Démons en PHP

de inetd à ZeroMQ

Démons en PHP

- **Introduction**
- **Démons basés sur (x)inetd**
- **Démons multi-processus**
- **Démons mono-processus**
- **ZeroMQ**

Introduction

« Démon » ?

Nom masculin
Barbarisme (daemon)
« Disk And Execution MONitor »

Programme indépendant
Tourne en tâche de fond

Serveur logiciel
Accepte des connexions entrantes

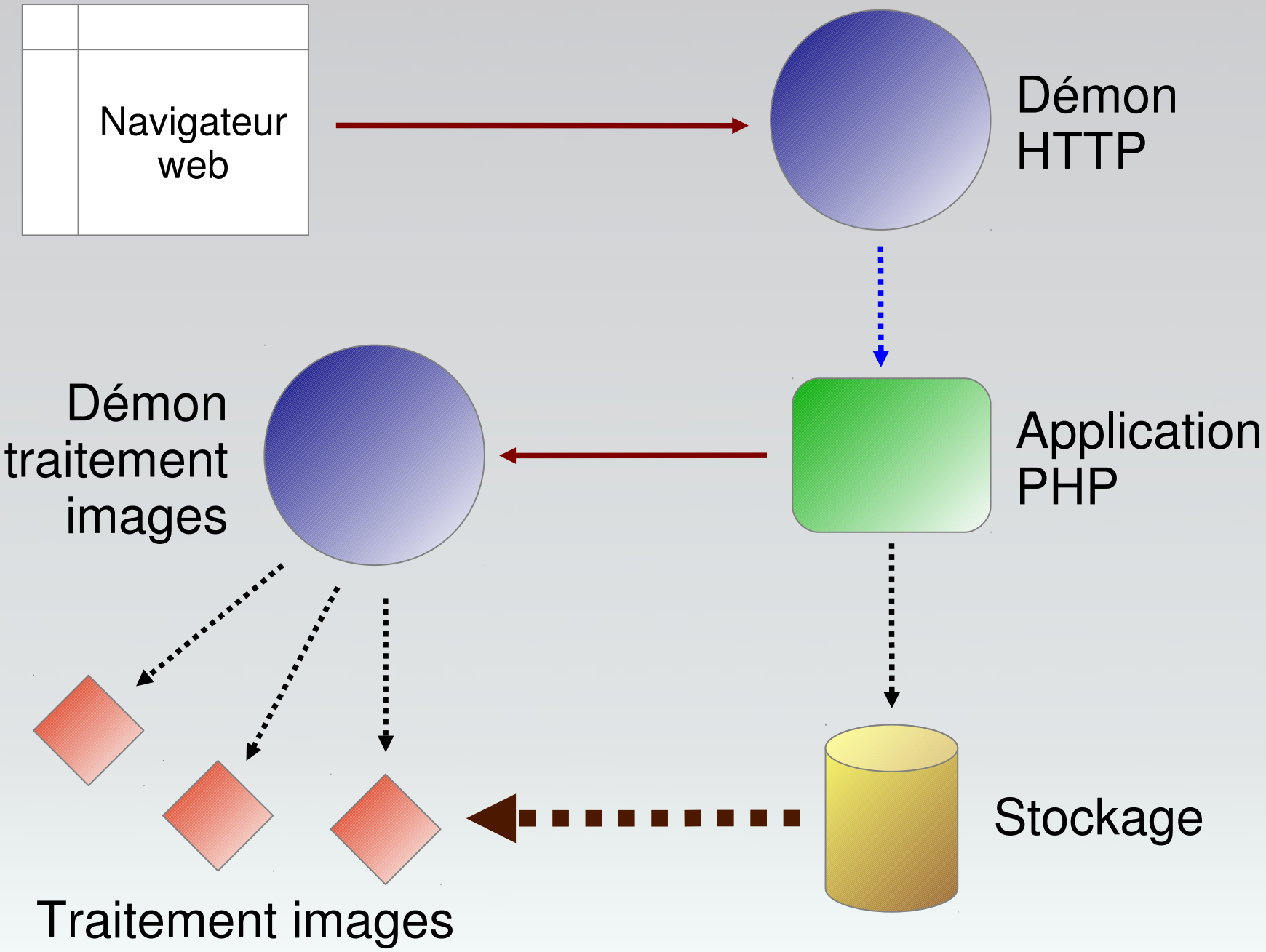
Quelques exemples

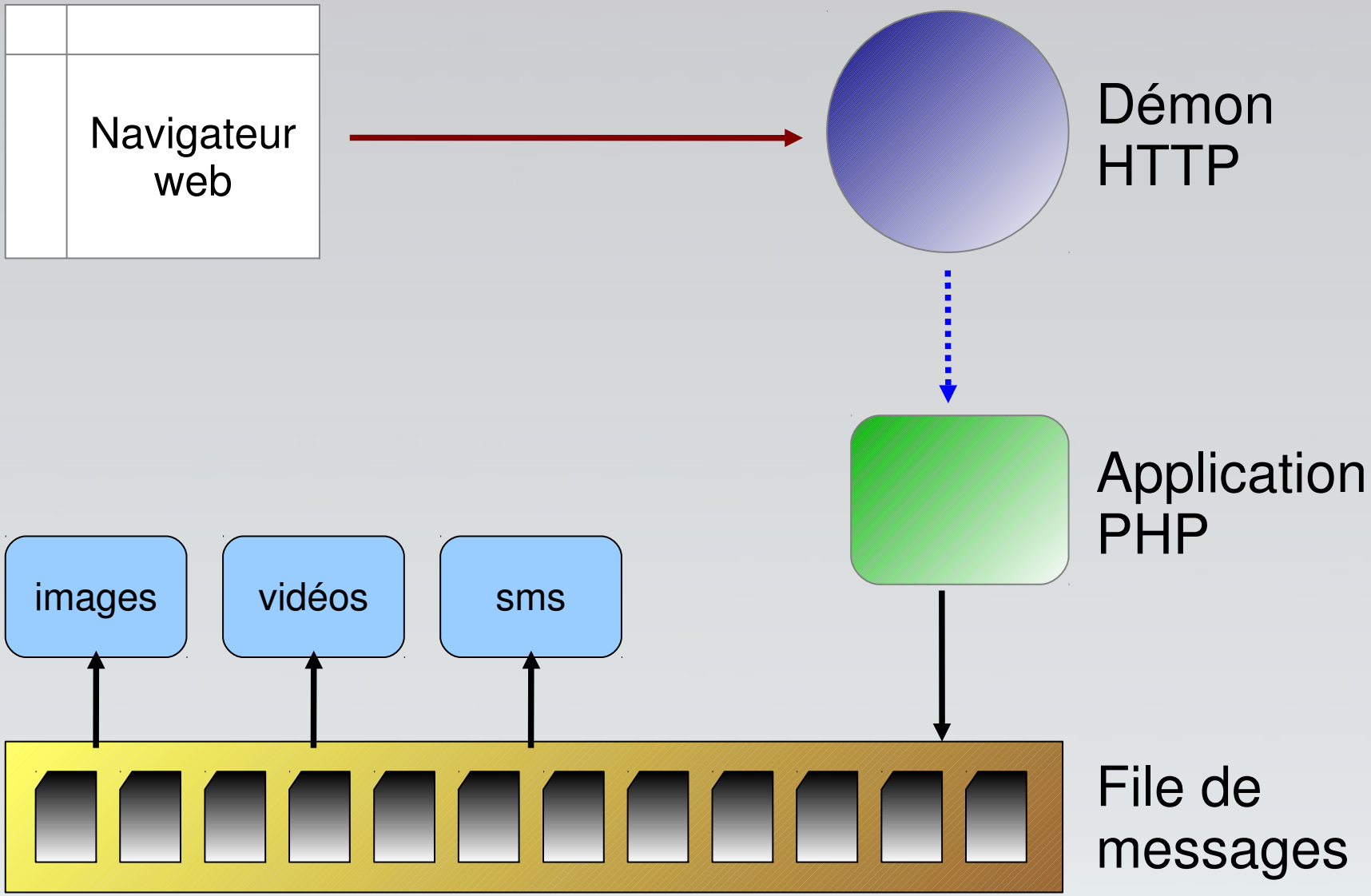
Démons réseau	Démons applicatifs
Apache (httpd)	Crontab (crond)
Impression (lpd / cupsd)	Périphériques (devfsd)
Connexion distante (sshd)	X server

