne publie que la porte d'entrée
- Tout le reste se joint par son nom, de l'intérieur

# À retenir (1/2)

- Le réseau par défaut relie tout le monde et ne connaît aucun nom
- Sur un réseau qu'on crée, un conteneur se joint par son nom
- Un nom est stable, une adresse ne l'est pas

## À retenir (2/2)

- `-p` ouvre une porte sur la machine, et elle l'est pour tout le réseau
- Un port de la machine ne sert qu'une fois, pas celui du conteneur
- Dans le doute, un réseau à soi : `host` et `none` sont des exceptions

# Réseau