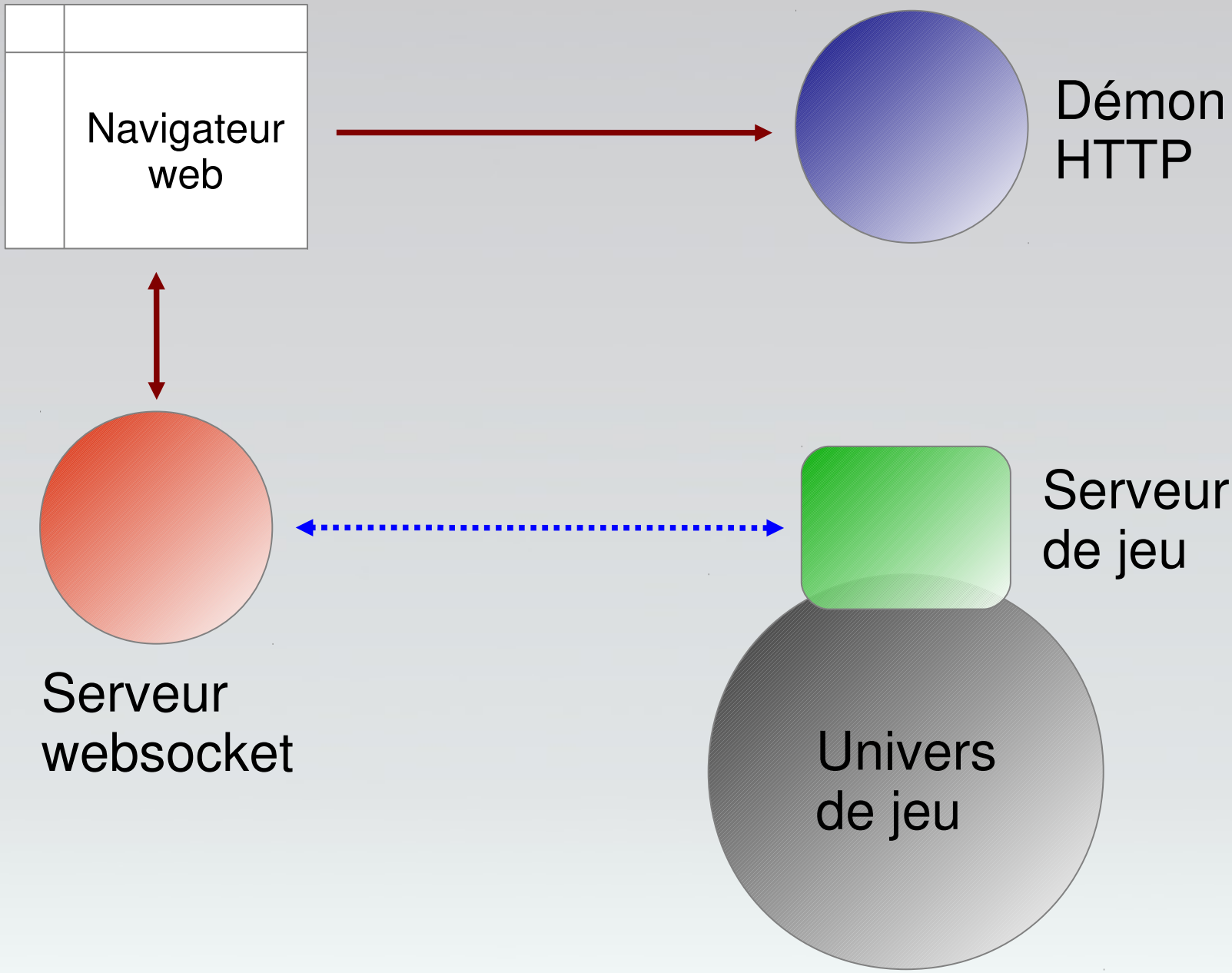
Mais pourtant...

~~**Le démon d'une application web,**~~
~~**c'est Apache / Lighttpd / Nginx !**~~

C'est de moins en moins vrai

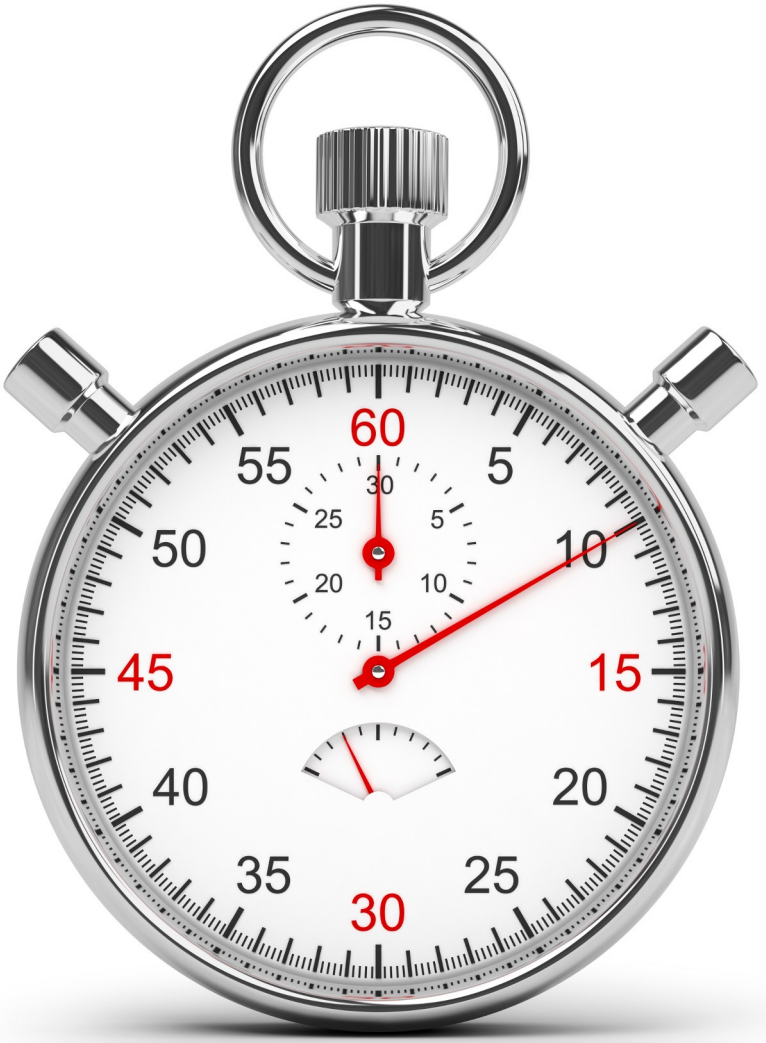

Navigateur web
Démon HTTP
Application PHP
Démon traitement images
Stockage
Traitement images


Navigateur web
Démon HTTP
Application PHP
images
vidéos
sms
File de messages


Navigateur
web
Démon
HTTP
Serveur
de jeu
Serveur
websocket
Univers
de jeu

Démons en PHP

Avantages	Inconvénients
Développement rapide	Connaissances requises
Bonnes performances	Consommation mémoire
Réutilisation de code	


60
55
5
50
10
45
15
40
20
35
25
30
30
25
5
20
10
15

Benchmark

ATOM N270 1.6 GHz – RAM 1 GO

Connexion unique

- Temps de connexion
- Envoi et réception (2 messages)

Connexions multiples

- 100 / 1000 connexions
- Temps de connexion
- Envoi et réception (1 message)

Implémentation de référence en C

Exemple : myecho

Code C

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <signal.h>

int main(void)
{
   int                  sock;
   int                  client_sock;
   struct sockaddr_in   addr;
   struct sockaddr_in   client_addr;
   socklen_t            addr_sz;
   size_t               sz;
   int                  offset = 0;
   char                 buffer[512];

   bzero(&addr, sizeof(addr));
   addr.sin_port = htons(11142);
   addr.sin_family = AF_INET;
   addr.sin_addr.s_addr = htonl(INADDR_ANY);

   sock = socket(AF_INET, SOCK_STREAM, 0);
   signal(SIGCHLD, SIG_IGN);
   bind(sock, (struct sockaddr*)&addr,
        sizeof(addr));
   listen(sock, 5000);

   addr_sz = sizeof(client_addr);
   while (1)
   {
      client_sock = accept(sock, (struct
                           sockaddr*)&client_addr,
                           &addr_sz);
      if (fork() == 0)
      {
         close(sock);
         while ((sz = read(client_sock, buffer,
                           512)) > 0)
         {
            write(client_sock, buffer, sz);
         }
         close(client_sock);
         exit(0);
      }
   }
   close(sock);
   return (0);
}
```

Exemple : myecho

Exécution

```
# telnet localhost 11142
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hello
Hello
Test 1
Test 1
Test 2
Test 2
```

Benchmark : connexion unique

Type	Connexion	Échange 1	Échange 2
C	5 µs	3 µs	2 µs

Benchmark : connexions multiples

Type	100 connexions	100 messages	1000 connexions	1000 messages	Maximum
C	0.04 s	0.05 s	6.5 s	0.2 s	> 1000

Démons basés sur (x)inetd

(x)inetd : super-démon

Écoute plusieurs ports

Instancie un programme à chaque connexion entrante

Transmission sur les entrées-sorties standards

Exemple : myecho

Code PHP

```
#!/usr/bin/php
<?php

while (!feof(STDIN)) {
    $s = fgets(STDIN);
    print($s);
}

?>
```

/home/amaury/myecho.php

Exemple : myecho

Configuration xinetd

```
service myecho
{
    type          = UNLISTED
    user          = root
    wait          = no
    port          = 11138
    server        = /home/amaury/myecho.php
    protocol      = tcp
    socket_type = stream
}
```

/etc/xinetd.d/myecho

(x)inetd

Avantages	Inconvénients
Développement rapide	Consommation mémoire
Aucune dépendance	Lenteur d'instanciation
Isolation des processus	Couche intermédiaire

Benchmark : connexion unique

Type	Connexion	Échange 1	Échange 2
C	5 µs	3 µs	2 µs
xinetd	5 µs	31 ms	2 µs

Benchmark : connexions multiples

Type	100 connexions	100 messages	1000 connexions	1000 messages	Maximum
C	0.04 s	0.05 s	6.5 s	0.2 s	> 1000
xinetd	0.4 s	1.8 s	-	-	< 120

Démons multi-processus

Principe

Simple : *inetd internalisé*

Un processus attend les connexions

Création d'un nouveau sous-processus à chaque connexion entrante

Exemple : myecho

Code PHP

```
#!/usr/bin/php
<?php

daemonize();
$socket = stream_socket_server('tcp://0.0.0.0:11138',
                               $errno, $errstr);
while (true) {
    $clientSock = stream_socket_accept($socket, -1);
    if (pcntl_fork() === 0) {
        fclose($socket);
        process($clientSock);
        exit(0);
    }
    fclose($clientSock);
}
```

Exemple : myecho

Code PHP

```php
function daemonize() {
    pcntl_signal(SIGCHLD, SIG_IGN);
    if (pcntl_fork())
        exit(0);
    posix_setsid();
}

function process($sock) {
   while (!feof($sock)) {
      $s = fgets($sock);
      fwrite($sock, $s);
   }
   fclose($sock);
}
```

Multi-processus

Avantages	Inconvénients
Latence réduite	Consommation mémoire
Aucune dépendance	Gestion des signaux
Isolation des processus	PHP non multi-thread

Benchmark : connexion unique

Type	Connexion	Échange 1	Échange 2
C	5 µs	3 µs	2 µs
xinetd	5 µs	31 ms	2 µs
multi	5 µs	8 µs	2 µs

Benchmark : connexions multiples

Type	100 connexions	100 messages	1000 connexions	1000 messages	Maximum
C	0.04 s	0.05 s	6.5 s	0.2 s	> 1000
xinetd	0.4 s	1.8 s	-	-	< 120
multi	3 s	0.05 s	60.5 s	0.2 s	> 1000

Démons mono-processus

Principe

Un seul processus

> attend les nouvelles connexions

> répond aux requêtes

Mono versus multi-processus

Multi	Mono
I/O lentes	I/O rapides
Code simple	Partage de données
Limites vite atteintes	Seul moyen de gérer beaucoup de connexions

Exemple : myecho

Code PHP

```
#!/usr/bin/php
<?php

daemonize();
$socket = stream_socket_server('tcp://0.0.0.0:11138', $errno, $errs);
$allSockets = array($socket);
while (true) {
   $read = $allSockets;
   stream_select($read, $w=null, $e=null, null);
   foreach ($read as $sock) {
      if ($sock === $socket)
         $allSockets[] = stream_socket_accept($socket, -1);
      else if (feof($sock))
         closeSocket($sock, $allSockets);
      else
         process($sock);
   }
}
```

Exemple : myecho

Code PHP

```php
function closeSocket($sock, &$allSockets) {
    fclose($sock);
    $key = array_search($sock, $allSockets);
    unset($allSockets[$key]);
}

function process($socket) {
   $s = fgets($socket);
   fwrite($socket, $s);
}
```

Mono-processus

Avantages	Inconvénients
Ressources préservées	Délai exponentiel
Aucune dépendance	Blocage sur I/O lentes
Partage de données	Fragilité relative (consommation mémoire, plantage applicatif, ...)

Benchmark : connexion unique

Type	Connexion	Échange 1	Échange 2
C	5 µs	3 µs	2 µs
xinetd	5 µs	31 ms	2 µs
multi	5 µs	8 µs	2 µs
mono	5 µs	3 µs	3 µs

Benchmark : connexions multiples

Type	100 connexions	100 messages	1000 connexions	1000 messages	Maximum
C	0.04 s	0.05 s	6.5 s	0.2 s	> 1000
xinetd	0.4 s	1.8 s	-	-	< 120
multi	3 s	0.05 s	60.5 s	0.2 s	> 1000
mono	0.04 s	0.05 s	57.5 s	3.6 s	> 1000

ZeroMQ

Présentation

Bibliothèque de fonctions réseau

~~Gestion de files de messages~~

Protocole spécifique

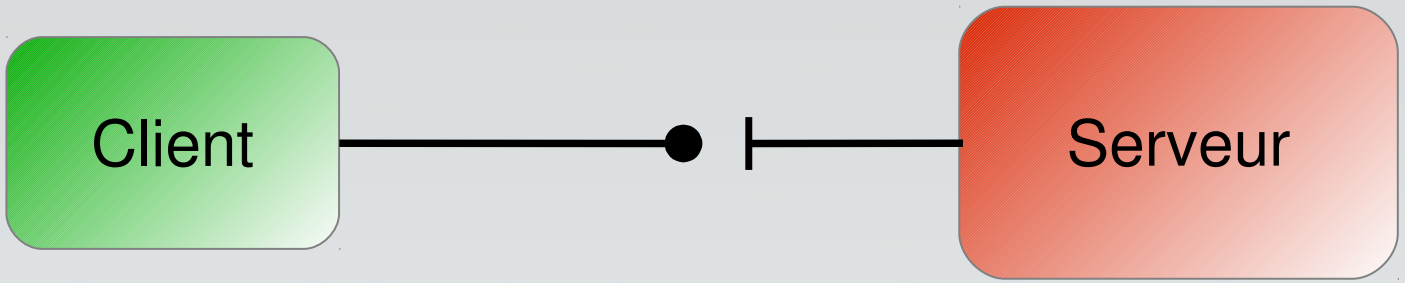
37 langages supportés

Redéfinition des concepts réseau

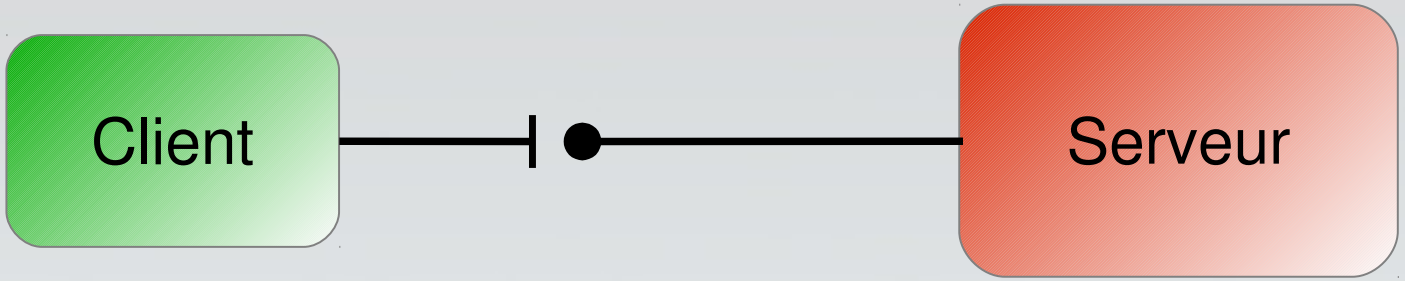
Méthodes de transport

Inter-threads	inproc
Inter-processus	ipc
Inter-machines	tcp

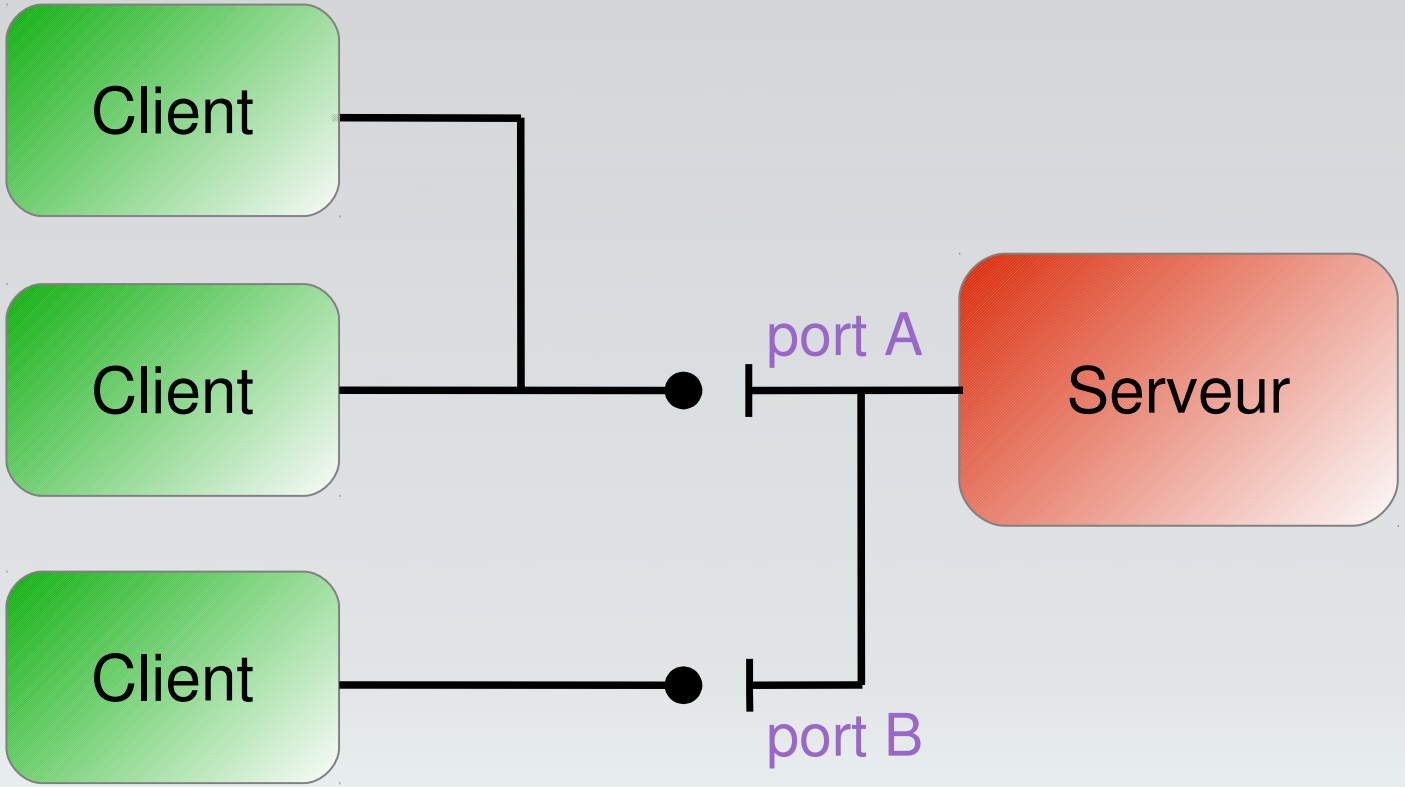
Découplage du sens de connexion



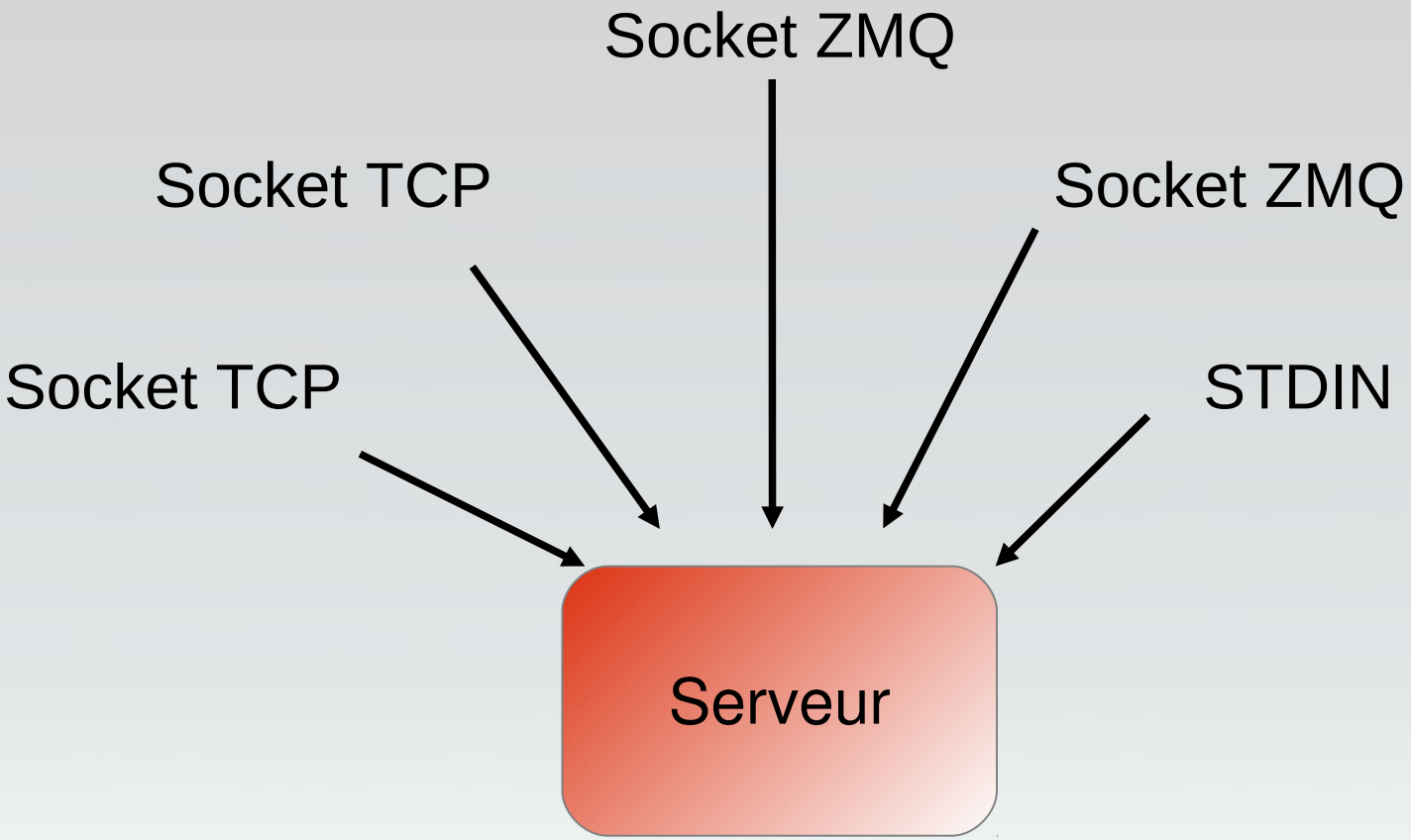
Découplage du sens de connexion



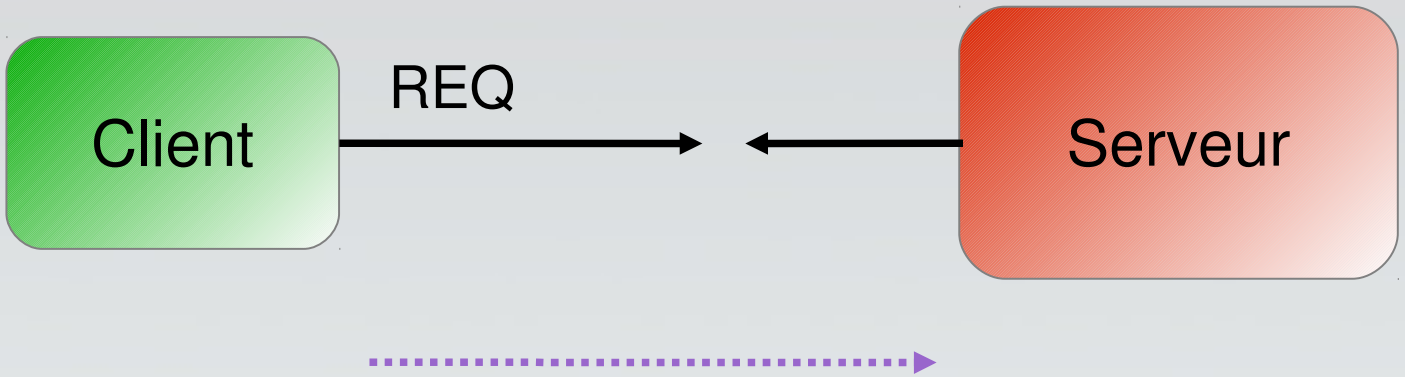
Socket multi-connexion



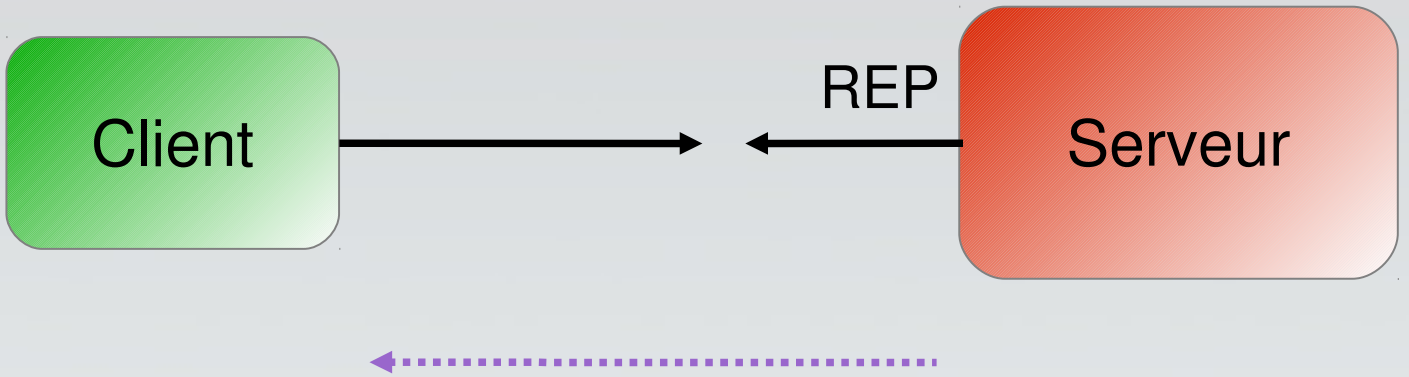
Polling



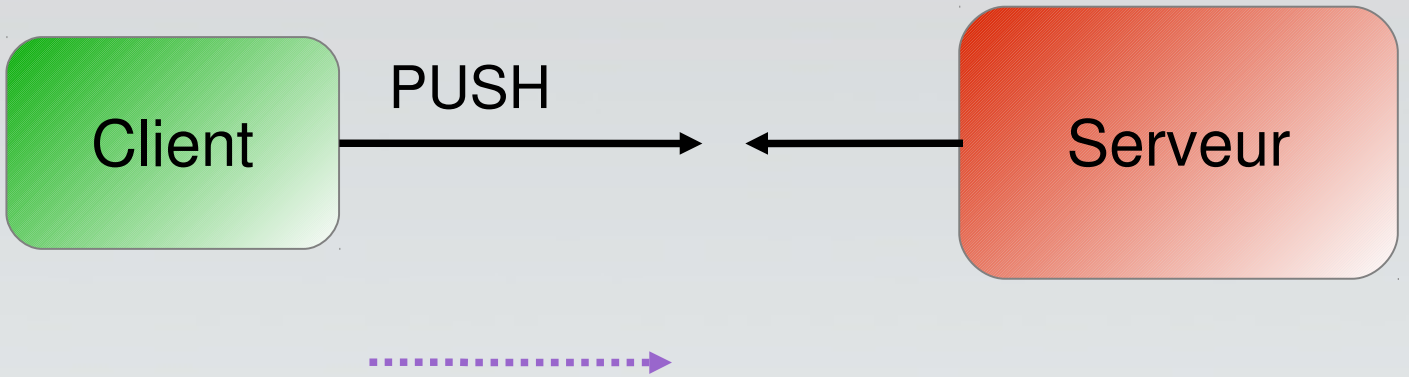
Connexion REQ/REP



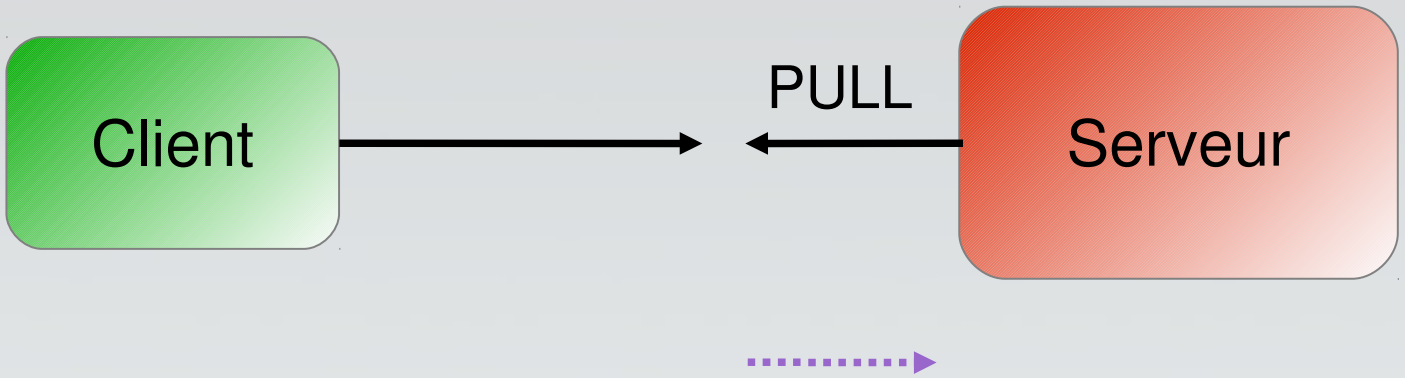
Connexion REQ/REP



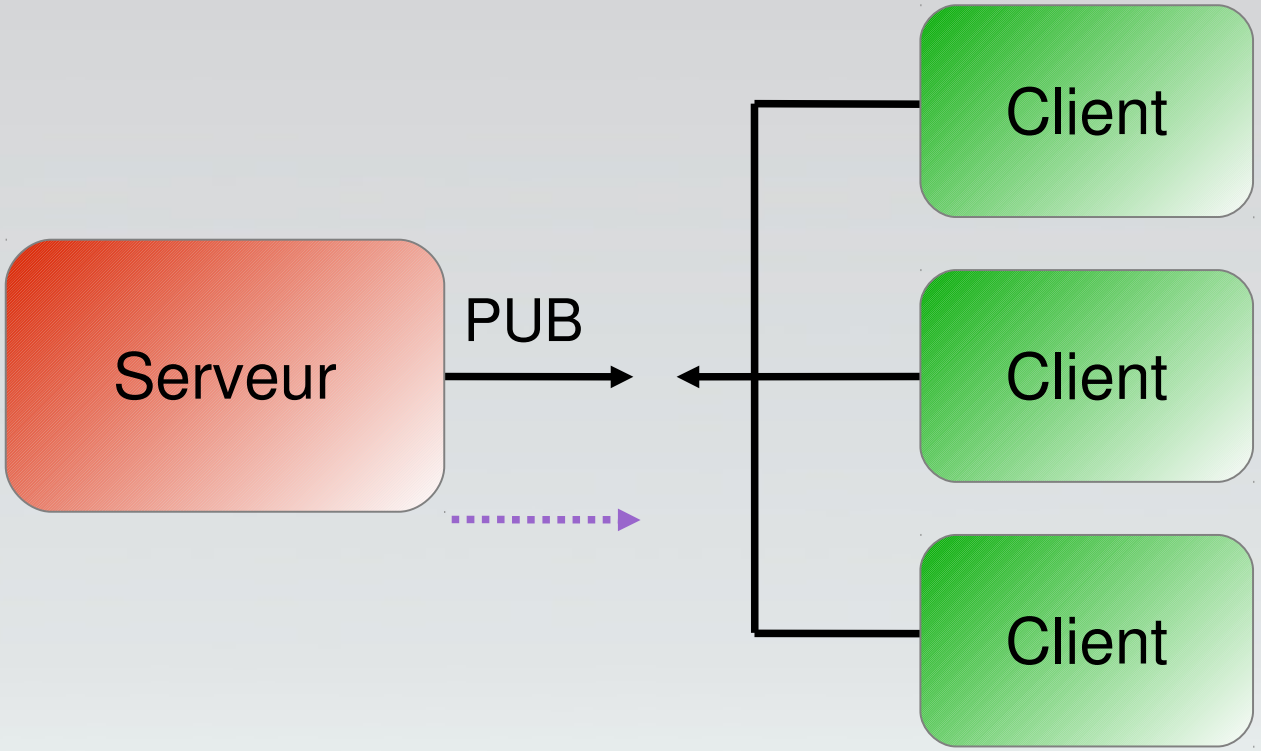
Connexion PUSH/PULL



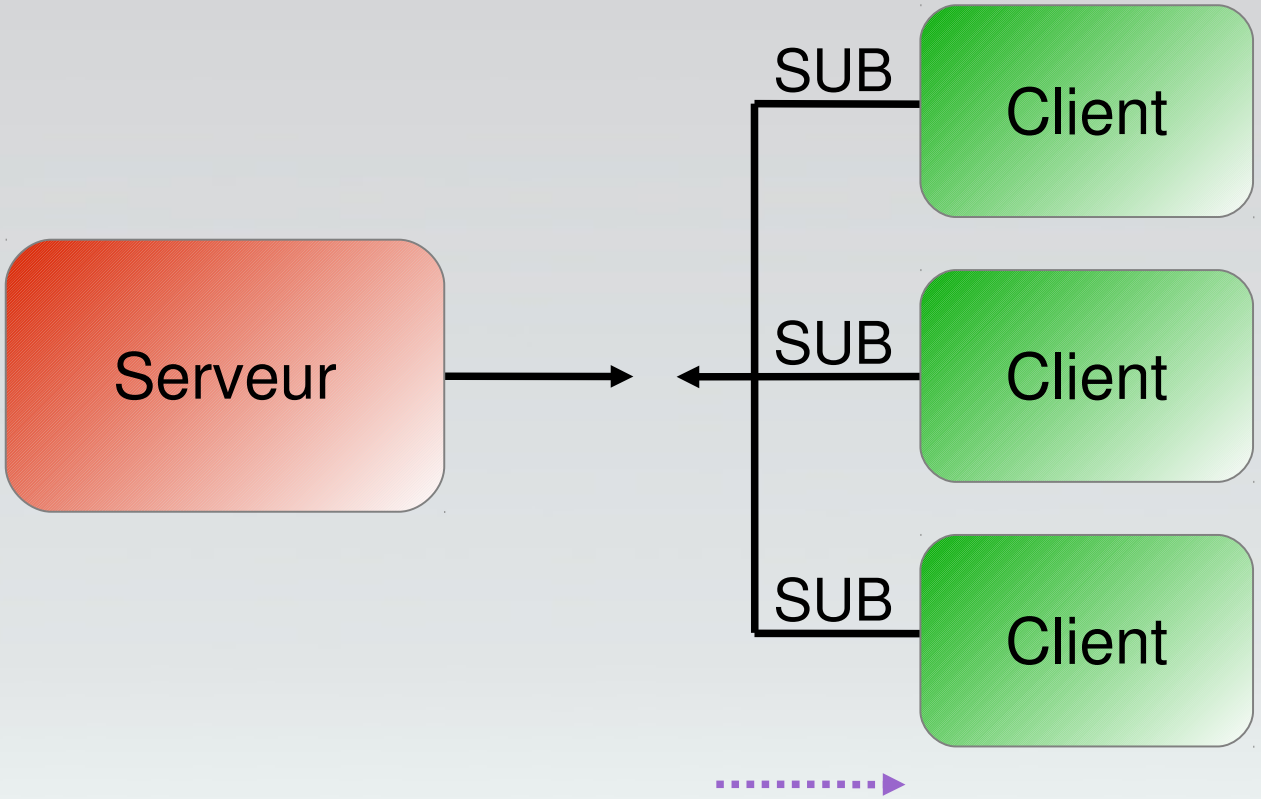
Connexion PUSH/PULL



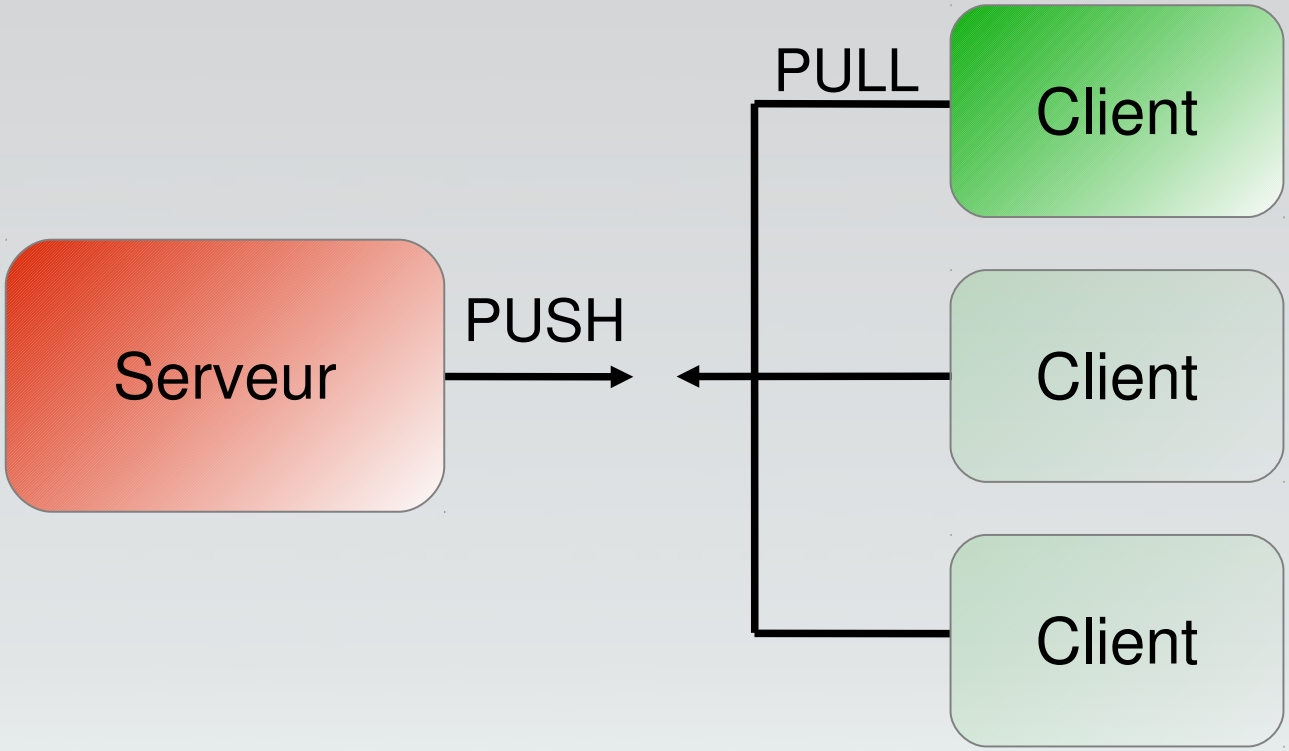
Connexion PUB/SUB



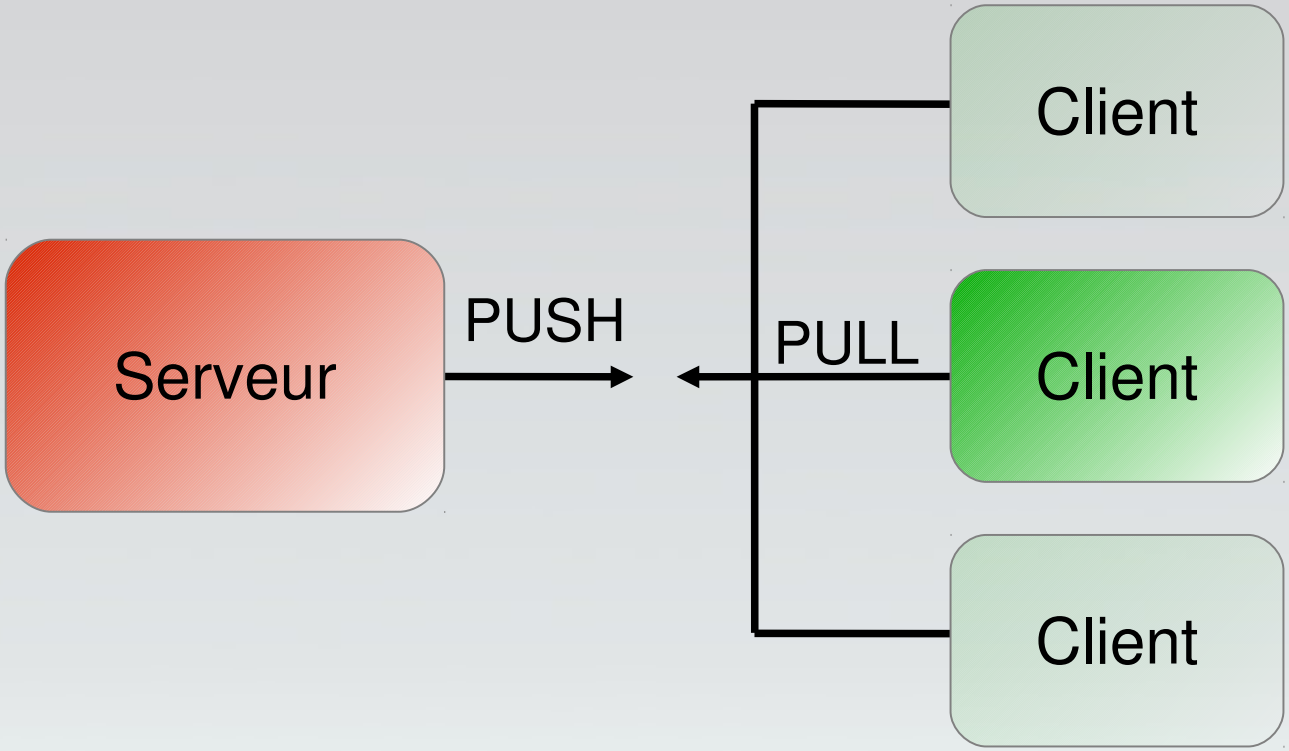
Connexion PUB/SUB



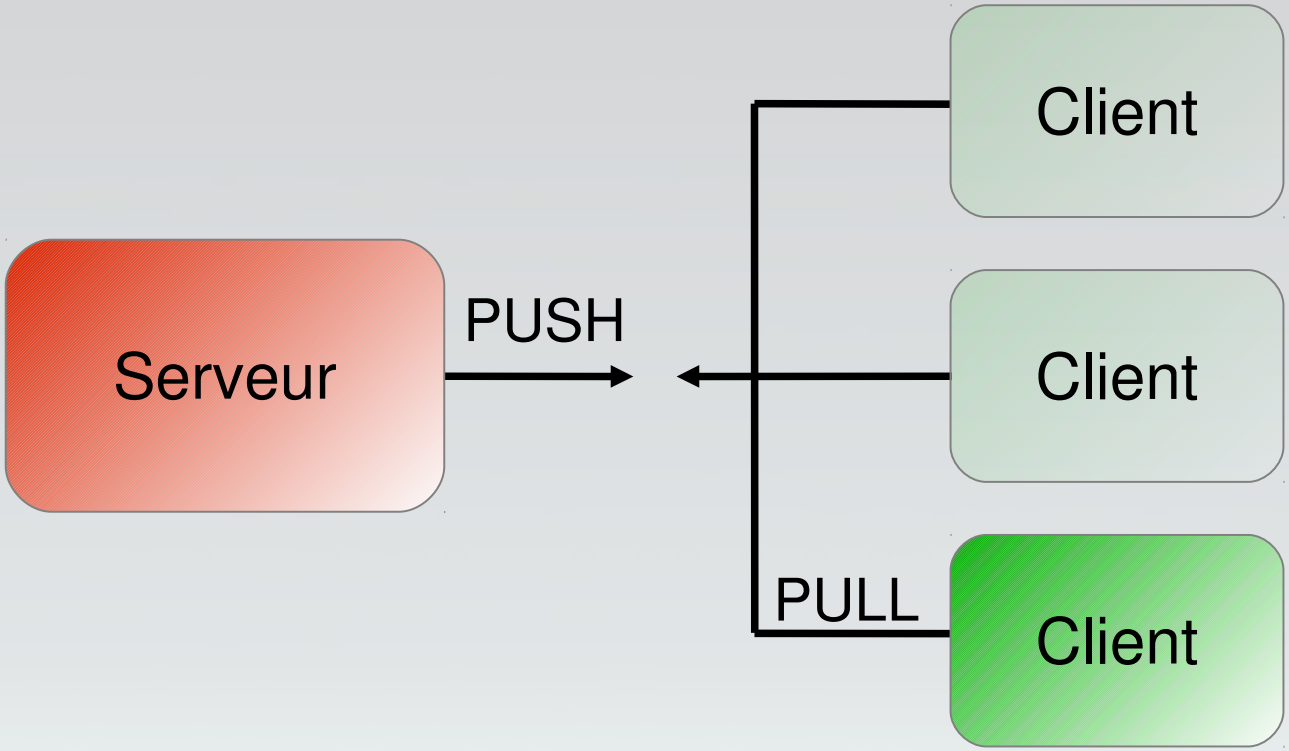
Load-balancing



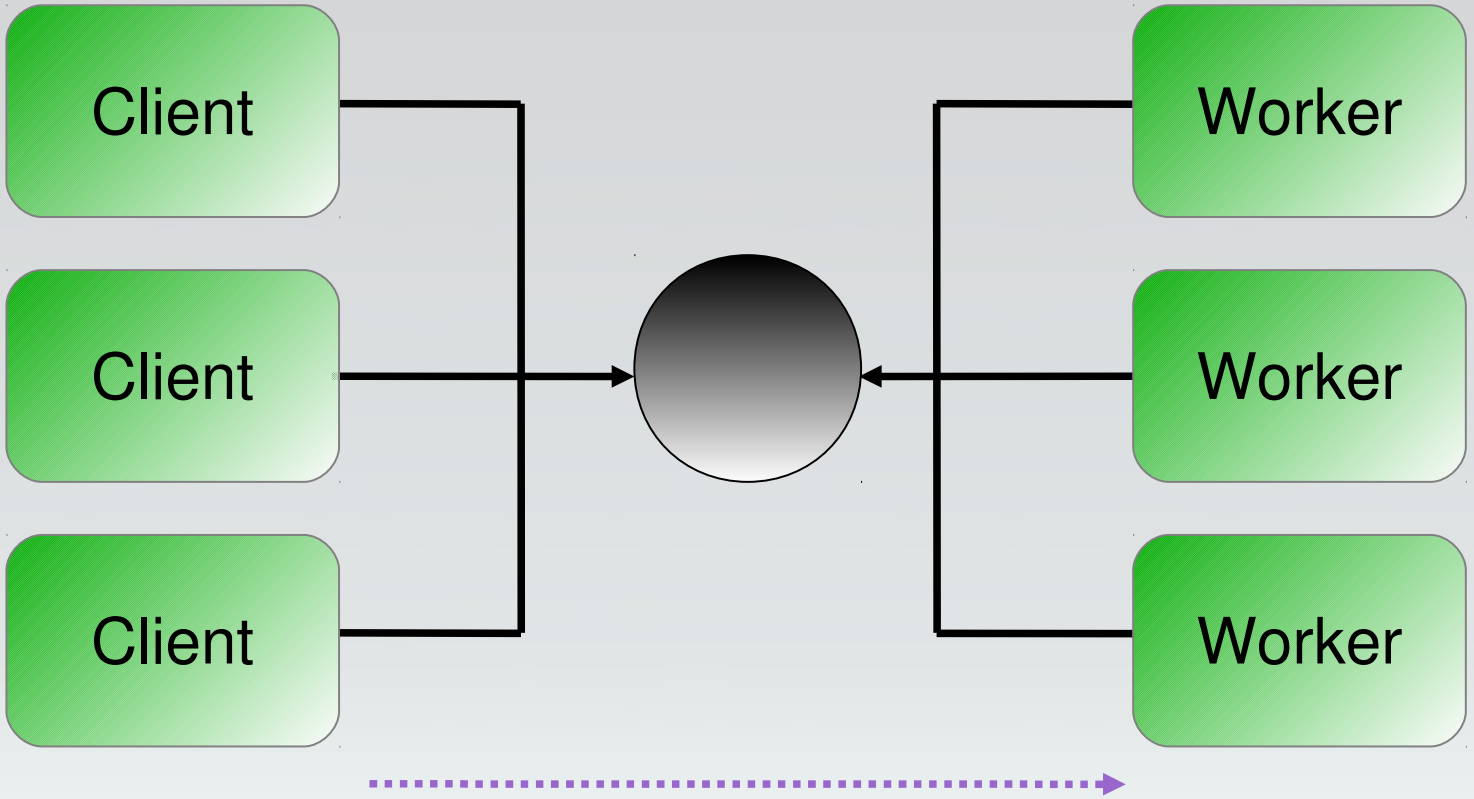
Load-balancing



Load-balancing



Pipeline



ZeroMQ : Installation

Linux Ubuntu 11.10

```
# apt-get install libzmq-dev

# apt-get install php-pear

# pear channel-discover pear.zero.mq

# pecl install zero.mq/zmq-beta

# echo "extension=zmq.so" >
    /etc/php5/conf.d/zmq.ini
```

Exemple : myecho

Code PHP : serveur ZMQ

```
#!/usr/bin/php
<?php

$ctx = new ZMQContext();
$server = new ZMQSocket($ctx, ZMQ::SOCKET_REP);
$server->bind('tcp://*:11141');

while (true) {
    $message = $server->recv();
    $server->send($message);
}
```

Exemple : myecho

Code PHP : client ZMQ

```
#!/usr/bin/php
<?php

$ctx = new ZMQContext();
$client = new ZMQSocket($ctx, ZMQ::SOCKET_REQ);
$client->connect('tcp://localhost:11141');

$client->send('bonjour');
$response = $client->recv();
print("Réponse : '$response'\n");
```

Benchmark : connexion unique

Type	Connexion	Échange 1	Échange 2
C	5 µs	3 µs	2 µs
xinetd	5 µs	31 ms	2 µs
multi	5 µs	8 µs	2 µs
mono	5 µs	3 µs	3 µs
zmq	2 ms	2 ms	4 µs

Benchmark : connexions multiples

Type	100 connexions	100 messages	1000 connexions	1000 messages	Maximum
C	0.04 s	0.05 s	6.5 s	0.2 s	> 1000
xinetd	0.4 s	1.8 s	-	-	< 120
multi	3 s	0.05 s	60.5 s	0.2 s	> 1000
mono	0.04 s	0.05 s	57.5 s	3.6 s	> 1000
zmq	0.05 s	0.12 s	0.6 s	0.8 s	> 1000

Conclusion

Démons en PHP 

Faciles et rapides à coder

Vraiment performant

(x)inetd, mono/multi-processus, ZeroMQ, libevent, eio, ...

geek-directeur-technique.com/zeromq

amaury@amaury.net